Московский государственный университет имени М.В.Ломоносова
Факультет Вычислительной математики и кибернетики



СУПЕРКОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ И ТЕХНОЛОГИИ

ВМК МГУ, 2020

Московский государственный университет имени М.В.Ломоносова
Факультет Вычислительной математики и кибернетики



СУПЕРКОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ И ТЕХНОЛОГИИ

СТРУКТУРА АЛГОРИТМОВ

Вл.В.Воеводин

Зав.кафедрой Суперкомпьютеров и квантовой информатики ВМК МГУ,
Директор НИВЦ МГУ,
чл.-корр. РАН, д.ф.-м.н., профессор

voevodin@parallel.ru

ВМК МГУ, 2020

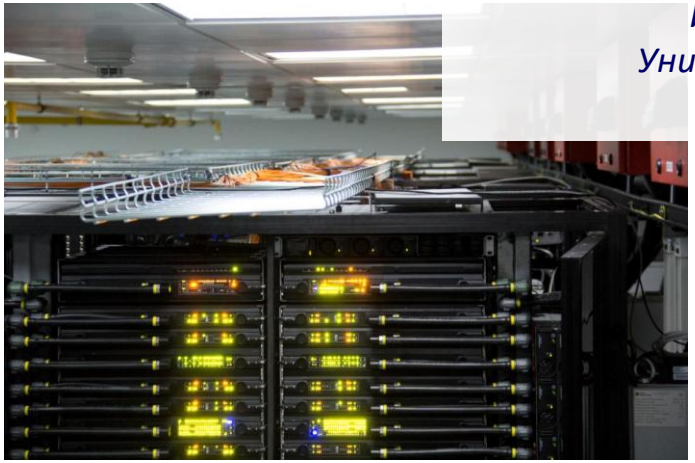
Суперкомпьютерный комплекс МГУ





Суперкомпьютерный центр МГУ сегодня:
Пользователи: 2955
Проекты: 880

Факультеты / Институты МГУ: 21
Институты РАН: 95
Университеты России: 102







1 стойка = 256 узлов: Intel Xeon (14c) + NVIDIA K40/P100 = 515 Tflop/s

Суперкомпьютер "Ломоносов-2" (7 стоек) = 4.9 Pflop/s

Вычислительные платформы МГУ





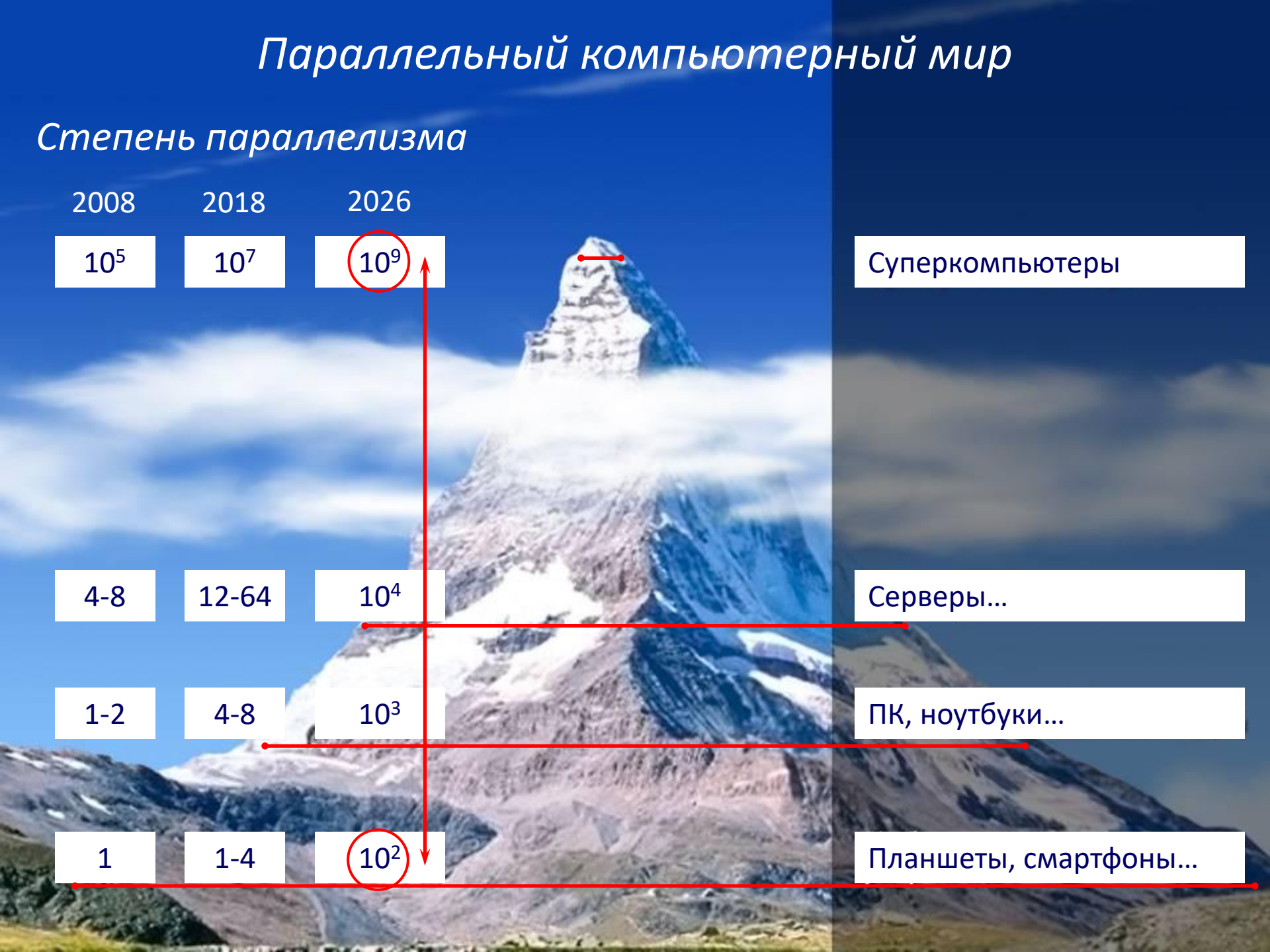


Технологическая основа:
Intel Xeon 4/6/10/12… ядер
SMP узел на 128 ядер
Intel Xeon Phi (KNL)
NVIDIA 2070 / 2090 / K40 / P100
IBM Power 8 / IBM Blue Gene/P
Памяти на узел: от 12GB до 2TB

Разнообразие велико…

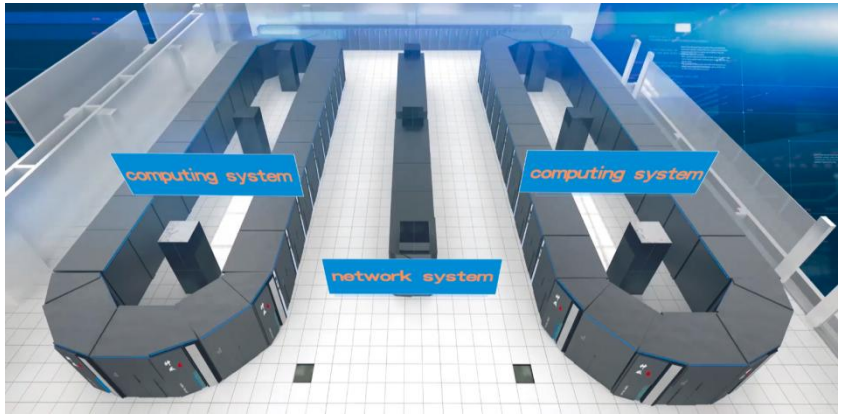
Параллельный компьютерный мир

Степень параллелизма



Суперкомпьютер Sunway TaihuLight, Китай
(#1 Top500 в 2016-2017 г.)







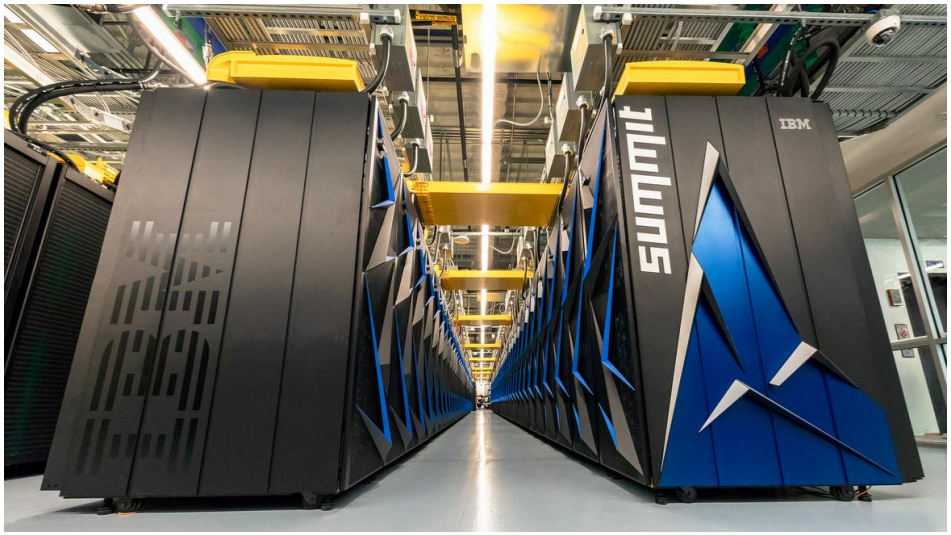
40 960 вычислительных узлов
40 960 CPUs (SW26010, 260 ядер)

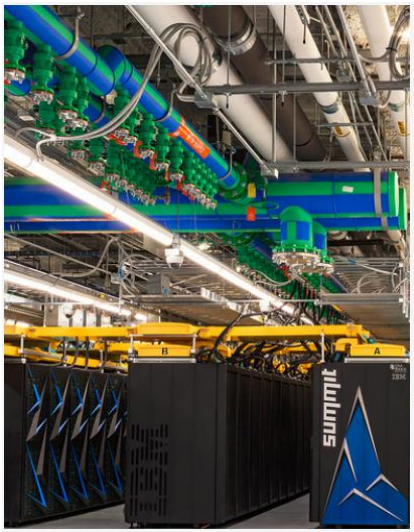
Всего: 10 649 600 процессорных ядер

Производительность:
Пик (теория): 125.4 Pflop/s
Тест Linpack: 93 Pflop/s (74%)

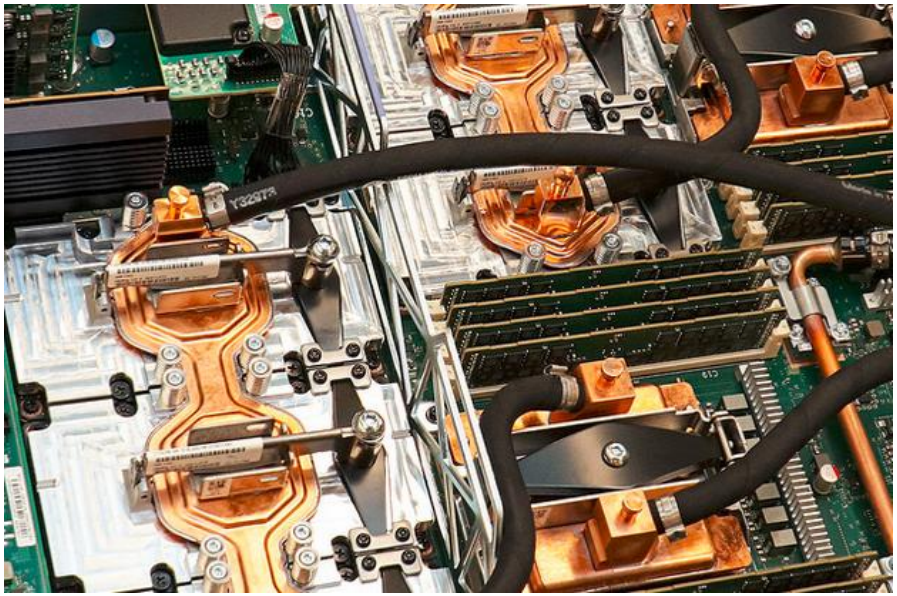
Оперативная память = 1.31 Пбайт
HDD = 20 ПБайт

Суперкомпьютер IBM Summit, США
(#1 Top500 в 2018-2019 г.)









4 608 вычислительных узлов,
в каждом узле:
2 x CPUs (IBM Power9, 22 ядра)
6 x GPU (NVIDIA Tesla V100)

Производительность:
Пик (теория): 200.8 Pflop/s
Тест Linpack: 148.6 Pflop/s (74%)

Оперативная память = 10 Пбайт
HDD = 250 ПБайт

Суперкомпьютер Fujitsu Fugaku, Япония
(#1 Top500 в 2020 г.)





158 976 выч.узлов, 432 стойки

CPU: A64FX ARM v8.2-A, 48/52 cores
Векторные инструкции (SVE),
аппаратная поддержка барьеров…
Int: 1,2,4,8 байт; Float: 16,32,64 бит

Производительность:
Пик (теория): 513 Pflop/s
Тест Linpack: 415 Pflop/s (81%)
Оперативная память = 4.85 Пбайт
Энергопотребление = 28 МВт
Площадь = 1920 м^2

Простые, но фундаментальные вопросы параллельных вычислений

Как решить задачу быстрее?





Параллелизм, конвейерность…

Как решить задачу намного быстрее?

Как организовать работу нескольких человек/ядер/процессоров…?

эффективную

миллионов




Теория / Практика…





Компьютеры с различной архитектурой…



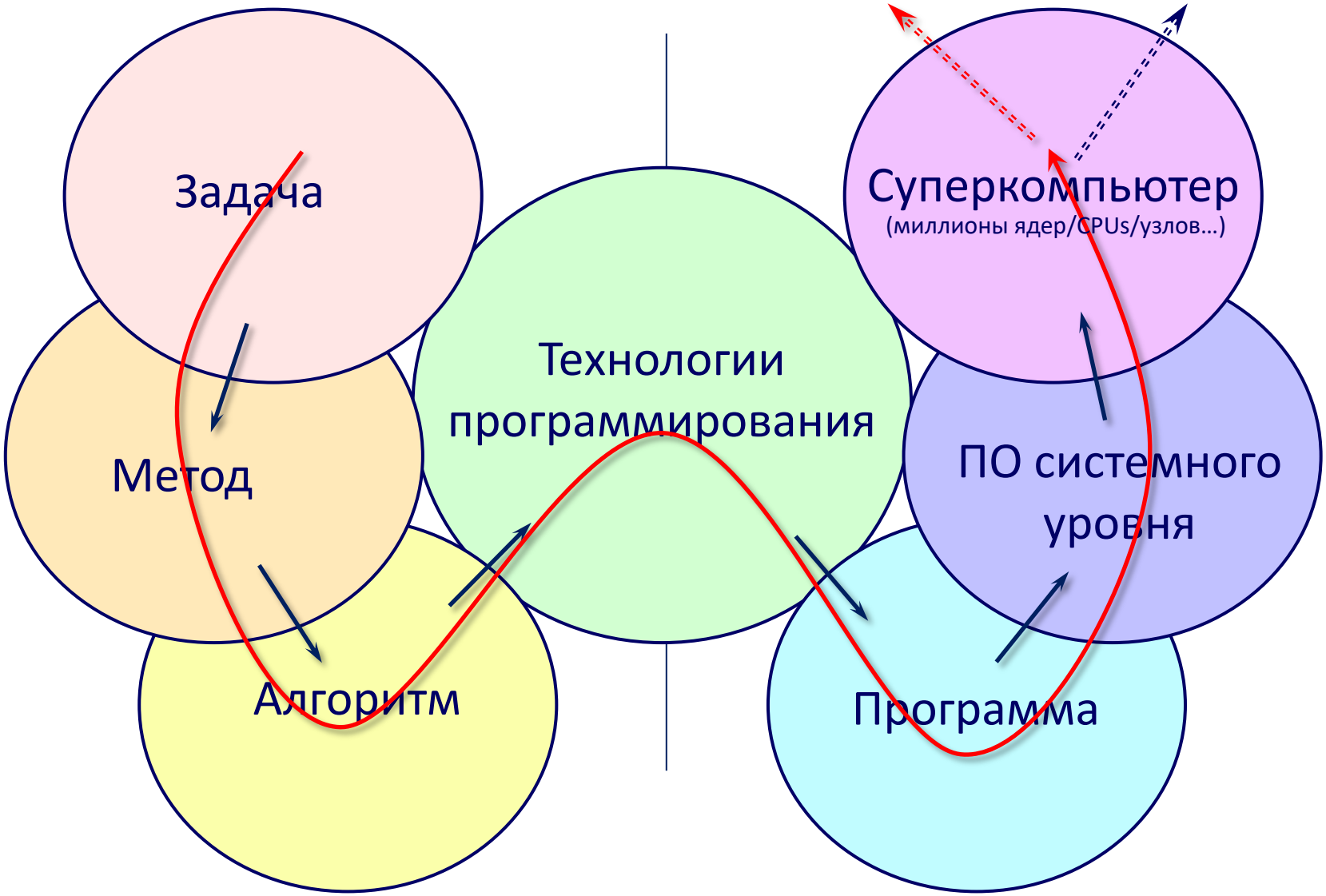
Синхронность / Асинх-ть / Модели…

Решение задач и вычислительные системы

(Реальность)
Реальная
производительность

Огромный
разрыв !

(Ожидания)
Пиковая
производительность



Алгоритмическая сторона

Компьютерная сторона

Тонкий анализ суперкомпьютерных приложений: динамика исполнения

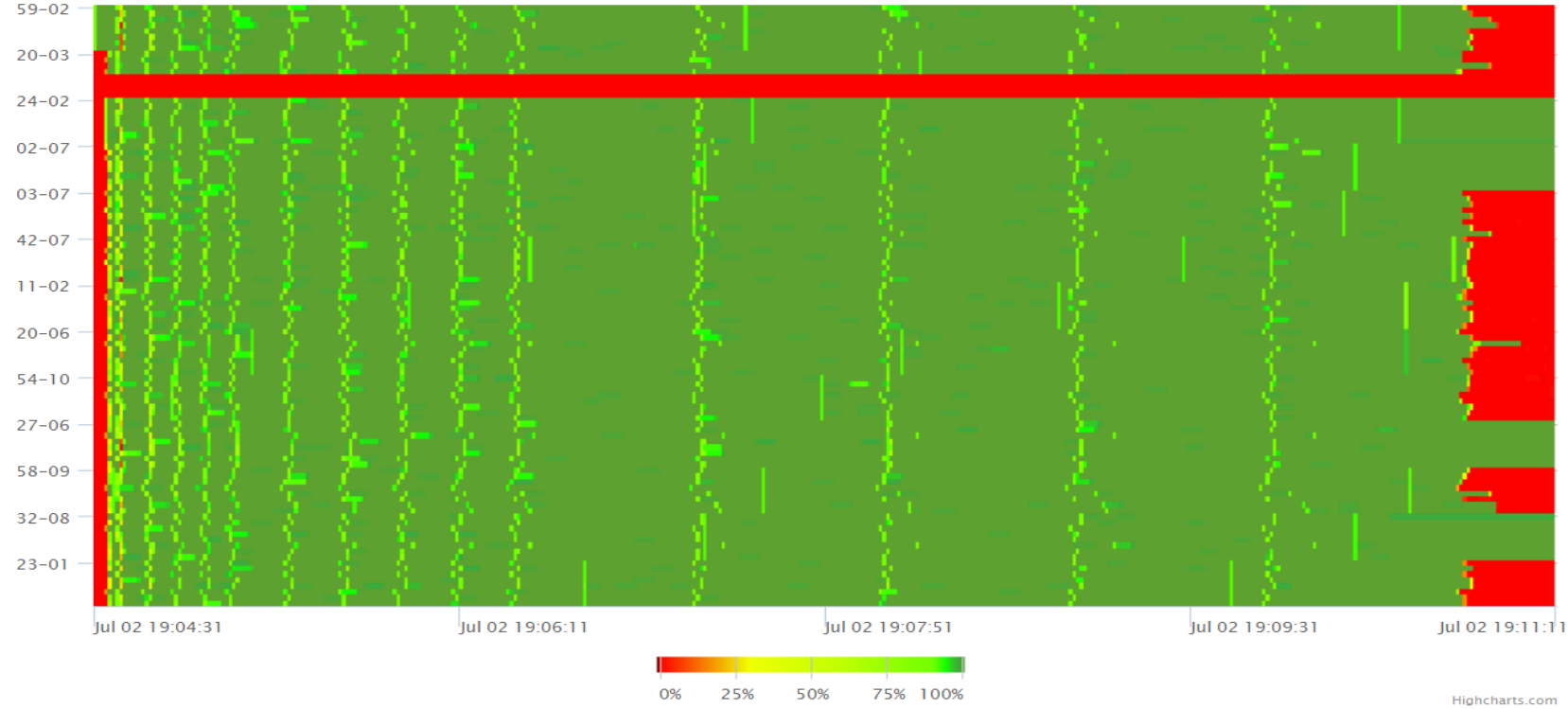
Информация о задаче № regular-1404302831-496016 пользователя

expand all

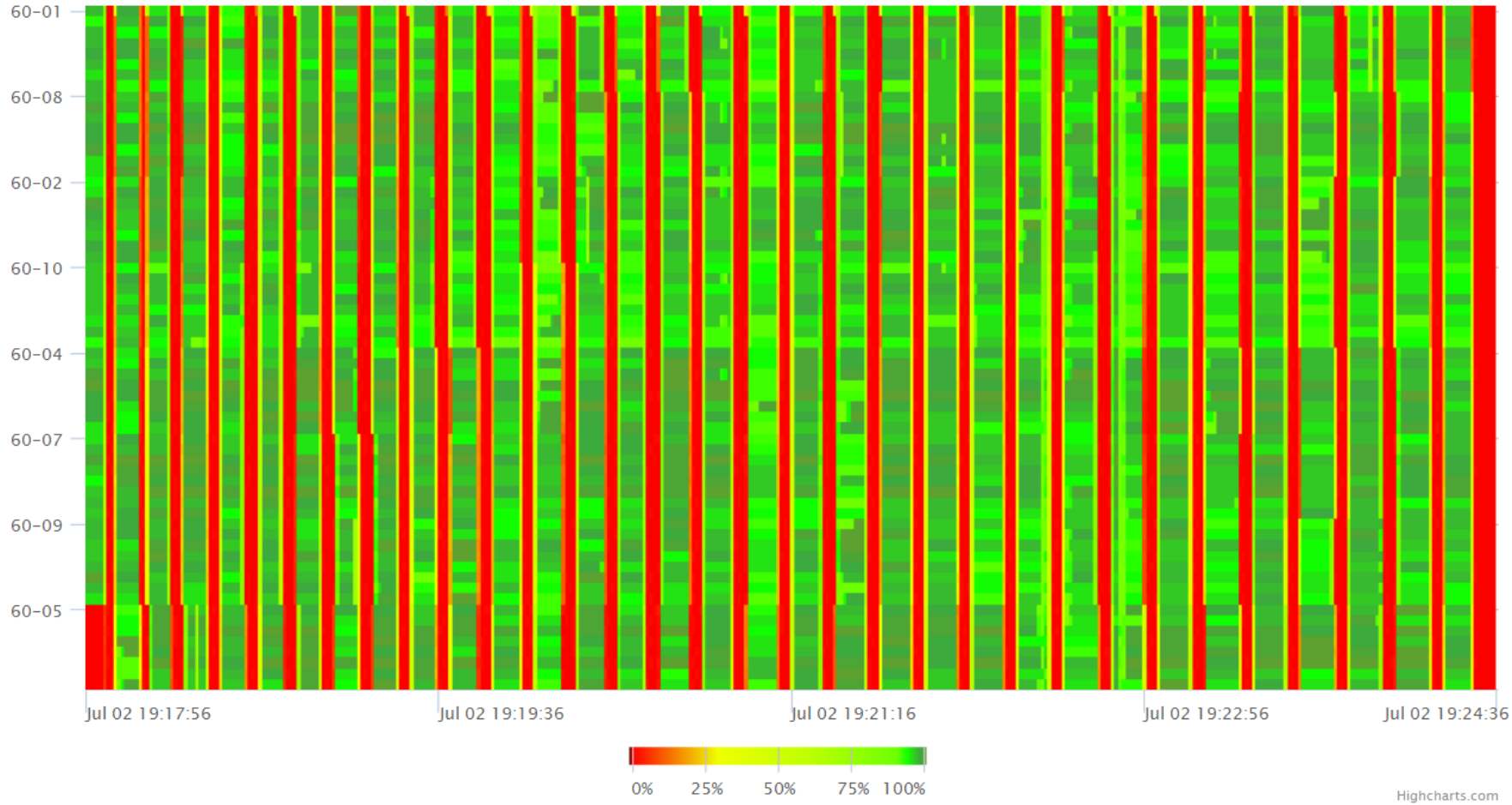
Информация о запуске

Строка запуска:	./2dxy_m01_p08.ex
Число ядер:	100
Номера узлов:	node-02-07,node-03-07,node-11-02,node-20-03,node-20-06,node-23-01,node-24-02,node-27-06,node-32-08,node-42-07,node-54-10,node-58-09,node-59-02
Дата постановки в очередь:	Wed, 02 Jul 2014 16:07:11 (1404302831)
Дата запуска:	Wed, 02 Jul 2014 19:04:29 (1404313469)
Дата окончания счета:	Wed, 02 Jul 2014 19:46:13 (1404315973)
Время счета:	0 days 0 hours 41 minutes 44 seconds
Время ожидания:	0 days 2 hours 57 minutes 18 seconds

Количе
процес



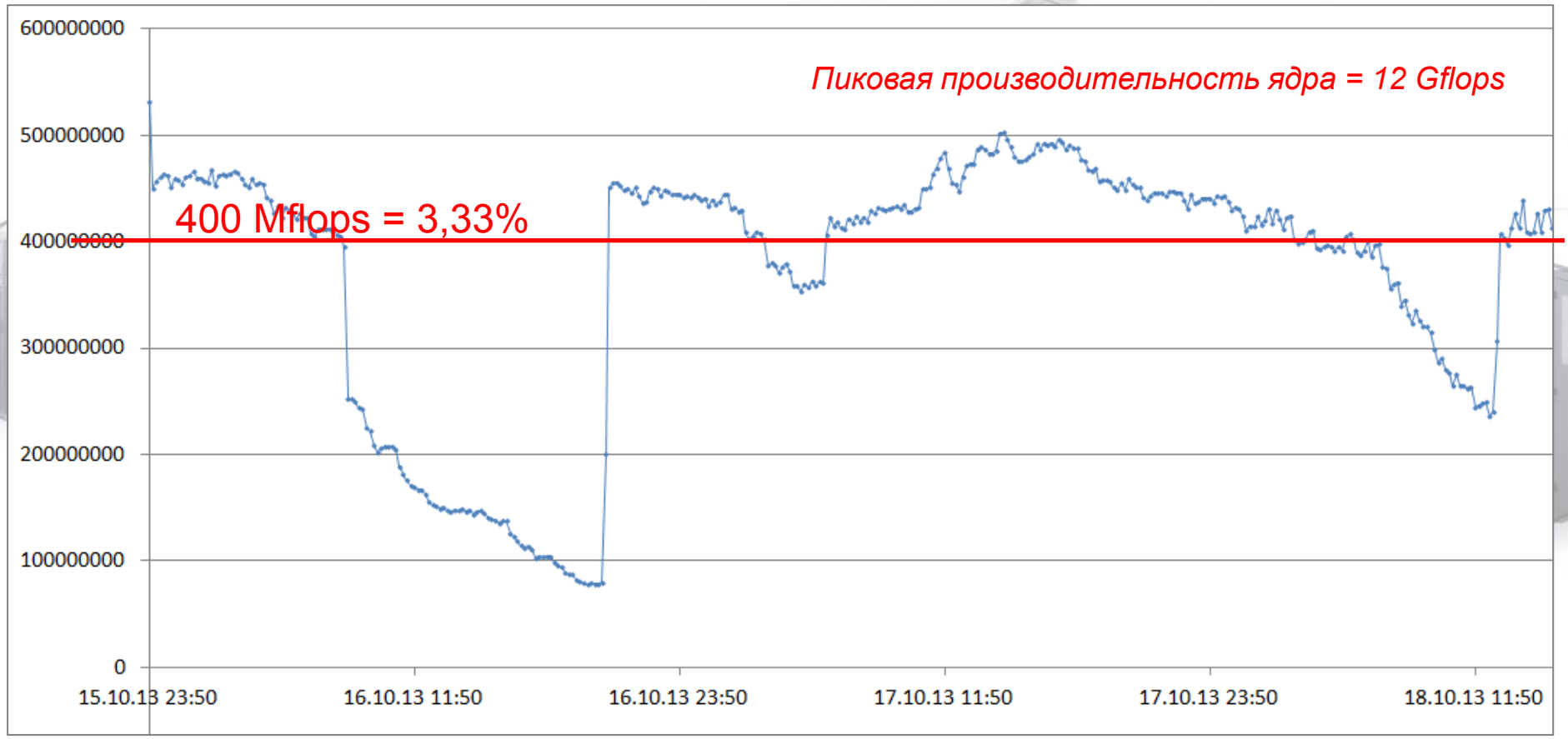
Тонкий анализ суперкомпьютерных приложений: динамика исполнения



Тонкий анализ суперкомпьютерных приложений: динамика исполнения



Эффективность суперкомпьютерных центров
(простая оценка)



Усредненная производительность одного ядра суперкомпьютера “Чебышев” за 3 дня

Хорошо ли мы знаем свойства и особенности параллельных алгоритмов ?

Должны ли мы думать о параллельных алгоритмах?

Да… К сожалению, да…

Хорошо ли мы знаем статические и динамические свойства параллельных программ?

Должны ли мы думать о свойствах параллельных программ?

Да… К сожалению, да…

Хорошо ли мы знаем архитектуры параллельных компьютеров ?

Должны ли мы думать об архитектурах?

Да… К сожалению, да…

Поколения архитектур и парадигмы программирования
(или как часто мы были вынуждены полностью переписывать приложения?)





Суперкомпьютер Cray-1

Векторно-конвейерные компьютеры

Середина 70-х годов.

Особенности архитектуры: векторные функциональные устройства, зацепление функциональных устройств, векторные команды в системе команд, векторные регистры.

Программирование: векторизация самых внутренних циклов.

Поколения архитектур и парадигмы программирования
(или как часто мы были вынуждены полностью переписывать приложения?)



Суперкомпьютер Cray X-MP



Суперкомпьютер Cray Y-MP

Векторно-параллельные компьютеры

Начало 80-х годов.

Особенности архитектуры: векторные функциональные устройства, зацепление функциональных устройств, векторные команды в системе команд, векторные регистры.
Небольшое число процессоров объединяются над общей памятью.

Программирование: векторизация самых внутренних циклов и распараллеливание на внешнем уровне, единое адресное пространство, локальные и глобальные переменные.

Поколения архитектур и парадигмы программирования
(или как часто мы были вынуждены полностью переписывать приложения?)



Суперкомпьютер Cray T3D



Суперкомпьютер Intel Paragon XPS140

Массивно-параллельные компьютеры

Начало 90-х годов.

Особенности архитектуры: тысячи процессоров объединяются с помощью коммуникационной сети по некоторой топологии, распределенная память.

Программирование: обмен сообщениями, отсутствие единого адресного пространства, PVM, Message Passing Interface. Необходимость выделения массового параллелизма, явного распределения данных и согласования параллелизма с распределением.

Поколения архитектур и парадигмы программирования
(или как часто мы были вынуждены полностью переписывать приложения?)



DEC AlphaServer



Суперкомпьютер Sun StarFire

Параллельные компьютеры с общей памятью

Середина 90-х годов.

Особенности архитектуры: сотни процессоров объединяются над общей памятью.

Программирование: единое адресное пространство, локальные и глобальные переменные, Linda, OpenMP.

Поколения архитектур и парадигмы программирования
(или как часто мы были вынуждены полностью переписывать приложения?)



Суперкомпьютер МГУ “Чебышев”



“K” суперкомпьютер

Кластеры из узлов с общей памятью

Начало 2000-х.

Особенности архитектуры: большое число многопроцессорных узлов объединяются вместе с помощью коммуникационной сети по некоторой топологии, распределенная память; в рамках каждого узла несколько (многоядерных) процессоров объединяются над общей памятью.

Программирование: неоднородная схема MPI+OpenMP;
необходимость выделения массового параллелизма, явное распределение данных, обмен сообщениями на внешнем уровне; распараллеливание в едином адресном пространстве, локальные и глобальные переменные на уровне узла с общей памятью.

Поколения архитектур и парадигмы программирования
(или как часто мы были вынуждены полностью переписывать приложения?)



Суперкомпьютер МГУ “Ломоносов”



Суперкомпьютер Tianhe-2

Кластеры из узлов с общей памятью с ускорителями

Середина 2000-х.

Особенности архитектуры: большое число многопроцессорных узлов объединяются вместе с помощью коммуникационной сети по некоторой топологии, распределенная память; в рамках каждого узла несколько (многоядерных) процессоров объединяются над общей памятью; на каждом узле несколько ускорителей (GPU, Phi).

Программирование:
MPI+OpenMP+OpenCL/CUDA;

Поколения архитектур и парадигмы программирования
(или как часто мы были вынуждены полностью переписывать приложения?)

С 1976 года до наших дней:

70-е – Векторизация циклов
80-е – Распараллеливание циклов (внешних) + Векторизация (внутренних)
90-е - MPI
середина 90-х - OpenMP
середина 2000-х - MPI+OpenMP
2010-е - CUDA, OpenCL, MPI+OpenMP+ускорители (GPU, Xeon Phi)

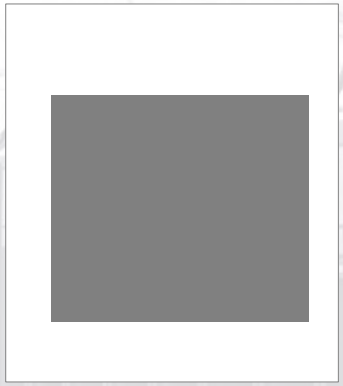
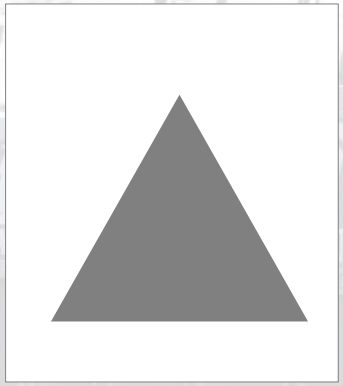
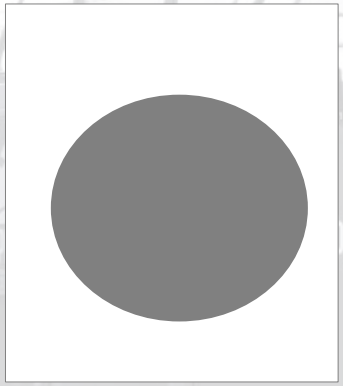
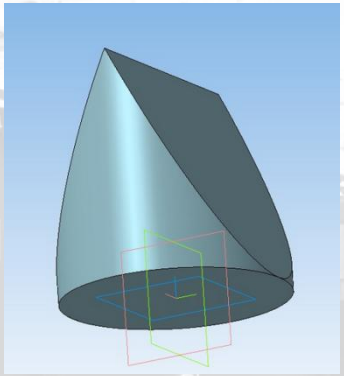
…

Изменения в архитектуре компьютеров не меняют алгоритмов! Но:

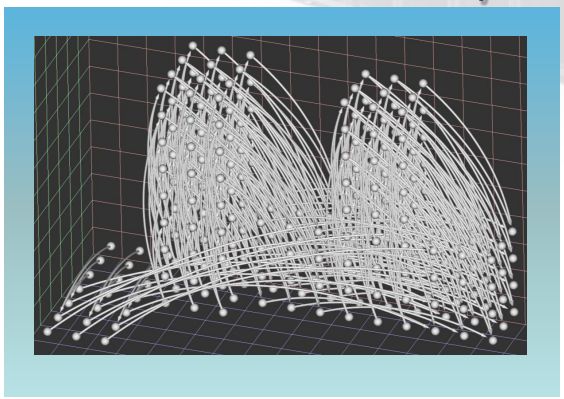
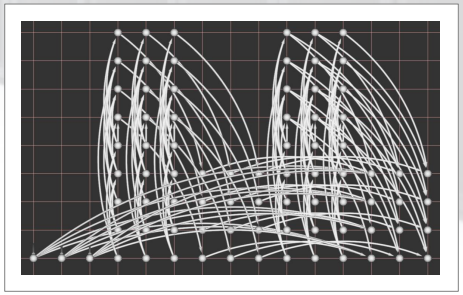
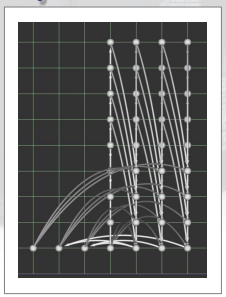
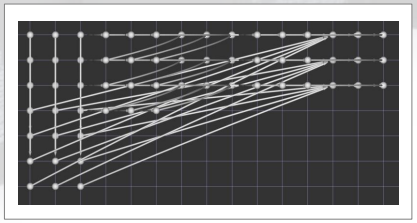
Для каждого поколения компьютеров мы вынуждены:
- Анализировать алгоритмы, чтобы понять, как их приспособить под новую компьютерную платформу ;
- Описывать найденные свойства, чтобы получить эффективную реализацию для новой платформы.

Изменения в архитектуре компьютеров не меняют алгоритмов!

Эти фигуры разные?

Каковы способы представления этого алгоритма?

…

Поколения архитектур и парадигмы программирования
(или как часто мы были вынуждены полностью переписывать приложения?)

С 1976 года до наших дней:

70-е – Векторизация циклов
80-е – Распараллеливание циклов (внешних) + Векторизация (внутренних)
90-е - MPI
середина 90-х - OpenMP
середина 2000-х - MPI+OpenMP
2010-е - CUDA, OpenCL, MPI+OpenMP+ускорители (GPU, Xeon Phi)

…

Можно ли выполнить такой анализ “раз и навсегда” ?

Изменения в архитектуре компьютеров не меняют алгоритмов! Но:

Для каждого поколения компьютеров мы вынуждены:
- Анализировать алгоритмы, чтобы понять, как их приспособить под новую компьютерную платформу ;
- Описывать найденные свойства, чтобы получить эффективную реализацию для новой платформы.

Что значит “выполнить анализ алгоритма”?

Что мы должны найти в алгоритмах?

“…выполнить анализ раз и навсегда…“ – как записать результаты?

Что представляет “единое” / “универсальное” описание алгоритмов?

Какие свойства алгоритмов нужно исследовать и описать чтобы получать эффективные реализации в будущем для будущих платформ?

Слишком много “простых” вопросов…



Хорошо ли мы знаем свойства, особенности, статические и динамические характеристики параллельных алгоритмов?

Какие свойства алгоритмов важны?

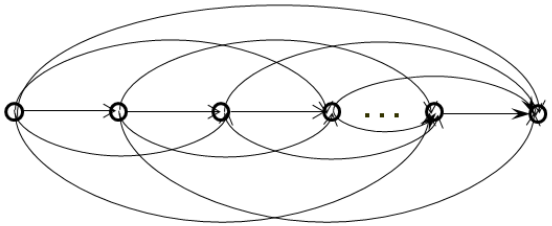
На какие свойства алгоритмов нужно обращать внимание?

И простые свойства могут быть важны…

(объем входных/выходных данных)

Нахождение транзитивного замыкания графа:





На входе: n вершин, n-1 дуга.

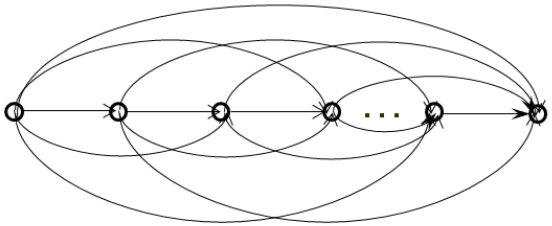
На выходе: n вершин, n(n-1)/2 дуга.

Социальные сети: 10^8 вершин, 10^{11} дуг. ⇨ ☹

И простые свойства могут быть важны…
(объём входных/выходных данных)

Нахождение транзитивного замыкания графа:





На входе: n вершин, n-1 дуга.

На выходе: n вершин, n(n-1)/2 дуга.

Социальные сети: 10^8 вершин, 10^{11} дуг. ⇨ ☹

Вычислительная мощность алгоритма = $\frac{\text{Число операций}}{\text{Объём данных}}$

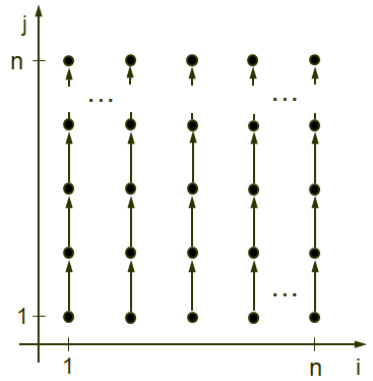
Тест Linpack (решение линейной системы): $\approx n$

Поэлементное сложение двух векторов: 1/3

Параллелизм бывает неудобным

(Что нужно знать про алгоритмы)

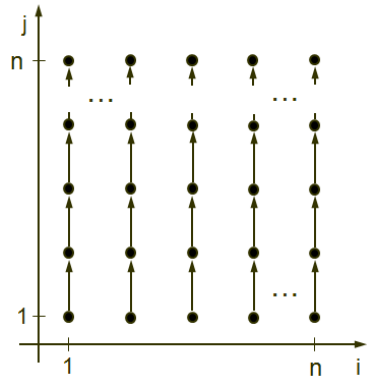
```
#pragma OpenMP parallel for
for( i = 1 ; i <= n ; ++i)
    for( j = 1 ; j <= m ; ++j)
        A[i][j] = (A[i][j] * A[i][j-1]) / 2 ;
```



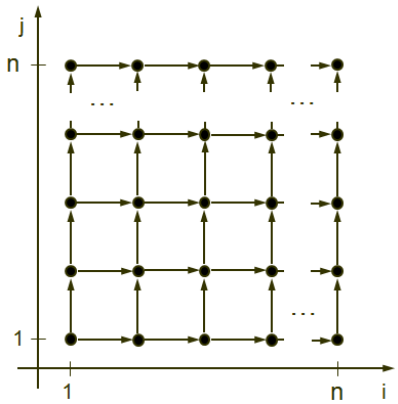
Параллелизм бывает неудобным

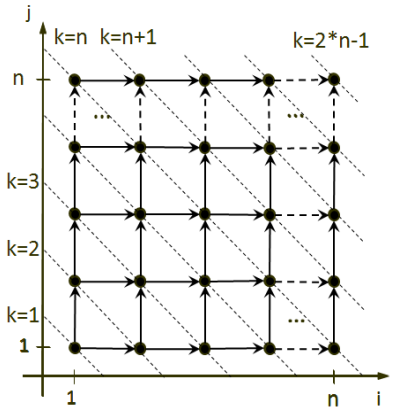
(Что нужно знать про алгоритмы)

```
#pragma OpenMP parallel for
for( i = 1 ; i <= n ; ++i)
    for( j = 1 ; j <= m ; ++j)
        A[i][j] = (A[i][j] * A[i][j-1]) / 2 ;
```




```
for( i = 1 ; i <= n ; ++i)
    for( j = 1 ; j <= m ; ++j)
        A[i][j] = (A[i-1][j] * A[i][j-1]) / 2 ;
```

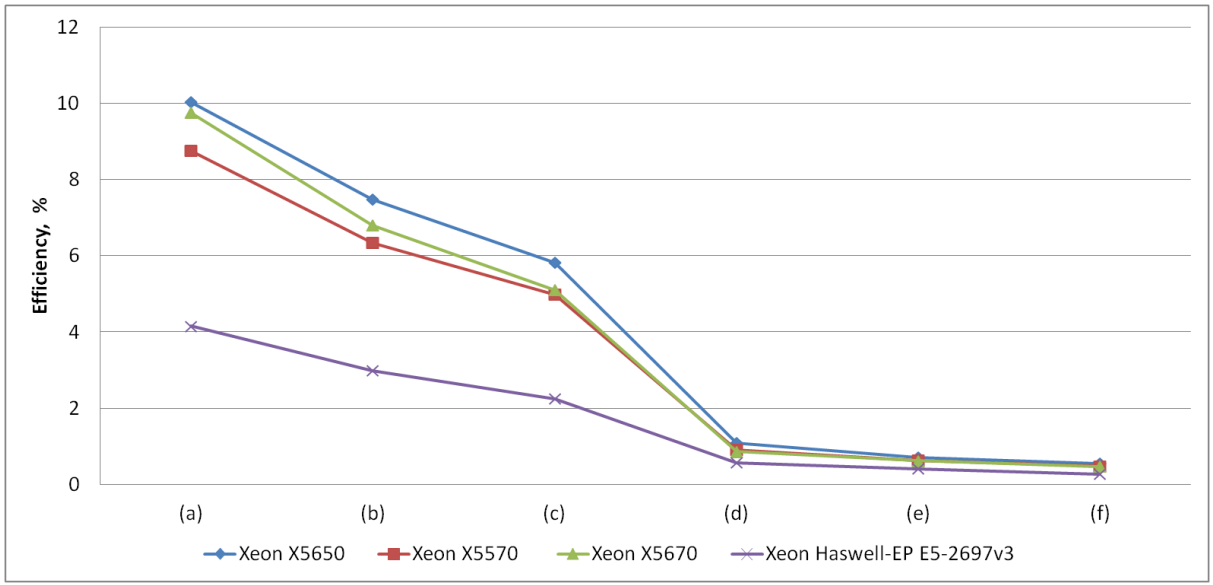




Локальность определяет многое

(Что нужно знать про алгоритмы)

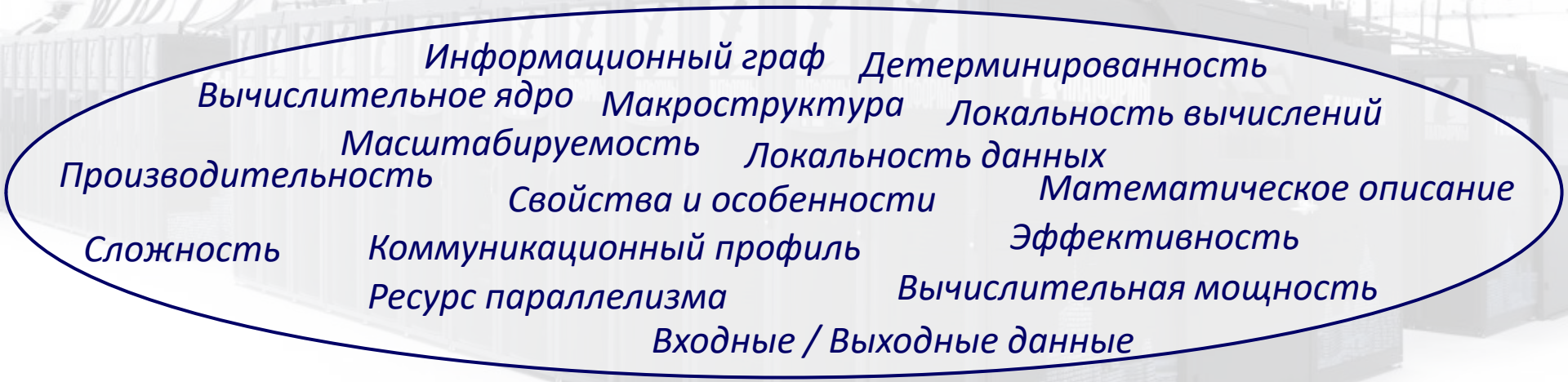
(a) A[i] = B[i]*x + c
(b) A[i] = B[i]*x + C[i]
(c) A[i] = B[i]*X[i] + C[i]
(d) A[ind[i]] = B[ind[i]]*x+c
(e) A[ind[i]] = B[ind[i]]*x+C[ind[i]]
(f) A[ind[i]] = B[ind[i]]*X[ind[i]]+C[ind[i]]





Какие свойства должны войти в “универсальное” (“полное”) описание алгоритмов?

Описание алгоритмов
(Что должно быть учтено в подобном описании?)



Описание алгоритмов
(на примере разложения Холецкого)

Кратко о важном

Свойства алгоритма:

- Последовательная сложность алгоритма: $O(n^3)$
- Высота ярусно-параллельной формы: $O(n)$
- Ширина ярусно-параллельной формы: $O(n^2)$
- Объём входных данных: $\frac{n(n+1)}{2}$
- Объём выходных данных: $\frac{n(n+1)}{2}$

1 Свойства и структура алгоритма

1.1 Общее описание алгоритма

Разложение Холецкого впервые предложено французским офицером и математиком Андре-Луи Холецким в конце Первой Мировой войны, незадолго до его гибели в бою в августе 1918 г. Идея этого разложения была опубликована в 1924 г. его сослуживцем. Потом оно было использовано поляком Т. Банашевичем в 1938 г. В советской математической литературе называется также методом квадратного корня [1-3]; название связано с характерными операциями, отсутствующими в родственном разложении Гаусса.

Первоначально разложение Холецкого использовалось исключительно для плотных симметричных положительно определенных матриц. В настоящее время его использование гораздо шире. Оно может быть применено также, например, к эрмитовым матрицам. Для повышения производительности вычислений часто применяется блочная версия разложения.

Для разреженных матриц разложение Холецкого также широко применяется в качестве основного этапа прямого метода решения линейных систем. В этом случае используют специальные упорядочивания для уменьшения ширины профиля исключения, а следовательно и уменьшения количества арифметических операций. Другие упорядочивания используются для выделения независимых блоков вычислений при работе на системах с параллельной организацией.

1.2 Математическое описание алгоритма

Исходные данные: положительно определённая симметрическая матрица A (элементы a_{ij}).

Вычисляемые данные: нижняя треугольная матрица L (элементы l_{ij}).

Формулы метода:

$l_{11} = \sqrt{a_{11}}$

Описание алгоритмов
(на примере разложения Холецкого)

Общее описание

For positive definite Hermitian matrices (*symmetric matrices in the real case*), we use the decomposition $A = LL^*$, where L is the lower triangular matrix, or the decomposition $A = U^*U$, where U is the upper triangular matrix. These forms of the Cholesky decomposition are equivalent in the sense of the amount of arithmetic operations and are different in the sense of data represntation. The essence of this decomposition consists in the implementation of formulas obtained uniquely for the elements of the matrix L from the above equality. The Cholesky decomposition is widely used due to the following features.

Математическое описание

Input data: a symmetric positive definite matrix A whose elements are denoted by a_{ij}).

Output data: the lower triangular matrix L whose elements are denoted by l_{ij}).

The Cholesky algorithm can be represented in the form

$$l_{11} = \sqrt{a_{11}},$$

$$l_{j1} = \frac{a_{j1}}{l_{11}}, \quad j \in [2, n],$$

$$l_{ii} = \sqrt{a_{ii} - \sum_{p=1}^{i-1} l_{ip}^2}, \quad i \in [2, n],$$

$$l_{ji} = \left(a_{ji} - \sum_{p=1}^{i-1} l_{ip} l_{jp} \right) / l_{ii}, \quad i \in [2, n-1], j \in [i+1, n].$$

Комментарии к алгоритму

The Cholesky decomposition allows one to use the so-called *accumulation mode* due to the fact that the significant part of computation involves *dot product operations*. Hence, these dot products can be accumulated in double precision for additional accuracy. In this mode, the Cholesky method has the least *equivalent perturbation*. During the process of decomposition, no growth of the matrix elements can occur, since the matrix is symmetric and positive definite. Thus, the Cholesky algorithm is unconditionally stable.

Описание алгоритмов
(на примере разложения Холецкого)

Вычислительное ядро

A computational kernel of its serial version can be composed of $\frac{n(n-1)}{2}$ dot products of the matrix rows:

$$\sum_{p=1}^{i-1} l_{ip} l_{jp}.$$

Операции чтения/записи крайне важны!

Последовательная сложность

The following number of operations should be performed to decompose a matrix of order n using a serial version of the Cholesky algorithm:

- n square roots,
- $\frac{n(n-1)}{2}$ divisiona,
- $\frac{n^3-n}{6}$ multiplications and $\frac{n^3-n}{6}$ additions (subtractions): the main amount of computational work.

Операции чтения/записи крайне важны!

Базовая схема реализации

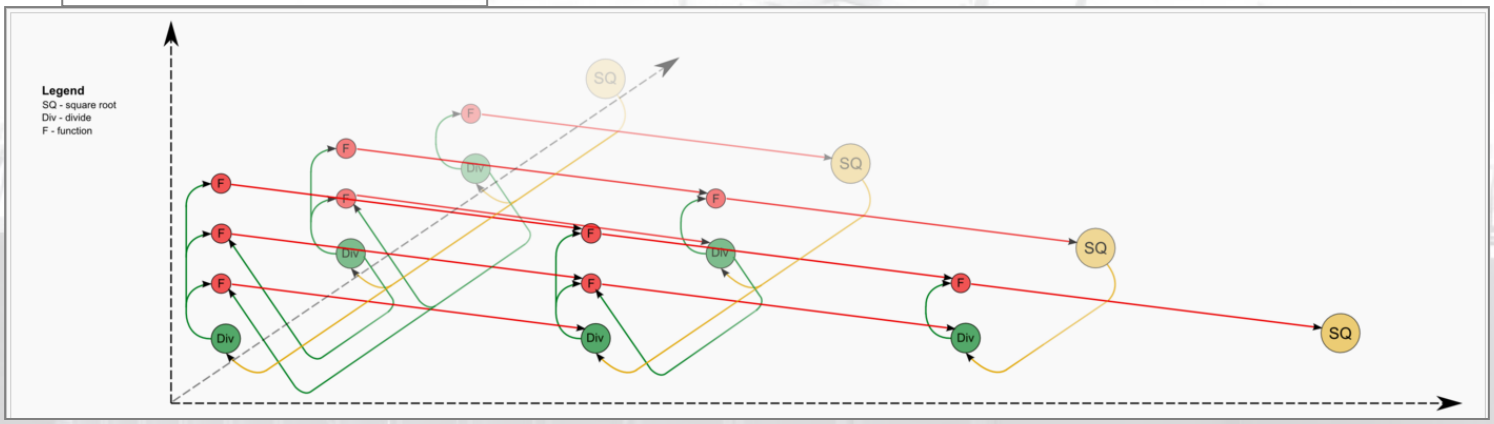
```
DO  I = 1, N
        S = A(I,I)
        DO  IP=1, I-1
                S = S - DPROD(A(I,IP), A(I,IP))
        END DO
        A(I,I) = SQRT (S)
        DO  J = I+1, N
                S = A(J,I)
                DO  IP=1, I-1
                        S = S - DPROD(A(I,IP), A(J,IP))
                END DO
                A(J,I) = S/A(I,I)
        END DO
END DO
```

Дополнительная информация

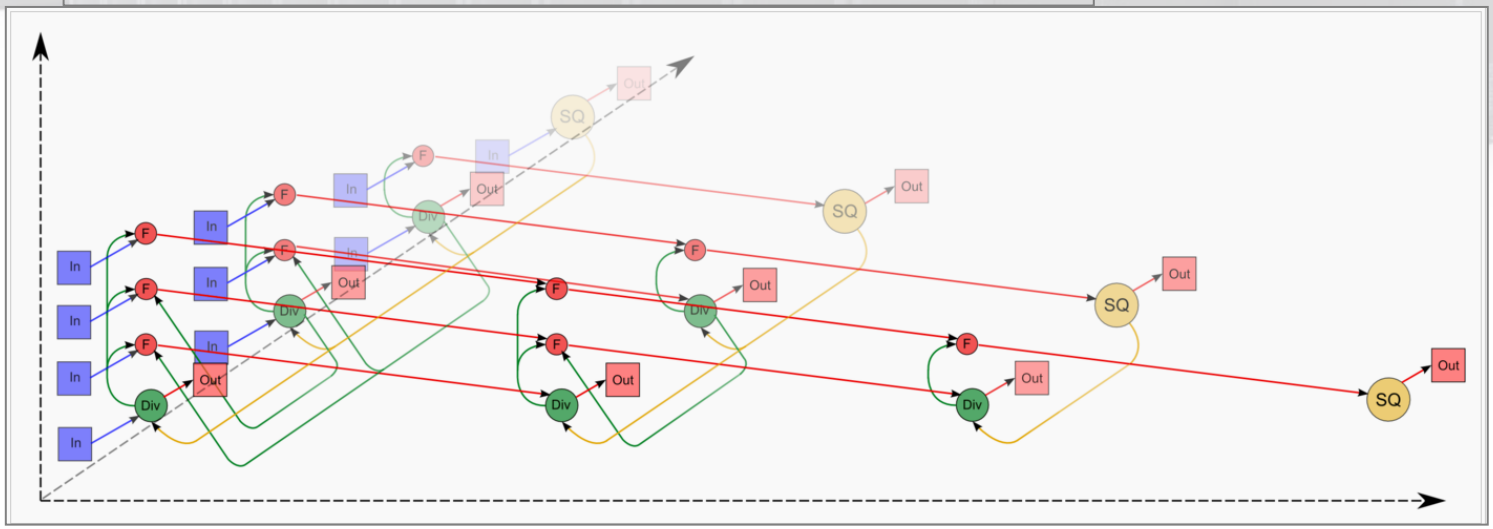
There exist block versions of this algorithm;

Описание алгоритмов
(на примере разложения Холецкого)

Информационная структура



Информационная структура с указанием входных и выходных данных



Описание алгоритмов
(на примере разложения Холецкого)

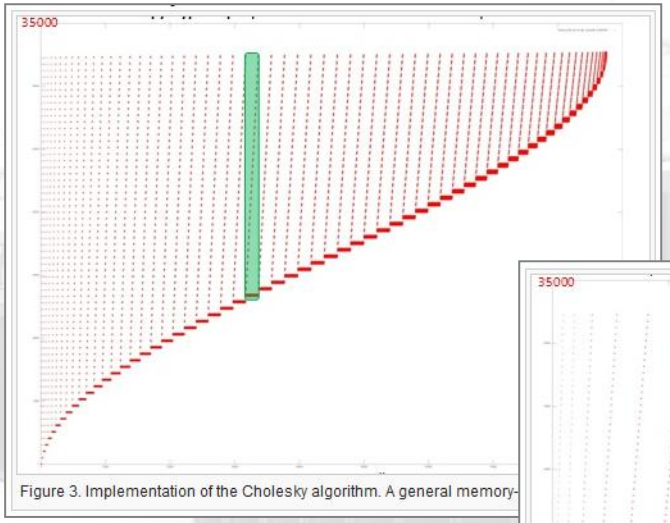


Figure 3. Implementation of the Cholesky algorithm. A general memory-

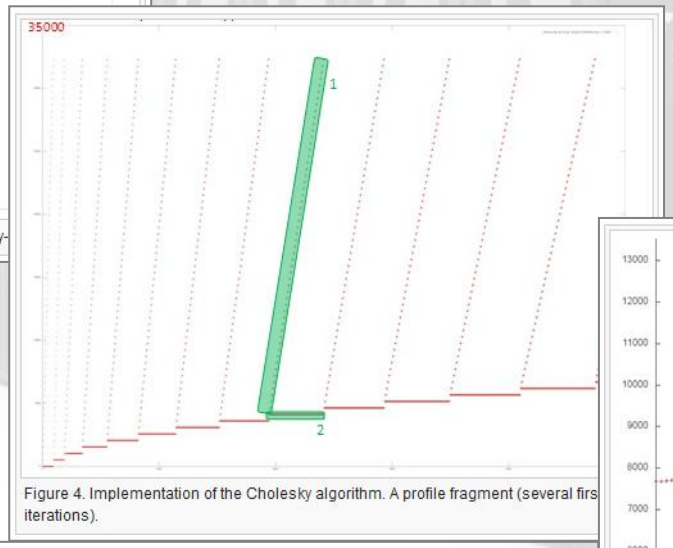


Figure 4. Implementation of the Cholesky algorithm. A profile fragment (several firs iterations).

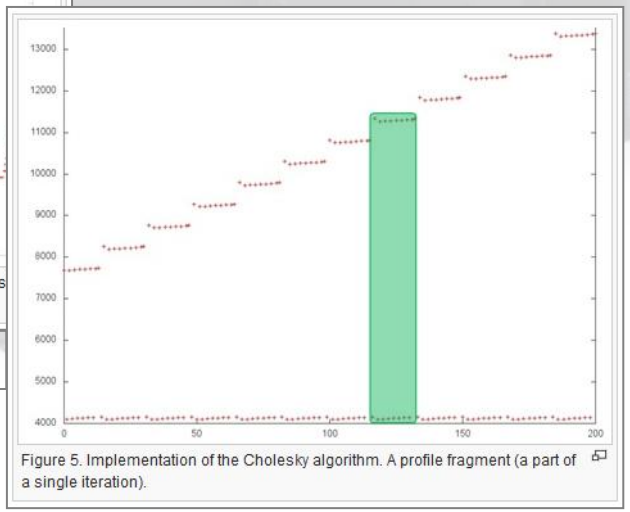
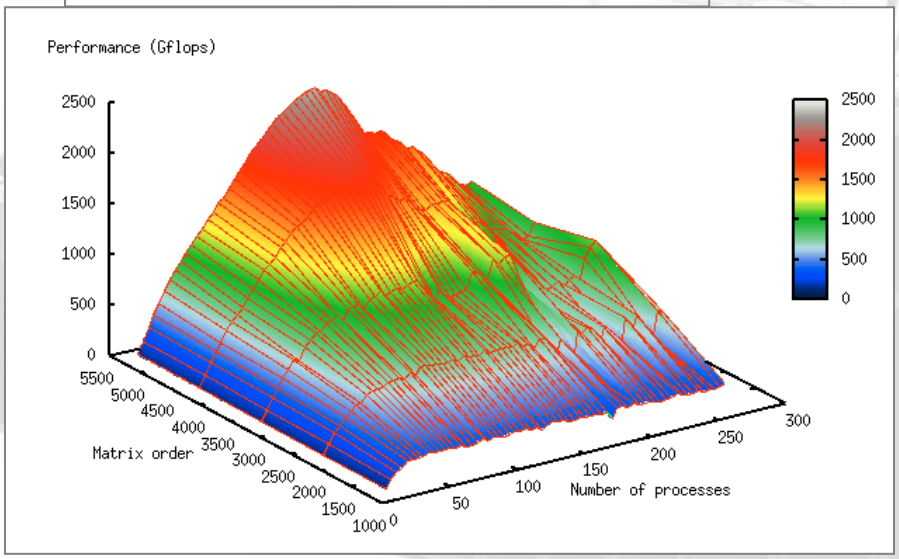


Figure 5. Implementation of the Cholesky algorithm. A profile fragment (a part of a single iteration).

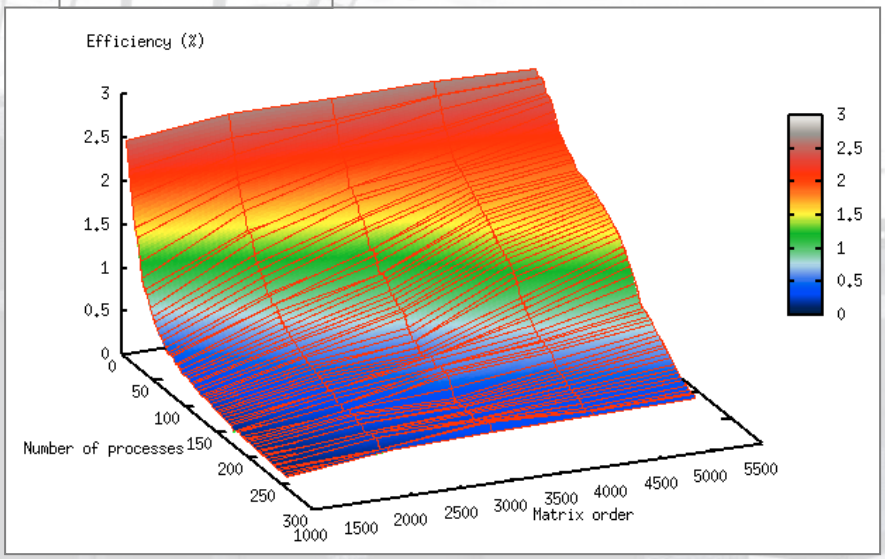
Локальность данных (профиль работы с памятью)

Описание алгоритмов
(на примере разложения Холецкого)

Масштабируемость (производительность) *



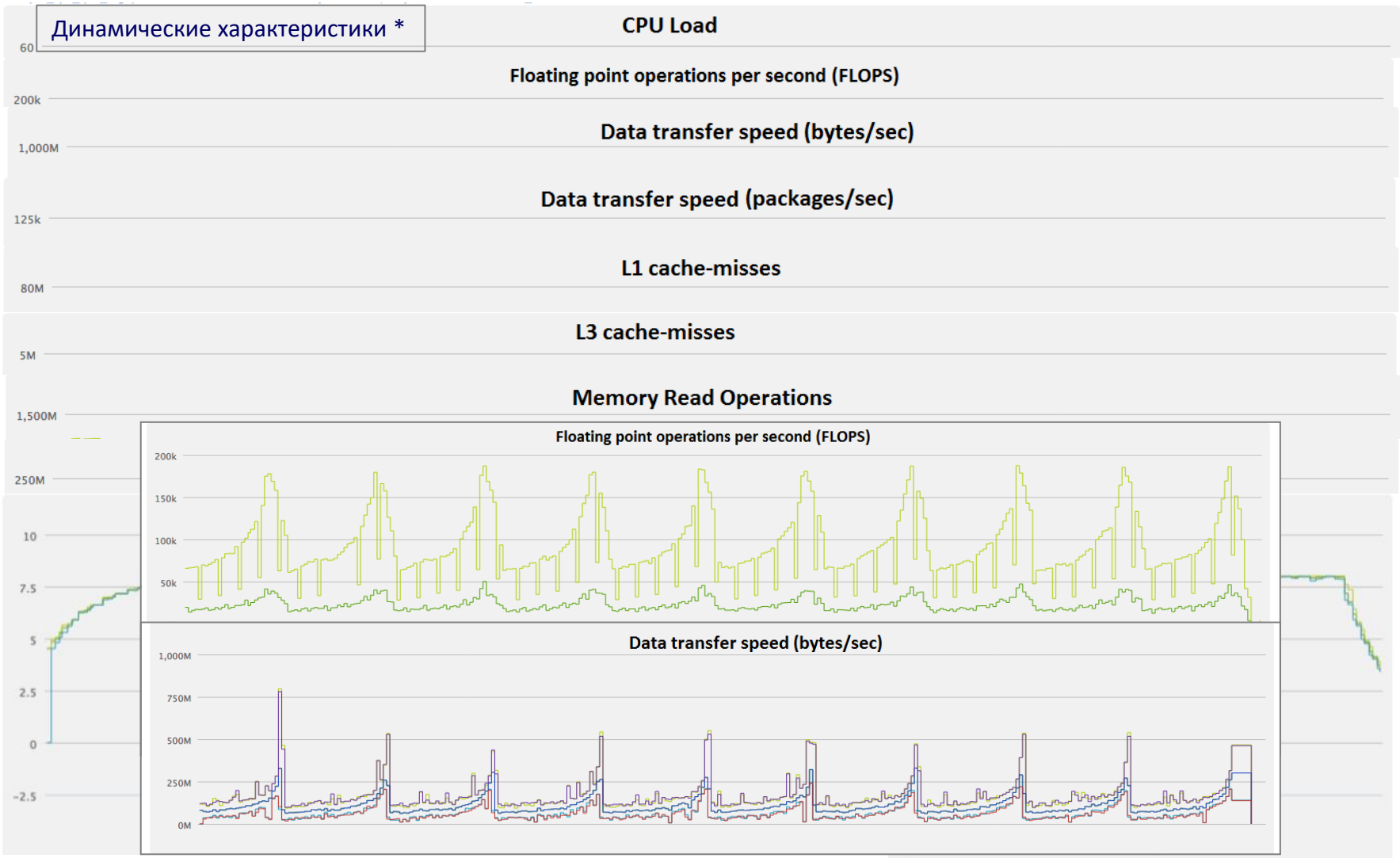
Эффективность *



* Масштабируемость, производительность, эффективность измерялись на суперкомпьютере МГУ “Ломоносов”.

Описание алгоритмов
(на примере разложения Холецкого)

Динамические характеристики *



* Динамические характеристики получены на суперкомпьютере МГУ “Ломоносов”.

Это очень полезная информация об алгоритме, она действительно нужна.

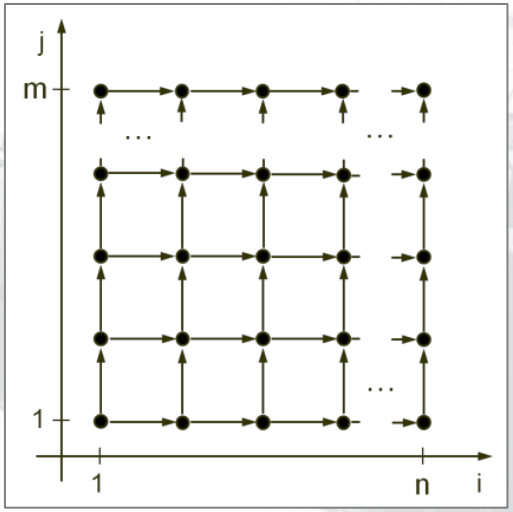
Но…

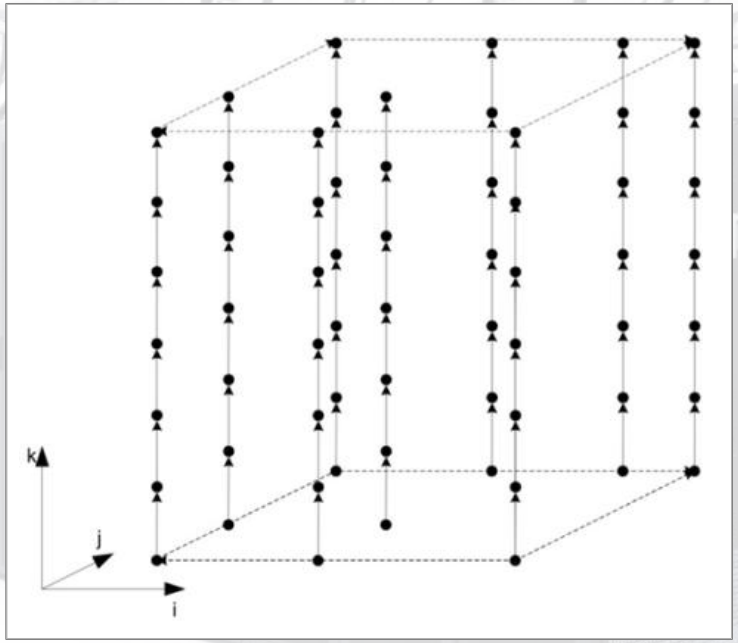
В процессе создания полного описания алгоритма возникает не просто много проблем.

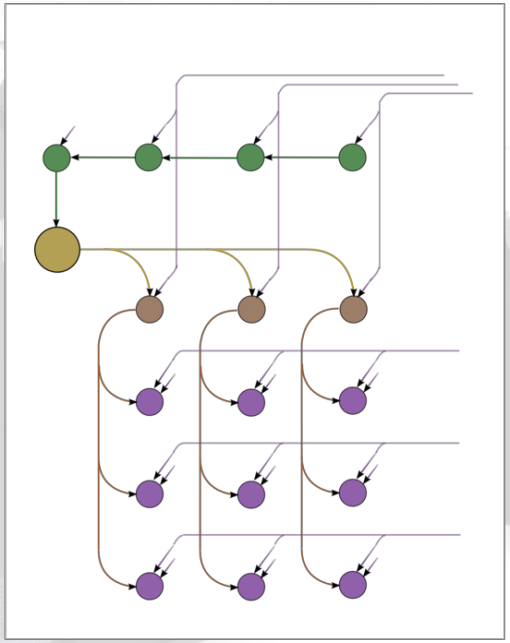
Возникает очень много сложных проблем !

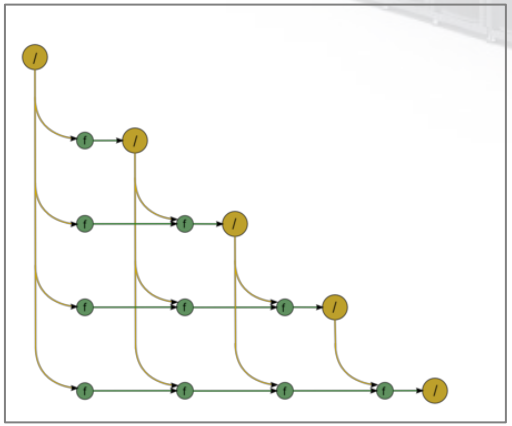
Информационная структура: как получать, описывать, показывать… ?

(сложности описания алгоритмов)





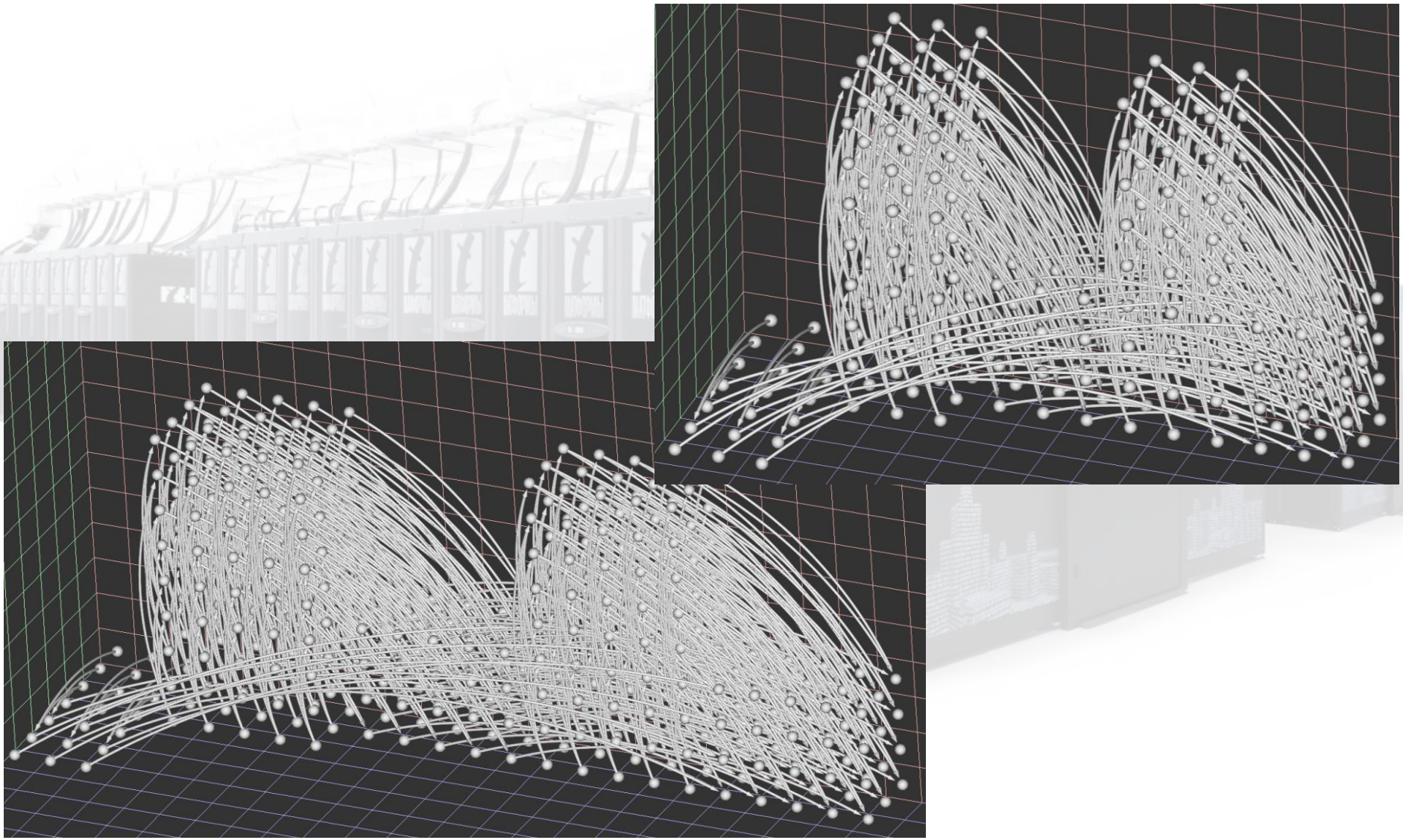




- Как изобразить потенциально бесконечный граф ?
- Как изобразить потенциально многомерный граф ?
- Как показать зависимость структуры графа от размера задачи ?

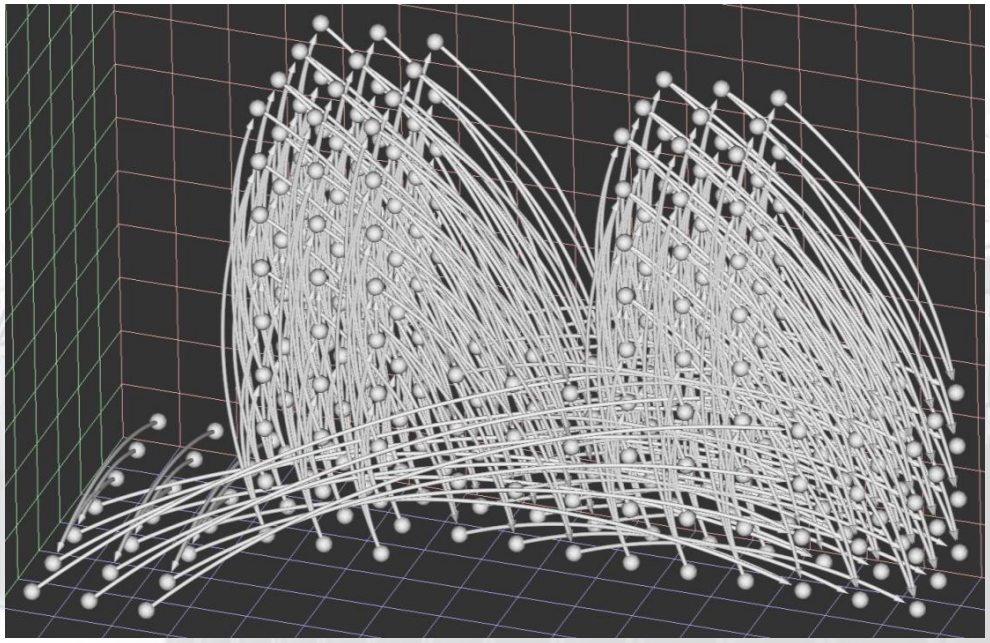
Информационная структура: как получать, описывать, показывать… ?

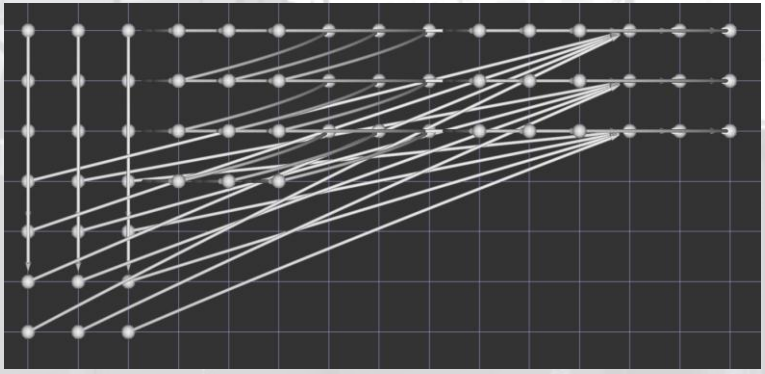
(сложности описания алгоритмов)

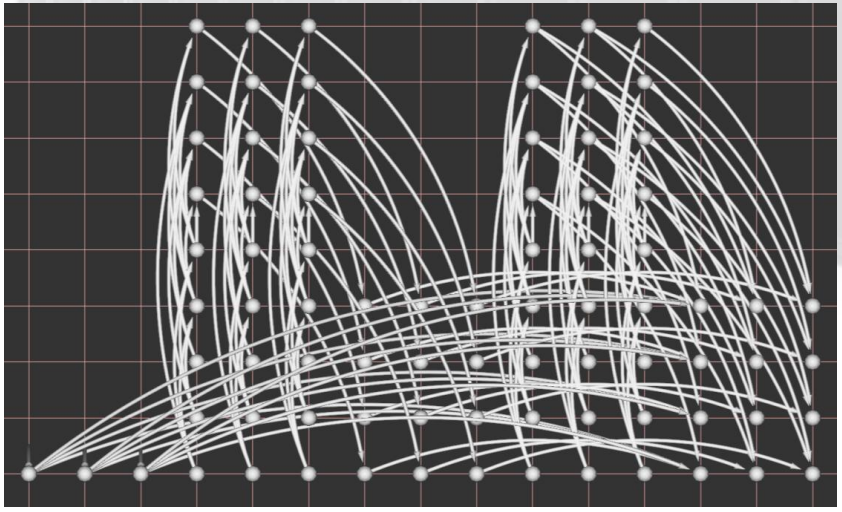


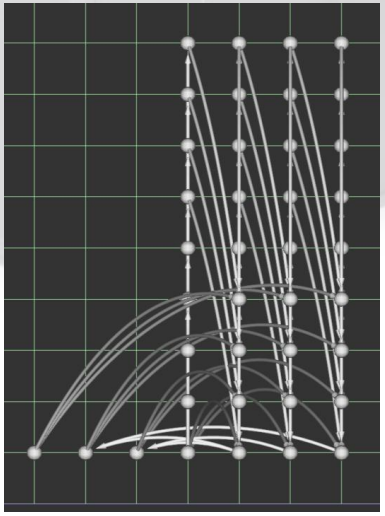
Информационная структура: как получать, описывать, показывать… ?

(сложности описания алгоритмов)

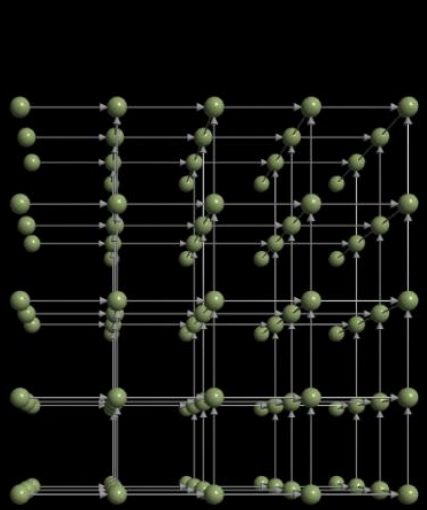


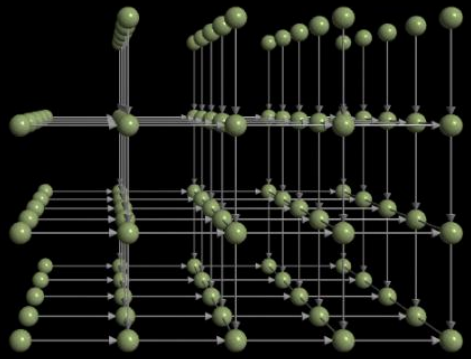


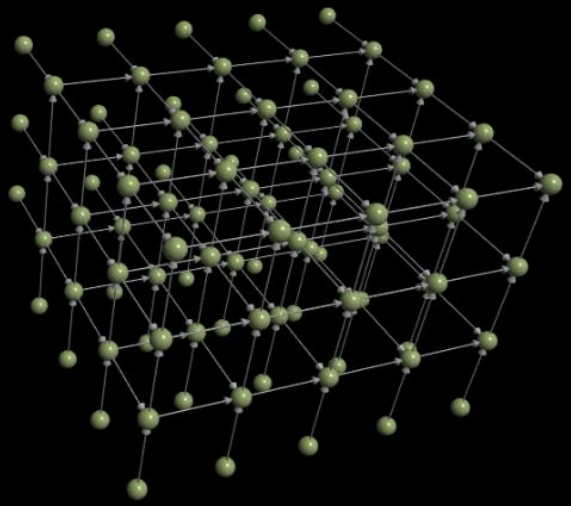


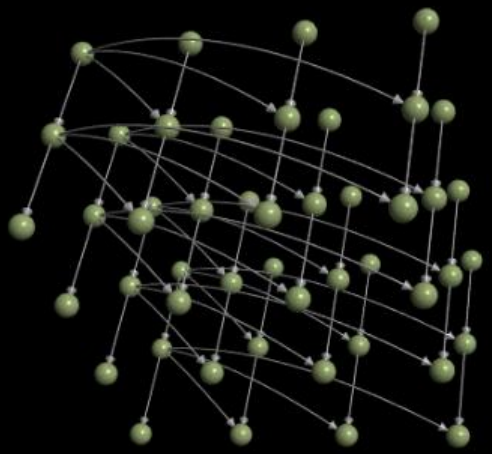


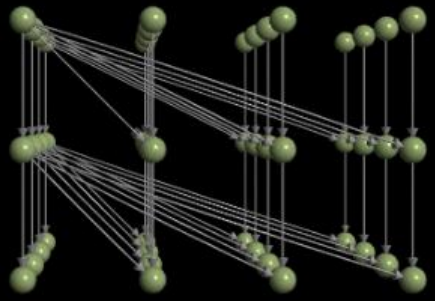
Информационная структура алгоритмов и программ

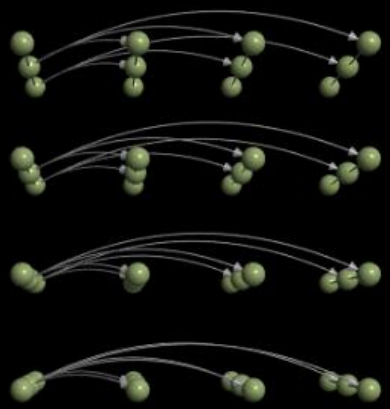




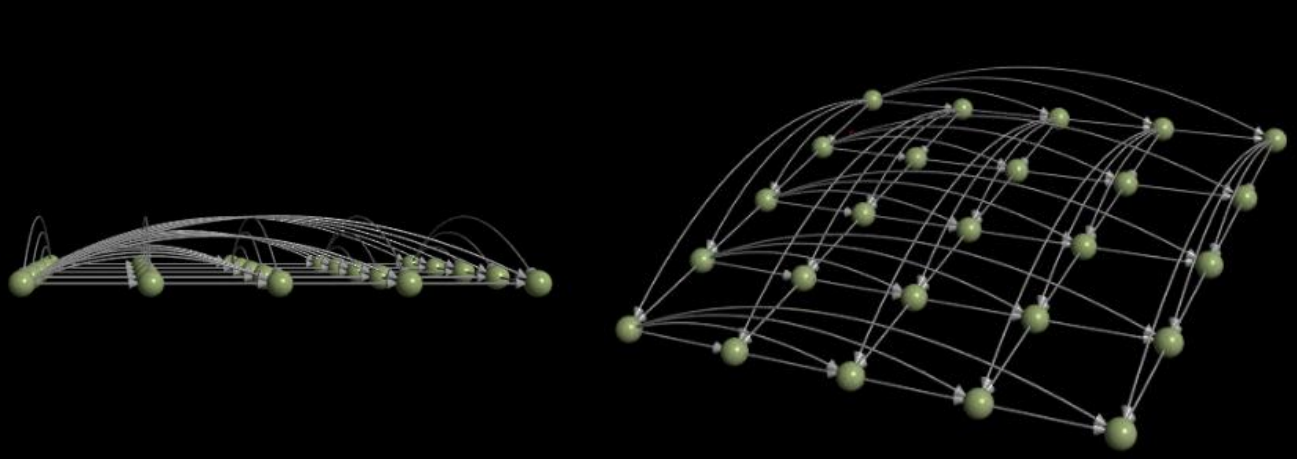




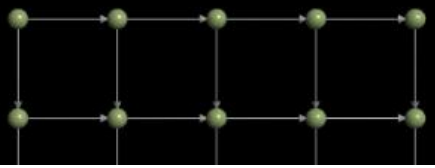


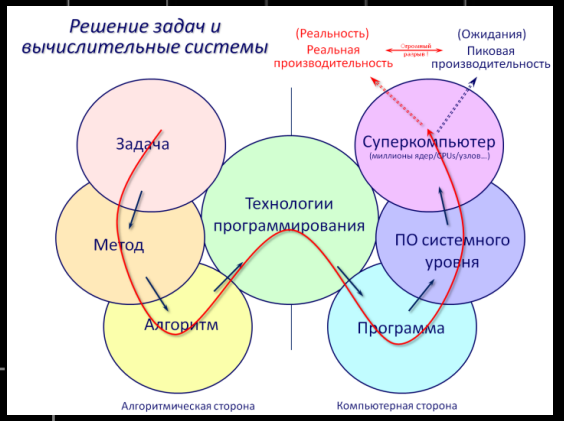


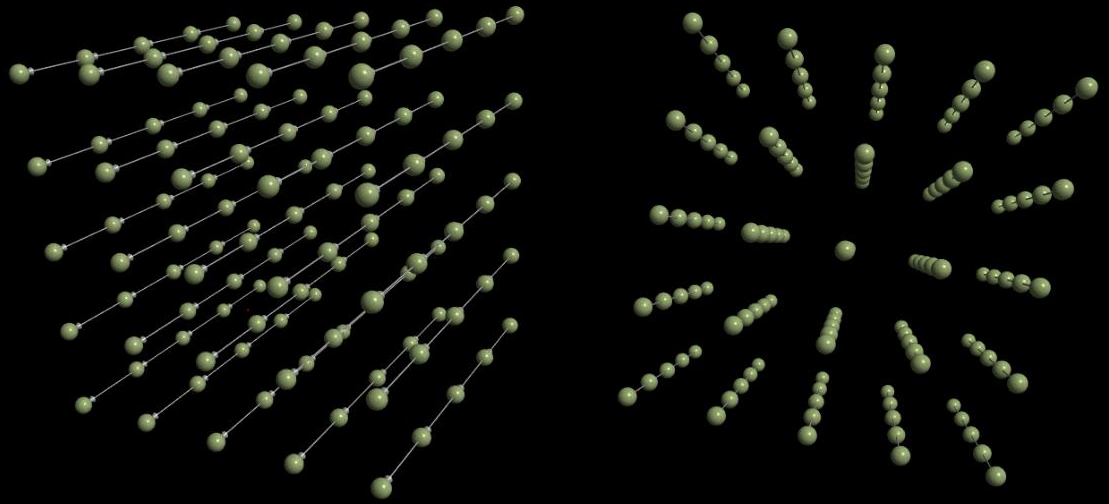
Информационная структура алгоритмов и программ



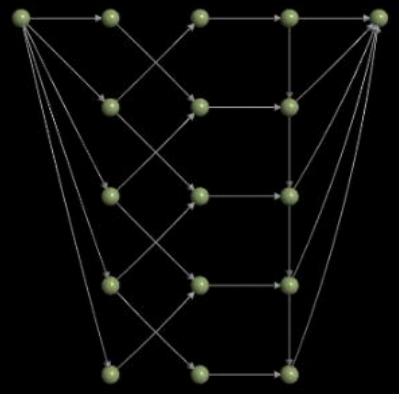
Типовые алгоритмические структуры







Огромный ресурс параллелизма



А такой структуры быть не может …

Информационная структура: как получать, описывать, показывать… ?

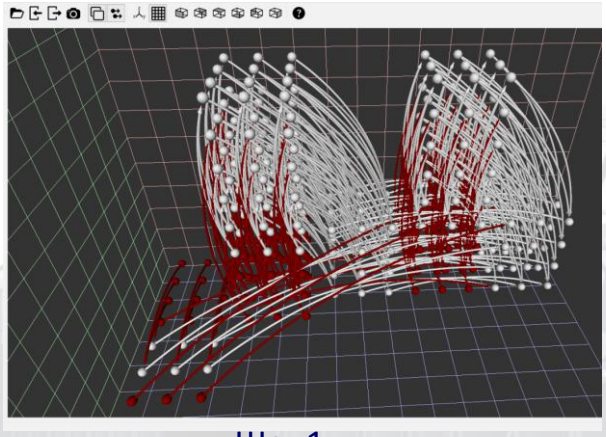
(сложности описания алгоритмов)



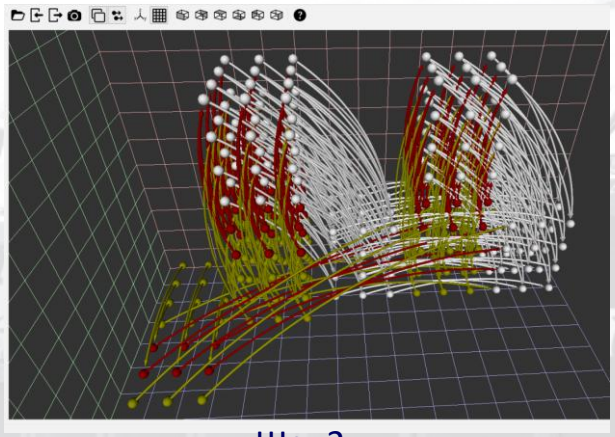
Как выразить имеющийся параллелизм и показать возможный способ параллельного исполнения ?

Информационная структура: как получать, описывать, показывать… ?

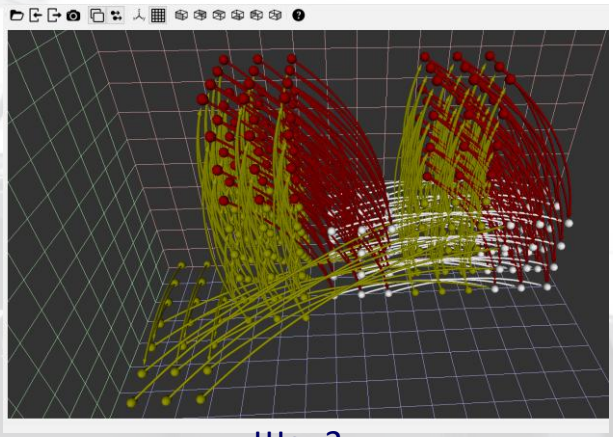
(сложности описания алгоритмов)



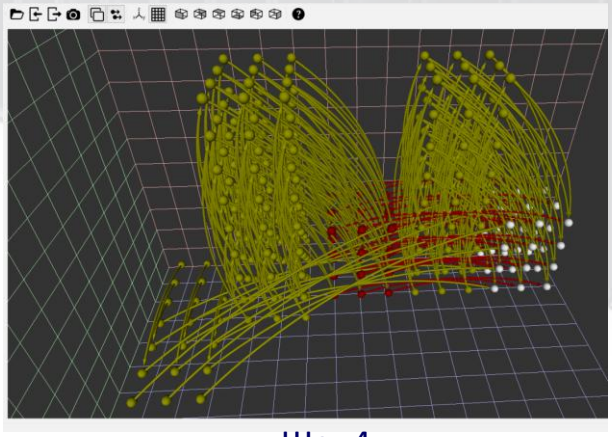
Шаг 1



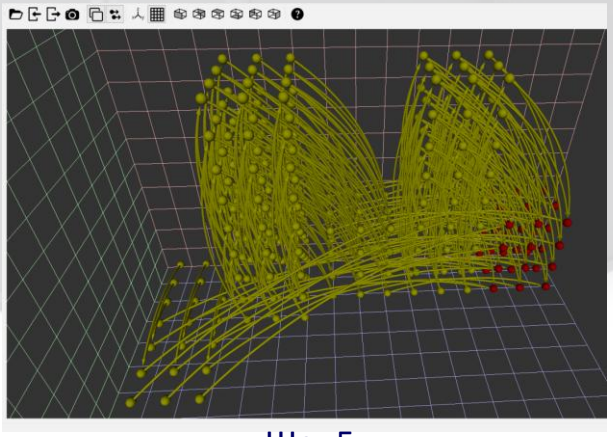
Шаг 2



Шаг 3



Шаг 4



Шаг 5

Цвета в ярусно-параллельной форме:

Красный – слой выполняющихся операций; **Зеленый** – уже выполненные операции;

Белый – операции должны выполниться позже.

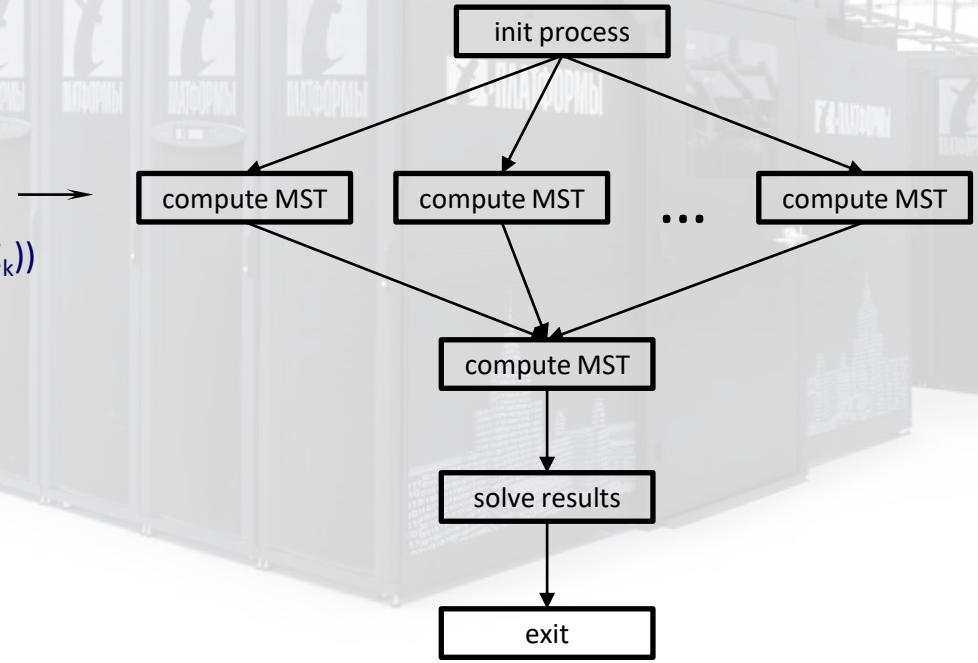
Потенциальный параллелизм: как находить, описывать, показывать… ?

(сложности описания алгоритмов)

$E = E_1 \cup E_2 \cup \ldots \cup E_k$

$MST(E) = MST(E_1 \cup E_2 \cup \ldots \cup E_k) =$

$= MST(\ MST(E_1) \cup MST(E_2) \cup \ldots \cup MST(E_k))$

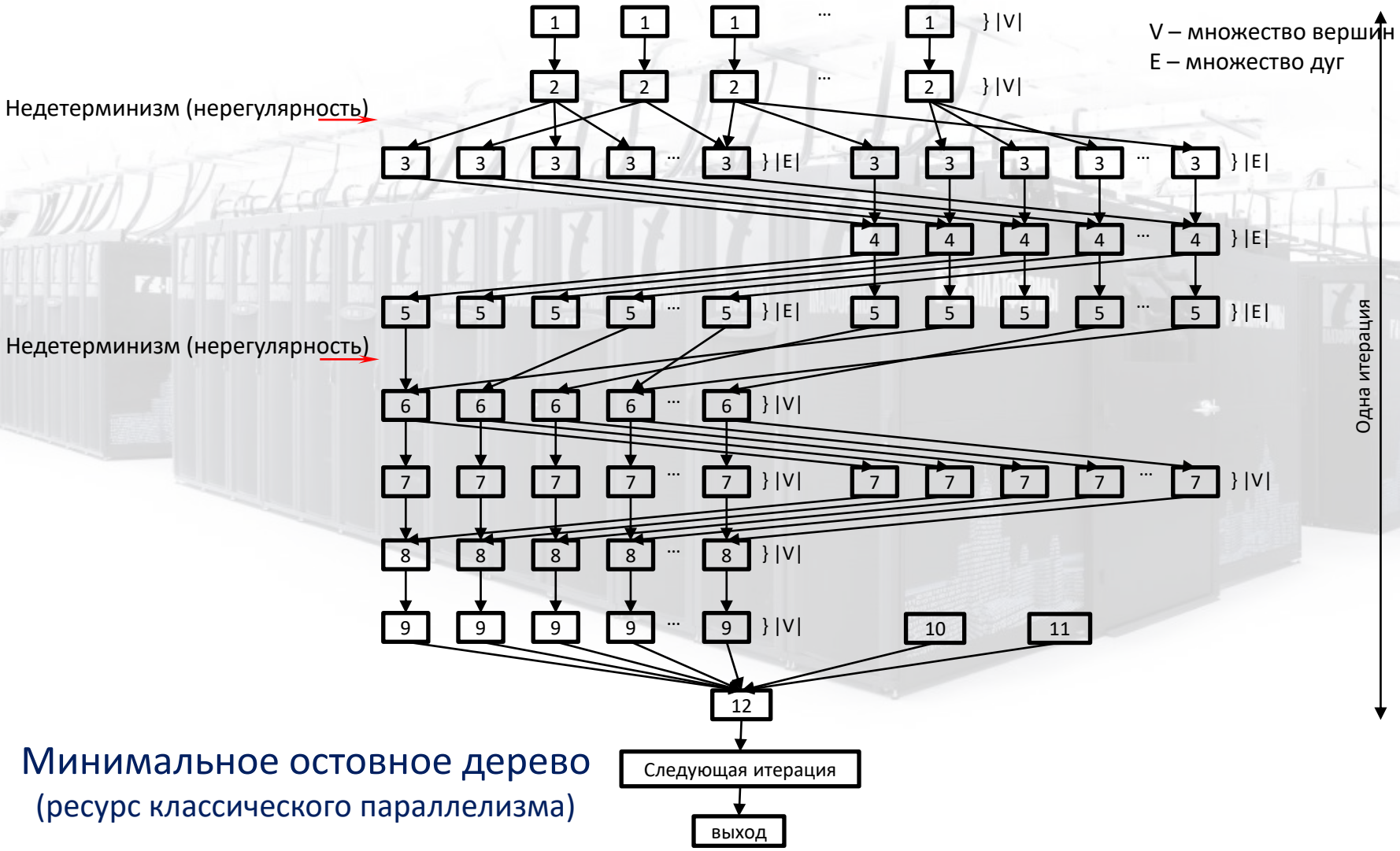


Минимальное остовное дерево (MST)

(ресурс “математического” параллелизма)

Потенциальный параллелизм: как находить, описывать, показывать… ?

(сложности описания алгоритмов)



Минимальное остовное дерево

(ресурс классического параллелизма)

Возможные источники несбалансированности: это важно, это нужно отметить

(сложности описания алгоритмов)

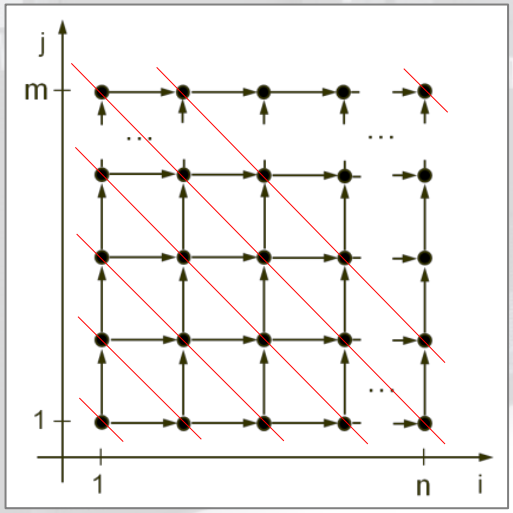


- Баланс между арифметическими операциями +/- and * ;

- Баланс между арифметическими операциями и “чтениями/записями”;

Возможные источники несбалансированности: это важно, это нужно отметить
(сложности описания алгоритмов)

- Баланс между арифметическими операциями +/- and * ;
- Баланс между арифметическими операциями и “чтениями/записями”;
- Баланс в числе операций, которые могут выполняться параллельно;



- Баланс между вычислительной и коммуникационной частями…

Возможные источники недетерминизма: это важно, это нужно отметить
(сложности описания алгоритмов)

• Структура входных данных (разреженные матрицы, графы произвольной структуры…);

• Итерационная природа алгоритмов;

• Датчики случайных чисел;

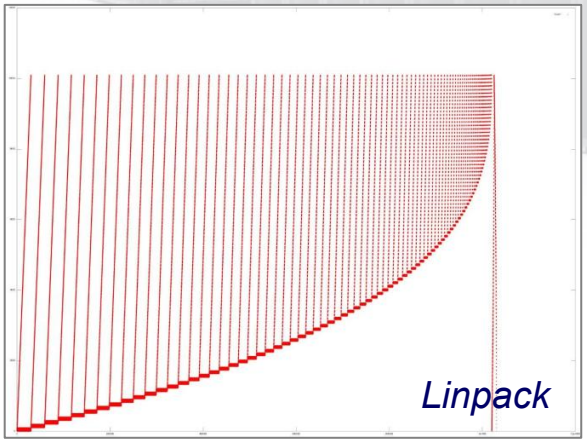
• Отсутствие повторяемости результатов из-за разного порядка выполнения ассоциативных операций…

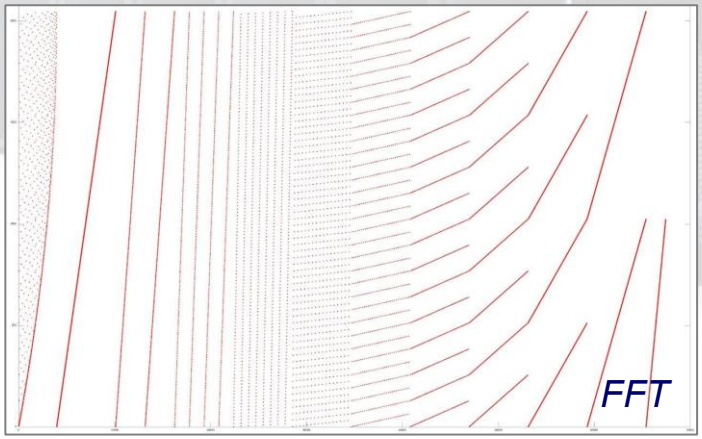
Локальность данных: множество открытых вопросов
(сложности описания алгоритмов)

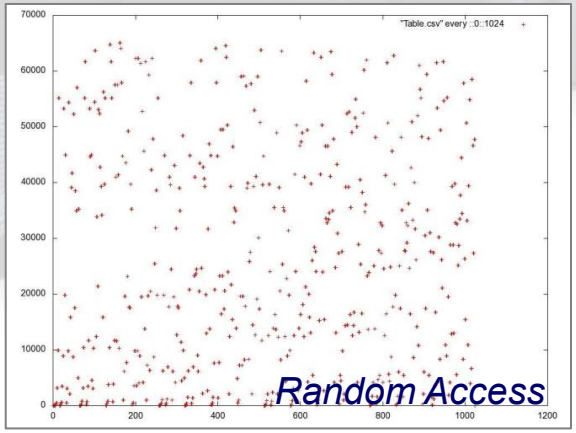
Как оценивать пространственную* и временную** локальность данных в программе ?

Как сравнивать пространственную и временную локальность данных разных программ ?

* Пространственная локальность данных – насколько близко друг к другу расположены последовательные обращения в память.
** Временная локальность данных – насколько часто происходят обращения к одним и тем же данным.







Локальность данных: множество открытых вопросов
(сложности описания алгоритмов)

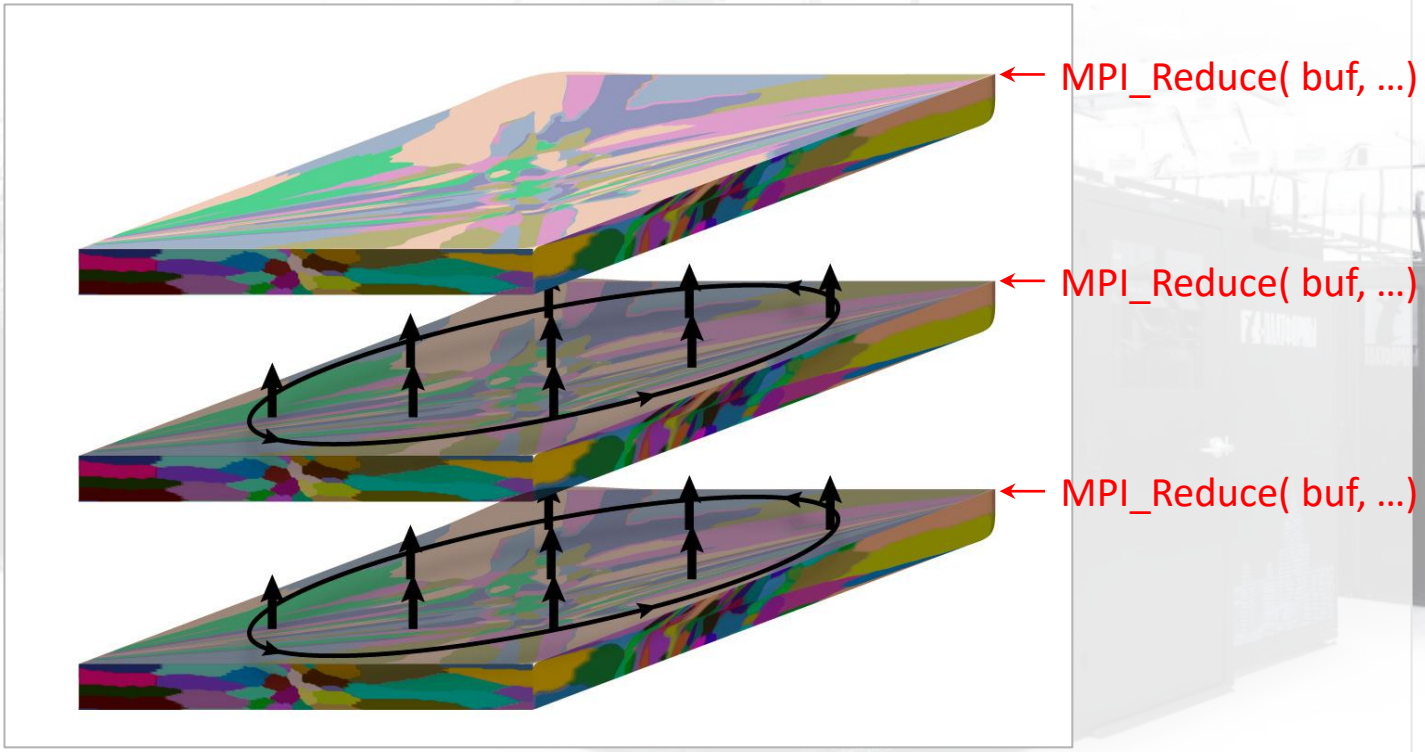
Как оценивать пространственную и временную локальность данных в программе ?

Как сравнивать пространственную и временную локальность данных разных программ ?

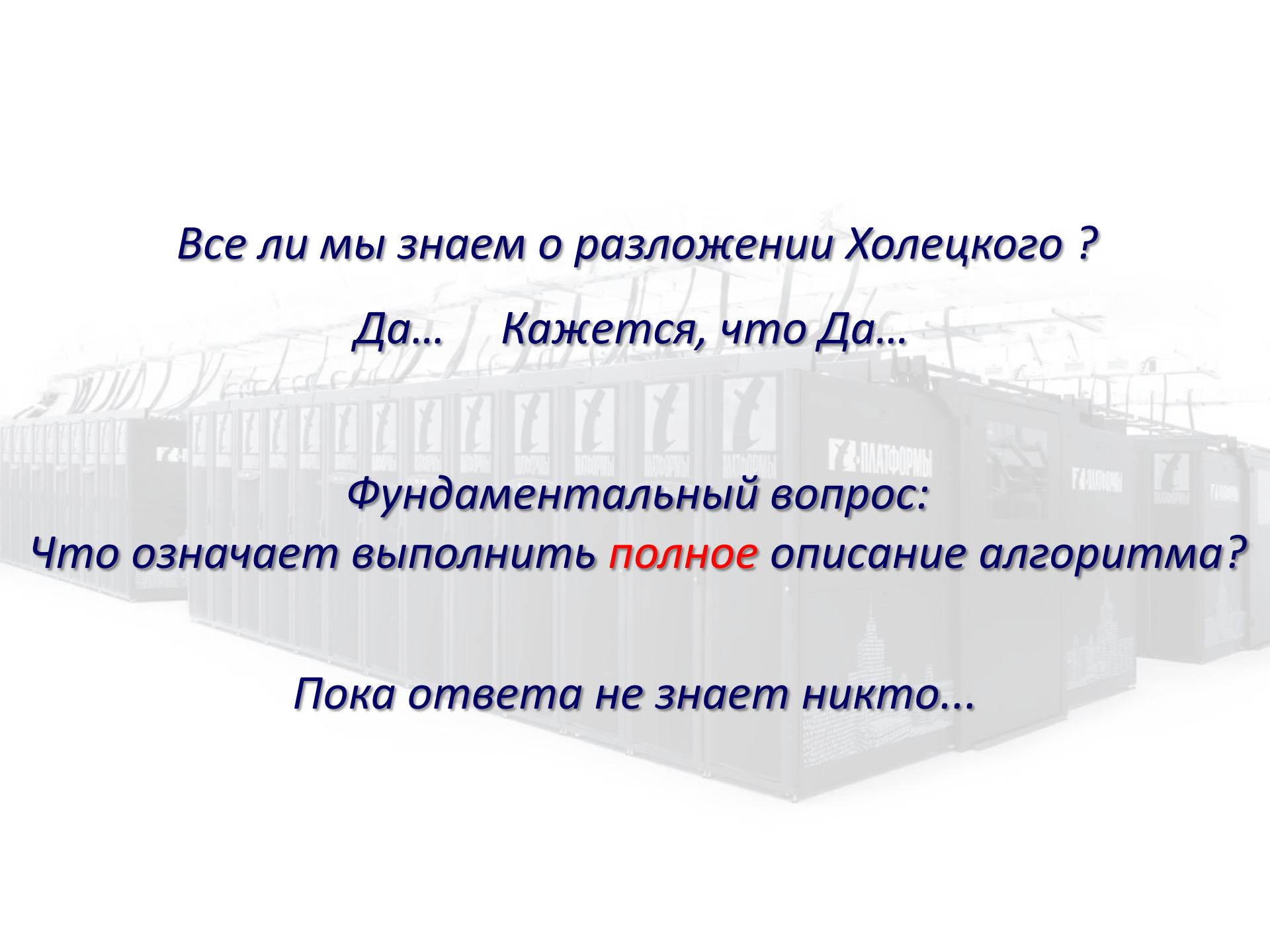
Можем ли мы предсказывать локальность данных в будущих реализациях, опираясь только на информацию об алгоритмах?

Что означает “локальность алгоритма” ?
Что означает “алгоритм обладает хорошей/плохой локальностью” ?
В алгоритмах нет структур данных, а значит и понятие “локальность” к ним напрямую не применимо! Вместе с этим, алгоритмы определяют суть программ, где локальность уместна и важна.

Коммуникационный профиль приложений
(сложности описания алгоритмов)



А что такое коммуникационный профиль приложений?
Как его получить и описывать? (Scalasca, Vampir…)



Все ли мы знаем о разложении Холецкого ?

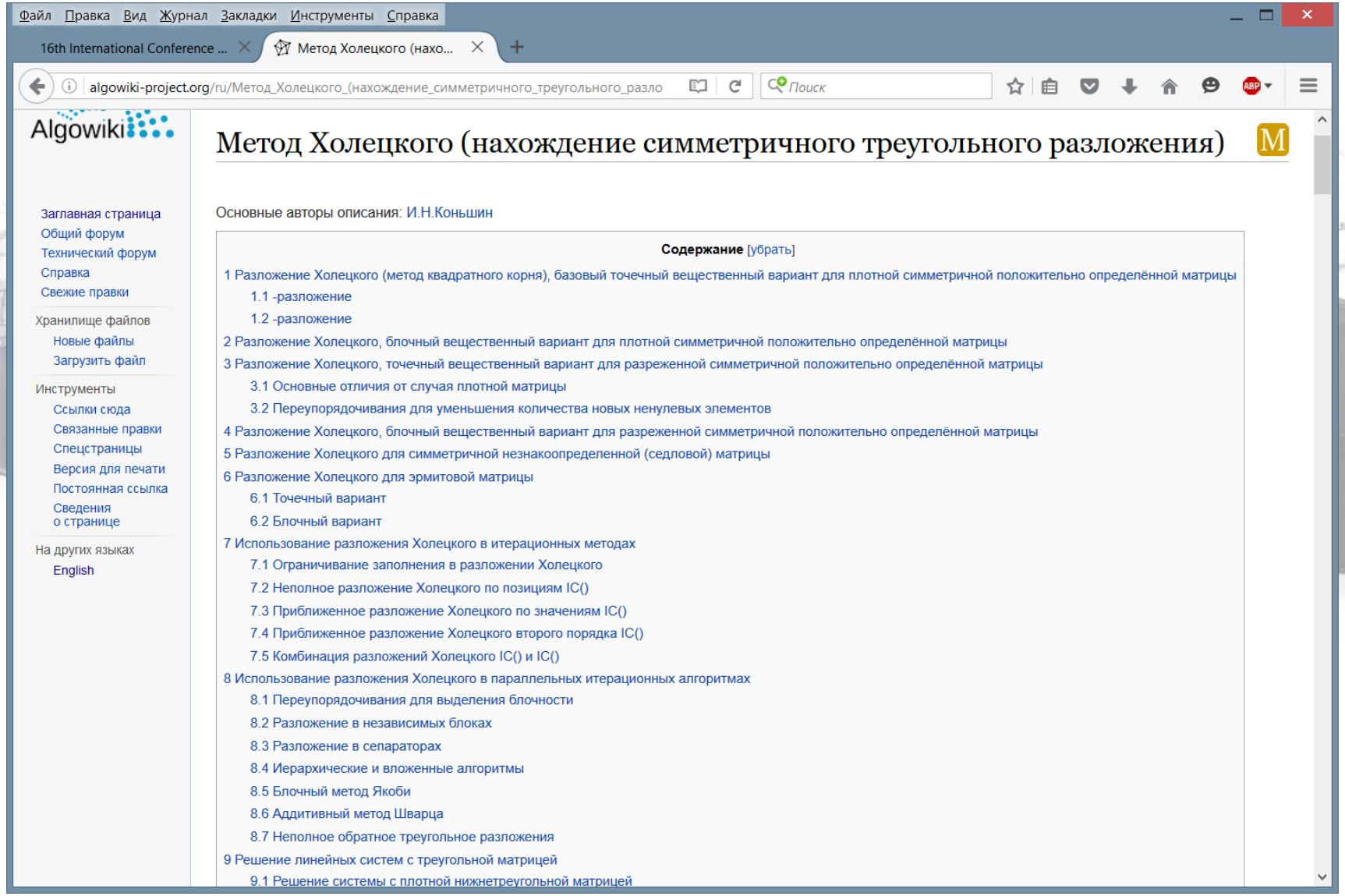
Да… Кажется, что Да…

Фундаментальный вопрос:

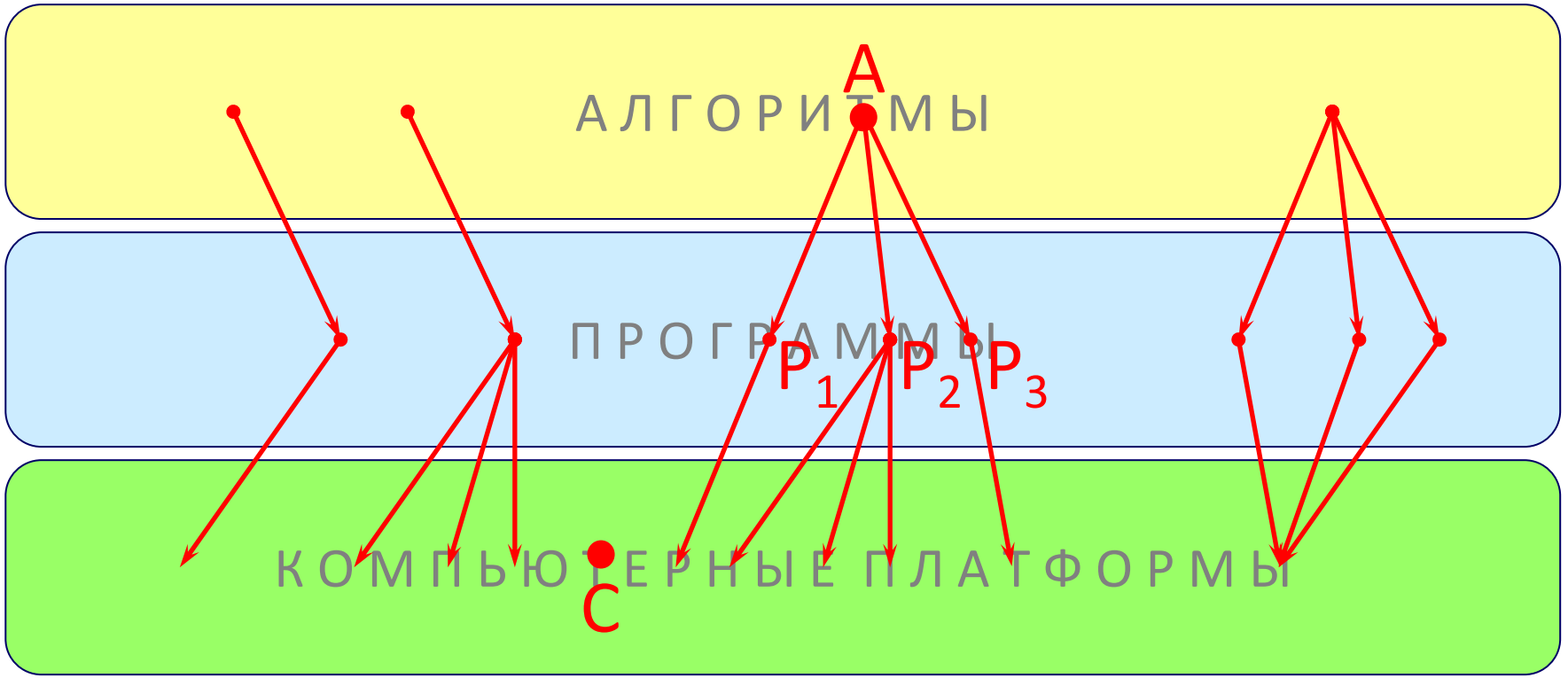
Что означает выполнить полное описание алгоритма?

Пока ответа не знает никто...

Разновидности одного алгоритма…
(сложности описания алгоритмов)



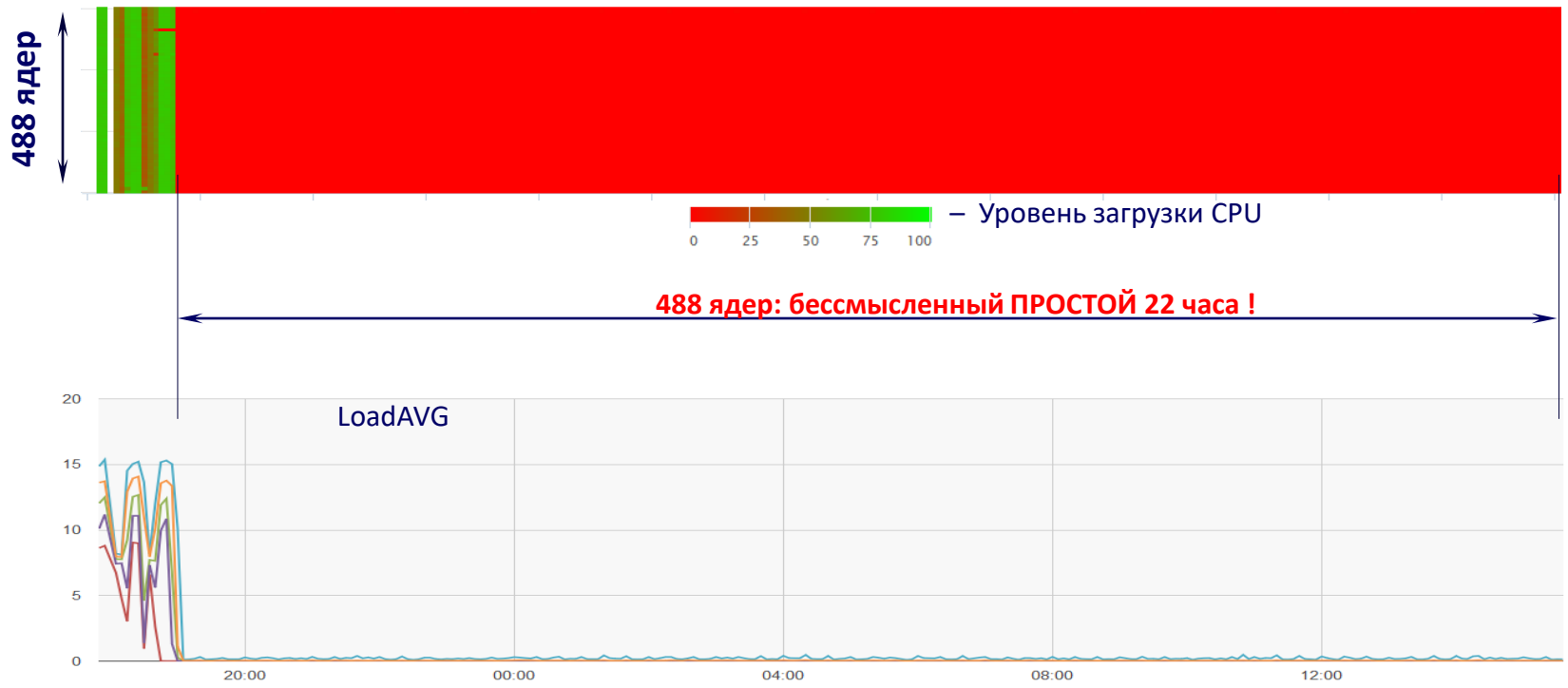
Алгоритмы и их реализации



Достаточно ли у нас информации об алгоритме A, чтобы создать эффективную реализацию для платформы C?

Можем ли мы использовать реализации P_1, P_2, P_3 или мы должны создавать еще одну реализацию P для платформы C?

Эффективность суперкомпьютерных приложений



Что может быть причиной такой ситуации?

- Ошибка в аппаратуре? Да, может быть и так…
- Ошибка в системном ПО? Да, может быть и так…
- Ошибка в коде? Да, может быть и так…
- Проблемы с алгоритмом? Да, может быть и так…

Идеи суперкомпьютерного кодизайна
(Supercomputing Co-Design)



Идеи суперкомпьютерного кодизайна
(Supercomputing Co-Design)



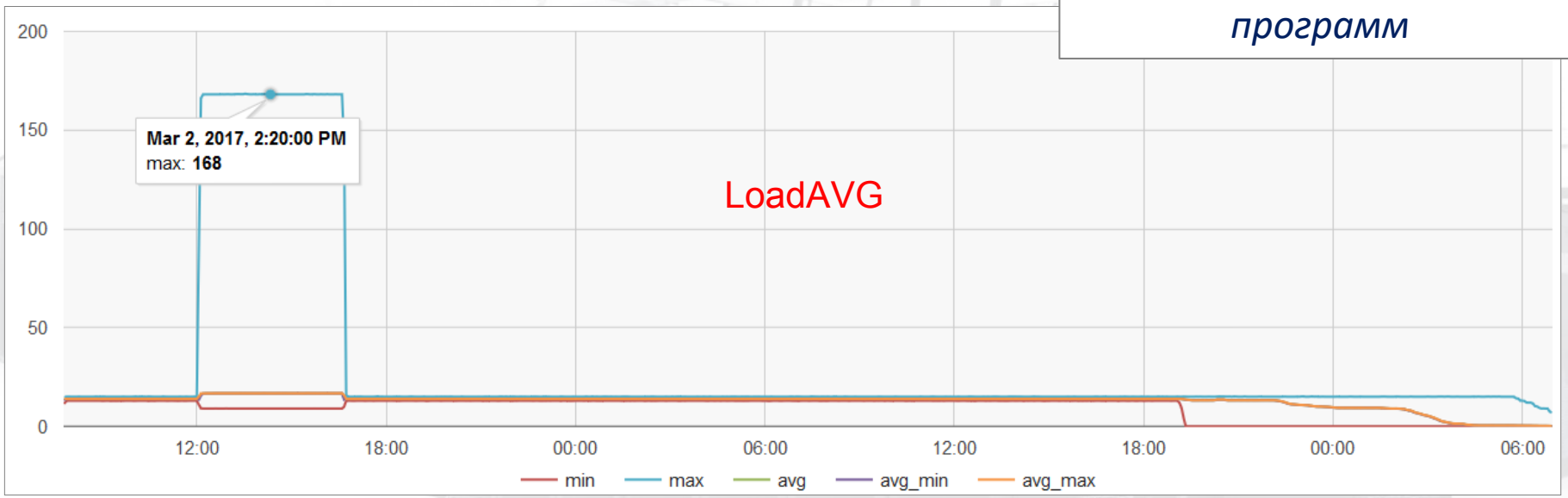
Идеи суперкомпьютерного кодизайна
(Supercomputing Co-Design)

Как обеспечить эффективность цепочки суперкомпьютерного кодизайна?

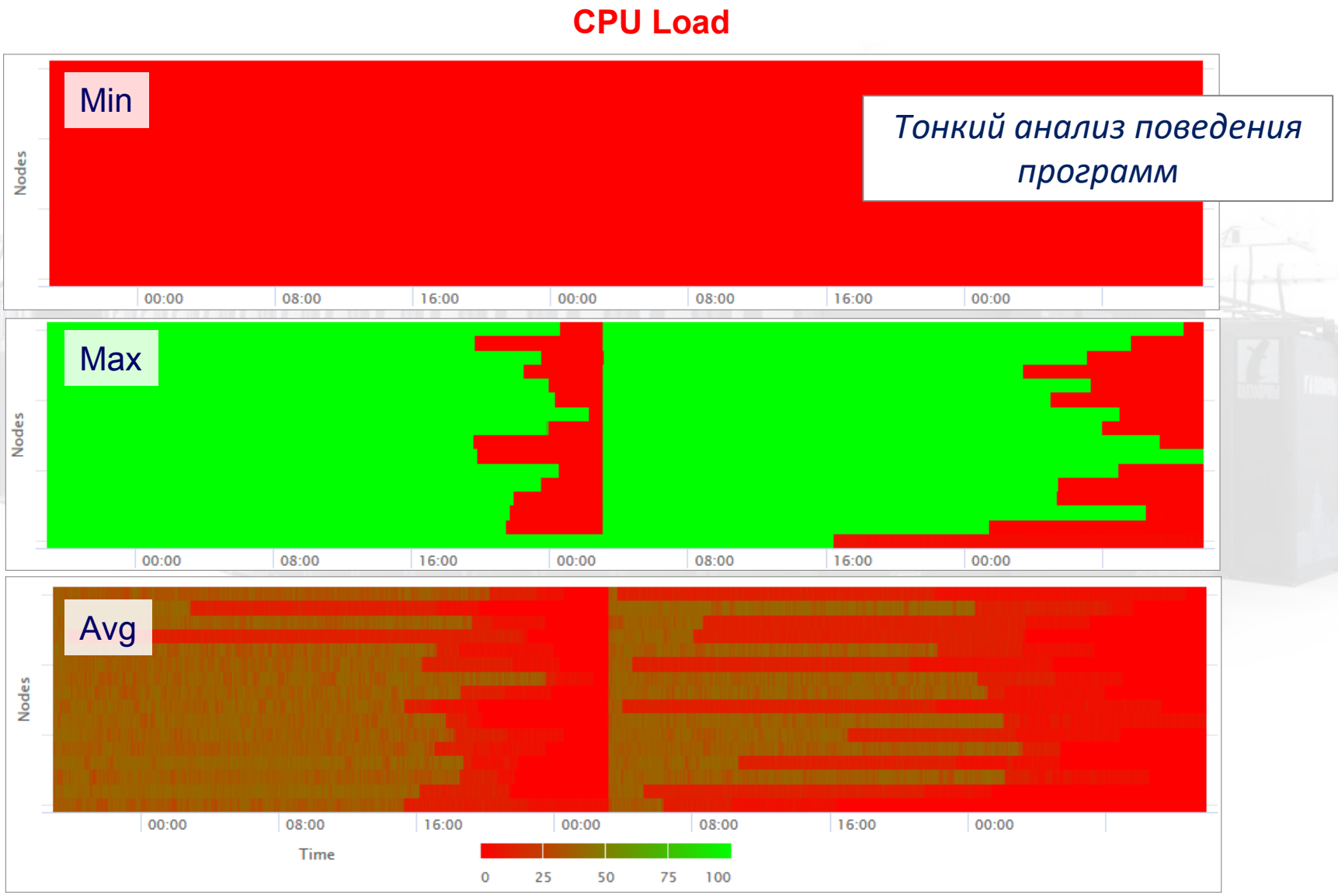


Эффективность суперкомпьютерных приложений

Тонкий анализ поведения программ



Эффективность суперкомпьютерных приложений

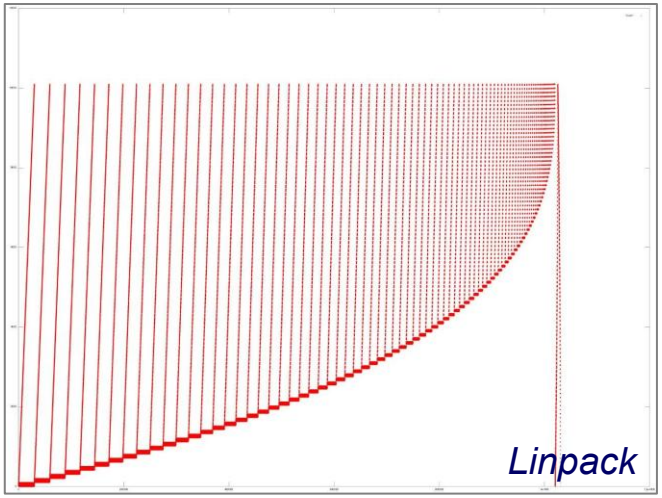


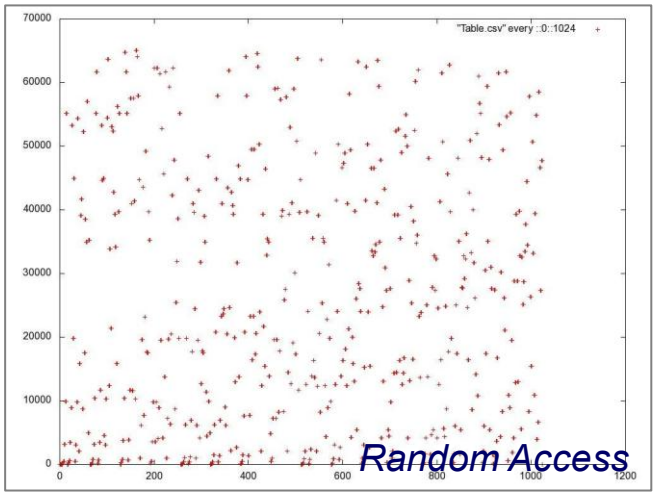
Тест HPCG для суперкомпьютеров
(еще один аргумент в пользу суперкомпьютерного кодизайна)

June 2017 HPCG Results

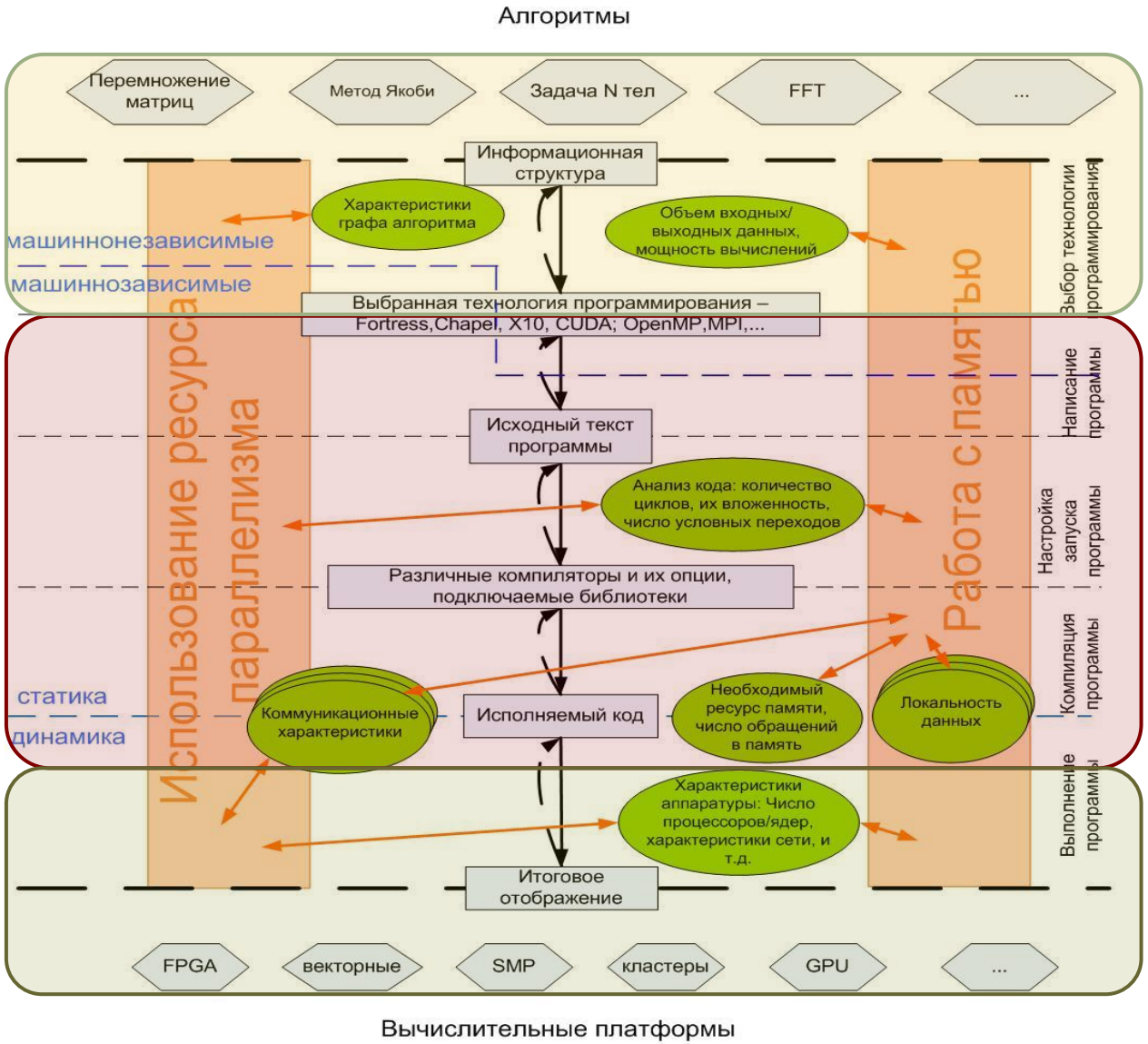
Rank	Site	Computer	Cores	HPL Rmax (Pflop/s)	TOP500 Rank	HPCG (Pflop/s)	Fraction of Peak
1	RIKEN Advanced Institute for Computational Science Japan	**K computer** – , SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10.510 94%	8	0.6027	5.3%
3	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** – Sunway MPP, SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93.015 74%	1	0.4808	0.4%
33	CEIST / JAMSTEC Japan	**Earth Simulator** – NEC SX-ACE, NEC SX-ACE, NEC	8,192	0.487		0.0578	11.0%

В чем причина достижения высокого или низкого % от пиковой производительности?





Элементы суперкомпьютерного кодизайна



Структура и свойства алгоритмов

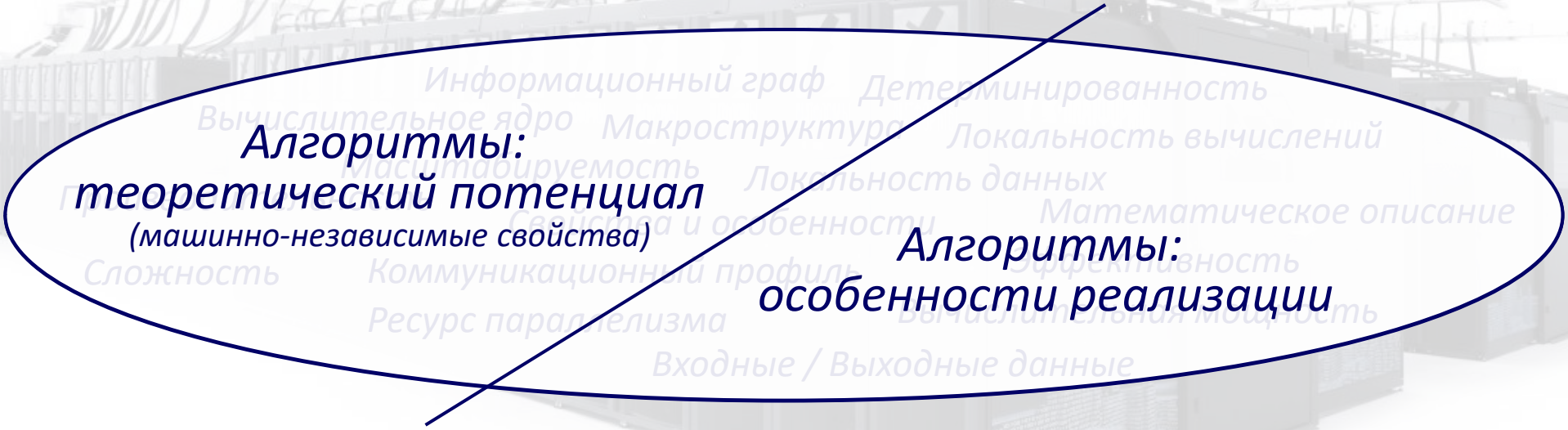
Да, можно: проект AlgoWiki

http://AlgoWiki-Project.org/

Можно ли выполнить такой анализ “раз и навсегда” ?

- Анализировать алгоритмы, чтобы понять, как их приспособить под новую компьютерную платформу ;

Структура и свойства алгоритмов
(от мобильных платформ до экзафлопсных суперкомпьютеров)



AlgoWiki

http://AlgoWiki-Project.org

Файл Правка Вид Журнал Закладки Инструменты Справка

Алговики

algowiki-project.org/ru/Открытая_энциклопедия_свойств_алгоритмов

Поиск

Войти Запрос учётной записи

Algowiki

Main page Обсуждение Читать Просмотр История Поиск

Открытая энциклопедия свойств алгоритмов

Заглавная страница
Общий форум
Технический форум
Справка
Свежие правки

Хранилище файлов
Новые файлы
Загрузить файл

Инструменты
Ссылки сюда
Связанные правки
Спецстраницы
Версия для печати
Постоянная ссылка
Сведения о странице

На других языках
English

Добро пожаловать! Присоединяйтесь!

AlgoWiki - это открытая энциклопедия по **свойствам алгоритмов и особенностям их реализации** на различных программно-аппаратных платформах от мобильных платформ до экзафлопсных суперкомпьютерных систем с возможностью коллективной работы всего мирового вычислительного сообщества.

Цель **AlgoWiki** - дать исчерпывающее описание алгоритма, которое поможет оценить его потенциал применительно к конкретной параллельной вычислительной платформе. Кроме классических свойств алгоритмов, например, последовательной сложности, в AlgoWiki представлены дополнительные сведения, составляющие в совокупности полную картину об алгоритме: параллельная сложность, параллельная структура, детерминированность, оценки локальности данных, эффективность и масштабируемость, коммуникационный профиль конкретных реализаций и многие другие.

Читать подробнее: О проекте

Структура проекта

Классификация алгоритмов - основной раздел AlgoWiki, содержащий описания всех алгоритмов. Алгоритмы добавляются в подходящий раздел классификации, при необходимости классификация расширяется за счет новых разделов.

Образцовая статья

Разложение Холецкого (метод квадратного корня)

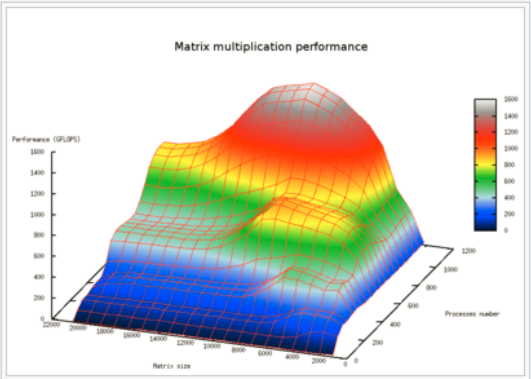
1 Свойства и структура алгоритма

1.1 Общее описание алгоритма

Свойства алгоритма:

- Последовательная сложность алгоритма: $O(n^3)$

Изображение дня



Производительность умножения плотных матриц

Организация работы

- Структура описания свойств алгоритмов
- Руководства по заполнению разделов описания
- Готовность статей
- Результаты прогона алгоритмов
- Глоссарий
- Помощь

Участники проекта

Описание алгоритмов в AlgoWiki
(предопределенная структура)

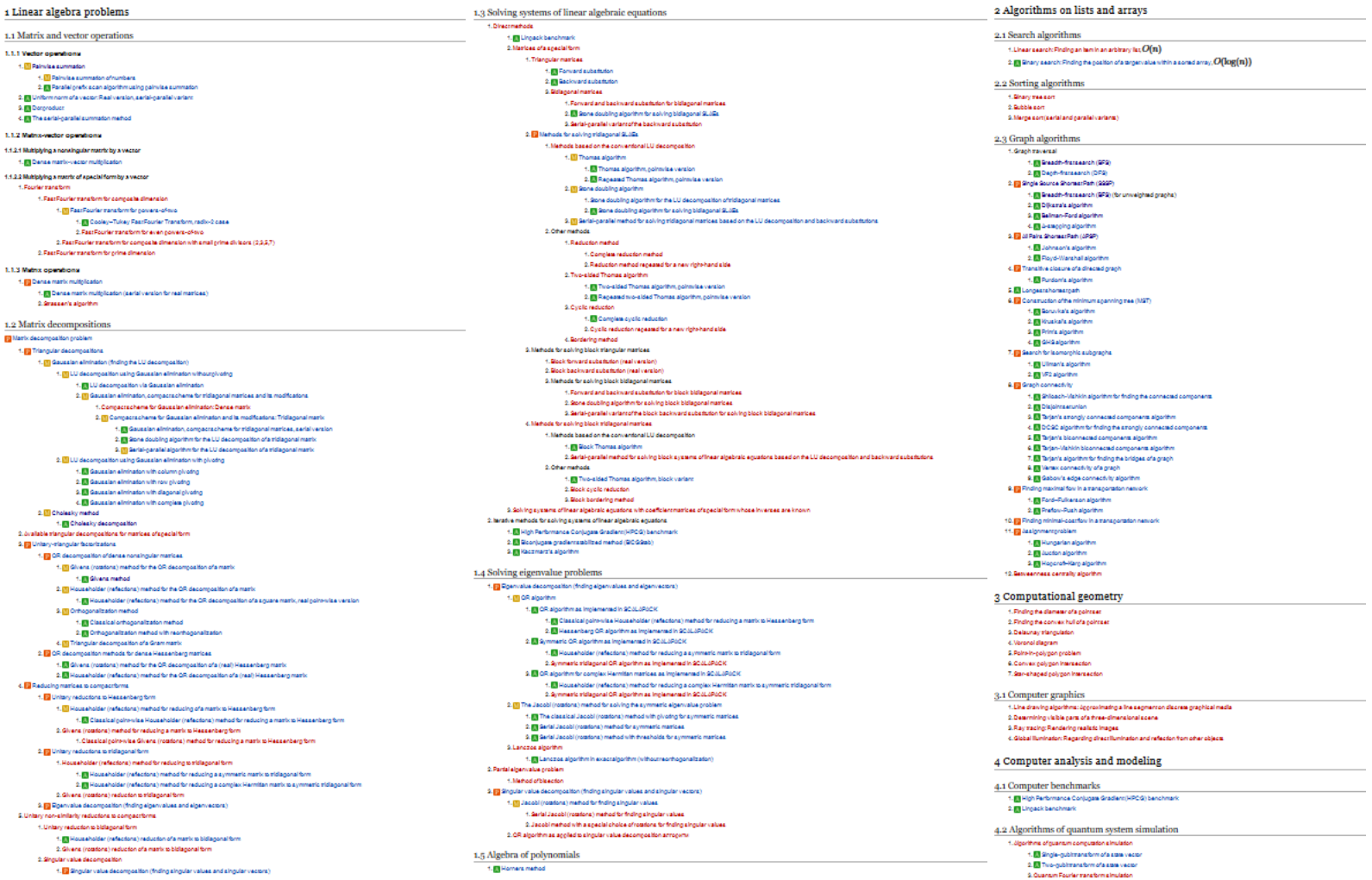
Содержание [убрать]

1 Свойства и структура алгоритмов
- 1.1 Общее описание алгоритма
- 1.2 Математическое описание алгоритма
- 1.3 Вычислительное ядро алгоритма
- 1.4 Макроструктура алгоритма
- 1.5 Схема реализации последовательного алгоритма
- 1.6 Последовательная сложность алгоритма
- 1.7 Информационный граф
- 1.8 Ресурс параллелизма алгоритма
- 1.9 Входные и выходные данные алгоритма
- 1.10 Свойства алгоритма

2 Программная реализация алгоритма
- 2.1 Особенности реализации последовательного алгоритма
- 2.2 Локальность данных и вычислений
- 2.3 Возможные способы и особенности параллельной реализации алгоритма
- 2.4 Масштабируемость алгоритма и его реализации
- 2.5 Динамические характеристики и эффективность реализации алгоритма
- 2.6 Выводы для классов архитектур
- 2.7 Существующие реализации алгоритма

3 Литература

AlgoWiki: классификация алгоритмов



1 Linear algebra problems

1.1 Matrix and vector operations

1.1.1 Vector operations
1. Pairwise summation
1. Pairwise summation of numbers
2. Parallel prefix scan algorithm using pairwise summation
2. Uniform norm of a vector: Real version, serial-parallel variant
3. Dot product
4. The serial-parallel summation method

1.1.2 Matrix-vector operations

1.1.2.1 Multiplying a nonsingular matrix by a vector
1. Dense matrix–vector multiplication

1.1.2.2 Multiplying a matrix of a special form by a vector
1. Fourier transform
1. FastFourier transform for composite dimension
1. FastFourier transform for powers-of-two
1. Cooley–Tukey FastFourier Transform, radix-2 case
2. FastFourier transform for even powers-of-two
2. FastFourier transform for composite dimension with small prime divisors (2,3,5,7)
2. FastFourier transform for prime dimension

1.1.3 Matrix operations
1. Dense matrix multiplication
1. Dense matrix multiplication (serial version for real matrices)
2. Strassen's algorithm

1.2 Matrix decompositions

Matrix decomposition problem
1. Triangular decompositions
1. Gaussian elimination (finding the LU decomposition)
1. LU decomposition using Gaussian elimination without pivoting
1. LU decomposition via Gaussian elimination
2. Gaussian elimination, compact scheme for tridiagonal matrices and its modifications
1. Compact scheme for Gaussian elimination: Dense matrix
2. Compact scheme for Gaussian elimination and its modifications: Tridiagonal matrix
1. Gaussian elimination, compact scheme for tridiagonal matrices, serial version
2. Bone doubling algorithm for the LU decomposition of a tridiagonal matrix
3. Serial-parallel algorithm for the LU decomposition of a tridiagonal matrix
2. LU decomposition using Gaussian elimination with pivoting
1. Gaussian elimination with column pivoting
2. Gaussian elimination with row pivoting
3. Gaussian elimination with diagonal pivoting
4. Gaussian elimination with complete pivoting
2. Cholesky method
1. Cholesky decomposition
2. Available triangular decompositions for matrices of special form
3. Unitary-triangular factorizations
1. QR decomposition of dense nonsingular matrices
1. Givens (rotations) method for the QR decomposition of a matrix
1. Givens method
2. Householder (reflections) method for the QR decomposition of a matrix
1. Householder (reflections) method for the QR decomposition of a square matrix, real pairwise version
3. Orthogonalization method
1. Classical orthogonalization method
2. Orthogonalization method with reorthogonalization
4. Triangular decomposition of a Gram matrix
2. QR decomposition methods for dense Hessenberg matrices
1. Givens (rotations) method for the QR decomposition of a (real) Hessenberg matrix
2. Householder (reflections) method for the QR decomposition of a (real) Hessenberg matrix
4. Reducing matrices to compact forms
1. Unitary reductions to Hessenberg form
1. Householder (reflections) method for reducing of a matrix to Hessenberg form
1. Classical pairwise Householder (reflections) method for reducing a matrix to Hessenberg form
2. Givens (rotations) method for reducing a matrix to Hessenberg form
1. Classical pairwise Givens (rotations) method for reducing a matrix to Hessenberg form
2. Unitary reductions to tridiagonal form
1. Householder (reflections) method for reducing to tridiagonal form
1. Householder (reflections) method for reducing a symmetric matrix to tridiagonal form
2. Householder (reflections) method for reducing a complex Hermitian matrix to symmetric tridiagonal form
2. Givens (rotations) reduction to tridiagonal form
3. Eigenvalue decomposition (finding eigenvalues and eigenvectors)
5. Unitary non-similarity reductions to compact forms
1. Unitary reduction to bidiagonal form
1. Householder (reflections) reduction of a matrix to bidiagonal form
2. Givens (rotations) reduction of a matrix to bidiagonal form
2. Singular value decomposition
1. Singular value decomposition (finding singular values and singular vectors)

1.3 Solving systems of linear algebraic equations
1. Direct methods
1. Lingack benchmark
2. Matrices of a special form
1. Triangular matrices
1. Forward substitution
2. Backward substitution
3. Bidiagonal matrices
1. Forward and backward substitution for bidiagonal matrices
2. Bone doubling algorithm for solving bidiagonal SLAEs
3. Serial-parallel variants of the backward substitution
2. Methods for solving tridiagonal SLAEs
1. Methods based on the conventional LU decomposition
1. Thomas algorithm
1. Thomas algorithm, pairwise version
2. Repeated Thomas algorithm, pairwise version
2. Bone doubling algorithm
1. Bone doubling algorithm for the LU decomposition of tridiagonal matrices
2. Bone doubling algorithm for solving bidiagonal SLAEs
3. Serial-parallel method for solving tridiagonal matrices based on the LU decomposition and backward substitutions
2. Other methods
1. Reduction method
1. Complex reduction method
2. Reduction method repeated for a new right-hand side
2. Two-sided Thomas algorithm
1. Two-sided Thomas algorithm, pairwise version
2. Repeated two-sided Thomas algorithm, pairwise version
3. Cyclic reduction
1. Complex cyclic reduction
2. Cyclic reduction repeated for a new right-hand side
4. Bordering method
3. Methods for solving block triangular matrices
1. Block forward substitution (real version)
2. Block backward substitution (real version)
3. Methods for solving block bidiagonal matrices
1. Forward and backward substitution for block bidiagonal matrices
2. Bone doubling algorithm for solving block bidiagonal matrices
3. Serial-parallel variants of the block backward substitution for solving block bidiagonal matrices
4. Methods for solving block tridiagonal matrices
1. Methods based on the conventional LU decomposition
1. Block Thomas algorithm
2. Serial-parallel method for solving block systems of linear algebraic equations based on the LU decomposition and backward substitutions
2. Other methods
1. Two-sided Thomas algorithm, block variant
2. Block cyclic reduction
3. Block bordering method
3. Solving systems of linear algebraic equations with coefficient matrices of special form whose inverses are known
2. Iterative methods for solving systems of linear algebraic equations
1. High Performance Conjugate Gradient (HPCG) benchmark
2. Biconjugate gradient-stabilized method (BICGStab)
3. Kaczmarz's algorithm

1.4 Solving eigenvalue problems
1. Eigenvalue decomposition (finding eigenvalues and eigenvectors)
1. QR algorithm
1. QR algorithm as implemented in SCALAPACK
1. Classical pairwise Householder (reflections) method for reducing a matrix to Hessenberg form
2. Hessenberg QR algorithm as implemented in SCALAPACK
2. Symmetric QR algorithm as implemented in SCALAPACK
1. Householder (reflections) method for reducing a symmetric matrix to tridiagonal form
2. Symmetric tridiagonal QR algorithm as implemented in SCALAPACK
3. QR algorithm for complex Hermitian matrices as implemented in SCALAPACK
1. Householder (reflections) method for reducing a complex Hermitian matrix to symmetric tridiagonal form
2. Symmetric tridiagonal QR algorithm as implemented in SCALAPACK
2. The Jacobi (rotations) method for solving the symmetric eigenvalue problem
1. The classical Jacobi (rotations) method with pivoting for symmetric matrices
2. Serial Jacobi (rotations) method for symmetric matrices
3. Serial Jacobi (rotations) method with thresholds for symmetric matrices
3. Lanczos algorithm
1. Lanczos algorithm in exact arithmetic (without reorthogonalization)
2. Partial eigenvalue problem
1. Method of bisection
3. Singular value decomposition (finding singular values and singular vectors)
1. Jacobi (rotations) method for finding singular values
1. Serial Jacobi (rotations) method for finding singular values
2. Jacobi method with a special choice of rotations for finding singular values
2. QR algorithm as applied to singular value decomposition arrangums

1.5 Algebra of polynomials
1. Horners method

2 Algorithms on lists and arrays

2.1 Search algorithms
1. Linear search: Finding an item in an arbitrary list, $O(n)$
2. Binary search: Finding the position of a target value within a sorted array, $O(\log(n))$

2.2 Sorting algorithms
1. Binary tree sort
2. Bubble sort
3. Merge sort (serial and parallel variants)

2.3 Graph algorithms
1. Graph traversal
1. Breadth-first search (BFS)
2. Depth-first search (DFS)
2. Single Source Shortest Path (SSSP)
1. Breadth-first search (BFS) (for unweighted graphs)
2. Dijkstra's algorithm
3. Bellman-Ford algorithm
4. Δ-stepping algorithm
3. All Pairs Shortest Path (APSP)
1. Johnson's algorithm
2. Floyd-Warshall algorithm
4. Transitive closure of a directed graph
1. Purdom's algorithm
5. Longest path problem
6. Construction of the minimum spanning tree (MST)
1. Boruvka's algorithm
2. Kruskal's algorithm
3. Prim's algorithm
4. GHS algorithm
7. Search for isomorphic subgraphs
1. Ullman's algorithm
2. VF2 algorithm
8. Graph connectivity
1. Shiloach–Vishkin algorithm for finding the connected components
2. Dinic's algorithm
3. Tarjan's strongly connected components algorithm
4. DCSC algorithm for finding the strongly connected components
5. Tarjan's biconnected components algorithm
6. Tarjan–Vishkin biconnected components algorithm
7. Tarjan's algorithm for finding the bridges of a graph
8. Vertex connectivity of a graph
9. Gabow's edge connectivity algorithm
9. Finding maximal flow in a transportation network
1. Ford–Fulkerson algorithm
2. Preflow–Push algorithm
10. Finding minimal-cost flow in a transportation network
11. Assignment problem
1. Hungarian algorithm
2. Auction algorithm
3. Hopcroft–Karp algorithm
12. Betweenness centrality algorithm

3 Computational geometry
1. Finding the diameter of a point set
2. Finding the convex hull of a point set
3. Delaunay triangulation
4. Voronoi diagram
5. Point-in-polygon problem
6. Convex polygon intersection
7. Star-shaped polygon intersection

3.1 Computer graphics
1. Line drawing algorithms: approximating a line segment on discrete graphical media
2. Determining visible parts of a three-dimensional scene
3. Ray tracing: Rendering realistic images
4. Global illumination: Regarding direct illumination and reflection from other objects

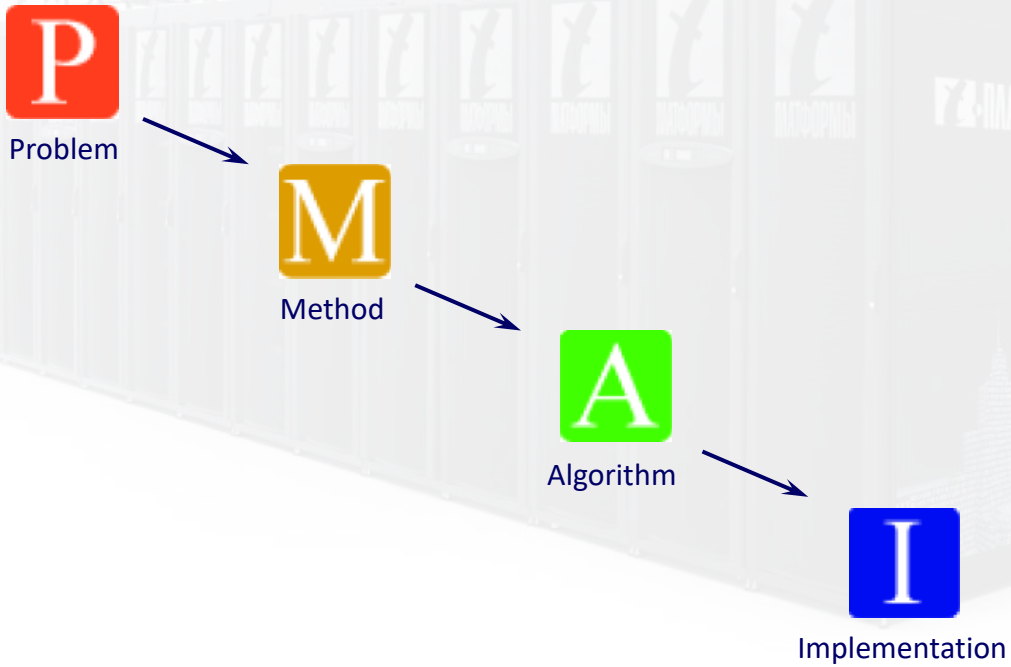
4 Computer analysis and modeling

4.1 Computer benchmarks
1. High Performance Conjugate Gradient (HPCG) benchmark
2. Linpack benchmark

4.2 Algorithms of quantum system simulation
1. Algorithms of quantum computers simulation
1. Single-qubit transform of a state vector
2. Two-qubit transform of a state vector
3. Quantum Fourier transform simulation

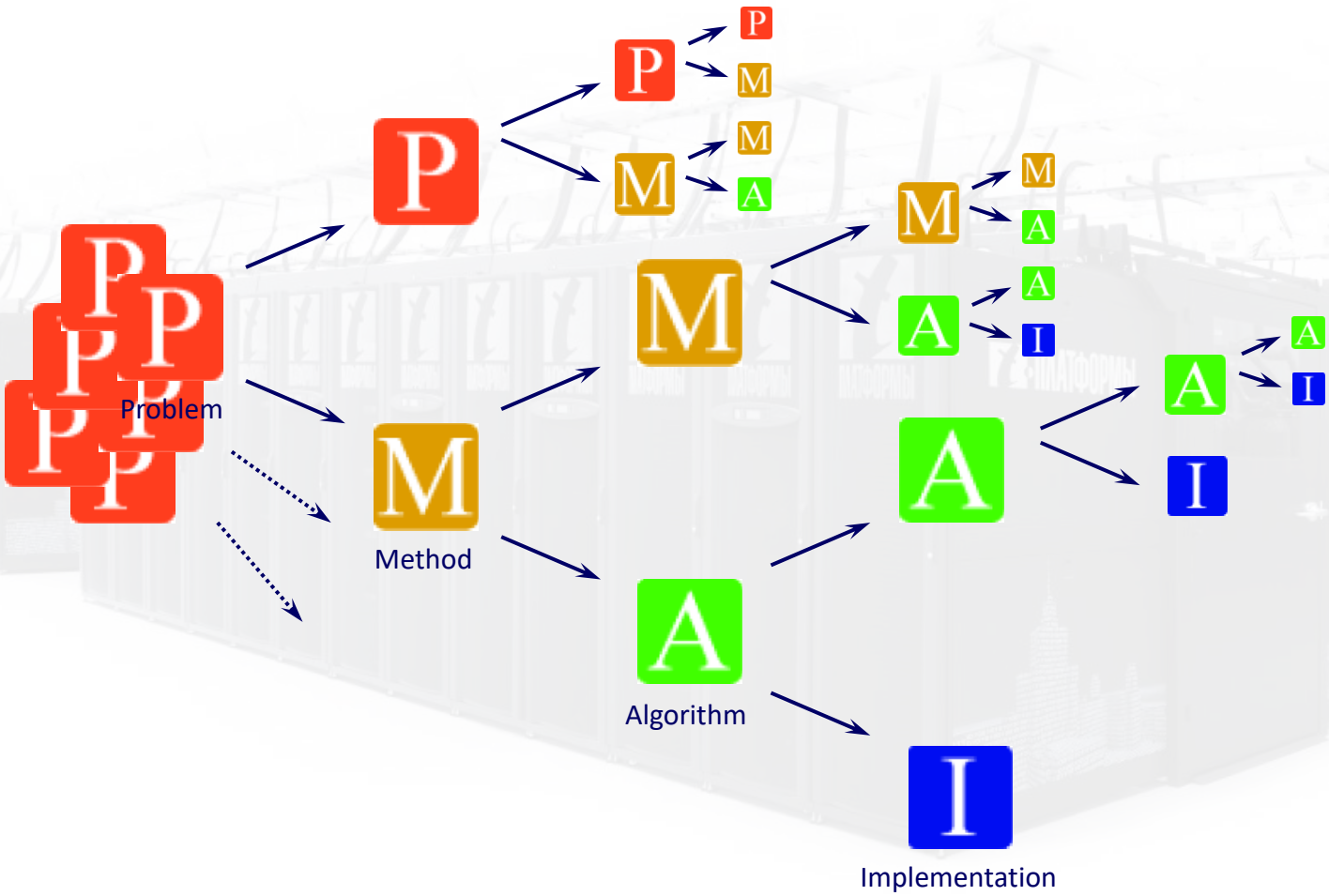
AlgoWiki: Problem – Method – Algorithm – Implementation

(по каждому алгоритму в AlgoWiki есть полное описание цепочки)



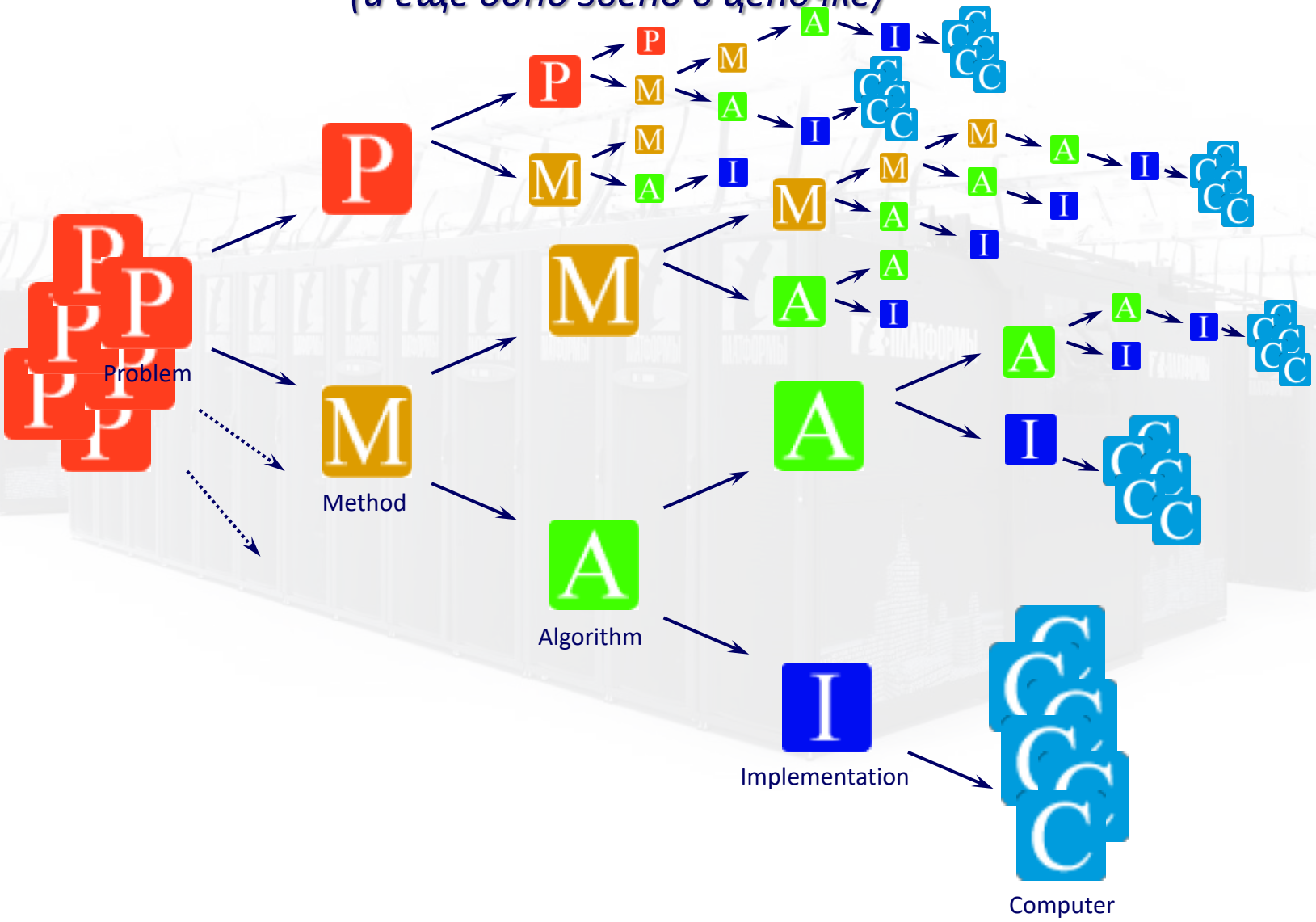
AlgoWiki: Problem – Method – Algorithm – Implementation

(на практике все разнообразнее и сложнее)



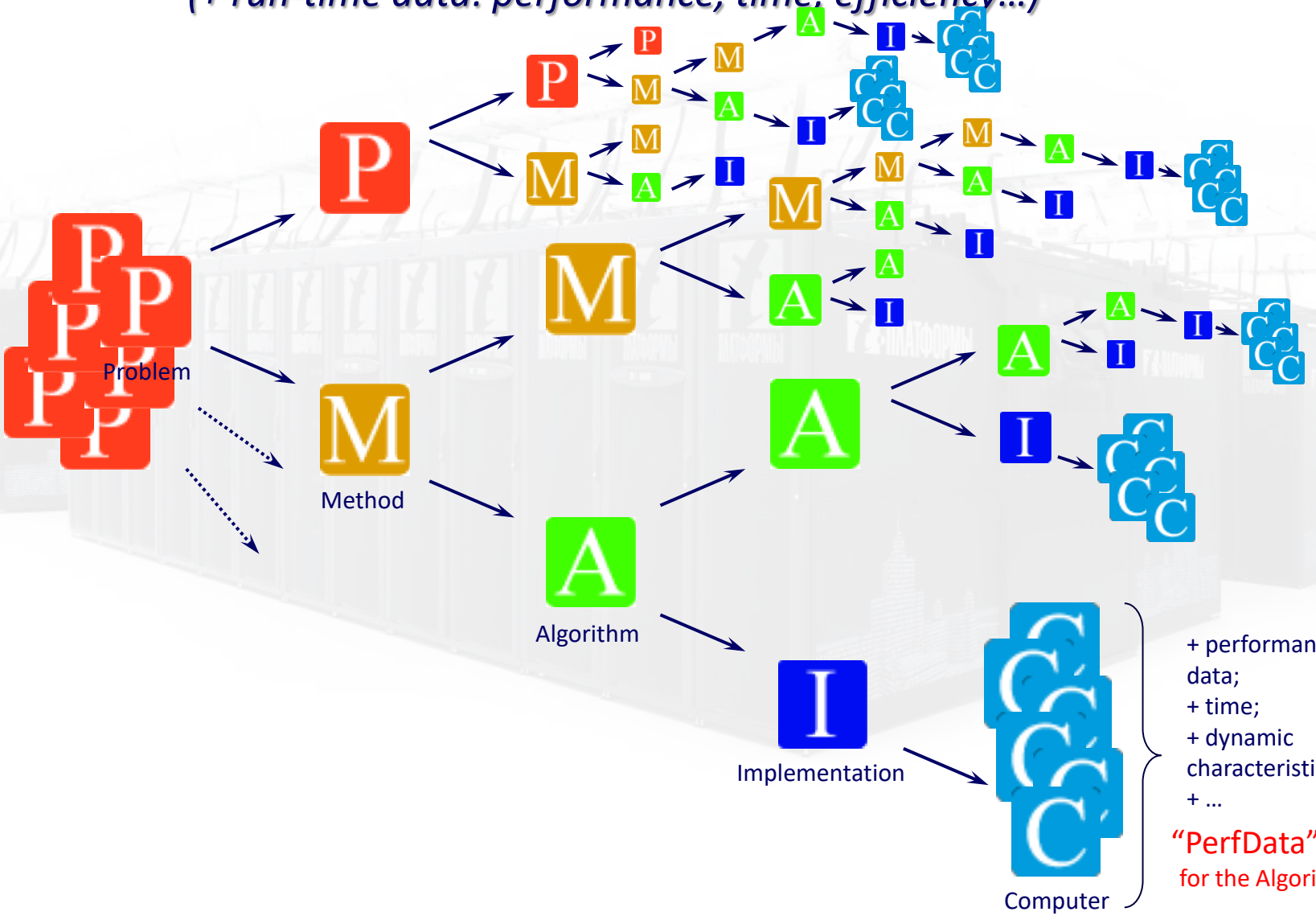
AlgoWiki: from Problems to various Implementations

(и еще одно звено в цепочке)



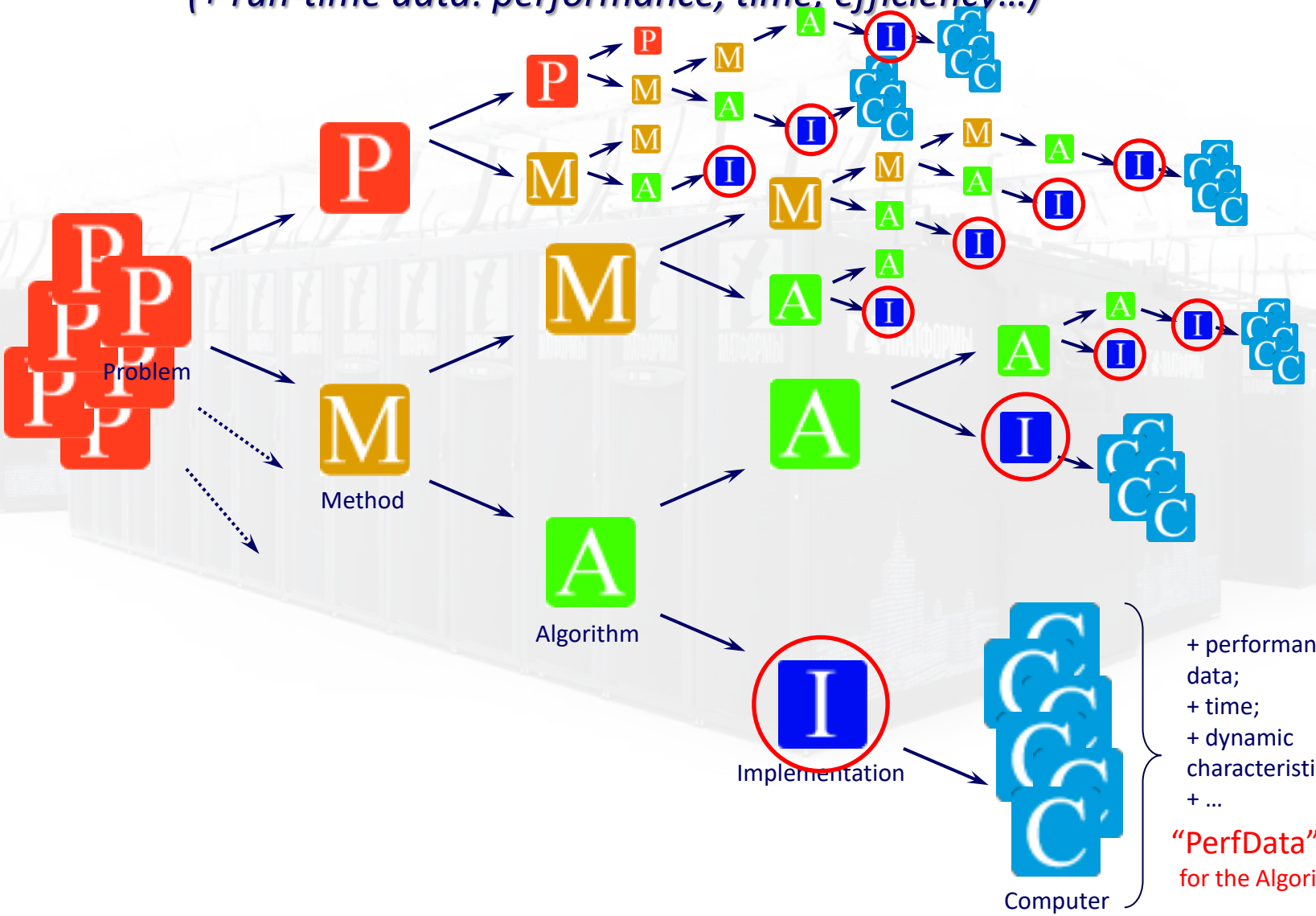
AlgoWiki: from Problems to various Implementations

(+ run-time data: performance, time, efficiency…)



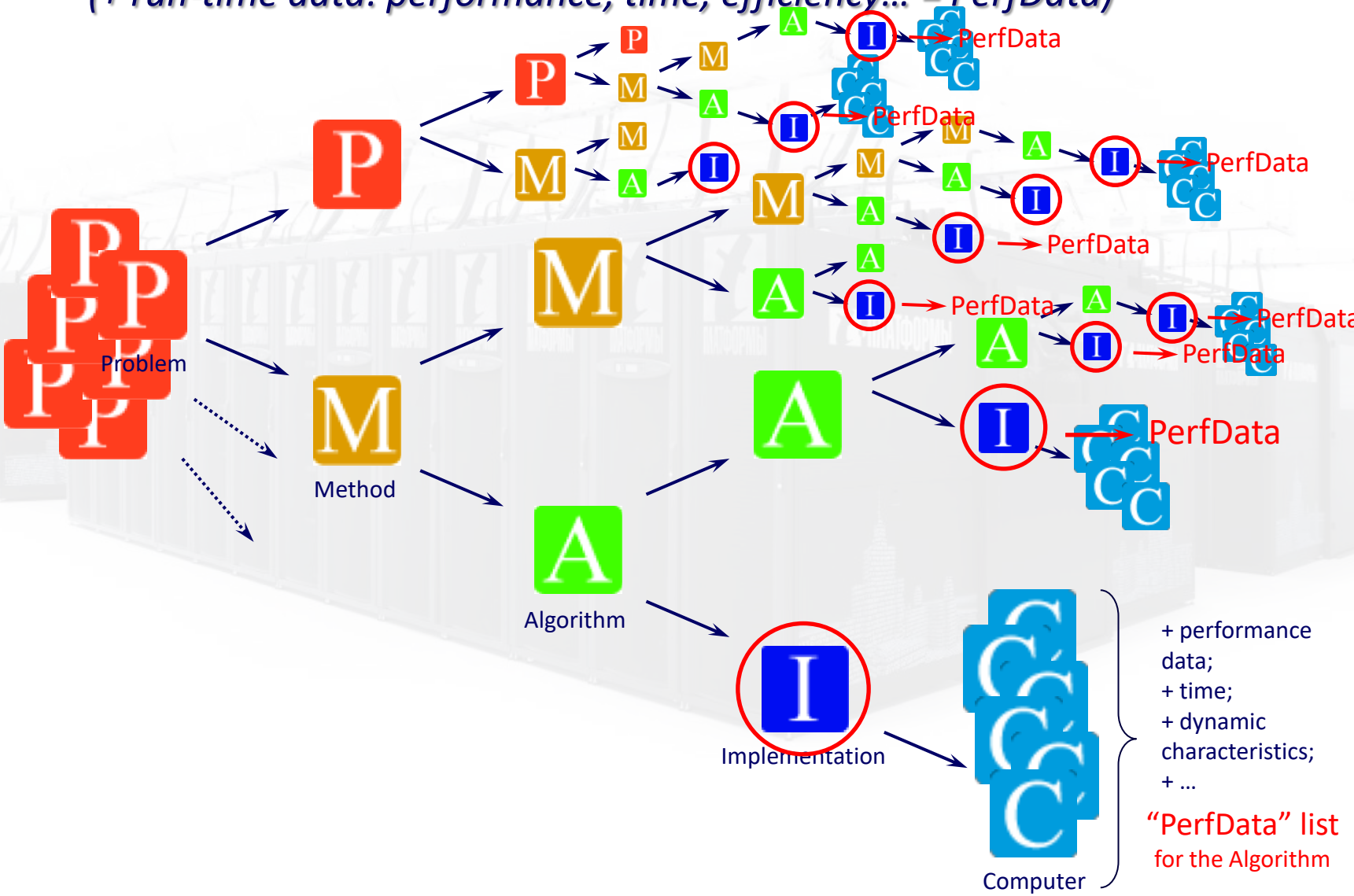
AlgoWiki: from Problems to various Implementations

(+ run-time data: performance, time, efficiency…)



AlgoWiki: from Problems to various Implementations

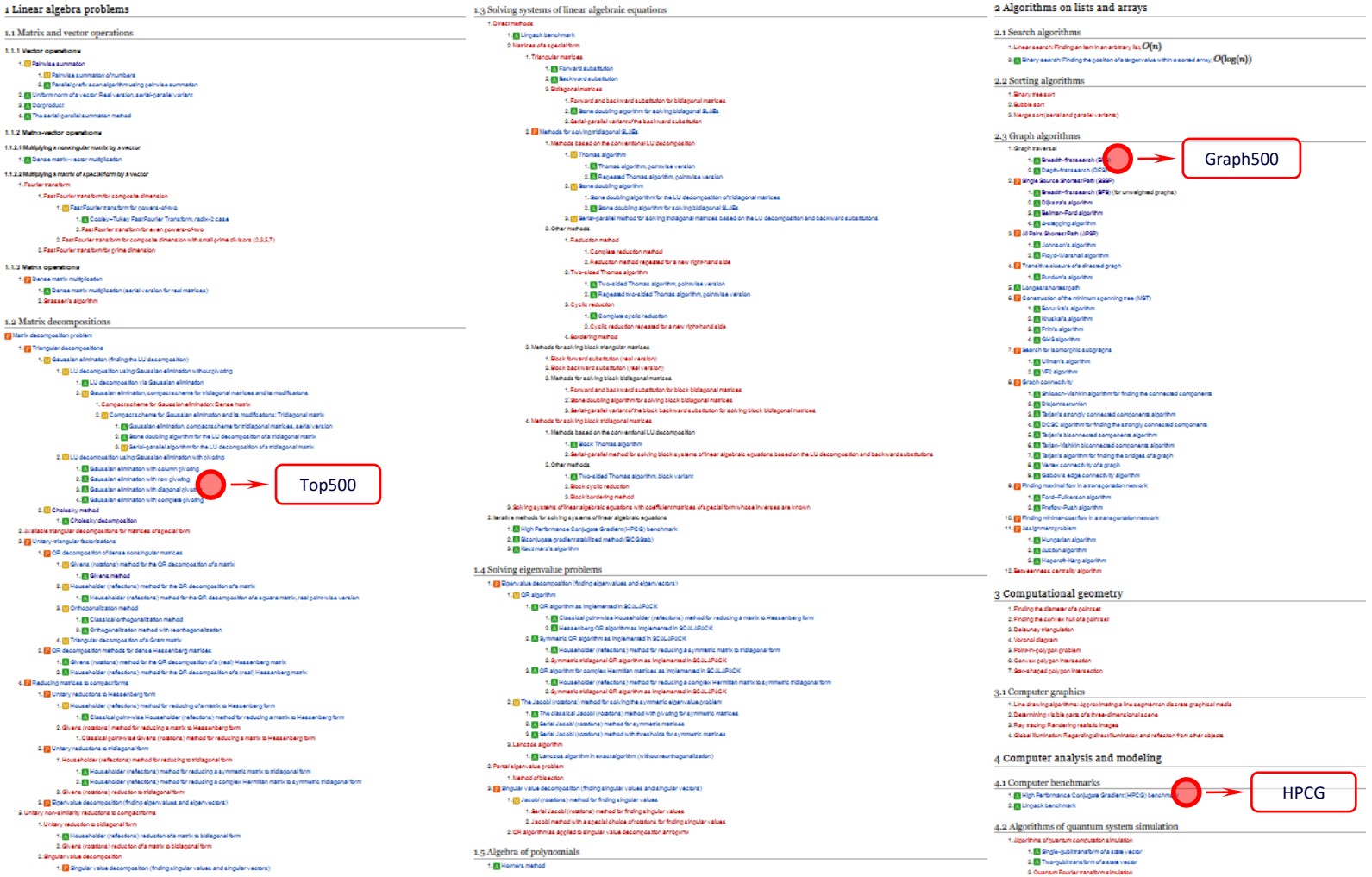
(+ run-time data: performance, time, efficiency… = PerfData)



Классификация алгоритмов и сравнение компьютеров

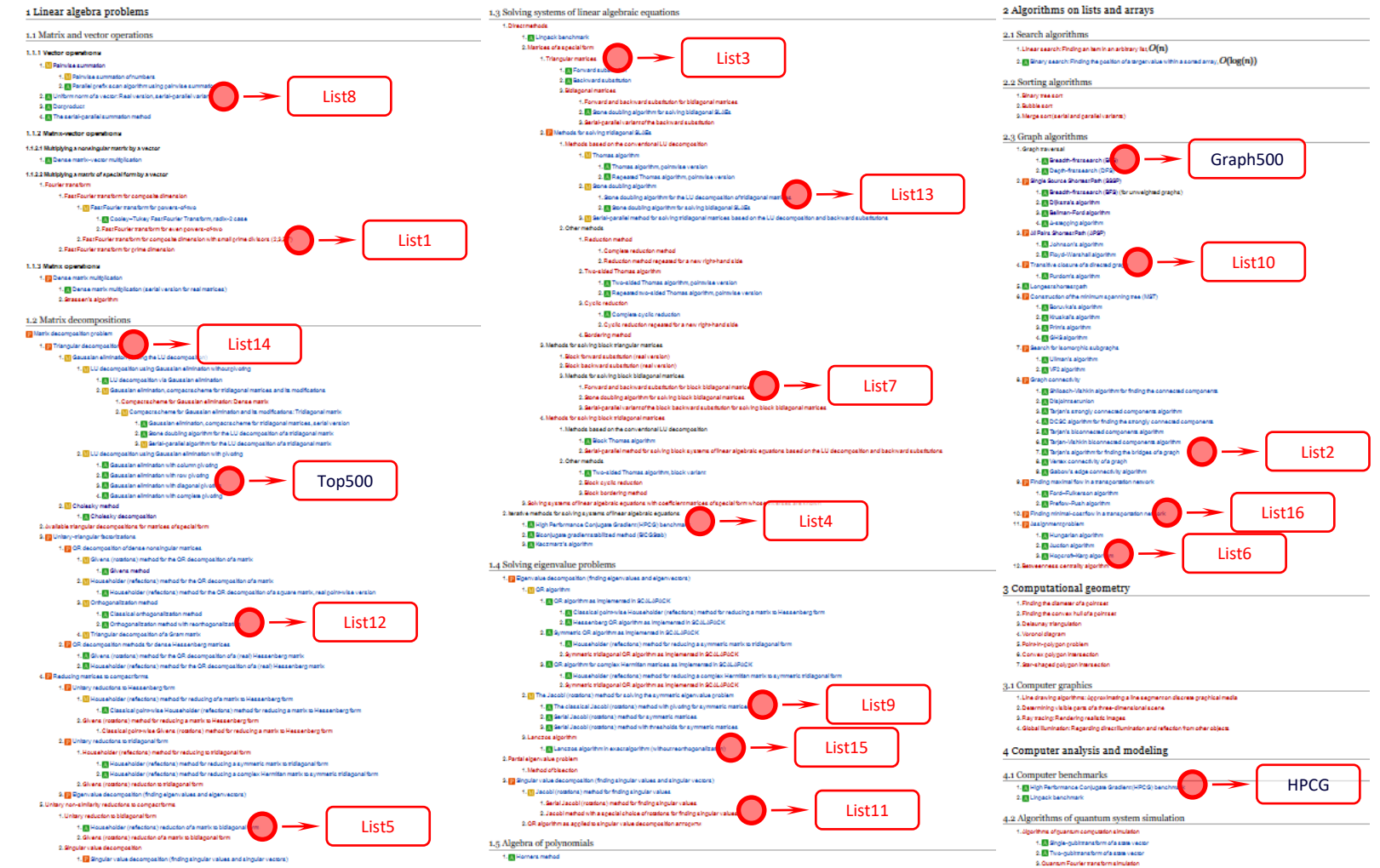
(Какие возможность есть сейчас? Три точки на всем множестве алгоритмов)

Algorithm classification



Классификация алгоритмов и сравнение компьютеров

(Что может добавить AlgoWiki к набору из трёх тестов?)



Зачетные задания по курсу “Суперкомпьютерное моделирование и технологии”

- *Все варианты заданий будут объявлены на http://Parallel.ru/SS20*

Московский государственный университет имени М.В.Ломоносова
Факультет Вычислительной математики и кибернетики



СУПЕРКОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ И ТЕХНОЛОГИИ

СТРУКТУРА АЛГОРИТМОВ

Вл.В.Воеводин

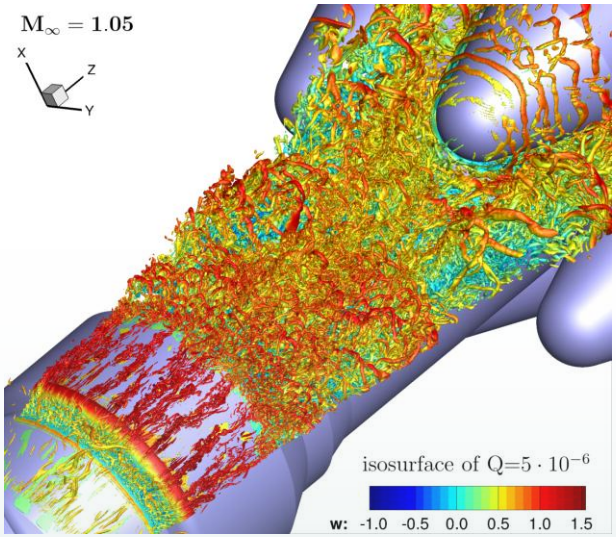
Зав.кафедрой Суперкомпьютеров и квантовой информатики ВМК МГУ,
Директор НИВЦ МГУ,
чл.-корр. РАН, д.ф.-м.н., профессор

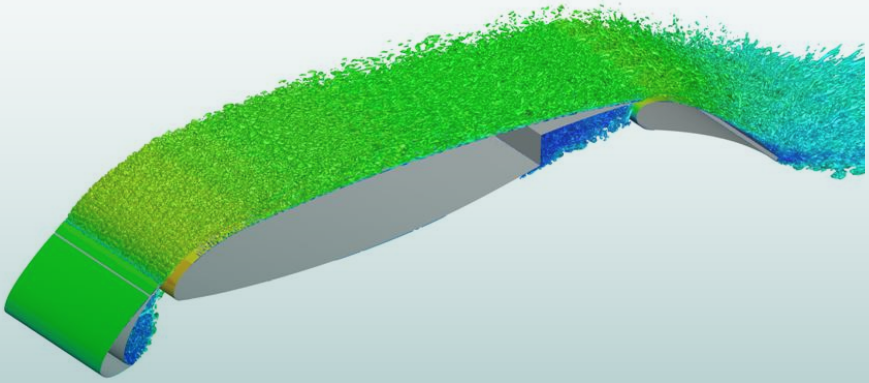
voevodin@parallel.ru

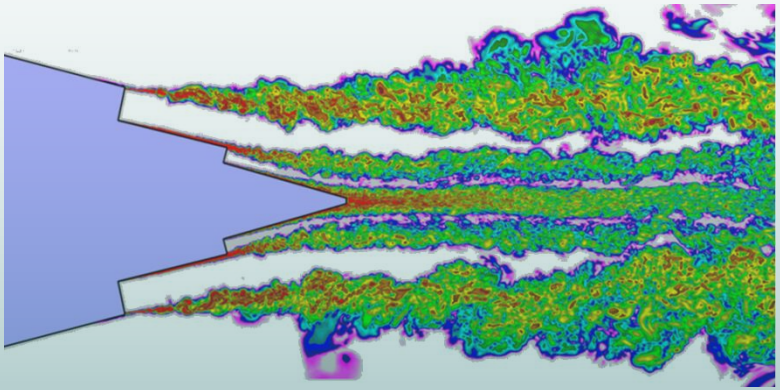
ВМК МГУ, 2020

Суперкомпьютерное моделирование турбулентности: авиационно-космические приложения









Андрей Горобец
ИПМ им. М. В. Келдыша РАН, сектор вычислительной аэродинамики и аэроакустики
http://caa.imamod.ru



Предсказательное суперкомпьютерное моделирование

Ну что сказать, ну что сказать,
Устроены так люди,
Желают знать, желают знать,
Желают знать что будет.

А улучшится ли прочность, если в автомобиле взять эту железяку выпилить, а сюда две приварить?

А если у самолета крыло вот так отогнуть и вот так немного повернуть, станет он лучше лететь?

А хватит ли выбега ротора турбогенератора для питания ГЦН?

…

- **Исследовать явление, узнать, да что там вообще происходит?**
- **Многовариантные расчеты – узнать, какой вариант лучше?**
- **Сертификационные испытания**

Предсказательное суперкомпьютерное моделирование

- **Лучше, если можно предсказать вообще без вычислений**
- **Если нет, то лучше, если можно посчитать на калькуляторе**
- **Если нет, то лучше, если можно посчитать десктопе**
- **Если нет, то лучше, если можно посчитать на серваке**
- **Если нет, то лучше, если можно посчитать на маленьком кластере**

…

и если уже никак по-другому не получается, то только

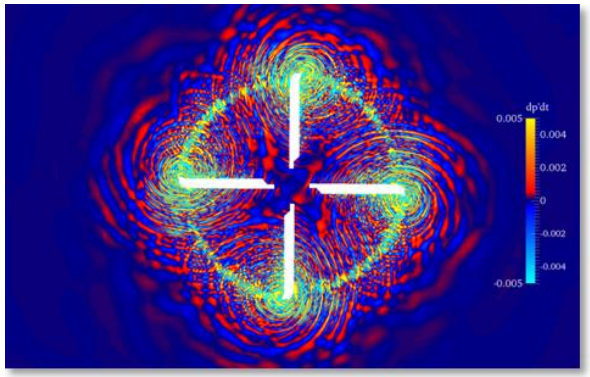
от отчаяния и безысходности

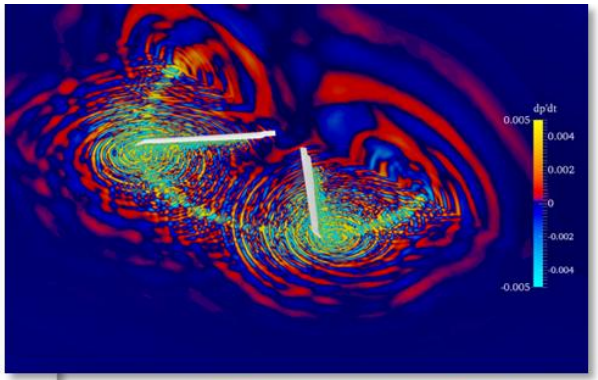
- **мы станем считать на суперкомпьютере**

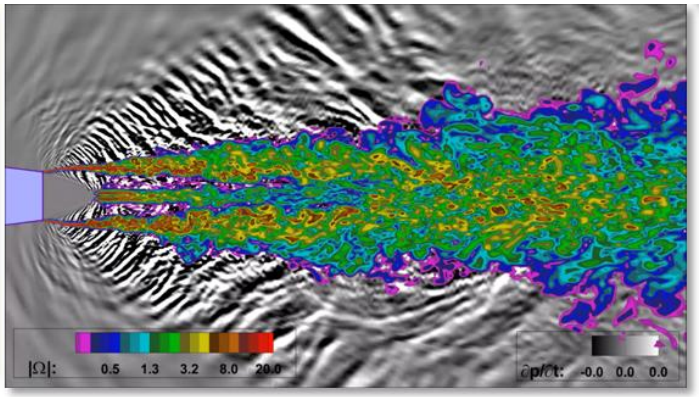
Посчитать должно быть либо выгоднее, чем натурный эксперимент

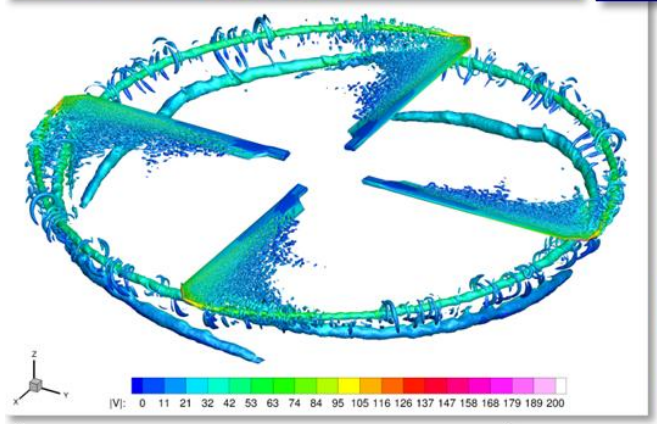
Ну или если натурный эксперимент невозможен

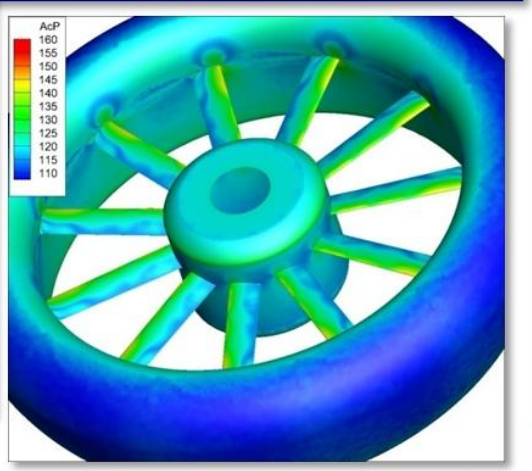
CFD и CAA приложения



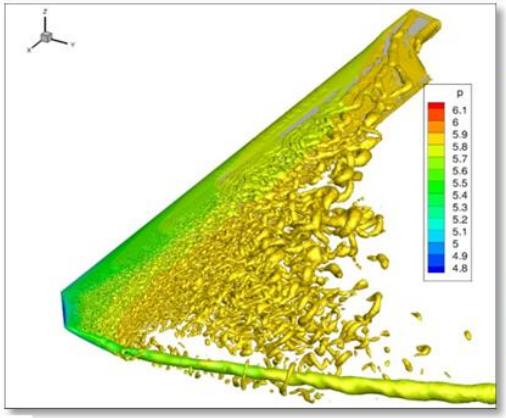


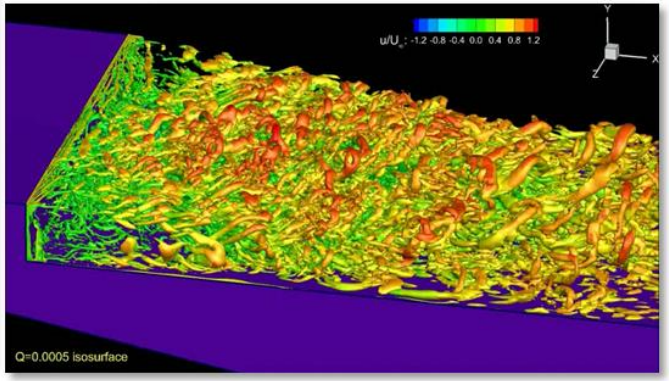


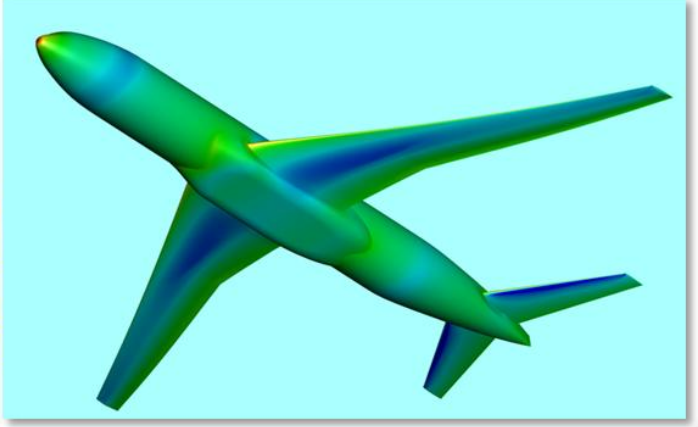












Суперкомпьютерное моделирование и летательные аппараты

авиалайнеры

мультикоптеры

ЦМУ Сверхзвук

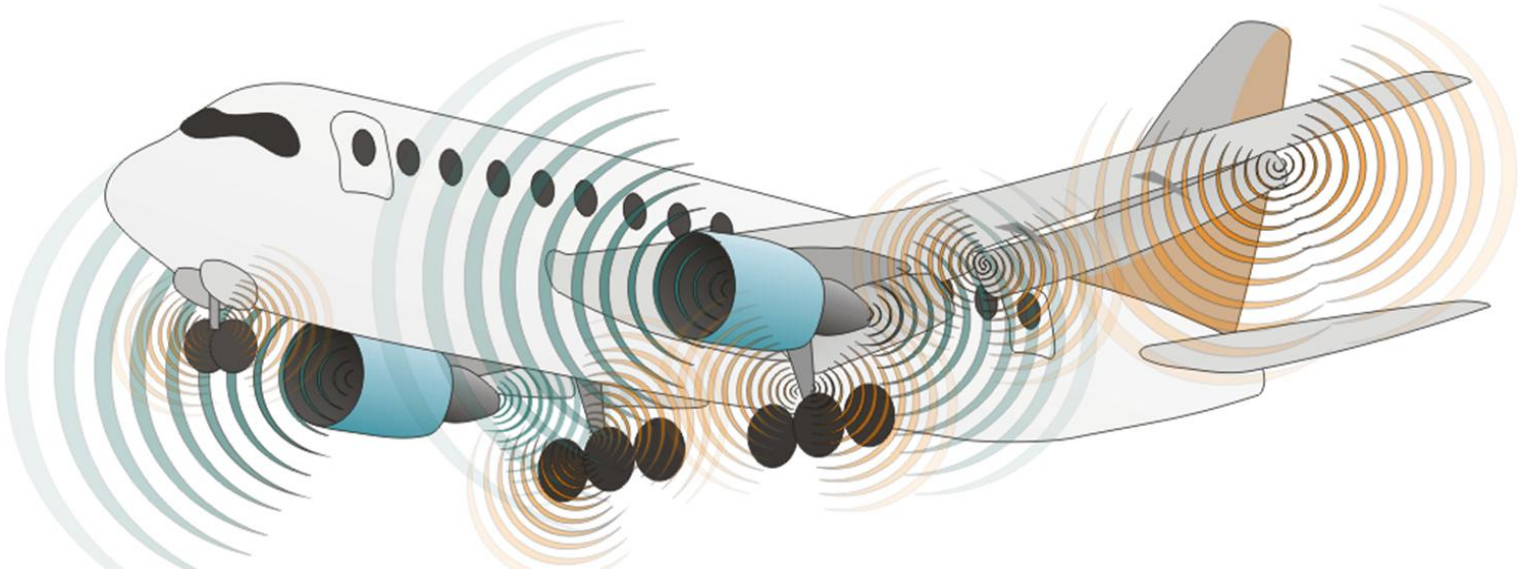
5-поколение

сверхзвуковые ПС

ПТВ

ПСВ

ШФДМС



аэродинамика

прочность

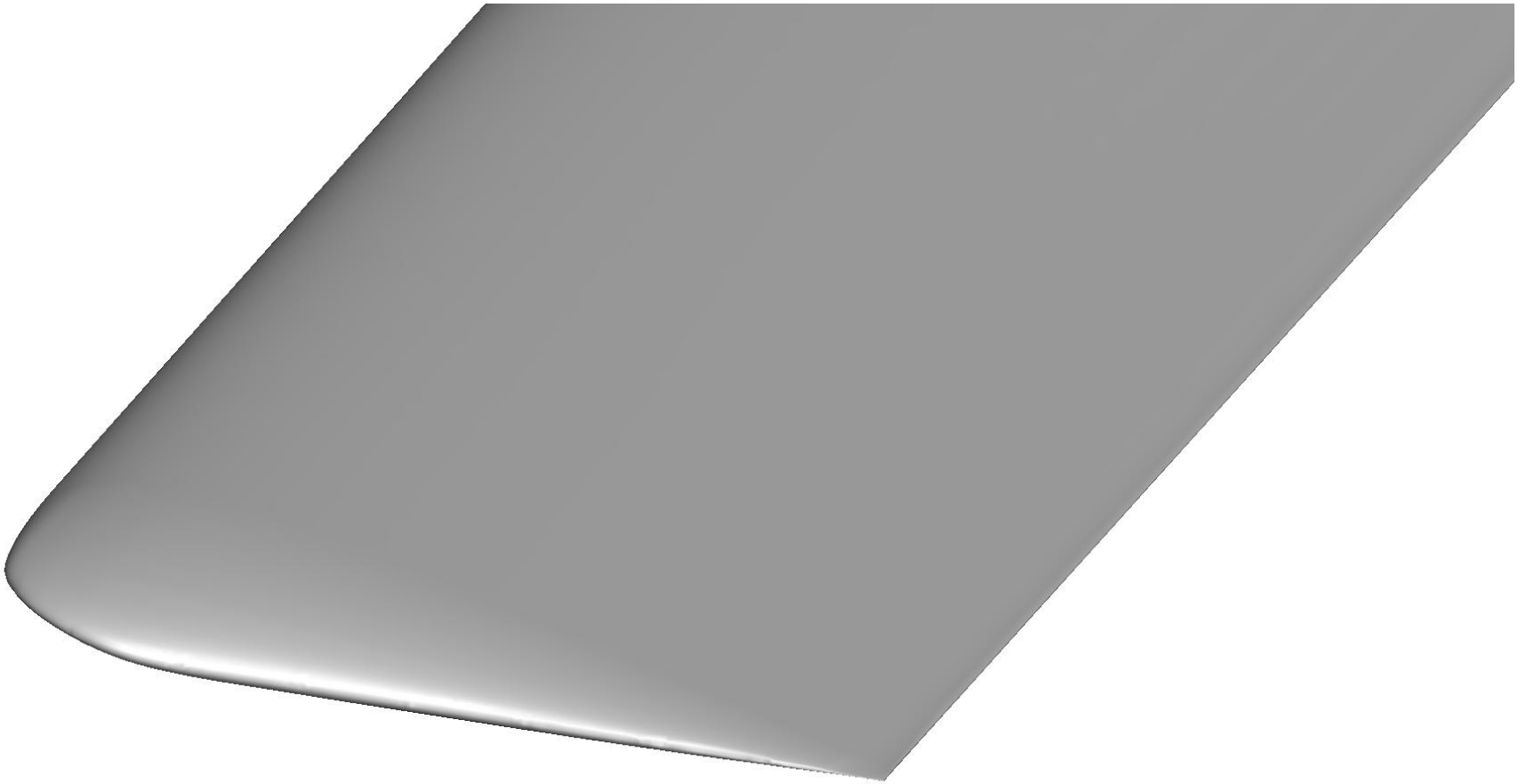
горение

вентиляция

аэроакустика

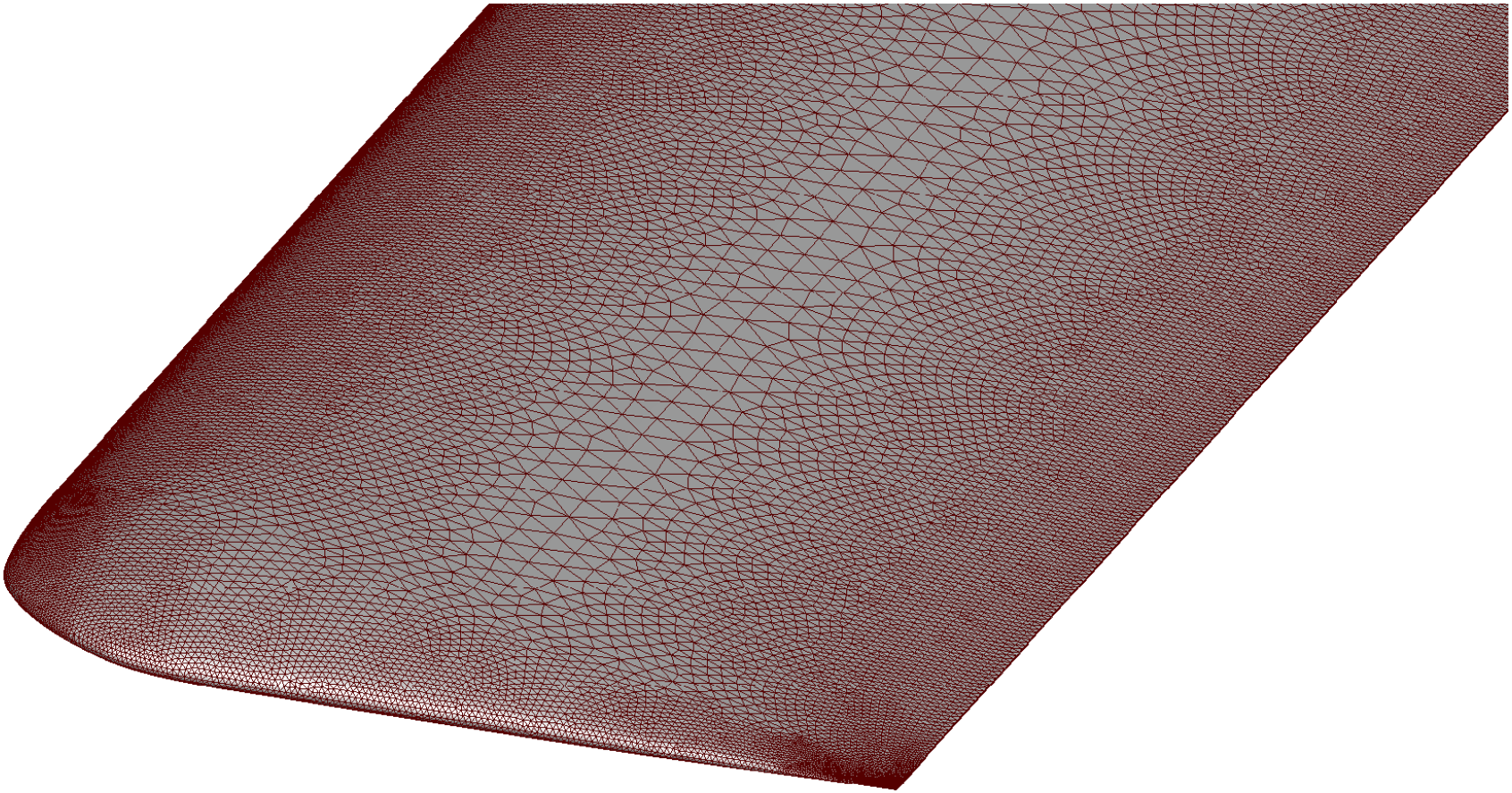
теплообмен

Как это работает…



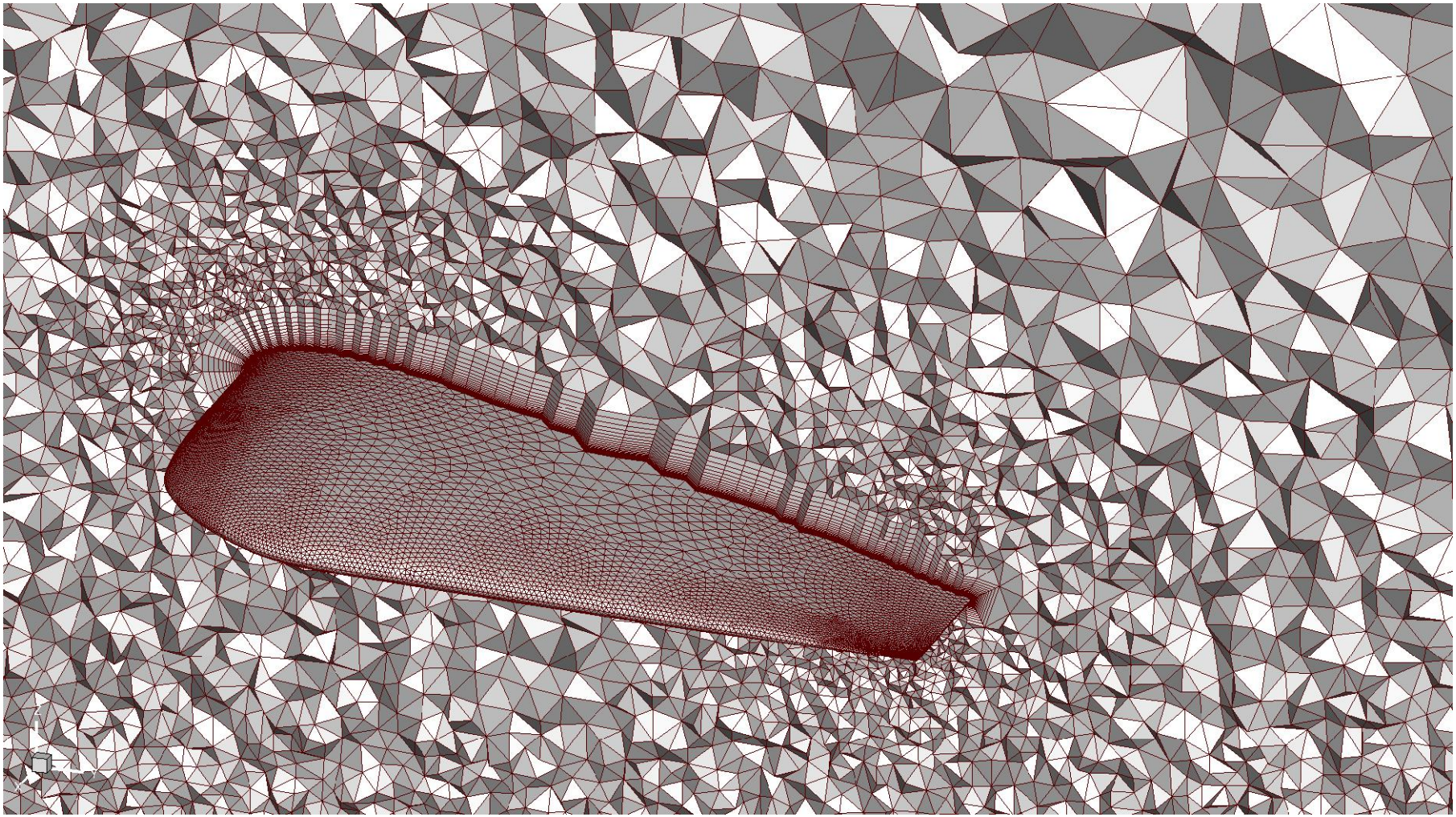
Берем CAD модель объекта

Как это работает…



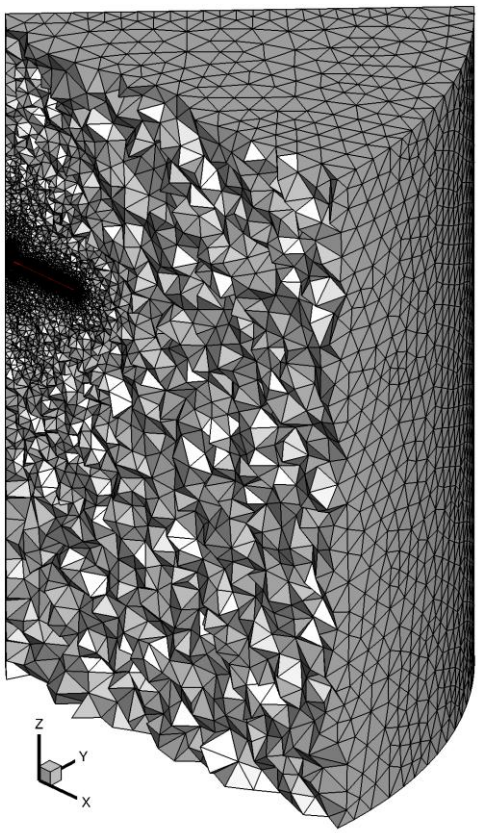
Строим сеточную модель

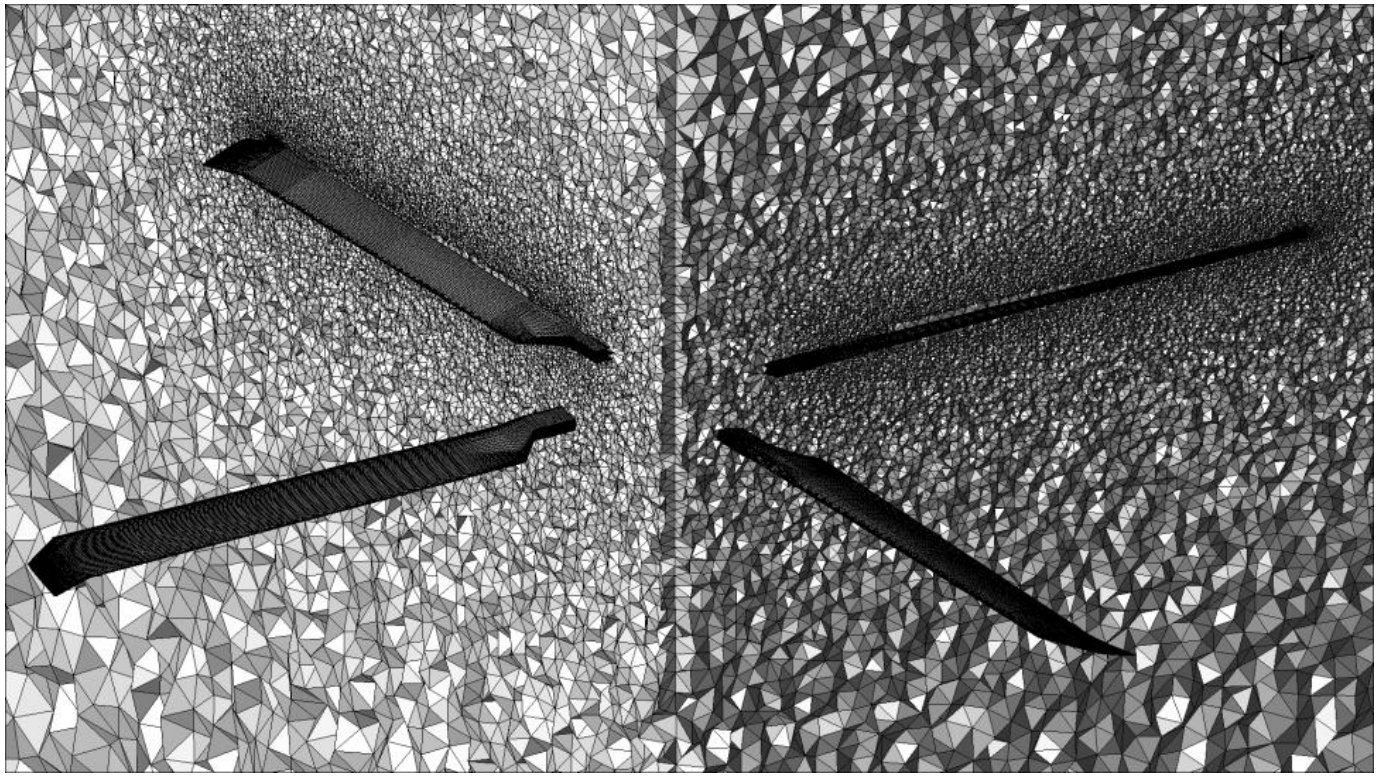
Как это работает…



Строим пространственную сетку в расчетной области

Как это работает…



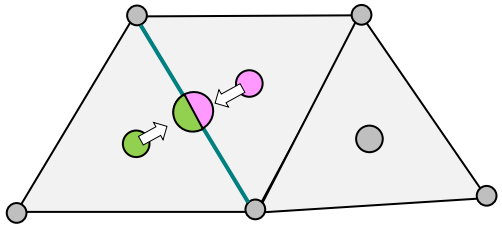


В ячейках сетки задаем физические переменные
(например, плотность, давление, компоненты вектора скорости)

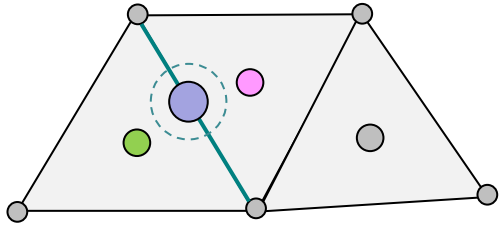
Как это работает…

Массивно-параллельная обработка сеточных данных
(узлы, грани, сеточные элементы, …)

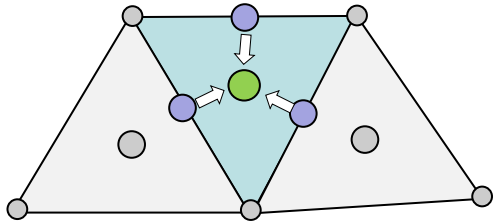
Реконструкция по значениям в ячейках значений на гранях



Расчет потока через грани
решение задачи Римана о распаде разрыва

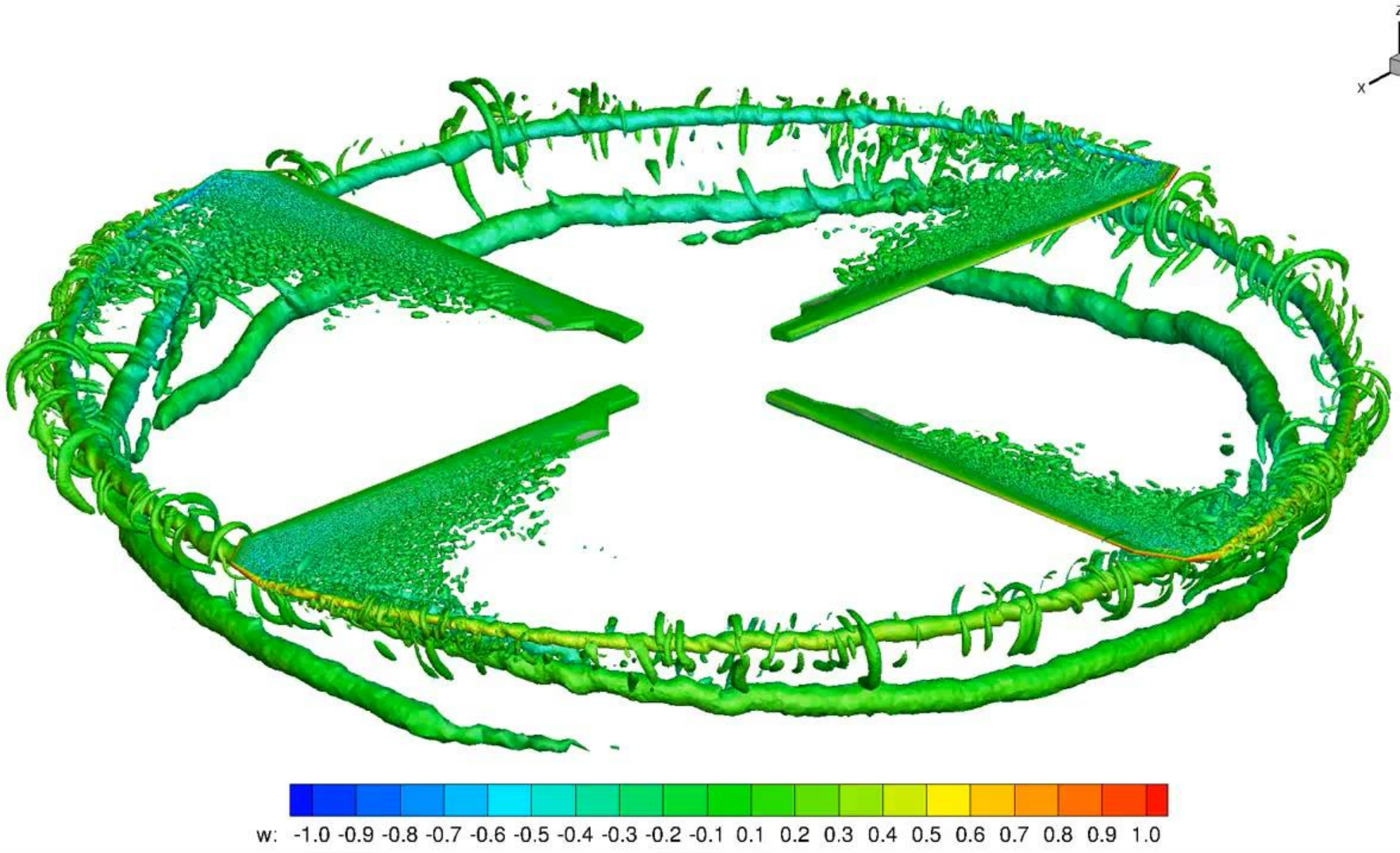


Переход на новый временной слой
суммирование потоков через грани в ячейки



Вычислительная газовая динамика – CFD

Как это работает…

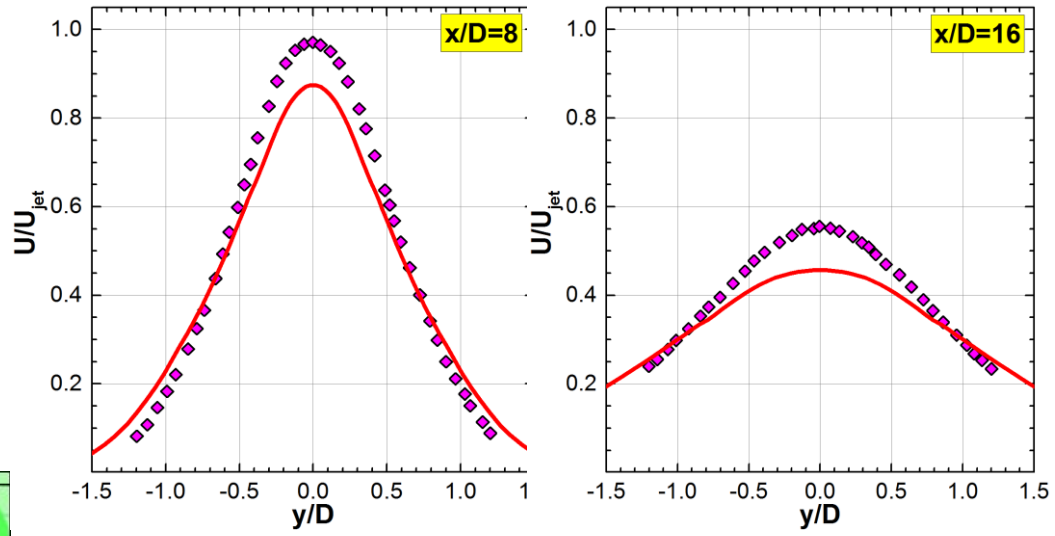


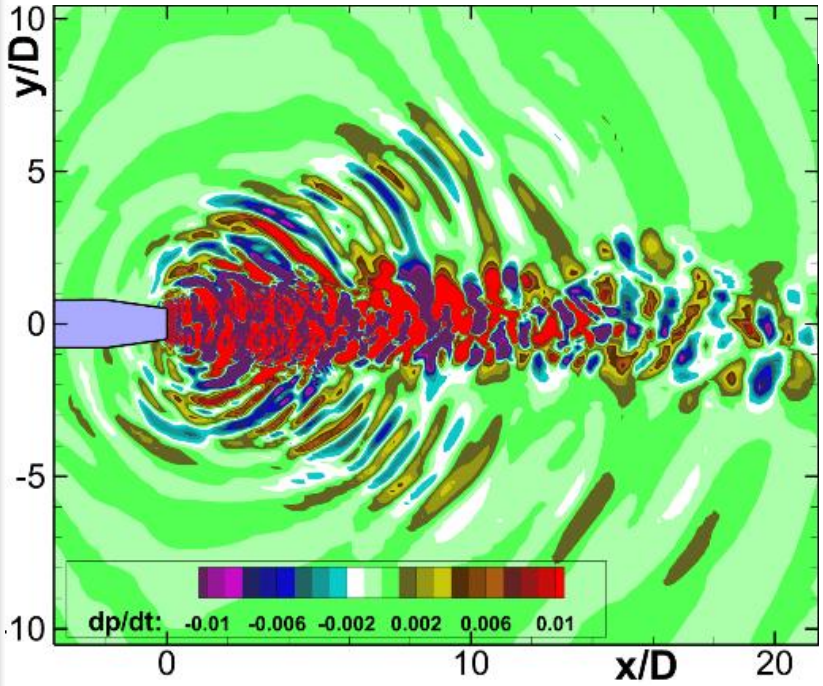
Влияние пространственного разрешения

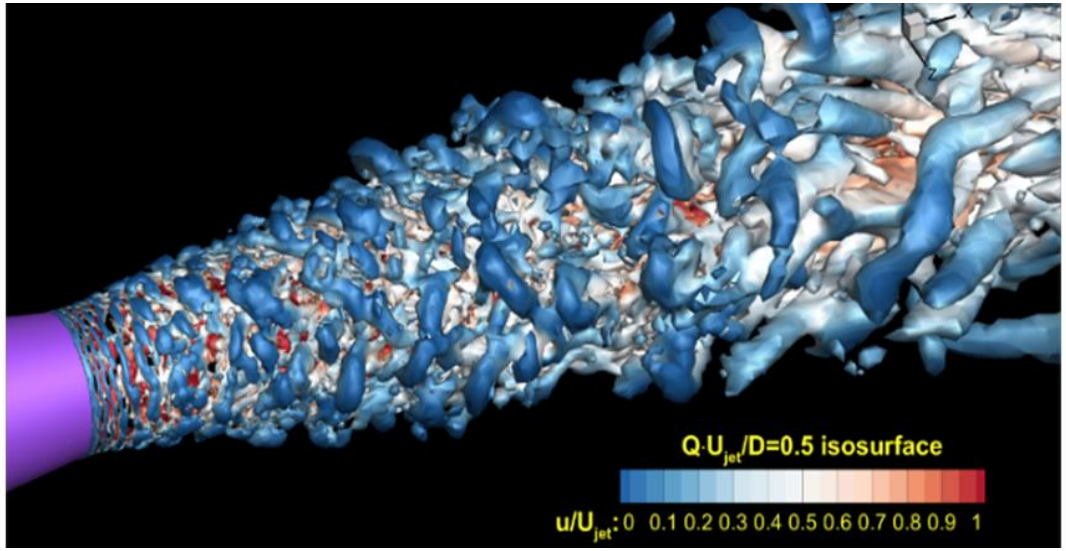
Сетка 1.5 млн узлов

Параметры течения:

Re=**$1.1 \cdot 10^6$**, M =**0.9**





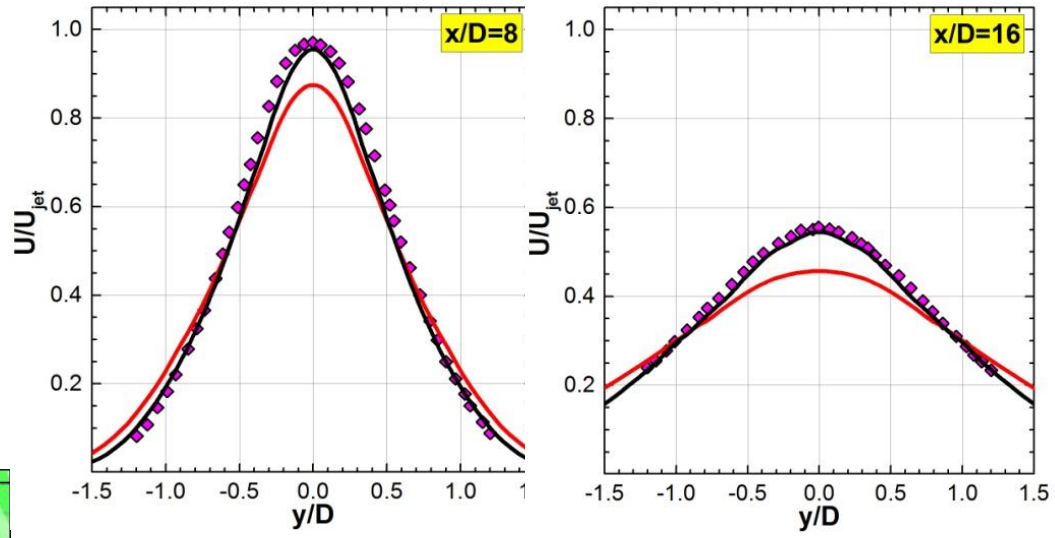


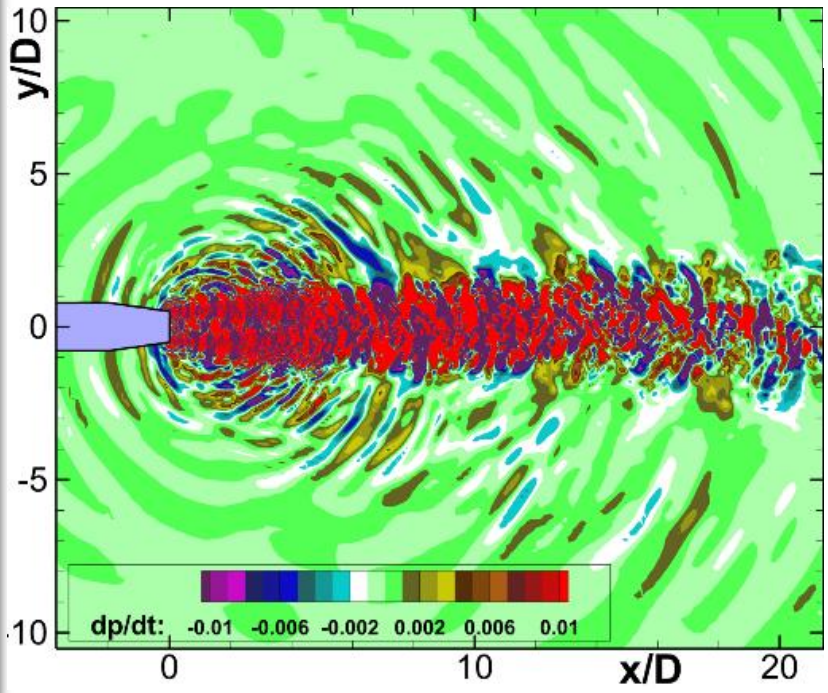
Влияние пространственного разрешения

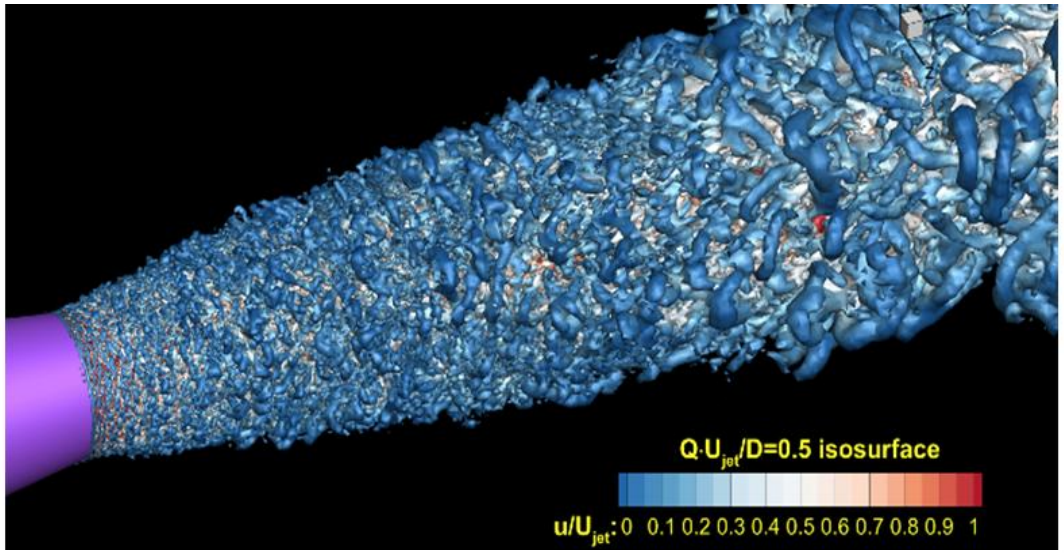
Сетка 8.8 млн узлов

Параметры течения:

Re=**$1.1 \cdot 10^6$**, M =**0.9**







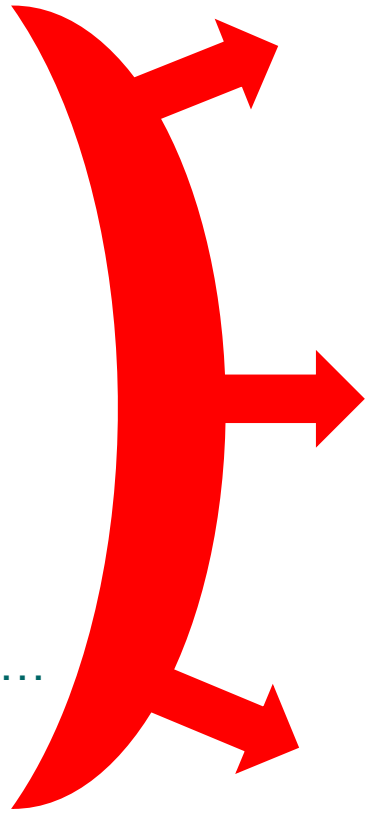
Высокая ресурсоемкость вихреразрешающего моделирования

Для точных результатов надо использовать подробную сетку и сделать много шагов по времени

Прожигание CPU времени космического масштаба (и космической же …)
Как бороться с этой неприятностью?

Наступление по трем фронтам:

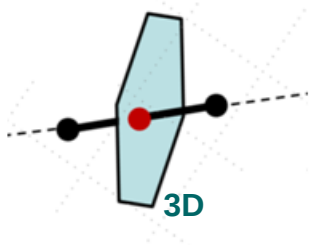
- **Экономичные и точные численные методы**
  численные схемы, модели турбулентности, …
- **Эффективные HPC реализации**
  масштабируемые алгоритмы, гетерогенные вычисления, …
- **Улучшение технологии выполнения расчета**
  адаптация сетки, ускорение развития течения, постпроцесс, …

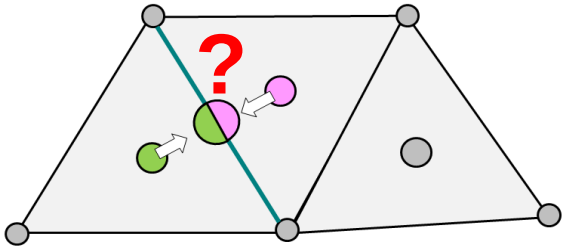


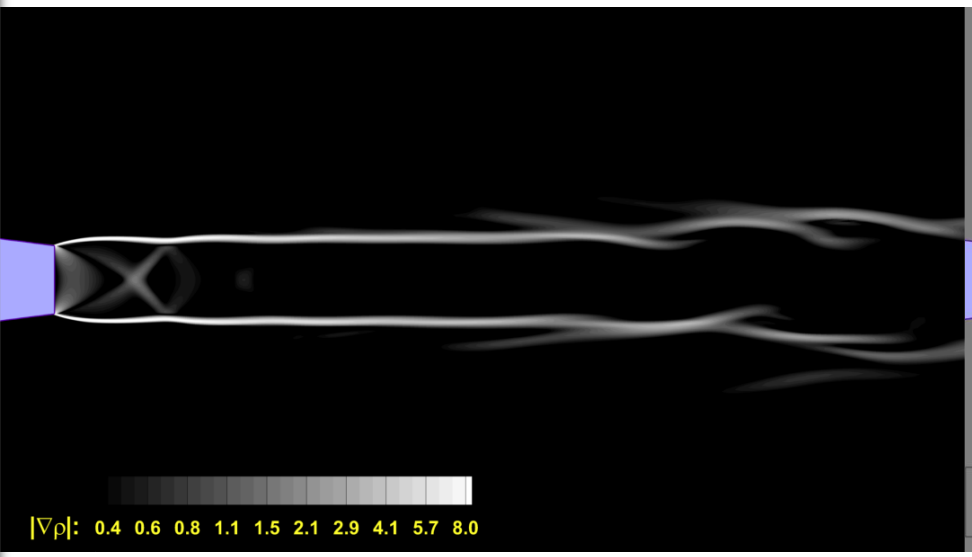
Почему так важна точность численной схемы?

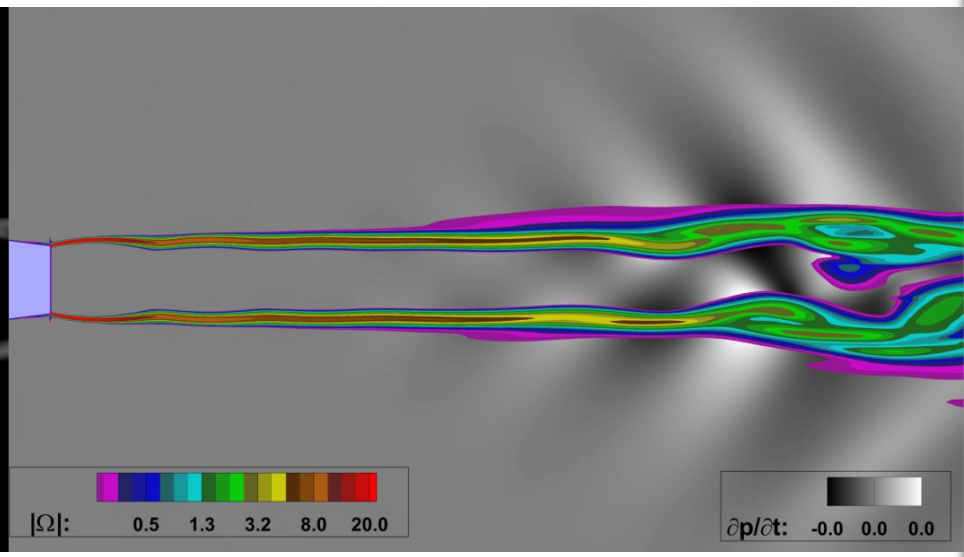
Пример: моделирование недорасширенной горячей струи

Базовая схема 1 порядка





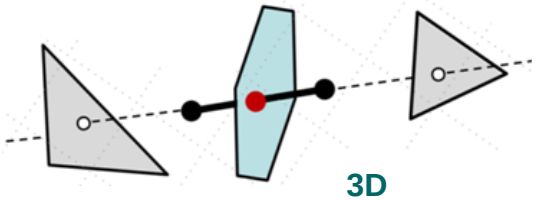




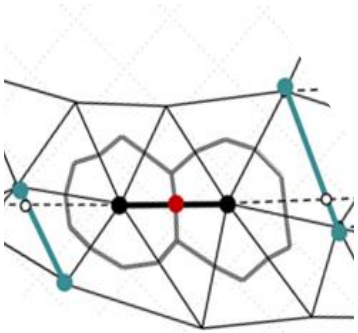
Экономичные и точные численные методы: EBR схемы

Пример: моделирование недорасширенной горячей струи

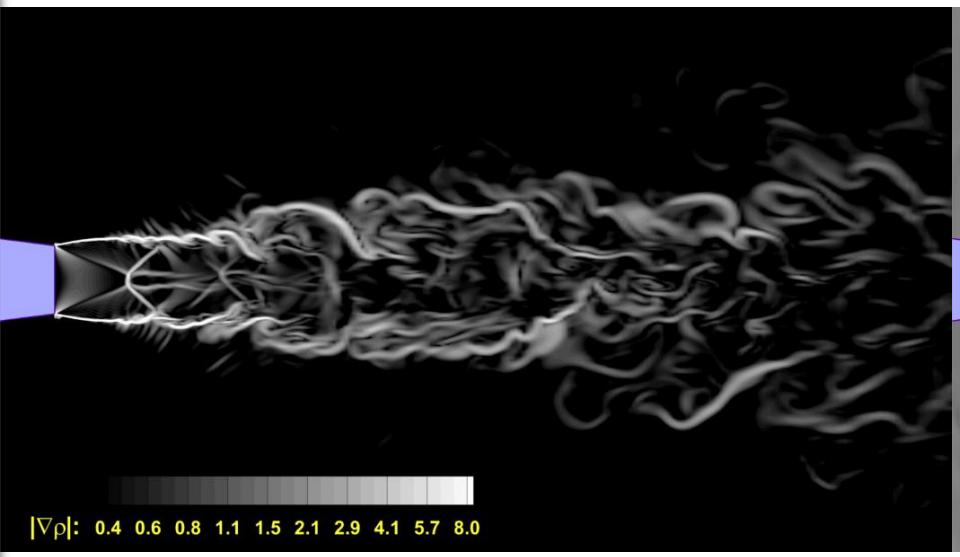
Схема EBR3

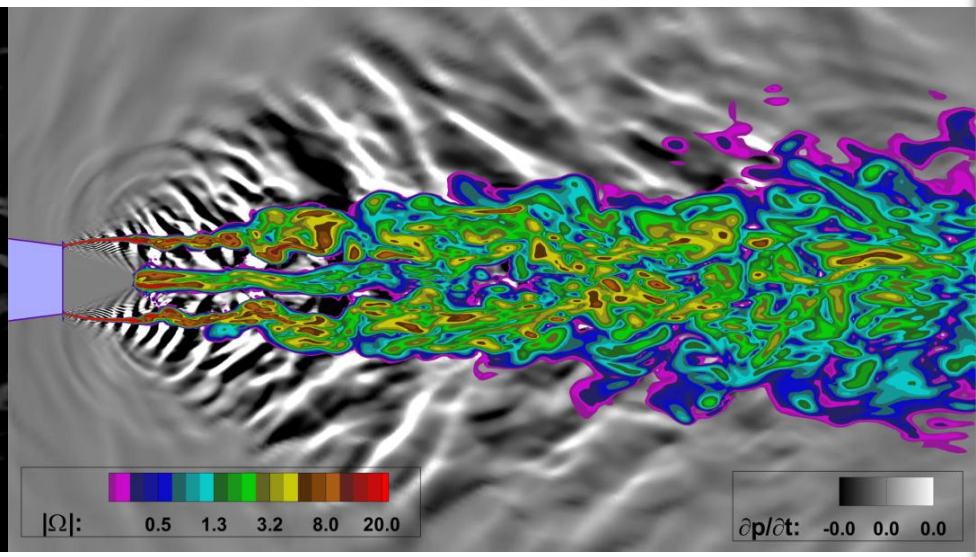


3D



2D

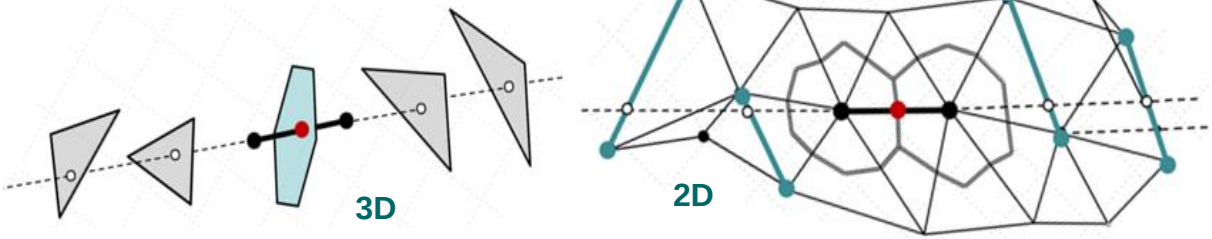


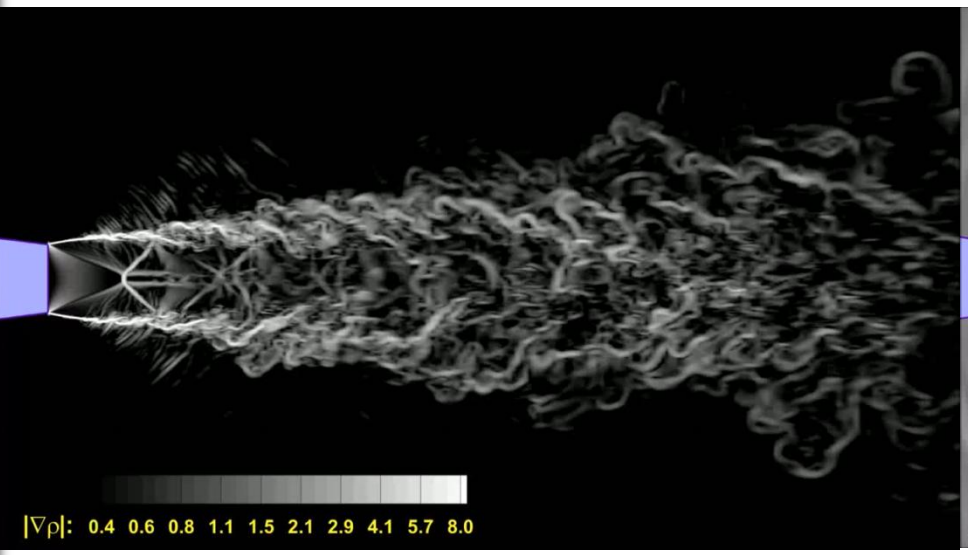


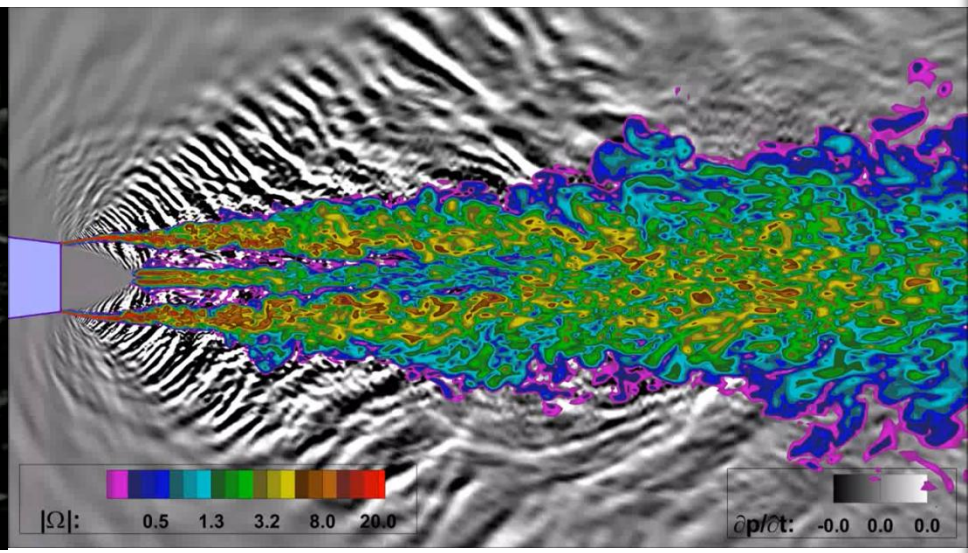
Экономичные и точные численные методы: EBR схемы

Пример: моделирование недорасширенной горячей струи

Схема EBR5





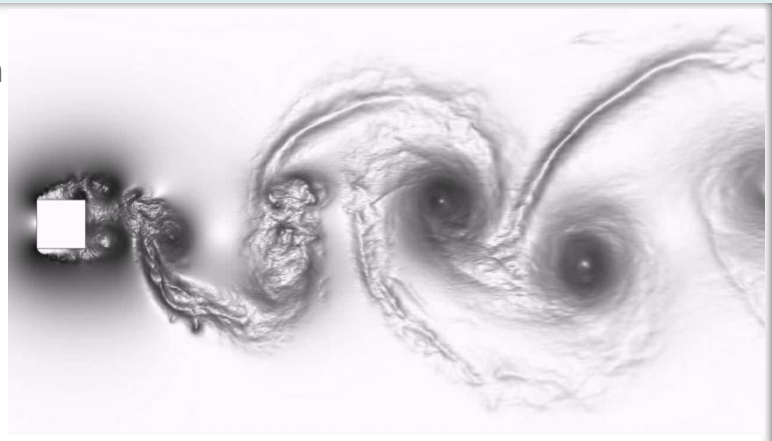


+15% к стоимости относительно базовой схемы

До **5-го порядка** точности (на ТС сетках)

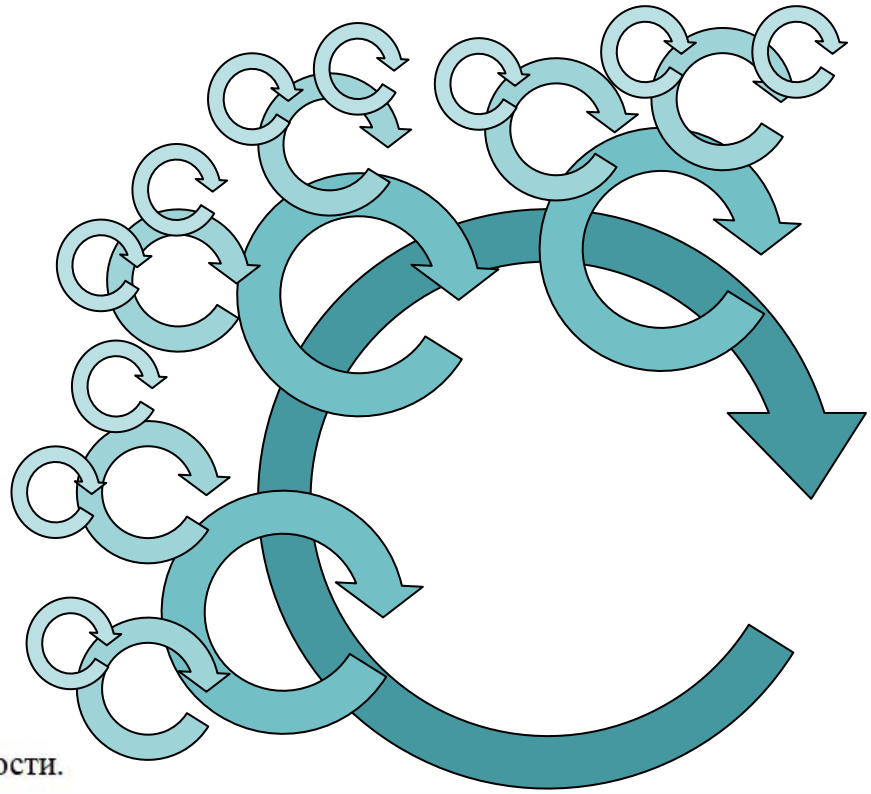
Моделирование турбулентности

- **Прямое численное моделирование DNS** 1-100M CPUh
  стоимость точность
- **Моделирование крупных вихрей LES**
  стоимость точность
- **RANS – осредненные уравнения Н-С**
  стоимость точность
- **RANS-LES гибридные подходы (DES)**
  стоимость точность



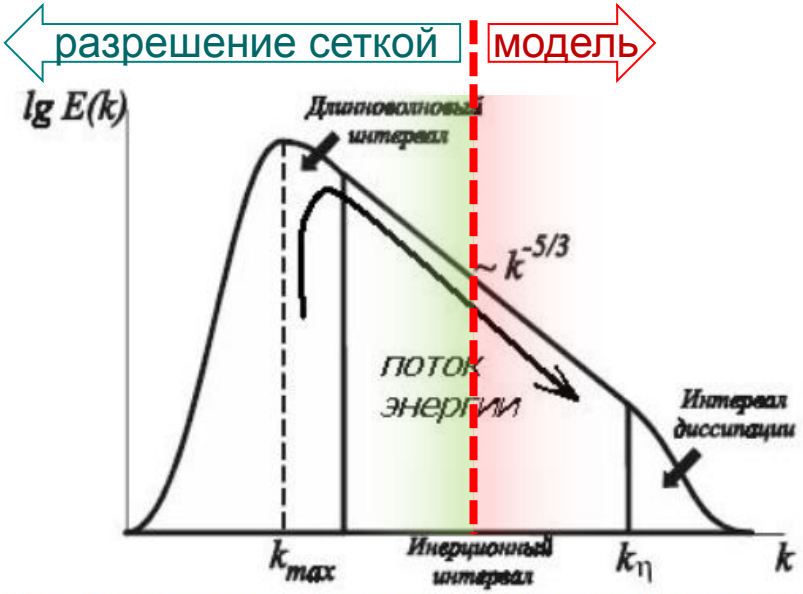


Энергетический спектр изотропной трехмерной турбулентности.

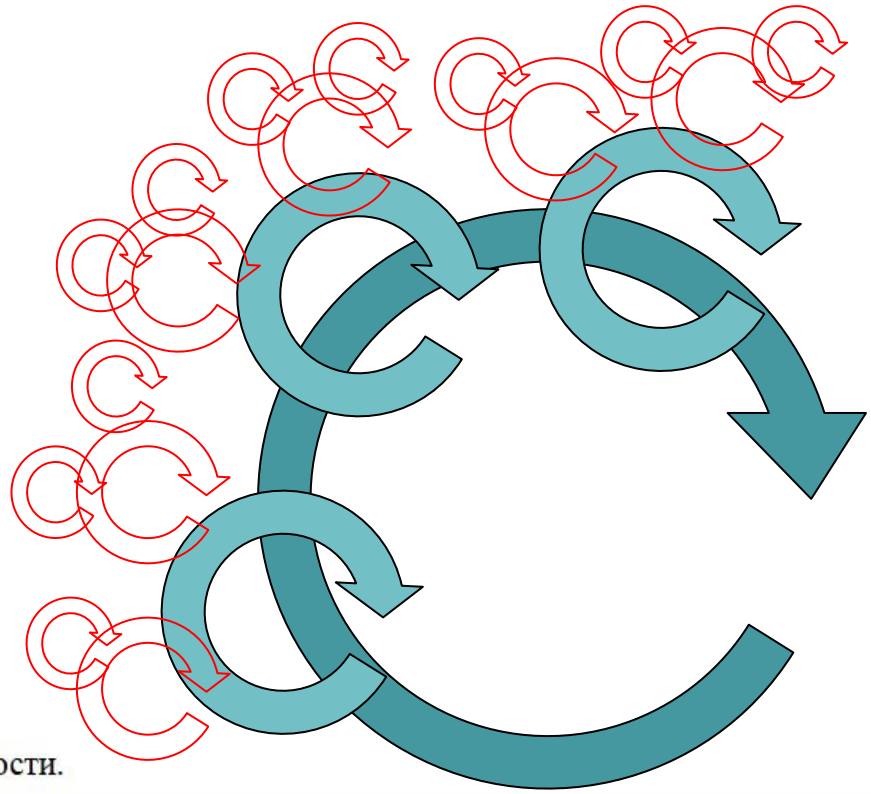


Моделирование турбулентности

- **Прямое численное моделирование DNS** 1 – 100M CPUh
  стоимость точность
- **Моделирование крупных вихрей LES** 100K – 10M CPUh
  стоимость точность
- **RANS – осредненные уравнения Н-С**
  стоимость точность
- **RANS-LES гибридные подходы (DES)**
  стоимость точность



Энергетический спектр изотропной трехмерной турбулентности.



Моделирование турбулентности

- **Прямое численное моделирование DNS** 1 – 100M CPUh

  СТОИМОСТЬ  ТОЧНОСТЬ 

- **Моделирование крупных вихрей LES** 100K – 10M CPUh

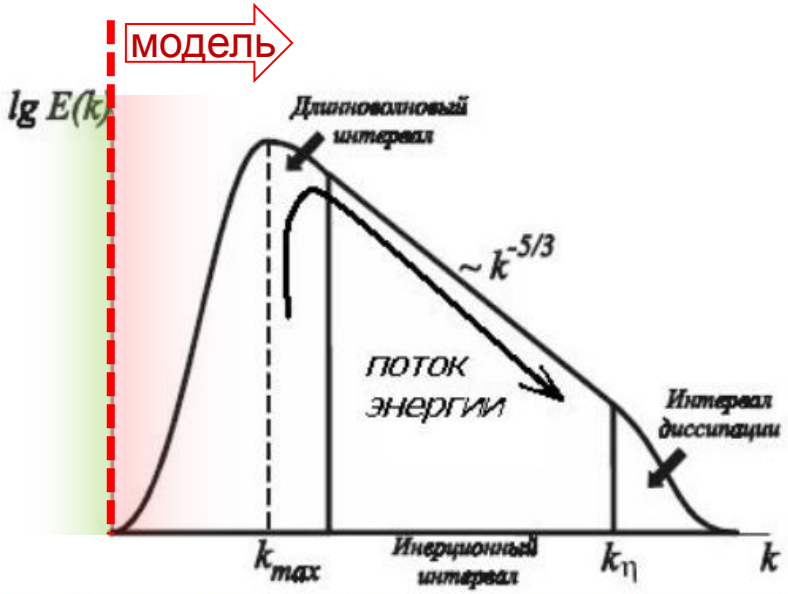
  СТОИМОСТЬ  ТОЧНОСТЬ 

- **RANS – осредненные уравнения Н-С** 100 – 10K CPUh

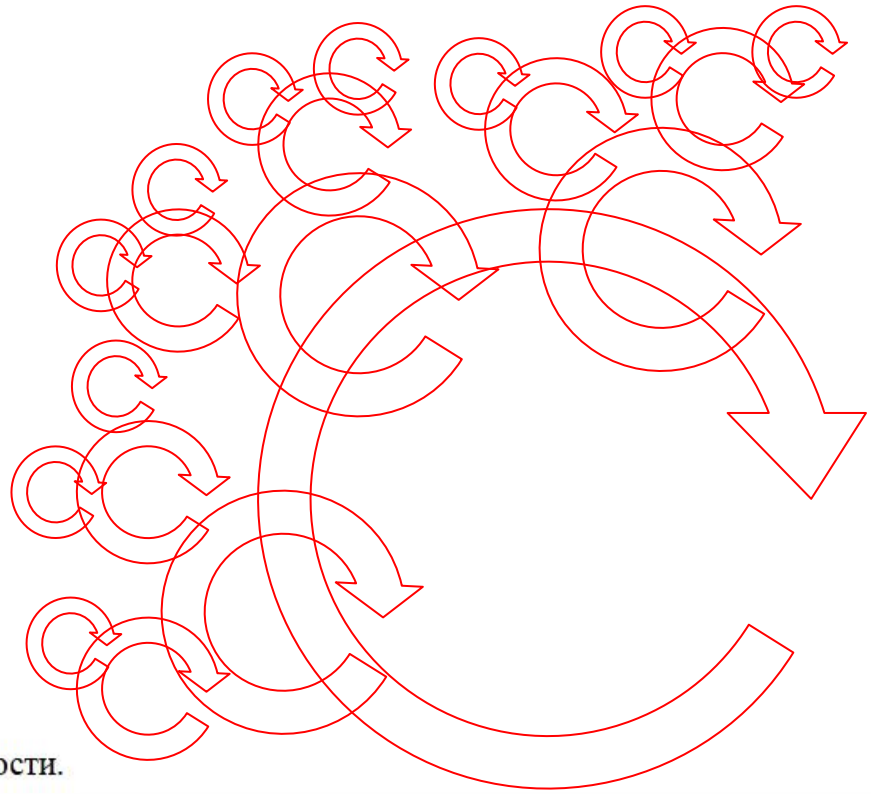
  СТОИМОСТЬ  ТОЧНОСТЬ 

- **RANS-LES гибридные подходы (DES)**

  СТОИМОСТЬ ТОЧНОСТЬ

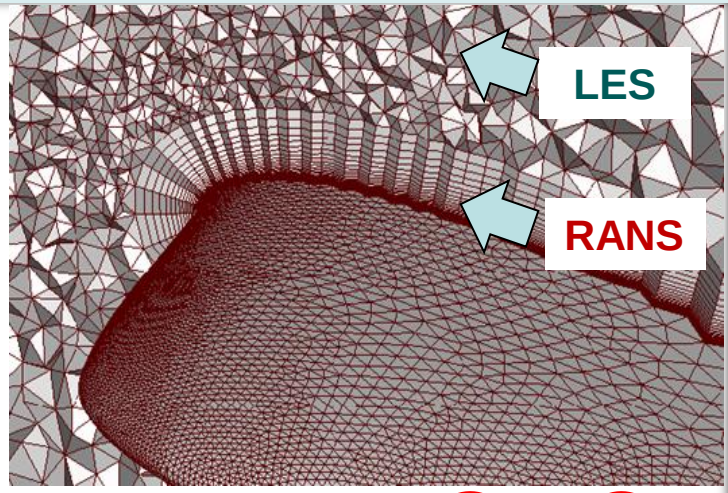


Энергетический спектр изотропной трехмерной турбулентности.

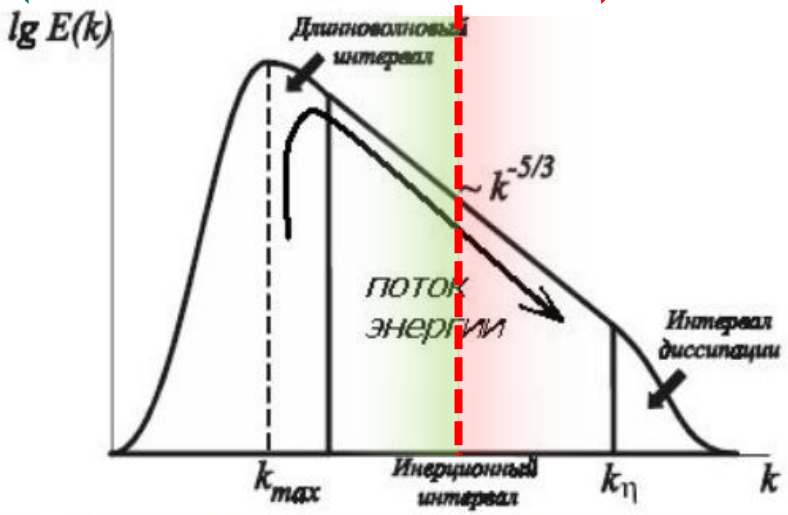


Моделирование турбулентности

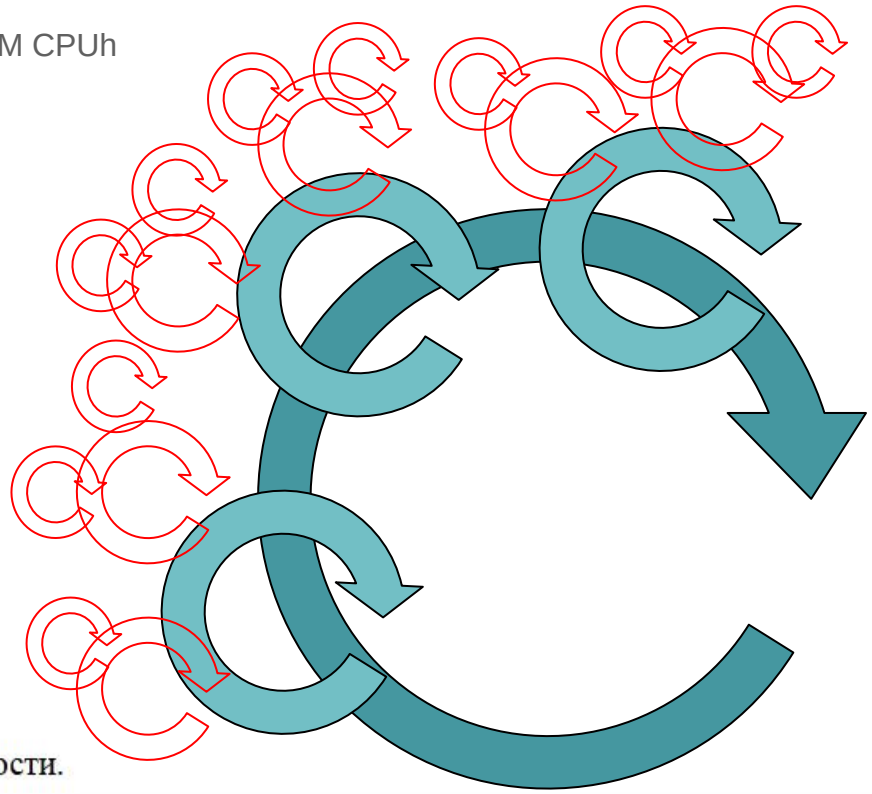
- **Прямое численное моделирование DNS** 1 – 100M CPUh
  стоимость ▬▬▬▬▬ точность ▬▬▬▬▬
- **Моделирование крупных вихрей LES** 100K – 10M CPUh
  стоимость ▬▬ точность ▬▬
- **RANS – осредненные уравнения Н-С** 100 – 10K CPUh
  стоимость ▮ точность ▮
- **RANS-LES гибридные подходы (DES)** 10K – 1M CPUh
  стоимость ▬▬ точность ▬▬



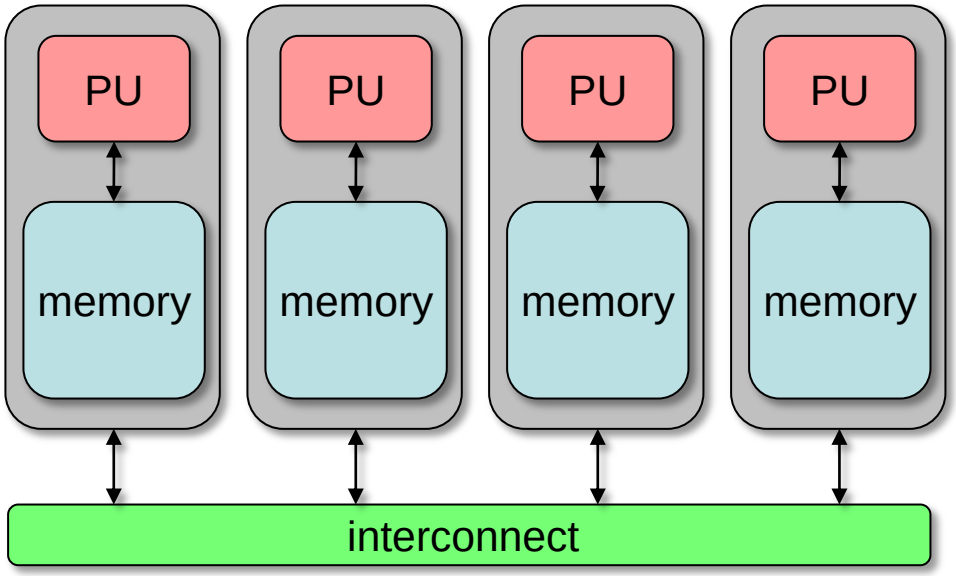
разрешение сеткой | модель



Энергетический спектр изотропной трехмерной турбулентности.



Высокопроизводительные вычисления – HPC

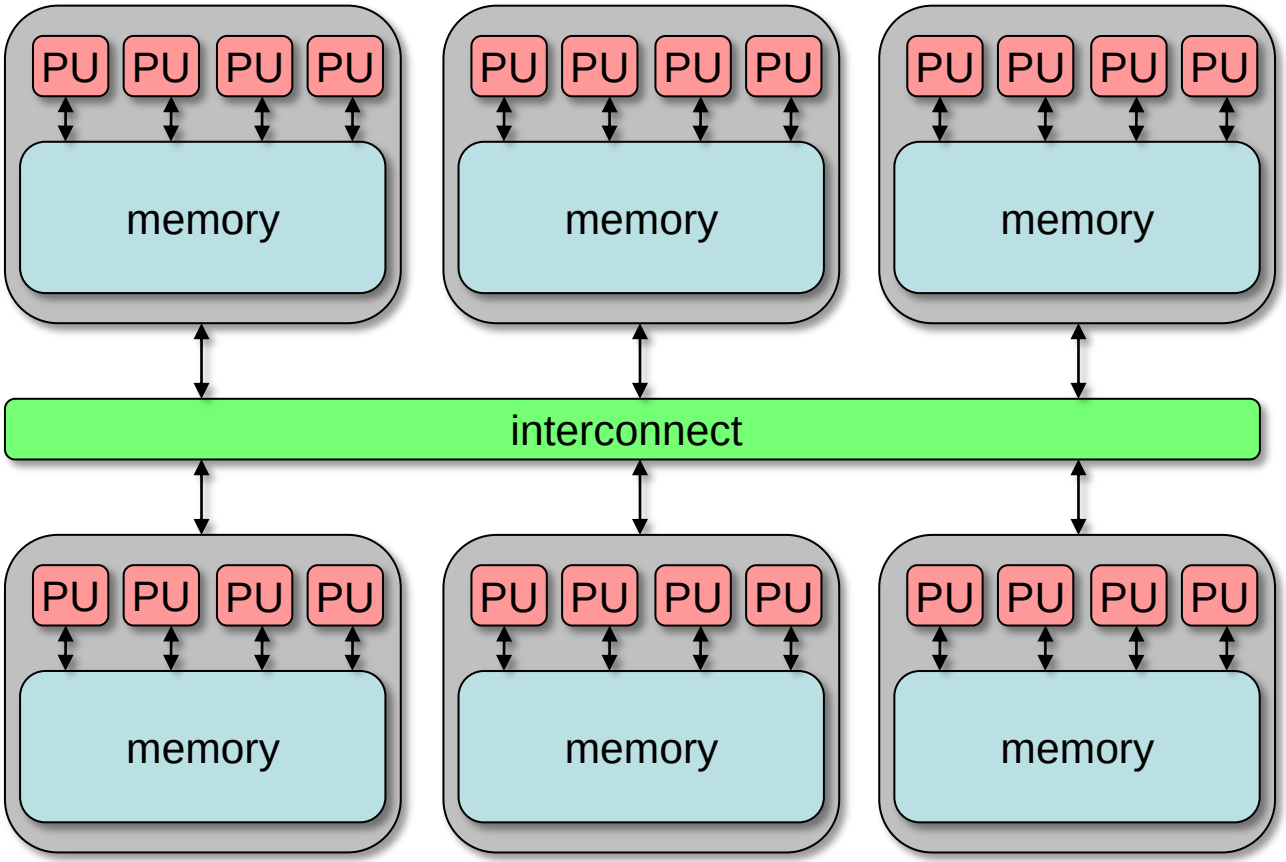




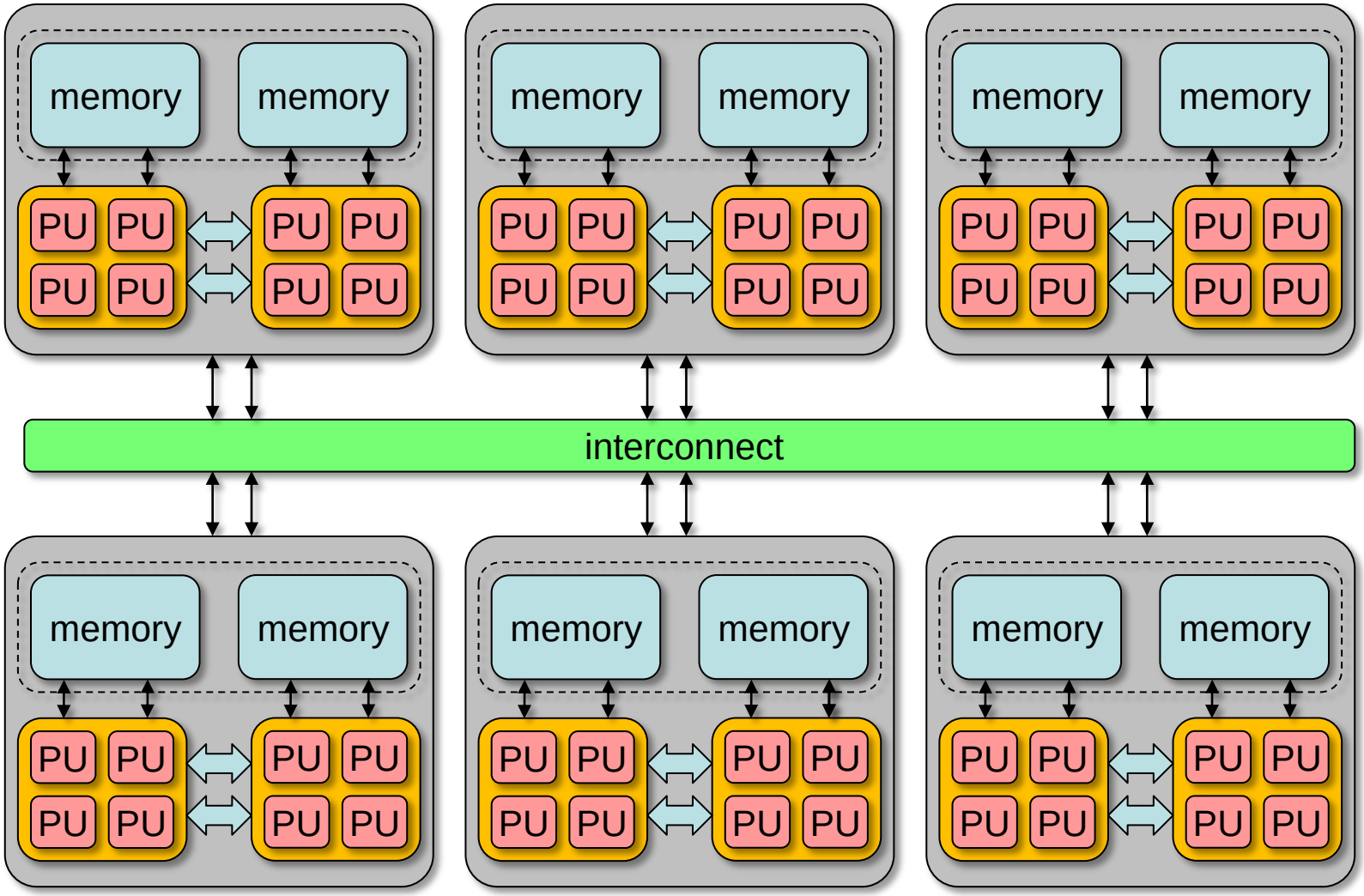




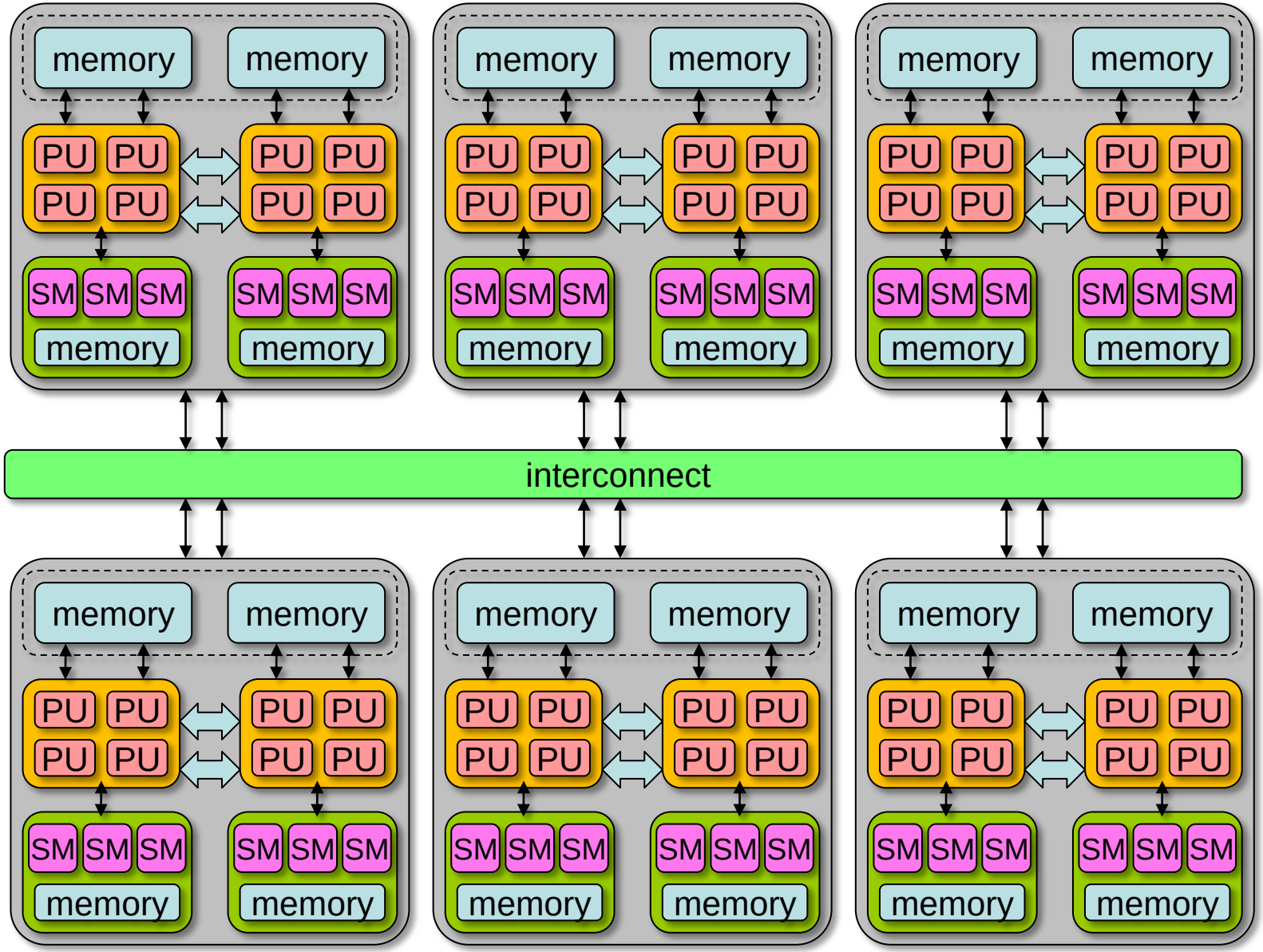
Высокопроизводительные вычисления – HPC



Высокопроизводительные вычисления – HPC



Высокопроизводительные вычисления – HPC



Суперкомпьютерная фаллометрия по HPL

HPL
High
Performance
Linpack

Прямое решение СЛАУ с плотной LU разложением

$O(N^2)$ по памяти

$O(N^3)$ FLOP

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science	7,299,072	415,530.0	513,854.7	28,335
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL	1,572,480	94,640.0	125,712.0	7,438
4	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
5	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
6	**HPC5** - PowerEdge C4140, Xeon Gold 6252 24C 2.1GHz, NVIDIA Tesla V100, Mellanox HDR Infiniband, Dell EMC Eni S.p.A. Italy	669,760	35,450.0	51,720.8	2,252
7	**Selene** - DGX A100 SuperPOD, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States	272,800	27,580.0	34,568.6	1,344
8	**Frontera** - Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR, Dell EMC Texas Advanced Computing Center/Univ. of Texas United States	448,448	23,516.4	38,745.9	

https://www.top500.org/

Рейтинг по странам

2010	Countries	Count
1	United States	282
2	United Kingdom	38
3	France	27
4	China	24
5	Germany	24
6	Japan	18
7	Russia	11
8	Sweden	8
9	New Zealand	7
10	Canada	7
11	Italy	7
12	Poland	5
13	India	5
14	Switzerland	5
15	Netherlands	4
16	Saudi Arabia	4
17	Finland	3
18	Spain	3
19	Denmark	3
20	Israel	2
21	Norway	2
22	Austria	2
23	South Korea	1
24	Brazil	1
25	Turkey	1

2015	Countries	Count
1	United States	233
2	Japan	40
3	Germany	38
4	China	37
5	United Kingdom	29
6	France	27
7	India	11
8	South Korea	9
9	Russia	8
10	Saudi Arabia	7
11	Poland	7
12	Switzerland	6
13	Canada	6
14	Brazil	6
15	Australia	5
16	Sweden	5
17	Italy	4
18	Netherlands	3
19	Norway	2
20	Denmark	2
21	Spain	2
22	Finland	2
23	Hungary	1
24	Belgium	1
25	Austria	1

2018	Countries	Count
1	China	206
2	United States	124
3	Japan	36
4	United Kingdom	22
5	Germany	21
6	France	18
7	Netherlands	9
8	Ireland	7
9	South Korea	7
10	Canada	6
11	India	5
12	Italy	5
13	Australia	5
14	Saudi Arabia	4
15	Russia	4
16	Poland	4
17	Switzerland	3
18	Sweden	3
19	Spain	2
20	Singapore	2
21	Norway	1
22	Finland	1
23	Taiwan	1
24	Czechia	1
25	Brazil	1

2020	Countries	Count
1	China	226
2	United States	113
3	Japan	29
4	France	19
5	Germany	16
6	Netherlands	15
7	Ireland	14
8	Canada	13
9	United Kingdom	10
10	Italy	7
11	Brazil	4
12	Singapore	4
13	South Korea	3
14	Norway	3
15	Saudi Arabia	3
16	Australia	2
17	United Arab Emirates	2
18	Sweden	2
19	Finland	2
20	Switzerland	2
21	Taiwan	2
22	Russia	2
23	India	2
24	Poland	1
25	Hong Kong	1

Суперкомпьютерная фаллометрия по HPCG

HPCG
High Performance Conjugate Gradient

Итерационное решение СЛАУ с разреженной матрицей

Компот из CG, MG, Гаусса-Зейделя

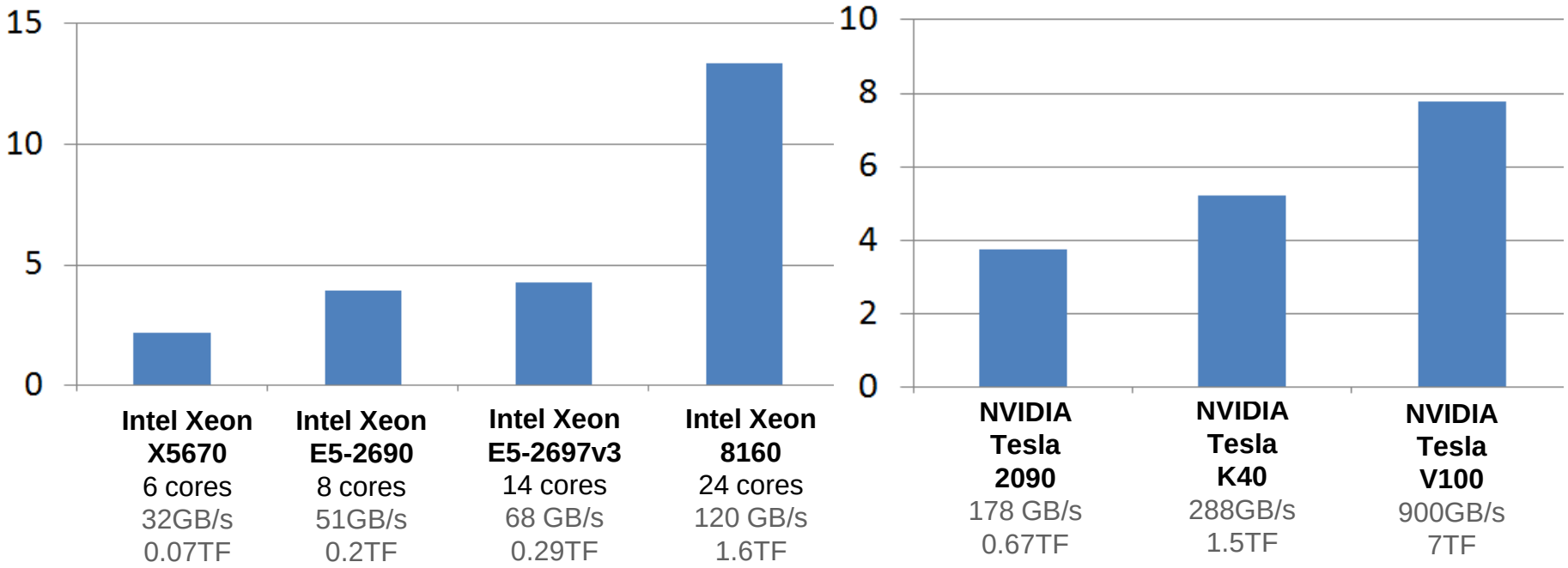
O(N) по памяти

O(N) FLOP

Rank	Site	Computer	Cores	HPL Rmax (Pflop/s)	TOP500 Rank	HPCG (Pflop/s)	Fraction of Peak
1	RIKEN Center for Computational Science Japan	**Supercomputer Fugaku** – Fujitsu A64X, A64FX 6635520C 2.2GHz, Tofu Interconnect D Fujitsu	6,635,520	415.530	1	13.366	2.6%
2	DOE/SC/ORNL USA	**Summit** – AC922, IBM POWER9 22C 3.07GHz, dual-rail Mellanox EDR Infiniband, NVIDIA Volta V100 IBM	2,414,592	148.600	2	2.926	1.5%
3	DOE/NNSA/LLNL USA	**Sierra** – S922LC, Power9 22C 3.1GHz, Mellanox EDR, NVIDIA Tesla V100 IBM / NVIDIA / Mellanox	1,572,480	94.640	3	1.796	1.4%
4	Eni Green Data Center Italy	**HPC5** – PowerEdge C4140, Intel Xeon-Gold 6252 24C 2.1GHz, Infiniband HDR100, Nvidia Tesla V100 PCIe 32GB Dell EMC	128	35.450	6	0.860	1.7%
5	DOE/NNSA/LANL/SNL USA	**Trinity** – Cray XC40, Intel Xeon E5-2698 v3 16C 2.3GHz, Aries, Intel Xeon Phi 7250 68C 1.4GHz Cray	979,072	20.159	11	0.546	1.3%
6	NVIDIA USA	**Selene** – NVIDIA DGX A100, AMD EPYC 7742 35200C 2.25GHz, Mellanox HDR InfiniBand, NVIDIA Tesla A100 40GB NVIDIA	272,800	27.580	7	0.509	1.5%
7	National Institute of Advanced Industrial Science and Technology (AIST) Japan	**AI Bridging Cloud Infrastructure (ABCI)** – PRIMERGY CX2570M4, Intel Xeon Gold 6148 20C 2.4GHz, Infiniband EDR, NVIDIA Tesla V100 Fujitsu	391,680	19.880	12	0.509	1.6%
8	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** – Cray XC50, Intel Xeon E5-2690v3 12C 2.6GHz, Cray Aries, NVIDIA Tesla P100 16GB Cray	387,872	21.230	10	0.497	1.8%
9	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** – Sunway MPP, SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93.015	4	0.481	0.4%
10	Korea Institute of Science and Technology Information Republic of Korea	**Nurion** – CS500, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path, Intel Xeon Phi 7250 Cray	570,020	13.929	18	0.391	1.5%

http://www.hpcg-benchmark.org/

Разрыв между производительностью и пропускной способностью



Отношение пиковой производительности
и пропускной способности памяти

SIMD extension vs Bandwidth

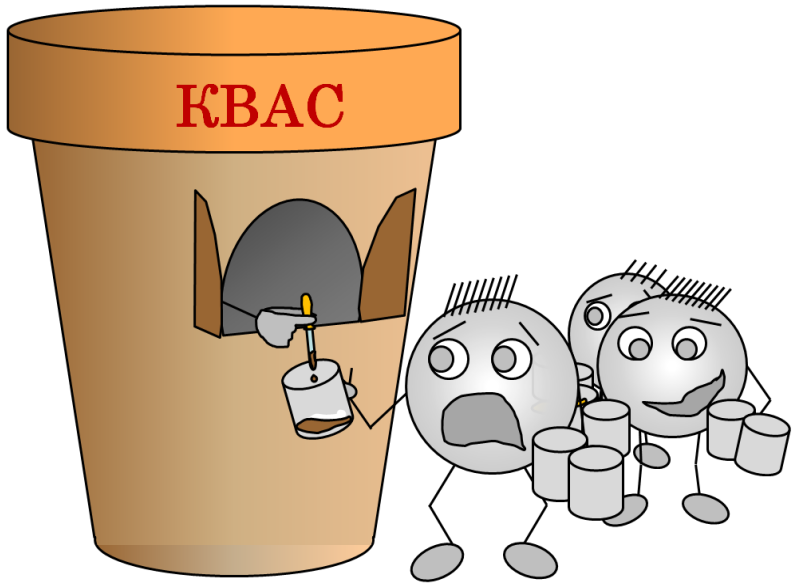
CPU 24C, 2 ГГц, **2** x AVX-512 FMA

Z[i] = X[i] + Y[i]*Z[i];

3 x 8B x 512/64 x 2 x 24 = 9216B x 2 ГГц = 18 ТБ/с

6 x DDR4-2666 BW = 6 x 2666 x 8 = 128 ГБ/с

1:144



Зоопарк вычислительных устройств

CPU: multicore, superscalar, out-of-order или WLIV, SIMD – Intel, AMD, IBM, ARM, Эльбрус, …













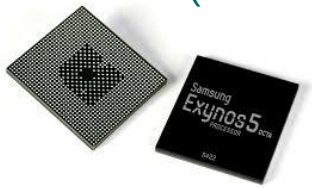
Manycore: Intel Xeon Phi, A64FX ARM







SoC CPU+GPU (MONT-BLANC 1 prototype)



Ultra-manycore: PEZY-SC, SunWay



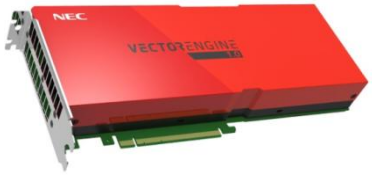


GPU: stream processing – NVIDIA, AMD, Intel





Векторные процессоры: SIMD – NEC SX

FPGA ускорители



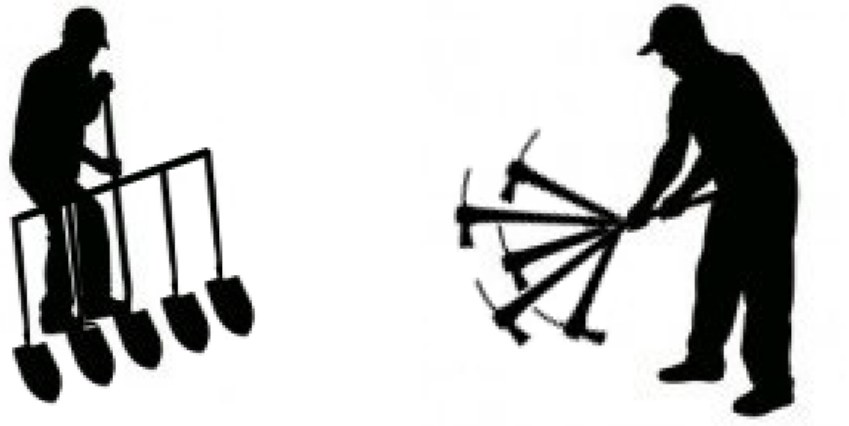


MIMD vs SIMD

MIMD



SIMD



Уровни параллелизма и средства разработки

DM MIMD

MPI

Суперкомпьютер

Гибридный узел

SM MIMD

OpenMP
OpenCL
MPI

Многоядерный CPU

Многопоточное ядро

SIMD

OpenMP, OpenCL,
директивы компилятора, …

Out-of-order, VLIW

SIMD расширение

ILP параллелизм

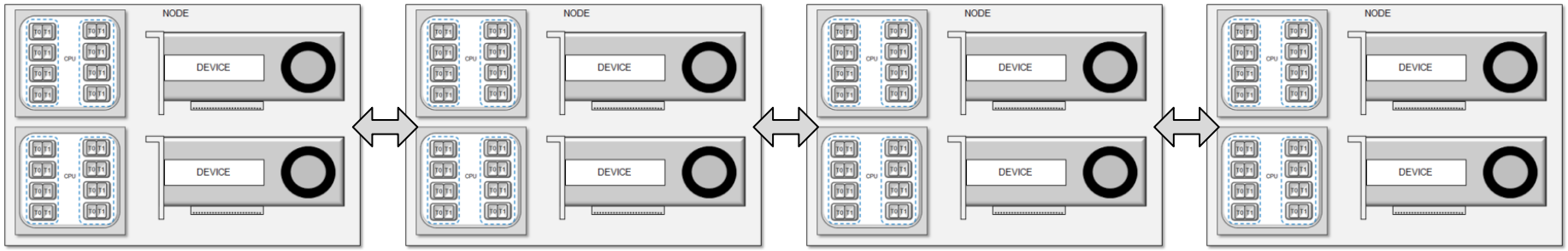
GPU ускоритель

Потоковый мультипроцессор

OpenCL

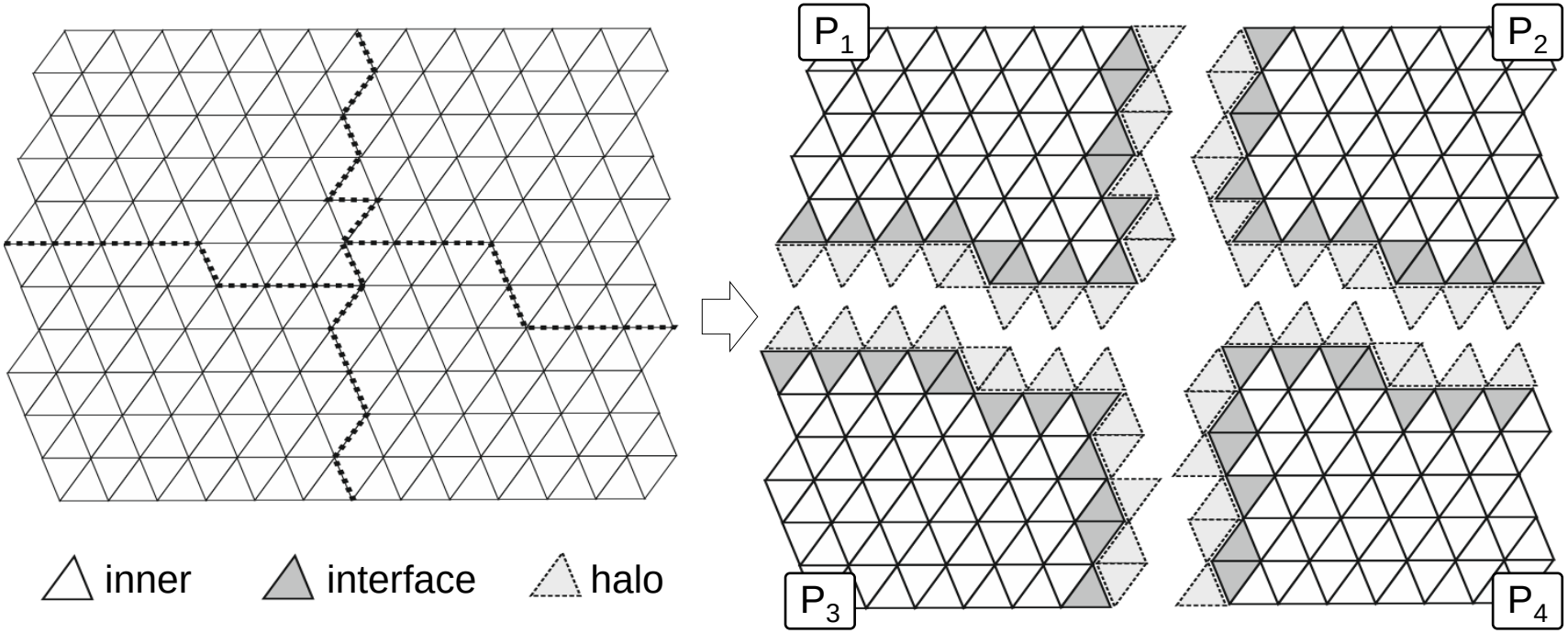
CUDA

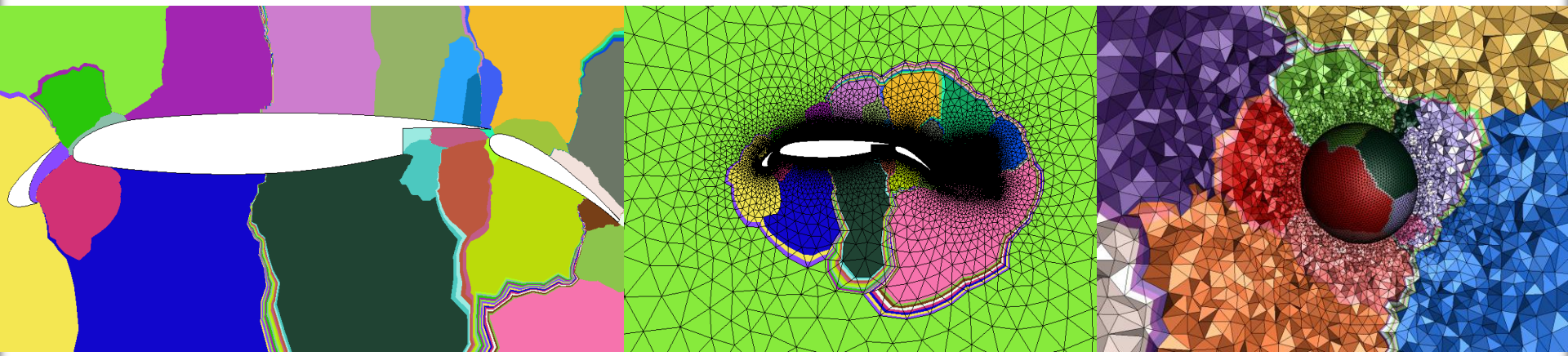
Stream Processing



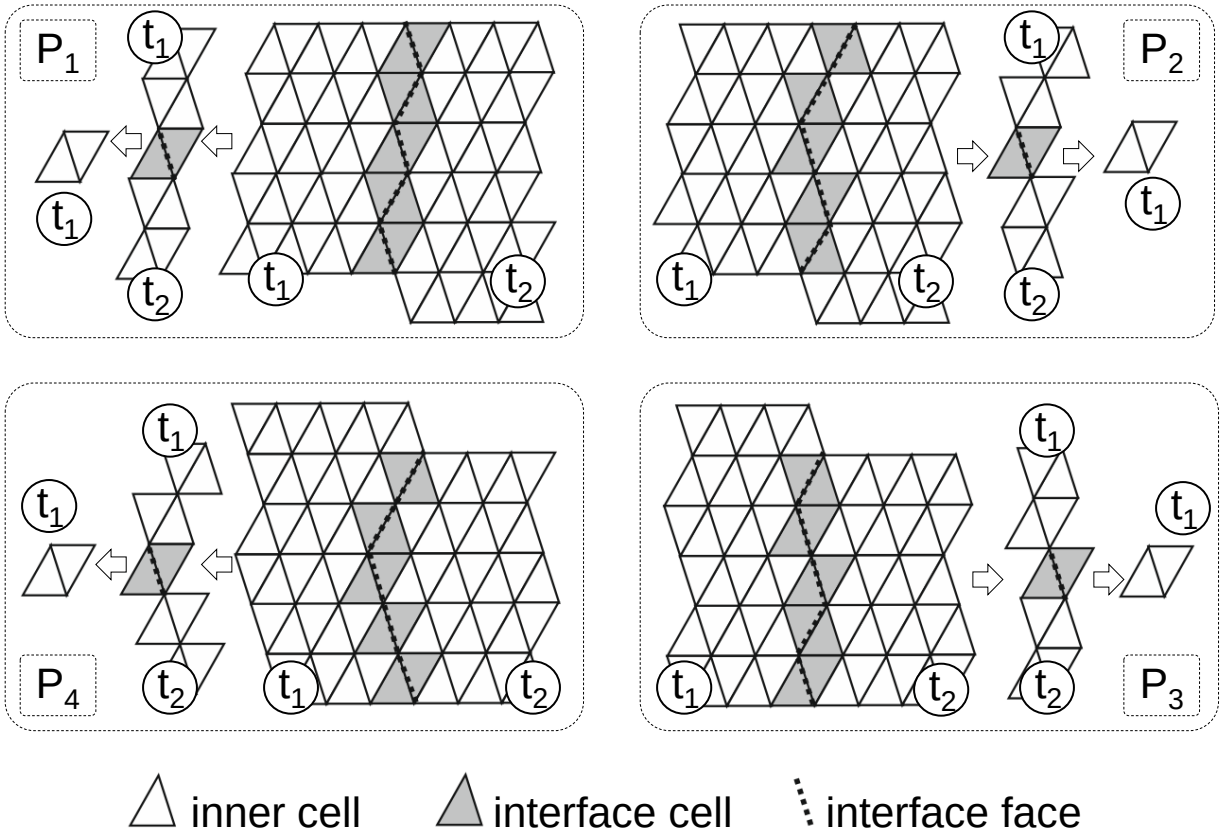
Многоуровневое распараллеливание

MPI – распараллеливание с распределенной памятью DM-MIMD





OpenMP – распараллеливание с общей памятью SM-MIMD

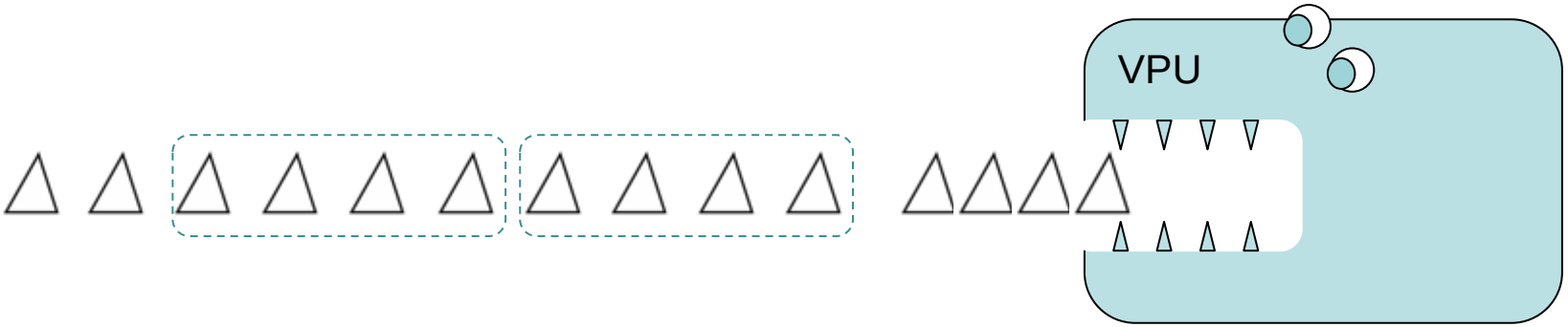


A. Gorobets. Parallel Algorithm of the NOISEtte Code for CFD and CAA Simulations. Lobachevskii J Math (2018) 39: 524. https://doi.org/10.1134/S1995080218040078

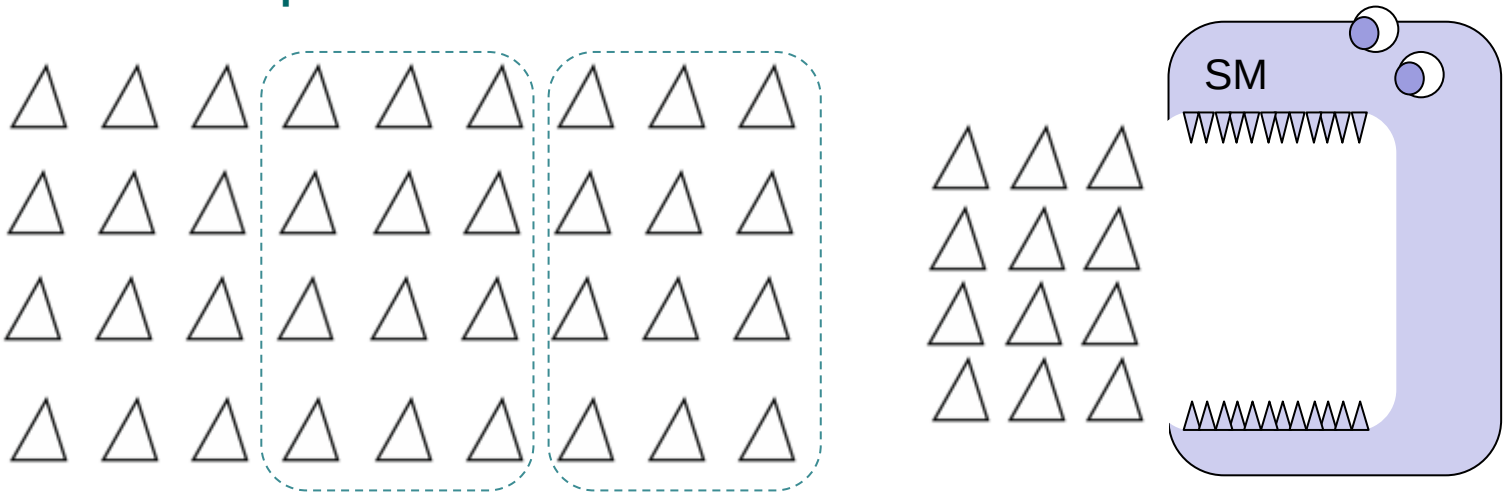
Многоуровневое распараллеливание

SIMD

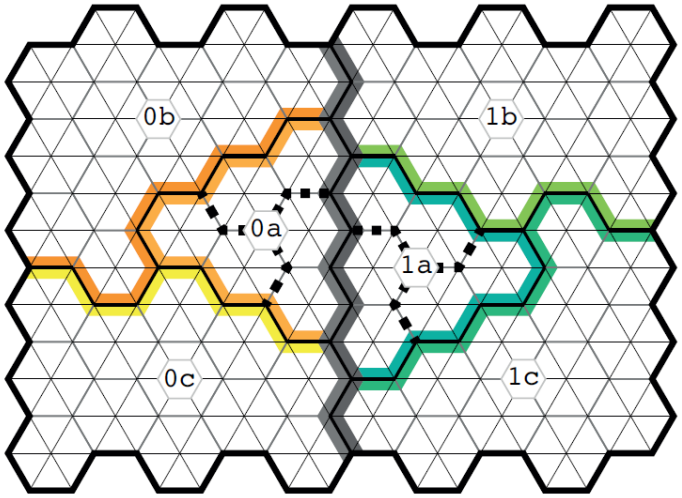
Векторизация



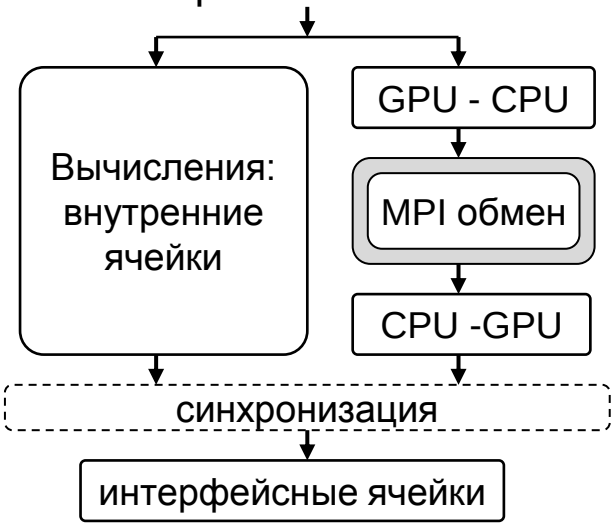
Потоковая обработка

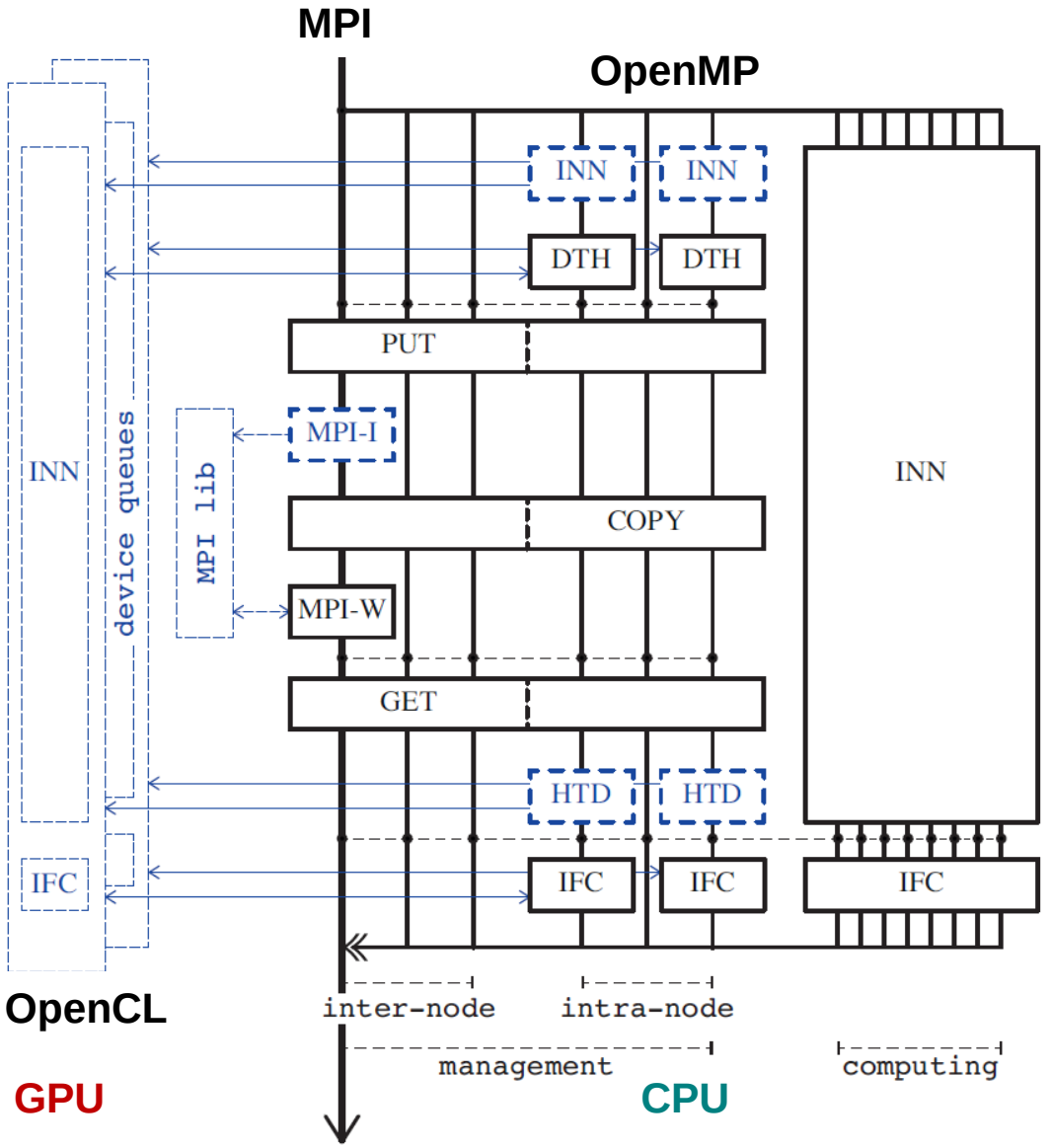


OpenCL – вычисления на графических процессорах GPU



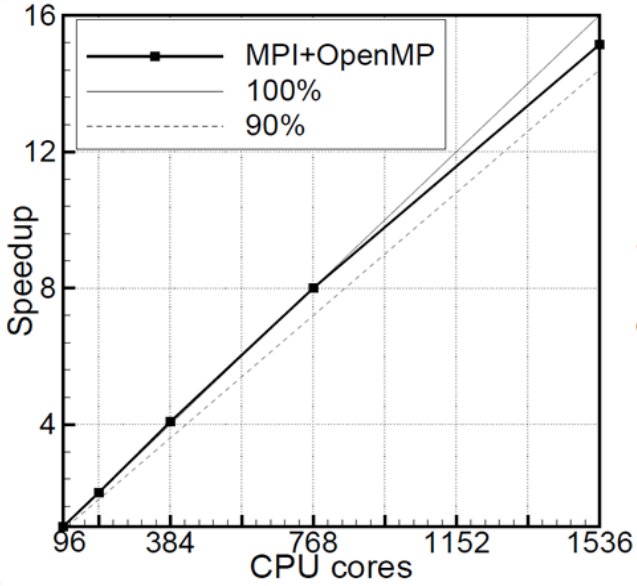
Overlap

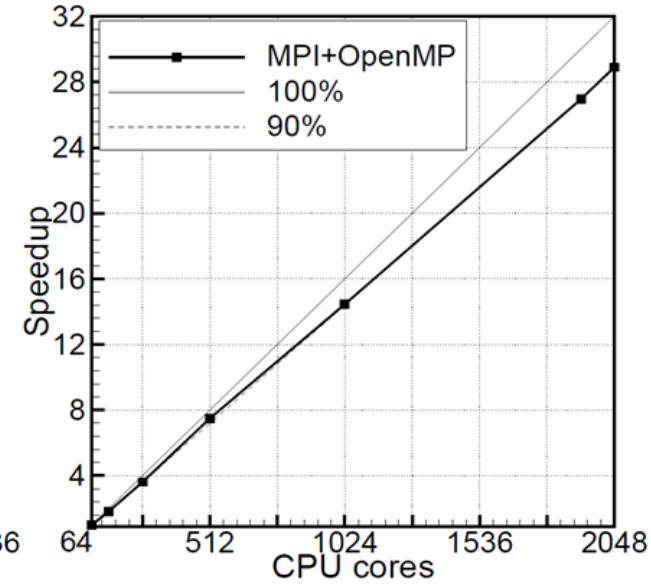


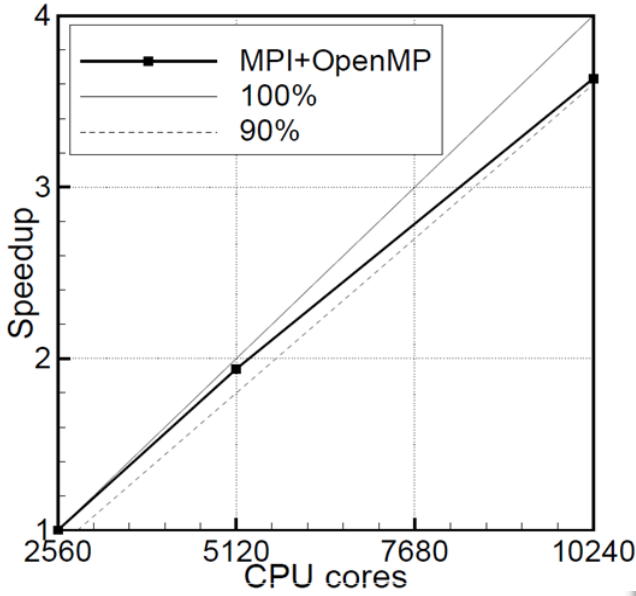


Параллельные технологии

- **Многоуровневое распараллеливание MPI+OpenMP+OpenCL для гибридных систем**
- **Высокая производительность вычислений**
- **Высокая параллельная эффективность на десятках тысяч ядер**
- **Расчетный алгоритм полностью портирован на GPU:** NVIDIA V100 = 100 ~ 150 ядер CPU
- **Переносимость: CPU, ускорители Intel Xeon Phi, GPU AMD и NVIDIA;** Linux, Windows

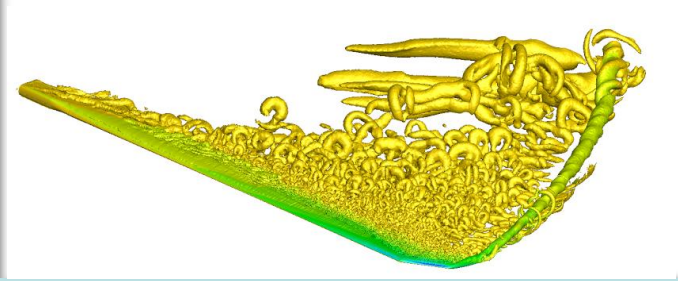




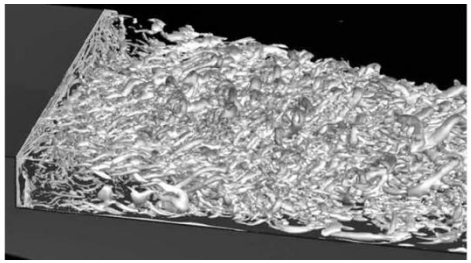


Параллельная эффективность в реальных расчетах (неявная схема, повышенная точность, DES)

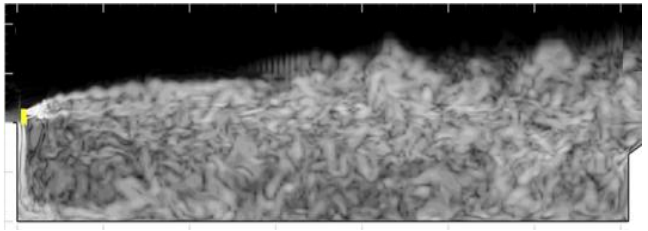
HPC4, 22 млн узлов, Лопасть винта



МВС-10П,13 млн узлов, BFS

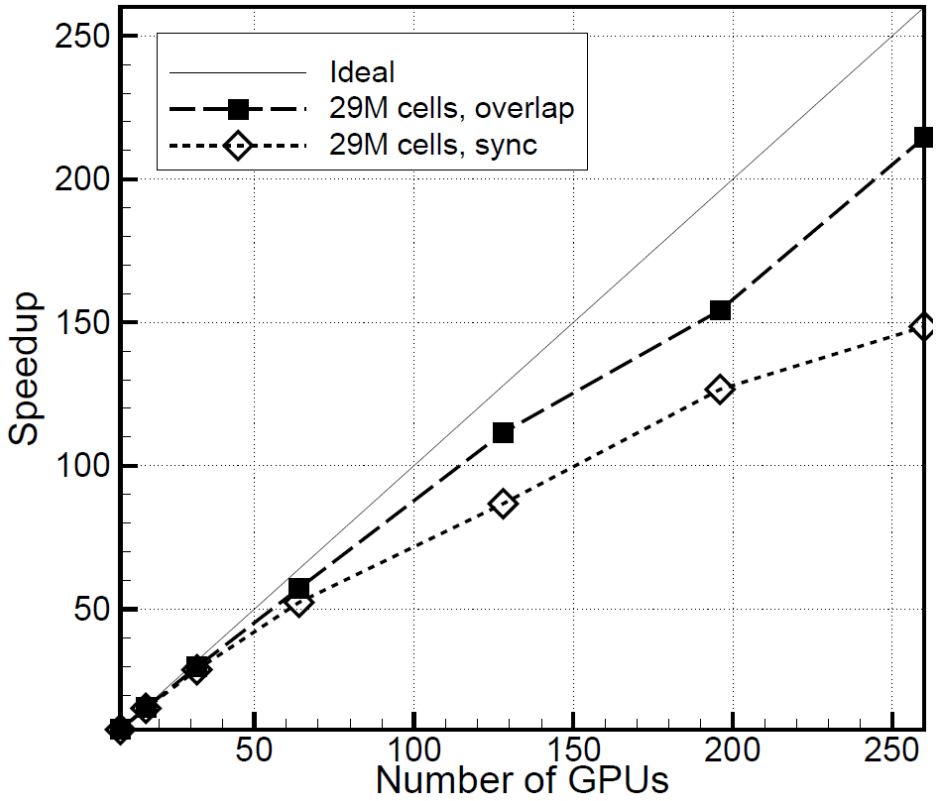


Ломоносов,160 млн узлов, каверна

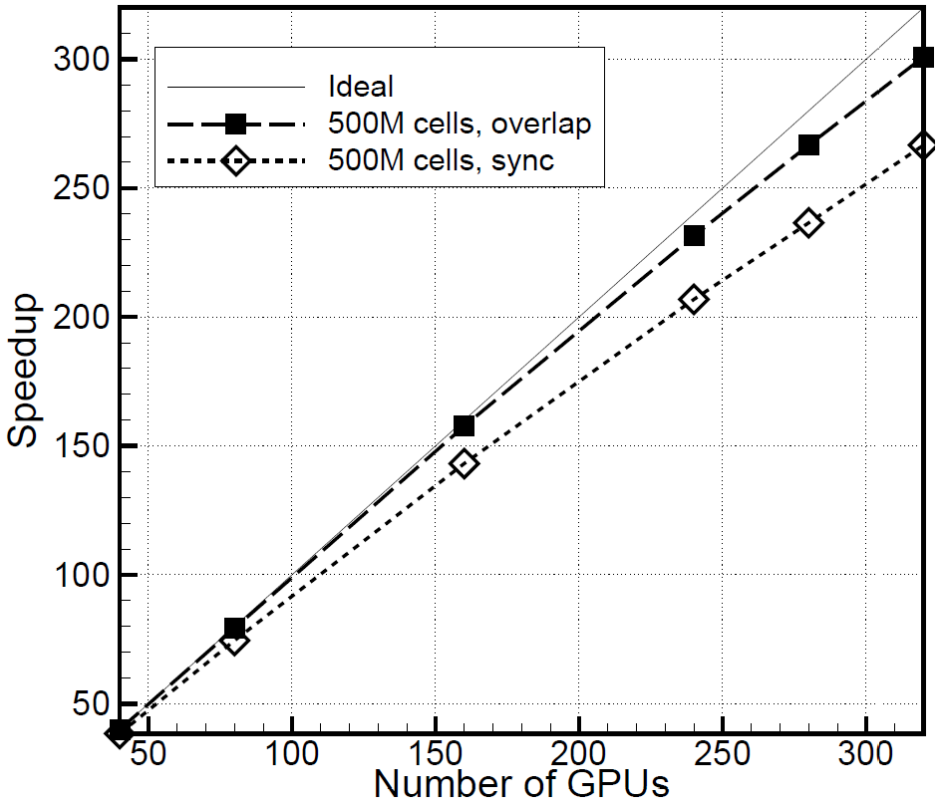


Мульти-GPU на гибридном суперкомпьютере

Ломоносов-2 (МГУ):
14C Xeon E5-2697v3
NVIDIA K40, IB FDR

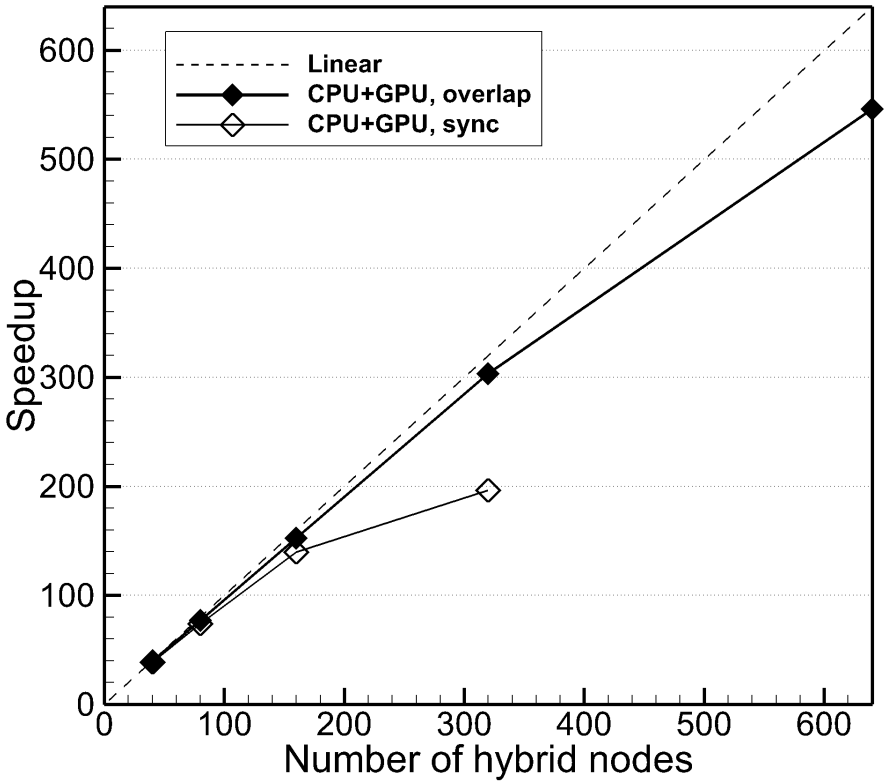


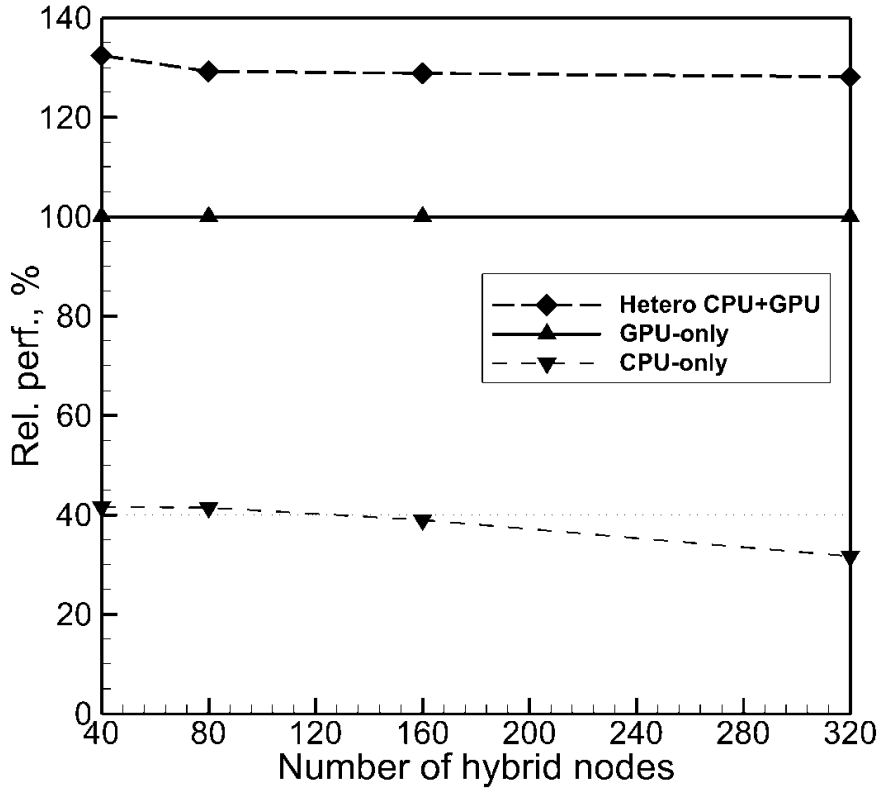
HPC5 (Курчатовский институт):
2x 8C Xeon E5-2650v2,
2x NVIDIA K80, IB FDR



MPI+OpenMP+OpenCL

Ломоносов-2: 14c Intel Xeon E5 v3 + GPU NVIDIA K40M



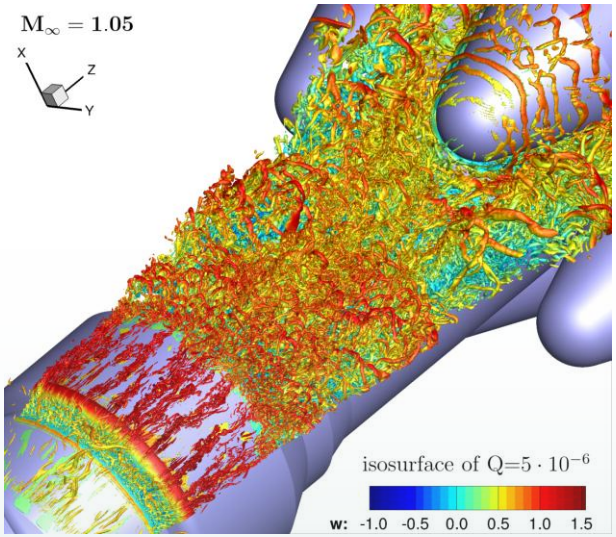


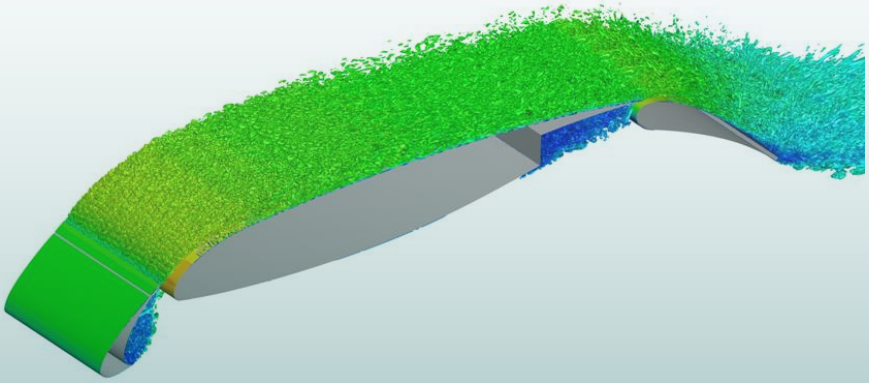
Сетка 500M ячеек

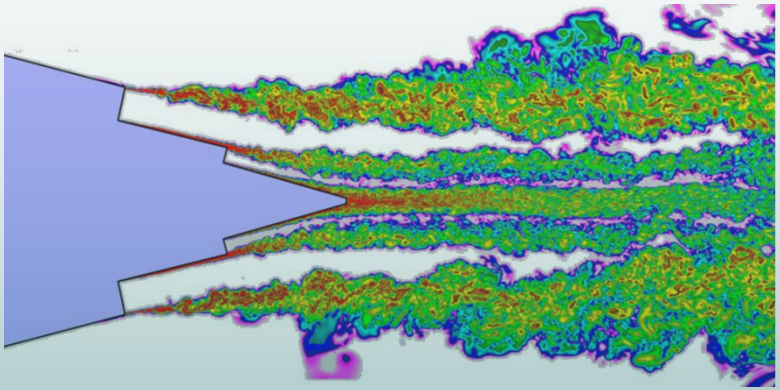
A.Gorobets, S.Soukov, P.Bogdanov. Multilevel parallelization for simulating turbulent flows on most kinds of hybrid supercomputers. Computers&Fluids. (2018) 173:171. https://doi.org/10.1016/j.compfluid.2018.03.011

Суперкомпьютерное моделирование турбулентности: авиационно-космические приложения





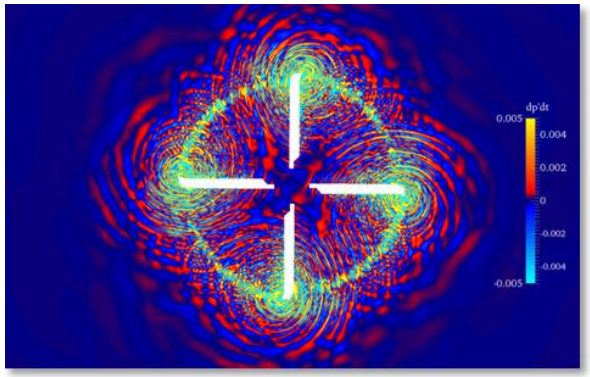


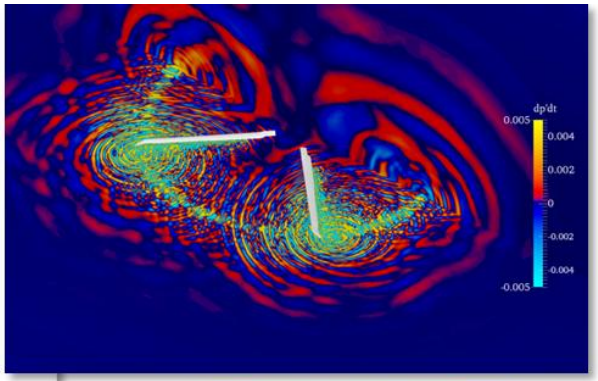


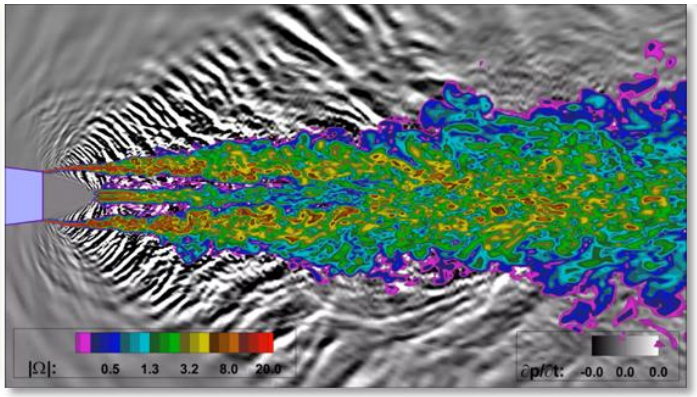
Андрей Горобец
ИПМ им. М. В. Келдыша РАН, сектор вычислительной аэродинамики и аэроакустики
http://caa.imamod.ru

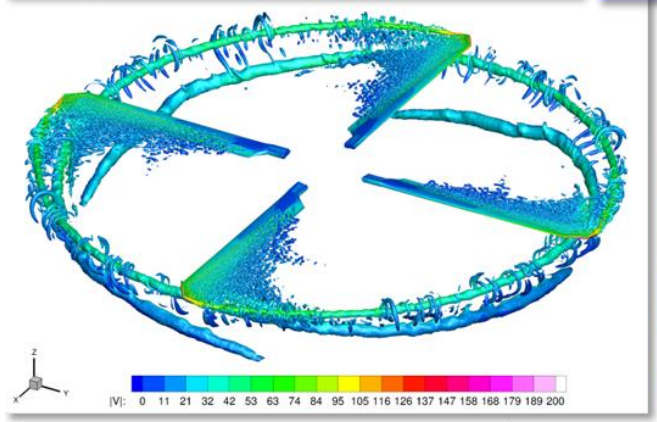


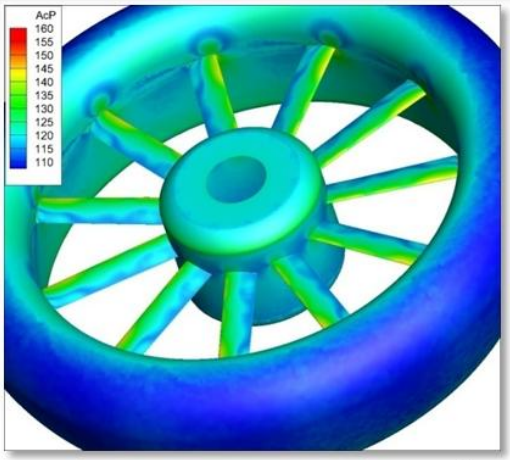
Предсказательное моделирование в CFD и CAA приложениях

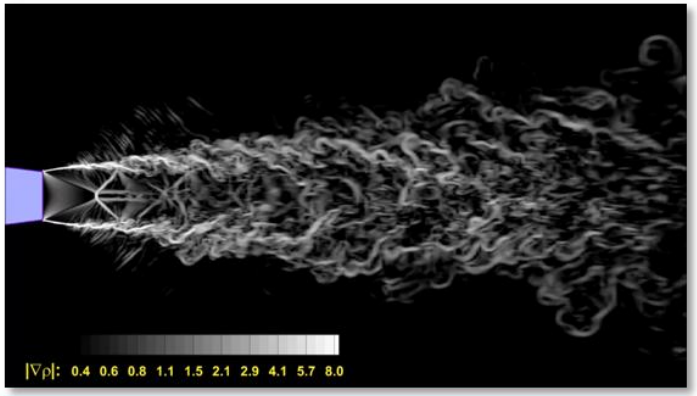


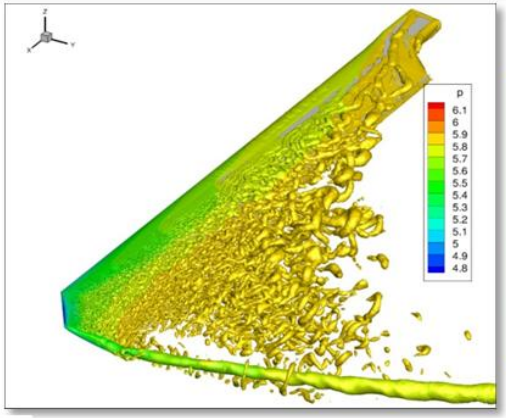


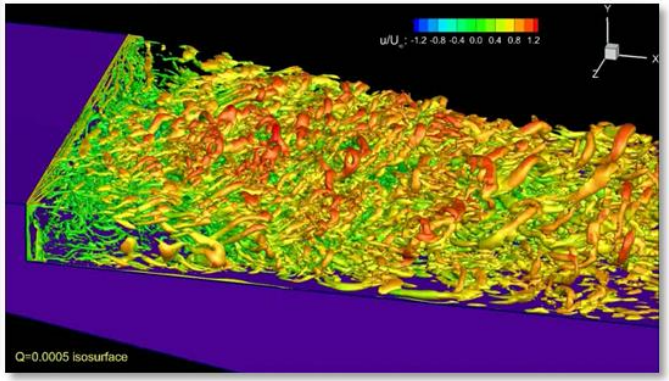


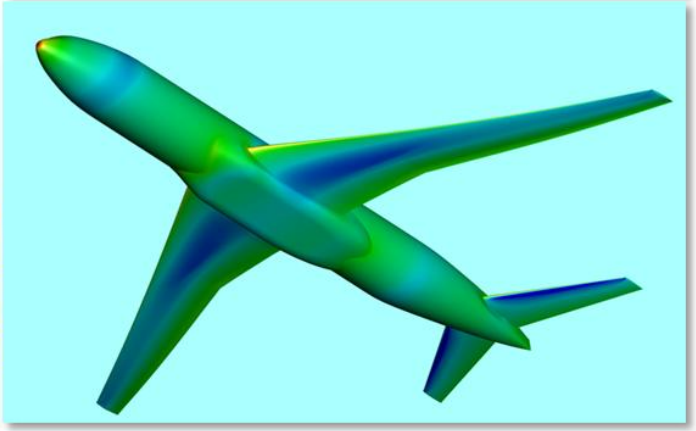












Предсказательное суперкомпьютерное моделирование

- **Лучше, если можно предсказать вообще без вычислений**
- **Если нет, то лучше, если можно посчитать на калькуляторе**
- **Если нет, то лучше, если можно посчитать десктопе**
- **Если нет, то лучше, если можно посчитать на серваке**
- **Если нет, то лучше, если можно посчитать на маленьком кластере**

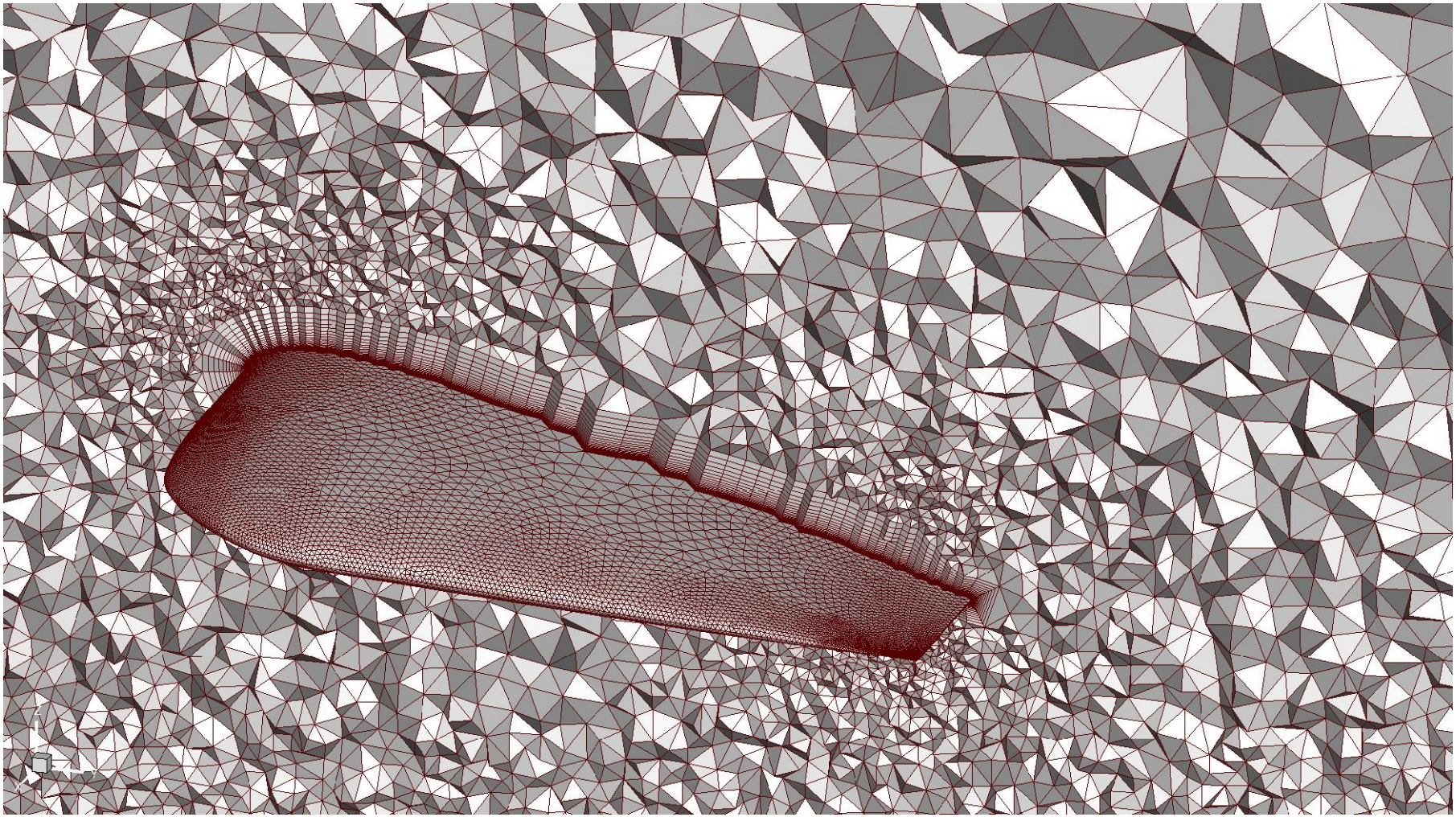
…

и если уже никак по-другому не получается, то только

от отчаяния и безысходности

- **мы станем считать на суперкомпьютере**

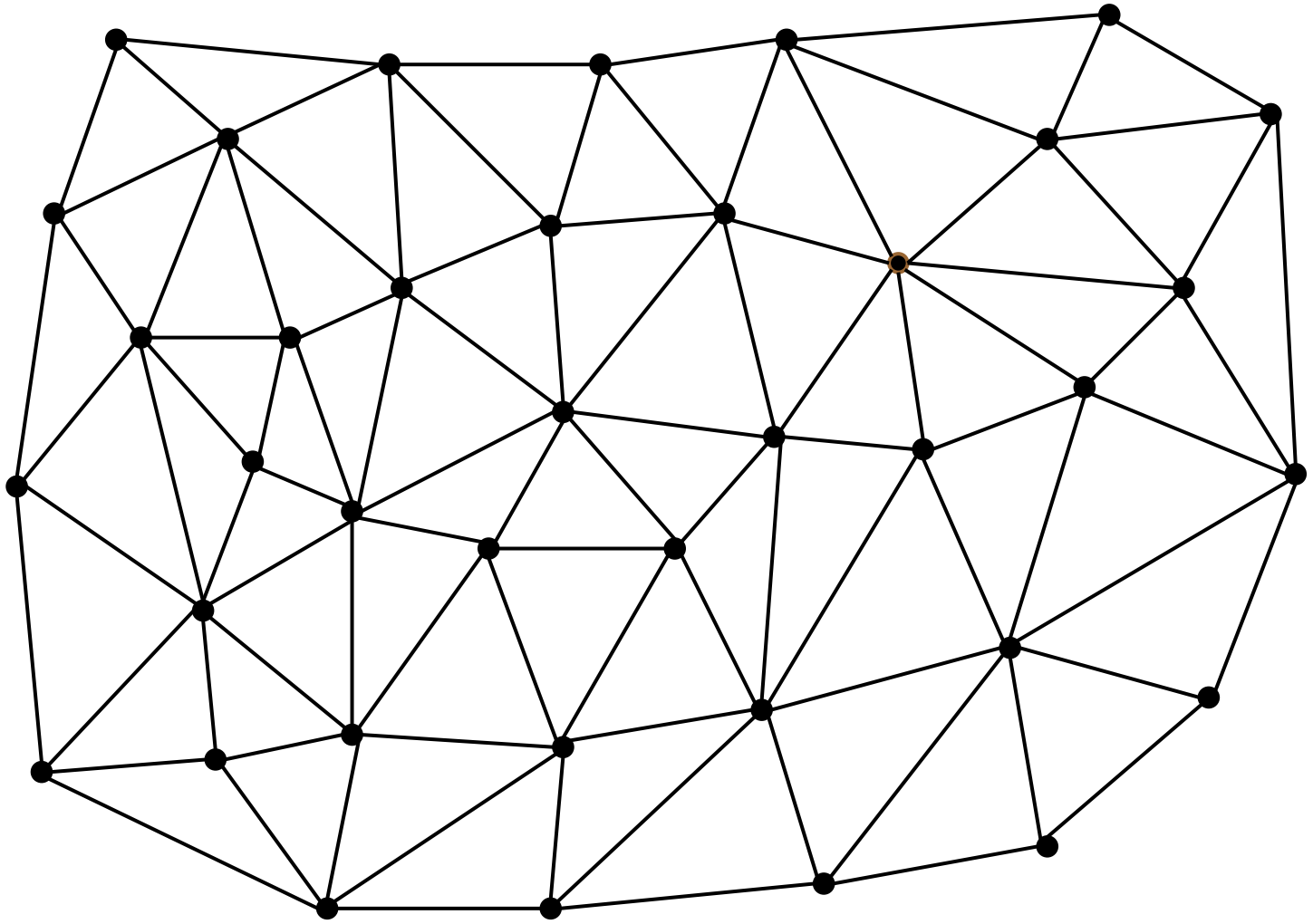
Как это работает…



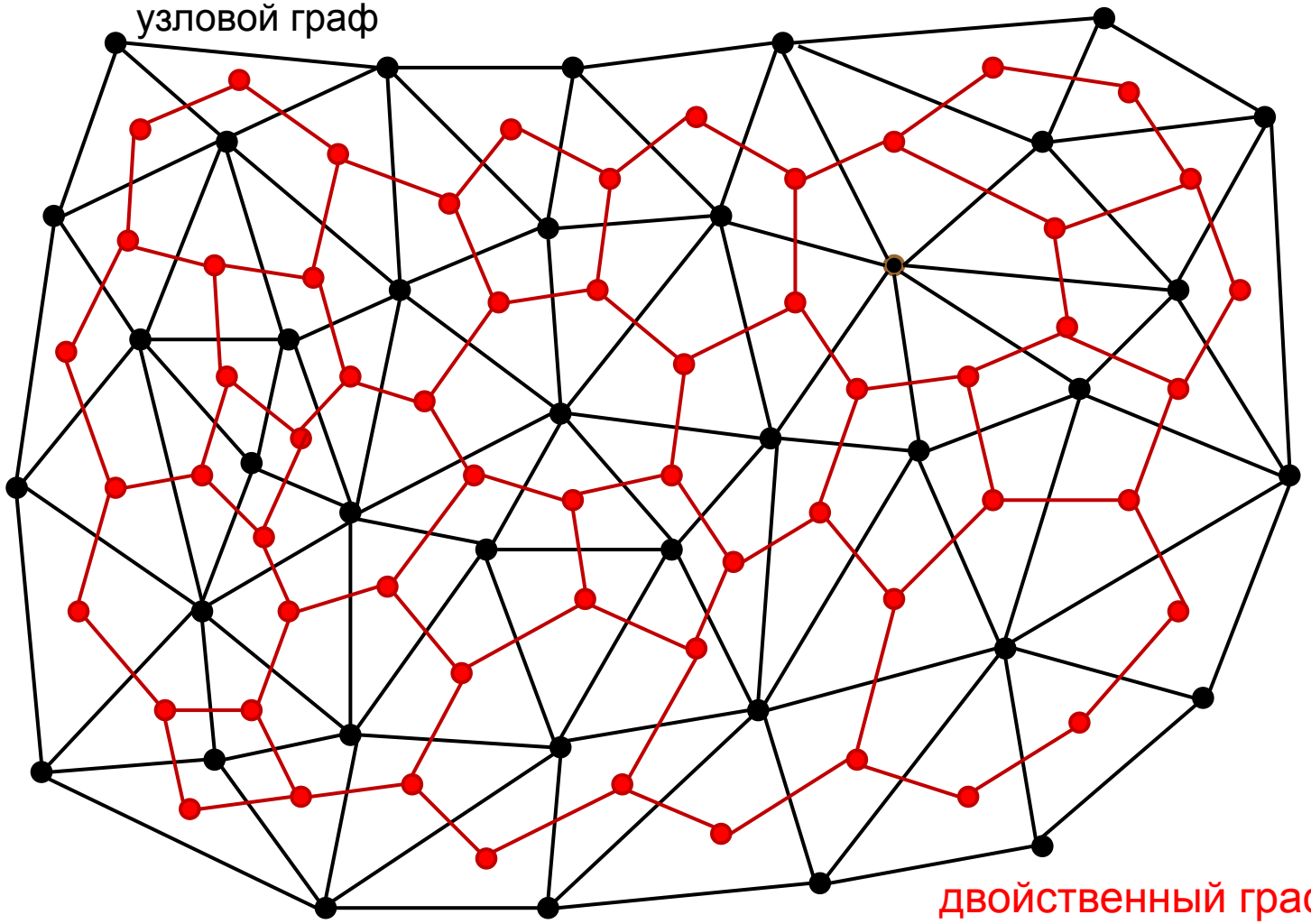
Строим пространственную сетку в расчетной области

Вычислительная газовая динамика – CFD

мы говорим сетка,
подразумеваем –
граф

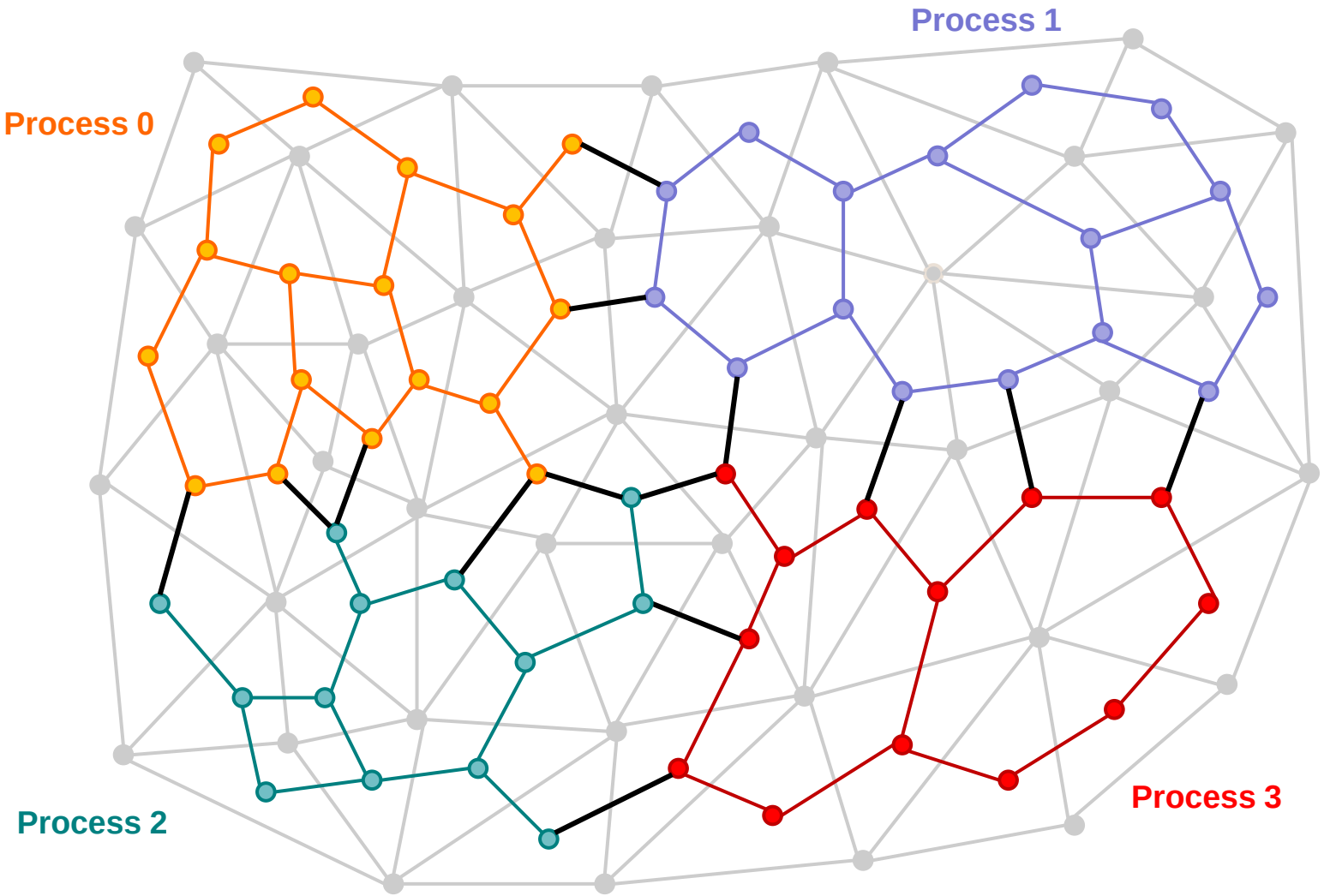


мы говорим сетка,
подразумеваем –
граф



Вычислительная газовая динамика – CFD

мы говорим сетка,
подразумеваем –
граф

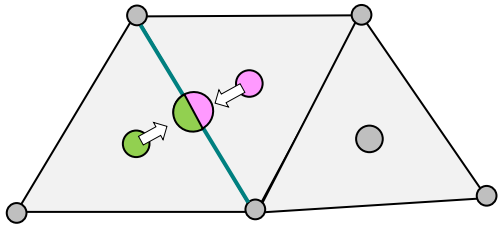


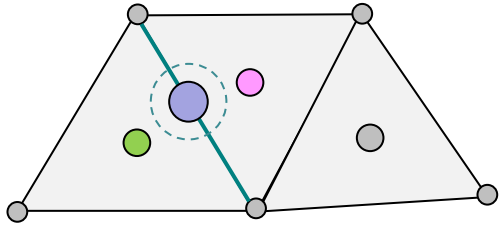
Массивно-параллельная обработка сеточных данных
(узлы, грани, сеточные элементы, …)

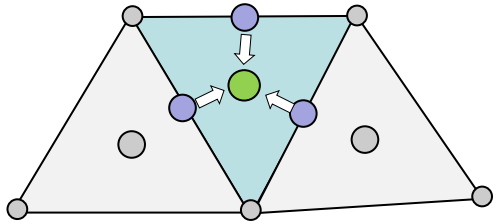
Реконструкция по значениям в ячейках значений на гранях

Расчет потока через грани
решение задачи Римана о распаде разрыва

Переход на новый временной слой
суммирование потоков через грани в ячейки







Предсказательное суперкомпьютерное моделирование

- **Лучше, если можно предсказать вообще без вычислений**
- **Если нет, то лучше, если можно посчитать на калькуляторе**
- **Если нет, то лучше, если можно посчитать десктопе**
- **Если нет, то лучше, если можно посчитать на серваке**
- **Если нет, то лучше, если можно посчитать на маленьком кластере**

…

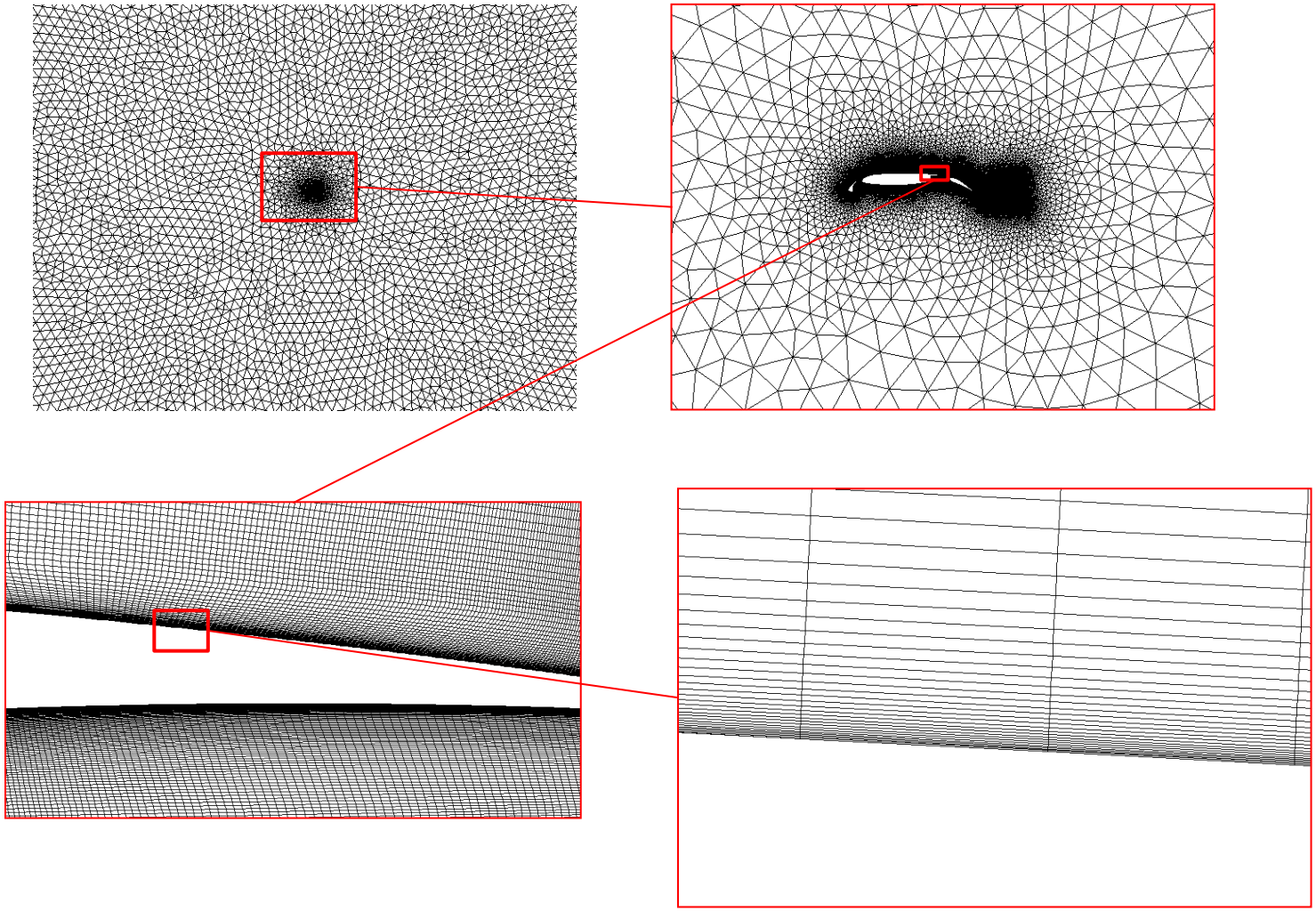
и если уже никак по-другому не получается, то только

от отчаяния и безысходности

- **мы станем считать на суперкомпьютере**

Снижение ресурсоемкости

- **Рациональное распределение сеточного разрешения**

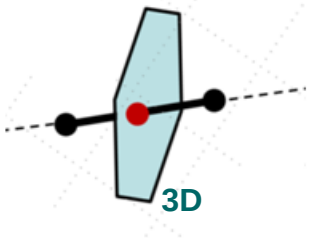


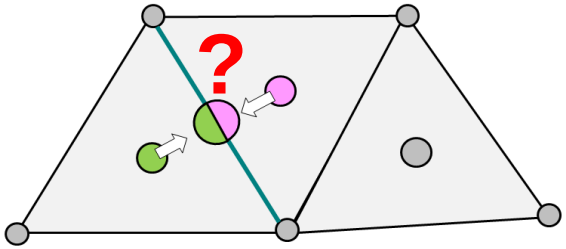
Снижение ресурсоемкости

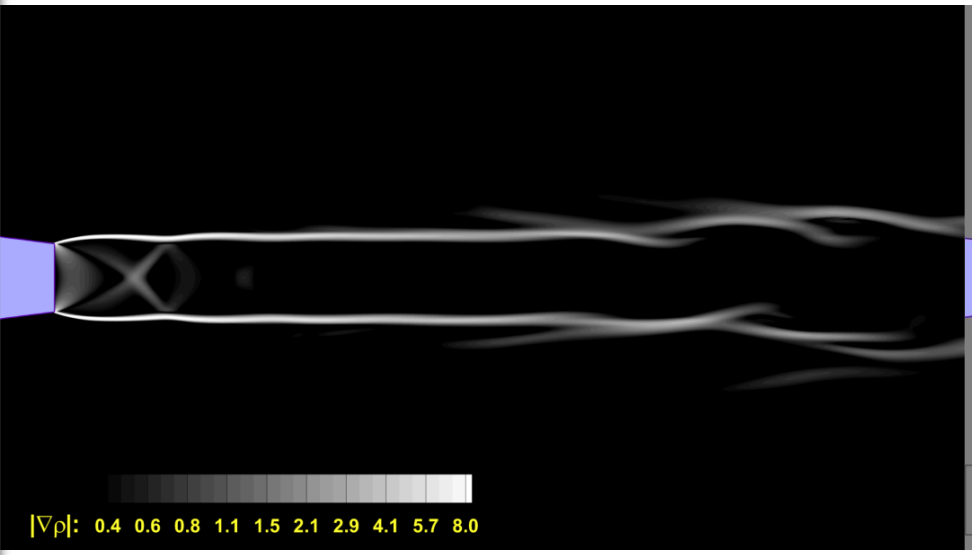
- **Схемы повышенной точности**

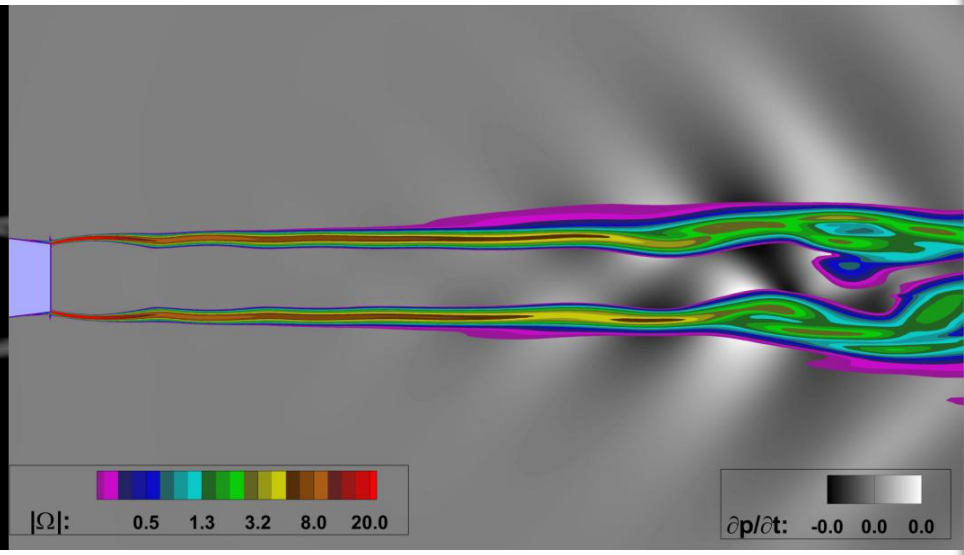
Пример: моделирование недорасширенной горячей струи

Базовая схема 1 порядка







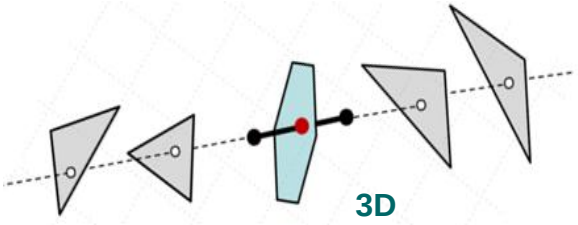


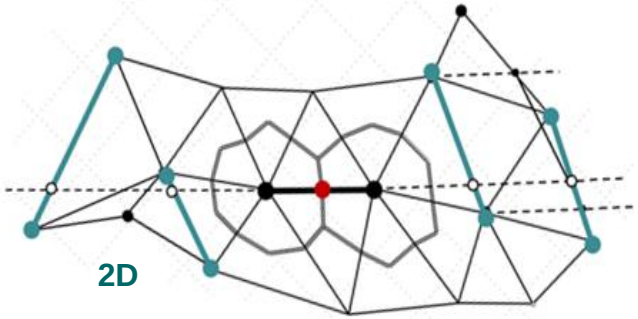
Снижение ресурсоемкости

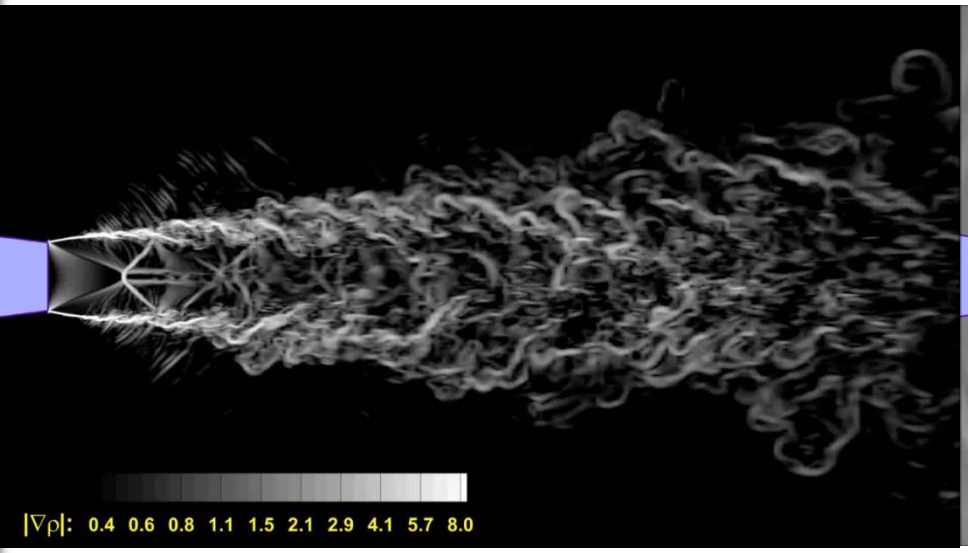
- **Схемы повышенной точности**

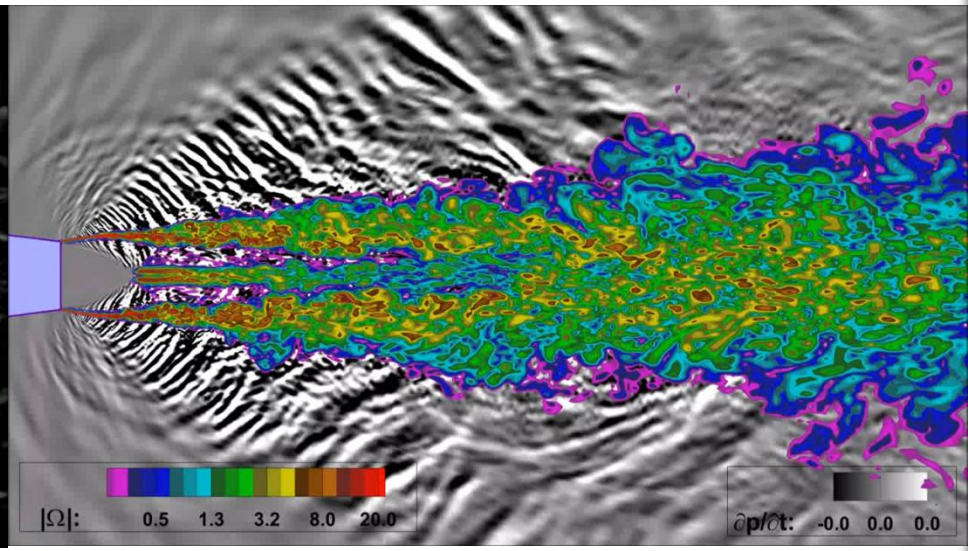
Пример: моделирование недорасширенной горячей струи

Схема EBR5





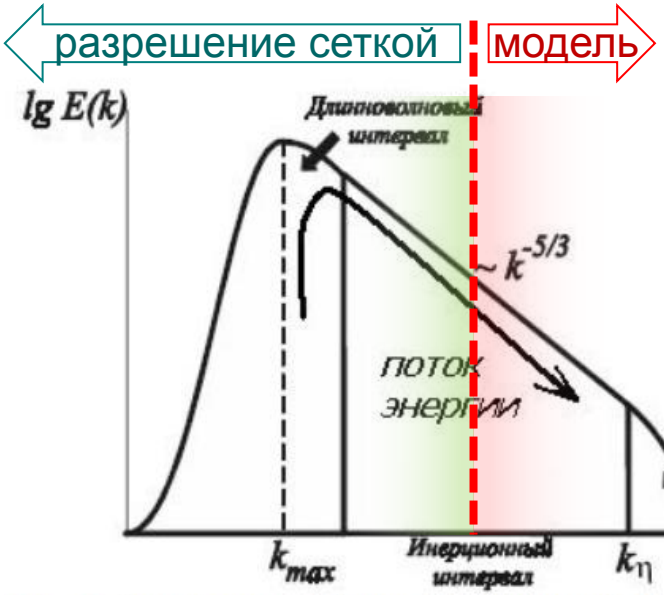




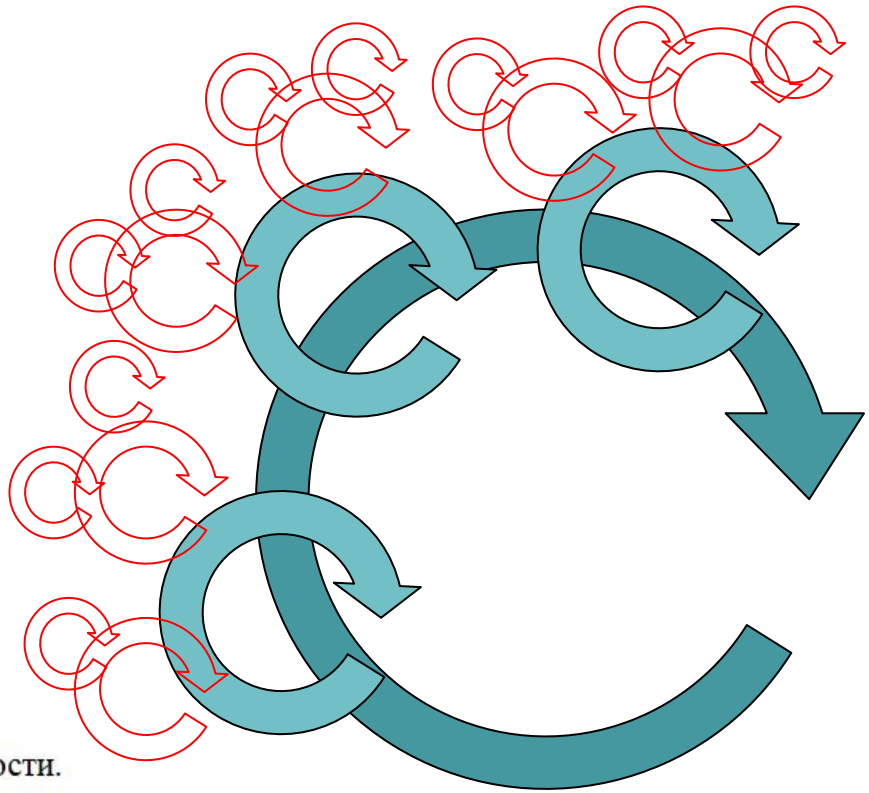
Снижение ресурсоемкости

- **Моделирование турбулентности**

Метод крупных вихрей LES
Метод отсоединенных вихрей DES



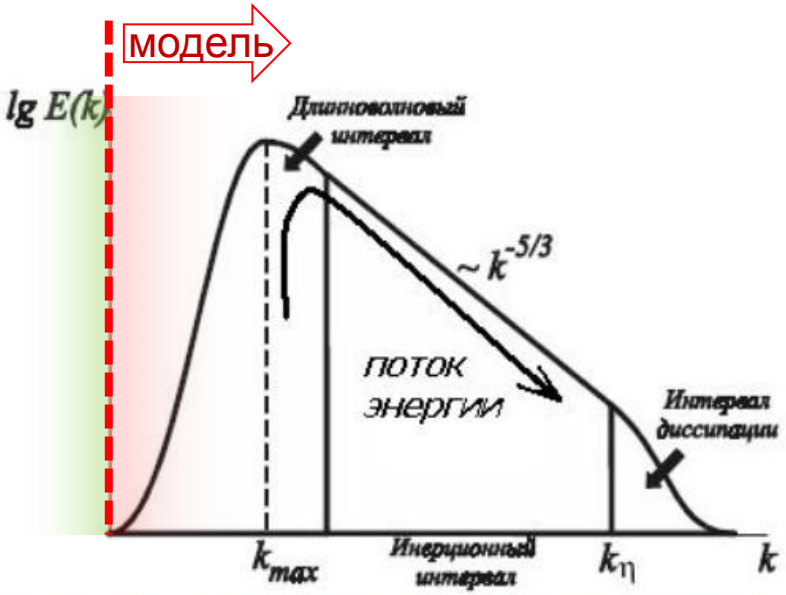
Энергетический спектр изотропной трехмерной турбулентности.



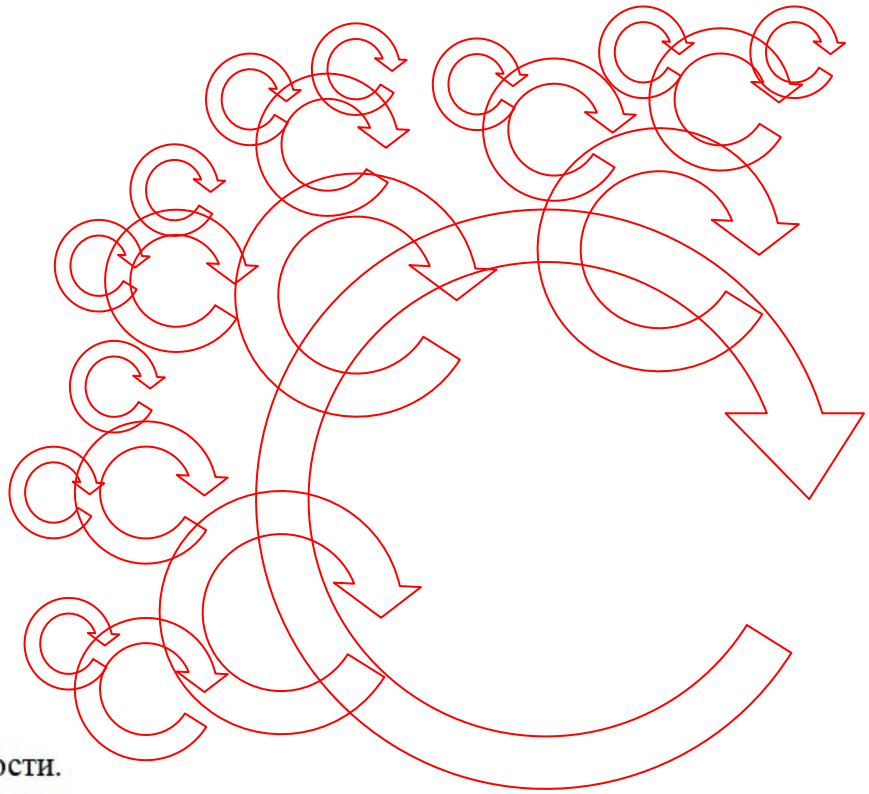
Снижение ресурсоемкости

- **Моделирование турбулентности**

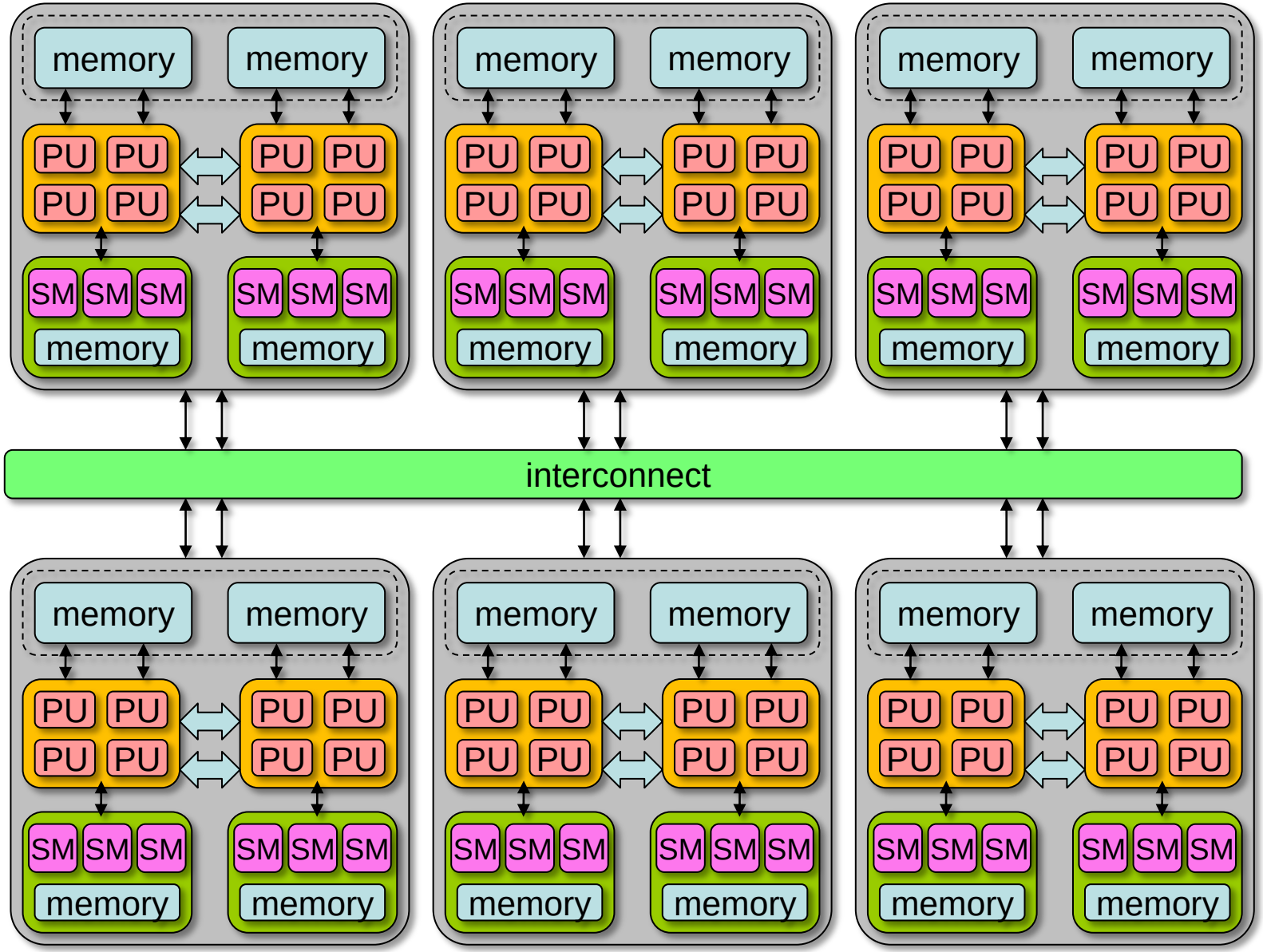
Метод RANS – осредненные по Рейнольдсу уравнения Навье – Стокса



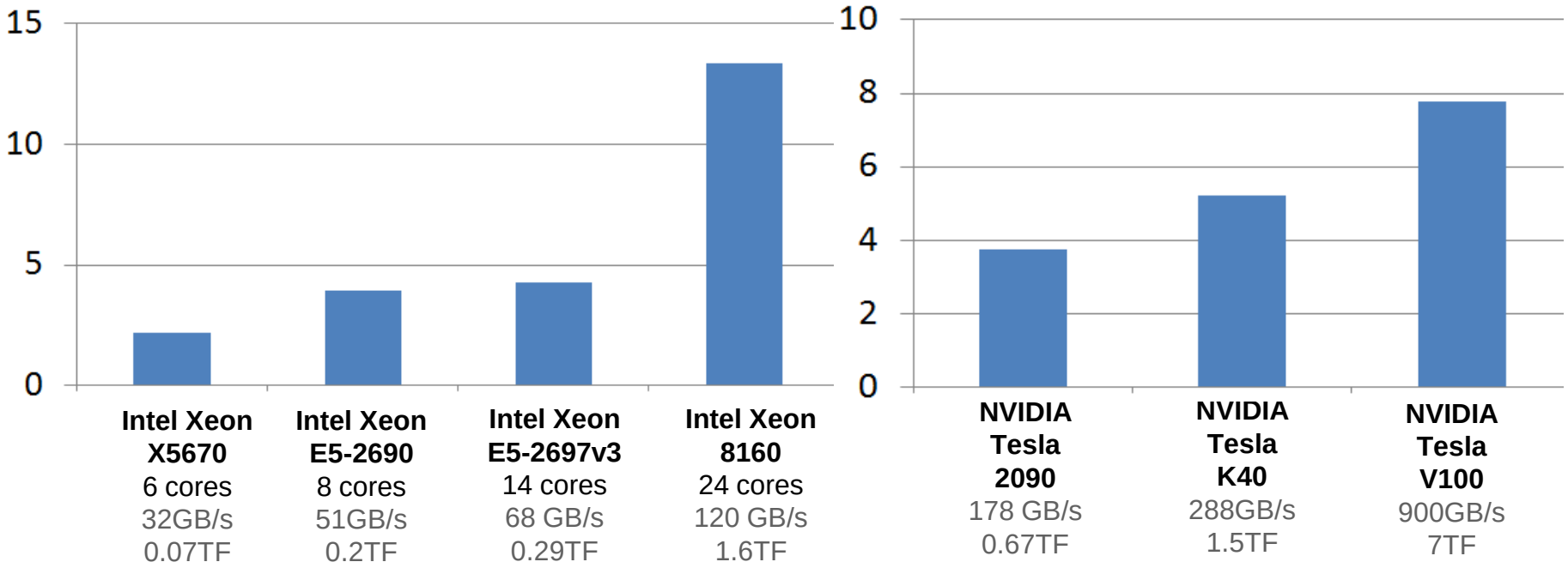
Энергетический спектр изотропной трехмерной турбулентности.



Высокопроизводительные вычисления – HPC



Разрыв между производительностью и пропускной способностью



Отношение пиковой производительности
и пропускной способности памяти

Уровни параллелизма и средства разработки

DM MIMD

MPI

Суперкомпьютер

Гибридный узел

SM MIMD

OpenMP
OpenCL
MPI

Многоядерный CPU

Многопоточное ядро

SIMD

OpenMP, OpenCL,
директивы компилятора, …

Out-of-order, VLIW

SIMD расширение

ILP параллелизм

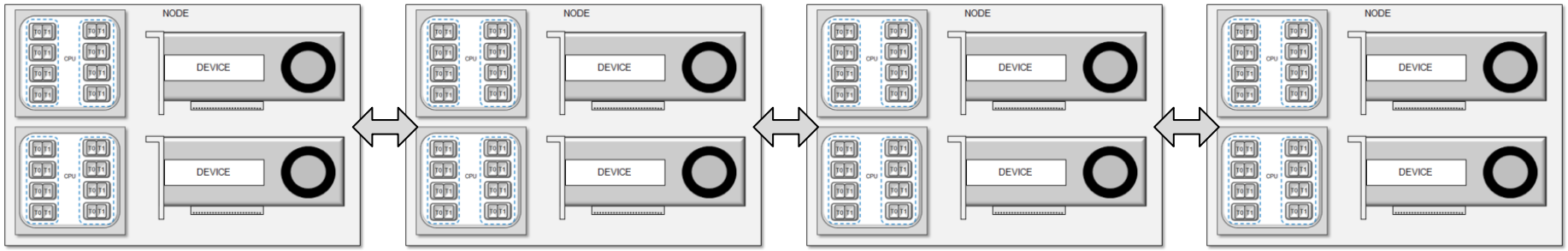
GPU ускоритель

Потоковый мультипроцессор

OpenCL

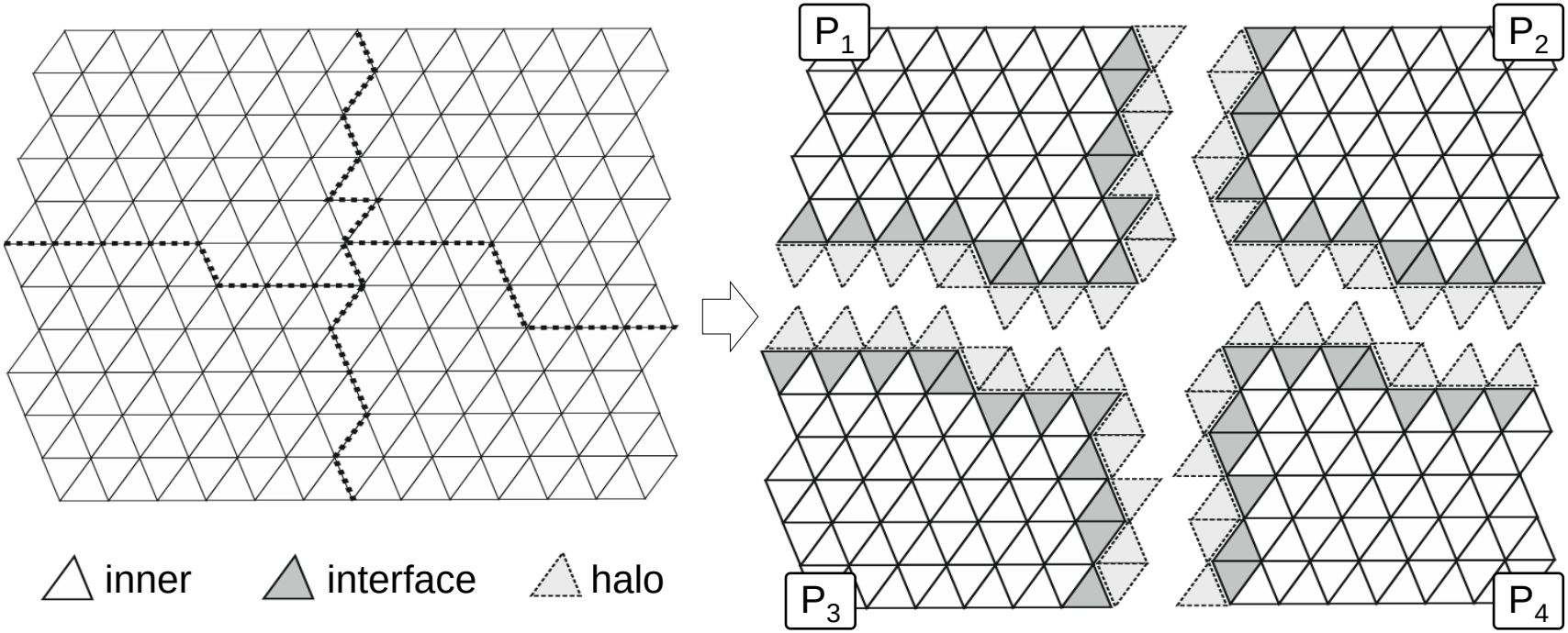
CUDA

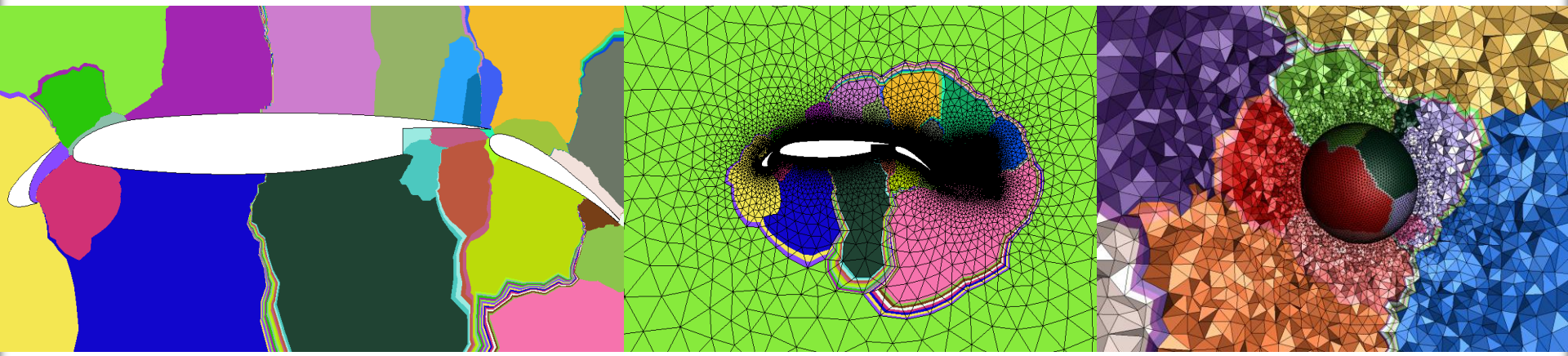
Stream Processing



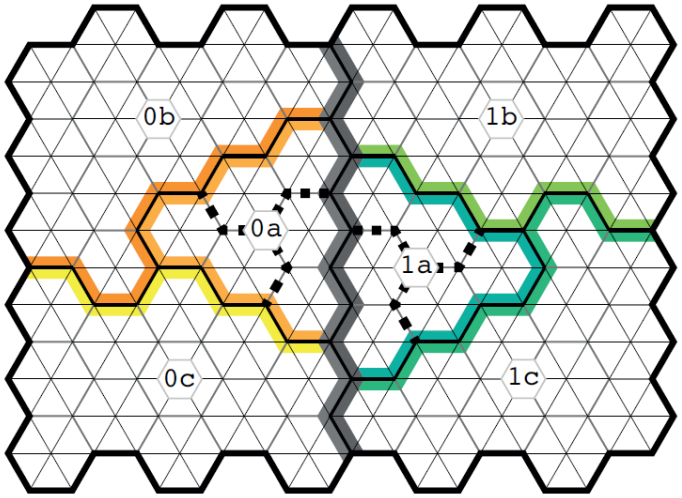
Многоуровневое распараллеливание

MPI – распараллеливание с распределенной памятью DM-MIMD

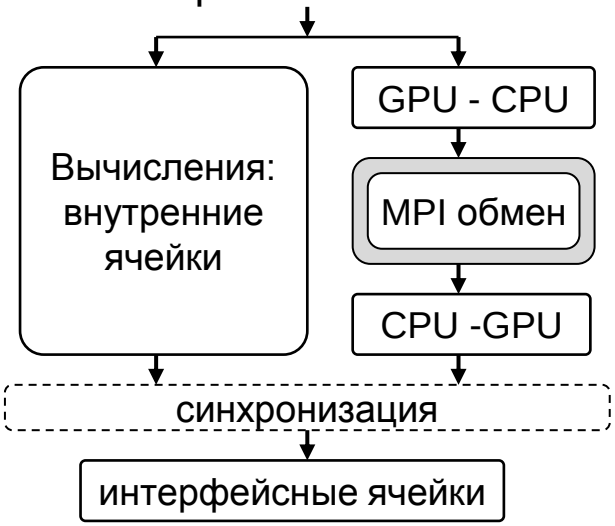


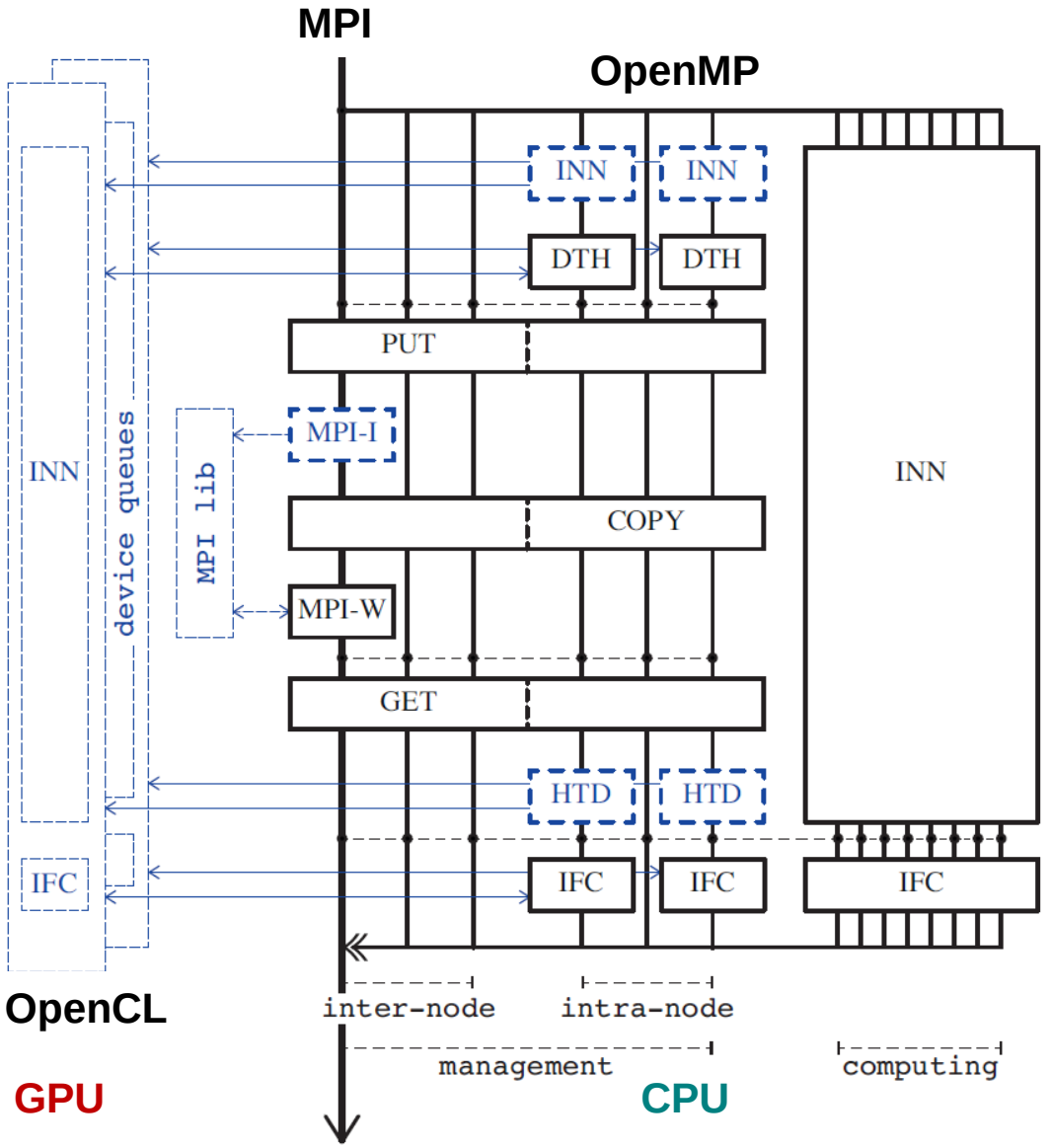


OpenCL – вычисления на графических процессорах GPU



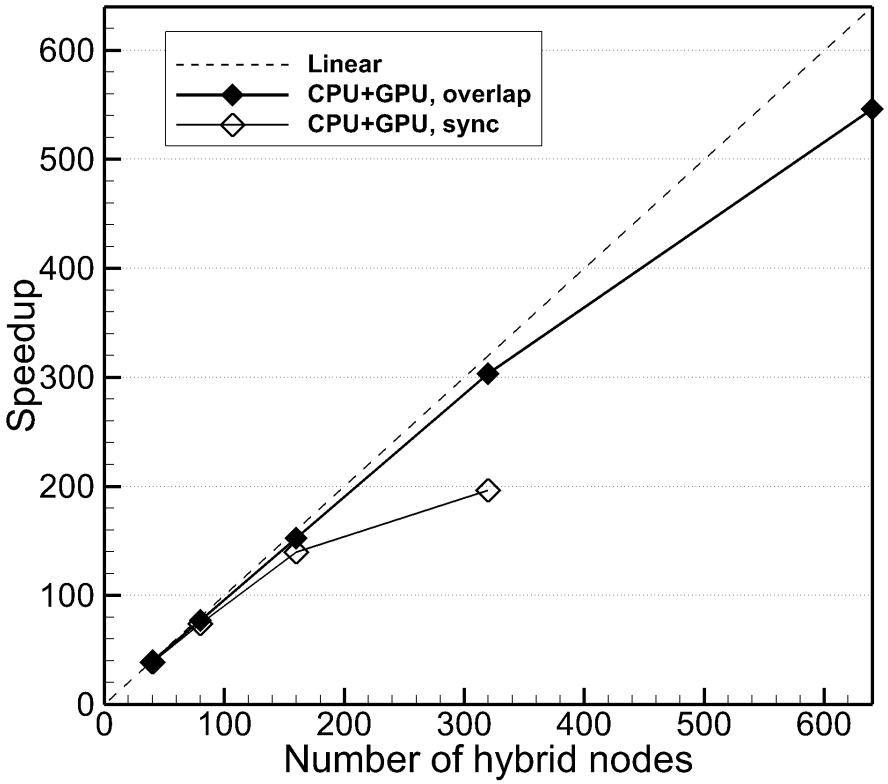
Overlap

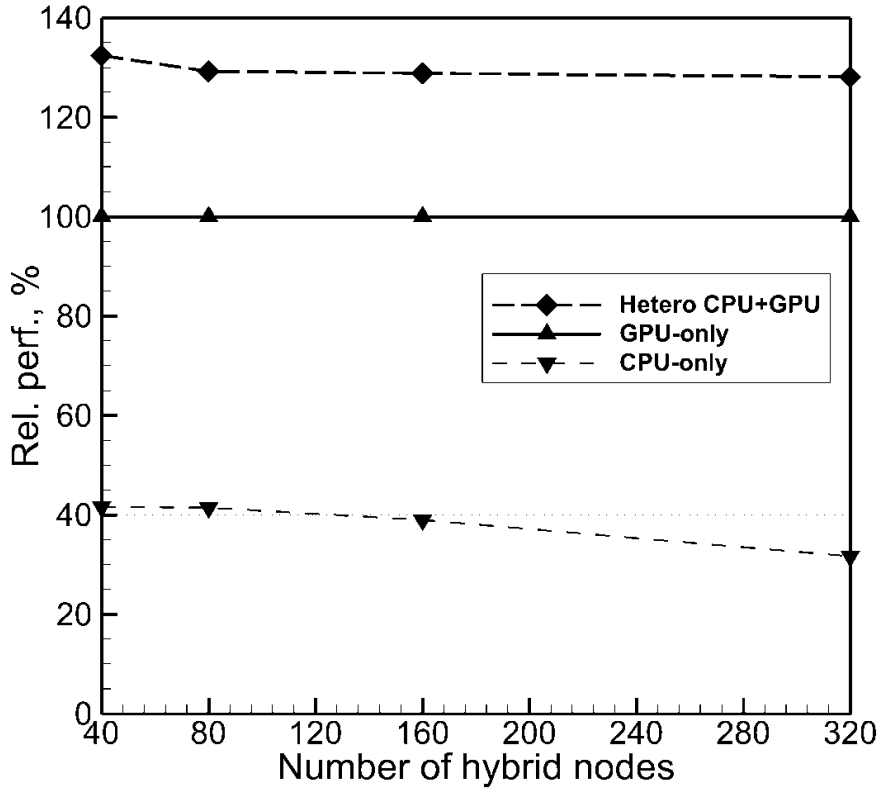




MPI+OpenMP+OpenCL

Ломоносов-2: 14c Intel Xeon E5 v3 + GPU NVIDIA K40M





Сетка 500M ячеек

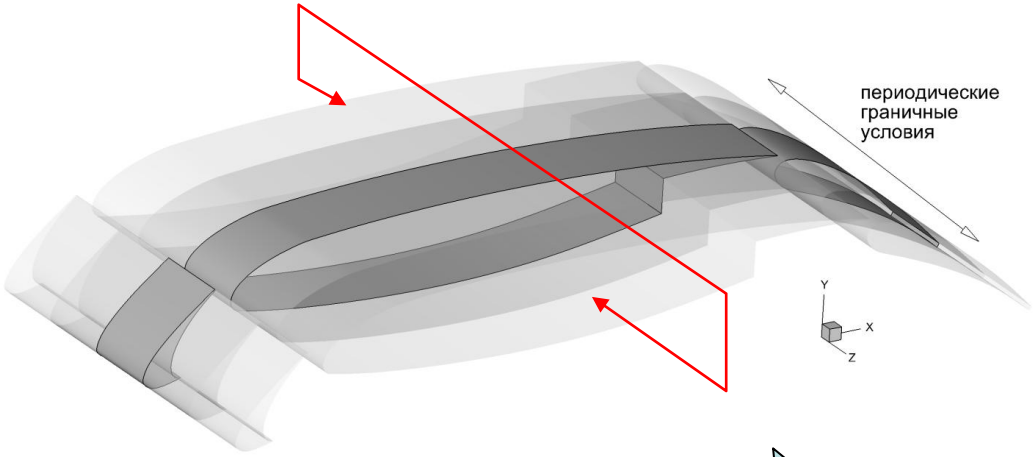
A.Gorobets, S.Soukov, P.Bogdanov. Multilevel parallelization for simulating turbulent flows on most kinds of hybrid supercomputers. Computers&Fluids. (2018) 173:171. https://doi.org/10.1016/j.compfluid.2018.03.011

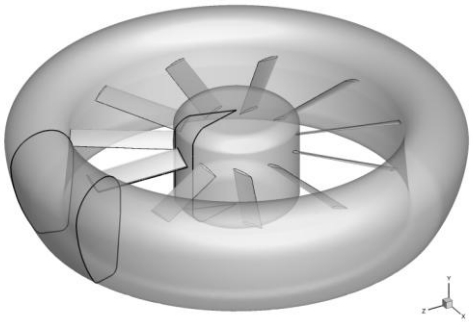
Схемы интегрирования по времени

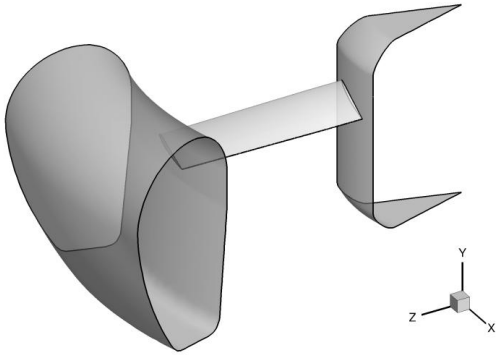
- **Неявные схемы – линеаризация по Ньютону**
  решение больших СЛАУ с блочной разреженной матрицей
- **Методы локального шага по времени**
- **Многосеточные методы – FAS MG**
- **Методы на основе LU-SGS**

- **Динамическая конфигурация метода**
  управление расчетом, автовыбор параметров численного метода
- **Runtime диагностика и анти-авост**
  автоконтроль корректности вычислений, коррекция без авоста
- **Смешанная точность – поля на FP64, геометрия, солвер СЛАУ на FP32**

- **Периодические граничные условия**



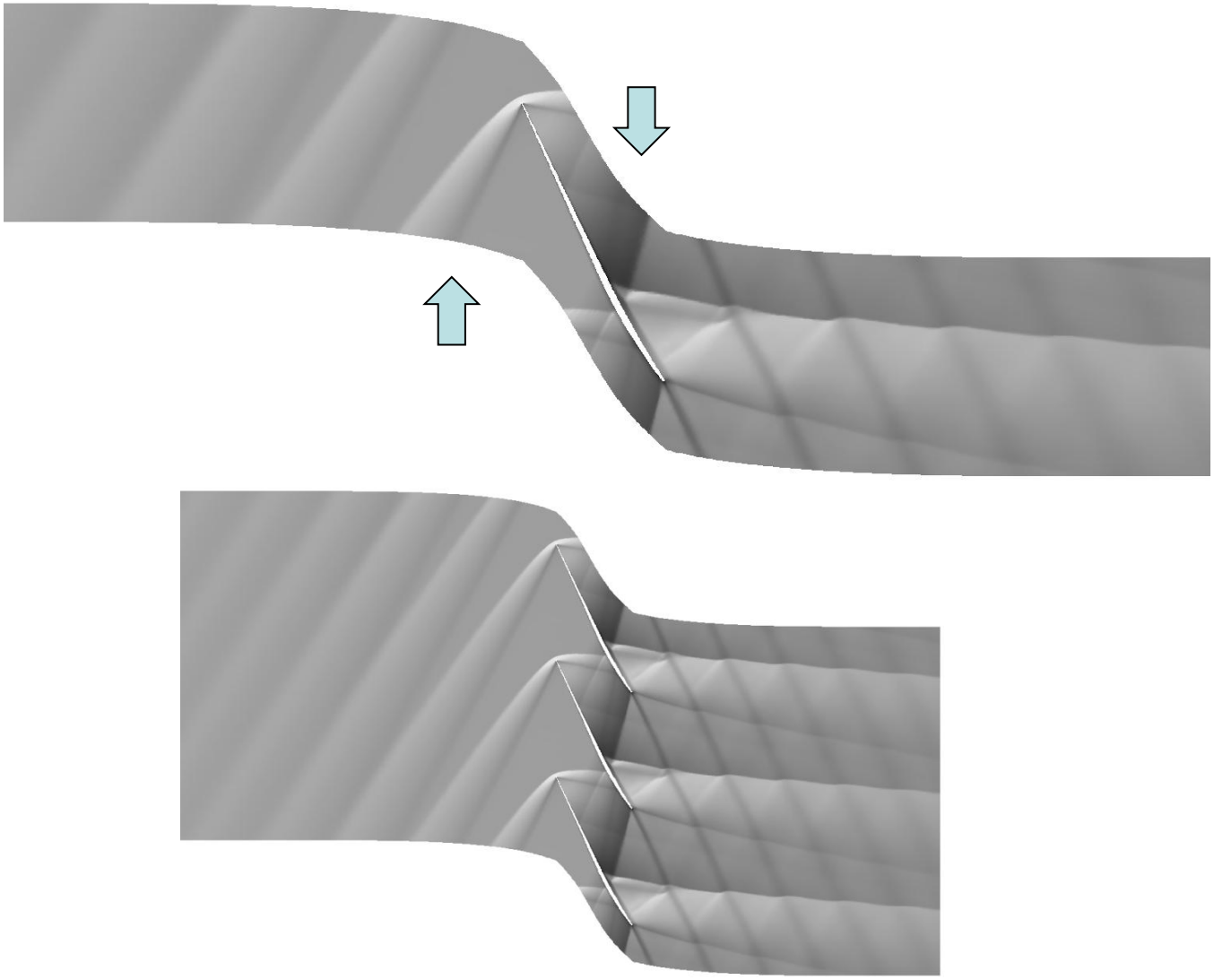




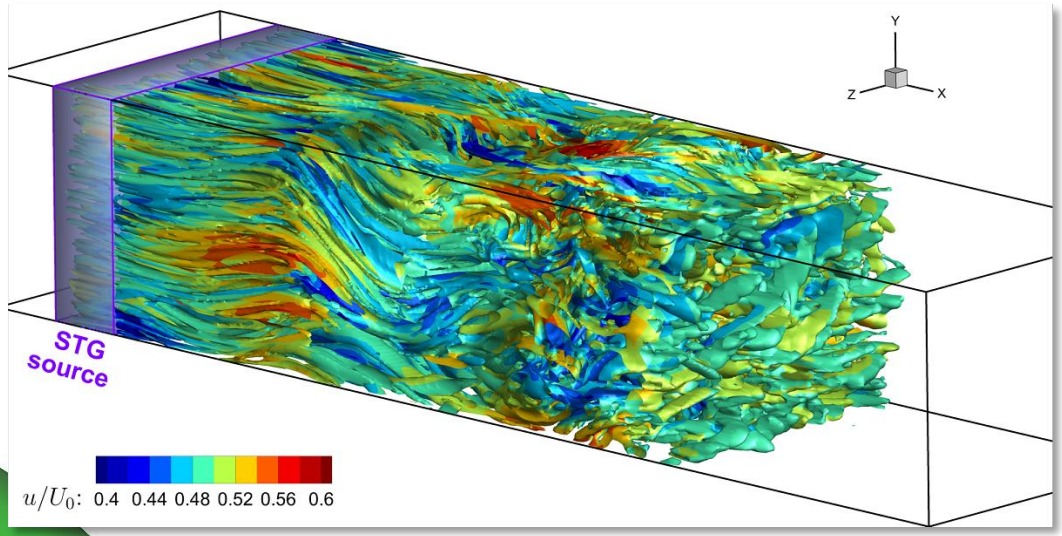
size matters

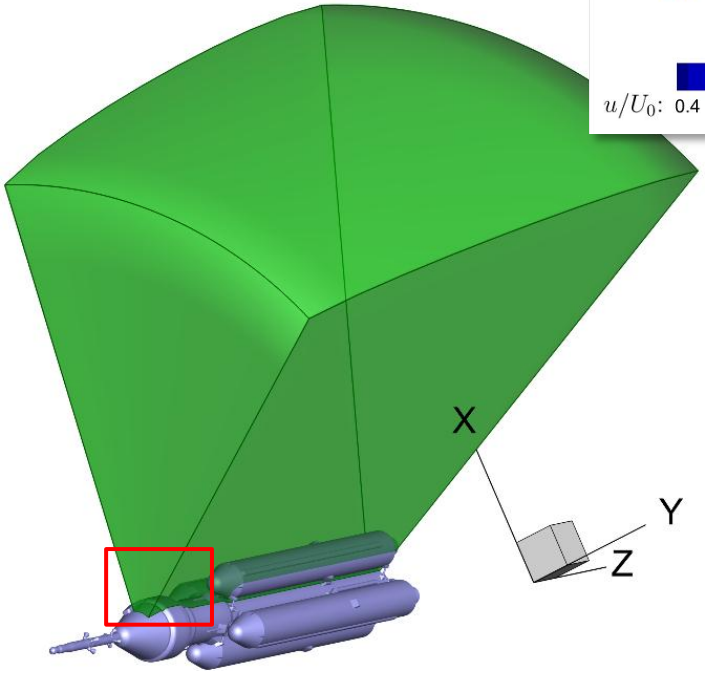


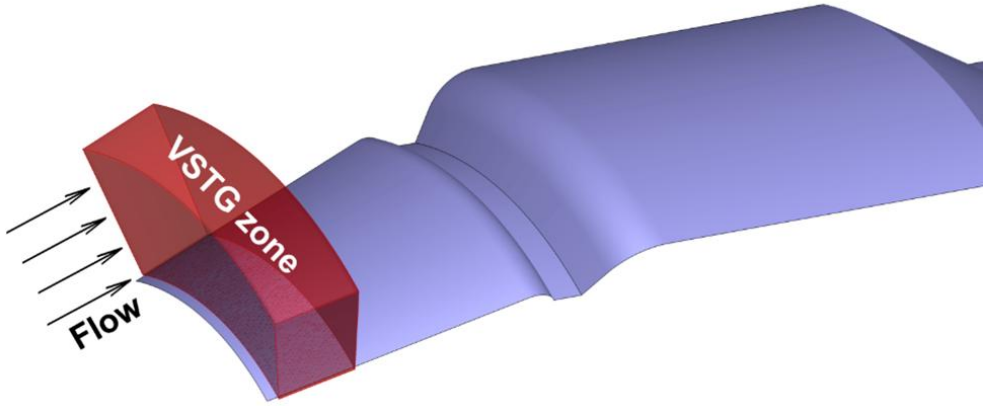
- Периодические граничные условия



- **Генерация синтетической турбулентности**
  моделирование фрагментов сложного объекта, RANS – LES интерфейс
- **Поглощающий слой**





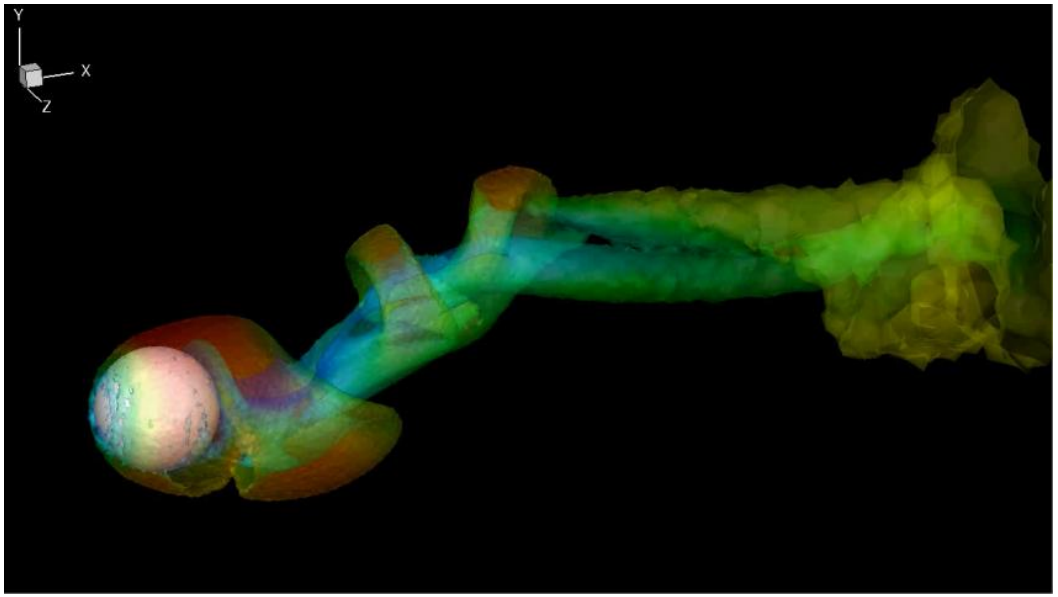


- **Погруженные граничные условия – IBC**

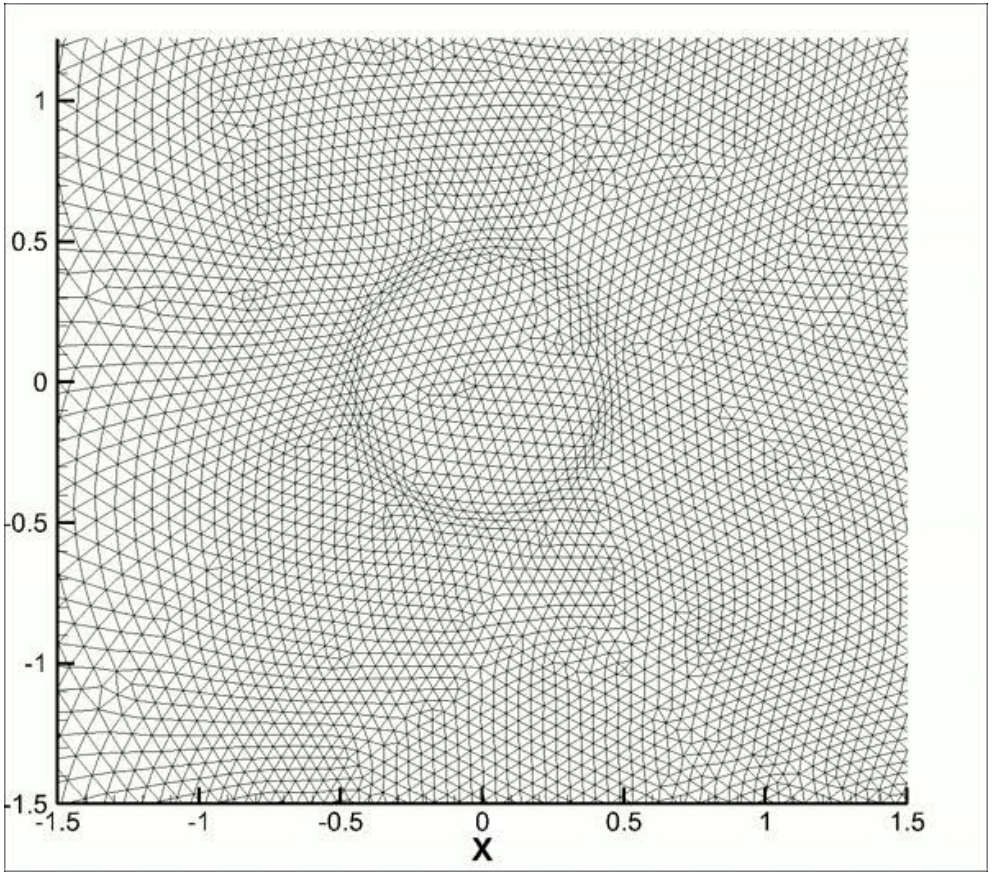
моделирование подвижных объектов

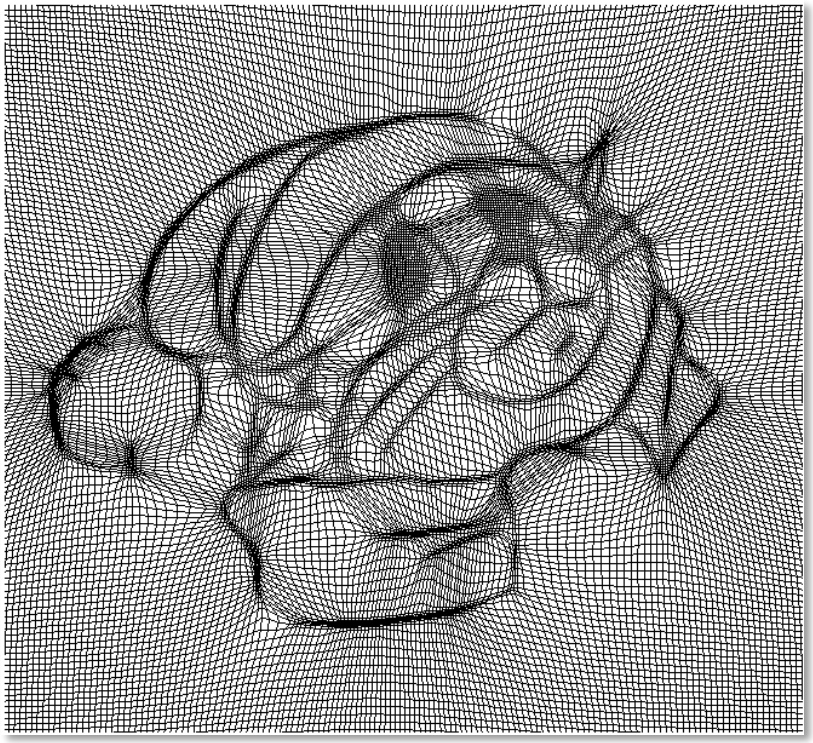
движение тел под действием сил, как внешних, так и индуцированных

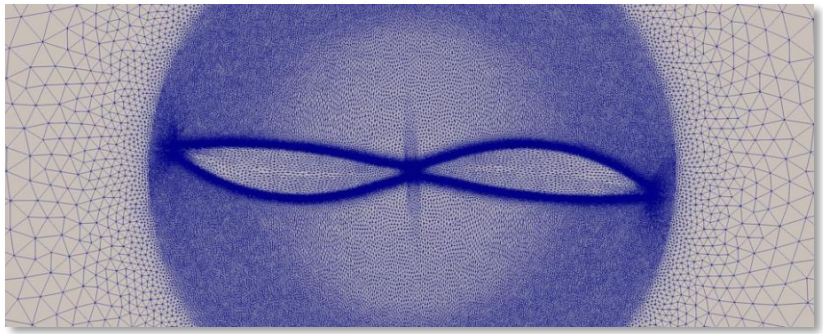
$$\begin{cases} \dfrac{\partial \mathbf{Q}_\eta}{\partial t} + \nabla \cdot \mathbf{F}\left(\mathbf{Q}_\eta\right) + \dfrac{1}{\eta}\chi \begin{pmatrix} 0 \\ \rho_\eta\left(\mathbf{u}_\eta - \mathbf{u}_{\Omega_o}\right) \\ E_\eta - E_{\Omega_o} \end{pmatrix} = \dfrac{1}{\mathrm{Re}} \nabla \cdot \mathbf{F}_v\left(\mathbf{Q}_\eta\right) \\ \mathbf{Q}_\eta\left(\mathbf{x}, t=0\right) = \mathbf{Q}_0\left(\mathbf{x}\right) \end{cases}$$



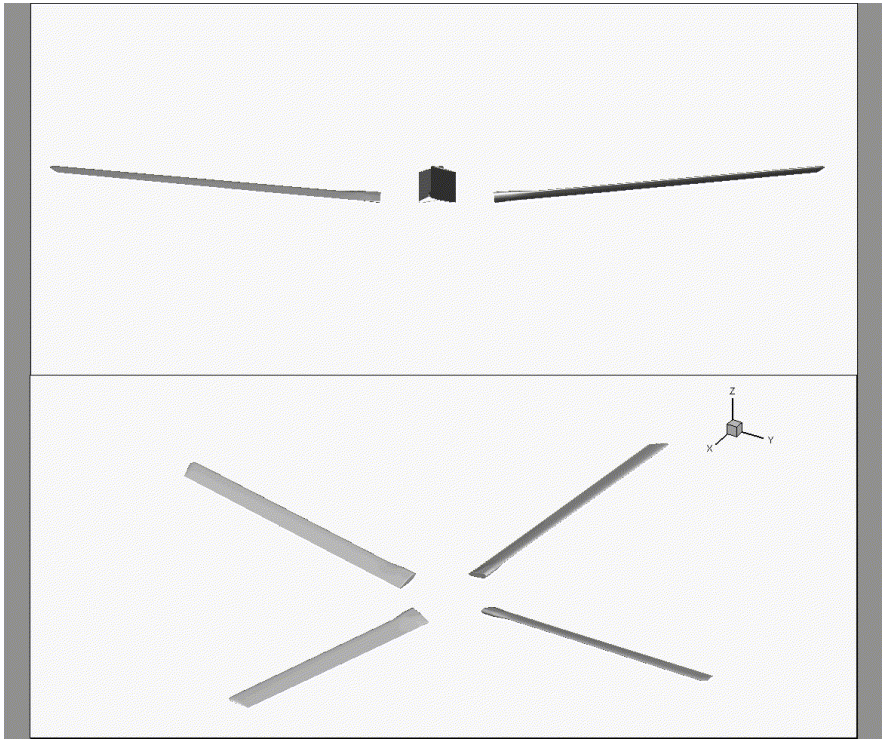
- **Динамическая адаптация сетки без изменения топологии**
  улучшение пространственного распределения узлов
  подвижные объекты IBC

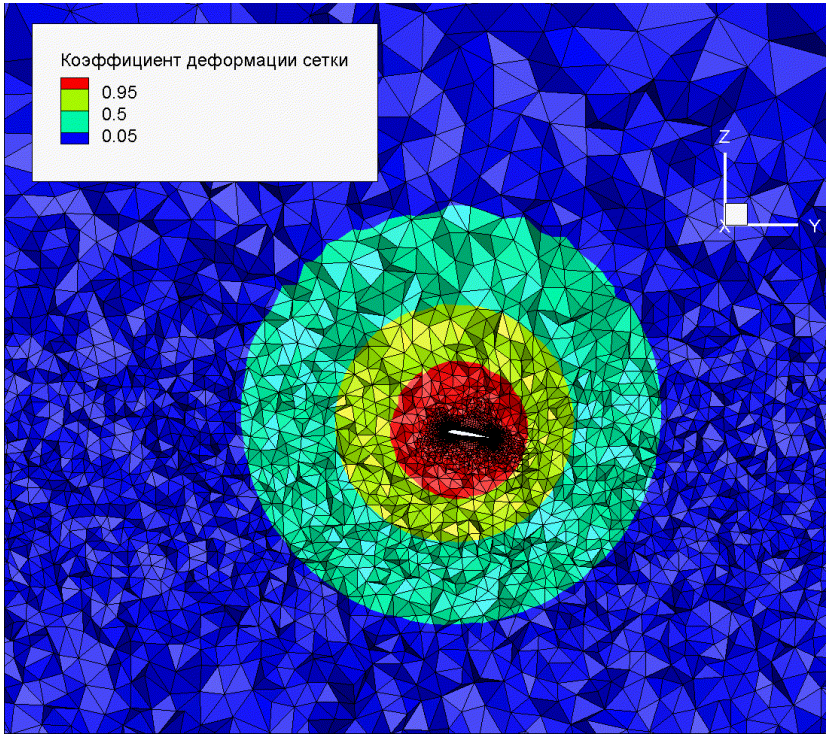




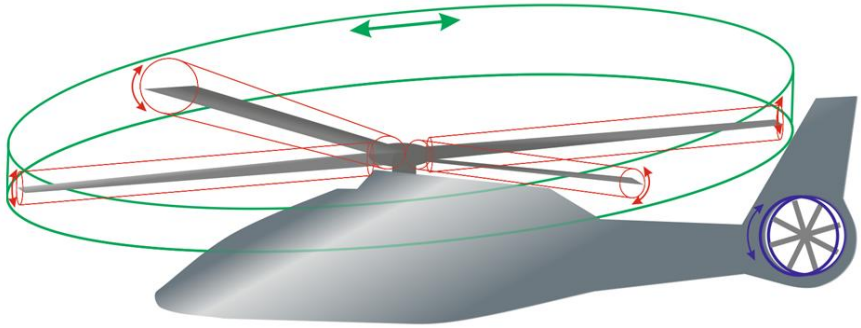


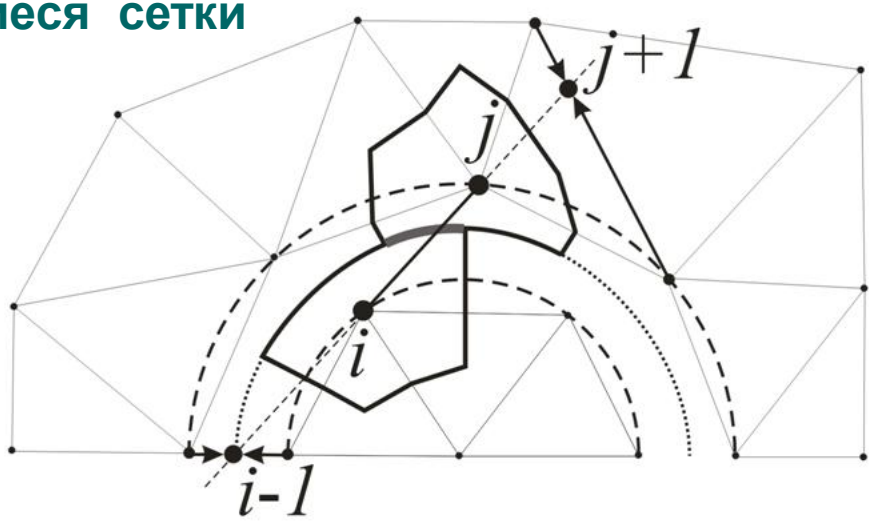
- **Деформируемые сетки**

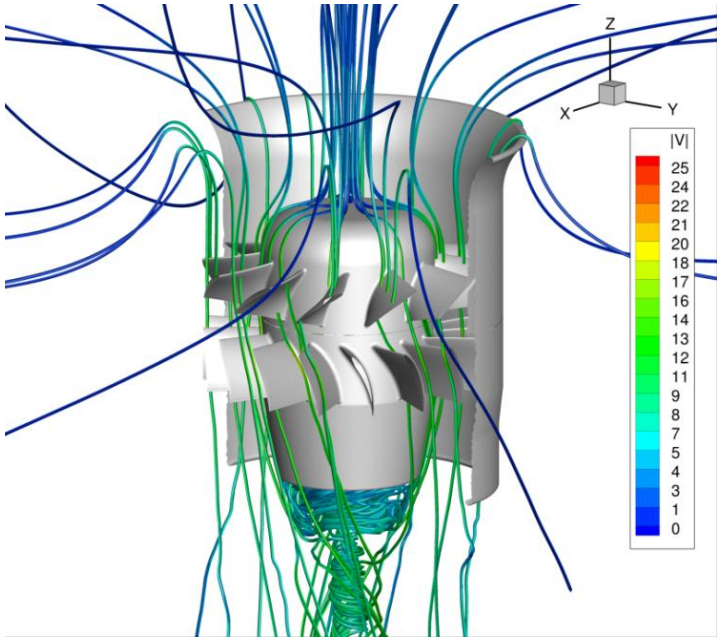


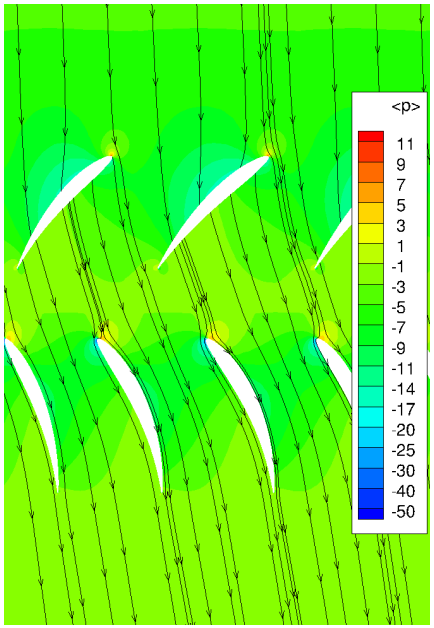


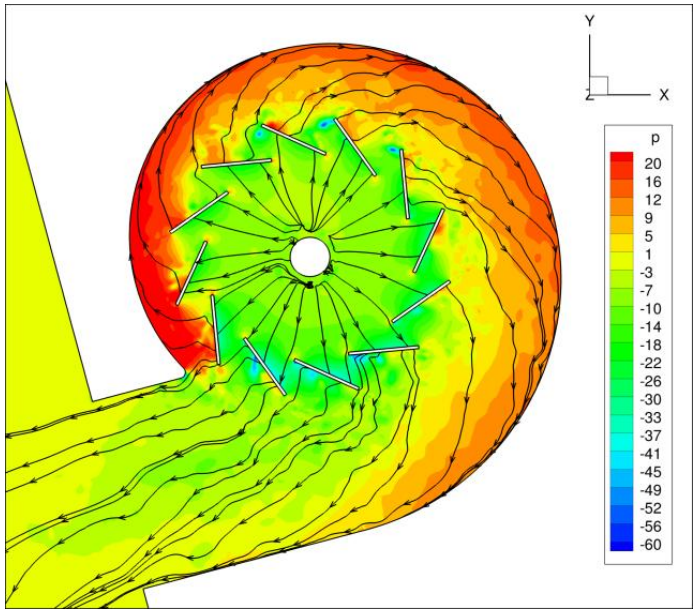
- **Скользящий интерфейс – вращающиеся сетки**
  ротор-статор взаимодействие





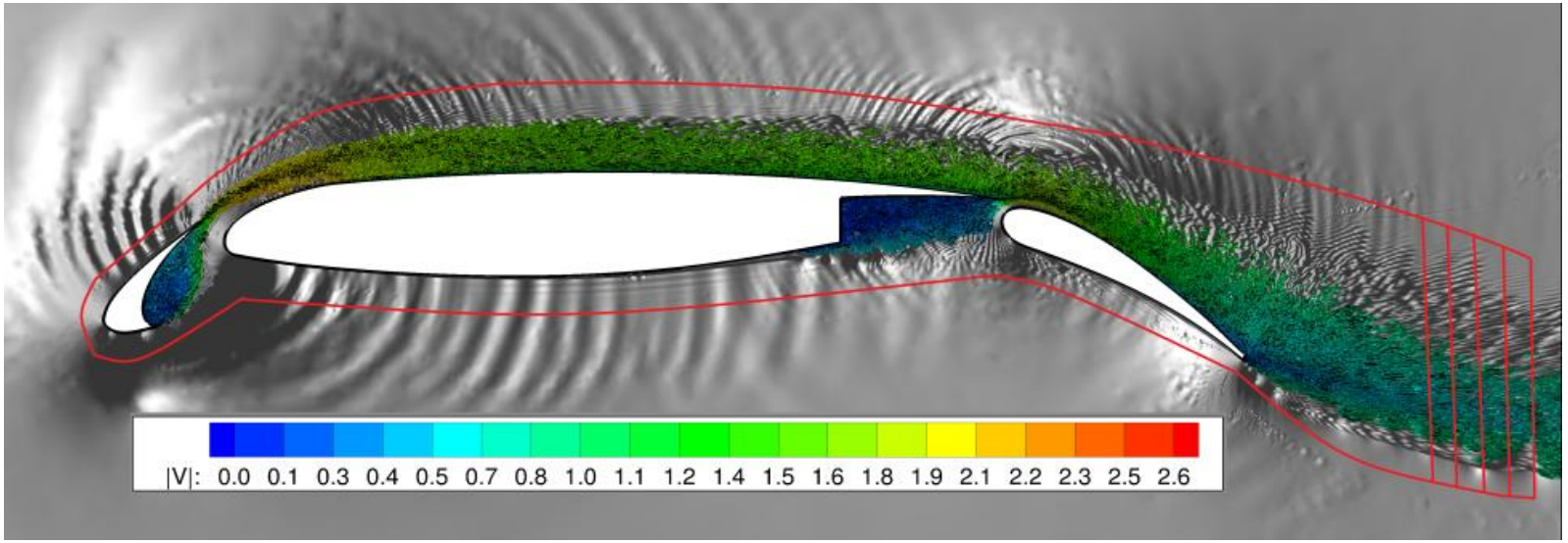




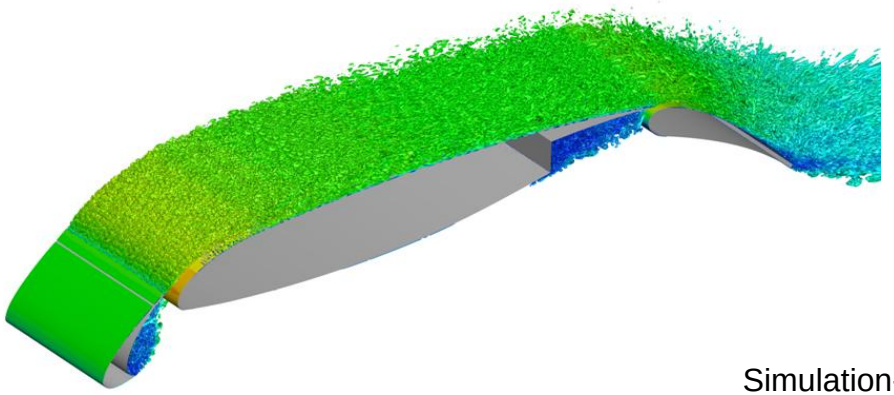


- **Акустика в дальнем поле – метод FW/H**
  параллельный постпроцессор для обработки акустических данных

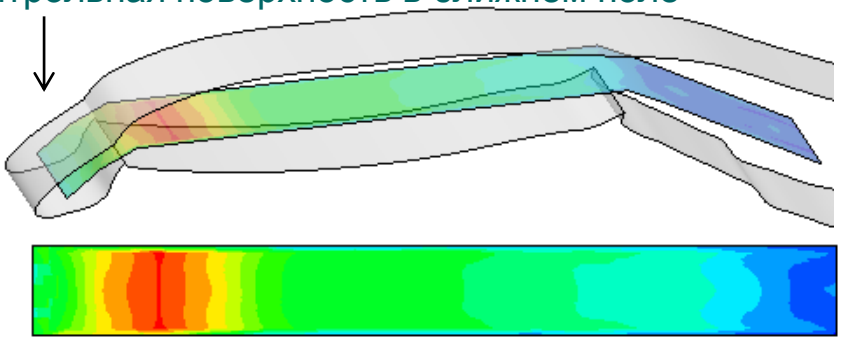




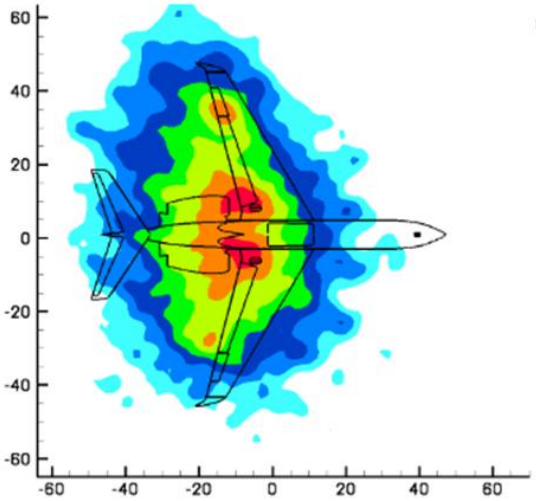
- **Численный beamforming**
  восстановление источника волнового уравнения во внутренней области

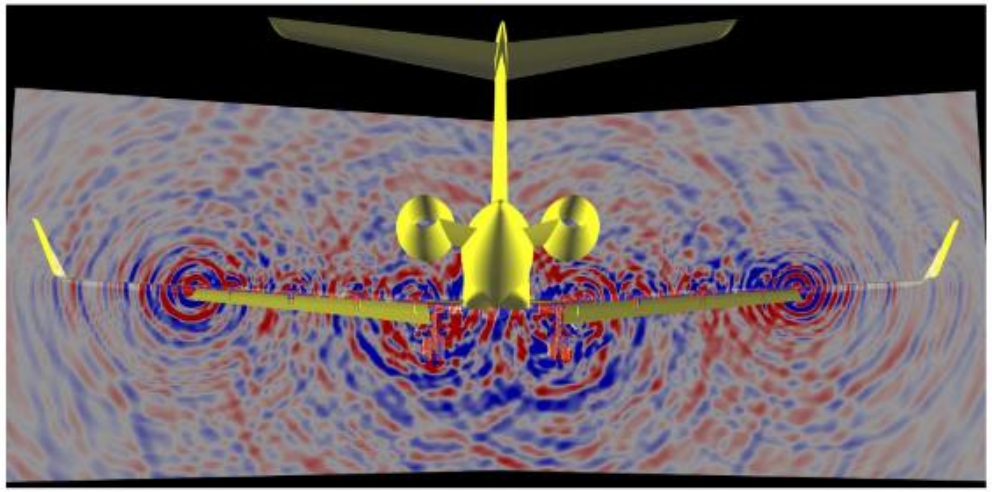


Контрольная поверхность в ближнем поле



Simulation-Based Airframe Noise Prediction of a Full-Scale, Full Aircraft
Mehdi R. Khorrami (NASA), Ehab Fares (Exa GmbH)



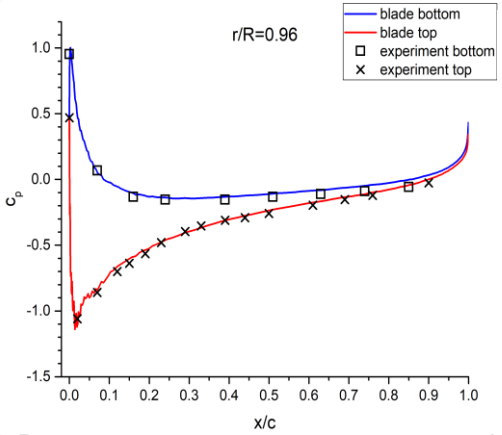


https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20160009089.pdf

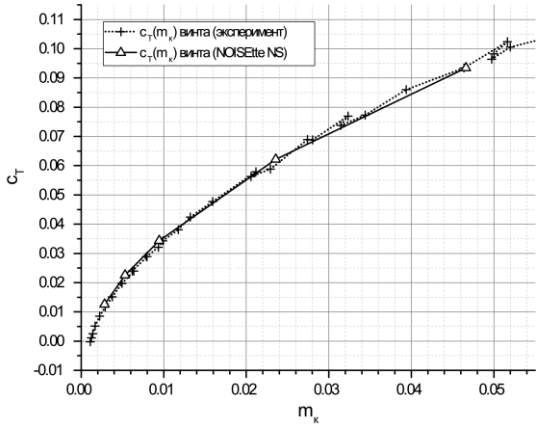
Примеры задач аэродинамики и аэроакустики

Расчеты по аэродинамическим и акустическим характеристикам винтов вертолета

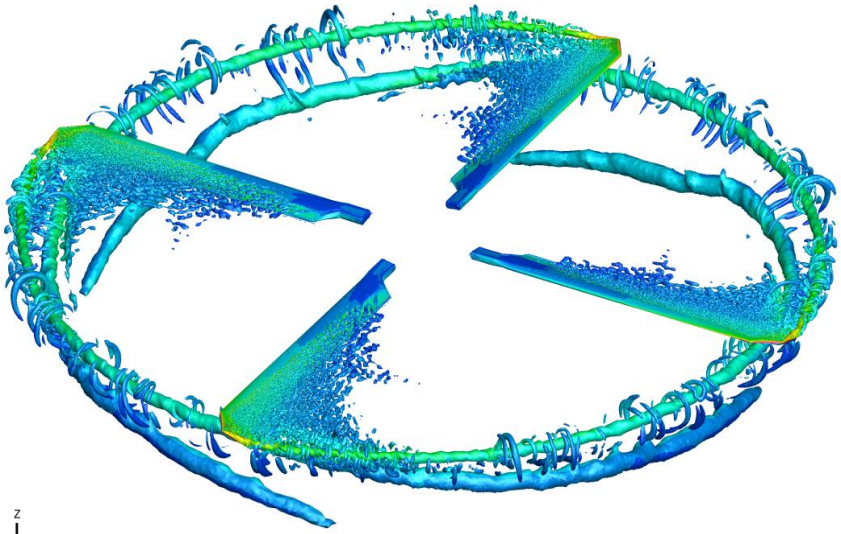
Сравнение с экспериментами

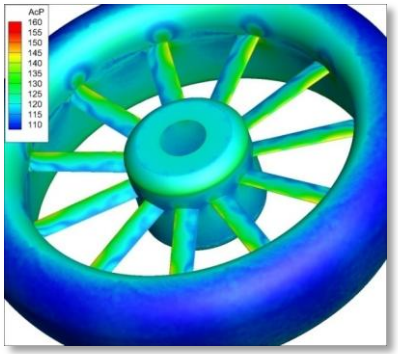


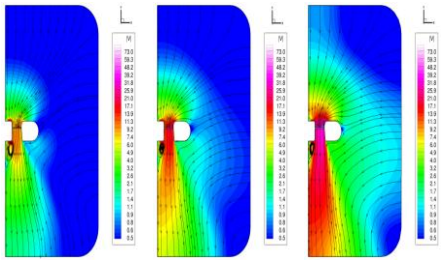
Распределение давления на лопасти НВ



Интегральные характеристики РВ







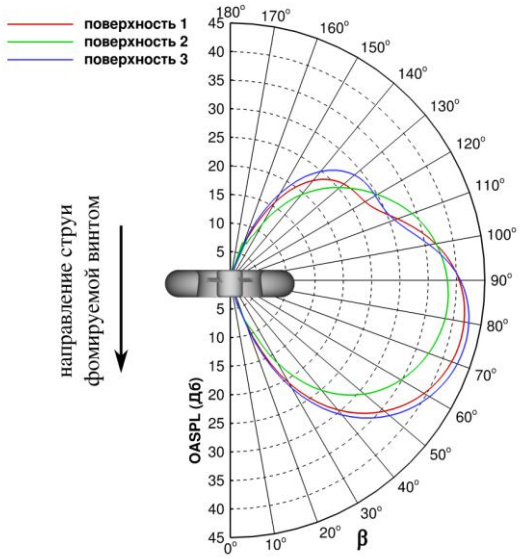
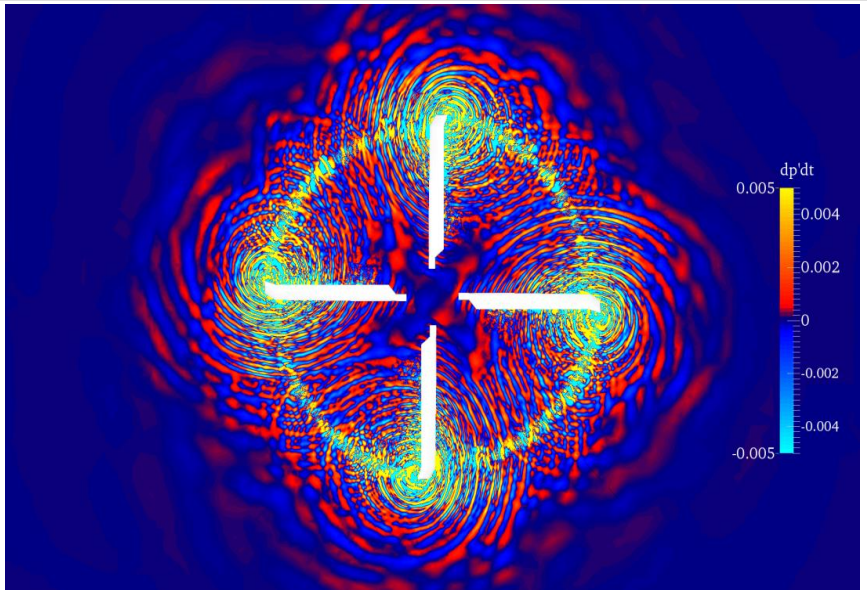
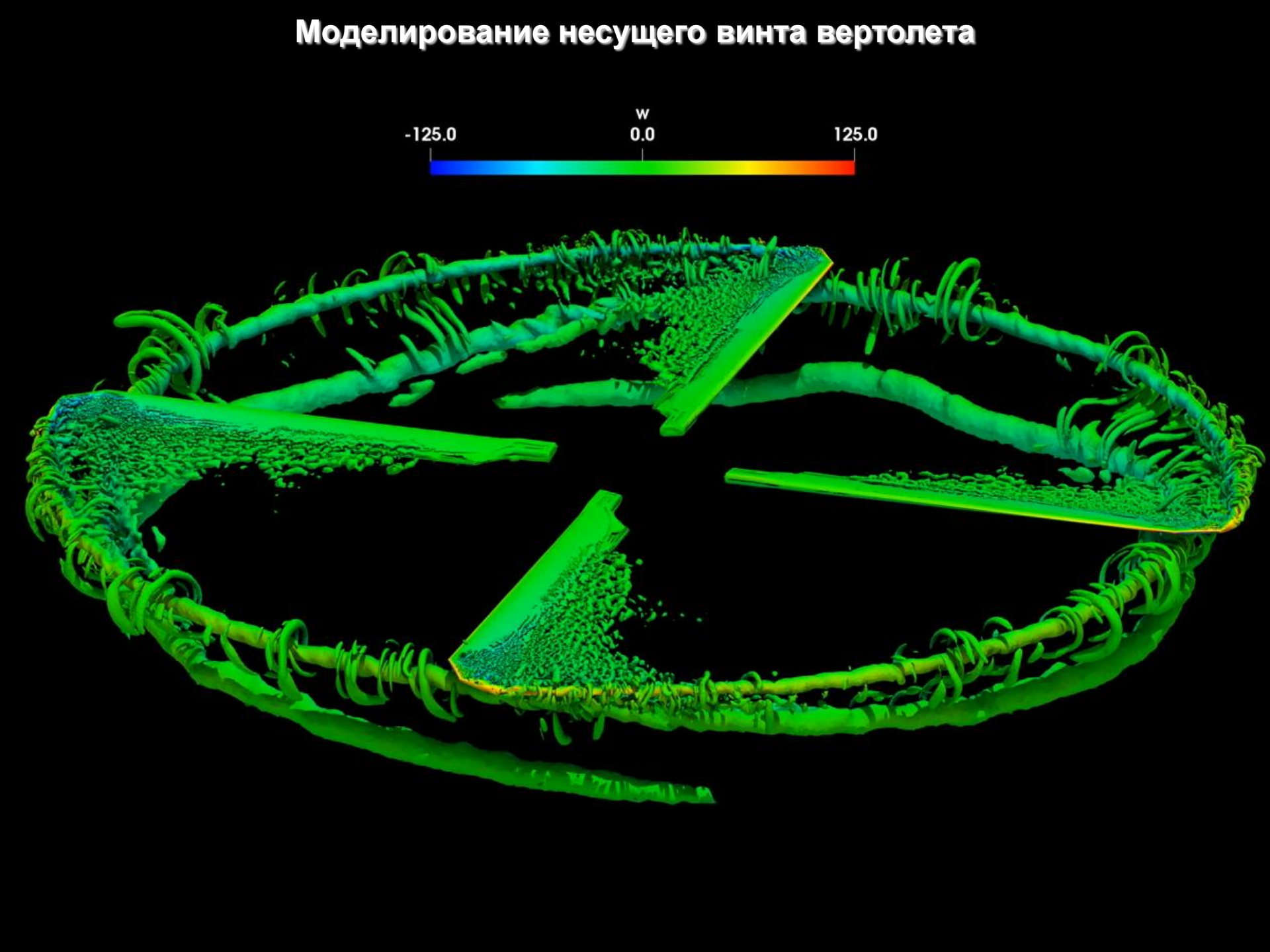


Диаграмма направленности шума РВ



Моделирование несущего винта вертолета



Аэродинамический шум от стреловидного крыла

3-компонентный профиль 30P30N

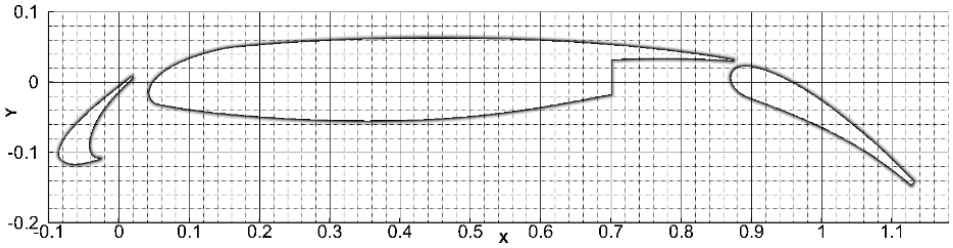
M_∞=0.17 Re_c=1.7·10^6

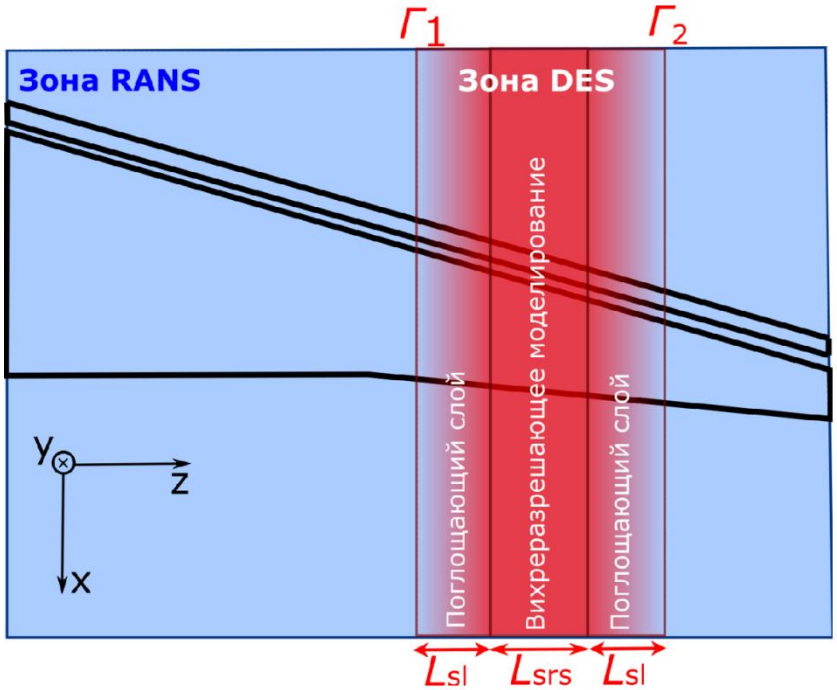
Подход: σ-DDES + Δ_ω

Метод: гибридная EBR CD+Upw

Сетки: 11-39 млн. узлов

+Технология поглощающего слоя (Sponge layer)



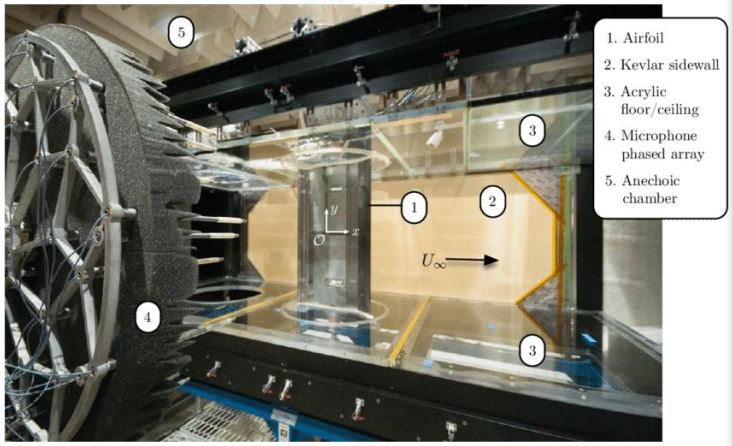


Цель

Отработка технологии расчета шума от сегмента стреловидного крыла

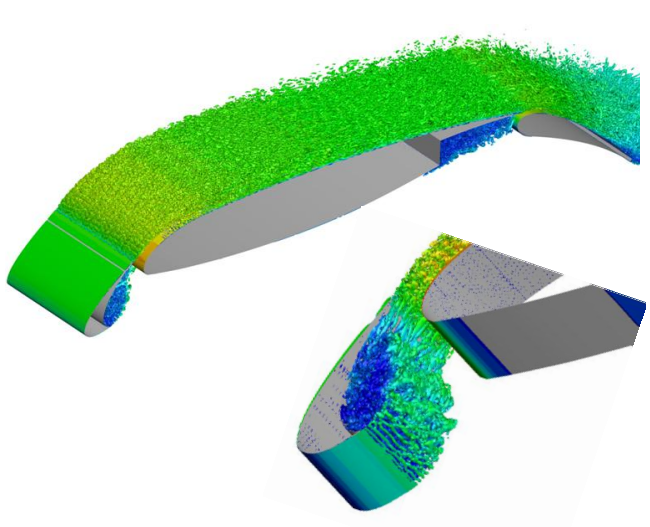
Этапы:

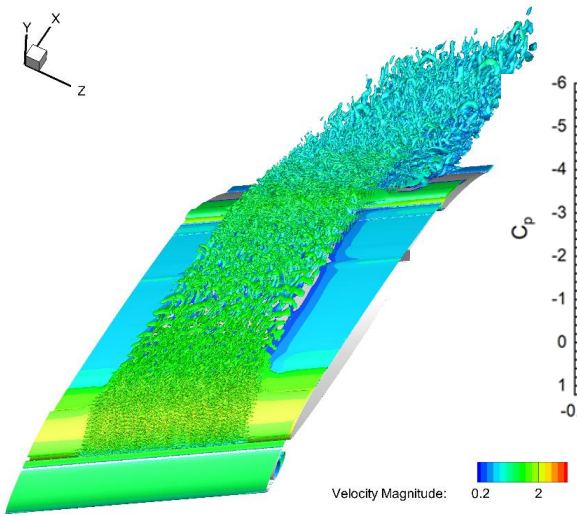
1. **RANS стреловидного крыла**
2. **DES сегмента крыла с буферной зоной до выходных боковых границ**

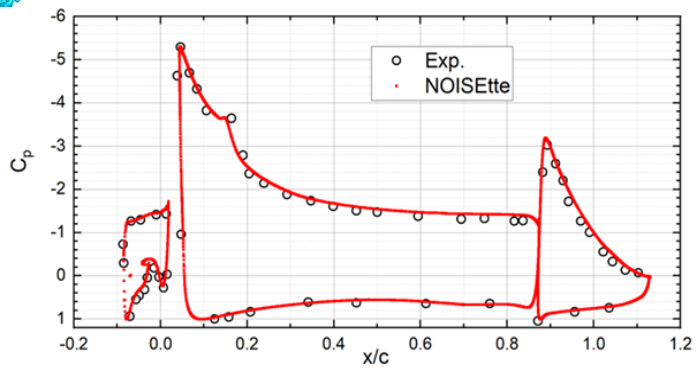


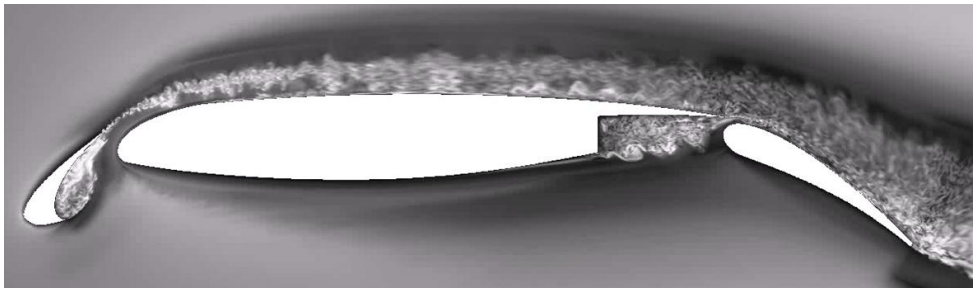
Pascioni, Kyle & Cattafesta, Louis. (2016)
Aeroacoustic Measurements of Leading-Edge Slat Noise
22nd AIAA/CEAS. DOI: 10.2514/6.2016-2960

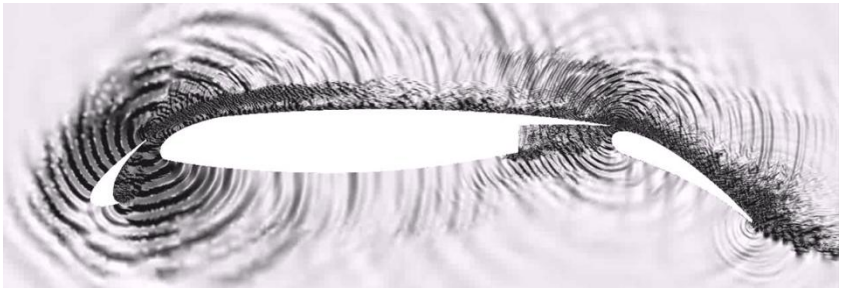
Аэродические и аэроакустические характеристики крыла с механизацией (ГСС)

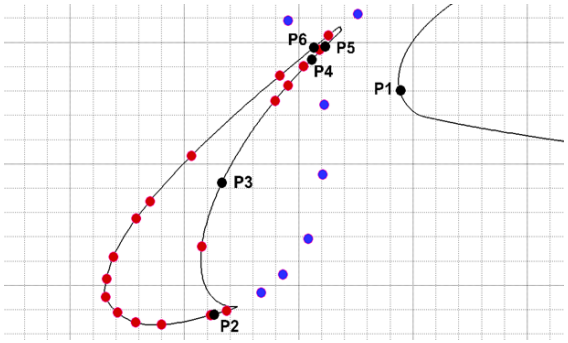


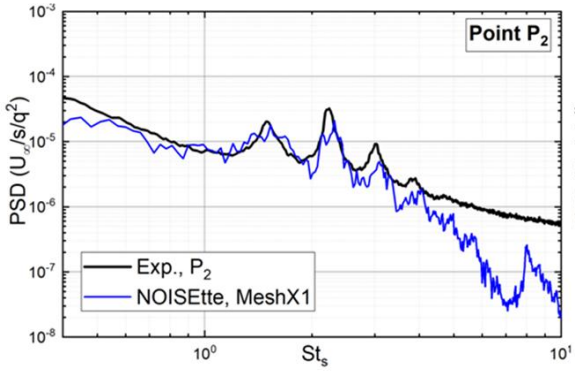


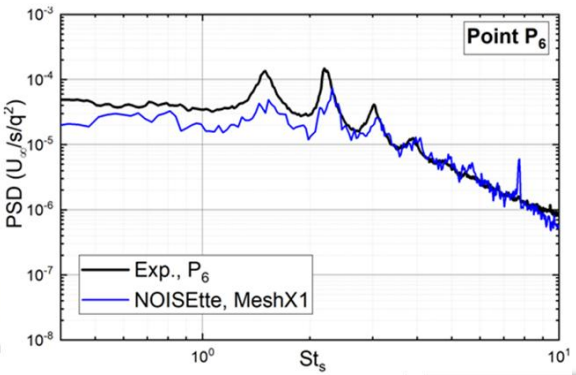








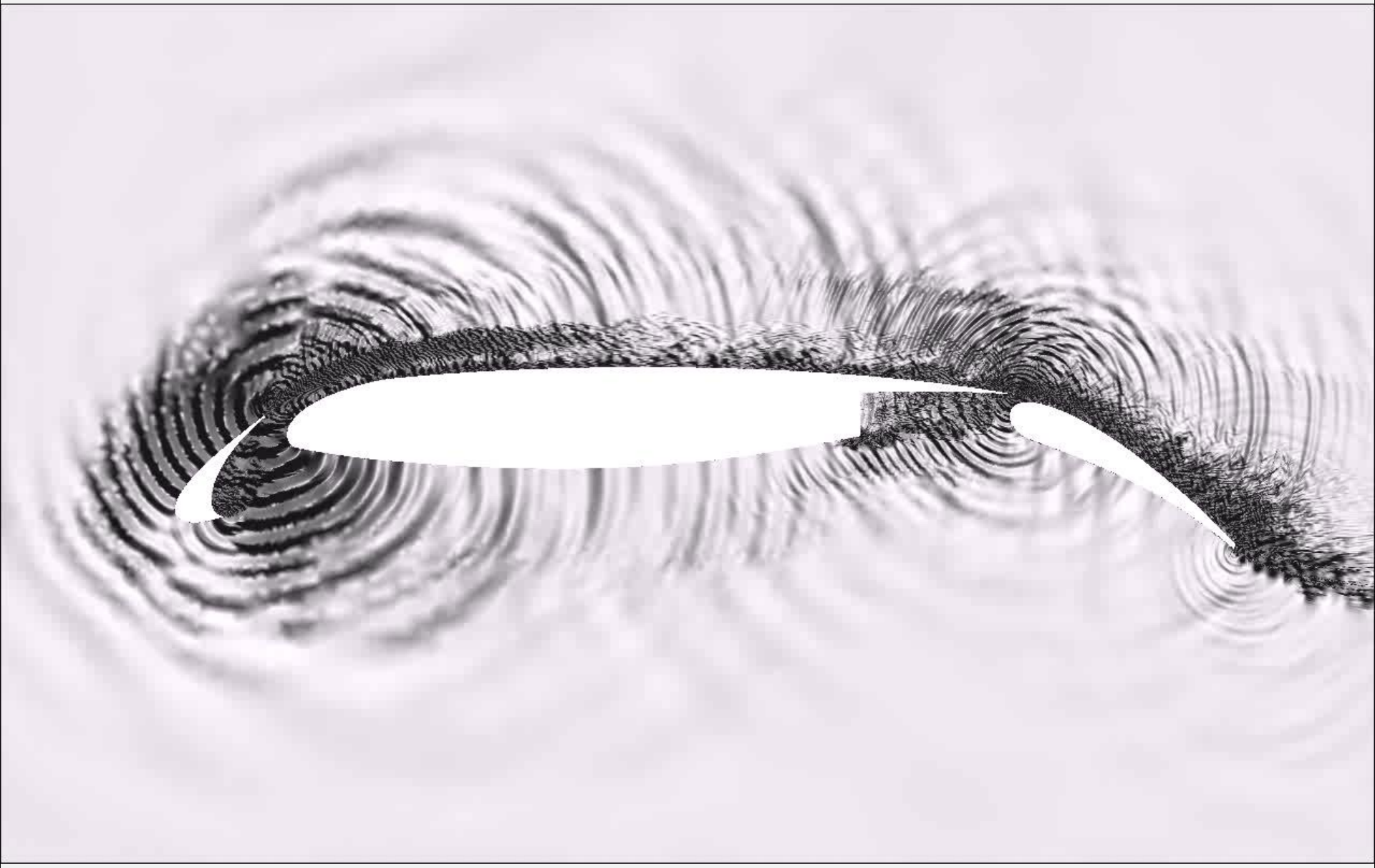




Моделирование крыла с механизацией



Моделирование крыла с механизацией



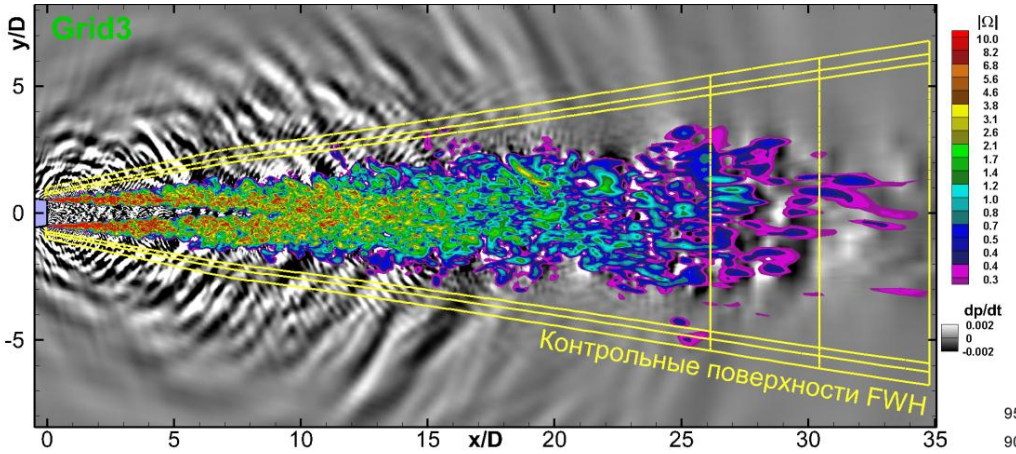
Примеры задач аэродинамики и аэроакустики

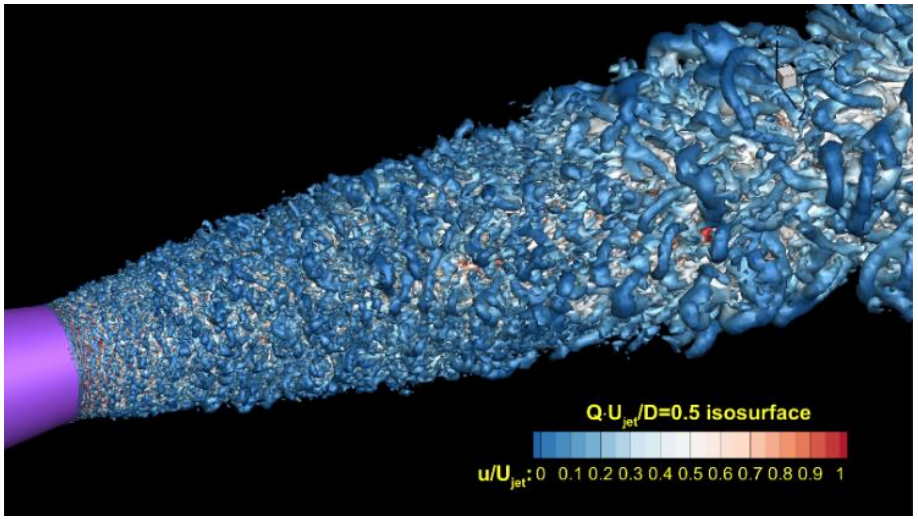
Затопленная ненагретая дозвуковая круглая струя

Параметры струи: $Re_D=1.1\cdot10^6$, $M_{jet}=0.9$

Сетки: 1.52 млн (Grid 1), 4.13 млн (Grid 2), 8.87 млн (Grid 3)

Подход: DDES [Shur et al., 2015]





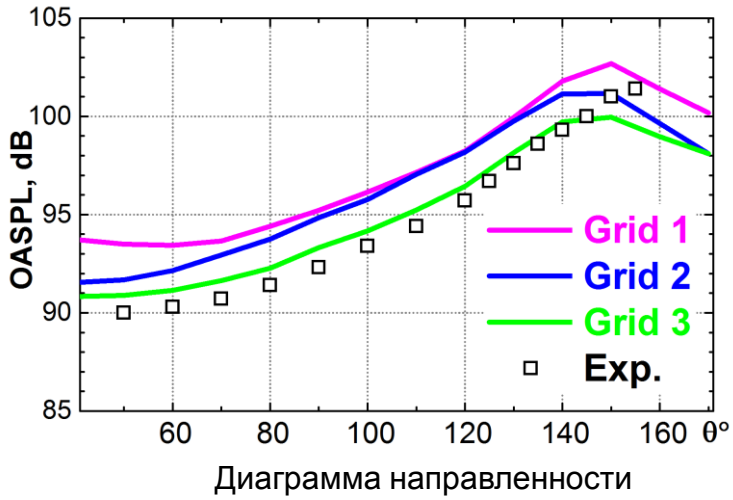
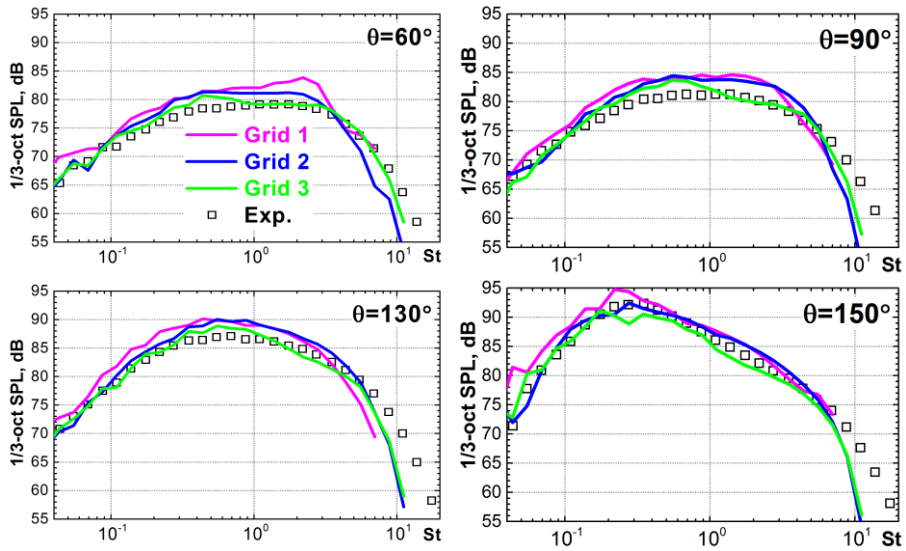


Диаграмма направленности



1/3 октавные спектры в точка дальнего поля, R=98D

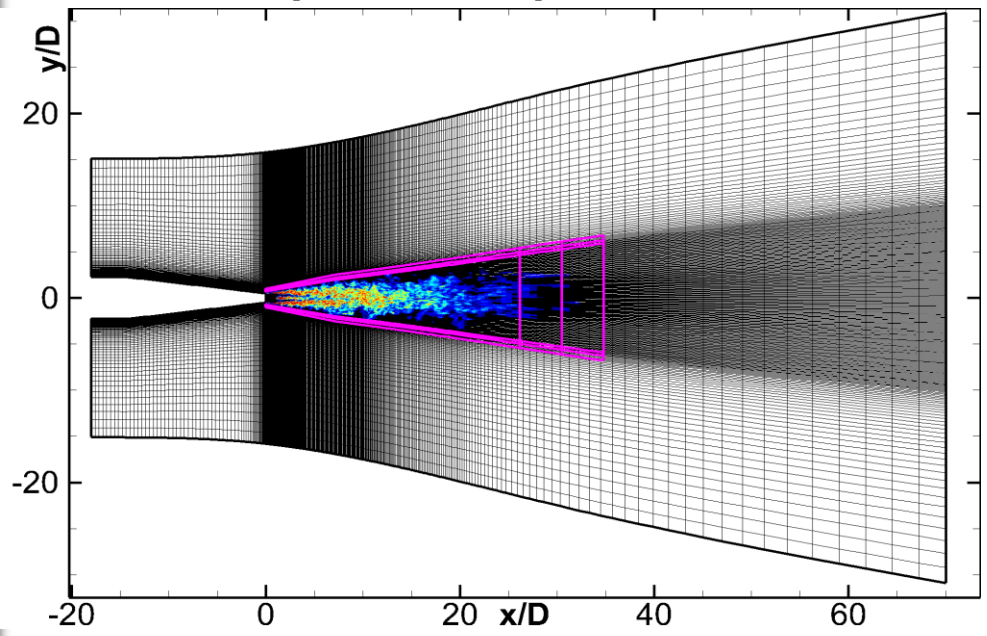
Примеры задач аэродинамики и аэроакустики

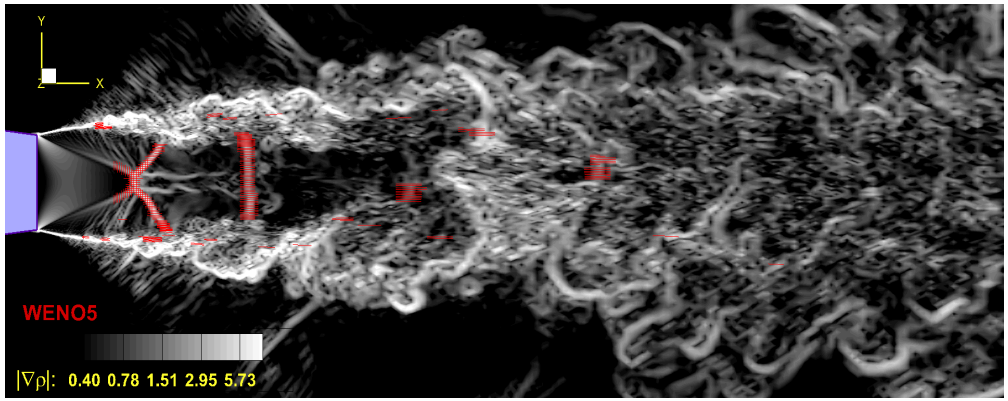
Горячая недорасширенная круглая струя

Параметры струи: $Re_D=1.27\cdot10^6$, NPR=3.86, M=1.54, $M_a=2.18$, $U_{jet}=748.4$ м/с

Сетка: 4.55 млн узлов

Подход: DDES [Shur et al., 2015]





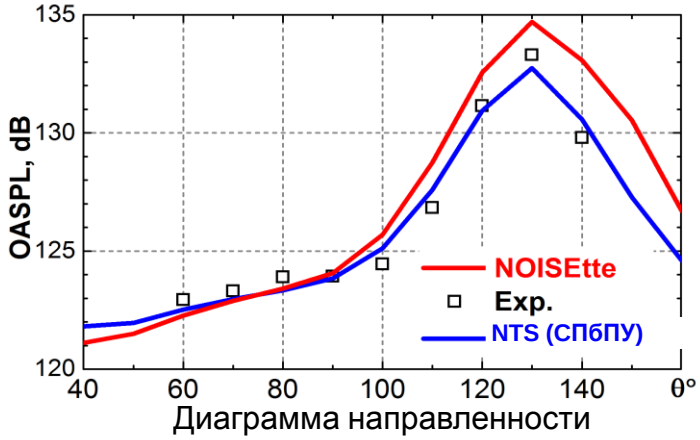
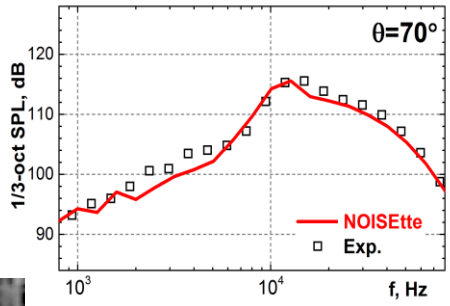
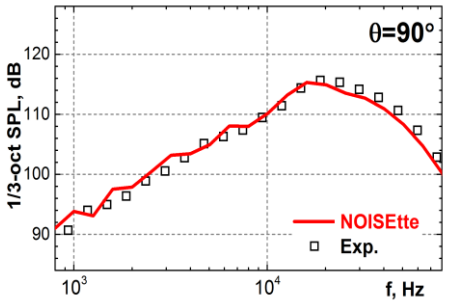
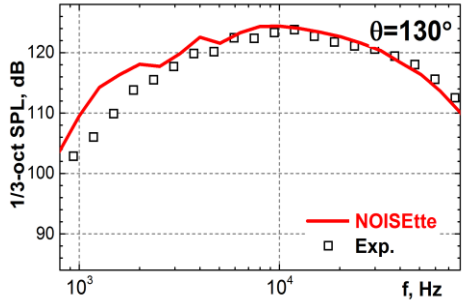


Диаграмма направленности



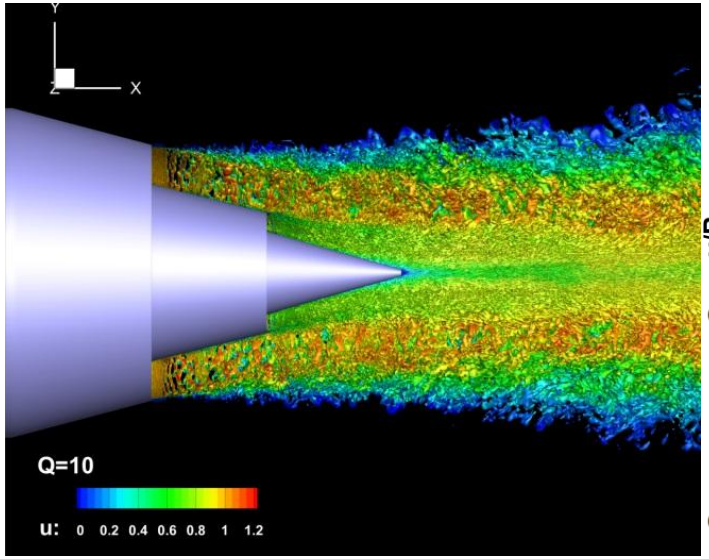


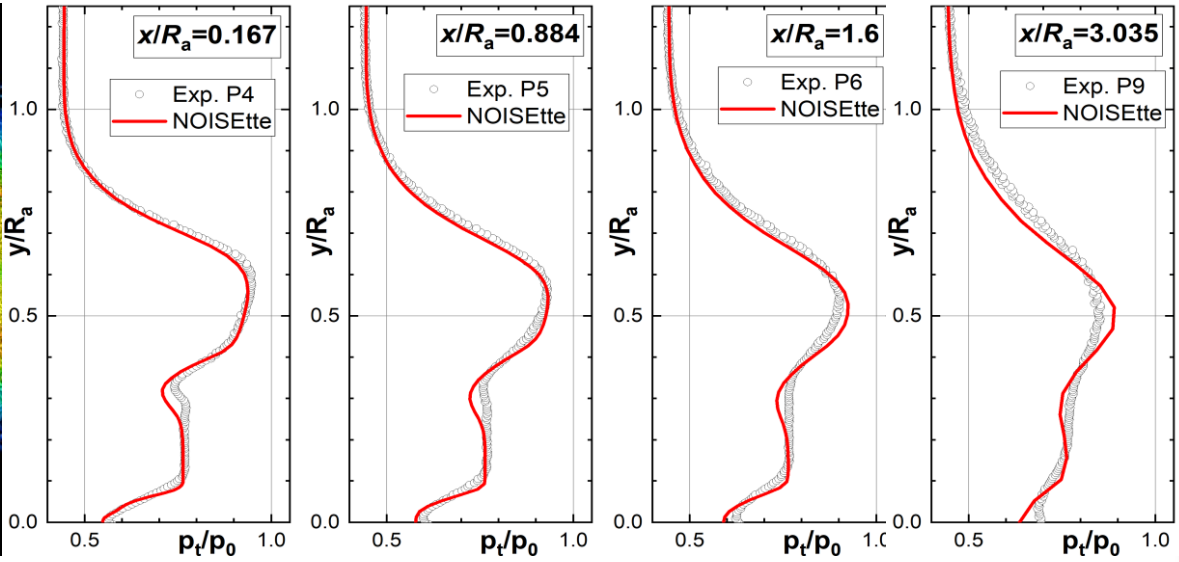


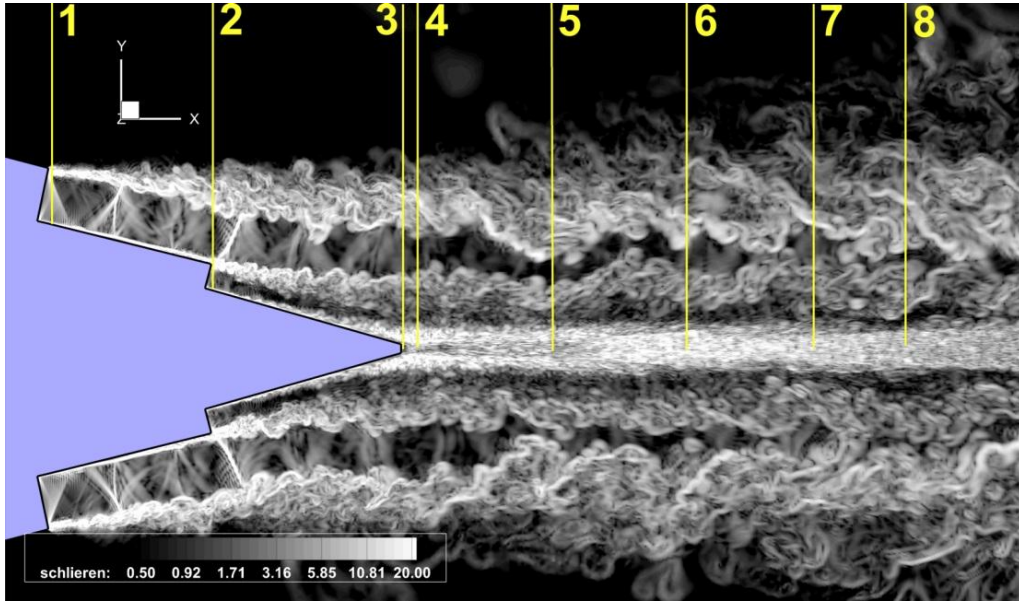
1/3 октавные спектры в точка дальнего поля, R=100D

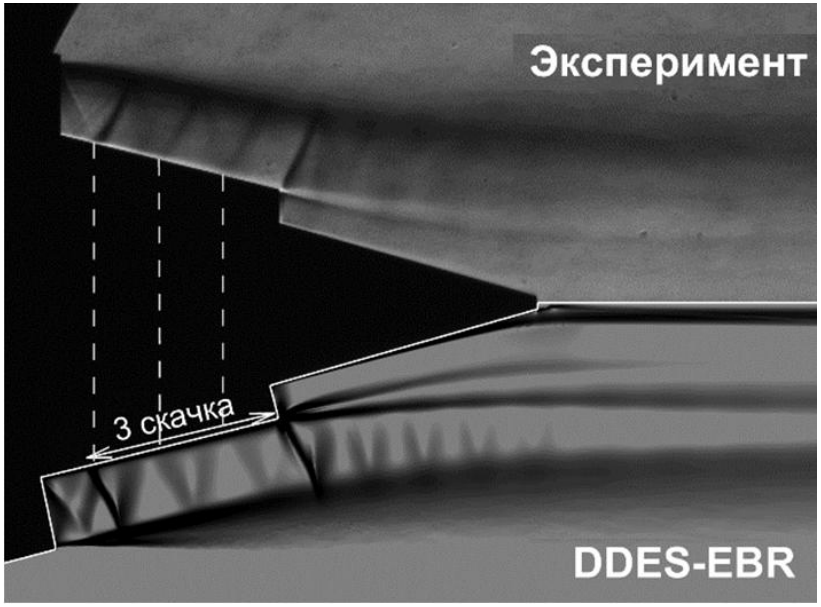
Примеры задач аэродинамики и аэроакустики

Струя из двухконтурного сопла (совместно с ЦАГИ), эксперимент ИТПМ СО РАН (В.И. Запрягаев)







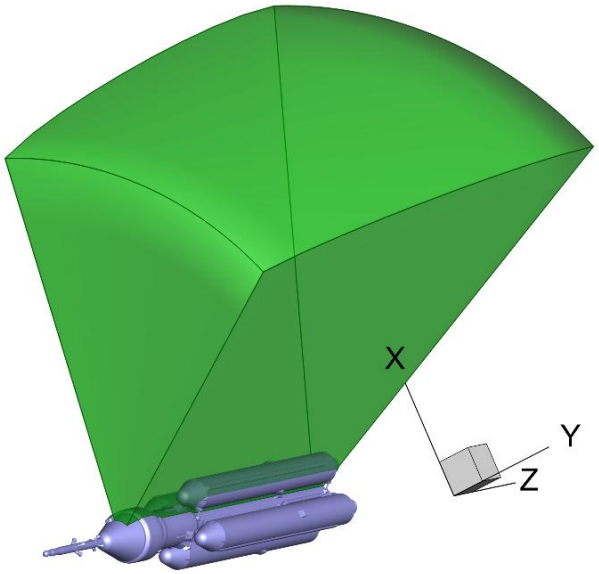


Примеры задач аэродинамики и аэроакустики

Аэроакустические нагрузки на поверхности ракеты-носителя (РКК «Энергия»)

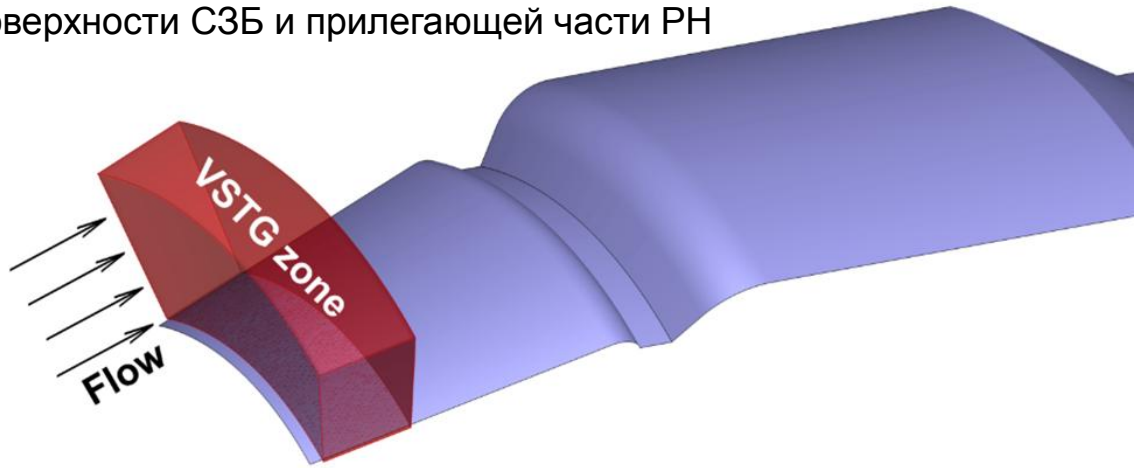


- **Подход к моделированию**
  - IDDES
  - VSTG: генератор синтетической турбулентности с объемным источником
  - гибридная EBR CD+Upw+WENO схема



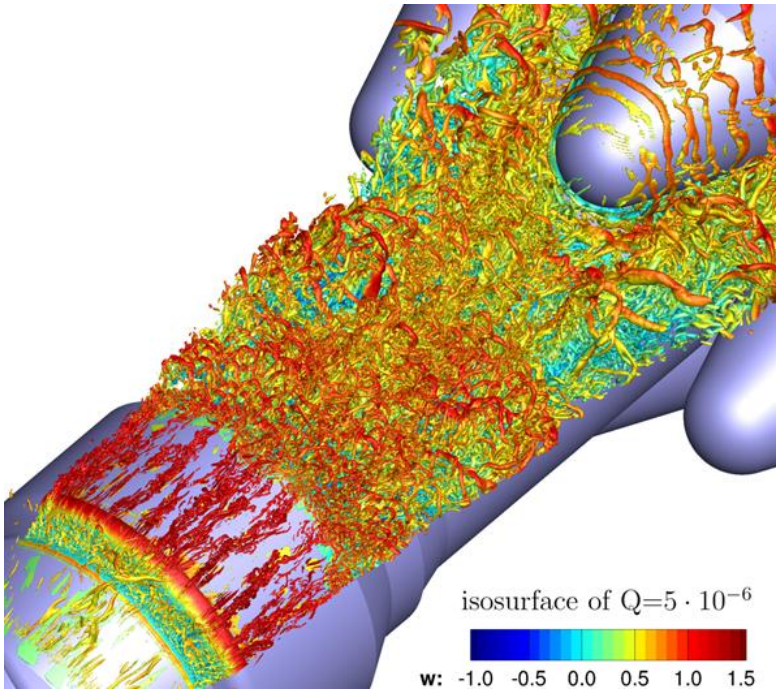
- **Вычислительный домен и сетки**
  - 45° без учета боковых блоков (27.1 млн.)
  - 90° с учетом боковых блоков (63.4 млн.)

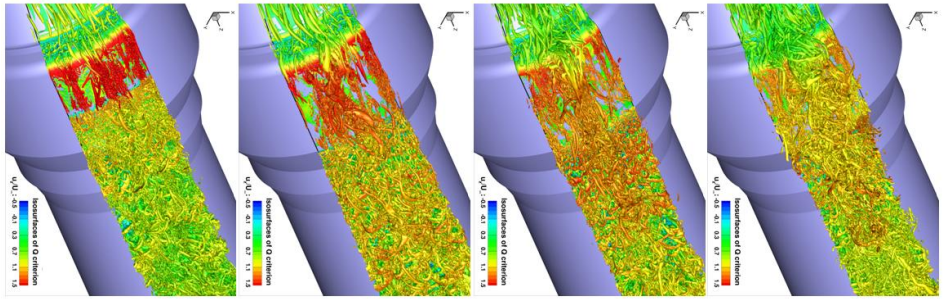
- **Задача**
  - оценка аэроакустических нагрузок на поверхности СЗБ и прилегающей части РН

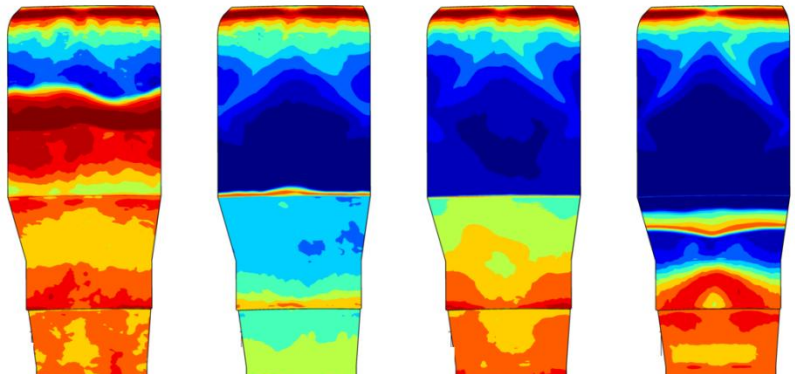


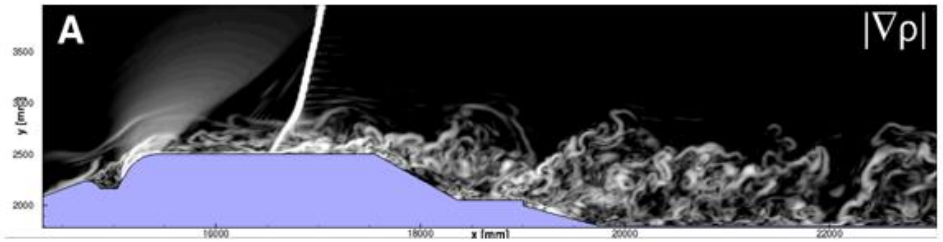
Примеры задач аэродинамики и аэроакустики

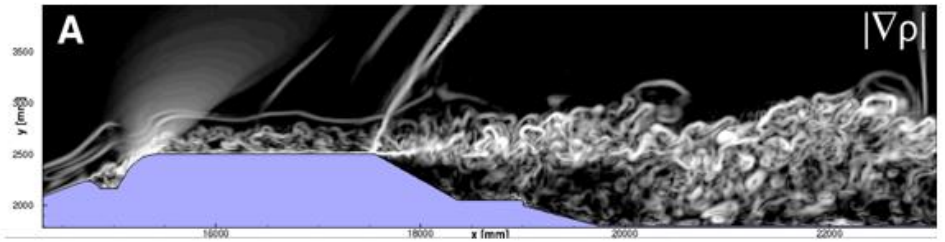
Аэроакустические нагрузки на поверхности ракеты-носителя (РКК «Энергия»)

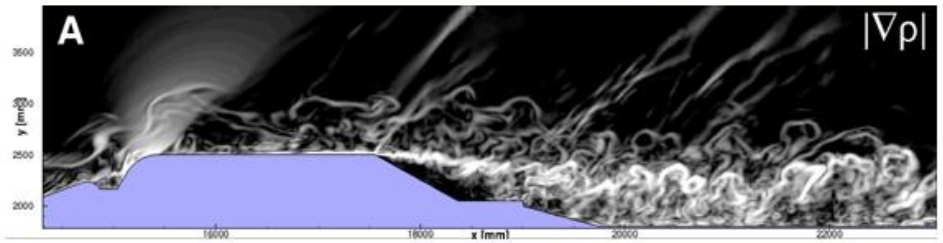


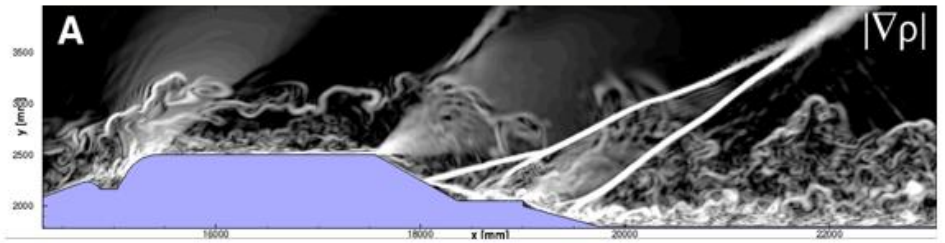






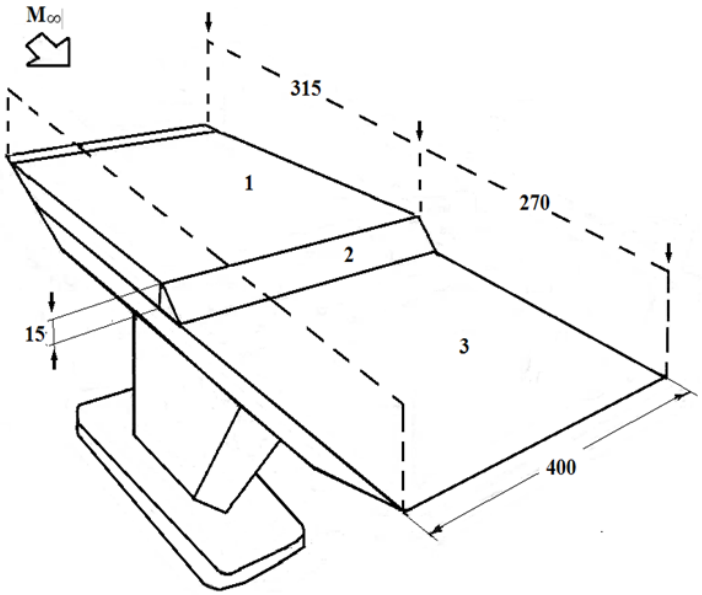






Примеры задач аэродинамики и аэроакустики

Сверхзвуковое обтекание наклонного обратного уступа



Данные ЦАГИ

Эксперимент ИТПМ СО РАН

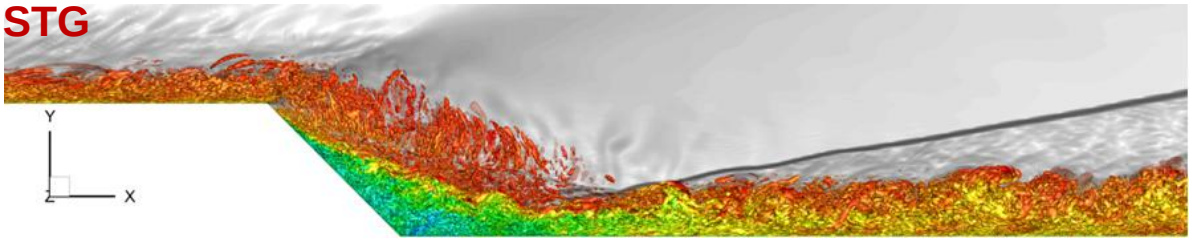
А.А. Желтоводов

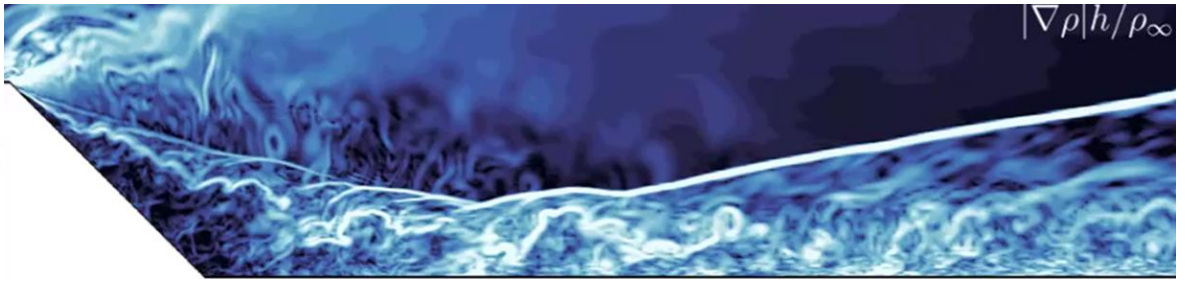
M_∞=3, Re_h=4.905·10^6

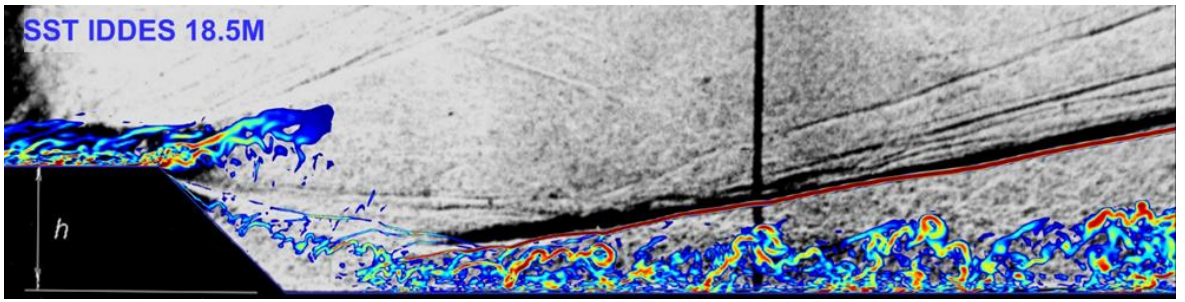
Подход: IDDES + STG

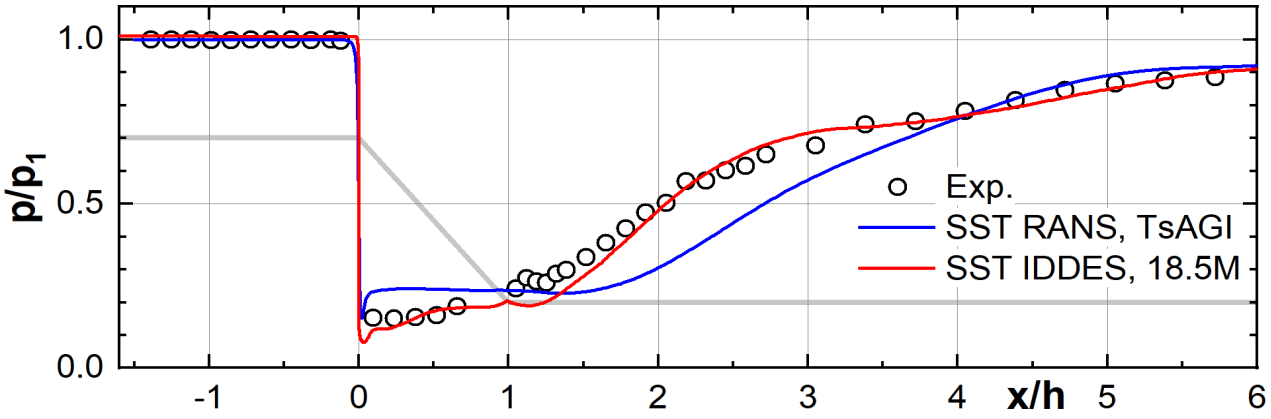
Сетка: 18.6 млн. узлов

Расход: 15K CPUh





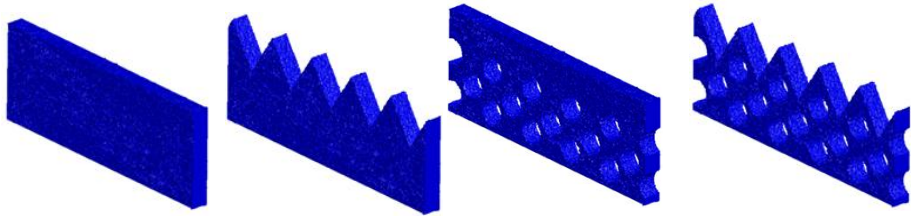


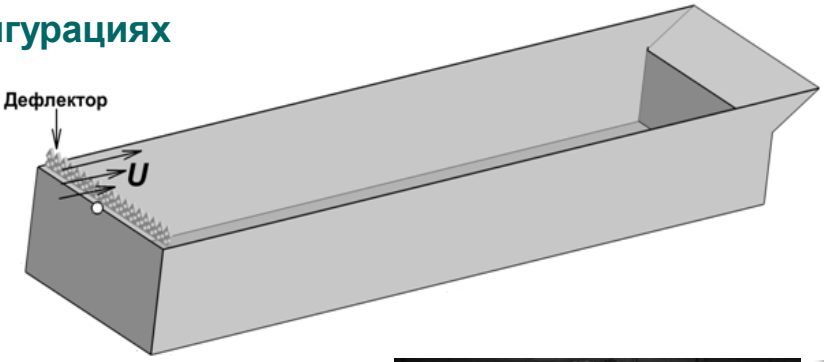


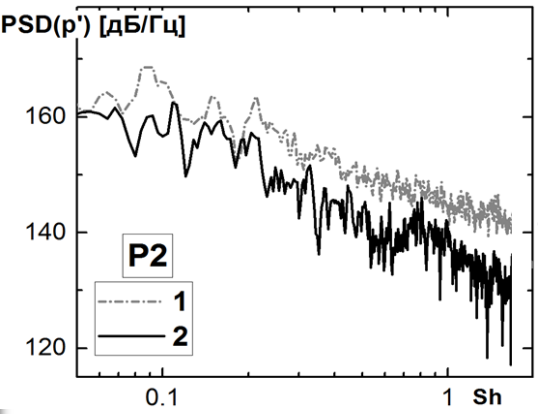
Примеры задач аэродинамики и аэроакустики

Исследование турбулентного течения и акустических нагрузок в каверне

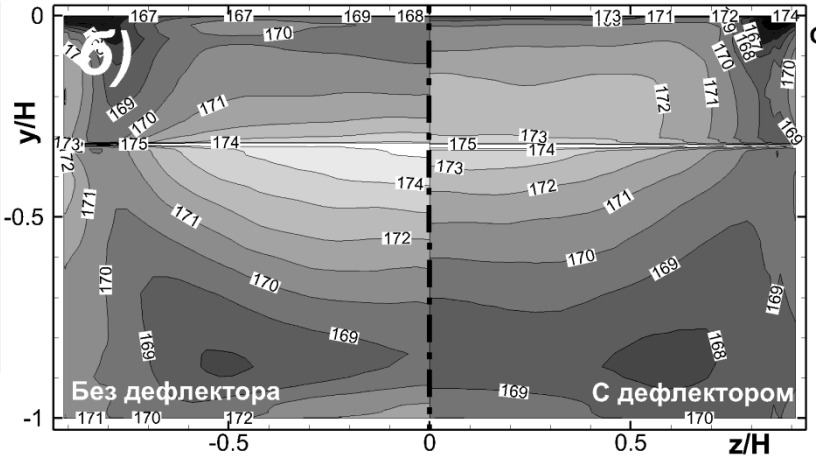
- IBC – моделирование дефлекторов в различных конфигурациях

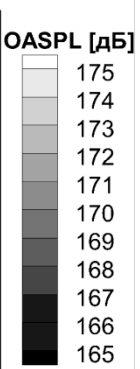


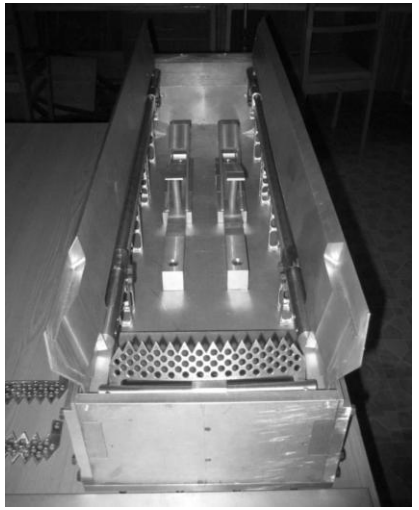


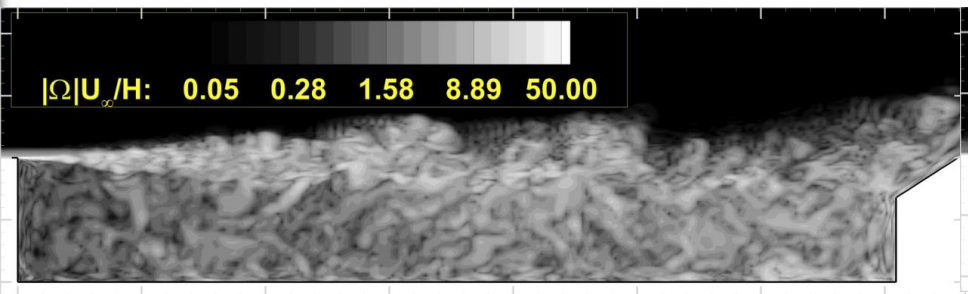


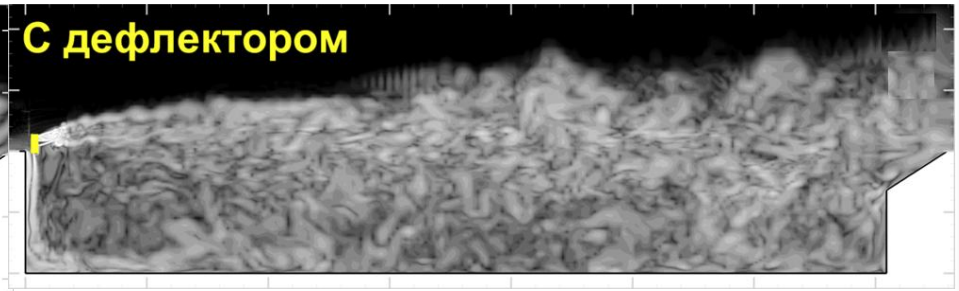
Контрольная точка на дне каверны











Примеры задач аэродинамики и аэроакустики

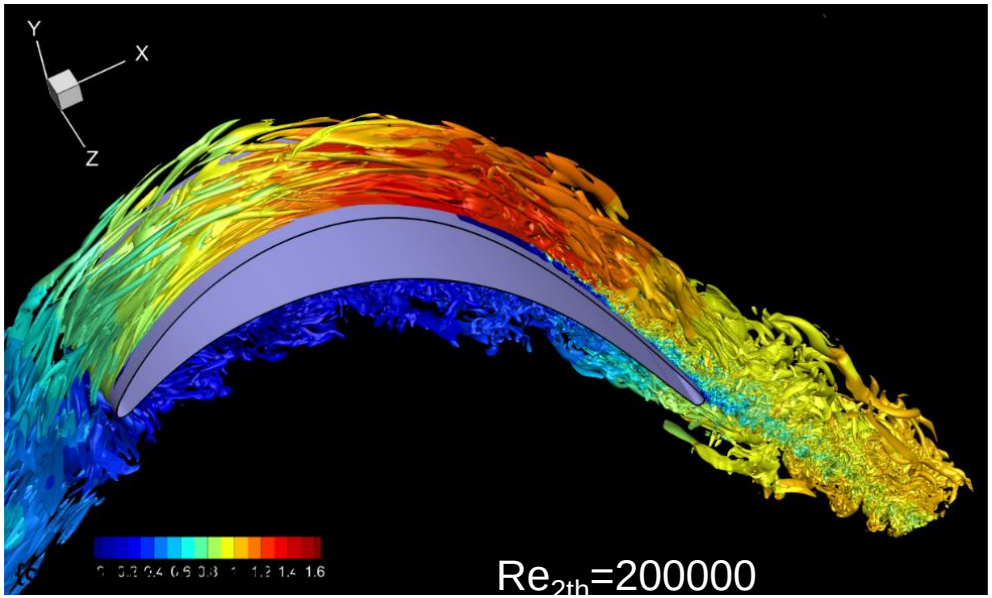
Турбины низкого давления

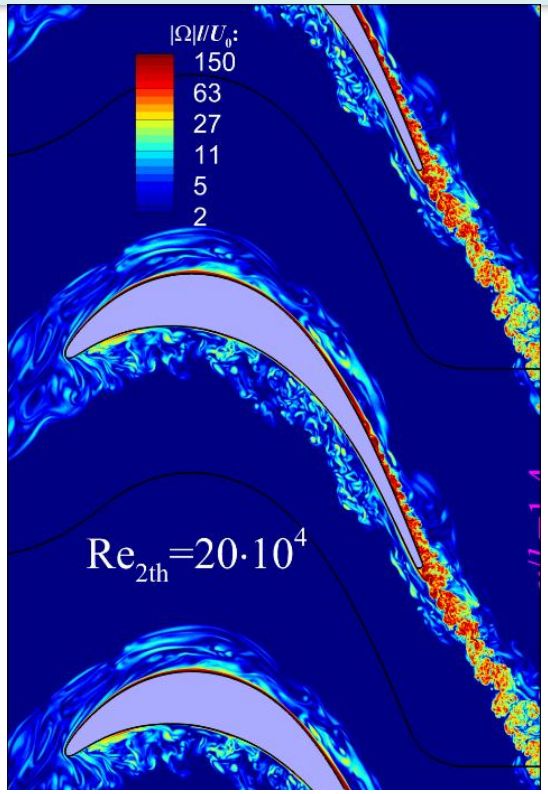
Цель: исследование эффекта потерь полного давления в турбинной высоконагруженной решетке при уменьшении числа Рейнольдса

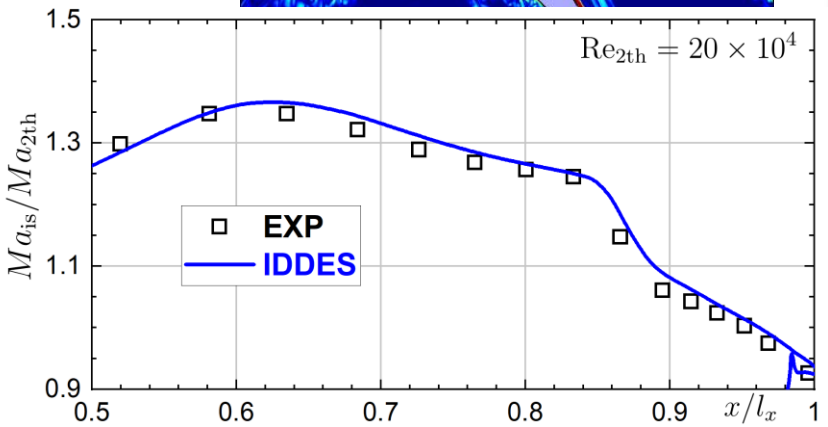
Постановка задачи (турбинная решётка T106C)

- гибридный RANS-LES **подход** IDDES
- **VSTG:** генератор синтетической урбулентности
- **сетка:** 80 млн. узлов

Вычислительная стоимость: 30K CPUh



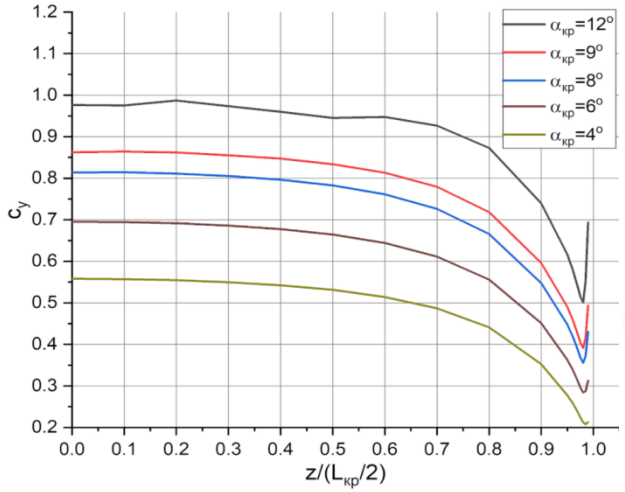


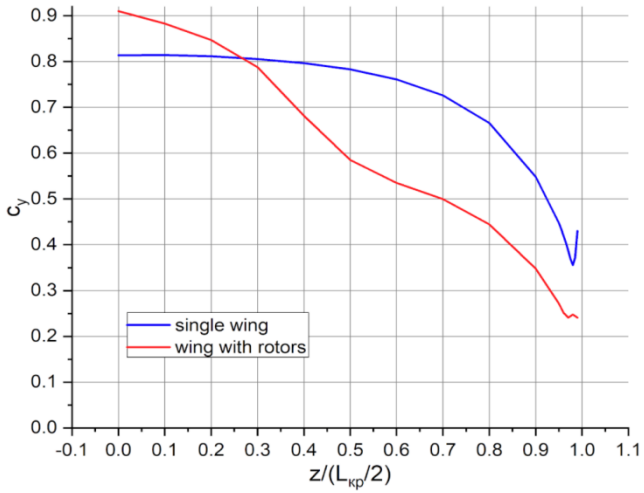


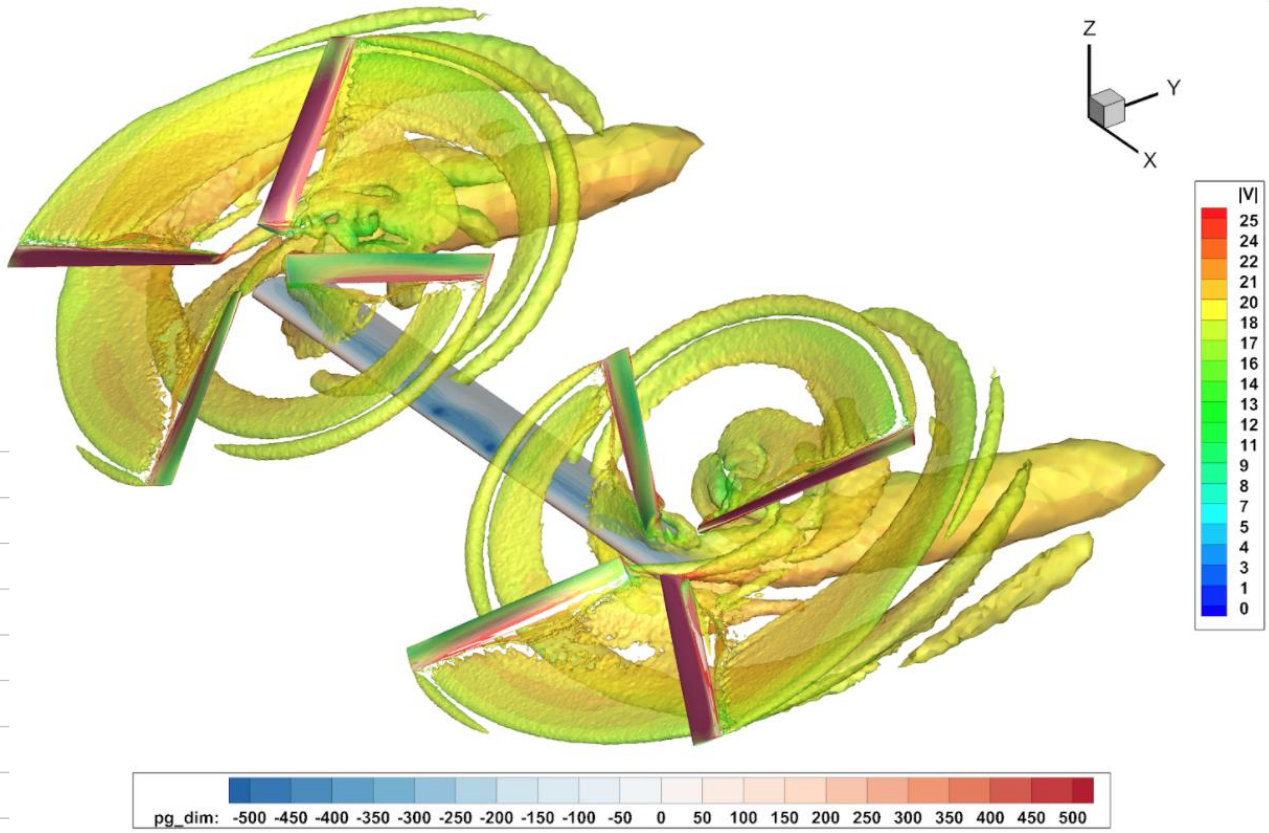
Изэнтропическое число Маха на поверхности

Примеры задач аэродинамики и аэроакустики

Обтекание крыла конвертоплана с поворотными несуще-тянущими винтами
в переходном режиме: показан угол атаки крыла 8°, угол установки винта 30° градусов
концевая скорость винта 75.5 м/с, скорость внешнего потока 18 м/с
четырехлопастные винты с круткой 38 градусов.







Заключение

- Высокопроизводительные вычисления – это сложно
- Суперкомпьютерное моделирование – это востребоавно
- Научный софт для фундаментальных исследований – нужен
- Промышленный расчетный софт для НИОКР – очень дорогой
- Смежные темы: 3D графика, игрушки, обработка данных, …

А если что-то сложно, очень сложно,
с высоким порогом вхождения, мало кто это умеет, …

И это что-то нужно, востребовано, это покупают за много денег…

Какой вывод?

1 Введение

1.1 Эволюция суперкомпьютерной архитектуры

Чтобы не нарушать традиции, начнем со статистики. Производительность суперкомпьютеров (все еще) интенсивно растет. За последние десять лет, с 2010 по 2020 год, пиковая производительность самого мощного суперкомпьютера в рейтинге Top 500 (https://top500.org) увеличилась в 220 раз.

Но за счет чего происходит этот рост производительности, и какая у этой медали обратная сторона? Мощностей все больше, но пользоваться HPC (High Performance Computing) системами становится сложнее. Упомянутый выше рейтинг строится на основе результатов по бенчмарку HPL (High Performance Linpack). В этом тесте производительности решается (очень) большая системка линейных уравнений с плотной матрицей параллельным прямым методом на основе LU (Lower-Upper) разложения. Объем данных $O(N^2)$, вычислительная сложность $O(N^3)$. За счет того, что вычислительных операций сильно больше, удается снимать аж под 90% от пика. Конечно, при таком соотношении объема данных и вычислений есть над чем поработать, что пооптимизировать. Такой тест слишком "удобен" для матчасти. Гораздо удобнее, чем абсолютно подавляющее большинство реальных приложений. Из-за такого удаления от реальности в качестве альтернативы, точнее, полной противоположности, появился бенчмарк HPCG (High Performance Conjugate Gradients). Этот тест даже, пожалуй, излишне жестокий по отношению к матчасти. Тут система с разреженной матрицей решается итерационным методом, для чего используется компот из метода сопряженных градиентов, многосеточного метода, симметричного метода Гаусса-Зейделя. Подробнее тут: https://www.hpcg-benchmark.org/ Результаты там, конечно, печальные. Гордо и с большим трудом достигаются значения в 1–2 % от пика. Это куда ближе к реальности. Но как же мы дошли до жизни такой?

А было оно примерно так. Сначала побеждали силы добра. Бурно росла производительность процессоров за счет повышения тактовой частоты. Десятки МГц, сотни МГц, несколько ГГц. Это светлая сторона силы. Далее рост закончился, поскольку слишком уж сильно растет энергопотребление с дальнейшим увеличением частоты. Стала расти производительность процессора при фиксированной частоте. Процессор за такт стал делать все больше и больше операций за счет параллелизма уровня инструкций. Размножился конвейер команд, появились multiple issue pipeline и прочая суперскалярность (с этими словечками мы далее будем разбираться подробнее). Затем пророс out-of-order – внеочередной исполнение инструкций, в процессорах появились более сложные планировщики (scheduler) типа многопортовых reservation station. Процессор научился на одном такте задействовать множество разнородных исполнительных устройств. Очень кстати подросла пропускная способность оперативной памяти. В итоге производительность заметно выросла даже без роста частоты. Вроде бы всё хорошо. Светлая сторона эволюции архитектуры. Сейчас начнется темная сторона.

Производительность многопроцессорной системы стала расти не столько за счет увеличения производительности процессора, сколько за счет многократного увеличения их количества. Рост числа процессоров в системе усложняет ее эффективное использование: требуются все более масштабируемые алгоритмы, изощренные схемы обменов данными, упрощение алгоритмов в пользу параллельной эффективности, но зачастую в ущерб общей вычислительной стоимости.

Дальше процессоры стали многоядерными – multicore. Сначала двухядерные, потом четырехядерные, и понеслось. С мультикорами еще можно было иметь дело, используя те же подходы к распараллеливанию с распределенной памятью, как и в случае одноядерных процессоров. Небольшие доработки обмена данными между параллельными процессами решали проблему. Затем процессоры стали существенно-многоядерные – manycore. Это когда в одном процессоре уже десятки ядер. Это все усложнило. Потребовалось сочетать распараллеливание с распределенной и общей памятью (с этими всеми распараллеливаниями мы далее подробно разберемся). Пришлось применять многоуровневое распараллеливание, усложнять балансировку, оптимизировать доступ к памяти в многопоточном режиме, учитывать NUMA (Non-Uniform Memory Access) фактор, и решать много других проблем.

С переходом на мэникоры число процессорных ядер в суперкомпьютерах дико выросло. В крупных системах это число давно перевалило за миллион. Естественно, лишь очень узкий класс алгоритмов может задействовать такие ресурсы, а сами задачи, которые такие алгоритмы могут решать, часто даже и близко не оправдывают использование таких вычислительных мощностей. Конечно, нас никто и не заставляет использовать такие машинки целиком. Это крупные системы коллективного доступа с большим количеством пользователей. Нас вполне устроит эффективно использовать хотя бы их малую часть. Но число процессоров и в этой "хотя бы их малой части" становится все больше и больше. Темная сторона набирает силу.

Началась гонка производительности. Захотелось делать устройства с все более высокой пиковой производительностью. Но как? Частота не растет. Out-of-order уже нехило прокачали, навороченные планировщики в ядре и так умеют эффективно задействовать множественные исполнительные устройства. Пропускная способность оперативной памяти уперлась и не хочет дальше расти (с причинами тоже будем разбираться). Что же делать? Правильно. Расширять векторные расширения. Искать потерянные ключи надо там, где проще – под фонарем. Исполнительные устройства для арифметики с плавающей точкой, так называемые FMA (Fused Multiply-Add) unit-ы научились выполнять команду сразу для двух наборов аргументов. Потом для четырех, потом для восьми... Современное ядро с двумя 512-битными FMA юнитами делает по 32 операции с плавающей точкой за такт (с этим всем тоже поразбираемся). Получается, пиковая производительность выросла многократно. А что произошло с эффективностью, то есть соотношением между фактически получаемой на практике производительностью и теоретической пиковой производительностью? Правильно – она многократно уменьшилась для подавляющего большинства приложений. Понадобился еще один уровень распараллеливания – векторизация, который все сильно усложняет, но обычно вообще мало что дает в силу ограничений пропускной способности памяти. И это соотношение пиковой производительности и пропускной способности памяти неуклонно растет. То есть даже очень-очень высокопараллельный алгоритм, очень-очень хорошо сделанный, с многоуровневым распараллеливанием, с векторизацией, все равно будет претендовать лишь на жалкие проценты от пиковой производительности. Потому что просто не сможет настолько быстро выгребать данные из памяти, чтобы обсчитывать их на такой скорости. Вот такая она – темная сторона. Дальше – темнее.

Архитектуры стали гибридными. В суперкомпьютере стали сочетаться вычислительные устройства принципиально различной архитектуры – центральные процессоры и массивно-параллельные ускорители-сопроцессоры. Наибольшее распространение в этой роли получили графические процессоры GPU (Graphics Processing Unit). И тут уже начинается настоящий мрак. GPU используют параллельную парадигму потоковой обработки, которая уж очень сильно ограниченная, то есть в нее вписывается лишь узкий класс алгоритмов. Отдельное от

процессора адресное пространство памяти, а точнее там их несколько разных видов памяти, сложная архитектура, еще более сложное распараллеливание и программная реализация. При этом остаются те же проблемы – пиковая производительность GPU еще более многократно превосходит возможности пропускной способности памяти, и достигаемый процент от пика стал еще более жалким. Но многократный выигрыш в скорости по сравнению с центральными процессорами вполне достижим, и это очень заманчиво. Поэтому приходится переделывать параллельные алгоритмы, портировать коды, еще больше их усложняя, осваивать гетерогенные вычисления, чтобы задействовать разнородные вычислительные ресурсы.

Продолжаем наше погружение во мрак HPC. За счет чего GPU могут быть так сильно быстрее процессора? Основная причина – GPU имеет свою встроенную память. У нее в разы более высокая пропускная способность, чем у памяти процессора. Сейчас (на момент написания текста) это обычно HBM2 (High-Bandwidth Memory), выдающая бэндвис (*рус. жарг.* от bandwidth) порядка 1 TB/s. Поскольку память интегрирована на устройстве, она имеет очень широкую шину. За один такт передается многократно большее количество байт данных. Процессор такую ширину себе позволить не может, так как его память живет отдельно на материнской плате (на него просто не влезет столько ножек, их и так уже обычно несколько тысяч).

Процессоры-мэникоры тоже осознали преимущество набортной памяти и начали наносить ответный удар. На них тоже появилась интегрированная память высокой пропускной способности. Первый широко распространенный пример тому – Intel Xeon Phi с набортной памятью MCDRAM (Multi-Channel DRAM). Более диковинные многоядерные сильно векторные устройства типа NEC SX-Aurora TSUBASA тоже обзавелись быстрой памятью HBM2 на борту. И вот, наконец, на вершине топа самых мощных суперкомпьютеров (см. https://www.top500.org/lists/top500/2020/06/) мы видим машинку на мэникорах (https://www.top500.org/system/179807/). 48-ядерные процессоры A64FX на основе архитектуры ARMv8 имеют на борту память HBM2 на 1 TB/s. Да, и там тоже используется векторное расширение на 512 бит, как и на Intel Xeon Phi. Набортная память хоть и быстрая, но ее – мало. Сейчас характерный размер – это 32 ГБ. Для такого мощного устройства это ни о чем. Поэтому, похоже, нас может ждать гибридизация памяти, как в случае Intel Xeon Phi – сочетание большой памяти низкой пропускной способности и ограниченной по объему памяти высокой пропускной способности. Программировать станет еще сложнее. Хотя еще есть надежды на спасение от гибридной памяти (с этим мы еще поразбираемся далее).

Очевидно, мэникоры тоже встали на темную сторону. Гонка за производительностью продолжается. Те системы и устройства, которые упоминаются выше в тексте, это по состоянию на сейчас. Какой может быть зоопарк вычислительных архитектур и суперкомпьютеров, когда вы будете это читать, даже страшно пытаться представить. Но хуже того. Этот зоопарк архитектур еще и порождает зоопарк средств разработки. Производители навязывают свои проприетарные фреймворки (*рус. жарг.* от framework), применимые только на их устройствах (как CUDA, например). Это еще более усложняет положение разработчика параллельных программ.

Системы усложняются, разработка усложняется, парадигмы параллельного программирования сужаются, множество совместимых с архитектурой алгоритмов тает на глазах. Резюмировать все вышесказанное можно очевидным выводом: когда пиковая производительность является самоцелью, в пределе будет система с бесконечно высокой производительностью, на которой, наконец, нельзя будет посчитать абсолютно ничего.

Звучит печально, но посмотрим на это с другой стороны. Высокопроизводительные вычисления очень востребованы. Множество отраслей промышленности и областей науки плотно сидит на софте для предсказательного моделирования. Как правило, очень дорогом софте. Вычислительный эксперимент на суперкомпьютере, про который мы далее поговорим, стал важным инструментом научно-технического прогресса. Так достаточно пафосно сформулировано? Для пущей мотивации вспомним также и смежные области – компьютерная графика (там многое близко к рассматриваемым в данном курсе сеточным методам), игрушки, базы данных, прочие задачи, связанные с обработкой данных большого объема. Там сильно важны параллельные технологии и производительность программного обеспечения. Значит, HPC – важное направление, знания в котором высоколиквидны.

А если что-то востребовано, но это что-то становится всё сложнее? Это значит, HPC программисты, параллельщики, хоть их работа и усложняется, становятся более ценными. При этом повышение порога вхождения снижает конкуренцию. Отметим этот факт, сделаем соответствующие выводы и, хитро улыбаясь, продолжим чтение.

1.2 Виды параллелизма

Наверное, все уже знают таксономию Флинна, которая классифицирует параллельные вычислительные архитектуры, используя четыре категории:

1) **SISD** – Single Instruction stream, Single Data stream;
2) **SIMD** – Single Instruction stream, Multiple Data streams;
3) **MISD** – Multiple Instruction streams, Single Data stream;
4) **MIMD** – Multiple Instruction streams, Multiple Data streams.

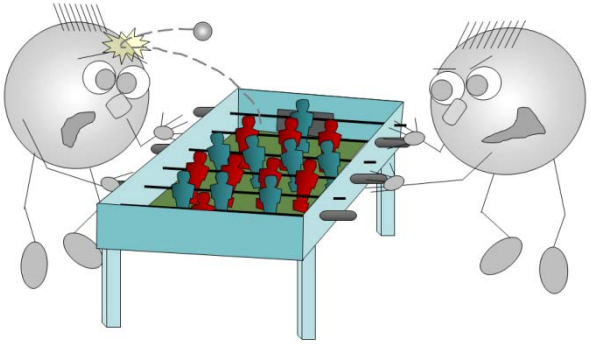
Далее нас будут интересовать только три вида: SISD, SIMD, MIMD.

SISD – а разве это параллельная архитектура? Существует расхожее мнение, что нет. Тех, кто придерживается такого мнения, "тычем носом" в понятие ILP – Instruction-Level Parallelism. С эти понятием далее будем подробно разбираться.

SIMD – это обычно имеется в виду векторный параллелизм, когда одна машинная команда выполняется для множественных наборов аргументов.

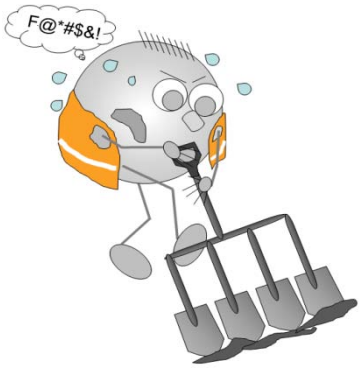
MISD – а что это за такая дичь? Действительно, сложно подобрать примеры такой системы. С некоторой натяжкой к этому виду можно отнести, например, компьютеры цифровой электродистанционной системы управления самолета. Эти компьютеры (например, на Airbus по два каких-нибудь Elevator Aileron Computer, на Boeing по три Primary Flight Computer, у Сухой Суперджет их тоже три штуки) работают параллельно, выполняют независимые потоки команд, но получают на вход идентичные наборы данных с органов управления. Один из компьютеров управляет полетом, другие его проверяют, сравнивают его результаты со своими результатами. В случае возникновения расхождений, если вдруг компьютер глюканет и решит покрутить "бочку", управление переходит к резервному компьютеру.

MIMD – наиболее распространенный вид параллелизма, к которому, во-первых, относятся многоядерные процессоры, многопроцессорные серверы, то есть параллельные системы с общей памятью, и, во-вторых, кластерные системы с распределенной памятью, то есть множественные компьютеры, объединенные сетью, у которых нет общего адресного пространства оперативной памяти. В первом случае это подвид **SM-MIMD** (Shared Memory MIMD), во втором – **DM-MIMD** (Distributed Memory MIMD).



Для лучшего понимания можно обратиться к примерам из реальной жизни. Показательным примером SIMD является настольный футбол, в котором на одной ручке крутится сразу несколько футболистов.

А вот несколько дорожных рабочих, копающих яму – это MIMD. Ну, или вот более знакомый москвичам пример: несколько рабочих выполняют разнородные операции – один отбойным молотком отковыривает недавно уложенный асфальт, другой выкорчевывает только недавно поменянные бордюры, третий циркуляркой пилит очередную плитку. Самый настоящий MIMD.



А вот если, например, один землекоп свяжет вместе сразу 4 лопаты и будет этим копать – это будет SIMD расширение землекопательного процессора. В общем-то, такое же бессмысленное и беспощадное, как и в обычном процессоре.

1.3 Зоопарк вычислительных архитектур

В высокопроизводительных вычислениях используются разные виды устройств. Некоторые достаточно диковинные. Приводить в пример конкретные современные модели, конечно, смысла мало. Через совсем небольшое время это станет перечислением музейных экспонатов. Можно попробовать в общих чертах изобразить общую суть.

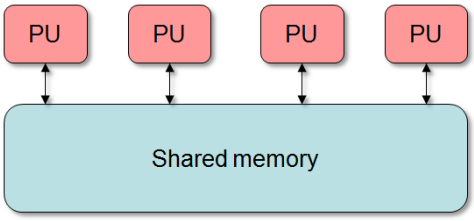
- Многоядерные серверные процессоры – CPU (Central Processing Unit). Параллелизм MIMD с общей памятью. Внутри ядер SISD параллелизм ILP – суперскалярность, внеочередное исполнение и VLIW; параллелизм SIMD – векторные расширения. Широко используются процессоры Intel, AMD. Есть еще процессоры IBM, процессоры на основе архитектуры ARM. Среди отечественных архитектур – Эльбрус, например.
- Существенно-многоядерные устройства (мэникоры) с набортной памятью высокой пропускной способности. Параллелизм такой же, как и в первом случае. Можно было бы просто сказать Intel Xeon Phi. Раньше их называли майками (*жарг.* от MIC – Many Integrated Core архитектуры Intel), теперь больше фаями (*жарг.* от Phi). Но, во-первых, эти устройства уже "ушли в закат". А, во-вторых, уже можно привести и другие аналоги, например, ARM-овые Fujitsu A64FX. Помимо набортной памяти, такие устройства имеют более простые ядра, но большего количества. Можно сказать, берут не умением, а числом. Эти два вида эволюционируют навстречу друг другу: в обычных CPU число ядер увеличивается в сторону мэникоров, а в мэникорах ядра становятся мощнее, что в пределе сходится к одному и тому же. Отличительным нюансом остаются гигабайты набортной памяти.
- Ultra-manycore устройства. Принцип примерно тот же. MIMD параллелизм. Но это уже сотни, а то и тысячи ядер. Но уже, возможно, совсем простых ядер. И со всякими другими упрощениями, такими как, например, отсутствие кэш-когерентности и прочие неприятности, осложняющие программирование. Например, долгое время держался на первой строчке мирового рейтинга суперкомпьютер Sunway TaihuLight на процессорах Sunway SW26010, не к ночи будь он помянут. Этот рекордно бесполезный (субъективное мнение автора) суперкомпьютер выдал аж 0.3% от пика на тесте HPCG. Потом удалось

довести до 0.4%, собрав, видимо, по всему миру лучших программистов машинных кодах. Другой пример – многотысячеядерники PEZY-SC. Вот тут есть про разные поколения с диаграмками https://en.wikichip.org/wiki/pezy/pezy-scx

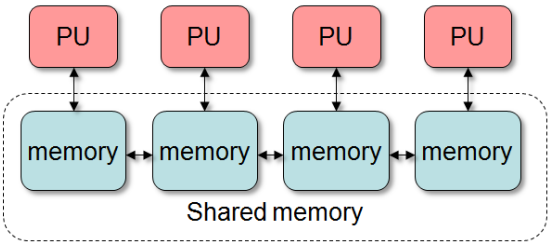
- Графические процессоры общего назначения – GP GPU (General Purpose GPU). Тут параллелизм заметно отличается. По Флинну это ближе всего к SIMD, но не SIMD, потому что еще и MIMD, но не совсем MIMD. Какой-то непонятный SIMD-o-MIMD. Или MIMD-o-SIMD. Назовем это устройствами, работающие в парадигме потоковой обработки. С этим мы будем разбираться долго и мучительно. Тут (сейчас) наиболее распространены устройства NVIDIA и AMD.
- Векторные процессоры. По сути те же многоядерники, возможно, с набортной памятью, но с очень широкими векторными расширениями. Пример такой архитектуры – устройства NEC SX-Aurora TSUBASA (192 операции с плавающей точкой за такт!) или их предшественники SX-ACE.
- Системы на кристалле – SoC (System-on-a-Chip). Вообще много что можно назвать SoC. Имеются в виду CPU с интегрированным графическим процессором. Некоторые устройства имеют GPU, поддерживающие вычисления в формате двойной точности, на которых вполне можно считать. Можно задействовать в вычислениях и ядра процессора и ресурсы GPU в качестве вычислительного сопроцессора (например, как тут – G.Oyarzun, R. Borrell, A. Gorobets, F. Mantovani, A. Oliva. Efficient CFD code implementation for the ARM-based Mont-Blanc architecture. Future Generation Computer Systems – The International Journal of eScience. Volume 79, Part 3, 2018. Pages 786-796. https://doi.org/10.1016/j.future.2017.09.029).
- Вычислительные ускорители FPGA (Field-Programmable Gate Array). Или по-нашему программируемая логическая интегральная схема (ПЛИС). В таких устройствах вычислительные подпрограммы "вживляются" в реконфигурируемую микросхему, также имеющую на борту быструю память. Такие устройства в данном курсе не рассматриваются. Подробности можно найти тут, например: Лацис А.О. Параллельная обработка данных. Издательство: М.: Академия. 2010. ISBN: 978-5-7695-5951-8. Там во второй части, главы 11 – 13.

1.4 Иерархия параллелизма

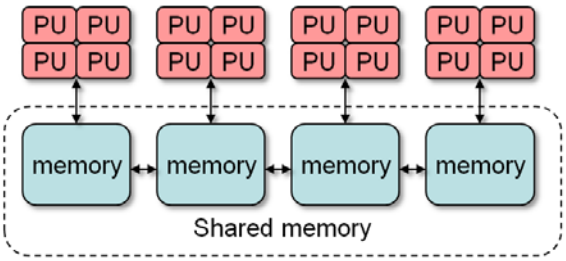
Современный суперкомпьютер сочетает различные виды параллелизма. Получается такая иерархия из нескольких уровней. Ну вот, возьмем, например, самое простое – один узел суперкомпьютера, обычный, не гибридный, то есть без ускорителей. В нем есть сколько-то процессорных ядер, у которых общая оперативная память. Это можно изобразить как систему с общей памятью – SM-MIMD: несколько PU – Processing Unit, то есть обрабатывающие устройства, соединены с общей памятью.



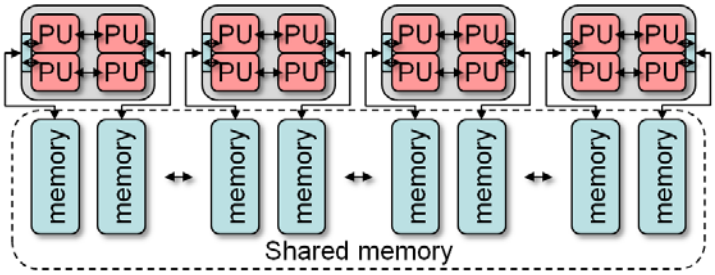
Все просто. Но мы-то знаем, а если не знаем, то скоро узнаем, что доступ к памяти неоднородный. На узле обычно не один процессор. А у каждого процессора свой контроллер памяти. Чтобы память была общей, имеется соединение между памятью на разных котроллерах по некоторой шине. Какая-то часть памяти ближе получается "ближе" к процессору, какая-то – "дальше". Возникает неоднородность – NUMA фактор. Нарисуем по-другому:



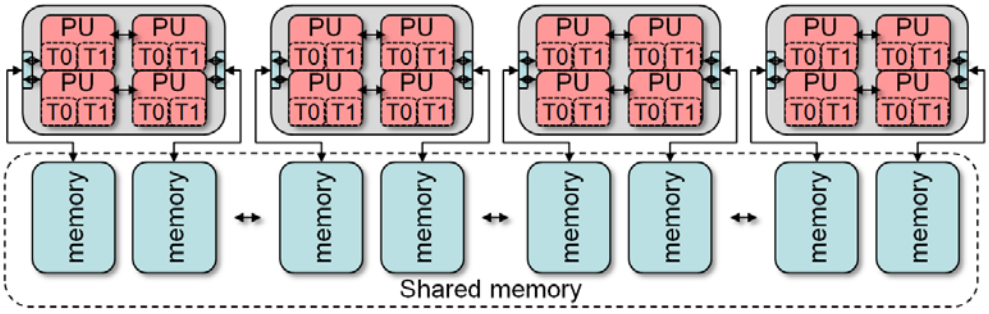
Так уже ближе к жизни. Но мы забыли, что процессоры многоядерные. Подправим картинку:



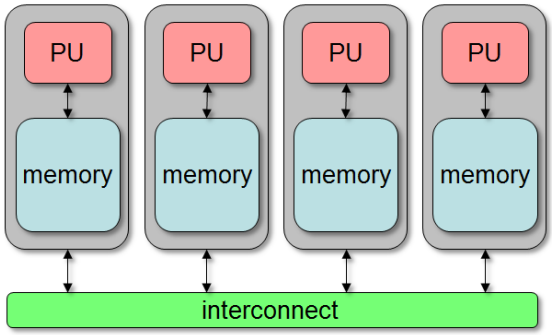
А еще вспомним, что у процессоров обычно более одного контроллера памяти, а ядра в процессорах связаны между собой по некоторой топологии, и не все ядра одинаково близки ко всем контроллерам. Еще приблизим картинку к реальности:



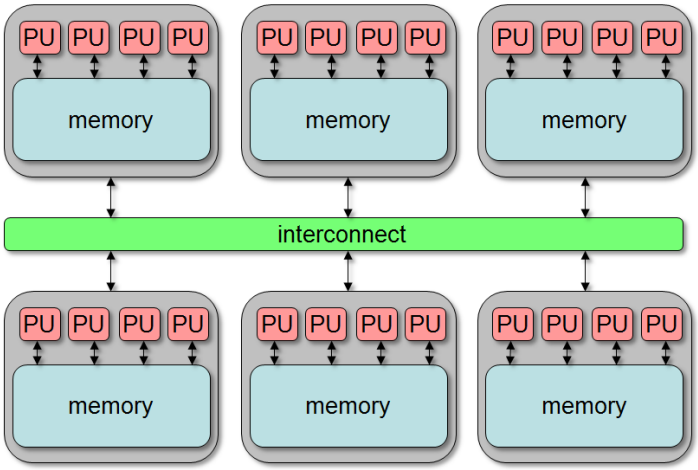
Как-то так, что ли? Ой, да, вспомним еще, что ядра – многопоточные, могут аппаратно поддерживать выполнение несколько потоков в режиме SMT (Simultaneous MultiThreading – одновременная многопоточность). Дорисуем каждому PU по паре T – threads. Получится вот какая-то такая совсем уже жуть:



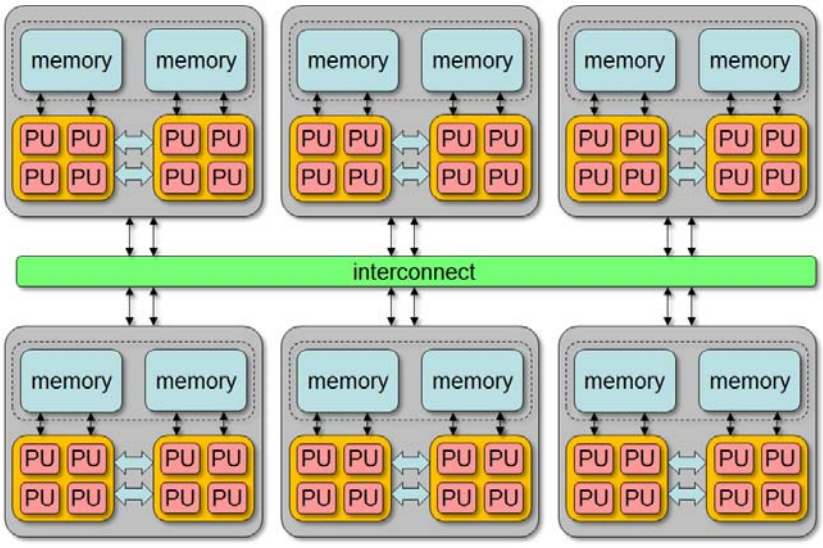
Вот уже внутри узла получается что-то похожее на какую-то иерархию. А теперь посмотрим на систему "сверху", с верхнего уровня параллелизма. Попробуем изобразить весь суперкомпьютер, то есть кластерную вычислительную систему. Это будет параллельная система с распределенной памятью – DM-MIMD. Нарисуем несколько PU, но теперь у каждого своя память, а соединены эти PU по коммуникационной сети:



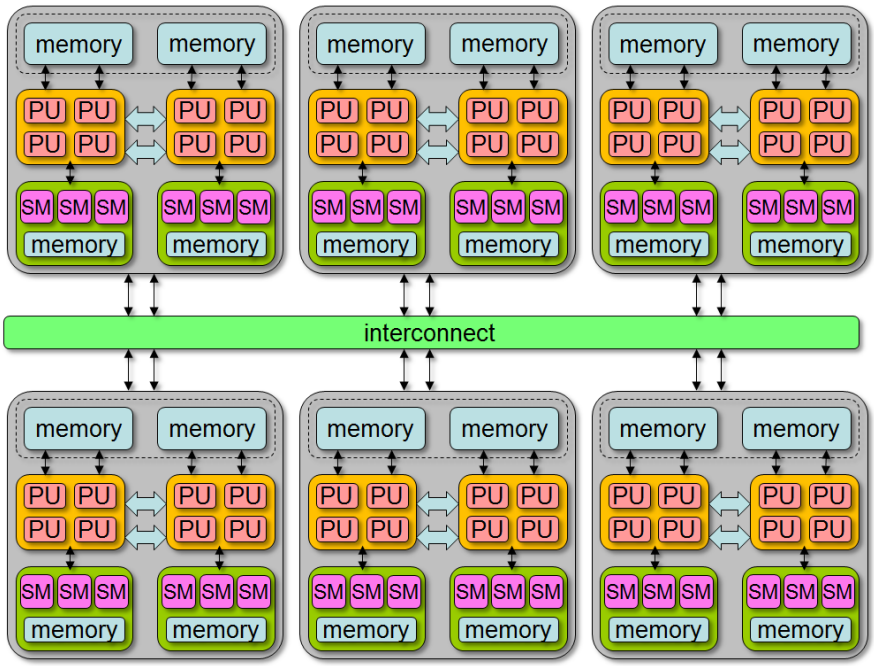
Теперь вспомним, что узлы – многоядерные. Дорисуем им по много PU:



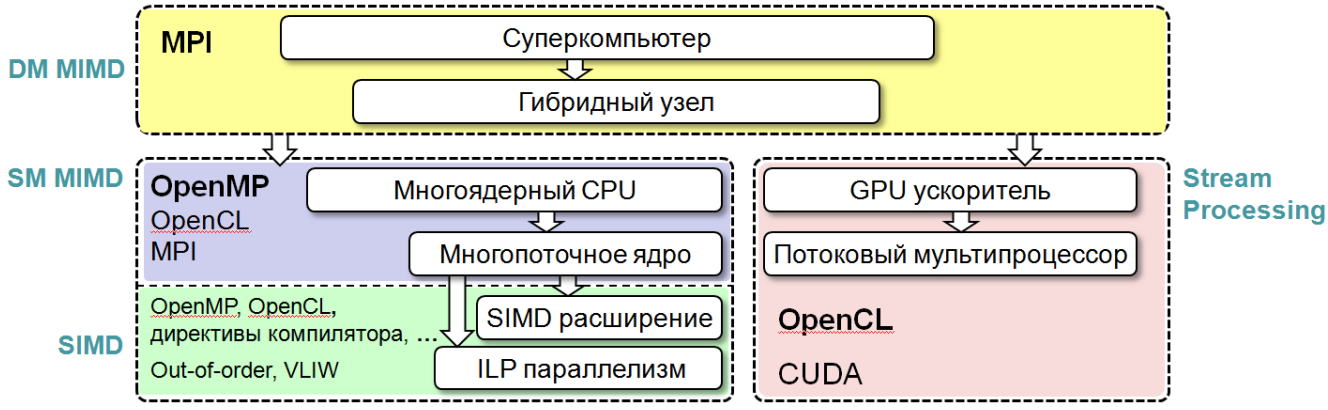
А теперь вспомним, что узлы – многопроцессорные, и память там общая, но с NUMA. А еще на узлах обычно по несколько сетевых адаптеров, поэтому нарисуем узлам хотя бы по две стрелочки в интерконнект:



А теперь самое страшное. Вспомним, наконец, что суперкомпьютеры в основном гибридные, и дорисуем узлам ускорители. Пусть это будут такие зелененькие штучки с множественными SM (Stream Multiprocessor). Да, не забудем, что ускорителям надо дорисовать свою собственную память. Получится нечто такое, на чем, пожалуй, лучше остановить творческий порыв в сфере изобразительного искусства:



В соответствии с иерархией параллелизма вычислительной системы выстраивается параллельный алгоритм. Там тоже получается иерархия, в которой будут разные параллельные модели и виды параллелизма, разные средства разработки. Получается целый software stack из разных фреймворков, интерфейсов программирования, библиотек:



На верхнем уровне работает DM-MIMD распараллеливание, в котором MPI (Message Passing Interface) обеспечивает взаимодействие между процессами параллельной программы на разных узлах суперкомпьютера. Внутри узлов используется многопоточное SM-MIMD распараллеливание средствами, например, OpenMP. Внутри ядер на нижнем уровне работает SIMD распараллеливание. На ускорителях работает свое отдельное распараллеливание средствами, например, OpenCL или CUDA. Со всеми этими уровнями параллелизма будем подробно разбираться в следующих главах.

1.5 Вычислительный эксперимент

Рост производительности вычислений, естественно, открывает все более широкие возможности для применения вычислительного эксперимента в промышленности и науке. Вычислительного эксперимент – это когда в компьютерном расчете предсказываются характеристики какого-то изделия в реальности, особенности протекания какого-то физического процесса. Расширение возможностей предсказательного суперкомпьютерного моделирования, в свою очередь, разгоняет научно-технический прогресс. Самолеты весят меньше, летают тише, долетают дальше, потребляют меньше топлива, становятся более безопасными. Тем же курсом следуют и все прочие вертолеты, теплоходы, автомобили, ракеты-носители космического назначения. Благодаря научному и инженерному вычислительному софту нефть добывается эффективнее, медицина лечит лучше, атомные электростанции становятся безопаснее, и много чего еще становится лучше, быстрее, выше, сильнее.

Как это работает? Предположим, конструктор конструирует новую модель автомобиля. Он думает, а что будет, если куда-то сюда приварить железку, а отсюда две железки выпилить? Станет ли оно прочнее и легче? Можно построить прототип и разбить его на краш-тесте. Дорого и долго. А можно взять инженерный софт для расчета прочности и деформации твердого тела, понажимать кнопки, пошевелить мышкой и получить результат: кузов сложился, а двигатель пролетел через салон в багажник. О, нет, непорядок. Не будем выпиливать ту железку. И не нужен дорогой краш-тест. Положим, другой конструктор придумывает новый самолет. Он думает, а что будет с аэродинамическими силами, если крыло вот так вот немного изогнуть, немного сместить и, цитируя классика, воткнуть и там два раза повернуть? Можно изготовить такую модель и продуть в аэродинамической трубе. А можно взять инженерный софт, понажимать кнопки, пошевелить мышкой и получить результат: подъемная сила упала, сила сопротивления выросла, а критический угол атаки стал такой маленький, что самолет свалится в штопор на первой воздушной ямке. Нет, непорядок. Не будем так гнуть крыло. Конструктор расстроен, так попробовал изогнуть крыло – хуже, сяк выгнуть – еще хуже. Что же делать? Тогда конструктор берет всю кучу мыслимых вариантов, как крыло можно погнуть, подвинуть, повернуть, и что-либо еще нехорошее с ним сделать. Он задает софту изменяемые параметры геометрии крыла и диапазоны этих изменений, запускает серию расчетов и идет домой отдыхать. Компьютер за него будет пробовать так крыло погнуть, сяк, туда сместить, сюда, обсчитает сотни вариантов изменения геометрии и выберет лучшие. Для каждого варианта геометрии будет выполнен расчет, который предскажет аэродинамические характеристики крыла. Это и есть HPC (High Performance Computing). А выбирать варианты геометрии будет специальный софт – оптимизатор. Он будет крутить какой-нибудь алгоритм, по которому будет варьировать параметры геометрии. Например, генетический алгоритм. В параметры текущего варианта геометрии вносятся изменения, обсчитывается "поколение" – набор разных вариантов изменений, из них выбираются лучшие варианты, и эти варианты "оставляют потомство". На следующем поколении, полученном из лучших вариантов предыдущего, опять произвольным образом вносятся изменения в параметры геометрии. Так гоняются массовые расчеты на суперкомпьютере, поколение за поколением, оптимизируется геометрия крыла. А конструктор в это время культурно отдыхает в баре с друзьями. Примерно так и происходит НИОКР (научно исследовательские и опытно-конструкторские работы) в куче отраслей промышленности. Фармацевты и химики тоже обычно не ковыряют вручную пинцетом разные варианты молекул, за них это делает суперкомпьютер. Он переберет миллионы вариантов молекул, просчитает миллионы вариантов химических реакций и скажет –

вот, всё перепробовал, опять получилось C_2H_5OH. Нефтяник тоже не будет сверлить скважины где попало, чтобы понять, как лучше выкачать нефть из тамошних глубоких недр. Он загонит результаты геологоразведки в инженерный софт. Софт по этим данным построит модель всяких там пластов, учтет всю эту пористость, проницаемость, геометрию, где там нефть, куда пихать воды, переберет варианты и просчитает, куда надо правильно забуриться. Атомщик, конструирующий атомную станцию, тоже не станет каждый раз проверять на реальном реакторе, а хватит ли выбега ротора турбогенератора для питания главного циркуляционного насоса? А то мало ли, вдруг не хватит. Он возьмет расчетный софт, всячески аттестованный и сертифицированный, и промоделирует на компьютере происходящие на реакторе процессы. Можно привести еще много примеров, но мы же боремся за краткость изложения. Да и общий смысл, наверное, начинает прослеживаться. Далее рассмотрим конкретный вид приложений – CFD (Computational Fluid Dynamics), а именно моделирование механики сплошной среды сеточными методами.

1.6 Суперкомпьютерное моделирование сеточными методами

В данном разделе рассказывается, как работает суперкомпьютерное моделирование на примере задач вычислительной газовой динамики. Этот раздел можно считать научпопом и при отсутствии любопытства пропустить. Вопросы на экзамене по содержанию этого раздела маловероятны.

1.6.1 Как это работает

Предположим, что читатель не проходит обучение на кафедре Вычислительных методов или Математической физики. Исходя из этого предположения, максимально упростим пояснения о том, как это всё работает. Пусть моделируется течение жидкости или газа. Это течение моделируется некоторой системой уравнений. Ну ладно, специально для читателей с вышеупомянутых кафедр, течение сжимаемого вязкого газа описывается системой безразмерных уравнений Навье – Стокса, которая в дивергентной форме (в виде законов сохранения) относительно вектора консервативных переменных $\mathbf{Q}=\left(\rho,\rho u,\rho v,\rho w,E\right)^T$ – плотности, трёх компонент импульса и полной энергии, имеет вид

$$\frac{\partial \mathbf{Q}}{\partial t}+\frac{\partial \mathbf{F}_1(\mathbf{Q})}{\partial x}+\frac{\partial \mathbf{F}_2(\mathbf{Q})}{\partial y}+\frac{\partial \mathbf{F}_3(\mathbf{Q})}{\partial z}=\frac{\partial \mathbf{F}_1^{NS}(\mathbf{Q})}{\partial x}+\frac{\partial \mathbf{F}_2^{NS}(\mathbf{Q})}{\partial y}+\frac{\partial \mathbf{F}_3^{NS}(\mathbf{Q})}{\partial z}.$$

Конвективные потоки:

$$\mathbf{F}_1(\mathbf{Q})=\left(\rho u,\ \rho u^2+p,\ \rho uv,\ \rho uw,\ u(E+p)\right)^T,$$
$$\mathbf{F}_2(\mathbf{Q})=\left(\rho v,\ \rho vu,\ \rho v^2+p,\ \rho vw,\ v(E+p)\right)^T,$$
$$\mathbf{F}_3(\mathbf{Q})=\left(\rho w,\ \rho wu,\ \rho wv,\ \rho w^2+p,\ w(E+p)\right)^T,$$

где полная энергия $E=\rho\left(u^2+v^2+w^2\right)/2+\rho\varepsilon$, а ε – внутренняя энергия.

Вязкие потоки:

$$\mathbf{F}_1^{NS}=\left(0,\tau_{xx},\tau_{xy},\tau_{xz},u\tau_{xx}+v\tau_{xy}+w\tau_{xz}-q_x\right)^T,$$
$$\mathbf{F}_2^{NS}=\left(0,\tau_{yx},\tau_{yy},\tau_{yz},u\tau_{yx}+v\tau_{yy}+w\tau_{yz}-q_y\right)^T,$$
$$\mathbf{F}_3^{NS}=\left(0,\tau_{zx},\tau_{zy},\tau_{zz},u\tau_{zx}+v\tau_{zy}+w\tau_{zz}-q_z\right)^T.$$

Компоненты вязкого тензора напряжений τ_{ij} и вектора теплового потока q_i имеют вид:

$$\tau_{xx}=\mu\left(2\frac{\partial u}{\partial x}-\frac{2}{3}\mathrm{div}\mathbf{u}\right),\quad \tau_{yy}=\mu\left(2\frac{\partial v}{\partial y}-\frac{2}{3}\mathrm{div}\mathbf{u}\right),\quad \tau_{zz}=\mu\left(2\frac{\partial w}{\partial z}-\frac{2}{3}\mathrm{div}\mathbf{u}\right),$$

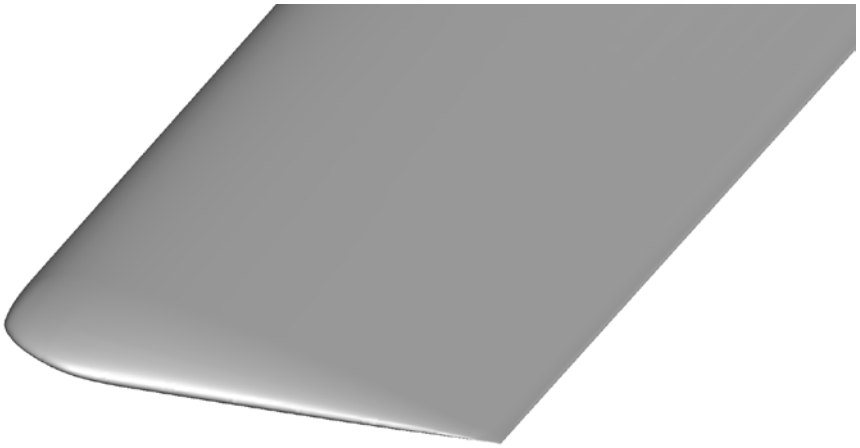
$$\tau_{xy}=\tau_{yx}=\mu\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right),\quad \tau_{xz}=\tau_{zx}=\mu\left(\frac{\partial u}{\partial z}+\frac{\partial w}{\partial x}\right),\quad \tau_{yz}=\tau_{zy}=\mu\left(\frac{\partial v}{\partial z}+\frac{\partial w}{\partial y}\right),$$

$$q_x=-\frac{\gamma\mu}{\mathrm{Pr}}\frac{\partial\varepsilon}{\partial x},\quad q_y=-\frac{\gamma\mu}{\mathrm{Pr}}\frac{\partial\varepsilon}{\partial y},\quad q_z=-\frac{\gamma\mu}{\mathrm{Pr}}\frac{\partial\varepsilon}{\partial z},$$

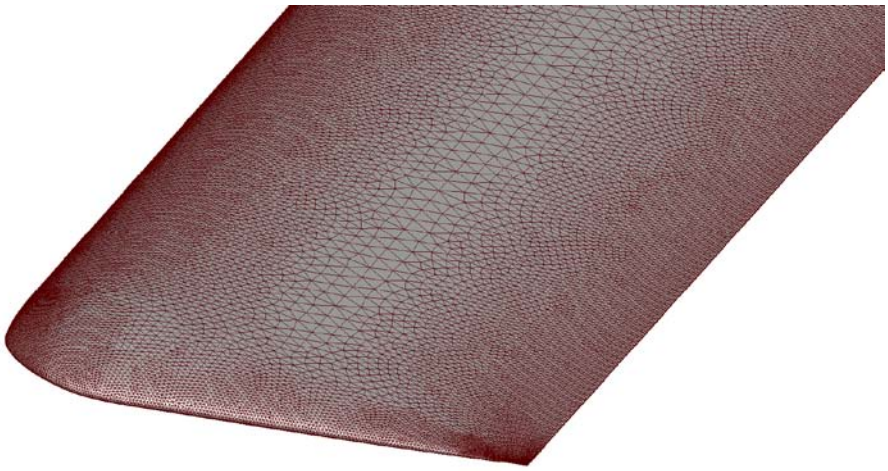
где μ – коэффициент динамической вязкости, $\mathbf{u}=(u,v,w)$ – вектор скорости.

Система уравнений Навье – Стокса замыкается уравнением состояния совершенного газа $p=\rho\varepsilon(\gamma-1)$, где γ есть показатель адиабаты. Вроде так. Наверняка где-нибудь ошибся в этих формулах. Ну да ладно. Пожалуй, хватит формул. Перейдем к сути.

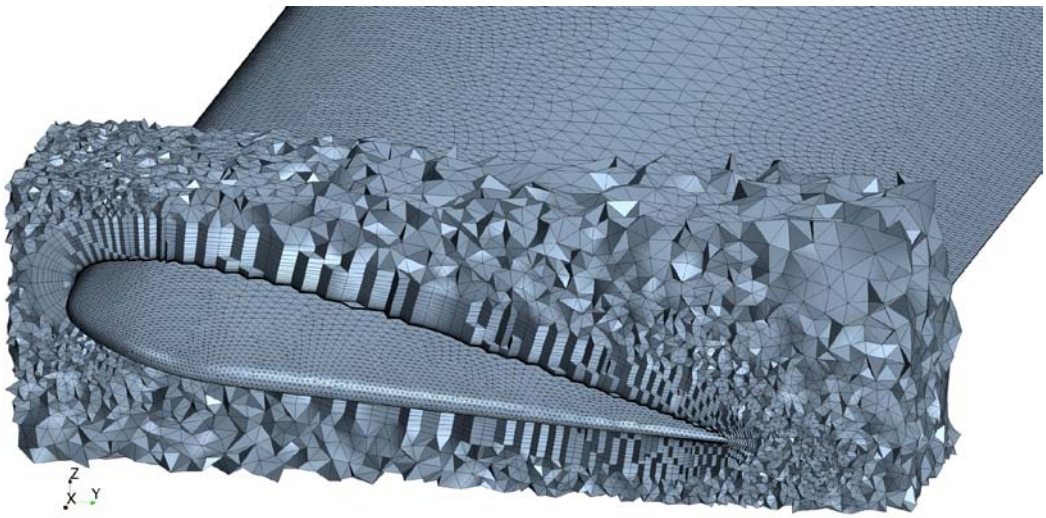
Пусть моделируется течение газа вокруг твердого тела. Пусть этим телом будет, например, лопасть несущего винта вертолета. Конструктор построил геометрию лопасти в CAD (Computer-Aided Design) системе. Вот, например, законцовка этой лопасти.



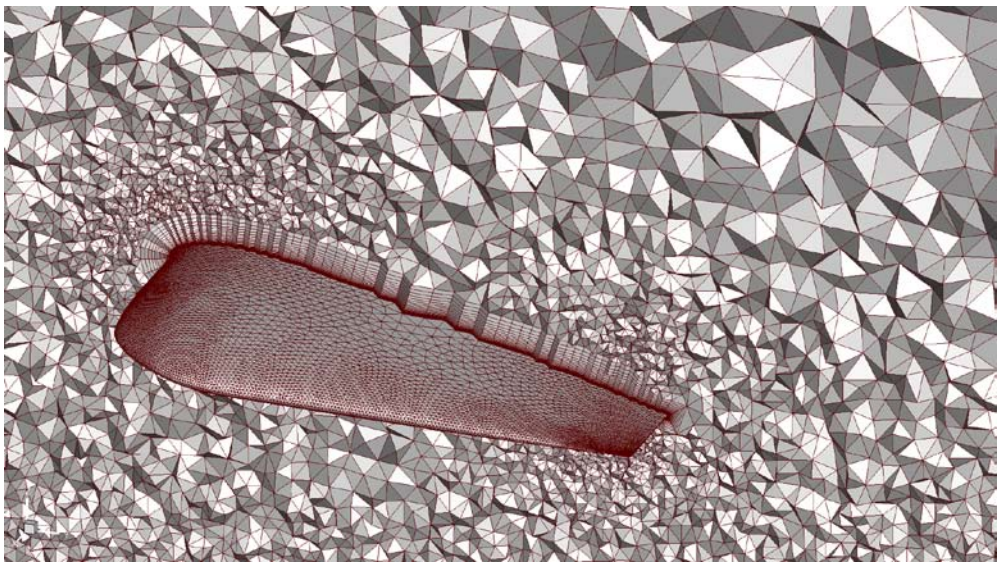
Затем геометрия подается на вход сеточному генератору. Эта программа построит на поверхности лопасти триангуляцию. У вас же был курс Машграфа? Это почти как в компьютерной графике, только сетка получается намного мельче, чем нужно для графики.



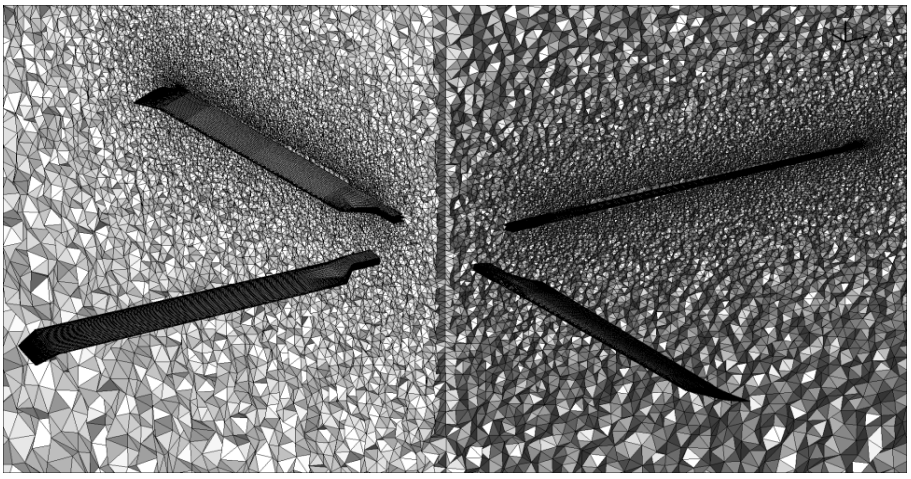
От триангуляции формируется объемная сетка, покрывающая расчетную область. Сетка состоит из сеточных элементов – тетраэдров, призм, пирамид, гексаэдров, например. Сеточными элементами заполняется весь моделируемый объем вокруг лопасти.



Вблизи поверхности лопасти – зона пограничного слоя, где сетка становится очень мелкой. В этой зоне образуется сдвиговый слой, ламинарное пристеночное течение переходит в отрывное турбулентное. Там требуется очень маленький шаг сетки по ортогональному поверхности направлению. Видно, как сильно уменьшается высота призм вблизи поверхности.

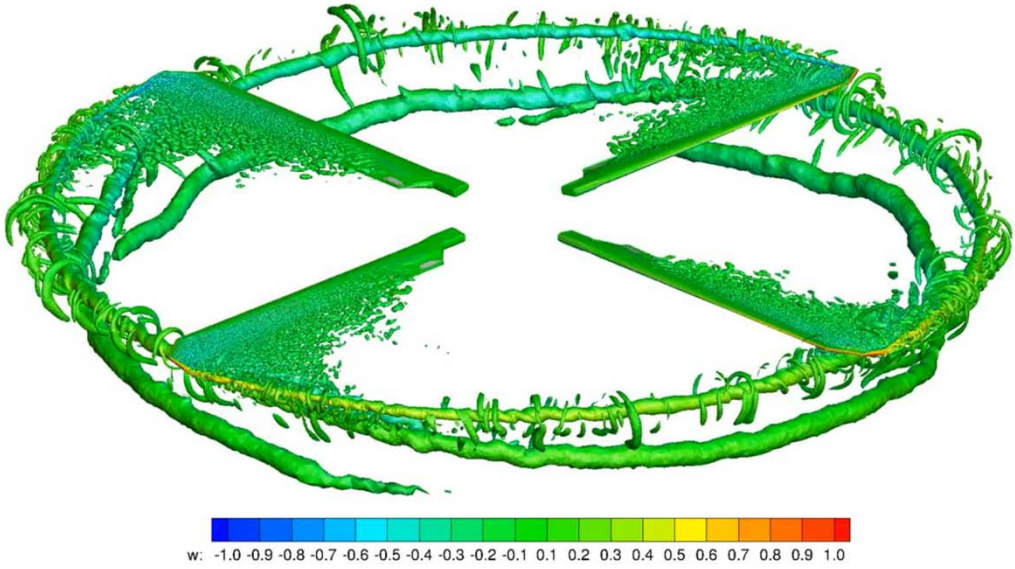


Так весь объем вокруг винта поделился на ячейки сетки – сеточные элементы.

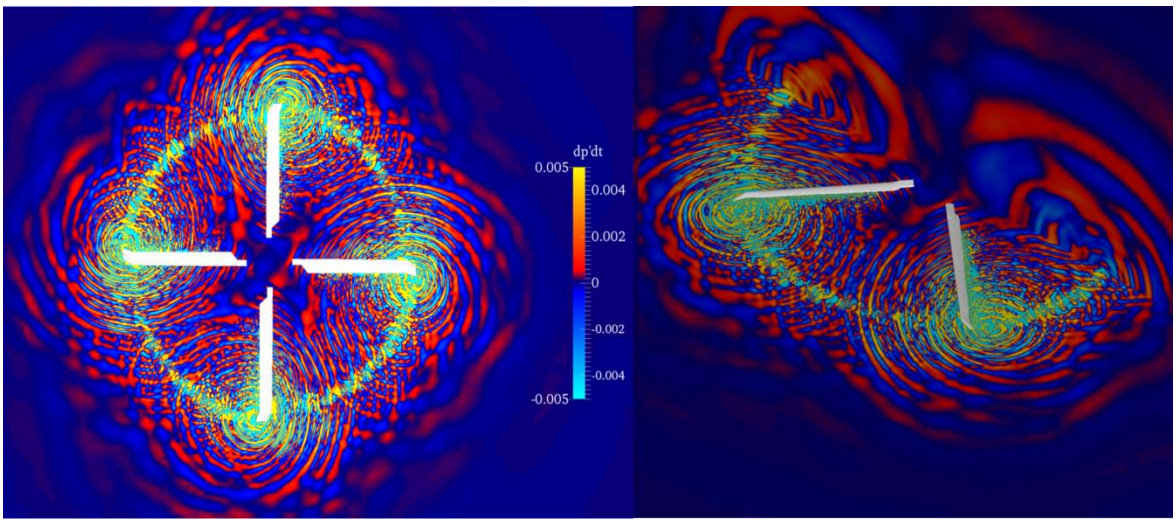


На этой сетке выполняется дискретизация уравнений Навье – Стокса, описывающих течение. В каждой ячейке сетки задаются физические величины – плотность, три компоненты импульса и полной энергия. Даже для простоты пусть это будет плотность, три компоненты вектора скорости и давление. Получается примерно как в ПЗС матрица в цифровом фотоаппарате, в которой в каждом пикселе заданы три компоненты цвета. Ну а тут точно также, только не цвета и не на плоскости, а в объеме (и не в покер, а в преферанс, и не выиграл, а проиграл). Ячейками сетки могут быть либо сами сеточные элементы, либо ячейки могут строиться вокруг сеточных узлов, но сути это не меняет.

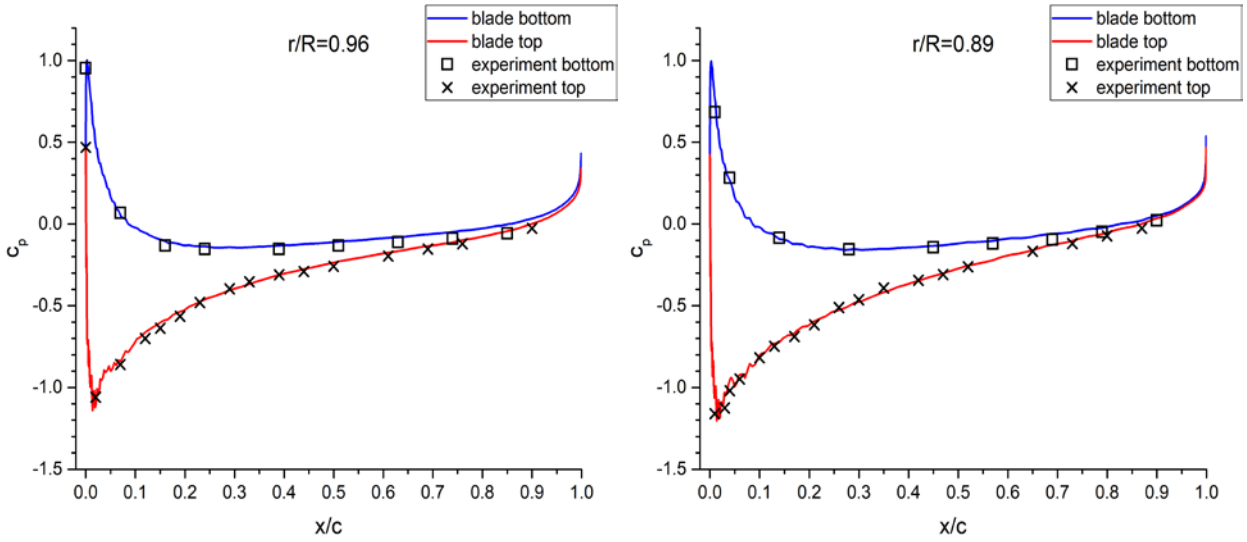
Итак, у нас есть миллионы ячеек сетки, в которых заданы по 5 переменных. Выполняется интегрирование по времени: время продвигается вперед на какую-то (очень) малую величину. Для всех ячеек сетки проводится расчет, в котором определяется, сколько жидкости или газа втекло-вытекло через каждую из граней ячеек. Узнав, сколько из каждой ячейки куда перетекло, вычисляем значение наших физических величин в каждой ячейке в новый момент времени, т.е. на новом временном слое. Продвигаясь вперед такими шагами по времени моделируется динамика течения, получается турбулентное течение вокруг вращающегося несущего винта вертолета.



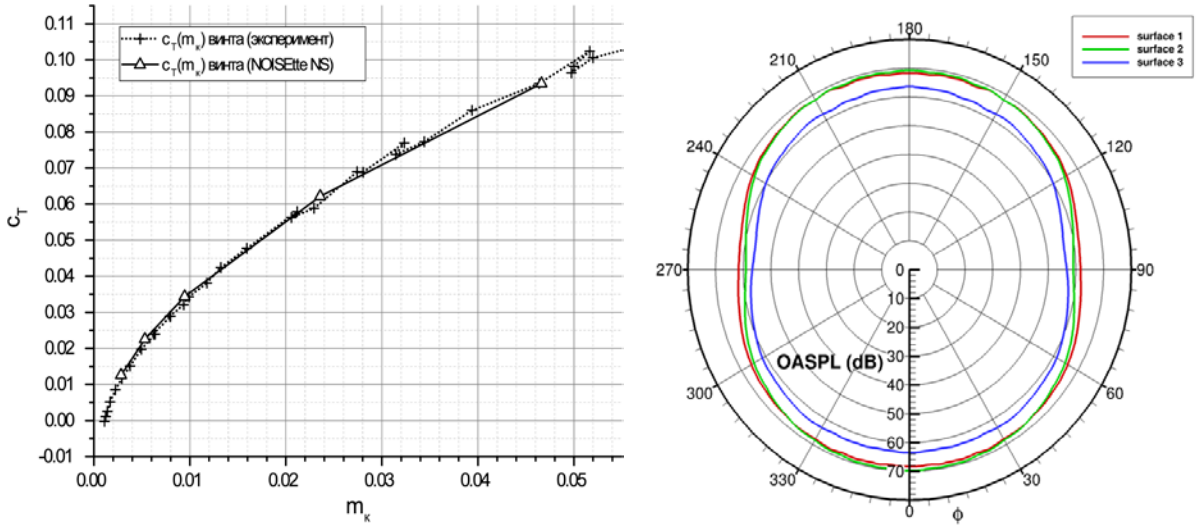
Воспроизвелись даже акустические поля, можно видеть шум, возникающий от взаимодействия турбулентных структур с законцовкой лопасти.



Конструктор, вернувшись на работу после тяжелых выходных, открыл результаты расчета и получил аэродинамические и акустические характеристики своего винта. Распределение коэффициента давления по поверхности лопасти,



аэродинамические силы – коэффициент тяги винта при различных углах установки лопасти, поляры винта (соотношение коэффициента тяги и момента – график ниже слева), диаграммы направленности шума (справа) и т.д.

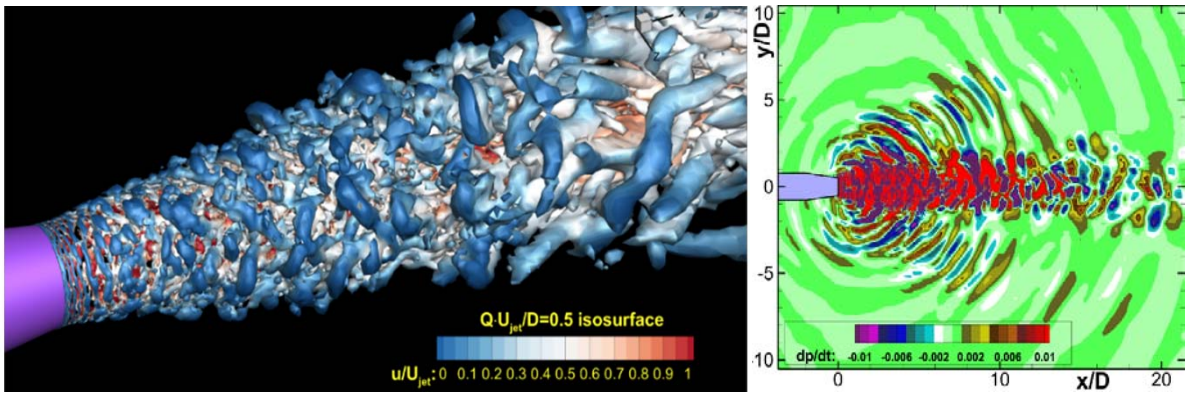


Довольный результатами конструктор, конечно, потом посмотрит, сколько прожглось процессорного времени на суперкомпьютере, посчитает, сколько это стоит, и подумает с тоской, что лучше бы заказал изготовление модели для натурного эксперимента. Но на то мы и изучаем этот курс, чтобы расчеты с вашей помощью становились эффективнее и дешевле.

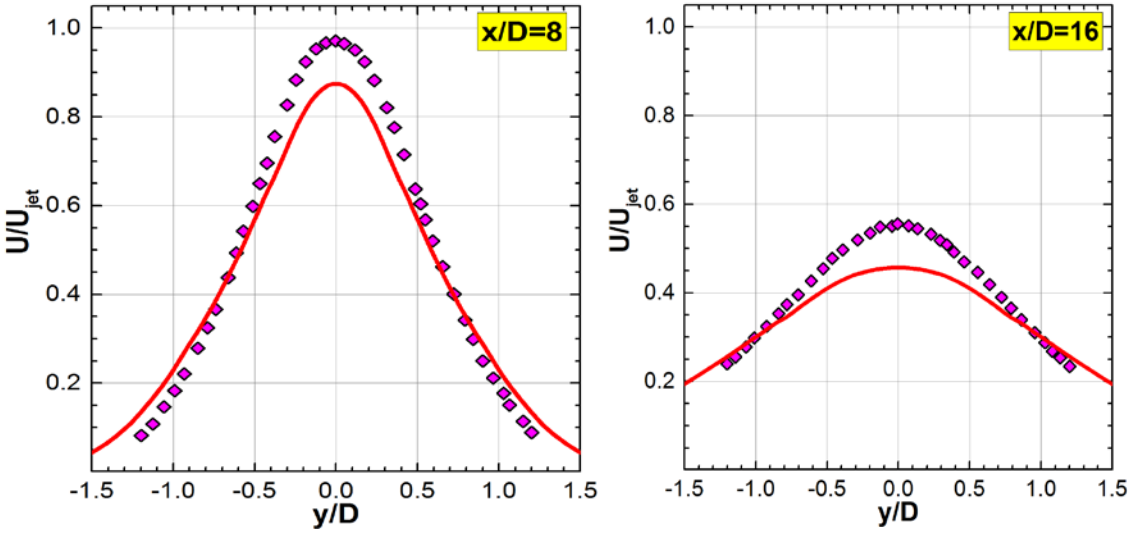
1.6.2 Пространственное разрешение и стоимость расчета

Почему, чтобы узнать правду, всё-таки зачастую проще продуть модель в трубе или сгонять полетать летающую лабораторию? Разберемся, откуда берется такая ресурсоемкость вычислительного эксперимента и зачем нужны суперкомпьютеры.

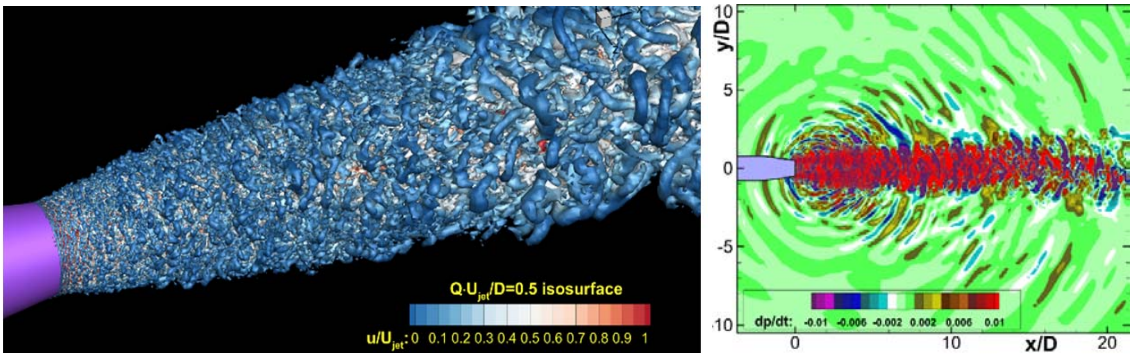
Чем лучше пространственное разрешение сетки, т.е. чем больше ячеек, этих "пикселей" в которых заданы переменные, тем точнее описывается течение и ближе к реальности результат. Вот, например, считаем мы зачем-то реактивную струю ($Re=1.1\cdot10^6$, $M=0.9$), взяли сетку на полтора "мегапикселя" – из 1.5 млн ячеек. Посчитали, получили какую-то картину течения.



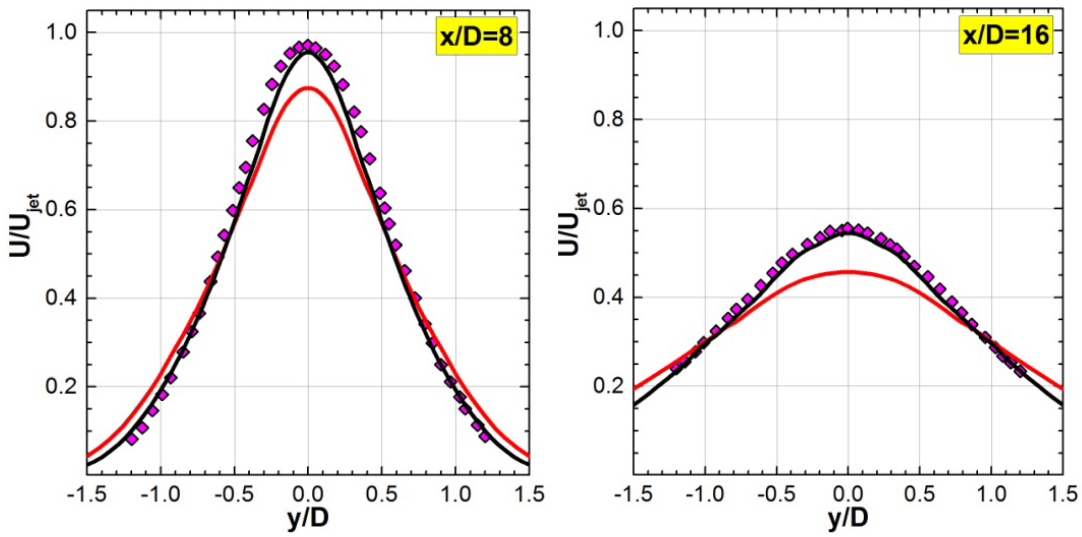
Видим турбулентные структуры, даже какие-то звуковые волны на правой картинке. Но это модельная конфигурация, у нее есть эталонное решение. Сравним его результатами расчета.



Вот, например, показан профиль скорости по линии, перпендикулярной оси струи, на разном удалении от сопла. Что-то результаты вообще не в дугу. Красная линия расчета сильно далека от ромбиков эксперимента. Берем сетку поподробнее, на 9 "мегапикселей". Сразу видно, что появились более мелкие структуры течения.



А что стало с графиками? Красная линия – было на сетке 1.5 млн ячеек, черная линия – стало на сетке из 9 млн ячеек.



Результаты расчета хорошо прилегли на эксперимент. Дело было в недостаточном для данного физического процесса сеточном разрешении. Для более сложных конфигураций нужны десятки миллионов ячеек. Для расчетов крупных частей летательного аппарата, таких как крыло самолета с выпущенной механизацией, могут потребоваться сотни миллионов, а то и миллиарды ячеек. Чтобы численно воспроизвести течение вокруг самолета или вертолета целиком (и получить близкий к реальности результат, а не просто никчемные цветные картинки!), может понадобиться сетка из десятков миллиардов ячеек и десятки (а то и сотни) миллионов процессорных часов. А откуда берутся требования к сеточному разрешению?

1.6.3 Моделирование турбулентности

Турбулентное течение состоит из вихрей различных размеров. Крупные вихри переходят в более мелкие, те в еще более мелкие, в еще-еще более мелкие, в еще совсем более мелкие, и так масштабы течения уменьшаются, пока не дойдут до Колмогоровского масштаба, на котором кинетическая энергия движения жидкости или газа переходит в тепло, во внутреннюю энергию, за счет вязкости среды. Происходит переход кинетической энергии по каскаду от крупных вихревых структур течения к более и более мелким.

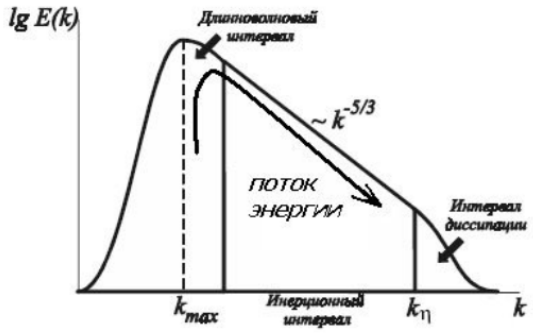


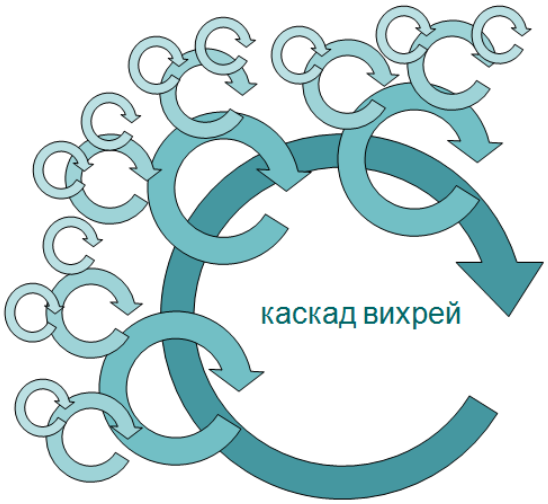
Рис. 1. Энергетический спектр изотропной трехмерной турбулентности.

То есть, чем больше в изучаемом течении перепад масштабов, тем больше нужно сеточных ячеек. У течения есть характеристика – число Рейнольдса (*Re*). Оно, по сути, определяет перепад масштабов от крупных структур до самых мелких, после которых происходит диссипация в тепло. Чем больше размер объекта или чем больше скорость течения или чем меньше вязкость среды – тем больше это число. Соответственно, тем более высокое нужно пространственное и временное разрешение.

Прямое численное моделирование – DNS (Direct Numerical Simulation). Если мы будем разрешать сеткой все релевантные масштабы течения, то это будет DNS. Это самый дорогой подход. Но этот подход и самый точный.

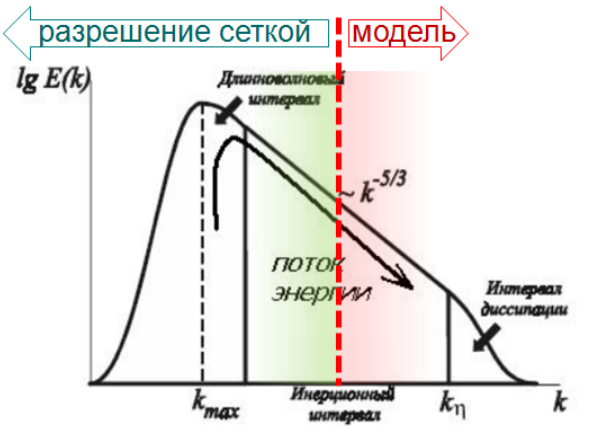


Энергетический спектр изотропной трехмерной турбулентности.

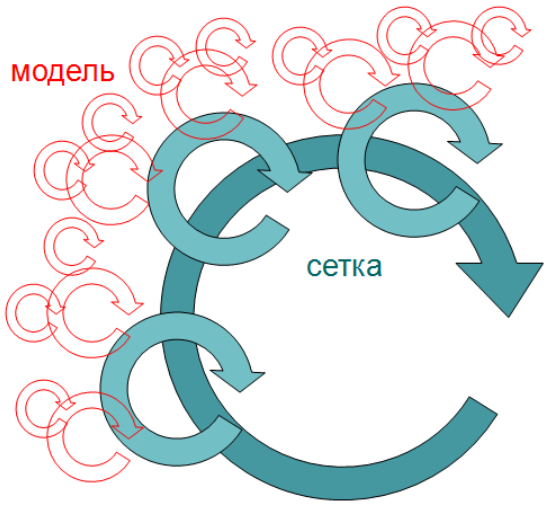


Такие расчеты, как правило, носят фундаментальный характер. Обычно это штучные расчеты, часто рекордные, в которых исследуются модельные конфигурации течения, сферические кони в вакууме. Целью таких расчетов является изучение физического процесса, получение надежных эталонных данных для разработки и визуализации моделей турбулентности. Ну, чтобы было, откуда взять ромбики на графиках, с которыми сравнивать свои кривые не в дугу результаты расчетов.

Метод крупных вихрей – LES (Large Eddy Simulation). Пусть сеткой мы можем разрешить только крупные структуры, а на мелкие структуры у нас не хватит вычислительных ресурсов. Обрубим в каком-то месте график энергетического спектра (по оси X). Мелкие структуры будем имитировать некоторой моделью, которая воспроизведет переход энергии дальше по каскаду, не разрешая сами структуры. Модель подсеточного масштаба заменит действие вихрей, меньших по размеру, чем ячейки сетки, некоторой вычисляемой величиной – вихревой вязкостью, с добавлением которой энергия с крупных вихрей будет соответствующим образом отводиться.

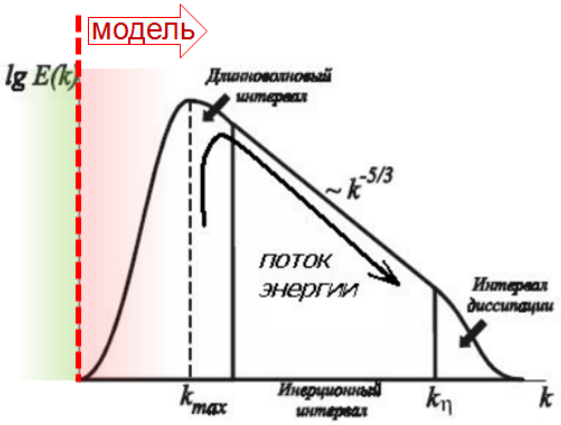


Энергетический спектр изотропной трехмерной турбулентности.

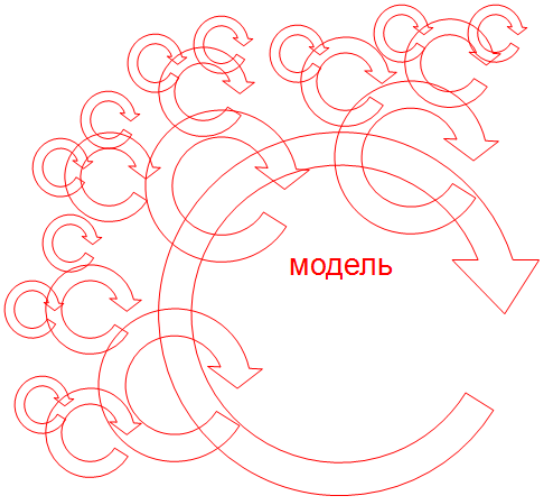


Осредненные уравнения – RANS (Reynolds Averaged Navier – Stokes). Если вообще все масштабы течения загнать в модель, то получится стационарный расчет RANS. Все масштабы

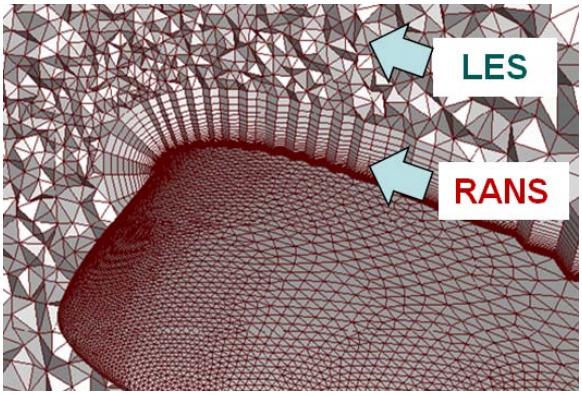
течения мы отдали на откуп модели и получили стационарное осредненное решение – как если бы в каждой ячейке был датчик измерений физических величин, и мы бы усредняли его показания на длительном интервале времени.



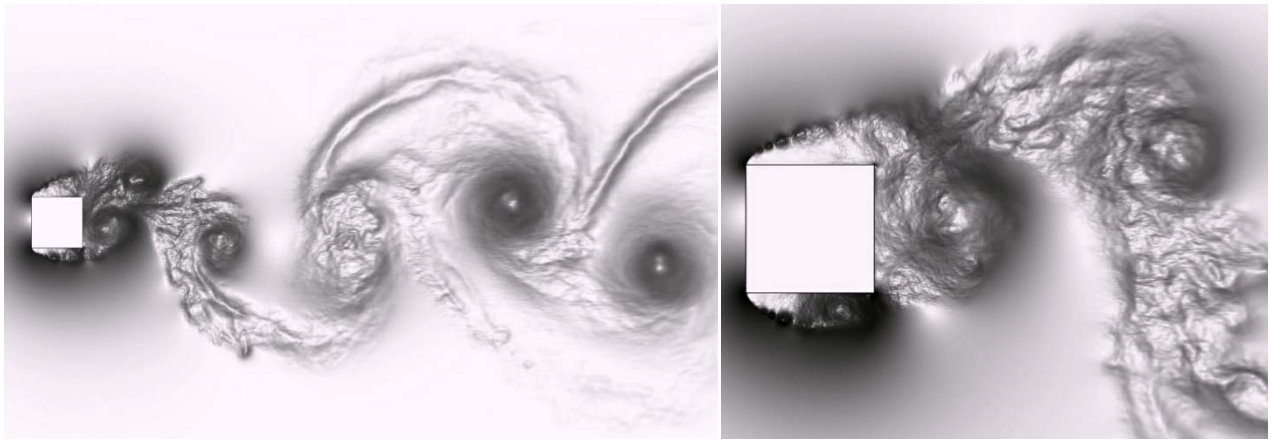
Энергетический спектр изотропной трехмерной турбулентности.



Метод отсоединенных вихрей (DES - detached eddy simulation) – сочетание LES и RANS. В зоне пограничного слоя, совсем вблизи поверхности твердого тела, где еще нет турбулентности, но нужен очень маленький шаг сетки в ортогональном поверхности направлении для корректного моделирования сдвигового слоя, будем использовать RANS. А в области отрывного течения, где уже гуляют турбулентные вихри, будем использовать LES. Выходит заметная экономия сетки.

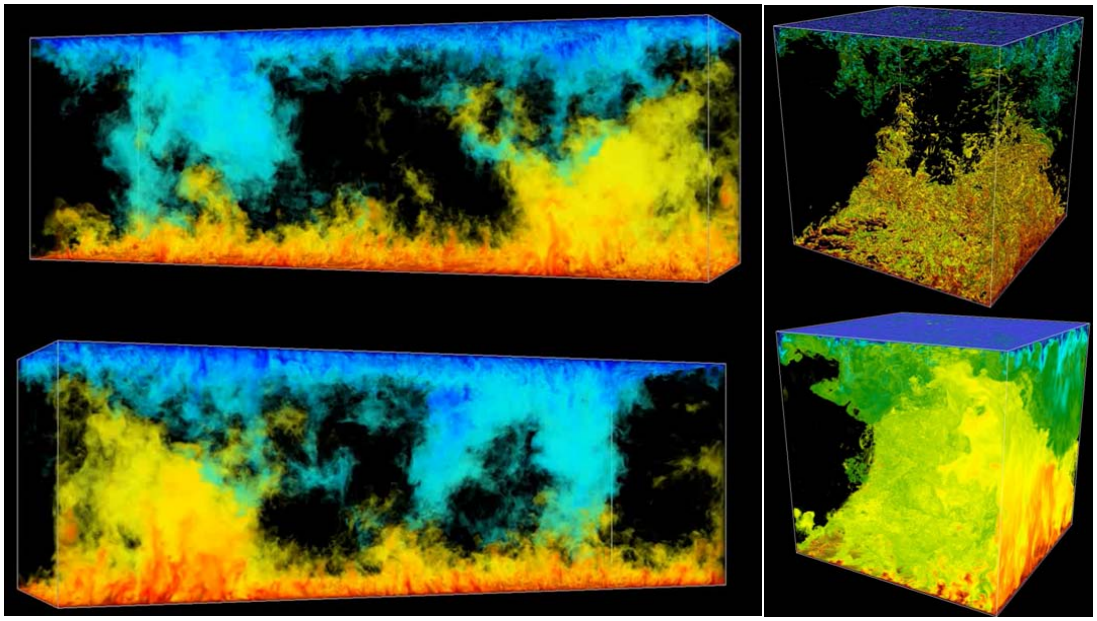


Получается, что самый дорогой подход – это DNS. Вот, например, расчет течения вокруг квадратного цилиндра (Re=22000, сетка 320 млн ячеек). Показано поле давления в центральном сечении. Понятно, что расчет делался не для того, чтобы улучшить аэродинамическую форму квадратного цилиндра. Изучается течение вокруг тела с острыми кромками, на которых возникает отрыв, слои смешения, неустойчивость Кельвина–Гельмгольца, образование вихревого следа.



https://doi.org/10.1016/j.compfluid.2015.09.013

Или вот, например, расчет конвекции Рэлея–Бенара. Расчеты были до числа Рэлея 10^{11}. Самый большой расчет был на сетке из 6 миллиардов ячеек. Он прожег 30 млн ядрочасов на суперкомпьютере, чтобы разрешить течение до мельчайших турбулентных структур и получить данные об эволюции топологии течения.



https://doi.org/10.1103/PhysRevFluids.5.024603 https://doi.org/10.1063/1.5005842

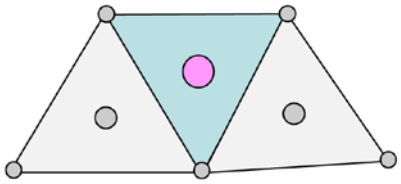
Для реальных задач аэродинамики такое пространственное разрешение непозволительно, моделирование напрямую уравнениями Навье – Стокса неприменимо. Нужны дополнительные модели и уравнения. LES сильно (на порядки) дешевле DNS. Гибридный подход DES многократно дешевле LES. RANS сильно дешевле DES.

Подходы DNS, LES, DES называют вихреразрешающими, поскольку вихри хоть какого-то хотя бы крупного масштаба воспроизводятся напрямую расчетом. Раз течение воспроизводится в динамике, то такие расчеты называют нестационарными, а также – точными по времени. Для получения осредненной картины течения, то есть статистики течения, нужно усреднять моментальные картины течения на достаточно длительном временном интервале. RANS – это стационарный расчет. Он дает сразу осредненную картину течения. Поэтому в нем нельзя получить никаких нестационарных характеристик – ни пульсаций давления на поверхности, ни спектров шума, ни вибрационных нагрузок, и т.д. Можно получать интегральные величины типа подъемной силы, сопротивления, моментов.

Основная масса расчетов промышленных задач – это RANS. Поскольку ресурсоемкость таких расчетов очень низкая. Но точность таких расчетов зачастую невысока. Потому что турбулентные структуры течения сильно зависят от задачи, от геометрии, от конфигурации течения. Особенно сильно зависят крупные структуры. Подобрать параметры модели так, чтобы она подходила ко всем течениям невозможно. Настроишь модель на течение вокруг крыла, она не будет работать на течении вокруг шасси, и т.д. Особенно плохо дело с отрывными течениями. Поэтому разрешая хотя бы крупные структуры методом LES, намного повышается точность результата. А мелкие структуры ведут себя уже более однородно и куда более схожи между собой в разных конфигурациях течения.

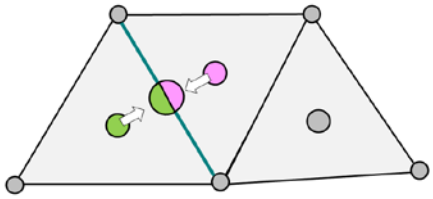
1.6.4 Точности численной схемы

Рассмотрим проблему на простейшем примере. В самом упрощенном виде получается примерно следующий алгоритм. Пусть у нас есть сетка. Для простоты изображения – двумерная треугольная. Рассмотрим маленький фрагмент сетки из нескольких элементов – треугольников. Кружочки в центрах масс треугольников отображают место, к которому относятся заданные на этих треугольниках физические переменные.

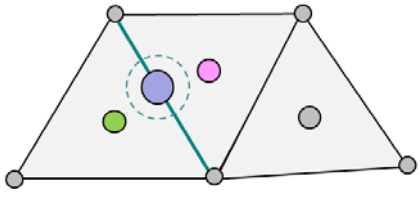


На сетке заданы сеточные функции для плотности, давления (или полной энергии), компонентов вектор скорости (или импульса). Эти сеточные функции описывают состояние моделируемой среды, в данном случае – сжимаемого газа, в какой-то момент времени. Чтобы узнать, как будет дальше развиваться течение газа, надо сделать (небольшой, очень небольшой) шаг по времени и найти сеточные функции на следующем временном слое. То есть надо для каждой ячейки (треугольника) посчитать, сколько втекло и вытекло газа через его границы. В двухмерном случае границы – это ребра треугольника, в трехмерном – грани тетраэдра, например. Вычислив потоки через границы элементов, мы найдем для каждого элемента значения переменных на новом временном слое.

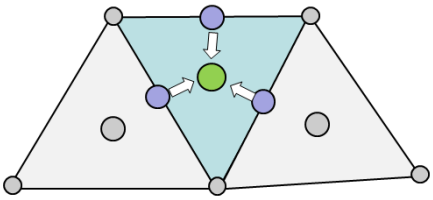
Чтобы посчитать потоки через грани, нам надо выполнить реконструкцию – узнать значения переменных в центрах масс граней. Для этого их надо туда как-то хитро интерполировать. Можно построить полином, можно нагородить разные интерполяционные конструкции, и много еще чего. Изобразим этот этап так:



После реконструкции мы для грани, разделяющей два элемента, назовём их – левый и правый, узнали значения переменных на этой грани со стороны каждого из элементов. На картинке выше это отмечено как розовый и зеленый полукружки в центре ребра. Теперь нам из этих двух наборов переменных надо получить один набор. Для этого решается задача Римана о распаде разрыва. Делает это специальная подпрограмма (*жарг.* – распадник). Мы получили значения потоков через грань (синий кружок на картинке ниже). Так надо сделать для всех граней между элементами.



Теперь осталось просуммировать потоки с граней в элементы, чтобы найти новые значения переменных в элементах. Поток с грани добавляется с плюсом в один элемент, и с обратным знаком в другой.



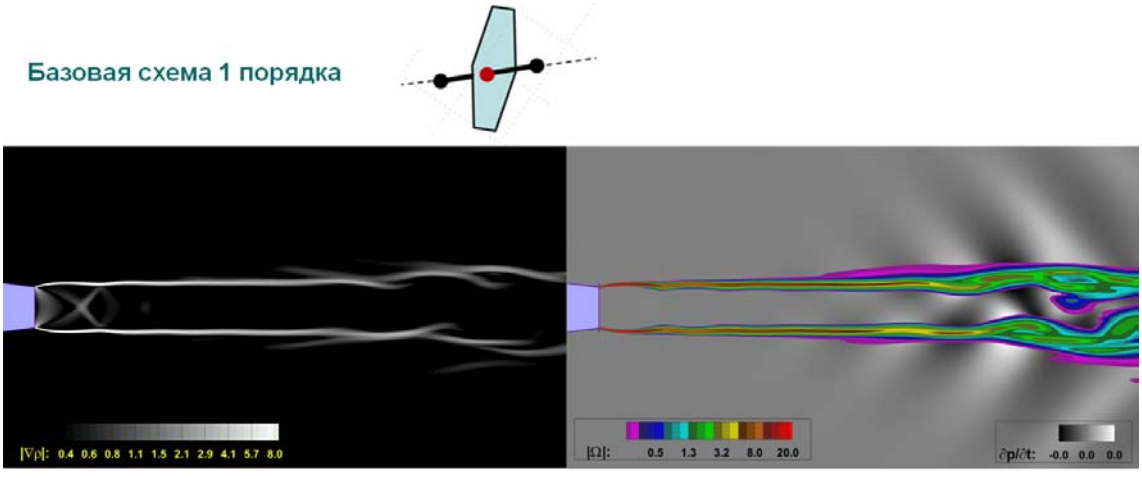
Получили следующий алгоритм:

1) реконструкция значений на гранях по значениям в ячейках (цикл по набору граней);
2) расчет потока через грань – решение задачи Римана о распаде разрыва (цикл по набору граней);
3) переход на новый временной слой: суммирование потоков через грани в ячейки (цикл по набору ячеек).

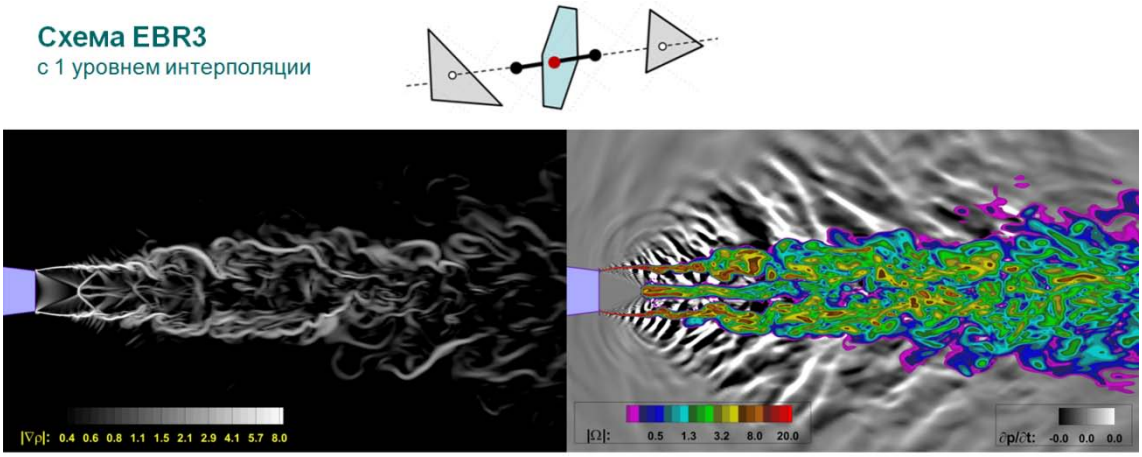
На каждом из этапов получается обработка набора из (очень) большого количества сеточных объектов (граней, ячеек и т.д.). Далее с помощью распараллеливания, то есть, распределяя эту обработку между множественными вычислительными устройствами, мы будем многократно ускорять расчет.

Но интересно, насколько же важна точность реконструкции на этапе 1? Рассмотрим примерчик – моделирование трехмерной реактивной струи, вытекающей из сопла круглого сечения. На трех картинках ниже показаны поля течения в центральном сечении. Поля показаны двумя способами: слева (на черном фоне) визуализация по градиенту давления, справа цветом выделены турбулентные структуры, а на сером фоне показаны акустические возмущения (поле производной давления по времени). Также показаны схемы интерполяционных конструкций, принцип работы которых в данном контексте не сильно важен. Важно, что когда мы наворачиваем мудреную интерполяцию по значениям в соседних ячейках, сильно повышается точность.

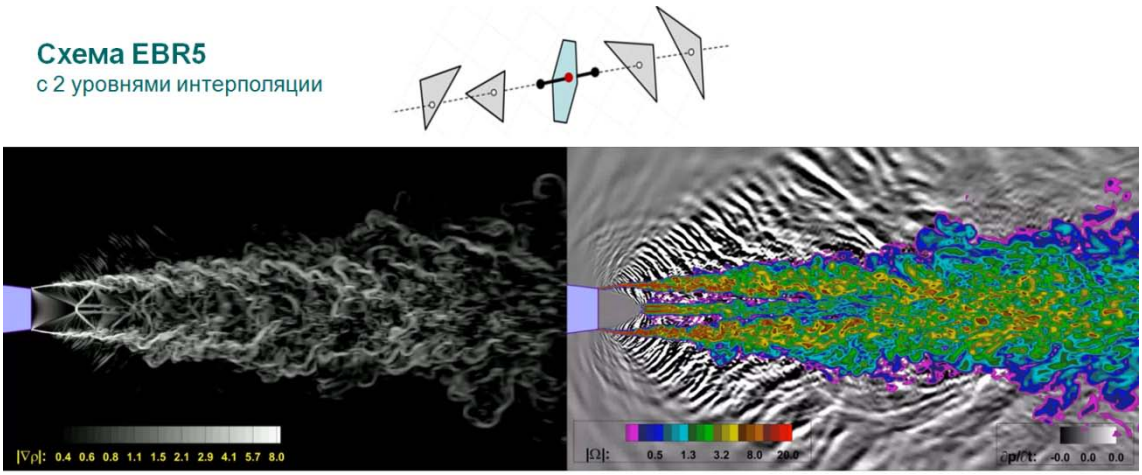
Если реконструкцию вообще не делать, просто брать на грань значения напрямую из ячеек, получится как на первой картинке ниже: ни турбулентности, ни акустики, вязкое ламинарное течение.



Включим схему повышенной точности с интерполяционными конструкциями EBR3. Результат показан на второй картинке ниже. Сразу видно, что появилась турбулентная картина течения, диски Маха, акустические возмущения.



Включим схему еще более высокой точности EBR5 с двухуровневыми конструкциями – появляются все детали течения, включая самые мелкие структуры.



Сетка во всех трех случаях одна и та же. Вычислительная стоимость расчета от применения схемы повышенной точности увеличилась менее чем раза в полтора. Чтобы достичь той же точности базовой схемой, понадобилась бы сетка раз так, наверное, в сотни поподробнее. Соответственно, в сотни раз больше процессорного времени. Выигрыш налицо. А что, так можно повышать точность до бесконечности, и получать такие картинки даже на очень грубой сетке? Увы, нет. Все равно упремся в необходимость разрешать масштаб мелких вихрей, хотя бы по несколько ячеек на вихрь надо обеспечить, как ни крути.

Параллельная обработка больших графов

Александр Сергеевич Семенов

www.dislab.org





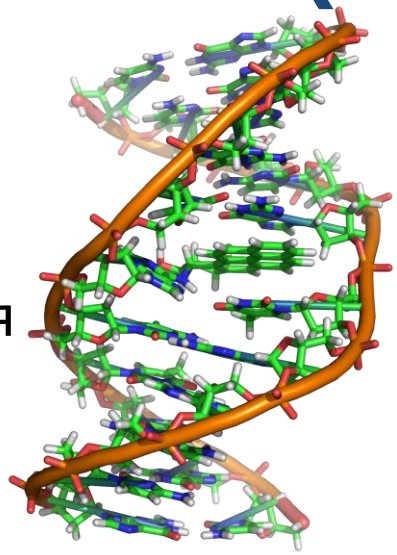
Откуда возникают большие графы?

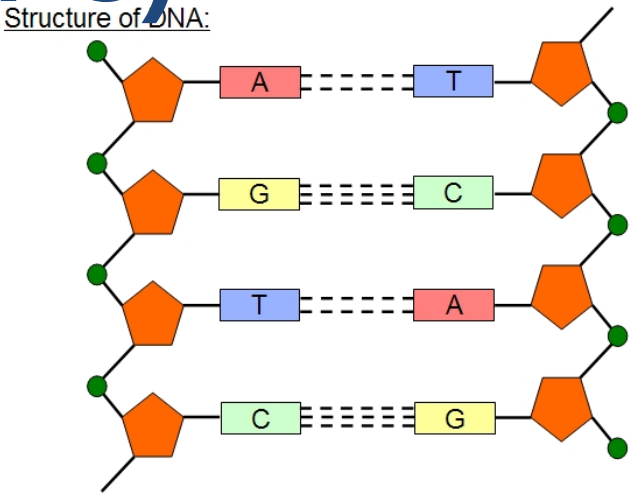
- **Интернет (WWW)**
  - На сентябрь 2016 – 47 миллиардов страниц
  - По оценке Google – более 1 триллиона
- **Социальные медиа**
  - Блогосфера: 2011 – 172 x 10^6 (+10^6/день)
  - Facebook: 2010 – 500 x 10^6, 2013 – 1:1 x 10^9 (650 x 10^6 акт.польз./день), 140 x 10^9 связей
  - LinkedIn: 2013 – 8 x 10^6, 60 x 10^6 связей
  - Twitter: 2011 – 140 x 10^6 сообщений/день
- **Транспортные сети**
- **Биоинформатика**
- **Бизнес-задачи**

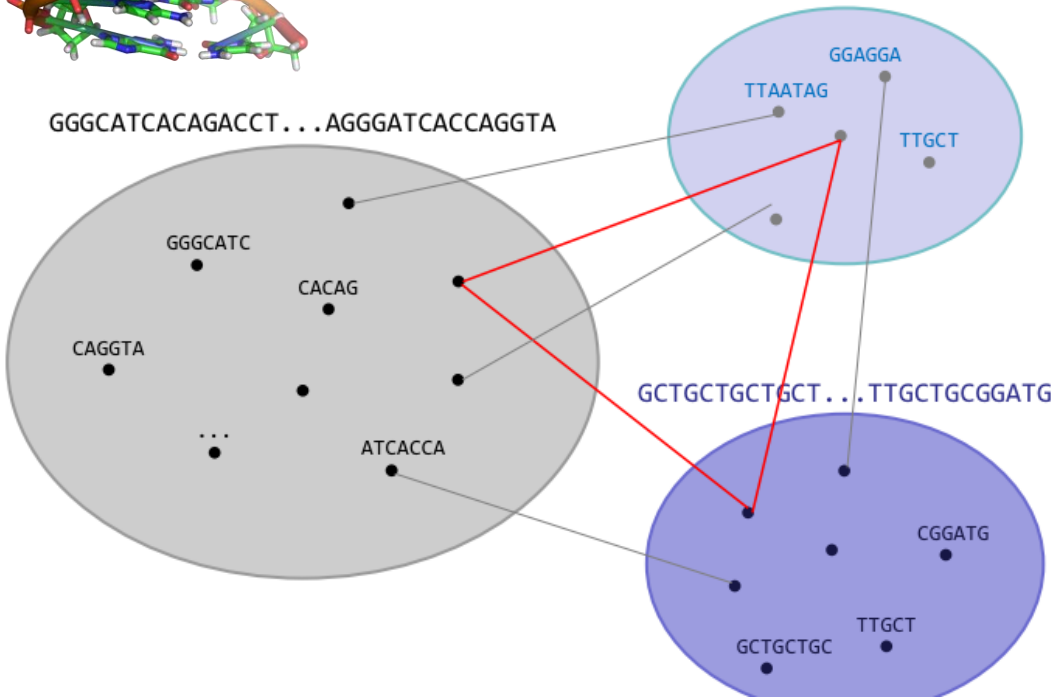
[1]http://www.worldwidewebsize.com

Биоинформатика: сходство организмов (HPC)

- Число долей 10^5
- Длина последовательности 10^9
- Вершин в доле 10^9 (берутся короткие слова)
- Всего вершин 10^{14}
- Найти слова, которые с заданной точностью встречаются во всех последовательностях, или
- Найти клику или плотный подграф (кластеризация), если ребро – характеристика сходства



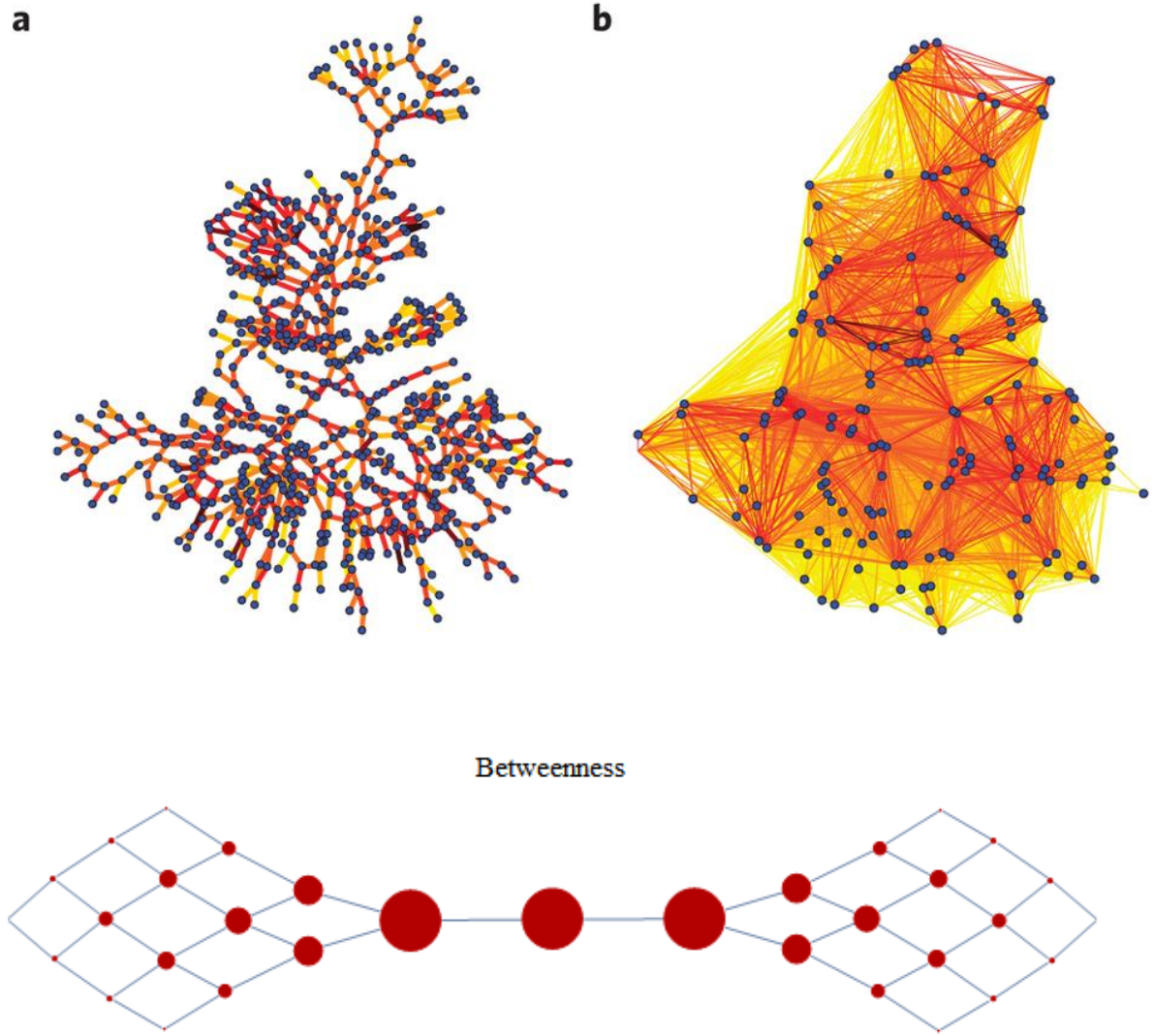




Электросети (HPC)

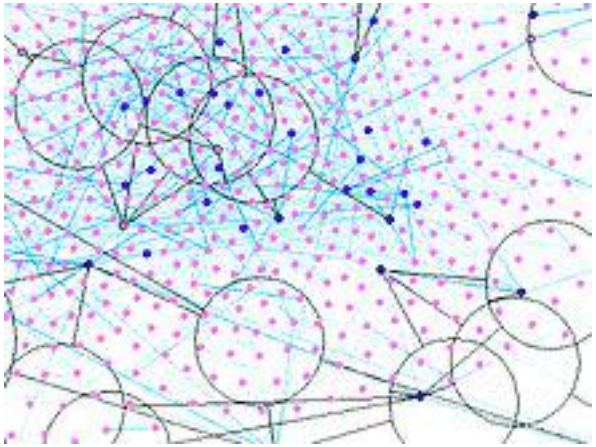
- Связанность
- Надежность

- Различные пути, betweenness centrality



Анализ социальных сетей (HPC)

- Анализ сообществ
- Понимание намерений
- Динамика популяции
- Распространение эпидемий
- Кластеризация

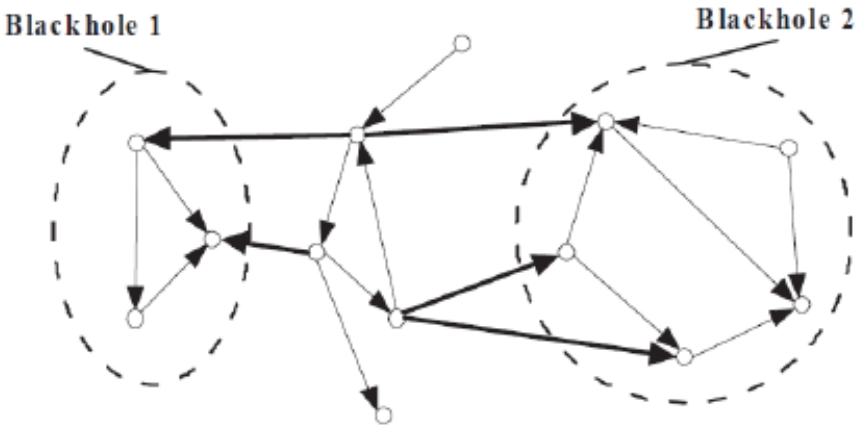




Бизнес-аналитика и кибербезопасность (Big Data&HPC)

- Задачи понимания данных из огромных массивов
- Выявление аномалий в данных
- Анализ данных
- Выявление мошенничества

- Паттерн «черные дыры»
- Machine Learning!



Признаки в графах для машинного обучения

- Вершины (степень, полустепени, betweenness centrality, PageRank)
- Пары вершин (количество общих соседей, вес ребра)
- Egonet (количество треугольников, количество ребер)
- Группа вершин (плотность = кол-во ребер/кол-во вершин, общий вес ребер)

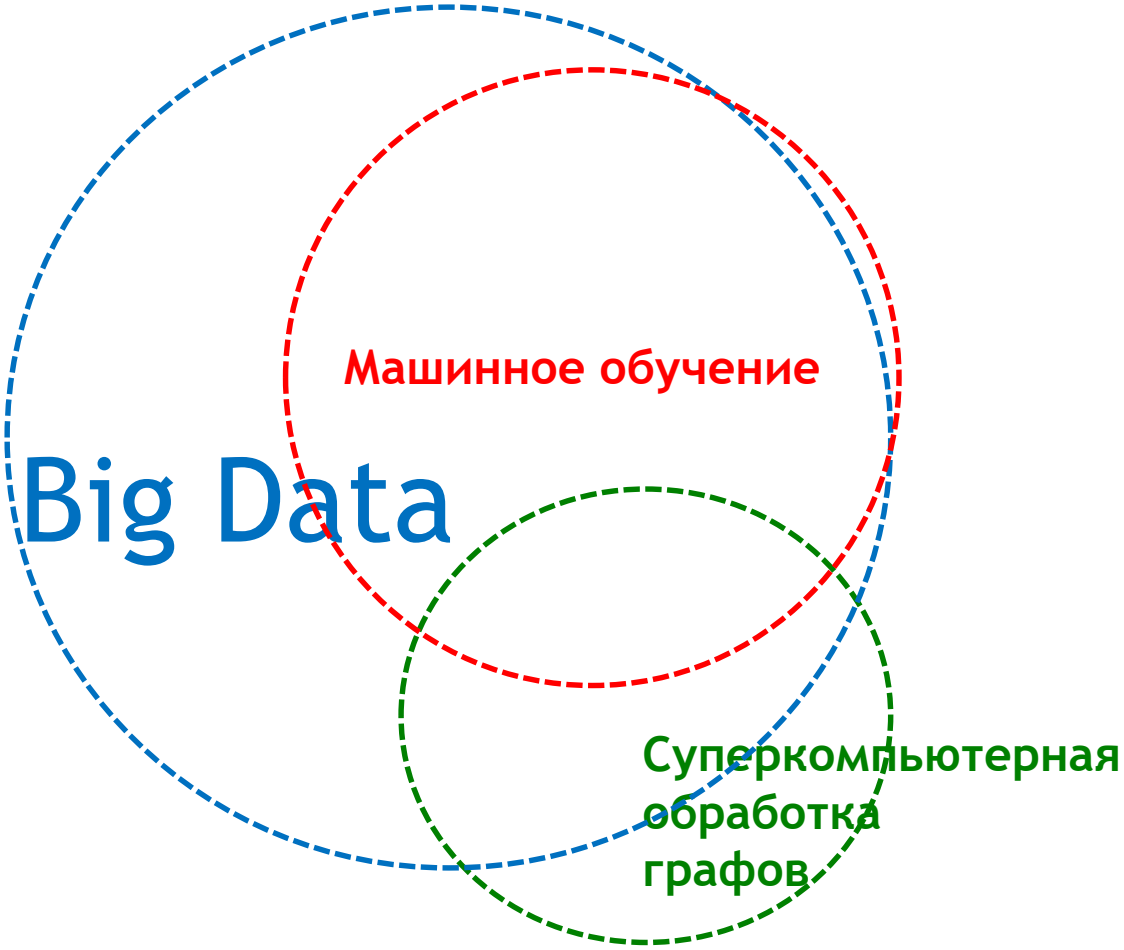
Классификация задач анализа графов

- По типу графов
  - статические графы (static graph analysis)
  - динамические графы (dynamic graph analysis)
  - обработка потоков вершин и ребер (streaming graph analysis)
- По типу обработки
  - в режиме реального времени (online)
  - в режиме выполнения заданий (offline, batch processing)

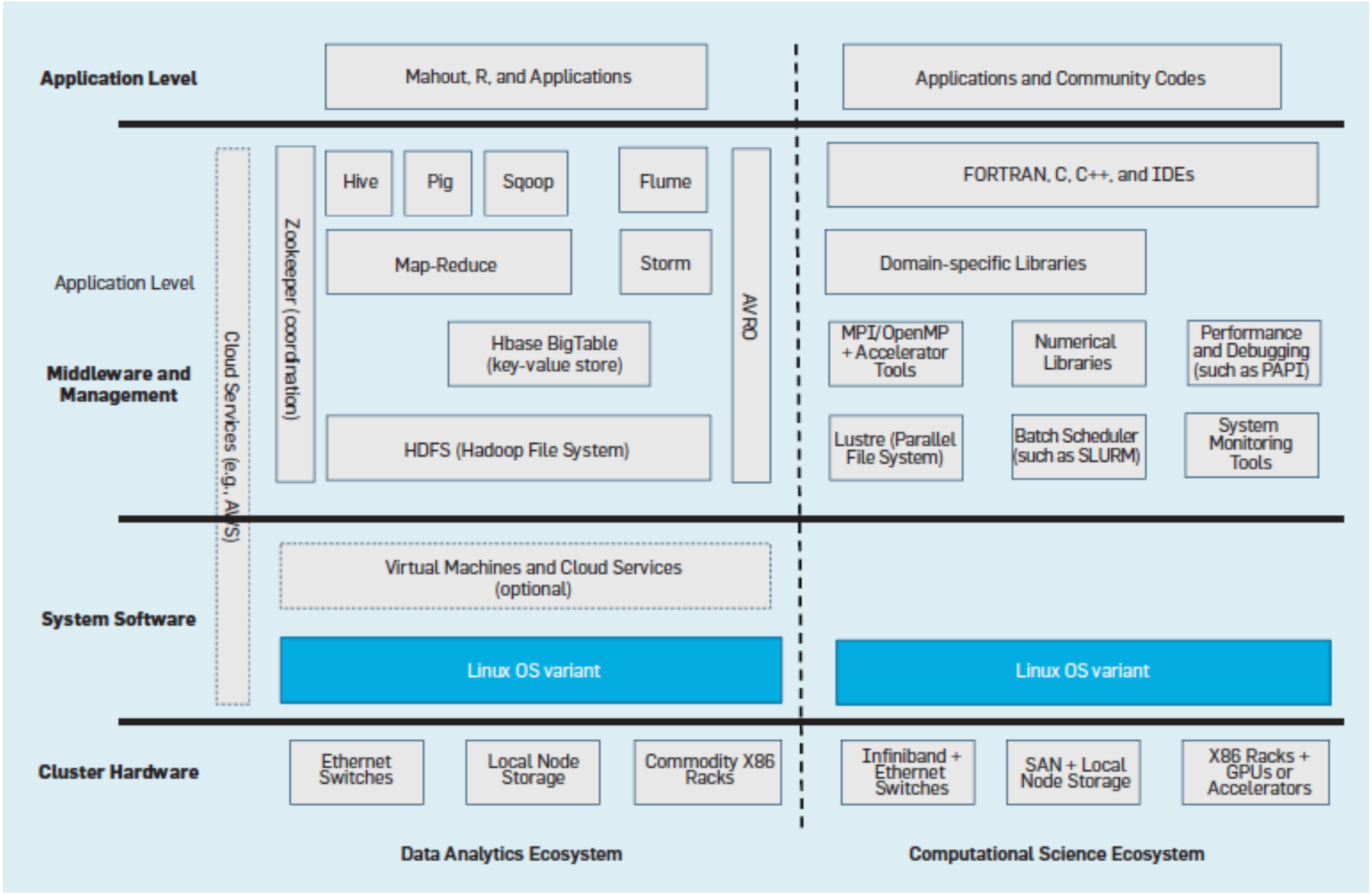
Программные модели и средства

- Реляционная модель
  - Oracle Spatial & Graph, SAP HANA, …
- Графовые СУБД
    - Neo4j, GraphBase, TigerGraph, NitrosBase RDF Storage, …
- Generic MapReduce:
  - Hadoop, Yarn, Dryad, Stratosphere, Haloop
- Graph-optimized: Pegasus, Surfer, GBASE, GraphX
- Проблемно-ориентированные языки программирования (DSL): Green-Marl, Exedra
- BSP
  - Parallel BGL
- Vertex-centric/BSP
  - Pregel (Giraph, Hama, Mizan, …)
- Vertex-centric/Data, Message-driven
  - GraphLab, SWARM, Trinity, Charm++, …
- Fine-grained Threaded Shared Memory/PGAS
  - GraphCT, STINGER, Grappa
- Технологии параллельного программирования
  - OpenMP, MPI, CUDA, …

Big Data vs HPC



Big Data vs HPC



План

- Виды графов
- Основные проблемы, возникающие при решении задач обработки графов
- Подходы к решению задач в рамках одного вычислительного узла
- Подходы к решению задач в рамках распределенной вычислительной системы

Виды графов

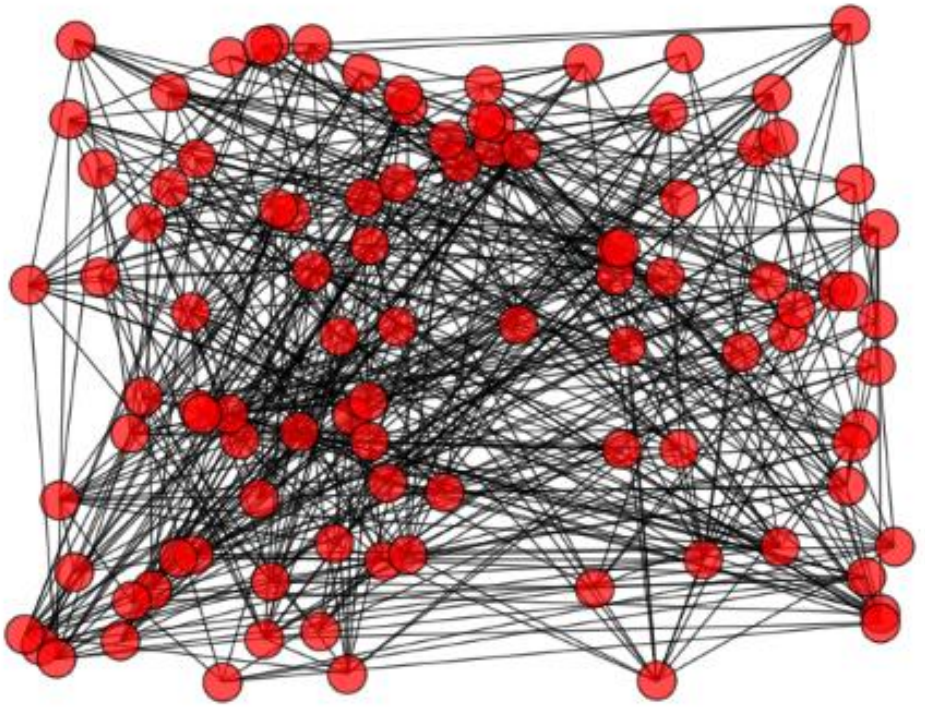
Виды графов. Случайные графы

- Random, Random Uniform, Erdos Renyi (1959)
- Обозначение: G(N, M), G(N, p)

N – количество вершин,

M – количество ребер

p – вероятность ребра между любыми вершинами



Свойства:

- Пусть $p = (c \ln N)/N$

- если $c > 1$, то почти всегда случайный граф связен.
- если $c < 1$, то почти всегда случайный граф не связный.

- Пусть $p = c / N$

- при любом $c < 1$ существует такая константа $\beta = \beta(c)$, что почти наверное каждая компонента связности случайного графа имеет не более $\beta \ln n$ вершин.
- при любом $c > 1$ существует такая константа $\gamma = \gamma(c) \in (0, 1)$, что почти наверное среди компонент случайного графа есть одна (гигантская), число вершин которой не меньше γn.

Виды графов. Степенной закон

- www, Социальные сети, Биоинформатика
- Графы small-world

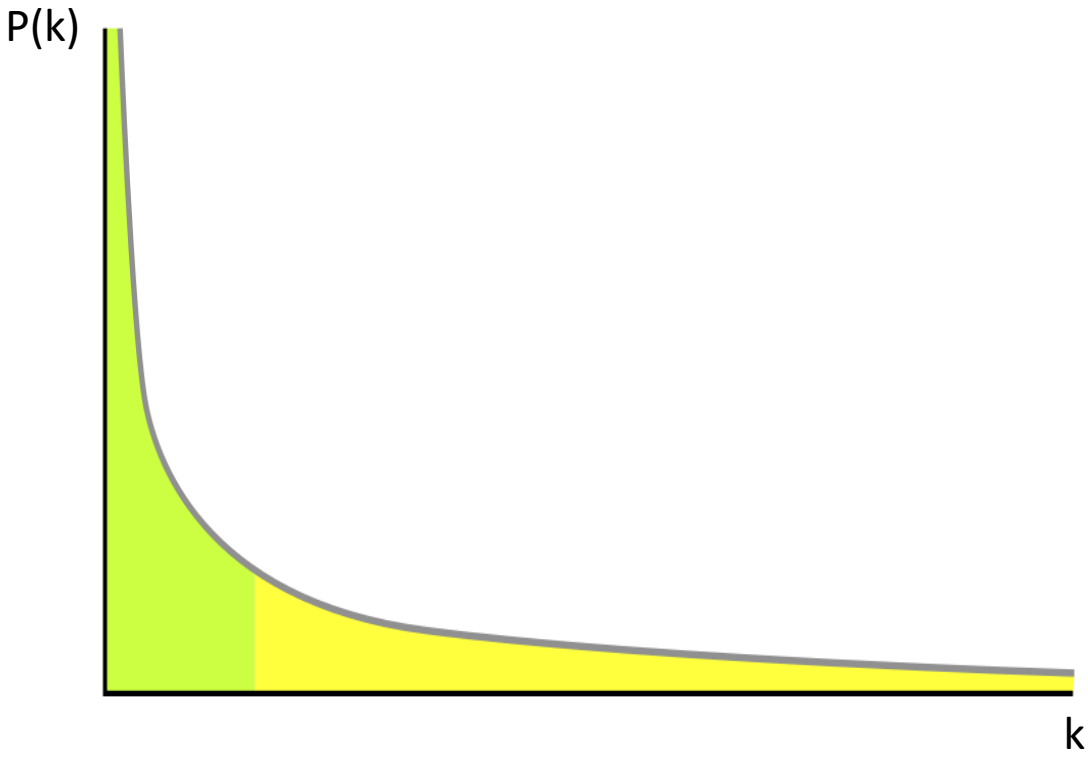
L ~ log N

- scale-free – графы,

доля $P(k) \sim k^{-tau}$, $2 < tau < 3$

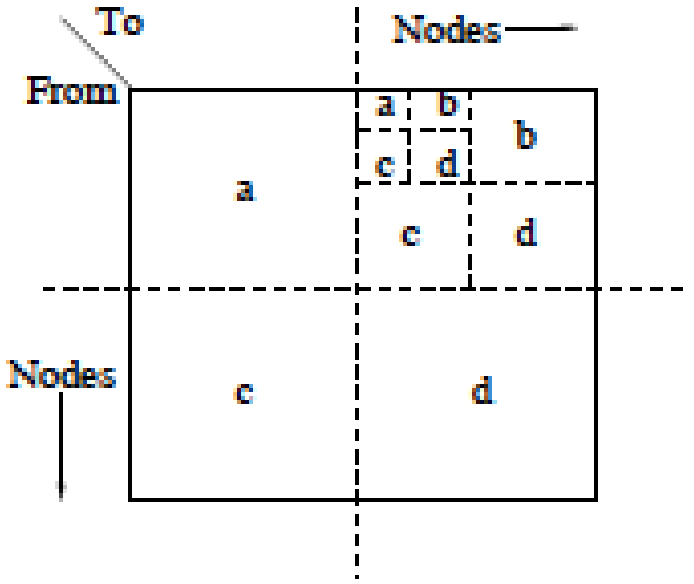
k – степень вершины

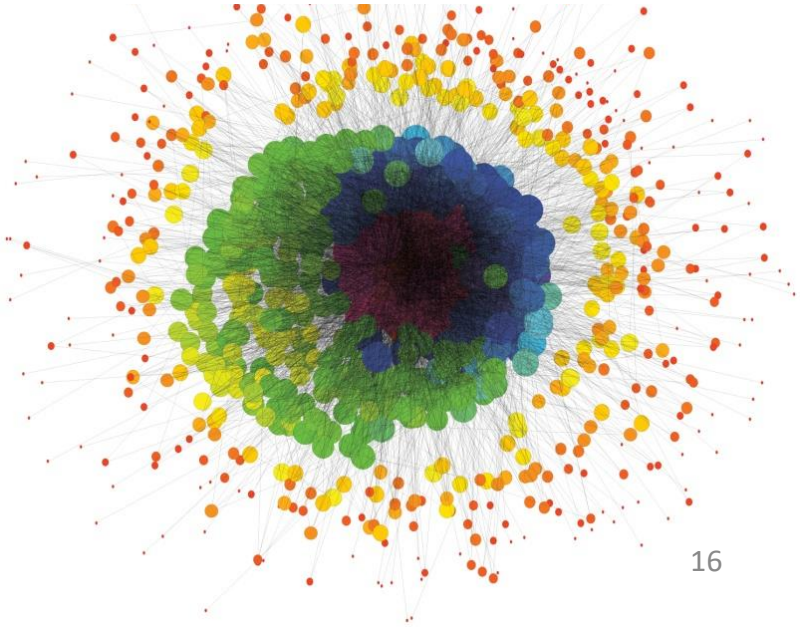
L ~ log log N



Виды графов. RMAT-граф

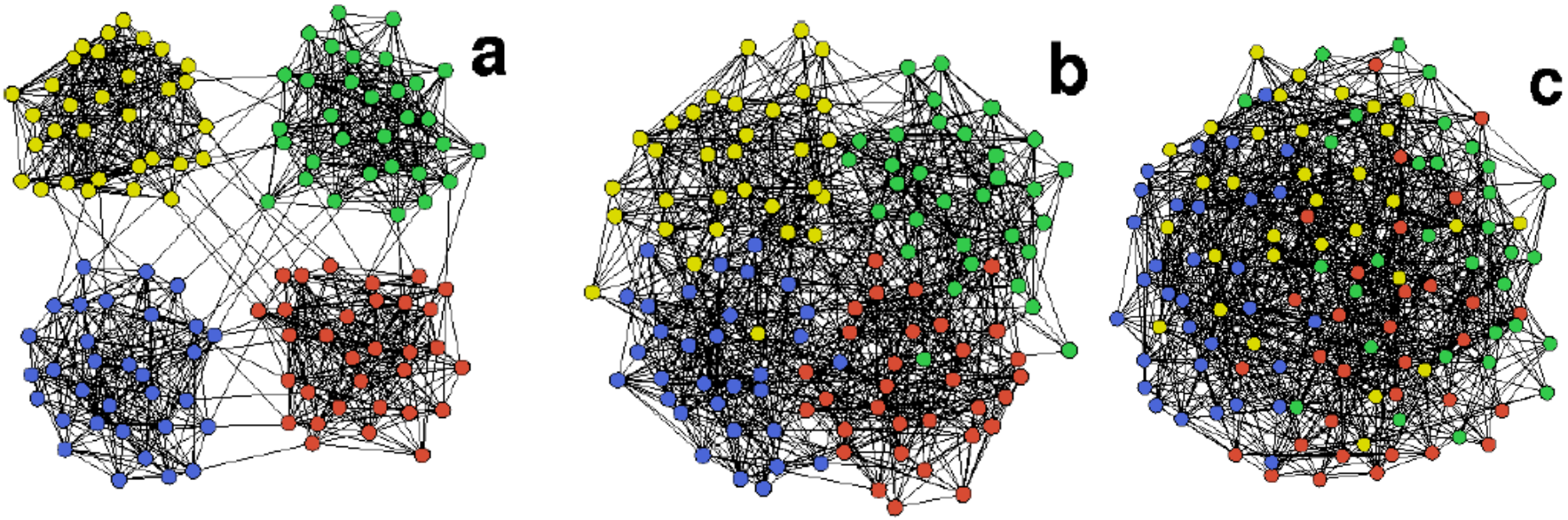
- a+b+c+d = 1
- Сообщества:
  - a и d – сообщества
  - b и c – связи между ними
  - наличие «подсообществ»
- может быть scale-free при a>=d
- случайная перестановка вершин





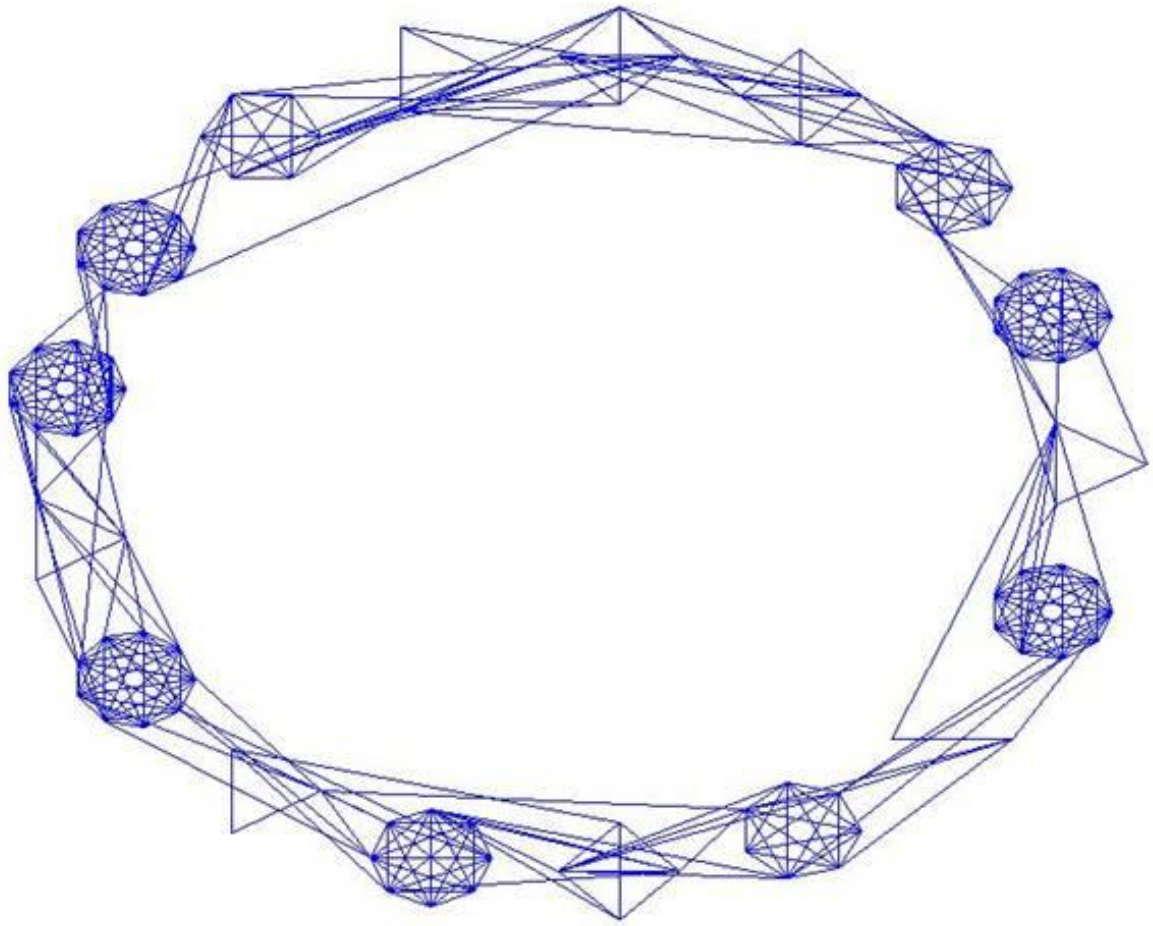
Виды графов. LFR*-граф

- **Параметры**:
  - mu ∈ [0;1], показывает количество связей вне сообщества
  - com_tau – показатель степени в законе распределения размеров сообществ
  - deg_tau – показатель степени в законе распределения степеней вершин



Виды графов. SSCA2-граф

- равномерное распределение случайных параметров
- случайная перестановка вершин



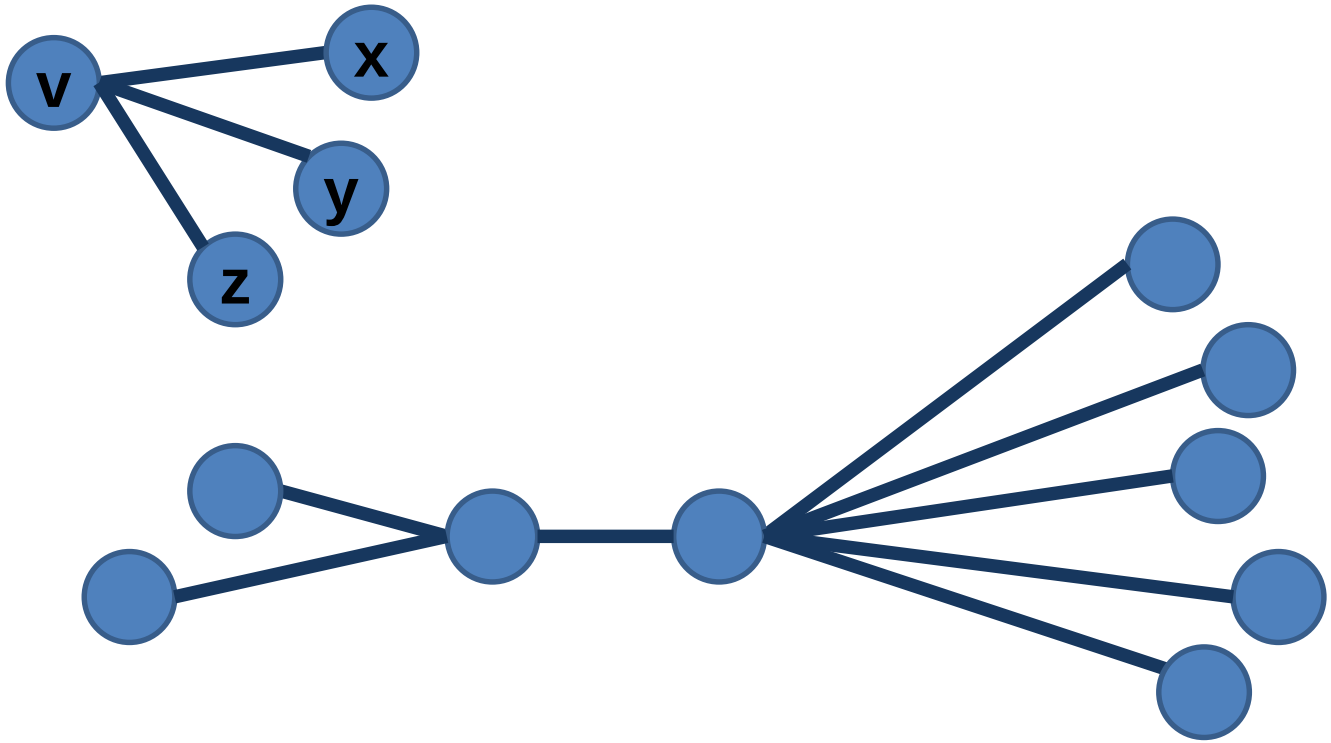
Основные проблемы, возникающие при решении задач обработки графов

Проблемы анализа больших графов

- **Data-driven computations.** Зависимость вычислений от данных (топологии графа). Невозможность применения методов статического распараллеливания вычислений.
- **Unstructured problems.** Работа с нерегулярными, неструктурированными данными, трудность распараллеливания.
- **Poor locality.** Низкая пространственно-временная локализация обращений к памяти.
- **High data access to computation ratio.** Преобладание команд доступа к памяти над командами выполнения арифметических операций.

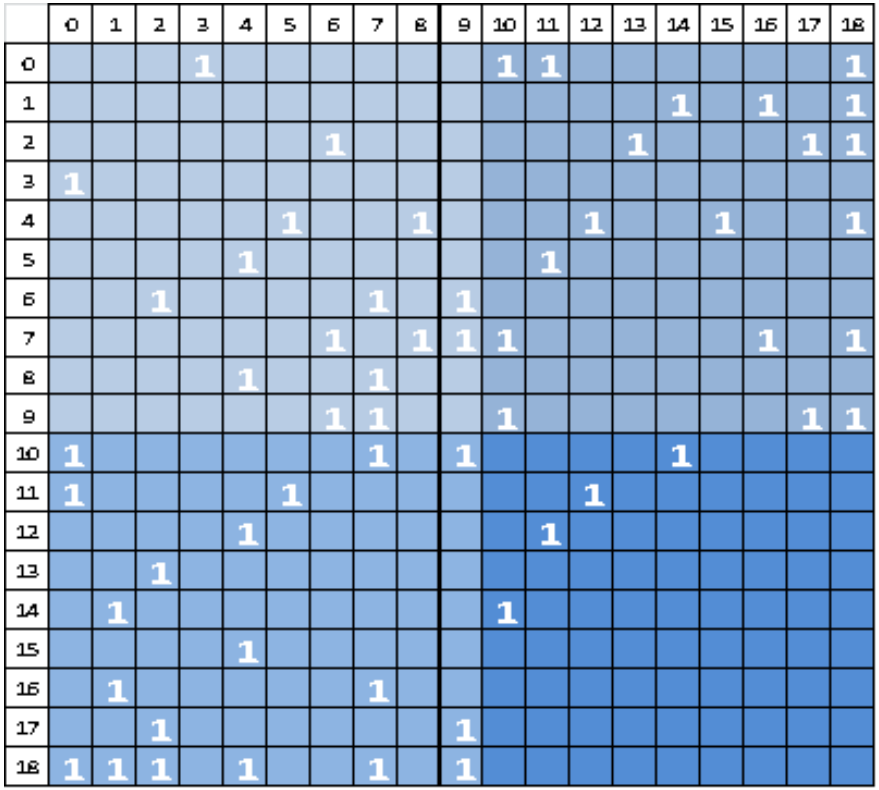
Проблемы анализа больших графов (1)

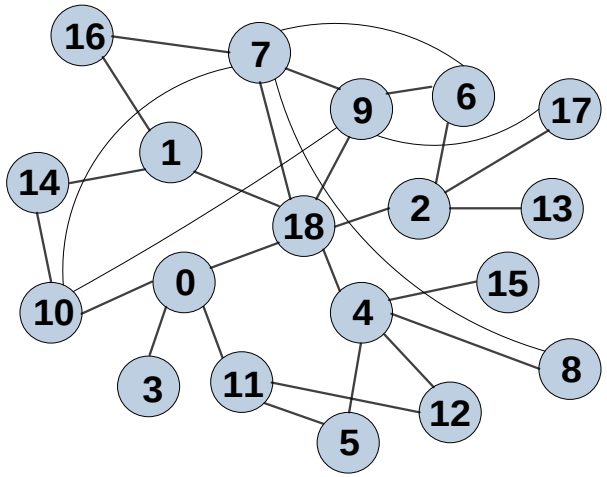
- **Data-driven computations.** Зависимость вычислений от данных (топологии графа). Невозможность применения методов статического распараллеливания вычислений.



Проблемы анализа больших графов (2)

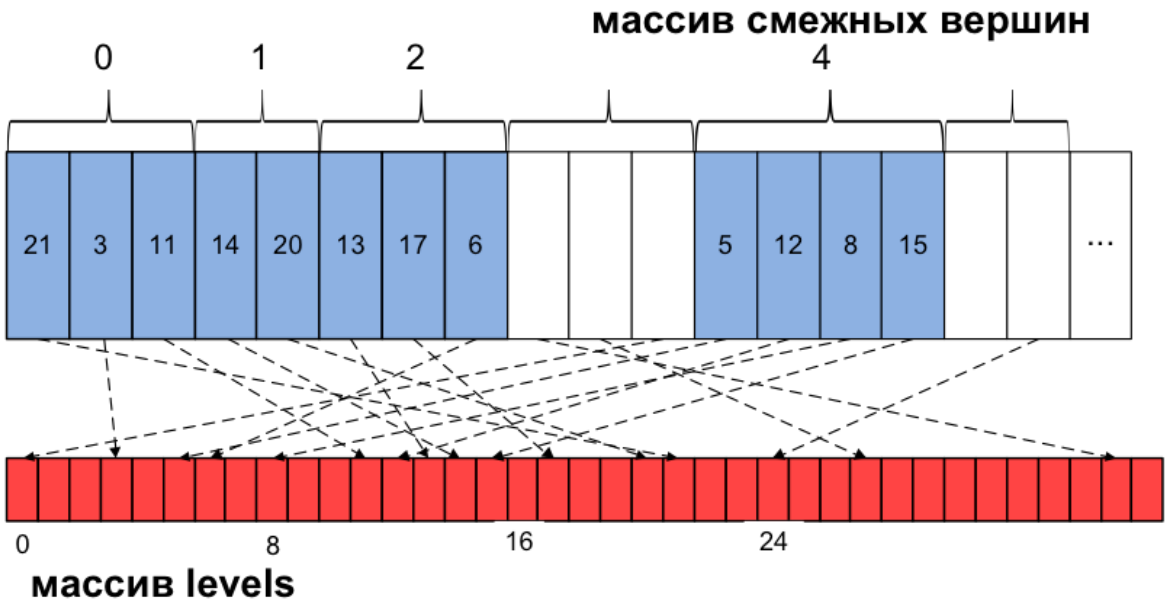
- **Unstructured problems.** Работа с нерегулярными, неструктурированными данными, трудность распараллеливания.

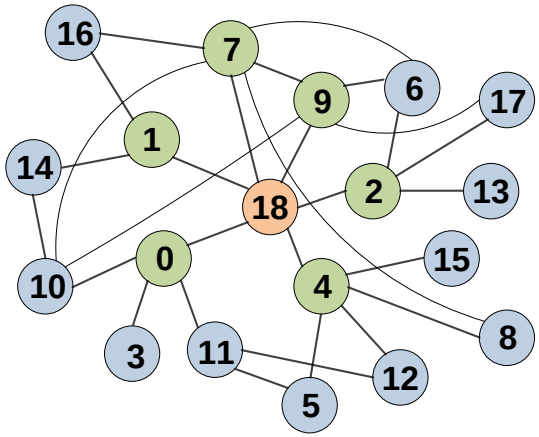




Проблемы анализа больших графов (3)

- **Poor locality.** Низкая пространственно-временная локализация обращений к памяти.





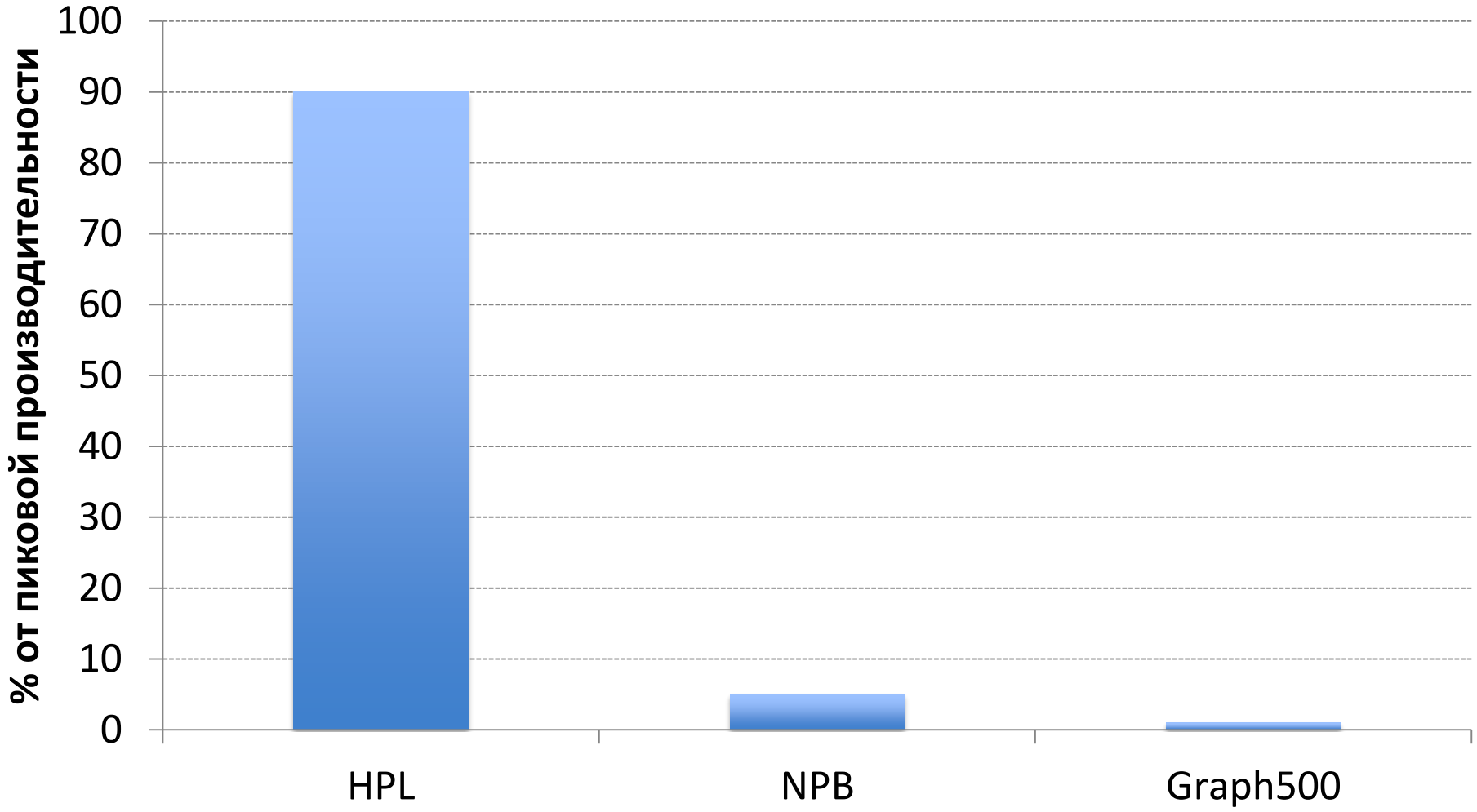
Проблемы анализа больших графов (4)

- **High data access to computation ratio.** Преобладание команд доступа к памяти над командами выполнения арифметических операций.

Intel E5-2680 v3, 2.5 ГГц

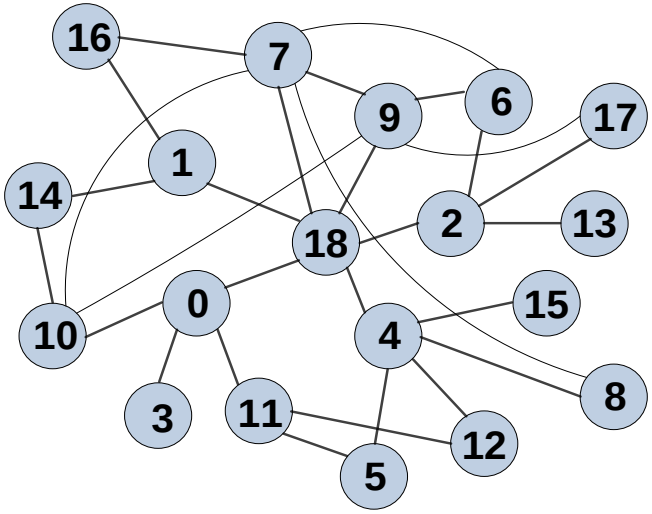
Параметр	Задержка, нс (такты)	ПС, ГБ/с
Регистр	(1)	–
Кэш L1	1.6 (4)	240
Кэш L2	4.4 (11)	160
Кэш L3	16 (40)	80
Память своего сокета	60	~55
Память чужого сокета	100	~30
Сеть Ангара	MPI – 1000 нс, SHMEM – 600 нс	

Проблема низкой реальной производительности



Проблемы и подходы к решению задач обработки графов в рамках одного вычислительного узла

Представление графа



- Размер: **O(N²)**
- Проверка связности: **O(1)**

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0				1							1	1							1
1															1		1		1
2							1							1				1	1
3	1																		
4						1			1				1			1			1
5					1							1							
6			1					1		1									
7							1		1	1	1						1		1
8					1			1											
9							1	1			1							1	1
10	1							1		1					1				
11	1					1							1						
12					1							1							
13			1																
14		1									1								
15					1														
16		1						1											
17			1							1									
18	1	1	1		1			1		1									

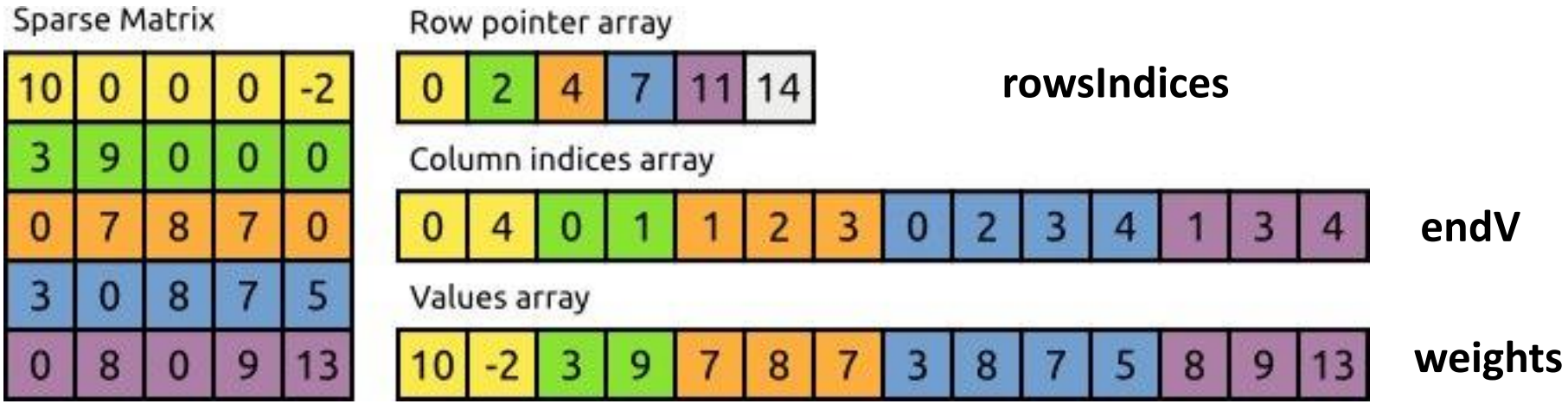
Форматы представления разреженных матриц

- **Доля ненулевых элементов мала**

Можно хранить только позиции и значения ненулевых элементов

- **Compressed Row Storage (CRS)**
- **Coordinate list (COO)**
- **DIA**
- **ELLPACK**
- **SELLPACK**
- **Оптимизированный под задачу**

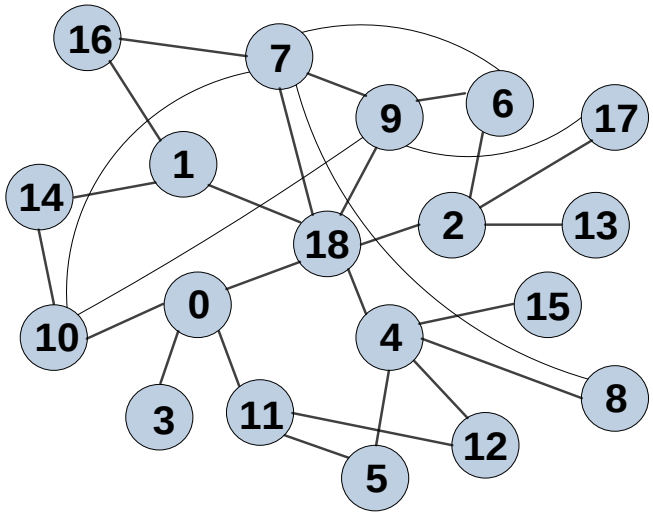
Внутреннее представление Compressed Row Storage (CRS)




```
for (int u = 0; u < G->n; u++) {
    for (int j = G->rowsIndices[u]; j < rowsIndices[u+1]; j++) {
      const int v = G->endV[j];
      const int w = G->weights[j];
      // обработка ребра u->v
    }
}
```

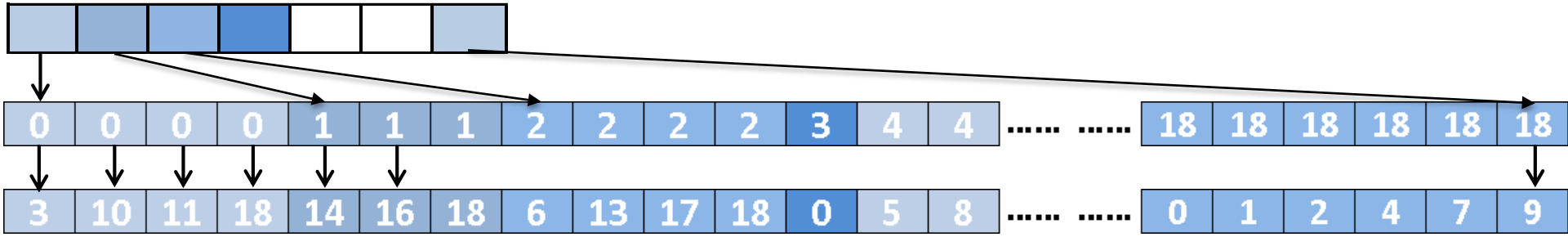
- Размер: **O(N+M)**
- Проверка связности: **O(log d_{max})**

Coordinate list (COO)



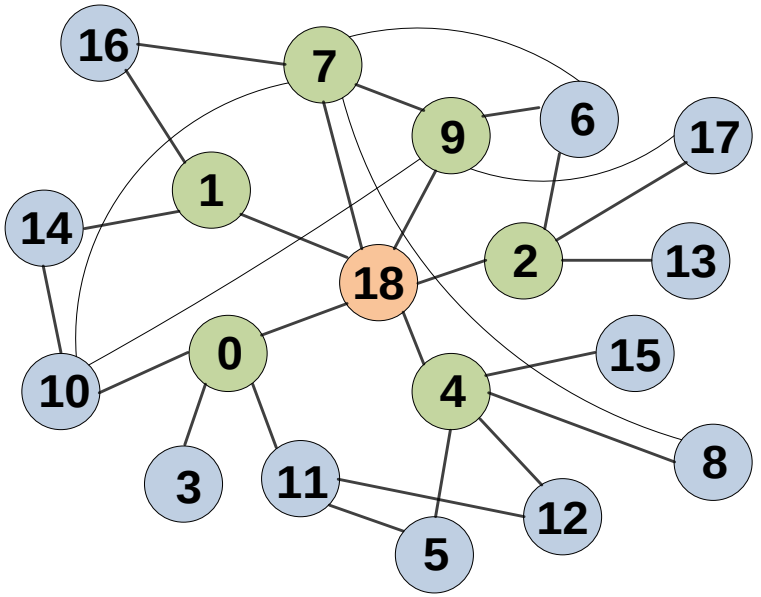
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0				1							1	1							1
1															1		1		1
2							1							1				1	1
3	1																		
4						1			1				1			1			1
5					1							1							
6			1					1		1									
7							1		1	1	1						1		1
8					1			1											
9							1	1			1							1	1
10	1							1		1					1				
11	1					1							1						
12					1							1							
13			1																
14		1									1								
15					1														
16		1						1											
17			1							1									
18	1	1	1		1			1		1									

- Размер: **O(N+M)**
- Проверка связности: **O(log d_{max})**



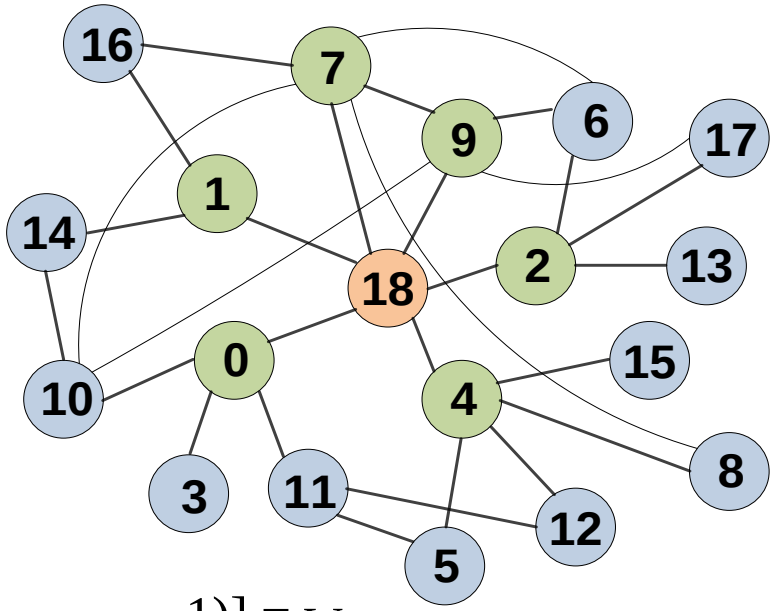
Поиск вширь в графе

```
Q = {v_start}
Visited = {v_start}
while Q ≠ {}
    Q_next = {}
    for all vertex ∈ Q do
        for all w: (vertex, w) ∈ E do
            if w ∉ Visited then
                Q_next = Q_next ∪ w
                Visited = Visited ∪ w
            endif
        end for
    end for
    Q = Q_next
end while
```

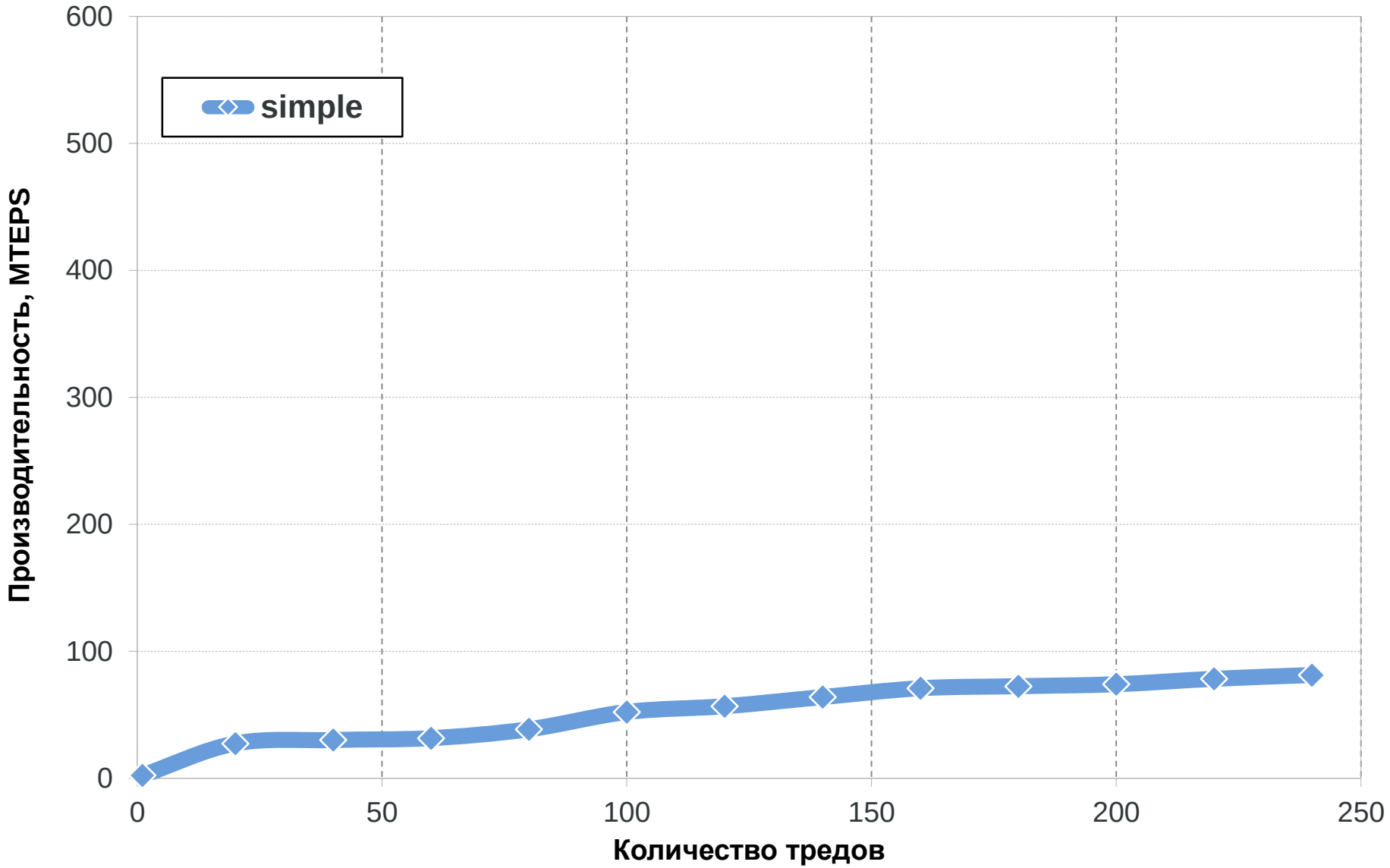


Поиск вширь в графе (BFS)
Подход Queue-based, алгоритм simple

$Q_{counter} = 1$
Q[0] = root
Visited[root] = 1
while $Q_{counter} > 0$
    $Q_{next_counter} = 0$
    #pragma omp parallel for
    for all vertex ∈ Q **do**
        for all w: (vertex, w) ∈ E **do**
            if Visited[w] == 0 **then**
                Q_{next}[__sync_fetch_and_add($Q_{next_counter}$, 1)] = w
                Visited[w] = 1
            endif
        end for
    end for
    swap(Q, Q_{next}) // *обмен Q и* Q_{next}
end while

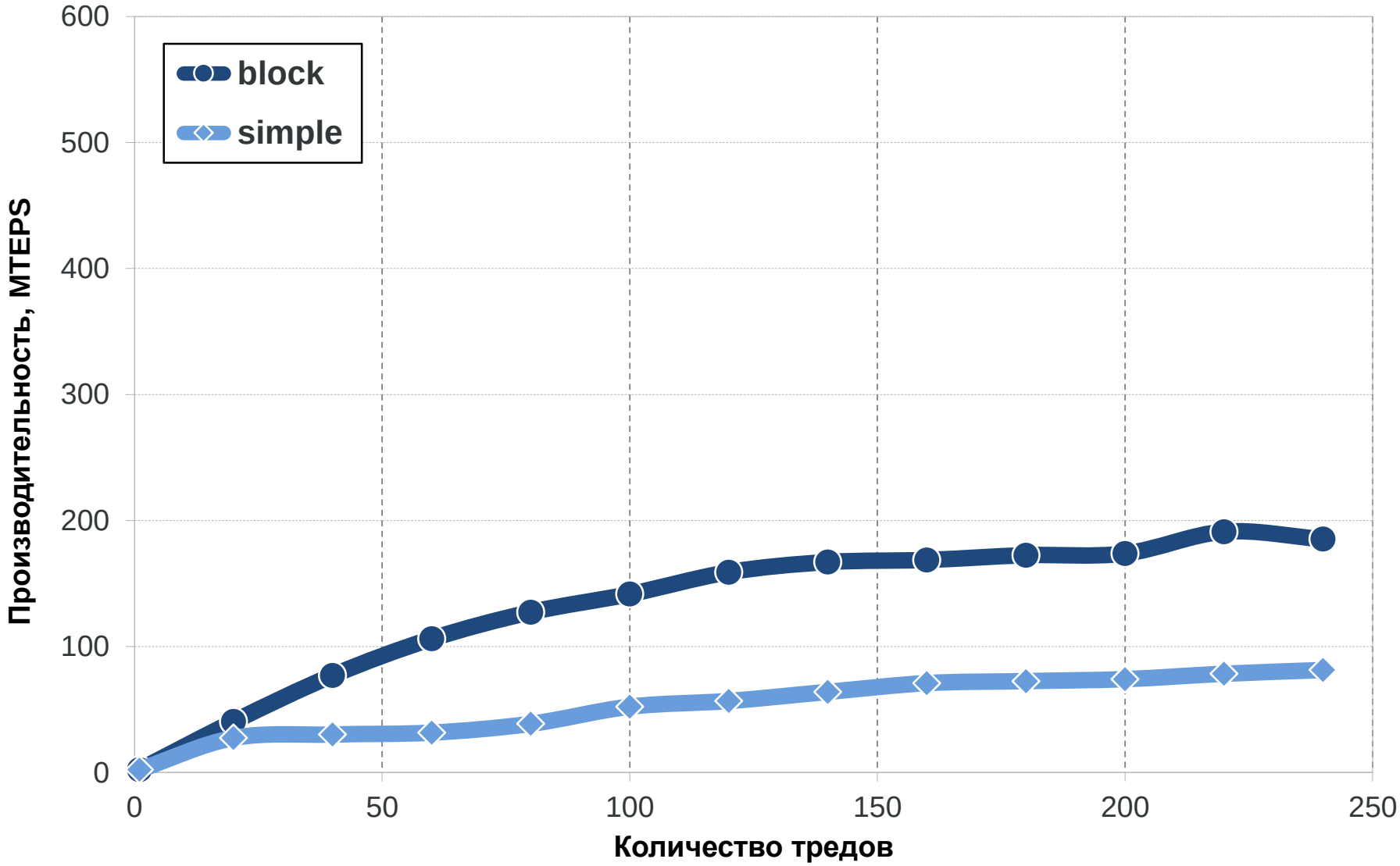


Производительность алгоритма simple в зависимости от числа используемых тредов на сопроцессоре Phi-5110P



Число вершин в графе: N = 2^{27} (134 млн), средняя связность вершины: k = 8

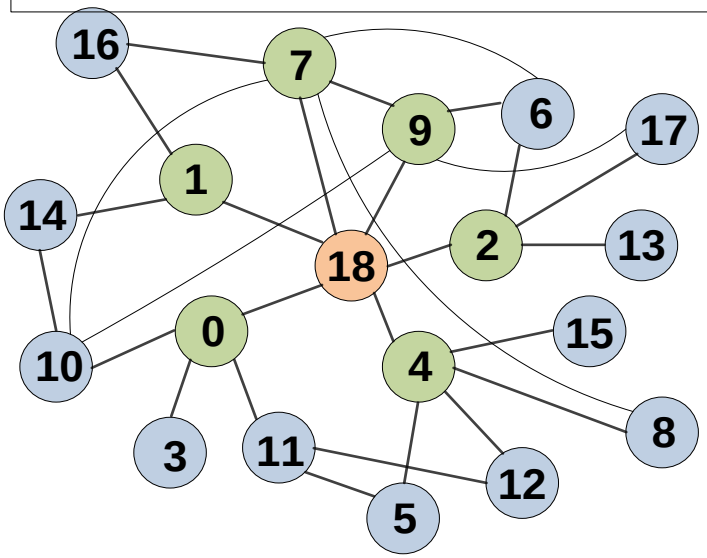
Производительность алгоритмов simple и block в зависимости от числа используемых тредов на сопроцессоре Phi-5110P

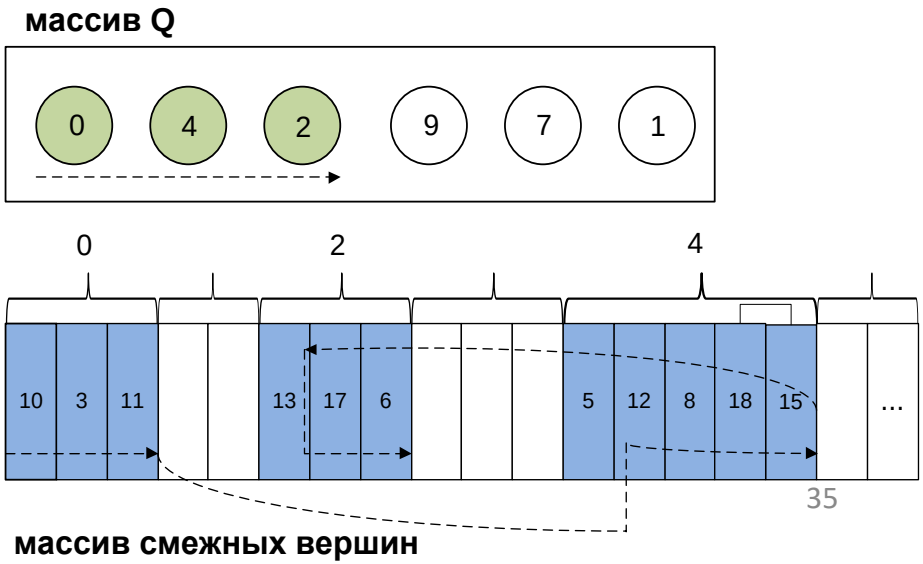


Число вершин в графе: N = 2^{27} (134 млн), средняя связность вершины: k = 8

Недостатки подхода Queue-based

```
#pragma omp parallel for
for all vertex ∈ Q  do
    for all w: (vertex, w) ∈ E  do
        if Visited[w] ==  0 then
            Q_next[__sync_fetch_and_add(Q_next_counter, 1)] = w
            Visited[w] = 1
        endif
    end for
end for
```

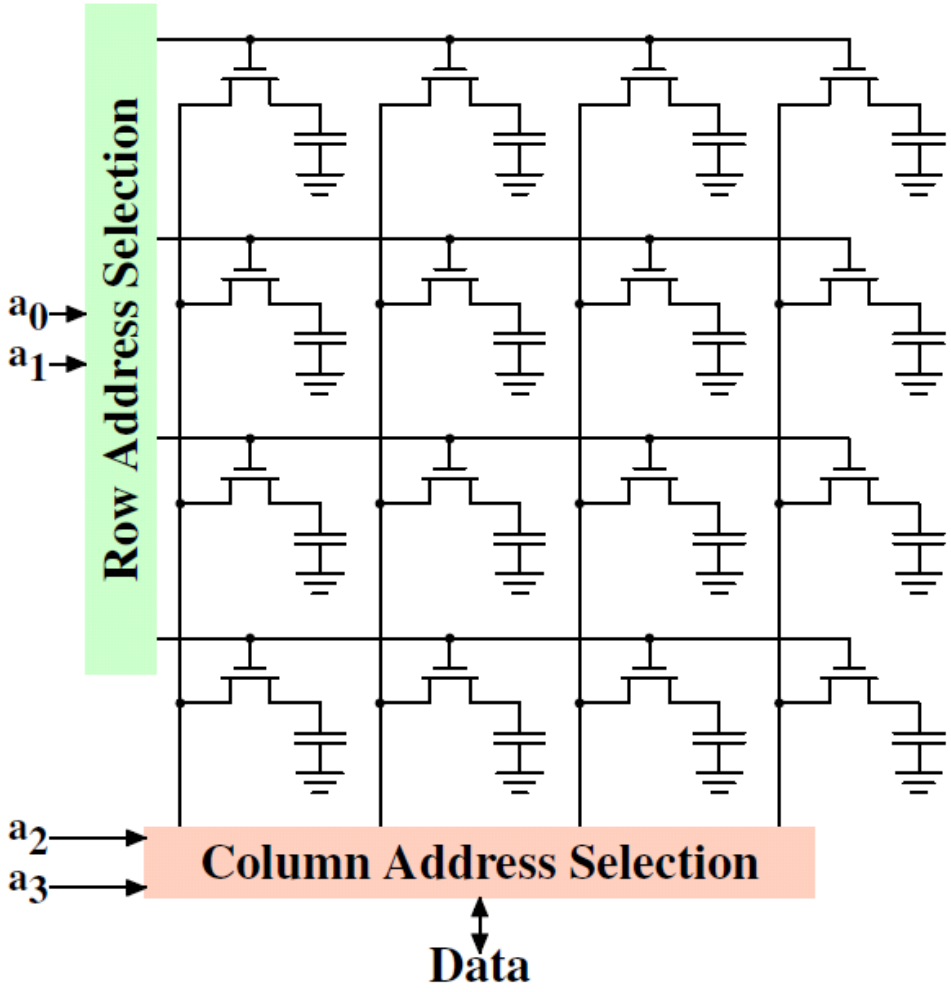




Память SDRAM

- Чтение памяти, необходимо подзаряжать конденсаторы
- Необходимость перезарядки конденсаторов (токи утечки)
- На все операции требуется время

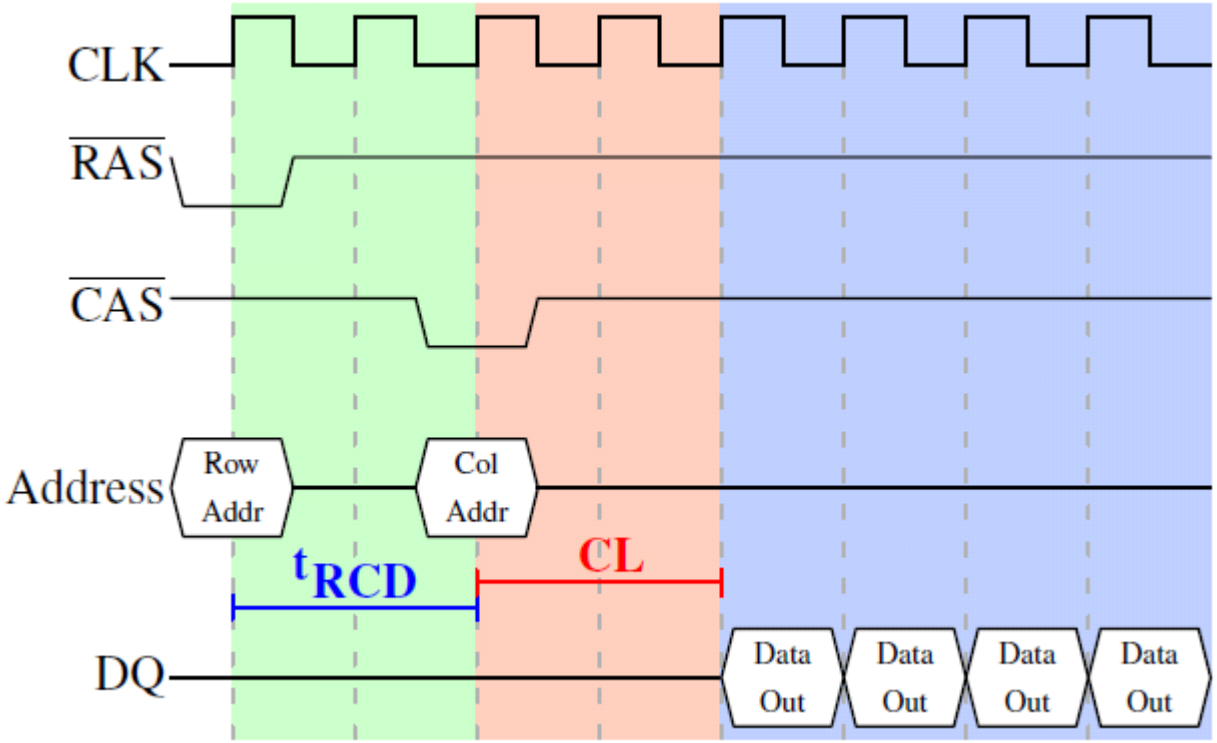
- Память организована как матрица



Drepper, U. (2007). What every programmer should know about memory. Red Hat, Inc, 11, 2007. http://rus-linux.net/lib.php?name=/MyLDP/hard/memory/memory.html

Память SDRAM

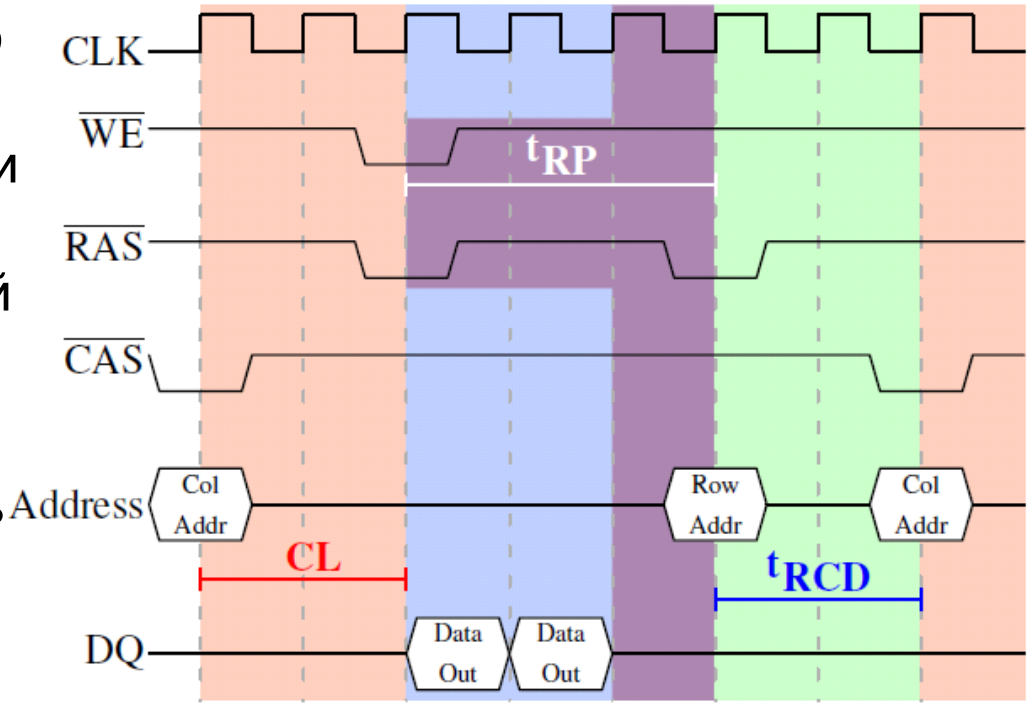
- На определение состояния и перезарядку требуется время



Память SDRAM

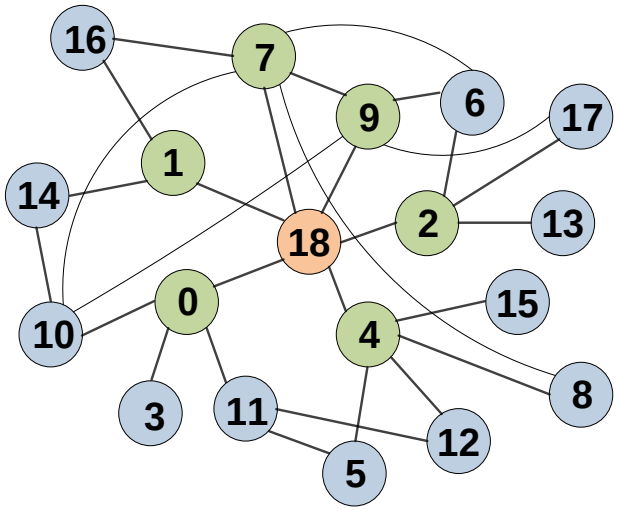
- Чтение памяти, необходимо подзаряжать конденсаторы
- Необходимость перезарядки конденсаторов (токи утечки)
- t_{RP} - время предварительной зарядки

- Каждая строка должна быть перезаряжена каждые 7.8 мкс

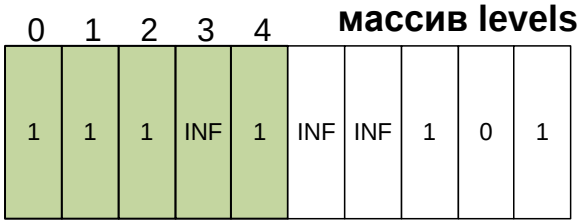


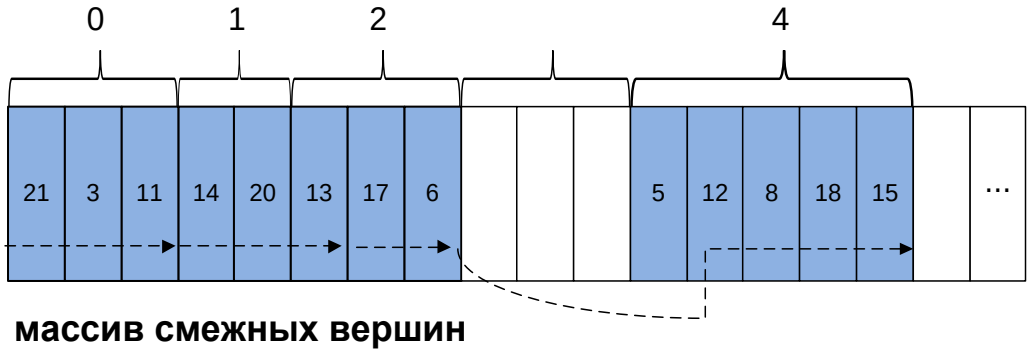
- **Задержки обращения в память при случайном доступе выше, чем при последовательном!**

Подход Read-based, алгоритм read

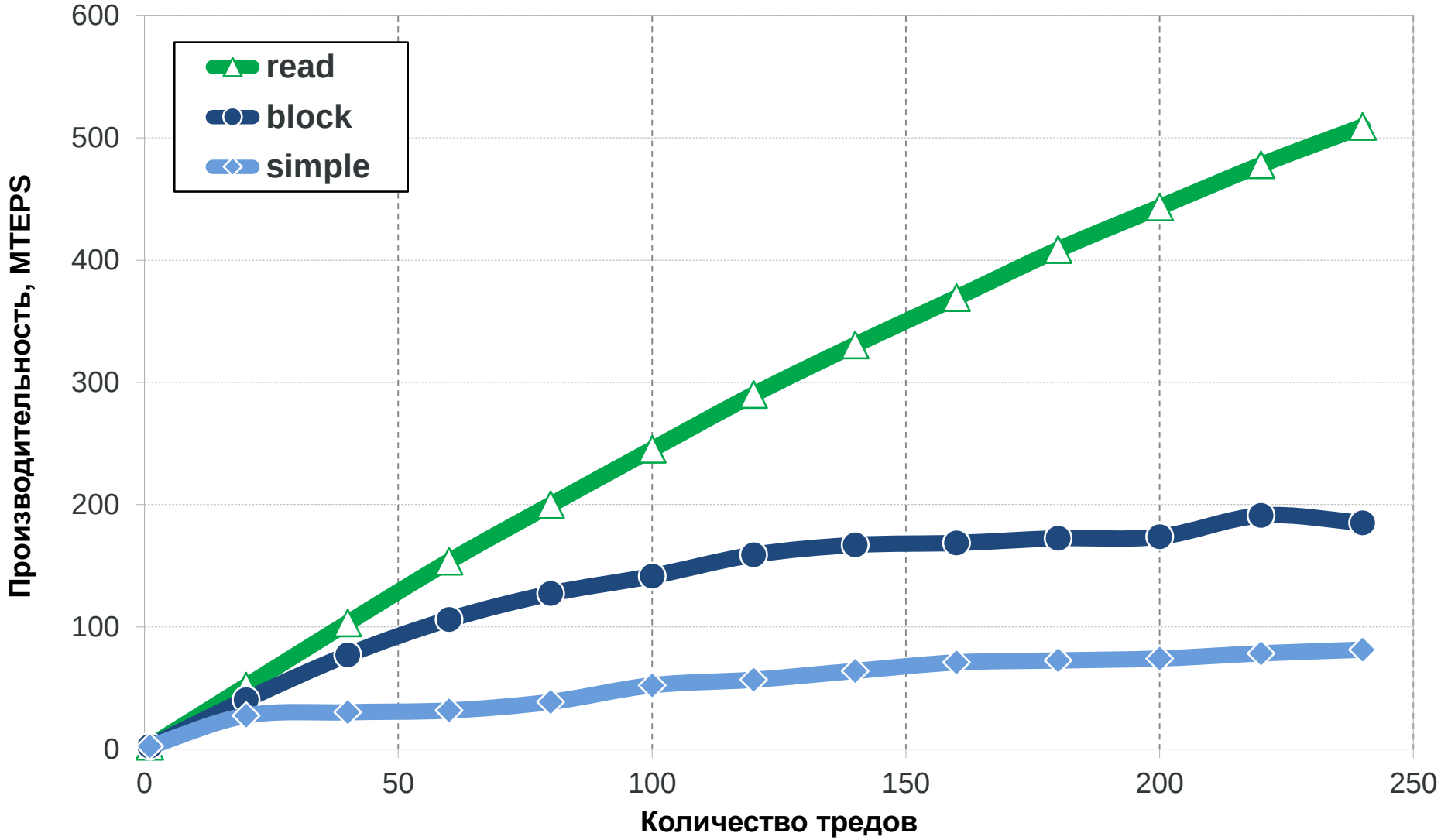



```
#pragma omp parallel for reduction (…)
for all vertex ∈ V do
    if levels[vertex] ≠ numLevel then
     continue
    for all w: (vertex, w) ∈ E  do
        if levels[w] == -1 then
            levels[w] = numLevel + 1
            nLevelVerts =  nLevelVerts + 1
        end if
    end for
end for
```



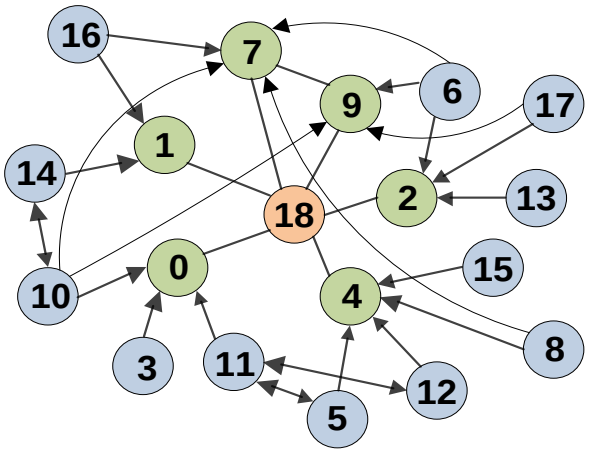


Производительность алгоритмов simple, block и read в зависимости от числа используемых тредов на сопроцессоре Phi-5110P

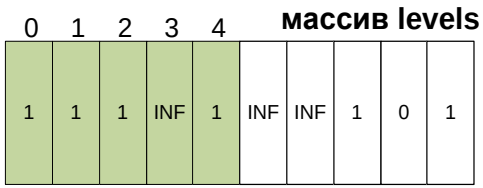


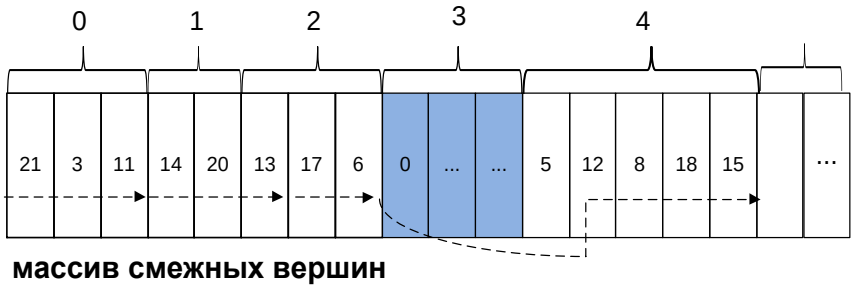
Число вершин в графе: N = 2^{27} (134 млн), средняя связность вершины: k = 8

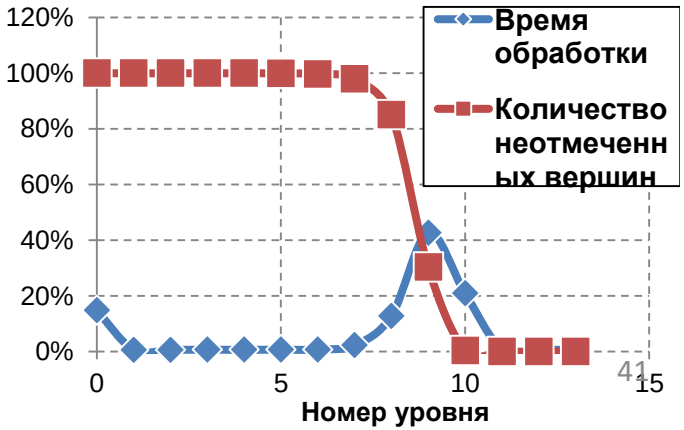
Алгоритм bottom-up-hybrid



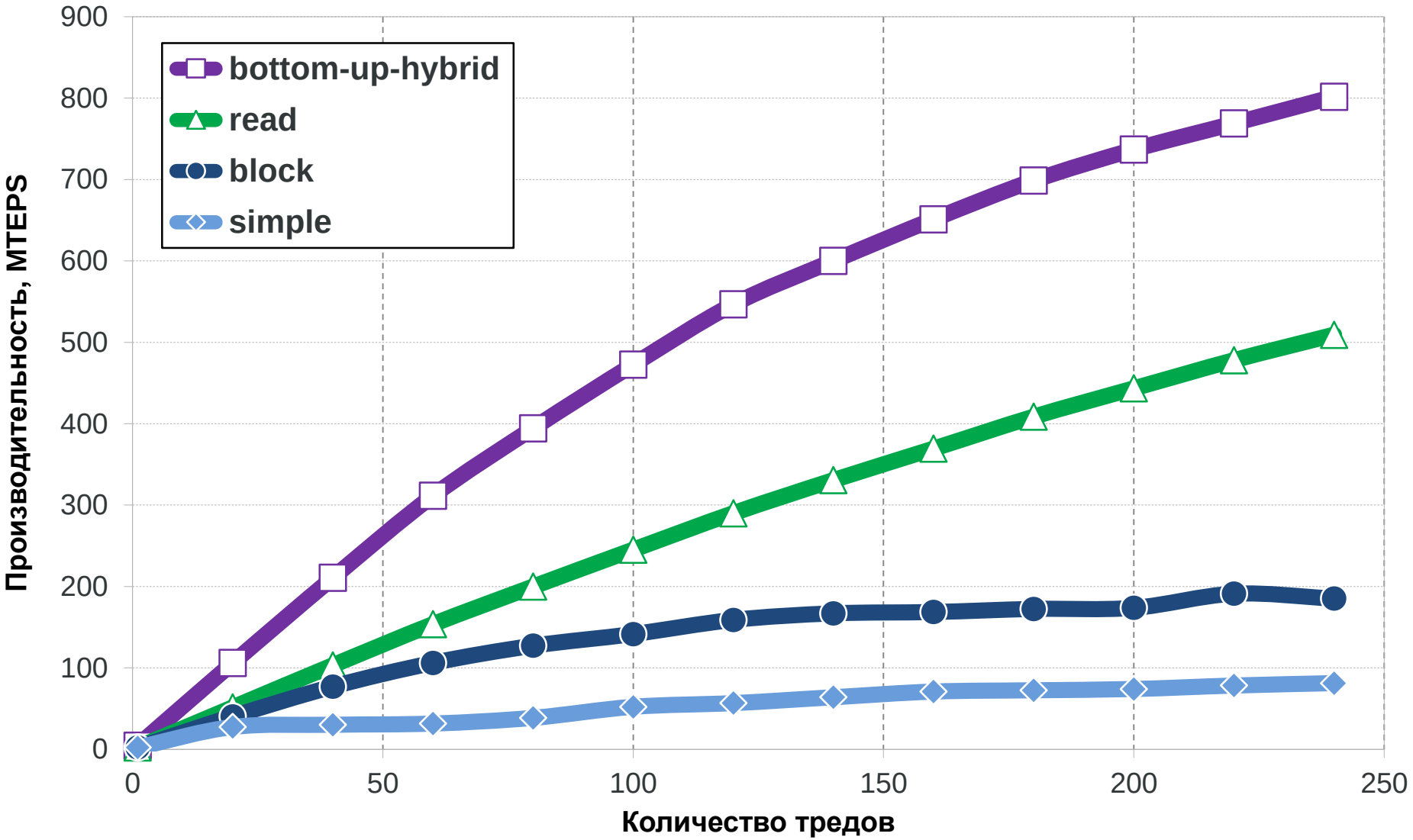

```
#pragma omp parallel for reduction (…)
for all vertex ∈ V do
        if levels[vertex] == -1 then
                for all w: (vertex, w) ∈ E  do
                        if levels[w] == numLevel then
                                levels[vertex] = numLevel + 1
                                nLevelVerts =  nLevelVerts + 1
                                break
                        end if
                end for
        end if
end for
```





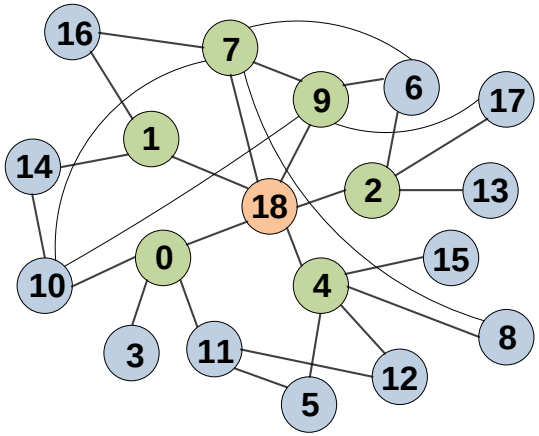


Производительность алгоритмов simple, block, read и bottom-up-hybrid в зависимости от числа используемых тредов на сопроцессоре Phi-5110P

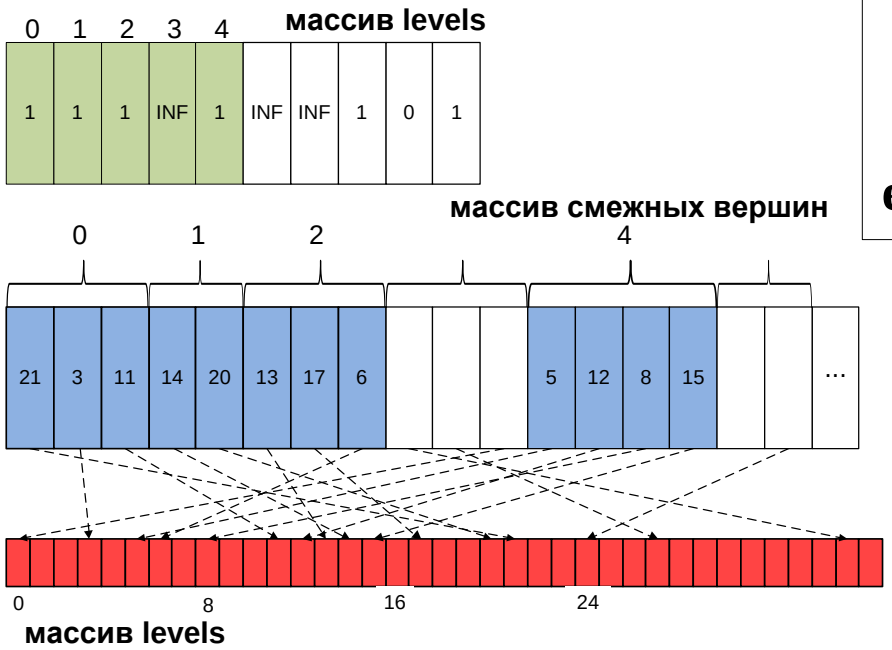


Число вершин в графе: N = 2^{27} (134 млн), средняя связность вершины: k = 8

Недостатки алгоритмов read и bottom-up-hybrid

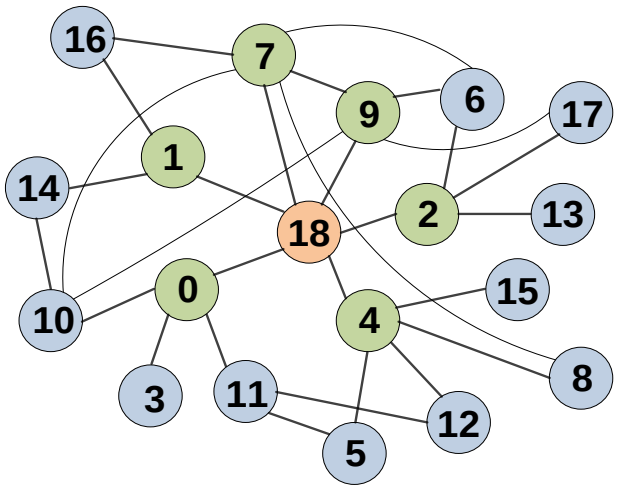


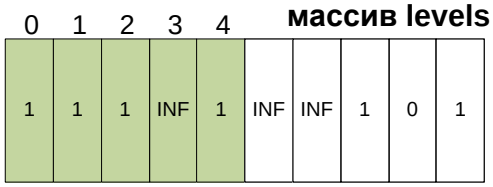

```
#pragma omp parallel for reduction
(…)
for all vertex ∈ V do
    if levels[vertex] ≠ numLevel then
        continue
    for all w: (vertex, w) ∈ E  do
        if levels[w] == -1 then
            levels[w] = numLevel + 1
        …
        end if
    end for
end for
```

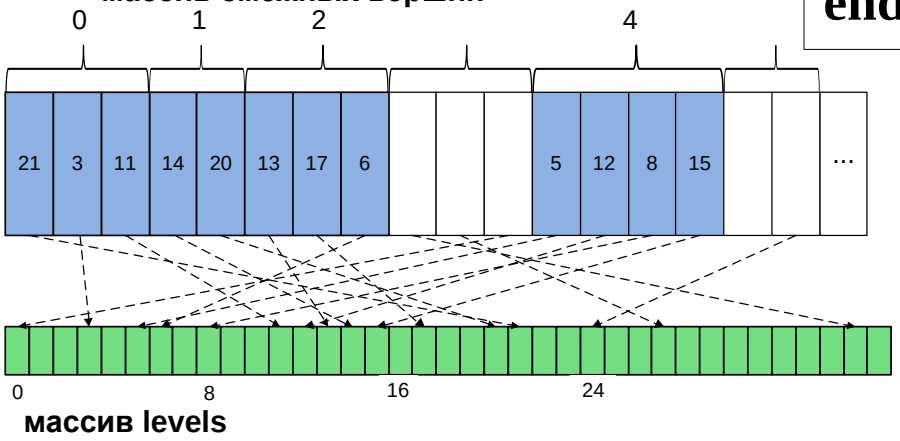


	SB	Phi-5110P
Частота, ГГц	2.2	1.05
Задержка обращения в память (такты)	~150	~300

Решение: ручная развертка цикла + использование prefetch



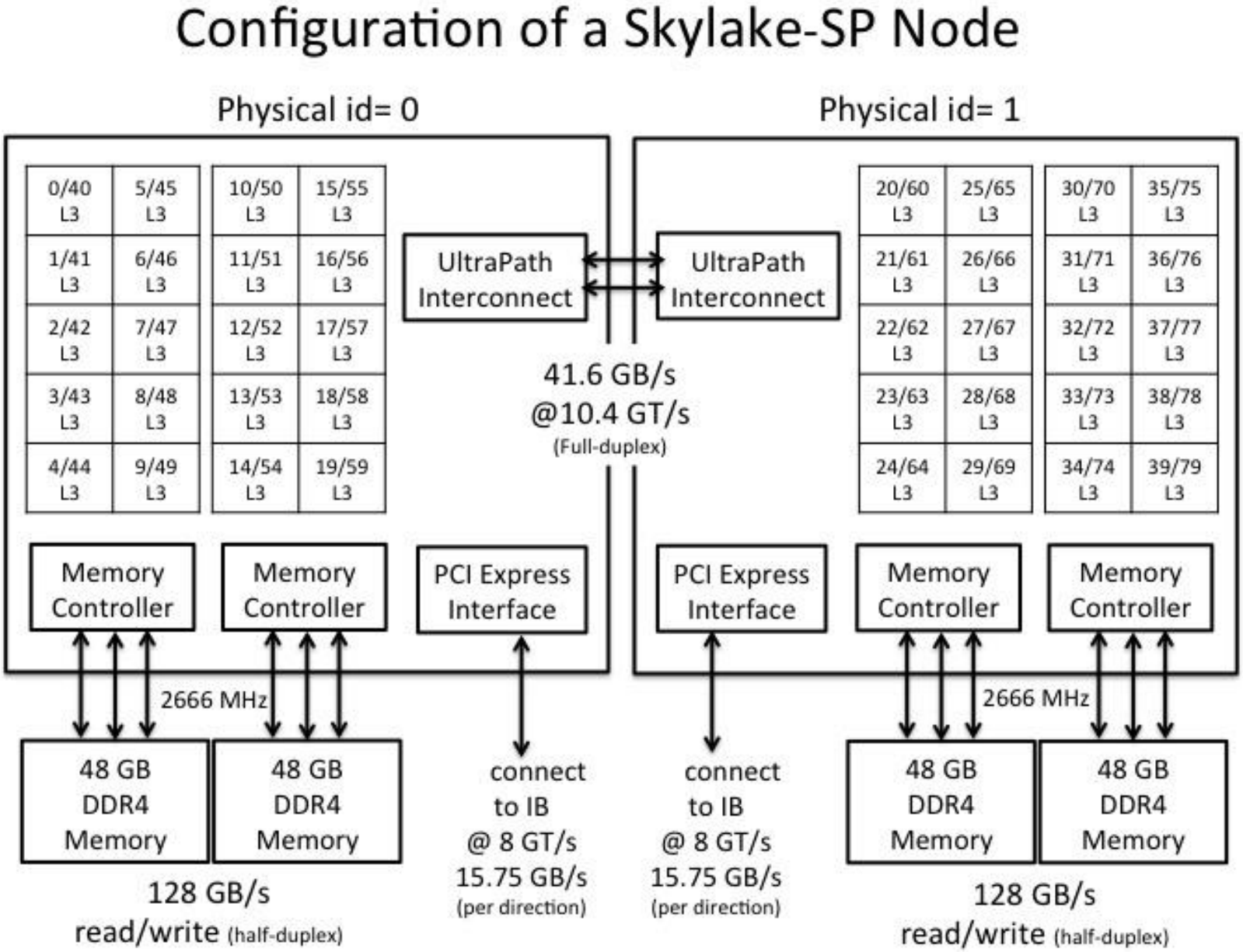




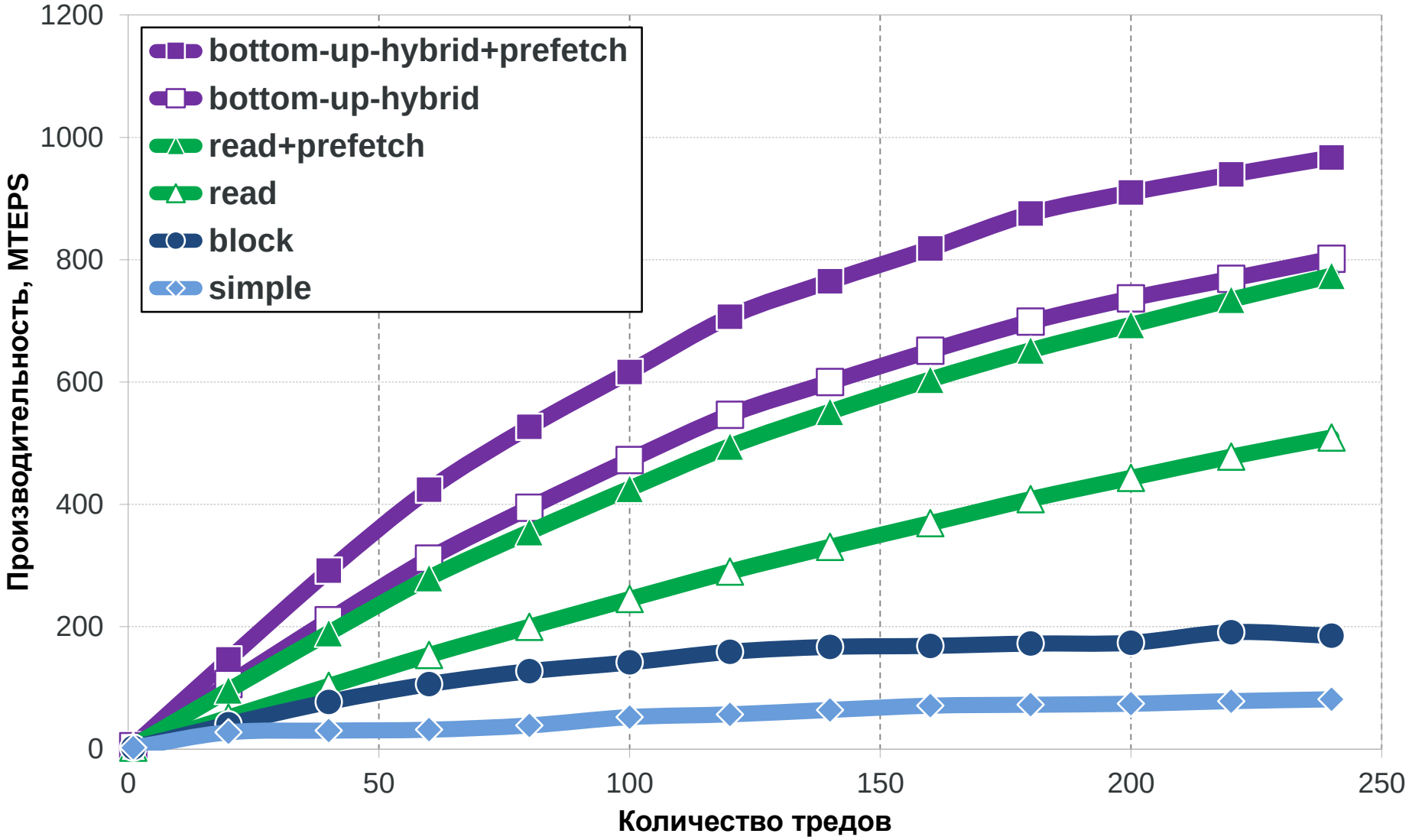

```
#pragma omp parallel for reduction (…)
for all vertex ∈ V do
    if levels[vertex] ≠ numLevel then continue
    for all w: (vertex, w) ∈ E  do
        prefetch(levels[w])
        …
        if levels[w] == -1 then
            levels[w] = numLevel + 1
        …
        end if
    end for
end for
```

	SB	Phi-5110P
Пиковая ПС памяти, ГБ/с	51	352
ПС чтения из памяти, ГБ/с; Последовательный / случайный доступ	42 / 3.3	183 / 3.8
Задержка, тактов	200	300

Архитектура процессора, контроллер DRAM

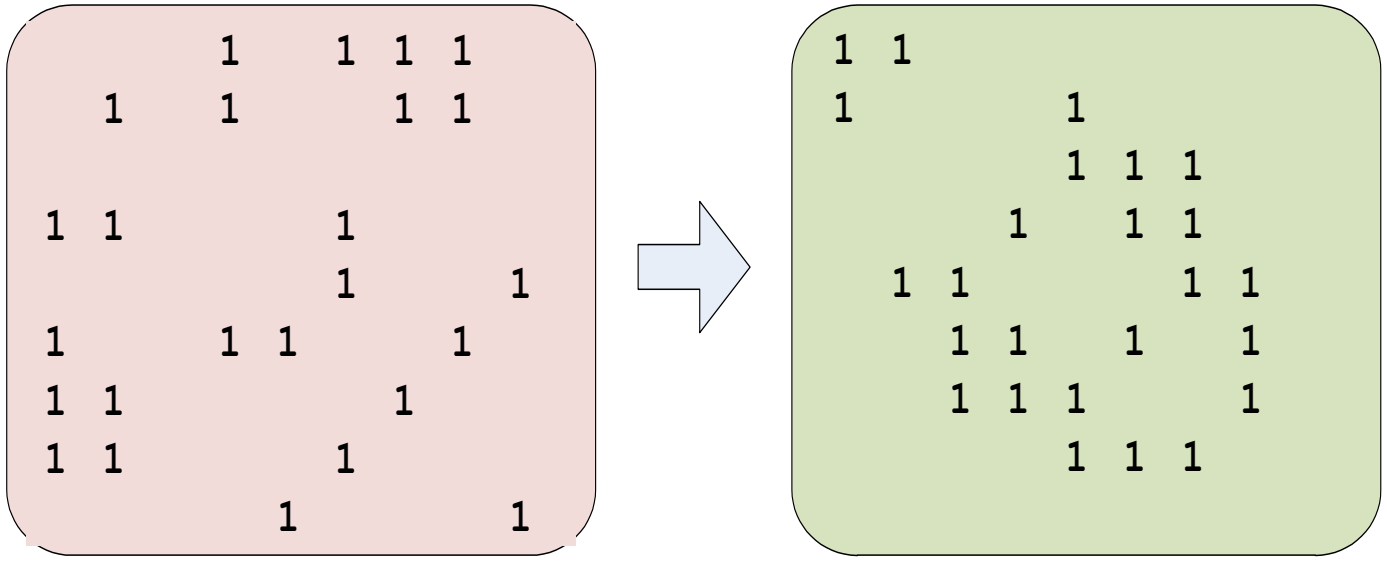


Производительность алгоритмов simple, block, read и bottom-up-hybrid с префетчем в зависимости от числа используемых тредов на сопроцессоре Phi-5110P



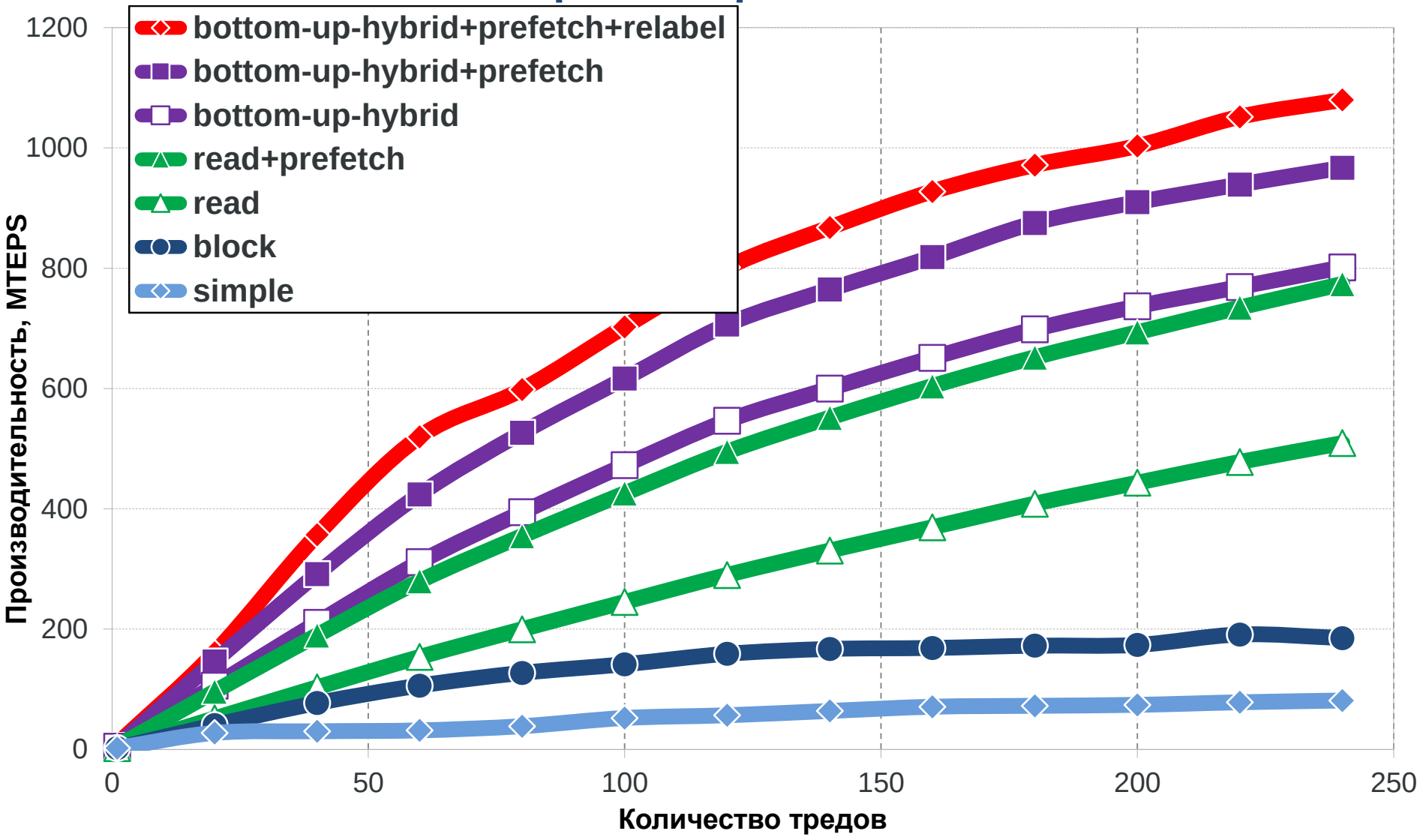
Число вершин в графе: N = 2^{27} (134 млн), средняя связность вершины: k = 8

Улучшение локализации: перестановка вершин



- Матрица смежности приводится к ленточному виду с уменьшением ширины ленты (алгоритм Reverse Cuthill-McKee) => уменьшается количество кэш-промахов
- Списки смежных вершин сортируются => уменьшается количество промахов в TLB
- Использование больших страниц

Производительность различных алгоритмов, с префетчем и перестановками в зависимости от числа используемых тредов на сопроцессоре Phi-5110P

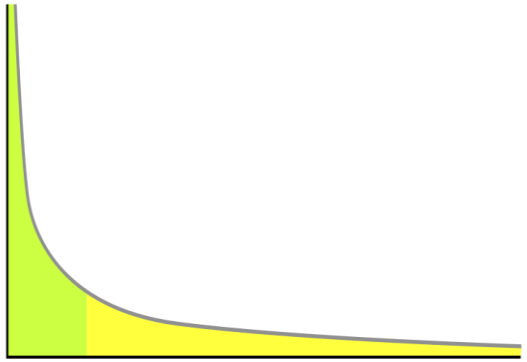


Число вершин в графе: N = 2^{27} (134 млн), средняя связность вершины: k = 8

Распараллеливание: дисбаланс вычислительной нагрузки

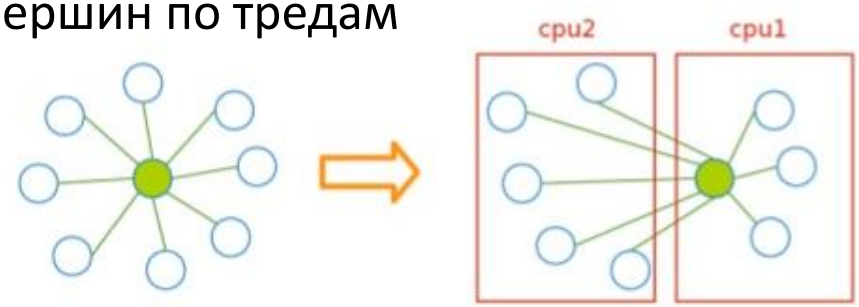
- **Проблема: неравномерность** итераций циклов

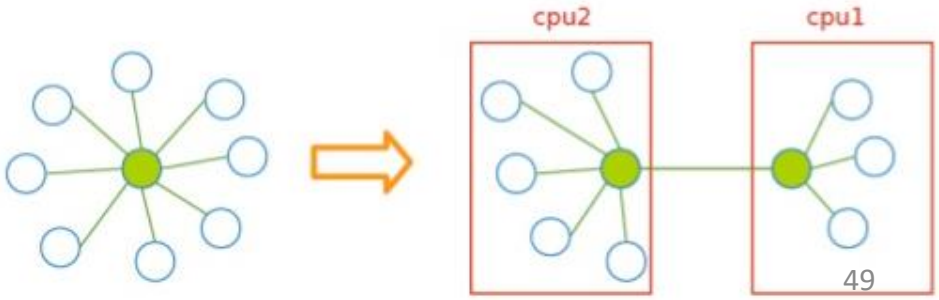
```
# pragma omp parallel for
for (int u = 0; u < G->n; u++)
      for (int j = G->rowsIndices[u]; j < rowsIndices[u+1]; j++) {
            ……
      }
```



- **Решение 1:** #pragma omp parallel for **schedule (guided)** – для **динамического** распределения вершин по тредам

- **Решение 2:** На этапе предобработки выполнение процедуры Vertex-cut: разделение вершины и **разрезание** списков смежности вершин





Большой объем памяти

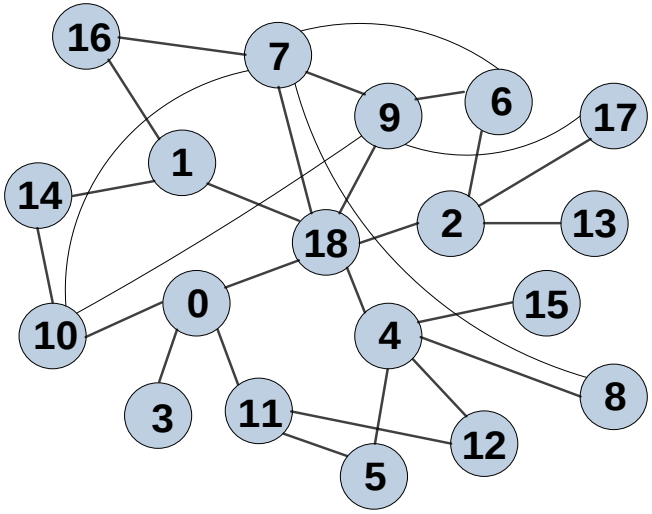
- **Проблема**: постоянная смена данных в кэше, низкие характеристики при случайном доступе
- **Решения** на этапе предобработки:
  - Хранение только половины графа (для неориентированного)
  - Удаление кратных ребер
  - Перестановка вершин (Cuthill-McKee)
  - Сжатие данных
    - edge_id_t: uint64_t --> uint32_t
  - Сортировка ребер каждой вершины
  - Сортировка всех ребер графа

Резюме: проблемы и подходы к решению задач в рамках одного узла

- Выбор оптимального представления графа
- По возможности организация последовательного доступа к данным
- По возможности избегать использовать межпотоковые синхронизации
- Стремиться работать не на задержке обращений к памяти, а на темпе
- Улучшение локализации
- Алгоритмические оптимизации
- Сжатие данных
- Аккуратная работа с памятью внутри NUMA-вычислительного узла
- Балансировка нагрузки
- Аккуратно измерять производительность

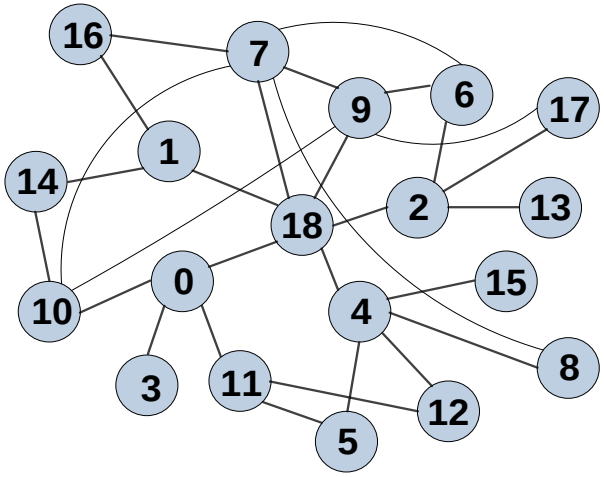
Проблемы и подходы к решению графовых задач на распределенной памяти

Представление графа

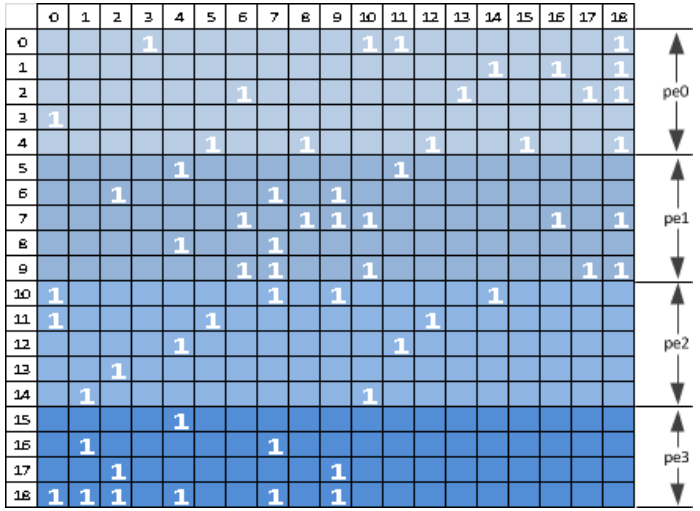


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0				1							1	1							1
1															1		1		1
2							1							1				1	1
3	1																		
4						1			1				1			1			1
5					1							1							
6			1					1		1									
7							1		1	1	1						1		1
8					1			1											
9							1	1			1							1	1
10	1							1		1					1				
11	1					1							1						
12					1							1							
13			1																
14		1									1								
15					1														
16		1						1											
17			1							1									
18	1	1	1		1			1		1									

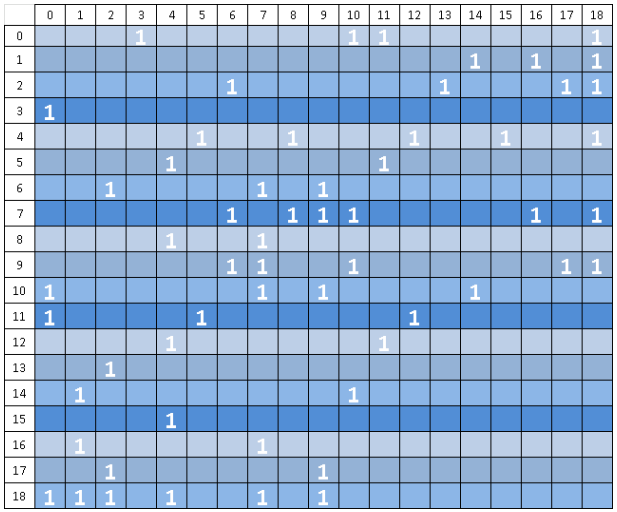
Распределение данных



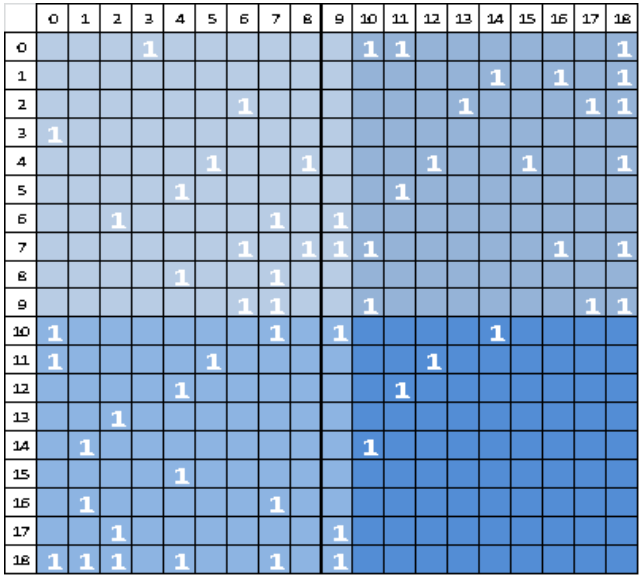
1D, блоками



1D, с чередованием

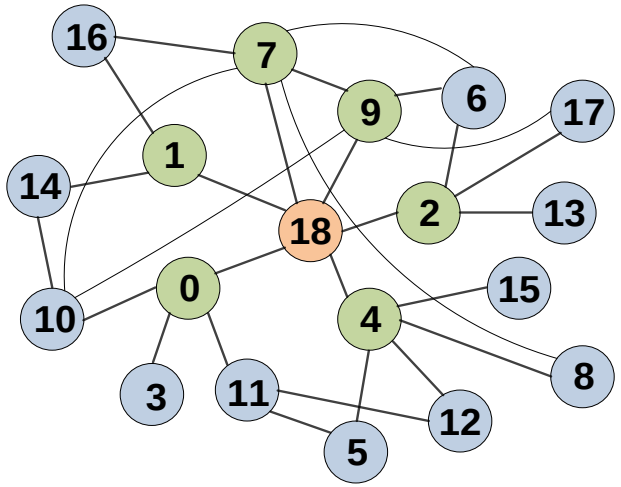


2D



Поиск вширь в графе, распределенная версия

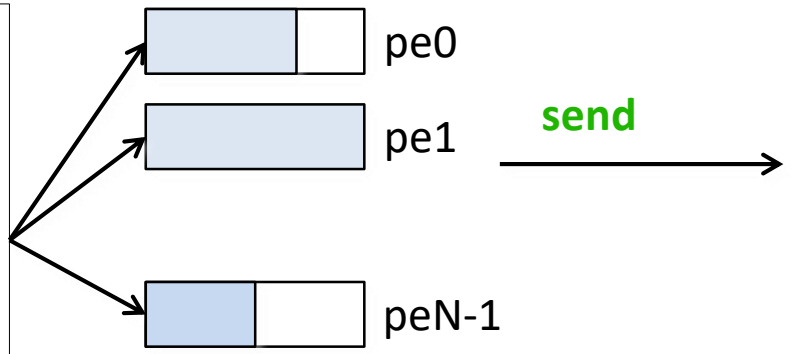
```
function ProcessQueue(Q, E)
    for all vertex ∈ Q  do
        for all w : (vertex, w) ∈ E  do
            send vertex, w to owner(w)
        end for
    end for
end function
```




```
function Receive(vertex, w)
    if w ∉ Visited then
        Q_next = Q_next ∪ w
        Visited = Visited ∪ w
        Parents(w) = vertex
    end if
end function
```

Поиск вширь в графе, агрегация сообщений

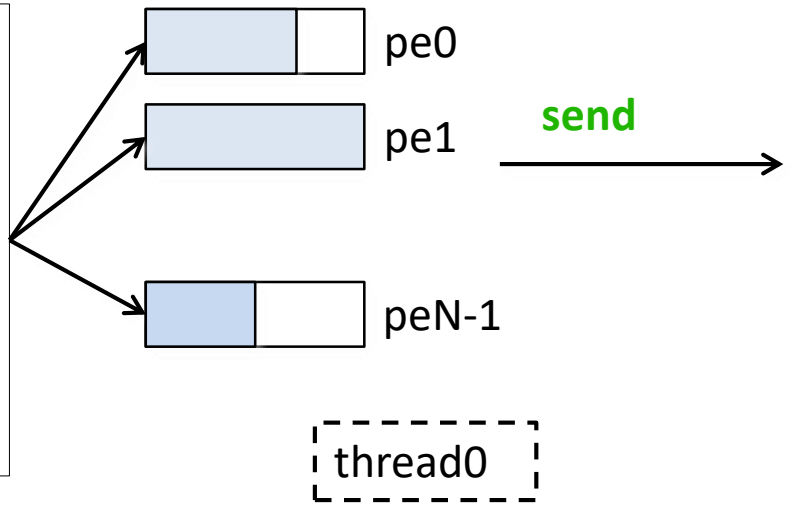
```
function ProcessQueue(Q, E)
    for all vertex ∈ Q  do
        for all w : (vertex, w) ∈ E  do
            send vertex, w to owner(w)
        end for
    end for
end function
```




```
function Receive(vertex, w)
    if w ∉ Visited then
        Q_next = Q_next ∪ w
        Visited = Visited ∪ w
        Parents(w) = vertex
    end if
end function
```

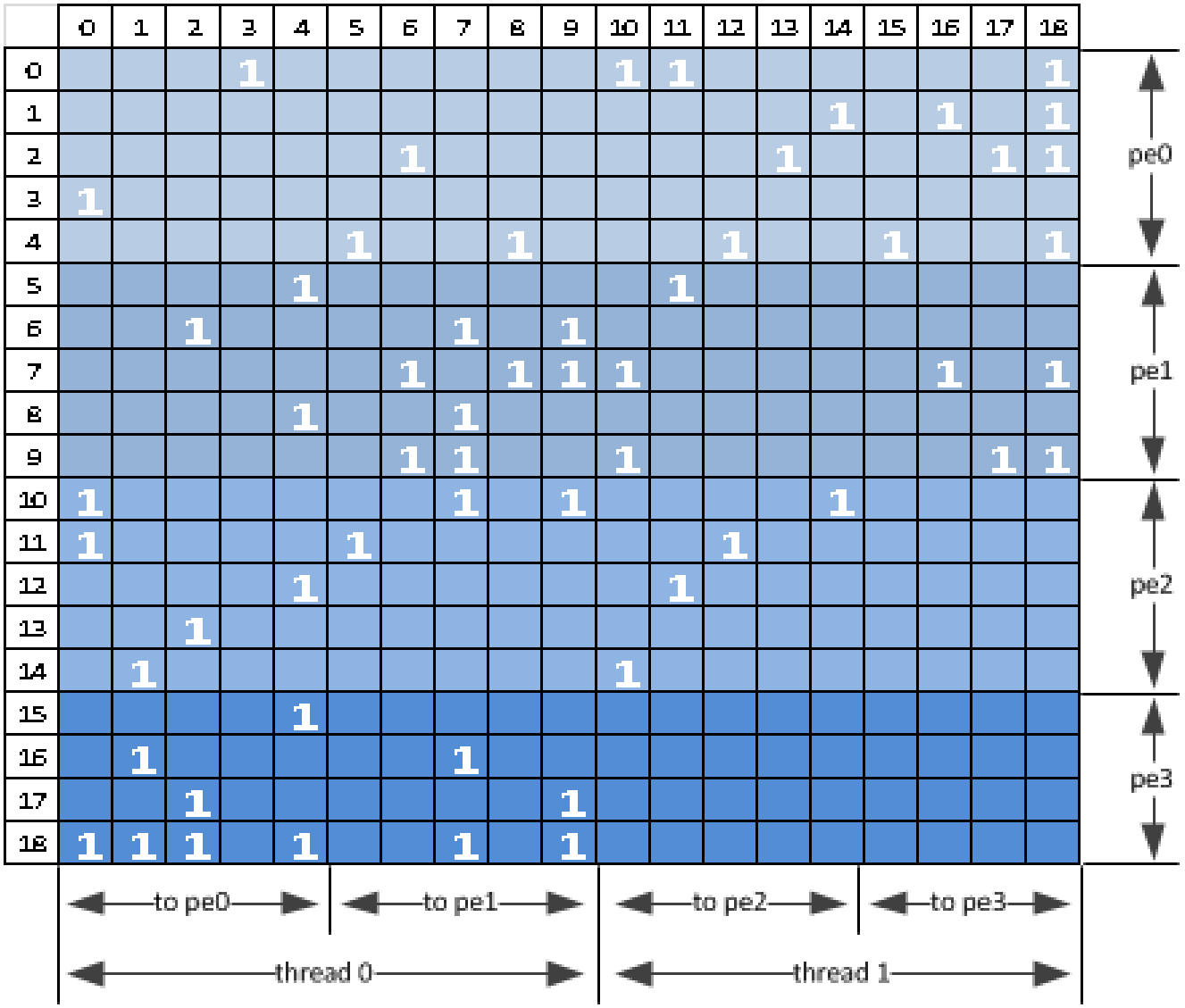
Поиск вширь в графе, параллельная отправка и прием

```
function ProcessQueue(Q, E)
    for all vertex ∈ Q  do
        for all w : (vertex, w) ∈ E  do
            send vertex, w to owner(w)
        end for
    end for
end function
```

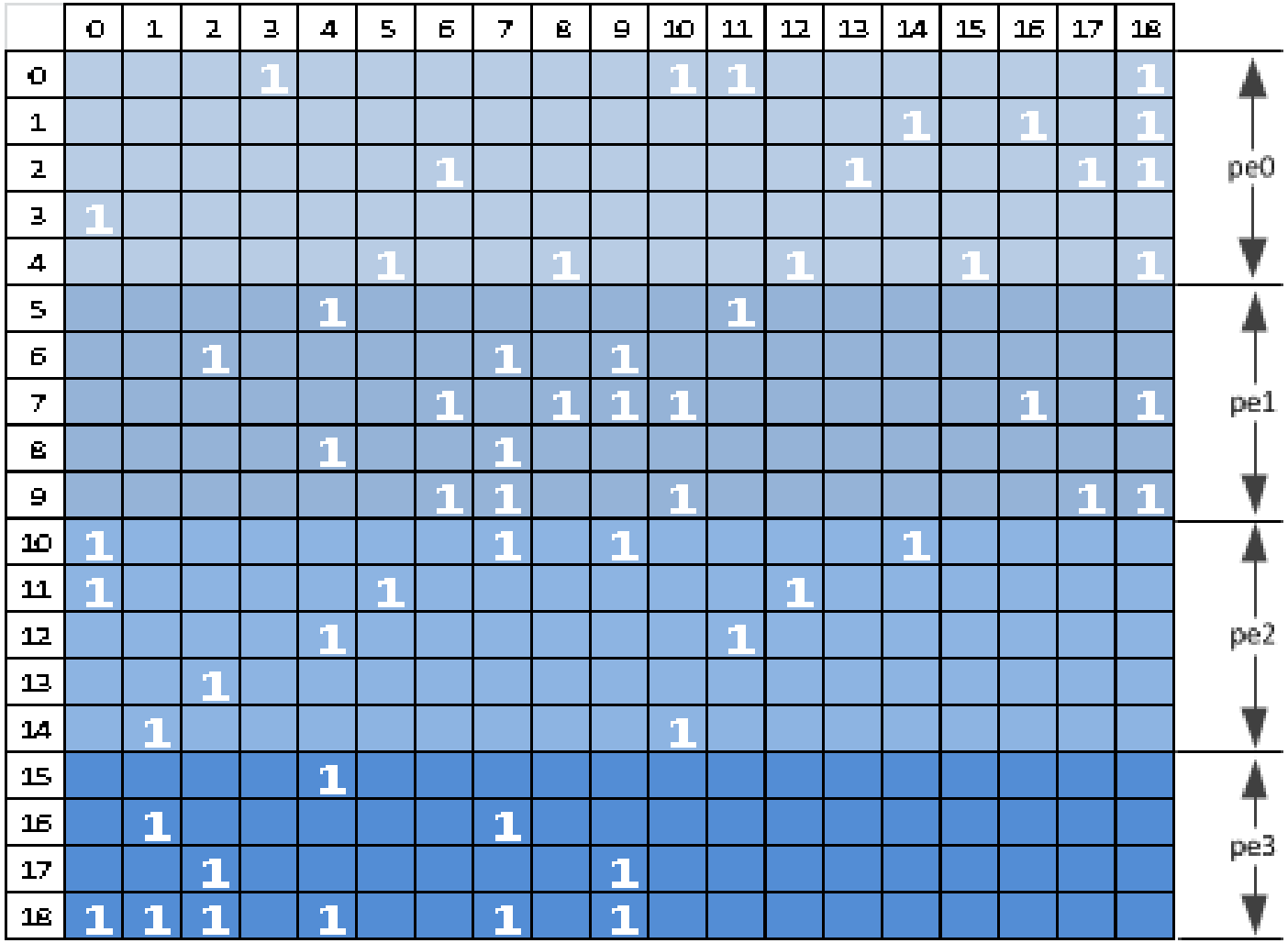



```
function Receive(vertex, w)
    if w ∉ Visited then
        Q_next = Q_next ∪ w
        Visited = Visited ∪ w
        Parents(w) = vertex
    end if
end function
```

Организация параллелизма потоков



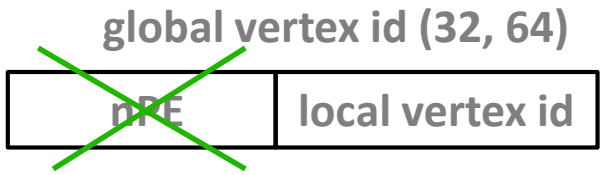
Хаотично расположенные вершины и ребра графа

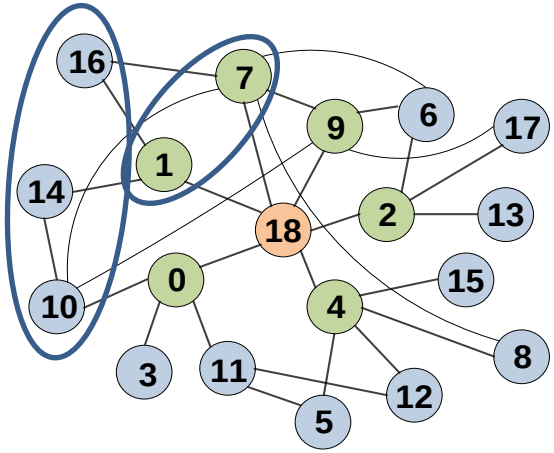


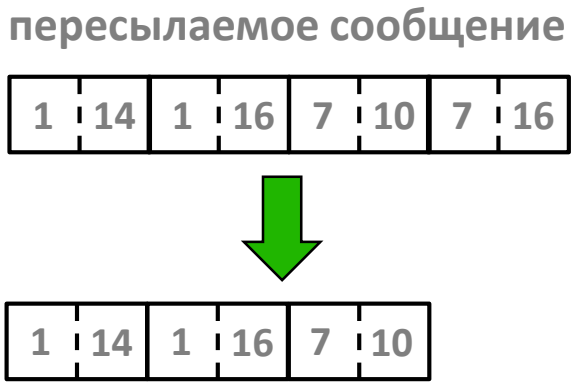
Шаблон обменов all-to-all

Уменьшение количества пересылаемых данных

- **Использование простаивающего процессора**
- **Сокращение пересылок**
  - **Отказ от лишней пересылаемой информации**
  - **Удаление дублирующей информации**
- **Сжатие данных**
  - **Использование знаний о структуре графа**







Графы реального мира. Степенной закон

- WWW, Социальные сети, Биоинформатика
- Графы small-world

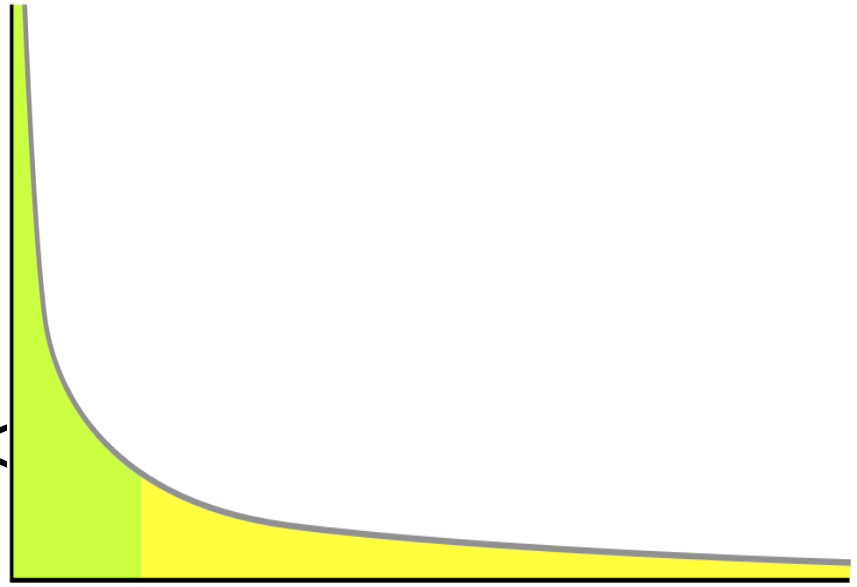
$L \sim \log N$,

- scale-free – графы,

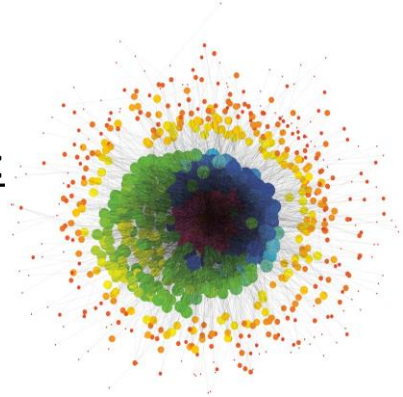
доля $P(k) \sim k^{-tau}$, $2 < tau <$

k – связность вершины

$L \sim \log \log N$



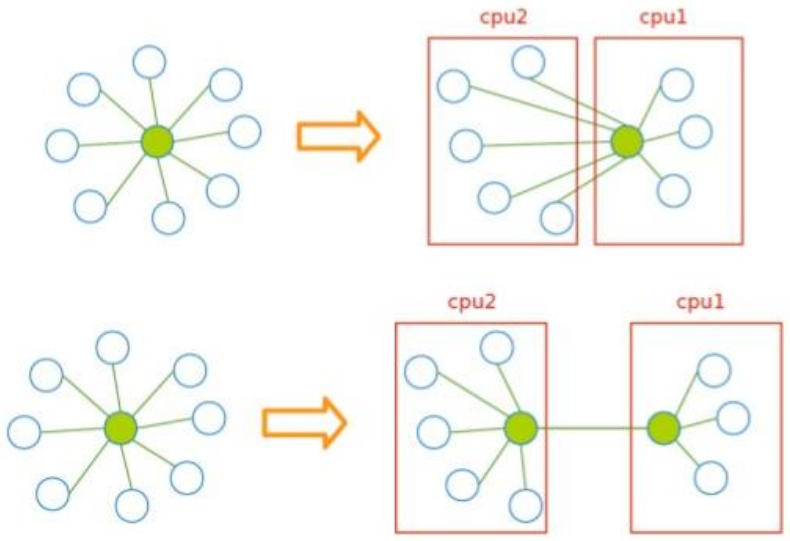
Граф Кронекера:



Балансировка нагрузки

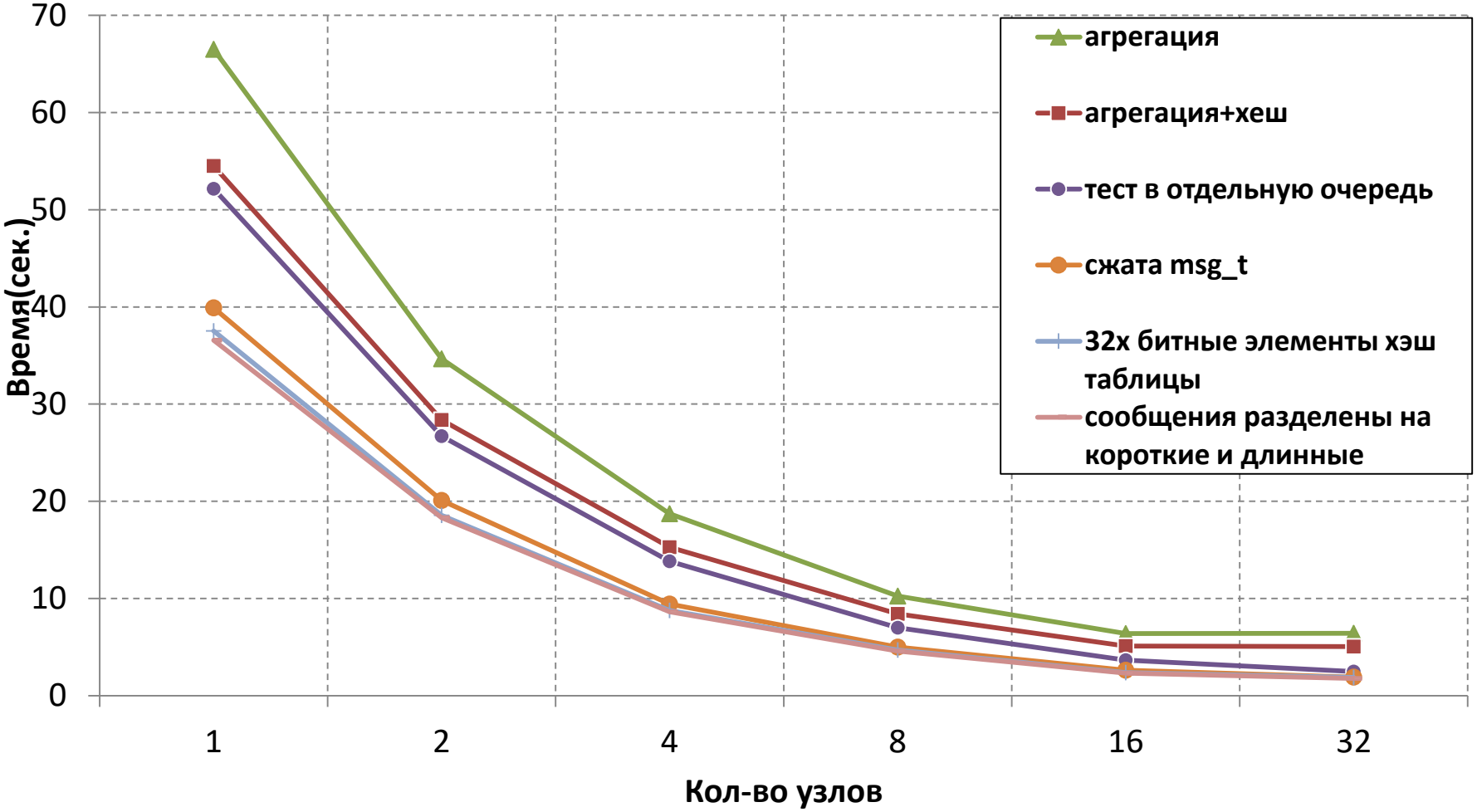
- При использовании большого числа вычислительных узлов особенно важна равномерная загрузка
- **Решение1:** На этапе предобработки выполнение процедуры Vertex-cut: разделение вершины и **разрезание** списков смежности вершин
- **Решение2:**

```
function ProcessQueue(Q, E)
    for all vertex ∈ Q  do
        for all w : (vertex, w) ∈ E  do
       if w ∈ Heavy then
            Out^H = Out^H ∪ w
            else
                send vertex, w to owner(w)
            end if
        end for
    end for
    broadcast Out^H
end function
```



Задача поиска минимального остовного дерева (MST)

Алгоритм Gallagher, Humblet, Spira. Сеть Ангара



Граф RMAT-23, средняя связность - 32

Проблемы и подходы к решению задач на распределенной памяти

- Выбор распределения данных
- Агрегация сообщений
- Организация внутриузлового параллелизма
- Уменьшение количества пересылаемых данных
- Балансировка нагрузки
- Использование эффективных коммуникаций
  - Аккуратно использовать MPI
- Алгоритмические оптимизации

Вопросы?

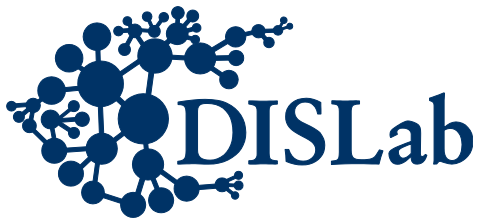
alxdr.semenov@gmail.com

Современные суперкомпьютерные системы

Александр Сергеевич Семенов

www.dislab.org





План

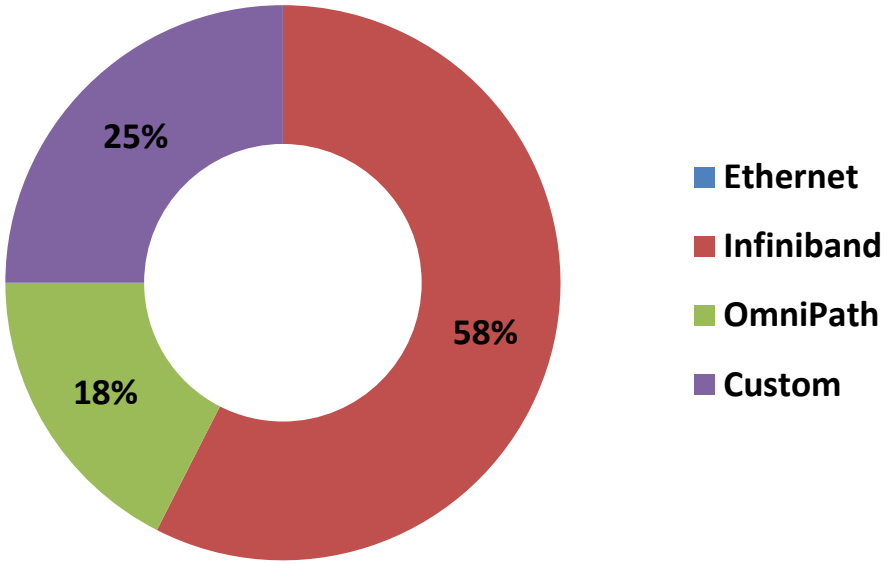
- Top500
- Теоретические сведения о сетях
- Сети
  - Intel Omni Path
  - Mellanox Infiniband EDR, HDR
  - TofuD
  - Cray Aries, Slingshot
  - Ангара, Ангара-2

Top500 (июнь 2020)

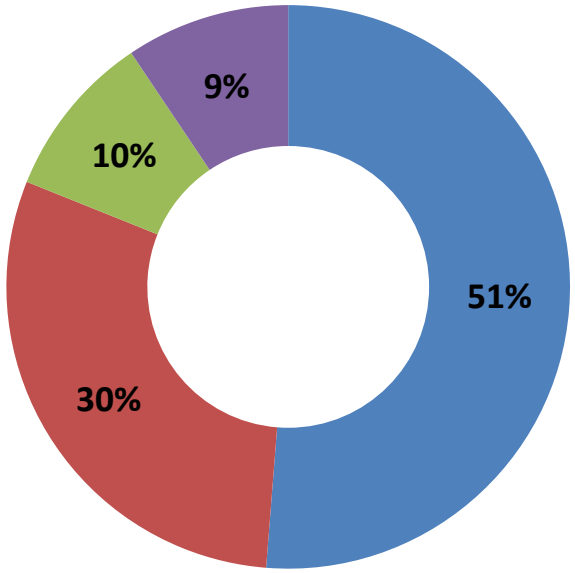
N	System	Peak, PFlops
1	Fugaku. Fujitsu A64FX 48c, TofuD	513.9
2	Summit. IBM Power9 22c, NVIDIA Volta GV100, IB EDR	200.8
3	Sierra. IBM Power9, NVIDIA Volta GV100, IB EDR	125.7
4	Sunway TaihuLight. SW 26010 260C, Sunway & Mellanox	125.4
5	Tianhe-2A. Intel Xeon E5-2692v2 12C, TH Express-2, Matrix-2000 , NUDT	100.7
6	HPC5. Dell EMC. Xeon Gold 6252 24C, NVIDIA Tesla V100, IB HDR	35.4

Top500 vs Interconnects

Top40



Top500





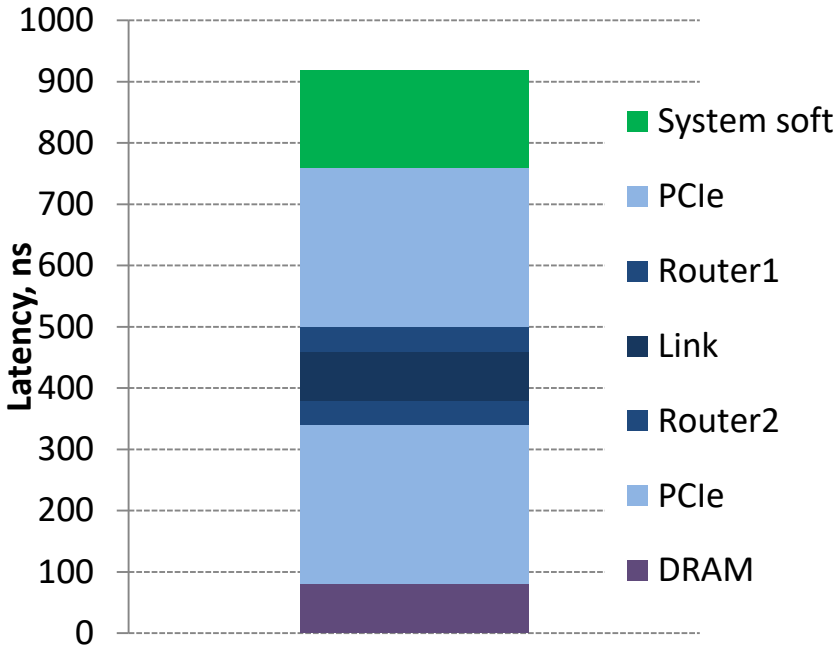
Характеристики коммуникационных сетей

- Задержка передачи сообщений
- Пропускная способность
- Темп выдачи сообщений
- Бисекционная пропускная способность

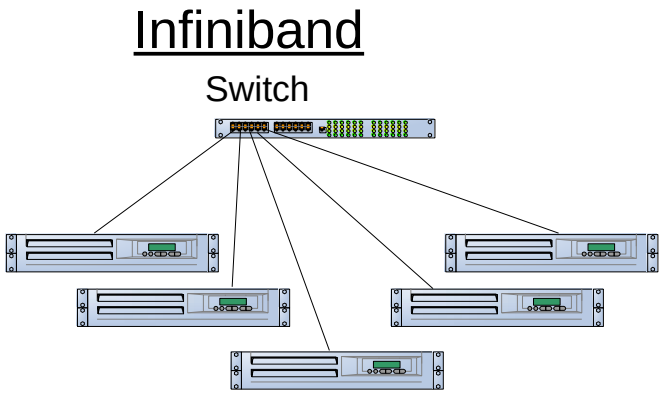
- Топология
- RDMA

Характеристики производительности: задержка



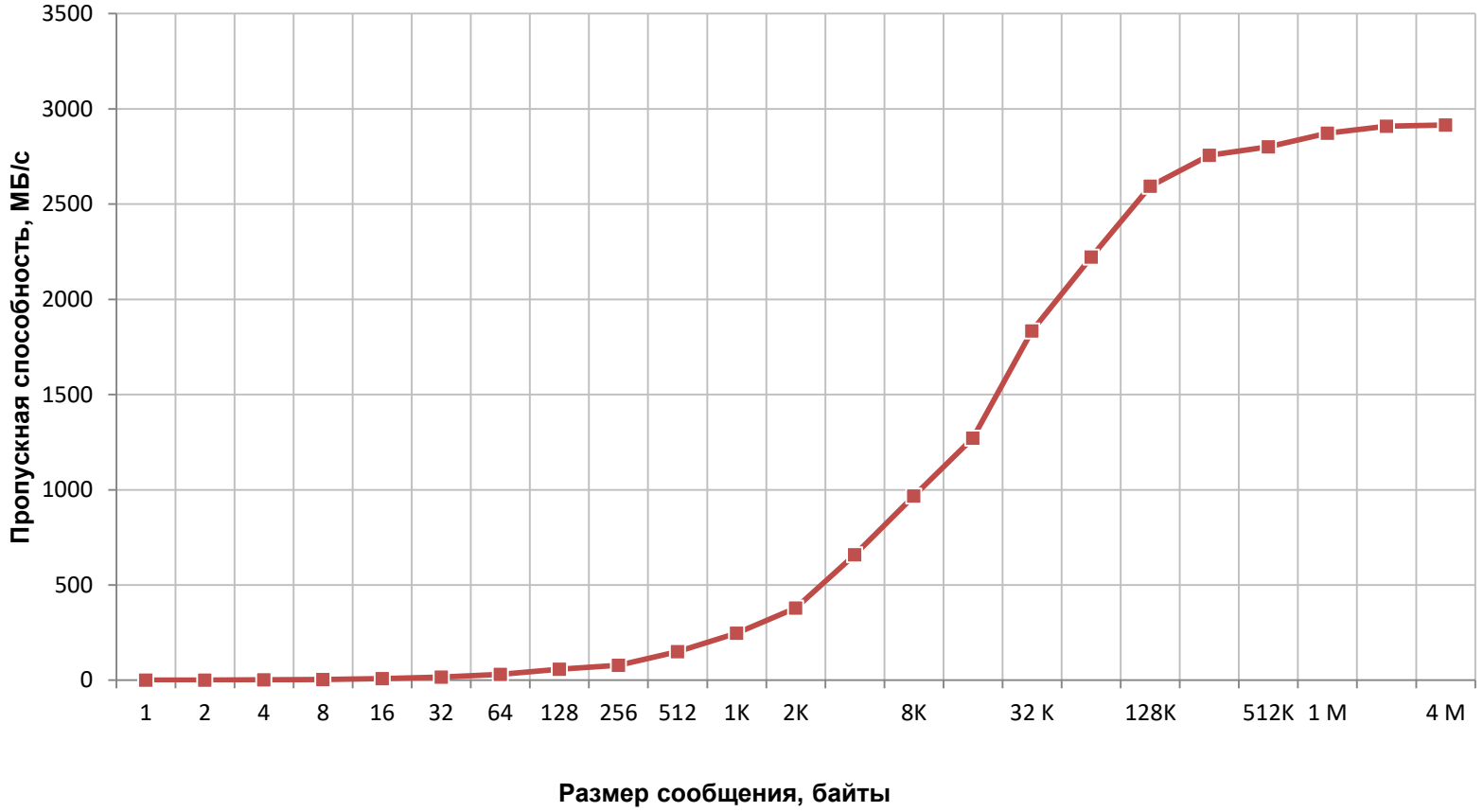


Суперкомпьютер «Ломоносов», сеть – Infiniband 4xQDR



$$t_{comm} = t_{start} + d * t_{hop} + size * t_{bw}$$

Характеристики производительности: пропускная способность



Суперкомпьютер «Ломоносов», сеть – Infiniband 4xQDR

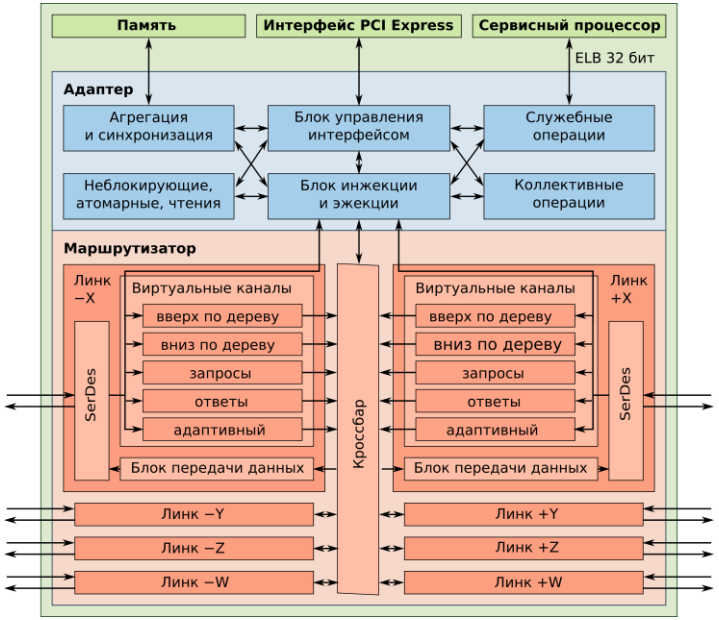
$$t_{comm} = t_{start} + d * t_{hop} + size * t_{bw}$$

Эффективность программ:

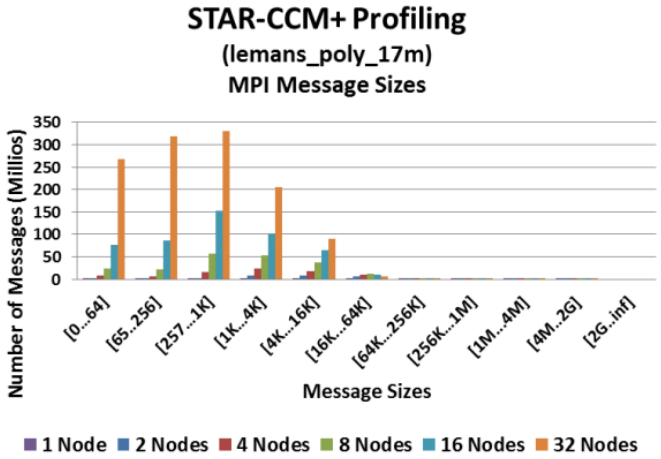
- Агрегация больших сообщений
- Минимизация расстояний пересылок
- Минимизация объемов коммуникаций

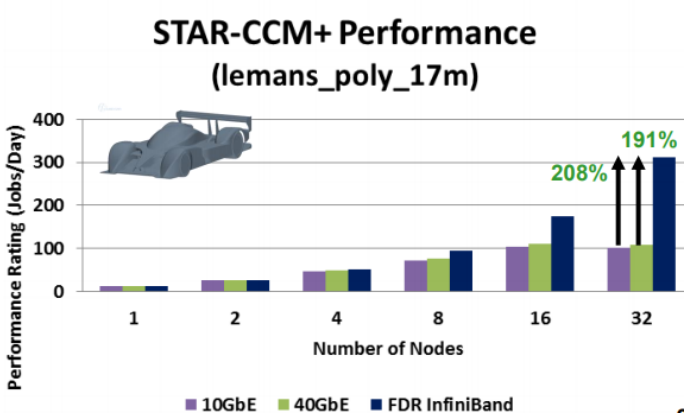
Характеристики производительности: темп выдачи сообщений

- MessageRate = BW (L) / L
- L – размер сообщений
- B – пропускная способность линка
- MessageRate < L/B



Dell PowerEdge R720xd 32-node (640-core) Jupiter cluster – Dual-Socket Hexa-Core Intel E5-2680 V2 @ 2.80 GHz CPUs. **20 процессов на узел**



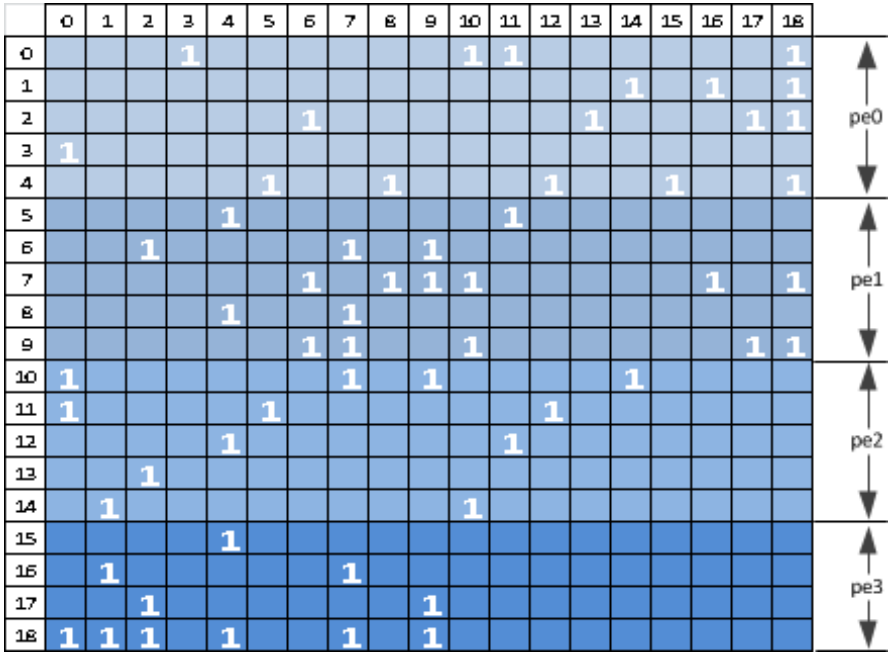


http://www.hpcadvisorycouncil.com/pdf/STAR-CCM_Analysis_Intel_E5_2680_V2.pdf

Характеристики производительности: бисекционная пропускная способность

- **Бисекционная плоскость – минимальный разрез, который разделяет сеть на две равные связные части**
- Бисекционная пропускная способность – пропускная способность каналов связи через бисекционную плоскость
- **В случае равномерных случайных посылок (all-to-all) каждый узел посылает сообщение через бисекционную плоскость с вероятностью ½**
- **Посылают все узлы – для линейной масштабируемости требуется N/2 линков в бисекционной плоскости**

Хаотично расположенные вершины и ребра графа

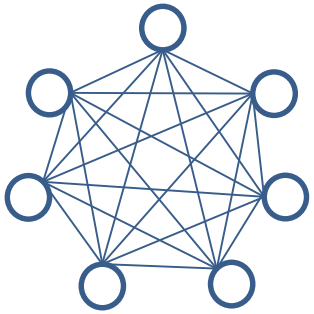


Шаблон обменов all-to-all

Топологии сетей (1)

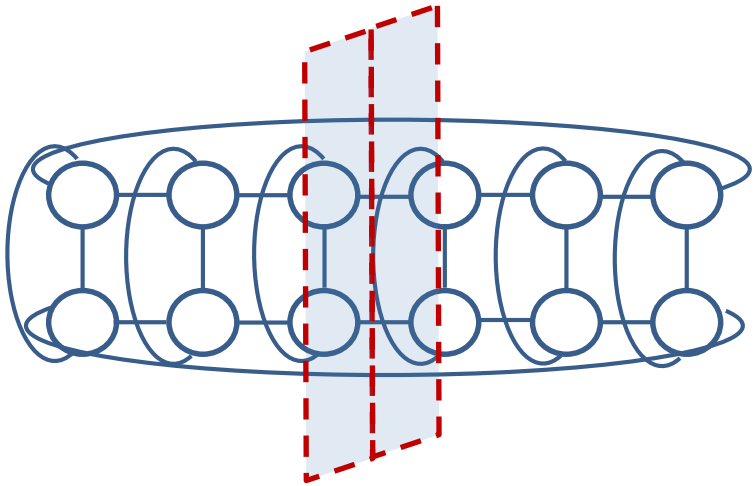
Полносвязная

- D_{max} = 1
- Идеальная топология: нет конкуренции, наименьшая задержка
- Высокая цена:
  - O(N) портов
  - $O(N^2)$ линков
  - не масштабируется



kD-тор = n1 x n2 x … x nk

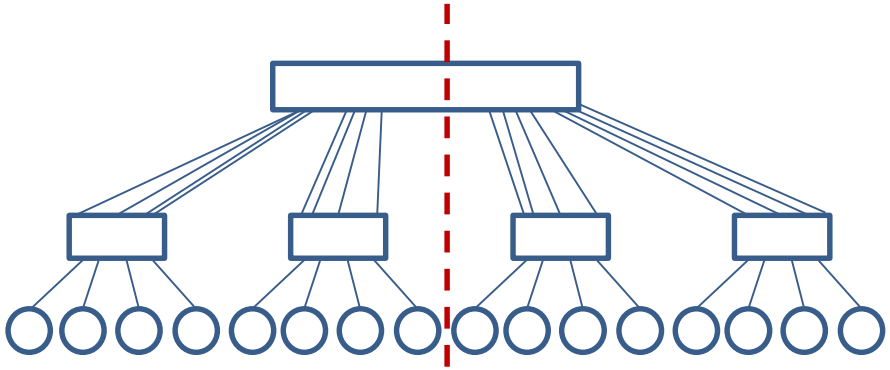
- D_{max} = $\lfloor n1/2 \rfloor + \lfloor n2/2 \rfloor + ... + \lfloor nk/2 \rfloor$
- Масштабируется
- Большое количество вариантов путей
- Бисекционная ширина 2N / Nmax
- low-radix



Топологии сетей (2)

Жирное дерево (FatTree)

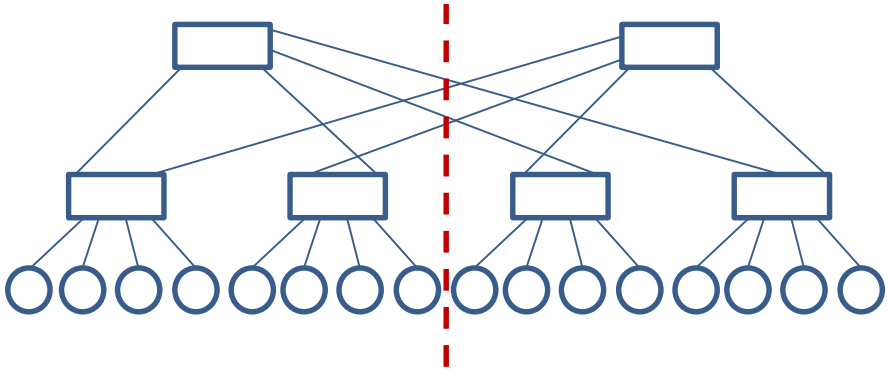
- непрямая
- $D_{max} = 4$
- Масштабируется
- Большое количество вариантов путей
- Бисекционная ширина N/2 (N/4, …)



FatTree – Folded Clos

- k – количество портов коммутатора
- c – количество узлов у коммутатора
- неблокирующая, блокирующая

блокирующая 2:1, k = 6

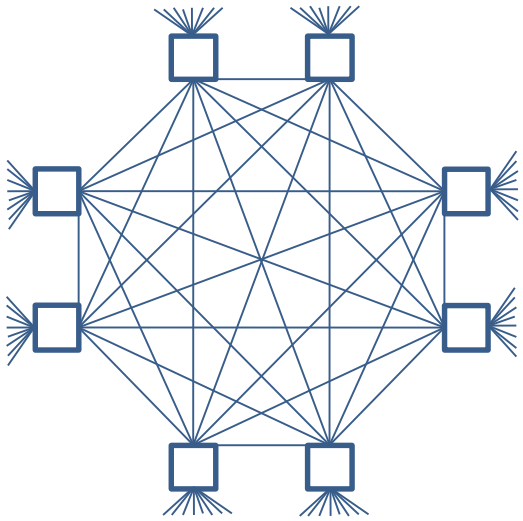


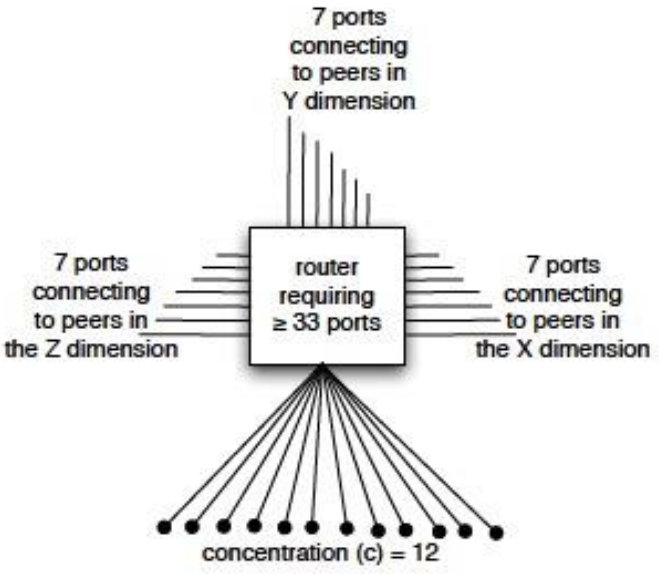
Топологии сетей (3)

Flattened Butterfly, 2007, FB(c, k-ary, n-flat)

- каждое измерение внутри полносвязно
- c – количество узлов у коммутатора
- k – размер измерения
- n – количество измерений
- $D_{max} = 2+n-1$

FB(c=8, 8-ary, 2-flat)



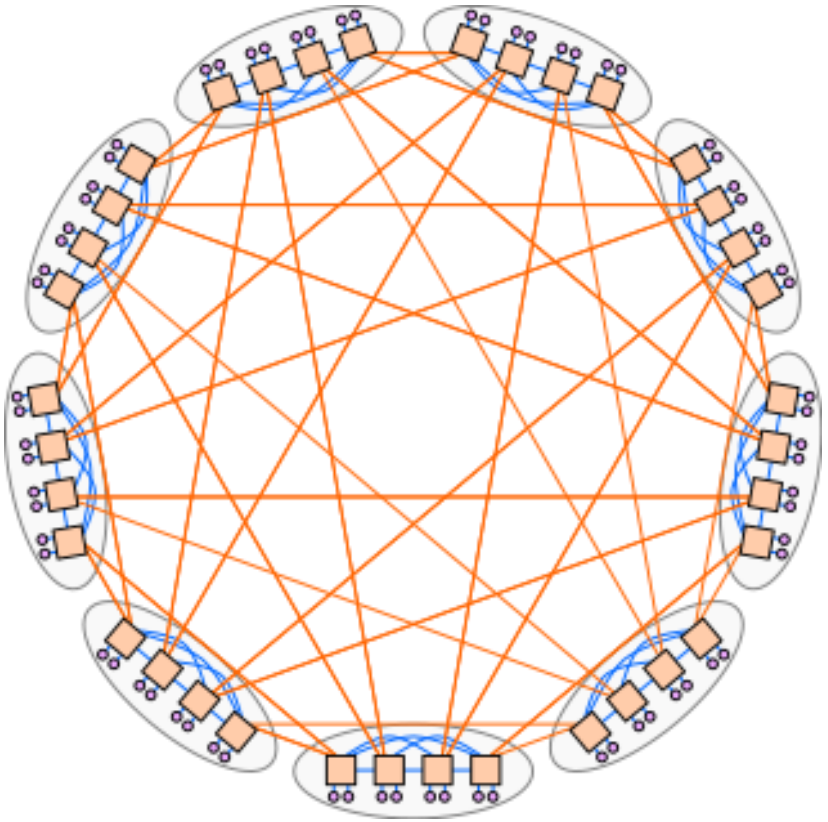


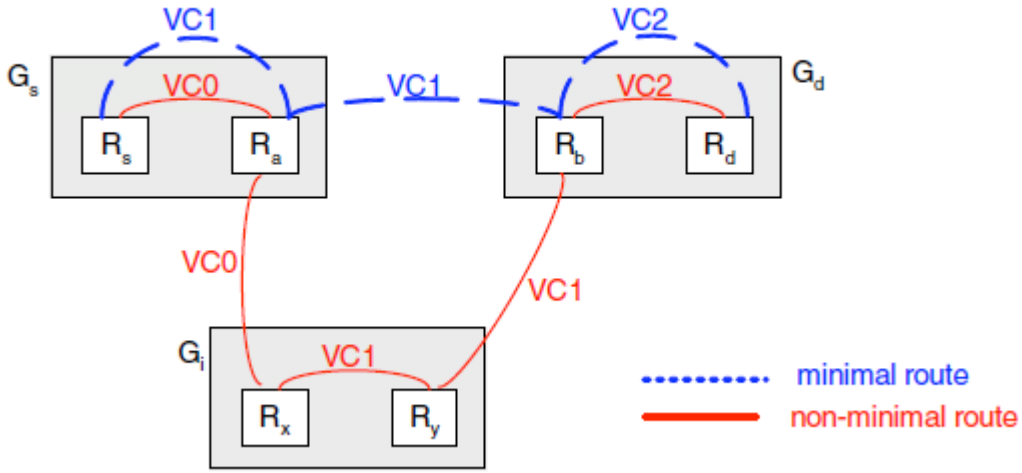
Топологии сетей (4)

Dragonfly, 2008

- Иерархическая
- Внутри группы узлов – все со всеми
- Группы между собой – все со всеми
- p – количество узлов у роутера
- a – количество роутеров в группе
- h – количество линков роутера для соединения с другими группами
- $D_{max} = 3$
- Адаптивные маршруты

p=2, a=4, h=8



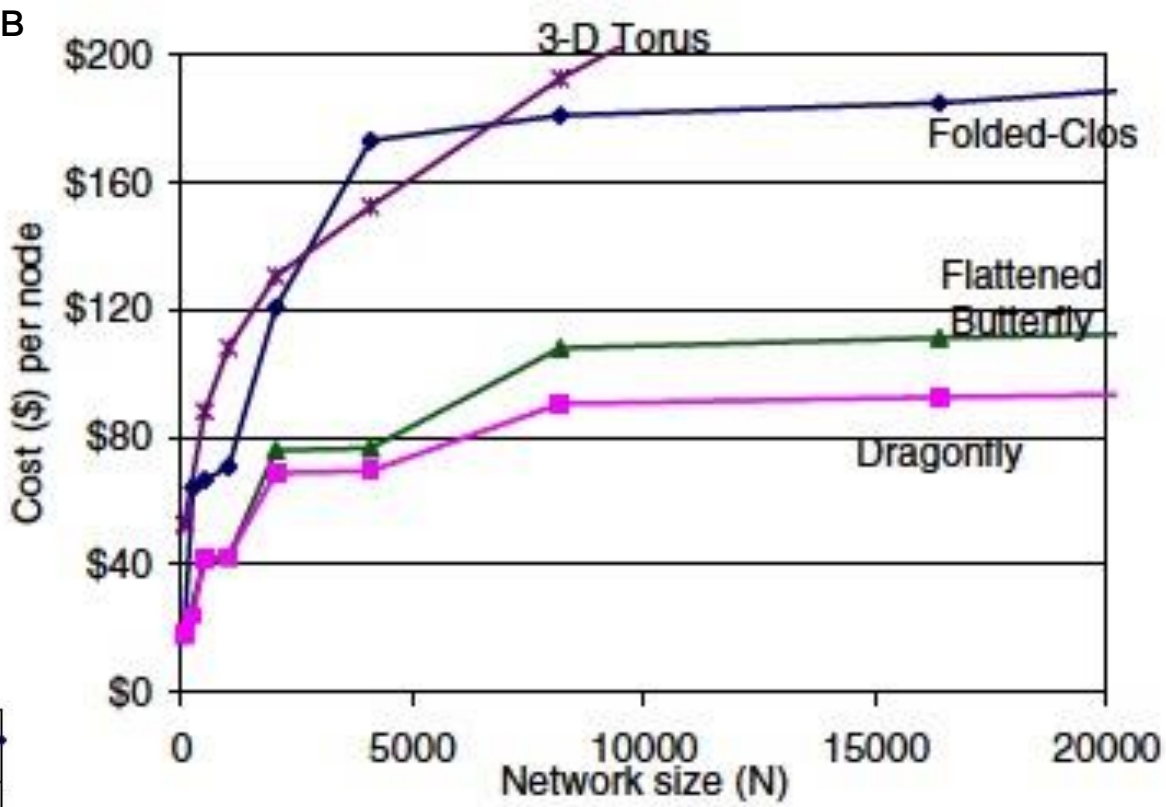


- **#10** Jun 2020 Top500: 21.2 Пфлопс

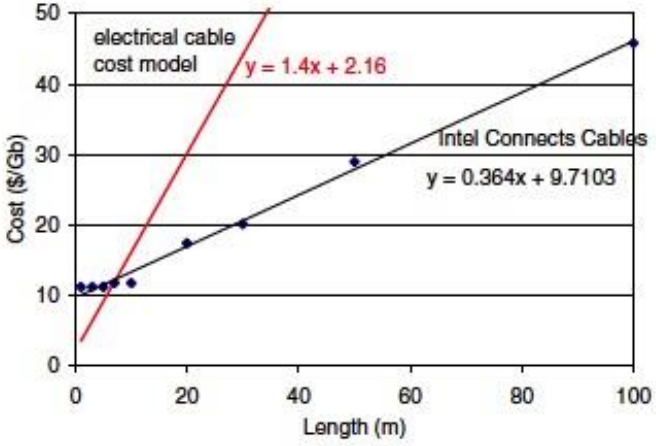
Топологии сетей (5)

Коммутаторы 64-radix
Dragonfly в группе 512 узлов

Сравнение стоимости



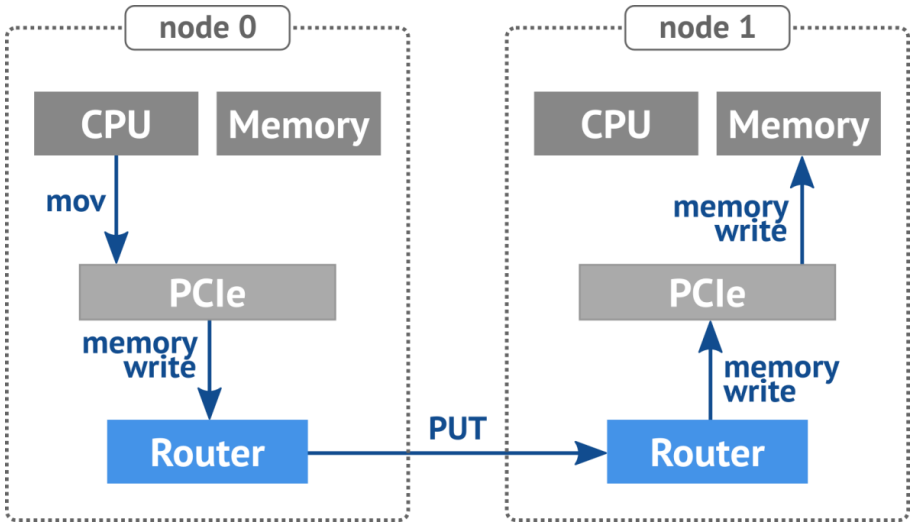
Модель



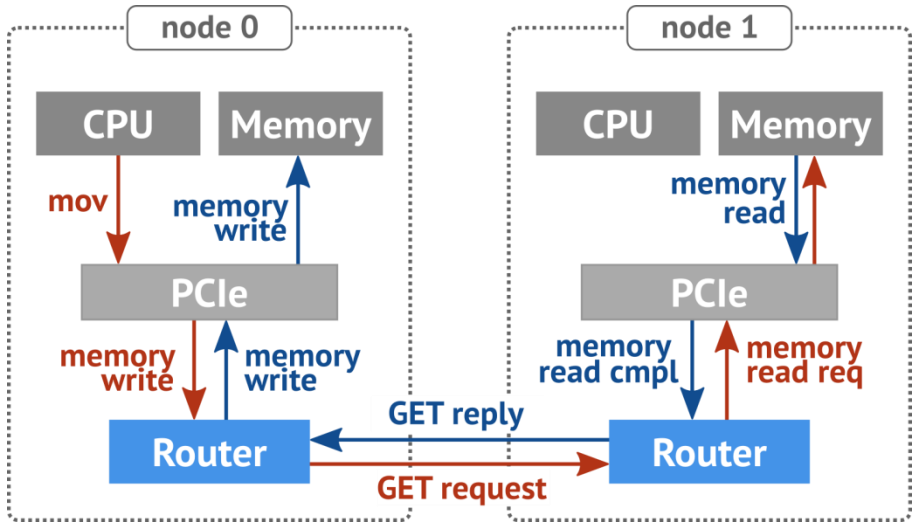
https://dl.acm.org/doi/pdf/10.1145/1394608.1382129?casa_token=M0kKxtw2n9oAAAAA:2K6ZKyilHwApHphsbAejJtCX4mjJmJkGogBOuLVg8d93G0Zr02h2uXk4Uy6l58QmGmzn3O6qJik

Механизм передачи RDMA

Удалённая запись



Удалённое чтение



Onload vs Offload*

100 Gbit/s Send-Receive

	CPU Utilization	CPU Frequency (at operation time)	CPU Instructions
Mellanox Infiniband EDR	0.8	59%	39M
Ангара	1.1		41M
Intel OmniPath	59.6%	100%	3725M

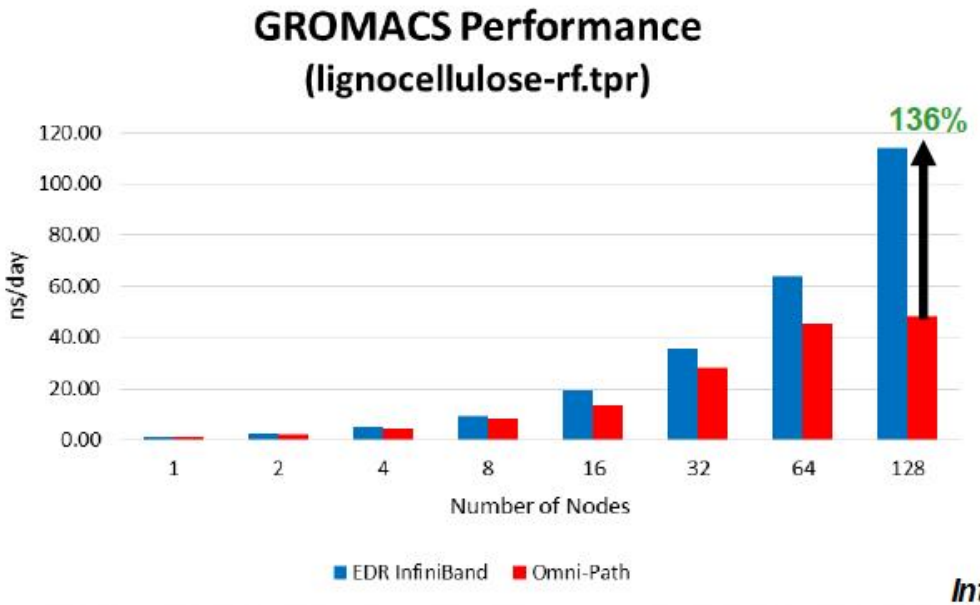
* - https://www.hpcwire.com/2016/06/18/offloading-vs-onloading-case-cpu-utilization

Intel Omni Path vs Infiniband EDR

GROMACS Performance – Interconnects

HPC ADVISORY COUNCIL

- **EDR InfiniBand enables higher scalability than Omni-Path for GROMACS**
  - InfiniBand delivers 136% better scaling versus Omni-Path for 128 nodes
  - 64 InfiniBand nodes delivers 33% higher performance compared to 128 Omni-Path nodes

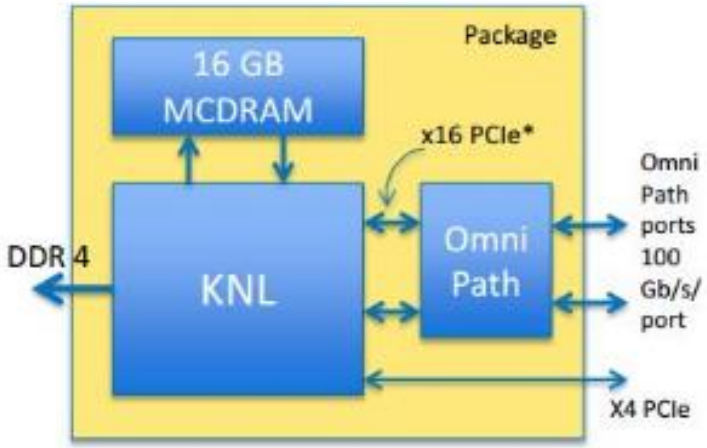


Higher is better

Intel MPI

Intel Omni Path (2015)

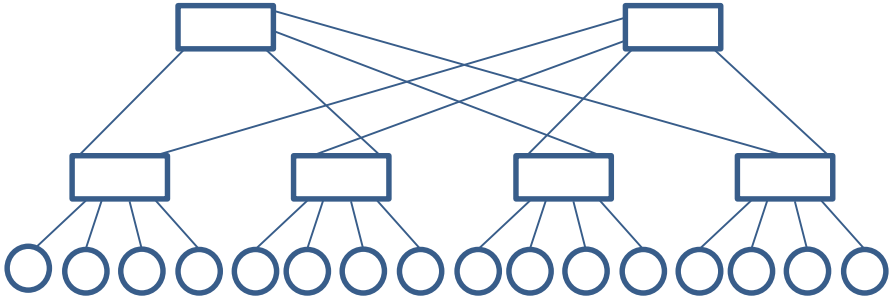
- 100 Gbit/s,
- osu_latency 0.93 us (Haswell)
- 24/48 портов switch, 192/768 director-switch











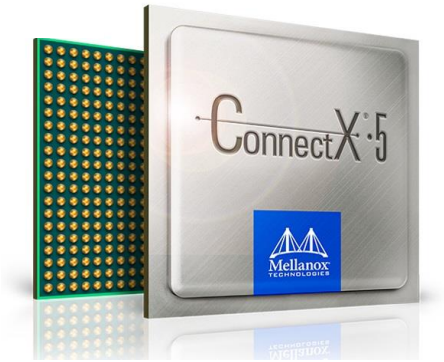


Infiniband EDR (2015)

- 100 Gbit/s, osu_latency 1 us
- 36/216/324/648 портов

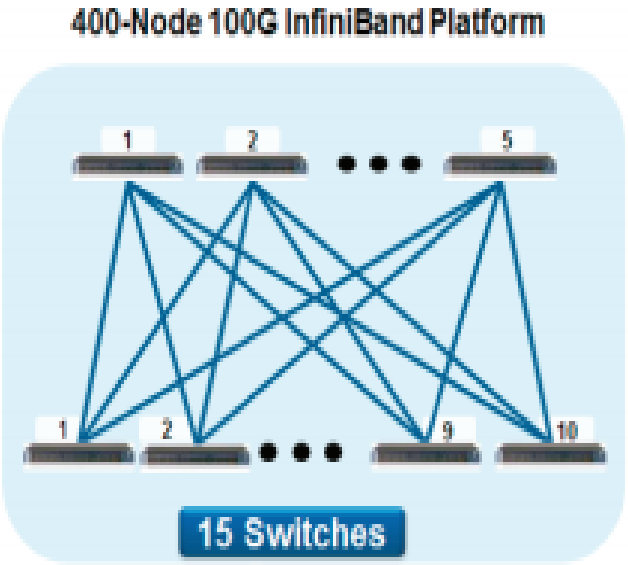


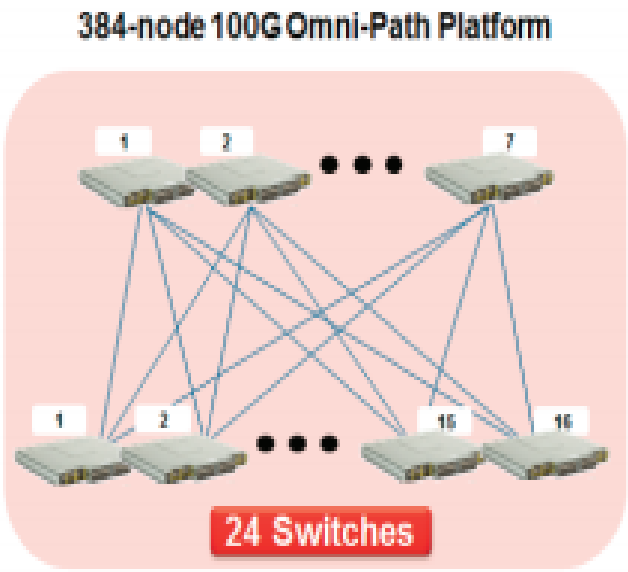




Infiniband HDR (2017)

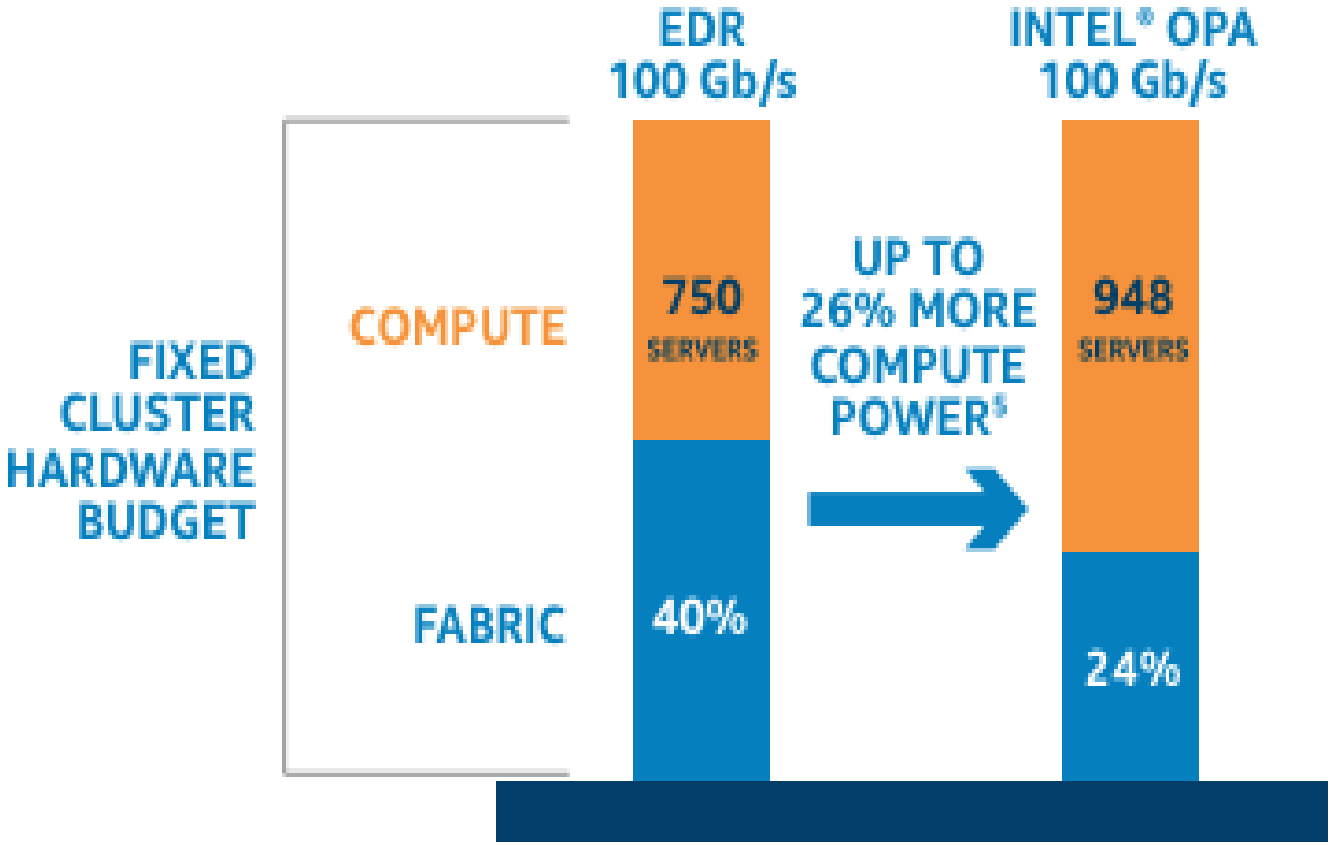
- HDR 200 Gbit/s
- 40/200/800 портов





http://www.mellanox.com/related-docs/whitepapers/WP_Introducing_200G_HDR_InfiniBand_Solutions.pdf

Intel Omni Path vs Infiniband EDR*

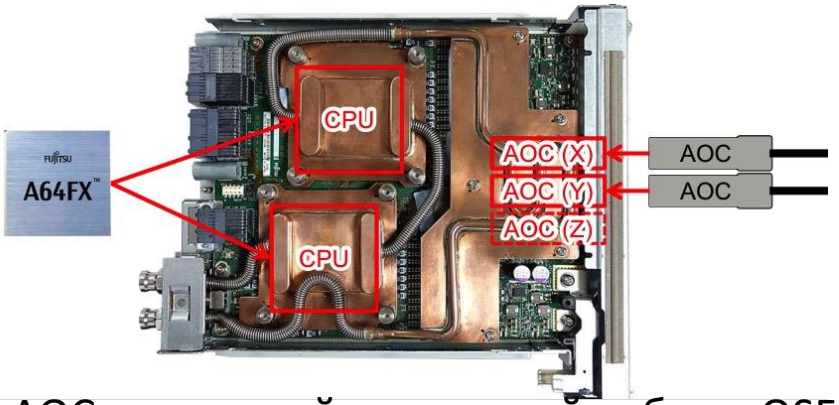


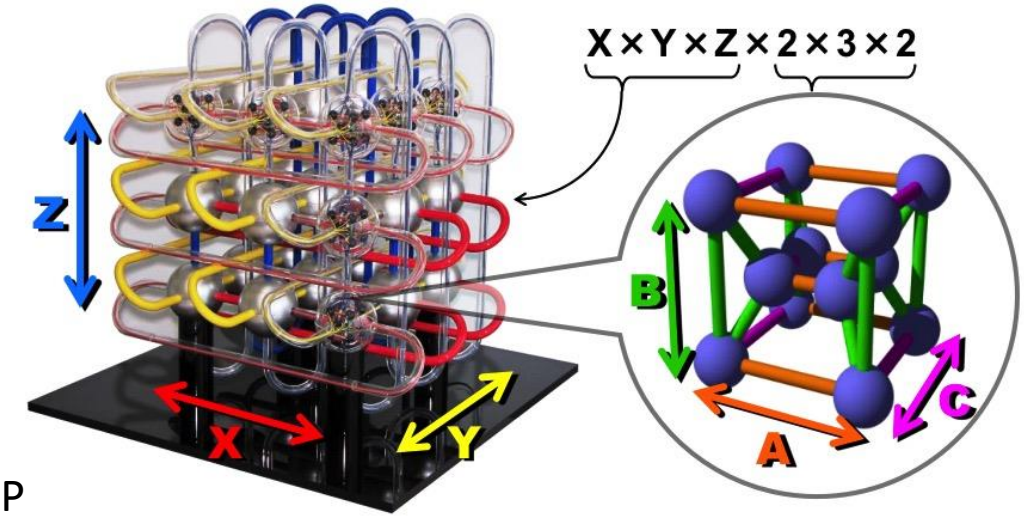
- https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/transforming-economics-hpc-fabrics-opa-brief.pdf

TofuD Interconnect (2018)

(X, Y, Z, A, B, C)

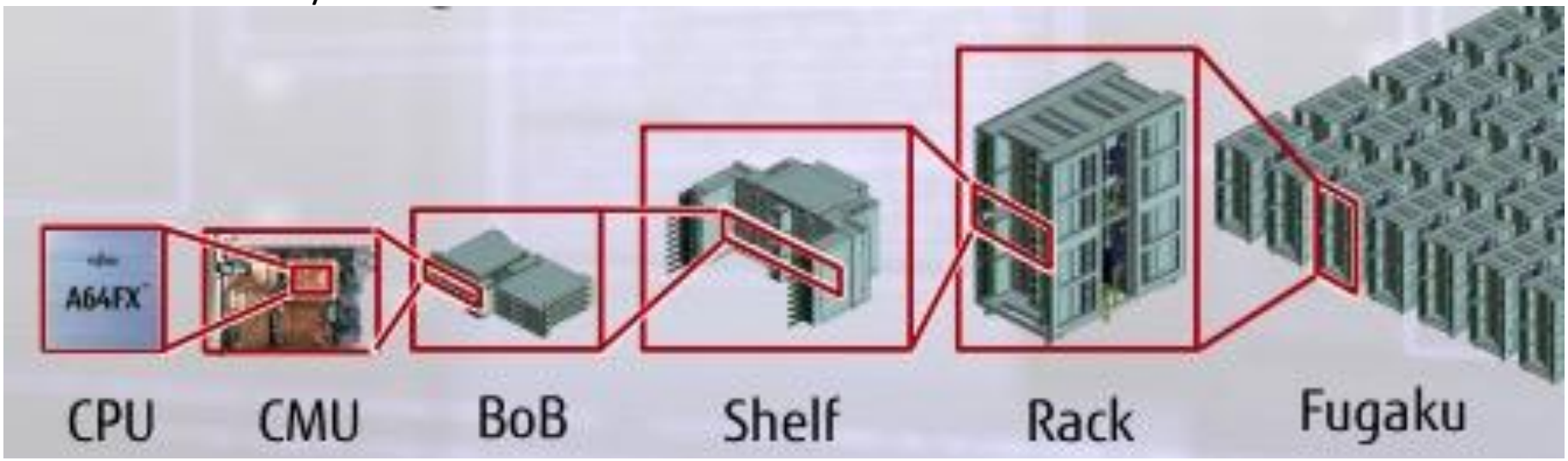
2xCPU: ось C





AOC – активный оптический кабель, QSFP

1 линк – 54.4 Гбит/с

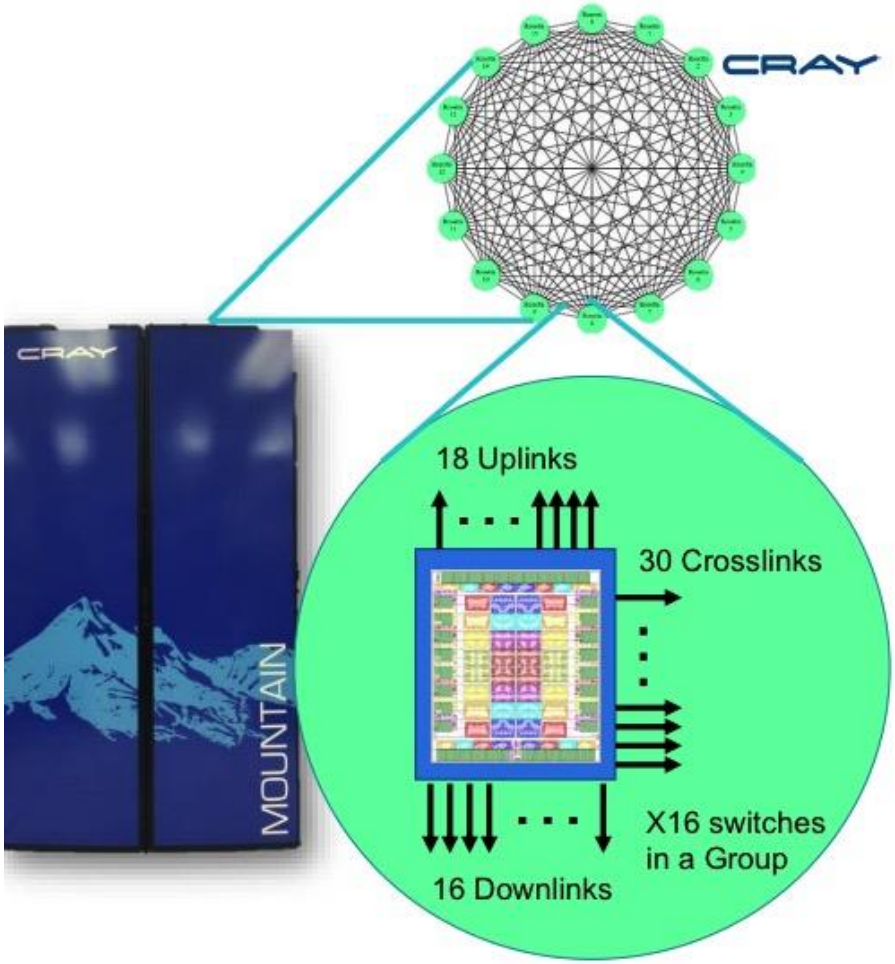


- **#1** Jun 2020 Top500: A64 FX 48C, 415 Пфлопс (80.9%, 2.6%)

https://www.fujitsu.com/jp/Images/the-tofu-interconnect-d.pdf

Cray Aries (2012), Slingshot (2019)

- Группа размещена в стойке, медные кабели
- 64-портовый коммутатор Rosetta, QSFP-DD
- Водяное охлаждение
- 1 порт – 200 Гбит/с
- Поддержка Ethernet

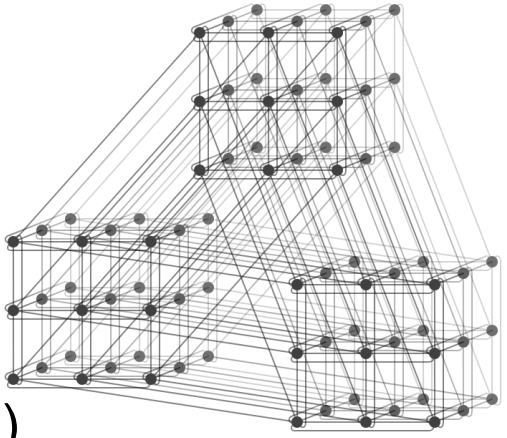




https://cug.org/proceedings/cug2019_proceedings/includes/files/inv112s1-file1.pdf

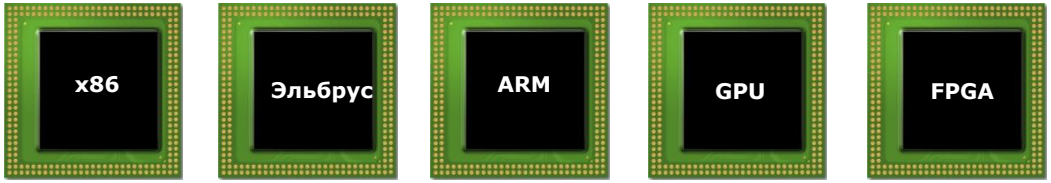
Коммуникационная сеть Ангара

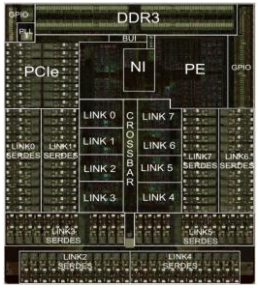
Ключевые особенности:



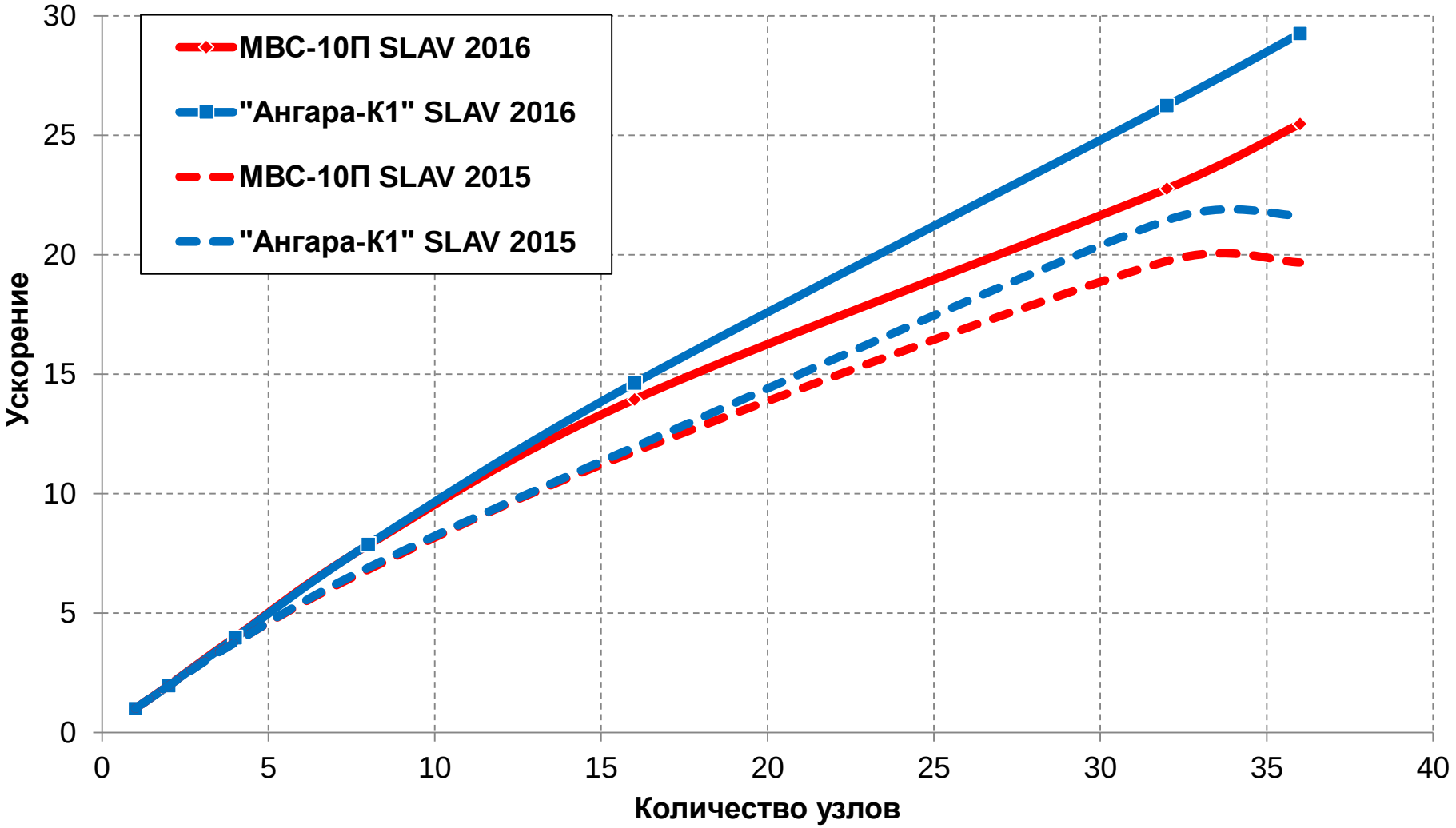
- Топология сети: 1D..4D-тор
- Адаптер на базе СБИС СБИС (65 нм, АО «НИЦЭВТ»)
- До 8 каналов связи с соседними узлами
- Прямой доступ в память удаленного узла (RDMA)
- Поддержка многоядерности
- Адаптивная передача пакетов
- Задержка на MPI ping-pong: 0,85/1,54 мкс (x86/Эльбрус-8С)
- Задержка на хоп: 130 нс
- Масштабирование: до 32K узлов
- Энергопотребление до 20 Вт
- Различные физические среды передачи данных







Модель прогноза погоды ПЛАВ



Коммуникационная сеть Ангара

1. Высокопроизводительное решение на базе FHFL адаптера и Samtec кабеля





2. Универсальное решение на базе 24-портового коммутатора, low-profile адаптера и CXP кабеля



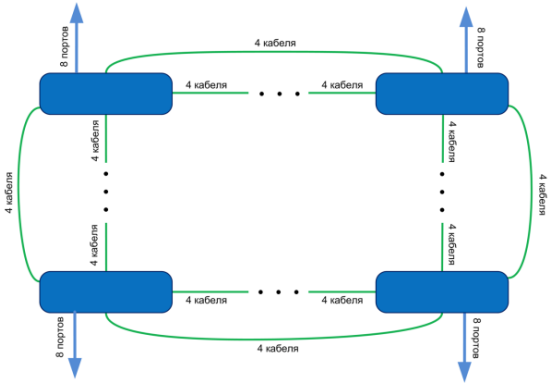




CXP-кабели

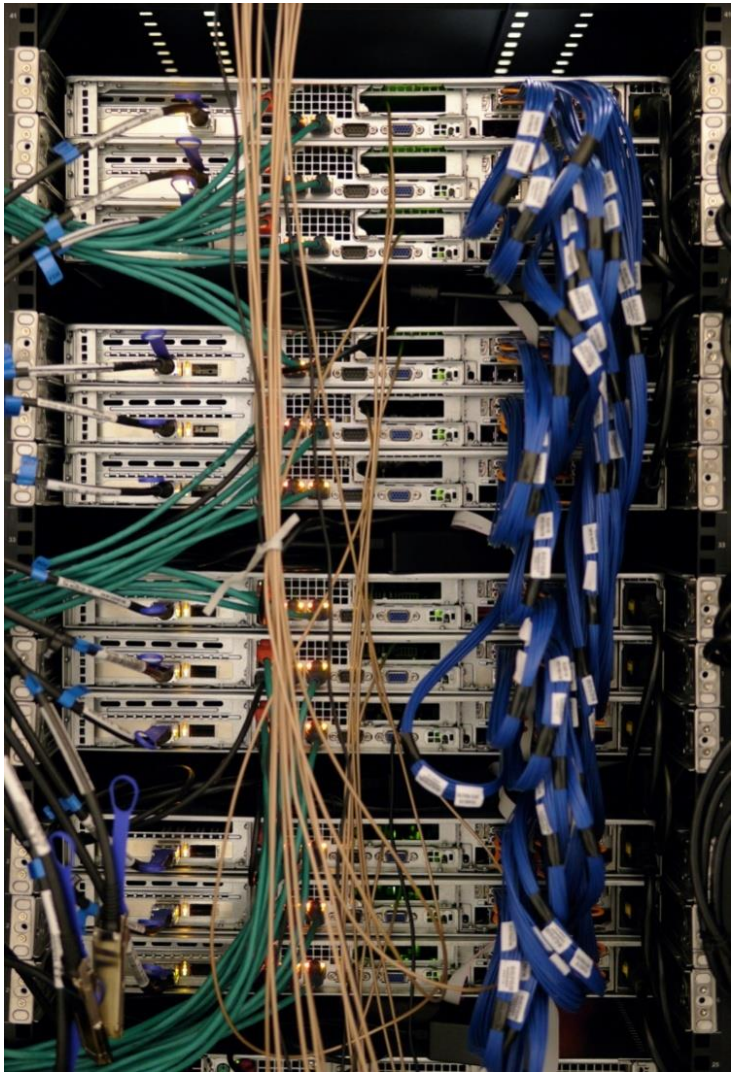
- Медные до 5 м
- Оптические до 100 м





Кластер с сетью Ангара

- **24 вычислительных узла**
  - Supermicro SuperServer 5017GR-TF
  - 2 процессора Intel Xeon E5-2630 (LGA2011, 6 ядер, 2.3 ГГц)
  - 64 ГБ
- **12 вычислительных узлов**
  - Supermicro SuperServer 5017GR-TF
  - процессор Intel Xeon E5-2660 (LGA2011, 8 ядер, 2.2 ГГц)
  - 64 ГБ
- **Сеть Ангара**
  - Адаптер EC8430, топология 3D-тор 3x3x4
  - Собственная реализация SHMEM
  - MPI: MPICH 3.0.4, MVAPICH2 1.9
- **Операционная система**
  - SLES 11 SP2, Linux 3.0.13-0.27-default
  - GCC 4.3.4 (revision 152973)



Архитектура Ангара-2

Особенности

- Топология до 6D-тор
- Пропускная способность 100/200/400 Гбит/с
- Гибкая детерминированная, зональная адаптивная маршрутизации
- Типы операций (Put, get, атомарные, с подтверждением, запись в сегменты, синхронизация, приоритеты)
- Типы пакетов – время жизни пакета, негарантированная доставка
- Технология multihost: одновременное использование 2x PCI Express 4.0 x16
- Защита данных, технология SR-IOV
- Исключительные операции, средства управления и мониторинга СБИС
- Счетчики производительности
- Отказоустойчивость на разных уровнях (линк, маршрутизация)
- Поддержка многоядерности (до 128 потоков)



Сетевое оборудование Ангара-2

- **Бескоммутаторное исполнение**
  - полноформатный адаптер PCI Express: 6 портов с разъёмами QSFP-DD по 200 Гбит/с
  - до 1024 вычислительных узлов в топологию до 3D-тор со сторонами 16x8x8
  - работа в режиме multihost до двух интерфейсов PCI Express 4.0 x16
- **Коммутаторное исполнение**
  - 40-портовый коммутатор, QSFP-DD 200 Гбит/с
  - low profile адаптер PCI Express по 2 порта с разъёмами QSFP-DD 200 Гбит/с
  - объединение до 20480 вычислительных узлов, топология объединения коммутаторов в топологию 4D-тор. Каждый коммутатор будет включать по 40 портов с разъёмами, а low profile адаптер – работа
  - в режиме multihost до двух интерфейсов PCI Express 4.0 x16



Вопросы?

alxdr.semenov@gmail.com

Суперкомпьютерное моделирование и технологии.

Лекция

Архитектура и программное обеспечение HPC-систем факультета ВМК

.

Попова Нина Николаевна

доцент кафедры СКИ

popova@cs.msu.su

23 сентября 2020 г.

Архитектура и программное обеспечение массивно-параллельной вычислительной системы

IBM Blue Gene / P

http://hpc.cs.msu.ru

Дополнительная информация на сайте

hpc@cmc | Высокопроизводительные вычисления на ВМК МГУ - Mozilla Firefox

File Edit View History Bookmarks Tools Help

hpc@cmc | Высокопроизводительные ...

hpc.cs.msu.su

Google

hpc@cmc Высокопроизводительные вычисления на ВМК МГУ

Blue Gene/P | Общие вопросы | Форум | Поддержка | Ссылки | Регистрация

- Blue Gene/P
  - Новым пользователям
  - Быстрый старт
  - Подключение
  - Файловая система
  - Разработка программ
  - Запуск заданий
  - Программное обеспечение
  - Официальная документация
  - FAQ
  - Серия Blue Gene в мире
  - Фотографии
  - Публикации
- Общие вопросы
  - Доступ по SSH
  - Выбор пароля
  - VPN-подключение
- Форум

Суперкомпьютер IBM Blue Gene/P на факультете ВМК МГУ



С 2008 года на факультете ВМК МГУ имени М. В. Ломоносова работает суперкомпьютер IBM Blue Gene/P, который является одной из первых систем данной серии среди установленных в мире. Архитектура Blue Gene была предложена компанией IBM в рамках проекта по исследованию возможностей достижения новых рубежей в супервычислениях. Более крупные машины данной серии в настоящее время занимают лидирующие позиции в списке пятисот самых мощных компьютеров мира Top500, а машина Blue Gene/P, установленная на ВМК МГУ, в редакции рейтинга от 18 ноября 2008 года оказалась на 128-м месте (в редакции от 16 ноября 2009 года — 348-е место). В списке самых высокопроизводительных компьютеров стран СНГ, опубликованном 22 сентября 2009 года, она находится на 4-й строчке.

Система IBM Blue Gene/P принадлежит к новому семейству суперкомпьютеров, обладающих высокой производительностью, масштабируемостью, возможностью обрабатывать данные большего объема, потребляя при этом значительно меньше энергии и занимая меньшую площадь по сравнению с предыдущими системами.

На факультете ВМК МГУ представлена конфигурация, состоящая из двух стоек, содержащих в общей сложности 2048

IBM Blue Gene

- **Blue Gene** –название архитектуры семейства СК
- **Проект начался в 1999 году**
  – IBM, Lawrence Livermore National Laboratory, Department of Energy
  – Первоначальное назначение: расшифровка генома, фолдинг белков
  –Три поколения: BG/L (2001), BG/P (2007), BG/Q (2012)
- **Основные свойства**
  – Масштабируемость (массивно-параллельные системы)
  – Энергоэффективность (1 процессор ~ 40 W лампочка)
- **Реализация**

-Сравнительно низкая тактовая частота процессора

-низкое энергопотребление и тепловыделение

-возможность плотной упаковки в стойку

-не требует специального кондиционирования

- Быстрая коммуникационная сеть

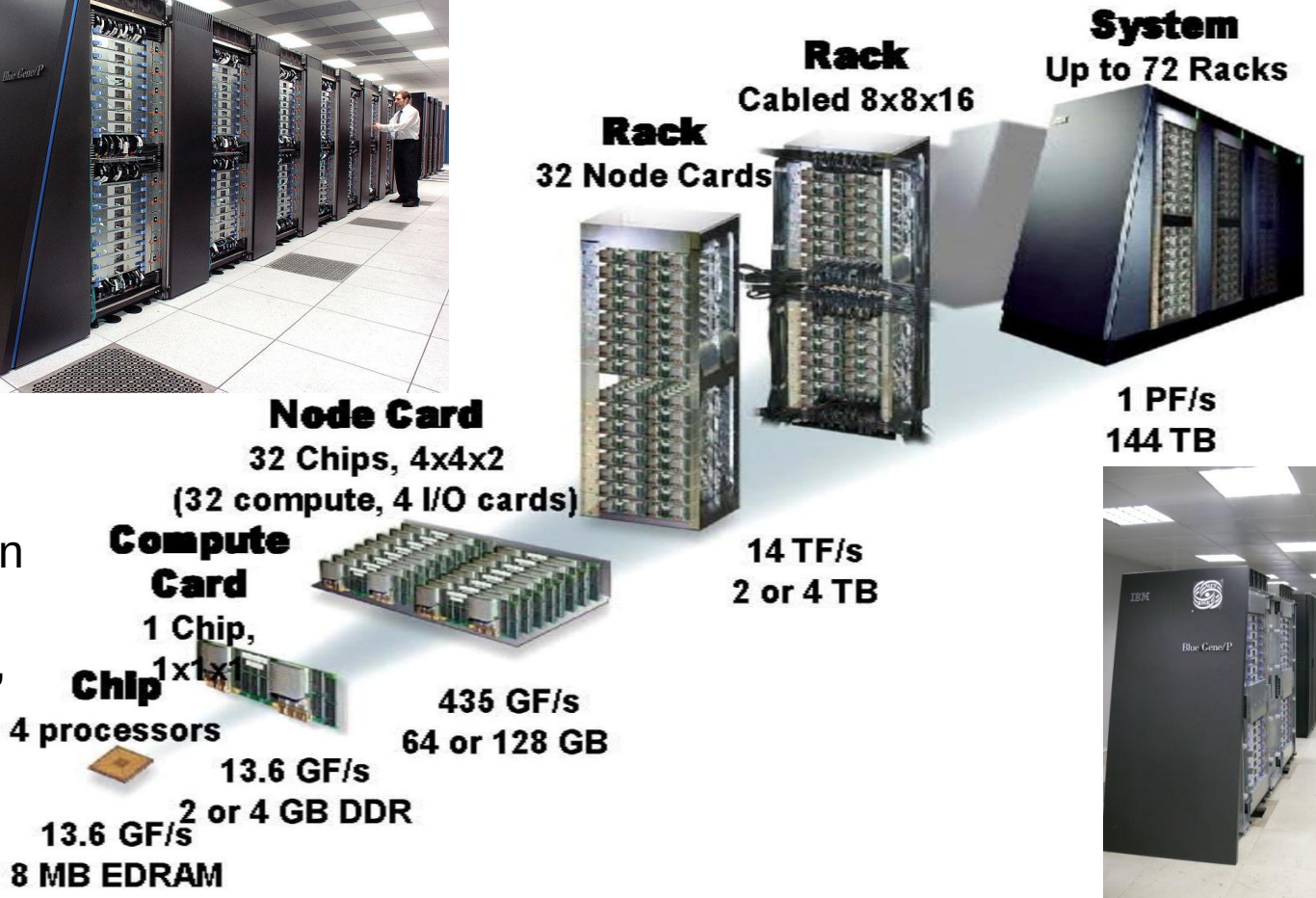
- баланс между скоростью процессора и коммуникации

– Облегченная ОС (минимизация«шума»)

Blue Gene/P Hardware



Argonne National Laboratory, in Lemont, Illinois, USA, 2007 .





Blue Gene P

1 стойка

- 1024 четырехъядерных вычислительных узлов
- производительность одного вычислительного узла – 13.6 GF/s
- производительность 1 стойки– 13.6 Tflops
- оперативная память одного узла – 2 GB
- суммарная оперативная память в стойке– 2 TB
- узлов ввода/вывода 8 – 64
- Размеры - 1.22 x 0.96 x 1.96
- занимаемая площадь 1.17 кв.м.
- энергопотребление (1 стойка) - 40 kW (max)

Конфигурация BlueGene P факультета ВМиК

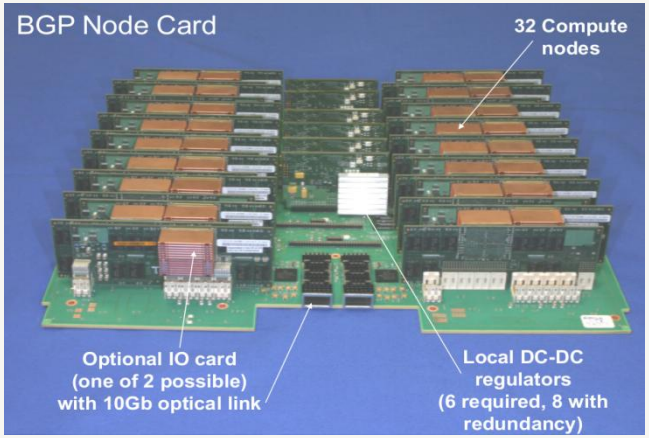
http://hpc.cs.msu.ru

- 2048 4-ех ядерных узлов
- пиковая производительность 27.2 Tflops
- Реальная производительность по тесту Linpack:
- 23.2 TFlops
- 85% от пиковой
- общий объем ОЗУ 4 ТВ



Компоненты Blue Gene P

- Основная единица – четырех ядерный вычислительный узел (процессор) , ядро – PowerPC 450 850Mhz + память (2GB)
- Карта узлов (Node card) = 32 вычислительных узла + до 2х узлов ввода-вывода



- Стойка – 32 карты узлов
- Число процессоров в стойке -1024
- Итоговое число ядер на стойку - 4096

Характеристики вычислительного узла

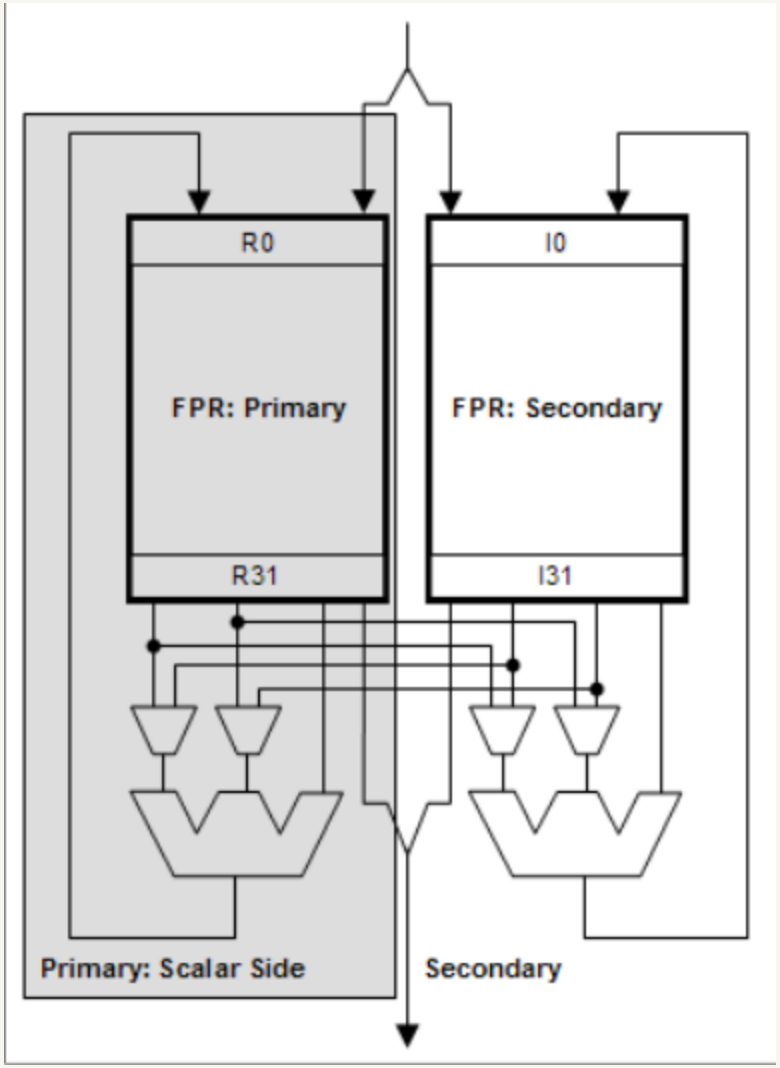
4-ех ядерный 32-битный процессор PowerPC 850 МГц:

- Двойное устройство для работы с вещественными числами с плавающей точкой (double precision)
- 2 Гб памяти
- Работает под управлением облегченной ОС:
  - Создание процессов и управление ими
  - Управление памятью
  - Отладка процессов
  - Ввод-вывод
- **Объем виртуальной памяти равен объему физической**

Память

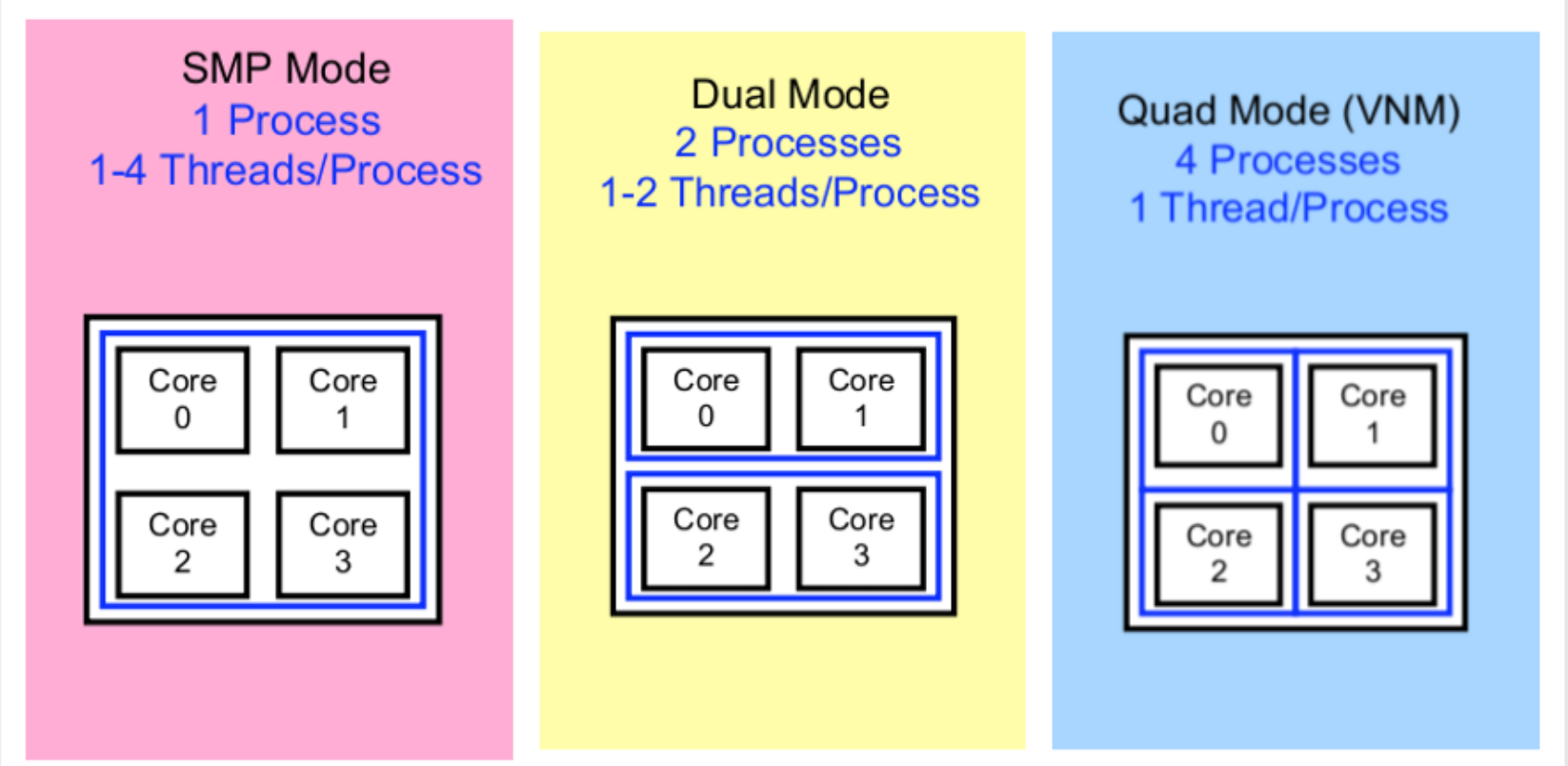
- Оперативная память – до 2GB на вычислительный узел, пропускная способность 13.6GBps
- Трёхуровневый кэш:
  - L1 – отдельный для каждого ядра, размер 32Kb
  - L2 – отдельный для каждого ядра, используется для предварительной выборки информации из кэша L1. Считывает\записывает по 16b за одно обращение.
  - L3 – разделен на две части по 4MB, доступ к ним имеют все четыре ядра, для каждого есть канал чтения и канал записи.

Double Hammer FPU



- SIMD инструкции могут выполняться одновременно на двух FPU
- Параллельные операции load/store
- Данные **должны** быть выровнены по 16-байтовой границе
  - Иначе производительность будет значительно снижена
  - Даже хуже, чем при использовании только одного FPU
- Компилятор сможет сгенерировать SIMD инструкции, только если данные в памяти расположены подряд (stride-one access)
  - Хотя при более высоких (-O4, -O5) уровнях оптимизации компилятор попытается сгенерировать SIMD инструкции и для данных, расположенных не подряд
  - **-O3 –qarch=450d –qtune=450**

Режимы выполнения процессов

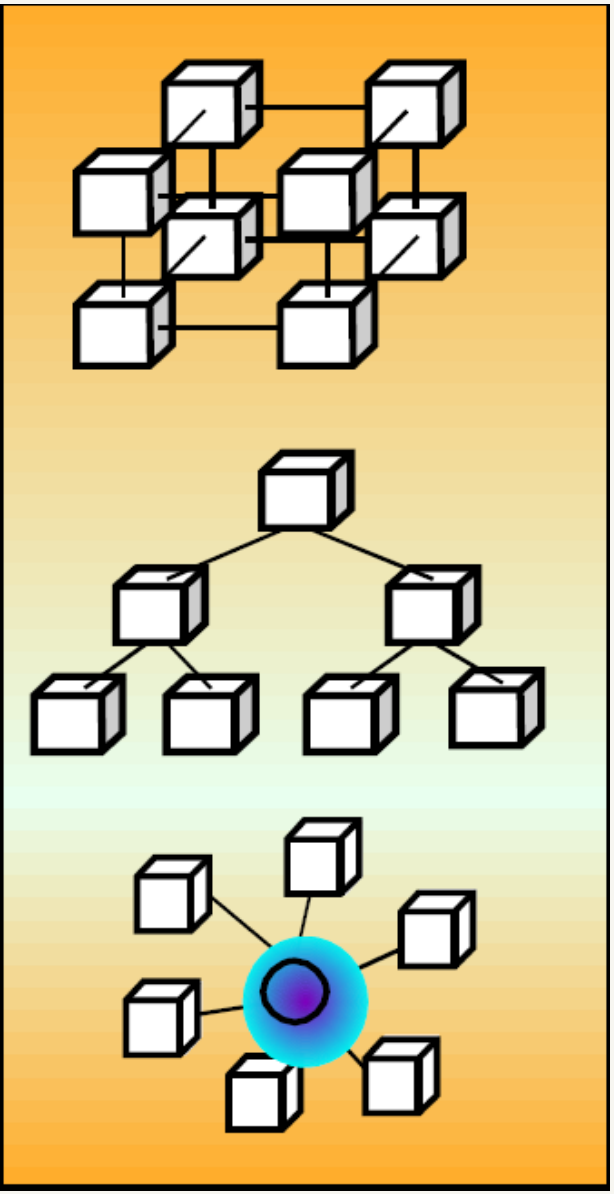


1 MPI процесс , 4 нити, 2 Гб памяти

2 MPI процесса , 2 нити, 1 Гб памяти

4 MPI процесс , 1 нить , 512 Кб памяти

Основные коммуникационные сети



3-мерный тор

•Виртуальная аппаратная маршрутизация без буферизации

•3.4 Гбит/с на всех 12 портах (5.1 ГБ/с на узле)

• Задержки между соседними узлами: 0.5 мс (аппаратная), 3 мс (MPI)

•Основная коммуникационная сеть

•Используется в том числе для многих коллективных операций

Коллективная сеть–дерево

•Для глобальной коммуникации один-ко-всем

•6.8 ГБ/с на порт

• Задержка на один проход дерева: аппаратная1.3 мс, MPI 5 мс

•Соединяет все вычислительные узлы и узлы ввода-вывода

•Используется для коллективных операций и коммуникатора MPI_COMM_WORLD

Высокоскоростная сеть для глобальных прерываний

Задержка на оповещение всех узлов: аппаратная 0.65 мс, MPI 5 мс

•Для MPI_Barrier

Компоненты Blue Gene/P

- Помимо вычислительных узлов, в состав системы также входят:
  - узлы ввода-вывода
  - узел управления системой
  - не менее одного узла front end (через них осуществляется доступ пользователей к системе)
  - сеть, связывающая компоненты системы
  - специализированная сеть для сообщения между сервисным узлом и узлами ввода-вывода

Процессоры ввода-вывода

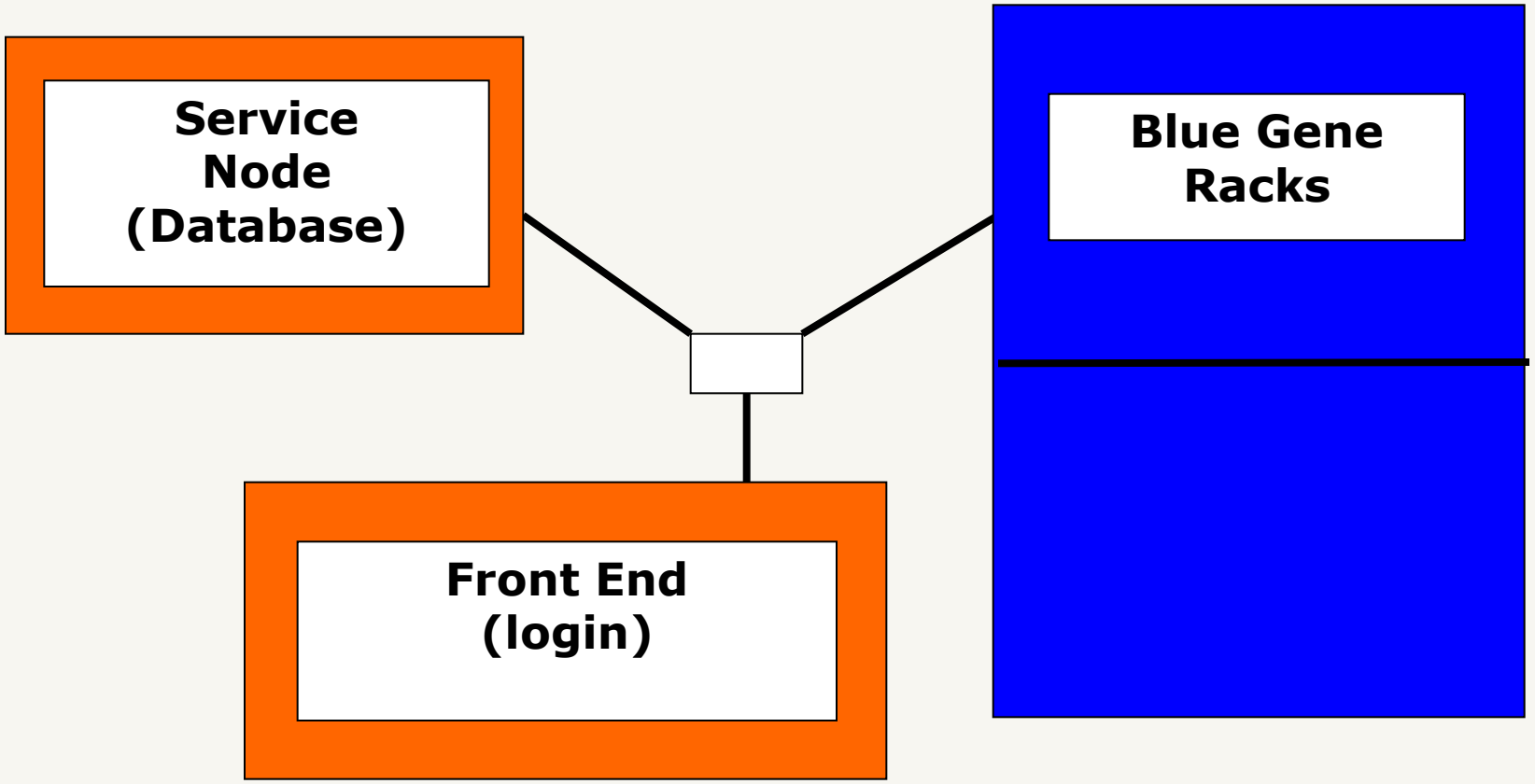
Отличия по сравнению с вычислительным узлом:

– Установлена полноценная ОС

– Отсутствует подключение к сети тору

– Имеется выход в 10-гигабитную сеть Ethernet

BlueGene/P



Состав ПО

- Linux® на узлах ввода\вывода
- MPI (MPICH2) и OpenMP (2.5)
- Стандартное семейство компиляторов IBM XL: XLC/C++, XLF
- Компиляторы GNU
- Система управления заданиями LoadLeveler
- Файловая система GPFS
- Инженерная и научная библиотека подпрограмм (ESSL), математическая библиотека (MASS)

ОС вычислительного узла BlueGene P

- Compute Node Kernel (CNK)
  - “linux-подобная” ОС
  - Нет некоторых системных вызово (fork() в основном). Ограниченная поддержка mmap(), execve()
  - Минимальное ядро – обработка сигналов, передача системных вызовов к узлам ввода-вывода, старт-завершение задач, поддержка нитей
  - Большинство приложений, которые работают под Linux, портируются на BG/P

Компиляторы Blue Gene

- IBM XL компиляторы (xlc, xlf77, xlf90)
- работают на front end узлах
  - Fortran: **mpixlf**, **mpixlf90**, **mpixlf95**
  - C: **mpixlc**
  - C++: **mpixlcxx**
- обычно являются скриптами
- GNU компиляторы существуют, но малоэффективны: **mpicc**

Ключи компилятора XL

- **–qarch=450 –qtune=450**
  - Инструкции только для одного из двух FPU
  - Используйте, если данные не выровнены по 16-байтовой границе
- **–qarch=450d –qtune=450**
  - Выравнивание!
- **–O3 (–qstrict)**
  - Минимальный уровень оптимизации под Double Hammer FPU
- **–O3 –qhot (= –qsimd)**
- **–O4 (–qnoipa)**
- **–O5**
- **–qdebug=diagnostic**
  - Информация по SIMD-инструкциям (только с qhot)
- **–qreport –qlist –qsource**
  - Много полезной информации в .lst файле
- **–qsmp=omp, – qsmp=auto**

OpenMP

- _r суффикс для имени компиляторов например, mpixlc_r

- –qsmp=omp

  указание компилятору интерпретировать OpenMP директивы

- Автоматическое распараллеливание

  -qsmp

Процессорные партиции

- Подмножества вычислительных узлов, выделяемых задаче
- Каждой задаче выделяется своя партиция
- Загрузка задачи на исполнение производится независимо от других задач
- Размер партиции определяется кратным 32
- (на текущий момент на системе ВМК - кратным 128)
- Для партиций размером кратным 512 поддерживается топология тора

Доступные размеры физических решеток:

–32: 4 x 4 x 2
–64: 4 x 4 x 4
–128: 4 x 4 x 8
–256: 8 x 4 x 8
–512: 8 x 8 x 8
–1024: 8 x 8 x 16
–1024: 8 x 16 x 8
–2048: 8 x 16 x 16

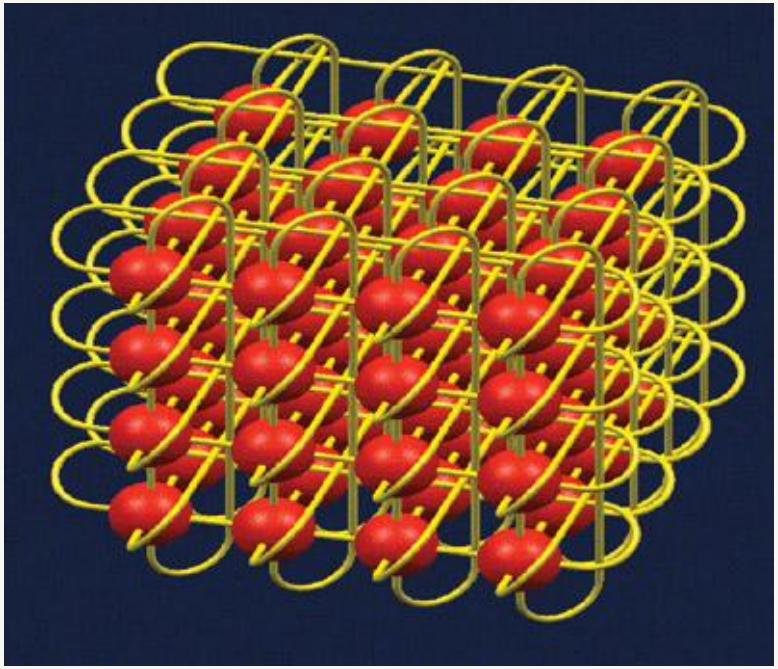
Чтобы указать конкретную размерность решетки для 1024 узлов, в командном файле при постановке программы на выполнение необходимо указать параметры
bg_rotate=false
bg_shape=8x8x16 либоbg_shape=8x16x8

Назначение процессов на процессоры (mapping)

Распределение процессов по процессорам по умолчанию:

- **режим SMP**

XYZT, где
<XYZ> - координаты процесса в торе,
T – номер ядра внутри процесса
Сначала увеличивается X – координата, затем Y и Z-координаты, после этого
T=0 в режиме SMP



- **Режим VN и DUAL:**

TXYZ

Подробная информация о размещении процессов – http://hpc.cs.msu.ru/bgp/jobs/mapping

Mapping

2 способа назначения процессов на процессоры:

- с помощью аргумента командной строки

  –mapfile TXYZ (задаем порядок TXYZ или другие перестановки X,Y,Z,T: TYXZ, TZXY и т.д.)

- указанием map- файла в командной строке

  –mapfile my.map, где my.map – имя файла.

- Синтаксис файла распределения – четыре целых числа в каждой строке задают координаты для каждого MPI-процесса (первая строка задает координаты для процесса с номером 0, вторая строка – для процесса с номером 1 и т.д.).

  0 0 0 1

  0 0 1 1

  Очень важно, чтобы этот файл задавал корректное распределение, с однозначным соответствием между номером процесса и координатами <X, Y, Z, T>.

Система управления заданиями LoadLeveler

- Оптимизирует использование имеющихся вычислительных ресурсов:
  - •Учет приоритета задач и пользователей
  - •Динамическое распределение ресурсов
- Управляет очередью заданий
- Для постановки задачи на выполнение требуется указать командный файл (скрипт с расширением .jcf)
- Скрипт для пользователей факультетской системы **mpisubmit.bg:**

mpisubmit.bg [SUBMIT_OPTIONS] EXECUTABLE [-- EXE_OPTIONS]

-n NUMBER — число запрашиваемых узлов

-m MODE — режим (SMP, DUAL, VN)

-w HH:MM:SS — время

-e ENV_VARS — переменные окружения

mpisubmit.bg -help

Основные команды LL

llq - просмотр очереди

llcancel – удаление задачи из очереди

llmap **-** просмотр занятости узлов

```
popova@fen1:~/SCMT_2020> mpisubmit.bg -m smp -n 400  -e "OMP_NUM_THREADS=4" ./ca
nnon_omp
llsubmit: Stdin job command file written to "/tmp/loadlx_stdin.25519.ogJFnA".
submit filter in use[ popova fen1.11182684 ]
llsubmit: Processed command file through Submit Filter: "/etc/LoadL/cmc_submit_f
ilter".
llsubmit: The job "fen1.bg.cmc.msu.ru.11182684" has been submitted.
popova@fen1:~/SCMT_2020> ддй
-bash: ддй: command not found
popova@fen1:~/SCMT_2020> llq
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
fen1.11182684.0          popova      9/29 16:22 R  50  n512_m30     fen1
fen1.11173777.0          edu-cmc-sk 12/19 07:39 I  50  n2048_m03
fen1.11179863.0          edu-cmc-sk  1/16 23:32 I  50  n2048_m03

3 job step(s) in queue, 2 waiting, 0 pending, 1 running, 0 held, 0 preempted
```

Компиляция программ

- **MPI :**

C: mpixlc -O3 hw.c -o hw

C++: mpixlcxx -O3 hw.cxx -o hw

Fortran: mpixlf90 -O3 hw.F90 -o hw

- **MPI + OpenMP:**

C: mpixlc_r -qsmp=omp -O3 hw.c -o hw

C++: mpixlcxx_r -qsmp=omp -O3 hw.cxx -o hw

Fortran: mpixlf90_r -qsmp=omp -O3 hw.F90 -o hw

Постановка на счет

MPI + OpenMP, 32 nodes, 15 mins

- SMP (4 потока на MPI процесс):

  mpisubmit.bg -n 32 -m SMP -w 00:15:00 -e "OMP_NUM_THREADS=4" hello

- DUAL (2 потока в MPI процессах):

  mpisubmit.bg -n 32 -m DUAL -w 00:15:00 -e "OMP_NUM_THREADS=2" hello

- VN (1 поток в MPI процессах):

  mpisubmit.bg -n 32 -m VN -w 00:15:00 -e "OMP_NUM_THREADS=1" hello

Рекомендации по отладке программ

- **На Blue Gene нет виртуальной памяти, поэтому необходимо тщательно следить за утечками памяти**

  – Application killed with signal 6

    - Недостаточно памяти

- **Ключ компилятора -g**
- **addr2line**

  –Утилита для анализа core файлов

- **Application killed with signal 7 (memory alignment error)**

  –BG_MAXALIGNEXP=1

  процесс умирает при первой ошибке выравнивания (полезно при оптимизации)

- –BG_MAXALIGNEXP=-1

  игнорировать ошибки выравнивания

Основной шаблон протокола работы пользователя (1)

1. Выход на BGP:

%ssh <опции> <логин>@ bluegene.hpc.cs.msu.ru

Например:

%ssh –i ~/.ssh/bgp.ppk user1@bluegene.hpc.cs.msu.ru

2. Копирование файлов с локального компьютера на Blue Gene/P:

(локальная машина)

%scp –i ~/.ssh/bgp.ppk hw.cpp user1@bluegene.hpc.cs.msu.ru:~/hw.cpp

Основной шаблон протокола работы пользователя (2)

3. Компиляция MPI-программы на языке C или C++ : (BGP, front-end)

```
%mpixlc hw.c -o hw
%mpixlcxx  hw.cpp -o hw
%mpixlf90 hw.f90  -o hw
```

4. Компиляция гибридной MPI-OpenMP программы:

```
%mpixlc_r -qsmp=omp hw.c -o hw
% mpixlcxx_r -qsmp=omp hw.cpp -o hw
```

Основной шаблон протокола работы пользователя (3)

5. Постановка MPI-программы в очередь задач с лимитом выполнения 15 минут на 128 узлов в режиме **VN** с параметром командной строки :

%mpisubmit.bg -w 00:15:00 -m VN -n 128 prog – 0.1 200

6. Постановка MPI+OpenMP программы **prog** в очередь задач с лимитом выполнения 15 минут на 128 узлов в режиме **SMP** с 4 нитями на каждом узле и с параметром командной строки **parameter**:

%mpisubmit.bg -w 00:15:00 -m SMP -n 128
-e «OMP_NUM_THREADS=4» prog -- parameter

Основной шаблон протокола работы пользователя (4)

7. Постановка MPI+OpenMP программы **prog** в очередь задач с лимитом выполнения 15 минут на 128 узлов в режиме **SMP** с 4 нитями на каждом узле и с параметром командной строки:

%mpisubmit.bg -w 00:30:00 -m SMP -n 1024 –mapfile XYZT -e «OMP_NUM_THREADS=4» example -- 100

Пример. Матричное умножение.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
 #define N 1024
 int main(int argc,char **argv) {
   int i,j,k;
   double A[N][N],B[N][N],C[N][N];
   unsigned int iseed = 0;
   struct timeval start;
   struct timeval stop;
   double time;   const  double  micro = 1.0e-06;
```

Пример. Матричное умножение.

```
srand(iseed);
   for(i=0;i<N;i++)
     for(j=0;j<N;j++) {
       A[i][j]=5-( 10.0 * rand() / ( RAND_MAX + 1.0 ) );
     B[i][j]=5-( 10.0 * rand() / ( RAND_MAX + 1.0 ) );
     C[i][j]=0.0;     }
   gettimeofday(&start,NULL);
   for (i=0; i<N; i++)
     for (k=0; k<N; k++)
       for(j=0; j<N; j++)
         C[i][j]+=A[i][k]*B[k][j];
```

Пример. Матричное умножение.

```
  gettimeofday(&stop,NULL);

  time = (stop.tv_sec - start.tv_sec) + micro*(stop.tv_usec -
   start.tv_usec);
    printf("Time=%f\n",time);
    return 0;
}
```

Анализ последовательной программы

- Компиляция

  bgxlc_r -O2 -qtune=450 -qarch=450 mxm.c -o mxm

- Запуск последовательной программы на одном узле

  mpisubmit.bg –n 1 mxm

- Результат:

16.799s (~128 MFlop/s, 3.7% от пиковой производительности)

Последовательная программа. Интенсивная оптимизация.

- *bgxlc_r -O3 -qstrict -qtune=450 -qarch=450 mxm.c -o mxm*

  В этом варианте компилятор выполняет некоторые интенсивные оптимизации памяти и времени компиляции в дополнение к тем, которые выполняются с -O2.

  Опция -qstrict предоставляется в -O3, чтобы отключить эту агрессивную оптимизацию.

- Результат

  6.26s

Использование Double Hummer инструкций

- *bgxlc_r -O3 -qstrict -qtune=450 -qarch=450d mxm.c -o mxm*

  С такими настройками компилятор попытается сгенерировать SIMD-код для двойного FPU
- 3.496s (~620 MFlop/s, 18% от пиковой производительности)

Высокоуровневые преобразования

- *bgxlc_r -O3 -qhot -qtune=450 -qarch=450d -g -qreport -qlist -qsource mxm.c -o mxm*
- В этой настройке компилятор включает такие преобразования, как обмен циклами, разбиение цикла и т. д. Кроме того, для генерации преобразований кода и списков оптимизации использованы параметры компилятора –qreport –qlist и –qsource.
- Файл листинга mxm.lst содержит информацию о преобразованном коде. Компилятор распознал схему Matrix Multiply и заменил соответствующие строки кода реализацией DGEMM компилятора XL. Это не всегда желаемое решение, так как результирующая производительность может быть ниже.
- 5.7s

Использование ESSL DGEMM

- Заменим ijk-цикл на обращение к функции библиотеки ESSL

  dgemm("T","T",s,s,s,one,&A,s,&B,s,zero,&C,s);

- *bgxlc_r -O3 -qhot -qtune=450 -qarch=450d -g -I/opt/ibmmath/essl/4.4/include -c mxm_dgemm.c*
- *bgxlc_r -O3 -qhot -qtune=450 -qarch=450d mxm_dgemm.o -g -L/opt/ibmmath/essl/4.4/lib -lesslbg -lesslsmpbg -L/opt/ibmcmp/xlf/bg/11.1/bglib -lxlf90_r -lxlopt -lxl -lxlfmath -lxlsmp -o mxm_dgemm*
- Результат: ***0.84s***

Матричное умножение. Параллельный алгоритм Кэннона.

- *mpixlc_r -O3 -qstrict -qtune=450 -qarch=450d -g cannon.c -o cannon*
- *mpisubmit.bg –n 1024 ./cannon*

Результаты:

- Размер матриц: 16384x16384
- Количество узлов: 1024
- Время = 23.2 сек.

Матричное умножение. Кэннона+dgemm

- *mpixlc_r -O3 -qstrict -qtune=450 -qarch=450d -g -I/opt/ibmmath/essl/4.4/include -c cannon_dgemm.c*
- *mpixlc_r -O3 -qstrict -qtune=450 -qarch=450d cannon_dgemm.o -g -L/opt/ibmmath/essl/4.4/lib -lesslbg -L/opt/ibmcmp/xlf/bg/11.1/bglib -lxlf90_r -lxlopt -lxl -lxlfmath -lxlsmp -o cannon_dgemm*
- *mpisubmit.bg –n 1024 ./cannon*

Результаты:

- Размер матриц: 16384x16384
- Количество узлов: 1024
- Время = **1.79s** (было 23.2 сек.)

Матричное умножение. Алгоритм Кэннона, MPI+OpenMP

- *mpixlc_r -O3 –qsmp=omp -qstrict -qtune=450 -qarch=450d -g cannon_omp.c -o cannon_omp*
- *mpisubmit.bg –n 256 -m SMP -e «OMP_NUM_THREADS=2» ./cannon_omp*

Результаты:

- Размер матриц: 16384x16384
- Количество узлов: 256
- Время = **180.8 s**

mpisubmit.bg –n 256 -m SMP -e «OMP_NUM_THREADS=4» ./cannon_omp

- Время = **90 s**

mpisubmit.bg –n 256 -m vn -e «OMP_NUM_THREADS=1» ./cannon_omp

Время = 89 s

Ссылки

- http://hpc.cs.msu.ru/bgp
- IBM System Blue Gene Solution: Blue Gene/P Application Development, SG24-7287-00
  redbooks.ibm.com/abstracts/sg247287.html



Архитектура и программное обеспечение высокопроизводительной вычислительной системы

Polus

http://hpc.cs.msu.ru

Общая схема организации HPC

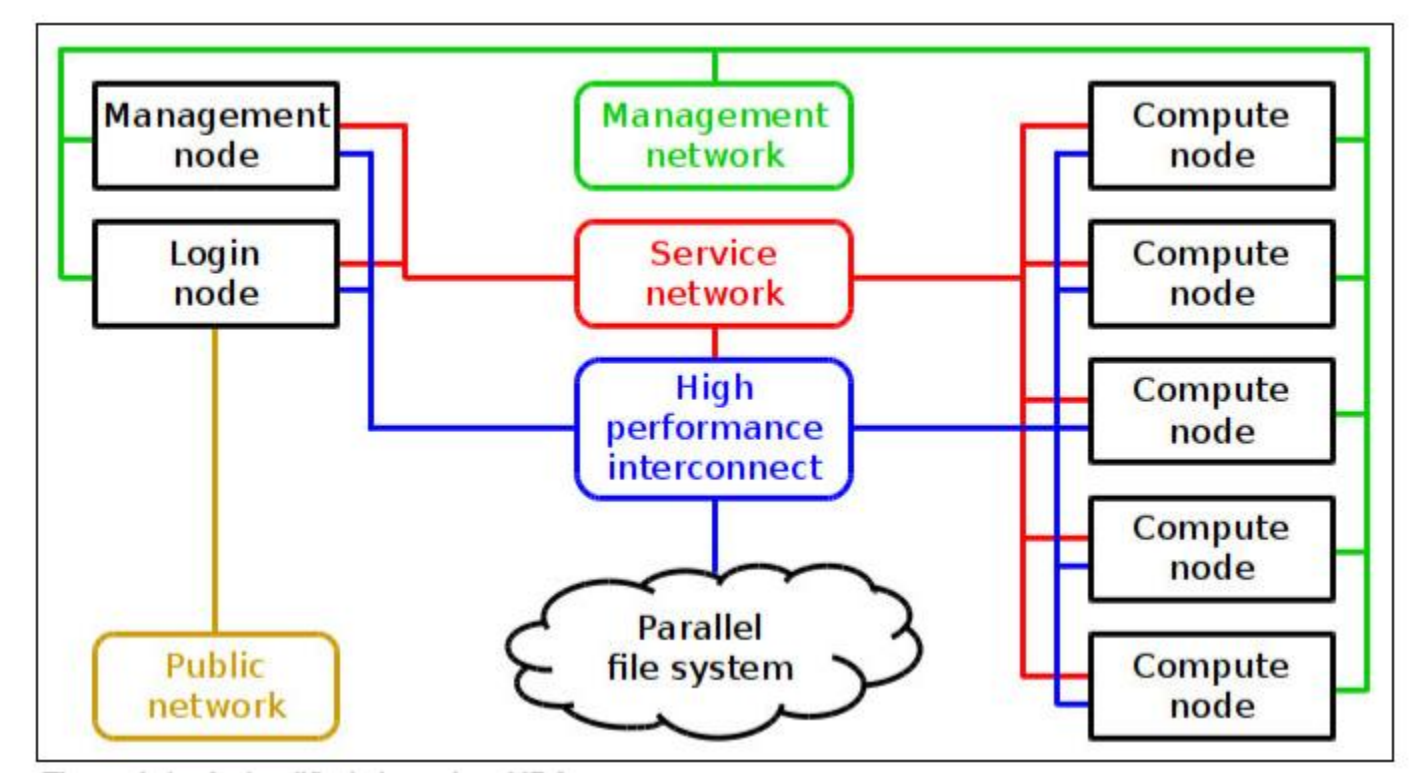
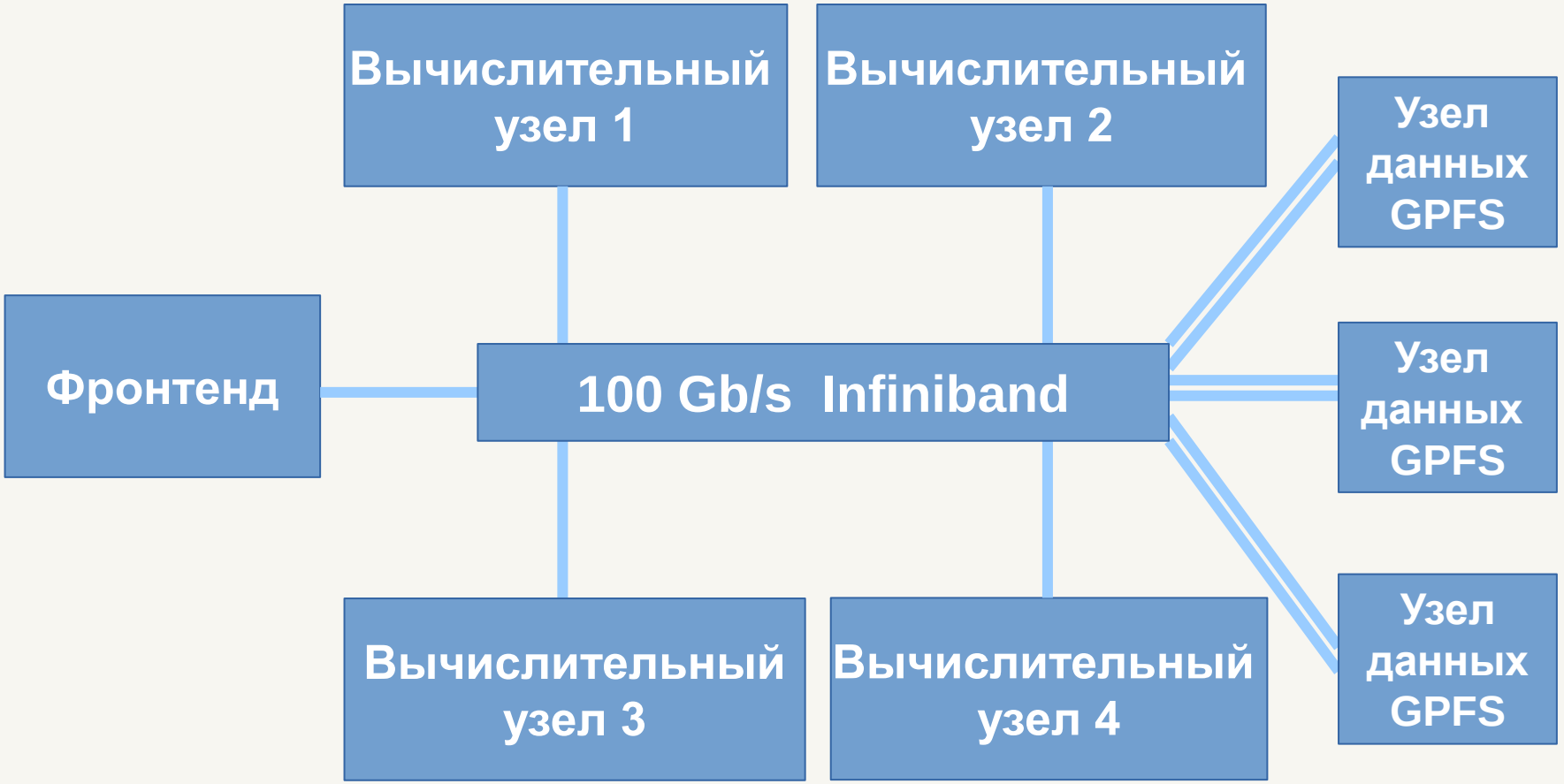


Схема ПВС IBM Polus



Кластер Polus

Пиковая производительность	55.84 TFlop/s
Производительность (Linpack)	40.39 TFlop/s (72% от пика)
Вычислительных узлов	5
На каждом узле:	
Процессоры IBM Power 8	2
NVIDIA Tesla P100	2
Число процессорных ядер	20
Число потоков на ядро	8
Оперативная память	256 Гбайт (1024 Гбайт узел 5)
Коммуникационная сеть	Infiniband / 100 Gb
Система хранения данных	GPFS
Операционная система	Linux Red Hat 7.5

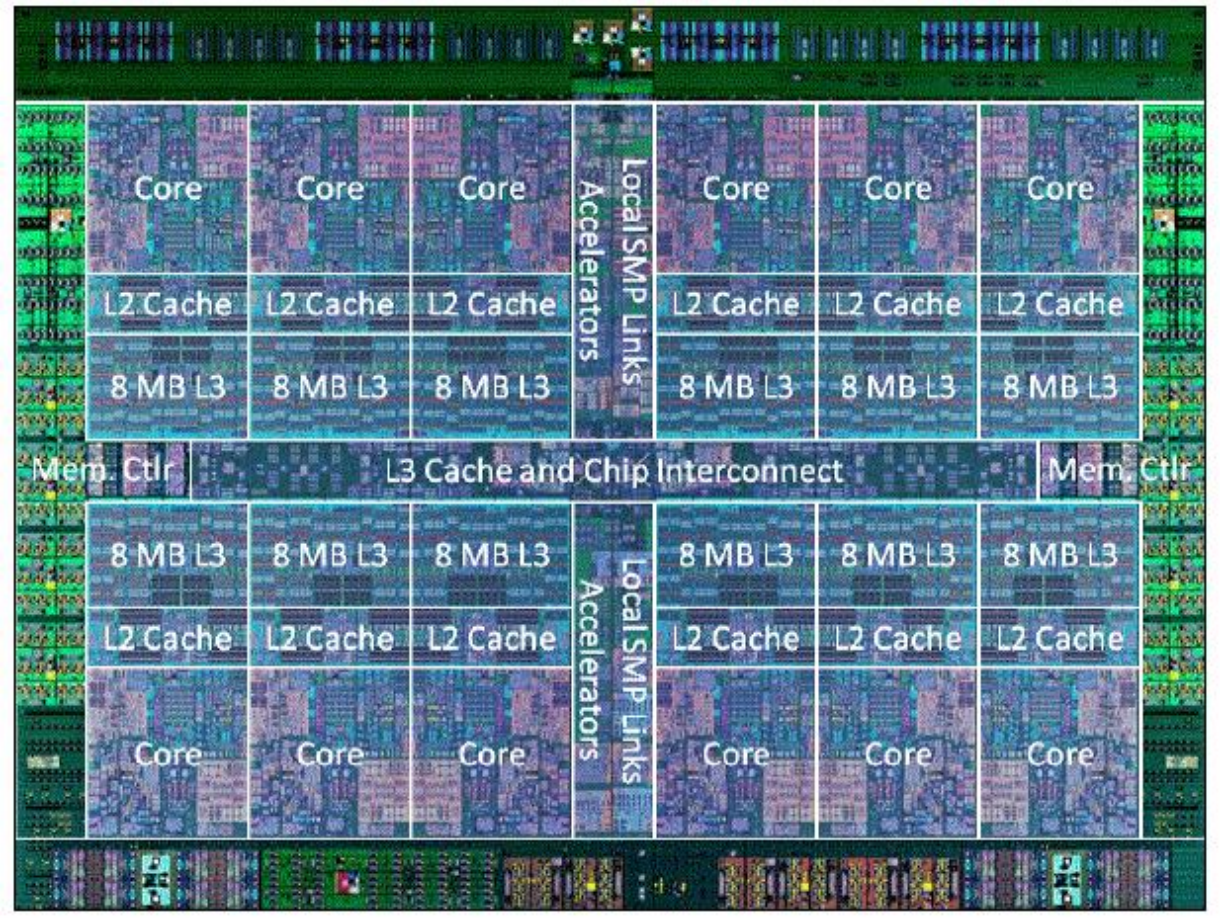
Характеристика узлов Polus

- Каждый узел имеет 2 сокета с 10 ядрами на сокет = 20 ядер на узел.
- Узлы используются в SMT8 режиме => 8 аппаратных потоков на ядро => 160 потоков на узел.
- Linux трактует каждый поток как логический CPU.
- Размещение и привязка потоков важны для хорошей производительности.
- Существенное влияние NUMA природы на доступ к памяти: лучше всего память, являющаяся локальной к сокету.

Организация кеш-памяти в Power8

- 64 KB L1 кеш данных на ядро, 3-5 тактов время доступа
- 32 KB L1 кеш инструкций на ядро.
- 512 KB L2 кеш на ядро, ~12 тактов время доступа.
- 8 MB L3 кеш на ядро (NUMA архитектура), ~27 тактов.

Схема процессора Power8

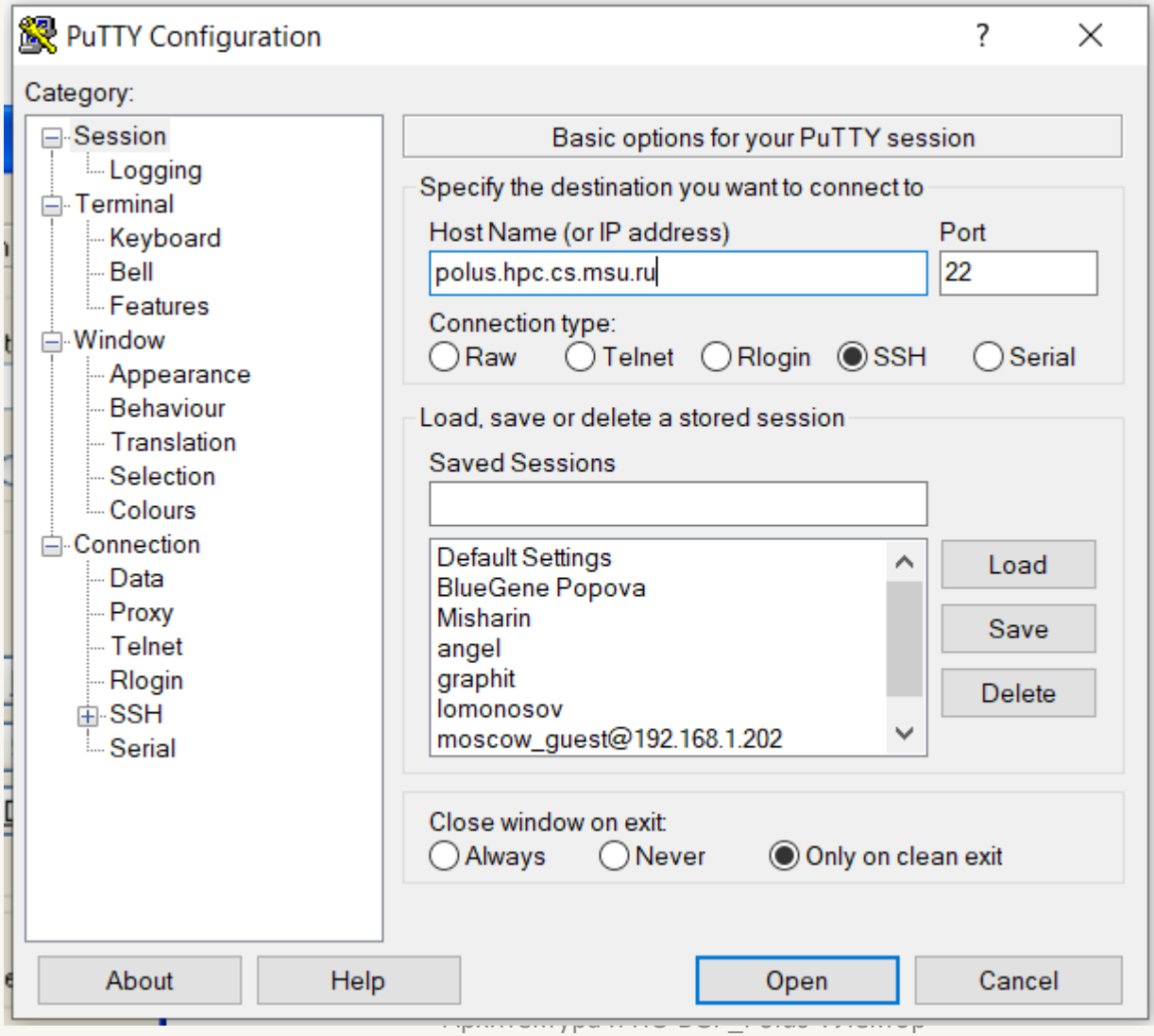


Курс "Суперкомпьютерное моделирование и технологии". Лекция "Архитектура и ПО BGP_Polus". Лектор Попова Н.Н.

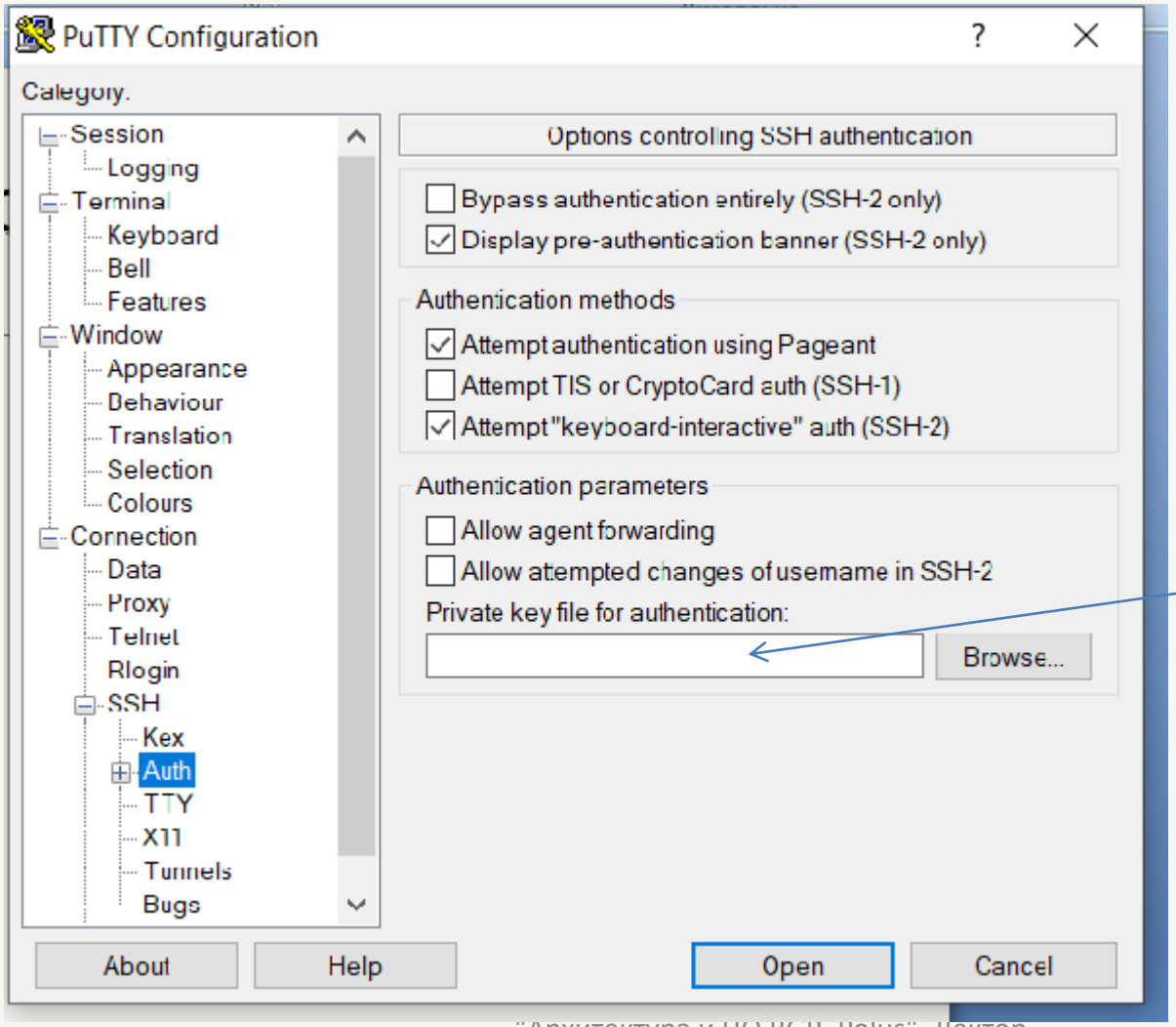
Организация работы пользователя на ВС Polus

Доступ на Polus. Putty

polus.hpc.cs.msu.ru



Доступ на Polus



Здесь указываете файл с приватным ключом

Инфраструктура modules

$ *module list* (показывает, какие модули загружены)
$ *module avail* (доступные модули)
$ module add module_name (добавляет нужный модуль)
$ *module unload module_name* (удаляет модуль)

Для создания и запуска MPI программ надо загрузить модуль SpectrumMPI.
% module load SpectrumMPI

Можно использовать OpenMPI
% module load OpenMPI

Компиляторы

- **Компиляторы** : clang, gnu, ibm xl … nvcc

  gcc, g++, gfortran : по умолчанию 4.8.5;
- Некоторые полезные опции

  -mcpu=power8 , -Ofast, -fopenmp
- Компиляторы GNU вызываются по умолчанию для nvcc
- Компиляция OpenMP-программы:

%*xlc_r –qsmp=smp <другие опции> -o <executable name>*

- Компиляция гибридной MPI+OpenMP-программы:

%mpixlc –qsmp=smp <другие опции> -o <executable name>

Постановка заданий на счет

- Скрипт **mpisubmit.pl**

%mpisubmit.pl [параметры скрипта] исполняемый_файл
[-- параметры исполняемого файла]

- Параметры скрипта:

-p <число процессов>

-t <число нитей>

-stdin <имя файла>

-h - выдает список опций

Постановка OpenMP-заданий на счет

%mpisubmit.pl -p 1 –t 8 <исполняемый_файл>

[-- параметры исполняемого файла]

- Если требуемое число нитей больше 8, параметр p должен быть кратным 4.
- Если на одном узле надо использовать больше 8 нитей – использовать скрипт OpenMP_job.lsf

```
#BSUB -n M
#BSUB -W 00:15
#BSUB -o my_job.%J.out
#BSUB -e my_job.%J.err
#BSUB -R "span[hosts=1]"
OMP_NUM_THREADS=N ./my_job
```

N<=160 – число потоков
M = [N/8]+1 – число ядер

%bsub < OpenMP_job.lsf

Система управления заданиями LSF. Основные команды

- ***bsub <job>*** : постановка задания в очередь
- ***bjobs*** : вывод списка заданий пользователя (по умолчанию формат вывода задается установкой LSB_BJOBS_FORMAT)
- ***bjobs -uall*** : список всех задач
- ***bjobs -l <jobid>*** : вывод информации по конкретной задче jobid
- ***bkill <jobid>*** : запрос к LSF на удаление задачи

Система управления заданиями LSF. Командные скрипты.

Пример: Командный скрипт для постановки OpenMP-программ в очередь на выполнение

```
#An example of LSF file. OpenMP_job.lsf
#BSUB -J "OpenMP_job"
#BSUB -o "OpenMP_job%J.out"
#BSUB -e "OpenMP_job%J.err"
#BSUB -R "affinity[core(N)]”
/polusfs/lsf/openmp/launchOpenMP.py ./Job_OMP
```

Комментарий

Специальный скрипт для привязки потока к ядру

Использование:

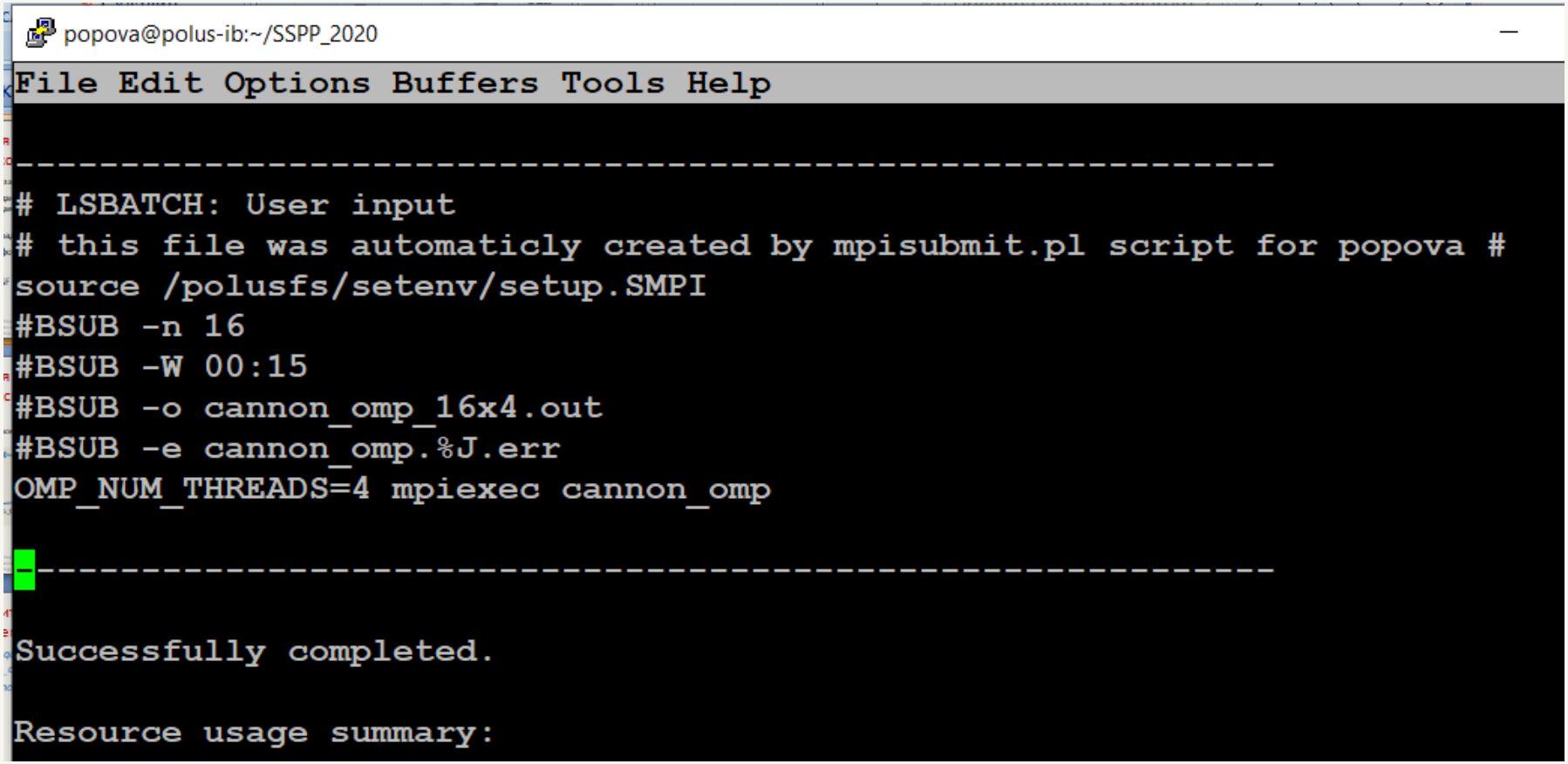
%bsub <OpenMP_job.lsf

Пример. Алгоритм Кэннона, MPI+OpenMP

- *mpixlc -O3 –qsmp=omp -qarch=pwr8 cannon_omp.c -o cannon_omp*
- *mpisubmit.pl –p 16 -t 4 –stdout cannon_omp_16x4.out cannon_omp*

```
[popova@polus-ib SSPP_2020]$ mpisubmit.pl -p 16 -t 4 --stdout cannon_omp_16x4.ou
t cannon_omp
Job <639344> is submitted to default queue <short>.
[popova@polus-ib SSPP_2020]$ bjobs
JOBID   USER                STAT  QUEUE      SLOTS NALLOC_SLOT JOB_NAME
             SUBMIT_TIME  ESTIMATED_START_TIME TIME_LEFT   PEND_REASON
639344  popova               RUN   short      16    16           *_THREADS=4 mpie
xec cannon omp Sep 29 22:36          -           0:11 L      -
```

Polus. Результат выполнения.



Polus. Результат выполнения.

```
    CPU time :                                   3467.97 sec.
    Max Memory :                                 9062 MB
    Average Memory :                             8430.56 MB
    Total Requested Memory :                     -
    Delta Memory :                               -
    Max Swap :                                   -
    Max Processes :                              20
    Max Threads :                                105
    Run time :                                   439 sec.
    Turnaround time :                            433 sec.

The output (if any) follows:

Number of MPI processes =16
 Number of OMP threads =1
Time: 429.9715s
```

Суперкомпьютерное моделирование и технологии

Лекция

Технология параллельного программирования MPI

Часть 1.

.

Попова Нина Николаевна

доцент кафедры СКИ

popova@cs.msu.su

14 октября 2020 г.

Архитектура параллельных вычислительных систем

- 2 основных класса параллельных ВС:
  - **системы с общей памятью** (многоядерные процессоры, специальные SMP-процессоры)
  - **системы с распределенной памятью**

    вычислительные узлы =

    многоядерные процессоры +

    графические ускорители

Модели параллельных программ

- **Системы с общей памятью**
  - Программирование основано на потоках
  - Программа строится на базе последовательной программы
  - Возможно автоматическое распараллеливание компилятором с использованием соответствующего ключа компилятора
  - Директивы компиляторов (**OpenMP**, OpenACC,…)

- **Системы с распределенной памятью**
  - Программа состоит из параллельных процессов
  - Явное задание коммуникаций между процессами – обмен сообщениями **“Message Passing”**

  Реализация - Message Passing библиотек:
    - MPI (“Message Passing Interface”)
    - PVM (“Parallel Virtual Machine”)
    - Shmem,

MPI

- MPI реализует модель передачи сообщений:

  Вычисления – множество процессов, взаимодействующих друг с другом посредством передачи сообщений

- MPI – **спецификация**, не язык

- Реализация спецификаций MPI реализуется в виде библиотек функций , вызываемых из языков программирования высокого уровня: C, Fortran

OpenMP

- OpenMP : **Open** specifications for **M**ulti **P**rocessing

  – многопоточный интерфейс, специально разработанный для поддержки параллельных вычислений

  - не предназначен для систем с распределенной памятью, не поддерживает передачу сообщений

  - рекомендуется для достижения максимального параллелизма на системах с общей памятью

Пример параллельной программы программы (C, OpenMP)

Сумма элементов массива

```
#include <stdio.h>
#define N 100000
int main()
{ double sum;
  double a[N];
  int   i, n =N;
  for (i=0; i<n; i++){
    a[i] = i*0.5;  }
 sum =0;
#pragma omp parallel for reduction (+:sum)
 for (i=0; i<n; i++)
  sum = sum+a[i];
printf (“Sum=%f\n”, sum);
}
```

Пример параллельной программы программы (C, MPI)

```
#include <stdio.h>
#include <mpi.h>
#define N 100000
int main(int argc, char *argv[])
{ double sum, local_sum;
  double *a;
  int  i, n =N;
  int size, myrank;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,
      &rank);
MPI_Comm_size(MPI_COMM_WORLD,
      &size);
n= n/ size;
a= (double *) malloc(n);
```

```
for (i=0;  i<n; i++)
{
    a[i] = (rank*n +i)*0.5;
  }
 local_sum =0;
for (i=0; i<n; i++)
   local_sum += a[i];
MPI_Reduce(&local_sum, &sum, 1,
 MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
If ( !rank)
    printf ("Sum =%f\n", sum);
MPI_Finalize();
return 0;
}
```

Гибрид: MPI+OpenMP

```
#include <stdio.h>
#include <mpi.h>
#define N 1024
int main(int argc, char *argv[])
{ double sum, local_sum;
  double *a;
  int  i, n =N;
  int size, myrank;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,
      &rank);
MPI_Comm_size(MPI_COMM_WORLD,
      &size);
n= n/ size;
a= (double *) malloc(n);
```

```
  for (i=0;  i<n; i++)
{
    a[i] = (rank*n+i)*0.5;
  }
 local_sum =0;
#pragma omp parallel for reduction
(+:sum)
for (i=0; i<n; i++)
  local_sum += a[i];
MPI_Reduce(&local_ sum,& sum, 1,
 MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
If ( !rank)
printf (“Sum =%f\n”, sum);
MPI_Finalize();
return 0;
}
```

MPI стандарт для построения параллельных программ для вычислительных систем с распределенной памятью

MPI – стандарт (формальная спецификация)

- MPI 1.1 Standard разрабатывался 92-94
- MPI 2.0 - 95-97
- MPI 2.1 - 2008
- MPI 3.0 – 2012
- MPI 3.1 - 2015
- Стандарты
  - **http://www.mcs.anl.gov/mpi**
  - **http://www.mpi-forum.org/docs/docs.html**

  Описание функций
  - **http://www-unix.mcs.anl.gov/mpi/www/**

Реализации MPI - библиотеки

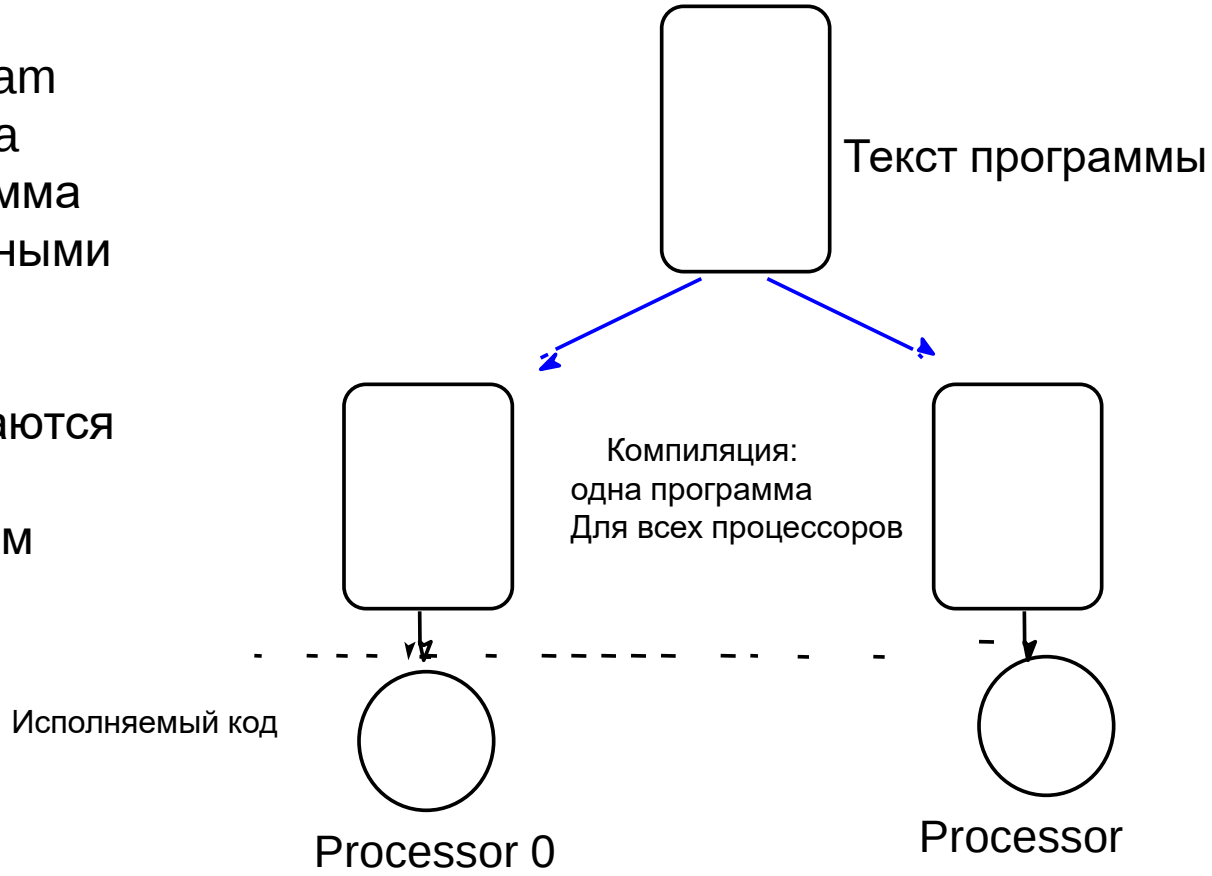
- MPICH (www.mpich.org , Argonne)
- LAM/MPI (в настоящее время не поддерживается)
- OpenMPI (www.open-mpi.org
  Los Alamos, Unis of Tennessee, Indiana & Stuttgart)
- Mvapich
- Коммерческие реализации Intel,IBM и др.

Модель MPI

- Параллельная программа состоит из процессов, процессы могут быть многопоточными.
- MPI реализует передачу сообщений между процессами.
- Межпроцессное взаимодействие предполагает:
  - синхронизацию
  - перемещение данных из адресного пространства одного процесса в адресное пространство другого процесса.

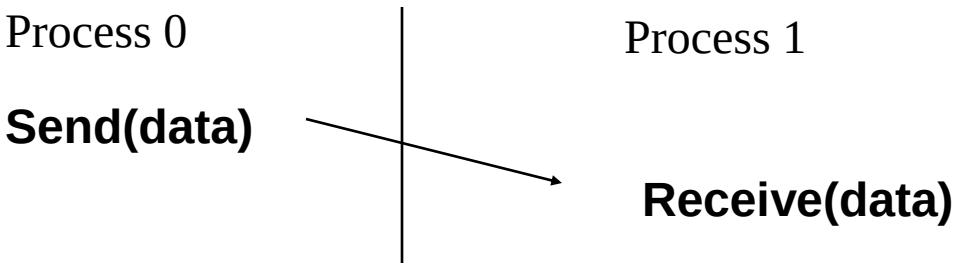
Модель MPI-программ

- **SPMD** – Single Program Multiple Data
- Одна и та же программа выполняется различными процессорами
- Управляющими операторами выбираются различные части программы на каждом процессоре.



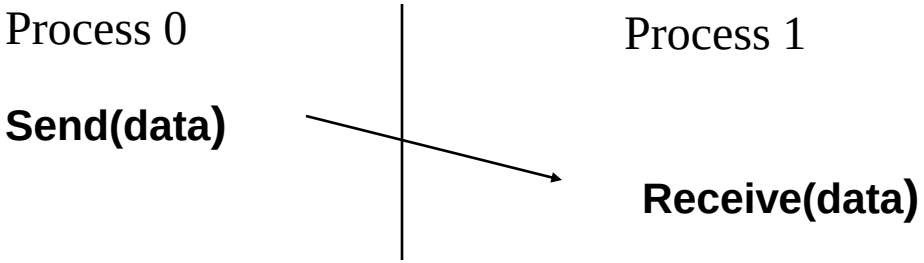
Основы передачи данных в MPI

- Данные посылаются одним процессом и принимаются другим.
- Передача и синхронизация совмещены.



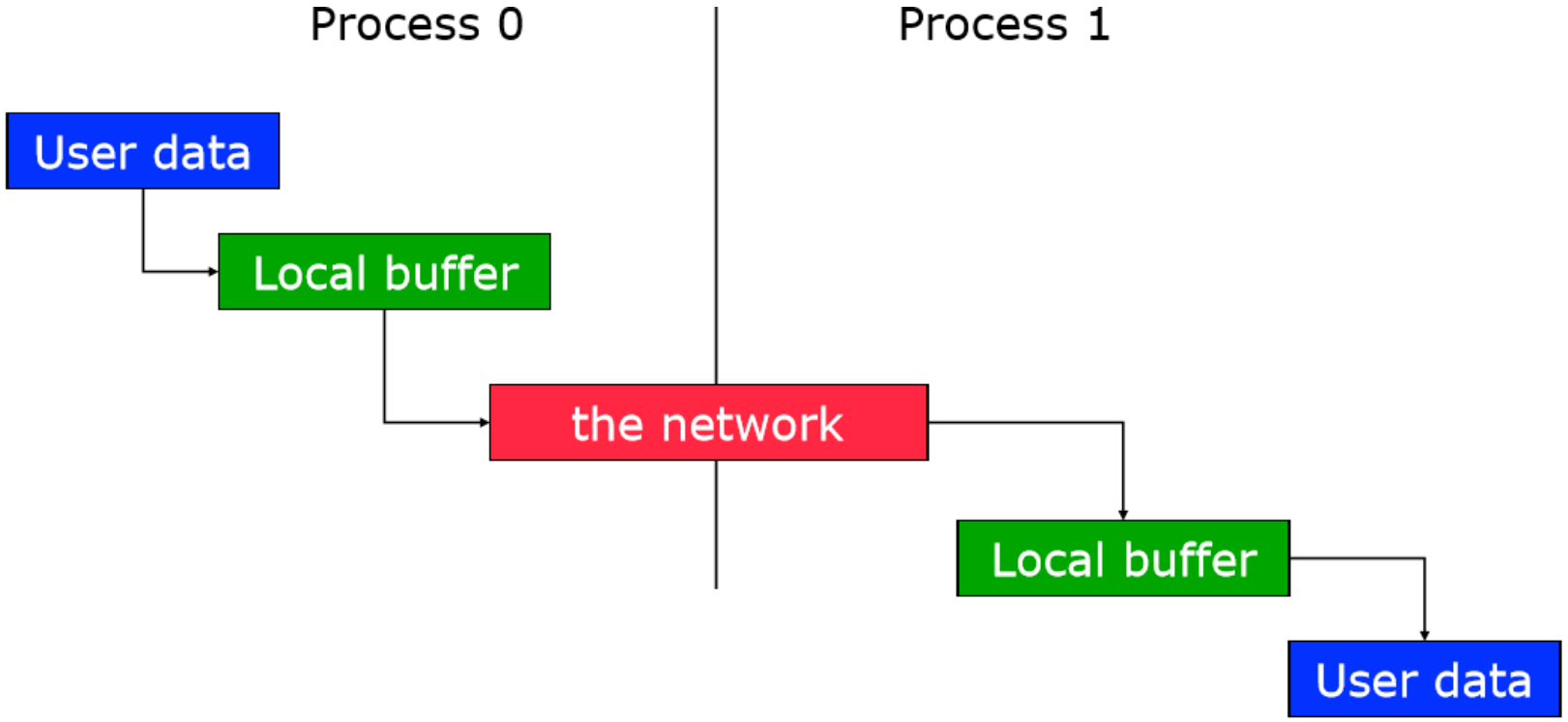
Основы передачи данных в MPI

- Необходимы уточнения процесса передачи



- Требуется уточнить:
  - Как должны быть описаны данные ?
  - Как должны идентифицироваться процессы?
  - Как получатель получит информацию о сообщении?
  - Что значить завершение передачи?

Возможная схема выполнения операций передачи сообщений



6 основных функций MPI

- **Как стартовать/завершить параллельное выполнение**
  - MPI_Init
  - MPI_Finalize
- **Кто я (и другие процессы), сколько нас**
  - MPI_Comm_rank
  - MPI_Comm_size
- **Как передать сообщение коллеге (другому процессу)**
  - MPI_Send
  - MPI_Recv

Основные понятия MPI

- Процессы объединяются в ***группы***.
- Группе приписывается ряд свойств (как связаны друг с другом и некоторые другие). Получаем ***коммуникаторы***
- Процесс идентифицируется своим номером в группе, привязанной к конкретному коммуникатору.
- При запуске параллельной программы создается специальный коммуникатор с именем ***MPI_COMM_WORLD***
- **Все** обращения к MPI функциям содержат коммуникатор, как параметр.

Понятие коммуникатора MPI

- **Все** обращения к MPI функциям содержат коммуникатор, как параметр.

- Наиболее часто используемый коммуникатор **`MPI_COMM_WORLD`**
  - определяется при вызове **`MPI_Init`**
  - содержит ВСЕ процессы программы

Типы данных MPI

- Данные в сообщении описываются тройкой: (***address, count, datatype***)
- ***datatype*** (типы данных MPI)

Signed
- MPI_CHAR
- MPI_SHORT
- MPI_INT
- MPI_LONG

Unsigned
- MPI_UNSIGNED_CHAR
- MPI_UNSIGNED_SHORT
- MPI_UNSIGNED
- MPI_UNSIGNED_LONG

- MPI_FLOAT
- MPI_DOUBLE
- MPI_LONG_DOUBLE

Тэг собщения

- Сообщение сопровождается определяемым пользователем признаком – целым числом – ***тэгом*** для идентификации принимаемого сообщения.
- Теги сообщений у отправителя и получателя должны быть согласованы. Можно указать в качестве значения тэга константу **`MPI_ANY_TAG`**.
- Некоторые не-MPI системы передачи сообщений называют тэг типом сообщения. MPI вводит понятие тэга, чтобы не путать это понятие с типом данных MPI.

Формат MPI-функций

C :

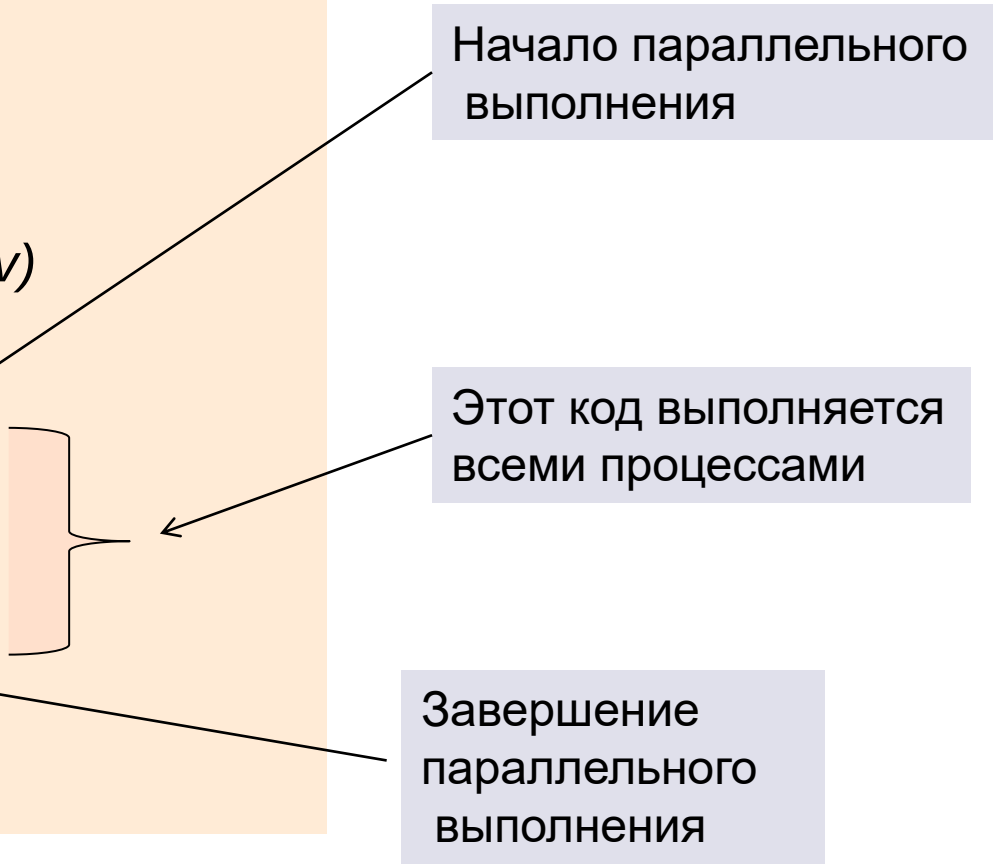
```
int MPI_Xxxxx(parameter,...);
```

C++ :

```
int MPI::Xxxxx(parameter,...);
```

Шаблон MPI-программы

```
#include < stdio.h>
#include <stdlib.h>
#include <mpi.h>
int main(int argc, char **argv)
 {
 MPI_Init(&argc, &argv);
 /* анализ аргументов */
 /*  программа */
  MPI_Finalize();
  exit (0);
 }
```



Hello, MPI world!

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv){
   int rank, size;
     MPI_Init(&argc, &argv);
     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
     MPI_Comm_size(MPI_COMM_WORLD, &size);
     printf("Hello, MPI world! I am %d of %d\n“,rank,size);
     MPI_Finalize();
     return 0; }
```

Коммуникатор MPI_COMM_WORLD

ID процесса

Число процессов

Трансляция MPI-программ

- Трансляция

  mpixlc –o <имя_программы> <имя>.c <опции>

  Например:

  mpixlc –o hw helloworld.c
- Запуск в интерактивном режиме

  mpirun –np 128 hw
- Постанока в очередь (Blue Gene/P)

  mpisubmit.bg –n 128 ./hw

Компиляция MPI-программ на Polus

- *module avail*
- *module load SpectrumMPI/10.1.0*
- Компиляция

  mpixlc –o hw helloworld.c

- Постановка на выполнение

  mpisubmit.pl [параметры скрипта] исполняемый_файл [-- параметры исполняемого файла]

  mpisubmit.pl –n 32 hw

Очереди Polus

- Проверка состояния очереди

  bjobs

или

  bjobs –u all

- Просмотр информации о задаче

  bjobs –l <job_ID>

- Какие очереди есть

  bqueues

- Удалить задачу из очереди

  bkill job_ID

Функции определения среды

```
int MPI_Init(int *argc, char ***argv)
```
должна первым вызовом, вызывается только один раз

```
int MPI_Comm_size(MPI_Comm comm, int *size)
```
число процессов в коммуникаторе

```
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```
номер процесса в коммуникаторе (нумерация с 0)

```
int MPI_Finalize()
```
завершает работу процесса

```
int MPI_Abort (MPI_Comm_size(MPI_Comm comm,
int*errorcode)
```
завершает работу программы

Инициализация MPI

- **MPI_Init** должна первым вызовом, вызывается только один раз

```
int MPI_Init(int *argc, char ***argv)
```

- Проверка была ли выполнена инициализация:

```
int MPI_Initialized( int *flag )
flag = true, если MPI уже инициализирована.
```

Завершение MPI-процессов

- Никаких вызовов MPI функций после

  int MPI_Finalize()

  int MPI_Abort(MPI_Comm comm, int errorcode)

  errorcode - код ошибки для возврата в среду исполнения.

Пользователь обязан гарантировать, что все незаконченные обмены будут завершены прежде, чем будет вызвана MPI_FINALIZE.

Если какой-либо из процессов не выполняет MPI_Finalize, программа зависает.

Количество процессов в коммуникаторе

- Размер коммуникатора

```
int MPI_Comm_size(MPI_Comm comm, int *size)
```

Результат – число процессов в коммуникаторе

MPI_Comm_rank
номер процесса (process rank)

- Process ID в коммуникаторе
  - Начинается с 0 до (*n*-1), где *n* – число процессов
- Используется для определения номера процесса в коммуникаторе

```
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```

Результат – номер процесса

Взаимодействие «точка-точка»

- Самая простая форма обмена сообщением
- Один процесс посылает сообщения другому
- Несколько вариантов реализации того, как пересылка и выполнение программы совмещаются

MPI_Send

*int MPI_Send(void *buf,int count, MPI_Datatype datatype,int dest,*
int tag, MPI_Comm comm)

buf		адрес буфера
count	–	число пересылаемых элементов
Datatype	–	MPI datatype
dest	-	rank процесса-получателя
tag	-	определяемый пользователем параметр,
comm	–	MPI-коммуникатор

Пример:

```
MPI_Send(data,500,MPI_FLOAT,6,33,MPI_COMM_WORLD)
```

MPI_Recv

*int MPI_Recv(void *buf,int count, MPI_Datatype datatype,int source, int tag, MPI_Comm comm,* ***MPI_Status *status*** *)*

buf адрес буфера
count – число пересылаемых элементов
Datatype – MPI datatype
source - rank процесса-отправителя
tag - определяемый пользователем параметр,
comm – MPI-коммуникатор,
status - статус

Пример:

```
MPI_Recv(data,500,MPI_FLOAT,6,33,MPI_COMM_WORLD,&stat)
```

Завершение

- “Завершение” передачи означает, что буфер в памяти, занятый для передачи, может быть безопасно использован для доступа, т.е.
  - Send: переменная, задействованная в передаче сообщения, может быть доступна для дальнейшей работы
  - Receive: переменная, получающая значение в результате передачи, может быть использована

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv){
   int rank, size; int x; MPI_Status status;
   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
    if(!rank) {
     MPI_Send (&x,1, MPI_INT, 1,0,MPI_COMM_WORLD);
      printf(" I am %d. I sent x to %d\n“,x,1);
    } else {
    MPI_Recv (&x,1, MPI_INT,
      0,0,MPI_COMM_WORLD,&status);
    printf(" I am %d. I send my rank to %d\n“,rank,0); }
   MPI_Finalize();
   return 0;
 }
```

0-ой процесс посылает x 1-ому процессу

1-ый процесс принимает целое число в переменную x

Программа будет работать правильно только для запуска на 2-ух процессах. ПОЧЕМУ?

Wildcarding (джокеры)

- Получатель может использовать джокер
  для получения сообщения от **ЛЮБОГО** процесса
  `MPI_ANY_SOURCE`

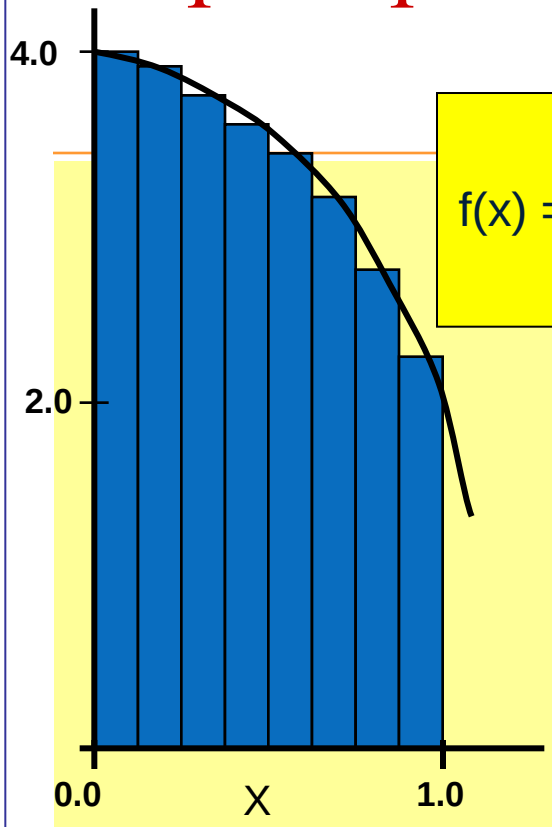
- Для получения сообщения с ЛЮБЫМ тэгом
  `MPI_ANY_TAG`

- Реальные номер процесса-отправителя и тэг возвращаются через параметр ***`status`***

Нулевые процессы

- В качестве отправителя или получателя, вместо номера может быть использовано специальное значение MPI_PROC_NULL.
- Обмен с процессом, который имеет значение MPI_PROC_NULL, не дает результата.
- Передача в процесс MPI_PROC_NULL успешна и заканчивается сразу, как только возможно.
- Прием от процесса MPI_PROC_NULL успешен и заканчивается сразу, как только возможно без измененения буфера приема.
-
- При выполнении приема из source = MPI_PROC_NULL в статус возвращает source = MPI_PROC_NULL, tag = MPI_ANY_TAG и count = 0

Пример: численное интегрирование



$$f(x) = \frac{4.0}{(1+x^2)}$$

```
static long num_steps=100000;
 double step, pi;

void main()
{  int i;
   double x, sum = 0.0;

   step = 1.0/(double) num_steps;
   for (i=0; i< num_steps; i++){
      x = (i+0.5)*step;
      sum = sum + 4.0/(1.0 + x*x);
   }
   pi = step * sum;
   printf("Pi = %f\n",pi);
}
```

Вычисление числа π с использованием MPI

```
#include "mpi.h"
#include <stdio.h>
int main (int argc, char *argv[])
{
    int n =100000, myid, numprocs, i;
    double mypi, pi, h, sum, x;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    h   = 1.0 / (double) n;
    sum = 0.0;
```

Вычисление числа π с использованием MPI

```
for (i = myid + 1; i <= n; i += numprocs)
  {
                x = h * ((double)i - 0.5);
                sum += (4.0 / (1.0 + x*x));
   }
   mypi = h * sum;
```

Вычисление числа π с использованием MPI

```
if (!myid){
   pi= mypi;
   for (i= 1; i<numproc; i++){
    MPI_Recv (&mypi,1, MPI_DOUBLE, MPI_ANY_SOURCE,
     MPI_ANY_TAG, MPI_COMM_WORLD,MPI_STATUS_IGNORE );
     pi+=mypi; }
    printf ( "PI is approximately %.16f ", pi);
     }
else
  MPI_Send(&mypi, 1,MPI_DOUBLE,0, 0, MPI_COMM_WORLD);
MPI_Finalize();
    return 0;
}
```

Информация о завершившемся приеме сообщения

- Возвращается функцией **`MPI_Recv`** через `параметр` **`status`**

- Содержит:
  - Source: ***`status.MPI_SOURCE`***
  - Tag: ***`status.MPI_TAG`***
  - Count: ***`MPI_Get_count`***

Полученное сообщение

- Может быть меньшего размера, чем указано в функции MPI_Recv, но НЕ БОЛЬШЕ!
- `count` – число реально полученных элементов

Функция для определения числа полученных элементов:

int ***MPI_Get_count*** (MPI_Status *status,
MPI_Datatype datatype, int *count)

MPI_Probe

int MPI_Probe(int *source*, int *tag*, MPI_Comm *comm*, MPI_Status* *status*)

Проверка статуса операции приема сообщения.

Параметры аналогичны функции MPI_Recv

Условия успешного взаимодействия «точка-точка»

- Отправитель должен указать правильный rank получателя
- Получатель должен указать верный rank отправителя
- Одинаковый коммуникатор
- Тэги должны соответствовать друг другу
- Буфер у процесса-получателя должен быть достаточного объема

Замер времени MPI_Wtime

- Время замеряется в секундах

```
double MPI_Wtime(void)
```

Пример.

```
double start, finish, time ;
start=-MPI_Wtime;
MPI_Send(…);
finish = MPI_Wtime();
time= start+finish;
```

Сумма элементов вектора с использованием MPI_Send и MPI_Recv

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#define N_LOCAL 1024
int main(int argc, char *argv[])
{ double sum_local, sum_global, start, finish;
  double a[N_LOCAL];
  int  i, n =N_LOCAL;
  int size, myrank;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
```

Сумма элементов вектора с использованием MPI_Send и MPI_Recv

```
If (!myrank)  /* process-root */
{ start = MPI_Wtime(); sum_global=0;
  for (i=1; i<size; i=+) {
  MPI_Recv( &sum_local, 1,  MPI_DOUBLE, MPI_ANY_SORCE,
  MPI_ANY_TAG, MPI_COMM_WORLD, &status);
  sum_global += sum_local; }
  printf (“ Sum of vector elements = %f \n Time =%f\n”, sum_global,
MPI_Wtime() - start);
}
 else { /* processes: 1 .. size-1 */
//Инициализация элементов вектора a
sum_local =0;
for(i=0; i<n; i++) sum_local+= a[i];
 MPI_Send (&sum_local, 1 ,  MPI_DOUBLE, 0,0, MPI_COMM_WORLD);
}
MPI_Finalized();
 exit (0); }
```

Литература

- *Антонов А. С.* Технологии параллельного программирования MPI и OpenMP: Учеб. пособие. Предисл.: В.А.Садовничий. — Издательство Московского университета М.:, 2012. — С. 344.
- Интернет ресурсы:
- www.parallel.ru, intuit.ru, www.mpi-forum.org, http://www.mcs.anl.gov/research/projects/mpi/www/www3

Суперкомпьютерное моделирование и технологии

Лекция

Технология параллельного программирования MPI. Часть 2.

.

Попова Нина Николаевна

доцент кафедры СКИ

popova@cs.msu.su

21 октября 2020 г.

MPI стандарт для построения параллельных программ для вычислительных систем с распределенной памятью

Основы MPI

- MPI реализует модель параллельного программирования для систем с распределенной памятью
- Не требует специальных компиляторов, реализуется в виде библиотек
- Содержит более 300 функций

Модель MPI

- Параллельная программа состоит из процессов, процессы могут быть многопоточными.
- MPI реализует передачу сообщений между процессами.
- Основная схема взаимодействия между 2-мя процессами: схема «рукопожатия» – процессы согласовывают передачу .
- Межпроцессное взаимодействие предполагает:
  - синхронизацию
  - перемещение данных из адресного пространства одного процесса в адресное пространство другого процесса.

6 основных функций MPI

- **Как стартовать/завершить параллельное выполнение**
  - MPI_Init
  - MPI_Finalize
- **Кто я (и другие процессы), сколько нас**
  - MPI_Comm_rank
  - MPI_Comm_size
- **Как передать сообщение коллеге (другому процессу)**
  - MPI_Send
  - MPI_Recv

Замер времени MPI_Wtime

- Время замеряется в секундах
- Выделяется интервал в программе

```
double MPI_Wtime(void)
```

Пример.

```
double start, finish, time ;
start=-MPI_Wtime;
MPI_Send(…);
finish = MPI_Wtime();
time= start+finish;
```

Сумма элементов вектора с использованием MPI_Send и MPI_Recv

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#define N_LOCAL 1024
int main(int argc, char *argv[])
{ double sum_local, sum_global, start, finish;
  double a[N_LOCAL];
  int  i, n =N_LOCAL;
  int size, myrank;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
```

Сумма элементов вектора с использованием MPI_Send и MPI_Recv

```
If (!myrank)  /* process-root */
{ start = MPI_Wtime(); sum_global=0;
  for (i=1; i<size; i=+) {
  MPI_Recv( &sum_local, 1,  MPI_DOUBLE, MPI_ANY_SOURCE,
  MPI_ANY_TAG, MPI_COMM_WORLD, &status);
  sum_global += sum_local; }
  printf (“ Sum of vector elements = %f \n Time =%f\n”, sum_global,
MPI_Wtime() - start);
}
 else { /* processes: 1 .. size */
sum_local =0;
for(i=0; i<n; i++) sum_local+= a[i];
 MPI_Send (&sum_local, 1 ,  MPI_DOUBLE, 0,0, MPI_COMM_WORLD);
}
MPI_Finalize();
 exit (0); }
```

MPI_Probe

int MPI_Probe(int *source*, int *tag*, MPI_Comm *comm*, MPI_Status* *status*)

Проверка статуса операции приема сообщения.

Параметры аналогичны функции MPI_Recv

Deadlocks

- Процесс 0 посылает большое сообщение процессу 1
  - Если в принимающем процессе недостаточно места в системном буфере, процесс 0 должен ждать пока процесс 1 не предоставит необходимый буфер.
  - Что произойдет:

Process 0	Process 1
Send(1)	**Send(0)**
Recv(1)	**Recv(0)**

- Называется “unsafe” потому, что зависит от системного буфера.

Пути решения «ubsafe» передач

- Упорядочить передачи:

Process 0	Process 1
`Send(1)`	`Recv(0)`
`Recv(1)`	`Send(0)`

- Использовать неблокирующие передачи:

Process 0	Process 1
`Isend(1)`	`Isend(0)`
`Irecv(1)`	`Irecv(0)`
`Waitall`	`Waitall`

Неблокирующие коммуникации

Цель – уменьшение времени работы параллельной программы за счет совмещения вычислений и обменов.

Неблокирующие операции завершаются, не дожидаясь окончания передачи данных. В отличие от аналогичных блокирующих функций изменен критерий завершения операций – немедленное завершение.

Проверка состояния передач и ожидание завершение передач выполняются специальными функциями.

Форматы неблокирующих функций

```
MPI_Isend(buf,count,datatype,dest,tag,comm,request)

MPI_Irecv(buf,count,datatype,source,tag,comm,
request)
```

request – “квитанция» о завершении передачи.
Тип: MPI_Requset
MPI_REQUEST_NULL – обнуление

MPI_Wait() ожидание завершения.

MPI_Test() проверка завершения. Возвращается флаг, указывающий на результат завершения.

Асинхронные передачи

*int MPI_Isend(void *buf,int count, MPI_Datatype datatype,int dest, int tag, MPI_Comm comm, MPI_Requset request)*

*int MPI_Irecv(void *buf,int count, MPI_Datatype datatype,int source, int tag, MPI_Comm comm, MPI_Status *status, MPI_Requset request)*

*int MPI_Iprobe(int source, int tag, MPI_Comm comm, int *flag, MPI_Status *status);*

*int MPI_Wait(MPI_Request *request, MPI_Status *status)*

*int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)*

flag : true, если передача завершена

Ожидание завершения асинхронных передач

int **MPI_Waitall (int count, MPI_Request array_of_requests[],**
MPI_Status array_of_statuses[])
// waits for all given communications to finish and fills in the statuses
Int **MPI_Waitany (int count, MPI_Request array_of_requests[], int *index,**
MPI_Status *status)
// waits for one of the given communications to finish, sets the index to indicate
// which one and fills in the status
int **MPI_Waitsome (int incount, MPI_Request array_of_requests[],**
int *outcount, int array_of_indices[], MPI_Status array_of_statuses[])
// waits for at least one of the given communications to finish, sets the number
// of communication requests that have finished, their indices and status

Проверка состояния передач

int **MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)**

// tests if the communication is finished. Sets flag to 1 and fills in the status if
// finished or sets the flag to 0 if not finished.

int **MPI_Testall(int count, MPI_Request array_of_requests[], int *flag,**
MPI_Status array_of_statuses[])

// test whether all given communications are finished. Sts flag to 1 and fills in
// the status aray if all are finished or sets the flag to 0 if not all are finished.

int **MPI_Testany(int count, MPI_Request array_of_requests[], int *index,**
int *flag, MPI_Status *status)

// test whether one of the given communications is finished. Sets flag to 1 and fills
// in the index and status if one finished or sets the flag to 0 if none is finished.

int **MPI_Testsome(int incount, MPI_Request array_of_requests[], int *outcount,**
int array_of_indices[], MPI_Status array_of_statuses[])

// tests whether some of the given communications is finished, sets the number
// of communication requests that have finished, their indices and statuses.

int **MPI_Cancel(MPI_Request *request)**

Deadlocks

Пример. Пересылка по кольцу

Пусть каждый i-ый процесс посылает сообщение i+1 процессу (по модулю = число процессов) и получает сообщение от i-1 процесса:

```
int a[10], b[10], size, myrank;
MPI_Status status;
...
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Send(a, 10, MPI_INT, (myrank+1)%size, 1,
       MPI_COMM_WORLD);
MPI_Recv(b, 10, MPI_INT, (myrank-1+size)%size, 1,
       MPI_COMM_WORLD, &status);
...
```

Deadlock

Deadlocks

Решение:

```
int a[10], b[10], size, myrank;
MPI_Status status;
...
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank%2 == 1) {
        MPI_Send(a, 10, MPI_INT, (myrank+1)%size, 1,
        MPI_COMM_WORLD);
        MPI_Recv(b, 10, MPI_INT, (myrank-1+size)%size, 1,
        MPI_COMM_WORLD,&status);
}
else {
        MPI_Recv(b, 10, MPI_INT, (myrank-1+size)%size, 1,
        MPI_COMM_WORLD,&status);
        MPI_Send(a, 10, MPI_INT, (myrank+1)%size, 1,
        MPI_COMM_WORLD);
}
...
```

Пример. Пересылка по кольцу с использованием асинхронных передач (1)

```
/******************************************************************************/
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
int numtasks, rank, next, prev, buf[2], tag1=1, tag2=2;
MPI_Request reqs[4];
MPI_Status stats[4];

MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

Пример. Пересылка по кольцу с использованием асинхронных передач (2)

```
**********************************************************************/
prev = rank-1;
next = rank+1;
if (rank == 0)  prev = numtasks - 1;
if (rank == (numtasks - 1))  next = 0;

MPI_Irecv(&buf[0], 1, MPI_INT, prev, tag1, MPI_COMM_WORLD, &reqs[0]);
MPI_Irecv(&buf[1], 1, MPI_INT, next, tag2, MPI_COMM_WORLD, &reqs[1]);

MPI_Isend(&rank, 1, MPI_INT, prev, tag2, MPI_COMM_WORLD, &reqs[2]);
MPI_Isend(&rank, 1, MPI_INT, next, tag1, MPI_COMM_WORLD, &reqs[3]);

MPI_Waitall(4, reqs, stats);
printf("Task %d communicated with tasks %d & %d\n",rank,prev,next);
MPI_Finalize();
}
```

Совмещение посылки и приема сообщений

```
int MPI_Sendrecv(
void *sendbuf, int sendcount,MPI_Datatype senddatatype, int dest, int sendtag,
void *recvbuf, int recvcount, MPI_Datatype recvdatatype, int source, int recvtag,
            MPI_Comm comm, MPI_Status *status)
```

Если использовать один буфер:

```
int MPI_Sendrecv_replace(
            void *buf, int count,MPI_Datatype datatype,
             int dest, int sendtag,
             int source, int recvtag,
            MPI_Comm comm, MPI_Status *status)
```

Пример использования MPI_Sendrecv

Пересылка по кольцу. Решение 2.

```
int a[10], b[10], size, myrank;
MPI_Status status;
...
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Sendrecv(a, 10, MPI_INT, (myrank+1)%size , 1,
                    b, 10, MPI_INT, (myrank-1+size)%size, 1,
                    MPI_COMM_WORLD, &status)

...
```

Коллективные передачи

- Передача сообщений между группой процессов
- Вызываются ВСЕМИ процессами в коммуникаторе
- Примеры:
  - Broadcast, scatter, gather (рассылка данных)
  - Global sum, global maximum, и т.д. (Коллективные операции)
  - Барьерная синхронизация

Характеристики коллективных передач

- Коллективные операции не являются помехой операциям типа «точка-точка» и наоборот
- Все процессы коммуникатора должны вызывать коллективную операцию
- Синхронизация не гарантируется (за исключением барьера).
  Завершение операции – локально в процессе
- Нет тэгов
- Принимающий буфер должен точно соответствовать размеру отсылаемого буфера
- Асинхронные коллективные передачи - в MPI-3

Функции коллективной передачи

- MPI_Bcast() – Broadcast (one to all)
- MPI_Reduce() – Reduction (all to one)
- MPI_Allreduce() – Reduction (all to all)
- MPI_Scatter() – Distribute data (one to all)
- MPI_Gather() – Collect data (all to one)
- MPI_Alltoall() – Distribute data (all to all)
- MPI_Allgather() – Collect data (all to all)

Барьерная синхронизация

- Приостановка процессов до выхода ВСЕХ процессов коммуникатора в заданную точку синхронизации

```
int MPI_Barrier (MPI_Comm comm)
```

Пример – упорядоченный вывод:

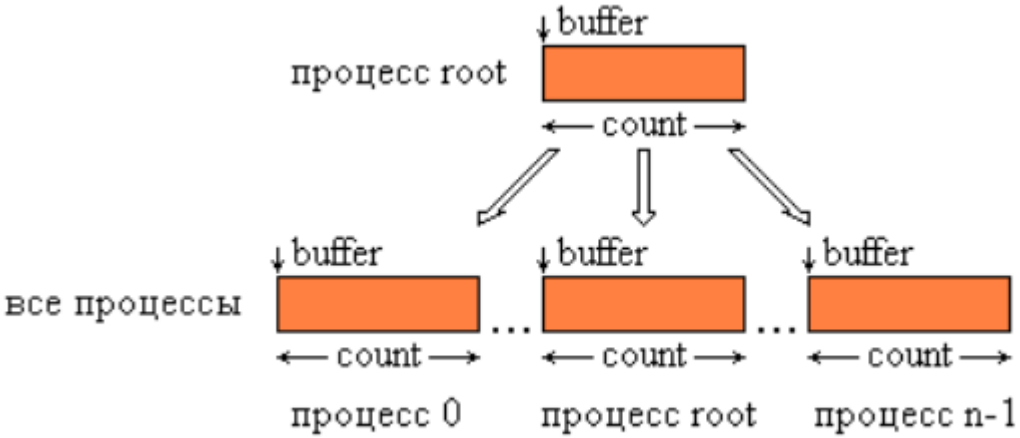
```
int size,rank;
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
for (i=0;i<size;i++)
{ MPI_Barrier(MPI_COMM_WORLD);
if (rank==i) printf("%d\n", rank);
}
```

Широковещательная рассылка

- One-to-all передача: один и тот же буфер отсылается от процесса root всем остальным процессам в коммуникаторе

```
int MPI_Bcast (void *buffer, int count,
  MPI_Datatype datatype,int root, MPI_Comm comm)
```

- Все процессы должны указать один тот же root и communicator



Широковещательная рассылка

- Асинхронная широковещательная рассылка (MPI-3)

```
int MPI_Ibcast (void *buffer, int count,
    MPI_Datatype datatype,int root, MPI_Comm comm,
  MPI_Request *request)
```

Пример использования асинхронной широковещательной рассылки (1)

```
#include <stdio.h>
 #include <string.h>
 #include "mpi.h“
 int main(int argc, char **argv) {
 char message[20];
int i, rank, size;
 MPI_Status status; MPI_Request request;
 int root = 0;
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 if (rank == root) { strcpy(message, "Hello, world"); }
 MPI_Ibcast(message, 13, MPI_CHAR, root, MPI_COMM_WORLD,
    MPI_Request *request);
printf( "Message from process %d : %s\n", rank, message);// ERROR
 MPI_Finalize(); }
```

Пример использования асинхронной широковещательной рассылки (2)

```
#include <stdio.h>
 #include <string.h>
 #include "mpi.h“
 int main(int argc, char **argv) {
 char message[20];
int i, rank, size;
 MPI_Status status; MPI_Request request;
 int root = 0;
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 if (rank == root) { strcpy(message, "Hello, world"); }
 MPI_Ibcast(message, 13, MPI_CHAR, root, MPI_COMM_WORLD,
     &request); MPI_wait (&request, &status);
printf( "Message from process %d : %s\n", rank, message);
 MPI_Finalize(); }
```

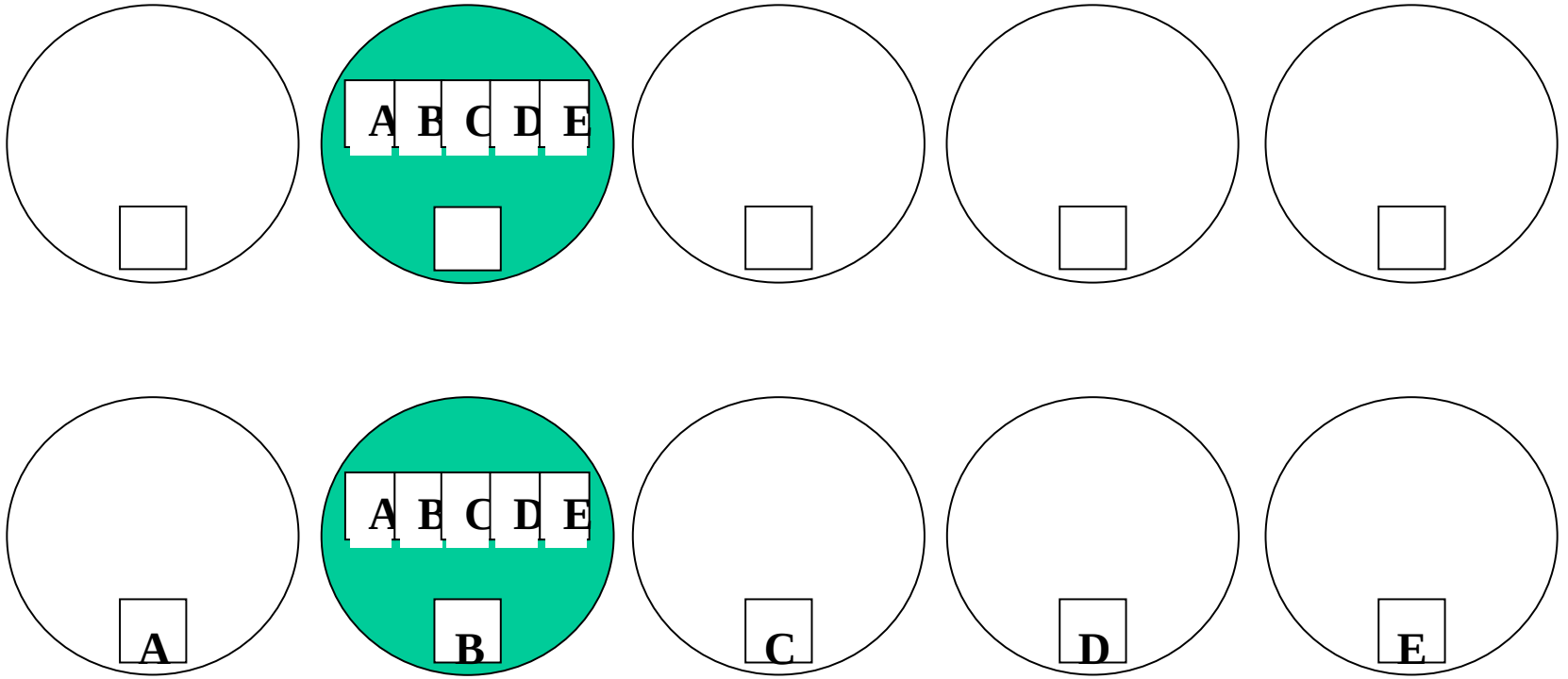
Функция Scatter рассылки блоков данных

- One-to-all communication: блоки данных одного размера из одного процесса рассылаются всем процессам коммуникатора (в порядке их номеров)

```
int MPI_Scatter(void* sendbuf, int sendcount,
    MPI_Datatype sendtype, void* recvbuf,
    int recvcount,MPI_Datatype recvtype, int root,
    MPI_Comm comm)
```

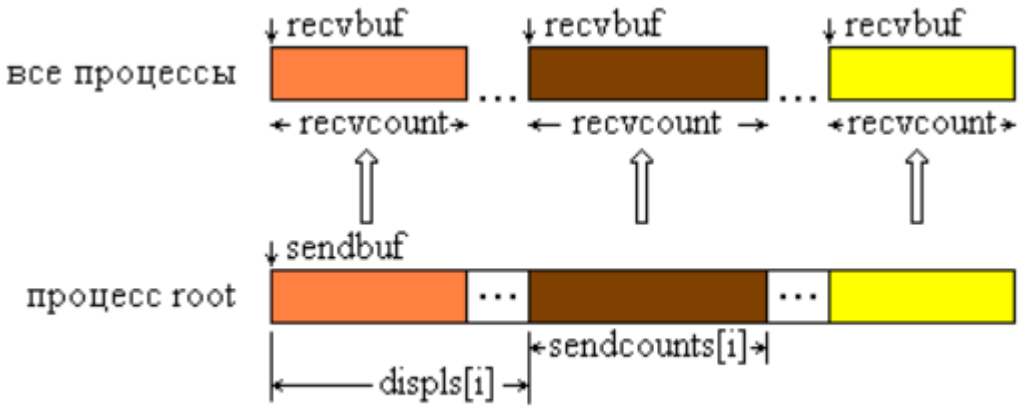
- *sendcount* – число элементов, посланных каждому процессу**, не общее число отосланных элементов;**
- send параметры (sendbuf, sendcount, sendtype) имеют смысл только для процесса root

Scatter – графическая иллюстрация

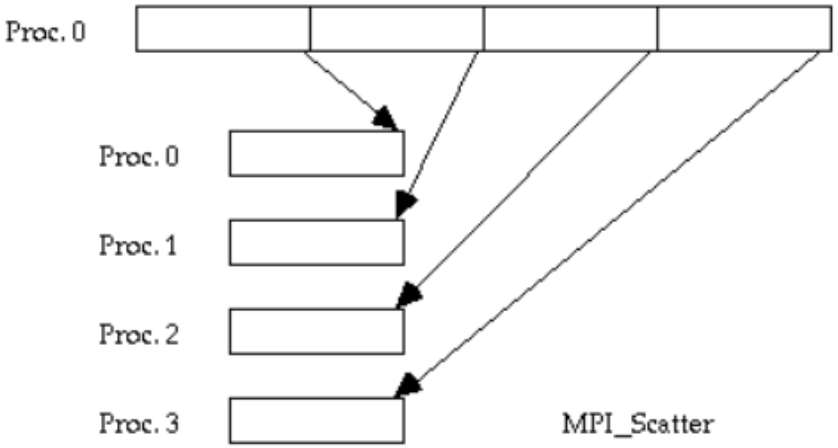


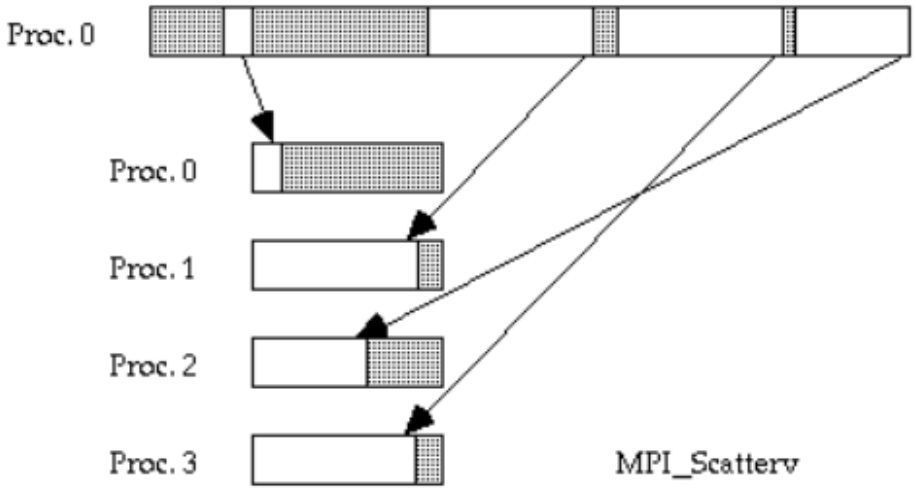
Функция Scatterv рассылки блоков разной длины

```
int MPI_Scatterv(void* sendbuf, int *sendcounts,
int *displs, MPI_Datatype sendtype, void* recvbuf,
int recvcount,
MPI_Datatype recvtype, int root, MPI_Comm comm)
```



Сравнение: MPI_Scatter & MPI_Scatterv





Пример использования MPI_Scatterv

Рассылка массива в случае, если размер массива N НЕ ДЕЛИТСЯ нацело на число процессов size

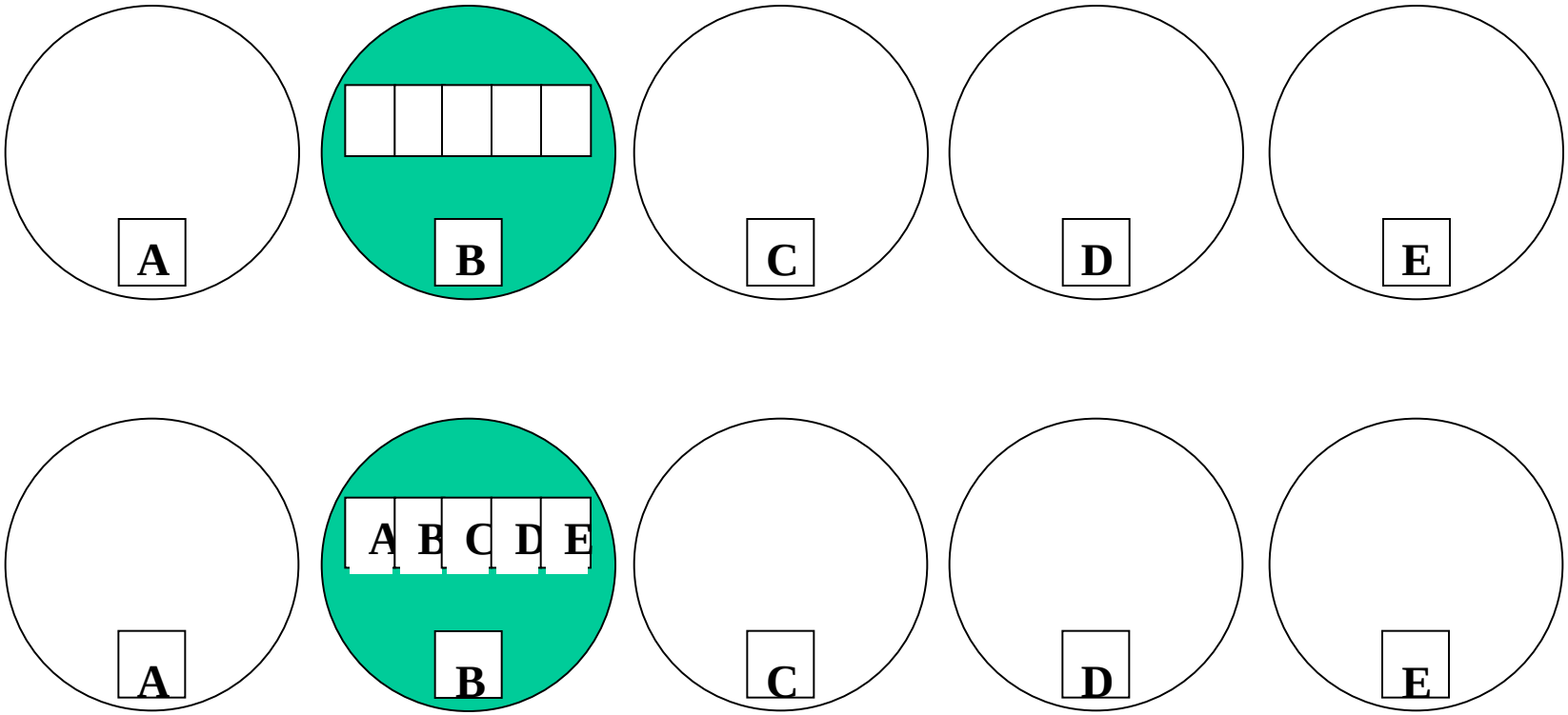
```
nmin = N/size;
 nextra = N%size;
 k = 0;
 for (i=0; i<size; i++) {
    if (i<nextra) sendcounts[i] = nmin+1;
    else sendcounts[i] = nmin; displs[i] = k; k = k+sendcounts[i]; }
  // need to set recvcount also ...
  MPI_Scatterv( sendbuf, sendcounts, displs, ...
```

Функция Gather сбора данных

- All-to-one передачи: блоки данных одинакового размера собираются процессом root
- Сбор данных выполняется в порядке номеров процессов
- Длина блоков предполагается одинаковой, т.е. данные, посланные процессом i из своего буфера sendbuf, помещаются в i-ю порцию буфера recvbuf процесса root. Длина массива, в который собираются данные, должна быть достаточной для их размещения.

```
int MPI_Gather(void* sendbuf, int sendcount,
  MPI_Datatype sendtype,
  void* recvbuf, int recvcount, MPI_Datatype recvtype,
  int root, MPI_Comm comm)
```

Gather – графическая илюстрация



Функция Gatherv сбора блоков данных разной длины

- Сбор данных разного размера в процессе root в порядке номеров процессов
- Длина блоков предполагается разной для процессов, т.е. данные, посланные процессом i из своего буфера sendbuf, помещаются в i-ю порцию буфера recvbuf процесса root. Начало i-ой порции определяется смещением, указанным в массиве displs. Длина массива, в который собираются данные, должна быть достаточной для их размещения.

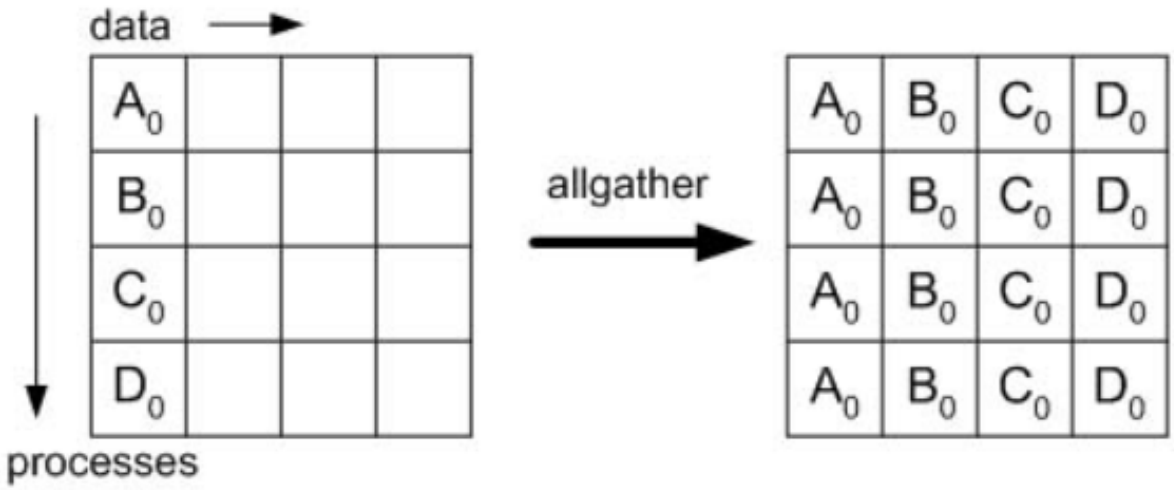
```
int MPI_Gatherv (void *sendbuf, int sendcount,
MPI_Datatype sendtype, void *recvbuf, int *recvcounts,
int *displs, MPI_Datatype recvtype, int root, MPI_Comm
comm)
```

MPI_Allgather

int MPI_Allgather(const void* sbuf, int scount, MPI_Datatype stype, void* rbuf, int rcount, MPI_Datatype rtype, MPI_Comm comm)

int MPI_Allgatherv(const void* sbuf, int scounts, MPI_Datatype stype, void* rbuf, const int rcounts[], const int displs[], MPI_Datatype rtype, MPI_Comm comm)

Иллюстрация MPI_Allgather



In Place

Позволяет избегать локальное копирование, например из **send** буфер в **receive** буфер.

В качестве одного из буферов (всегда меньшего, если они различаются по размеру) можно использовать специальное значение MPI_IN_PLACE .Тип данных и количество передаваемых элементов этого буфера игнорируются.

Для MPI_Gather и MPI_Reduce это значение используется для **send** buffer.

Для MPI_Scatter это значение используется для **receive** buffer.

Пример использования In Place

```
int value = …;
MPI_Gather(&value,              1,   MPI_INT,
                  recv_buf,           1,   MPI_INT,
                   root, comm);
intvalue = …;
if (rank == root) {
    recv_buf[root] = value;
    MPI_Gather(MPI_IN_PLACE, 1, MPI_INT,
                      recv_buf,          1, MPI_INT,
                       root, comm);}
 else {
   MPI_Gather(&value,              1,  MPI_INT,
                      recv_buf,          1, MPI_INT,
                       root, comm);
}
```

Требуется различать root и не-root

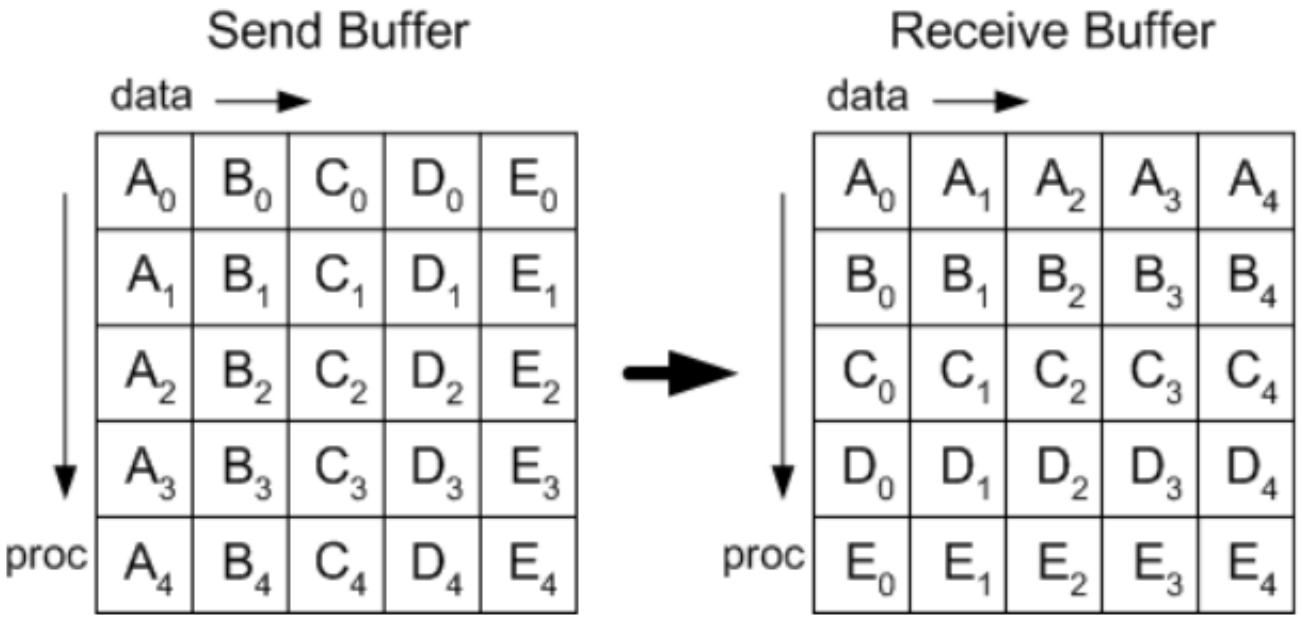
MPI_ALLTOALL

```
int MPI_Alltoall(
void* sendbuf, int sendcount, MPI_Datatype sendtype,
 void* recvbuf, int recvcount,  MPI_Datatype recvtype,
 MPI_Comm comm);
```

Описание:

- Рассылка сообщений от каждого процесса каждому
- j-ый блок данных из процесса i принимается j-ым процессом и размещается в i-ом блоке буфера recvbuf

MPI_ALLTOALL



Глобальные операции редукции

- Операции выполняются над данными, распределенными по процессам коммуникатора
- Примеры:
  - Глобальная сумма или произведение
  - Глобальный максимум (минимум)
  - Глобальная операция, определенная пользователем

Общая форма

```
int MPI_Reduce(void* sendbuf, void* recvbuf,
int count, MPI_Datatype datatype, MPI_Op op,
int root,MPI_Comm comm)
```

- **count** число операций “*op*” выполняемых над последовательными элементами буфера **sendbuf**
- (также размер **recvbuf**)

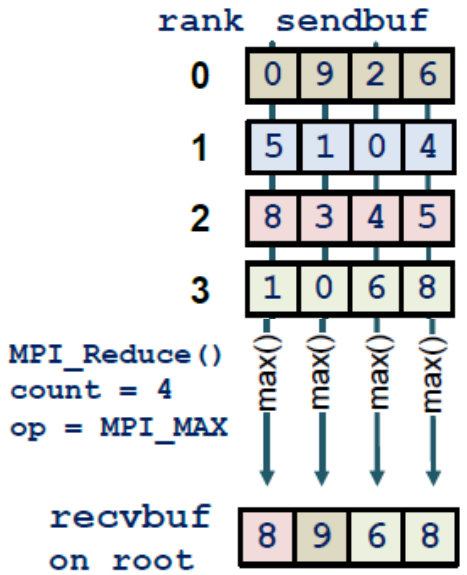
- **op** является ассоциативной операцией, которая выполняется над парой операндов типа **datatype** и возвращает результат того же типа

Предопределенные операции редукции

MPI Name	Function
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical AND
MPI_BAND	Bitwise AND
MPI_LOR	Logical OR
MPI_BOR	Bitwise OR
MPI_LXOR	Logical exclusive OR
MPI_BXOR	Bitwise exclusive OR
MPI_MAXLOC	Maximum and location
MPI_MINLOC	Minimum and location

Пример MPI_Reduce

MPI_Reduce (sendbuf, recvbuf, 4, MPI_INT,
MPI_MAX, 0, MPI_COMM_WORLD)



Варианты реализации MPI_Reduce

int MPI_Reduce(const void* sbuf, void* rbuf, int count, MPI_Datatype stype, MPI_Op op, int root, MPI_Comm comm)

int MPI_Allreduce(const void* sbuf, void* rbuf, int count MPI_Datatype stype, MPI_Op op, MPI_Comm comm)

int MPI_Reduce_scatter_block(const void* sbuf, void* rbuf, int rcount, MPI_Datatype stype, MPI_Op op, MPI_Comm comm)

int MPI_Reduce_scatter(const void* sbuf, void* rbuf, const int[] rcount, MPI_Datatype stype, MPI_Op op, MPI_Comm comm)

int MPI_Ireduce(const void* sbuf, void* rbuf, int count, MPI_Datatype stype, MPI_Op op, int root, MPI_Comm comm, MPI_Request *request)

In Place

В качестве одного из буферов в коллективных функциях (всегда меньшего, если они различаются по размеру) можно использовать специальное значение **MPI_IN_PLACE** .Тип данных и количество передаваемых элементов этого буфера игнорируются.

Это позволяет избегать локальное копирование, например из **send** буфер в **receive** буфер.

Для MPI_Gather и MPI_Reduce MPI_IN_PLACE используется для **send** buffer.

Для MPI_Scatter это значение используется для **receive** buffer.

Пример использования In Place

```
int value = …;
MPI_Gather(&value,              1,   MPI_INT,
                  recv_buf,           1,   MPI_INT,
                   root, comm);
intvalue = …;
if (rank == root) {
    recv_buf[root] = value;
    MPI_Gather(MPI_IN_PLACE, 1, MPI_INT,
                      recv_buf,          1, MPI_INT,
                       root, comm);}
 else {
   MPI_Gather(&value,              1,  MPI_INT,
                      recv_buf,          1, MPI_INT,
                       root, comm);
}
```

Требуется различать root и не-root

Пример использования In Place в MPI_Reduce

Неверно:

```
double result;
MPI_Reduce(&result,&result,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM
    _WORLD);
```



ОШИБКА

Верно:

```
if (rank==0)
    MPI_Reduce(&result,MPI_IN_PLACE,1,MPI_DOUBLE,MPI_SUM,0,
    MPI_COMM_WORLD);
else
    MPI_Reduce(NULL,&result,1,MPI_DOUBLE,MPI_SUM,0,
    MPI_COMM_WORLD);
```

Вычисление числа π с использованием MPI. Вариант 2.

```
#include "mpi.h"
#include <stdio.h>
int main (int argc, char *argv[])
{
    int n =100000, myid, numprocs, i;
    double mypi, pi, h, sum, x;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    h   = 1.0 / (double) n;
    sum = 0.0;
```

Вычисление числа π с использованием MPI (2)

```
for (i = myid + 1; i <= n; i += numprocs)
  {
              x = h * ((double)i - 0.5);
              sum += (4.0 / (1.0 + x*x));
   }
   mypi = h * sum;
   MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM,
              0, MPI_COMM_WORLD);
   if (myid == 0) printf("pi is approximately
                          %.16f", pi);
   MPI_Finalize();
   return 0;
}
```

Суперкомпьютерное моделирование и технологии

Лекция

Технология параллельного программирования MPI.
Часть 3.

.

Попова Нина Николаевна

доцент кафедры СКИ

popova@cs.msu.su

28 октября 2020 г.

Тема

- Виртуальные топологии MPI
- Исследование эффективности решения задачи Дирихле для уравнения Лапласа

Виртуальная топология MPI

- Во многих параллельных приложениях линейная нумерация процессов не соответствует логической структуре задачи. Часто в задачах процессы упорядочиваются в логические структуры 2D или 3D-решетки. Виртуальные топологии MPI предоставляют удобный механизм именования процессов в группе внутри коммуникатора.
- Необходимо четко различать виртуальную топологию MPI и физическую топологию системы. Виртуальная топология MPI является машинно-независимой характеристикой приложения. Mapping (отображение виртуальной топологии на физическую топологию системы) является машинно-зависимым процессом и выполняется пользователем с помощью изменения настроек на целевой системе. находится вне компетенции MPI (http://www.mpi-forum.org). Удачный выбор виртуальной топологии для заданной физической топологии системы может существенно улучшить выполнение коммуникаций на целевой системе.

Понятие коммуникатора MPI

- Коммуникатор - управляющий объект, представляющий группу процессов, которые могут взаимодействовать друг с другом
- Специальный MPI-тип MPI_Comm

Виртуальные топологии

- Удобный способ именования процессов
- Упрощение написания параллельных программ
- Оптимизация передач
- Возможность выбора топологии, соответствующей логической структуре задачи

Предопределенные коммуникаторы

- MPI-1 поддерживает 3 предопределенных коммуникатора:
  - MPI_COMM_WORLD
  - MPI_COMM_NULL
  - MPI_COMM_SELF
- Только MPI_COMM_WORLD используется для передачи сообщений
- Нужны для реализации ряда функций MPI

Использование MPI_COMM_WORLD

- Содержит все доступные на момент старта программы процессы
- Обеспечивает начальное коммуникационное пространство
- Простые программы часто используют только MPI_COMM_WORLD
- Сложные программы дублируют и производят действия с копиями MPI_COMM_WORLD

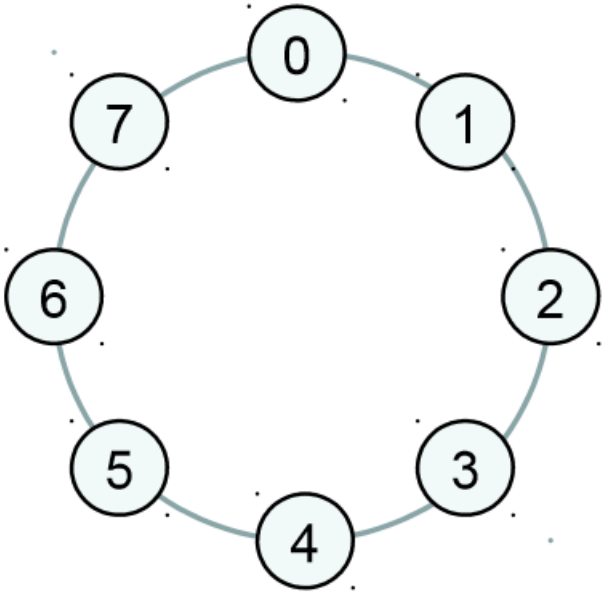
Использование MPI_COMM_NULL

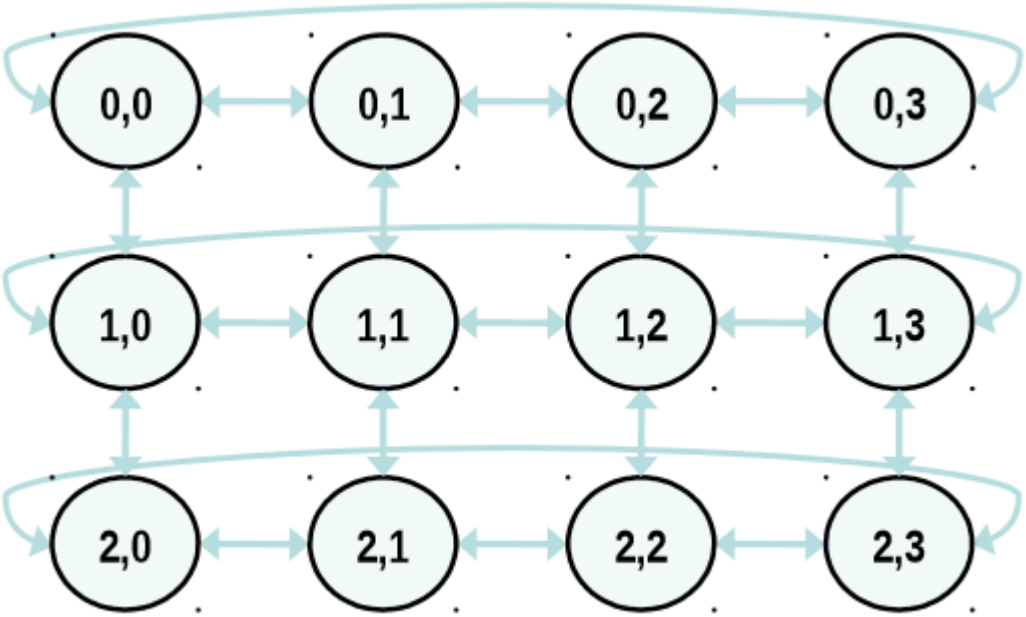
- Нет реализации такого коммуникатора
- Не может быть использован как параметр ни в одной из функций
- Может использоваться как начальное значение коммуникатора
- Возвращается в качестве результата в некоторых функциях
- Возвращается как значение после операции освобождения коммуникатора

Использование MPI_COMM_SELF

- Содержит только локальный процесс
- Обычно не используется для передачи сообщений
- Содержит информацию:
  - кэшированные атрибуты, соответствующие процессу
  - предоставление единственного входа для определенных вызовов

Пример виртуальных топологий





Виртуальные топологии

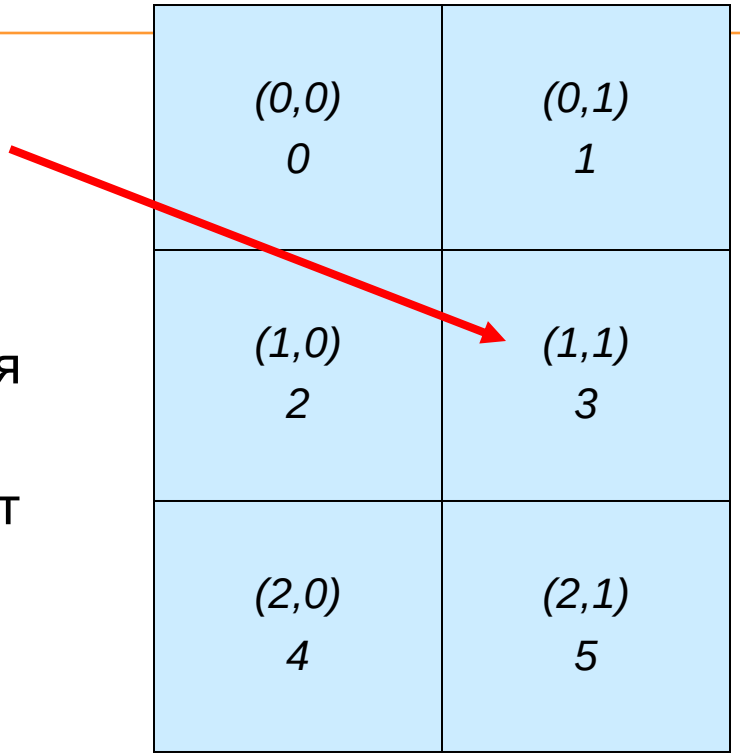
- Основные функции:
  - MPI_CART_CREATE
  - MPI_DIMS_CREATE
  - MPI_CART_COORDS
  - MPI_CART_RANK
  - MPI_CART_SUB
  - MPI_CARTDIM_GET
  - MPI_CART_GET
  - MPI_CART_SHIFT

Как использовать виртуальные топологии

- Создание топологии – новый коммуникатор
- MPI обеспечивает “mapping functions”
- Mapping функции вычисляют ранг процессов, базируясь на топологии

2D решетка

- Отображает линейно упорядоченный массив в 2-мерную решетку (2D Cartesian topology),
- Пример: номер 3 адресуется координатами *(1,1).*
- Каждая клетка представляет элемент 3*x2* матрицы.
- Нумерация начинается с 0.
- Нумерация построчная.



Создание виртуальной топологии решетка

```
int MPI_Cart_create (MPI_Comm   comm_old,
                     int ndims,
                     int *dims,
                     int *periods,
                     int reorder,
                     MPI_Comm *comm_cart)
```

Параметры

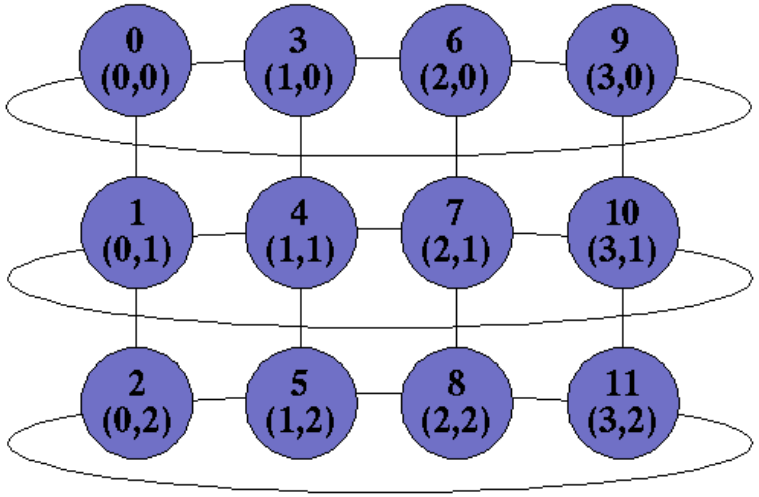
`comm_old`	старый коммуникатор
`ndims`	размерность
`Periods`	логический массив, указывающий на циклическое замыкание: TRUE/FALSE => циклическое замыкание на границе
`reorder`	возможная перенумерация
`comm_cart`	новый коммуникатор

Пример виртуальной топологии решетка

```
MPI_Comm vu;
int dim[2], period[2], reorder;

dim[0]=4; dim[1]=3;
period[0]=TRUE; period[1]=FALSE;
reorder=TRUE;

MPI_Cart_create(MPI_COMM_WORLD,2,
                dim,period,reorder,&vu);
```



Координаты процесса в виртуальной решетке

```
int MPI_Cart_coords (
    MPI_Comm comm,          /* Коммуникатор */
      int rank,             /* Ранг процесса */
       int numb_of_dims,    /* Размер решетки */
       int   coords[]       /* координаты процесса в решетке */
     )
```

MPI_CART_RANK

*int **MPI_Cart_rank**(MPI_Comm comm, int *coords, int *rank)*

Перевод логических координат процесса в решетке в ранг процесса.

- Если i-ое направление размерности периодическое и i-ая координата выходит за пределы, значение автоматически сдвигается *0<coords(i)<dims(i).*
- В противном случае - ошибка

Определение сбалансированного распределения процессов по решетке

int **MPI_Dims_create** (int nnodes, int ndims, int *dims)

nnodes - число процессов

ndims - размер решетки

dims - число элементов по измерениям решетки

- Помогает определить сбалансированное распределение процессов по измерениям решетки.
- Если dims[i] – положительное целое, это измерение не будет модифицироваться

dims before call	Function call	dims on return
(0, 0)	MPI_DIMS_CREATE(6, 2, dims)	(3, 2)
(0, 0)	MPI_DIMS_CREATE(7, 2, dims)	(7, 1)
(0, 3, 0)	MPI_DIMS_CREATE(6, 3, dims)	(2, 3, 1)
(0, 3, 0)	MPI_DIMS_CREATE(7, 2, dims)	erroneous call

Пример использования MPI_Dims_create

```
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);

int dim[3];
dim[0] = 0; // let MPI arrange
dim[1] = 0; // let MPI arrange
dim[2] = 3; // I want exactly 3 planes

MPI_Dims_create(nprocs, 3, dim);

if (dim[0]*dim[1]*dim[2] < nprocs) {
  fprintf(stderr, "WARNING: some processes are not in use!\n"
}

int period[] = {1, 1, 0};
int reorder = 0;

MPI_Cart_create(MPI_COMM_WORLD, 3, dim, period, reorder,
&cube_comm);
```

MPI_CART_SHIFT

- Получение номеров посылающего (**`source`**) и принимающего (**`dest`**) процессов в декартовой топологии коммуникатора **`comm`** для осуществления сдвига вдоль измерения **`direction`** на величину **`disp`**.

```
int MPI_Cart_shift( MPI_Comm comm, int direction, int displ, int
    *source, int *dest )
```

MPI_CART_SHIFT

int **MPI_Cart_shift**(MPI_Comm comm, int direction, int displ, int *source, int *dest)

comm - коммуникатор с декартовой топологией;

direction - измерение, вдоль которого выполняется сдвиг;

disp - величина сдвига (может быть как положительной, так и отрицательной; >0 – сдвиг влево/вверх, <0 – сдвиг вправо/вниз)

source - номер процесса, от которого должны быть получены данные;

dest - номер процесса, которому должны быть посланы данные.

MPI_CART_SHIFT

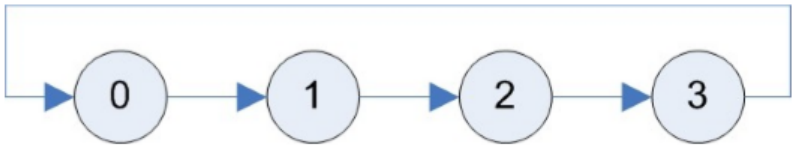
Для периодических измерений осуществляется циклический сдвиг, для непериодических – линейный сдвиг.

Для **`n`**-мерной декартовой решетки значение **`direction`** должно быть в пределах от **`0`** до **`n-1`**.

Значения **`source`** и **`dest`** можно использовать, например, для обмена функцией **`MPI_Sendrecv`**.

В случае линейного сдвига в качестве source или dest можно использовать MPI_PROC_NULL.

Пример: Sendrecv в 1D решетке



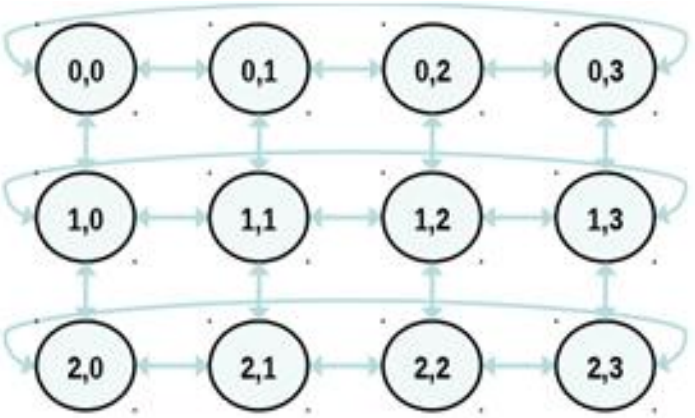

```
...
int dim[1],period[1];
dim[0] = nprocs;
period[0] = 1;
MPI_Comm ring_comm;

MPI_Cart_create(MPI_COMM_WORLD, 1, dim, period, 0, &ring_comm);

int source, dest;
MPI_Cart_shift(ring_comm, 0, 1, &source, &dest);

MPI_Sendrecv(right_bounday, n, MPI_INT, dest, rtag,
             left_boundary, n, MPI_INT, source, ltag,
             ring_comm, &status);
...
```

Пример: Sendrecv в 2D решетке




```
…
int dim[] = {4,3};
int period [] = {1,0};
int source, dest;

MPI_Comm grid_comm;

MPI_Cart_create (MPI_COMM_WORLD,2,
    dim, period, 0 , &grid_comm);
for (int dimension = 0; dimension <2;  dimension++) {
   for ( int versus = -1; versus < 2; versus+=2) {
      MPI_Cart_shift (grid_comm, dimension, versus, &source, &dest);
      MPI_Sendrecv( buffer, n, MPI_INT, source, stag,
            buffer, n, MPI_INT, dest, dtag,
            grid_comm, &status);
   }
}
```

Создание подрешетки

int **MPI_Cart_sub** (MPI_Comm comm_old,
int remain_dims[], MPI_Comm *new_comm)

comm_old – старый коммуникатор

i-ый элемент в remain_dims показывает, содержится ли i-ая размерность в подрешетке (true) или нет (false) (вектор логических элементов)

new_comm – новый коммуникатор

```
int dim[] = {2,3,4};
int remain_dims[]={1,0,1}; // 3 comm with 2x4 processes 2D grid
…
int remain_dims[]={0,0,1}; // 6 comm with 4 processes 1D topology
```

MPI_CARTDIM_GET

- Определение числа измерений в решетке.

  int MPI_Cartdim_get(MPI_Comm comm, int* ndims)

- comm коммуникатор (решетка)
- ndims число измерений. Возвращаемый параметр.

Исследование эффективности решения задачи Дирихле для уравнения Лапласа

- Задача Дирихле для уравнения Лапласа (1).

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} + \dfrac{\partial^2 u}{\partial z^2} = 0, (x, y, z) \in D, \\ u(x, y, z) = g(x, y, z), (x, y, z) \in D^0, \end{cases} \quad (1)$$

где $u(x, y, z)$ - функция, удовлетворяющая в области D уравнению Лапласа и принимающая на границе D^0 области D значения $g(x, y, z)$.

Подобная модель может применяться для описания установившегося течения жидкости, стационарных тепловых полей, процессов теплопередачи с внутренними источниками тепла и деформации упругих пластин.

Постановка задачи. Разностная схема.

- Численный подход к решению задачи (1) основан на замене производных соответствующими конечными разностями (2).

$$\frac{u_{i+1,j,k}-2u_{i,j,k}+u_{i-1,j,k}}{h^2}+\frac{u_{i,j+1,k}-2u_{i,j,k}+u_{i,j-1,k}}{h^2}+\frac{u_{i,j,k+1}-2u_{i,j,k}+u_{i,j,k-1}}{h^2}=0, \quad (2)$$

где $h \geq 0$ *- шаг сетки,* $u_{i,j,k}$ *- значение функции u(x, y, z) в точке* $x = x_i = ih, i = \overline{0, M+1}, y = y_j = jh, j = \overline{0, N+1}, z = z_k = kh, h = \overline{0, L+1},$ *где* M, N, L *- количество внутренних узлов по каждой координате в области* D.

Метод Якоби.

- Одним из простейших методов решения полученной системы (2) является итерационный метод Якоби (3).

$$u_{i,j,k}^{n} = (u_{i-1,j,k}^{n-1} + u_{i+1,j,k}^{n-1} + u_{i,j+1,k}^{n-1} + u_{i,j-1,k}^{n-1} + u_{i,j,k+1}^{n-1} + u_{i,j,k-1}^{n-1})/6$$

$$u_{i,j,k}^{n} = g_{i,j,k}, (x,y,z) \in D^{0}, n = 1,2,.. \qquad (3)$$

где n - номер итерации.

Метод Якоби – тестовая задача

- Метод Якоби решения разностной задачи для уравнения Лапласа – это типичная **тестовая задача** в параллельном программировании. Он включает в себя общие шаблоны для большинства параллельных задач.

- Метод гарантирует однозначность результатов независимо от способа распараллеливания, но требует использования дополнительного объема памяти. Из-за медленной сходимости метода на практике ему обычно предпочитают более быстрые.

Уравнение Лапласа. 2D вариант.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \qquad x, y \in [0,1] \qquad (1)$$

Краевые условия:

$$u(x,0) = \sin(\pi x) \qquad 0 \le x \le 1$$
$$u(x,1) = \sin(\pi x)e^{-x} \qquad 0 \le x \le 1 \qquad (2)$$
$$u(0,y) = u(1,y) = 0 \qquad 0 \le y \le 1$$

Аналитическое решение:

$$u(x,y) = \sin(\pi x)e^{-xy} \qquad x, y \in [0,1] \qquad (3)$$

Дискретизация уравнения Лапласа

$$u_{i,j}^{n+1} \cong \frac{u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}}{4} \qquad i = 1,2,\text{K},m; \;\; j = 1,2,\text{K},m \qquad (4)$$

где *n* и *n+1* текущий и следующий шаг,

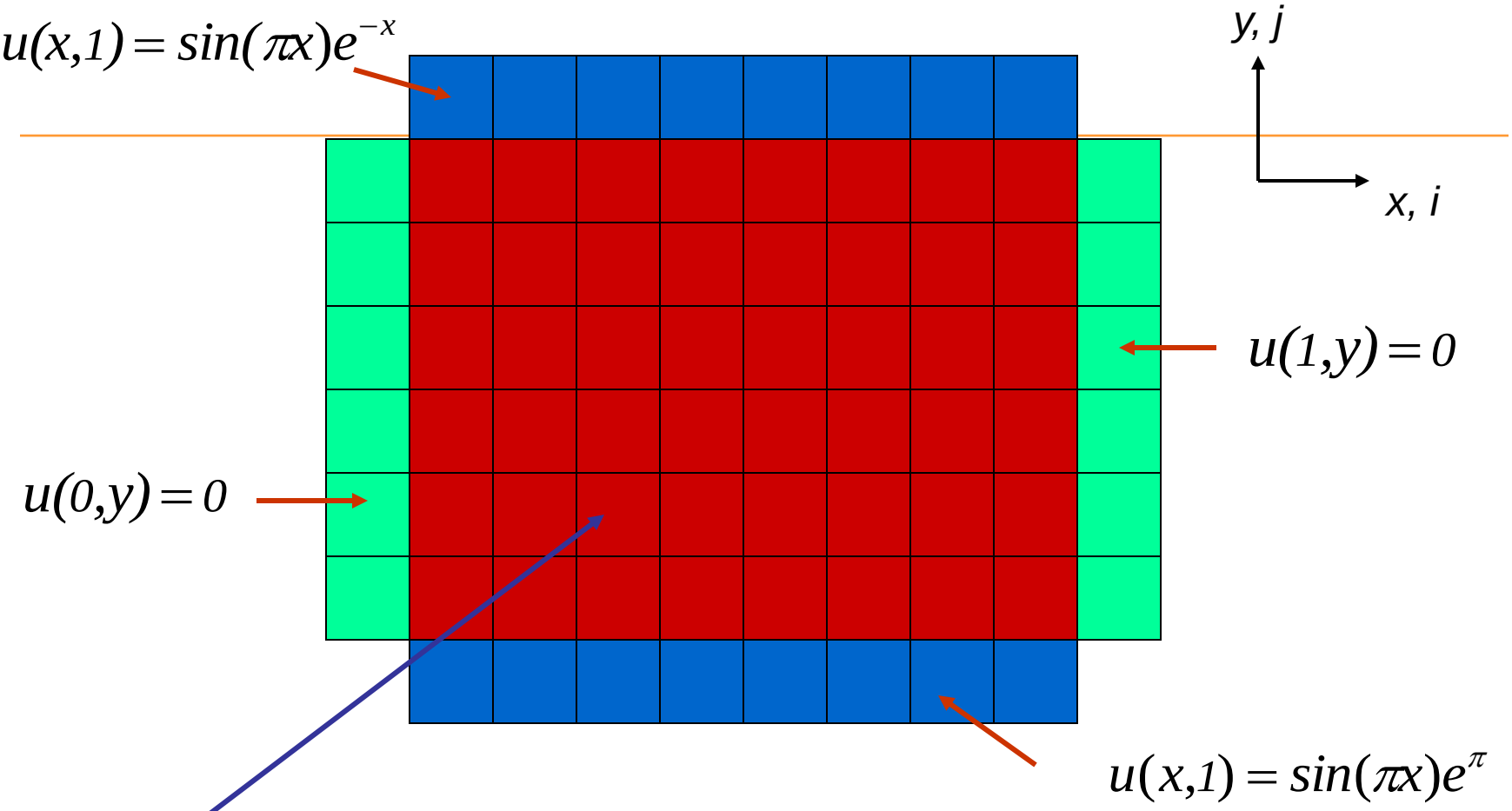
$$u_{i,j}^{n} = u^{n}(x_i, y_j) \qquad i = 0,1,2,\text{K},m+1; \;\; j = 0,1,2,\text{K},m+1 \qquad (5)$$

$$= u^{n}(i\Delta x, j\Delta y)$$

Для простоты

$$\Delta x = \Delta y = \frac{1}{m+1}$$

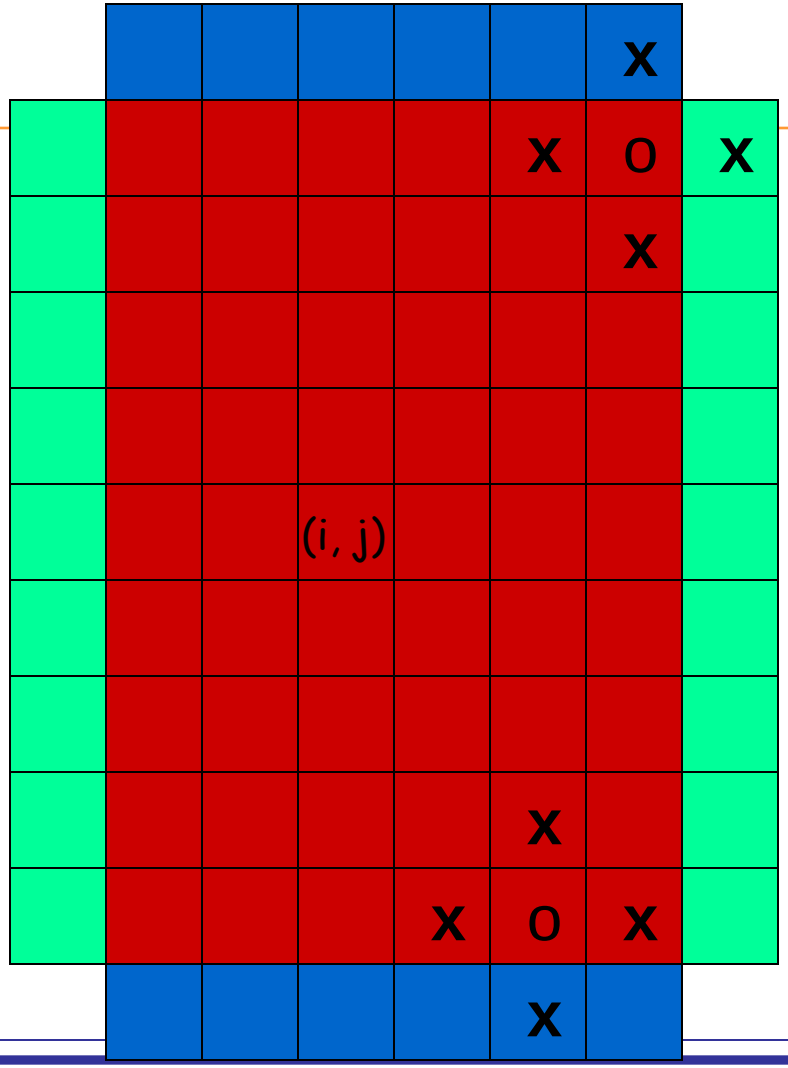
Вычислительная область



$$u_{i,j}^{n+1} \cong \frac{u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}}{4} \qquad i = 1,2,\text{K},m; \quad j = 1,2,\text{K},m$$

5-точечный шаблон

$$u_{i,j}^{n+1} \cong \frac{u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}}{4}$$



внутренняя область, на которой ищется решение уравнения

граничная область.

Голубые клетки – неоднородные граничные условия,

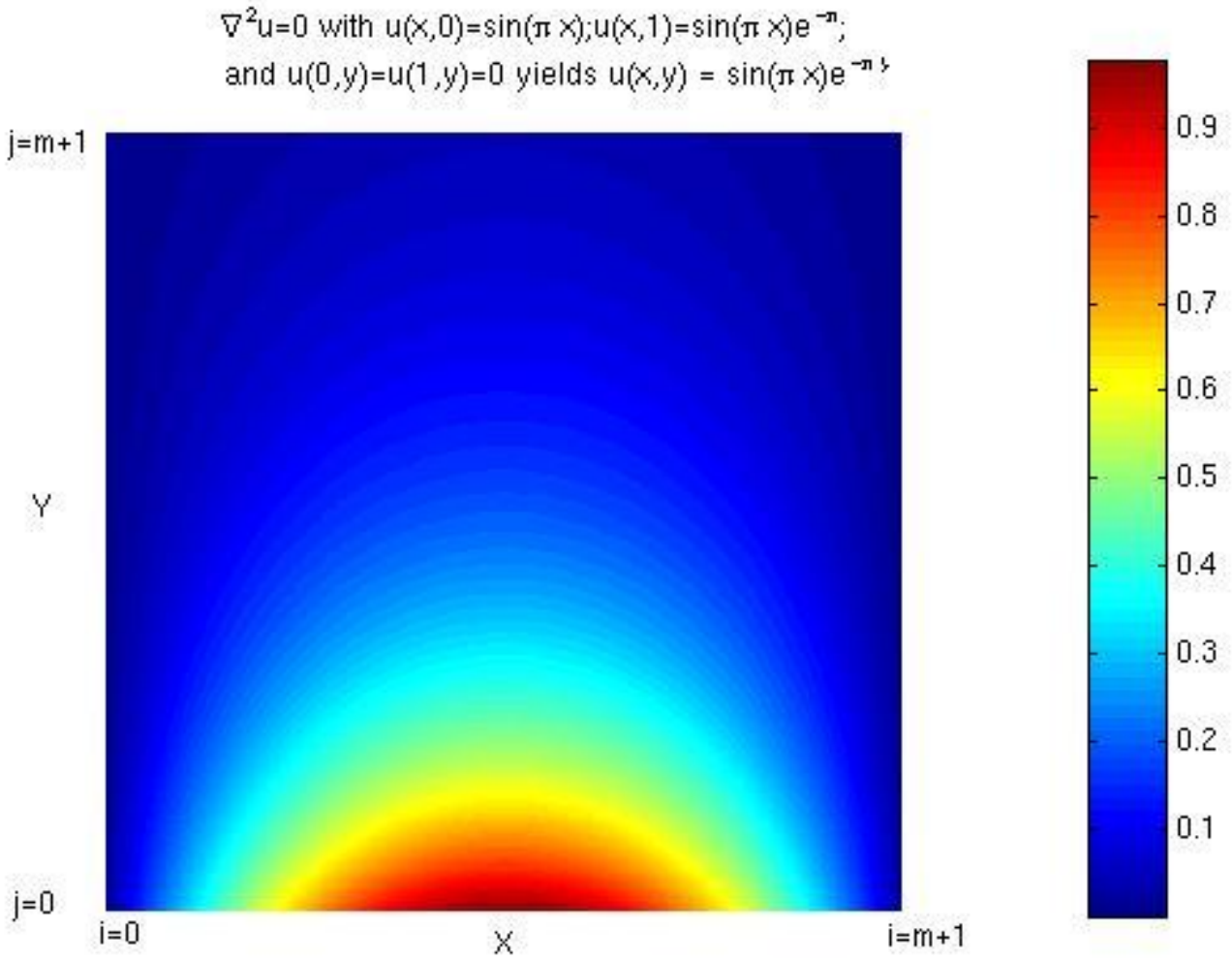
Зеленые - однородные

Схема метода Якоби

1. Задание начальных значений *и во всех внутренних точках (i,j) в момент времени n=0*.
2. Используя 5-точечный шаблон вычислить значения во внутренних точках $u_{i,j}^{n+1}$ *(i,j)*.
3. Завершить процесс, если заданная точность достигнута.
4. Иначе: $u_{i,j}^{n} = u_{i,j}^{n+1}$ для всех внутренних точек.
5. Перейти на шаг 2.

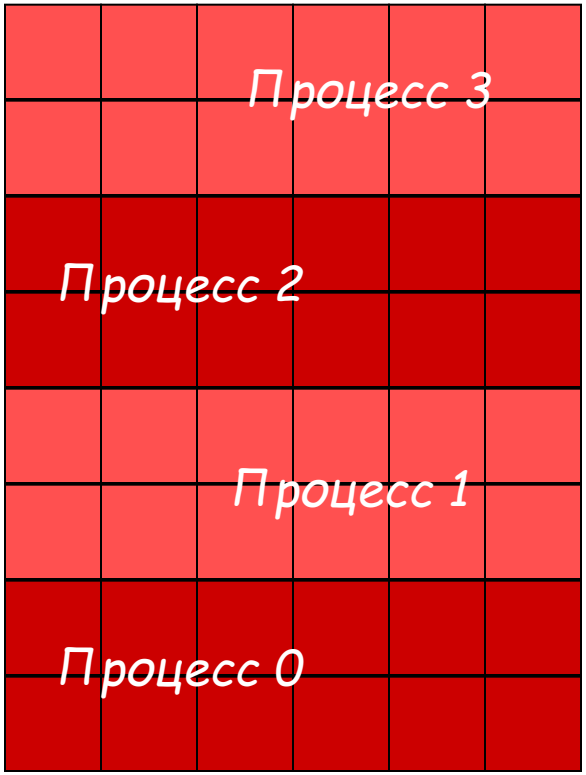
Это очень простая схема. Медленно сходится, поэтому не используется для решения реальных задач.

Решение (линии уровня)

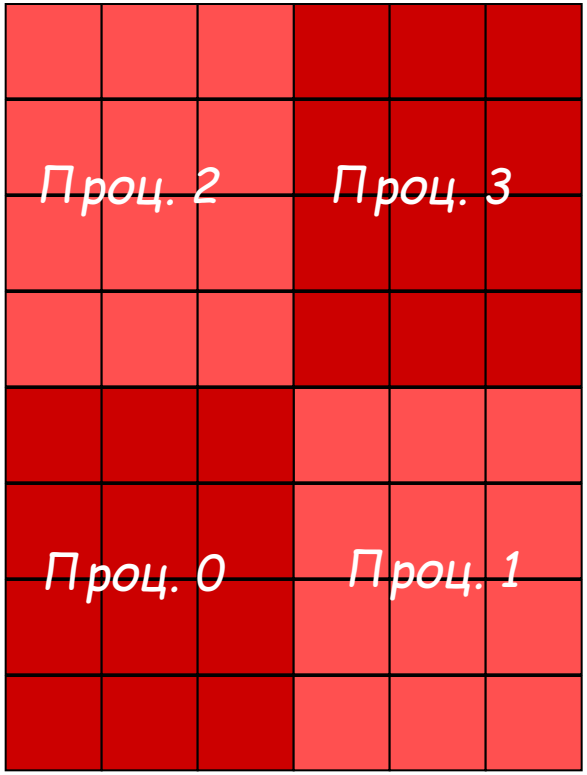


Параллельная реализация

1D Domain Decomposition



2D Domain Decomposition



Распределение данных – 1D

5-точечная схема требует значений от соседних потоков

Необходим обмен данными на границах области

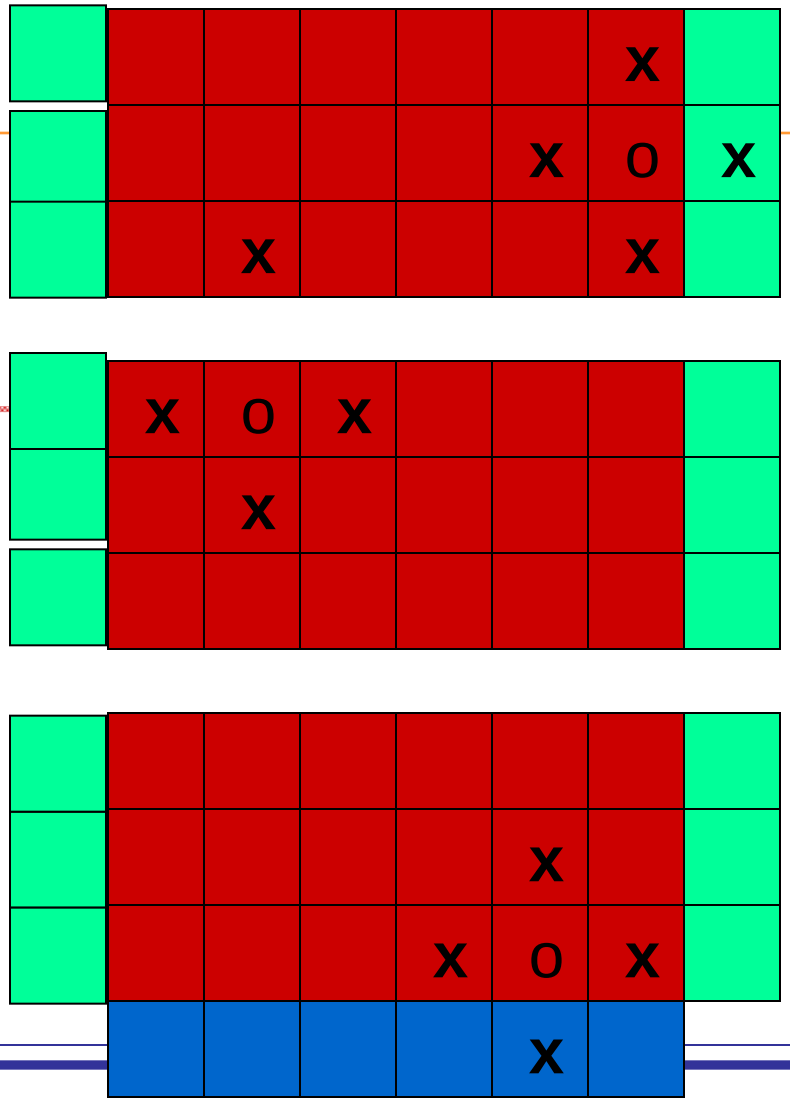
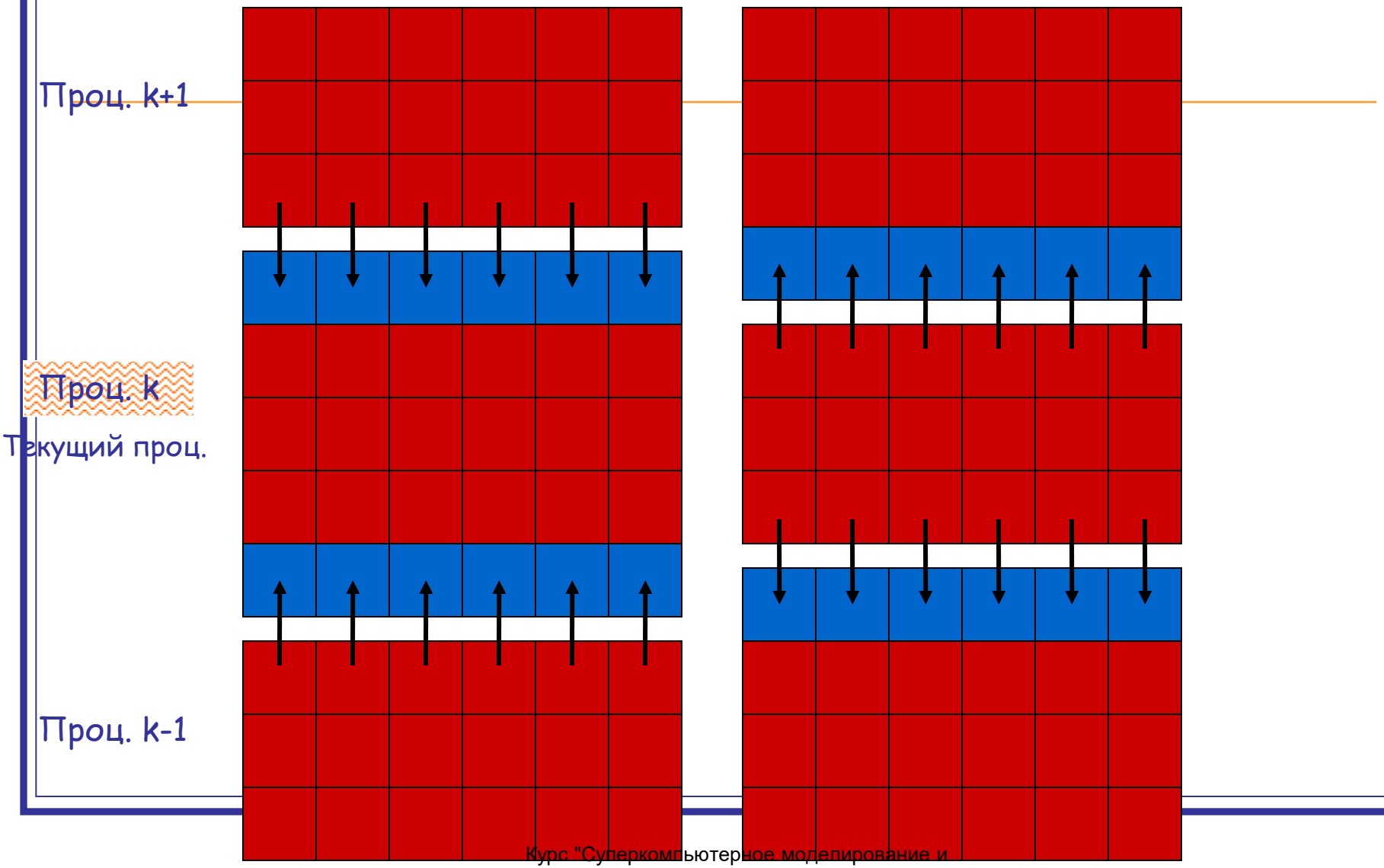


Схема передачи данных



3D реализация алгоритма.

- В алгоритме используются два буфера с данными. Один - для значений, полученных на предыдущей итерации, другой - для значений вычисляемых на их основе на новой итерации.
- Перед выполнением алгоритма нужно записать в один из буферов начальное приближение и установить порядок буферов. На каждой итерации алгоритма вычисляется погрешность Δ. Алгоритм останавливается, когда Δ становится меньше некоторого числа или после выполнения заданного числа итераций.
- На процессор, выполняющий обработку какого-либо блока данных, должны быть продублированы боковые грани соседних блоков. Дублирование осуществляется перед началом выполнения каждой очередной итерации метода.

Структура алгоритма.

```
// действия, выполняемые на
каждом процессоре
do {
  // обмен границами с соседями
  // обработка блока
  // вычисление общей
  //  погрешности вычислений
 while (Δ < eps );
```

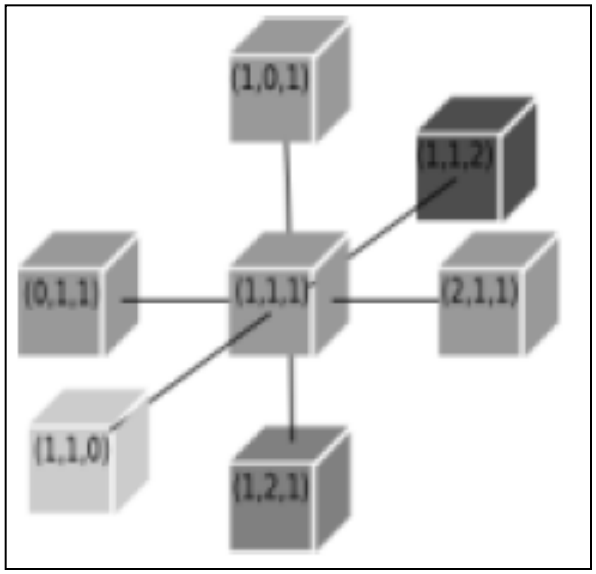
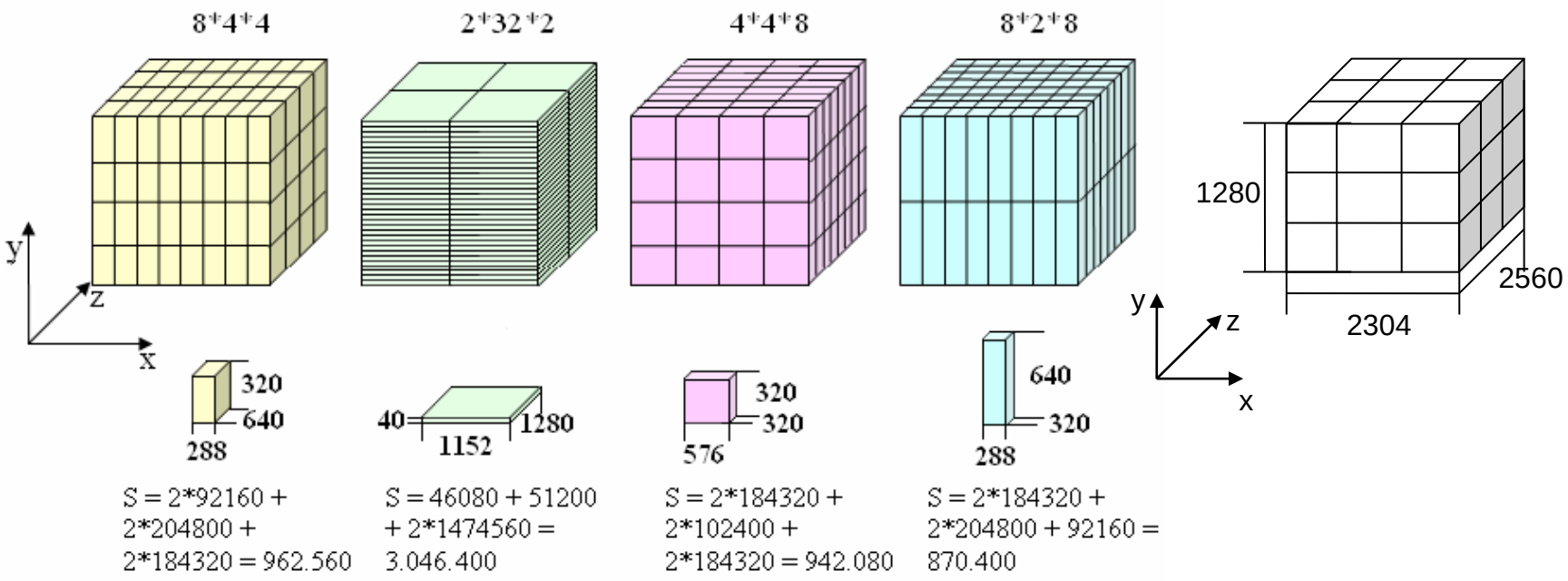


Схема обмена данными: направления взаимодействия между соседними процессами.

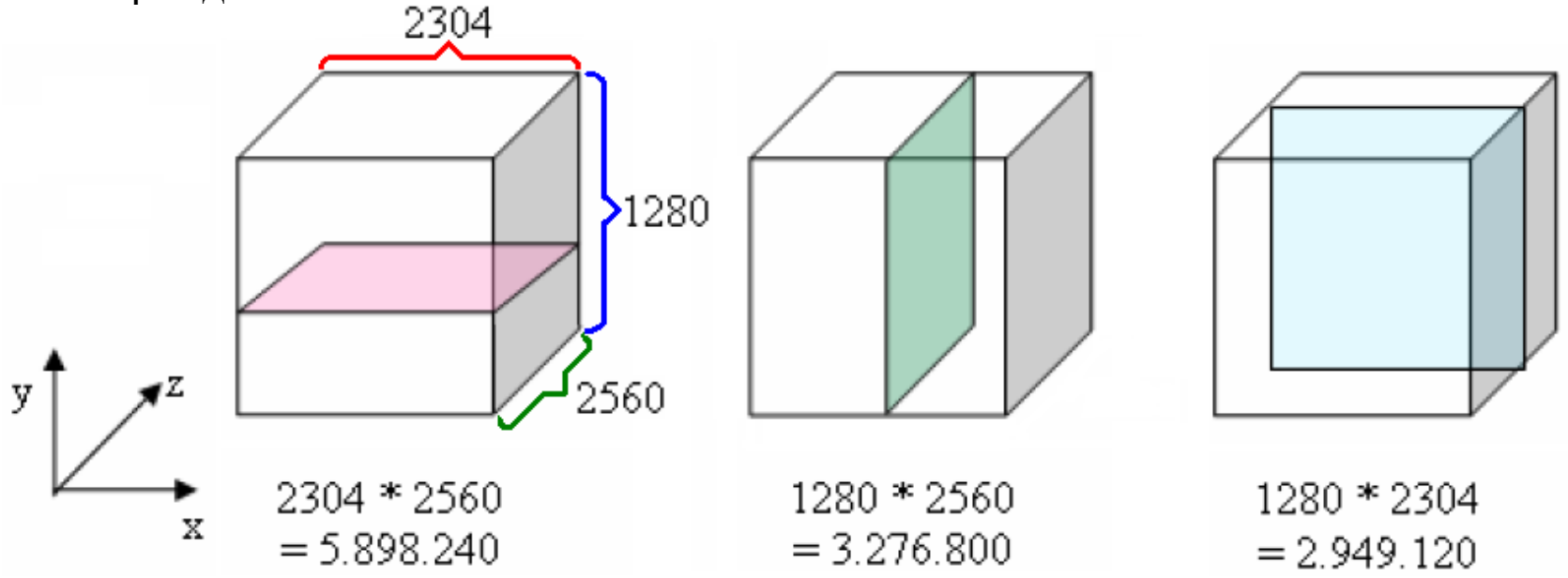
Объем коммуникаций при различных виртуальных топологиях MPI.

- В задаче виртуальная топология влияет, в первую очередь, на объем пересылаемых процессами сообщений, который определяется площадью боковых граней. Размер задачи (2304*1280*2560).



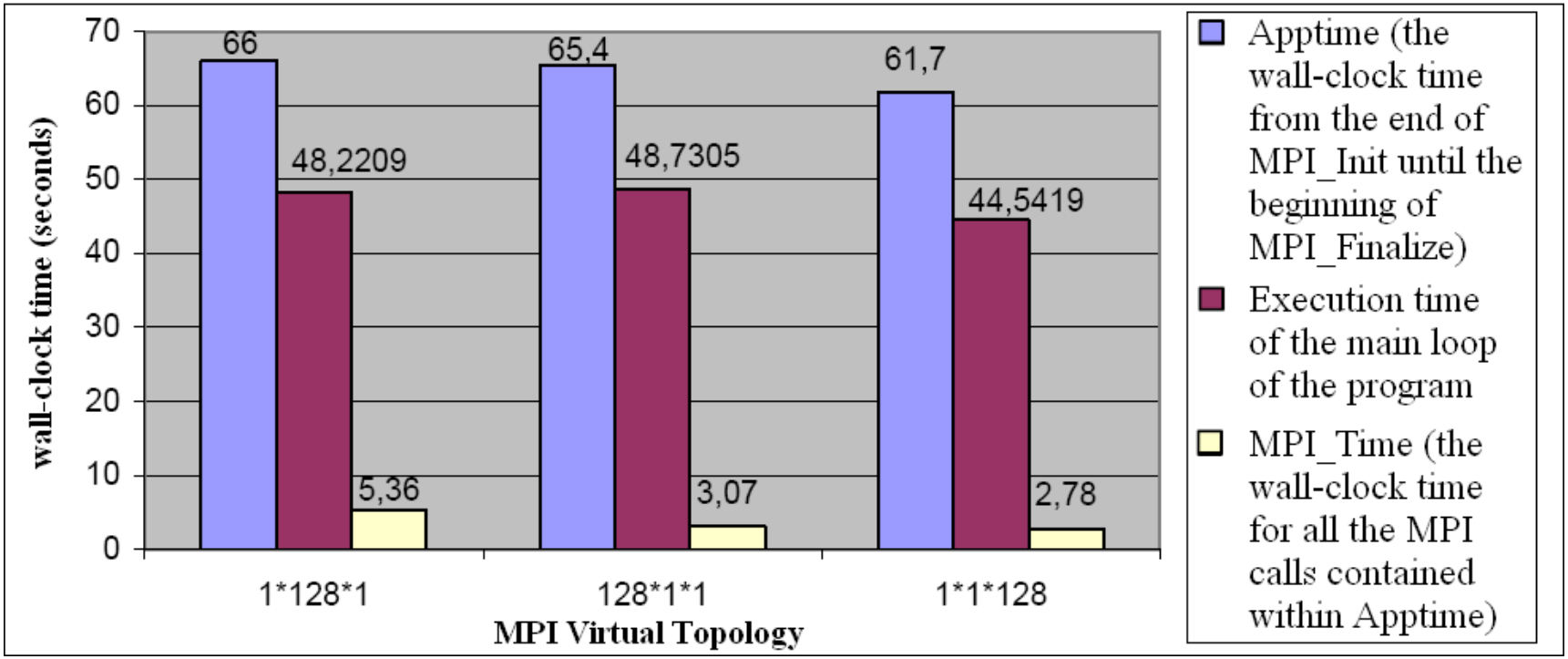
Результаты экспериментов. Распределение данных полосами.

Результаты тестов на партиции из **128** вычислительных узлов (**512 ядер**) BGP.
Размер задачи 2304*1280*2560 элементов типа double.



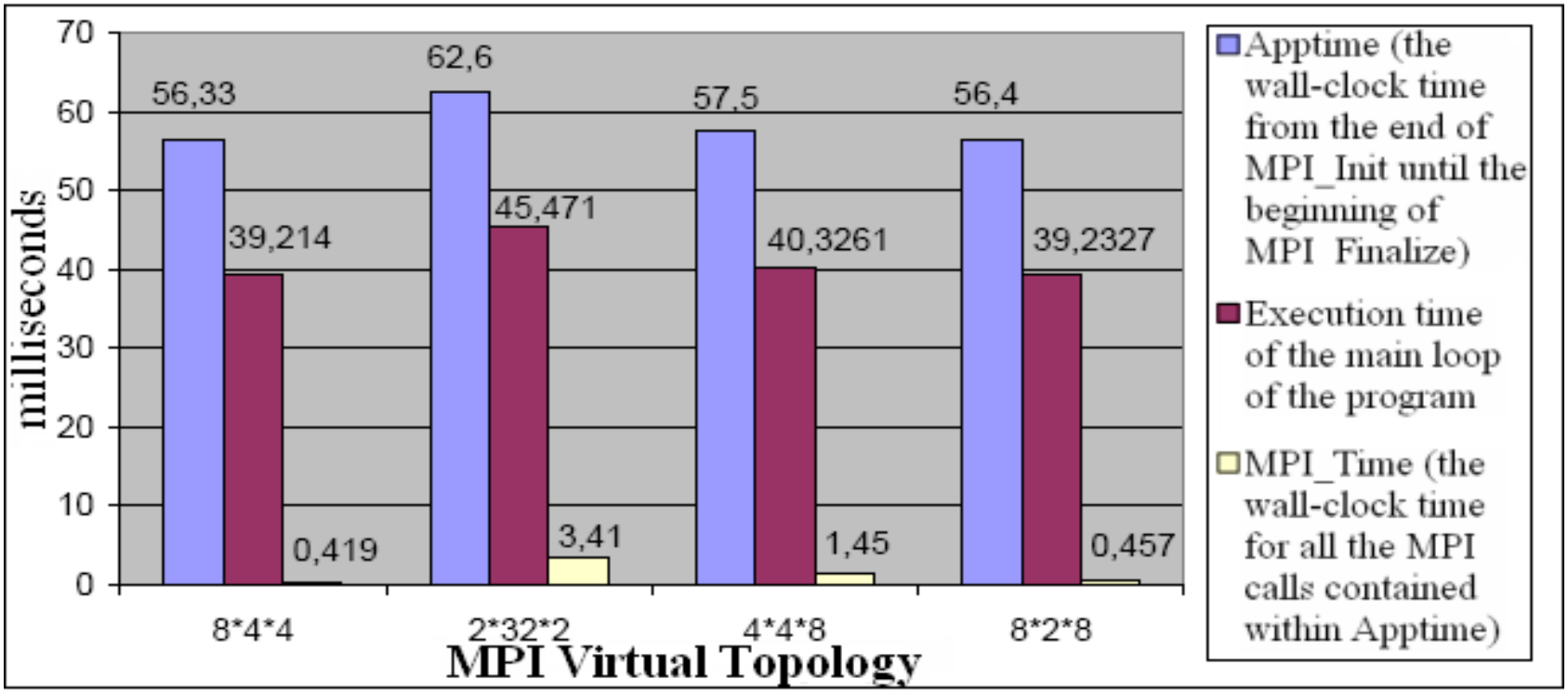
Размер одного сообщения - теневой грани (количество чисел типа double) при различных используемых виртуальных топологиях одномерной решетки.

Результаты экспериментов. Распределение данных полосами.



Среднее время выполнения **для различных виртуальных топологий** в режиме **SMP** на **128 вычислительных узлах** при распределении данных полосами.

Результаты экспериментов. Распределение данных блоками.



Среднее время выполнения для различных виртуальных топологий в режиме **SMP** на **128 вычислительных узлах** при распределении данных блоками.

Фрагменты программы. struct Matrix3.

```
struct Matrix3
{
        int size_x, size_y, size_z;
        double * data;

        Matrix3 (int m, int n, int k);
        /*индекс в одномерном массиве*/
        int ind (int i, int j, int k);
        double & el(int i, int j, int k);
        void print();
        void fill();
        ~Matrix3();
};
```

Фрагменты программы. struct Process.

```
struct Process
{
    Process(int size_x, int size_y, int size_z, int proc_number[], double eps);
    int jacobi_method();//метод Якоби
    void PrintAllData();
    int size, rank;

private:
    MPI_Comm cart_comm;
    int proc_number[3]; //количество процессов в каждом измерении
    int coords[3];
    double eps; //точность
    Matrix3 block1, block2;//текущее значение блока и буфер с предыдущим
    int current_block;//куда записывается значение новой итерации
```

Фрагменты программы. struct Process

```
    Matrix2 up, down, left, right, front, back; //границы или теневые грани
    //буфера для формирования сообщений
    Matrix2 buf_up, buf_down, buf_left, buf_right;
    omp_lock_t dmax_lock;
    double jacobi_iteration();//итерация метода Якоби
    void CreateCommunicator();
    void FillData();
    void ExchangeData();
};

void Process :: CreateCommunicator()
{
        int Periods[3] = {0, 0, 0};     //коммуникатор - трехмерная решетка
        MPI_Cart_create(MPI_COMM_WORLD, 3, proc_number, Periods, 0,
                        &cart_comm);
        MPI_Cart_coords(cart_comm, rank, 3, coords);
}
```

Фрагменты программы. main.

```
int main(int argc, char * argv[]) {
    int data_size[3];
    int block_number[3];
    double eps;
    AnalyseCommandString(argc, argv, data_size, block_number, eps);
    MPI_Init(&argc, &argv);

    Process P(data_size[0]/block_number[0], data_size[1]/block_number[1],
                        data_size[2]/block_number[2], block_number, eps);
    P.PrintAllData();
    double time1, time2;

    time1 = MPI_Wtime();
    P.jacobi_method();
    time2 = MPI_Wtime();

    P.PrintAllData();
    if (P.rank == 0) cout << time2 - time1 << endl;
    MPI_Finalize(); }
```

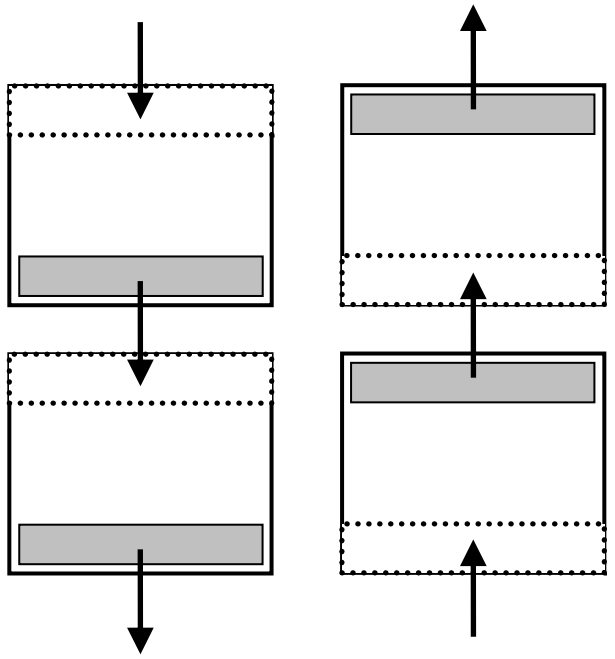
Фрагменты программы. jacobi_method()

```
int Process :: jacobi_method() {
    double delta, delta_local;
    int i = 0;
    do {
            i++;
            current_block = i%2 + 1;
            ExchangeData(); // обмен теневыми гранями
            delta_local = jacobi_iteration();
            // выполнение редукционной операции
            MPI_Allreduce(&delta_local, &delta, 1, MPI_DOUBLE,
            MPI_SUM, MPI_COMM_WORLD);
            delta = sqrt(delta);
            current_block = (current_block == 1) ? BLOCK2 : BLOCK1;
    }
    while(delta > eps);
    if (rank == 0) cout <<" delta = " << delta << endl;
    return I; }
```

Обмен данными в программе

- Процессу необходимо обменяться данными с 6 соседями.
- Из-за того что данные хранятся в памяти не непрерывно, для выполнения межпроцессорных обменов предварительно формируются передаваемые сообщения.

  Потом при помощи функций MPI_Cart_shift происходит определение ранга процесса, которому надо послать данные, и процесса, от которого данные необходимо получить. После чего при помощи MPI_Sendrecv - обмен данными с соседними процессами.
- Всего происходит 6 обменов, по два на каждое измерение.



Обмен данными вдоль одного измерения

Фрагменты программы. Обмен данными - влево вдоль оси X.

```
MPI_Cart_shift(cart_comm, 0, -1, &rank_recv, &rank_send);
if (coords[0] != 0 && coords[0] != proc_number[0] - 1) {
        MPI_Sendrecv(buf_left.data, size_y * size_z, MPI_DOUBLE,
        rank_send, TAG_X, right.data, size_y * size_z, MPI_DOUBLE,
        rank_recv, TAG_X, cart_comm, &status);
}
else if (coords[0] == 0 && coords[0] != proc_number[0] – 1){
        MPI_Recv(right.data, size_y * size_z, MPI_DOUBLE, rank_recv,
TAG_X, cart_comm, &status);
}
else if (coords[0] == proc_number[0] - 1 && coords[0] != 0){
        MPI_Send(buf_left.data, size_y * size_z, MPI_DOUBLE,
        rank_send, TAG_X, cart_comm);
}
```

Пример командного (.jcf) файла.

```
#@ job_type = bluegene
#@ class = n2048_m30
#@ initialdir = /gpfs/data/kaelin/jacobi
#@ error = jacobi_p.$(jobid).err
#@ output = jacobi_p.$(jobid).out
#@ wall_clock_limit = 00:30:00
#@ bg_size = 2048
#@ queue
/bgsys/drivers/ppcfloor/bin/mpirun  -exe
/gpfs/data/kaelin/jacobi/jac_Hdebug -env OMP_NUM_THREADS=4 -
mode SMP -mapfile XYZ -args "9216 2560 5120 32 8 8 5"
```

llsubmit test.jcf постановка задачи в очередь

llq просмотреть очередь заданий

llcancel fen1.157.0 удалить задание из очереди

Профилирование. Библиотека mpiP. (http://mpip.sourceforge.net)

Для профилирования использована библиотека **mpiP**. Она позволяет узнать:

- *mapping*
- *процент от времени выполнения программы, который заняли MPI вызовы*
- *время выполнения отдельных MPI-вызовов, его разброс для различных процессоров и нескольких выполнений одного вызова*
- *статистику для размеров сообщений, передаваемых MPI-вызовами.*

- Использование библиотеки mpiP занимает очень небольшой процент от времени работы приложения, вся информация об MPI-вызовах собирается локально в рамках отдельных MPI-процессов и собирается только на последнем этапе работы приложения для формирования общего отчета.

Файл вывода библиотеки MpiP

```
@ mpiP
@ Command : /gpfs/data/kaelin/jacobi/jac_Hdebug 2304 1280 2560 1 1 128 5
@ Version                  : 3.1.2
@ MPIP Build date          : Mar  9 2009, 22:17:50
@ Start time               : 1970 01 01 00:00:40
@ Stop time                : 1970 01 01 00:01:42
@ Timer Used               : gettimeofday
@ MPIP env var             : [null]
@ Collector Rank           : 0
@ Collector PID            : 100
@ Final Output Dir         : .
@ Report generation        : Single collector task
@ MPI Task Assignment      : 0 Rank 0 of 128 <0,0,0,0>  R01-M1-N08-J23
@ MPI Task Assignment      : 1 Rank 1 of 128 <1,0,0,0>  R01-M1-N08-J04
@ MPI Task Assignment      : 2 Rank 2 of 128 <2,0,0,0>  R01-M1-N08-J12
@ MPI Task Assignment      : 3 Rank 3 of 128 <3,0,0,0>  R01-M1-N08-J31
```

Файл вывода библиотеки MpiP

```
---------------------------------------
@--- MPI Time (seconds) ---------------
---------------------------------------
Task    AppTime    MPITime     MPI%
   0       61.7       2.78     4.50
   1       61.7       2.78     4.50
   2       61.7       2.78     4.50
   3       61.7       2.78     4.50
```

AppTime – общее время работы приложения
MPITime – время, которое заняли MPI-вызовы

Вызовы MPI-функций:

```
---------------------------------------------------------------
@--- Callsites: 34 ---------------------------------------------
---------------------------------------------------------------
 ID Lev File/Address        Line Parent_Funct             MPI_Call
  1   0 0x01002517                [unknown]               Recv
  2   0 0x01002477                [unknown]               Send
  3   0 0x010029db                [unknown]               Allreduce
```

Файл вывода библиотеки MpiP

```
---------------------------------------------------------------
@- Aggregate Time (top twenty, descending, milliseconds)--
---------------------------------------------------------------
Call                 Site       Time     App%     MPI%      COV
Sendrecv               31   1.6e+05     2.02    44.92     0.02
Sendrecv               32  1.59e+05     2.02    44.83     0.01
Allreduce              11  1.48e+04     0.19     4.16     0.21
```

20 MPI-вызовов, занявших наибольшее суммарное (сумма - по всем вызовам и процессам время).

```
---------------------------------------------------------------
@- Aggregate Sent Message Size (top twenty, descending, bytes)
---------------------------------------------------------------
Call                 Site       Count       Total       Avrg  Sent%
Sendrecv               32        2520   5.95e+10   2.36e+07  49.61
Sendrecv               31        2520   5.95e+10   2.36e+07  49.61
```

20 MPI-вызовов, передавших наибольший суммарный объем сообщений (сумма - по всем вызовам и процессам).

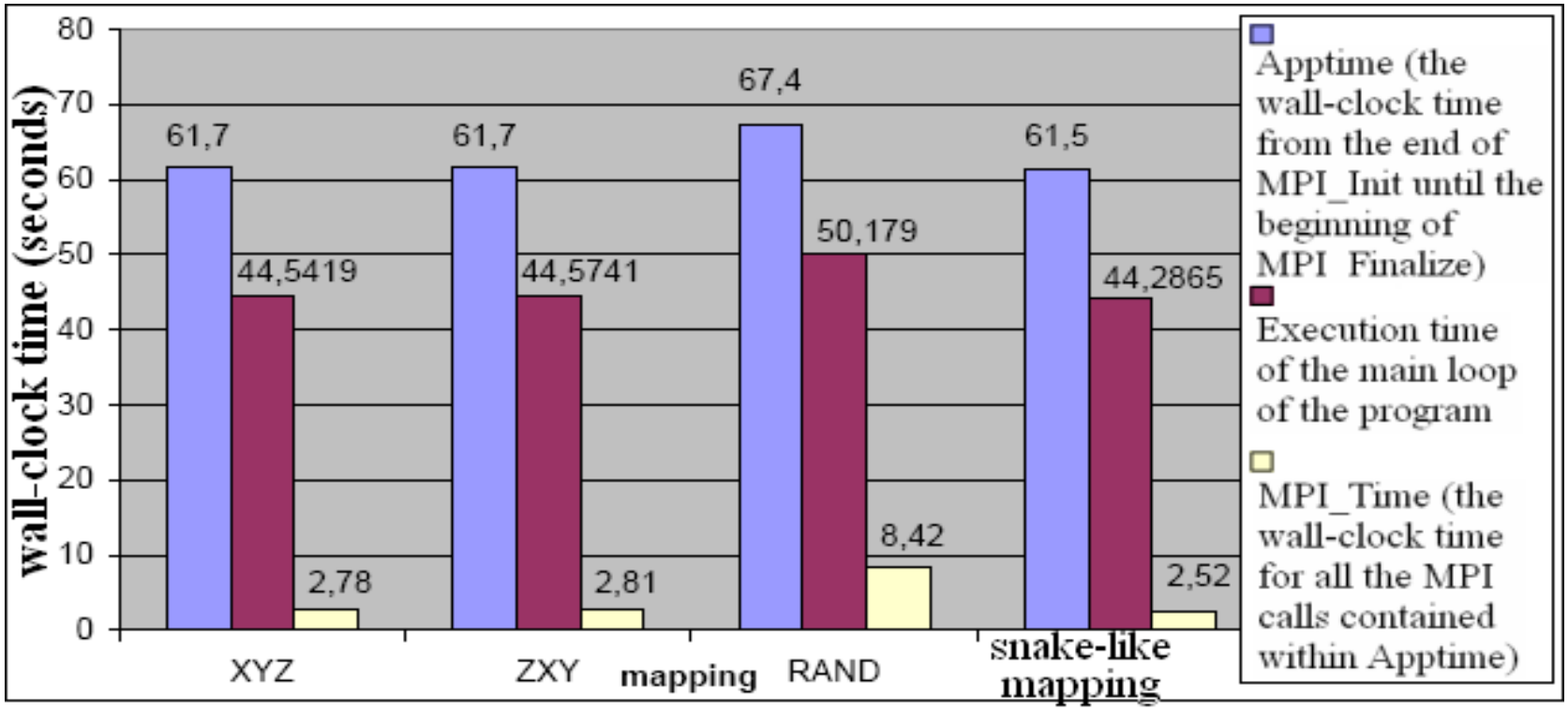
Файл вывода библиотеки MpiP

Статистика для времени отдельных MPI-вызовов (аналогичная таблица есть для размеров сообщений).

```
---------------------------------------------------------------
@--- Callsite Time statistics (all, milliseconds): 3840 --
---------------------------------------------------------------
Name      Site Rank  Count    Max  Mean    Min    App%    MPI%
Sendrecv    32    1     20   63.1    63     63    2.04   45.35
Sendrecv    32    2     20   63.7    63     63    2.04   45.35
Sendrecv    32    3     20     63    63     63    2.04   45.33
Sendrecv    32    4     20   63.1    63     63    2.04   45.34
.....................................................................
Sendrecv    32    *   2520    187  63.3   62.9    2.02   44.83
```

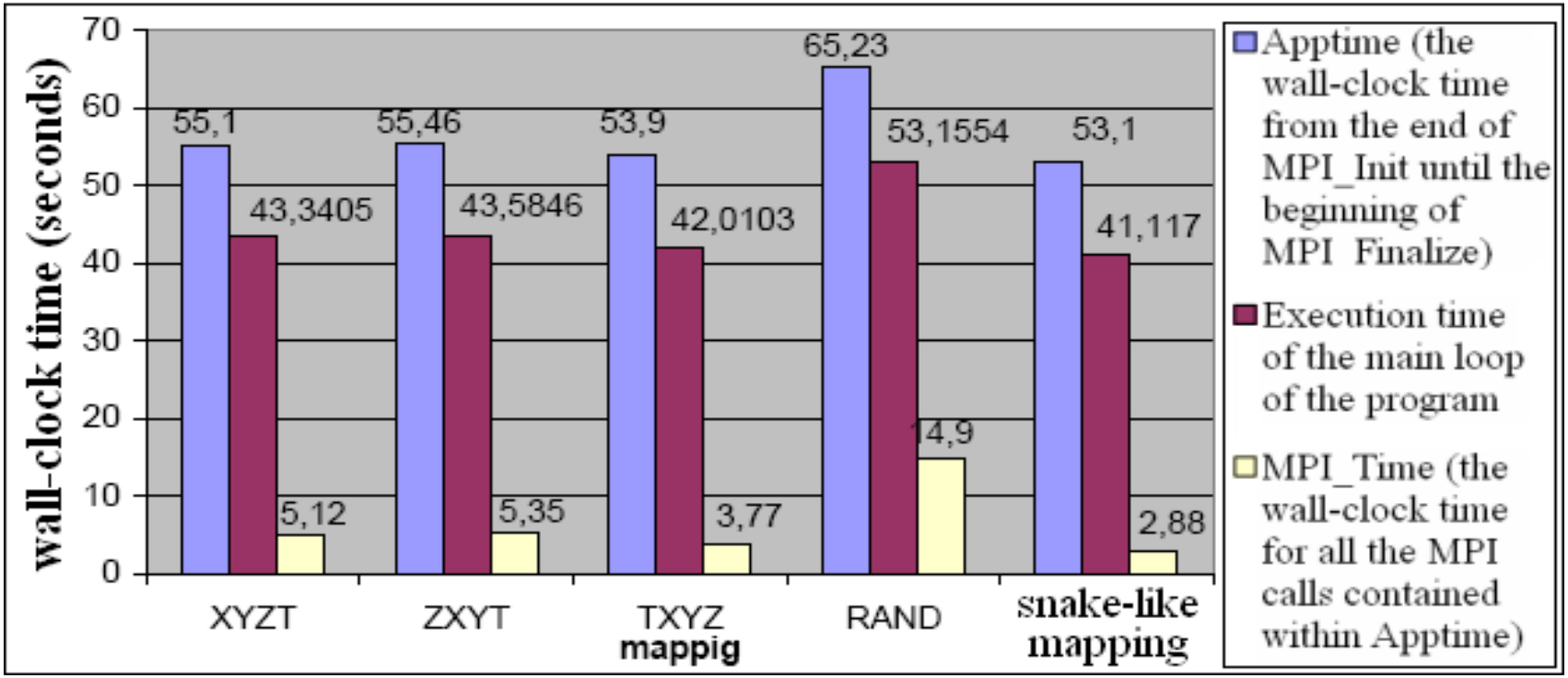
Общая статистика для этого вызова по всем MPI-процессам

Распределение данных полосами. Mapping.



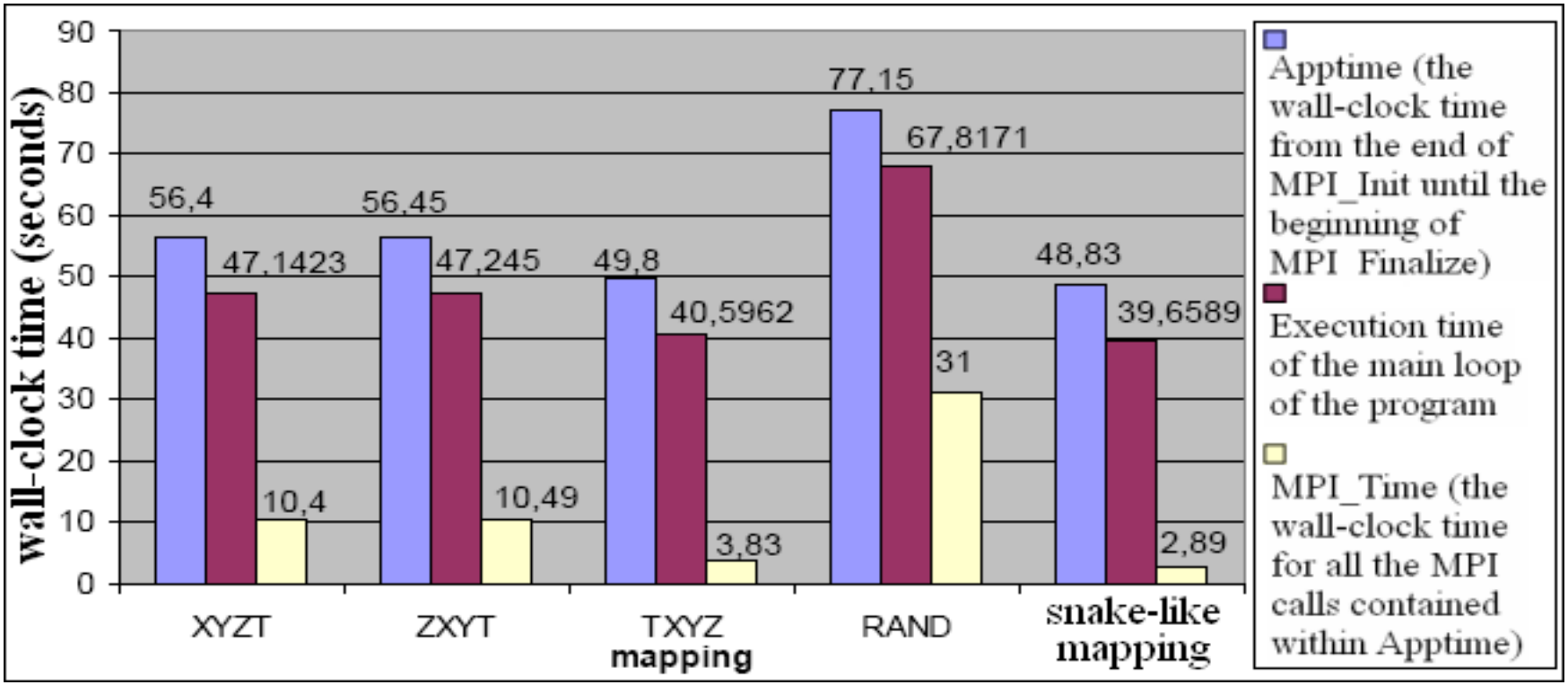
Среднее время выполнения **для различного mappinga** в режиме **SMP** на **128 вычислительных узлах** при виртуальной топологии **1*1*128**.

Распределение данных полосами. Mapping.



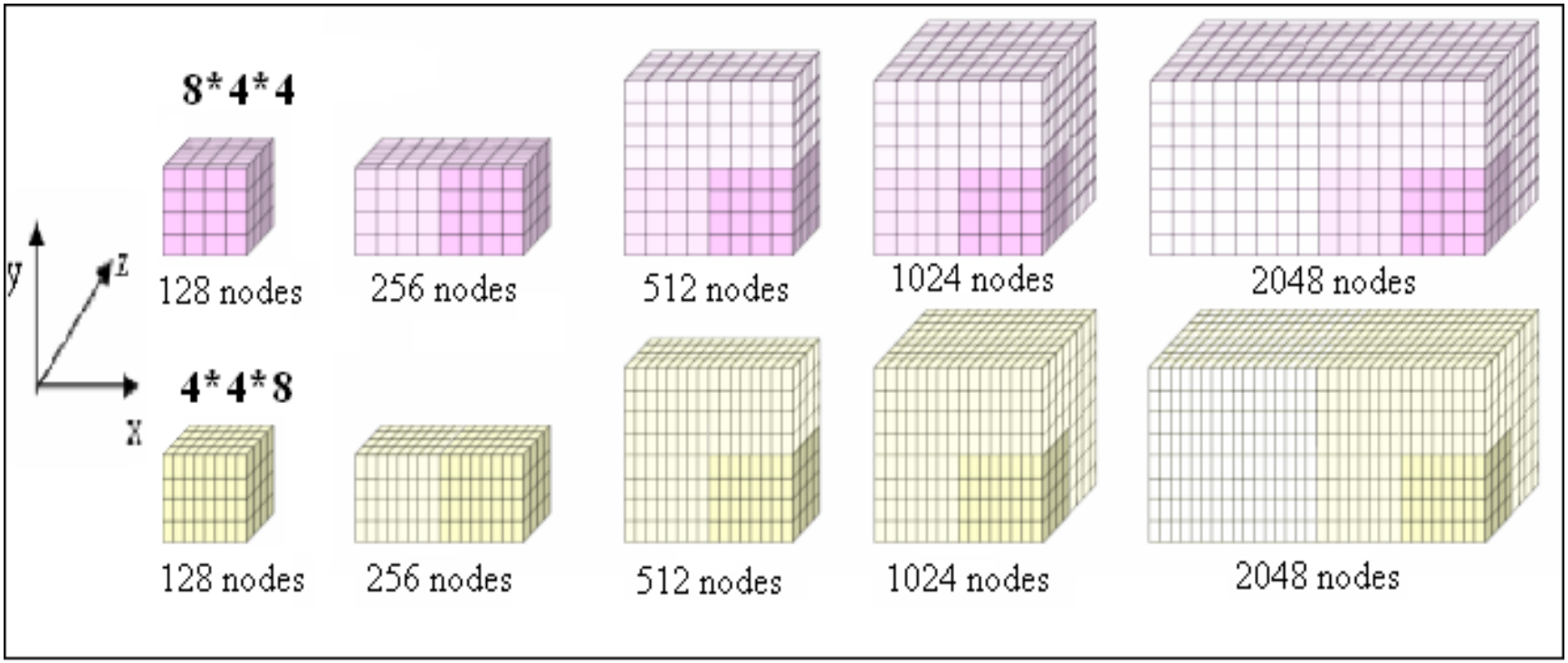
Среднее время выполнения **для различного mappinga** в режиме **DUAL** на **128 вычислительных узлах** при виртуальной топологии **1*1*256**.

Распределение данных полосами. Mapping.



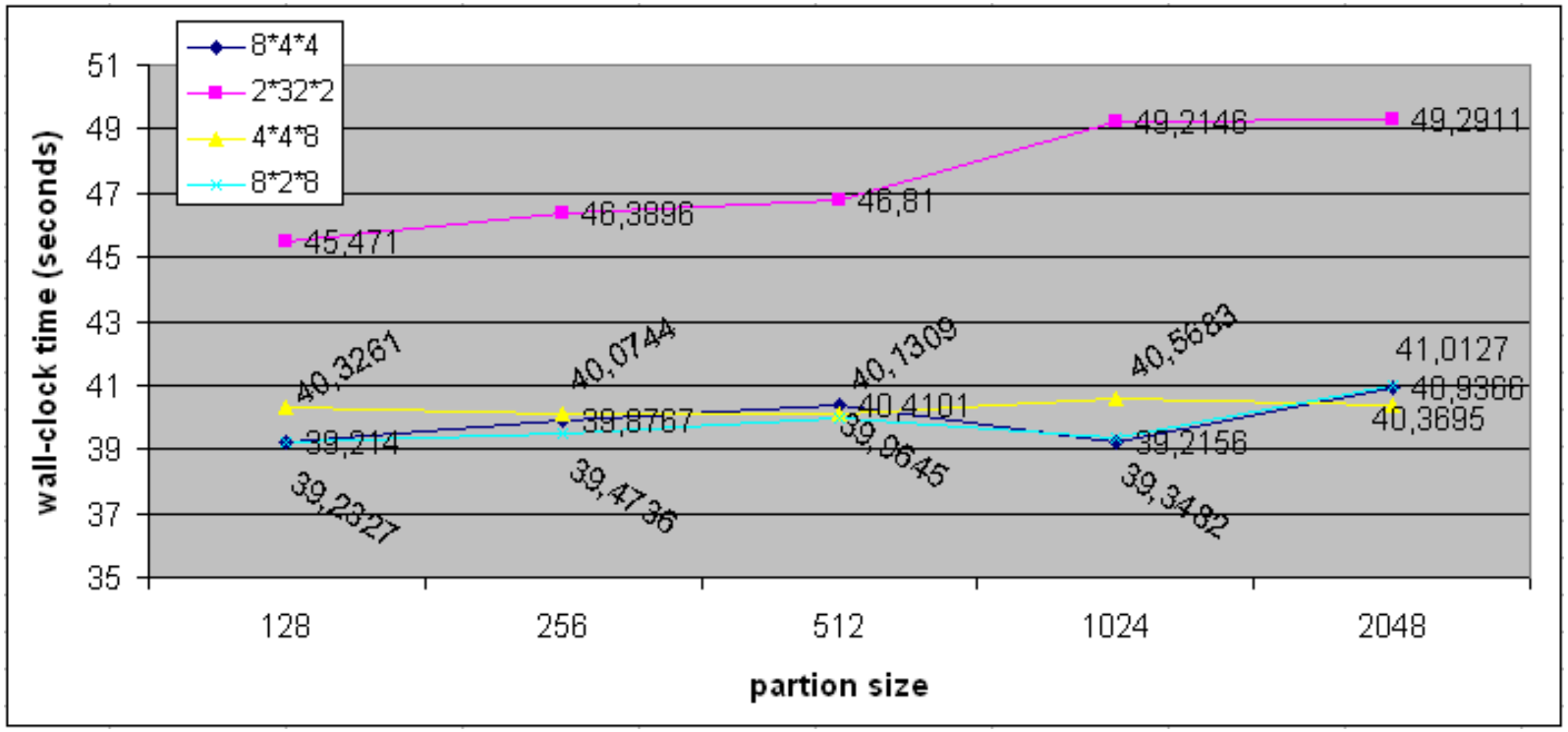
Среднее время выполнения **для различного mappinga** в режиме **VN** на **128 вычислительных узлах** при виртуальной топологии **1*1*512**.

Результаты экспериментов. Блочное распределение. Масштабируемость.



Организация эксперимента по исследованию масштабируемости. Направления увеличения размеров задачи и партиции.

Результаты экспериментов. Блочное распределение. Масштабируемость.



Время выполнения главного цикла программы для различных виртуальных топологий и размеров партиции (режим SMP).

Современные архитектуры НС.

Дмитрий Буряк.
к.ф.-м.н.
МГУ им. М.В. Ломоносова,
Исследовательская лаборатория LG Electronics
dyb04@yandex.ru

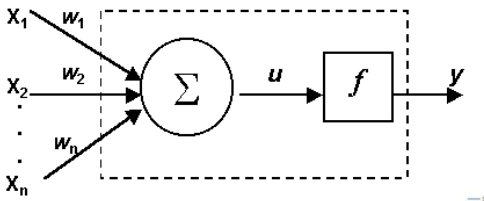
Содержание

- НС прямого распространения
- Современные алгоритмы обучения НС
- Методы повышения эффективности обучения
- Сверточные НС
- Современные архитектуры НС для классификации изображений
- Рекуррентные НС.

НС прямого распространения

Основные элементы НС

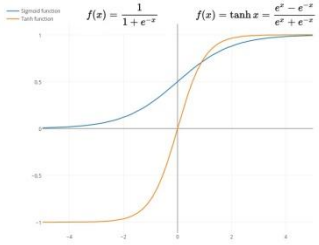
- Нейрон

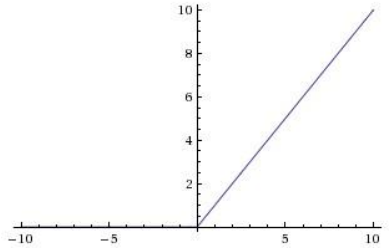


$$u = \sum_{i=1}^{n} w_i x_i \qquad y = f(u) = f\left(\sum_{i=1}^{n} w_i x_i\right)$$

f(u) – функция активации

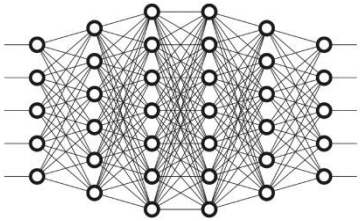
- Виды функции активации
  - Сигмоида
  - Гиперболический тангенс
  - ReLU (Rectified Linear Unit)





$$f(u) = \max(0, u)$$

- Нейронная сеть (НС) – направленный взвешенный граф
  - Задать архитектуру.
  - Подготовить обучающую выборку.
  - Провести обучение – вычислить веса НС



Алгоритм обратного распространения ошибки (1)

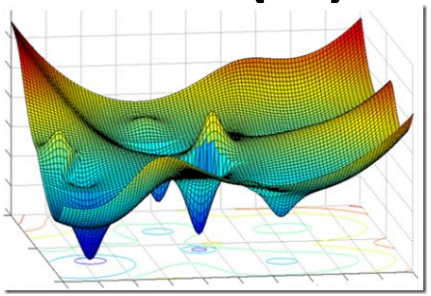
- Функция ошибки $E = F(d, y(x, w))$

  x – известные входные вектора;

  d – требуемые выходные значения сети;

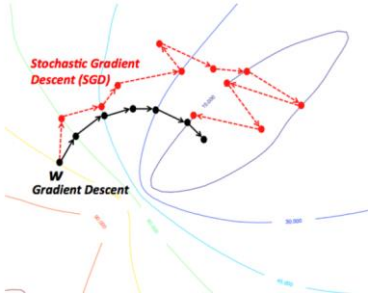
  $y(x,w)$ – вычисленные выходные значения сети;

  w – вектор весов сети.



- Поиск минимума функции ошибки

$$\hat{w} = \arg\min_{w} F(d, y(x, w))$$



- Метод обучения – градиентный спуск

$$\Delta w_{pq}^{k} = -\eta \cdot \frac{\partial E}{\partial w_{pq}^{k}}$$

$$w_{pq}^{k}(t+1) = w_{pq}^{k}(t) + \Delta w_{pq}^{k}$$

Алгоритм обратного распространения ошибки (2)

1. Рассчитать выходы сети.
2. Рассчитать для выходного слоя:
3. Рассчитать для всех остальных слоев:
4. Скорректировать веса:
5. Если ошибка существенна, перейти на шаг 1.

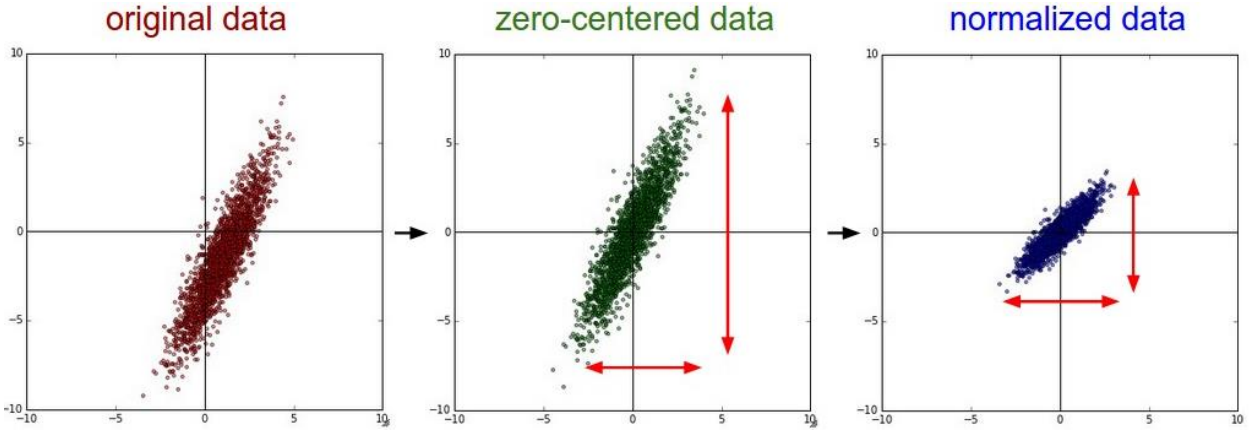
$$\delta_i = (y_i - d_i) \cdot \frac{dy_i}{du_i} \qquad \Delta w_{pi}^N = -\eta \delta_i \, \nu_p^{N-1}$$

$$\delta_p^{k-1} = \left[\sum_q \delta_q^k \cdot w_{pq}^k \right] \cdot \frac{d\nu_p^{(k-1)}}{du_p^{(k-1)}} \qquad \Delta w_{pq}^k = -\eta \delta_q^k \nu_p^{k-1}$$

$$w_{pq}^k(t+1) = w_{pq}^k(t) + \Delta w_{pq}^k$$

Предобработка входных данных

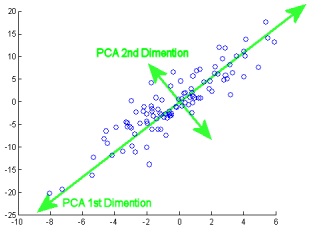
- Вычитание среднего
- Нормализация

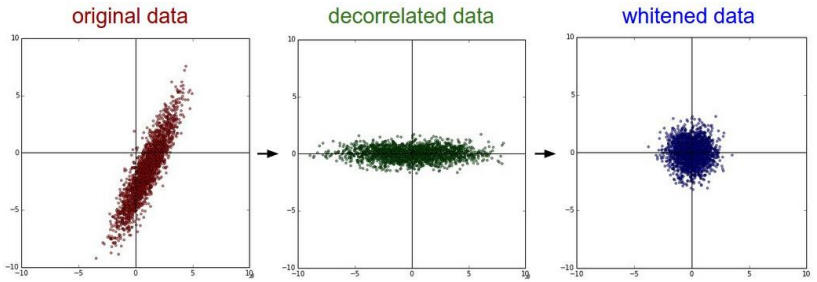


$$y_{ij} = x_{ij} - \underset{i}{mean}(x_{ij})$$

$$z_{ij} = \frac{y_{ij}}{\underset{i}{std}(y_{ij})} \quad z_{ij} = \frac{y_{ij} - \min_i(y_{ij})}{\max_i(y_{ij}) - \min_i(y_{ij})}$$

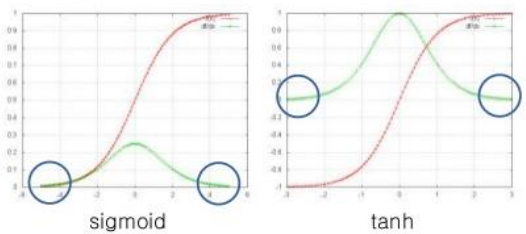
- Декорреляция





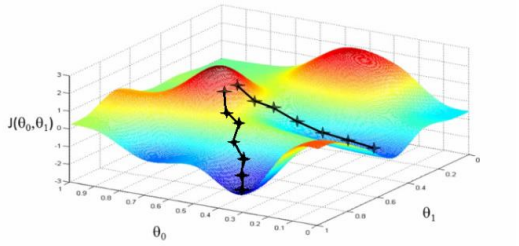
Проблемы алгоритма обратного распространения ошибки

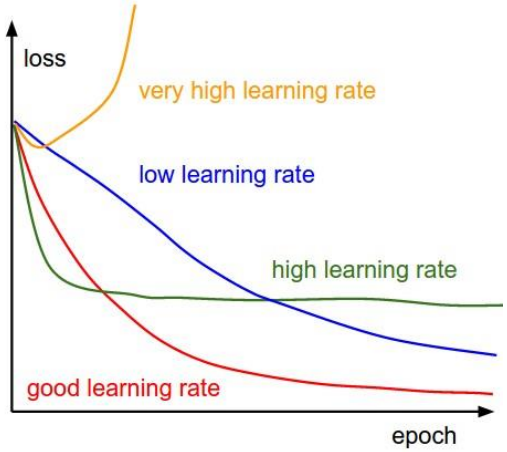
1. Градиент стремится к 0.





2. Обнаружение локального минимума.





3. Выбор скорости обучения.

4. Исчезающий градиент.



Современные алгоритмы обучения

Стохастический градиентный спуск

- ❑ Основной алгоритм для глубокого обучения.
- ❑ Базируется на принципах градиентного спуска.
- ❑ Вычисление градиента по каждому примеру вычислительно затратно → выбор случайных подмножест из обучающей выборки.

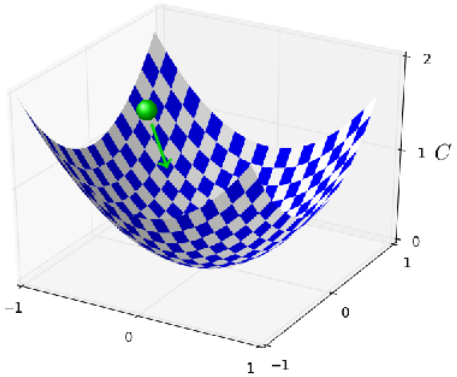
$$w_k \to w'_k = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \to b'_l = b_l - \eta \frac{\partial C}{\partial b_l}.$$

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{X_j}$$

$$w_k \to w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_l \to b'_l = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l},$$

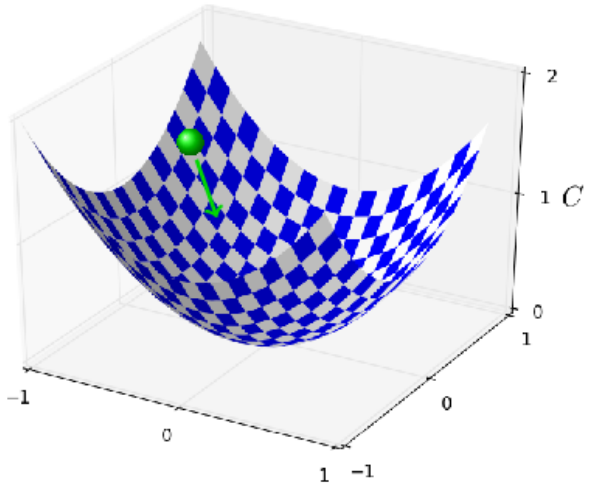


Использование момента

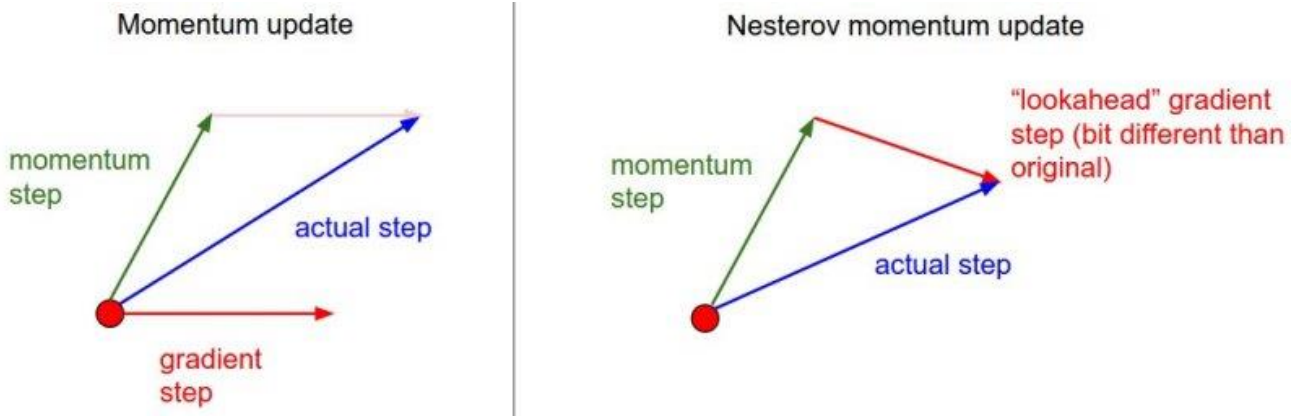
- Идея накопления импульса.
- Оценка «скорости» градиента.
- Применение коэффициента «трения» ~ коэффициент момента.

$$v \to v' = \mu v - \eta \nabla C$$
$$w \to w' = w + v'.$$

- Изменение коэффициента момента в процессе обучения 0.5→0.9



Момент Нестерова



- Оценка градиента в предсказанной позиции функции ошибки

Изменение скорости обучения

- Ступенчатое уменьшение
  - в ***к*** раз каждые ***n*** итераций обучения;
  - уменьшение коэффициента обучения, если ошибка на подтверждающей выборке перестала уменьшаться.

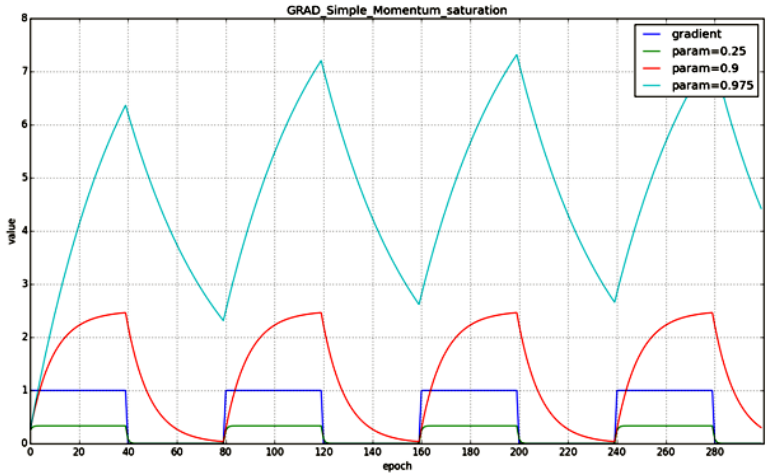
- Экспоненциальное $\eta = \alpha_0 e^{-\beta t}$

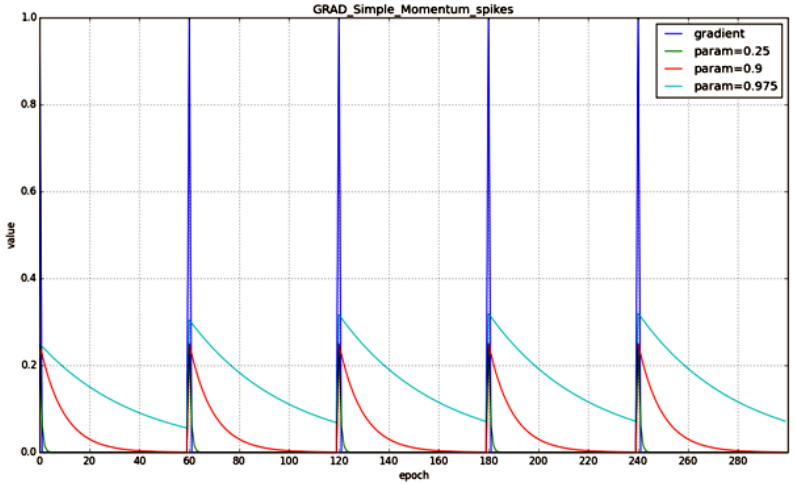
- Гиперболическое $\eta = \frac{\alpha_0}{1 + \beta t}$

Обзор алгоритмов обучения*. Момент.

$$v_t = \gamma v_{t-1} + \eta g_t \qquad g_t = \nabla_\theta C(\theta)$$

$$\theta_{t+1} = \theta_t - v_t \qquad \gamma = 0.9$$

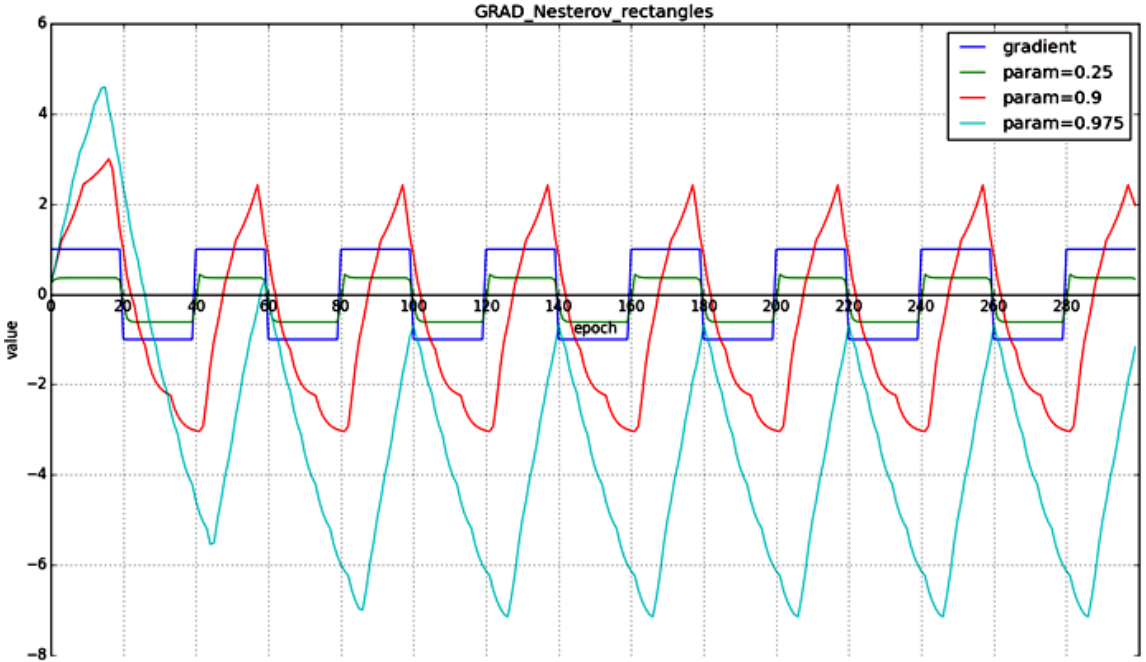




** https://habrahabr.ru/post/318970/*

Обзор алгоритмов обучения. Момент Нестерова.

$$v_t = \gamma v_{t-1} + \eta g_t(\theta - \gamma v_{t-1})$$
$$\theta_{t+1} = \theta_t - v_t$$



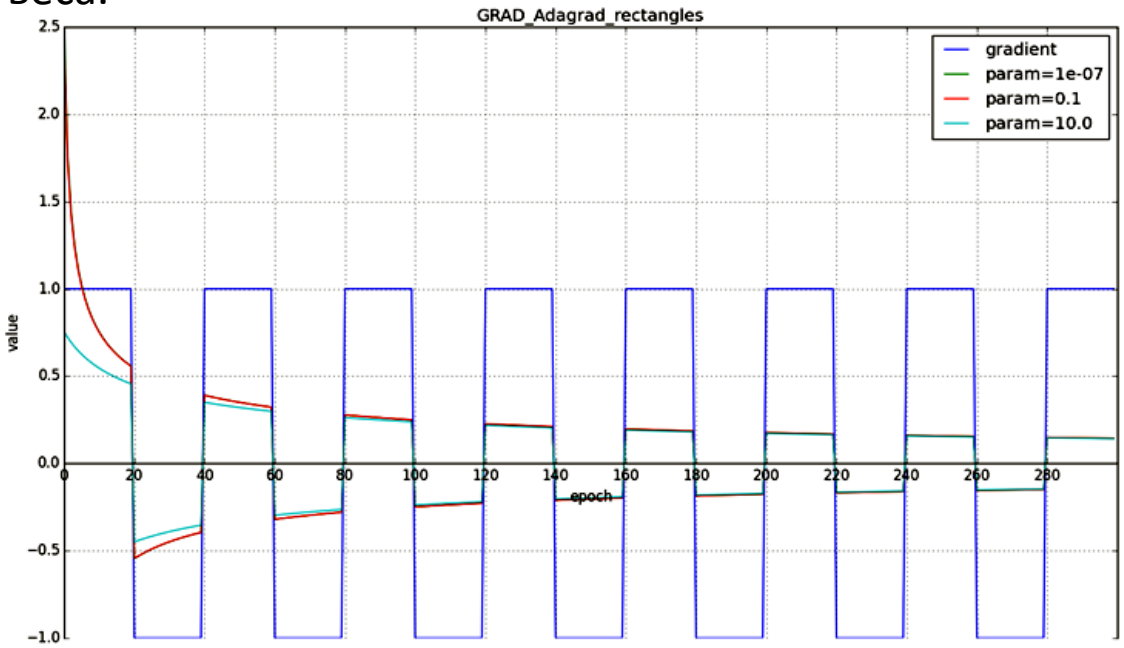
Обзор алгоритмов обучения. Adagrad.

- Учет частоты обновления веса.
- Устойчивость к выбору скорости обучения.

$$G_t = G_t + g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$

ϵ — сглаживающий параметр

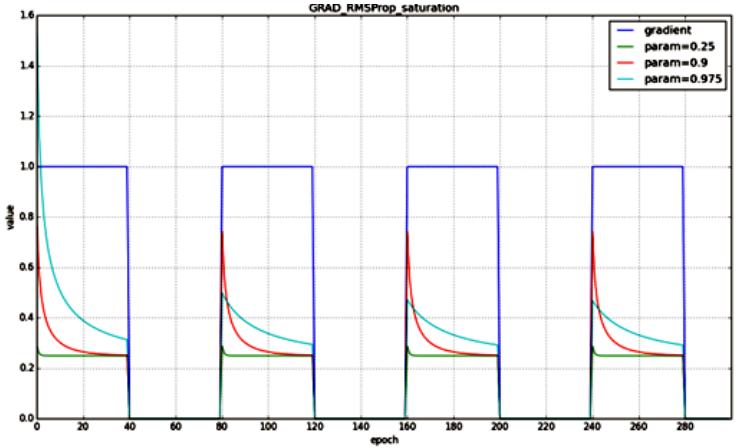


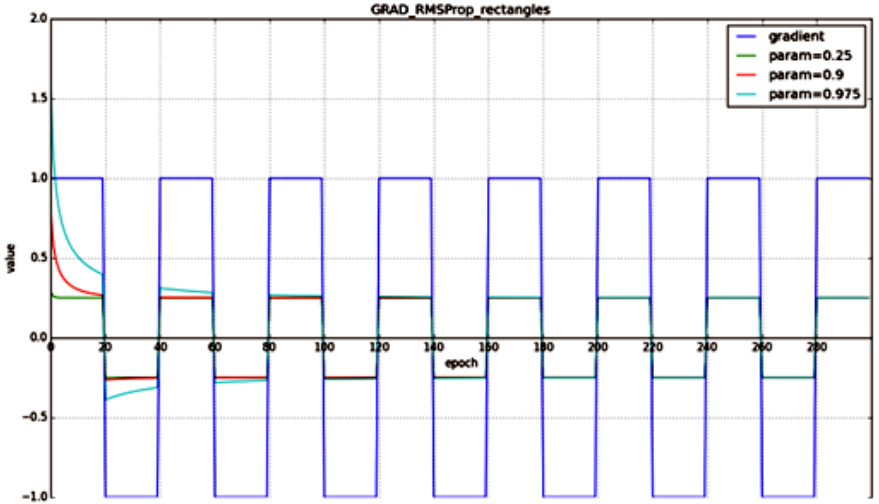
Обзор алгоритмов обучения. RMSProp.

- Учет частоты обновления веса за счет усреднения.

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$





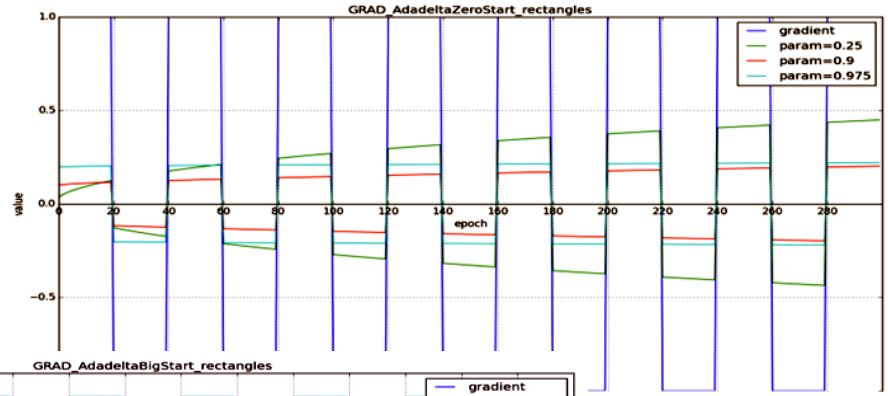
Обзор алгоритмов обучения. Adadelta.

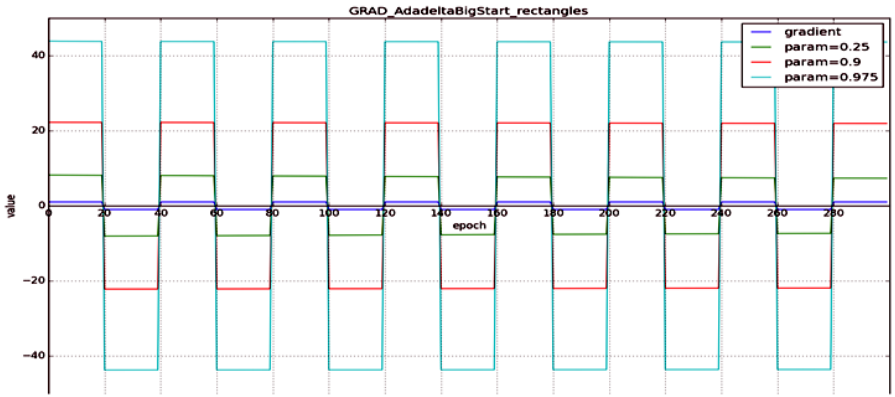
$$\Delta\theta = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

$$\theta_{t+1} = \theta_t - \frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2$$

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$





Обзор алгоритмов обучения. Adam.

- Момент + учет частоты обновления веса.

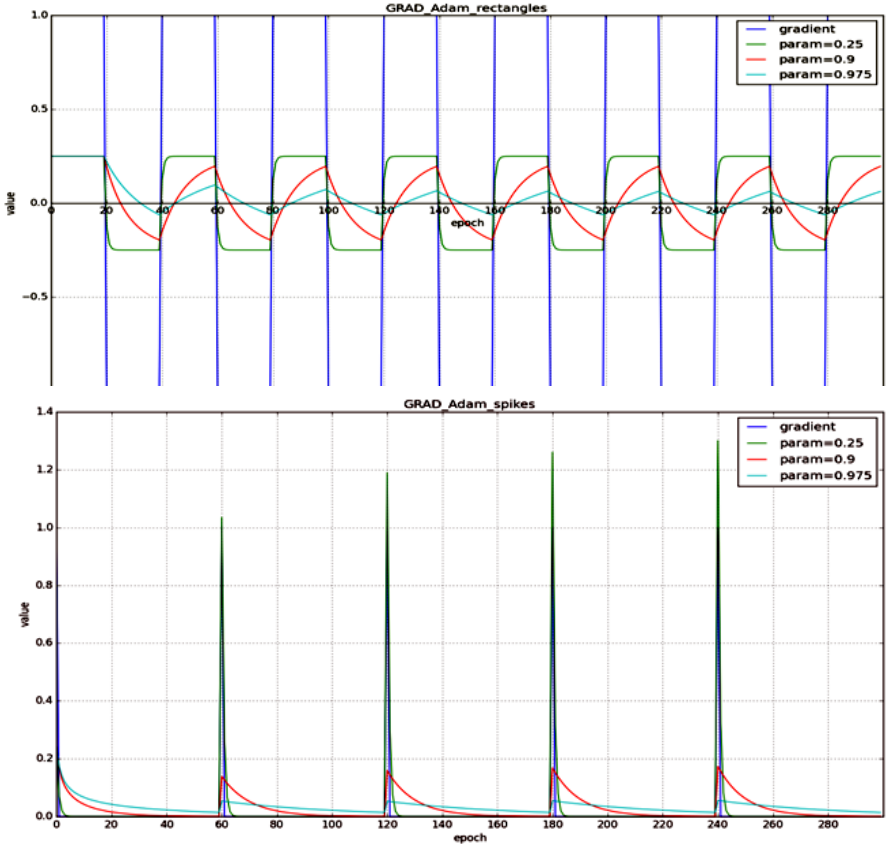
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

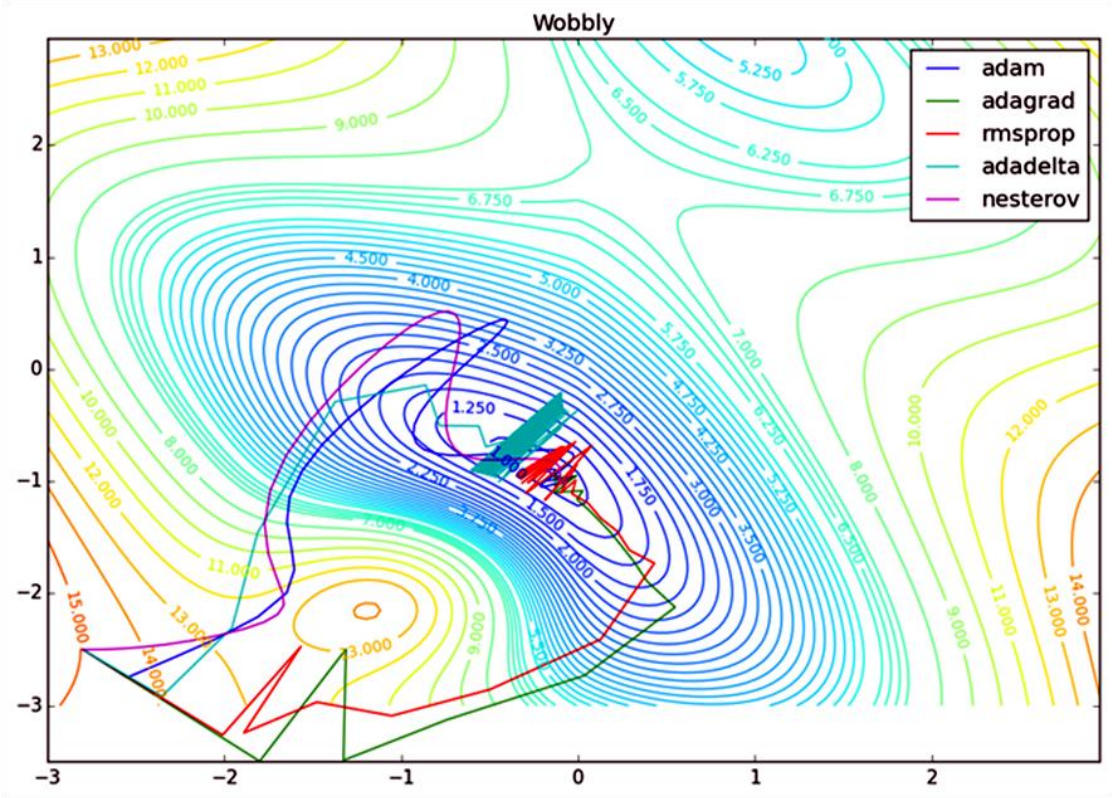
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \; \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

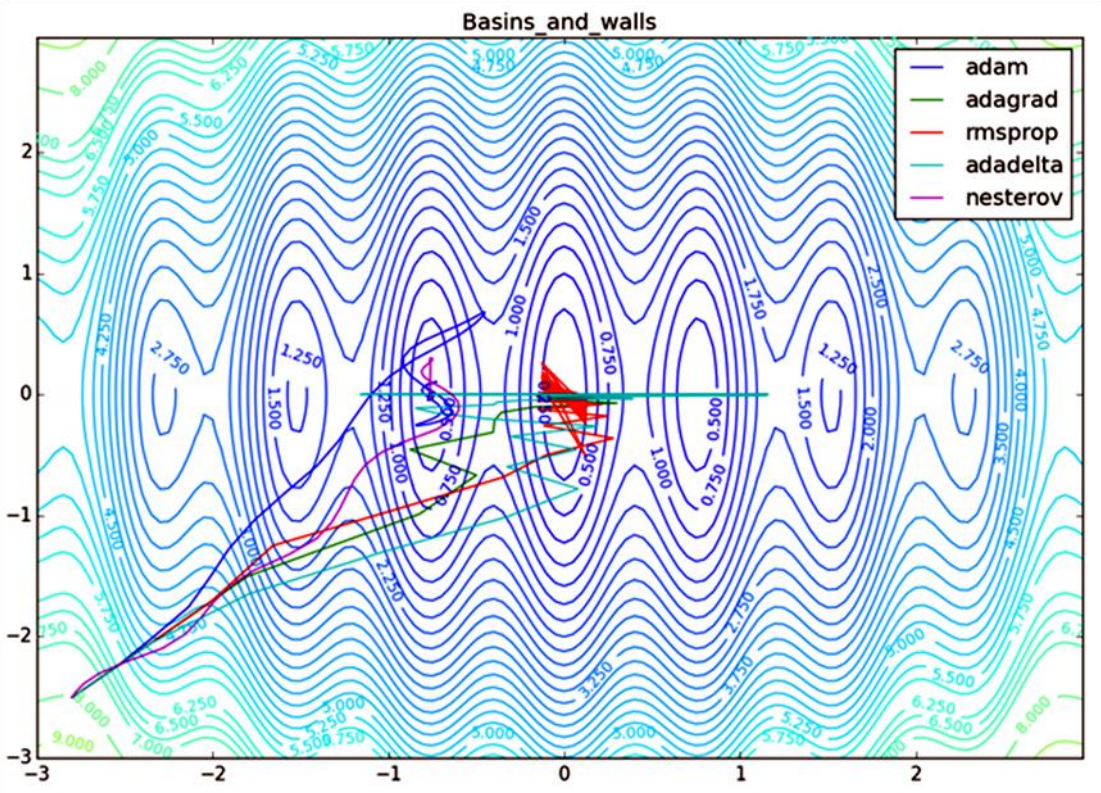
$$\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$$



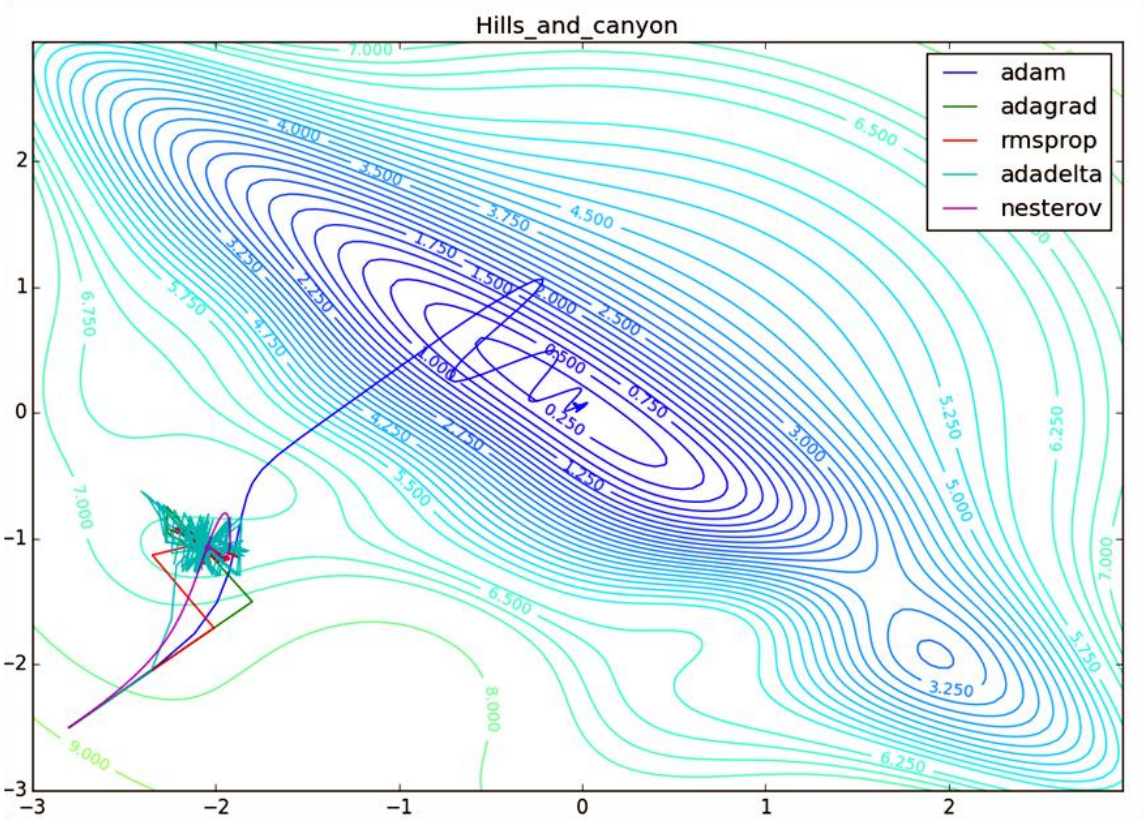
Обзор алгоритмов обучения. Эксперименты (1).



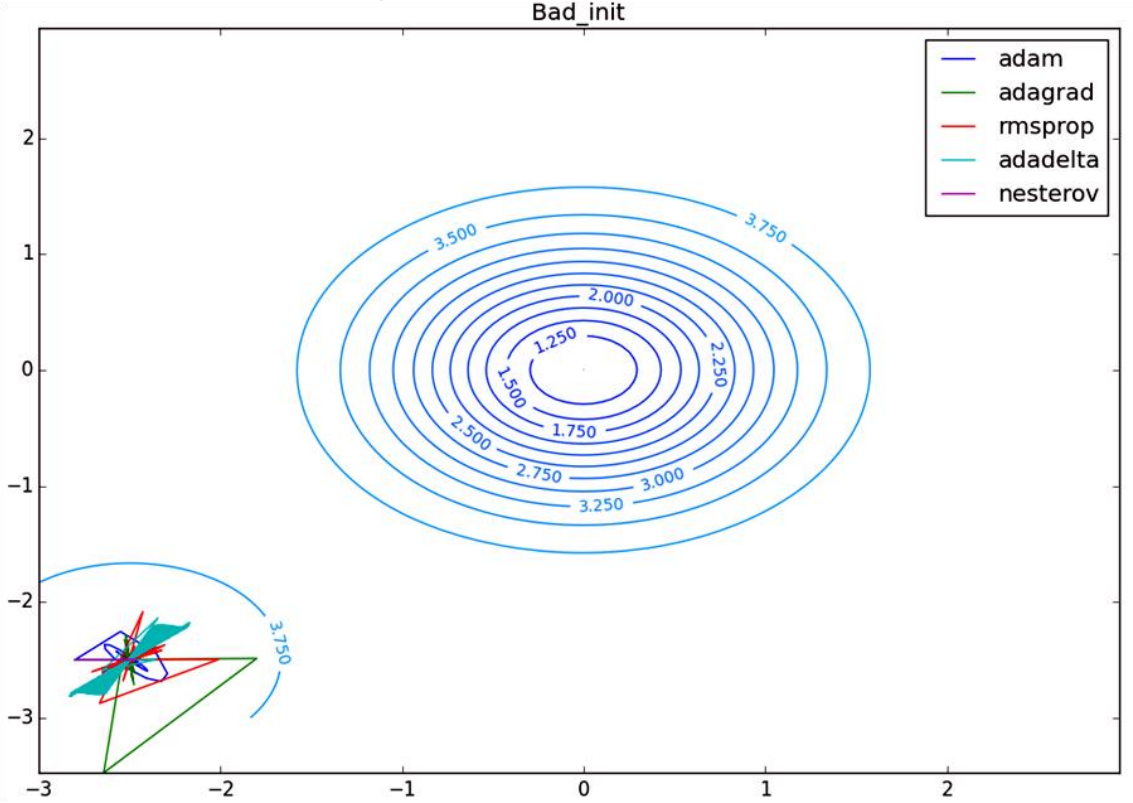
Обзор алгоритмов обучения. Эксперименты (2).



Обзор алгоритмов обучения. Эксперименты (3).



Обзор алгоритмов обучения. Эксперименты (4).



Методы повышения эффективности обучения

Batch normalization

- Нормализация + декорреляция входных данных → повышение эффективности обучения.
- Нарушение полученных свойств входных векторов для промежуточных данных во внутренних слоях (internal covariance shift).
- Идея: проводить предобработку входных данных для каждого внутреннего слоя.
- Оптимизация вычислительных затрат → нормализация внутренних данных (без декорреляции).

Sergey Ioffe, Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015

Batch normalization

Input: Network N with trainable parameters Θ; subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$

1: $N_{\text{BN}}^{\text{tr}} \leftarrow N$ // Training BN network

2: **for** $k = 1 \ldots K$ **do**

3: Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Alg. 1)

4: Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead

5: **end for**

6: Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$

7: $N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$ // Inference BN network with frozen // parameters

8: **for** $k = 1 \ldots K$ **do**

9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.

10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\text{E}[x] \leftarrow \text{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\text{Var}[x] \leftarrow \frac{m}{m-1}\text{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

11: In $N_{\text{BN}}^{\text{inf}}$, replace the transform $y = \text{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\text{E}[x]}{\sqrt{\text{Var}[x]+\epsilon}}\right)$

12: **end for**

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$; Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

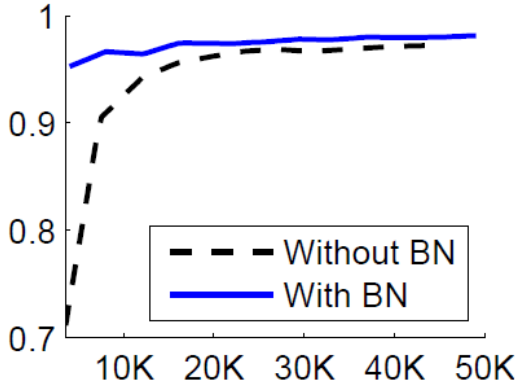
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

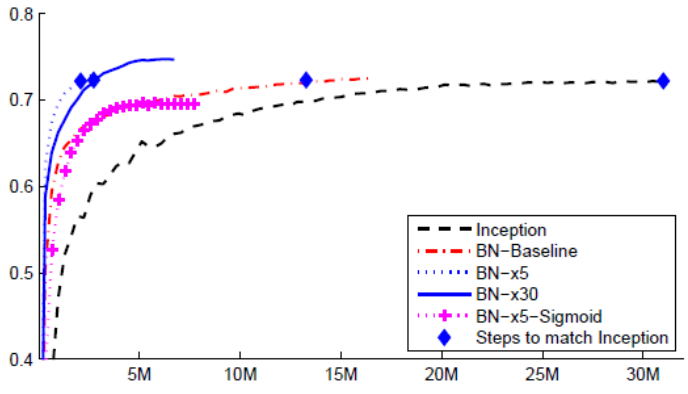
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- Классификация MNIST
- Сеть 784x100x100x100x10, 50000 обучающих примеров



- Классификация ImageNet
- Сеть $13.6*10^6$ параметров, 1000 классов

Model	Steps to 72.2%	Max accuracy
Inception	$31.0 \cdot 10^6$	72.2%
BN-Baseline	$13.3 \cdot 10^6$	72.7%
BN-x5	$2.1 \cdot 10^6$	73.0%
BN-x30	$2.7 \cdot 10^6$	74.8%
BN-x5-Sigmoid		69.8%



Регуляризация L2

- Регуляризация - метод предотвращения переобучения НС.
- Введение штрафа для больших весов.

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2$$

- λ - коэффициент регуляризации.

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w$$
$$\frac{\partial C}{\partial b} = \frac{\partial C_0}{\partial b}.$$

$$b \to b - \eta \frac{\partial C_0}{\partial b}$$
$$w \to w - \eta \frac{\partial C_0}{\partial w} - \frac{\eta\lambda}{n} w$$
$$= \left(1 - \frac{\eta\lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w}$$

- Масштабирование веса перед коррекцией по градиентному спуску.

Регуляризация L1

- Введение штрафа для больших весов.

$$C = C_0 + \frac{\lambda}{n} \sum_w |w|$$

- λ - коэффициент регуляризации.

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} \operatorname{sgn}(w) \qquad w \to w' = w - \frac{\eta\lambda}{n} \operatorname{sgn}(w) - \eta \frac{\partial C_0}{\partial w}$$

- Уменьшение веса на фиксированную величину

- Для регуляризации L2 значение уменьшения веса зависит от его величины.

$$w \to w' = w \left(1 - \frac{\eta\lambda}{n}\right) - \eta \frac{\partial C_0}{\partial w}$$

Регуляризация Max-norm

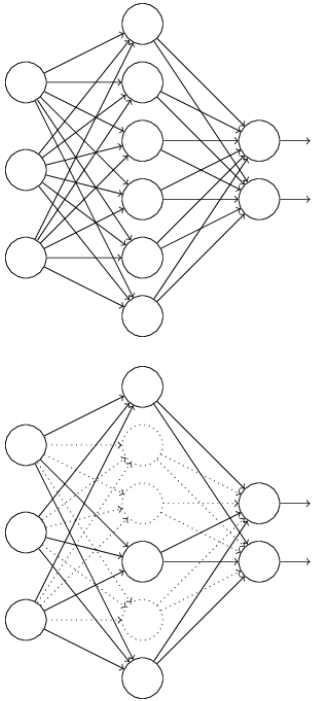
- Ограничения нормы вектора весов для каждого нейрона.

$$||\mathbf{w}||_2 \leq c$$

- c - гиперпараметр.
- Реализация через нормировку вектора весов при невыполнении неравенства.
- Эффективна при совместном использовании с Dropout

Dropout

- Инструмент регуляризации.
- Модификация архитектуры сети в процессе обучения.
- Упрощенная схема Dropout
    1. Временно удалить из НС половину случайно выбранных внутренних нейронов с соответствующими связями.
    2. Провести итерацию обучение на пакете: обновление связей оставшихся нейронов.
    3. Восстановить удаленные нейроны и их связи.
    4. Повторить п. 1 – 3.
- Перед применением сети уменьшить внутренние веса в 2 раза.



N. Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting, 2014.

Dropout. Пример использования.

- MNIST.

Входной вектор 784 элемента.

10 классов.

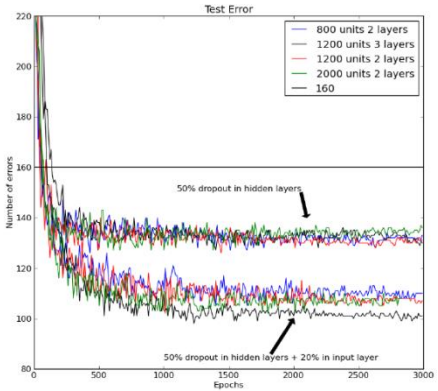
10000 тестовых изображений.

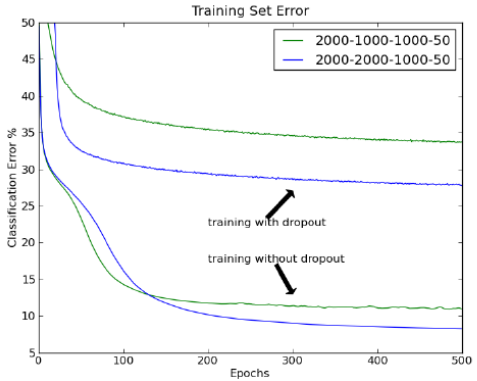
- Reuters. Классификация документов.

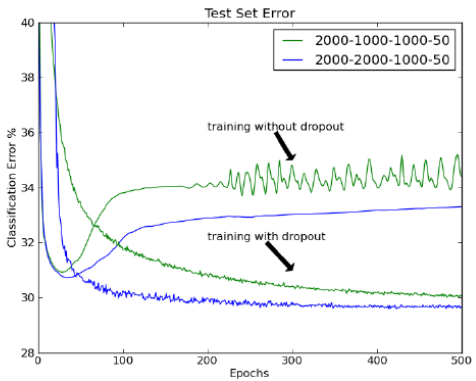
Входной вектор 2000 элементов.

50 классов.

~200000 тестовых документов.



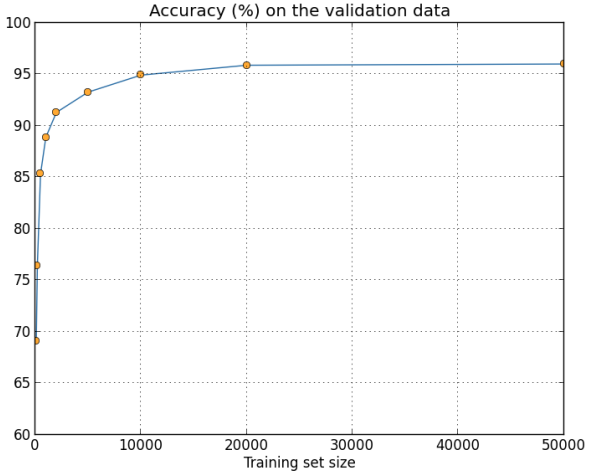


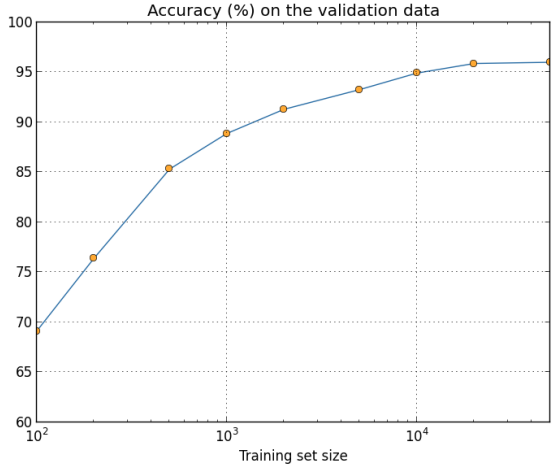


G.E. Hinton et al, Improving neural networks by preventing co-adaptation of feature detectors, 2012

Увеличение обучающей выборки.

- Способствует росту обобщающей способности НС





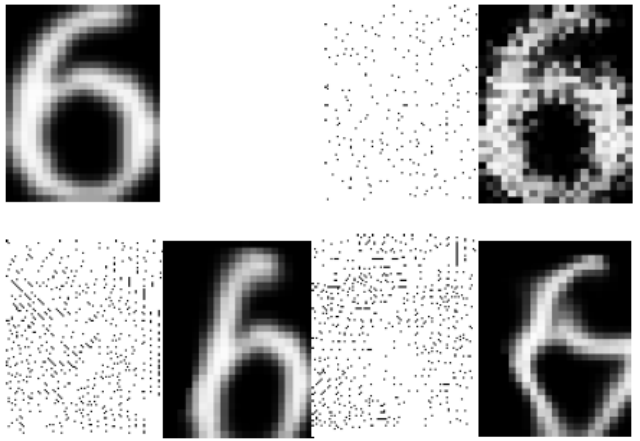
Сеть
784x30x10

- «Естественное» получение новых примеров часто сопряжено со сложностями

Методы генерации обучающих примеров.

- MNIST, MLP 784x800x10

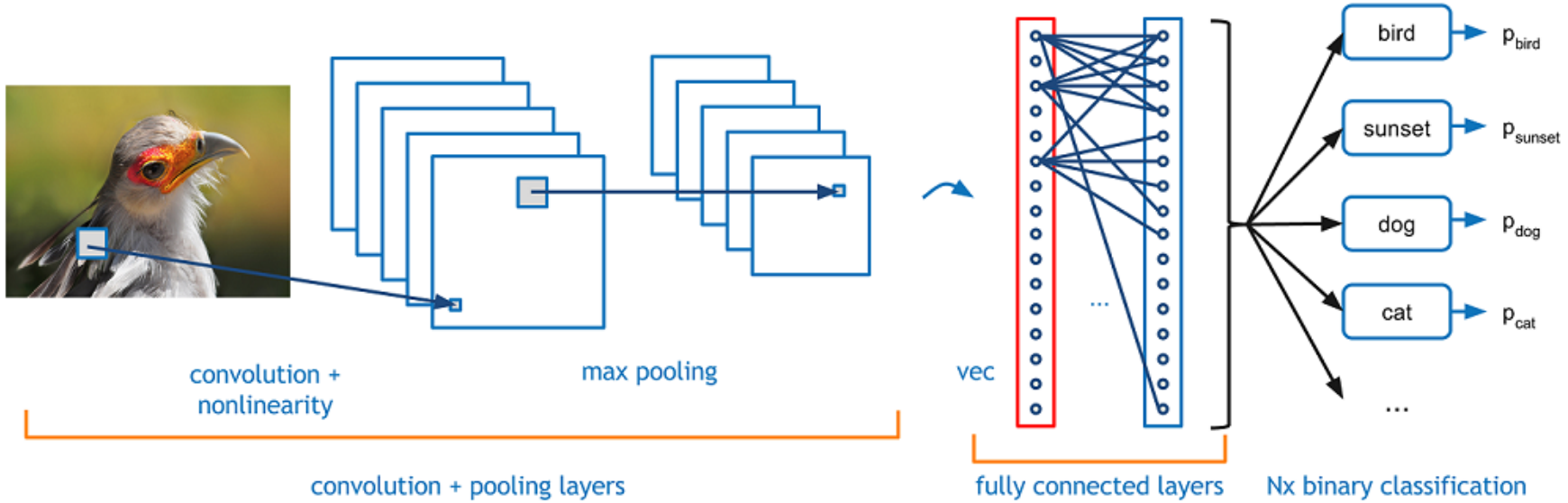
Тип искажения	Ошибка на тестовой выборке
Нет	1.6%
аффинные преобразования	1.1%
«эластичные» искажения	0.4%



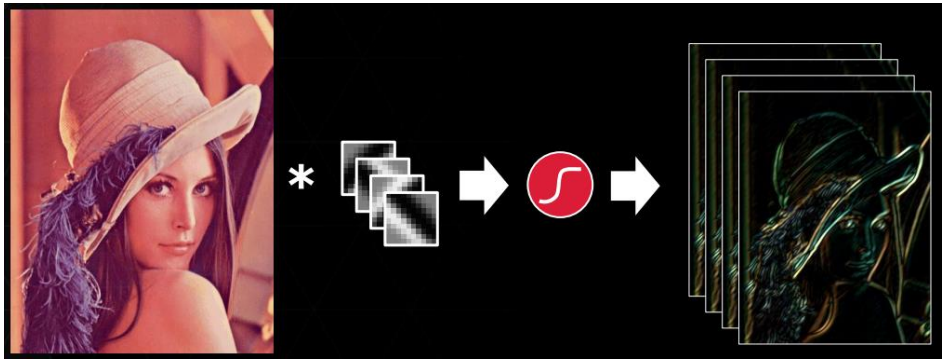
P. Y. Simard et al. Best practices for convolutional neural networks applied to visual document analysis, 2003

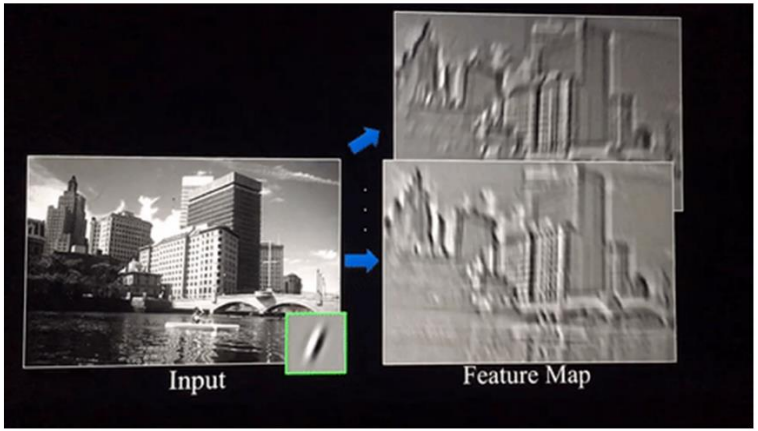
Сверточные НС

Типы слоев


convolution +
nonlinearity
max pooling
vec
bird
sunset
dog
cat
...
p_{bird}
p_{sunset}
p_{dog}
p_{cat}
convolution + pooling layers
fully connected layers
Nx binary classification

Типы слоев. Свертка



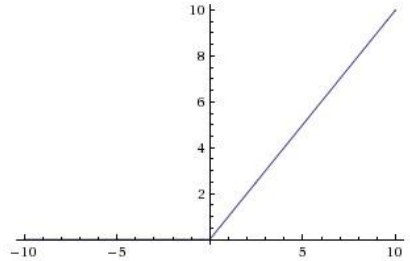


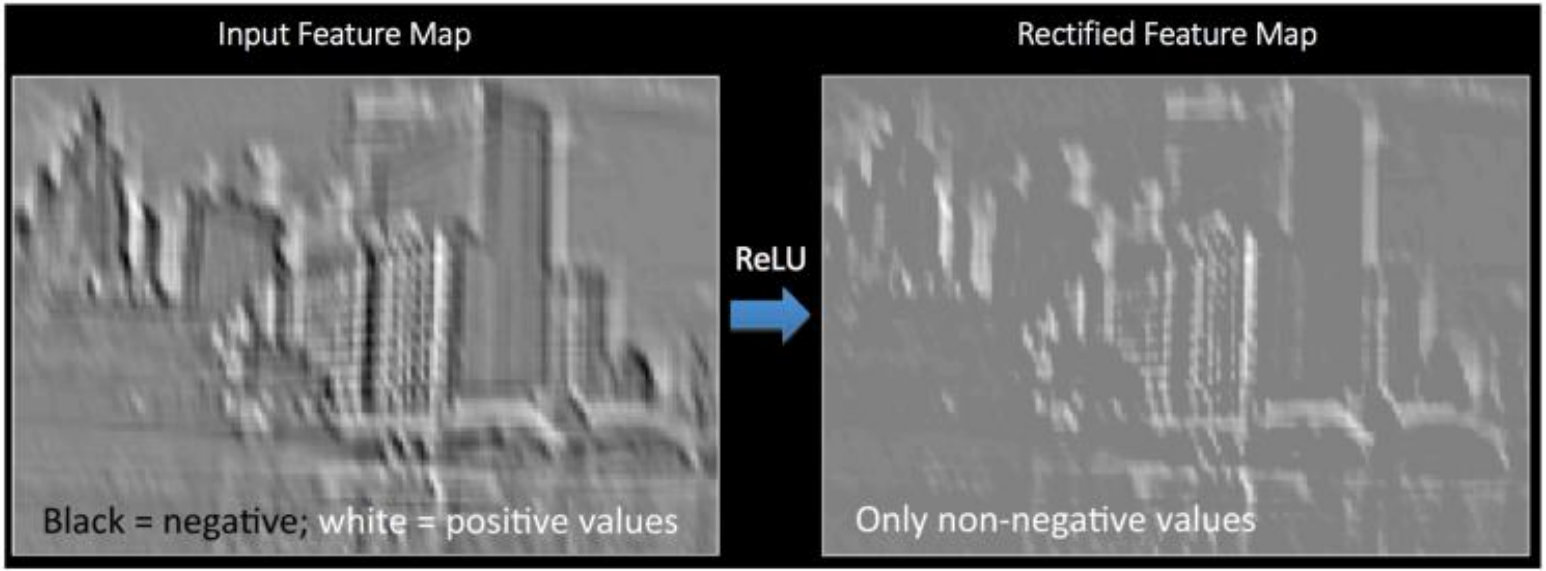
- Свертка входного образа с набором ядер
- Применение нелинейной функции активации
- Результат – множество карт признаков

Типы слоев. Функция активации

- ReLU – Rectified Linear Unit:

$$f(u) = \max(0, u)$$

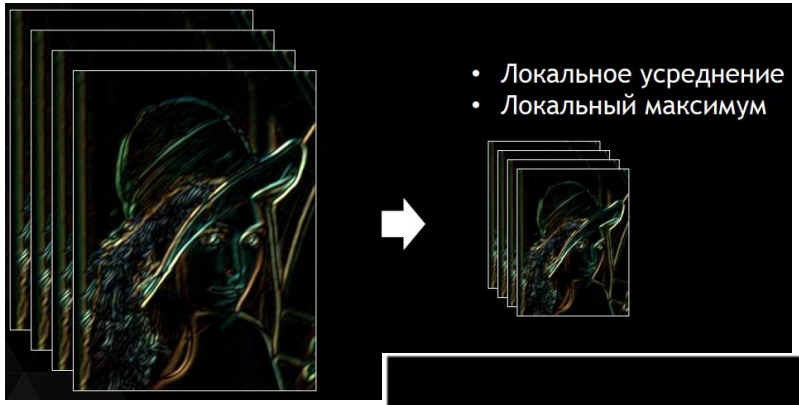


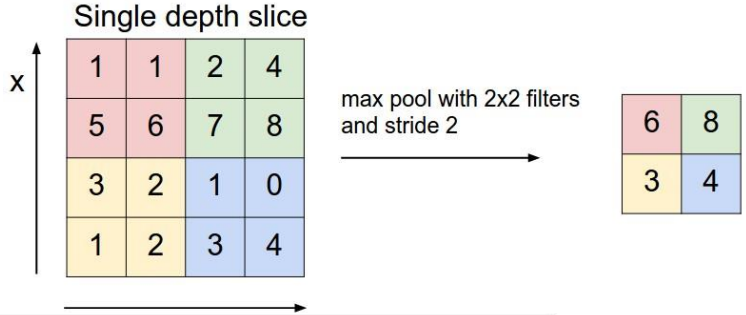


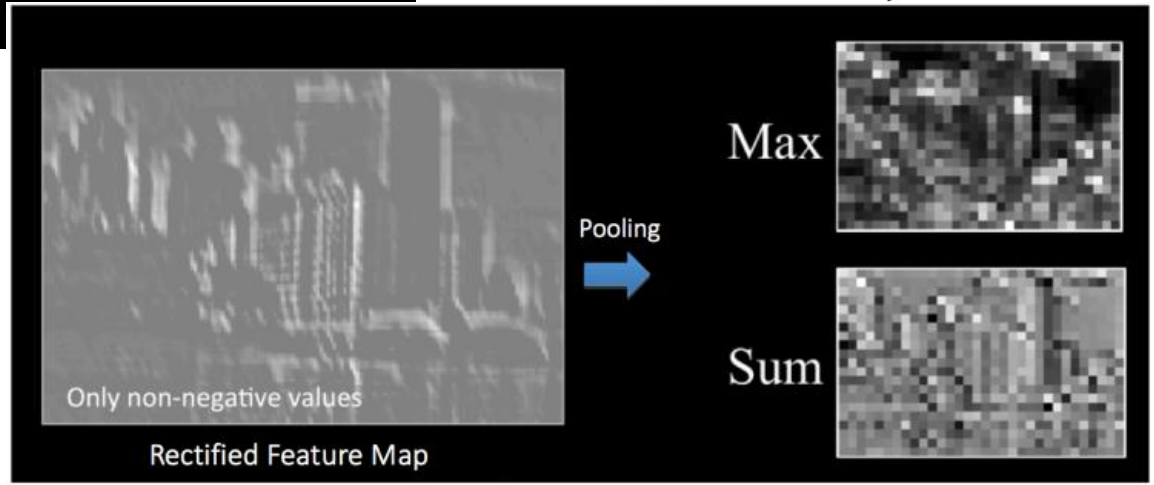
Типы слоев. Свойства ReLu

- Преимущества
  - Решение проблемы «vanishing gradient»
  - Проще для вычисления
- Недостатки
  - Функция не ограничена
  - Возможность переобучения
- ReLu vs сигмоида
  - Для ряда задач важны ограниченность и симметричность
  - В 95% случаев ReLu эффективнее

Типы слоев. Понижение размерности (pooling)







Типы слоев. Полносвязанный слой

- Архитектура аналогичная персептрону
- Один или несколько слоев
- Проведение классификации на основе выделенных признаков

Обучение

- Основные проблемы
  - Большое число связей
  - Размер обучающей выборки
  - Переобучение
  - Затухание градиента
- Методы
  - Обратное распространение ошибки
  - Стохастический градиентный спуск
  - Регуляризация
  - Dropout
  - Batch normalization

Современные архитектуры НС для классификации изображений

База изображений ImageNet

- ImageNet
  - Создана в 2009
  - Более 14М изображений с аннотациями
  - 1М изображений с отмеченными объектами
  - Более 20К классов (несколько сотен изображений каждого класса





- ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
  - Проводился с 2010г по 2017г;
  - Классификация изображений, обнаружение объектов.



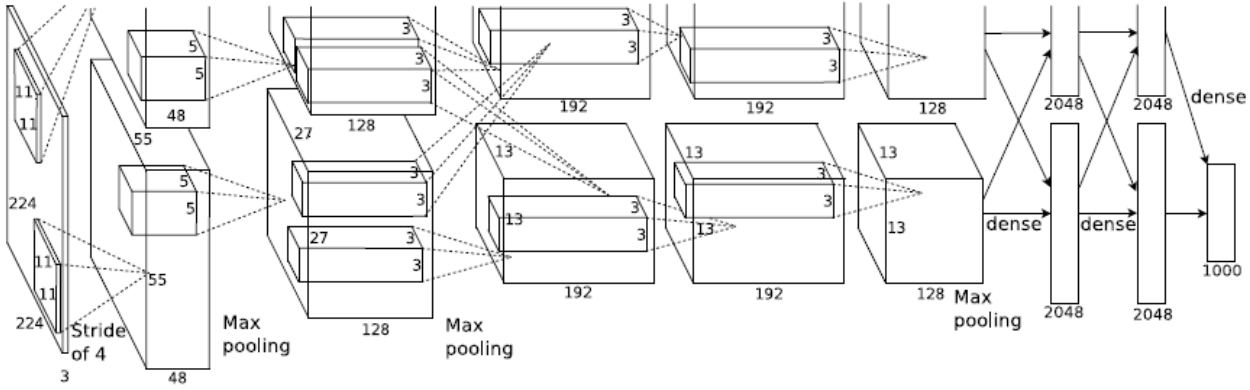


- ILSVRC. Классификация изображений
  - 1.2М изображений (обучающая выборка)
  - 100К тестовых изображений
  - 1000 классов



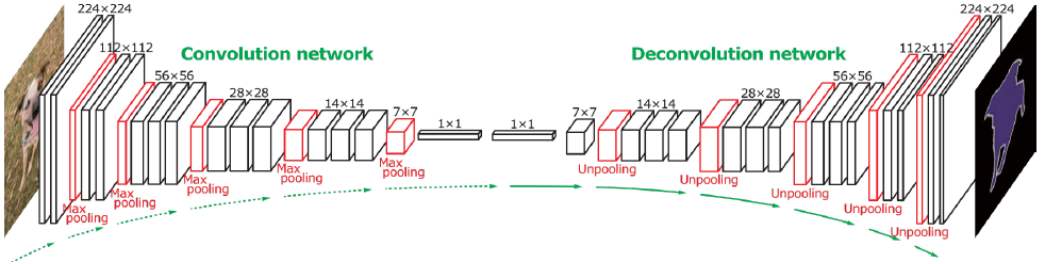
Победитель ILSVRC-2012. AlexNet

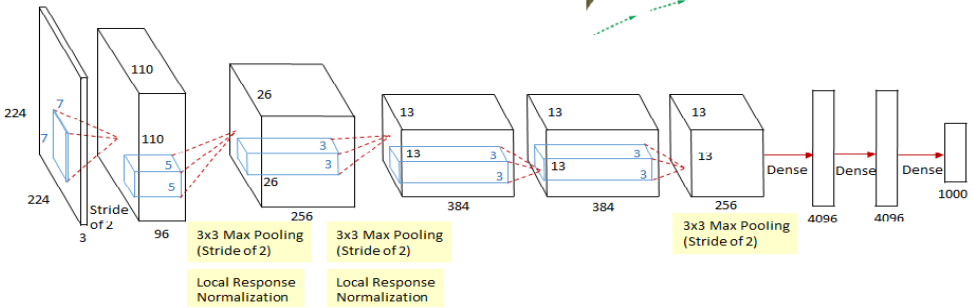
- AlexNet (A. Krizhevsky et al, 2012)
  - Первая сверточная сеть победитель ILSVRC;
  - Ошибка Top-5: **15.3%** (второй результат SIFT – 26.2%);
  - Сверточная НС, 8 слоев (5 сверточных), 60M параметров;
  - ReLU, Dropout, Увеличение обучающей выборки (сдвиг, отражение);
  - Обучение заняло 5 дней на нескольких GPU.



Победитель ILSVRC-2013. Clarifai

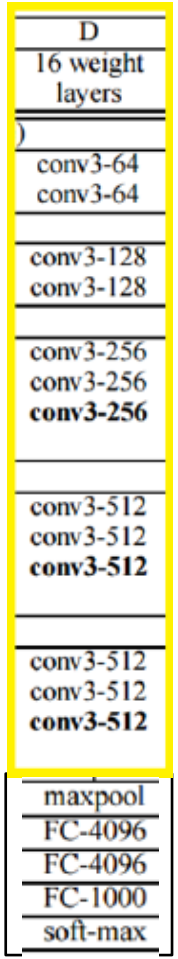
- ZFNet (Clarifai – стартап авторов ZFNet) – (M. Zeiler et al, 2014)
  - Ошибка Top-5: **11.2%** (несколько сетей);
  - Визуализация активаций внутренних слоев - Deconvnet;
  - Анализ внутренних шаблонов → тонкая настройка AlexNet;
  - Фильтры первого слоя (11x11) → (7x7); шаг свертки первого слоя 4 → 2;
  - Обучение 12 дней.





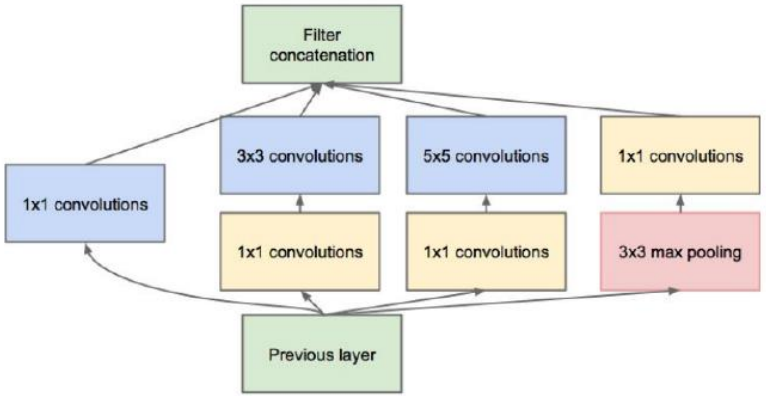
VGG Net

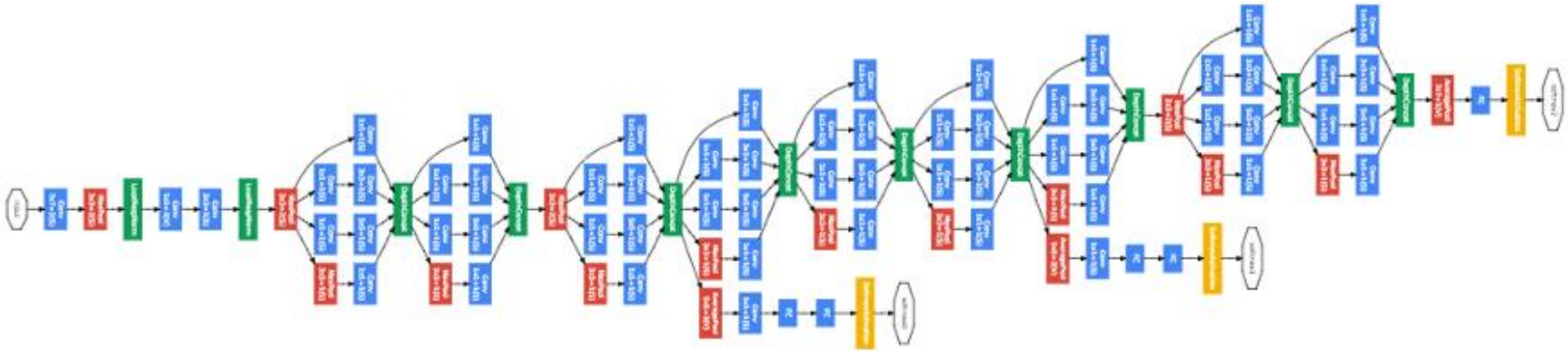
- VGG (K.Simonyan, A. Zisserman, 2014)
  - Ошибка Top-5: **7.3%** (ILSVRC-2014);
  - Классическая CNN;
  - Применение ядер 3x3;
  - (3x3)+(3x3) ~ (5x5); (3x3)+(3x3)+(3x3) ~ (7x7);
  - Обучение 4GPUx(3 недели);
  - Активно применяется при решении задач.



Победитель ILSVRC-2014. GoogLeNet

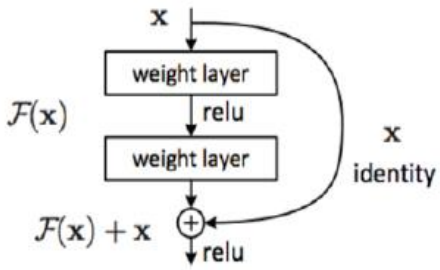
- GoogLeNet – (K. Szegedy et al, 2015)
  - Ошибка Top-5: **6.7%**;
  - «Параллельные свертки-Конкатенация»: Inception модуль.
  - 22 модуля, более 50 сверточных слоев;
  - Несколько выходных слоев;
  - Размер 55Мб (VGG16 – 490Мб);
  - Обучение 7 дней на нескольких GPU;

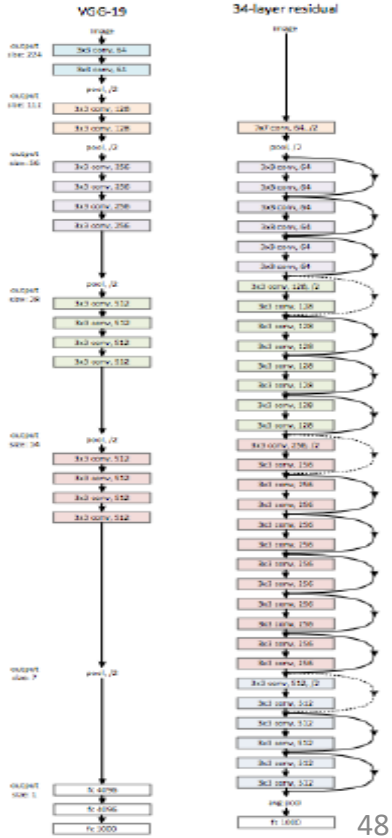




Победитель ILSVRC-2015. ResNet

- ResNet – (K. He et al, 2015)
  - Ошибка Top-5: **3.57%**;
  - Рост глубины → меньше ошибка → сложность оптимизации;
  - «Остаточные» связи – единичное отображение, нет дополнительных параметров;
  - «Остаточные» соединения → лучше распространение градиента
  - 152 слоя, ядра 3x3;
  - Обучение 21 день на 8 GPU.

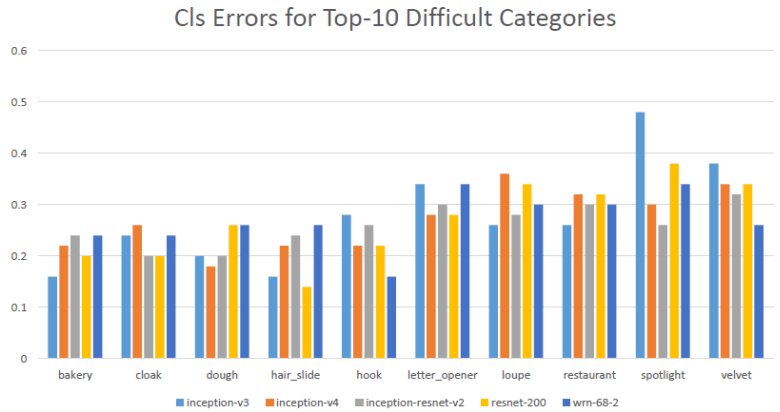




Победитель ILSVRC-2016. Trimps-Soushen

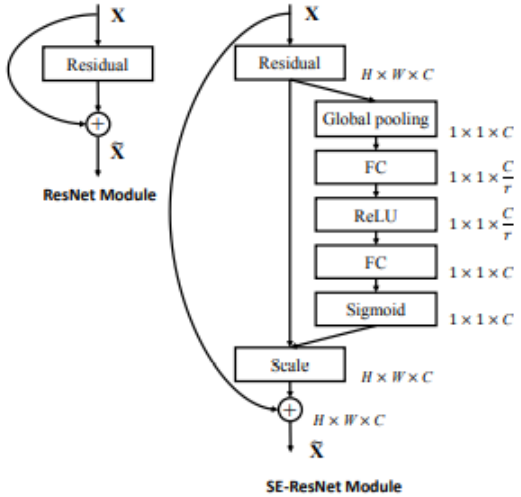
- Trimps-Soushen
  - Ошибка Top-5: **2.99%**;
  - Использование предобученных сетей для выявления Top-10 наиболее сложных категорий в ImageNet;
  - Inception (v3, v4), Inception-ResNet-v2, Pre-Activation ResNet-200, Wide ResNet;
  - Нет лучшей сети для всех категорий из Top-10 + слабая коррелированность решений → объединение решений.

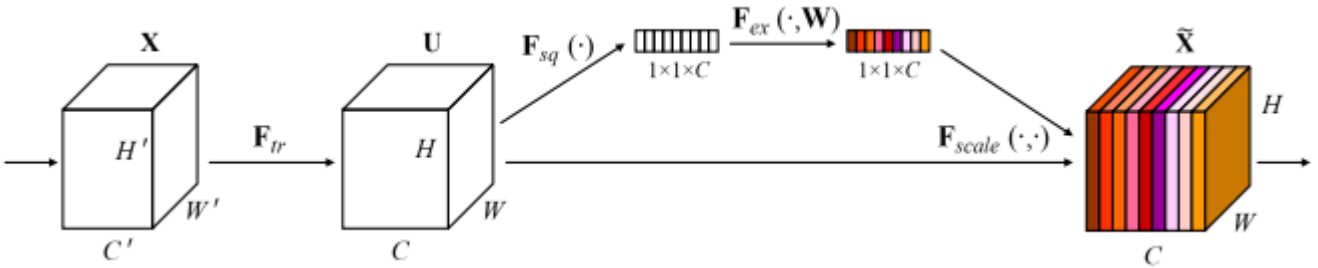
	Inception-v3	Inception-v4	Inception-Resnet-v2	Resnet-200	Wrn-68-3	Fusion（Val.）	Fusion（Test）
Err. (%)	4.20	4.01	3.52	4.26	4.65	2.92 (-0.6)	2.99



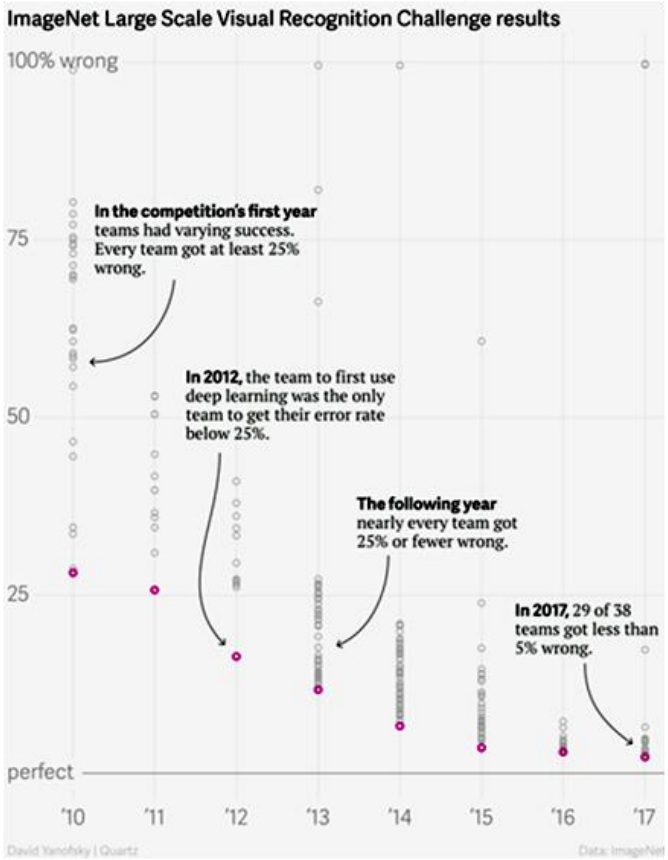
Победитель ILSVRC-2017. SENet

- SENet-154 – (J. Hu et al, 2017)
  - Ошибка Top-5: **2.25%**;
  - Взвешивание карт признаков;
  - «Squeeze and Excitation» (SE) модуль;
  - Интеграция в различные существующие архитектуры: Inception, ResNet и т.д.
  - Победитель SE+ResNeXt;
  - Обучение на 64 GPU для поддержки пакетов размером 2048 изображений.



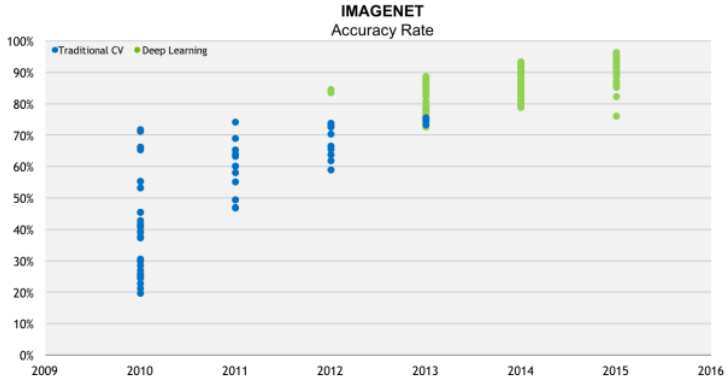


Статистика ILSVRC



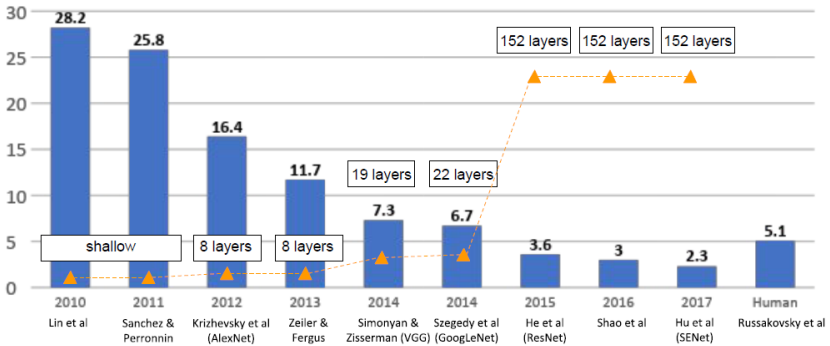
Сеть	Ошибка Top-5
AlexNet	15.3%
ZFNet	11.2%
VGG	7.3%
GoogLeNet	6.7%
ResNet	3.57%
Trimps-Soushen	2.99%
SENet-154	2.25%

Развитие НС для ILSVRC

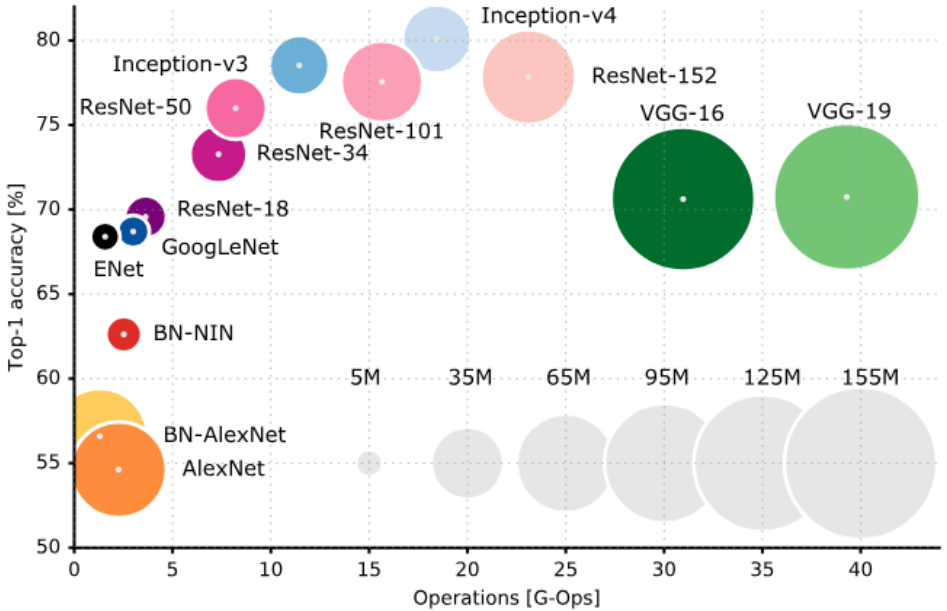


https://blogs.nvidia.com/blog/2016/06/29/deep-learning-6/

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture09.pdf



https://culurciello.github.io/tech/2016/06/20/training-enet.html

Рекуррентные НС

Решаемые задач

Обработка данных, составленных в последовательности.

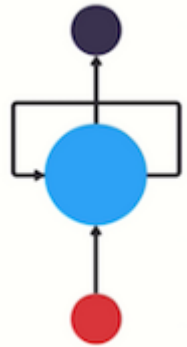
- ❑Распознавание речи (речь-в-текст)
- ❑Генерация речи (текст-в-речь)
- ❑Перевод с одного языка на другой
- ❑Предсказание биржевых индексов
- ❑Составление описаний изображений (изображение-в-текст)

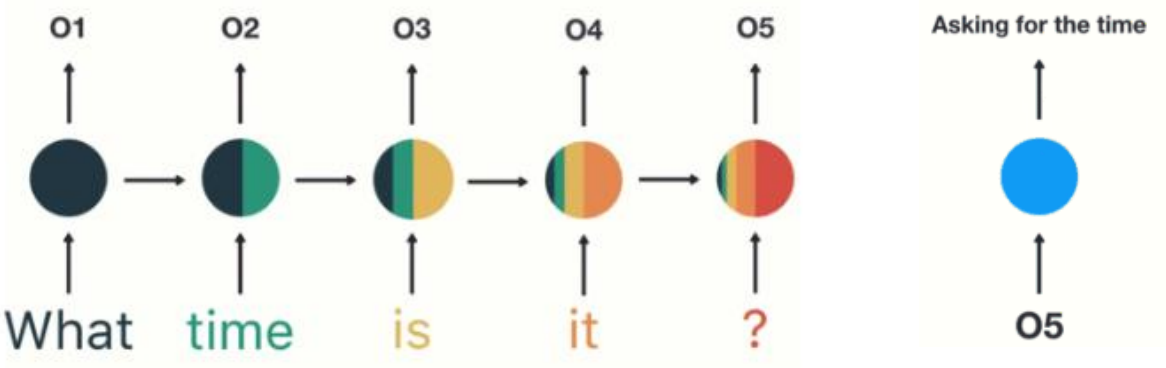


https://github.com/danieljl/keras-image-captioning

Структура рекуррентного нейрона

- ❑ Введение обратной связи для передачи информации о предшествующих входах (состояние нейрона).
- ❑ Соответствует многослойному персептрону, каждый слой имеет собственный вход, разделяемые веса.
- ❑ Вычисление признакого представления входной последовательности (рекуррентные слои) → классификация (персептрон)
- ❑ Обучение – back-propagation through time.





https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9

Back-propagation through time

- Дано: последовательности входов и заданных выходов сети.
- Шаг алгоритма BPTT:
  - «Развернуть» сеть и вычислить последовательность выходов
  - Вычислить функцию ошибку для всех выходов
  - Обновить веса, выполнив обратный проход.
- Вычислительно сложный для длинных последовательностей.

- Шаг алгоритма Truncated BPTT, входная последовательность длины K1:
  - «Развернуть» сеть и вычислить последовательность выходов для K2 входов
  - Вычислить функцию ошибку для всех выходов
  - Обновить веса, выполнив обратный проход.
- Подбор значений K1 и K2.

Проблема исчезающего градиента

- ❑ Уменьшение значения градиента от слоя к слою.
- ❑ Проблема для многослойных сетей → для рекуррентных тоже.
- ❑ Ранние слои отвечают за обработку начальных элементов последовательности → малые значения градиента, небольшая коррекция весов.
- ❑ «Забывание» долгосрочных зависимостей.

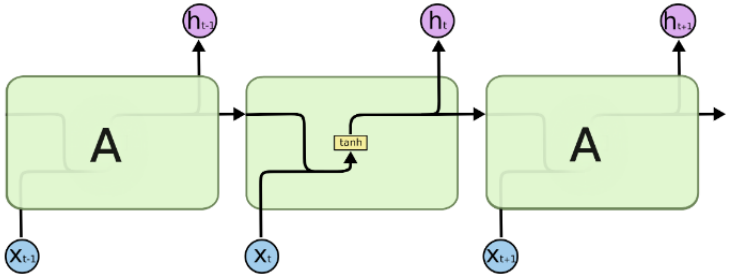


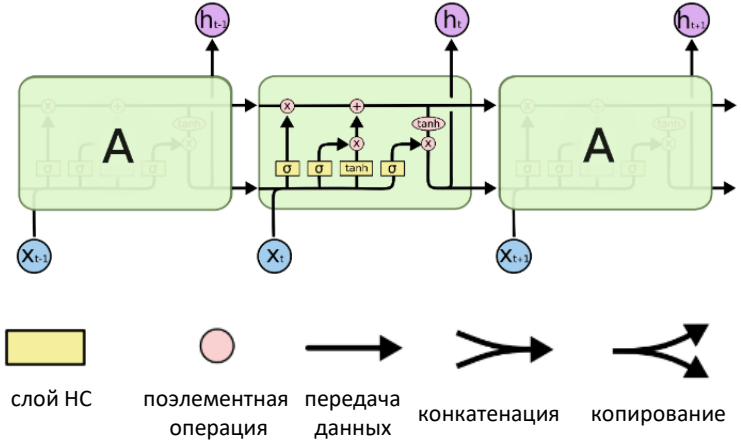
▪Влияние первых слов в последовательности на выход последнего нейрона стремится к 0

https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9

LSTM нейрон

- LSTM – Long Short Term Memory.
- Разработан для запоминания долгосрочных зависимостей.

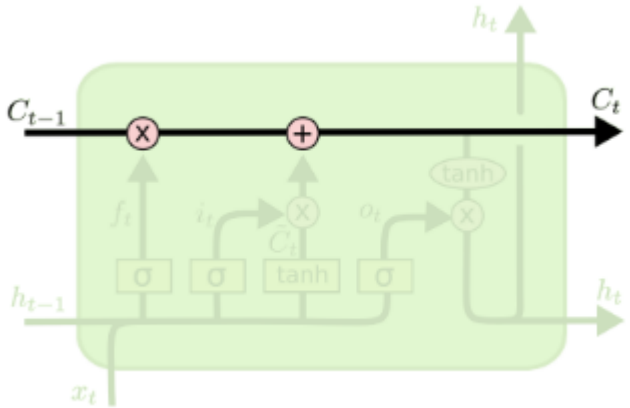


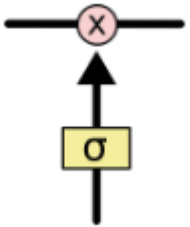


https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Состояние и вентили (gates)

- ❑ Вектор состояния сохраняет контекст (временные зависимости).
- ❑ Gate (вентиль) контролирует добавление и удаление информации из вектора состояния.
- ❑ Выход вентиля формируется сигмоидой.
- ❑ Типы вентилей: добавление информации (input gate), забывание информации (forget gate), формирование выхода LSTM нейрона (output gate).

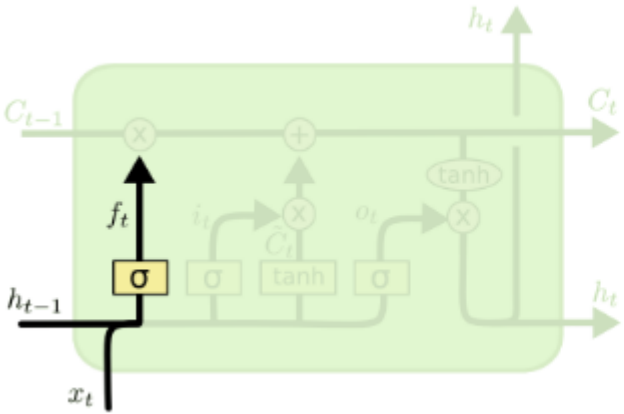




https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Forget gate

❑ Forget gate – вентиль для контроля «забывания» информации в векторе состояний.

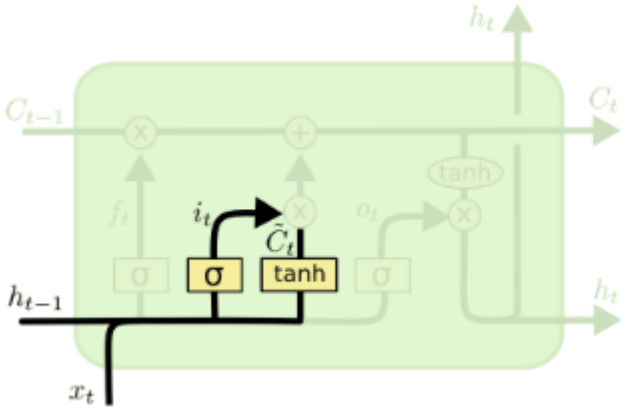


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Input gate

❑ Input gate – вентиль для определения, в каком виде входная информация в текущий момент времени войдет в вектор состояния.



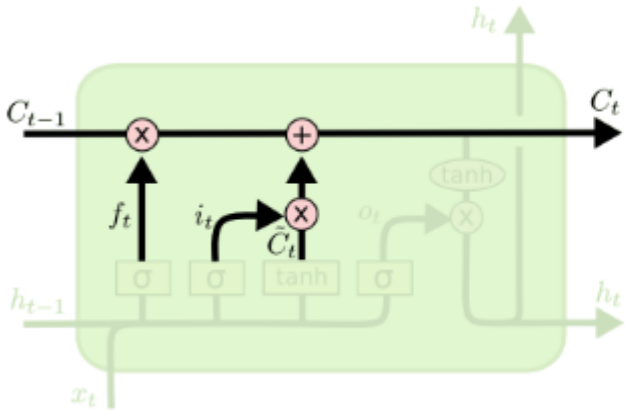
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Обновление вектора состояния

❑ Вектор состояний в текущий момент времени – линейная комбинация его предыдущего варианта и нового кандидата, полученного по входной информации.

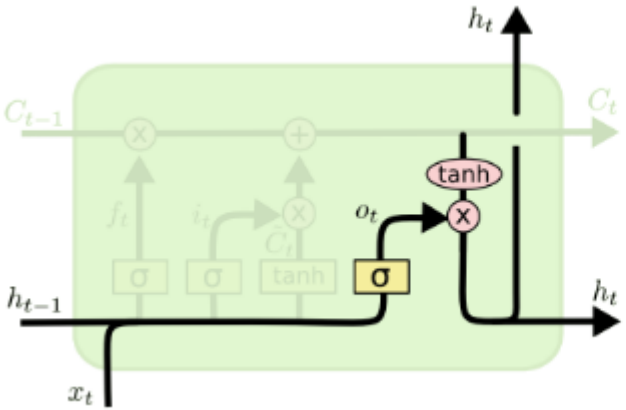


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Выходное значение

❑ Выходное значение определяется выходом традиционного полносвязанного слоя, умноженного на преобразованный вектор состояний.



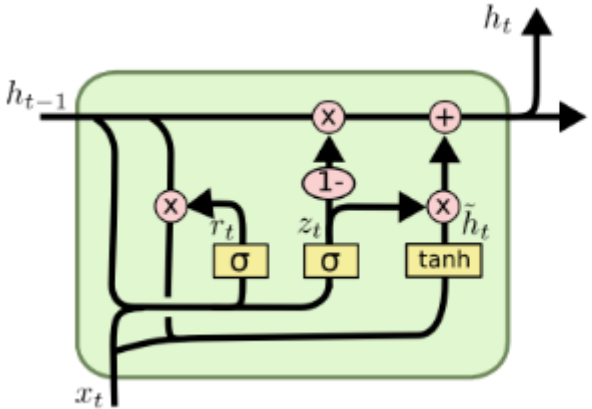
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gate Recurrent Unit

- Gate Recurrent Unit (GRU) – упрощенный вариант LSTM нейрона.
- Уменьшено число вентилей.
- Совмещение вектора состояний и выхода нейрона.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

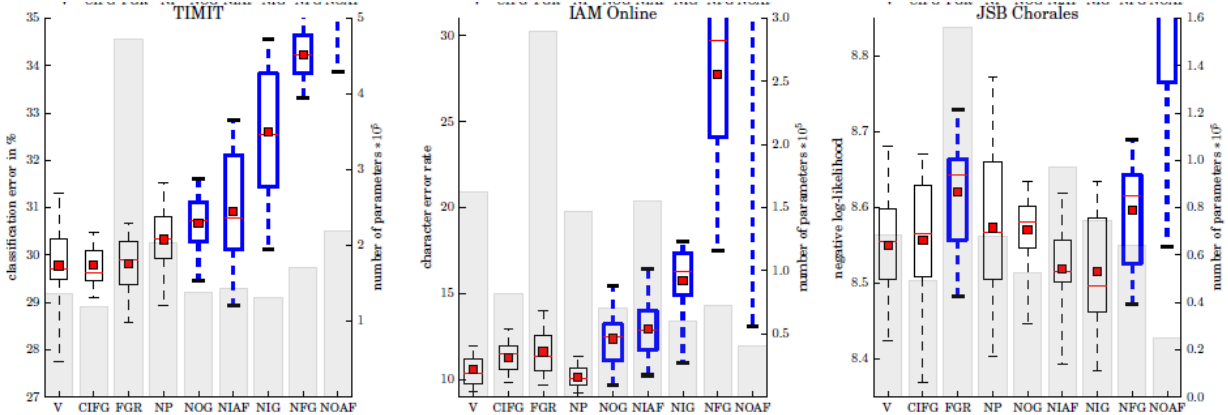
Сравнение рекуррентных нейронов

❑ Сравнение различных конфигураций LSTM нейрона: отключение вентилей, исключение нелинейных функций активации.

Одна из конфигураций реализует GRU (CIFG).

❑ 3 тестовые задачи: распознавание речи (TIMIT), распознавание рукописных предложений (IAM online), предсказание мелодии (JSB Chorales, Бах).

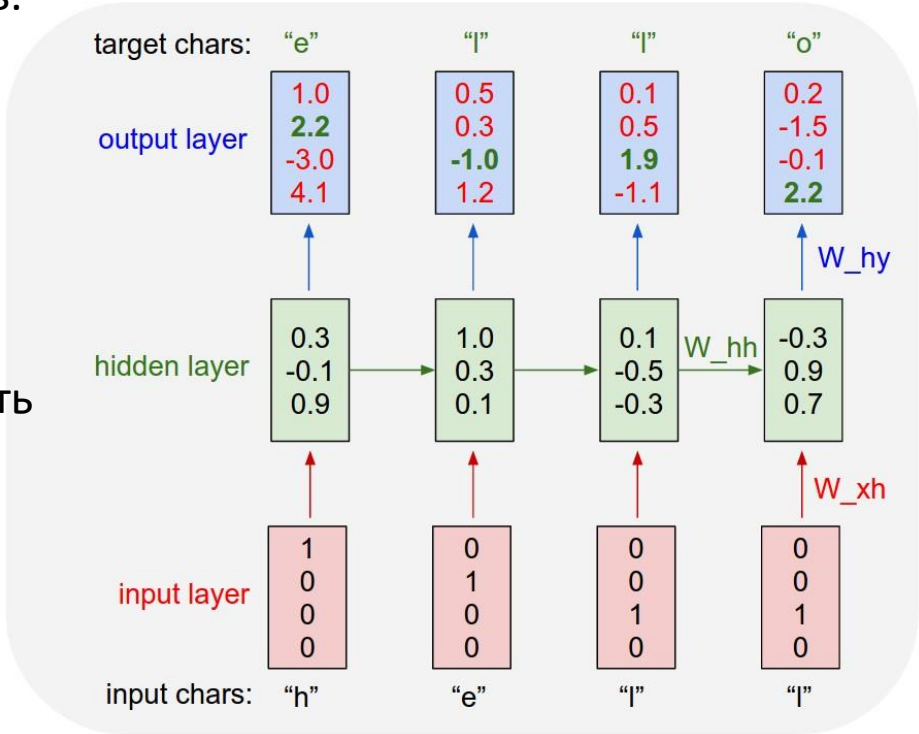
❑ Ни одна из конфигураций не является абсолютно лучшей. Ряд вариантов вычислительно более эффективные.



https://arxiv.org/pdf/1503.04069.pdf

Пример построения

- Задача: посимвольная генерация слов.
  Алфавит: 4 буквы (h, e, l, o).
- Архитектура:
  Вход – 4-мерный бинарный вектор (1-из-k).
  Внутренний слой – 3 рекуррентных нейрона
  Выход – 4-мерный вектор (уверенность появления буквы).
- Обучение: генерация слова “hello”



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Примеры генераторов текста

- Задача: генерация текста
- Архитектура:
  Вход – текущий символ: бинарный вектор (1-из-k).
  Внутренний слой – 3 слоя по 512 LSTM нейронов
  Выход – вектор степени уверенность появления слова из словаря.
- Обучение: набор текстов.
- Примеры генераторов:
  - произведения Шекспира
  - учебник по алгебраической геометрии
  - исходный код Линукса

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Генератор «произведений» Шекспира

- Обучающая выборка – все произведения Шекспира (4.4Мб)

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
```

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Генератор «алгебраическая геометрия»

- Обучающая выборка – учебник по алгебраической геометрии в Latex.

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram

$$\begin{array}{ccc} \mathcal{S} & \longrightarrow & \\ \downarrow & & \\ \xi & \longrightarrow & \mathcal{O}_{X'} \\ & & \uparrow \\ \mathrm{gor}_s & & \end{array} \qquad \begin{array}{ccc} & = \alpha' & \longrightarrow \\ & \updownarrow & \\ & = \alpha' & \longrightarrow \alpha \end{array} \qquad \begin{array}{c} X \\ \downarrow \\ \mathrm{d}(\mathcal{O}_{X_{Jk}}, \mathcal{G}) \end{array}$$

$$\mathrm{Spec}(K_\psi) \qquad \mathrm{Mor}_{Sets}$$

is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_i}^{-1} \mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Генератор «С код»

- Обучающая выборка - исходный С код Линукса (474Мб).

```
static void num_serial_settings(struct tty_struct *tty)
{
  if (tty == tty)
    disable_single_st_p(dev);
  pci_disable_spool(port);
  return 0;
}

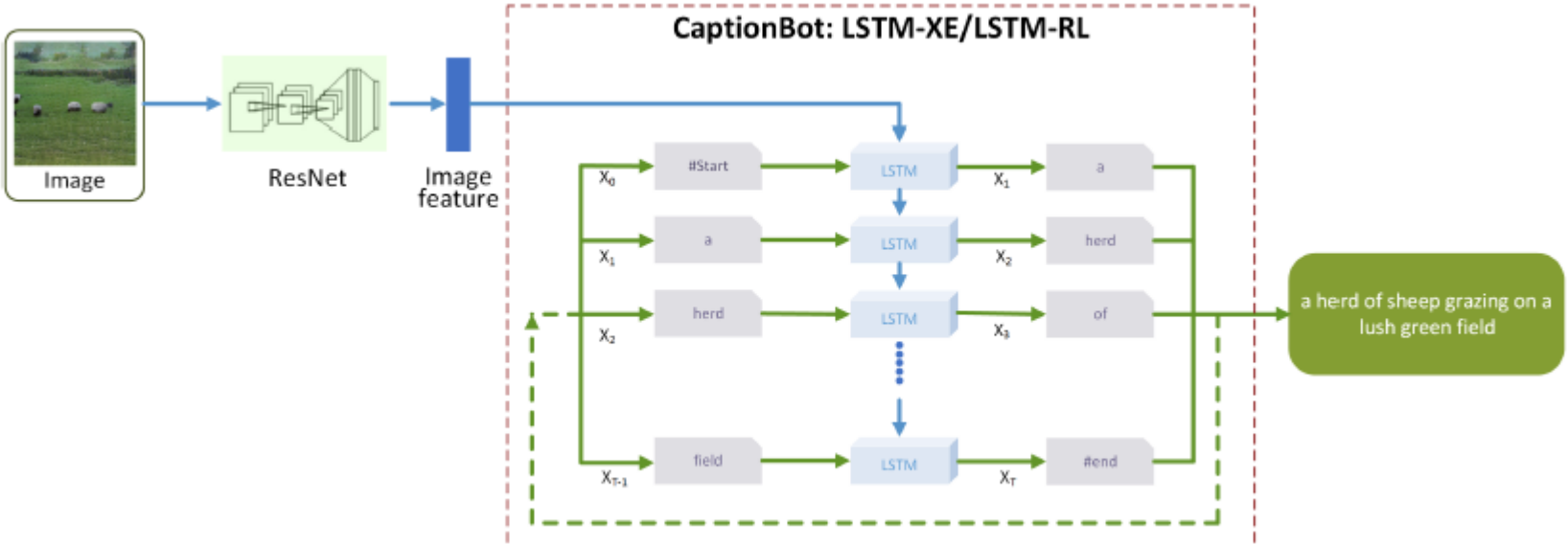
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000fffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
```

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Составление описания изображения

- Задача: генерация словесного описания изображения (Image captioning).
- Вход: изображение
- Выход: вектор «вероятностей» появления каждого слова из словаря



Q. Huang, et al., Turbo Learning for CaptionBot and DrawingBot, 2018

ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ ГРАФИЧЕСКИХ ПРОЦЕССОРОВ (GPU) NVIDIA. ОСНОВНЫЕ РАЗЛИЧИЯ CPU И GPU.

АФАНАСЬЕВ ИЛЬЯ
AFANASIEV_ILYA@ICLOUD.COM

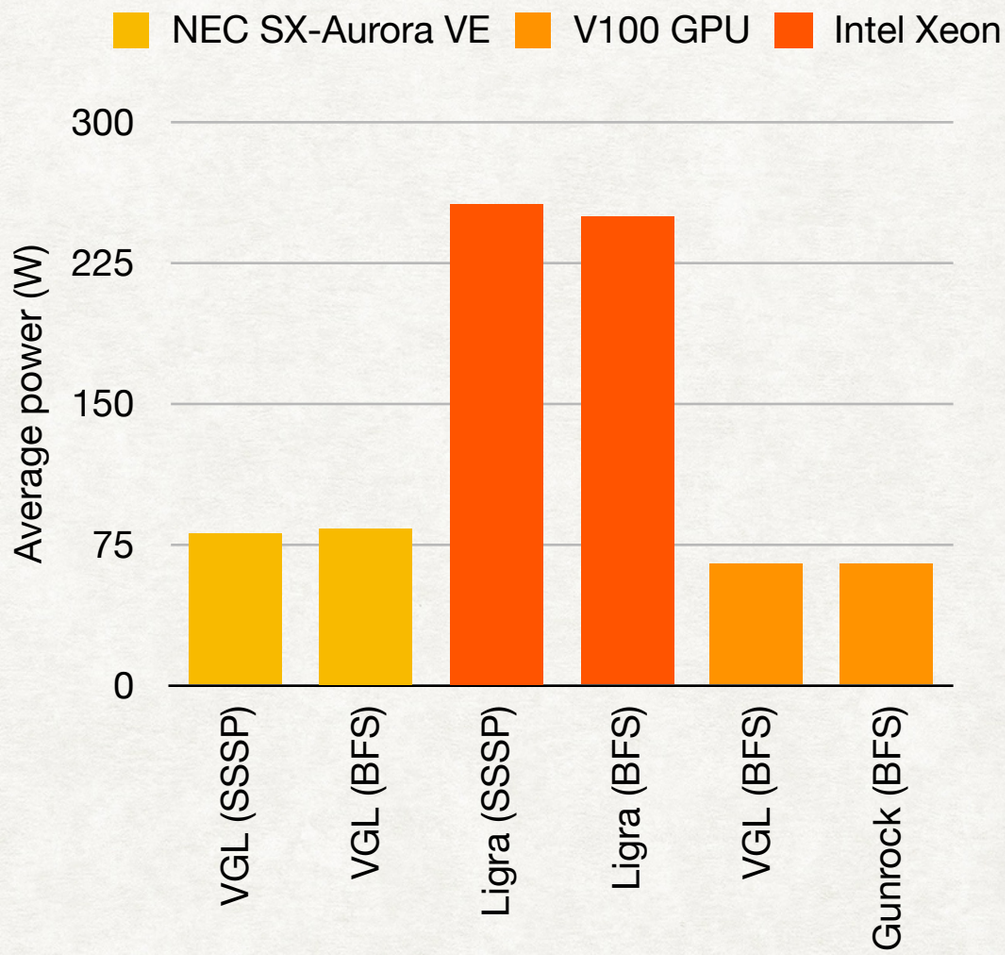
GPGPU & CUDA

- GPU - Graphics Processing Unit (графический ускоритель)
- GPGPU - General-Purpose computing on GPU, вычисления общего вида на GPU;
- Первые GPU от NVIDIA с поддержкой GPGPU – GeForce восьмого поколения, G80 (2006 г);
- CUDA – Compute Unified Device Architecture, Программно-аппаратная архитектура от Nvidia, позволяющая производить вычисления с использованием графических процессоров

ОСНОВНЫЕ ПРЕИМУЩЕСТВА GPU ПО СРАВНЕНИЮ С CPU

- Высокое соотношение цена / производительность
- Высокое соотношение производительность / энергопотребление

Architecture	Memory Bandwidth	VRAM	GPU
Ampere	1555GB/sec	40GB	GA100
Volta	900GB/sec	16GB/32GB	GV100
Pascal	720GB/sec	16GB	160GB/ sec



- Память с высокой пропускной способностью (до 1.5 TB/s)

TOP500 (ИЮНЬ 2020)

1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science	7,299,072	415,530.0	513,854.7	28,335
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox	1,572,480	94,640.0	125,712.0	7,438
4	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi	10,649,600	93,014.6	125,435.9	15,371
5	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT National Super Computer Center in Guangzhou	4,981,760	61,444.5	100,678.7	18,482
6	**HPC5** - PowerEdge C4140, Xeon Gold 6252 24C 2.1GHz, NVIDIA Tesla V100, Mellanox HDR Infiniband, Dell EMC Eni S.p.A.	669,760	35,450.0	51,720.8	2,252
7	**Selene** - DGX A100 SuperPOD, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation	272,800	27,580.0	34,568.6	1,344
8	**Frontera** - Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR, Dell EMC Texas Advanced Computing Center/Univ. of Texas	448,448	23,516.4	38,745.9	
9	**Marconi-100** - IBM Power System AC922, IBM POWER9 16C 3GHz, Nvidia Volta V100, Dual-rail Mellanox EDR Infiniband, IBM	347,776	21,640.0	29,354.0	1,476
10	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100, Cray/HPE Swiss National Supercomputing Centre (CSCS)	387,872	21,230.0	27,154.3	2,384

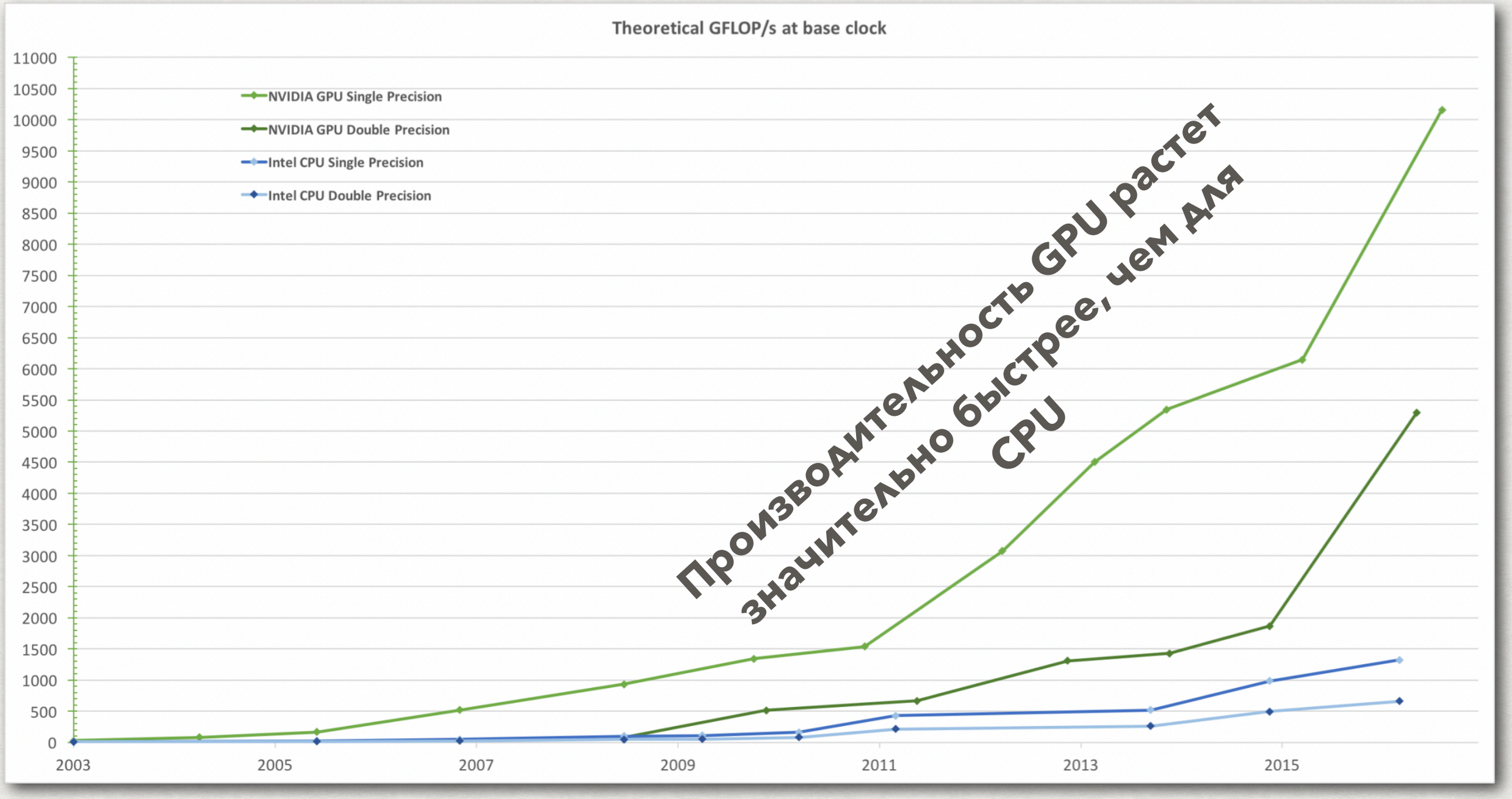
6/10 первых суперкомпьютеров top500 используют NVIDIA GPU

GREEN500 (ИЮНЬ 2020)

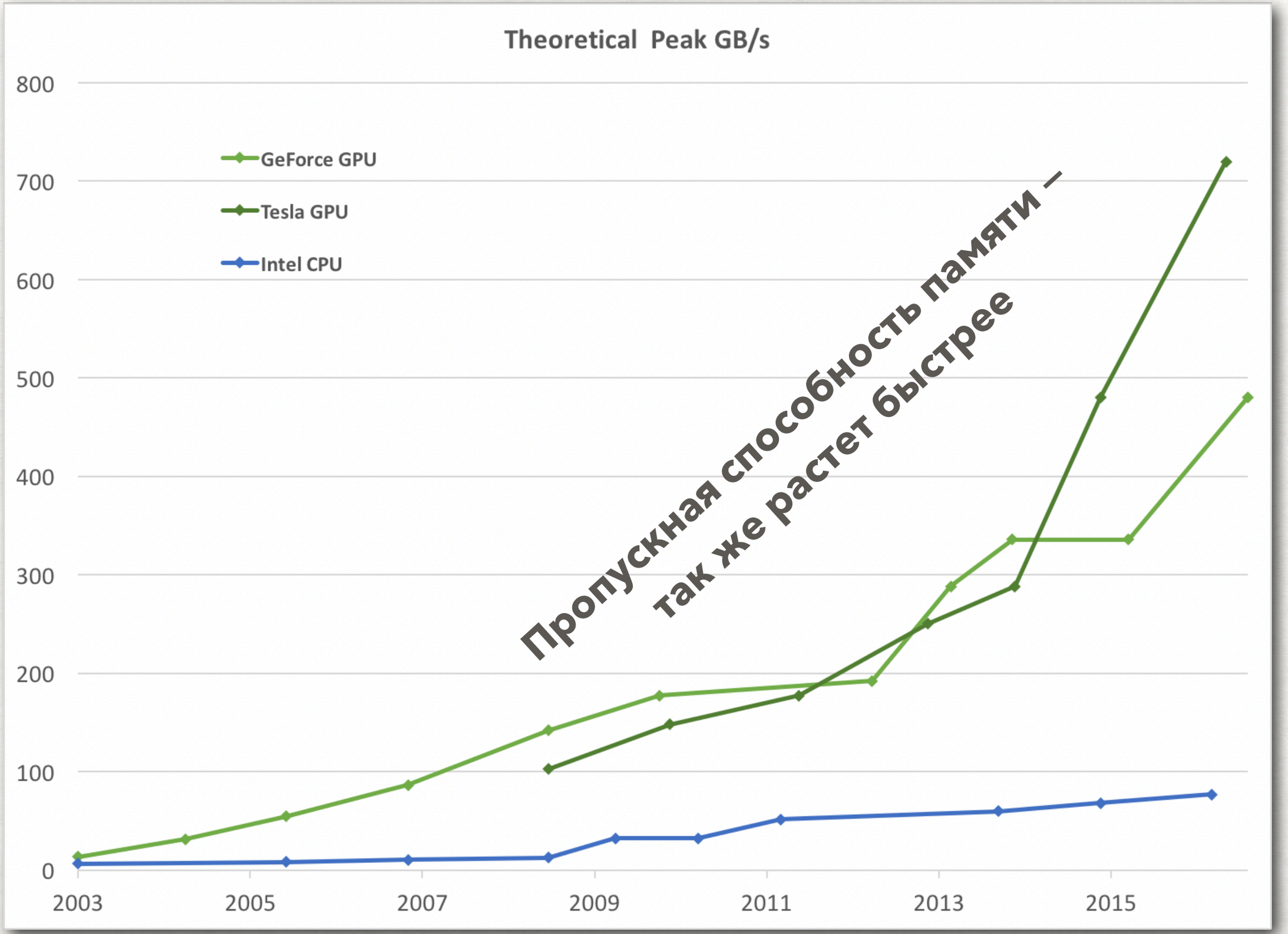
1	393	**MN-3** - MN-Core Server, Xeon 8260M 24C 2.4GHz, MN-Core, RoCEv2/MN-Core DirectConnect, Preferred Networks	2,080	1,621.1	77	21.108
2	7	**Selene** - DGX A100 SuperPOD, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia	272,800	27,580.0	1,344	20.518
3	468	**NA-1** - ZettaScaler-2.2, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 700Mhz, PEZY Computing / Exascaler Inc.	1,271,040	1,303.2	80	18.433
4	204	**A64FX prototype** - Fujitsu A64FX, Fujitsu A64FX 48C 2GHz, Tofu interconnect D, Fujitsu Fujitsu Numazu Plant	36,864	1,999.5	118	16.876
5	26	**AiMOS** - IBM Power System AC922, IBM POWER9 20C 3.45GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM	130,000	8,339.0	512	16.285
6	6	**HPC5** - PowerEdge C4140, Xeon Gold 6252 24C 2.1GHz, NVIDIA Tesla V100, Mellanox HDR Infiniband, Dell EMC	669,760	35,450.0	2,252	15.740
7	421	**Satori** - IBM Power System AC922, IBM POWER9 20C 2.4GHz, Infiniband EDR, NVIDIA Tesla V100 SXM2, IBM	23,040	1,464.0	94	15.574
8	2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM	2,414,592	148,600.0	10,096	14.719
9	1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu	7,299,072	415,530.0	28,335	14.665
10	9	**Marconi-100** - IBM Power System AC922, IBM POWER9 16C 3GHz, Nvidia Volta V100, Dual-rail Mellanox EDR Infiniband, IBM	347,776	21,640.0	1,476	14.661

6/10 первых суперкомпьютеров green500 используют NVIDIA GPU

ВЫЧИСЛИТЕЛЬНАЯ МОЩНОСТЬ



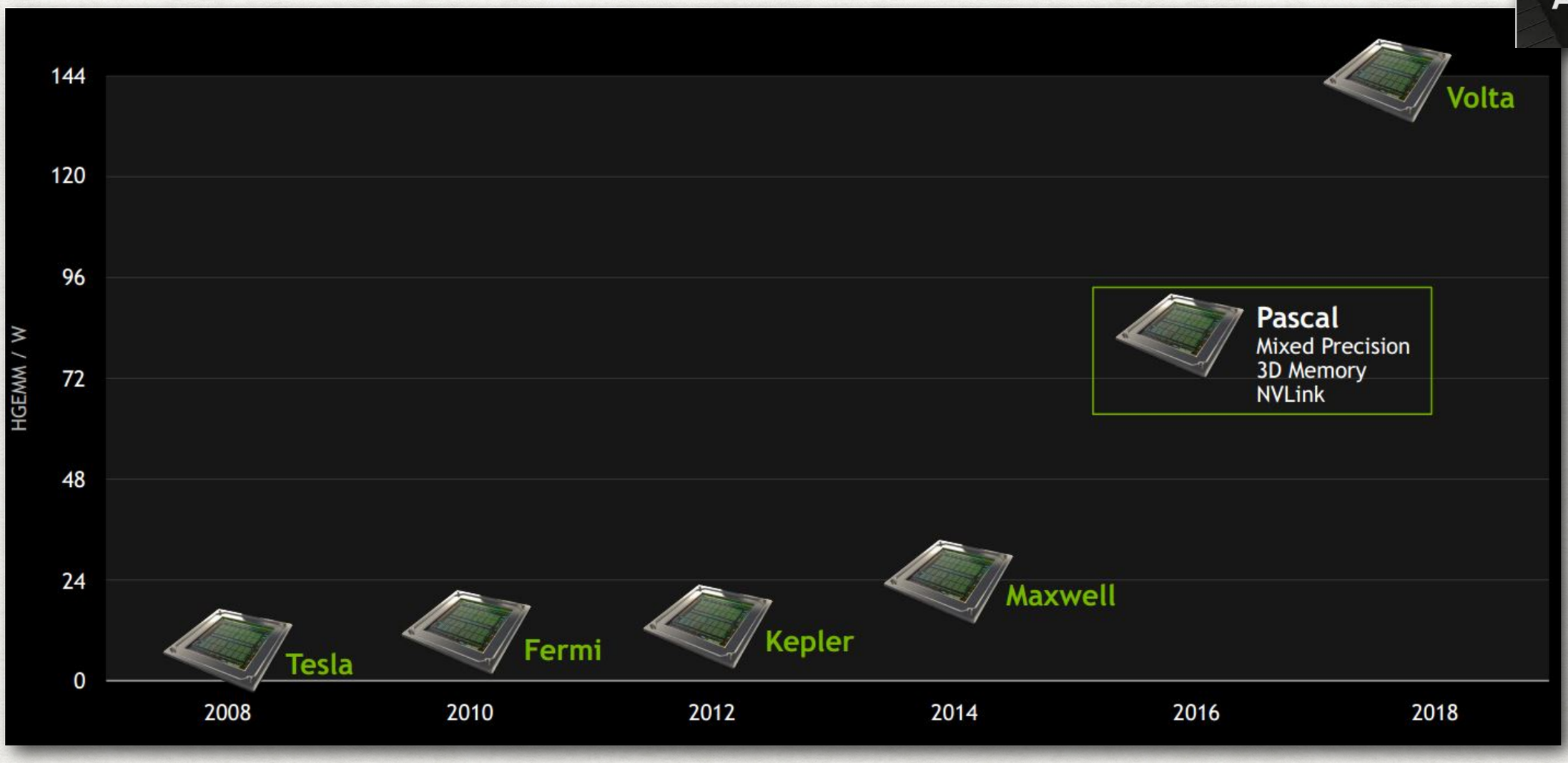
ПРОПУСКНАЯ СПОСОБНОСТЬ ПАМЯТИ



СЕМЕЙСТВА GPU УСТРОЙСТВ



GPU ROADMAP

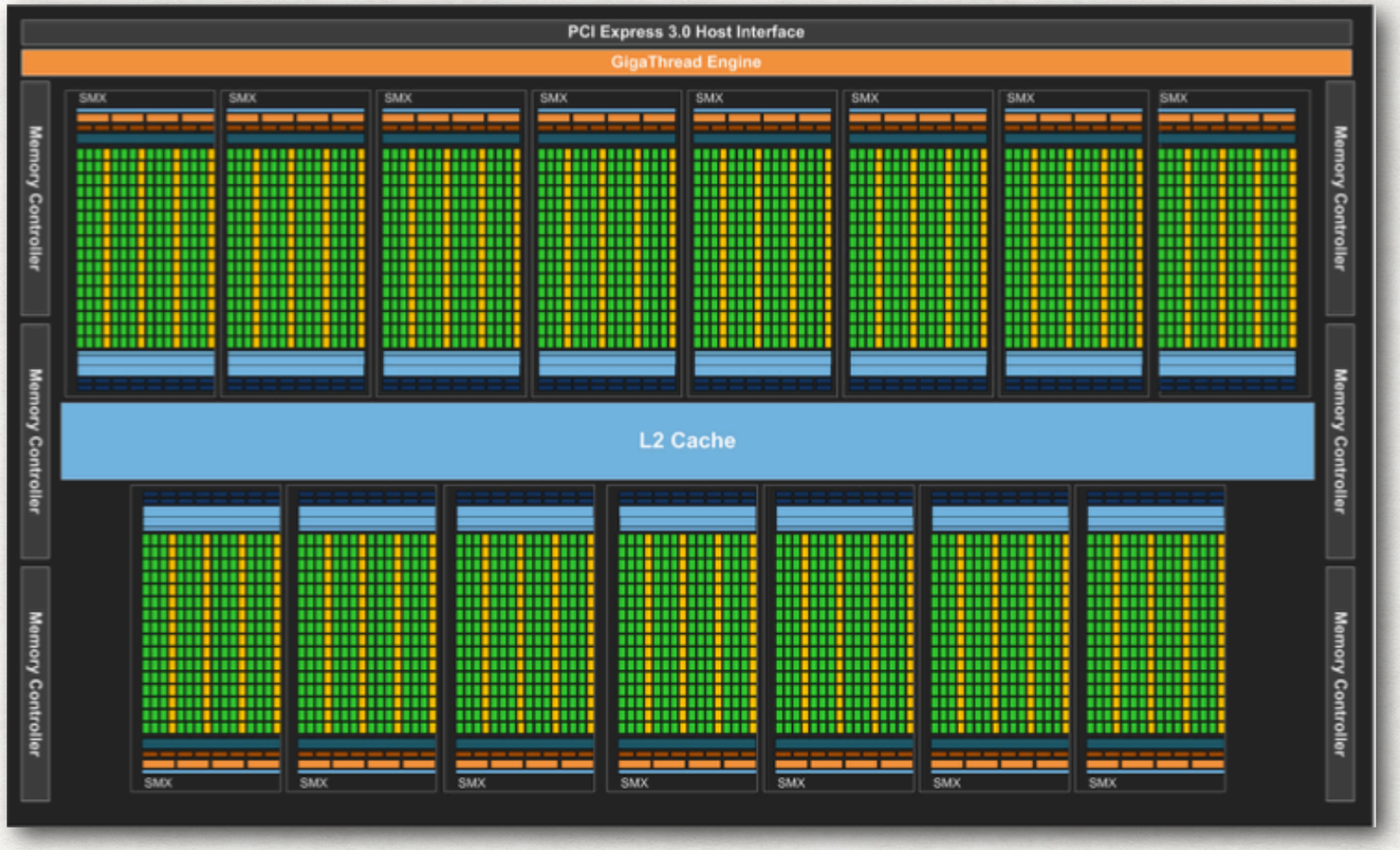

144
120
96
72
48
24
0
HGEMM / W
2008
2010
2012
2014
2016
2018
Tesla
Fermi
Kepler
Maxwell
Pascal
Mixed Precision
3D Memory
NVLink
Volta
GEFORCE RTX
AMPERE

КАКИМИ ОСОБЕННОСТЯМ ОБЛАДАЮТ СОВРЕМЕННЫЕ GPU?

АРХИТЕТКРА KEPLER

Особенности Kepler (на примере K40):

- 15 SMX (потоковые мультипроцессоры), 2880 CUDA-ядер
- 1536 kB L2 кэш
- Интерконнект – PCI Express 3 (15 GB/s)



4.29 TFLOPS – одинарная точность

1.43 TFLOPS – двойная точность

- Hyper Q – 32 аппаратных очереди для multitasking
- GPU Direct – прямое копирование между host <-> device
- Динамический параллелизм (вложенный вызов ядер)

АРХИТЕТКРА PASCAL

Особенности Pascal (на примере P100):

- 56 SMs, 3584 FP32 CUDA-cores
- HBM2 память (до 16 GB, до 720 GB/s)
- L2 кэш размера 4 MB
- Интерконнект – NVLink (80 GB/s)

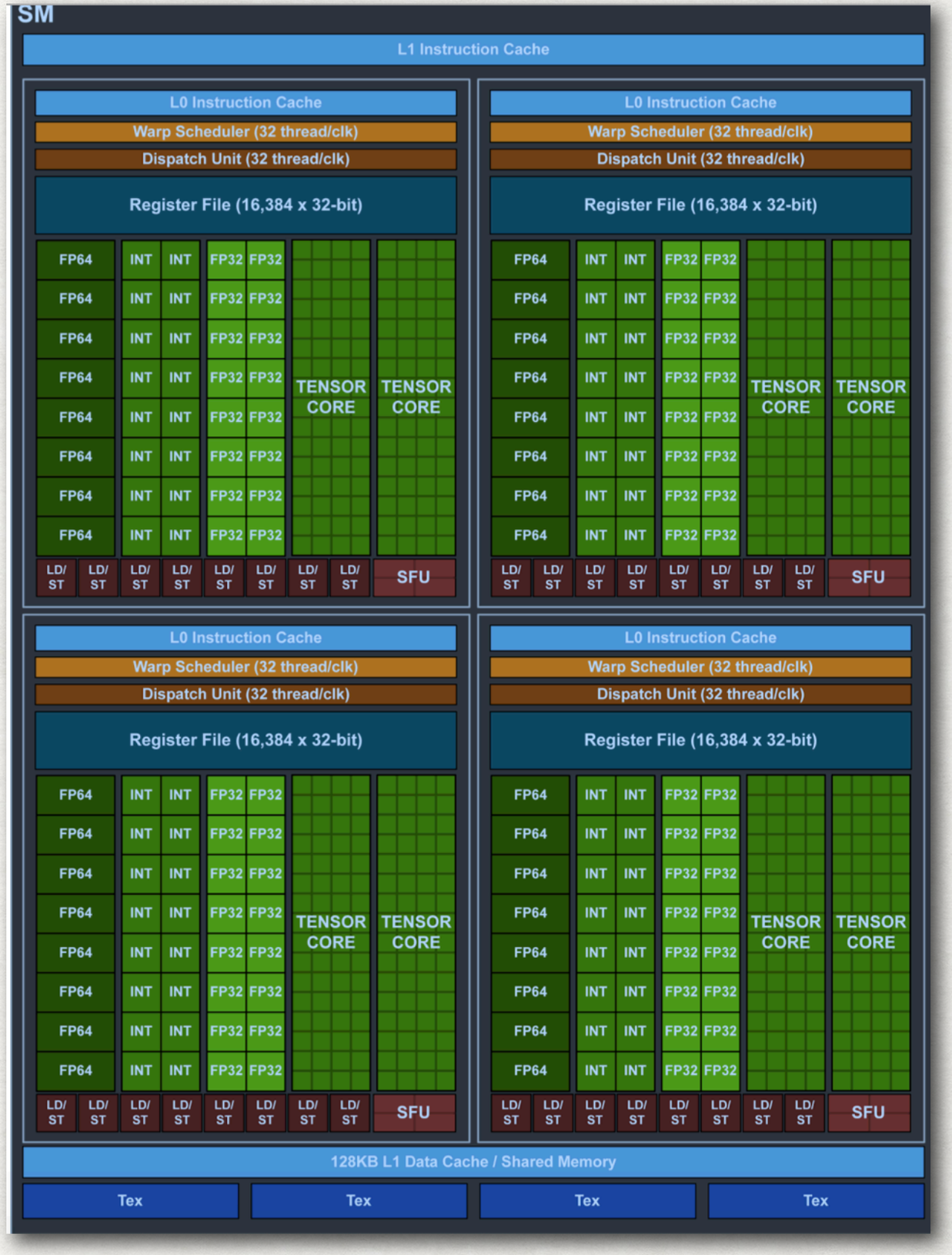


- 5,3 TFLOPS of double precision
- 10,6 TFLOPS of single precision
- 21,2 TFLOPS of half-precision – вычисления с половинной точностью для DL

АРХИТЕТКРА VOLTA

Особенности Volta (на примере V100):

- 84 SMs, 5120 FP32 CUDA-cores, 640 Tensor-cores
- HBM2 память (до 32 GB, до 900 GB/s)
- L2 кэш размера 16 MB
- Интерконнект – NVLink 2.0 (160 GB/s)
- 7 TFLOPS of double precision
- 14 TFLOPS of single precision
- 112 TFLOPS tensor cores, half precision – DL



АРХИТЕТКРА AMPERE

Особенности Ampre (на примере A100):

- 84 SMs, 8192 FP32 CUDA-cores, 512 Tensor-cores (3d generation)
- HBM2 память (до 40 GB, до 1.5 TB/s)
- L2 кэш размера 40 MB, программный prefetching данных
- Интерконнект – NVLink 3.0 (320 GB/s)
  - TF32 Tensor Core для обработки FP32 data
  - Multi-Instance GPU
- 9.7 TFLOPS of double precision
- 19.5 TFLOPS of single precision
- 312 TFLOPS tensor cores, half precision



СРАВНИТЕЛЬНЫЕ ХАРАКТЕРИСТИКИ АРХИТЕКТУР GPU

- Основные тенденции: увеличение производительности, пропускной способности памяти, объема памяти, пропускной способности интерконнекта, упрощение написания программ (атомарные операции, unified memory, динамический параллелизм), поддержка вычислений с половинной точностью

	Fermi	Kepler GK110	Maxwell GM100	Maxwell GM200	Pascal P100
Streaming Multiprocessors *	16	15	16	24	56
PCI-e Version	v2	v3	v3	v3	v3, NVLink
L2 cache, KB *	768	1536	2048	3072	4096
CUDA Cores (per SM)	32	192	128	128	64
LD/ST Units (per SM)	16	32	32	32	16
SFUs (per SM)	4	32	32	32	16
Warp schedulers (per SM)	2	4	4	4	2
Shared Memory (per SM), KB	16/48	16/32/48	96	96	64
Registers (per SM)	32K	65K	65K	65K	65K
Active Blocks per SM	8	16	32	32	32
Memory Bandwidth, GB/s	144	250	336	288	720

PCI-e bandwidth: v2 - 8 GB/s, v3 - 15.75 GB/s

Note: (*) parameters are specific amongst different GPU series and models.

В ЧЕМ ПРИНЦИПИАЛЬНЫЕ ОТЛИЧИЯ CPU ОТ GPU?

ТИПОВАЯ АРХИТЕКТУРА CPU

- Небольшое число (~4-30) мощных независимых ядер; ~3.0 ГГц каждое;
- Поддержка виртуальных потоков (Hyper-Threading)
- 3х уровневый кеш (L1 + L2 + L3)
- Разделяемый между ядрами кеш L3 до 50 МБ
- Приватные кэши L1=32KB (data) + 32KB (Instructions), L2=256KB
- Обращения к памяти обрабатываются **отдельно** для каждого процесса\нити

ТИПОВАЯ АРХИТЕКТУРА GPU (НА ПРИМЕРЕ PASCAL)

- В основе построения GPU лежат потоковые мультипроцессоры (SM, SMX), 84 штуки
- SM - «Единица» построения устройства (как ядро в CPU)
- В состав каждого потокового мультипроцессора входит большое количество легковесных ядер



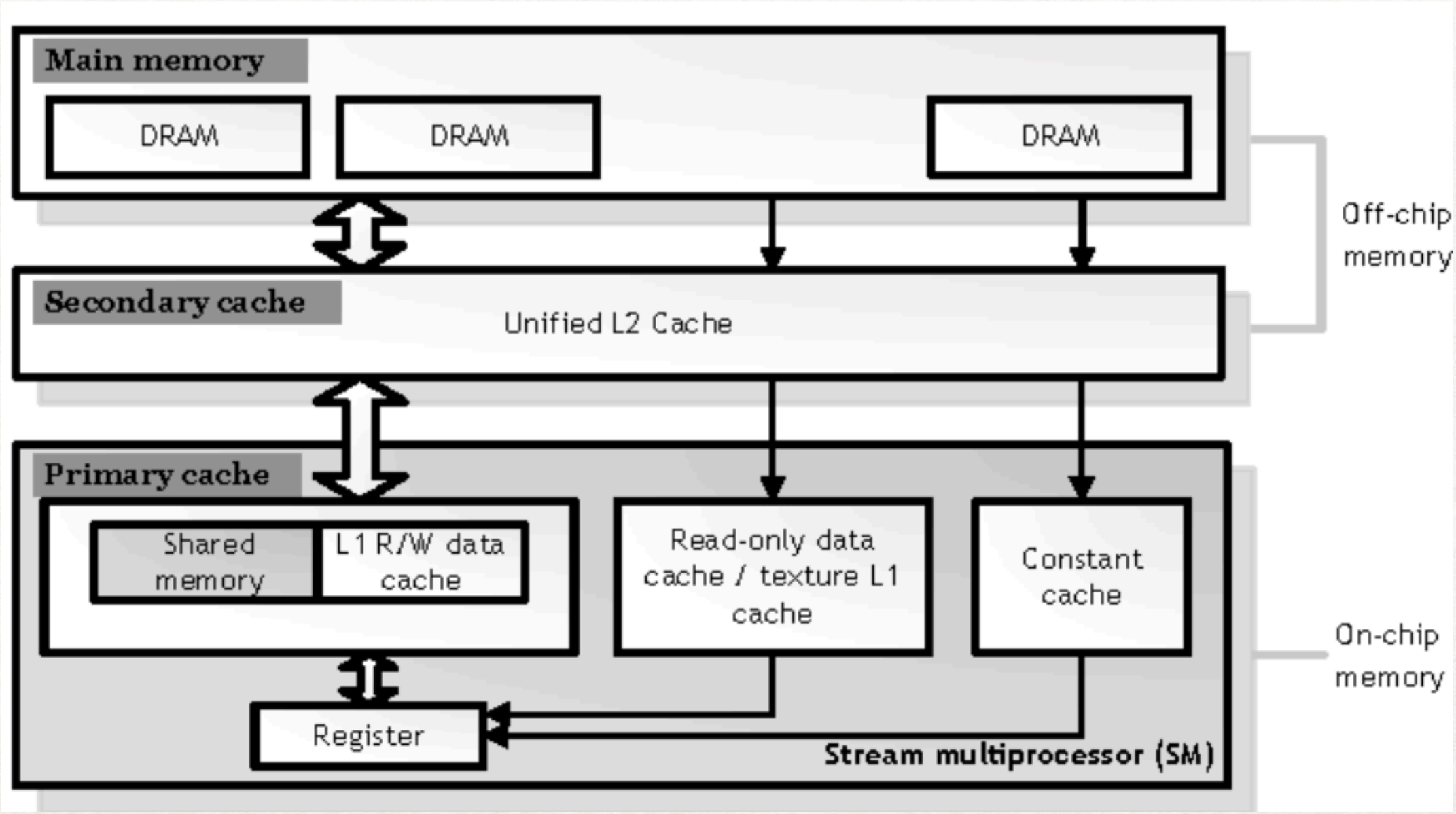
АРХИТЕКТУРА ПОТОКОВОГО МУЛЬТИПРОЦЕССОРА (PASCAL)

- Потоковый мультипроцессор - «Единица» построения устройства (как ядро в CPU)
- 64 скалярных ядра CUDA Core, ~1.5ГГц
- CUDA ядра имеют разный тип для различных типов операций (FLT, DBL, LT/ST, и др.)
- 2 Warp Scheduler (планировщик варпов) – SIMD работа CUDA ядер (коллективная)
- Файлы регистров
- L1 кэша/разделяемая память



АРХИТЕКТУРА ПОТОКОВОГО МУЛЬТИПРОЦЕССОРА (PASCAL)

- Всего 2 уровня кэш-памяти, L1 размера 64 KB, L2 размера 4 MB
- L2 и device memory **разделяются между всеми мультипроцесорами**
- L1/shared/constant/texture - **приватные для каждого из мультипроцеесоров**



КАК РАБОТАЕТ GPU?

(УПРОЩЕННО)

- Итак, у нас есть **большое количество физических CUDA-ядер** (64*84=5376 для P100 GPU)
- **Каждое из яде**р поддерживает выполнение операций **определенного типа** (FP, DP, load/store/SFU…)
- Пользователем создается большое число виртуальных нитей (>> числа ядер), после чего каждое из физических ядер **обрабатывает команды определённого типа**, встречающиеся в потоке управления нитей
- GPU-нити работают принципиально отличным образом от CPU нитей – (1) SIMD, (2) коллективный доступ к иерархии памяти, (3) синхронизации между разными SM, разными CUDA-ядрами не всегда возможны/эффективны, и т.д.

ЗАЧЕМ СОЗДАВАТЬ НИТЕЙ БОЛЬШЕ, ЧЕМ ЕСТЬ ФИЗИЧЕСКИХ CUDA-ЯДЕР?

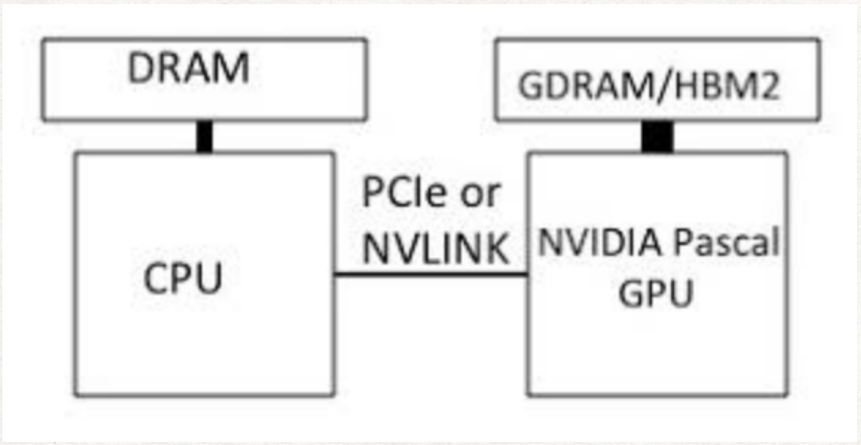
- **Цель**: эффективно загружать CUDA-ядра
- **Проблема**: латентность памяти – большинство команд требует загрузки операндов из подсистемы памяти
- **Решение**:
    - CPU: Сложная **иерархия кешей** (3-4 уровня) большого размера (до 100 МБ)
    - GPU: Много нитей, скрывать обращения одних нитей в память за вычислениями других нитей за счёт быстрого **переключения контекста**
        - как только часть нитей запрашивает данные из памяти, они «снимаются» с физических ядер и ожидают получения данных из памяти, в то время как на их место помещаются другие нити
        - при переключении контекста на GPU **не происходит сохранения в память значений регистров**, обуславливающих значительные накладные расходы при переключении контекстов на CPU
        - именно для этого на GPU нужно запускать нитей >> доступного количества физических CUDA-ядер
- За счёт наличия сотен ядер и поддержки миллионов нитей GPU может эффективно задействовать высокую теоретическую пропускную способность памяти (~1 TB/s)

ИТОГИ: СРАВНЕНИЕ GPU И CPU

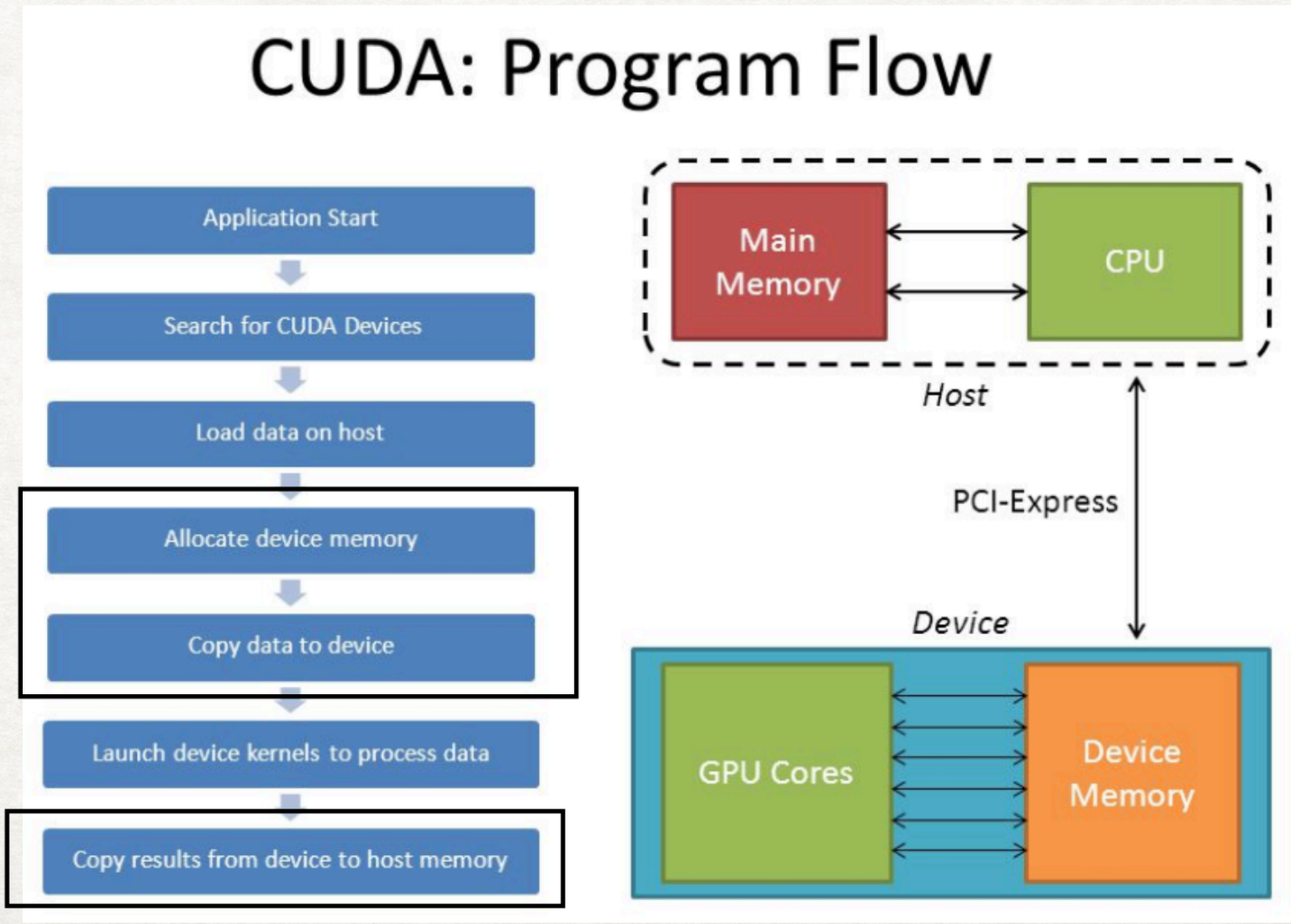
- **Сотни упрощённых вычислительных CUDA-ядер**, работающих на небольшой тактовой частоте ~1.8 ГГц
- **Небольшие кеши** на GPU
  - 64 CUDA-ядра разделяют 64 КБ L1
  - L2 общий для всех потоковых мультипроцессоров ~4 MB, L3 отсутствует
- Оперативная память с **высокой пропускной способностью** и **высокой латентностью**, оптимизированная для коллективного доступа
- Поддержка **миллионов виртуальных нитей**, **быстрое переключение контекста** для групп нитей

GPU — СОПРОЦЕССОР

- GPU имеет отдельную память и подключается к CPU при помощи PCI или NVLINK
- Интерконнект PCI/NVLINK часто является узким местом
- Начиная с архитектуры Pascal поддержка единого виртуального адресного пространства (Unified Memory) – пользователь может не заботиться о явных пересылках данных



Часть программы, отвечающая за пересылку данных!



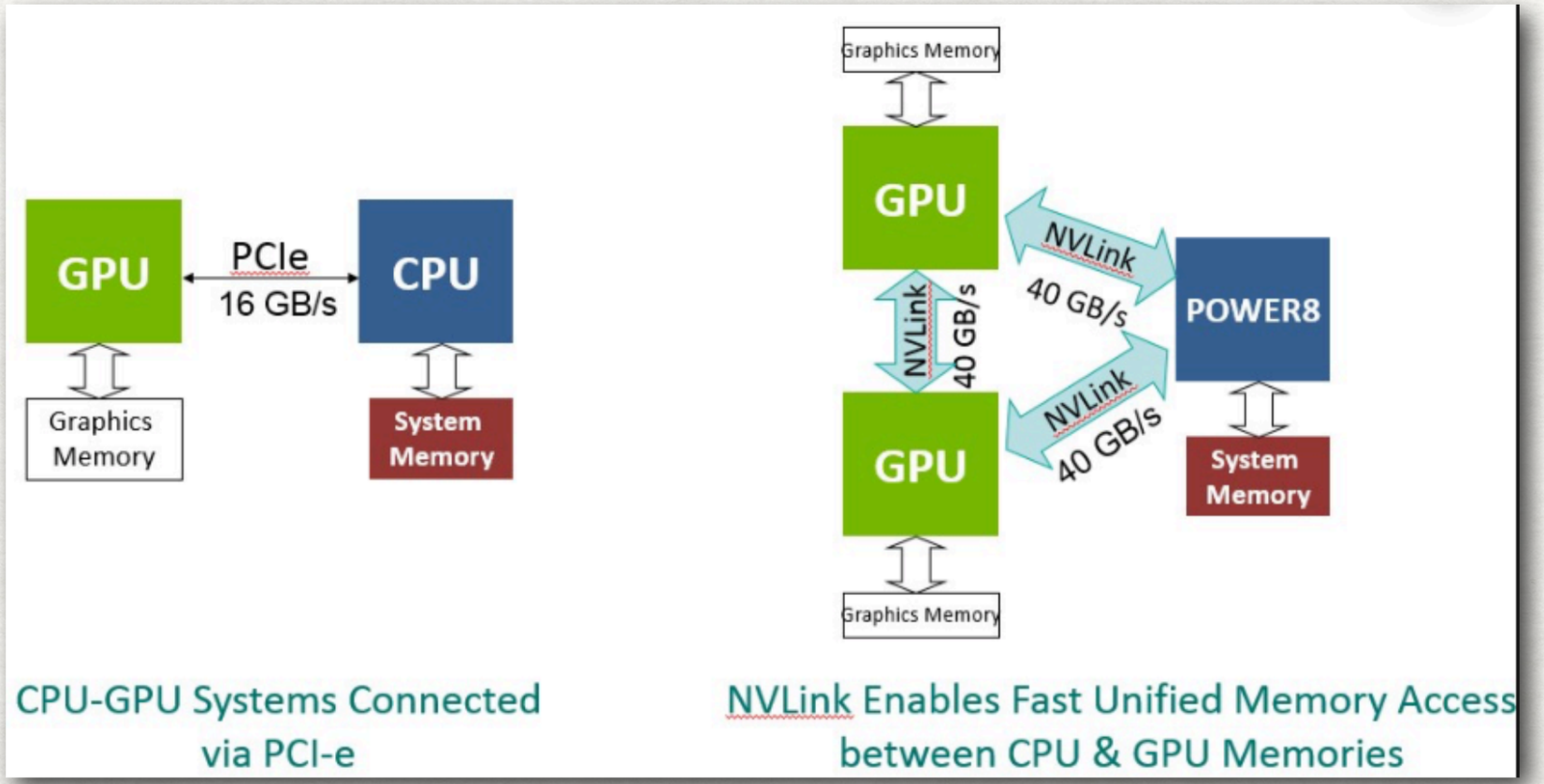
ПЕРЕСЫЛКИ ДАННЫХ CPU <-> GPU

- Когда происходят передачи данных GPU↔CPU?

  (1) при передаче входных/выходных массивов
  (2) при переходе к последовательным вычислениями
  (3) при загрузке бинарного кода ядер, передача параметров в ядра, и т.д.

- Какая пропускная способность современных интерконнектов?

- Как пересылки данных влияют на производительность?



ВЫЧИСЛИТЕЛЬНАЯ МОЩНОСТЬ (FLOPS/BYTE)

- Позволяет оценить, насколько приложение подходит для реализации на GPU с точки зрения **передачи данных по шине**
- Перед использованием GPU для реализуемого важно оценить соотношение: $\frac{FLOPS}{input_data_size + output_data_size}$ – **вычислительная мощность**
- при малых значениях вычислительной мощности получить значительное ускорение за счет использования GPU – зачастую невозможно!
- пример:
  - SAXPY: $\frac{2 \times n}{3 \times n} = \frac{2}{3}$ - not ok
  - умножение плотных матриц: $\frac{2 \times n^3}{2 \times n^2} = n$ - ok

CUDA TOOLKIT

- cuda toolkit - важный инструмент разработки CUDA-приложений (содержит библиотеки, компилятор, средства анализа эффективности CUDA-приложений и т.д.)
- Доступен для бесплатной загрузки по ссылке: https://developer.nvidia.com/cuda-toolkit
- Windows, Linux, MacOS
- Последняя версия - 10.1 Update 2

COMPONENTS AND RESOURCES



ВОПРОСЫ?

ПРОГРАММНАЯ МОДЕЛЬ CUDA. ПАРАЛЛЕЛЬНОЕ СЛОЖЕНИЕ ВЕКТОРОВ НА GPU СРЕДСТВАМИ CUDA.

АФАНАСЬЕВ ИЛЬЯ
AFANASIEV_ILYA@ICLOUD.COM

CUDA: ГИБРИДНОЕ ПРОГРАММИРОВАНИЕ

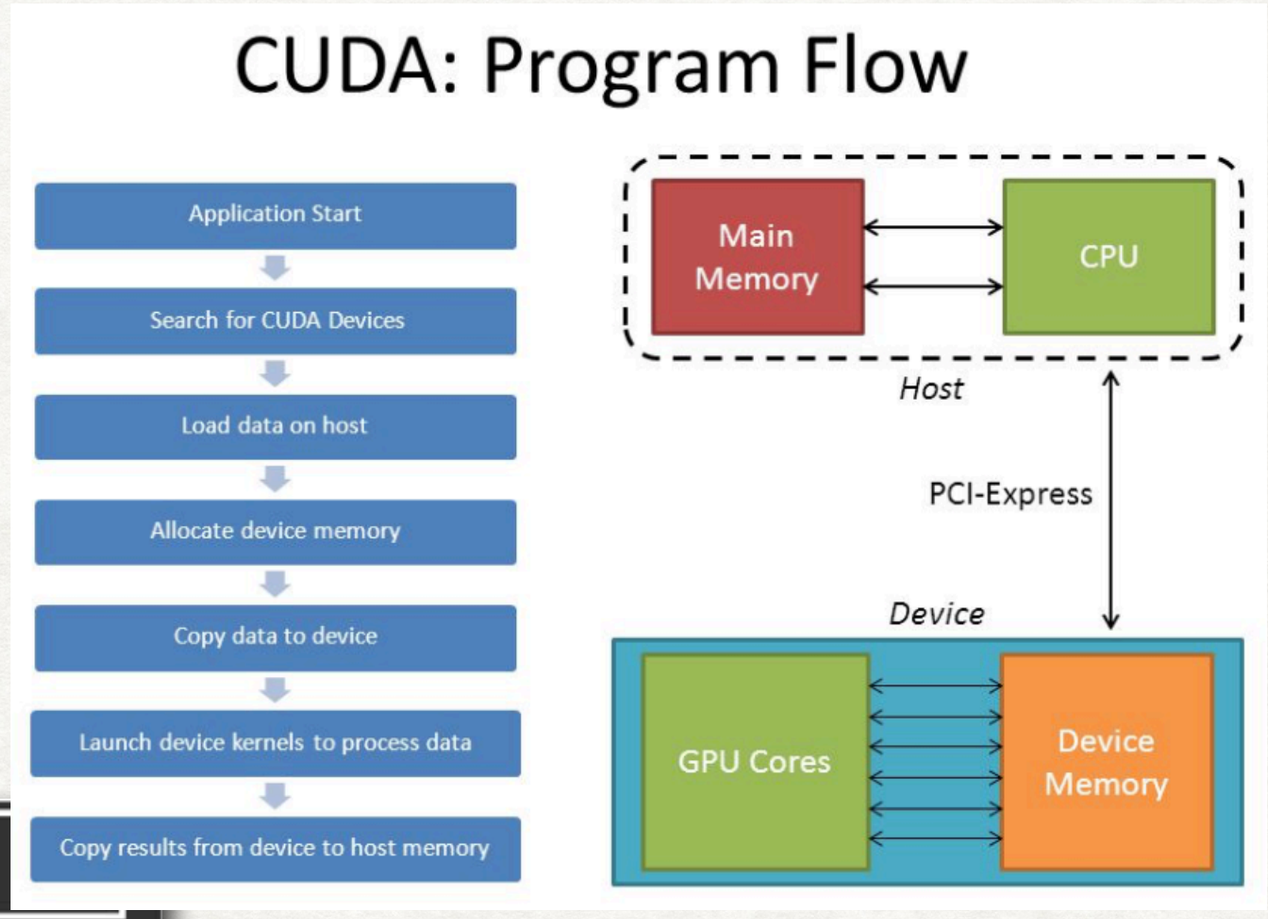
- **CUDA** (изначально аббр. от англ. Compute Unified Device Architecture) – программно-аппаратная архитектура параллельных вычислений, которая позволяет разрабатывать программы для графических процессоров NVIDIA
- Программа, использующая GPU, состоит из:
    - Кода для GPU, описывающего вычисления, выполняемые на графическом ускорителе
    - Кода для CPU, в котором осуществляется:
        - Управление памятью GPU - выделение / освобождение
        - Обмен данными между GPU/CPU
        - Запуск **кода для GPU**
        - Обработка результатов и прочий последовательный код

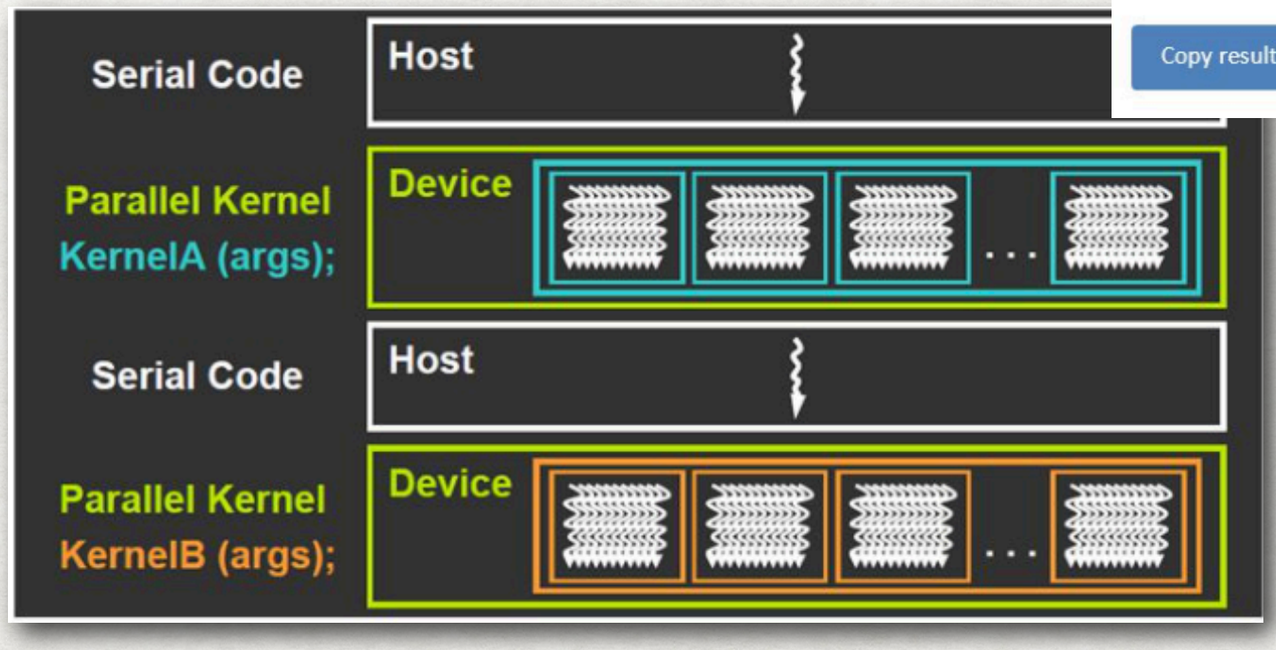
ТЕРМИНОЛОГИЯ (КОД)

- CPU будем далее называть **«хостом»** (от англ. host), код для CPU - код для хоста, **«хост-код»** (host-code)
- GPU будем далее называть **«устройством»** или **«девайсом»** (от англ. device), код для GPU – **«код для устройства», «девайс-код»** (device-code)
- Хост выполняет последовательный код (а при необходимости и параллельный), в **котором содержатся вызовы функций управления устройством**

CUDA-ПРОГРАММА

(HOST CODE + DEVICE CODE)





КАК ХОСТ-КОД ВЗАИМОДЕЙСТВУЕТ С GPU?

- Код для CPU дополняется вызовами **специальных функций для работы с устройством** – выделение памяти, копирования данных, получение свойств GPU
- Код для CPU может компилироваться обычным компилятором GCC/ICC (с подключением специальных библиотек)
    - Кроме конструкций запуска ядра <<<...>>> !
- Возможна компиляция всего кода с использованием компилятора NVCC (доступен в CUDA Toolkit, предустановлен на кластерах)
- Функции для GPU линкуются из динамических библиотек (если используется обычный компилятор)

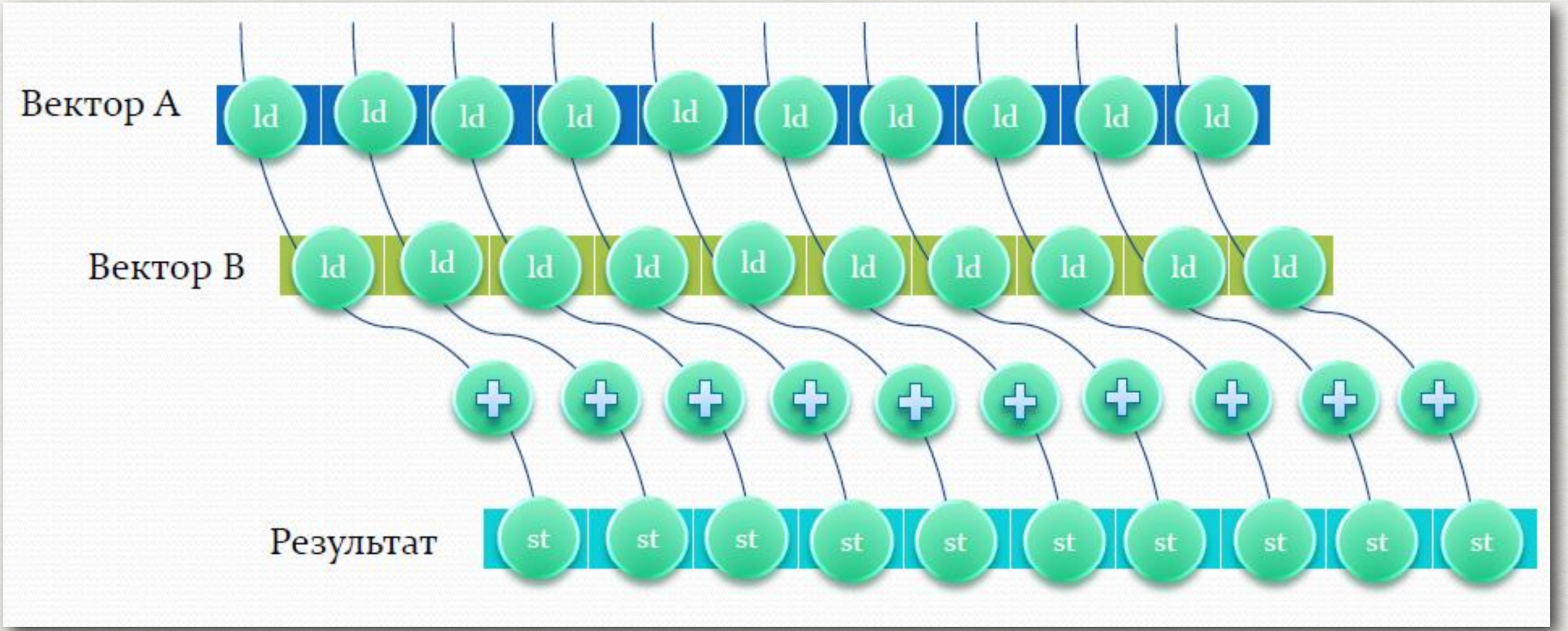
КОД ДЛЯ GPU (DEVICE-CODE)

- Код для GPU пишется на C++ с некоторыми расширениями:
  - Атрибуты функций, переменных и структур
  - Встроенные функции
  - Математика, реализованная на GPU
  - Синхронизации, коллективные операции
  - Векторные типы данных
  - Встроенные переменные:
    threadIdx, blockIdx, gridDim, blockDim
  - Шаблоны для работы с текстурами
- GPU-код компилируется специальным компилятором NVCC (входящим в состав CUDA Toolkit)

ПРИМЕР: СЛОЖЕНИЕ ВЕКТОРОВ

CPU:

```
for(int i=0; i<N; ++i)
      c[i]= a[i] + b[i];
```



ПРИМЕР: СЛОЖЕНИЕ ВЕКТОРОВ

Как будет выглядеть CUDA-программа, складывающая вектора?

Хост-код:

```
{
    // Переслать данные с CPU на GPU
    // Запустить вычисления на N GPU-нитях
    // Скопировать результат с GPU на CPU
}
```

Девайс-код:

```
{
    c[IDX] = a[IDX] + b[IDX]; // на каждой нити
}
```

GPU: НИТИ, БЛОКИ, GRID

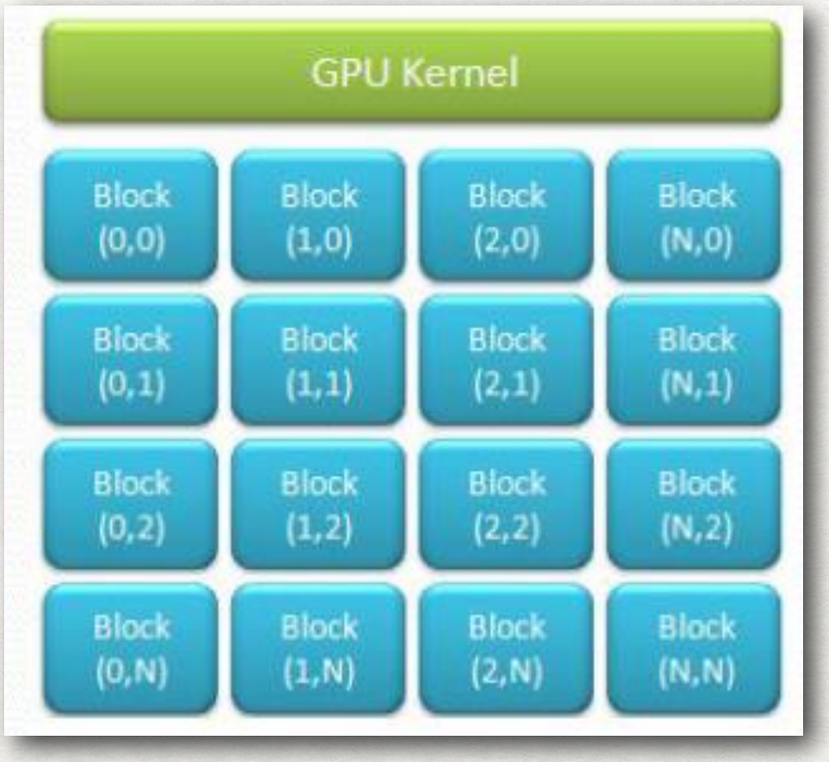
- Хост может запускать на GPU множества виртуальных нитей
    - Каждая из нитей использует для вычислений различные CUDA-ядра (в зависимости от типа выполняемой инструкции)
    - Нити группируются в виртуальные блоки (число блоков и размер блока задаёт пользователь, но с некоторыми аппаратными ограничениями, таким образом регулируя общее число нитей)
- Каждый потоковый мультипроцессор GPU (SM) обрабатывает несколько блоков
- Грид (от англ. Grid-сетка) – множество блоков одинакового размера, на которых запускается GPU-код
- Положение нити в блоке и блока в гриде индексируются по трём измерениям (x, y, z) - как у каждой нити, так и у каждого блока есть трехмерный индекс

РАЗМЕРЫ GRID

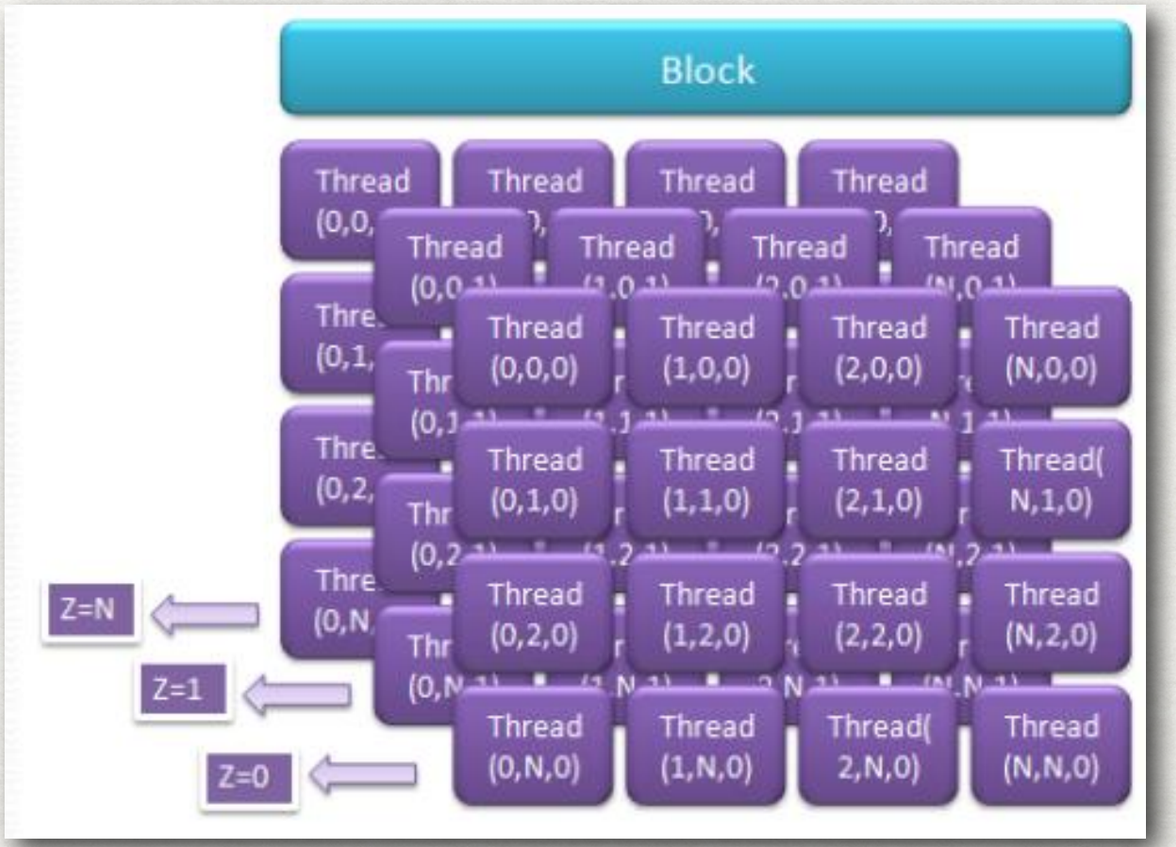
- Грид задаётся количеством блоков по [X, Y, Z] (размер грида в блоках) и размерами каждого блока по [X, Y, Z]

- Каждый из размеров может быть равен 1. Важные частые случаи - одномерный (Z=1) и двумерный грид (Z=1, Y=1), используемые для обработки векторов и матриц соотвественно.

двумерный грид

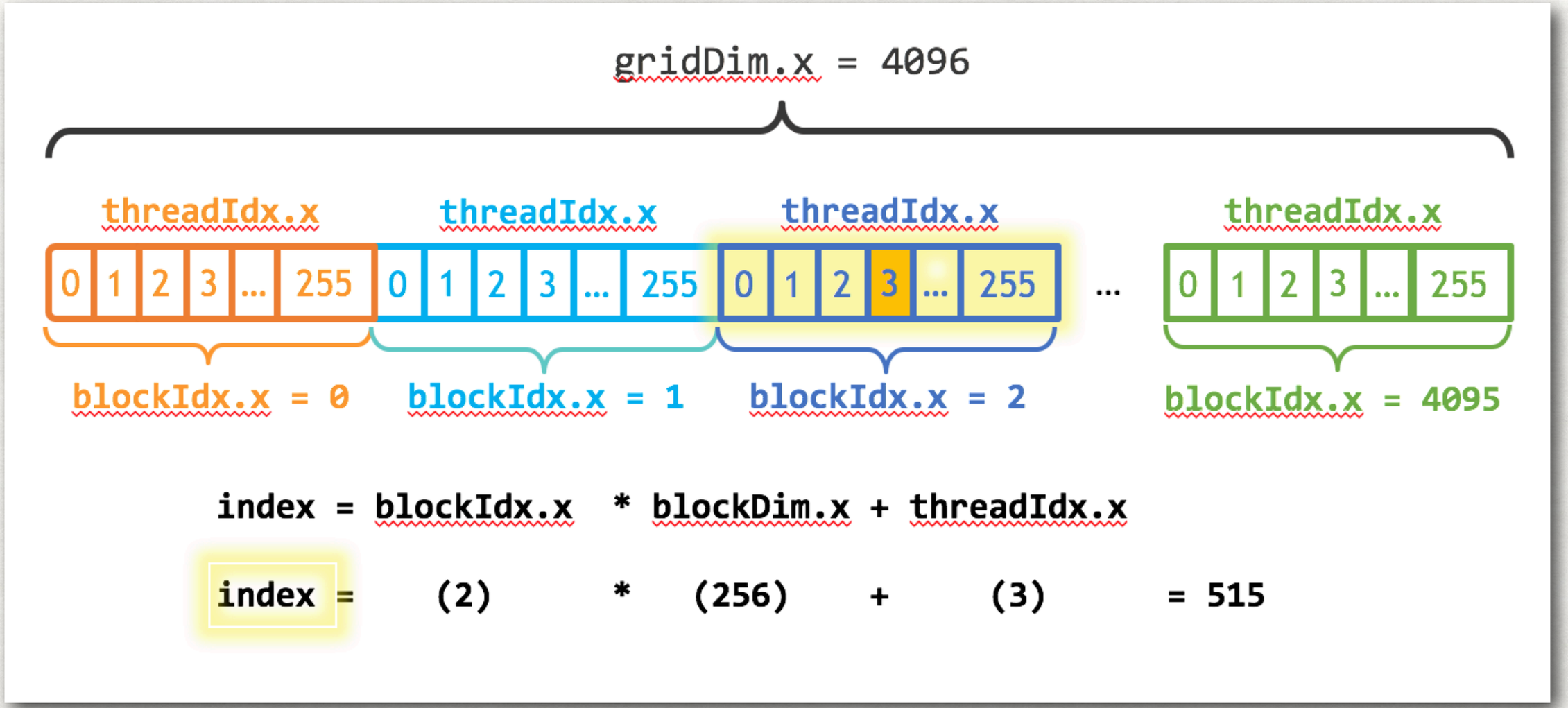


двумерный блок

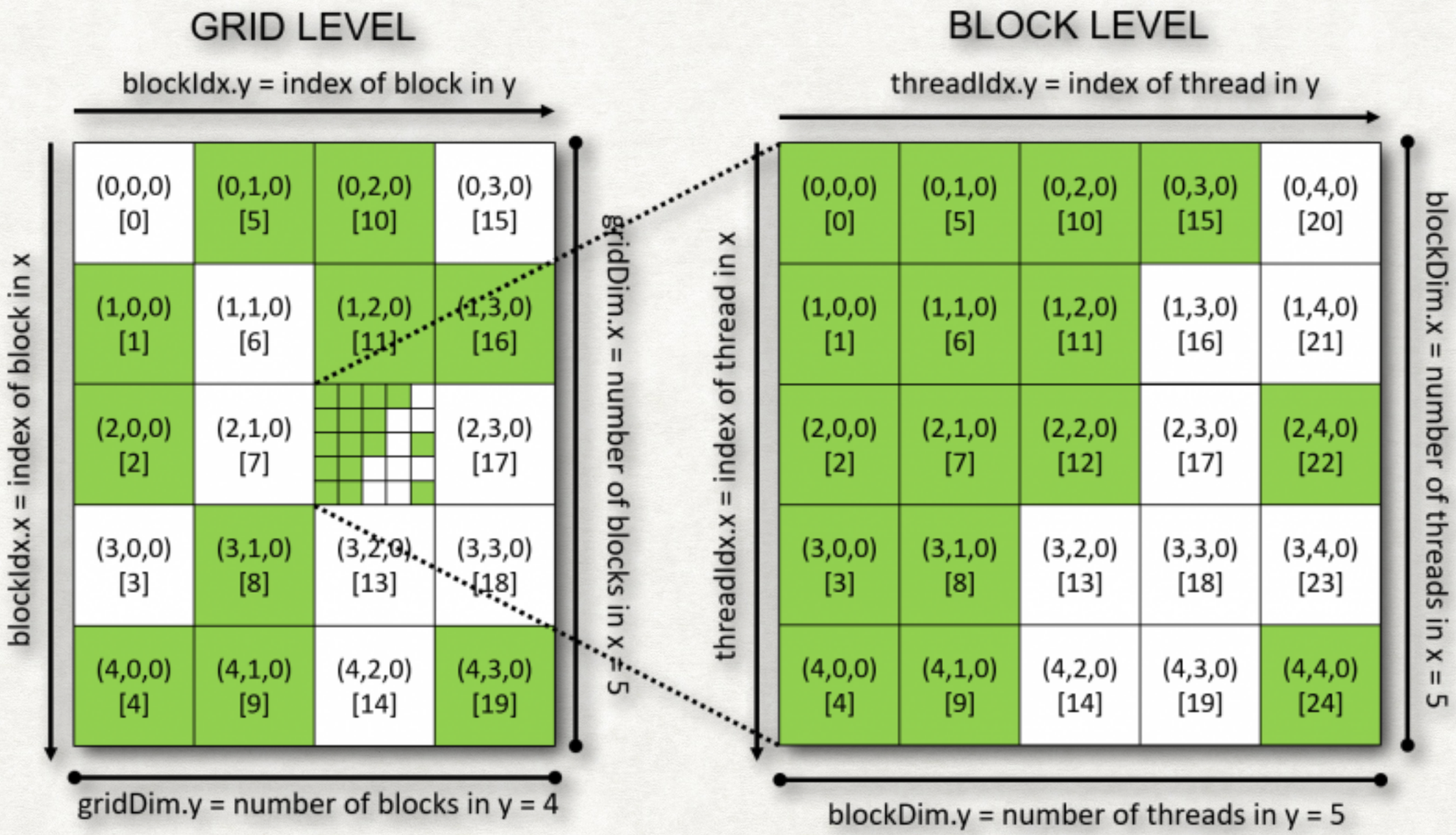


ОДНОМЕРНЫЙ ГРИД

- 4096 блоков
- в каждом блок 256 нитей
- всего 1048576 нитей



ДВУМЕРНЫЙ ГРИД И ДВУМЕРНЫЙ БЛОК



CUDA KERNEL («ЯДРО»)

- Поведение каждой из CUDA-нитей описывается специальными CUDA-ядрами
- Каждая нить выполняет копию специально оформленных функций «ядер», компилируемых для GPU
- В простейшем случае каждая из нитей выполняет одну и ту же версию кода над разными данными (data-driven параллелизм), однако в общем случае поведение нити может зависеть от её индекса
- У ядер нет возвращаемого значения (void)
- Перед описанием ядра должен присутствовать обязательный атрибут __global__
- CUDA-ядра описываются **только** в .cu файлах, компилируемых при помощи NVCC
- Пример:

```
__global__ void kernel (int param1, float *param2)
 {
    // execute device code
 }
```

ТЕРМИНОЛОГИЯ (НИТИ И ЯДРА)

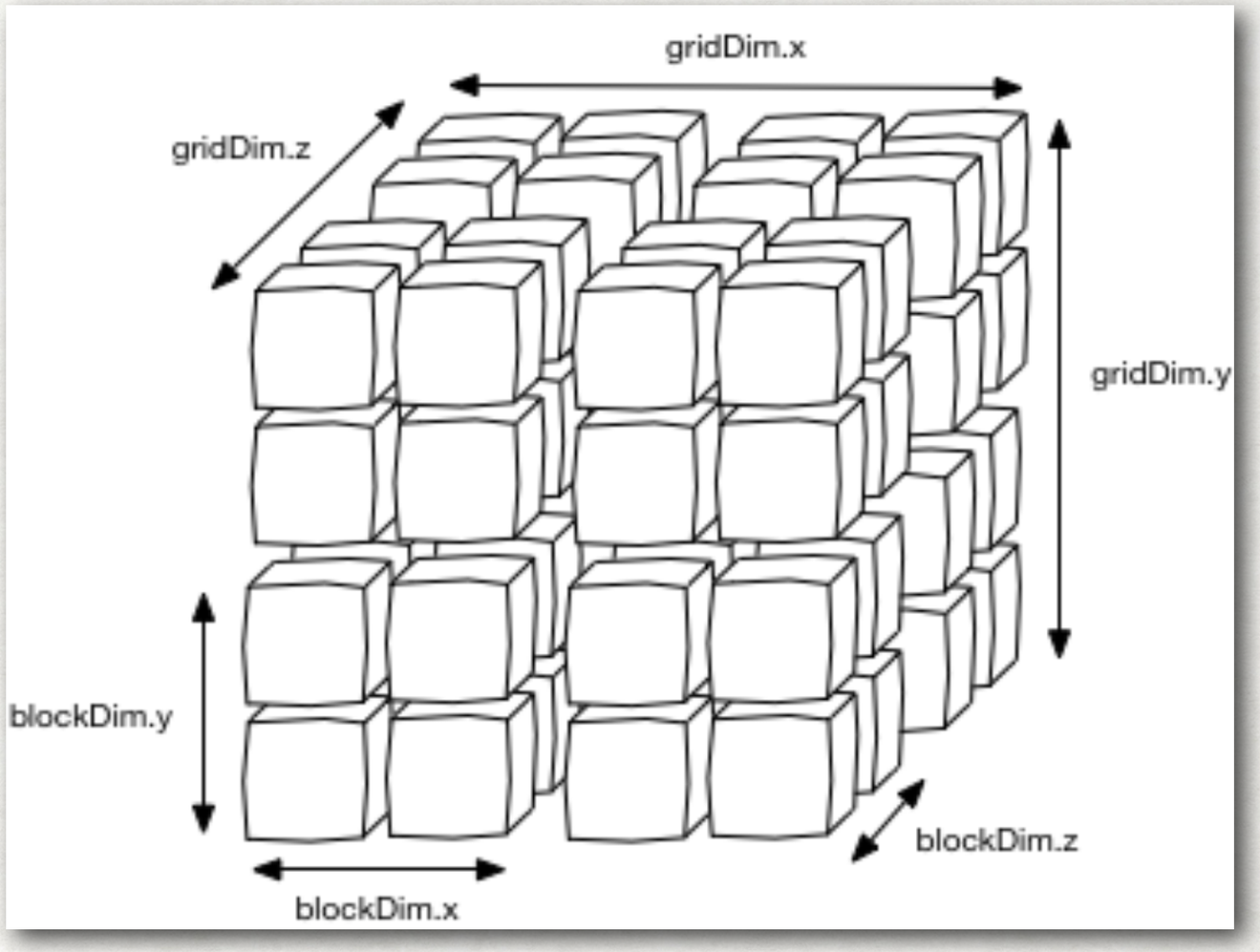
- **Хост** запускает вычисление ядра на **гриде** нитей (либо просто хост запускает ядро на GPU)
- Одно и то же ядро может быть запущено на разных **гридах** (с разной конфигурацией число блоков/размер блока)
- «Ядро» – программа, поток управления, определяет что делать
- «Грид» – определяет сколько делать

СИНТАКСИС ВЫЗОВА ЯДРА

- **kernel <<< execution configuration >>> (params);**
  - “kernel”-имя ядра
  - “params”-параметры ядра,копию которых получит каждая нить
  - execution configuration-Dg,Db,Ns,S
    - dim3 Dg - размеры грида в блоках, Dg.x * Dg.y * Dg.z число блоков
    - dim3 Db - размер каждого блока, Db.x * Db.y * Db.z - число нитей в блоке
    - size_t Ns – размер динамически выделяемой общей памяти (опционально)
    - cudaStream_t S - поток, в котором следует запустить ядро (опционально)
- struct dim3 – стуктура, определённая в CUDAToolkit
  - Состоит из трех полей: unsigned x,y,z
  - Конструктор dim3(unsigned x=1, unsigned y=1, unsigned z=1)

КАК НИТЬ МОЖЕТ ОПРЕДЕЛИТЬ СВОЮ ПОЗИЦИЮ В ГРИДЕ?

- При помощи специализированных встроенных переменных:
  - threadIdx.x / threadIdx.y / threadIdx.z - индексы нити в блоке
  - blockIdx.x / blockIdx.y / blockIdx.z – индексты блока в гриде
  - blockDim.x / blockDim.y / blockDim.z – размеры блоков в нитях
  - gridDim.x / gridDim.y / gridDim.z – размеры грида в блоках



ПОЗИЦИЯ НИТИ В GRID

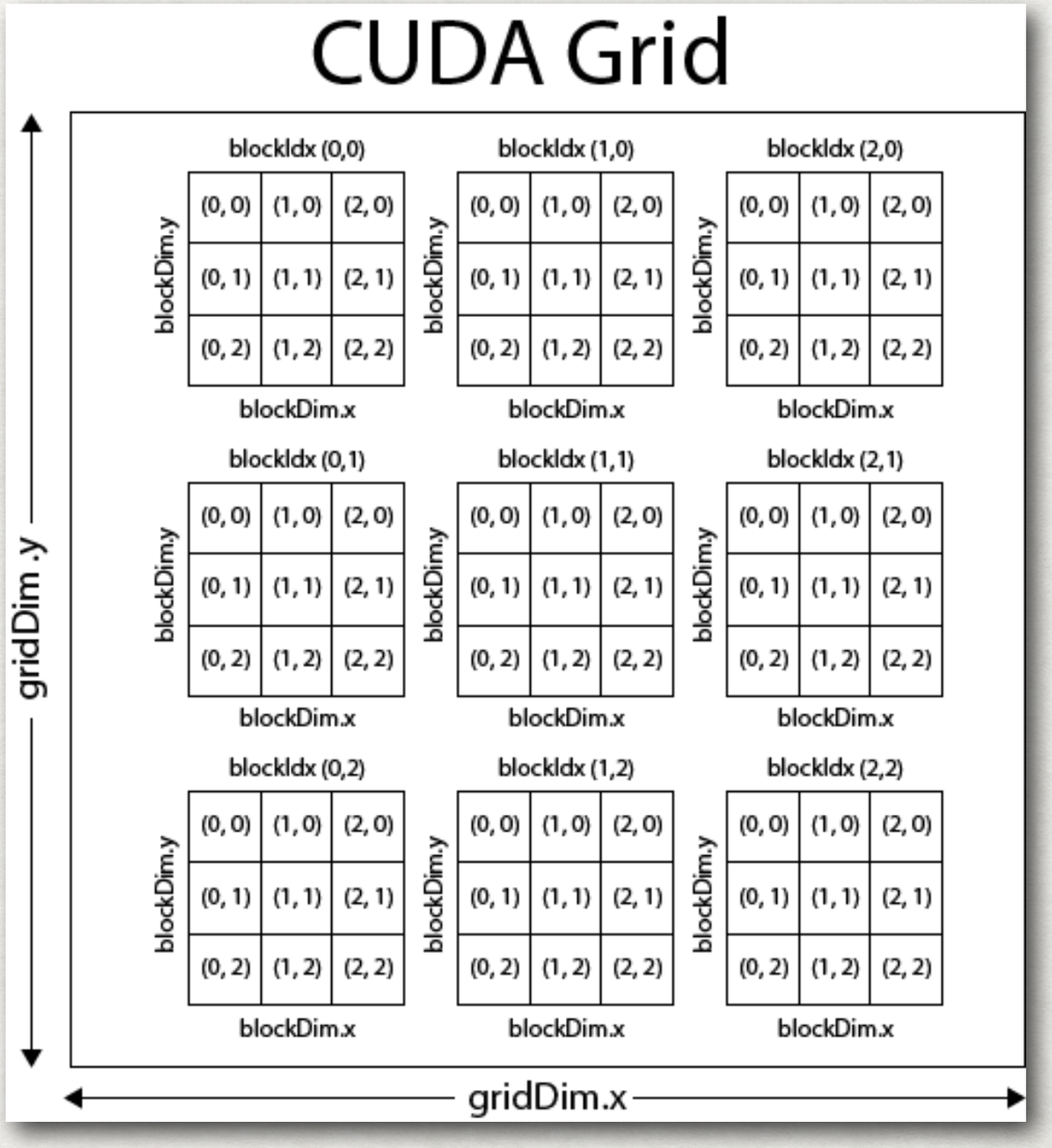
- Индекс нити в одномерном GRID:

```
int global_index = blockIdx.x * blockDim.x + threadIdx.x;
```

- Индекс нити в трехмерном GRID:

```
int gridSizeX = blockDim.x *
gridDim.x; int gridSizeZ = ... ;
gridSizeY = ...;
int gridSizeAll = gridSizeX * gridSizeY
* gridSizeZ;

int threadLinearIdx = (threaIdx.z *
gridSizeY + threaIdx.y) * gridSizeX +
threadIdx.x;
```



ПРИМЕР: СЛОЖЕНИЕ ВЕКТОРОВ, ДЕВАЙС-КОД

- Как реализовать сложение векторов на GPU?
- Пусть данные уже находятся в памяти GPU
- Тогда:

1. создадим грид, с числом нитей равным длине входных векторов
2. каждая нить определит свой линейный индекс i в гриде и выполнит операцию c[i] = a[i]+b[i]

- Как определить рассчитать размер грида?
    - Возьмем максимальный размер блок (1024 для P100 GPU)
    - Число блоков = (N - 1)/1024 + 1
    - При таком подходе будет создано достаточно блоков и нитей, чтобы каждая сложила соответствующие пары элементов векторов

ПРИМЕР: СЛОЖЕНИЕ ВЕКТОРОВ, ДЕВАЙС-КОД

```
__global__ void sum_kernel( int *A, int *B, int *C )
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x; //определить свой индекс
    int a = A[idx]; //считать нужный элемент A
    int b = B[idx]; // считать нужный элемент B
    C[idx] = a + b; //записать результат суммирования
}
```

- Каждая нить:
  - Получает копию параметров (В данном случае, это адреса вектором на GPU)
  - Определяет своё положение в гриде threadLinearIdx
  - Считывает из входных векторов элементы с индексом threadLinearIdx и записывает их сумму в выходной вектор по индексу threadLinearIdx
  - рассчитывает один элемент выходного массива
- Вопрос: какой есть недостаток в данном коде?

ПРИМЕР: СЛОЖЕНИЕ ВЕКТОРОВ, ХОСТ-КОД

- А как же хост-код? Как скопировать данные на GPU и запустить ядро?
- Необходимо:

1. Выделить память на устройстве
2. Переслать на устройство входные данные
3. Рассчитать грид, размер грида зависит от размера задачи
4. Запустить вычисления на гриде, в конфигурации запуска указываем грид
5. Переслать результат вычислений с устройства на хост

ВЫДЕЛЕНИЕ И ОСВОБОЖДЕНИЕ ПАМЯТИ

- cudaError_t cudaMalloc (void** devPtr, size_t size)
  Выделяет size байтов линейной памяти на устройстве и возвращает указатель на выделенную память в *devPtr. Память не обнуляется. Адрес памяти выровнен по 512 байт

- cudaError_t cudaFree (void* devPtr)
  Освобождает память устройства на которую указывает devPtr.

- Аналогичны стандартным вызовам Malloc/free

КОПИРОВАНИЕ ДАННЫХ

- cudaError_t cudaMemcpy (void* dst, const void* src, size_t count, cudaMemcpyKind kind)
- Копирует count байтов из памяти, на которую указывает src в память, на которую указывает dst, kind указывает направление передачи
- cudaMemcpyHostToHost – копирование между двумя областями памяти на хосте
- cudaMemcpyHostToDevice – копирование с хоста на устройство
- cudaMemcpyDeviceToHost – копирование с устройства на хост
- cudaMemcpyDeviceToDevice – между двумя областями памяти на устройстве

СЛОЖЕНИЕ ВЕКТОРОВ, ПОЛНЫЙ ПРИМЕР

```
int main()
{
    // Size of vectors
    int n = 100000;

    // Host vectors
    double *h_a, *h_b, *h_c;

    // Size, in bytes, of each vector
    size_t bytes = n*sizeof(double);

    // Allocate memory for each vector on host
    h_a = (double*)malloc(bytes);
    h_b = (double*)malloc(bytes);
    h_c = (double*)malloc(bytes);

    int i;
    // Initialize vectors on host
    for( i = 0; i < n; i++ )
    {
       h_a[i] = sin(i)*sin(i);
       h_b[i] = cos(i)*cos(i);
    }
```

СЛОЖЕНИЕ ВЕКТОРОВ, ПОЛНЫЙ ПРИМЕР

```
// Device input vectors
double *d_a, *d_b, *d_c;

// Size, in bytes, of each vector
size_t bytes = n*sizeof(double);

// Allocate memory for each vector on GPU
cudaMalloc(&d_a, bytes);
cudaMalloc(&d_b, bytes);
cudaMalloc(&d_c, bytes);

// Copy host vectors to device
cudaMemcpy( d_a, h_a, bytes, cudaMemcpyHostToDevice);
cudaMemcpy( d_b, h_b, bytes, cudaMemcpyHostToDevice);

int blockSize, gridSize;
// Number of threads in each thread block
blockSize = 1024;
// Number of thread blocks in grid
gridSize = (n-1)/blockSize + 1;
// Execute the kernel
vecAdd<<<gridSize, blockSize>>>(d_a, d_b, d_c, n);


sum = x + y;
// Copy array back to host
cudaMemcpy( h_c, d_c, bytes, cudaMemcpyDeviceToHost );

// Release device memory
cudaFree(d_a);
cudaFree(d_b);
cudaFree(d_c);
```

КОМПИЛЯЦИЯ

(ОСОБОЕ ОТНОШЕНИЕ К .CU ФАЙЛАМ)

- При работе с CUDA используются расширения C/Си++:
  - Конструкция запуска ядра<<<....>>>
  - Встроенные переменные threadIdx, blockIdx
  - Квалификаторы __global__ __device__ и т.д.
  - ....
- Эти расширения могут быть обработаны только в *.cu файлах!
- В этих файлах можно не делать #include <cuda_runtime.h>
- Вызовы библиотечных функций вида cuda* можно располагать в *.cpp файлах
- Они будут слинкованы обычным линковщиком из библиотеки libcudart.so

НЕСКОЛЬКО ВОПРОСОВ...

- Пусть у нас есть массивно-параллельная задача (большой объем параллелизма над данными)
- Достаточно ли просто породить большое число нитей и запустить их на GPU для написания эффективной CUDA-программы?
- Нет. Крайне важно понимать, как нити обрабатываются аппаратурой GPU, о чём и пойдет речь в дальнейших лекциях.

ИЗМЕРЕНИЕ ВРЕМЕНИ РАБОТЫ ЯДРА

(БОЛЕЕ ПОДРОБНО О CUDA-СОБЫТИЯХ НА СЛЕДУЮЩИХ ЛЕКЦИЯХ)

```
cudaEvent_t start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);

cudaMemcpy(d_x, x, N*sizeof(float), cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, N*sizeof(float), cudaMemcpyHostToDevice);

cudaEventRecord(start);
saxpy<<<(N+255)/256, 256>>>(N, 2.0f, d_x, d_y);
cudaDeviceSynchronize();
cudaEventRecord(stop);

cudaMemcpy(y, d_y, N*sizeof(float), cudaMemcpyDeviceToHost);

cudaEventSynchronize(stop);
float milliseconds = 0;
cudaEventElapsedTime(&milliseconds, start, stop);
```

ЗАДАНИЯ

ЗАДАНИЕ 1

(СЛОЖЕНИЕ ВЕКТОРОВ)

- Компиляция и запуск алгоритма сложения векторов на системе Polus
- nvcc -O3 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_60,code=sm_60 main.cu
- Самостоятельно реализовать проверку корректности результата (host и device)
- Доработать программу, чтобы она могла производить сложение векторов произвольного размера

ЗАДАНИЕ 2

(РАБОТА С ДВУМЕРНЫМ ГРИДОМ)

- Реализуем ядро транспонирования матрицы a[i][j] = a[j][i] (можно без хост кода, итоговую версию запустим в конце дня)
- Как сконфигурировать грид для матрицы размера N?
- Какую операцию будет производить каждая из нитей?
- Как необходимо хранить матрицу?
- Как скопировать матрицу в память GPU и обратно?

ВОПРОСЫ?

ОСОБЕННОСТИ ВЫПОЛНЕНИЯ GPU-ПРОГРАММ. АСИНХРОННОСТЬ В CUDA, ОБРАБОТКА ОШИБОК, ИЗМЕРЕНИЕ ВРЕМЕНИ ВЫПОЛНЕНИЯ.

АФАНАСЬЕВ ИЛЬЯ
AFANASIEV_ILYA@ICLOUD.COM

ПОВТОРЕНИЕ

- GPU позволяет крайне эффективно ускорять массивно-параллельные задачи, как memory-bound так и compute-bound
- Pascal, Volta, Ampere – современные GPU семейства Tesla (для HPC)
- Мы работаем на системе Polus, 2x P100 GPU

АППАРАТНЫЕ ХАРАКТЕРИСТИКИ СОВРЕМЕННОГО GPU (АРХИТЕКТУРЫ PASCAL)

Особенности Pascal (на примере P100):

- 56 SMs, 3584 FP32 CUDA-cores
- HBM2 память (до 16 GB, до 720 GB/s)
- L2 кэш размера 4 MB
- Интерконнект – NVLink (80 GB/s)



- 5,3 TFLOPS of double precision
- 10,6 TFLOPS of single precision
- 21,2 TFLOPS of half-precision – вычисления с половинной точностью для DL

АРХИТЕКТУРА ПОТОКОВОГО МУЛЬТИПРОЦЕССОРА (PASCAL)

- Потоковый мультипроцессор - «Единица» построения устройства (как ядро в CPU)
- 64 скалярных ядра CUDA Core, ~1.5ГГц
- CUDA ядра имеют разный тип для различных типов операций (FLT, DBL, LT/ST, и др.)
- 2 Warp Scheduler (планировщик варпов) – SIMD работа CUDA ядер (коллективная)



ОСНОВНЫЕ ОТЛИЧИЯ GPU ОТ CPU

- У GPU – сотни упрощённых вычислительных ядер, работающих на небольшой тактовой частоте ~1.8 ГГц. (значит в программах важен массивный параллелизм и load-balancing), у CPU – ядер мало, высокая таковая частота
- У GPU – небольшие кэши (значит в программах важна локальность), у CPU – большие
  - 64 CUDA-ядра разделяют 64 КБ L1
  - L2 общий для всех потоковых мультипроцессоров ~4 MB, L3 отсутствует
- У GPU оперативная память с высокой пропускной способностью и высокой латентностью, оптимизированная для коллективного доступа (а значит GPU позволяет ускорять memory-bound задачи), у CPU – нет
- У GPU – высокая производительность (в FLOPS), тензорные ядра, половинная арифметика (для DL)
- У GPU поддержка миллионов виртуальных нитей, быстрое переключение контекста для групп нитей (для эффективного задействования ППС памяти), у CPU – сложная иерархия кэшей

CUDA-КОД СОСТОИТ ИЗ ХОСТ- И ДЕВАЙС- ЧАСТЕЙ

- Программа, использующая GPU, состоит из:
    - Кода для GPU, описывающего необходимые вычисления и работу с памятью устройства – CUDA-kernels, __global__, __device__
- Кода для CPU, в котором осуществляется:
    - Управление памятью GPU – выделение / освобождение – cudaMalloc, …
    - Обмен данными между GPU/CPU – cudaMemcpy…
    - Запуск кода для GPU – kernel <<<grid, block>>>(…)
    - Обработка результатов и прочий последовательный код

CUDA-КОД СОСТОИТ ИЗ ХОСТ- И ДЕВАЙС- ЧАСТЕЙ

- Программа, использующая GPU, состоит из:
    - Кода для GPU, описывающего необходимые вычисления и работу с памятью устройства – CUDA-kernels, __global__, __device__
- Кода для CPU, в котором осуществляется:
    - Управление памятью GPU - выделение / освобождение – cudaMalloc, ...
    - Обмен данными между GPU/CPU – cudaMemcpy...
    - Запуск кода для GPU – kernel <<<grid, block>>>(...)
    - Обработка результатов и прочий последовательный код
- Хост-код выполняется на ядре центрального процессора и использует специализированное CUDA Runtime API для выделений памяти на GPU, копирований данных, получения свойств GPU и др.

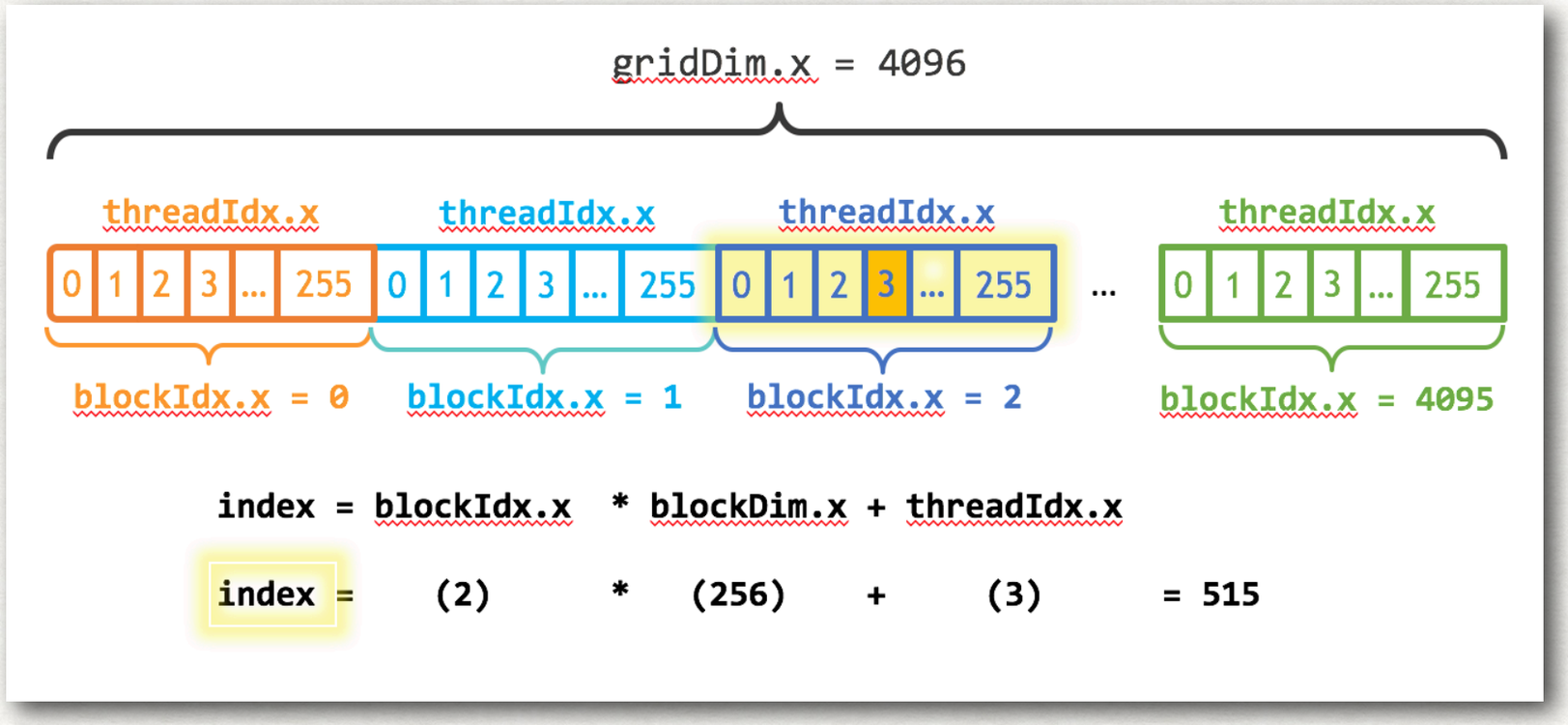
CUDA-ЯДРА — ОСНОВА GPU КОДА

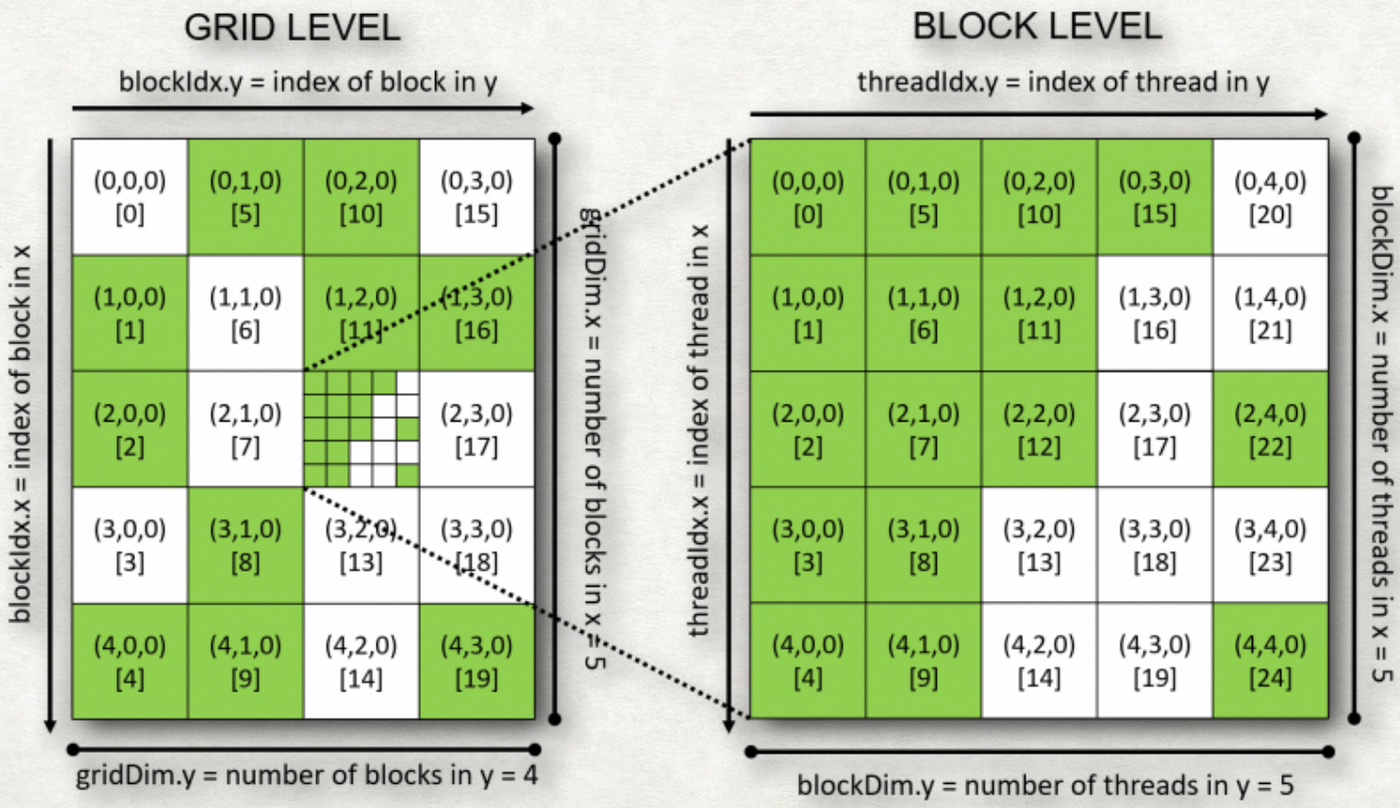
- Ядро описывает - что нужно делать каждой нити (поток управления, алгоритм)

```
__global__ void sum_kernel( int *A, int *B, int *C )
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x; //определить свой индекс
    int a = A[idx]; //считать нужный элемент A
    int b = B[idx]; // считать нужный элемент B
    C[idx] = a + b; //записать результат суммирования
}
```

- Например, для сложения векторов каждая нить:
  - Получает копию параметров (В данном случае, это адреса вектором на GPU)
  - Определяет своё положение в гриде threadLinearIdx
  - Считывает из входных векторов элементы с индексом threadLinearIdx и записывает их сумму в выходной вектор по индексу threadLinearIdx
  - рассчитывает один элемент выходного массива

CUDA ЯДРА ЗАПУСКАЮТСЯ НА ГРИДЕ, ОН БЫВАЕТ ОДНОМЕРНЫЙ, ДВУМЕРНЫЙ И ТРЕХМЕРНЫЙ





- Грид определяет ОБЪЕМ работы (сколько нужно делать), его размер зависит от размера задачи
- Например – длина векторов, размер матриц

ГРИД РАЗБИВАЕТСЯ НА БЛОКИ, КАЖДЫЙ БЛОК — ОБРАБАТЫВАЕТСЯ SM GPU

- Нити блока обрабатываются CUDA-cores (с рисунка), они бывают разного типа для разных типов инструкций (FLT, DBL, LD/ST, …)

- Индекс нити в одномерном GRID:

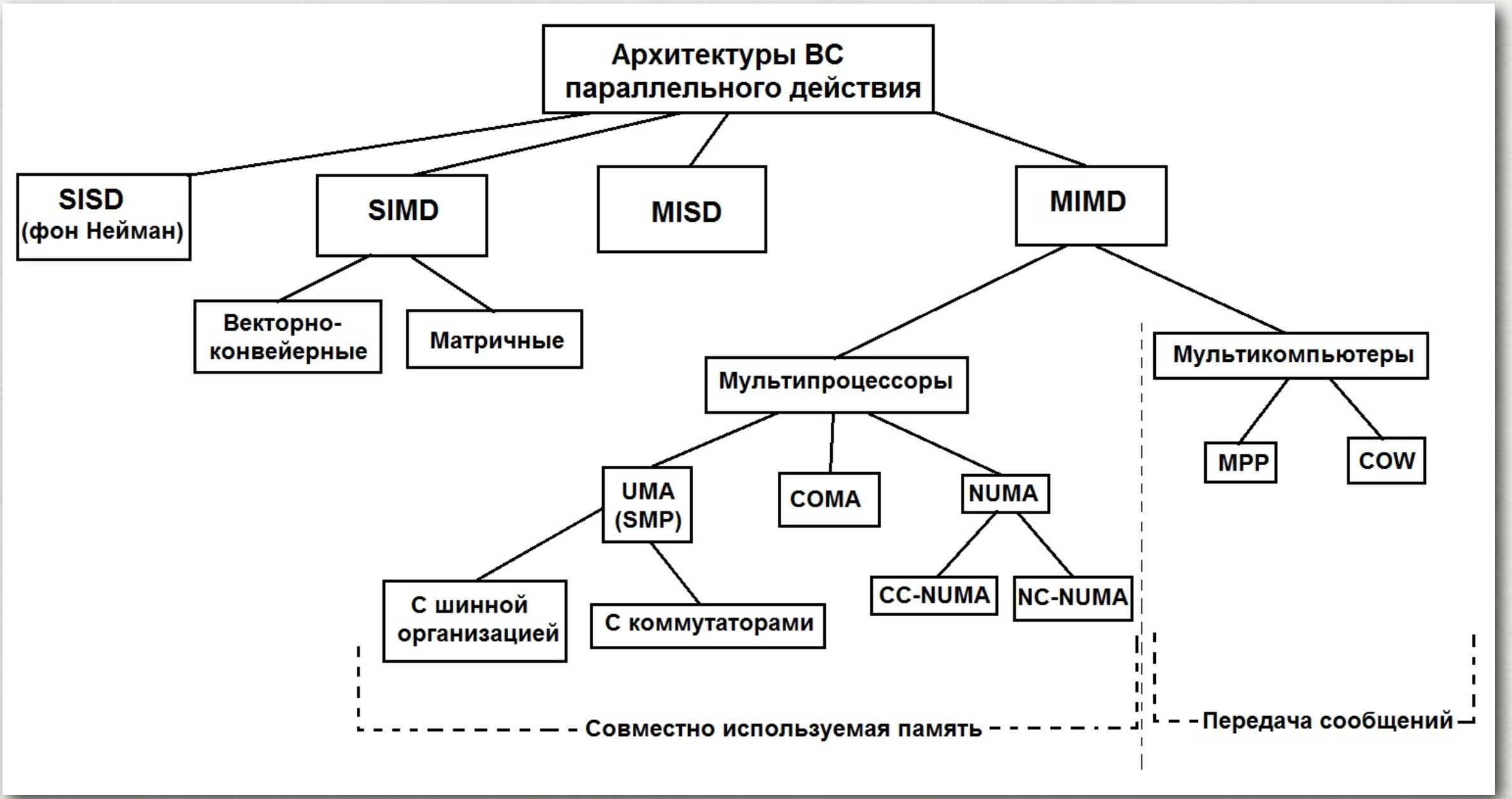
```
int global_index = blockIdx.x * blockDim.x + threadIdx.x;
```

ЕСТЬ ЛИ ВОПРОСЫ ПО ПОВТОРЕННОМУ МАТЕРИАЛУ?

МОДЕЛЬ ВЫПОЛНЕНИЯ CUDA-ПРОГРАММ

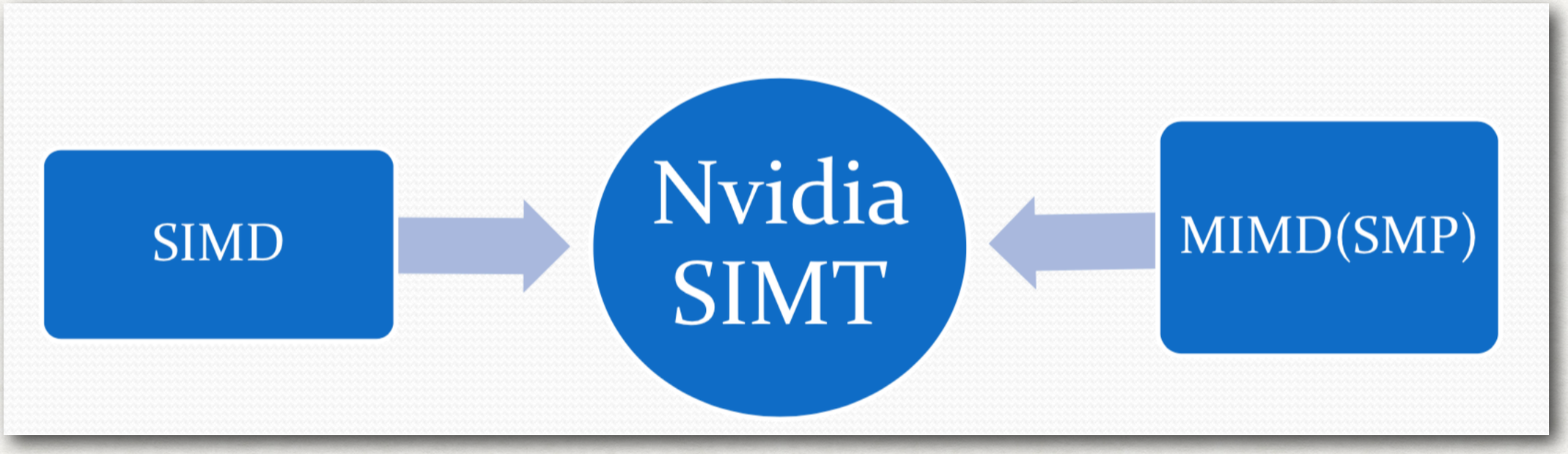
- На предыдущей лекции мы выяснили, что CUDA-ядра запускаются на множестве нитей
- Однако, необходимо понять, как CUDA-нити отображаются на архитектуру GPU (ядра, мультипроцессоры, кэши, память), а так же каким отличительными особенностями обладают
- Отличительные особенности:
  - SIMT модель вычислений (рассмотрена далее)
  - модель доступа к памяти нитей (рассмотрена на следующей лекции)
  - доступ нитей к иерархии памяти (более подробно во второй день)

КЛАССИФИКАЦИЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ ПО ФЛИННУ



А ГДЕ ЖЕ В ДАННОЙ КЛАССИФИКАЦИИ GPU?

- У NVIDIA собственная модель исполнения, имеющая черты как SIMD, так и MIMD
- Nvidia SIMT: Single Instruction – Multiple Thread



SIMT: ВИРТУАЛЬНЫЕ НИТИ, БЛОКИ

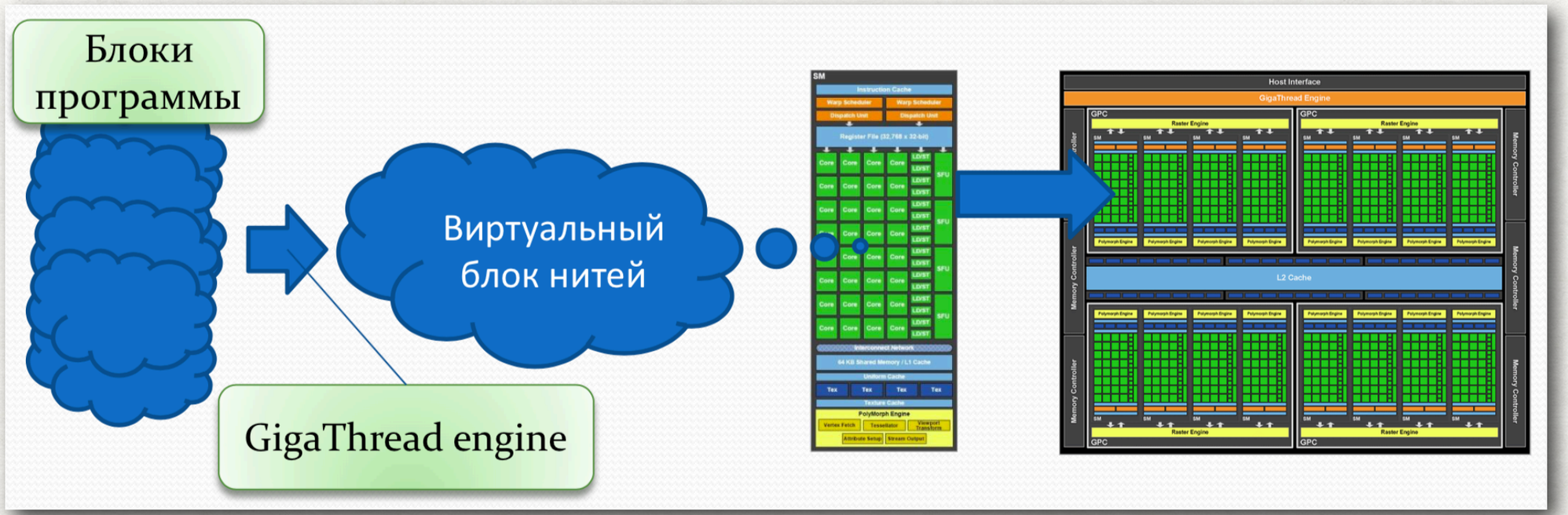
(ВЫЧИСЛЕНИЯ С ТОЧКИ ЗРЕНИЯ ПРОГРАММИСТА)

- **Виртуально**, все нити
  - выполняются параллельно (MIMD)
  - Имеют одинаковые права на доступ к памяти (MIMD :SMP)
- Нити разделены на группы одинакового размера (блоки):
- В общем случае, **глобальная синхронизация всех нитей невозможна**, нити из разных блоков выполняются полностью независимо
- Есть локальная синхронизация внутри блока, нити из одного блока могут взаимодействовать через специальную память
- Нити не мигрируют между блоками. Каждая нить находится в своём блоке с начала выполнения и до конца

SIMT: АППАРАТНОЕ ВЫПОЛНЕНИЕ

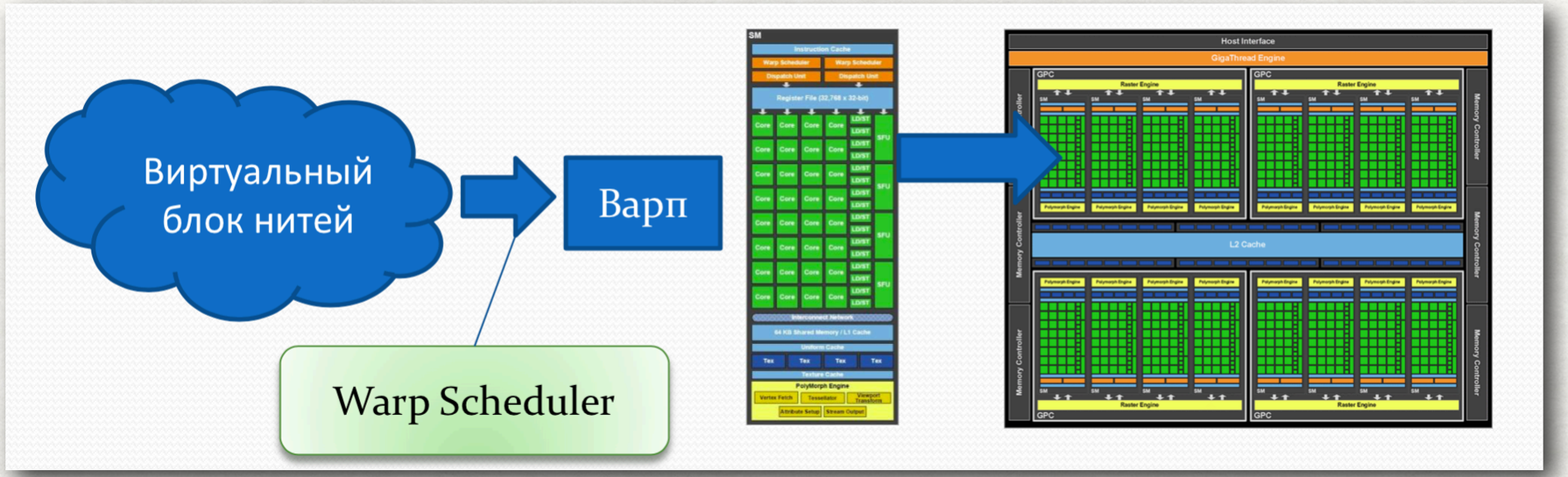
(ВЫЧИСЛЕНИЯ С ТОЧКИ ЗРЕНИЯ АППАРАТУРЫ)

- Все нити из одного блока выполняются на одном мультипроцессоре (SM)
- Максимальное число нитей в блоке – 1024 (зависит от архитектуры)
- Блоки не мигрируют между SM
- Распределение блоков по мультироцесссорам непредсказуемо
- Каждый SM работает независимо от других
- Блоков может быть значительно больше, чем мультипроцессоров GPU



ВАРПЫ (WARPS)

- Блоки нитей по фиксированному правилу (циклично в порядке увеличения индекса нити) разделяются на группы по 32 нити, называемые варпами (warp)
- Все нити варпа одновременно выполняют одну общую инструкцию (SIMD-выполнение)
- Warp Scheduler (планировщик варпов) на каждом цикле работы выбирает варп, все нити которого готовы к выполнению следующей инструкции и запускает весь варп



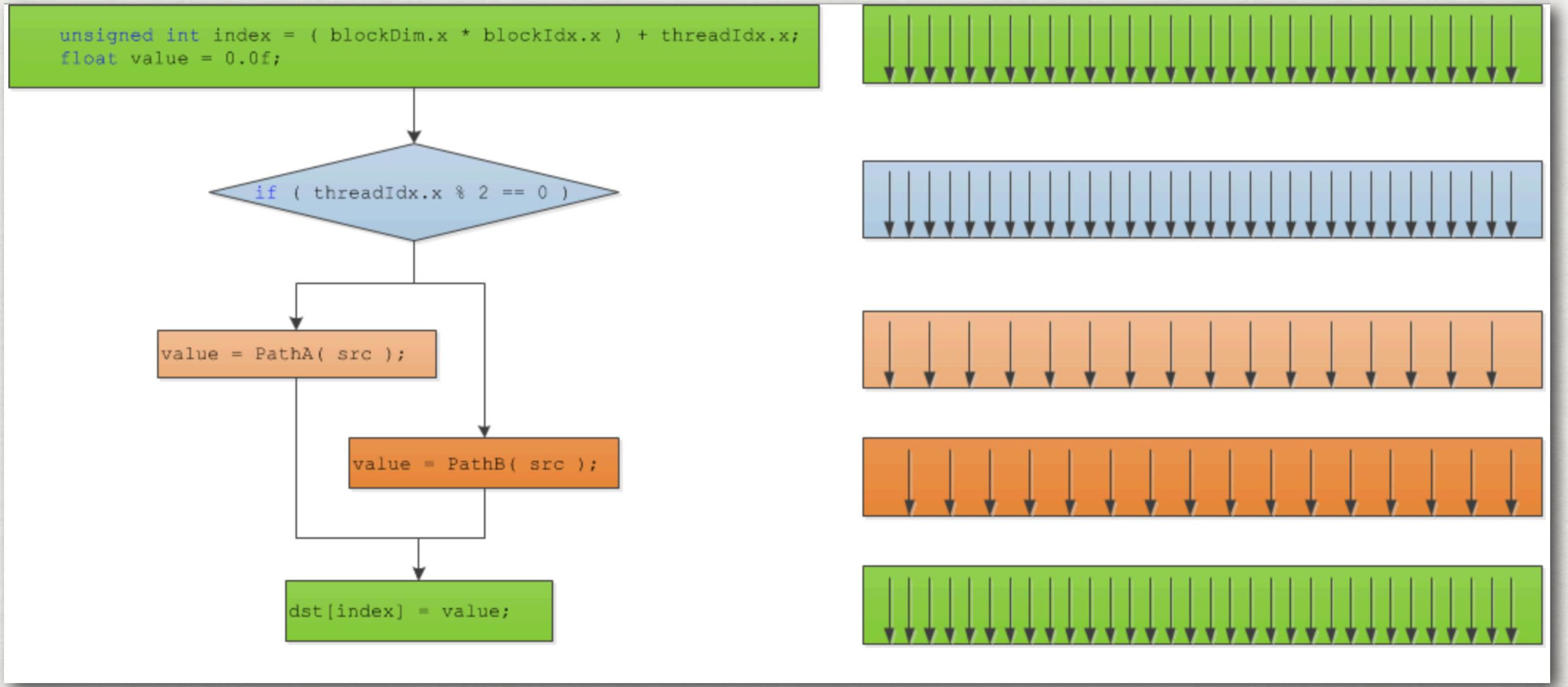
ВЕТВЛЕНИЕ (BRANCHING)

- Если все нити варпа одновременно выполняют одну и ту же инструкцию, то как быть, если часть нитей эту инструкцию выполнять не должна?
- Пример:

if(<условие, зависшее от индекса нити>), где значение условия различается для нитей одного варпа

- Эти нити «замаскируются» нулями в специальном наборе регистров и не будут её выполнять, т.е. будут простаивать
- Другие примеры - switch+case, while, for ...

ВЕТВЛЕНИЕ (BRANCHING)



SIMD И ЗАГРУЗКА ДАННЫХ ИЗ ПАМЯТИ?

- Пусть warp выполняет инструкцию загрузки данных из памяти
- Аналогично с CPU, GPU использует кэши (рассмотрены на следующей лекции)
- Если хотя бы одна из нитей промахивается при попытке загрузить данные из кэша, все нити варпа будут ожидать загрузки данных
- Память GPU оптимизирована для коллективной обработки обращений от нитей – механизм транзакций (будет подробно рассмотрен на следующей лекции)

АСИНХРОННОСТЬ В CUDA

- Чтобы GPU больше времени работало в фоновом режиме, параллельно с CPU, некоторые вызовы являются асинхронными (отправляют команду на устройство и сразу возвращают управление хосту)

- К таким вызовам относятся:

  - Запуски ядер

  - Копирование между двумя областями памяти **на устройстве** (deviceToDevice)

  - Копирования, выполняемые функциями с окончанием *Async cudaMemSet – присваивает всем байтам области памяти на устройстве одинаковое значение (чаще всего используется для обнуления)

АСИНХРОННОСТЬ В CUDA

- Почему тогда верно работает код?

//запуск ядра (асинхронно)

sum_kernel<<< blocks, threads >>>(aDev, bDev, cDev);

//переслать результаты обратно на хост

cudaMemcpy(cHost, cDev, nb, cudaMemcpyDeviceToHost);

- Ведь хост вызывает cudaMemcpy до завершения выполнения ядра!
- Ответ: Вызов ядра и cudaMemcpy попадают в один поток (поток по умолчанию), гарантируется их последовательное выполнение
- Поэтому, для асинхронности во многих случаях нужно использовать **механизм CUDA-потоков** (про них будет рассказано в будущих лекциях)

ОБРАБОТКА ОШИБОК

- Коды всех возникающих ошибок автоматически записываются в единственную специальную хостовую переменную типа enum cudaError_t
- Эта переменная в каждый момент времени равна коду последней ошибки, произошедшей в системе
  - cudaError_t cudaPeekAtLastError() – возвращает текущее значение этой переменной
  - cudaError_t cudaGetLastError() - возвращает текущее значение этой переменной и присваивает ей cudaSuccess
  - const char* cudaGetErrorString (cudaError_t error) –по коду ошибки возвращает её текстовое описание

ПРИМЕР ОБРАБОТКИ ОШИБОК

```
#define SAFE_CALL( CallInstruction ) { \
    cudaError_t cuerr = CallInstruction; \
    if(cuerr != cudaSuccess) { \
        printf("CUDA error: %s at call \"" #CallInstruction "\"\n", cudaGetErrorString(cuerr)); \
            throw "error in CUDA API function, aborting..."; \
    } \
}

#define SAFE_KERNEL_CALL( KernelCallInstruction ){ \
    KernelCallInstruction; \
    cudaError_t cuerr = cudaGetLastError(); \
    if(cuerr != cudaSuccess) { \
        printf("CUDA error in kernel launch: %s at kernel \"" #KernelCallInstruction "\"\n", cudaGetErrorString(cuerr)); \
            throw "error in CUDA kernel launch, aborting..."; \
    } \
    cuerr = cudaDeviceSynchronize(); \
    if(cuerr != cudaSuccess) { \
        printf("CUDA error in kernel execution: %s at kernel \"" #KernelCallInstruction "\"\n", cudaGetErrorString(cuerr));
\
            throw "error in CUDA kernel execution, aborting..."; \
    } \
}
```

ЗАДАНИЯ

ЗАДАНИЕ 1

(SIMD МОДЕЛЬ ВЫЧИСЛЕНИЙ)

- добавить в сложение векторов ситуацию дивергенции нитей внутри варпа

  (1) if(threadIdx % 2 == 0) // почему дивергенции здесь не будет?

  c[i] = a[i] + b[i];

  else

  c[i] = a[i] * b[i];

  (2) while(i < threadIdx)

  {

  //do somethin

  res_data[idx] += a[idx] * i;

  }

- Сравнить время выполнения с дивергенцией и без
- nvprof --metrics warp_execution_efficiency ./a.out

ВОПРОСЫ?

CUDA-ПОТОКИ (STREAMS). ПРОГРАММИРОВАНИЕ СИСТЕМ С НЕСКОЛЬКИМИ ГРАФИЧЕСКИМИ ПРОЦЕССОРАМИ (MULTI-GPU)

АФАНАСЬЕВ ИЛЬЯ
AFANASIEV_ILYA@ICLOUD.COM

CUDA STREAMS

ПРЕДПОСЫЛКА

- Время работы GPU приложения = Время выполнения ядер + **время копирования данных**
- 2 подхода:

1. ускорение непосредственно копирований (Pinned memory)
2. **выполнение копирований параллельно с вычислениями (и не только, CUDA streams)**

CUDA STREAMS, ОПРЕДЕЛЕНИЕ

- **CUDA Stream** - последовательность команд для GPU (запуски ядер, копирования памяти и т.д.), исполняемая строго последовательно
- По-умолчанию, все команды помещаются в «Default Stream», равный нулю (именно в нём мы до этого и работали)
- Однако, пользователь может сам создавать новые потоки и распределять команды между ними
- Только команды из разных потоков, отличных от потока по-умолчанию, могут выполняться параллельно
- В общем случае порядок выполнения команд из различных потоков не определен (без явных синхронизаций, выполняемых пользователем)

СОЗДАНИЕ CUDA ПОТОКОВ

- Поток привязывается к текущему активному GPU (подробнее во второй части)
- Перед отправлением команды нужно переключаться на устройство, к которому привязан поток
- Если попробовать отправить в него команду при другом активном устройстве, будет ошибка
- Пример создания потоков:

```
cudaStream_t stream;

cudaStreamCreate(&stream);

// so some job

cudaStreamDestroy(stream);
```

- cudaStreamDestroy не выполняет синхронизацию
- Управление возвращается хостовому процессу сразу, реальное освобождение ресурсов произойдет после завершения всех команд потока

ПАРАЛЛЕЛЬНАЯ РАБОТА НА СИСТЕМАХ С GPU

- Параллельная работа хоста и устройства
- Параллельная работа нескольких GPU
- **Параллельное выполнение различных ядер и копирований данных на одном устройстве**

ПАРАЛЛЕЛЬНОЕ ВЫПОЛНЕНИЕ КОМАНД НА GPU

- Команды из разных потоков, отличных от потока по-умолчанию, могут исполняться параллельно
- Возможные случаи:
  1. Параллельные копирование и выполнение ядра
  2. Параллельные выполнение ядер
  3. Параллельные копирования с хоста на устройство и с устройства на хост
- Рассмотрим далее примеры для каждого из случаев

ПРИМЕР ПАРАЛЛЕЛЬНОГО КОПИРОВАНИЯ И ЗАПУСКА ЯДРА

- Пример:

```
cudaStream_t stream1 , stream2 ; // создаем переменные, соотвествующие каждому потоку

cudaStreamCreate(&stream1); // инициализируем потоки

cudaStreamCreate(&stream2); // инициализируем потоки

// начинаем копирование в потоке №1
cudaMemcpyAsync(data_d,data_h,size ,cudaMemcpyHostToDevice,stream1);

// начинаем вычсиления в потоке №2

kernel<<<grid , block , 0, stream2 >>>(...);

cudaStreamDestroy(stream1 );

cudaStreamDestroy(stream2 );
```

- Если cudaDeviceProp::asyncEngineCount > 0 устройство может выполнять параллельно копирование и счет ядра
- Хостовая память долна быть page-locked

ПРИМЕР ПАРАЛЛЕЛЬНОГО ВЫПОЛНЕНИЯ ЯДЕР

- Пример:

```
cudaStreamCreate(&stream1);

cudaStreamCreate(&stream2);

kernel1<<<grid, block, 0, stream1>>>(data_1);

kernel2<<<grid, block, 0, stream2>>>(data_2);
```

- Если cudaDeviceProp::concurrentKernels > 0 устройство может выполнять ядра параллельно

ПРИМЕР ПАРАЛЛЕЛЬНОГО КОПИРОВАНИЯ В ОБЕ СТОРОНЫ

- Пример:

```
cudaMallocHost(&aHost, size);

 cudaMallocHost(&bHost, size);
// создать потоки
cudaMemcpyAsync( aDev, aHost, size, cudaMemcpyHostToDevice,
stream1);

cudaMemcpyAsync( bHost, bDev, size, cudaMemcpyDeviceToHost,
stream2);

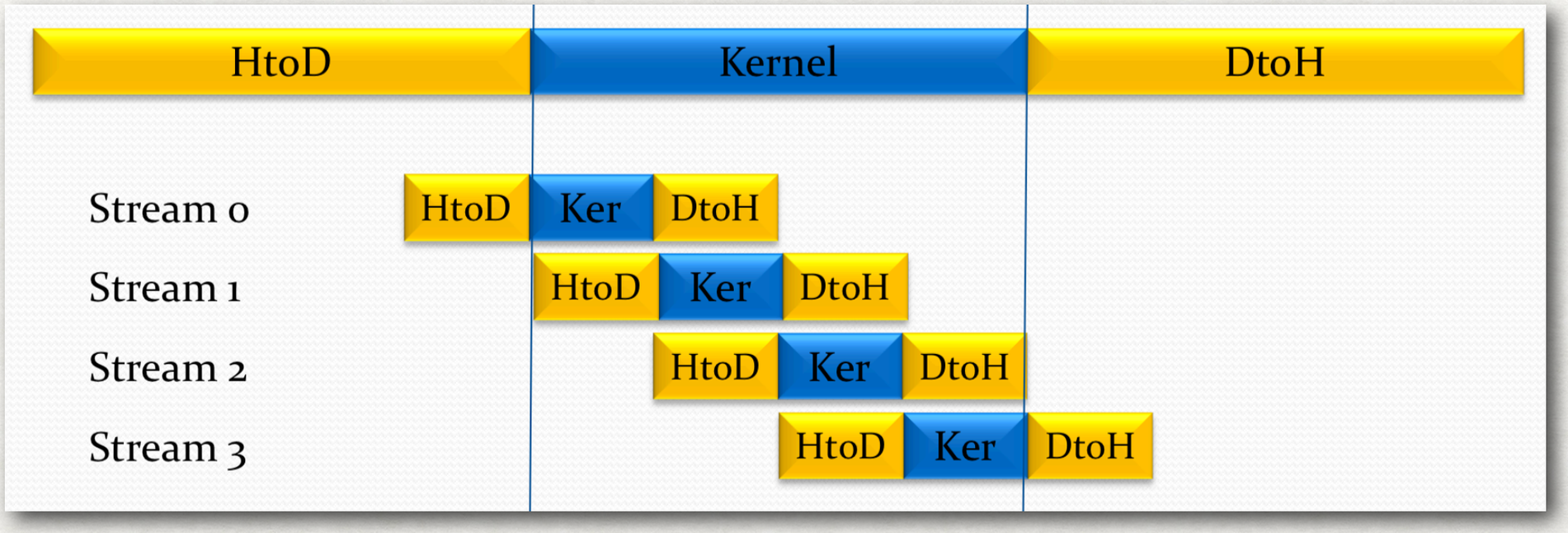
kernel<<<grid, block, 0, stream3>>>(...);
```

КОГДА СЛЕДУЕТ ПРИМЕНЯТЬ ПОТОКИ? ПРОСТОЙ СЛУЧАЙ

- Пусть в нашей задаче время копирований данных и время выполнения ядра сопоставимо
- Тогда можно разделить грид на части и запустим то же ядро на подгридах – результат не изменится
- Каждой подзадаче нужна только часть данных для старта
- Задачи

Было:

Стало:



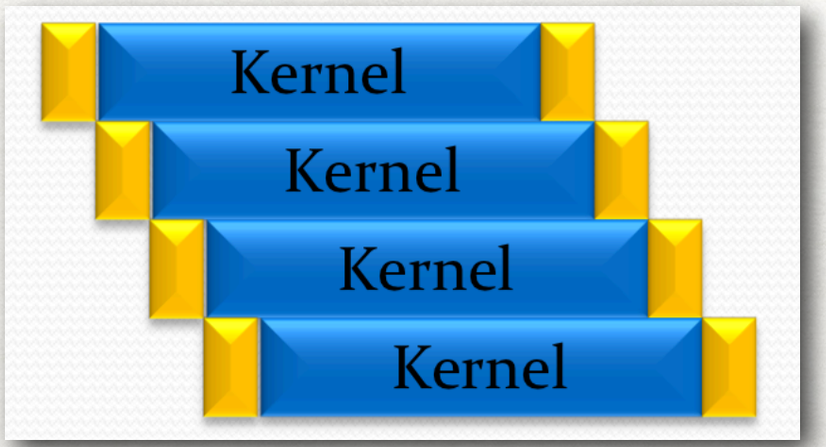
ПРИМЕР С НЕБОЛЬШИМИ КОПИРОВАНИЯМИ

- Ускорение в предыдущем случае достигается только за счет совмещения копирований и вычислений
- В случае, если вычислений значительно больше, чем копирований - ускорения получить не удасться, так как вычисления не станут производиться быстрее только за счет разбиения вычислительной части на подзадачи

Было:



Стало:



ВЫВОДЫ

- В случае, если в программе мало копирований с запусками ядер на больших гридах - не стоит пытаться ускорить программу за счет использования потоков

MULTI-GPU

ПРЕДПОСЫЛКА

- Зачем вообще использовать Multi GPU?
  - Нужно обрабатывать модели большего размера (не хватает памяти)
  - Нужны более быстрые вычисления
  - Нужна большая энергоэффективность

CUDA CONTEXT

- Контексты устройств неявно создаются при инициализации CUDA-runtime
- На каждом устройстве создается по одному контексту – «primary-контекст»
- Все нити программы совместно их используют
- Инициализация CUDA-runtime происходит неявно, при первом вызове любой функции, не относящейся к Device / Version Management (см. Toolkit Reference Manual)

CUDA CONTEXT

- В каждой CPU-нити может быть только один активный контекст в каждый момент времени
- cudaSetDevice(n) - переключение между устройствами (=между контекстами)
- cudaDeviceReset() - уничтожает primary-контекст, активный в данный момент
- При этом будет освобождена вся память, выделенная в контексте
- При необходимости, новый контекст будет неявно создан в дальнейшем

CUDA CONTEXT AND CUDA STREAM

- Функция cudaStreamCreate создает соответствующий ресурс в активном контексте
- Если активный контекст отличен от того, в котором создан поток/событие:
- Отправление команды в поток вызовет ошибку
- cudaEventRecord() для события вызовет ошибку
- cudaEventElapsedTime() вызовет ошибку, если события созданы в разных контекстах

ПРИМЕР

```
cudaSetDevice(0);
cudaStream_t s0;
cudaStreamCreate(&s0); // создать поток на device 0

cudaSetDevice(1);// переключить контекст на device 1

cudaStream_t s1;
cudaStreamCreate(&s1); // создать поток на device 1

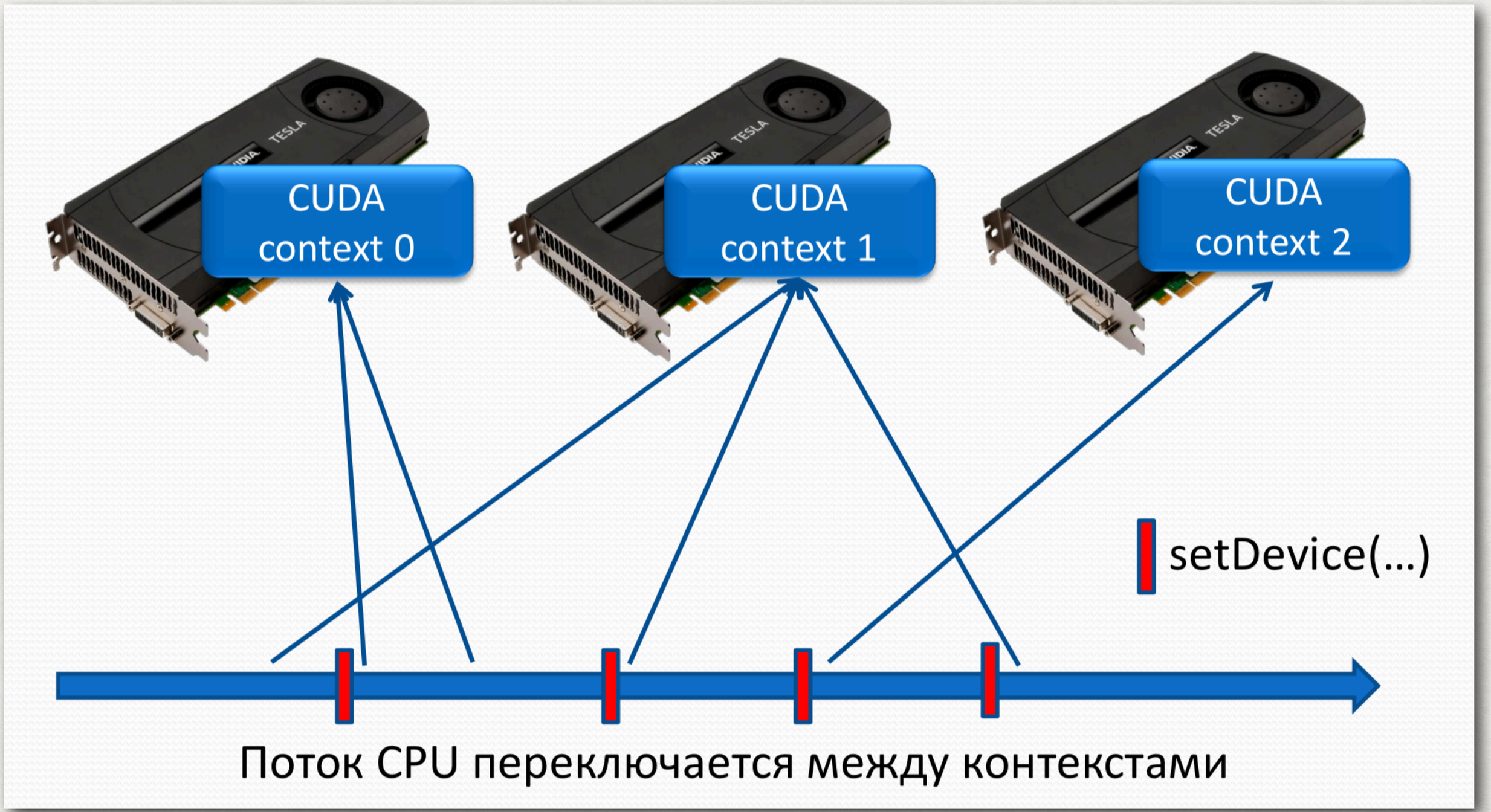
MyKernel<<<100, 64, 0, s1>>>();
MyKernel<<<100, 64, 0, s0>>>(); // ошибка !!
```

- А сделать как правильно?

MULTI GPU

- Два основных возможных подхода:
  - Multi-GPU & single CPU thread (switching contexts)
  - Multi-GPU & multiple CPU threads (using OpenMP)

Multi-GPU & single CPU thread



Поток CPU переключается между контекстами

ПРИМЕР: ВЫДЕЛЕНИЕ ПАМЯТИ

- Пусть у нас есть deviceCount GPU-устройств
- cudaMalloc бликирует нить, выделения памяти будут производиться последовательно!

```
for(int device = 0; device < deviceCount; device++)
{
        cudaSetDevice(device);
        cudaMalloc(…);
}
```

ПРИМЕР: ЗАПУСК ЯДЕР

```
for(int device = 0; device < deviceCount; device++)
{
    cudaSetDevice(device);
    cudaMemcpyAsync(…);
    kernel<<<…>>> (….);
    cudaMemcpyAsync(…..);
}
```

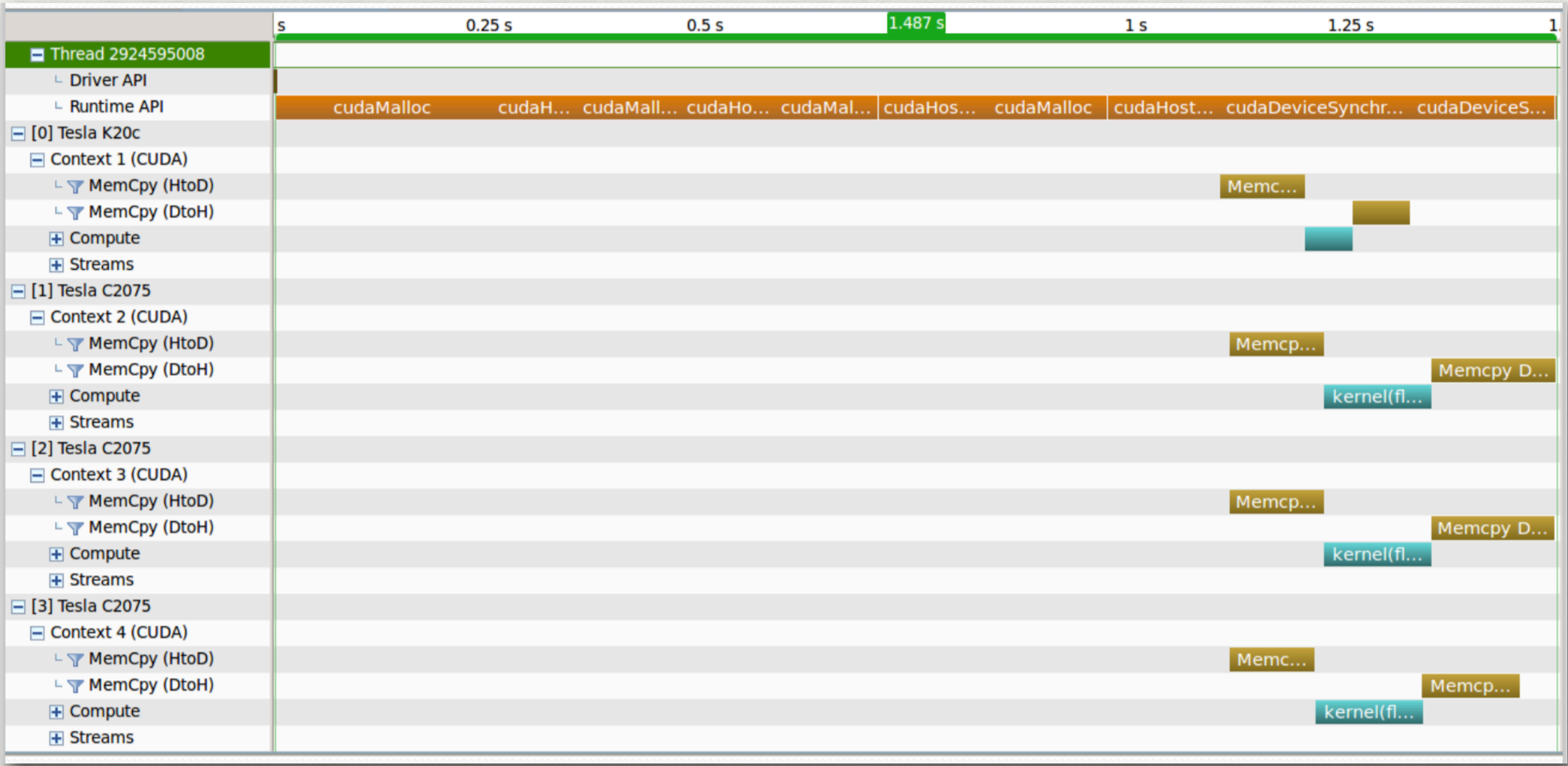
ПРИМЕР: СИНХРОНИЗАЦИЯ

- Выполняем синхронизацию в отдельном цикле!

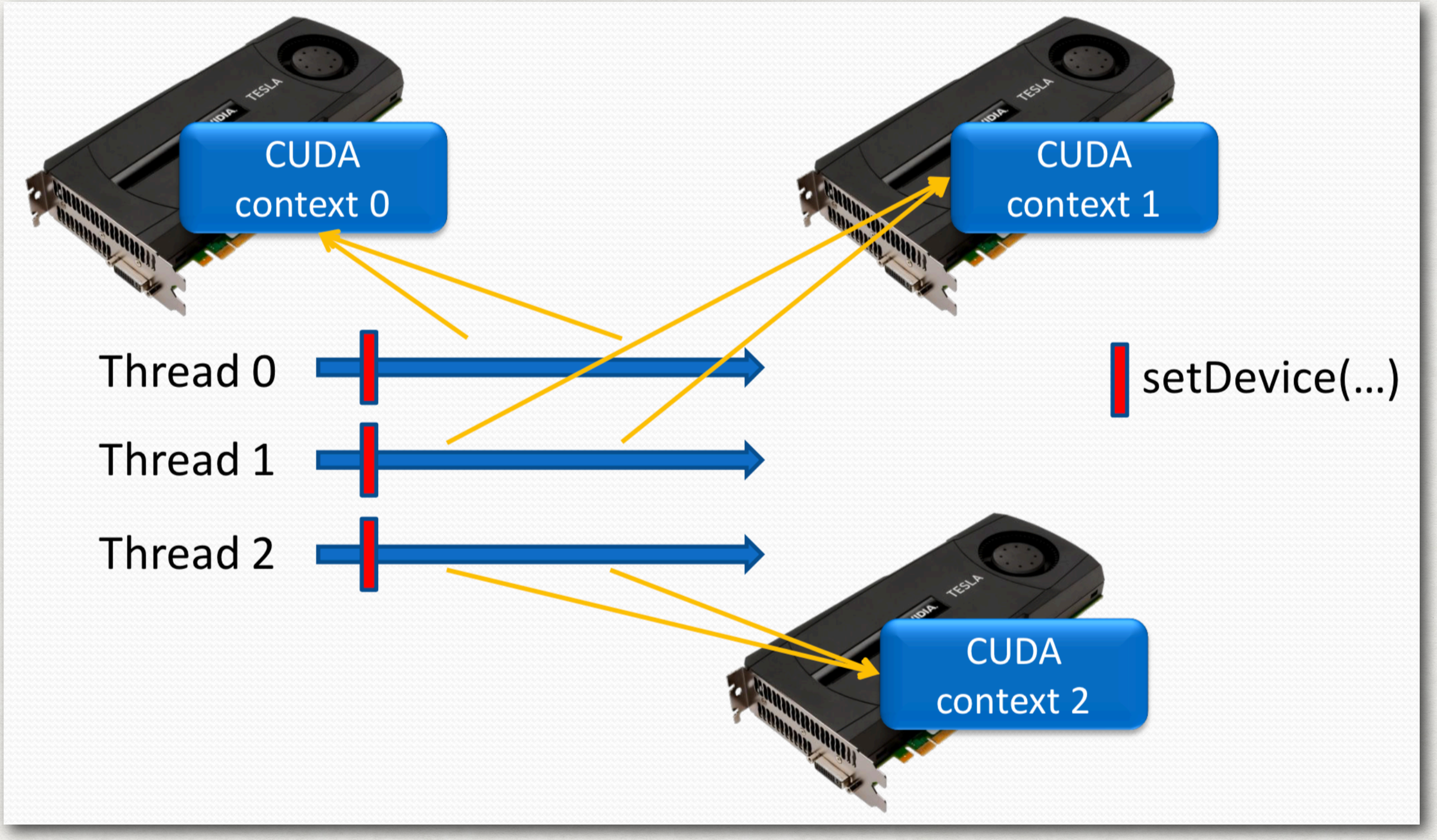
```
for(int device = 0; device < deviceCount; device++)
{
    cudaSetDevice(device);
    cudaDeviceSynchronize();
}
```

NVVP

Последовательные выделения памяти



MULTI-GPU & MULTIPLE CPU THREADS (OPENMP)



КАК КОМПИЛИРОВАТЬ?

- Поддержка OpenMP встроена в популярные компиляторы Intel icc/ifort, gcc/gfortran, MS cl, IBM xlc
- Обычный компилятор компилирует OpenMP директивы и функции при указании специального флага компиляции (для распознавания директив) и линковки (для линковки omp-функций)
- icc –openmp
- gcc –fopenmp
- cl -/openmp
- xlc -qsmp

OPENMP + NVCC

- Два подхода
- Единая компиляция:

```
nvcc -Xcompiler -fopenmp main.cu
```

- Раздельная компиляция:

```
nvcc kernel.cu
$gcc –fopenmp –I/opt/cuda/include main.c
$gcc –fopenmp –L/opt/cuda/lib –lcudart main.o kernel.o
```

- Раздельная компиляция предпочтительнее, так как могут использоваться более оптимизированные компиляторы для оптимизации CPU кода

УЛУЧШАЕМ ВЫДЕЛЕНИЕ ПАМЯТИ ПРИ ПОМОЩИ OPENMP

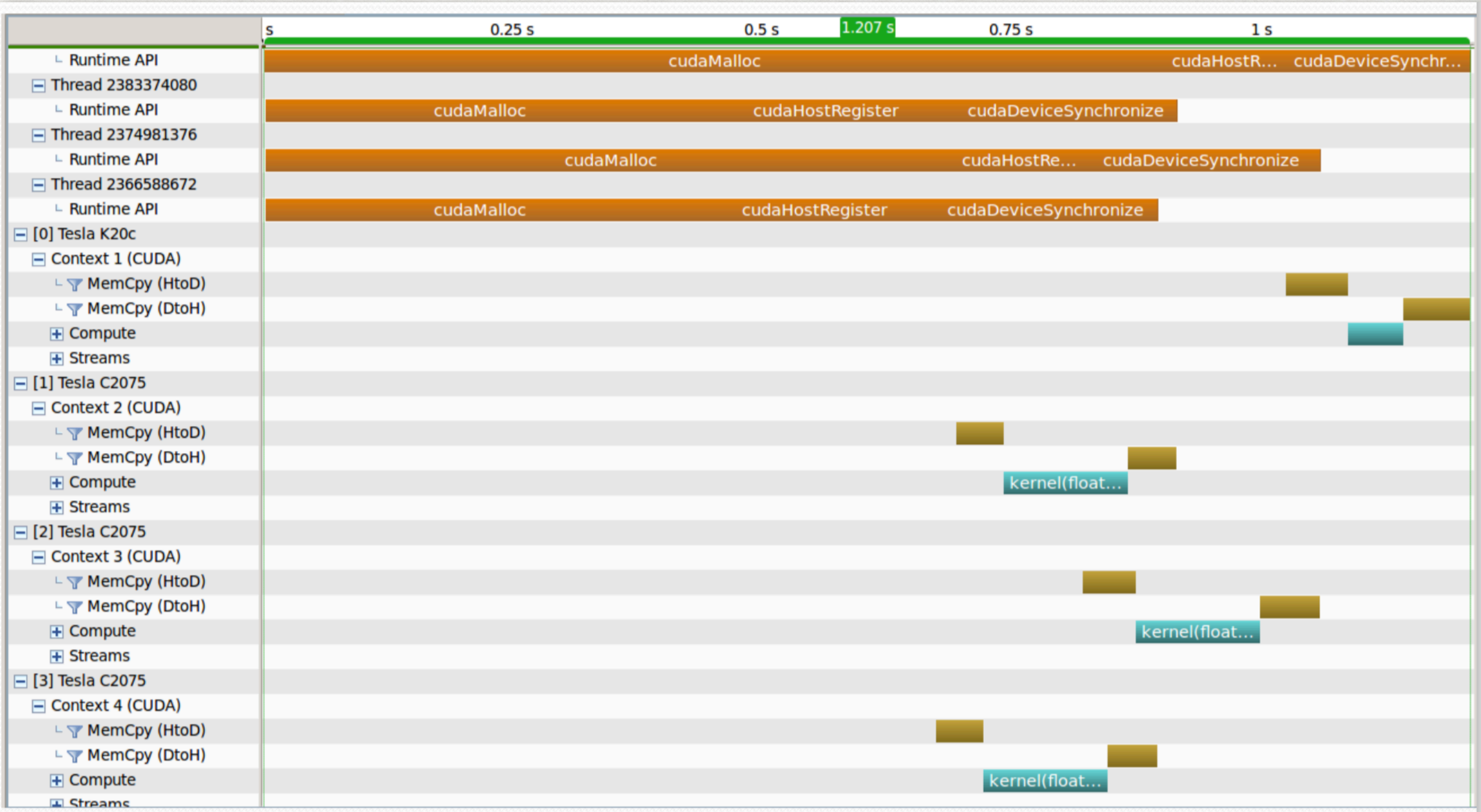
- Теперь выделения памяти будут происходить параллельно
- Не требуется отдельных циклов для синхронизации и запуска команд на GPU

```
int deviceCount;
cudaGetDeviceCount(&deviceCount);
#pragma omp parallel num_threads(deviceCount)
{
    int device = omp_get_thread_num();
    cudaSetDevice(device); // устанавливаем для каждого потока свой контекст
    cudaMalloc(devPtr + device, elemsPerDevice * sizeof(float));

    cudaHostRegister(…);


    // прочие команды, копирования, запуски ядер
    cudaDeviceSynchronize();
}
```

NVVP (MULTIPLE THREADS)

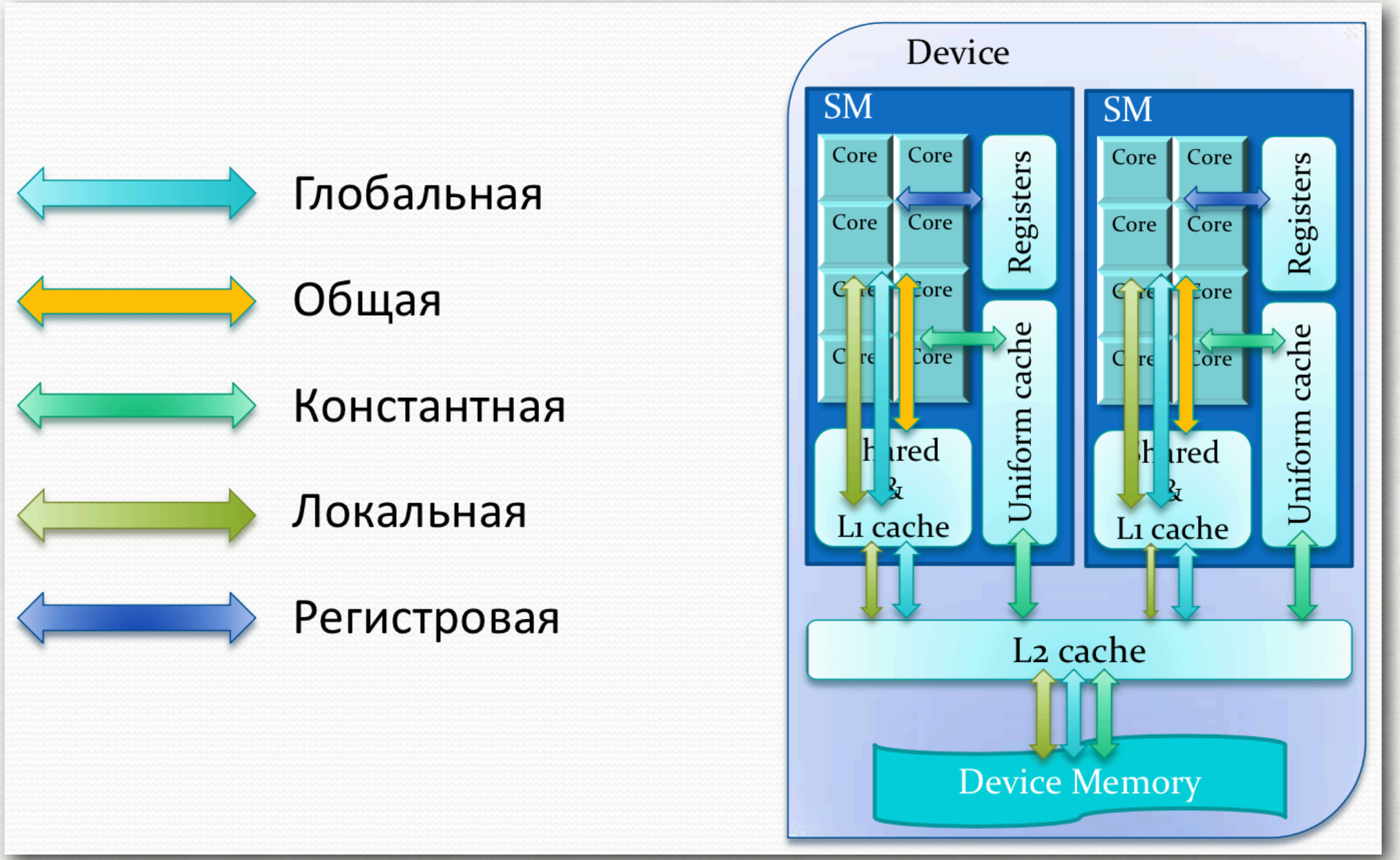


ИЕРАРХИЯ ПАМЯТИ GPU. ШАБЛОНЫ ДОСТУПА К ГЛОБАЛЬНОЙ ПАМЯТИ. РЕГИСТРОВАЯ ПАМЯТЬ.

АФАНАСЬЕВ ИЛЬЯ
AFANASIEV_ILYA@ICLOUD.COM

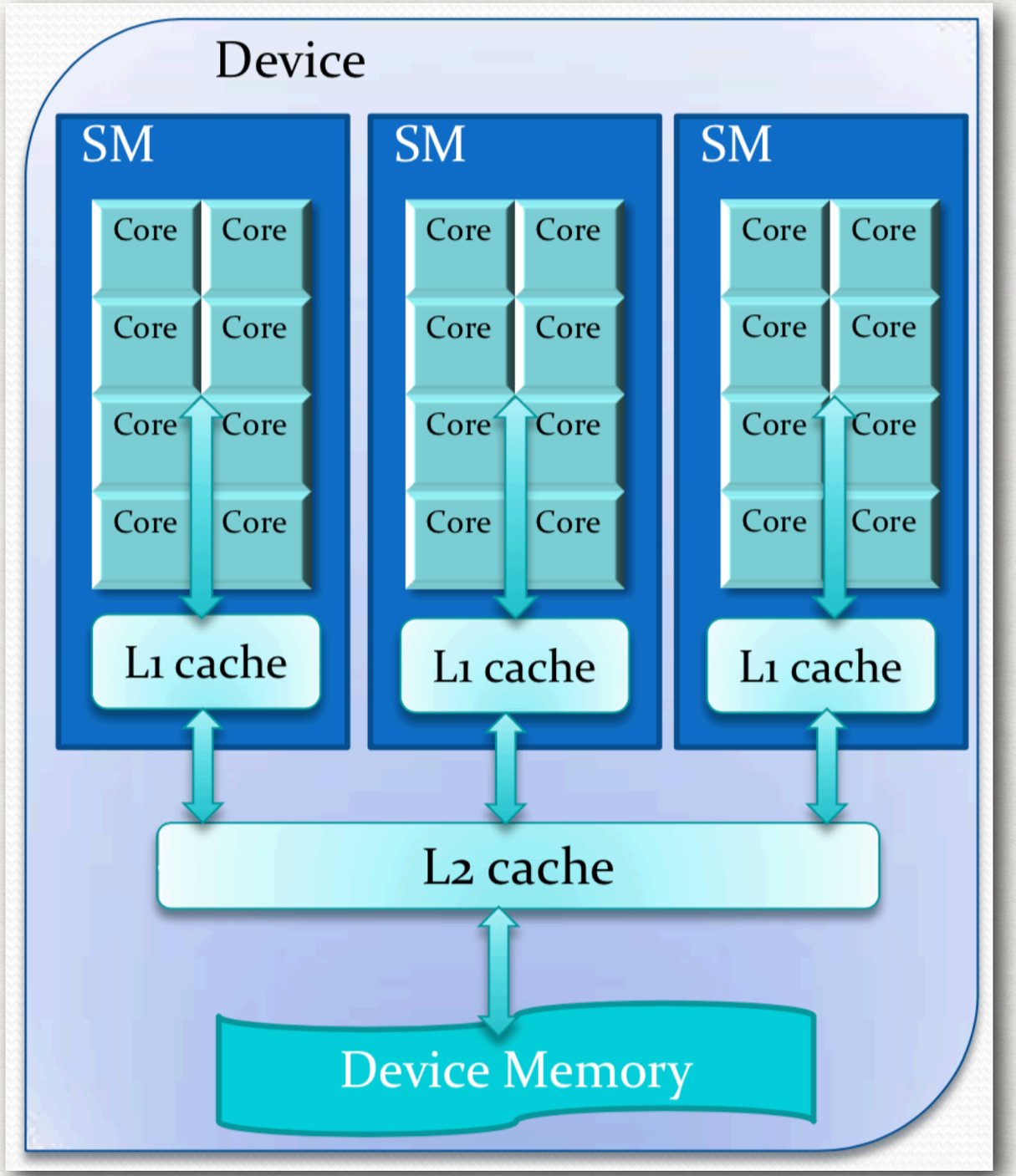
ИЕРАРХИЯ ПАМЯТИ GPU

- Иерархия памяти GPU достаточно сложна…
- Начнем рассмотрение с глобальной памяти, отвечающей за обработку транзакций, проходящих через кэши L1,L2 и device-память



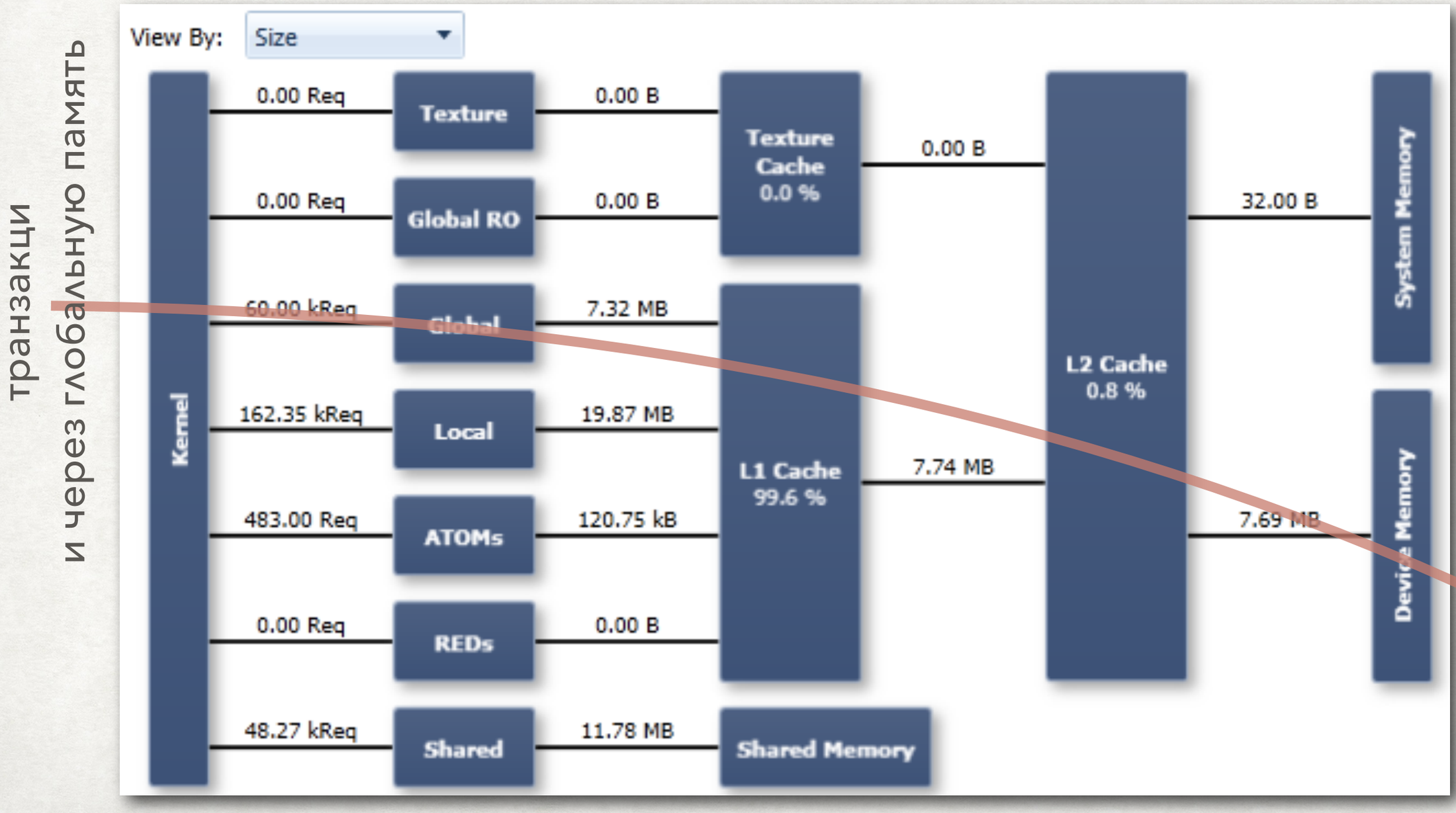
ГЛОБАЛЬНАЯ ПАМЯТЬ

- Расположена в DRAM GPU
- Объём 12-16Gb
- Обращения кешируются в кешах L1 и L2: обращение к памяти имеет вида L1 -> L2 -> DRAM
- L1 – на каждом мультипроцессоре максимальный размер - 48KB минимальный - 16KB
- L2 – на устройстве (разделяется всем мультипроцессорами), максимальный размер ~2MB



ИЕРАРХИЯ ПАМЯТИ GPU

- Глобальная память != Device память
- Обращения кешируются в кешах L1 и L2: обращение к памяти имеет вида L1 -> L2 -> DRAM



транзакции и через глобальную п

ГЛОБАЛЬНАЯ ПАМЯТЬ

- Выделение:
  - Динамически с хоста через cudaMalloc()
  - Динамически из ядер через malloc()
  - Статически - глобальная переменная с атрибутом __device__

ДИНАМИЧЕСКИ С ХОСТА

```
__global__ void kernel(int *arrayOnDevice)
{
    arrayOnDevice[threaIdx.x] = threaIdx.x;
}
int main()
{
    size_t size = 0;
    void *devicePtr = NULL;
    int hostMem[512];
    cudaMalloc(&devicePtr, sizeof(hostMem));
    cudaMemcpy(devicePtr, hostMem, size, cudaMemcpyHostToDevice);
    kernel<<<1,512>>>(devicePtr);
}
```

ВАРПЫ И ГЛОБАЛЬНАЯ ПАМЯТЬ

- Глобальная память оптимизирована с целью увеличения полосы пропускания при коллективных обращениях нитей к памяти
- Warp - группа из 32 нитей, выполняющих одну и ту же инструкцию в каждый момент времени (прошлые лекции)
- Как следствие, все нити одного варпа одновременно запрашивают данные из памяти (при выполнении инструкции работы с данными)
- Аппаратура обрабатывает обращения нитей варпов к памяти специализированным образом!

ТРАНЗАКЦИИ

- Транзакция – выполнение загрузки из глобальной памяти сплошного отрезка в 128 байт, с началом кратным 128 (naturally aligned)
- Инструкция обращения в память выполняется одновременно для всех нитей варпа (SIMT)
  - Выполняется столько транзакций, сколько нужно для покрытия обращений всех нитей варпа (32 нити)
  - Если нужен один байт – все равно загрузится 128
  - Выровненные 4х байтные обращения к подряд идущим данным будут выполнены за одну транзакцию (128 = 32 * 4)

ПРИМЕРЫ ТРАНЗАКЦИЙ (1)

Обращения нитей варпа



Все обращения умещаются в одну транзакцию

512	640	768

Обращения нитей варпа



Порядок не важен, главное, чтобы попадали в одну кеш-линию

512	640	768

ПРИМЕРЫ ТРАНЗАКЦИЙ (2, 3)

Обращения нитей варпа



Запрашивается 128 байт, но с не выровненного адреса

2 транзакции - 256 байт



Запрашивается 128 байт, но обращения разбросаны в пределах трёх кеш-линий -

3 транзакции – 384 байта

Обращения нитей варпа



А КАК ЖЕ КЭШИ?

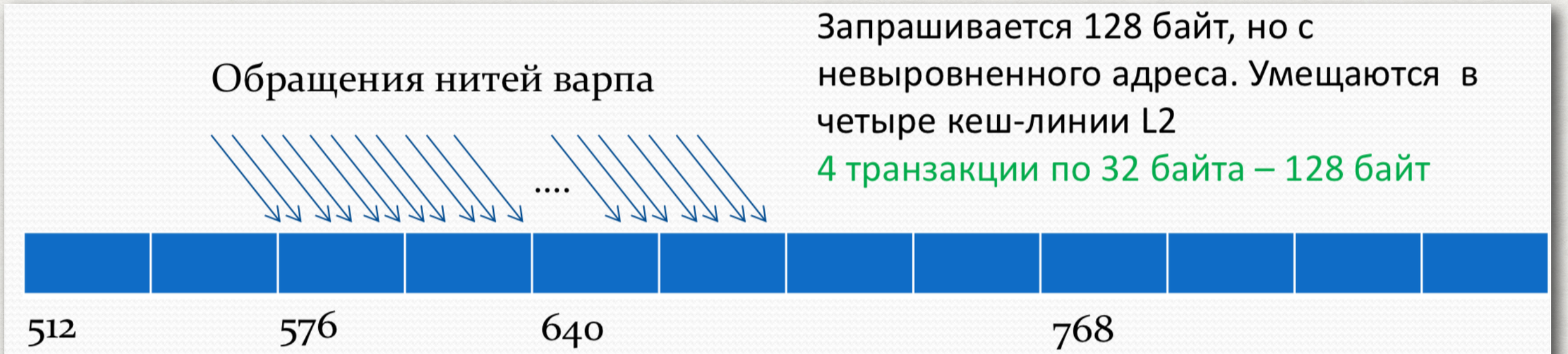
- На самом деле ядра взаимодействуют не с памятью напрямую, а с кешами
- Транзакция – выполнение загрузки кеш-линии
- У кеша L1 кеш-линии имеют размер 128байт
- У кэша L2 - 32байта
- Кеш грузит из памяти всю кеш-линию, даже если нужен один байт
- Можно обращаться в память минуя кеш L1 (!) - тогда все транзакции будут иметь размер 32 байта

ВКЛЮЧЕНИЕ И ОТКЛЮЧЕНИЕ L1

- Кеширование в L1 можно отключить при компиляции
- nvcc-Xptxas-dlcm=ca - кеширование в L1 **включено** (по умолчанию)
- nvcc-Xptxas-dlcm=cg - кэширование в L1 **выключено**
- **В зависимости от указанной опции в** бинарном коде обращения в глобальную память будут транслированы в инструкции, не использующие/не использующие кеш L1 при выполнении
- Различия именно на уровне бинарного кода – другие инструкции ассемблера

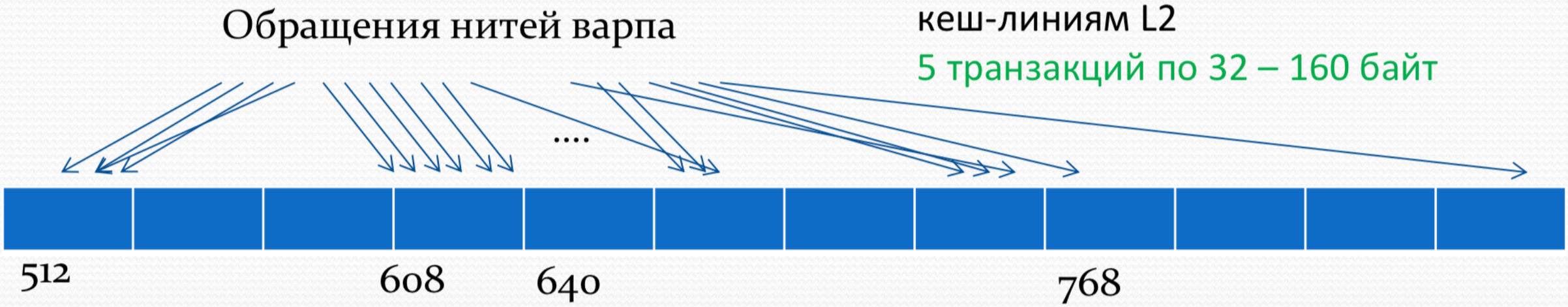
ПРИМЕР (L1 ВЫКЛЮЧЕН)

- L1 выключен, транзакции имеют размер 32
- Пусть нити варпа запрашивают 4х байтные данные (128 байт суммарно)



Запрашивается 128 байт, но с невыровненного адреса. Умещаются в четыре кеш-линии L2

4 транзакции по 32 байта – 128 байт

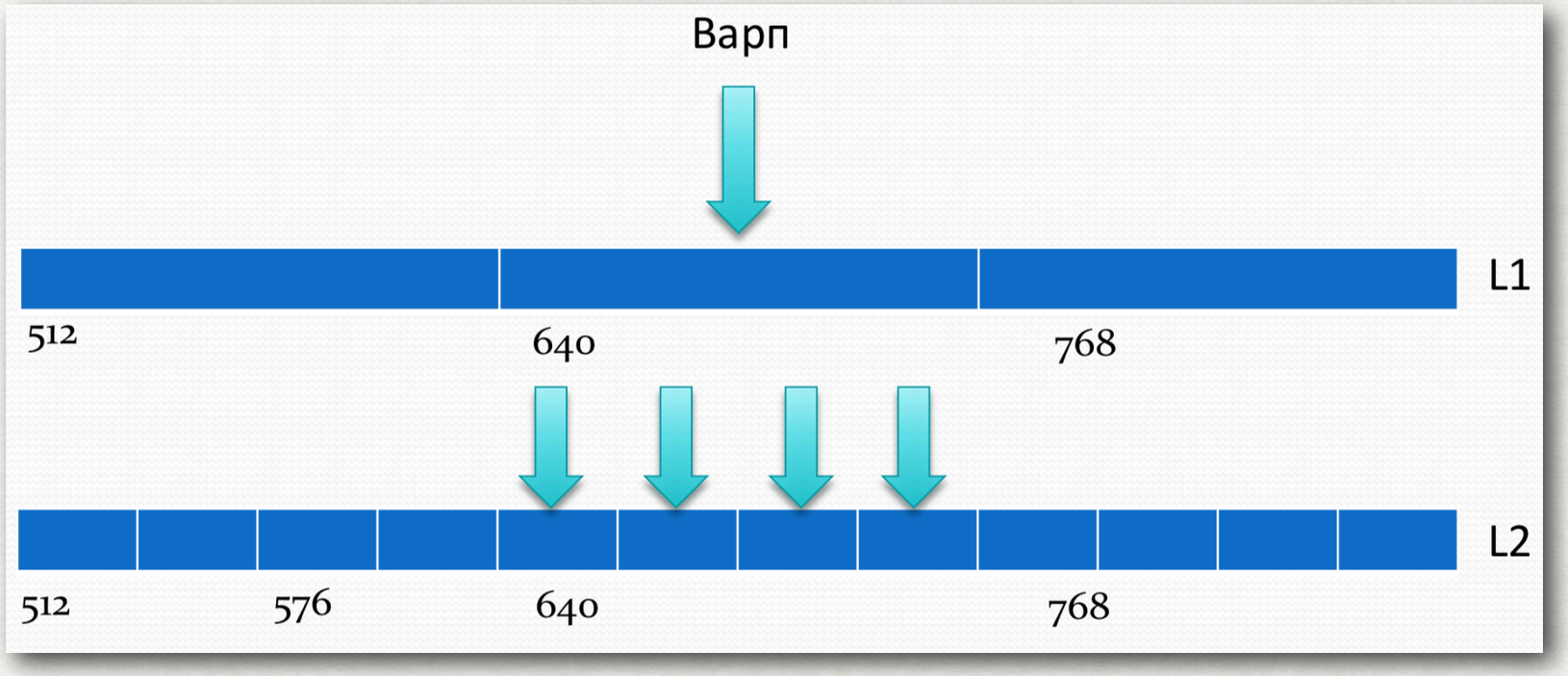


Запрашивается 128 байт, но обращения разбросаны по пяти кеш-линиям L2

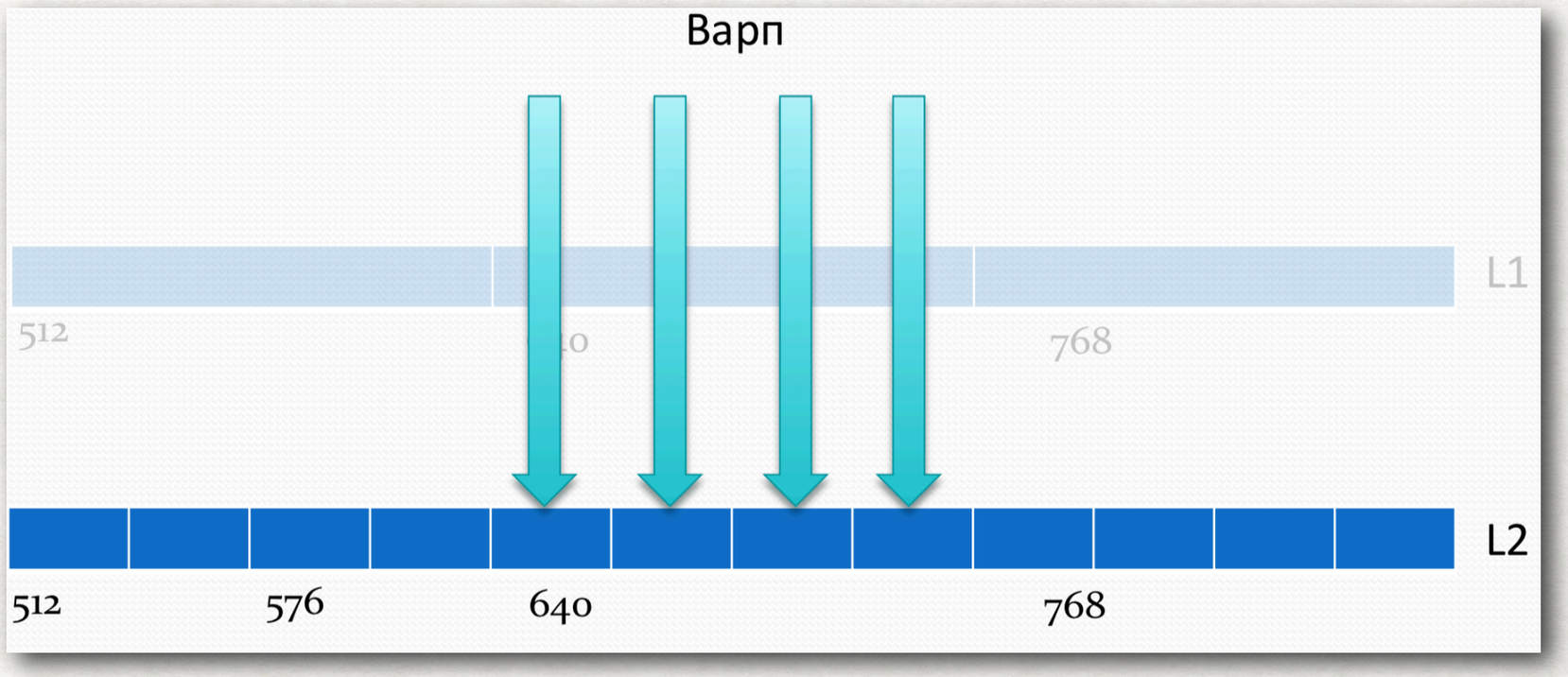
5 транзакций по 32 – 160 байт

ПРИМЕР (L1 ВКЛЮЧЕН/ВЫКЛЮЧЕН)

L1 включен



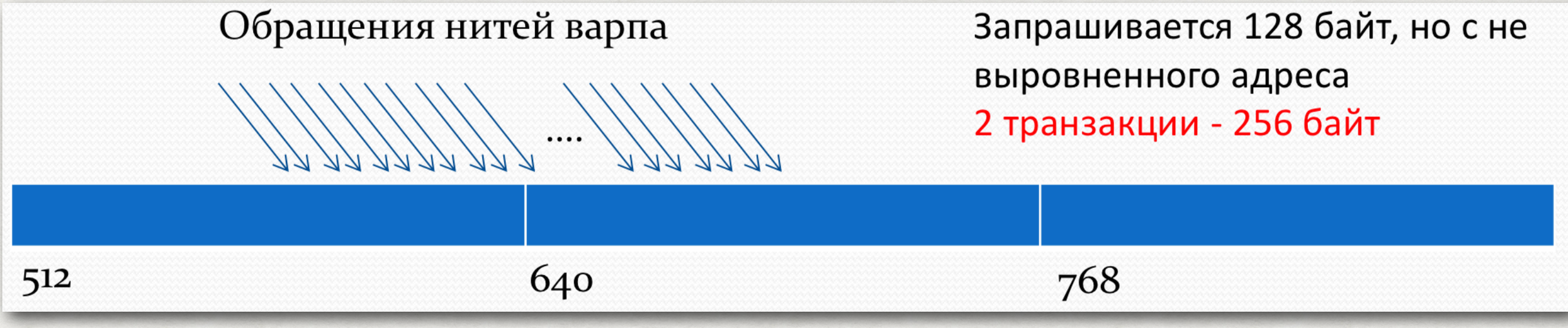
L1 выключен



ПРИМЕР (L1 ВКЛЮЧЕН/ВЫКЛЮЧЕН)

НЕ ВЫРОВНЕННЫЙ ПОСЛЕДОВАТЕЛЬНЫЙ ДОСТУП

L1 включен



Запрашивается 128 байт, но с не выровненного адреса

2 транзакции - 256 байт

L1 выключен



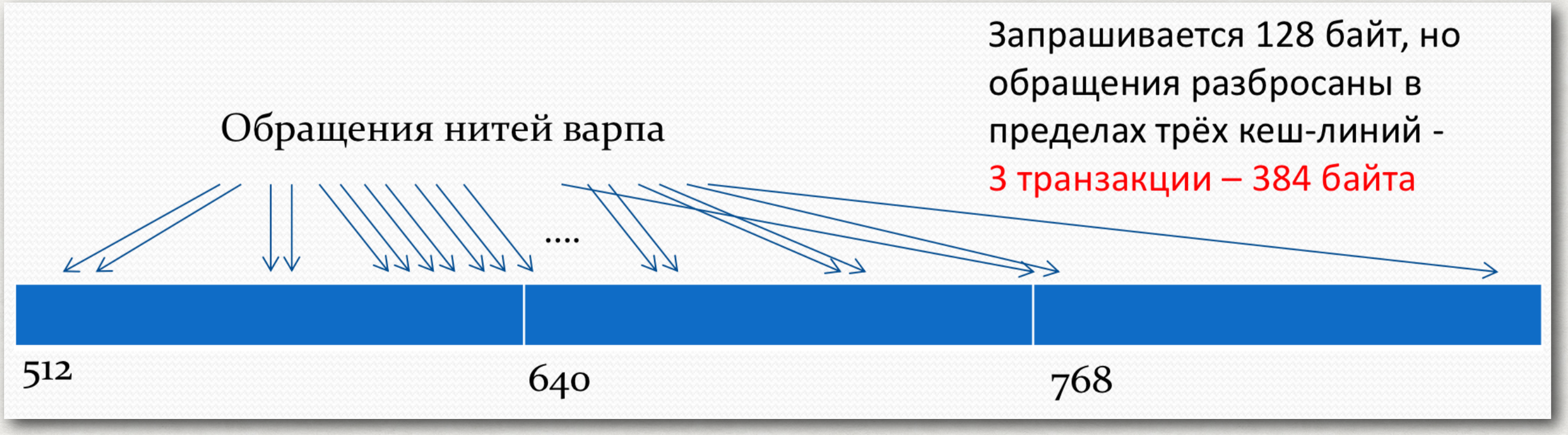
Запрашивается 128 байт, но с невыровненного адреса. Умещаются в четыре кеш-линии L2

4 транзакции по 32 байта – 128 байт

ПРИМЕР (L1 ВКЛЮЧЕН/ВЫКЛЮЧЕН)

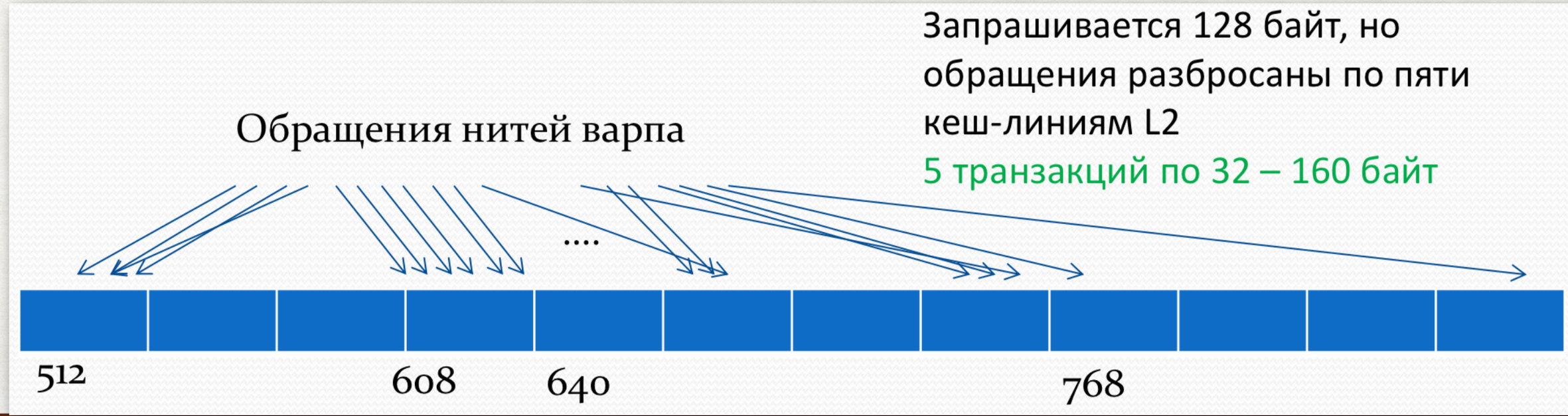
СЛУЧАЙНЫЙ ДОСТУП

L1 включен



Запрашивается 128 байт, но обращения разбросаны в пределах трёх кеш-линий -
3 транзакции – 384 байта

L1 выключен



Запрашивается 128 байт, но обращения разбросаны по пяти кеш-линиям L2
5 транзакций по 32 – 160 байт

ВЫВОДЫ

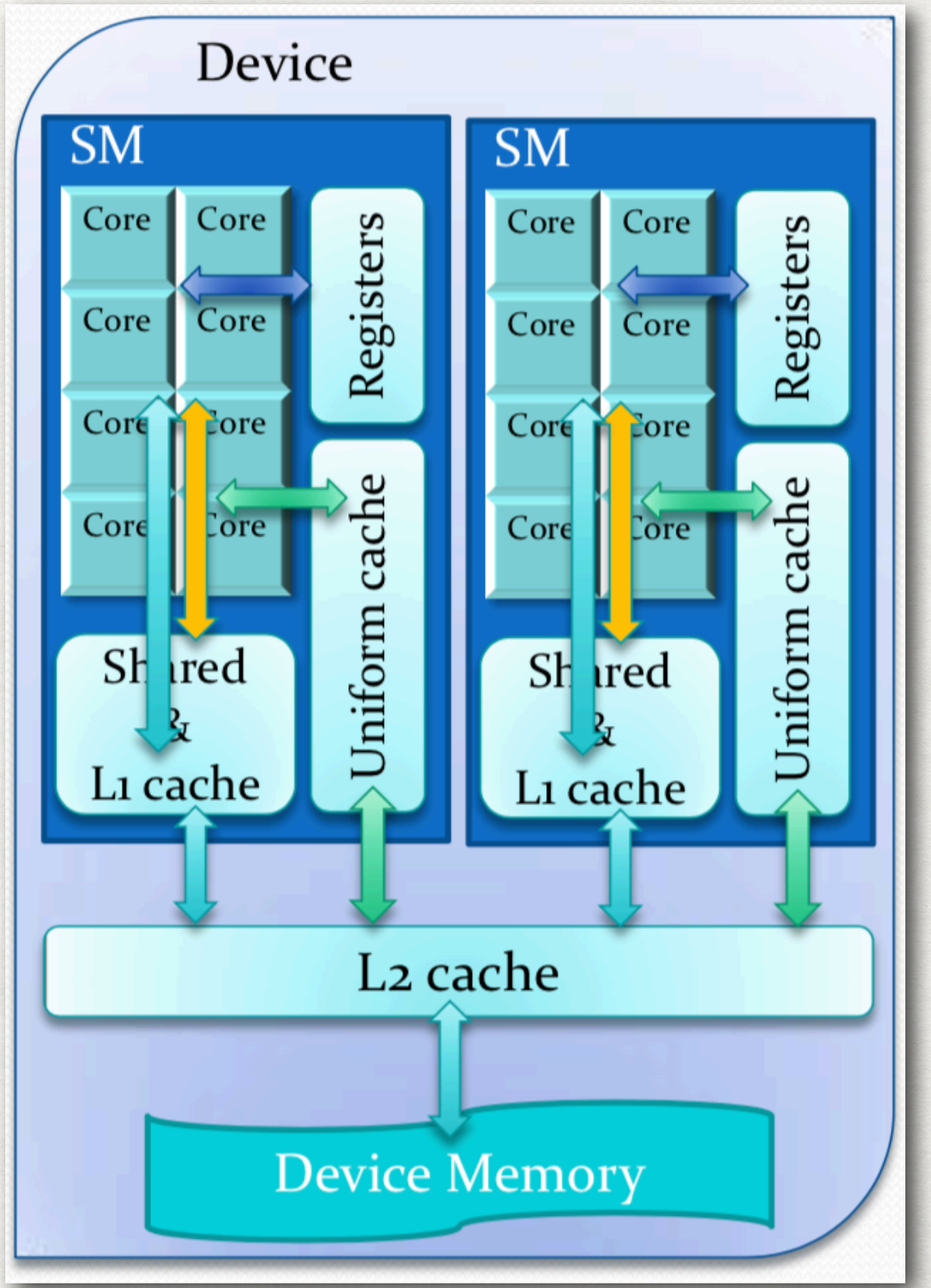
- Если в ядре не используется общая память (см. далее), то заведомо стоит включить cudaFuncCachePreferL1

- Если разреженный доступ – кеширование в L1 отключаем

- В общем случае, стоит проверить производительность работы всех 4-х вариантов: (-dlcm=ca, -clcm=cg)x(16KB, 48KB)

РЕГИСТРОВАЯ ПАМЯТЬ

- Расположена на мультипроцессоре
- Самая быстрая память
- Каждый мультипроцессор содержит 32768 32-битных регитров
  - 128 KB регистров
  - Параметр устройства regsPerBlock
- Отдельной нити доступно максимум 63 регистра
- Регистры распределяются во время компиляции
- Каждая нить является эксклюзивным пользователем своих регистров на всё время выполнения ядра. Доступ к регистрам других нитей запрещён



ВОПРОСЫ?

БИБЛИОТЕКИ С ПОДДЕРЖКОЙ CUDA

АФАНАСЬЕВ ИЛЬЯ
AFANASIEV_ILYA@ICLOUD.COM

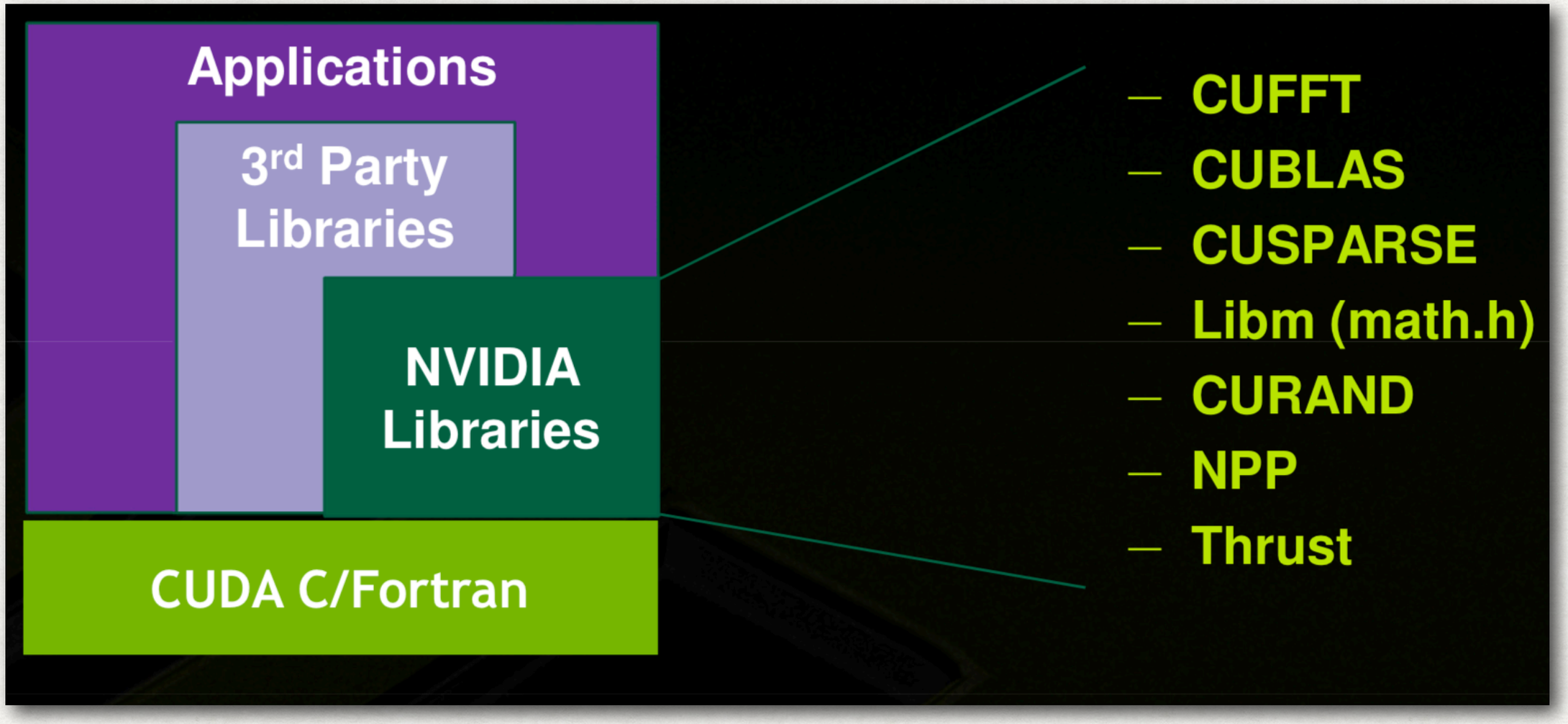
CUDA LIBRARIES

- Представляют собой готовые к использованию базовые блоки для построения программ
- Просты в использовании
- Оптимизированы для максимальной производительности
- Идеальны для приложений, которые уже используют аналогичные CPU библиотеки

CUDA LIBRARIES

- Доступны следующие библиотеки:
  - cuFFT – Fast Fourier Transforms Library
  - cuBLAS – Complete BLAS Library
  - cuSPARSE – Sparse Matrix Library
  - cuRAND – Random Number Generation (RNG) Library
  - NPP – Performance Primitives for Image & Video Processing
  - Thrust – Templated C++ Parallel Algorithms & Data
  - Structures math.h - C99 floating-point Library
  - NVGraph - graph processing library
- Включены в CUDA Toolkit

CUDA LIBRARIES



О ЧЁМ СЕГОДНЯ ПОЙДЕТ РЕЧЬ

- Краткий обзор функциональности библиотек, примеры их использования
- Для более подробных деталей использования каждой библиотеки следует обратиться к официальной документации

CUBLAS

CUBLAS

- Реализация BLAS (Basic Linear Algebra Subprograms) ? для GPU
- Поддерживает все функции из BLAS
  - Level1 (вектор-вектор): O(N) ?

    AXPY : y=alpha.x+y
    DOT : dot=x.y

  - Level 2(матрица-вектор): O(N2)

    Vector multiplication by a General Matrix : GEMV

    Triangular solver : TRSV

  - Level3(матрица-матрица): O(N3)

    General Matrix Multiplication : GEMM

    Triangular Solver : TRSM

- Как и BLAS, CUBLAS хранит матрицы по столбцам, а не строкам (как в Fortran)

CUBLAS

- Поддерживает 4 типа – Float, Double, Complex, Double Complex – соотвествующие префиксы: S, D, C, Z
- Содержит 152 процедуры : S(37), D(37), C(41), Z(41)
  Принцип построения названия функций: cublas + BLAS name
- Пример: cublasSGEMM
  - S: single precision (float)
  - GE: general
  - M: multiplication
  - M: matrix

CUBLAS

- Интерфейс CUBLAS библиотеки расположен в файле cublas.h
- Функции имеют имена в соответствии со следующей конвенцией: cublas + BLAS name (напр, cublasSGEMM)
- Обработка ошибок
- Вычислительные функции CUBLAS не возвращают ошибки
- CUBLAS позволяет получить последнюю произошедшую ошибку при помощи случайных функций
- Вспомошательные функции CUBLAS:
  Memory allocation, data transfer
- Пример использования библиотеки будет рассмотрен далее после CURAND

CURAND

CURAND

- Генерировать качественные последовательности случайных чисел параллельно не так просто
- Не делайте это сами, используйте библиотеку!
- Большой набор генераторов случанйных чисел для различных распределений: XORWOW, MRG323ka, MTGP32, (scrambled) Sobol, uniform, normal, log-normal
- Двойная и одинарная точность:
- У CURAND есть 2 API:
  API для CPU:подходит для генерации больших порций случайнх чисел на
- API для GPU: в случае, когда случайные числа должны быть генерированы внутри ядра

ИСПОЛЬЗОВАНИЕ CURAND

1. Инициализировать генератор:

curandCreateGenerator()

2. Устанавливаем seed:

curandSetPseudoRandomGeneratorSeed()

3. Генерируем данные задав распределение:

curandGenerateUniform()/(curandGenerateUniformDouble(): Uniform
curandGenerateNormal()/cuRandGenerateNormalDouble(): Gaussian
curandGenerateLogNormal/curandGenerateLogNormalDouble(): Log-Normal

4. Уничтожаем генератор:

curandDestroyGenerator()

CURAND API

- Генерируем множество случайных чисел разом на центральном процессоре, CPU API:

```
#include <curand.h>

curandCreateGenerator(&gen, CURAND_RNG_PSEUDO_DEFAULT); curandGenerateUniform(gen, d_data, n);
```

- Генерируем по одному случайному числу на каждой из GPU нитей, GPU API:

```
#include <curand_kernel.h>
__global__ void generate_kernel(curandState *state)

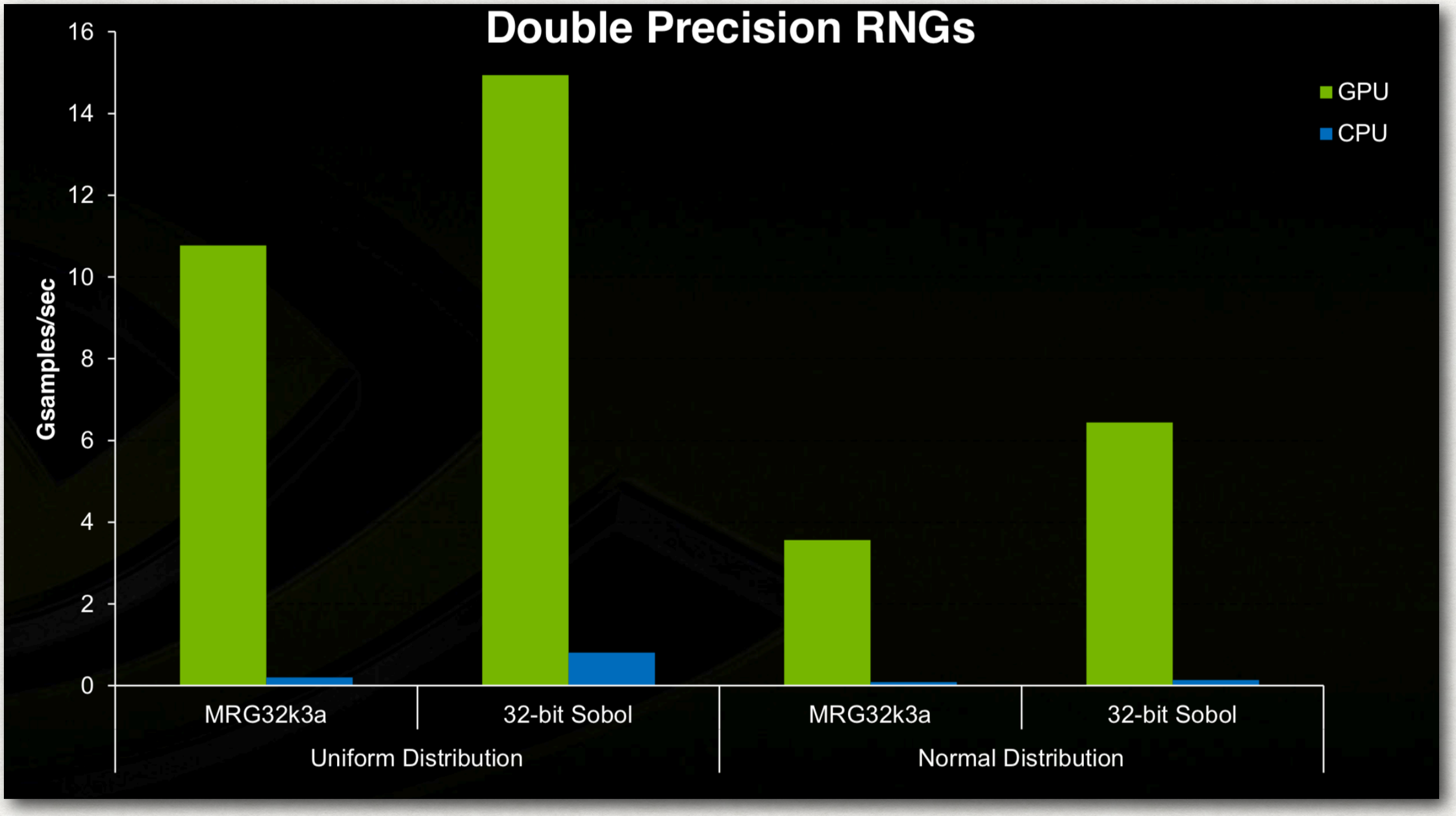
{

        int id = threadIdx.x + blockIdx.x * 64;

        x = curand(&state[id]);

}
```

CURAND PERFORMANCE



CUBLAS + CURAND MATRIX MULTIPLICATION EXAMPLE

```
1  #include <cstdlib>
2
3  int main() {
4      // Allocate 3 arrays on CPU
5      int nr_rows_A, nr_cols_A, nr_rows_B, nr_cols_B, nr_rows_C, nr_cols_C;
6
7      // for simplicity we are going to use square arrays
8      nr_rows_A = nr_cols_A = nr_rows_B = nr_cols_B = nr_rows_C = nr_cols_C = 3;
9
10     float *h_A = (float *)malloc(nr_rows_A * nr_cols_A * sizeof(float));
11     float *h_B = (float *)malloc(nr_rows_B * nr_cols_B * sizeof(float));
12     float *h_C = (float *)malloc(nr_rows_C * nr_cols_C * sizeof(float));
13
14     // Allocate 3 arrays on GPU
15     float *d_A, *d_B, *d_C;
16     cudaMalloc(&d_A,nr_rows_A * nr_cols_A * sizeof(float));
17     cudaMalloc(&d_B,nr_rows_B * nr_cols_B * sizeof(float));
18     cudaMalloc(&d_C,nr_rows_C * nr_cols_C * sizeof(float));
19
20     // ....
21
22     //Free GPU memory
23     cudaFree(d_A);
24     cudaFree(d_B);
25     cudaFree(d_C);
26
27     // Free CPU memory
28     free(h_A);
29     free(h_B);
30     free(h_C);
31
32     return 0;
33 }
```

GENERATING RANDOM MATRICES

```
1  ...
2  #include <curand.h>
3  ...
4
5  // Fill the array A(nr_rows_A, nr_cols_A) with random numbers on GPU
6  void GPU_fill_rand(float *A, int nr_rows_A, int nr_cols_A) {
7      // Create a pseudo-random number generator
8      curandGenerator_t prng;
9      curandCreateGenerator(&prng, CURAND_RNG_PSEUDO_DEFAULT);
10
11     // Set the seed for the random number generator using the system
clock
12     curandSetPseudoRandomGeneratorSeed(prng, (unsigned long long)
clock());
13
14     // Fill the array with random numbers on the device
15     curandGenerateUniform(prng, A, nr_rows_A * nr_cols_A);
16 }
17
18 ...
```

PERFORMING MULTIPLICATION

```
 1 ...
 2 #include <cublas_v2.h>
 3 ...
 4
 5 // Multiply the arrays A and B on GPU and save the result in C
 6 // C(m,n) = A(m,k) * B(k,n)
 7 void gpu_blas_mmul(const float *A, const float *B, float *C, const int m, const int k,
const int n) {
 8     int lda=m,ldb=k,ldc=m;
 9     const float alf = 1;
10     const float bet = 0;
11     const float *alpha = &alf;
12     const float *beta = &bet;
13
14     // Create a handle for CUBLAS
15     cublasHandle_t handle;
16     cublasCreate(&handle);
17
18     // Do the actual multiplication
19     cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, m, n, k, alpha, A, lda, B, ldb, beta,
C, ldc);
20
21     // Destroy the handle
22     cublasDestroy(handle);
23 }
```

ИСПОЛЬЗУЕМ THRUST ДЛЯ УПРОЩЕНИЯ ПРИМЕРА С УМНОЖЕНИЕМ МАТРИЦ

```
int main() {
    // Allocate 3 arrays on CPU
    int nr_rows_A, nr_cols_A, nr_rows_B, nr_cols_B, nr_rows_C, nr_cols_C;

    // for simplicity we are going to use square arrays
    nr_rows_A = nr_cols_A = nr_rows_B = nr_cols_B = nr_rows_C = nr_cols_C = 3;

    thrust::device_vector<float> d_A(nr_rows_A * nr_cols_A), d_B(nr_rows_B * nr_cols_B), d_C(nr_rows_C *
nr_cols_C);

    // Fill the arrays A and B on GPU with random numbers
    GPU_fill_rand(thrust::raw_pointer_cast(&d_A[0]), nr_rows_A, nr_cols_A);
    GPU_fill_rand(thrust::raw_pointer_cast(&d_B[0]), nr_rows_B, nr_cols_B);

    // Optionally we can print the data
    std::cout << "A =" << std::endl;
    print_matrix(d_A, nr_rows_A, nr_cols_A);
    std::cout << "B =" << std::endl;
    print_matrix(d_B, nr_rows_B, nr_cols_B);

    // Multiply A and B on GPU
    gpu_blas_mmul(thrust::raw_pointer_cast(&d_A[0]), thrust::raw_pointer_cast(&d_B[0]),
thrust::raw_pointer_cast(&d_C[0]), nr_rows_A, nr_cols_A, nr_cols_B);

    //Print the result
    std::cout << "C =" << std::endl;
    print_matrix(d_C, nr_rows_C, nr_cols_C);

    return 0;
}
```

CUSPARSE

CUSPARSE

- Различные форматы хранения разреженных матриц
  - ✓ COO (наиболее общий)
  - ✓ CSR
  - ✓ CSC
  - ✓ DIAG
  - ✓ ELL
  - ✓ HYB (ELL + CSR)
- В основном матрично-векторно операции
- Наименования функций по следующей конвенции:

cusparse<Type>[<sparse data format>]<operation>[<sparse data format>]

- Например: single precision, sparse matrix (in csr storage) x dense vector => cusparseScsrmv

CUSPARSE ФОРМАТЫ

Coordinate (COO) Format

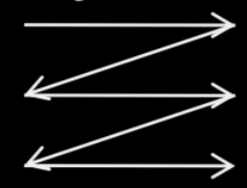
Row Index	1	2	2	3	4	4	4
Col Index	1	1	2	3	1	3	4
Values	1.0	2.0	3.0	4.0	5.0	6.0	7.0

	1	2	3	4
1	1.0			
2	2.0	3.0		
3			4.0	
4	5.0		6.0	7.0

Compressed Sparse Row (CSR)

Row Ptr	1	2	4	5	8		
Col Index	1	1	2	3	1	3	4
Values	1.0	2.0	3.0	4.0	5.0	6.0	7.0

row-major order



Compressed Sparse Column (CSC)

Row Index	1	2	4	2	3	4	4
Col Ptr	1	4	5	7	8		
Values	1.0	2.0	3.0	4.0	5.0	6.0	7.0

column-major order



PRESENTED BY NVIDIA.

CUSPARSE EXAMPLES

- Level 2 and 3
  - ✓ csrmv (matrix-vector multiplication)

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \alpha \begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix} \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \end{bmatrix} + \beta \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

  - ✓ csrmm (matrix-tall-matrix multiplication)

$$\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ x_{41} & x_{42} \end{bmatrix} = \alpha \begin{bmatrix} 1.0 & & & \\ 2.0 & 3.0 & & \\ & & 4.0 & \\ 5.0 & & 6.0 & 7.0 \end{bmatrix} \begin{bmatrix} 1.0 & 5.0 \\ 2.0 & 6.0 \\ 3.0 & 7.0 \\ 4.0 & 8.0 \end{bmatrix} + \beta \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ x_{41} & x_{42} \end{bmatrix}$$

NPP

NPP

- C/c++ примтивы (primitives) – low level API:
- Простая интеграция в код
- Примерно 350 функций обработки изображений
- Примерно 100 функций обработки сигналов

THRUST

THRUST

- Библиотека шаблонов для CUDA – аналогична C++ STL
- Контейнеры – управление паматью на хосте и устрйостве: thrust::host_vector<T>, thrust:device_vector<T> – позволяет избегать множества типичных ошибок
- Итераторы
  – Автоматически перемещает данные между хостом и устройством
  – d_vec.begin()
- Алгоритмы
  – Сортировка, редукция, scan, и.т.д.: thrust::sort()

THRUST: АЛГОРИТМЫ

- Операции над элементами векторов:
  – for_each, transform, gather, scatter ...

- Редукции:
  – reduce, inner_product, reduce_by_key ...

- Префикс-суммы:
  – inclusive_scan, inclusive_scan_by_key ...

- Сортировки
  – sort, stable_sort, sort_by_key ...

THRUST: ПРИМЕР

```
#include <thrust/host_vector.h>

#include <thrust/device_vector.h>

#include <thrust/sort.h>

#include <cstdlib.h>


int main(void)

{

        // generate 32M random numbers on the host

        thrust::host_vector<int> h_vec(32 << 20);

        thrust::generate(h_vec.begin(), h_vec.end(), rand);

        // transfer data to the device

        thrust::device_vector<int> d_vec = h_vec;

        // sort data on the device (846M keys per sec on GeForce GTX 480)

        thrust::sort(d_vec.begin(), d_vec.end());

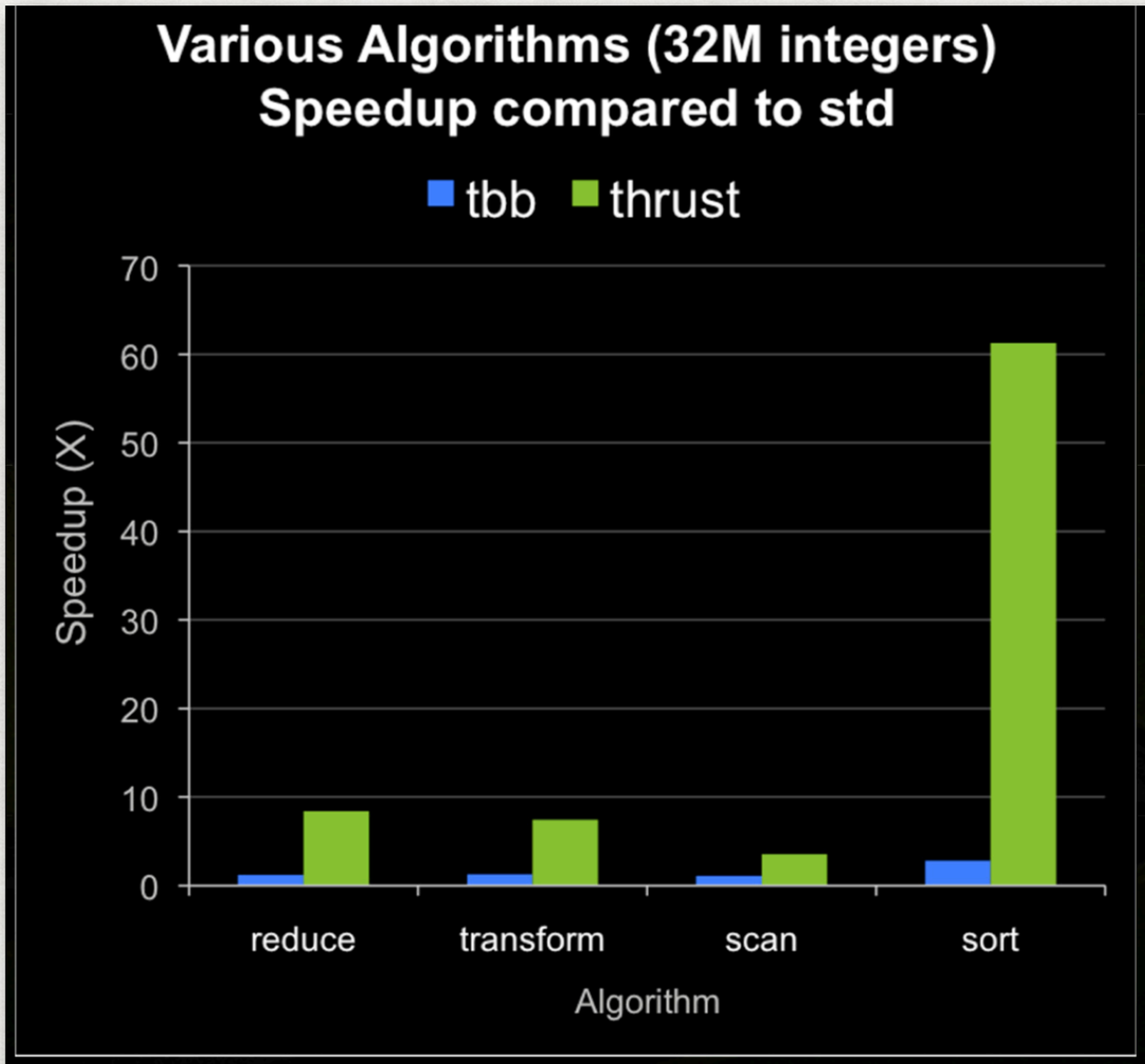
        // transfer data back to host

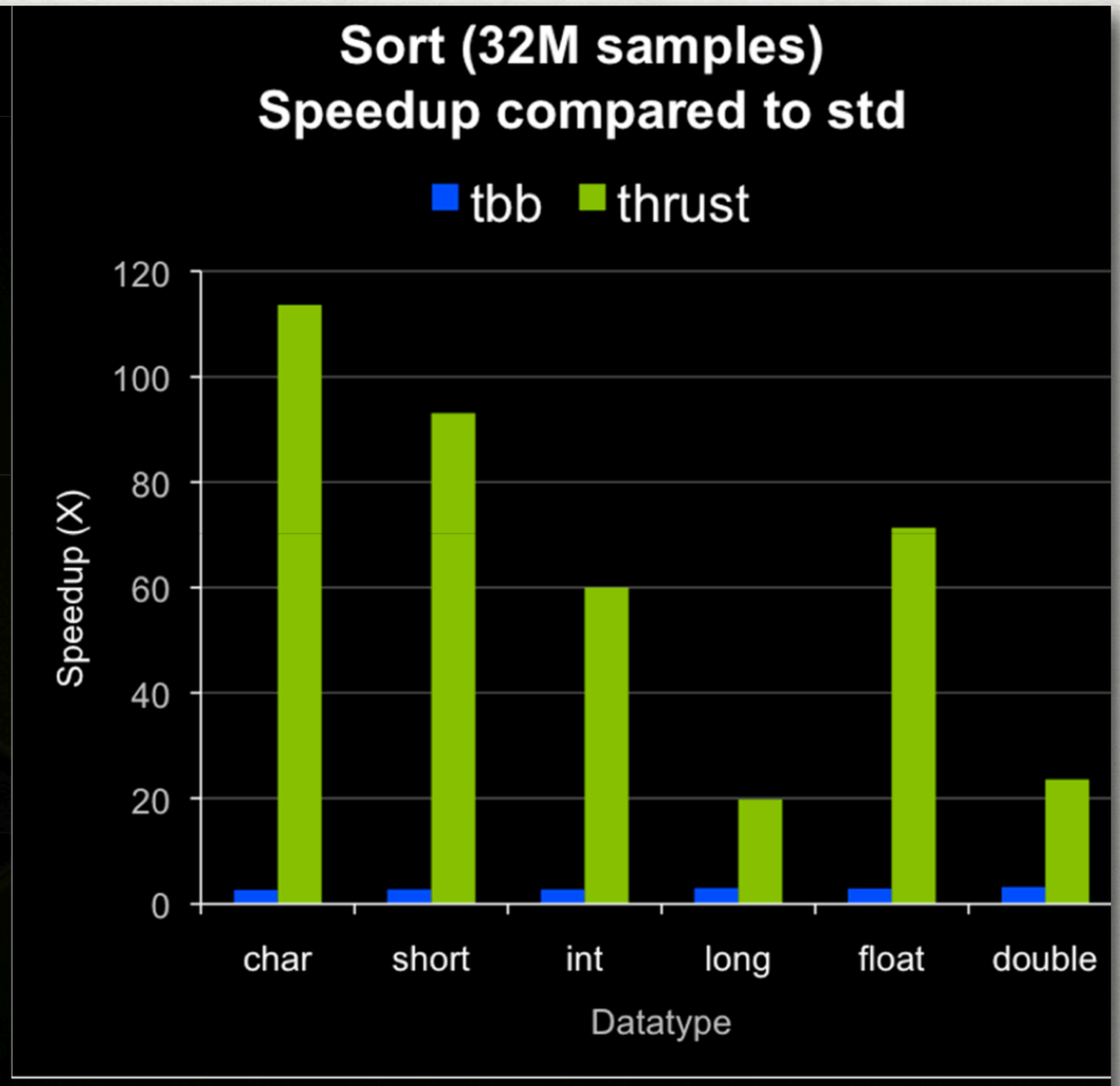
        thrust::copy(d_vec.begin(), d_vec.end(), h_vec.begin());

        return 0;

}
```

THRUST: PERFORMANCE





* Thrust 4.0, NVIDIA Tesla C2050 (Fermi)

* Core i7 950 @ 3.07GHz

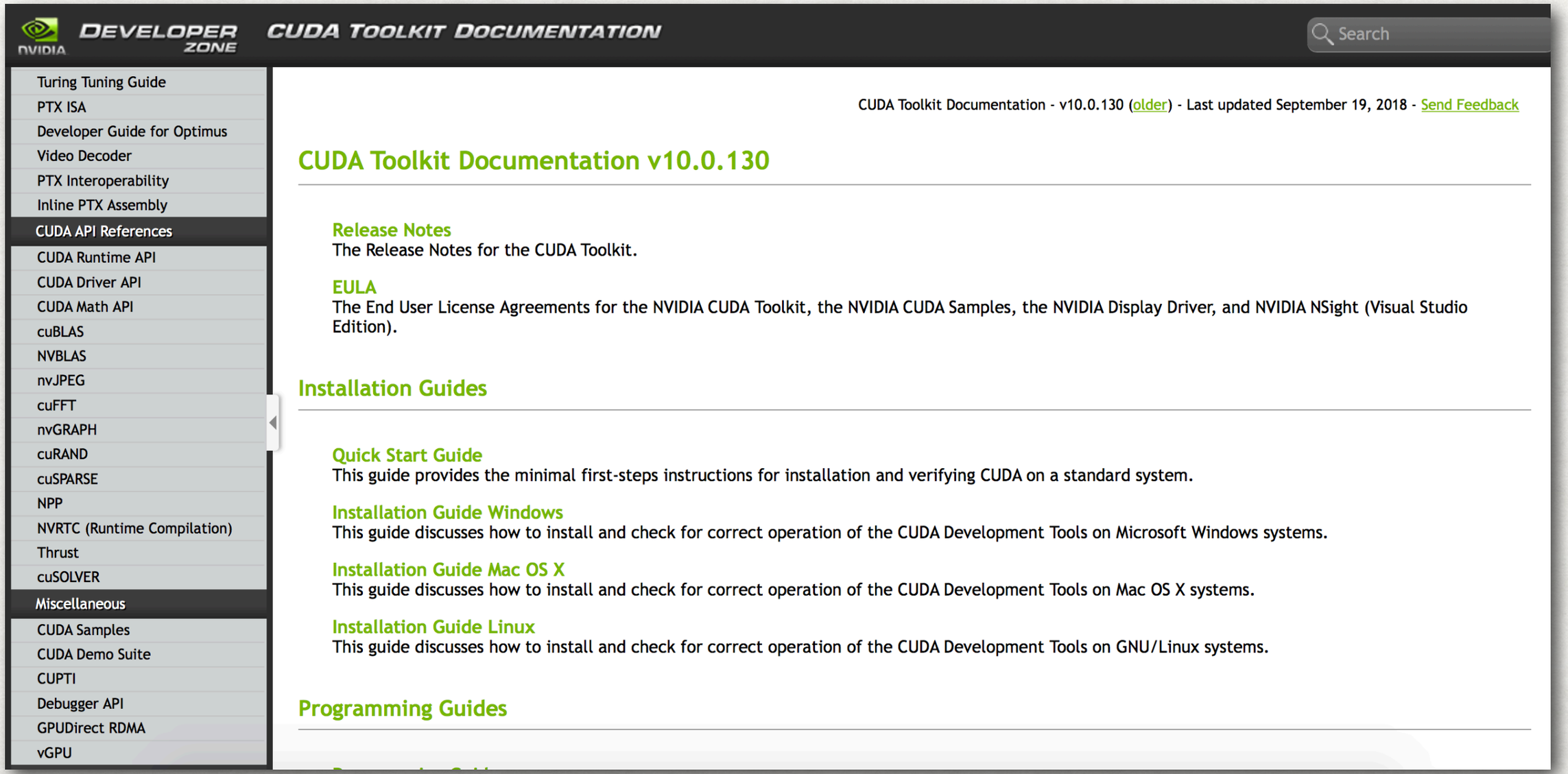
NVGRAPH

NVGRAPH

- Библиотека аналитики больших графов
- Поддерживает 3 широко-используемых графовых алгоритмов
- **Page Rank**
- **Single Source**
- **Single Source Widest Path**

ГДЕ ИСКАТЬ ИНФОРМАЦИЮ ПО БИБЛИОТЕКАМ?

- В документации CUDA Toolkit - https://docs.nvidia.com/cuda/index.html

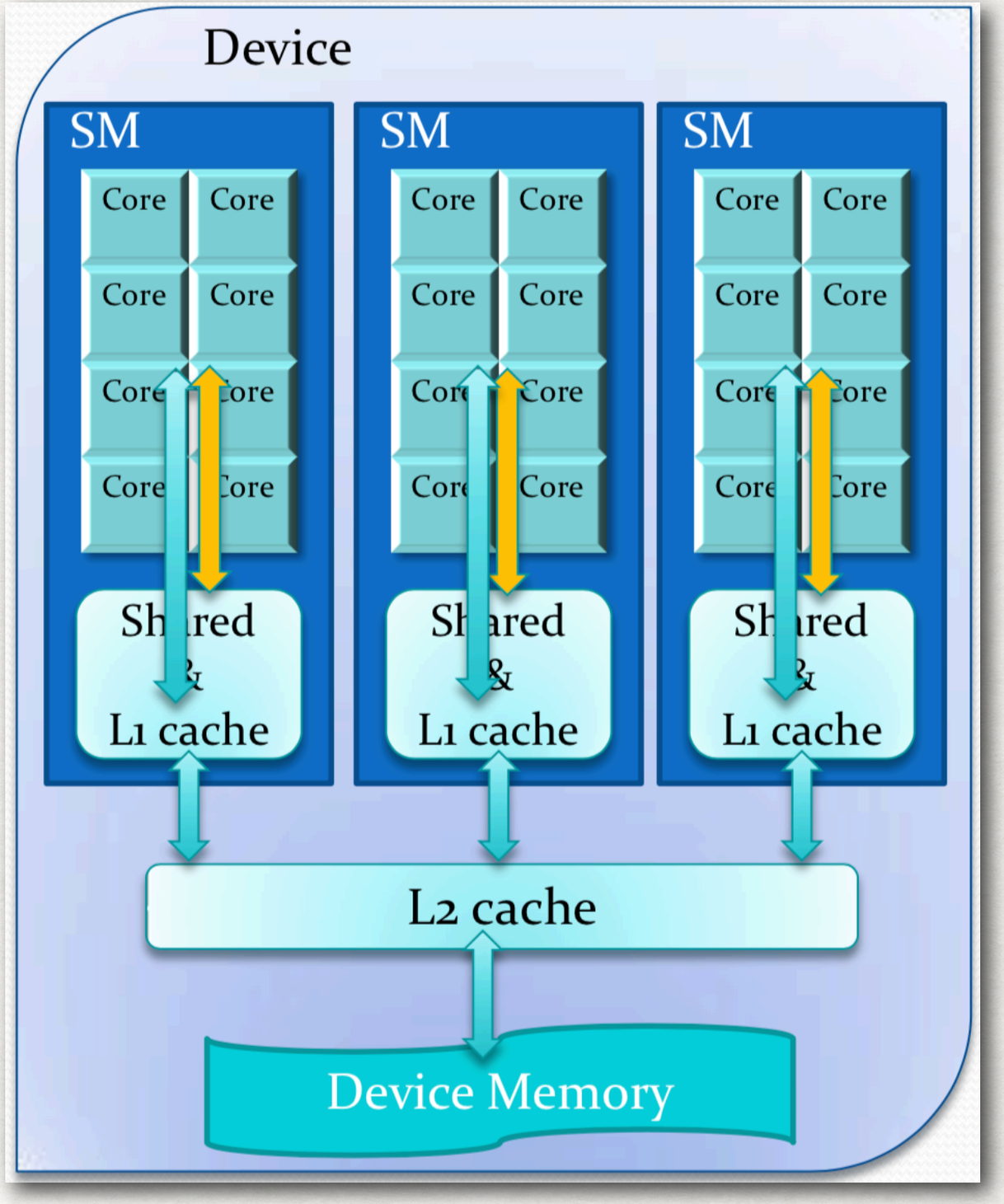


РАБОТА С РАЗДЕЛЯЕМОЙ ПАМЯТЬЮ. СИНХРОНИЗАЦИИ.

АФАНАСЬЕВ ИЛЬЯ
AFANASIEV_ILYA@ICLOUD.COM

РАЗДЕЛЯЕМАЯ ПАМЯТЬ

- Расположена в том же устройстве, что и кеш L1
- Совместно используется (разделяется) всеми нитями виртуального блока
- Если на мультипроцессоре работает несколько блоков - общая память делится между ними поровну
- У каждого блока своё ограниченное адресное пространство общей памяти (64 KB)
- Конфигурации:
  - 16KB общая память, 48KB L1
  - 48KB общая память, 16KB L1 - по умолчанию



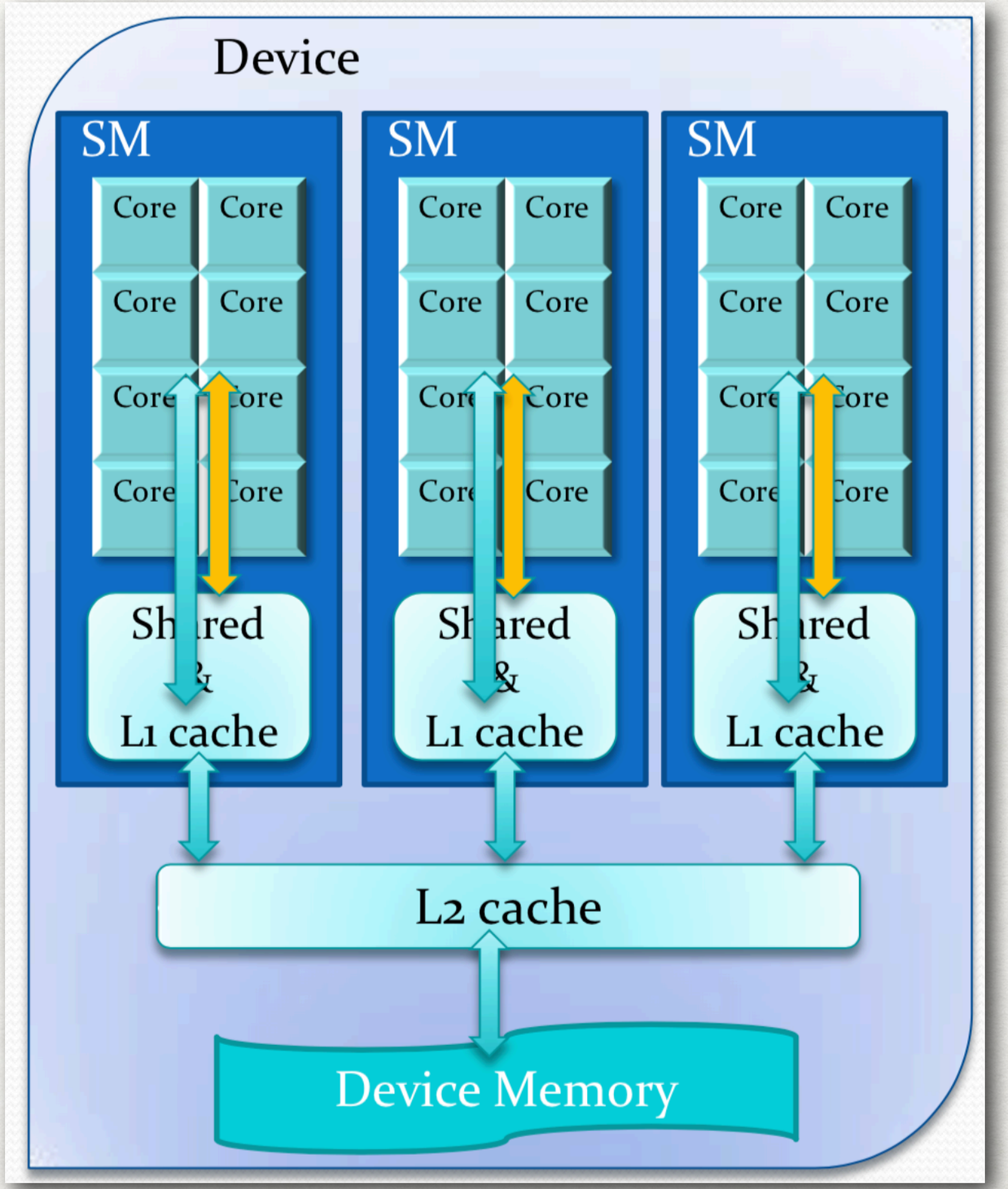
ЗАЧЕМ ИСПОЛЬЗОВАТЬ РАЗДЕЛЯЕМУЮ ПАМЯТЬ?

- Обмены данными между ядрами (рассмотрим для начала)
- Быстрый программный кэш

ОБМЕНЫ ДАННЫМИ ЧЕРЕЗ РАЗДЕЛЯЕМАЯ ПАМЯТЬ

- Через глобальную память возможны обмены данными между любыми двумя ядрами GPU (даже с **различных мультипроцессоров**)

- Через разделяемую память возможно обмены между ядрами **одного мультипроцессора**



ВЫДЕЛЕНИЕ РАЗДЕЛЯЕМОЙ ПАМЯТИ

(СТАТИЧЕСКОЕ)

- 2 способа
    - статически
    - динамически
- Статически: В ядре (kernel) объявляем статический массив или переменную с атрибутом __shared__
- Пример:

```
#define SIZE 1024
__global__ void kernel()
   {
    __shared__ int array[SIZE];  // можно выделять как массив
    __shared__ float varSharedMem;  // так и переменную
}
```

ВЫДЕЛЕНИЕ РАЗДЕЛЯЕМОЙ ПАМЯТИ

(ДИНАМИЧЕСКОЕ)

Динамически:

- В GPU коде объявляем указатель для доступа к общей памяти:

```
__global__ void kernel()
{
    extern __shared__ int array[];
}
```

- В третьем параметре конфигурации запуска указываем сколько общей памяти нужно выделить **каждому(!)** блоку

```
kernel<<<gridDim, blockDim, SIZE >>>(params)
```

- Все переменные extern __shared__ type var[] указывают на начало динамической общей памяти, выделенной блоку

ОСОБЕННОСТИ ИСПОЛЬЗОВАНИЯ

- Переменные с атрибутом __shared__ с точки зрения программирования:
  - Индивидуальны для каждого блока и привязаны к его личному пространству общей памяти (каждый блок нитей видит «своё» значение)
  - Существуют только на время жизни блока
  - Не доступны с хоста или из других блоков
  - Не могут быть проинициализированы при объявлении

ПРОБЛЕМА СИНХРОНИЗАЦИИ

(ОБМЕНЫ ДАННЫМИ ЧЕРЕЗ РАЗДЕЛЯЕМУЮ ПАМЯТЬ)

Рассмотрим пример ядра, запускаемого на одномерном линейном гриде:

```
__global__ void kernel(int *global_data)
{
    __shared__ int shmem[BLOCK_SIZE];
    shmem[threadIdx.x] = global_data(threadIdx.x);
    int a = shmem[(threadIdx.x + 1 )% BLOCK_SIZE];
}
```

- Каждая нить:
    - Записывает элемент global_data от своего индекса в соответствующую ей ячейку массива разделяемой памяти
    - Читает из массива элемент, записанный соседней нитью
- Чем этот код лучше в сравнении с вариантом global_data[(threadIdx.x + 1)% BLOCK_SIZE] ?

ПРОБЛЕМА СИНХРОНИЗАЦИИ

(ОБМЕНЫ ДАННЫМИ ЧЕРЕЗ РАЗДЕЛЯЕМУЮ ПАМЯТЬ)

- Варпы одного блока выполняются в непредсказуемом порядке
- Может получиться, что нить ещё не записала элемент, соседняя уже пытается его считать!
- read-after-write, write-after-read, write-after-write конфликты
- разделяемая память «делится» между нитями одного блока, а значит необходима синхронизация внутри нитей одного и того же блока

СИНХРОНИЗАЦИЯ НИТЕЙ БЛОКА

- Для явной синхронизации внутри блока предусмотрена встроенная функция `void __syncthreads();`
- При вызове этой функции нить блокирует ся до момента, когда:
  - все нити в блоке достигнут данную точку
  - результаты всех инициированных к данному моменту операций с глобальной\общей памятью, станут видны всем нитям блока
- `__syncthreads()` можно вызвать в ветвях условного оператора только если результат его условия одинаков во всех нитях блока, иначе выполнение может зависнуть или стать непредсказуемым

СИНХРОНИЗАЦИЯ НИТЕЙ БЛОКА

(КОРРЕКТНЫЙ ПРИМЕР)

```
__global__ void kernel(int *global_data)
{
    __shared__ int shmem[BLOCK_SIZE];
    shmem[threadIdx.x] = global_data(threadIdx.x);
    __syncthreads();
    int a = shmem[(threadIdx.x + 1 )% BLOCK_SIZE];
}
```

- Каждая нить:
    1. Записывает данные от своего индекса в соответствующую ей ячейку массива
    2. Ожидает завершения операций в других нитях
    3. Читает из массива элемент, записанный соседней нитью

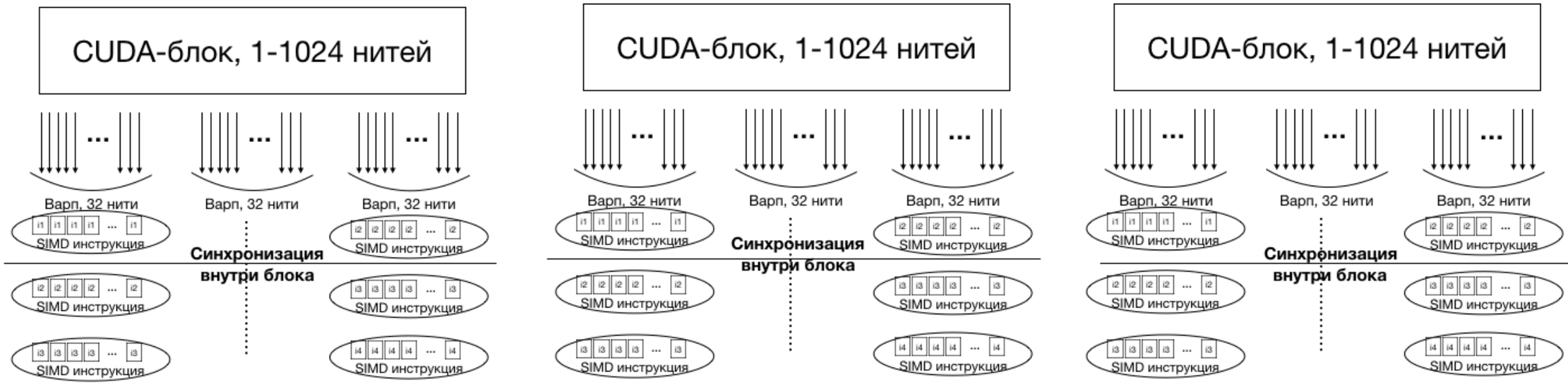
2.

СИНХРОНИЗАЦИЯ ВСЕХ CUDA-НИТЕЙ?

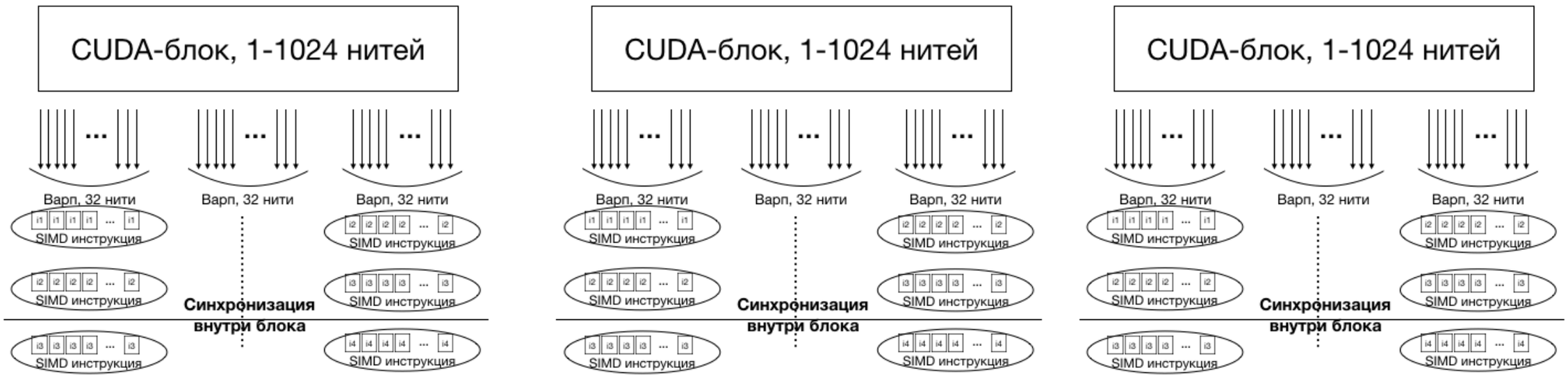
- Итак, конструкция __syncthreads() позволяет синхронизировать нити **внутри одного блока**
- Возможно ли синхронизировать нити различных блоков (или все нити)?
- Порядок вычисления блоков на мультипроцессорах не детерминирован
- Синхронизация нитей возможна только после завершения ядра и запуска нового (!)

СИНХРОНИЗАЦИЯ НИТЕЙ

ядро 1



ядро 2



СТРАТЕГИЯ ИСПОЛЬЗОВАНИЯ РАЗДЕЛЯЕМОЙ ПАМЯТИ (1)

- Общая память по смыслу является кешем, управляемым пользователем
- Имеет низкую латентность - расположена на том же оборудовании, что и кеш L1, скорость загрузки сопоставима с регистрами
- Приложение явно выделяет и использует общую память
- Пользовать сам выбирает что, как и когда в ней хранить
- Шаблон доступа может быть произвольным, в отличие от L1

СТРАТЕГИЯ ИСПОЛЬЗОВАНИЯ РАЗДЕЛЯЕМОЙ ПАМЯТИ (2)

- Даже если аппаратный кеш L1 «справляется» с запросами (L1 load hit ~ 100%) использование общей памяти позволяет его дополнительно разгрузить и полностью использовать аппаратуру
- Иначе 16KB (или даже 48KB, если забыли выставить режим) быстрой памяти простаивают

СТРАТЕГИЯ ИСПОЛЬЗОВАНИЯ РАЗДЕЛЯЕМОЙ ПАМЯТИ (3)

- Нити блока коллективно

1. Загружают данные из глобальной памяти в общую (каждая нить делает часть этой загрузки)

2. Синхронизуются (чтобы никакая нить не начинала чтение данных, загружаемых другой нитью, до завершения их загрузки)

3. Используют загруженные данные для вычисления результаты (если нити что-то пишут в общую память, то также может потребоваться синхронизация)

4. Записывают результаты обратно в глобальную память

ПРИМЕР ИСПОЛЬЗОВАНИЯ ОБЩЕЙ ПАМЯТИ

(ЯДРО)

```
__global__ void kernel(int sizeOfArray1, int sizeOfArray2, int *devPtr, int *res)
{
    extern __shared__ int dynamicMem[]; // указатель на динамическую общую память
     __shared__ int staticMem[1024]; // статический массив в общей памяти
    __shared__ int var; // переменная в общей памяти
    int *array1 = dynamicMem;//адрес первого массива в динамической общей памяти
    int *array2 = array1 + sizeOfArray1; //адрес второго массива в динамической
    общей памяти
    staticMem[threadIdx.x] = devPtr[threadIdx.y];// загрузка данных в общую память
    __syncthreads(); // подождать пока все нити завершат запись
    array2[threadIdx.x] = 2 * staticMem[(threadIdx.x - 10) % blockDim.x]; //
    обратится к элементу, //записанному другой нитью
    __syncthreads(); // подождать пока все нити завершат запись
    res[threadIdx.x] = array2[threadIdx.x]; // записать результаты в глобальную
    память
}
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ ОБЩЕЙ ПАМЯТИ

(ЗАПУСК ЯДРА НА ХОСТЕ)

```
int *devPtr;
cudaMalloc(&devPtr, 1024*sizeof(int));
kernel<<<3,1024,1024*sizeof(int)>>>(512,512, devPtr);
```

- Запускаются три блока по 1024 нити
- Третий параметр - размер общей памяти динамически выделяемой каждому блоку в байтах
- Таким образом, каждому блоку динамически выделяется по 4KB общей памяти
- Если суммарный (статически + динамически) запрашиваемый объём общей памяти превосходит доступный (16KB или 48KB), произойдёт ошибка запуска ядра

ШАБЛОНЫ ЗАПИСИ В РАЗДЕЛЯЕМУЮ ПАМЯТЬ

Что произойдет, если несколько нитей варпа пытаются записать по одному и тому же адресу ?

- Запись будет выполнена только одной нитью
- Какой именно - неизвестно
- Если по одному и тому же адресу пишут нити из разных варпов, то результат непредсказуем, т.к. непредсказуем порядок варпов и неизвестно какой варп будет писать последним

БАНКИ ОБЩЕЙ ПАМЯТИ

Общая память разделена на независимые модули одинакового размера, называемые «банками». В рассматриваемой архитектуре 32 банка.

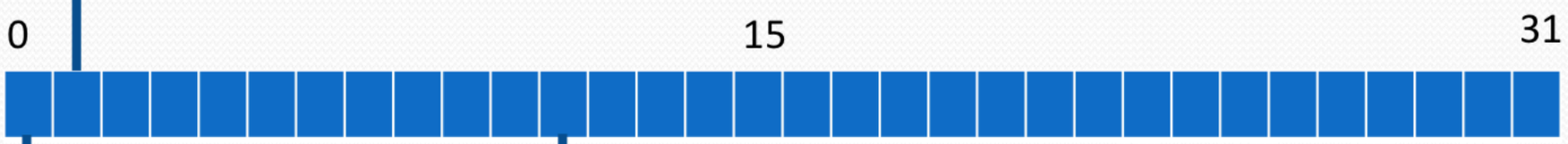
- Последовательные 4-байтные данные располагаются в разных банках
- Номер банка для слова по адресу addr: (addr / 4) % 32
- Каждый банк может выдать за 2 такта одно 32-битное слово (4 байта)

БАНКИ ОБЩЕЙ ПАМЯТИ

Банк 1

4	
132	
260	
772	
1028	
…	

Банк 0	Банк 1	Банк 2	Банк 3	….
0	4	8	12	…
128	132	136	140	….
256	260	264	268	…
…	…	…	…	…



Банк 0

0	
128	
256	
768	
1024	
…	

Банк 11

44	
172	
300	
812	
1068	
…	

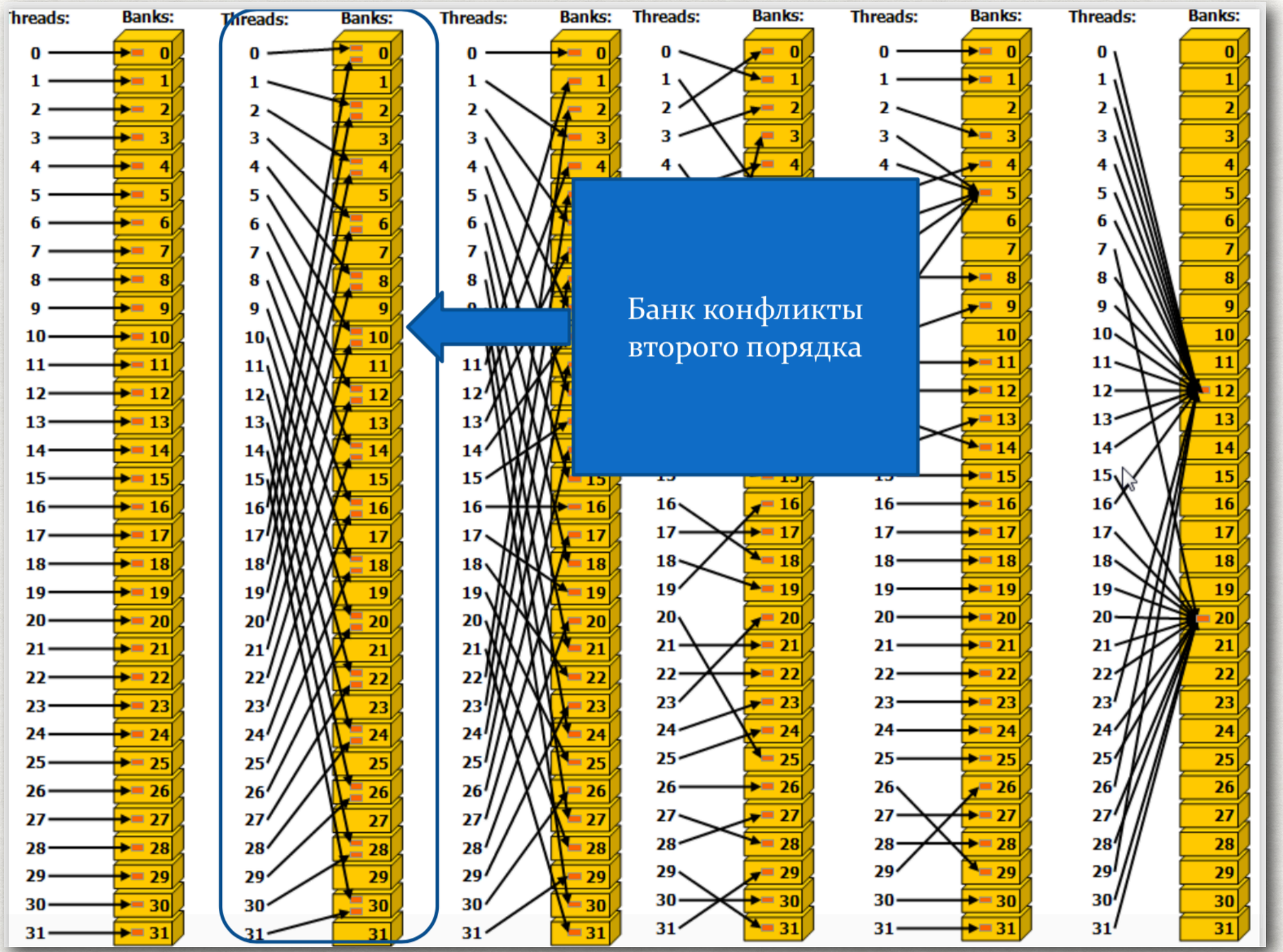
ОБРАЩЕНИЯ В ОБЩУЮ ПАМЯТЬ И БАНКИ

- Обращение выполняется одновременно всеми нитями варпа(SIMT)
- Банки работаю параллельно
- Если варпу нитей нужно получить 32 4-байтных слова, расположенных в разных банках, то такой запрос будет выполнен одновременно всеми банками - каждый банк выдаст соответствующее слово
- Пропускная способность разделяемой памяти = 32 х пропускная способность банка
- Если часть нитей (или все) обращаются к одному и тому же 4-х байтному слову, то нужное слово будет считано из банка и роздано соответствующим нитям (broadcast) без накладных расходов

БАНК КОНФЛИКТЫ

- Если хотя бы два нужных варпу слова расположены в одном банке, то такая ситуация называется «банк конфликтом» и обращение в глобальную память будет «сериализованно»:
    - Такое обращение аппаратно разбивается на серию обращений, не содержащих банк конфликтов
    - Если число обращений, на которое разбит исходный запрос, равно n, то такая ситуация называется банк-конфликтом порядка n
    - Пропускная способность при этом падает в n раз

БАНК КОНФЛИКТЫ



ПРИМЕРЫ БАНК КОНФЛИКТОВ

```
extern __shared__ float char[];
float data = shared[BaseIndex + s * threadIx.x]; // конфликты зависят от s
```

- Нити `threadIx.x` и (`threadIx.x + n`) обращаются к элементам из одного и того же банка когда s*n делится на 32 (число банков).

- S =1 :
  `shared[BaseIndex + threadIx.x] // нет конфликта`

- S=2:
  `shared[BaseIndex + 2*threadIx.x] // конликта 2-го порядка`

- Например, между нитями `threadIx.x=0 и (threadIx.x = 16)` - попадают в один банк!

ПРИМЕРЫ БАНК КОНФЛИКТОВ

(ЧАСТАЯ ПРОБЛЕМА)

- Пусть в общей памяти выделена плоская плотная матрица шириной, кратной 32, и соседние нити варпа обращаются к соседним элементам столбца

```
__shared__ int matrix[32][32] matrix[thredIdx.x][4] = 0;
```

- Решение: набивка

```
__shared__ int matrix[32][32 + 1] matrix[thredIdx.x][4] = 0; //нет конфликта
```

ВОПРОСЫ?

ИНТЕГРИРОВАННЫЕ СРЕДЫ РАЗРАБОТКИ С ПОДДЕРЖКОЙ CUDA, ИНСТРУМЕНТЫ ДЛЯ ОТЛАДКИ И ПРОФИЛИРОВАНИЯ. ОПТИМИЗАЦИЯ GPU ПРОГРАММ.

АФАНАСЬЕВ ИЛЬЯ
AFANASIEV_ILYA@ICLOUD.COM

ОСНОВНЫЕ СРЕДСТВА ПРОФИЛИРОВКИ CUDA-ПРИЛОЖЕНИЙ

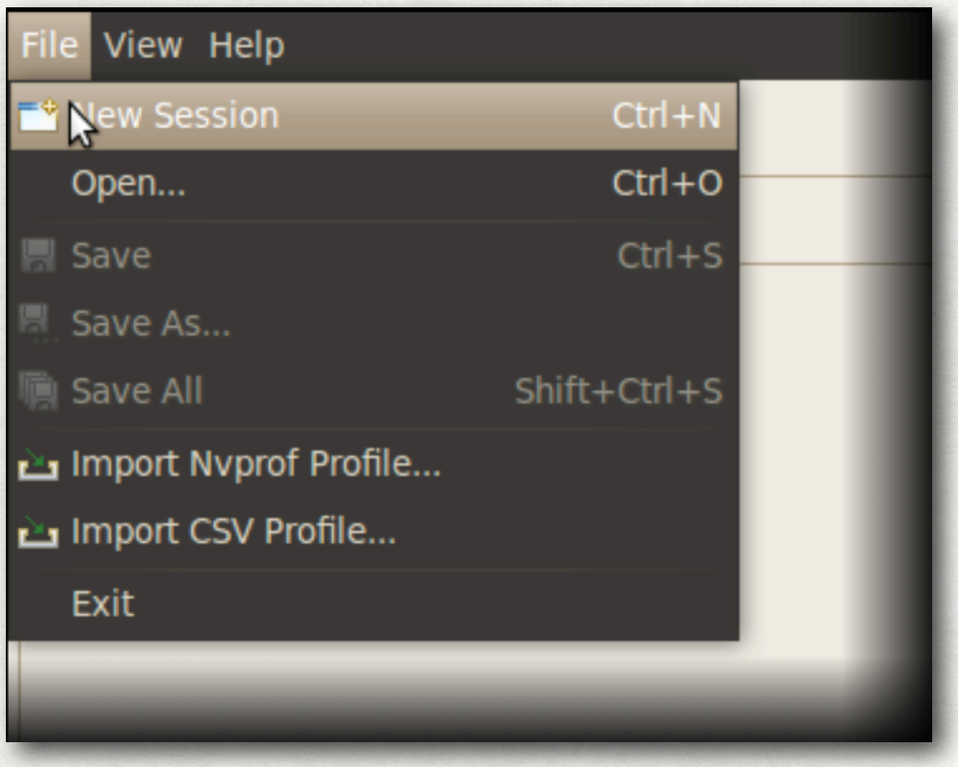
- NVIDIA® Visual Profiler
  – Standalone (nvvp)
  – Integrated into NVIDIA® NsightTM Eclipse Edition (nsight)

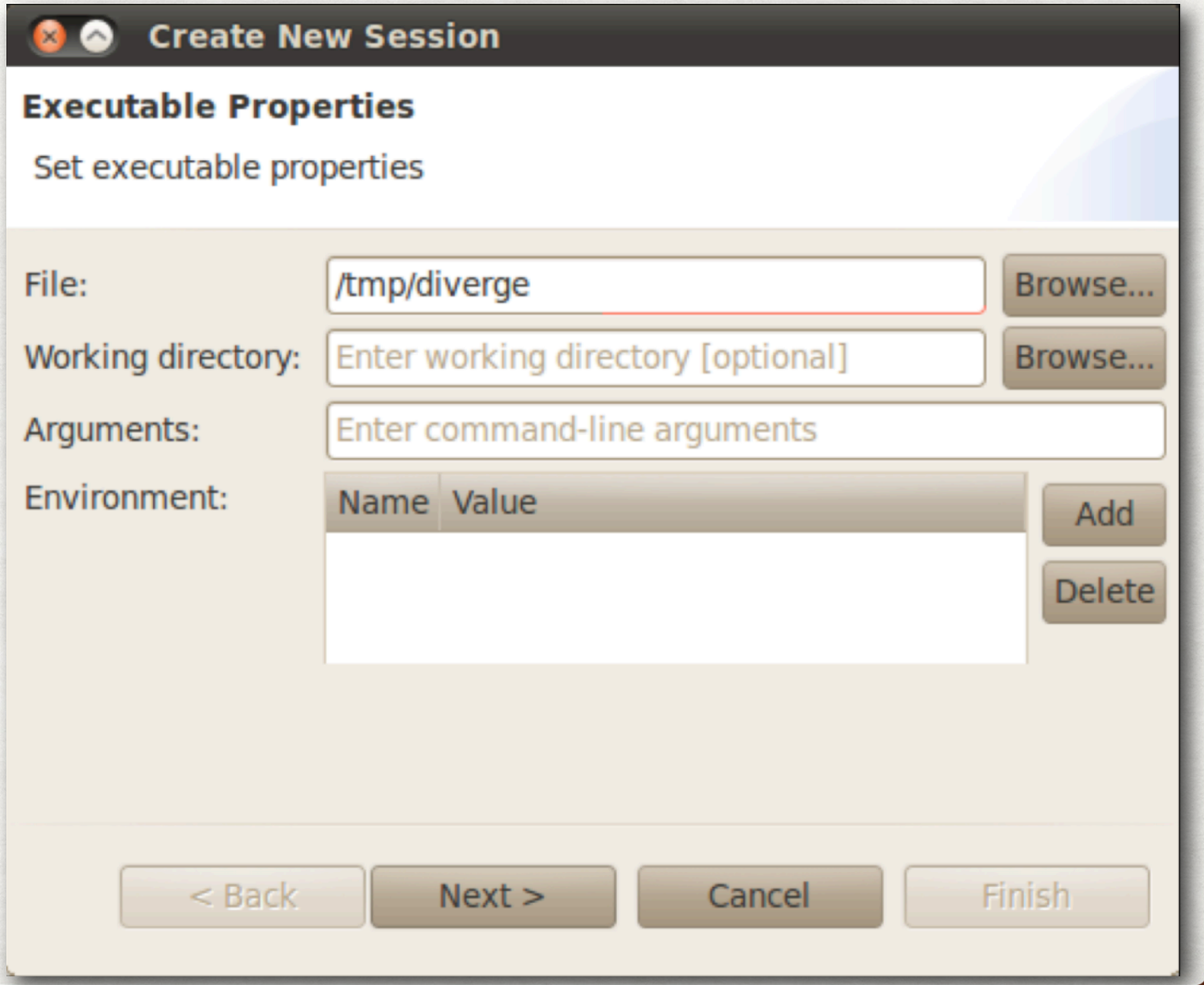
- nvprof
  – Command-line profiler

NVIDIA® VISUAL PROFILER

НАЧАЛО РАБОТЫ

- Начнем рассмотрение с обзора функционала NVIDIA Visual Profiler
- Возможен как запуск новой серии (на системах с GPU), так и импорт данных профилировки, собранных удаленно (при помощи nvprof)

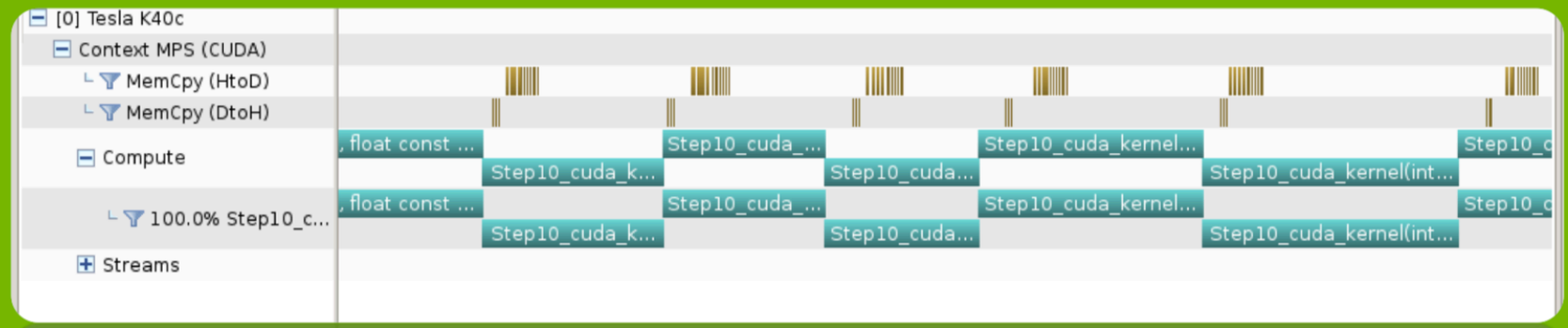




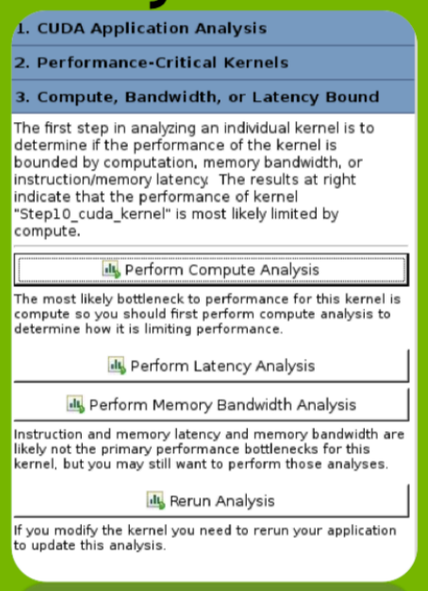
NVIDIA® VISUAL PROFILER

ОСНОВНЫЕ ВОЗМОЖНОСТИ

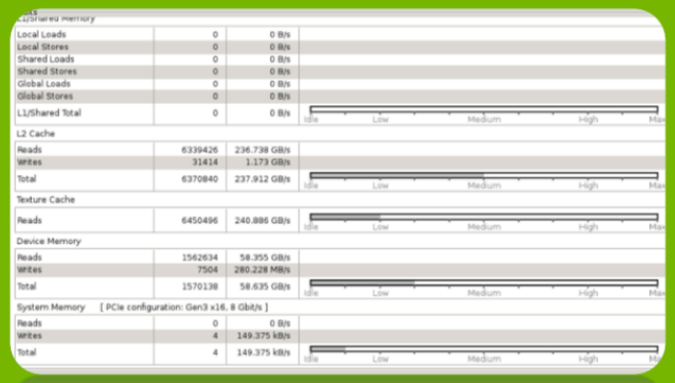
Timeline

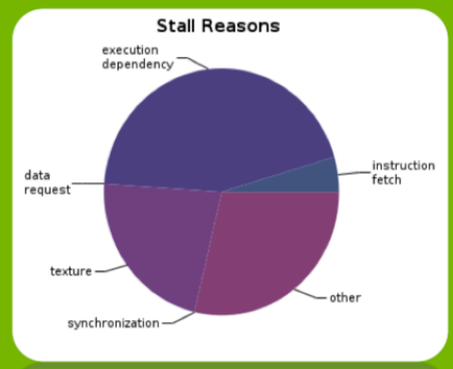


Guided System



Analysis

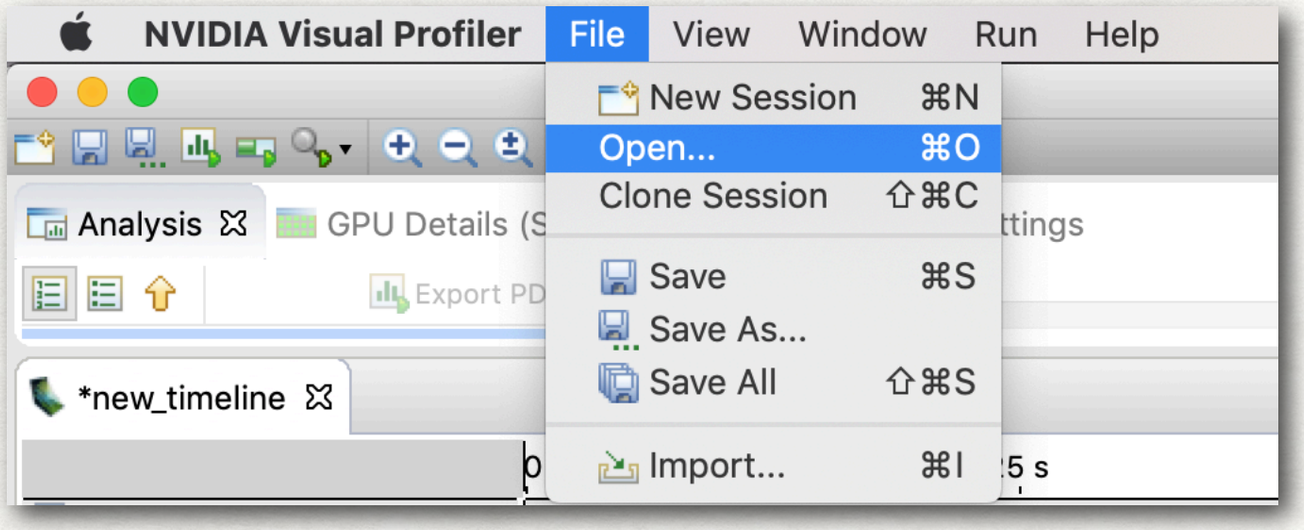


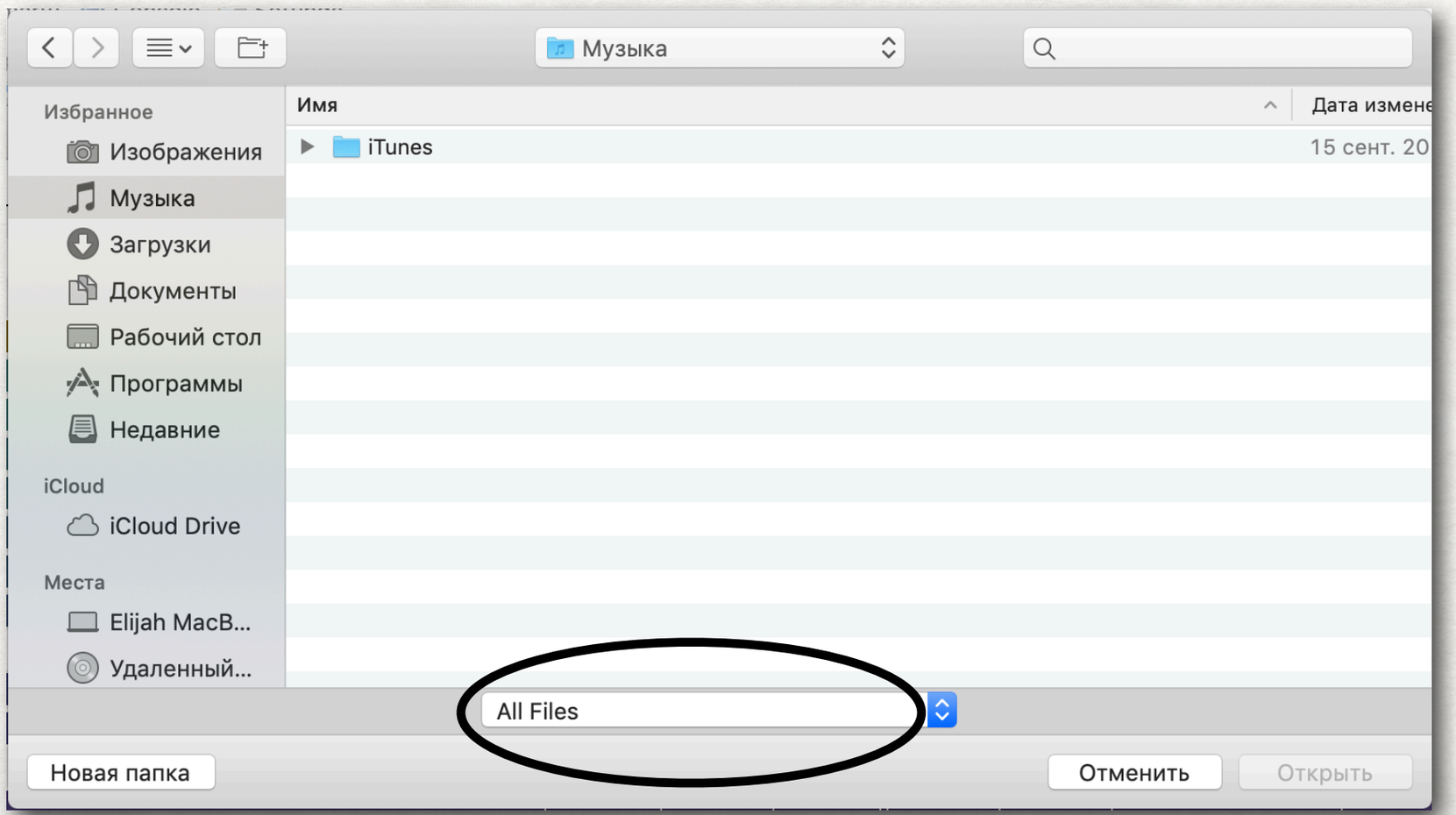


ИМПОРТ УДАЛЕННОЙ СЕССИИ

- Для оптимизации GPU-приложений, работающих на кластере, удобнее всего использовать удаленный сбор метрик, с последующим их анализом про помощи Visual Profiler
- Для этого необходимо иметь установленный локально CUDA Toolkit
- Запустить nvvp (например, выполнив /usr/local/cuda/bin/nvvp в командной строке)
- На удаленном кластере выполнить сбор данных профилировки: nvprof -o timeline_short ./app_name app_param_1 app_param_2
- Загрузить полученный файл timeline_short на локальный диск

ИМПОРТ УДАЛЕННОЙ СЕССИИ

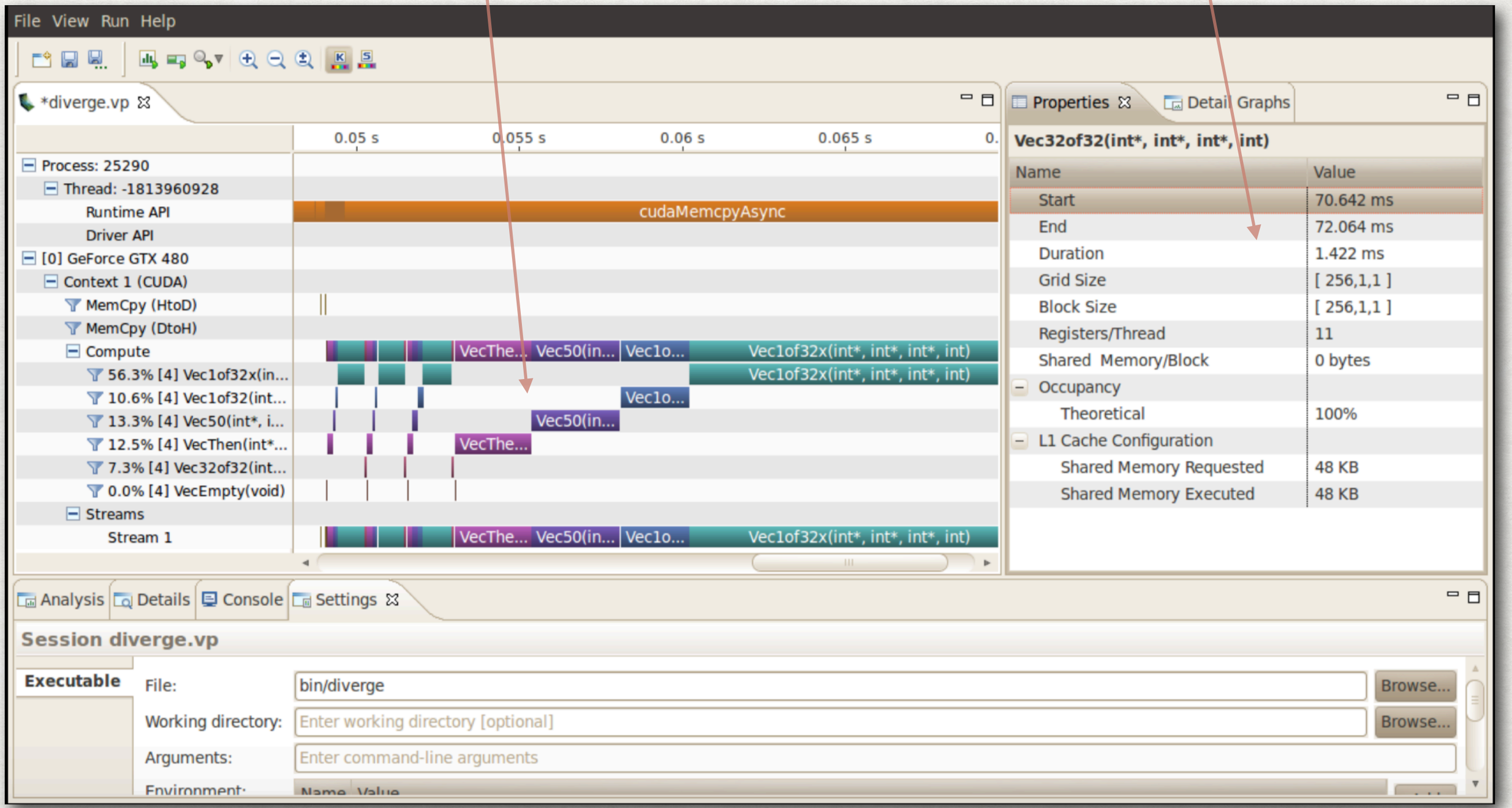




NVIDIA® VISUAL PROFILER

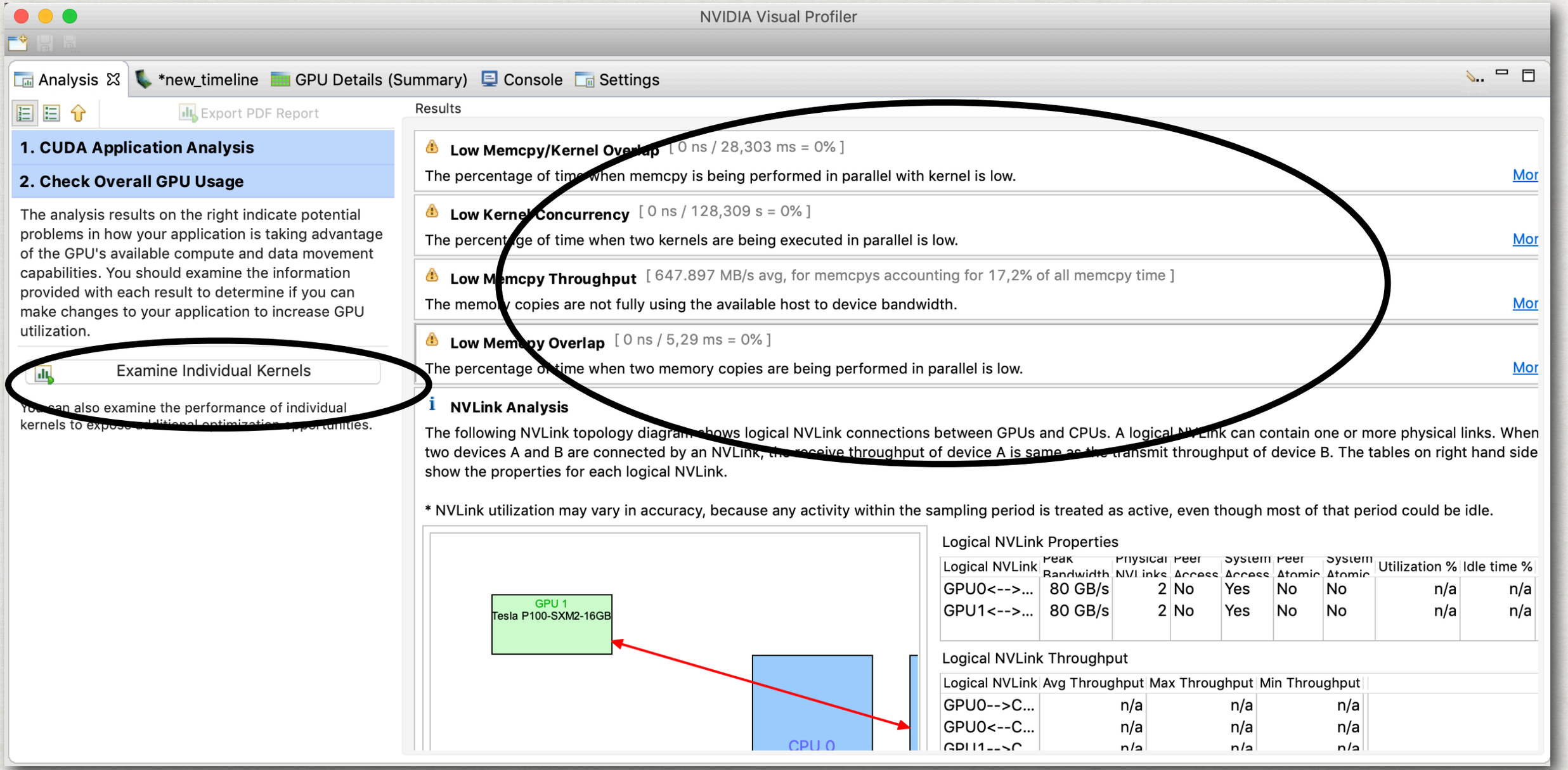
timeline активностей GPU

краткая информация о ядре



ОКНО АНАЛИЗА

- Основные пробелы взаимодействия CPU-GPU
- Оценка приоритета оптимизации ядер (examine individual kernels)
- Анализ использования NVLINK



ЧТО ПОЗВОЛЯЕТ ПОНЯТЬ TIMELINE?

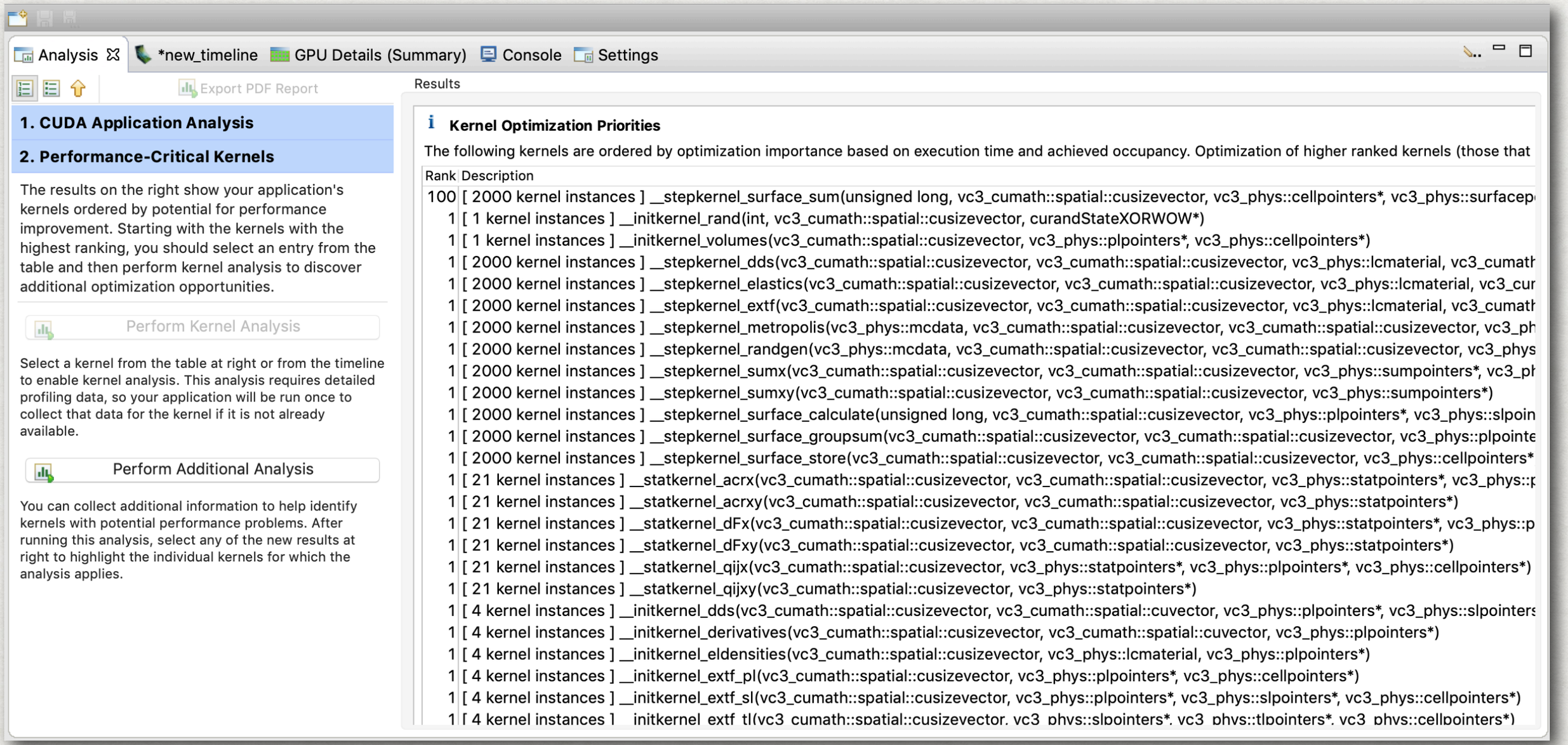
- Соотношение CPU и GPU активностей
- Соотношение времён передачи данных, работы ядер, работы с CUDA-API
- Частоту вызова ядер
- Наиболее длительные ядра
- и т.д.
- для сбора более подробной информации о работе отдельных ядер необходимо использовать GPU-метрики (события)

С ЧЕГО НАЧИНАТЬ ПРОЦЕСС ОПТИМИЗАЦИИ?

- Пусть у нас есть CPU+GPU приложение, и мы оптимизирует только его GPU часть
- Изначально нужно понять, что необходимо оптимизировать: CUDA-ядра или пересылки данных между CPU и GPU
- Оценить суммарное время выполнения пересылок и вычислений можно из timeline
- Время пересылок >> времени вычислений - оптимизируем пересылки данных (параллельные пересылки и вычисления, уменьшение объема пересылаемых данных, генерация данных на GPU, и т.д.)
- Время пересылок << времени вычислений - оптимизируем CUDA-ядра (рассмотрим далее)

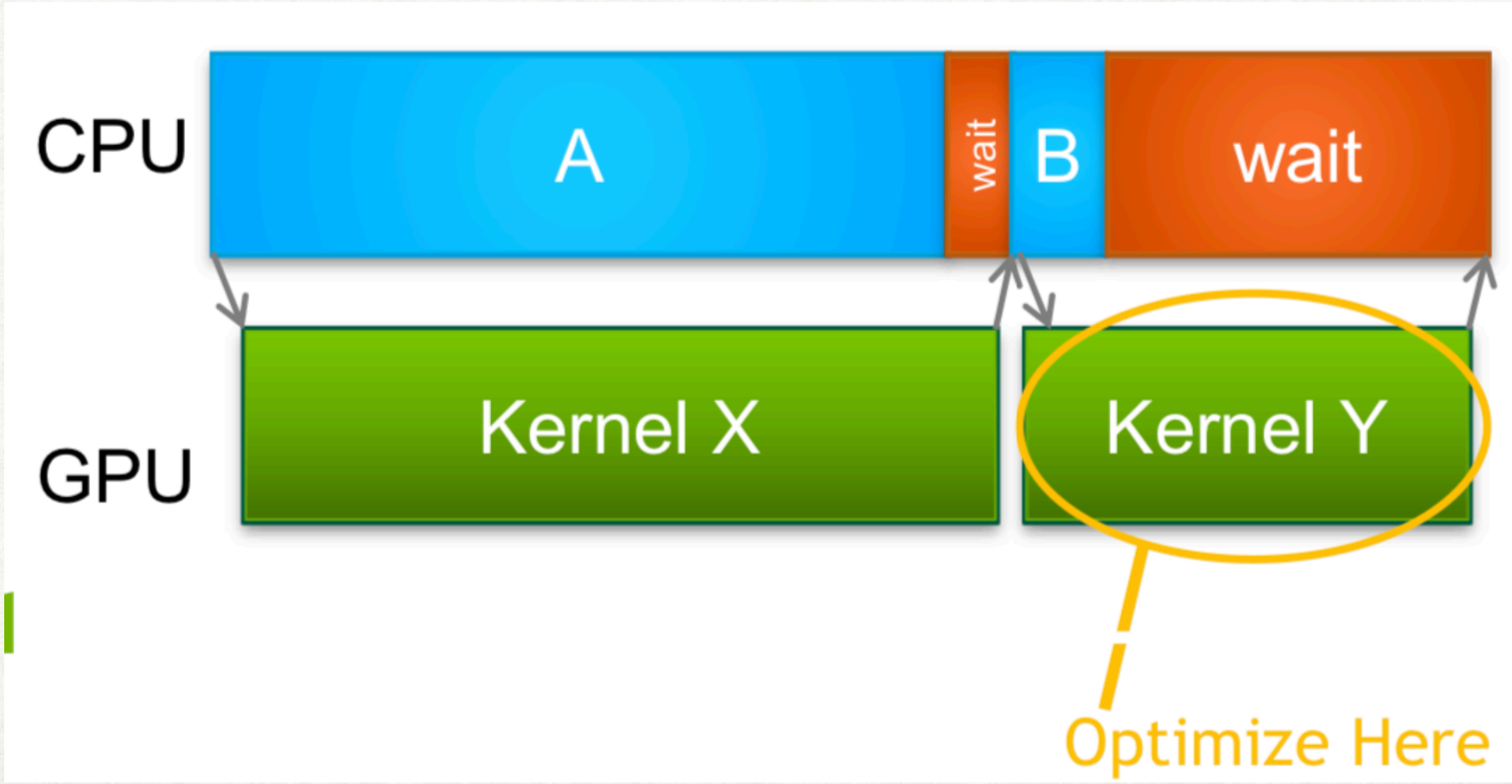
ПУСТЬ ПЕРЕСЫЛКИ ДАННЫХ - НЕ УЗКОЕ МЕСТО

- Меню examine individual kernels позволяет оценить приоритет оптимизации ядер



ВСЕГДА ЛИ НЕОБХОДИМО ОПТИМИЗИРОВАТЬ САМОЕ ДОЛГОЕ ЯДРО?

- Да, если в программе нет CPU-части
- Иначе может возникнуть проблема:



NVPROF TRACE

- nvprof так же позволяет собирать и анализировать информацию об активностях GPU
- nvprof --print-gpu-trace ./app_name
- nvprof --print-summary ./app_name
- дополнительные возможности: –print-openacc-trace,–print-openacc-summary

--PRINT-GPU-TRACE

```
==48511== Profiling application: ./a.out 1024 2
==48511== Profiling result:
   Start  Duration            Grid Size      Block Size     Regs*    SSMem*    DSMem*      Size  Throughput
SrcMemType  DstMemType           Device   Context    Stream  Name
408.67ms  125.03us                    -               -         -         -         -  4.0000MB  31.242GB/s      Pinned      Device
Tesla P100-SXM2         1         7  [CUDA memcpy HtoD]
408.82ms  129.48us                    -               -         -         -         -  4.0000MB  30.168GB/s      Pinned      Device
Tesla P100-SXM2         1         7  [CUDA memcpy HtoD]
409.01ms  5.8812ms          (128 128 1)       (8 8 1)        32        0B        0B         -           -           -           -  Tesla
P100-SXM2         1         7  gpu_matrix_mult_right_order(float*, float*, float*, int, int, int) [218]
414.90ms  5.8836ms          (128 128 1)       (8 8 1)        32        0B        0B         -           -           -           -  Tesla
P100-SXM2         1         7  gpu_matrix_mult_right_order(float*, float*, float*, int, int, int) [227]
420.81ms  124.68us                    -               -         -         -         -  4.0000MB  31.330GB/s      Device      Pinned
Tesla P100-SXM2         1         7  [CUDA memcpy DtoH]
```

Regs: Number of registers used per CUDA thread. This number includes registers used internally by the CUDA driver and/or tools and can be more than what the compiler shows.
SSMem: Static shared memory allocated per CUDA block.
DSMem: Dynamic shared memory allocated per CUDA block.
SrcMemType: The type of source memory accessed by memory operation/copy
DstMemType: The type of destination memory accessed by memory operation/copy

--PRINT-SUMMARY

```
==49616== Profiling application: ./a.out 1024 2
==49616== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   96.81%  11.756ms         2  5.8781ms  5.8705ms  5.8858ms
gpu_matrix_mult_right_order(float*, float*, float*, int, int, int)
                   2.15%  260.60us         2  130.30us  129.32us  131.28us  [CUDA memcpy HtoD]
                   1.05%  126.96us         1  126.96us  126.96us  126.96us  [CUDA memcpy DtoH]
      API calls:   88.87%  139.75ms         4  34.939ms  471.82us  138.32ms  cudaMallocHost
                   7.46%  11.736ms         1  11.736ms  11.736ms  11.736ms  cudaThreadSynchronize
                   1.33%  2.0851ms         2  1.0425ms  1.0405ms  1.0446ms  cuDeviceTotalMem
                   0.72%  1.1245ms         4  281.12us  170.12us  412.50us  cudaFreeHost
                   0.60%  947.41us       188  5.0390us     244ns  191.76us  cuDeviceGetAttribute
                   0.34%  536.54us         3  178.85us  146.67us  241.41us  cudaMalloc
                   0.30%  467.72us         3  155.91us  143.36us  179.70us  cudaMemcpy
                   0.26%  405.14us         3  135.05us  126.09us  151.78us  cudaFree
                   0.05%  79.751us         2  39.875us  39.515us  40.236us  cuDeviceGetName
                   0.04%  58.841us         2  29.420us  10.474us  48.367us  cudaLaunch
                   0.01%  19.157us         4  4.7890us  3.0180us  8.5140us  cudaEventRecord
                   0.01%  13.745us         2  6.8720us  1.5640us  12.181us  cudaEventCreate
                   0.01%  9.2350us         2  4.6170us  4.4160us  4.8190us  cudaEventElapsedTime
                   0.01%  8.9390us        12     744ns     256ns  5.7500us  cudaSetupArgument
                   0.01%  8.3160us         2  4.1580us  3.5310us  4.7850us  cudaEventSynchronize
                   0.00%  2.5350us         3     845ns     373ns  1.7110us  cuDeviceGetCount
                   0.00%  1.5940us         4     398ns     301ns     526ns  cuDeviceGet
                   0.00%  1.4910us         2     745ns     309ns  1.1820us  cudaConfigureCall
```

GPU METRICS/EVENTS

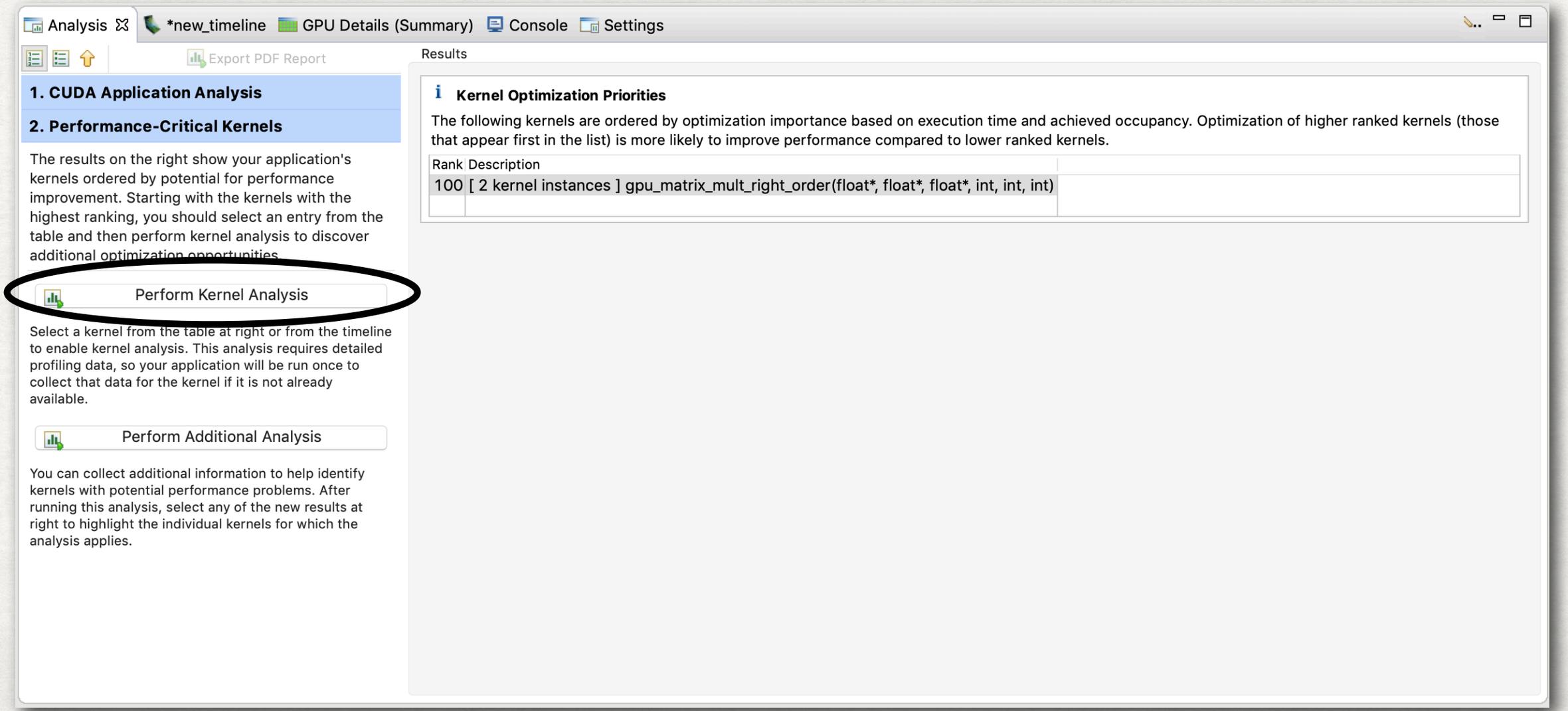
- nvprof поддерживает сбор различного рода аппаратных событий и метрик
- список всех доступных метрик можно получить при помощи: nvprof –query-metrics (с кратким описанием)
- пример метрик: l2_read_transactions,gld_throughput, и др.
- список метрик и их метрик доступны здесь: https://docs.nvidia.com/cuda/profiler-users-guide/index.html#metrics-reference

NVPROF METRICS & EVENTS

- Для сбора **всех** доступных на устройстве метрик и событий можно использовать ~/local_cuda/bin/nvprof --events all --metrics all ./prog_name
- Можно уточнить ядра и события для сбора: nvprof --kernels kernel_name --metrics issue_slot_utilization,int_fu_utilization,special_fu_utilization,half_precision_fu_utilization,single_precision_fu_utilization,double_precision_fu_utilization ./prog_name
- На основе собираемых событий можно проводить анализ в nvvp. Для этого необходимо:
  - ~/local_cuda/bin/nvprof --analysis-metrics -o full_timeline ./prog_name
  - скопировать файл со значениями метрик full_timeline и трассой профилировки на локальный диск
  - открыть данный файл в nvvp (open -> ...)

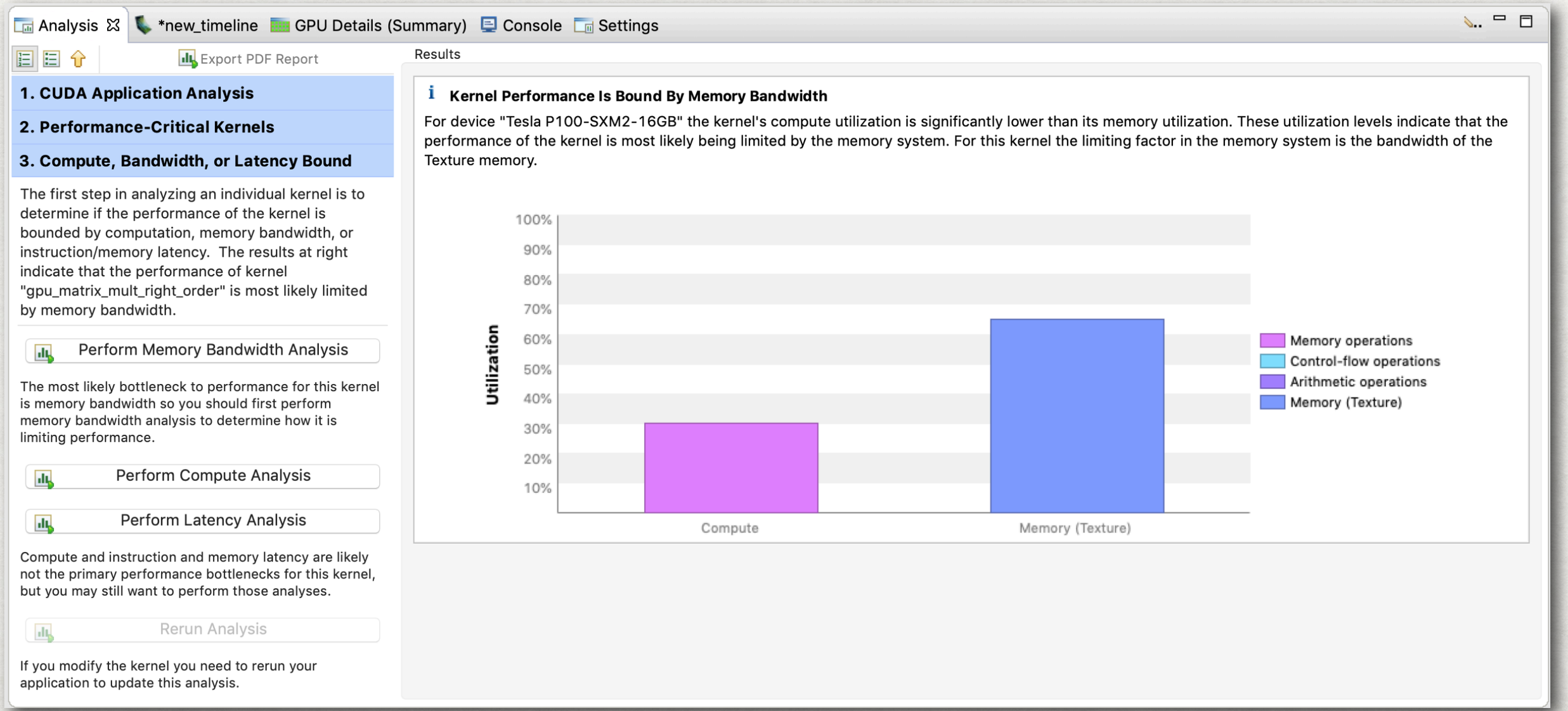
ОПТИМИЗАЦИЯ ЯДЕР

- Предположим мы можем выделить наиболее «тяжеловесное» ядро с высоким приоритетом оптимизации
- Меню «perform kernel analysis» (!)



KERNEL ANALYSIS

- Существуют memory-bound, latency-bound, compute-bound ядра
- Для каждого типа ядер существуют различные стратегии оптимизации



MEMORY-BOUND ЯДРА

- memory_utilisation > compute utilisation
- memory_utilisation = max(dram_utilization, shared_utilization, l2_utilization, tex_utilization)
- compute_resource_utlization = max(int_fu_utilization,special_fu_utilization, half_precision_fu_utilization,single_precision_fu_utilization,double_precision_fu_utilization)
- memory-bound ядра интенсивно используют пропускную способность одного (или нескольких) уровней иерархии памяти
- обычно high(dram_utilization) <=> high dram bandwidth

ОПТИМИЗАЦИЯ MEMORY-BOUND ЯДЕР

- Основные стратегии оптимизации memory-bound ядер
  - исправление шаблона доступа к данным (уменьшение числа транзакций, выравнивание)
  - локализация обращений к памяти - размещение данных на более «высоком» уровне иерархии памяти (кэширование, разделяемая память)
  - размещение данных на регистрах (переиспользование данных)
  - может помочь опция «perform memory bandwidth analysis» в NVVP

PERFORM MEMORY BANDWIDTH ANALYSIS

	Transactions	Bandwidth	Utilization
Shared Memory			
Shared Loads	0	0 B/s	
Shared Stores	0	0 B/s	
Shared Total	0	0 B/s	Idle Low Medium High Max
L2 Cache			
Reads	58054754	341.315 GB/s	
Writes	262224	1.542 GB/s	
Total	58316978	342.857 GB/s	Idle Low Medium High Max
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	1073741826	986.364 GB/s	
Global Stores	262144	1.541 GB/s	
Texture Reads	268435456	1,578.183 GB/s	
Unified Total	1342439426	2,566.088 GB/s	Idle Low Medium High Max
Device Memory			
Reads	3182423	18.71 GB/s	
Writes	190356	1.119 GB/s	
Total	3372779	19.829 GB/s	Idle Low Medium High Max

COMPUTE-BOUND ЯДРА

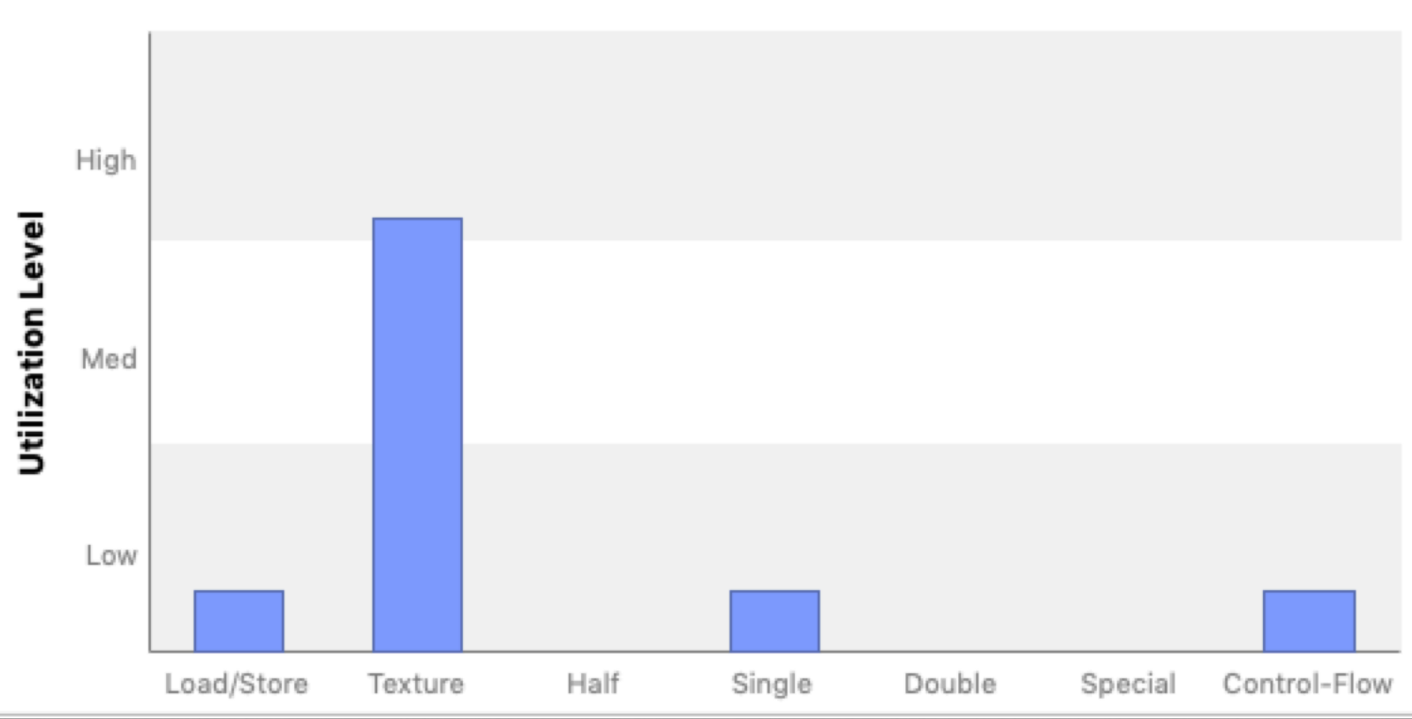
- compute utilisation > memory_utilisation
- compute-bound ядра интенсивно используют вычислительных ядра различных типов
- основные стратегии оптимизации compute-bound ядер:
    - развертка циклов
    - double -> float
    - и т.д.

COMPUTE-BOUND ЯДРА

Function Unit Utilization

Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is not limited by overuse of any function unit.

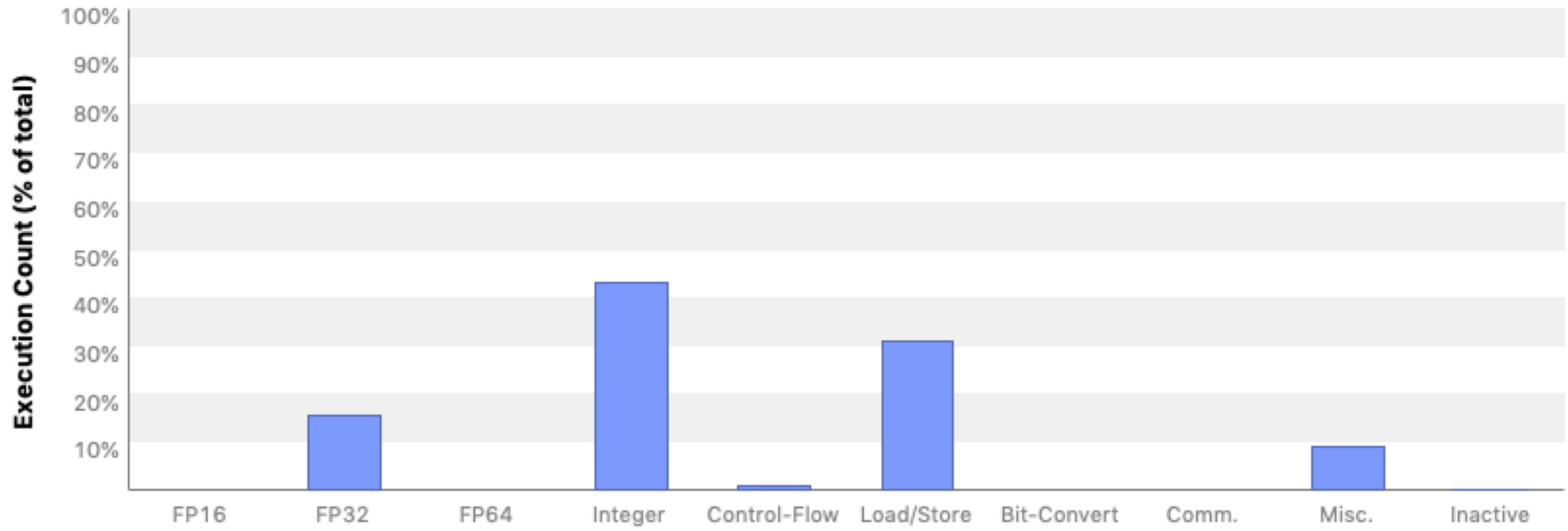
Load/Store - Load and store instructions for shared and constant memory.
Texture - Load and store instructions for local, global, and texture memory.
Half - Half-precision floating-point arithmetic instructions.
Single - Single-precision integer and floating-point arithmetic instructions.
Double - Double-precision floating-point arithmetic instructions.
Special - Special arithmetic instructions such as sin, cos, popc, etc.
Control-Flow - Direct and indirect branches, jumps, and calls.



COMPUTE-BOUND ЯДРА

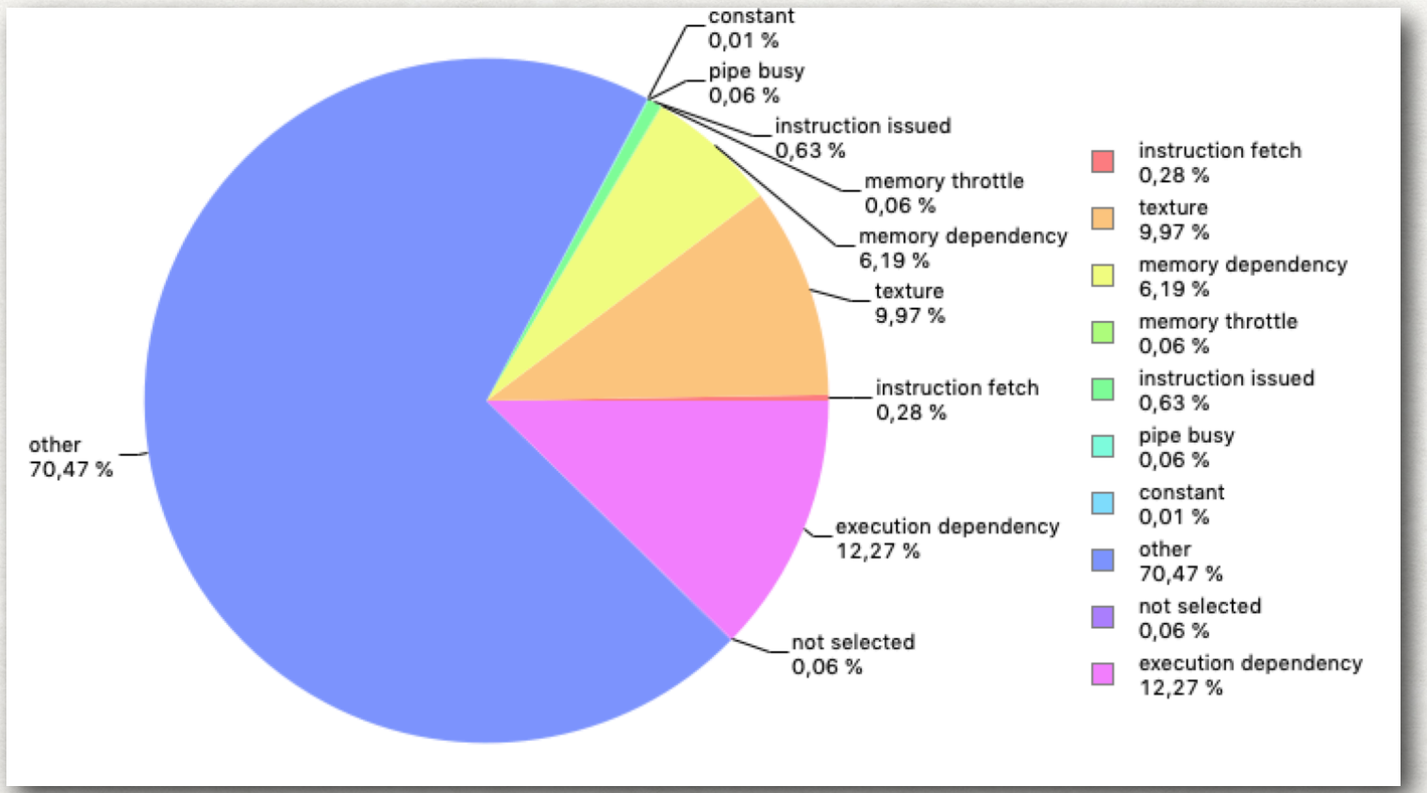
Instruction Execution Counts

The following chart shows the mix of instructions executed by the kernel. The instructions are grouped into classes and for each class the chart shows the percentage of thread execution cycles that were devoted to executing instructions in that class. The "Inactive" result shows the thread executions that did not execute any instruction because the thread was predicated or inactive due to divergence.



LATENCY-BOUND ЯДРА

- compute_utilisation && memory_utilisation is low
- основные стратегии оптимизации latency-bound ядер:
    - увеличиение используемого числа блоков/нитей (низкое ocupancy)
    - изучения причин простоя вычислительных ядер



ВОПРОСЫ?



МГУ имени М.В.Ломоносова

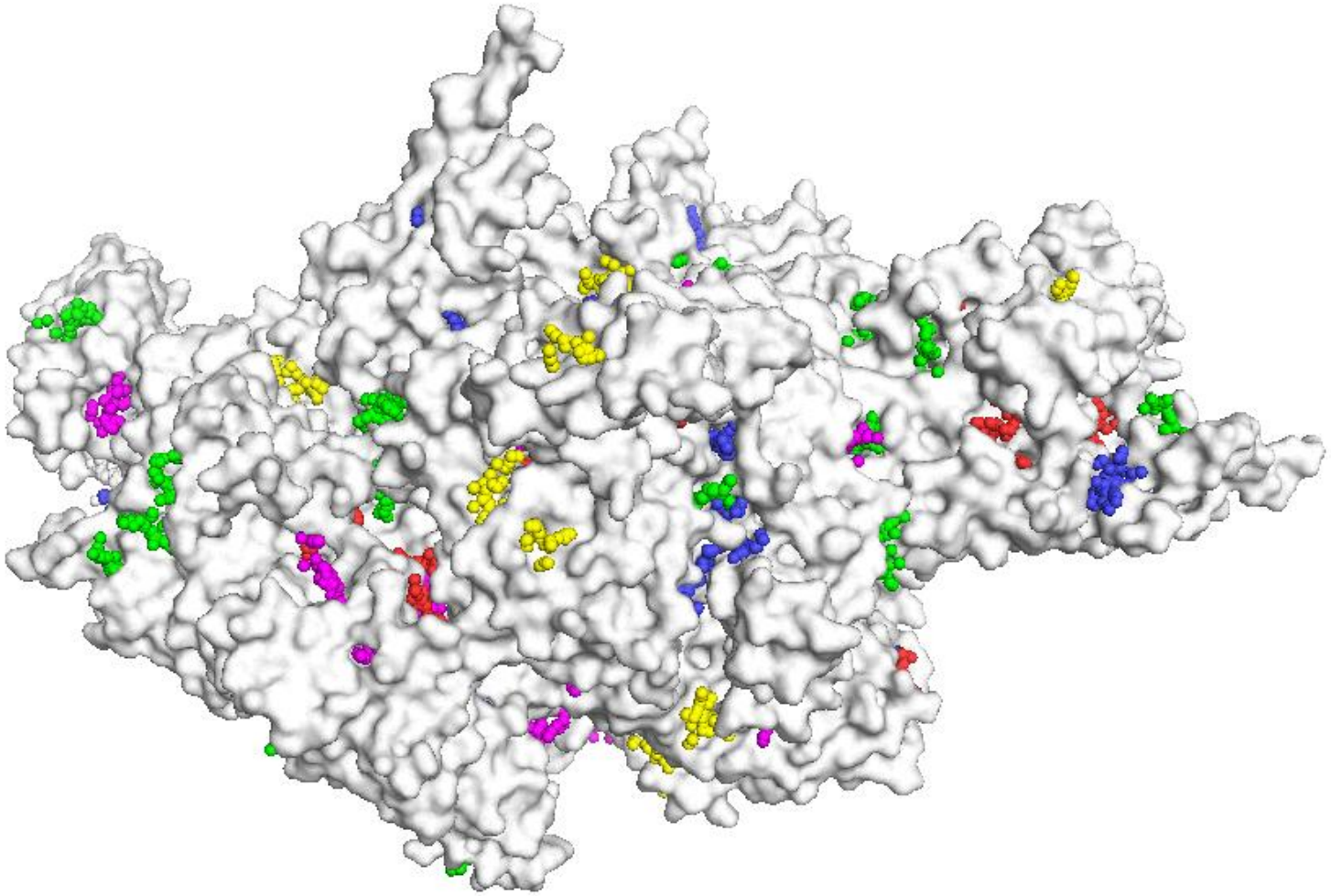
Многоуровневое суперкомпьютерное моделирование белков для решения задач биотехнологии и drug discovery

Дмитрий Суплатов

в.н.с. НИИ Физико-химической биологии
имени А. Н. Белозерского МГУ

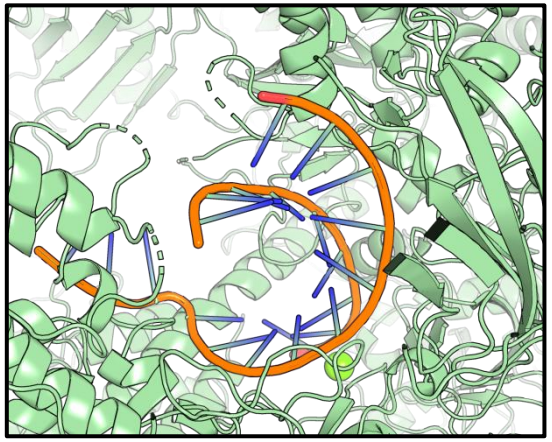
25 ноября 2020

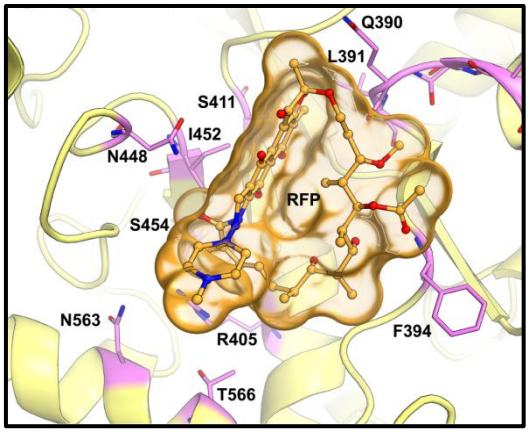
Поверхность практически любого белка содержит большое количество потенциально важных участков связывания лигандов



Structure of DNA-dependent RNA polymerase

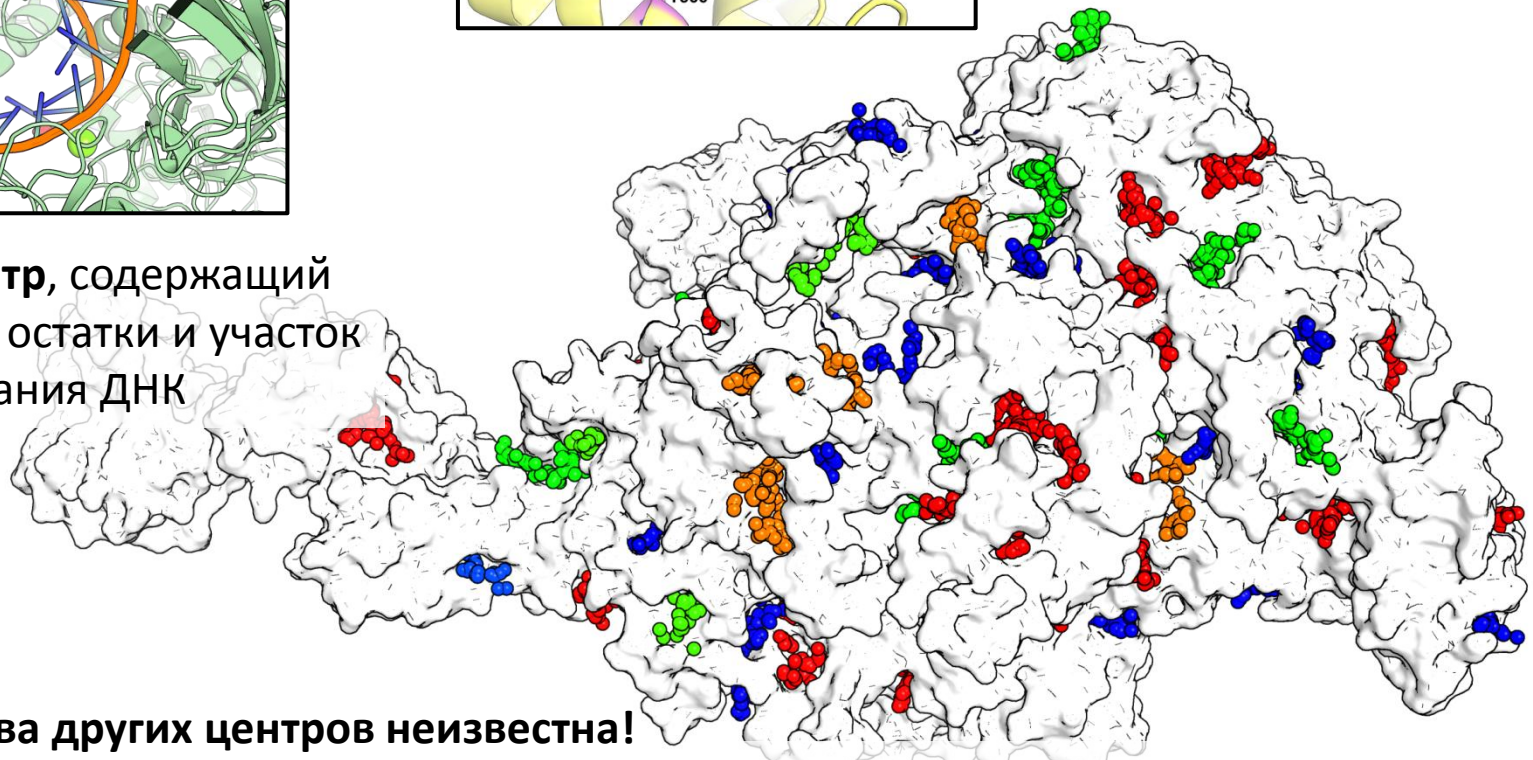
Структура ДНК-зависимой РНК-полимеразы





Несколько известных **аллостерических сайтов**, способных связывать регуляторные лиганды

Активный центр, содержащий каталитические остатки и участок связывания ДНК

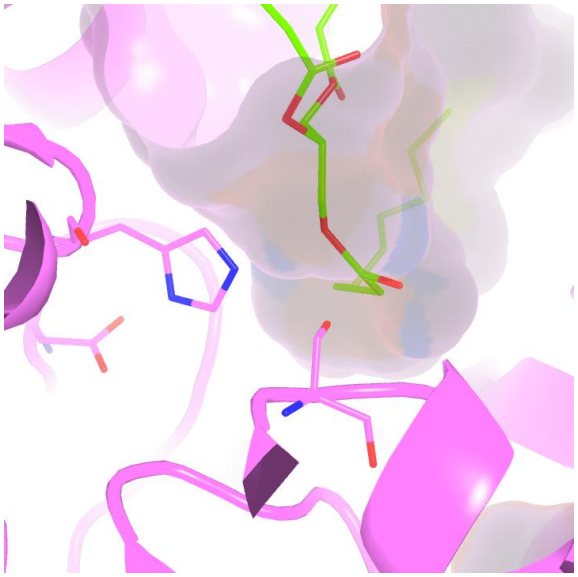


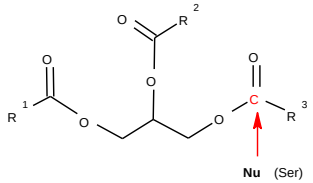
Роль большинства других центров неизвестна!

Д. Суплатов, В. Швядас (2015) *Acta Naturae*, 7(4), 39-52.

Даже похожие центры могут выполнять разные функции

Lipases
(EC 3.1.1.3)

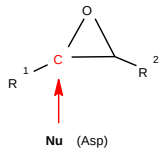




Triacylglycerol

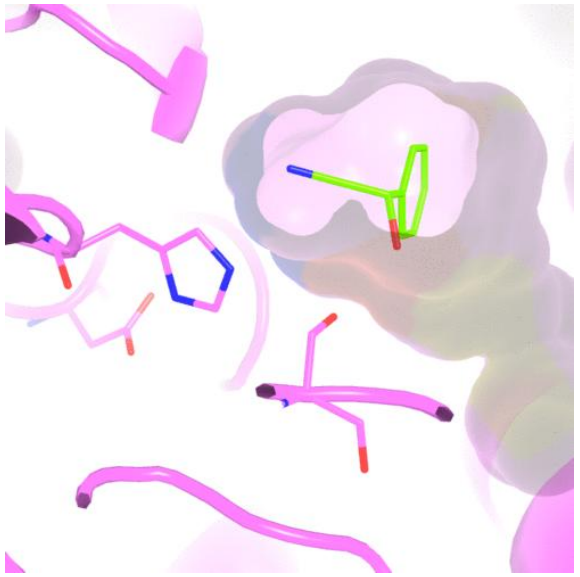
Epoxide hydrolases
(EC 3.3.2.-)

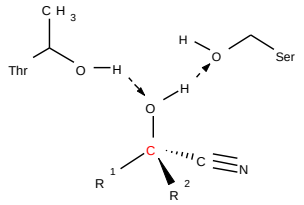




Epoxide

Hydroxynitrilase
(EC 4.1.2.47)





Cyanohydrin

Изучение структурно-функциональных взаимосвязей в белках/ферментах

- Как аннотировать **все важные центры** в структурах белков?
- Почему структурно **похожие** активные центры в гомологичных ферментах катализируют **различные** химические превращения?
- Как изучать структурно-функциональные взаимосвязи и предсказывать **изменения структуры**, которые приводят к **функциональному разнообразию**?
- Как определять **«горячие точки»** в структуре для того, чтобы **изменять функцию** ферментов и создавать более эффективные биокатализаторы?
- Как выбирать лиганды, комплементарные функциональным и регуляторным центрам белков, для направленно**го воздействия на функцию**?

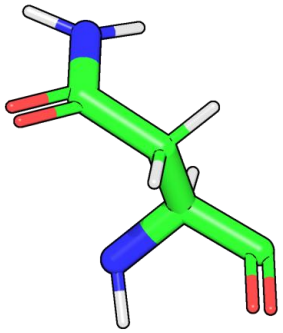
Часть I: Введение (белки)

Что такое белок?

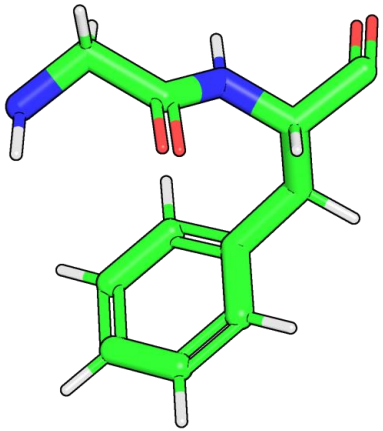
- высокомолекулярные органические вещества, состоящие из альфа-аминокислот, соединённых в цепочку пептидной связью

Что такое белок?

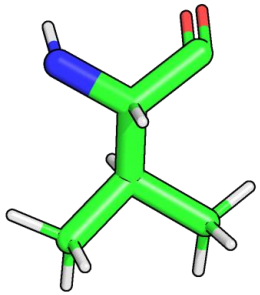
- высокомолекулярные органические вещества, состоящие из альфа-аминокислот, соединённых в цепочку пептидной связью



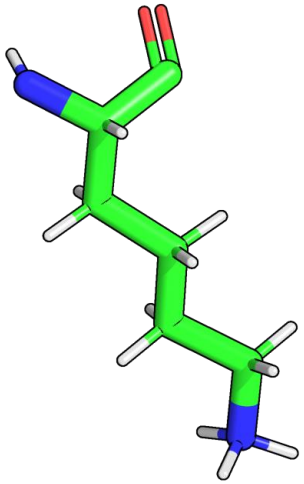
Аспарагин



Фенилаланин



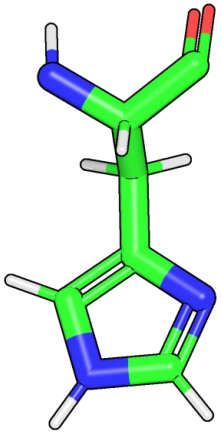
Валин



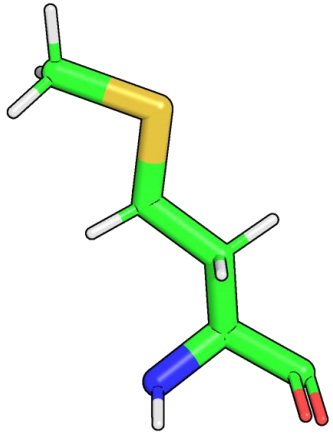
Лизин

Что такое белок?

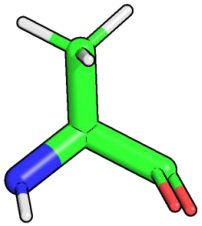
- высокомолекулярные органические вещества, состоящие из альфа-аминокислот, соединённых в цепочку пептидной связью



Гистидин



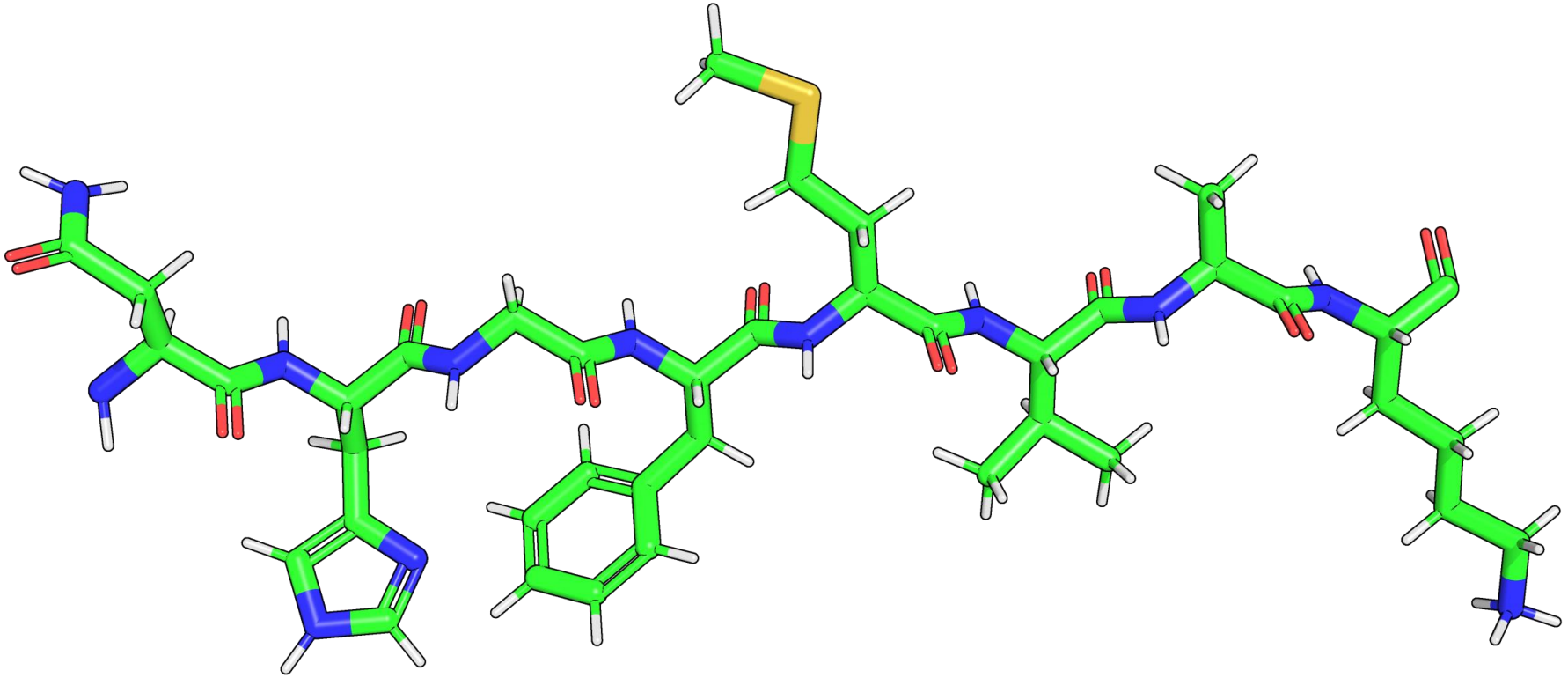
Метионин



Аланин

Что такое белок?

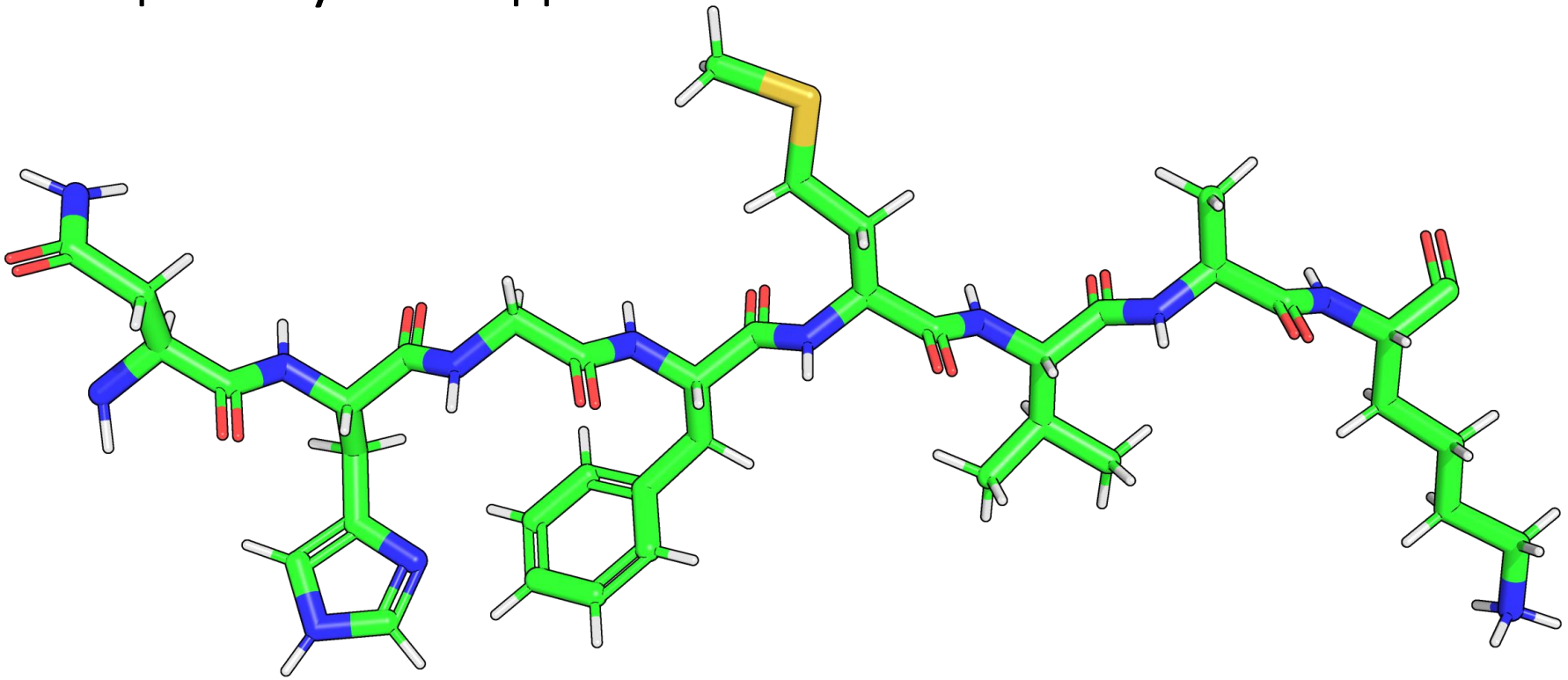
- высокомолекулярные органические вещества, состоящие из альфа-аминокислот, соединённых в цепочку пептидной связью



Аспарагин | Гистидин | Фенилаланин | Метионин | Валин | Аланин | Лизин

Что такое белок?

- высокомолекулярные органические вещества, состоящие из альфа-аминокислот, соединённых в цепочку пептидной связью: **NHPMVAK**



Аспарагин | Гистидин | Фенилаланин | Метионин | Валин | Аланин | Лизин

Что такое белок?

Биополимер длинной от нескольких десятков до нескольких тысяч аминокислот

Белок: уровни организации

Первичная структура

или

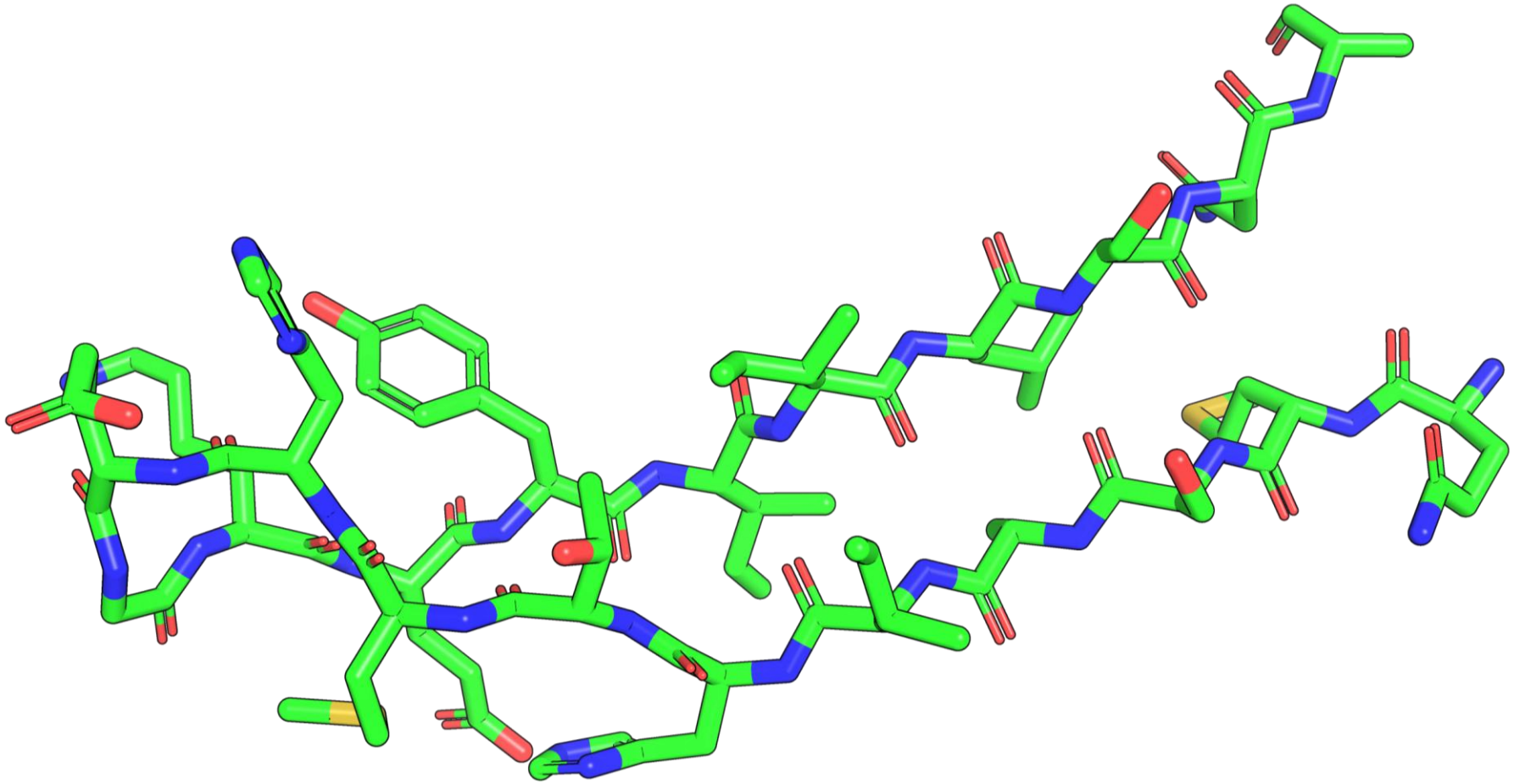
Аминокислотная последовательность

```
MKNRNRMIVN CVTASLMYYW SLPALAEQSS SEIKIVRDEY GMPHIYANDT WHLFYGYGYV
VAQDRLFQME MARRSTQGTV AEVLGKDFVK FDKDIRRNYW PDAIRAQIAA LSPEDMSILQ
GYADGMNAWI DKVNTNPETL LPKQFNTFGF TPKRWEPFDV AMIFVGTMAN RFSDSTSEID
NLALLTALKD KYGVSQGMAV FNQLKWLVNP SAPTTIAVQE SNYPLKFNQQ NSQTAALLPR
YDLPAPMLDR PAKGADGALL ALTAGKNRET IAAQFAQGGA NGLAGYPTTS NMWVIGKSKA
QDAKAIMVNG PQFGWYAPAY TYGIGLHGAG YDVTGNTPFA YPGLVFGHNG VISWGSTAGF
GDDVDIFAER LSAEKPGYYL HNGKWVKMLS REETITVKNG QAETFTVWRT VHGNILQTDQ
TTQTAYAKSR AWDGKEVASL LAWTHQMKAK NWQEWTQQAA KQALTINWYY ADVNGNIGYV
HTGAYPDRQS GHDPRLPVPG TGKWDWKGLL PFEMNPKVYN PQSGYIANWN NSPQKDYPAS
DLFAFLWGGA DRVTEIDRLL EQKPRLTADQ AWDVIRQTSR QDLNLRLFLP TLQAATSGLT
QSDPRRQLVE TLTRWDGINL LNDDGKTWQQ PGSAILNVWL TSMLKRTVVA AVPMPFDKWY
SASGYETTQD GPTGSLNISV GAKILYEAVQ GDKSPIPQAV DLFAGKPQQE VVLAALEDTW
ETLSKRYGNN VSNWKTPAMA LTFRANNFFG VPQAAAEETR HQAEYQNRGT ENDMIVFSPT
TSDRPVLAWD VVAPGQSGFI APDGTVDKHY EDQLKMYENF GRKSLWLTKQ DVEAHKESQE
VLHVQR
```

Текстовая форма представления белка

Белок: уровни организации

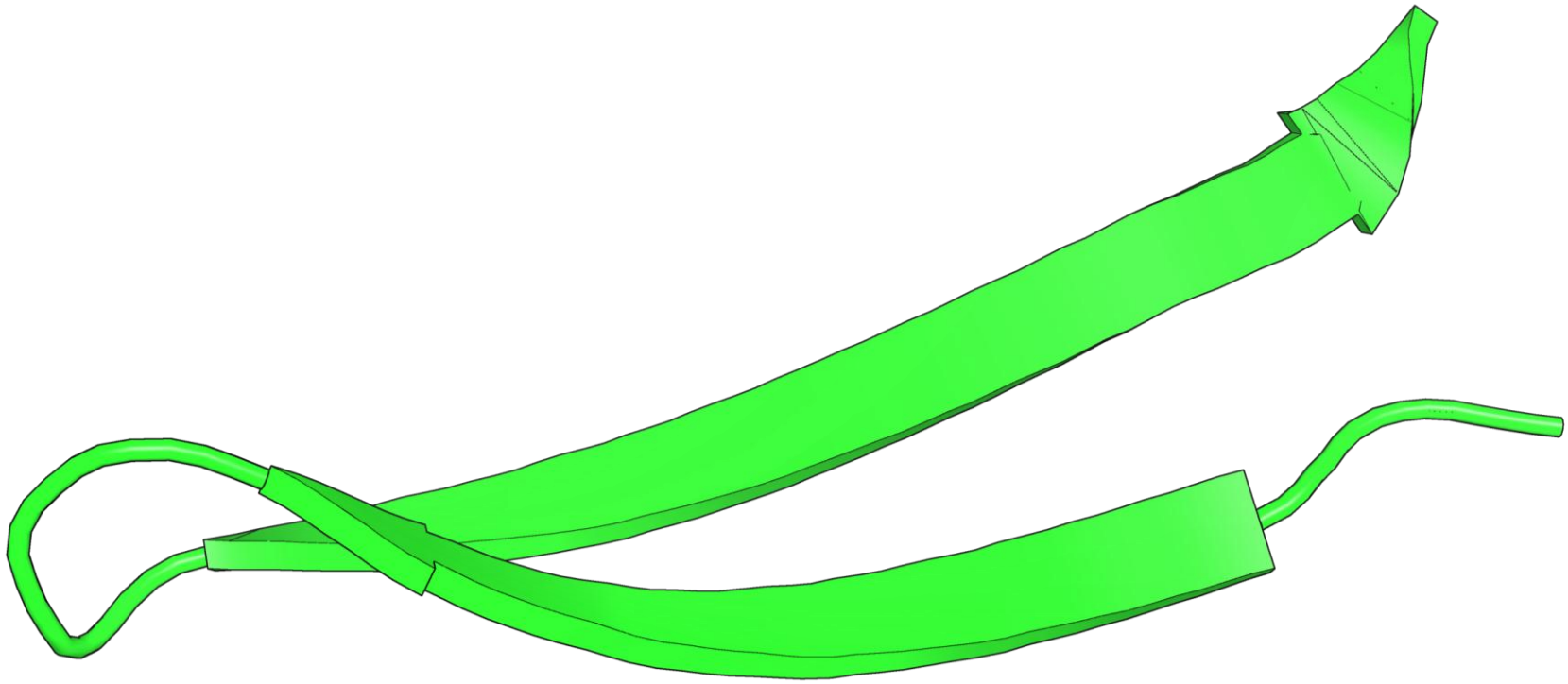
Вторичная структура *или* регулярная структура



Локальная организация аминокислот в пространстве

Белок: уровни организации

Вторичная структура *или* регулярная структура

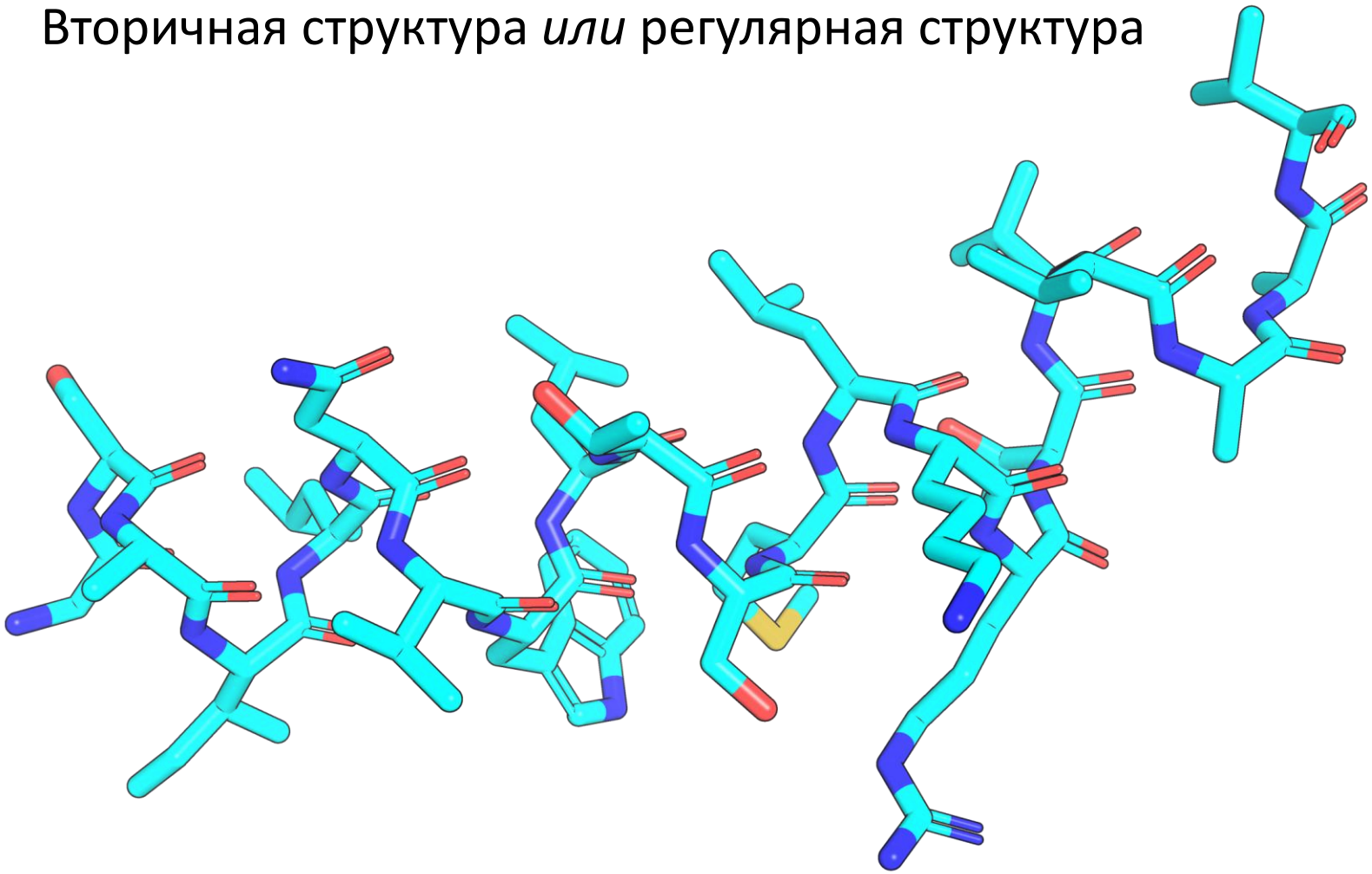


β-слой

Локальная организация аминокислот в пространстве

Белок: уровни организации

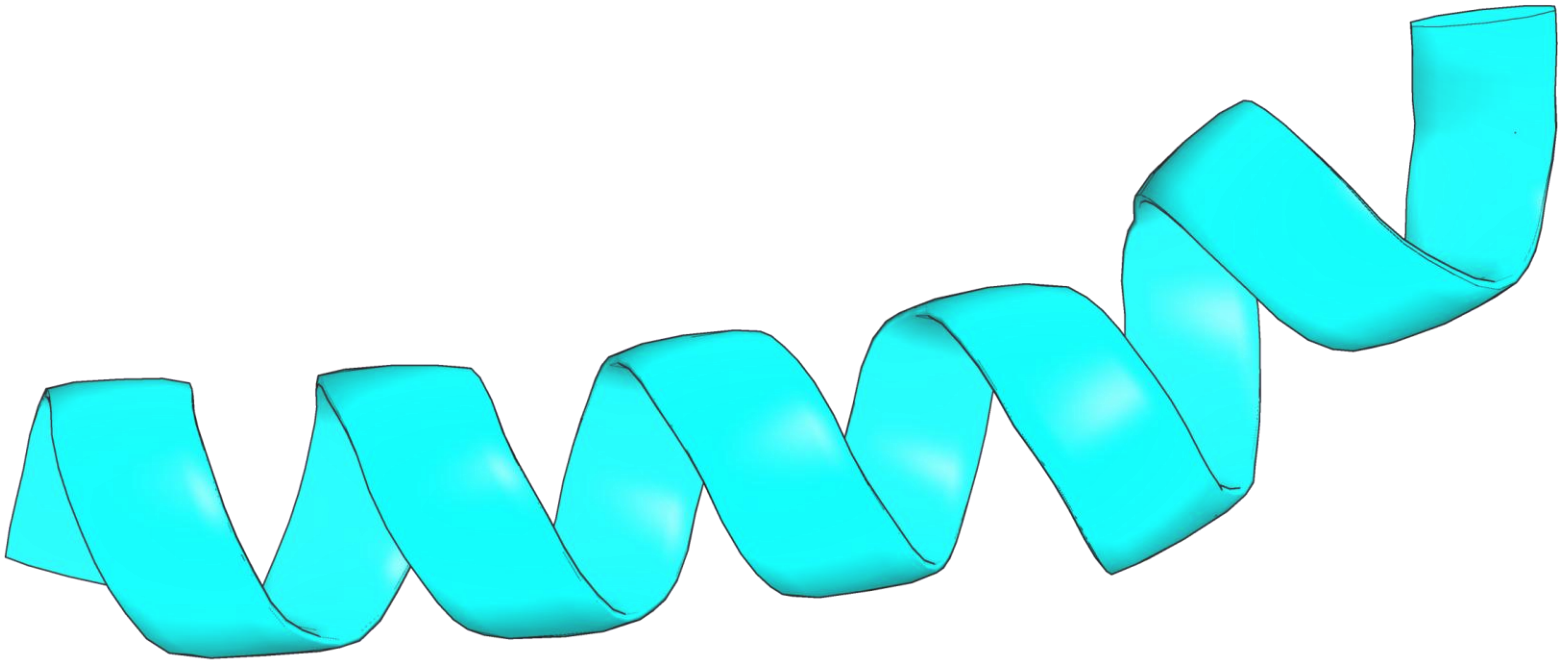
Вторичная структура *или* регулярная структура



Локальная организация аминокислот в пространстве

Белок: уровни организации

Вторичная структура *или* регулярная структура

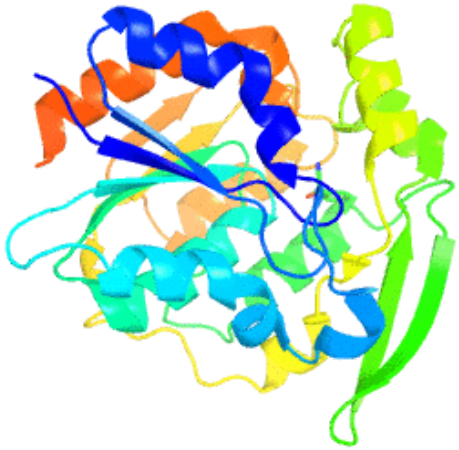


α-спираль

Локальная организация аминокислот в пространстве

Белок: уровни организации

Третичная структура *или* 3D- (трехмерная) структура



Hydroxynitrile lyase from *H.brasiliensis*
(4.1.2.**47**)

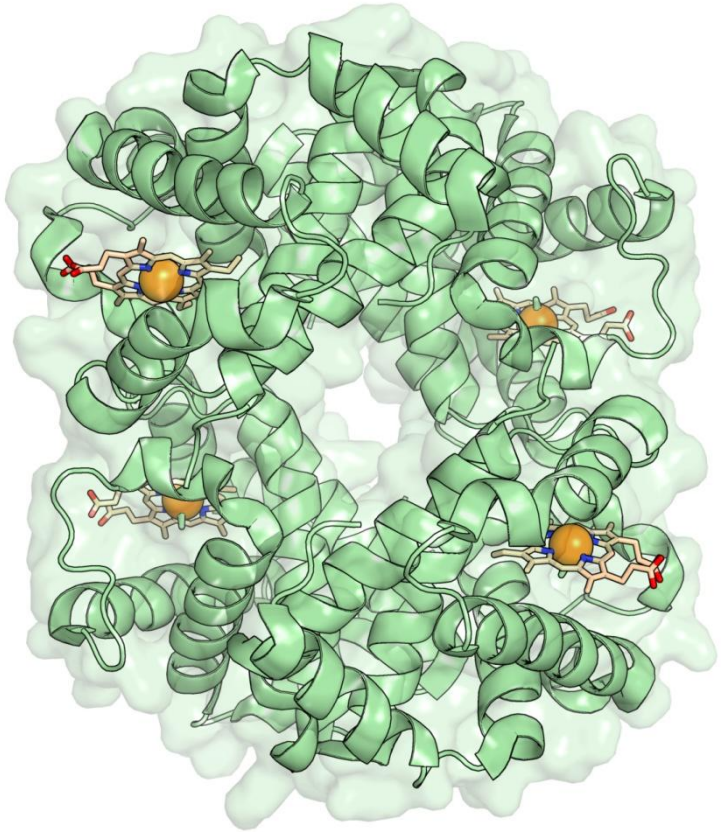


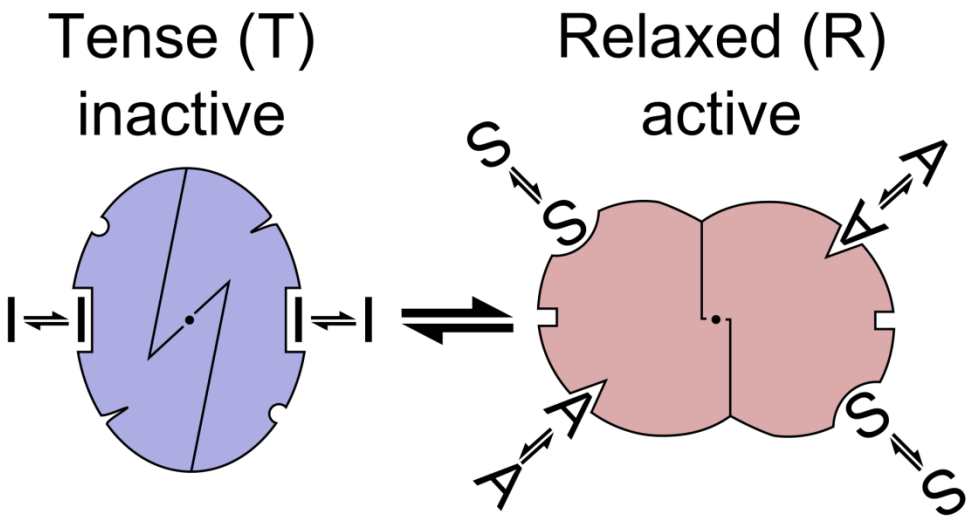
Hydroxynitrile lyase from *S.bicolor*
(4.1.2.**11**)

Расположение атомов аминокислот белка в трехмерном пространстве

Белок: уровни организации

Четвертичная структура

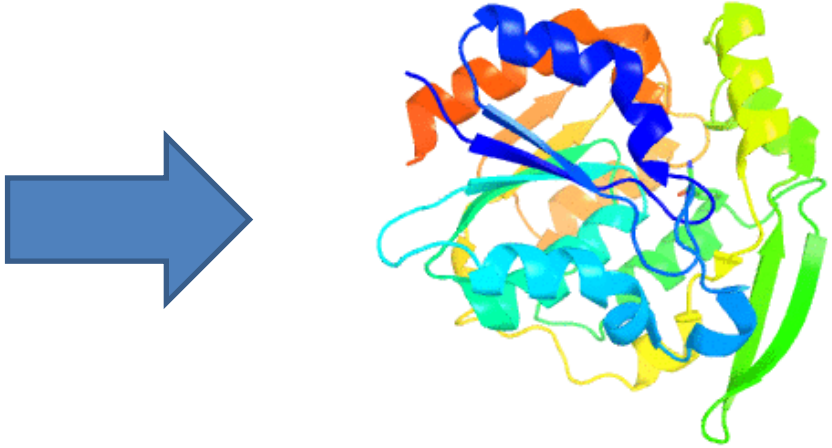




Уровень организации некоторых сложно устроенных белков, состоящий из нескольких субъединиц

Аминокислотная последовательность белка однозначно задает его структуру и функцию

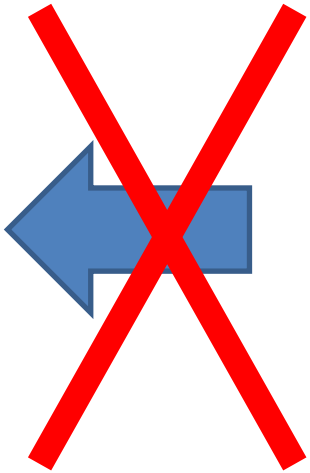
```
MKNRNRMIVNCVTASLMYYWSLPALAEQSS
SEIKIVRDEYGMPHIYANDTWHLFYGYGYV
VAQDRLFQMEMARRSTQGTVAEVLGKDFVK
FDKDIRRNYWPDAIRAQIAALSPEDMSILQ
GYADGMNAWIDKVNTNPETLLPKQFNTFGF
TPKRWEPFDVAMIFVGTMANRFSDSTSEID
NLALLTALKDKYGVSQGMAVFNQLKWLVNP
SAPTTIAVQESNYPLKFNQQNSQTAALLPR
YDLPAPMLDRPAKGADGALLALTAGKNRET
IAAQFAQGGANGLAGYPTTSNMWVIGKSKA
QDAKAIMVNGPQFGWYAPAYTYGIGLHGAG
YDVTGNTPFAYPGLVFGHNGVISWGSTAGF
GDDVDIFAERLSAEKPGYYLHNGKWVKMLS
REETITVKNGQAETFTVWRTVHGNILQTDQ
TTQTAYAKSRAWDGKEVASLLAWTHQMKAK
NWQEWTQQAAKQALTINWYYADVNGNIGYV
HTGAYPDRQSGHDPRLPVPGTGKWDWKGLL
PFEMNPKVYNPQSGYIANWNNSPQKDYPAS
DLFAFLWGGADRVTEIDRLLEQKPRLTADQ
AWDVIRQTSRQDLNLRLFLPTLQAATSGLT
QSDPRRQLVETLTRWDGINLLNDDGKTWQQ
PGSAILNVWLTSMLKRTVVAAVPMPFDKWY
SASGYETTQDGPTGSLNISVGAKILYEAVQ
GDKSPIPQAVDLFAGKPQQEVVLAALEDTW
ETLSKRYGNNVSNWKTPAMALTFRANNFFG
VPQAAAEETRHQAEYQNRGTENDMIVFSPT
TSDRPVLAWDVVAPGQSGFIAPDGTVDKHY
EDQLKMYENFGRKSLWLTKQDVEAHKESQE
VLHVQR
```

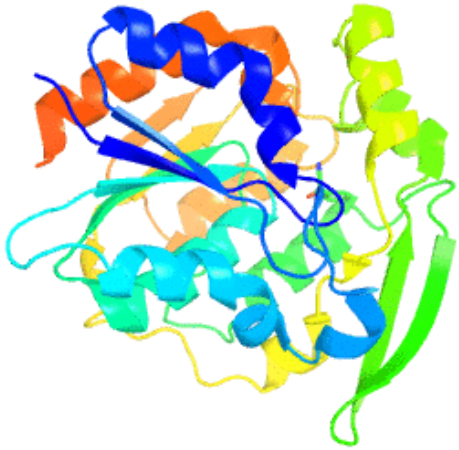


Не совсем так, но в первом приближении сойдет

Эквивалентная трехмерная структура белка может быть реализована схожими, но существенно различающимися аминокислотными последовательностями

MKNRNRMIVNCVTASLMYYWSLPALAEQSS
SEIKIVRDEYGMPHIYANDTWHLFYGYGYV
VAQDRLFQMEMARRSTQGTVAEVLGKDFVK
FDKDIRRNYWPDAIRAQIAALSPEDMSILQ
GYADGMNAWIDKVNTNPETLLPKQFNTFGF
TPKRWEPFDVAMIFVGTMANRFSDSTSEID
NLALLTALKDKYGVSQGMAVFNQLKWLVNP
SAPTTIAVQESNYPLKFNQQNSQTAALLPR
YDLPAPMLDRPAKGADGALLALTAGKNRET
IAAQFAQGGANGLAGYPTTSNMWVIGKSKA
QDAKAIMVNGPQFGWYAPAYTYGIGLHGAG
YDVTGNTPFAYPGLVFGHNGVISWGSTAGF
GDDVDIFAERLSAEKPGYYLHNGKWVKMLS
REETITVKNGQAETFTVWRTVHGNILQTDQ
TTQTAYAKSRAWDGKEVASLLAWTHQMKAK
NWQEWTQQAAKQALTINWYYADVNGNIGYV
HTGAYPDRQSGHDPRLPVPGTGKWDWKGLL
PFEMNPKVYNPQSGYIANWNNSPQKDYPAS
DLFAFLWGGADRVTEIDRLLEQKPRLTADQ
AWDVIRQTSRQDLNLRLFLPTLQAATSGLT
QSDPRRQLVETLTRWDGINLLNDDGKTWQQ
PGSAILNVWLTSMLKRTVVAAVPMPFDKWY
SASGYETTQDGPTGSLNISVGAKILYEAVQ
GDKSPIPQAVDLFAGKPQQEVVLAALEDTW
ETLSKRYGNNVSNWKTPAMALTFRANNFFG
VPQAAAEETRHQAEYQNRGTENDMIVFSPT
TSDRPVLAWDVVAPGQSGFIAPDGTVDKHY
EDQLKMYENFGRKSLWLTKQDVEAHKESQE
VLHVQR



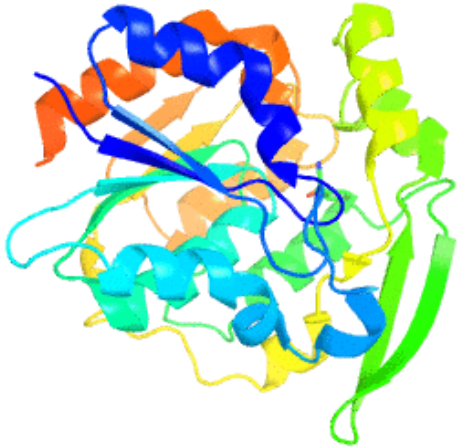


Мутация (замена) в аминокислотной последовательности белка может привести к изменению его функции

MKNRNRMIVNCVTASLMYYWSLPALAEQSSSEIKTVRDEYGMPHIYANDTWHLFYGYGYVVAQDRLFQME
MARRSTQGTVAEVLGKDFVKFDKDIRRNYWPDAIRAQIAALSPEDMSILQGYADGMNAWIDKVNTNPETL
LPKQFNTFGFTPKRWEPFDVAMIFVGTMANRFSDSTSEIDNLALLTALKDKYGVSQGMAVFNQLKWLVNP

Мутация R→K

MKNRNRMIVNCVTASLMYYWSLPALAEQSSSEIKTVRDEYGMPHIYANDTWHLFYGYGYVVAQDRLFQME
MARRSTQGTVAEVLGKDFVKFDKDIRRNYWPDAIKAQIAALSPEDMSILQGYADGMNAWIDKVNTNPETL
LPKQFNTFGFTPKRWEPFDVAMIFVGTMANRFSDSTSEIDNLALLTALKDKYGVSQGMAVFNQLKWLVNP



Белок с новой функцией

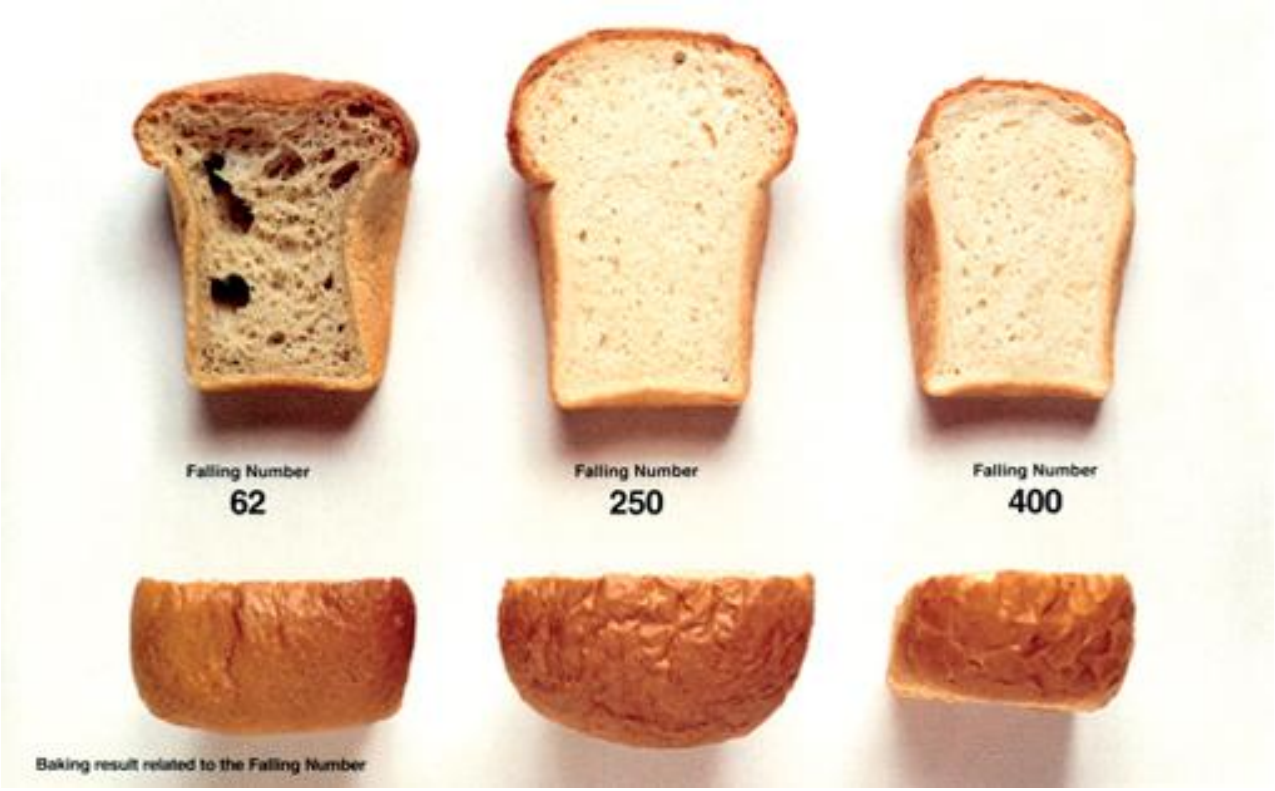
Часть II: Белковая инженерия

Белковая инженерия

- Область науки (биотехнологии), которая занимается разработкой препаратов белков/ферментов с полезными/улучшенными свойствами;

Где нужны «полезные белки»?

- Пищевая промышленность;



Использование амилаз в хлебопекарном деле

Где нужны «полезные белки»?

- Химическая промышленность/бытовая химия;



протеазы, амилазы и липазы в стиральном порошке



Органический синтез оптически чистых аминов и эфиров с использованием липаз

Где нужны «полезные белки»?

- Медицина/фармакология



Зубная паста, содержащая лактопероксидазу



Синтез полусинтетических бета-лактамных антибиотиков с использованием фермента пенициллинацилазы

Где нужны «полезные белки»?

- Очищающие или моющие средства;
- Продукты (сыры, молоко, фрукто-содержащие продукты, …);
- Выпечка;
- Напитки;
- Текстиль;
- Производство бумаги;
- Органический синтез;
- Производство на основе кожи;
- Личная гигиена;
- …..

Белковая инженерия: актуальность

- Область науки (биотехнологии), которая занимается ***разработкой*** препаратов белков/ферментов с полезными/улучшенными свойствами;

Белковая инженерия: актуальность

Получение препаратов белков/ферментов с:

- Улучшенной стабильностью
  - При высоких значениях температуры, которые используются в промышленности для ускорения протекания реакций и предотвращения заражения среды;
  - При экстремальных значениях pH;
  - В органических средах;
- Увеличенной активностью;
- Расширенной субстратной специфичностью;

Белковая инженерия: подходы

- Область науки, которая занимается ***разработкой*** препаратов белков/ферментов с полезными/улучшенными свойствами;

Как?

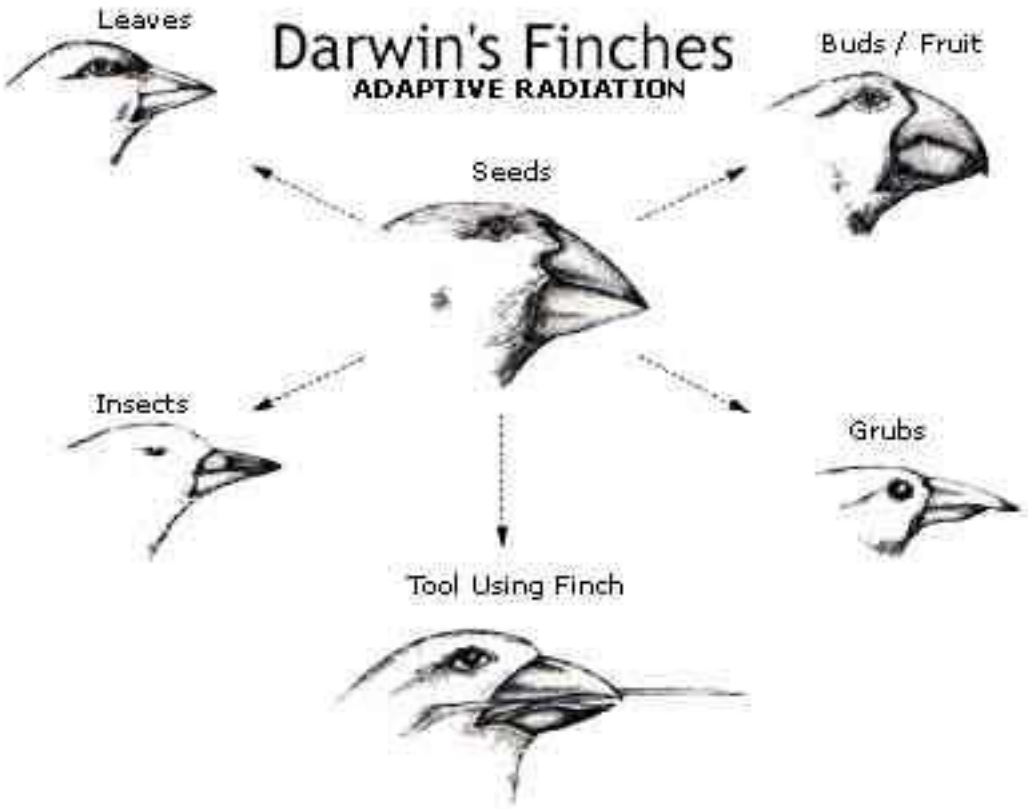
Наблюдение и использование



Флорис Клас ван Дейк (1613) Натюрморт с фруктами, орехами и сыром

Теория эволюции Дарвина

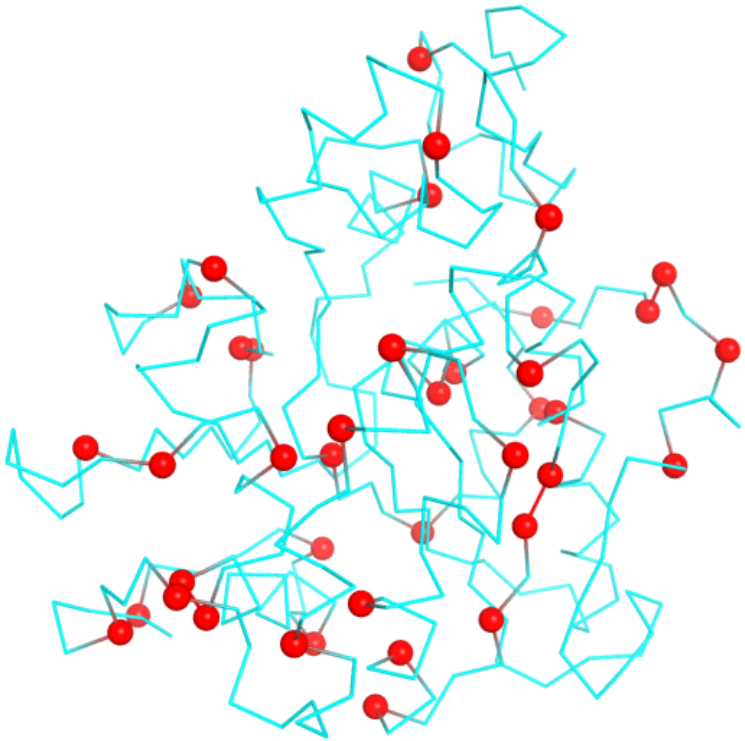
Главным фактором эволюции является естественный отбор



Закрепление **случайно возникших изменений**, дающих конкурентное преимущество в конкретных условиях окружающей среды

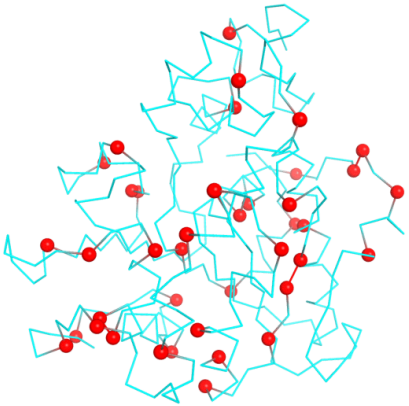
Стохастический подход в белковой инженерии

Попробуем вносить случайные изменения непосредственно в последовательность белка – возможно, в какой-то момент мы случайно изменим его свойства в нужную нам сторону



Стохастический подход в белковой инженерии

Попробуем вносить случайные изменения непосредственно в последовательность белка – возможно, в какой-то момент мы случайно изменим его свойства в нужную нам сторону



Большая библиотека мутантов (10^6)

Проверка свойств мутантов

Мутант с нужными свойствами найден?

Нет - продолжаем

Внесение случайных мутаций в структуру белка

Да - завершаем

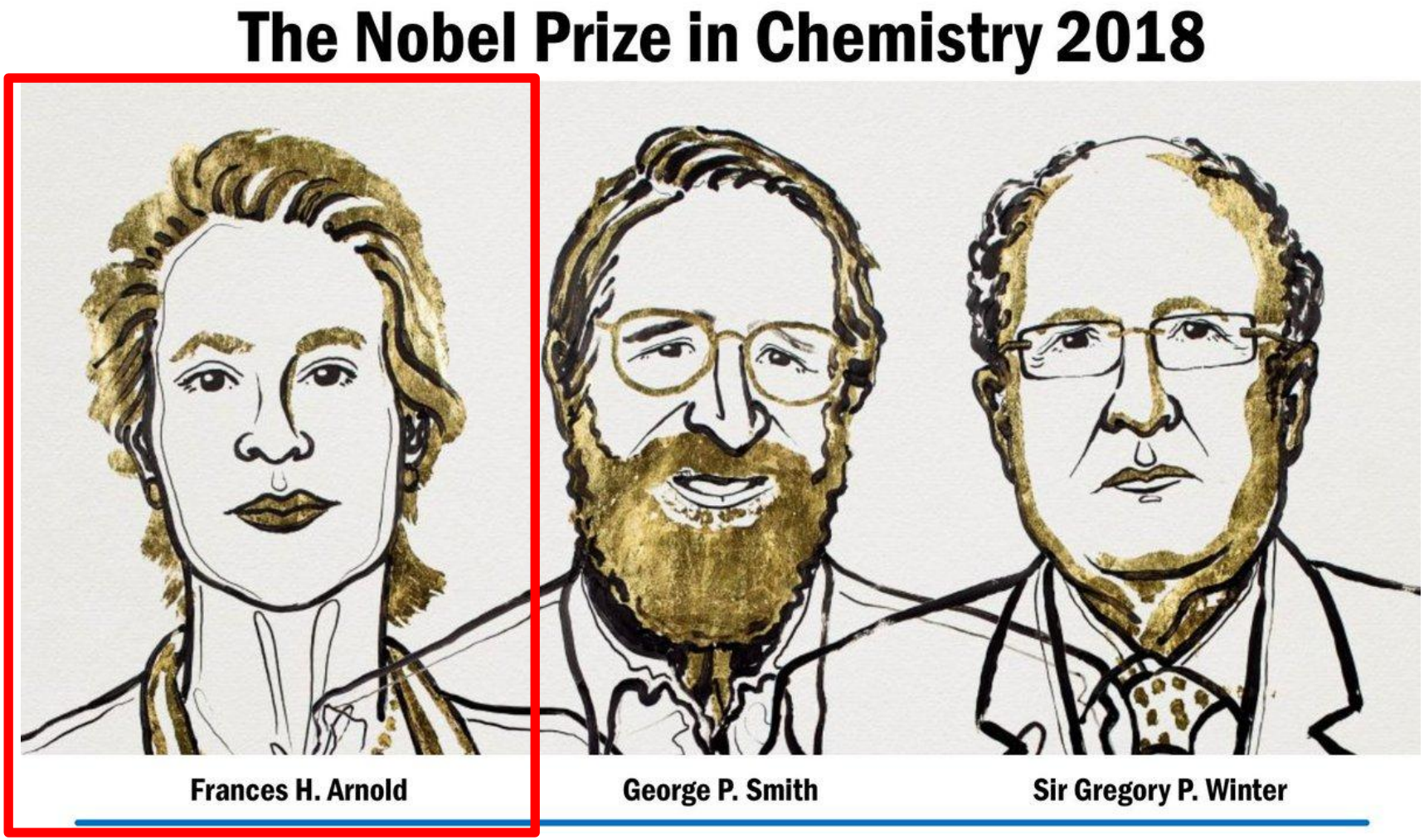
«Направленная эволюция» белков в лабораторных условиях

Возможно, наиболее широко применяемый на практике подход к получению препаратов белков/ферментов с «улучшенными» свойствами

«Направленная эволюция» белков в лабораторных условиях

- Хороший подход или нет?

«Направленная эволюция»



«Направленная эволюция» белков в лабораторных условиях

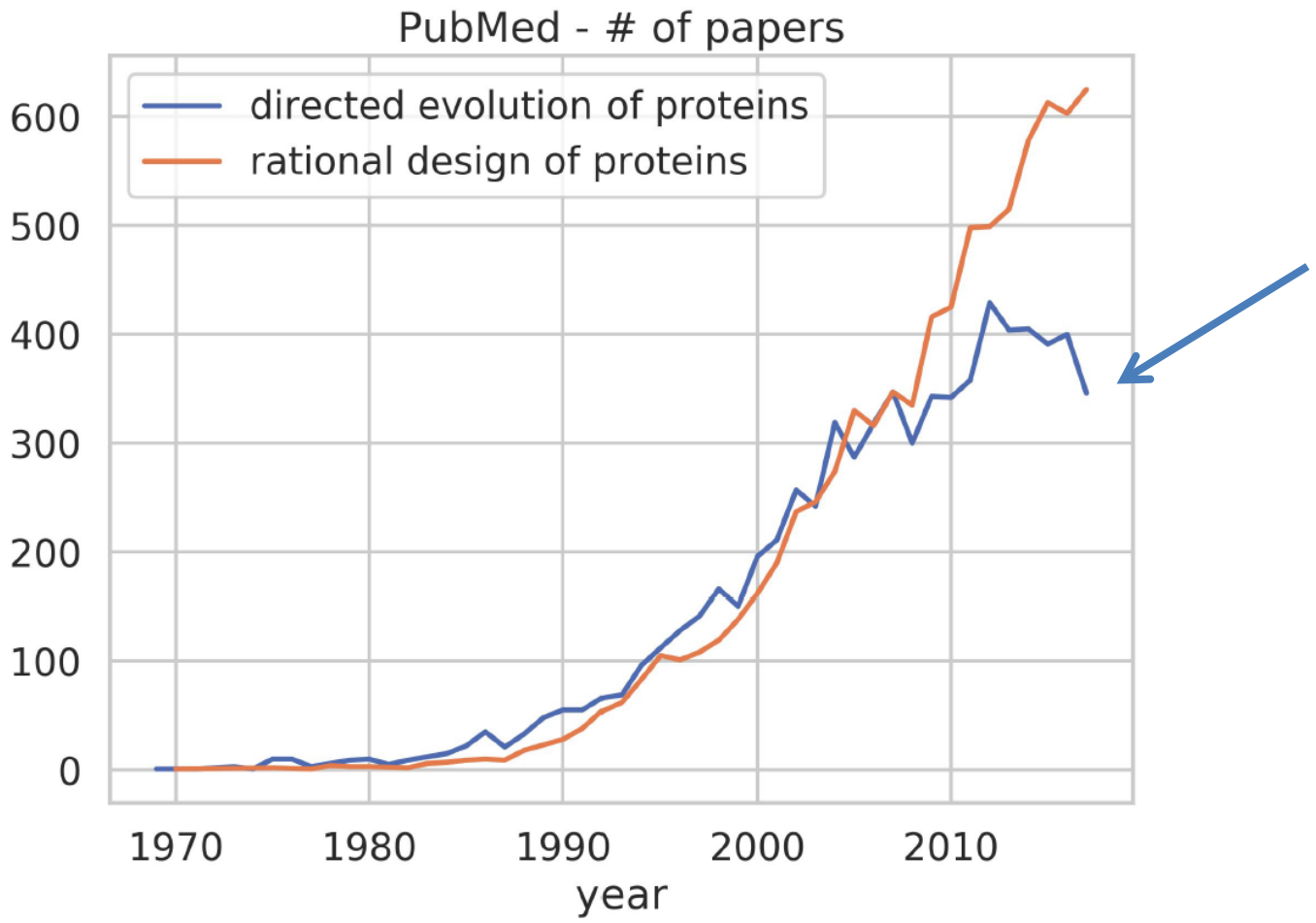
- Хороший подход или нет?

«Направленная эволюция» белков в лабораторных условиях

- Хороший подход или нет?

  Не очень …

«Направленная эволюция» белков в лабораторных условиях

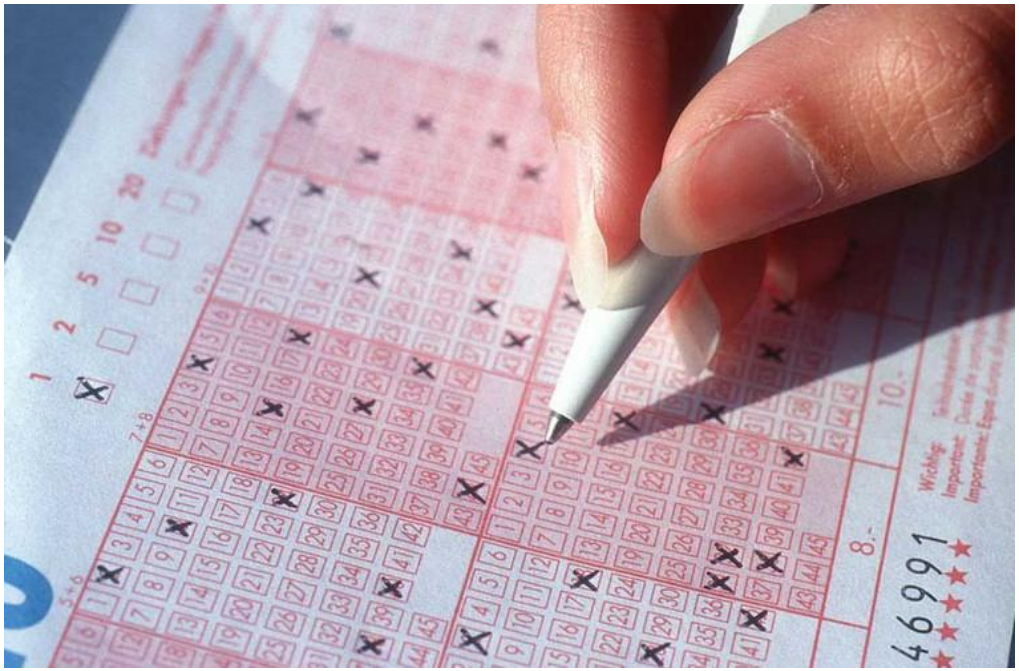


Число возможных изменений (мутаций) в структуре белка астрономически велико

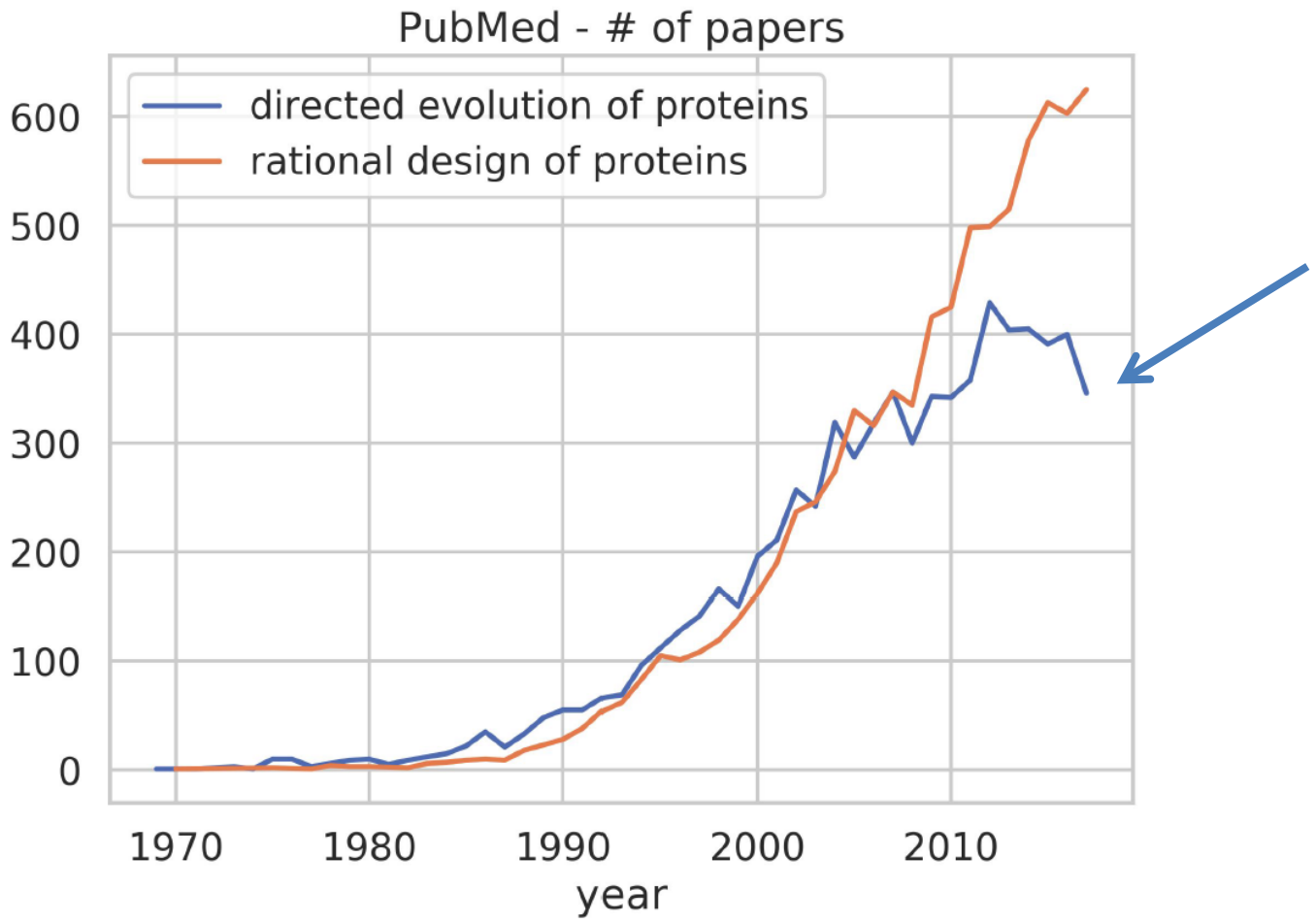
- 20 «букв» в белковом «алфавите»;
- **20^{300}** (практически бесконечное) число возможных вариантов в белке длинной 300 позиций;
- **Подавляющее число этих вариантов имеют «негативных» (неактивный) фенотип** (т.е. «не работают», «плохо работают» или «хуже работают» по сравнению с белком «дикого типа»);
- Экспериментально можем изучить не более нескольких миллионов мутантов (**10^6**).

Фундаментальная причина низкой эффективности стохастических подходов (направленной эволюции) в белковой инженерии

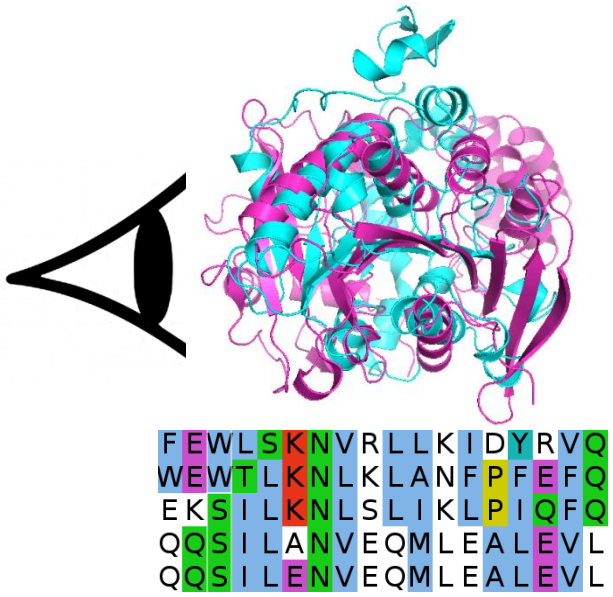
Вероятность случайно угадать небольшое количество «правильных ответов» за 10^6 попыток из 20^{300} возможных вариантов очень невелика



«Направленная эволюция» белков в лабораторных условиях



Эмпирический рациональный дизайн



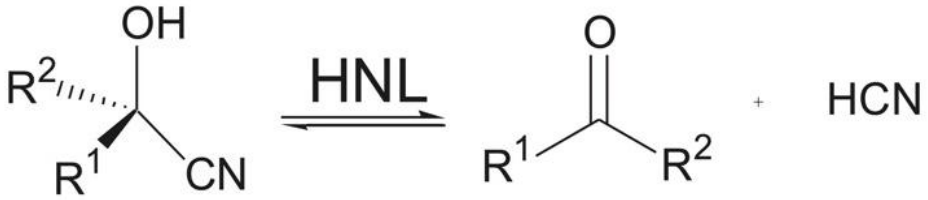
Попытка «на глаз» (на уровне ручного экспертного анализа) определить конкретные изменения структуры белка, приводящие к «улучшению» его свойств

Эмпирический рациональный дизайн

Экспертный структурный анализ

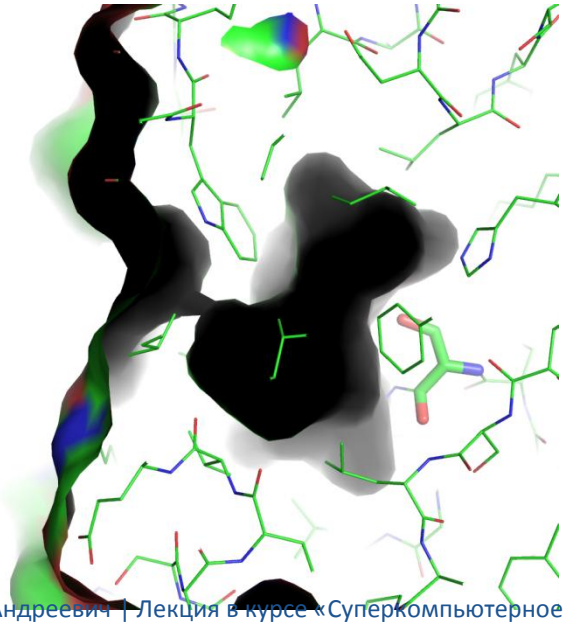
Объект: оксинитрилаза из *Manihot esculenta*

Реакция: стереоселективное расщепление циангидринов до альдегидов и кетонов



Проблема: ограниченная субстратная специфичность

Объяснение: активный центр доступен из растворителя через узкий канал

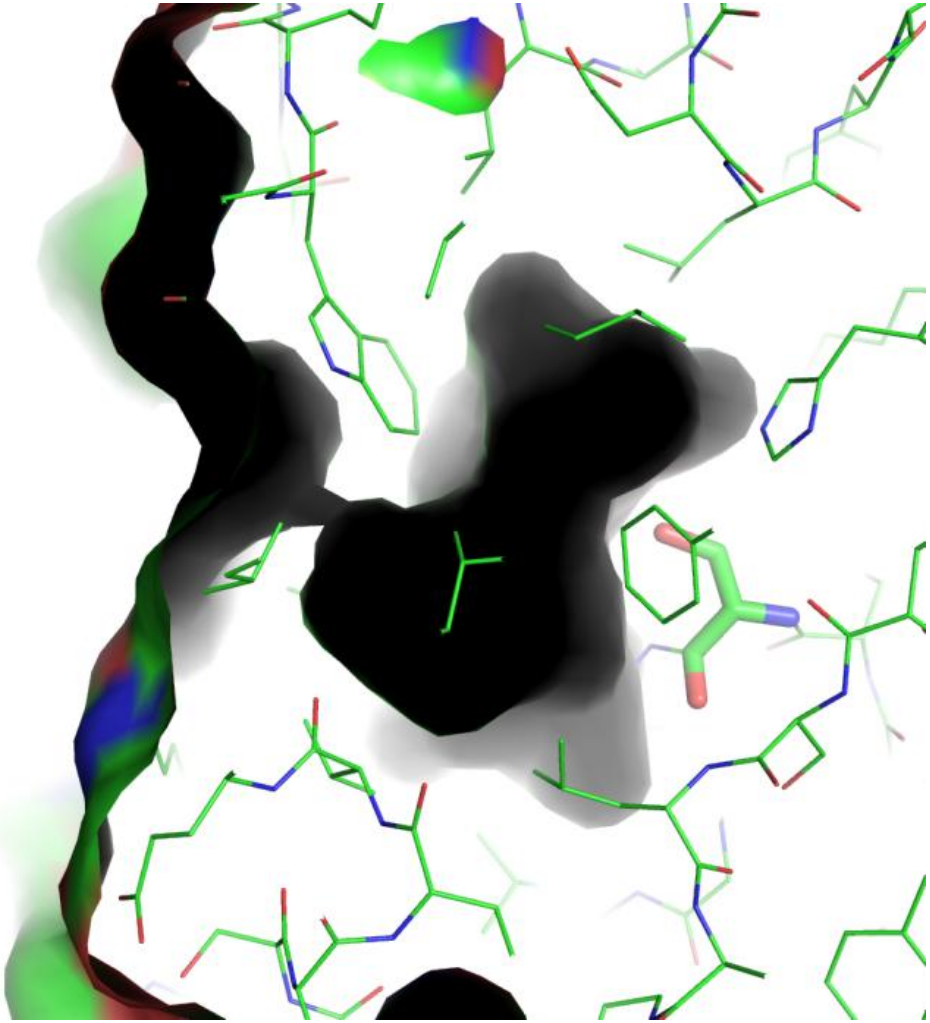


Lauble, H., Miehlich, B., Förster, S., Kobler, C., Wajant, H., & Effenberger, F. (2002). Protein science, 11(1), 65-71.

Эмпирический рациональный дизайн

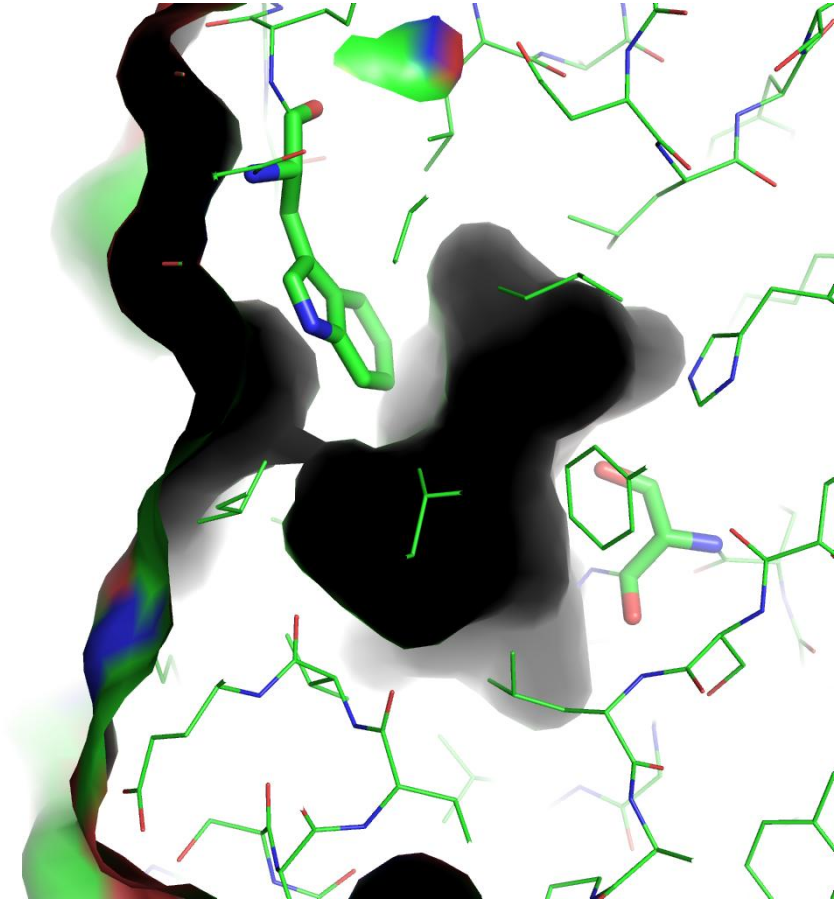
Экспертный структурный анализ

Проблема: ограниченная субстратная специфичность

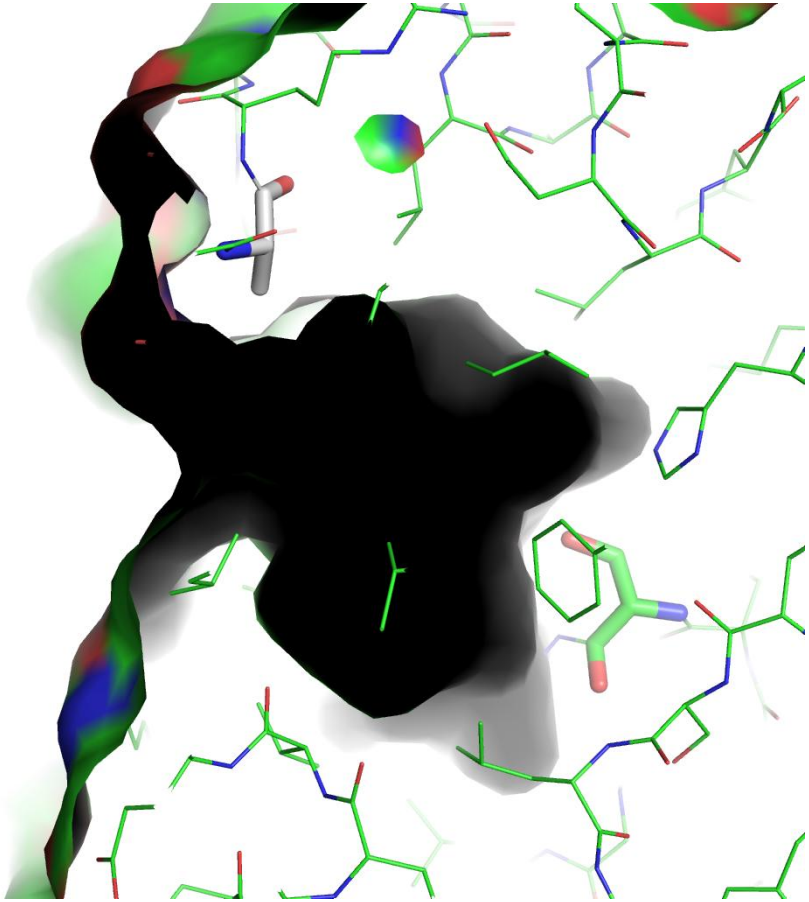


Эмпирический рациональный дизайн

Экспертный структурный анализ



W128



W128 → A

Мутация большого гидрофобного остатка W128 приводит к расширению канала, через который субстрат попадает из растворителя в активный центра

Lauble, H., Miehlich, B., Förster, S., Kobler, C., Wajant, H., & Effenberger, F. (2002). *Protein science*, 11(1), 65-71.

Эмпирический рациональный дизайн

- Хороший подход или нет?

Эмпирический рациональный дизайн

- Хороший подход или нет?

  Не очень … потому что в большинстве случаев все не так просто…

Белковая инженерия: современное представление

«Правильную» мутацию крайне сложно угадать случайным перебором или увидеть эксперту «на глаз»

Белковая инженерия: современное представление

Путь к эффективному получения и практическому использованию полезных белков лежит **через понимание** механизмов их действия на молекулярном уровне

Систематический рациональный дизайн

- Попытка **вначале изучить и понять механизм** искомого свойства на молекулярном уровне с использованием комплекса методов (прежде всего – методов компьютерной биологии и теоретической химии);

- **Затем**, на основе достигнутого понимания механизма – **реализовать направленные изменения** в белке ограниченным количеством мутаций.

Suplatov, D., Voevodin, V., & Švedas, V. (2015). Robust enzyme design: Bioinformatic tools for improved protein stability. *Biotechnology journal*, 10(3), 344-355.

Часть II: Методы

Три уровня суперкомпьютерного моделирования белков

Уровень 1: Биоинформатический анализ всей доступной информации;

Уровень 2: Автоматизированный анализ текстов ранее опубликованных исследований;

Уровень 3: Молекулярное моделирование для понимания механизмов действия

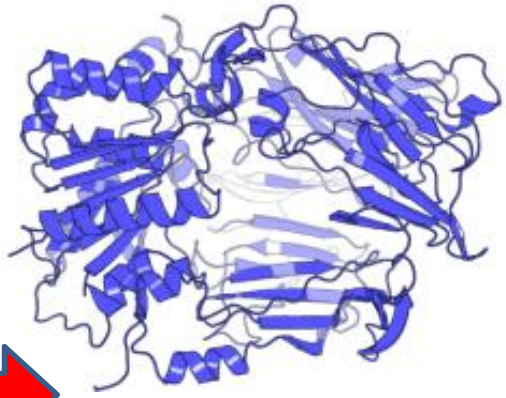
Три уровня суперкомпьютерного моделирования белков

Уровень 1: Биоинформатический анализ всей доступной информации;

Уровень 2: Автоматизированный анализ текстов ранее опубликованных исследований;

Уровень 3: Молекулярное моделирование для понимания механизмов действия

Изучение белков/ферментов на уровне суперсемейств

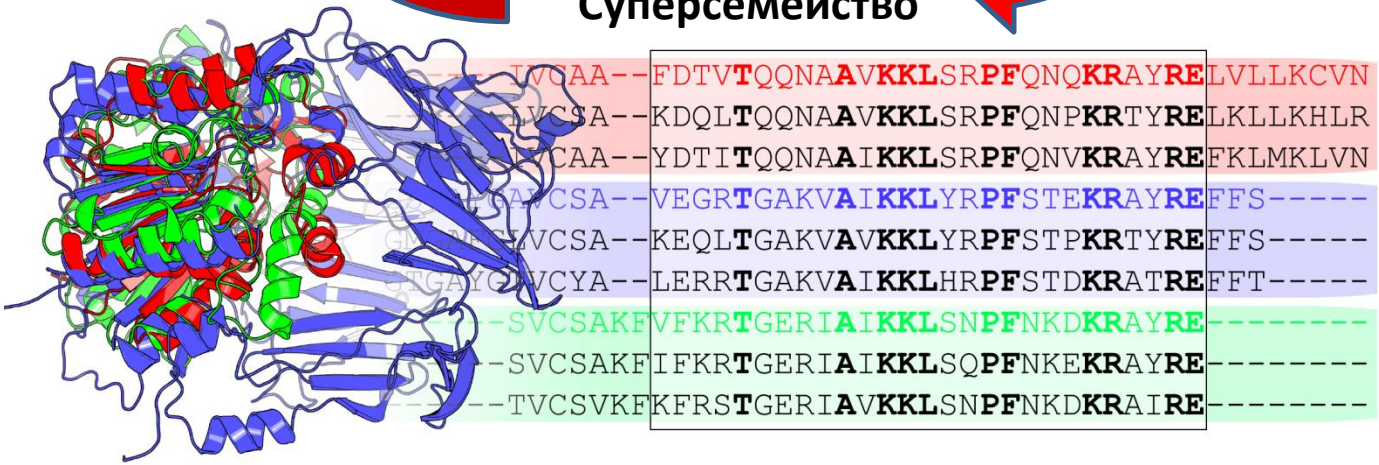


Белок

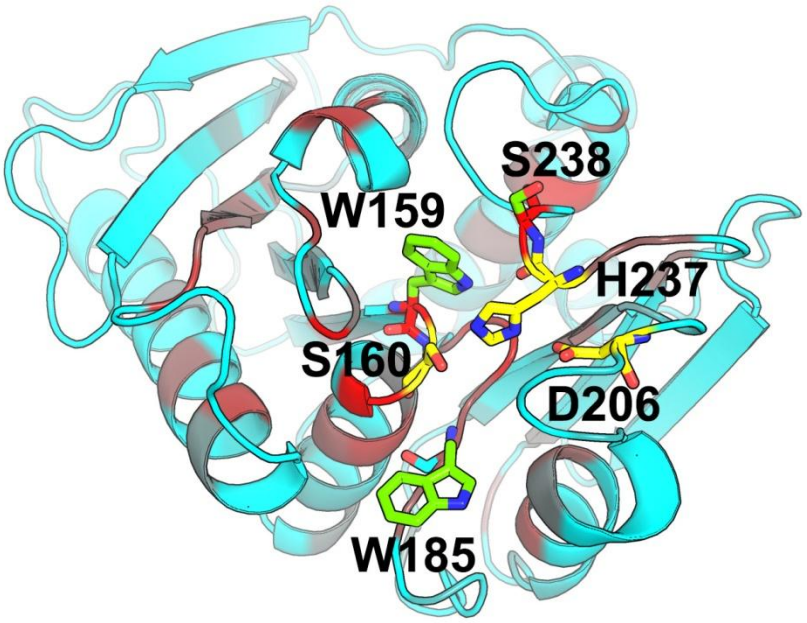
Изучение структурно-функциональных взаимосвязей

Сбор и анализ всей доступной информации о структурах и последовательностях родственных белков

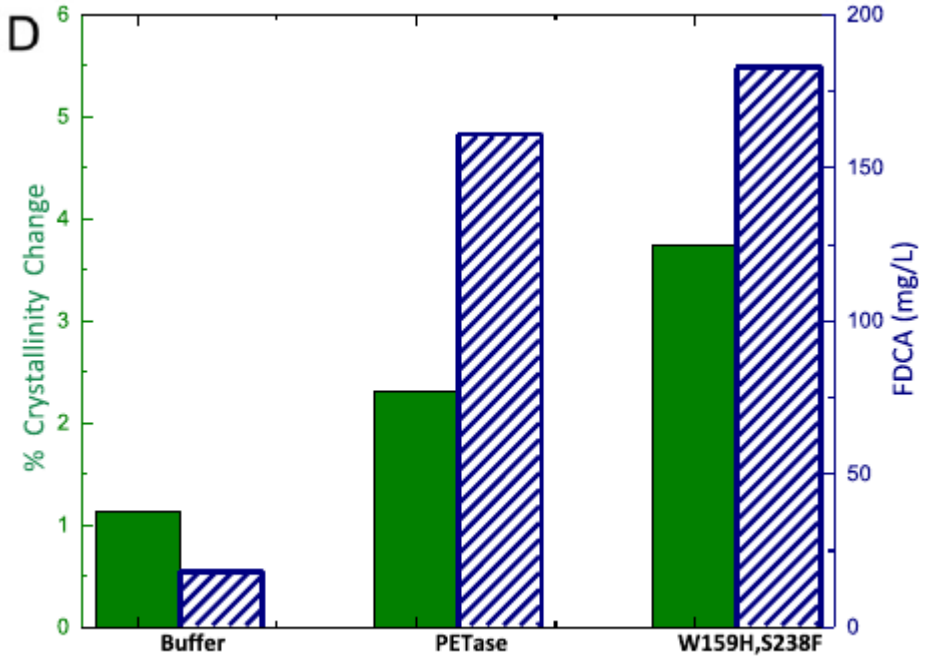
Суперсемейство



Изучение и инженерия ароматической полиэстеразы для переработки (деградации) пластика



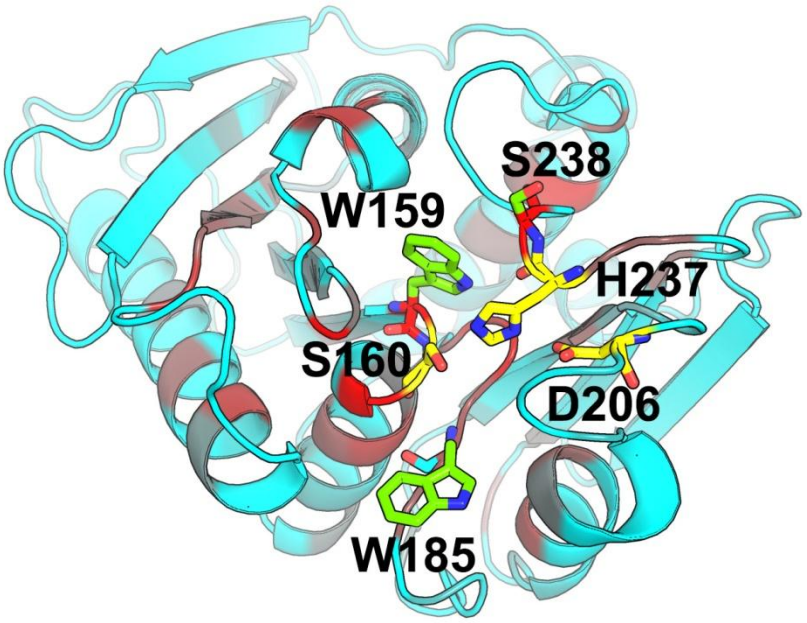
Mustguseal + Zebra bioinformatic analysis

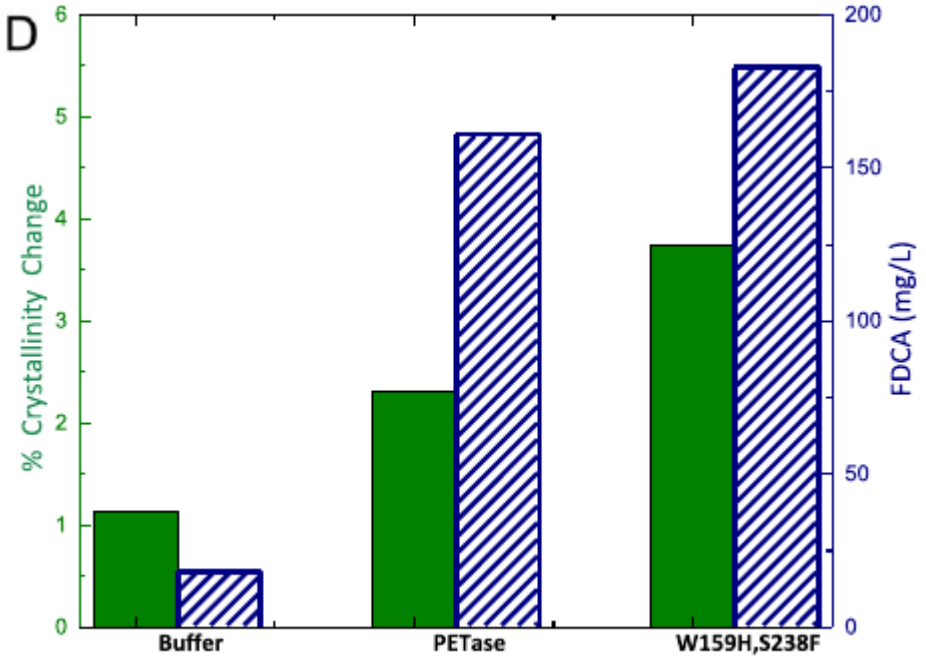


Austin et al., *PNAS* 2018

Rank	Residue	Family-1	Family-2	Family-3	Family-4	Family-5	Family-6	Family-7	Family-8	Family-9	Family-10
6	**S238**	G (50.2%) A (49.8%)	S (47.5%) E (21.1%)	**F (92.4%)**	L (96.1%)	Q (61.4%) G (12.3%)	L (97.8%)	A (94.7%)	G (78.8%) S (14.2%)	V (98.3%)	S (48.5%) G (30.6%) A (20.0%)
8	**W159**	F (86.5%) Y (13.0%)	F (99.8%)	F (72.5%) Y (18.2%)	**H (99.0%)**	Y (72.9%) F (27.0%)	A (87.4%) S (10.7%)	N (99.9%)	F (99.9%)	L (84.0%) Q (13.6%)	Y (61.4%) F (38.4%)
15	**W185**	R (40.9%) V (18.7%) G (12.6%)	Y (71.0%) W (25.6%)	P (85.2%) Q (8.8%)	P (98.8%)	M (91.7%) L (7.1%)	R (98.5%)	S (98.8%)	R (54.2%) A (21.9%) M (11.0%)	**A (60.6%)** S (19.6%) G (10.5%)	G (21.0%) A (19.4%) N (15.6%)

Изучение и инженерия ароматической полиэстеразы для переработки (деградации) пластика





Mustguseal + Zebra bioinformatic analysis

Austin et al., *PNAS* 2018

Rank	Residue	Family-1	Family-2	Family-3	Family-4	Family-5	Family-6	Family-7	Family-8	Family-9	Family-10
6	**S238**	G (50.2%) A (49.8%)	S (47.5%) E (21.1%)	**F (92.4%)**	L (96.1%)	Q (61.4%) G (12.3%)	G (78.8%) S (14.2%)				S (38.5%) A (30.6%) A (20.0%)
8	**W159**	F (86.5%) Y (13.0%)	F (99.8%)	F (72.5%) Y (18.2%)	**H (99.0%)**	Y (72.9%) F (27.0%)	A (87.4%) S (10.7%)			L (84.6%)	Y (61.4%) Q (13.6%) F (38.4%)
15	**W185**	R (40.9%) V (18.7%) G (12.6%)	Y (71.0%) W (25.6%)	P (85.2%) Q (8.8%)	P (98.8%)	M (91.7%) L (7.1%)	R (98.5%)	S (98.8%)	R (54.2%) A (21.9%) M (11.0%)	**A (60.6%)** S (19.6%) G (10.5%)	G (21.0%) A (19.4%) N (15.6%)

Т.е. аминокислотные остатки, внесение которых в соответствующие позиции приводит к улучшению полезных свойств PET-азы, **уже встречаются в суперсемействе**

Биоинформатический анализ суперсемейств

- Мутации/замены, которые нужно вводить в «наш» белок для достижения желаемого эффекта, уже могут присутствовать среди его гомологов (которые реализуют эти самые искомые свойства, просто их еще никто не изучал);

- Как извлекать эту информацию из всех данных о суперсемействе?

Изучение белков/ферментов на уровне суперсемейств

Суперсемейство

- **Общая структурная организация и сходство базовых функций** дают основание предполагать общее эволюционное происхождение
- Белки одного суперсемейства произошли от очень далекого общего предка и характеризуются **значительным функциональным разнообразием**

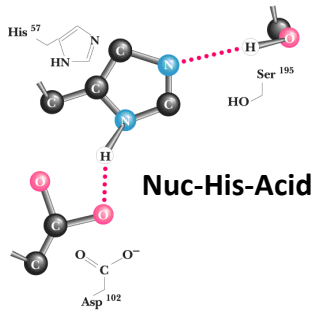
Семейство

- **Эволюционно близкие белки** с высокой степенью сходства по аминокислотной последовательности, что подразумевает наличие близкого общего предка
- **Менее выраженное разнообразие свойств**: как правило обладают общим механизмом реакции, но могут превращать различные субстраты

Под-семейство

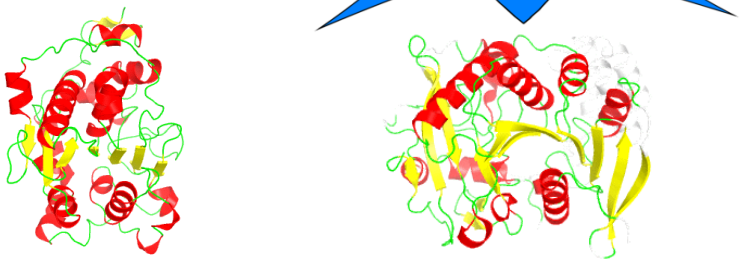
- **Еще более близкие белки**, характеризуемые высокой степенью сходства по первичной и третичной структуре, и **гомогенностью функции**





Nuc-His-Acid

α/β-гидролазы



Липазы

Эпоксид-гидролазы



Пептидазы

Построение множественного выравнивания суперсемейства белков

«Затравка»
(например, наиболее изученный белок суперсемейства)

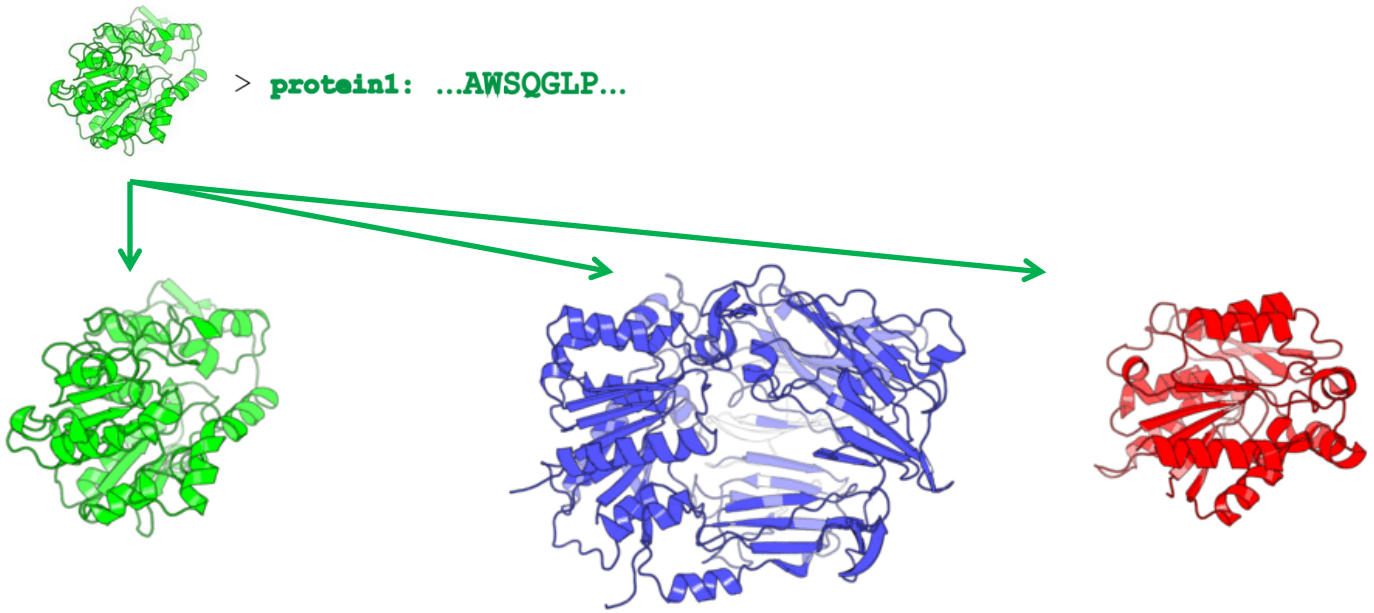


> protein1: ...AWSQGLP...

Построение множественного выравнивания суперсемейства белков

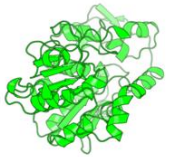
«Затравка»
(например, наиболее изученный белок суперсемейства)

Поиск по структурному сходству
Построение выборки ***эволюционно удаленных*** родственных белков на основании поиска в публичных базах данных



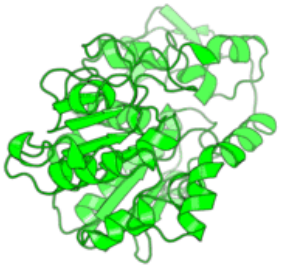
Построение множественного выравнивания суперсемейства белков

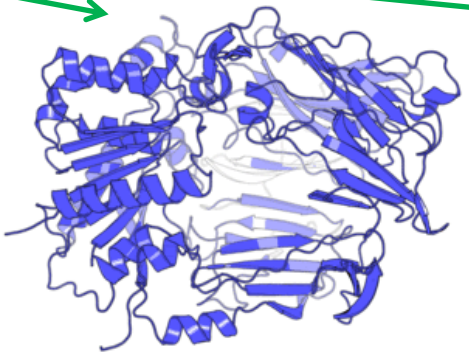
«Затравка»
(например, наиболее изученный белок суперсемейства)



> protein1: ...AWSQGLP...

Поиск по структурному сходству
Построение выборки ***эволюционно удаленных*** родственных белков на основании поиска в публичных базах данных



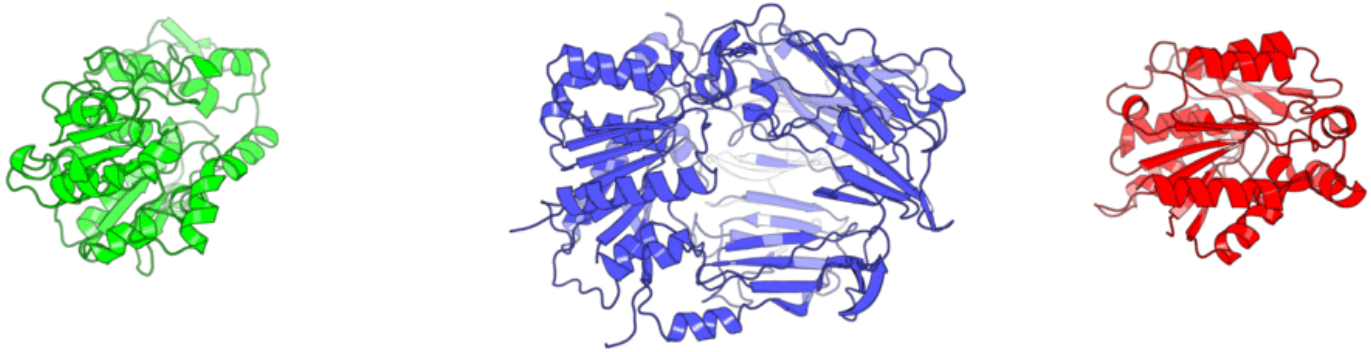




Поиск по сходству последовательностей
Построение выборки ***эволюционно близких*** белков на основании поиска в публичных базах данных

> protein1: ...AWSQGLP...	> protein1: ...DGTMFDP...	> protein1: ...KCDP...
> protein2: ...ASSQGAP...	> protein2: ...NATDP...	> protein2: ...RCDP...
> protein3: ...LSNLDP...	> protein3: ...DATMYDP...	> protein3: ...RCDP...
> protein4: ...WGSNAP...	> protein4: ...DGTEP...	> protein4: ...KCEP...
> protein5: ...LASNLP...	> protein5: ...NATMFDP...	> protein5: ...KCDP...

Построение множественного выравнивания суперсемейства белков



Выравнивание последовательностей

Сравнение «алфавита» аминокислот с использованием эмпирических моделей аминокислотных замен

```
> protein1: ...AWSQGLP...
> protein2: ...ASSQGAP...
> protein3: ...L-SNLDP...
> protein4: ...WGSNA-P...
> protein5: ...LASNL-P...
```

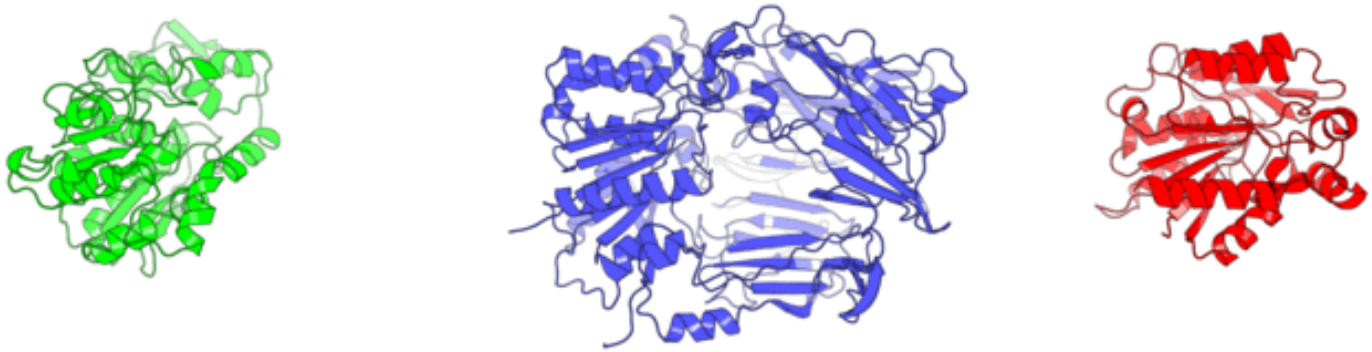
```
> protein1: ...DGTMFDP...
> protein2: ...NAT--DP...
> protein3: ...DATMYDP...
> protein4: ...DGT--EP...
> protein5: ...NATMFDP...
```

```
> protein1: ...KCDP...
> protein2: ...RCDP...
> protein3: ...RCDP...
> protein4: ...KCEP...
> protein5: ...KCDP...
```

Построение множественного выравнивания суперсемейства белков

Выравнивание структур

Поиск оптимального наложения координат атомов в трехмерном пространстве



Выравнивание последовательностей

Сравнение «алфавита» аминокислот с использованием эмпирических моделей аминокислотных замен

```
> protein1: ...AWSQGLP...
> protein2: ...ASSQGAP...
> protein3: ...L-SNLDP...
> protein4: ...WGSNA-P...
> protein5: ...LASNL-P...
```

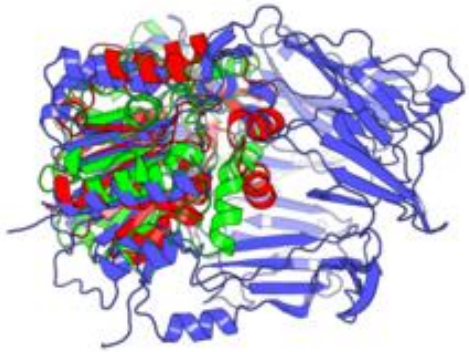
```
> protein1: ...DGTMFDP...
> protein2: ...NAT--DP...
> protein3: ...DATMYDP...
> protein4: ...DGT--EP...
> protein5: ...NATMFDP...
```

```
> protein1: ...KCDP...
> protein2: ...RCDP...
> protein3: ...RCDP...
> protein4: ...KCEP...
> protein5: ...KCDP...
```

Построение множественного выравнивания суперсемейства белков

Выравнивание структур

Поиск оптимального наложения координат атомов в трехмерном пространстве



Выравнивание последовательностей

Сравнение «алфавита» аминокислот с использованием эмпирических моделей аминокислотных замен

```
> protein1: ...AWSQGLP...
> protein2: ...ASSQGAP...
> protein3: ...L-SNLDP...
> protein4: ...WGSNA-P...
> protein5: ...LASNL-P...
```

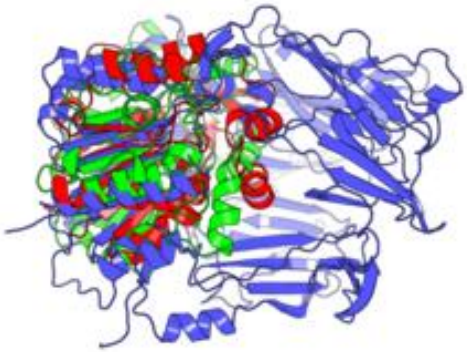
```
> protein1: ...DGTMFDP...
> protein2: ...NAT--DP...
> protein3: ...DATMYDP...
> protein4: ...DGT--EP...
> protein5: ...NATMFDP...
```

```
> protein1: ...KCDP...
> protein2: ...RCDP...
> protein3: ...RCDP...
> protein4: ...KCEP...
> protein5: ...KCDP...
```

Построение множественного выравнивания суперсемейства белков

Структурно-опосредованное множественное выравнивание аминокислотных последовательностей

Эволюционно удаленные белки, которые потеряли сходство по последовательностям в процессе естественного отбора и специализации от общего предка, сравниваются посредством выравнивания структур, в то время как выравнивание аминокислотных последовательней дает полезный результат в случае близких гомологов



```
> protein1: ...AWSQGLP...
> protein2: ...ASSQGAP...
> protein3: ...L-SNLDP...
> protein4: ...WGSNA-P...
> protein5: ...LASNL-P...
```

```
> protein1: ...DGTMFDP...
> protein2: ...NAT--DP...
> protein3: ...DATMYDP...
> protein4: ...DGT--EP...
> protein5: ...NATMFDP...
```

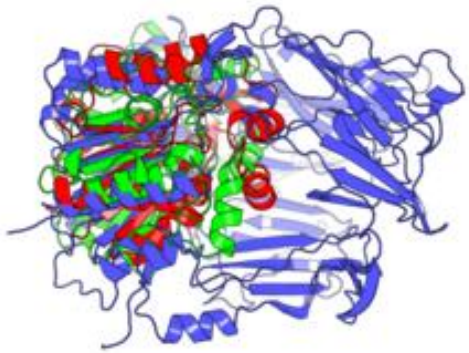
```
> protein1: ...KCDP...
> protein2: ...RCDP...
> protein3: ...RCDP...
> protein4: ...KCEP...
> protein5: ...KCDP...
```

Построение множественного выравнивания суперсемейства белков

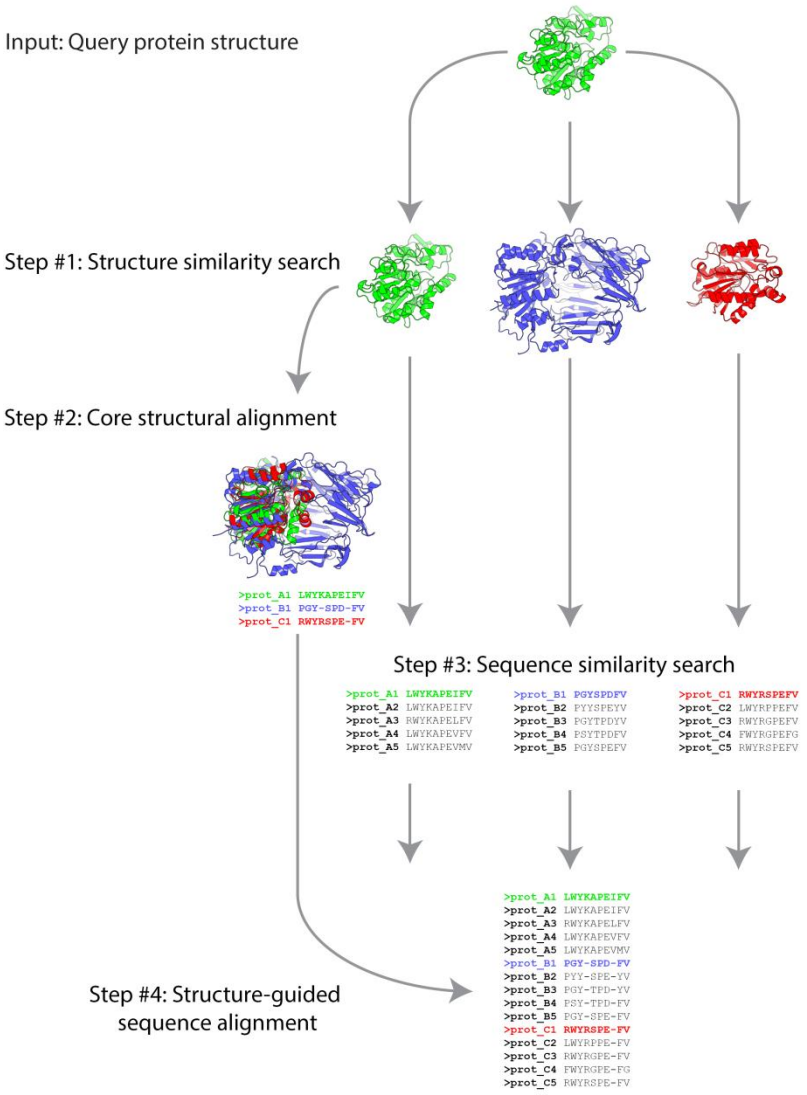
Структурно-опосредованное множественное выравнивание

Эволюционно удаленные белки, которые потеряли сходство по последовательностям в процессе естественного отбора и специализации от общего предка, сравниваются посредством выравнивания структур, в то время как выравнивание аминокислотных последовательней дает полезный результат в случае близких гомологов

```
> protein1: ...A-WSQGLP...
> protein2: ...A-SSQGAP...
> protein3: ...L--SNLDP...
> protein4: ...W-GSNA-P...
> protein5: ...L-ASNL-P...
> protein1: ...DG-TMFDP...
> protein2: ...NA-T--DP...
> protein3: ...DA-TMYDP...
> protein4: ...DG-T--EP...
> protein5: ...NA-TMFDP...
> protein1: ...K--C--DP...
> protein2: ...R--C--DP...
> protein3: ...R--C--DP...
> protein4: ...K--C--EP...
> protein5: ...K--C--DP...
```



Построение множественного выравнивания суперсемейства белков



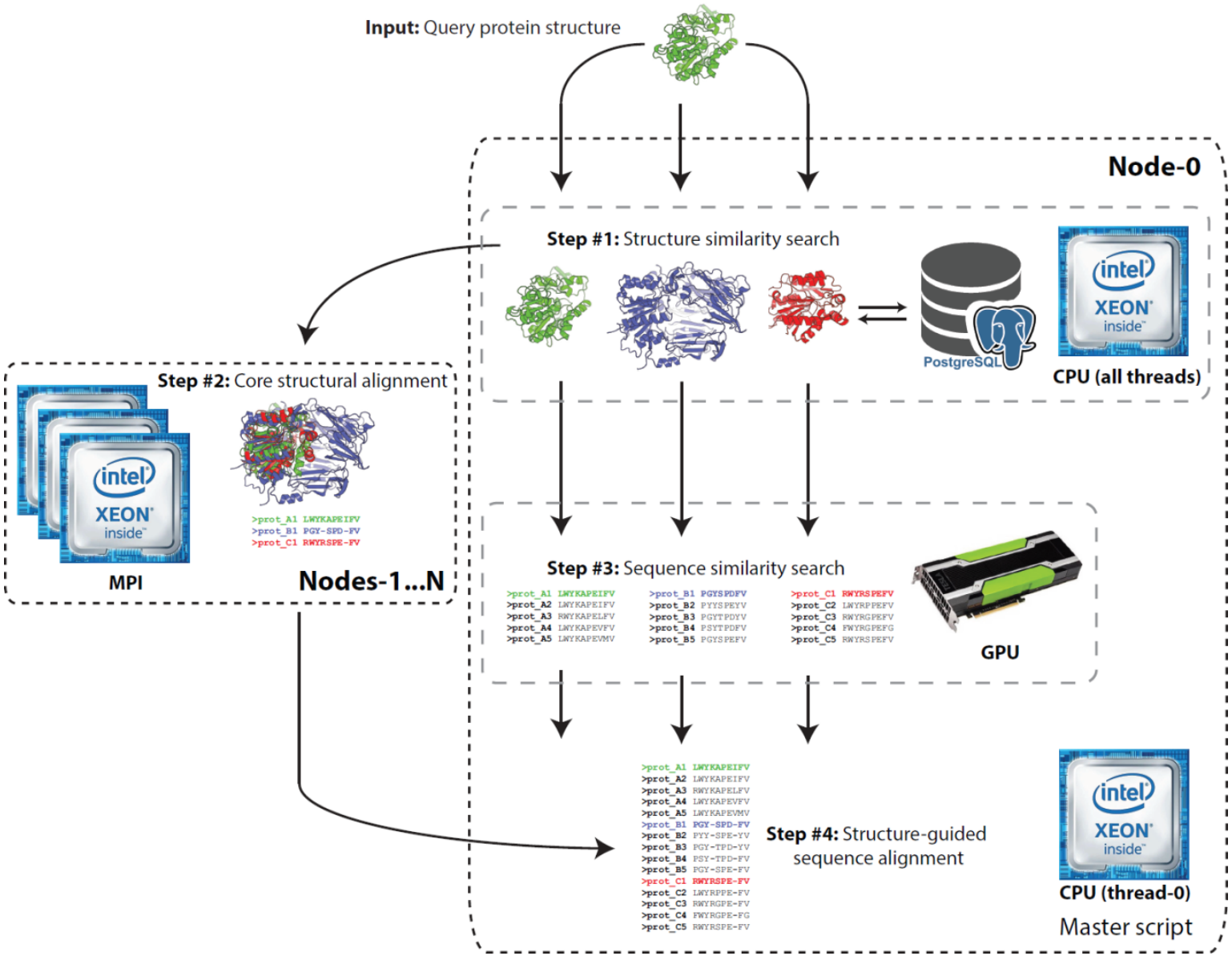
https://biokinet.belozersky.msu.ru/mustguseal

Конвейер из последовательных и независимых в программном отношении этапов, которые примерно равнозначных по вычислительной трудоемкости;

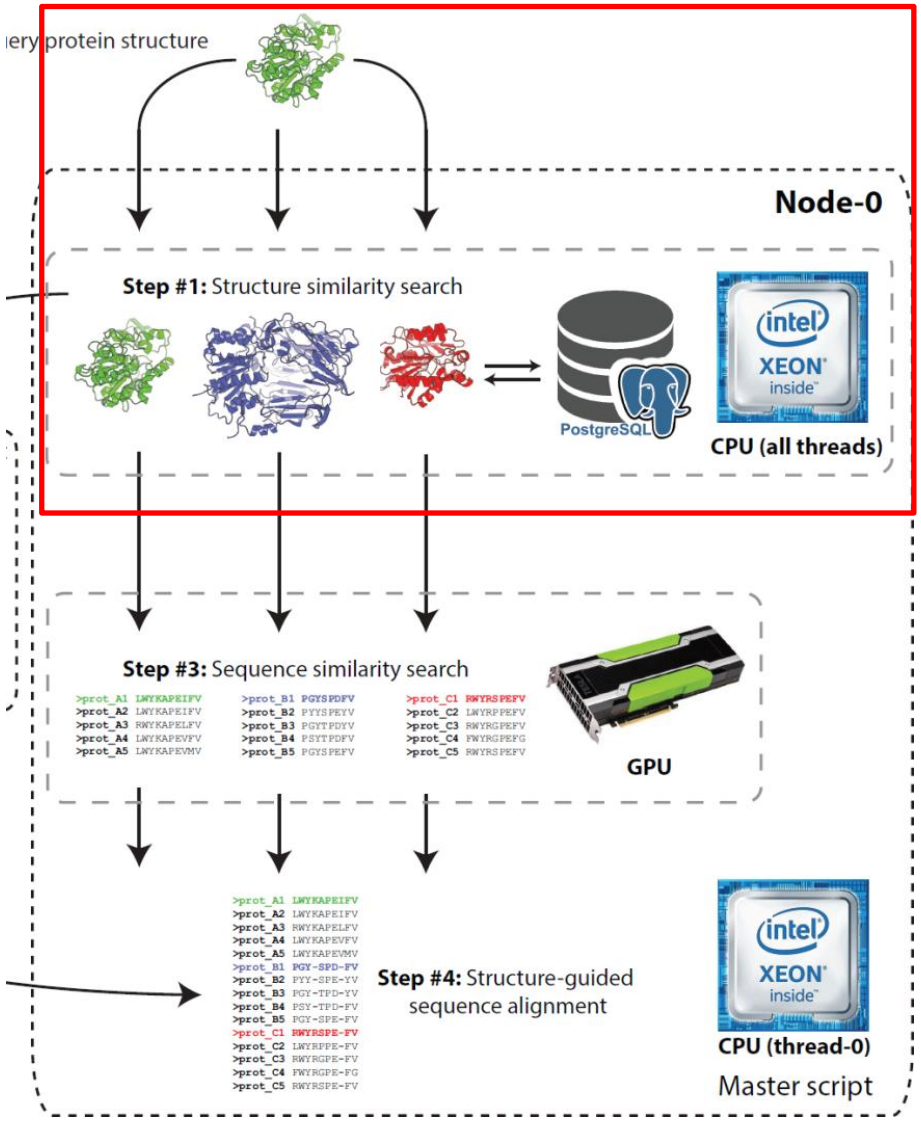
Для эффективной работы нужно оптимизировать все стадии единого комплексного решения;

Suplatov D.A., Kopylov K.E., Popova N.N., Voevodin V.V., Švedas V.K. (2018) Mustguseal: a Server for Multiple Structure-Guided Sequence Alignment of Protein Families. *Bioinformatics*, 34(9), 1583-1585

Гибридный подход к построению множественного выравнивания суперсемейства белков



Построение множественного выравнивания суперсемейства белков

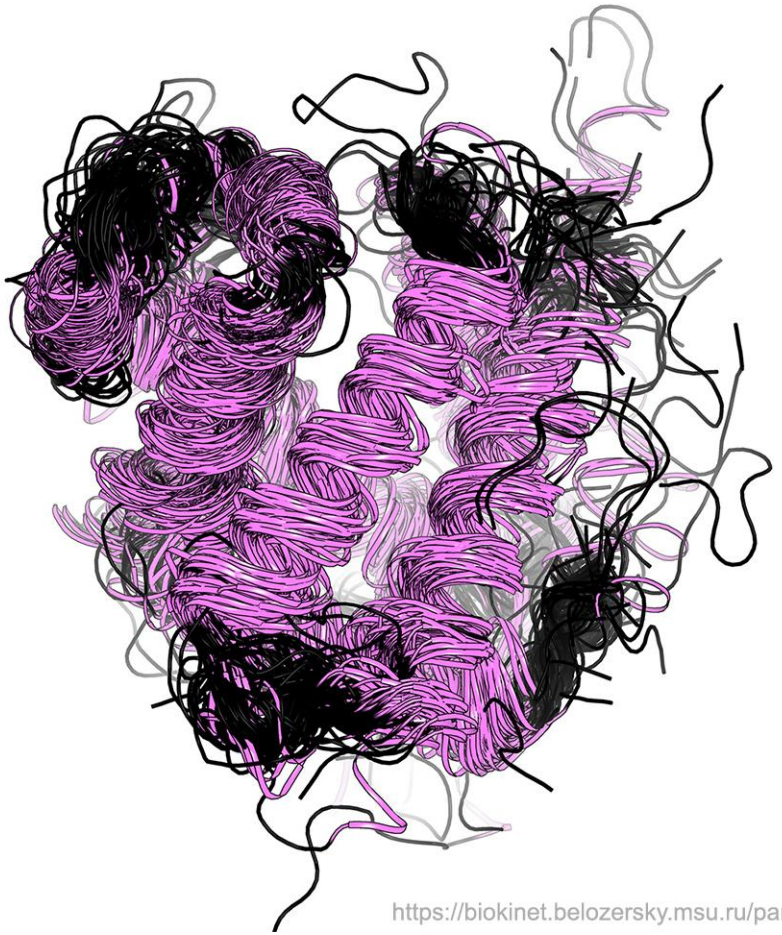


Этап 1: Поиск по структурному сходству

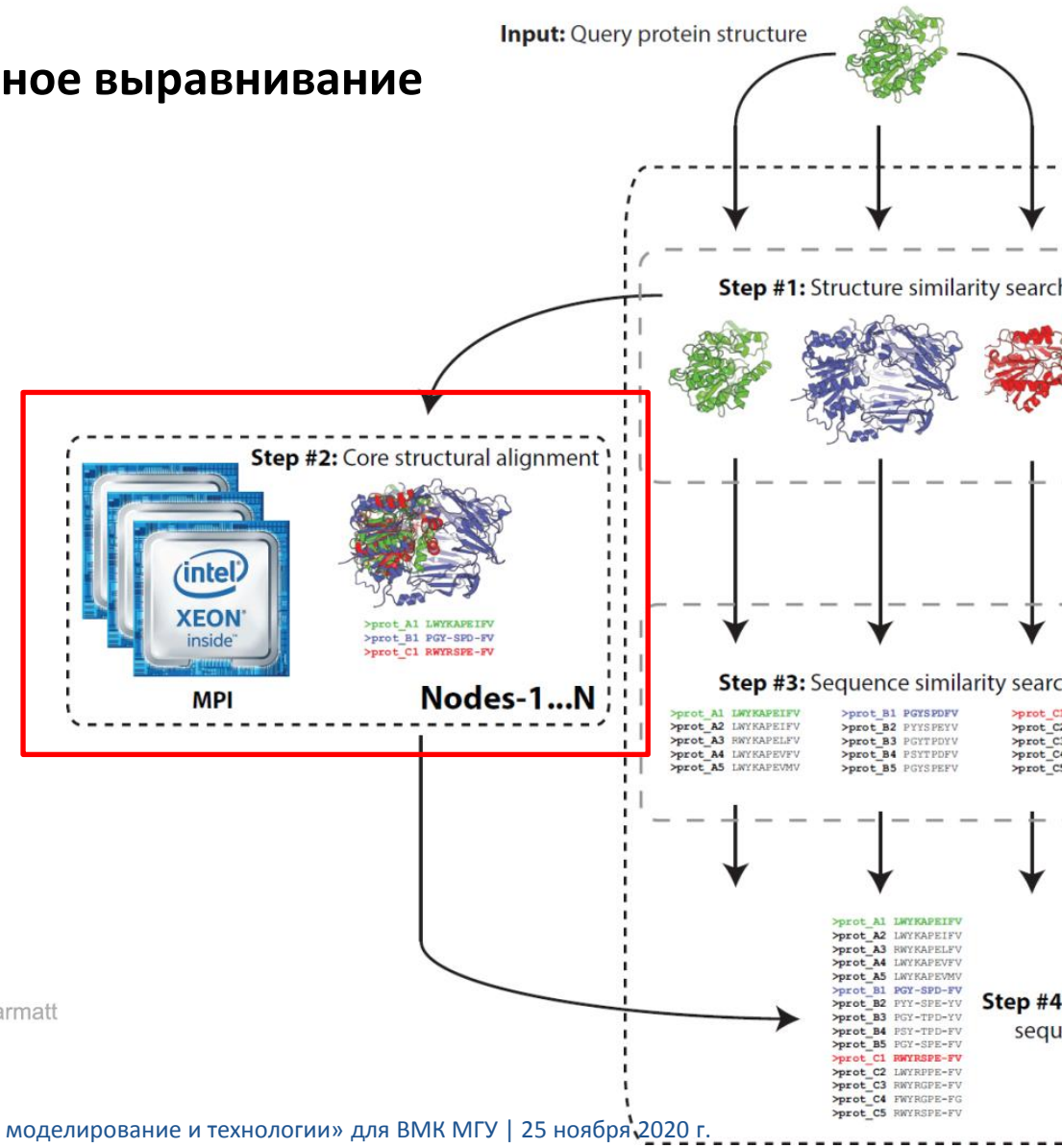
- Поиск в базе данных PDB ~330 тыс. структур отдельных цепей;
- Задача поиска в базе разделяется на ~30000 – 330000 подзадач, которые полностью независимы друг от друга;
- Решается через распараллеливание в рамках общей памяти. За счет препроцессинга базы данных и удаления заведомо непохожих структур из поиска задача может быть решена за ~10-60 минут на Intel Xeon Gold 6126 CPU,
- Однажды посчитанные парные выравнивания сохраняются в базе данных на основе PostGreSQL для повторного использования. Повторное использование ранее посчитанных результатов может сократить время этапа до нескольких секунд.

Построение множественного выравнивания суперсемейства белков

Этап 2: Множественное структурное выравнивание



https://biokinet.belozersky.msu.ru/parmatt



Построение множественного выравнивания суперсемейства белков

Этап 2: Множественное структурное выравнивание

- Все существующие программы для построения множественного 3D-выравнивания белков способны использовать только одно ядро CPU (в лучшем случае – частично ускоряются в режиме OpenMP);

- Это связано с устаревающим представление о том, что информации о 3D-структур белков «мало».

Построение множественного выравнивания суперсемейства белков

Этап 2: Множественное структурное выравнивание

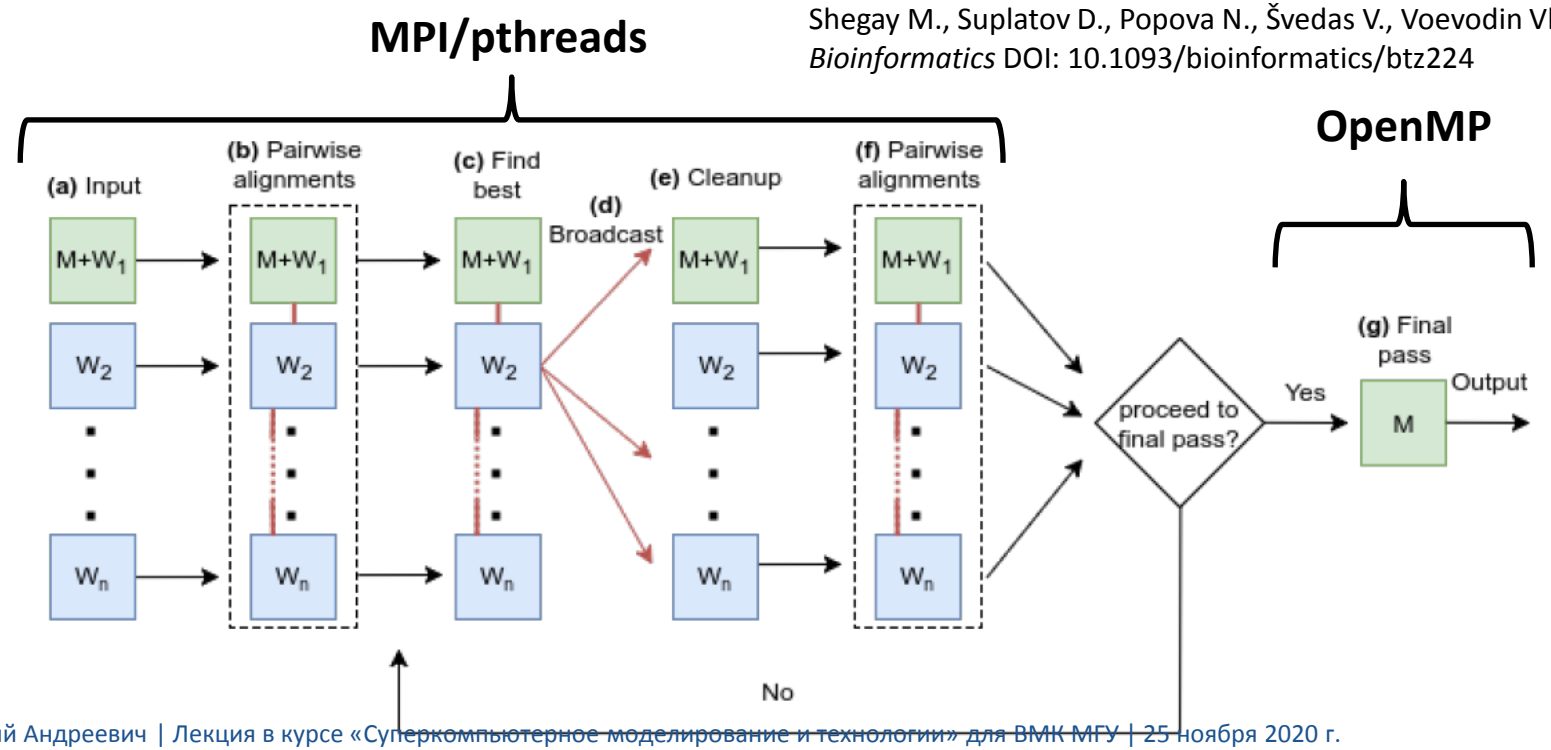
- Когда-то информации о 3D-структурах белков действительно было мало, но теперь ее доступность в базе PDB растет почти в геометрической прогрессии;
- Сегодня суперсемейства белков могут быть представлены тысячами структур, и сравнительный анализ такого объема 3D-данных на одном CPU не всегда возможен;
- Очевидно, что уже в ближайшем будущем использование одного CPU для множественного 3D-выравнивания структур белков станет невозможным.



parMATT – первая программа для построения множественных 3D-выравниваний белков на суперкомпьютере

- **parMATT** - это гибридная параллельная ре-имплементация алгоритма MATT с использованием MPI/pthreads/OpenMP; Наиболее сложные в вычислительном отношении этапы алгоритма MATT - первоначальное построение парных выравниваний между всеми входными структурами и последующее прогрессивное множественное выравнивание – были распараллелены с использованием MPI и pthreads, а заключительный этап уточнения был оптимизирован путем введения поддержки OpenMP;

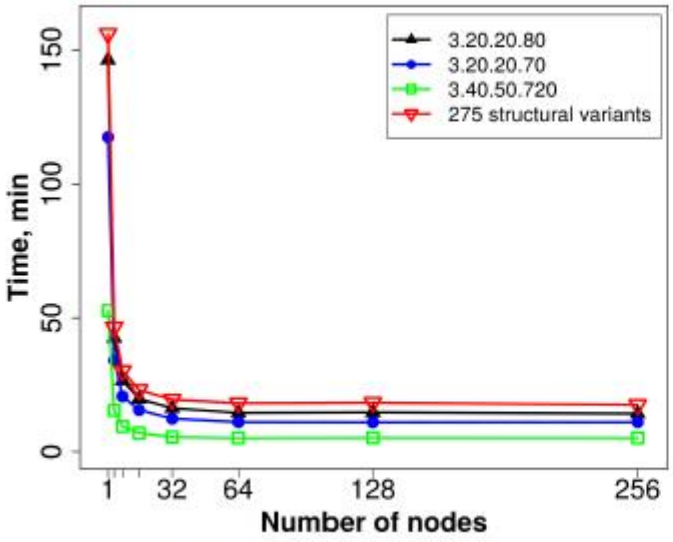
Shegay M., Suplatov D., Popova N., Švedas V., Voevodin Vl. (2019) *Bioinformatics* DOI: 10.1093/bioinformatics/btz224



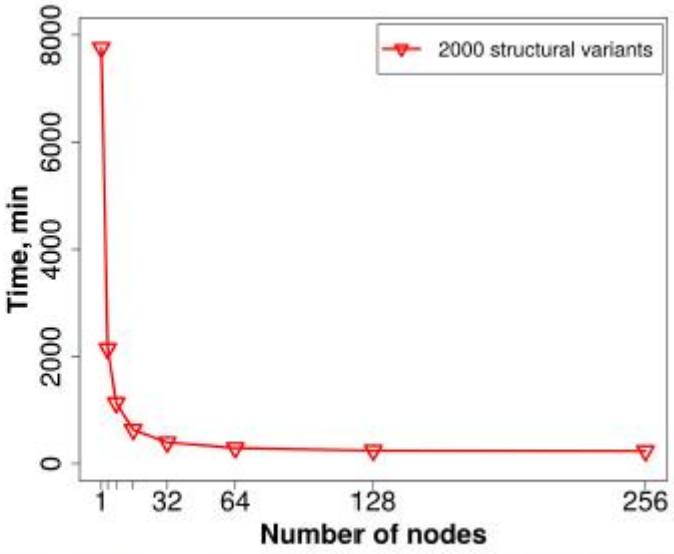
parMATT – первая программа для построения множественных 3D-выравниваний белков на суперкомпьютере

- parMATT может работать быстрее, чем исходный MATT, даже на одном многоядерном процессоре, но главное свойство новой программы – принципиальное ускорение при использовании классических кластеров и суперкомпьютеров).
- Ускорение и «точка насыщения» зависят от свойств входных данных – в первую очередь от числа и размера 3D-структур;
- Использование parMATT позволяет ускорить выравнивание выборки из 2000 структур в ~32 раза на 128 узлах суперкомпьютера «Ломоносов-1» (с 6 дней до 4 часов)

Shegay M., Suplatov D., Popova N., Švedas V., Voevodin Vl. (2019) *Bioinformatics* DOI: 10.1093/bioinformatics/btz224



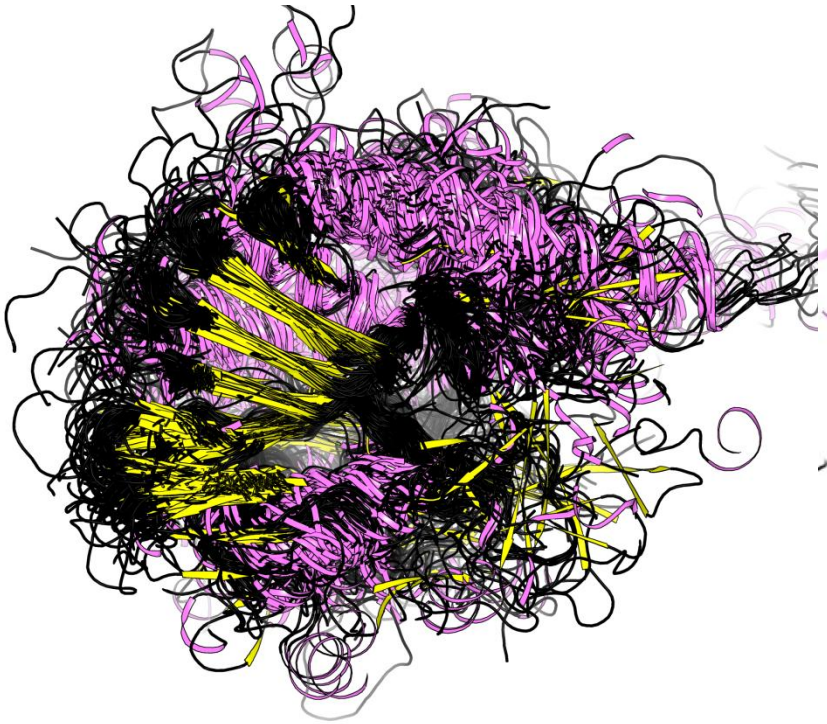
(A) Larger CATH sets



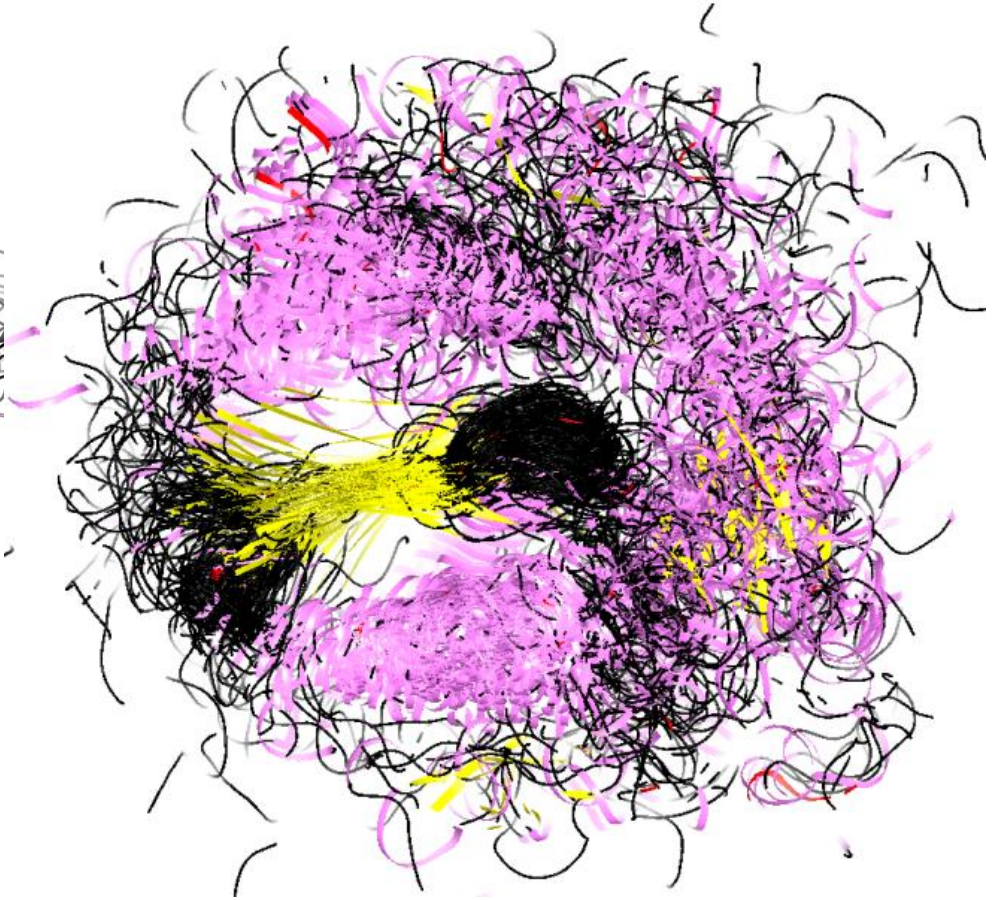
(B) MD-generated ensemble of structural variants

Построение множественного выравнивания суперсемейства белков

Этап 2: Множественное структурное выравнивание



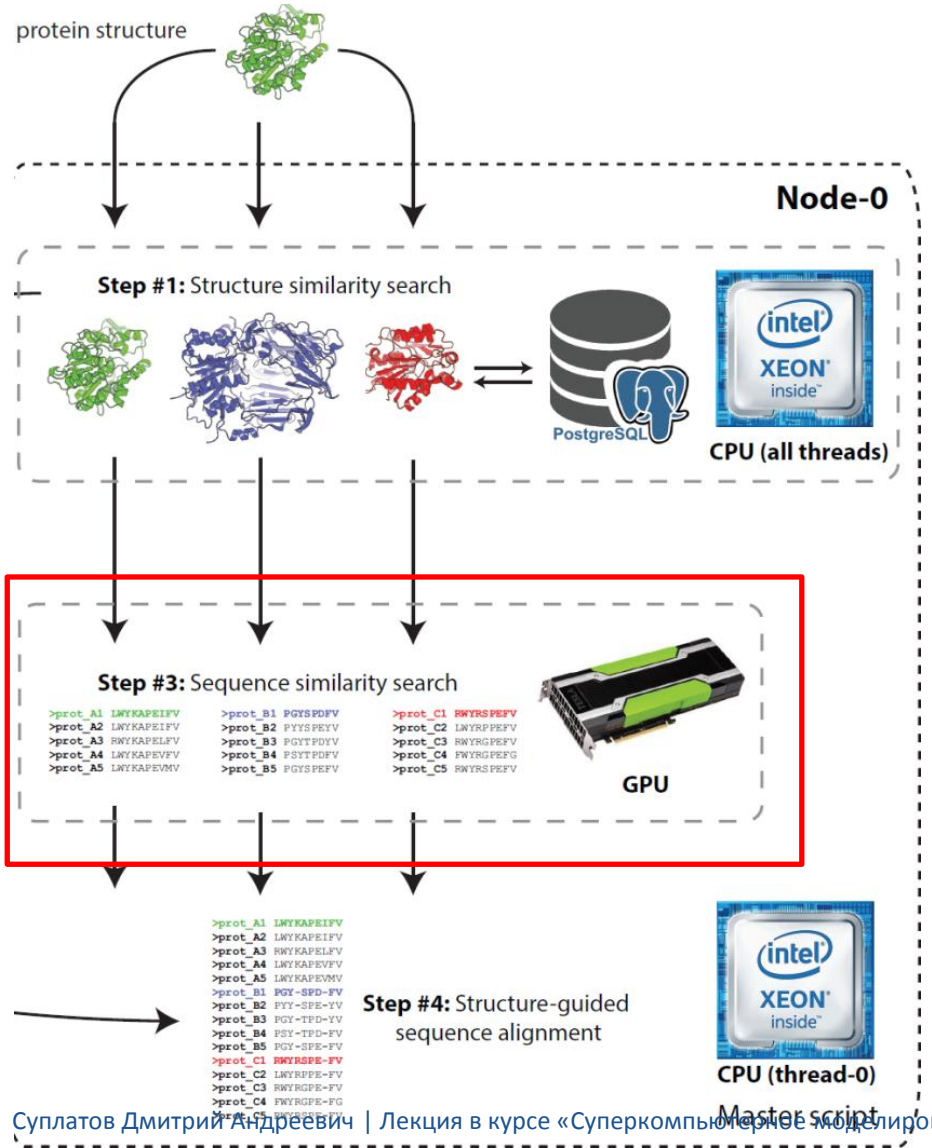
Выравнивание структур родственных ГАФД и ГАФД-подобных белков



Выравнивание структур родственных белков суперсемейства α/β-гидродаз

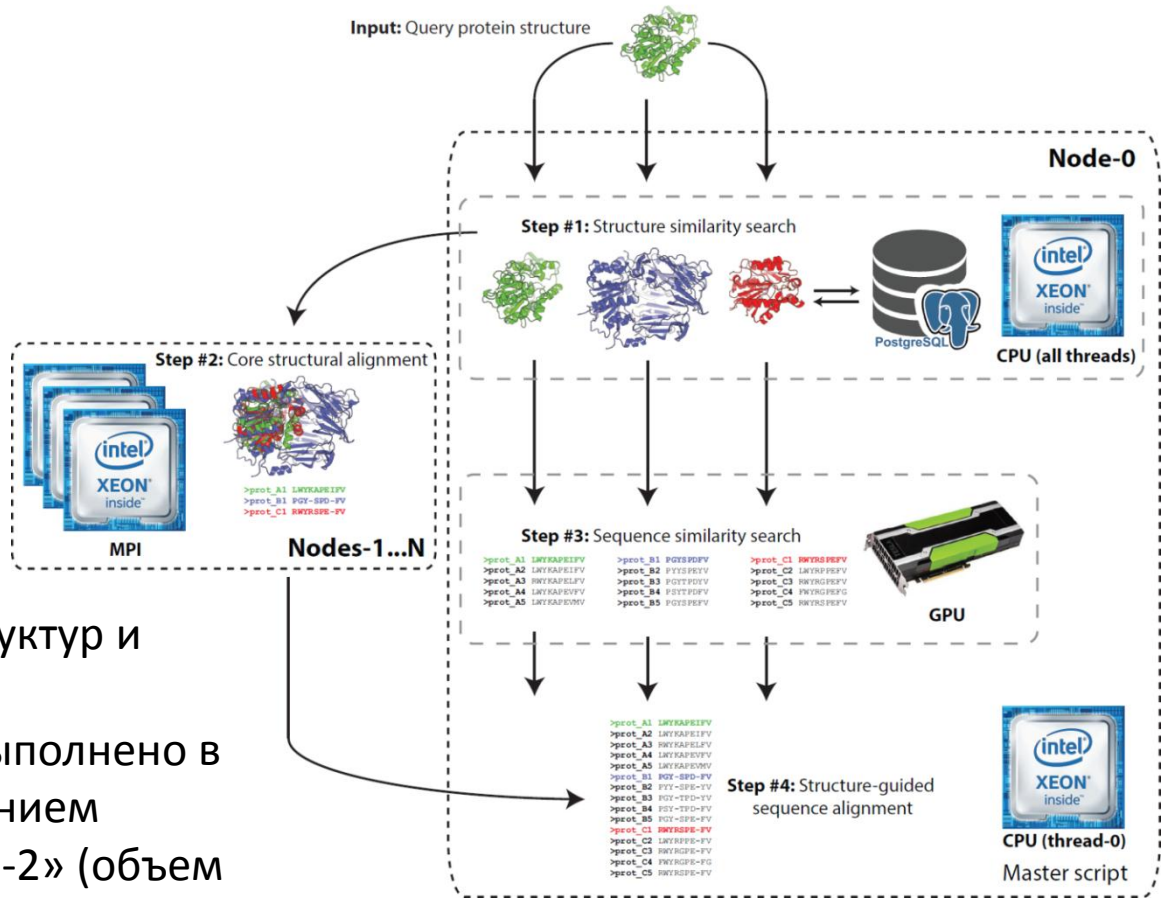
Построение множественного выравнивания суперсемейства белков

Этап 3: Поиск по сходству и выравнивание аминокислотных последовательностей



- Размер банка данных UniProtKB в мае 2019 года составляет ~156’000’000 последовательностей (увеличился в два раза за два года);
- Используется **GPU-версия** алгоритма BLAST;
- На одном узле раздела “Pascal” (12-ядерный CPU и 2 GPU) может быть одновременно запущено 12 параллельных BLAST поисков без потери скорости расчета.

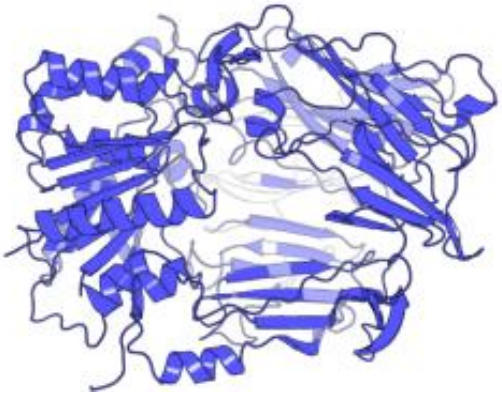
Гибридный подход к построению множественного выравнивания суперсемейства белков



Выравнивание сотен тысяч структур и последовательностей белков суперсемейства может быть выполнено в течение 12 часов с использованием суперкомпьютера «Ломоносов-2» (объем ресурсов зависит от характеристик входных данных)

Dmitry Suplatov, Yana Sharapova, Maxim Shegay, Nina Popova, Kateryna Fesko, Vladimir Voevodin and Vytas Švedas (2019) High-Performance Hybrid Computing for Bioinformatic Analysis of Protein Superfamilies – включено в предварительную программу конференции RuSCDays'19

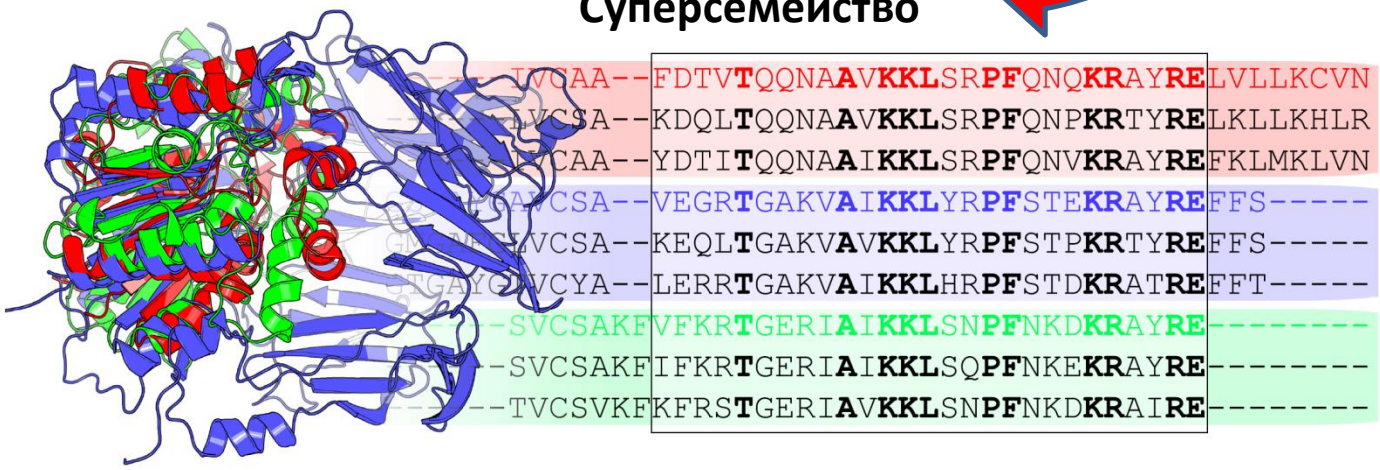
Биоинформатический анализ суперсемейств



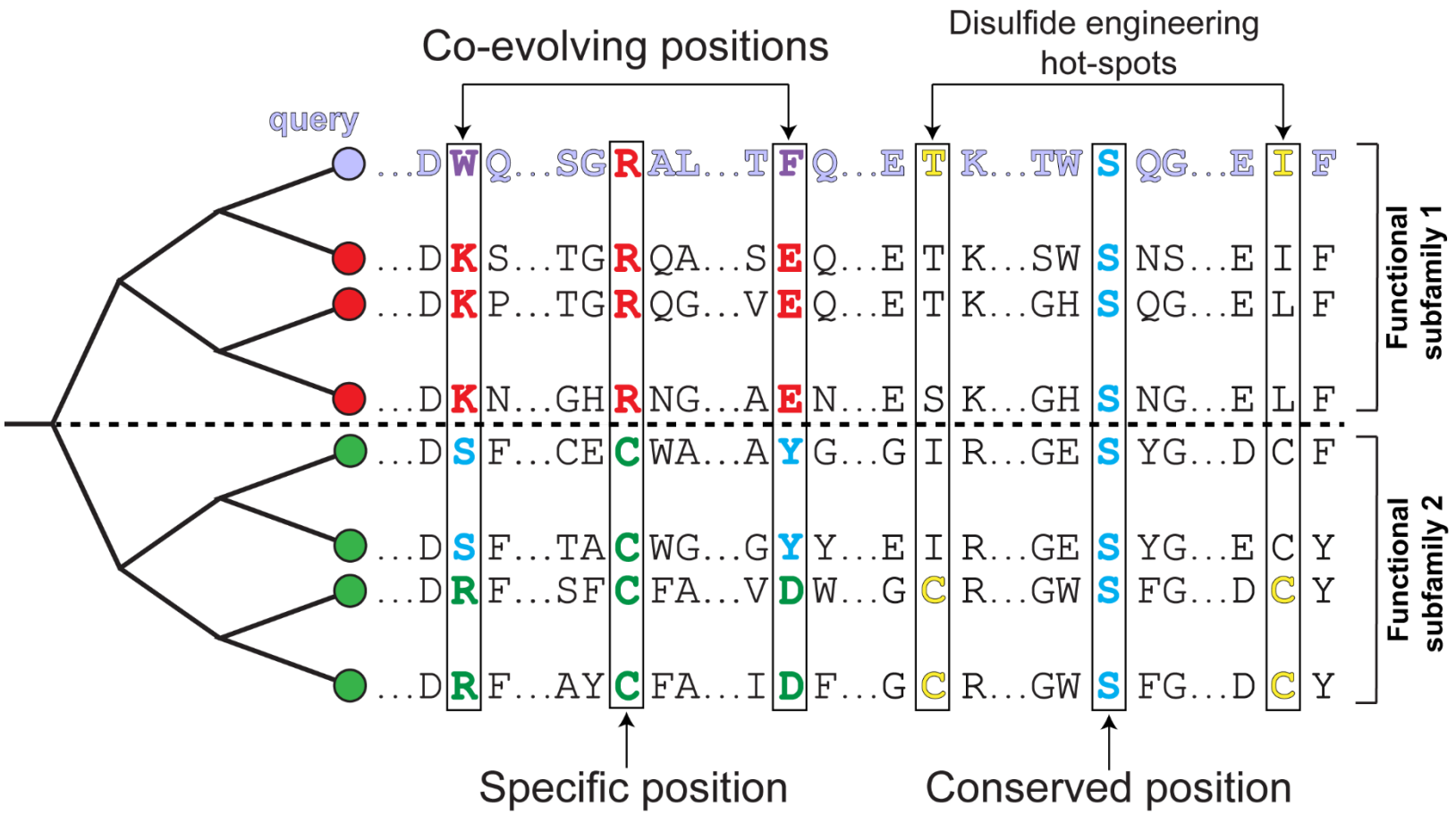
Белок

Выравнивание построено!
Что дальше?

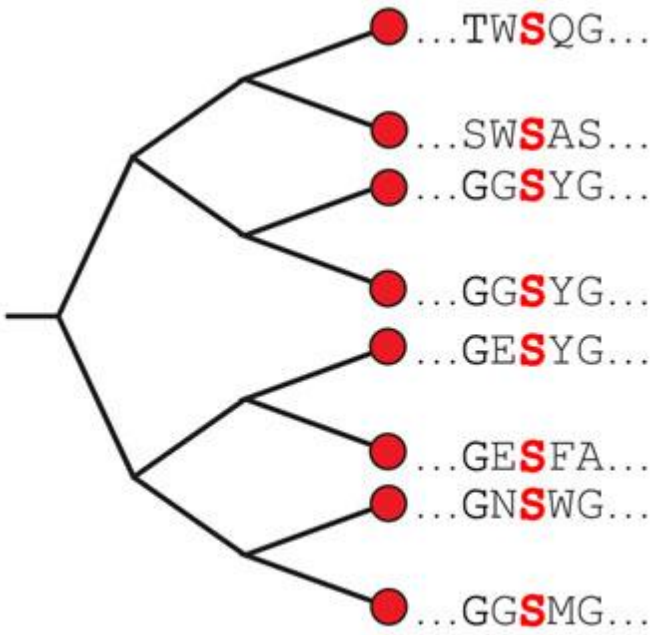
Суперсемейство



Вариабельность аминокислот в участке структуры фермента и его гомологов в суперсемействе помогает понять роль остатков в функции и регуляции



Консервативные позиции

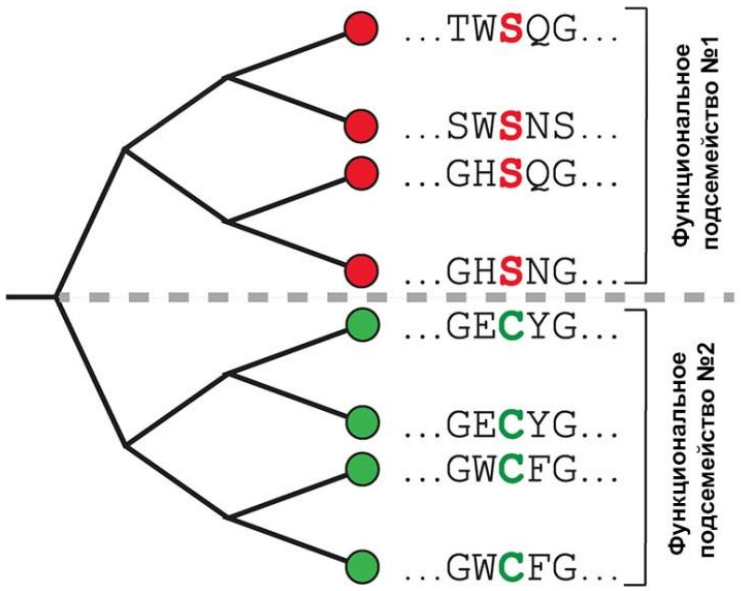


Консервативные позиции

Консервативные позиции в большом функционально разнообразоном суперсемействе – **детерминанты общих/ключевых свойств**, **важных для всех белков** с общей структурой (например, непосредственно участвующих в каталитическом превращении субстратов)

Пожалуй, **не стоит вносить изменения/мутации в такие позиции**, поскольку это может привести к полной потере функционирования фермента

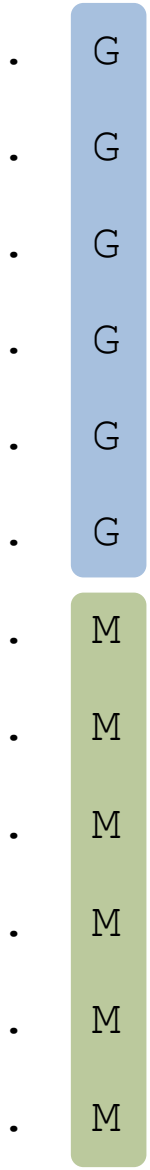
Специфические позиции



Специфические позиции подсемейств

Такие позиции, которые консервативны только между отдельными семействами/подсемействами, но различающиеся между ними, могут являться **детерминантами функционального разнообразия** между гомологами и могут быть использованы в качестве **«горячих точек замен»** для внесения мутаций

Специфические позиции



1. Информацию о вариабельных, эволюционно «гибких» **позициях**, которые отвечают за функциональное разнообразие между гомологами

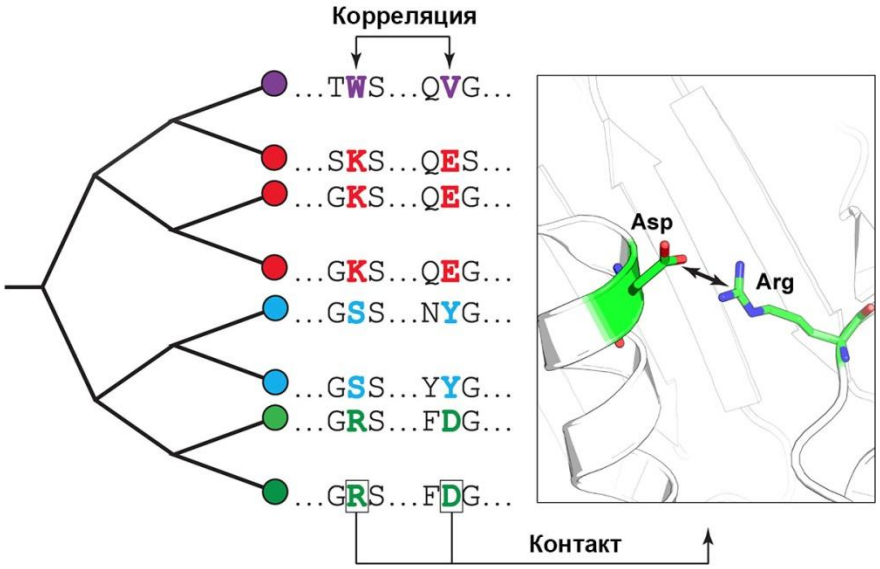
2. Предложения о **заменах**, необходимых для переключения функции между подсемействами (G ↔ M)

Биоинформатический анализ дает нам **воспроизводимую** и **экспериментально проверяемую** гипотезу для последующего изучения

- **Консервативные** и **специфические позиции** могут помочь понять, как фермент выполняет присущую ему функцию;
- **Консервативные позиции** обозначают сайты, в которых в ходе эволюции не допускались изменения, поскольку только определенные аминокислотные остатки были способны обеспечить необходимую структуру и функцию. Их следует исключить из списка потенциальных сайтов мутации для белковой инженерии, поскольку их замена, как правило, снижает каталитическую активность;
- **Специфические позиции**, как детерминанты функционального разнообразия в суперсемействе, могут быть выбраны в качестве горячих точек для мутации в попытке улучшить белок дикого типа для конкретной цели.

Suplatov et al. (2020) *Nucleic Acids Research* 10.1093/nar/gkaa276

Коррелирующие (ко-эволюционирующие) позиции

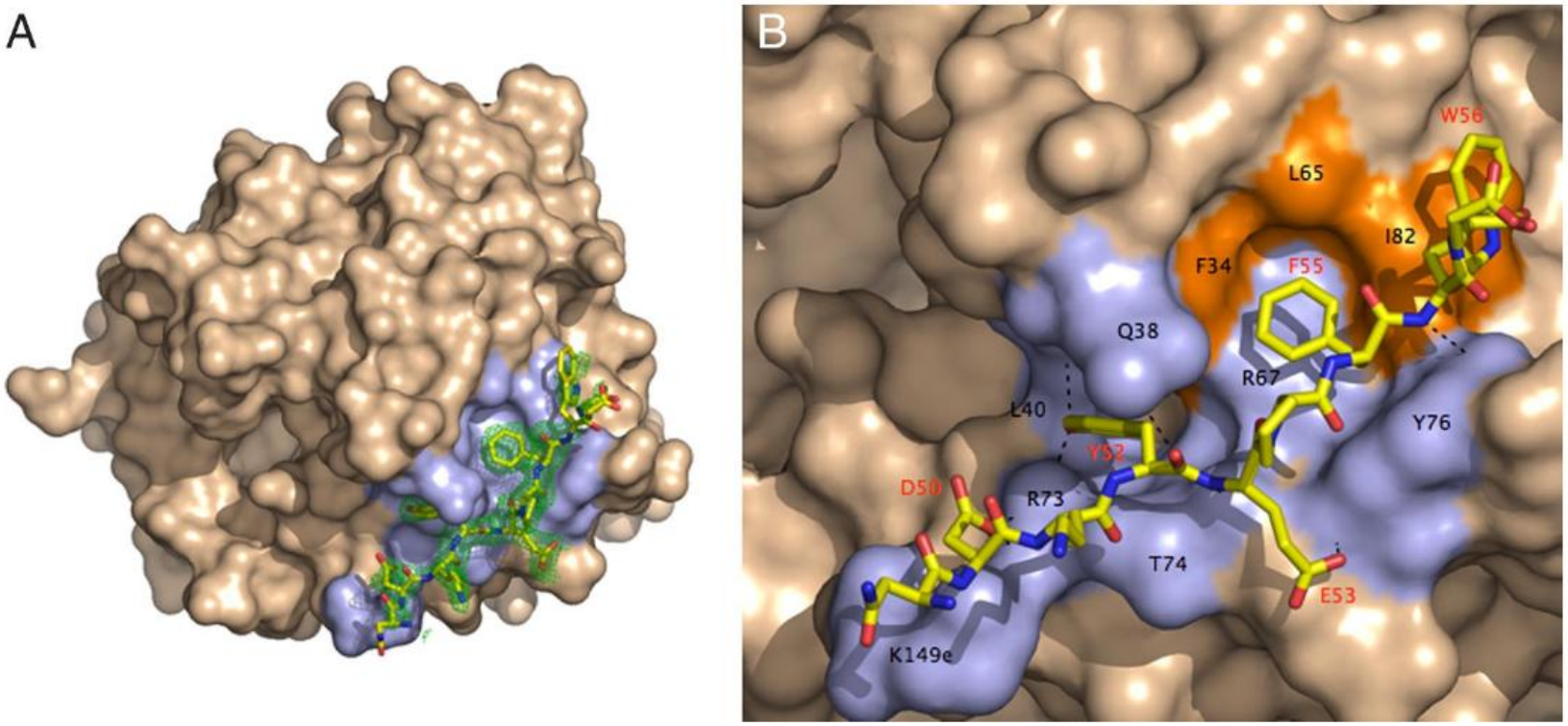


Коррелирующие (ко-эволюционирующие) позиции

Корреляция изменчивости в двух позициях может говорить о наличии структурной и/или функциональной взаимосвязи между ними.

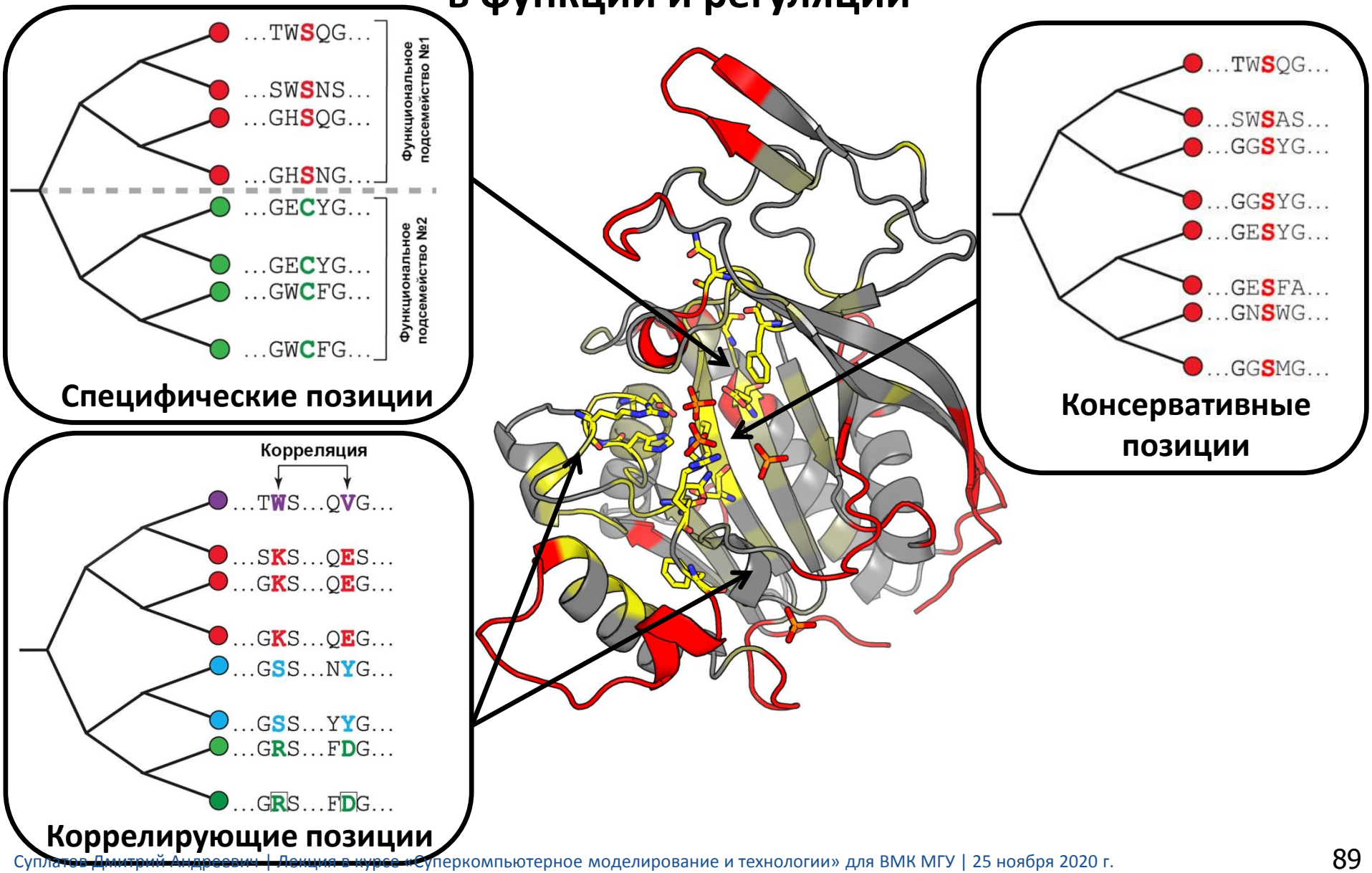
- Остатки в двух позициях могут образовывать физическое взаимодействие, поддерживающее структуру белка в активном состоянии;

Коррелирующие (ко-эволюционирующие) позиции



Коррелирующие позиции могут образовывать «цепочки» на поверхности белка, которые передают регуляторный сигнал между разными центрами.

Вариабельность аминокислот в участке структуры фермента и его гомологов в суперсемействе помогает понять роль остатков в функции и регуляции

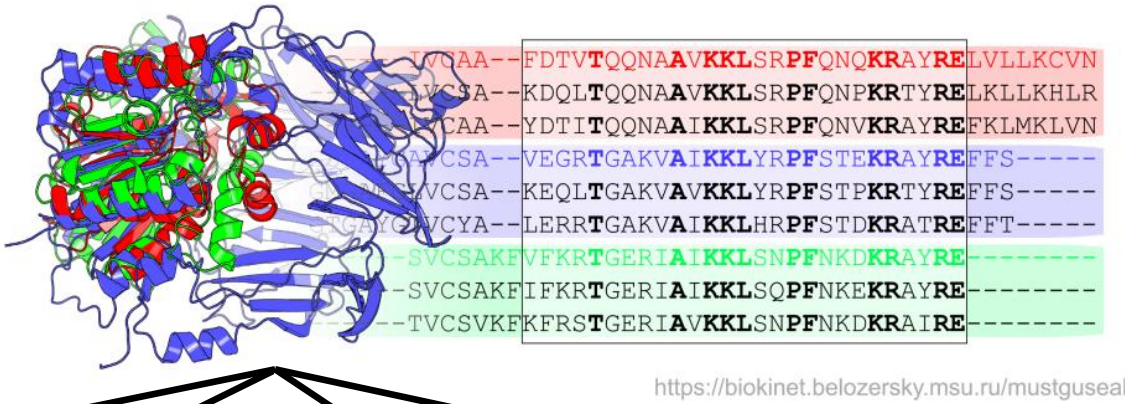


Открытая он-лайн платформа для изучения структуры, функции и регуляции белков методами биоинформатики

Mustguseal

Сбор всей доступной информации о последовательностях и структурах родственных белков

Bioinformatics, 2018



Zebra

Анализ механизмов функционального разнообразия в родственных белках

Nucleic Acids Research, 2020

pocketZebra

Поиск ранее неизвестных сайтов связывания субстратов и регуляторных лигандов в структурах белков

Nucleic Acids Research, 2014

visualCMAT

Поиск ко-эволюционирующих позиций в структурах родственных белков, изучение путей аллостерической коммуникации

J Bioinform Comput Biol., 2018

Yosshi

Изучение роли дисульфидных связей в функции и регуляции родственных белков

Nucleic Acids Research, 2019

Три уровня суперкомпьютерного моделирования белков

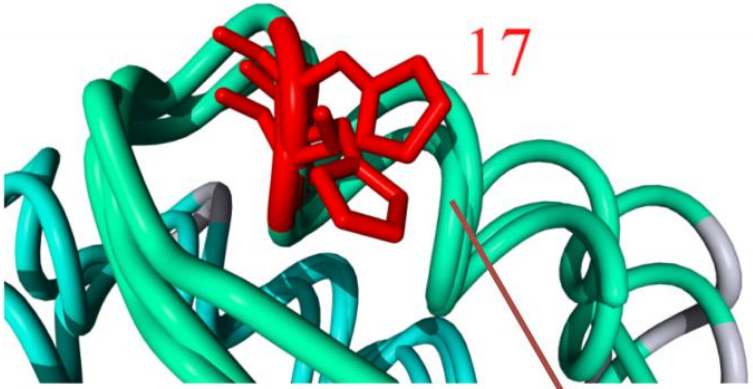
Уровень 1: Биоинформатический анализ всей доступной информации;

Уровень 2: Автоматизированный анализ текстов ранее опубликованных исследований;

Уровень 3: Молекулярное моделирование для понимания механизмов действия

Автоматизированный анализ литературы

Мы хотим оперировать всей информацией, ранее опубликованной по «нашей» теме




```
                10        15  17
HUMAN      15   RQLANYVHSK
CHICK      11   -ALADYVHAR
MOUSE      42   AFLADYAHSL
YEAST      12   GHVADHLHN-
```



Novel Tools for Extraction and Validation of Disease-Related Mutations Applied to Fabry Disease

HGVS HUMAN GENOME VARIATION SOCIETY

Remko Kuipers,[1,2†] Tom van den Bergh,[2†] Henk-Jan Joosten,[2,3] Ronald H. Lekanne dit Deprez,[4] Marcel MAM Mannens,[4*] and Peter J. Schaap[2,5]

[1]Nijmegen Centre for Molecular Life Sciences, Radboud University Nijmegen Medical Centre, Nijmegen, The Netherlands; [2]Laboratory of Microbiology, Wageningen University, Wageningen, The Netherlands; [3]Bio-Prodict, Wageningen, The Netherlands; [4]Department of Clinical Genetics, Academic Medical Centre, Amsterdam, The Netherlands; [5]Laboratory of Systems and Synthetic Biology, Wageningen University, Wageningen, The Netherlands

Communicated by Alastair F. Brown

Received 7 January 2010; accepted revised manuscript 23 June 2010.
Published online 13 July 2010 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/humu.21317

ABSTRACT: Genetic disorders are often caused by nonsynonymous nucleotide changes in one or more genes associated with the disease. Specific amino acid changes, however, can lead to large variability of phenotypic expression. For many genetic disorders this results in an increasing amount of publications describing phenotype-associated mutations in disorder-related genes. Keeping up with this stream of publications is essential for molecular diagnostics and translational research purposes but often impossible due to time constraints: there are simply too many articles to read. To help solve this problem, we have created Mutator, an automated method to extract mutations from full-text articles. Extracted mutations are crossreferenced to sequence data and a scoring method is applied to distinguish false-positives. To analyze stored and new mutation data for their (potential) effect we have developed Validator, a Web-based tool specifically designed for DNA diagnostics. Fabry disease, a monogenetic gene disorder of the GLA gene, was used as a test case. A structure-based sequence alignment of the alpha-amylase superfamily was used to validate results. We have compared our data with existing Fabry mutation data sets obtained from the HGMD and Swiss-Prot databases. Compared to these data sets, Mutator extracted 30% additional mutations from the literature. Hum Mutat 31:1026–1032, 2010. © 2010 Wiley-Liss, Inc.

KEY WORDS: Fabry; GLA; database; 3DM; validator; Mutator; alpha-amylase

Introduction

Due to the ease of today's gene sequencing methods, the relation between genes and corresponding diseases has been unraveled for several genetic disorders. Moreover, the specific sequencing of disease-related genes in patients has enormously increased the available mutation data in the literature. For some extensively investigated genes, gene-specific mutation databases are generated by extraction of mutational information from the literature. Examples of such mutation databases are the IARC TP53 Mutation database [Petitjean et al., 2007] and UMD p53 database for the tumor repressor gene TP53 [Olivier et al., 2002]. For molecular diagnostics and translational research these databases are used as reference to distinguish between naturally occurring single nucleotide polymorphisms (SNPs) and (potentially) pathogenic mutations in patients. Populating these databases usually requires manual intervention, which makes it difficult to generate and maintain mutation databases. Therefore, up-to-date mutational databases are only available for a select number of disease-related genes.

In 2004, a tool MuteXt [Horn et al., 2003] was described for the automatic extraction of mutational information from literature. This tool was specifically designed for populating the nuclear receptor [Van Durme et al., 2003] and GPCR [Horn et al., 2004] Molecular Class-Specific Information Systems with mutation data. We have used the MuteXt method as basis for a new tool, Mutator, which can automatically extract and store mutational information from the literature for genes that are related to a genetic disorder.

Mutator was used to create a Fabry mutational database (FMDB). Fabry disease is an X-linked inborn error of glycosphingolipid cata[illegible]from mutations in the alpha-galactosidase A g[illegible]0644) at Xq22.1. Currently, two main Fabry[illegible]tation data sets exist; the Human Genome[illegible](HGMD) [Stenson et al., 2009], and a collection of mutations automatically extracted from the UniProt databases [Yip et al., 2008]. The HGMD database is more complete because here mutational information is extracted from the literature. However, maintaining this database requires manual intervention. Our method extracts mutations from full text publications in a fully automated manner. The result shows an almost 100% coverage of mutations listed in the combined Uniprot and HGMD databases. Moreover, Mutator extracted from the literature 30% additional mutations covering 25% additional amino acid positions.

Human alpha-galactosidase is a member of the alpha-amylase protein superfamily. In the past, it was shown that protein superfamily-derived data contextually stored in a Molecular Class-Specific Information System (MCSIS) can be used to describe individual functions of residues in proteins [Folkertsma et al.,

Additional Supporting Information may be found in the online version of this article.
†The first two authors contributed equally to this work.
*Correspondence to: Marcel MAM Mannens, Department of Clinical Genetics, DNA diagnostics, Academic Medical Centre, Meibergdreef 15, Amsterdam 1105 AZ, The Netherlands. E-mail: m.a.mannens@amc.uva.nl

PLoS ONE (2017) 12(5): e0176427

… mutation
… … H49A …

Систематизация литературы

- Не только последовательности и структуры белков нуждаются в анализе и систематизации с использованием программных подходов;

- Научных публикаций становится так много, что для их анализа также необходимы программы и подходы биоинформатики (text mining)

Автоматизированное извлечение и классификация данных, опубликованных в литературе

- «COVID-19»: более **50 000 статей**;
- Дизайн ферментов: более **300 000 статей только за пять лет**;
- Эти статьи могут содержать информацию, полезную для конкретного нового исследования, однако прочитать все эти работы вручную не представляется возможным.

Автоматизированный анализ литературы

Как оперировать всей информацией,
ранее опубликованной по «нашей» теме

Как собрать и систематизировать всю литературу по теме исследования?

- **Поиск по ключевым словам** (PubMed) – очень ограниченный в своих возможностях инструмент, в лучшем случае позволяет собрать коллекцию статей в широкой области;

- **«Интеллектуальный анализ текстов»** представляет собой активно развивающееся междисциплинарное направление на стыке лингвистики и науки об анализе данных («data science»), потенциал которого существенно возрос в последние годы в связи с началом широкого использования нейросетевых подходов и GPU

Распознавание PDF текстов научных публикаций с учетом их особой структуры

{
  "name" : "32783664.0.pdf",
  "metadata" : {
    "source" : "CRF",
    "**title**" : "Psychological teleconsultations in patients suffering from chronic skin diseases during the COVID-19 era: a service to improve patients' quality of life",
    "authors" : ["Claudio Marasca", "Anna De Rosa", "Gabriella Fabbrocini", "Mariateresa Cantelli", "Angela Patrì", "Maria Vastarella", "Lucia Gallo", "Francesca di Vico",
 "Serena Poggi", "Angelo Ruggiero"],
    "emails" : ["angeloruggiero1993@libero.it"],
    "sections" : [{
      "heading" : "**Abstract**",
      "text" : "The COVID-19 era represented an important stressful event affecting population in many different ways, with important negative impact on social, working and relational life. Indeed, the homeisolation in addition to the high-level of distress given by fear of infection, has significantly resulted in a large number of psychological-consequences. Moreover, in order to guarantee a continuity of care, different measures have been applied among hospitals such as the implementation of teledermatology services. For these reasons, we implemented at our Dermatological-Clinic psychological video-consultations through our teledermatology-services. Herein we report our experience of 23 psychological-video-consultations, which led to achieve a significant reduction of DLQI (from 4.4±3.9 at baseline to 1.6±2.5 at week-4) in patients suffering from chronic skin conditions during the COVID-19 era.\nAc ce\npt ed\nM an\nus cr\nipt"
    }, {
      "heading" : "**Introduction**",
      "text" : "Since the COVID-19-outbreak, different measures have been applied among hospitals in order to limit the infection's spread as well as to guarantee a continuity of cares. 1 These measures included the implementation of teledermatology services, with video-call-visits, phone-calls, e-mails and text-messages to be used for either the "online-triaging" and "real-time control visits" in order to limit in-person appointments and at the same time supporting patients. 2,3 During the COVID-19era, the home-isolation in addition to the high-level of distress given by fear of infection, has significantly resulted in a large number of psychological-consequences.4 Moreover, patients\nsuffering from chronic inflammatory skin conditions may feel stigmatized and this further intensifies their lack of self-confidence and self-esteem; feelings of shame and guilt increase the tendency toward suicidal-ideation. 5 Hence, COVID-19-era represented a stressful-life-event which may have constituted a major risk for diseases recurrences, exacerbating the symptoms' severity and duration. 5 For these reasons, in order to guarantee continuity-of-cares as well as ensure better psychological-outcomes, we implemented at our Dermatological-Clinic (University-of-NaplesFederico-II) psychological video-consultations through our teledermatology-services. Herein, we report our experience of 23 psychological-video-consultations, which led to achieve a significant reduction of DLQI in patients suffering from chronic skin conditions during the COVID-19 era."
    }
…

Named-entity recognition (NER)

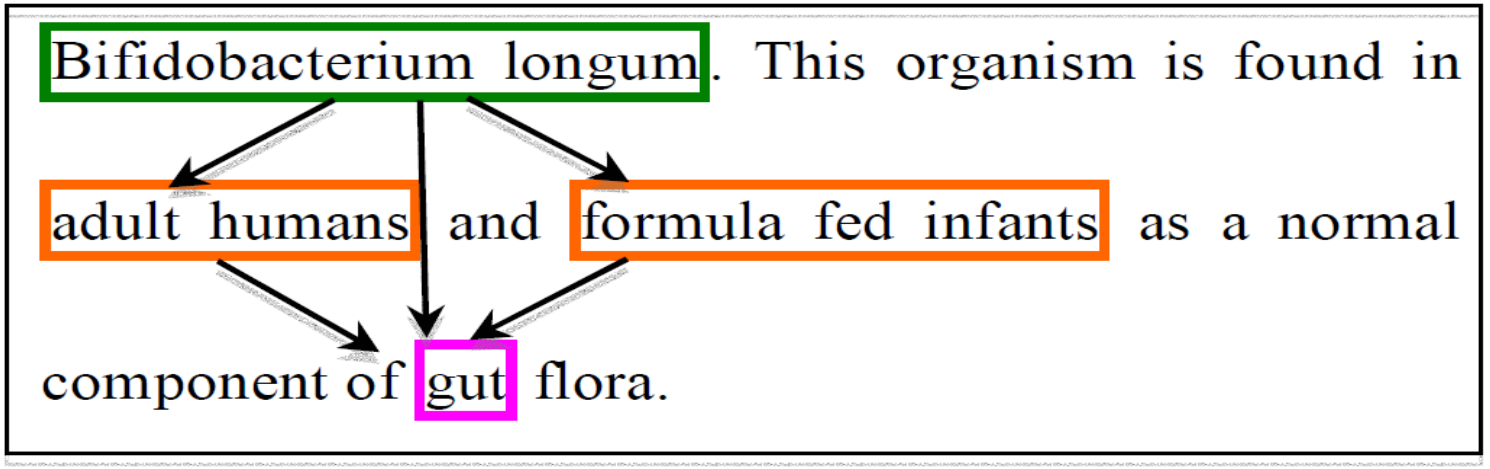
Распознавание «сущностей» в тексте (упоминания мутаций, названия белка, организма и т.д.)



Autophagy maintains tumour growth through circulating arginine. Autophagy captures intracellular components and delivers them to lysosomes, where they are degraded and recycled to sustain metabolism and to enable survival during starvation1-5. Acute, whole-body deletion of the essential autophagy gene Atg7 in adult mice causes a systemic metabolic defect that manifests as starvation intolerance and gradual loss of white adipose tissue, liver glycogen and muscle mass1. Cancer cells also benefit from autophagy. Deletion of essential autophagy genes impairs the metabolism, proliferation, survival and malignancy of spontaneous tumours in models of autochthonous cancer6,7. Acute, systemic deletion of Atg7 or acute, systemic expression of a dominant-negative ATG4b in mice induces greater regression of KRAS-driven cancers than does tumour-specific autophagy deletion, which suggests that host autophagy promotes tumour growth1,8. Here we show that host-specific deletion of Atg7 impairs the growth of multiple allografted tumours, although not all tumour lines were sensitive to host autophagy status. Loss of autophagy in the host was associated with a reduction in circulating arginine, and the sensitive tumour cell lines were arginine auxotrophs owing to the lack of expression of the enzyme argininosuccinate synthase 1. Serum proteomic analysis identified the arginine-degrading enzyme arginase I (ARG1) in the circulation of Atg7-deficient hosts, and in vivo arginine metabolic tracing demonstrated that serum arginine was degraded to ornithine. ARG1 is predominantly expressed in the liver and can be released from hepatocytes into the circulation. Liver-specific deletion of Atg7 produced circulating ARG1, and reduced both serum arginine and tumour growth. Deletion of Atg5 in the host similarly regulated [corrected] circulating arginine and suppressed tumorigenesis, which demonstrates that this phenotype is specific to autophagy function rather than to deletion of Atg7. Dietary supplementation of Atg7-deficient hosts with arginine partially restored levels of circulating arginine and tumour growth. Thus, defective autophagy in the host leads to the release of ARG1 from the liver and the degradation of circulating arginine, which is essential for tumour growth; this identifies a metabolic vulnerability of cancer. (PMID:30429607)

Relation Extraction (RE)

Распознавание взаимосвязанных «сущностей»



«Интеллектуальный анализ текстов»

- Не поиск по ключевым словам, а попытка научить машину распознавать конкретную информацию;
- Обучение происходит на большом, но конечном, числе примеров;
- Полученная в результате модель «искусственного интеллекта» (нейронная сеть) способна распознавать информацию за пределами тренировочной выборки

«Интеллектуальный анализ текстов»

NER: распознавание химического соединения

- We have previously detailed our assessment of **aspirin** and non-steroidal antiinflammatory drugs (NSAID) use in the NHS cohort

«Интеллектуальный анализ текстов»

NER: распознавание химического соединения

- Molecular modelling has confirmed that doramapimod combines the features of type I and II inhibitors of p38α MAPK, i.e. can directly and indirectly compete with the ATP binding.

Где в этом предложении химическое соединение?

«Интеллектуальный анализ текстов»

NER: распознавание химического соединения

- Molecular modelling has confirmed that **doramapimod** combines the features of type I and II inhibitors of p38α MAPK, i.e. can directly and indirectly compete with the **ATP** binding.

«Интеллектуальный анализ текстов»

Разметка B-I-O для NER (распознавание сущностей)

Scp160p O
is O
a O
1222 O
amino O
acid O
Saccharomyces B
cerevisiae I
protein O
that O
contains O
14 O
copies O
of O
the O
hnRNP O
K O
homology O
(O
KH O
…

Parkinson O
' O
s O
disease O
(O
PD O
) O
randomized O
to O
receive O
10 O
mg O
selegiline B
per O
day O
and O
L B
- I
dopa I
compared O
with O
those O
taking O
L B
- I
dopa I
alone O
. O

In O
colon B
carcinoma I
cells O
, O
loss O
of O
APC O
leads O
to O
the O
accumulation O
of O
betacatenin O
in O
the O
nucleus O
…

- Организмы
- Химические соединения
- Болезни

«Интеллектуальный анализ текстов»

Разметка для RE (распознавание взаимосвязей)

A recent study demonstrated the association of **@GENE$** located in the upstream region with risk for or protection from **@DISEASE$** (SLE). **1 (есть взаимосвязь)**

Based on **@GENE$** analyses, 41 **@DISEASE$** patients and 12 healthy controls were studied. **0 (нет взаимосвязи)**

«Интеллектуальный анализ текстов»

- Разметки используются для тренировки нейронных сетей, предназначенных для решения одной конкретной задачи (распознавания мутаций в тексте или извлечение эффектов мутаций);

- Полученные нейросети используются для анализа и систематизации литературы по теме исследования.

Дипломная/курсовая работа в области «Интеллектуального анализа текстов»

- Использование нейросетевых подходов для интеллектуального анализа литературы и извлечения информации о результатах экспериментальных исследований белков/ферментов;
- Классификация экспериментальной информации о результатах сайт-направленного мутагенеза белков;
- Разработка полезных инструментов и веб-приложений для белковой инженерии.



Sequence analysis

Mustguseal: a server for multiple structure-guided sequence alignment of protein families

Dmitry A. Suplatov[1], Kirill E. Kopylov[2], Nina N. Popova[3], Vladimir V. Voevodin[3,4] and Vytas K. Švedas[1,2,*]

Nucleic Acids Research, 2020, Vol. 48, Web Server issue W65–W71

Zebra2: advanced and easy-to-use web-server for bioinformatic analysis of subfamily-specific and conserved positions in diverse protein superfamilies

Dmitry Suplatov, Yana Sharapova, Elizaveta Geraseva and Vytas Švedas

ABSTRACT

Zebra2 is a highly automated web-tool to search for subfamily-specific and conserved positions (i.e. the determinants of functional diversity as well as the key catalytic and structural residues) in protein superfamilies. The bioinformatic analysis is facilitated by Mustguseal—a companion web-server to automatically collect and superimpose a large representative set of functionally diverse homologs with high structure similarity but low sequence identity to the selected query protein. The results are automatically prioritized and provided at four information levels to facilitate the knowledge-driven expert selection of the most promising positions on-line: as a sequence similarity network; interfaces to sequence-based and 3D-structure-based analysis of conservation and variability; and accompanied by the detailed annotation of proteins accumulated from the integrated databases with links to the external resources. The integration of Zebra2 and Mustguseal web-tools provides the first of its kind out-of-the-box open-access solution to conduct a systematic analysis of evolutionarily related proteins implementing different functions within a shared 3D-structure of the superfamily, determine common and specific patterns of function-associated local structural elements, assist to select hot-spots for rational design and to prepare focused libraries for directed evolution. The web-servers are free and open to all users at https://biokinet.belozersky.msu.ru/zebra2, no login required.

INTRODUCTION

Nucleic Acids Research, 2019, Vol. 47, Web Server issue W108–W114

Yosshi: a web-server for disulfide engineering by bioinformatic analysis of diverse protein families

Dmitry Suplatov*, Daria Timonina, Yana Sharapova and Vytas Švedas

ABSTRACT

Disulfide bonds play a significant role in protein stability, function or regulation but are poorly conserved among evolutionarily related proteins. The Yosshi can help to understand the role of S–S bonds by comparing sequences and structures of homologs with diverse properties and different disulfide connectivity patterns within a common structural fold of a superfamily, and assist to select the most promising hot-spots to improve stability of proteins/enzymes or modulate their functions by introducing naturally occurring crosslinks. The bioinformatic analysis is supported by the integrated Mustguseal web-server to construct large structure-guided sequence alignments of functionally diverse protein families that can include thousands of proteins based on all available information in public databases. The Yosshi+Mustguseal is a new integrated web-tool for a systematic homology-driven analysis and engineering of S–S bonds that facilitates a broader interpretation of disulfides not just as a factor of structural stability, but rather as a mechanism to implement functional diversity within a superfamily. The results can be downloaded as a content-rich PyMol session file or further studied online using the HTML5-based interactive analysis tools. Both web-servers are free and open to all users at https://biokinet.belozersky.msu.ru/yosshi and there is no login requirement.

INTRODUCTION

d.a.suplatov@belozersky.msu.ru

Nucleic Acids Research 2020, 2019;
Bioinformatics 2019, 2018

Три уровня суперкомпьютерного моделирования белков

Уровень 1: Биоинформатический анализ всей доступной информации;

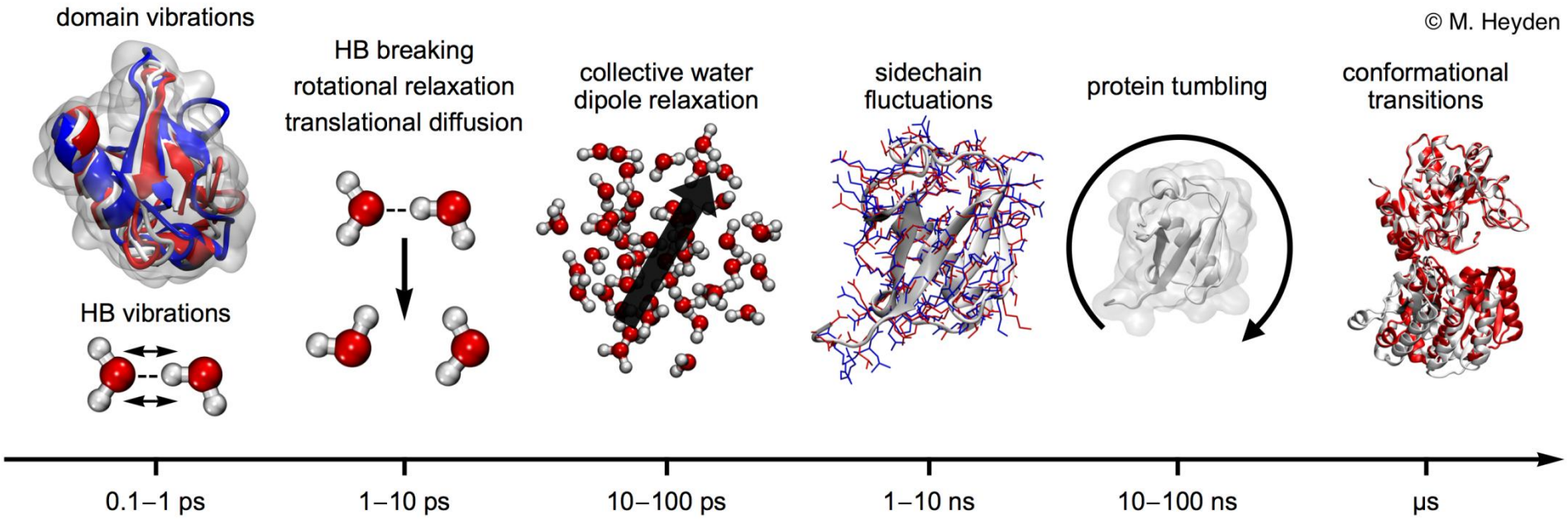
Уровень 2: Автоматизированный анализ текстов ранее опубликованных исследований;

Уровень 3: Молекулярное моделирование для понимания механизмов действия

Уровень 3:
Молекулярное моделирование

Как изучить роль позиций, отобранных на основании биоинформатического анализа?

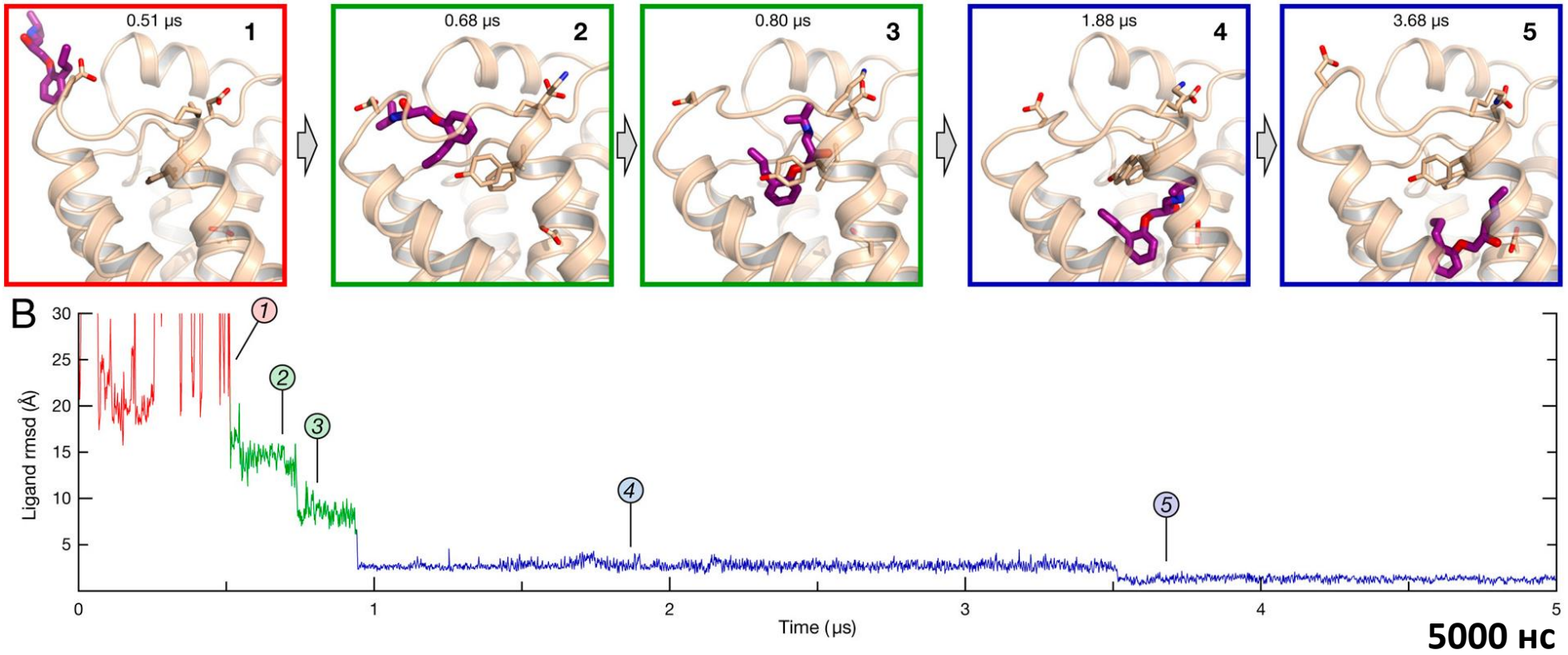
Молекулярная динамика: метод для изучения структуры белка в движении



https://isearch.asu.edu/profile/3160407

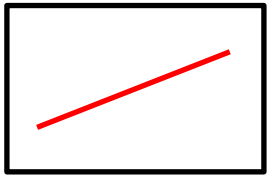
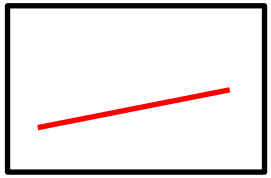
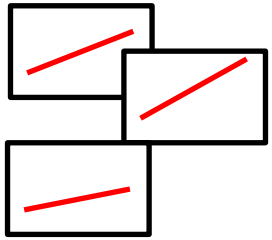
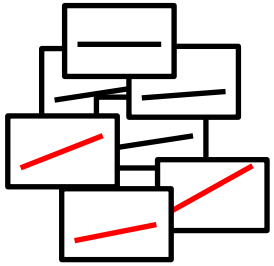
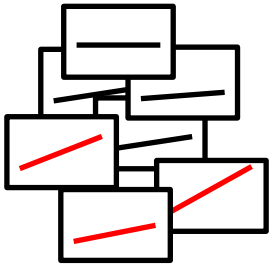
Для изучения более сложных конформационных изменений в структуре белка необходимо рассчитывать более длинные траектории молекулярной динамики

Примеры работы с использованием молекулярной динамики (2011 год)



Dror R. O. et al. Pathway and mechanism of drug binding to G-protein-coupled receptors //Proceedings of the National Academy of Sciences. – 2011. – Т. 108. – №. 32. – С. 13118-13123.

Объем вычислений молекулярной динамики *в повседневной лабораторной практике*

До 2000	2000 – 2008	2008 – 2014	2014-2017	**2018**
< 1 нс 	10 нс 	10 – 100 нс 	100 нс 	**1000 нс** 
		Чебышев Ломоносов-1	Ломоносов-2 	Ломоносов-2 

Объем вычислений молекулярной динамики *в повседневной лабораторной практике*

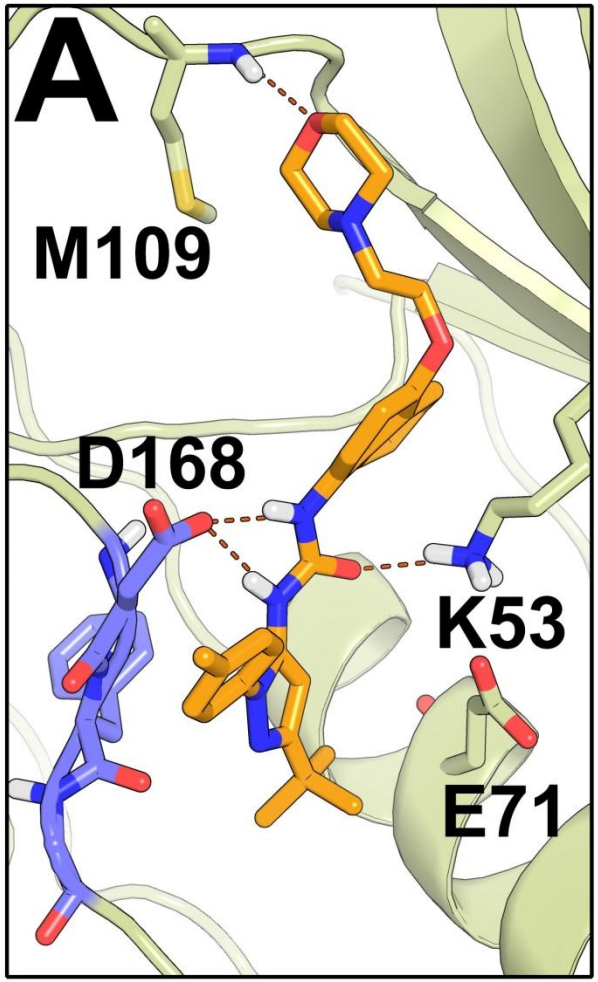
Современные GPU-ускорители позволяют использовать молекулярную динамику на еще недавно невообразимом уровне в повседневной практике:

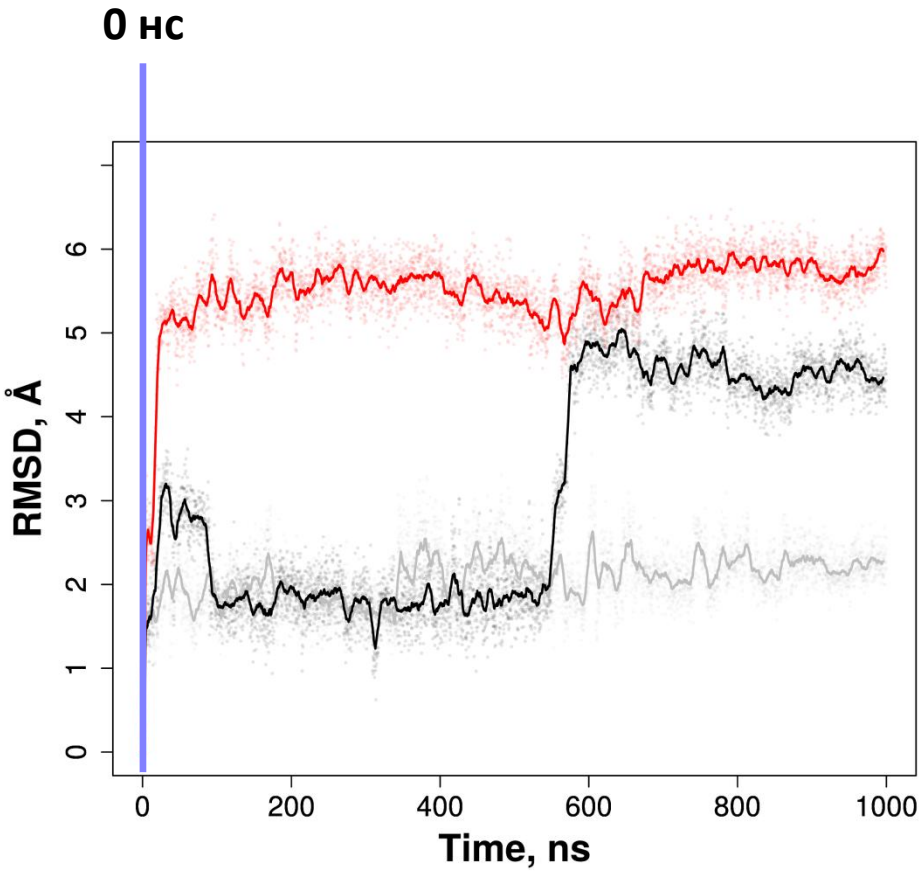
- Позволяют считать все более длинные (1000+ нс) траектории для больших систем (100+ тыс. атомов);
- Позволяют одновременно вычислять несколько независимых траекторий;

Объем вычислений молекулярной динамики *в повседневной лабораторной практике*

- Зачем считать длинные траектории?
  - Чтобы получить возможность увидеть и изучить конформационные перестройки, возникающие на больших временных шкалах;

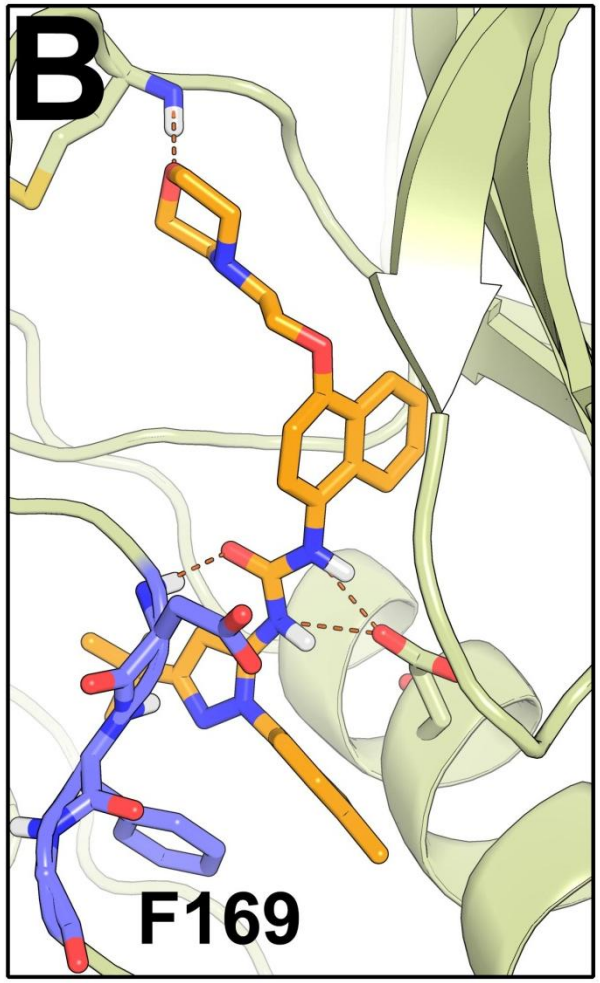
Симуляция структурной подвижности комплекса p38a MAPK человека с ингибитором дорамапимодом

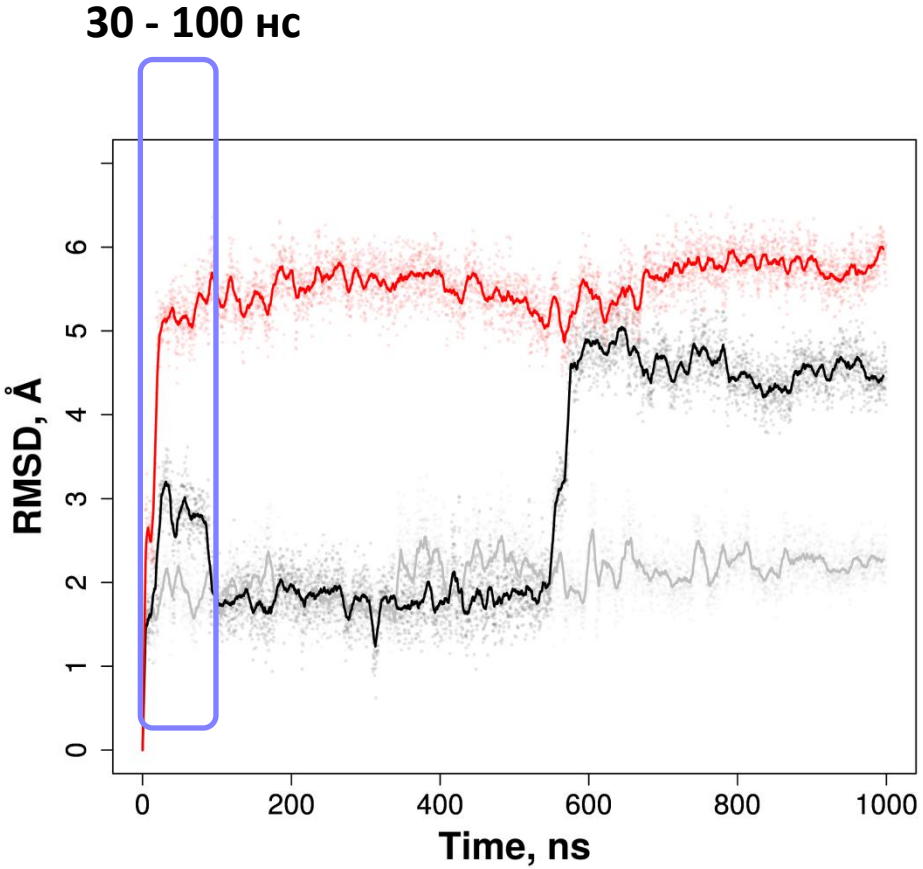




Suplatov, D. A., Kopylov, K. E., Sharapova Y.A., Švedas V. (2019) *J. Biomol. Struct. Dyn.,* 37(8), 2049-2060

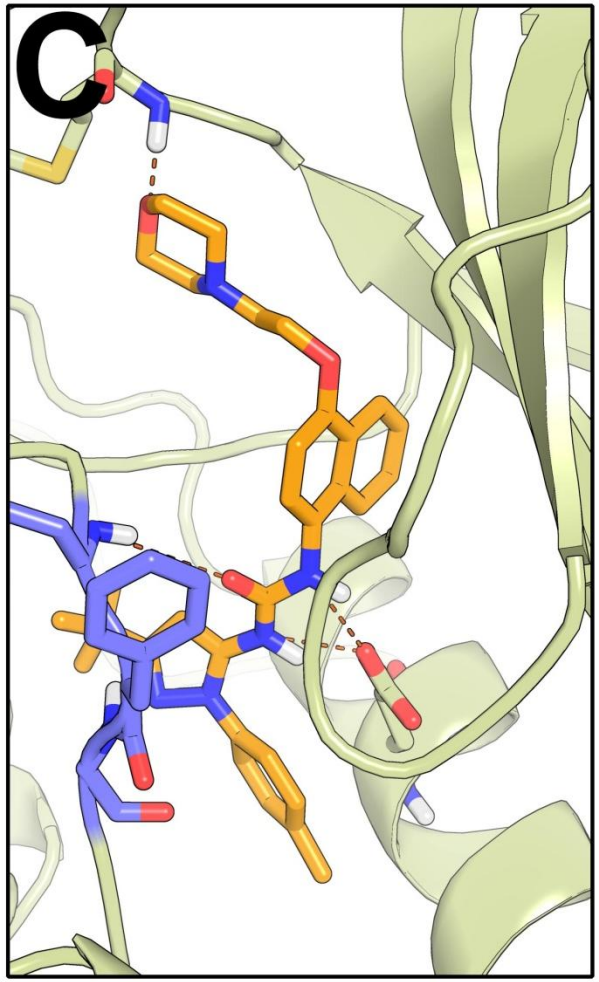
Симуляция структурной подвижности комплекса p38a MAPK человека с ингибитором дорамапимодом

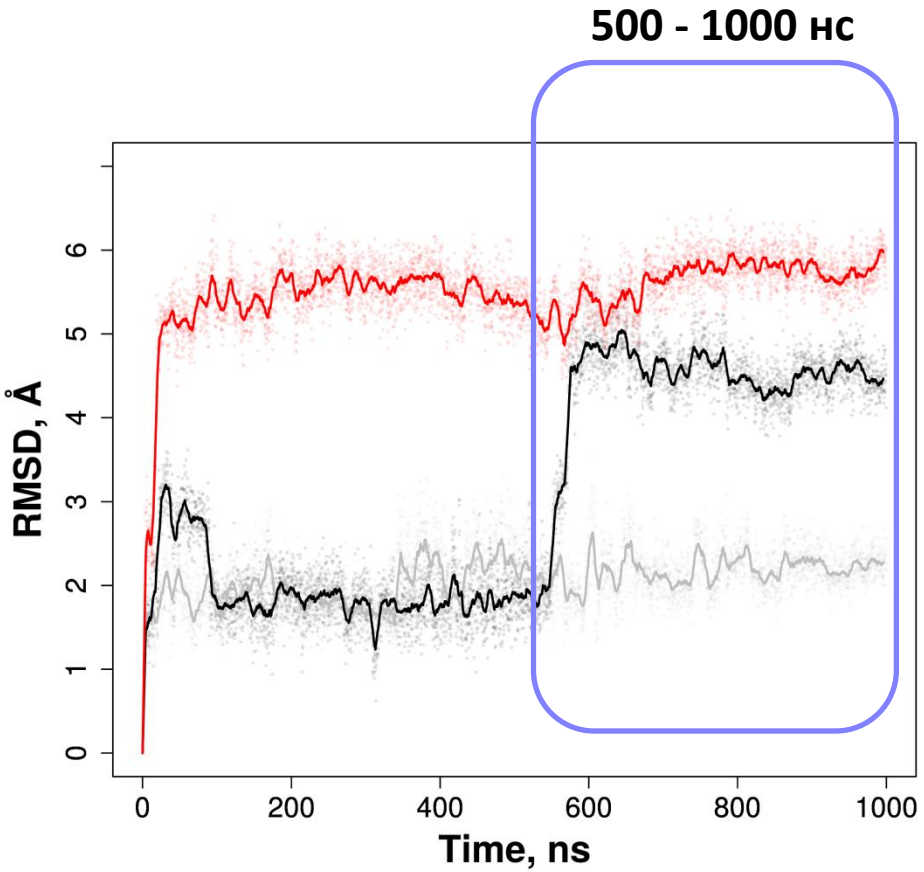




Suplatov, D. A., Kopylov, K. E., Sharapova Y.A., Švedas V. (2019) *J. Biomol. Struct. Dyn.,* 37(8), 2049-2060

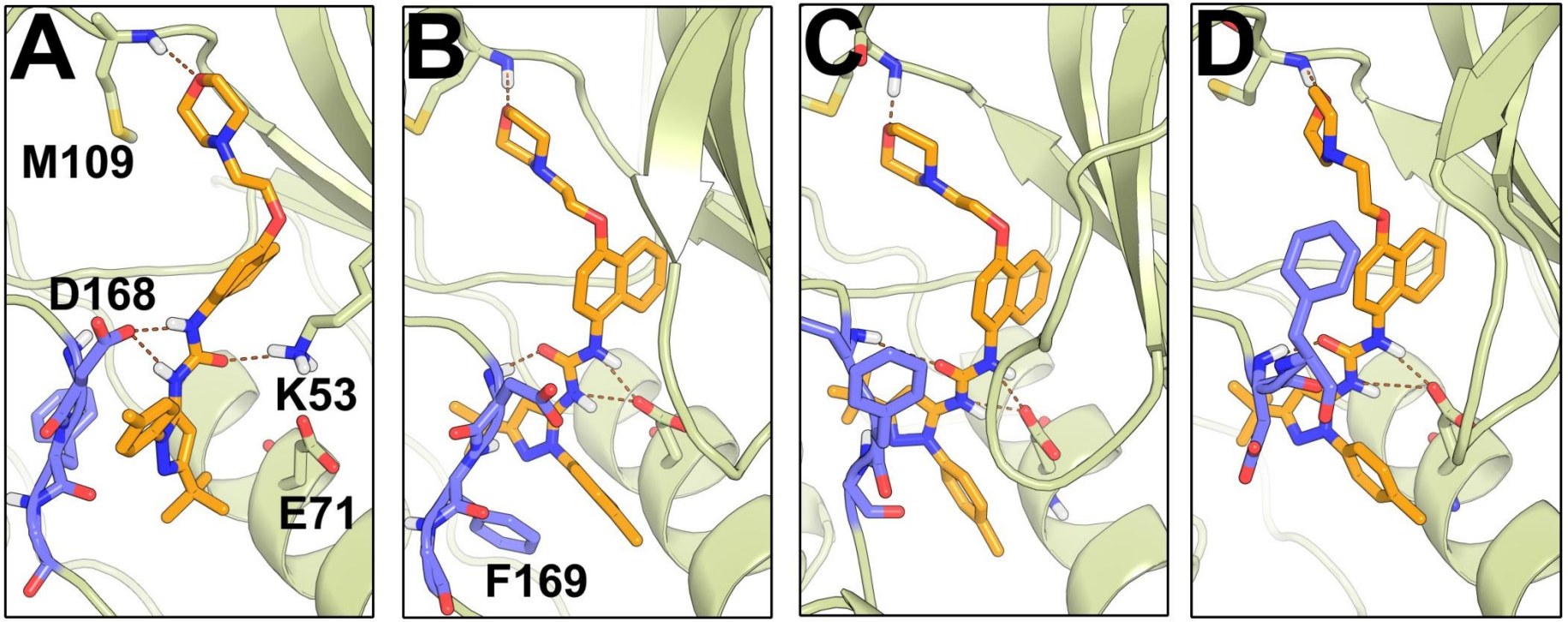
Симуляция структурной подвижности комплекса p38a MAPK человека с ингибитором дорамапимодом





Suplatov, D. A., Kopylov, K. E., Sharapova Y.A., Švedas V. (2019) *J. Biomol. Struct. Dyn.,* 37(8), 2049-2060

Длительная симуляция структурной подвижности p38a MAPK человека позволила изучить последовательные стадии механизма связывания дорамапимода



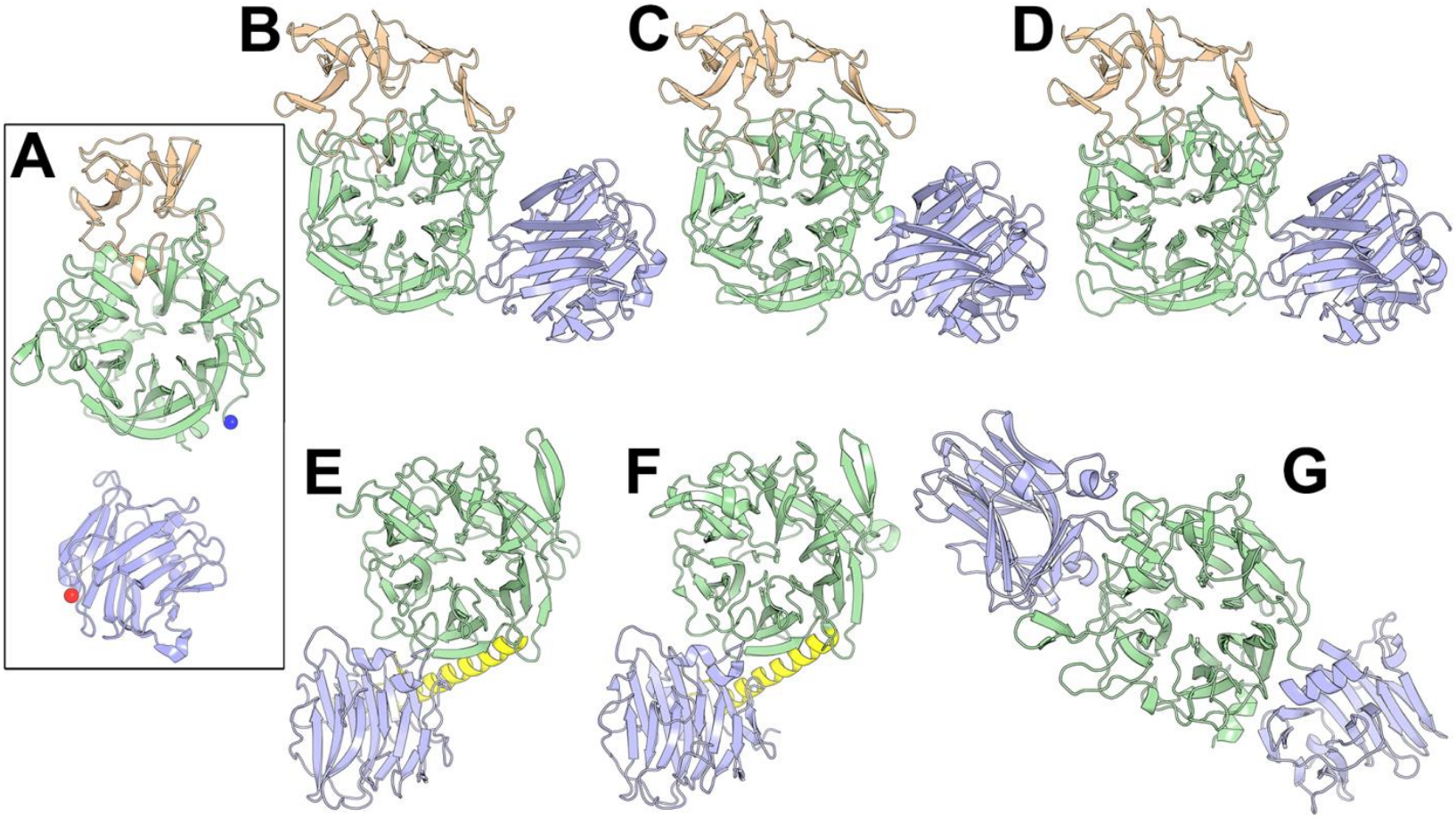
0 нс **30 - 100 нс** **500 - 1000 нс** **PDB структура**

Suplatov, D. A., Kopylov, K. E., Sharapova Y.A., Švedas V. Human p38α Mitogen-Activated Protein Kinase in the Asp168-Phe169-Gly170-in (DFG-in) state can bind allosteric inhibitor Doramapimod (2019) *J. Biomol. Struct. Dyn.,* 37(8), 2049-2060

Объем вычислений молекулярной динамики *в повседневной лабораторной практике*

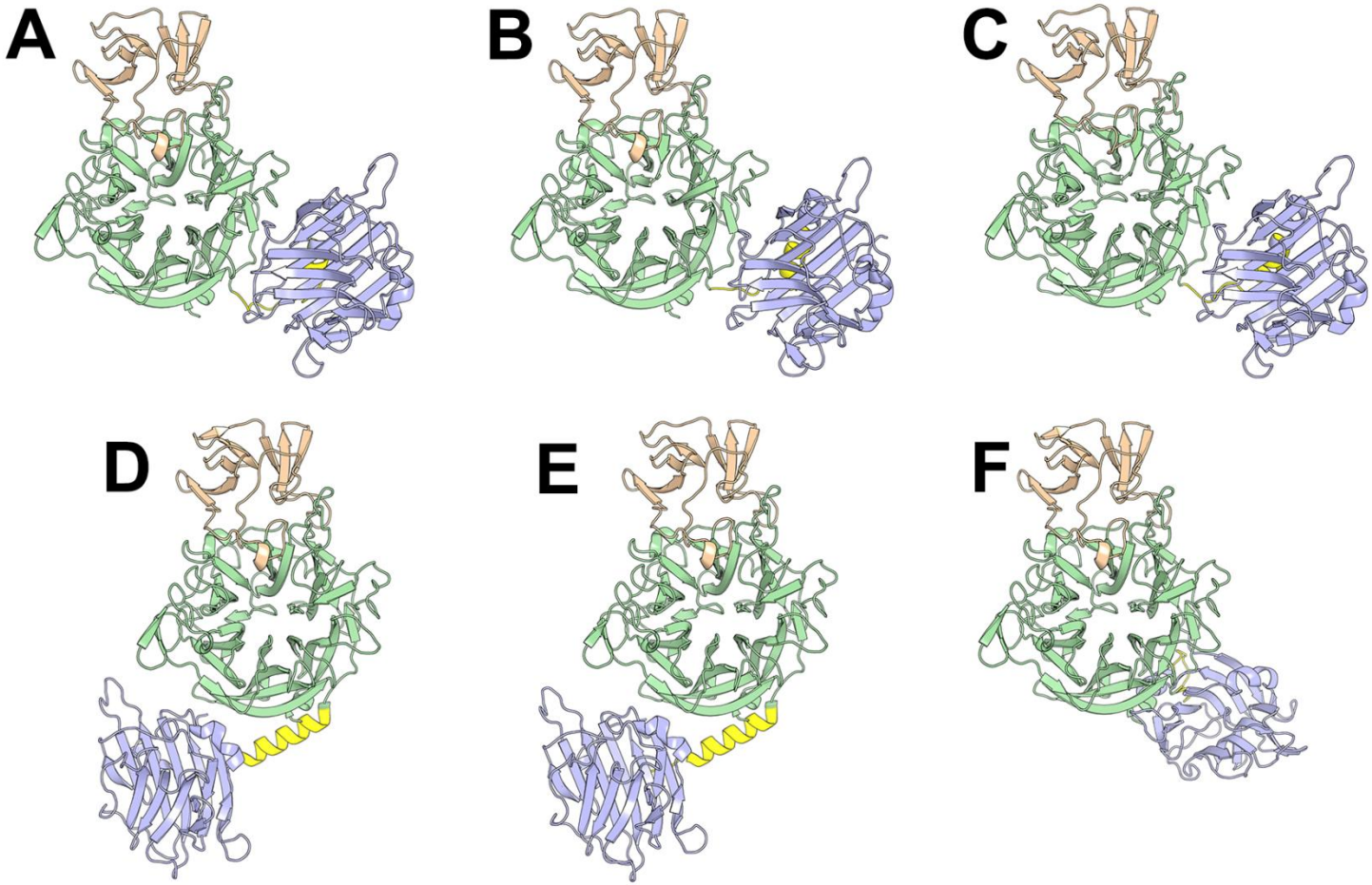
- Зачем считать несколько параллельных траекторий?
  - Для запуска траекторий с разными начальными условиями с целью собрать больше информации о конкретном объекте для последующего анализа;
  - Для запуска траекторий с разными параметрами/настройками молекулярной динамики (например, при разных температурах);
  - Для одновременного изучения разных объектов/моделей, что позволяет ускорить исследование в целом;

Симуляция структурной подвижности для изучения доменной организации бактериальных нейраминдаз



Sharapova Y., Suplatov D., Švedas V. Neuraminidase A from Streptococcus Pneumoniae Has a Modular Organization of Catalytic and Lectin Domains Separated by a Flexible Linker. *The FEBS Journal*. 2018. vol. 285, no. 13. pp. 2428–2445. DOI: 10.1111/febs.14486.

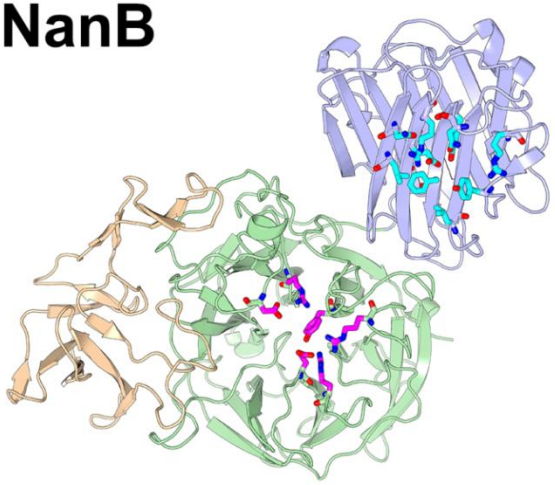
Симуляция структурной подвижности для изучения доменной организации бактериальных нейраминдаз



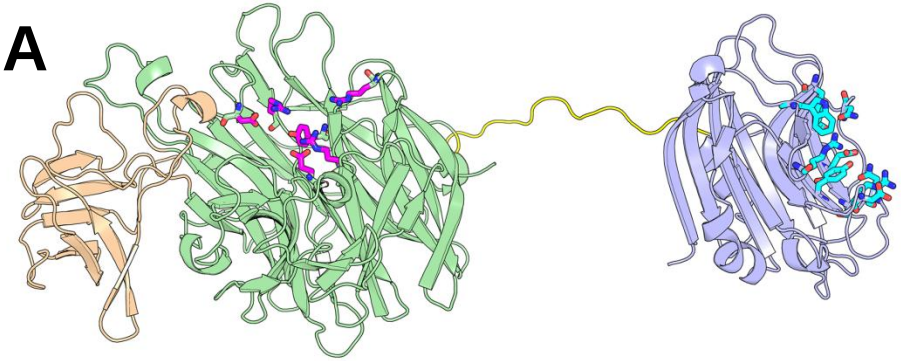
Sharapova Y., Suplatov D., Švedas V. Neuraminidase A from Streptococcus Pneumoniae Has a Modular Organization of Catalytic and Lectin Domains Separated by a Flexible Linker. *The FEBS Journal*. 2018. vol. 285, no. 13. pp. 2428–2445. DOI: 10.1111/febs.14486.

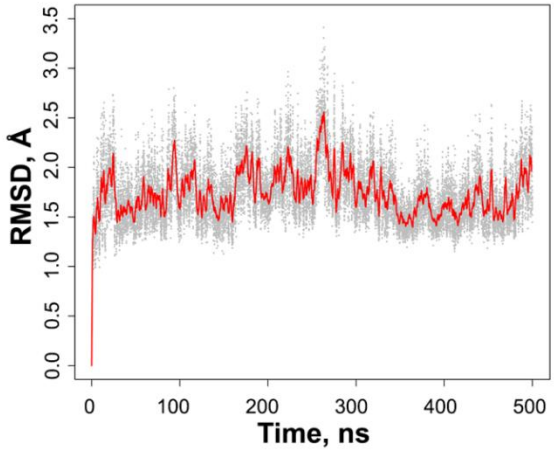
Симуляция структурной подвижности для изучения доменной организации бактериальных нейраминдаз

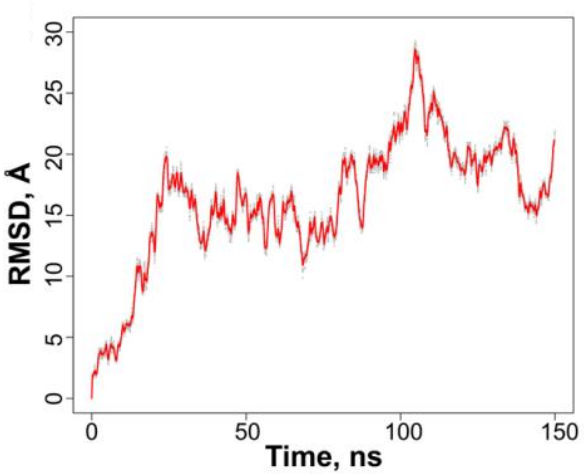
NanB



NanA







Sharapova Y., Suplatov D., Švedas V. Neuraminidase A from Streptococcus Pneumoniae Has a Modular Organization of Catalytic and Lectin Domains Separated by a Flexible Linker. *The FEBS Journal*. 2018. vol. 285, no. 13. pp. 2428–2445. DOI: 10.1111/febs.14486.

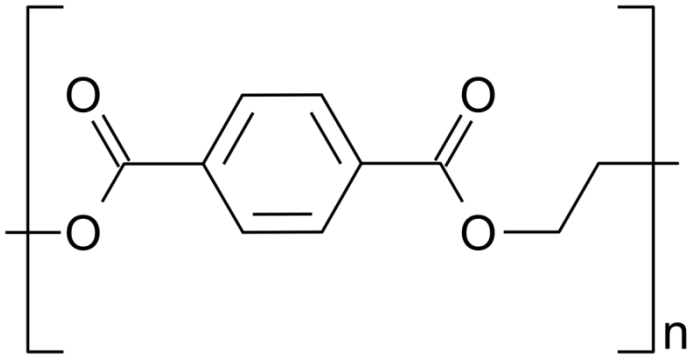
Систематический рациональный дизайн

Междисциплинарный подход к белковой инженерии

- **Биоинформатический анализ**: Выбор «наиболее перспективных» hot-spots на основе систематизации информации о последовательностях, структурах, аннотациях белков/ферментов;
  - в том числе, «интеллектуальный анализ» литературы для ее автоматической аннотации и систематизации.
- **Молекулярное моделирование**: изучение роли «hot-spots» и замен, рекомендованных биоинформатическим анализом, механизмов влияния на функцию;

Часть III: Задачи

Полиэтилентерефталат (PET) - один из самых распространенных видов пластика







Бутылки для напитков, косметики, бытовой химии

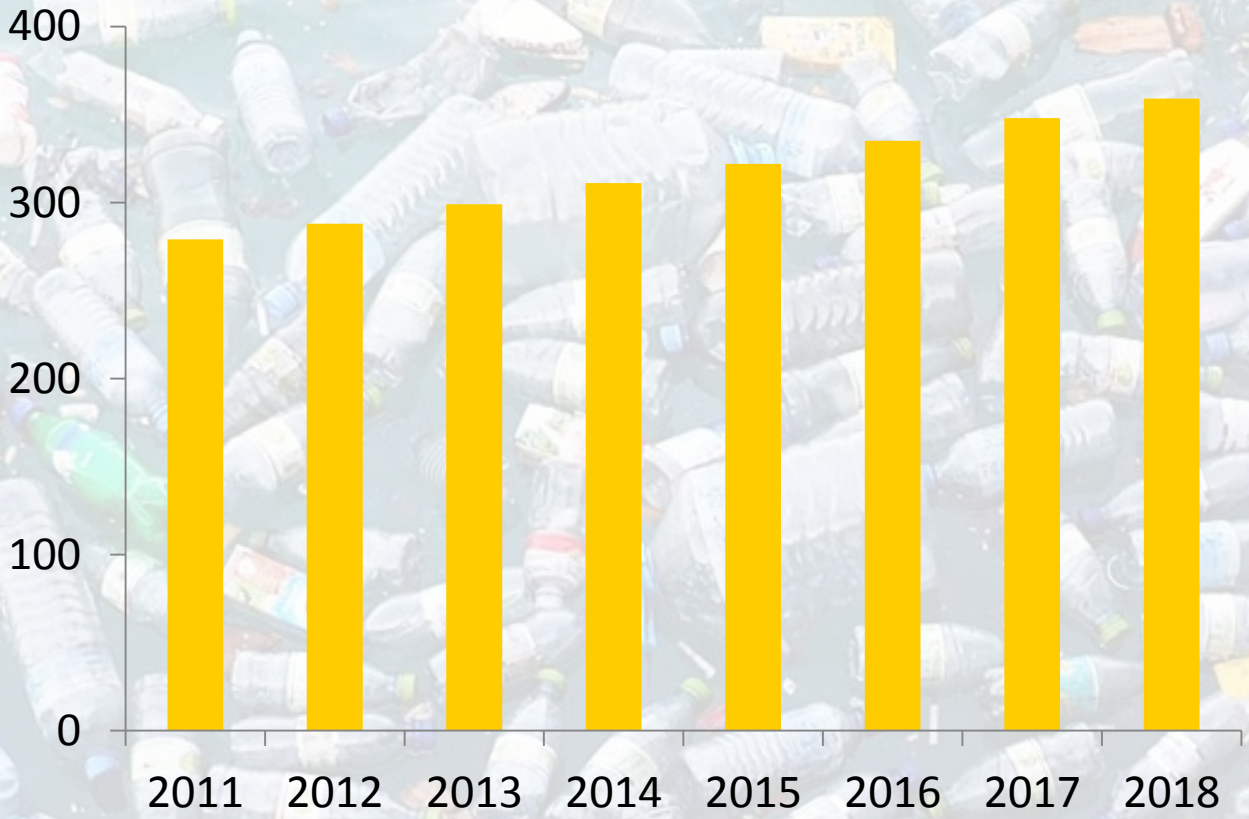


Одноразовая упаковка



Текстиль

Глобальное производство пластика, млн. тонн



https://www.plasticseurope.org/en/resources/market-data

Переработка PET:

Механическая – прессование, измельчение, плавление, гранулирование.

При каждом цикле переработки полимерные цепочки рвутся, укорачиваются, что приводит к потере первичных свойств и ухудшению качества полимера. Поэтому переработать пластик механически возможно лишь несколько раз, после чего он окажется на свалке.

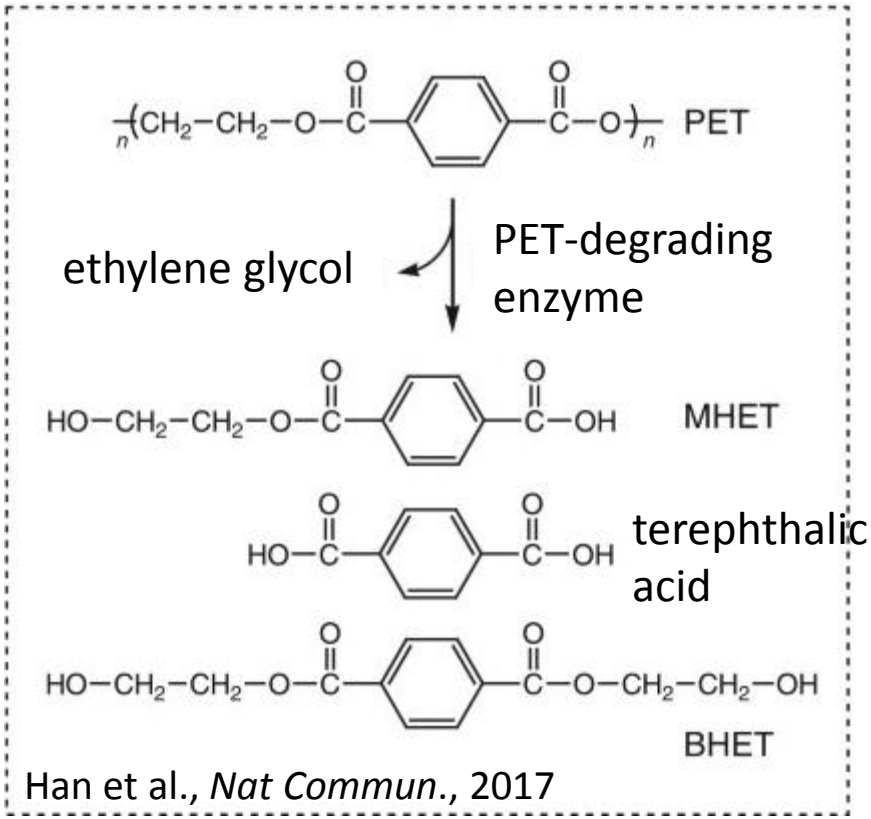
Химическая – кислый/щелочной гидролиз, метанолиз и тд. до исходных мономеров без потери качества.

Затрачивается больше невозобновляемых ресурсов и денег;
может приводить к образованию токсических примесей и отходов.

Биодеградация – разложение полимера на исходные мономеры микроорганизмами.

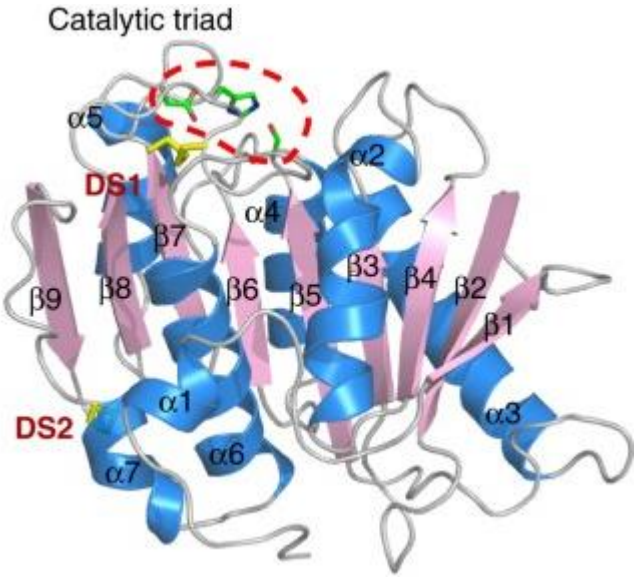
Экологично и безопасно, но крайне медленно и неэффектино.

Биодеградация PET



Han et al., *Nat Commun.*, 2017

Все известные ферменты, способные деградировать PET, относятся к суперсемейству α/β-гидролаз



Однако речь идет о микрограммах – граммах продукта за дни – месяцы ферментативной реакции, что несопоставимо с масштабами производства PET



nature

Explore our content ⌄ Journal information ⌄

nature › articles › article

Article | Published: 08 April 2020

An engineered PET depolymerase to break down and recycle plastic bottles

V. Tournier, C. M. Topham, A. Gilles, B. David, C. Folgoas, E. Moya-Leclair, E. Kamionka, M.-L. Desrousseaux, H. Texier, S. Gavalda, M. Cot, E. Guémard, M. Dalibey, J. Nomme, G. Cioci, S. Barbe, M. Chateau, I. André ✉, S. Duquesne ✉ & A. Marty ✉

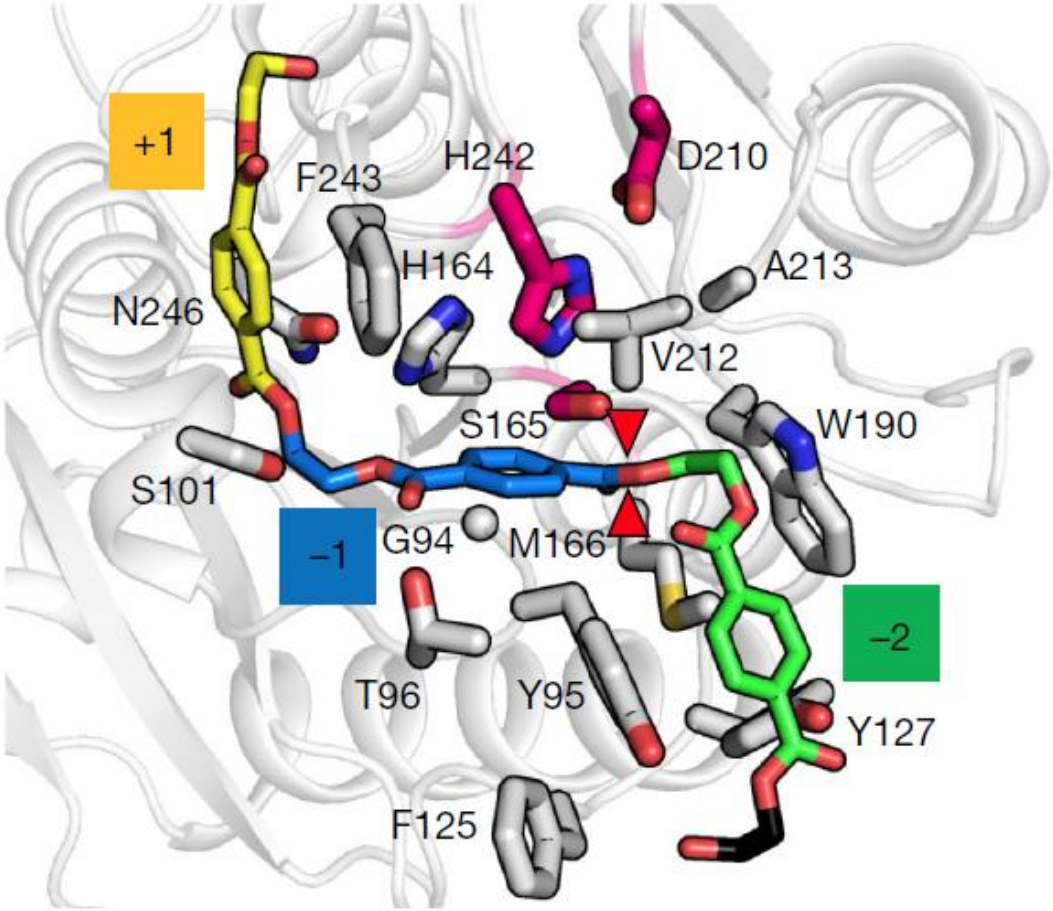
Nature **580**, 216–219(2020) | Cite this article

37k Accesses | **28** Citations | **1284** Altmetric | Metrics

Скорость деполимеризации ПЭТ 1 мг LCC-кутиназы дикого типа – 93.2 **мг** продукта (терефталевой кислоты) в час при 65°C.

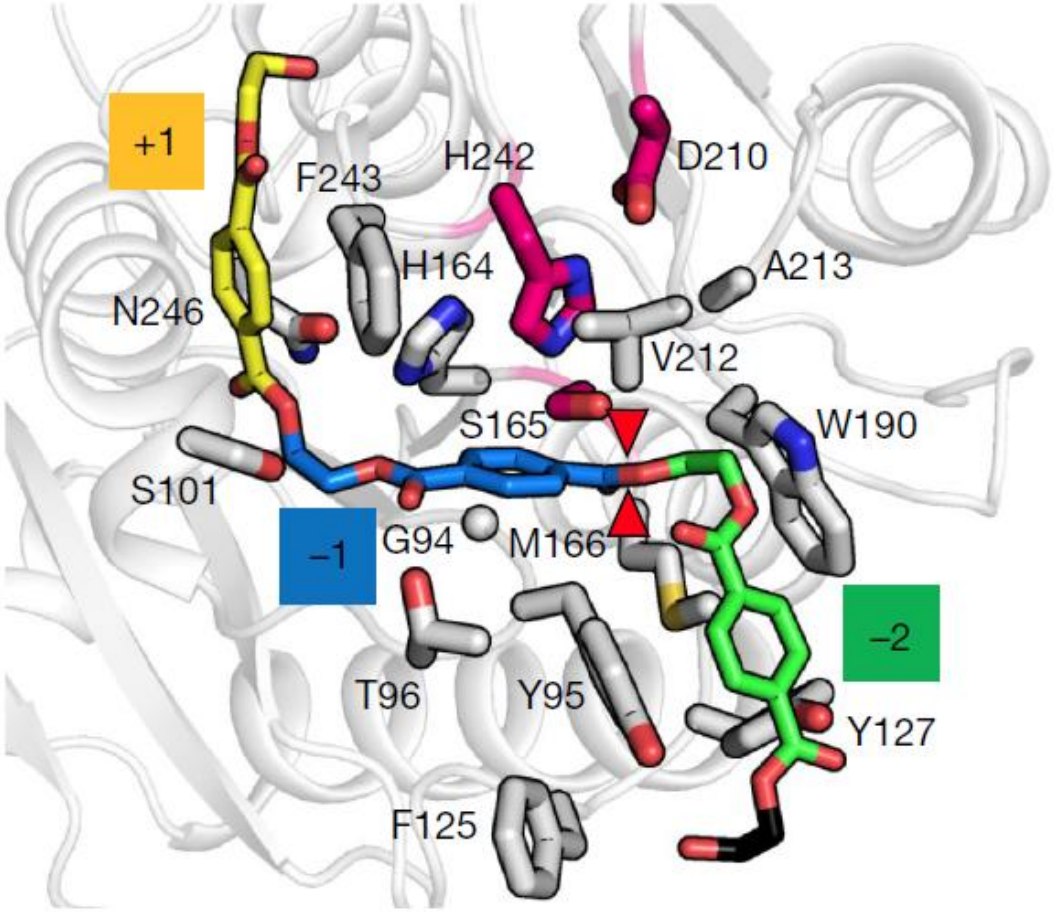
Авторы пишут, что оптимизировали этот фермент настолько, что стало возможным его использование для **промышленной переработки** пластиковых бутылок.

Белковая инженерия PET-деградирующей компостной LCC-кутиназы



- В кристаллографическую структуру дикого типа вручную докировали кусочек ПЭТ-субстрата таким образом, чтобы карбонильный кислород разрезаемой связи был связан в оксианионном центре, а концевые бензольные кольца образовывали π-стэкинг с F243 и Y95.
- Стоит отметить, что механизм связывания ПЭТ неизвестен, и его взаимодействие с F243 и Y95 – лишь догадка авторов.

Белковая инженерия PET-деградирующей компостной LCC-кутиназы



Рационализация:

- Выбраны 11 из 15 остатков, контактировавших с субстратом в полученной модели комплекса

Рандомизация:

- Позиции подвергнуты насыщающему мутагенезу

Белковая инженерия PET-деградирующей компостной LCC-кутиназы

Enzyme	Specific activity compared to wild-type LCC (%) ± s.d.	ΔTm (°C) ± s.d.
Wild-type LCC	100	0
T96M	72.5 ± 3.8	+2.7 ± 0.2
Y127G	66.6 ± 6.5	+2.3 ± 0.0
F243I	127.5 ± 6.4	-3.1 ± 0.0
F243W	118.4 ± 6.8	+1.4 ± 0.0
N246D	104.3 ± 2.9	+3.2 ± 0.0
N246M	99.6 ± 2.2	+3.3 ± 0.2
D238C/S283C	72.4 ± 2.8	+9.5 ± 0.0

- Из 209 мутантов лишь 2 показали улучшенную на 20-30% активность (F243I и F243W);
- Эти варианты попытались скомбинировать с еще четырьмя мутациями, которые, хоть и снизили активность фермента, но немного повысили его термостабильность;
- С этой же целью ввели дисульфидную связь.

Восемь мутантов были проверены в лабораторных условиях и в маленьком биореакторе. В условиях биореактора лучшим оказался вариант **F243I/D238C/S283C/Y127G**

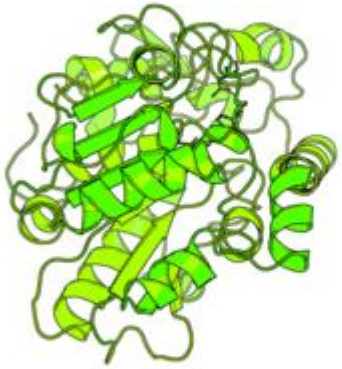
Белковая инженерия PET-деградирующей компостной LCC-кутиназы

- Улучшенный фермент позволяет получать 200 грамм терефталевой кислоты из килограмма суспензированного PET в течение 10 часов при концентрации фермента 3мг на 1г PET и температуре 72°C.

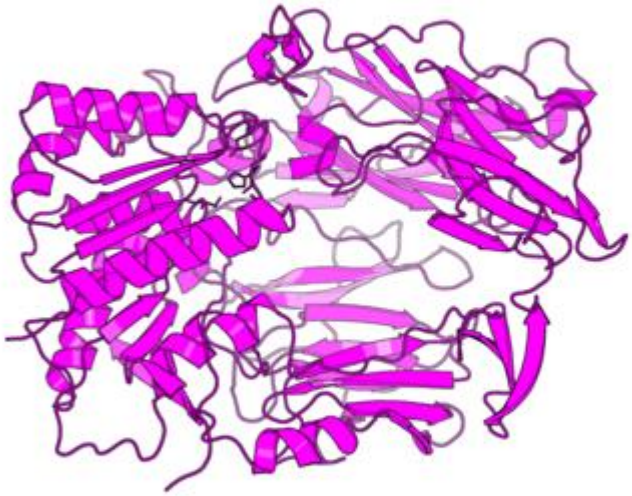
- Согласно подсчетам авторов, стоимость фермента, необходимого для переработки 1 тонны PET, составляет ~4% от цены за тонну virgin PET при стоимости производства кутиназы 25$ за килограмм фермента.

Липаза и амидаза суперсемейства α/β-гидролаз

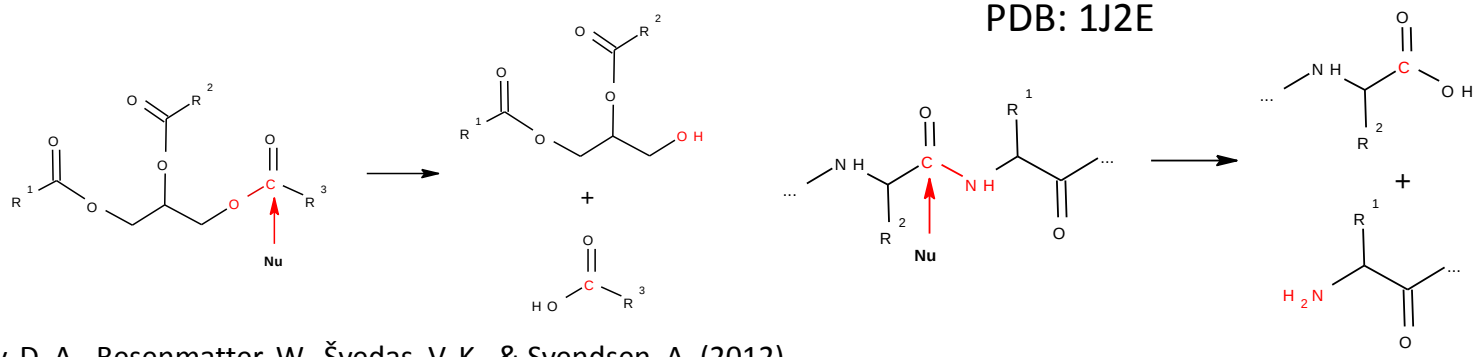
Эволюционно родственные ферменты с разными каталитическими активностями в рамках общей структурной организации



Candida antarctica
Липаза Б, PDB:1TCB



Дипептидил **пептидаза** IV
PDB: 1J2E



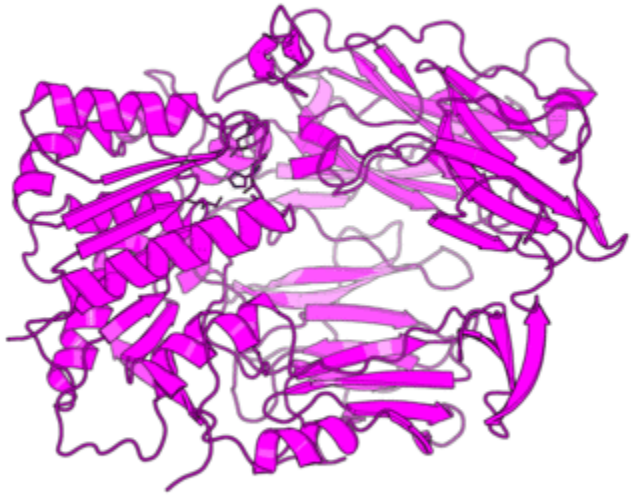
Suplatov, D. A., Besenmatter, W., Švedas, V. K., & Svendsen, A. (2012). *Protein Engineering, Design & Selection*, 25(11), 689-697.

Липаза и амидаза суперсемейства α/β-гидролаз

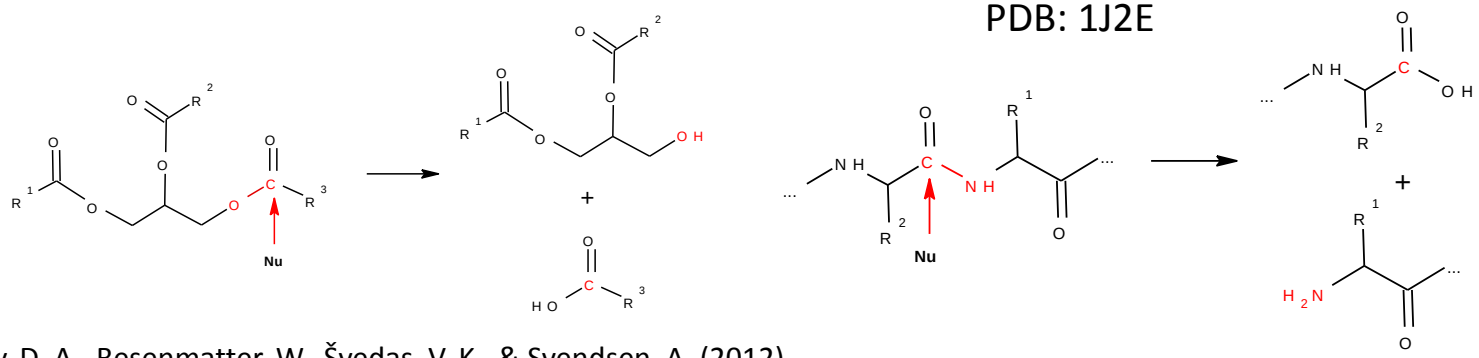
Эволюционно родственные ферменты с разными каталитическими активностями в рамках общей структурной организации



Candida antarctica
Липаза Б, PDB:1TCB



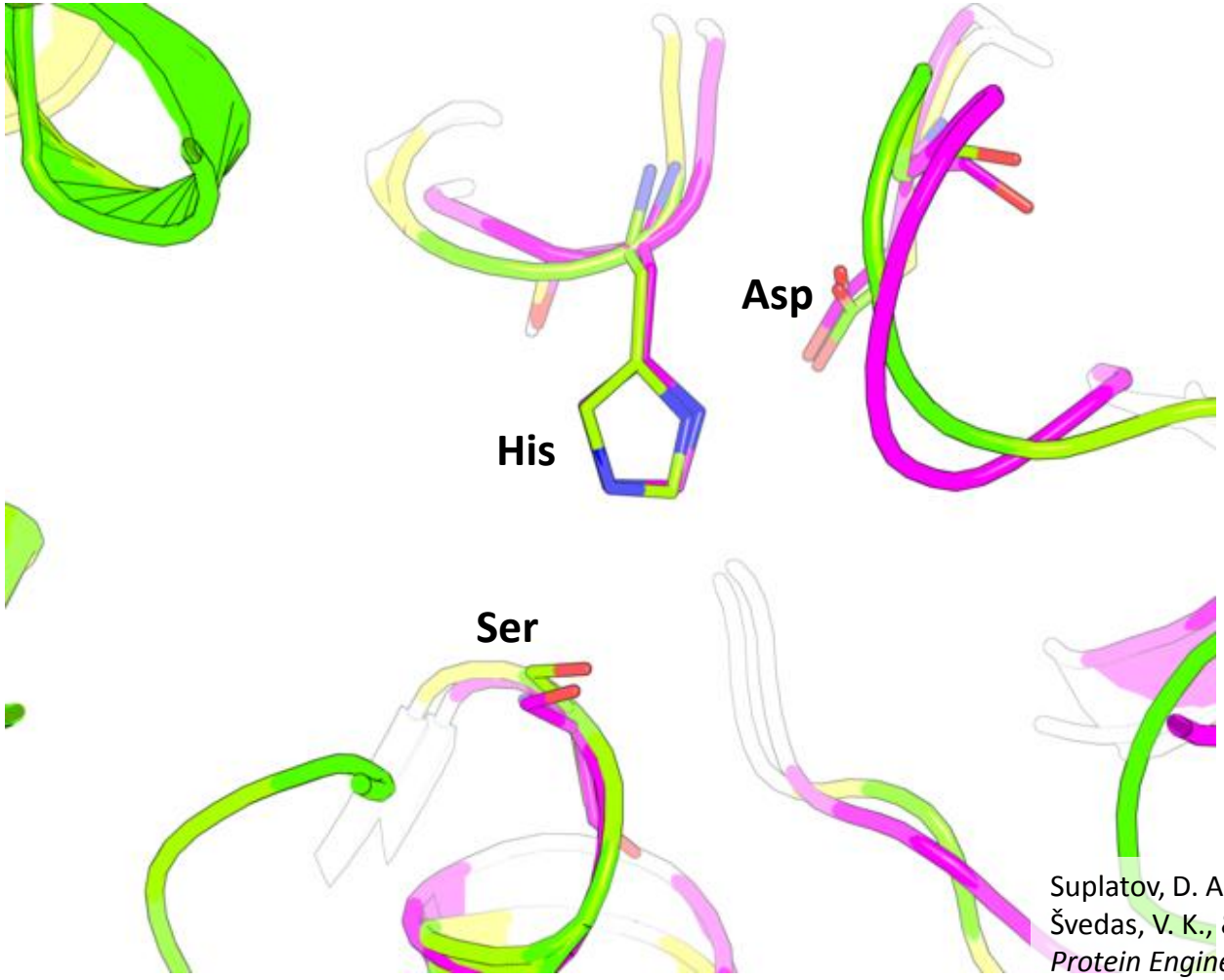
Дипептидил **пептидаза** IV
PDB: 1J2E



Suplatov, D. A., Besenmatter, W., Švedas, V. K., & Svendsen, A. (2012). *Protein Engineering, Design & Selection*, 25(11), 689-697.

Липаза и амидаза суперсемейства α/β-гидролаз

Эволюционно родственные ферменты с разными каталитическими активностями в рамках общей структурной организации



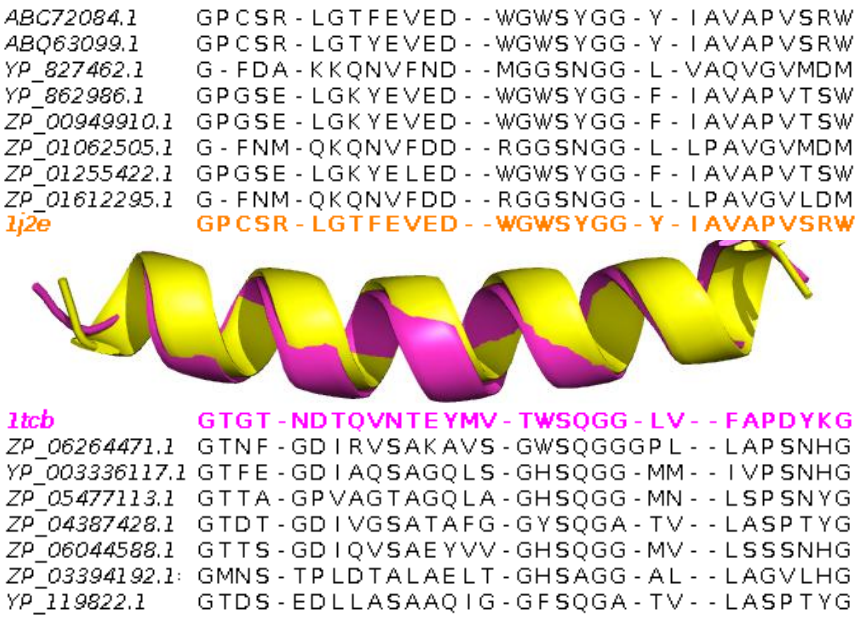
Suplatov, D. A., Besenmatter, W., Švedas, V. K., & Svendsen, A. (2012). *Protein Engineering, Design & Selection*, 25(11), 689-697.

Каталитическая триада консервативна как по последовательности, так и по структуре

Биоинформатический анализ информации о последовательностях, структурах и функциях α/β-гидролаз

- Структурное выравнивание липаз и пептидаз
  - **Липаза** Б из *Candida antarctica* (1TCB)
  - Сериновая карбокси**пептидаза** пшеницы (1WHS)
  - Дипептидил **пептидазы** человека 4 и 7 (1J2E и 3N0T)
  - Пролиновая имино**пептидаза** (1AZW)

- Структурно-опосредованное множественное выравнивание аминокислотных последовательностей



Использование специфических позиций подсемейства как «горячих точек замен»

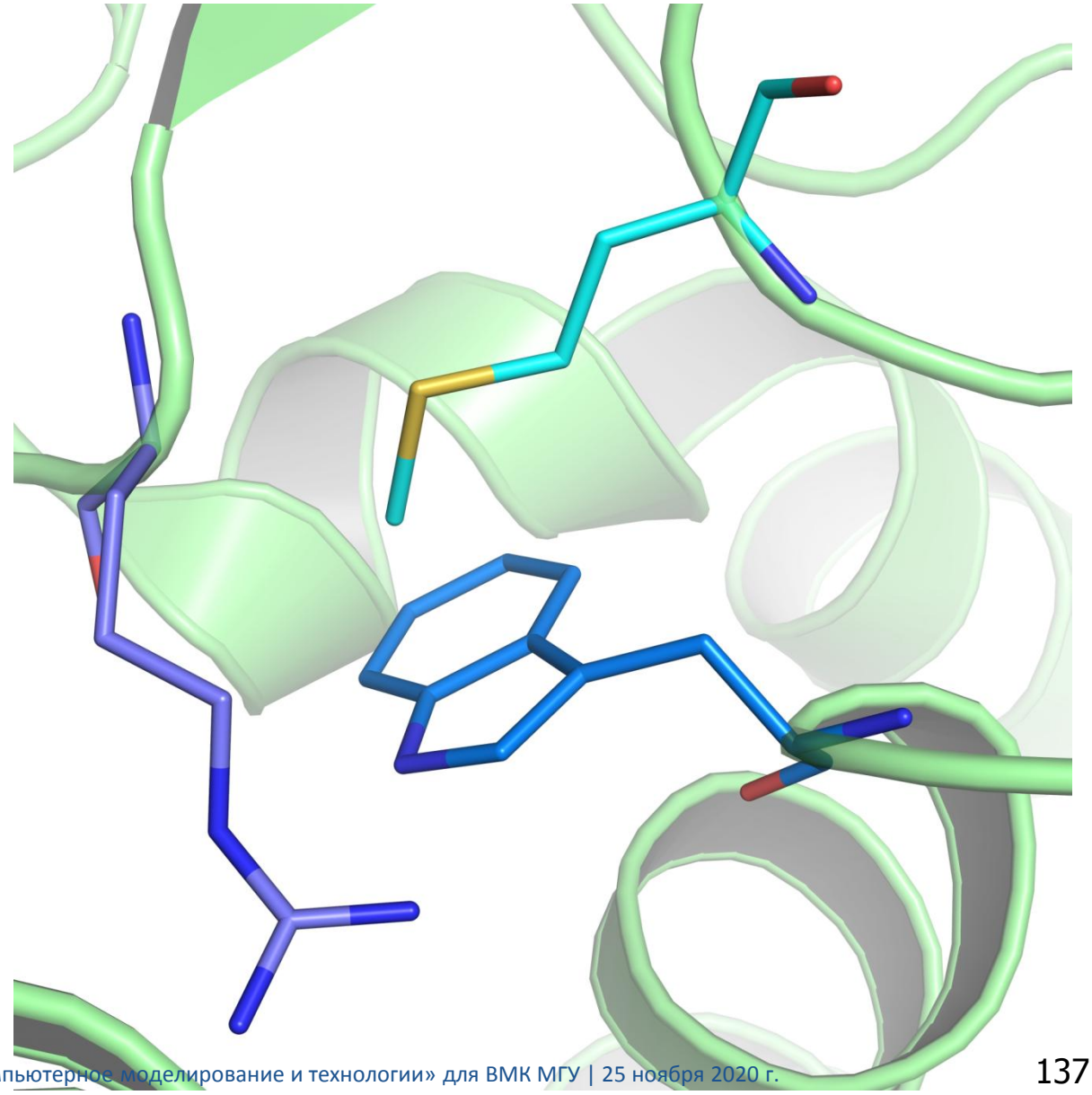
Для внесения амидазной активности в липазу

Липазы

I	L	C
I	L	C
I	L	C
I	L	C
I	L	C
I	L	C

Амидазы

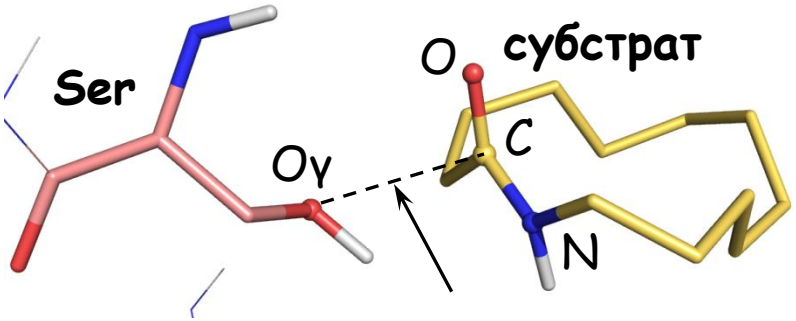
M	R	W
M	R	W
M	R	W
M	R	W
M	R	W
M	R	W



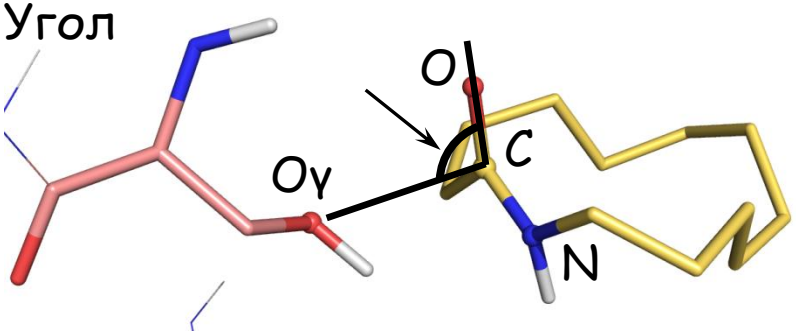
Молекулярное моделирование фермент-субстратных комплексов

Использование геометрических критериев амидазной каталитической активности для отбора наиболее перспективных образцов фермента

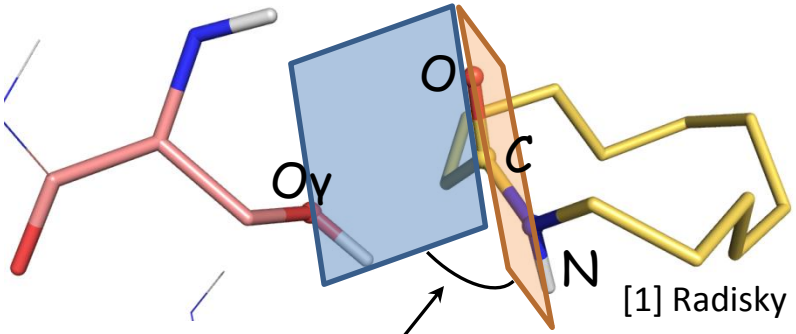
1. Расстояние



2. Угол



3. Двугранный угол



In silico библиотека:

10^3-10^5 мутантов

2.72 – 2.9 Å

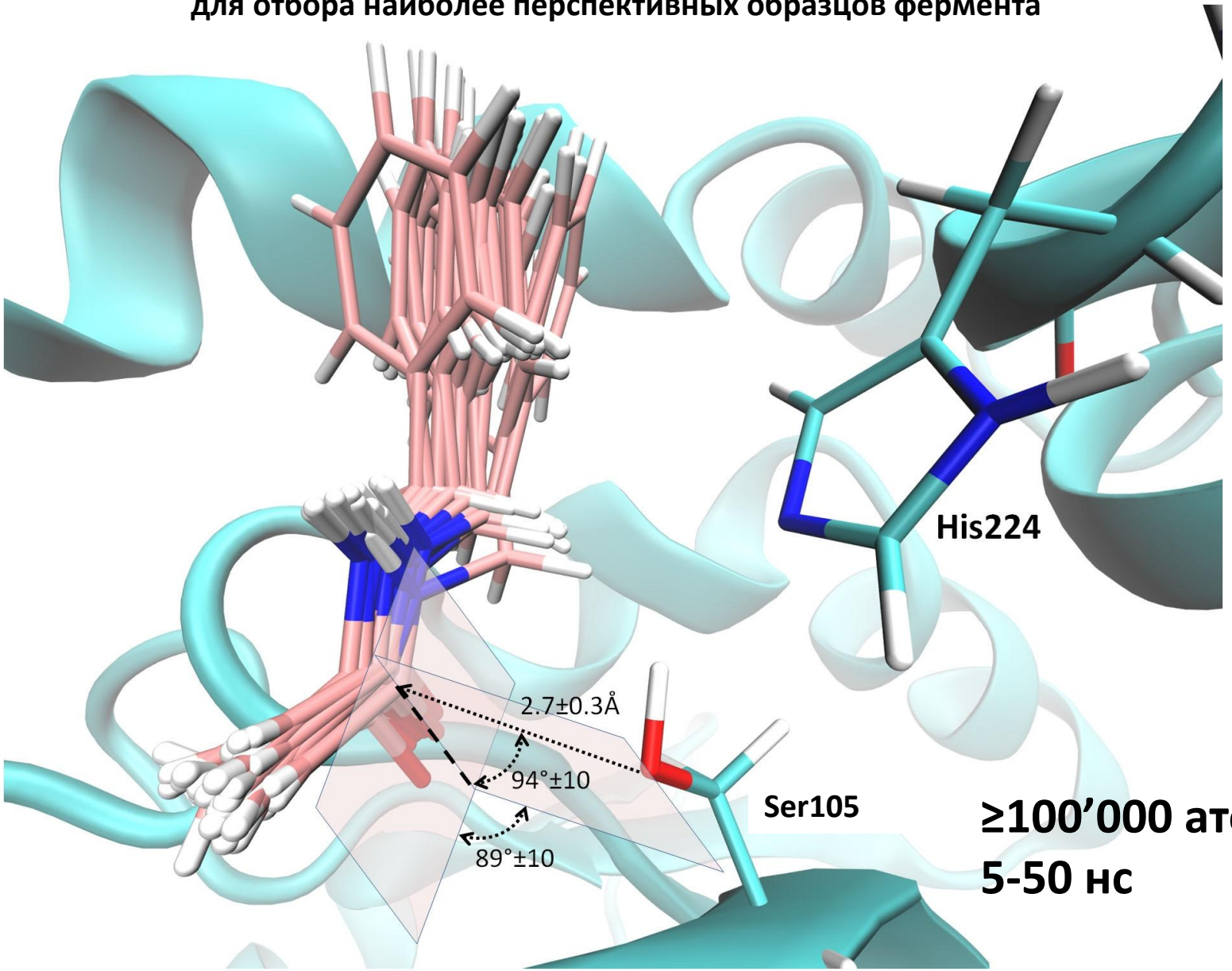
89-92 °

84-94 °

[1] Radisky E.S. & Koshland D.E. Jr., Proc Natl Acad Sci USA, 2002

Молекулярное моделирование фермент-субстратных комплексов

Использование геометрических критериев амидазной каталитической активности для отбора наиболее перспективных образцов фермента



≥100’000 атомов
5-50 нс

Амидазная активность мутантов липазы Б

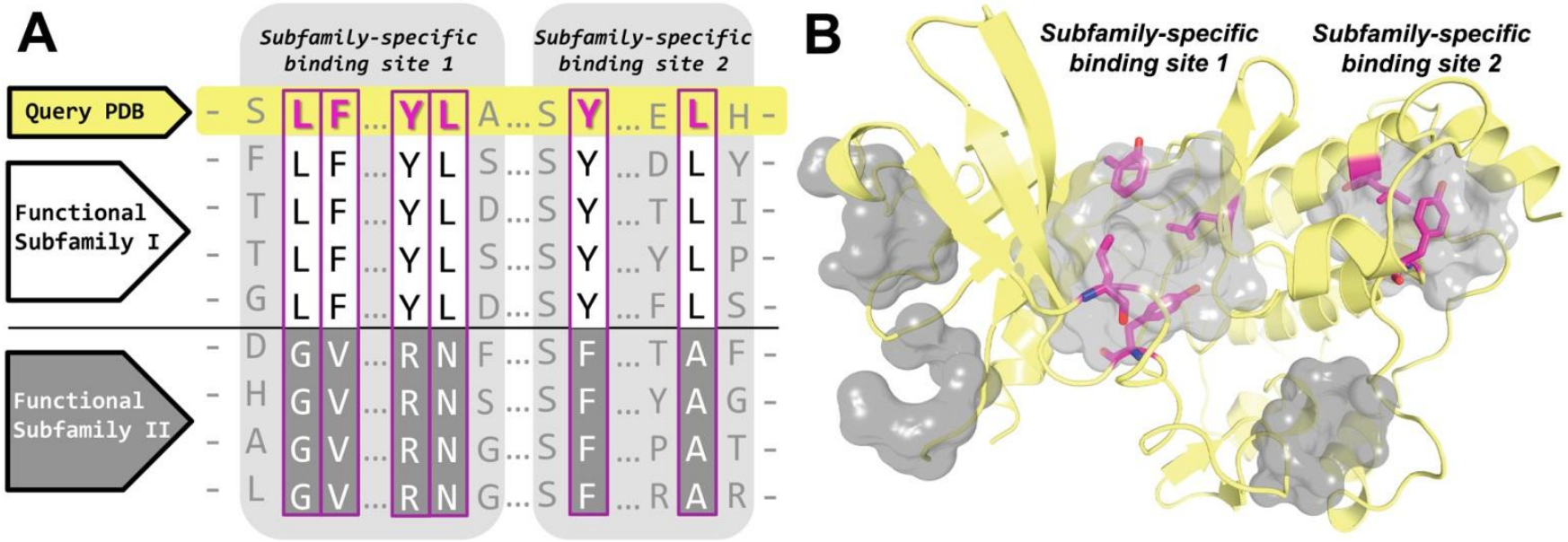
Измеренная *in silico* и экспериментальными методами

novozymes®
Rethink Tomorrow

Мутант	Моделирование							Эксперимент
	Субстрат C(N)=O RMSD, Å	Расстояние, Å				Угол, °		
		Ser105 γO ··· C	Thr40 N ··· O	Thr40 γ1O ··· O	Gln106 N ··· O	$A_{O\cdots C=O}$	$D_{O\cdots C(N)=O}$	
G39A T103G W104F L278A	0.85±0.33	3.11±0.23	3.34±0.61	2.79± 0.22	3.79± 0.76	94.99±10.75	94.84±14.73	x 11.2
G39A T103G W104Y L278A	0.89±0.26	3.28±0.23	3.07±0.22	2.78± 0.20	3.57± 0.32	91.07±8.06	111.73±11.00	x 2.8
G39A T103G W104Q L278A	0.93±0.37	3.19±0.25	3.26±0.35	3.49±0.63	3.90±0.44	79.40±10.53	106.16±12.14	x 1.9
G39A W104F L278A	1.01±0.30	3.27±0.22	3.01±0.19	2.88± 0.24	3.46± 0.29	95.72±8.70	91.40±28.18	x 6.3
G39A W104F	1.19±0.49	3.26± 0.27	4.73±1.53	5.72± 1.00	5.18± 1.22	108.53± 27.84	91.76± 29.41	x 4.2
G39A T103G L278A	1.88±0.55	3.48±0.34	5.39±0.82	4.12± 0.70	6.39± 0.89	133.55±22.79	89.10±22.43	x 3.8
G39A L278A	1.91±0.44	3.34± 0.28	5.15± 0.64	3.91± 0.60	6.11± 0.66	126.06± 20.27	88.87± 13.79	x 3.3
G39A T103G	2.29±0.79	3.45±0.56	5.46±1.09	4.10± 0.92	6.19± 1.01	123.60±25.19	78.94±46.04	x 0.8
Фермент дикого типа	2.67±1.20	4.52±0.96	6.92±1.47	5.18±1.19	7.40±1.29	137.23±23.82	68.67±136.28	1.0
G39A W104F A225K L278A	3.13±0.96	5.34± 1.04	6.46± 1.03	4.79± 1.01	8.10± 1.36	148.14± 21.18	70.15±61.19	x 0.8

Suplatov, D. A., Besenmatter, W., Švedas, V. K., & Svendsen, A. (2012). **Bioinformatic analysis of alpha/beta-hydrolase fold enzymes reveals subfamily-specific positions responsible for discrimination of amidase and lipase activities**. *Protein Engineering Design and Selection*, 25(11), 689-697.

Annotation of a novel allosteric sites by bioinformatics analysis



- Known catalytic and allosteric sites in enzymes are enriched by subfamily-specific positions;
- A scoring function based on SSPs can be used to **rank binding sites** by functional/regulatory significance;

Suplatov et al., *Nucleic Acids Res.*, 2014

Annotation of a novel allosteric site in Glyceraldehyde 3-phosphate dehydrogenase from *M.tuberculosis*

- Three ligands were designed by molecular modeling that bind to the novel site and **inhibit the bacterial GAPHD but not the human homolog** at the tested concentrations;

- One of those ligands inhibits the growth of *Mycobacterium tuberculosis* H37Rv cell culture.

Suplatov D., Schmalhausen E., Muronets V., Švedas V. (2018) RU2661151 “Selective inhibitors of glyceraldehyde-3-phosphate dehydrogenase of Mycobacteria”

Благодарности

Участники междисциплинарного проекта

НИИ ФХБ имени А.Н.Белозерского МГУ, ФББ МГУ,

- Швядас Витас
- Суплатов Дмитрий Андреевич
- Шарапова Яна Александровна
- Тимонина Дарья Сергеевна
- Панин Николай Владимирович
- Дробот Виктор Валерьевич
- Никулин Максим Владимирович
- Муронец Владимир Израилевич
- Шмальгаузен Елена Викторовна

НИВЦ МГУ, ВМК МГУ

- Воеводин Владимир Валентинович
- Попова Нина Николаевна
- Лукашевич Наталья Валентиновна
- Шегай Максим Викторович

Institute of Organic Chemistry, Graz University of Technology (Австрия)
Kateryna Fesko

Благодарности

Финансовая поддержка

РФФИ №18-29-13060 МК – в части создания многоуровневой биоинформатической платформы

РФФИ №19-04-01297 А – в части разработки комплексной методологии для изучения конформационной пластичности участков связывания в структурах белков/ферментов

РФФИ №20-07-00970 А – в части разработки и оптимизации ресурсоемких приложений сравнительного анализа белков для высокопроизводительных вычислительных систем

РНФ №15-14-00069 – экспериментальные исследования

СУПЕРКОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ И ТЕХНОЛОГИИ



Применение суперкомпьютерного моделирования для медицины

Сулимов Владимир Борисович

Научно-Исследовательский Вычислительный Центр МГУ

Лекция № 1

Научно-исследовательский вычислительный центр МГУ имени М.В.Ломоносова

Лаборатория вычислительных систем и прикладных технологий программирования

- Разработка программ докинга
- Разработки новых лекарств путем поиска ингибиторов заданных белков-мишеней в базах данных соединений
- Применение методов квантовой химии
- Отбор молекул в базах данных, содержащих миллионы лекарственно подобных органических молекул
- Разработка новых противовирусных препаратов прямого действия на белки-мишени коронавируса SARS-CoV-2
- Разработка и применение байесовских сетей для персонализированной медицины

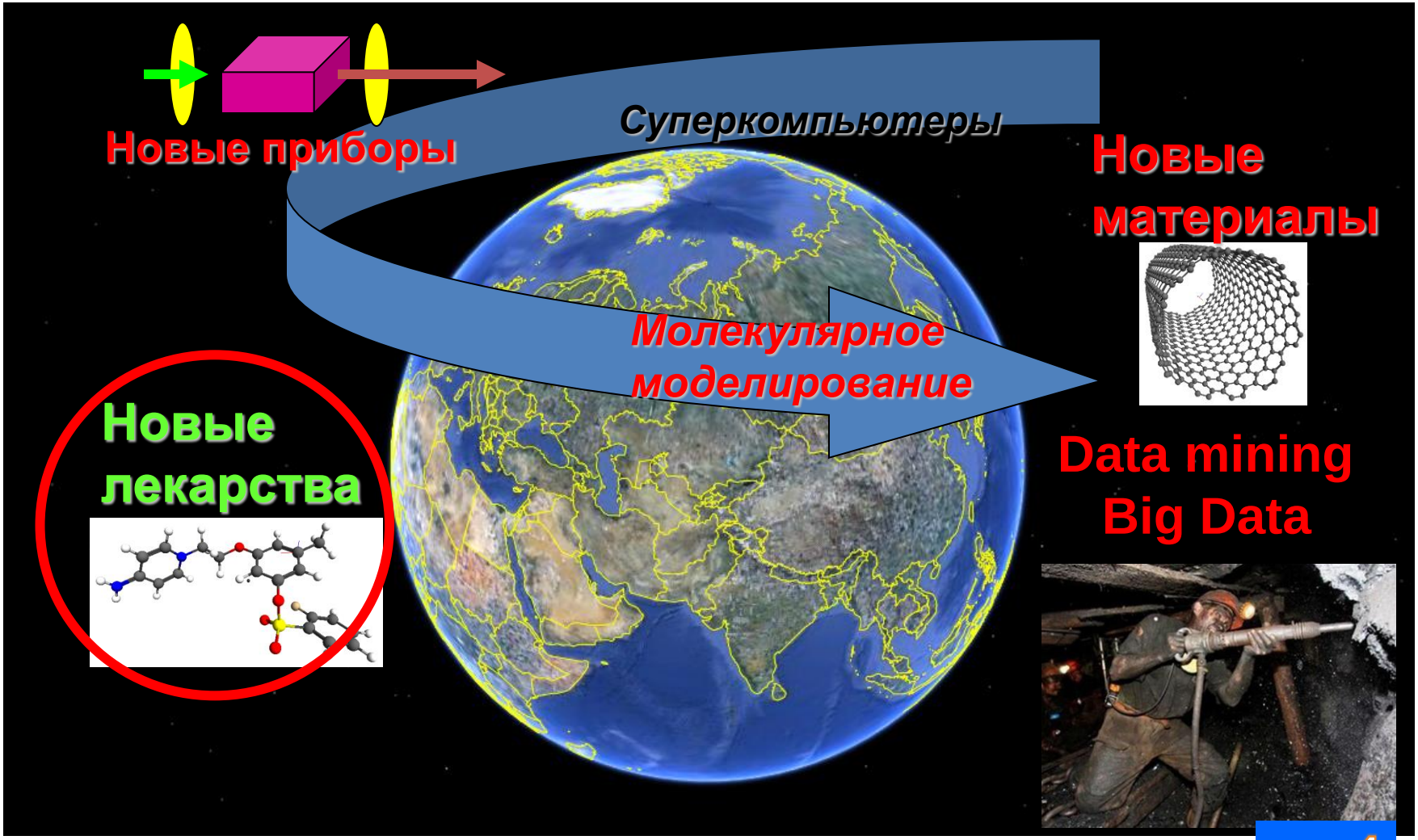
Суперкомпьютер МГУ Ломоносов-2 (2017) производитель: Т-платформы (Россия)



Пиковая производительность: 4,946 Пфлопс
Число вычислительных ядер: 64 384

Top50 – #2
Top500 – #130
(June 2020)

Мощные **суперкомпьютеры** дали начало Новой Эры в моделирование на атомно-молекулярном уровне и при обработке информации – data mining – Big Data



Пандемия COVID-19 высветила проблемы с разработкой лекарств

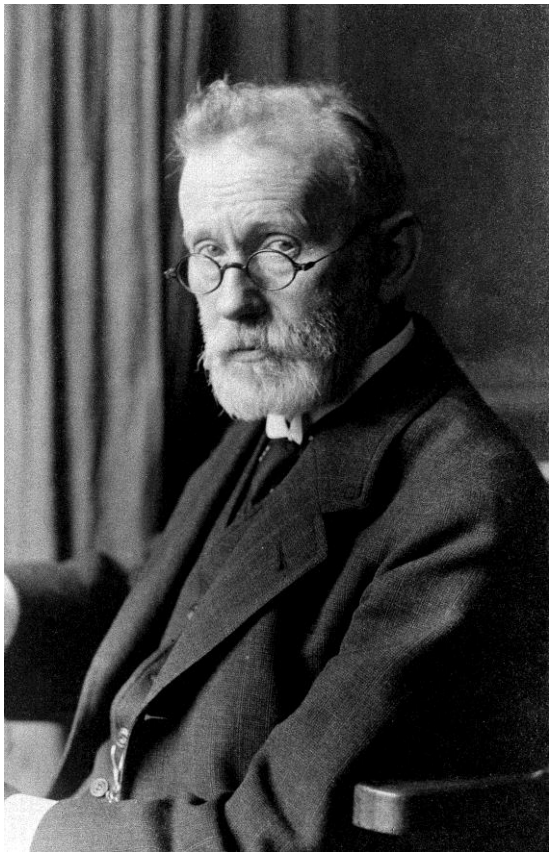
- Государство не контролирует, не планирует и не управляет разработкой лекарств
- Разработкой лекарств в основном занимаются транснациональные фармкомпании
- Рациональная разработка недостаточно отработана
- Начальный этап разработки в основном проводится экспериментально, методом проб и ошибок, часто вслепую
- Компьютерное моделирование пока не играет существенной роли в успехе разработки лекарств
- Суперкомпьютерные вычислительные мощности выросли до уровня достаточного для практическом применении молекулярного моделирования при разработках лекарств

Естественные науки

- Математика
- Механика
- **Физика**
- **Химия**
- Кибернетика
- **Биофизика, Биохимия, Биология**

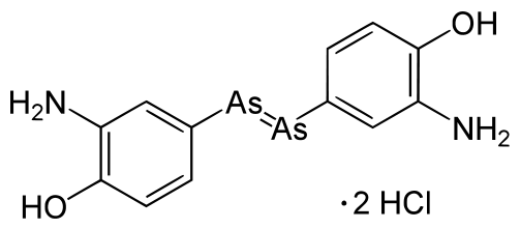
Важнейшая роль
всех этих наук на начальном этапе разработки нового лекарства

Пауль Эрлих (Paul Ehrlich) 1854 – 1915



- Нобелевский лауреат по Физиологии и Медицине 1908 год за работы в области иммунологии совместно с Ильей Ильичом Мечниковым
- Основоположник Химиотерапии и рациональной разработки новых лекарств

Разработал первый эффективный препарат для лечения сифилиса (1907-1910): Ehrlichschen **Präparat 606** или сальварсан активен в отношении бактерии – *бледной трепонемы – возбудителя сифилиса*; **Präparat 914** водорастворим.



Лекарство убивает бактерию, но не влияет на организм

Парадигма действия лекарства

- Для многих заболеваний известны белки, определяющие развитие патологии.
- Такие белки – мишени для действия лекарств
- Молекулы-ингибиторы лекарства **избирательно связываются с активным центром** белка-мишени и блокируют его работу
- Это приводит к излечению заболевания

Суперкомпьютерная разработка лекарства

Болезнь (COVID-19)

↓

Белок Вируса

↓

Активный Центр Белка

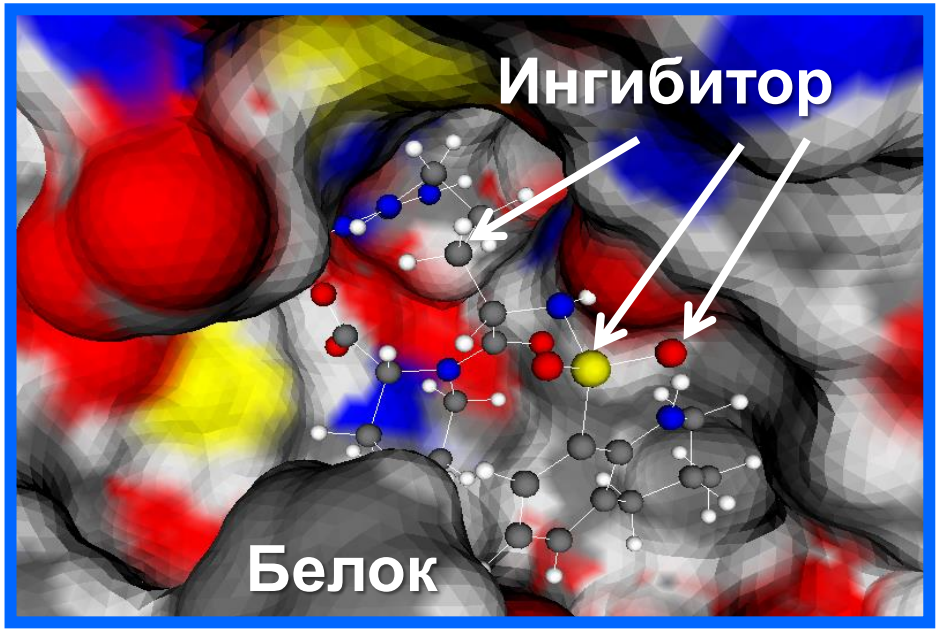
↓

Блокировка работы активного центра

↑

Молекула (Ингибитор)

Суперкомпьютеры помогают найти молекулу лекарства





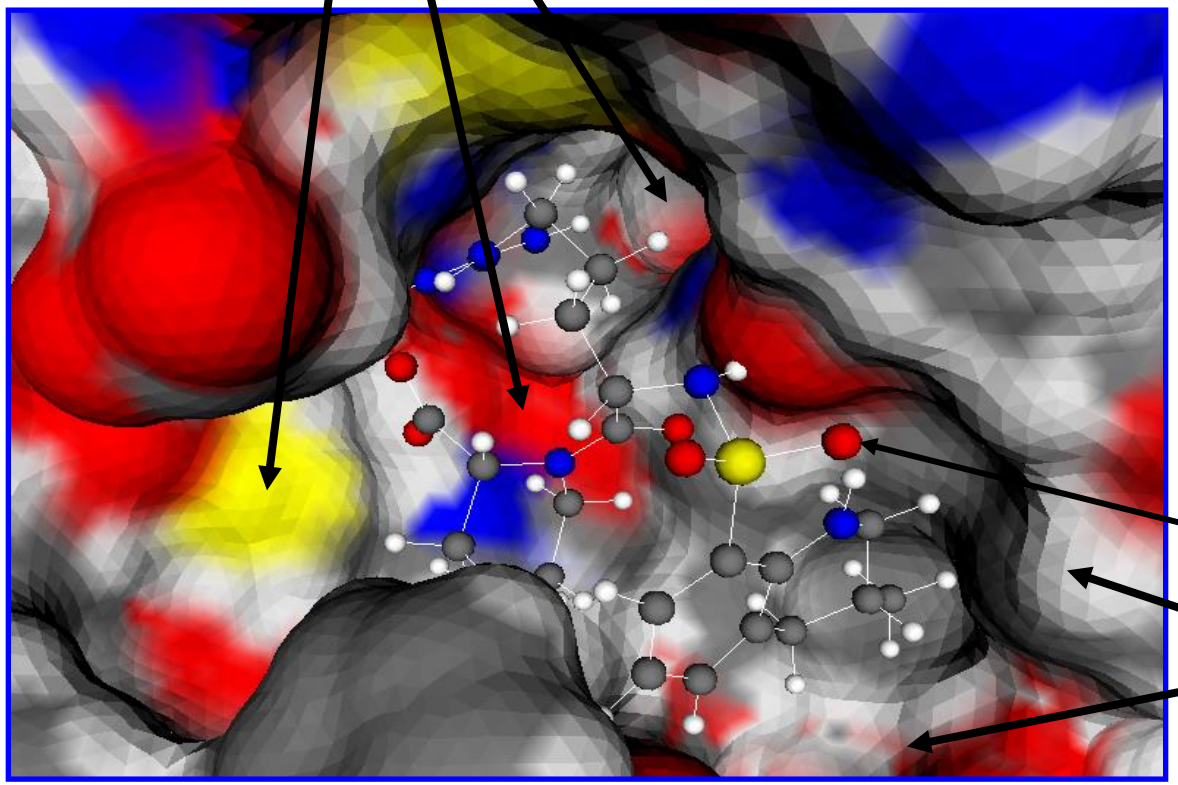
Блокирование работы белка осуществляется молекулами – ингибиторами

Для многих болезней известны белки-мишени

Основа нового лекарства – новые ингибиторы **$500 M**

Активный Центр Белка-Мишени

15 Лет!



50% времени затрачивается на разработку ингибиторов: экспериментально методом проб и ошибок

Ингибитор

Белок

Суперкомпьютеры ускоряют разработку ингибиторов

Новые лекарства

Новые ингибиторы – основа для открытия новых лекарств

Цель применения молекулярного моделирования: найти среди существующих молекул или придумать новую молекулу (лиганд) кандидат в ингибиторы заданного белка-мишени

Лиганд (Ligand) от латинского слова ligare = to bind: молекула, которая может связаться с биомолекулой

Рациональная разработка лекарств

Конвейер разработки нового лекарства

Поиск среди десятков тысяч молекул
Тестирование *in vitro*: белок + субстрат + молекула
Тестирование на клеточных культурах

Доклинические испытания на животных
Большой % отсева соединений из-за токсичности

Клинические испытания на людях
Большой % отсева соединений из-за токсичности

Стадии разработки нового лекарства

7-15 Лет! **$1000 M**

in vitro, in vivo

или

in silico

5,000 to 100,000 соединений - перебор

250 Кандидатов в доклинические испытания



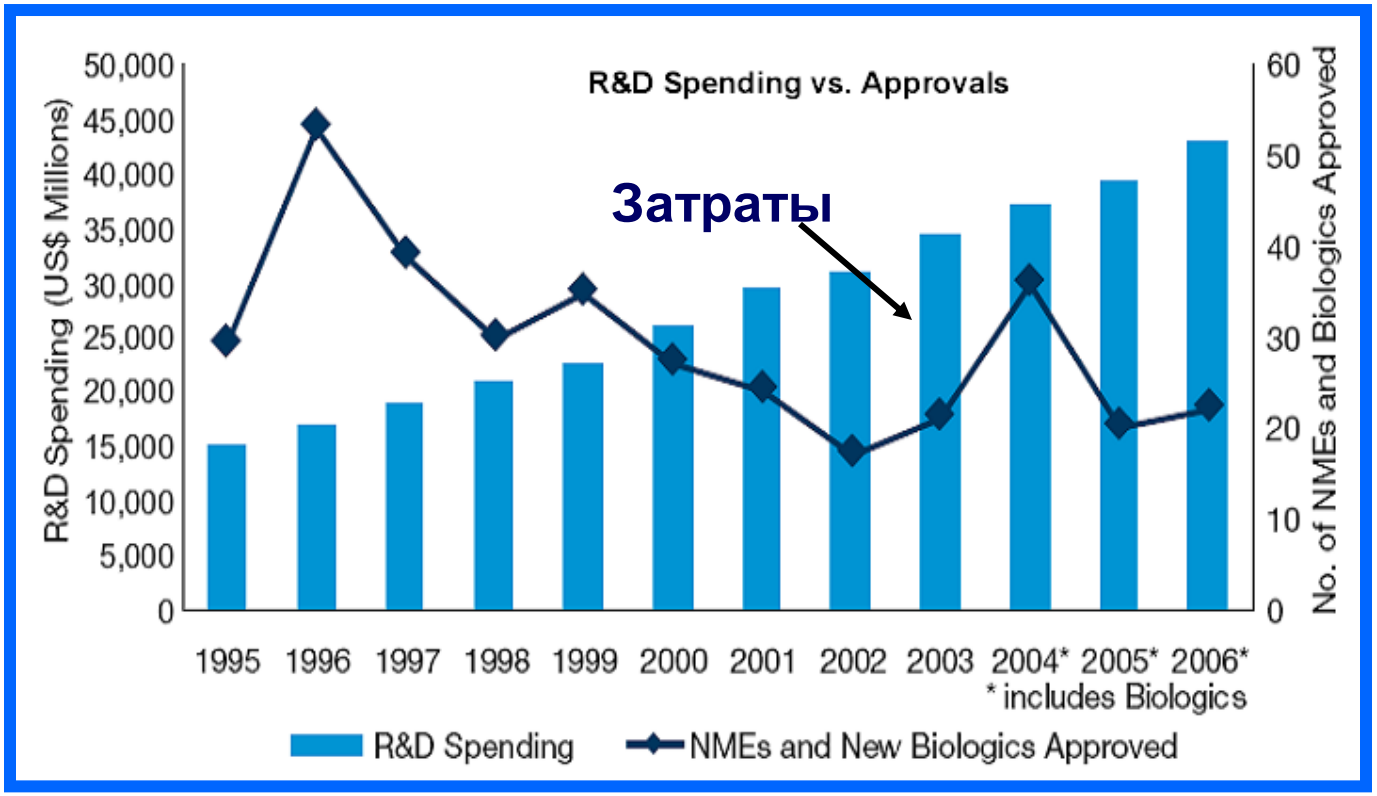
5 кандидатов – клинические испытания

- **80% проходят фазу I**
- **30% проходят фазу II**
- **80% проходят фазу III**

1 соединение становится разрешенным лекарством

Существующие методы поиска ингибиторов ведут в тупик !

Затраты на исследования растут линейно



NMEs

New Molecular Entities

Новые Молекулярные Объекты

Число новых лекарств – новых химических соединений, ежегодно появляющихся на рынке сокращается

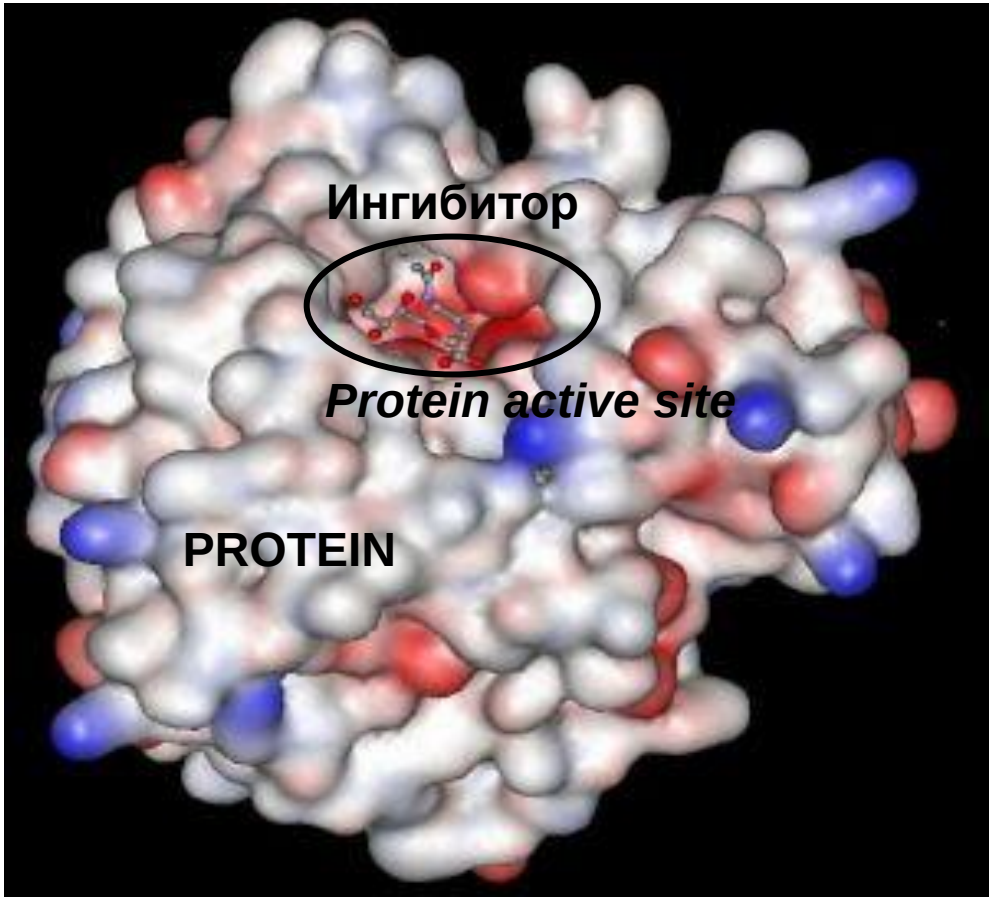
Выход из тупика есть!

Молекулярное Моделирование

- Вычисление межмолекулярных взаимодействий
- Современная теория растворов – практика вычислений взаимодействия молекул в растворах
- Новые алгоритмы поиска глобального минимума на сложной многомерной энергетической поверхности
- Использование суперкомпьютеров – многопроцессорные параллельные вычисления
- Молекулярная механика
- Молекулярная динамика
- Квантовая химия
- Компьютерный перебор больших баз данных органических соединений

Поиск и разработка новых ингибиторов – молекул, специфически связывающихся с заданным белком-мишенью

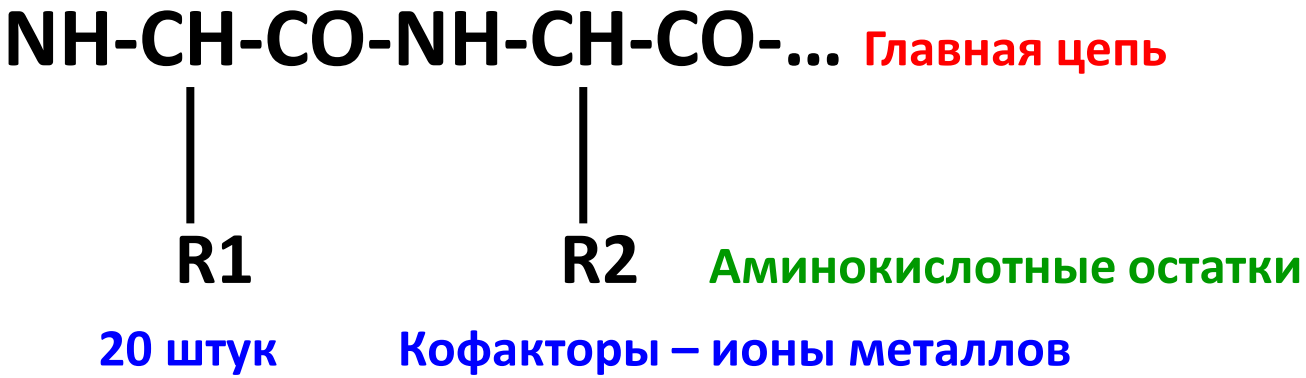
- Белки – тысячи атомов
- Лиганды – десятки атомов
- Слабые межмолекулярные взаимодействия
- В водном растворе
- Нужна **высокая точность** вычислений энергии взаимодействия лиганда с белком ~ **1 kcal/mol**
- Поиск глобального **минимума энергии** на очень сложной энергетической поверхности



Influenza Neuraminidase / 2-deoxy-N-acetylneuraminic acid complex

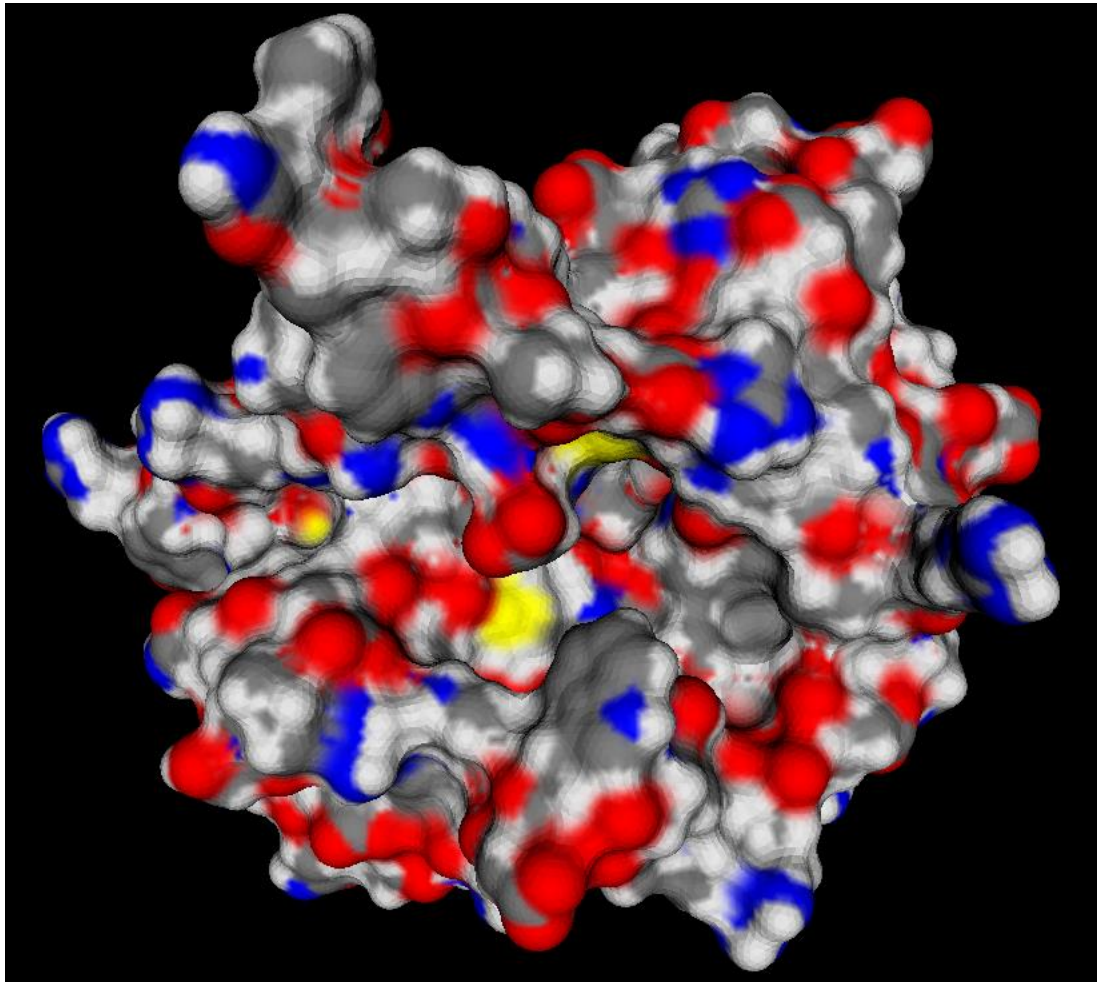
Молекулы

- Белки – протеины – тысячи атомов
  - Полипептидная цепь:



- Органические молекулы – десятки атомов: миллиарды молекул: C, N, O, H, S, Cl, F, Br ...

Поверхность доступная растворителю вокруг белка тромбина



Здесь различные цвета различных областей поверхности соответствуют разным типам атомов, находящихся вблизи этих областей:

- **Красный – кислород**
- **Синий – азот**
- **Серый – углерод**
- **Белый – водород**
- **Желтый – сера**

Структуры молекул белков

- Белки кристаллизуются и методом рентгеноструктурного анализа экспериментально определяется пространственное расположение их атомов
- Определяются декартовы координаты «всех» атомов белка
- Декартовы координаты атомов комплексов белок-лиганд
- Информация о белках и комплексах белков с лигандами собрана в базе данных Protein Data Bank (PDB) http://www.rcsb.org/

Общее количество структур – 171588 (ноябрь 2020); 69511 в 2010

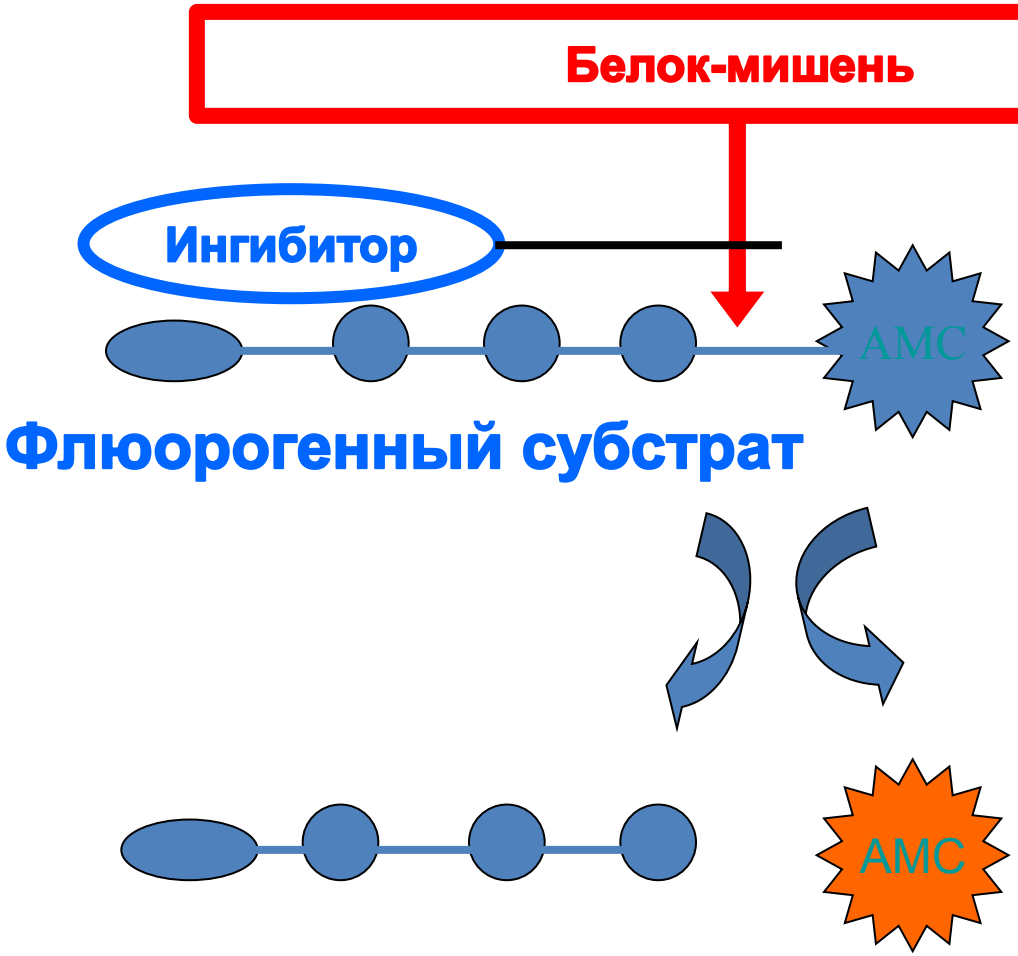
Белки, X-ray – 134334 структур (28 ноября 2020)

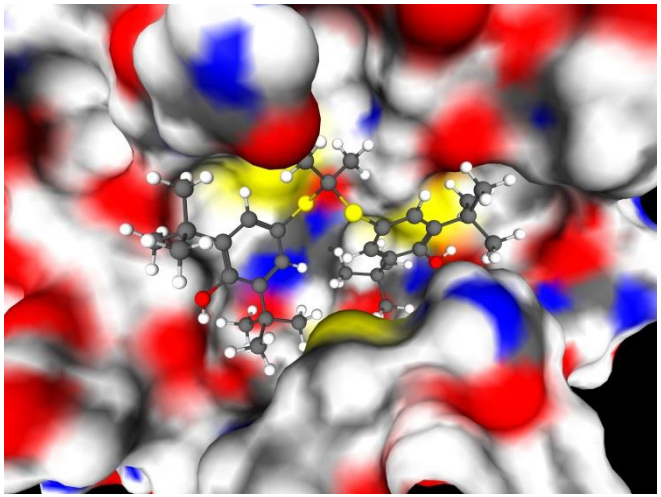
Resolution характеризует точность определения координат атомов в структуре кристалла. Чем меньше этот параметр тем лучше. Можно работать, когда Resolution < 3.0 Å.

Что измеряется в эксперименте? С чем сравнивать расчет?

- Измерение IC50 – ингибирующая концентрация: концентрация ингибитора, при которой активность белка подавляется на 50%
- Чем меньше IC50 , тем выше ингибирующая активность вещества: IC50 = 1 наномоль лучше, чем IC50 =1 микромоль
- Чем больше энергия связывания белок-лиганд, тем меньше IC50 и лучше ингибитор
- Белок вместе с ингибитором кристаллизуются, применяется метод рентгеноструктурного анализа для определения координат атомов комплекса белок-лиганд и определения положения лиганда в активном центре белка

Измерение активности (IC50) ингибиторов белка-мишени





Флюоресцирующий продукт (AMC)

Докинг, скоринг и скрининг – основа конструирования лекарств

Молекулярное моделирование помогает быстрее и дешевле выполнять начальную стадию разработки

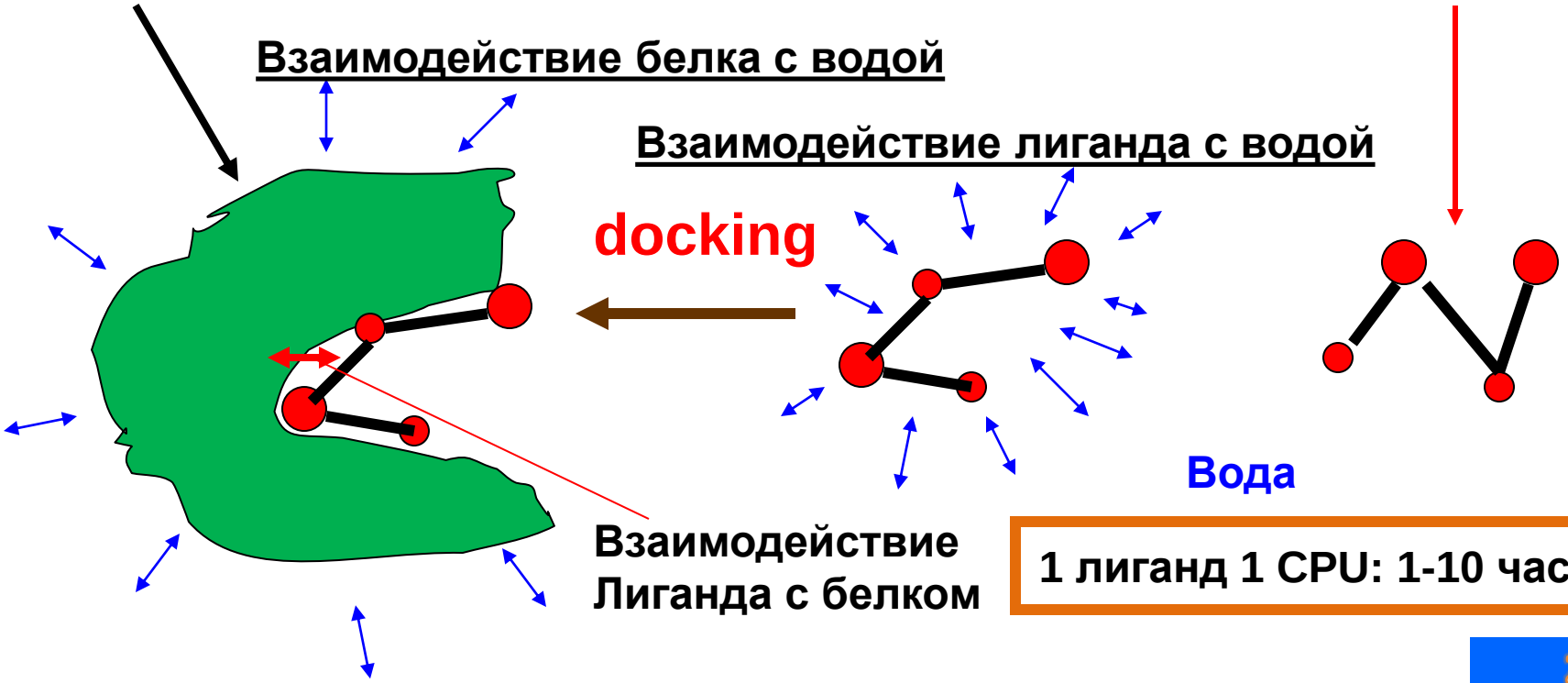
- **Докинг** – позиционирование лиганда в активном центре белка
- **Скоринг** – оценка энергии связывания лиганда с белком: чем сильнее лиганд связывается с белком, тем эффективнее лекарство – меньшая концентрация лекарства даст терапевтический эффект
- **Скрининг** – перебор больших баз данных молекул с целью поиска кандидатов в ингибиторы – **нужны суперкомпьютеры**

Докинг – позиционирование лиганда в активном центре белка
Скоринг – оценка энергии связывания

Нужна высокая точность: ~ 1 kcal/mol ~ 0.05 eV

Белок: тысячи атомов

Лиганд: десятки атомов



1 лиганд 1 CPU: 1-10 часов

Программы докинга стали популярны. Десятки программ докинга востребованы при разработке лекарств. Каждая программа – уникальное сочетание моделей и приближений, скорости

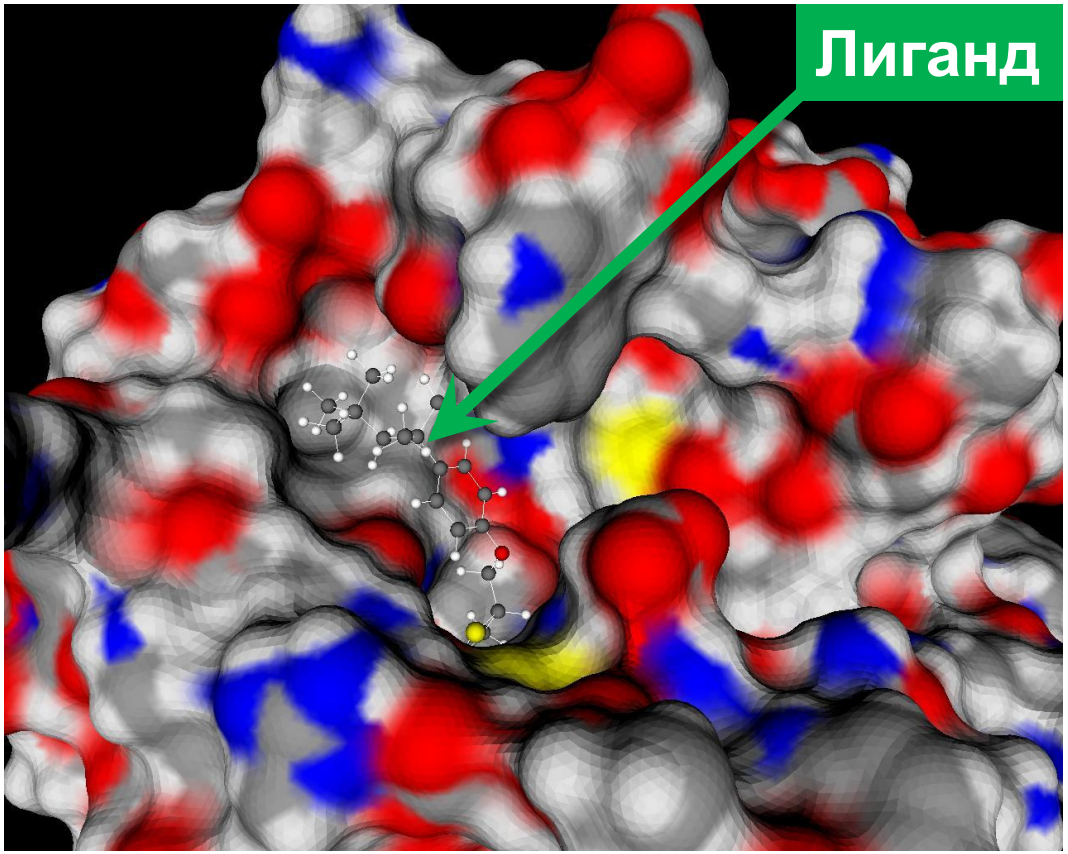
Возможно ли увеличить точность докинга?

- **Точность позиционирования – *удовлетворительна***
- **Точность расчетов энергии связывания белок-лиганд ΔG_{bind} – *плохая***

Точность докинга → **Эффективность разработки лекарств**

Перебор больших баз данных органических молекул:
Поиск молекул с наибольшей энергией связывания с белком-мишенью

Парадигма докинга: в эксперименте лиганд связывается вблизи **глобального минимума** энергии системы белок-лиганд



Докинг – нахождение Глобального минимума энергии системы белок-лиганд

Межмолекулярное взаимодействие

- Правильный учет сил между молекулами – основа успеха молекулярного моделирования лекарств
- Эти силы могут быть правильно рассчитаны только методами квантовой механики
- Силы между атомами и структура молекул правильно рассчитываются только методами квантовой механики
- Вывод: конструировать лекарства надо методами квантовой механики

Молекулярная механика

- Квантовая механика и квантовая химия
- Точные законы – очень трудоемкие вычисления – одна молекула из нескольких десятков атомов – десятки и сотни часов
- Что делать?
- Создать силовое поле (force field) – набор потенциалов, описывающих взаимодействие атомов и молекул, как классических частиц: расчеты – десятки минут для тысяч атомов

Силовое поле MMFF94 – создано на основе квантово-механических расчетов

$$U_{MMFF} = \sum EB_{ij} + \sum EA_{ijk} + \sum ESB_{ijk} + \sum EOOP_{ijk;l} + \sum ET_{ijk;l} + \sum EVdW_{ij} + \sum EQ_{ij}$$

$$EQ_{ij} = 332.0716 \frac{q_i q_j}{R_{ij}}$$ – электростатические взаимодействия

$$EVdW_{ij} = \varepsilon_{IJ} \left(\frac{1.07 R_{IJ}^*}{R_{ij} + 0.07 R_{IJ}^*} \right)^7 \left(\frac{1.12 R_{IJ}^{*\,7}}{R_{ij}^{\,7} + 0.12 R_{IJ}^{*\,7}} - 2 \right)$$ – Силы Ван дер Ваальса

$$EB_{ij} = 143.9325 \frac{kb_{ij}}{2} \Delta r_{ij}^{\,2} \left(1 + cs \Delta r_{ij} + 7/12\, cs^{\,2} \Delta r_{ij}^{\,2}\right)$$ – Изменение дли связей

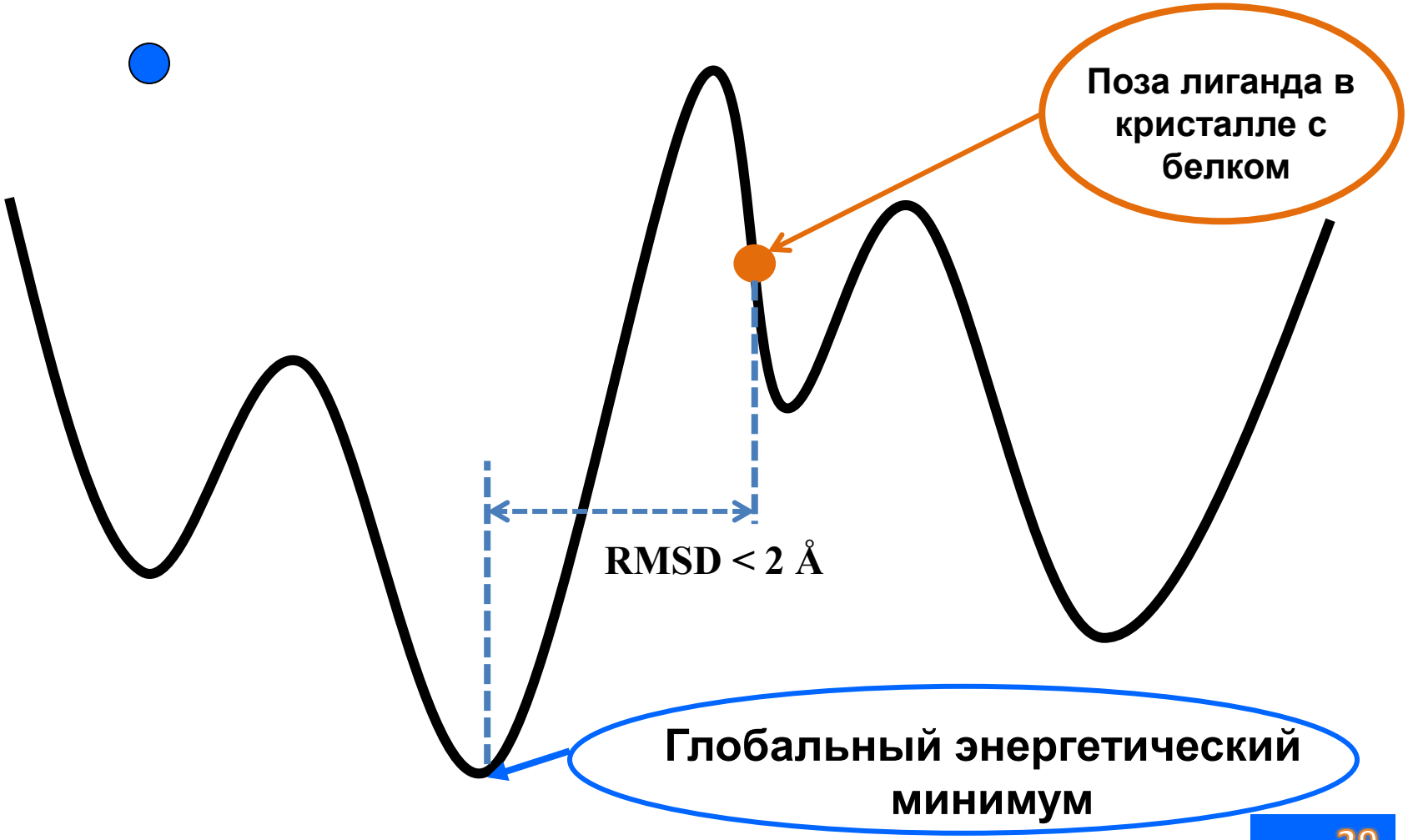
$$EA_{ijk} = 0.043844 \frac{ka_{ijk}}{2} \Delta \vartheta_{ijk}^{\,2} \left(1 + cb \Delta \vartheta_{ijk}\right)$$ – Деформация валентных углов

$$ESB_{ijk} = 2.51210 \left(kba_{ijk} \Delta r_{ij} + kba_{kji} \Delta r_{kj}\right) \Delta \vartheta_{ijk}$$ – Деформация и углов, и связей

$$ET_{ijkl} = 0.5\left(V_1\left(1 + \cos \varphi\right) + V_2\left(1 + \cos 2\varphi\right) + V_3\left(1 + \cos 3\varphi\right)\right)$$ Изменение торсионных углов

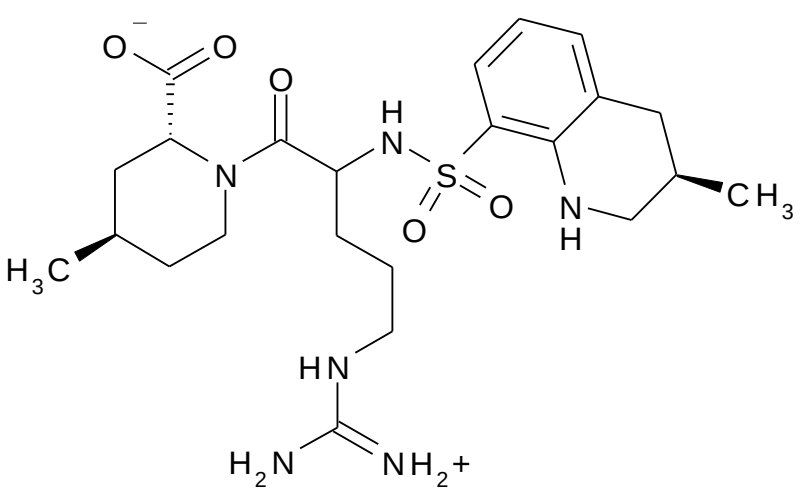
$$EOOP_{ijk;l} = 0.043844 \frac{koop_{IJK;L}}{2} \chi_{ijk;l}^{\,2}$$ – Выход атома I из плоскости 3х атомов I,j,k

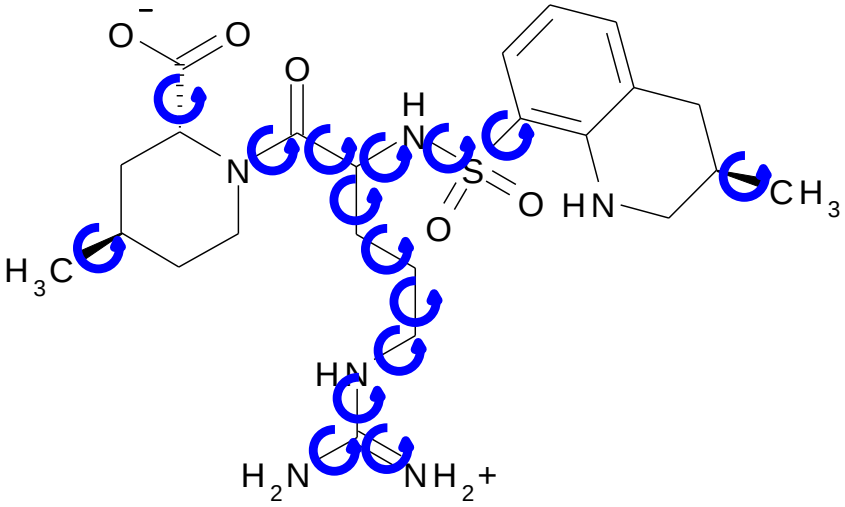
На многомерной энергетической поверхности системы белок-лиганд очень много локальных минимумов



Основная проблема докинга: многомерность пространства глобальной оптимизации

Типичный лиганд – ингибитор тромбина, Аргатробан, антикоагулянт – препятствует свертываемости крови





Получаем задачу на нахождение глобального минимума в пространстве 15 + 6 = 21 измерений!

Систематический перебор невозможен

Разработаны 3 программы докинга

- **Классическая программа докинга SOL**
  - **SOL** сравнима с другими программами докинга десять тысяч лигандов в неделю на Ломоносов-2

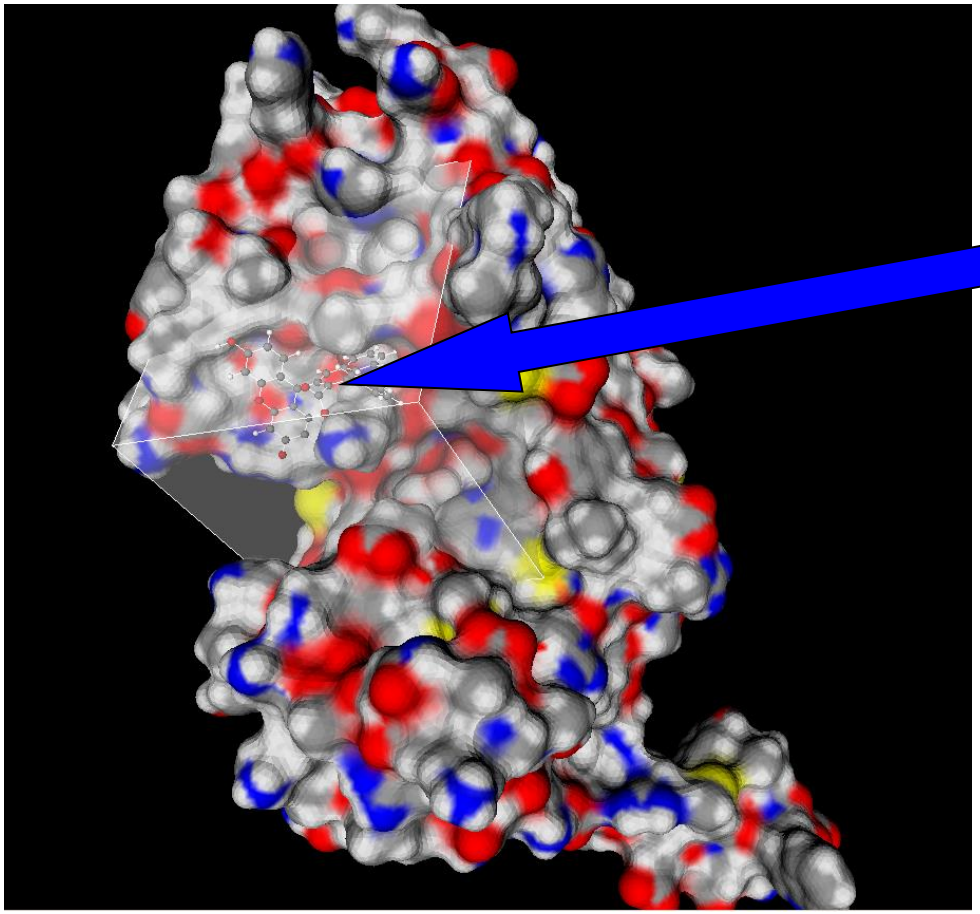
- **Суперкомпьютерные программы обобщенного докинга: находят спектр низкоэнергетических минимумов**
  - **FLM**, массивные параллельные локальные оптимизации энергии из случайных начальных поз лиганда в жестком белке:
    - ✓ Дает возможность сравнивать разные алгоритмы и энергетические функции
    - ✓ Квазидокинг с помощью квантово-химических методов
  - **SOL-P**, алгоритм глобальной оптимизации на основе тензорных поездов
    - ✓ Докинг очень гибких лигандов - олигопептидов
    - ✓ Докинг гибких лигандов в белки с подвижными атомами

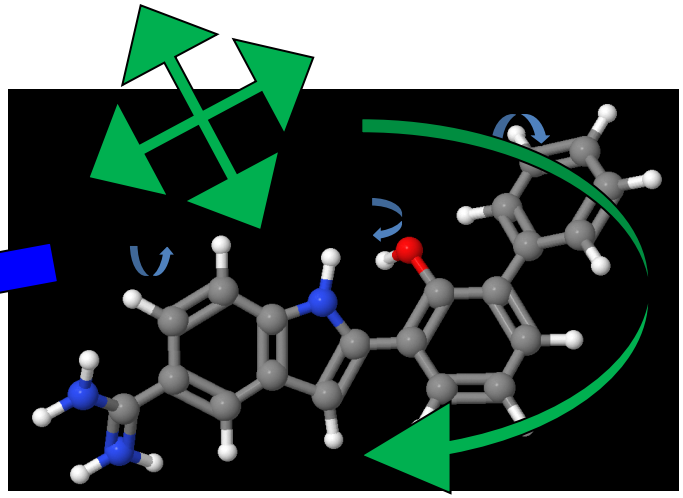
Оригинальная программа докинга SOL – классический докинг

Позиционирование лиганда в активном центре заданного белка-мишени

- Жёсткий белок и гибкий лиганд
- Генетический алгоритм поиска положения лиганда в белке, соответствующего самой низкой энергии системы белок-лиганд
- Универсальное силовое поле MMFF94 фирмы Merck
- Активный центр белка в виде набора сеток различных потенциалов – жесткий белок
- Учет влияния растворителя - воды
- Гибкий лиганд и учет его внутренних напряжений

Программы докинга: позиционирование молекул в активном центре белка-мишени





<u>Десятки тысяч молекул</u>:

- **3 трансляции как целое тело**
- **3 вращения как целое тело**
- **$\leq$ <u>15 внутренних вращений</u>**

<u>Белок</u>: жесткий

Открытие ингибиторов: тромбина, урокиназы (uPA), факторов свертываемости крови Xa, XIa

Модель белка – программа SOLGRID

- Белок жесткий
- Белок представлен сеткой потенциалов:
  - Кулоновские взаимодействия
  - Ван-дер-ваальсовы взаимодействия
  - Взаимодействие с растворителем
- Сетка потенциалов вычисляется заранее
- Сетка – это набор «сеток» потенциалов для разных типов атомов и разных типов взаимодействий
- Область докинга: куб с ребром 22 Ангстрема
  - 101 x 101 x 101 точек
  - длину ребра куба можно менять

Сетка потенциалов

- Атом каждого типа помещается в узлы сетки и вычисляется энергия его взаимодействия со всеми атомами протеина
- Создаваемая GRID – это целый набор сеток
- При докинге энергия лиганда в поле белка вычисляется как сумма энергий каждого атома лиганда в поле белка
- Энергия каждого атома интерполируется по восьми ближайшим точкам сетки

Временные затраты при докинге

- Большинство существующих программ докинга тратят < 1 минуты на докинг одного лиганда – **много упрощений**
- В нашу программу SOL заложен принцип:

«Тише едешь – дальше будешь»

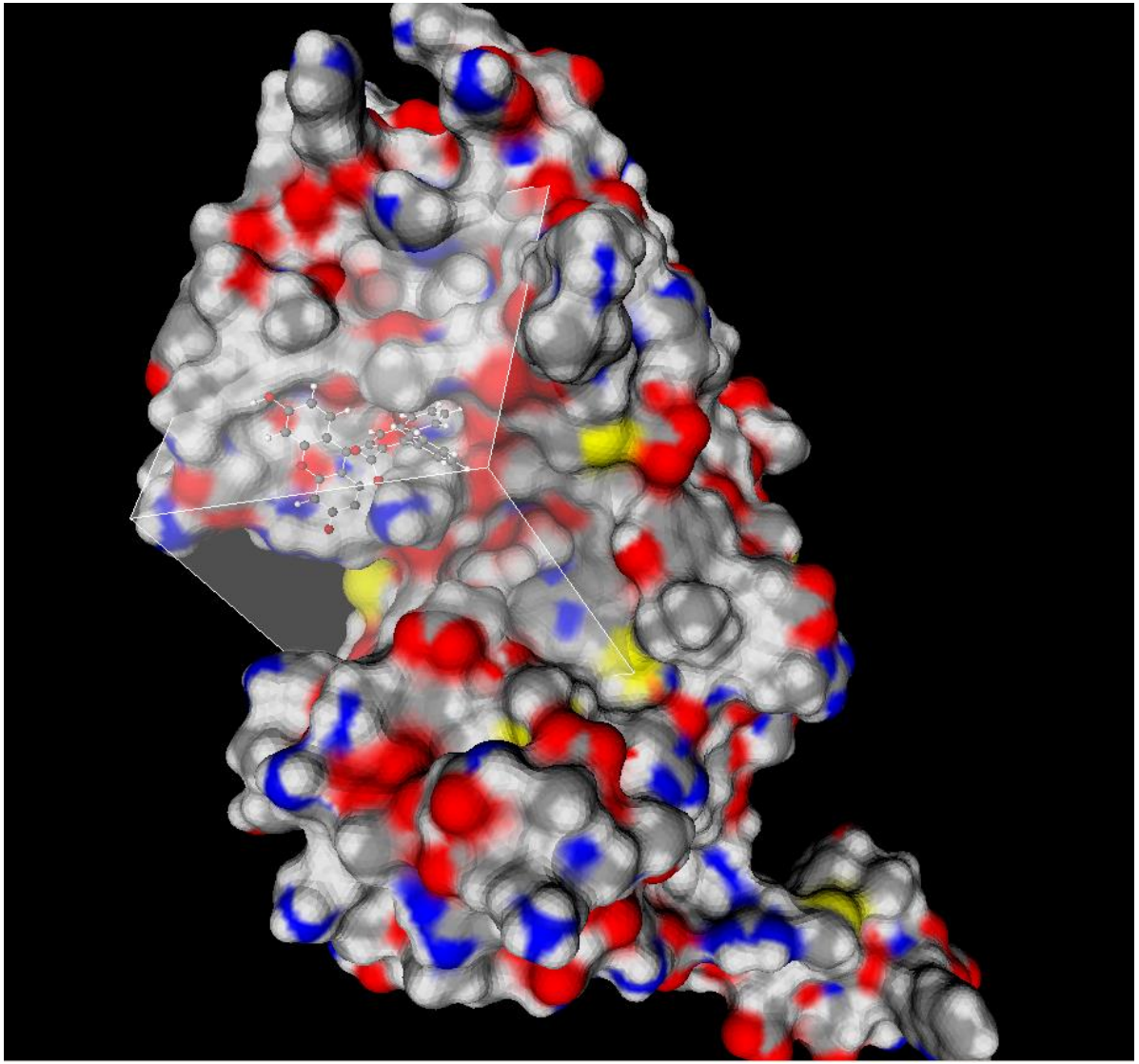
аккуратные вычисления: силовое поле MMFF94, учет растворителя, «крутые» параметры генетического алгоритма

На докинг одного лиганда: **1- 5 часов** на

1 CPU в зависимости от размеров и гибкости лиганда

- SOL адаптирована для суперкомпьютера: докинг десятков тысяч лигандов – 1 лиганд на 1 вычислительном ядре
- SOL многопроцессорный докинг 1 лиганда (минуты)

Программа SOLGRID

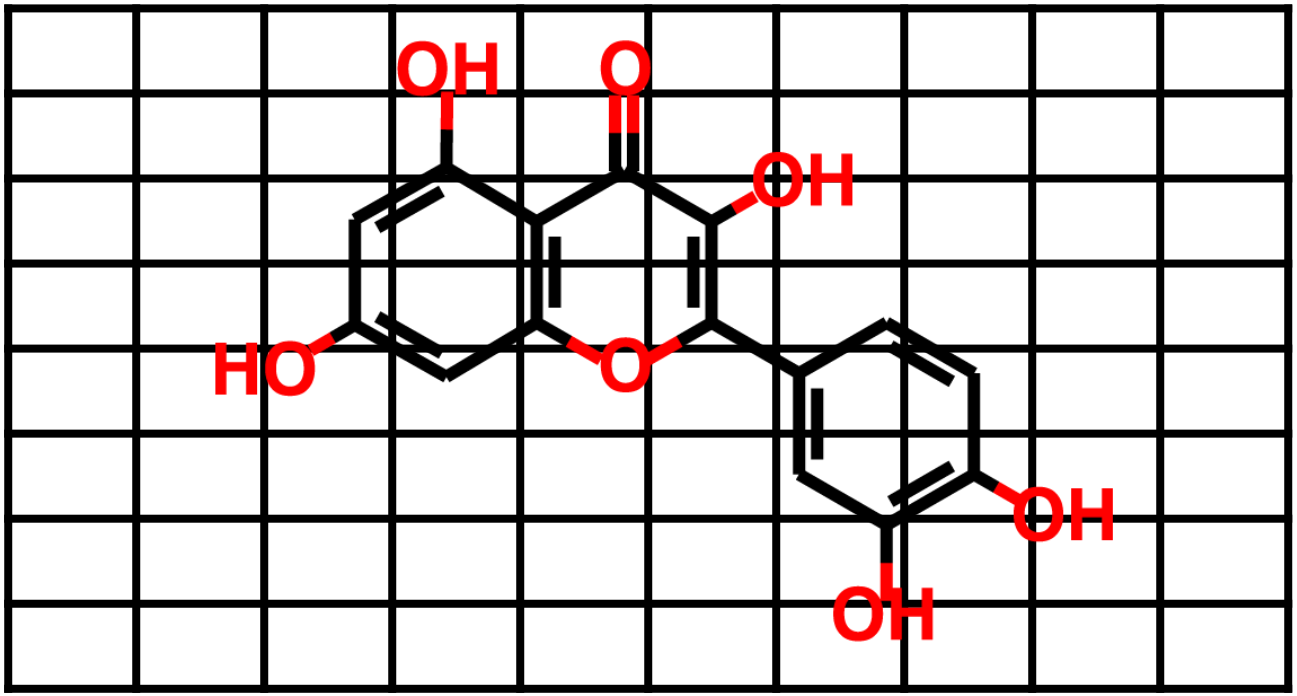


Вычисление энергии лиганда в поле белка

$$E_{lig-protein} = \sum_{i=1}^{N} E_i$$

N – Количество атомов в лиганде

E_i – энергия *i*-ого атома в поле белка



Если атом попадает между узлов сетки, то его энергия вычисляется в результате интерполяции по значениям энергии в **8 ближайших узлах**

Все потенциалы взаимодействия атомов лиганда с белком для всех типов атомов (C, N, H, S, O …) во всех узлах 3х-мерной сетки, покрывающей весь активный центр белка-мишени

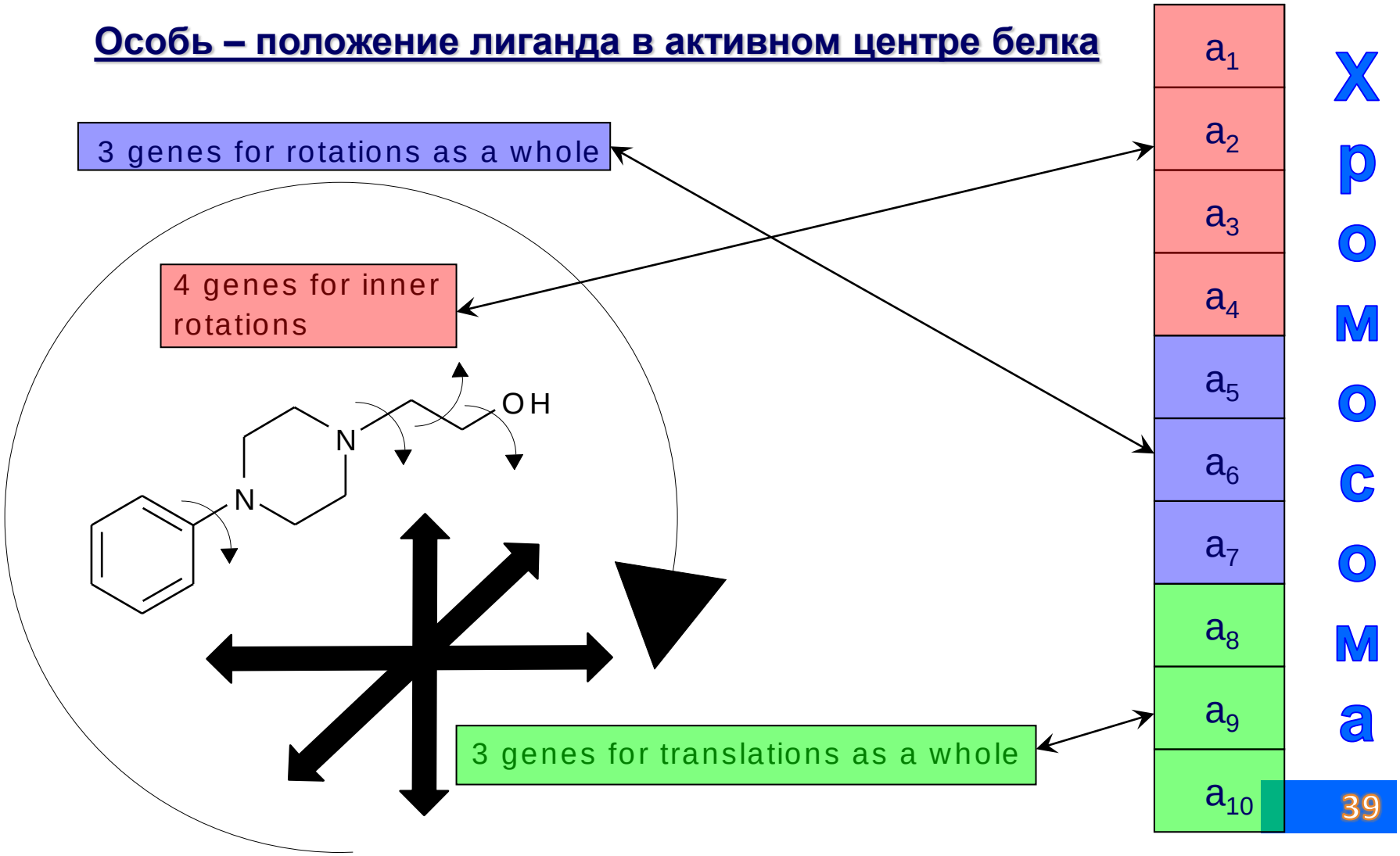
Программа докинга SOL

ФЕНОТИН ГЕНОТИН

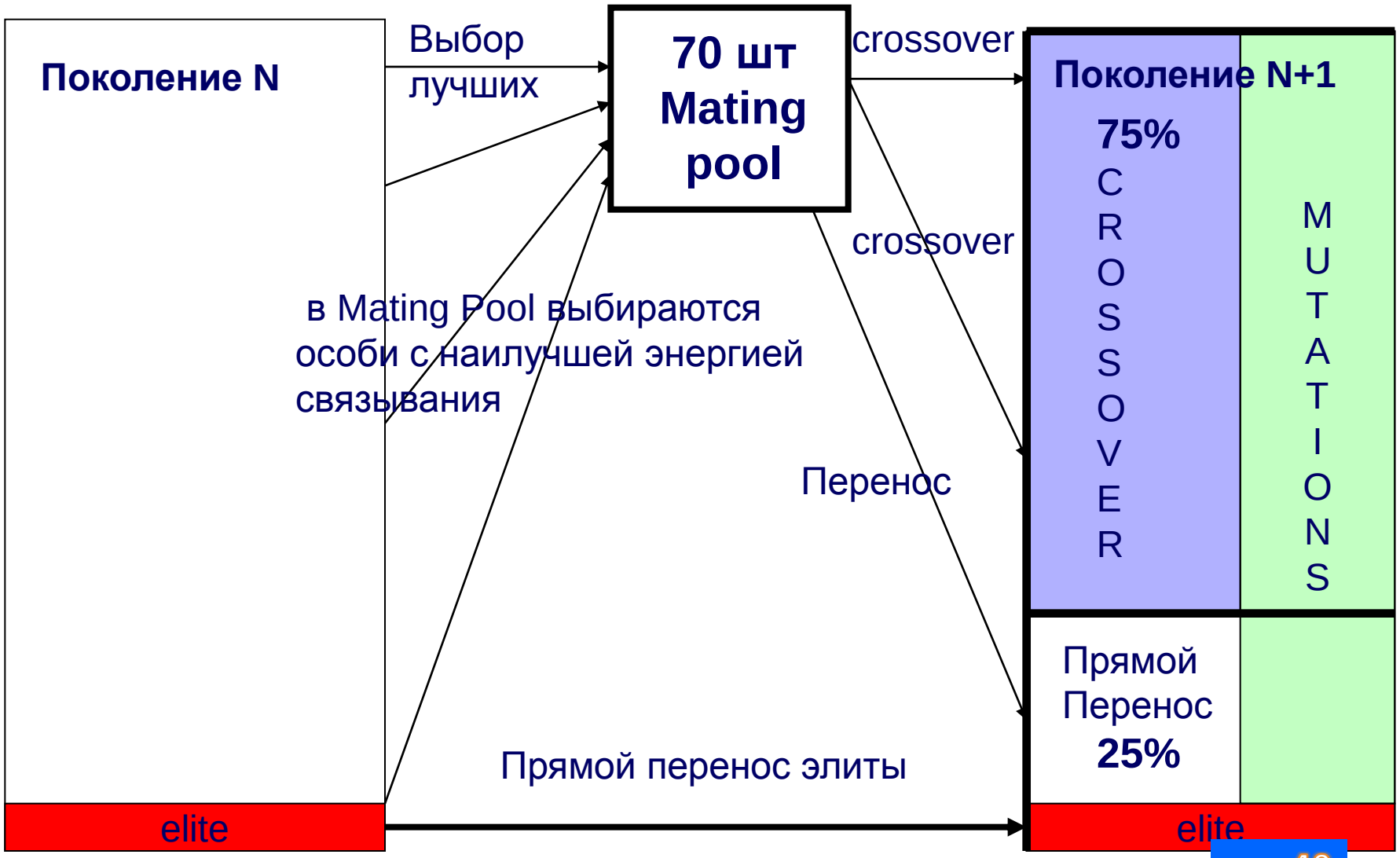
Эволюция популяции особей

$0 \le a_i \le 1$

Особь – положение лиганда в активном центре белка



Генетический алгоритм глобальной оптимизации



Поколение N → Поколение N+1

- **Прямой перенос элиты**
- **Отбор в Mating pool: 70 лучших по энергии особей с учетом нишинга**
- **Случайно выбранная в Mating pool пара особей с помощью crossover'а создает 1 особь в новом поколении – 75% всего нового поколения**
- **25% нового поколения – прямой перенос случайно выбранной в Mating pool особи**
- **Все особи нового поколения подвергаются мутациям**

Нишинг (Niching)

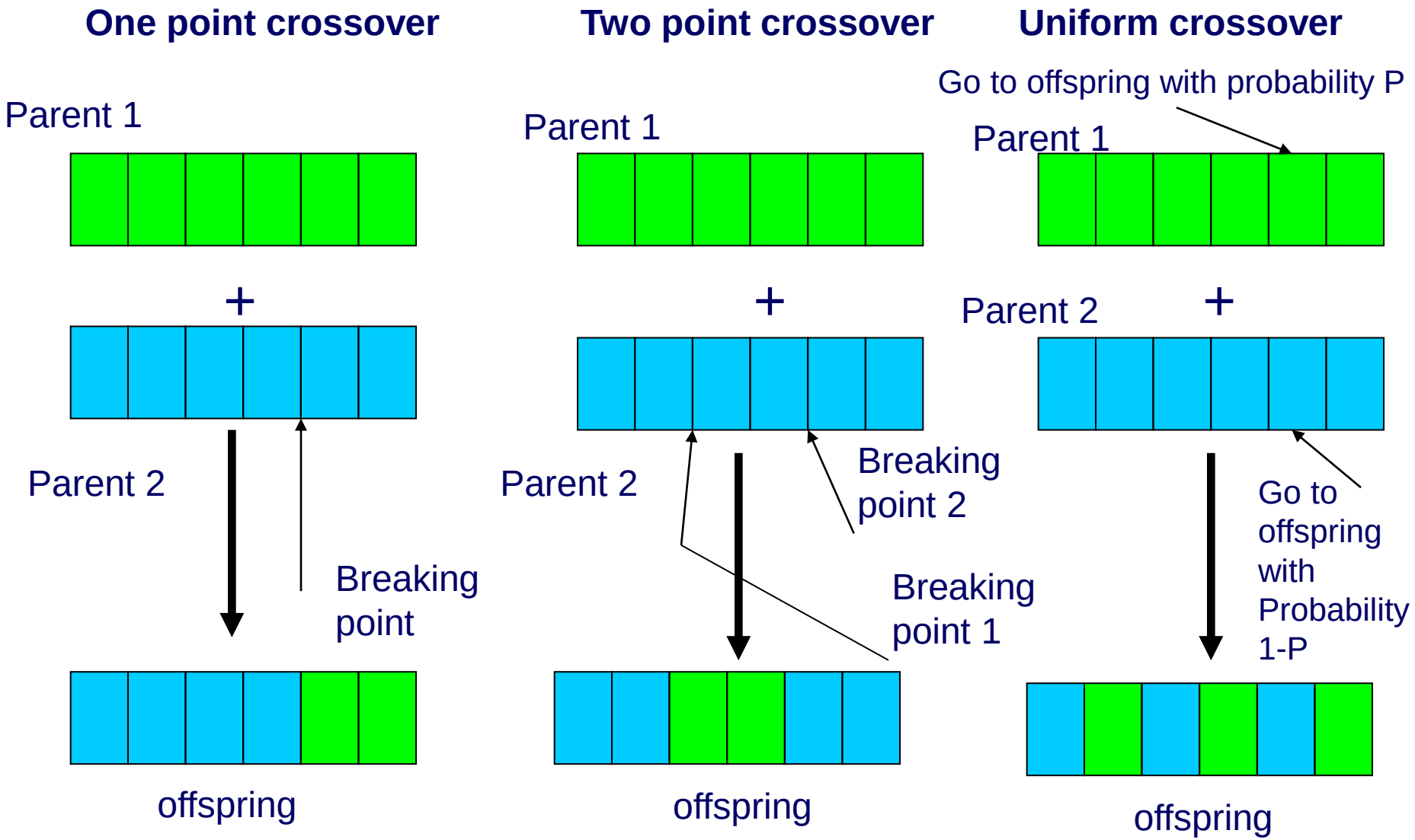
Увеличивает популяционное разнообразие и предотвращает преждевременную сходимость генетического алгоритма

После выбора очередной особи в Mating Pool вычисляется «расстояние» **D** между ее генотипом и генотипом оставшихся особей. К энергии оставшихся особей добавляется положительный **штраф ≈ 1/D**,
т.е. чем **ближе** особи друг к другу,
тем **больше штраф**



$$D = \sqrt{\sum_{i=1}^{N} \left(b_i - a_i \right)^2}$$

CROSSOVER


One point crossover
Two point crossover
Uniform crossover
Parent 1
Parent 2
Breaking point
offspring
Parent 1
Parent 2
Breaking point 2
Breaking point 1
offspring
Go to offspring with probability P
Parent 1
Parent 2
Go to offspring with Probability 1-P
offspring

Выбор параметров докинга

- **Генетический алгоритм докинга – Вероятностный процесс**
  - **Число попыток поиска минимума: 50**
  - **Размер популяции: 30000 – число начальных позиций**
  - **Размер Mating Pool: 70**
  - **Число поколений: 500-1500**

- **Объем памяти под все переменные: 512 MB**

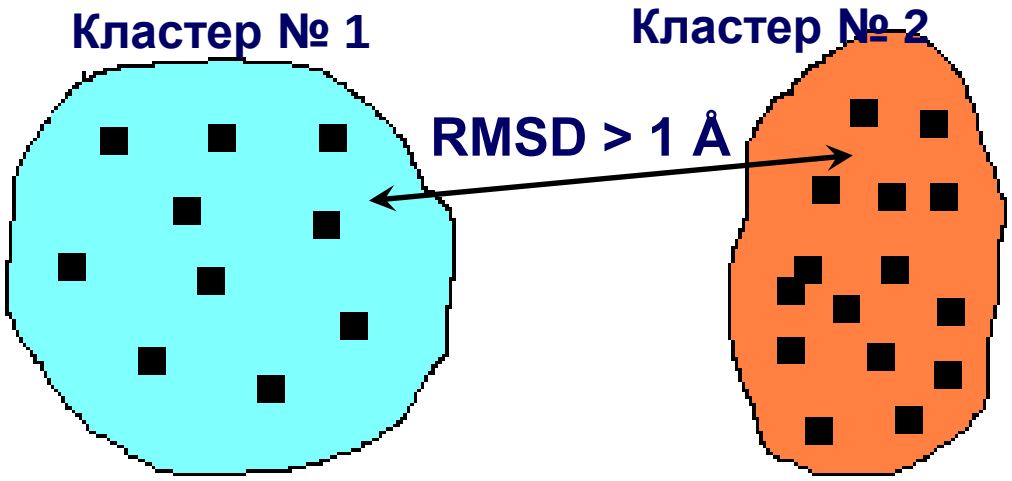
Решения задачи докинга

- Приготовление сетки потенциалов, имитирующих белок-мишень – 3-5 часов – однократно.
- Докинг: на каждый лиганд надо 1-10 часов
- Поиск наиболее энергетически выгодных поз лиганда: энергия лиганда в поле белка + энергия внутренних напряжений лиганда
- Решения задачи докинга: позы лиганда в белке
- 50 независимых запусков Генетического Алгоритма дают 50 решений - 50 лучших поз лиганда: энергия наиболее отрицательная
- Совпадение решений - кластеризация

Кластеризация решений : 50 решений после 50 независимых запусков генетического алгоритма

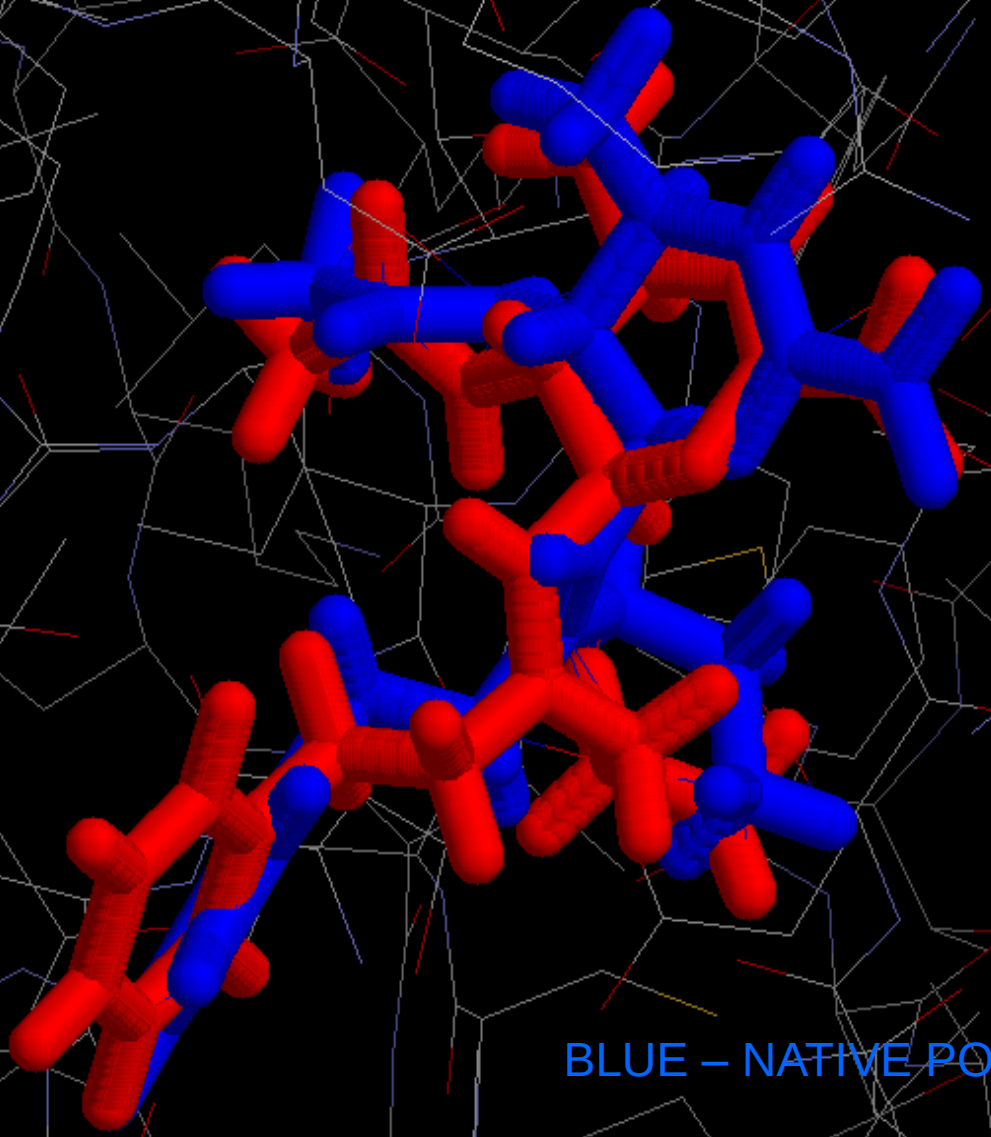
- Вычисление средне-квадратичного отклонения (RMSD) решений – поз лиганда, друг от друга
- При вычислении RMSD берутся разности декартовых координат одних и тех же атомов лиганда в двух позах
- Все решения разбиваются на группы (кластеры): внутри одного кластера RMSD < 1 Å между любыми двумя позами
- Кластеры нумеруются по возрастанию энергии лучшей позы лиганда в кластере: кластер №1 содержит позу с самой низкой энергией

Если в кластер №1 попадает большой процент из 50 решений, то задача глобальной оптимизации – **задача докинга**, решена с высокой надежностью



DOCKING RESULT vs. PDB POSITION (SIALIDASE B, PDB:1A4Q

RMSD:1.6Å



BLUE – NATIVE POSITION FROM PDB

RED – DOCKED POSITION

ДОКИНГ

ПРИМЕНЕНИЕ МОЛЕКУЛЯРНОГО МОДЕЛИРОВАНИЯ ДЛЯ РАЗРАБОТКИ НОВЫХ ЛЕКАРСТВ

В книге рассматриваются методы молекулярного моделирования, применяемые при рациональной разработке лекарств. Центральное место занимает докинг – метод молекулярного моделирования, с помощью которого молекулы кандидаты в лекарства позиционируются в активных центрах белков-мишеней, и оценивается энергия связывания белок-лиганд. Проблема докинга сводится к задаче поиска глобального минимума на сложной многомерной энергетической поверхности. Рассмотрены программы докинга на базе алгоритмов: генетическом и случайном поиске методом Монте Карло, а также на новом алгоритме суперкомпьютерного докинга на основе разложения многомерных массивов в тензорные поезда, с помощью которого успешно проведен докинг гибкого лиганда в белки-мишени с подвижными атомами на энергетической поверхности с более 150 измерениями. Рассматриваются некоторые популярные силовые поля, недавно разработанные методы квантовой химии и модели растворителя.

...Surely every medicine is an innovation; and he that will not apply new remedies, must expect new evils...

...Несомненно каждое лекарство есть инновация; и кто не будет применять новые средства, должен ждать новых бед...

Sir Francis Bacon (1561-1626)
Of Innovations



ISBN 978-5-98956-025-7

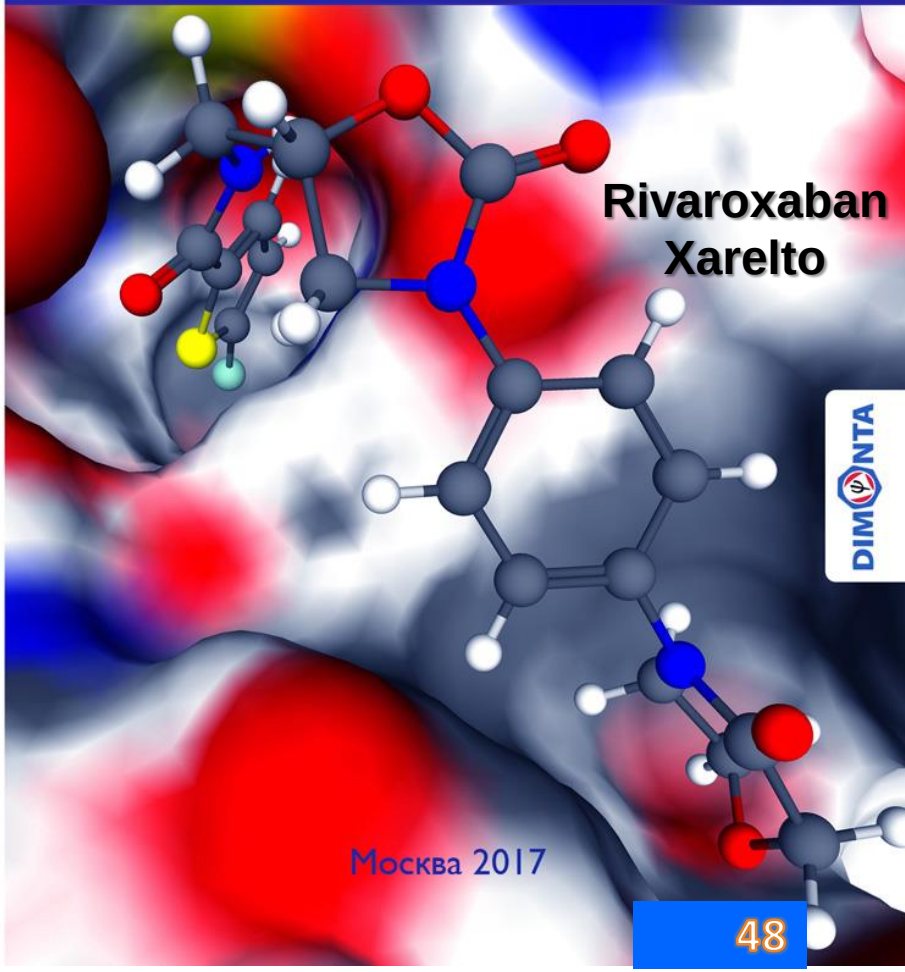
В.Б. Сулимов | А.В. Сулимов
ДОКИНГ

ИИНТЕЛЛ

В.Б. Сулимов, А.В. Сулимов

ДОКИНГ

ПРИМЕНЕНИЕ МОЛЕКУЛЯРНОГО МОДЕЛИРОВАНИЯ ДЛЯ РАЗРАБОТКИ НОВЫХ ЛЕКАРСТВ



Москва 2017

Спасибо за внимание



- *… Surely every medicine is an innovation; and he that will not apply new remedies, must expect new evils …*
- *… Каждый медицинский метод есть инновация; а кто не хочет применять новые средства, должен ждать новых бед …*

Sir Francis Bacon (1561-1626)

OF INNOVATIONS

СУПЕРКОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ И ТЕХНОЛОГИИ



Применение суперкомпьютерного моделирования для медицины

Сулимов Владимир Борисович

Научно-Исследовательский Вычислительный Центр МГУ

Лекция № 2

Докинг

- Первые программы докинга появились в конце 1980х: Силовых полей тогда не было, не было и многопроцессорных компьютеров. Простейшие модели докинга – жесткий лиганд в жесткий белок: ключ (лиганд) соответствует замку (белок)
- Простейшие силовые поля, докинг на настольном компьютере, докинг на ноутбуке – нужно посчитать многие тысячи молекул.
  Мантра всех программ докинга: кто быстрее!
- **Миллениум:** Созданы силовые поля, глобальная оптимизация на сложной многомерной энергетической поверхности

Докинг

- 2005-2010 - Суперкомпьютеры, **мантра: кто быстрее, кто больше,** виртуальный скрининг многих десятков тысяч молекул, сейчас – миллионы и миллиарды молекул
- Суперкомпьютеры: **кто точнее?** Многопроцессорный докинг 1 лиганда, минимум упрощений и без подгоночных параметров
- Точность определяет эффективность применения компьютерного моделирования для разработки лекарств, эффективной всего конвейера разработки лекарств.

- **Квантовый докинг – энергия с помощью квантовой механики**
- **2х ступенчатый виртуальный скрининг:**
  - **Скрининг больших баз данных молекул – классический докинг SOL**
  - **Квантовый докинг лучших молекул, отобранных SOL**

Подготовка белка для докинга

- Начальная стадия подготовки белка
- Упрощение белка
- Восстановление структуры белка – «штопка дыр»: восстановление пропущенных атомов и аминокислотных остатков
- Подготовка белка
- Расстановка атомов водорода
- Определение зарядовых состояний а/к остатков
- Определение области докинга

Расстановка атомов водорода на белке – собственная программа APLITE

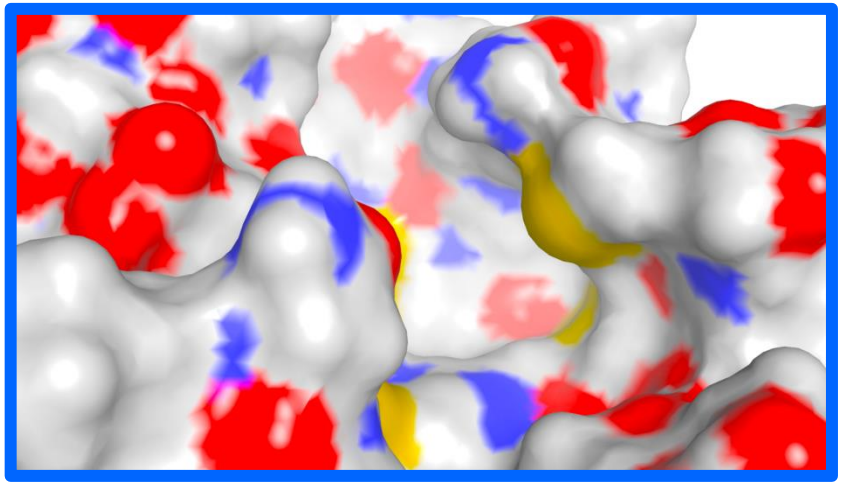
Расстановка атомов водорода на лигандах – внешние программы

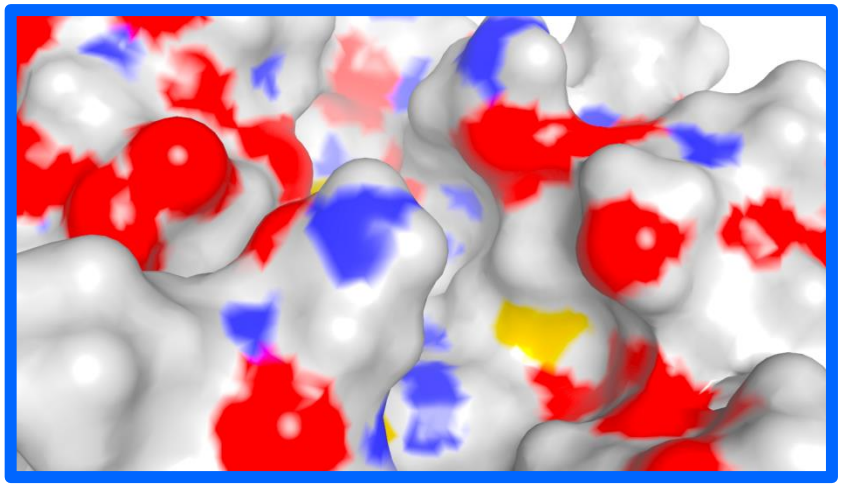
Новый белок-мишень: проверка работоспособности программы докинга

- Из Protein Data Bank берется структура комплекса белка-мишени с закристаллизованным ингибитором, называемым **нативным лигандом**
- Структура должна иметь хорошее разрешение: лучше 2.5 Å
- Лиганд должен быть низкомолекулярным
- Готовятся 3D-структуры белка-мишени и лиганда, добавляются атомы водорода
- Докинг нативного лиганда в этот белок-мишень
- **Хороший докинг: RMSD < 2** Å Плохой докинг: RMSD > 2 Å
- Если известны ингибиторы этого белка, то проводится их докинг, определяются значения скоринг-функции, и по ним определяется **порог**, разделяющий активные соединения от неактивных

Выбор структуры белка-мишени главной протеазы коронавируса

- **5 структур (< 2A) из Protein Data Bank**
- **Нативный докинг закристаллизованных соединений и кросс-докинг**
- **Выбор лучшей структуры для докинга**
  - **Позиционирование**
  - **Форма активного центра**





Скрининг существующих лекарств

- Использование для скрининга двух баз данных лекарственных соединений ZINC
  - «DrugBank FDA» only (1657 штук 2d структур)
  - «DrugBank-approved» (3004 штук 2d структур)
- 2909 уникальных 2d структур
- 3734 протонированных 2d структур (Ph=7.4)
- 7224 развернутых 3d структур – докинг с помощью SOL
- Для 210 лучших по докингу молекул методами квантовой-химии проведены расчеты энтальпии связывания
- По результатам докинга и квантовой химии отобрано 20 лучших молекул кандидатов в ингибиторы главной протеазы SARS-CoV-2 для экспериментальной проверки
- Поиск молекул-ингибиторов в других библиотеках существующих соединений

Поиск ингибиторов в базах данных существующих органических соединений

- Такие соединения не надо синтезировать, они есть «на полке»
- Многие такие базы есть в электронном виде, так что молекулы легко подготовить для докинга
- Если среди многих тысяч молекул докинг найдёт некоторое количество молекул кандидатов в ингибиторы, их можно заказать и измерить их ингибирующую активность экспериментально in vitro
- Не надо тратить на синтез новых соединений несколько месяцев.
- Найденные молекулы с экспериментально подтвержденной способностью ингибировать данный белок – «ХИТЫ», являются основой для дизайна новых молекул – более сильных и селективных ингибиторов, и их синтеза: оптимизация «хитов» приводит к лидерному соединению
- Дизайн новых молекул: создание библиотек виртуальных молекул, их докинг, отбор лучших кандидатов, обсуждение с химиками-синтетиками, снова дизайн, …, согласование молекул для их синтеза

Алгоритм разработки ингибиторов

- Докинг молекул из баз данных доступных соединений (синтезированы, их можно купить или заказать бесплатно)
- Дизайн новых молекул, докинг виртуальных молекул, выбор лучших, согласование их синтеза



Острая токсичность на мышах: летальная доза ингибитора LD50

Антивирусные лекарства

- Ингибирование белка-мишени: эксперименты in vitro в тестах «белок-субстрат-лиганд»
- Подавление размножения вируса в культуре клеток
- Цитотоксичность: при какой концентрации ингибитора клетки умирают
- Индекс селективности: во сколько раз концентрация ингибитора, когда клетки умирают, больше концентрации, когда вирусы дохнут

Программы докинга стали популярны. Десятки программ докинга востребованы при разработке лекарств. Каждая программа – уникальное сочетание моделей и приближений, скорости

Возможно ли увеличить точность докинга?

- Точность позиционирования – ***удовлетворительна***
- Точность расчетов энергии связывания белок-лиганд ΔG_{bind} – ***плохая***

Точность докинга → **Эффективность разработки лекарств**

Задача обобщенного докинга: увеличить точность

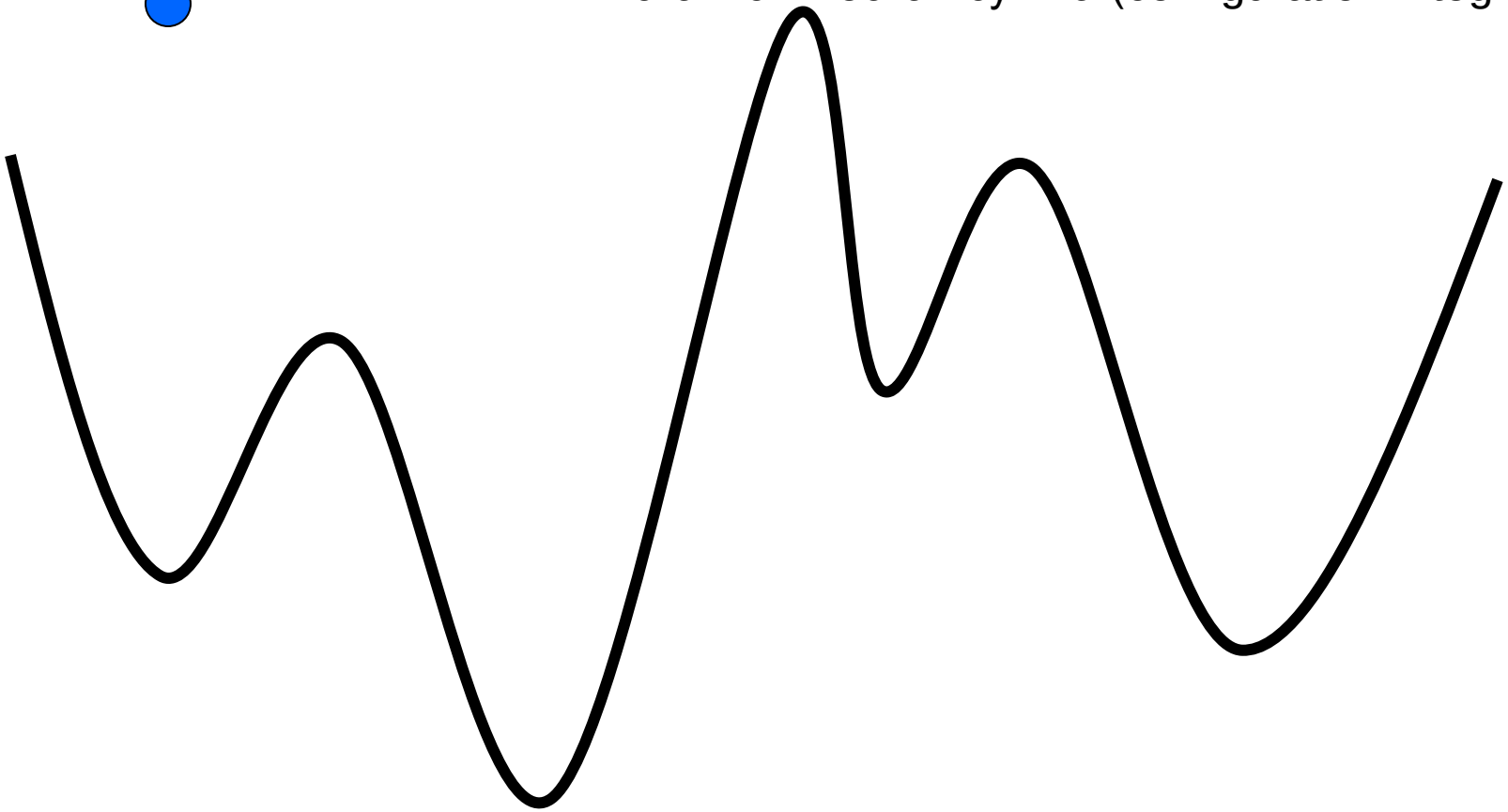
- **Гибкий лиганд + Гибкий белок**: Подвижность атомов белка и лиганда учитывается в процессе докинга одновременно и на одинаковой основе
- *Генетический алгоритм (один из самых популярных алгоритмов докинга) не справляется с глобальной оптимизации при размерности энергетической поверхности* **> 20 – нужен новый алгоритм докинга**
- Энергия комплекса белок-лиганд вычисляется в рамках силового поля **MMFF94** без упрощений
- Нет заранее вычисляемой сетки потенциалов взаимодействия атомов лиганда с белком
- Докинг осуществляется путем поиска целого спектра низкоэнергетических минимумов системы белок-лиганд
- Точности позиционирования определяет точность расчета энергии связывания белок-лиганд

Жёсткий белок → **Белок с подвижными атомами**

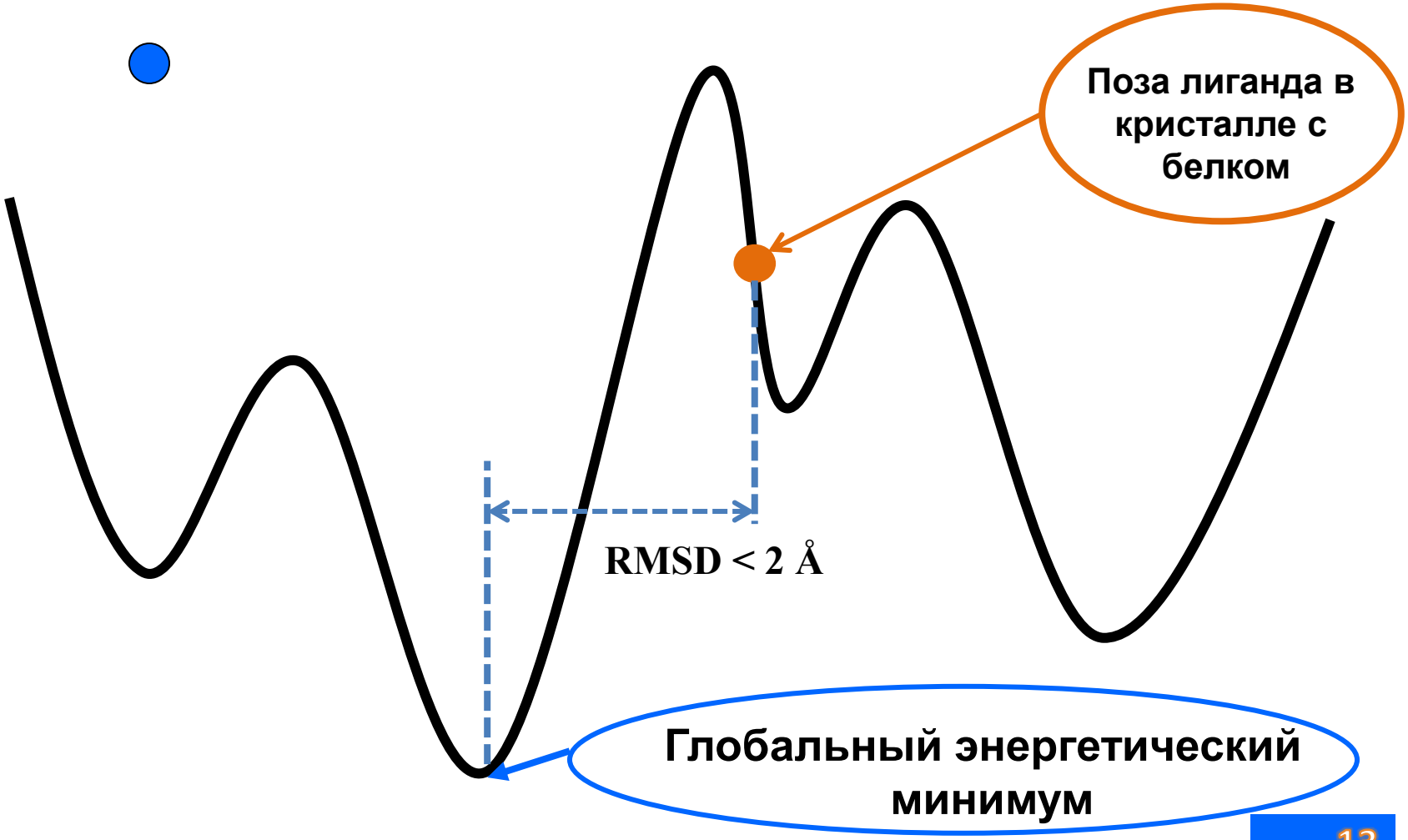
Движение лиганда в белке-мишени

$$G = -RT * Ln(Z) \qquad Z = \frac{1}{(2\pi\hbar)^{3n}} \int e^{-(U+W)/kT} \, dx_1 \ldots dx_{3n} \, dp_1 \ldots dp_{3n}$$

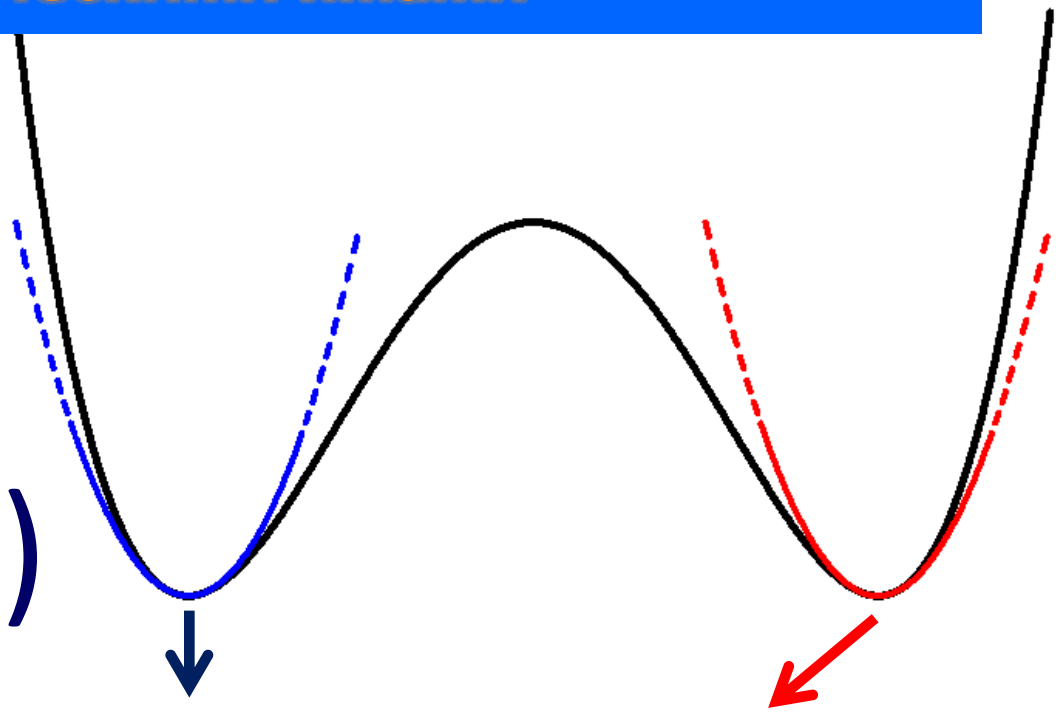
статистическая сумма (configuration integral)



На многомерной энергетической поверхности системы белок-лиганд очень много локальных минимумов



Аппроксимация U(x) независимыми гармоническими ямами



$$G = -kT * \ln(Z)$$

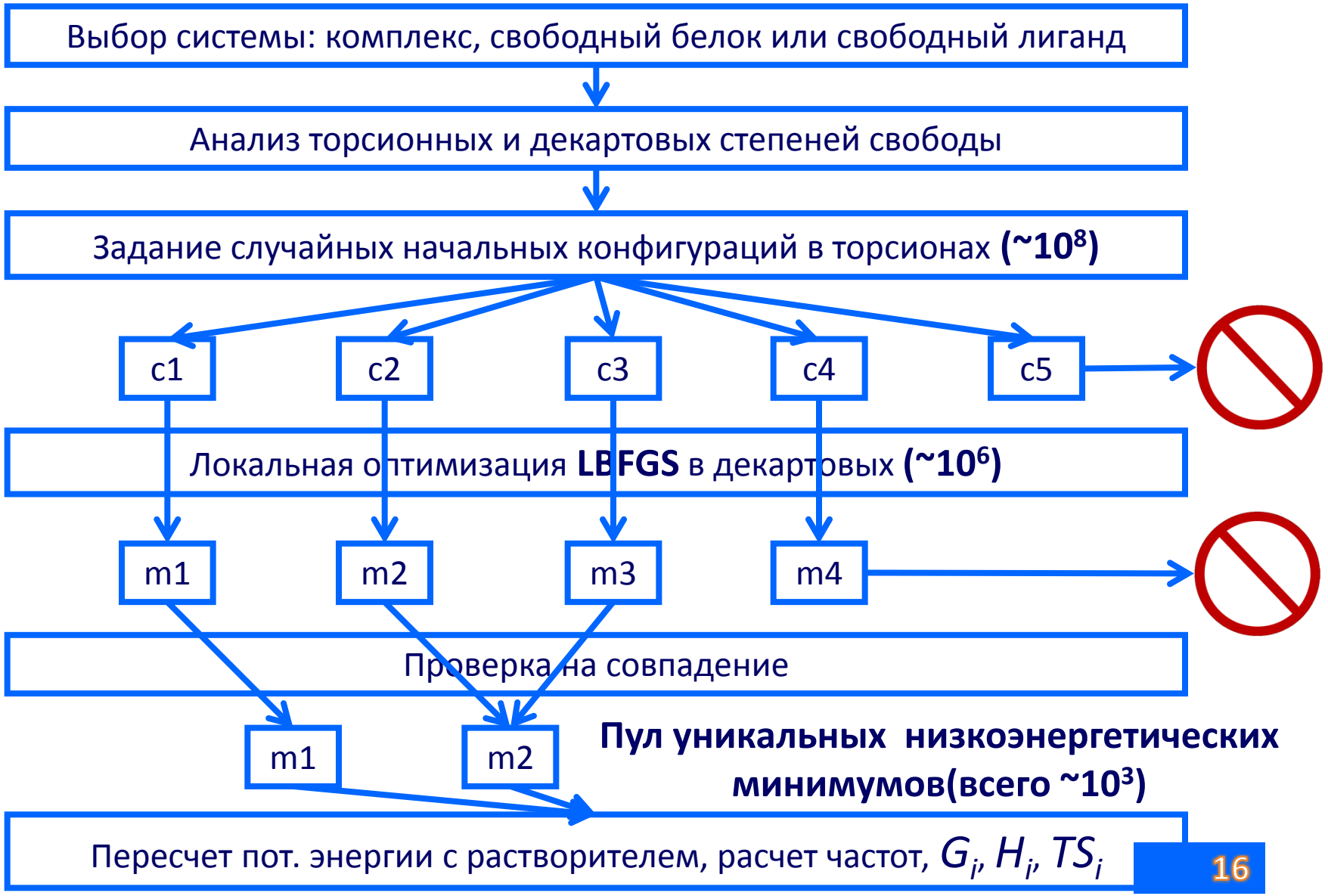
$$Z = Z_1 + Z_2$$

$$Z_i = e^{-\frac{E_0{}^i}{kT}} * \prod_j \frac{e^{-\hbar\omega_j{}^i / 2kT}}{1 - e^{-\hbar\omega_j{}^i / kT}}$$

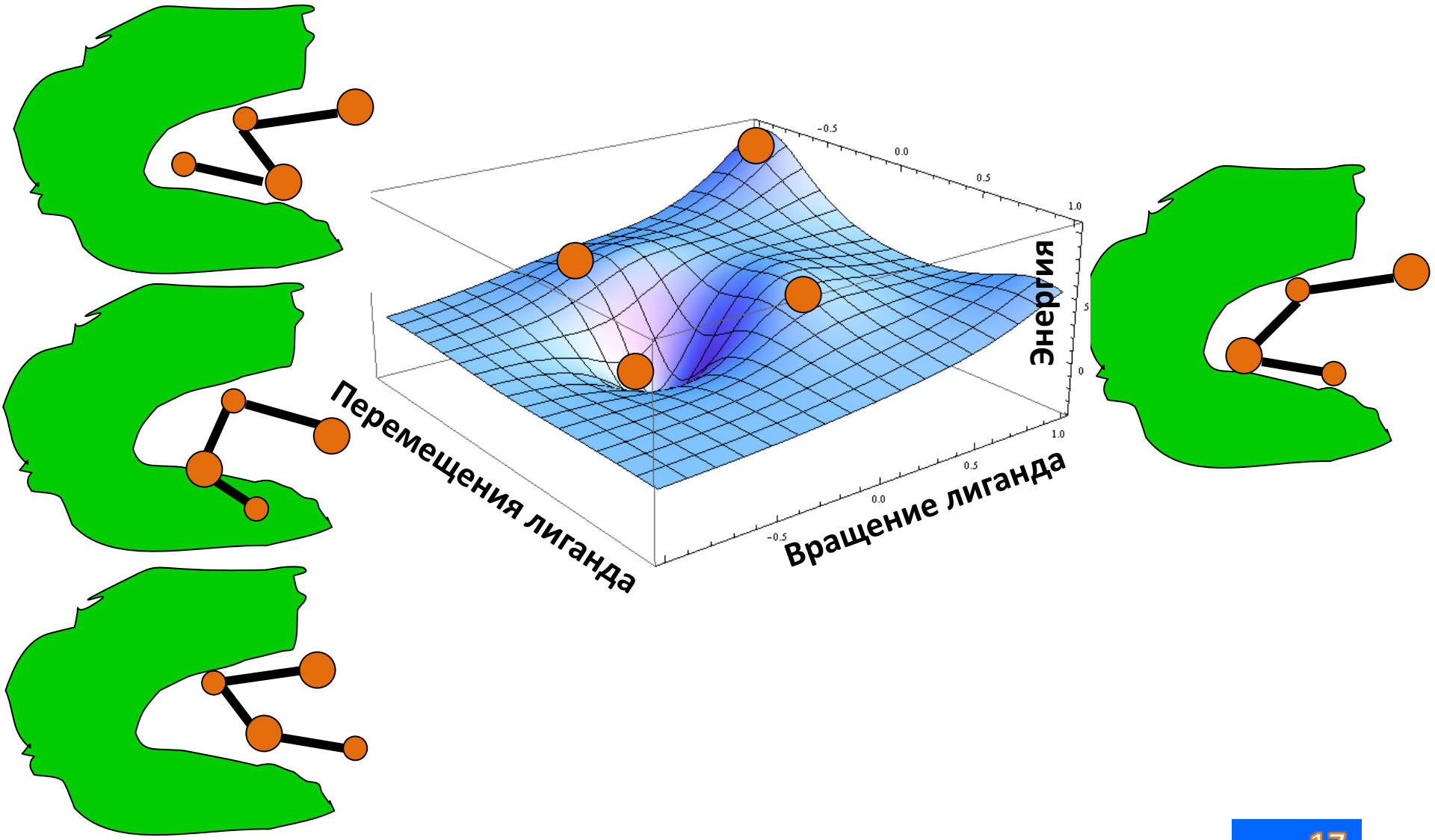
Программа FLM

- FLM не использует заранее рассчитанную сетку потенциалов
- Жесткий белок, случайное «бросание» гибкого лиганда
- Локальная оптимизация энергии системы белок-лиганд из случайной позы лиганда: варьируются положения всех атомов лиганда
- В вакууме или с учетом растворителя в континуальной модели
- Силовое поле MMFF94, без упрощений, без подгоночных коэффициентов
- Исчерпывающий поиск спектра низкоэнергетических минимумов комплекса белок-лиганд
- Многие тысячи минимумов – отбор уникальных «на лету»
- Параллельные вычисления - докинг 1 лиганда: на **8191 ядрах** занимает несколько часов Ломоносова ≈10 000 CPU·часов

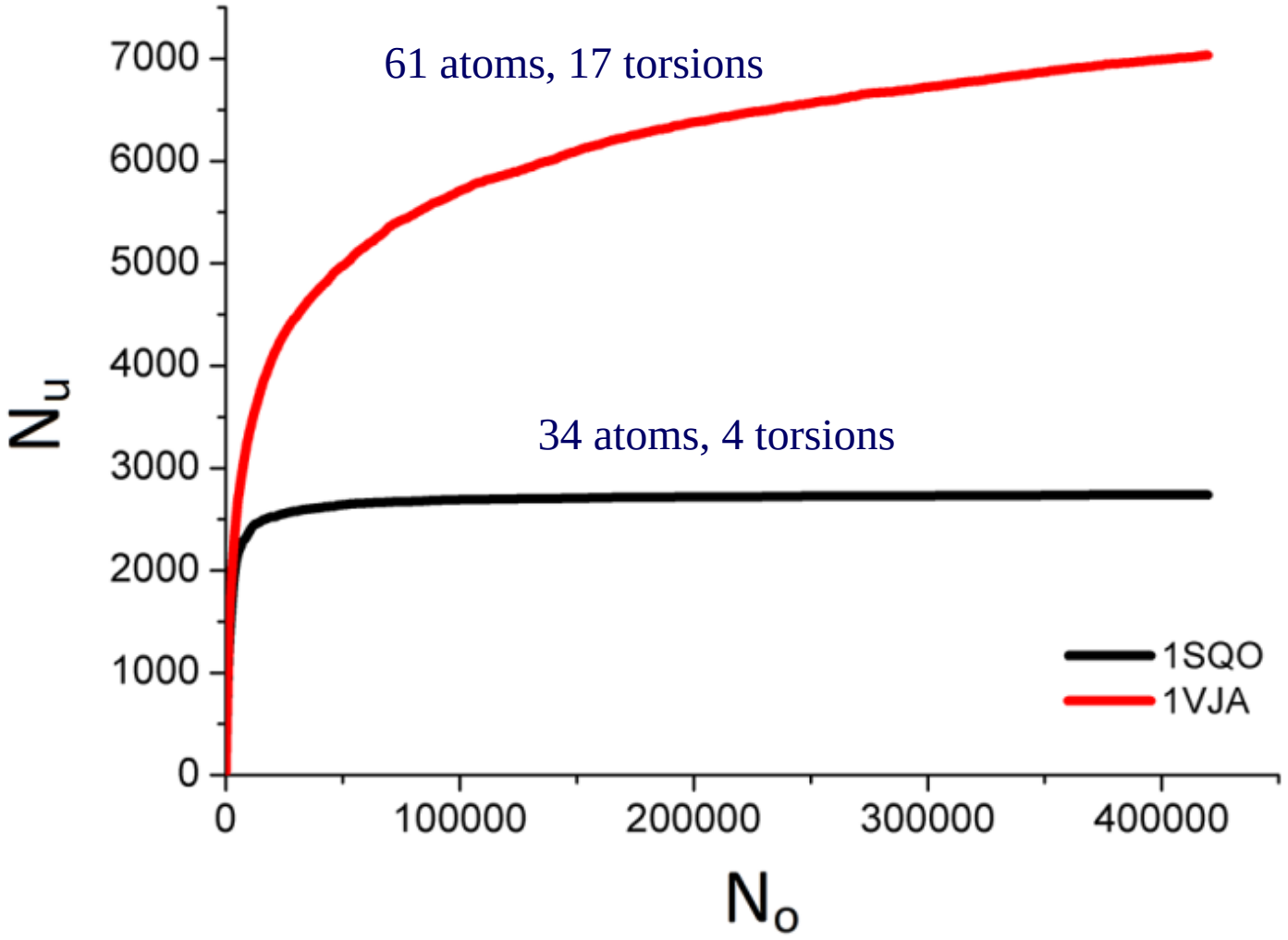
Программа FLM



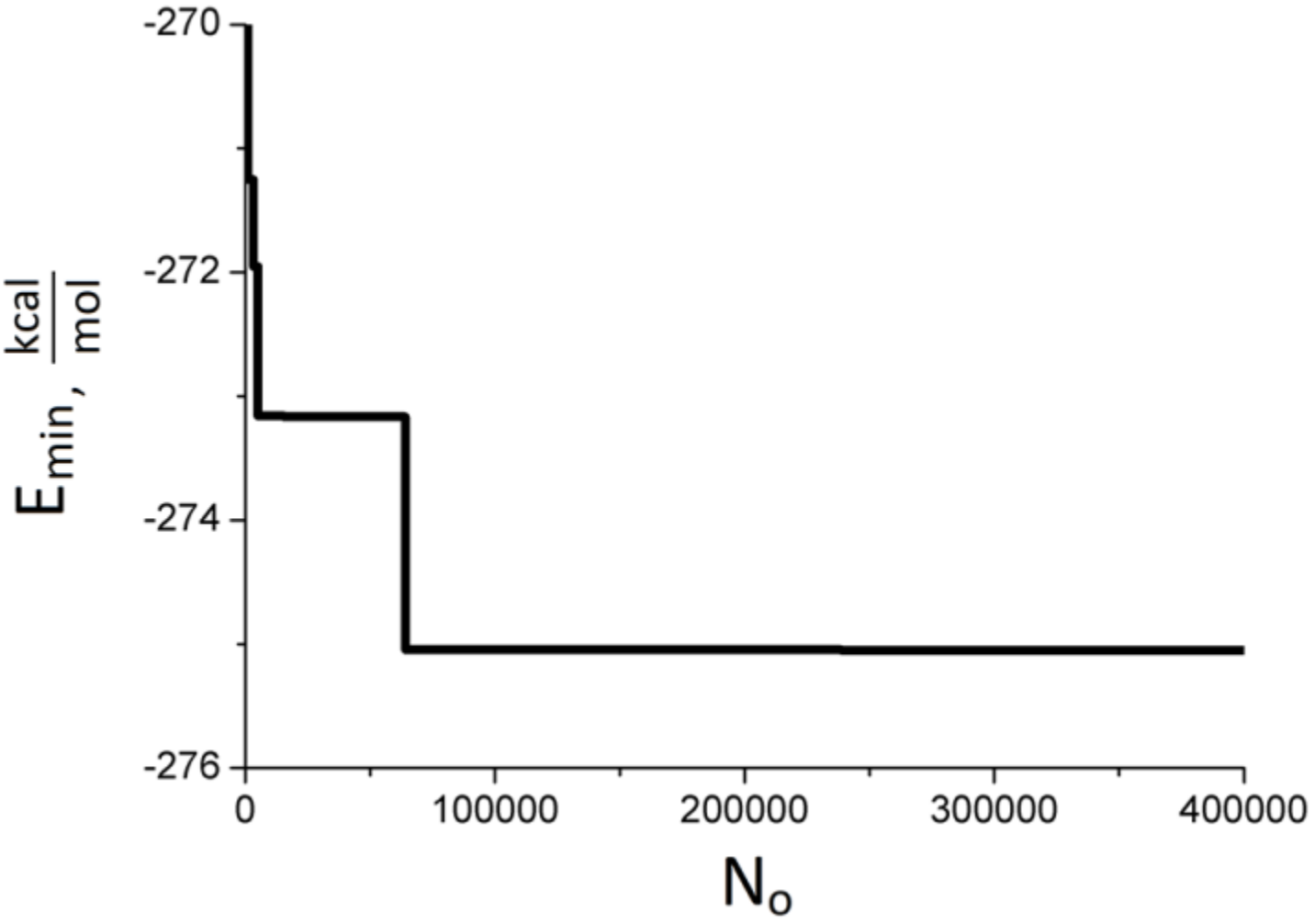
Программа FLM



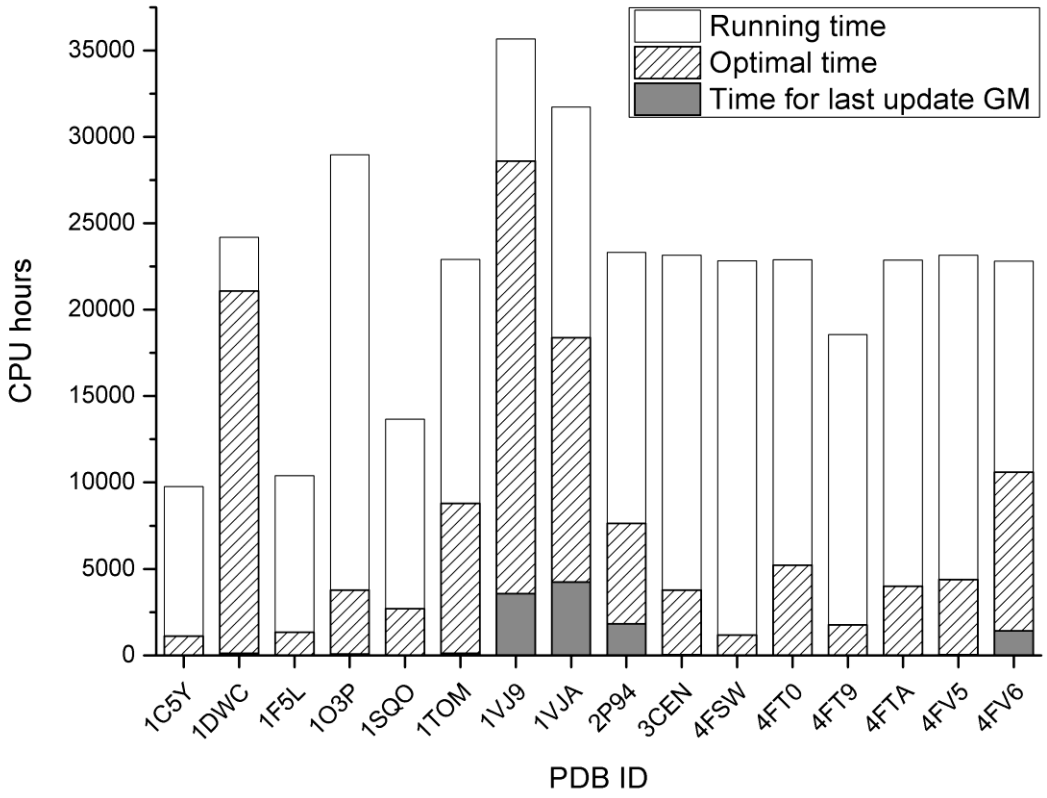
Nu – число обновлений пула уникальных низкоэнергетических минимумов
No – число оптимизаций



E_{min} – глобальный минимум энергии
N_o – число оптимизаций, комплекс 1VJA



Время счета FLM



- CPU зависит от числа атомов лиганда и от числа внутренних вращательных степеней свободы
- Глобальный минимум находится гораздо быстрее, чем весь пул низкоэнергетических минимумов

Докинг с помощью программы FLM

- FLM опирается на парадигму докинга: лиганд связывается с белком вблизи глобального минимума системы белок-лиганд
- FLM – это инструмент для тестирования применимости энергетических функций системы белок-лиганд для докинга
- FLM – это инструмент для тестирования эффективности алгоритмов глобальной оптимизации энергии системы белок-лиганд
- FLM нужно около 10 000 CPU*часов для докинга 1 лиганда
  - **FLM не может быть использована для скрининга больших баз данных – докинг только отдельных лигандов**
  - **FLM не может быть использована для докинга с учетом подвижности атомов белка**
  - **FLM для проведения квантово-химического докинга - квазидокинг**

Квазидокинг – путь к квантово-химическому докингу

Рельеф многомерной поверхности энергии белок-лиганд слабо зависит от выбора силового поля или метода квантовой химии – минимумы находятся примерно там же

- Используя программу **FLM** в вакууме, находим несколько тысяч низкоэнергетических минимумов комплекса белок-лиганд в силовом поле MMFF94 – обычно 4 тысяч минимумов хватает – это показали исследования десятков комплексов белок-лиганд
- Энергия каждого найденного минимума вычисляется с помощью полуэмпирического квантово-химического метода **PM7** без дополнительной локальной оптимизации
- Выявляется поза с самой низкой энергией PM7. Парадигма докинга выполняется почти для всех (нескольких десятков) тестовых комплексов
- Проводится локальная оптимизация энергии **PM7** из позы лиганда с наинизшей энергией при варьировании положений атомов лиганда. Это и есть лучшая поза лиганда

Что такое квантовая химия?

P.A.M.Dirac **Erwin Schrödinger** **Werner Heisenberg** **В.А. Фок**






Dirac's famous statement

The underlying physical laws necessary for the mathematical theory of a large part of physics and the whole of chemistry are thus completely known, and the difficulty is only that exact application of these laws leads to equations much too complicated to be soluble. **1929 г.**

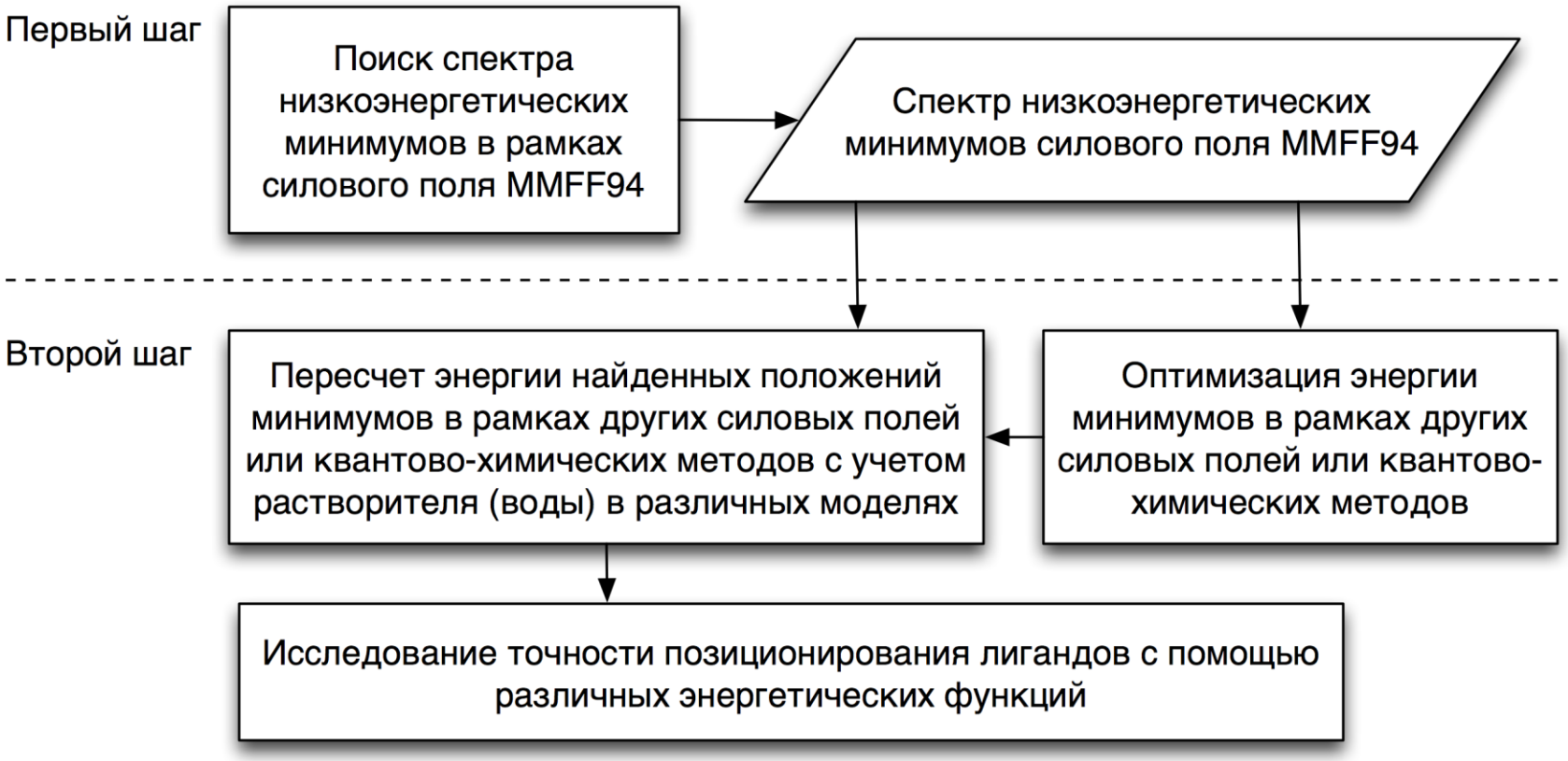
Взаимодействия между атомами надо рассчитывать с помощью методов квантовой механики

- Методы квантовой механики, применяемые к молекулам, называются методами квантовой химии
- Самосогласованное решение для описания распределения электронов в молекуле, многоэлектронная волновая функция, электронная плотность, тщательный квантовых эффектов, учет электронных корреляций, богатые базисы
- Тяжелые ядра двигаются медленно, электроны быстро: адиабатическое приближение – решается уравнение Шредингера для каждого положения электронных ядер
- Методы квантовой химии достигли высокого уровня точности за последние 50 лет, особенно, когда появились доступные суперкомпьютеры
- Требуют больших вычислительных ресурсов
  - Реально рассчитать систему МАКСИМУМ из нескольких десятков (сотен) атомов
  - Многопроцессорные, многочасовые расчеты
- Для решение уравнения Шредингера используются базисные функции и в конечном счете всё сводится к самосогласованному решению матричных уравнений

Квантово-химический пакет MOPAC

- Полуэмпирические методы квантовой химии: на основе физических соображений некоторые интегралы в матричном уравнении Шредингера заменяются нулями, некоторые вычисляются, другие заменяются аналитическими формулами; эти методы развиваются около 50 лет
- Недостатки этих упрощений компенсируются введением параметров, которые подгоняются, либо под эксперимент, либо под точные расчета ab initio
- Новый квантово-химический полуэмпирический метод PM7:
  - Улучшенные описания межмолекулярных взаимодействий
- Можно проводить расчеты комплексов белок-лиганд – несколько тысяч атомов
- Для каждого расчета комплекса белок-лиганд необходимо:
  - При оптимизации лиганда около **10 часов на 1 вычислительном ядре**
  - Без оптимизации лиганда (1SCF) **несколько минут**

Процедура квазидокинга

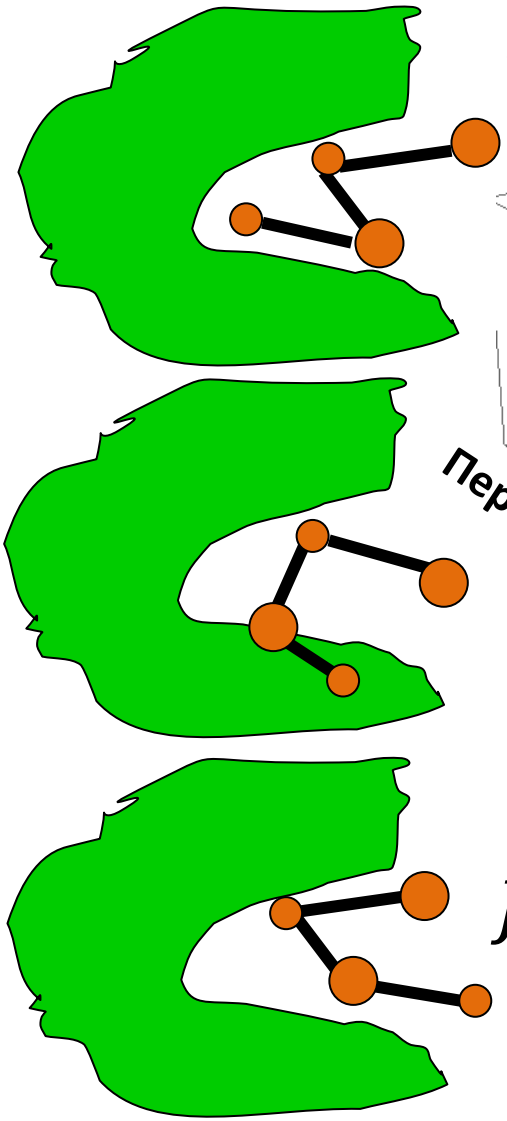


A.V.Sulimov et al. Journal of Molecular Graphics and Modelling, 2017, 78:139-147

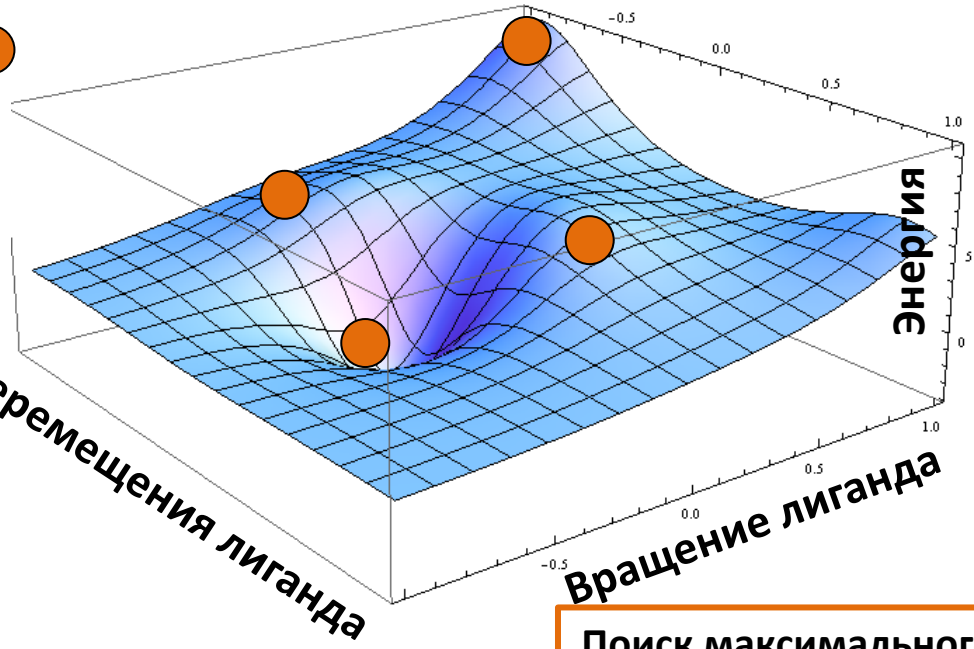
Суперкомпьютерная программа докинга SOL-P: ТТ-алгоритм докинга основан на методе глобальной оптимизации с использованием разложение тензоров в тензорные поезда – Tensor Trains

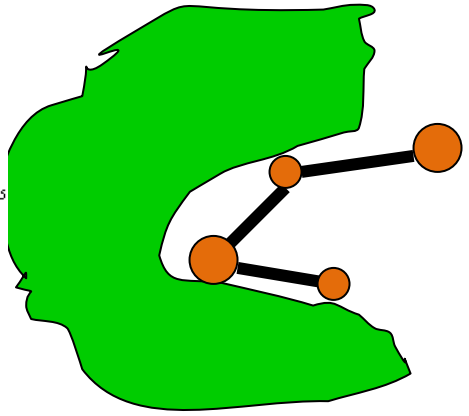
- Классическое силовое поле MMFF94, без упрощений, без подгоночных параметров
- Нет сетки потенциалов взаимодействия белок-лиганд
- Многопроцессорные параллельные вычисления: несколько сотен вычислительных ядер
- Непрерывная функция энергии системы белок-лиганд преобразуется с помощью очень мелкой сетки в многомерный тензор большого размера
- К этому тензору применяются современные методы тензорного анализа, развитые в Институте вычислительной математики РАН, для поиска максимального по модулю элемента тензора – он и будет соответствовать глобальному оптимуму исходной непрерывной функции

Программа докинга SOL-P: Преобразование непрерывной функции энергии в многомерный тензор Число измерений = Числу степеней свободы системы



Сетка дискретизации энергии: $n = 2^{16}$ – по каждому измерению





Поиск максимального по модулю элемента многомерного тензора

$$f(x_1, x_2, \ldots, x_d) \Rightarrow A(i_1, i_2, \ldots i_d)$$

Поиск глобального минимума непрерывной функции

i=1, 2, 3,…2^{16}

Разложение тензора в тензорный поезд

Многомерный массив (тензор) $A \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ может быть представлен в форме:

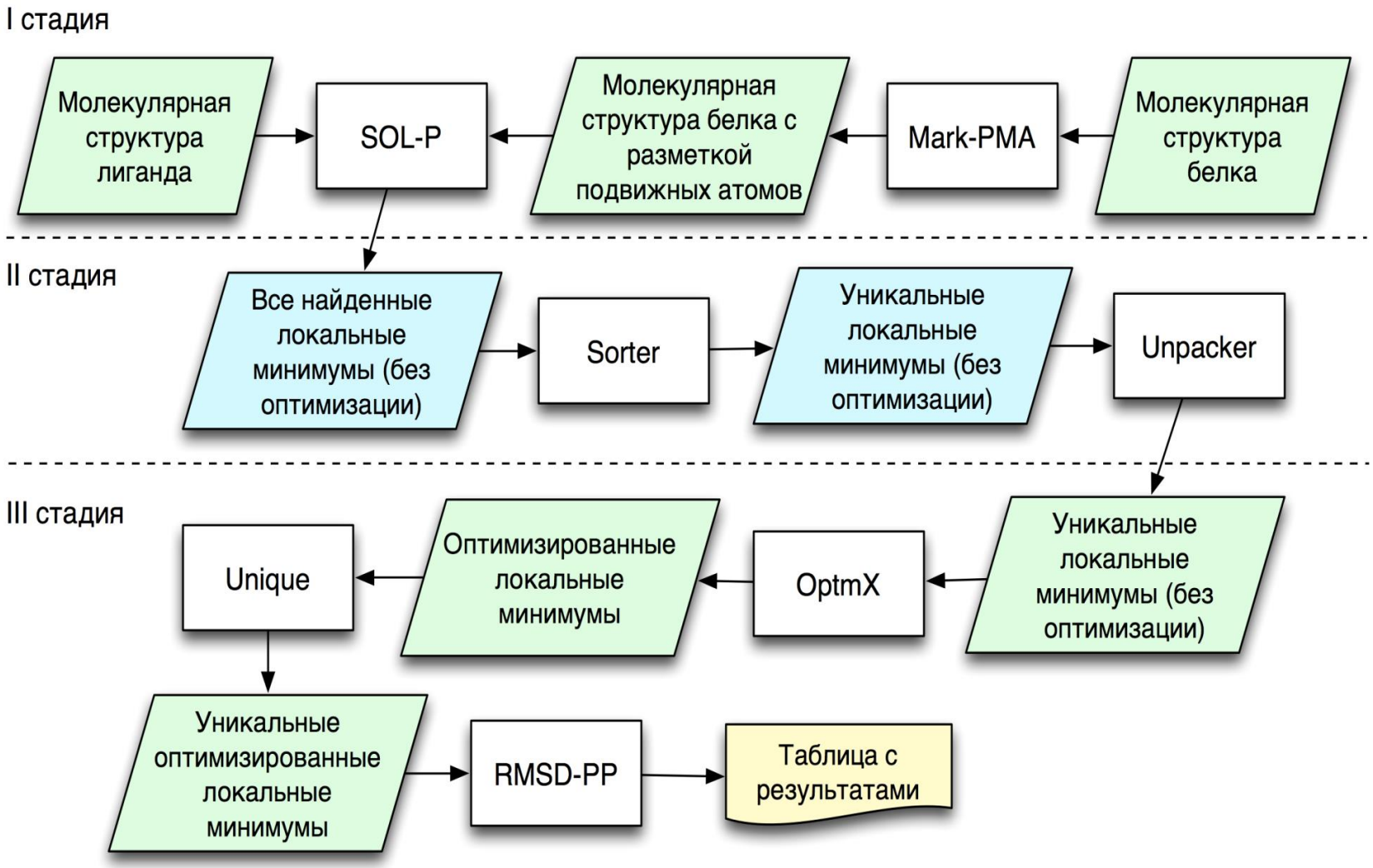
$$A(i_1, \ldots, i_d) =$$

$$= \sum_{\alpha_1=1,\ldots,\alpha_{d-1}=1}^{r_1,\ldots,r_d} G_1(i_1, \alpha_1) G_2(\alpha_1, i_2, \alpha_2) \ldots G_{d-1}(\alpha_{d-2}, i_{d-1}, \alpha_{d-1}) G_d(\alpha_{d-1}, i_d)$$

- $r_1, \ldots, r_{d-1}$ - TT-ранги данного тензора
- 3х мерные тензоры $G_i \in \mathbb{R}^{r_{i-1} \times n_i \times r_i}$ - вагоны

Если тензор получен при дискретизации регулярной гладкой функции – например, энергии комплекса белок-лиганд, то **TT ранги – небольшие числа**

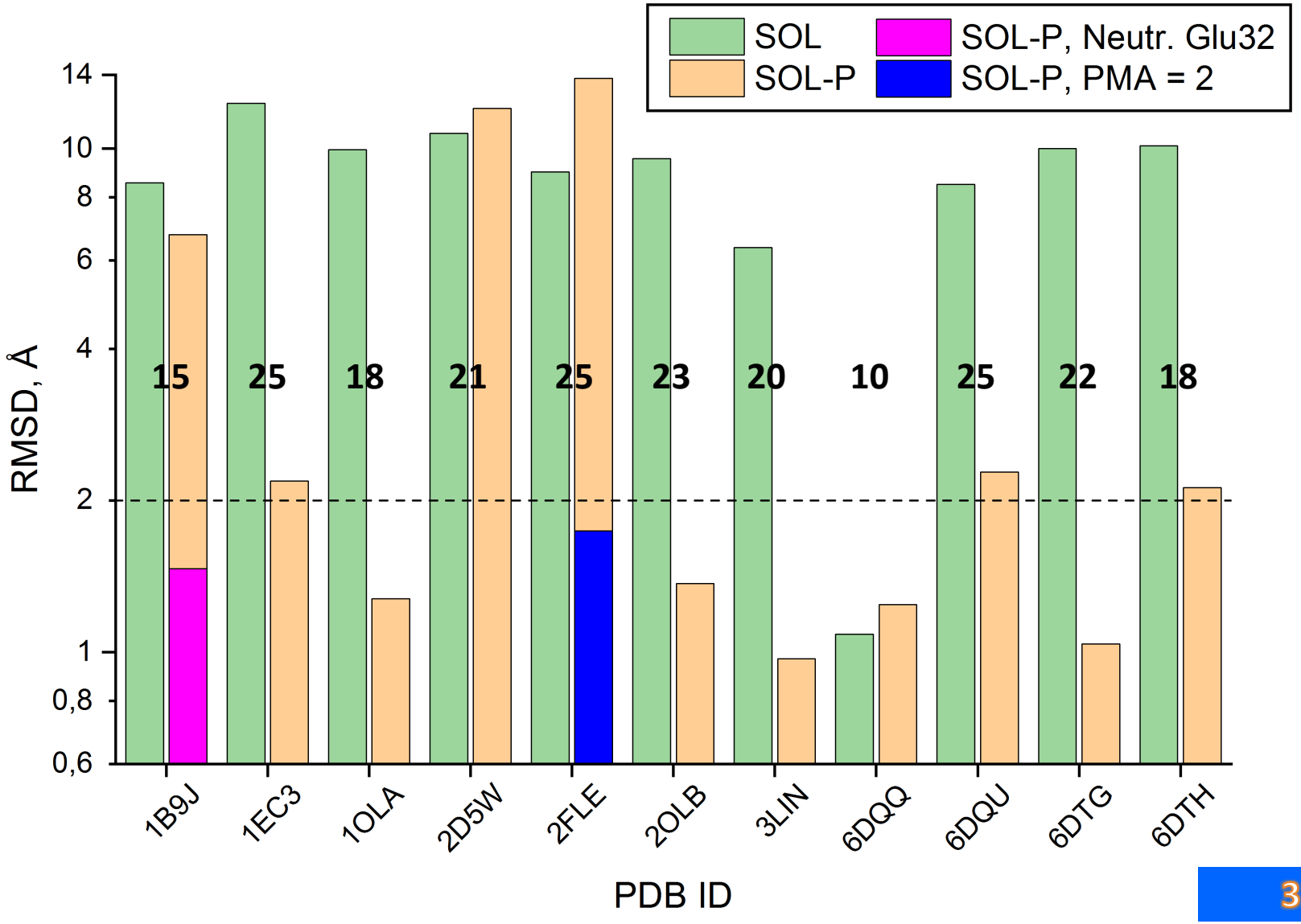
Блок-схема программы SOL-P



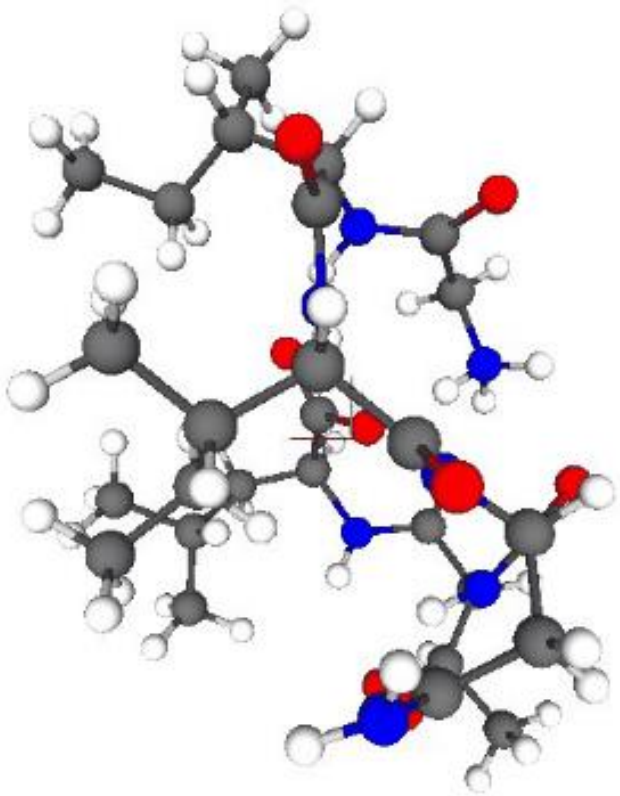
SOL-P суперкомпьютерная программа докинга Tensor Глобальная оптимизация энергии с помощью тензорных поездов Tensor Trains

- Почти для всех тестовых комплексов SOL-P находит тот же глобальный минимум энергии, который находит FLM
- Для жесткого белка SOL-P работает гораздо быстрее, чемFLM:
  - SOL-P нужно 100 CPU*часов
  - FLM нужно 10 000 CPU*часов
- SOL-P успешно проводит докинг лигандов, имеющих до 25 торсионов; для сравнения
  - Для SOL максимальное число торсионов 10-15,
  - Для Glide, GOLD and ICM max торсионов 10
- SOL-P может проводить докинг гибких лигандов в белки с несколькими десятками подвижных атомов: до 157 степеней свободы белка

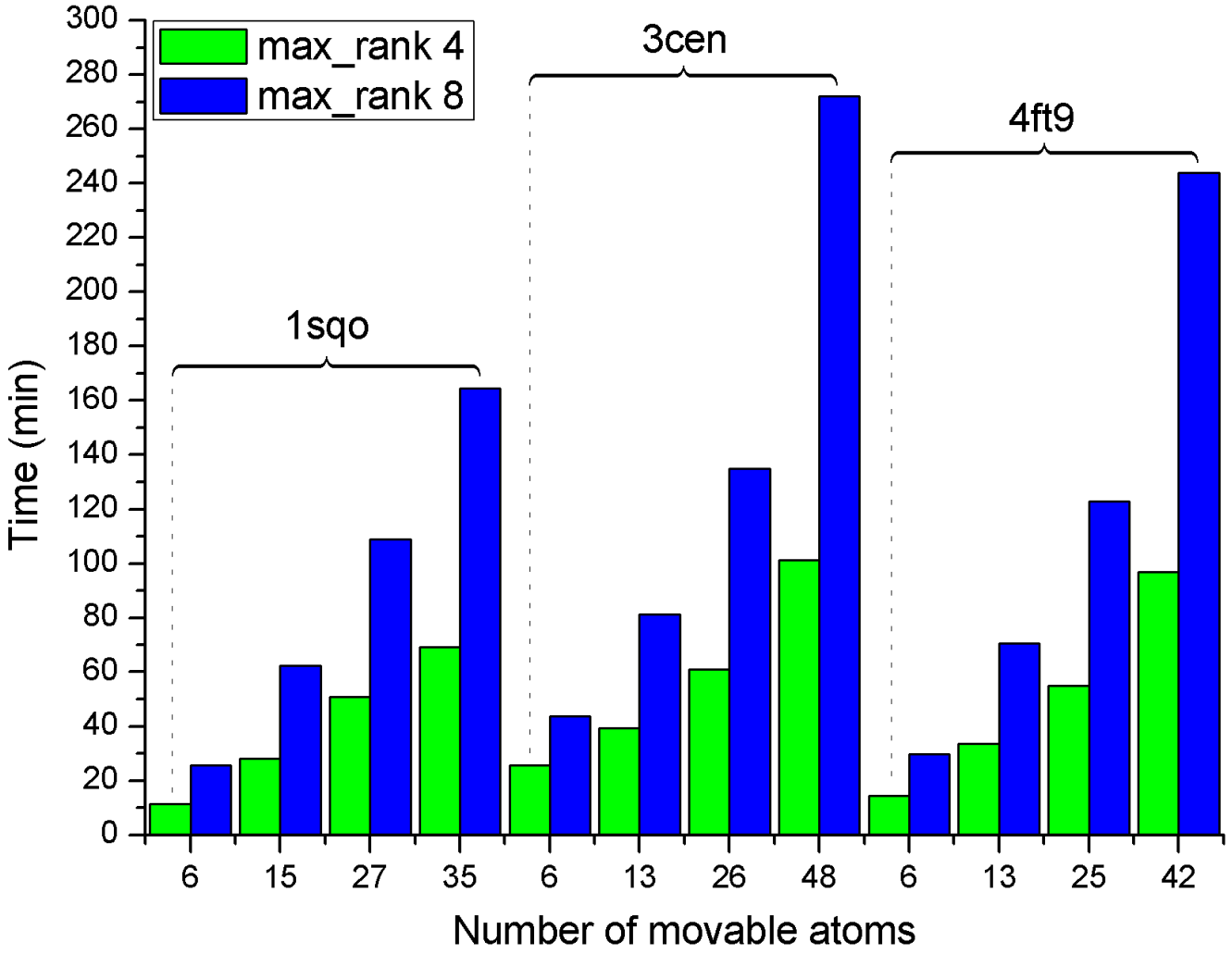
Докинг гибких лигандов



Гибкий олигопептид Gly-Ile-Ile-Asn-Thr-Leu
25 внутренних вращательных степеней свободы



SOL-P: времена докинга 1 лиганда на 256 cores



Программы докинга

- **AutoDock** – The Scripps Research Institute – **USA**
- **ICM** – Molsoft, LLC (Abagyan) San Diego, CA – **USA**
- **DOCK6** – University of California, San Francisco – **USA**
- **GOLD** - The Cambridge Crystallographic Data Centre – **UK**
- FlexX – BioSolveIT – **Germany**
- Sybyl – Tripos – **USA**
- Discovery Studio – Accelrys – **USA**
- Glide – Schrödinger – **USA**
- Lead-Finder – Moltech, Ltd – **Россия**
- **SOL – НИВЦ МГУ** – **Россия**
- **FLM – НИВЦ МГУ** – **Россия**
- **SOL-P – НИВЦ МГУ** – **Россия**

Программы докинга против COVID-19

- В настоящее время известно около 50 программ докинга и с десяток сайтов для выполнения докинга online
- Каждая программа уникальна совокупностью используемых моделей, аппроксимаций и подгоночных параметров
- Большинство программ рассчитаны на работу на ноутбуке (1 CPU) и быстрый докинг – меньше минуты на 1 лиганд
- Пандемия высветила чрезвычайную востребованность докинга, особенно докинга больших баз данных, имеющих сотни тысяч и даже миллионы молекул
- Цель – быстрая разработка противовирусных препаратов
- Несколько белков коронавируса SARS-CoV-2 считаются терапевтическими мишенями, и их атомистическая пространственная структура измерена – можно проводить **Computer Aided Structural Based Drug Design**

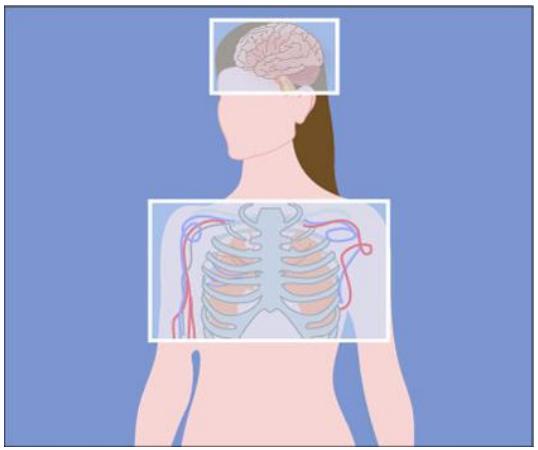
Экспертные системы персонифицированной медицины



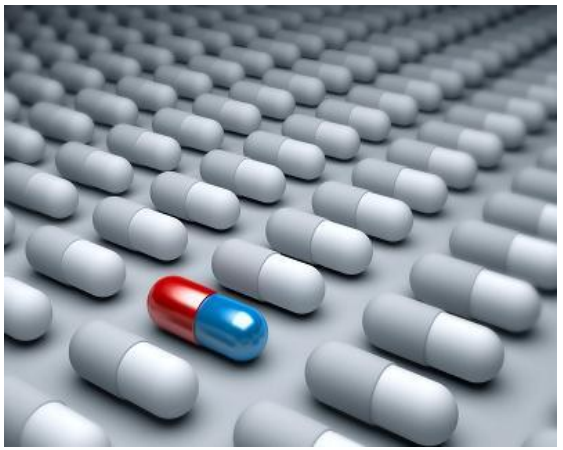
Цель – предсказание исхода для нового пациента

Задача – построение байесовской сети с наилучшим качеством предсказания

Методика – вероятностная модель на основе байесовских сетей

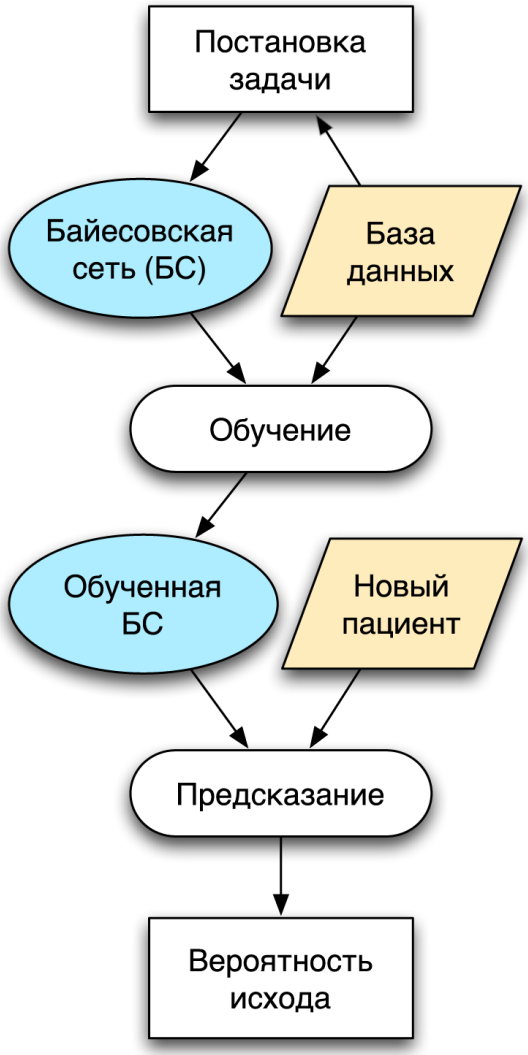






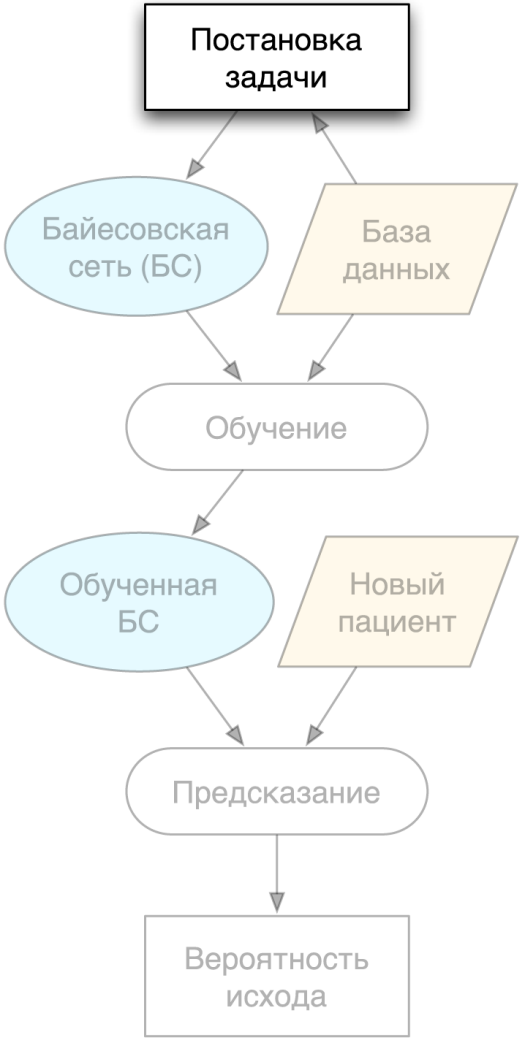
Модель экспертной системы

- Целью экспертной системы является предсказание исхода для нового пациента на основе известных событий
- Данные пациентов предоставляются в базе данных содержащей как клинические, так и генетические параметры
- Предсказание осуществляется на основе вероятностной модели, опирающейся на методику байесовских сетей



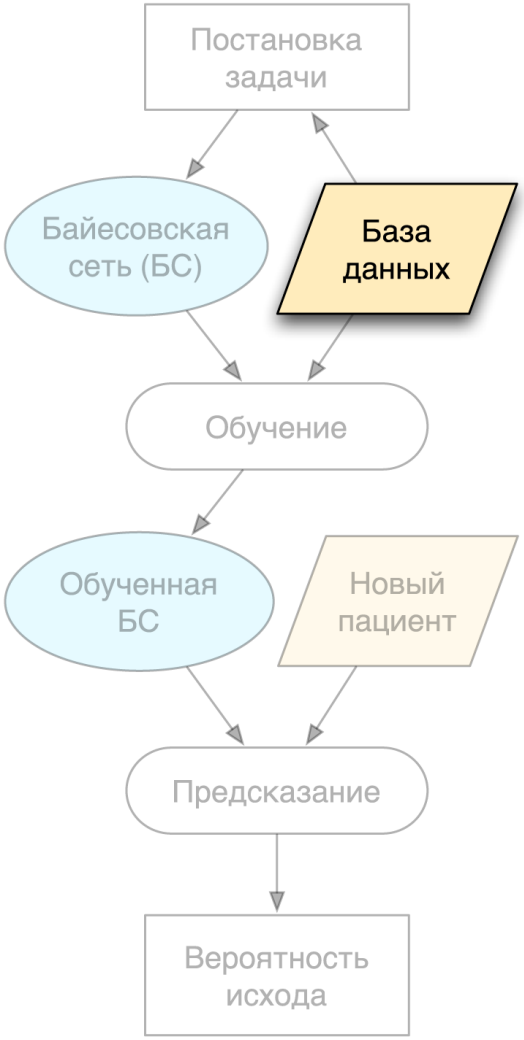
Постановка задачи

- Необходимо сформулировать исход - что мы предсказываем и на какой момент времени?
- Для всех прошедших событий на указанный момент времени необходимо определить точный исход
- Параметры на основе которых осуществляется предсказание должны быть идентичные (как для прошедших событий, так и для новых пациентов)

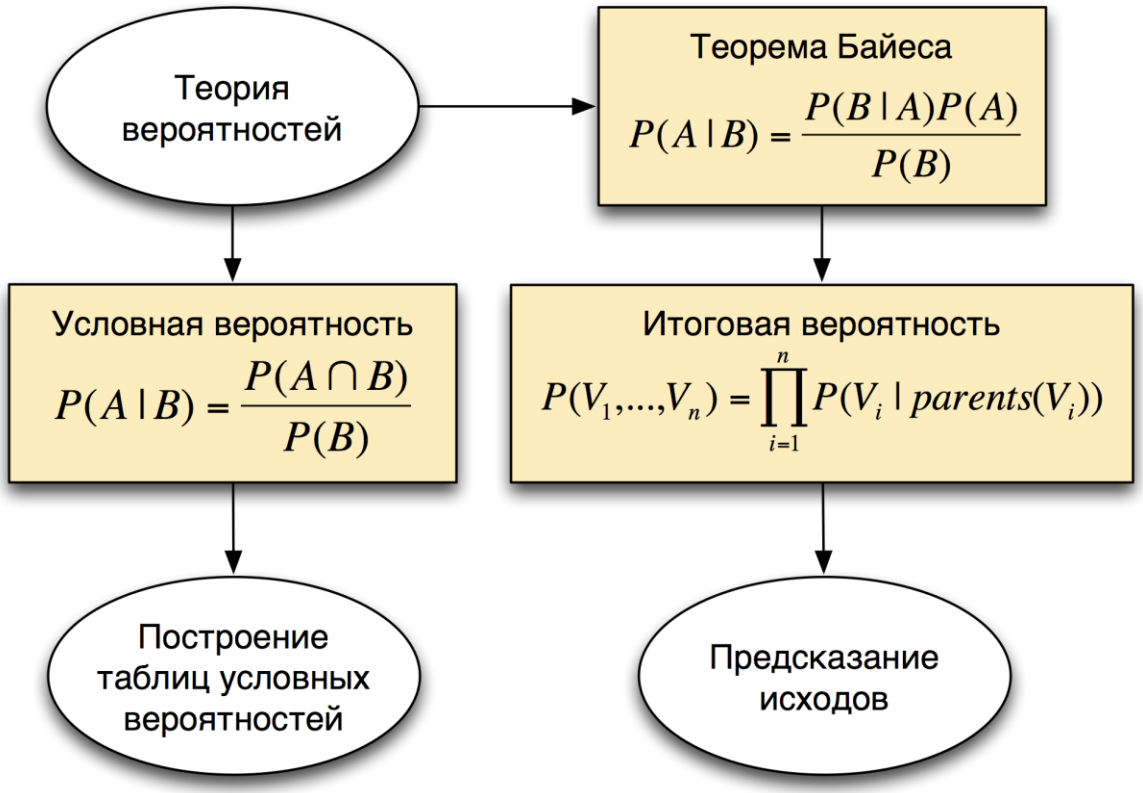


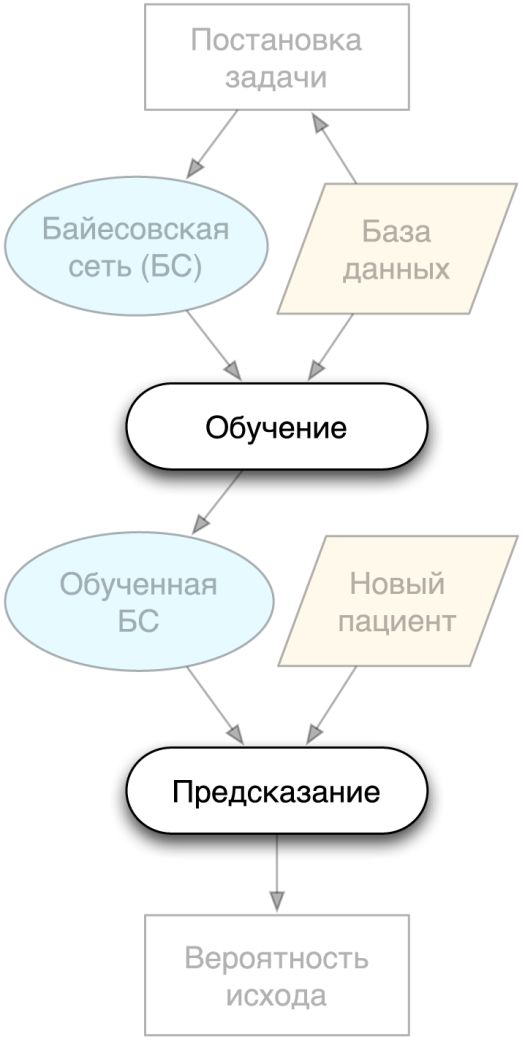
База данных

- Используются данные биохимических анализов
пациентов, физических показателей и данных геномики (полиморфизм, экспрессия и т.д.)
- Для параметров с непрерывными значениями проводится дискретизация путем разбиения на интервалы
- В базе данных должны быть представлены пациенты для всех вариантов исхода



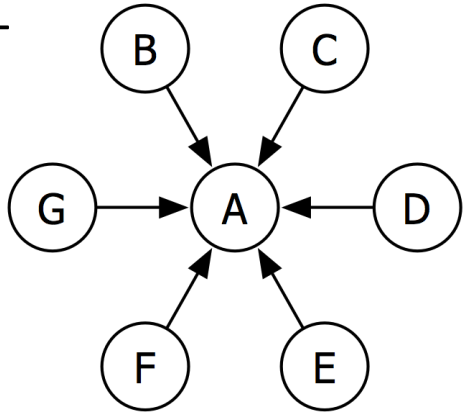
Вероятностная модель

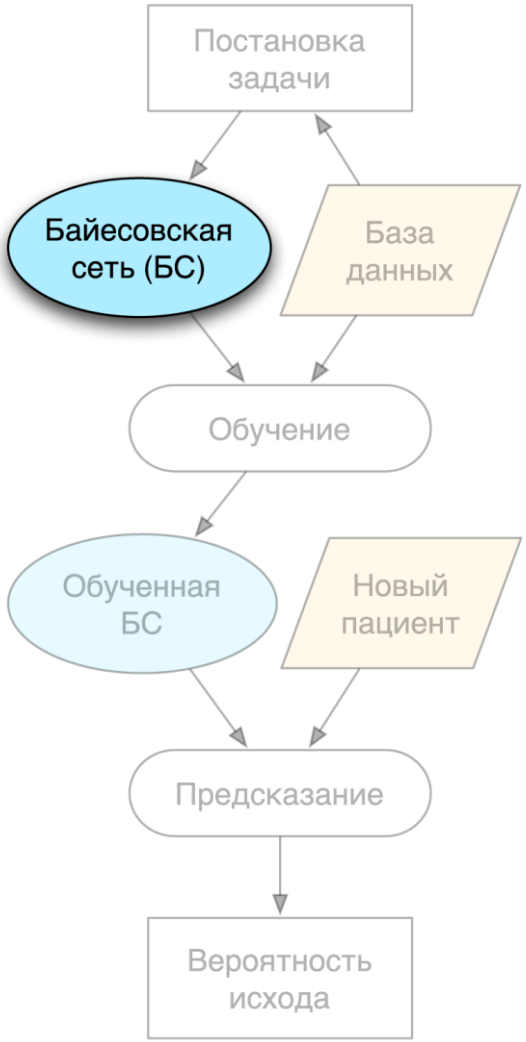




Байесовская сеть

- Байесовская сеть (БС) – это ориентированный ациклический граф
- Узлы БС связаны стрелками: причина-следствие

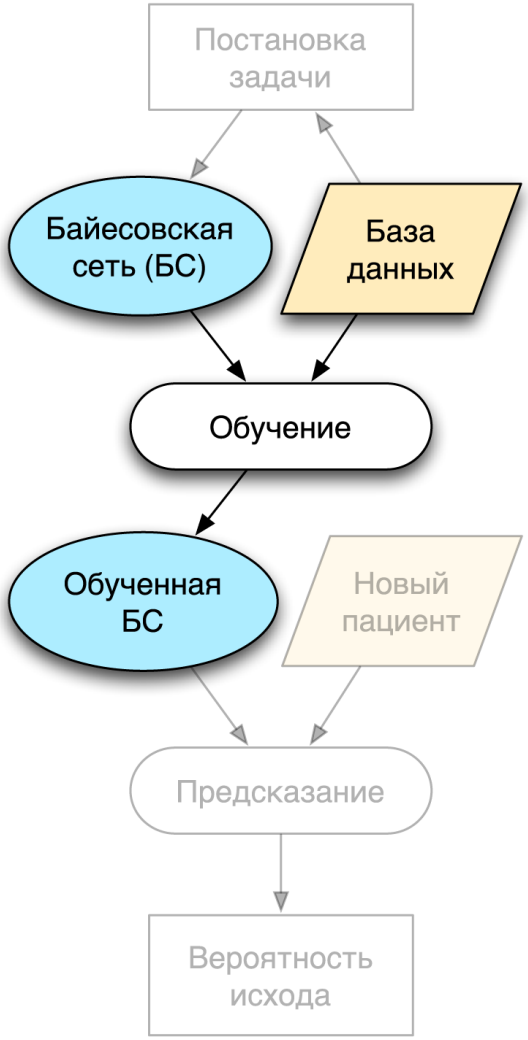




- Каждому узлу БС соответствуют таблицы условных вероятностей
- Каждому узлу БС соответствует один из выбранных параметров базы данных
- Конечной точкой (КТ) называется параметр соответствующий исходу события

Обучение

- Вычисление значений таблиц условных вероятностей осуществляется путем обучения БС на базе данных с известными исходами
- В результате получается обученная БС, содержащая заполненные таблицы условных вероятностей



Таблицы условных вероятностей

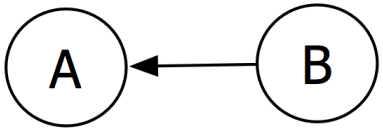


Таблица узла B

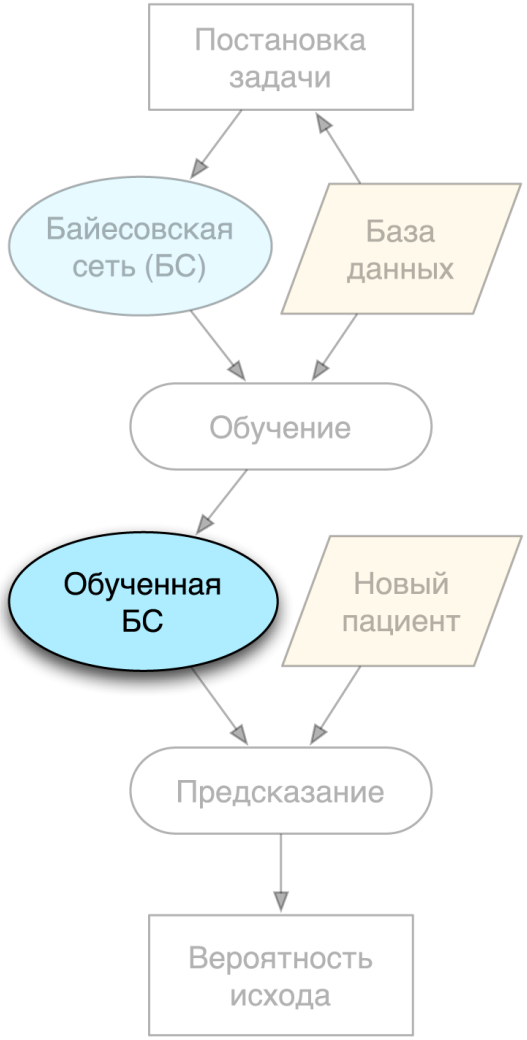
B = 0	B = 1
4/9	5/9

Таблица узла A

	A = 0	A = 1	A = 2
B = 0	1/4	2/4	1/4
B = 1	3/5	1/5	1/5

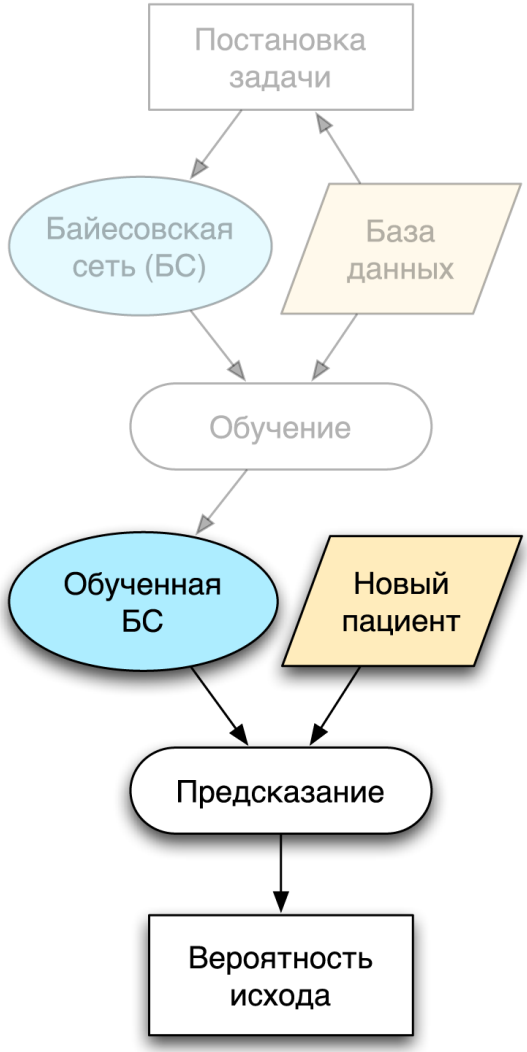
База данных

A	B
0	0
0	1
0	1
0	1
1	0
1	0
1	1
2	0
2	1



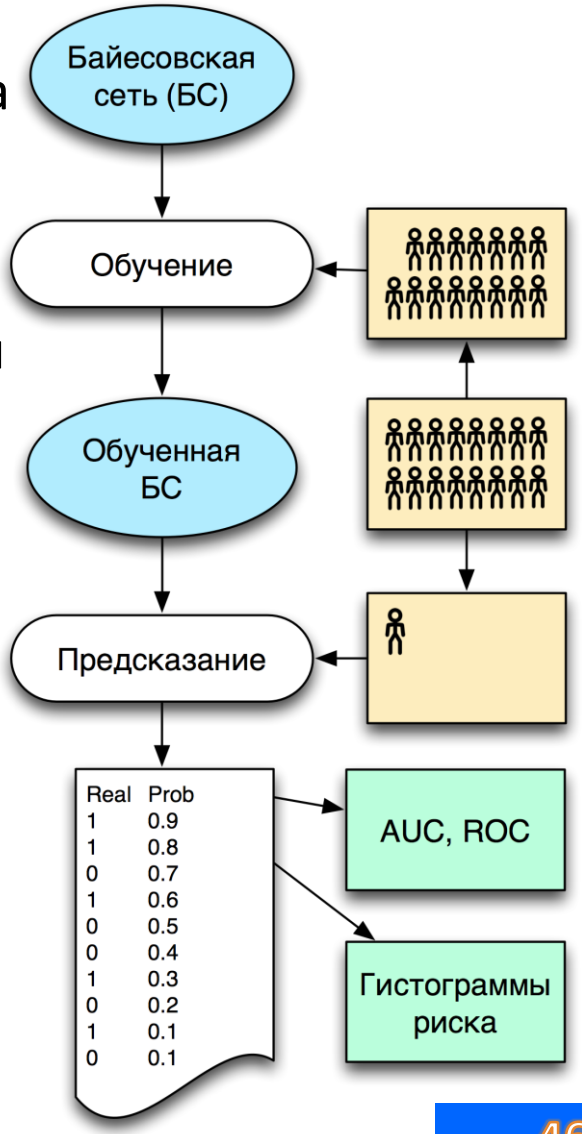
Предсказание

- С помощью обученной БС для новых пациентов можно вычислить условную вероятность исхода
- Вычисление осуществляется путем перемножения потенциалов таблиц условных вероятностей соответствующих значениям нового пациента



Валидация

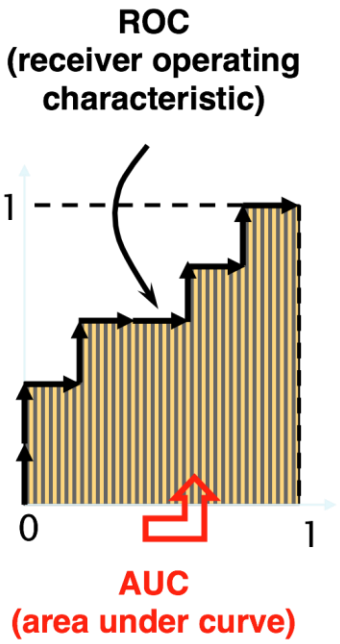
- Удаление из базы данных i-го пациента и обучение на ней БС
- Опрос (предсказание) обученной БС для i-го пациента и запись вероятности КТ в ROC-таблицу
- Добавление в базу данных i-го пациента и повторение для всех
- Сортировка ROC-таблицы
- Построение ROC-кривой и расчет площади под ней (AUC)
- Построение гистограмм риска



ROC-кривая

- ROC-кривая – интегральный итог проверки совпадения предсказаний с реальными исходами
- Качество предсказания различных БС можно сравнить, вычисляя для каждой из них величину AUC – площадь под ROC-кривой

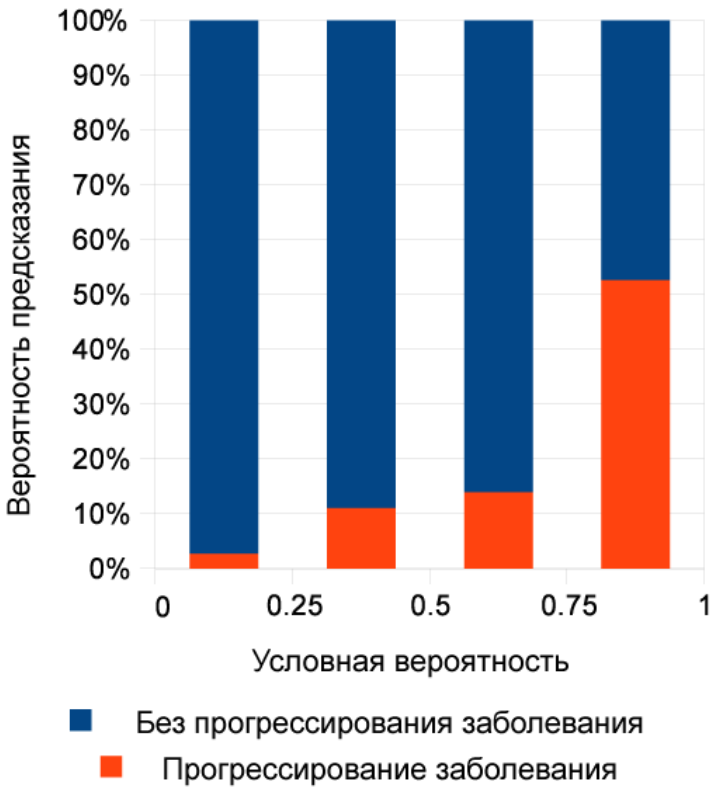
Real	Prob
1	0.9
1	0.8
0	0.7
1	0.6
0	0.5
0	0.4
1	0.3
0	0.2
1	0.1
0	0.1



- Чем ближе значение AUC к единице, тем выше качество предсказания – меньше ложных предсказаний
- Метод ROC-кривых адекватно сравнивает эффективность сетей с разным количеством переменных

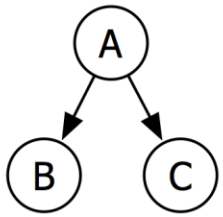
Гистограммы риска

- Гистограммы риска строятся на основании ROC-таблиц
- Условные вероятности разбиваются на N (4) одинаковых интервалов
- Каждый интервал соответствует определенной группе риска
- Гистограммы позволяют связать условную вероятность состояния КТ, с априорной вероятностью предсказания

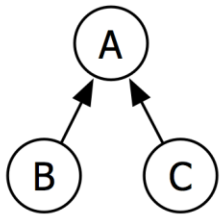


Изменение топологии

- По теореме Байеса с учетом независимости параметров БС можно поменять местами причину и следствие



$$P(A,B,C) = P(B \mid A) * P(C \mid A) * P(A) =$$
$$P(B,C \mid A) * P(A)$$



$$P(A,B,C) = P(A \mid B,C) * P(B) * P(C) =$$
$$= \frac{P(B,C \mid A) * P(A) * P(B) * P(C)}{P(B,C)} =$$
$$= \frac{P(B,C \mid A) * P(A) * P(B) * P(C)}{P(B) * P(C)} =$$
$$= P(B,C \mid A) * P(A)$$

теорема Байеса

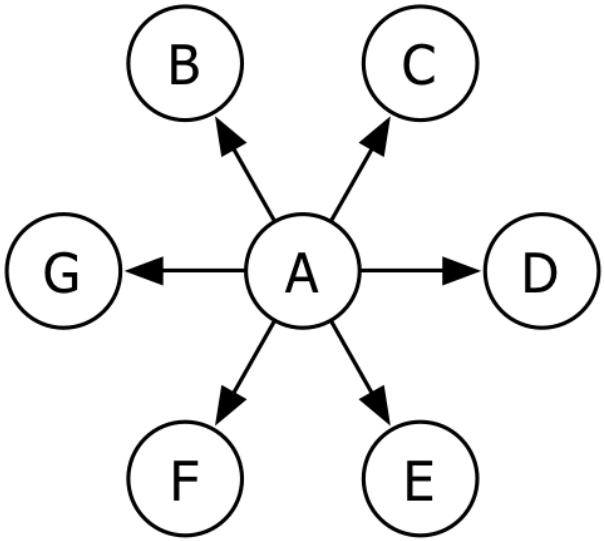
$$P(A \mid B,C) = \frac{P(B,C \mid A) * P(A)}{P(B,C)}$$

независимость параметров

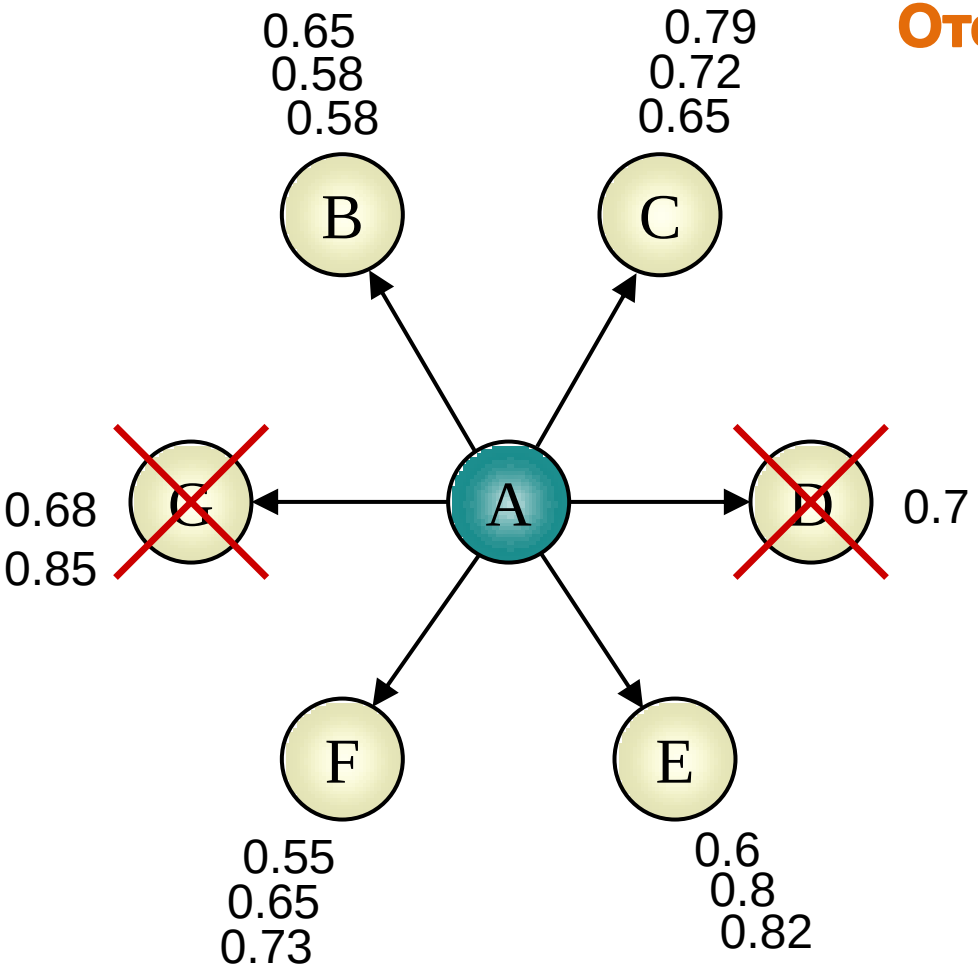
$$P(B,C) = P(B) * P(C)$$

Наивная топология

- Топология БС в которой узел A является КТ, а остальные узлы являются следствием КТ называется наивной
- Наивная топология позволяет упростить алгоритм обучения БС путем независимого вычисления таблиц условных вероятностей



Оптимизация сети по числу узлов



Отсутствие каких переменных повышает AUC?

Начальный AUC: 0.6

AUC: 0.7

AUC: 0.85

- Повышение качества предсказания
- Упрощение сети

Преимущества Байесовских сетей

- Работа с неполными наборами данных
- Генерации структурных и причинных взаимосвязей между компонентами
- Улучшение качества предсказания по мере накопления новых данных
- Использование предварительных знаний о системе и статистической информации, полученной в процессе экспериментальных наблюдений системы

Применение БС

- Предсказание повторного возникновения Острого Коронарного Синдрома (ОКС)
- Прогнозирования исходов рака молочной железы
- Прогнозирования иммунного ответа детей, привитых против кори, краснухи и эпидемического паротита
- Влияние генетических параметров на риск возникновения инсульта, а также на известные факторы риска развития атеросклероза
- Прогнозирование особенностей течения хронического гепатита С

Выводы

- Применение экспертных систем является одним из приоритетных направлений персонифицированной медицины
- Технология байесовских сетей успешно работает с базами данных пациентов, имеющими пропуски и содержащими большое количество параметров
- Оптимизация байесовских сетей приводит к увеличению качества предсказания и уменьшению числа прогностических параметров
- Предложенная модель экспертной системы успешно реализована и применена для нескольких проблем персонифицированной медицины

Спасибо за внимание



- *… Surely every medicine is an innovation; and he that will not apply new remedies, must expect new evils …*
- *… Каждый медицинский метод есть инновация; а кто не хочет применять новые средства, должен ждать новых бед …*

Sir Francis Bacon (1561-1626)

OF INNOVATIONS