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

СУПЕРКОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ

ВМК МГУ, 2017

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

СУПЕРКОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ

СТРУКТУРА АЛГОРИТМОВ

Вл.В.Воеводин

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

voevodin@parallel.ru

ВМК МГУ, 2017

Суперкомпьютерный комплекс МГУ

Суперкомпьютерный центр МГУ сегодня:
Пользователи: 2955
Проекты: 880

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

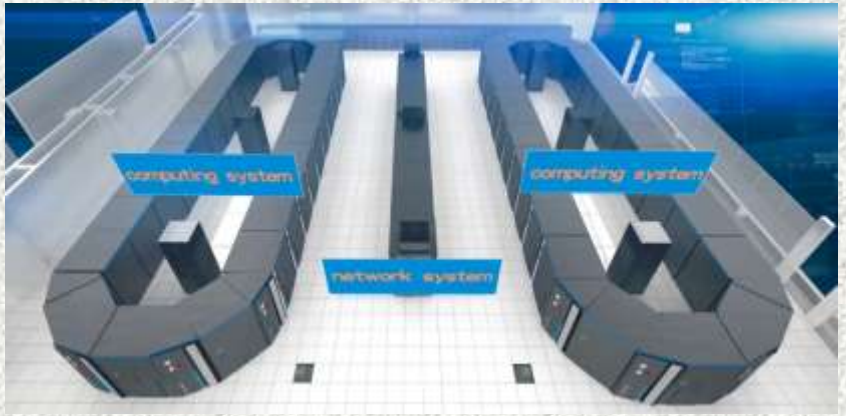
1 стойка = 256 узлов: Intel Xeon (14c) + NVIDIA K40= 515 Tflop/s

Суперкомпьютер "Ломоносов-2" (6 стоек) = 2.9 Pflop/s


Степень параллелизма
2005
2016
2025
10^4
10^6
10^9
Суперкомпьютеры
2-4
12-64
10^4
Серверы…
1
4-8
10^3
ПК, ноутбуки…
1
1-4
10^2
Планшеты, смартфоны…

Суперкомпьютер Sunway TaihuLight System, Китай

(#1 Top500 в 2016 г.)



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

Всего: 10 649 600 ядер

1,31 PBytes
15,4 MW (6 Gflops/W)

Производительность:
Peak: 125,4 Pflop/s
Linpack: 93 Pflop/s (74%)

Тонкий анализ суперкомпьютерных приложений: динамика исполнения

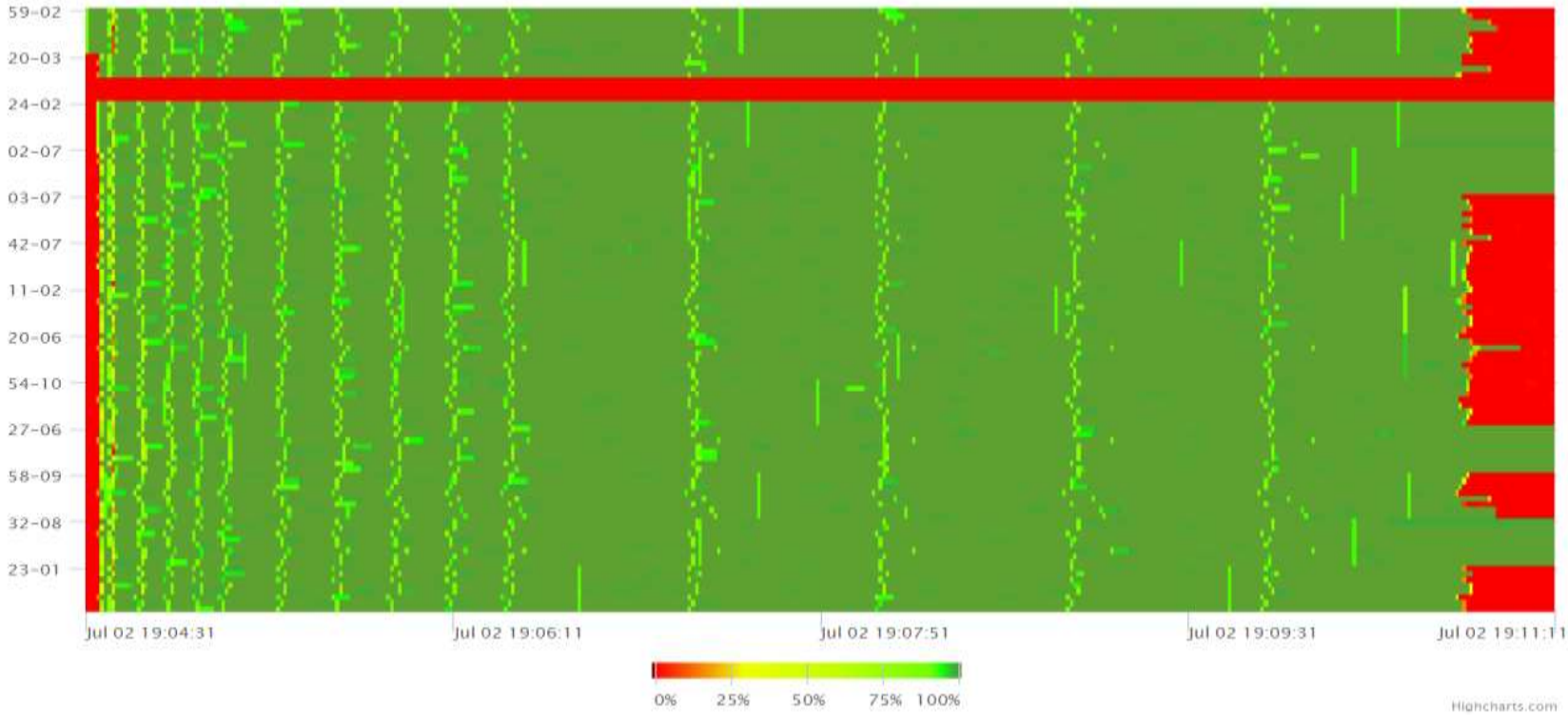
Информация о задаче № regular-1404302831-496016
пользователя

expand all

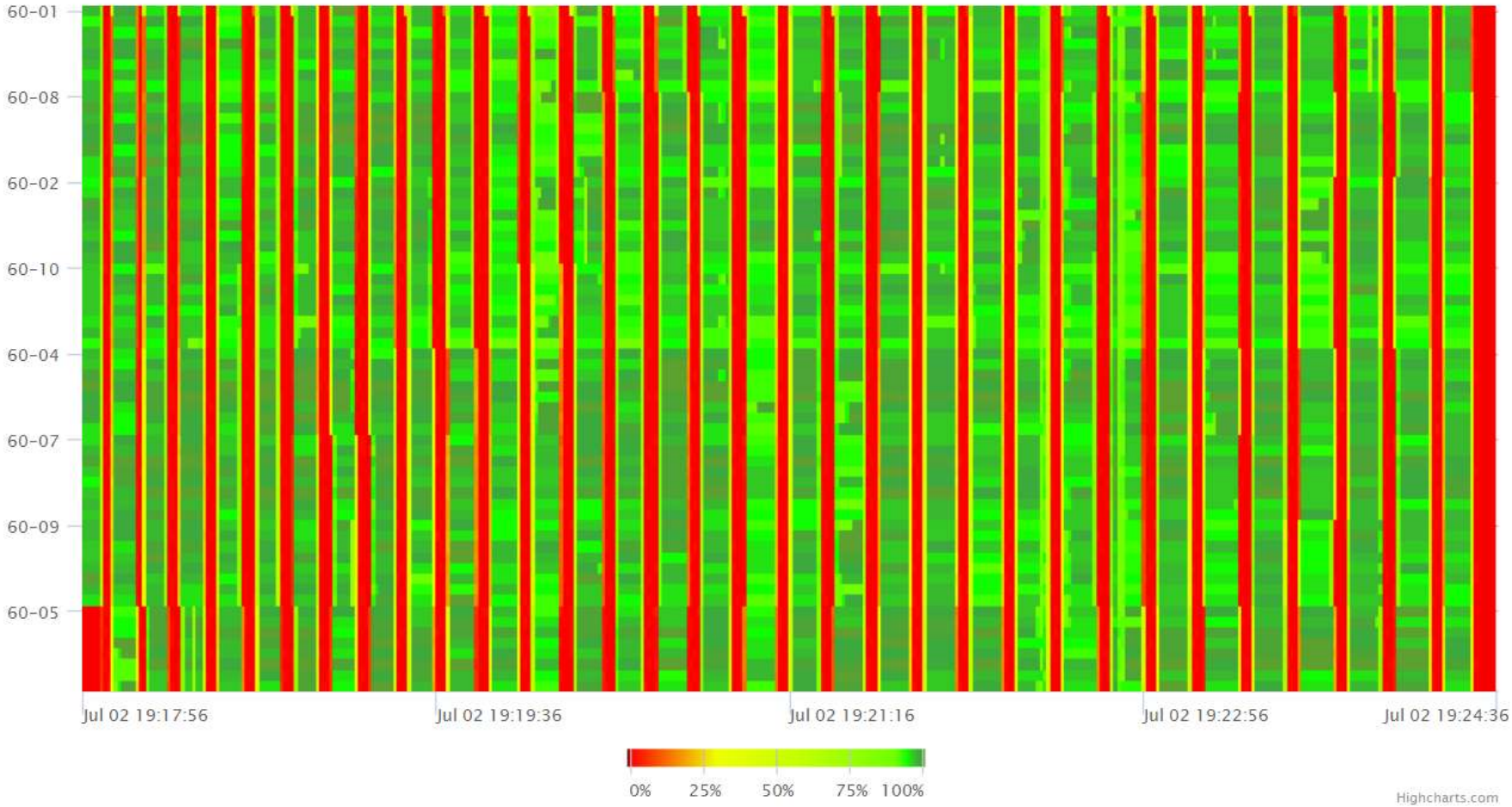
Информация о запуске

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

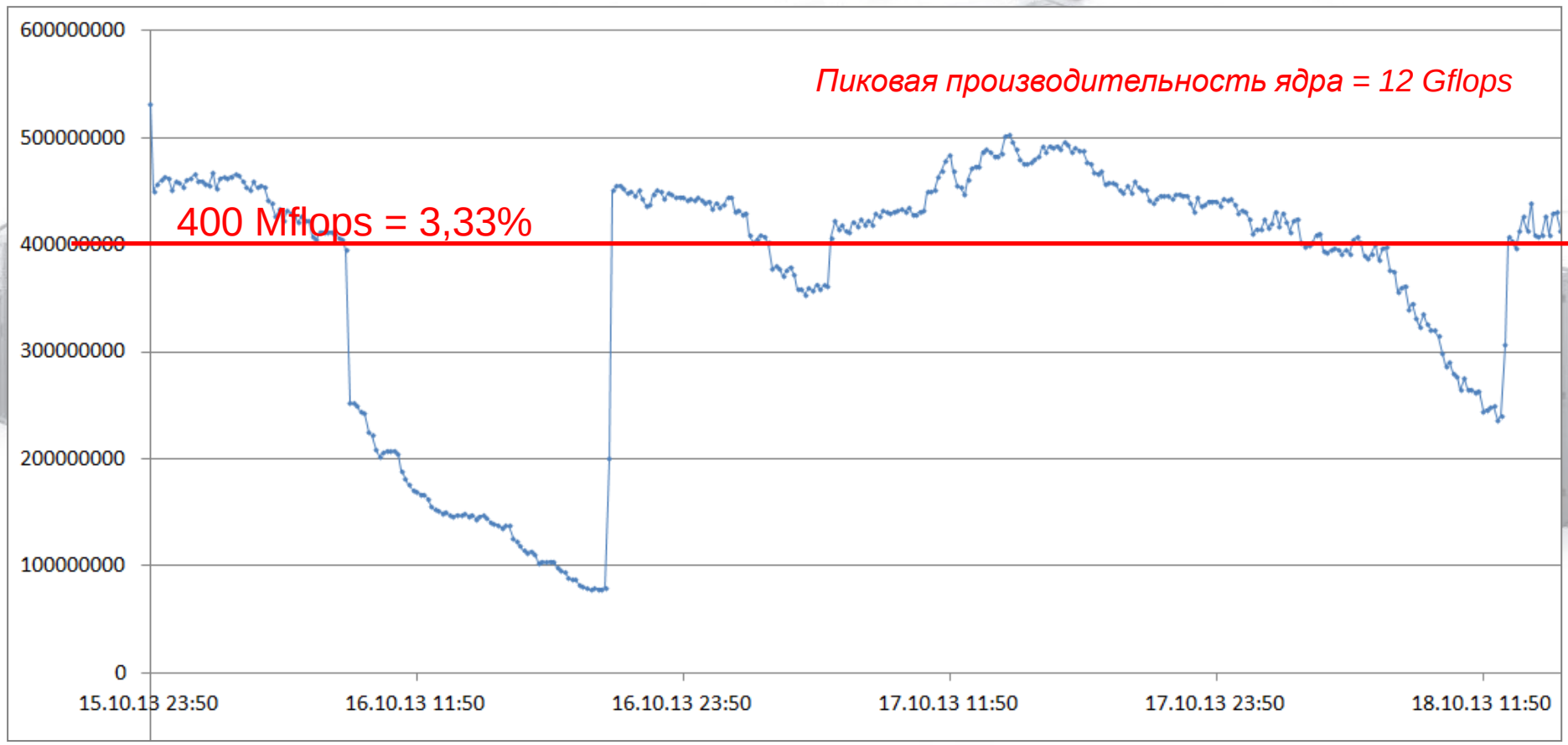


Тонкий анализ суперкомпьютерных приложений: динамика исполнения



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)

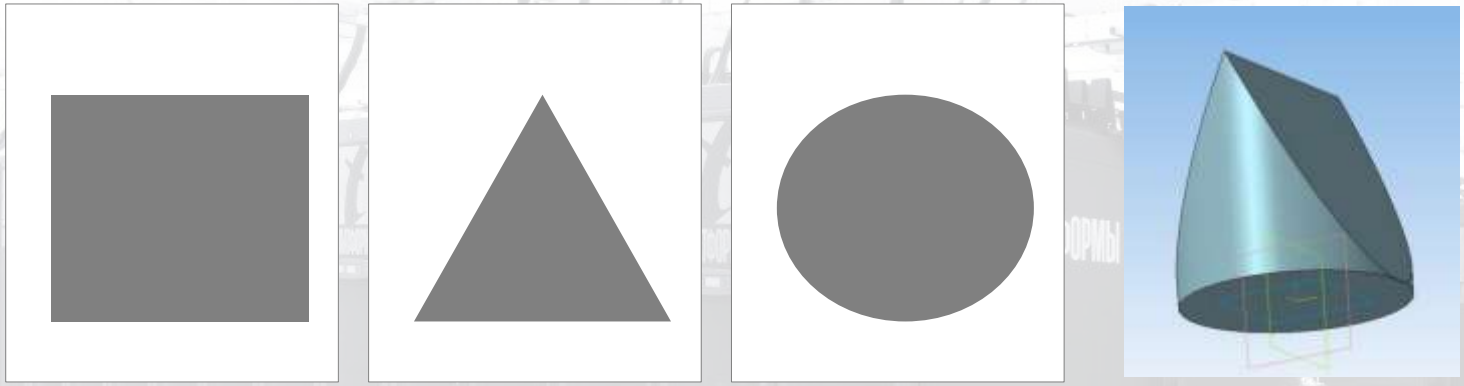
…

Изменения в архитектуре компьютеров не меняют алгоритмов! Но:

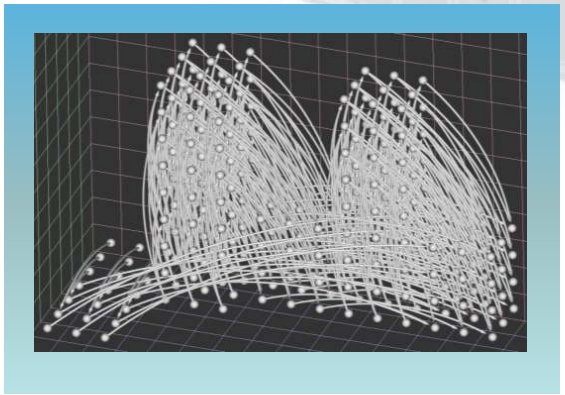
Для каждого поколения компьютеров мы вынуждены:
- Анализировать алгоритмы, чтобы понять, как их приспособить под новую компьютерную платформу ;
- Описывать найденные свойства, чтобы получить эффективную реализацию для новой платформы.

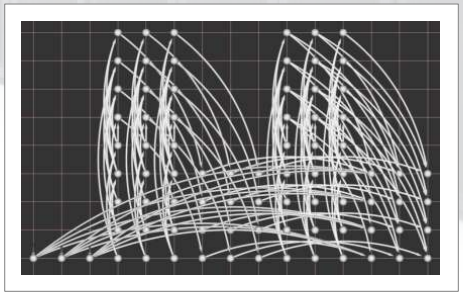
Изменения в архитектуре компьютеров не меняют алгоритмов!

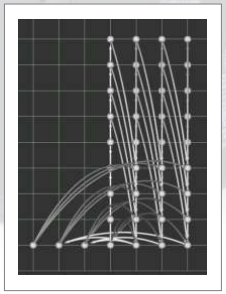
Эти фигуры разные?

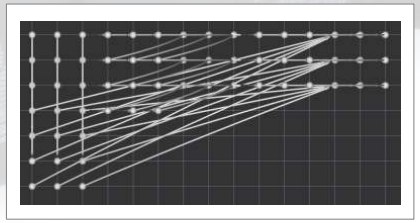


Каковы способы представления этого алгоритма?









…

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…

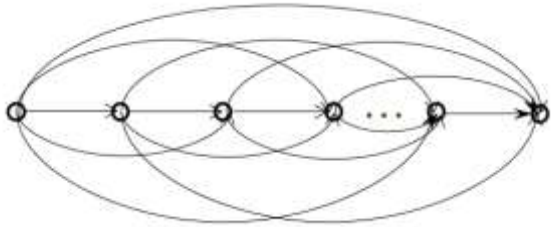
Хорошо ли мы знаем свойства, особенности, статические и динамические характеристики параллельных алгоритмов?

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.

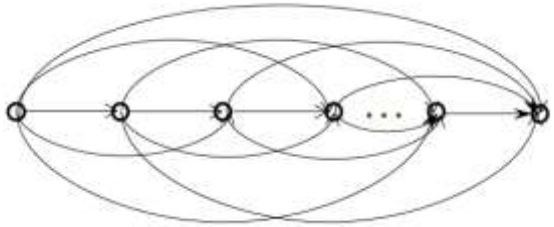
На выходе: n вершин, n(n-1)/2 дуга.

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. ⇨ ☹

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

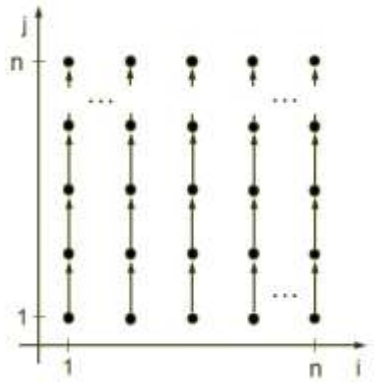
Тест Linpack (решение линейной системы): $\approx n$

Поэлементное сложение двух векторов: 1/3

Параллелизм бывает неудобным

(Что нужно знать про алгоритмы)

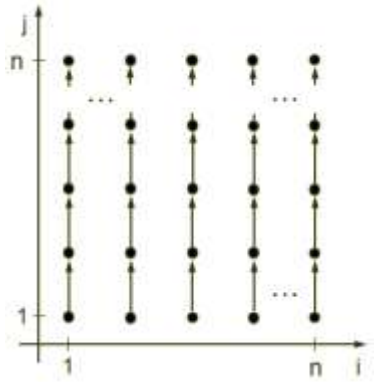
```
#pragma OpenMP parallel for
for( i = 1 ; i <= n ; ++i)
    for( j = 1 ; j <= m ; ++j)
        A[i][j] = (A[i][j] * A[i][j-1]) / 2 ;
```



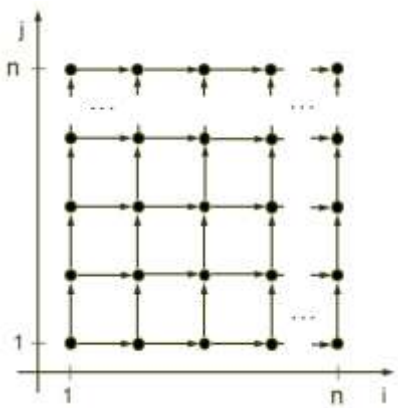
Параллелизм бывает неудобным

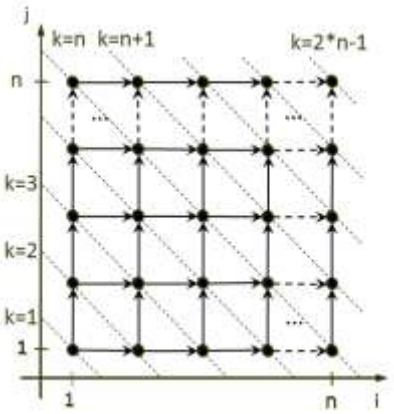
(Что нужно знать про алгоритмы)

```
#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 ;
```

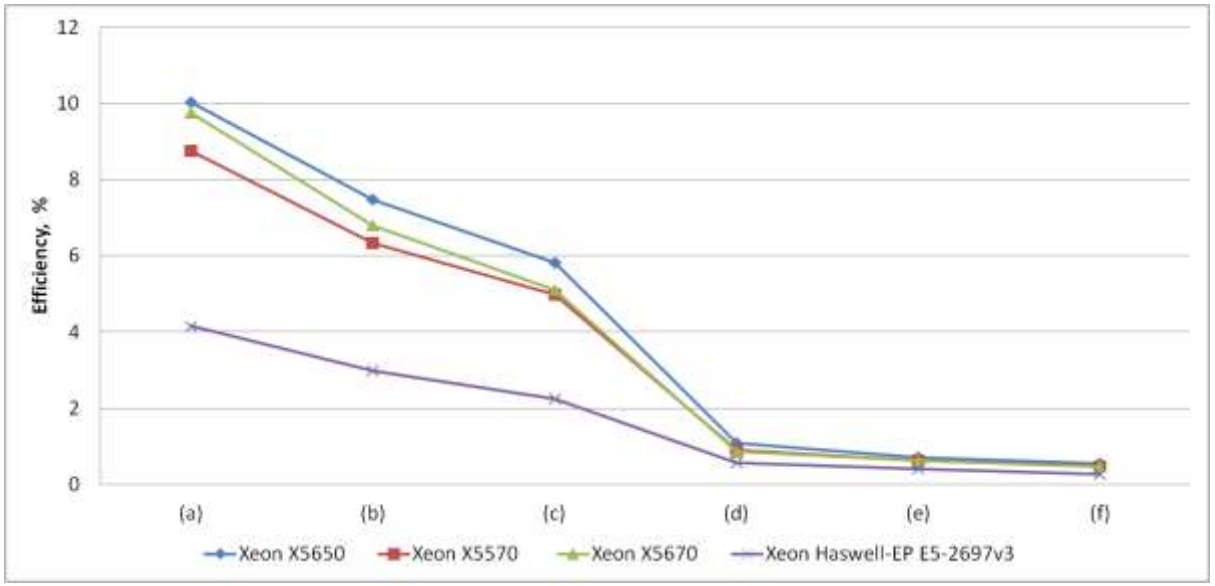




Локальность определяет многое

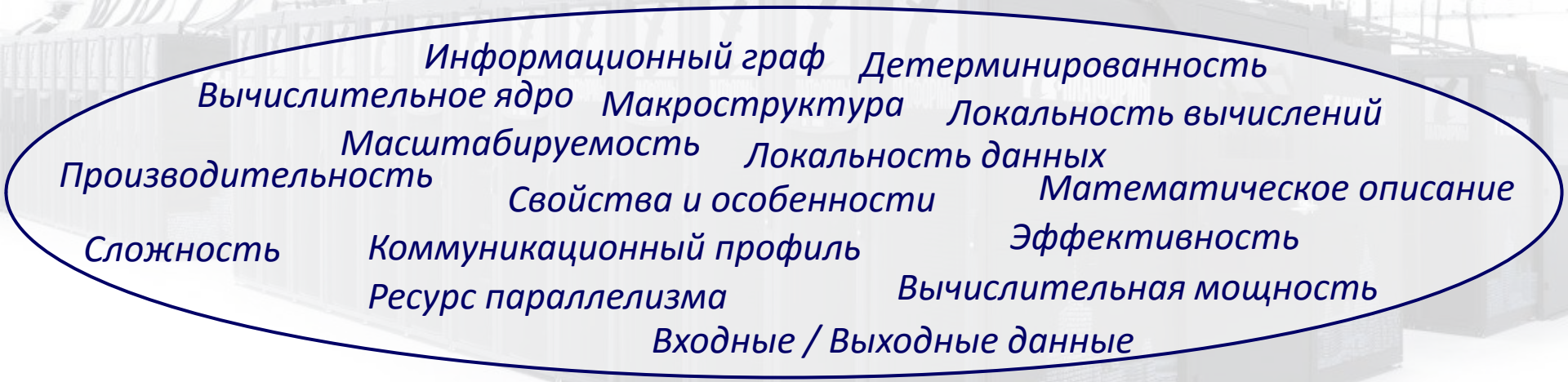
(Что нужно знать про алгоритмы)

(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

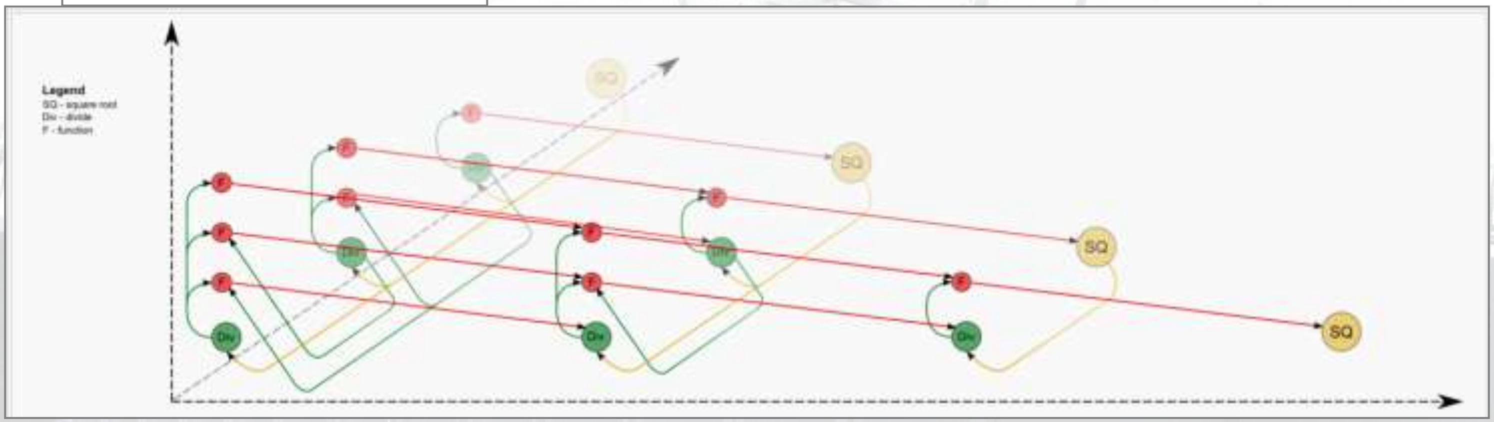
```
DO  I = 1, N
        S = A(I,I)
        DO  IP=1, I-1
                S = S - DPROD(A(I,IP), A(I,IP))
        END DO
        A(I,I) = SQRT (S)
        DO  J = I+1, N
                S = A(J,I)
                DO  IP=1, I-1
                        S = S - DPROD(A(I,IP), A(J,IP))
                END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

Дополнительная информация

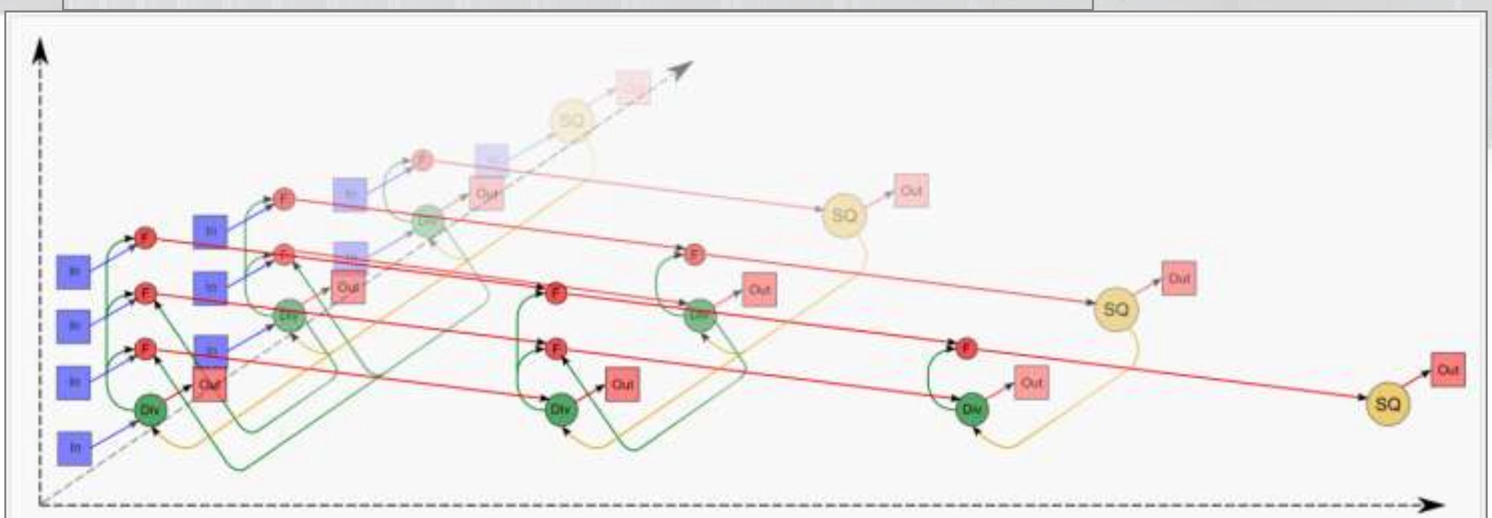
There exist block versions of this algorithm;

Описание алгоритмов
(на примере разложения Холецкого)

Информационная структура



Информационная структура с указанием входных и выходных данных



Описание алгоритмов
(на примере разложения Холецкого)

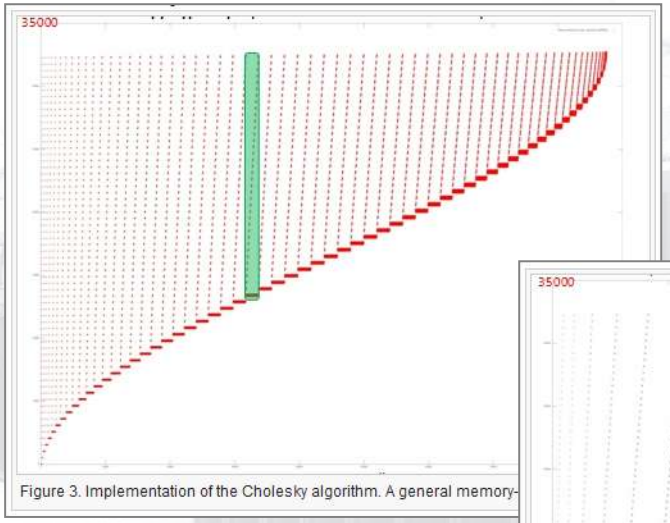


Figure 3. Implementation of the Cholesky algorithm. A general memory-

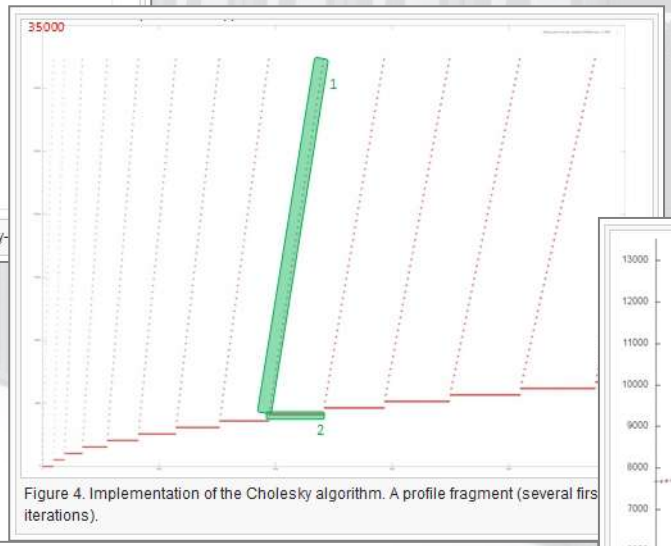


Figure 4. Implementation of the Cholesky algorithm. A profile fragment (several firs iterations).

Локальность данных (профиль работы с памятью)

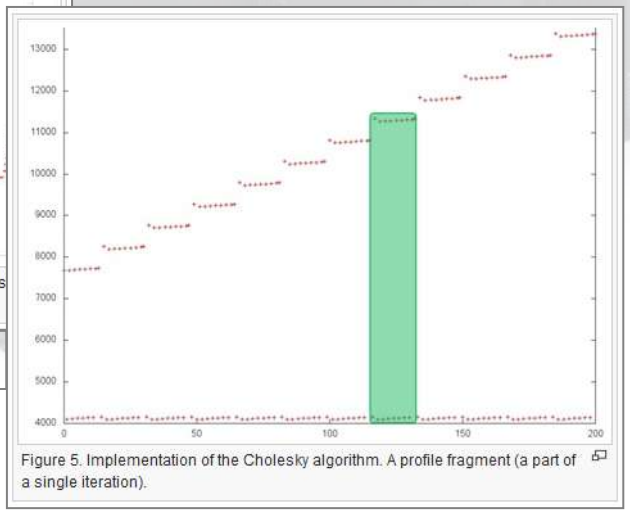
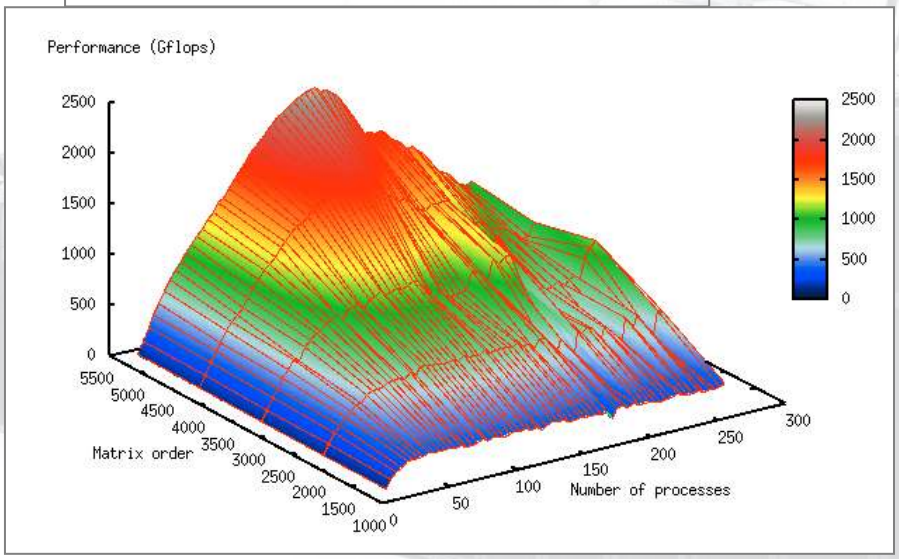


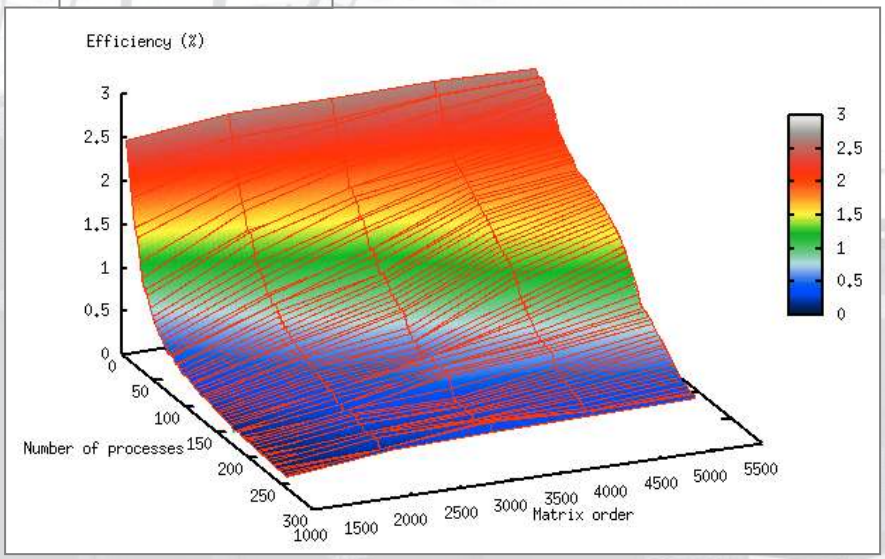
Figure 5. Implementation of the Cholesky algorithm. A profile fragment (a part of a single iteration).

Описание алгоритмов
(на примере разложения Холецкого)

Масштабируемость (производительность) *



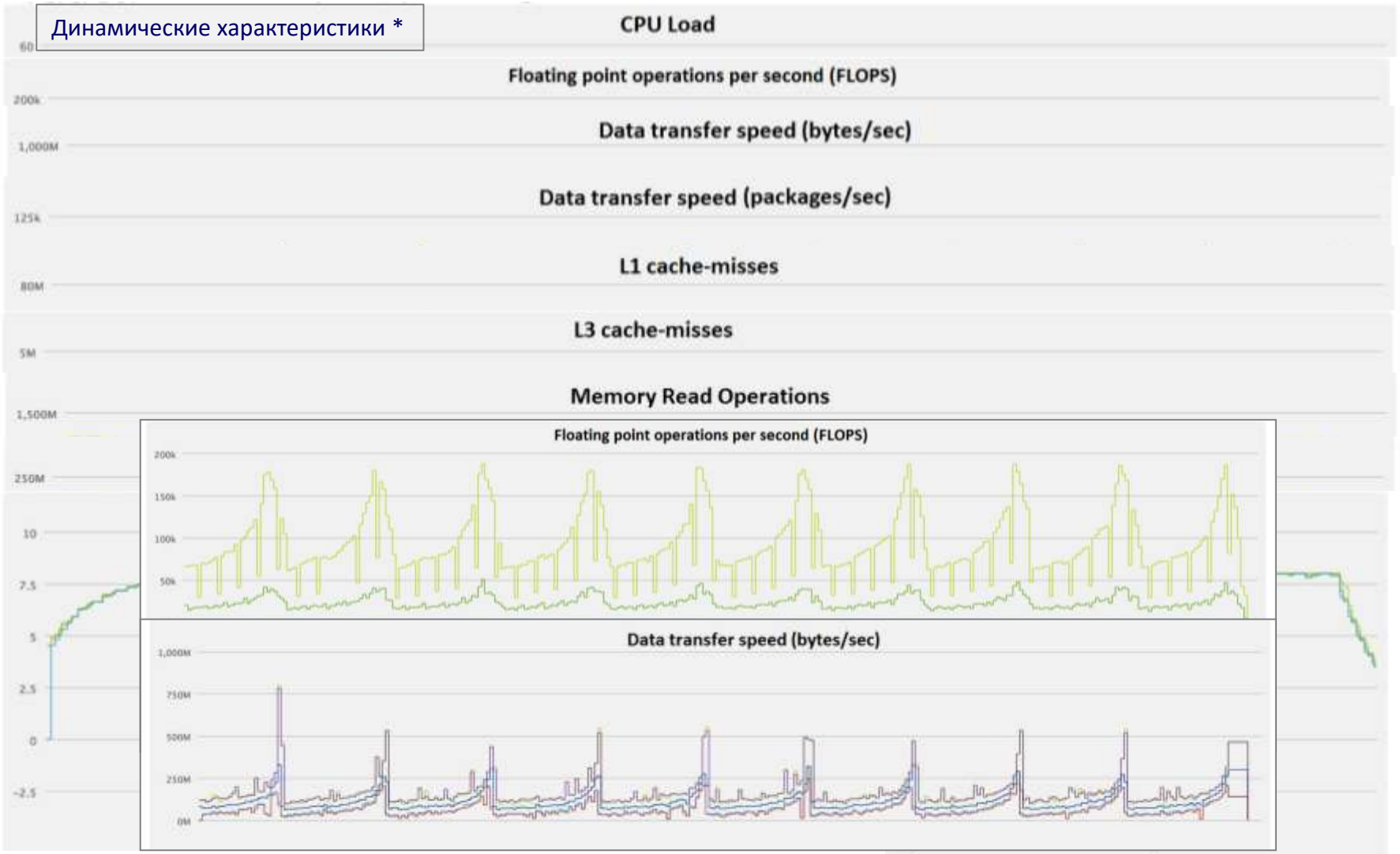
Эффективность *



* 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.

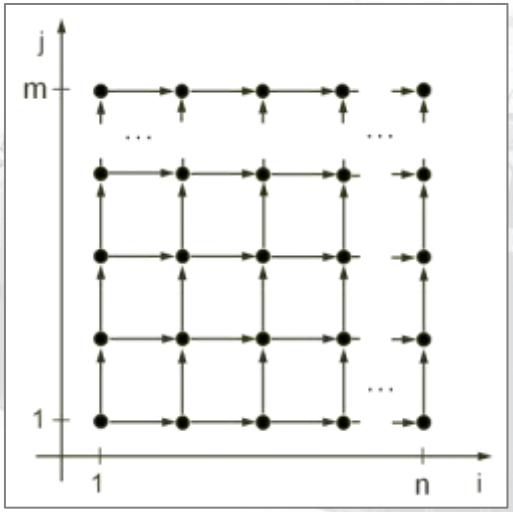
Но…

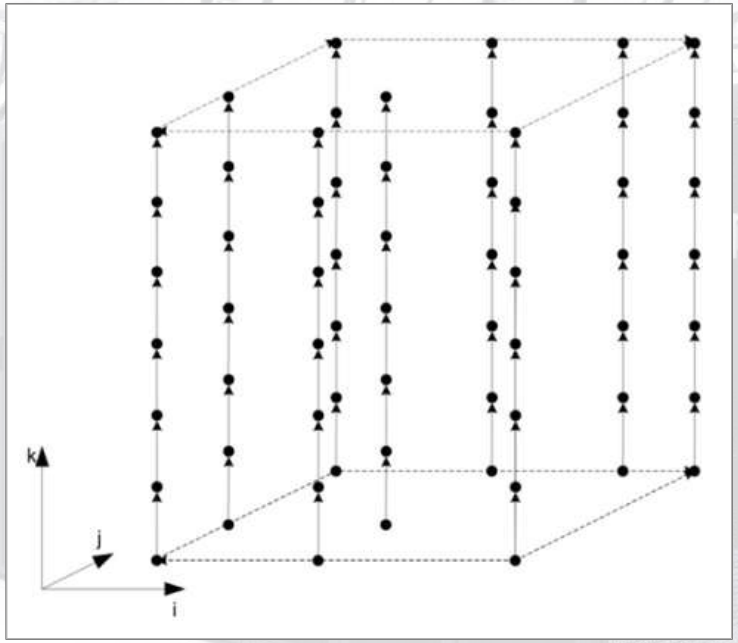
В процессе создания полного описания алгоритма возникает не просто много проблем.

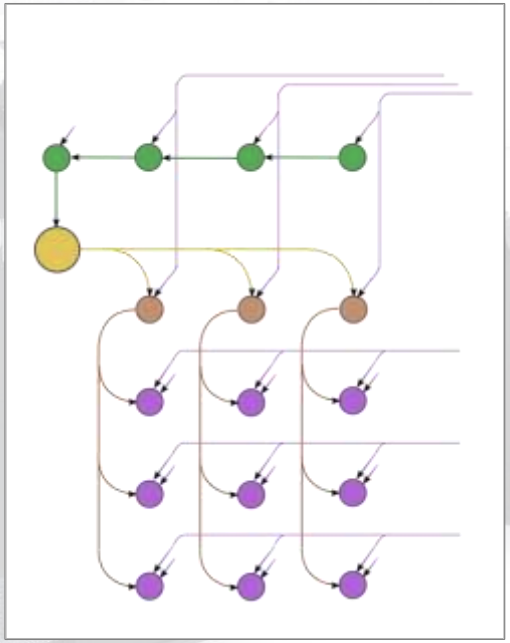
Возникает очень много сложных проблем !

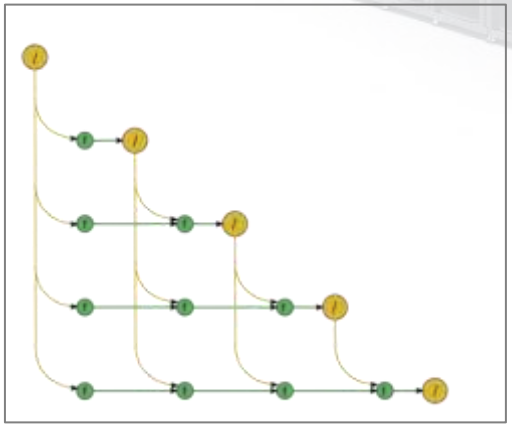
Информационная структура: как получать, описывать, показывать… ?

(сложности описания алгоритмов)





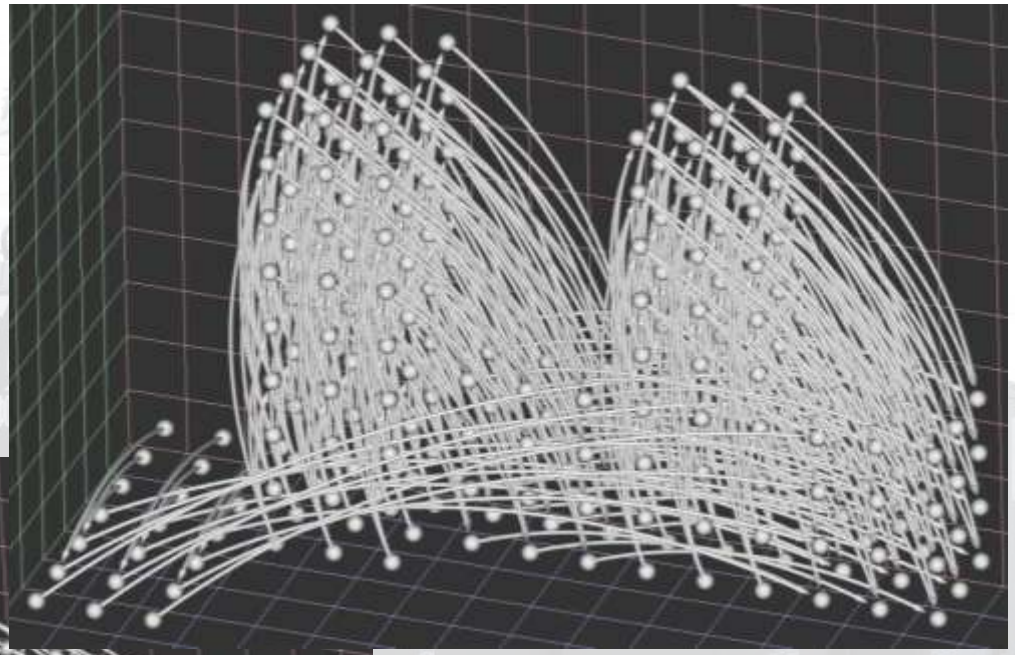


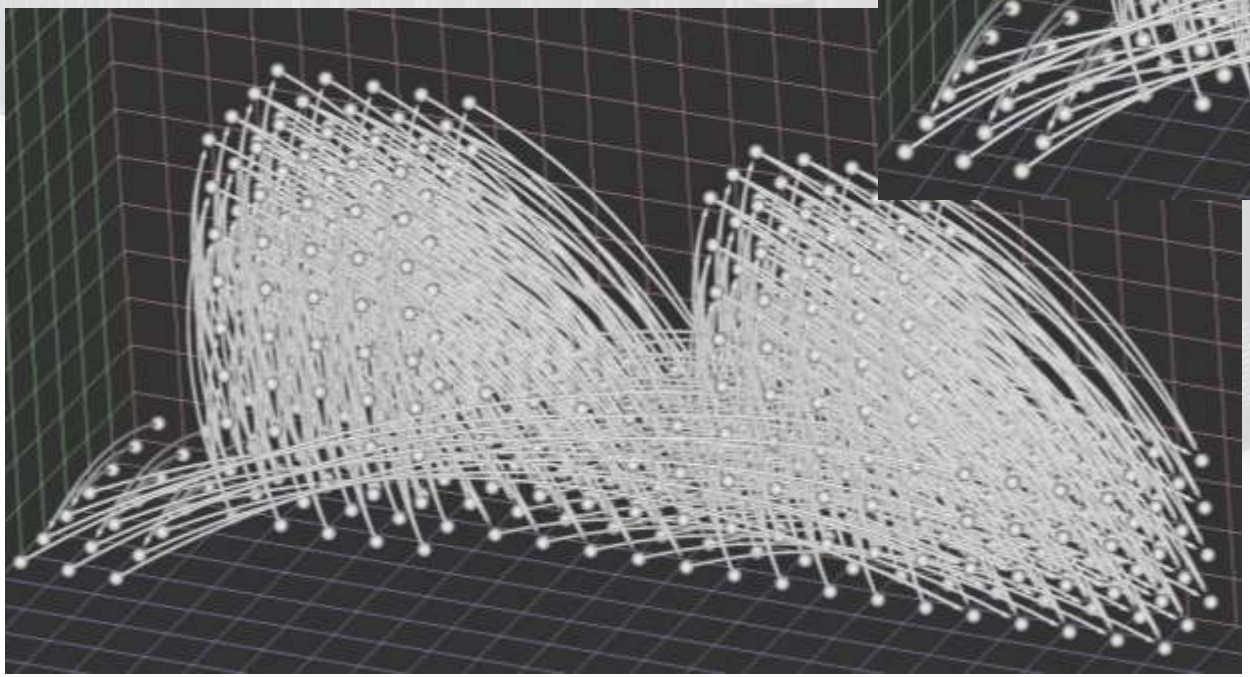


- Как изобразить потенциально бесконечный граф ?
- Как изобразить потенциально многомерный граф ?
- Как показать зависимость структуры графа от размера задачи ?

Информационная структура: как получать, описывать, показывать… ?

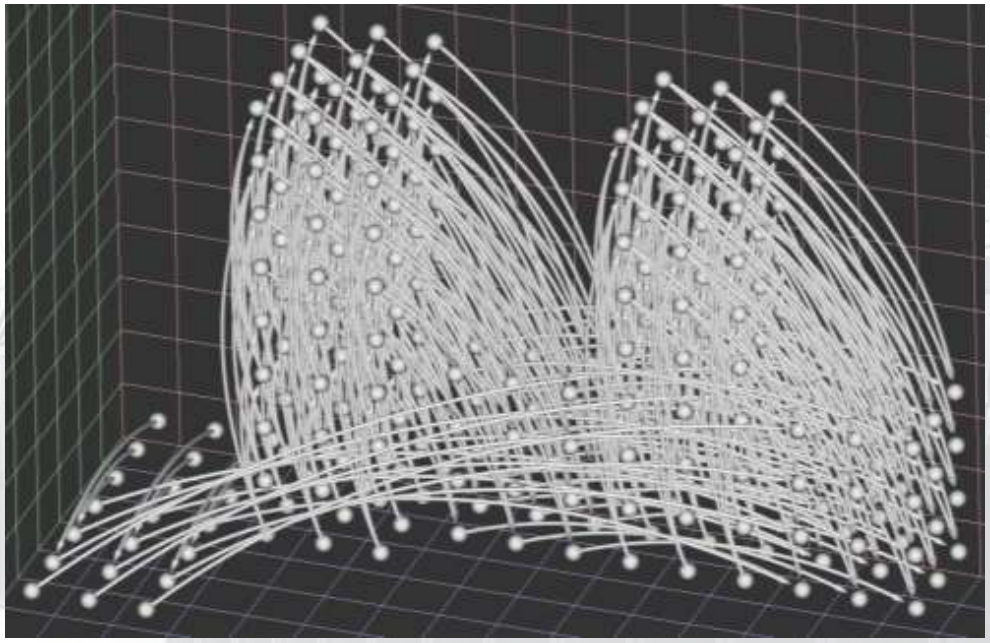
(сложности описания алгоритмов)

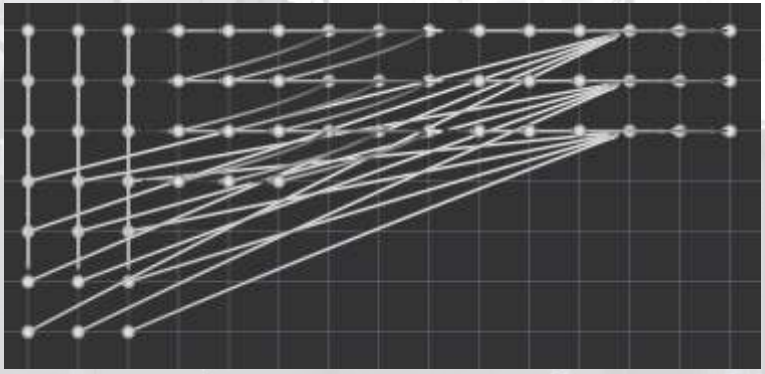


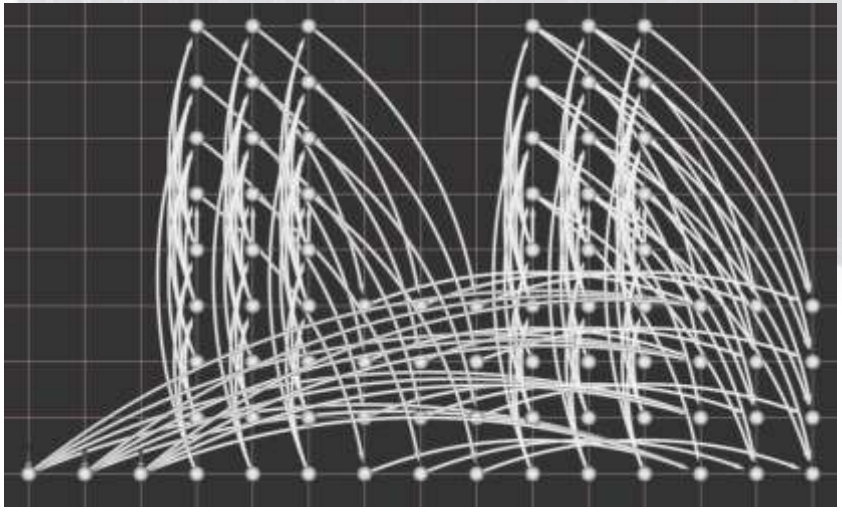


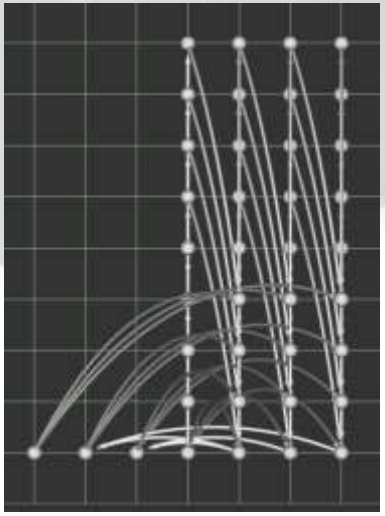
Информационная структура: как получать, описывать, показывать… ?

(сложности описания алгоритмов)









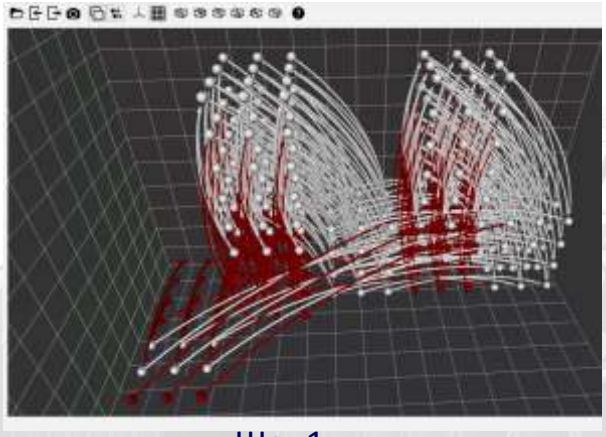
Информационная структура: как получать, описывать, показывать… ?

(сложности описания алгоритмов)

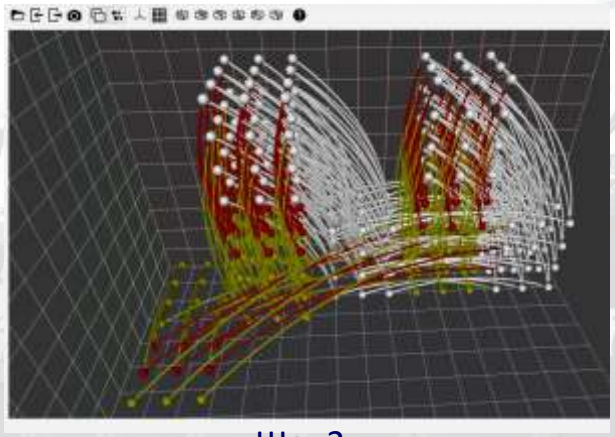
Как выразить имеющийся параллелизм и показать возможный способ параллельного исполнения ?

Информационная структура: как получать, описывать, показывать… ?

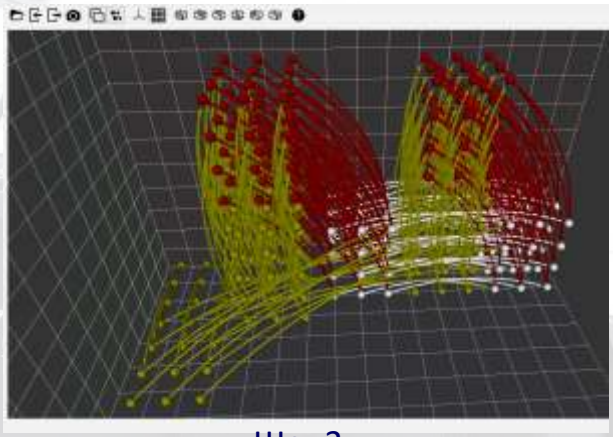
(сложности описания алгоритмов)



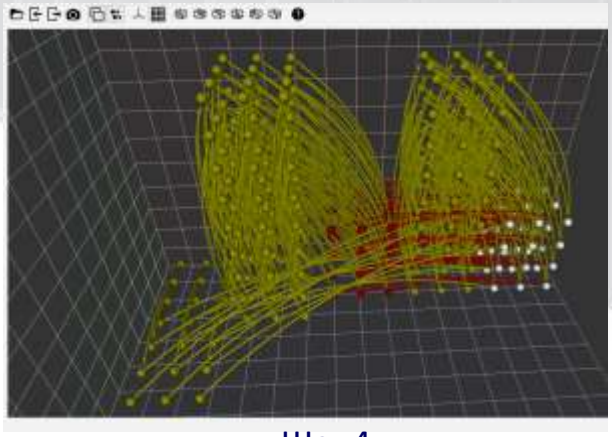
Шаг 1



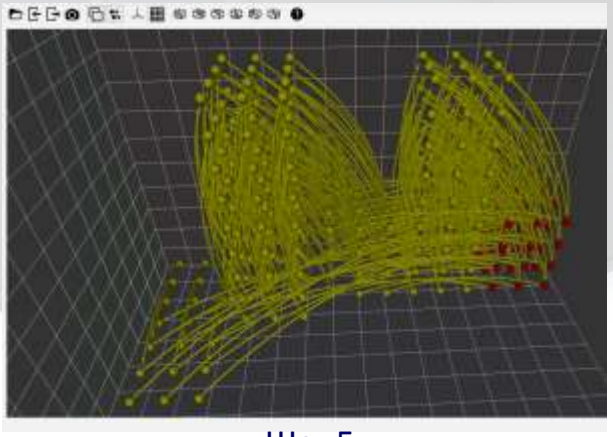
Шаг 2



Шаг 3



Шаг 4



Шаг 5

Цвета в ярусно-параллельной форме:

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

Белый – операции должны выполниться позже.

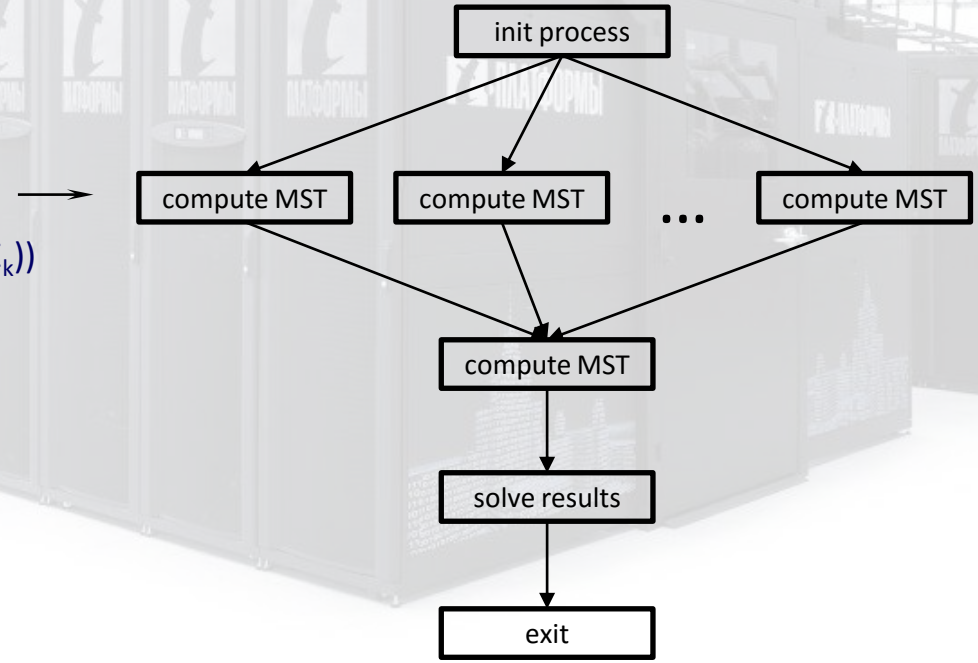
Потенциальный параллелизм: как находить, описывать, показывать… ?

(сложности описания алгоритмов)

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

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

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

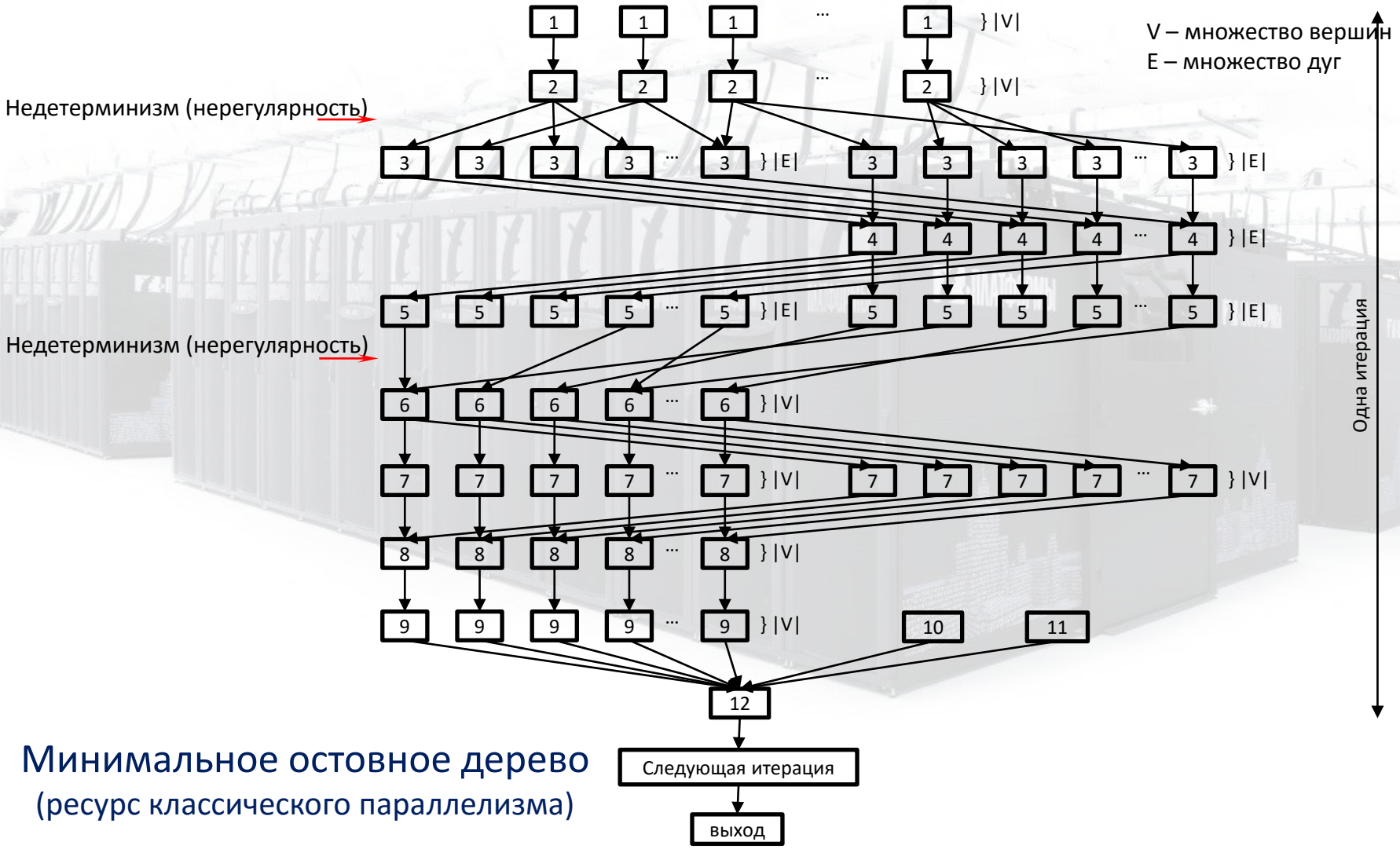


Минимальное остовное дерево (MST)

(ресурс “математического” параллелизма)

Потенциальный параллелизм: как находить, описывать, показывать… ?

(сложности описания алгоритмов)



Минимальное остовное дерево

(ресурс классического параллелизма)

Структура алгоритмов: возможные варианты для параллельного кода

(сложности описания алгоритмов)

Предположим, что мы знаем потенциальный параллелизм алгоритма...
Как выразить “потенциальную свободу” в выборе походящей формы для параллельной программы ?

Структура алгоритмов: возможные варианты для параллельного кода

(сложности описания алгоритмов)

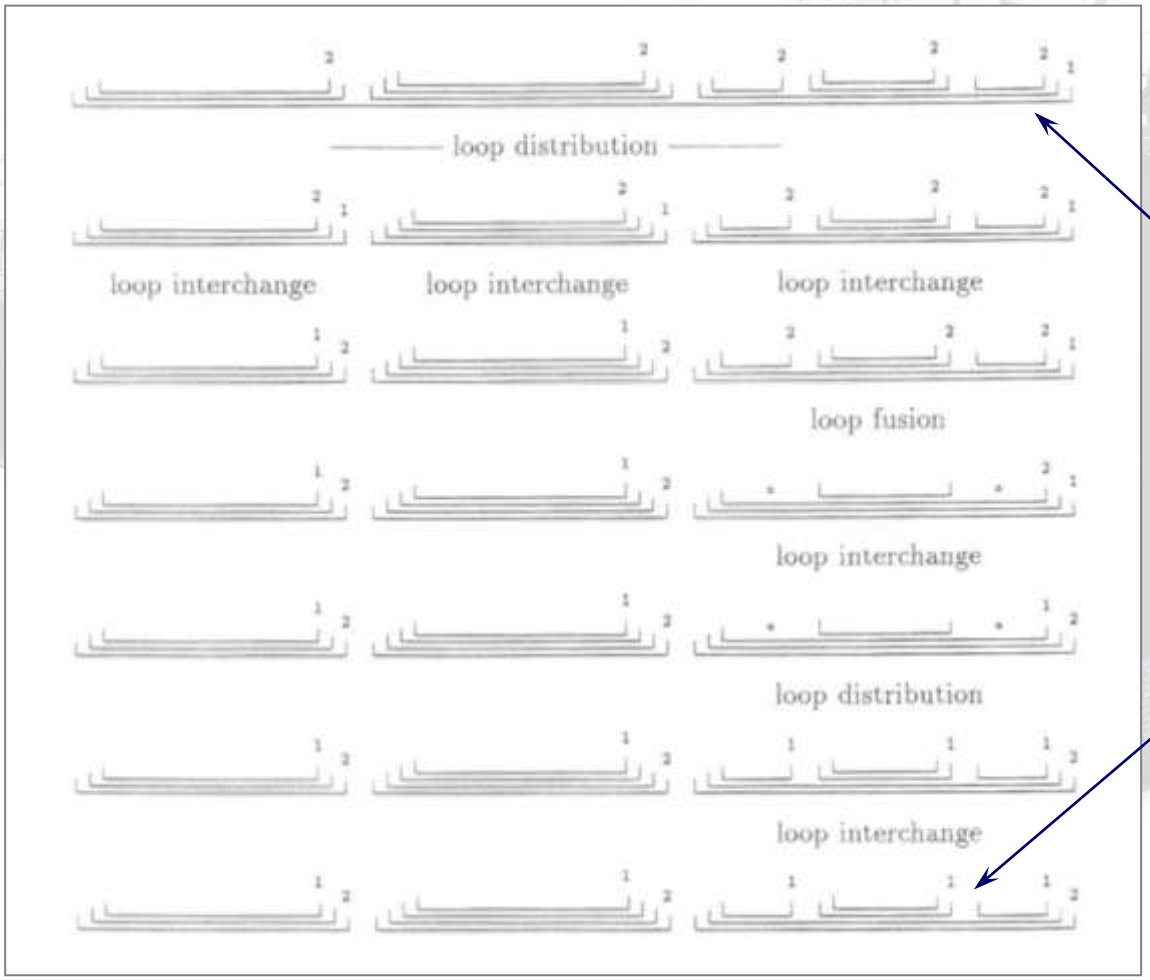
Циклический профиль алгоритма / программы

Самый внешний цикл (отмечен “1”) - параллельный.

Можно ли сделать данный цикл самым внутренним, чтобы векторизовать?

Структура алгоритмов: возможные варианты для параллельного кода

(сложности описания алгоритмов)



Эта последовательность преобразований делает исходный внешний цикл (1) внутренним и позволяет векторизовать программу (алгоритм).

Как показать подобный потенциал преобразований в общем случае?

Возможные источники несбалансированности: это важно, это нужно отметить

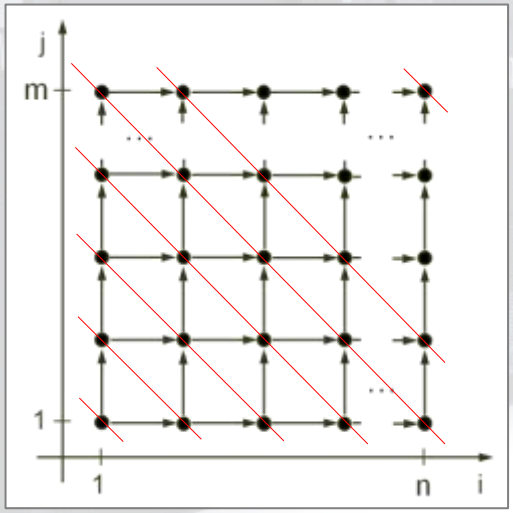
(сложности описания алгоритмов)

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;

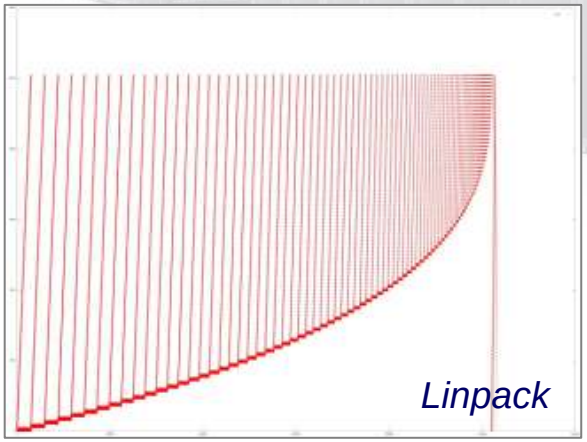
Отсутствие повторяемости результатов из-за разного порядка выполнения ассоциативных операций…

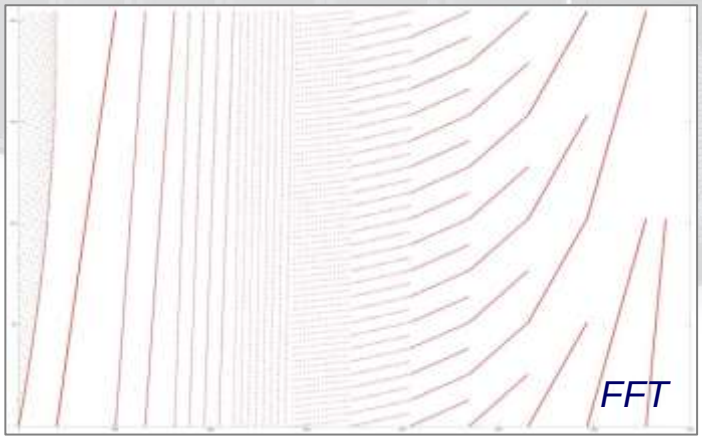
Локальность данных: множество открытых вопросов
(сложности описания алгоритмов)

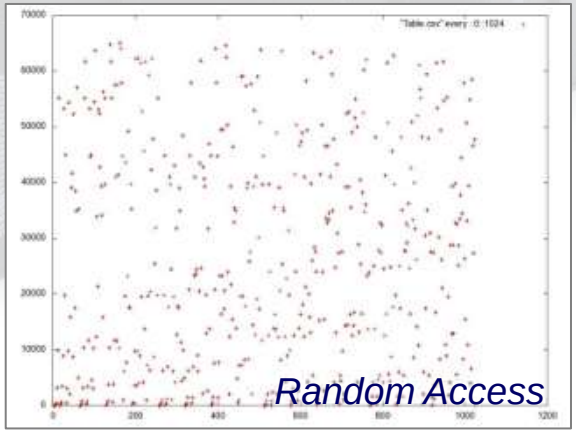
Как оценивать пространственную* и временную** локальность данных в программе ?

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)

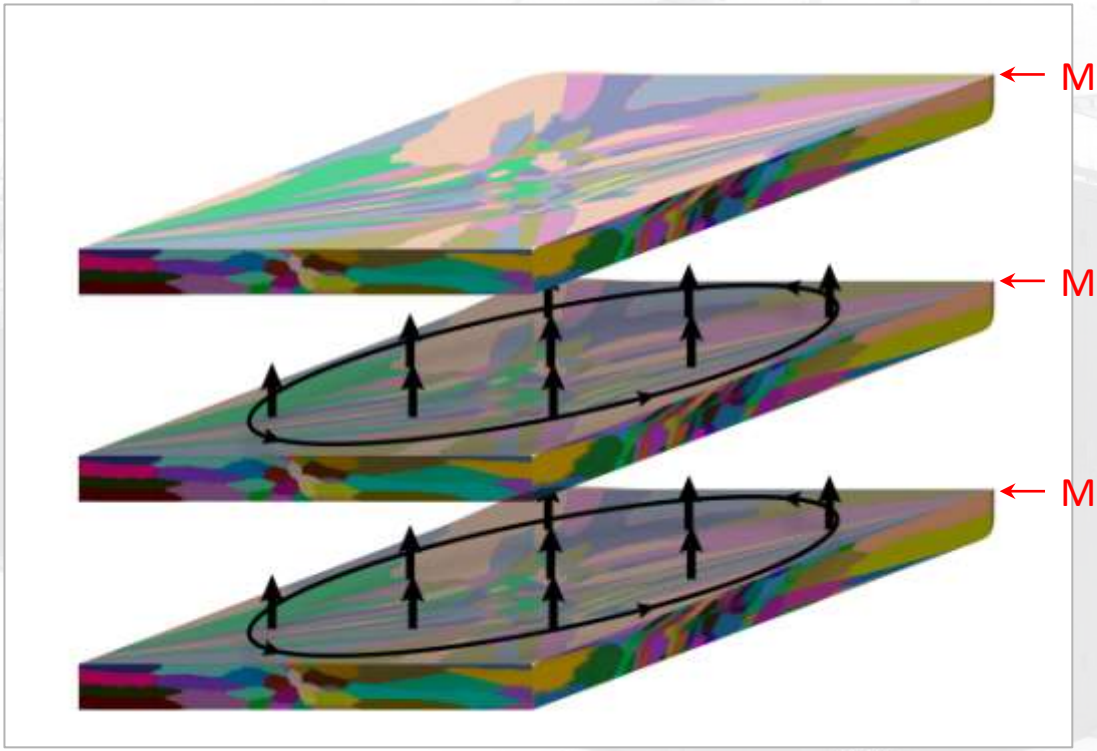
Как оценивать пространственную и временную локальность данных в программе ?

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 ?

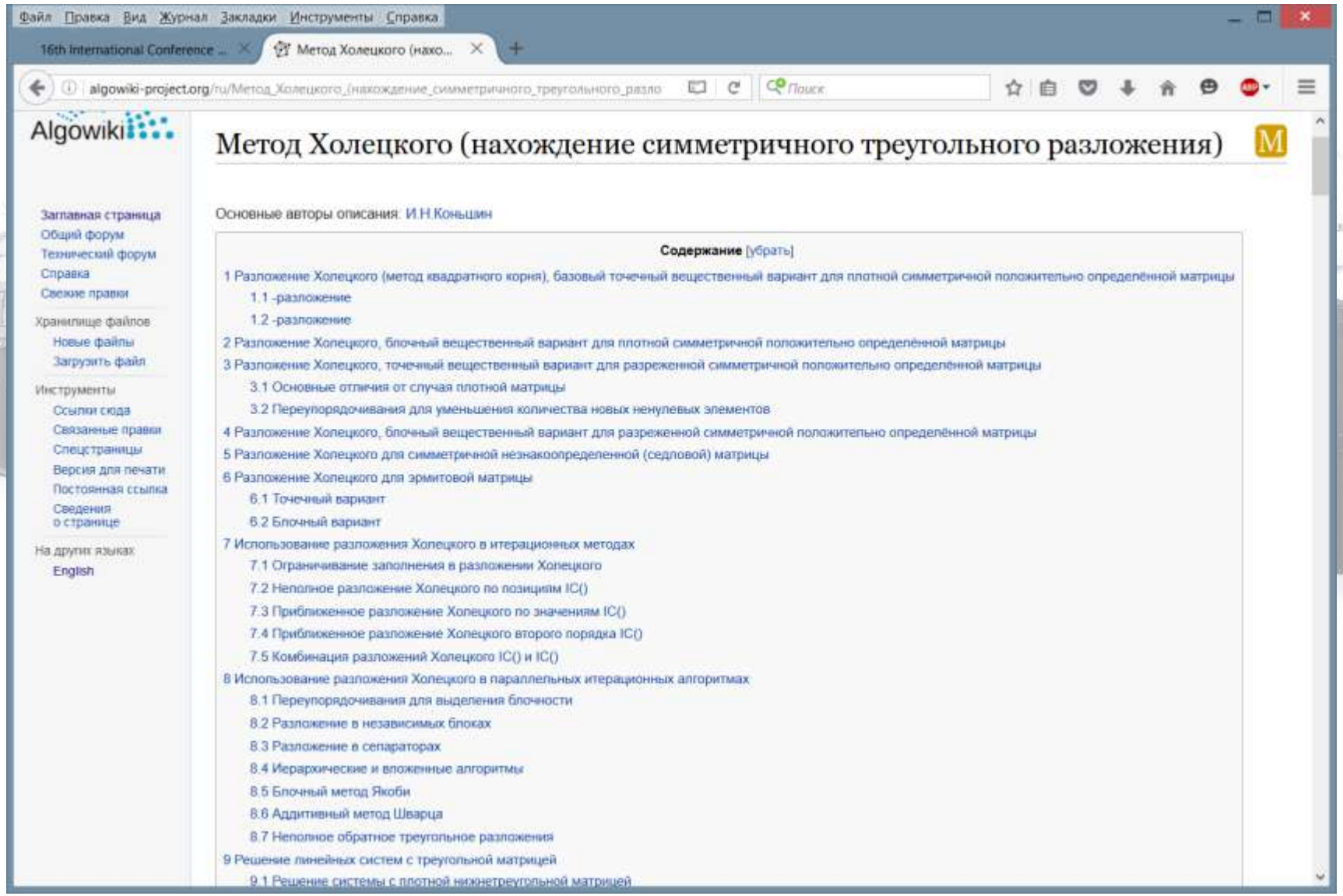
Да… Кажется, что Да…

Фундаментальный вопрос:

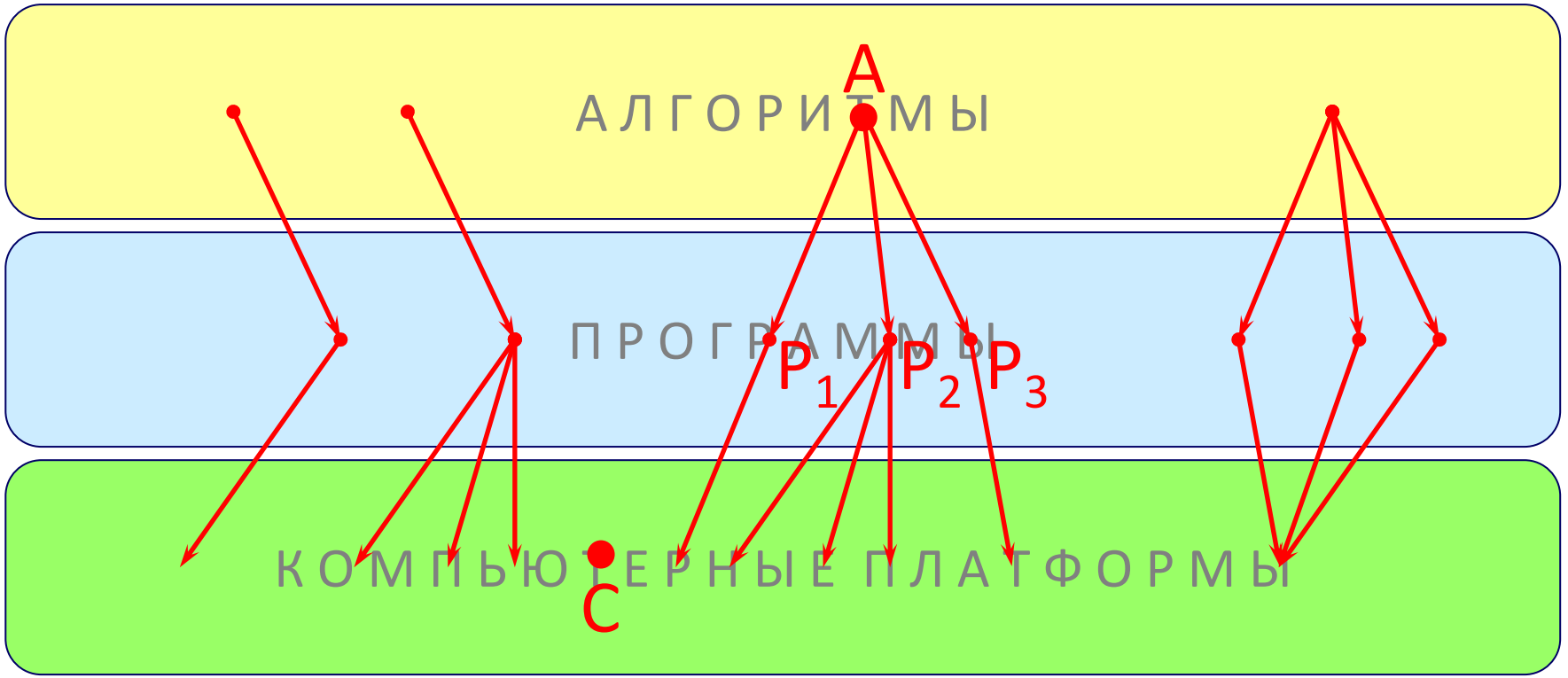
Что означает выполнить полное описание алгоритма?

Пока ответа не знает никто...

Разновидности одного алгоритма…
(сложности описания алгоритмов)



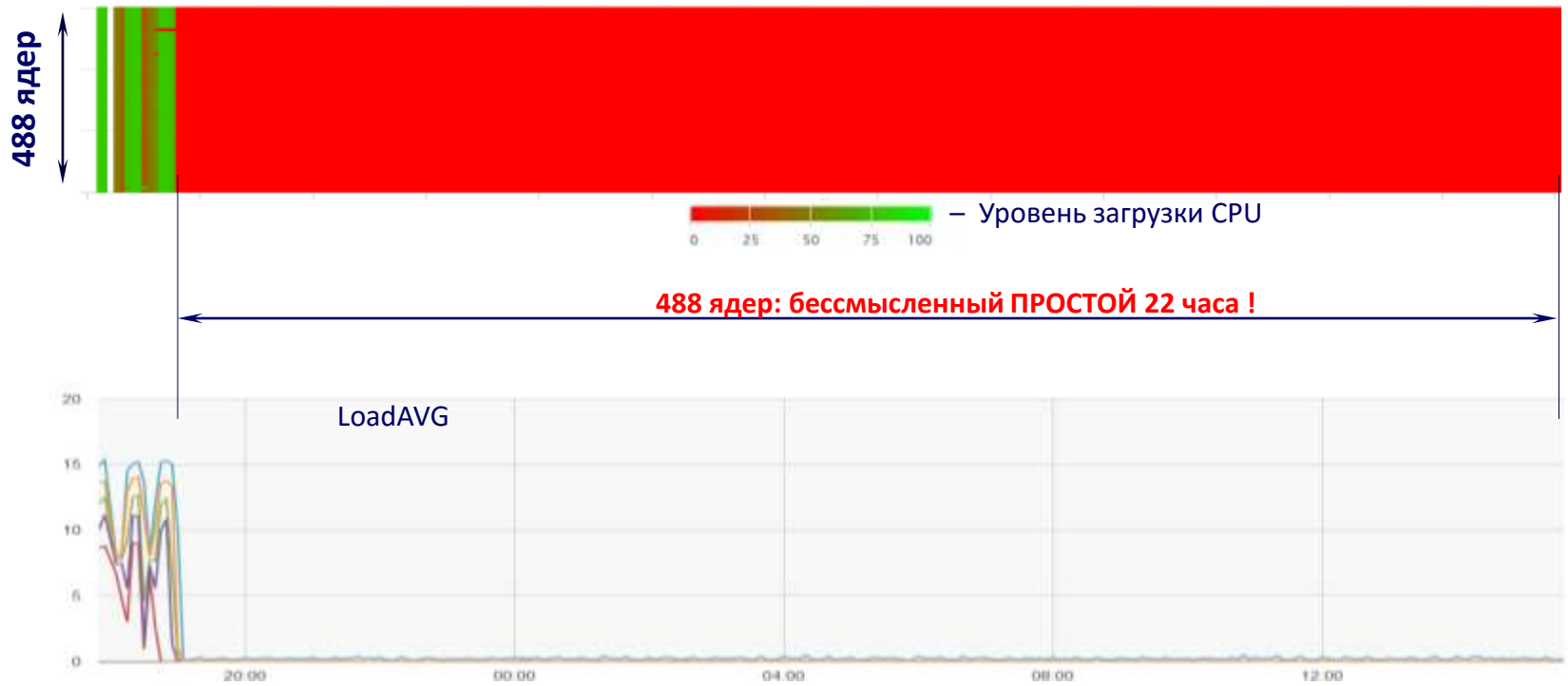
Алгоритмы и их реализации



Достаточно ли у нас информации об алгоритме А, чтобы создать эффективную реализацию для платформы С?

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

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)

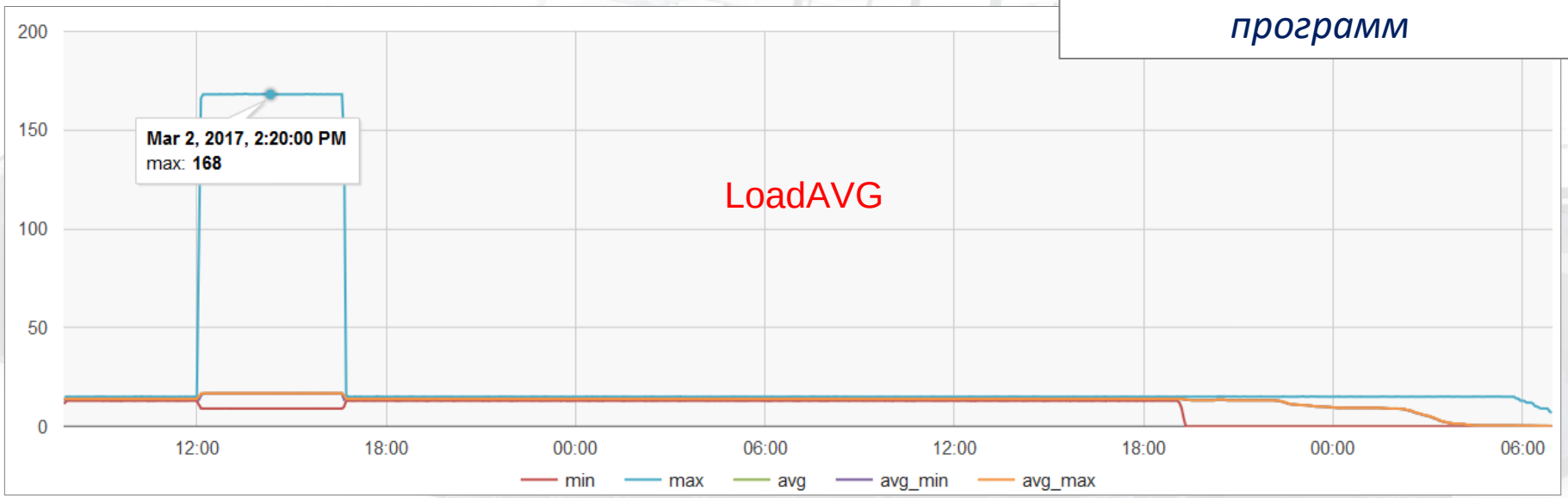
Идеи суперкомпьютерного кодизайна
(Supercomputing Co-Design)

Идеи суперкомпьютерного кодизайна
(Supercomputing Co-Design)

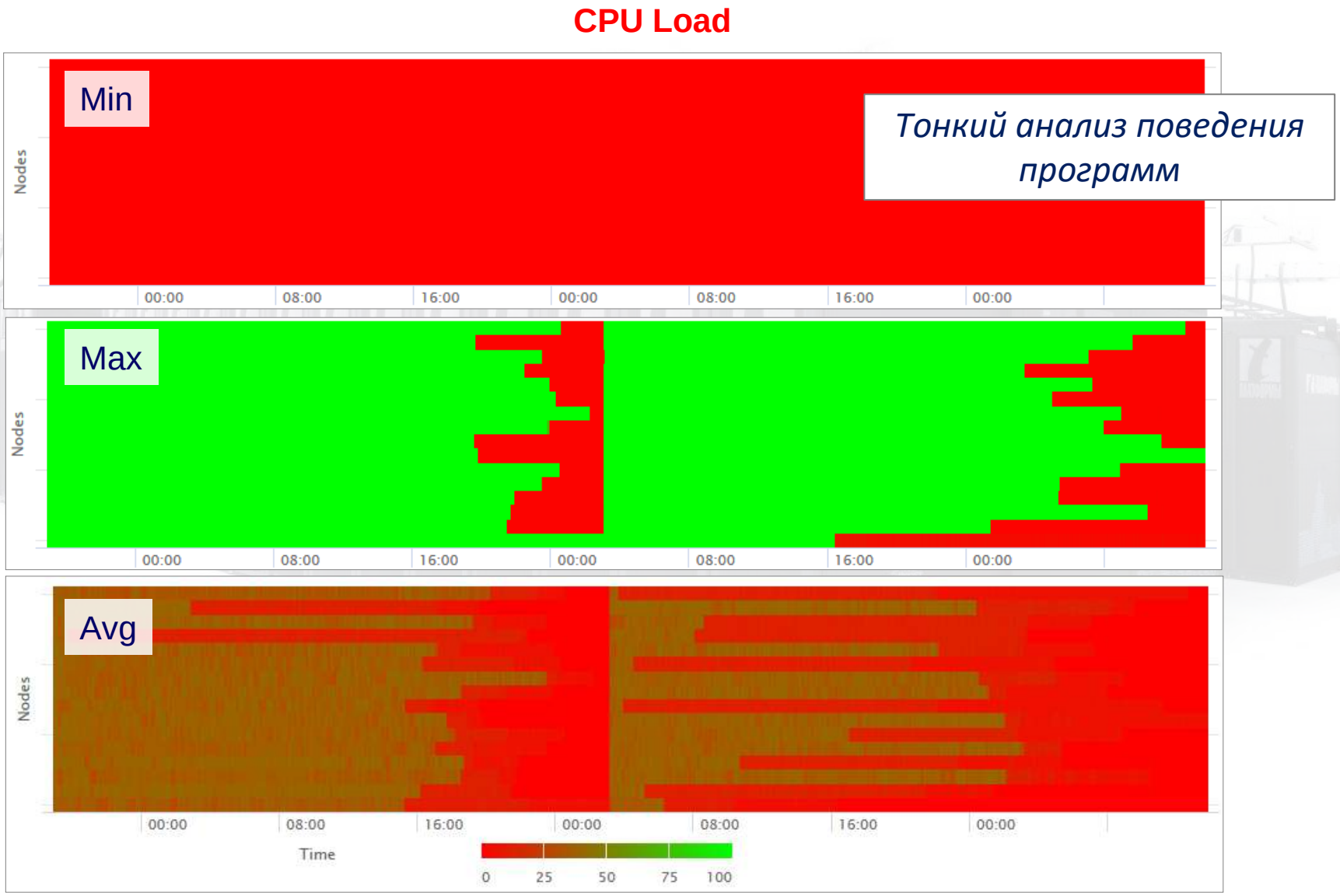
Как обеспечить эффективность цепочки суперкомпьютерного кодизайна?

Эффективность суперкомпьютерных приложений

Тонкий анализ поведения программ



Эффективность суперкомпьютерных приложений

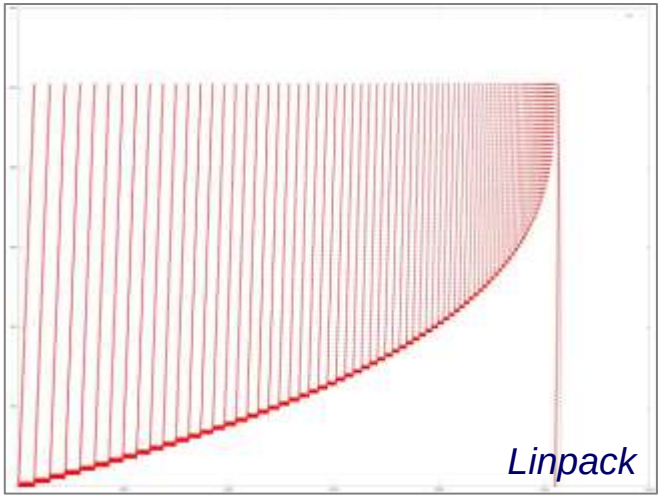


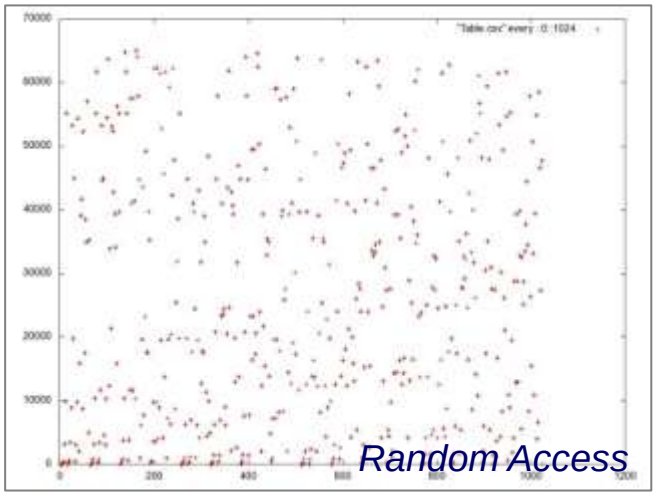
Тест HPCG для суперкомпьютеров
(еще один аргумент в пользу суперкомпьютерного кодизайна)

June 2017 HPCG Results

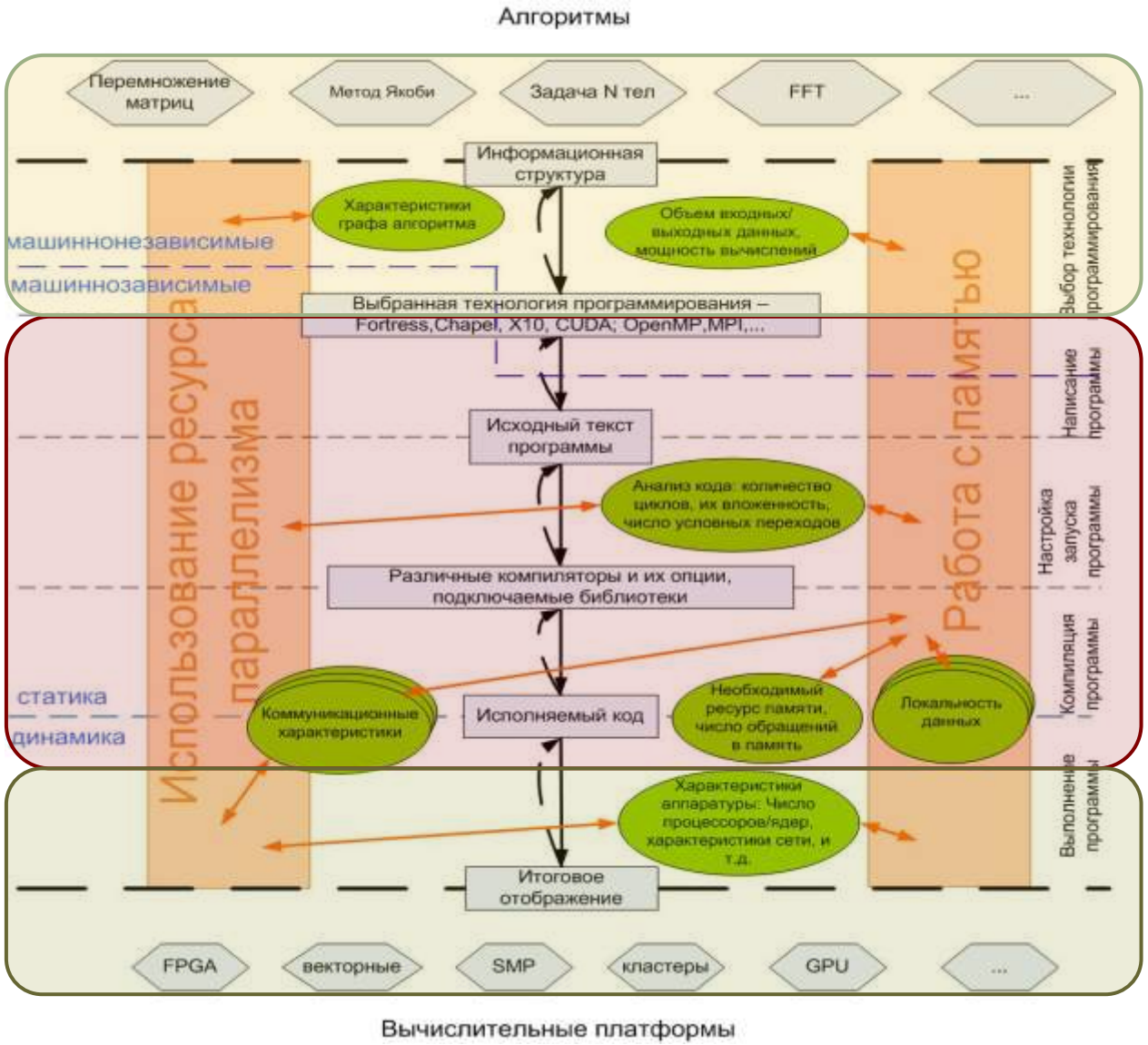
Rank	Site	Computer	Cores	HPL Rmax (Pflop/s)	TOP500 Rank	HPCG (Pflop/s)	Fraction of Peak
1	RIKEN Advanced Institute for Computational Science Japan	**K computer** – , SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10.510 94%	8	0.6027	5.3%
3	National Supercomputing Center in Wuxi China	**Sunway 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

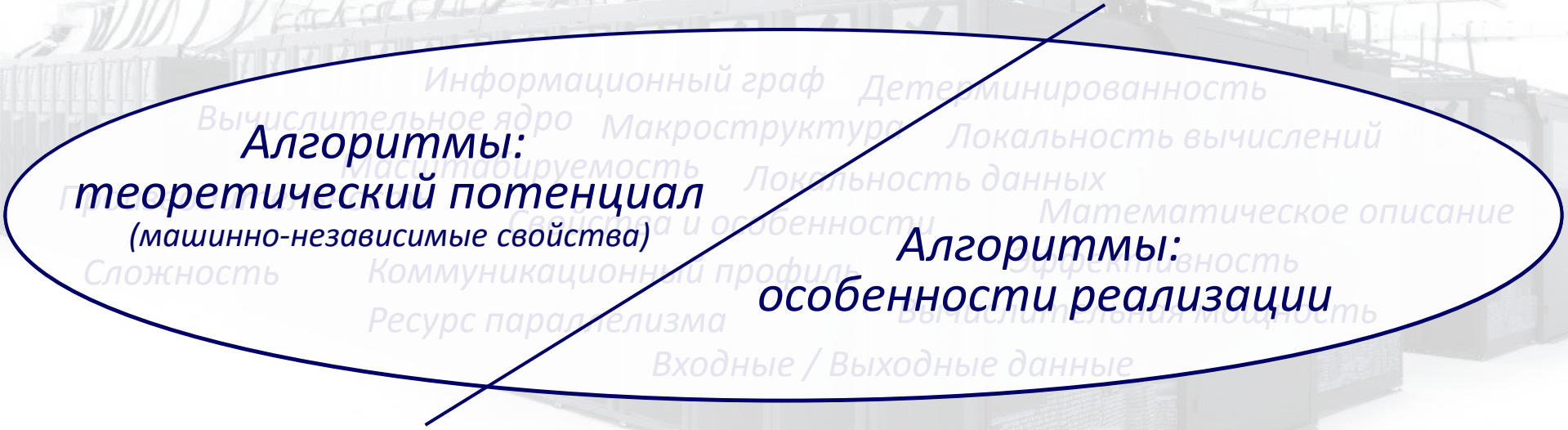
Да, можно: проект AlgoWiki

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)

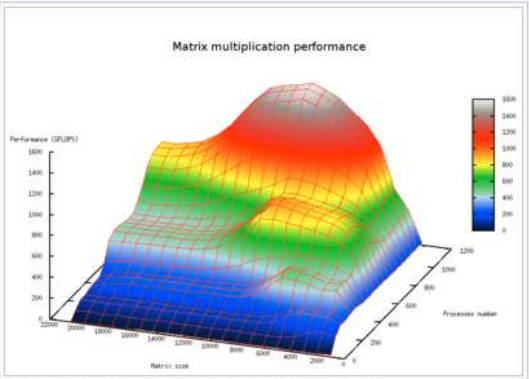
1 Свойства и структура алгоритма

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

Внимание, зачетное задание!

Исследование масштабируемости и эффективности реализаций алгоритмов на параллельных вычислительных системах

Полное описание задания, сроки выполнения, технология регистрации и распределения алгоритмов, выбор алгоритмов и платформ, форма отчета по заданию: все это будет представлено на http://Parallel.ru/SS17

Задание:

- *выбирается алгоритм,*
- *выбирается параллельная вычислительная система,*
- *берется параллельная программа, реализующая данный алгоритм,*
- *выполняется серия запусков программы на вычислительной системе для определения:*
  - *зависимости производительности от числа процессоров и размеров задачи,*
  - *нахождения числа процессоров и размеров задачи, при которых достигается максимальная производительность.*

Масштабируемость алгоритма

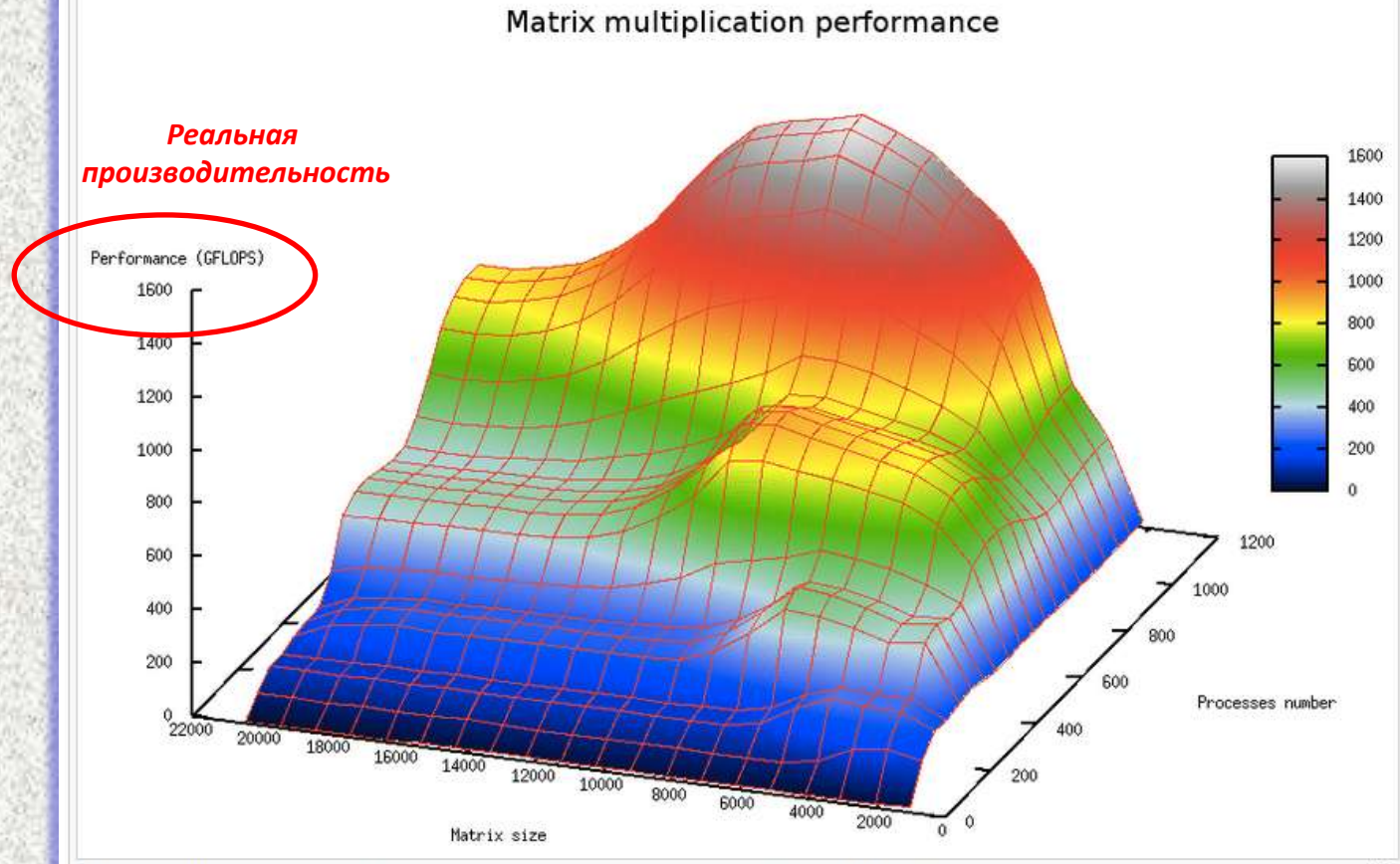


Рис.5 Масштабируемость классического алгоритма умножения плотных матриц в зависимости от числа процессоров и размера задачи

Важно: подобрать такие размеры задачи и числа процессоров, чтобы отразить на графике характерные точки, описывающие свойства алгоритма и программы.

Внимание, зачетное задание!

Исследование масштабируемости и эффективности реализаций алгоритмов на параллельных вычислительных системах

Полное описание задания, сроки выполнения, технология регистрации и распределения алгоритмов, выбор алгоритмов и платформ, форма отчета по заданию: все это будет представлено на http://Parallel.ru/SS17

Задание:

- *выбирается алгоритм,*
- *выбирается параллельная вычислительная система,*
- *берется параллельная программа, реализующая данный алгоритм,*
- *выполняется серия запусков программы на вычислительной системе для определения:*
  - *зависимости производительности от числа процессоров и размеров задачи,*
  - *нахождения числа процессоров и размеров задачи, при которых достигается максимальная производительность.*

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

СУПЕРКОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ

СТРУКТУРА АЛГОРИТМОВ

Вл.В.Воеводин

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

voevodin@parallel.ru

ВМК МГУ, 2017