Список вопросов к экзамену по курсу

«Суперкомпьютерное моделирование и технологии»,

магистры 2 года обучения, сентябрь – декабрь 2016 г.

1. Поколения архитектур компьютеров и парадигмы программирования. Архитектурные особенности современных микропроцессоров.
2. Технологии Интел для высокопроизводительных вычислений.
3. Стратегия развития процессоров архитектуры POWER.
4. Программно-аппаратная архитектура суперкомпьютеров Ломоносов и Blue Gene/P.
5. Последовательная и параллельная сложность алгоритмов, информационный граф и ресурс параллелизма алгоритмов.
6. Организация параллельных вычислений с использованием технологии передачи сообщений MPI. Основные группы функций MPI. Обработка ошибок в MPI.
7. Функции двухточечных передач данных в MPI. Способы организации неблокирующих передач.
8. Организация коллективных передач данных в MPI: назначение, основные функции.
9. Понятие о виртуальной топологии процессов в MPI. Функции MPI для работы с виртуальными топологиями. Использование виртуальных топологий для реализации сеточных задач.
10. Задача Дирихле для уравнения Пуассона в прямоугольнике, разностная аппроксимация задачи на прямоугольной неравномерной сетке.
11. Метод скорейшего спуска и метод сопряженных градиентов для разностной задачи Дирихле.
12. Одномерное и двумерное разбиение прямоугольной сетки на домены. Сравнение методов разбиения. Алгоритм определения размеров домена.
13. Суперкомпьютерное моделирование турбулентных течений.
14. Использование суперкомпьютеров для решения задач молекулярного моделирования.
15. Архитектурные особенности графических процессоров, направленные на массивно-параллельные вычисления. Особенности работы с памятью графического процессора.
16. Методы эффективной организации параллельных вычислений на графических процессорах.
17. Тензорные методы представления многомерных массивов.
18. Принцип интерференции в квантовой механике.

Литература

1. Презентации лекций
2. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. - СПб.: БХВ-Петербург, 2002. - 608 с.
3. Антонов А.С. Технологии параллельного программирования MPI и OpenMP: Учеб. пособие. - М.: Издательство Московского университета, 2012.-344 с.-(Серия "Суперкомпьютерное образование"). ISBN 978-5-211-06343
4. Головизнин В.М., Зайцев М.А., Карабасов С.А., Короткин И.А. Новые алгоритмы вычислительной гидродинамики для многопроцессорных вычислительных комплексов./М.: Издательство Московского университета, 2013, 472 с.
5. *Якобовский М.В.* Введение в параллельные методы решения задач: Учебное пособие . – М.: Издательство Московского университета, 2012. – 328 с.
6. А. В. Боресков и др. Параллельные вычисления на GPU. Архитектура и программная модель CUDA: Учебное пособие.-Издательство Московского университета, 2012, 336 стр.
7. Д. Сандерс, Э. Кэндрот Технология CUDA в примерах. Введение в программирование графических процессоров. 2015, 232 с.
8. *http://AlgoWiki-Project.org*

Билет 1
Поколения архитектур компьютеров и парадигмы программирования. Архитектурные особенности современных микропроцессоров.

Поколения архитектур компьютеров и парадигмы программирования

Середина 70-х годов: векторно-конвейерные компьютеры

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

Программирование: векторизация самых внутренних циклов.

Начало 80-х годов: векторно-параллельные компьютеры

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.

Начало 90-х годов: массивно-параллельные компьютеры

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.

Середина 90-х годов: параллельные компьютеры с общей памятью

Особенности архитектуры: сотни процессоров объединяются над общей памятью.

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

Начало 2000-х: кластеры из узлов с общей памятью

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.

Середина 2000-х: кластеры из узлов с общей памятью с ускорителями

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;

Понятия «поколения архитектур и парадигмы программирования» эквивалентны вопросу «как часто мы 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) …

Виден ли конец процессу переписывания программ?
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).

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

Описание алгоритмов:
-Математическое описание
-Сложность
-Информационный граф
-Свойства и особенности
-Ресурс параллелизма
-Масштабируемость
-Локальность данных
-Детерминированность
-Вычислительная мощность
-Входные / Выходные данные
-Вычислительное ядро
-Макроструктура
-Локальность вычислений

-Коммуникационный профиль
-Производительность
-Эффективность

Описание алгоритмов состоит из:
-теоретический потенциал (машинно-независимые свойства)
-особенности реализации

Информационная структура: (сложности описания алгоритмов)
- Как изобразить потенциально бесконечный граф ?
- Как изобразить потенциально многомерный граф ?
- Как показать зависимость структуры графа от размера задачи ?

Способы:
Минимальное остовное дерево (MST) (ресурс “математического” параллелизма)
Минимальное остовное дерево (ресурс классического параллелизма)

Архитектурные особенности современных микропроцессоров

Рассматриваются на основе изучения семейства МП Intel x86.

№	Особенность	В чем заключается	Где впервые появилась
1.	Многозадачность	Возможность работы в одном из двух режимов: реальном (real) и защищенном (protected). В *реальном* режиме возможно выполнение только одной программы. Адресация оперативной памяти без специальных драйверов ограничивается 1Мб. В *защищенном*(protected) режиме обеспечивается выполнение сразу нескольких программ за счет переключения между задачами («переключение контекста процессора»). Адресация основной памяти расширена до 4 ГБ (в последних МП – до 100 ГБ).	Intel 80286
2.	Поддержка системы виртуальных машин	Дальнейшее развитие принципа многозадачности, возможность моделирования в одном МП работу нескольких компьютеров, управляемых *разными* ОС.	Intel 80386

3.	Конвейерная обработка команд	Одновременное выполнение разных тактов последовательных команд в разных частях МП с непосредственной передачей результатов выполнения из одной части МП в другую. Позволяло достигнуть пятикратного увеличения производительности МП.	Intel 80286
4.	Кэширование данных	Использование высокоскоростного буфера для обмена данными между микропроцессорной памятью (регистрами МП) и основной памятью ЭВМ. В кэш-память заранее копируются те участки памяти, с которыми собирается работать МП. Управление процессом кэширования осуществляется кэш-контроллером и производится*параллельно* с работой центрального процессора. Современные ЭВМ имеют иерархически организованную кэш-память (до 3 уровней).	Intel i386SLC, Intel i486 Многоуровневое кэширование – Intel Pentium II
5.	Суперскалярная архитектура	Наличие в микропроцессоре более 1 конвейера для выполнения команд (параллелизм на уровне инструкций)	Intel Pentium
6.	Суперскалярная архитектура с поддержкой внеочередного исполнения команд («динамическое исполнение»)	Наличие в микропроцессоре более 1 конвейера для выполнения команд, а также специальных схем, позволяющих изменить изначальную последовательность выполнения команд (не нарушая смысла алгоритма) с целью параллельной загрузки всех конвейеров.	Intel Pentium Pro
7.	Гарвардская архитектура процессора	В кэш-памяти 1 уровня предусмотрено разделение команд и данных, которые хранятся отдельно друг от друга для повышения эффективности обработки	Intel Pentium Pro

8.	Расширенный набор инструкций	Новые команды, расширяющие базовый набор инструкций МП, для работы с мультимедийной информацией и одновременной однотипной обработки множественных данных.	Intel Pentium MMX, Intel Pentium III, Intel Pentium IV, Семейство Intel Core, Intel Core 2
9.	Гибридизация RISC и CISC архитектуры	Преобразование стандартных x86-инструкций в RISC-подобные команды фиксированной длины. Еще не выполненные команды записываются в кэш инструкций в том порядке, в котором они будут подаваться на исполняющие устройства (конвейеры) МП. В кэш-памяти может храниться до 12000 микрокоманд. Перевод инструкций формата x86 в микрокоманды ядра процессора происходит асинхронно с работой основных исполняющих устройств.	Intel Pentium IV
10.	Технология одновременной многопоточности	Эмуляция двух логических исполняющих устройств на одном физическом с целью более эффективно исполнять параллельно запущенные потоки команд (параллелизм на уровне потоков)	Intel Pentium IV Hyper-Threading
11.	Многоядерные процессоры	Объединение двух или более исполняющих устройств на одной ИС, действующих как единое устройство. Обычно имеют общий кэш и интерфейсную систему для связи с другими устройствами ЭВМ	Процессоры семейства Intel Core (Intel Core Duo, Intel Core 2 Duo, Pentium Dual Core, Intel core 2 Quad и др.)

12.	Технология автоматического увеличения тактовой частоты процессора	Для обеспечения дополнительной производительности и при условии соблюдения ограничений по мощности, температуре и току, процессор может автоматически «разгоняться», то есть увеличивать рабочую тактовую частоту всех своих ядер.	Процессоры Core i5, i7

Билет 2

Источник –https://www.dropbox.com/sh/pr6dnsq3yrptadp/AAAOL4MesSQVeeKDdFACT7axa?dl=0 (Lect_2_Semin.pdf)

У Intel есть 2 серии процессоров, предназначенных для высокопроизводительных вычислений

1. Intel Xeon Processor
2. Intel Xeon Phi Processor

В каждой серии происходит эволюция процессоров от поколения к поколению по разным стратегиям:

1. Для Intel Xeon Processor это –
   1.1 увеличение количества ядер в процессоре от поколения к поколению
   1.2 увеличение производительности ядра/потока с каждым поколением
   1.3 затраты энергии должны оставаться на месте или снижаться с новым поколением
   1.4 сбалансированная платформа (память, ввод/вывод, вычисление)

2. Для Intel Xeon Phi Processor это –
   2.1 максимизация вычисления на ватт
   2.2 максимизация совокупной производительности за счет подбора производительности на ядре/нити
   2.3 затраты энергии должны оставаться на месте или снижаться с новым поколением
   2.4 оптимизация для высокопараллельных задач

Среди последних больших процессоров

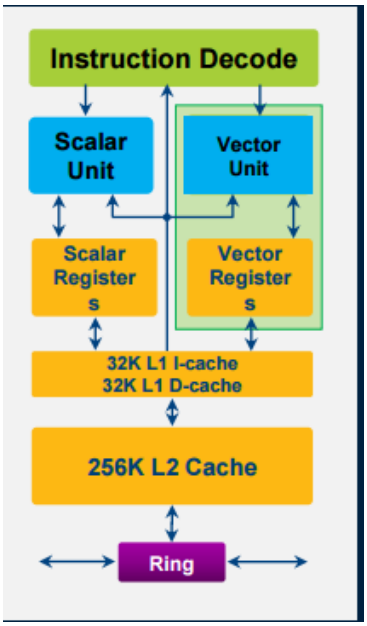
Intel® Xeon Phi™ coprocessor code-named Knights Corner (61 ядро, 244 нити, 512 битная архитектура)

А также

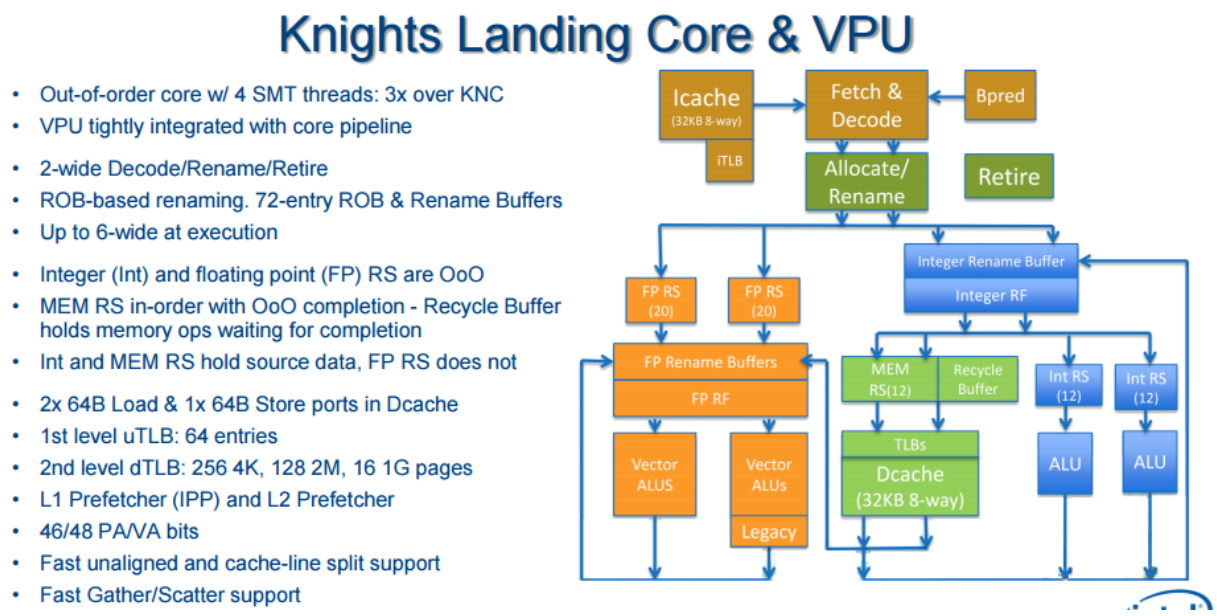
Intel ® Xeon Phi™ processor code-named Knights Landing (72, 288,512)

Intel Knights Corner состоит из скалярных и векторных ядер, с скалярными и векторными регистрами и совместным КЭШем 1-го и 2-го уровня.

Скалярные ядра – на базе Pentium, 64 битное машинное слово, векторные - 512-bit SIMD Instructions – not Intel® SSE, MMX™, or Intel® AVX



Архитектура Knight Landing еще сложнее, я пожалуй просто вставлю слайд:



Технология SIMD

SIMD-компьютеры состоят из одного командного процессора (управляющего модуля), называемого контроллером, и нескольких модулей обработки данных, называемых процессорными элементами. Управляющий модуль принимает, анализирует и выполняет команды. Если в команде встречаются данные, контроллер рассылает на все процессорные элементы команду, и эта команда выполняется на нескольких или на всех процессорных элементах. Каждый процессорный элемент имеет свою собственную память для хранения данных. Одним из преимуществ данной архитектуры считается то, что в этом случае более эффективно реализована логика вычислений. До половины логических инструкций обычного процессора связано с управлением выполнением машинных команд, а остальная их часть относится к работе с внутренней памятью процессора и выполнению арифметических операций. В SIMD-компьютере управление выполняется контроллером, а «арифметика» отдана процессорным элементам.

AVX

Advanced Vector Extensions (AVX) — расширение системы
команд x86 для микропроцессоров Intel и AMD, предложенное Intel в марте 2008.[1]

AVX предоставляет различные улучшения, новые инструкции и новую схему кодирования машинных кодов.

В презентации речь идет о **AVX-512**

AVX-512 расширяет AVX до векторов длиной 512 бит при помощи кодировки с префиксом EVEX. Расширение AVX-512 вводит 32 векторных регистра (ZMM), каждый по 512 бит, 8 регистров масок, 512-разрядные упакованные форматы для целых и дробных чисел и операции над ними, тонкое управление режимами округления (позволяет переопределить глобальные настройки), операции broadcast, подавление ошибок в операциях с дробными числами, операции gather/scatter, быстрые математические операции, компактное кодирование больших смещений. AVX-512 предлагает совместимость с AVX, в том смысле, что программа может использовать инструкции как AVX, так и AVX-512 без снижения производительности. Регистры AVX (YMM0-YMM15) отображаются на младшие части регистров AVX-512 (ZMM0-ZMM15), по аналогии с SSE и AVX регистрами

Особенностью AVX-512 является использование 8-ми масковых регистров.

Билет 3

Стратегия развития процессоров архитектуры POWER (Лекция HPC-HighPerformanceComputing)

Power Systems IBM

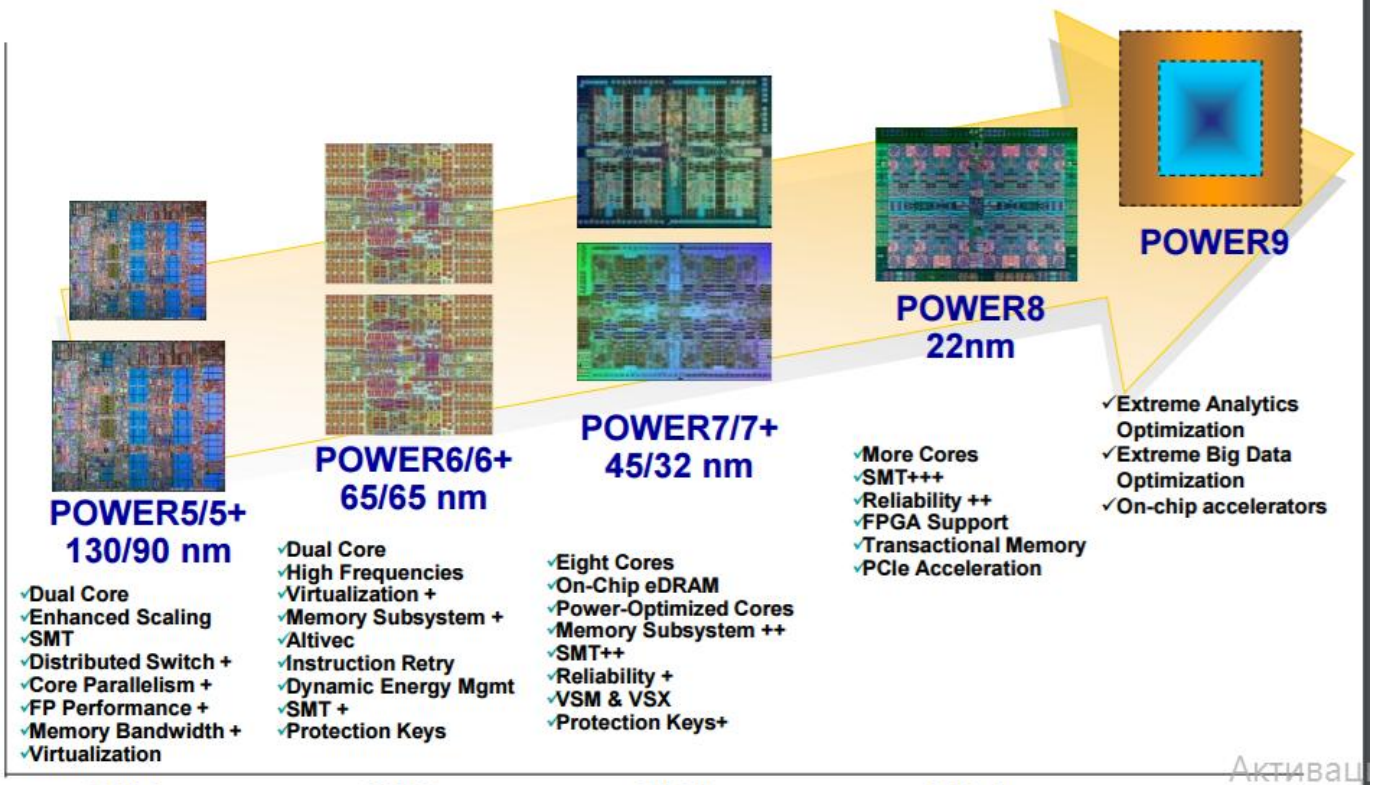
Стратегия развития процессоров архитектуры POWER

- Консолидация усилий и фокус на одном процессоре (чипе) общего назначения для каждого поколения
  - ❖Дизайн для более плотной интеграции с вспомогательным оборудованием
  - ❖Множественный дизайн модулей обеспечивает различные комбинации памяти и шин I/O
- Использование ускорителей подключаемых к процессору для соответствующих платформ и приложений
  - ❖FPGA для коммерческих задач, таких как Java, СУБД, аналитика
  - ❖GPU для научных и вычислительных задач

Power Systems

Планы развития процессора POWER

IBM



Процессор POWER8

Технология

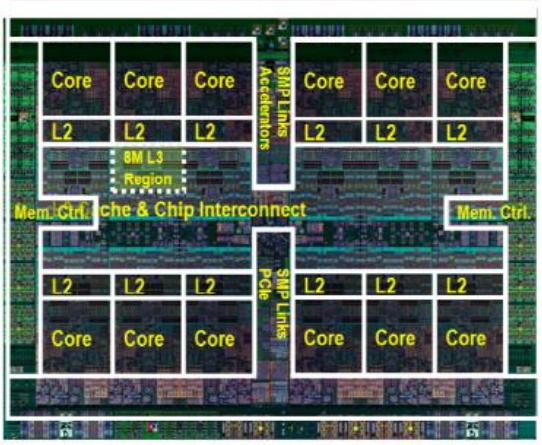
22nm SOI, eDRAM, 650mm2, 4.2B transistors

Ядра

- **12 ядер (SMT8)**
- 8 dispatch, 10 issue, 16 exec pipe
- **2X internal data flows/queues**
- Enhanced prefetching
- **64K кэш данных, 32K кэш инструкций**

Акселераторы

- **Криптография**
- **Расширение памяти**
- **Транзакционная память**
- **Поддержка VMM**
- **Перемещение данных / VM**



Energy Management

- On-chip Power Management Micro-controller
- Integrated Per-core VRM
- **Critical Path Monitors**

Увеличенные кэши

- 512 KB SRAM L2 / core
- **96 MB eDRAM shared L3**
- **Up to 128 MB eDRAM L4 (off-chip)**

Память

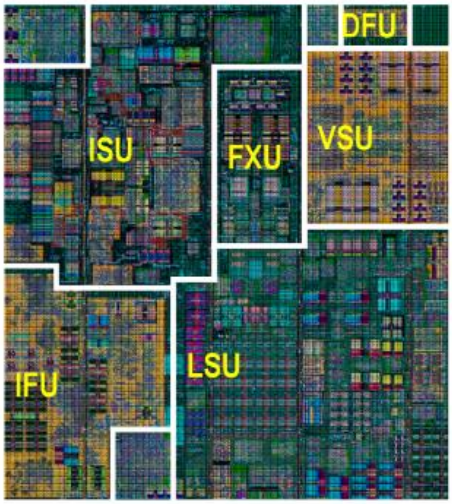
- **Up to 230 GB/s sustained bandwidth**

Шинные интерфейсы

- Durable open memory attach interface
- **Интегрированный PCIe G3**
- SMP Interconnect
- **CAPI (Coherent Accelerator Processor Interface)**

Ядро POWER8 (относительно POWER7)

- **SMT4 → SMT8**
- 8 dispatch
- 10 issue
- **16 execution pipes**: 2 FXU, 2 LSU, **2 LU**, 4 FPU, 2 VMX, 1 Crypto, 1 DFU, 1 CR, 1 BR
- Larger Issue queues (4 x 16-entry)
- Larger global completion, **Load/Store reorder**
- Improved branch prediction
- Improved unaligned storage access



- 2x L1 data cache (64 KB)
- 2x outstanding data cache misses
- 4x translation Cache

Wider Load/Store

- 32B → 64B L2 to L1 data bus
- 2x data cache to execution dataflow

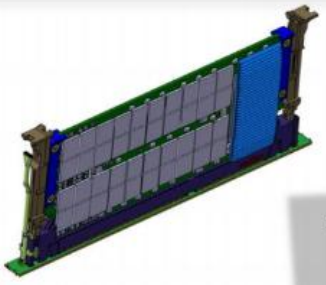
Enhanced Prefetch

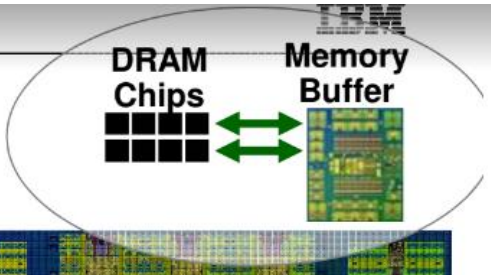
- **Instruction speculation awareness**
- Data prefetch depth awareness
- Adaptive bandwidth awareness
- Topology awareness

Power Systems

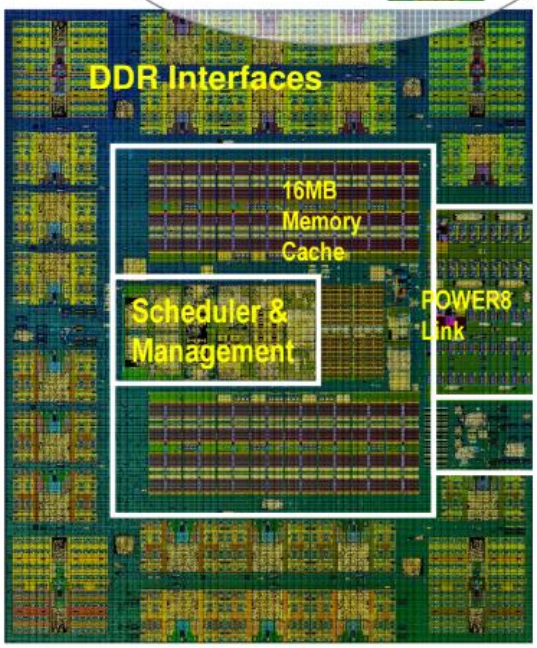
…with 16MB Cache…

IBM





“L4 cache”



Модули памяти наполняются интеллектом
- Умная система кэширования
- Оптимизация энергии
- Надежность

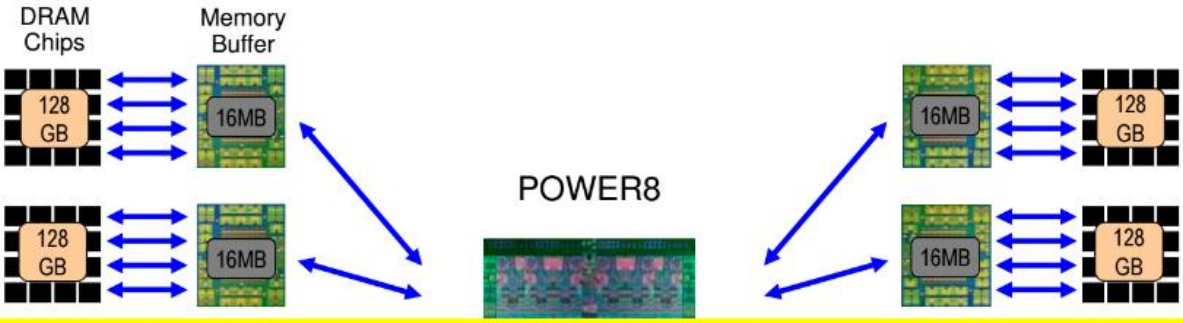
Оптимизированный интерфейс
- 9.6 GB/s high speed interface
- Интеллектуальная надежность
- Изоляция сбоев на лету

Уникальная производительность
- Уменьшенная латентность fastpath
- Cache → latency/bandwidth, partial updates
- Логика предсказания
- 22nm SOI for optimal performance / energy
- 15 metal levels (latency, bandwidth)

Power Systems

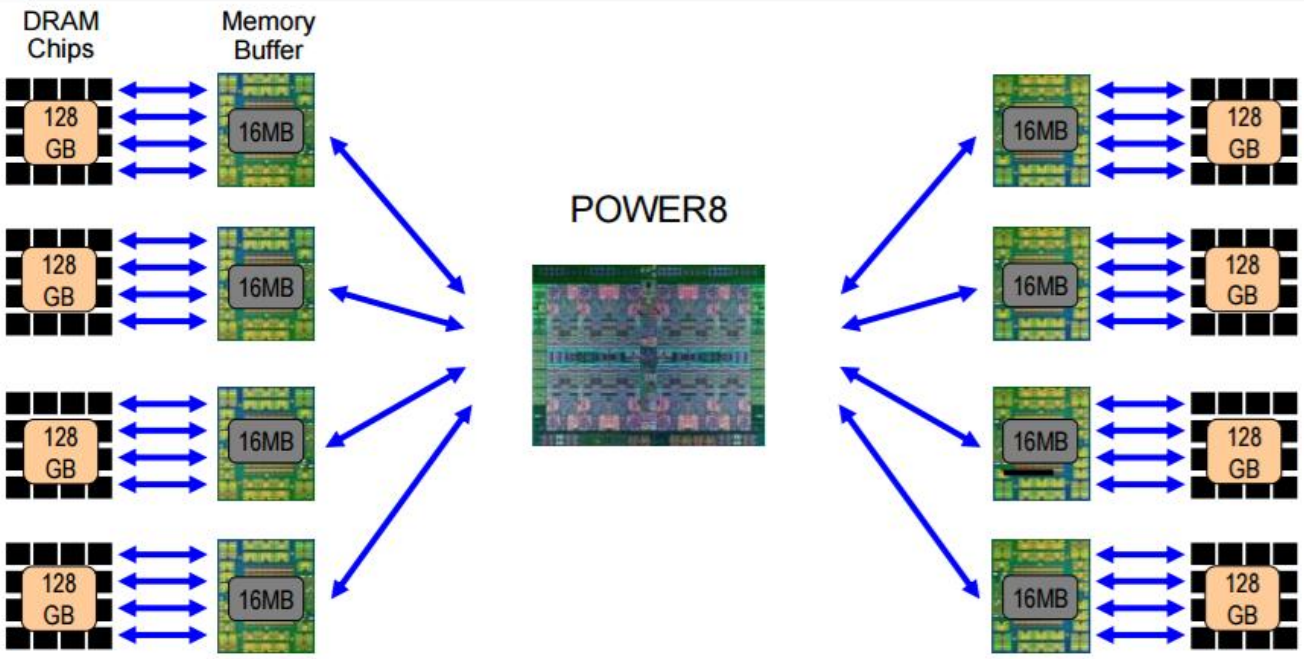
Организация памяти в POWER8

IBM



- У Intel нет L4 и они показывают цифры “to the DIMM”
- Наши 230 ГБ/с вполне достижимы в реальных условиях
- Цифры “to-DIMM” теоретические, реально достижимые намного ниже (из-за используемых протоколов DIMM, это справедливо для всех производителей)

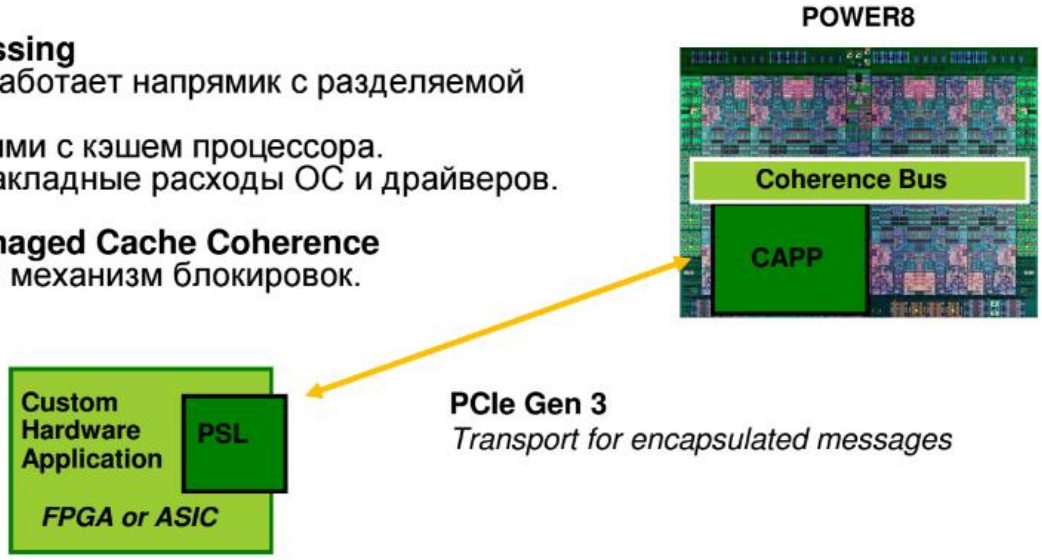
- ***8 скоростных каналов, каждый до 9.6 Гб/с до 230 ГБ/с в устойчивом режиме (sustained)***
- ***До 32 портов DDR выдающих в пике 410 ГБ/с (на уровне DRAM)***
- ***До 1 ТБ памяти на сокет (для старших версий – до 2 ТБ на сокет)***



- ***Up to 8 high speed channels, each running up to 9.6 Gb/s for up to 230 GB/s sustained***
- ***Up to 32 total DDR ports yielding 410 GB/s peak at the DRAM***
- ***Up to 1 TB memory capacity per fully configured processor socket***

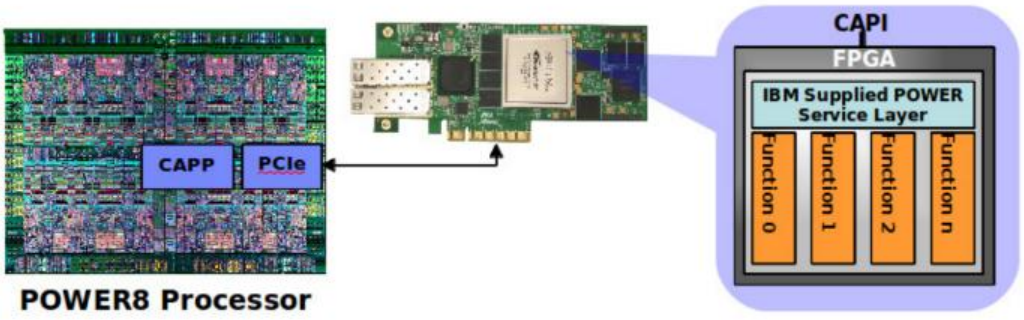
Virtual Addressing
•Ускоритель работает напрямик с разделяемой памятью
•Обмен данными с кэшем процессора.
•Исключает накладные расходы ОС и драйверов.

Hardware Managed Cache Coherence
•Стандартный механизм блокировок.



Специализированные контроллеры
Программные ускорители

Coherent Accelerator Processor Interface (CAPI) Flow

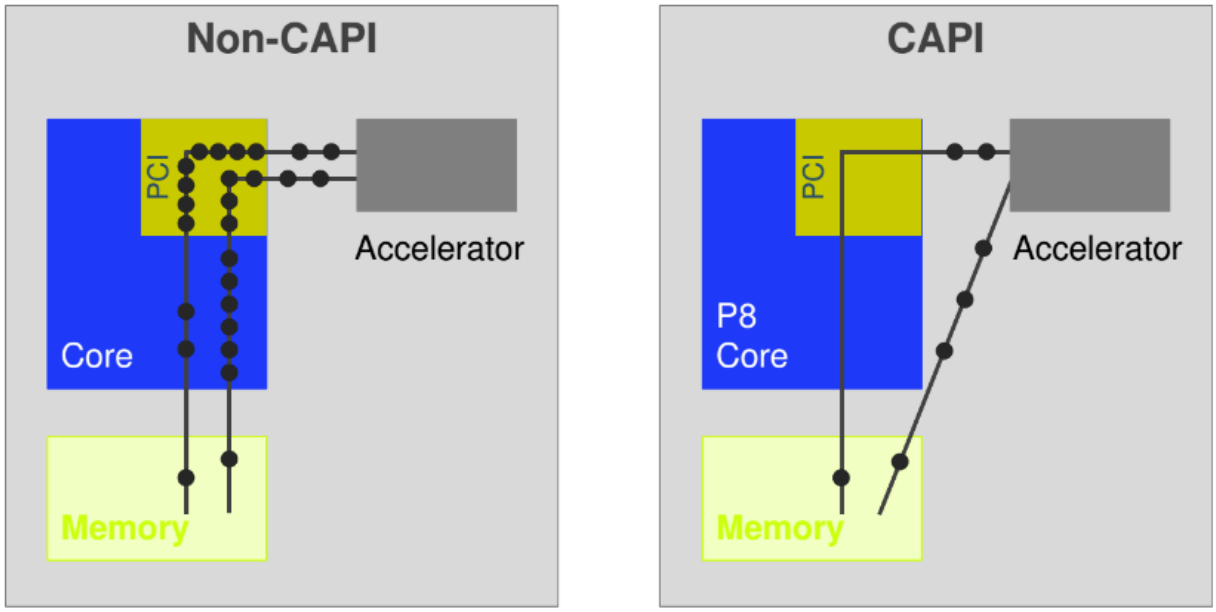


Типичный процесс работы I/O

Процесс при использовании когерентной памяти

Активация

Coherent Accelerator Processor Interface

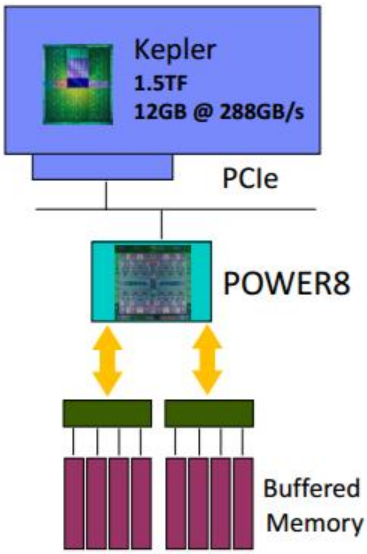


Power Systems

CPU и GPU будут связаны ещё плотнее

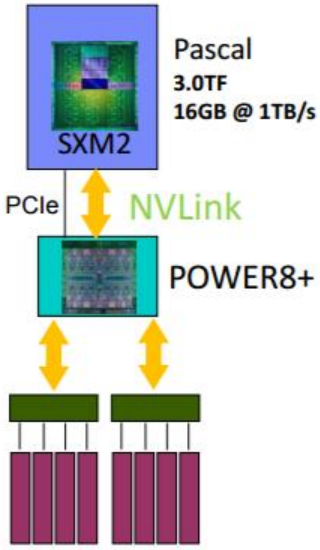
IBM

Kepler
CUDA 5.5 – 7.0
close



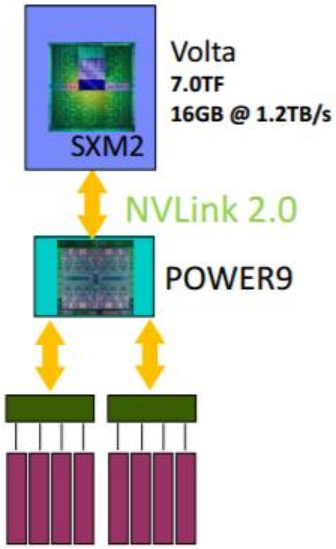
2014-2015

Pascal
CUDA 8
closer



2016

Volta
CUDA 9
Cache Coherent



2017

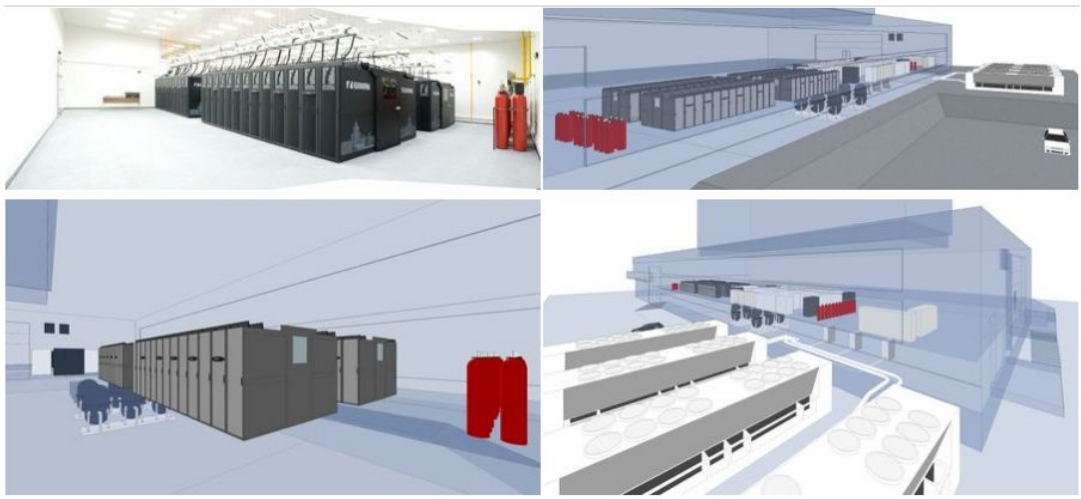
Программно-аппаратная архитектура Ломоносов и BlueGene\P

Суперкомпьютер «Ломоносов», 2015 год (из лекций)

Пиковая производительность	1700.21 TFlop/s
Производительность (Linpack)	901.90 TFlop/s
Эффективность	53%
Вычислительных узлов (Intel)	5 104
Вычислительных узлов (ГПУ)	1 065
Процессоры Intel Xeon 5570, 5670	12 346
NVIDIA Tesla X2070	2 130
Число процессорных ядер (x86)	52 168
Число процессорных ядер (ГПУ)	954 240
Оперативная память	92 ТБайт
Коммуникационная сеть	QDR Infiniband / 10 GE
Система хранения данных	1.75 ПБайт, Lustre, NFS, ...
Операционная система	Clusrtx T-Platforms Edition
Занимаемая площадь (вычислитель)	252 м2
Энергопотребление (вычислитель)	2.8 МВт

Суперкомпьютерный комплекс "Ломоносов"

Общий вид комплекса:



Суперкомпьютер "ЛОМОНОСОВ"

Суперкомпьютер «Ломоносов» — первый **гибридный суперкомпьютер** такого масштаба в России и Восточной Европе. В нём используется 3 вида вычислительных узлов и процессоры с различной архитектурой. В качестве основных узлов, обеспечивающих свыше 90 % производительности системы, используется blade-платформаT-Blade2. Предполагается использовать суперкомпьютер для решения ресурсоёмких вычислительных задач в рамках фундаментальных научных исследований, а также для проведения научной работы в области разработки алгоритмов и программного обеспечения для мощных вычислительных систем.

Последние несколько лет рост интереса к суперкомпьютерной тематике связан с выходом на рынок так называемых **гибридных суперкомпьютеров. То есть суперкомпьютеров, которые наряду с центральным процессором традиционной архитектуры используют для вычислений специализированные процессоры, в частности, графические.**

Общая характеристика

Основные технические характеристики суперкомпьютера "Ломоносов"	
Пиковая производительность	1,7 Пфлопс
Производительность на тесте Linpack	901.9 Тфлопс
Число вычислительных узлов x86	5 104
Число графических вычислительных узлов	1 065
Число вычислительных узлов PowerXCell	30
Число процессоров/ядер x86	12 346 / 52 168
Число графических ядер	954 240
Оперативная память	92 ТБ
Общий объем дисковой памяти вычислителя	1,75 ПБ
Основной тип процессора	Intel Xeon X5570/Intel Xeon 5670, Nvidia X2070
Число типов вычислительных узлов	8
Основной тип вычислительных узлов	ТВ2-XN
System/Servise/Management Network	QDR Infiniband 4x/10G Ethernet/Gigabit Ethernet

Система хранения данных	Параллельная файловая система Lustre, файловая система NFS, иерархическая файловая система StorNext, система резервного копирования и архивирования данных
Операционная система	Clustrx T-Platforms Edition
Занимаемая площадь	252 м^2
Потребление энергии	2,6 МВт
Вес всех составляющих	Более 75 тонн
Производитель	Т-Платформы(link is external)

Площади помещений:

- Вычислитель: 252 кв. м
- СБЭ (система бесперебойного электропитания): 246 кв.м.
- ГРЩ (главный распределительный щит): 85 кв. м.
- Климатическая система: 216 кв. м.

Энергопотребление:

- Пиковая мощность вычислителя (1,7 Tflops): 2,6 МВт
- Средняя мощность инфраструктуры: 740 КВт.
- Пиковая мощность инфраструктуры при внешней температуре 35 цельсия: 1,2 МВт
- Средняя суммарная мощность комплекса: 2,57 МВт
- Пиковая суммарная мощность комплекса (при 35 цельсия): 3,05 МВт.

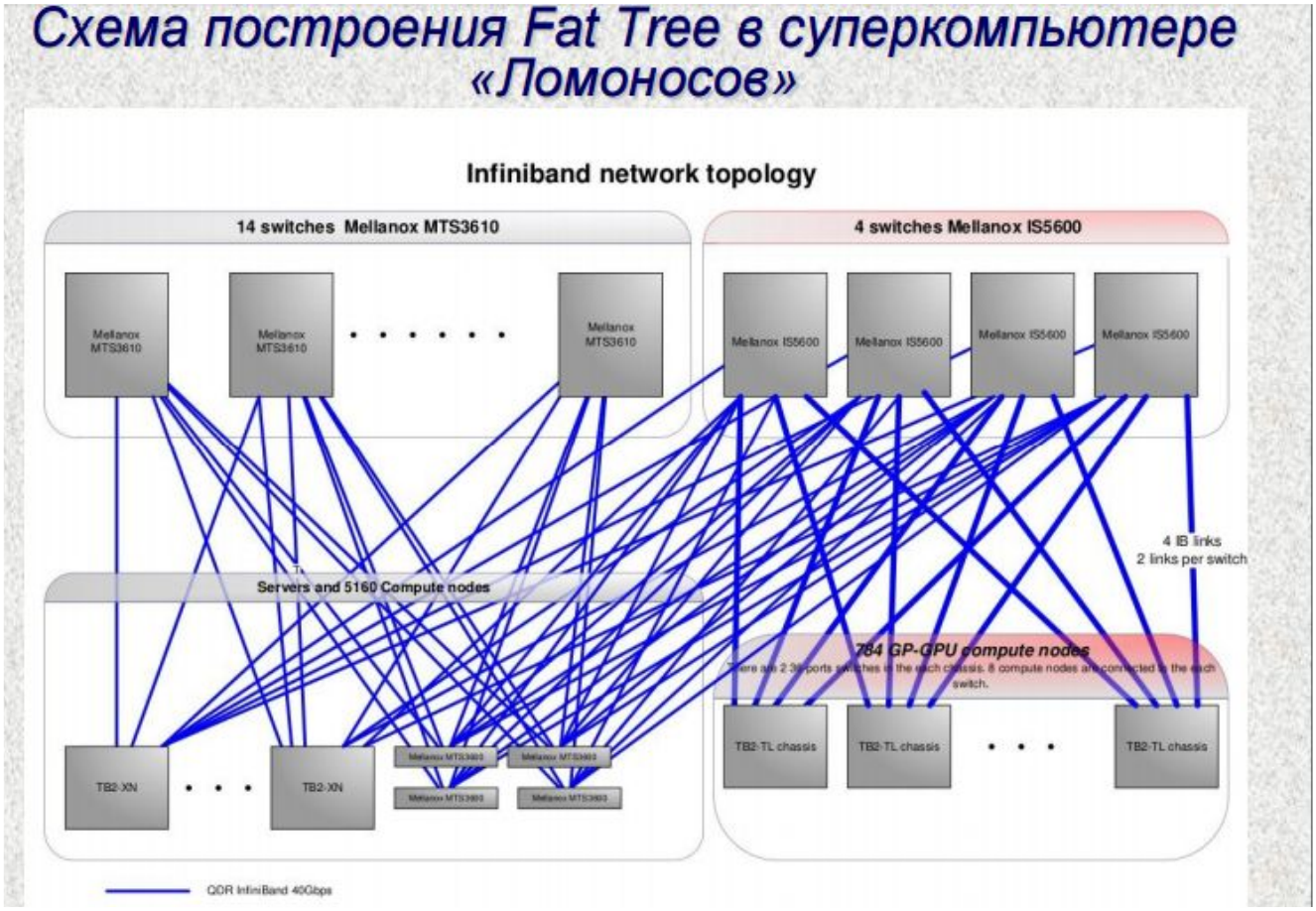
Вычислительные узлы и сети Группы вычислительных узлов:

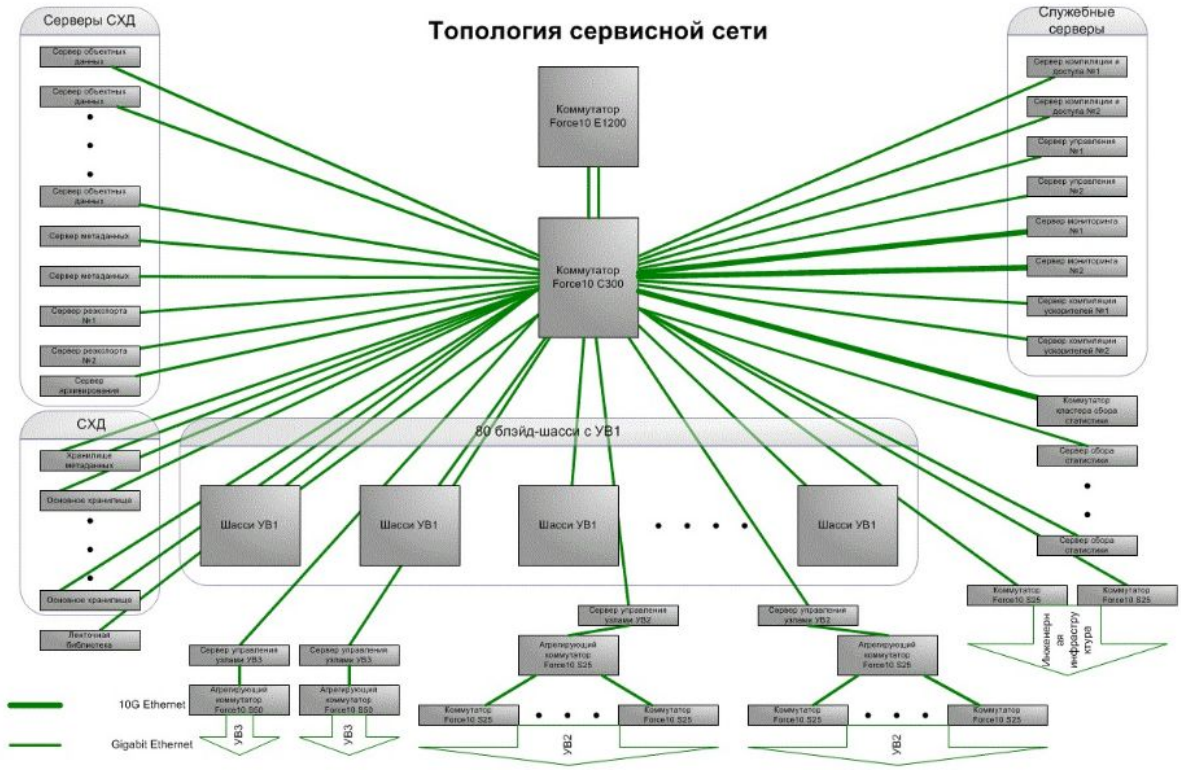
Тип	**Процессоры**	**Кол-во ядер**	**Опер. память, ГБ**	**Сумм. кол-во процес.**	**Сумм. кол-во ядер**	**Кол-во узлов**
T-Blade2(link is external)(УВ1)	2 x Intel® Xeon 5570 Nehalem	2 x 4	12	8 320	33 280	4 160
T-Blade1(УВ2)	2 x Intel® Xeon 5570 Nehalem	2 x 4	24	520	2 080	260
T-Blade2(link is external)(УВ1)	2 x Intel® Xeon 5670 Westmere	2 x 6	24	1 280	7 680	640
T-Blade1(УВ2)	2 x Intel® Xeon 5670 Westmere	2 x 6	48	80	480	40
Узлы на базе IBM® Cell (УВ3)	PowerXCell 8i	8	16	60	480	30

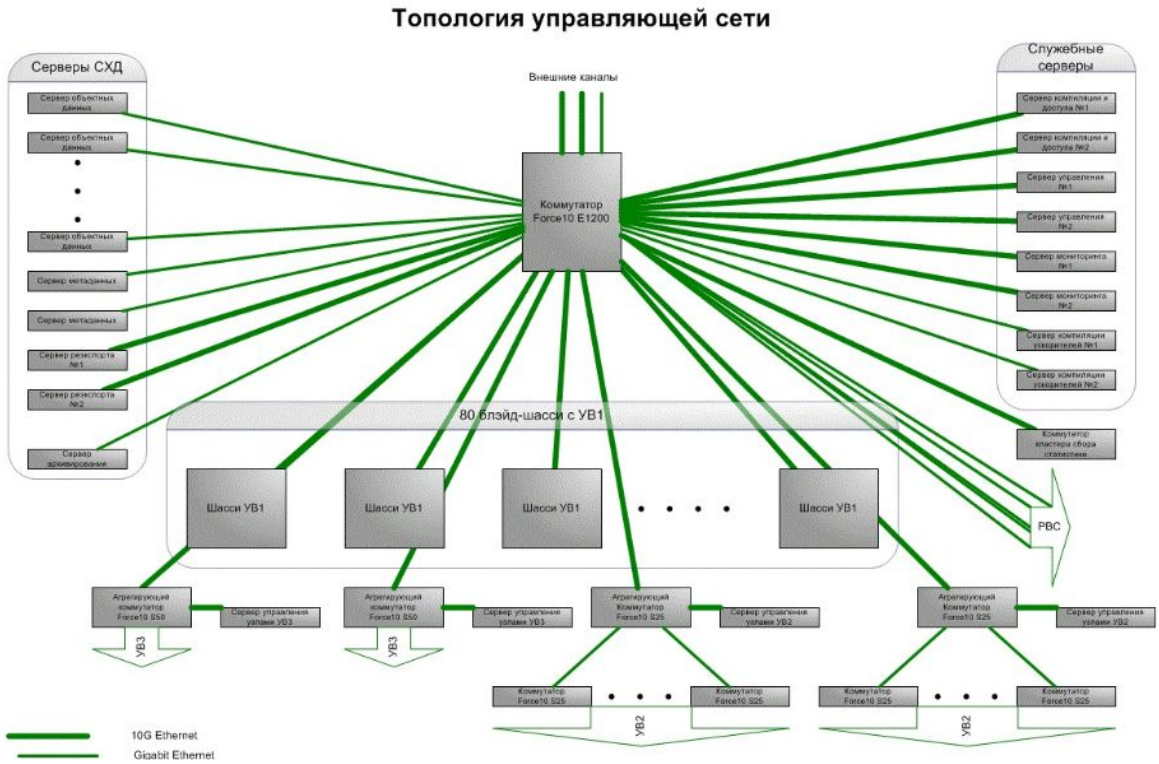
Все узлы связаны тремя независимыми сетями:

- Системная сеть - QDR InfiniBand, 40 Гбит/сек (схема)
- Сервисная сеть - Ethernet, 10 Гбит/сек, 1 Гбит/сек и 100 Мбит/сек (схема)
- Управляющая сеть - Ethernet, 10 Гбит/сек и 1 Гбит/сек (схема)
- Сеть барьерной синхронизации и сеть глобальных прерываний, Т-Платформы

Сеть **fat tree** (рус. *утолщенное дерево*) — топология компьютерной сети, изобретённая Чарльзом Лейзерсоном из MIT, является дешевой и эффективной для суперкомпьютеров[1]. В отличие от классической топологии дерево, в которой все связи между узлами одинаковы, связи в утолщенном дереве становятся более широкими (толстыми, производительными по пропускной способности) с каждым уровнем по мере приближения к корню дерева. Часто используют удвоение пропускной способности на каждом уровне.







Программное обеспечение

- Средства архивации данных: bacula 3 (Т-Платформы), StorNext (Quantum), NetBackup (Symantec)
- Передача файлов: SCP, SFTP
- Управление заданиями и ресурсами: SLURM 2.0
- Среды исполнения: OpenMPI 1.4, MVAPICH 1.1, IntelMPI 4

- Языки программирования: C/C++, Fortran 77/90/95
- Наборы компиляторов: Intel 12, GNU 4.4, Pathscale, PGI
- Средства отладки и анализа производительности: Intel® ITAC 12, grpof 4, Intel® vTune 4, Intel® Thread Checker, Acumem ThreadSpotter, IDB, Allinea DDT
- Системы контроля версий: SVN, GIT
- Языки сценариев: Perl, Python

HPC-приложения MOLPRO, версия 2010.1 (установлено в /opt/molpro2010.1/) - доступен ТОЛЬКО сотрудникам МГУ.

Система хранения данных

Что снижает производительность компьютеров с распределенной памятью?

1. Закон Амдала
2. Латентность передачи по сети
3. Пропускная способность каналов передачи данных
4. Особенности использования SMP-узлов
5. Балансировка вычислительной нагрузки
6. Возможность асинхронного счета и передачи данных
7. Особенности топологии коммуникационной сети
8. Производительность отдельных процессоров
9. ...

http://users.parallel.ru/wiki/pages/22-config

Конфигурация суперкомпьютеров

Ломоносов-2

Раздел	узлов/x86 ядер (x86 ядер на узел)	GPU-карт на узле	память	диски	max ядро-часов	max/default время (часов на задачу)	max задач	max запущенных задач	max ядер на задачу, не более
compute .	1024 / 14336 (14)	1	64 ГБ (4,5 ГБ/ядро)	нет	нет	72/24	3	3	нет

- Объём памяти на GPU: 11.56 GB
- Модель GPU: Tesla K40s
- Модель CPU: Intel Xeon E5-2697 v3 2.60GHz

Ломоносов

Раздел	узлов/x86 ядер (x86 ядер на узел)	GPU-карт на узле	память	диски	max ядро-часов	max/default время (часов на задачу)	max задач	max запущенных задач	max ядер на задачу, не более
regular4 .	4096 / 32768 (8)	0	12 ГБ (1,5 ГБ/ядро)	нет	нет	72/24	3	3	1024
regular6 .	596 / 7152 (12)	0	12 ГБ (1 ГБ/ядро)	нет	нет	72/24	3	3	512
hdd4	260 / 2080 (8)	0	12 ГБ (1,5 ГБ/ядро)	есть	нет	72/24	2	2	256
hdd6	32 / 384 (12)	0	48 ГБ (4 ГБ/ядро)	есть	нет	72/24	2	2	128
gpu	830 / 6640 (8)	2	24 ГБ (3 ГБ/ядро)	нет	нет	72/24	3	3	256
smp	1 / 128 (128)	0	2ТБ (16 ГБ/ядро)	есть	нет	72/24	3	2	128
test	64 / 512 (8)	0	12 ГБ (1,5 ГБ/ядро)	нет	нет	0,25 (15 минут)/0,25	3	1	128
gputest	16 / 128 (8)	2	24 ГБ (3 ГБ/ядро)	нет	64	0,25 (15 минут)/0,25	3	1	64

Очередь по умолчанию - **regular4** “max задач” включает в себя и задачи на счёте и в очереди.

- Объём памяти на GPU: 5.25 GB
- Модель GPU: Tesla X2070
- Модель CPU:
  - regular4, hdd4: Intel Xeon X5570 2.93GHz
  - gpu: Intel Xeon E5630 2.53GHz
  - regular6, hdd6: Intel Xeon X5670 2.93GHz

Описание вычислительного комплекса IBM Blue Gene/P

IBM Blue Gene/P — массивно-параллельная вычислительная система, которая состоит из двух стоек, включающих **8192 процессорных ядер** (2 x 1024 четырехъядерных вычислительных узлов), **с пиковой производительностью 27,9 терафлопс** (27,8528 триллионов операций с плавающей точкой в секунду).

Массивно-параллельные системы с распределенной памятью
• Высокая плотность упаковки
– процессоры с низким энергопотреблением (40 W ~лампочка)
• Высокопроизводительный интерконект
– несколько комутационных подсистем для различных целей
• Ультра легкая ОС
– выполнение вычислений и ничего лишнего
• Стандартное ПО Standard software
– Fortran/C/C++ и MPI

- 2048 4-ех ядерных узлов
- пиковая производительность 27.2 Tflop/s
- Реальная производительность по тесту Linpack: 23.2 Тфлоп/с
- 85% от пиковой
- общий объем ОЗУ 4 ТВ

Характеристики системы:

- две стойки с вычислительными узлами и узлами ввода-вывода
- 1024 четырехъядерных вычислительных узла в каждой из стоек
- 16 узлов ввода-вывода в стойке (в текущей конфигурации активны 8, т.е. одна I/O-карта на 128 вычислительных узлов)
- выделенные коммуникационные сети для межпроцессорных обменов и глобальных операций
- программирование с использованием MPI, OpenMP/pthreads, POSIX I/O
- высокая энергоэффективность: **~ 372 MFlops/W** (см. список Green500)
- система воздушного охлаждения

1 стойка (rack, cabinet) состоит из двух midplane'ов.
В midplane входит 16 node-карт (compute node card),
на каждой из которых установлено 32 вычислительных узла (compute card).
Midplane, 8 x 8 x 8 = 512 вычислительных узлов,
— минимальный раздел, на котором становится доступна топология трехмерного тора;
для разделов меньших размеров используется топология трехмерной решетки.
Node-карта может содержать до двух узлов ввода-вывода (I/O card).
Вычислительный узел включает в себя четырехъядерный процессор, 2 ГБ общей памяти и сетевые интерфейсы.

- 1024 четырехъядерных вычислительных узлов
- производительность одного вычислительного узла – 13.6 GF/s
- производительность 1 стойки– 13.9 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)

Микропроцессорное ядро:

- модель: PowerPC 450
- рабочая частота: 850 MHz
- адресация: 32-битная
- кэш инструкций 1-го уровня (L1 instruction): 32 KB
- кэш данных 1-го уровня (L1 data): 32 KB
- кэш предвыборки (L2 prefetch): 14 потоков предварительной выборки (stream prefetching): 14 x 256 байтов
- два блока 64-битной арифметики с плавающей точкой (Floating Point Unit, FPU), каждый из которых может выдавать за один такт результат совмещенной операции умножения-сложения (Fused Multiply-Add, FMA)
- пиковая производительность: 2 FPU x 2 FMA x 850 MHz = 3,4 GFlop/sec per core

Вычислительные узлы и I/O-карты в аппаратном смысле неразличимы и являются взаимозаменяемыми, разница между ними состоит лишь в способе их использования. У них нет локальной файловой системы, поэтому все операции ввода-вывода перенаправляются внешним устройствам.

Вычислительной узел:

- четыре микропроцессорных ядра PowerPC 450 (4-way SMP)
- пиковая производительность: 4 cores x 3,4 GFlop/sec per core = 13,6 GFlop/sec
- пропускная способность памяти: 13,6 GB/sec
- 2 ГБ общей памяти
- 2 x 4 МБ кэш-памяти 2-го уровня (в документации по BG/P носит название L3)
- легковесное ядро (compute node kernel, CNK), представляющее собой Linux-подобную операционную систему, поддерживающую значительное подмножество Linux-совместимых системных вызовов
    - Создание процессов и управление ими
    - Управление памятью
    - Отладка процессов
    - Ввод-вывод
- асинхронные операции межпроцессорных обменов (выполняются параллельно с вычислениями)
- операции ввода-вывода перенаправляются I/O-картам через сеть коллективных операций
- Двойное устройство для работы с вещественными числами с плавающей точкой (double precision)
- Объем виртуальной памяти равен объему физической

Узел ввода-вывода:

- не учитывается при расчете пиковой производительности
- использует сеть коллективных операций для коммуникаций с вычислительными узлами
- подключен к внешним устройствам через Ethernet-порт посредством 10-гигабитный функциональной сети
- операционная система на основе Linux (Mini-Control Program, MCP) с минимальным набором пакетов, необходимых для поддержки клиента сетевой файловой системы и Ethernet-подключений

Коммуникационные сети:

- **трехмерный тор (three-dimensional torus)**
  - сеть общего назначения, объединяющие все вычислительные узлы; предназначена для операций типа «точка-точка»
  - вычислительный узел имеет двунаправленные связи с шестью соседями
  - пропускная способность каждого соединения — 425 MB/s (5,1 GB/s для всех 12 каналов)
  - латентность (ближайший сосед):
    - 32-байтный пакет: 0,1 μs
    - 256-байтный пакет: 0,8 μs
- **глобальные коллективные операции (global collective)**
  - коммуникации типа «один-ко-многим» (broadcast-операции и редукция)
  - используется вычислительными узлами для обменов с I/O-картами
  - каждый вычислительный узел и I/O-карта имеют три двунаправленные связи
  - пропускная способность каждого соединения — 850 MB/s (1,7 GB/s для двух каналов)
  - латентность (полный обход): 3,0 μs
- **глобальные прерывания (global interrupt)**
  - операции барьеров и прерываний (глобальные AND- и OR-операции)
- **функциональная сеть**
  - соединяет узлы ввода-вывода с внешним окружением
  - 10-гигабитная оптическая Ethernet-сеть
- **сервисная сеть (service/control)**
  - загрузка, мониторинг, диагностика, отладка, доступ к счетчикам производительности
  - гигабитная Ethernet-сеть (4 соединения на стойку)

Чтобы разгрузить процессорное ядро от операций, связанных с передачей сообщений по сети трехмерного тора, используется устройство прямого доступа к памяти (direct memory access, DMA). Кроме уменьшения нагрузки на ядро, этот механизм уменьшает вероятность взаимной блокировки процессов, обменивающихся сообщениями, которая может возникнуть вследствие ошибок программиста.

Окружение Blue Gene/P включает

- **фронтэнд (front end node)** — система, открытая для доступа по протоколу SSH; служит для доступа пользователей на вычислительный комплекс; вся связь с комплексом осуществляется только через эту машину; предназначена для разработки пользователями программ, компилирования проектов и постановки задач в очередь; работа с ней осуществляется в интерактивном режиме
- **сервисный узел (service node)** — обеспечивает контроль над системой Blue Gene/P; к этой машине доступа по SSH нет

- **систему управления высокопроизводительной параллельной файловой системой IBM General Parallel File System (GPFS)**

Назначение	**Модель**	**Процессоры**	**Количество**
фронтэнд	IBM pSeries 55A	POWER5+	2
сервисный узел			2
GPFS-серверы	IBM x3650	Intel Xeon	16

Для коммутации оптических линий служит высокопроизводительный свитч IBM Ethernet Switch B08R со 112 портами 10 Gb Ethernet: 64 порта используются для подключения узлов ввода-вывода вычислительной системы Blue Gene/P, 32 порта служат для подключения GPFS-серверов, управляющих параллельной файловой системой, к четырем портам подключены фронтэнды и сервисные узлы, остальные 12 портов используются для инфраструктурных нужд, либо зарезервированы для будущего использования. Гигабитный Ethernet скоммутирован на четыре 48-портовых свитча Cisco.

Для организации файлового хранилища используется 16 дисковых систем DS3512, каждая из которых включает по две дополнительных дисковых полки EXP3512. В основу сети хранения данных положены два 80-портовых коммутатора IBM System Storage SAN80B-4 Fibre Channel.

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

5. Последовательная и параллельная сложность алгоритмов, информационный граф и ресурс параллелизма алгоритма.

Algowiki

Последовательная сложность алгоритма - число операций, которые нужно выполнить при его последовательном исполнении.

Параллельная сложность - число шагов, за которое можно выполнить данный алгоритм в предположении доступности неограниченного числа необходимых процессоров (других устройств, вычислительных узлов, ядер и т.п.)

Парал. сложность алгоритма понимается как высота канонической ярусно-параллельной формы.

Информационной граф - ориентированный ациклический мультиграф, вершины которого соответствуют операциям алгоритма, а дуги - передачи данных между ними.

Вершины графа ал. могут соединяться несколькими дугами, в частности когда в качестве разных аргументов одной операции используется одна и та же величина.

Граф алгоритма почти всегда является параметризированным графом. В частности, его вид часто зависит от входных данных.

Граф алгоритма используется как удобное представление алгоритма при исследовании его структуры, ресурса параллелизма, а также других свойств. Его можно рассматривать как параметризован. информационную историю.

Ресурс параллелизма алгоритма. Включает:

- Оценку параллельной сложности алгоритма
- Структура параллелизма (иерархичность)
- Ключевые параллельные ветви в терминах конечного и массового параллелизма.

Конечный парал-зм - парал-м, определенной информационной независимостью некоторых фрагментов в тексте программы.

Массовый парал-м - парал-м, определенной информационной независимостью итераций циклов программы.

Такие бывают

Сконечной пар-м - пар-м в пр-ве итераций, определенной поверхностями уровней разверток.

Координатный пар-м - ~~параллелизм~~ частный случай сконечного пар-ма, опред. циклами ParDo, при кот. информационно независимые вершины лежат на гиперплоскостях, перпенд. одной из коорд. осей.

Билет 6

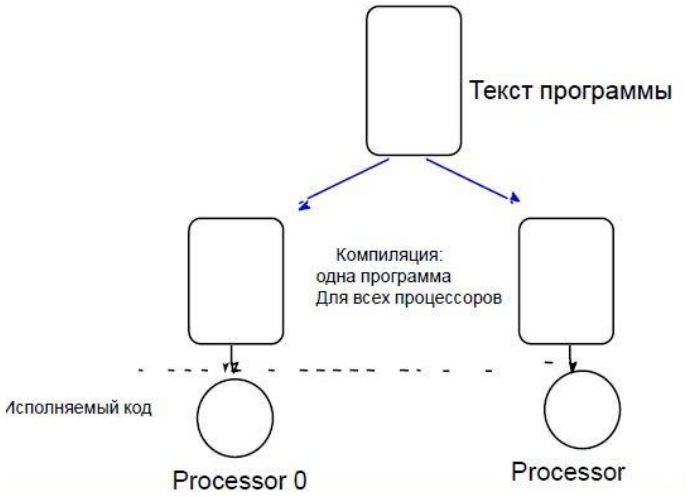
Организация параллельных вычислений с использованием технологии передачи сообщений MPI. Основные группы функций MPI. Обработка ошибок в MPI.

MPI
- MPI 1.1 Standard разрабатывался 92-94
- MPI 2.0 - 95-97
- MPI 2.1 - 2008
- MPI 3.1 – 2015

Реализации MPI
- MPICH
- LAM/MPI
- Mvapich
- OpenMPI
- 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.

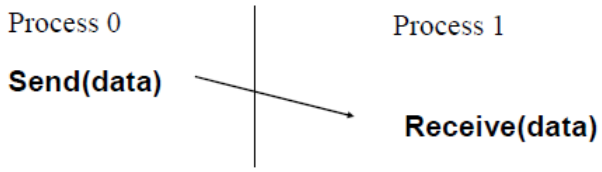
Модель MPI-программ
SPMD – Single Program Multiple Data
•Одна и та же программа выполняется различными процессорами
•Управляющими операторами выбираются различные части программы на каждом процессоре.



Модель выполнения MPI- программы
- Запуск: ***mpirun***
- При запуске указываем число требуемых процессоров **np** и название программы: пример: ***mpirun – np 3 prog***
- Каждый процесс MPI-программы получает два значения:
  - ***np –*** *число процессов*
  - ***rank*** из диапазона [0 … ***np-1***] – номер процесса
- Любые два процесса могут непосредственно обмениваться данными с помощью функций передачи сообщений

Основы передачи данных в MPI

- Технология передачи данных MPI предполагает кооперативный обмен.
- Данные посылаются одним процессом и принимаются другим.
- Передача и синхронизация совмещены.



Основные группы функций MPI

1. Определение среды
2. Передачи «точка-точка»
3. Коллективные операции
4. Производные типы данных
5. Группы процессов
6. Виртуальные топологии
7. Односторонние передачи
8. Параллельный ввод-вывод

Основные понятия

- Процессы объединяются в ***группы***.
- Каждое сообщение посылается в рамках некоторого ***контекста*** и должно быть получено в том же контексте.
- Группа и контекст вместе определяют ***коммуникатор***.
- Процесс идентифицируется своим ***номером*** в группе, ассоциированной с коммуникатором.

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

- Другие предопределенные коммуникаторы:

- **MPI_COMM_SELF** – только один (собственный) процесс
- **MPI_COMM_NULL** – пустой коммуникатор

Типы данных MPI

- Данные в сообщении описываются тройкой: (***address, count, datatype***), где
- ***datatype*** определяется рекурсивно как :

➢ предопределенный базовый тип, соответствующий типу данных в базовом языке (например, MPI_INT, MPI_DOUBLE_PRECISION)
➢ Непрерывный массив MPI типов
➢ Векторный тип
➢ Индексированый тип
➢ Произвольные структуры

- MPI включает функции для построения пользовательских типов данных, например, типа данных, описывающих пары (int, float).

Базовые MPI-типы данных

MPI datatype	**C datatype**
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int

MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double

Специальные типы MPI

- MPI_Comm
- MPI_Status
- MPI_datatype

Понятие тэга

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**.

MPI helloworld.c

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char **argv){
MPI_Init(&argc, &argv);
printf("Hello, MPI world\n");
MPI_Finalize();
return 0; }
```

Функции определения среды

- ***int MPI_Init(int *argc, char ***argv)*** – инициализация MPI. Должна быть 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

Обработка ошибок MPI-функций.

Определяется константой ***MPI_SUCCESS***
I

```
int error;
……
error = MPI_Init(&argc, &argv));
If (error != MPI_SUCCESS)
{
fprintf (stderr, " MPI_Init error \n");
return 1;
}
```

MPI_Comm_size - Количество процессов в коммуникаторе (Размер коммуникатора)
int MPI_Comm_size(MPI_Comm comm, int *size)
Результат – число процессов

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-отправителя и получателя

int MPI_Comm_rank(MPI_Comm comm, int *rank)

Результат – номер процесса

Завершение MPI-процессов

(Никаких вызовов MPI функций после)

int MPI_Finalize()

int MPI_Abort (MPI_Comm_size(MPI_Comm comm, int*errorcode)

Трансляция MPI-программ

- Трансляция

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

Например:

mpicc –o hw helloworld.c

- Запуск в интерактивном режиме

mpirun –np 128 hw

Билет 7

Взаимодействие «точка-точка»

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

Варианты передачи «точка-точка»

- Синхронные пересылки
- Асинронные передачи
- Блокирующие передачи
- Неблокирующие передачи

Функции MPI передачи «точка-точка»

Point-to-Point Communication Routines		
MPI_Bsend	MPI_Bsend_init	MPI_Buffer_attach
MPI_Buffer_detach	MPI_Cancel	MPI_Get_count
MPI_Get_elements	MPI_Ibsend	MPI_Iprobe
MPI_Irecv	MPI_Irsend	MPI_Isend
MPI_Issend	MPI_Probe	MPI_Recv
MPI_Recv_init	MPI_Request_free	MPI_Rsend
MPI_Rsend_init	MPI_Send	MPI_Send_init
MPI_Sendrecv	MPI_Sendrecv_replace	MPI_Ssend
MPI_Ssend_init	MPI_Start	MPI_Startall
MPI_Test	MPI_Test_cancelled	MPI_Testall
MPI_Testany	MPI_Testsome	MPI_Wait
MPI_Waitall	MPI_Waitany	MPI_Waitsome

Блокирующие и неблокирующие передачи

- Определяют, при каких условиях операции передачи завершаются:

- **Блокирующие**: возврат из функций передачи сообщений только по завершению передачи
- **Неблокирующие** : немедленный возврат из функций, пользователь должен контролировать завершение передач

Основа 2-точечных обменов

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

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

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_Send(data,500,MPI_FLOAT,6,33,MPI_COMM_WORLD)

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***

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
- **count** – число реально полученных элементов

C:
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

Пример

```
if (rank == 0) {
// Send size of integers to process 1
MPI_Send(buf, size, MPI_INT, 1, 0,
MPI_COMM_WORLD);
printf("0 sent %d numbers to 1\n", size);
            } else if (rank == 1) {
               MPI_Status status;
//       Probe for an incoming message from process
MPI_Probe (0, 0, MPI_COMM_WORLD,
&status);
MPI_Get_count (&status, MPI_INT,
&size);
```

```
int* number_buf = (int*)malloc(sizeof(int) * size);
//    Now receive the message with the allocated buffer
MPI_Recv (number_buf, size, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
printf("1 dynamically received %d numbers from 0.\n",
number_amount); free(number_buf);
}
```

Совмещение передач типа «отсылка-прием»

```
int MPI_Sendrecv (void *sendbuf,
int sendcount, MPI_Datatype sendtype, int dest, int sendtag,
void *rcvbuf, int rcvcount, MPI_Datatype rcvtype, int source, int rcvtag,
MPI_Comm comm,
MPI_Status *status)
```

Обмен данными одного типа с замещением

```
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)
```

Неблокирующие коммуникации

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

Неблокирующие операции завершаются, не дожидаясь окончания передачи данных.

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)

Проверка завершения операций **MPI_Wait()** and **MPI_Test().**

MPI_Wait() ожидание завершения.

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

Замер времени MPI_Wtime

- Время замеряется в секундах
- Выделяется интервал в программе

double MPI_Wtime(void);

Пример.

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

Прием/передача сообщений без блокировки

int MPI_Isend(void *buf, int count, MPI_Datatype datatype, int dest, int msgtag, MPI_Comm comm, MPI_Request *request)

buf - адрес начала буфера посылки сообщения

count - число передаваемых элементов в сообщении

datatype - тип передаваемых элементов

dest - номер процесса-получателя

msgtag - идентификатор сообщения

comm - идентификатор группы

OUT request - идентификатор асинхронной передачи

Передача сообщения, аналогичная MPI_Send, однако возврат из подпрограммы происходит сразу после инициализации процесса передачи без ожидания обработки всего сообщения, находящегося в буфере buf. Это означает, что нельзя повторно использовать данный буфер для других целей без получения дополнительной информации о завершении данной посылки. Окончание процесса передачи (т.е. того момента, когда можно переиспользовать буфер buf без опасения испортить передаваемое сообщение) можно определить с помощью параметра request и процедур MPI_Wait и MPI_Test.

Сообщение, отправленное любой из процедур MPI_Send и MPI_Isend, может быть принято любой из процедур MPI_Recv и MPI_Irecv.

int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source, int msgtag, MPI_Comm comm, MPI_Request *request)

OUT buf - адрес начала буфера приема сообщения

count - максимальное число элементов в принимаемом сообщении

datatype - тип элементов принимаемого сообщения

source - номер процесса-отправителя

msgtag - идентификатор принимаемого сообщения

comm - идентификатор группы

OUT request - идентификатор асинхронного приема сообщения

Прием сообщения, аналогичный MPI_Recv, однако возврат из подпрограммы происходит сразу после инициализации процесса приема без ожидания получения сообщения в буфере buf. Окончание процесса приема можно определить с помощью параметра request и процедур MPI_Wait и MPI_Test.

int MPI_Wait(MPI_Request *request, MPI_Status *status)

request - идентификатор асинхронного приема или передачи

OUT status - параметры сообщения

Ожидание завершения асинхронных процедур MPI_Isend или MPI_Irecv, ассоциированных с идентификатором request. В случае приема, атрибуты и длину полученного сообщения можно определить обычным образом с помощью параметра status.

int MPI_Waitall(int count, MPI_Request *requests, MPI_Status *statuses)

count - число идентификаторов

requests - массив идентификаторов асинхронного приема или передачи

OUT statuses - параметры сообщений

Выполнение процесса блокируется до тех пор, пока все операции обмена, ассоциированные с указанными идентификаторами, не будут завершены. Если во время одной или нескольких операций обмена возникли ошибки, то поле ошибки в элементах массива statuses будет установлено в соответствующее значение.

Билет 8

Организация коллективных передач данных в MPI: назначение, основные функции.

Функции коллективных передач

Collective Communication Routines		
MPI_Allgather	MPI_Allgatherv	MPI_Allreduce
MPI_Alltoall	MPI_Alltoallv	MPI_Barrier
MPI_Bcast	MPI_Gather	MPI_Gatherv
MPI_Op_create	MPI_Op_free	MPI_Reduce
MPI_Reduce_scatter	MPI_Scan	MPI_Scatter
MPI_Scatterv		

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)
- Нет неблокирующих коллективных операций
- Нет тэгов
- Принимающий буфер должен точно соответствовать размеру отсылаемого буфера

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 Scatter:
- One-to-all communication: различные данные 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 имеют смысл только для процесса root

Глобальные операции редукции:

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

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

где:

-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

Функция **MPI_Gather** производит сборку блоков данных, посылаемых всеми процессами группы, в один массив процесса с номером root. Длина блоков предполагается одинаковой. Объединение происходит в порядке увеличения номеров процессов- отправителей. То есть 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)

IN	sendbuf	-	адрес начала размещения посылаемых данных;
IN	sendcount	-	число посылаемых элементов;
IN	sendtype	-	тип посылаемых элементов;
OUT	recvbuf	-	адрес начала буфера приема (используется только в процессе-получателе root);

IN recvcount - число элементов, получаемых от каждого процесса (используется только в процессе-получателе root);
IN recvtype - тип получаемых элементов;
IN root - номер процесса-получателя;
IN comm - коммуникатор.

Функция MPI_Alltoall совмещает в себе операции Scatter и Gather и является по сути дела расширением операции Allgather, когда каждый процесс посылает различные данные разным получателям. Процесс i посылает j-ый блок своего буфера sendbuf процессу j, который помещает его в i-ый блок своего буфера recvbuf. Количество посланных данных должно быть равно количеству полученных данных для каждой пары процессов.

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

IN sendbuf - адрес начала буфера посылки;
IN sendcount - число посылаемых элементов;
IN sendtype - тип посылаемых элементов;
OUT recvbuf - адрес начала буфера приема;
IN recvcount - число элементов, получаемых от каждого процесса;
IN recvtype - тип получаемых элементов;
IN comm - коммуникатор.

Функция **MPI_OP_CREATE** связывает определенную пользователем глобальную операцию с указателем op, который впоследствии может быть использован в MPI_REDUCE, MPI_ALLREDUCE,

MPI_SCATTER и MPI_SCAN. Определенная пользователем операция предполагается ассоциативной. Если commute = true, то операция должна быть как коммутативной, так и ассоциативной. Если commute = false, то порядок операндов фиксирован, операнды располагаются по возрастанию номеров процессов, начиная с нулевого. Порядок оценки может быть изменен, чтобы использовать преимущество ассоциативности операции. Если commute = true, то порядок оценки может быть изменен, чтобы использовать достоинства коммутативности и ассоциативности.

MPI_OP_CREATE(function, commute, op)

Function-определяемая пользователем операция (функция) commute-true, если операция коммутативна; иначе false.
Op-операция (дескриптор)

Функция **MPI_Reduce_scatter** совмещает в себе операции редукции и распределения результата по процессам.

MPI_Reduce_scatter(void* sendbuf, void* recvbuf, int *recvcounts, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

IN sendbuf - адрес начала входного буфера;
OUT recvbuf - адрес начала буфера приема;
IN revcount - массив, в котором задаются размеры блоков, посылаемых процессам;
IN datatype - тип элементов во входном буфере;
IN op - операция, по которой выполняется редукция;
IN comm - коммуникатор.

Функция **MPI_Scaterv** является векторным вариантом функции MPI_Scatter, позволяющим посылать каждому процессу различное количество элементов. Начало расположения элементов блока, посылаемого i-му процессу, задается в массиве смещений displs, а число посылаемых элементов - в массиве sendcounts. Эта функция является обратной по отношению к функции MPI_Gatherv.

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

IN sendbuf - адрес начала буфера посылки (используется только в процессе-отправителе root);
IN sendcounts - целочисленный массив (размер равен числу процессов в группе), содержащий число элементов, посылаемых каждому процессу;
IN displs - целочисленный массив (размер равен числу процессов в группе), i-ое значение определяет смещение относительно начала sendbuf для данных, посылаемых процессу i;
IN sendtype - тип посылаемых элементов;
OUT recvbuf - адрес начала буфера приема;
IN recvcount - число получаемых элементов;
IN recvtype - тип получаемых элементов;
IN root - номер процесса-отправителя;
IN comm - коммуникатор.

Функция **MPI_Allgatherv** является аналогом функции MPI_Gatherv, но сборка выполняется всеми процессами группы. Поэтому в списке параметров отсутствует параметр root.

```
int MPI_Allgatherv(void* sendbuf, int sendcount, MPI_Datatype sendtype, void*
rbuf, int *recvcounts, int *displs,
MPI_Datatype recvtype, MPI_Comm comm)
```

IN sendbuf - адрес начала буфера передачи;
IN sendcount - число посылаемых элементов;
IN sendtype - тип посылаемых элементов;
OUT rbuf - адрес начала буфера приема;
IN recvcounts - целочисленный массив (размер равен числу процессов в группе), содержащий число элементов, которое должно быть получено от каждого процесса;
IN displs - целочисленный массив (размер равен числу процессов в группе), i-ое значение определяет смещение относительно начала rbuf i-го блока данных;
IN recvtype - тип получаемых элементов;
IN comm - коммуникатор.

MPI_ALLTOALLV обладает большей гибкостью, чем функция MPI_ALLTOALL, поскольку размещение данных на передающей стороне определяется аргументом sdispls, а на стороне приема - независимым аргументом rdispls.

Функция **MPI_OP_FREE** маркирует определенную пользователем операцию редукции для удаления и устанавливает значение MPI_OP_NULL для аргумента op.

MPI_OP_FREE(op)

Функция **MPI_SCAN** используется, чтобы выполнить префиксную редукцию данных, распределенных в группе. Операция возвращает в приемный буфер процесса i редукцию значений в посылающих буферах процессов с номерами 0, ..., i (включительно). Тип поддерживаемых операций, их семантика, и ограничения на буфера посылки и приема - такие же, как и для MPI_REDUCE.

MPI_SCAN(sendbuf, recvbuf, count, datatype, op, comm) Sendbuf-начальный адрес буфера посылки (альтернатива) Recvbuf-начальный адрес буфера приема (альтернатива) Count-количество элементов в буфере приема (целое) Datatype-тип данных элементов в буфере приема (дескриптор) Op-операция (дескриптор)
Comm-коммуникатор (дескриптор)

Функция **MPI_Allreduce** сохраняет результат редукции в адресном пространстве всех процессов, поэтому в списке параметров функции отсутствует идентификатор корневого процесса root. В остальном, набор параметров такой же, как и в предыдущей функции.

int MPI_Allreduce(void* sendbuf, void* recvbuf, int count,
MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

IN sendbuf - адрес начала входного буфера;
OUT recvbuf - адрес начала буфера приема;
IN count - число элементов во входном буфере;
IN datatype - тип элементов во входном буфере;
IN op - операция, по которой выполняется редукция;
IN comm - коммуникатор.

Функция барьерной синхронизации **MPI_BARRIER** блокирует вызывающий процесс, пока все процессы группы не вызовут её. В каждом процессе управление возвращается только тогда, когда все процессы в группе вызовут процедуру.

MPI_BARRIER(comm)

Comm-коммуникатор(дескриптор)

9. Понятие о виртуальной топологии процессов в MPI. Функции MPI для работы с виртуальными топологиями. Использование виртуальных топологий для реализации сеточных задач.

стр. 55

Топология - это механизм сопоставления процессам некоторого коммуникатора альтернативной схемы адресации.

В MPI топологии виртуальны. Может использоваться системой для опт-ии распред. процессов по физическим процессорам.

В MPI предусмотрены 2 вида топологий:
- декартова топология
- топология графа.

Декартова топология

- логическая топология, опред. многомерной решеткой.
- обобщение линейной и матричной топологий на произв. число измерений.

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

int MPI_CART_CREATE (MPI_Comm old_comm,
int ndims, int * dim_size, int * periods,
int reorder, MPI_Comm * new_comm)

Создает структуру "прямоугольная решетка" произвольной размерности.

ndims - число размерностей
dim_size - массив числа эл-тов в каждом
periods - логический массив, опред. яв-ся ли решетка периодической вдоль каждого измерения
reorder - логич. параметр, опред., что при значении true система разрешено менять порядок нумерации процессов для оптимиз. распред. по физическим процессорам.

Должна быть вызвана всеми процессами коммуникатора old_comm.

* MPI_CARD_COORDS (MPI_Comm comm,
int rank, int maxdims, int * coords)

Определение координат процесса по его рангу.

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

Опред. ранга процесса по его координатам

* int MPI_CARD_SUB (MPI_Comm old_comm, int * belongs, MPI_Comm * new_comm)

- belongs - массив, содерж. номера подгр-в, принадлежащих новому коммуникатору.

- Исп-ся для разделения комм-ра на подгруппы
- Создает коммуникаторы меньшей размерности

* int MPI_Carddim_get (MPI_Comm comm, int * ndims)

Получение размерности декартовой топологии.

* int MPI_Card_get (MPI_Comm comm, int ndims, int * dims, int * periods, int * coords)

- Получение информации о ~~координатах~~ декартовой
- топологии.

* int MPI_CARD_SHIFT (MPI_Comm comm, int direction, int dispe, int * source, int * dest)

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

- Для периодич. измерений – циклический сдвиг, для непериодических – линейный
- direction $\in [0, n-1]$ для n-мерной решетки
- source и dest можно исп. для пересылок

Использование виртуальных топологий для сеточных задач

Пример: ур-е Лапласа на прямоугольнике

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

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

5-точечный шаблон

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

Создается декартова топология

2	5	×	8
1	4	× ● ×	7
		×	
пр-сс 0	3		6

Необходим обмен данными на границах области процесса

(Можно реализовать с использованием описанных выше фун-ий)

Задание по курсу «Суперкомпьютерное моделирование и технологии»

Октябрь 2016

Содержание

1 Введение

Требуется методом конечных разностей приближенно решить задачу Дирихле для уравнения Пуассона в прямоугольной области. Задание необходимо выполнить на следующих ПВС Московского университета:

1. IBM Blue Gene/P,
2. «Ломоносов»

2 Математическая постановка дифференциальной задачи

В прямоугольной области

$$\Pi = [A_1, A_2] \times [B_1, B_2]$$

требуется найти дважды гладкую функцию $u = u(x, y)$, удовлетворяющую дифференциальному уравнению

$$-\Delta u = F(x, y), \quad A_1 < x < A_2,\ B_1 < y < B_2 \tag{1}$$

и дополнительному условию

$$u(x,y) = \varphi(x,y) \tag{2}$$

во всех граничных точках (x,y) прямоугольника. Оператор Лапласа Δ определен равенством:

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

Функции $F(x,y)$, $\varphi(x,y)$ считаются известными и определяются вариантом задания.

3 Разностная схема решения задачи.

В расчетной области П определяется прямоугольная сетка

$$\bar{\omega}_h = \{(x_i, y_j),\ i = 0,1,2,\ldots,N_1,\ j = 0,1,2,\ldots,N_2\},$$

где $A_1 = x_0 < x_1 < x_2 < \ldots < x_{N_1} = A_2$ – разбиение отрезка $[A_1, A_2]$ оси (ox),

$B_1 = y_0 < y_1 < y_2 < \ldots < y_{N_2} = B_2$– разбиение отрезка $[B_1, B_2]$ оси (oy).

Через ω_h обозначим множество внутренних, а через γ_h — множество граничных узлов сетки $\bar{\omega}_h$. Пусть $h_i^{(1)} = x_{i+1} - x_i$, $i = 0,1,2,\ldots,N_1-1$, $h_j^{(2)} = y_{j+1} - y_j$, $j = 0,1,2,\ldots,N_2-1$ – переменный шаг сетки по оси абсцисс и ординат соответственно. Средние шаги сетки определяются равенствами:

$$\hbar_i^{(1)} = 0.5\big(h_i^{(1)} + h_{i-1}^{(1)}\big),\ \ \hbar_j^{(2)} = 0.5\big(h_j^{(2)} + h_{j-1}^{(2)}\big).$$

Рассмотрим линейное пространство H функций, заданных на сетке ω_h. Будем считать, что в пространстве H задано скалярное произведение и евклидова норма

$$\big(u,v\big) = \sum_{i=1}^{N_1-1}\sum_{j=1}^{N_2-1} \hbar_i^{(1)}\hbar_j^{(2)} u_{i\,j} v_{i\,j}, \quad \|u\| = \sqrt{(u,u)}, \tag{3}$$

где $u_{i\,j} = u(x_i, y_j)$, $v_{i\,j} = v(x_i, y_j)$ – любые функции из пространства H.

Для аппроксимации уравнения Пуасона (1) воспользуемся пятиточечным разностным оператором Лапласа, который во внутренних узлах сетки определяется равенством:

$$-\Delta_h p_{i\,j} = \frac{1}{\hbar_i^{(1)}}\left(\frac{p_{i\,j} - p_{i-1\,j}}{h_{i-1}^{(1)}} - \frac{p_{i+1\,j} - p_{i\,j}}{h_i^{(1)}}\right) + \frac{1}{\hbar_j^{(2)}}\left(\frac{p_{i\,j} - p_{i\,j-1}}{h_{j-1}^{(2)}} - \frac{p_{i\,j+1} - p_{i\,j}}{h_j^{(2)}}\right).$$

Здесь предполагается, что функция $p = p(x_i, y_j)$ определена во всех узлах сетки $\bar{\omega}_h$.

Приближенным решением задачи (1), (2) называется функция $p = p(x_i, y_j)$, удовлетворяющая уравнениям

$$\begin{aligned} -\Delta_h p_{i\,j} &= F(x_i, y_j), \quad (x_i, y_j) \in \omega_h, \\ p_{i\,j} &= \varphi(x_i, y_j), \quad (x_i, y_j) \in \gamma_h. \end{aligned} \tag{4}$$

Эти соотношения представляют собой систему линейных алгебраических уравнений с числом уравнений равным числу неизвестных и определяют единственным образом неизвестные значения $p_{i\,j}$. Совокупность уравнений (4) называется разностной схемой для задачи (1), (2).

4 Метод решения системы линейных алгебраических уравнений.

Приближенное решение системы уравнений (4) может быть получено итерационным методом скорейшего спуска. В этом методе начальное приближение

$$p_{ij}^{(0)} = \varphi(x_i, y_j), \quad (x_i, y_j) \in \gamma_h,$$

во внутренних узлах сетки $p_{ij}^{(0)}$ – любые числа. Метод является одношаговым. Итерация $p^{(k+1)}$ вычисляется по итерации $p^{(k)}$ согласно равенствам:

$$p_{ij}^{(k+1)} = p_{ij}^{(k)} - \tau_{k+1} r_{ij}^{(k)},$$

где невязка

$$\begin{aligned} r_{ij}^{(k)} &= -\Delta_h p_{ij}^{(k)} - F_(x_i, y_j), \quad (x_i, y_j) \in \omega_h, \\ r_{ij}^{(k)} &= 0, \quad (x_i, y_j) \in \gamma_h. \end{aligned} \tag{5}$$

Итерационный параметр

$$\tau_{k+1} = \frac{\left(r^{(k)}, r^{(k)}\right)}{\left(-\Delta_h r^{(k)}, r^{(k)}\right)}.$$

Известно, что с увеличением номера итерации k последовательность сеточных функций $p^{(k)}$ сходится к точному решению p задачи (4) по норме пространства H, то есть

$$\|p - p^k\| \to +\infty, \quad k \to +\infty.$$

Существенно большей скоростью сходимости обладает метод сопряженных градиентов. Начальное приближение $p^{(0)}$ и первая итерация $p^{(1)}$ вычисляются так же, как и в методе скорейшего спуска. Последующие итерации осуществляются по формулам:

$$p_{ij}^{(k+1)} = p_{ij}^{(k)} - \tau_{k+1} g_{ij}^{(k)}, \quad k = 1, 2, \ldots$$

Здесь

$$\tau_{k+1} = \frac{\left(r^{(k)}, g^{(k)}\right)}{\left(-\Delta_h g^{(k)}, g^{(k)}\right)},$$

вектор

$$\begin{aligned} g_{ij}^{(k)} &= r_{ij}^{(k)} - \alpha_k g_{ij}^{(k-1)}, \quad k = 1, 2, \ldots, \\ g_{ij}^{(0)} &= r_{ij}^{(0)}, \end{aligned}$$

коэффициент

$$\alpha_k = \frac{\left(-\Delta_h r^{(k)}, g^{(k-1)}\right)}{\left(-\Delta_h g^{(k-1)}, g^{(k-1)}\right)}.$$

Вектор невязки $r^{(k)}$ вычисляется согласно равенствам (5). Итерационный процесс останавливается, как только

$$\|p^{(n)} - p^{(n-1)}\| < \varepsilon, \tag{6}$$

где ε – заранее выбранное положительное число. Заметим, что в последнем неравенстве вместо евклидовой сеточной нормы можно использовать любую другую норму пространства H, например, максимум-норму:

$$\|p\| = \max_{\substack{0<i<N_1 \\ 0<j<N_2}} \left|p(x_i, y_j)\right|. \tag{7}$$

10. Правая часть и граничное условие

$$F(x,y) = \frac{x^2+y^2}{4(4+xy)^{3/2}}, \quad \varphi(x,y) = \sqrt{4+xy},$$

соответственно, прямоугольник $\Pi = [0,4] \times [0,4]$.

11. Правая часть и граничное условие

$$F(x,y) = \frac{8(1-x^2-y^2)}{(1+x^2+y^2)^3}, \quad \varphi(x,y) = \frac{2}{1+x^2+y^2},$$

соответственно, прямоугольник $\Pi = [0,2] \times [0,2]$.

12. Правая часть и граничное условие

$$F(x,y) = \frac{8(1-x^2-y^2)}{(1+x^2+y^2)^3}, \quad \varphi(x,y) = \frac{2}{1+x^2+y^2},$$

соответственно, прямоугольник $\Pi = [-2,2] \times [-2,2]$.

Для аппроксимации дифференциальной задачи предлагается использовать равномерную прямоугольную сетку:

$$\begin{aligned} x_i &= A_2(i/N_1) + A_1(1-i/N_1), \ i = 0,1,2,\ldots,N_1, \\ y_j &= B_2(j/N_2) + B_1(1-j/N_2), \ j = 0,1,2,\ldots,N_2, \end{aligned} \tag{8}$$

либо неравномерную сетку, определенную равенствами:

$$\begin{aligned} x_i &= A_2 f(i/N_1) + A_1(1-f(i/N_1)), \ i = 0,1,2,\ldots,N_1, \\ y_j &= B_2 f(j/N_2) + B_1(1-f(j/N_2)), \ j = 0,1,2,\ldots,N_2. \end{aligned} \tag{9}$$

Здесь

$$f(t) = \frac{(1+t)^q - 1}{2^q - 1}, \quad 0 \leqslant t \leqslant 1,$$

где $q > 0$ – фиксированное число. Конкретный вид сетки определяется вариантом задания.

Приближенное решение разностной схемы (4) следует вычислять методом сопряженных градиентов. Для остановки итерационного процесса предлагается использовать условие (6), положив $\varepsilon = 10^{-4}$. Векторная норма может быть определена равенствами (3) либо соотношением (7) в зависимости от варианта задания.

Для каждого из перечисленных выше наборов функций $F(x,y)$, $\varphi(x,y)$ требуется
– подобрать точное решение задачи Дирихле,
– методом сопряженных градиентов построить приближенное решение на сетке с числом узлов $N_1 = N_2 = 1000$, определить погрешность решения

$$\psi = \|u(x_i, y_j) - p_{i\,j}\|,$$

– методом сопряженных градиентов построить приближенное решение на сетке с числом узлов $N_1 = N_2 = 2000$ и вновь определить погрешность решения.

Расчеты необходимо проводить на многопроцессорных вычислительных комплексах IBM Blue Gene/P и «Ломоносов», используя различное количество узлов сетки, указанное в требованиях к отчету. Для каждого расчета определить его продолжительность и ускорение по сравнению с аналогичным расчетом на одном вычислительном узле. При распараллеливании программы необходимо использовать двумерное разбиение области на подобласти прямоугольной формы, в каждой из которых отношение θ количества узлов по ширине и длине должно удовлетворять неравенствам $0.5 \leqslant \theta \leqslant 2$.

Билет 11
Метод скорейшего спуска и метод сопряженных градиентов для разностной задачи Дирихле.

$$-\Delta_h p_{ij} = F(x_i, y_j), \quad (x_i, y_j) \in \omega_h,$$
$$p_{ij} = \varphi(x_i, y_j), \quad (x_i, y_j) \in \gamma_h.$$

Приближенное решение системы уравнений

может быть получено итерационным методом скорейшего спуска. В этом методе начальное приближение $p_{ij}^{(0)} = \varphi(x_i, y_j), \quad (x_i, y_j) \in \gamma_h,$, во внутренних узлах сетки $p_{ij}^{(0)}$ - любые числа. Метод является одношаговым. Итерация $p^{(k+1)}$ вычисляется по итерации согласно равенствам: $p_{ij}^{(k+1)} = p_{ij}^{(k)} - \tau_{k+1} r_{ij}^{(k)}$, .

где невязка

$$r_{ij}^{(k)} = -\Delta_h p_{ij}^{(k)} - F(x_i, y_j), \quad (x_i, y_j) \in \omega_h,$$
$$r_{ij}^{(k)} = 0, \quad (x_i, y_j) \in \gamma_h.$$

Итерационный параметр

$$\tau_{k+1} = \frac{\left(r^{(k)}, r^{(k)}\right)}{\left(-\Delta_h r^{(k)}, r^{(k)}\right)}.$$

Известно, что с увеличением номера итерации k последовательность сеточных функций $p^{(k)}$ сходится к точному решению задачи по норме пространства H, то есть:

$$\|p - p^k\| \to +\infty, \quad k \to +\infty.$$

Существенно большей скоростью сходимости обладает метод сопряженных градиентов. Начальное приближение и первая итерация вычисляются так же, как и в методе скорейшего спуска. Последующие итерации осуществляются по формулам:

$$p_{ij}^{(k+1)} = p_{ij}^{(k)} - \tau_{k+1} g_{ij}^{(k)}, \quad k = 1, 2, \ldots$$

Здесь

$$\tau_{k+1} = \frac{\left(r^{(k)}, g^{(k)}\right)}{\left(-\Delta_h g^{(k)}, g^{(k)}\right)},$$

вектор

$$g_{ij}^{(k)} = r_{ij}^{(k)} - \alpha_k g_{ij}^{(k-1)}, \quad k = 1, 2, \ldots,$$
$$g_{ij}^{(0)} = r_{ij}^{(0)},$$

Коэффициент

$$\alpha_k = \frac{\left(-\Delta_h r^{(k)}, g^{(k-1)}\right)}{\left(-\Delta_h g^{(k-1)}, g^{(k-1)}\right)}.$$

Вектор невязки $r^{(k)}$ вычисляется согласно равенствам

$$r_{ij}^{(k)} = -\Delta_h p_{ij}^{(k)} - F_(x_i, y_j), \quad (x_i, y_j) \in \omega_h,$$
$$r_{ij}^{(k)} = 0, \quad (x_i, y_j) \in \gamma_h.$$

Итерационный процесс останавливается, как только

$$\|p^{(n)} - p^{(n-1)}\| < \varepsilon,$$

где ε – заранее выбранное положительное число. Заметим, что в последнем неравенстве вместо евклидовой сеточной нормы можно использовать любую другую норму пространства H, например, максимум-норму:

Задание 2. Исследование эффективности решения задачи Дирихле для уравнения Лапласа (один из вариантов)

- 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 $g(x, y, z)$.

М.В.Абакумов, А.В.Гулин Лекции по численным методам математической физики. Учебное пособие. – М.-:ИНФРА-М, 2013.-158

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.

- *итерационный метод Якоби (3).*

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

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

где n - номер итерации.

Уравнение Лапласа (2D)

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

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

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

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

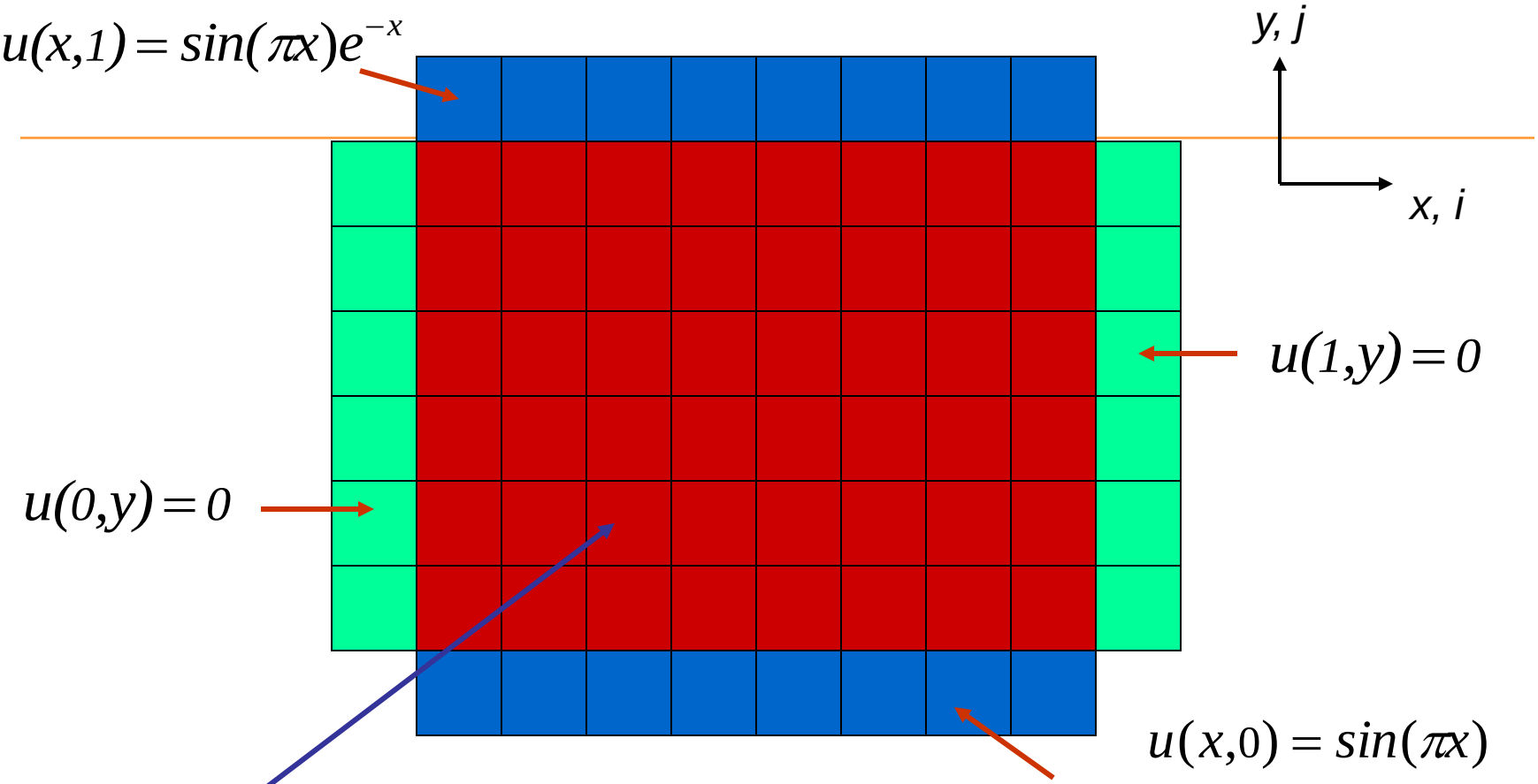
$$u_{i,j}^{n} = u^{n}(x_i, y_j) \qquad i = 0,1,2,\ldots,m+1; \;\; j = 0,1,2,\ldots,m+1 \qquad (5)$$

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

Для простоты

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

Вычислительная область



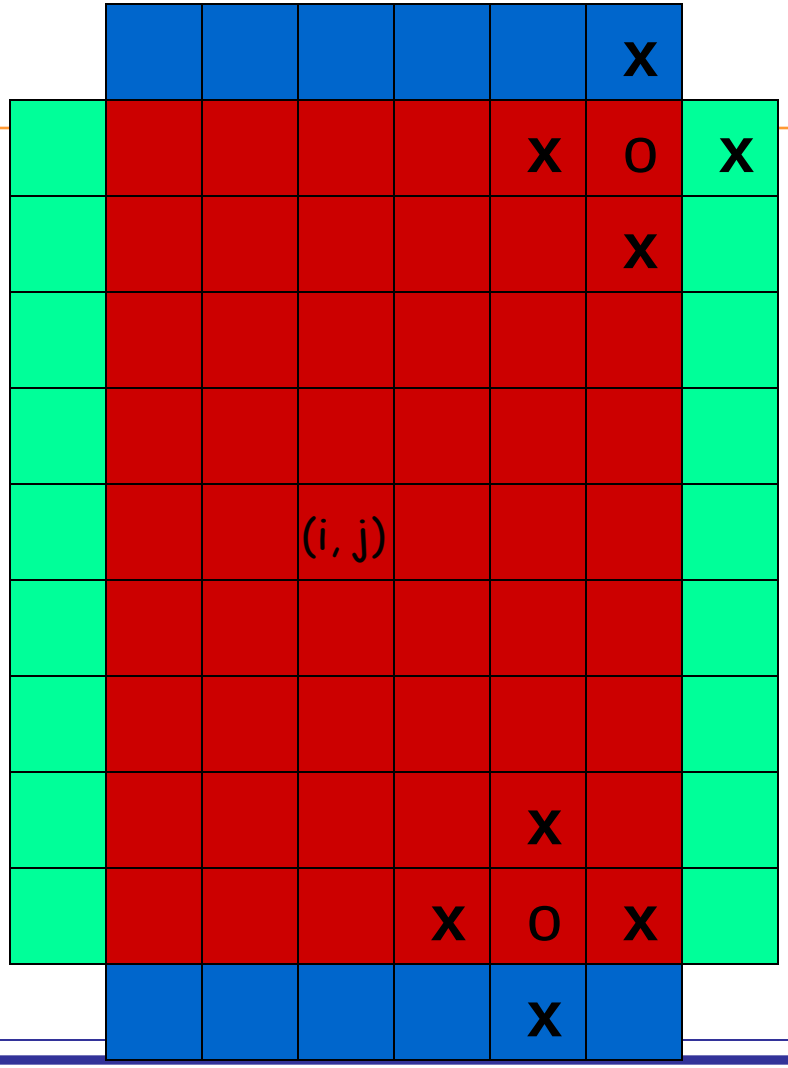
$$u_{i,j}^{n+1} \cong \frac{u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}}{4} \qquad i = 1,2,\ldots,m; \quad j = 1,2,\ldots,m$$

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.

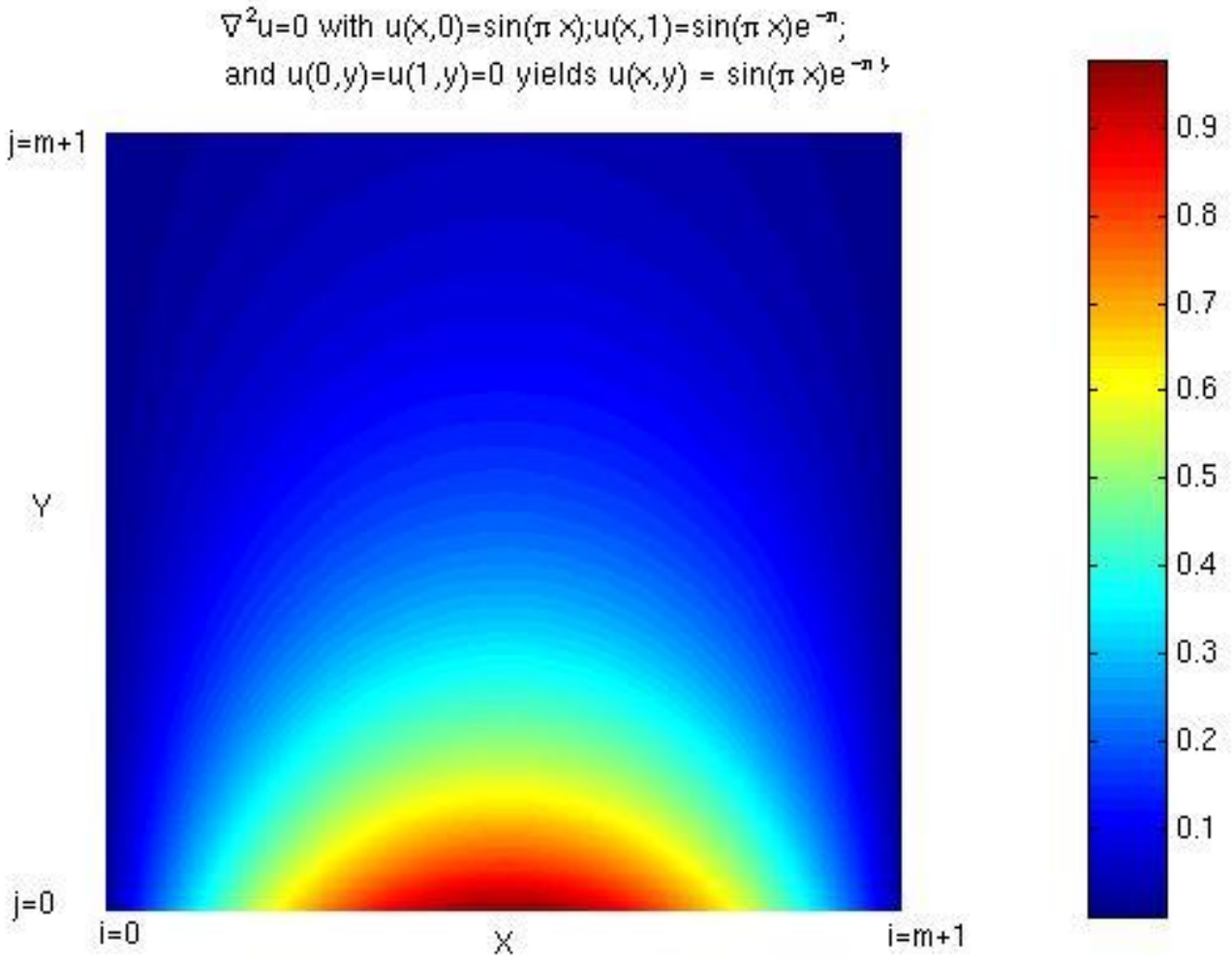


Метод Якоби

1. Задать начальные значения *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.

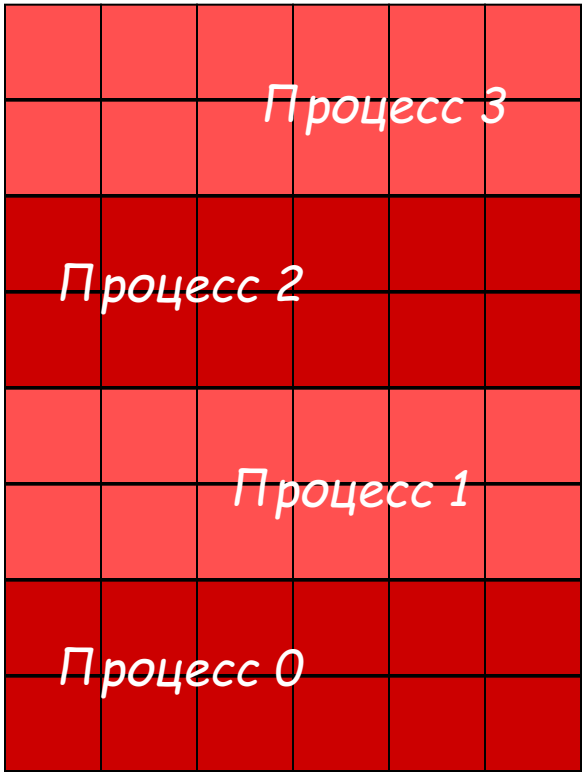
Это очень простая схема. Медленно сходится, поэтому не используется для решения реальных задач.

Решение (линии уровня)

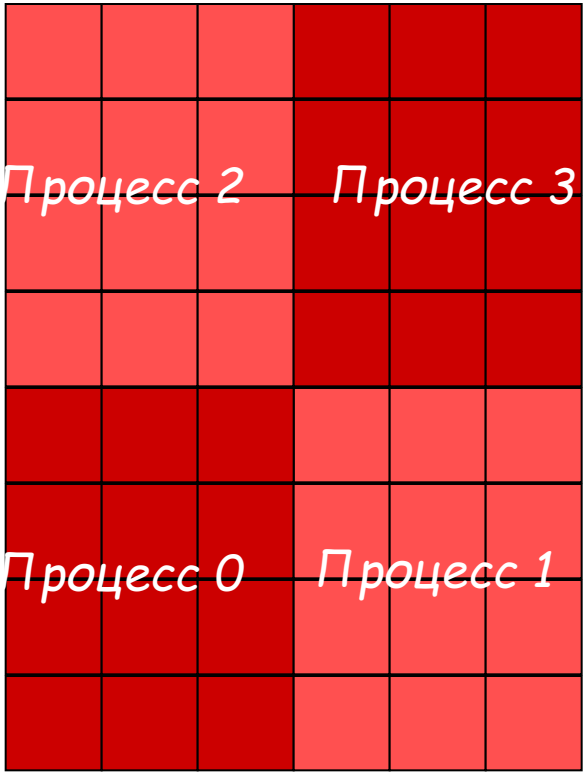


Параллельная реализация

1D Domain Decomposition



2D Domain Decomposition



Распределение данных – 1D

5-точечная схема требует значений от соседних потоков

Необходим обмен данными на границах области

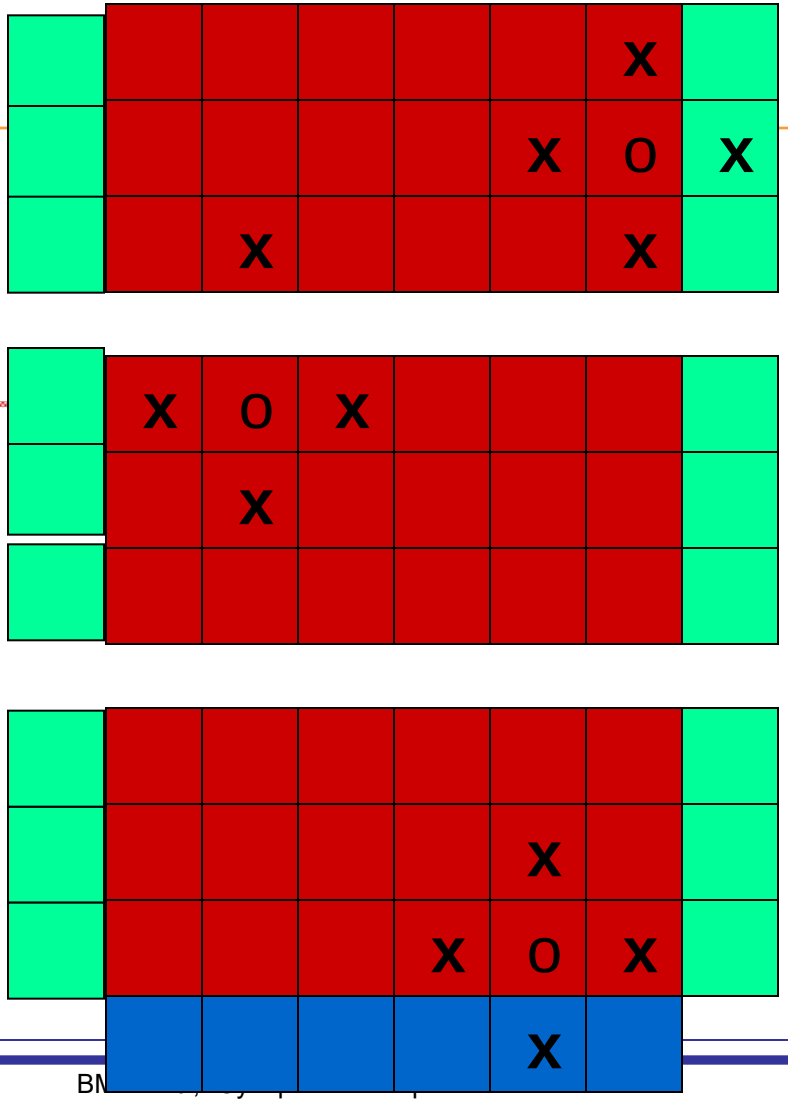
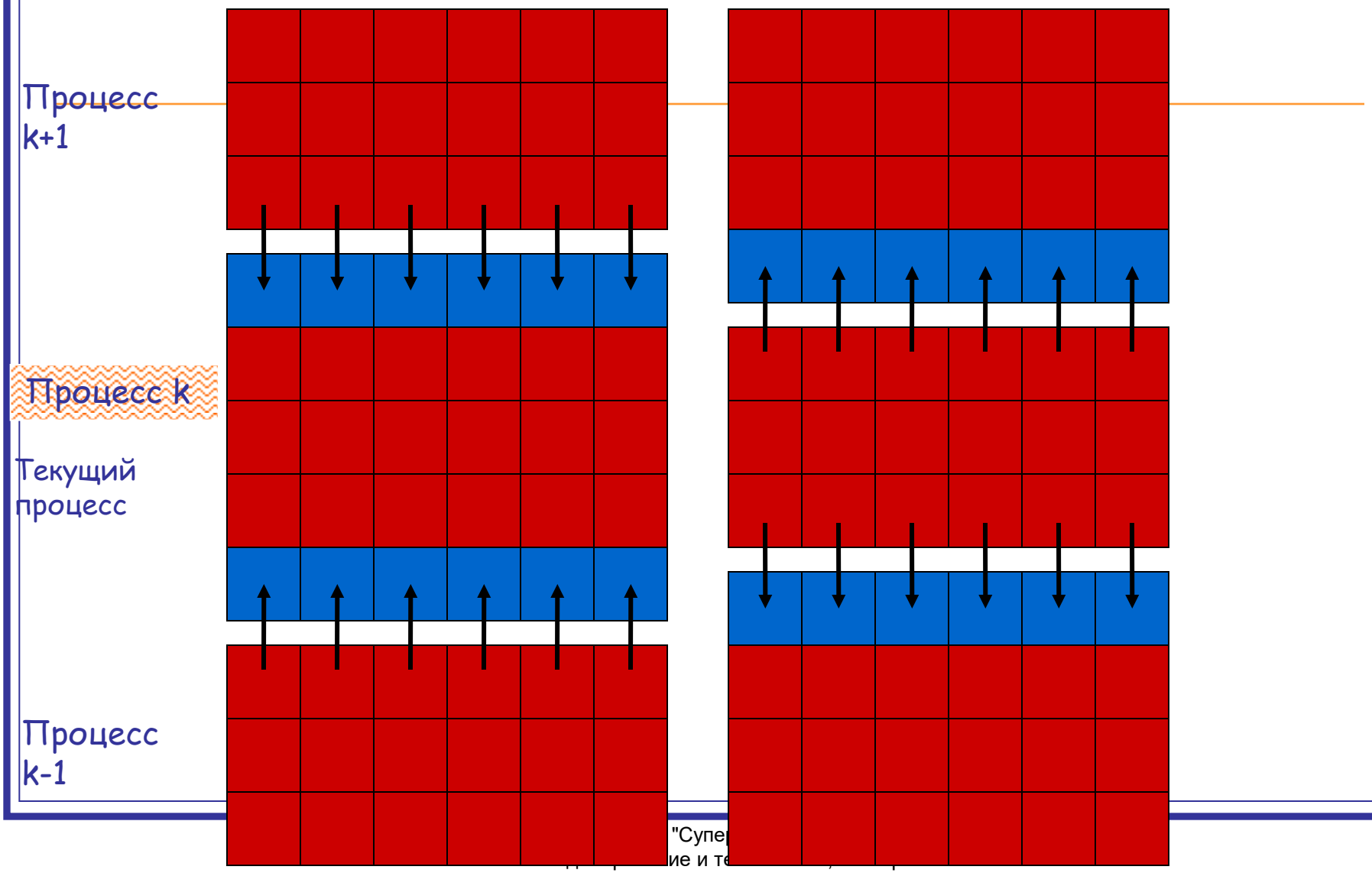


Схема передачи данных



Задача считается на сетке, в каждом узле сетки вычисляются некоторые значения, то есть каждому узлу нужны память и вычислительные ресурсы => узлы нужно распределить между процессами, чтобы все быстро считалось. Домен – область узлов сетки, которая достанется процессу после разбиения всей сетки между ними. Процесс считает и хранит в памяти только свою область и посылает сообщения с вычислениями своей области кому нужно(в данном случае всем соседям –процессам).

Одномерное разбиение видимо лучше тем, что соседа только два => посылать меньше сообщений. Хуже видимо тем, что верхняя граница эффективного количества процессов ниже, потому что минимальный домен будет 1xN, то есть всего в N процессах есть смысл. А если схема не пятиточечная, а какая-нибудь другая, то вообще всё плохо может быть, придётся общаться больше чем с одним соседом в каждом направлении общения. Этого в лекциях, кажется, нет, если неверно или есть еще идеи, излагайте!

Алгоритм определения размеров домена – число ячеек сетки делим на число процессов, получаем число ячеек на процесс. Ну и получаем из этого количества стороны домена исходя из совета во втором задании не «не больше, чем 2 к 1». Этого я тоже не нашел нигде.

Билет 14

Использование суперкомпьютеров для решения задач молекулярного моделирования.

Источники:

- Лекции (Lect MolecularDynamics Shumkin.pdf)
- Молекулярные переключатели и наноэлектроника
- Вычислительные нанотехнологии
- Первая задача: Моделирование из первых принципов молекулярного переключателя на основе реакции изомеризации
- Вторая задача: Моделирование из первых принципов фазового перехода в аморфном углероде

Задача 1

Происхождение проблемы: наращивание производительности микрочипов за счет уменьшения транзисторов (уменьшать кремниевые транзисторы можно только до 2 нм в толщину) => переход к принципиально новым технологиям (молекулярные переключатели). Возможное уменьшение последних до 2х порядков. Надо только идентифицировать и понять наименьшие физические системы, способные к переключению между состояниями с высоким и низким сопротивлением. Задача объемная (расчет многомерных задач для тысячи ядер и электронов), без суперкомпьютеров не обойтись.

Теоретическое исследование наблюдаемых эффектов и оптимизация процесса переключения проводятся в сотрудничестве с исследовательской лабораторией IBM в Цюрихе (одна из ведущих в мире по созданию нанотехнологий; эксперименты по молекулярным переключателям начаты в 2007г).

Постановка задачи: моделирование молекулярных переключателей (между состояниями с высоким и низким сопротивлением) на основе одной молекулы (нафталоцианина) под действием тока сканирующего туннельного микроскопа (CTM, STM).

Для переключения электропроводности (изменения сопротивления) используется процесс молекулярной (водородной) изомеризации – два атома водорода в центральной полости молекулы под действием внешнего потенциала, создаваемого STM, переходят от одних атомов азота к двум другим атомам азота, что эквивалентно повороту молекулы на 90 градусов в плоскости молекулы (рис.B на стр.2 лекции).

Логический ключ в одну молекулу == молекула, в которой "переключается" сопротивление.

Метод моделирования наносистем через **вычислительные модели “из первых принципов”** (FPMD - First-Principles Molecular Dynamics): молекулярная структура состоит из положительных ядер и отрицательных электронов, между которыми действуют кулоновские силы взаимодействия, и в моделях отсутствуют какие-либо подгоночные параметры.

- Построение: уравнение Шредингера (силы, действующие на ядра со стороны электронов; определяет положение электронов; задача на собственные значения) и кулоновское взаимодействие заряженных частиц (положение ядер). Электронная структура получается для каждой молекулы решением стационарного уравнения Шредингера для каждого шага классической молекулярной динамики ядер. Волновые функции – решения уравнений Шредингера – представляются в виде разложения по плоским волнам. Собственные значения определяют уровни энергии электронных состояний, а собственные функции − электронную конфигурацию системы.
- Проблемы при переходе от одной молекулы к сплошной среде (наномасштаб - мезомасштаб): в целом может отличаться от отдельной свободной молекулы; вероятная переключения в одной молекуле могут привести к переключениям в другой молекулярной матрице; эффективность переключения зависит от места, над которым находится игла микроскопа, даже в области одной молекулы.
- Огромное число молекул (~25000атомов) => требует суперкомпьютеров
- Масштабируемость FPMD кодов

Для решения используются 2 модели (последовательно):

1. Модель FPMD (квантовомеханический уровень): нахождение термодинамического равновесия и профиля свободной энергии (FES, = путь химической реакции) (второе моделируется методом метадинамики)
2. **Многомасштабная модель** (успешна ли моделированная система): соединение полученных на первом этапе профилей свободной энергии и исследований устойчивости системы на основе возбужденных орбиталей.

Моделирование молекулярного переключения:

1. Оптимизация геометрии молекулы: устанавливается атомный состав молекулы и вычисляются равновесные межатомные связи, которые соответствуют минимальной энергии.
2. Термолизация молекулы и термодинамическое равновесие.
3. Выбор коллективных и координационных переменных (для использования метод метадинамики для нахождения пути реакции изомеризации)
4. Моделирование переключения водорода вдоль выбранной координационной переменной.

Метод метадинамики: эффективное вычисление поверхности свободной энергии FES вдоль координационной переменной и получение энергетического барьера реакции изомеризации (бонус: эволюция электронной плотности в процессе метадинамики).

s – координационная переменная, r_{ij} - расстояние между i-м атомом водорода и j-м атомом азота:

$$s = \sum_{i,j \in A} \frac{1 - (r_{ij})^6}{1 - (r_{ij})^{12}}$$

Свободная энергия: F(s) = −k_BT ln(, где k_B − константа Больцмана, T − абсолютная температура, U(r) - потенциал взаимодействия, R_I - координата ядра I.
Искусственный потенциал V (s,t) есть сумма отталкивающихся потенциальных холмов в форме гауссианов,

$$V(s,t) = -\sum_{t_i < t} H \cdot exp\left[-\frac{(s(t) - s^i)^2}{2W_i^2}\right]$$

каждый из которых имеет высоту H: .
Плюс уравнения квантовомеханические: $\mu s^{\cdot\cdot} = -k(s - S\{R_J\})) - \partial/\partial s\, V(s,t)$
R_I

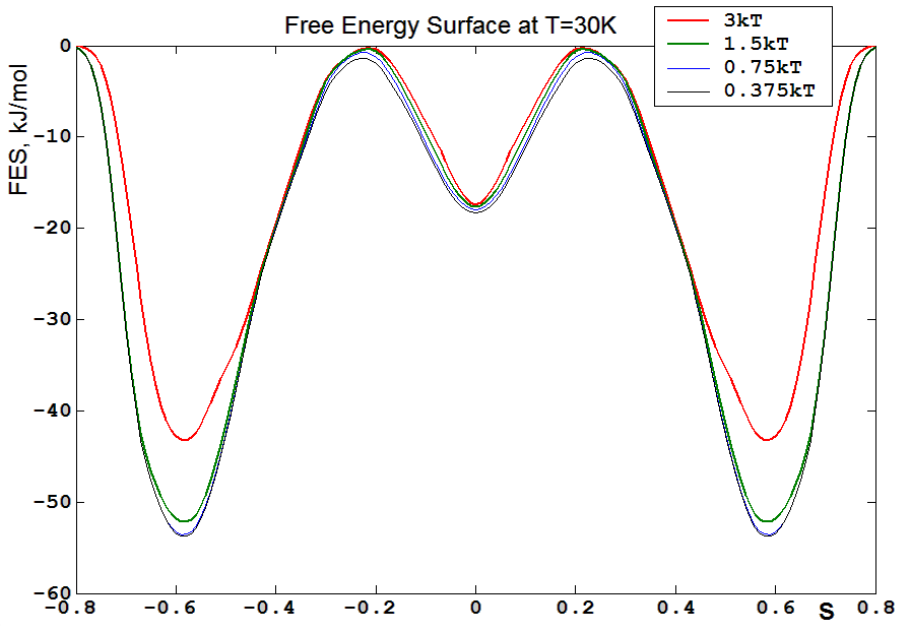
$M_I \ddot{R}_I = -\nabla_{R_I} U(R_I) - \frac{\partial S(\{R_J\})}{\partial R_I}$ k(S$(\{R_J\}) - s$), где M_I , - масса и координата ядра, μ, R_J - фиктивная масса и координата электрона.

Потенциал вынуждает систему посещать те точки в конфигурационном пространстве, которые соответствуют яме реагента. Метадинамика добавляет холмы в состояние реагента до тех пор, пока не скомпенсирует первую яму свободной энергии так, что теперь система может найти низшее переходное состояние к следующему локальному минимуму (продукту реакции). Когда все локальные минимумы свободной энергии заполнены холмами, система может свободно двигаться от состояния реагента к состоянию продукта. В результате профиль может быть получен с произвольной точностью как взятый с

$$(s) = -\sum_{t_i < t} H_i \cdot exp\left[-\frac{(s(t) - s^i)^2}{2W_i^2}\right]$$

обратным знаком потенциал: F

Сходимость достигнута путем проведения последовательных расчетов с уменьшающейся высотой гауссиана: H_b=52 кДж/моль = 0.54 эВ



Время вычислений для одной молекулы на BlueGene/P:

- 82 атома, 129 орбиталей, 540 000 плоских волн для каждой орбитали, шаг по времени 0.1125 фс, сетка 320x320x192, размер ячейки 30x30x18 Å (10^{-10} м)
- На 512 узлах – время 20 ч

Рис. на стр. 5 - структура молекулы и значение электронной плотности вокруг ядра.

Задача 2. Классическая молекулярная динамика со связями

Постановка задачи: переключение структуры аморфного углерода (a-C), обладающего большим сопротивлением, к кристаллической структуре с малым сопротивлением под воздействием тока атомносилового микроскопа (АСМ).

Для моделирования системы здесь также используется модель FPMD.

Метод решения:

1. Уравнения Ньютона для ядер (интегрируются алгоритмом Верлета) в усредненном поле электронов.
   $$M_I \ddot{R}_I = -\sum_J \nabla_{R_I} \frac{Z_I Z_J}{|R_I - R_J|} + \int \rho(r) \nabla_{R_I} \frac{Z_I}{|R_I - r|} dr$$
2. Уравнение Шредингера для электронов (нахождение электронной плотности) – нелинейная задача на собственные значения
   $\rho(r) = \sum_{n=1}^{N_e} |\psi_n(r)|^2$ - электронная плотность, N_e число электронов в системе,
   $\left(-\frac{1}{2}\Delta + U_{eff}(\rho, r)\right)\psi_n(r) = \epsilon_n \psi_n(r)$,
   $U_{eff}(\rho, r) = U_H(\rho, r) + U_{xc}(\rho, \nabla\rho, r) - \sum_I \frac{+Z_I}{|R_I - r|} + \sum_{I<J} \frac{Z_I Z_J}{|R_I - R_J|}$, $\Delta U_H = -4\pi\rho$
   U_{eff} – эффективный потенциал, E_{XC} – обменно-корреляционная энергия.
3. Электронные орбитали раскладываются в ряд Фурье: $\psi_n(r,t) = \frac{1}{\sqrt{\Omega}} \sum_k c_{n,k}(t) e^{ikr}$, где Ω объем ячейки, k вектор обратной решетки, $c_{n,k}$ фурье-коэффициенты орбитали n.
   Для преобразования величин используется быстрое преобразование (БПФ).

Схема решения: по коэффициентам $c_{n,k}$ с использованием “3dFFT” (трехмерное быстрое преобразование Фурье) вычисляются элементы матрицы гамильтониана H. Затем решается нелинейная задача на собственные значения и проводится пересчет плотности и эффективного потенциала в реальном пространстве с использованием “3dFFT”.

Параллельная программа на BlueGene/P: интегрирование 200 уравнений для ядер и решение 400 трехмерных задач для электронов. Время одного расчета на 512 узлах занимает ≈ 20 часов. Возможно использование гибридной схемы распараллеливания MPI/OpenMP.

Уровни распараллеливания:

1. **Крупноблочное** распараллеливание на распределенной памяти MPI – распределение коэффициентов волновых функций для всех электронных состояний на все процессоры. Распределение данных минимизирует число передач с поддержанием загрузки в обоих пространствах.
   3dFFT Реальное пространство <=> K-пространство
2. Для **длинных циклов** - “OpenMP” на общей памяти узла.
3. **Taskgroups** – группы процессоров. Процессоры организованы как двумерная сетка. Схема требует в два раза меньше коммуникаций, чем обычная схема. Процессорные группы могут быть оптимально распределены по тору.
4. **Методы копирования.** Интегралы по траекториям в молекулярной динамике. Квантовые ядра. Многомерная метадинамика.

Отражение задач на архитектуру суперкомпьютера:

- Blue Gene/P. Архитектура трехмерного тора дополненная сетью дерева
- Возникают условия на дизайн FPMD кодов
- Для P< 1024 (BGP) проблема неважна
- Для P=65536 (BGL) хороший выбор отображения дает 60% ускорения
- Оптимизация отображения - N! отображений.
- Проблема использования параллельных библиотек (ScaLAPACK). Узлы не под контролем кода моделирования
- Оптимизация - критическая процедура для больших разбиений
- Разработка удобных программ для визуализации трафика сообщений на сети тора. Создание автоматизированных процедур отображения
- Оптимизированный код Qbox показал производительность 207 TFlop/s (наивысшая производительность на научных приложениях). Blue Gene/L

Производительность CPMD (Car-Parrinello Molecular Dynamics, КПМД, численный квантовомеханический код молекулярной динамики, который используется в FPMD) на Blue Gene/P:

- CPMD масштабируется до 128000 процессоров
- 100 атомов масштабируются до 2000 процессоров с 70% эффективности и длительностью производственного цикла ~600ps/неделя (МГУ)
- Для систем ~1000 атомов масштабируется на 8000-16000 процессоров с эффективностью 80% и длительностью производственного цикла ~ 20 ps/неделя
- Наибольшая система в настоящий момент 20000 атомов на 16 стойках.

(15) Архитектурные особенности графических процессоров, направленные на массивно-параллельные вычисления. Особенности работы с памятью графического пр-ра.

GPU - Graphics Processing Unit

GPGPU - General-purpose computing on GPU - вычисление общего назначения на GPU. Первые GPU от NVIDIA с поддержкой GPGPU - GeForce восьмого поколения, 680 (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.

Архитектура GPU NVIDIA:

CPU Intel Core i7

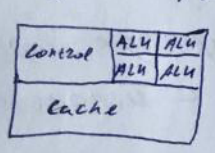
- Небольшое число мощных независимых ядер
- До 22 ядер ~3.0 ГГц каждое;
- Поддержка виртуальных потоков (Hyper-Threading)
- 3-х уровневый кэш, большой кэш L3 до 50 Мб
- На каждое ядро L1 = 32 КБ (data) + + 32 КБ (instructions), L2 = 256 КБ
- Обращения в память обрабатываются отдельно для каждого процесса/нити

GPU Streaming Multiprocessor (SM)

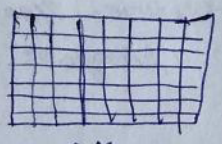
- Потоковый мультипроцессор
- "Единица" построения устройства (как ядро в CPU)
  - 32 скалярных ядра CUDA Core, 1.5 ГГц
  - 2 Warp scheduler
  - Файл регистров, 128 КБ
  - 2х уровневый кэш
  - Текстурные юниты
  - 16 x Special Function Unit (SFU) - интерполяция и трансцендентная математика одинарной точности
  - 16x Load/Store

Сравнение GPU и CPU

- Сотни упрощенных вычислительных ядер, работающих на небольшой тактовой частоте ~1.8 ГГц
- Небольшие кеши на GPU
  - 32 CUDA-ядра разделяют 64 КБ L1
  - L2 общий для всех CUDA ядер 2 МБ, 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.



CPU


GPU

Утилизация латентности памяти

- Цель: эффективно загружать CUDA-ядра
- Проблема: латентность памяти (это временная задержка сигнала при работе оперативной памяти)
- Решение:
  - CPU: сложная иерархия кешей
  - GPU: много нитей, покрывать обращение одних нитей в память вычислениями в других за счет быстрого переключения контекста
- За счёт наличия сотен ядер и поддержки миллионов нитей (потребителей) на GPU легче утилизировать всю полосу пропускания.

Билет 16

Методы эффективной организации параллельных вычислений на графических процессорах.

1) Утилизация латентности памяти

Цель: эффективно загружать Ядра

Проблема: латентность памяти

Решение: CPU: Сложная иерархия кешей

GPU: Много нитей, покрывать обращения одних нитей в память вычислениями в других за счёт быстрого переключения контекста. За счёт наличия сотен ядер и поддержки миллионов нитей (потребителей) на GPU легче утилизировать всю полосу пропускания

2) SIMT (Single instruction, multiple thread) и масштабирование

Виртуальная

- GPU может поддерживать миллионы виртуальных нитей

- Виртуальные блоки независимы

Программу можно запустить на любом количестве SM

Аппаратная

- Мультипроцессоры независимы

Можно «нарезать» GPU с различным количеством SM

3) 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 (если CUDA_LAUNCH_BLOCKING не установлена на 1)

- Копирование между двумя областями памяти на устройстве

- Копирование с хоста на устройство менее 64KB

- Копирования, выполняемые функциями с окончанием *Async

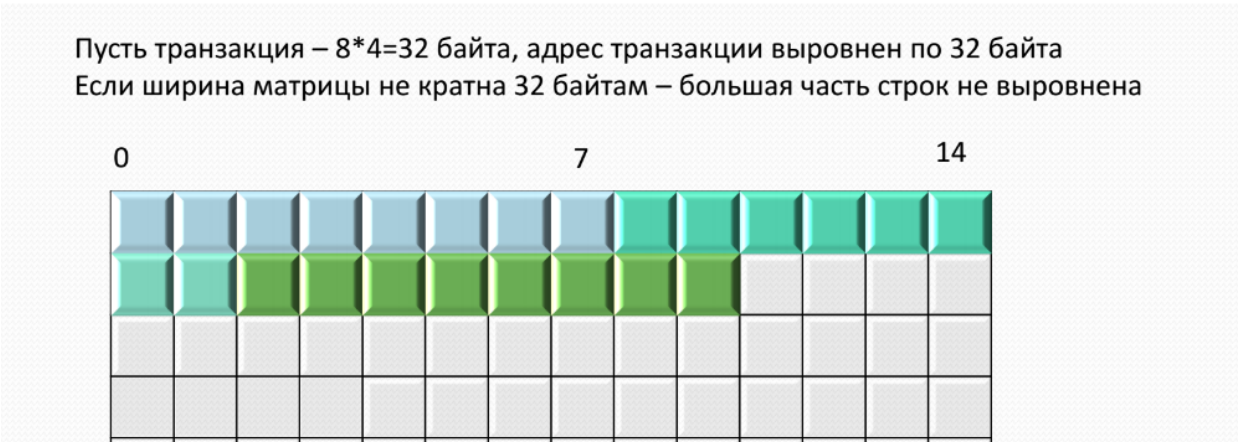
- cudaMemSet – присваивает всем байтам области памяти на устройстве одинаковое значение (чаще всего используется для обнуления)

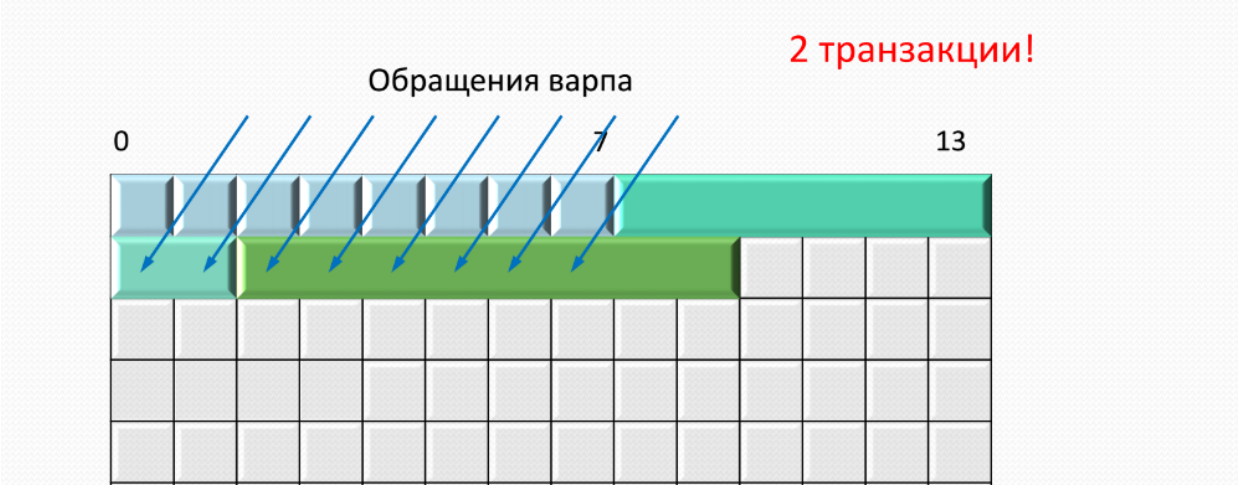
4) Работа с глобальной памятью

- Расположена в **DRAM GPU**
- Объём до 4Gb
  - Параметр устройства totalGlobalMem
- **Кешируется** в кешах L1 и L2:
  - L1 – на каждом мультипроцессоре
    максимальный размер 48KB
    минимальный размер 16KB
  - L2 – на устройстве
    максимальный размер 768 KB
    Параметр устройства l2CacheSize

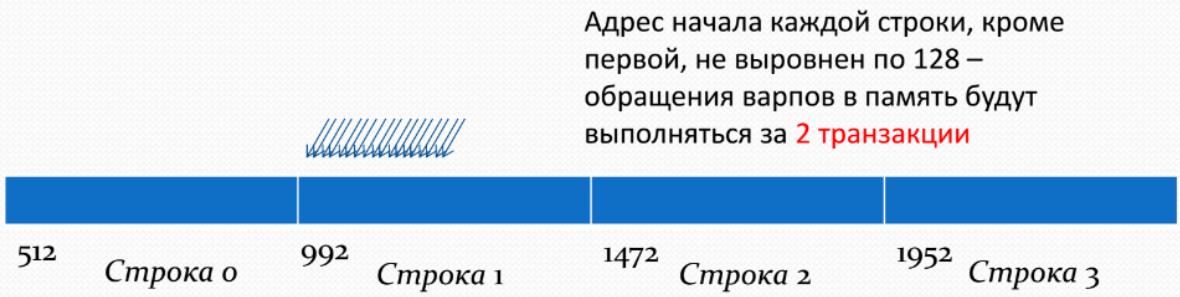
а) Обращения нитей варпа в память должны быть пространственно-локальными

б) Начала строк матрицы должны быть выровнены

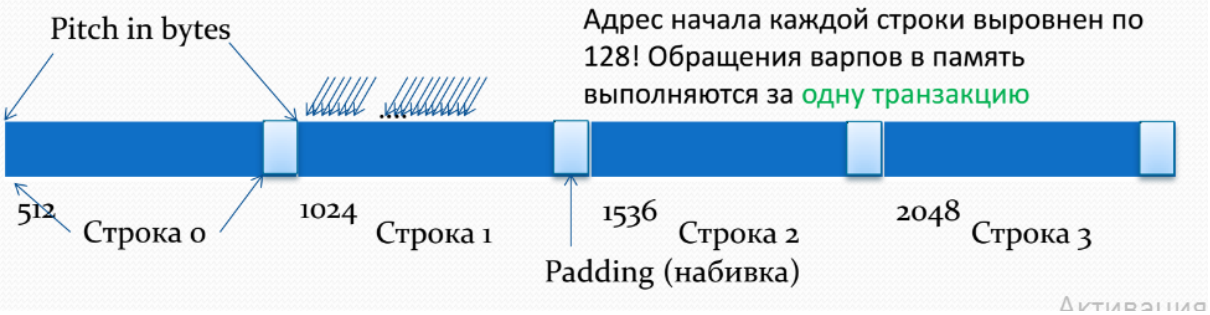




- Матрицы хранятся в линейном виде, по строкам
- Пусть длина строки матрицы – 480 байт (120 float)
  - обращение – matrix[idy*120 + idx]



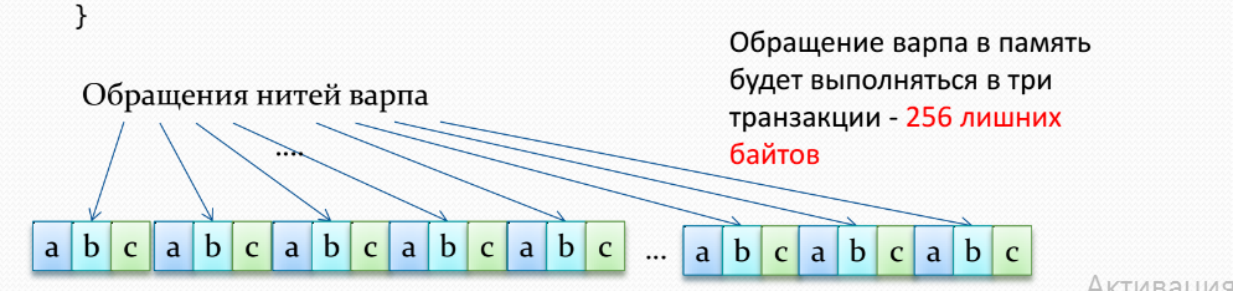
- Дополним каждую строку до размера, кратного 128 байтам – в нашем случае, 480 + 32 = 512, это наш pitch – фактическая ширина в байтах
- Эти байты никак не будут использоваться, т.е. 32/512=6% лишней памяти будет выделено (Но для больших матриц эта доля будет существенно меньше)
- Зато каждая строка будет выровнена по 128 байт
  - Обращение matrix[idy*128+ idx]



в) Массивы структур -> структура с массивами

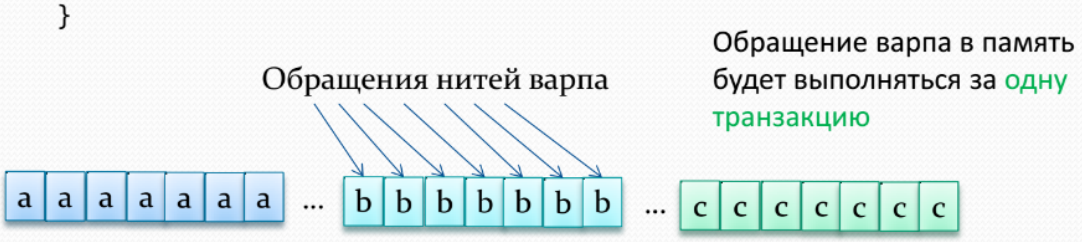
Массивы структур:

```
struct example {
   int a;
   int b;
   int c;
}
__global__ void kernel(example * arrayOfExamples) {
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    arrayOfExamples[idx].c =
         arrayOfExamples[idx].b + arrayOfExamples[idx].a;
}
```

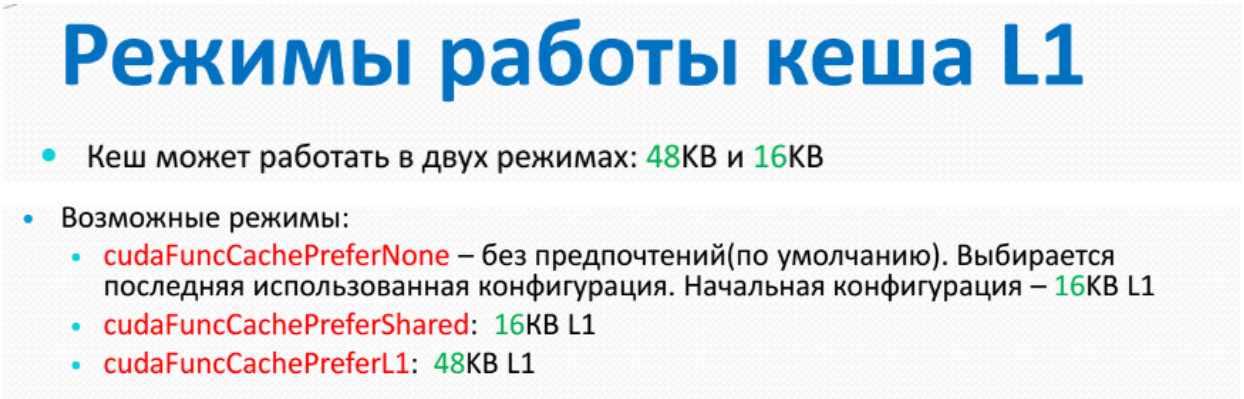


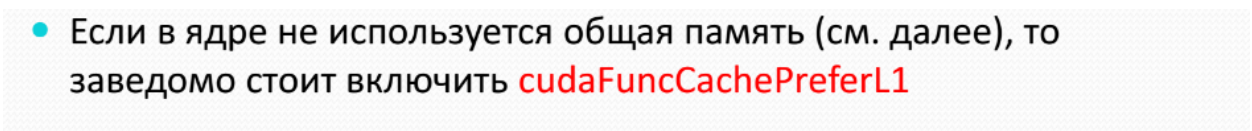
Структуры с массивами:

```
struct example {
   int *a;
   int *b;
   int *c;
}
__global__ void kernel(example arrayOfExamples) {
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    arrayOfExamples.c[idx] =
         arrayOfExamples.b[idx] + arrayOfExamples.a[idx];
}
```



г) 16KB vs 48KB L1

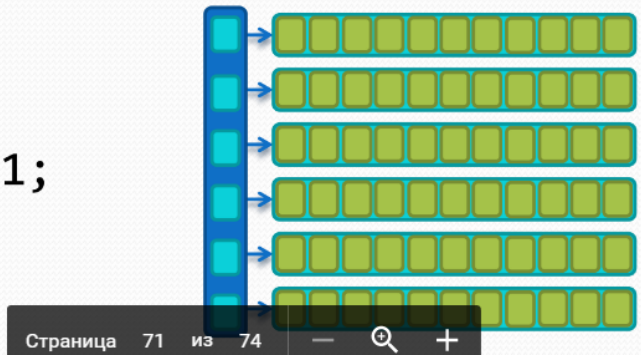




д) Избегаем косвенной адресации

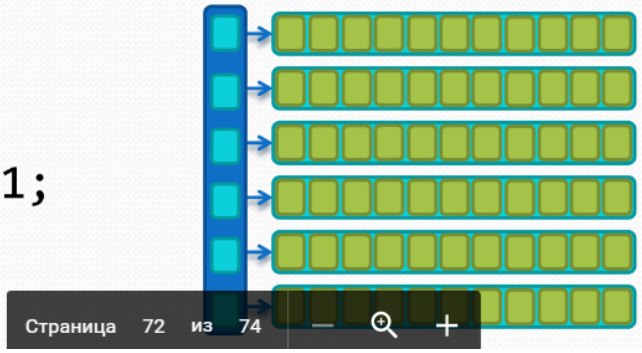
- Использование косвенной адресации нежелательно, поскольку требует двух чтений из памяти (сначала A[i], потом A[i][j])

```
float **A;
A[i][j] = 1;
```



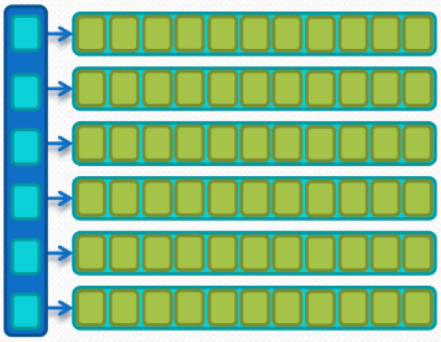
- A[i] для разных нитей варпа скорее всего будут в одной кеш-линии -> одно обращение к кешу
- Но A[i,j] в общем случае могут быть «разбросаны»

```
float **A;
A[i][j] = 1;
```



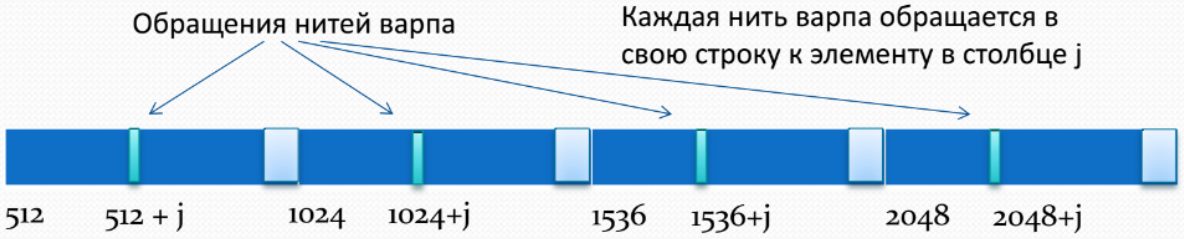
- Принципиального решения нет!
- Скорее всего придется переработать алгоритм

```
float **A;
A[i][j] = 1;
```



е) Избегаем обращений нитей варпа к столбцу матрицы

- Матрица расположена по строкам, а обращение идёт по столбцам

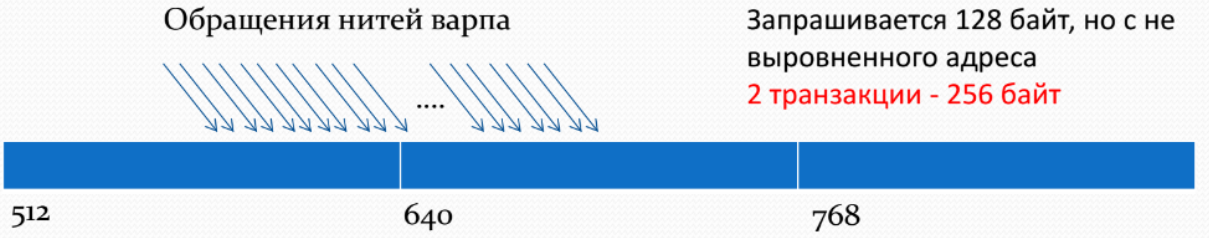


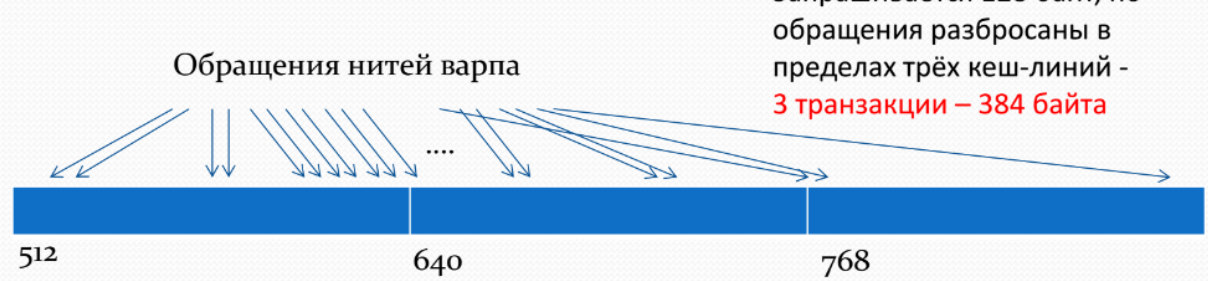
Если матрица имеет размер больше 128 байт, то эти обращения ни за что не «влезут» в одну транзакцию!

- Решение – хранить матрицу в транспонированном виде!
  - В этом случае обращения по столбцам превратятся в обращения к последовательным адресам
  - Выделять память под транспонированную матрицу также через `cudaMallocPitch`

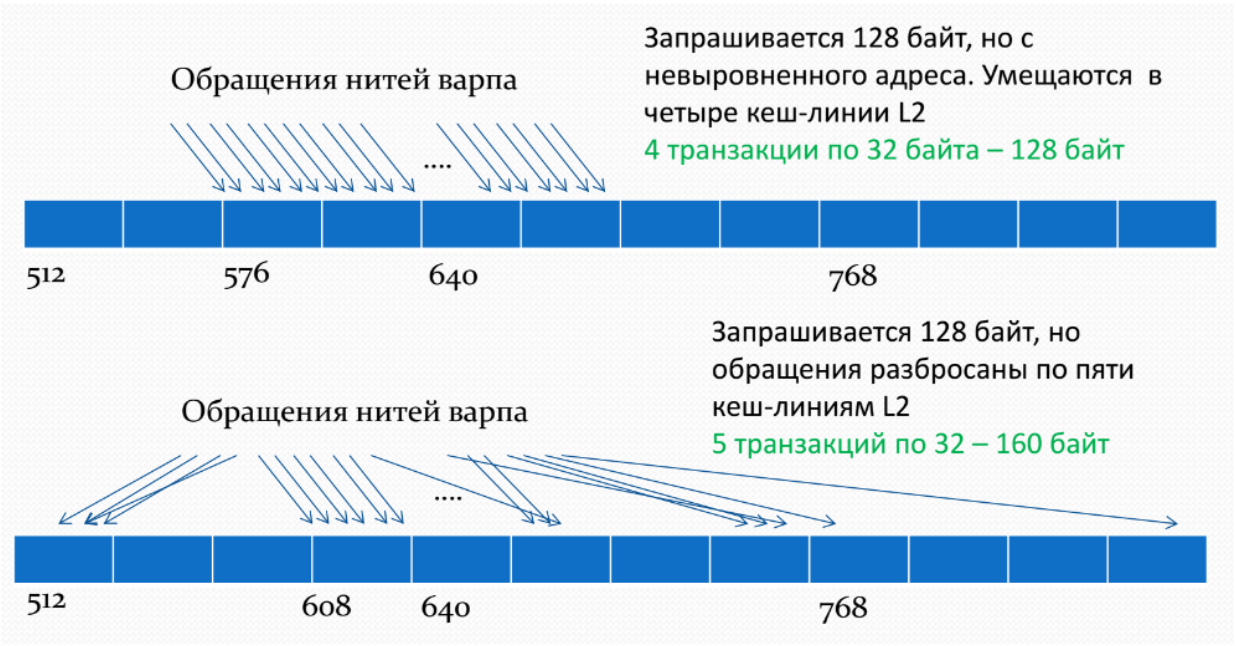
ж) В случае сильно разреженного доступа проверяем работу с отключенным кешем

- 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
  - Если нужен один байт – все равно загрузится 128





- Ядра взаимодействуют не с памятью напрямую, а с кешами
- Транзакция – выполнение загрузки кеш-линии
  - У кеша **L1** кеш-линии **128** байт, у **L2** - **32** байта, **naturally aligned**
  - Кеш грузит из памяти всю кеш-линию, даже если нужен один байт
- Можно обращаться в память минуя кеш L1
  - Транзакции будут по 32 байта



Билет 17

Источник - https://www.dropbox.com/sh/pr6dnsq3yrptadp/AAB_ZriXJwQF-45lt_IirLcFa/Lect%20Tensor%20Tyrtyshnikov.pdf?dl=0

Тензор, по-простому, это многомерная матрица. Существует специальное каноническое представление тензоров

Для этого вводится понятия чистого тензора (скелетона, или тензора 1-го ранга):

$$a(i,j,k) = u(i)v(j)w(k).$$

Любой тензор в каноническом представлении задается как сумма чистых тензоров:

$$a(i,j,k) = \sum_{\alpha=1}^{r} u(i,\alpha)v(j,\alpha)w(k,\alpha)$$

И выходит, задается 3-мя матрицами

Быстрое умножение

Существует быстрый алгоритм умножения матриц – алгоритм Штрассена

Так вот, этот алгоритм можно объяснить с точки зрения разложения тензора в

$$\begin{bmatrix} c_1 & c_2 \\ c_3 & c_4 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} \quad c_k = \sum_{i=1}^{n^2} \sum_{j=1}^{n^2} h_{ijk}\ a_i\ b_j$$

$$h_{ijk} = \sum_{\alpha=1}^{R} u_{i\alpha}\ v_{j\alpha}\ w_{k\alpha}$$

$$\Rightarrow \quad c_k = \sum_{\alpha=1}^{R} w_{k\alpha} \left(\sum_{i=1}^{n^2} u_{i\alpha} a_i \right) \left(\sum_{j=1}^{n^2} v_{j\alpha} b_j \right)$$

канонический вид.

Для n=2 все ясно, и количество умножений = 7 (что совпадает с результатом Штрассена)

А для бОльшей размерности результат получается рекурсивно

Используя рекурсию, можно перемножить две матрицы порядка n = с затратой не

более $= n^{\log_2 7}$ умножений и сложений-вычитаний.

Тензорный поезд

Тензорный поезд – это разложение тензора размерности n (имеется ввиду, что количество измерений – n штук) в произведение n матриц, первая матрица – вектор, последняя – вектор-столбец, матрицы посередине – прямоугольные матрицы.

$$a(i_1, i_2, i_3, i_4, i_5) =$$
$$\sum g_1(i_1, \alpha_1) g_2(\alpha_1, i_2, \alpha_2) g_3(\alpha_2, i_3, \alpha_3) g_4(\alpha_3, i_4, \alpha_4) g_5(\alpha_4, i_3)$$
$$= \underbrace{A_1^{(i_1)}}_{1\times r_1} \underbrace{A_2^{(i_2)}}_{r_1\times r_2} \underbrace{A_3^{(i_3)}}_{r_2\times r_3} \underbrace{A_4^{(i_4)}}_{r_3\times r_4} \underbrace{A_5^{(i_5)}}_{r_4\times 1}$$

Имеет смысл применять тензорный поезд, когда Тензор – разреженный

Интерполяция матриц:

Можно интерполировать матрицу по строкам или по столбцам следующим образом:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \to \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} A_{11}^{-1} [A_{11} A_{12}]$$

Сложение элементов вектора очень большой размерности

Пусть имеется вектор $a = (a_1, \dots, a_N), N = 10^{83}$

Представляем его в виде тензора:

$$a(i) = a(i_1, \dots, i_{83})$$

Каждый элемент вектора => каждый элемент тензора представляется в виде разложения тензорным поездом:

$$i = \overline{i_1 i_2 \dots i_d} \qquad d = 83$$

$$a(i) = a(i_1, \dots, i_d) = \sum_{\alpha_1, \dots, \alpha_{d-1}} g_1(i_1, \alpha_1) g_2(\alpha_1, i_2, \alpha_2) \dots g_d(\alpha_{d-1}, i_d)$$

$$\sum_{i_1, \dots, i_d} a(i_1, \dots, i_d) = \sum_{\alpha_1, \dots, \alpha_{d-1}} \hat{g}_1(\alpha_1) \hat{g}_2(\alpha_1, \alpha_2) \dots \hat{g}_d(\alpha_{d-1})$$

$$\hat{g}_k = \sum_{i_k} g_k$$

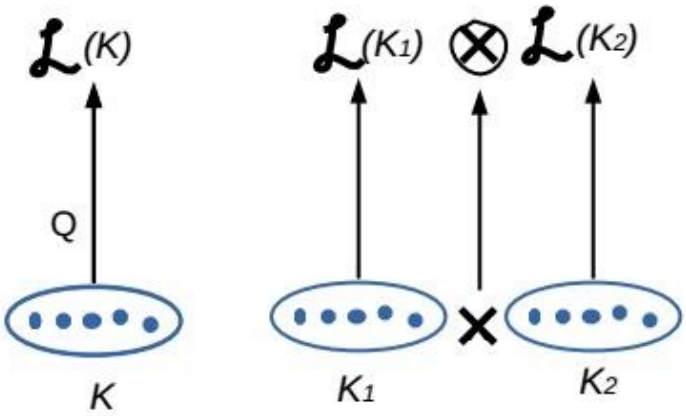
Данный инструмент хорошо использовать для вычисления одномерных интегралов большой размерности а также многомерных интегралов.

Билет 18
Принцип интерференции в квантовой механике

Квантовая теория должна объяснять все.
1900 - 1960 гг.: анализ микромира с точки зрения точного описания отдельных частиц, погружение "вглубь" микрокосма.
1970 - н.в. : синтез сложных систем на микроуровне, кондесированные состояния , квантовый компьютер, суперкомпьютерное моделирование динамики на квантовом уровне.
Далее: квантовая биология (много-частичная КМ необходима для объяснения отдельных эффектов), квантовая экономика, квантовая политика, квантовое принятие решений (отдельные результаты).

Квантовые состояния как линейные комбинации классических

Процедура квантования - переход к линейной оболочке. $\mathcal{L}(K)$ состоит из линейных комбинаций $|\Psi\rangle = \sum_j \lambda_j |j\rangle$, где все $|j\rangle \in K$ считаются взаимно ортогональными с единичной нормой. Объединение реальных систем приводит к тензорному произведению - что ведет к экспоненциальному росту размерности пространства квантовых состояний.
Основная проблема: Сколько элементов может быть в множестве классических состояний K?



Принцип интерференции

Амплитуда вдоль одного пути умножается на комплексное число, пропорциональное пройденному пути.

Амплитуды вдоль разных путей, приходящих в одну точку, складываются. При этом надо учесть всевозможные пути, ведущие в данную точку.

Вероятность есть квадрат модуля амплитуды.

Это правило точно формализуется в виде матричной эволюции

Зерно разрешения - фундаментальное число, требуемое для математической корректности квантовой механики

Заряд электрона - амплитуды взаимодействия его с фотоном.
Заряд электрона точно не известен.
Фиксируется зерно разрешения по пространству и времени dx и dt.
Устанавливается пробное значение заряда электрона и его массы.
Определяется амплитуда основного процесса: взаимодей ствия электрона с фотоном, так что вероятность, рассчитанная по этиой амплитуде, дает согласие с экспериментами.
При уменьшении зерна разрешимости dx, dt пробные заряды и массы изменятся, но вероятность будет только точнее совпадать с экспериментами. При этом значения заряда и массы, подобранные для одного эксперимента, дают столь же точное предсказание для всех других экспериментов.
Сходимость процесса назначения зарядов и масс называется теоремой и перенормировках квантовой электродинамики. Она была строго доказана Н.Боголюбовым в 1955 г.

Обычная квантовая механика. Эволюция квантового состояния во времени

1. Эволюция в присутствии наблюдателя, который измеряет систему. Измерение системы, находящейся в состоянии $|\Psi\rangle = \sum_j \lambda_j |j\rangle$ есть случайная величина, принимающая значения $|j\rangle$ с вероятностями $p_j = |\lambda_j|^2$.
2. Эволюция в отсутствии наблюдателя: решение уравнения Шредингера $ih|\dot{\Psi}\rangle = H|\Psi\rangle$

$$|\Psi(t)\rangle = U_t|\Psi(0)\rangle,\ U_t = e^{-\frac{i}{h}Ht}$$

$U_t:\ |\Psi(0)\rangle \to |\Psi(t)\rangle$ - оператор эволюции, $H = E_{kin} + E_{pot}$- эрмитов оператор энергии; для одной частицы в потенциале V гамильтониан $H = \frac{p^2}{2m} + V(r),\ p = -ih\nabla$. Для конечномерного приближения H и U_t- матрицы, $|\Psi\rangle$ - столбец, зависящий от времени.

Интеграл Фейнмана по путям - непрерывный аналог матричной механики

Если есть k шагов продолжительности dt: $U_t = U_k U_{k-1} \dots U_0$, матричный элемент перехода будет

$$u_t(i,j) = \sum_{q_1, q_2, \dots, q_k} u_{dt}(q_1, j) u_{dt}(q_2, q_1) \dots u_{dt}(i, q_k)$$

что в непрерывном случае дает оператор эволюции в виде фейнмановского ядра

$$K(2,1) = \int\limits_{\gamma:\ 1 \to 2} exp(\frac{i}{h} S[\gamma]) \mathcal{D}\gamma$$

где действие S вдоль траектории γ есть

$S[\gamma] = \int\limits_{t_0}^{t_1} L(\dot{x}, x, t) dt,\ L = E_{kin} - E_{pot},\ \gamma:\ x = x(t),\ t_0 \le t \le t_1$. Эволюция имеет вид

$$\Psi(2) = \int K(2,1) \Psi(1) d1, \quad 1 = x_1,\ 2 = x_2$$

Классическая механика как следствие интерференции амплитуд

Классическая траектория γ_{cl} отличается от прочих тем, что на ней $\frac{\delta S[\gamma_{cl}]}{\delta \gamma} = 0$. Поэтому в интеграле

$$K(2,1) = \int\limits_{\gamma:\ 1 \to 2} exp(\frac{i}{h} S[\gamma]) \mathcal{D}\gamma$$

окрестности классической траектории складываются конструктивно, а окрестности прочих - деструктивно, если среднее действие на элементарном шаге превосходит постоянную Планка $h \approx 10^{-27} erg\ sec$. Если же среднее действие мало, неклассические траектории могут дать серьезный вклад.

Необходимость применять квантовую механику зависит от продолжительности dt элементарного шага моделирования процесса, то есть от сценария процесса.

Например, в задаче вычисления возможных состояний ассоциации молекул можно считать ядра классическими, а электроны нужно считать - квантовыми (модель Борна-Оппергеймера).

Квантовый компьютер как вызов квантовой теории

1. Квантовый алгоритм есть рецепт специально оранизованного классического управления Гамильтонианом $H = H(t)$. Квантовое вычисление - соответствующая этому гамильтониану эволюция волновой функции квантового состояния определенной системы частиц.

Предположим, что нет никаких ограничений на размер множества K классических состояний, подлежащих квантованию. Тогда эволюция волновой функции $|\Psi\rangle$ при этом может привести к непредсказуемому результату, который невозможно получить никаким классическим алгоритмом в обозримое время. Такие способы управления называются быстрыми квантовыми алгоритмами.

2. Всем математическим методам квантовой теории можно придать форму эффективных классических алгоритмов.

3. **Физика квантовых компьютеров требует новых методов и более общего взгляда на область приложений квантовой механики.**

4. Критерий качества математической модели: неизбежная редукция волновой функции от унитарной эволюции в ходе вычисления должна совпадать с наблюдаемой в экспериментах декогерентностью - спонтанным отклонением наблюдаемыой динамики от унитарного закона.