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

* **Векторно-конвейерные компьютеры**

Середина 70-х годов.

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

**Программирование:** векторизация самых внутренних циклов.

CRAY-1

Суперкомпьютер Cray X-MP

* **Векторно-параллельные компьютеры**

Начало 80-х годов.

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

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

Суперкомпьютер Cray Y-MP , CRAY X-MP

* **Массивно-параллельные компьютеры**

Начало 90-х годов.

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

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

Cray T3D Intel Paragon XPS140

* **Параллельные компьютеры с общей памятью**

Середина 90-х годов.

**Особенности архитектуры:** сотни процессоров объединяются над общей памятью. **Программирование:** единое адресное пространство, локальные и глобальные переменные, Linda, OpenMP

DEC AlphaServer

Суперкомпьютер Sun StarFire

* **Кластеры из узлов с общей памятью Начало 2000-х.**

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

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

Суперкомпьютер МГУ “Чебышев” “K” суперкомпьютер

* **Кластеры из узлов с общей памятью с ускорителями**

Середина 2000-х.

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

**Программирование:** MPI+OpenMP+OpenCL/CUDA;

Суперкомпьютер МГУ “Ломоносов” Суперкомпьютер Tianhe-2

**АРХИТЕКТУРНАЯ Особенность**

|  |  |  |  |
| --- | --- | --- | --- |
| № | 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 |

1. **Технологии Интел для высокопроизводительных вычислений.**
2. **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

1. **Программно-аппаратная архитектура суперкомпьютеров Ломоносов и Blue Gene/P.**

# 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).

**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](http://green500.org/))
* система воздушного охлаждения

**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
* асинхронные операции межпроцессорных обменов (выполняются параллельно с вычислениями)
* операции ввода-вывода перенаправляются I/O-картам через сеть коллективных операций

**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](http://hpc.cmc.msu.ru/bgp/development/networks#torus) (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](http://hpc.cmc.msu.ru/bgp/development/networks#collective) (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)

**Описание суперкомпьютера Ломоносов.**

1. **Последовательная и параллельная сложность алгоритмов, информационный граф и ресурс параллелизма алгоритмов.**

**Последовательная сложность** (serial complexity) алгоритма - число операций, которые нужно выполнить при его последовательном исполнении.

**Параллельная сложность** (parallel complexity) алгоритма - число шагов, за которое можно выполнить данный алгоритм в предположении доступности неограниченного числа необходимых процессоров (функциональных устройств, вычислительных узлов, ядер и т.п.). Параллельная сложность алгоритма понимается как высота канонической [ярусно-параллельной формы](https://algowiki-project.org/ru/%D0%93%D0%BB%D0%BE%D1%81%D1%81%D0%B0%D1%80%D0%B8%D0%B9#.D0.AF.D1.80.D1.83.D1.81.D0.BD.D0.BE-.D0.BF.D0.B0.D1.80.D0.B0.D0.BB.D0.BB.D0.B5.D0.BB.D1.8C.D0.BD.D0.B0.D1.8F_.D1.84.D0.BE.D1.80.D0.BC.D0.B0_.D0.B3.D1.80.D0.B0.D1.84.D0.B0_.D0.B0.D0.BB.D0.B3.D0.BE.D1.80.D0.B8.D1.82.D0.BC.D0.B0).

**Граф алгоритма** (algorithm graph) - это ориентированный ациклический мультиграф, вершины которого соответствуют операциям алгоритма, а дуги - передаче данных между ними. Вершины графа алгоритма могут соединяться несколькими дугами, в частности, когда в качестве разных аргументов одной и той же операции используется одна и та же величина. Граф алгоритма почти всегда является параметризованным 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 параметризованную информационную историю. Он сохраняет её информативность, при это обладая компактностью за счёт параметризации. Разработана методика построения графа алгоритма по исходному тексту программ.

**Ресурс параллелизма.**

Здесь приводится оценка [*параллельной сложности*](https://algowiki-project.org/ru/%D0%93%D0%BB%D0%BE%D1%81%D1%81%D0%B0%D1%80%D0%B8%D0%B9#.D0.9F.D0.B0.D1.80.D0.B0.D0.BB.D0.BB.D0.B5.D0.BB.D1.8C.D0.BD.D0.B0.D1.8F_.D1.81.D0.BB.D0.BE.D0.B6.D0.BD.D0.BE.D1.81.D1.82.D1.8C) алгоритма: числа шагов, за которое можно выполнить данный алгоритм в предположении доступности неограниченного числа необходимых процессоров (функциональных устройств, вычислительных узлов, ядер и т.п.). Параллельная сложность алгоритма понимается как высота канонической ярусно-параллельной формы [[1]](https://algowiki-project.org/ru/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BE%D0%BF%D0%B8%D1%81%D0%B0%D0%BD%D0%B8%D1%8F_%D1%81%D0%B2%D0%BE%D0%B9%D1%81%D1%82%D0%B2_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%BE%D0%B2#cite_note-VVVVVV-1). Необходимо указать, в терминах каких операций дается оценка. Необходимо описать сбалансированность параллельных шагов по числу и типу операций, что определяется шириной ярусов канонической ярусно-параллельной формы и составом операций на ярусах.

Параллелизм в алгоритме часто имеет естественную иерархическую структуру. Этот факт очень полезен на практике, и его необходимо отразить в описании. Как правило, подобная иерархическая структура параллелизма хорошо отражается в последовательной реализации алгоритма через циклический профиль результирующей программы (конечно же, с учетом графа вызовов), поэтому циклический профиль ([п.1.5](https://algowiki-project.org/ru/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BE%D0%BF%D0%B8%D1%81%D0%B0%D0%BD%D0%B8%D1%8F_%D1%81%D0%B2%D0%BE%D0%B9%D1%81%D1%82%D0%B2_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%BE%D0%B2#.D0.9E.D0.BF.D0.B8.D1.81.D0.B0.D0.BD.D0.B8.D0.B5_.D1.81.D1.85.D0.B5.D0.BC.D1.8B_.D1.80.D0.B5.D0.B0.D0.BB.D0.B8.D0.B7.D0.B0.D1.86.D0.B8.D0.B8_.D0.BF.D0.BE.D1.81.D0.BB.D0.B5.D0.B4.D0.BE.D0.B2.D0.B0.D1.82.D0.B5.D0.BB.D1.8C.D0.BD.D0.BE.D0.B3.D0.BE_.D0.B0.D0.BB.D0.B3.D0.BE.D1.80.D0.B8.D1.82.D0.BC.D0.B0)) вполне может быть использован и для отражения ресурса параллелизма.

Для описания ресурса параллелизма алгоритма (ресурса параллелизма информационного графа) необходимо указать ключевые параллельные ветви в терминах [*конечного*](https://algowiki-project.org/ru/%D0%93%D0%BB%D0%BE%D1%81%D1%81%D0%B0%D1%80%D0%B8%D0%B9#.D0.9A.D0.BE.D0.BD.D0.B5.D1.87.D0.BD.D1.8B.D0.B9_.D0.BF.D0.B0.D1.80.D0.B0.D0.BB.D0.BB.D0.B5.D0.BB.D0.B8.D0.B7.D0.BC) и [*массового*](https://algowiki-project.org/ru/%D0%93%D0%BB%D0%BE%D1%81%D1%81%D0%B0%D1%80%D0%B8%D0%B9#.D0.9C.D0.B0.D1.81.D1.81.D0.BE.D0.B2.D1.8B.D0.B9_.D0.BF.D0.B0.D1.80.D0.B0.D0.BB.D0.BB.D0.B5.D0.BB.D0.B8.D0.B7.D0.BC) параллелизма.

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

**Модель MPI**

ν Параллельная программа состоит из процессов, процессы могут быть многопоточными.

ν MPI реализует передачу сообщений между процессами.

ν Межпроцессорное взаимодействие предполагает:

ν синхронизацию

ν перемещение данных из адресного пространства одного процесса в адресное пространство другого процесса.

**Основы передачи данных в MPI**

ν Технология передачи данных MPI предполагает кооперативный обмен.

ν Данные посылаются одним процессом и принимаются другим.

ν Передача и синхронизация совмещены.

**Основные группы функций MPI**

ν Определение среды

ν Передачи «точка-точка»

ν Коллективные операции

ν Производные типы данных

ν Группы процессов

ν Виртуальные топологии

ν Односторонние передачи

ν Параллельный ввод-вывод

**Обработка ошибок MPI-функций**

Определяется константой MPI\_SUCCESS

1. **Функции двухточечных передач данных в MPI. Способы организации неблокирующих передач.**

**Варианты передачи «точка-точка»**

ν Синхронные пересылки

ν Асинхронные передачи

ν Блокирующие передачи

ν Неблокирующие передачи

|  |  |
| --- | --- |
| **Blocking sends** | **MPI\_Send(buffer,count,type,dest,tag,comm)** |
| **Non-blocking sends** | **MPI\_Isend(buffer,count,type,dest,tag,comm,request)** |
| **Blocking receive** | **MPI\_Recv(buffer,count,type,source,tag,comm,status)** |
| **Non-blocking receive** | **MPI\_Irecv(buffer,count,type,source,tag,comm,request)** |

**Неблокирующие:** немедленный возврат из функций, пользователь должен контролировать завершение передач

**Блокирующие:** возврат из функций передачи сообщений только по завершению передачи

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.

MPI\_Iprobe - Неблокирующая функция чтения параметров полученного сообщения

MPI\_Request\_free - Функция снятия запроса без ожидания завершения неблокирующей операции

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

MPI\_Bcast() – Broadcast (one to all) - Широковещательная рассылка данных

MPI\_Reduce() – Reduction (all to one)

MPI\_Allreduce() – Reduction (all to all)

MPI\_Scatter() – Distribute data (one to all)

MPI\_Gather() – Collect data (all to one) - производит сборку блоков данных, посылаемых всеми процессами группы, в один массив процесса с номером root.

MPI\_Alltoall() – Distribute data (all to all)

MPI\_Allgather() – Collect data (all to all) - выполняется так же, как MPI\_Gather, но получателями являются все процессы группы.

MPI\_Barrier() - блокирует работу вызвавшего ее процесса до тех пор, пока все другие процессы группы также не вызовут эту функцию. Завершение работы этой функции возможно только всеми процессами одновременно (все процессы "преодолевают барьер" одновременно).

*MPI\_Gatherv* - позволяет собирать блоки с разным числом элементов от каждого процесса, так как количество элементов, принимаемых от каждого процесса, задается индивидуально с помощью массива recvcounts.

MPI\_Allgatherv - является аналогом функции MPI\_Gatherv, но сборка выполняется всеми процессами группы.

**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

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

**Декартова топология**

ν Логическая топология, определяемая многомерной решеткой.

ν Обобщение линейной и матричной топологий на произвольное число измерений.

ν Для создания коммуникатора с декартовой топологией используется функция MPI\_Cart\_create. С помощью этой функции можно создавать топологии с произвольным числом измерений, причем по каждому измерению в отдельности можно накладывать периодические граничные условия.

**Основные функции:**

MPI\_CART\_CREATE - Создает структуру «прямоугольная решетка» произвольной размерности.

MPI\_CART\_COORDS - Функция определения координат процесса по его идентификатору

MPI\_CART\_RANK - Функция получения идентификатора процесса по его координатам

MPI\_CART\_SUB - Используется для разделения коммуникатора на подгруппы . MPI\_CART\_SUB создает новый коммуникатор меньшей размерности

MPI\_CARTDIM\_GET

MPI\_CART\_GET - Функция возвращает число измерений в декартовой топологии ndims для коммуникатора comm. Результат может быть использован в качестве параметра для вызова функции MPI\_Cart\_get, которая служит для получения более детальной информации.

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

1. **Задача Дирихле для уравнения Пуассона в прямоугольнике, разностная аппроксимация задачи на прямоугольной неравномерной сетке.**

**Задача Дирихле:**

В прямоугольной области Π = [𝐴1, 𝐴2] × [𝐵1, 𝐵2] требуется найти дважды гладкую функцию 𝑢 = 𝑢(𝑥, 𝑦), удовлетворяющую дифференциальному уравнению –∆𝑢 = 𝐹(𝑥, 𝑦), 𝐴1 < 𝑥 < 𝐴2, 𝐵1 < 𝑦 < 𝐵2 и дополнительному условию 𝑢(𝑥, 𝑦) = 𝜙(𝑥, 𝑦) во всех граничных точках (𝑥, 𝑦) прямоугольника. Оператор Лапласа ∆ определен равенством: . Функции 𝐹(𝑥, 𝑦), 𝜙(𝑥, 𝑦) считаются известными.

**Разностная аппроксимация.**

1. **Метод скорейшего спуска и метод сопряженных градиентов для разностной задачи Дирихле.**

**Метод скорейшего спуска.**

В этом методе начальное приближение внутренних узлах сетки – любые числа. Метод является одношаговым. Итерация вычисляется по итерации 𝑝(𝑘) согласно равенствам: где невязка Итерационный параметр . 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 ‖ → +∞, 𝑘 → +∞.

**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: 𝑘 = 1, 2, Здесь , 𝑘 = 1,2,…., коэффициент Вектор невязки 𝑟(𝑘) вычисляется согласно равенствам (5). Итерационный процесс останавливается, как только ‖𝑝(𝑛) − 𝑝(𝑛−1) ‖ < 𝜀, (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 0<𝑖<𝑗

1. **Одномерное и двумерное разбиение прямоугольной сетки на домены. Сравнение методов разбиения. Алгоритм определения размеров домена.**
2. **Суперкомпьютерное моделирование турбулентных течений.**

😭

1. **Использование суперкомпьютеров для решения задач молекулярного моделирования.**

**😭**

1. **Архитектурные особенности графических процессоров, направленные на массивно-параллельные вычисления. Особенности работы с памятью графического процессора.**

😭

1. **Методы эффективной организации параллельных вычислений на графических процессорах.**

😭

1. **Тензорные методы представления многомерных массивов.**

😭

1. **Принцип интерференции в квантовой механике.**

😭Процедура квантования – переход к линейной оболочке.

**Принцип интерференции.**

* Амплитуда вдоль одного пути умножается на комплексное число пропорциональное пройденному пути.
* Амплитуда вдоль разных путей , приходящих в одну точку, складываются. При этом нужно учесть всевозможные пути, ведущие в данную точку.
* Вероятность есть квадрат модуля амплитуды.
* Это правило точно формализуется в виде матричной эволюции.