Высокопроизводительные вычисления

Автор: Дружинин Денис Вячеславович

Программа курса

1 Параллелизм компьютерных вычислений

2 **Архитектура высокопроизводительных вычислительных систем**

- 2.1 Классификация вычислительных систем
- 2.2 Классификация MIMD систем

3 **Grid-системы**

4 **Облачные технологии**

5 **Общие вычисления на видеокарте (GPGPU)**

- 5.1 Понятие о GPGPU
- 5.2 Nvidia CUDA

6 **Программирование для высокопроизводительных вычислений**

- 6.1 Методология проектирования параллельных алгоритмов
- 6.2 Декомпозиция для выделения параллелизма

I. Параллелизм компьютерных вычислений

Причины вычислительного параллелизма

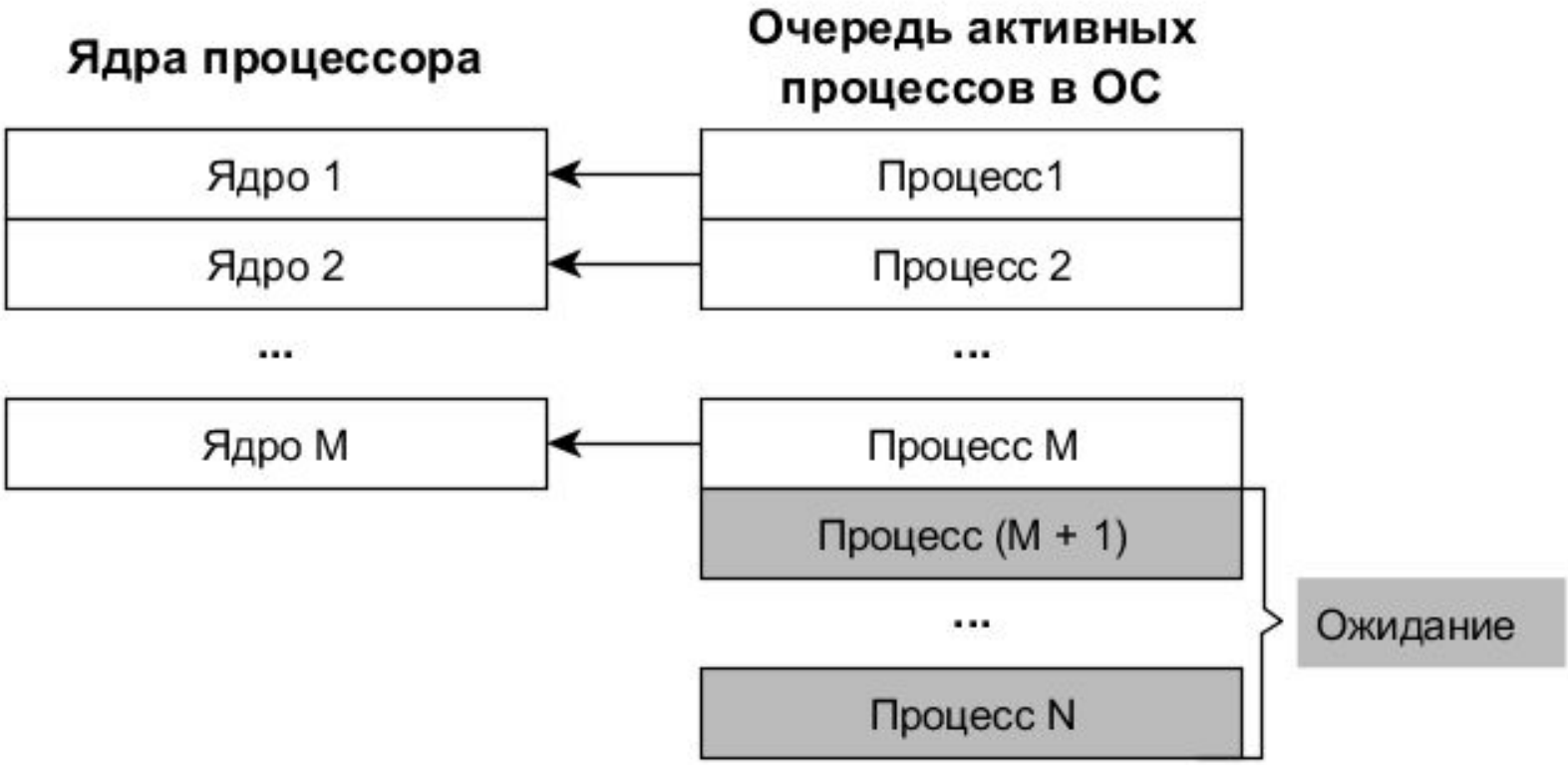
1. Независимость потоков команд, одновременно существующих в системе.
2. Несвязанность данных, обрабатываемых в одном потоке команд.

Пример несвязанных данных:

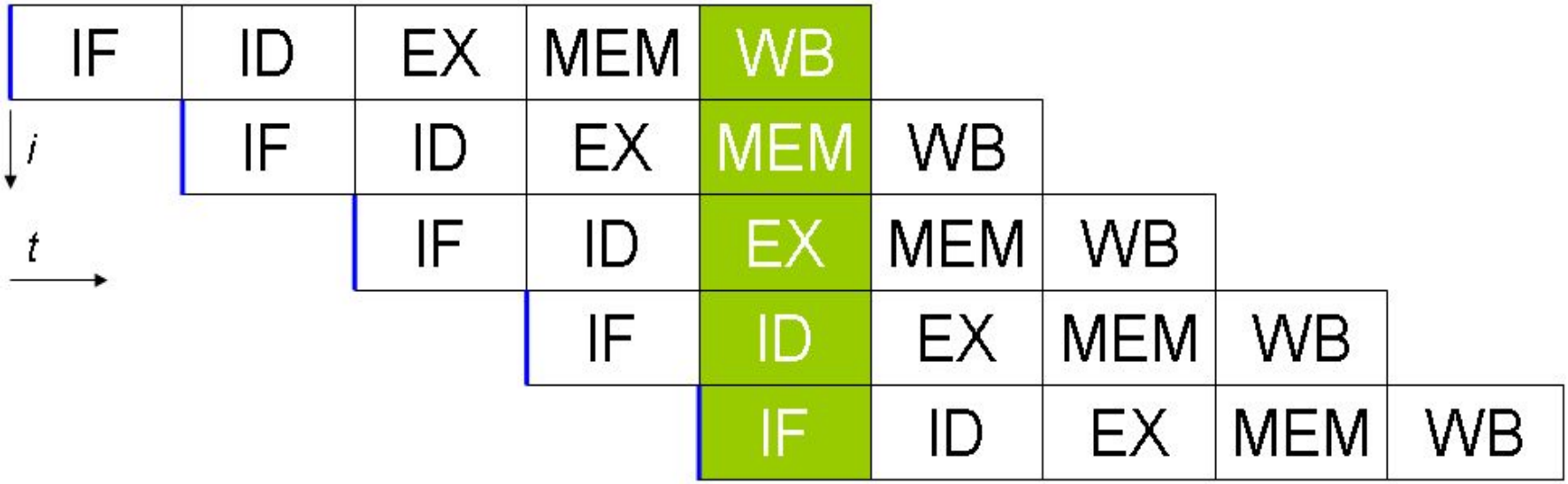
$A = B + C;$

$D = E \times F.$

Классификация уровней параллелизма, предложенная П. Треливеном.



Мультипроцессирование в ОС

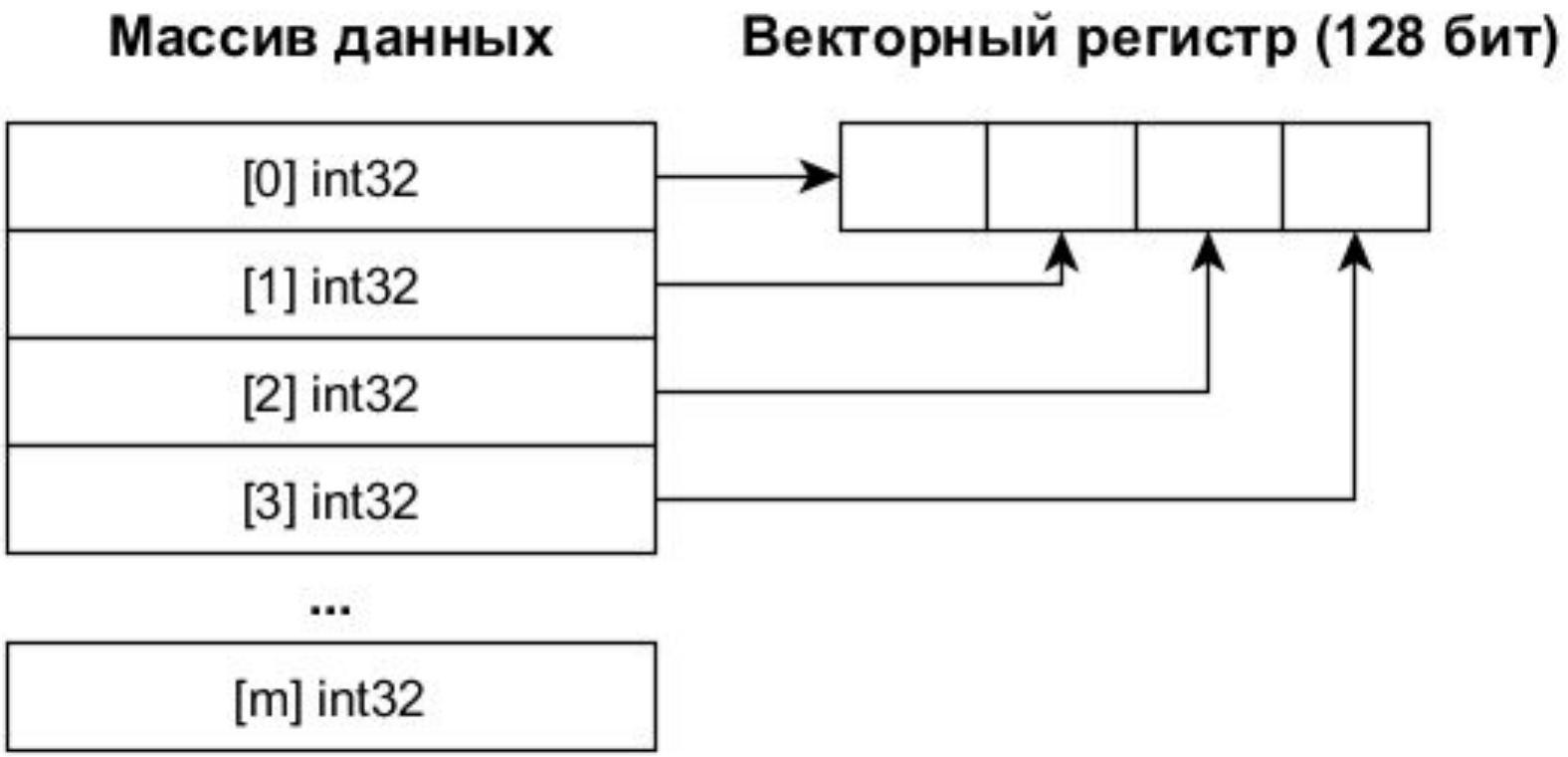


Простой пятиуровневый конвейер в RISC-процессорах

Принцип многофункциональной обработки

Самостоятельные арифметические устройства в составе центрального процессора (основные):

1. Сложитель.
2. Умножитель.
3. Делитель.
4. Устройство выполнения логических операций.
5. Устройство выполнения сдвиговых операций.



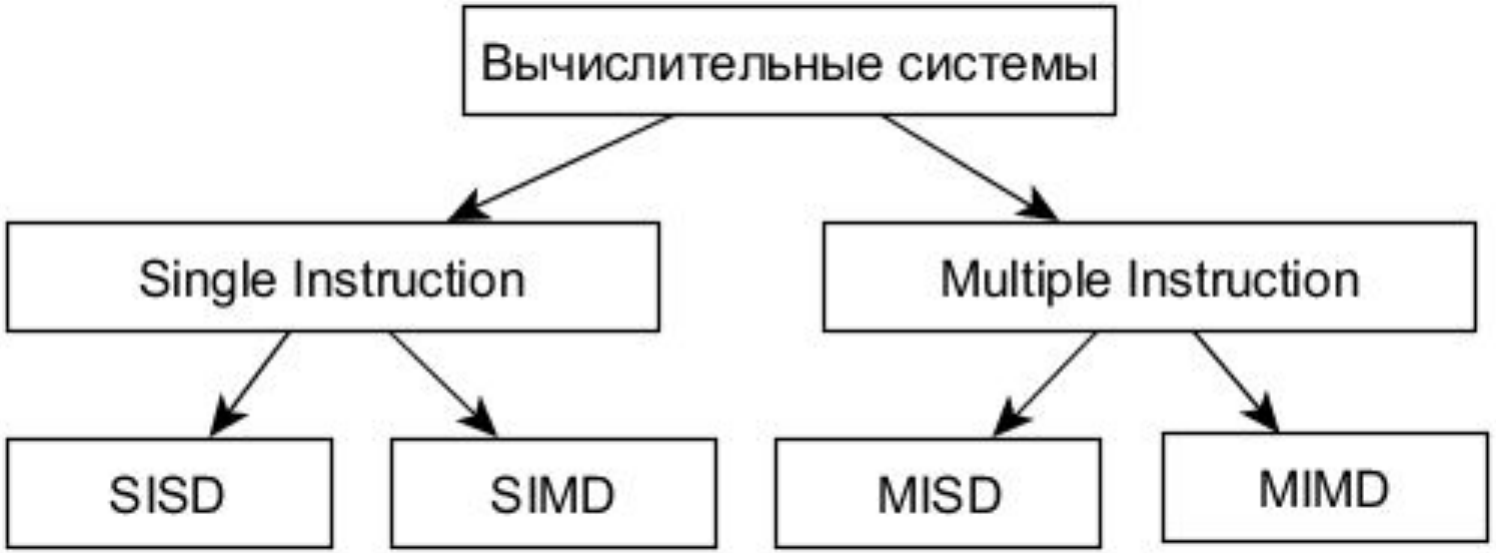
Векторная обработка данных

Intel Xeon Phi

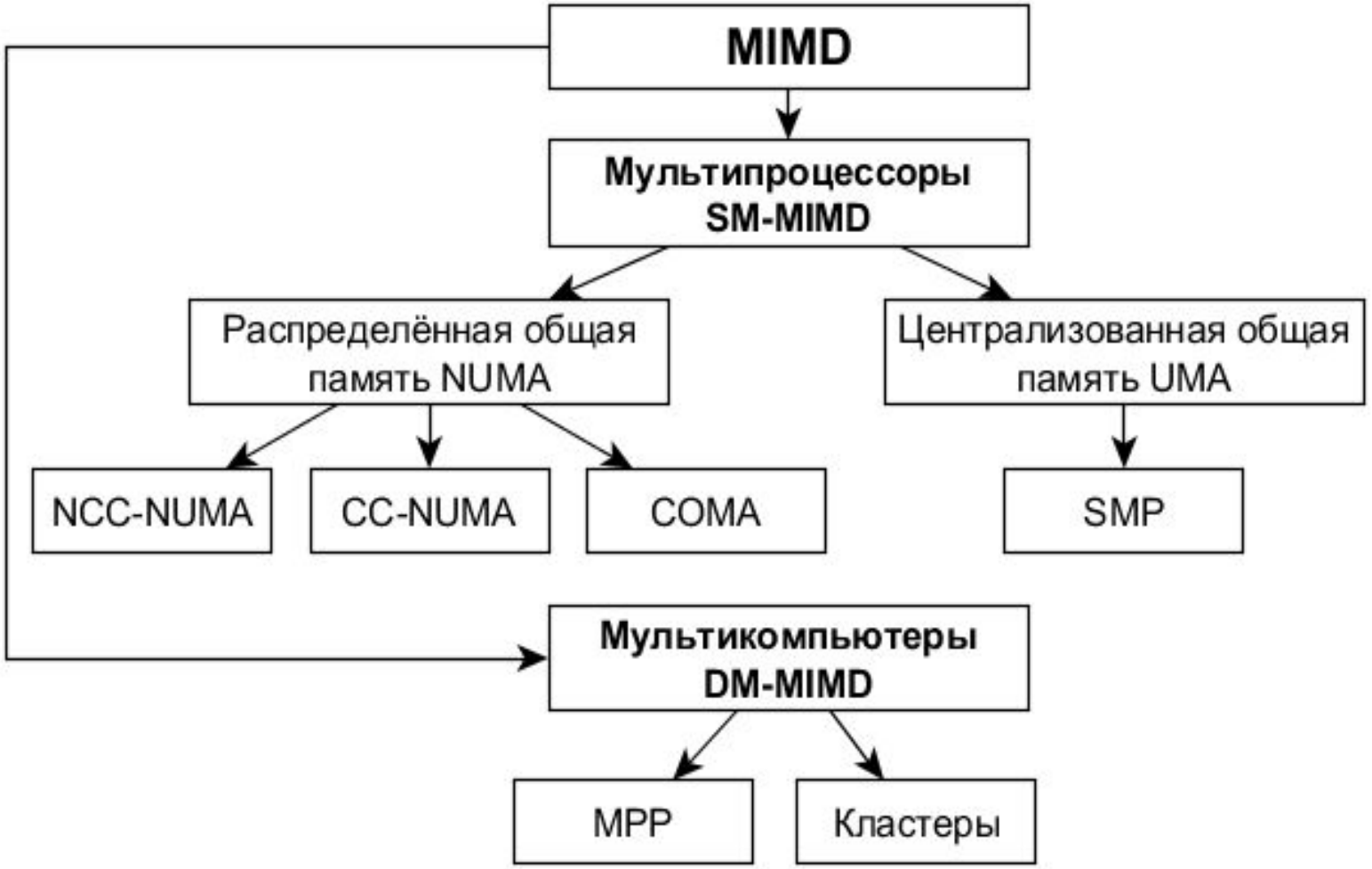
Процессор с ***512***-битными векторными регистрами.

Техпроцесс: 14 нм
Количество ядер: 72
Частота ядра: 1,5 ГГц
Объём кэш-памяти (K2): 36 Мб

II. Архитектура высокопроизводительных вычислительных систем



Классификация вычислительных систем Флинна



Классификация MIMD систем

COMA

Основные особенности:

1. Отсутствие ОП, наличие вместо неё большого кэша на каждом узле.
2. Адрес переменной не фиксирован на протяжении работы программы.
3. Выполнение копирования данных при доступе на чтение и выполнение перемещения при доступе на запись.

Виды вычислительных кластеров

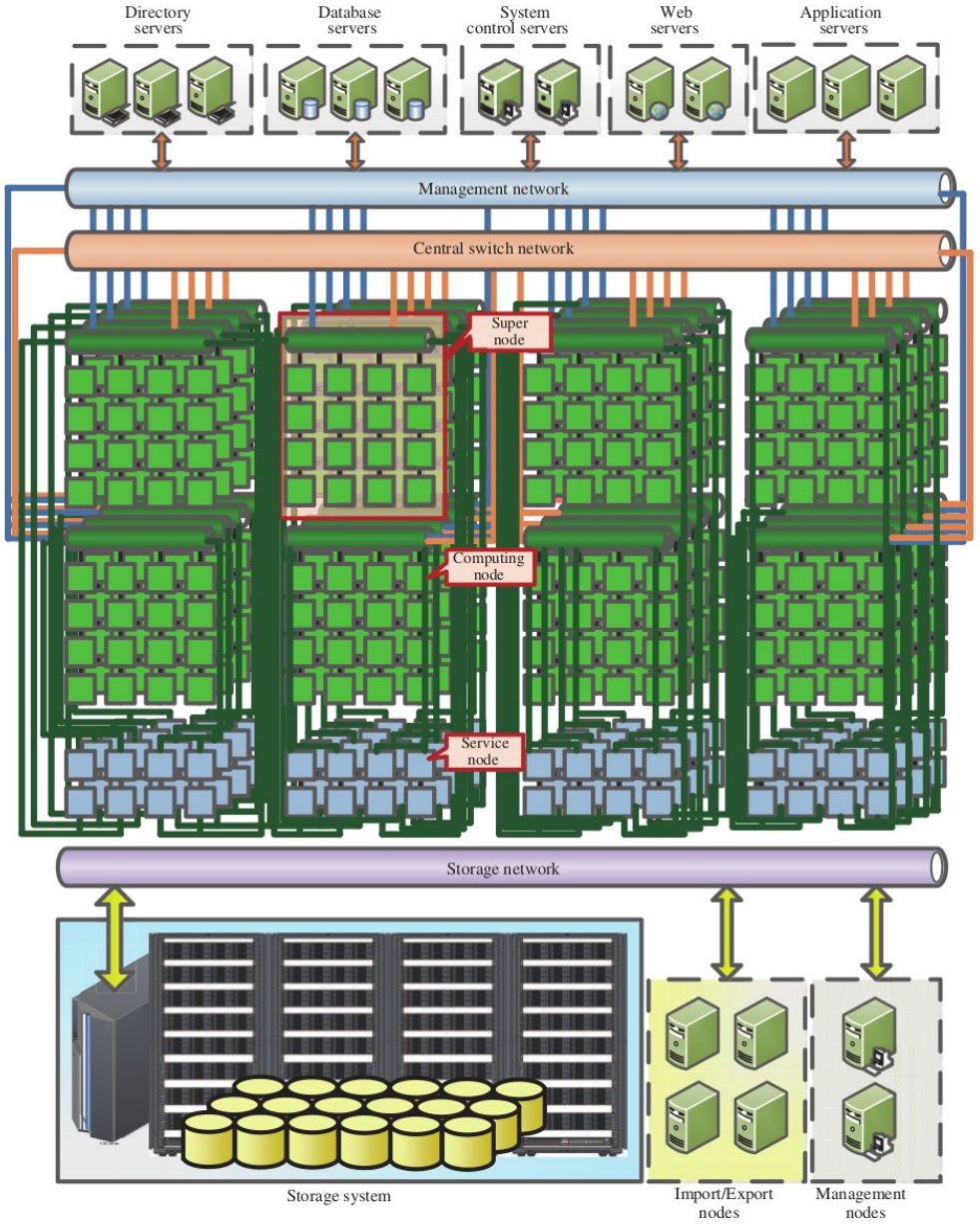
1. Кластеры, узлами которых являются ПК.
2. Кластеры, узлами которых являются мультипроцессоры.
3. Кластеры, включающие ПК и мультипроцессоры.

MIMD системы

- SM-MIMD (Shared Memory Multiple Instruction, Multiple Data)
- DM-MIMD (Distributed Memory Multiple Instruction, Multiple Data)
- UMA (Uniform Memory Access)
- NUMA (Non-Uniform Memory Access)
- CC-NUMA (Cache Coherent Non-Uniform Memory Access)
- COMA (Cache-Only Memory Architecture)
- NCC-NUMA (Non-Cache Coherent Non-Uniform Memory Access)
- MPP (Massively Parallel Processor)

Sunway TaihuLight

Пиковая теоретическая производительность: 125.4 Пфлопс
Производительность в соответствии с тестом LINPACK : 93 Пфлопс
Оперативная память 1.31 Пб
Количество ядер: 10649600
Потребляемая мощность 15 мВт



Архитектура суперкомпьютера Sunway TaihuLight

Узел суперкомпьютера Sunway TaihuLight

SW26010 – процессор китайской архитектуры
и производства. Содержит:

- 256 вычислительных ядер
- 4 ядра управления

Вычислительный кластер «СКИФ Cyberia»

Пиковая теоретическая производительность:	62,351 Тфлопс
Производительность в соответствии с тестом LINPACK :	47.88 Тфлопс
Оперативная память	360 Тб
Количество ядер:	5304
Потребляемая мощность	300 кВт

Узлы суперкомпьютера «СКИФ Cyberia»

- 282 узла/564 двухъядерных процессора Intel Xeon 5150, 2,66ГГц (Woodcrest)/8Gb RAM

- 190 узлов/360 шестиядерных процессоров IntelXeon 5670, 2,93ГГц (Westmere)/24Gb RAM (T-Blade 1.1)

- 40 узлов/80 шестиядерных процессоров IntelXeon 5670, 2,93ГГц (Westmere)/48Gb RAM (T-Blade 1.1)

- 128 узлов/256 шестиядерных процессоров IntelXeon 5670, 2,93ГГц (Westmere)/24Gb RAM (T-Blade 2)

III. Грид-системы

Грид-система (***grid***) представляет собой программно-аппаратный комплекс, построенный на основе кластерного вычислителя.
Грид-системы ещё называют метакомпьютерами или «виртуальнымм суперкомпьютерами».

Классификация грид-систем

С точки зрения выделения вычислительных ресурсов грид-системы классифицируют следующим образом:

- Добровольные
- Научные
- Коммерческие

Berkeley Open Infrastructure for Network Computing (BOINC)

средняя производительность > 130 терафлопс

количество участников 3 млн.

IV. Облачные технологии

Суть ***облачных технологий*** (облачных вычислений) состоит в предоставлении программных и виртуализированных аппаратных ресурсов в качестве сервиса

Свойства облачных технологий

- Самообслуживание по требованию
- Универсальный доступ по сети
- Объединение ресурсов
- Быстрая эластичность
- Учёт потребления

Классификация облачных сервисов по типу ресурса

- SaaS (Software as a Service)
- PaaS (Platform as a Service)
- IaaS (Infrastructure as a Service)
- DaaS (Desktop as a Service, Data as a Service)
- CaaS (Communications as a Service)

Модели развёртывания облачных систем

- Частное облако
- Публичное облако
- Общественное облако
- Гибридное облако

MapReduce

Функция высшего порядка – в программировании функция, принимающая в качестве аргументов другие функции или возвращающая другую функцию в качестве результата.

Технология MapReduce основана на использовании двух функций высшего порядка – ***map*()** и ***reduce*()**.

MapReduce

- ***Map*** – функция высшего порядка, которая применяет переданную в качестве аргумента функцию к каждому элементу списка, переданного в качестве другого аргумента. Map возвращает список, элементом которого является результат выполнения функции-аргумента.
- ***Reduce (свёртка)*** – функция высшего порядка, которая производит преобразование структуры данных к единственному атомарному значения при помощи заданной функции.

MapReduce

Шаг 1. Подготовка входных данных для функции *map*(). Каждый узел получает данные, соответствующие ключу K_i.

Шаг 2. Выполнение пользовательской функции, переданной в функцию *map*(). Функция *map*() выполняется единожды для каждого ключа K_i: $T_i = map(K_i)$

Шаг 3. Распределение T_i по reduce-узлам.

Шаг 4. Выполнение пользовательской функции, переданной в функцию *reduce*(). Функция *reduce*() выполняется единожды для каждого значения T_i: $R_i = reduce(Ti)$

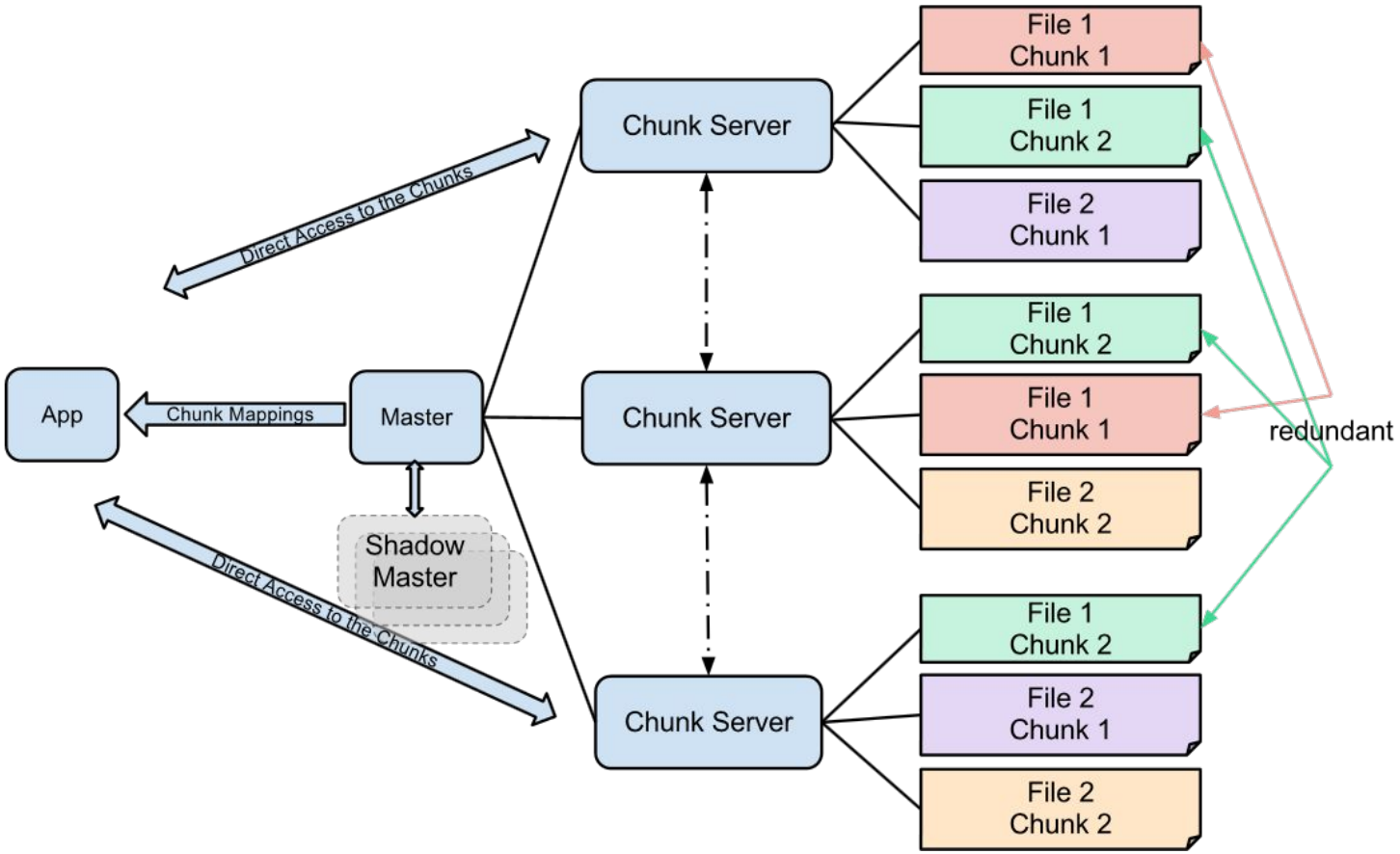
Шаг 5. Вычисление конечного результата.

Распределённые файловые системы

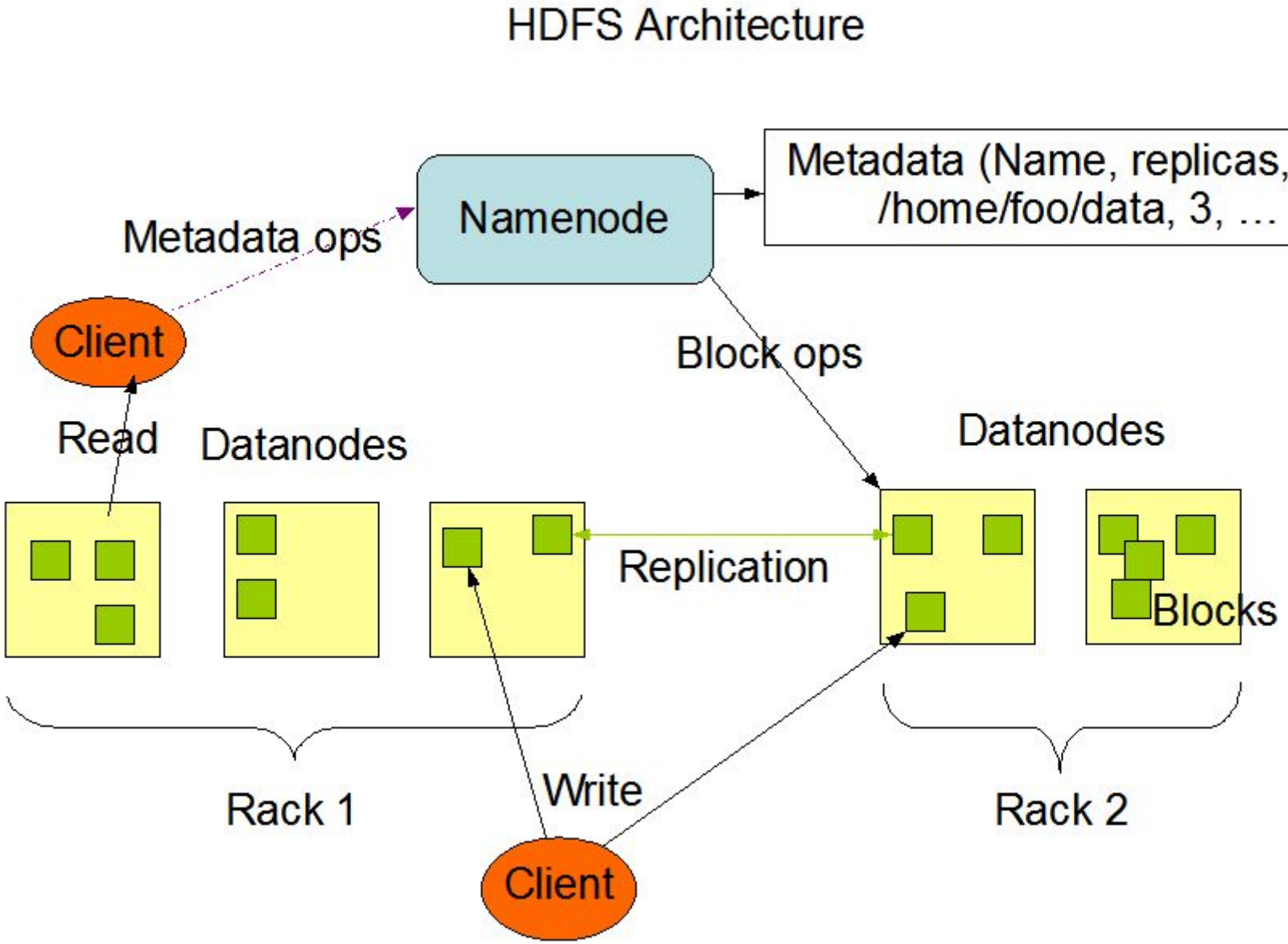
Распределённая файловая система (РФС) – это клиент-серверное приложение, которое позволяет клиенту хранить и обращаться к данным, сохранённым на сервере так, как если бы эти данные хранились локально на клиентской стороне.

Распределённые файловые системы

- РФС отличается от ***распределённого хранилища данных*** тем, что для доступа к распределённым данным первая использует тот же интерфейс, что и для доступа к локальным данным.



Принцип работы GoogleFS

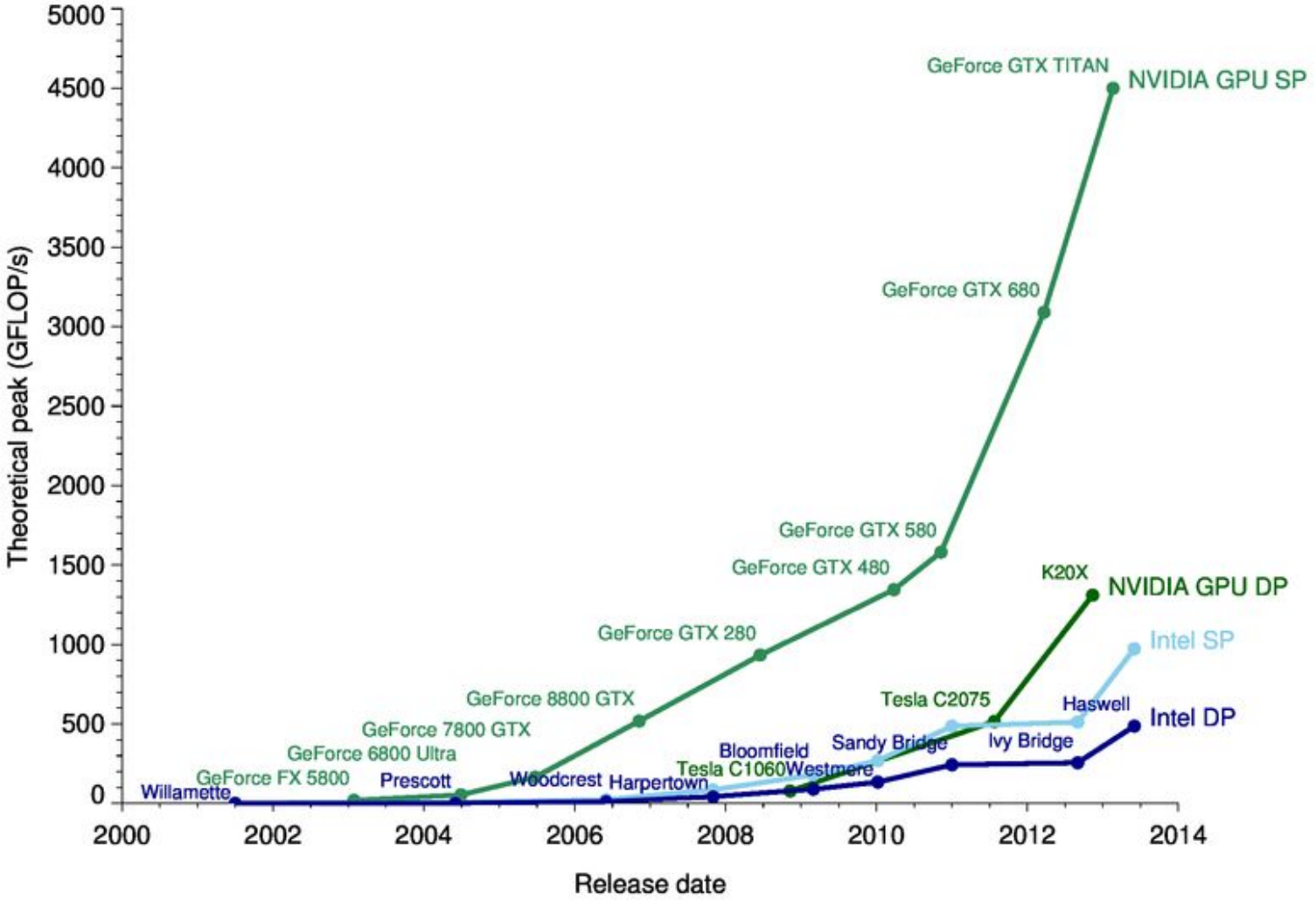


Принцип работы HDFS

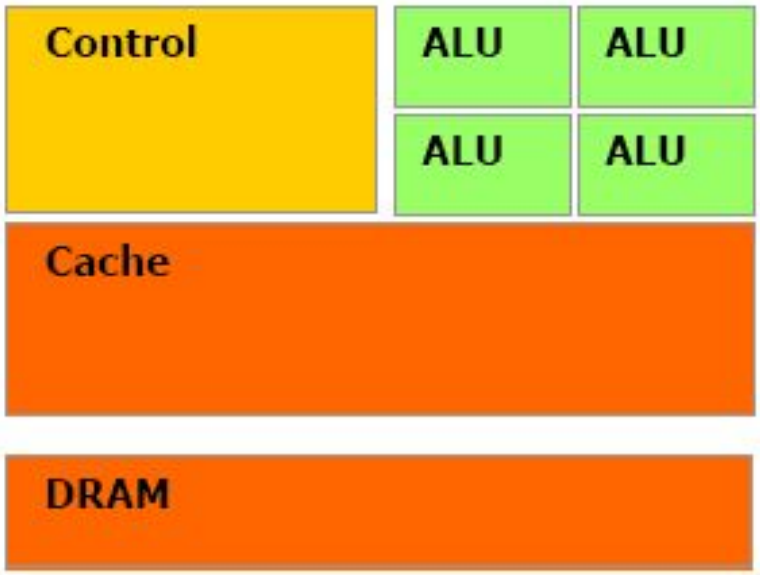
V. GPGPU
Общие вычисления на видеокарте

Определение

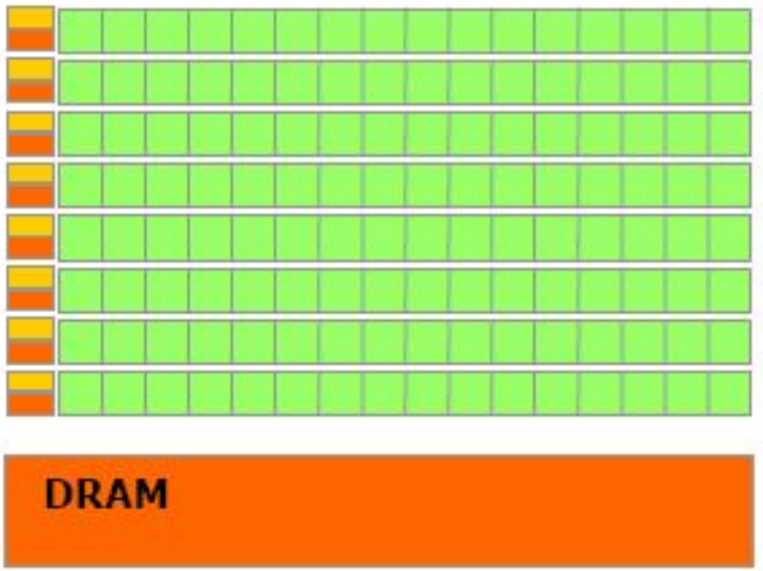
- ***GPGPU*** (General-Purpose computation on Graphics Processing Units – универсальные вычисления на видеокарте) – направление информатики, посвящённое способам использования вычислительных ресурсов видеокарты для решения задач, не связанных напрямую с визуализацией.



Сравнение производительности ЦП и видеокарты



CPU

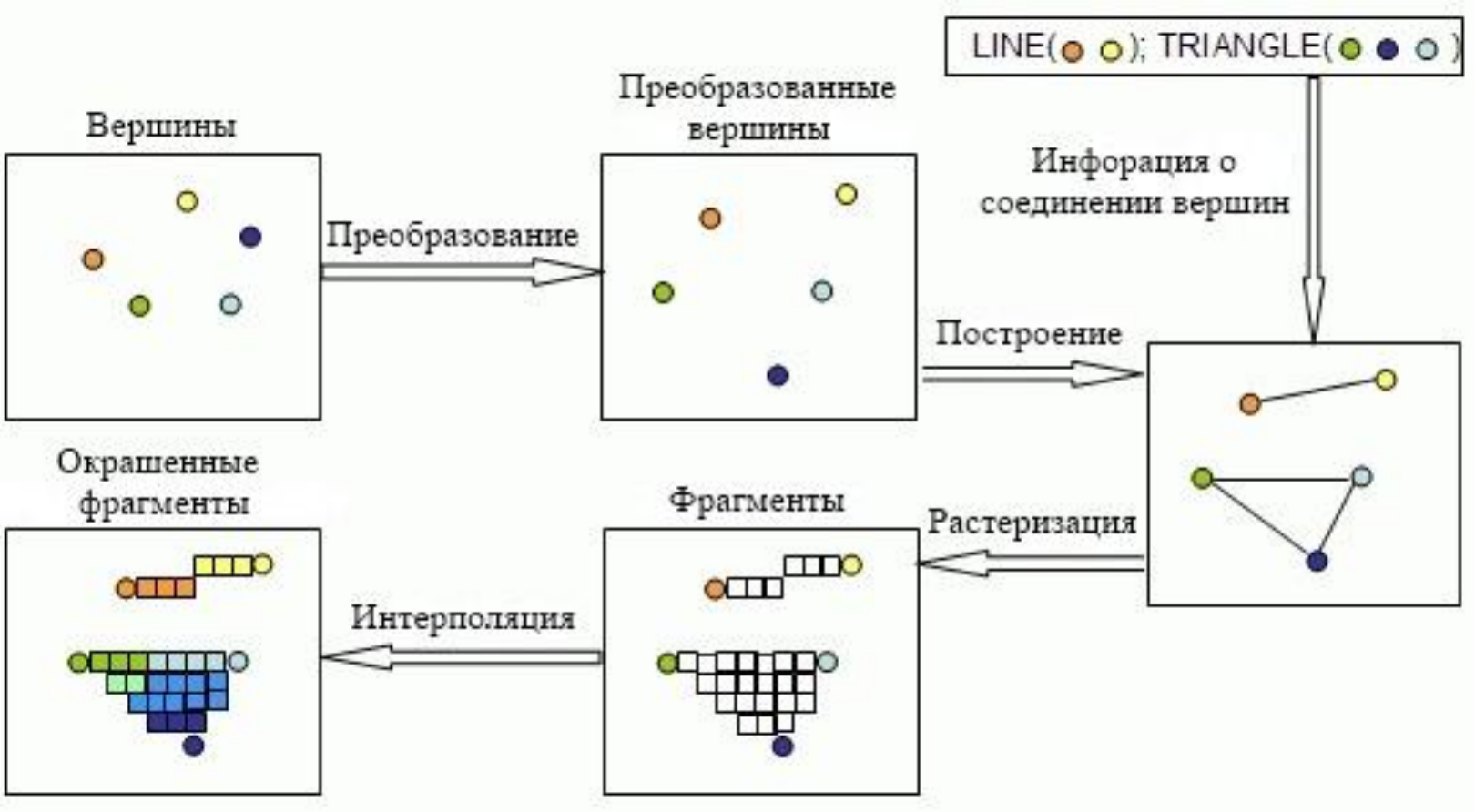


GPU

Сравнение архитектуры ЦП и видеокарты

Memory type	Memory Bandwidth (GB/sec)
DDR4-3200	**25**
GDDR5X	**484**

Укрупнённая схема графического конвейера



Пример работы графического конвейера

Пиксельные шейдеры

Программы, написанными на си-подобном языке программирования (например, High Level Shader Language - высокоуровневый язык шейдеров), и выполняются на процессоре видеокарты без участия центрального процессора.

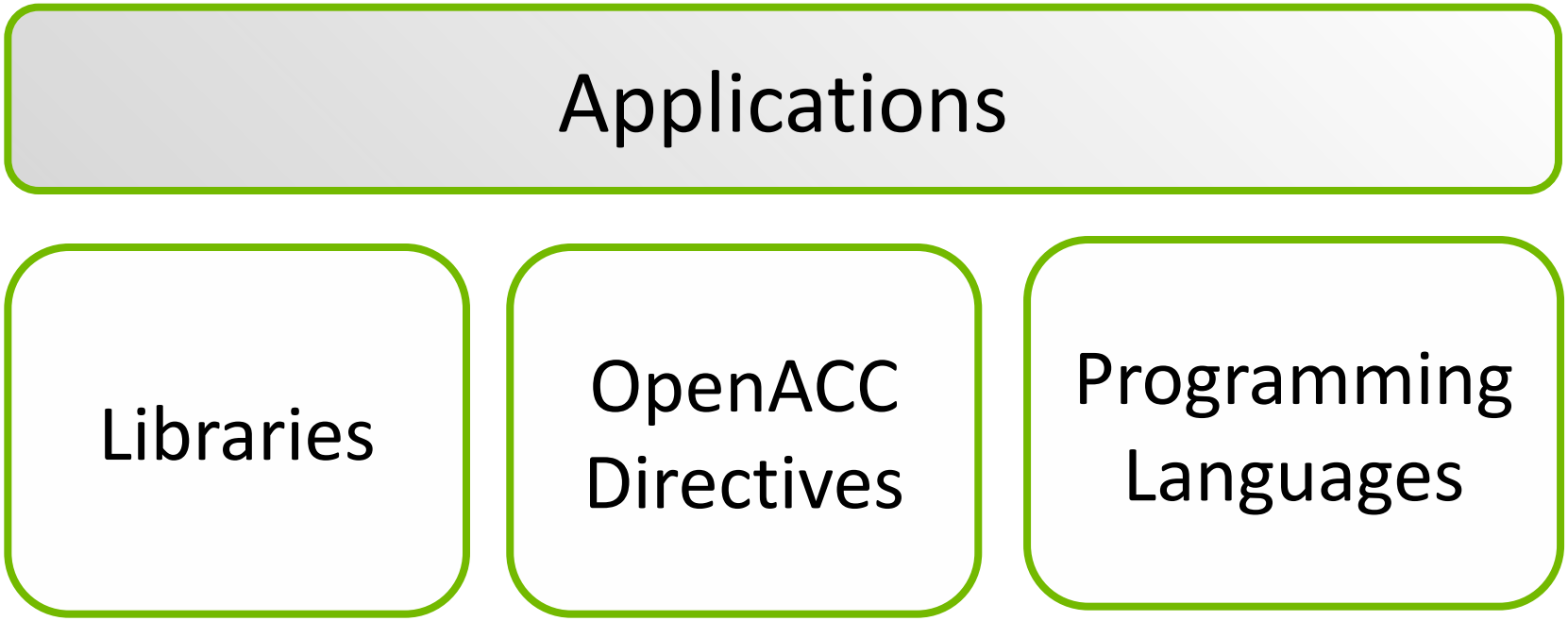
Недостатки:

1. Способ распараллеливания жёстко фиксирован - шейдер выполняется один раз для каждого пикселя результирующей текстуры, причём предполагается, что изменяться будут только те байты результирующей текстуры, которые соответствуют этому пикселю.
2. Существует ряд ограничений на формат результирующей текстуры в пиксельных шейдерах. Например, при использовании пиксельных шейдеров совместно с DirectX9.0c не поддерживается однобитовый формат.

Программный интерфейс	**Универсальность относительно типа ускорителя**	**Объём требуемых изменений в программе для внедрения**
Nvidia CUDA	Видеокарты Nvidia	Большой
Open Computing Language (OpenCL)	Да	Большой
DirectCompute	Да	Большой
Open Accelerators (OpenACC)	Да	Средний
Microsoft C++ Accelerated Massive Parallelism (C++ AMP)	Да	Средний

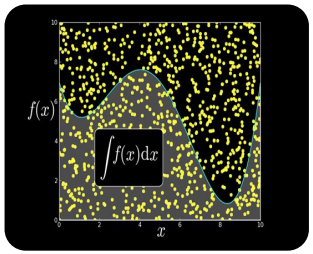
Основные программные интерфейсы для доступа к вычислительным ресурсам видеокарты

3 Ways to Accelerate Applications (From Nvidia)

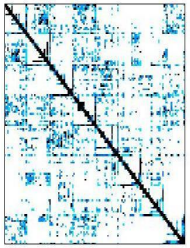


Some GPU-accelerated Libraries

NVIDIA cuBLAS



NVIDIA cuRAND



NVIDIA cuSPARSE

NVIDIA NPP

GPU VSIPL

Vector Signal Image Processing

CULA tools

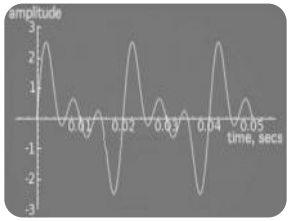
GPU Accelerated Linear Algebra

MAGMA

ICL UT

Matrix Algebra on GPU and Multicore

open source initiative



NVIDIA cuFFT

ROGUE WAVE SOFTWARE

IMSL Library

ArrayFire Matrix Computations

CUSP

Sparse Linear Algebra

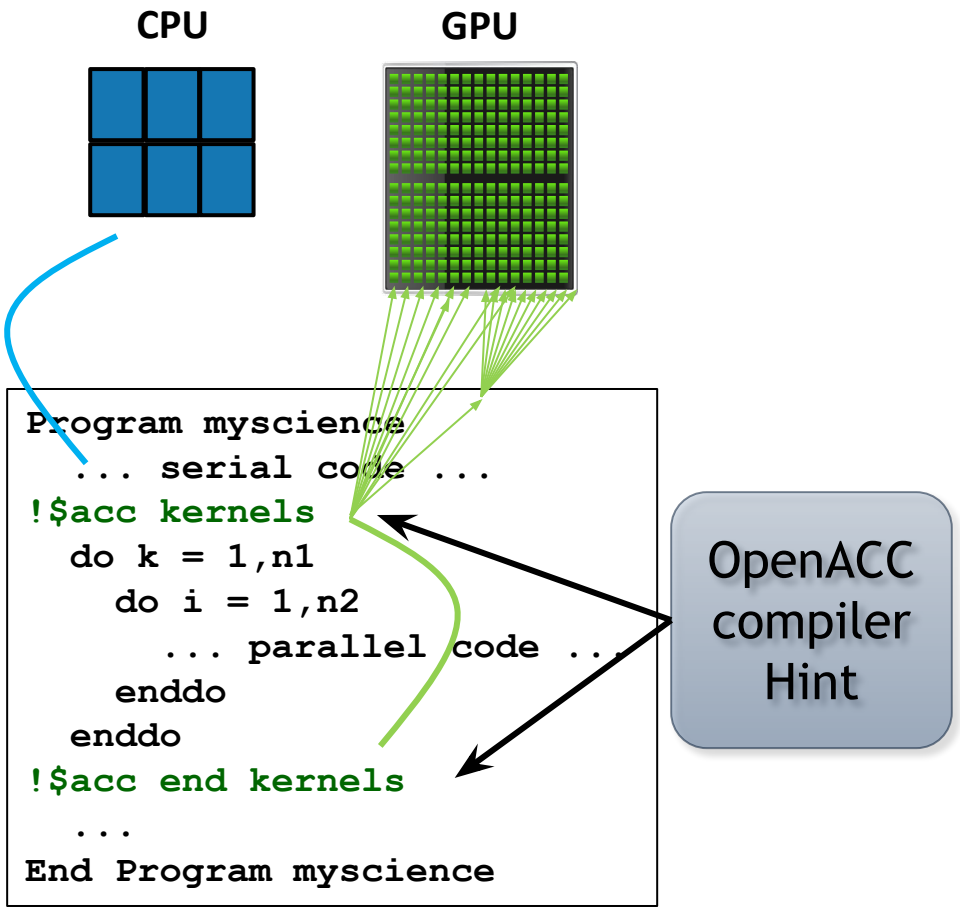
open source initiative

Thrust

C++ STL Features for CUDA

open source initiative

OpenACC Directives



Simple Compiler hints

Compiler Parallelizes code

Works on many-core GPUs & multicore CPUs

```
#pragma acc data copy(A) create(Anew)
while ( error > tol  &&  iter  <  iter_max )  {
  error = 0.0;
  #pragma acc kernels
  {
  for (  int  j = 1; j < n-1;  j++ )  {
    for (  int  i = 1; i < m-1; i++ )  {
       Anew [j] [i] = 0.25 * ( A [j] [i+1] + A [j] [i-1] +
                                    A [j-1] [i] + A [j+1] [i];
       error = fmax ( error, fabs (Anew [j] [i] - A [j] [i];
    }
  }

  for (  int j = 1; j < n-1; j++) {
     for (int = i; i < m-1; i++ )  {
       A [j] [i] = Anew [j] [i];
     }
    }
  }

   if (iter % 100 == 0) printf ("%5d, %0.6f\n", iter, error);
   iter++;
}
```

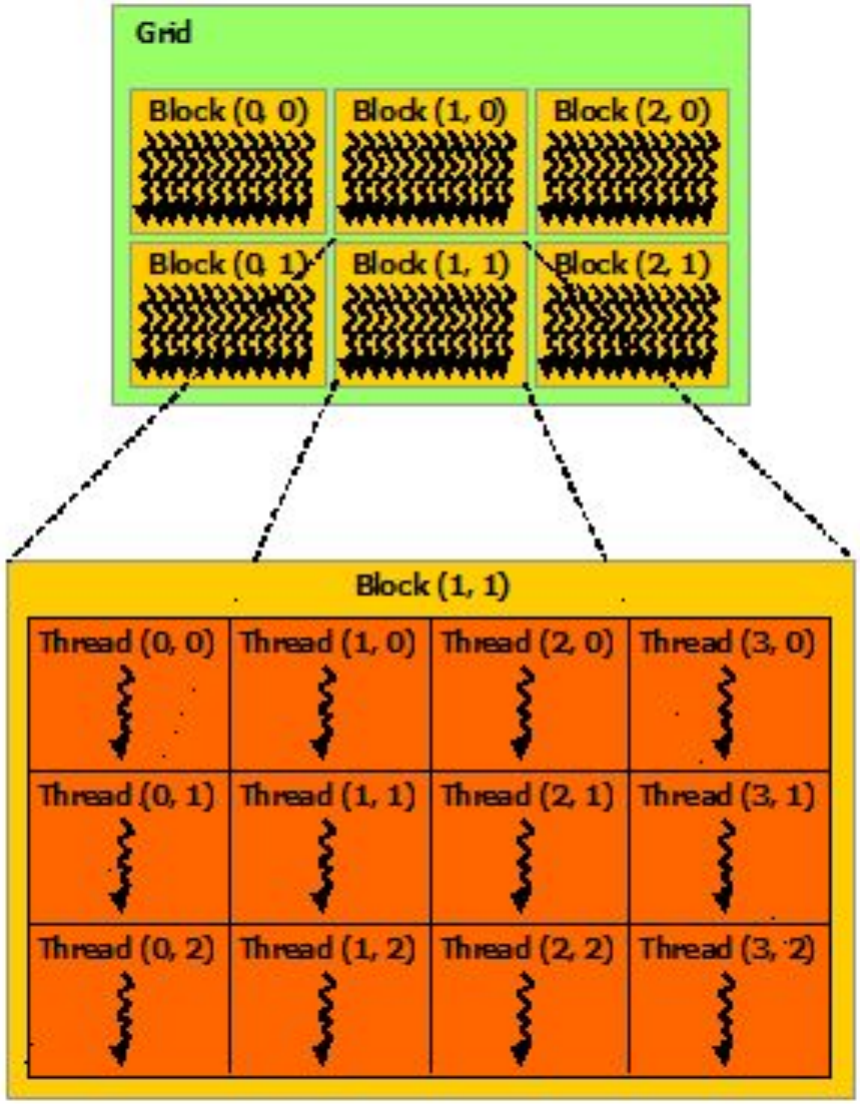
Использование директив OpenACC для распараллеливания метода Якоби

Метод Якоби – метод из численной линейной алгебры для решения систем линейных уравнений.

Nvidia CUDA

- **__global__** спецификатор *ядра* (*kernel*) – функции выполняемой N раз N различными потоками.
- **threadIdx** встроенная переменная, хранящая идентификатор потока.

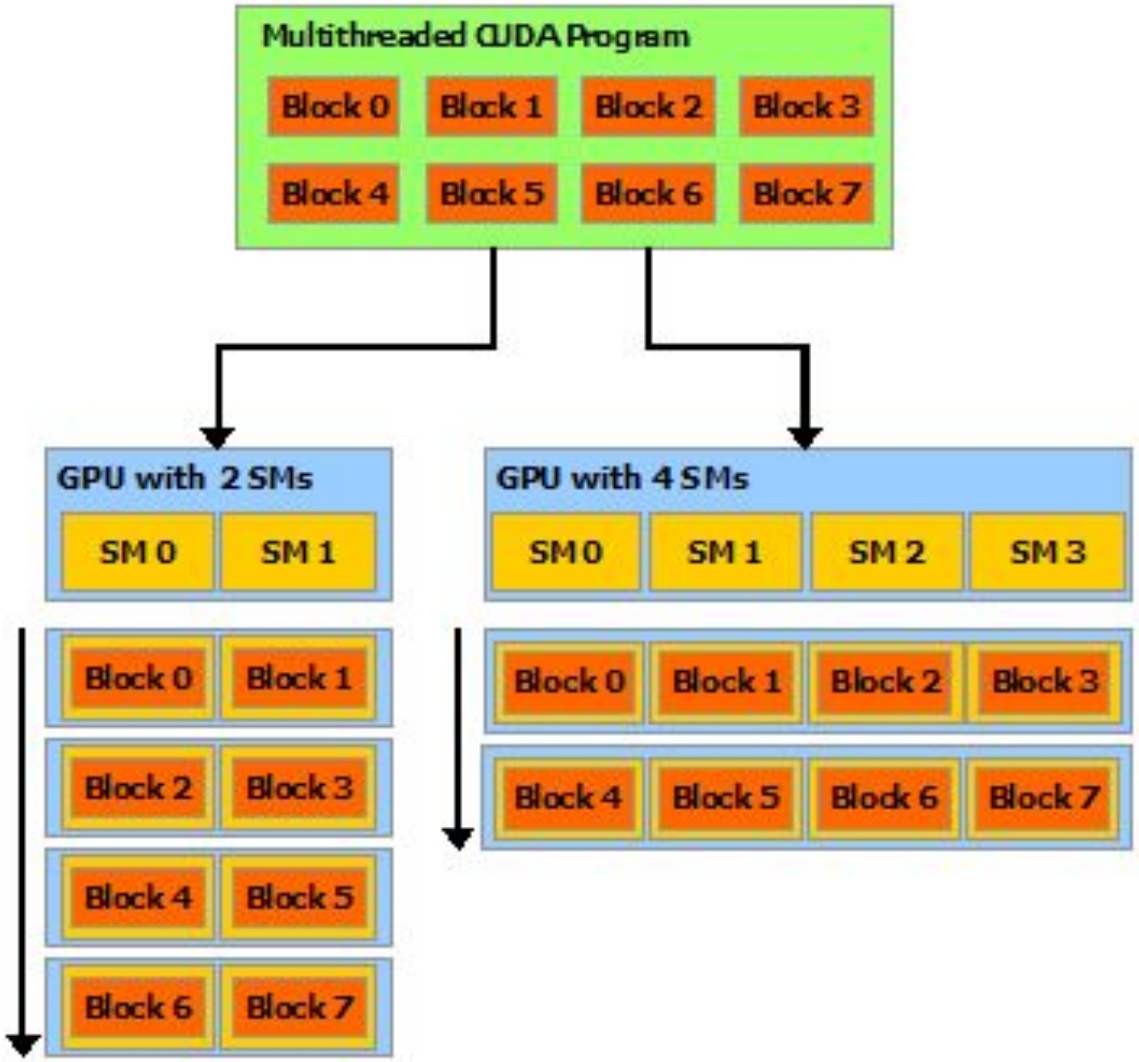
```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x; C[i] = A[i] + B[i];
}

int main() {
     ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C); ...
}
```



Иерархия потоков

Синхронизация потоков в блоке

- __syncthreads() работает как барьер, который поток может пересечь, только когда все потоки в блоке дойдут до этой точки.



Аппаратная
реализация

Архитектура SIMT (Single-Instruction, Multiple-Thread)

- warp группа из 32 потоков, исполняющих одну инструкцию в один момент времени.

№	Тип памяти	Доступ CPU	Доступ GPU	Уровень выделения	Скорость работы
1	Регистры	Нет	R/W	per-thread	высокая
2	Локальная	Нет	R/W	per-thread	низкая
3	Глобальная	R/W	R/W	per-grid	низкая
4	Разделяемая	Нет	R/W	per-block	высокая
5	Константная	R/W	R/O	per-grid	высокая
6	Текстурная	R/W	R/O	per-grid	высокая

Типы памяти в технологии CUDA

Nvidia CUDA SDK

- Расширенный язык C
- Компилятор nvcc
- Отладчик gdb для GPU
- Профайлер

Профилирование — сбор характеристик работы программы, таких как время выполнения отдельных фрагментов, число верно предсказанных условных переходов, число кэш-промахов и т. д.

Использование n блоков

```
__global__ void calculate(float* A, float* B, float* C, int n) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < n) {
        // Process element with index i
    }
}

const int MAX_BLOCKS_DIM_X_PER_GRID = 65535;
const int BLOCK_SIZE = 64;

int getBlocksCount(int size) {
    int result = size / BLOCK_SIZE + ((size % BLOCK_SIZE) && 1);
    return min(result, MAX_BLOCKS_DIM_X_PER_GRID);
}
```

Что делать, если элементов данных больше, чем можно создать потоков?

1. Обработка одним потоком m последовательных элементов

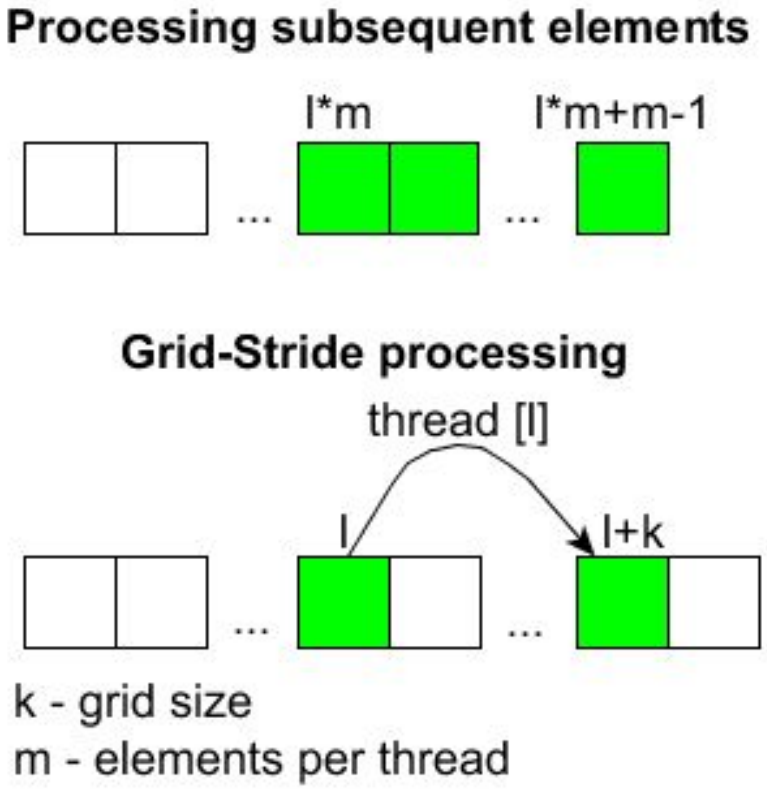
```
__global__ void calculateSubsequent(float* A, float* B, float* C, int n, int
elementsPerThread) {
    int startIndex = (blockIdx.x * blockDim.x + threadIdx.x) * elementsPerThread;
    int max = min(startIndex + elementsPerThread, n);
    for (int i = startIndex; i < max; i++) {
       // Process element with index i
    }
}

int getBlocksCount(int size, int elementsPerThread) {
    int normalizedSize = size / elementsPerThread + ((size % elementsPerThread) && 1);
    int result = normalizedSize / BLOCK_SIZE + ((normalizedSize % BLOCK_SIZE) && 1);
    return min(result, MAX_BLOCKS_DIM_X_PER_GRID);
}
```

Что делать, если элементов данных больше, чем можно создать потоков?

2. Обработка потоком каждого (I + k)-того элемента, где I – глобальный индекс потока, k – размер сетки (grid)

```
__global__ void calculateSubsequent(float* A, float* B, float* C, int n) {
    for (int i = blockIdx.x * blockDim.x + threadIdx.x; i < n; i += blockDim.x * gridDim.x) {
        // Process element with index i
    }
}
```

Сравнение подходов обработки структур данных большого размера



Свёртка условных переходов

```
if (difference != 0) {
      output |= byteMask;
}
```

```
output |= (difference && 1) * byteMask;
```

Результатом выполнения операции difference && 1 будет 1, если разница difference не равна 0, и 0, если разница difference равна 0. В маске byteMask единице равен только один бит (соответствующий текущему биту в байте результата). Таким образом, если цвета сравниваемых пикселей равны (разница difference равна 0), то текущий бит в байте результата output останется равным 0, иначе будет установлен в 1.

GEFORCE GTX 1080

GPU Engine Specs:

- NVIDIA CUDA® Cores 2560
- Base Clock (MHz) 1607
- Boost Clock (MHz) 1733

Memory Specs:

- Memory Speed 10 Gbps
- Standard Memory Config 8 GB GDDR5X
- Memory Interface Width 256-bit
- Memory Bandwidth (GB/sec) 320

- Graphics Card Power (W) 180 W

Intel® Core™ i7-6950X Processor Extreme Edition

# of Cores	10
# of Threads	20
Processor Base Frequency	3 GHz
Max Turbo Frequency	3.5 GHz
Cache	25 MB
Intel® Turbo Boost Max Technology 3.0 Frequency ‡	4 GHz
Power in Turbo Boost mode	140 W

Дополнительная информация

- Общая документация http://docs.nvidia.com/cuda/cuda-c-programming-guide
- Первая лабораторная работа https://nvidia.qwiklab.com
- Как сделать программу более гибкой относительно размера входных данных https://devblogs.nvidia.com/parallelforall/cuda-pro-tip-write-flexible-kernels-grid-stride-loops/

Вопросы по V главе

1. В чём отличие архитектуры современных видеокарт от архитектуры центрального процессора?
2. Какие характеристики современных видеокарт позволяют использовать их для общих вычислений?
3. Расскажите о принципе функционирования графического конвейера.
4. Какая технология GPGPU была исторически первой, назовите её плюсы и минусы.
5. Назовите основные программные интерфейсы для доступа к вычислительным ресурсам видеокарты, дайте им характеристику по универсальности относительно типа ускорителя и уровню сложности внедрения в существующую программу.
6. Расскажите о Open ACC.
7. Как происходит выполнение CUDA-программы?
8. Что такое warp (ворп)?
9. Назовите типы памяти в технологии CUDA, дайте характеристику каждому из них.
10. К какому типу вы бы отнесли видеокарту с поддержкой технологии CUDA в классификации MIMD-систем?

VI. Программирование для высокопроизводительных вычислений

Проблемы параллельного программирования

- Равномерная загрузка процессоров / узлов (*балансировка*)
- Обмен информацией между процессорами
  - Минимизация объёма данных, которыми обмениваются узлы.
  - Повышение эффективности такого обмена.

Методология организации параллельных вычислений для SIMD архитектуры

1) Выявление ресурсоёмких и вычислительно сложных частей программы.

2) Анализ возможности обособления этих частей программы для дальнейшего распараллеливания.

3) Распараллеливание обособленных на шаге (2) частей программы путём организации конвейерной или векторной обработки данных.

Методология организации параллельных вычислений для MIMD архитектуры

1) Разделение вычислений на независимые части

2) Выделение информационных зависимостей

3) Масштабирование задач

4) Распределение подзадач между процессорами

Показатели качества параллельных методов

- Ускорение (speedup)

  $S_p(n)=T_1(n)/T_p(n)$
- Эффективность (efficiency)

  $E_p(n)=T_1(n)/(pT_p(n))=S_p(n)/p$
- Стоимость (cost)

  $C_p=pT_p(n)$
- Масштабируемость (scalability) вычислений
  - сильная
  - слабая

Библиотеки для обмена сообщениями

- MPI (Message Passing Interface)
- PVM (Parallel Virtual Machines)

Предназначены для вычислительных систем с распределённой памятью

MPI (Message Passing Interface)

Существуют бесплатные и коммерческие реализации почти для всех суперкомпьютерных платформ, а также для сетей рабочих станций

MPI

- Обмены типа точка-точка
- Коллективные обмены
  - Барьерная синхронизация
  - Передача от одного узла всем в группе
  - Передача от всех узлов в группе одному
- и многое другое, всего более **500** функций

OpenMP

- Интерфейс OpenMP задуман как стандарт для программирования на масштабируемых SMP-системах (SSMP,ccNUMA, etc.) в модели общей памяти (shared memory model).
- В стандарт OpenMP входят спецификации набора директив компилятора, процедур и переменных среды.

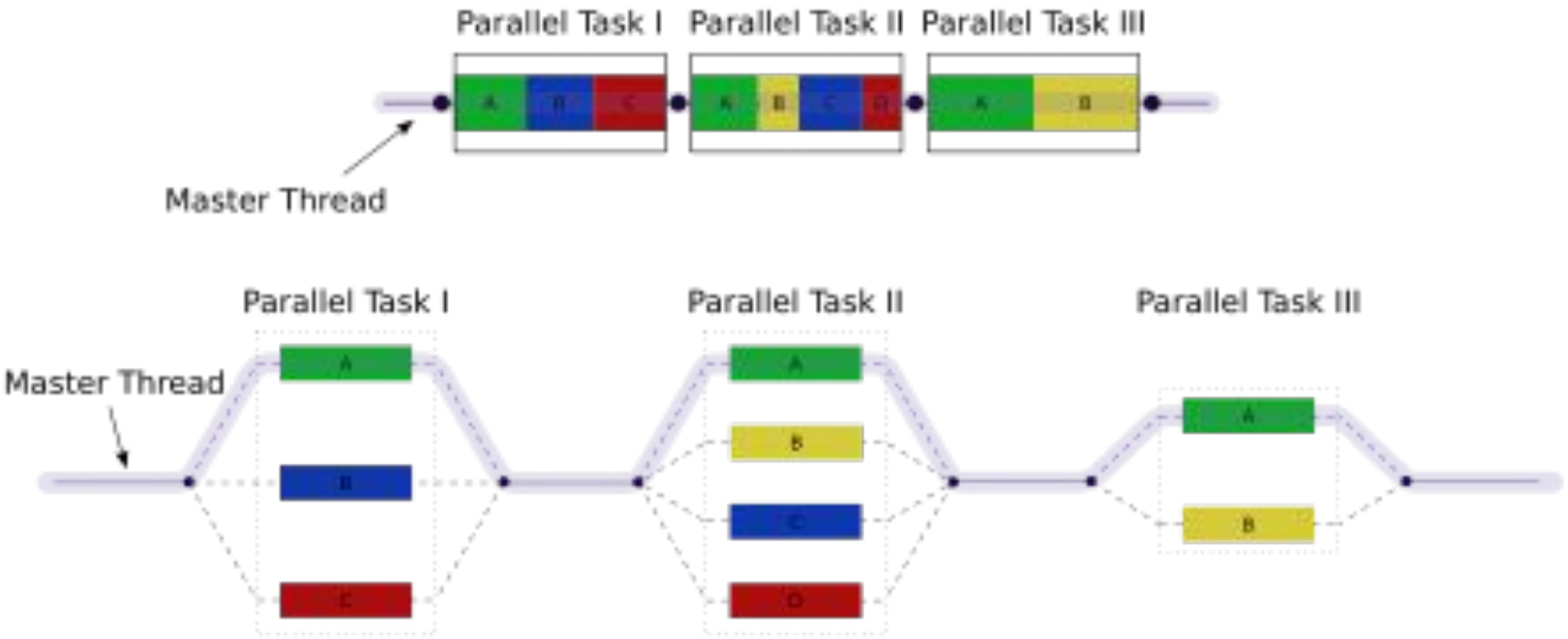
Почему не использовать MPI для вычислителей с общей памятью?

Модель передачи сообщений

- недостаточно эффективна на SMP-системах
- относительно сложна в освоении, так как требует мышления в "невычислительных" терминах

Преимущества OpenMP

- «Инкрементального распараллеливание»
- OpenMP - достаточно гибкий механизм
- OpenMP-программа на однопроцессорной платформе может быть использована в качестве последовательной программы механизм



Принцип параллельной обработки данных в OpenMP

Пример программы с использованием OpenMP

```
#pragma omp parallel
{
    #pragma omp for
    for(int n = 0; n < 10; ++n)
    {
      printf(" %d", n);
    }
    printf(".");
}
```

Результат: **0 5 6 7 1 8 2 3 4 9.**

Код, преобразованный компилятором

```
int this_thread = omp_get_thread_num(), num_threads = omp_get_num_threads();
int my_start = (this_thread) * 10 / num_threads;
int my_end   = (this_thread+1) * 10 / num_threads;
for(int n = my_start; n < my_end; ++n)
{
    printf(" %d", n);
}
```

Лабораторные работы

Ознакомление

1. https://nvidia.qwiklab.com Бесплатные занятия -> Introduction to Accelerated Computing.
2. Программа, вычитающая вектора.
   a. Константно заданный вектор (как в примере).
   b. Вектор произвольного размера (тестировать хотя бы до 1 млн. элементов). Размер вводится в качестве параметра во время работы программы.

Полномасштабные задания

1. Количество вхождений каждого символа в тексте. Текст подгружается из файла. Каждый символ кодируется одним байтом.
2. RLE. Можно реализовывать неоптимальным образом, например, использовать байтовый формат. С помощью видеокарты требуется ускорить только кодирование. Декодирование может быть реализовано на ЦП.
3. Замена цветов на их коды. Входные данные: изображение в формате RGB24, где заведомо количество цветов не превышает порога в 256 значений. Нужно на ЦП определить, какие есть цвета, составить палитру.
   b. Сравнить с использованием константной памяти и без неё.
   c. Перенести этап формирования палитры на видеокарту.

Для каждой лабораторной нужно будет сравнить с ЦП-реализацией по скорости выполнения и полученный результат, чтобы подтвердить корректность GPU-ускоренной реализации.