Московский государственный университет им. М.В. Ломоносова

Параллельное программирование для многопроцессорных систем с общей и распределенной памятью

Разработчики:
В.И. Лаева, e-mail: lvi@math.tsu.ru
А.А. Трунов, e-mail: trunov@math.tsu.ru
Томский государственный университет

Направление 010400.62
«Прикладная математика и информатика»

Проект комиссии Президента по модернизации и техническому развитию экономики России
«Создание системы подготовки высококвалифицированных кадров в области суперкомпьютерных технологий и специализированного программного обеспечения»

Содержание курса

- Введение в параллельное программирование с использованием стандарта OpenMP. Параллельные области
- Параллельные циклы. Секции. Директивы master, single, workshare, threadprivate.
- Синхронизация в OpenMP
- Введение в параллельное программирование с использованием технологии CUDA
- Иерархия памяти ГПУ и работа с ней в CUDA
- Работа с разделяемой памятью. Синхронизация в CUDA

Содержание лекции

- Графические процессорные устройства (ГПУ): архитектура. Программирование для ГПУ: CUDA C/Fortran, OpenCL, Accelerator.
- Типовой алгоритм для вычислений на ГПУ.
- Модель параллельной программы CUDA. Иерархия потоков.
- Расширения CUDA для языков C/C++/Fortran.
- Запуск и управление вычислениями на ГПУ.
- Компиляция и запуск CUDA-программ.
- Пример CUDA-программы

Графические процессорные устройства

nVidia

Tesla, GTX, …

AMD

ATI Radeon,
ATI FireStream, …

Архитектура ГПУ

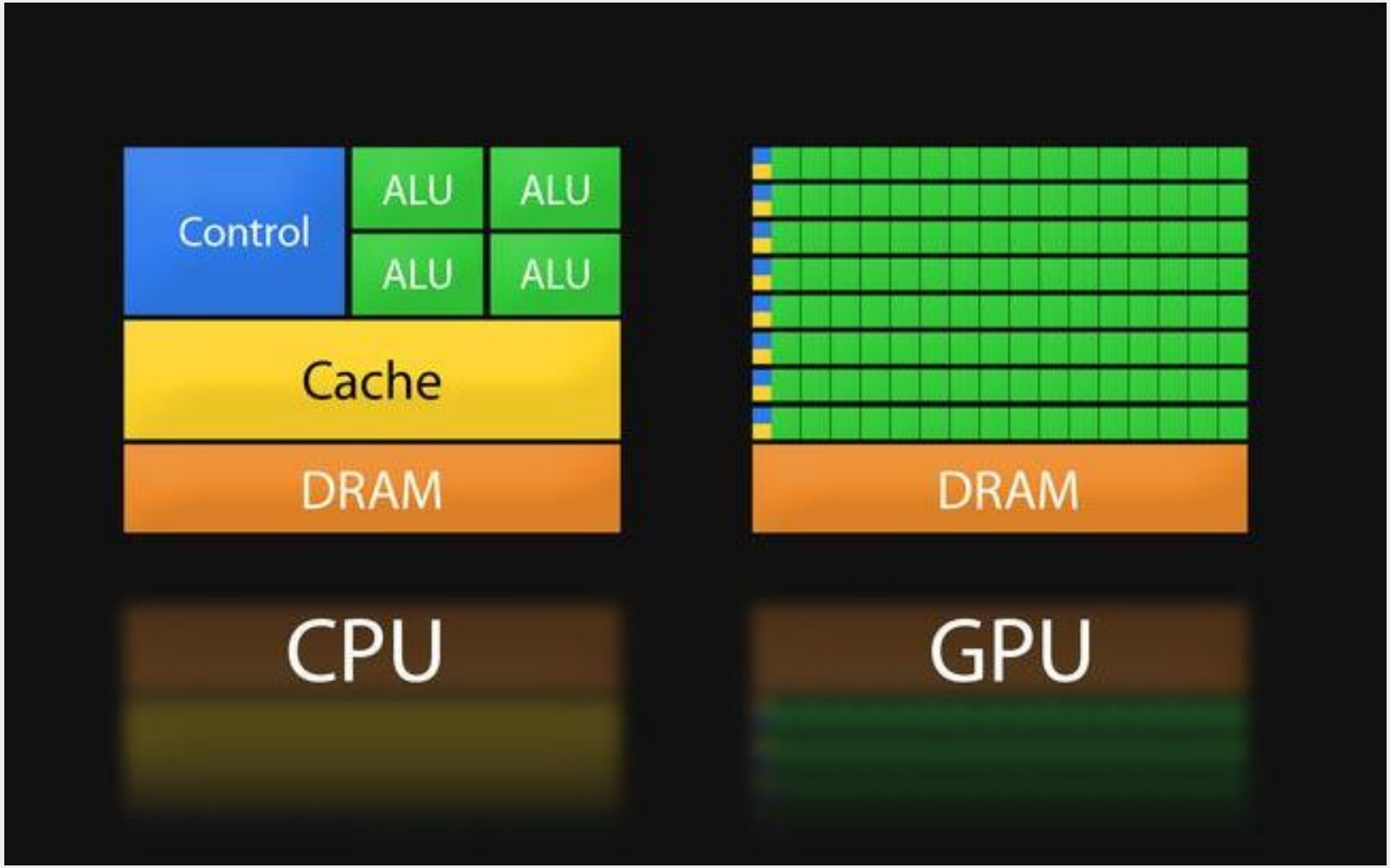
- **ГПУ** = набор слабо связанных мультипроцессоров (несколько десятков).

- **Мультипроцессор** = набор потоковых ядер (~ 10), *регистров* (~ 10 кб) и *разделяемой памяти* (~ 10 кб).
- Разделяемая память = аналог кэш в ЦПУ.

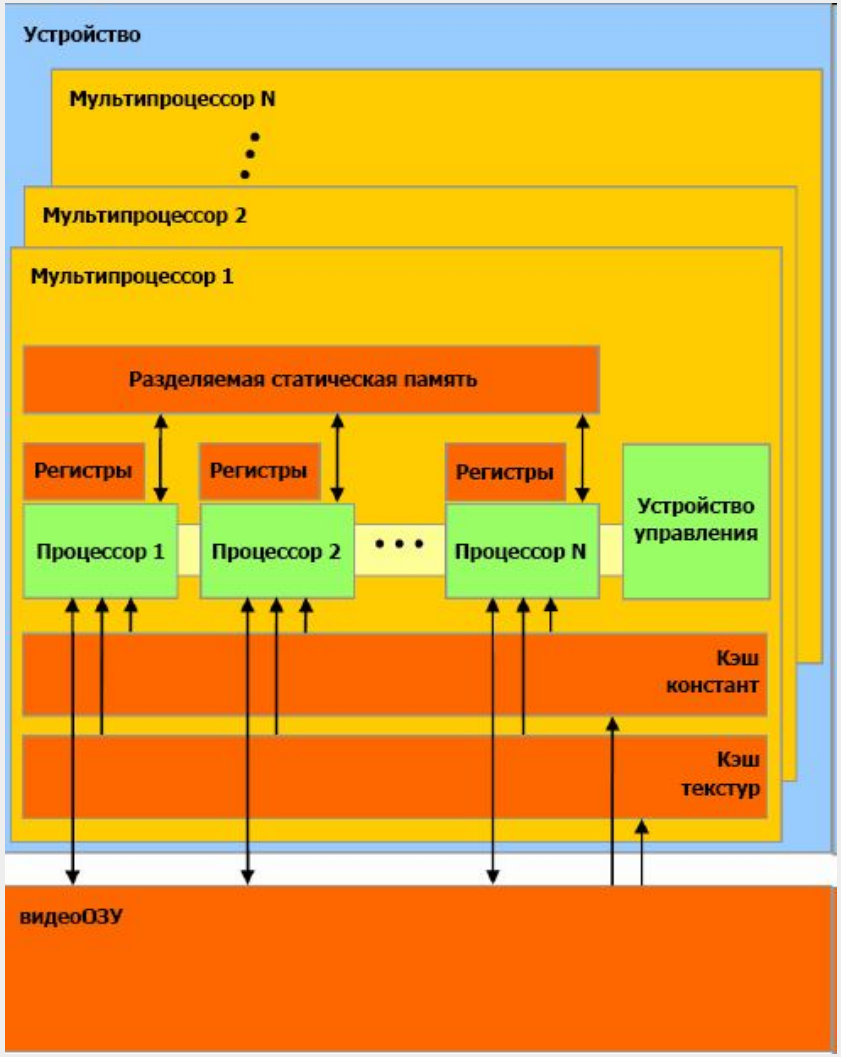
Архитектура ГПУ

- Мультипроцессоры не синхронизированы между собой.
- Каждое **потоковое ядро** в пределах мультипроцессора выполняет одну и ту же инструкцию (SIMD).
- Обычно ГПУ имеет собственную оперативную память (~ 10 Гб).

Архитектура ГПУ



Архитектура ГПУ



Ограничения ГПУ

- Ориентация на задачи с мелкозернистым параллелизмом.
- Для получения значительного ускорения требуется глубокое знание архитектуры ГПУ и относительно низкоуровневое программирование.
- Реальная производительность ограничена скоростью передачи данных между CPU и GPU.

Технологии программирования ГПУ

- **CUDA C** (nVidia)

  расширение языка C + библиотеки организации вычислений + стандартные библиотеки cuBLAS и cuFFT
- **CUDA Fortran** (PGI)

  расширение языка Fortran 2003
- **OpenCL** – открытый стандарт, поддержанный Apple, AMD, ARM, IBM, Intel, Motorola, nVidia и др. производителями.

  Расширение языка C и библиотеки организации вычислений.

Технологии программирования ускорителей и гибридных платформ

- **Accelerator** (PGI)
  Директивы компилятору (аналог OpenMP).
  Поддержка C99 и Fortran95/2003
  Неявная модель программирования

- **Ct** (Intel)
  Библиотека (framework) для C++
  Ориентация на параллельность по данным
  Технология основана на RapidMind

Решаемые задачи на ГПУ

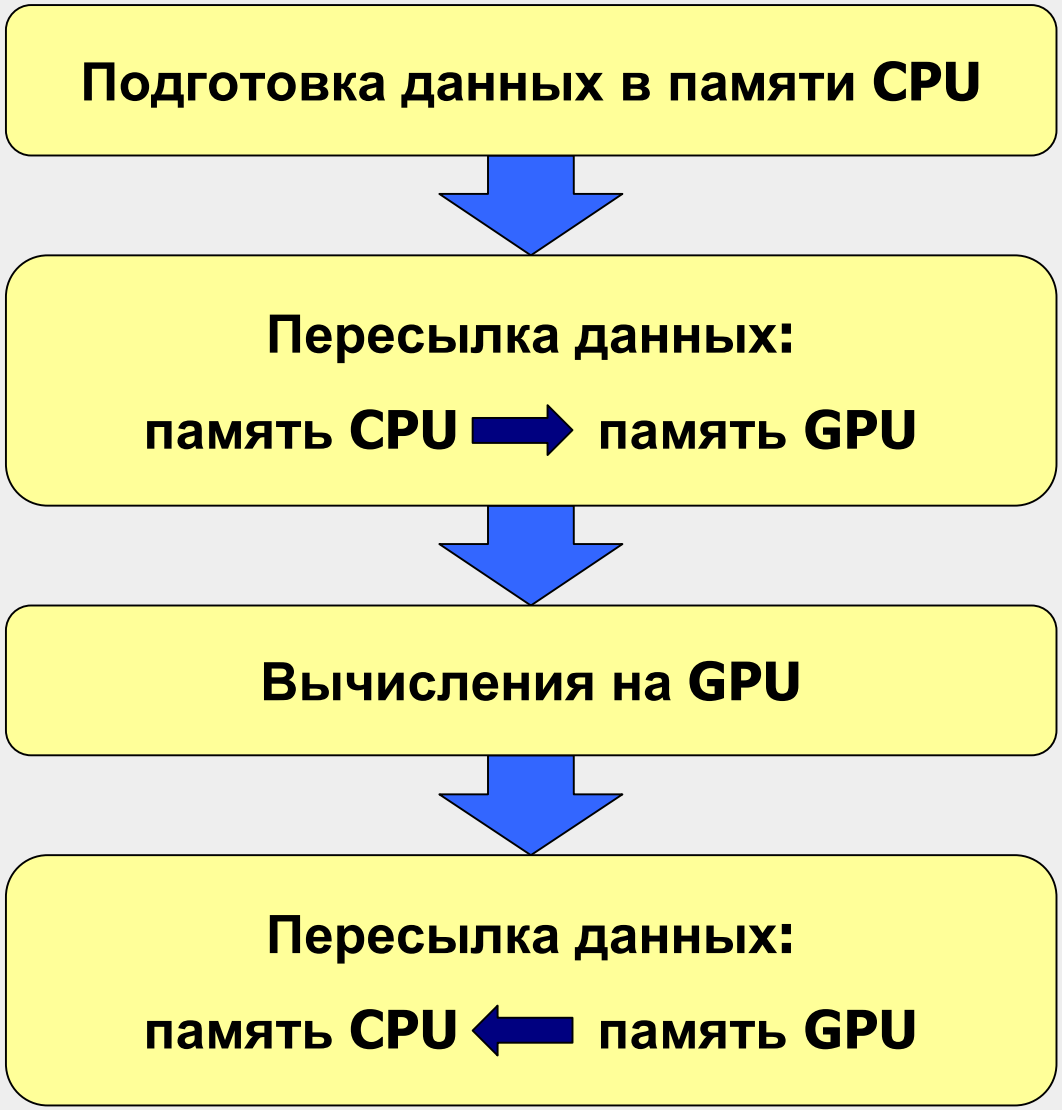
- Вычислительная гидродинамика / электродинамика / теплопередача / астрофизика / молекулярная динамика …

  сеточные методы

  методы частиц

  гибридные методы (частицы-в-ячейках)
- Методы Монте-Карло
- Клеточные автоматы
- Линейная алгебра
- …

Типовой алгоритм взаимодействия с ГПУ



CUDA

- CUDA == Compute Unified Device Architecture
- ≈ Унифицированная архитектура вычислительного устройства
- поддержка языков C/C++/Fortran/OpenCL/DirectCompute/Java/Python
- первоначальная ориентация на ГПУ, однако есть реализации для ЦПУ (например, от PGI)
- библиотеки: cuBLAS, cuFFT, Video, PhysX, ...
- библиотека времени выполнения (run-time)
- драйвер ГПУ

Модель параллельной программы CUDA. Иерархия потоков. Блоки

- Все параллельные потоки (нити, треды, threads) выполняются на одном мультипроцессоре.
- На одном мультипроцессоре могут выполняться несколько блоков одновременно.
- На одном мультипроцессоре могут выполняться несколько блоков последовательно.
- Мультипроцессор выполняет потоки в пределах блока группами (ворпами, warps).

Модель параллельной программы CUDA. Иерархия потоков. Блоки

- Потоки внутри блока имеют многомерную индексацию: 1D / 2D / 3D.
- Поток в пределах блока нумеруется при помощи структуры threadIdx, состоящей из полей threadIdx.x, threadIdx.y, threadIdx.z.
- Некоторые из полей могут быть равны 1, что позволяет говорить об 1D / 2D / 3D индексации.
- Размерности блока потоков определяется структурой blockDim, состоящей из полей blockDim.x, blockDim.y, blockDim.z.

Модель параллельной программы CUDA. Иерархия потоков. Блоки

- Можно перейти от многомерной индексации потоков (в пределах блока) к линейной, используя простые формулы:

$$n_{2D} = \text{blockDim.x} * \text{threadIdx.y} + \text{threadIdx.x}$$

$$n_{3D} = \text{blockDim.x} * \text{blockDim.y} * \text{threadIdx.z} + \text{blockDim.x} * \text{threadIdx.y} + \text{threadIdx.x}$$

Модель параллельной программы CUDA. Иерархия потоков. Грид

- Блоки потоков группируются в грид (grid) - множество потоков, решающих общую задачу (выполняющих одно ядро, kernel).
- Блоки внутри грида имеют многомерную индексацию: 1D / 2D / 3D.
- Блок в пределах грида нумеруется при помощи структуры blockIdx, состоящей из полей blockIdx.x, blockIdx.y, blockIdx.z.
- Некоторые из полей индекса блока могут быть равны 1, что позволяет говорить об 1D / 2D / 3D индексации.

Модель параллельной программы CUDA. Иерархия потоков

Грид

Блок (0,0)		
(0,0)	(1,0)	(2,0)
(0,1)	(1,1)	(2,1)

Блок (1,0)		
(0,0)	(1,0)	(2,0)
(0,1)	(1,1)	(2,1)

Блок (2,0)		
(0,0)	(1,0)	(2,0)
(0,1)	(1,1)	(2,1)

Блок (0,1)		
(0,0)	(1,0)	(2,0)
(0,1)	(1,1)	(2,1)

Блок (1,1)		
(0,0)	(1,0)	(2,0)
(0,1)	(1,1)	(2,1)

Блок (2,1)		
(0,0)	(1,0)	(2,0)
(0,1)	(1,1)	(2,1)

Расширения CUDA для языков C/C++/Fortran

- ГПУ программируются на языках высокого уровня, основанных на традиционных последовательных языках.
- В языки C/C++/Fortran вводятся новые сущности, позволяющие учитывать архитектуру (неполный список):
  - спецификаторы типа памяти объекта;
  - спецификаторы типа устройства, на котором выполняется функция;
  - встроенные векторные типы данных;
  - встроенные переменные;

Расширения CUDA для языков C/C++/Fortran

- функции синхронизации потоков;
- функции синхронизации памяти;
- доп. математические функции;
- функции для работы с текстурами;
- функции для работы с поверхностями;
- атомарные операции (реализуются как функции);
- синтаксис вызова ЦПУ функции, выполняющейся на ГПУ
- библиотека времени выполнения (работа с памятью ГПУ, получение информации о ГПУ, управление ГПУ)

Запуск вычислений на ГПУ

```
/* вызов ядра - код выполняется на ЦПУ */
dim3 dimBlock(thread_block_size);
dim3 dimGrid(N / dimBlock.x);
matvecmul_kernel <<<dimGrid, dimBlock>>> (A_d, x_d, y_d, N);


/* определение ядра - код выполняется на ГПУ */
__global__ void matvecmul_kernel(...)
{
...
}
```

Запуск вычислений на ГПУ

```
/* Шаблон запуска ядра */
kernel <<<dimGrid, dimBlock, [Nshared, Stream]>>> (...);
```

dim3 dimGrid - структура, определяющая размерность и размер грида

dim3 dimBlock - структура, определяющая размерность и размер каждого блока

size_t Nshared [= 0] - определяет кол-во байт разделяемой (shared) памяти, динамически выделяемой на блок (в добавок к статически выделяемой памяти)

cudaStream_t Stream [= 0] - определяет ассоциированный с ядром стрим (stream)

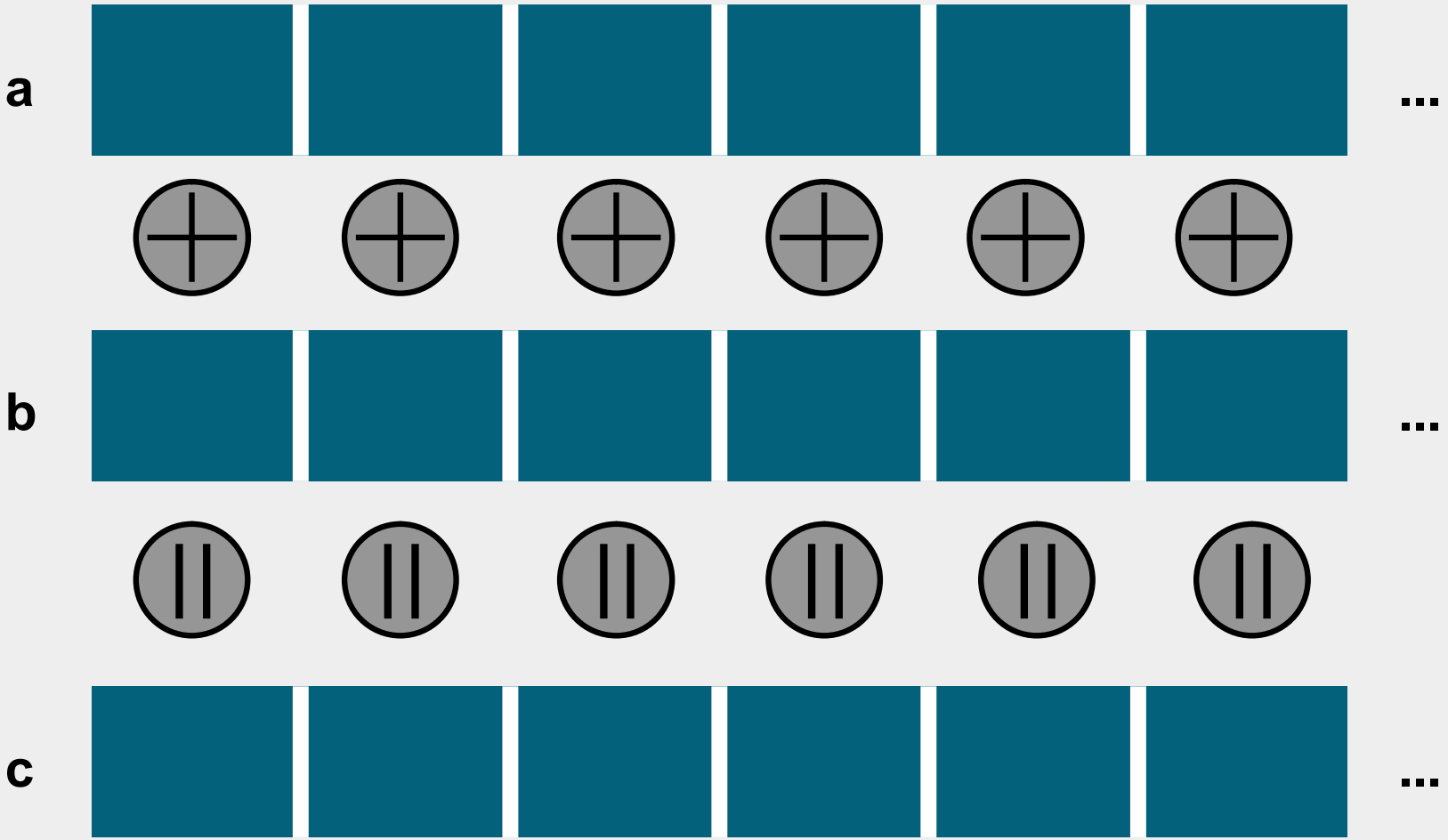
Компиляция и запуск CUDA-программ

- **nvcc** - драйвер компилятора
- nvcc разделяет код, выполняемый на ЦПУ и ГПУ
- Код для ЦПУ модифицируется для компиляции с использованием традиционных компиляторов (<<< ... >>> заменяется на функцию времени выполнения CUDA)
- Код для ЦПУ может быть скомпилирован отдельно пользователем или nvcc может вызвать соответствующий компилятор
- Код для ГПУ компилируется в бинарную форму (cubin) или в инструкции ассемблера (PTX)

Компиляция и запуск CUDA-программ

- Пример компиляции:
  - nvcc --ptxas-options=-v -code=sm_21 file1.cu file2.c -o prog
- Пример запуска в ОС GNU\Linux:
  - ./prog

Сложение векторов



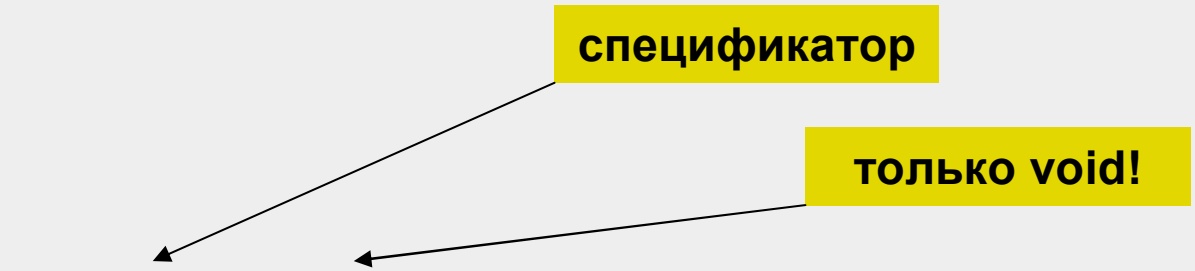
Пример CUDA-программы: код ЦПУ

```
#define BLOCK_SIZE 256
int main(int argc, char** argv)
{
    float *d_A, *d_B, *d_C;
    int N = BLOCK_SIZE*100;
    size_t size = N * sizeof(float);
    float *h_A = (float*)malloc(size); float *h_B = (float*)malloc(size);
    float *h_C = (float*)malloc(size);
    Init(h_A, N); Init(h_B, N);
    cudaMalloc((void**)&d_A, size); cudaMalloc((void**)&d_B, size);
    cudaMalloc((void**)&d_C, size);
    cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

    int threadsPerBlock = BLOCK_SIZE; int blocksPerGrid = N / threadsPerBlock;
    VecAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N); cudaMemcpy(h_C,
      d_C, size, cudaMemcpyDeviceToHost);
      ...
}
```

вызов ядра

Пример CUDA-программы: код ГПУ




```
__global__ void VecAdd(const float *A, const float *B, float *C, int N)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    C[i] = A[i] + B[i];
}
```

Спецификаторы функций

Спецификатор	Выполняется на	Вызывается с
`__device__`	ГПУ	ГПУ
`__global__`	ГПУ	ЦПУ
`__host__`	ЦПУ	ЦПУ

- **`__device__`** и **`__host__`** могут быть использованы совместно

Особенности функции-ядра (kernel)

- Определяется спецификатором `__global__`
- Не возвращает значения (void)
- Не поддерживает рекурсию
- Не поддерживает статических переменных (static)
- Не поддерживает переменное кол-во аргументов
- Вызывается только с ЦПУ
- При вызове требует специального синтаксиса <<<...>>>
- Вызывается асинхронно

Особенности функции, вызываемой с ГПУ

- Определяется спецификатором `__device__`
- Не поддерживает рекурсию
- Не поддерживает статических переменных (static)
- Не поддерживает переменное кол-во аргументов
- Не поддерживает взятие указателя на функцию

Вопросы для обсуждения

- Каковы достоинства и недостатки ГПУ?
- Как соотносятся аппаратная архитектура ГПУ и программная модель CUDA?
- Для чего применяется многомерная индексация потоков?
- Какие добавления привнесены в последовательные языки программирования технологией CUDA?
- Почему код, выполняющийся на ЦПУ, до его компиляции модифицируется компилятором nvcc?

Темы заданий для самостоятельной работы

- Параллельная реализация поэлементного произведения векторов
- Параллельная реализация суммирования матриц
- Параллельная реализация матрично-векторного произведения
- Параллельная реализация матрично-матричного произведения

Литература

- http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf
- http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Best_Practices_Guide.pdf
- http://www.steps3d.narod.ru/tutorials/cuda-tutorial.html
- Сандерс Дж., Кэндрот Э. Технология CUDA в примерах. Введение в программирование графических процессоров. М.: ДМК Пресс, 2011 г., ISBN 978-5-94074-504-4, 978-0-13-138768-3

Содержание курса

- Введение в параллельное программирование с использованием стандарта OpenMP. Параллельные области
- Параллельные циклы. Секции. Директивы master, single, workshare, threadprivate.
- Синхронизация в OpenMP
- Введение в параллельное программирование с использованием технологии CUDA
- Иерархия памяти ГПУ и работа с ней в CUDA
- Работа с разделяемой памятью. Синхронизация в CUDA