СУПЕРКОМПЬЮТЕРНЫЙ КОНСОРЦИУМ УНИВЕРСИТЕТОВ РОССИИ

Проект

Создание системы подготовки высококвалифицированных кадров в области суперкомпьютерных технологий и специализированного программного обеспечения

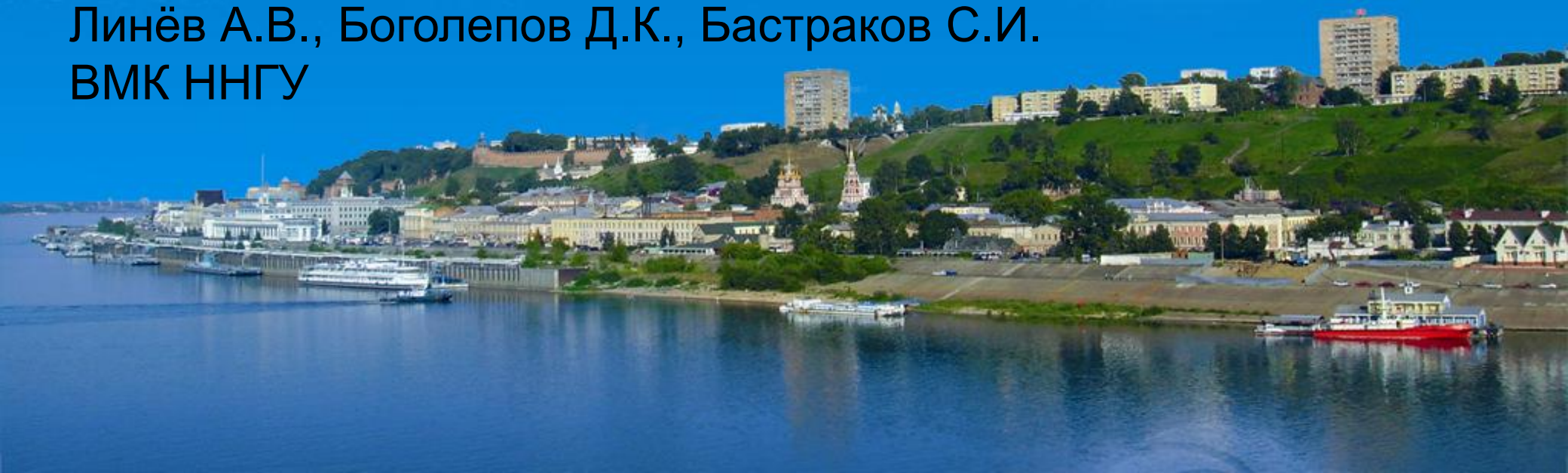
Московский государственный университет им. М.В. Ломоносова

Нижегородский государственный университет
им. Н.И. Лобачевского
- Национальный исследовательский университет -

Технологии параллельного программирования для процессоров новых архитектур

Лекция 2. Вычисления с использованием графических процессоров. Технология CUDA

Линёв А.В., Боголепов Д.К., Бастраков С.И.
ВМК ННГУ

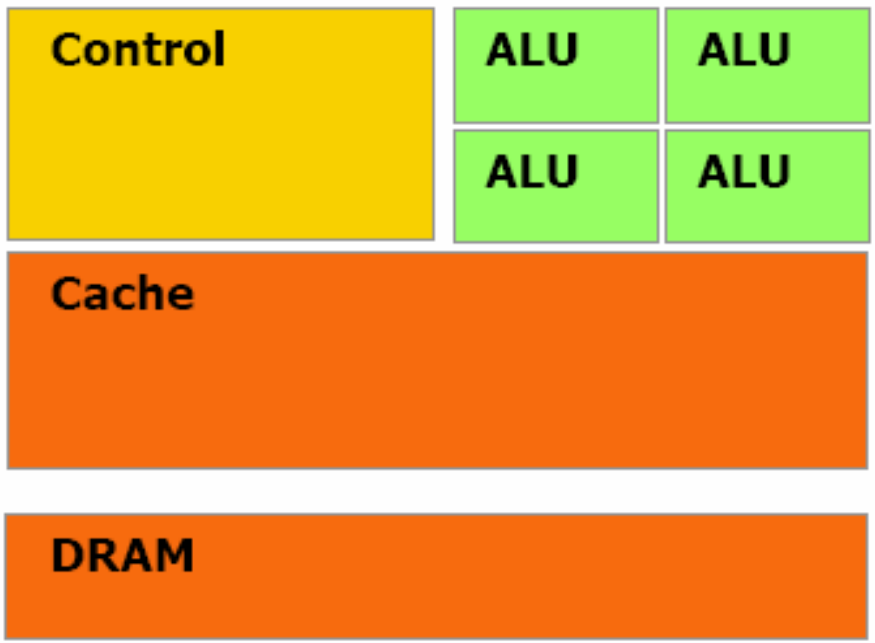


Содержание

- Архитектура ГПУ NVIDIA
- Модель выполнения
- Язык CUDA C

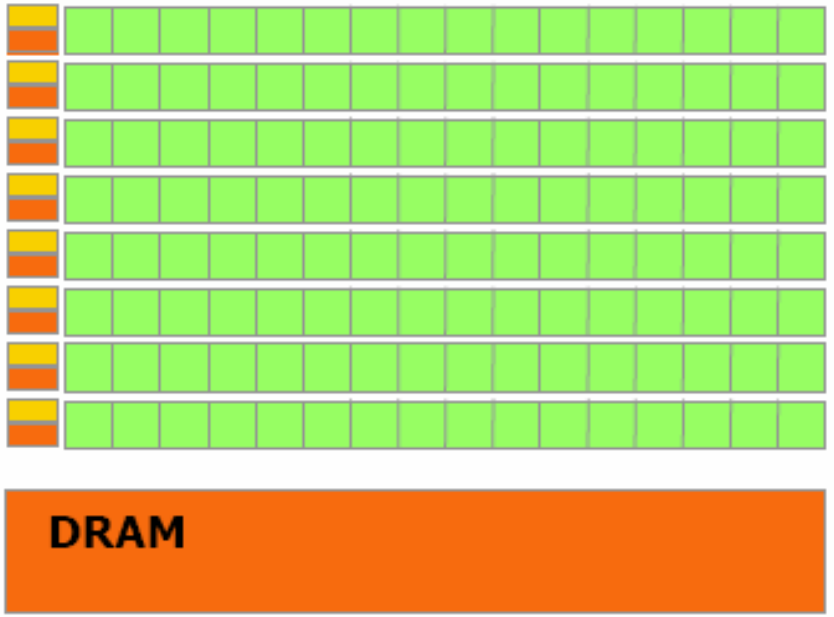
Архитектура ГПУ NVIDIA

Архитектура ЦПУ и ГПУ…



ЦПУ

“cache-oriented”



ГПУ

“cache-miss oriented”

Источник: NVIDIA CUDA C Programming Guide v. 3.2

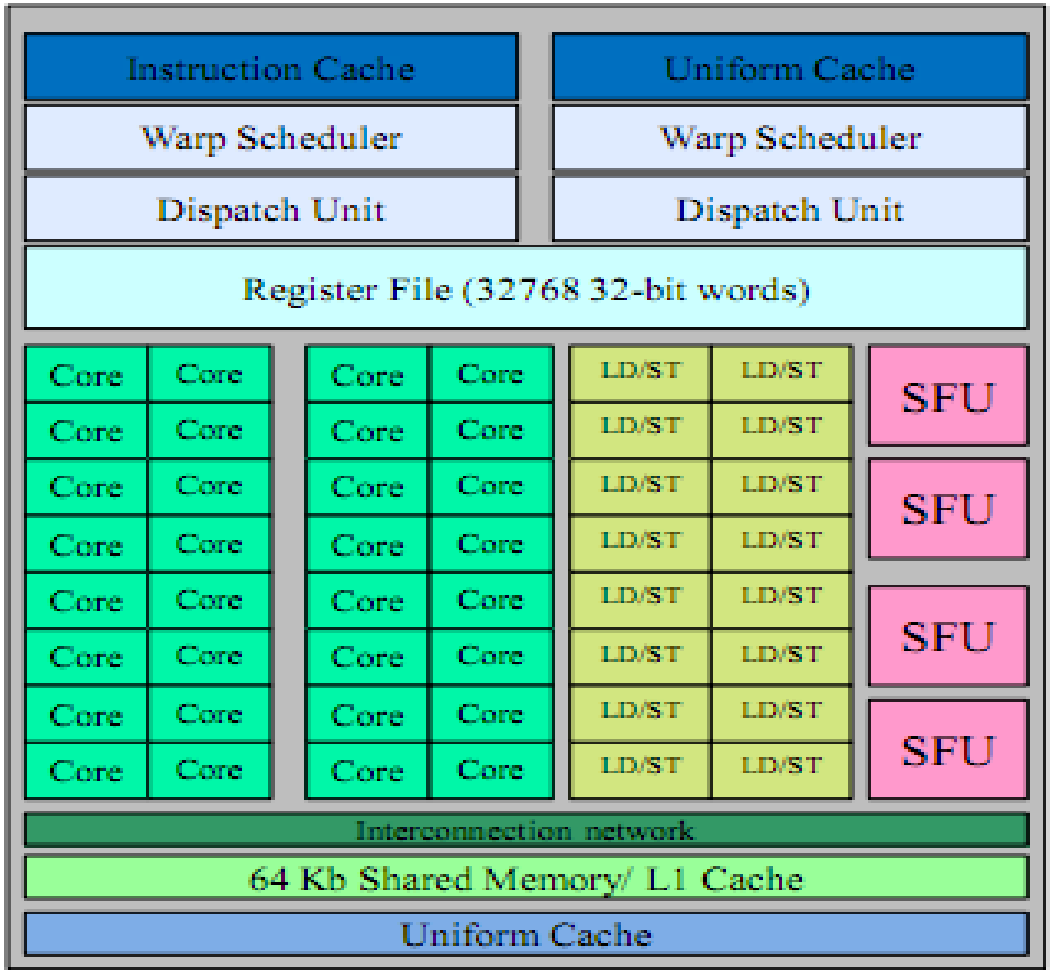
Архитектура ЦПУ и ГПУ

- ГПУ предназначен для вычислений
  - **параллельных по данным**: одна и та же операция выполняется над многими данными параллельно
  - в которых отношение вычислительных операций к числу операций по доступу к памяти велико
- Вместо кэша и сложных элементов управления на кристалле размещено большее число вычислительных элементов
- Латентность памяти покрывается за счет большого количества легковесных потоков
- Постепенно сложность архитектуры ГПУ повышается

Архитектура ГПУ NVIDIA

- ГПУ – массивно-параллельный многоядерный процессор
- Состоит из **мультипроцессоров** (*streaming multiprocessor*, *MP*), каждый из которых содержит несколько **CUDA-ядер** (*CUDA core*) и общую для них память
  - В архитектурах до Fermi аналоги CUDA-ядер назывались **скалярными процессорами** (*scalar processor*, *SP*)
- CUDA-ядра внутри одного мультипроцессора работают как SIMD
- Легковесные потоки, встроенный планировщик

Мультипроцессор Fermi



Источник: А.В. Боресков, А.А. Харламов «Архитектура и программирование массивно-параллельных вычислительных систем»

Архитектура Fermi

- Новейшая на текущий момент архитектура NVIDIA
- Добавлен общий L2-кэш для глобальной памяти
- На каждом мультипроцессоре добавлен L1-кэш для глобальной памяти. Физически расположен вместе с разделяемой памятью, возможность настройки размеров L1-кэша и разделяемой памяти 48kB/16kB
- Значительно улучшена производительность арифметики с плавающей точкой двойной точности и SFU

Иерархия памяти

- **Глобальная** (*device/global*) – общая для устройства. Кэшируется на Fermi.
- **Константная** и **текстурная** – подобласти глобальной памяти. Кэшируются на всех ГПУ.
- **Разделяемая** (*shared*) – общая для всех CUDA-ядер в одном MP.
- **Регистры** (*register*) – (логически) эксклюзивны для CUDA-ядра.
- **Локальная** (*local*) – (логически) эксклюзивна для CUDA-ядра, физически расположена в подобласти глобальной памяти.

Вычислительные возможности…

- **Вычислительные возможности** (*compute capability*) служат для классификации устройств по качественным архитектурным особенностям
- Определяются при помощи числового индекса, состоящего из номеров главной (major) и второстепенной (minor) ревизий, записывается в виде major.minor
  - Например, 1.3
- Архитектура Fermi имеет номер главной ревизии 2, все более ранние архитектуры имеют номер главной ревизии 1

Вычислительные возможности

- Минимальный индекс 1.0
- Поддержка вычислений с плавающей точкой в двойной точности, начиная с индекса 1.3
- Поддержка классов (C++), начиная с индекса 2.0
- Полная поддержка C++ на устройствах индекса 2.0 и выше, начиная с версии CUDA 4.0
- От вычислительных возможностей существенно зависят требования для эффективного доступа к глобальной памяти и возможность выполнения специфических операций

Модель выполнения

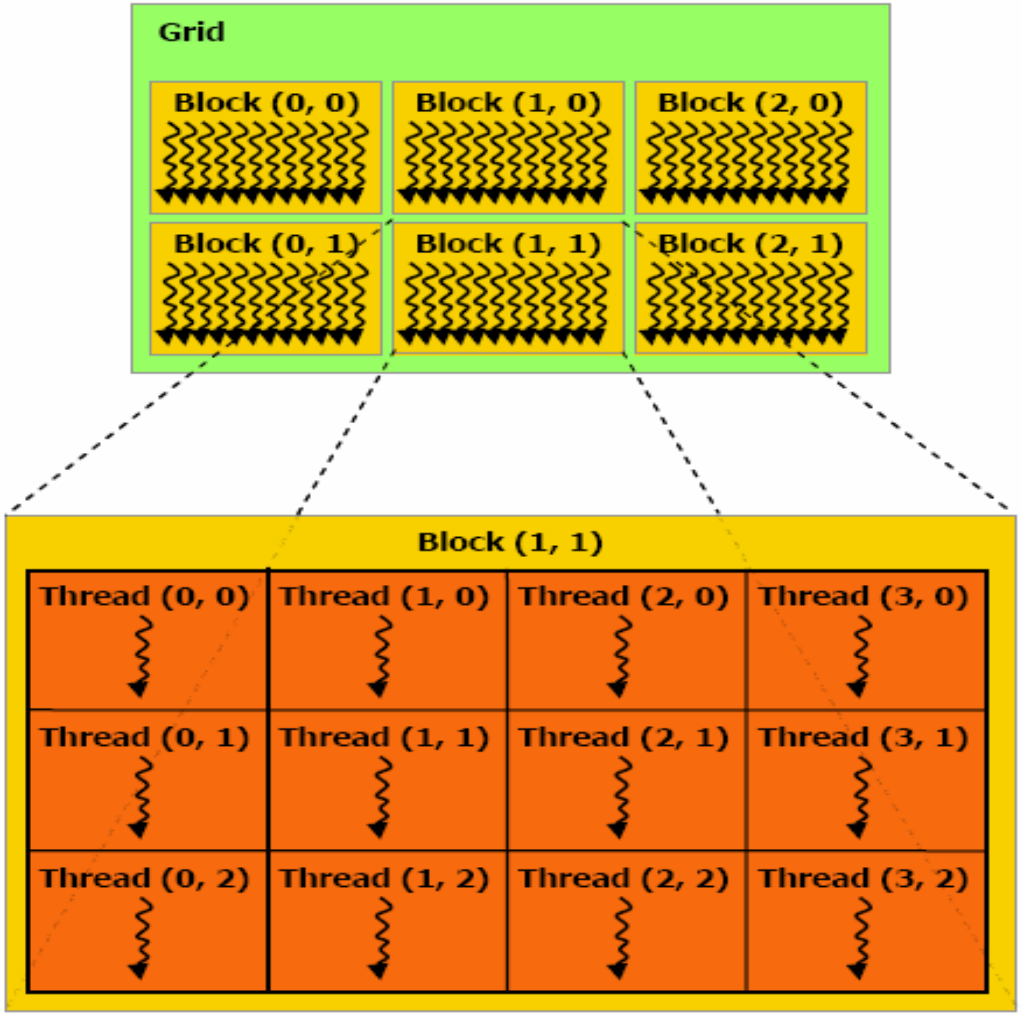
Основные понятия

- Большое количество **потоков** (*thread*) параллельно выполняют одну и ту же функцию – **ядро** (*kernel*)
- Потоки группируются в **блоки** (*thread blocks*)
- Каждый блок исполняется на одном мультипроцессоре, его потоки – на CUDA-ядрах данного мультипроцессора
- Блоки объединяются в **решетку/сетку блоков** (*grid*)
- Ядро выполняется на решетке из блоков
- Размер блока и размер решетки блоков задается при вызове ядра

Идентификаторы

- Каждый поток и блок потоков имеют **идентификаторы**
  - Block ID (1D, 2D или 3D)
  - Thread ID (1D, 2D или 3D)
- Многомерная индексация может упрощать декомпозицию многомерных данных
- Пример: ядро выполняет умножение матриц, каждый поток вычисляет один элемент результирующей матрицы. Удобно использовать двумерные индексы, например, x-компоненту для номеров строк и y-компоненту для столбцов

Идентификаторы: пример



Источник: NVIDIA CUDA C Programming Guide v. 4.0

Взаимодействие потоков

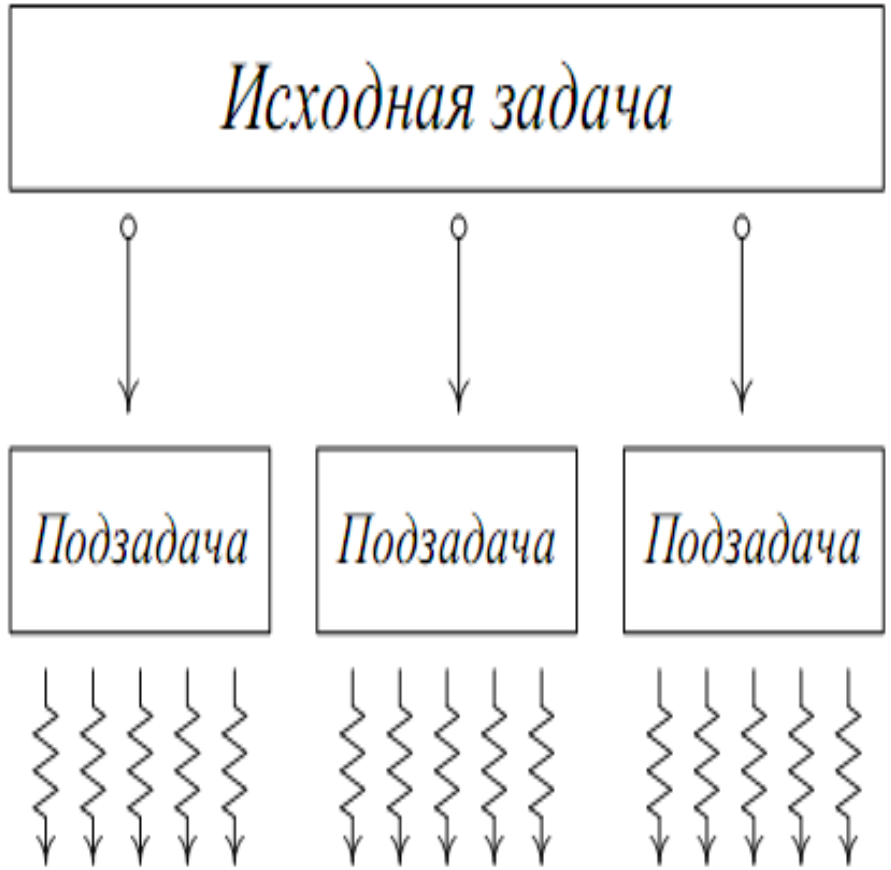
- Потоки внутри одного блока выполняются на одном мультипроцессоре, они способны взаимодействовать между собой посредством
  - разделяемой памяти
  - точек синхронизации
- Два потока из различных блоков могут взаимодействовать лишь через глобальную память
- Атомарные функции для разделяемой и глобальной памяти

Выполнение блоков

- Автоматическое распределение блоков на мультипроцессоры
- Каждый блок целиком выполняется одним мультипроцессором
- При наличии достаточного количества ресурсов, несколько блоков могут «одновременно» исполняться на одном мультипроцессоре
- Наличие большого количества блоков открывает возможности для автоматической масштабируемости с ростом числа мультипроцессоров

Типичная схема организации вычислений

- Крупнозернистый параллелизм на уровне блоков
- Мелкозернистый параллелизм на уровне потоков внутри блока
- Исходная задача разбивается на независимые подзадачи, каждая из которых решается параллельно одним блоком



Источник: *А.В. Боресков, А.А. Харламов «Архитектура и программирование массивно-параллельных вычислительных систем»*

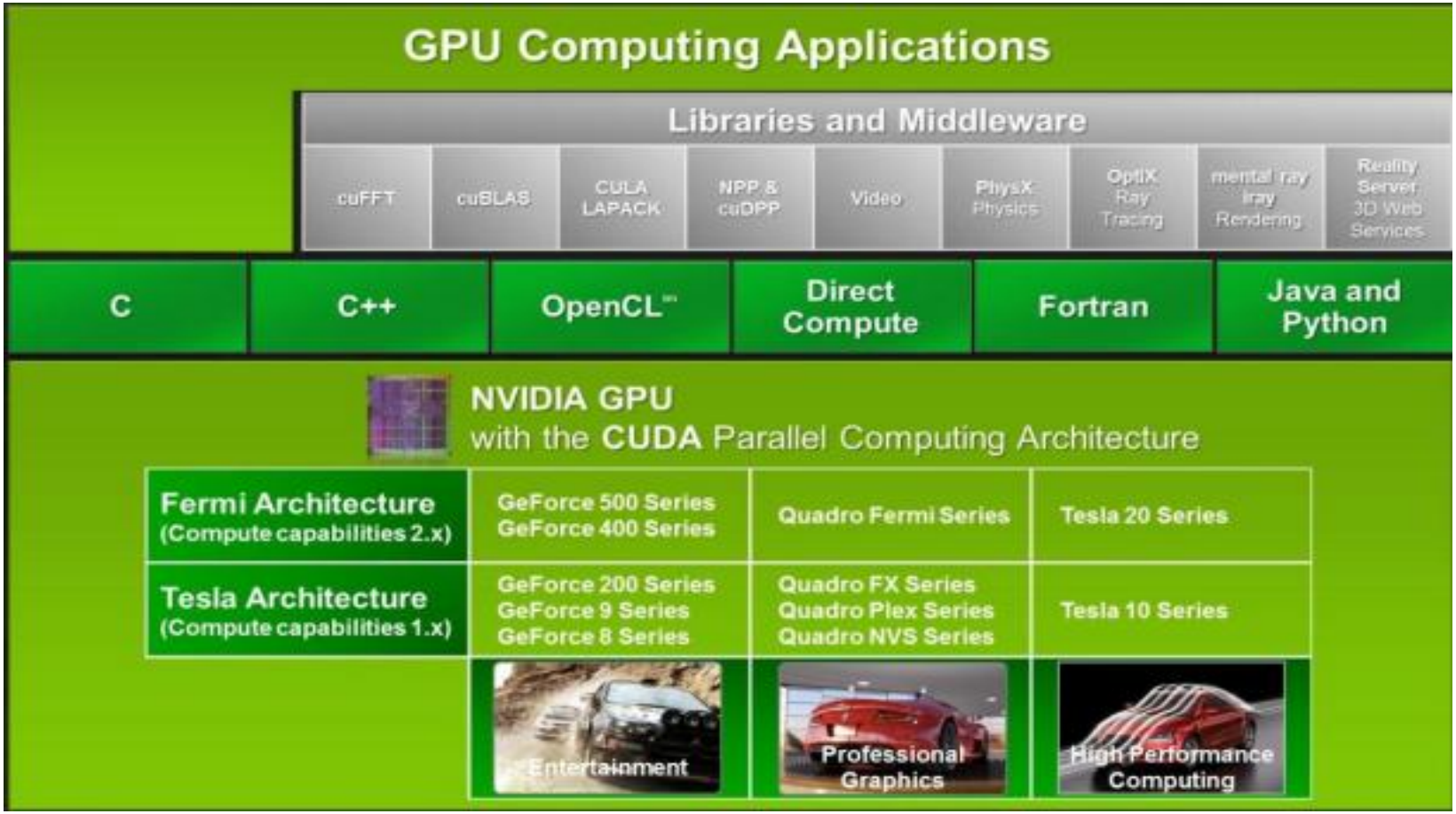
Язык CUDA C

NVIDIA CUDA

- **CUDA –** *Compute Unified Device Architecture*
- Программно-аппаратная платформа для параллельных вычислений от NVIDIA
- Раскрывает потенциал ГПУ для вычислений общего назначения
- Не поддерживается другими производителями

NVIDIA CUDA

Поддержка различных интерфейсов программирования:



Источник: NVIDIA CUDA C Programming Guide v. 4.0

Комплект поставки CUDA

- CUDA driver
- CUDA toolkit
  - компилятор
  - профилировщик
  - оптимизированные библиотеки
  - документация
- GPU Computing SDK
- Поддержка основных операционных систем

CUDA C

- CUDA C – расширение C/C++, включающее
  - квалификаторы функций
  - квалификаторы типов памяти
  - встроенные переменные
- Содержит элементы C++. Полная поддержка C++ для современных устройств.
- Терминология
  - **хост** (*host*) = ЦПУ
  - **устройство** (*device*) = ГПУ
  - **ядро** (*kernel*) – подпрограмма, параллельно выполняемая потоками на ГПУ

Квалификаторы функций

Квалификатор	*Выполняется на:*	*Вызывается с:*
__host__	*host*	*host*
__global__	*device*	*host*
__device__	*device*	*device*

- **__host__** (по умолчанию) – функция, вызываемая с хоста и выполнямая на нем
- **__global__** – функция, вызываемая с хоста и выполняемая потоками на устройстве (ядро)
- **__device__** – функция, вызываемая (одним потоком) и выполняемая на устройстве

Квалификаторы переменных…

- **__device__** объявляет переменную, размещаемую на устройстве
  - размещается в глобальном пространстве памяти
  - время жизни переменной совпадает со временем жизни приложения
  - доступ к переменной может быть осуществлен из всех потоков, выполняемых на устройстве, а также с хоста через библиотеки времени выполнения
- **__constant__** объявляет переменную, размещаемую на устройстве в константной области памяти и доступную только на чтение

Квалификаторы переменных

- **__shared__** объявляет переменную, которая
  - размещается в пространстве разделяемой памяти мультипроцессора, на котором исполняется блок потоков
  - время жизни переменной совпадает со временем жизни блока потоков
  - доступ к переменной может быть осуществлен из потоков, принадлежащих блоку потоков

Встроенные векторные типы

- [u]char[1..4], [u]int[1..4], [u]long[1..4], float[1..4], double2 – являются структурами, доступ через .x, .y, .z, .w
- Нет конструкторов, но есть make_<type name>
- Нет встроенных векторных арифметических операций
- dim3 = uint3 + конструктор, по умолчанию заполняется 1. Используется для задания числа блоков и потоков при вызове ядер.
- Могут использоваться в хостовой (C/C++) части кода

Встроенные переменные

- В коде для ГПУ доступны следующие встроенные переменные
  - **gridDim** – переменная типа dim3, содержит текущую размерность решетки
  - **blockIdx** – переменная типа uint3, содержит индекс блока потоков внутри решетки
  - **blockDim** – переменная типа dim3, содержит размерность блока потоков
  - **threadIdx** – переменная типа uint3, содержит индекс потока внутри блока потоков
- Данные переменные предназначены только для чтения

Вычисление уникального индекса потока

- Нет встроенной переменной для получения индекса потока среди всех потоков (т.е. всех потоков всех блоков)
- Он может быть вычислен через значения других встроенных переменных
- Для простоты будем считать, что используется только x-компонента индексов

idx = blockIdx.x * blockDim.x + threadIdx.x

Искомый индекс

Смещение нулевого потока данного блока относительно нулевого потока нулевого блока

Смещение потока относительно нулевого потока данного блока

CUDA API

- Состав CUDA API
  - управление устройствами
  - управление контекстами
  - управление памятью
  - управление процессом выполнения
- Все функции возвращают значение типа **cudaError_t**, **cudaSuccess** в случае успешного завершения функции
- Уровни API
  - низкоуровневое (CUDA driver API): cu*
  - высокоуровневое (C runtime for CUDA): cuda*

Управление устройствами…

- Перечисление устройств
  - cudaError_t **cudaGetDeviceCount**(int* count) – возвращает число доступных устройств
  - cudaError_t **cudaGetDevice** (int* dev) – возвращает используемое устройство
  - cudaError_t **cudaGetDeviceProperties** (struct cudaDeviceProp* prop, int dev) – возвращает структуру, содержащую свойства устройства

Управление устройствами

- Выбор устройства
  - cudaError_t **cudaSetDevice** (int dev) – устанавливает устройство с заданным номером
  - cudaError_t **cudaChooseDevice** (int* dev, const struct cudaDeviceProp* prop) – выбирает устройство, в наибольшей степени соответствующее конфигурации

Управление памятью

- cudaError_t **cudaMalloc** (void** devPtr, size_t count) – выделяет память на устройстве и возвращает указатель на нее
- cudaError_t **cudaFree** (void* devPtr) – освобождает память на устройстве
- cudaError_t **cudaMemcpy** (void* dst, const void* src, size_t count, enum cudaMemcpyKind kind) – осуществляет копирование данных между хостом и устройством (блокирующий вариант)
- **cudaMemcpyAsync** – неблокирующий вариант

Вызов ядер

- По умолчанию вызов ядра является асинхронным
- Функция ядра должна быть вызвана с указанием **конфигурации исполнения**
- Конфигурация определяется использованием выражения специального вида **<<<Dg, Db, Ns, S>>>** между именем функции и списком ее аргументов
  - **Dg** – определяет размерность и размер сетки, общее число блоков равно Dg.x * Dg.y * Dg.z
  - **Db** – определяет размер блока потоков, общее количество потоков в блоке равно Db.x * Db.y * Db.z
  - Ns, S могут быть использованы по умолчанию

Вызов ядер: пример

- Пример:

```
__global__ void kernel_func() { … }
…
dim3 dim_grid(100, 50);
dim3 dim_block (8, 8, 8);
kernel_func<<<dim_grid, dim_block>>>( );
```

Синхронизация

- void **__syncthreads**() – барьерная синхронизация в для потоков внутри одного блока (вызывается внутри ядра)
- Атомарные арифметические функции для глобальной и разделяемой памяти
- **cudaThreadSyncronize**() – барьерная синхронизация для всех вызовов функций на устройстве (вызывается хостом)
  - Вызовы ядер асинхронны
- Возможна более сложная синхронизация на основе cudaStream_t и cudaEvent_t

“CUDA Hello, World!”

```
#include <cstdlib>
#include <iostream>
#include <cuda_runtime.h>

__global__ void hello(int * output)
{
    int globalIdx = blockIdx.x * blockDim.x + threadIdx.x;
    output[globalIdx] = globalIdx;
}
```

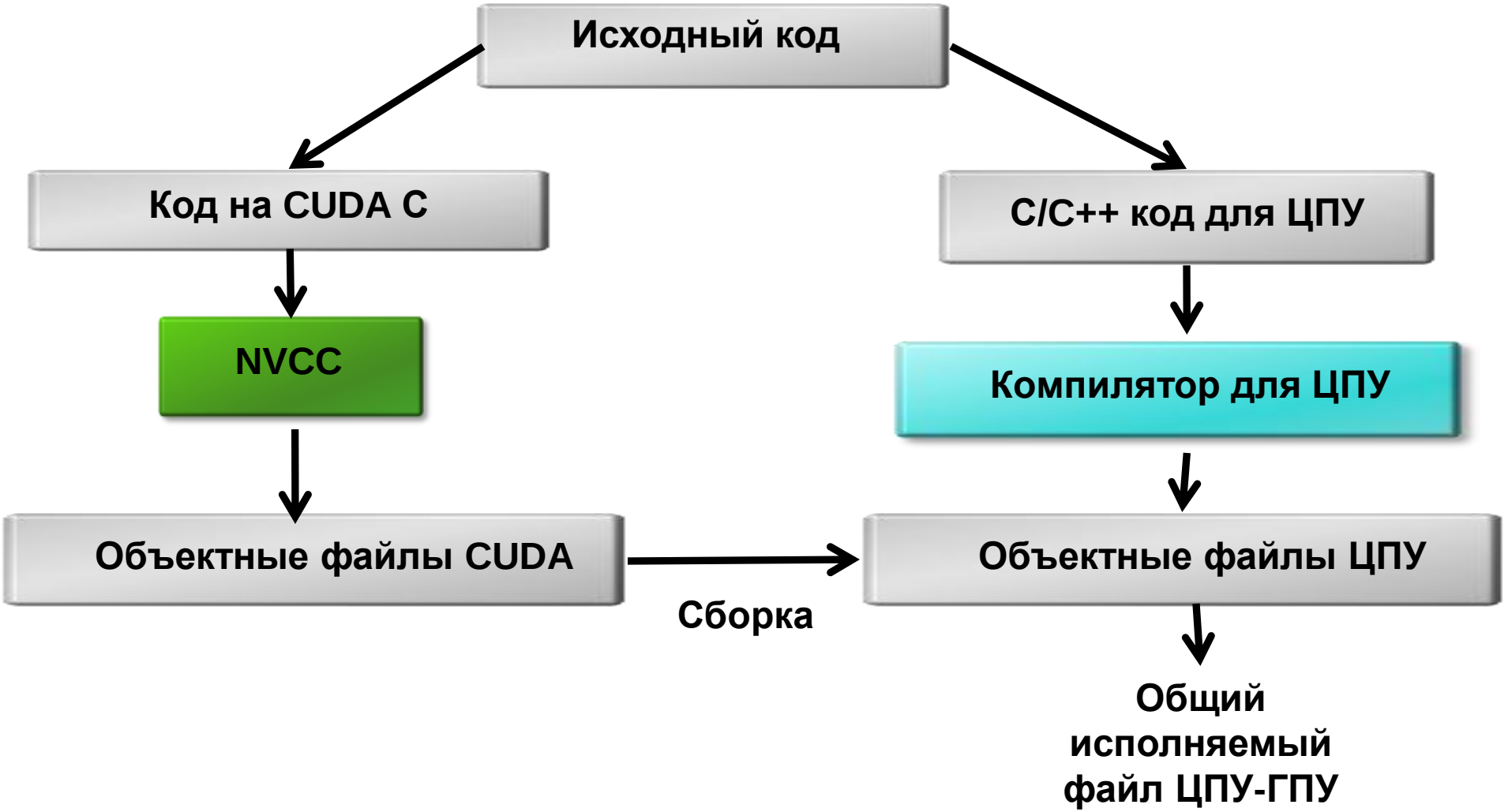
“CUDA Hello, World!”

```
int main() {
    int buffer_size = 4 * 512;
    int * buffer = new int[buffer_size];
    int * buffer_gpu;
    cudaMalloc((void **) &buffer_gpu,
                    buffer_size * sizeof(float));
    hello<<<4, 512>>>(buffer_gpu);
    cudaMemcpy(buffer, buffer_gpu, buffer_size * sizeof(float),
                        cudaMemcpyDeviceToHost);
    cudaFree(buffer_gpu);  delete [] buffer;
    return 0;
}
```

Компиляция и сборка

- Компилятор nvcc
- Build rules для Microsoft Visual Studio
- В CUDA до 4.0 поддерживались версии MSVS 2005 и 2008. В CUDA 4.0 добавлена поддержка MSVS 2010

Компиляция и сборка



Заключение

- ГПУ имеют сложную архитектуру и иерархию памяти
- Язык CUDA C является расширением C/C++ для программирования ГПУ
- Важнейшая концепция – ядро
  - Функция, вызываемая с ЦПУ и выполняемая на ГПУ
  - Ядро выполняется решеткой блоков потоков
  - Количество и размер блоков потоков задаются при запуске ядра

Литература

- А.В. Боресков, А.А. Харламов «Основы работы с технологией CUDA»
- Д. Сандерс, Э. Кэндрот «Технология CUDA в примерах: введение в программирование графических процессоров» (пер. с англ.)

Контакты

- Нижегородский университет,
- Факультет вычислительной математики и кибернетики

- Линёв Алексей Владимирович (alin@unn.ru)
- Боголепов Денис Константинович (denisbogol@gmail.com)
- Бастраков Сергей Иванович (sergey.bastrakov@gmail.com)

СПАСИБО ЗА ВНИМАНИЕ

Вопросы ?

О проекте

Целью проекта является создание национальной системы подготовки высококвалифицированных кадров в области суперкомпьютерных технологий и специализированного программного обеспечения.

Задачами по проекту являются:

Задача 1. Создание сети научно-образовательных центров суперкомпьютерных технологий (НОЦ СКТ).

Задача 2. Разработка учебно-методического обеспечения системы подготовки, переподготовки и повышения квалификации кадров в области суперкомпьютерных технологий.

Задача 3. Реализация образовательных программ подготовки, переподготовки и повышения квалификации кадров в области суперкомпьютерных технологий.

Задача 4. Развитие интеграции фундаментальных и прикладных исследований и образования в области суперкомпьютерных технологий. Обеспечение взаимодействия с РАН, промышленностью, бизнесом.

Задача 5. Расширение международного сотрудничества в создании системы суперкомпьютерного образования.

Задача 6. Разработка и реализации системы информационного обеспечения общества о достижениях в области суперкомпьютерных технологий.

См. http://www/hpc-education.ru