Программная модель CUDA

Храмченков Э.М.[1,2]

1. Казанский федеральный университет
2. НИИСИ РАН

Computer Science клуб, Казань, 21.10-23.10, 2015 г.

CUDA

* **CUDA** - **C**ompute **U**nified **D**evice **A**rchitecture

CUDA

* **CUDA** - **C**ompute **U**nified **D**evice **A**rchitecture
* Программная модель – включает вычислительный параллелизм и структуру памяти непосредственно в язык программирования

CUDA

* **CUDA** - **C**ompute **U**nified **D**evice **A**rchitecture
* Программная модель – включает вычислительный параллелизм и структуру памяти непосредственно в язык программирования
* CUDA – **кроссплатформенная** система компиляции и исполнения программ, части которых работают на CPU и GPU

CUDA

* **CUDA** - **C**ompute **U**nified **D**evice **A**rchitecture
* Программная модель – включает вычислительный параллелизм и структуру памяти непосредственно в язык программирования
* CUDA – **кроссплатформенная** система компиляции и исполнения программ, части которых работают на CPU и GPU
* Поддерживается всеми **GPU NVIDIA,** начиная с серии **GeForce 8**

CUDA и ЯП

* Для создания **CUDA-программ** используются **языки высокого уровня**

CUDA и ЯП

* Для создания **CUDA-программ** используются **языки высокого уровня**
* В настоящий момент существуют **компиляторы C++** и **Fortran**

CUDA и ЯП

* Для создания **CUDA-программ** используются **языки высокого уровня**
* В настоящий момент существуют **компиляторы C++** и **Fortran**
* Компилятор Fortran **платный,** предоставляется компанией PGI

CUDA и ЯП

* Для создания **CUDA-программ** используются **языки высокого уровня**
* В настоящий момент существуют **компиляторы C++** и **Fortran**
* Компилятор Fortran **платный,** предоставляется компанией PGI
* Компилятор C++ **бесплатный,** предоставляется компанией Nvidia в составе **Nvidia CUDA Toolkit**

CUDA и ЯП

* Для создания **CUDA-программ** используются **языки высокого уровня**
* В настоящий момент существуют **компиляторы C++** и **Fortran**
* Компилятор Fortran **платный,** предоставляется компанией PGI
* Компилятор C++ **бесплатный,** предоставляется компанией Nvidia в составе **Nvidia CUDA Toolkit**
* Есть варианты использования для Java, Python, C#...

CUDA и OS

* Поддерживаются **Linux**, **MacOS** и **Windows** системы

CUDA и OS

* Поддерживаются **Linux**, **MacOS** и **Windows** системы
* В Linux требуется установка пакета CUDA, который присутствует во многих менеджерах пакетов

CUDA и OS

* Поддерживаются **Linux**, **MacOS** и **Windows** системы
* В Linux требуется установка пакета CUDA, который присутствует во многих менеджерах пакетов
* Для **Microsoft Visual Studio 2010, 2012, 2013** поддерживается интеграция CUDA Toolkit в состав IDE (ОС **Microsoft Windows 7 и 8.1**)

CUDA и OS

* Поддерживаются **Linux**, **MacOS** и **Windows** системы
* В Linux требуется установка пакета CUDA, который присутствует во многих менеджерах пакетов
* Для **Microsoft Visual Studio 2010, 2012, 2013** поддерживается интеграция CUDA Toolkit в состав IDE (ОС **Microsoft Windows 7 и 8.1**)
* **Visual Studio 2015 не поддерживает** интеграцию с CUDA Toolkit

Основные принципы

* **Программа** на CUDA использует как **CPU** так и **GPU**

Основные принципы

* **Программа** на CUDA использует как **CPU** так и **GPU**
* Программа состоит из **последовательных** и **параллельных** участков **кода**

Основные принципы

* **Программа** на CUDA использует как **CPU** так и **GPU**
* Программа состоит из **последовательных** и **параллельных** участков **кода**
* На **CPU**, который называют **host**, выполняется **последовательная** часть кода, подготовка и вызов GPU кода

Основные принципы

* **Программа** на CUDA использует как **CPU** так и **GPU**
* Программа состоит из **последовательных** и **параллельных** участков **кода**
* На **CPU**, который называют **host**, выполняется **последовательная** часть кода, подготовка и вызов GPU кода
* На **GPU**, который называют **device**, выполняется **параллельная** часть кода

Основные принципы

* **Программа** на CUDA использует как **CPU** так и **GPU**
* Программа состоит из **последовательных** и **параллельных** участков **кода**
* На **CPU**, который называют **host**, выполняется **последовательная** часть кода, подготовка и вызов GPU кода
* На **GPU**, который называют **device**, выполняется **параллельная** часть кода
* **CPU** и **GPU** обладают **независимой раздельной памятью**

Основные принципы

* Код на GPU называется **ядром** (**kernel**)

Основные принципы

* **Код** на **GPU** называется **ядром** (**kernel**)
* Ядро выполняется параллельно множеством **нитей/потоков** (**threads**)

Основные принципы

* **Код** на **GPU** называется **ядром** (**kernel**)
* Ядро выполняется параллельно множеством **нитей/потоков** (**threads**)
* Каждый поток выполняет **один и тот же код**

Основные принципы

* **Код** на **GPU** называется **ядром** (**kernel**)
* Ядро выполняется параллельно множеством **нитей/потоков** (**threads**)
* Каждый поток выполняет **один и тот же код**
* **Разница** между **нитями CPU** и **GPU:**

Основные принципы

* Код на GPU называется **ядром** (**kernel**)
* Ядро выполняется параллельно множеством **нитей/потоков** (**threads**)
* Каждый поток выполняет **один и тот же код**
* Разница между нитями CPU и GPU:
  * Нить GPU очень **легкая, контекст минимален,** выделение происходит **быстро**

Основные принципы

* Код на GPU называется **ядром** (**kernel**)
* Ядро выполняется параллельно множеством **нитей/потоков** (**threads**)
* Каждый поток выполняет **один и тот же код**
* Разница между нитями CPU и GPU:
  * Нить GPU очень **легкая, контекст минимален,** выделение происходит **быстро**
  * Эффективное использование GPU – вызов **тысячи нитей**

Основные принципы

* Код на GPU называется **ядром** (**kernel**)
* Ядро выполняется параллельно множеством **нитей/потоков** (**threads**)
* Каждый поток выполняет **один и тот же код**
* Разница между нитями CPU и GPU:
  * Нить GPU очень **легкая, контекст минимален,** выделение происходит **быстро**
  * Эффективное использование GPU – вызов **тысячи нитей**
  * Максимальная эффективность на CPU – количество **нитей равно числу ядер,** или кратно больше

Основные принципы

* Работа нитей соответствует принципу **SIMD**

Основные принципы

* Работа нитей соответствует принципу **SIMD**
* Однако, только нити в пределах одной группы (**warp**) выполняются **физически одновременно**

Основные принципы

* Работа нитей соответствует принципу **SIMD**
* Однако, только нити в пределах одной группы (**warp**) выполняются **физически одновременно**
* В архитектуре Fermi **warp = 32 threads**

Основные принципы

* Работа нитей соответствует принципу **SIMD**
* Однако, только нити в пределах одной группы (**warp**) выполняются **физически одновременно**
* В архитектуре Fermi **warp = 32 threads**
* **Управление** работой **варпов** – на **аппаратном** уровне

Основные принципы

* Работа нитей соответствует принципу **SIMD**
* Однако, только нити в пределах одной группы (**warp**) выполняются **физически одновременно**
* В архитектуре Fermi **warp = 32 threads**
* **Управление** работой **варпов** – на **аппаратном** уровне
* Каждый поток имеет **идентификатор**, позволяющий вычислить значение в памяти

Основные принципы

* Ядро запускает на выполнение **сетку (grid) блоков потоков (thread blocks)**

Основные принципы

* Ядро запускает на выполнение **сетку** (**grid**) **блоков потоков** (**thread blocks**)
  * Потоки внутри блока взаимодействуют посредством **разделяемой памяти** (быстрая память)

Основные принципы

* Ядро запускает на выполнение **сетку** (**grid**) **блоков потоков** (**thread blocks**)
  * Потоки внутри блока взаимодействуют посредством **разделяемой памяти** (быстрая память)
  * Потоки внутри блока могут **синхронизироваться** (не все потоки блока выполняются физически одновременно)

Основные принципы

* Ядро запускает на выполнение **сетку** (**grid**) **блоков потоков** (**thread blocks**)
  * Потоки внутри блока взаимодействуют посредством **разделяемой памяти** (быстрая память)
  * Потоки внутри блока могут **синхронизироваться** (не все потоки блока выполняются физически одновременно)
  * Потоки, находящиеся в различных блоках, **не могут** взаимодействовать

Основные принципы

* Такая модель позволяет **прозрачно масштабировать** программы на различные GPU

Основные принципы

* Такая модель позволяет **прозрачно масштабировать** программы на различные GPU
* **Блоки** потоков назначаются на **любой** из доступных **мультипроцессоров**

Основные принципы

* Такая модель позволяет **прозрачно масштабировать** программы на различные GPU
* **Блоки** потоков назначаются на **любой** из доступных **мультипроцессоров**
* **Размеры** блоков и сеток задаются **программистом**

Основные принципы

* Такая модель позволяет **прозрачно масштабировать** программы на различные GPU
* **Блоки** потоков назначаются на **любой** из доступных **мультипроцессоров**
* **Размеры** блоков и сеток задаются **программистом**
* Разумный **компромисс** между **взаимодействием** потоков и **быстродействием**

Основные принципы

* **Поток** GPU имеет **координаты** во вложенных трехмерных **декартовых** равномерных сетках **«индексы блоков»** и **«индексы потоков внутри каждого блока»**

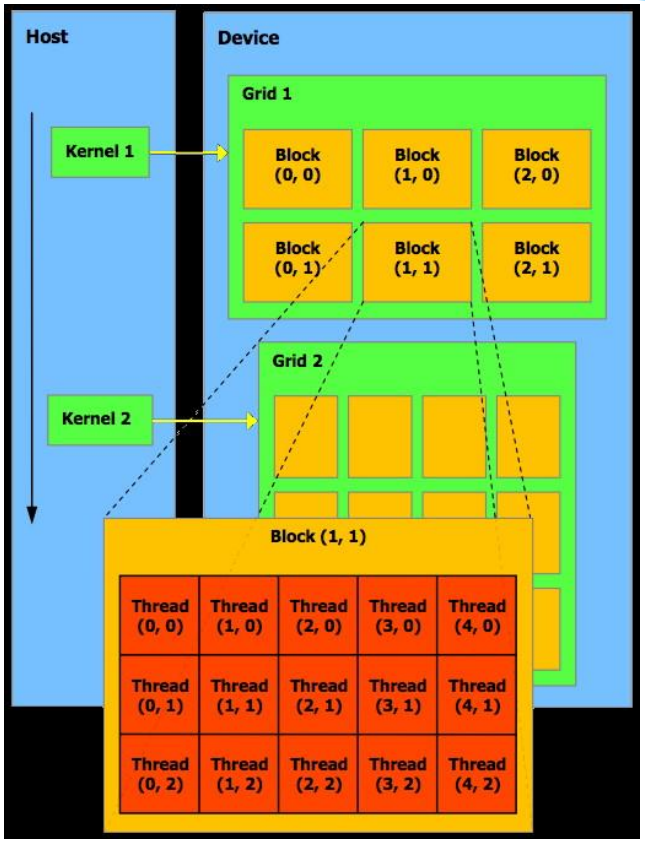
Основные принципы

* **Поток** GPU имеет **координаты** во вложенных трехмерных **декартовых** равномерных сетках **«индексы блоков»** и **«индексы потоков внутри каждого блока»**
* В **контексте** каждого потока значения **координат** и размерностей доступны через **встроенные переменные** threadIdx, blockIdx и blockDim, gridDim

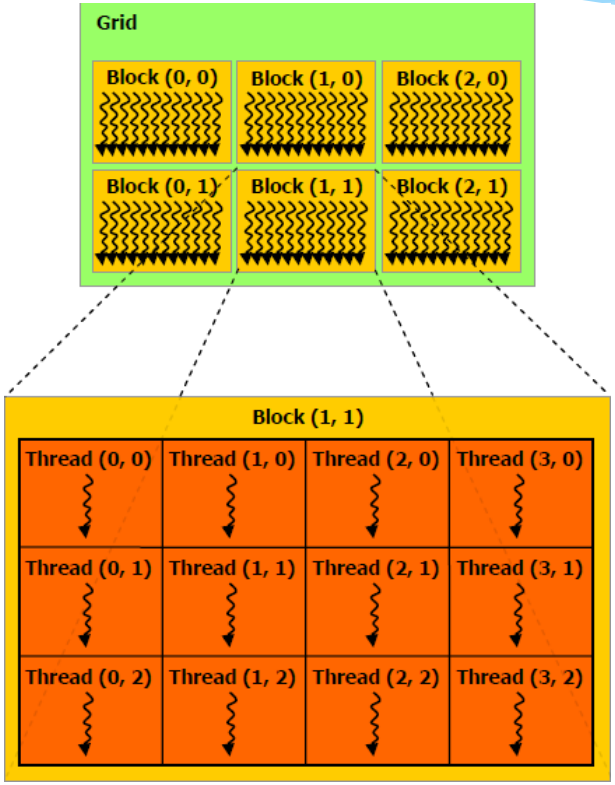
Основные принципы

* **Поток** GPU имеет **координаты** во вложенных трехмерных **декартовых** равномерных сетках **«индексы блоков»** и **«индексы потоков внутри каждого блока»**
* В **контексте** каждого потока значения **координат** и размерностей доступны через **встроенные переменные** threadIdx, blockIdx и blockDim, gridDim
* По **аналогии** с функциями **MPI** – **MPI_Comm_rank** и **MPI_Comm_size**

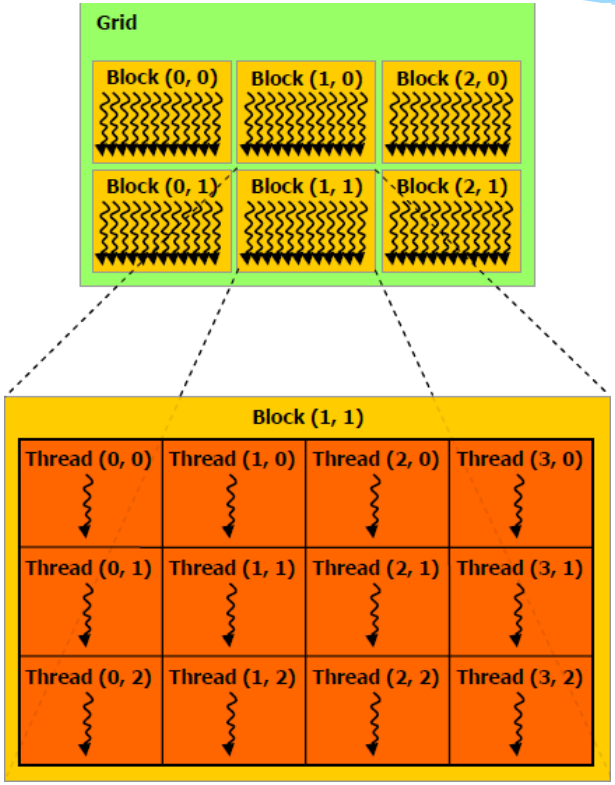
Потоки, блоки, сетки



Потоки, блоки, сетки



Потоки, блоки, сетки



CUDA «Hello, world»

CUDA «Hello, world»

* Программа складывает 2 массива
* Код ядра начинается с определения глобального индекса массива, зависящего от координат нити
* Соответствие нитей и частей задачи может быть любым, например, одна нить может обрабатывать не один элемент массива, а определенный диапазон
* Глобальной памятью GPU можно управлять с хоста

CUDA «Hello, world»

* В функции sum память выделяемая на GPU заполняется копией данных из памяти хоста, затем производится запуск ядра sum_kernel, синхронизация и копирование результатов обратно в память хоста
* В конце производится высвобождение ранее выделенной глобальной памяти GPU
* Такая **последовательность** действий **характерна** для любого **CUDA-приложения**

Расширения языка

* **Атрибуты функций** – показывают где будет выполняться функция и откуда она может быть вызвана

Расширения языка

* **Атрибуты функций** – показывают где будет выполняться функция и откуда она может быть вызвана
* **Атрибуты переменных** – задают тип используемой памяти

Расширения языка

* **Атрибуты функций** – показывают где будет выполняться функция и откуда она может быть вызвана
* **Атрибуты переменных** – задают тип используемой памяти
* **Встроенные переменные** – содержат контекстную информацию относительно текущей нити

Расширения языка

* **Атрибуты функций** – показывают где будет выполняться функция и откуда она может быть вызвана
* **Атрибуты переменных** – задают тип используемой памяти
* **Встроенные переменные** – содержат контекстную информацию относительно текущей нити
* **Дополнительные типы данных** – определяют несколько новых векторных типов

Расширения языка

* **Атрибуты функций** – показывают где будет выполняться функция и откуда она может быть вызвана
* **Атрибуты переменных** – задают тип используемой памяти
* **Встроенные переменные** – содержат контекстную информацию относительно текущей нити
* **Дополнительные типы данных** – определяют несколько новых векторных типов
* **Оператор запуска ядра** – определяет иерархию нитей, очередь команд и размер разделяемой памяти

Атрибуты функций

Атрибут	Функция выполняется на	Функция вызывается из
__device__	device (GPU)	device (GPU)
__global__	device (GPU)	host (CPU)
__host__	host (CPU)	host (CPU)

Атрибуты функций

* Атрибут **__global__** обозначает **ядро**, и соответствующая **функция** CUDA C должна возвращать значение типа **void**

Атрибуты функций

* Атрибут **__global__** обозначает **ядро**, и соответствующая **функция** CUDA C должна возвращать значение типа **void**
* На функции **__device__** и **__global__** накладываются ограничения:

Атрибуты функций

* Атрибут **__global__** обозначает **ядро**, и соответствующая **функция** CUDA C должна возвращать значение типа **void**
* На функции **__device__** и **__global__** накладываются ограничения:
  * **не поддерживаются static**-переменные внутри функции

Атрибуты функций

* Атрибут **__global__** обозначает **ядро**, и соответствующая **функция** CUDA C должна возвращать значение типа **void**
* На функции **__device__** и **__global__** накладываются ограничения:
  * **не поддерживаются static**-переменные внутри функции
  * **не поддерживается переменное** число входных **аргументов**

Атрибуты переменных

Атрибут	Размещение	Доступна	Вид доступа
__device__	device (GPU)	device (GPU)	R
__constant__	device (GPU)	device (GPU)/host (CPU)	R/W
__shared__	device (GPU)	block	RW

Атрибуты переменных

* Атрибуты **не могут** быть применены к полям структуры (**struct** или **union**)

Атрибуты переменных

* Атрибуты **не могут** быть применены к полям структуры (**struct** или **union**)
* Переменные могут использоваться только в пределах одного файла, их **нельзя** объявлять как **extern**

Атрибуты переменных

* Атрибуты **не могут** быть применены к полям структуры (**struct** или **union**)
* Переменные могут использоваться только в пределах одного файла, их **нельзя** объявлять как **extern**
* **Запись** в переменные типа **__constant__** может осуществляться **только CPU** при помощи специальных функций

Атрибуты переменных

* **__shared__** переменные **не могут** инициализироваться при **объявлении**

Атрибуты переменных

* **__shared__** переменные **не могут** инициализироваться при **объявлении**
* Каждая __global__ функция **должна находиться в одном исходном файле** вместе со всеми __device__ функциями и переменными, которые она использует

Встроенные типы

* **1/2/3/4**-мерные **векторные** типы на **основе** char, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long, longlong, float и double

Встроенные типы

* **1/2/3/4**-мерные **векторные** типы на **основе** char, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long, longlong, float и double
* **Компоненты** векторных типов имеют имена **x, y, z и w**

Встроенные типы

* **1/2/3/4**-мерные **векторные** типы на **основе** char, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long, longlong, float и double
* **Компоненты** векторных типов имеют имена **x, y, z и w**
* **Пример**:
  * int2 a = make_int2 (1, 7); // Создает вектор (1, 7)
  * float3 u = make_float3 (1, 2, 3.4f); // Создает вектор (1.0f, 2.0f, 3.4f)

Встроенные типы

* Для этих типов **не поддерживаются** векторные операции, так, например сложение двух векторов надо проводить **поэлементно**

Встроенные типы

* Для этих типов **не поддерживаются** векторные операции, так, например сложение двух векторов надо проводить **поэлементно**
* Добавлен тип **dim3**, используемый для задания **размерностей блоков** потоков и **сеток** блоков

Встроенные типы

* Для этих типов **не поддерживаются** векторные операции, так, например сложение двух векторов надо проводить **поэлементно**
* Добавлен тип **dim3**, используемый для задания **размерностей блоков** потоков и **сеток** блоков
* Этот тип **основан** на **uint3**, элементы по **умолчанию** инициализируются **1:**
  * dim3 blocks (16, 16); // Эквивалентно blocks (16, 16, 1)
  * dim3 grid (256); // Эквивалентно grid (256, 1, 1)

Встроенные переменные

* **gridDim** – размер **сетки** (имеет тип dim3)

Встроенные переменные

* **gridDim** – размер **сетки** (имеет тип dim3)
* **blockDim** – размер **блока** (имеет тип dim3)

Встроенные переменные

* **gridDim** – размер **сетки** (имеет тип dim3)
* **blockDim** – размер **блока** (имеет тип dim3)
* **blockIdx** – **индекс** текущего **блока** в сетке (имеет тип uint3)

Встроенные переменные

* **gridDim** – размер **сетки** (имеет тип dim3)
* **blockDim** – размер **блока** (имеет тип dim3)
* **blockIdx** – **индекс** текущего **блока** в сетке (имеет тип uint3)
* **threadIdx** – **индекс** текущего **потока** в блоке (имеет тип uint3)

Встроенные переменные

* **gridDim** – размер **сетки** (имеет тип dim3)
* **blockDim** – размер **блока** (имеет тип dim3)
* **blockIdx** – **индекс** текущего **блока** в сетке (имеет тип uint3)
* **threadIdx** – **индекс** текущего **потока** в блоке (имеет тип uint3)
* **warpSize** – размер **warp'a** (имеет тип int)

Оператор вызова ядра

* **kernel_name <<<Dg,Db,Ns,S>>> (args)**

Оператор вызова ядра

* **kernel_name <<<Dg,Db,Ns,S>>> (args)**
* **kernel_name** – это имя или адрес соответствующей __global__ функции

Оператор вызова ядра

* **kernel_name <<<Dg,Db,Ns,S>>> (args)**
* **kernel_name** – это имя или адрес соответствующей __global__ функции
* **Параметр Dg** типа dim3 задает размерности сетки блоков (число блоков в сетке блоков)

Оператор вызова ядра

* **kernel_name <<<Dg,Db,Ns,S>>> (args)**
* **kernel_name** – это имя или адрес соответствующей __global__ функции
* **Параметр Dg** типа dim3 задает размерности сетки блоков (число блоков в сетке блоков)
* **Параметр Db** типа dim3 задает размерности блока нитей (число потоков в блоке)

Оператор вызова ядра

* **Необязательный параметр Ns** типа size_t задает дополнительный объем разделяемой памяти в байтах (по умолчанию – 0), которая должна быть динамически выделена каждому блоку (в дополнение к статически выделенной)

Оператор вызова ядра

* **Необязательный параметр Ns** типа size_t задает дополнительный объем разделяемой памяти в байтах (по умолчанию – 0), которая должна быть динамически выделена каждому блоку (в дополнение к статически выделенной)
* **Параметр S** типа cudaStream_t ставит вызов ядра в определенную очередь команд (CUDA Stream), по умолчанию – 0

Оператор вызова ядра

* **Вызов функции-ядра** также может иметь произвольное фиксированное число **параметров,** суммарный **размер** которых **не превышает 4 Кбайт**

Оператор вызова ядра

* **Вызов функции-ядра** также может иметь произвольное фиксированное число **параметров**, суммарный **размер** которых **не превышает 4 Кбайт**
* **Пример –** n **потоков, блоки** 16x16**, сетка** из n/256 блоков, 2 **параметра:**
  * my_kernel<<<dim3(n/256),dim3(16,16),512,my_stream>>>(a, n)

Встроенные функции

* Для **CUDA** реализованы **математические функции,** совместимые с **ISO C**
* Также имеются соответствующие **аналоги,** вычисляющие результат с **пониженной точностью** например, ***__sinf*** для ***sin***
* Полный **список** функций – в **документации** CUDA Toolkit

CUDA runtime API

* CUDA Runtime API – **библиотека функций,** обеспечивающих:
  * управление GPU
  * работу с контекстом
  * работу с памятью
  * работу с модулями
  * управление выполнением кода
  * работу с текстурами
  * взаимодействие с OpenGL и Direct3D

CUDA runtime API

* CUDA Runtime API **делится** на **два** уровня: **driver API** и **CUDA API**

CUDA runtime API

* CUDA Runtime API **делится** на **два** уровня: **driver API** и **CUDA API**
* **driver API** является более **низкоуровневым** и требует **явной инициализации** устройства

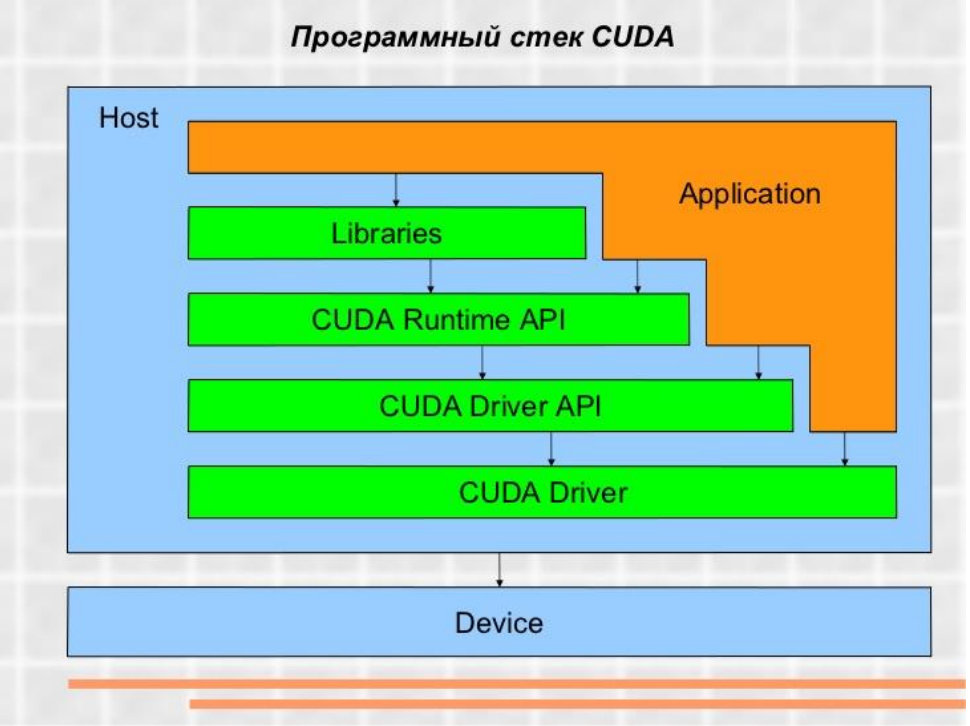
CUDA runtime API

* CUDA Runtime API **делится** на **два** уровня: **driver API** и **CUDA API**
* **driver API** является более **низкоуровневым** и требует **явной инициализации** устройства
* В **CUDA API инициализация** происходит **неявно** при первом **вызове** любой функции **библиотеки**

CUDA runtime API

* CUDA Runtime API **делится** на **два** уровня: **driver API** и **CUDA API**
* **driver API** является более **низкоуровневым** и требует **явной инициализации** устройства
* В **CUDA API инициализация** происходит **неявно** при первом **вызове** любой функции **библиотеки**
* Напрямую с **устройством** работает **драйвер,** на нем **базируется** CUDA Runtime API

Программный стек CUDA



Асинхронное исполнение

* Многие **функции** CUDA являются **асинхронными, управление** в вызывающую функцию **возвращается** до **завершения** требуемой операции:
  * Запуск ядра
  * Функции копирования и инициализации памяти, имена которых оканчиваются Async
  * Функции копирования памяти device ↔ device внутри устройства и между устройствами

Асинхронное исполнение

* С **асинхронностью** связаны объекты **CUDA streams** (потоки исполнения), позволяющие группировать последовательности операций, которые необходимо выполнять в строго определенном порядке

Асинхронное исполнение

* С **асинхронностью** связаны объекты **CUDA streams** (потоки исполнения), позволяющие группировать последовательности операций, которые необходимо выполнять в строго определенном порядке
* При этом **порядок** выполнения операций между **разными** CUDA streams **не является** строго **определенным** и может изменяться

Обработка ошибок

* **Каждая функция** CUDA runtime API (кроме запуска ядра) **возвращает** значение типа $\boldsymbol{cudaError\_t}$

Обработка ошибок

* **Каждая функция** CUDA runtime API (кроме запуска ядра) **возвращает** значение типа $\boldsymbol{cudaError\_t}$
* При **успешном** выполнении функции возвращается значение $\boldsymbol{cudaSuccess}$**, иначе возвращается код ошибки**

Обработка ошибок

* **Каждая функция** CUDA runtime API (кроме запуска ядра) **возвращает** значение типа ***cudaError_t***
* При **успешном** выполнении функции возвращается значение ***cudaSuccess*, иначе возвращается код ошибки**
* **Текстовое описание** ошибки и **последняя** ошибка:
  * char* cudaGetErrorString (cudaError_t code);
  * cudaError_t cudaGetLastError();

Компиляция CUDA ядра

* **nvcc – компилятор** для CUDA, исходные файлы ***.cu**
* Основные опции командной строки – в документации CUDA Toolkit
* Работа с компилятором похожа на работу с gcc
* Пример компиляции одного исходного файла:
  * nvcc -arch=sm_20 –O2 -o test test.cu

To be continued…