Разработка параллельных программ для GPU

Введение в CUDA

Краткий обзор архитектурных особенностей GPU

АППАРАТНЫЕ ОСОБЕННОСТИ GPU

Основные тенденции

- Переход к многопроцессорным системам
- Развития технологий параллельного программирования
  - OpenMP, MPI, TPL etc.
  - **Простота в использовании**

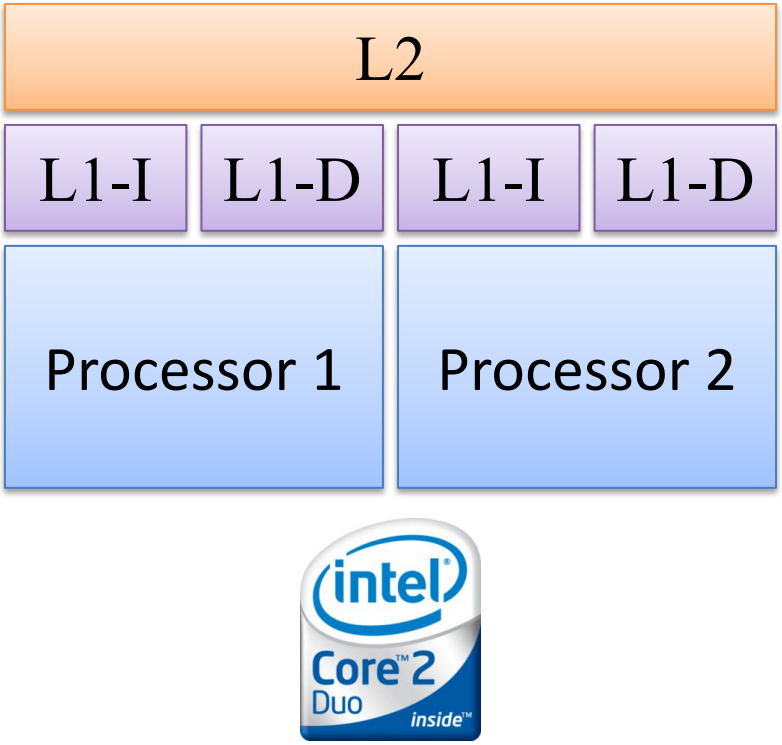
Классификация архитектур

- Виды параллелизма
  - На уровне данных (Data)
  - На уровне задач (Instruction)

	Single Instruction (SI)	Multiple Instruction (MI)
Single Data (SD)	**SISD**	MISD
Multiple Data (MD)	**SIMD**	MIMD

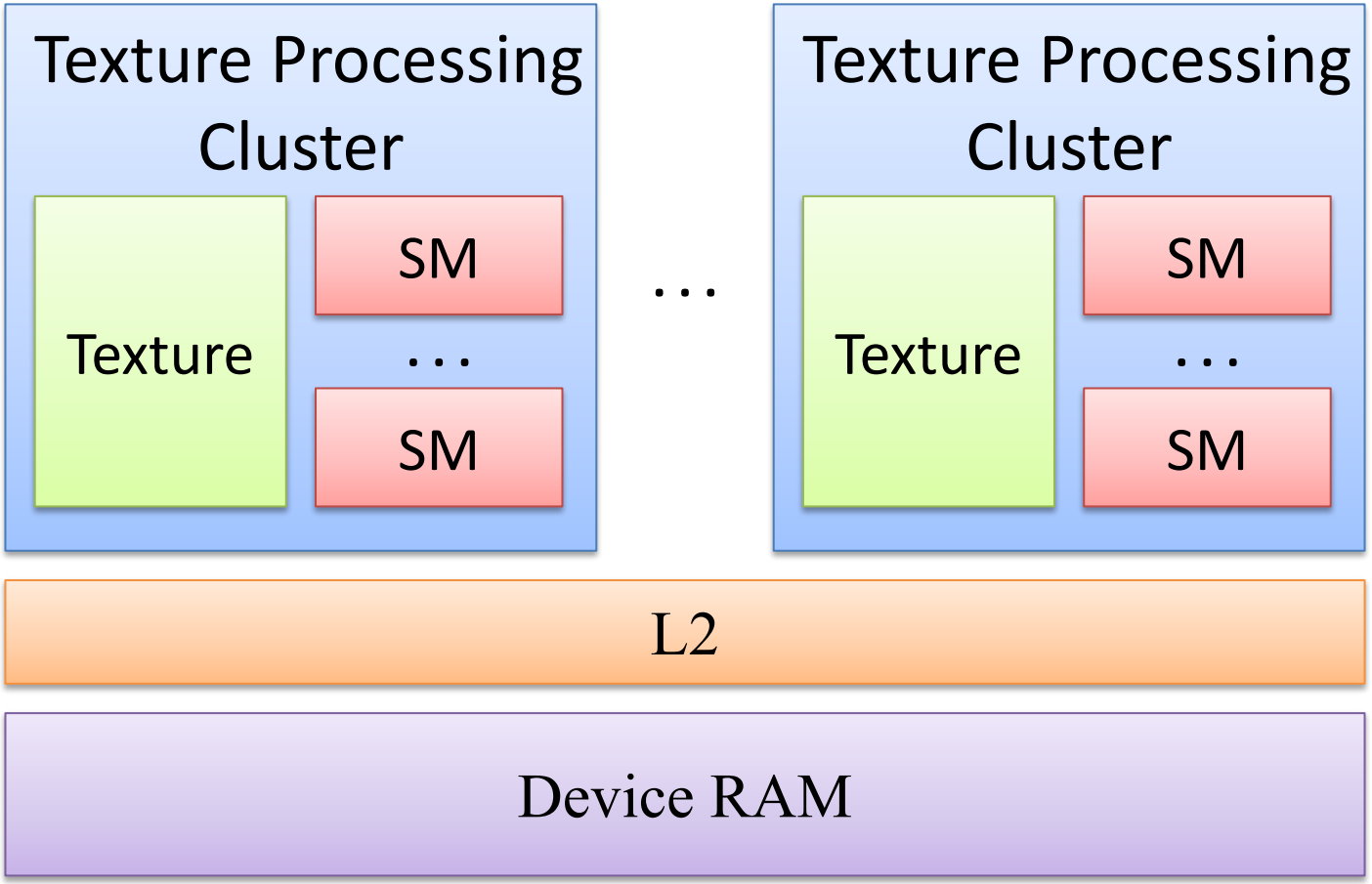
GPU: SIMT** – Single Instruction Multiple **Thread***

Архитектура многоядерных CPU

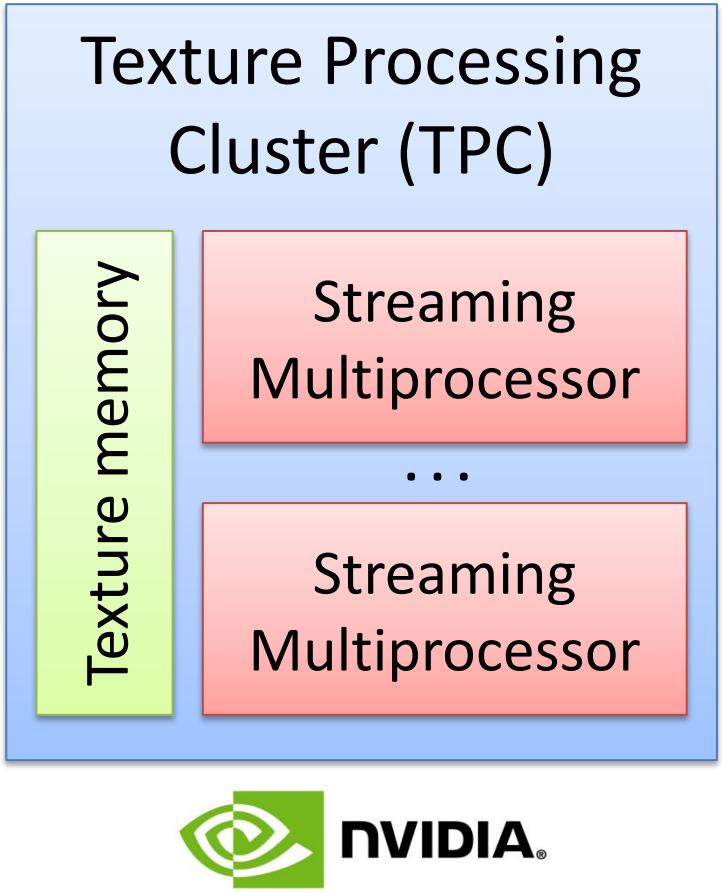


- Кэш первого уровня
  - для инструкций (L1-I)
  - для данных (L1-D)
- Кэш второго уровня
  - на одном кристалле
  - используется раздельно
- **Проблема синхронизации кэш-памяти**

Архитектура GPU: Device

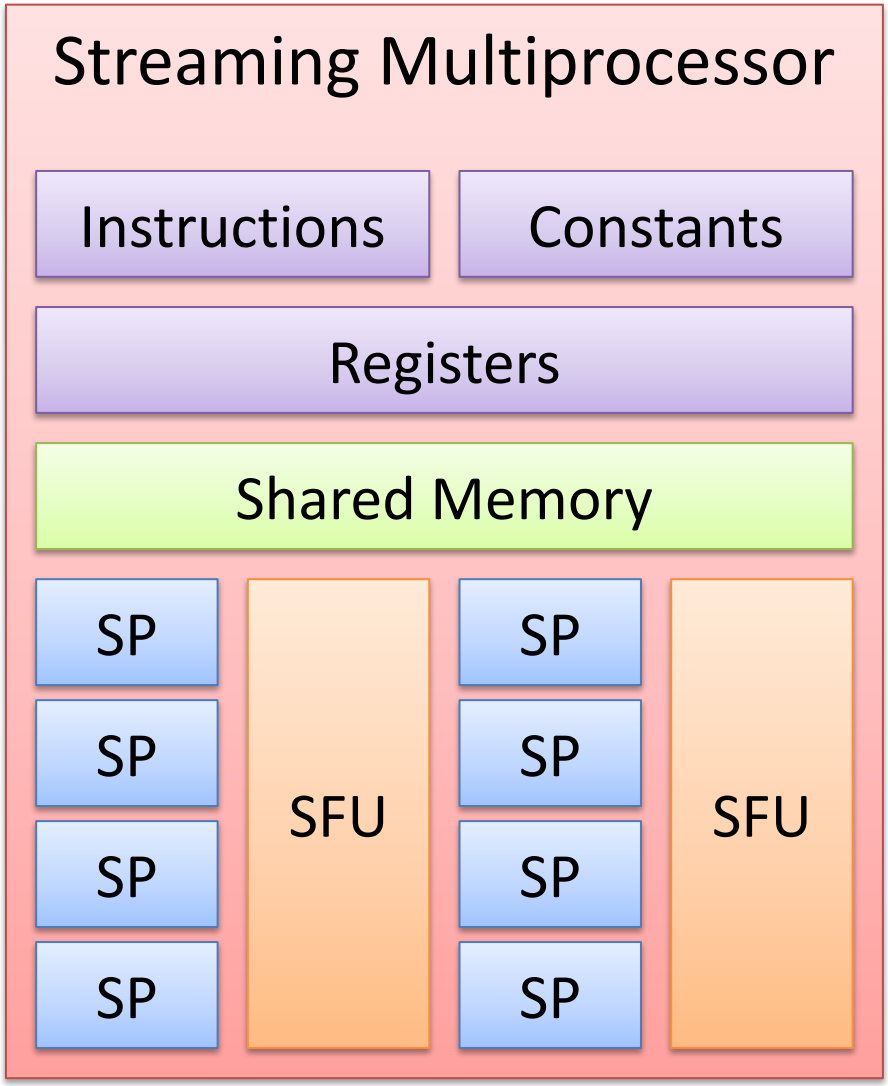


Архитектура GPU: TPC



- Кластер текстурных блоков (TPC)
  - Память для текстур
  - Потоковый мультипроцессор

Архитектура GPU: SM



- Память констант
- Память инструкций
- Регистровая память
- **Разделяемая память**

- 8 скалярных процессоров

- 2 суперфункциональных блока

NVIDIA

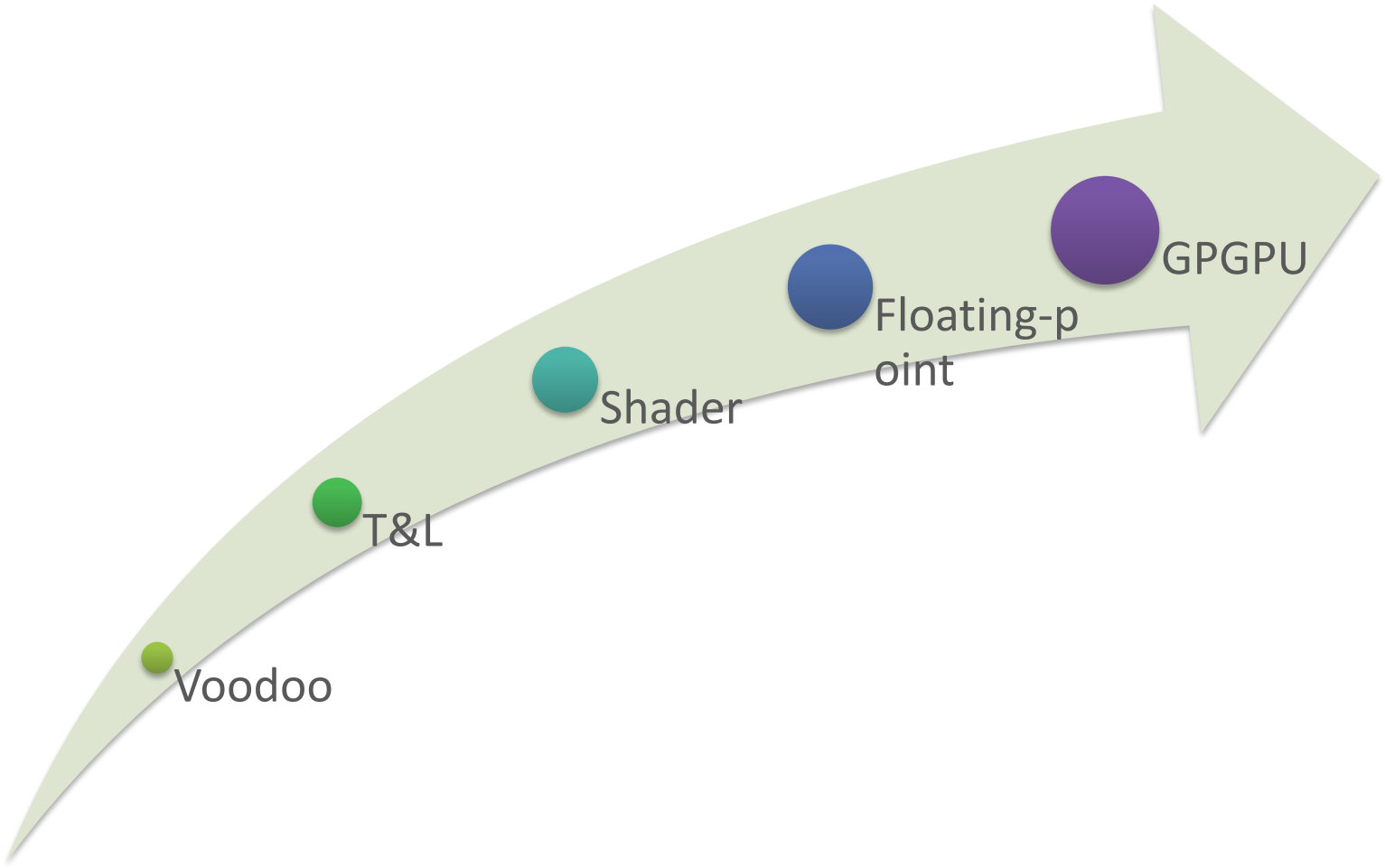
Основные отличия GPU от CPU

- Высокая степень параллелизма (SIMT)
- Минимальные затраты на кэш-память
- Ограничения функциональности

Развитие технологии неграфических вычислений

РАЗВИТИЕ ВЫЧИСЛЕНИЙ НА GPU

Эволюция GPU



GPGPU

- General-Purpose Computation on GPU
  - Вычисления на GPU общего (неграфического) назначения
  - AMD FireStream
  - **NVIDIA CUDA**
  - DirectCompute (DirectX 10)
  - **OpenCL**

OpenCL

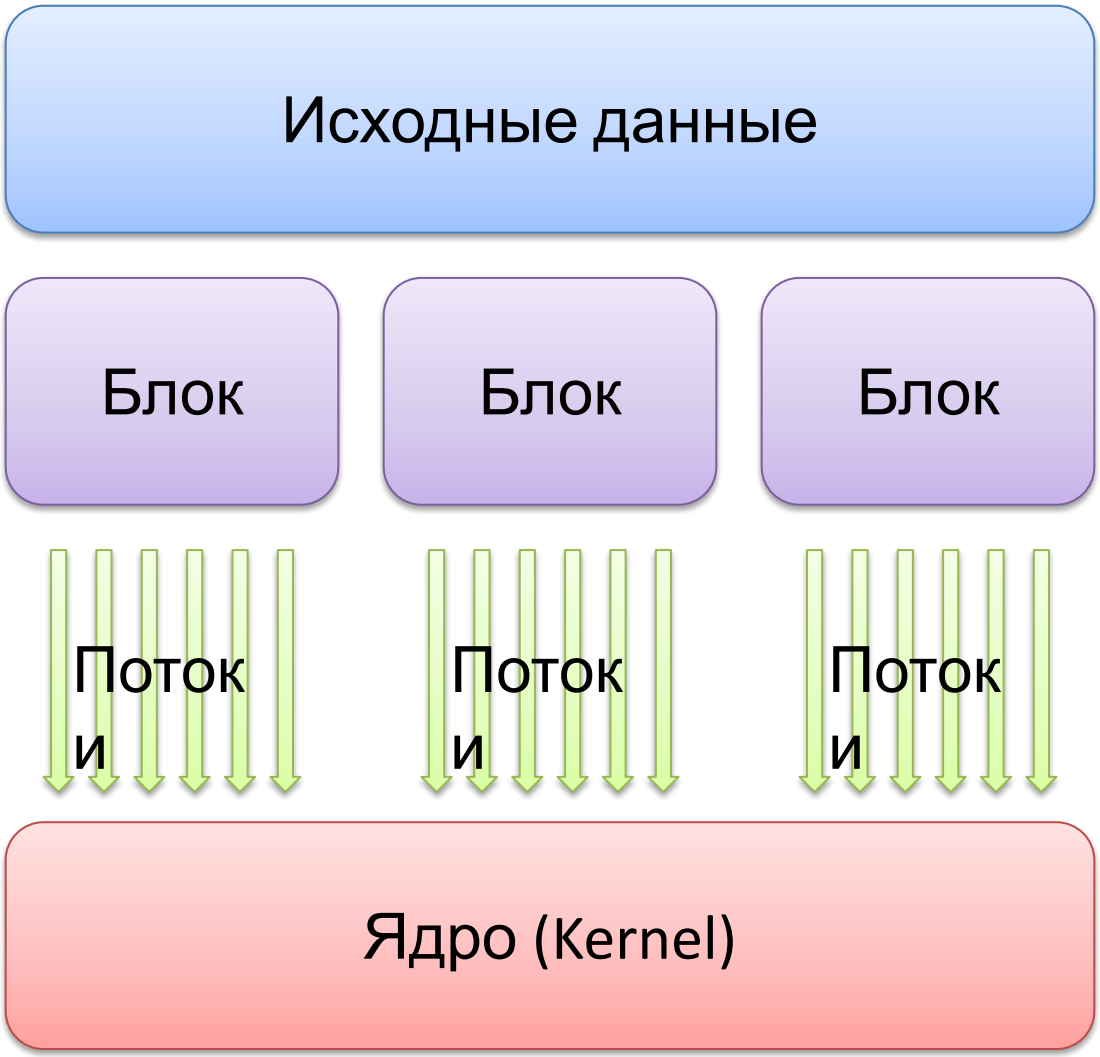
Основные понятия и определения CUDA

ПРОГРАММНАЯ МОДЕЛЬ CUDA

CUDA – Compute Unified Device Architecture

- **Host** – CPU (Central Processing Unit)
- **Device** – GPU (Graphics Processing Unit)

Организация работы CUDA GPU

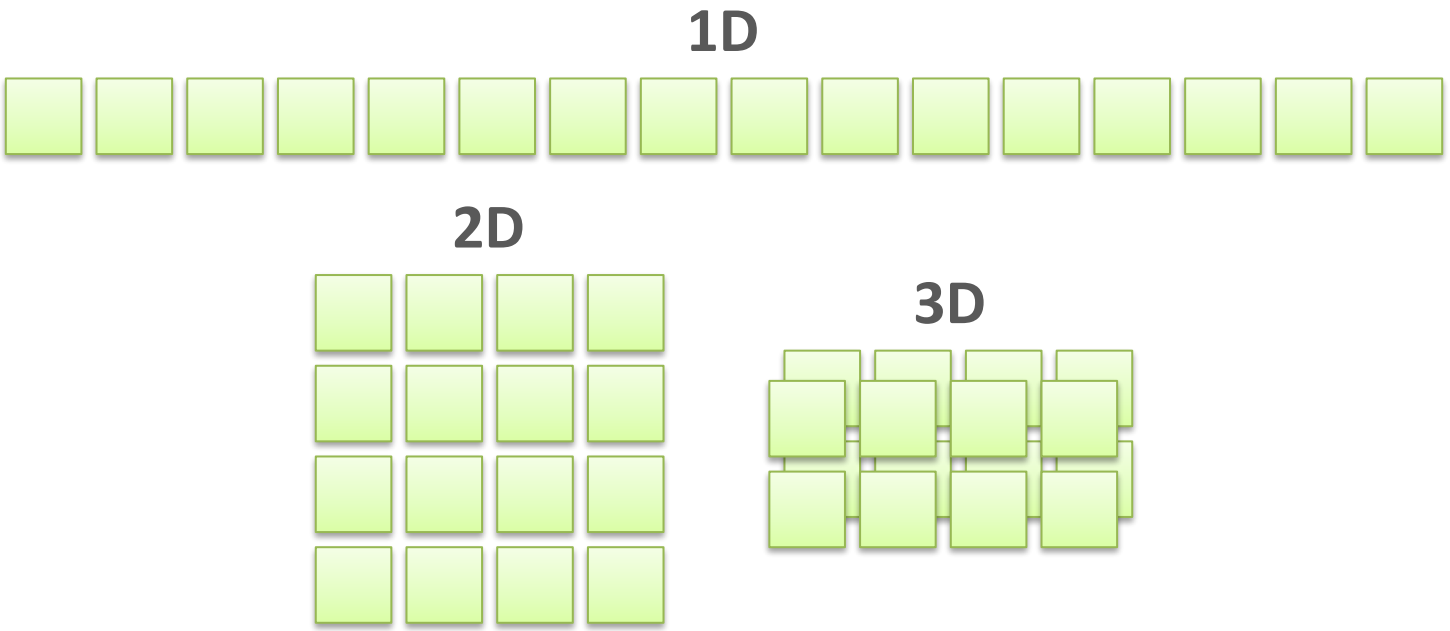


Warp и латентность

- Warp
  - Порция потоков для выполнения на потоковом мультипроцессоре (SM)
- Латентность
  - Общая задержка всех потоков warp’а при выполнении инструкции

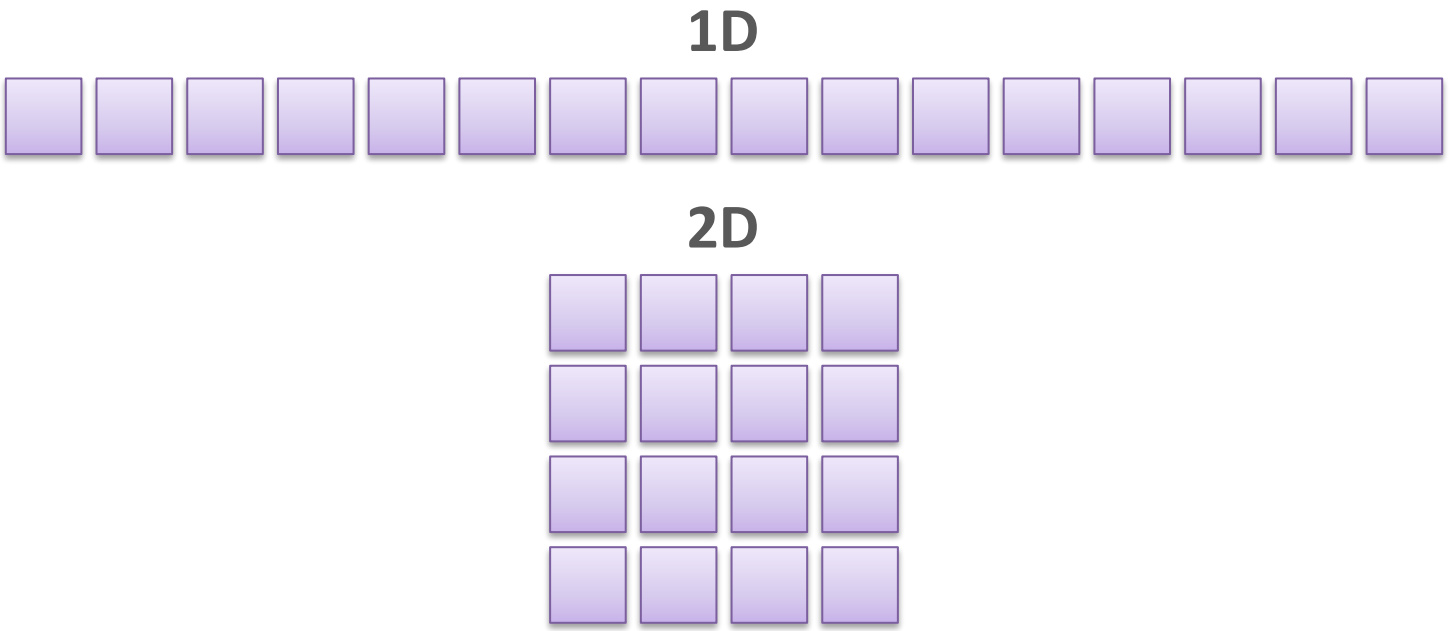
Топология блоков (block)

- Возможна 1, 2 и 3-мерная топология
- Количество потоков в блоке ограничено (512)



Топология сетки блоков (grid)

- Возможна 1 и 2-мерная топология
- Количество блоков в каждом измерении ограничено $65536=2^{16}$



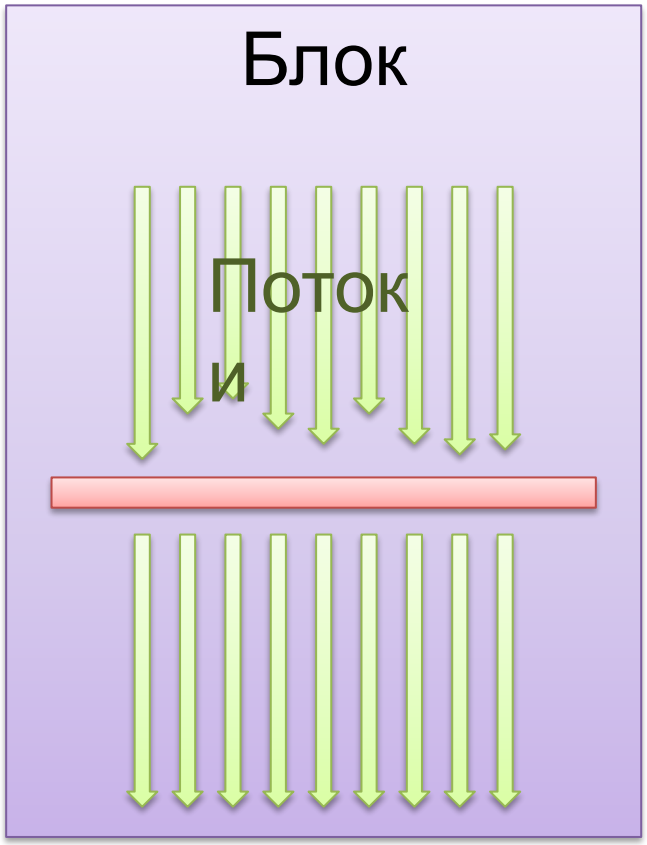
Адресация элементов данных

- CUDA предоставляет встроенные переменные, которые идентифицируют блоки и потоки
  - **blockIdx**
  - **blockDim**
  - **threadIdx**

1D Grid & 2D Block:

```
int dataIdnex = blockIdx.x * blockDim.x + threadIdx.x
```

Барьерная синхронизация



- Синхронизация потоков блока осуществляется встроенным оператором **__synchronize**

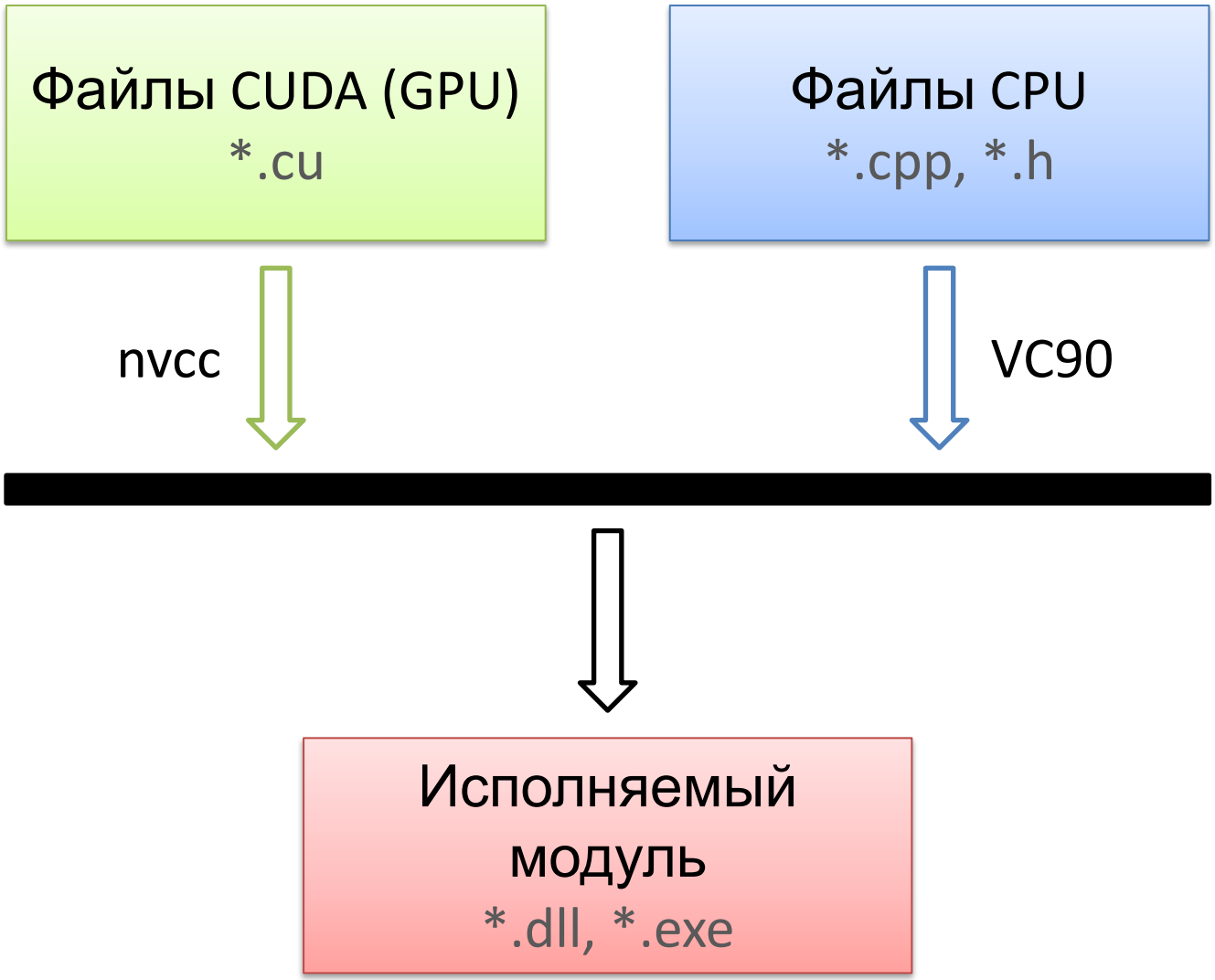
Особенности написания программ для GPU CUDA

CUDA: РАСШИРЕНИЕ C++

Расширение языка C++

- Новые типы данных
- Спецификаторы для функций
- Спецификаторы для переменных
- Встроенные переменные (для ядра)
- Директива для запуска ядра

Процесс компиляции



Типы данных CUDA

- 1, 2, 3 и 4-мерные вектора базовых типов
  - Целые: (u)char, (u)int, (u)short, (u)long, longlong
  - Дробные: float, double
  - **Пример:** float(1), float2, float3, float4

- dim3 ~ uint3
  - **Пример:** dim3(n) = uint(n,1,1)

Спецификаторы функций

Спецификатор	Выполняется	Вызывается
__device__	device	device
__global__	device	host
__host__	host	host

Спецификаторы функций

- Ядро помечается __global__
- **Ядро не может возвращать значение**

- Возможно совместное использование __host__ и __device__

- Спецификаторы __global__ и __host__ не могут использоваться совместно

Ограничения функций GPU

- Не поддерживается рекурсия
- Не поддерживаются static-переменные
- Нельзя брать адрес функции __device__
- Не поддерживается переменное число аргументов

Спецификаторы переменных

Спецификатор	Находится	Доступна	Вид доступа
__device__	device	device	R
__constant__	device	device / host	R / R/W
__shared__	device	**block**	R/W

Ограничения переменных GPU

- Переменные __shared__ не могут инициализироваться при объявлении
- Запись в __constant__ может производить только host через CUDA API
- Спецификаторы нельзя применять к полям структур и union

Переменные ядра

- dim3 gridDim
- unit3 blockIdx
- dim3 blockDim
- uint3 threadIdx
- int warpSize

Директива запуска ядра

- Kernel<<<blocks, threads>>>(data)

  - blocks – число блоков в сетке
  - threads – число потоков в блоке

Общая структура программы CUDA

```
__global__ void Kernel(float* data)
{
    . . .
}

void main()
{
    . . .

    Kernel<<<blocks, threads>>>(data);

    . . .
}
```

Предустановки

- Видеокарта NVIDIA с поддержкой CUDA
- Драйвера устройства с поддержкой CUDA

- NVIDIA CUDA Toolkit
- NVIDIA CUDA SDK
- NVIDIA Nsight

- Visual Studio 2008+
- Компилятор Visual C++ 9.0+

Литература

- NVIDIA Developer Zone
  - http://developer.nvidia.com/cuda
- NVIDAI CUDA – Неграфические вычисления на графических процессорах
  - http://www.ixbt.com/video3/cuda-1.shtml
- Создание простого приложения CUDA в Visual Studio 2010
  - http://mezhov.blogspot.com/2011/09/cuda-visual-studio-2010.html

ВОПРОСЫ?