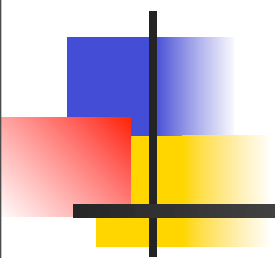
intro

v1.2

Программные интерфейсы CUDA



Лекция 2

Лектор: **Селиверстов Евгений Юрьевич**

Модель потоков

Поток (thread) – аппаратно-программный примитив, очень высокая скорость, выполняется простейшим АЛУ (SP) на ГП.

Ядро (kernel) – функция, выполняемая потоками.

Все ядра имеют одинаковый код, но различаются по данным и, быть может, потоку управления.

Основное отличие SIMT.

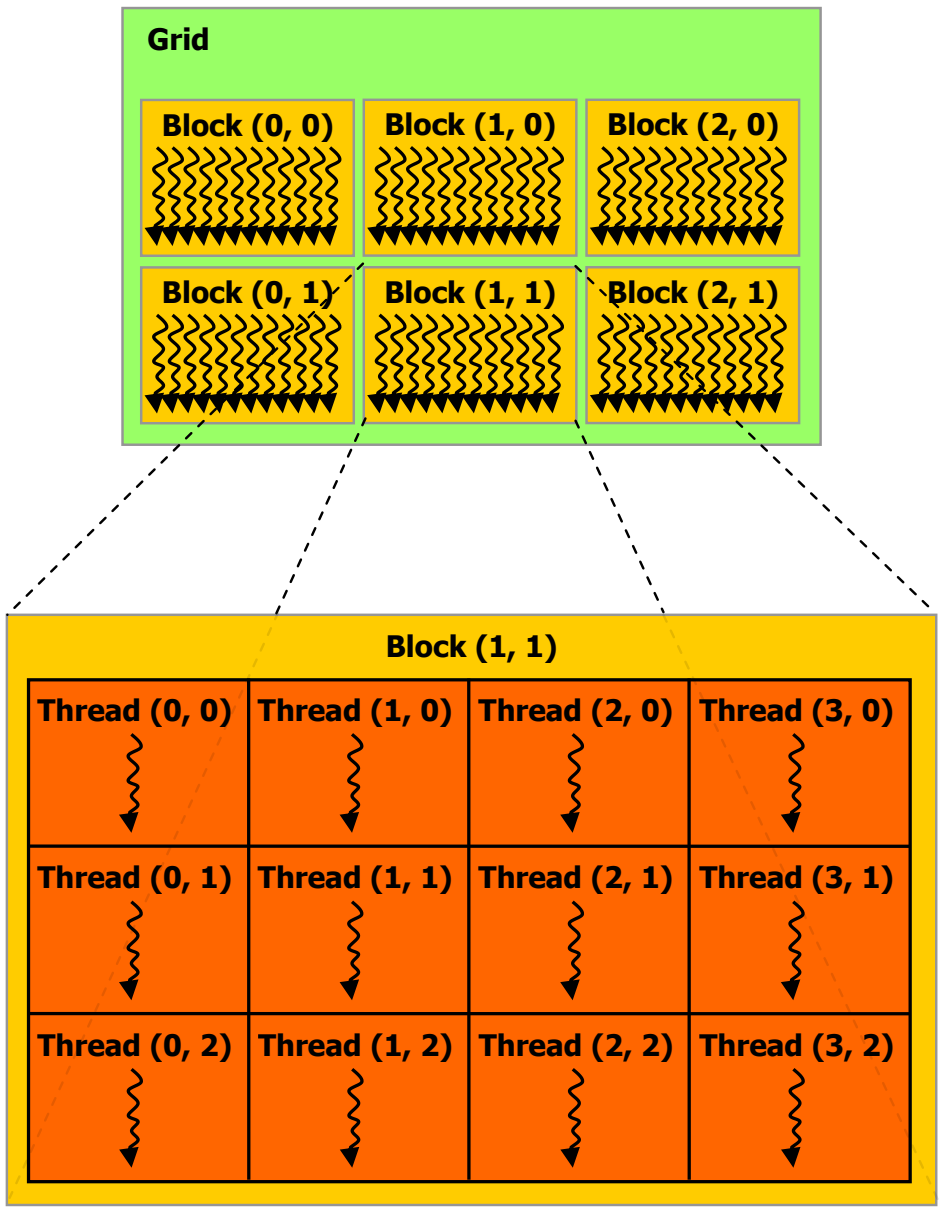
Модель потоков

Структура:
Потоки объединяются в блоки
Блоки объединяются в сеть

Коммуникация:
Коммуникация потоков в пределах блока через общую память и сихронизацию.
Коммуникация блоков невозможна.

Такая схема позволяет масштабирование сети или сетей по числу мультипроцессоров.

Модель потоков



Типы памяти

Все уровни модели памяти отображаются на конкретные типы памяти ГП

Уровни модели памяти:
- локальная
- разделяемая
- глобальная (в т.ч. текстурная, константная)

Аппаратная память:
- регистровая
- общая
- DRAM на MP
- DRAM на ГП

Типы аппаратной памяти

Регистровая память
- регистры для каждого SP
- мало
- наиболее быстрая

Локальная память
- DRAM (область, специфичная для MP)
- кэшируемая на fermi
- используется при недостатке регистров

Типы аппаратной памяти

Общая память
- регистры на MP
- 16 Кб на MP
- быстрая
- организация в виде банков

Глобальная память
- DRAM
- кэшируемая на fermi
- наиболее медленная

Модель памяти

CUDA – система с неоднородной памятью.

Уровни доступа:
1. поток:
- регистры (r/w)
- локальная память (r/w)
Время жизни: как у потока

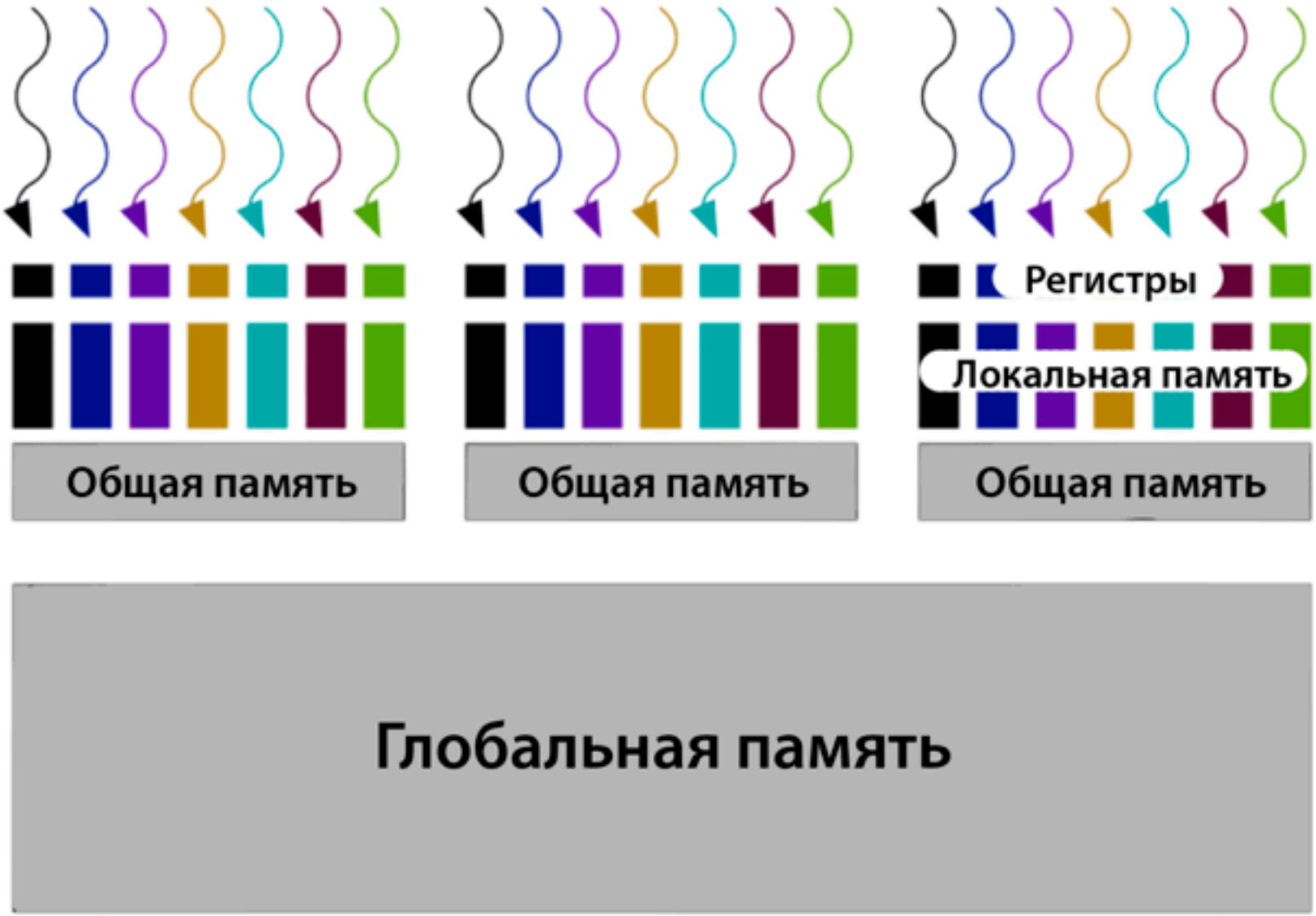
2. блоки:
- общая память (r/w)
Время жизни: как у блока

Модель памяти

3. сеть:
- глобальная память (r/w)
- константная память (r)
- текстурная память (r)

Время жизни: ручное выделение/освобождение

Модель памяти



Программирование CUDA

- Язык CUDA C
- Основные функции и типы данных
- Схема программ
- Управление памятью
- Управление выполнением
- Отладка

Программные интерфейсы

Интерфейсы:

- CUDA C - высокоуровневое API, диалект C
Библиотека: cudart, префиксы функций `cuda`

- Driver API - низкоуровневое API, независимое от языка
Библиотека: nvcuda, префиксы функций `cu`

- OpenCL

Всё построено над PTX.

CUDA C API

- диалект C/C++
компилируется с помощью nvcc
Добавлены дополнительные типы, спецификаторы, операторы

- библиотека CUDA runtime
Содержит функции по работе с устройством, памятью, потоками данных, вычислительными блоками, мостами с OpenGL/DirectX

Типы

Скалярные типы

`u?(char|short|int|long|float|double)`

Векторные типы

`u?(char|short|int|long|float|double)[234]`

Содержат поля `x,y,z,w`.

Размерность

`dim3`

Прочие типы

`cudaArray, texture` и т.д.

Встроенные переменные

`gridDim` - тип dim3

`blockIdx` - тип uint3

`blockDim` - тип dim3

`threadIdx` - тип uint3

`warpSize` - тип int

Спецификаторы функций

Записываются перед именем функции.

`__host__` (по умолчанию)
функция исполняется на хосте, вызывается с хоста

`__device__`
функция исполняется на устройстве, вызывается с устройства

`__global__`
функция исполняется на устройстве, вызывается с хоста

Спецификаторы переменных

Записываются перед именем переменной.

`__device__`
переменная в глобальной памяти

`__constant__`
переменная в константной памяти

`__shared__`
переменная в разделяемой памяти

Если нет спецификатора, то регистровая или локальная переменная

Запуск ядра

Специальный синтаксис запуска ядра

```
Kernel<<<DGrid, DBlock, M, S>>>(args)
```

Конфигурация выполнения:
- `DGrid` (dim3) - размер вычислительной сети
- `DBlock` (dim3) - размер вычислительного блока
- `M` (size_t) - объем разделяемой памяти блока
- `S` (cudaStream_t, опциональный) - поток
- args - аргументы, передаваемые через разделяемую память

Функция ядра - со спецификатором `__global__`

Запуск ядра

Ядра могут запускаться асинхронно - одновременно с кодом хоста.

Параллельное выполнение ядер - только в Fermi.

В процессе выполнения ядрам доступны индексы:
- `threadIdx` - индекс потока в блоке
- `blockIdx` - индекс блока в сети
- `blockDim` - размер блока в потоках
- `gridDim` - размер сети в блоках

Синхронизация

Синхронизация возможна только в пределах вычислительного блока.

Барьерная синхронизация

```
void __syncthreads()
```

Версия с предикатом (CC 2.0):

```
int __syncthreads(int pred)
```

Специальные функции

Голосование (warp vote, CC 1.2)

```
int __all (int pred)
int __any (int pred)
unsigned int __ballot (int pred)
```

Атомарные функции (CC 1.1)

```
int atomicAdd (int* ptr, int val)
```

а также

```
atomic(Sub, Exch, Min, Max, Inc, Dec, CAS)
```

Арифметические функции

Стандарт на вычисления с плавающей точкой IEEE 754.

Целочисленные вычисления: 24 и 32 бита
Вычисления с плавающей точкой:
32 и 64 бита (ср.: до 80 бит на CPU)

Типы:
- с одинарной точностью (float)
- с двойной (double, CC 1.2)

Более быстрые instrinsic-функции (префикс __).
Точность уточняется в документации.

Синтаксис C++

Полностью поддерживается в хост-коде.

Поддерживается в ядрах в CC 2.0:
- шаблоны
- перегрузка функций и операторов
- классы и наследование
- пространства имён
- параметры по умолчанию
- исключения

Библиотека Runtime API

Библиотека обеспечивает инициализацию, управление модулями (PTX-кодом), управление контекстами, эмуляция устройства.

Различные функции библиотеки могут быть вызваны из хост-программы или из ядра.

Префиксы функций: `cuda`

Возвращаемое значение - код ошибки `cudaError_t`

Адресные пространства

Адресные пространства хоста и GPU независимы.
Следует различать указатели на память хоста и память устройства.
Способ доступа к памяти устройства различается для глобальной, константной, текстурной и локальной.

В Fermi адресные пространства устройства и хоста общие.

Управление памятью

Выделение памяти на хосте

Адресное пространство CPU

Выделение:
- malloc – обычное выделение памяти
- cudaMallocHost
- cudaHostAlloc (далее)

Освобождение:
- free
- cudaFree

Управление памятью

- **Память хост-машины**
- Память графического процессора
- Копирование данных

Зафиксированная память

Фиксированные страницы виртуальной памяти

Не сбрасываются на диск
Позволяют драйверу осуществлять эффективное копирование

Обозначение в документации - page locked, pinned pages

Управление:

```
cudaHostAlloc(void** hostptr, size, flags)
cudaHostFree(void* hostptr)
cudaHostRegister()
```

Зафиксированная память

Виды:
- перемещаемая (cudaHostAllocPortable)
используется несколькими потоками хоста

- некэшируемая (cudaHostAllocWriteCombined)
отключено кэширование на ЦП
быстрая запись
медленное чтение

- отображаемая (cudaHostAllocMapped)
выделение памяти сразу в памяти устройства
отображаемые зафиксированные страницы

Зафиксированная память

Отображаемая память (cudaHostAllocMapped)

`cudaHostAlloc` – указатель на память хост
`cudaHostGetDevicePointer` - указатель на память ГП

Избегаются лишние вызовы функций копирования (`cudaMemcpy`)

Общее адресное пространство

CUDA 4.x + CC 2.0+ (Fermi) + 64-бит

- адресное пространство указателей одно (64-бит)
- даже для нескольких ГП
- выделяемая память portable
- копирование хост-ГП не нужно

Управление памятью

- Память хост-машины
- **Память графического процессора**
- Копирование данных

Управление глобальной памятью

Выделение памяти на устройстве

Адресное пространство GPU
Массивы

1. Тип данных:
- простой указатель
- массив cudaArray (для текстур)

2. Размерность массива

Управление глобальной памятью

Обычные массивы

одномерный массив:

```
cudaMalloc (void** devptr, size_t size)
```

двумерный массив:

```
cudaMallocPitch (void** devptr, size_t* pitch,
   size_t width, size_t height)
```

трехмерный массив:

```
cudaMalloc3D (cudaArray*;, channelFormat*,
   size_t width, size_t height)
```

Управление глобальной памятью

Массивы cudaArray

одномерный массив,
двумерный массив:

```
cudaMallocArray (cudaArray*, channelFormat*,
   size_t width, size_t height, int flags)
```

трехмерный массив:

```
cudaMalloc3DArray (cudaArray*, channelFormat*,
   cudaExtent extent, int flags)
```

Автоматическое выравнивание 2D

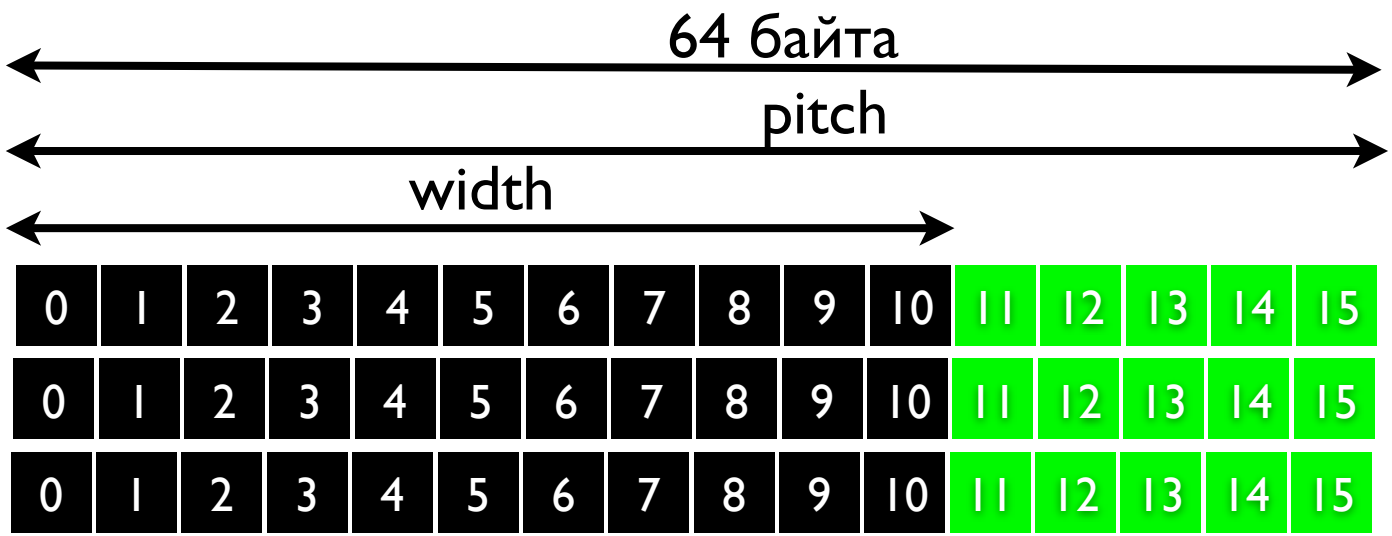
```
cudaMallocPitch (void** devptr, size_t* pitch,
    size_t width, size_t height)
```

Вход: `width, height`

Выход: `ptr, pitch`

Выравнивание в пределах транзакции памяти (32 или 64 байта).

```
T* elem = (T*)((char*)ptr + row * pitch) + col;
```



Автоматическое выравнивание 3D

```
cudaMalloc3D (cudaPitchedPtr* devptr,
    cudaExtent extent)
```

Вход: `cudaExtent { width, height, depth }`

Выход: `cudaPitchedPtr { ptr, pitch }`

Два питча:

- pitch по ширине (`pitch`)
- pitch по глубине (`pitch * height`)

```
T* elem = (T*)((ptr + z * pitch * height) +
                     row * pitch) +
                     col;
```

Управление глобальной памятью

Освобождение памяти:

обычные массивы:

```
cudaFree (void* devptr)
```

массивы cudaArray:

```
cudaFreeArray (cudaArray* devptr)
```

Копирование данных

- синхронное
- асинхронное

Синхронное копирование:

```
cudaMemcpy (void* dst, void* src,
    size_t count, enum cudaMemcpyKind kind)
```

Тип копирования:

- `cudaMemcpyHostToHost`
- `cudaMemcpyHostToDevice`
- `cudaMemcpyDeviceToHost`
- `cudaMemcpyDeviceToDevice`

Управление кэшем

Кэши глобальной памяти в архитектуре Fermi:
- L1 на разделяемой памяти мультипроцессора
- L2 для всех мультипроцессоров

Кэширование управляется флагами nvcc:
`-Xptxas -dlcm=ca` – включение L1+L2
`-Xptxas -dlcm=cg` – включение L2

Варианты распределения кэша L1 для ядра:
• 48 Кб общей памяти, 16 Кб кэша
• 16 Кб общей памяти, 48 Кб кэша
Управление:
`cudaFuncSetCacheConfig(kernelfun, flag)`

Общая память

Общая (разделяемая) для потоков в блоке.
Спецификатор `__shared__`.

Аргументы ядра находятся также в общей памяти.

Размер общей памяти на блок указывается при запуске ядра:

```
Kernel<<<grid, block, shmem, stream>>(...);
```

Предпочтителен одновременный доступ соседних потоков к соседним банкам общей памяти (16 банков).

Доступ должен быть синхронизирован.

Общая память

1. Объявление для статических массивов:

```
__shared__ float data[32];
```

2. Синтаксис объявления, если объем разделяемой памяти задается при запуске ядра:

```
extern __shared__ float data[];
```

При этом используется ручное управление смещениями в массиве.

```
extern __shared__ float data[];
...
// первые 10 float
float* first = data+0;
 // вторые 10 float
float* second = data+10*sizeof(float);
```

Константная память

Память находится в блоке констант DRAM (64 Кб). Скорость памяти как у регистровой, если доступ кэшированный. Иначе как к глобальной.

Спецификатор `__constant__`.

Функции доступа:

```
cudaMemcpyToSymbol(const char* dst, void* src,
    size_t count);
cudaMemcpyFromSymbol(void* dst, const char* src,
    size_t count);
```

Запись возможна только из хоста.

Доступ к переменным

В CUDA возможно получить адрес переменной по её названию (константе с текстом).

Обычно используется для константной памяти.

Получение указателя на переменную в памяти хоста для устройства - `cudaHostGetDevicePointer`

Получение указателя на переменную в памяти устройства для хоста - `cudaGetSymbolAddress`

Память текстур

Текстурная память кэшируемая
Доступ быстрее глобальной
Возможна автоматическая нормализация
Возможны немногопоточные чтение+запись

Текстура – область памяти, доступная для выборки по координатам.
Тексель – элемент текстуры

Свойства:
- размерность – текстурные координаты
- тип данных
- размер
- нормализованность

Память текстур

1. режимы адресации:
   - циклический (wrap)
   - граничный (clamp)

2. нормализованность значения на [-1;1] или [0;1]

3. нормализованность координат на [0;1)

4. фильтрация значения
   - точечная
   - билинейная

Память текстур

Два способа доступа:
- обычная память
- cudaArray

При доступе через обычную память недоступны:
- фильтрация
- режимы адресации
- нецелочисленные координаты

Память текстур

Порядок работы:

1. объявление текстурной ссылки:

```
texture<Type, Dim, ReadMode> texRef;
```

2. настройка параметров текстурной ссылки
3. создание массива (обычный или cudaArray)
4. связывание текстурной ссылки и массива

```
cudaBindTexture2D
cudaBindTextureToArray
```

5. доступ в ядре к текстурной ссылке

```
tex1D, tex2D, tex3D
tex2D(texRef, float x, float y)
```

или

```
tex1Dfetch, tex2Dfetch, tex3Dfetch
```

Память текстур

Пример

Объявление и связывание:

```
texture <float, 2, cudaReadModeElementType> texRef;
float* devPtr;
size_t width = 64, height = 64, pitch;
cudaMallocPitch(&devPtr, &pitch,
                width * sizeof(float), height);
cudaChannelFormatDesc channel =
                cudaCreateChannelDesc<float>();
cudaBindTexture2D(0, texRef, devPtr, &channelDesc,
                  width, height, pitch);
```

Доступ:

```
float value = tex2Dfetch(texRef, 10, 15);
```

Поверхности

Surface

CC 2.x (Fermi)

Похожи на текстуры

Управление памятью

- Память хост-машины
- Память графического процессора
- **Копирование данных**

Копирование данных

Копирование матриц:

```
cudaMemcpy
cudaMemcpy2D
cudaMemcpy3D
```

Копирование константной памяти:

```
cudaMemcpyFromSymbol
cudaMemcpyToSymbol
```

Копирование массивов cudaArray:

```
cudaMemcpyArrayToArray
cudaMemcpyFromArray
cudaMemcpyToArray
```

Асинхронное копирование данных

Поток – последовательность операций копирования памяти и запуска ядра

Дескриптор потока - `cudaStream_t`

Управление:

```
cudaStreamCreate
cudaStreamDestroy
```

Копирование с использованием потока:

```
cudaMemcpyAsync(void* dst, void* src, size_t count,
     enum cudaMemcpyKind kind, cudaStream_t stream)
```

Ядру передаётся дескриптор потока:

```
Kernel<<<grid, block, shmem, stream>>(...);
```

Асинхронное выполнение

Асинхронные сочетания:

• одновременное выполнение ядра и копирования
CC 1.1

• одновременное выполнение копирований
CC 2.x

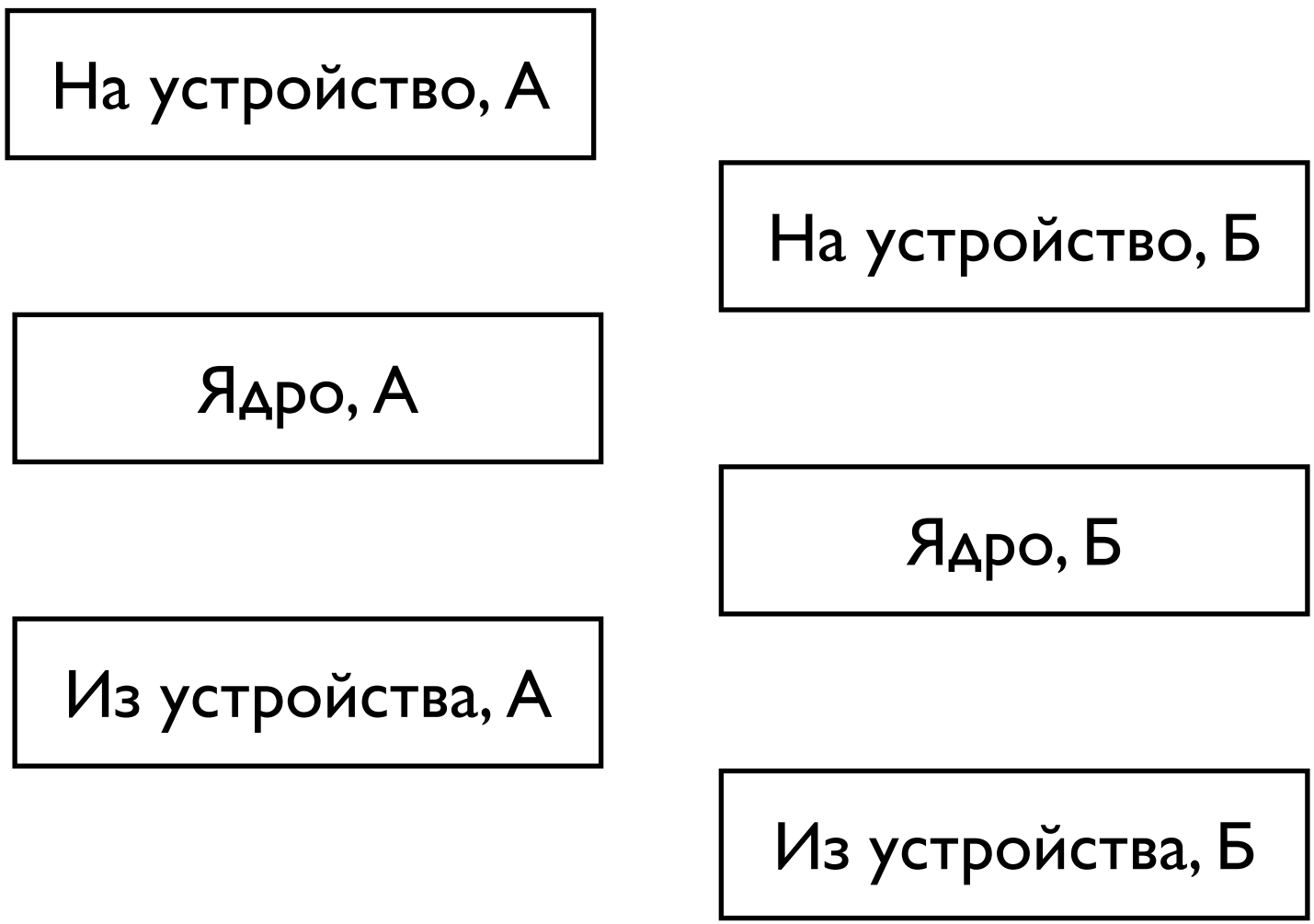
• одновременное выполнение нескольких ядер
CC 2.x

Асинхронное копирование данных

```
    for (int i = 0; i < 2; ++i)
    cudaMemcpyAsync(devIn + i * size, host + i * size,
                    size, cudaMemcpyHostToDevice, stream[i]);
for (int i = 0; i < 2; ++i)
    MyKernel<<<100, 512, 0, stream[i]>>>
          (devOut + i * size, devIn + i * size, size);
for (int i = 0; i < 2; ++i)
    cudaMemcpyAsync(host + i * size, devOut + i * size,
                    size, cudaMemcpyDeviceToHost, stream[i]);
```

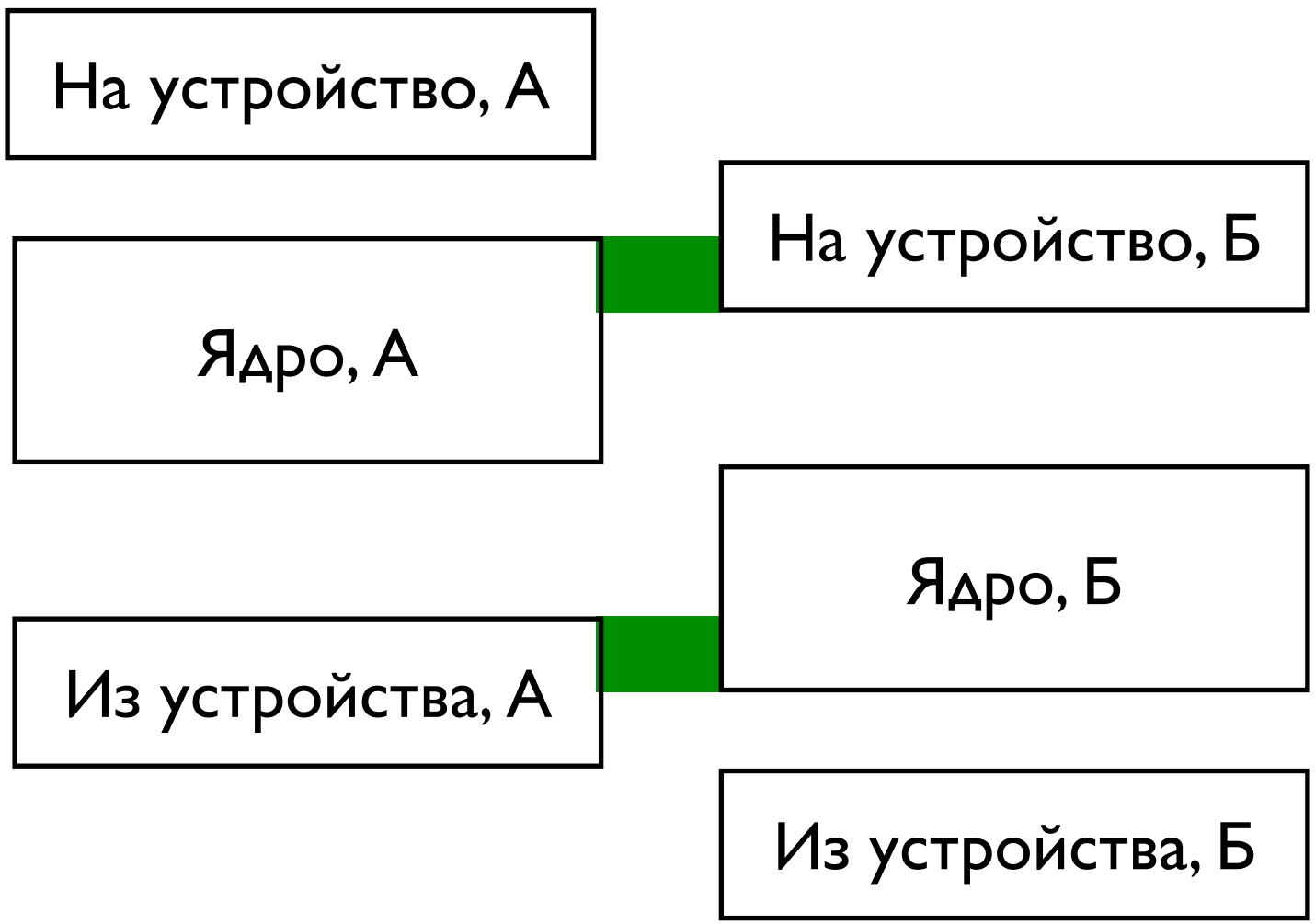
Асинхронное копирование данных

Пессимистический случай



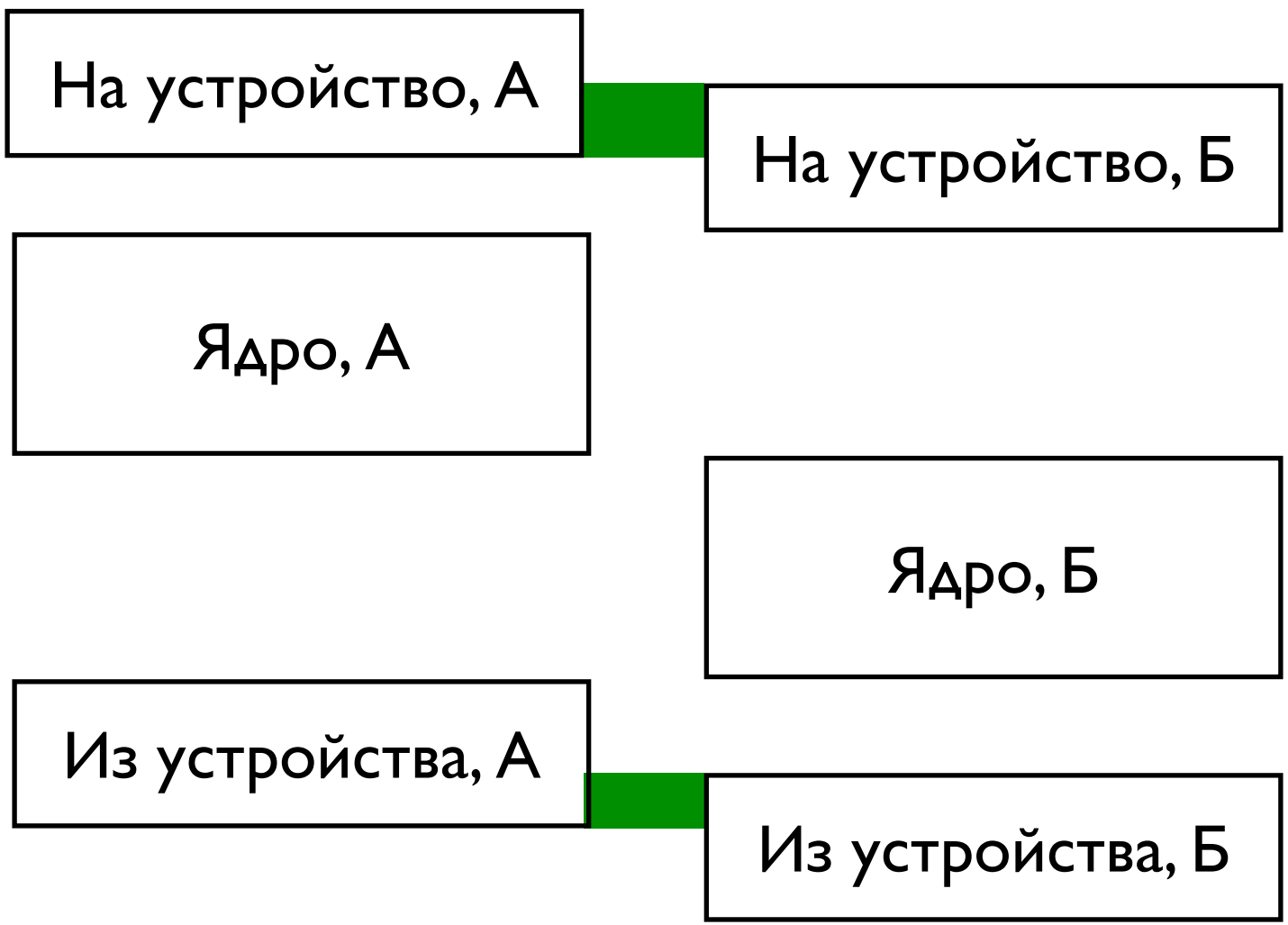
Асинхронное копирование данных

одновременное выполнение ядра и копирования



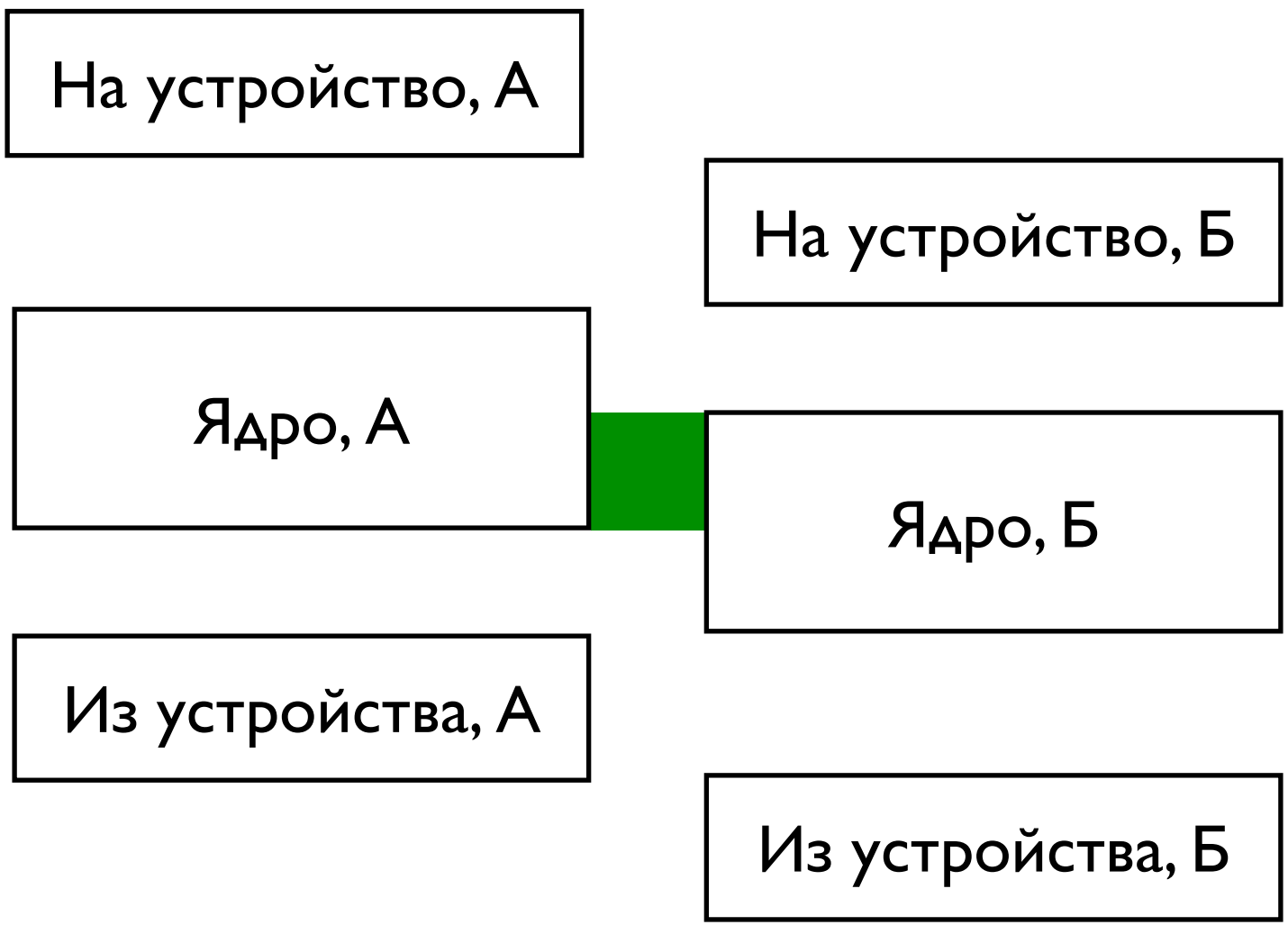
Асинхронное копирование данных

одновременное выполнение копирований



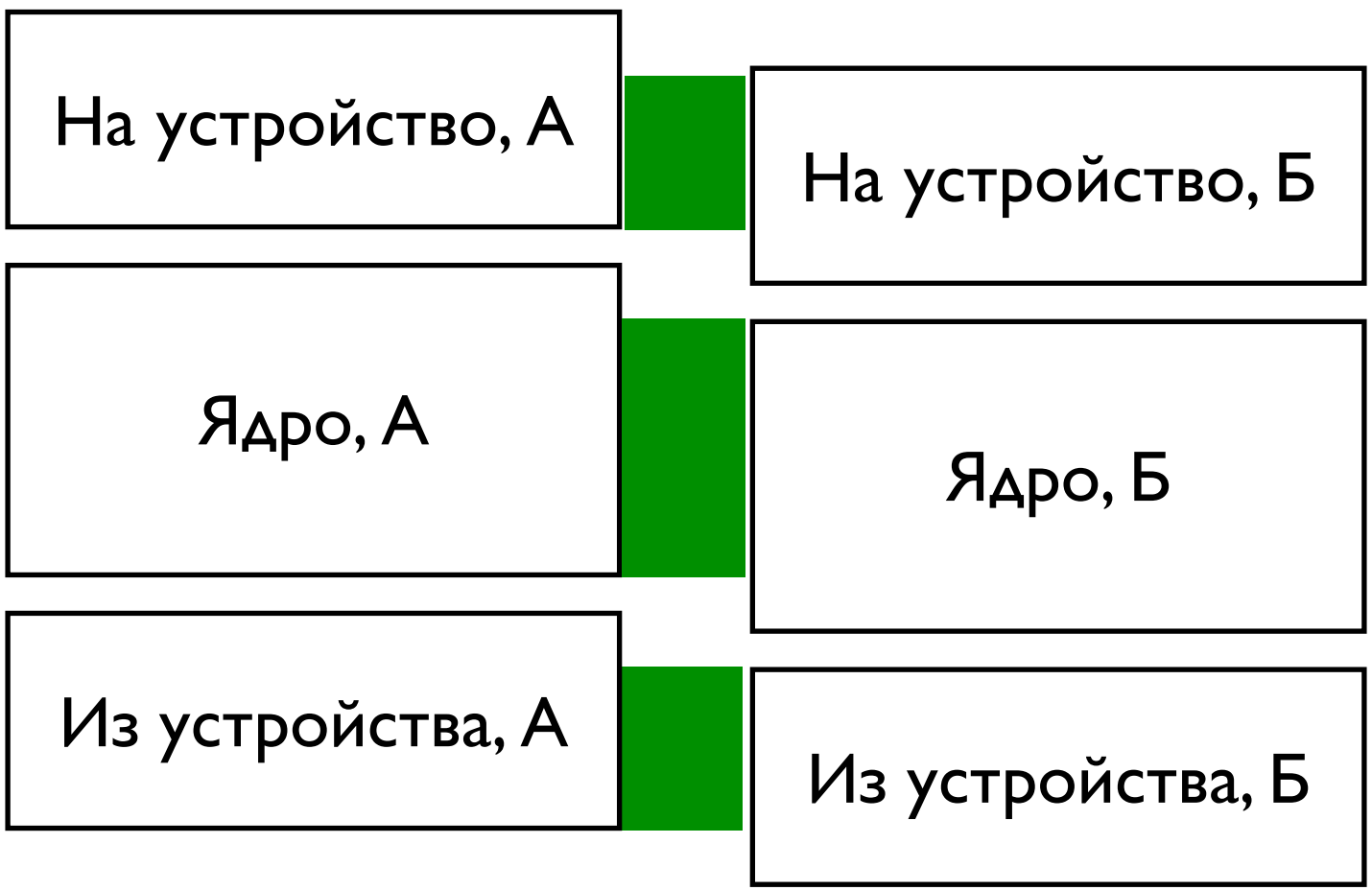
Асинхронное копирование данных

одновременное выполнение нескольких ядер



Асинхронное копирование данных

Оптимистический случай



Отладка

- Внимательное написание программы
- cuda-gdb (unix)

На основе GDB

- Parallel NSight (windows)

Расширение Visual Studio 2008/2010
Среда разработки – кодирование, отладка, профилировка

- cuda-memcheck
- профилировка драйвером (CUDA_PROFILE)
- функция printf (CC 2.x, fermi)
- GPU ocelot – компиляция PTX в x86

Эмуляция устройства

SDK <=2.0
Режим программной эмуляции устройства - опция компилятора -deviceemu.

Потоки эмулируются потоками ОС, блоки выполняются последовательно.

+ возможость отладки
+ диагностика отладочной печатью
+ возможность профилировки
+ обнаружение конфликтов памяти
- невозможно отловить ошибки синхронизации
- реализации IEEE 754 на хосте и GPU отличны

Конфигурации отладки

Отладка текущей видеокарты невозможна.
Решение:
• отладка на локальной машине только со второй видеокарты.
• отладка по сети (ssh, remote desktop, vnc)

Запуск больших программ CUDA может подвешивать графическую оболочку.
Решение:
• консольный сеанс (unix)
• доступ по сети

Driver API

Большинство функций Runtime API реализованы через одноименные функции Driver API.

Основные концепты:
- устройство
- модуль
- контекст
- ядро

Типовые характеристики

CC 1.0 - 2.0

Размеры сети: 65535 × 65535
Размеры блока: 512 × 512 × 64

Число блоков на MP: 8
Число варпов на MP: 24-48
Число потоков на MP: 768/1536
Число регистров на MP: 8000-32000
Объем разделяемой памяти на MP: 16/48 Кб
Объем локальной памяти на поток: 16/512 Кб
Объем памяти констант: 64 Кб

Библиотеки

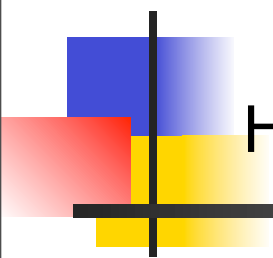
- CUBLAS
- CUFT
- CUSPARSE
- CURAND
- THRUST

CUDA 4.x

- CC 2.0+ (Fermi, Kepler)
- LLVM
- printf в ядре
- общее адресное пространство
- улучшенная поддержка multi-gpu

Ресурсы этого курса

http://omniverse.ru/bmstu/cuda-rk6



Наш FTP: ftp://cad.bmstu.ru/manichev/ZHUK-CUDA-6-kurs/

NVidia: http://nvidia.com/cuda

CUDA C Programming Guide

CUDA C Best Practices Guide

CUDA Toolkit Reference Manual

Programming Massively Parallel Processors: http://amzn.to/99ulEZ

CUDA By Example: http://amzn.to/9lbtEw

Курс МГУ: http://groups.google.com/group/cudacsmsusu