

Лекция 1

(введение)

Содержание лекции

- модели параллельных вычислений;
- аппаратные особенности графических процессоров;
- архитектура CUDA – основные свойства и принципы;
- программная модель: хост, устройства, ядра, иерархия нитей (*threads*);
- установка CUDA Toolkit.

Модель параллельных вычислений на GPU (обзор моделей параллелизации)

Модель	Программные средства	Архитектура ВС
Общая память	POSIX (pthread), WinAPI(CreateThread), OpenMP...	MIMD, разделяемая память
Обмен сообщениями	MPI (Message Passing Interface): OpenMPI, MPICH, LAM (Local Area Multivomputer); PVM (Parallel Virtual Machine)...	MIMD, распределенная и разделяемая память
Параллелизм данных	Языки .NET, Python...	MIMD/SIMD

Архитектура фон Неймана

Центральный процессор

Шина

ОЗУ

Арифметико-
логическое
устройство

Устройство
управления

0



A0

1



0011 0101 1110 0111

...



F



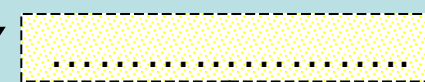
00



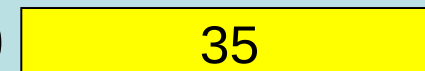
01



XY



A0



A1




XY



FF



 Регистры общего назначения – сумматор, регистр данных, адресный регистр и т.д.

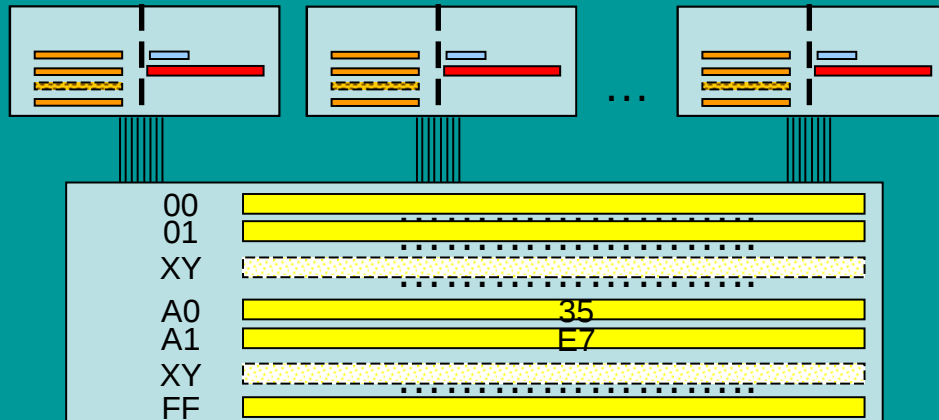
 Счетчик команд

 Ячейки памяти

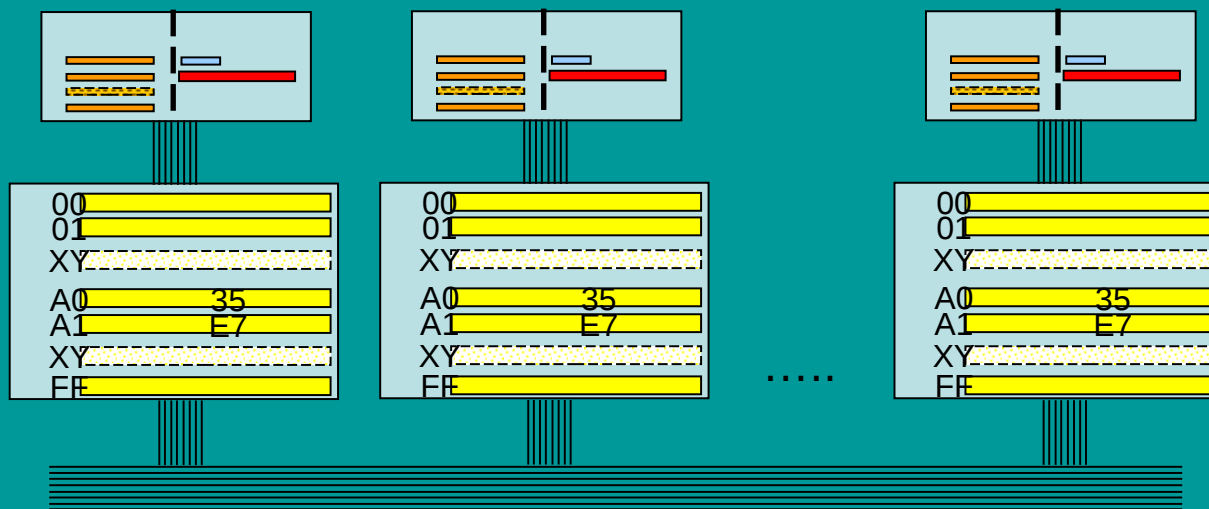
 Регистр команд

Основные архитектуры производительных ВС

Разделяемая память

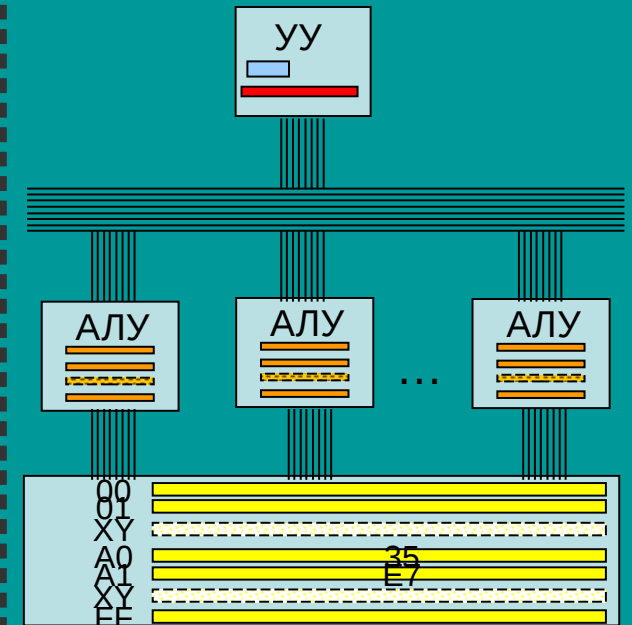


Распределенная память



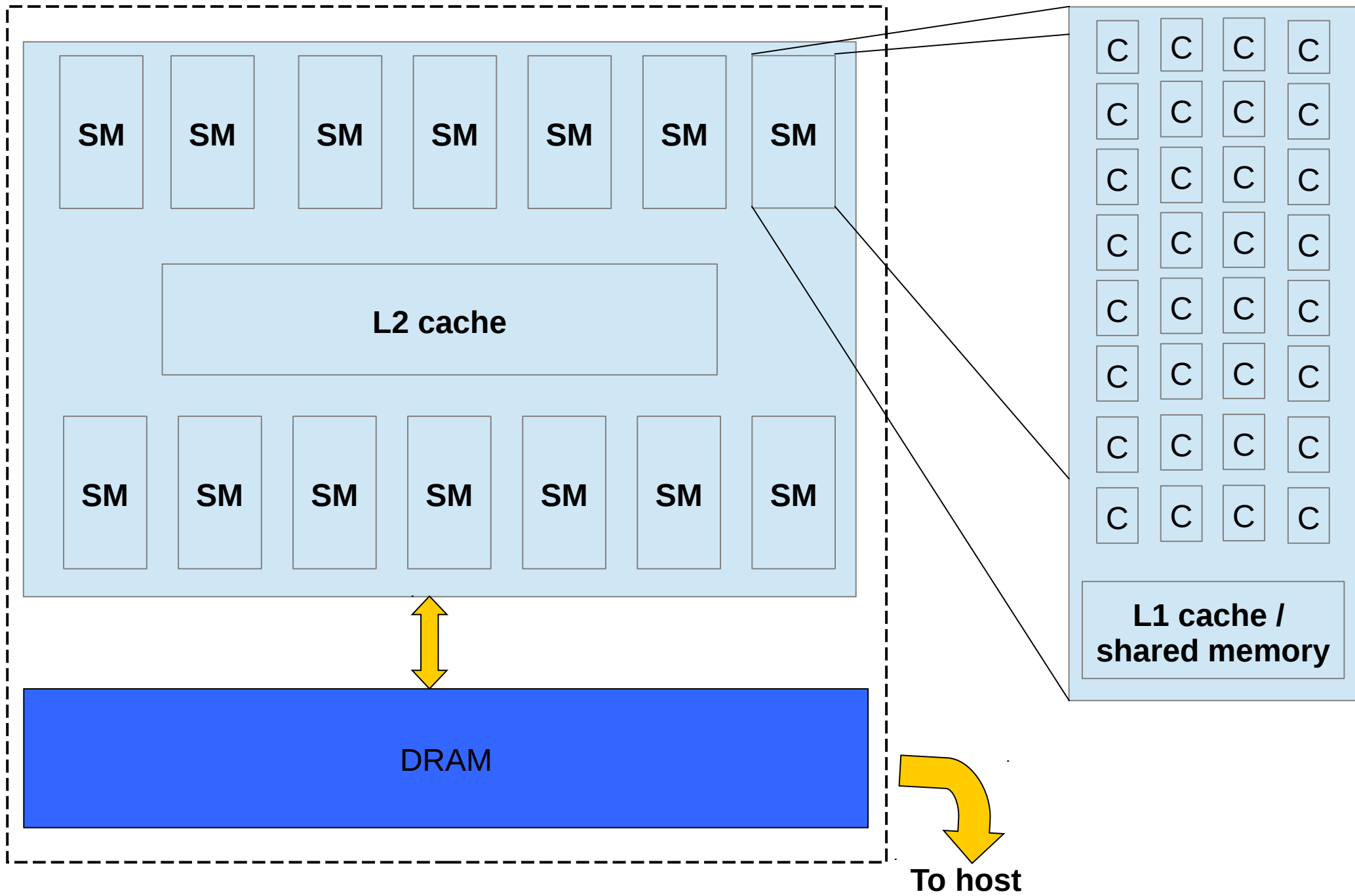
MIMD

Разделяемая память



SIMD

GPU (Fermi architecture)



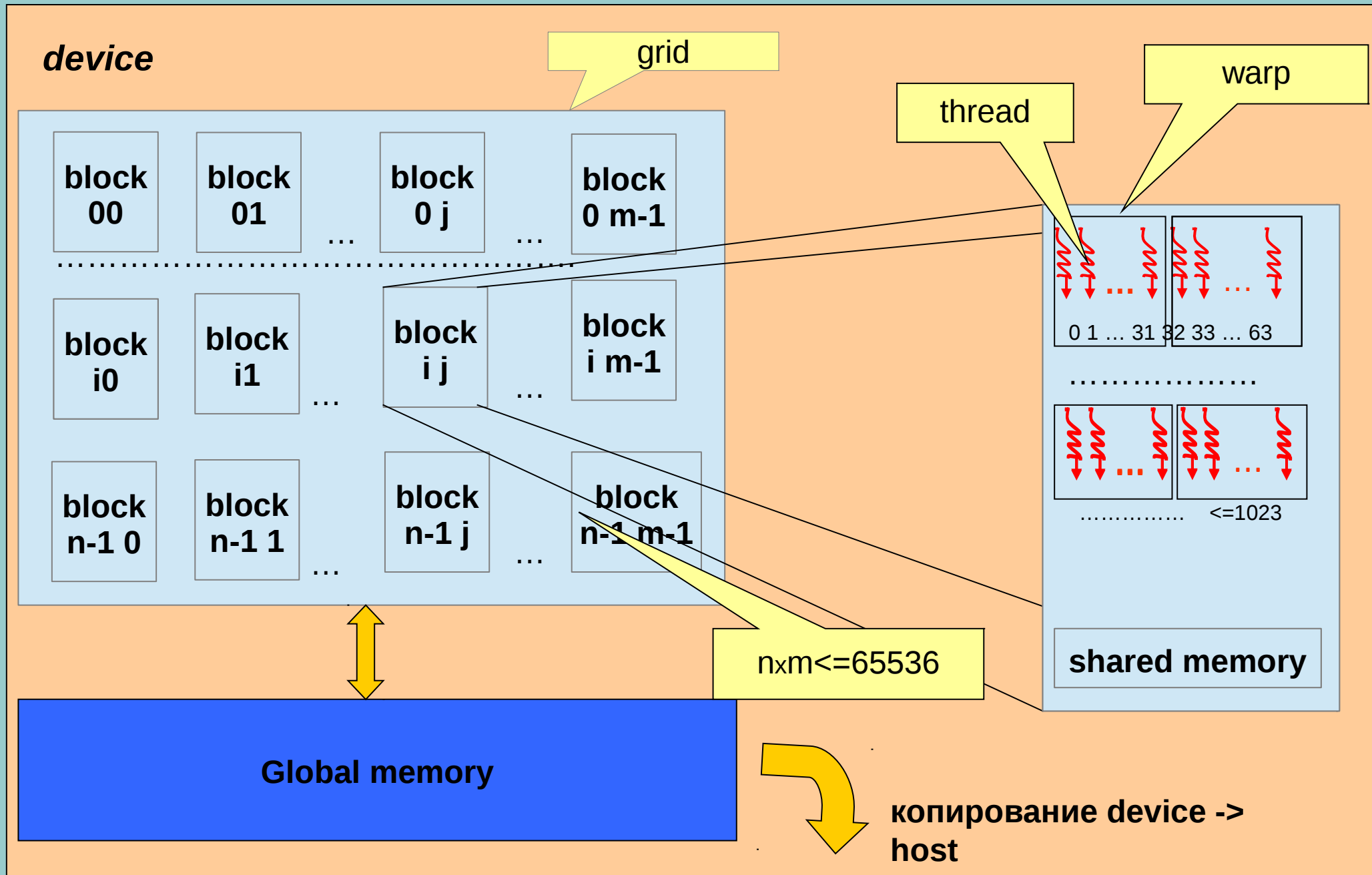
Логическое представление GPU

Активное использование графических процессоров (GPU) для прикладных расчетов научно-технического назначения во многом связано с предоставлением компанией NVIDIA технологии **CUDA** (*Cuda Unified Device Architecture*). Технология CUDA предоставляет понятную для прикладного программиста абстракцию графического процессора (GPU) и простой интерфейс прикладного программирования (*API – Application Programming Interface*).

По терминологии *CUDA* вычислительный узел с *CPU* и *main memory* называется **host**, *GPU* называется **device**. Программа, выполняемая на *host*'е содержит код – ядро (**kernel**), который загружается на *device* в виде многочисленных копий. Все копии загруженного кода – нити (**threads**), объединяются в блоки (**blocks**) по 512-1024 нити в каждом. Все блоки объединяются в сеть (**grid**) с максимальным количеством блоков 65536. Все нити имеют совместный доступ на запись/чтение к памяти большого объема - *global memory*, на чтение к кэшируемым **constant memory** и **texture memory**. Нити одного блока имеют доступ к быстрой памяти небольшого объема – **shared memory**.

CUDA (Compute Unified Device Architecture)

- cuda предоставляет абстракцию GPU для программистов



Программный интерфейс CUDA

CUDA C - расширение языка C , и набор функций и структур *CUDA API* предоставляют простой инструмент для программирования на GPU.

Некоторые конструкции программного интерфейса CUDA.

Функция-ядро (*kernel*)

Код, выполняемый на устройстве (ядро), определяется в виде функции типа *void* со спецификатором `__global__`:

```
__global__ void gFunc(<params>){...}
```

Конфигурация нитей

При вызове ядра программист определяет количество нитей в блоке и количество блоков в `grid`. При этом допустима линейная, двумерная или трехмерная индексация нитей:

```
gFunc<<<dim3(bl_xdim, bl_ydim, bl_zdim),  
           dim3(th_xdim, th_ydim, th_zdim)>>>(<params>);
```


Программный интерфейс CUDA (самый простой пример)

```
#include <cuda.h>
#include <stdio.h>

__global__ void gTest(float* a){
    a[threadIdx.x+blockDim.x*blockIdx.x]=(float)(threadIdx.x+blockDim.x*blockIdx.x);
}

int main(){
    float *da, *ha;
    int num_of_blocks=10, threads_per_block=32;
    int N=num_of_blocks*threads_per_block;

    ha=(float*)calloc(N, sizeof(float));
    cudaMalloc((void**)&da, N*sizeof(float));

    gTest<<<dim3(num_of_blocks), dim3(threads_per_block)>>>(da);
    //cudaThreadSynchronize();
    CudaDeviceSynchronize();
    cudaMemcpy(ha,da,N*sizeof(float), cudaMemcpyDeviceToHost);

    for(int i=0;i<N;i++)
        printf("%g\n", ha[i]);

    free(ha);
    cudaFree(da);
    return 0;
}
```

CUDA C - расширение языка C , и набор функций и структур CUDA API предоставляют простой инструмент для программирования на GPU.

Комментарии: использование глобальной памяти

```
cudaError_t  cudaMalloc (void ** devPtr, size_t size)
```

```
cudaError_t  cudaFree (void * devPtr)
```

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

```
enum cudaError
{
    cudaSuccess = 0,
    cudaErrorMissingConfiguration,
    cudaErrorMemoryAllocation,
    cudaErrorInitializationError,
    cudaErrorLaunchFailure,
    .....
};

typedef enum cudaError cudaError_t;
```

```
enum cudaMemcpyKind
{
    cudaMemcpyHostToHost = 0,
    cudaMemcpyHostToDevice,
    cudaMemcpyDeviceToHost,
    cudaMemcpyDeviceToDevice
};
```

Глобальная память **выделяется только на хосте**, к глобальной памяти возможен **доступ только на устройстве**.

Документация CUDA: <http://docs.nvidia.com/cuda/index.html>

Комментарии: встроенные типы и переменные

- uint3 **threadIdx** – индекс нити в блоке
- dim3 **blockDim** – размер блока
- uint3 **blockIdx** – индекс блока в гриде
- dim3 **gridDim** - размер грида
- int **warpSize** - количество нитей в варпе (warp)

Комментарии: синхронизация всех нитей

Запуск ядра на устройстве (вызов функции с модификатором `__global__`) происходит в асинхронном режиме.

Для синхронизации нитей служат следующие вызовы:

```
//cudaThreadSynchronize(); //устаревшая функция (depricated)  
cudaDeviceSynchronize();
```

Упражнение

```
.....  
for(int i=0; i<N; i++){  
    a[i]=b[i]+c[i];  
    a[i]*=a[i];  
}  
.....
```

```
.....  
i=threadIdx.x+blockDim.x*blockIdx.x  
a[i]=b[i]+c[i];  
a[i]*=a[i]; (?)  
.....
```

Распараллельте цикл:

- инициализируйте массивы a, b, c на хосте;
- скопируйте их на устройство;
- разберитесь с подсказкой справа;
- определите, примерно, когда происходит насыщение ускорения;
- выберите оптимальную конфигурацию нитей.

Установка CUDA Toolkit (docs.nvidia.com/cuda/)

LINUX: docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html

rpm-накет

Click on the green buttons that describe your target platform. Only supported platforms will be shown.

Operating System	Windows	Linux	Mac OSX			
Architecture	x86_64	ppc64le				
Distribution	Fedora	OpenSUSE	RHEL	CentOS	SLES	SteamOS
	Ubuntu					
Version	13.2					
Installer Type	runfile (local)	rpm (local)	rpm (network)			

Download Target Installer for Linux OpenSUSE 13.2 x86_64

cuda-repo-opensuse132-7-5-local-7-5-18.x86_64.rpm (md5sum: e6708f4b38cc9ed546d3a04f7c731698)

Download (1.1 GB)

Installation Instructions:

1. `sudo rpm -i cuda-repo-opensuse132-7-5-local-7-5-18.x86_64.rpm`
2. `sudo zypper refresh`
3. `sudo zypper install cuda`

For further information, see the [Installation Guide for Linux](#) and the [CUDA Quick Start Guide](#).

После установки драйвера: `sudo nvidia-xconfig`

run-файл

Click on the green buttons that describe your target platform. Only supported platforms will be shown.

Operating System	Windows	Linux	Mac OSX			
Architecture	x86_64	ppc64le				
Distribution	Fedora	OpenSUSE	RHEL	CentOS	SLES	SteamOS
	Ubuntu					
Version	13.2					
Installer Type	runfile (local)	rpm (local)	rpm (network)			

Download Target Installer for Linux OpenSUSE 13.2 x86_64

cuda_7.5.18_linux.run (md5sum: 4b3bcecf0dfc35928a0898793cf3e4c6)

Download (1.1 GB)

Installation Instructions:

1. Run `sudo sh cuda_7.5.18_linux.run`
2. Follow the command-line prompts

The GPU Deployment Kit is available as a separate download [here](#).

For further information, see the [Installation Guide for Linux](#) and the [CUDA Quick Start Guide](#).

1. Выгрузить X-server (`sudo init 3`)
2. Блокировать загрузку драйвера **nouveau**

Установка CUDA Toolkit (docs.nvidia.com/cuda/)

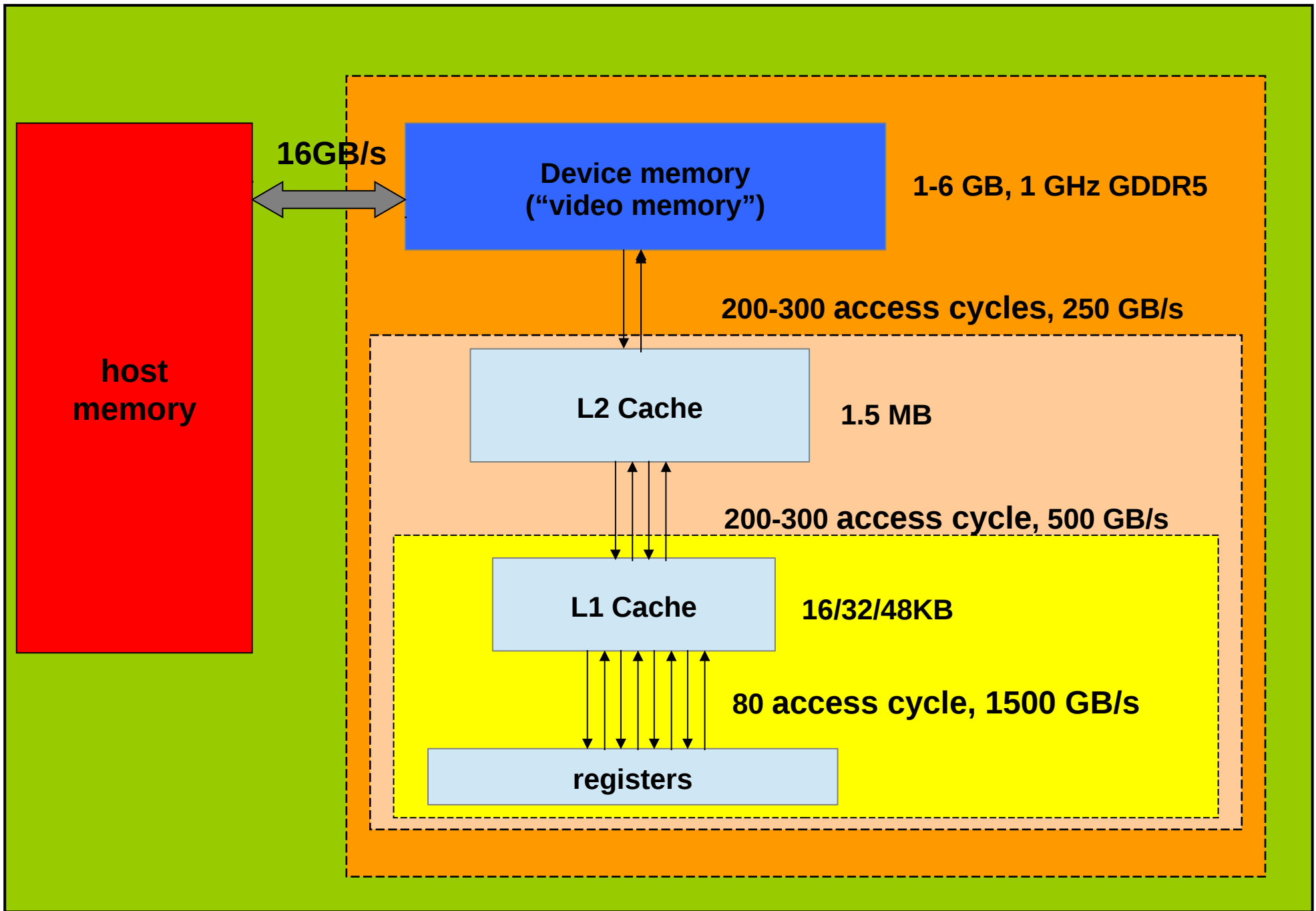
Проверка установки:

- **Перегрузка**
- **Запись в ~/.bashrc:**
\$ export PATH=/usr/local/cuda-7.5/bin:\$PATH
\$ export LD_LIBRARY_PATH=/usr/local/cuda-7.5/lib64:\$LD_LIBRARY_PATH
- **Копирование файлов в папку пользователя:** \$ cuda-install-samples-7.5.sh <dir>
- **Компиляция, компоновка и запуск примера (*make*, затем *./<file_executable>*)**

Спасибо за внимание

Дополнение.

memory hierarchy (Fermi, Kepler)



Compute capabilities (вычислительные возможности)

Compute capabilities (вычислительные возможности) представляют спецификацию GPU. Особенности “вычислительных возможностей” включают допустимость операций с плавающей точкой, допустимость атомарных операций, возможность синхронизации нитей, кэшируемость глобальной памяти и т.д. Описание различных версий Compute capabilities можно найти, например, в CUDA C Programming Guid – руководстве по CUDA C компании NVIDIA.

Архитектура GPU	Compute capabilities	Версия CUDA
Tesla	1.*	CUDA 2.*-3.*
Fermi	2.*	CUDA 4.*-5.*
Kepler	3.*	CUDA 5.*
Maxwell	5.*	CUDA 6.*-7.*
Pascal	6.*	CUDA 8
Volta	7.*	

Характеристики GPU на серверах 'home' и 'dew'

Detected 1 CUDA Capable device(s)

Device 0: "GeForce GTX 560 Ti"

CUDA Driver Version / Runtime Version 5.0 / 5.0

CUDA Capability Major/Minor version number: 2.1

Total amount of global memory: 2048 MBytes (2147024896 bytes)

(8) Multiprocessors x (48) CUDA Cores/MP: 384 CUDA Cores

GPU Clock rate: 1645 MHz (1.64 GHz)

Memory Clock rate: 2004 Mhz

Memory Bus Width: 256-bit

L2 Cache Size: 524288 bytes

Max Texture Dimension Size (x,y,z) 1D=(65536), 2D=(65536,65535), 3D=(2048,2048,2048)

Max Layered Texture Size (dim) x layers 1D=(16384) x 2048, 2D=(16384,16384) x 2048

Total amount of constant memory: 65536 bytes

Total amount of shared memory per block: 49152 bytes

Total number of registers available per block: 32768

Warp size: 32

Maximum number of threads per multiprocessor: 1536

Maximum number of threads per block: 1024

Maximum sizes of each dimension of a block: 1024 x 1024 x 64

Maximum sizes of each dimension of a grid: 65535 x 65535 x 65535

Maximum memory pitch: 2147483647 bytes

Texture alignment: 512 bytes

Concurrent copy and kernel execution: Yes with 1 copy engine(s)

Run time limit on kernels: Yes

Integrated GPU sharing Host Memory: No

Support host page-locked memory mapping: Yes

Alignment requirement for Surfaces: Yes

Device has ECC support: Disabled

Device supports Unified Addressing (UVA): Yes

Device PCI Bus ID / PCI location ID: 1 / 0

Compute Mode:

< Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

Detected 1 CUDA Capable device(s)

Device 0: "GeForce GTX 650"

CUDA Driver Version / Runtime Version 5.5 / 5.5

CUDA Capability Major/Minor version number: 3.0

Total amount of global memory: 2048 MBytes (2147155968 bytes)

(2) Multiprocessors x (192) CUDA Cores/MP: 384 CUDA Cores

GPU Clock rate: 1110 MHz (1.11 GHz)

Memory Clock rate: 2500 Mhz

Memory Bus Width: 128-bit

L2 Cache Size: 262144 bytes

Max Texture Dimension Size (x,y,z) 1D=(65536), 2D=(65536,65536), 3D=(4096,4096,4096)

Max Layered Texture Size (dim) x layers 1D=(16384) x 2048, 2D=(16384,16384) x 2048

Total amount of constant memory: 65536 bytes

Total amount of shared memory per block: 49152 bytes

Total number of registers available per block: 65536

Warp size: 32

Maximum number of threads per multiprocessor: 2048

Maximum number of threads per block: 1024

Maximum sizes of each dimension of a block: 1024 x 1024 x 64

Maximum sizes of each dimension of a grid: 65535 x 65535 x 65535

Maximum memory pitch: 2147483647 bytes

Texture alignment: 512 bytes

Concurrent copy and kernel execution: Yes with 1 copy engine(s)

Run time limit on kernels: Yes

Integrated GPU sharing Host Memory: No

Support host page-locked memory mapping: Yes

Alignment requirement for Surfaces: Yes

Device has ECC support: Disabled

Device supports Unified Addressing (UVA): Yes

Device PCI Bus ID / PCI location ID: 1 / 0

Compute Mode:

< Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

Характеристики GPU на гибридном кластере 'nusc.ru'

Found 3 CUDA Capable device(s)

Device 0: "Tesla M2090"

CUDA Driver Version / Runtime Version 4.2 / 4.1

CUDA Capability Major/Minor version number: 2.0

Total amount of global memory: 5375 MBytes (5636554752 bytes)

(16) Multiprocessors x (32) CUDA Cores/MP: 512 CUDA Cores

GPU Clock Speed: 1.30 GHz

Memory Clock rate: 1848.00 Mhz

Memory Bus Width: 384-bit

L2 Cache Size: 786432 bytes

Max Texture Dimension Size (x,y,z) 1D=(65536),

2D=(65536,65535), 3D=(2048,2048,2048)

Max Layered Texture Size (dim) x layers 1D=(16384) x 2048,

2D=(16384,16384) x 2048

Total amount of constant memory: 65536 bytes

Total amount of shared memory per block: 49152 bytes

Total number of registers available per block: 32768

Warp size: 32

Maximum number of threads per block: 1024

Maximum sizes of each dimension of a block: 1024 x 1024 x 64

Maximum sizes of each dimension of a grid: 65535 x 65535 x 65535

Maximum memory pitch: 2147483647 bytes

Texture alignment: 512 bytes

Concurrent copy and execution: Yes with 2 copy engine(s)

Run time limit on kernels: No

Integrated GPU sharing Host Memory: No

Support host page-locked memory mapping: Yes

Concurrent kernel execution: Yes

Alignment requirement for Surfaces: Yes

Device has ECC support enabled: Yes

Device is using TCC driver mode: No

Device supports Unified Addressing (UVA): Yes

Device PCI Bus ID / PCI location ID: 6 / 0

Compute Mode:

< Exclusive Process (many threads in one process is able to use
::cudaSetDevice() with this device) >

Device 1: "Tesla M2090"

CUDA Driver Version / Runtime Version 4.2 / 4.1

CUDA Capability Major/Minor version number: 2.0

Total amount of global memory: 5375

MBytes (5636554752 bytes)

(16) Multiprocessors x (32) CUDA Cores/MP: 512

CUDA Cores

GPU Clock Speed: 1.30 GHz

Memory Clock rate: 1848.00 Mhz

Memory Bus Width: 384-bit

L2 Cache Size: 786432 bytes

Max Texture Dimension Size (x,y,z)

1D=(65536), 2D=(65536,65535), 3D=(2048,2048,2048)

Max Layered Texture Size (dim) x layers

1D=(16384) x 2048, 2D=(16384,16384) x 2048

Total amount of constant memory: 65536

bytes

Total amount of shared memory per block: 49152

bytes

Total number of registers available per block: 32768

Warp size: 32

Maximum number of threads per block: 1024

Maximum sizes of each dimension of a block: 1024 x

1024 x 64

Maximum sizes of each dimension of a grid: 65535 x

65535 x 65535

.....

Device 2: "Tesla M2090"

CUDA Driver Version / Runtime Version 4.2 / 4.1

CUDA Capability Major/Minor version number: 2.0

Total amount of global memory: 5375

MBytes (5636554752 bytes)

(16) Multiprocessors x (32) CUDA Cores/MP: 512

CUDA Cores

GPU Clock Speed: 1.30 GHz

Memory Clock rate: 1848.00 Mhz

Memory Bus Width: 384-bit

L2 Cache Size: 786432 bytes

Max Texture Dimension Size (x,y,z)

1D=(65536), 2D=(65536,65535), 3D=(2048,2048,2048)

Max Layered Texture Size (dim) x layers

1D=(16384) x 2048, 2D=(16384,16384) x 2048

Total amount of constant memory: 65536

bytes

Total amount of shared memory per block: 49152

bytes

Total number of registers available per block: 32768

Warp size: 32

Maximum number of threads per block: 1024

Maximum sizes of each dimension of a block: 1024 x

1024 x 64

Maximum sizes of each dimension of a grid: 65535 x

65535 x 65535

.....

Гибридный кластер НГУ (<http://nusc.ru>)

Кластер состоит из 12 узлов *HP SL390s G7*, каждый из которых содержит два 6-ядерных CPU *Xeon X5670* и три графические карты *NVIDIA Tesla M2090*.

Каждый GPU имеет **512 ядер** (cores) с частотой 1.3GHz и память размером **6GB** с пропускной способностью (bandwidth) 177 GB/s.

