

# 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*);
- программный интерфейс CUDA;
- 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



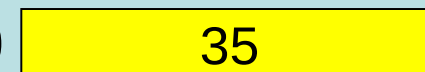
01



XY



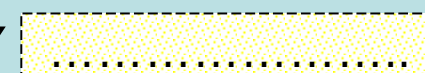
A0



A1




XY



FF



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

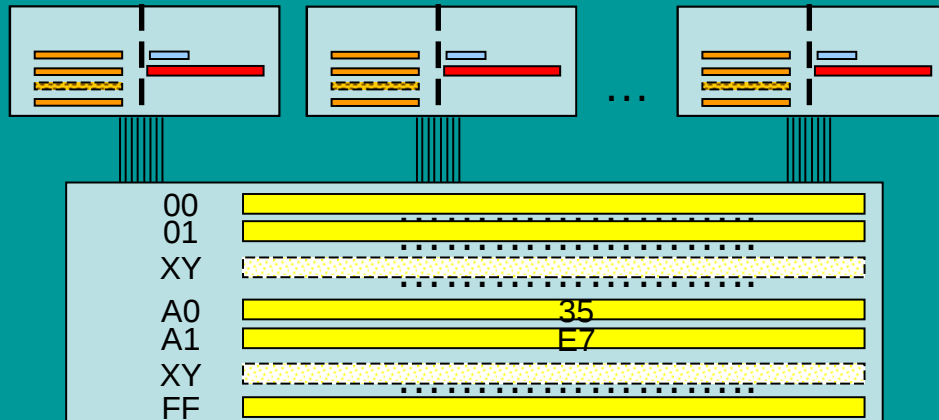
 Счетчик команд

 Ячейки памяти

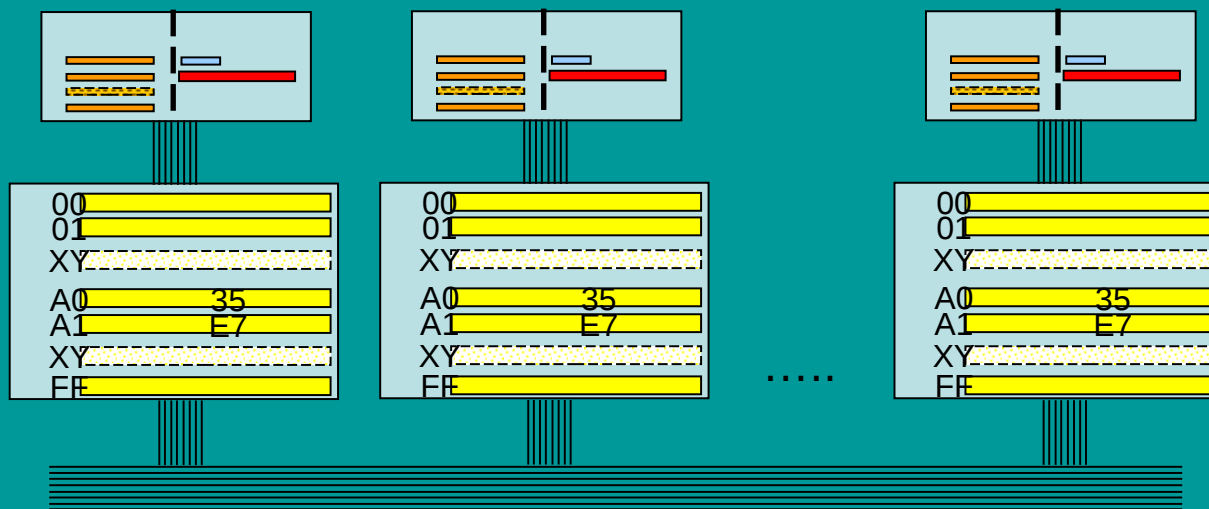
 Регистр команд

# Основные архитектуры производительных ВС

*Разделяемая память*

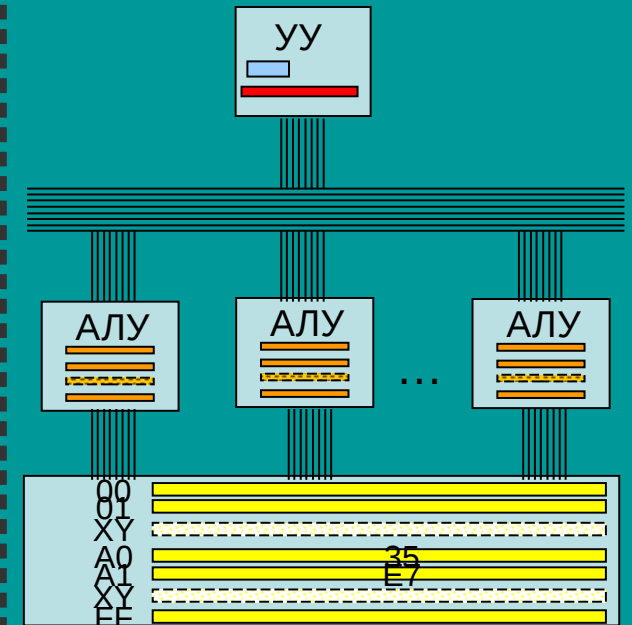


*Распределенная память*



MIMD

*Разделяемая память*



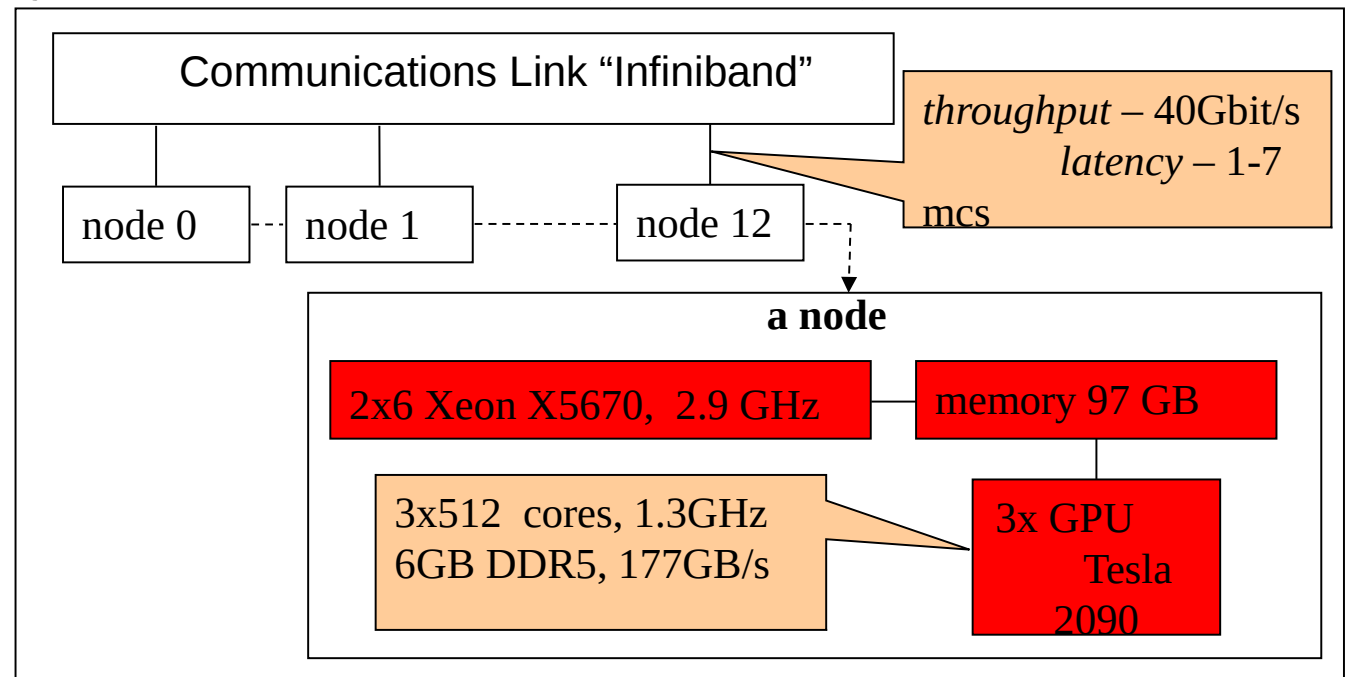
SIMD

# Гибридный кластер НГУ

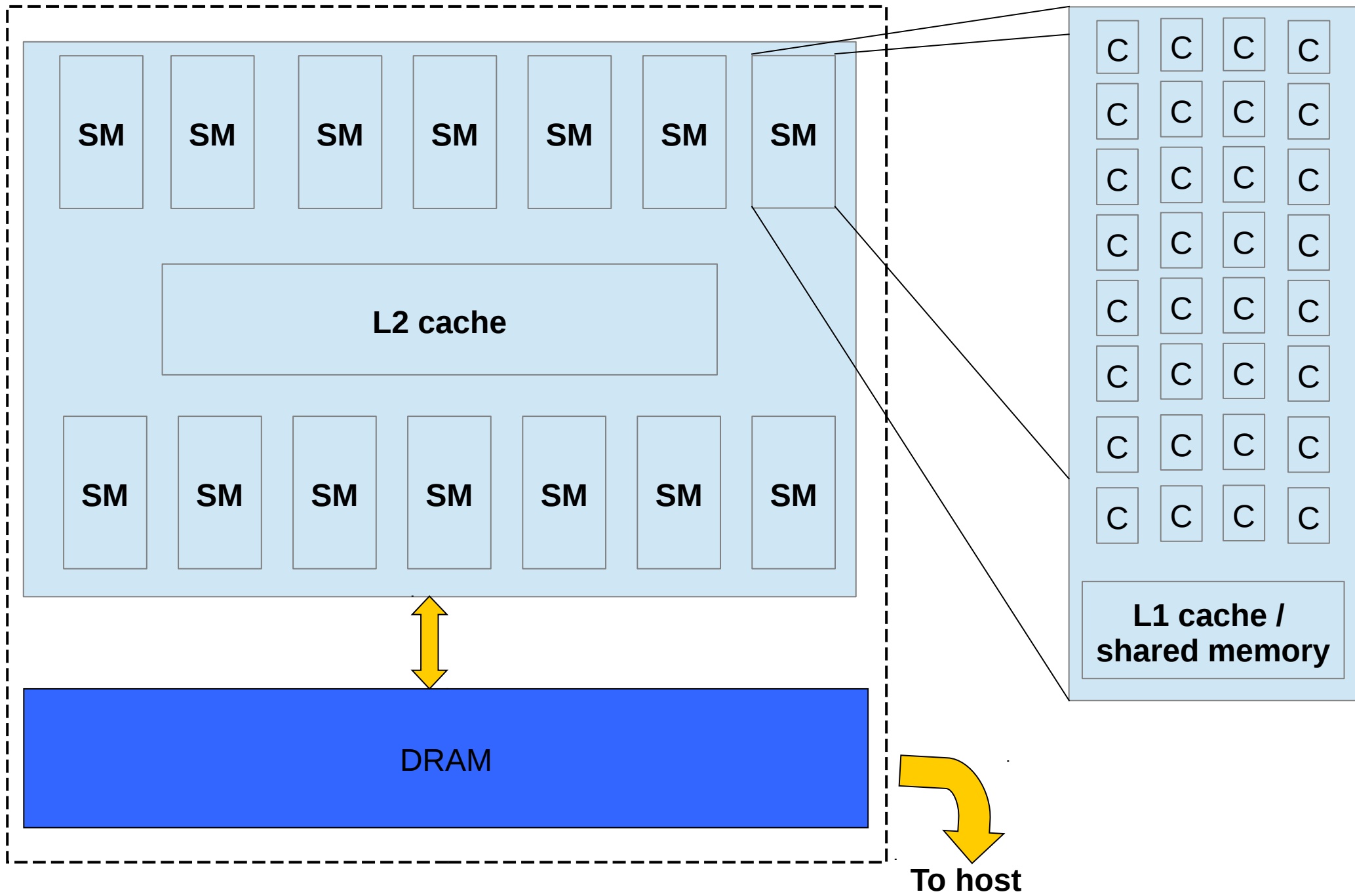
(<http://nusc.ru>)

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.



# 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*).

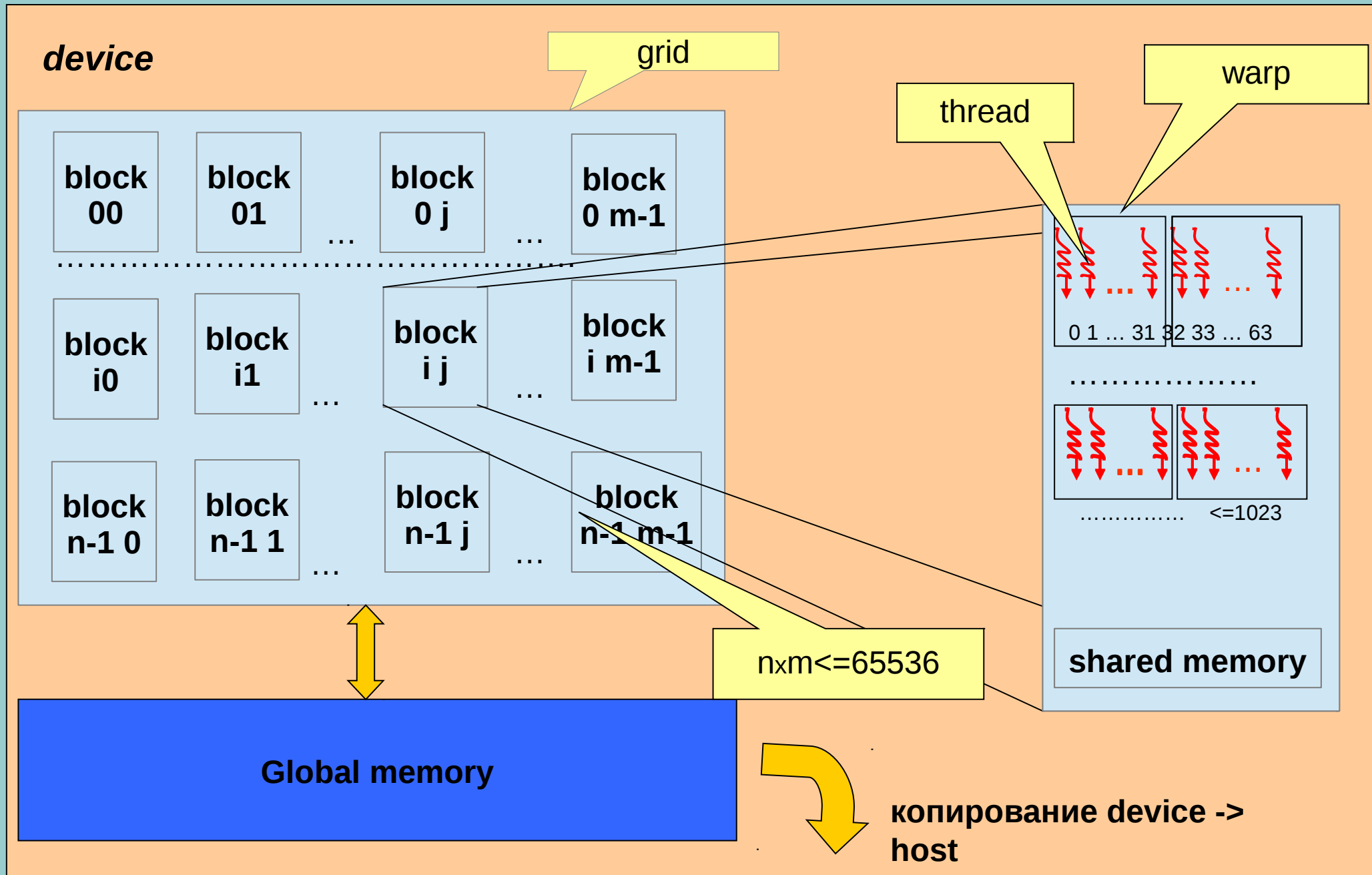
## Логическое представление GPU

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

Расширение языка C *CUDA C* — спецификаторы функций и переменных, специальные директивы, встроенные переменные и новые типы данных, а так же набор функций и структур данных *CUDA API*, предоставляют простой инструмент для программирования на GPU.

## Функция-ядро (*kernel*)

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

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

# Программный интерфейс CUDA

## 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);
}
```

# Программный интерфейс CUDA (самый простой пример)

```
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));
```

# Программный интерфейс CUDA (самый простой пример)

```
gTest<<<dim3(num_of_blocks),  
        dim3(threads_per_block)>>>(da);  
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;  
}
```

## *Комментарии: использование глобальной памяти*

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

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

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

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

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

## *Комментарии: использование глобальной памяти*

### **enum cudaError**

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

### **enum cudaMemcpyKind**

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

```
typedef enum cudaError cudaError_t;
```



## 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;
- определите, примерно, когда происходит насыщение ускорения;
- выберите оптимальную конфигурацию нитей.

## Порядок работы

```
> ssh malkov@oak.cpct.sibsutis.ru
malkov@oak.cpct.sibsutis.ru's password:
Last login: Mon Feb  5 18:22:23 2018 from 5.128.191.230
[malkov@oak ~]$
[malkov@oak ~]$ ssh cngpu2
malkov@cngpu2's password:
Last login: Mon Feb  5 18:17:28 2018 from 192.168.1.254
[malkov@cngpu2 ~]$
[malkov@cngpu2 ~]$ cd Malkov/gr-511
[malkov@cngpu2 gr-511]$ mkdir brig1
[malkov@cngpu2 gr-511]$ cd brig1
[malkov@cngpu2 brig1]$
```

## Порядок работы

```
[malkov@cngpu2 brig1]$ vi lab1.cu
```

```
[malkov@cngpu2 brig1]$ ls -l
```

```
total 564
```

```
-rw-r--r--. 1 malkov masters 1091 Feb  6 17:27 lab1.cu
```

```
[malkov@cngpu2 brig1]$ nvcc lab1.cu -o lab1
```

```
nvcc warning : The 'compute_20', 'sm_20', and 'sm_21'  
architectures are deprecated, and may be removed in a future  
release (Use -Wno-deprecated-gpu-targets to suppress warning).
```

```
[malkov@cngpu2 brig1]$ ls -l
```

```
total 564
```

```
-rwxr-xr-x. 1 malkov masters 570720 Feb  6 17:48 lab1
```

```
-rw-r--r--. 1 malkov masters 1091 Feb  6 17:27 lab1.cu
```

## Порядок работы

```
[malkov@cngpu2 brig1]$ nvprof ./lab1
```

```
==10980== Profiling application: ./lab1
```

```
==10980== Profiling result:
```

Time(%)	Time	Calls	Avg	Min	Max	Name
52.01%	7.9990us	1	7.9990us	7.9990us	7.9990us	[CUDA memcpy DtoH]
47.99%	7.3810us	1	7.3810us	7.3810us	7.3810us	gTest(float*)

```
==10980== API calls:
```

Time(%)	Time	Calls	Avg	Min	Max	Name
99.06%	36.870ms	1	36.870ms	36.870ms	36.870ms	cudaMalloc

# Установка CUDA Toolkit ([docs.nvidia.com/cuda/](https://docs.nvidia.com/cuda/))

LINUX: [docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html](https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html)

## *rpm-накем*

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/](https://docs.nvidia.com/cuda/))

### Проверка установки:

- **Перезагрузка**
- **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