Семинар 13
Технология CUDA
Введение

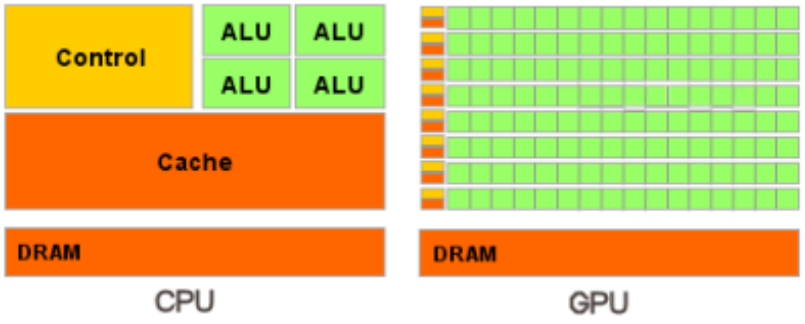
Михаил Курносов

E-mail: mkurnosov@gmail.com
WWW: www.mkurnosov.net

Цикл семинаров «Основы параллельного программирования»
Институт физики полупроводников им. А. В. Ржанова СО РАН
Новосибирск, 2015

GPU – Graphics Processing Unit

- **Graphics Processing Unit (GPU)** – графический процессор, специализированный многопроцессорный ускоритель с общей памятью
- Большая часть площади чипа занята элементарными ALU/FPU/Load/Store модулями
- Устройство управления (control unit) относительно простое по сравнению с CPU
- GPU управляется с CPU: копирование данных между оперативной памятью узла и GPU, запуск программ и др.

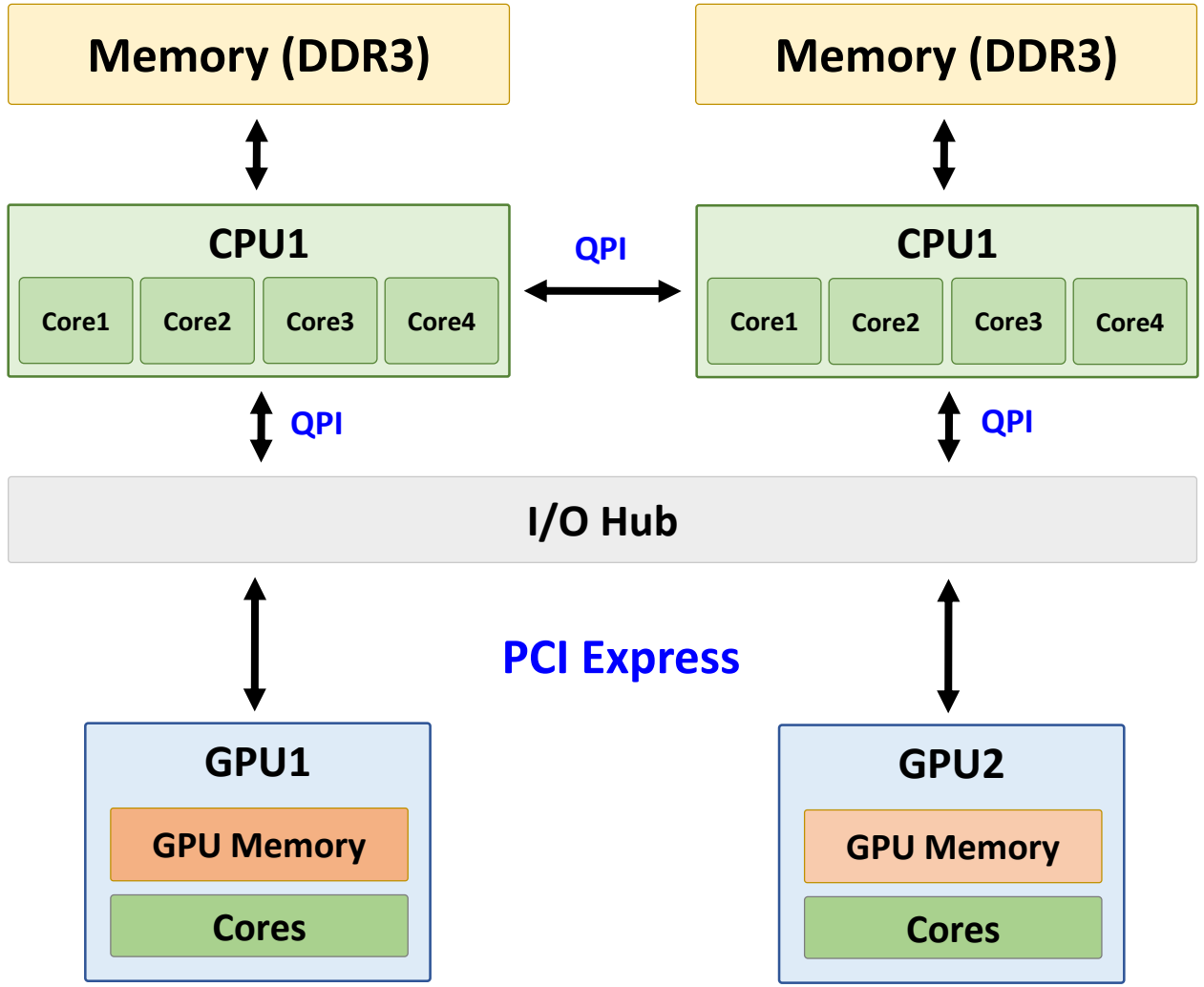


NVIDIA GeForce GTX 780
(Kepler, **2304 cores**, GDDR5 3 GB)

AMD Radeon HD 8970
(**2048 cores**, GDDR5 3 GB)

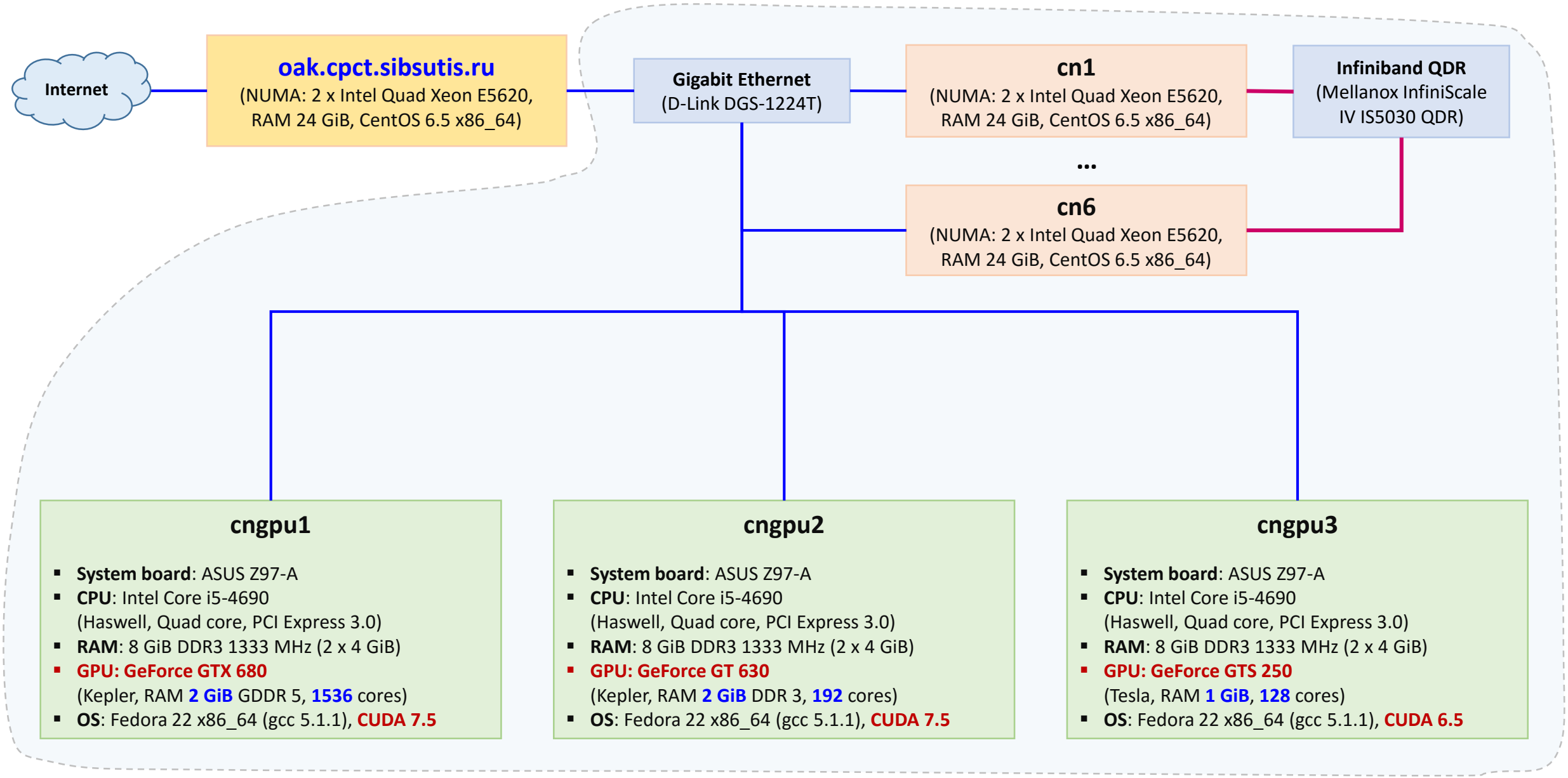
Гибридные вычислительные узлы

- Несколько GPU
- CPU управляет GPU через драйвер
- Узкое место – передача данных между памятью CPU и GPU
- GPU не является bootable-устройством



74 системы из Top500 (Nov. 2014)
оснащены ускорителями (NVIDIA, ATI, Intel Xeon Phi, PEZY SC)

GPU-подсистема кластера oak.cpct.sibsutis.ru



NVIDIA CUDA

- **NVIDIA CUDA** – программно-аппаратная платформа для организации параллельных вычислений на графических процессорах

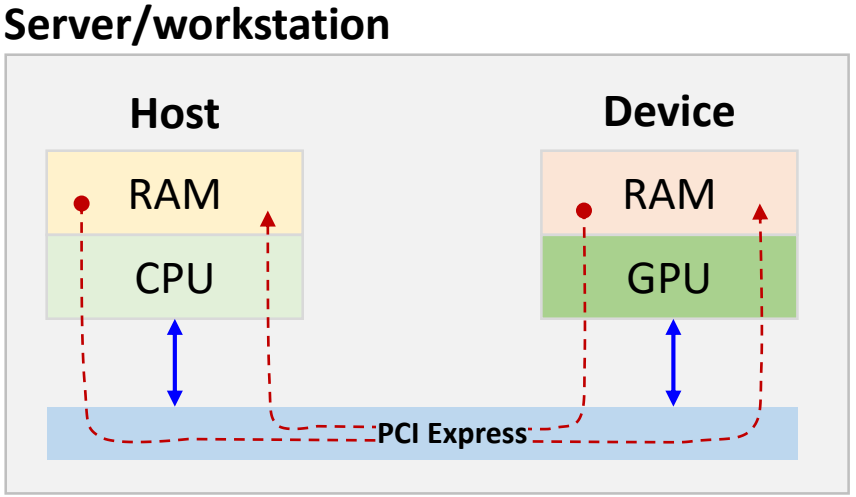
- **NVIDIA CUDA SDK**:
  - архитектура виртуальной машины CUDA
  - компилятор C/C++
  - драйвер GPU

- **ОС**: GNU/Linux, Apple Mac OS X, Microsoft Windows
- **2006** – CUDA 1.0
- **2015 – CUDA 7.5** (Unified memory, Multi-GPU)
- **Микроархитектуры**: **Tesla** (GeForce 8), **Fermi** (GeForce 400, 500), **Kepler** (GeForce 600, 700), **Maxwell** (GeForce 700, 800, 900)

http://developer.nvidia.com/cuda

Основные понятия NVIDIA CUDA

- **Хост (host)** – узел с CPU и его память
- **Устройство (device)** – графический процессор и его память
- **Ядро (kernel)** – это фрагмент программы, предназначенный для выполнения на GPU



- CUDA-программа и ее входные данные находятся в памяти хоста
- Программа компилируется и запускается на хосте
- В CUDA-программе выполняются следующие шаги:

1. Копирование данных из памяти хоста в память устройства
2. Запуск ядра (kernel) на устройстве
3. Копирование данных из памяти устройства в память хоста

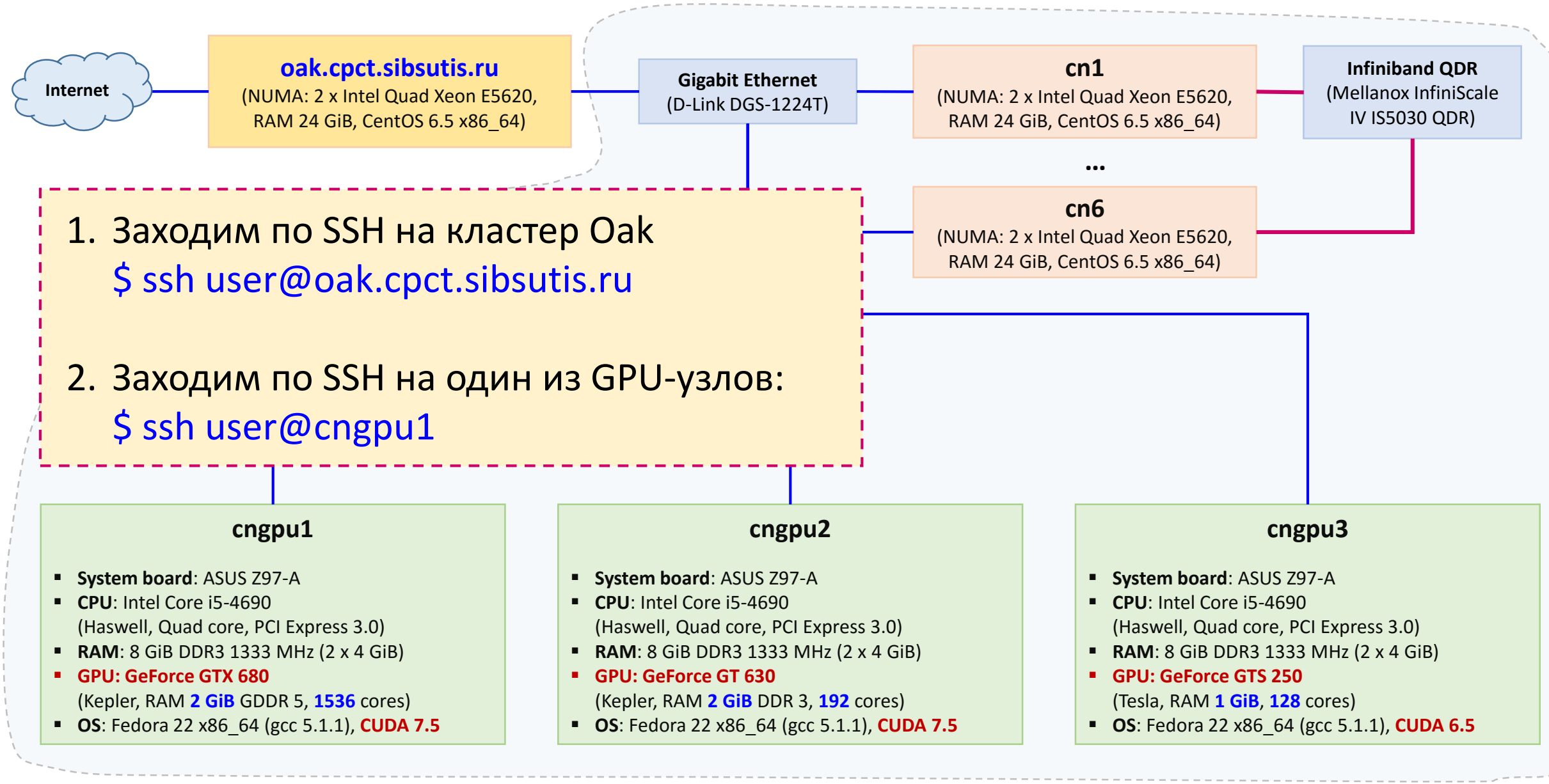
CUDA Hello World

```
/*
 * hello.cu:
 *
 */

#include <stdio.h>

__global__ void mykernel()
{

}

int main()
{
    /* Запуск ядра на GPU – один поток */
    mykernel<<<1,1>>>();
    printf("Hello, CUDA World!\n");
    return 0;
}
```

Спецификатор **__global__** сообщает компилятору, что функция предназначена для выполнения на GPU

“<<< >>>” – запуск заданного числа потоков на GPU

GPU-подсистема кластера oak.cpct.sibsutis.ru



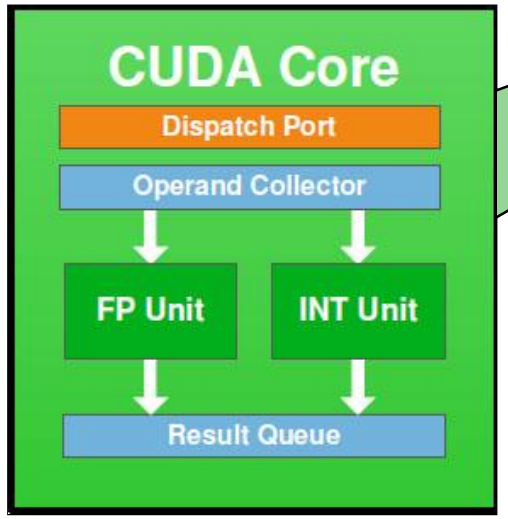
CUDA Hello World

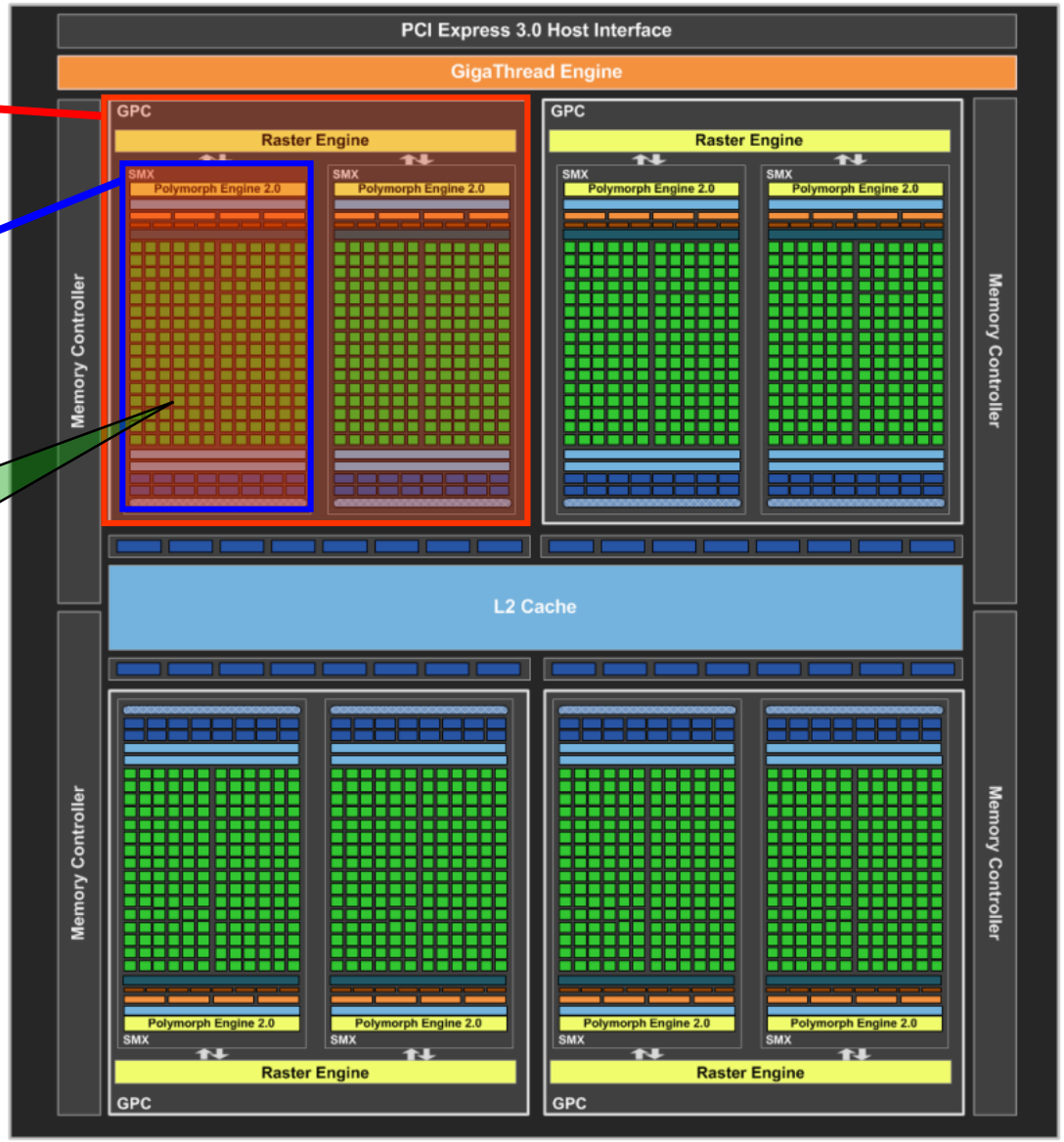
```
# Подключаемся по SSH к узлу с GPU: cngpu1, cngpu2 или cngpu3
$ ssh user1@cngpu1

# Компиляция CUDA-программы
$ nvcc –o hello ./hello.cu

# Подготовка паспорта задачи для системы очередей SLURM
$ cat ./task.job
#!/bin/bash

#SBATCH --job-name=MyCUDAJob

./hello

# Запуск задачи через систему очередей SLURM
$ sbatch ./task.job
Submitted batch job 75

$ cat ./slurm-75.out
Hello, CUDA World!
```

NVIDIA GeForce 680 (GK104, Kepler microarch.)

4 Graphics Processing Clusters (GPC)
2 Streaming Multiprocessor (SMX)

Streaming Multiprocessor (SMX)
192 cores, 32 special function units,
32 LD/ST units, 4 warp schedulers





http://www.nvidia.com/content/PDF/product-specifications/GeForce_GTX_680_Whitepaper_FINAL.pdf

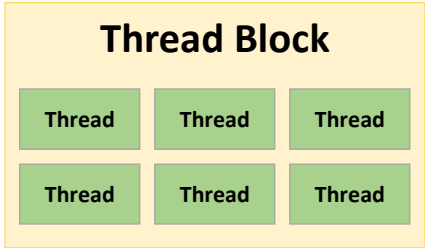
Информация об устройстве (cngpu1)

```
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "GeForce GTX 680"
  CUDA Driver Version / Runtime Version          7.5 / 7.5
  CUDA Capability Major/Minor version number:    3.0
  Total amount of global memory:                 2048 MBytes (2147287040 bytes)
  ( 8) Multiprocessors, (192) CUDA Cores/MP:     1536 CUDA Cores
  GPU Max Clock rate:                            1058 MHz (1.06 GHz)
  Memory Clock rate:                             3004 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 524288 bytes
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  ...
```

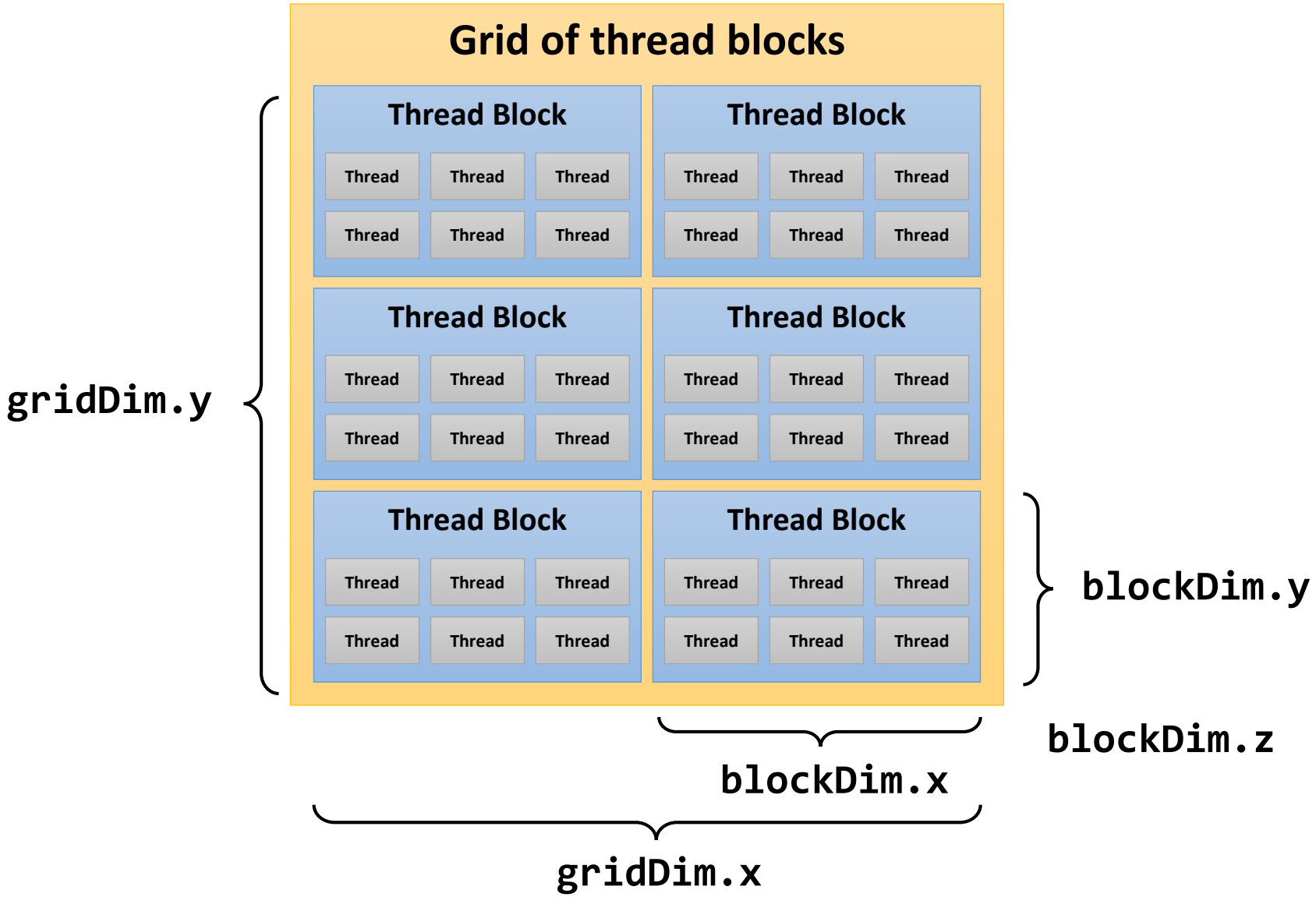
/opt/NVIDIA_CUDA-7.5_Samples

Вычислительные потоки CUDA

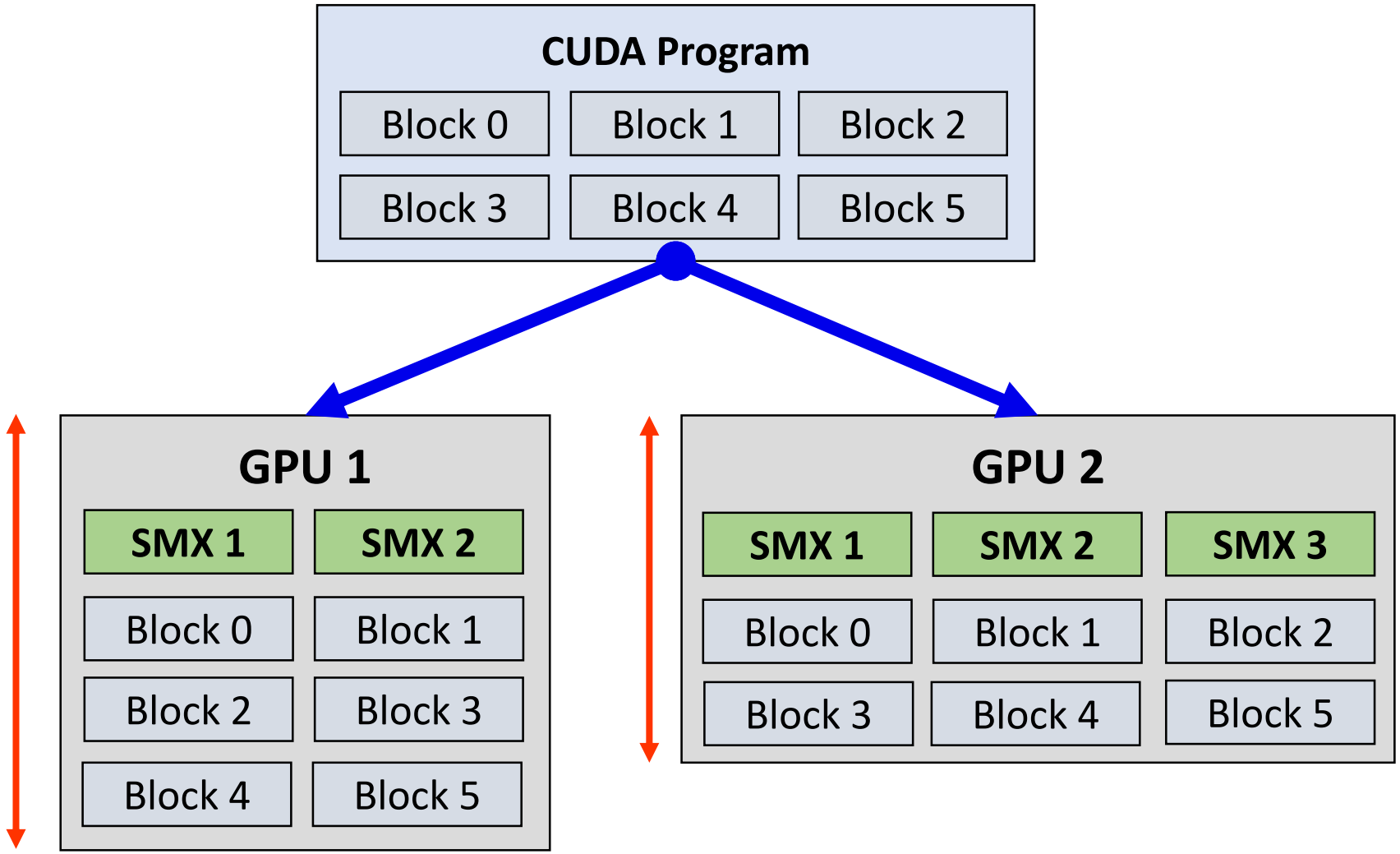
- **Номер потока (thread index)** – это трехкомпонентный вектор (координаты потока)
- Потоки логически сгруппированы в одномерный, двухмерный или трёхмерный *блок* (thread block)
- Количество потоков в блоке ограничено (в Kepler 1024)
- Блоки распределяются по потоковым мультипроцессорам SMX
- Предопределенные переменные
  - **`threadIdx.{x, y, z}`** – номер потока
  - **`blockDim.{x, y, z}`** – размерность блока
  - **`blockIdx.{x, y, z}`** – номер блока



Вычислительные потоки CUDA



Выполнение CUDA-программы



Сложение векторов (CPU version)

```
/*
 * Host code (CPU version)
 */
void vadd(float *a, float *b, float *c, int n)
{
    for (int i = 0; i < n; i++)
        c[i] = a[i] + b[i];
}
```

Сложение векторов (GPU version)

```
#include <cuda_runtime.h>

enum { NELEMS = 1024 * 1024 };

int main()
{
    /* Allocate vectors on host */
    size_t size = sizeof(float) * NELEMS;
    float *h_A = malloc(size);
    float *h_B = malloc(size);
    float *h_C = malloc(size);
    if (h_A == NULL || h_B == NULL || h_C == NULL) {
        fprintf(stderr, "Allocation error.\n");
        exit(EXIT_FAILURE);
    }

    for (int i = 0; i < NELEMS; ++i) {
        h_A[i] = rand() / (float)RAND_MAX;
        h_B[i] = rand() / (float)RAND_MAX;
    }
```

Сложение векторов (GPU version)

```
/* Allocate vectors on device */
float *d_A = NULL, *d_B = NULL, *d_C = NULL;
if (cudaMalloc((void **)&d_A, size) != cudaSuccess) {
    fprintf(stderr, "Allocation error\n");
    exit(EXIT_FAILURE);
}
if (cudaMalloc((void **)&d_B, size) != cudaSuccess) {
    fprintf(stderr, "Allocation error\n");
    exit(EXIT_FAILURE);
}
if (cudaMalloc((void **)&d_C, size) != cudaSuccess) {
    fprintf(stderr, "Allocation error\n");
    exit(EXIT_FAILURE);
}
```

Сложение векторов (GPU version)

```
/* Copy the host vectors to device */
if (cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice) != cudaSuccess) {
    fprintf(stderr, "Host to device copying failed\n");
    exit(EXIT_FAILURE);
}
if (cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice) != cudaSuccess) {
    fprintf(stderr, "Host to device copying failed\n");
    exit(EXIT_FAILURE);
}

/* Launch the kernel */
int threadsPerBlock = 1024;
int blocksPerGrid =(NELEMS + threadsPerBlock - 1) / threadsPerBlock;
vadd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, NELEMS);
if (cudaGetLastError() != cudaSuccess) {
    fprintf(stderr, "Failed to launch kernel!\n");
    exit(EXIT_FAILURE);
}
```

$$blocks = \left\lceil \frac{N}{threads} \right\rceil = \left\lfloor \frac{N + threads - 1}{threads} \right\rfloor$$

Сложение векторов (GPU version)

```
    /* Copy the device vectors to host */
    if (cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost) != cudaSuccess) {
        fprintf(stderr, "Device to host copying failed\n");
        exit(EXIT_FAILURE);
    }

    for (int i = 0; i < NELEMS; ++i) {
        if (fabs(h_A[i] + h_B[i] - h_C[i]) > 1e-5) {
            fprintf(stderr, "Result verification failed at element %d!\n", i);
            exit(EXIT_FAILURE);
        }
    }

    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);
    free(h_A);
    free(h_B);
    free(h_C);
    cudaDeviceReset();
    return 0;
}
```

Сложение векторов (GPU version)

```
__global__ void vadd(const float *a, const float *b, float *c, int n)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < n)
        c[i] = a[i] + b[i];
}
```

Сложение векторов (scalability)

NELEMS	cngpu1 GeForce GTX 680	cngpu2 GeForce GT 630	cngpu3 GeForce GTS 250 (threadsPerBlock = 512)
2^20	CPU version (sec.): 0.001658 GPU version (sec.): 0.000161 Memory ops. (sec.): 0.002341 Memory bw. (MiB/sec.): 5125.94 CPU perf (MFLOPS): 603.15 GPU perf (MFLOPS): 6213.78 **Speedup: 10.30** Speedup (with mem ops.): 0.66	CPU version (sec.): 0.002311 GPU version (sec.): 0.001054 Memory ops. (sec.): 0.002514 Memory bw. (MiB/sec.): 4773.49 CPU perf (MFLOPS): 432.71 GPU perf (MFLOPS): 948.72 **Speedup: 2.19** Speedup (with mem ops.): 0.65	CPU version (sec.): 0.002195 GPU version (sec.): 0.001993 Memory ops. (sec.): 0.009481 Memory bw. (MiB/sec.): 1265.70 CPU perf (MFLOPS): 455.61 GPU perf (MFLOPS): 501.77 **Speedup: 1.10** Speedup (with mem ops.): 0.19
10 * 2^20	CPU version (sec.): 0.015133 GPU version (sec.): 0.000892 Memory ops. (sec.): 0.021551 Memory bw. (MiB/sec.): 5568.15 CPU perf (MFLOPS): 660.81 GPU perf (MFLOPS): 11211.72 **Speedup: 16.97** Speedup (with mem ops.): 0.67	CPU version (sec.): 0.014987 GPU version (sec.): 0.009118 Memory ops. (sec.): 0.021882 Memory bw. (MiB/sec.): 5484.00 CPU perf (MFLOPS): 667.25 GPU perf (MFLOPS): 1096.72 **Speedup: 1.64** Speedup (with mem ops.): 0.48	CPU version (sec.): 0.017394 GPU version (sec.): 0.006778 Memory ops. (sec.): 0.101824 Memory bw. (MiB/sec.): 1178.51 CPU perf (MFLOPS): 574.91 GPU perf (MFLOPS): 1475.36 **Speedup: 2.57** Speedup (with mem ops.): 0.16

Задания

- Подключиться к кластеру oak.cpct.sibsutis.ru и зайти на любой из GPU-узлов
- Скомпилировать и запустить через систему очередей пример addvec_prof (/home/pub/)
- Модифицировать пример addvec_prof для работы с массивами типа double. Оценить производительность программы.

- Реализовать CUDA-версию функции SAXPY ($y_i = \alpha \cdot x_i + y_i$)

```
__global__ void saxpy(float *x, float *y, float a, int n)
```