Семинар 14
Технология CUDA
Потоки CUDA

Михаил Курносов

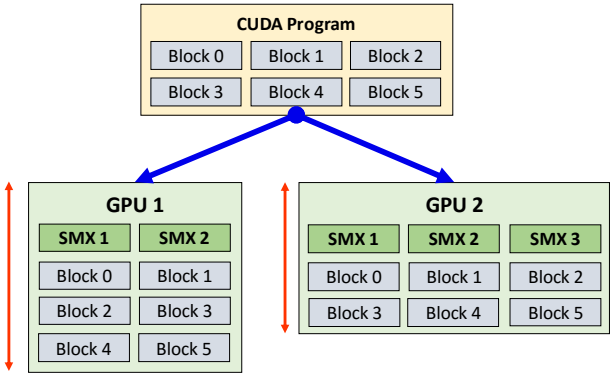
E-mail: mkurnosov@gmail.com
WWW: www.mkurnosov.net

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

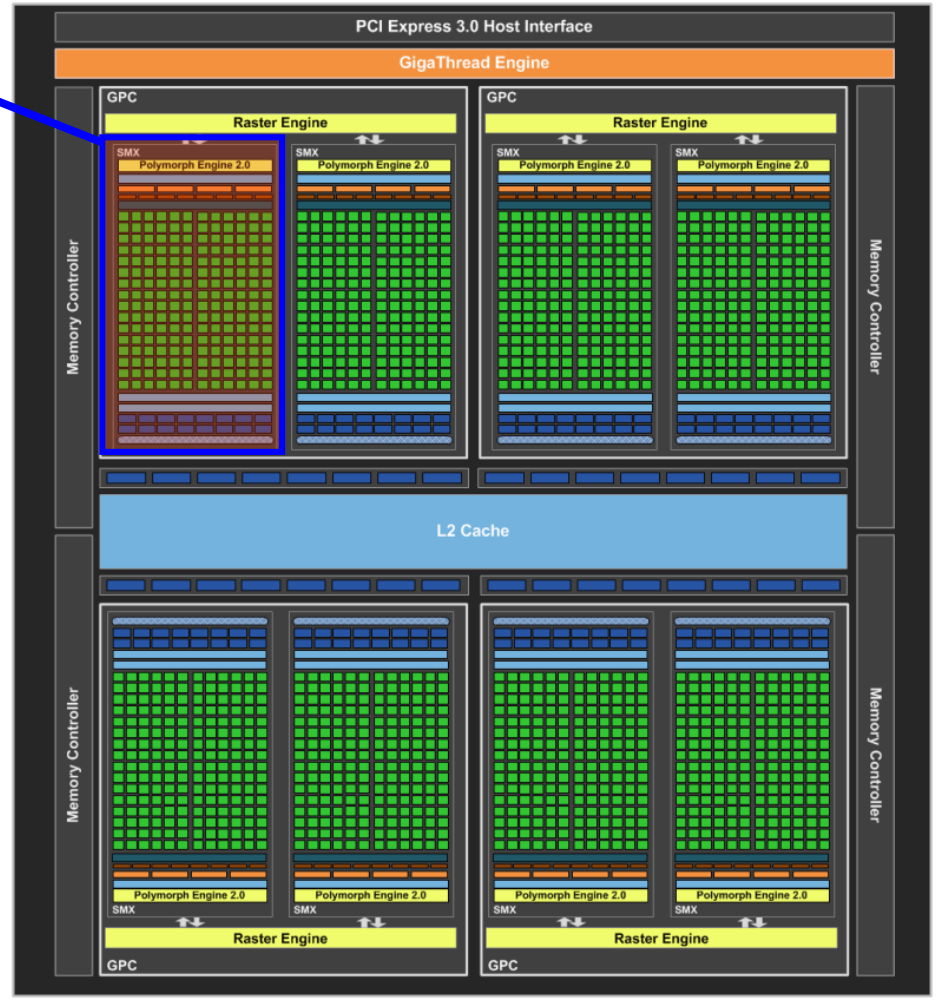
Архитектура SIMT (Single-Instruction, Multiple-Thread)

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

- Блоки потоков распределяются по SM
- Если число блоков меньше количества SM, то часть процессоров GPU простаивает
- Порядок выполнения блоков заранее неизвестен – алгоритмы для GPU не должны быть чувствительны к порядку выполнения блоков



NVIDIA GeForce 680 (GK104, Kepler)

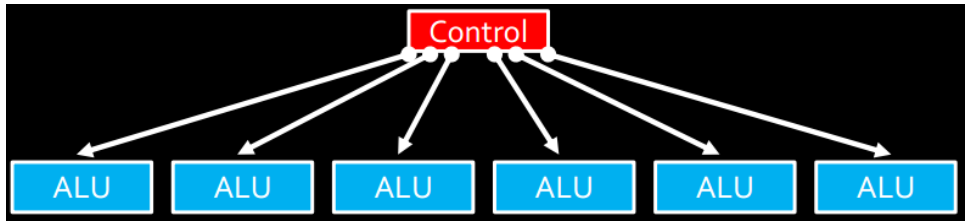


http://www.nvidia.com/content/PDF/product-specifications/GeForce_GTX_680_Whitepaper_FINAL.pdf

Архитектура SIMT (Single-Instruction, Multiple-Thread)

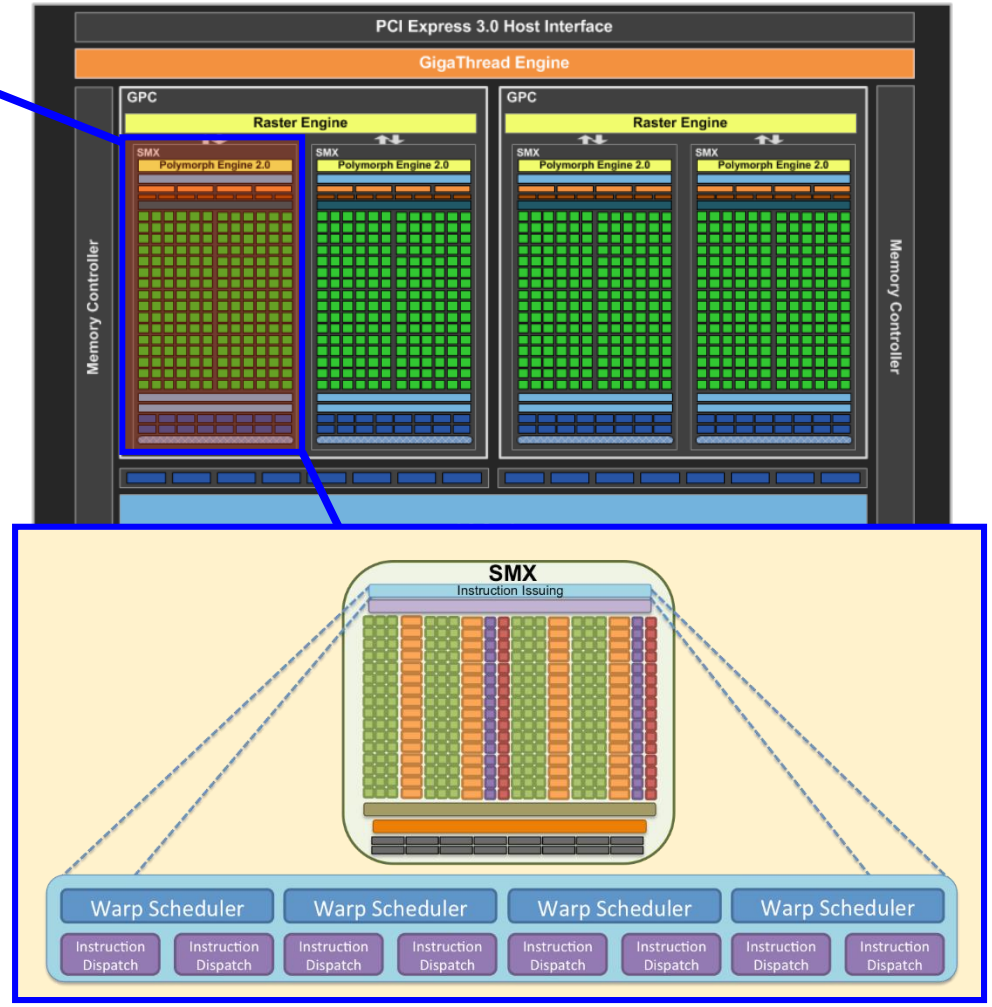
Streaming Multiprocessor (SM)
192 cores, 32 special function units,
32 LD/ST units, **4 warp schedulers**

- Блоки назначенные на SM логически разбиваются на **группы (warps)** по 32 потока
- Потоки всегда одинаково распределяются по warp-ам (детерминировано, на базе номера потока)
- Если размер блока не кратен размеру группы, то последняя группа (warp) потоков блокируется



http://mc.stanford.edu/cgi-bin/images/3/34/Darve_cme343_cuda_3.pdf

NVIDIA GeForce 680 (GK104, Kepler)

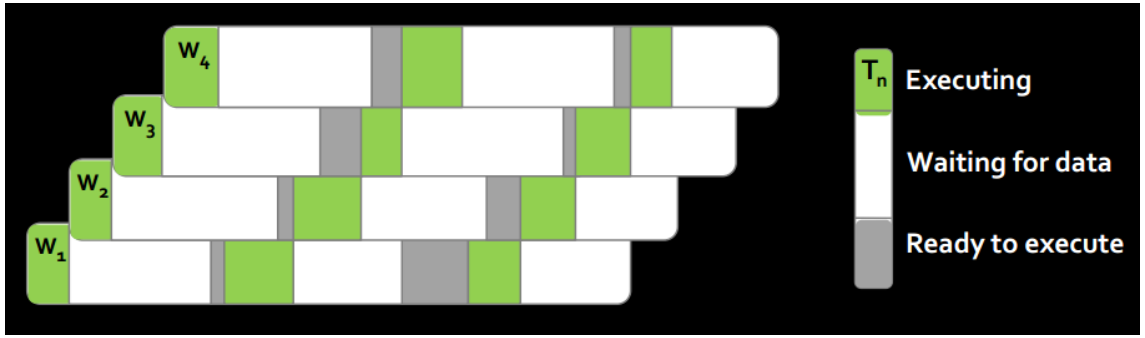


http://www.nvidia.com/content/PDF/product-specifications/GeForce_GTX_680_Whitepaper_FINAL.pdf

Архитектура SIMT (Single-Instruction, Multiple-Thread)

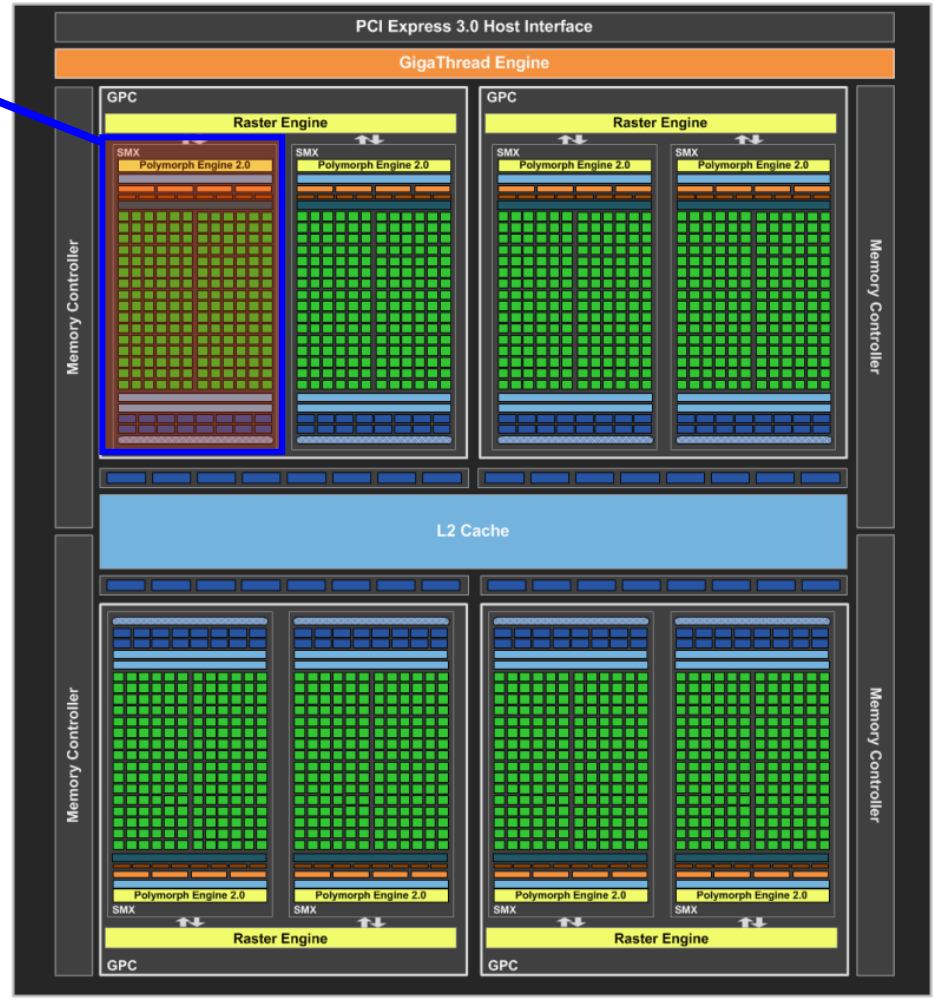
Streaming Multiprocessor (SM)
192 cores, 32 special function units,
32 LD/ST units, **4 warp schedulers**

- Планировщик (warp scheduler) запускает на выполнение группу потоков, у которых «готовы» входные данные
- Все активные потоки группы в каждый момент времени выполняют одну и ту же инструкцию
- Планировщик выполняет переключение контекста между группами потоков



http://mc.stanford.edu/cgi-bin/images/3/34/Darve_cme343_cuda_3.pdf

NVIDIA GeForce 680 (GK104, Kepler)

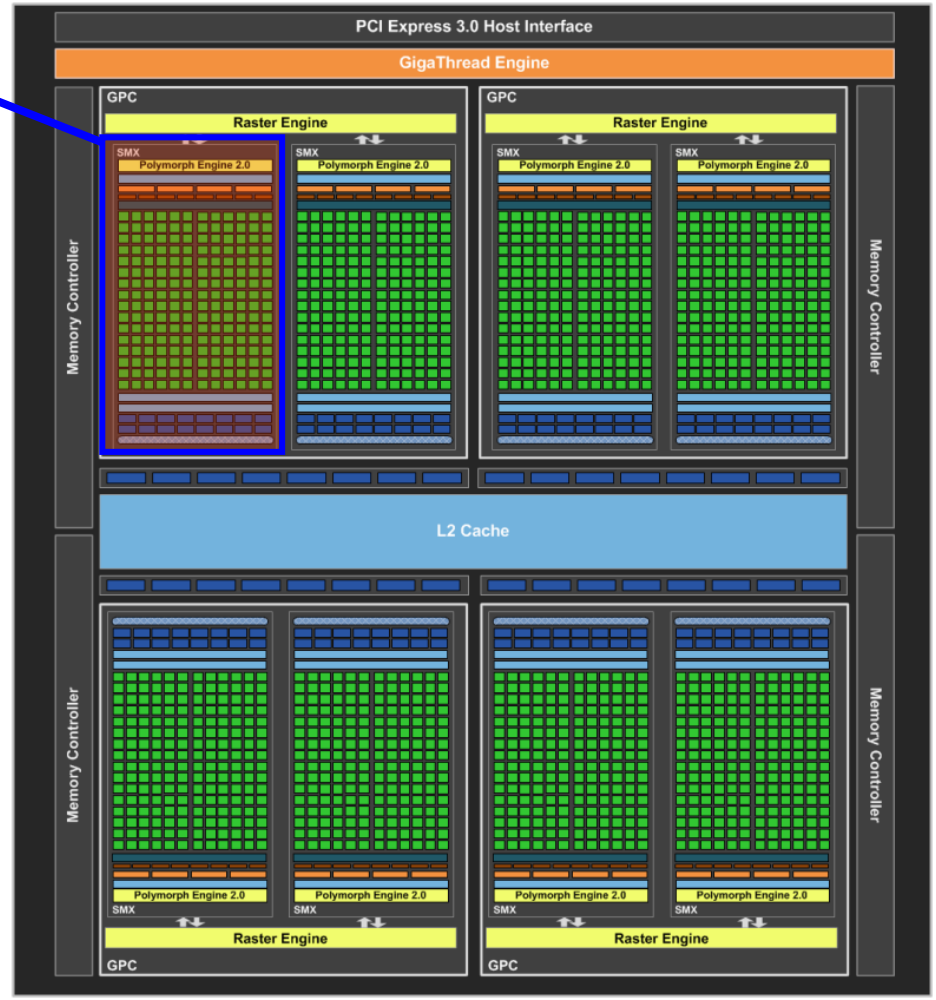


http://www.nvidia.com/content/PDF/product-specifications/GeForce_GTX_680_Whitepaper_FINAL.pdf

Архитектура SIMT (Single-Instruction, Multiple-Thread)

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

NVIDIA GeForce 680 (GK104, Kepler)



Control Flow Divergence

- Все потоки группы в каждый момент времени выполняют одну и ту же инструкцию
- Что если потоки управления расходятся?

```
__global__ void kernel()
{
    if (val < 50) {
        // branch 1
    } else {
        // branch 2
    }
}
```

Потоки группы выполняют обе ветви

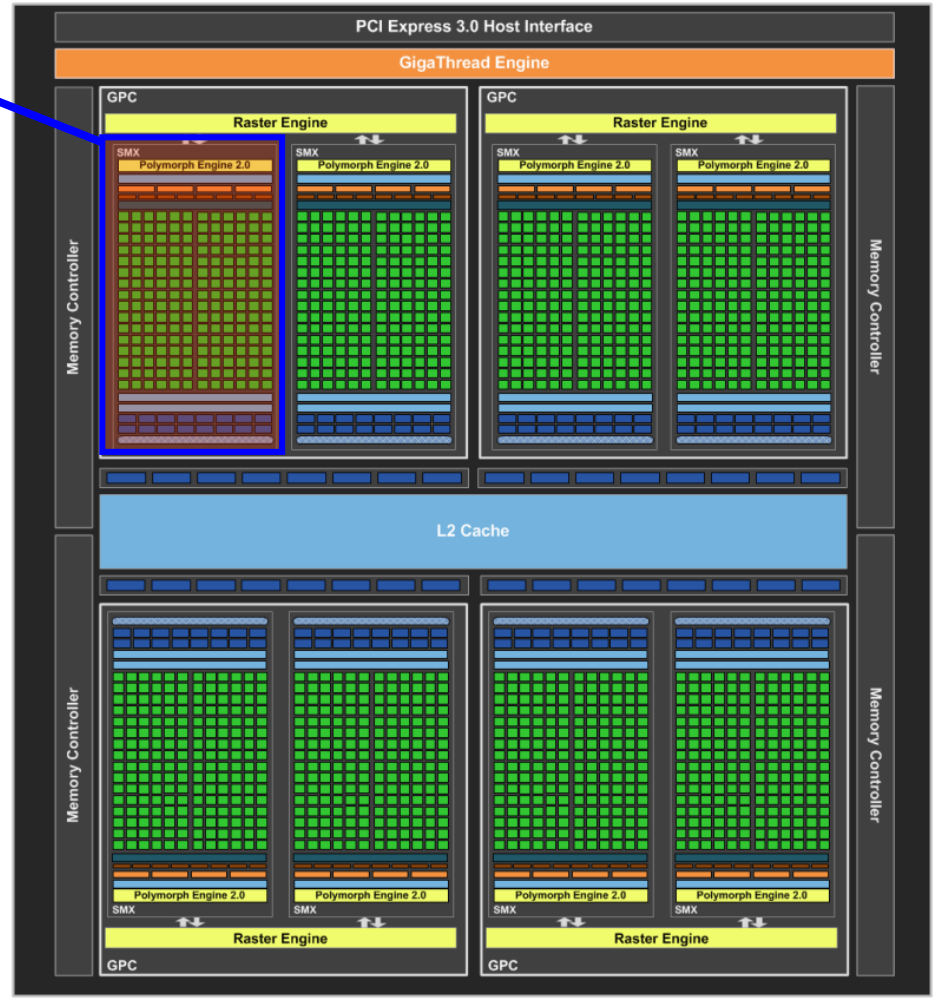
Потоки, которые не должны выполнять ветвь, блокируются

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

Архитектура SIMT (Single-Instruction, Multiple-Thread)

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

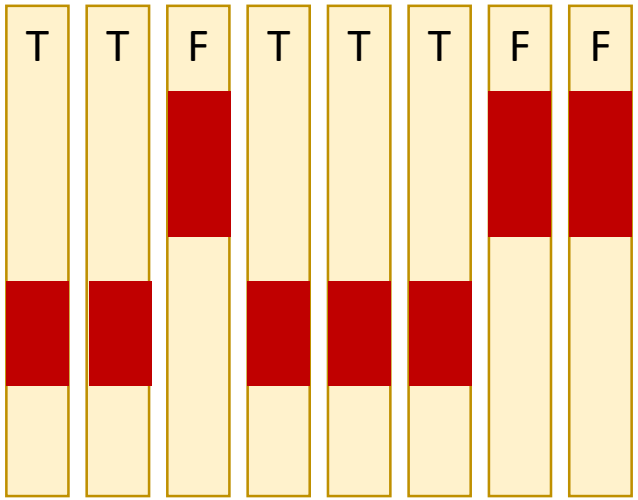
NVIDIA GeForce 680 (GK104, Kepler)



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

Control Flow Divergence

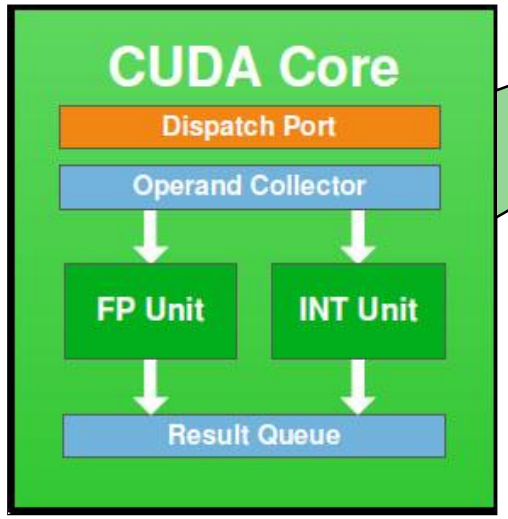
```
__global__ void kernel()
{
    if (val < 50) {
        // branch 1
        // branch 1
        // branch 1
    } else {
        // branch 2
        // branch 2
    }
}
```

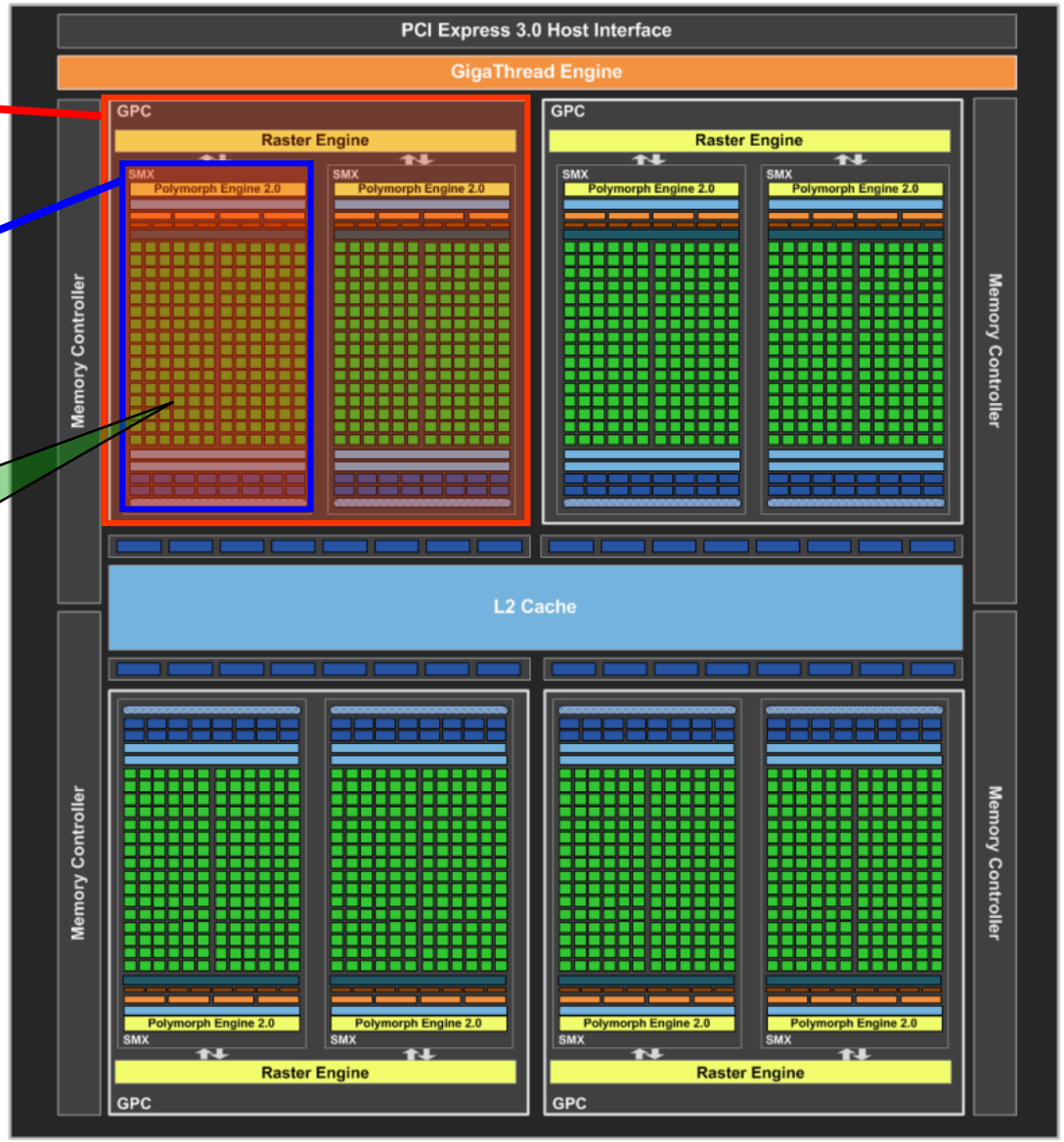


NVIDIA GeForce 680 (GK104, Kepler microarch.)

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





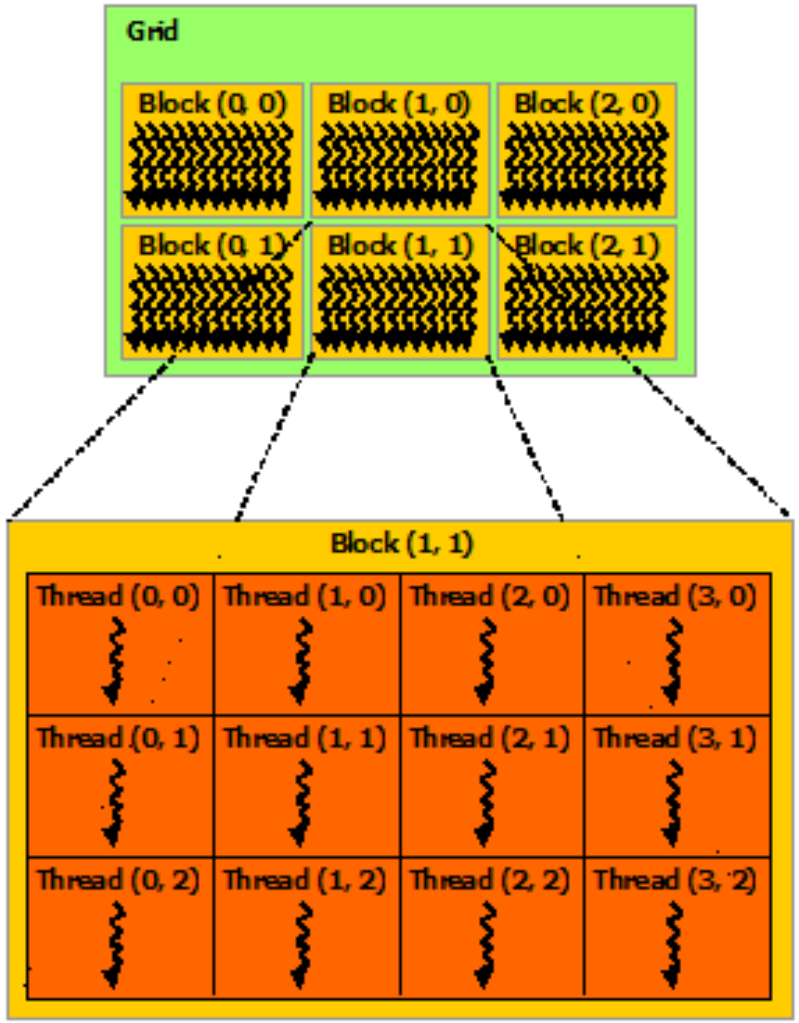
http://www.nvidia.com/content/PDF/product-specifications/GeForce_GTX_680_Whitepaper_FINAL.pdf

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

Иерархия потоков



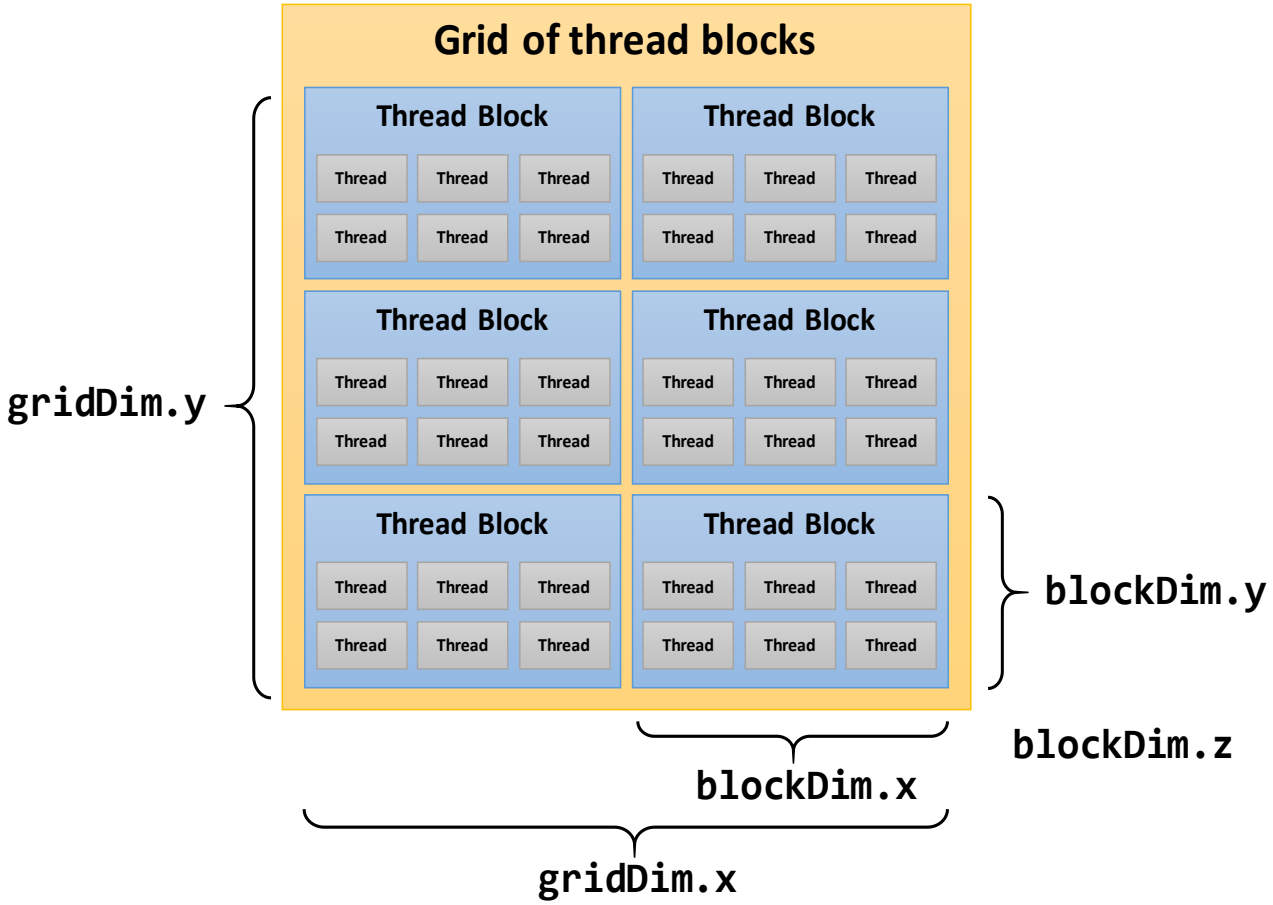
- **Сетка потоков** (1D, 2D, 3D) – совокупность блоков потоков
- Сетка потоков, CUDA-программа, выполняется GPU

- **Блок потоков** (1D, 2D, 3D) – совокупность потоков
- Блоки распределяются по потоковым мультипроцессорам

```
Device 0: "GeForce GTX 680"
  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)
  ...
```

Иерархия потоков

- **Предопределенные переменные**
  - **`threadIdx.{x, y, z}`** – номер потока в блоке
  - **`blockDim.{x, y, z}`** – размерность блока
  - **`blockIdx.{x, y, z}`** – номер блока в сетке

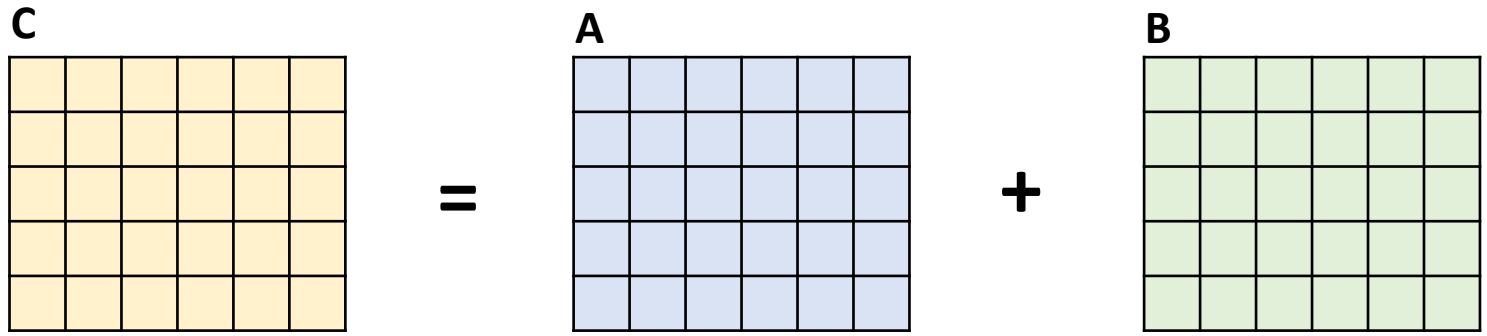


- **Здание структуры сетки**

```
mykernel<<<B,T>>>(arg1, … );
```

  - B – структура сетки
  - T – структура блока

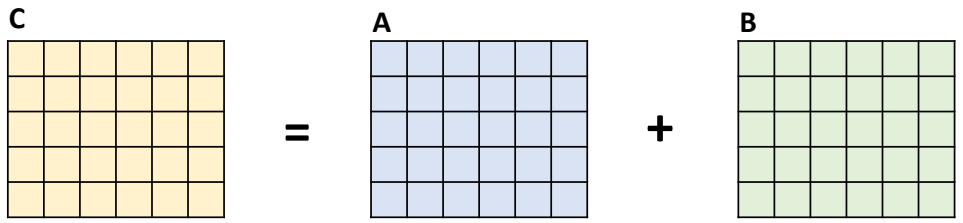
Сложение матриц: CPU



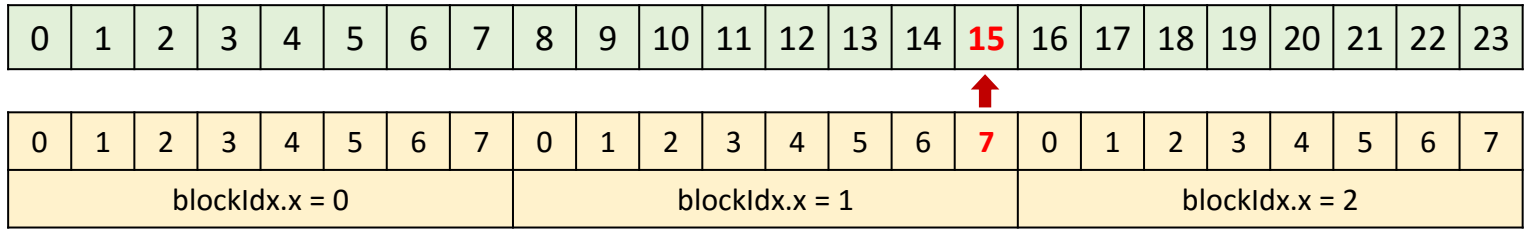

```
void matadd_host(float *a, float *b, float *c, int m, int n)
{
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            int idx = i * n + j;
            c[idx] = a[idx] + b[idx];
        }
    }
}
```

Сложение матриц: CUDA 1D

- Каждый поток вычисляет один элемент c[i * n + j] матрицы
- Одномерные сетка блоков потоков и одномерные блоки потоков



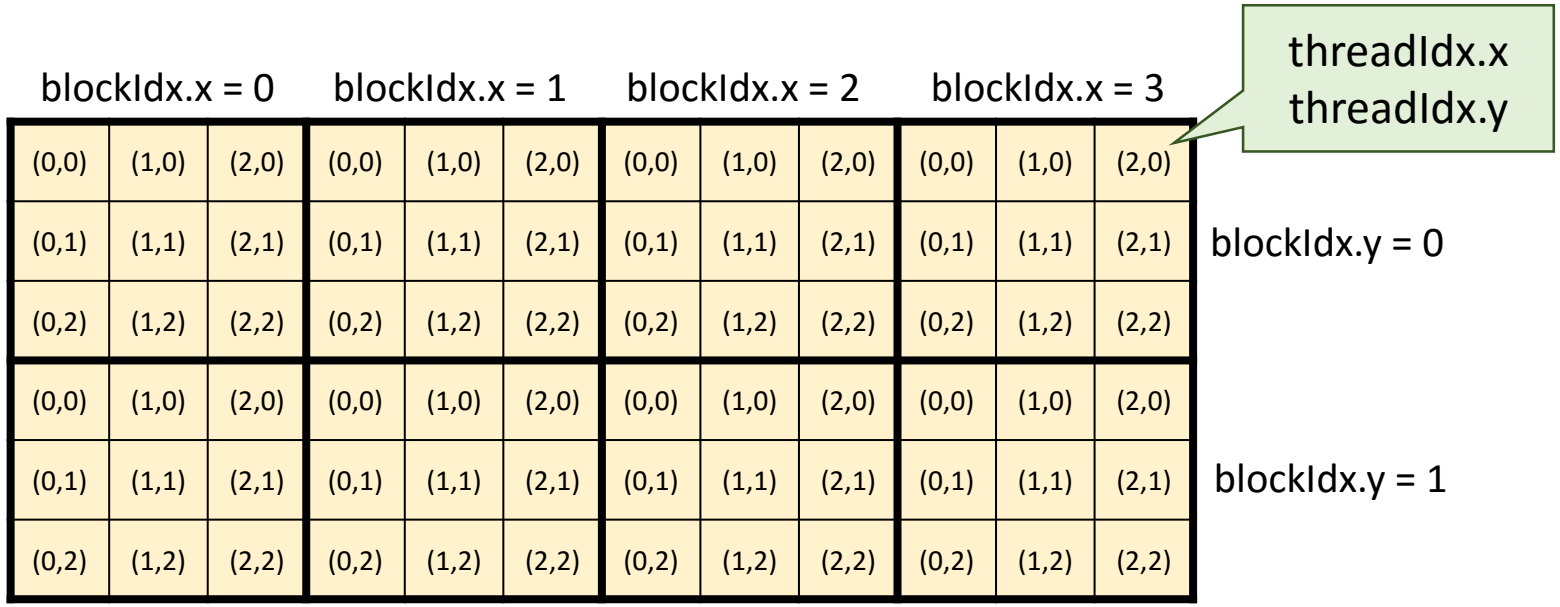
1D-сетка из
3-х 1D-блоков потоков




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

{
    int threadsPerBlock = 1024;
    int blocksPerGrid =(ROWS * COLS + threadsPerBlock - 1) / threadsPerBlock;
    matadd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, ROWS, COLS);
}
```

Сложение матриц: CUDA 2D

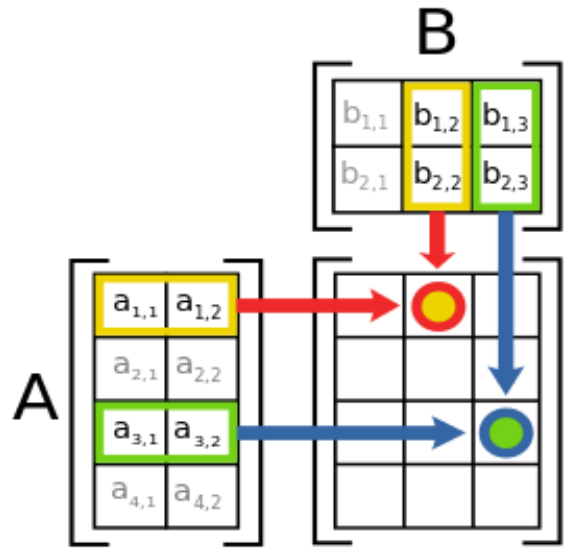
- Каждый поток вычисляет один элемент c[i * n + j] матрицы
- 2D-сетка блоков потоков и 2D-блоки потоков




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

{
    int threadsPerBlockDim = 32;
    dim3 blockDim(threadsPerBlockDim, threadsPerBlockDim, 1);            // Grid: X x Y x Z=1
    int blocksPerGridDimX = ceilf(COLS / (float)threadsPerBlockDim);
    int blocksPerGridDimY = ceilf(ROWS / (float)threadsPerBlockDim);
    dim3 gridDim(blocksPerGridDimX, blocksPerGridDimY, 1);               // Blocks: X x Y x Z=1
    matadd<<<gridDim, blockDim>>>(d_A, d_B, d_C, ROWS, COLS);
}
```

Умножение матриц (CPU)

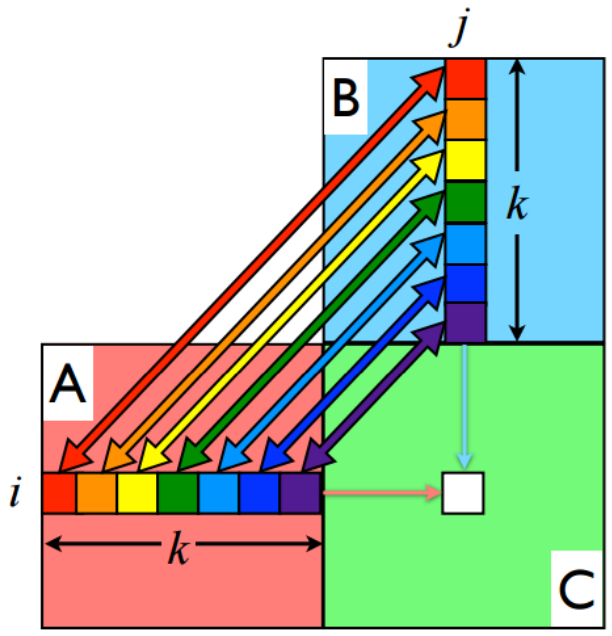
```
void sgemm_host(float *a, float *b, float *c, int n)
{
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            float s = 0.0;
            for (int k = 0; k < n; k++)
                s += a[i * n + k] * b[k * n + j];
            c[i * n + j] = s;
        }
    }
}
```



Умножение матриц (CUDA)

- **Каждый поток вычисляет один элемент результирующей матрицы C**
- Общее число потоков N * N

```
__global__ void sgemm(const float *a, const float *b, float *c, int n)
{
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    if (row < n && col < n) {
        float s = 0.0;
        for (int k = 0; k < n; k++)
            s += a[row * n + k] * b[k * n + col];
        c[row * n + col] = s;
    }
}
{
    int threadsPerBlockDim = 32;
    dim3 blockDim(threadsPerBlockDim, threadsPerBlockDim, 1);
    int blocksPerGridDimX = ceilf(N / (float)threadsPerBlockDim);
    int blocksPerGridDimY = ceilf(N / (float)threadsPerBlockDim);
    dim3 gridDim(blocksPerGridDimX, blocksPerGridDimY, 1);
    sgemm<<<gridDim, blockDim>>>(d_A, d_B, d_C, N);
}
```



http://users.wfu.edu/choss/CUDA/docs/Lecture%205.pdf