



Архитектура **Tesla.**

Программно-аппаратный стек  
**CUDA.**

- Лекторы:
  - Боресков А.В. (ВМиК МГУ)
  - Харламов А.А. (Харламов А.А. (Nvidia) Харламов А.А. (Nvidia))

# Примеры многоядерных систем



- На первой лекции мы рассмотрели
  - Intel Core 2 Duo
  - SMP
  - Cell
  - BlueGene/L
  - G80 / Tesla

# Примеры многоядерных систем



- Мы хотели обратить ваше внимание на следующие особенности:
  - 1) Как правило вычислительный узел – достаточно маломощный процессор
  - 2) Вычислительные узлы имеют свою оперативную память и свой кэш
  - 3) Вычислительные узлы объединяются в более крупные блоки
  - 4) Крупные блоки могут объединяться с целью наращивания вычислительной мощности

# Tesla vs GeForce



- У кого есть вопросы в чем разница?

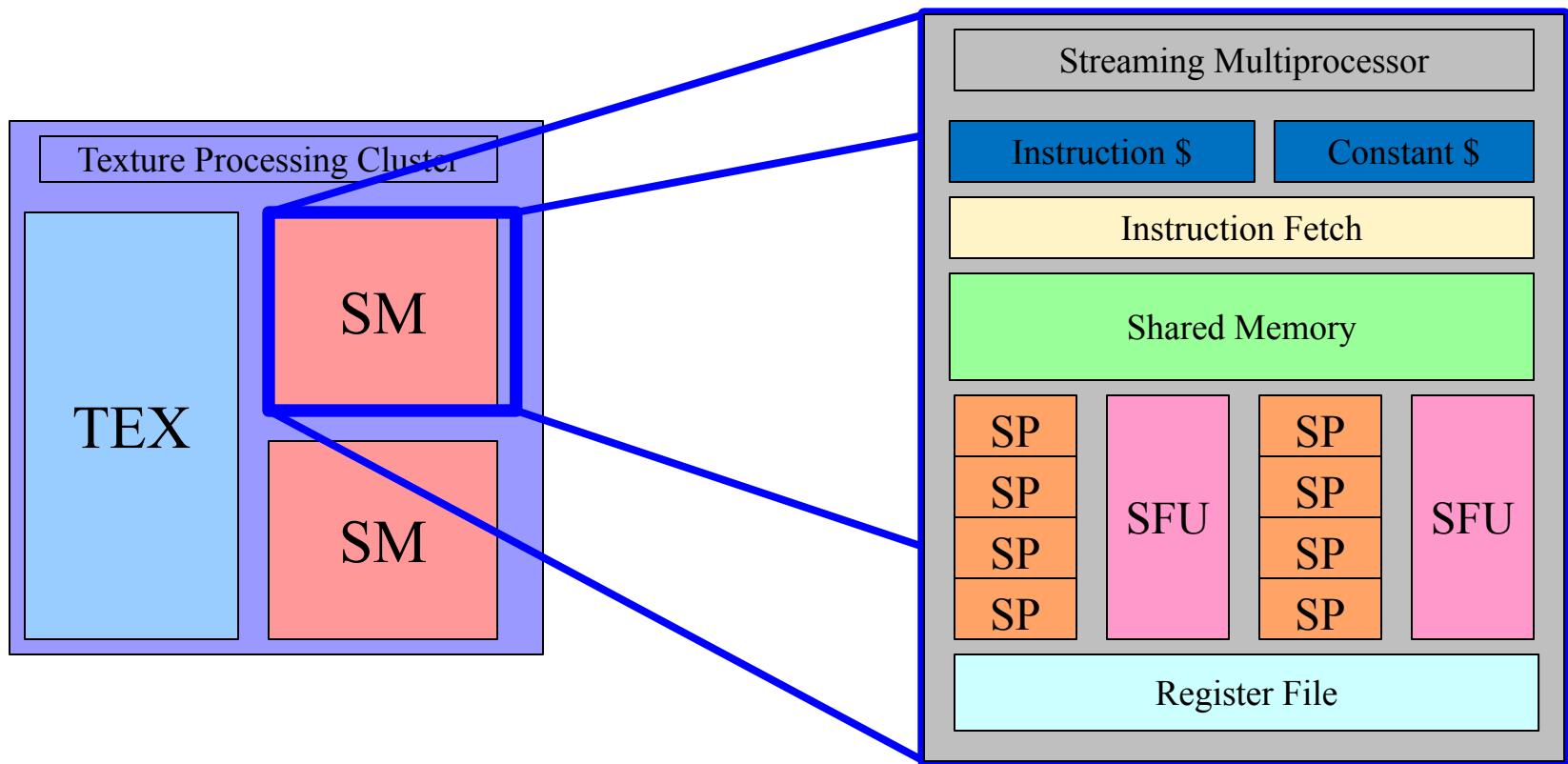
# План



- Архитектура Tesla
- Программная модель CUDA
- Синтаксические особенности CUDA

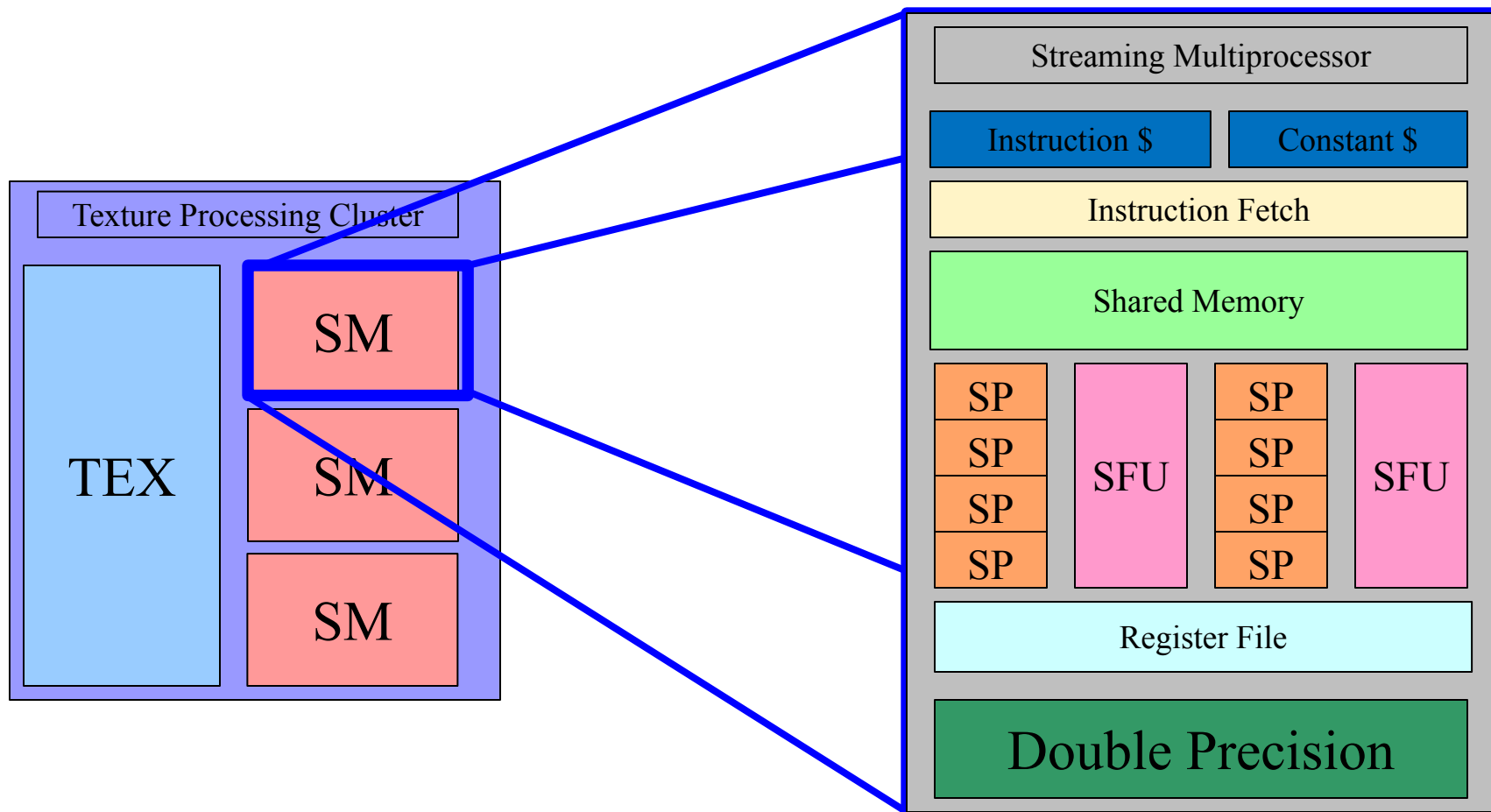
# Архитектура **Tesla**:

## Мультипроцессор **Tesla 8**

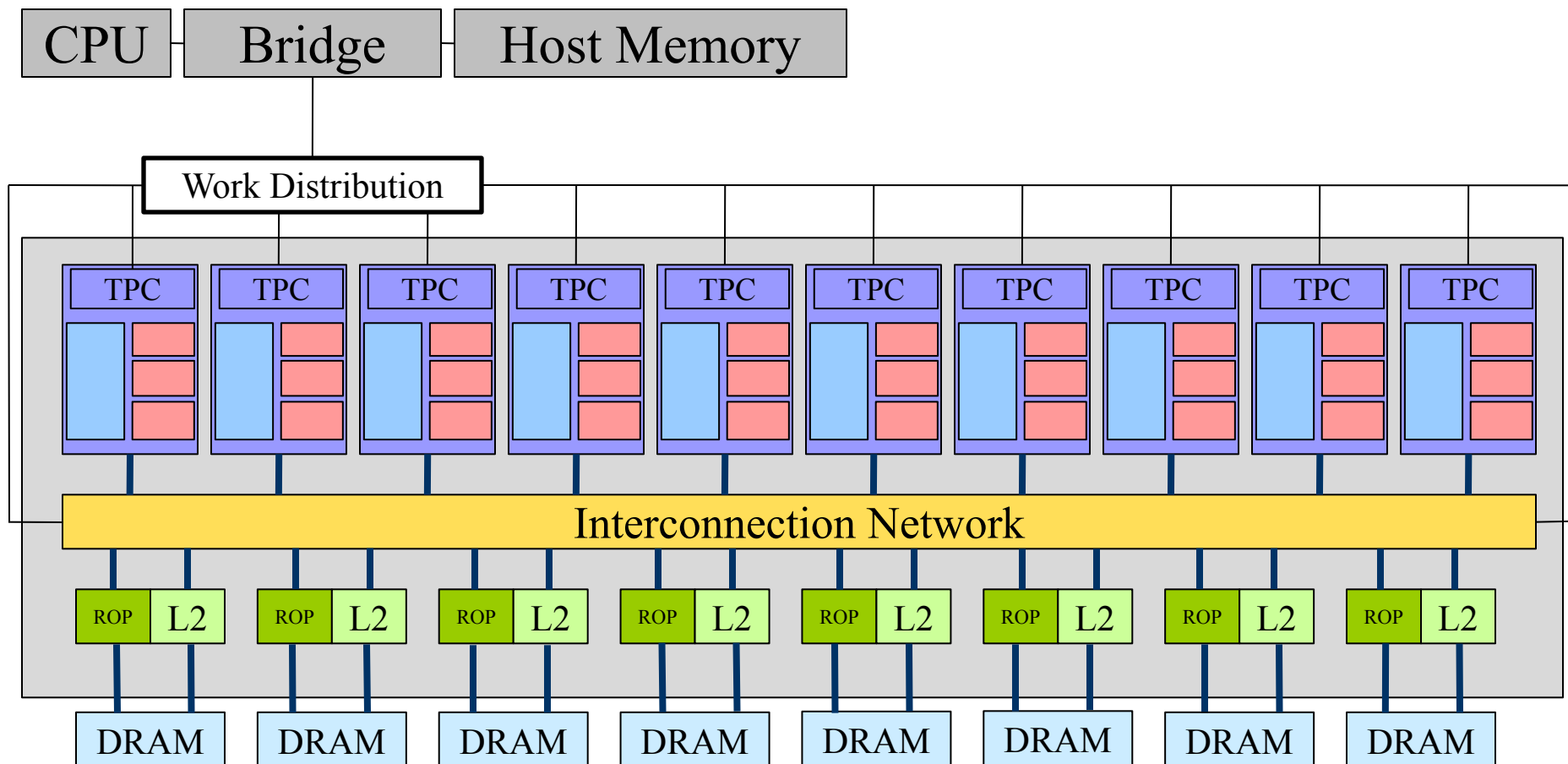


# Архитектура **Tesla**

## Мультипроцессор **Tesla 10**



# Архитектура **Tesla 10**





# Архитектура



- Масштабируемость:
  - [+] [-] SM внутри TPC
  - [+] [-] TPC
  - [+] [-] DRAM партиции
- Схожие архитектуры:
  - Tesla 8: 8800 GTX
  - Tesla 10: GTX 280

# Технические детали



- **RTM `CUDA Programming Guide`**
- **Run `CUDAHelloWorld`**
  - Печатает аппаратно зависимые параметры
    - Размер shared памяти
    - Кол-во SM
    - Размер warp'a
    - Кол-во регистров на SM
    - Т.д.

# План



- Архитектура Tesla
- Программная модель CUDA
- Синтаксические особенности CUDA

# Программная модель **CUDA**

- GPU (*device*) это вычислительное устройство, которое:
  - Является сопроцессором к CPU (*host*)
  - Имеет собственную память (DRAM)
  - Выполняет одновременно **очень много** нитей

# Программная модель **CUDA**

- Последовательные части кода выполняются на CPU
- Массивно-параллельные части кода выполняются на GPU как ядра
- Отличия нитей между CPU и GPU
  - Нити на GPU очень «легкие»
  - HW планировщик задач
  - Для полноценной загрузки GPU нужны тысячи нитей
    - Для покрытия латентностей операций чтения / записи
    - Для покрытия латентностей sfu инструкций

# Программная модель **CUDA**

- Параллельная часть кода выполняется как большое количество нитей
- Нити группируются в блоки фиксированного размера
- Блоки объединяются в сеть блоков
- Ядро выполняется на сетке из блоков
- Каждая нить и блок имеют свой идентификатор

# Программная модель **CUDA**

- Десятки тысяч потоков

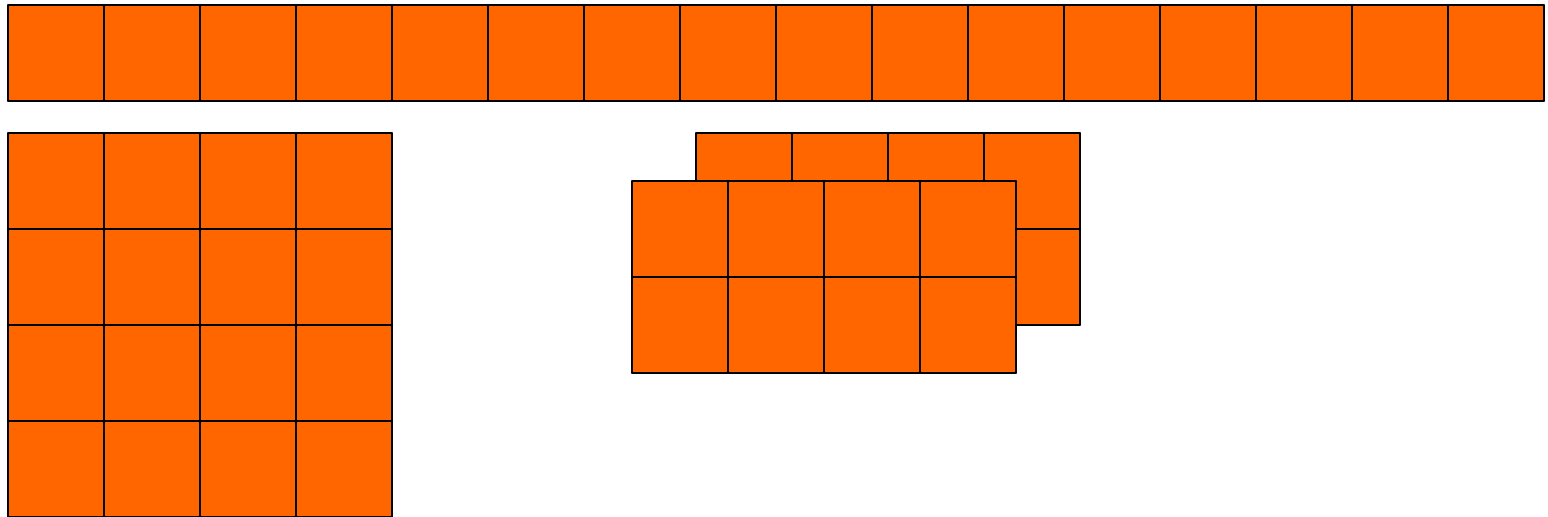
```
for (int ix = 0; ix < nx; ix++)  
{  
    pData[ix] = f(ix);  
}
```

```
for (int ix = 0; ix < nx; ix++)  
    for (int iy = 0; iy < ny; iy++)  
    {  
        pData[ix + iy * nx] = f(ix) * g(iy);  
    }
```

```
for (int ix = 0; ix < nx; ix++)  
    for (int iy = 0; iy < ny; iy++)  
        for (int iz = 0; iz < nz; iz++)  
        {  
            pData[ix + (iy + iz * ny) * nx] = f(ix) * g(iy) * h(iz);  
        }
```

# Программная модель **CUDA**

- Потоки в CUDA объединяются в блоки:
  - Возможна 1D, 2D, 3D топология блока
    - Общее кол-во потоков в блоке ограничено
    - В текущем HW это 512 потоков





# Программная модель **CUDA**

- Потоки в блоке могут разделять ресурсы со своими соседями

```
float g_Data[gN];  
for (int ix = 0; ix < nx; ix++)  
{  
    pData[ix] = f(ix, g_Data[ix / n]);  
}
```

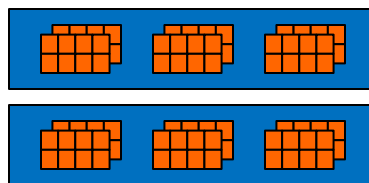
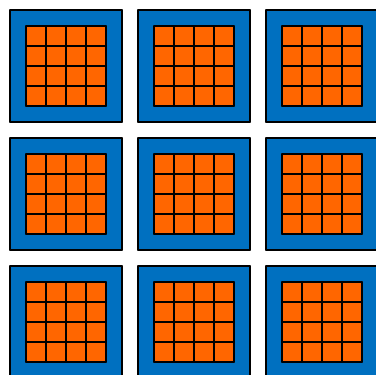
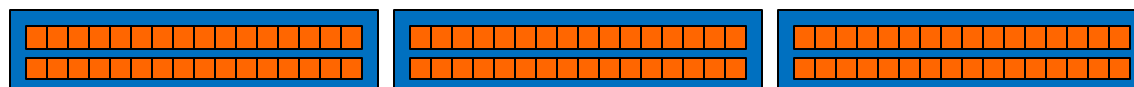
# Программная модель **CUDA**



- Блоки могут использовать shared память
  - Т.к. блок целиком выполняется на одном SM
  - Объем shared памяти ограничен и зависит от HW
- Внутри Блока потоки могут синхронизоваться
  - Т.к. блок целиком выполняется на одном SM

# Программная модель **CUDA**

- Блоки потоков объединяются в сетку (grid) потоков
  - Возможна 1D, 2D топология сетки блоков потоков



# План



- Архитектура Tesla
- Программная модель CUDA
- Синтаксические особенности CUDA

# Синтаксис **CUDA**



- CUDA – это расширение языка C
  - [+] спецификаторы для функций и переменных
  - [+] новые встроенные типы
  - [+] встроенные переменные (внутри ядра)
  - [+] директива для запуска ядра из C кода
- Как скомпилировать CUDA код
  - [+] nvcc компилятор
  - [+] .cu расширение файла

# Синтаксис **CUDA**

## Спецификаторы

- Спецификатор функций

Спецификатор	Выполняется на	Может вызываться из
<code>__device__</code>	device	device
<code>__global__</code>	device	host
<code>__host__</code>	host	host

- Спецификатор переменных

Спецификатор	Находится	Доступна	Вид доступа
<code>__device__</code>	device	device	R
<code>__constant__</code>	device	device / host	R / W
<code>__shared__</code>	device	block	RW / <b><code>__syncthreads()</code></b>

# Синтаксис **CUDA**

## Встроенные переменные

- Сравним CPU код vs CUDA kernel:

```
float * pData;  
for (int ix = 0; ix < nx; ix++)  
{  
    pData[ix] = pData[ix] + 1.0f;  
}
```

Пусть  $nx = 2048$   
Пусть в блоке 256  
ПОТОКОВ

□ КОЛ-ВО БЛОКОВ =  
 $2048 / 256 = 8$



```
__global__ void incKernel ( float * pData )  
{  
    [ 0 .. 7 ] [ == 256 ] [ 0 .. 255 ]  
    int idx = blockIdx.x * blockDim.x + threadIdx.x;  
    pData [idx] = pData [idx] + 1.0f;  
}
```

# Синтаксис **CUDA**

## Встроенные переменные

- В любом CUDA kernel'е доступны:

- `dim3` `gridDim;`
- `uint3` `blockIdx;`
- `dim3` `blockDim;`
- `uint3` `threadIdx;`
- `int` `warpSize;`

`dim3` – встроенный тип, который используется для задания размеров kernel'а  
По сути – это `uint3`.



# Синтаксис **CUDA**

## Директивы запуска ядра

- Как запустить ядро с общим кол-во тредов равным `nx`?

```
float * pData;
```

```
dim3 threads(256, 1, 1);      Неявно предполагаем,  
dim3 blocks(nx / 256, 1);    что nx кратно 256
```

```
incKernel<<<blocks, threads>>> ( pData );
```

<<< , >>> угловые скобки, внутри которых задаются параметры запуска ядра:

- Кол-во блоке в сетке
- Кол-во потоков в блоке
- ...

# Как скомпилировать **CUDA** код

- NVCC – компилятор для CUDA
  - Основными опциями команды **nvcc** являются:
  - **-deviceemu** - компиляция в режиме эмуляции, весь код будет выполняться в многопоточном режиме на CPU и можно использовать обычный отладчик (хотя не все ошибки могут проявиться в таком режиме)
  - **--use\_fast\_math** - заменить все вызовы стандартных математических функций на их быстрые (но менее точные) аналоги
  - **-o <outputFileName>** - задать имя выходного файла
- CUDA файлы обычно носят расширение **.cu**

# Ресурсы нашего курса



- [CUDA.CS.MSU.SU](https://cuda.cs.msu.su)

- Место для вопросов и дискуссий
- Место для материалов нашего курса
- Место для ваших статей!
  - Если вы нашли какой-то интересный подход!
  - Или исследовали производительность разных подходов и знаете, какой из них самый быстрый!
  - Или знаете способы сделать работу с CUDA проще!

- [Steps3d](https://steps3d.com)

- [www.nvidia.ru](https://www.nvidia.ru)

