Введение в параллельные вычисления

Лекция 12. NVIDIA CUDA (часть 2)

КС-40, КС-44
РХТУ

Преподаватель
Митричев Иван Игоревич, к.т.н.,
ассистент кафедры ИКТ

2017

Где мне найти Compute Capability для моей видеокарты?

Compute Capability (CC) = SM version
https://developer.nvidia.com/cuda-gpus

Соответствие первой цифры Compute Capability и архитектуры SM устройств

7 – *Volta*
6 - *Pascal*
5 - *Maxwell*
3 - *Kepler*
2 - *Fermi*
1 - *Tesla*

Блоки, SMы…

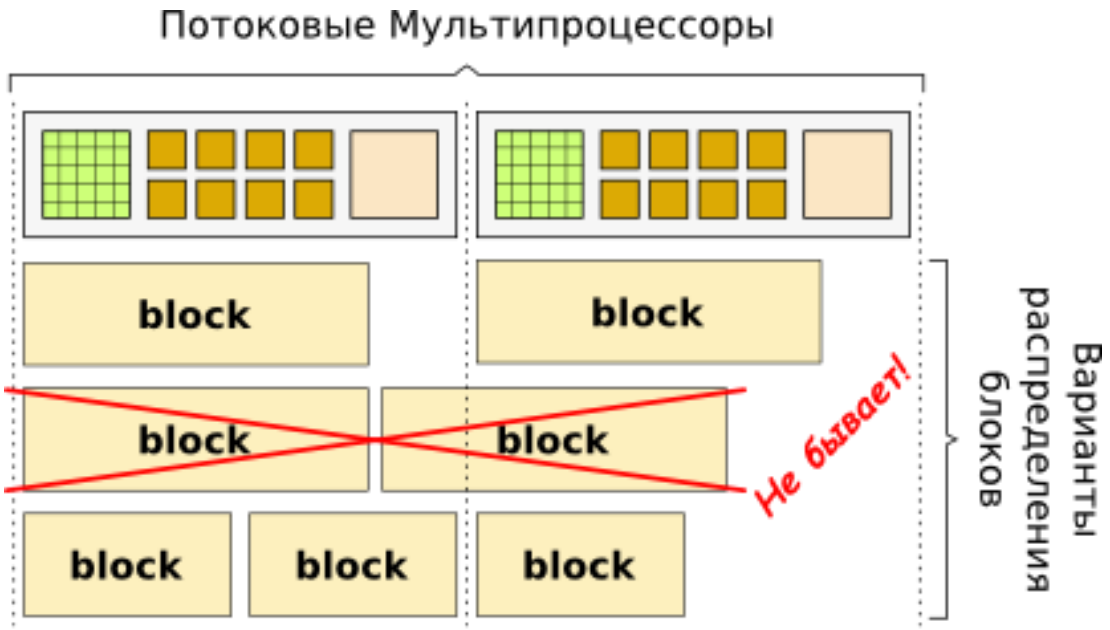


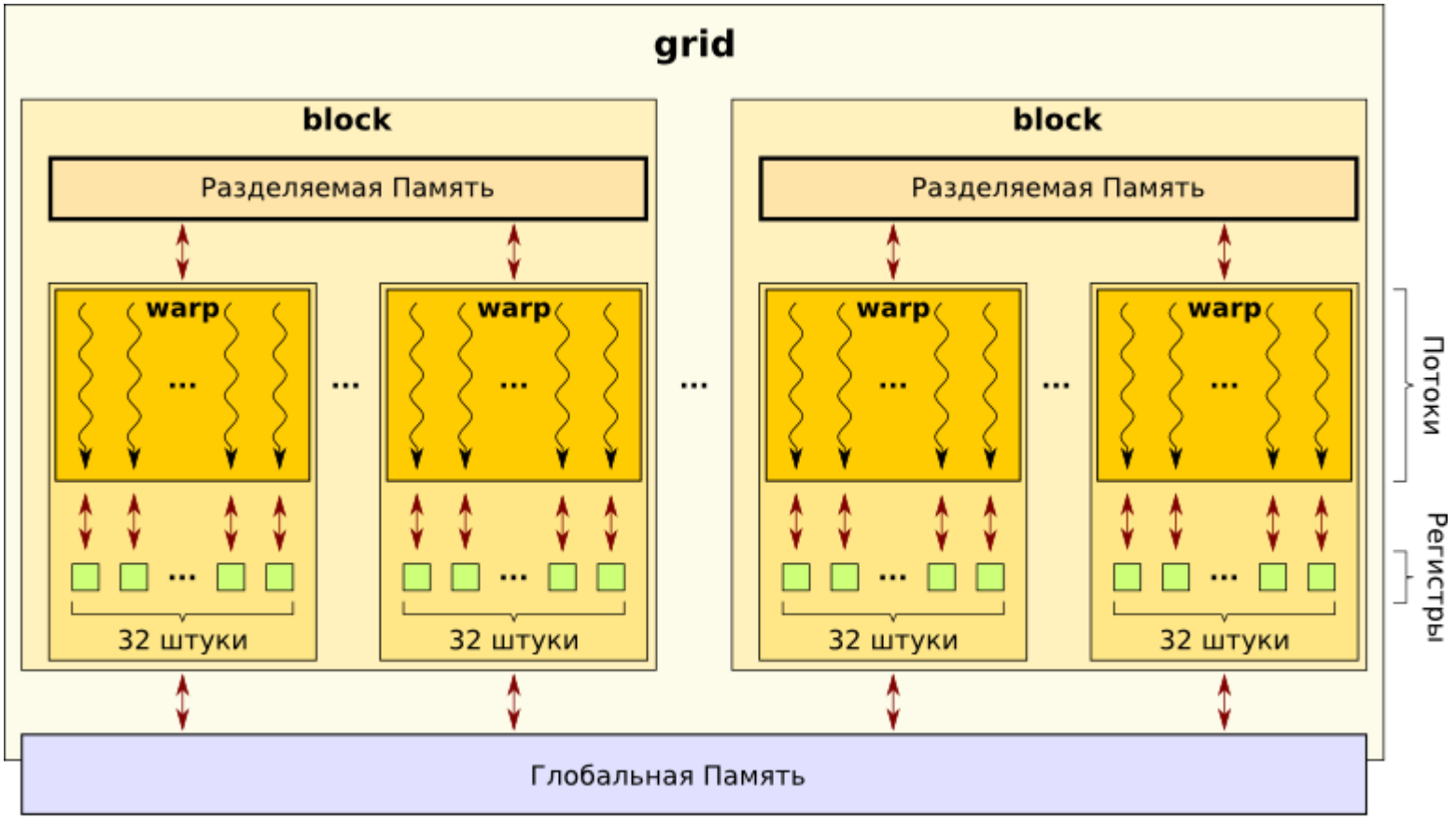
Рис. 3. Блоки и мультрипроцессоры (SM).

“Все потоки каждого блока запускаются строго на одном SM, поэтому имеют доступ только к его ресурсам. Однако, на одном SM может запускаться более одного блока (см. Рис.3), и ресурсы будут разделятся между ними поровну”

https://habrahabr.ru/post/151897/

Как осуществляется доступ в память?

В каждый момент времени 1 потоковый мультипроцессор (SM) (вспомним, на нем для исполнения находится определенное число блоков, но блок не может разделяться для исполнения по нескольким SM!) исполняет 1 warp (32 потока). Исполнение бОльшего числа потоков на SM осуществляется переключением контекста (аналогично в ОС организована мультизадачность) – т.е., последовательным выбором разных/одних и тех же warp.

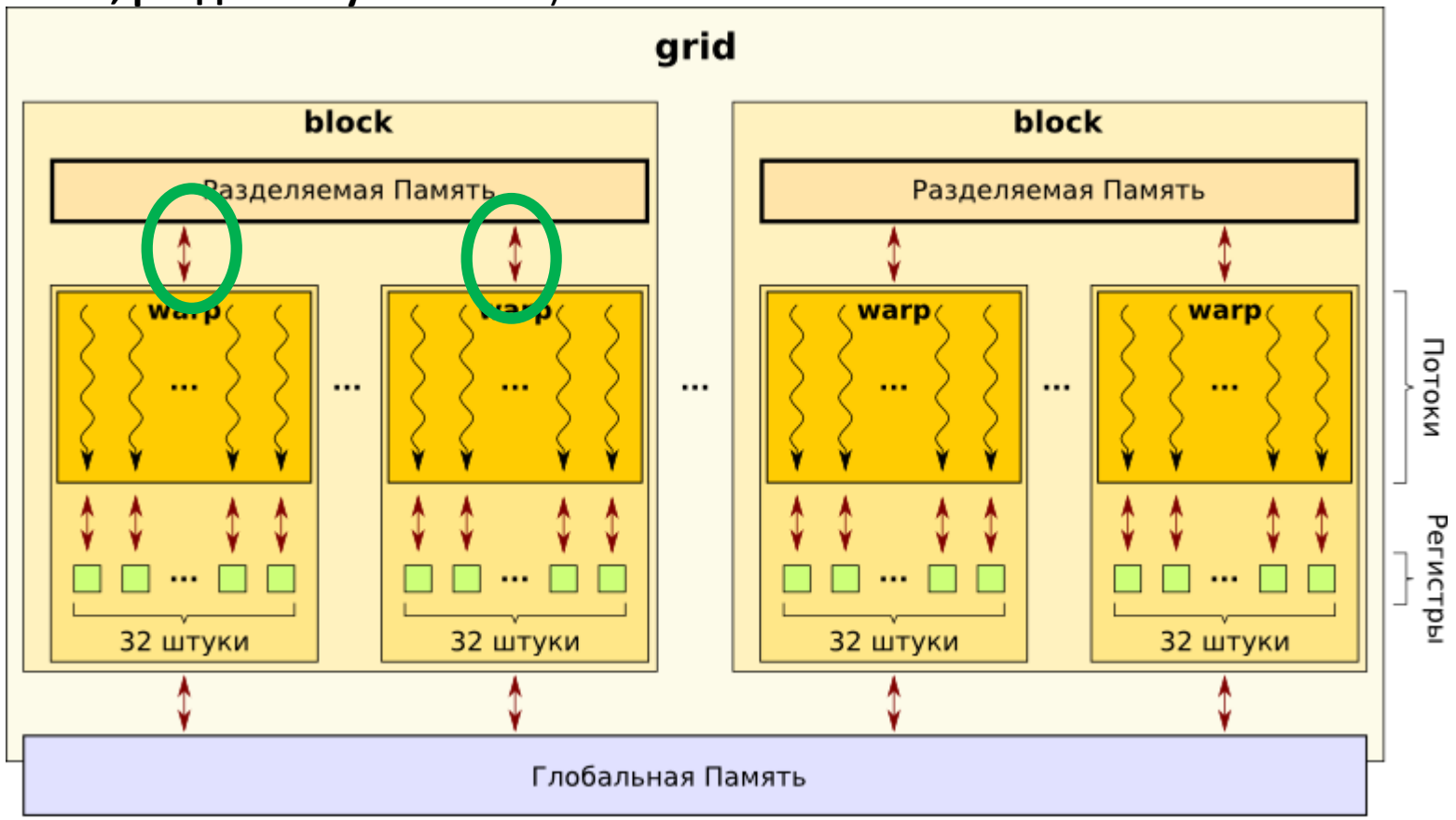


https://habrahabr.ru/post/151897/

Как осуществляется доступ в память?

Может случиться так, что некоторые warp уже закончили свою работу, а остальные – еще нет. Если следующее действие в вашем коде на устройстве предполагает, что предыдущая операция полностью завершена всеми потоками (например, первое действие сложение матриц C=A+B, второе D=C*E), то между ними необходимо синхронизация!!!

__syncthreads(); // -> на устройстве (синхронизируем все потока внутри **блока. Т.е., разделяемую память**)

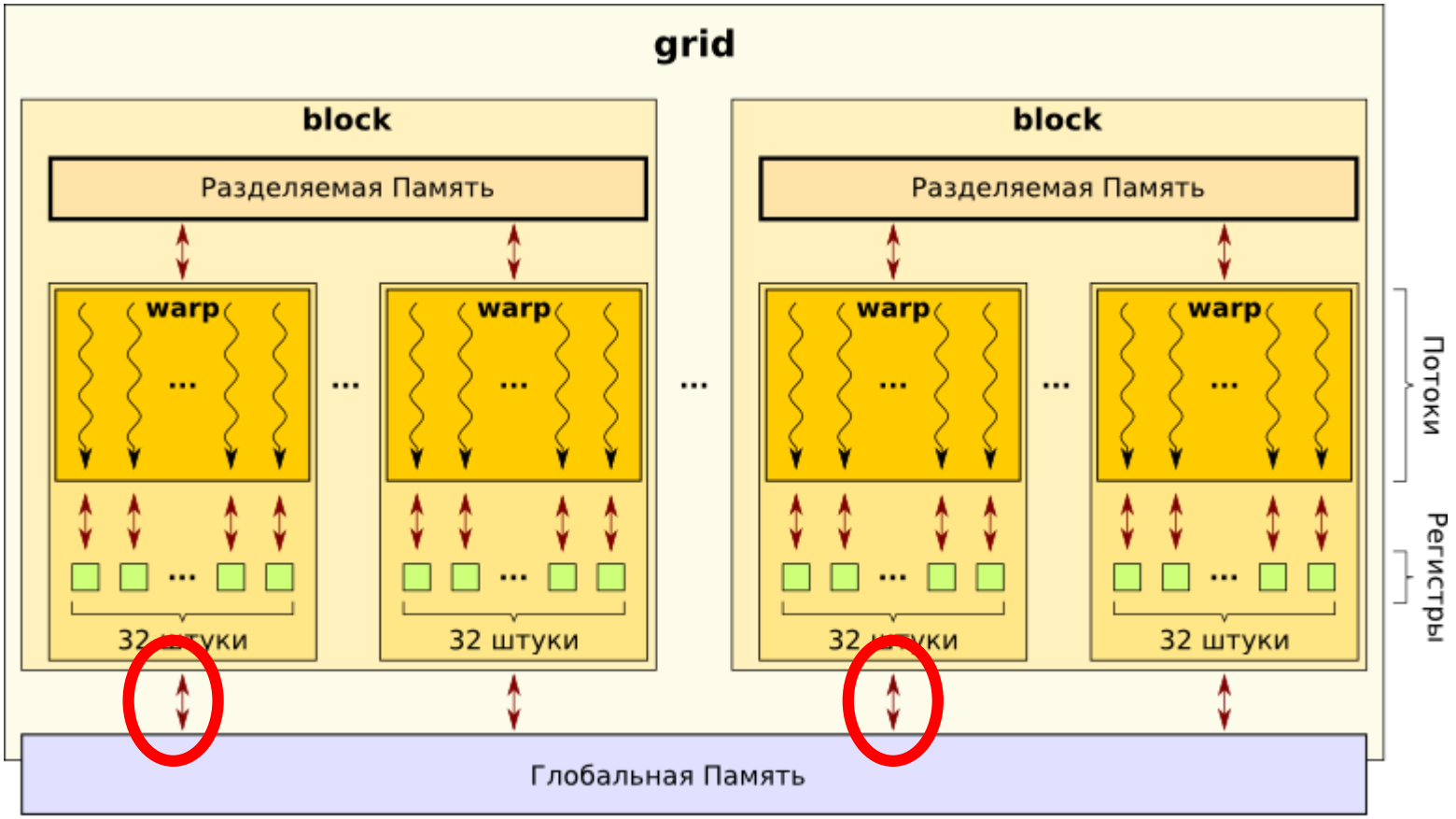


https://habrahabr.ru/post/151897/

Как осуществляется доступ в память?

__syncthreads(); // -> на устройстве (синхронизируем все потока внутри **блока**)

А как синхронизировать все блоки? (т.е., состояние глобальной памяти)



https://habrahabr.ru/post/151897/

Пример: ошибочный доступ к глобальной памяти

Рис. 4. Схема тестового алгоритма (для 128 потоков).

Пусть в глобальной памяти адаптера выделены два участка: под массивы **X**[] и **P**[] по 128 элементов. Пусть массив **X**[] записывается с хоста (центральным процессором из оперативной памяти компьютера). Создадим grid из двух блоков по 64 потока в каждом — то есть всего 128 потоков (см. Рис.4).

два блока

Тест:

Этап (**i**): каждый поток с номером *j* будет складывать между собой все элементы массива **X**[], записывая результат в **P**[j].

Этап (**ii**): каждый *j*-ый поток начнет суммирование всех элементов массива **P**[], записывая их в соответствующие **X**[j].

https://habrahabr.ru/post/151897/

Пример: ошибочный доступ к глобальной памяти (среднее арифметическое элементов)

```
__global__ void Kernel(float *X, float *P)
{
const int N = 128; // Число элементов и используемых потоков в константе.
const int index = threadIdx.x + blockIdx.x*blockDim.x; // Номер потока.
float a; // Аккумулятор в регистре. Каждому потоку свой.
/* этап (i): */
a = X[0];
for(int j = 1; j < N; ++j) // Собственно, цикл суммирования
    a += X[j];
P[index] = a / N; // Отнормируем, чтобы не получалось больших чисел.
/* конец этапа (i). */
/* этап (ii): */
a = P[0];
for(int j = 1; j < N; ++j) // Собственно, цикл суммирования
    a += P[j];
X[index] = a / N; // Отнормируем, чтобы не получалось больших чисел.
/* конец этапа (ii). */
}  // Результат: в массиве X периодически части элементов по 32 штуки неверно рассчитаны! (не все элементы равны). warp!
```

https://habrahabr.ru/post/151897/

Пример: ошибочный доступ к глобальной памяти

Следующий код может давать ошибочный результат, но уже не в пределах warp, а сразу целого блока:

```
__global__ void Kernel(float *X, float *P) {
 ...
 /* конец этапа (i). */
__syncthreads();
/* этап (ii): */
... }
```

https://habrahabr.ru/post/151897/

Пути синхронизации данных между блоками

1) Разделить код на две kernel-функции
2) Придумать собственный механизм синхронизации – объявить атомарную/глобальную переменную, которая бы хранила число потоков, который завершили первый этап.

```
__device__ int blockcounter;
….
__syncthreads();
atomicAdd(&blockcounter,1);
….
```

В CUDA блоки могут выполняться даже последовательно (нет никаких правил относительно их одновременного выполнения). Поэтому, любые глобальные барьеры типа 1 или 2 скажутся на производительности. 1 – оптимальный вариант.

3) CC 6.x и выше – кооперативные группы потоков.

```
grid_group g = this_grid();
g.sync();
```

https://habrahabr.ru/post/151897/
https://devtalk.nvidia.com/default/topic/526338/synchronize-all-blocks-in-cuda/
https://stackoverflow.com/questions/6404992/cuda-block-synchronization