CUDA. Лимитирующие факторы

Ахтямов Павел

МФТИ

Как замерить скорость работы?

Как замерить скорость работы?

- `cudaEvent_t` - событие времени

Как замерить скорость работы?

- `cudaEvent_t` - событие времени
- `cudaEventCreate` - создание события

Как замерить скорость работы?

- `cudaEvent_t` - событие времени
- `cudaEventCreate` - создание события
- `cudaEventRecord` - записываем событие

Как замерить скорость работы?

- `cudaEvent_t` - событие времени
- `cudaEventCreate` - создание события
- `cudaEventRecord` - записываем событие
- `cudaEventSynchronize` - ожидаем исполнение события

Как замерить скорость работы?

- `cudaEvent_t` - событие времени
- `cudaEventCreate` - создание события
- `cudaEventRecord` - записываем событие
- `cudaEventSynchronize` - ожидаем исполнение события
- Зачем ждать событие???

Зачем ждать событие?

Зачем ждать событие?

- Ядро (**kernel**) исполняется **асинхронно**!

Зачем ждать событие?

- Ядро (**kernel**) исполняется **асинхронно**!
- Замер времени покажет 0!

Зачем ждать событие?

- Ядро (**kernel**) исполняется **асинхронно**!
- Замер времени покажет 0!
- Поставили “трекер“ прохождения дистанции!

Зачем ждать событие?

- Ядро (**kernel**) исполняется **асинхронно**!
- Замер времени покажет 0!
- Поставили “трекер“ прохождения дистанции!
- На финише синхронизируем время!

Пример

```
40      cudaEvent_t start;
41      cudaEvent_t stop;
42
43      // Creating event
44      cudaEventCreate(&start);
45      cudaEventCreate(&stop);
46
47
48      cudaEventRecord(start);
49      add<<<numBlocks, blockSize>>>(N, d_x, d_y);
50
51      cudaEventRecord(stop);
52
53      cudaMemcpy(y, d_y, size, cudaMemcpyDeviceToHost);
54
55      cudaEventSynchronize(stop);
56
57      float milliseconds = 0;
58
59      cudaEventElapsedTime(&milliseconds, start, stop);
60
61      std::cout << milliseconds << " elapsed" << std::endl;
```

Посчитаем количество операций!

Посчитаем количество операций!

- RTX 2080 Ti: 4352 ядра

Посчитаем количество операций!

- RTX 2080 Ti: 4352 ядра
- Частота ядра: 1545 МГц

Посчитаем количество операций!

- RTX 2080 Ti: 4352 ядра
- Частота ядра: 1545 МГц
- 2 операции за цикл

Посчитаем количество операций!

- RTX 2080 Ti: 4352 ядра
- Частота ядра: 1545 МГц
- 2 операции за цикл
- $4352 \times 1.545 \times 2 = 13,5\,TFLOPs$

Посчитаем количество операций!

- RTX 2080 Ti: 4352 ядра
- Частота ядра: 1545 МГц
- 2 операции за цикл
- $4352 \times 1.545 \times 2 = 13,5\,TFLOPs$
- Чем приходится платить?

А что с памятью?

А что с памятью?

- Посчитаем пропускную способность

А что с памятью?

- Посчитаем пропускную способность
- $BW = Freq_{RAM} \times Bit \times Efficiency$

А что с памятью?

- Посчитаем пропускную способность
- $BW = Freq_{RAM} \times Bit \times Efficiency$
- $Freq_{RAM} = 1750$МГц

А что с памятью?

- Посчитаем пропускную способность
- $BW = Freq_{RAM} \times Bit \times Efficiency$
- $Freq_{RAM} = 1750$МГц
- $Bit = 352$

А что с памятью?

- Посчитаем пропускную способность
- $BW = Freq_{RAM} \times Bit \times Efficiency$
- $Freq_{RAM} = 1750$МГц
- $Bit = 352$
- $Efficiency = 8$:

А что с памятью?

- Посчитаем пропускную способность
- $BW = Freq_{RAM} \times Bit \times Efficiency$
- $Freq_{RAM} = 1750$МГц
- $Bit = 352$
- $Efficiency = 8$:
- GDDR6 - 4 передачи за цикл

А что с памятью?

- Посчитаем пропускную способность
- $BW = Freq_{RAM} \times Bit \times Efficiency$
- $Freq_{RAM} = 1750$МГц
- $Bit = 352$
- $Efficiency = 8$:
- GDDR6 - 4 передачи за цикл
- особенности DDR - 2 передачи

А что с памятью?

- Посчитаем пропускную способность
- $BW = Freq_{RAM} \times Bit \times Efficiency$
- $Freq_{RAM} = 1750$МГц
- $Bit = 352$
- $Efficiency = 8$:
- GDDR6 - 4 передачи за цикл
- особенности DDR - 2 передачи
- Итого: 616 ГБ/с

Эффективная пропускная способность

Эффективная пропускная способность

- $BW_{eff} = \frac{Bytes}{t}$

Эффективная пропускная способность

- $BW_{eff} = \frac{Bytes}{t}$
- *Bytes* - необходимо посчитать все операции чтения и записи

Пример: SAXPY

Пример: SAXPY

- $y = ax + y$

Пример: SAXPY

- $y = ax + y$
- Читаем x

Пример: SAXPY

- $y = ax + y$
- Читаем x
- Читаем y

Пример: SAXPY

- $y = ax + y$
- Читаем x
- Читаем y
- Пишем y

Пример: SAXPY

- $y = ax + y$
- Читаем x
- Читаем y
- Пишем y
- Получаем: $3N$ операций

Пример: SAXPY

- $y = ax + y$
- Читаем x
- Читаем y
- Пишем y
- Получаем: $3N$ операций
- Всего: $12N$ байт (float)

Пример: SAXPY

- $y = ax + y$
- Читаем x
- Читаем y
- Пишем y
- Получаем: $3N$ операций
- Всего: $12N$ байт (float)
- 2^{28} элементов: 5.77×10^{-3} с

Пример: SAXPY

- $y = ax + y$
- Читаем x
- Читаем y
- Пишем y
- Получаем: $3N$ операций
- Всего: $12N$ байт (float)
- 2^{28} элементов: 5.77×10^{-3} с
- $BW = \frac{2^{28} \cdot 12}{5.77 \cdot 10^{-3}} \approx 557$ ГБ/с

Пример: SAXPY

- $y = ax + y$
- Читаем x
- Читаем y
- Пишем y
- Получаем: $3N$ операций
- Всего: $12N$ байт (float)
- 2^{28} элементов: 5.77×10^{-3} с
- $BW = \frac{2^{28} \cdot 12}{5.77 \cdot 10^{-3}} \approx 557$ ГБ/с
- $\eta = \frac{557}{616} \approx 90,5\%$

Как можно ускорить?

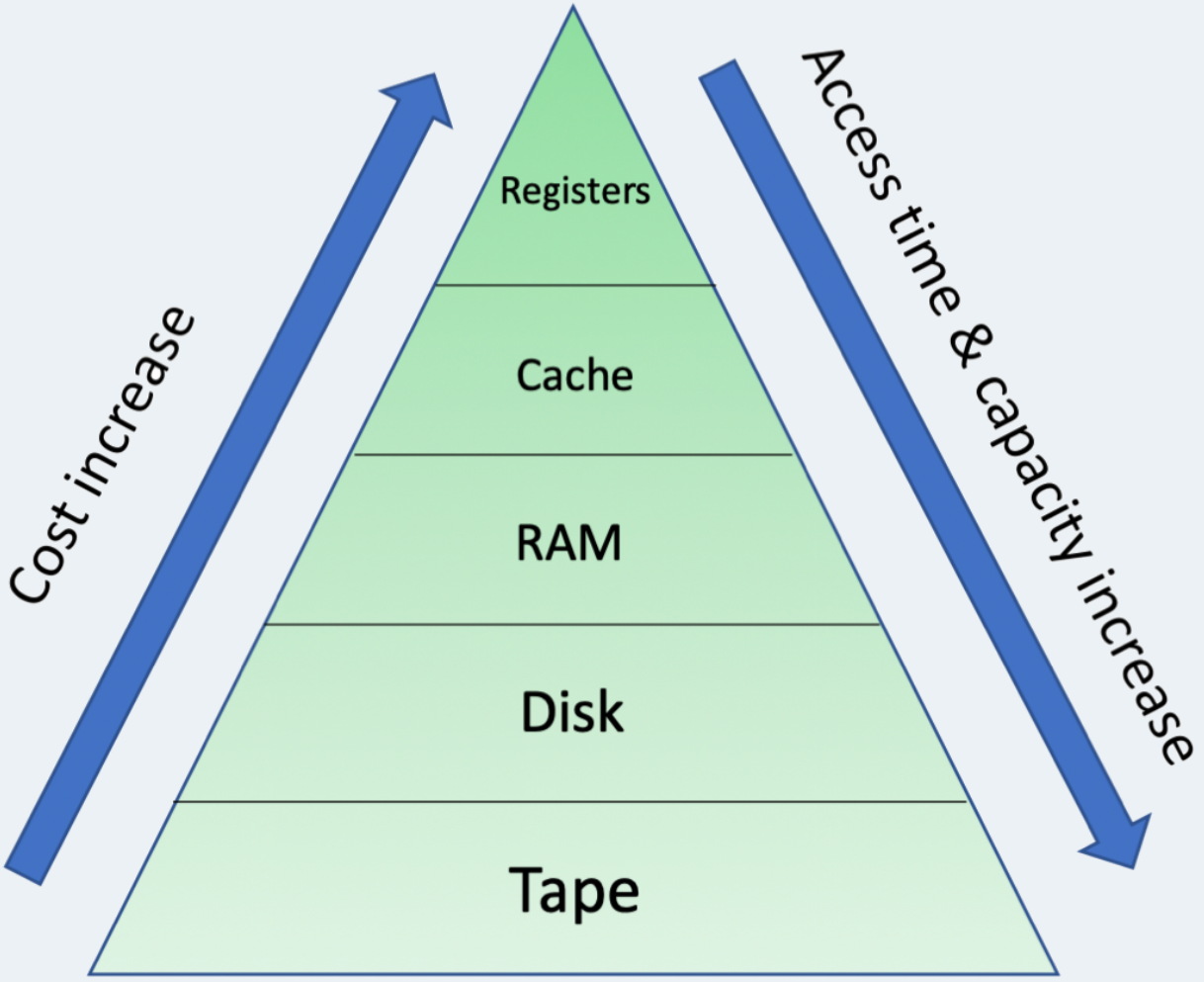
Как можно ускорить?

- Заблокировать память на Host-е

Как можно ускорить?

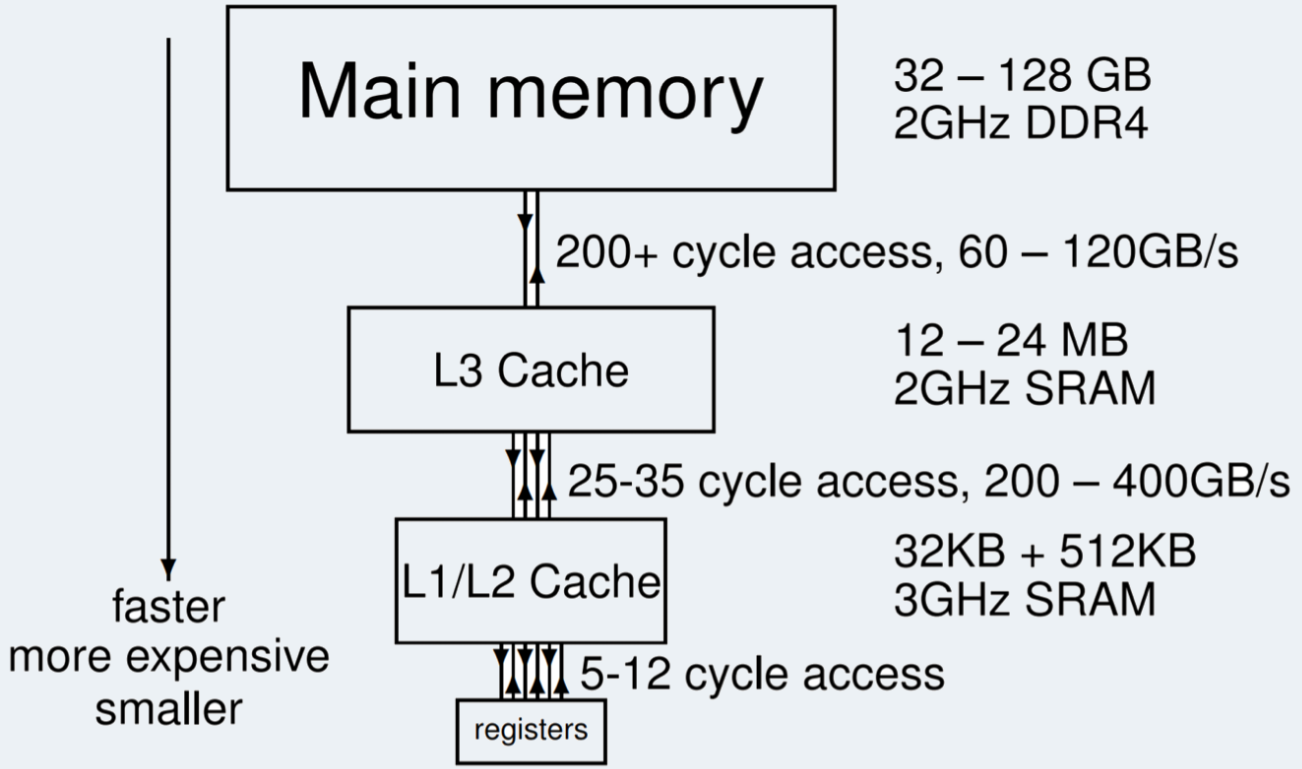
- Заблокировать память на Host-е
- Функция *cudaHostAlloc*

Иерархия



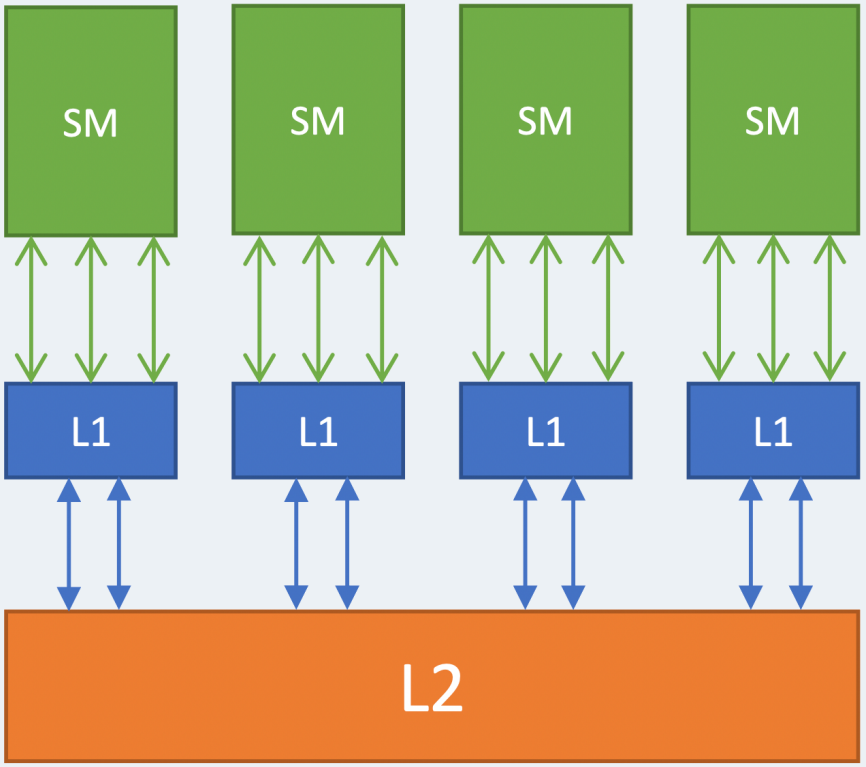
people.maths.ox.ac.uk

Иерархия CPU



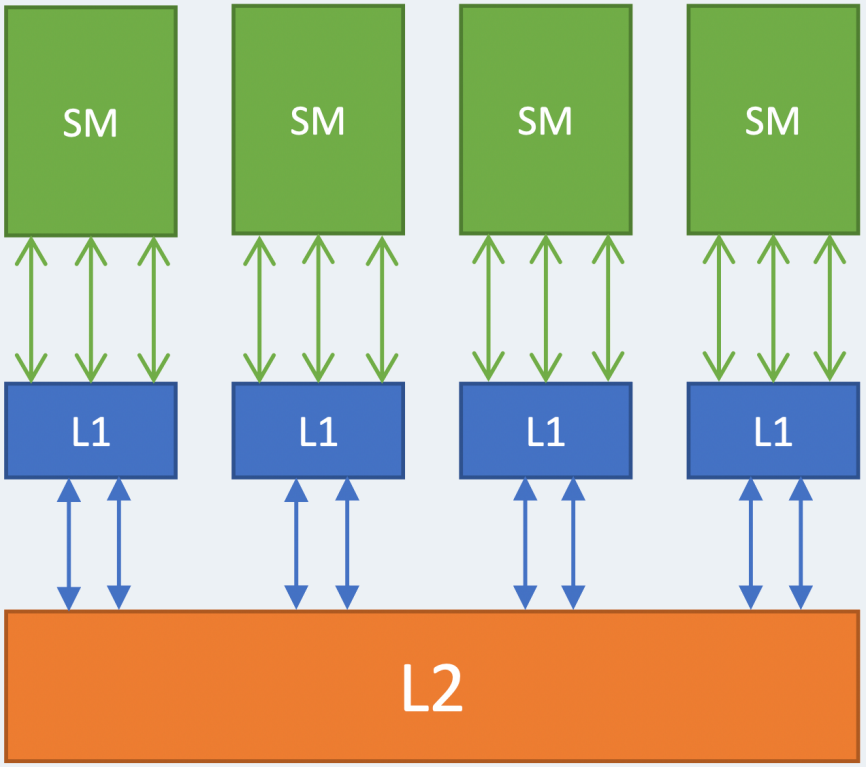
people.maths.ox.ac.uk

Иерархия GPU



people.maths.ox.ac.uk

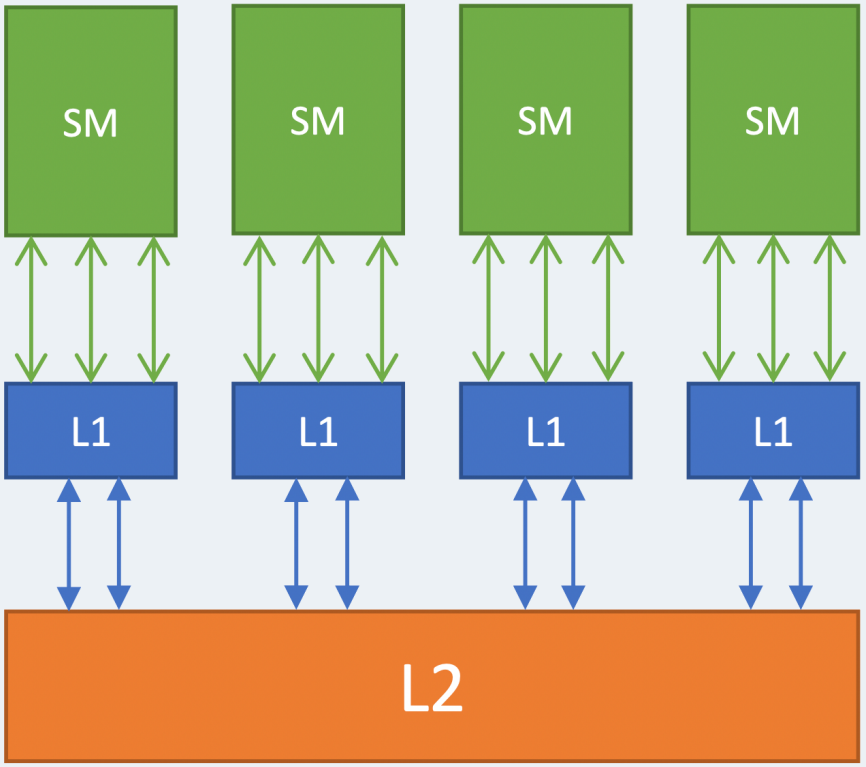
Иерархия GPU



people.maths.ox.ac.uk

- Ширина кеш-линии: 128 байт!

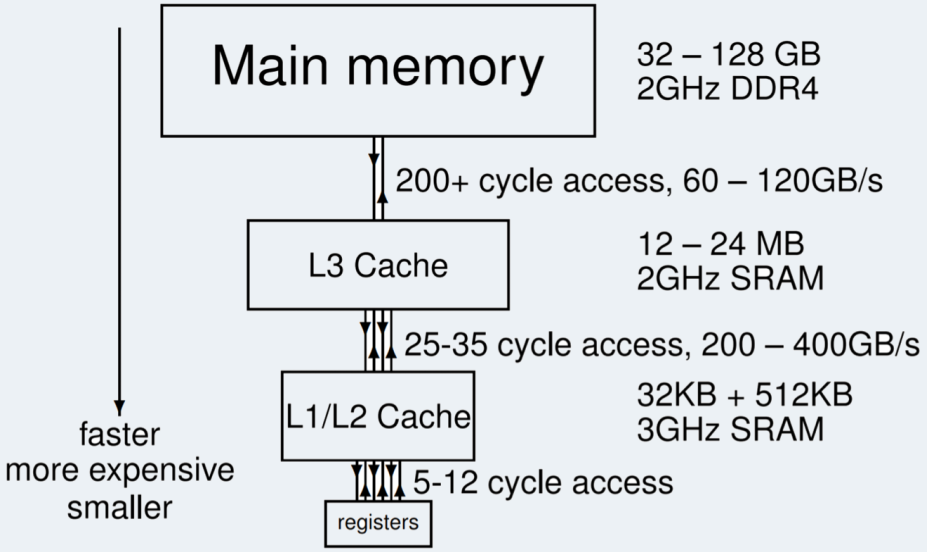
Иерархия GPU



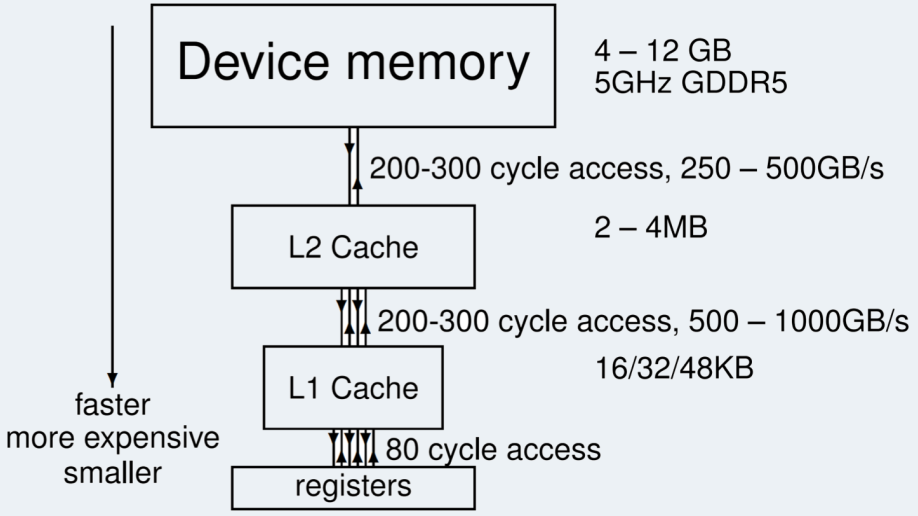
people.maths.ox.ac.uk

- Ширина кеш-линии: 128 байт!
- Вмещаются 32 float-а!

Сравнение

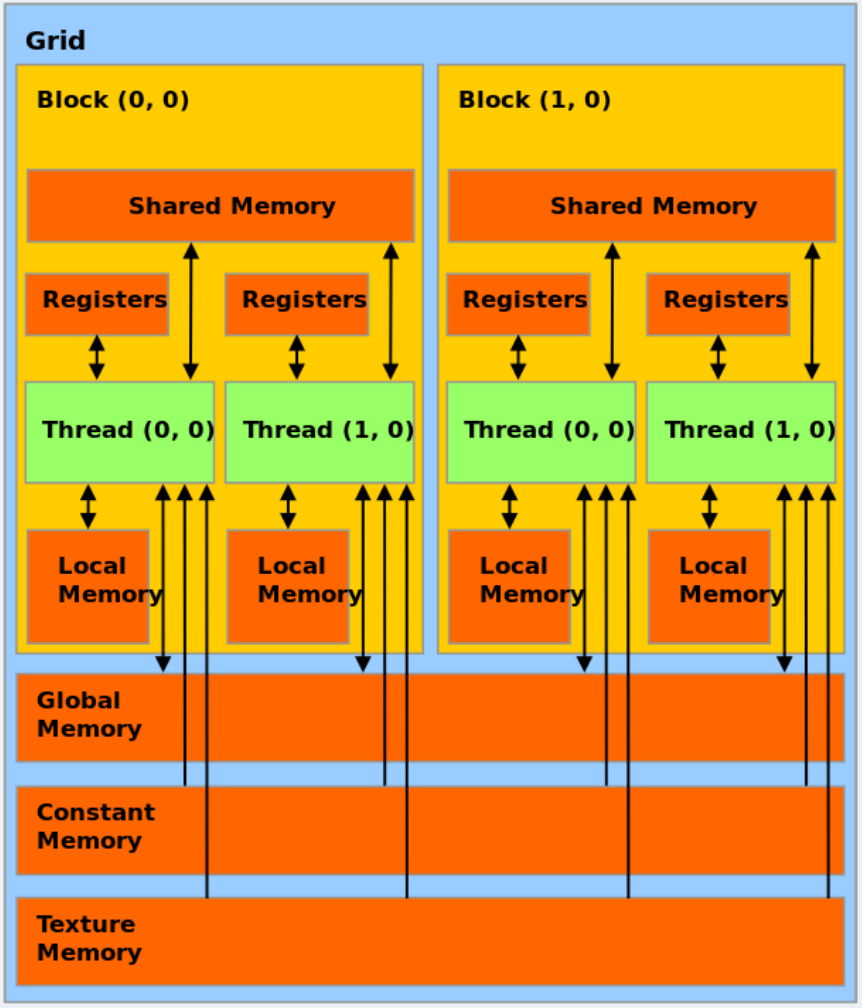


CPU



GPU

Типы памяти



nvidia.com

Разделяемая память

Разделяемая память

- Уникальная особенность видеокарты - использование на уровне кода L1 кеша.

Разделяемая память

- Уникальная особенность видеокарты - использование на уровне кода L1 кеша.
- Ключевое слово - `__shared__`

Разделяемая память

- Уникальная особенность видеокарты - использование на уровне кода L1 кеша.
- Ключевое слово - `__shared__`
- Данные при этом распространяются между всеми потоками в одном блоке!

Разделяемая память

- Уникальная особенность видеокарты - использование на уровне кода L1 кеша.
- Ключевое слово - `__shared__`
- Данные при этом распространяются между всеми потоками в одном блоке!
- Появляются локальные массивы с быстрой скоростью доступа!! (несколько TB/s)

Разделяемая память

- Уникальная особенность видеокарты - использование на уровне кода L1 кеша.
- Ключевое слово - `__shared__`
- Данные при этом распространяются между всеми потоками в одном блоке!
- Появляются локальные массивы с быстрой скоростью доступа!! (несколько TB/s)
- Правда, удовольствие ограниченное: 48KB на один блок!

Как определить размер shared-памяти?

Как определить размер shared-памяти?

- Статически - указать размер массива

Как определить размер shared-памяти?

- Статически - указать размер массива
- Динамически - делаем `extern __shared__`

Как определить размер shared-памяти?

- Статически - указать размер массива
- Динамически - делаем `extern __shared__`
- Передаём третьим параметром в вызов ядра количество байт в shared-памяти

Как определить размер shared-памяти?

- Статически - указать размер массива
- Динамически - делаем `extern __shared__`
- Передаём третьим параметром в вызов ядра количество байт в shared-памяти

```
< < < numBlocks, blockSize, shmem_per_block > > >
```

Синхронизация

```
#define BLOCKDIMX 128

__shared__ ut[BLOCKDIMX];

__global__ void lap(float *u) {

          int tid = threadIdx.x + blockIdx.x*blockDim.x;

          ut[threadIdx.x] = u[tid];
          __syncthreads();

          if(threadIdx.x < BLOCKDIMX - 32)
          u[tid] = (ut[threadIdx.x]-ut[threadIdx.x + 32]);

}
```

people.maths.ox.ac.uk

Доступ к данным

Доступ к данным

- Размер блока - 256

Доступ к данным

- Размер блока - 256
- Хотим потоком обрабатывать 2 элемента

Доступ к данным

- Размер блока - 256
- Хотим потоком обрабатывать 2 элемента
- **Способ 1**

Доступ к данным

- Размер блока - 256
- Хотим потоком обрабатывать 2 элемента
- **Способ 1** : обрабатывать $(0, 1)$, $(2, 3)$, ...

Доступ к данным

- Размер блока - 256
- Хотим потоком обрабатывать 2 элемента
- **Способ 1** : обрабатывать $(0, 1)$, $(2, 3)$, ...
- **Способ 2**

Доступ к данным

- Размер блока - 256
- Хотим потоком обрабатывать 2 элемента
- **Способ 1** : обрабатывать $(0, 1)$, $(2, 3)$, ...
- **Способ 2** : обрабатывать $(0, 256)$, $(1, 257)$, ...

Пример кода

```
 5 #define ILP 8
 6
 7 __global__
 8 void add(int n, float* x, float* y, float* z) {
 9     int tid = threadIdx.x + ILP * blockDim.x * blockIdx.x;
10     for (int i = 0; i < ILP; ++i) {
11         int current_tid = tid + i * blockDim.x;
12
13         z[current_tid] = 2.0f * x[current_tid] + y[current_tid];
14     }
15 }
16
17 __global__
18 void stupid_add(int n, float* x, float* y, float* z) {
19     int index = blockIdx.x * blockDim.x + threadIdx.x;
20     int actual_tid = ILP * index;
21     for (int i = 0; i < ILP; ++i) {
22         int current_tid = actual_tid + i;
23         z[current_tid] = 2.0f * x[current_tid] + y[current_tid];
24     }
25 }
```

Результаты

Результаты

- Способ 1 - 18.6 с

Результаты

- Способ 1 - 18.6 с
- Способ 2 - 5.8 с

Результаты

- Способ 1 - 18.6 с
- Способ 2 - 5.8 с
- $(0, 256)$, $(1, 257)$ - быстрее!

Результаты

- Способ 1 - 18.6 с
- Способ 2 - 5.8 с
- $(0, 256)$, $(1, 257)$ - быстрее!
- Пропуск, равный размеру блока!

Объединённый доступ

people.maths.ox.ac.uk

Объединённый доступ

people.maths.ox.ac.uk

- Чтобы не сломать кеш-линию, внутри warp-а должен быть последовательный доступ!

Объединённый доступ

people.maths.ox.ac.uk

- Чтобы не сломать кеш-линию, внутри warp-а должен быть последовательный доступ!
- Чем больше непоследовательных доступов, тем больше время работы!

Объединённый доступ

people.maths.ox.ac.uk

- Чтобы не сломать кеш-линию, внутри warp-а должен быть последовательный доступ!
- Чем больше непоследовательных доступов, тем больше время работы!

Поток управления

В видеокарте - **предсказательное вычисление**:

Поток управления

В видеокарте - **предсказательное вычисление**:

- вычисляется ветка с `if` и с `else`

Поток управления

В видеокарте - **предсказательное вычисление**:

- вычисляется ветка с `if` и с `else`
- вычисляется значение в условии на warp-е

Поток управления

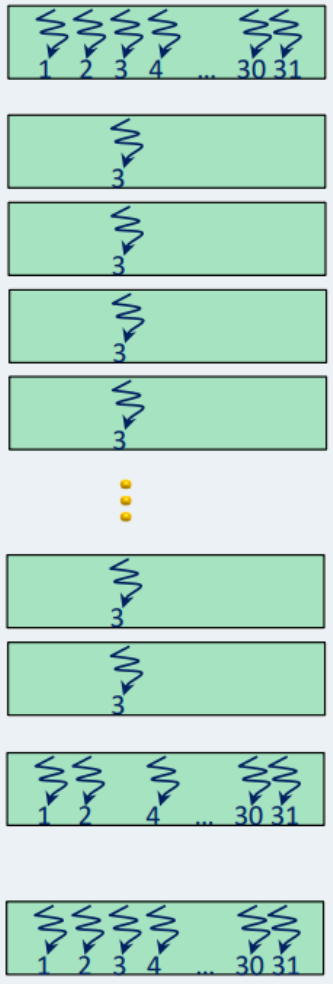
В видеокарте - **предсказательное вычисление**:

- вычисляется ветка с `if` и с `else`
- вычисляется значение в условии на warp-е
- выполняется conditional jump

Простаивание warp-а

```
array[threadIdx.x] = 0.0;

if(threadIdx.x == 3) {
  for(i=0; i < 1000; i++) {
    array[threadIdx.x]++;
  }
} else {
  array[threadIdx.x] = 3.14;
}
```



people.maths.ox.ac.uk

Простаивание warp-а

- Выполняется только поток 3!

people.maths.ox.ac.uk

Простаивание warp-а

people.maths.ox.ac.uk

- Выполняется только поток 3!
- Только для отладки и синхронизации!

Особенности синхронизации!

Особенности синхронизации!

- Нельзя использовать счётчик внутри блока!

Особенности синхронизации!

- Нельзя использовать счётчик внутри блока!
- Нужны атомарные операции:

Особенности синхронизации!

- Нельзя использовать счётчик внутри блока!
- Нужны атомарные операции:
  - `atomicAdd(&count, 1)`

Особенности синхронизации!

- Нельзя использовать счётчик внутри блока!
- Нужны атомарные операции:
  - `atomicAdd(&count, 1)`
  - `T out = atomicExch(T* address, T val)`

Синхронизация между блоками?

Синхронизация между блоками?

- Берём блокировку

Синхронизация между блоками?

- Берём блокировку
- `__device__ int lock = 0`

Синхронизация между блоками?

- Берём блокировку
- `__device__ int lock = 0`
- `do {} while (atomicCas(&lock, 0, 1));`

Синхронизация между блоками?

- Берём блокировку
- `__device__ int lock = 0`
- `do {} while (atomicCas(&lock, 0, 1));`
- `__threadfence();`

Синхронизация между блоками?

- Берём блокировку
- `__device__ int lock = 0`
- `do {} while (atomicCas(&lock, 0, 1));`
- `__threadfence();`
- shared - внутри блока

Синхронизация между блоками?

- Берём блокировку
- `__device__ int lock = 0`
- `do {} while (atomicCas(&lock, 0, 1));`
- `__threadfence();`
- shared - внутри блока
- device - глобально!

Синхронизация между блоками?

- Берём блокировку
- `__device__ int lock = 0`
- `do {} while (atomicCas(&lock, 0, 1));`
- `__threadfence();`
- shared - внутри блока
- device - глобально!
- `lock = 0;`

Информация по регистрам

Информация по регистрам

- 2^{16} регистров на блок

Информация по регистрам

- 2^{16} регистров на блок
- До 255 регистров на поток

Информация по регистрам

- 2^{16} регистров на блок
- До 255 регистров на поток
- Максимальное число потоков на SM - 1024

Информация по регистрам

- 2^{16} регистров на блок
- До 255 регистров на поток
- Максимальное число потоков на SM - 1024
- Размер блока 1024 - регистров на поток $65536/1024 = 64$

Информация по регистрам

- 2^{16} регистров на блок
- До 255 регистров на поток
- Максимальное число потоков на SM - 1024
- Размер блока 1024 - регистров на поток $65536/1024 = 64$
- Размер блока 32 - регистров $65536/32 = 2048 \rightarrow 255$

Информация по регистрам

- 2^{16} регистров на блок
- До 255 регистров на поток
- Максимальное число потоков на SM - 1024
- Размер блока 1024 - регистров на поток $65536/1024 = 64$
- Размер блока 32 - регистров $65536/32 = 2048 \rightarrow 255$
- Необходим компромисс!