Память

RAM память

Тип

Объем

Частота работы

Тайминг

Пропускная способность

DDR vs GDDR

- Graphics Double Data Rate
- Более высокая частота работы
- Более низкое энергопотребление
- Специальное управление буфером ввода/вывода

Отличие от CPU

- В CPU значимую часть занимают кэши различных уровней

- В CPU обычно не предоставляется прямой доступ к управлению кэшами

Глобальная память

- расположена на плате GPU
- может быть выделена с CPU (CUDA_API) или с нитей
- доступ для чтения / записи для всех нитей
- высокая латентность
- кэшируется начиная с Fermi

Регистровая память

- регистры распределяются между блоками на этапе компиляции
- у каждой нити монопольный доступ к нескольким регистрам на все время исполнения ядра
- доступ к регистрам других нитей запрещен
- минимальная латентность

Локальная память

- размещена в DRAM
- время доступа порядка 400-800 тактов
- хранит union-ы, массивы размеры которых неизвестны в момент компиляции

Разделяемая память

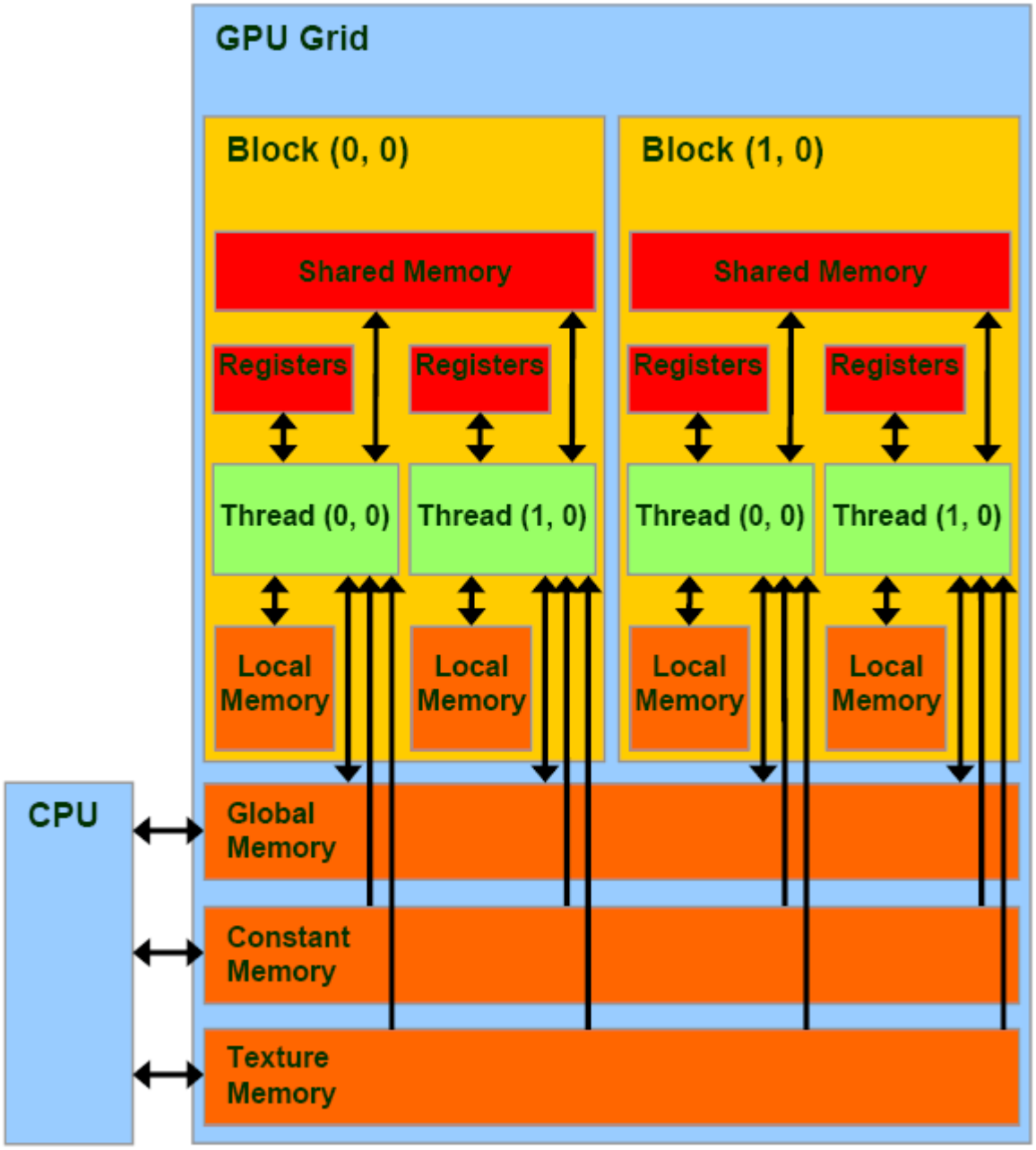
- расположена в мультипроцессоре
- выделяется на уровне блоков
- общее ограничение по объему
- малое время доступа
- может использоваться всеми нитями блока для чтения и записи

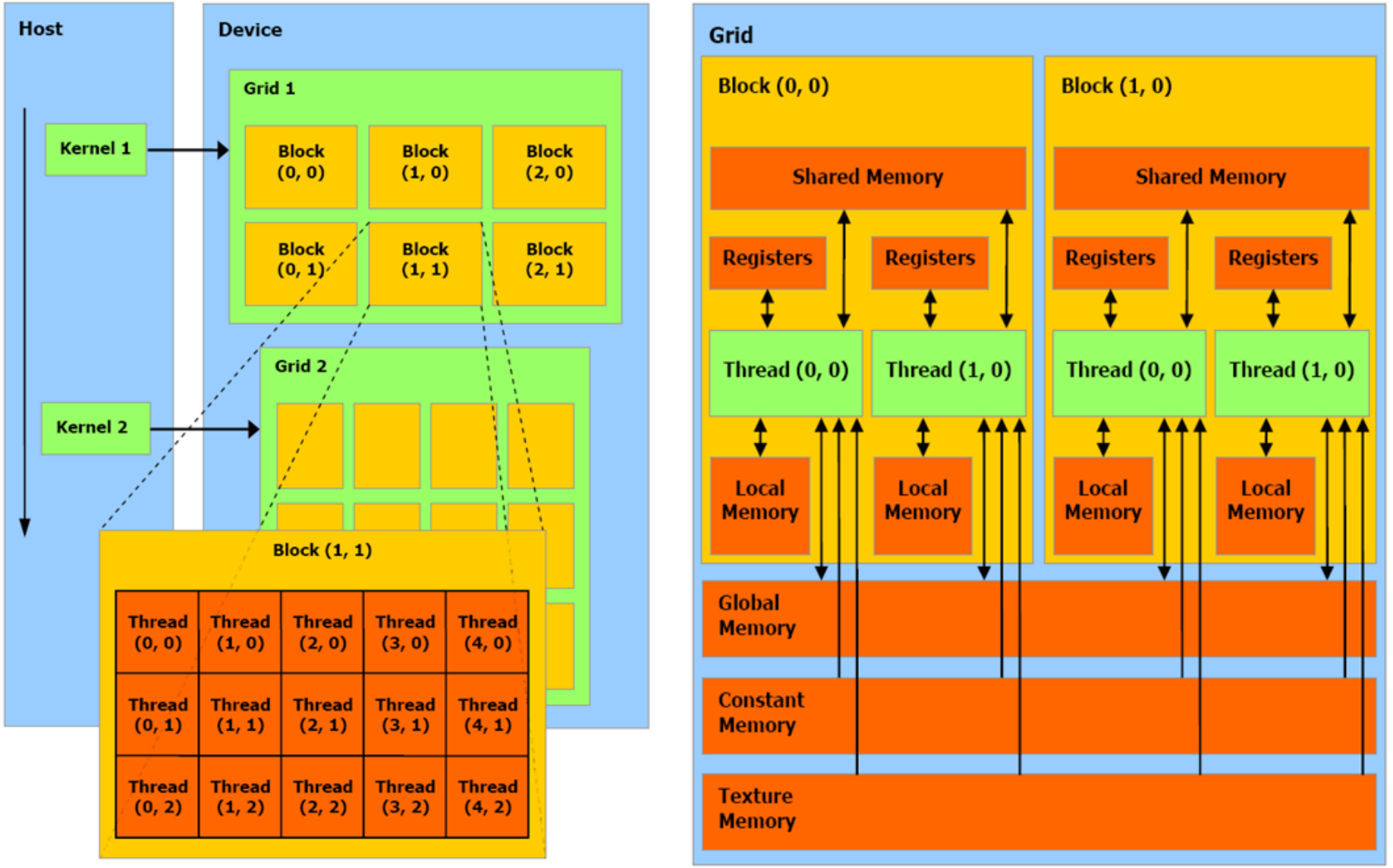
Константная и текстурная памяти

- расположены в DRAM
- независимый кэш
- относительно высокая скорость доступа
- доступ сразу всем нитям только на чтение
- общий объем ограничен

Виды памяти

- Регистры
- Локальная
- Разделяемая
- Глобальная
- Константная
- Текстурная
- CPU RAM


GPU Grid
Block (0, 0)
Block (1, 0)
Shared Memory
Shared Memory
Registers
Registers
Registers
Registers
Thread (0, 0)
Thread (1, 0)
Thread (0, 0)
Thread (1, 0)
Local Memory
Local Memory
Local Memory
Local Memory
CPU
Global Memory
Constant Memory
Texture Memory


Host
Device
Grid 1
Kernel 1
Block (0, 0)
Block (1, 0)
Block (2, 0)
Block (0, 1)
Block (1, 1)
Block (2, 1)
Grid 2
Kernel 2
Block (1, 1)
Thread (0, 0)
Thread (1, 0)
Thread (2, 0)
Thread (3, 0)
Thread (4, 0)
Thread (0, 1)
Thread (1, 1)
Thread (2, 1)
Thread (3, 1)
Thread (4, 1)
Thread (0, 2)
Thread (1, 2)
Thread (2, 2)
Thread (3, 2)
Thread (4, 2)
Grid
Block (0, 0)
Block (1, 0)
Shared Memory
Shared Memory
Registers
Registers
Registers
Registers
Thread (0, 0)
Thread (1, 0)
Thread (0, 0)
Thread (1, 0)
Local Memory
Local Memory
Local Memory
Local Memory
Global Memory
Constant Memory
Texture Memory

Регистровая память

Расположение:	мультипроцессор
Кэшируемость:	нет
Доступ:	GPU – R/W, CPU – нет
Скорость доступа:	максимальный
Время жизни:	thread

Не управляется из программного кода, размещение данных в регистрах выполняет компилятор

Локальная память

- не управляется из программного кода
- может быть использована компилятором при большом количестве локальных переменных в функции
- значительно медленнее, чем регистровая
- не предусмотрено явных средств, позволяющих блокировать использование локальной памяти

Локальная память

Расположение:	DRAM
Кэшируемость:	нет
Доступ:	GPU – R/W, CPU – нет
Скорость доступа:	низкая
Время жизни:	thread

проанализировать код и исключить лишние локальные переменные

Константная память

Кэшируемая область DRAM фиксированного размера, доступная с GPU только для чтения, для чтения и записи с хоста.

```
cudaError_t cudaMemcpyToSymbol
  (const char * symbol, const void * src, size_t count, size_t offset, enum cudaMemcpyKind kind);

cudaError_t cudaMemcpyFromSymbol
  (void * dst, const char * symbol, size_t count, size_t offset, enum cudaMemcpyKind kind);

cudaError_t cudaMemcpyToSymbolAsync
  (const char * symbol, const void * src, size_t count, size_t offset, enum cudaMemcpyKind kind,
   cudaStream_t stream);

cudaError_t cudaMemcpyFromSymbolAsync
   (void * dst, const char * symbol, size_t count, size_t offset, enum cudaMemcpyKind kind,
   cudaStream_t stream);
```

Константная память

Константная память выделяется непосредственно в коде программы при помощи спецификатора __constant__

```
__constant__ float constData [256];      // константная память GPU
float hostData [256];                    // данные в памяти CPU
...
// Скопировать данные из памяти CPU в константную память GPU
cudaMemcpyToSymbol(constData, hostData,
sizeof(data), 0, cudaMemcpyHostToDevice);

__global__ void kernel (float* pos)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    pos[index] = pos[index] * constData[idx];
}
```

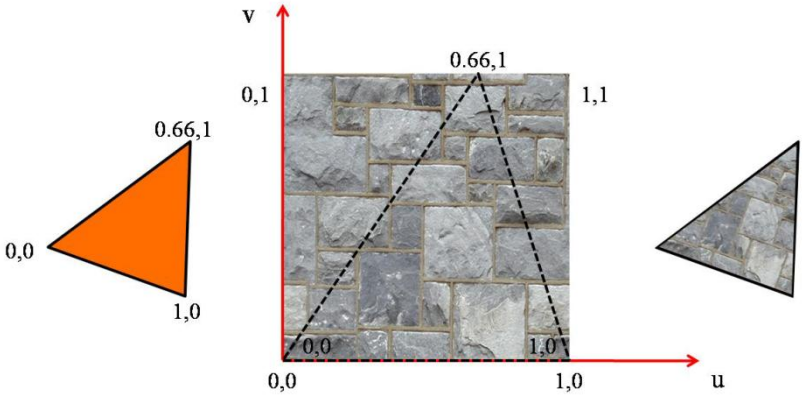
Константная память

Расположение:	DRAM
Кэшируемость:	да
Доступ:	GPU – R/O, CPU – R/W
Скорость доступа:	высокая (cache) / низкая
Время жизни:	выделяется CPU

Подходит для размещения небольшого объема часто используемых неизменяемых данных, которые должны быть доступны всем нитям.

Текстурная память

текстурная память
+ аппаратные схемы
интерполяции
= текстурные блоки



Количество текстурных блоков зависит от архитектуры, используются в графических задачах для заполнения треугольников двумерными изображениями (текстурами)

Текстурная память

В текстурном блоке аппаратно реализованы функции:

- фильтрация текстурных координат
- билинейная или точечная интерполяция
- разумное возвращаемое значение в случае, когда значения текстурных координат выходят за допустимые границы
- обращение по нормализованным или целочисленным координатам
- возвращение нормализованных значений
- Кэширование данных

Текстурная память

Предназначена главным образом для работы с текстурами и имеет специфические особенности в адресации, чтении и записи данных.

Выделяется с помощью *cudaMallocArray*

```
cudaError_t cudaMallocArray(struct cudaArray **arrayPtr,
    const struct cudaChannelFormatDesc *desc,
    size_t width, size_t height);

cudaError_t cudaFreeArray(struct cudaArray *array);
```

Текстурная память

arrayPtr – не простой указатель, а непрозрачный контейнер

Доступ можно получить через «текстурные ссылки»

Задача абстракции - отделить данные и способ их хранения от интерфейса доступа к ним

Для чтения из ядра сначала ассоциировать с его текстурной ссылкой *cudaBindTextureToArray*

Чтение текстуры производится с помощью *tex1D, tex2D, tex3D*

Глобальная память

- максимальный доступный объем
- произвольный доступ
- сохраняет целостность данных на протяжении всего времени жизни приложения
- основное хранилище для передачи данных между ядрами

Глобальная память

*cudaError_t cudaMalloc(void ** devPtr, size_t size);*

*cudaError_t cudaFree(void * devPtr);*

*cudaError_t cudaMallocPitch(void ** devPtr, size_t * pitch,size_t width, size_t height);*

*cudaError_t cudaMemcpy (void * dst, const void * src, size_t size,*
enum cudaMemcpyKind kind);

*cudaError_t cudaMemcpyAsync (void * dst, const void * src, size_t size,*
enum cudaMemcpyKind kind, cudaStream_t stream);

Глобальная память

- прямая адресация (адресная арифметика)
- указатель в контексте GPU
- копирование данных
  - внутри устройства device < = > device

    144 Гбайт / сек (Tesla C2050)
  - между хостом и устройством host <= > device

    PCI-E 2.0 до 8 Гбайт/сек, на практике не более 4 Гбайт/сек

Глобальная память

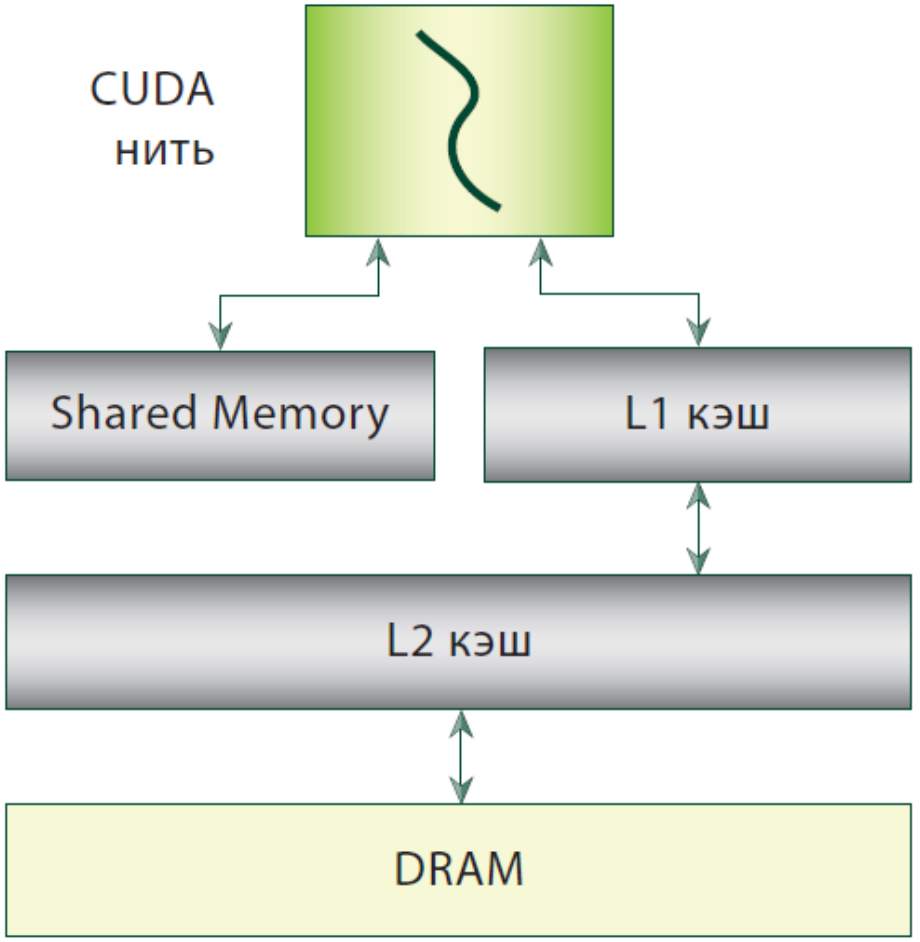
двухуровневое кэширование начиная с Fermi

L2 = 768 Кбайт

L1 + Shared =

16 + 48 (default)

48 + 16



Глобальная память

транспонирование матрицы

```
__global__ void transpose ( float * inData, float * outData, int n )
{
unsigned int xIndex = blockDim.x * blockIdx.x + threadIdx.x;
unsigned int yIndex = blockDim.y * blockIdx.y + threadIdx.y;
unsigned int inIndex = xIndex + n * yIndex;
unsigned int outIndex = yIndex + n * xIndex;
outData [outIndex] = inData [inIndex];
}
```

Глобальная память

Расположение:	DRAM
Кэшируемость:	да, но не стоит надеяться
Доступ:	GPU – R/W, CPU – R/W
Скорость доступа:	низкая
Время жизни:	выделяется CPU

Передача данных между CPU и GPU, хранение данных во время работы программы.

Разделяемая память

- расположена в мультипроцессоре
- доступна всем нитям в одном блоке
- 16 / 48 Кбайт (compute capability 2.x)
- Объем делится поровну между всеми блоками, запущенными на мультипроцессоре

Разделяемая память

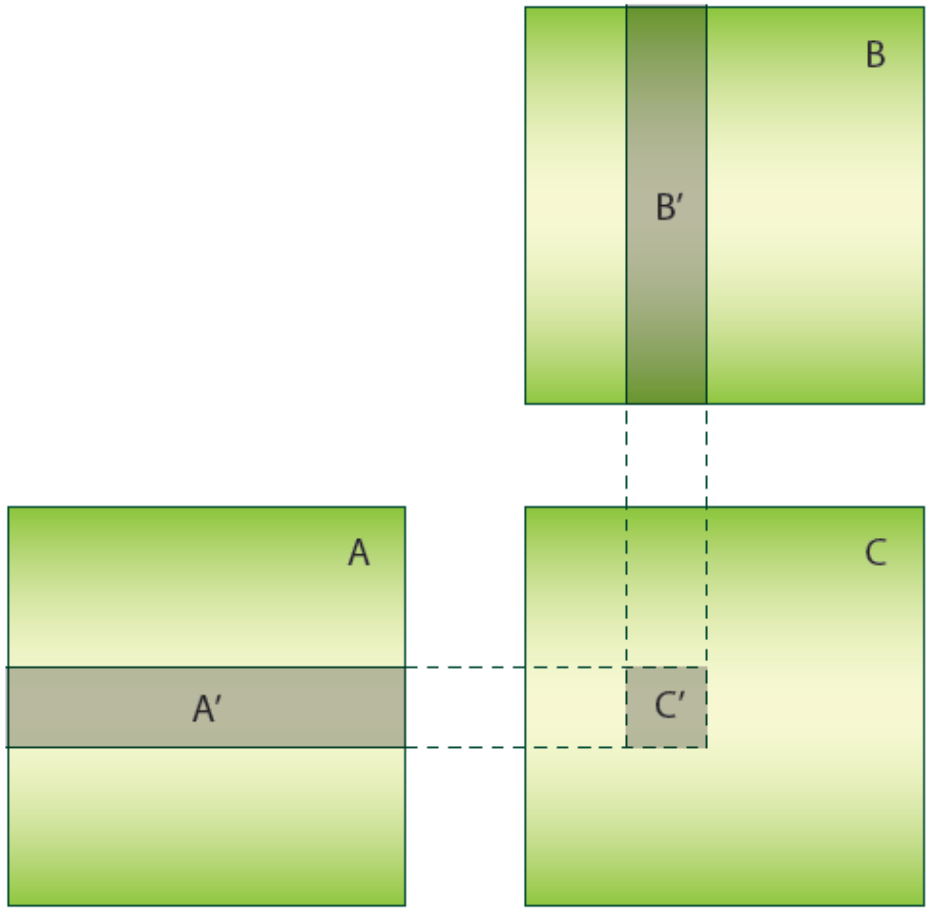
```
__global__ void kernel1(float* a)
{
    // Явно задано выделить 256*4 байт на блок.
    __shared__ float buf [256];
    // Запись значения из глобальной памяти в разделяемую.
    buf [threadIdx.x] = a [blockIdx.x * blockDim.x + threadIdx.x];
    …
    __syncthreads ()
    …
    // Запись результата обратно в глобальную память
}
```

Разделяемая память

Умножение матриц

C’ – подматрица размером 16x16

A’, B’ – полосы подматриц размером Nx16



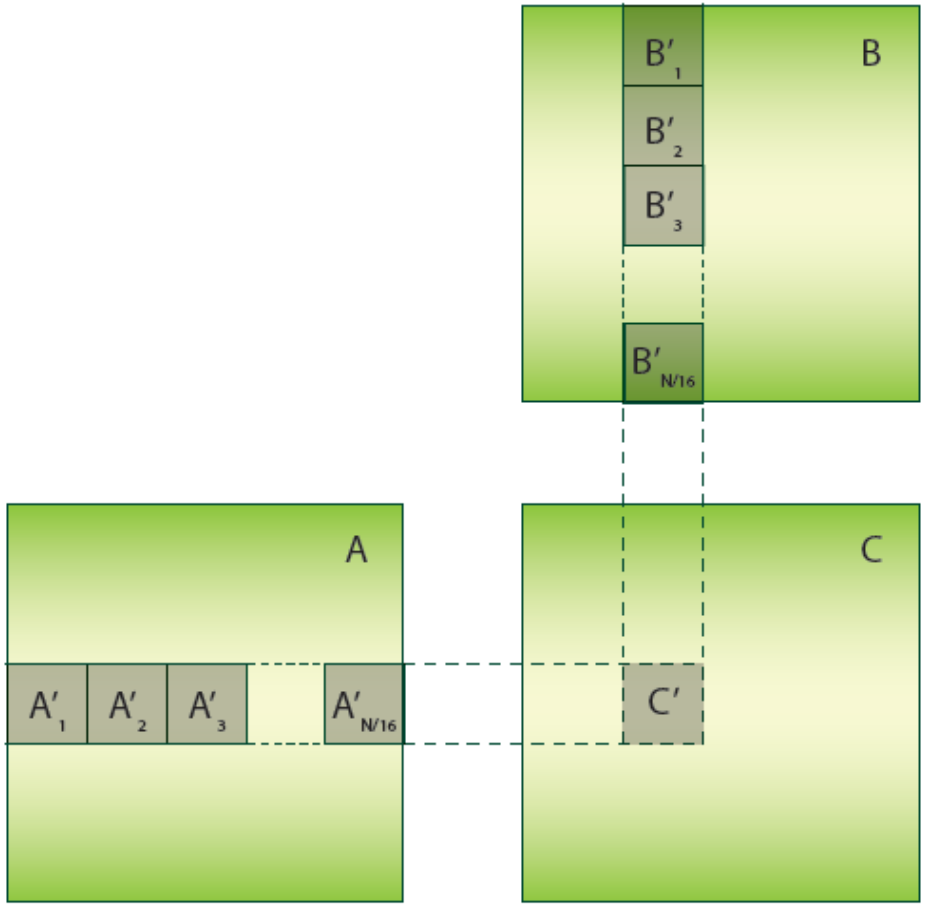
Разделяемая память

Разбить A’, B’ на квадратные

Подматрицы 16x16

Загружать A'_i и B'_i в разделяемую память поочередно

Снижение обращений в глобальную память



Разделяемая память

Перемножение матриц 2048x2048 на Tesla C2070

Без использования разделяемой памяти

количество чтений из глобальной памяти: 38 535 168

время выполнения: 324.63 мс

С использование разделяемой памяти

количество чтений из глобальной памяти: 1 196 032

время выполнения: 93.26 мс

Разделяемая память

Расположение:	мультипроцессор
Кэшируемость:	нет
Доступ:	GPU – R/W, CPU – нет
Скорость доступа:	высокая
Время жизни:	block

Используется в качестве управляемого кэша первого уровня