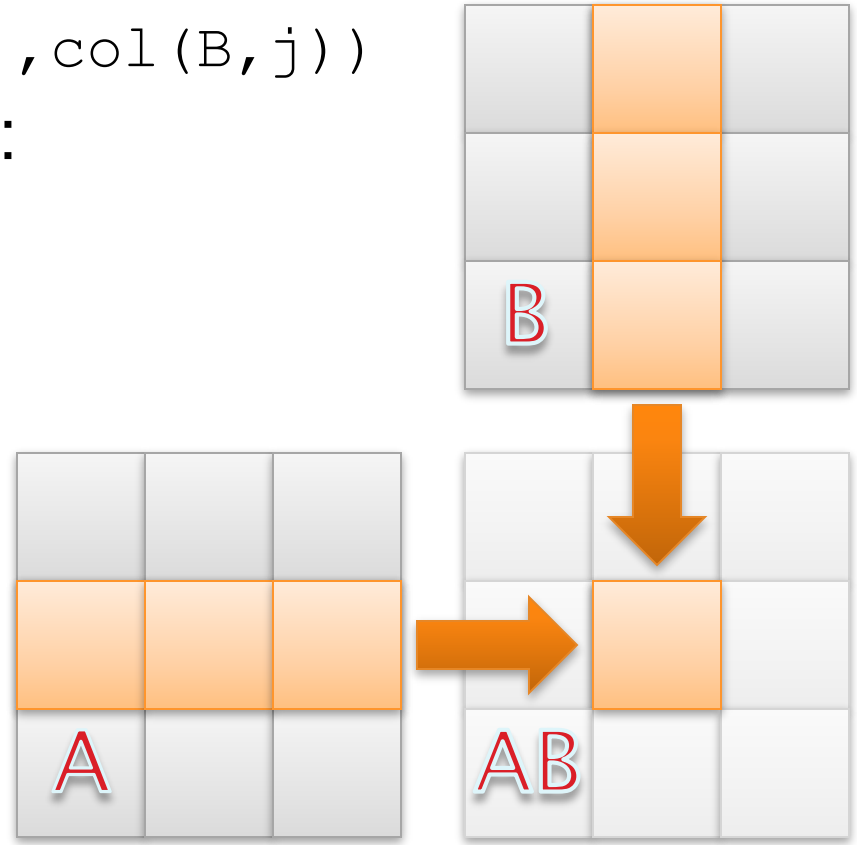
Пример: перемножение матриц

- AB = A * B
  - AB_{ij} = `dot(row(A,i),col(B,j))`
- Распараллеливаем:
  - Thread – AB_{ij}
  - 2D kernel



Первая попытка

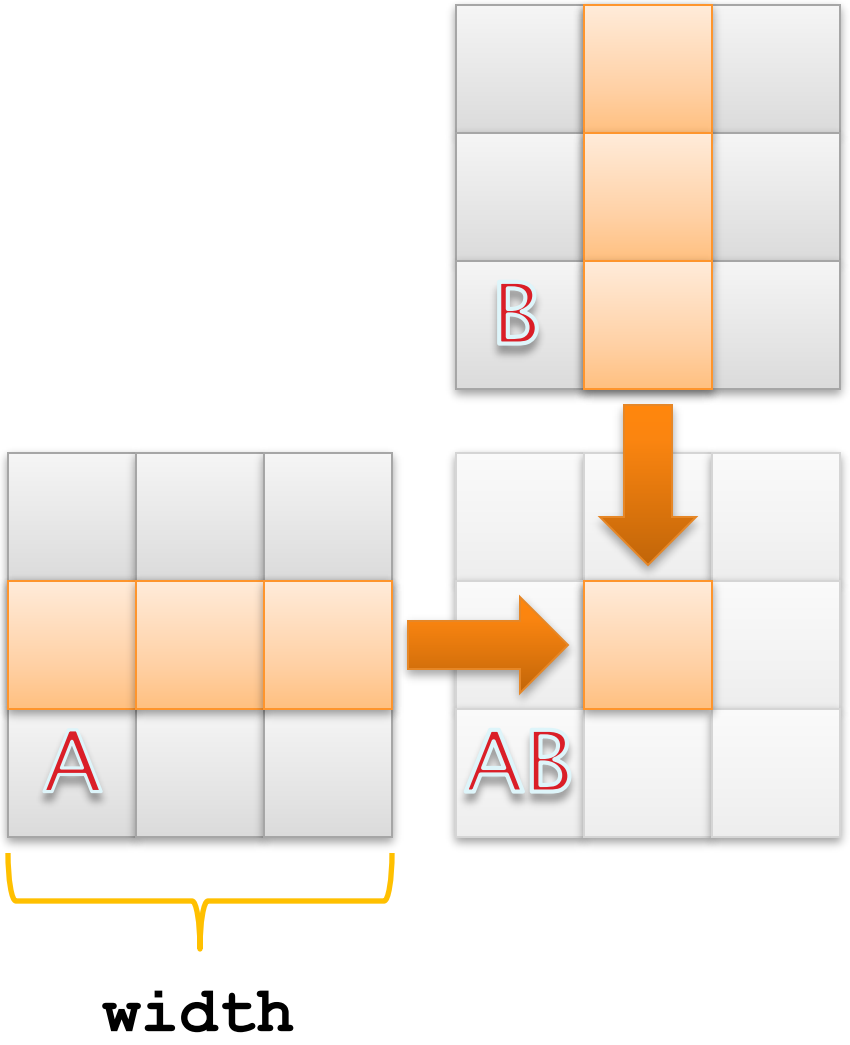
```
__global__ void mat_mul(float *a, float *b,
                        float *ab, int width)
{
  // calculate the row & col index of the element
  int row = blockIdx.y*blockDim.y + threadIdx.y;
  int col = blockIdx.x*blockDim.x + threadIdx.x;

  float result = 0;

  // do dot product between row of a and col of b
  for(int k = 0; k < width; ++k)
    result += a[row*width+k] * b[k*width+col];

  ab[row*width+col] = result;
}
```

Оценим производительность

Кол-во считываемых байт для каждого произведения `[row*width+k]*b[k*width+col]`	2 (a & b) = 8 Bytes
Кол-во операций	2 (multiply & addition)
Арифметическая интенсивность	8 Bytes / 2 ops = 4 B/op
Пиковая производительность GeForce GTX 1060?	4 TFLOPS
С какой скоростью нужно считывать данные для достижения пиковой производительности	GMAC * Peak FLOPS = 4 * 4 = 16 TB/s
Реальная пропускная способность памяти GeForce GTX 1060	192 GB/s
Какая получится производительность?	Actual BW / GMAC = 192 / 4 = 48 GFLOPS

Идея: использовать __shared__ память для экономии повторных обращений к глобальной памяти

- Каждый элемент–подматрица считывается блоком потоков в __shared__ память
- Потоки перемножают подматрицы внутри разделяемого кэша



Версия 2

```
__global__ void mat_mul(float *a, float *b,
                        float *ab, int width)
{
  int tx = threadIdx.x, ty = threadIdx.y;
  int bx = blockIdx.x,  by = blockIdx.y;
  // выделение памяти в кэше
  __shared__ float s_a[TILE_WIDTH][TILE_WIDTH];
  __shared__ float s_b[TILE_WIDTH][TILE_WIDTH];

  int row = by*blockDim.y + ty;
  int col = bx*blockDim.x + tx;
  float result = 0;
```

Версия 2

```
// цикл по подматрицам в полосе
for(int p = 0; p < width/TILE_WIDTH; ++p)
{
  // загрузка подматриц в кеш __shared__
  s_a[ty][tx] = a[row*width + (p*TILE_WIDTH + tx)];
  s_b[ty][tx] = b[(p*TILE_WIDTH + ty)*width + col];
  __syncthreads();

  // вычисление произведения подматриц s_a и s_b
  for(int k = 0; k < TILE_WIDTH; ++k)
    result += s_a[ty][k] * s_b[k][tx];
  __syncthreads();
}
      ab[row*width+col] = result;
}
```

Размеры

- Размер блока
  - `TILE_WIDTH` = 32 ⇒ 32*32 = 1024 потока
- Размер грида
  - матрица 1024*1024 ⇒ 32*32 = 1024 подматриц
  - `TILE_WIDTH` = 32 ⇒ 2 блока на мультипроцессор SM, 2048 потоков
  - Полная загруженность GeForce GTX 1060

- Каждый блок считывает 2 * 1024 = 2048 чисел и делает 1024 * (2 * 32) = 65 536 операций
  - Вычисления больше не упираются в пропускную способность памяти

Сравнение алгоритмов

	Простой	Оптимизированный
Чтений из глоб. памяти	$2N^3$	$2N^2$ *(N/`TILE_WIDTH`)
Производительность	17 GFLOPS	640 GFLOPS
SLOCs	20	44
Ускорение	1x	37x
Ускорение/SLOC	1x	18x

- Усложнение программы окупилось
- Теоретически можно добиться еще большей производительности

Ограничения, связанные с памятью

Ресурс	Доступно на GTX1060 SM	Ограничение
Регистры	64KB	< 65536 / 1024 threads = 65 per thread
`__shared__` память	48KB	<= 48KB / 2 blocks = 24KB per block

- Эффективное использование различных видов памяти снижает количество обращений к глобальной
- Ресурсы ограничены!
- Чем больше требуется памяти, тем меньше потоков можно запустить на одном мультипроцессоре