CMPSCI 691AD – General Purpose Computation on the GPU

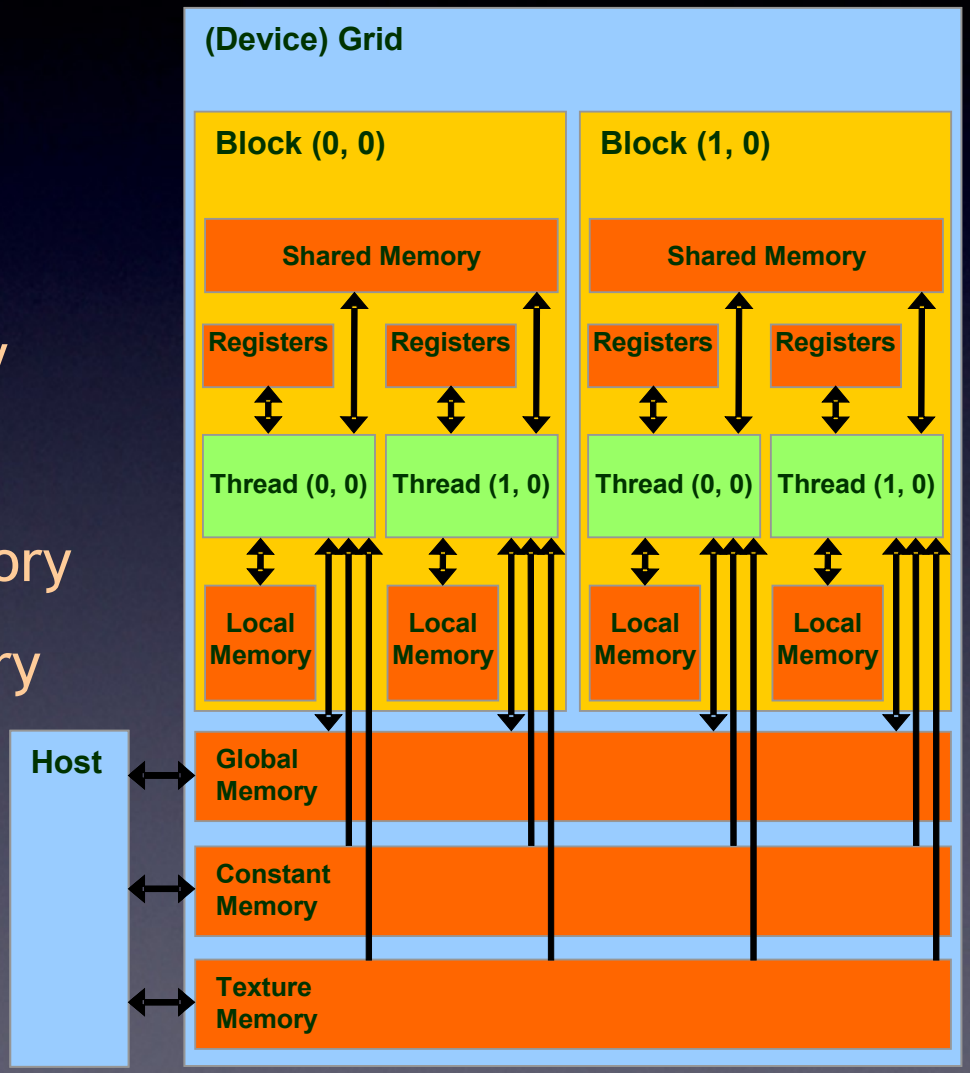
Spring 2009

Lecture 8: CUDA Memory Models I

Rui Wang

CUDA Memory Hierarchy

- Available Memory Space:
  - R/W per-thread registers
  - R/W per-thread local memory
  - R/W per-block shared memory
  - R/W per-grid global memory
  - R-only per-grid constant memory
  - R-only per-grid texture memory
- Host can R/W global, constant, and texture memory.



Typical Data Movement

Host Memory

Device (global) Memory

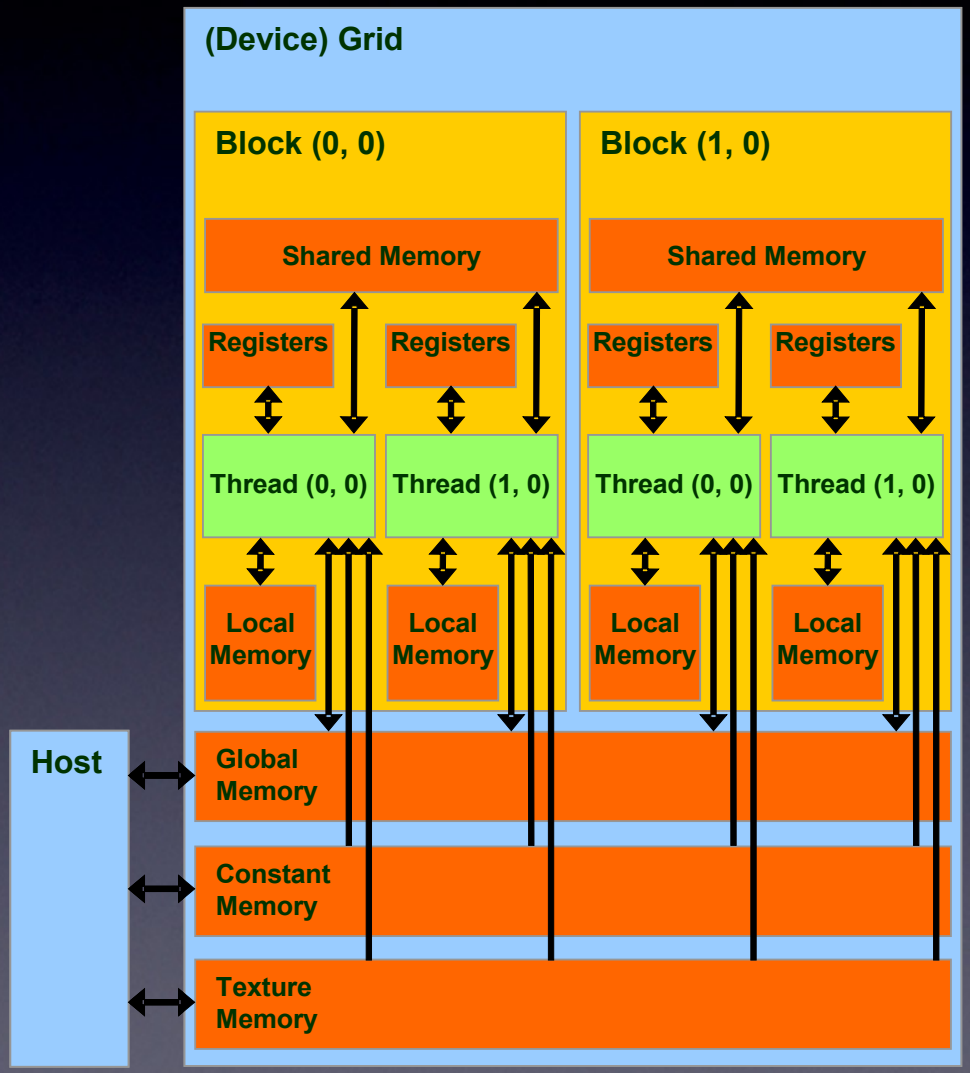
[Shared Memory]

Computation

[Shared Memory]

Device (global) Memory

Host Memory

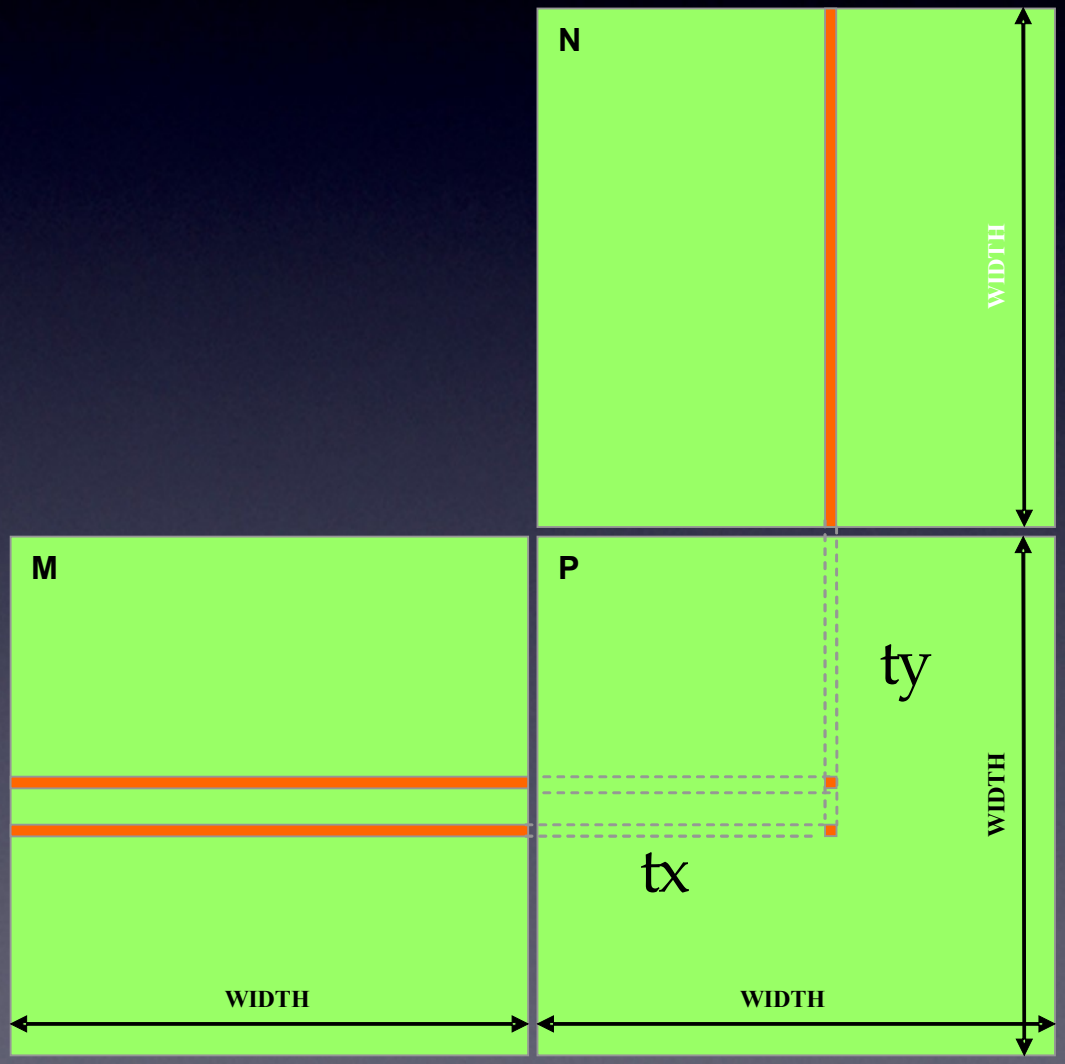


Take Advantage of Shared Memory

- Shared memory is user-managed L1 cache, and is much faster to access than global memory (4 clock cycle vs. 400-600 clock cycles)
- Take advantage of shared memory to:
  - Reduce access to global memory if the same global memory location will be accessed multiple times.

Example: Matrix Multiplication

- First try: directly mapping matrix mult to the GPU:



Example: Matrix Multiplication

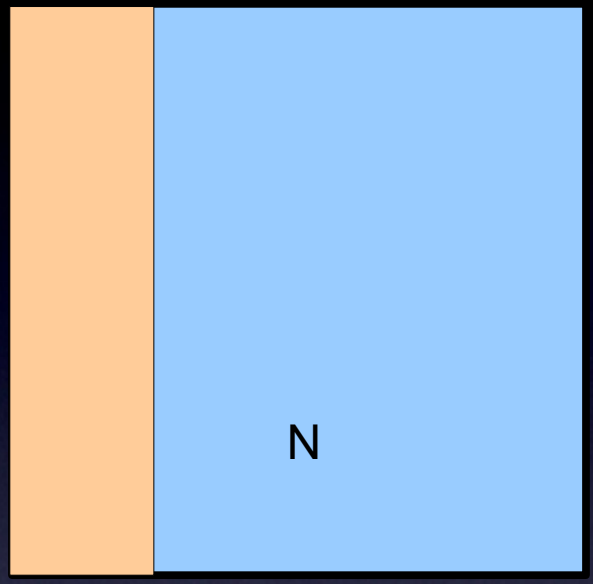
- Compute to Global Memory Access (CGMA) ratio?
  - 1.0
- 8800 GTX bandwidth: 86.4 GB/s
  - 1 FP (4 bytes) per FP operation, and 1.0 CGMA means the maximum operations we can perform is at 21.6 GFLOPS, significantly less than the peak performance of 367 GFLOPS (if no memory bottleneck)
  - From another angle: each data element in the source array is read WIDTH times during the computation, huge waste of memory bandwidth.

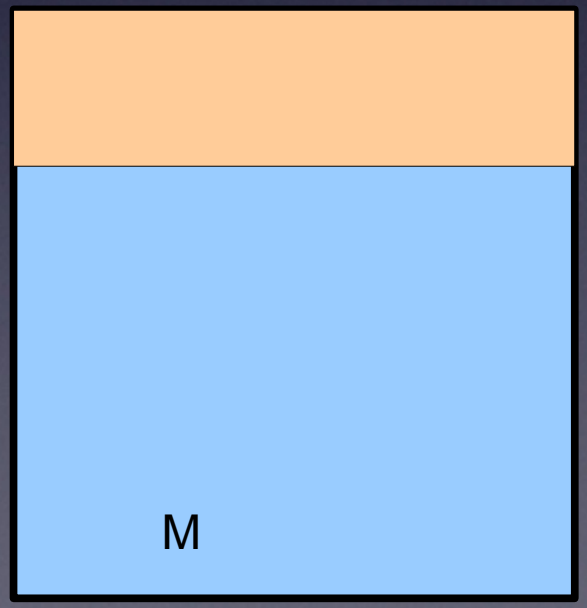
Take Advantage of Shared Memory

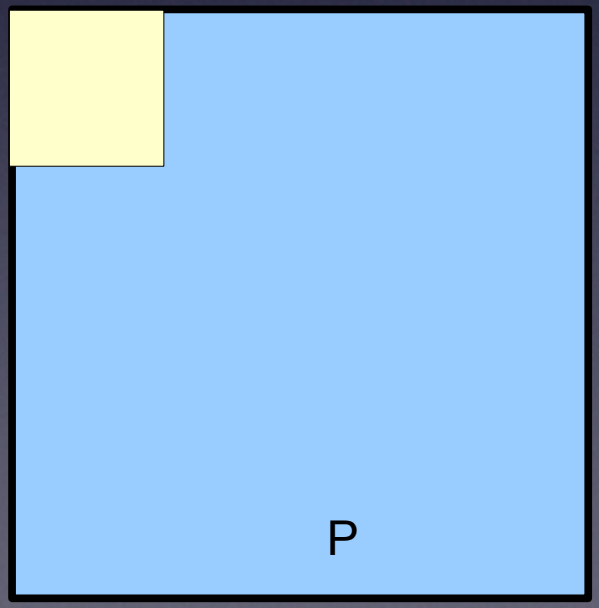
- Take advantage of shared memory to:
  - Reduce access to global memory if the same global memory location will be accessed multiple times.
    - Load data once to shared memory, reuse them many times

Take Advantage of Shared Memory

- Each element in the source array is only read once from the global memory and stored in shared memory. So subsequent access to the data is almost free. Great!
- But...



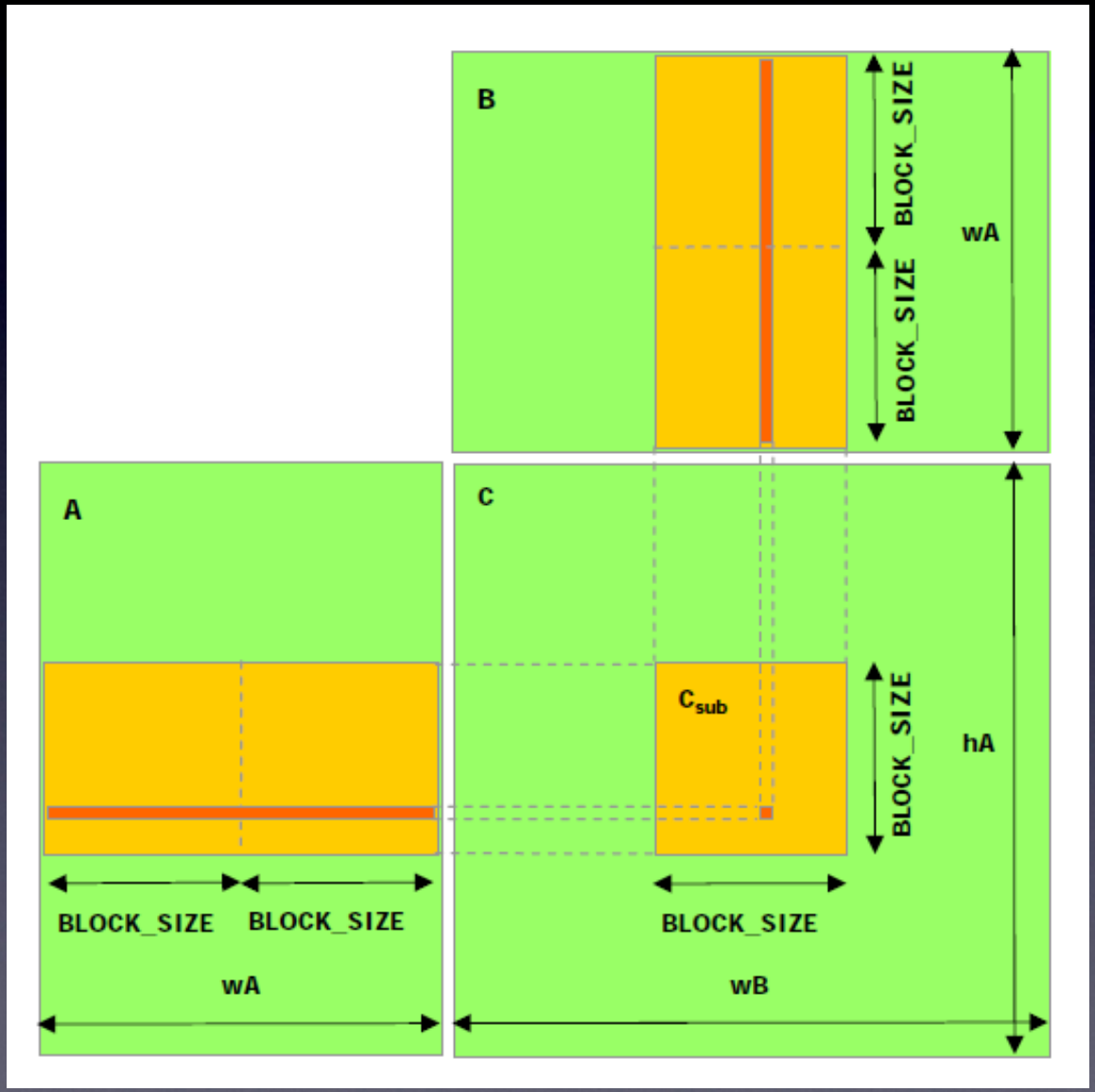


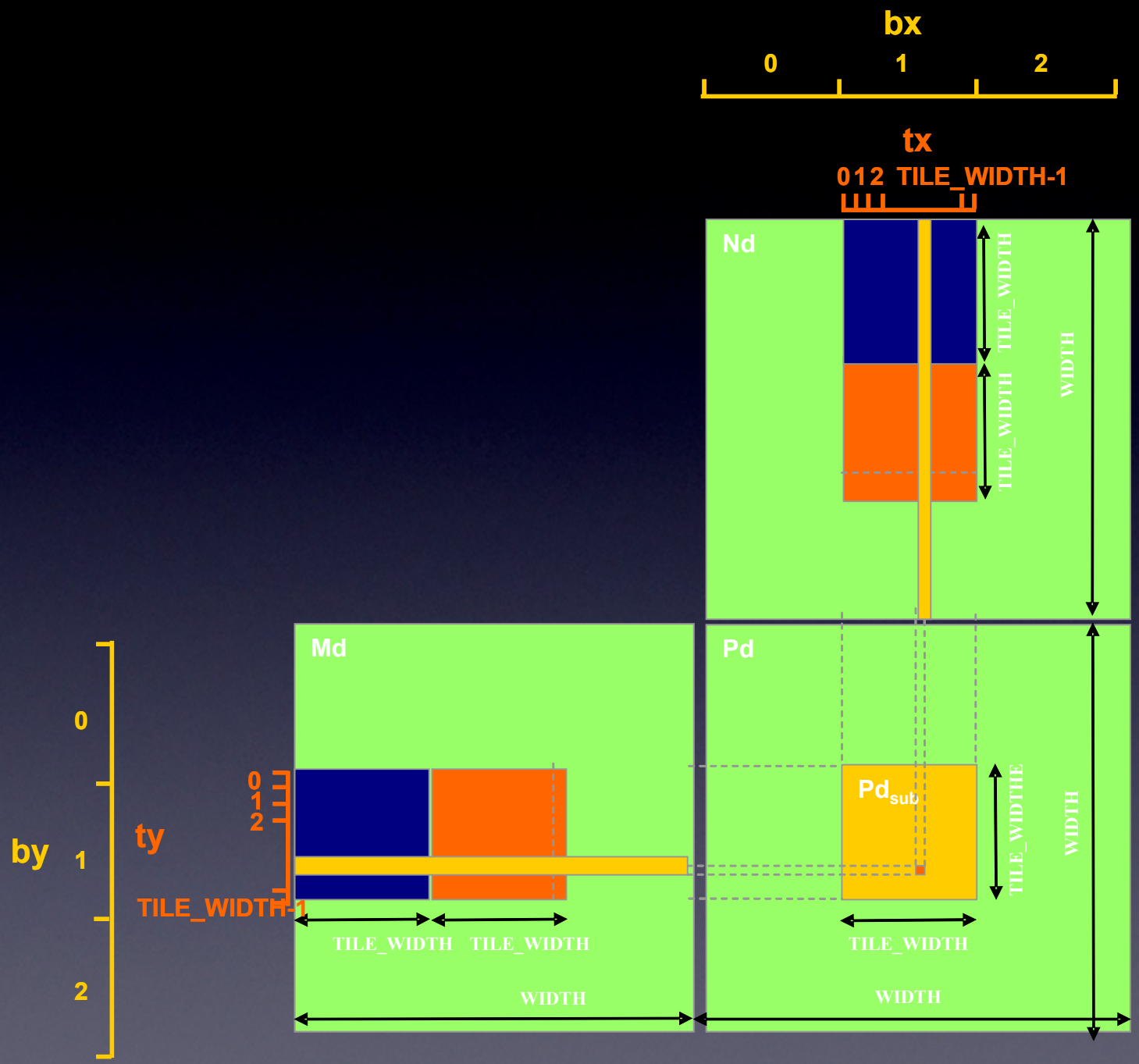


Take Advantage of Shared Memory

- Take advantage of shared memory to:
  - Reduce access to global memory if the same global memory location will be accessed multiple times.
    - Load data once to shared memory, reuse them many times
    - But share memory is limited, so we need to partition the data into smaller chunks

Take Advantage of Shared Memory




bx
0
1
2
tx
0 1 2 TILE_WIDTH-1
Nd
TILE_WIDTH
TILE_WIDTH
WIDTH
Md
Pd
by
0
1
2
ty
0
1
2
TILE_WIDTH-1
Pd_sub
TILE_WIDTHE
WIDTH
TILE_WIDTH
TILE_WIDTH
WIDTH
TILE_WIDTH
WIDTH

Take Advantage of Shared Memory

```
__global__ void MatrixMulKernel(float* Md, float* Nd, float* Pd, int Width)
{
1.  __shared__float Mds[TILE_WIDTH][TILE_WIDTH];
2.  __shared__float Nds[TILE_WIDTH][TILE_WIDTH];

3.  int bx = blockIdx.x;  int by = blockIdx.y;
4.  int tx = threadIdx.x; int ty = threadIdx.y;

// Identify the row and column of the Pd element to work on
5.  int Row = by * TILE_WIDTH + ty;
6.  int Col = bx * TILE_WIDTH + tx;

7.   float Pvalue = 0;
// Loop over the Md and Nd tiles required to compute the Pd element
8.    for (int m = 0; m < Width/TILE_WIDTH; ++m) {

// Coolaborative loading of Md and Nd tiles into shared memory
9.      Mds[ty][tx] = Md[Row][m*TILE_WIDTH + tx];
10.     Nds[ty][tx] = Nd[m*TILE_WIDTH + ty][Col];
11.   __Syncthreads();

12.   for (int k = 0; k < TILE_WIDTH; ++k)
13.       Pvalue += Mds[ty][k] * Nds[k][tx];

14.   Pd[Row][Col] = Pvalue;
    }
}
```

Take Advantage of Shared Memory

- How many times is each element in the source array read from global memory during this computation? (assume block size is 16)

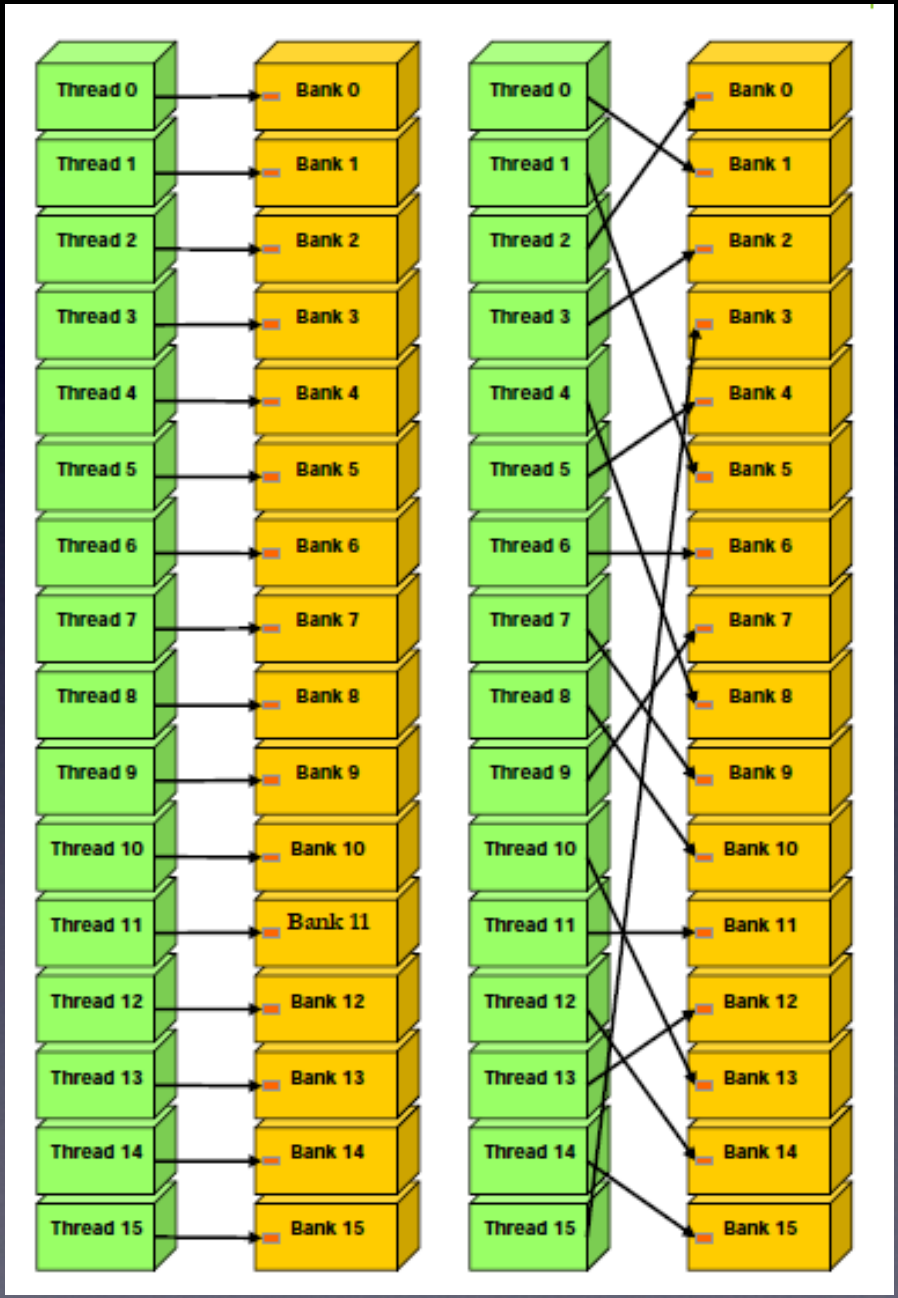
Take Advantage of Shared Memory

- How many times is each element in the source array read from global memory during this computation? (assume block size is 16)
  - Reduces global memory access by 16x!
  - CGMA is 16, so can achieve 21.6 x 16 = 345 GFLOPS, almost the peak performance of the hardware!

Shared Memory Bank Conflicts

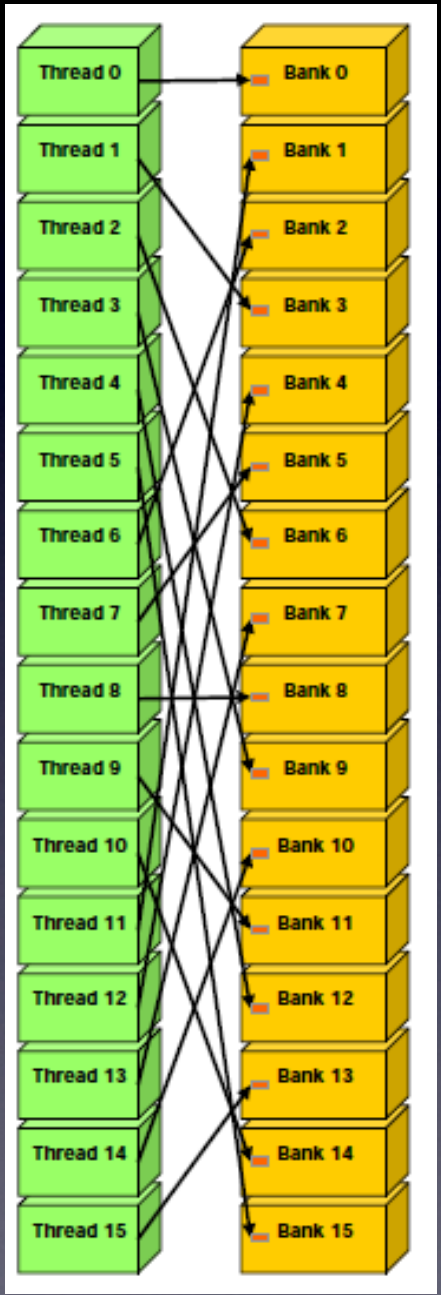
- Each SM has 16KB shared memory arranged in 16x 1KB memory banks; each successive 4 Byte word is assigned to successive banks.
- Within a half-warp (every 16 threads), if every thread accesses a successive 4 Byte word, then all the accesses can be performed together.

Shared Memory Bank Conflicts



Conflict Free

Shared Memory Bank Conflicts



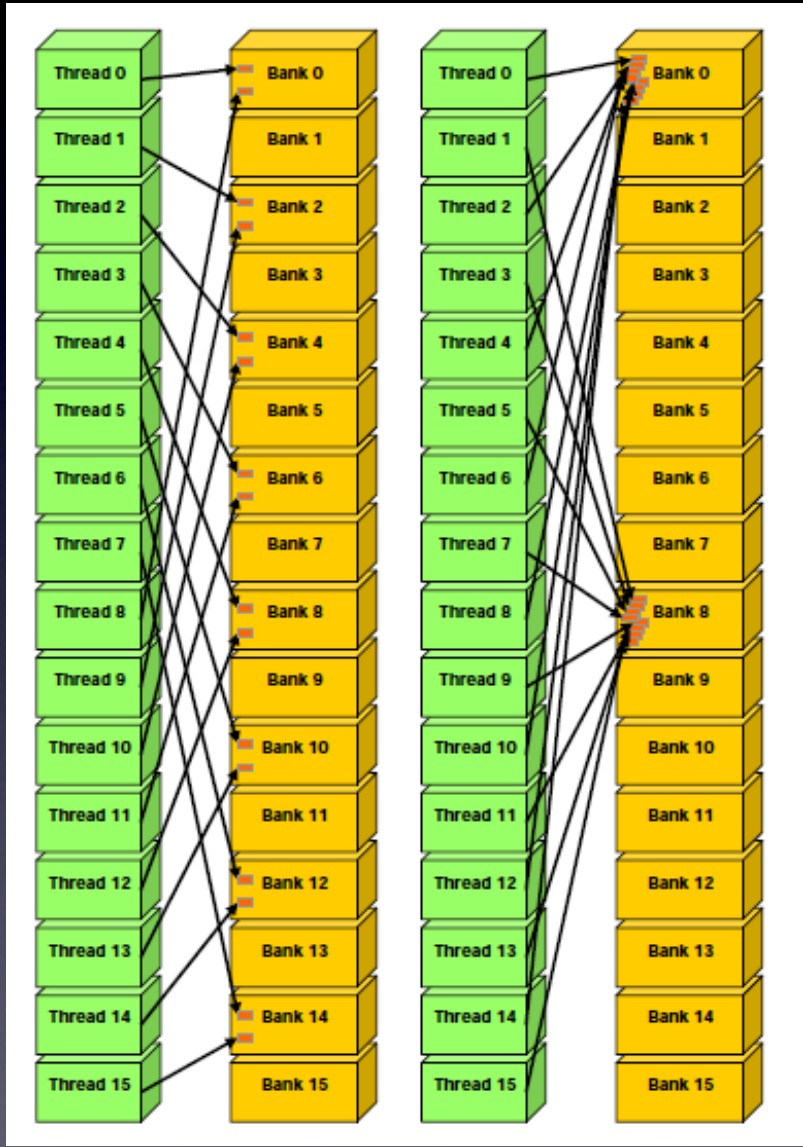
Stride = 3
Conflict Free

Shared Memory Bank Conflicts

- Each SM has 16KB shared memory arranged in 16x 1KB memory banks; each successive 4 Byte word is assigned to successive banks.
- Within a half-warp (every 16 threads), if every thread accesses a successive 4 Byte word, then all the accesses can be performed together.
- However, if two threads try to access shared memory elements that are in the same bank, we have a bank conflict.
- If the number of memory requests for the same bank is n, then we say this is a n-way bank conflicts.

Shared Memory Bank Conflicts

Stride = 2
2-way conflict



Stride = 8
8-way conflict