CUDA Memories

John H. Osorio Ríos

Importance of Memory Access Efficiency (1/2)

```
__global__ void MatrixMulKernel(float* d_M, float* d_N, float* d_P, intWidth) {

  // Calculate the row index of the d_P element and d_M
  int Row = blockIdx.y*blockDim.y+threadIdx.y;

  // Calculate the column index of d_P and d_N
  int Col = blockIdx.x*blockDim.x+threadIdx.x;

  if ((Row < Width) && (Col < Width)) {
    float Pvalue = 0;
    // each thread computes one element of the block sub-matrix
    for (int k = 0; k < Width; ++k) {
      Pvalue += d_M[Row*Width+k]*d_N[k*Width+Col];
    }
    d_P[Row*Width+Col] = Pvalue;
  }

}
```

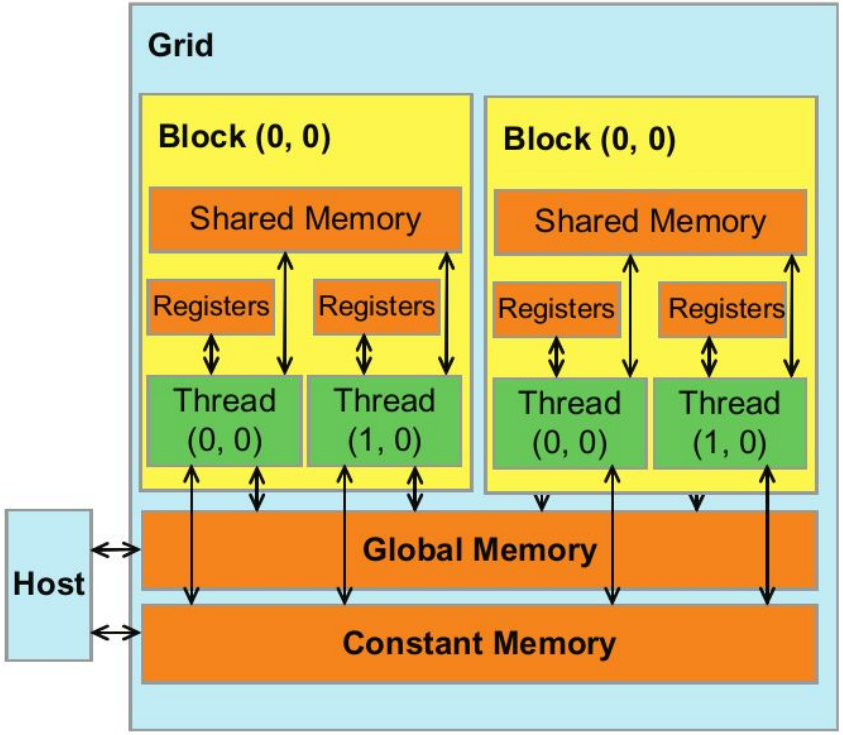
Importance of Memory Access Efficiency (2/2)

```
for (int k = 0; k < Width; ++k)
  Pvalue += d_M[Row*Width + k] * d_N[k*Width + Col];
```

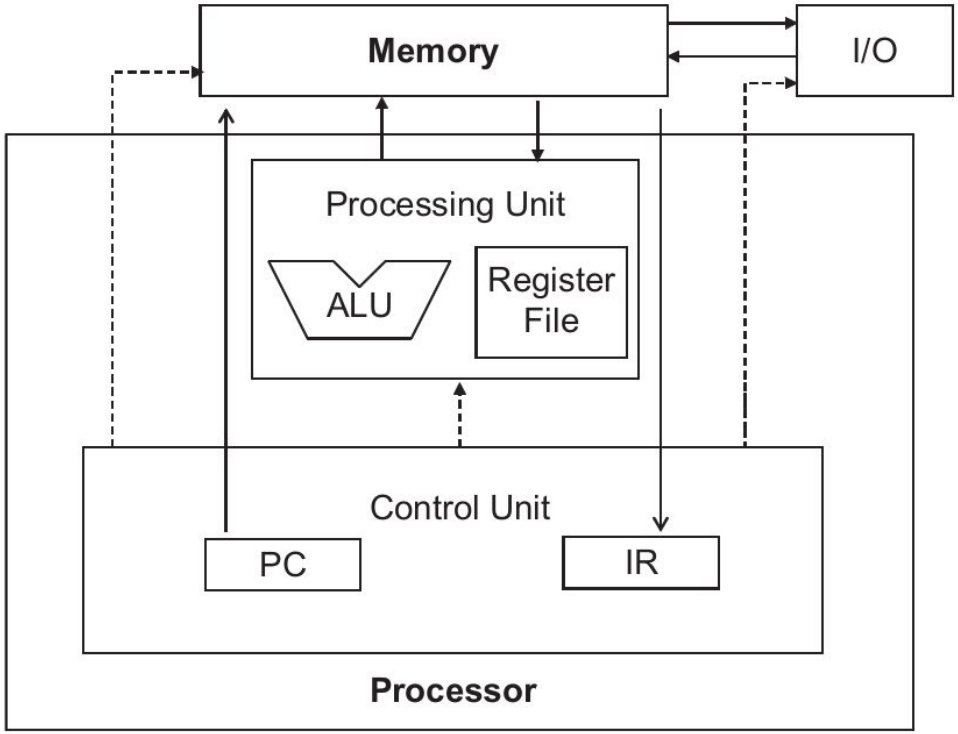
Compute to Global Memory Access (CGMA) ratio

- Our application has 1.0 CGMA Ratio
- 200 GB/s memory bandwidth.
- 4 bytes operands.
- 50 Giga single-precision Operands per Second
- Our Application has 50 GFLOPS
- Peak single precision performance 1500 GFLOPS

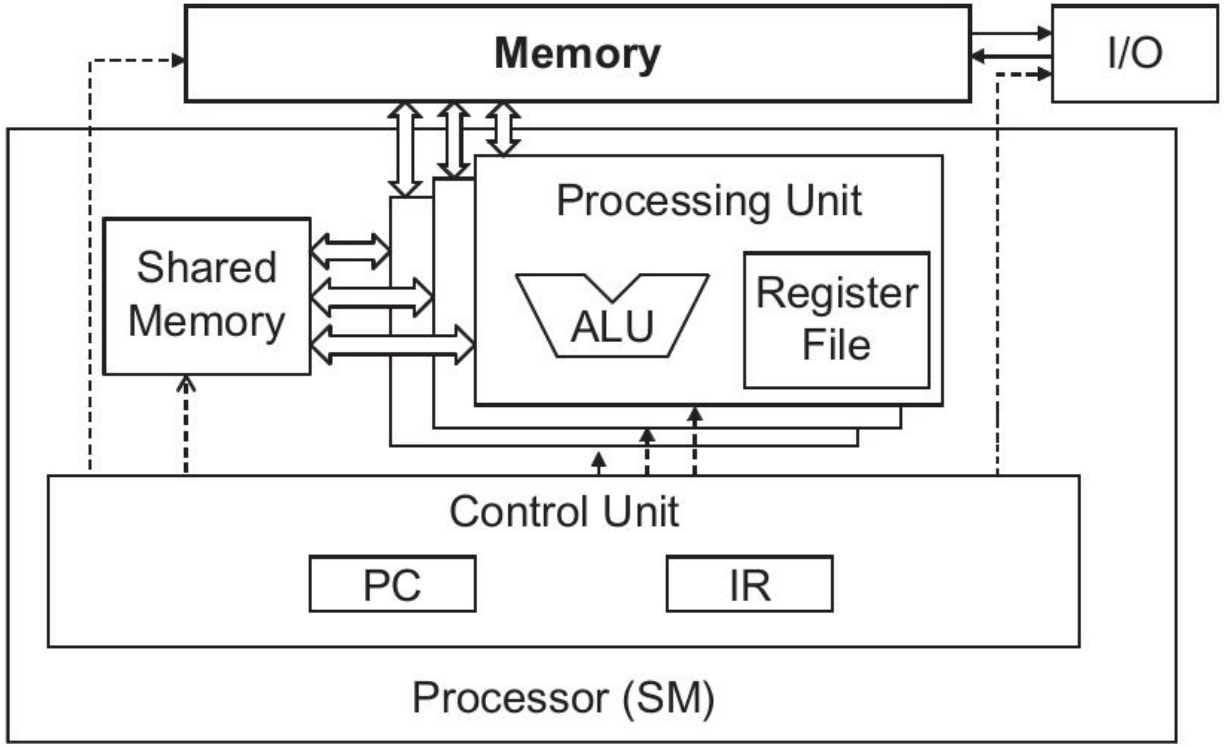
CUDA Device Memory Types (1/4)



CUDA Device Memory Types (2/4)



CUDA Device Memory Types (3/4)



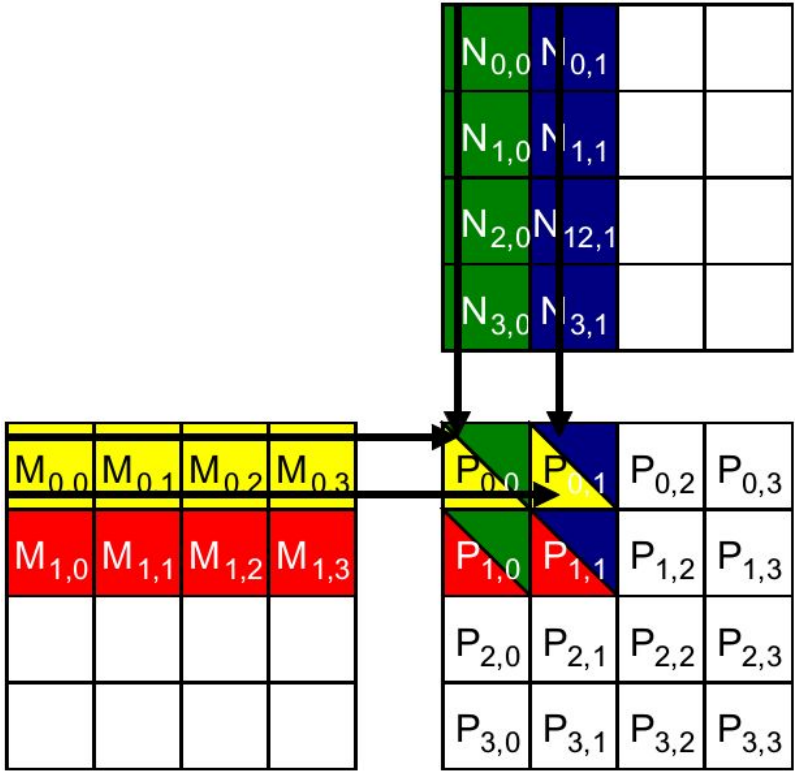
CUDA Device Memory Types (4/4)

Table 5.1 CUDA Variable Type Qualifiers

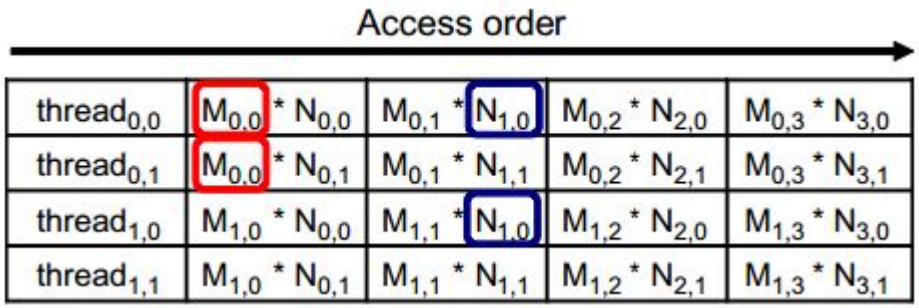
Variable Declaration	Memory	Scope	Lifetime
Automatic variables other than arrays	Register	Thread	Kernel
Automatic array variables	Local	Thread	Kernel
`__device__ __shared__ int SharedVar;`	Shared	Block	Kernel
`__device__ int GlobalVar;`	Global	Grid	Application
`__device__ __constant__ int ConstVar;`	Constant	Grid	Application

A Strategy for Reducing Global Memory Traffic (1/4)

- Global Memory
  - Large
  - Slow
- Shared Memory
  - Small
  - Fast



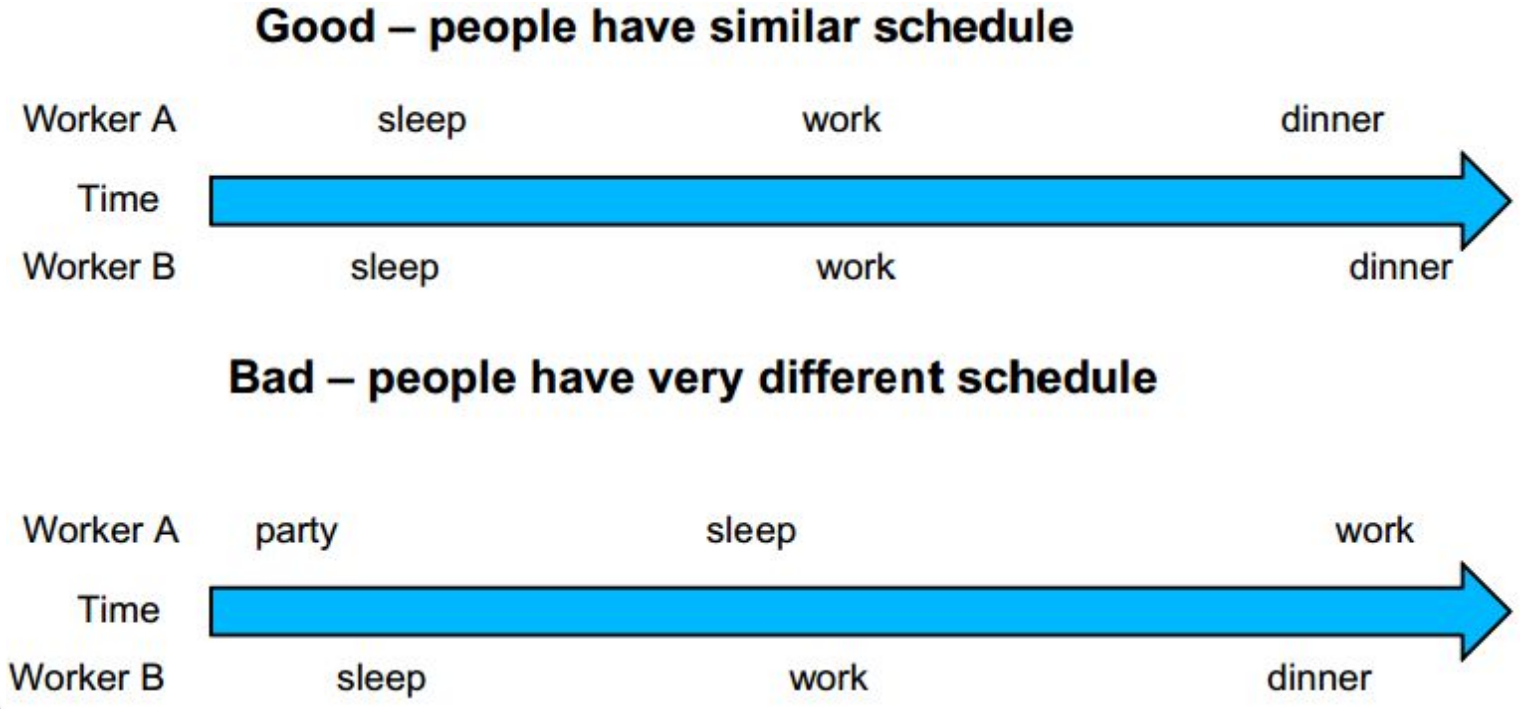
A Strategy for Reducing Global Memory Traffic (2/4)



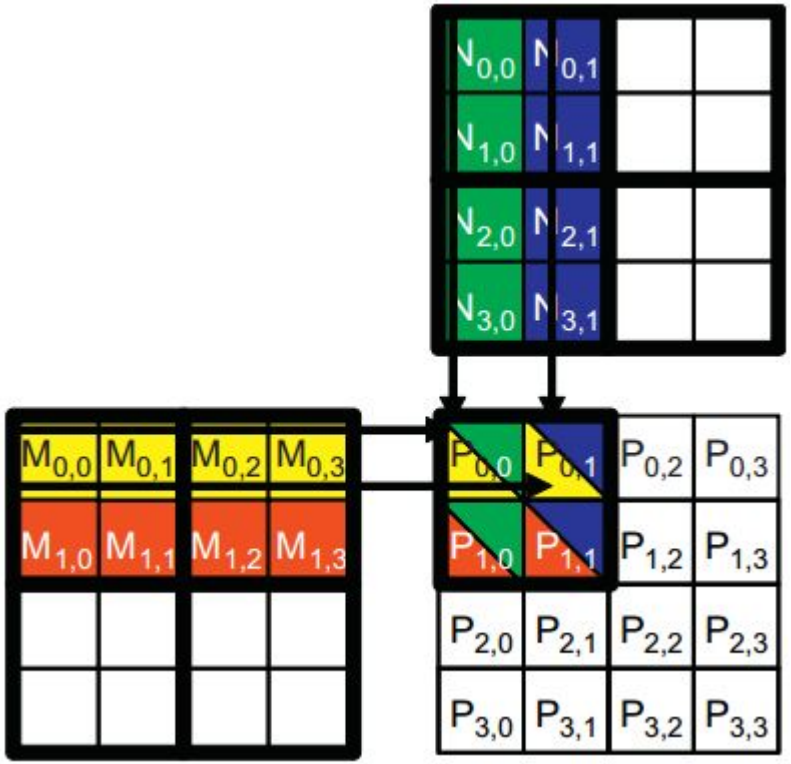
- With NXN Blocks
  - The access to global memory reduce by 1/N

A Strategy for Reducing Global Memory Traffic (3/4)

A Strategy for Reducing Global Memory Traffic (4/4)



A Tiled Matrix-Matrix Multiplication Kernel (1/3)



A Tiled Matrix-Matrix Multiplication Kernel (2/3)

	Phase 1			Phase 2		
$thread_{0,0}$	**$M_{0,0}$** ↓ $Mds_{0,0}$	**$N_{0,0}$** ↓ $Nds_{0,0}$	$PValue_{0,0}$ += $Mds_{0,0}$*$Nds_{0,0}$ + $Mds_{0,1}$*$Nds_{1,0}$	**$M_{0,2}$** ↓ $Mds_{0,0}$	**$N_{2,0}$** ↓ $Nds_{0,0}$	$PValue_{0,0}$ += $Mds_{0,0}$*$Nds_{0,0}$ + $Mds_{0,1}$*$Nds_{1,0}$
$thread_{0,1}$	**$M_{0,1}$** ↓ $Mds_{0,1}$	**$N_{0,1}$** ↓ $Nds_{1,0}$	$PValue_{0,1}$ += $Mds_{0,0}$*$Nds_{0,1}$ + $Mds_{0,1}$*$Nds_{1,1}$	**$M_{0,3}$** ↓ $Mds_{0,1}$	**$N_{2,1}$** ↓ $Nds_{0,1}$	$PValue_{0,1}$ += $Mds_{0,0}$*$Nds_{0,1}$ + $Mds_{0,1}$*$Nds_{1,1}$
$thread_{1,0}$	**$M_{1,0}$** ↓ $Mds_{1,0}$	**$N_{1,0}$** ↓ $Nds_{1,0}$	$PValue_{1,0}$ += $Mds_{1,0}$*$Nds_{0,0}$ + $Mds_{1,1}$*$Nds_{1,0}$	**$M_{1,2}$** ↓ $Mds_{1,0}$	**$N_{3,0}$** ↓ $Nds_{1,0}$	$PValue_{1,0}$ += $Mds_{1,0}$*$Nds_{0,0}$ + $Mds_{1,1}$*$Nds_{1,0}$
$thread_{1,1}$	**$M_{1,1}$** ↓ $Mds_{1,1}$	**$N_{1,1}$** ↓ $Nds_{1,1}$	$PValue_{1,1}$ += $Mds_{1,0}$*$Nds_{0,1}$ + $Mds_{1,1}$*$Nds_{1,1}$	**$M_{1,3}$** ↓ $Mds_{1,1}$	**$N_{3,1}$** ↓ $Nds_{1,1}$	$PValue_{1,1}$+= $Mds_{1,0}$*$Nds_{0,1}$ + $Mds_{1,1}$*$Nds_{1,1}$

time →

A Tiled Matrix-Matrix Multiplication Kernel (3/3)

```
__global__ void MatrixMulKernel(float* d_M, float* d_N, float* d_P,
    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 d_P 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 d_M and d_N tiles required to compute d_P element
8.   for (int m = 0; m < Width/TILE_WIDTH; ++m) {

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

12.    for (int k = 0; k < TILE_WIDTH; ++k) {
13.      Pvalue += Mds[ty][k] * Nds[k][tx];
       }
14.    __syncthreads();
     }
15.  d_P[Row*Width + Col] = Pvalue;
   }
```

THANKS

john@sirius.utp.edu.co