# Parallel Computing Workshop

Ms. Kranti Ingale

Prime Minister's Research Fellows (**PMRF**) Scholar

CSE, IIT Madras

# Roadmap

- Introduction to GPU
- CUDA Program Flow and CPU-GPU Communication
- Thread organization (Grids, Blocks, Threads, 1D/2D)
    - WARP & Thread Divergence
- CUDA Memory Model
- CUDA Functions
- CUDA Thrust

# Roadmap

- Introduction to GPU
- CUDA Program Flow and CPU-GPU Communication
- Thread organization (Grids, Blocks, Threads, 1D/2D)
  - WARP & Thread Divergence
- CUDA Memory Model
- CUDA Functions
- CUDA Thrust

# Recap : Standard Variables

- Dimension of Grid (Number of blocks in the grid)
  gridDim.x , gridDim.y , gridDim.z

- Dimension of block (Number of threads in a block)
  blockDim.x , blockDim.y , blockDim.z

# Recap : Standard Variables

- Block Index

  blockIdx.x , blockIdx.y,blockIdx.z

- Thread Index

  threadIdx.x , threadIdx.y, threadIdx.z

# Recap : Example

- dim3 nblock(3,4,1) ;
- dim3 threadPerBlock(5,4,1) ;

- dkernel<<<nblock, threadPerBlock>>>();

- dkernel<<<(**3,4,1**), (**5,4,1**)>>>();

# Recap : Example

- dkernel<<<(**3,4,1**), (**5,4,1**)>>>();
- gridDim.x = 3 ; gridDim.y = 4 ; gridDim.z = 1
- blockDim.x = 5 ; blockDim.y = 4 ; blockDim.z = 1
- blockIdx.x = 0..2 ; blockIdx.y = 0..3 ; blockIdx.z = 0
- threadIdx.x = 0..4 ; threadIdx.y = 0..3 ; threadIdx.z = 0

# Matrix Multiplication

**Matrix 1**

$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{pmatrix}$$

**Matrix 2**

$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{pmatrix}$$

**Matrix 1 * Matrix 2**

$$\begin{pmatrix} 1*1+1*2+1*3 & 1*1+1*2+1*3 & 1*1+1*2+1*3 \\ 2*1+2*2+2*3 & 2*1+2*2+2*3 & 2*1+2*2+2*3 \\ 3*1+3*2+3*3 & 3*1+3*2+3*3 & 3*1+3*2+3*3 \end{pmatrix}$$

=

**Matrix 1 * Matrix 2**

$$\begin{pmatrix} 6 & 6 & 6 \\ 12 & 12 & 12 \\ 18 & 18 & 18 \end{pmatrix}$$

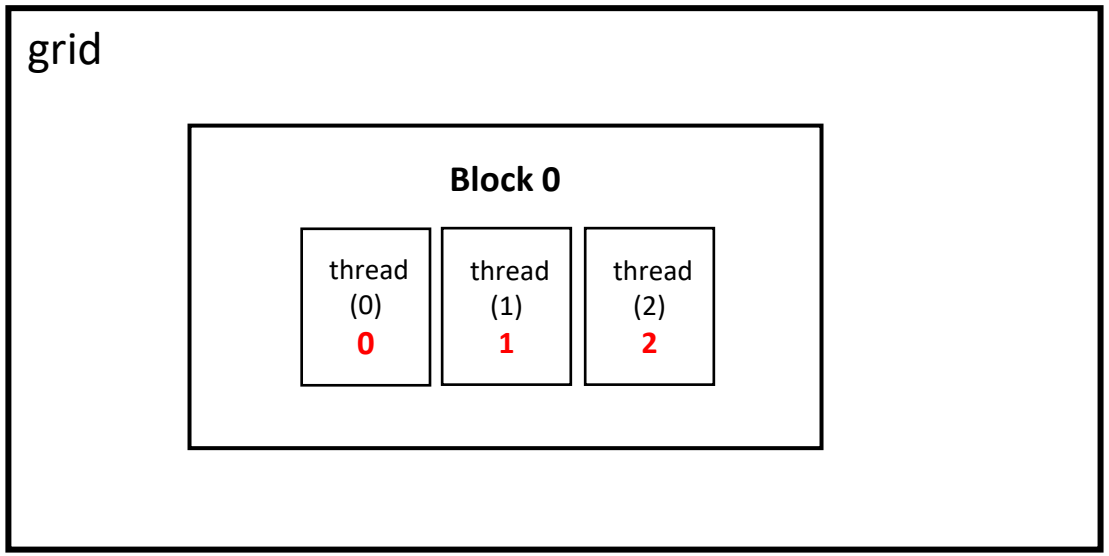
# Matrix Multiplication

```
void Mult(unsigned *matrix1,Unsigned *matrix2, unsigned *result, unsigned n)
{
for (unsigned i = 0; i < n; i++) {
for (unsigned j = 0; j < n;j++)  {

    for (unsigned k = 0; k < n; k++)  {
            result[i * n + j] +=  matrix[i * n + k] * matrix[k * n + j] ;
} } } }
```

# Matrix Multiplication

```
__global__ Mult(unsigned *matrix1,Unsigned *matrix2, unsigned *result, unsigned n)
{
int id = blockDim.x * blockIdx.x + threadIdx.x;

for (unsigned j = 0; j < n;j++)  {

    for (unsigned k = 0; k < n; k++)  {
            result[id * n + j] +=  matrix1[id * n + k] * matrix2[k * n + j] ;
} } } }


Mult<<<1,n>>>(A,B,C,n);
```

# Matrix Multiplication

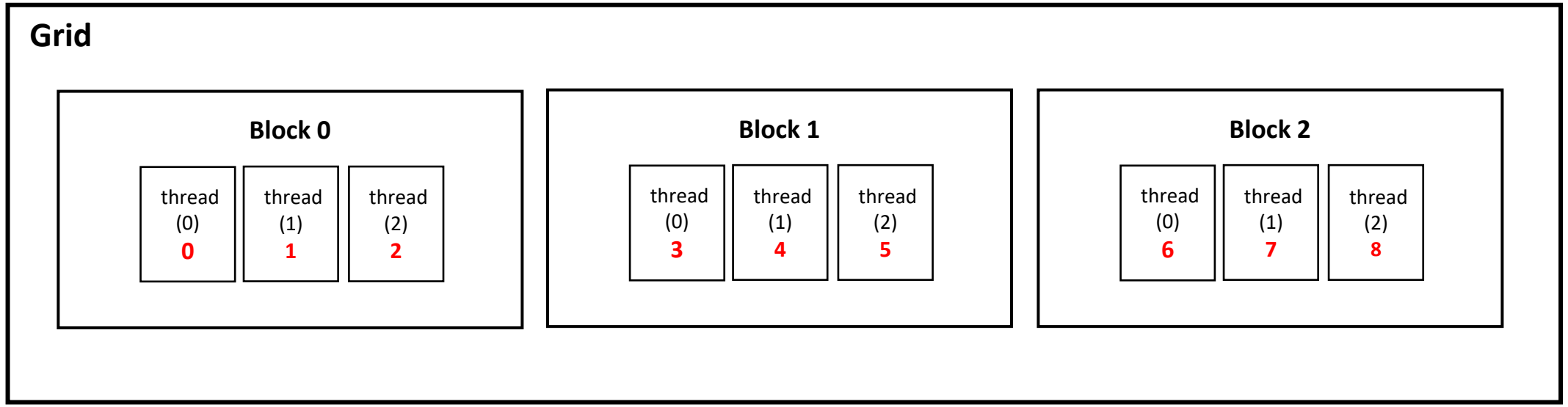



**Mult<<<1,n>>>(A,B,C,n);  // n=3**

# Matrix Multiplication

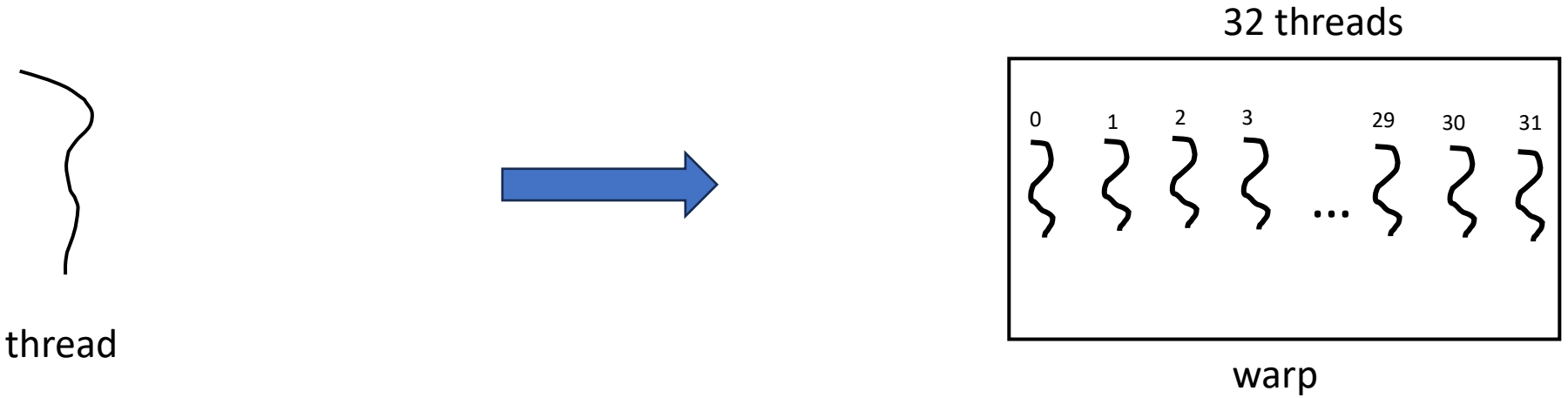
```
__global__ Mult(unsigned *matrix1,Unsigned *matrix2, unsigned *result, unsigned n)
{
int id = blockDim.x * blockIdx.x + threadIdx.x;
int i= id / n;
int j= id % n;

for (unsigned k = 0; k < n; k++)  {
            result[id * n + j] +=  matrix1[id * n + k] * matrix2[k * n + j] ;
} } } }


Mult<<<n,n>>>(A,B,C,n);
```

# Matrix Multiplication

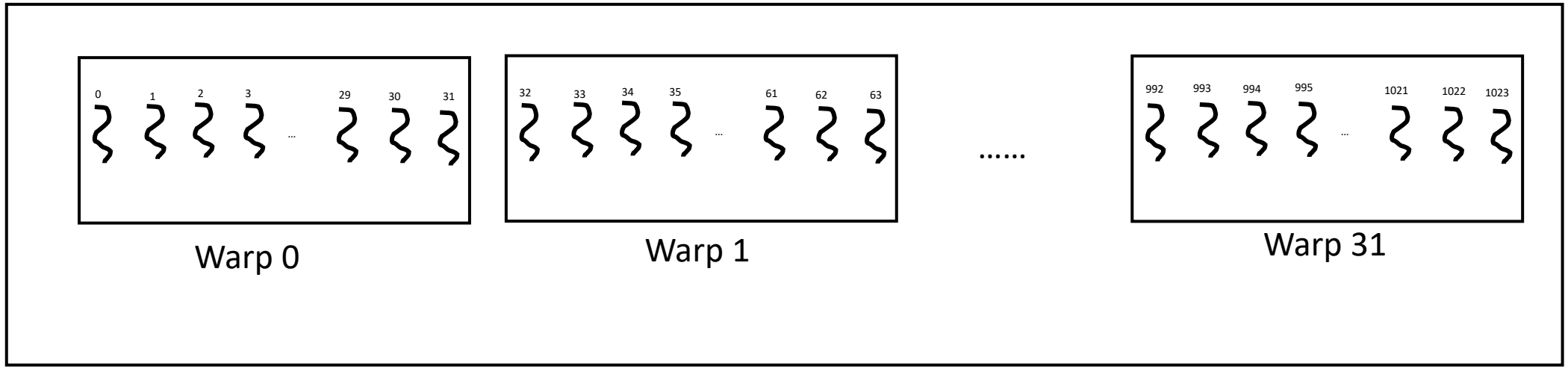



**Mult<<<n,n>>>(A,B,C,n); // n=3**

# COMPUTATIONAL HIERARCHY

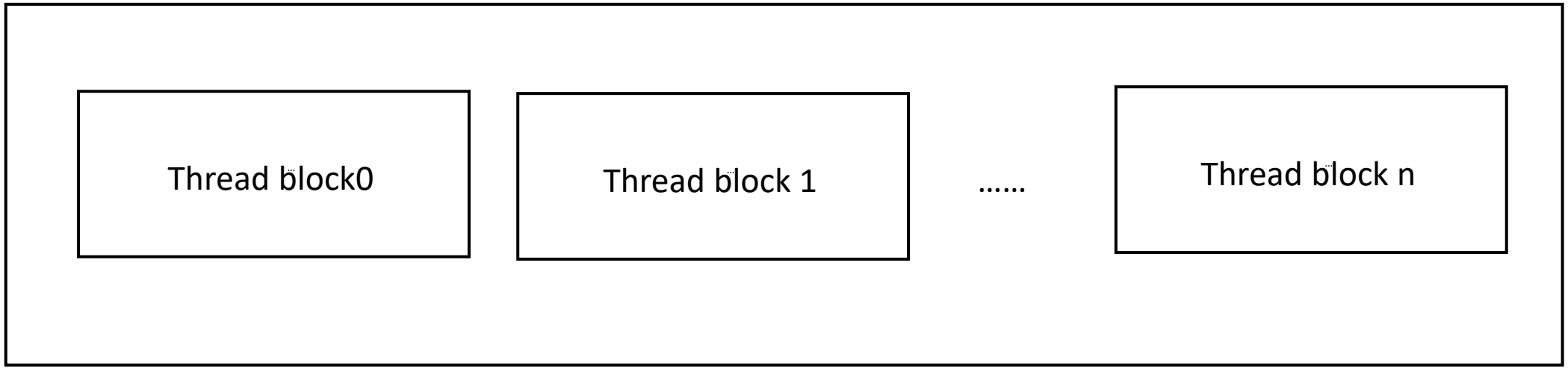


# COMPUTATIONAL HIERARCHY



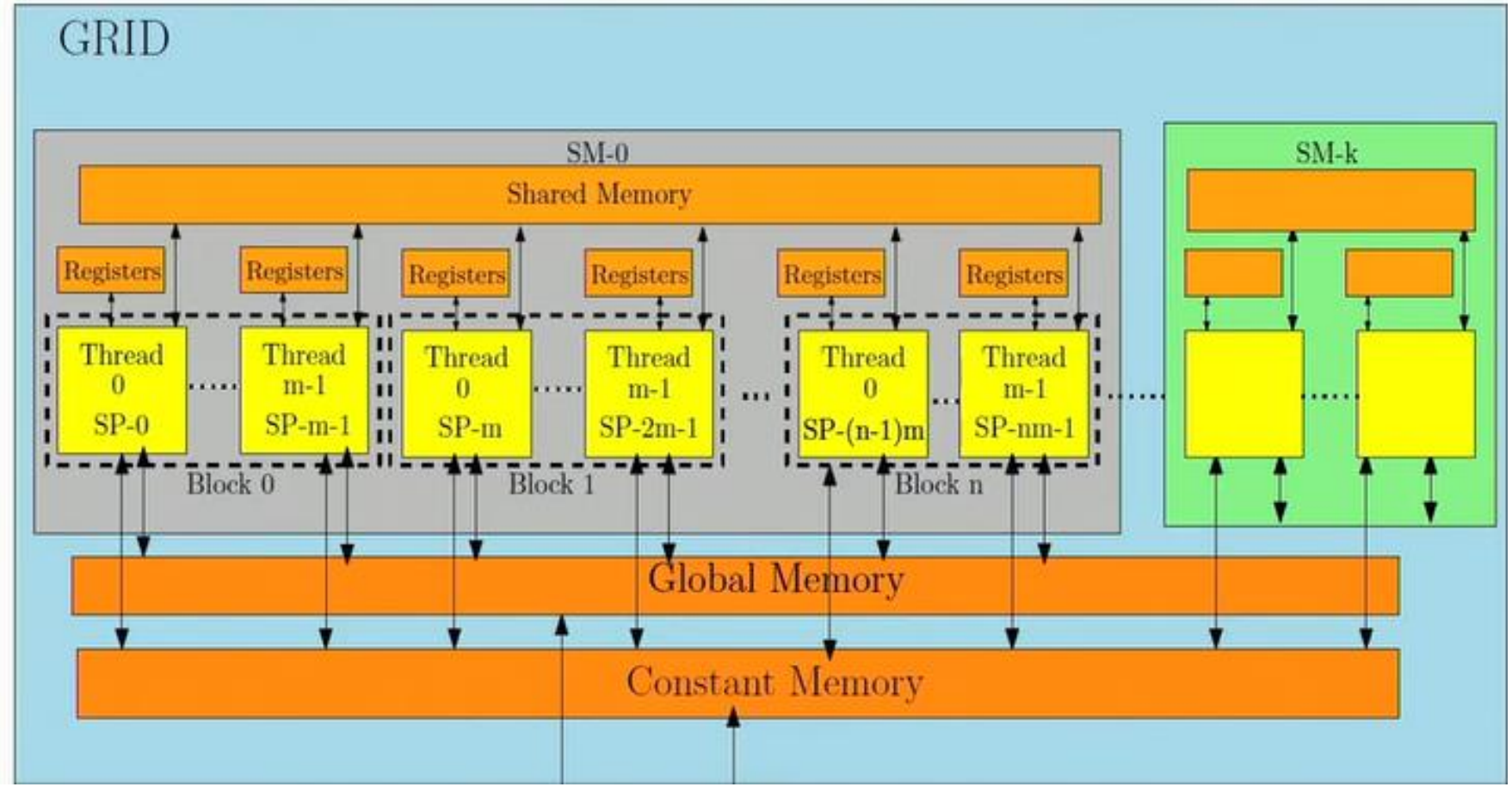

1 Thread Block = 1024 threads

# COMPUTATIONAL HIERARCHY




~ 1-5 blocks per SM

# Thread mapping to Hardware




~ 1-5 blocks per SM

# Example

- Grid < 6,2,1>  Block = <5,4,1>
- Number of SM's = 6
- SP's per SM = 40

# Example

- Grid < 6,2,1>  Block = <5,4,1>
- Number of SM's = 6
- SP's per SM = 40
- Number of thread block = 6 * 2 = 12

# Example

- Grid < 6,2,1>  Block = <5,4,1>
- Number of SM's = 6
- SP's per SM = 40
- Number of thread block = 6 * 2 = 12
- Threads per thread_block = 5 * 4 = 20
- Two thread block are mapped to one SM

# Thread mapping to Hardware