

# 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

# WARP

- A set of consecutive threads (32 thread) that execute in **Single Instruction, Multiple Data (SIMD)** fashion

# WARP

- A set of consecutive threads (32 thread) that execute in **Single Instruction, Multiple Data (SIMD)** fashion
- Warp threads are fully synchronized there is an implicit barrier after each instruction

# SIMD

```
__global__ void Mykernel()
{
    Printf("Thread id = %d" ,
threadIdx.x);
}
int main() {
    Mykernel<<<1,33>>>();
    CDS();
    return 0;
}
```

# SIMD

```
__global__ void Mykernel()
{
    printf("Thread id = %d",
    threadIdx.x);
}

int main() {
    Mykernel<<<1,33>>>();
    CDS();
    return 0;
}
```

## Output :

Thread id = 32  
Thread id = 0  
Thread id = 1  
...  
Thread id = 31

# SIMD

```
__global__ void Mykernel()
{
    Printf("Thread id = %d" ,
    threadIdx.x);
}

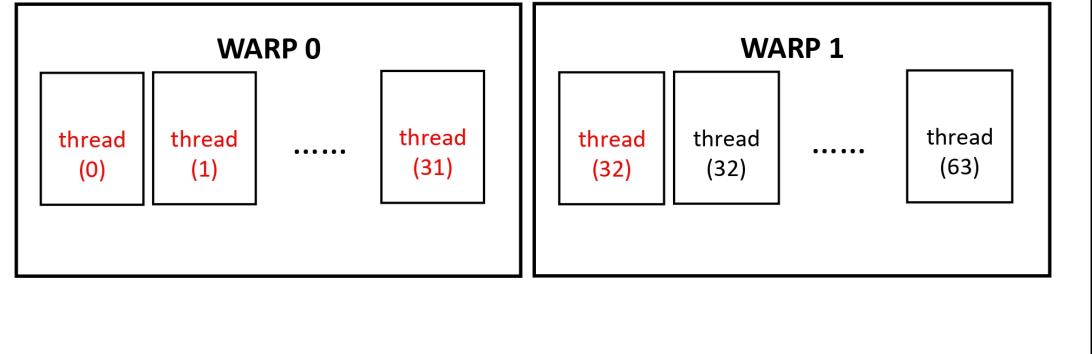
int main() {
    Mykernel<<<1,33>>>();
    CDS();
    return 0;
}
```

# SIMD

```
__global__ void Mykernel()  
{  
    printf("Thread id = %d",  
        threadIdx.x);  
}  
  
int main() {  
    Mykernel<<<1,33>>>();  
    CDS();  
    return 0;  
}
```

Grid

Block 0



Thread id's are unique per thread blocks.

# SIMD

```
__global__ void Mykernel()  
{  
    printf("Thread id = %d",  
        threadIdx.x);  
}  
  
int main() {  
    Mykernel<<<1,33>>>();  
    CDS();  
    return 0;  
}
```

Output :

Thread id = 32 → WARP 1

Thread id = 0

Thread id = 1

...

Thread id = 31

# SIMD

```
__global__ void Mykernel()  
{  
    printf("Thread id = %d",  
        threadIdx.x);  
}  
  
int main() {  
    Mykernel<<<1,33>>>();  
    CDS();  
    return 0;  
}
```

Output :

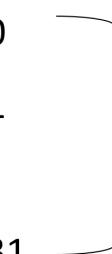
Thread id = 32 → WARP 1

Thread id = 0

Thread id = 1

...

Thread id = 31



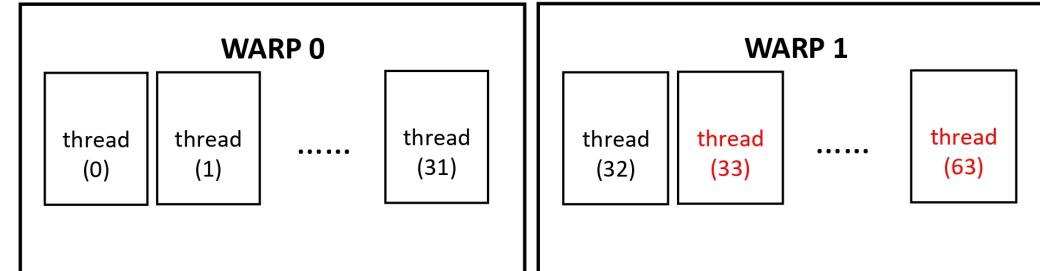
WARP 0

WARP 1

# SIMD

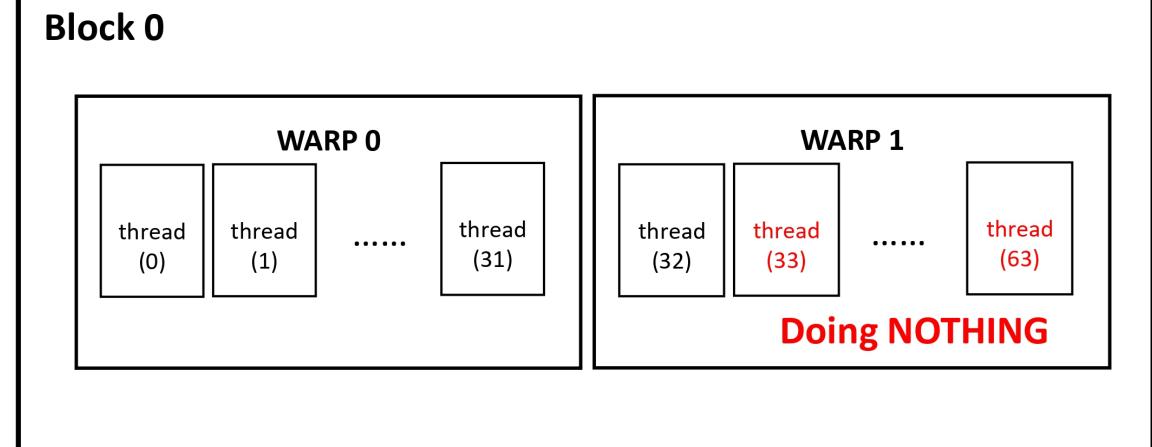
```
__global__ void Mykernel()  
{  
    Printf("Thread id = %d",  
        threadIdx.x);  
}  
int main() {  
    Mykernel<<<1,33>>>();  
    CDS();  
    return 0;  
}
```

Block 0



# SIMD

```
__global__ void Mykernel()  
{  
    Printf("Thread id = %d",  
        threadIdx.x);  
}  
  
int main() {  
    Mykernel<<<1,33>>>();  
    CDS();  
    return 0;  
}
```

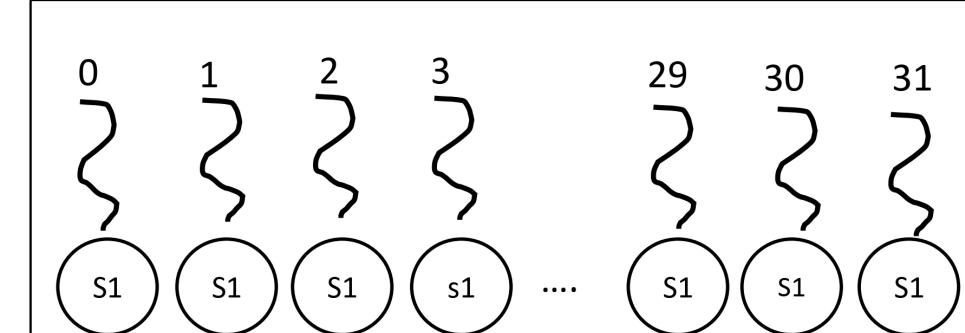


# Thread-Divergence

```
__global__ void Mykernel (int *arr )
{
    unsigned id = blockIdx.x * blockDim.x + threadIdx.x;
    If(id%2==0) arr[id]=0;
    else arr[id]=1;
    printf(" From gpu side ");
}
Mykernel<<<1,32>>>(arr);
```

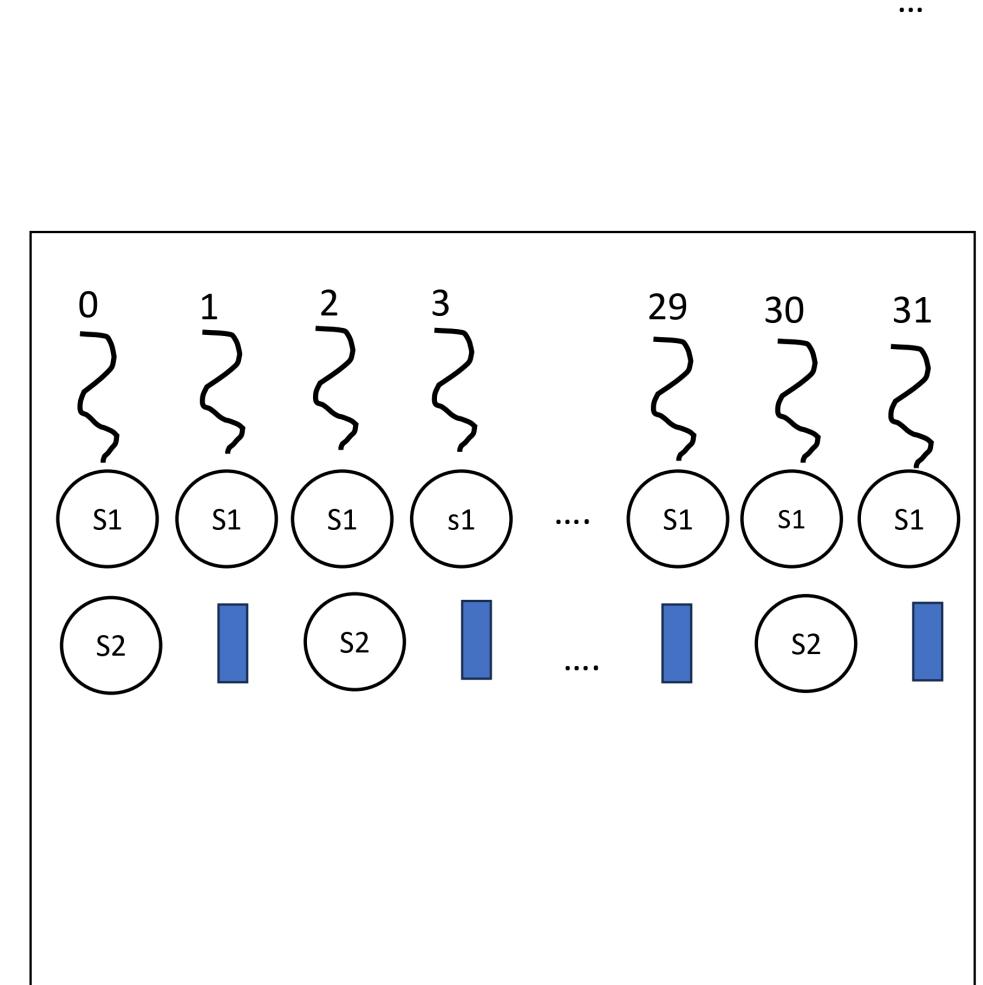
# Thread-Divergence

```
__global__ void Mykernel (int *arr )  
{  
unsigned id = blockIdx.x * blockDim.x + threadIdx.x;  
If(id%2==0) arr[id]=0;  
else arr[id]=1;  
printf(" From gpu side ");  
}  
Mykernel<<<1,32>>>(arr);
```



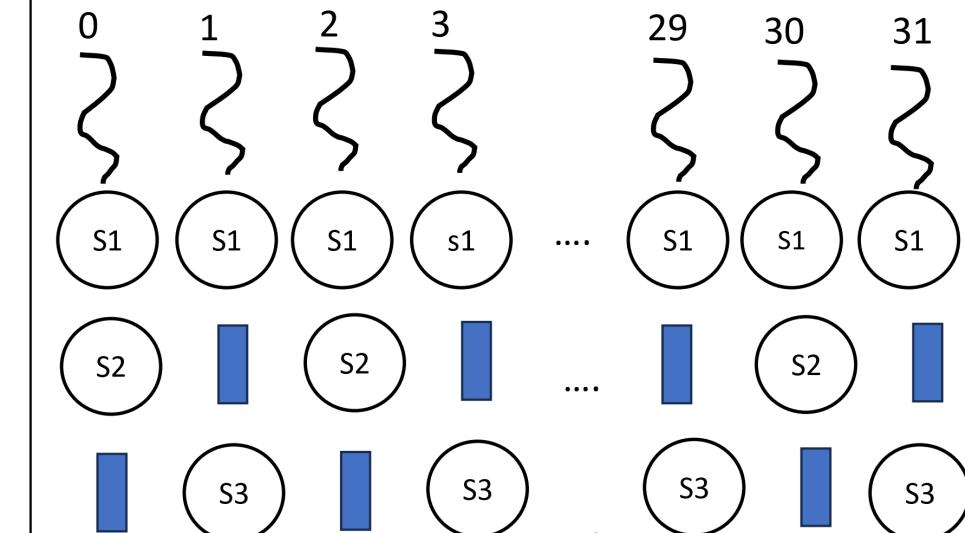
# Thread-Divergence

```
__global__ void Mykernel (int *arr )  
{  
    unsigned id = blockIdx.x * blockDim.x + threadIdx.x;  
If(id%2==0) arr[id]=0;  
    else arr[id]=1;  
    printf(" From gpu side ");  
}  
Mykernel<<<1,32>>>(arr);
```



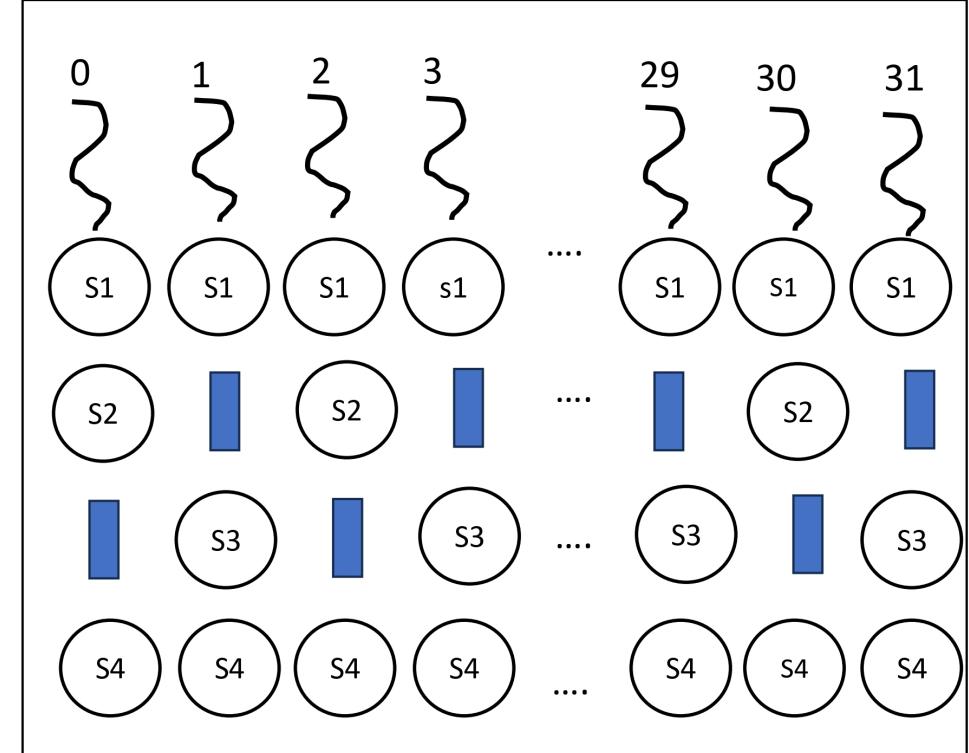
# Thread-Divergence

```
__global__ void Mykernel (int *arr )  
{  
    unsigned id = blockIdx.x * blockDim.x + threadIdx.x;  
    If(id%2==0) arr[id]=0;  
else arr[id]=1;  
    printf(" From gpu side ");  
}  
Mykernel<<<1,32>>>(arr);
```



# Thread-Divergence

```
__global__ void Mykernel (int *arr )  
{  
    unsigned id = blockIdx.x * blockDim.x + threadIdx.x;  
    If(id%2==0) arr[id]=0;  
    else arr[id]=1;  
printf(" From gpu side ");  
}  
Mykernel<<<1,32>>>(arr);
```



# Thread-Divergence

- Warps are the smallest execution units in GPUs.

# Thread-Divergence

- Warps are the smallest execution units in GPUs.
- In GPU warps, all threads within a warp must execute the same instruction simultaneously.

# Thread-Divergence

- Warps are the smallest execution units in GPUs.
- In GPU warps, all threads within a warp must execute the same instruction simultaneously.
- Warp divergence occurs when threads within a warp follow different instruction sequences, often due to branch instructions.

# Thread-Divergence

- Warps are the smallest execution units in GPUs.
- In GPU warps, all threads within a warp must execute the same instruction simultaneously.
- Warp divergence occurs when threads within a warp follow different instruction sequences, often due to branch instructions.
- This adds sequentially to the execution.

# Is there is Thread-Divergence here ?

```
__global__ void Mykernel() {  
    int id = unsigned id = blockIdx.x * blockDim.x + threadIdx.x;;  
    switch(id) {  
        case 0: ... ;      thread 0 of WARP-X & Remaining threads NOP  
        case 1: ... ;  
        .  
        case n: .... ;  
    }  
}
```

# Is there is Thread-Divergence here ?

```
__global__ void Mykernel() {  
    int id = unsigned id = blockIdx.x * blockDim.x + threadIdx.x;;  
    switch(id) {  
        case 0: ... ;  
        case 1: ... ; thread 1 of WARP-X & Remaining threads NOP  
        .  
        case n: .... ;  
    }  
}
```

# Is there is Thread-Divergence here ?

```
__global__ void Mykernel() {  
    int id = unsigned id = blockIdx.x * blockDim.x + threadIdx.x;;  
    switch(id) {  
        case 0: ... ;  
        case 1: ... ;  
        .  
        case n: .... ;  thread n of WARP-X & Remaining threads NOP  
    }  
}
```

# Is there is Thread-Divergence here ?

```
__global__ void Mykernel() {  
    int id = unsigned id = blockIdx.x * blockDim.x + threadIdx.x;;  
    switch(id) {  
        case 0: ... ;  
        case 1: ... ;  
        .  
        case n: .... ;  thread n of WARP-X & Remaining threads NOP  
    }  
}
```

**sequentially** to the execution

# Is there is thread divergence here ?

```
__global__ void Mykernel()  
{  
    id = ... ;  
    if(id/32) {  
        printf(" if ");  
    } else printf("else");  
}
```

# Is there is thread divergence here ?

```
__global__ void Mykernel()  
{  
    id = ... ;  
    if(id/32) {  
        printf(" if ");  
    } else printf("else");  
}
```

**Output :**

**No ,** because we are getting same  
truth values for the warp

# Is there is thread divergence here ?

```
__global__ void Mykernel ()  
{  
    if(size < N){  
        printf("%d",threadIdx.x);  
    } else printf("else");  
}
```

# Is there is thread divergence here ?

```
__global__ void Mykernel ()  
{  
if(size < N){  
printf("if");  
} else printf("else");  
}
```

**Output :**

**No ,** because we are getting same  
truth values for the warps.

# Is there is thread divergence here ?

```
__global__ void Mykernel()
{
    id = ... ;
    if(id/2) {
        printf("if");
    }else printf("else");
}
```

# Is there is thread divergence here ?

```
__global__ void Mykernel()  
{  
    id = ... ;  
    if(id/2) {  
        printf("if");  
    }else printf("else");  
}
```

## Output :

**Yes ,** because we are getting different truths for different threads of warps

# Can we remove thread divergence ?

```
if( A == B )  
A = C + 10;  
else  
A = B + 10;
```

**Assumption (A==B | A==C)**

# Can we remove thread divergence ?

```
if( A == B )  
A = C + 10;  
else  
A = B + 10;
```

**Assumption (A==B | A==C)**

**Output :**

A = B + C + 10 - A

# Can we remove thread divergence ?

```
if( A == B )  
A = C + W;  
else if (A == C)  
A = W + B;  
else  
A= B + C ;
```

**Assumption (A==B || A==W || A==C)**

# Can we remove thread divergence ?

```
if( A == B )  
A = C + W;  
else if (A == C)  
A = W + B;  
else  
A= B + C ;
```

**Assumption (A==B || A==W || A==C)**

**Output :**

A = B + C + W - A

# Can we remove thread divergence ?

```
if((tx & 1) == 0) {  
    tx_index = tx + 1;  
} else  
    tx_index = tx - 1;
```

# Can we remove thread divergence ?

```
if((tx & 1) == 0) {  
    tx_index = tx + 1;  
} else  
    tx_index = tx - 1;
```

**Output :**

```
int tx_index = tx ^ 1;
```

# Can we remove thread divergence ?

```
if((tx &1)== 0) {  
    A[tx+1] = B[tx] * A[tx+1];  
} else  
    A[tx-1] = B[tx] * A[tx-1];
```

# Can we remove thread divergence ?

```
if((tx &1)== 0) {  
    A[tx+1] = B[tx] * A[tx+1];  
} else  
    A[tx-1] = B[tx] * A[tx-1];
```

## Output :

```
int tx_index = tx ^ 1 ;  
A[tx_index] = B[tx] * A[tx_index];
```

# Can we remove thread divergence ?

```
if( id<=0)
    arr[id] = 0;
else
    arr[id] = 1;

dkernel<<<1,32>>>()
```

# Can we remove thread divergence ?

```
if( id<=0)
arr[id] = 0;
else
arr[id] = 1;

dkernel<<<1,32>>>()
```

## Output :

Id = 0 or positive number(1..31)  
arr[id] = [1+(-id>>31)] \* -1 +1  
ID=1,2...31  
1....0001>>31 → 111...11111  
-1+1 → 0  
→1

# Thank You