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)
- 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)
- CUDA Memory Model
- CUDA Functions
- CUDA Thrust

Shared Memory / L1 cache

- **Terminology: within a block, threads share data via shared memory**
- Extremely fast on-chip memory, user-managed
- Declare using __shared__ allocated per block
- Data is not visible to threads in other blocks

- Eg.

```
__shared__ unsigned a;
__shared__ unsigned a[N];
```

Example

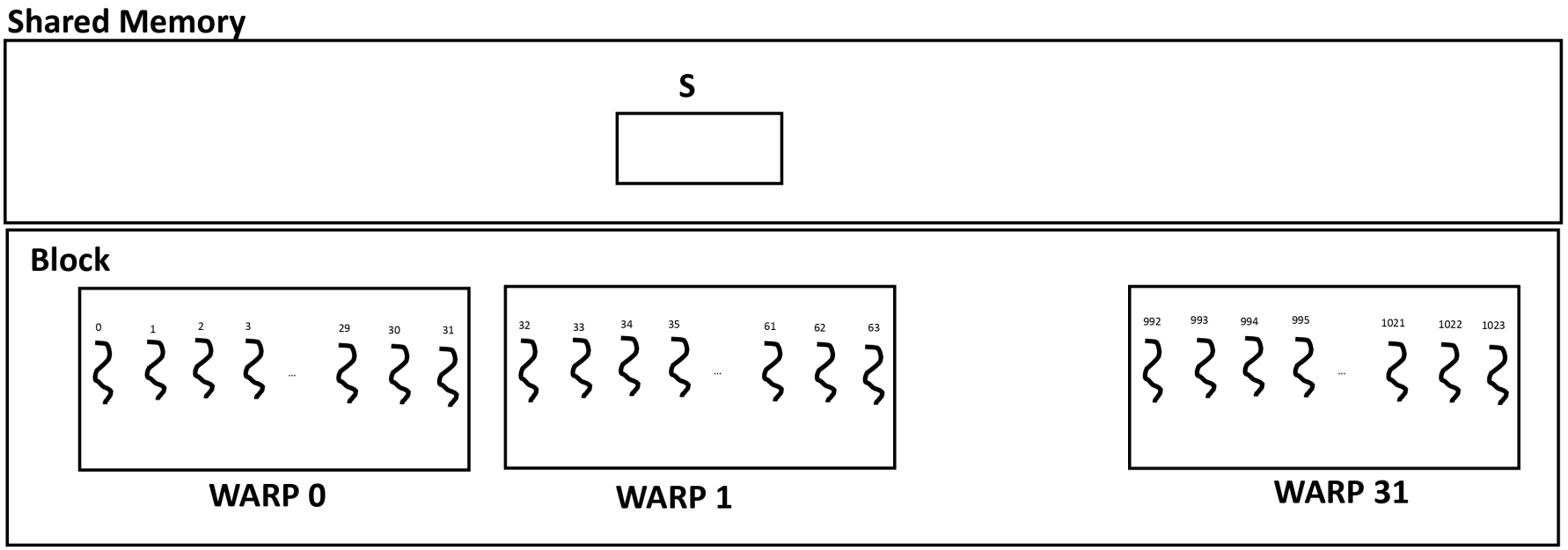
```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;
if (id == 1) s += 2;
if (id == 0) printf("\n S = %d", s);
}
int main() {
Mykernel<<<1, BLOCKSIZE>>>();
cudaDeviceSynchronize(); }
```

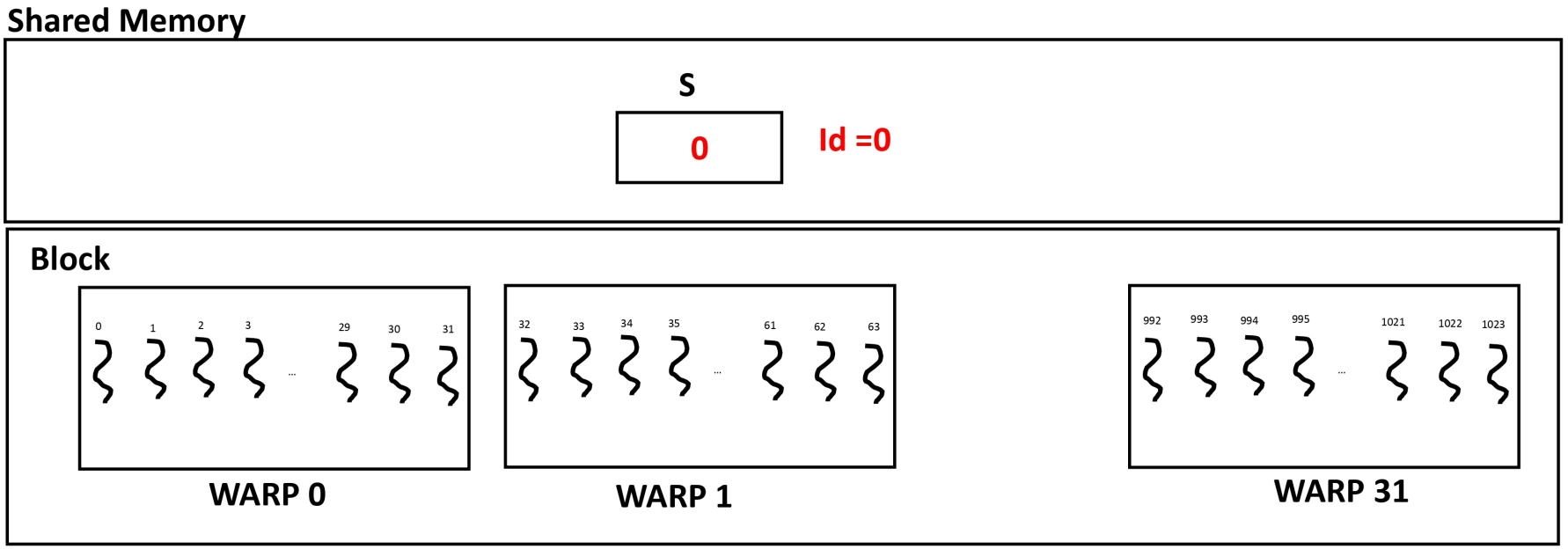
Example

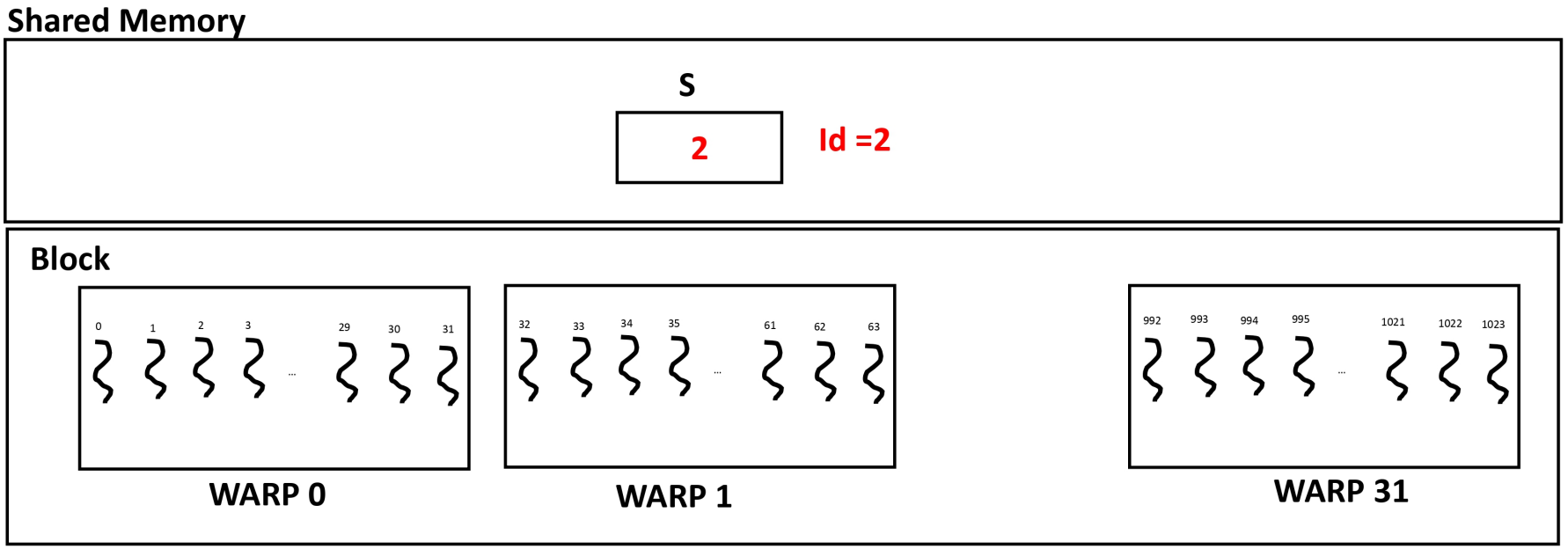
```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;
if (id == 1) s += 2;
if (id == 0) printf("\n S = %d", s);
}
int main() {
Mykernel<<<1, BLOCKSIZE>>>();
cudaDeviceSynchronize(); }
```

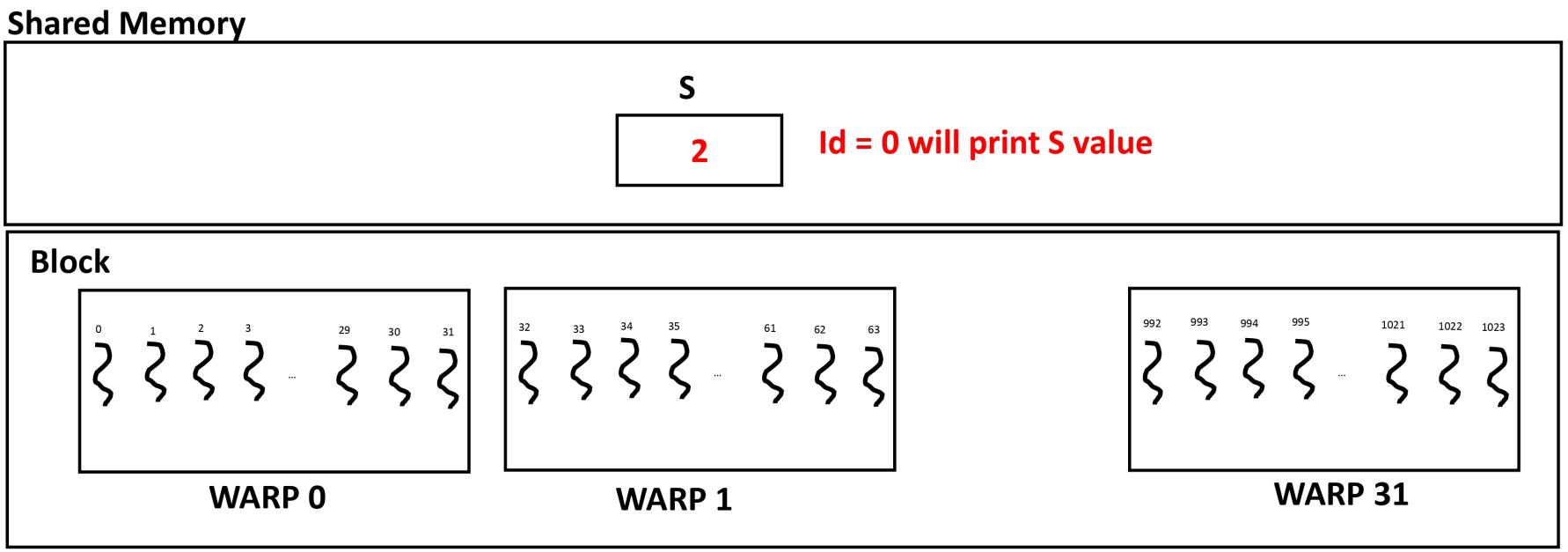
Output :

2


Shared Memory
S
Block
0
1
2
3
29
30
31
32
33
34
35
61
62
63
992
993
994
995
1021
1022
1023
WARP 0
WARP 1
WARP 31


Shared Memory
S
0
Id =0
Block
0 1 2 3 ... 29 30 31
32 33 34 35 ... 61 62 63
992 993 994 995 ... 1021 1022 1023
WARP 0
WARP 1
WARP 31


Shared Memory
S
2
Id =2
Block
0
1
2
3
29
30
31
32
33
34
35
61
62
63
992
993
994
995
1021
1022
1023
WARP 0
WARP 1
WARP 31


Shared Memory
S
2
Id = 0 will print S value
Block
0 1 2 3 ... 29 30 31
32 33 34 35 ... 61 62 63
992 993 994 995 ... 1021 1022 1023
WARP 0
WARP 1
WARP 31

Example-1

```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;
if (id ==62) s += 2;
if (id == 0) printf(“\n S = %d", s);
}
int main() {
Mykernel<<<1, BLOCKSIZE>>>();
cudaDeviceSynchronize(); }
```

Example-1

```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;
if (id ==62) s += 2;
if (id == 0) printf("\n S = %d", s);
}
int main() {
Mykernel<<<1, BLOCKSIZE>>>();
cudaDeviceSynchronize(); }
```

S1

Example-1

```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;
if (id ==62) s += 2;
if (id == 0) printf("\n S = %d", s);
}
int main() {
Mykernel<<<1, BLOCKSIZE>>>();
cudaDeviceSynchronize(); }
```

S1

S2

Example-1

```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;
if (id ==62) s += 2;
if (id == 0) printf("\n S = %d", s);
}
int main() {
Mykernel<<<1, BLOCKSIZE>>>();
cudaDeviceSynchronize(); }
```

S1

S2

S3

Example-1

```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;
if (id ==62) s += 2;
if (id == 0) printf("\n S = %d", s);
}
int main() {
Mykernel<<<1, BLOCKSIZE>>>();
cudaDeviceSynchronize(); }
```

S1

S2

S3

Output :

2

S1 → S2 → S3

Example-1

```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;
if (id ==62) s += 2;
if (id == 0) printf("\n S = %d", s);
}
int main() {
Mykernel<<<1, BLOCKSIZE>>>();
cudaDeviceSynchronize(); }
```

S1

S2

S3

Output :

0

S2 → S1 → S3

Example-1

```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;        ⟶ S1
if (id ==62) s += 2;       ⟶ S2
if (id == 0) printf("\n S = %d", s);  ⟶ S3
}
int main() {
Mykernel<<<1, BLOCKSIZE>>>();
cudaDeviceSynchronize(); }
```

Output :

Garbage value

Data race for S variable between S1 and S2 statement

Example-2

```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;
if (id ==1000) s = s + 2;
if (id == 9) s = s + 1 ;
if (id == 0) printf("\n S = %d", s);  }
int main() {
Mykernel<<<1, BLOCKSIZE>>>();
cudaDeviceSynchronize(); }
```

Example-2

```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;
if (id ==1000) s = s + 2;
if (id == 9) s = s + 1 ;
if (id == 0) printf("\n S = %d", s);  }
int main() {
Mykernel<<<1, BLOCKSIZE>>>();
cudaDeviceSynchronize(); }
```

S1

S2

S3

S4

Example-2

```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;                          S1
if (id ==1000) s = s + 2;                    S2
if (id == 9) s = s + 1 ;                     S3
if (id == 0) printf("\n S = %d", s);  }      S4
int main() {
Mykernel<<<1, BLOCKSIZE>>>();
cudaDeviceSynchronize(); }
```

Output :

3

S1 → S2 → S3 → S4

Example-2

```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;                          S1
if (id ==1000) s = s + 2;                    S2
if (id == 9) s = s + 1 ;                     S3
if (id == 0) printf("\n S = %d", s);  }      S4
int main() {
Mykernel<<<1, BLOCKSIZE>>>();
cudaDeviceSynchronize(); }
```

Output :

3

S1 → S2 → S3 → S4

S1 → S3 → S2 → S4

Example-2

```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;                          S1
if (id ==1000) s = s + 2;                    S2
if (id == 9) s = s + 1 ;                     S3
if (id == 0) printf("\n S = %d", s);  }      S4
int main() {
Mykernel<<<1, BLOCKSIZE>>>();
cudaDeviceSynchronize(); }
```

Output :

1

S2 → S1 → S3 → S4

Example-2

```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;                    → S1
if (id ==1000) s = s + 2;              → S2
if (id == 9) s = s + 1 ;               → S3
if (id == 0) printf("\n S = %d", s);  }  → S4
int main() {
Mykernel<<<1, BLOCKSIZE>>>();
cudaDeviceSynchronize(); }
```

Output :

1

S2 → S1 → S3 → S4

S1 → S3 → S4 → S2

Example-2

```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;  ⟶ S1
if (id ==1000) s = s + 2;  ⟶ S2
if (id == 9) s = s + 1 ;  ⟶ S3
if (id == 0) printf("\n S = %d", s);  }  ⟶ S4
int main() {
Mykernel<<<1, BLOCKSIZE>>>();
cudaDeviceSynchronize(); }
```

Output :

2

After S1 statement , Data race for S2 and S3. S3 updates S=1 but then S2 overwrite it with S=2.

Example-2

```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;          ⟶ S1
if (id ==1000) s = s + 2;    ⟶ S2
if (id == 9) s = s + 1 ;     ⟶ S3
if (id == 0) printf("\n S = %d", s);  }  ⟶ S4
int main() {
Mykernel<<<1, BLOCKSIZE>>>();
cudaDeviceSynchronize(); }
```

Output :

Garbage value

Data race for S1 and S2. S1 updates S=0 but then S2 overwrite it with S= garbage.

Example-3

```
__global__ void Mykernel() {
  __shared__ unsigned s;
  int id= threadIdx.x;
  if(blockIdx.x ==0) {
      if (id == 1) s = 9;
      if (id == 0) printf("\n block id = %d S = %d",blockIdx.x,s); }
  else {
      if(id==25) s = 4;
      if (id == 0) printf("\n block id = %d S = %d ",blockIdx.x,s);}
}
int main() { Mykernel<<<2, BLOCKSIZE>>>(); CDS(); }
```

Output :

block id = 1 s=4

block id = 0 s=9

Example-4

```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;
if (id ==1) s = s + 2;
if (id == 9) s = s + 1 ;
if (id == 0) printf("\n S = %d", s);  }
int main() {
Mykernel<<<2, BLOCKSIZE>>>();
cudaDeviceSynchronize(); }
```

Deterministic output

```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;
if (id ==1000) s = s + 2;
if (id == 9) s = s + 1 ;
if (id == 0) printf("\n S = %d", s);  }
int main() {
Mykernel<<<2, BLOCKSIZE>>>();
cudaDeviceSynchronize(); }
```

NO Deterministic output

__syncthreads()

- To coordinate the execution of multiple threads, CUDA allows threads in the same block to coordinate their activities by using a barrier synchronization function **__syncthreads()**
- All threads in a block will be blocked at the calling location until each of them reaches the location.
- Threads in different blocks **cannot** synchronize.
- CUDA runtime system can execute blocks in any order.

Example-5

```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;
__syncthreads ();
if (id ==62) s = s + 2;
__syncthreads();
if (id == 0) printf(“\n S = %d", s);
}
int main() { Mykernel<<<1, BLOCKSIZE>>>(); CDS(); }
```

Example-5

```
#define BLOCKSIZE 1024
__global__ void Mykernel() {
__shared__ unsigned s;
int id= threadIdx.x;
if (id == 0) s = 0;
__syncthreads ();
if (id ==62) s = s + 2;
__syncthreads();
if (id == 0) printf(“\n S = %d", s);
}
int main() { Mykernel<<<1, BLOCKSIZE>>>(); CDS(); }
```

Output :

2

Problem

```
If (condition) {
  ..
__syncthreads();
 ..
}
```

```
if (condition) {
  ..
__syncthreads();
 ..
}
else {
..
 __syncthreads();
..
}
```

Problem

```
If (condition) {
  ..
__syncthreads();
 ..
}
```

DEADLOCK

```
If (condition) {
  ..
__syncthreads();
 ..
}
else {
..
 __syncthreads();
..
}
```

Solution to avoid deadlock

```
if (condition) {
  ..
}
else {
..

}
 __syncthreads();
```

Dynamically Allocated Shared Memory

- It becomes useful when the required amount of shared memory is not known during compile-time.

- This is specified as the third parameter of kernel launch.

- **Syntax** :

  kernel_name<<<numBlocks, threadsPerBlock, size in byte>>>();

  extern __shared__ <variable Name> ;

Example-6

```
__global__ void dynshared() {
extern __shared__ int A[];
A[threadIdx.x] = threadIdx.x;
__syncthreads();
if (threadIdx.x % 2 == 0) printf("%d\n", A[threadIdx.x]);
}
int main() {
int n;
scanf(“%d”, &n);
dynshared<<<1, n, n * sizeof(int)>>>();
CDS(); }
```

Example-6

```
__global__ void dynshared() {
extern __shared__ int A[];
A[threadIdx.x] = threadIdx.x;
__syncthreads();
if (threadIdx.x % 2 == 0) printf("%d\n", A[threadIdx.x]);
}
int main() {
int n;
scanf(“%d”, &n);
dynshared<<<1, n, n * sizeof(int)>>>();
CDS(); }
```

Output :

Prints even number

Thank You