CS516: Parallelization of Programs

CUDA Thread Organization

Vishwesh Jatala

Assistant Professor

Department of EECS

Indian Institute of Technology Bhilai

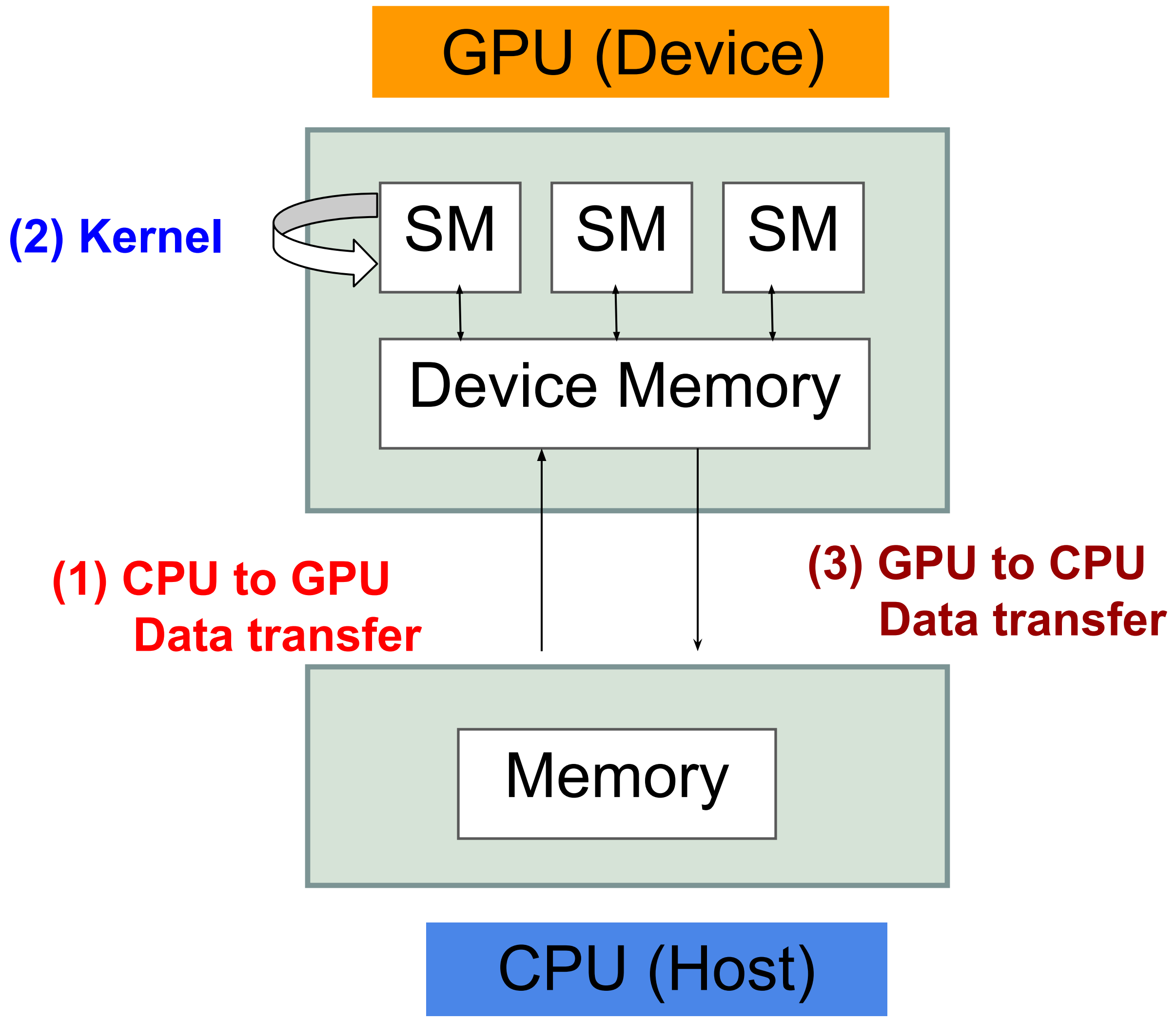
vishwesh@iitbhilai.ac.in

2022-23M

Outline

- Continue with CUDA Programming
  - Thread organization
  - Examples

CUDA Programming Flow



VectorAdd in CUDA

- For given two vectors A and B both having size N (where N<=1024), write a CUDA program to compute C=A+B

VectorAdd in CUDA

```
#define N 512
int main(void) {
    int *a, *b, *c; // host copies of a, b, c
    int *dev_a, *dev_b, *dev_c; //device copies of a, b, c
    int size = N * sizeof(int);
    …
    // Alloc space for host copies of a, b, c and
    // setup input values
    a = (int *)malloc(size); random_ints(a, N);
    b = (int *)malloc(size); random_ints(b, N);
    c = (int *)malloc(size);

    // Alloc space for device copies of a, b, c
    cudaMalloc((void **)&dev_a, size);
    cudaMalloc((void **)&dev_b, size);
    cudaMalloc((void **)&dev_c, size);
```

VectorAdd in CUDA

```
    // Copy inputs to device
    cudaMemcpy(dev_a, a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, b, size, cudaMemcpyHostToDevice);

    // Launch add() kernel on GPU with N threads
    add<<<1,N>>>(dev_a, dev_b, dev_c);

    // Copy result back to host
    cudaMemcpy(c, dev_c, size, cudaMemcpyDeviceToHost);
    // Cleanup
    free(a); free(b); free(c);
    cudaFree(dev_a); cudaFree(dev_b); cudaFree(dev_c);
    return 0;
}
```

VectorAdd in CUDA

```
__global   void add(int *a, int *b, int *c) {
        c[threadIdx.x] = a[threadIdx.x] + b[threadIdx.x];
}
```

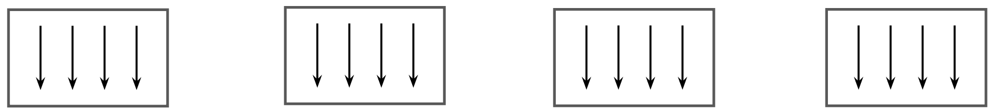
Practise Problem-1

- For given two matrices M and N both having size k*k (where k<=1024), write a CUDA program to perform M+N
  - *Hint: Allocate M and N single dimension array having k*k elements.*

Thread Configuration

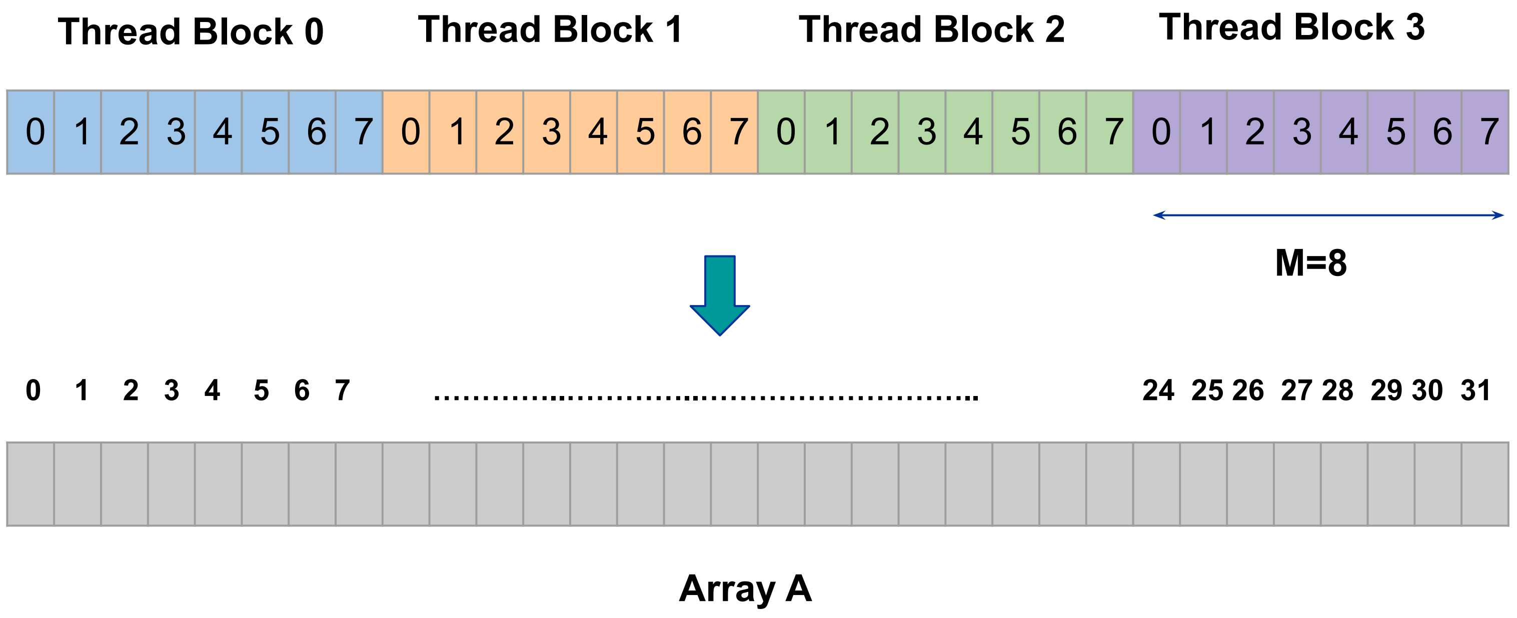
```
add<<<ThreadConfig>>> (dev_a, dev_b, dev_c);
```

```
add<<<ThreadBlocks, Threads>>> (dev_a, dev_b, dev_c);
```



```
blockIdx.x -> For identifying the block
threadIdx.x -> For identifying the thread within
a thread block
blockDimx.x -> Size of thread block
```

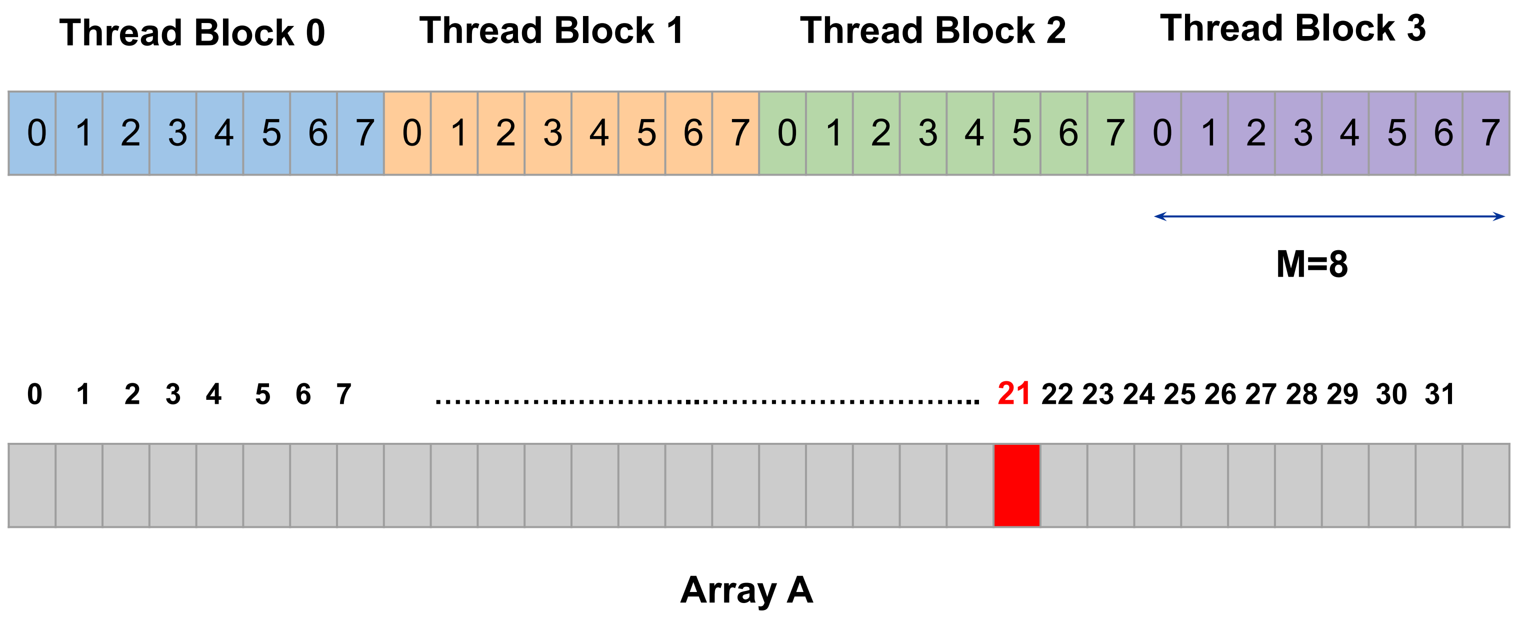
Indexing Arrays with Threads and Thread Blocks



What is the array index accessed by thread having threadIdx.x from the blockIdx.x?

```
int index = threadIdx.x + blockIdx.x * M;
```

Indexing Arrays with Threads and Thread Blocks



Which **threadIdx.x** and **BlockIdx.x** will operate on index **21?**

```
index = threadIdx.x + blockIdx.x * M;
   21 =           5 +          2   * 8
```

VectorAdd in CUDA with Thread and Blocks

```
__global__ void add(int *a, int *b, int *c) {
    int index = threadIdx.x + blockIdx.x * blockDim.x;
    c[index] = a[index] + b[index];
}
```

VectorAdd in CUDA

```
#define N 512
int main(void) {
    int *a, *b, *c; // host copies of a, b, c
    int *dev a, *dev b, *dev c; //device copies of a, b, c
    int size = N * sizeof(int);
    …
    // Alloc space for host copies of a, b, c and
    // setup input values
    a = (int *)malloc(size); random ints(a, N);
    b = (int *)malloc(size); random_ints(b, N);
    c = (int *)malloc(size);

    // Alloc space for device copies of a, b, c
    cudaMalloc((void **)&dev a, size);
    cudaMalloc((void **)&dev_b, size);
    cudaMalloc((void **)&dev_c, size);
```

VectorAdd in CUDA

```
    // Copy inputs to device
    cudaMemcpy(dev_a, a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, b, size, cudaMemcpyHostToDevice);

    // Launch add() kernel on GPU with N threads
    add<<<N/THREADS_PER_BLOCK,THREADS_PER_BLOCK>>>
              (dev_a, dev_b, dev_c);

    // Copy result back to host
    cudaMemcpy(c, dev_c, size, cudaMemcpyDeviceToHost);
    // Cleanup
    free(a); free(b); free(c);
    cudaFree(dev_a); cudaFree(dev_b); cudaFree(dev_c);
    return 0;
}
```

Threadblock configuration

```
add<<<ThreadBlocks, Threads>>> (dev_a, dev_b, dev_c);
```

- Thread block configuration
  - User choice
  - Depends on problem size
- Problem size = 32768 (1024 * 32)
  - Threadblocks = 32, No of threads/thread block = 1024
  - Threadblocks = 128, No of threads/thread block = 256

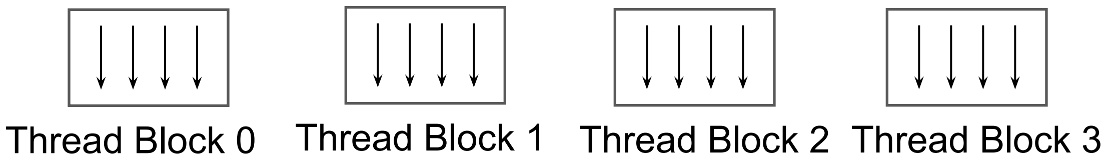
Practise Problem-2

- For given two matrices M and N both having size k*k, write a CUDA program to perform M+N using threads and thread blocks
  - ❑ `Assume threads in a thread block as THREADS_PER_BLOCK`

Thread Configuration

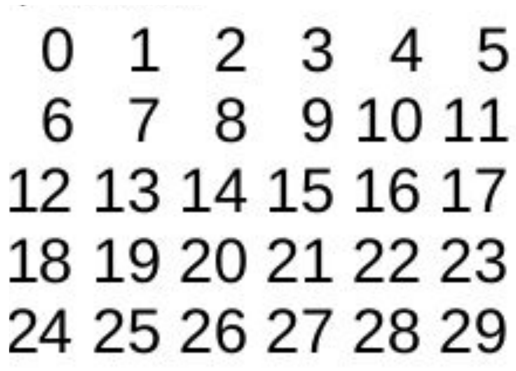
```
add<<<ThreadConfig>>> (dev_a, dev_b, dev_c);
```

```
add<<<ThreadBlocks, Threads>>> (dev_a, dev_b, dev_c);
```



Exercise: 1D Thread Organization

- For a given matrix A of size N*M, write a CUDA program to initialize the matrix elements as below



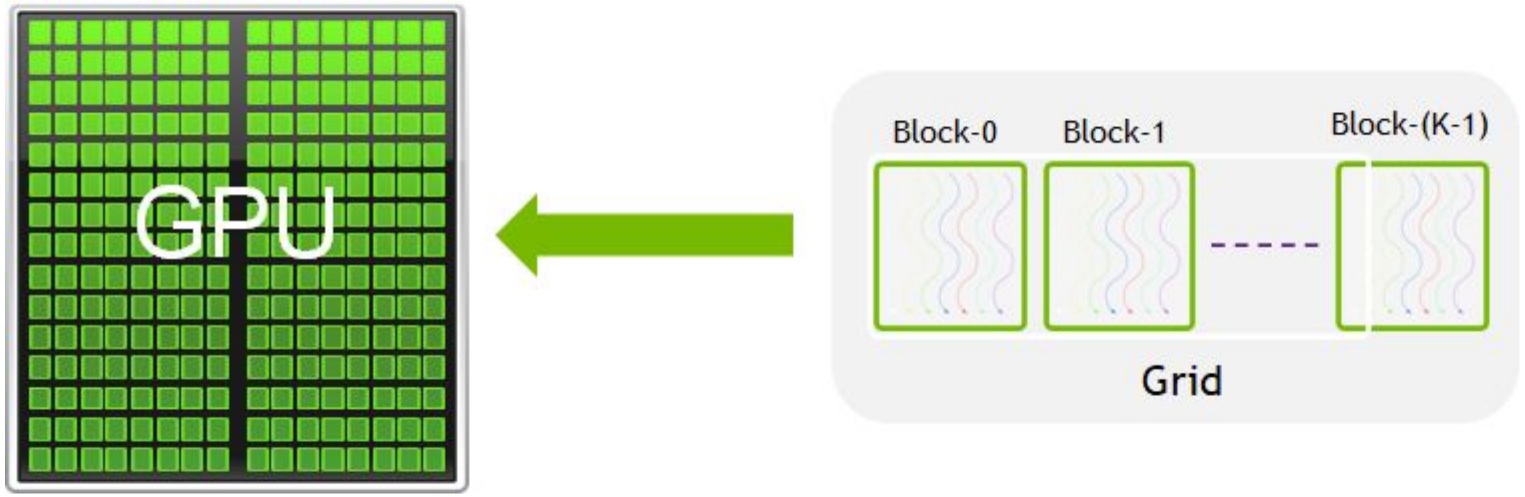
0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29

- Assumptions:
  - Matrix is stored in the single dimensional array.
  - No. of thread blocks = N
  - No. of threads in each block = M

1D

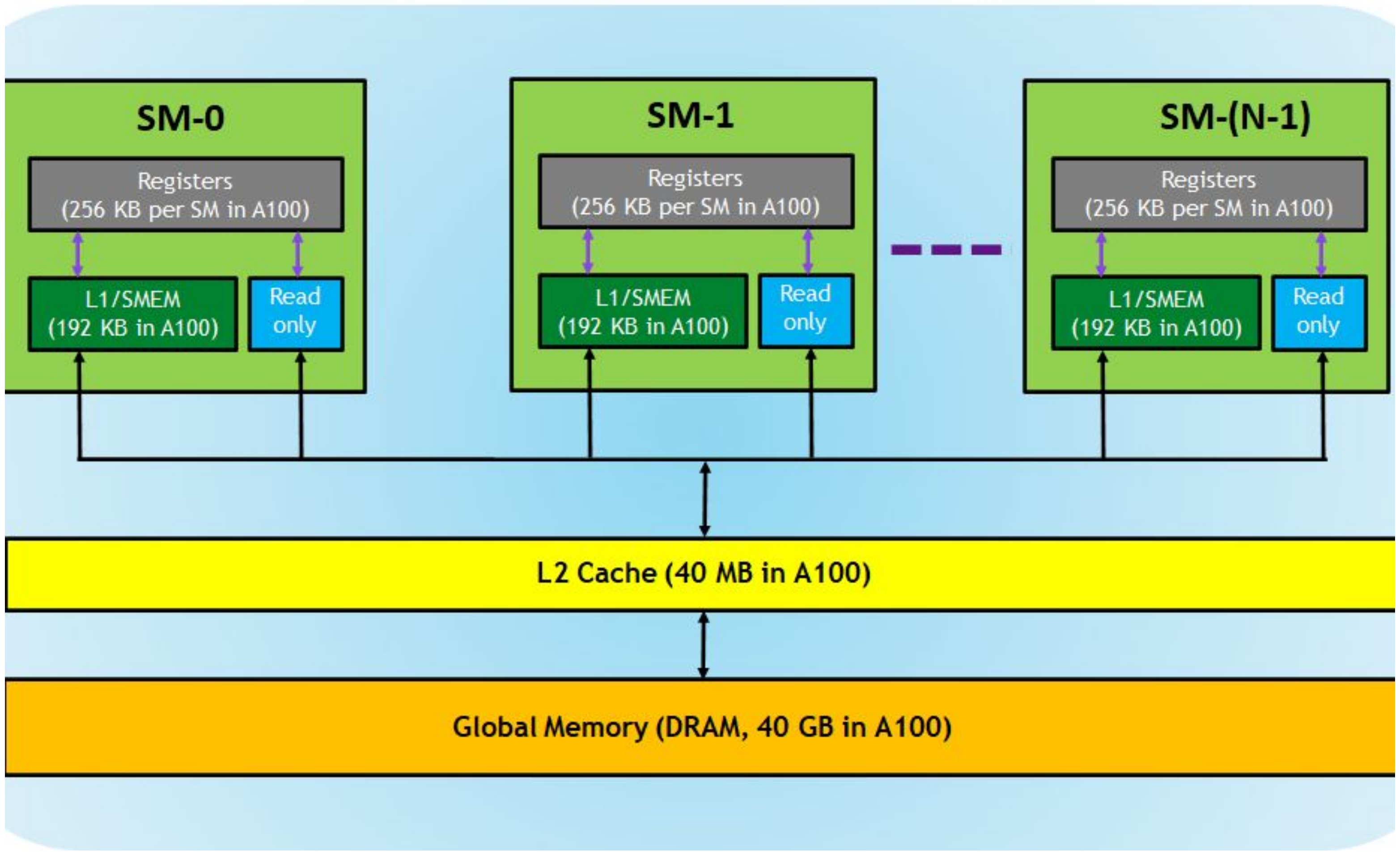
```
#include <stdio.h>
#include <cuda.h>
__global__ void dkernel(unsigned *matrix) {
        unsigned id = blockIdx.x * blockDim.x + threadIdx.x;
        matrix[id] = id;
}
#define N      5
#define M      6
int main() {
        unsigned *matrix, *hmatrix;

        cudaMalloc(&matrix, N * M * sizeof(unsigned));
        hmatrix = (unsigned *)malloc(N * M * sizeof(unsigned));

        dkernel<<<N, M>>>(matrix);
        cudaMemcpy(hmatrix, matrix, N * M * sizeof(unsigned), cudaMemcpyDeviceToHost);

        for (unsigned ii = 0; ii < N; ++ii) {
                for (unsigned jj = 0; jj < M; ++jj) {
                        printf("%2d ", hmatrix[ii * M + jj]);
                }
                printf("\n");
        }
    return 0;
}
```

Why Thread Blocks?



But why not 1 thread block with all the threads in it?

GPU Architecture

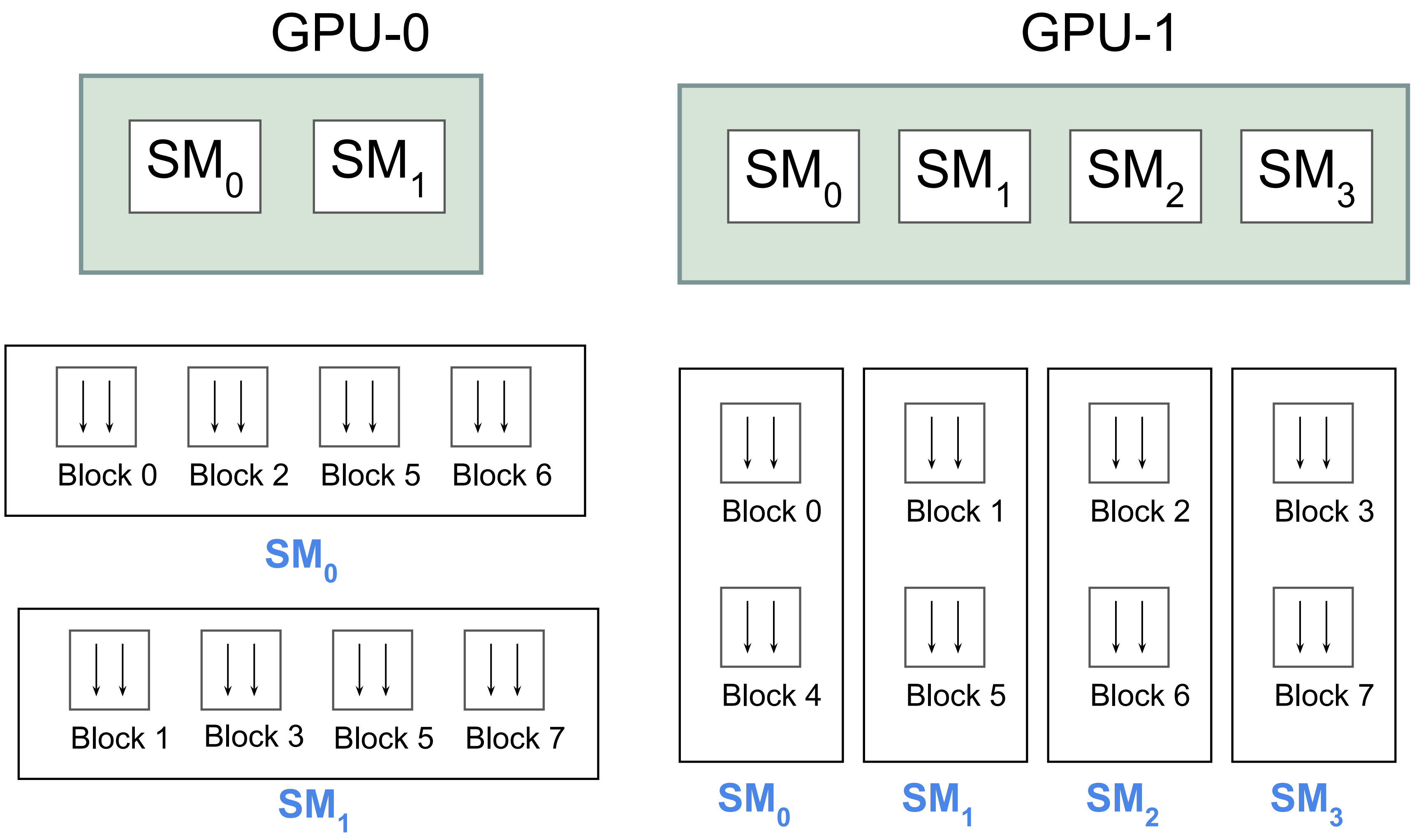


Few Constraints

- Thread block size has limit
  - Each thread executes the same program
  - Each thread requires some registers
  - Number of registers in each SM is finite

```
__global__ void add(int *a, int *b, int *c) {
    int index = threadIdx.x + blockIdx.x * blockDim.x;
    c[index] = a[index] + b[index];
}
```

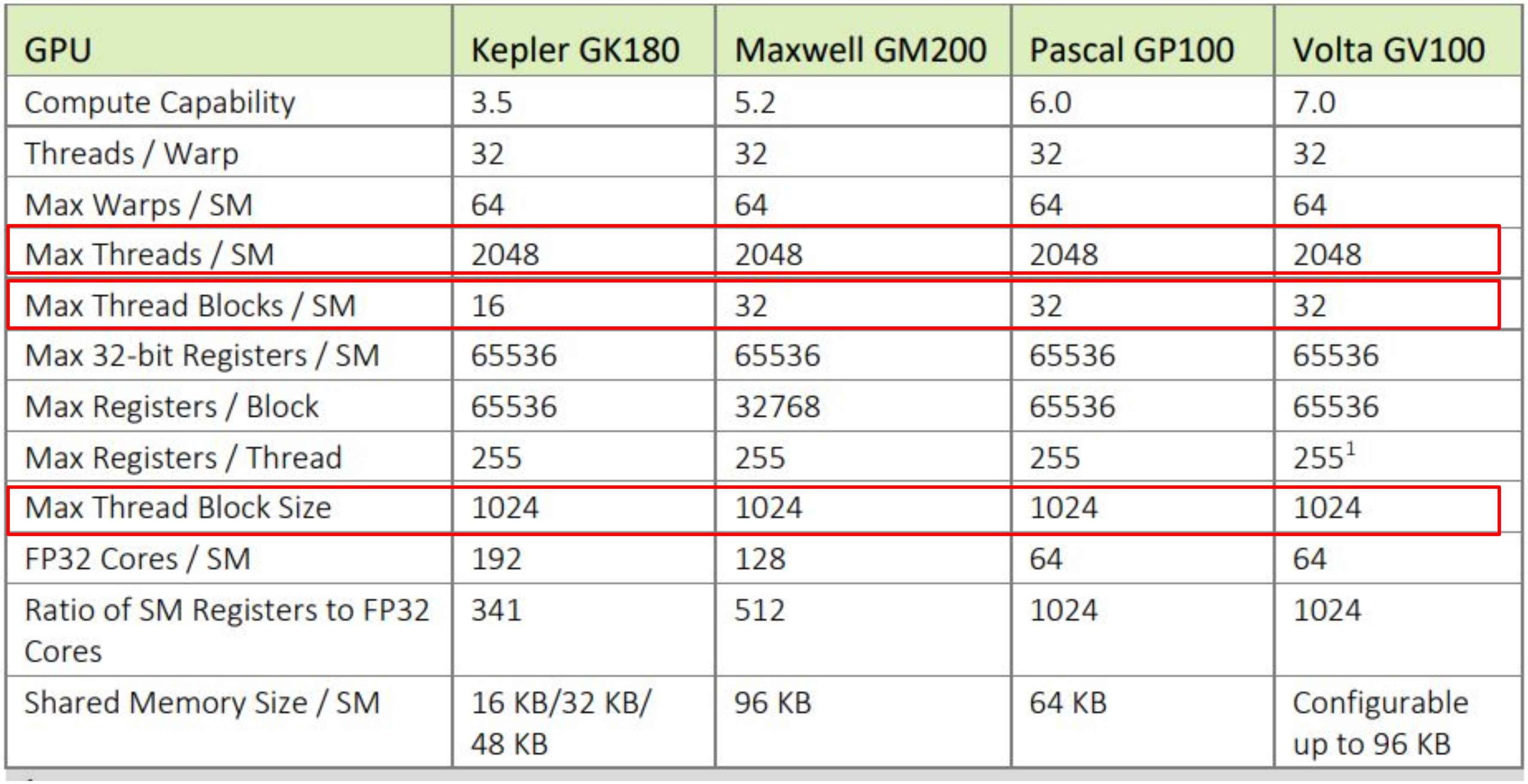
Scalability



Few Constraints

- Thread block size has limit
- Max number of threads blocks reside per SM
- Max threads reside per SM

Compute Capabilities



GPU	Kepler GK180	Maxwell GM200	Pascal GP100	Volta GV100
Compute Capability	3.5	5.2	6.0	7.0
Threads / Warp	32	32	32	32
Max Warps / SM	64	64	64	64
Max Threads / SM	2048	2048	2048	2048
Max Thread Blocks / SM	16	32	32	32
Max 32-bit Registers / SM	65536	65536	65536	65536
Max Registers / Block	65536	32768	65536	65536
Max Registers / Thread	255	255	255	255[1]
Max Thread Block Size	1024	1024	1024	1024
FP32 Cores / SM	192	128	64	64
Ratio of SM Registers to FP32 Cores	341	512	1024	1024
Shared Memory Size / SM	16 KB/32 KB/ 48 KB	96 KB	64 KB	Configurable up to 96 KB

Source: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

Summary

- CUDA Programming
  - Thread organizations:
  - Examples
- Next Lecture
  - Thread organization (2D & 3D)
  - GPU Instruction Execution

References

- CS6023 GPU Programming
  - https://www.cse.iitm.ac.in/~rupesh/teaching/gpu/jan20/
- Miscellaneous resources from internet