CHAPTER 4

Compute architecture and scheduling

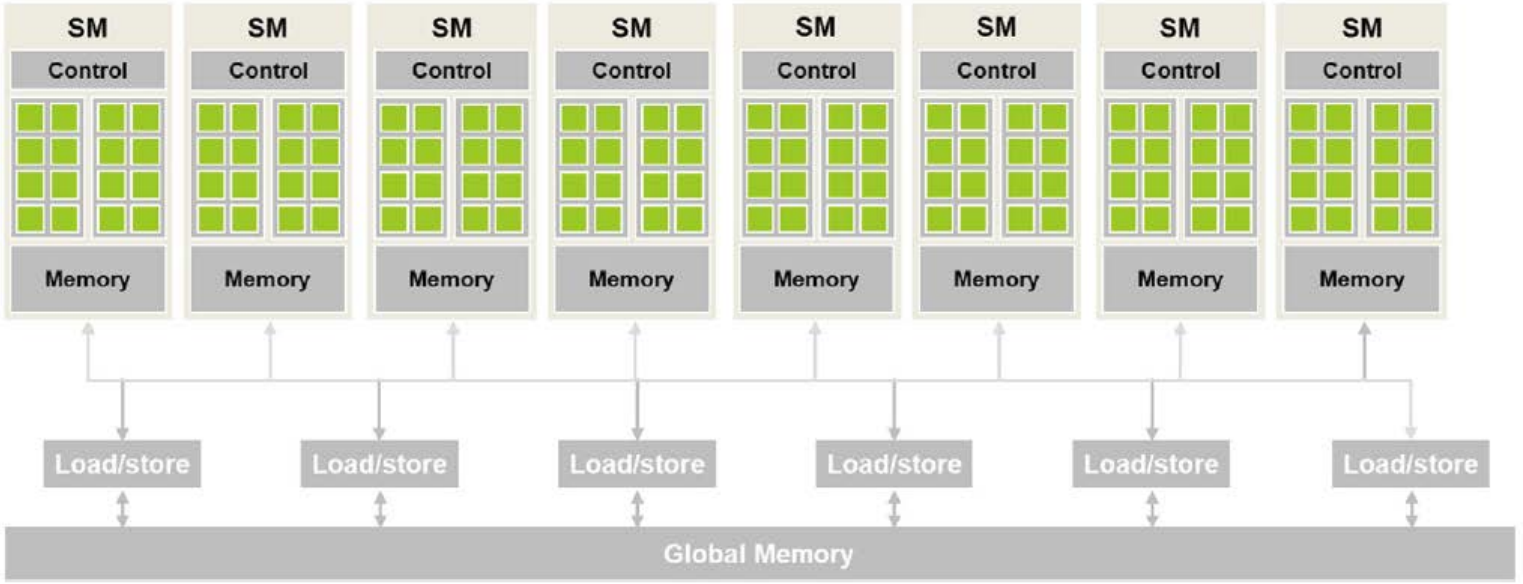


FIGURE 4.1

Architecture of a CUDA-capable GPU.

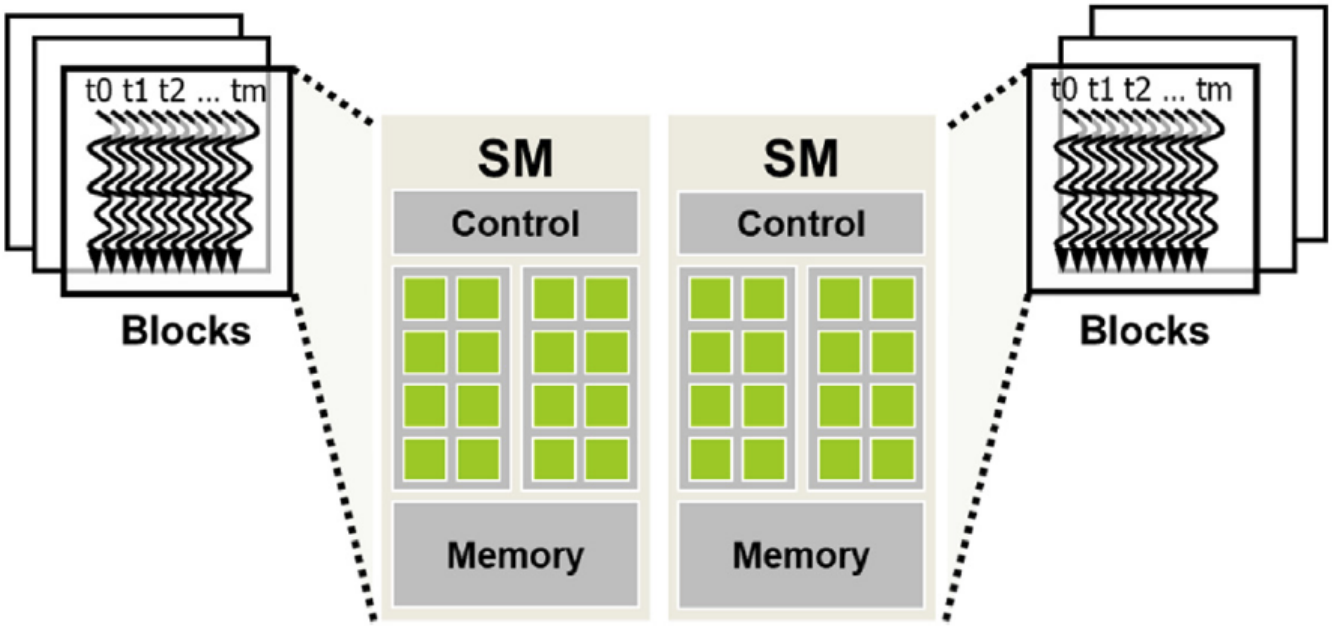


FIGURE 4.2

Thread block assignment to streaming multiprocessors (SMs).

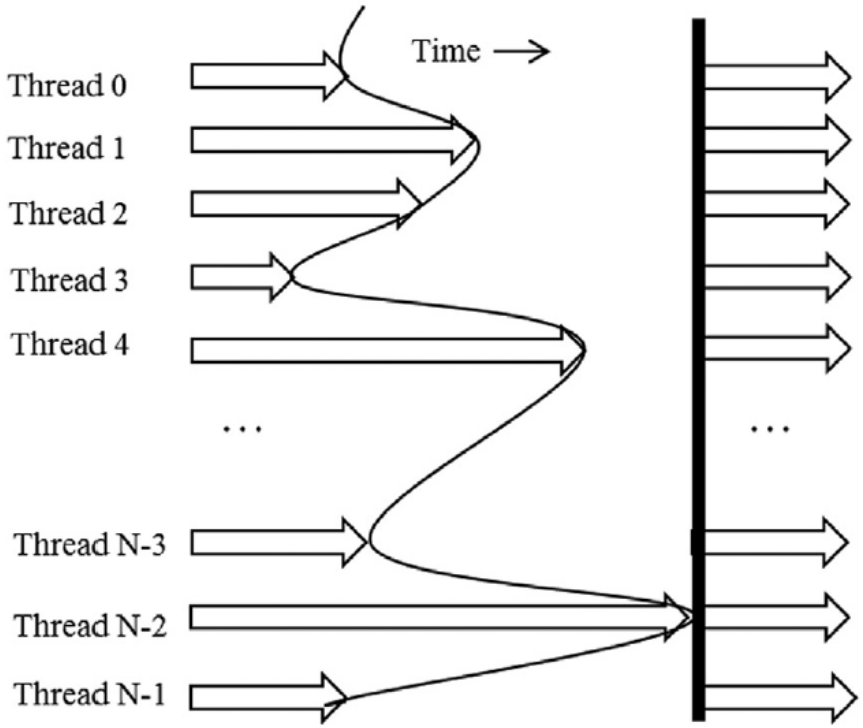


FIGURE 4.3

An example execution of barrier synchronization. The arrows represent execution activities over time. The vertical curve marks the time when each thread executes the__syncthreads statement. The empty space to the right of the vertical curve depicts the time that each thread waits for all threads to complete. The vertical line marks the time when the last thread executes the __syncthreads statement, after which all threads are allowed to proceed to execute the statements after the __syncthreads statement.

```
01    void incorrect_barrier_example(int n) {
02       ...
03       if (threadIdx.x % 2 == 0) {
04            ...
05            __syncthreads{};
06       }
07       else {
08            ...
09            __syncthreads{};
10        }
11    }
```

FIGURE 4.4

An incorrect use of `__syncthreads()`

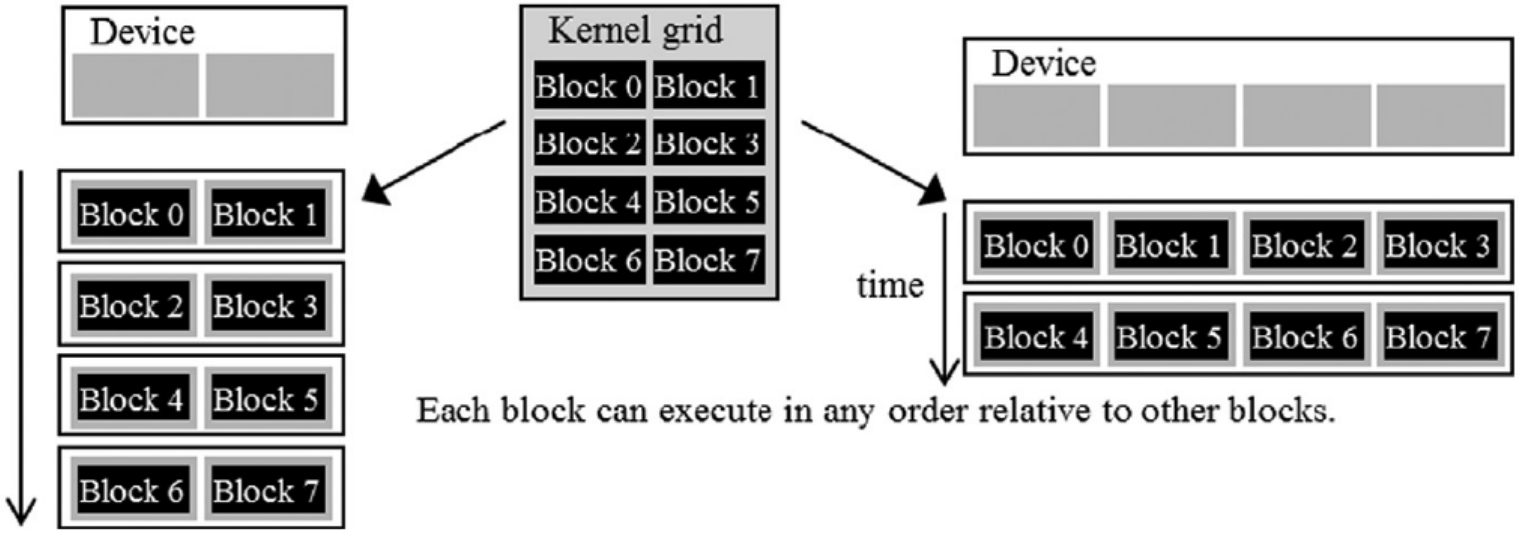


FIGURE 4.5

Lack of synchronization constraints between blocks enables transparent scalability for CUDA programs.

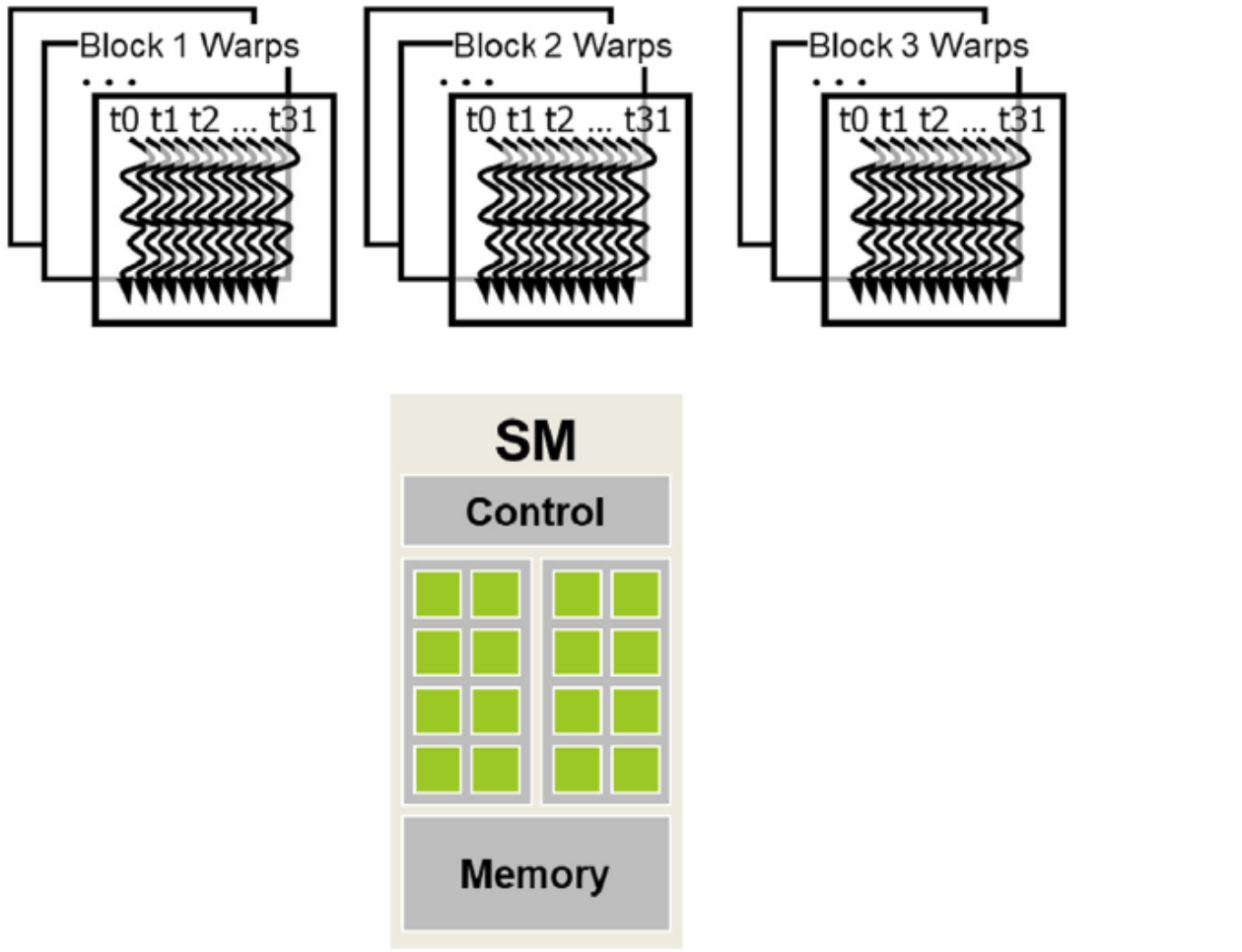


FIGURE 4.6

Blocks are partitioned into warps for thread scheduling.

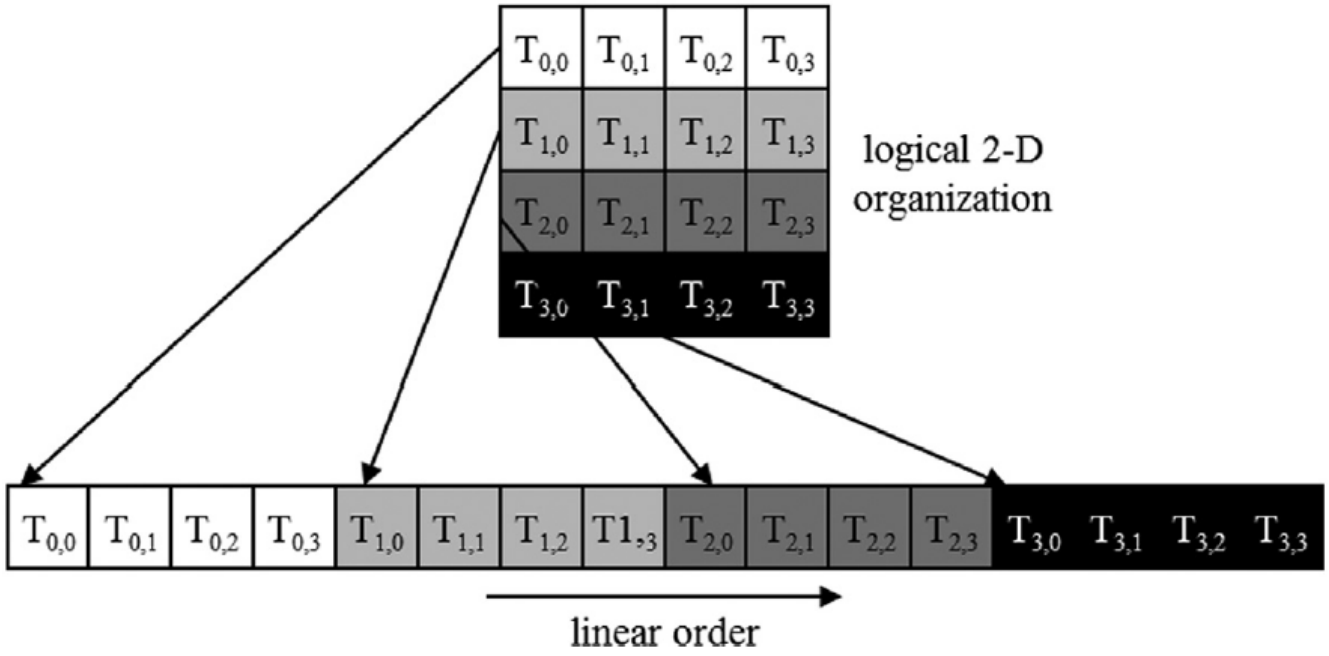


FIGURE 4.7

Placing 2D threads into a linear layout.

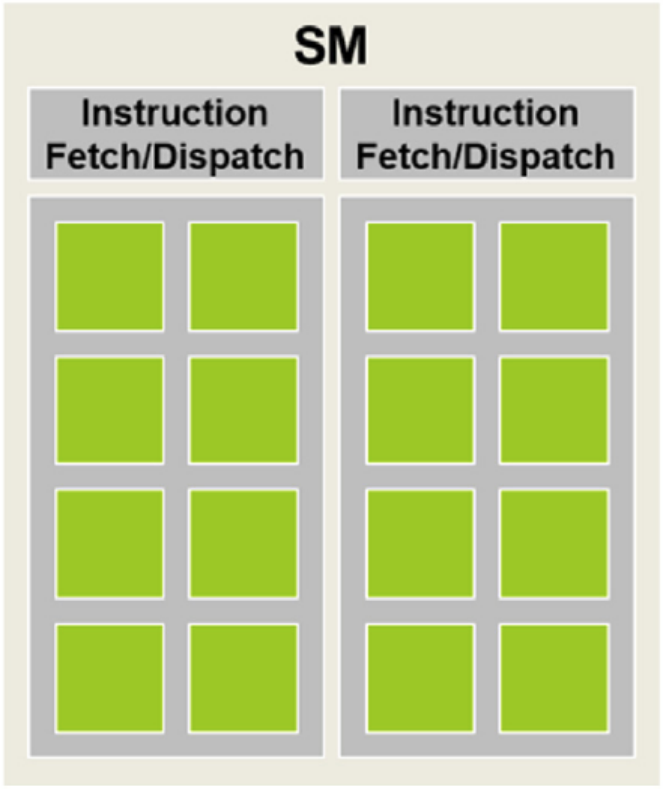


FIGURE 4.8

Streaming multiprocessors are organized into processing blocks for SIMD execution.

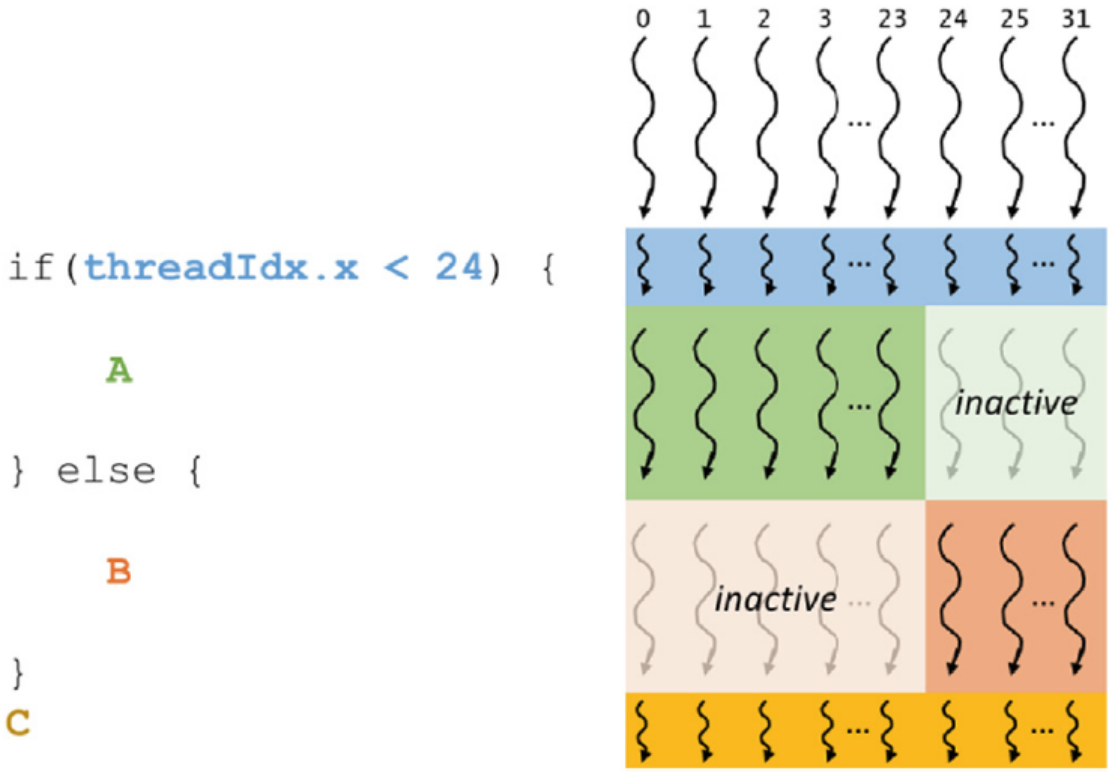


FIGURE 4.9

Example of a warp diverging at an if-else statement.

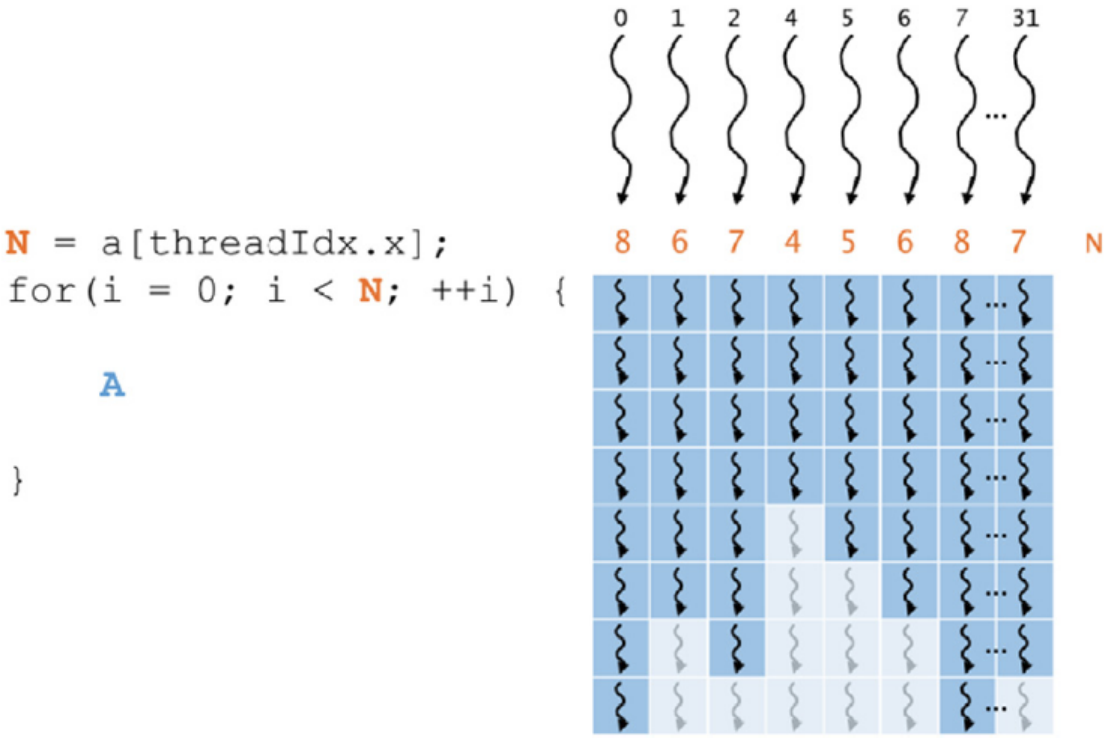
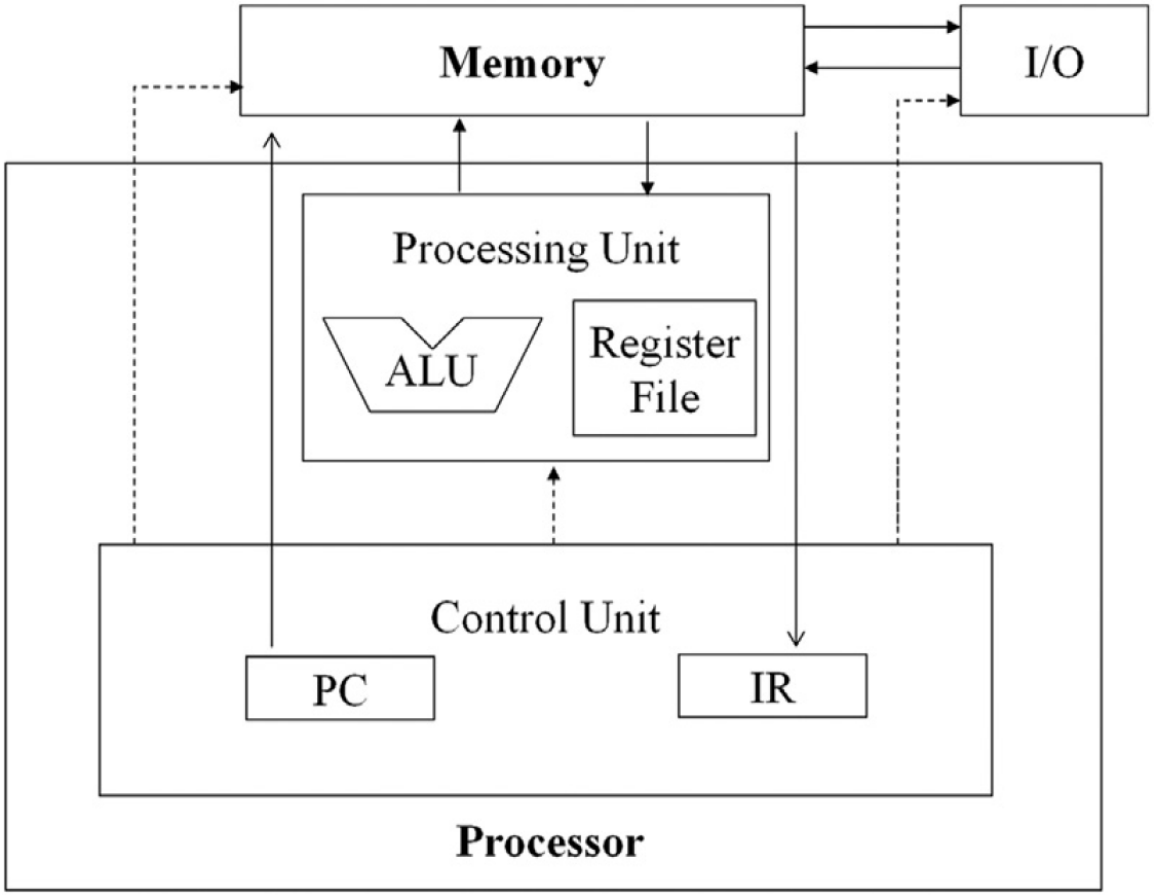
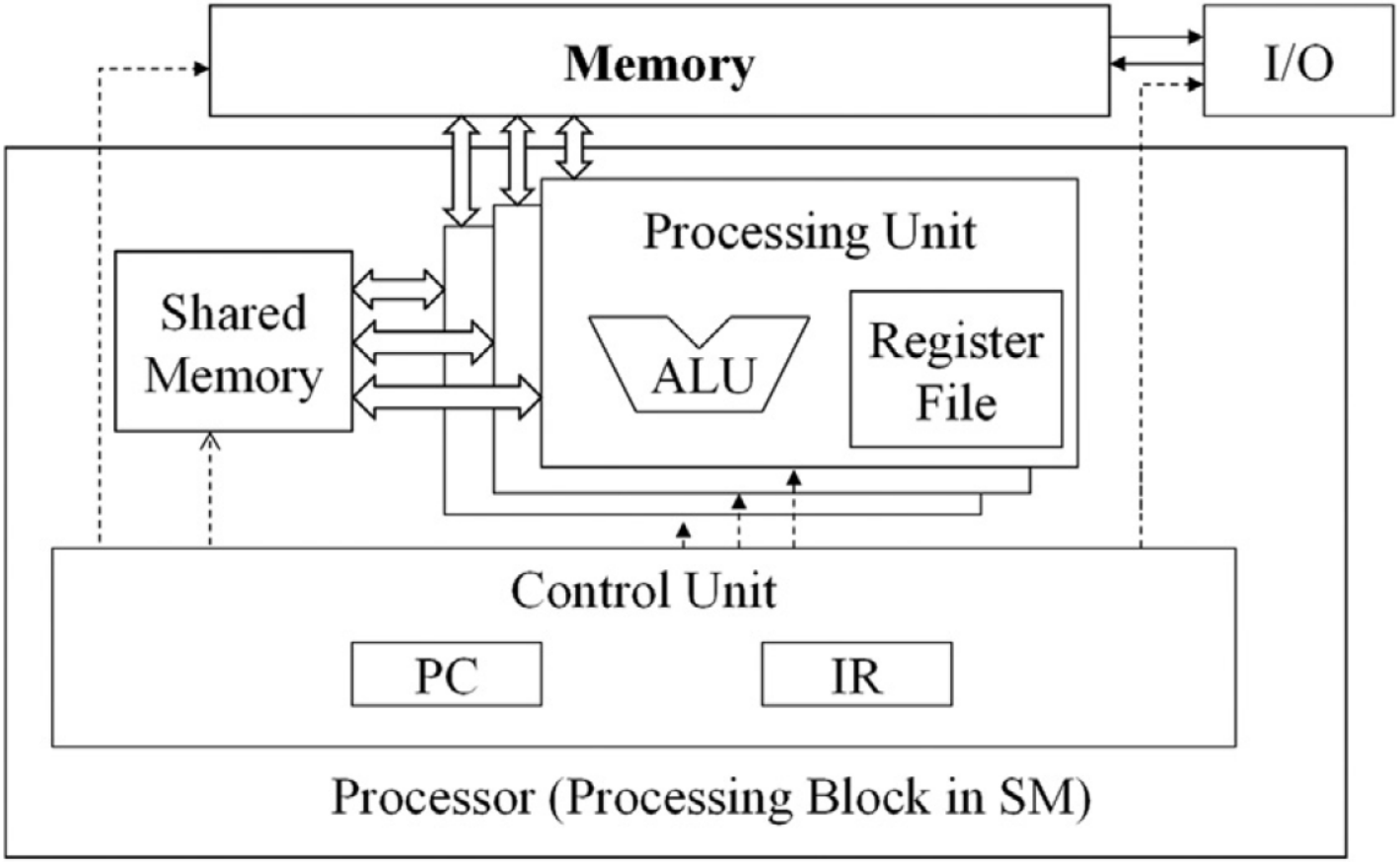


FIGURE 4.10

Example of a warp diverging at a for-loop.



In-text figure 1



In-text figure 2

```
01    __global__ void foo_kernel(int* a, int* b) {
02        unsigned int i = blockIdx.x*blockDim.x + threadIdx.x;
03        if(threadIdx.x < 40 || threadIdx.x >= 104) {
04            b[i] = a[i] + 1;
05        }
06        if(i%2 == 0) {
07            a[i] = b[i]*2;
08        }
09        for(unsigned int j = 0; j < 5 - (i%3); ++j) {
10            b[i] += j;
11        }
12    }
13    void foo(int* a_d, int* b_d) {
14        unsigned int N = 1024;
15        foo_kernel <<< (N + 128 - 1)/128, 128 >>>(a_d, b_d);
16    }
```

In-text figure 3

```
int devCount;
cudaGetDeviceCount(&devCount);
```

In-text figure 4

```
cudaDeviceProp devProp;
for(unsigned int i = 0; i < devCount; i++) {
    cudaGetDeviceProperties(&devProp, i);
    //     Decide     if     device     has     sufficient
resources/capabilities
}
```

In-text figure 5