3
CUDA and GPU Concepts

3.1
GPU Architecture

When programming in a CUDA-capable GPU, one must keep in mind its architecture and parallelism properties for they have an important role and impact on the performance of a GPU simulation. The architecture of a modern GPU is organized into a set of multiprocessors (SMs), each of which contains a number of streaming processors (SPs), as shown in Figure 3.1. CUDA arranges a group of threads to make up a thread block in which they can cooperate and synchronize amongst themselves. Subsequently, a grid is made up of a group of blocks. The device memory space is organized as follows. Global memory is an off-chip memory with slow access that can be accessed by all threads. Texture access is cached, as well as the constant memory, which is also read-only and can be accessed by all threads as well. Shared memory is an on-chip memory space that can be accessed by all threads in a block. Threads within a thread block can use shared memory to cooperate amongst themselves, and this is a good alternative for optimizing a program. Finally, each thread has its own memory space known as local memory which resides on global memory. Figure 3.2 illustates the CUDA memory hierarchy. Two important terms when programming in massively parallel processors are kernel and warp. A kernel is a function executed in parallel on the device (GPU) called from the host (CPU) side, while a warp is a group of threads executing synchronously within a block.

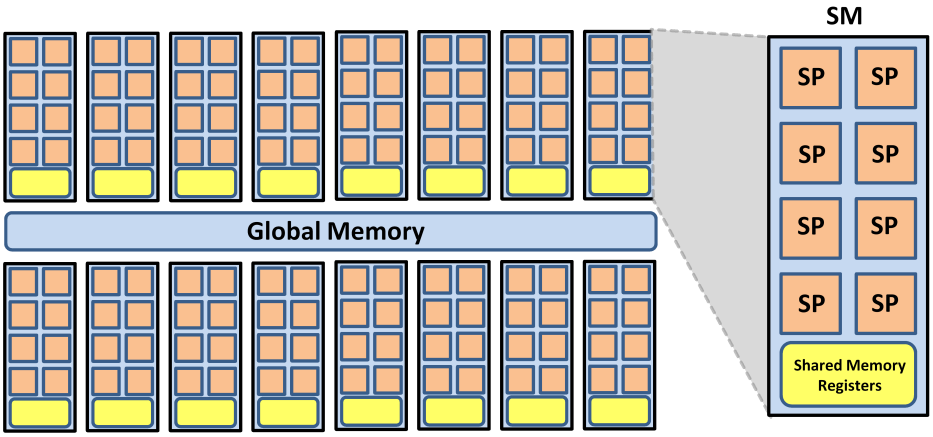

Figure 3.1: Diagram of a G80 architecture with 16 SMs and 128 SPs, based on the figures presented in (17).

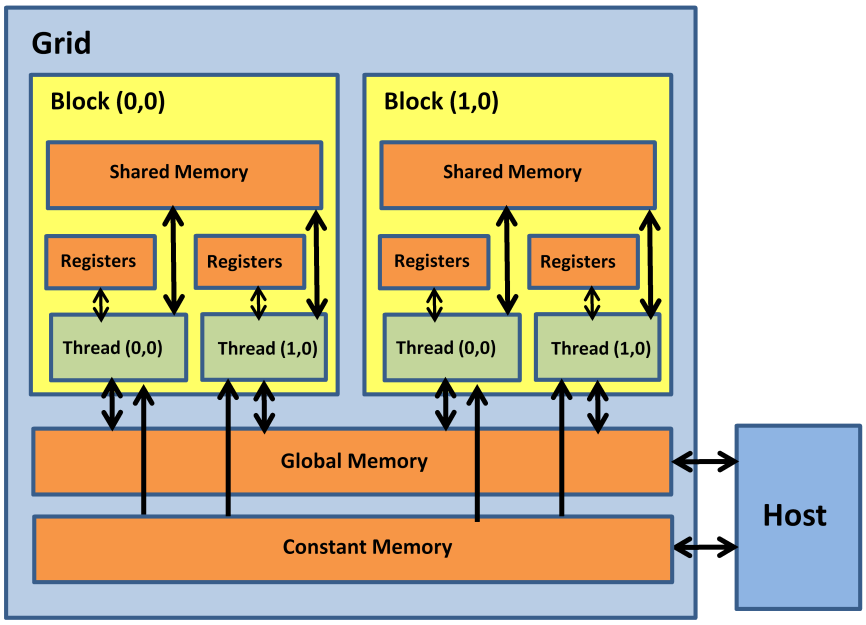

Figure 3.2: CUDA memory hierarchy, based on the figures presented in (17).

3.2 Optimization

It is important to highlight some CUDA programming issues. In a kernel execution, each thread must write on a different memory space as they are being executed concurrently, thus avoiding writing conflicts. All threads within a warp execute the same instruction (SIMD architecture), so it is suggested

that there are no conditinals and loops that lead to thread divergency within the warp. When multiple global memory accesses are coalesced into a single memory transaction by the device (i.e. proper memory access alignment and contiguity), we achieve a coalesced reading. When seeking a performance optimization in a kernel execution, it is best to minimize global memory accesses, since they are slow. When access to global memory is mandatory, coalesce reading helps increase the simulation performance. To coalesce, each half warp must access contiguous 4, 8, or 16-byte words lying in the same 64 or 128-byte segment (for compute capabilities 1.0 or 1.1), which sometimes is difficult to achieve in actual simulation. Also, it is important to avoid bank conflicts when using shared memory. Shared memory is divided into banks, and if multiple threads in the same half-warp access the same bank, access must be serialized. To avoid bank conflicts, all threads of a half-warp must access different banks or all of them must read the identical address. Finally, in order to reach an optimal kernel performance, one has to maximize thread occupancy, defined as the ratio of the number of resident warps to the maximum number of resident warps, depending on the GPU architecture (17). The ways occupancy can be maximized include minimizing the number of registers per thread and shared memory.