Problem 5

Even Flørenæs

Fall 2016

TDT4200 Parallel Computing
Department of Computer Science

Contents

1 Theory

1.1 Difference between CPU and GPU architecture

The CPU is composed of a few cores with lots of cache memory that can manage to handle a few software threads at a time. In contrast, a GPU is composed of hundreds or thousands of cores that can handle thousands of threads simultaneously. The ability of a GPU with 100+ cores to process thousands of threads can accelerate some software by 100x over a CPU alone. What's more, the GPU achieves this acceleration while being more power and cost-efficient than a CPU.

In CUDA a program on the GPU is executed by group of thread blocks. The number of threads to be executed in a thread block is specified by the host code upon launch of the device kernel. Each block may have up to 3 dimensions and the threads are grouped in multiples of 32 (a.k.a warps) that act as a synchronized Single Instruction on Multiple Threds (SIMT). The GPU SM(X) schedules threads in groups of 32 parallel SIMTs threads called warps. The warp size is introduced so CUDA can take advantage of certain hardware-based optimizations provided on NVIDIA devices, including the warp scheduler. The threads with a given block may share data and exhibit synchronized execution. Generally, only one kernel executes at a certain time, and a thread block executes on one multiprocessor.

1.2 Thread divergence

Threads from a block are bundled into fixed-size waprs for execution on a CUDA core. Where the threads within a warp must follow the same execution trajectory. Meaning all threads must execute the same instruction at the same time. If the code of the instruction contains one or multiple if-then-else statements the threads may branch to different locations depending on the evaluation of the statement being true or false. In this case the threads will diverge in execution. This is not allowed in the CUDA platform. In the CUDA platform the solution to this problem is to execute the then part of the if-then-else statement, and the nrpoceed to the else part. While executing the then part, all threads that evaluated to false are effectively deactivated. When executing the then part, all threads that evaluated to false are effectively deactivated. When execution proceeds to the else condition, the situation is reversed. As you can see, the then and else parts are not executed in parallel, but in serial. This serialization can result in a significant performance loss.