Problem set 5

Theory questions

EVEN FLØRENÆS

TDT 4200

PARALLEL COMPUTING

FALL 2016

Problem 1

a) Briefly explain the difference between the architecture of CPUs and GPUs.

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 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.

b) Explain the difference between thread-block and a warp

In CUDA a program on the GPU is executed by a god 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 synchronised Single Instruction on Multiple Threads (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 optimisations provided on NVIDIA devices, including the warp scheduler.

The threads within a given block may share data and exhibit synchronised execution. Generally, only one kernel executes at a certain time, and a thread block executes on one multiprocessor.

c) What is thread divergence?

Threads from a block are bundled into fixed-size warps for execution on a CUDA core. Where the threads within a warp must follow the same execution trajectory. Meaning all threads must excuse the same instruction at the same time. This implies that the threads cannot have diverge development through the threads instructions. 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 diverges 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 then proceed to the else part. While 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 serialisation can result in a significant performance loss.

Problem 2

Mention and briefly explain each of the logical CUDA memory spaces

Global memory	Slow memory slots which are uncached. Possibility to do both read and write to and from. Requires sequential & aligned 16 byte reads and writes to be fast (coalesced read/write)
Texture memory	Read only memory. Cache optimised for 2D spatial access pattern.
Constant memory	This where constants and kernel arguments are stored. Slow, but with cache (8 kb)
Shared memory:	Fast but take care of bank conflicts. Exchange data between threads in a block.
Local memory	Slow & uncached, but automatic coalesced reads and writes. Used for whatever does not fit into the registers.
Registers	Fastest memory. Scope is thread local.