CPS notes

Multi-threading

Thread

* A thread of execution is the smallest sequence of computer instructions controlled by the operating system
* Operating systems mange threads
* Threads are part of a process
* Difference is process has isolated memory
* Threads are made of fibres
* Threads Lifecycle
  + New
  + Runnable
  + Running
  + Waiting
  + Dead

Scheduling

* Scheduling is the method of allocating threads process times
* In general, os will conduct scheduling in a round robin
* Each thread is given a timeslice once this is complete the thread is pre-empted
* Two main types are Pre-emptive and Cooperative Multitasking
* Preemptive is an operative controlled approach allowing the programmer to not worry about scheduling however it means you have less control over setting priorities
* Cooperative multitasking involves threads deciding when they can access the processor this is more complicated

Context Switching

* Context Switching occurs when we switch threads
* Is the bane of multi-threading
* Need to save, the load and continue running
* Occurs a interrupt, waiting and multi-tasking
* Computationally intensive require processor time

Mutexs and Futures

* Mutexs (mutual exclusion point) is an object that guard a object that allows threads to share resources
* Futures send of work to be done and return a result later

OpenMP

* Aims to consider theads a low level concept
* Is preempitve

GPU

Parallel Problems Types

* Task Parallelism
* Data Parallelism

Pipeline

* The GPU forms a pipeline transforming point data to pixel data
* Data Driven
* Work is broken into 3-D grid
* The GPU runs instances of our kernel
* Each work item (“Thread”) has an ID
* N compute units execute N work items in waves

GPU performance

* About 150 times faster than top end CPU
* Copying data to and from the GPU is your bottleneck
* Memory bus speed to GPU is slow

Problem Solving on the GPU

* Matrix output
* Each thread has an ID
* Output based on this location ID
* Think in for loops mainly nested for loops up to three levels
* Which parts cannot be parallelised?
* Look at for loops
* Think how data can be worked on
  + Can you leave data on the GPU
  + Con you stage work and leave it on the GPU
  + Do you need to reduce results in some manner
  + What happens during sequential parts?
* Avoid memory copying
* Each point must be calculated within isolation

Concurrent and parallel Design and Theory

Concurrency

* Is a system that has multiple computations executing simultaneously
* Two main pluses separating concerns and performance
* On Performance
  + Task Parallelism involves multiple tasks running
  + Data Parallelism involves the same program to run on multiple data sets

Flynn’s Taxonomy

|  |  |  |
| --- | --- | --- |
|  | Single Instruction | Multiple Instruction |
| Single Data | (SISD) | (MISD) |
| Multiple Data | (SIMD) | (MIMD) |

* MIMD
  + Usually dealing with task Parallelism
  + Weak for data parallel problems
* SIMD
  + Works best with data parallelisation problems
  + Works as
    - Get data
    - Spilt data across processors
    - Get results
  + Good for streaming data
* Single Program Multiple Data
* Multiple Program Multiple Data

I/O issues

* One of the biggest uses of concurrency in standard application is dealing with I/O
* I/O is very slow in comparison
* Better to spawn another thread
* Same with GUI

Models of Concurrency

* Two main models Shared Memory and Message passing
* Shared Memory
  + Processes share memory locations and communicate by altering these memory locations
  + Have to lock critical memory locations
  + Becomes very complicated
* Message Passing
  + Processes communicate with each other by sending messages and is simpler
  + Synchronous communication and pure non-shared message passing can have an overhead while waiting occurs

Scaling

* Scaling is how the performance of a parallel application changes as the number of processors is increased

Strong scaling

* Total problem size stays the same number of processors increase

Weak Scaling

* Problem size increases while number of processors stays the same