Concurrency

Some Basics

Joseph Kehoe[1]

[1]Department of Computing and Networking
Institute of Technology Carlow

CDD101, 2017

Table of Contents

Table of Contents

Composition of Concurrent Algorithms

- Algorithms consist of one or more tasks acting on data
- Dependencies will exist between the tasks

Data Dependency one task requires data to be “prepared” by another task before it can start

Control Dependency Task side effects need to be ordered e.g. I/O Operations

Fork-Join

Fork-Join is one popular way of managing these depencies

- New control flows (concurrent tasks) are created at a fork point
  - One splits into many
- Synchronisation occurs when tasks are merged into a single control flow
  - Many become one
- Each control flow is sequential

Table of Contents

Data versus Function

Data Parallelism

- Parallelism grows as data grows
- This is scalable

Functional Parallelism

- Divide task into multiple concurrent tasks
- Not as scalable (why?)

Regular versus Irregular

Regular Parallelism

- Tasks are similar
- Tasks have predictable dependencies
- E.g. Matrix Multiplication

Irregular Parallelism

- Tasks are dissimilar in a manner that creates unpredictable dependencies
- E.g. Search in games

Table of Contents

Thread Parallelism

- Each task has its own flow of control
- Useful for all types of parallelism

Hardware Thread: Thread that is supported by hardware e.g. seperate core for each thread (parallel)

Software Thread: Software based thread e.g. time slicing of processor time (concurrent)

Hyperthread: Core has duplicated some components to allow it to run two threads at once

Vector Parallelism

- Single control flow can operate on multiple data elements
  - Useful for regular parallelism (mainly)
- Intel AVX allows register to hold many (8) 32 bit floating point numbers
  - All can be acted on simultaneously
- Requires less silicon to implement than thread based parallelism
- This approach can emulate thread parallelism by using Packing or Masking
  - These “threads” are called fibers

Table of Contents

Flynn's Categories

SISD	Single Instruction, Single Data
SIMD	Single Instruction, Multiple Data
MIMD	Multiple Instruction, Multiple Data
MISD	Multiple Instruction, Single Data

GPU Vendors use:

SIMT	Single Instruction, Multiple Threads (SIMT). A Tiled SIMD where each SIMD processor emulates multiple threads (fibers, really) using masking

von Neumann Bottleneck

There is a memory hierarchy where each level can be more than an order of magnitude slower than the next (from fastest to slowest)

- Registers (on each core)
- L1 Cache (Instruction and data caches) on each core
- L2 Cache shared between multiple cores
- L3 Cache one per processor
- RAM Shared by everone on board
- Main Memory (SSD or mechanical) Shared by everyone on box

Table of Contents

Performance

Lots of things affect performance but at a high level remember the following:

Data locality
: Keep data close to the thread using it

Slack
: Have more potential tasks than there is actual parallelism

Avoid having too many threads
: One per core is ideal. Use a thread pool!

More on performance later!