SIT315 M2.S3P: Concurrent and Parallel Programming Introduction

Codey Funston | S222250824
No Group

Activity 1 – Parallel and Concurrent Programming

1. Concurrent programming is the practice of managing multiple processes in a program in an overlapping way with context switching to move between them. For example, for a single core web server to handle multiple requests it must context switch between listening for and handling requests. It may also have a database that it gets data from and while the process handling the databases queries is waiting for a response, the web server can do a context switch and focus on something else. Parallel programming involves utilising multiple cores in a multi-core CPU to perform multiple processes at the exact same time. From a basic view, parallel programs can parallelise threads to do the same task but on different data, or different tasks entirely. Distributed computing is very similar to parallel programming, except that the threads are distributed across multiple CPUs and communicate via network protocols instead of within the same operating system. An example of this would be a search engine. Since it is performing such a computationally intensive task and a single multi-cored CPU may not be enough, the task is horizontally scaled/distributed across multiple CPUs.

2. Amdahl's law tells us how much we can expect a program to speed up based on the number of available processors on your machine and the proportion of the program that can be parallelised. The formula for speed up is $\frac{1}{1-p+\frac{p}{n}}$ which means there will be an increase in speed up as the number of processors, n, or the proportion of the program possible for parallelisation, p, increases. The reason that in this unit the speedups were very large for matrix/vector filling and basic operations is because almost the entire process is easy to do in parallel since there isn't much coordination needed between individual threads since they work on isolated parts of the data.

3. Option 2 provides a greater speedup based on the following math comparing proportion of the original program, and the speed up of the changed section, it is pretty cool math but fairly simple logic:

$$let\ original\ program\ time = k$$

$$option\ 1\ time = (0.20 \times k) \times 0.10 + 0.80 \times k = 0.82k$$

$$option\ 2\ time = (0.50 \times k) \times 0.50 + 0.50 \times k = 0.75k$$

Activity 2 – Flynn's Taxonomy

1. SISD is the best architecture to begin with because it is how basic single-core CPUs act. You have one thread performing one task on one piece of data. The next easiest to understand is SIMD which is a form of data parallelism in which you have multiple threads performing the same task but on the same data (different sections). An example is GPUs that are designed for parallel data processing. Next is MIMD, this is hardware that has multiple threads performing tasks on multiple pieces of data. This is what most modern CPUs are like. MISD is an unusual concept at first because it involves moving through a piece of data multiple times doing different things. A cool example is for redundancy where you cannot afford to lose any information or processing such as in some real time systems like space travel hardware.

2. Array Processor is a way to organise a parallelisation with multiple available processors and a singular "control unit". It involves a single instruction stream being sent to the processors but multiple data streams. An example is in a C++ program where the master thread assigns other threads parts of data to work on, all with the same instruction. Vector addition is a good example of this.

3. SPMD and MPMD are like higher level versions of SIMD and MIMD respectively. SPMD involves multiple processors all executing the same program with different data, whereas MPMD has each processor executing a different program but also with the same data. This high level implementation is often done across distributed computers, for example you could assign each computer to run the same executable but on different data which would be SPMD. Or for MPMD, you might want to perform different types of data analysis for a scientific computing application across different pieces of data, with different executables for each analysis type running on different computers.