Christian Dominic Angus

COT 4431 – Assignment 1 Part 1

1. **Why is parallel computing needed for large scientific problems? What are the different principles of parallel computing (i.e., how parallel computing is realized)?**

   Computational science and engineering problems have forced us to innovate how we've built computers. They have become faster because of the number of transistors that we're able to put into a single computer core, as well as the ability to run these cores simultaneously, which is known as parallel computing. These innovations have made larger computational problems significantly less time consuming to perform.

   For example, for us to predict the weather, we need to be able to process data from previous days and predict the weather at least the day before it happens. Otherwise, the prediction would become irrelevant. Without parallel processing it would take days to process all the data required to make a prediction, that's why parallel computing is important.

   The first principle of parallel computing is finding enough parallelism, this assumes that there is enough work to parallelize. It uses Amdahl's Law, where s is the fraction of work done sequentially, therefore, 1-s is the fraction of work parallelizable and p is the number of processors and p = 1/(s+(1-s)/p) or 1/s, which means that the more parallelizable work there is, the time it takes to execute the task is sped up.

   Another principle of parallel is granularity, which ties in from the first principle, since there is a large amount of work to be done, the work gets divided into how much work can be done in parallel.

   Locality, then makes sure that tasks are done locally in terms of memory hierarchy since moving data from one memory hierarchy to another memory hierarchy costs more in arithmetic. Load balance accounts for the time that some processors in the system are idle waiting for another due to insufficient parallelism.

   Coordination and synchronization ensure that data is shared safely, where parallel programming gets divided into two layers, efficiency and productivity layers. Where the efficiency layer is used by experts where they build libraries implementing kernels, frameworks and operating systems, and the productivity layer is used to build application by composing frameworks and libraries, a lot of details of the machine is hidden from this layer for safety. And lastly is performance modeling, debugging and tuning.

2. **What are the computational motifs and/or dwarfs that we discussed in the lectures? Why are they important?**

Finite State Machine, Combinational, Graph Traversal, Structured Grid, Dense Matrix, Sparse Matrix, Spectral (FFT), Dynamic Programming, N-Body, MapReduce, Backtrack (B&B), Graphical Models, and Unstructured Grid. These computational modules and algorithms are quite common in today's computational science and engineering and commercial applications that we should spend time and invest in using parallel computing in applications that use these motifs. For example, in the discussion, it was shown that health applications rely heavily on graph traversal algorithms, therefore, parallel computing should be applied to the computation of the specific health application. These applications rely heavily on how fast things can be computed, especially with health-related computations, which can be time restricted, so time is of the essence in these applications. Parallel computing in these motifs really allows these algorithms to perform faster, therefore more computation can be performed in the time saved.

These motifs are important because they are used in everyday applications, not just computational science and engineering, but as well as commercial applications. Things like databases, video games, machine learning use these algorithms, but also health applications, image analysis, speech analysis, music, and our internet. These algorithms are present in almost all the things that we do throughout the day. Running Spotify, an online music service, alone checks off two, music and internet, things that use the motifs discussed in our lecture. So, it is worth investing time, money and energy into utilizing parallel computing into these motifs so that not only do they run faster and more efficient, but it also saves the end users time, which then allows us to do more things with our current technology.

3. **What is meant by memory-hierarchy in multicore architectures and why are they important? And what is meant by gap between the computing and communication bandwidth?**

Each processor core has a memory controller that can utilize multiple types of memory. Memories generally become slower the larger it is. Cache memory is one of the fastest memory types, 1ns and 10ns for on-chip cache and second level cache (SRAM), respectively, for data retrieval, but they only have a few kilobytes and megabytes for on-chip and SRAM, respectively. Main memory (DRAM) are the physical modules usually installed on the motherboard, they come in gigabytes but are slower than cached memory, 100ns, for data retrieval. Secondary storage (disks) is another memory type, often in the form of hard disks or solid-state disks, they are generally used for mass storage, versus volatile storage in the case of DRAM, because they are slow to access, 10ms, for data retrieval.

These memory hierarchies look to exploit locality to improve the average time it takes to process data. For example, it would be fast to process a photo stored in DRAM, however, it would not be economical for the system to do so, since a photo can reach upwards of a few gigabytes, which in lower end systems can consume most if not all their DRAM capacity, which is simultaneously being used by other programs. Therefore, programs prioritize what does and what does not go into fast and slow memory.

Despite how fast processors have progressed, growing about 60% every year, memory performance and capacity have not kept up, only 7% every year. Therefore, the performance improvement between processor and memory grows 50% each year.

Considering the performance gap between the processor and the memory, it is important to keep in mind that algorithms need to be done in a local memory hierarchy, so that data does not have to traverse one memory hierarchy to another, which will cause a slowdown in performance.

4. **Explain the different kind of localities in parallel algorithmic design? What is meant by pipelining?**

Temporal locality eliminates memory operations by saving small values in cache and reusing them. Spatial locality takes advantage of better bandwidth by getting a chunk of memory and saving it in cache and using the whole chunk. These two types of localities maximize the reusing of recently accessed data and minimize the volume of data-exchanged and the frequency of interaction with slower memory channels, like the DRAM, allowing parallel algorithms to perform faster.

Pipelining is a form of parallelism, it's like an assembly line in a factory, where one task is done before another. It helps with bandwidth but not latency, a pipeline is slowed down by its slowest stage.

An example of pipelining would be running laundry with different sets of clothes but with only one set of washer and dryer, if it takes 30 minutes to wash clothes, 40 minutes to dry and 20 to fold, you can start another wash cycle once the first set of clothes go into the dryer, then once the first set of clothes leave the dryer, the second set go into the dryer and a third set goes into the washer. The number of sets of clothes you do is the equivalent of your bandwidth, you can do as much as you can despite only having one pair of washer and dryer, but pipelining all of them still does not help with how much time it takes to completely do all the clothes.