Peri Hassanzadeh
ECE 1570 High Performance Computing
Dr. Bigrigg
December 3, 2021

Solution Scaling and Applications

Scaling

Solution scaling deals with how the solution time varies with the number of processors. I will now analyze the scalability of each solution and how it could potentially increase or decrease the performance based on which implementation was chosen.

Before I get into analyzing each solution and its corresponding execution time and relate that to its performance, I wanted to mention that each time is smaller when using the CRC server compared to my local machine. Although the server is a shared machine and the times may differ depending if it is being used by other processes, each time I have tested the times of each implementation, the times were faster than that of my local machine. This excludes MPI since I am unable to run that application on my local machine.

Now I will start analyzing the times of each implementation, mentioning that of the server and my local machine, why I think certain implementations are faster than others, and what I could have done to possibly speed up each implementation.

The serial implementation ran at a speed of 1,600 microseconds on my local machine which is approximately 0.0016 seconds. On the server, it ran at a speed of 1,296 microseconds which is approximately 0.1296 seconds. The serial version is more than two times as fast on the server as on my local machine. In order to speed up the execution, parallel programming is introduced with threads first and then cluster implementations.

The data parallel or thread version ran at a speed of 18,002 microseconds or 0.018002 seconds on my local machine and 649 microseconds or .000649 seconds on the server. Meaning the data parallel version is almost 28 times as fast when run on the server compared to the local machine. In order to improve this implementation, I think I could've created more threads to split up the matrices to be worked on rather than just splitting the data in half and distributing it to two threads. I could have done one thread per matrix and then been able to compile the results in one file after the calculation of each thread. I feel like the more threads present, the less time it would take for each thread to complete its tasks and therefore since they can all run at the same time, the overall execution time would be sped up. When comparing the data parallel implementation to the serial implementation, there is a much greater speed up of nearly 2 times on the server. Although the local machine was much slower. This would lead me to believe that the program has strong scalability since the ability of the parallel algorithm to achieve performance gains proportional to the number of processors being used increases when regarding the server results. When programming in parallel, the number of processors being utilized increases. This is the same for clusters as well, but the results may differ in terms of scalability.

The cluster version ran at a speed of 6.1979284e-04 on the server. I was unable to run this version on my local computer so I won't be able to compare the times from local to server. In terms of comparing clusters to data and serial versions, this implementation is much faster than the serial version and very similar to the data parallel version using threads. The scalability for this implementation is very strong as more processors are being used the program would become faster from my knowledge. With additional time, I would experiment by using additional processors and being able to split the data up differently. I should have done more research on the proper implementation and way to design a program utilizing clusters.

Applications

Serial, data parallel, and cluster implementations can be used for many applications and aid in speeding up the execution of many use cases. By adding additional processors in concurrent use, scalability, performance enhancement, and efficiency are all likely to increase or strengthen. When doing research for this project, I saw many implementations of thread cluster computing. Cluster computing is used by many large companies to manage large amounts of data and workload by being able to work in parallel with many nodes and processors. For example, Capital One uses clusters in the cloud to manage large amounts of data and even manage its applications.

Data parallelism can be used any time there is a large amount of data that needs to have the same operations on it. For example parsing resumes for job applicants and filtering out which applicants are potential candidates for an interview without having to look through every resume by hand. A program could split the resumes into equal groups and have the same operations performed on them and then at the end have the main thread create a list of all the potential candidates that would be likely to have the qualifications to receive an interview.

Serial implementations are very common in many programming examples such as programming a webpage or demonstrating a depth-first search algorithm; these types of applications run each line by line and nothing runs at the same time as each other. In other words, there is no threading, busy-waits, or mutexes used in a serial version of a program. Note that some of these applications could be changed to data parallel or cluster computing in certain scenarios where scalability, efficiency, and performance metrics are important to the developer or critical to the application.