CIS 520 - Project 4 Performance Analysis

Introduction

The goal of this assignment is to learn how to implement threading and parallelization in an application using different tools, including Pthreads, MPI and OpenMP and monitor their performance. In this document, we will be comparing the impact certain specs have on performance, including input and number of cores.

Implementation

The implementation of our application is rather simple, consisting of two helper functions, find_max_ascii and processLines, as well as a main function. The main function initializes the threads, passes them arguments, and as they end, merge the threads to the main one and print the data.

Evaluation

It seems that each implementation seems to do well in certain situations.

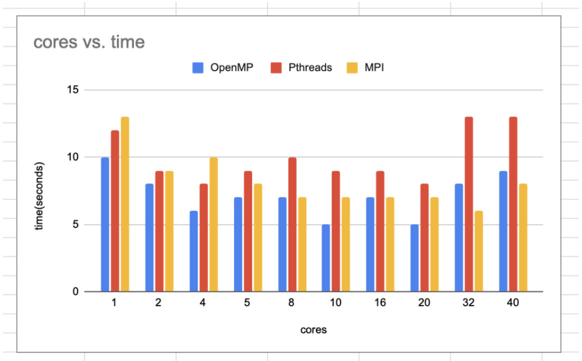


Figure 1. Cores vs Time

Let's start with OpenMP. In terms of speed, OpenMP has the best overall performance. The fastest time occurs at 10 and 20 cores while the slowest time is at 1 core. In terms of memory usage, it seems that each core simply requests 1 gigabyte. Adding nodes seems to negatively affect the performance of the application in terms of speed, spiking from 5 seconds at 1 node with 20 cores to 8 seconds at 2 cores and 32.

nodes	cores	gb	gb requested	time
1	1	0.01	1	:10
1	2	0.01	2	:08
1	4	0.01	4	:06
1	5	0.01	5	:07
1	8	0.01	8	:07
1	10	0.01	10	:05
1	16	0.01	16	:07
1	20	0.01	20	:05
2	32	0.01	32	:08
2	40	0.04	40	:09

Figure 2. OpenMP data

Our team wasn't able to get a very precise reading on the memory used, but at .01 across the board with a 4 times jump at the end makes it clear that more cores and more nodes affect time and memory usage negatively.

Next, Pthreads. Pthreads seemed to perform the worst in terms of execution time, but was by far the most efficient in requesting memory.

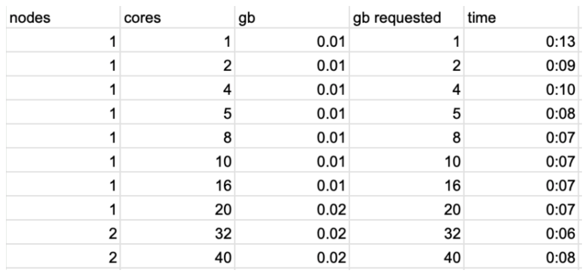
nodes	cores	gb used	gb requested	time
1	1	0.02	0.06	:12
1	2	0.01	0.12	:09
1	4	0.01	0.23	:08
1	5	0.01	0.29	:09
1	8	0.01	0.47	:10
1	10	0.03	0.59	:09
1	16	0.01	0.94	:09
1	20	0.01	1.17	:08
2	32	0.01	1.88	:13
2	40	0.01	2.34	:13

Figure 3. Pthreads data

While both OpenMPI and MPI requested 1 gigabyte per core, Pthreads requested about .06 gigabytes per core. Inexplicably, there were .02 gb used at 1 core and .03 used at 10 cores. 1 core had a spike in run time, but at 10 cores, there wasn't any anomaly in terms of run time.

Finally, we have MPI. MPI displayed its value in Figure 1, where you can see that as the number of cores increased, the run time decreased.

This means that the value of MPI is in its ability to decrease runtime by utilizing multiple cores. The memory requested is still not as good as Pthreads, but is on par with OpenMP.



nodes	cores	gb	gb requested	time
1	1	0.01	1	0:13
1	2	0.01	2	0:09
1	4	0.01	4	0:10
1	5	0.01	5	0:08
1	8	0.01	8	0:07
1	10	0.01	10	0:07
1	16	0.01	16	0:07
1	20	0.02	20	0:07
2	32	0.02	32	0:06
2	40	0.02	40	0:08

Figure 4. MPI data

MPI also has consistent memory used, gradually working its way up to .02 gb at 32 and 40 cores. Due to the lack of precision, it’s difficult to say for sure, but it appears that the memory used is correlated with the number of nodes, as this increase in memory corresponds with an increase in nodes used.