ECE 666
Renato Oliveira
PUID: 033167709

Homework 2 Report
Parallel Matrix Multiplication and Sorting using MPI

1: Introduction

This report explores multi-core processors and parallel programming using MPI (Message Passing Interface) to speed up compute-intensive workloads across computer clusters. This report aims to analyze the performance and the potential issues associated with concurrency.

2: Procedure

Using the same procedure as the "pthreads" assignment, the algorithm divides the matrices into chunks based on the number of processors specified when launching "mpiexec". The algorithm will use MPI to send parts of the matrices to be processed by other processors. Each processor (worker) will start at a specific index into the matrix row/column array. There is no need for synchronization since the chunks are evenly divided, and a worker does not touch another's worker data. As for the sorting, the chosen sorting algorithm is the quicksort since the divided-and-conquer nature of the algorithm can be easily parallelized by dividing the data into blocks and sending them to each processor in the network with a final sort by the main processor on the worker's data.

3: Results

3.1 Matrix multiplication test results

Results show a significant speed-up depending on the number of processors doing the work. Running the matrix multiplication test with one processor, the performance is worse than executing the same code without MPI. This issue is most likely because of the overhead associated with the MPI initialization and message passing. The speed of execution starts to out-perform the sequential work when the number of processors available is increased. Using two processors, the parallel work completes with half the time of the one processor execution. The same can be said when the number of processors grows to four and eight.

3.1.1 Matrix Multiplication results table

DIM 4096X4096	1 processor	2 processors	4 processors	8 processors
Matrix Multi	854175.083 ms	410062.285 ms	208747.153 ms	111555.971 ms

3.2 Parallel Quicksort test results

In the case of the quicksort test, the results are puzzling. Using only one processor, the performance was better than running the same code with multiple processors. The possible cause of this phenomenon is associated with the quicksort parallel algorithm. After the worker processors complete their work, the main processor will perform a final sort to position the chunks of data in sorted order. Running the quicksort on one processor is equivalent to running the quicksort sequentially with only one call to quicksort. When multiple processors are in play, each worker will perform the sort on its data chunk and return it. The main processor will do a final sort to order the data processed by the workers. It clear that using multiple processors requires two sorting phases compared to one sorting stage when running sequentially—disregarding the one processor run, the sorting performance increases as the number of processors increases.

3.2.1 Parallel Quicksort result table

DIM 4096X4096	1 processor	2 processors	4 processors	8 processors
Quicksort	2097.730 ms	3180.404 ms	2712.217ms	2465.803 ms

4: Conclusion

In conclusion, Message Passing Interface is another robust framework to develop parallel software. The advantage of MPI over threads is the ability to scale. Supercomputers rely on MPI to interconnect and divided work across multiple computers to speed up a given task.