2010 Summer Research Project

Elad Mazurek

Project Background

This summer I worked in conjunction with other students on a project whose ultimate goal was to further contribute to the knowledge of how sensors can be networked together and used to gather data. My portion of the work involved behavior prediction and learning how parallelization of execution can lead to performance optimization.

My work

To familiarize myself with the different parallelization techniques I reviewed the Wave2D application and learned how multithreading and MPI (Message Passing Interface) are used to distribute work load across multiple threads or computers respectively. While learning to do so, I also created a new version that included the ability to save and resume simulation progress.

For the next phase of the project I had to utilize the same methodology to a heat distribution prediction application. After converting the existing application from a Java applet to a standalone application, I parallelized the program using both multithreads and MPI. I also added the same save and resume modifications to the MPI version, as well as a testing mode to both versions, to help analyze any performance improvements that may have been made due to the addition of parallelism.

Results

Heat2D Multithreading

As the test results below show, compared to the sequential execution version of the same Heat2D application, I found the multithreaded execution to run at least 27% faster. breaking up the simulation space into “stripes” allowed all calculations to happen simultaneously, and resulted in faster run times.

Heat2D MPI

Unfortunately, I was unable to show performance improvement using MPI because of the communication overhead that MPI incurs during execution. Though this was the case, I believe that for larger test sets and problems that involve more processor intensive computations, using MPI would be beneficial. This is based on the observation I made while testing, that even though MPI was 30 times slower to execute a method once, after 1000 intervals it almost completely caught up. Also, MPI test results from the Wave2D application show significant improvements when using MPI.

Wave2D Multithreading

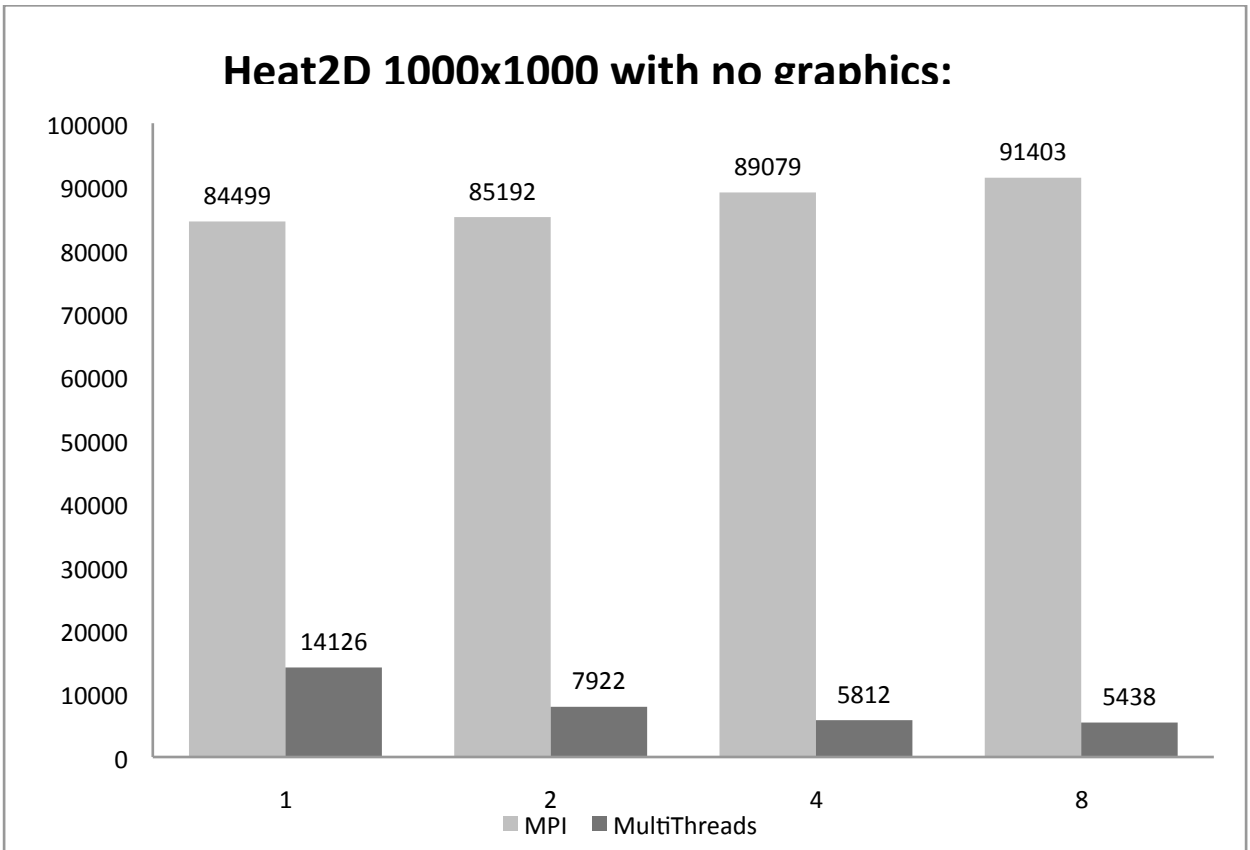
Results follow the Heat2D example. Significant run time improvements can be seen by increasing the number of threads running the program

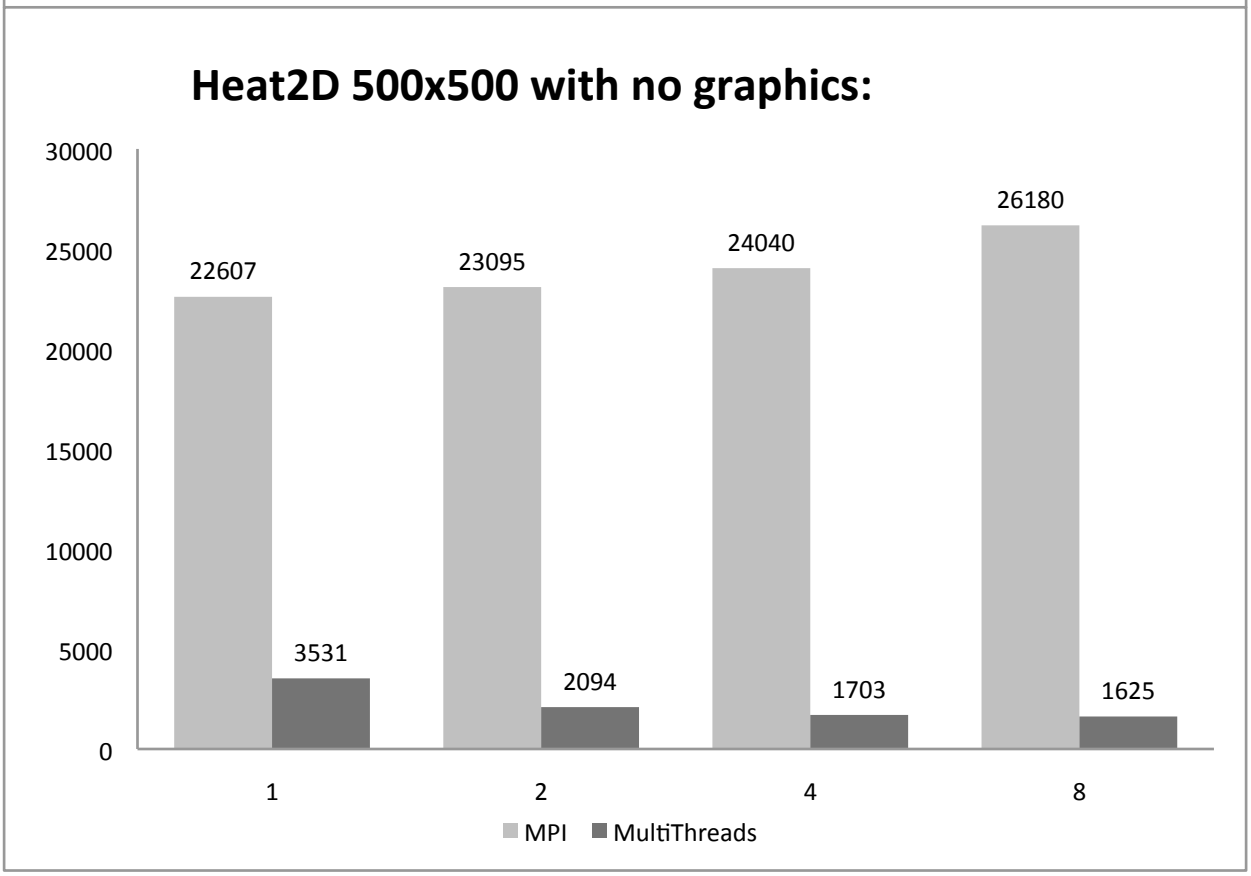
Wave2D MPI

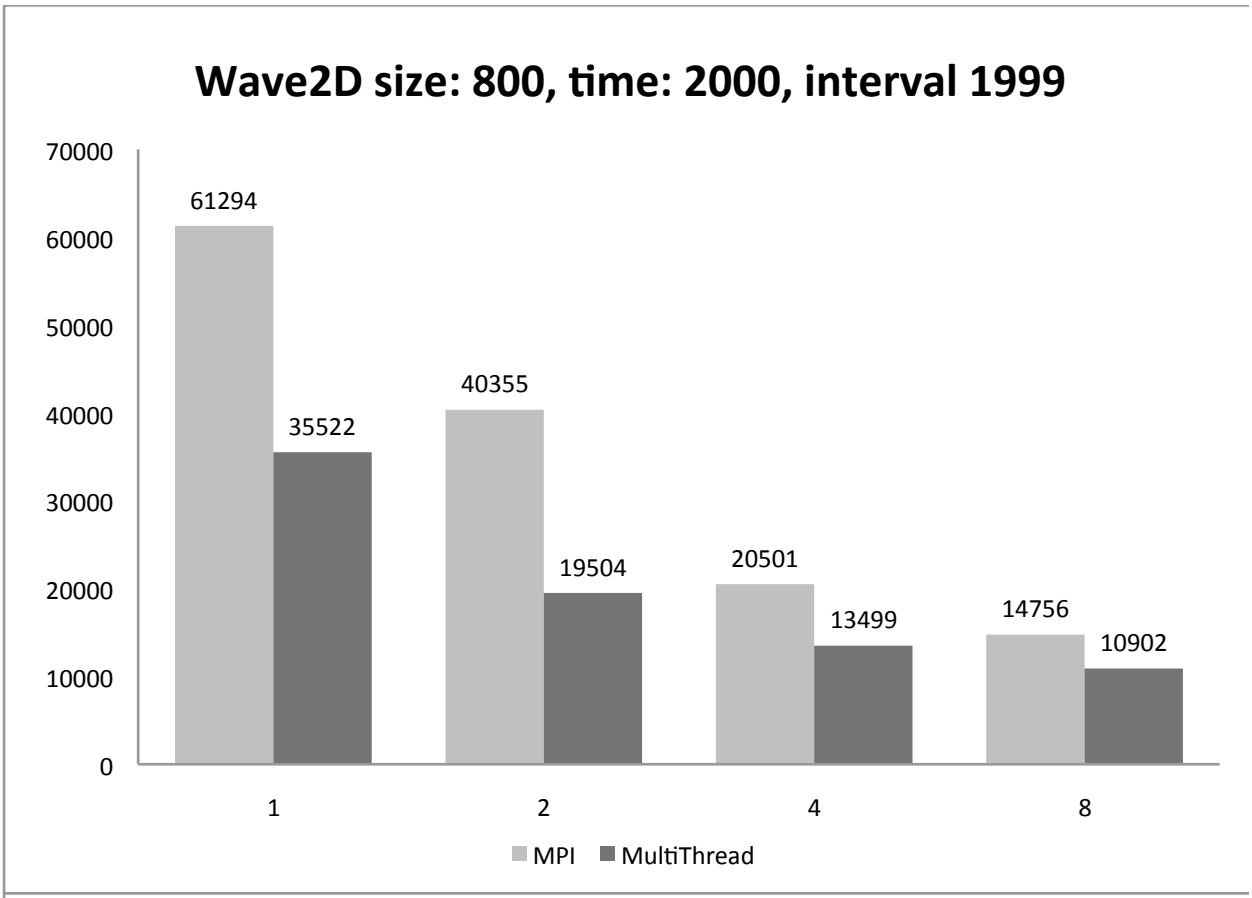
Run time improvements were quite noticeable using MPI with this application. In the larger data set, going from one rank to two decreased the run time by 34%. Dividing up the workload between 8 computers decreased the original run time by 76%.

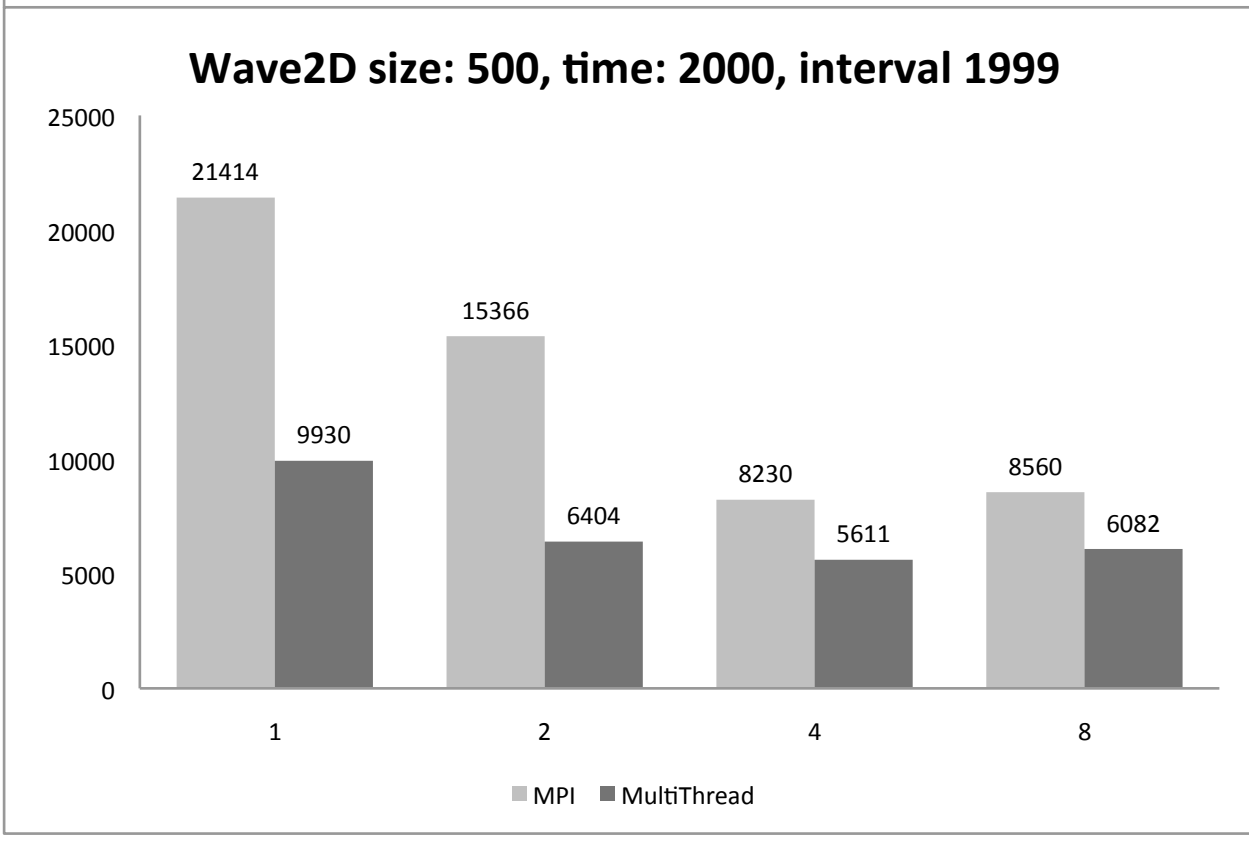
Test result Graphs

Attached.


Heat2D 1000x1000 with no graphics:
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
84499
14126
85192
7922
89079
5812
91403
5438
1
2
4
8
MPI
MultiThreads


Heat2D 500x500 with no graphics:
30000
25000
20000
15000
10000
5000
0
22607
3531
23095
2094
24040
1703
26180
1625
1
2
4
8
MPI
MultiThreads


Wave2D size: 800, time: 2000, interval 1999
70000
60000
50000
40000
30000
20000
10000
0
61294
35522
40355
19504
20501
13499
14756
10902
1
2
4
8
MPI
MultiThread


Wave2D size: 500, time: 2000, interval 1999
25000
20000
15000
10000
5000
0
21414
9930
15366
6404
8230
5611
8560
6082
1
2
4
8
MPI
MultiThread