Project 4 Report – MPI Hotplate

Kyle Pontius

Introduction

Parallel processing is a powerful tool that requires careful planning. If we want to tackle large problems, with lots of data, we have two excellent technologies that we've touched on this class. First are multithreaded applications leveraging OMP or p-threads. This consists of two or more threads running simultaneously on a single machine, preferably with a multi-core processor. Second is networked, distributed processing using MPI. This distributes pieces of a problem to multiple machines, which then return a piece of the result to be merged.

For me, this problem posed several interesting questions, some of which were: How will MPI's communication time stack up against OMP's nearly instant performance? Do separate nodes with a given problem size perform better than separate cores (working on a thread) with that same problem size? Which type of parallelism's performance degrades more quickly as problem size significantly increases? While not each of these questions will specifically be answered, there were many that were. This was a fun and meaningful learning activity for me.

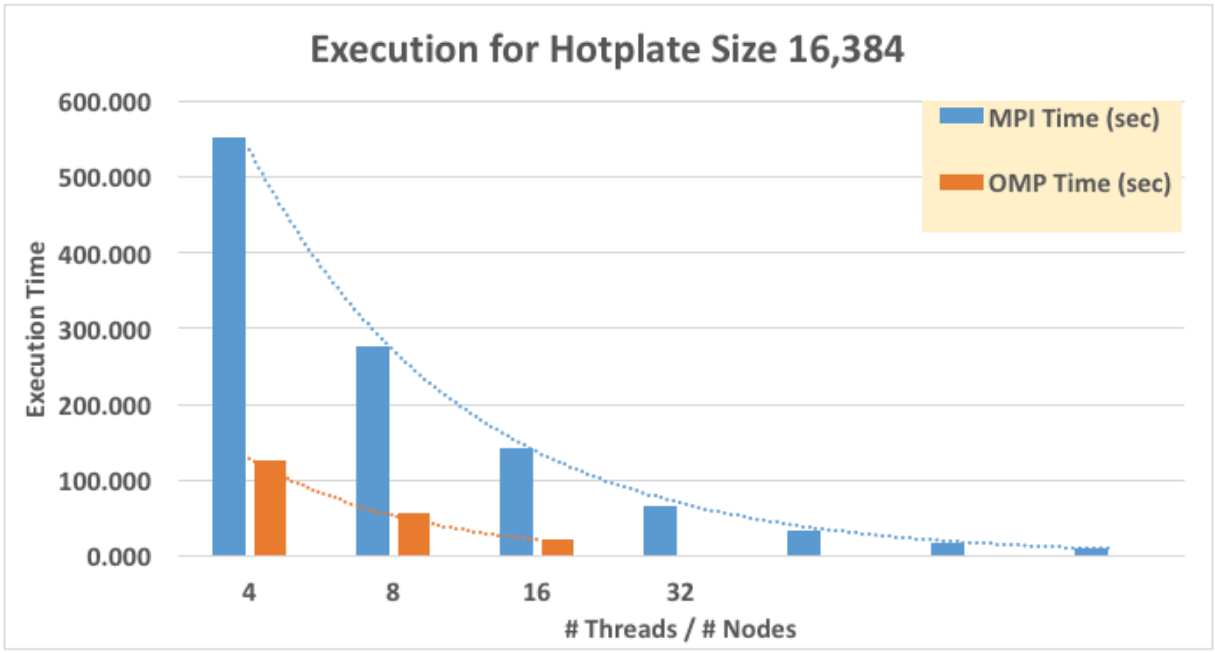
Approach

Please note, as I haven't finished the p-threads project my results will focus on the performance results for OpenMP and MPI. My approach for graphing the results is to compare the number of nodes for MPI and the number of threads for OMP. While this is not an excellent apples-to-apples comparison, I did find it enlightening to view the results with this perspective. Even though the communication time with MPI was significantly higher than that of OMP, relatively speaking, the computation power seems reasonably comparable.

Experimental Setup

For my tests, I've decided to run both the OMP code and MPI code on the FSL Supercomputer. In an effort to minimize performance noise and inconsistencies from the results, I've run every test twice and reported the average. In my experience, the results were very consistent, always falling within a few seconds of each other. I believe this will help the results more accurately reflect the real execution time of the program. Additionally, in order to get a good feel for the performance of MPI I also ran up to the highest number of nodes FSL would allow me to do within my scheme, which turned out to be 256 nodes. OMP, however, was limited to up to 16 threads as the FSL didn't offer available nodes with a higher thread count. The last thing worth pointing out is that I limited the lowest result on OMP and MPI to 4 threads/nodes because MPI's results timed out at 10 minutes below 4 nodes. This felt like a

reasonable low water mark as the trends are easily visible before going below that point, and we covered execution time below 4 threads on our Project #1 report.



Results

My hotplate's relaxation took about 360 iterations. Looking more closely at the graph there are three elements that really jump out. First, both execution times decrease at a rapid rate; with MPI decreasing a little faster than OMP (that fact is difficult to see from the graph, so I included the numbers themselves ***after*** the report's conclusions). The trend line placed on both graphs is exponential, with MPI following the trend line more closely. If this doesn't make much sense, please let me clarify. As you can see from the code, my start and end time for the MPI Hotplate implementation begins after much of the "serial" code has been executed. This execution time does ***not*** include much of the code that initializes MPI itself. In essence, this means that the execution times really represents the speedup in the parallelization portion of the code, not the serial portion.

Second, the execution time for both OMP and MPI's graphs behave in very similar ways, decreasing at nearly the same rate. Specifically, the difference in speed between 4 and 8 threads is very similar to the change in performance for 16 and 32 nodes. This leads me to believe that the speedup achieved by both, when we minimize or disregard communication time altogether, is effectively very comparable for some problems. This is interesting to me because it answers one of my initial questions about which performance degrades more quickly, mentioned previously in the introduction. Also, it concretely depicts to us that while MPI and OMP effectively work differently, they scale in a very similar way. Understanding this,

and the effect scaling has on either implementation, can really help in determining which is the better tool to use for a given situation. After using both, problems with large datasets and relatively few chunks lend themselves well to multithreading with OMP; however, when working with datasets that can be broken up into hundreds of pieces, MPI is very robust.

Third, the differences between OMP and MPI's performance speak volumes. One difference is their ability to scale. OMP effectively can only go up to the number of threads the hardware supports. Generally, this is reached very quickly at scale. However, the communication time between threads is extremely small, relative to MPI, and lends itself well to small, but parallelizable tasks such as image editing. MPI, on the other hand, can scale to immense proportions. I was able to stretch to 256 nodes, which is genuinely impressive. Imagine what the potential could be given a much higher number of nodes to work with. However, the communication cost for such a large network demands larger problem sizes to be worthwhile. Obviously, in our case, the problem size would have to increase significantly to be effective at scale, but the potential computation power is very exciting!

Conclusions

Three points really stood out to me in reviewing the data from my application. First, speedup with this embarrassingly parallelizable problem is enormous, for both OMP and MPI. Second, the execution time's growth is very similar for both MPI and OMP, leading me to believe that both are similarly robust given the proper problem size and data set. Third, because of the limitations of hardware, software, and the problem itself, effective planning practices are crucial to deciding between OMP and MPI; both have evident strengths and weaknesses.

Additional Results

	# Threads / # Nodes						
	4	8	16	32	64	128	256
MPI Time (sec)	559.785	277.246	141.482	72.176	32.816	16.460	10.730
	543.710	274.417	142.263	60.120	32.434	17.299	9.560
Avg MPI Time	551.748	275.832	141.873	66.148	32.625	16.880	10.145
% Difference From Prev		50%	49%	53%	51%	48%	40%
OMP Time (sec)	122.994	56.546	34.269	-	-	-	-
	127.590	56.638	36.644	-	-	-	-
% Difference From Prev		56%	35%				
Avg OMP Time	125.292	56.592	35.457	-	-	-	-