Jonathan Gregory

CS 441

Final Project

Functional Programming / Concurrency

December 10, 2016

Summary and Explanation of the Results of the Final Project

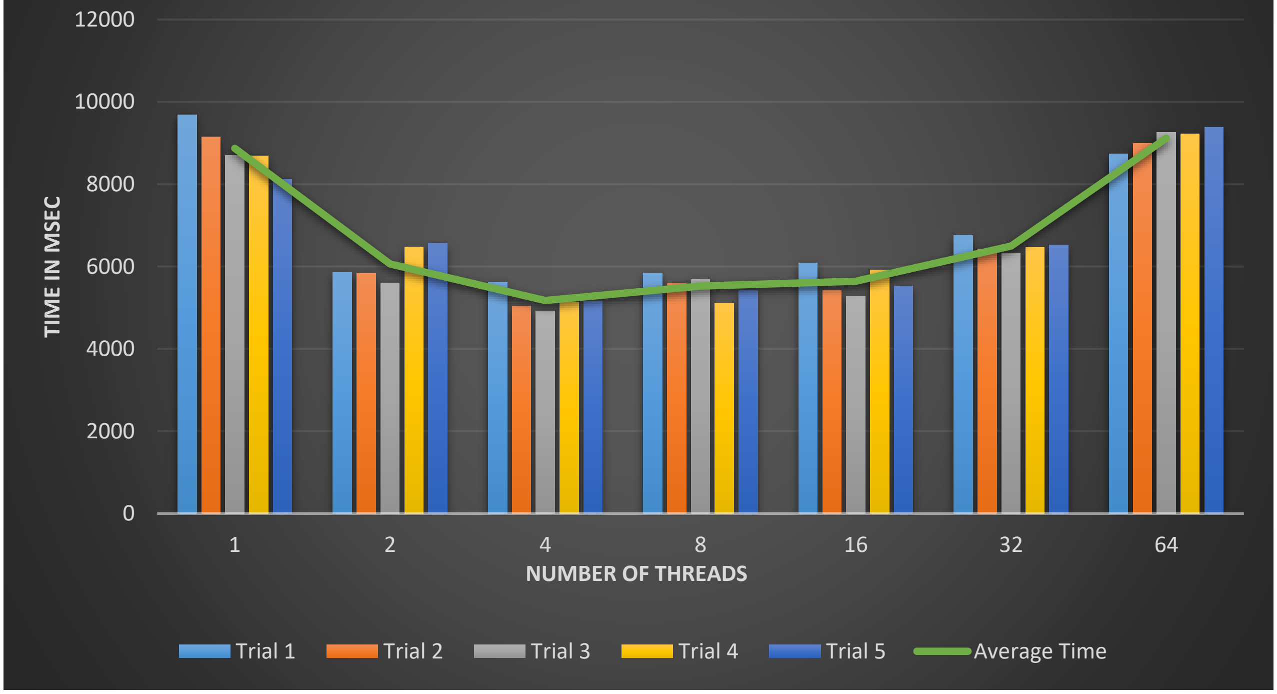
I created a program in Clojure that would read in the numbers_txt.txt file containing 1,000,000 numbers and performed a mergesort on it. The program repeated the mergesort with 1, 2, 4, 8, 16, 32, and 64 threads. I noticed that the execution time tended to decrease as the number of threads increased. However, I observed that after 4 threads, the execution time would marginally increase after that. I was interested to find out how much the execution time would increase if it had 64 threads so I added that into the program. I found that having 2, 4, 8, 16, and 32 threads all executed faster than just one thread alone. When I increased the number of threads to 64 I found that it took longer to finish than just having one thread run the mergesort unaided. I wanted to continue to test my theory by adding even more threads; however, I found that my program would not compile when I tried to run 128 threads. I assume that it was a limitation due to my hardware or the program itself. I have recorded the results below as seen in Figure 1 and graphed the time as a function of the number of threads as seen in Figure 2.

I hypothesize that since the optimal number of threads to perform my task per my results was 4 and since I ran this on my laptop with an i5-5200U processor with 4 logical processors, adding any more threads than my computer had processors to compute caused the threads to queue up and cause a significant amount of overhead time for each thread running. Per an article by Brad Cypert, the overhead time for spinning up multiple threads can be relatively high, especially on lower end machines. I suspect that the overhead time is the same for 1, 2, and 4 threads since they are all able to run concurrently on my computer. Adding more than the 4 threads causes additional overhead time increasing the total time to finish the operation.

Figure 1: Time Trials in Milliseconds for Execution of Mergesort VS Number of Threads

Threads	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average Time
1	9689.739792	9152.869057	8694.95842	8692.106976	8110.855037	8868.105856
2	5854.736817	5827.328987	5604.66159	6479.50501	6555.128005	6064.272082
4	5610.6346	5046.539382	4916.018148	5134.389251	5164.219784	5174.360233
8	5849.182286	5585.310768	5688.734881	5098.189622	5417.616721	5527.806856
16	6091.027421	5414.945824	5273.681289	5918.48517	5530.065432	5645.641027
32	6755.975227	6426.644118	6334.283815	6460.818531	6520.73198	6499.690734
64	8732.760121	8988.045581	9264.959003	9217.774874	9378.677614	9116.443439

Figure 2: Time as a Function of the Number of Threads



Resources Used on the Project:

leontalbot. 2012. Clojure : How to define a function that prints a string without putting quotes to variable. (February 2012). Retrieved December 9, 2016 from http://stackoverflow.com/questions/9107043/clojure-how-to-define-a-function-that-prints-a-string-without-putting-quotes-t

Brad Cypert. 2016. Understanding Clojure's Map & PMap. (June 2016). Retrieved December 9, 2016 from http://www.bradcypert.com/understanding-clojures-map-pmap/

John Gabriele . Introduction to Clojure. Retrieved December 10, 2016 from http://clojure-doc.org/articles/tutorials/introduction.html

Rich Hickey. doseq. Retrieved December 10, 2016 from https://clojuredocs.org/clojure.core/doseq

Kim Hirokuni. About. Retrieved December 10, 2016 from https://kimh.github.io/clojure-by-example/

Alexei Sholik. 2012. Mergesort in Clojure. (March 2012). Retrieved December 9, 2016 from https://gist.github.com/alco/2135276

Chris Wphoto. 2016. Multi Threaded Merge Sort in Clojure 2 - 4 - 8 - 16 - 32 Threads. (May 2016). Retrieved December 10, 2016 from https://gist.github.com/ChrisWphoto/c6f86f34cd7ade5055aec743ec990765