Multi-Threaded Efficiency Study

By: Joseph Hales

CS 345

3/23/16

Multi-threaded programming is difficult and sometimes counter intuitive. One might ask if it is worth the extra development time, and chance for errors or bugs. A primary focus of such a discussion would revolve the potential speed up by using multi-threading. Another concern would be how many threads to use for a given application. This white paper will demonstrate an experiment with CPU and IO intensive applications with variable number of threads on several different systems to analyze the reward for using multi-threaded programming.

Experiment Design:

The experiment will be designed to showcase the potential speed up of threads in two situations, IO intensive and CPU intensive programs. The IO program will simulate normal I/O using waits of various sizes, and the CPU intensive program will perform the computational intensive work of multiplying matrices. For I/O, we will use a list of 1000 different I/O operations with different hard coded waits. For matrix multiplication, we will use 2 matrices size 1000 by 1000 filled with random numbers. For both, we will test how changing the number of threads used affects performance. We use 1, 2, 3, 4, 5, 6, 7, 8, 16, 32, 64, 128, 256, 512, and 1000 as our data points for number of threads.

The experiments will also test the effectiveness of threads on various systems. We use systems with the following CPU configurations:

* Intel® Core™ i5-2400 CPU, 3.1 GHz, 4 processors, non-hyper-threaded (System 1)
* Intel® Pentium® 4 CPU, 3.4 GHz, 1 processor, hyper-threaded (System 2)
* Intel® Core™ 2 6400 CPU, 2.13 GHz, 2 processors, non-hyper-threaded (System 3)
* Intel® Xeon® E5345 CPU, 2.3 GHz, 8 processors, non-hyper-threaded (System 4)

Expectations:

I have the following expectations:

1. Concurrent thread count increase will only increase performance up to the number of logical cores in the system for computationally heavy work.
2. Increasing the number of concurrent threads beyond the number of logical cores will not impact performance much, due to the amount of overhead required is insignificant compared to natural time (as in time perceived without the aid of tools).
3. Older systems will not be able to handle the large amount of threads some of these tests use, resulting in segmentation faults.
4. Increasing concurrent threads for I/O will always increase the speed at which the tasks are completed, to a minimum of the longest I/O job plus overhead (or if applicable the minimum would be reached at the highest thread count a system could handle.

Experiment Results:

Within each system, threading followed an expected pattern. For CPU intensive testing, we found that all systems increased performance most when threads increased to the number of cores. After that, System 2 and System 3 continued to increase as the number of threads increased. System 2’s and System 3’s performance improvement should be attributed to reaching the maxium number of threads, resulting in early termination of the program, and their later improvements should be discarded. As an aside, for systems 1 and 4, the number of cores was no longer the bottleneck, but rather the overhead involved. With future experimentation, it would be recommended to involve more computationally heavy functions to use as a test to really show the advantage.

When getting a closer view of the computationally heavy tests with lower thread counts, we can see that the greatest gain is having a second thread for all systems. After that, matching the number of logical cores is best. System 2 demonstrates interesting behavior when used with odd numbers of threads greater than 1. It could be attributed to “fitting” thread computation together between threads, since major elements of the hyper-threaded CPU is only available to one thread at a time. The next test point was at 16 threads, but it might be best if tests 8-16 were performed, just to see what would happen to the 8 core machine.

As for I/O heavy tests, the overall graph is quite clear: the more threads the better, so long as the system can handle it.

Conclusion:

After looking at the data we find that these experiements lend strength to the proposed idea, that the general rule of thumb for using threads is to match the number of logical cores in the system for computationally heavy work, or to use as many as the system will allow for I/O heavy work.