White Paper CS 345

By David Lambertson

Operating Systems

April 1st, 2016

1. **Introduction**

CPU times, number of threads, type of system, and the size of the workload. All of these factor into how quick a program or process can run. But what difference do they really make? In this paper, we will discuss these different factors to determine how they affect different processes. The two questions we will discuss are “What should be considered when determining the number of threads that a CPU intensive program should use?” and “What should be considered when determining the number of threads that an IO intensive program should use?”

1. **Background**

In the many tests we conducted, we took different variables into account. These are: type of system, the type of workload, the size of the workload, number of threads, and the real-time it took for the process to finish executing. We used two different programs, named ‘doIOjobs’ and ‘matrixMult’. ‘doIOjobs’ simulated different input/output tasks by using the usleep() function. It varied the time as to give lifelike simulation of different types of input/output tasks. The user is able to specify the number of threads to use on each run of this program.

The ‘matrixMult’ program lets the user specify how big each matrix is on the command line and how many threads will be used to run the program. It then fills the matrices with random data and does matrix multiplication.

Not only can we change the parameters to each program, but we had access to different types of systems each running different processors and different settings. We used this to also see the difference systems can have on the process run time.

By running these two programs with different parameters, we are able to use the command ‘time’ to basically have a stop-watch time them. We can take this data and form some conclusions about what kind of factors you should use to determine the correct number of threads.

The command we used to collect our data is “([doIO or matrix].sh) 2>&1 | grep "real" | cut -c6- > file.txt”. We created a shell script which ran our command a total of 128 times, increasing the thread count each time. For my data though, I only use the first 8 data points. We then piped the time message into a file for later use.

1. **Solution**
   1. **What should be considered when determining the number of threads that a CPU intensive program should use?**

To answer this question, I will be using the Matrix Multiplication program given to us. Through this I can change the sizes of the matrices, the number of threads and the system I run the program on. I started out by running it on a computer named Jordan 2008 server, having the specs of Intel® Xeon® E5345 CPU, 2.3 GHz, 8 processors, non-hyper-threaded. Running the program on this machine, on average it gave me the speed up time of 1.351087 from the first 8 runs. Averaging all 128 runs gave me an average close to 1 which is interesting. It is interesting because it means overall, it didn’t have a good speed up so we can conclude that once we hit having 8 or more threads, the speed up of the process is minimal.

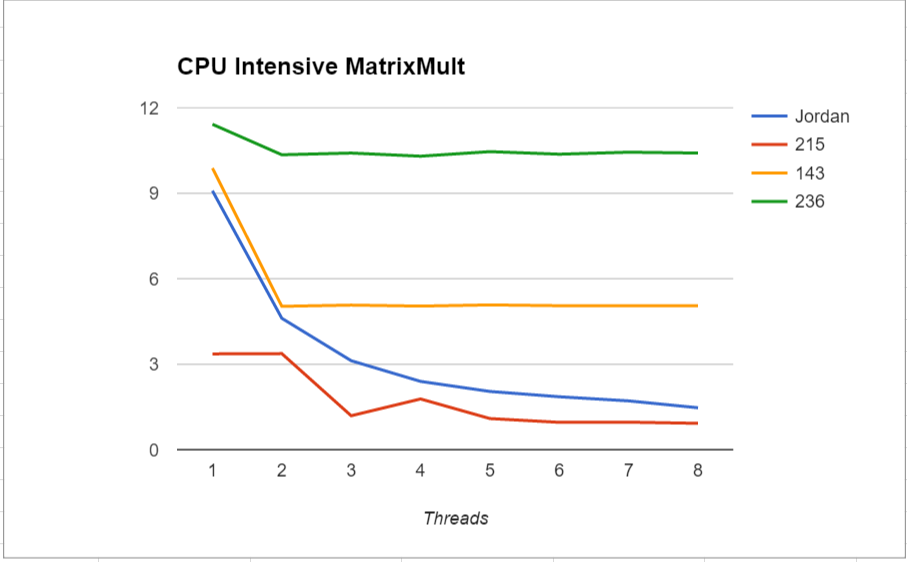
Running the exact same process on a different computer named 215 n Figure 1 with the specs of Intel® Core™ i5-2400 CPU, 3.1 GHz, 4 processors, non-hyper-threaded the results were quite different. Looking at the data over the 128 runs we got skewed data with off shoots here and there. I think it is that the processor was also running other programs.

Figure - Matrix Multiplication CPU Intensive

In total, we did tests on four different machines, including the two previously spoken about. The other two systems are Intel® Core™ 2 6400 CPU, 2.13 GHz named 143 in Figure 1, 2 processors, non-hyper-threaded and Intel® Pentium® 4 CPU, 3.4 GHz, 1 processor, hyper-threaded named 236 in Figure 1. In Figure 1 we have the data from the first 8 runs of each machine. Looking at the stats of each system and the graph, I can conclude the number of processors also affects the increase in speed. Through these tests, I can see that for CPU intensive processes it is not necessary to increase the number of threads above at least 16 to get an increase. Any higher and the time does not get any better.

* 1. **What should be considered when determining the number of threads that an I/O intensive program should use?**

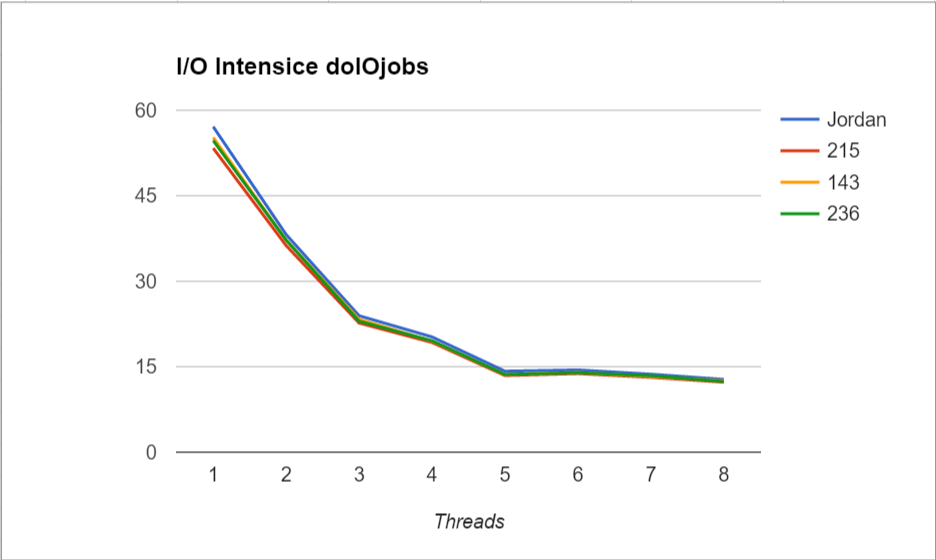
With I/O intensive programs, we did a similar thing. We made a script to run our “doIOjobs” program. We wrote these all to files and used our data to come up with speed up times. We ran our test on the same four systems as we did for the CPU intensive program. The graph we were able to get from our data can be seen in Figure 2. As we can see from the data, the change between systems is minimal, but they all come close to being the same speed by the time we have 8 threads. We can conclude that the number of threads can decrease the time when doing heavy I/O tasks, but only to a certain point as with the CPU intense programs.

Figure - I/O Intensive

1. **Conclusion**

After all these tests and analyzing the data my conclusion is that while threads help to increase the time, once you hit about 16 threads the time that the process speeds up is so minimal there is no point in going above that number. It is interesting to see the numbers on different computers too. Looking between the two, it seems that both were able to average around the same speed up time. Overall though, it looks like the system with more processors had a higher average speed up time. Adding more processors can help with CPU intensive, but does not change much when it comes to the I/O intensive program.