3/30/2021

Michael Heninger, with data contributions from Chatlen Wilkinson and Daniel Strickland

BYU-I, CS345

Picking a thread-count

A white paper considering multiple factors

# Abstract:

This white paper investigates and discusses the various factors that can affect how many threads of execution are optimal for a given program. Specifically, we consider the bounding factor of the program’s performance (IO or CPU operations) as well as the number and type of hardware threads available. For IO-bound programs, we find that there is no specific limit on the maximum number of execution threads to be used, but there are diminishing returns for more threads and the speed of said operations and the potential for cache thrashing must be considered. For CPU-bound programs, we find there is no advantage to having more threads of execution than there are hardware threads available, and some hardware architectures offer no advantages for exceeding the number of hardware cores, depending on the math operations done in the program. For smaller-scale CPU-bound tasks, a limited number of hardware threads is acceptable due to little absolute-time inefficiency. For larger-scale CPU-bound tasks, increasing the number of hardware threads available (by upgrading/adding CPUs) is advantageous as it will significantly affect the absolute (non-percentage) amount of time used.

# Introduction

This white paper investigates and discusses the various factors that can affect how many threads of execution are optimal for a given program. Specifically, we consider the bounding factor of the program’s performance (IO or CPU operations) as well as the number and type of hardware threads available for use in execution. We investigated using five different hardware configurations and two test programs, one with simulated IO bounds and one with CPU bounds. Additional details on our experiments, as well as our analysis and conclusions are described further below.

# Background

Different programs make use of system resources in different ways. Some programs are purely deterministic and sequential, and thus can make very limited or no use of parallelization. Other programs and applications lend themselves toward parallelization quite well, as multiple parts of a program are independent of each other or can be completed in any order. The limit of optimization via parallelization can be found using Amdahl’s Law (Amdahl, 1967). Parallelization will never be complete for most programs, as there typically must be a section of code combining the results from various parallel threads. Therefore, there will be diminishing returns with increasing the number of execution threads, until some optimal solution for the given program is found. This is something further explored in (Ramalakshmi & Kompala, 2017), specifically in regards to some image-processing algorithms. We must also consider the hardware the program is running on, as different hardware can have very different characteristics with regards to multicore execution and SMT (multiple hardware threads per core) (Saravanan, 2013).

# Experiment

In this experiment, we used five different hardware configurations. Their relevant details are:

* LinuxLab: Xeon e5-2670 V3 @ 2.3GHz (virtualized to 2 cores) 2 cores 4 threads
* Michael’s (battery): i7-8750H @ 2.2GHz in power-saving mode 6 cores 12 threads
* Michael’s (wall): i7-8750H @ 2.2GHz in high-power mode 6 cores 12 threads
* Chatlen’s: i7-7700HQ @ 2.8GHz 4 cores 8 threads
* Daniel’s: i7-7500U @ 2.7GHz 2 cores 4 threads

We tested them each using standard scripts that ran two test programs with varying parameters, and recorded the user, system, and real times for each run. Those parameters were:

* I/O bound test (simulating I/O with sleeps), 1-16 threads (later extended to 1-64 threads)
* CPU bound test, multiplying 2 500x500 matrixes, 1-16 threads
* CPU bound test, multiplying 2 1000x1000 matrixes, 1-16 threads
* CPU bound test, multiplying 2 2000x2000 matrixes, 1-16 threads

For further details on the scripts and test programs, as well as the collected data, see the link found in Appendix A.

# Results

Figure 1 (below) shows us the combined results of all our tested systems on the IO-bounded test. As we can see, all of our systems took almost exactly the same amount of time for each number of threads to complete this task, indicating that this test is not hardware-bound. There is also a general trend, indicated by the trend line, that shows more threads tends to be better for this IO bounded task. This remains the case no matter how many hardware threads are available, as each thread spends a lot of time waiting in sleep. After our initial results, we later checked the trend out to more threads of execution, where it appears to continue and nearly asymptotically approach the minimum time of 2.97 seconds, as expected for Amdahl’s Law. Both our user and system times were negligible in these tests, though we might expect as the number of threads increases significantly for these to go up. For IO bounded tasks, it is therefore preferable to have many more execution threads than hardware threads, up to a limit of the number of tasks to be done assuming no groupings of tasks.

|  |
| --- |
|  |
|  |
| Figure 1: IO-intensive real-time results |

Figure 2 (below) shows us the amount of time each of our systems spent on a CPU-bounded task per number of execution threads, as well as each system’s total core count (solid) and hardware thread count (dashed). From these graphs, we can see each task’s time was improved with more threads, up until about the number of total cores. In two cases there were some additional gains up until the number of hardware threads was reached. From this data, it is clear that the limit of appreciable improvement in real-time is the number of execution threads matching number of hardware threads available. As some resources are shared between hardware threads on the same core, some systems see no time benefit from going above the number of hardware cores as well, necessitating testing specific workloads on actual hardware to definitively determine the ideal number of threads. There was no appreciable difference between smaller and larger tasks, which is expected as our CPU-bound test has very little serial overhead in comparison to the main, parallelized task.

|  |  |
| --- | --- |
|  |  |
|  |  |
| Figure 2: CPU-intensive real-time results | |

Figure 3 (below) shows us the trends in user-time on our CPU-bound tasks. The graph shows that total CPU time for each task was nearly constant despite the number of cores (the CPU time was just split into smaller “chunks” to reduce real execution time). However, these trends slope slightly upwards, indicating that using fewer threads is ultimately more efficient in CPU time, and thus if real-time is bounded (as we discussed above) the number of threads should be minimized to that bound to maximize efficiency. Our user-time data for our IO bound task was so small as to be immeasurable, but we would expect something similar as our number of threads increases. Our system-time usage on both tasks was also negligible as to be immeasurable, as we used very few system calls. If we were to use more system calls (as would be necessary to create significantly more threads) we might also expect an upward trend, as system time is used to spool up execution threads.

|  |
| --- |
|  |
| Figure 3: User Time trends |

# Conclusion

In this whitepaper we investigated the performance ramifications of increasing the number of execution threads a program has. We found that for IO-bounded programs/tasks, the ideal number of threads is tied to the task being done and not necessarily to the hardware, and adding more execution threads will asymptoticly approach the ideal (assuming there is no cache-thrashing or other issues of efficiency). Only if one hardware thread is completely active servicing requests would additional hardware threads be appreciably helpful. For CPU-bound tasks, maximum efficiency is found when the number of execution threads is less than or equal to the number of hardware threads available. As we found, not all architectures can allow programs to run equally with multiple hardware threads per core, so testing with a specific hardware setup and actual program would be necessary to completely optimize a solution. For scalable parallelized CPU-bound tasks, adding additional hardware threads will increase performance, but with diminishing percentage returns as predicted by Amdahl’s Law (Amdahl, 1967). Thus, if tasks are kept small, then a smaller number of hardware threads is acceptable due to smaller absolute (non-percentage) time loss. If tasks are very large, then more threads would appreciably decrease the absolute time taken.

# Appendix A

See <https://github.com/cougarEngineer/Threadcount-whitepaper> for all related code and raw data.

# References

Amdahl, G. M. (1967). Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference* (pp. 483–485). Atlantic City, New Jersey: Association for Computing Machinery.

Ramalakshmi, E., & Kompala, N. (2017). Multi-threading image processing in single-core and multi-core CPU using R language. *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)* (pp. 1-5). IEEE.

Saravanan, V. a. (2013). Performance Analysis of Multi-Threaded Multi-Core CPUs. *Proceedings of the First International Workshop on Many-Core Embedded Systems* (pp. 49–53). Tel-Aviv, Israel: Association for Computing Machinery.