Multithreading Analysis
Nov 29, 2018

Table 1 shows the execution time of different threads for different array sizes. The size of the array is the value of n squared. As shown below, the threads did not execute for the same time for the same array sizes. Figure 1 shows the speedup vs thread graphs. It demonstrates that multi-thread programming is not a one-size fits all solution to improve the speed up of programs.

value of n	Execution time (s)	Thread
256	0.321479	0
512	1.172866	0
1024	25.363683	0
2048	423.536973	0
4096	4475.77321	0
256	1.304	2
512	1.405	2
1024	12.618	2
2048	210.576	2
4096	1751.343	2
256	1.307	4
512	1.04	4
1024	40.828	4
2048	229.592	4
4096	1440.41	4
256	0.418	6
512	0.411	6
1024	17.829	6
2048	218.093	6

4096		6
256	0.796	8
512	1.313	8
1024	40.666	8
2048	216.672	8
4096	960.1222*	8

Table 1:

Note: The array of size 4096 for threads=6 was throwing segmentation faults and the simulation for threads=8 stopped after 960.122 seconds without actually completing the whole task.

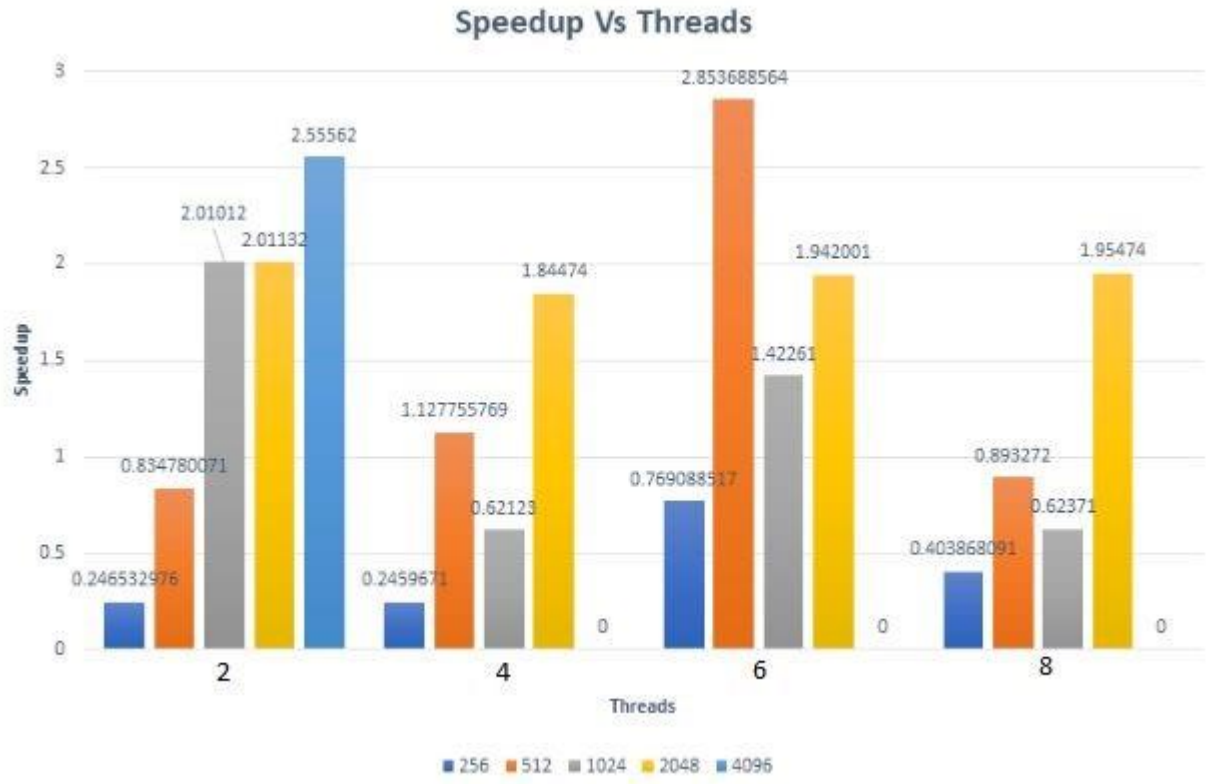

Figure 1: speedup vs threads

For small array sizes, as the number of threads increases, the execution time becomes worse. An example of this is array size 256 squared, which runs in 0.321 seconds without threads and 1.304 seconds with two threads. For large array sizes, as the number of threads increases, the execution time becomes better. An example of this is array size 1024, which runs in 25.363 seconds without threads and 12.618 seconds with two threads. Similar pattern is observed for

execution times of threads on various arrays. This because, some computation completed faster by fewer threads. Adding a large number of threads to computation that can be handled by fewer threads adds time overhead as threads have to switch to run the critical code.

Figure 2: Execution time vs threads

According the data obtained, the best speedup is found to be for the array size=512 with 6 threads. As we keep increasing the number of threads for a size, it maximizes the resource utilization at the cost of more hardware support. The tradeoff is usually between performance and cost. In the mentioned scenario, the tradeoff seems to be well balanced hence resulting in a speedup better than the ones achieved elsewhere.

Multi-thread programming is one way to improve the performance of parallel programs; however, there are other techniques to improve the performance of parallel programs. It should be kept in mind, that trying to use parallelization on serial programs will not improve the performance, as seen with small arrays with n > 0 threads. Nonetheless, multiprocessing and multitasking are two other approaches to improve programs which benefit from parallelization. Multitasking refers to the ability of an operating system to execute multiple processes on a single CPU, and multiprocessing is the execution of one program on multiple CPUs. There are also other research discussing ways to improve parallel execution of programs. One example is the Message-Passing-Based Parallel Programs on virtualized multi-core processors. It is a way to control the compiler to optimize tasks running in parallel.