This is a program that computes a hash value for an input file using multi-threading to improve the execution time. The program allows the user to input a file name and a desired number of threads to assist in the hashing of the file.

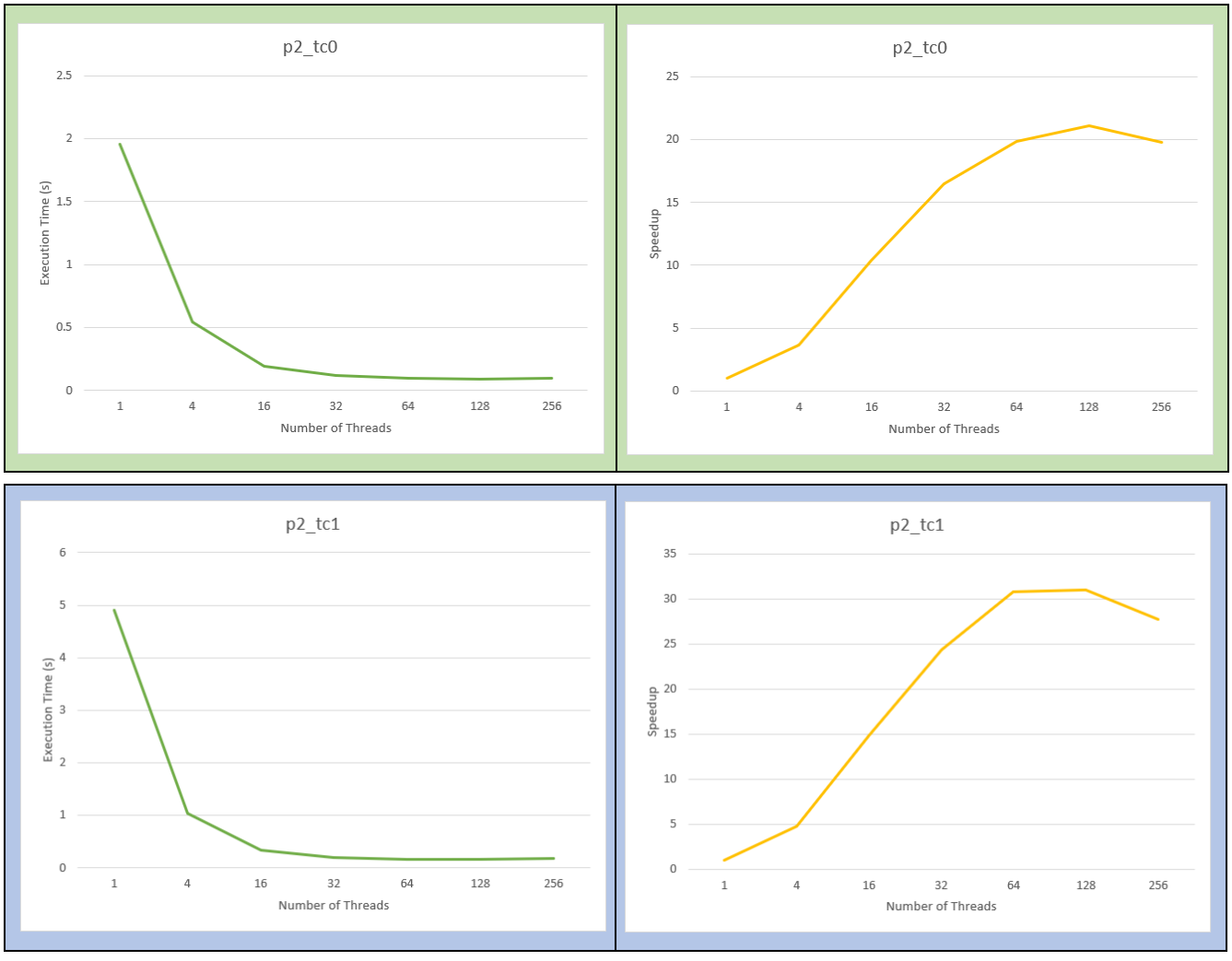
The experimental set-up was on a server which has the following specs: x86_64 architecture, with the Intel(R) Xeon(R) CPU E5-3695 vs2 @ 2.40GHz CPU, made by GenuineIntel that consisted of 48 CPUs (2 sockets, 12 cores per socket, and 2 threads per core).

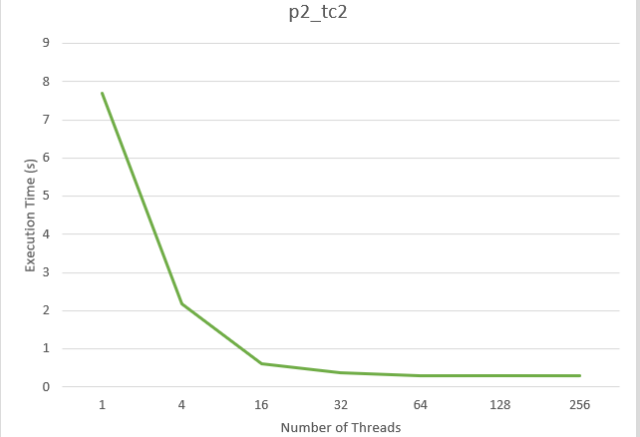
The main point of interest was to see how the runtime is affected as the number of threads is increased on the same file input. The execution of the program consisted of five test files: each file progressively bigger in size. The experiment tested it out at these increments of threads: 1, 4, 16, 32, 64, 128 and 256. Each increment we tested around 3-5 times and took the average of those runs to ensure an accurate execution time (I did not have independent use of the server where the test ran so other programs executing at the same time could cause ours to slow).

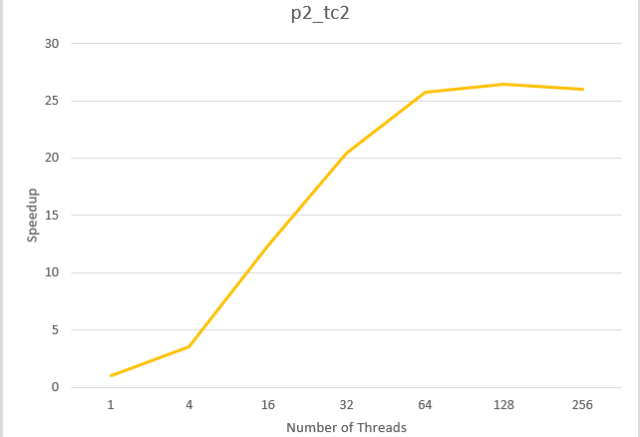
What was discovered was that the execution time does consistently go down as threads are increased, with the exception of the running time of threads between 128 and 256, in some test cases their execution times were extremely close and only marginally faster, but in some cases the time was marginally slower. Overall, the speedup shows diminishing returns as the number of threads increases. The biggest spike of improved execution time happens from 4 to 32 threads, with threads 1-16 seeing an approximate doubling of speedup (this is the only time in which the speedup is close to proportional to the number of threads increased), this slows down slightly from 32-128 threads and significantly less improved (in some cases worse) from 128-256 threads. Below is a table of each test case (labelled: p2_tc0-p2_tc4) with execution time in seconds and speedup (time taken for single thread/ time taken for n threads).

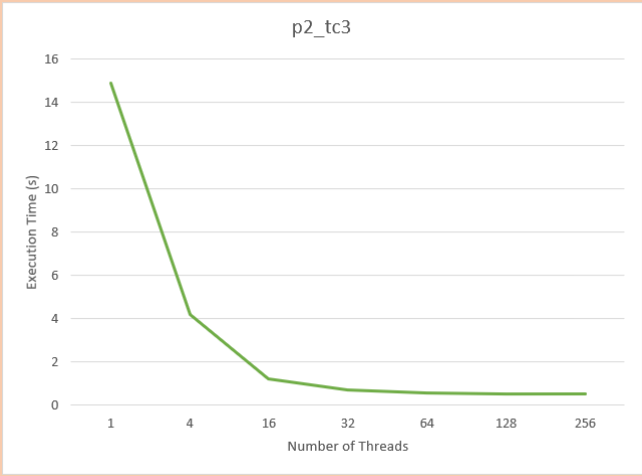
| | Number of Threads | 1 | 4 | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|---|---|---|
| p2_tc0 | Execution Time (s) | 1.96 | 0.54 | 0.189 | 0.119 | 0.0987 | 0.093 | 0.09929 |
| | Speedup | 1 | 3.62963 | 10.37037 | 16.47059 | 19.85816 | 21.07527 | 19.74016 |
| p2_tc1 | Execution Time (s) | 4.9 | 1.038 | 0.33 | 0.201 | 0.159 | 0.158 | 0.177 |
| | Speedup | 1 | 4.720617 | 14.84848 | 24.37811 | 30.81761 | 31.01266 | 27.68362 |
| p2_tc2 | Execution Time (s) | 7.697 | 2.165 | 0.622 | 0.376 | 0.299 | 0.2906 | 0.296 |
| | Speedup | 1 | 3.555196 | 12.3746 | 20.47074 | 25.74247 | 26.48658 | 26.00338 |
| p2_tc3 | Execution Time (s) | 14.9 | 4.218 | 1.204 | 0.719 | 0.585 | 0.546 | 0.543 |
| | Speedup | 1 | 3.53248 | 12.37542 | 20.72323 | 25.47009 | 27.28938 | 27.44015 |
| p2_tc4 | Execution Time (s) | 38.5 | 8.479 | 2.254 | 1.435 | 1.108 | 1.062 | 1.059 |
| | Speedup | 1 | 4.54063 | 17.08075 | 26.82927 | 34.74729 | 36.25235 | 36.35505 |

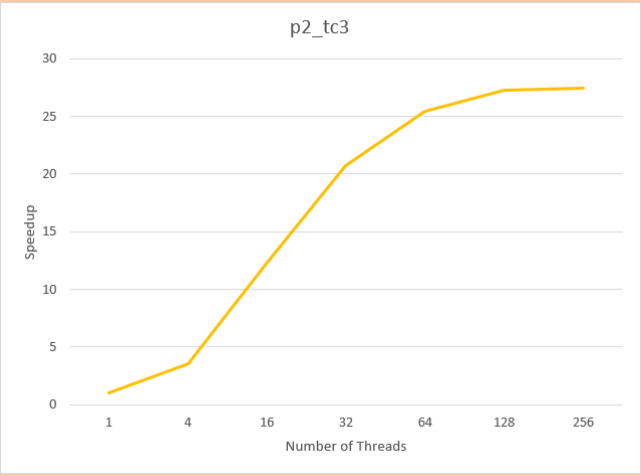
Each of these test cases are graphed below to show the execution time vs. number of threads, as well as the speedup vs number of threads.

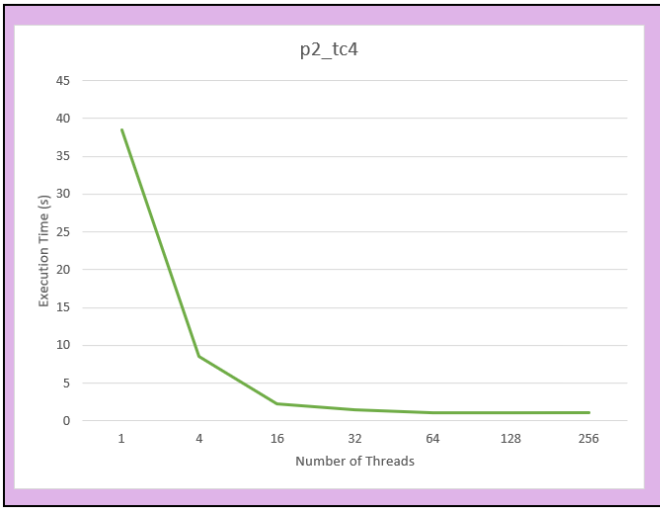


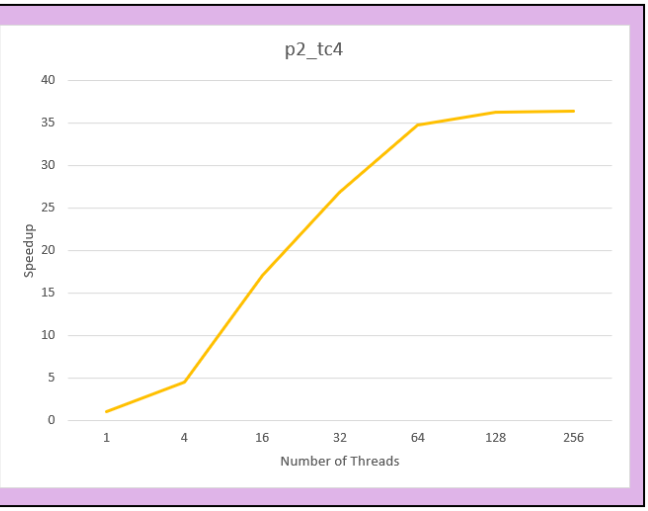

p2_tc2
Execution Time (s)
0
1
2
3
4
5
6
7
8
9
1
4
16
32
64
128
256
Number of Threads


p2_tc2
Speedup
0
5
10
15
20
25
30
1
4
16
32
64
128
256
Number of Threads


p2_tc3
Execution Time (s)
0
2
4
6
8
10
12
14
16
1
4
16
32
64
128
256
Number of Threads


p2_tc3
Speedup
0
5
10
15
20
25
30
1
4
16
32
64
128
256
Number of Threads


p2_tc4
Execution Time (s)
0
5
10
15
20
25
30
35
40
45
1
4
16
32
64
128
256
Number of Threads


p2_tc4
Speedup
0
5
10
15
20
25
30
35
40
1
4
16
32
64
128
256
Number of Threads

In conclusion, the speedup achieved by adding threads was a large improvement to the program's ability to hash files efficiently. Depending on the number of threads added (minimum 4, maximum 256) showed an improvement of a minimum of ~3.53x speedup and a maximum of ~36.35x speedup.