Multithreaded Hash Tree Report

Experimental set-up:

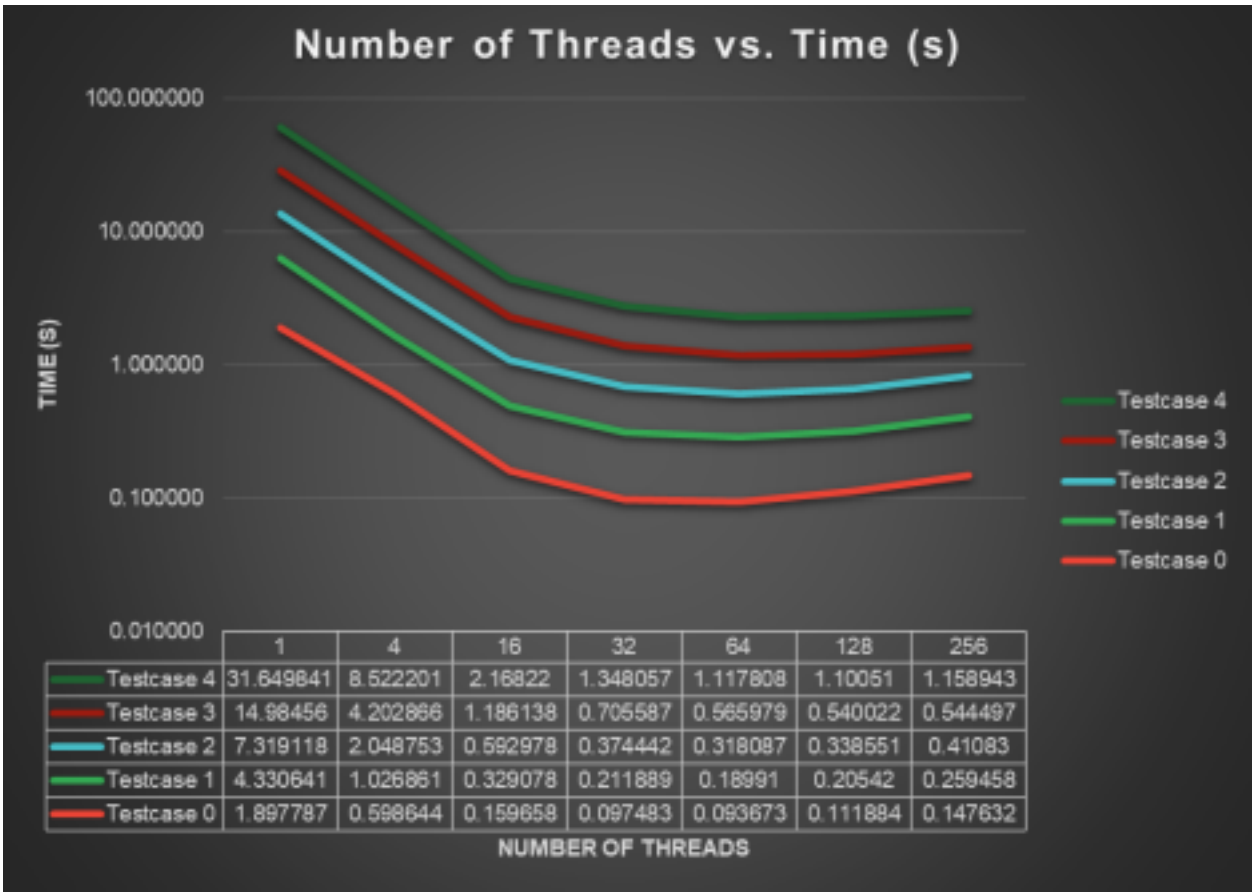
When testing code for all test cases from 1 to 256 threads, the system used was the cs3 machine of UTD. The computer was Dell PowerEdge R720 with 2 twelve-core 2.4GHz Intel Xeon, 192 GB RAM. The operating system used was Linux – CentOS 7.9 x86_64.
The hash function used for this experiment is Jenkins one_at_a_time hash. The program computed the hash value using the number of threads specified by the user and returned the time in seconds. The time started right before the program created the first thread (root thread), and ended when the root thread terminated. This data was recorded to construct graphs and provide conclusions from analyzing the results.

Specific System Specifications:

```
{cslinux3:~} lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                48
On-line CPU(s) list:   0-47
Thread(s) per core:    2
Core(s) per socket:    12
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 62
Model name:            Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz
Stepping:              4
CPU MHz:               2915.771
CPU max MHz:           3200.0000
CPU min MHz:           1200.0000
BogoMIPS:              4800.06
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              30720K
```

Analysis:

1. Number of Threads vs Time

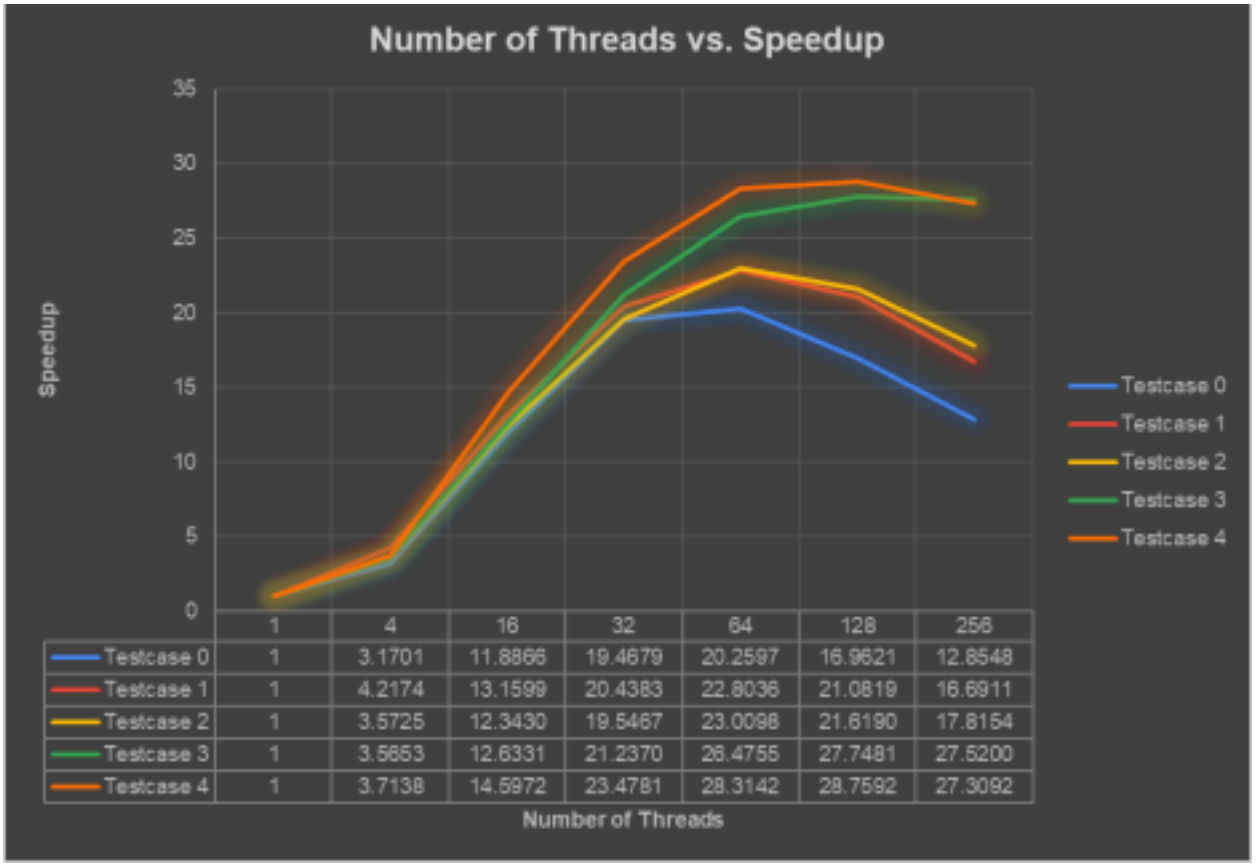


	1	4	16	32	64	128	256
Testcase 4	31.649841	8.522201	2.16822	1.348057	1.117808	1.10051	1.158943
Testcase 3	14.98456	4.202866	1.186138	0.705587	0.565979	0.540022	0.544497
Testcase 2	7.319118	2.048753	0.592978	0.374442	0.318087	0.338551	0.41083
Testcase 1	4.330641	1.026861	0.329078	0.211889	0.18991	0.20542	0.259458
Testcase 0	1.897787	0.598644	0.159658	0.097483	0.093673	0.111884	0.147632

The time taken to compute hash values does not always decrease when the number of threads is increased. Based on observation, starting from 32 threads, there were not any huge differences in the time taken to compute hash values. In fact, it took slightly more time to compute hash values
with 256 threads compared to 128 threads. In some cases, it could also be seen that using 128 threads was slower or almost equal to using 64 threads, particularly for the lower-sized files.

However, when the number of threads were increased from 1 to 4, or 4 to 16, it was apparent that there was a significant gain in terms of speed, with the time taken decreasing by almost seventy-five percent in each case. As the number of threads increased beyond 16, the change in time wasn't as drastic compared to using 4 and 16 threads.

2. Number of Threads vs. Speedup



	1	4	16	32	64	128	256
Testcase 0	1	3.1701	11.8866	19.4679	20.2597	16.9621	12.8548
Testcase 1	1	4.2174	13.1599	20.4383	22.8036	21.0819	16.6911
Testcase 2	1	3.5725	12.3430	19.5467	23.0098	21.6190	17.8154
Testcase 3	1	3.5653	12.6331	21.2370	26.4755	27.7481	27.5200
Testcase 4	1	3.7138	14.5972	23.4781	28.3142	28.7592	27.3092

The speedup achieved by increasing the number of threads is not always proportional to the number of threads increased. When 4 threads were used, the speedup was approximately equal to 4. However, the more threads that were utilized, the gains in speedup time were lower proportionally compared to the number of threads, which are approximately 13 for 16 threads, 20 for 32 threads, and 25 for 64 threads. From 128 threads, the speedup time started to decrease.

Conclusion:

In conclusion, although increasing the number of threads should make the time taken to compute hash value decrease, increasing the number of threads beyond the number of available cores can lead to diminishing returns or even a slowdown due to contention for resources such as cache and memory bandwidth. Therefore, it is important to optimize the number of threads used for a particular hashing task to achieve the best performance. It can also be observed that the speedup isn't proportional relative to the number of threads. The ideal number of threads observed in this experiment for computing the hash value is 64.