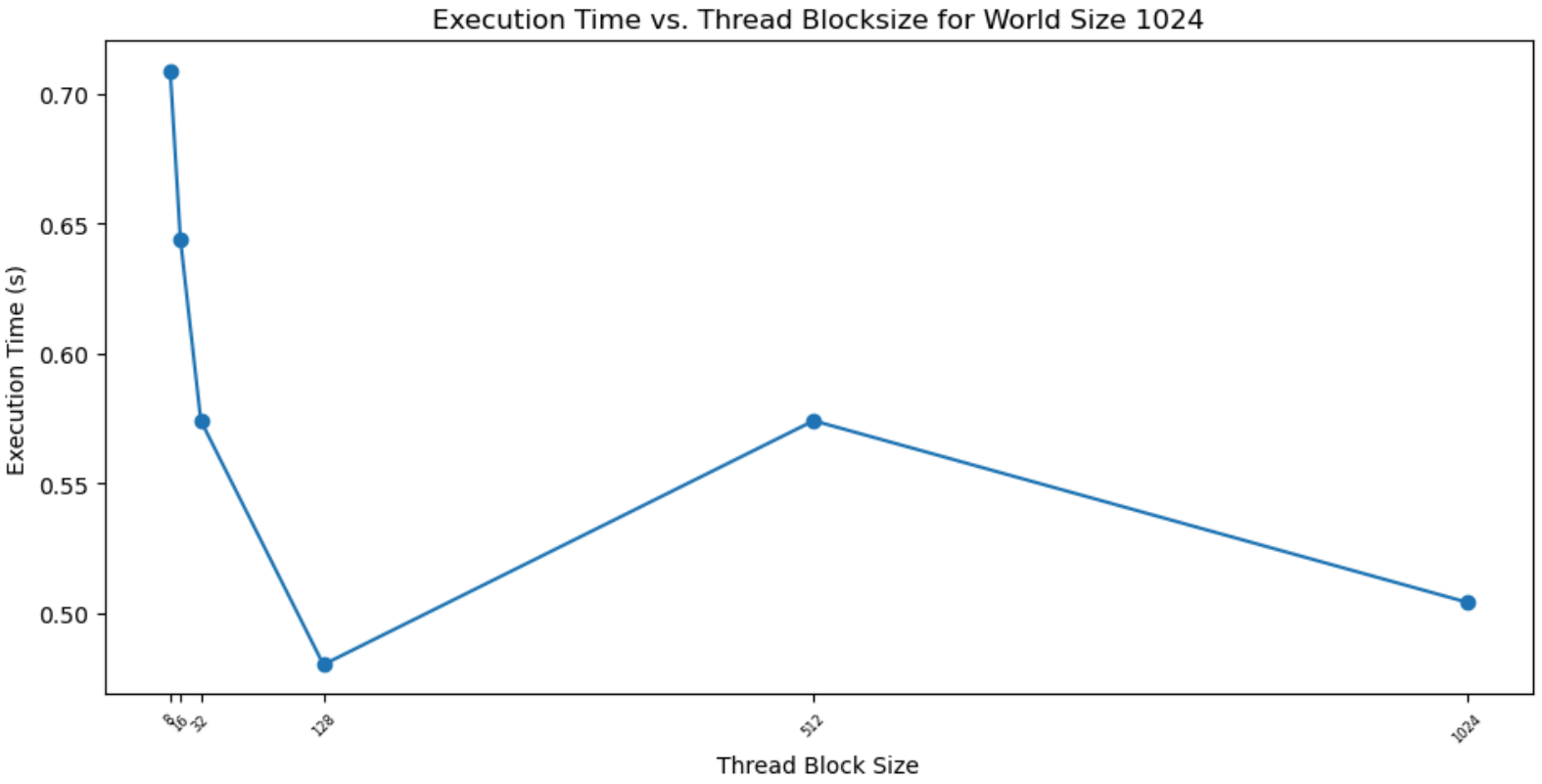
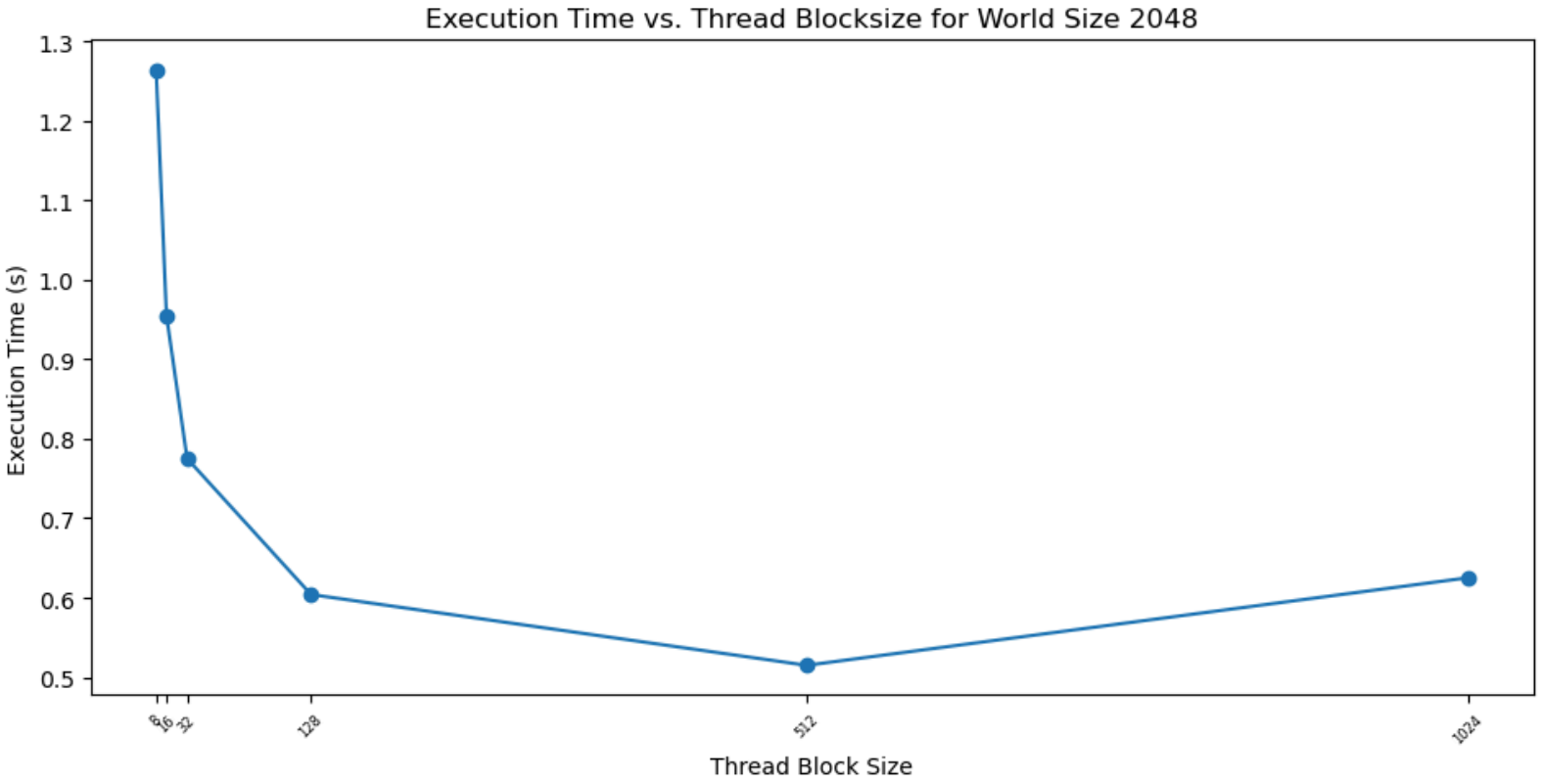
# Tables and Graphs for Execution Time

## Execution Time

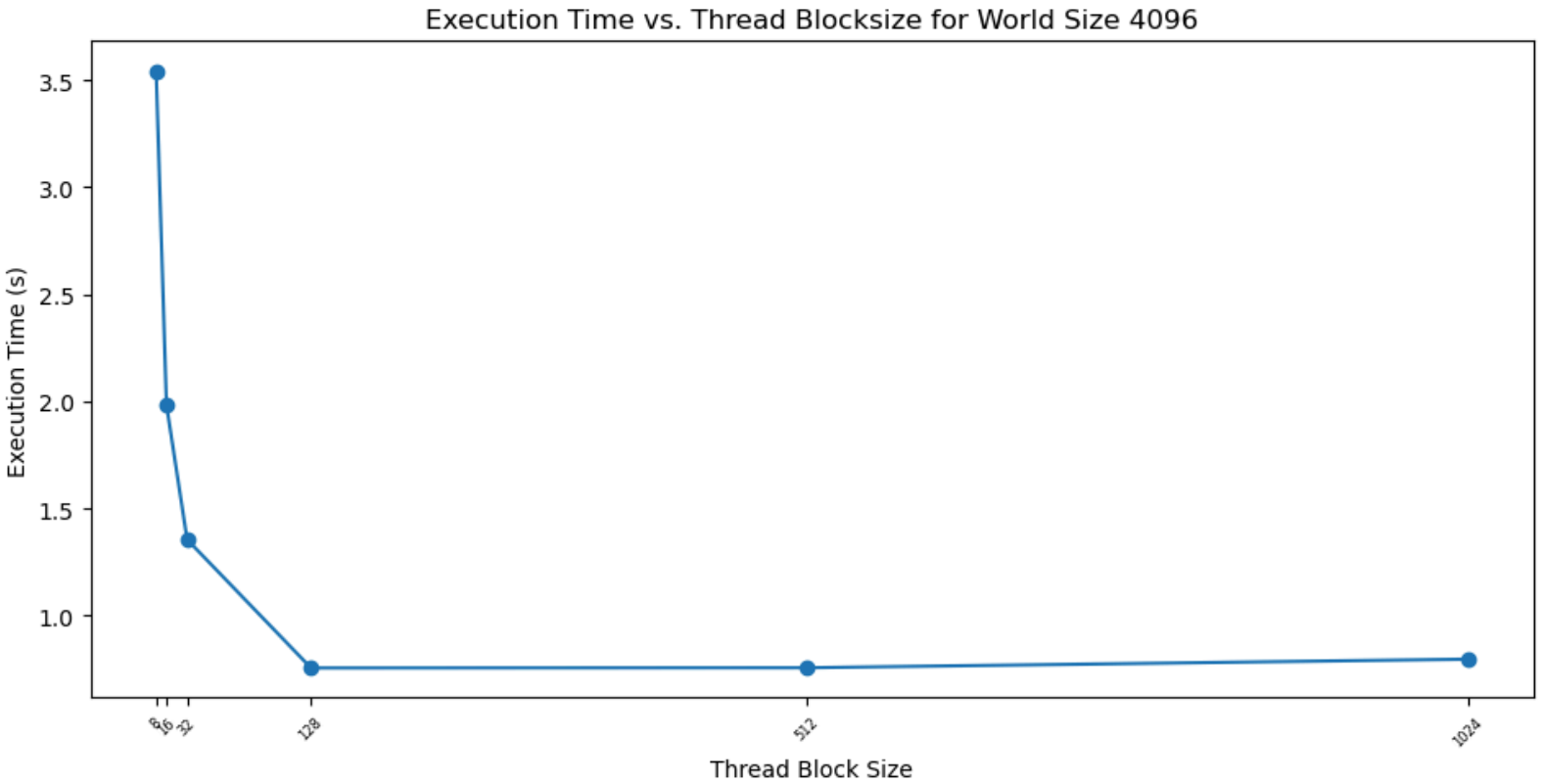
| Execution Time (s) | 8 | 16 | 32 | 128 | 512 | 1024 |
|---|---|---|---|---|---|---|
| 1024 | .709 | .644 | .574 | .48 | .574 | .504 |
| 2048 | 1.264 | .955 | .775 | .604 | .515 | .625 |
| 4096 | 3.544 | 1.984 | 1.354 | .754 | .755 | .795 |
| 8192 | 12.725 | 6.614 | 3.544 | 1.335 | 1.304 | 1.475 |
| 16384 | 49.124 | 24.905 | 12.704 | 3.725 | 3.786 | 4.135 |
| 32768 | 194.944 | 97.854 | 49.334 | 13.085 | 13.614 | 14.845 |
| 65536 | 776.734 | 388.605 | 194.502 | 49.075 | 12.645 | 6.885 |



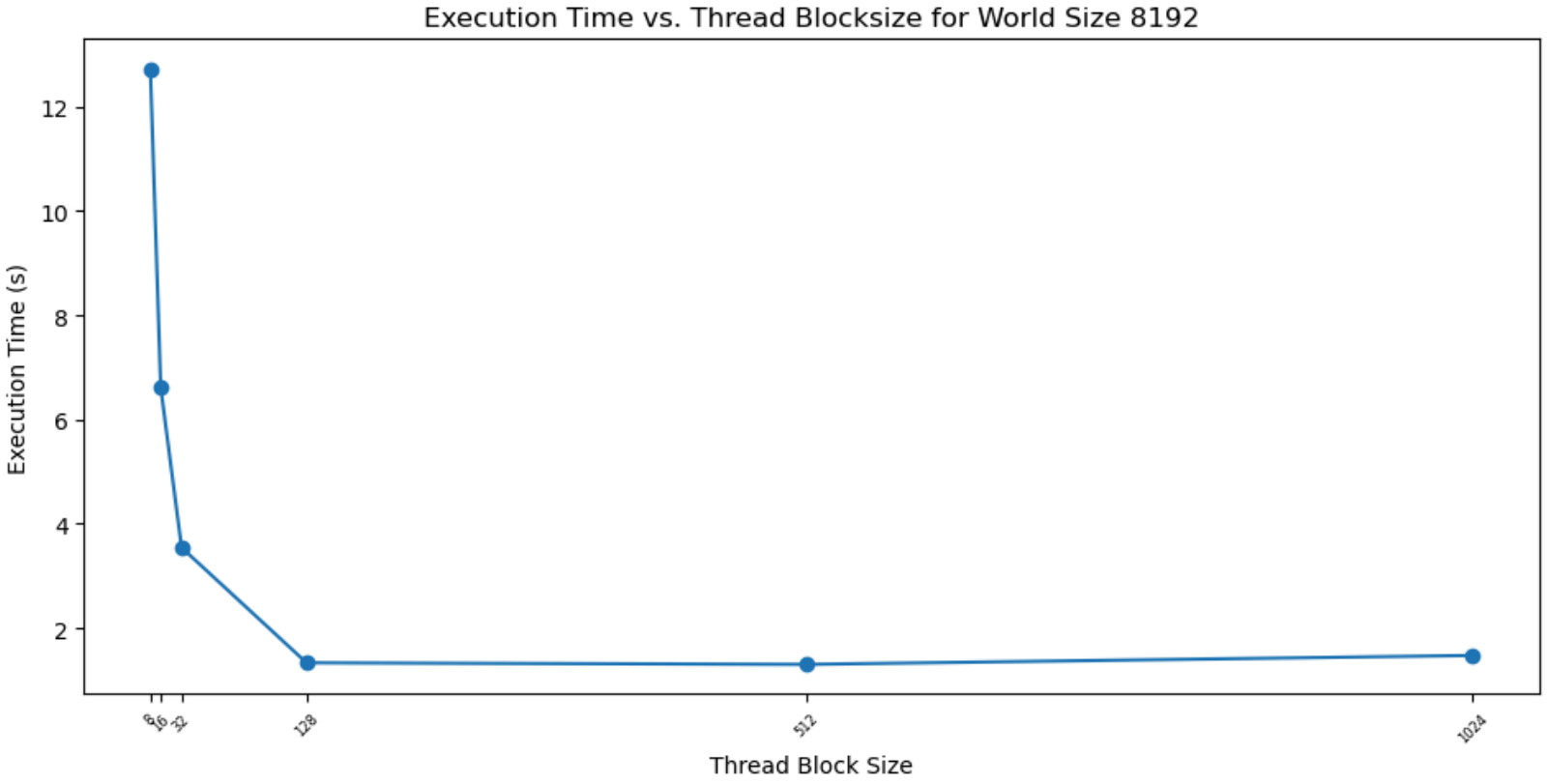

For world size 1024, the thread block size that yields the fastest execution time is 128.



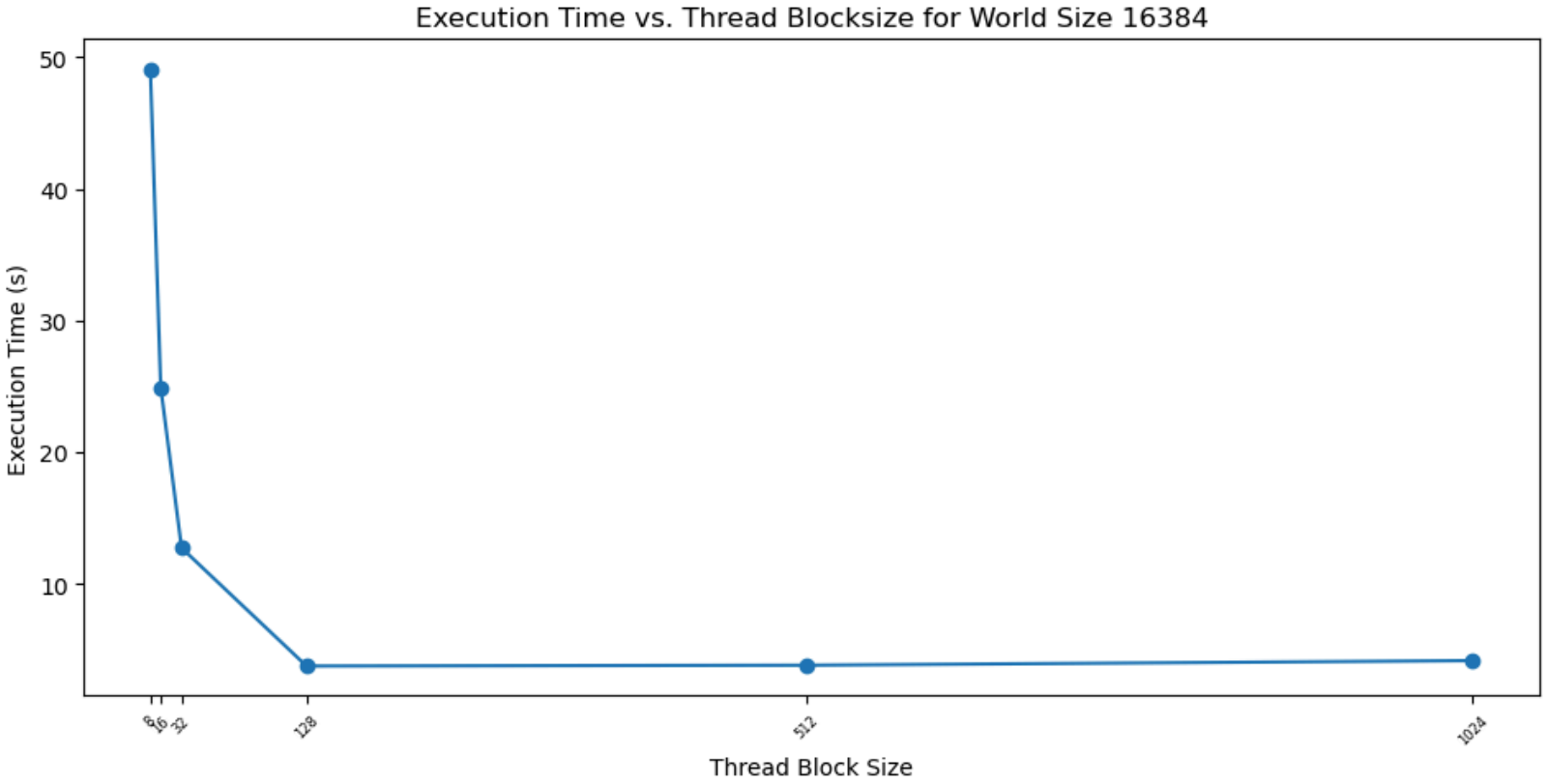

For world size 2048, a block size of 512 threads is optimal, with the fastest execution time.



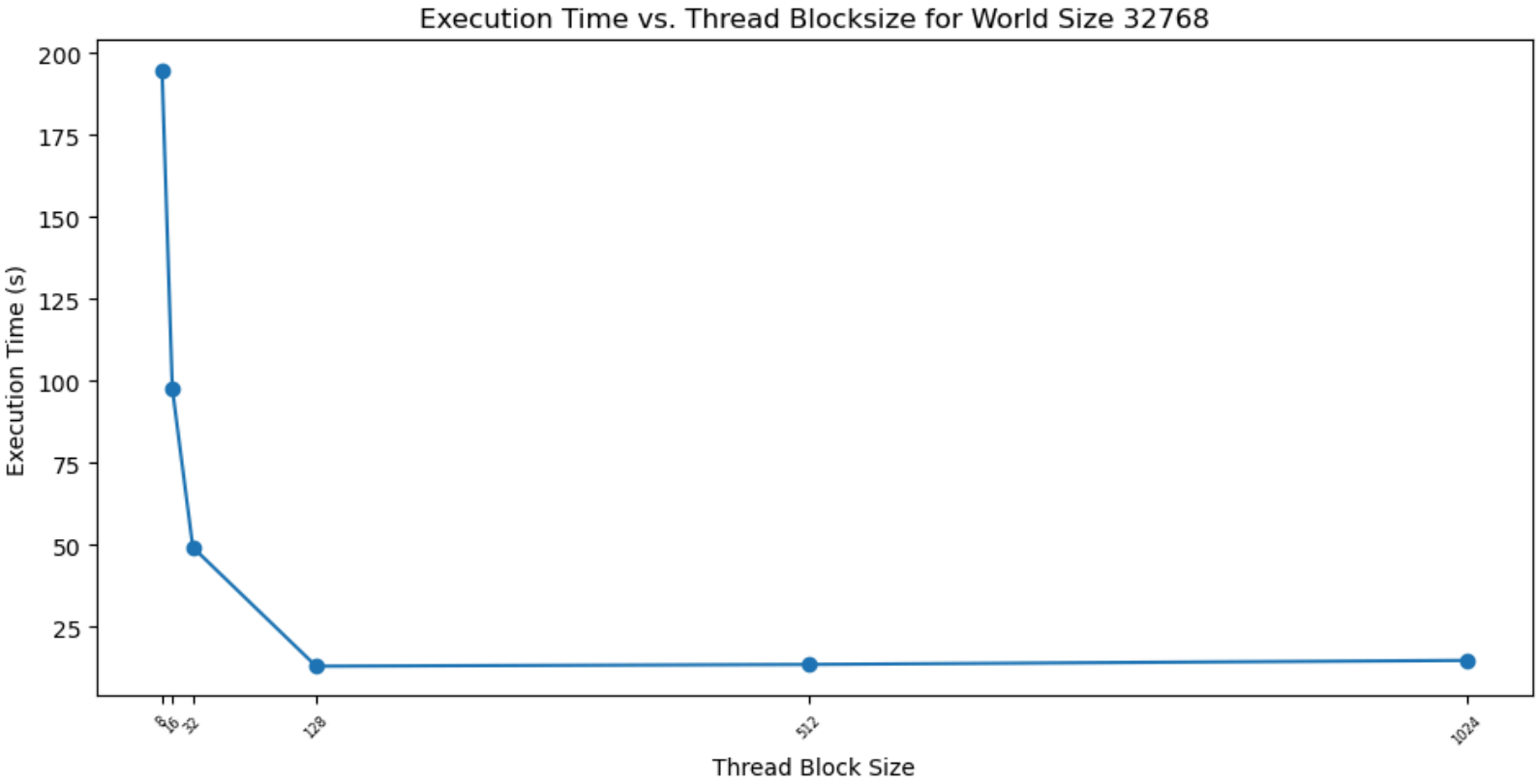

For world size 4096, a block size of 128 threads results in the fastest execution time.



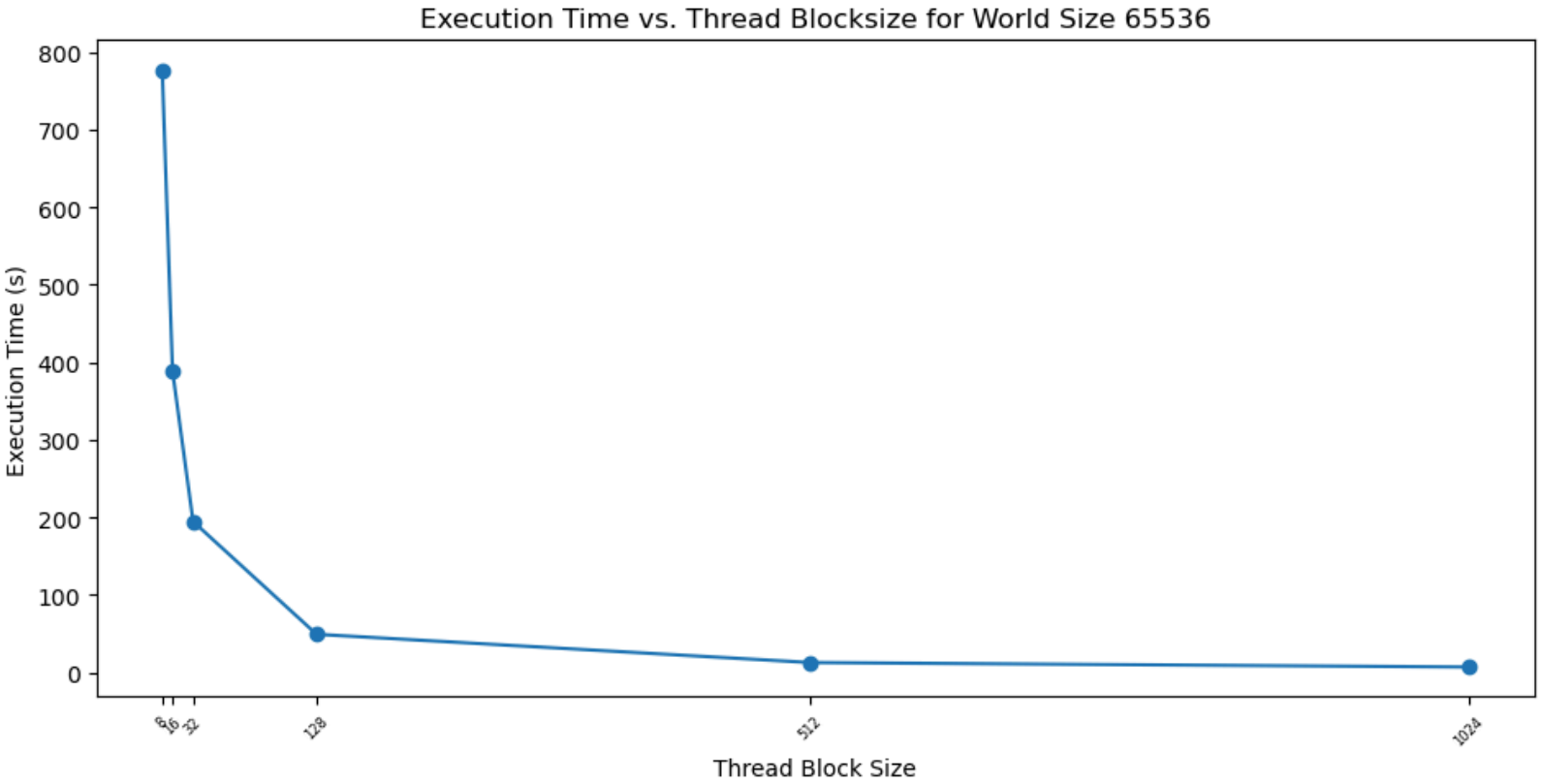

For a world size of 8192, a thread block size of 512 results in the fastest execution time.




For a world size of 16384, a block size of 128 yields the fastest execution time.




For a world size of 32768, block size 128 again yields the fastest execution time.




For a world size of 65536, the largest block size 1024, yields the fastest execution time.

After analyzing the execution time for the different world sizes, it seems that the optimal block size varies depending on the world size. This seems to suggest that the relationship between world size and block size is not linear and can be influenced by multiple factors.

For the smaller world sizes ( 1024, 4096, 16384, 32768 ), a block size of 128 threads seems most efficient, which could indicate that for the computation and memory access patterns associated with these world sizes, a configuration of 128 threads optimizes the GPU's execution efficiency. Memory access patterns could mean that different block sizes affect memory coalescing. Better coalescing means fewer memory transactions, which can lead to lower execution time.

For a medium world size of 2048, a larger block size of 512 threads is optimal. This could be due to a more complex computation that benefits from more threads working concurrently within a block to utilize memory more efficiently. Moreover, different sizes of data may benefit from different block sizes due to how they utilize the GPU's shared memory and registers.

For the largest world size of 65536, the block size of 1024 threads is optimal. Larger world sizes might benefit from larger block sizes due to increased parallelism and reduced scheduling overhead.

The variance when the block size is 512 threads for world sizes 2048 and 8192 indicates that this configuration might be hitting a balance between efficient use of the GPU's resources and avoiding launching too many blocks, which could cause scheduling overhead.

The number of active warps per multiprocessor can also affect this. A block size that maximizes occupancy without exhausting shared memory or register file can lead to better performance. Finally, smaller block sizes might lead to more predictable control flow, reducing the penalty from warp divergence.

When the block size was less than or equal to the CUDA hardware of 32 threads, the execution time was generally high across all world sizes. This is likely because smaller block sizes do not fully utilize warp. Since a warp is a group of 32 threads that execute in SIMD (single instruction, multiple data) fashion, having less than 32 threads in a block may lead to idle threads within a warp, resulting in lower computational efficiency and underutilization of the GPU's processing power.

These insights show that it can be important to tune the block size to match both the GPU architecture and the specific demands of the computation being performed.

**“Cell updates per second” (CUDA)**

| “Cell updates / second” (cuda) | 8 | 16 | 32 | 128 | 512 | 1024 |
|---|---|---|---|---|---|---|
| 1024 | 1514445449.92 | 1667300968.94 | 1870630355.40 | 2236962133.33 | 1870630355.40 | 2130440126.98 |
| 2048 | 3397917164.55 | 4497347953.92 | 5541893285.16 | 7110873006.62 | 8339742322.33 | 6871947673.60 |
| 4096 | 4847592884.87 | 8659208258.06 | 12688234257.01 | 22784972392.57 | 22754793621.19 | 21609898344.65 |
| 8192 | 5400351806.36 | 10390002530.39 | 19390371539.50 | 51475263472.65 | 52698985226.99 | 46589475753.22 |
| 16384 | 5595592926.96 | 11037057094.71 | 21637114841.30 | 73792726696.37 | 72603778907.55 | 66475914617.65 |
| 32768 | 5640140900.85 | 11236246119.48 | 22287096683.34 | 84028401052.80 | 72603778907.55 | 66475914617.65 |
| 65536 | 5662229941.14 | 11317524249.82 | 22611831812.03 | 89618879492.69 | 347809134923.21 | 638786711852.43 |

**“Cell updates per second” (C)**

| “Cell updates/ second (C) | 1024 | 2048 | 4096 | 8192 | 16384 | 32768 | 65536 |
|---|---|---|---|---|---|---|---|
| | 44739242666.6 | 2147483648000.00 | 8589934592000.00 | 34359738368000.00 | 137438953472000.00 | 549755813888000.00 | 2199023255552000.00 |

From the above, it seems the highest "cell updates per second" rate is for the world size 65536 with a block size of 1024, which is 63,878,671,185.43 updates/second.

Compare this with the C-code serial version. For world size 65536, the C-code version's rate is 2,555,520 updates/second.

To calculate how much faster the CUDA version is compared to the C-code version, we use the following formula:

Speedup Factor= C-code version’s rate/ CUDA version’s rate

=

63,878,671,185.43/ 2,555,520.

The CUDA version is approximately
24,996.35 times faster than the C-code serial version running on a single core of a POWER9 processor for the fastest "cells updates per second" rate configuration

To run my CUDA code, I added a 4th argument to take in block size and adjusted everything accordingly. I also created a bash script to run the performance analysis and calculate both execution time as well as the “cell updates per second”