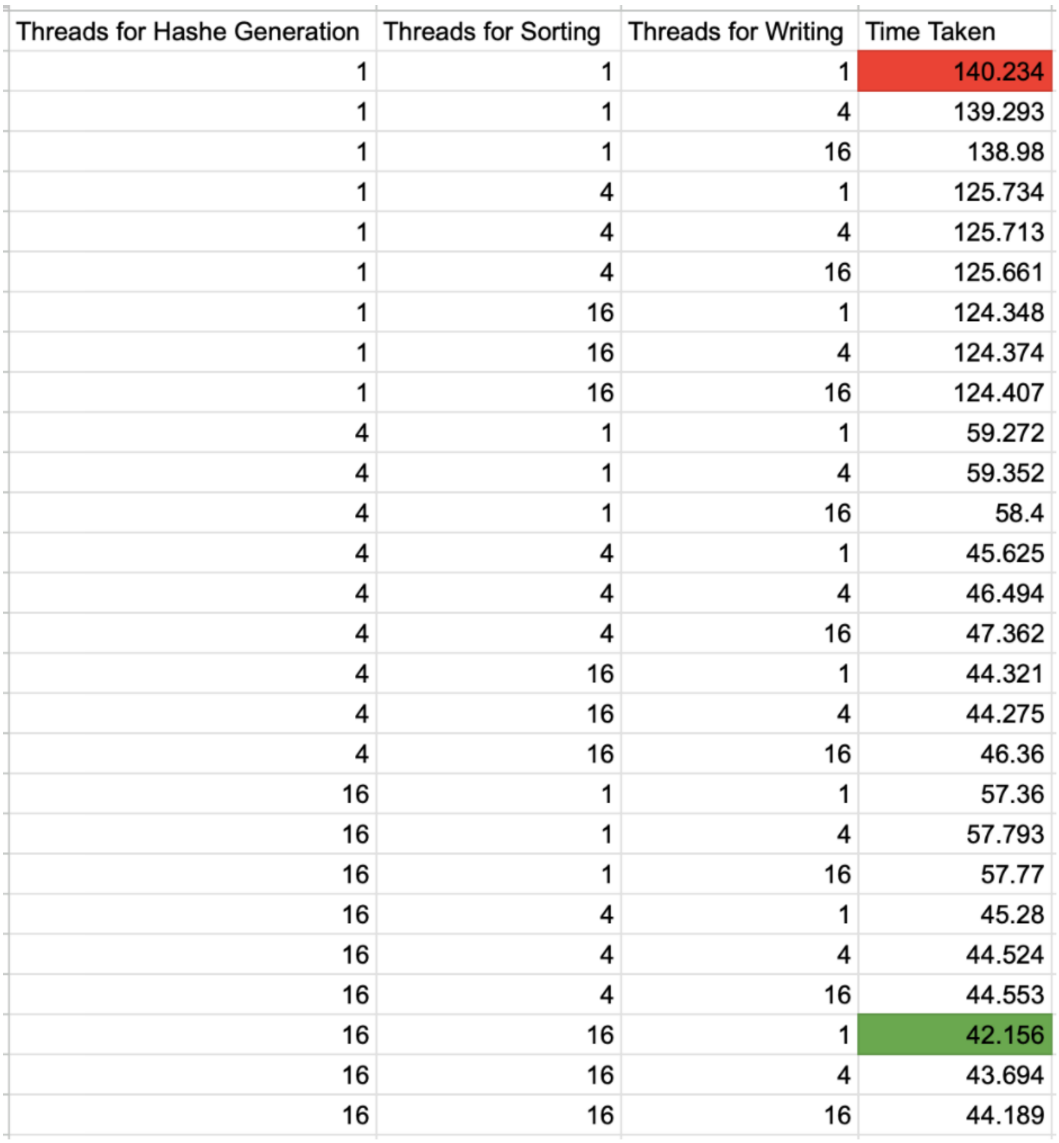
Benchmarking Memory & Storage

1GB File Benchmarking

Table

Below is the table which shows the result of all the 27 experiments conducted for generating 1GB file.



Threads for Hashe Generation	Threads for Sorting	Threads for Writing	Time Taken
1	1	1	140.234
1	1	4	139.293
1	1	16	138.98
1	4	1	125.734
1	4	4	125.713
1	4	16	125.661
1	16	1	124.348
1	16	4	124.374
1	16	16	124.407
4	1	1	59.272
4	1	4	59.352
4	1	16	58.4
4	4	1	45.625
4	4	4	46.494
4	4	16	47.362
4	16	1	44.321
4	16	4	44.275
4	16	16	46.36
16	1	1	57.36
16	1	4	57.793
16	1	16	57.77
16	4	1	45.28
16	4	4	44.524
16	4	16	44.553
16	16	1	42.156
16	16	4	43.694
16	16	16	44.189

Best Combination

From the table, we can see that the lowest time taken to generate the file is by using Hash generation threads=16, Sorting threads=16 and Write threads=1. This combination is showing the best result because hashes are getting generated concurrently by all the threads. So, higher number of threads would mean higher number of hashes being generated simultaneously.

After a set of hashes is generated, They are sorted and written to file. To sort the hashes, I have used multithreaded merge sort in which the data is divided into chunks and all the threads sort their chunks and later merge together. Due to higher number of threads, many chunks are sorted at the same time and results in lesser time needed to sort the data.

After chunks of hashes are generated, sorted and written to files, they are getting merged to a single file by multiple threads and then those files are merged to a single file. Since we are generating small file, while doing this, if there are multiple threads, they have to merge and generate files and then again merge those files. This results in higher number of I/O operations which take more time and the overhead of synchronizing the threads becomes prominent which results in slower writing speeds. If only 1 thread is writing to the file, it would be faster as only one file is being generated.

Worst Combination

You can see the worst combination is 1,1,4. It should be 1,1,16 but sometimes when hashes are generated which are a bit sorted, time varies. But, from the graphs given below, you can see that avg time taken by threads=16 for writing files take the most time. Thus, we can conclude 1,1,16 is the worst combination of threads.

Due to less threads, less number of hashes are generated at a time which results in higher duration for generation.

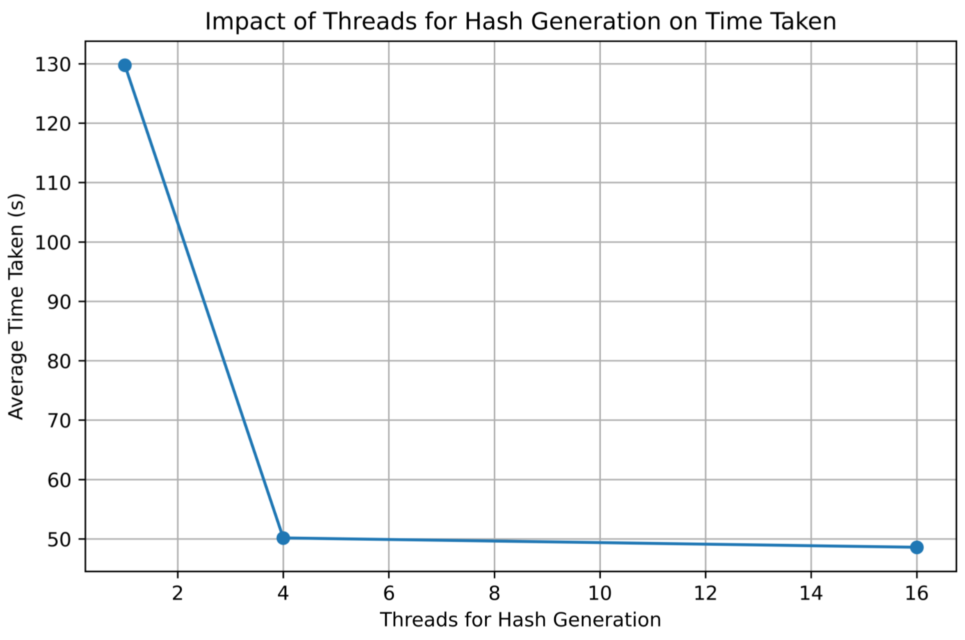
Less number of threads for sorting results in less number of chunks being sorted at the same time which takes more time to sort.

Higher number of threads for writing results in more time which is explaines in the previous section.

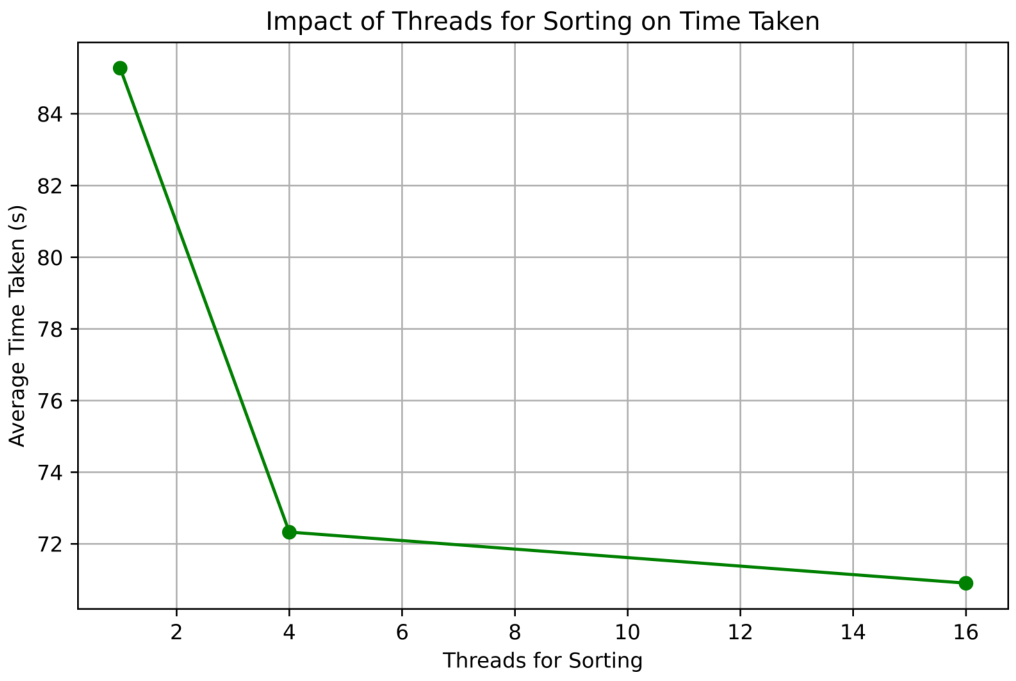
Graphs

Below are 3 graphs plotted based on Threads for Hash Generation, Threads for Sorting and Threads for Writing. Average time taken for file generation for a particular thread count is considered for plotting the graph. For Hash generation graph, average time is calculated based on the thread count. For example, for thread=1 for hash generation, all the time taken values irrespective of sorting and writing thread count are taken into consideration to calculate average time taken. Similarly, it has been done for Sorting graph and Writing graph.

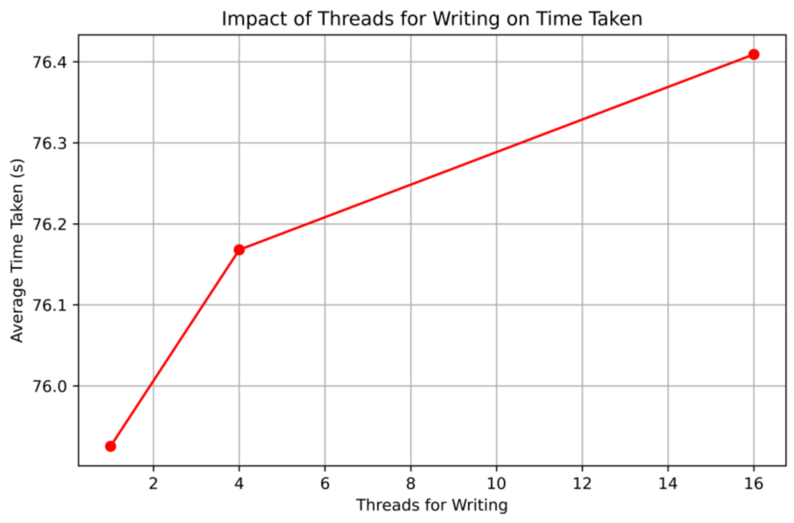
1. Impact of Threads for Hash Generation on Time Taken



2. Impact of Threads for Sorting on Time Taken



3.Impact of Threads for Writing on Time Taken



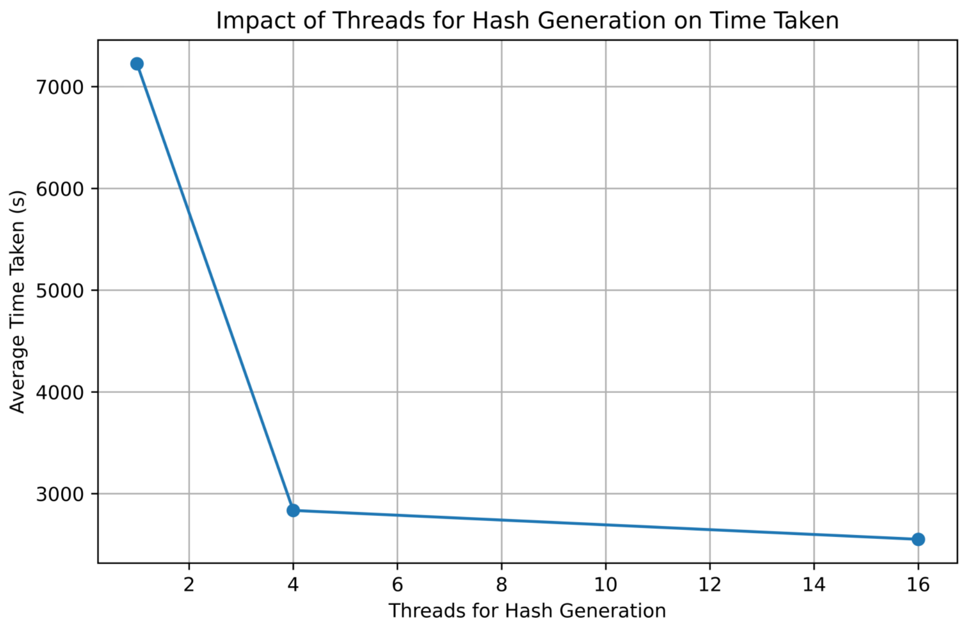
64GB FILE Benchmarking

Table

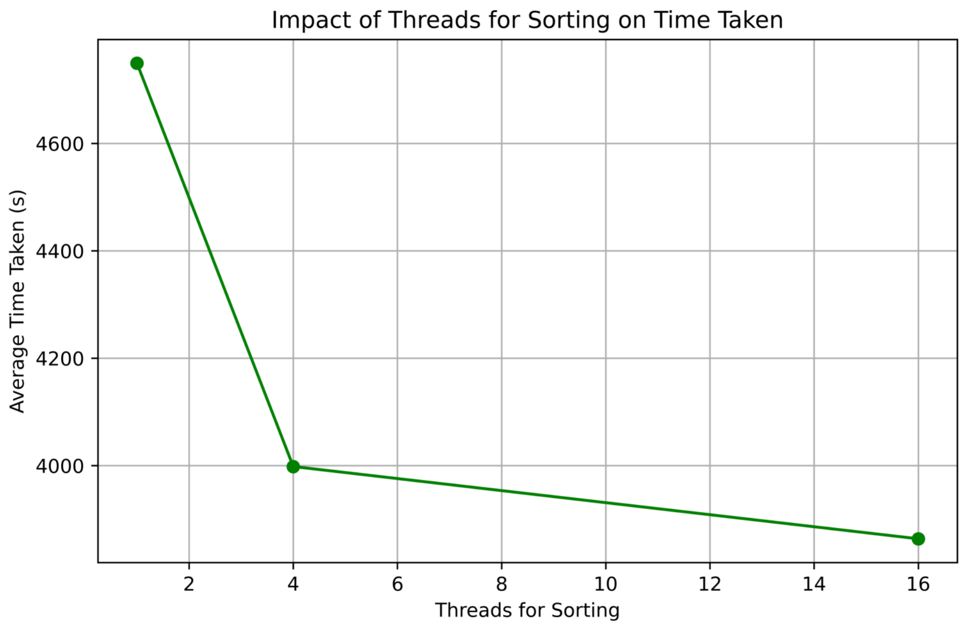
Threads for Hash Generation	Threads for Sorting	Threads for Writing	Time Taken
1	1	1	9022.402
1	1	4	7647.529
1	1	16	7365.94
1	4	1	7034.902
1	4	4	6927.789
1	4	16	6766.033
1	16	1	6908.444
1	16	4	6750.822
1	16	16	6593.571
4	1	1	3512.416
4	1	4	3357.656
4	1	16	3095.2
4	4	1	2789.125
4	4	4	2623.182
4	4	16	2510.186
4	16	1	2614.013
4	16	4	2558.575
4	16	16	2457.08
16	1	1	3040.08
16	1	4	2904.029
16	1	16	2796.81
16	4	1	2505.84
16	4	4	2465.772
16	4	16	2361.309
16	16	1	2342.017
16	16	4	2315.782
16	16	16	2232.96

Graphs

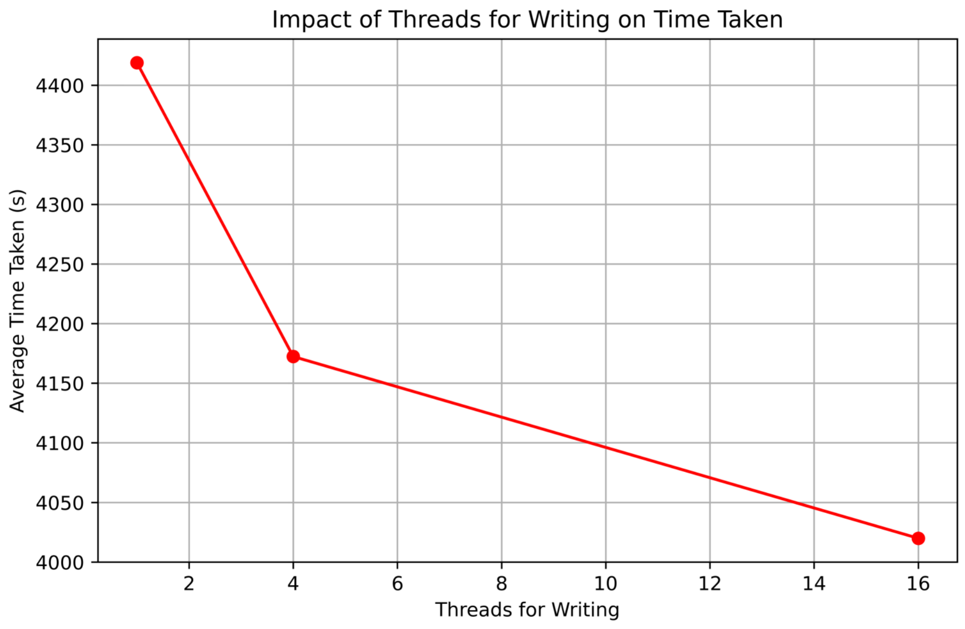
1.Impact of Threads for Hash Generation on Time Taken



2.Impact of Threads for Sorting on Time Taken



3.Impact of Threads for Writing on Time Taken



Best Combination

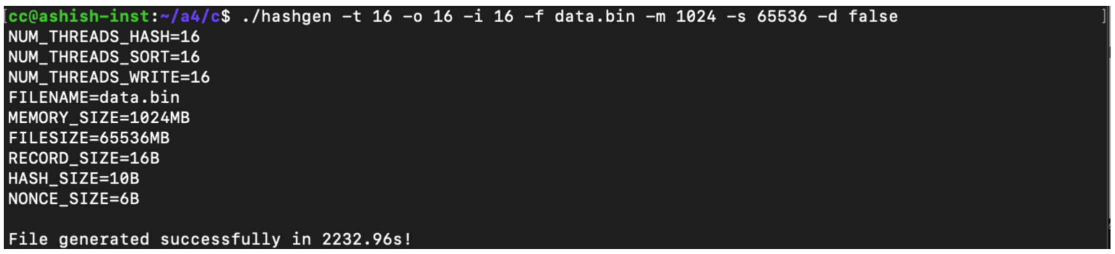
For generating 64gb file, the best combination of threads is threads=16 for hash generation, threads=16 for sorting, threads=16 for writing. The reason for using threads=16 for both hash generation and sorting remains the same as it is for 1GB File. But, for writing, threads=16 performs better here because multiple chunks are getting sorted at the same time and the overhead for synchronization is minimal compared to the File size being generated.

Worst Combination

For generating 64gb file, the worst combination of threads is threads=1 for hash generation, threads=1 for sorting, threads=1 for writing. It is because, file to be generated is huge and worker threads are very less. This causes very slow generation of hashes, sorting and writing to file.

Benchmarking 64GB File with different Memory Sizes

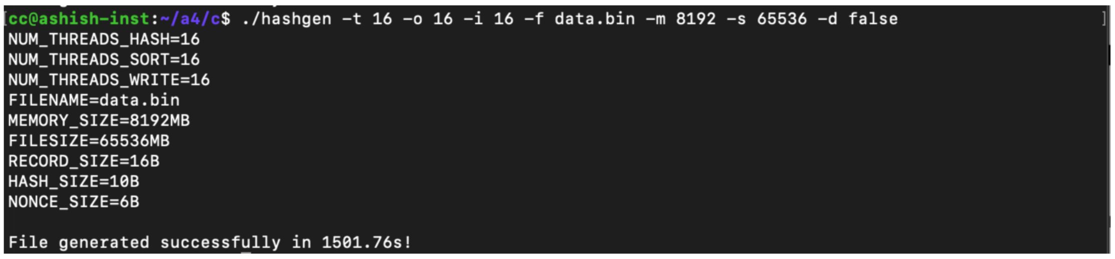
Memory = 1024 Time Taken : 2232.96s



```
cc@ashish-inst:~/a4/c$ ./hashgen -t 16 -o 16 -i 16 -f data.bin -m 1024 -s 65536 -d false
NUM_THREADS_HASH=16
NUM_THREADS_SORT=16
NUM_THREADS_WRITE=16
FILENAME=data.bin
MEMORY_SIZE=1024MB
FILESIZE=65536MB
RECORD_SIZE=16B
HASH_SIZE=10B
NONCE_SIZE=6B

File generated successfully in 2232.96s!
```

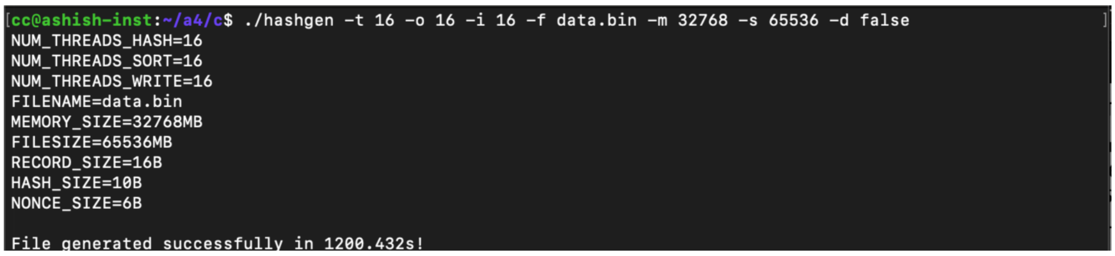
Memory = 8192 Time Taken : 1501.76s



```
cc@ashish-inst:~/a4/c$ ./hashgen -t 16 -o 16 -i 16 -f data.bin -m 8192 -s 65536 -d false
NUM_THREADS_HASH=16
NUM_THREADS_SORT=16
NUM_THREADS_WRITE=16
FILENAME=data.bin
MEMORY_SIZE=8192MB
FILESIZE=65536MB
RECORD_SIZE=16B
HASH_SIZE=10B
NONCE_SIZE=6B

File generated successfully in 1501.76s!
```

Memory = 32768 Time Taken : 1200.432s



```
cc@ashish-inst:~/a4/c$ ./hashgen -t 16 -o 16 -i 16 -f data.bin -m 32768 -s 65536 -d false
NUM_THREADS_HASH=16
NUM_THREADS_SORT=16
NUM_THREADS_WRITE=16
FILENAME=data.bin
MEMORY_SIZE=32768MB
FILESIZE=65536MB
RECORD_SIZE=16B
HASH_SIZE=10B
NONCE_SIZE=6B

File generated successfully in 1200.432s!
```

Using Higher Memory helps in sorting and merging as more data can be stored in memory rather than disk which is slower compare to RAM.

Conclusion

To summarize, the code is scalable. As you increase the file size to be generated it is able to handle that efficiently. It provides good concurrency as well because as you increse the worker threads, file is being generated faster. When you increse the Memory size, the time taken to generate file is decreasing which shows the improvement in performance.