# CS553 Homework #4 (revision 2)

## Benchmarking Memory & Storage

***Instructions:***

- *Assigned date: Wednesday October 8th, 2025*
- *Due date: 11:59PM on Monday October 27th, 2025*
- *Maximum Points: 100 (+20 extra credit)*
- *This assignment can be done in groups of 2 students*
- *Please post questions on Canvas*
- *Only a softcopy submission is required through GIT: push changes to GIT repository*
- *Late submission will be penalized at 20%; late submissions beyond 24 hours will receive 0 points*

### 1 Your Assignment

This project aims to teach you about file I/O and benchmarking. You must C or C++ for this assignment. You can use the following libraries: STL, Boost, PThread, OMP, BLAKE3, SYCL, and CUDA, MMAP, CUFile. You must use Linux system for your development and evaluation. The performance evaluation should be done on Chameleon on Ubuntu Linux 24.03.

You will make use of a hashing algorithm:

- Blake3:
  - Repo: https://github.com/BLAKE3-team/BLAKE3
  - Paper: https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf

Your benchmark will use a 6-byte NONCE to generate $2^{26}$ (SMALL) or $2^{32}$ (LARGE) BLAKE3 hashes of 10-bytes long each and store them in a file on disk in sorted order (sorted by 10-byte hash). A record could be defined in C as follows:

```
#define NONCE_SIZE 6
#define HASH_SIZE 10

// Structure to hold a 16-byte record
typedef struct {
     uint8_t hash[HASH_SIZE]; // hash value as byte array
     uint8_t nonce[NONCE_SIZE]; // Nonce value as byte array
} Record;
```

Your file should be 1GB (SMALL) or 64GB (LARGE) in size when your benchmark completes (64GB = $2^{32}$*(10B+6B)) – your file should be written in binary to ensure efficient write and read to this data; do not write the data in ASCII. You are to parallelize your various stages (hash generation, sort, and disk write) with a pool of threads per stage, that can be controlled separately.

You will have several command line arguments that you will explore from a performance point of view for the SMALL 1GB workload. There are 42=3x7x3 experiments to run, so make sure to automate the execution of these runs in a bash script (e.g. 3 nested loops in bash should work).

1. Maximum memory allowed to use (MB): 256, 512, 1024
2. Number of compute threads: 1, 2, 4, 8, 12, 24, 48
3. Number of write threads: 1, 2, 4

Each experiment should take less than a minute, depending on the hardware, processor type, core counts, and hard drive technology. These experiments should take less than 1 hour to run.

Now evaluate the large workload with 64GB data. There are 7=7x1x1 experiments to run, so make sure to automate the execution of these runs in a bash script.

1. Maximum memory allowed to use (MB): 1024, 2048, 4096, 8192, 16384, 32768, 65536
2. Number of compute threads: 24
3. Number of write threads: 1

Each experiment should take a few minutes to tens of minutes, depending on the hardware, processor type, core counts, and hard drive technology. These experiments should take less than 3 hours to run.

Fill in the table below for the small workload (K=26):

| Threads | Memory Size | IO Threads (1) | IO Threads (2) | IO Threads (4) |
|---|---|---|---|---|
| 1 | 256 | | | |
| 1 | 512 | | | |
| 1 | 1024 | | | |
| 2 | 256 | | | |
| 2 | 512 | | | |
| 2 | 1024 | | | |
| 4 | 256 | | | |
| 4 | 512 | | | |
| 4 | 1024 | | | |
| 8 | 256 | | | |
| 8 | 512 | | | |
| 8 | 1024 | | | |
| 12 | 256 | | | |
| 12 | 512 | | | |
| 12 | 1024 | | | |
| 24 | 256 | | | |
| 24 | 512 | | | |
| 24 | 1024 | | | |
| 48 | 256 | | | |

| | | | | |
|---|---|---|---|---|
| 48 | 512 | | | |
| 48 | 1024 | | | |

Fill in the table below for the large workload (K=32):

| Threads | Memory Size | IO Threads (1) |
|---|---|---|
| 24 | 1024 | |
| 24 | 2048 | |
| 24 | 4096 | |
| 24 | 8192 | |
| 24 | 16384 | |
| 24 | 32768 | |
| 24 | 65536 | |

Plot the results of these 42+7 experiments and identify the best combination of command line arguments. You should create 4 separate plots:

1. K=26, Fixed write threads = 1, vary Memory Size and Compute threads
2. K=26, Fixed write threads = 2, vary Memory Size and Compute threads
3. K=26, Fixed write threads = 4, vary Memory Size and Compute threads
4. K=32, Fixed write threads = 1, Fixed compute threads = 24, vary Memory Size

Explain why you believe the best and worst configurations make sense.

Here are the command line arguments your program should have:

```
cc@hw4-raicu-skylake:~/vault$ ./vaultx -h
Usage: ./vaultx [OPTIONS]

Options:
  -a, --approach [task|for]    Parallelization mode (default: for)
  -t, --threads NUM            Hashing threads (default: all cores)
  -i, --iothreads NUM          Number of I/O threads to use (default: 1)
  -c, --compression NUM        Compression: number of hash bytes to discard from the
end (0..HASH_SIZE)
  -k, --exponent NUM           Exponent k for 2^K iterations (default: 26)
  -m, --memory NUM             Memory size in MB (default: 1)
  -f, --file NAME              Final output file (moved/renamed to at end)
  -g, --file_final NAME        Temporary file (intermediate output)
  -d, --debug [true|false]     Enable per-search debug prints (default: false)
  -b, --batch-size NUM         Batch size (default: 1024)
  -p, --print NUM              Print NUM records and exit
  -s, --search                 Enable search of specified number of records
  -q, --difficulty             Set difficulty for search in bytes
  -h, --help                   Display this help message
Example:
  ./vaultx -t 24 -i 1 -m 1024 -k 26 -g memo.t -f memo.x -d true
```

```
cc@hw4-raicu-skylake:~/vault$ ./vaultx -t 24 -i 1 -m 256 -k 26 -g memo.t -f
k26-memo.x -d true
Selected Approach           : for
Number of Threads           : 24
Exponent K                  : 26
File Size (GB)              : 1.00
File Size (bytes)           : 1073741824
Memory Size (MB)            : 256
Memory Size (bytes)         : 268435456
Number of Hashes (RAM)      : 16777216
Number of Hashes (Disk)     : 67108864
Size of HASH                : 10
Size of NONCE               : 6
Size of MemoRecord          : 16
Rounds                      : 4
Number of Buckets           : 16777216
Number of Records in Bucket : 1
BATCH_SIZE                  : 1024
Temporary File              : memo.t
Final Output File           : k26-memo.x
[1.34] HashGen 25.00%: 15.35 MH/s : I/O 87.84 MB/s
[2.36] HashGen 50.00%: 16.48 MH/s : I/O 94.31 MB/s
[3.43] HashGen 75.00%: 15.72 MH/s : I/O 89.97 MB/s
[4.44] HashGen 100.00%: 16.54 MH/s : I/O 94.63 MB/s
[5.53] Shuffle 0.00%: 105.95 MB/s
[6.15] Shuffle 25.00%: 155.86 MB/s
[6.76] Shuffle 50.00%: 155.97 MB/s
[7.38] Shuffle 75.00%: 155.98 MB/s
File 'memo.t' removed successfully.
Total Throughput: 6.80 MH/s  103.79 MB/s
Total Time: 9.866384 seconds
```

cc@hw4-raicu-skylake:~/vault$ ./vaultx -t 24 -i 1 -m 256 -k 26 -g memo.t -f k26-memo.x -d false
vaultx t24 i1 m256 k26 6.82 104.02 9.844270

You should be able to verify that the data you are writing is correct.

```
cc@hw4-raicu-skylake:~/vault$ ./vaultx -t 24 -i 1 -m 256 -k 26 -g memo.t -f k26-memo.x
-d false -v true
vaultx t24 i1 m256 k26 6.71 102.45 9.995002
verifying sorted order by bucketIndex of final stored file...
Size of 'k26-memo.x' is 1073741824 bytes.
[1.22] Verify 25.00%: 236.97 MB/s
[2.36] Verify 50.00%: 237.32 MB/s
[3.56] Verify 75.00%: 226.82 MB/s
[4.72] Verify 100.00%: 237.26 MB/s
sorted=42413664 not_sorted=0 zero_nonces=24695200 total_records=67108864
```

You should be able to print the contents:

```
cc@hw4-raicu-skylake:~/vault$ ./vaultx -k 26 -f k26-memo.x -p 20
[0] stored: BLANK nonce: BLANK
[16] stored: 0000009fcad95785e04e nonce: 180358880493568
[32] stored: BLANK nonce: BLANK
[48] stored: BLANK nonce: BLANK
[64] stored: 000001bb43255f863484 nonce: 8216607981568
[80] stored: BLANK nonce: BLANK
[96] stored: BLANK nonce: BLANK
[112] stored: 000001012778f96f23bd nonce: 85673457680384
[128] stored: BLANK nonce: BLANK
[144] stored: 000002bf1e0337e0221b nonce: 38873715507200
[160] stored: 000002475cea698b6761 nonce: 113079190945792
[176] stored: 0000023640c2b8cc714d nonce: 258720072400896
[192] stored: 0000037941eb83a46ad1 nonce: 155531872305152
[208] stored: 0000036e8631726b50e8 nonce: 12661244887040
[224] stored: BLANK nonce: BLANK
[240] stored: BLANK nonce: BLANK
[256] stored: 00000431689ee0812def nonce: 82166532800512
[272] stored: 000004b754e76ae7186d nonce: 260124744679424
[288] stored: 00000449cce82e5bd961 nonce: 231432249278464
[304] stored: BLANK nonce: BLANK
```

Once you have everything done, we want to search through the file. You are going to generate random search queries based on the difficulty, and execute the search, and report statistics at the end of the search.

```
cc@hw4-raicu-skylake:~/vault$ ./vaultx -k 26 -f k26-memo.x -s 10 -q 3 -d true
searches=10 difficulty=3
Parsed k                     : 26
Nonce Size                   : 6
Record Size                  : 16
Hash Size                    : 10
On-disk Record Size          : 16
Number of Buckets            : 16777216
Number of Records in Bucket  : 4
Number of Hashes      : 67108864
File Size to be read (bytes) : 1073741824
File Size to be read (GB)    : 1.000000
Actual file size on disk     : 1073741824 bytes
[0] 1213ae MATCH 1213aef913a754c0aee2 185081113280512 time=0.053 ms comps=3
MATCH 1213aea8365ef5c396d8 153924027875328 time=0.053 ms comps=3
MATCH 1213aeb822e61a7e3ff7 161018189774848 time=0.053 ms comps=3
[1] c0f629 MATCH c0f629767847ab1875da 144034999828480 time=0.042 ms comps=2
MATCH c0f629d980132fa9c1a0 214554336493568 time=0.042 ms comps=2
[2] 08259a MATCH 08259a407625ea40f530 77072651452416 time=0.041 ms comps=4
MATCH 08259a3f70d4a742e072 58336628244480 time=0.041 ms comps=4
MATCH 08259a99379ef1a9efb3 48278586654720 time=0.041 ms comps=4
MATCH 08259a94723745fba6ec 172800476184576 time=0.041 ms comps=4
[3] deb0c7 MATCH deb0c7cfefbe66aa729b 52254786715648 time=0.040 ms comps=4
MATCH deb0c765426831f044be 10813519757312 time=0.040 ms comps=4
```

```
MATCH deb0c754426e38fb0e46 74989642514432 time=0.040 ms comps=4
MATCH deb0c7694ed820fe4941 29447235436544 time=0.040 ms comps=4
[4] 6ede11 MATCH 6ede11e8594021311f93 252420261216256 time=0.041 ms comps=4
MATCH 6ede11973a3e379fa70a 245398073638912 time=0.041 ms comps=4
MATCH 6ede11665a9f2e53cdbb 271868913254400 time=0.041 ms comps=4
MATCH 6ede11a889620aa92f47 98897779359744 time=0.041 ms comps=4
[5] 51f748 MATCH 51f748283b73e213e1b9 56298481778688 time=0.040 ms comps=4
MATCH 51f74873c2839b4e9bca 4112498425856 time=0.040 ms comps=4
MATCH 51f74820d16012dffda9 169079474421760 time=0.040 ms comps=4
MATCH 51f7489f406b08a47b26 104413758947328 time=0.040 ms comps=4
[6] 329b1f MATCH 329b1f159c3ba5023832 226377760571392 time=0.040 ms comps=4
MATCH 329b1fc9247f229b3c3f 256692545650688 time=0.040 ms comps=4
MATCH 329b1ff9ca18daa70a40 122168046321664 time=0.040 ms comps=4
MATCH 329b1ff34468c61ba528 127185390272512 time=0.040 ms comps=4
[7] 21aefa MATCH 21aefa03676de73cc537 180669460381696 time=0.041 ms comps=1
[8] 9e19a3 MATCH 9e19a3c2a0ac1cba1ef0 55166271422464 time=0.053 ms comps=3
MATCH 9e19a3ac1dc534aa8820 201843733102592 time=0.053 ms comps=3
MATCH 9e19a3e15fe2834bea19 79994940424192 time=0.053 ms comps=3
[9] 66a446 MATCH 66a44638447bf016491c 234814452203520 time=0.041 ms comps=4
MATCH 66a4468c151addf14d5d 162249603874816 time=0.041 ms comps=4
MATCH 66a446f61984400b77bb 262952477392896 time=0.041 ms comps=4
MATCH 66a4461ea8513780f333 142229553283072 time=0.041 ms comps=4
Search Summary: requested=10 performed=10 found_queries=10 total_matches=33 notfound=0
total_time=0.000433 s avg_ms=0.043 ms searches/sec=23081.842 total_seeks=10
avg_seeks_per_search=1.000 total_comps=33 avg_comps_per_search=3.300
avg_matches_per_found=3.300

cc@hw4-raicu-skylake:~/vault$ ./vaultx -k 26 -f k26-memo.x -s 10 -q 4 -d true
searches=10 difficulty=4
Parsed k                     : 26
Nonce Size                   : 6
Record Size                  : 16
Hash Size                    : 10
On-disk Record Size          : 16
Number of Buckets            : 16777216
Number of Records in Bucket  : 4
Number of Hashes      : 67108864
File Size to be read (bytes) : 1073741824
File Size to be read (GB)    : 1.000000
Actual file size on disk     : 1073741824 bytes
[0] 3267c542 NOTFOUND time=0.052 ms comps=0
[1] 26992e1a NOTFOUND time=0.042 ms comps=4
[2] 6caaeaad NOTFOUND time=0.041 ms comps=4
[3] 5353ac49 NOTFOUND time=0.041 ms comps=4
[4] fb623cf5 NOTFOUND time=0.041 ms comps=4
[5] b7c255f2 NOTFOUND time=0.041 ms comps=2
[6] acbbc8a2 NOTFOUND time=0.040 ms comps=3
[7] 5c5f7153 NOTFOUND time=0.040 ms comps=4
[8] bf143a9a NOTFOUND time=0.041 ms comps=2
[9] ffd3869c NOTFOUND time=0.040 ms comps=3
Search Summary: requested=10 performed=10 found_queries=0 total_matches=0 notfound=10
total_time=0.000419 s avg_ms=0.042 ms searches/sec=23893.378 total_seeks=10
```

```
avg_seeks_per_search=1.000 total_comps=30 avg_comps_per_search=3.000
avg_matches_per_found=0.000
```

You must run a number of search workloads to fill in the following table:

| K | Difficulty | Number of Searches | Average number of disk seek per search | Average Data Read per search (bytes) | Total Time for all searches | Time (ms) / search | Throughput search/sec | Number of Searches Found | Number of Searches Not Found |
|---|---|---|---|---|---|---|---|---|---|
| 26 | 3 | 1000 | | | | | | | |
| 26 | 4 | 1000 | | | | | | | |
| 26 | 5 | 1000 | | | | | | | |
| 32 | 3 | 1000 | | | | | | | |
| 32 | 4 | 1000 | | | | | | | |
| 32 | 5 | 1000 | | | | | | | |

You are to create a makefile that helps build your benchmark, as well as run it through the benchmark bash script. You must be able to configure the NONCE_SIZE and HASH_SIZE at compile time, or possibly at runtime through a command line interface (e.g. make vaultx_x86_c NONCE_SIZE=6 RECORD_SIZE=16)

Other requirements:

- You must write all benchmarks from scratch. Do not use code you find online, as you will get 0 credit for this assignment. This is also an individual group assignment, make sure you do this assignment by yourself.
- All of the benchmarks will have to evaluate concurrency performance; concurrency can be achieved using processes or threads.
- Not all timing functions have the same accuracy; you must find one that has at least 1ms accuracy or better, assuming you are running the benchmarks for at least seconds at a time.
- Since there are many experiments to run, you must use a bash script to automate the performance evaluation.
- You must use binary data when writing in this benchmark.
- No GUIs are required. Simple command line interfaces are required.

## 3 Extra Credit

**Compression (up to 10%):** If you can find a way to reduce the amount of storage needed, to save less data than 16 bytes per record, you will likely get improved performance.
**Leaderboard (up to 10%):** If you will publish your best results of the 64GB file with various memory configurations, you will get up to 10% extra credit points depending on the ranking in the leaderboard. The #1 spot on the night of the deadline will receive 10% extra credit (with

proper verification). The #2 spot will receive 9%. #3 will get 8%, and so on. The top 9 spots will receive between 2% and 10% extra credit. Everyone else will receive 1% extra credit for submitting. Your solution must be correct functionally to receive the extra credit. Leaderboard submission form: https://docs.google.com/forms/d/e/1FAIpQLSfNUoa_yuwbD8Os1JSnrK8Vfpbpst0qCMxbaIrL-1OloAOZ6A/viewform

### 3 What you will submit

When you have finished implementing the complete assignment as described above, you should submit your solution to your private git repository. Each program must work correctly and be detailed in-line documented. You should hand in:

1. **Source code and compilation (60%):** All of the source code, as well as bash scripts; in order to get full credit for the source code, your code must have in-line documents, must compile (with a build file), and must be able to run a variety of benchmarks through command line arguments as outlined.
2. **Report / Performance (40%):** A separate (typed) design document (named hw4-report.pdf) describing the results in a table format and in plots. You must summarize your findings, in terms of scalability, concurrency, and performance. For your particular setup (everyone will be different since you are running on different hardware), what seems to be the best configuration and worst configuration for each workload, in terms of concurrency level and record size? Highlight the best one in green, and worst one in red for the SMALL workload, and the LARGE workload. Can you explain in words why they are the best, or the worst, and why the data you have makes sense. You must include the sample output of running K=26 for both in-memory and out-of-memory, along with verification, print of first 100 records, and searching for 10 random queries for difficulty of 3 and 4.

You will have to submit your solution to a private git repository created for you. The repository is create through GitHub Classroom and you will need to accept the assignment before you can clone it, by accessing this invitation link: https://classroom.github.com/a/3z7gXw1N. The first time you access an invitation link the system will ask you to identify yourself. Please select the identifier that contains your school email address. After that you can clone the repository. Then you will have to add or update your source code, documentation, and report. All repositories will be collected automatically at midnight on the day of the deadline, and then one last time 24 hours after the deadline. The timestamp on your git commits will be used to determine if the submission is on-time; do not make edits to your repository after the deadline. If you cannot access your repository, contact the TAs. You can find a git cheat sheet here: https://www.git-tower.com/blog/git-cheat-sheet/

**Submit step: code/report through GIT.**

**Grades for late programs will be lowered 20%; no submission will be accepted more than 24 hours late**