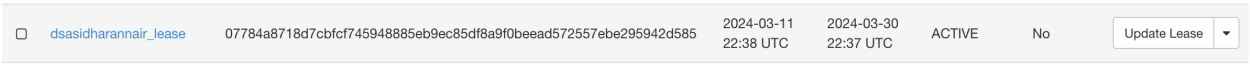
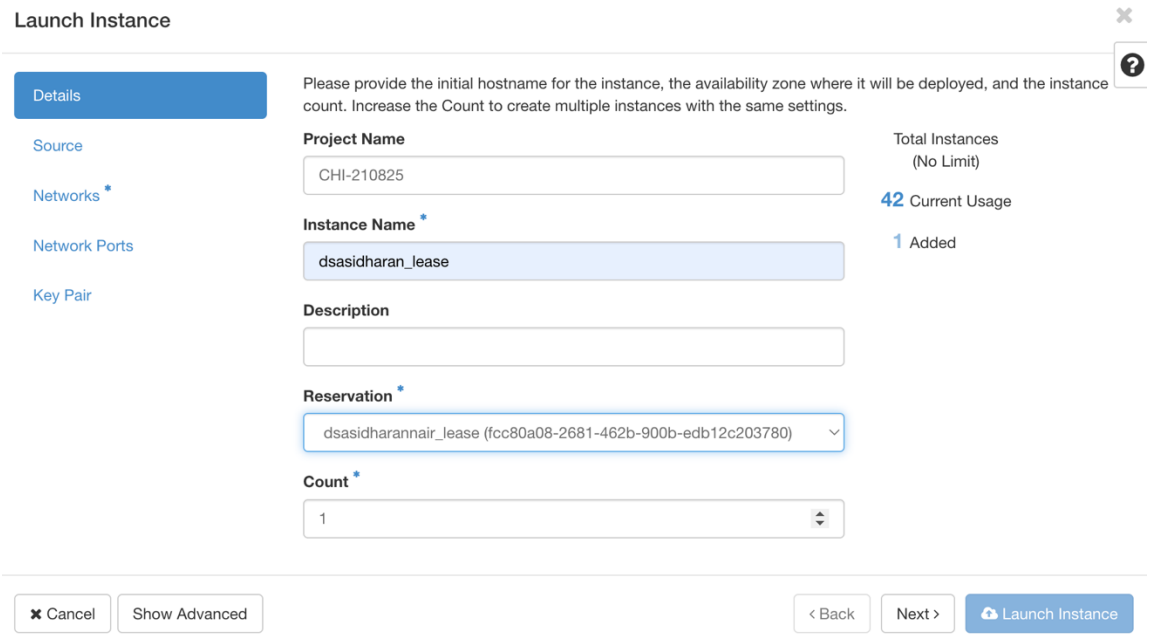
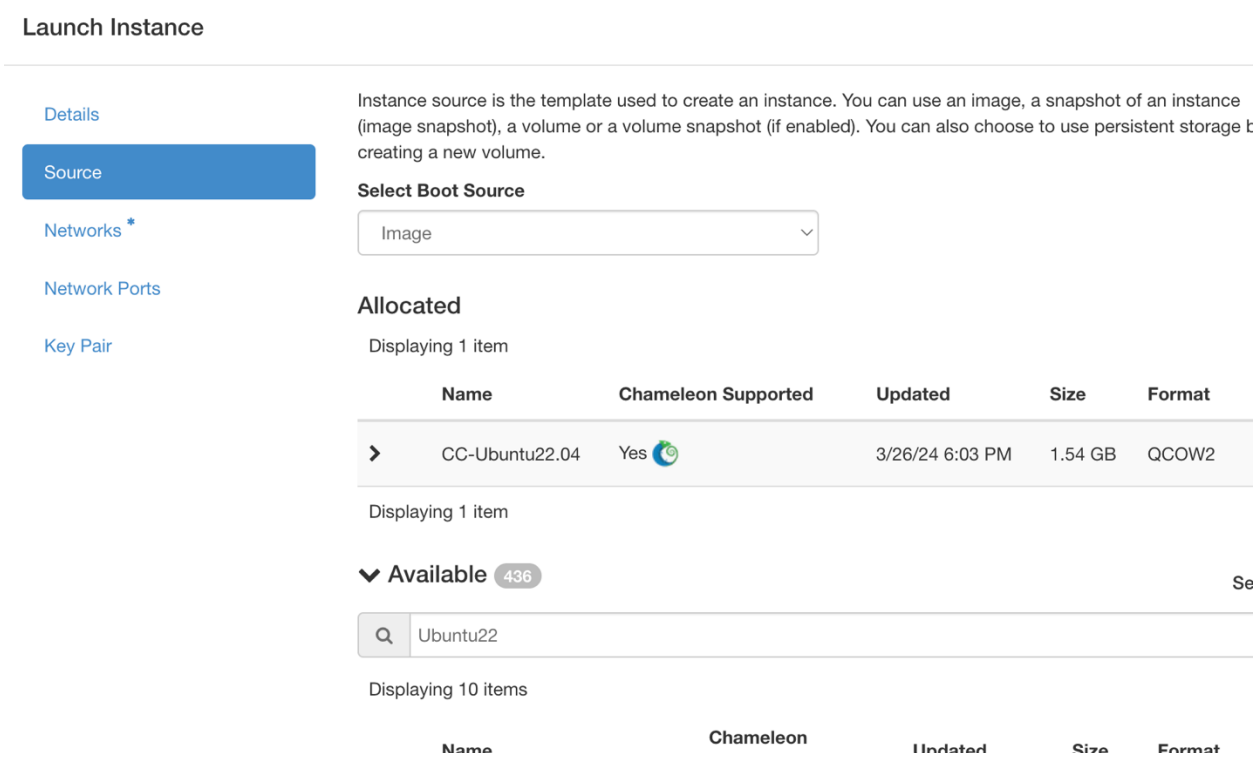
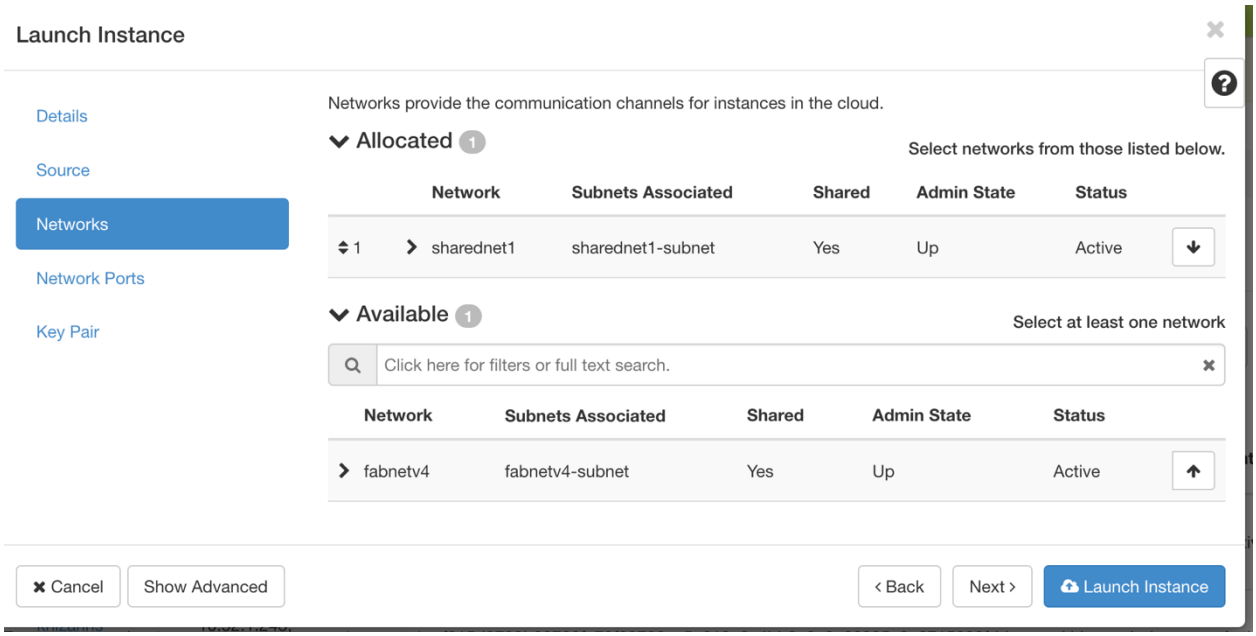
Homework 4 – Report
-Darshan S Nair

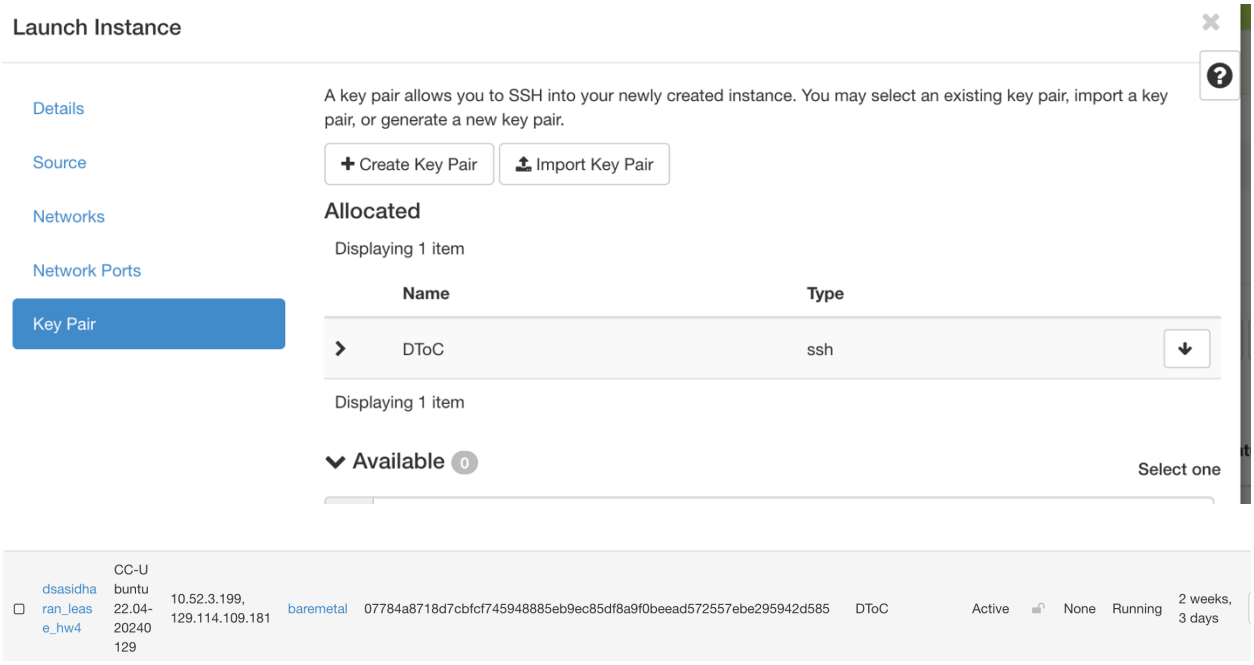
STEP 1 : Creating the Chameleon Instance






Launch Instance
Details
Source
Networks *
Network Ports
Key Pair
Instance source is the template used to create an instance. You can use an image, a snapshot of an instance (image snapshot), a volume or a volume snapshot (if enabled). You can also choose to use persistent storage b creating a new volume.
Select Boot Source
Image
Allocated
Displaying 1 item
Name
Chameleon Supported
Updated
Size
Format
CC-Ubuntu22.04
Yes
3/26/24 6:03 PM
1.54 GB
QCOW2
Displaying 1 item
Available 436
Se
Ubuntu22
Displaying 10 items
Chameleon


Launch Instance
Details
Source
Networks
Network Ports
Key Pair
Networks provide the communication channels for instances in the cloud.
Allocated 1
Select networks from those listed below.
Network
Subnets Associated
Shared
Admin State
Status
1
sharednet1
sharednet1-subnet
Yes
Up
Active
Available 1
Select at least one network
Click here for filters or full text search.
Network
Subnets Associated
Shared
Admin State
Status
fabnetv4
fabnetv4-subnet
Yes
Up
Active
Cancel
Show Advanced
Back
Next
Launch Instance



STEP 2: Writing Code and Running Tests

There are a few screenshots below that are part of the code I wrote. I have included screenshots of the bash scripts as well as the code running. For the full code refer to the hashgen.c file in the repo.

```
    max_records= (max_mem * 1048576) / 16;
    int iterations;
    if(file_size == 1024){
        total_hash_count = HASH_COUNT_SMALL;
    }else{
        total_hash_count = HASH_COUNT_BIG;
    }
    iterations= total_hash_count / max_records;

    double start_time = get_current_time();

    for (int j = 0; j < iterations; j++){

        hashes = malloc(max_records * sizeof(Record));

        for (int i = 0; i < hash_threads; i++) {
            struct ThreadArgs *args = malloc(sizeof(struct ThreadArgs));
            args->thread_index = i;
            args->iteration_index = (ull)j;
            pthread_create(&hash_threads_array[i], NULL,hashing, args);
        }
        for (int i = 0; i < hash_threads; i++) {
            pthread_join(hash_threads_array[i], NULL);
        }

        for (int i = 0; i < sort_threads; i++) {
            pthread_create(&sort_threads_array[i], NULL, sorting, (void *)(intptr_t)i);
        }

        for (int i = 0; i < sort_threads; i++) {
            pthread_join(sort_threads_array[i], NULL);
        }

        if(write_threads >= sort_threads){
            for (int i = 0; i < sort_threads; i++) {
                struct WriteThreadArgs1 *args = malloc(sizeof(struct WriteThreadArgs1));
                args->i = i;
```

A Sample Invocation

```
cc@dsasidharan-lease-hw4:~$ ./hashgen -t 16 -o 1 -i 4 -f data.bin -m 128 -s 1024 -d true
NUM_THREADS_HASH=16
NUM_THREADS_SORT=1
NUM_THREADS_WRITE=4
FILENAME=data.bin
MEMORY_SIZE=128MB
FILE_SIZE=1024MB
RECORD_SIZE=16B
HASH_SIZE=10B
NONCE_SIZE=6B
[0][HASHGEN]: 12.50% completed, ETA 20.8 seconds, 8388608/67108864 hashes, 43.0 MB/sec
[1][HASHGEN]: 25.00% completed, ETA 18.1 seconds, 16777216/67108864 hashes, 42.4 MB/sec
[2][HASHGEN]: 37.50% completed, ETA 15.3 seconds, 25165824/67108864 hashes, 41.8 MB/sec
[3][HASHGEN]: 50.00% completed, ETA 11.7 seconds, 33554432/67108864 hashes, 43.9 MB/sec
[4][HASHGEN]: 62.50% completed, ETA 8.5 seconds, 41943040/67108864 hashes, 45.2 MB/sec
[5][HASHGEN]: 75.00% completed, ETA 5.7 seconds, 50331648/67108864 hashes, 44.8 MB/sec
[6][HASHGEN]: 87.50% completed, ETA 2.8 seconds, 58720256/67108864 hashes, 45.7 MB/sec
[7][HASHGEN]: 100.00% completed, ETA 0.0 seconds, 67108864/67108864 hashes, 46.4 MB/sec
Moving to External Sort, 8 sub files detected
External Sort Started. Expected 16 flushes for 8 files
[8][SORT]: 6.25% completed, ETA 9.2 seconds, 1/16 flushes, 104.3 MB/sec
[9][SORT]: 12.50% completed, ETA 8.2 seconds, 2/16 flushes, 109.7 MB/sec
[10][SORT]: 18.75% completed, ETA 7.4 seconds, 3/16 flushes, 111.7 MB/sec
[11][SORT]: 25.00% completed, ETA 6.7 seconds, 4/16 flushes, 115.1 MB/sec
[12][SORT]: 31.25% completed, ETA 5.7 seconds, 5/16 flushes, 123.4 MB/sec
[13][SORT]: 37.50% completed, ETA 4.9 seconds, 6/16 flushes, 129.8 MB/sec
[14][SORT]: 43.75% completed, ETA 4.3 seconds, 7/16 flushes, 134.7 MB/sec
[15][SORT]: 50.00% completed, ETA 3.7 seconds, 8/16 flushes, 138.7 MB/sec
[16][SORT]: 56.25% completed, ETA 3.2 seconds, 9/16 flushes, 142.0 MB/sec
[17][SORT]: 62.50% completed, ETA 2.7 seconds, 10/16 flushes, 144.7 MB/sec
[18][SORT]: 68.75% completed, ETA 2.2 seconds, 11/16 flushes, 147.1 MB/sec
[19][SORT]: 75.00% completed, ETA 1.7 seconds, 12/16 flushes, 149.1 MB/sec
[20][SORT]: 81.25% completed, ETA 1.3 seconds, 13/16 flushes, 150.7 MB/sec
[21][SORT]: 87.50% completed, ETA 0.8 seconds, 14/16 flushes, 152.2 MB/sec
[22][SORT]: 93.75% completed, ETA 0.4 seconds, 15/16 flushes, 153.5 MB/sec
[23][SORT]: 100.00% completed, ETA 0.0 seconds, 16/16 flushes, 154.7 MB/sec
Completed 1024 MB file data.bin in 28.80 seconds : 2.33 MH/s 35.55 MB/s
```

```
cc@dsasidharan-lease-hw4:~$ ./hashverify -f data.bin -p 10
Printing first 10 of file 'data.bin'...
[0] Hash: 000000744acc5940a01b : d92c93010000 : 26422489
[16] Hash: 0000009fcad95785e04e : a40913010000 : 18024868
[32] Hash: 000001012778f96f23bd : 4deb68030000 : 57207629
[48] Hash: 000001bb43255f863484 : 077914000000 : 1341703
[64] Hash: 0000023640c2b8cc714d : eb4df6030000 : 66473451
[80] Hash: 000002475cea698b6761 : 66d84d020000 : 38656102
[96] Hash: 000002bf1e0337e0221b : 235afe010000 : 33446435
[112] Hash: 0000030bc90ed2b07331 : 987112000000 : 1208728
[128] Hash: 0000033687c51d72ee2d : 0f903f000000 : 4165647
[144] Hash: 0000036e8631726b50e8 : 0b83ed010000 : 32342795
```

```
cc@dsasidharan-lease-hw4:~$ ./hashverify -f data.bin -v true
Read 1073741824 bytes and found all records are sorted.
```

run_bench_s.sh

```bash
#!/bin/bash

EXE="./build/hashgen"
VER="./hashverify"
values=(1 4 16)

> "results_s.csv"

for t in "${values[@]}"; do
    for o in "${values[@]}"; do
        for i in "${values[@]}"; do
            echo "Starting Benchmark for -t $t -o $o -i $i -f "data.bin" -m 128 -s 1024 -d false config"
            output=$("$EXE" -t "$t" -o "$o" -i "$i" -f "data.bin" -m 128 -s 1024 -d "false")
            echo "$output"
            time_taken=$(echo "$output" | awk '{print $7}')
            MHs=$(echo "$output" | awk '{print $8}')
            MBs=$(echo "$output" | awk '{print $9}')
            echo "$t-$o-$i,$time_taken,$MHs,$MBs" >> "results_s.csv"
            $VER -f "data.bin" -v "true"
            rm "data.bin"
        done
    done
done

python3 generate_graph.py s
```

I have a similar script for 64 GB as well

This is the execution of the benchmark :

```
Read 1073741824 bytes and found all records are sorted.
Starting Benchmark for -t 4 -o 1 -i 1 -f data.bin -m 128 -s 1024 -d false config
hashgen t4 o1 i1 m128 s1024 29.09 2.31 35.20
Read 1073741824 bytes and found all records are sorted.
Starting Benchmark for -t 4 -o 1 -i 4 -f data.bin -m 128 -s 1024 -d false config
hashgen t4 o1 i4 m128 s1024 29.59 2.27 34.60
Read 1073741824 bytes and found all records are sorted.
Starting Benchmark for -t 4 -o 1 -i 16 -f data.bin -m 128 -s 1024 -d false config
hashgen t4 o1 i16 m128 s1024 28.54 2.35 35.88
Read 1073741824 bytes and found all records are sorted.
Starting Benchmark for -t 4 -o 4 -i 1 -f data.bin -m 128 -s 1024 -d false config
hashgen t4 o4 i1 m128 s1024 33.73 1.99 30.36
Read 1073741824 bytes and found all records are sorted.
Starting Benchmark for -t 4 -o 4 -i 4 -f data.bin -m 128 -s 1024 -d false config
hashgen t4 o4 i4 m128 s1024 32.68 2.05 31.34
Read 1073741824 bytes and found all records are sorted.
Starting Benchmark for -t 4 -o 4 -i 16 -f data.bin -m 128 -s 1024 -d false config
hashgen t4 o4 i16 m128 s1024 32.92 2.04 31.10
Read 1073741824 bytes and found all records are sorted.
Starting Benchmark for -t 4 -o 16 -i 1 -f data.bin -m 128 -s 1024 -d false config
hashgen t4 o16 i1 m128 s1024 97.43 0.69 10.51
Read 1073741824 bytes and found all records are sorted.
Starting Benchmark for -t 4 -o 16 -i 4 -f data.bin -m 128 -s 1024 -d false config
hashgen t4 o16 i4 m128 s1024 99.74 0.67 10.27
Read 1073741824 bytes and found all records are sorted.
Starting Benchmark for -t 4 -o 16 -i 16 -f data.bin -m 128 -s 1024 -d false config
hashgen t4 o16 i16 m128 s1024 97.03 0.69 10.55
Read 1073741824 bytes and found all records are sorted.
Starting Benchmark for -t 16 -o 1 -i 1 -f data.bin -m 128 -s 1024 -d false config
hashgen t16 o1 i1 m128 s1024 27.19 2.47 37.66
Read 1073741824 bytes and found all records are sorted.
Starting Benchmark for -t 16 -o 1 -i 4 -f data.bin -m 128 -s 1024 -d false config
hashgen t16 o1 i4 m128 s1024 28.29 2.37 36.19
Read 1073741824 bytes and found all records are sorted.
Starting Benchmark for -t 16 -o 1 -i 16 -f data.bin -m 128 -s 1024 -d false config
hashgen t16 o1 i16 m128 s1024 27.77 2.42 36.88
Read 1073741824 bytes and found all records are sorted.
Starting Benchmark for -t 16 -o 4 -i 1 -f data.bin -m 128 -s 1024 -d false config
hashgen t16 o4 i1 m128 s1024 31.87 2.11 32.13
Read 1073741824 bytes and found all records are sorted.
Starting Benchmark for -t 16 -o 4 -i 4 -f data.bin -m 128 -s 1024 -d false config
hashgen t16 o4 i4 m128 s1024 31.04 2.16 32.99
Read 1073741824 bytes and found all records are sorted.
Starting Benchmark for -t 16 -o 4 -i 16 -f data.bin -m 128 -s 1024 -d false config
hashgen t16 o4 i16 m128 s1024 31.13 2.16 32.90
Read 1073741824 bytes and found all records are sorted.
Starting Benchmark for -t 16 -o 16 -i 1 -f data.bin -m 128 -s 1024 -d false config
hashgen t16 o16 i1 m128 s1024 95.71 0.70 10.70
Read 1073741824 bytes and found all records are sorted.
Starting Benchmark for -t 16 -o 16 -i 4 -f data.bin -m 128 -s 1024 -d false config
hashgen t16 o16 i4 m128 s1024 97.81 0.69 10.47
Read 1073741824 bytes and found all records are sorted.
Starting Benchmark for -t 16 -o 16 -i 16 -f data.bin -m 128 -s 1024 -d false config
hashgen t16 o16 i16 m128 s1024 96.48 0.70 10.61
Read 1073741824 bytes and found all records are sorted.
```

STEP 3: Gathering and Analyzing the Results

Cpu Specifications

```
[cc@dsasidharan-lease-hw4:~$ lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         46 bits physical, 57 bits virtual
  Byte Order:            Little Endian
CPU(s):                  160
  On-line CPU(s) list:   0-159
Vendor ID:               GenuineIntel
  Model name:            Intel(R) Xeon(R) Platinum 8380 CPU @ 2.30GHz
    CPU family:          6
    Model:               106
    Thread(s) per core:  2
    Core(s) per socket:  40
    Socket(s):           2
    Stepping:            6
    CPU max MHz:         3400.0000
    CPU min MHz:         800.0000
    BogoMIPS:            4600.00
    Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon
                         pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic movbe
                          popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb cat_l3 invpcid_single intel_ppin ssbd mba ibrs ibpb stibp ibrs_enhanced tpr_shadow v
                         nmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid cqm rdt_a avx512f avx512dq rdseed adx smap avx512ifma clflushopt clwb intel_pt avx512cd sha_n
                         i avx512bw avx512vl xsaveopt xsavec xgetbv1 xsaves cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local split_lock_detect wbnoinvd dtherm ida arat pln pts hwp hwp_act_window hwp_epp
                         hwp_pkg_req avx512vbmi umip pku ospke avx512_vbmi2 gfni vaes vpclmulqdq avx512_vnni avx512_bitalg tme avx512_vpopcntdq la57 rdpid fsrm md_clear pconfig flush_l1d arch_capabilities
Virtualization features:
  Virtualization:        VT-x
Caches (sum of all):
  L1d:                   3.8 MiB (80 instances)
  L1i:                   2.5 MiB (80 instances)
  L2:                    100 MiB (80 instances)
  L3:                    120 MiB (2 instances)
NUMA:
  NUMA node(s):          2
  NUMA node0 CPU(s):     0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46,48,50,52,54,56,58,60,62,64,66,68,70,72,74,76,78,80,82,84,86,88,90,92,94,96,98,100,102,104,106,108,110,112,114,11
                         6,118,120,122,124,126,128,130,132,134,136,138,140,142,144,146,148,150,152,154,156,158
  NUMA node1 CPU(s):     1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51,53,55,57,59,61,63,65,67,69,71,73,75,77,79,81,83,85,87,89,91,93,95,97,99,101,103,105,107,109,111,113,115,11
                         7,119,121,123,125,127,129,131,133,135,137,139,141,143,145,147,149,151,153,155,157,159
Vulnerabilities:
  Gather data sampling:  Mitigation; Microcode
  Itlb multihit:         Not affected
  L1tf:                  Not affected
  Mds:                   Not affected
  Meltdown:              Not affected
  Mmio stale data:       Mitigation; Clear CPU buffers; SMT vulnerable
  Retbleed:              Not affected
  Spec rstack overflow:  Not affected
  Spec store bypass:     Mitigation; Speculative Store Bypass disabled via prctl and seccomp
  Spectre v1:            Mitigation; usercopy/swapgs barriers and __user pointer sanitization
  Spectre v2:            Mitigation; Enhanced IBRS, IBPB conditional, RSB filling, PBRSB-eIBRS SW sequence
  Srbds:                 Not affected
  Tsx async abort:       Not affected
```

Disk Specifications

```
lsblk: command not found
cc@dsasidharan-lease-hw4:~$ lsblk
NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
loop0     7:0    0  63.9M  1 loop /snap/core20/2105
loop2     7:2    0  40.4M  1 loop /snap/snapd/20671
loop3     7:3    0  63.9M  1 loop /snap/core20/2182
loop4     7:4    0    87M  1 loop /snap/lxd/27428
loop5     7:5    0  39.1M  1 loop /snap/snapd/21184
loop6     7:6    0    87M  1 loop /snap/lxd/27948
sda       8:0    0 447.1G  0 disk
├─sda1    8:1    0   550M  0 part /boot/efi
├─sda2    8:2    0     8M  0 part
├─sda3    8:3    0 446.5G  0 part /
└─sda4    8:4    0  64.8M  0 part
cc@dsasidharan-lease-hw4:~$
```

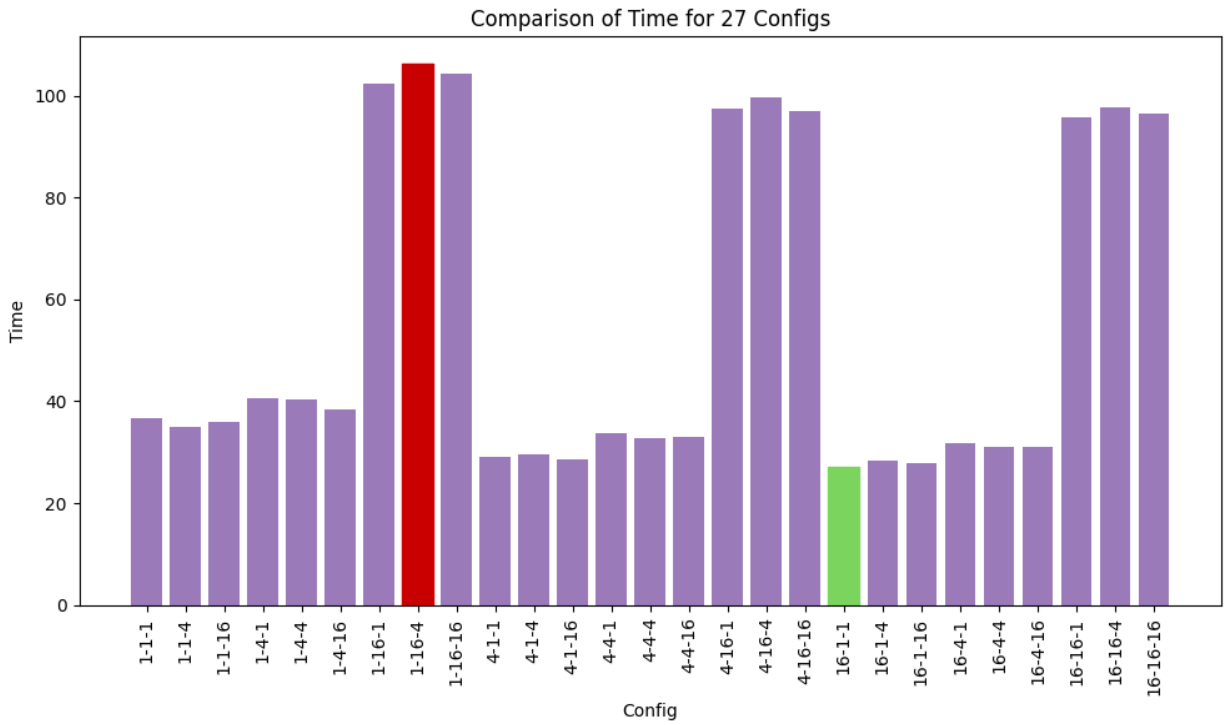
I do not have an SSD or NVME ☹

Results for 1GB file and 128MB Memory Specs

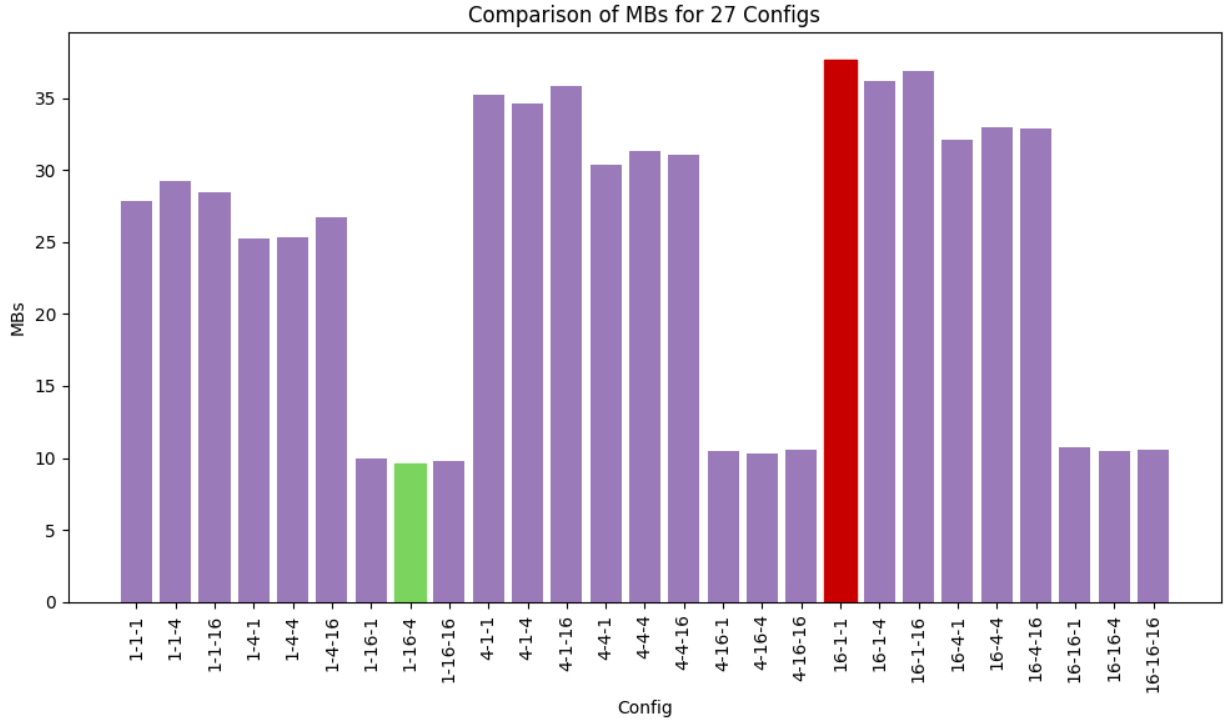
Hash Threads	Sort Threads	Write Threads	Time(s)	MH/s	MB/s
1	1	1	36.81	1.82	27.82

1	1	4	35.02	1.92	29.24
1	1	16	35.96	1.87	28.48
1	4	1	40.54	1.66	25.26
1	4	4	40.41	1.66	25.34
1	4	16	38.38	1.75	26.68
1	16	1	102.39	0.66	10
1	16	4	106.28	0.63	9.63
1	16	16	104.32	0.64	9.82
4	1	1	29.09	2.31	35.2
4	1	4	29.59	2.27	34.6
4	1	16	28.54	2.35	35.88
4	4	1	33.73	1.99	30.36
4	4	4	32.68	2.05	31.34
4	4	16	32.92	2.04	31.1
4	16	1	97.43	0.69	10.51
4	16	4	99.74	0.67	10.27
4	16	16	97.03	0.69	10.55
16	1	1	27.19	2.47	37.66
16	1	4	28.29	2.37	36.19
16	1	16	27.77	2.42	36.88
16	4	1	31.87	2.11	32.13
16	4	4	31.04	2.16	32.99
16	4	16	31.13	2.16	32.9
16	16	1	95.71	0.7	10.7
16	16	4	97.81	0.69	10.47
16	16	16	96.48	0.7	10.61

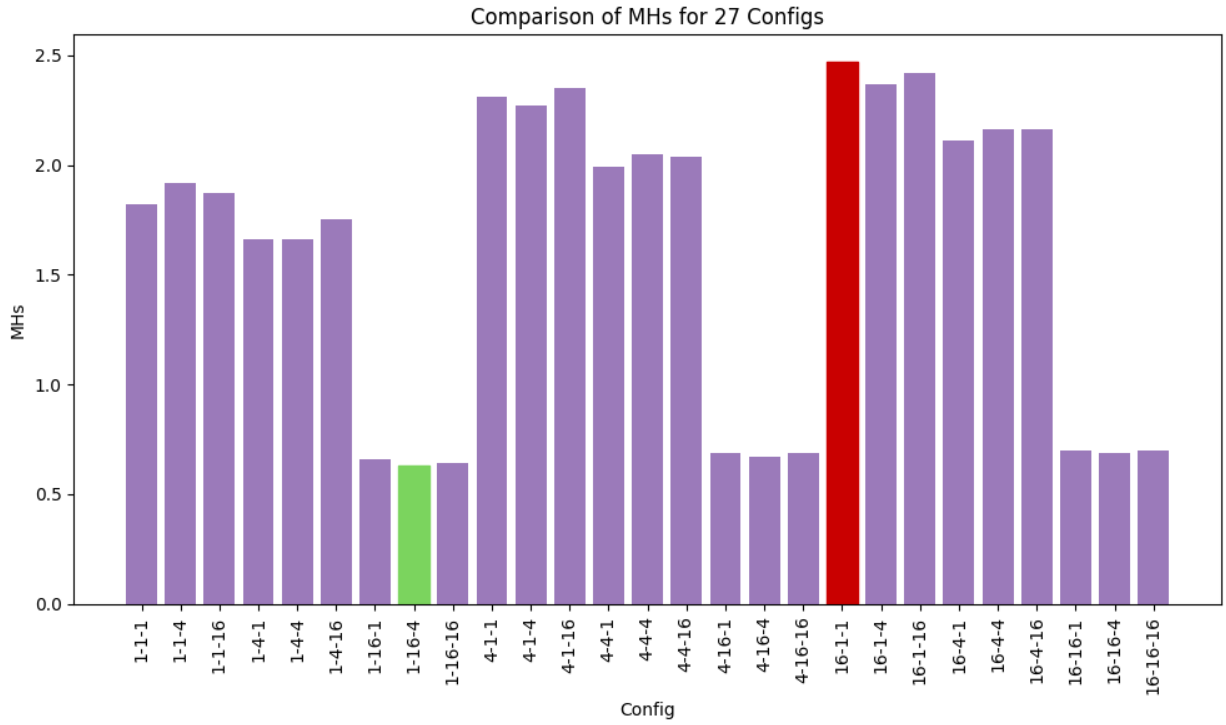
Time Graph


Comparison of Time for 27 Configs
Time
0
20
40
60
80
100
1-1-1
1-1-4
1-1-16
1-4-1
1-4-4
1-4-16
1-16-1
1-16-4
1-16-16
4-1-1
4-1-4
4-1-16
4-4-1
4-4-4
4-4-16
4-16-1
4-16-4
4-16-16
16-1-1
16-1-4
16-1-16
16-4-1
16-4-4
16-4-16
16-16-1
16-16-4
16-16-16
Config

MB/s Graph


Comparison of MBs for 27 Configs
MBs
0
5
10
15
20
25
30
35
1-1-1
1-1-4
1-1-16
1-4-1
1-4-4
1-4-16
1-16-1
1-16-4
1-16-16
4-1-1
4-1-4
4-1-16
4-4-1
4-4-4
4-4-16
4-16-1
4-16-4
4-16-16
16-1-1
16-1-4
16-1-16
16-4-1
16-4-4
16-4-16
16-16-1
16-16-4
16-16-16
Config

MH/s Graph



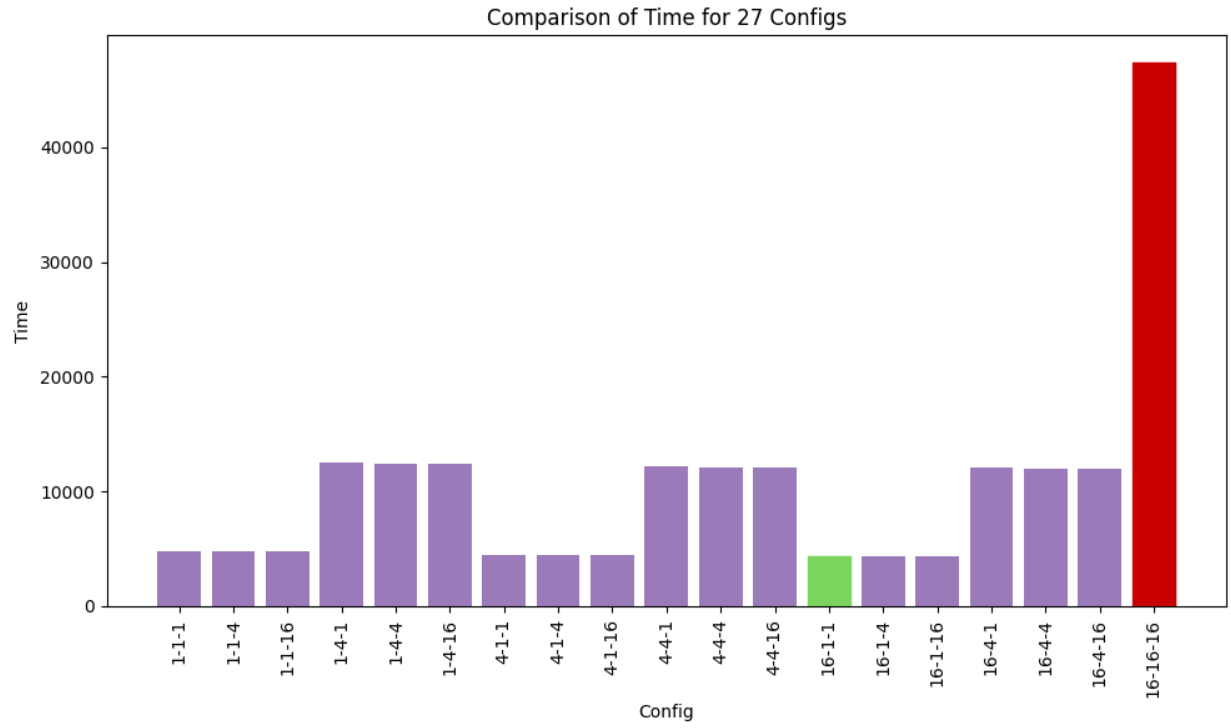
Results for 64GB file and 1024MB Memory Specs

I was unable to run the rest of the 8 configurations where the number of the sort threads is 16 as each test takes close to 13 hours to run.

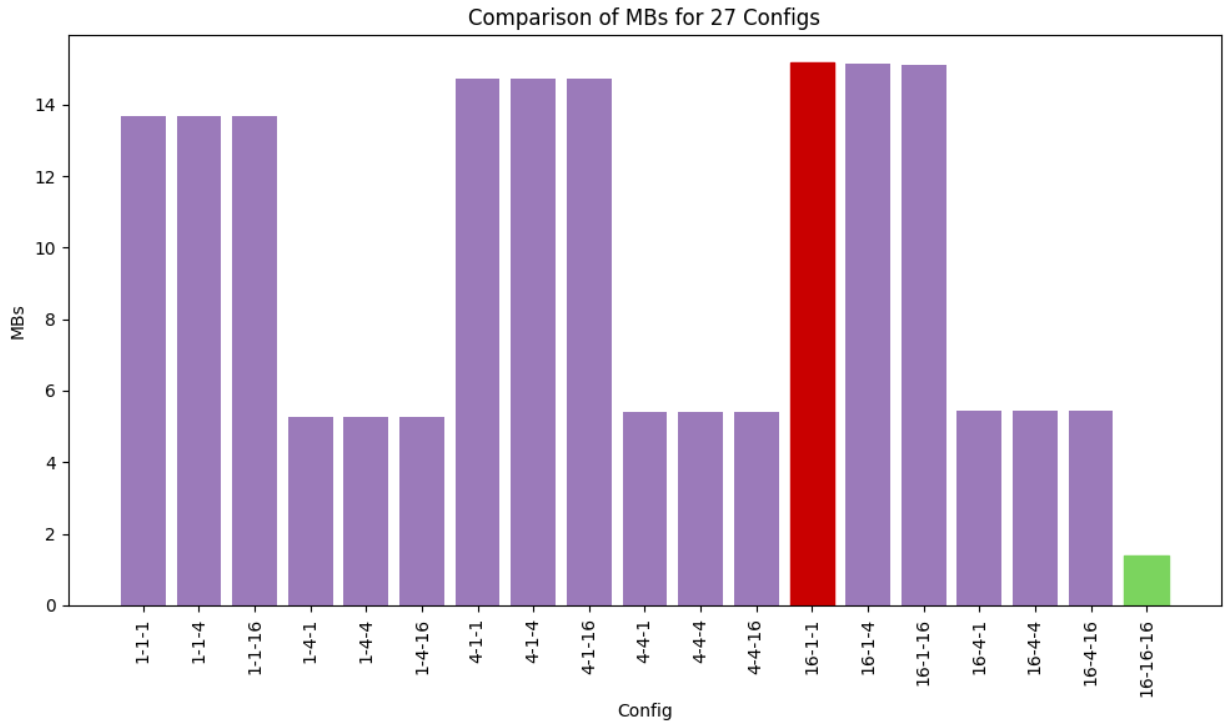
Hash Threads	Sort Threads	Write Threads	Time(s)	MHs	MBs
1	1	1	4790.8	0.9	13.68
1	1	4	4787.17	0.9	13.69
1	1	16	4785.9	0.9	13.69
1	4	1	12462.55	0.34	5.26
1	4	4	12450.31	0.34	5.26
1	4	16	12440.67	0.35	5.27
4	1	1	4451.03	0.96	14.72
4	1	4	4447.53	0.97	14.74
4	1	16	4449.88	0.97	14.73
4	4	1	12160.44	0.35	5.39

4	4	4	12118.91	0.35	5.41
4	4	16	12127.63	0.35	5.4
16	1	1	4317.52	0.99	15.18
16	1	4	4326.76	0.99	15.15
16	1	16	4332.3	0.99	15.13
16	4	1	12046.88	0.36	5.44
16	4	4	12022.77	0.36	5.45
16	4	16	12016.42	0.36	5.45
16	16	16	47370.59	0.09	1.38

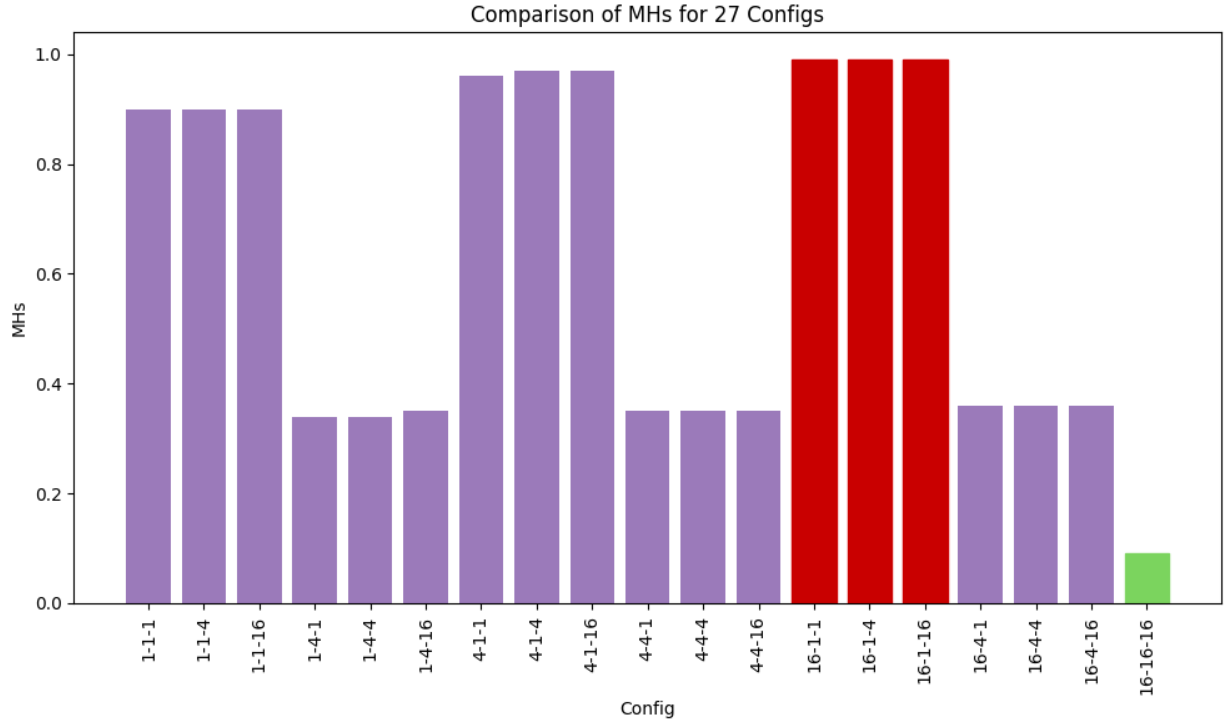
Time Graph



MB/s Graph


Comparison of MBs for 27 Configs
MBs
0
2
4
6
8
10
12
14
1-1-1
1-1-4
1-1-16
1-4-1
1-4-4
1-4-16
4-1-1
4-1-4
4-1-16
4-4-1
4-4-4
4-4-16
16-1-1
16-1-4
16-1-16
16-4-1
16-4-4
16-4-16
16-16-16
Config

MH/s


Comparison of MHs for 27 Configs
MHs
0.0
0.2
0.4
0.6
0.8
1.0
1-1-1
1-1-4
1-1-16
1-4-1
1-4-4
1-4-16
4-1-1
4-1-4
4-1-16
4-4-1
4-4-4
4-4-16
16-1-1
16-1-4
16-1-16
16-4-1
16-4-4
16-4-16
16-16-16
Config

Analysis of Scalability

The program is somewhat scalable from the 1GB case to the 64GB case. Although the metrics like MB/s and MH/s decreases for the 65GB case when compared to the 1GB case which is somewhat expected of the program. One key factor in scalability is the number of sort threads for the program. As the number of sort threads determine the number of sorted sub files that would be created, which is then external sorted into a single file. It can be noted that as the number of sort threads increases so does the number of sub files which increases the complexity of the external sort. This directly contributes to the issue of scalability for the program. As the number of hashes to be created is much more for the 64GB case there are more subfiles and hence takes longer. This is why the performance ios better when the number of sort threads is the lowest.

Analysis of Concurrency

Most of the concurrency in the program comes from the first half of the program when the hashes are created, sorted and written to the file. The external sort exhibits limited concurrency. Only concurrency is the part of reading part of the sub files into memory rest of the program operated without concurrency. Therefore, it can be observed that cases where the number of hash threads and number of write threads is high, the performance and the time taken is considerably better. For the number of sort threads it is essentially a tradeoff analysis between the number of subfiles created and how fast the program runs for hash generation. For the 1GB case there is not much difference but the 64Gb case has a significant difference based on the number of sort threads. It essentially can be seen that the lower the number of sort threads the better the performance, which is expected due to the time complexity of the external sort.

Analysis of Performance

When moving from the 1GB to the 64GB example, the program shows a reasonable degree of scalability, albeit measurements such as Mega Bytes per second (MB/s) and Mega Hashes per second (MH/s) for the bigger dataset show a discernible decline. The greater complexity brought about by using more sort threads is the main cause of this decrease in performance. Because it directly affects the amount of sorted subfiles generated, the number of sort threads is a crucial factor in determining the external sort process's efficiency. Consequently, more subfiles result from more sort threads, which lengthens the sorting process and impairs scalability. The program's concurrency is most noticeable in the early phases of sorting, hash generation, and file writing, when more write and hash threads can use enhanced concurrency to boost performance. However, there is only a little amount of concurrency in the external sort phase, which is mostly limited to reading subfiles into memory. It's interesting to note that there is a trade-off between performance and the number of sort threads; configurations with fewer sort threads typically produce better results. Overall, while the program demonstrates a degree of scalability and concurrency utilization, there remains ample room for optimization, particularly in

streamlining the external sort process and fine-tuning the allocation of hash, sort, and write threads to maximize performance under varying workload conditions. Furthermore, by dividing a file into multiple sections and using the I/O threads to write to the file, we can improve the overall concurrency of the program. But I have not implemented that in my program.