Final Project Report

The following programs are run on Ubuntu container using docker on a M1 MacBook Pro machine with 10 cores and 32GB memory.

Part 1 Basics

A program is written using C library open() to open the file passed from the argument filename. read() and/or write() is called depending on the second argument. Block size and block count from the argument are used to specify how much to read/write.

Usage: ./run <filename> [-r|-w] <block_size> <block_count>

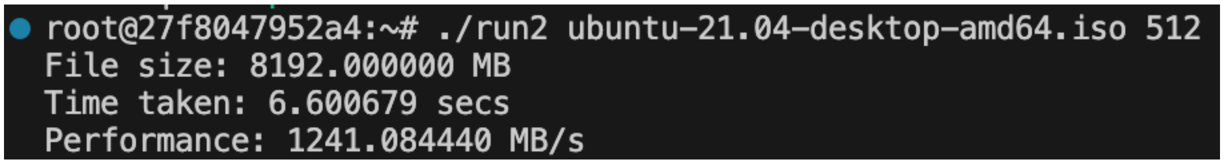
Part 2 Measurement

In order to find the optimal file size that can be read with the runtime between 5 and 15 seconds, we started off with block count = 1 and keeps doubling the block count until the runtime gets greater than 5 seconds. We chose to perform time calculation with gettimeofday() since it has more precision in decimals rather than the current time in seconds that time() returns.

- Input: block size
- Output: file size

Usage: ./run2 <filename> <block_size>

Sample output:



```
root@27f8047952a4:~# ./run2 ubuntu-21.04-desktop-amd64.iso 512
File size: 8192.000000 MB
Time taken: 6.600679 secs
Performance: 1241.084440 MB/s
```

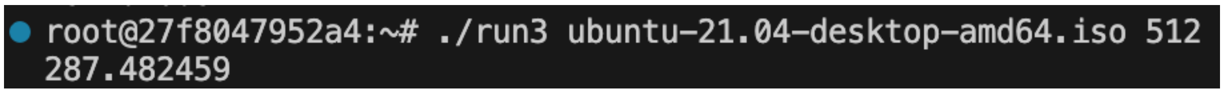
Part 3 Raw Performance

We used xorbuf() provided in the sample code while producing the performance graph. In order to produce the graph with different block sizes, we created a script **run3.sh** that runs **run3.c** with block size = 8, 8*2, 8*2*2, … , to 4194304.

- Input: block size
- Output: performance in MiB/s

Usage: ./run3 <filename> <block_size>

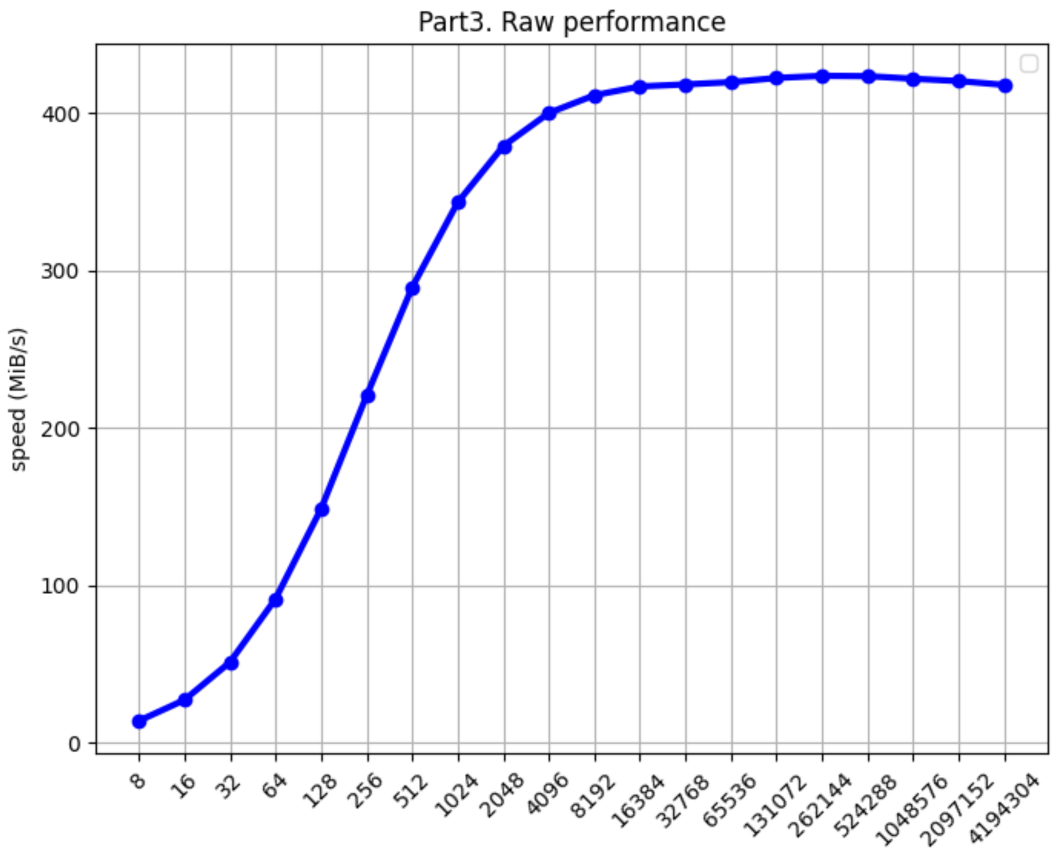
Sample output:



```
root@27f8047952a4:~# ./run3 ubuntu-21.04-desktop-amd64.iso 512
287.482459
```

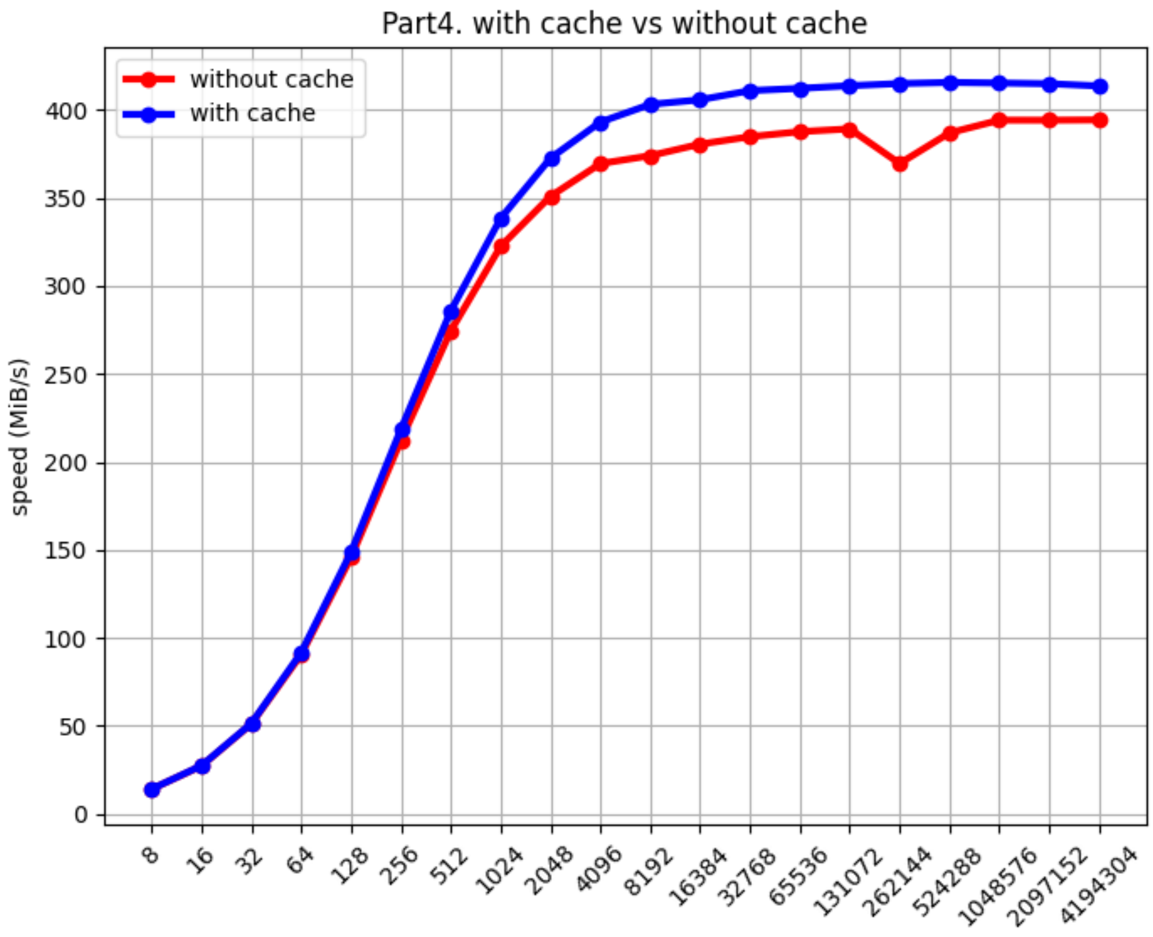
Script usage to run with different block size:

- chmod +x ./run3.sh
- ./run3.sh



Part 4 Caching

To run the program with and without caching, we created a script to clear the disk cache and run the program two times with every different number of block size. “sudo sh -c "/usr/bin/echo 3 > /proc/sys/vm/drop_caches”"was used in the script to clear the cache.



Script usage:

- chmod +x ./run4.sh
- ./run4.sh

Extra credit: Why “3"?

There are numbers “1”, “2”, “3” that represent 3 different level of cache:

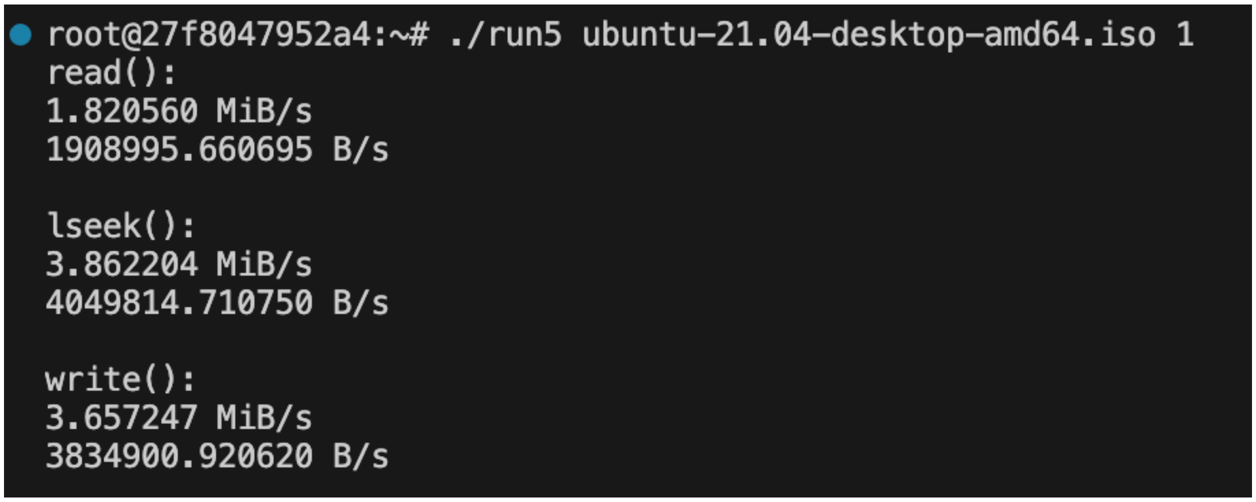
- echo 1: clear page cache
- echo 2: clear dentries and inodes
- echo 3: clear page cache, dentries and inodes

Page cache is a segment of RAM that is used to store recently accessed data from secondary storage. It is used to speed up access on disk.
Dentries are data structures that represent directories or folders. Every dentry maps an inode number to a filename and a parent directory.
Inodes are data structures that store file metadata, except file name and data content. They would contain information about ownership, access mode, file type, and more.

Part 5 System Calls

We did experiments with block size = 1 byte and tried various system calls to see their performance. The results are as follows:

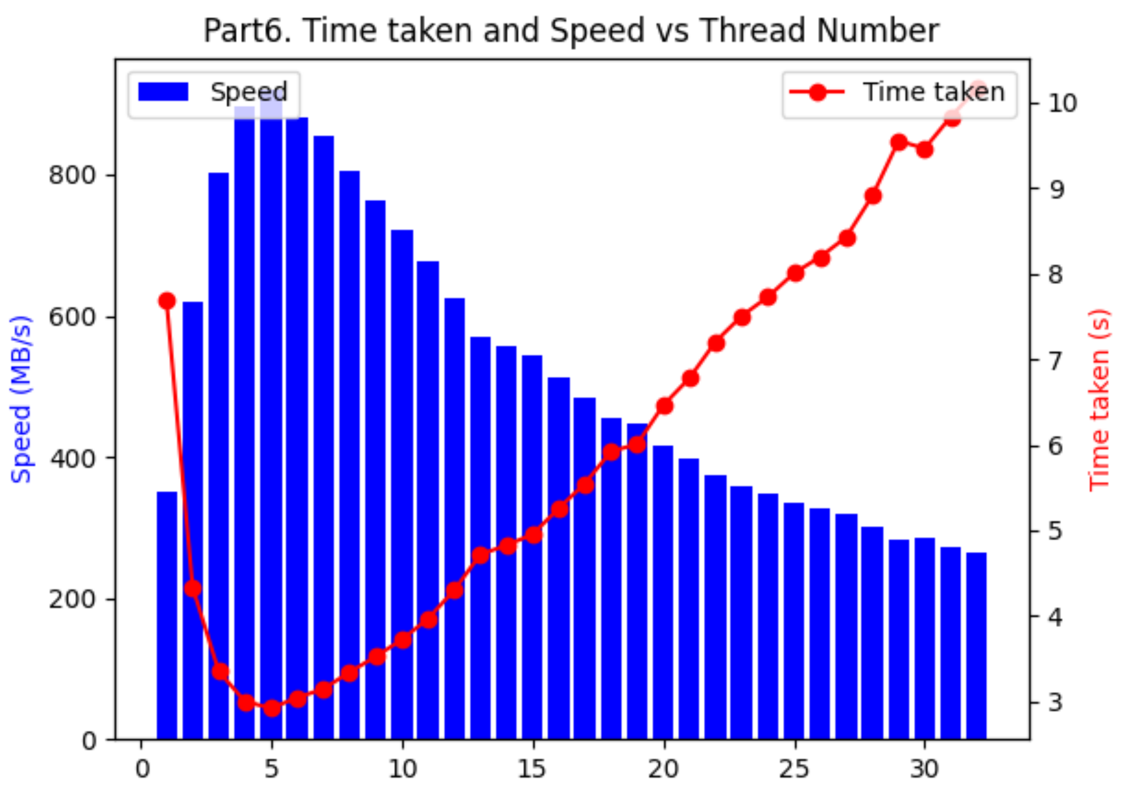
System call	Block size in byte	Performance in B/s	Performance in MiB/s
read()	1	1908995.660695	1.820560
lseek()	1	4049814.710750	3.862204
write()	1	3834900.920620	3.657247



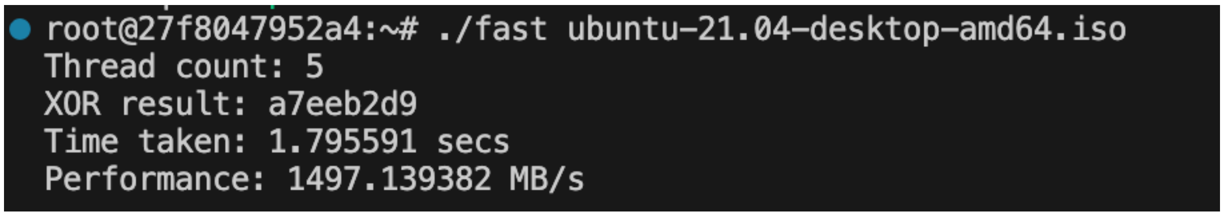
```
root@27f8047952a4:~# ./run5 ubuntu-21.04-desktop-amd64.iso 1
read():
1.820560 MiB/s
1908995.660695 B/s

lseek():
3.862204 MiB/s
4049814.710750 B/s

write():
3.657247 MiB/s
3834900.920620 B/s
```

Part 6 Raw Performance

From the previous experiments, we have chosen the block size to be 262144. In order to determine the optimal number of threads, we made a script to run the program with 1-32 threads (graph below).



Therefore, we have chosen to proceed with thread = 5 to get the best performance.



```
● root@27f8047952a4:~# ./fast ubuntu-21.04-desktop-amd64.iso
  Thread count: 5
  XOR result: a7eeb2d9
  Time taken: 1.795591 secs
  Performance: 1497.139382 MB/s
```

Usage: ./fast <filename>

	Performance in MiB/s	Seconds
Non-cached	1091.834302	2.462141
Cached	1419.892599	1.893277