# Contents

# 1. Benchmarking

## a. Process

Following Benchmarking experiments were commenced as part of Assignment

1. CPU Benchmarking
    a. FLOPS and IOPS computations
    b. Sampling FLOPS & IOPS counts on a regular interval for a longer period
2. Memory Benchmarking
3. Disk Benchmarking

### i. Coding Highlights

- Each Benchmarking type or subtype has two functions,
    - a function containing the actual Benchmarking logic and
    - other function to create & govern threads and prints the output on screen. This function calls the actual Benchmarking logic in threads.
- Complete assignment is coded in C and all the code is written in one single file.
- Code is developed based on the type or subtype of Benchmarking
- Time duration is computed using “struct timeval” structure variables
- Except CPU Benchmarking, all the various combinations of different Benchmarking Tests are part of the same function or module
- Various structures are defined and used to store results for various benchmarking experiments

### ii. CPU Benchmarking

CPU Benchmarking experiment is divided into two parts.

1. Computation of FLOPS & IOPS using a fixed number of iterations
2. Sampling FLOPS & IOPS on a regular interval over a longer period

First part is done by doing multiple Floating Point and Integer addition & multiplication operations in a loop. The loop is repeated 1,000,000,000 times. Total time taken to completed the loop is measured in seconds. Formula for Giga FLOPS / IOPS computation is

$$\frac{Total\ operations\ done}{Total\ time\ taken * 1{,}000{,}000{,}000}$$

After finishing all the computations, the required parameters or values (like total operations done, time taken as start & end clock variables, etc.) are stored into structure array.
The first part is repeated using 1, 2 & 4 threads each for FLOPS & IOPS computation.

Second part is done by doing using a similar concept like timer. An additional function governs all 4 threads. In this additional function, it sleeps for the sampling interval and then increments a counter variable. The thread function keeps storing the required parameters or values (like total operations done during sampling interval, total operations done till now, exact time of sampling (considering the noise), etc.) are stored into structure array for each individual thread.
Formula for final FLOPS / IOPS count for each sampling interval is

$$\frac{Total\ operations\ done\ during\ sampling\ interval\ by\ all\ 4\ threads}{Average\ of\ exact\ sampling\ time\ (including\ the\ noise)\ for\ all\ 4\ threads * 1000000000}$$

## iii. Memory Benchmarking

Memory Benchmarking requires Throughput & Latency computations. Throughput and Latency computations are done by doing multiple “memcpy” and “memset” operations respectively. Two character pointers (Source & Destination) are used to transfer the data. A large block (typically 100 MB) of memory is allocated and initialized with random characters.
Value for Total no. of blocks is computed as

$$\frac{Total\ Memory\ (typically\ 100\ MB)}{Block\ Size}$$

Block Sizes to be considered are 1 Byte, 1 KB & 1 MB. Two separate loops are repeated for “memcpy” and “memset” respectively to compute Throughput and Latency respectively.

Formulae for Throughput and Latency computations are

$$Throughput = \frac{Total\ Memory\ transferred}{Total\ time\ taken * 1024 * 1024}$$

$$Latency = \frac{Total\ time\ taken * 1000}{Total\ Memory\ transferred}$$

In case of more than one Thread, the above mentioned formulae changes to

$$Throughput = \frac{Total\ Memory\ transferred\ across\ all\ threads}{Average\ time\ taken\ across\ all\ threads * 1024 * 1024}$$

$$Latency = \frac{Average\ time\ taken\ across\ all\ threads * 1000}{Total\ Memory\ transferred\ across\ all\ threads}$$

Since, “memcpy” functions first read the contents from source and then writes it into the destination, it is assumed to be one single operation.

Similarly, Latency is computed using “memset” function with different Block Sizes. After finishing all the computations, the required parameters or values (like total memory read / copied, time taken as start & end clock variables, Throughput, Latency, Sequential / Random Access, etc.) are stored into structure array.
Memory Benchmarking is done using 1 & 2 threads each for Sequential and Random combinations.

## iv. Disk Benchmarking

Disk Benchmarking is very much similar to Memory Benchmarking. Set of Linux based APIs (read, open, lseek, write, etc.) are used to work on the Disk Files in an optimized manner.
A large block (typically 100 MB) of memory is allocated and initialized with random characters. This variable is then written into file. This newly created file works as source for File Read operations. Content of the same variable are written into Sequential and Random Access files separately.

A separate file is prepared to perform Sequential and Random Read operations. A large file (typically 100 MB) is written or read during file Read and Write operations.

Block Sizes to be considered are 1 Byte, 1 KB & 1 MB. This benchmarking is primarily divided into 4 parts

1. Sequential Read using read function
2. Sequential Write using write function
3. Random Read using lseek and read functions
4. Random write using lseek and write functions

Value for Total no. of blocks is computed as

$$\frac{Total\ Memory\ (typically\ 100\ MB)}{Block\ Size}$$

Block Sizes to be considered are 1 Byte, 1 KB & 1 MB. Two separate loops are repeated to compute Throughput and Latency for Read and Write operations.

Formulae for Throughput and Latency computations are

$$Throughput = \frac{Total\ Memory\ Read\ /\ Written}{Total\ time\ taken * 1024 * 1024}$$

$$Latency = \frac{Total\ time\ taken * 1000}{Total\ Memory\ Read\ /\ Written}$$

In case of more than one Thread, the above mentioned formulae changes to

$$Throughput = \frac{Total\ Memory\ Read\ /\ Written\ across\ all\ threads}{Average\ time\ taken\ across\ all\ threads * 1024 * 1024}$$

$$Latency = \frac{Average\ time\ taken\ across\ all\ threads * 1000}{Total\ Memory\ Read\ /\ Writtenacross\ all\ threads}$$

After finishing each of these 4 computations, the required parameters or values (like total memory read / written, Block size, time taken as start & end clock variables, etc.) are stored into structure array.

Disk Benchmarking is done using 1 & 2 threads each for Sequential and Random Read & Write combinations.

## b. Tradeoffs made

### i. Generic

- No log file generation, only the output can be redirected to a text file
- Program would have been made more flexible & parametric by making minor changes & accepting inputs from users

### ii. CPU

- Due to the complexity in understanding and fine-tuning of LINPACK Benchmarking, program's performance is no match with LINPACK output
- Due to AWS t2.micro instance, unable to benchmark experiment with multicore and many-core architecture
- Experiment can be tested with more number of threads

- Similar to LINPACK, we could have used Linear Equation Solver or some other complex operations for benchmarking

### iii. Memory

- Disabling or manipulating the Cache could have given better results
- Scarce resource availability on internet on Memory Benchmarking makes it difficult to understand it programmatically

### iv. Disk

- During Sequential Read and Write operations, cache is used which is more of a Memory Benchmarking than Disk Benchmarking

## c. Possible Improvements

- If structure and their variables are defined properly then program can be made more flexible in terms of the number of threads, sampling intervals, running duration, etc
- Better and optimized performance can be achieved by understanding LINPACK's internal logic in detail
- Testing the Memory Throughput and Latency by avoiding the cache usage
- More fine-tuning to achieve better and optimized benchmarking process
- Testing the Disk Throughput and Latency by avoiding the cache usage
- Testing the Disk Throughput and Latency by using a big disk file (typically 1.5 or 2 GB)