

ILLINOIS INSTITUTE OF TECHNOLOGY

CLOUD COMPUTING

CS 553

Benchmarking Report

27th March, 2018

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | Introduction | | | 3 |
| 2 | CPU | |  | 4 |
|  | 2.1 | Program Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 4 |
|  | 2.2 | Theoretical performance . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 4 |
|  | 2.3 | LINPACK Performance . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 5 |
|  | 2.4 | Result . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 6 |
|  | 2.5 | Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 6 |
| 3 | Memory | |  | 7 |
|  | 3.1 | Program Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 7 |
|  | 3.2 | Theoretical bandwidth . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 7 |
|  | 3.3 | Stream benchmark . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 8 |
|  | 3.4 | Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  Analysis | | 9  10 |
| 4 | Disk |  |  | 11 |
|  | 4.1 | Program Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 11 |
|  | 4.2 | IOzone . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 11 |
|  | 4.3 | Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 12 |
| 5 | Network | |  | 17 |
|  | 5.1 | Program Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 17 |
|  |  | 5.1.1 | Throughput Measurement . . . . . . . . . . . . . . . . . . . . . . . . . . | 17 |
|  |  | 5.1.2 | Latency Measurement . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 17 |
|  | 5.2 | iperf3 Benchmarks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 17 |
|  | 5.3 | ping Benchmark . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 17 |
|  | 5.4 | Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 18 |
|  |  | 5.4.1 TCP Throughput . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 18 |
|  |  | 5.4.2 UDP Throughput . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 18 |
|  | 5.5 | 5.4.3 | TCP and UDP Latency . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  Analysi…………………….. 20 | 20 |
|  |  |  |  |  |

1. Introduction

In computing, a benchmark is the act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object, normally by running a number of standard tests and trials against it. The term 'benchmark' is also mostly utilized for the purposes of elaborately designed benchmarking programs themselves. Benchmarking is usually associated with assessing performance characteristics of computer hardware, for ex-ample, the floating point operation performance of a CPU. Benchmarks provide a method of comparing the performance of various subsystems across di erent chip/system architectures.

The program code design of CPU, Memory and Disk Benchmark is structured as depicted in the below image:

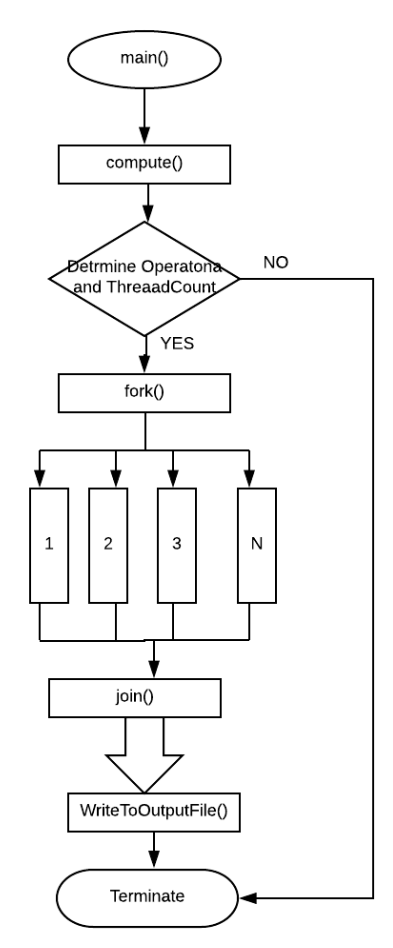


Figure1. C Code Structure

1. CPU

2.1 Program Design

For CPU Benchmarking, a multi-threaded program is created in order to execute several arithmetic operations at a time.

The benchmark programs executes the defined arithmetic operation in the program using a loop. Thread count is used to determine the number of iterations of the loop. Here we are computing the time required by the threads to run the loop successfully.

FLOPS Calculations:

***FLOPS (Double Precision) = Number of operations/ Time required by threads***

2.2 Theoretical performance

The theoretical value for CPU Benchmark are calculated or Hyperion cluster 129.114.33.105.

The command lscpu was used to get the cluster information.

The hardware has 16 socket, 1 core each. The peak clock speed is 2.3 Ghz. Number of CPU is 16. The processor Sandy Bridge Instruction Per cycle(IPC) are given as below.

Dual Precision: 8 DP IPC

Single Precision: 16 SP IPC

Half Precision:32 HP IPC

Quarter Precision: 64 QP IPC

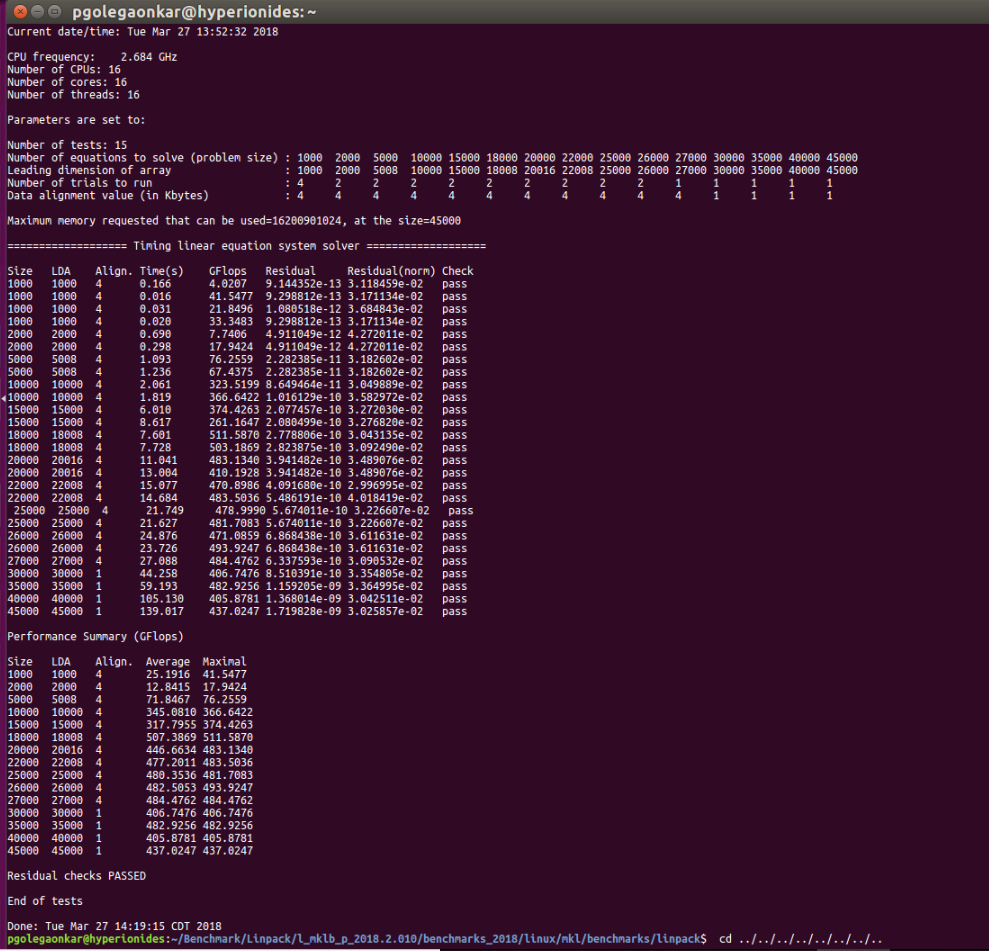
The formula for calculating the theoretical performance of a CPU is:

***FLOPS= (Number of sockets) \* (Number of cores per socket) \* (clock speed) \* (IPC)***

By the aforementioned formula, the result would be: 2380.8 GFlops

2.3 LINPACK Performance

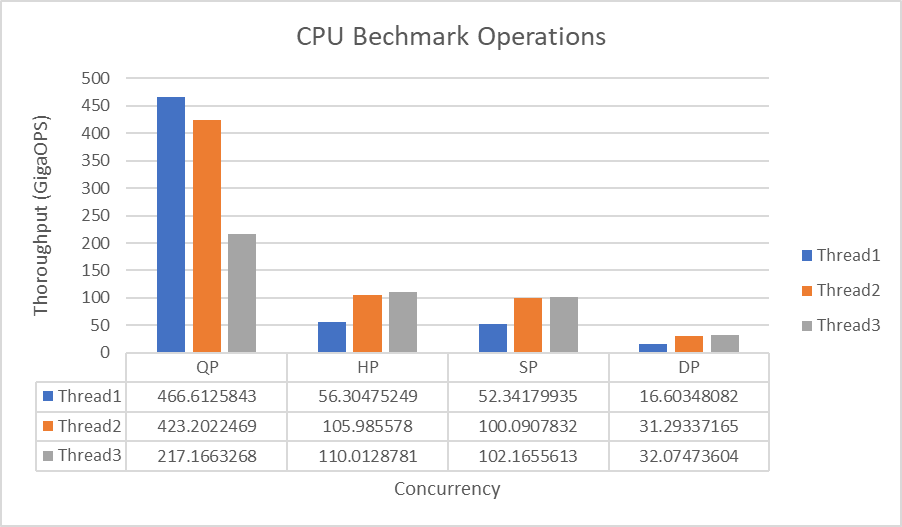
Linpack benchmark was run on the with 48 Threads and and a peak performance of **437.024**7 GFlops was achieved for a problem size of **N=45000**. The key screen shots are below. The detailed output is available in linpack output.txt.



2.4 Result

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Workload** | **Concurrency** | **MyCPUBench** | **HPL** | **Theoretical** | **MyCPUBench** | **HPL** |
|  |  | **Measured** | **Measured** | **Ops/Sec** | **Efficiency (%)** | **Efficiency** |
|  |  | **Ops/Sec** | **Ops/Sec** | **(GigaOPS)** |  | **(%)** |
|  |  | **(GigaOPS)** | **(GigaOPS)** |  |  |  |
| QP | 1 | 466.6125843 | N/A | 2355.2 | 19.8120153 | N/A |
| QP | 2 | 423.2022469 | N/A | 2355.2 | 17.9688454 | N/A |
| QP | 4 | 217.1663268 | N/A | 2355.2 | 9.220717 | N/A |
| HP | 1 | 56.30475249 | N/A | 1177.6 | 4.7813139 | N/A |
| HP | 2 | 105.985578 | N/A | 1177.6 | 9.000134 | N/A |
| HP | 4 | 110.0128781 | N/A | 1177.6 | 9.3421262 | N/A |
| SP | 1 | 52.34179935 | N/A | 588.8 | 8.8895719 | N/A |
| SP | 2 | 100.0907832 | N/A | 588.8 | 16.999114 | N/A |
| SP | 4 | 102.1655613 | N/A | 588.8 | 17.351488 | N/A |
| DP | 1 | 16.60348082 | 234.28 | 294.4 | 5.6397693 | 79.57880435 |
| DP | 2 | 31.29337165 | 257.12 | 294.4 | 10.629542 | 87.33695652 |
| DP | 4 | 32.07473604 | 268.064 | 294.4 | 10.8949511 | 91.05434783 |

Table 2.1 CPU Throughput (measured in GigaOps)



2.5 Analysis

Compared to theoretical, our code achieves approximately 25% efficiency. Compared to theoretical, Linpack achieves 196.46% efficiency

1. Memory

In this program we are measuring the efficiency and throughput of the memory with a help of a simple CPU program consisting of the three basic operation.

1. Sequential read and write
2. Sequential write
3. Random write

Operation are tested on concurrency of one, two and four threads and on four different block sizes ie 1 Byte, 1 Kbytes, 10 Megabytes and 100 Megabytes.

3.1 Program Design

We are calculating the monotonic time taken for the operations to execute. memcpy() is used to read from the source and store in the destination. Source and Destination array for memcpy are set to the fixed size of the memory.

For Sequential Write, the destination memory is access sequentially.

For Random Write, the destination memory is accessed randomly with rand() function.

3.2 Theoretical bandwidth

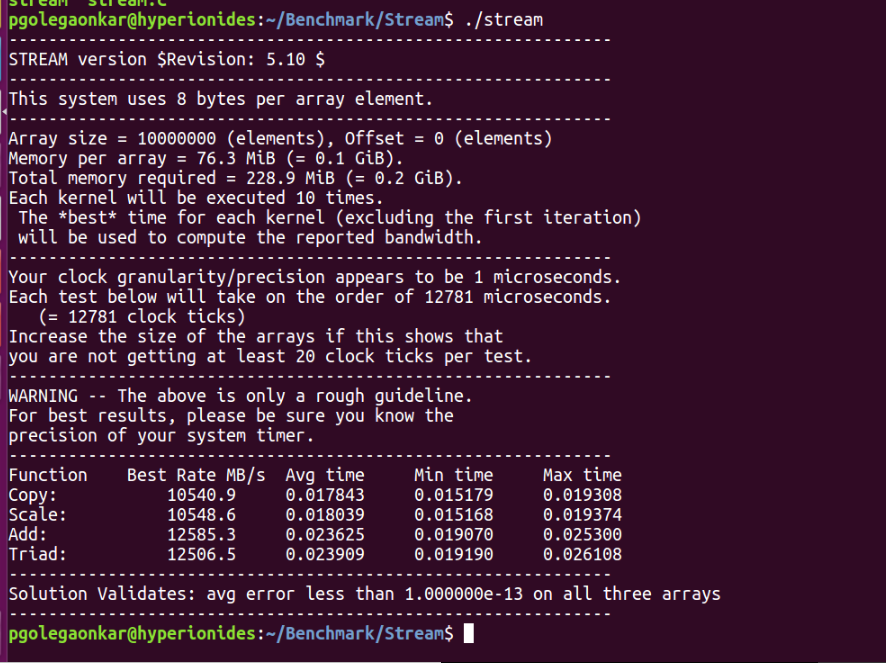
The formula for calculation of the theoretical bandwidth of memory in MBps is as follows:

***(Memory clock \* bus Width/8) \* DDR type multiplier***

For the given cluster the memory id DDR4 with clock speed of 2133 MHz and bus width of 64 bits. The multiplier is 2. Hence the bandwidth would be (2133\*64/8) \* 2=34.1 GB/s

3.3 Stream benchmark

I have used stream benchmark for this benchmark. The obtained result from the benchmarking tool provides us a maximum bandwidth of **10540.9 MBps** as shown in the screenshot below. The result can also be read by accessing stream result.txt in the submission folder.



3.4 Results

The results are calculated on Hyperion cluster.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Work**  **-load** | **Concyrreny** | **Block Size** | **MyRAMBench Measured Throughput**  **(GB/sec)** | **pmbw Measured Throughput**  **(GB/sec)** | **Theoretical Throughput (GB/sec)** | **MyRAMBench Efficiency (%)** | **pmbw Efficiency (%)** |
| RWS | 1 | 1KB | 36.103810 | 14 | 68.25 | 52.894705 | 20.5110173 |
| RWS | 1 | 1MB | 41.638692 | 17.12 | 68.25 | 41.638692 | 25.0820441 |
| RWS | 1 | 10MB | 74.735427 | 15.28 | 68.25 | 109.492832 | 22.3863104 |
| RWS | 2 | 1KB | 42.192252 | 20.16 | 68.25 | 61.814715 | 29.535865 |
| RWS | 2 | 1MB | 37.677628 | 11.52 | 68.25 | 55.200463 | 16.8776371 |
| RWS | 2 | 10MB | 66.641076 | 28.8 | 68.25 | 97.634020 | 42.1940928 |
| RWS | 4 | 1KB | 41.605666 | 19.28 | 68.25 | 60.955324 | 28.246601 |
| RWS | 4 | 1MB | 39.874447 | 19.28 | 68.25 | 58.418962 | 46.1790905 |
| RWS | 4 | 10MB | 57.892227 | 25.76 | 68.25 | 84.816320 | 37.7402719 |
| RWR | 1 | 1KB | 240.848596 | 3.12 | 68.25 | 352.860695 | 4.57102672 |
| RWR | 1 | 1MB | 136.063789 | 0.488 | 68.25 | 199.343338 | 0.71495546 |
| RWR | 1 | 10MB | 94.928943 | 0.392 | 68.25 | 139.077800 | 0.57430849 |
| RWR | 2 | 1KB | 26.494744 | 7.12 | 68.25 | 38.816725 | 10.4313174 |
| RWR | 2 | 1MB | 71.889248 | 0.992 | 68.25 | 105.322972 | 1.45335209 |
| RWR | 2 | 10MB | 83.354221 | 0.68 | 68.25 | 122.119990 | 0.99624941 |
| RWR | 4 | 1KB | 38.816725 | 15.52 | 68.25 | 38.343192 | 22.7379278 |
| RWR | 4 | 1MB | 67.042300 | 2.408 | 68.25 | 98.221842 | 3.52789498 |
| RWR | 4 | 10MB | 109.048471 | 0.728 | 68.25 | 122.119990 | 1.0665729 |

Table 3.1 Memory Throughput

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Work**  **-load** | **Con- currency** | **Block Size** | **MyRAMBench Measured Latency (us)** | **pmbw Measured Latency**  **(us)** | **Theoretical Latency (us)** | **MyRAMBench Efficiency (%)** | **pmbw Efficiency (%)** |
| RWS | 1 | 1B | 0.000052 | 0.021 | 0.014 | 0.369983 | 150 |
| RWS | 2 | 1B | 0.000060 | 0.0259 | 0.014 | 0.426071 | 185 |
| RWS | 4 | 1B | 0.000056 | 0.0273 | 0.014 | 0.399270 | 195 |
| RWR | 1 | 1B | 0.000269 | 0.133 | 0.014 | 1.924428 | 950 |
| RWR | 2 | 1B | 0.005539 | 0.245 | 0.014 | 39.564591 | 1750 |
| RWR | 4 | 1B | 0.005277 | 0.336 | 0.014 | 37.695960 | 2400 |

Table 3.2 Latency Table of Memory Benchmark

* 1. Analysis

From throughput we can conclude that sequential read and write have better performance compared to Random Read and write

RWS gives us les latency compared to RWR. Also this latency is less compared to the theoretical and pmbw values. Number of threads increases the latency for RWR operations because of random selection location for reading and writing data.

1. Disk

In this program we are measuring the latency and throughput of the memory with a help of a simple CPU program consisting of the three basic operation .

1. Sequential read
2. Sequential write
3. Random Read
4. Random Write

Operation are tested for latency on concurrency of 1, 2, 3, 4, 8, 16 and 32 threads for block sizes 1KiloBytes on the space allocated for the input files of 1GB

Operation are tested for throughput on concurrency of one, two and four threads and on four different block sizes ie 1 Megabytes, 10 Megabytes and 100 Megabytes on the space allocated for the input files of 10GB

The theoretical performance of the provided disk, as advertised by the manufacturer is 6Gbps. It is also important to note that the standard benchmarking tool used in this case is called IOzone, which, like Stream, is freely available on the Internet and can be used to measure the performance/speed of the computer's disk.

4.1 Program Design

We are calculating the monotonic time taken for the operations to execute on file of Size of 10GB. This 10GB file is created by filling in dummy data with the help of a char array.

For write operation, fwrite() is used to read from the char array and write into an file. We have used “w+” so that different thread writing to file can append the data to the file to create a 10 GB size.

For read operation, fread() is used to used the read the file.

For Random operation, rand() is used to access the random portions of the file and seek() to set the the stream in order to be used with fread().

4.2 IoZone

The iozone results were evaluated on the Prometheus. The following command was exceuted

./iozone -s 10G -r 1M -t 1 -I 0 -f //tmp/file.txt

4.3 Results

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Work-load | Concureny | Block Size | MyDiskBench Thoroughput  (MB/sec) | IOZone  Thoroughput  (MB/sec) | Theoretical  Throughput  (MB/sec) | MyDiskBench  Efficiency (%) | IOZone  Efficiency (%) |
| RS | 1 | 1MB | 293.55264 | 300 | 372 | 78.912 | 80.6451613 |
| RS | 1 | 10MB | 218.18916 | 300 | 372 | 58.653 | 80.6451613 |
| RS | 1 | 100MB | 212.598 | 300 | 372 | 57.15 | 80.6451613 |
| RS | 2 | 1MB | 484.18776 | 319.2 | 744 | 65.079 | 42.9032258 |
| RS | 2 | 10MB | 528.04656 | 331.2 | 744 | 70.974 | 44.516129 |
| RS | 2 | 100MB | 535.41216 | 355.2 | 744 | 71.964 | 47.7419355 |
| RS | 4 | 1MB | 890.03232 | 331.2 | 1488 | 59.814 | 22.2580645 |
| RS | 4 | 10MB | 918.82512 | 348 | 1488 | 61.749 | 23.3870968 |
| RS | 4 | 100MB | 944.67168 | 360 | 1488 | 63.486 | 24.1935484 |
| WS | 1 | 1MB | 104.353776 | 306 | 172 | 60.6708 | 177.906977 |
| WS | 1 | 10MB | 136.3014 | 300 | 172 | 79.245 | 174.418605 |
| WS | 1 | 100MB | 140.38812 | 276 | 172 | 81.621 | 160.465116 |
| WS | 2 | 1MB | 171.3636 | 358.8 | 344 | 49.815 | 104.302326 |
| WS | 2 | 10MB | 269.69256 | 378 | 344 | 78.399 | 109.883721 |
| WS | 2 | 100MB | 290.80728 | 288 | 344 | 84.537 | 83.7209302 |
| WS | 4 | 1MB | 285.01776 | 280 | 688 | 41.427 | 40.6976744 |
| WS | 4 | 10MB | 144.08784 | 296 | 688 | 20.943 | 43.0232558 |
| WS | 4 | 100MB | 140.18688 | 288 | 688 | 20.376 | 41.8604651 |
| RR | 1 | 1MB | 286.74 | 108 | 540 | 53.1 | 20 |
| RR | 1 | 10MB | 279.3042 | 118.4 | 540 | 51.723 | 21.9259259 |
| RR | 1 | 100MB | 280.908 | 116.8 | 540 | 52.02 | 21.6296296 |
| RR | 2 | 1MB | 451.5912 | 127.2 | 1080 | 41.814 | 11.7777778 |
| RR | 2 | 10MB | 490.5684 | 135.2 | 1080 | 45.423 | 12.5185185 |
| RR | 2 | 100MB | 550.8324 | 136 | 1080 | 51.003 | 12.5925926 |
| RR | 4 | 1MB | 1043.928 | 128 | 2160 | 48.33 | 5.92592593 |
| RR | 4 | 10MB | 1088.8344 | 129.6 | 2160 | 50.409 | 6 |
| RR | 4 | 100MB | 992.8008 | 132 | 2160 | 45.963 | 6.11111111 |
| WR | 1 | 1MB | 314.9784 | 192 | 410 | 76.824 | 46.8292683 |
| WR | 1 | 10MB | 144.2052 | 193.6 | 410 | 35.172 | 47.2195122 |
| WR | 1 | 100MB | 144.1314 | 198.4 | 410 | 35.154 | 48.3902439 |
| WR | 2 | 1MB | 189.3708 | 224 | 820 | 23.094 | 27.3170732 |
| WR | 2 | 10MB | 300.366 | 226.4 | 820 | 36.63 | 27.6097561 |
| WR | 2 | 100MB | 181.4004 | 244 | 820 | 22.122 | 29.7560976 |
| WR | 4 | 1MB | 154.6848 | 160.8 | 1640 | 9.432 | 9.80487805 |
| WR | 4 | 10MB | 300.9564 | 168 | 1640 | 18.351 | 10.2439024 |
| WR | 4 | 100MB | 134.0208 | 168.8 | 1640 | 8.172 | 10.2926829 |

Table 4.1 Disk Throughput (measured in MB/sec)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Work** | **Con-** | **Block** | **MyDiskBench** | **IOZone** | **Theoretical** | **MyDiskBench** | **IOZone** |
|  |  |  | **Measured** | **Measured** | **Latency** | **Efficiency (%)** | **Efficiency** |
|  |  |  | **Latency (ms)** | **Latency** | **(ms)** |  | **(%)** |
|  |  |  |  | **(ms)** |  |  |  |
| RR | 1 | 1KB | 0.00078 | 1.56 | 0.5 | 0.156 | 312 |
| RR | 2 | 1KB | 0.0006 | 1.56 | 0.5 | 0.12 | 312 |
| RR | 4 | 1KB | 0.00078 | 2.4 | 0.5 | 0.156 | 480 |
| RR | 8 | 1KB | 0.00108 | 1.44 | 0.5 | 0.216 | 288 |
| RR | 16 | 1KB | 0.0012 | 0.72 | 0.5 | 0.24 | 144 |
| RR | 32 | 1KB | 0.00168 | 1.32 | 0.5 | 0.336 | 264 |
| RR | 64 | 1KB | 0.00216 | 1.32 | 0.5 | 0.432 | 264 |
| RR | 128 | 1KB | 0.0018 | 0.672 | 0.5 | 0.36 | 134.4 |
| WR | 1 | 1KB | 0.00234 | 2.28 | 0.5 | 0.468 | 456 |
| WR | 2 | 1KB | 0.00192 | 2.28 | 0.5 | 0.384 | 456 |
| WR | 4 | 1KB | 0.072 | 3.72 | 0.5 | 14.4 | 744 |
| WR | 8 | 1KB | 0.114 | 0.9 | 0.5 | 22.8 | 180 |
| WR | 16 | 1KB | 0.156 | 0.972 | 0.5 | 31.2 | 194.4 |
| WR | 32 | 1KB | 0.54 | 0.6 | 0.5 | 108 | 120 |
| WR | 64 | 1KB | 0.00252 | 0.66 | 0.5 | 0.504 | 132 |
| WR | 128 | 1KB | 0.00222 | 0.804 | 0.5 | 0.444 | 160.8 |

Table 4.2 Disk Latency(measured in ms)

4.4 Analysis

Number of threads increases the latency and throughput. Also block size increases the throughput because the increase in the block sizes decreases the number of disk read and write operations. For Sequential Read and Random read the throughput is increasing whereas for Sequential Write and Random Write its varies

* Network

5.1 Program Design

The developed code is working efficiently for 1, 2 and 4 thread. The results are evaluated only for 1, 2 and 4 threads. For testing purpose, the client and server are run on the same machine. I have used Hyperionides for Testing.

5.1.1 Throughput Measurement

The following is the setup for TCP throughput measurement.

At the server when there is a new incoming connection, after the initial setup (3-way Hand-shake), the server launches a new thread to handle the incoming request. The parent thread keeps waiting for new incoming connection while the child thread starts sending 8GBN (Strong scaling) data to the client.

The child threads in the client receives receives 8GBN of data, discard it and die. The parent thread then waits for all the children to complete and stops the timer. Computes the difference between end time and start time. Let the di erence be T .

hroughput = 8 8 Gigabits Gbps

T seconds

The experimental setup for UDP is the same with the following exceptions.

5.1.2 Latency Measurement

As described in the previous section we use one thread in server to handle one thread in client. For measuring TCP the experimental setup remains the same as above, but now server restart

5.2 iperf3 Benchmarks

With client, server on the local machine and TCP the throughput measured with iperf3 is 28.3 GBits/sec

With client, server on the local machine and UDP the throughput measured with iperf3 is 12.3 Gbits/sec

The results are available in the Benchmark/iperf3 folder for inspection.

5.3 ping Benchmark

The latency measured on the local loopback is 0.023 ms. The results are available in the network Benchmark/ping folder.

5.4 Results

5.4.1 TCP Throughput

The following are the results obtained when client and server are on the same machine. The means of communication is a TCP socket. Since the machine on which the experiment had 16 cores, we see that strong scaling improves the throughput. This is actually true because each core can now parallel y send and receive data.

5.4.2 UDP Throughput

With client and server on the local loopback, UDP as means of the communication the following results are obtained.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Proto- col** | **Concurrency** | **Block Size** | **MyNETBench**  **(GBits/sec)** | **iperf3**  **(GBits/sec)** | **Efficiency(%)** |
| TCP | 1 | 1000 | 13.769764 | 28.2 | 48.8289504 |
| TCP | 2 | 1000 | 12.986831 | 28.2 | 46.0525922 |
| TCP | 4 | 1000 | 13.097273 | 28.2 | 46.4442305 |
| TCP | 1 | 32000 | 128.886118 | 28.2 | 457.042972 |
| TCP | 2 | 32000 | 124.949474 | 28.2 | 443.083241 |
| TCP | 4 | 32000 | 122.590954 | 28.2 | 434.719695 |
| UDP | 1 | 1000 | 2.83746 | 12.3 | 23.0687805 |
| UDP | 2 | 1000 | 2.423992 | 12.3 | 19.707252 |
| UDP | 4 | 1000 | 2.353443 | 12.3 | 19.1336829 |
| UDP | 1 | 32000 | 56.921753 | 12.3 | 462.77848 |
| UDP | 2 | 32000 | 57.576391 | 12.3 | 468.10074 |
| UDP | 4 | 32000 | 48.662105 | 12.3 | 395.62687 |

Table 5.1 Network Throughput

5.4.3 TCP and UDP Latency

With client and server on the same machine and TCP socket, the latency results are tabulated below. The latency increases with threads. I suspect the reason to be as too little work done in each thread is masked by overhead of creating, maintaining the threads.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Protocol | Concurrency | Message Size | Latency | ping latency | measured Efficiency |
| TCP | 1 | 1B | 0.000599 | 0.023 | 2.60434783 |
| TCP | 2 | 1B | 0.000698 | 0.023 | 3.03478261 |
| TCP | 4 | 1B | 0.000781 | 0.023 | 3.39565217 |
| UDP | 1 | 1B | 0.002981 | 0.023 | 12.9608696 |
| UDP | 2 | 1B | 0.003095 | 0.023 | 13.4565217 |
| UDP | 4 | 1B | 0.002811 | 0.023 | 12.2217391 |

Table 5.2 Network Latency

5.5 Analysis

The throughput increases drastically for the increase in the number of thread.

The Throughput for TCP will always be greater than the UDP which can be seen from the reading as well the graph. UDP property satisfies correctly in the designed code. Latency increases for TCP protocol as the thread count increases. But for UDP due to it property the latency varies as few of the datagram may get lost while transmission.