**Cloud Computing Programming Assignment 1**

**Performance Evaluation**

This document contains the experimental results for the following benchmark

For CPU benchmark, we measure the performance by calculating processor speed for performing instructions per cycle

For Memory benchmark, we measure the throughputs (sequential read, sequential write, random read and random write) and latency (random read, random write) for various block size (1B, 1KB, 1MB) and number of threads (1 and 2).

For Disk benchmark, we measured the throughputs (sequential read, sequential write, random read and random write) and latency (random read, random write) for various block size (1B, 1KB, 1MB) and number of threads (1 and 2).

**1. System Domain**

For all benchmark, the experiments were performed on Linux- Ubuntu, Intel i3 processor DDR3 with 2 cores, RAM of 2GB, and DISK 20GB.

**2. Program Output Graphs and Results**

This part shows the experiment result for each benchmark and give explanations for the trends in the results.

**A CPU benchmark Results**

CPU benchmark result is to find maximum GIOPS and GFLOPS performed by processor for different number of threads.

Below graph shows the experiment results for 1, 2 and 4 threads for GFlops and GIops.

GFlops obtained are between 0.0005 to 0.01 GFLOPS and GIOPS between 0.1 to 0.4 as we vary the no of threads

Thus we derive that as we increase the thread size the No of instructions per second decreases

**Theoretical peak performance of the processor is given by:**

**= No of Cores \*instruction per cycle \* clock rate in Ghz**

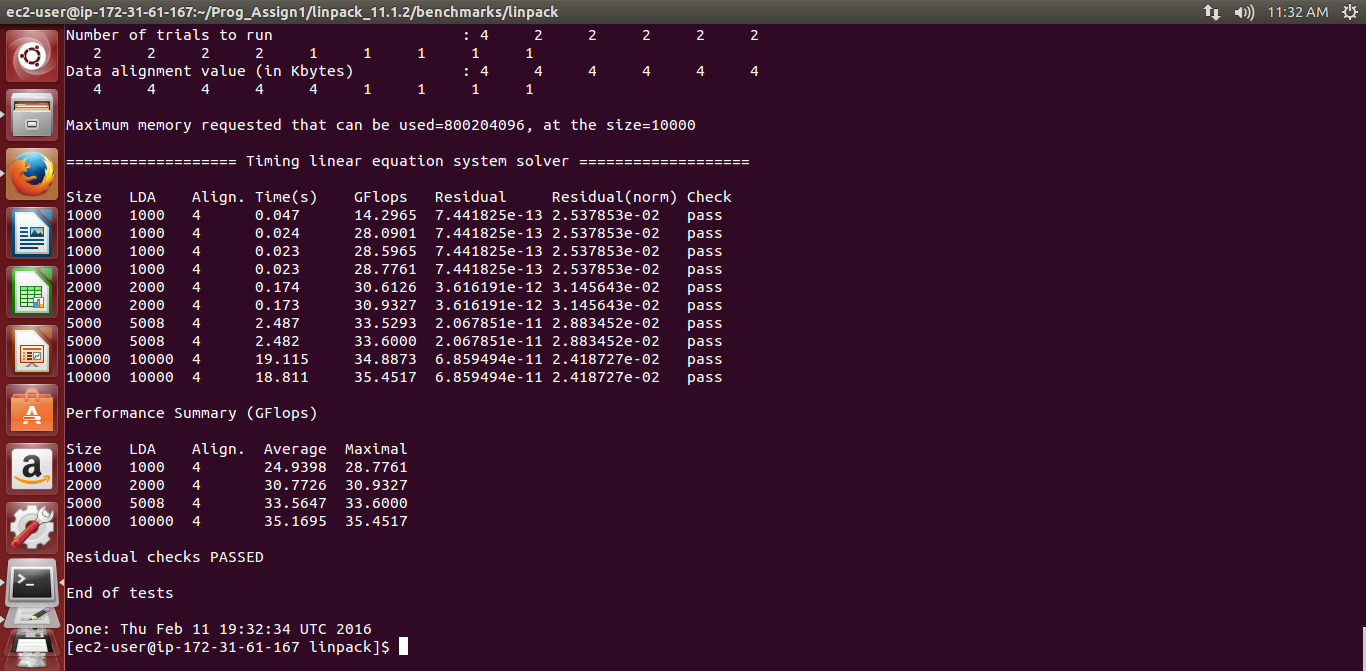
**=1\*4\*2.5=10 Giga Flops per second**

**Maximum Practical Performance we got from our output= 0.4**

**Efficiency= (0.4/10)\*100= 4%**

**4% efficiency achieved as compared to theoretical performance.**

**Linpack Benchmark Results**



From code output practical performance we got 0.4 GFLOPS per Second

Practical Value we get from linpack

= 35.4723/18.8=1.886

Efficiency achieved on comparison to theoretical performance

=1.886/10=18%

2) Sample of Graph for 10 minutes plotting Instructions per second based on 4 thread

As the time increase Flops and IOPS keep on fluctuating in a certain range. Performance drops in certain interval value and reaches back to original value

This occur due to time lapse in synchronization and CPU processing

THREAD /TIME

**2) Disk benchmark**

A Throughput V/s size

The graph displays the implementation of sequential and random throughput for disk read and write.

As the size increases throughput increases exponentially for random and sequential read and write of disk

As the bytes increase reading randomly gives more throughput megabytes read per second than random read

Random writes are more efficient than sequential writes for throughput

Read is performed faster than write.

Read throughput is more than write throughput

B Latency V/s Size

Graph shows the implementation and comparison of number of threads in sequential read, sequential write, random read and random write with respect to latency in milliseconds.

Sequential Latency takes more time than random latency to reach the disk for both reading and writing. Hence, we can say that sequential access is less efficient.

Write takes more time than read to reach disk.

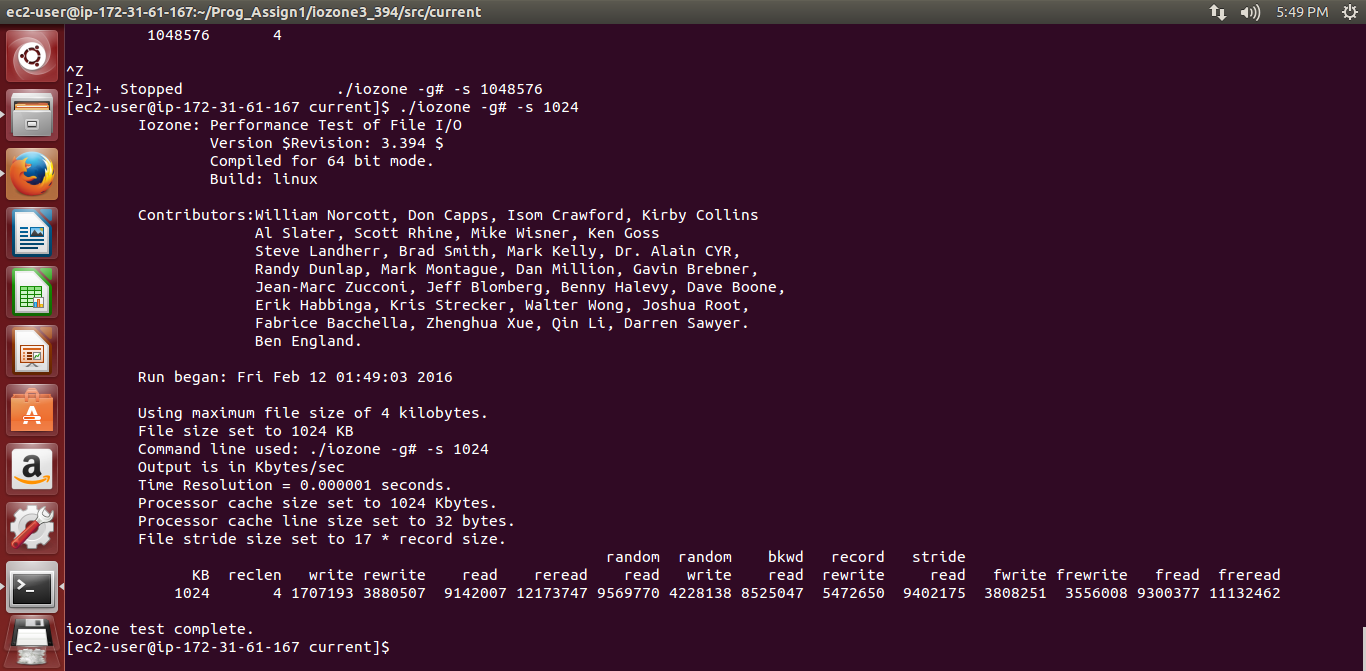
Therefore random access is more efficient with respect to latency and throughput.

Throughput on increasing no of threads for 1 MB

Latency on increasing no of threads for 1 MB

C IOZONE Benchmark

Below screen shot shows the execution of Iozone benchmark for 1024 kb (1 MB)



Theoretical Performance of disk by Iozone:

Read Throughput 9300377 kb/sec =9082.39 MB/sec

Write Throughput 3808251/1024 =3718.99 MB/Sec

Practical output maximum throughput achieved read sequential= 7095.97

Practical output maximum throughput achieved write sequential=3619.9

Efficiency achieved for read (%) = 7095.97 \*100/9082.39= 78.12 %

Efficiency achieved for write (%) = 3619.9 \*100/3718.99= 97.33 %

Theoretical Throughput for processor

Transfer rate=6 Gb/sec

Cache Size=16

Revolution per minute =5400

Latency Achieved in program output= 0.53

Efficiency Achieved= 0.53/7.3=7%

Theoretical Throughput for sequential write=80.8 mb/sec

D Program Output Values

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| size | threads | Access | throughput | latency | operation |
| 1 | 1 | sequential | 4.465004 | 0.000214 | Read |
| 1 | 2 | sequential | 4.360963 | 0.000219 | Read |
| 1 | 1 | random | 3.210147 | 0.000297 | Read |
| 1 | 2 | random | 3.067479 | 0.000311 | Read |
| 1024 | 1 | sequential | 3495.892 | 0.000279 | Read |
| 1024 | 2 | sequential | 3478.261 | 0.000281 | Read |
| 1024 | 1 | random | 3766.478 | 0.000259 | Read |
| 1024 | 2 | random | 3612.39 | 0.00027 | Read |
| 1048576 | 1 | sequential | 5774.505 | 0.173175 | Read |
| 1048576 | 2 | sequential | 6142.742 | 0.162794 | Read |
| 1048576 | 1 | random | 8088.979 | 0.123625 | Read |
| 1048576 | 2 | random | 8802.333 | 0.113606 | Read |
| 1 | 1 | sequential | 1.594541 | 0.000598 | Write |
| 1 | 2 | sequential | 1.557635 | 0.000612 | Write |
| 1 | 1 | random | 1.319972 | 0.000722 | Write |
| 1 | 2 | random | 1.304455 | 0.000731 | Write |
| 1024 | 1 | sequential | 1092.344 | 0.000894 | Write |
| 1024 | 2 | sequential | 1115.084 | 0.000876 | Write |
| 1024 | 1 | random | 1353.592 | 0.000721 | Write |
| 1024 | 2 | random | 1347.232 | 0.000725 | Write |
| 1048576 | 1 | sequential | 3619.909 | 0.27625 | Write |
| 1048576 | 2 | sequential | 3926.573 | 0.254675 | Write |
| 1048576 | 1 | random | 7095.973 | 0.140925 | Write |
| 1048576 | 2 | random | 7234.254 | 0.138231 | Write |

1. **Memory benchmark results**

A Throughput V/s Size

As we increase the memory size Sequential memory access gives more throughput. Random access is less efficient for copying bytes of data than sequential as we increase the memory size

B Latency V/s Size

As we increase the size of memory latency for accessing the memory increases exponentially.

s

Latency increases exponentially for both random and sequential access as we increase the memory bytes to copy

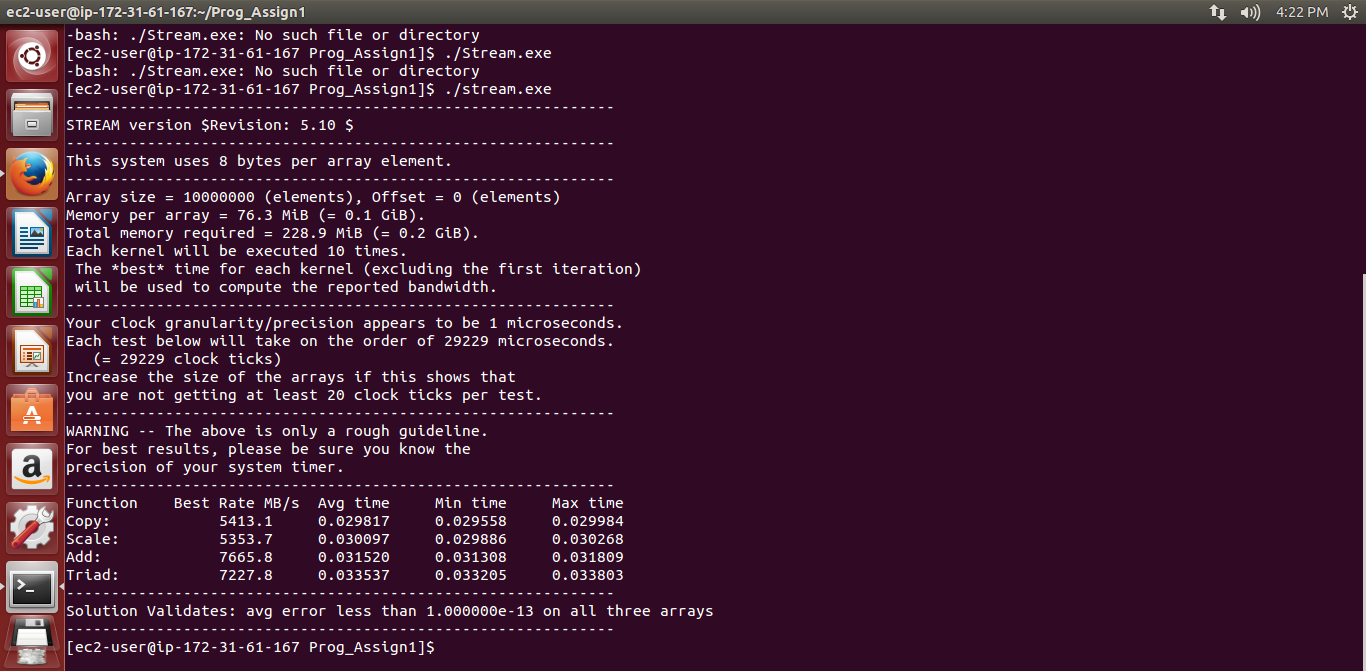
The graph shows the comparison between sequential and random access throughput and latency for multiple threads.

As we increase threads throughput for accessing the memory increases and latency decreases

Throughput on increasing no of threads for 1 MB

Latency on increasing no of threads for 1 MB

C Stream Benchmark



Average Latency for Stream=0.029

Average Latency Achieved from Program=0.034

Efficiency achieved= (0.034/0.029)\*100

Clocks =33

Theoretical memory throughput = 8 \*clock\_rate=8\*33=264 mb/sec

D Program Output Values

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| size | threads | access | throughput | latency |
| 1 | 1 | sequential | 330.6147 | 0.000003 |
| 1 | 2 | sequential | 332.9622 | 0.000003 |
| 1 | 1 | random | 24.4916 | 0.000039 |
| 1 | 2 | random | 20.4465 | 0.000047 |
| 1024 | 1 | sequential | 52390.31 | 0.000019 |
| 1024 | 2 | sequential | 56677.29 | 0.000017 |
| 1024 | 1 | random | 24949.32 | 0.000039 |
| 1024 | 2 | random | 24756.3 | 0.000039 |
| 1048576 | 1 | sequential | 26233.81 | 0.038119 |
| 1048576 | 2 | sequential | 27753.69 | 0.036031 |
| 1048576 | 1 | Random | 25898.35 | 0.038612 |
| 1048576 | 2 | Random | 28782.15 | 0.034744 |