Benchmark different parts of a computer system

Veereshwaran Rangasamy Chettiar Ramamoorthi

vrangasa@hawk.iit.edu

Performance Evaluation

1. Introduction

Benchmark experiments were executed on Amazon EC2 t2.micro instance and the results are recorded and plotted in a graph to showcase the results.

2. Benchmark Experiments

2.1 CPU Benchmarking:

Theoretical calculation:

- # of cores* Instruction Per Cycle * Speed in GHz
- Amazon AWS t2.micro instance has **10 GFlops**

Benchmark Result:

- With the benchmark experiments, an average of **4.44 GFLOPS** is benchmarked, which is 43% efficient.
- With the benchmark experiments, an average of **7.06 GIOPS** is benchmarked.

LINPACK Result:

- LINPACK resulted in an average of **19.75 GFlops**

Experiment

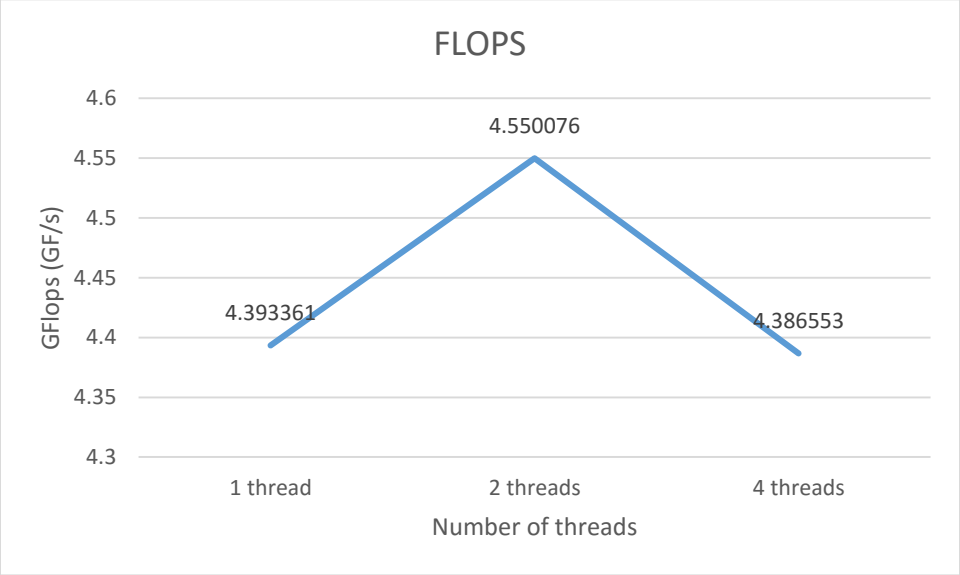
In this benchmarking, 8 experiments are being implemented and run on Amazon EC2 t2.micro instance.

Parameter Space:

Threads	1, 2, 4
Operation Types	Integer Operation, Floating point Operation

Threads	4
Operation Types	Integer Operation, Floating point Operation
Time period	600 seconds

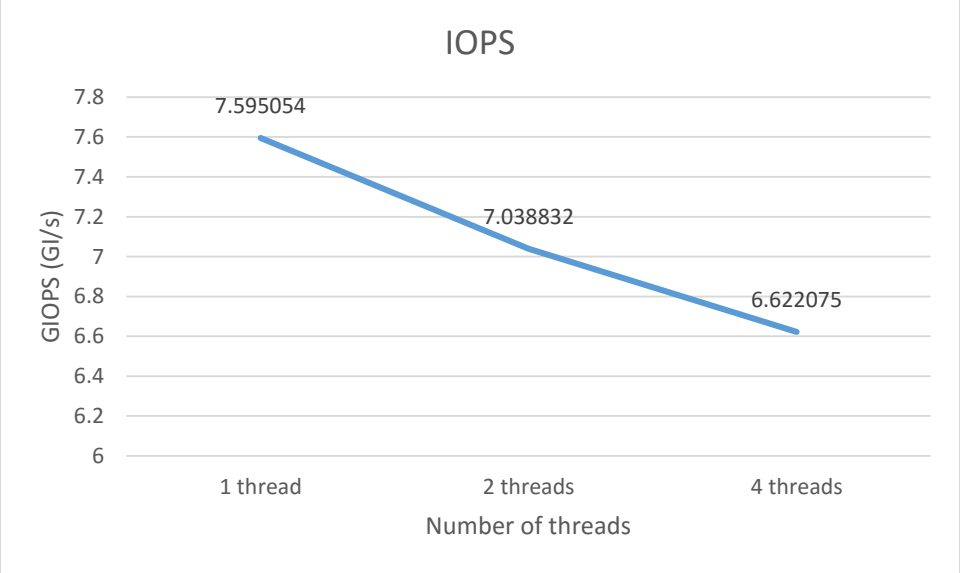
Graphs and Plots



Graph 1: GFLOPS

Graph 1 depicts the FLOPs benchmarked on running the benchmark experiment with 1 thread, 2 threads, and 4 threads.

On an average, a **4.31 GFLOPS** is achieved.



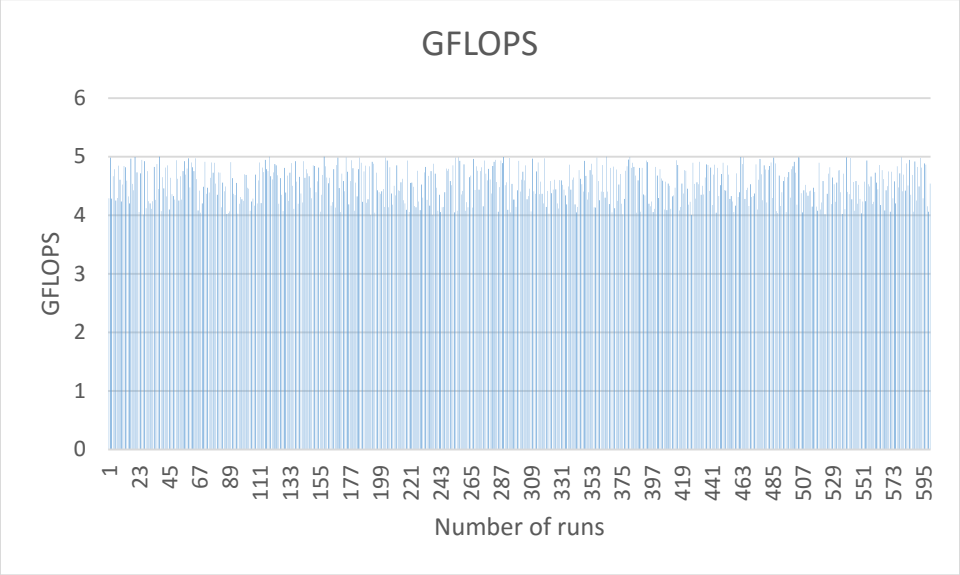
Graph 2: GIOPS

Graph 2 depicts the IOPs benchmarked on running the benchmark experiment with 1 thread, 2 threads, and 4 threads.

On an average, a **7.06 GIOPS** is achieved.

Four threads are spawned for 10 minutes to calculate the GFLOPS by each thread and then their values are recorded each second and it is being represented in the following graph.

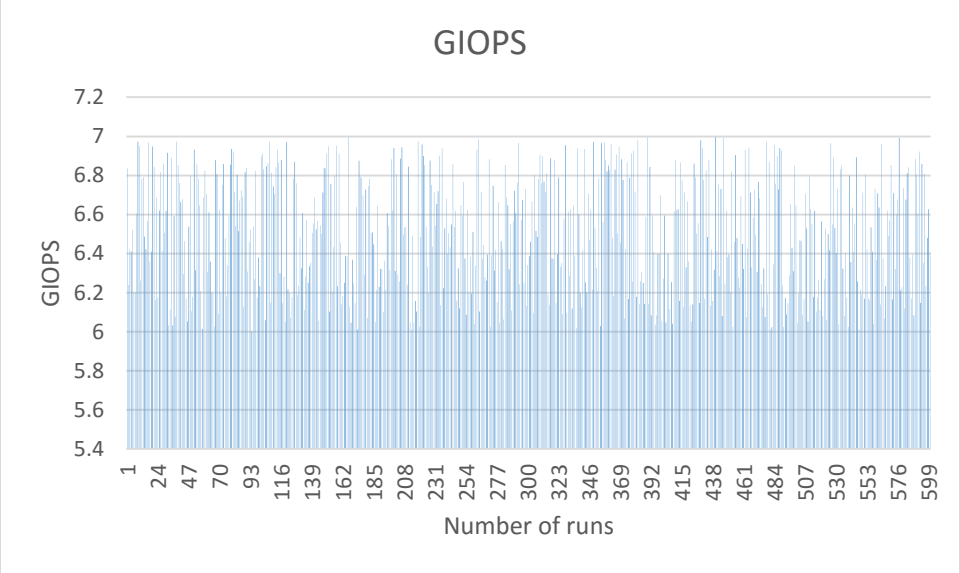
The value represents the sum of GFLOPs of all four threads.



Graph 1.a: GFLOPS of four threads running for 600 seconds

Four threads are spawned for 10 minutes to calculate the GIOPS by each thread and then their values are recorded each second and it is being represented in the following graph.

The value represents the sum of GIOPs of all four threads.



Graph 2.a: GIOPS of four threads running for 600 seconds

LINPACK:

```
This is a SAMPLE run script for SMP LINPACK. Change it to reflect
the correct number of CPUs/threads, problem input files, etc..
Fri Feb 12 05:21:10 UTC 2016
Intel(R) Optimized LINPACK Benchmark data

Current date/time: Fri Feb 12 05:21:10 2016

CPU frequency:    2.945 GHz
Number of CPUs: 1
Number of cores: 1
Number of threads: 1

Parameters are set to:

Number of tests: 15
Number of equations to solve (problem size) : 1000  2000  5000  10000 15000 18000 20000
22000 25000 26000 27000 30000 35000 40000 45000
Leading dimension of array             : 1000  2000  5008  10000 15000 18008 20016 22008
25000 26000 27000 30000 35000 40000 45000
Number of trials to run                : 4    2    2    2    2    2    2    2    2    2    1    1    1    1    1
Data alignment value (in Kbytes)       : 4    4    4    4    4    4    4    4    4    4    4    1    1
1    1

Maximum memory requested that can be used=800204096, at the size=10000

================== Timing linear equation system solver ==================

Size   LDA    Align. Time(s)    GFlops   Residual     Residual(norm) Check
1000   1000   4      0.054      12.4500  9.900691e-13 3.376390e-02   pass
1000   1000   4      0.036      18.5216  9.900691e-13 3.376390e-02   pass
1000   1000   4      0.035      18.9241  9.900691e-13 3.376390e-02   pass
1000   1000   4      0.037      17.9251  9.900691e-13 3.376390e-02   pass
2000   2000   4      0.273      19.5721  4.053480e-12 3.526031e-02   pass
2000   2000   4      0.267      19.9920  4.053480e-12 3.526031e-02   pass
5000   5008   4      4.004      20.8226  2.336047e-11 3.257429e-02   pass
5000   5008   4      3.954      21.0879  2.336047e-11 3.257429e-02   pass
10000  10000  4      30.800     21.6512  1.124127e-10 3.963786e-02   pass
10000  10000  4      31.713     21.0284  1.124127e-10 3.963786e-02   pass

Performance Summary (GFlops)

Size   LDA    Align.  Average  Maximal
1000   1000   4       16.9552  18.9241
2000   2000   4       19.7820  19.9920
5000   5008   4       20.9553  21.0879
```

```
10000  10000  4      21.3398  21.6512

Residual checks PASSED

End of tests

Done: Fri Feb 12 05:22:34 UTC 2016
```

LINPACK resulted in an average of **19.758075 GFlops**.

I have achieved **4.5 GFlops** on an average.

2.2 Memory Benchmarking:

Theoretical calculation:

- Base DRAM clock frequency * Number of data transfers per clock * Memory bus (interface) width * Number of interfaces
- Since Amazon AWS **t2.micro instance does not give out the clock frequency**, recording the **local PC's theoretical value** as follows:
  1600000000*2*64*2 = 409600 Mbits per second = 51200 MB/s = **51.2 GB/s**

Benchmark Result:

- With the benchmark experiments, an average of **3.2 GB/s** is benchmarked for **sequential** read+write operation
- With the benchmark experiments, an average of **1.8 GB/s** is benchmarked for **random** read+write operation.

STREAM:

- STREAM resulted in an average of 19.75 GFlops **11.167 GB/s**

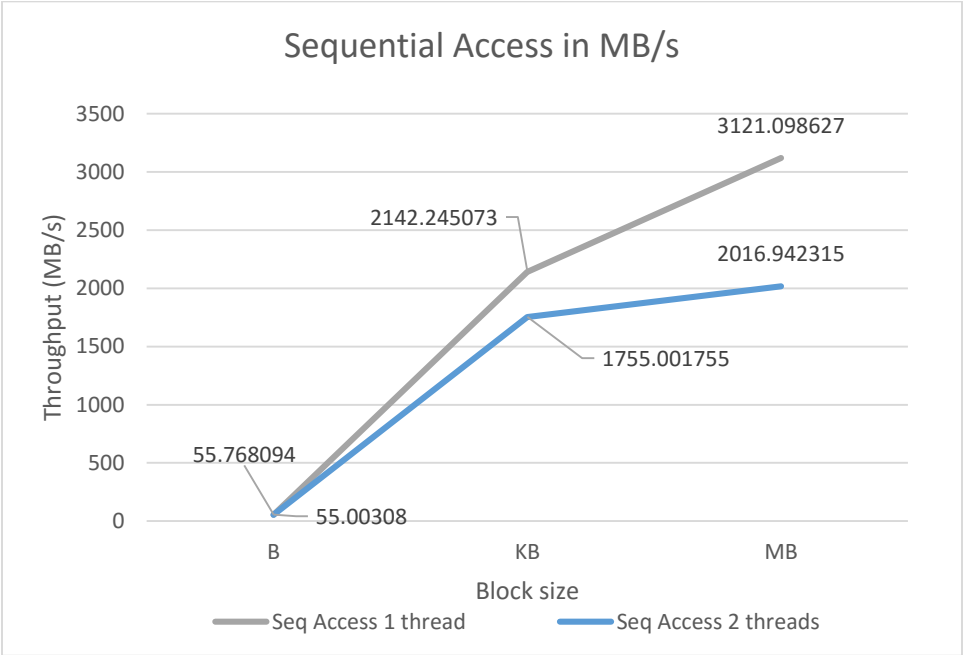
Experiment

In this benchmarking, 12 experiments are being implemented and run on Amazon EC2 t2.micro instance.

Parameter Space:

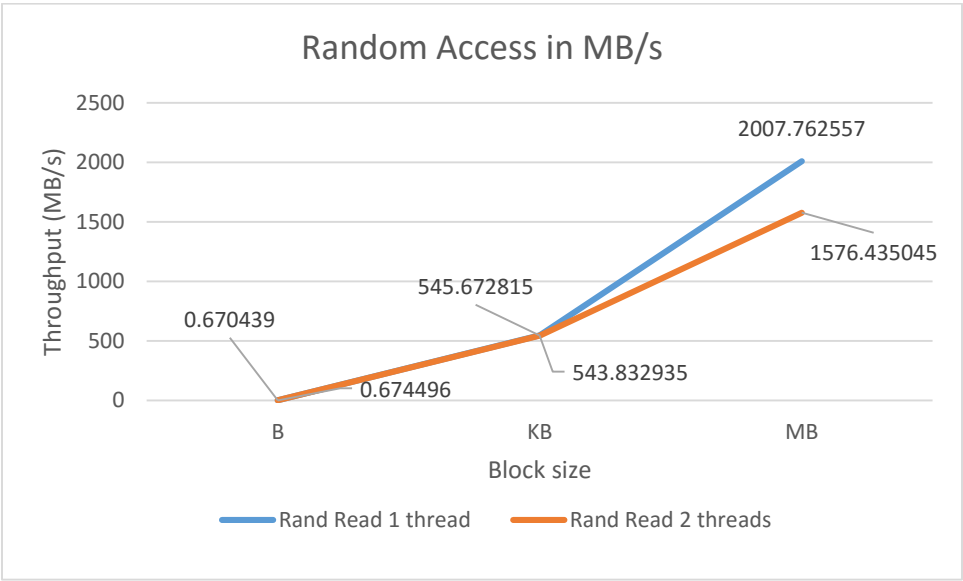
Threads	1, 2
Operation Types	Read+Write
Access Types	Sequential, Random
Block size	1B, 1KB, 1MB

Graphs and Plots:



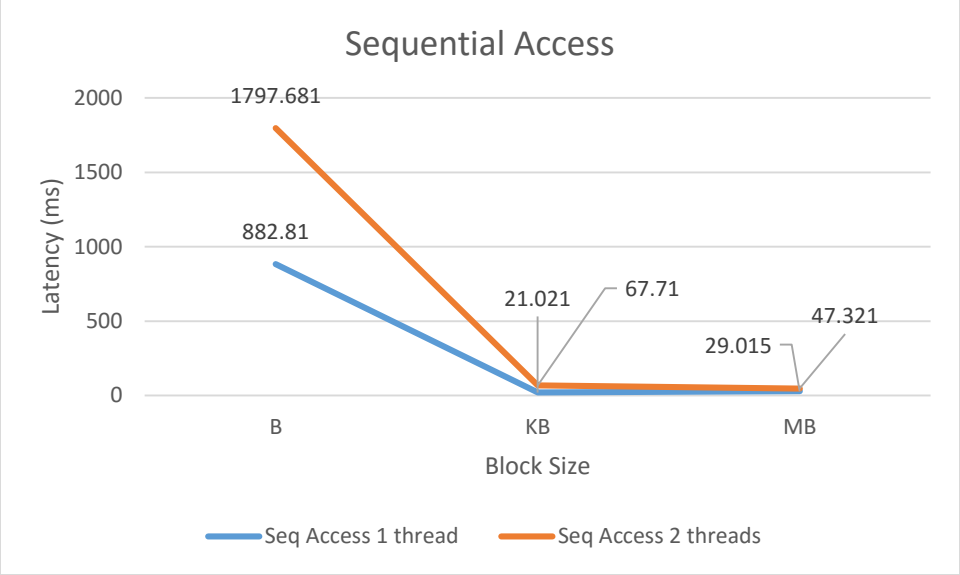
Graph 3: Throughput of Sequential Access (memory)

Graph 3 depicts clearly that, the smaller the accessing block size, lower the throughput; bigger the block size access, higher the throughput.



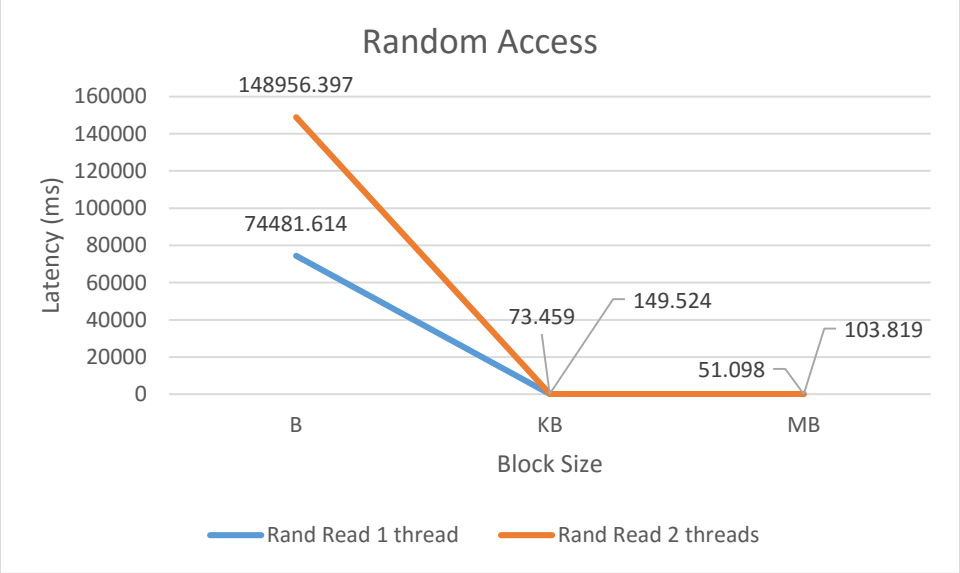
Graph 4: Throughput of Random Access (memory)

Graph 4 also depicts that, the smaller the accessing block size, lower the throughput and vice versa. Also notice that random access of 1KB is relatively lower than the sequential access of 1KB in Graph 3. And the same can be seen for 1MB also.



Graph 5: Latency of Sequential Access (memory)

Graph 5 shows that, the smaller the accessing block size, higher the latency and vice versa.



Graph 6: Latency of Random Access (memory)

Graph 6 depicts that, the smaller the accessing block size, higher the latency and vice versa. Also notice that random access of 1KB is relatively higher than the sequential access of 1KB in Graph 5. And the same can be seen for 1MB also.

STREAM:

```
=== CPU cache information ===
CPU /sys/devices/system/cpu/cpu0 Level 1 Cache: 32K (Data)
CPU /sys/devices/system/cpu/cpu0 Level 1 Cache: 32K (Instruction)
CPU /sys/devices/system/cpu/cpu0 Level 2 Cache: 256K (Unified)
CPU /sys/devices/system/cpu/cpu0 Level 3 Cache: 30720K (Unified)
Total CPU system cache: 31752192 bytes
```

```
Suggested minimum array elements needed: 14432814
Array elements used: 14432814

=== CPU Core Summary ===
processor       : 0
model name    : Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz
cpu MHz                 : 2400.062
siblings: 1

=== Check and build stream ===

=== Testing up to 1 cores ===

-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 8 bytes per array element.
-------------------------------------------------------------
Array size = 14432814 (elements), Offset = 0 (elements)
Memory per array = 110.1 MiB (= 0.1 GiB).
Total memory required = 330.3 MiB (= 0.3 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Number of Threads requested = 1
Number of Threads counted = 1
-------------------------------------------------------------
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 14068 microseconds.
   (= 14068 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:          10929.8     0.021421     0.021128     0.021761
Scale:         10690.0     0.021975     0.021602     0.022243
Add:           11966.6     0.029243     0.028946     0.029561
Triad:         12157.0     0.028911     0.028493     0.029426
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
```

2.3 Disk Benchmarking:

Theoretical calculation:

- Since Amazon EC2 t2.micro uses **EBS storage**, theoretical value will differ from the benchmarked result. So optical hard disk test result is taken into account for theoretical calculation. So local PC's manufacturer advertised as **124.75 MB/s**

Benchmark Result:

- With the benchmark experiments, an average of **4.31 GFLOPS** is benchmarked, which is 43% efficient.
- With the benchmark experiments, an average of **7.06 GIOPS** is benchmarked.

IOZONE:

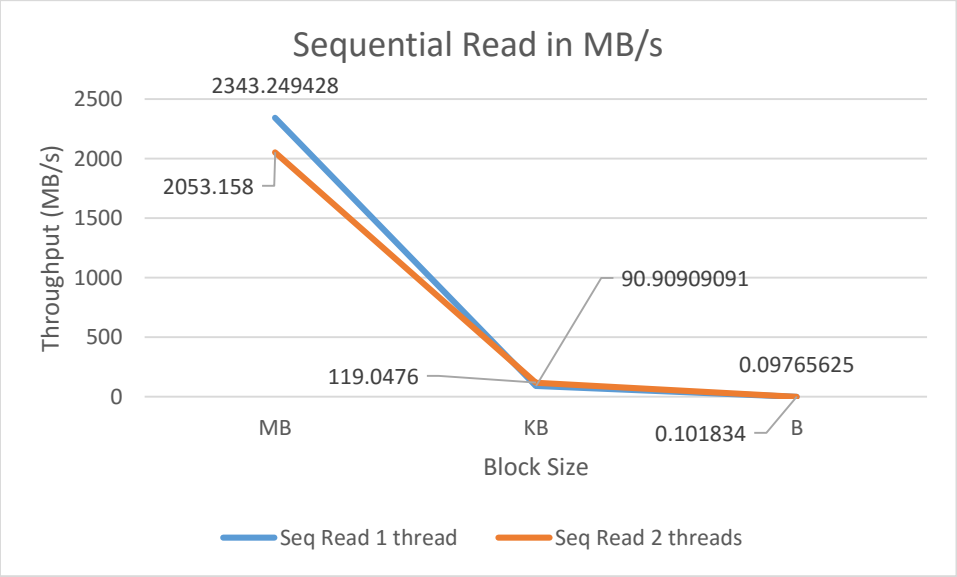
- IOZONE resulted in an average of 19.75 GFlops

In this benchmarking, 24 experiments are being implemented and run on Amazon EC2 t2 micro instance.

Parameter Space:

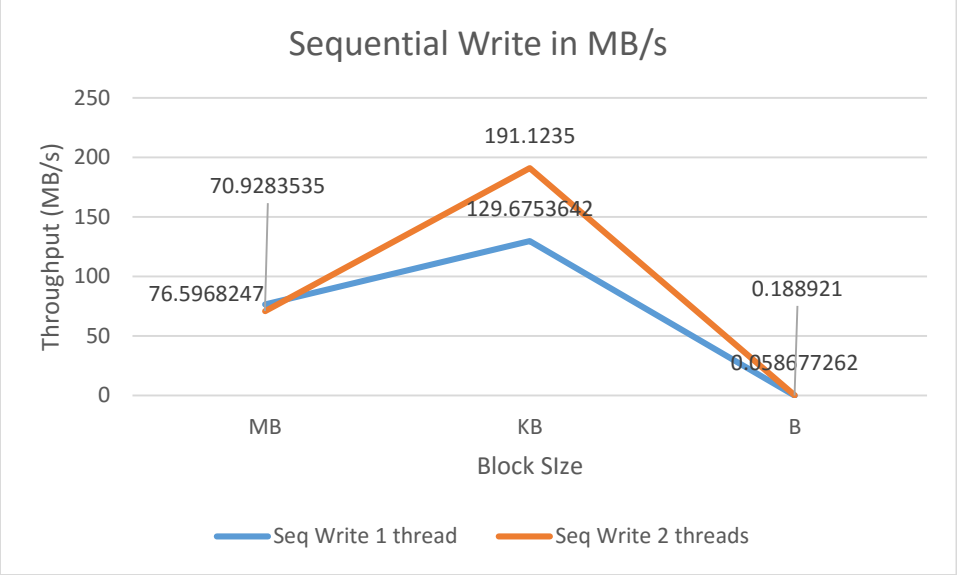
Threads	1, 2
Operation Types	Read, Write
Access Types	Sequential, Random
Block size	1B, 1KB, 1MB

Graphs and Plots:



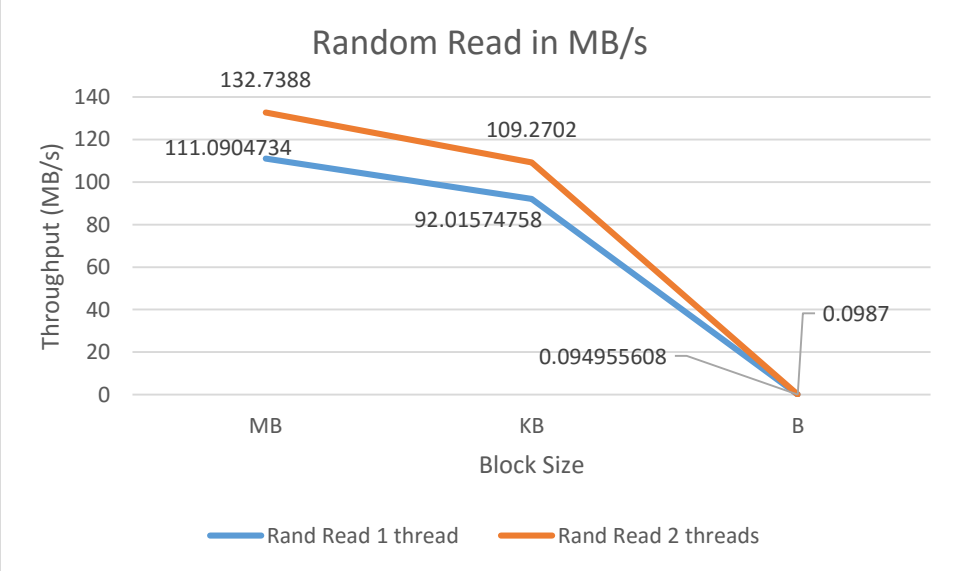
Graph 7: Sequential Read of disk

Graph 7 shows the experiment results of performing sequential reading experiments with various parameters. This graph explains that reading big sectors of disk achieved a higher throughput and reading small sectors of disk achieved a lower throughput.



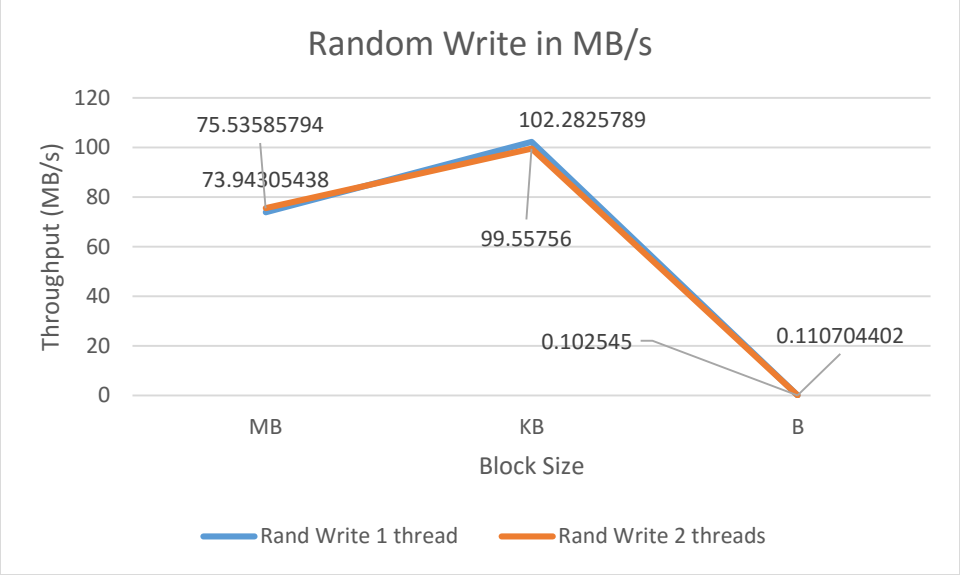
Graph 8: Sequential Write of disk

Graph 8 shows the experiment results of performing sequential writing experiments with various parameters. This graph explains that reading big sectors of disk achieved a higher throughput and reading small sectors of disk achieved a lower throughput. Even though there is some discrepancy in the performance of 1KB and 1MB, the graph explains that, varying the accessing block size, varies the throughput of sequential write.



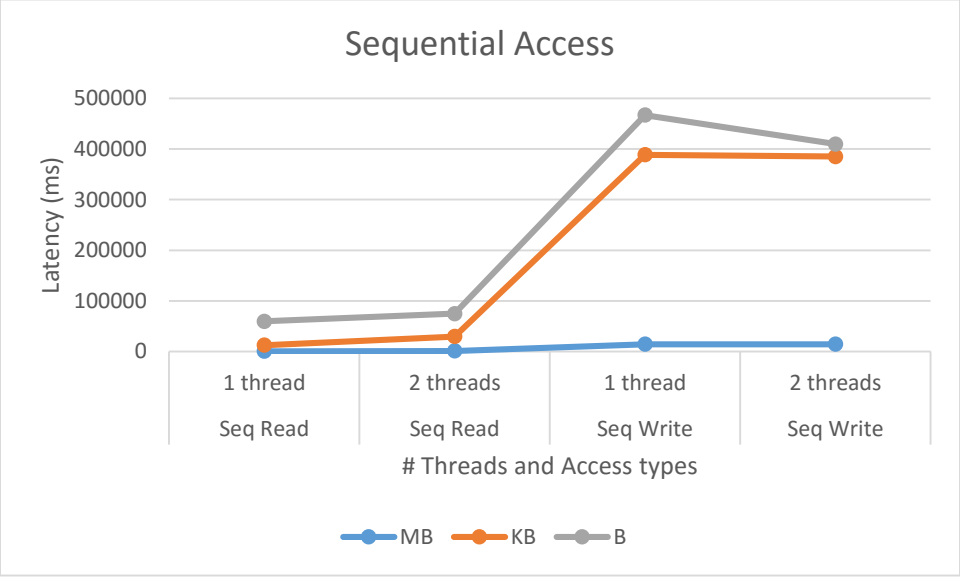
Graph 9: Random Read of disk

On comparing Graph 7 and Graph 9, the throughput of sequential read results a higher value than the random read. The random address calculation may owe to some extent for the depreciation of throughput, but random read always results in a lower throughput than sequential read.



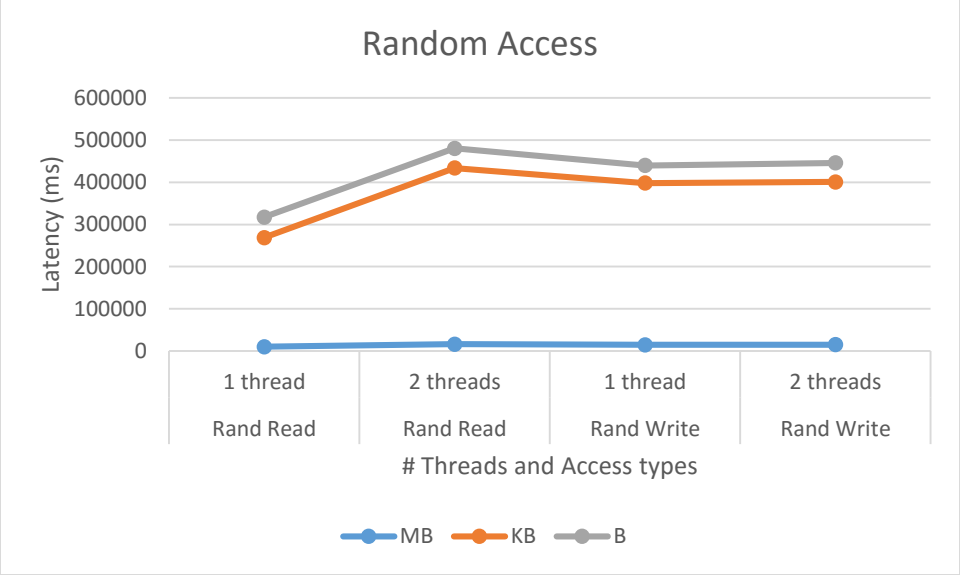
Graph 10: Random Write

On comparing Graph 8 and Graph 10, the throughput of sequential write results a higher value than the random write. The random address calculation may owe to some extent for the depreciation of throughput, but random write always results in a lower throughput than sequential write.



Graph 11: Latency of Sequential Access

Graph 11 shows the latency of performing sequential access of disk using different number of threads.



Graph 12: Latency of Random Access

Graph 12 shows the latency of performing random access of disk using different number of threads.

IOZONE:

Running IOZONE with the following parameters:

- one thread (-t 1)
- use DIRECT I/O for all file operations and inform the filesystem that all operations are to bypass the buffer cache and go directly to disk (-I)

$ iozone -t 1 -R -I

```
"Throughput report Y-axis is type of test X-axis is number of processes"
"Record size = 4 Kbytes "
"Output is in Kbytes/sec"

"  Initial write "    4367.44

"        Rewrite "    4122.12

"           Read "    8699.34

"        Re-read "    9287.79

"    Reverse Read "    9422.68

"     Stride read "    9137.65

"     Random read "    9320.64

" Mixed workload "    9517.22
```

```
"   Random write "    4109.75

"        Pwrite "    4104.11

"         Pread "    8383.28

"        Fwrite " 1684557.62

"         Fread " 3788747.25


iozone test complete.
```

Running IOZONE with the following parameters:

- two threads (-t 2)
- use DIRECT I/O for all file operations and inform the filesystem that all operations are to bypass the buffer cache and go directly to disk (-I)

$ iozone -t 2 -R -I

```
"Throughput report Y-axis is type of test X-axis is number of processes"
"Record size = 4 Kbytes "
"Output is in Kbytes/sec"

"  Initial write "    8992.14

"       Rewrite "    8241.59

"          Read "   17324.19

"       Re-read "   15775.08

"   Reverse Read "   16133.51
"    Stride read "   17197.98

"    Random read "   17538.02

" Mixed workload "   12217.48

"   Random write "    6139.95

"        Pwrite "    8762.28

"         Pread "   14162.67
```

```
"        Fwrite " 2950470.12

"        Fread " 2801528.75


iozone test complete.
```

3. Conclusion

In this work, performance of Amazon EC2 t2.micro's different computer components such as CPU, Memory and Disk are being analyzed and benchmarked by varying various parameter spaces. Even though maximum efficiency is not reached by the experiments, the knowledge of benchmarking different parts of a computer system is learnt through these experiments.