**Illinois Institute of Technology**

**CLOUD COMPUTING**

**Performance Evaluation Document**

**Programming Assignment 1**

**Sudipta Swarnakar -**

**Sohan Samant – A20378826**

**CPU Benchmarking**

1. **Floating Point Operation: –**

To process floating point operations, we run number of floating point instructions with different threads such as 1thread, 2 threads, 4 threads and 8 threads. We are calculating GFLOPS and as we see the numbers of threads are increasing, the load on the processor is also increasing as multiple threads are executing at same instructions for certain amount of time. Now here due to the core of the processors and concurrent execution of threads in parallel, the amount of GFLOPS for thread 1,2,4 and 8 are almost similar. This is due to the new feature called as pipeline feature which is available in the processor. This means we can execute multiple instructions at the same time.

|  |  |  |
| --- | --- | --- |
| **Threads** | **Operation Type** | **GFLOPS** |
| 1 | Floating Point | 12.30164 |
| 2 | Floating Point | 13.558009 |
| 4 | Floating Point | 13.014781 |
| 8 | Floating Point | 13.673878 |

1. **Integer Operations:**

Here we are processing integer operations by running different threads such as 1 thread, 2 threads, 4 threads and 8 threads. This will give us GIOPS. From the numbers we can observer that as we increase the thread count the load on the processor increases because multiple threads are executing simultaneously for certain instructions. Due to the core of the processor and concurrent execution of the threads in parallel, the amount of GIOPS achieved is almost constant. This is due to the new feature called as pipeline feature which is available in the processor. This means we can execute multiple instructions at the same time.

|  |  |  |
| --- | --- | --- |
| **Threads** | **Operation Type** | **IOPS** |
| 1 | Integer | 12.129114 |
| 2 | Integer | 12.706480 |
| 4 | Integer | 13.060921 |
| 8 | Integer | 13.569179 |

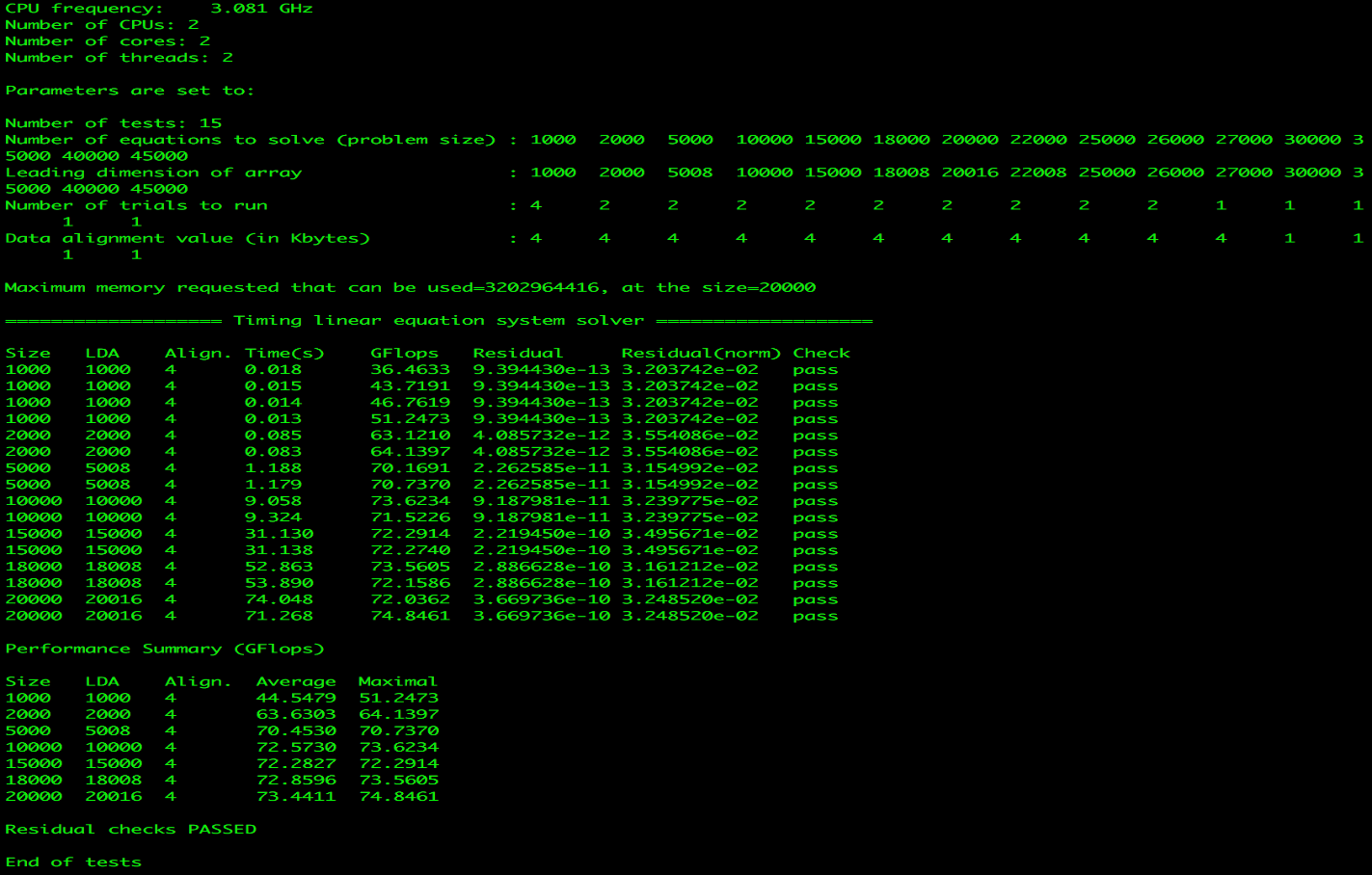
Theoretical Peak Performance = GHz \* Number of Core \* Number of Instruction/Cycle\*Number of CPUs

= 2.299\*8\*2\*2= 73.568 flops/sec

Compared to Theoretical performance, the efficiency achieved by running the benchmark on Medium Chameleon Instance having 2 cores with 2 CPUs was near to around 20% of the Theoretical performance specified by the Intel Family for Xeon E5-2670 v2.

**Linpack Benchmarking:**

We ran Intel’s Linpack benchmark across all cores for two CPUs and recorded the best performance given by it.



From the above Linpack result, it’s clear that the performance we are getting is almost same as theoretical performance which is due the fact that Linpack utilizes AVX instruction to achieve high performance.

**GPU Benchmarking**

For GPU benchmarking, we have used Chameleon Baremetal Instance with CUDA 8 which is having two Nvidia Tesla P100-PCIE-16GB GPU devices.

Below is the details give for Tesla P100

Device Name: Tesla P100-PCIE-16GB

Max Threads Per Block:1024

SM Count:56

Warp Size:32

Clock Rate:1328500

Memory Clock Rate (KHz): 715000

Memory Bus Width (bits): 4096

**Double Precision Floating Point Operation(GFLOPS):**

We leveraged CUDA library to find GFLOPS for the above GPU device. In order to calculate we have simply written code using CUDA C++ where we are declaring two arrays in Host(CPU) and then copying them to Device(GPU) in order to find a new sum values for second array. We have declared host arrays and devices arrays in floating point thus device kernel will perform floating point operation and we can simply calculate GFLOPS. We have run this operation on GPU for 5 times in order to get best performance.

|  |  |  |
| --- | --- | --- |
| **Operation No.** | **Operation Type** | **GFLOPS** |
| 1 | Floating Point | 4094.439697 |
| 2 | Floating Point | 4102.579590 |
| 3 | Floating Point | 4102.579590 |
| 4 | Floating Point | 4094.439697 |
| |  |  |  | | --- | --- | --- | | 5 | Floating Point | 4070.212158 | | Floating Point | 4070.212158 |

**From the above table it’s quite clear that GFLOPS is not fluctuating much. So the average GFLOPS can be calculated as, GFLOPS= 4092.850=4.092850 TFLOPS**

Theoretical GFLOPS = Number of Devices\*CUDA Cores \*clock speed =2\*3584\*(1.328 GHz)= 9519.104 GFLOPS=9.519104 TFLOPS

**Integer Operation Per Second (GIOPS):**

**Just like GFLOPS,** we have declared host arrays and devices arrays in Integer point thus device kernel will perform integer point operation and we can simply calculate GIOPS. We have run this same operation on GPU for 5 times in order to get best performance.

|  |  |  |
| --- | --- | --- |
| **Operation No.** | **Operation Type** | **GIOPS** |
| 1 | Integer | 4094.439697 |
| 2 | Integer | 3991.484619 |
| 3 | Integer | 3983.779053 |
| 4 | Integer | 3999.220215 |
| 5 | Integer | 4006.985596 |

**From the above table it’s quite clear that GIOPS is not fluctuating much. So the average GIOPS can be calculated as, GIOPS= 3992.220**

Theoretical GIOPS = Number of Devices\*CUDA Cores \*clock speed =2\*3584\*(1.328 GHz) = 9519.104 GIOPS=9.519104 TFLOPS

**Half Precision Floating Point Operation (GHOPS):**

After doing our research we found that the best way to implement Half Precision operation for GPU is to use functions from CUDA library like **\_\_float2half** and **\_\_half2float** which can be easily used to convert double precision floating point(fp32) to half precision floating point(fp16). In order to store fp16 data points, we are declaring half data type arrays in device which will stored converted float to half data. We have run this same operation on GPU for 5 times in order to get best performance.

|  |  |  |
| --- | --- | --- |
| **Operation No.** | **Operation Type** | **GHOPS** |
| 1 | Half Precision Floating Point(fp16) | 14201.219238 |
| 2 | Half Precision Floating Point(fp16) | 14102.219238 |
| 3 | Half Precision Floating Point(fp16) | 14078.25610 |
| 4 | Half Precision Floating Point(fp16) | 14170.212158 |
| 5 | Half Precision Floating Point(fp16) | 14478.256104 |

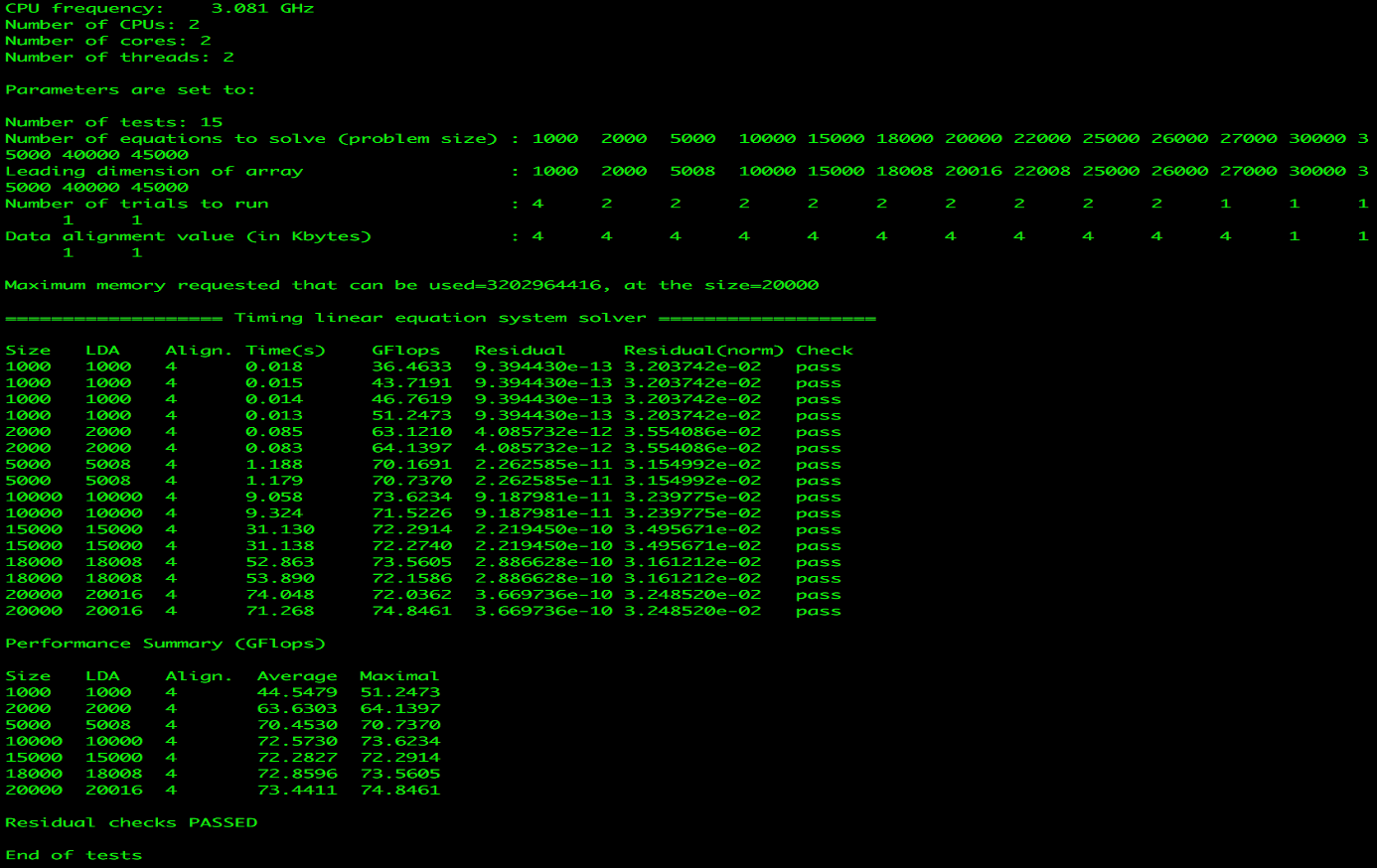
**From the above table it’s quite clear that GHOPS is not fluctuating much. So the average GHOPS can be calculated as, GHOPS= 142201.25158=14.2201 TFLOPS**

Theoretical GHOPS = 4\*Number of Devices\*CUDA Cores \*clock speed =2\*3584\*(1.328 GHz) = 38076.416GFLOPS=38.076416 TFLOPS

**From the above theoretically calculated performance, we can conclude that we are achieving around 44% performance for GFLOPS and GIOPS whereas we are getting 36% performance for GHOPS in comparison to the theoretical values.**

**Linpack Benchmarking:**

We ran Linpack benchmark across all cores and found that the result is close to the theoretically computed values

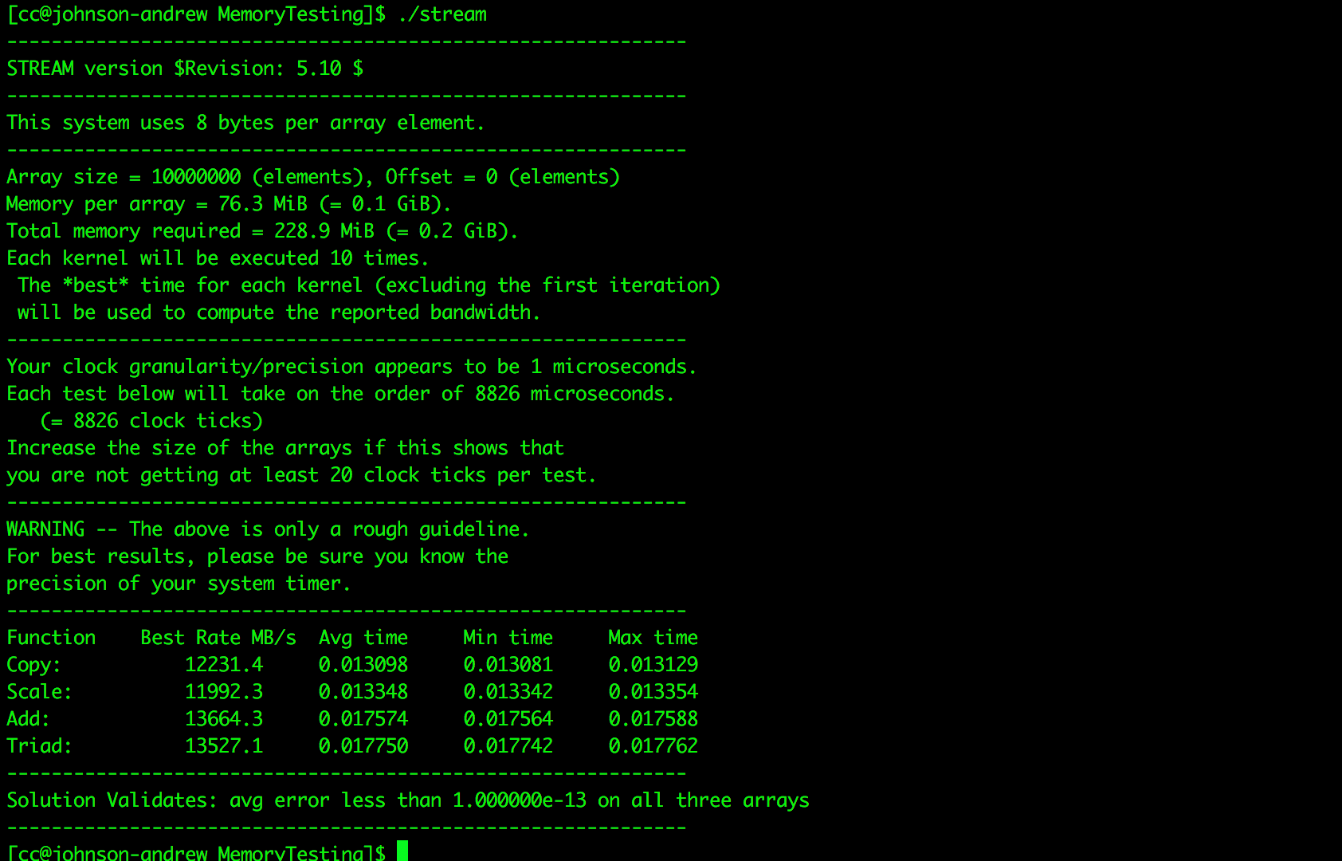


**Memory Benchmarking**

In this benchmark we find how fast the memory is while reading and writing data to and fro. Here we give the set of input such as 8B, 8KB, 8MB and 80MB and then take the thread combination and then analyze disk performance for throughput and latency. From the graph we can analyze with varying number of threads the capability of memory to read and write.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operation Type** | **Block Size** | **Thread Count** | **Throughput (Mbps)** | **Latency(ms)** |
| Sequential Read | 8 | 1 | 169.759186 | 0.000047 |
| Sequential Read | 8 | 2 | 135.831696 | 0.000059 |
| Sequential Read | 8 | 4 | 37.478107 | 0.000213 |
| Sequential Read | 8 | 8 | 26.479593 | 0.000302 |
| Sequential Read | 8192 | 1 | 2427.105957 | 0.003375 |
| Sequential Read | 8192 | 2 | 809.160278 | 0.010124 |
| Sequential Read | 8192 | 4 | 413.483582 | 0.019812 |
| Sequential Read | 8192 | 8 | 162.815979 | 0.050315 |
| Sequential Read | 8388608 | 1 | 2870.511475 | 2.923385 |
| Sequential Read | 8388608 | 2 | 2793.887695 | 3.00356 |
| Sequential Read | 8388608 | 4 | 2582.033447 | 3.25 |
| Sequential Read | 8388608 | 8 | 1819.513306 | 4.612007 |
| Sequential Read | 83886080 | 1 | 3391.296143 | 25.27479 |
| Sequential Read | 83886080 | 2 | 3337.477295 | 25.68236 |
| Sequential Read | 83886080 | 4 | 3277.560059 | 26.15186 |
| Sequential Read | 83886080 | 8 | 2575.013428 | 33.28693 |
| Random Read | 8 | 1 | 113.300598 | 0.000071 |
| Random Read | 8 | 2 | 25.658356 | 0.000312 |
| Random Read | 8 | 4 | 15.658356 | 0.00202 |
| Random Read | 8 | 8 | 12.532296 | 0.048621 |
| Random Read | 8192 | 1 | 2422.153076 | 0.003382 |
| Random Read | 8192 | 2 | 1205.471558 | 0.006796 |
| Random Read | 8192 | 4 | 417.694214 | 0.019612 |
| Random Read | 8192 | 8 | 168.487198 | 0.00202 |
| Random Read | 8388608 | 1 | 2891.761475 | 2.901902 |
| Random Read | 8388608 | 2 | 2799.989746 | 2.997014 |
| Random Read | 8388608 | 4 | 2496.302246 | 3.361615 |
| Random Read | 8388608 | 8 | 1775.008789 | 4.727643 |
| Random Read | 83886080 | 1 | 3417.323486 | 25.08229 |
| Random Read | 83886080 | 2 | 3366.408447 | 25.46164 |
| Random Read | 83886080 | 4 | 3219.644043 | 26.62228 |
| Random Read | 83886080 | 8 | 2784.338867 | 30.78443 |
| Sequential Write | 8 | 1 | 1155.922485 | 0.000007 |
| Sequential Write | 8 | 2 | 1225.922485 | 0.000014 |
| Sequential Write | 8 | 4 | 1695.922485 | 0.000028 |
| Sequential Write | 8 | 8 | 1695.922485 | 0.000056 |
| Sequential Write | 8192 | 1 | 4338.583008 | 0.001888 |
| Sequential Write | 8192 | 2 | 4281.585449 | 0.001913 |
| Sequential Write | 8192 | 4 | 4254.851562 | 0.001925 |
| Sequential Write | 8192 | 8 | 3630.906006 | 0.002256 |
| Sequential Write | 8388608 | 1 | 4394.284668 | 1.909664 |
| Sequential Write | 8388608 | 2 | 4343.451172 | 1.909664 |
| Sequential Write | 8388608 | 4 | 4283.863281 | 1.958888 |
| Sequential Write | 8388608 | 8 | 3700.756836 | 2.267539 |
| Sequential Write | 83886080 | 1 | 4421.876465 | 19.38414 |
| Sequential Write | 83886080 | 2 | 4407.891602 | 19.44564 |
| Sequential Write | 83886080 | 4 | 4374.070801 | 19.596 |
| Sequential Write | 83886080 | 8 | 3780.623047 | 22.672 |
| Random Write | 8 | 1 | 70.066154 | 0.000114 |
| Random Write | 8 | 2 | 80.04314 | 0.000224 |
| Random Write | 8 | 4 | 92.02114 | 0.000364 |
| Random Write | 8 | 8 | 100.04324 | 0.000673 |
| Random Write | 8192 | 1 | 3261.63208 | 0.002512 |
| Random Write | 8192 | 2 | 2969.819092 | 0.002758 |
| Random Write | 8192 | 4 | 2262.494629 | 0.003621 |
| Random Write | 8192 | 8 | 1358.243408 | 0.006031 |
| Random Write | 8388608 | 1 | 5206.95459 | 1.611615 |
| Random Write | 8388608 | 2 | 5159.625488 | 1.626399 |
| Random Write | 8388608 | 4 | 4907.574219 | 1.70993 |
| Random Write | 8388608 | 8 | 4425.447754 | 1.896217 |
| Random Write | 83886080 | 1 | 5480.25293 | 15.64057 |
| Random Write | 83886080 | 2 | 5229.282227 | 16.39121 |
| Random Write | 83886080 | 4 | 4691.696777 | 18.26936 |
| Random Write | 83886080 | 8 | 4267.166992 | 20.08693 |

1. **Reading**
2. **Writing**
   1. **theoretical memory bandwidth of your memory, based on the type of memory and the speed.**
3. **Stream Benchmarking**



**GPU Memory Performance**

Device Name: Tesla P100-PCIE-16GB

Memory Clock Rate (KHz):715000

Memory Bus Width (bits):4096

**Peak Memory Bandwidth (GB/s):732.159973**

Sequential Read Bandwidth for GPU Memory in (GB/s):587.272339

Sequential Write Bandwidth for GPU Memory in (GB/s):213.996552

**Peak Memory Bandwidth (GB/s):732.159973**

Sequential Read Bandwidth for GPU Memory in (GB/s):587.587524

Sequential Write Bandwidth for GPU Memory in (GB/s):223.110580

**Peak Memory Bandwidth (GB/s):732.159973**

Sequential Read Bandwidth for GPU Memory in (GB/s):582.028564

Sequential Write Bandwidth for GPU Memory in (GB/s):211.291489

**Peak Memory Bandwidth (GB/s):732.159973**

Sequential Read Bandwidth for GPU Memory in (GB/s):588.290894

Sequential Write Bandwidth for GPU Memory in (GB/s):213.168152

**Peak Memory Bandwidth (GB/s):732.159973**

Sequential Read Bandwidth for GPU Memory in (GB/s):582.088501

Sequential Write Bandwidth for GPU Memory in (GB/s):211.219589

**Measure the GPU memory performance using sequential read+write and sequential write at full concurrency with large block sizes; compare the measured memory bandwidth with the theoretical memory bandwidth, and explain your results; compare the host memory performance to that of the GPU memory; the GPU memory experiment is extra credit 2**

**Disk Benchmarking**

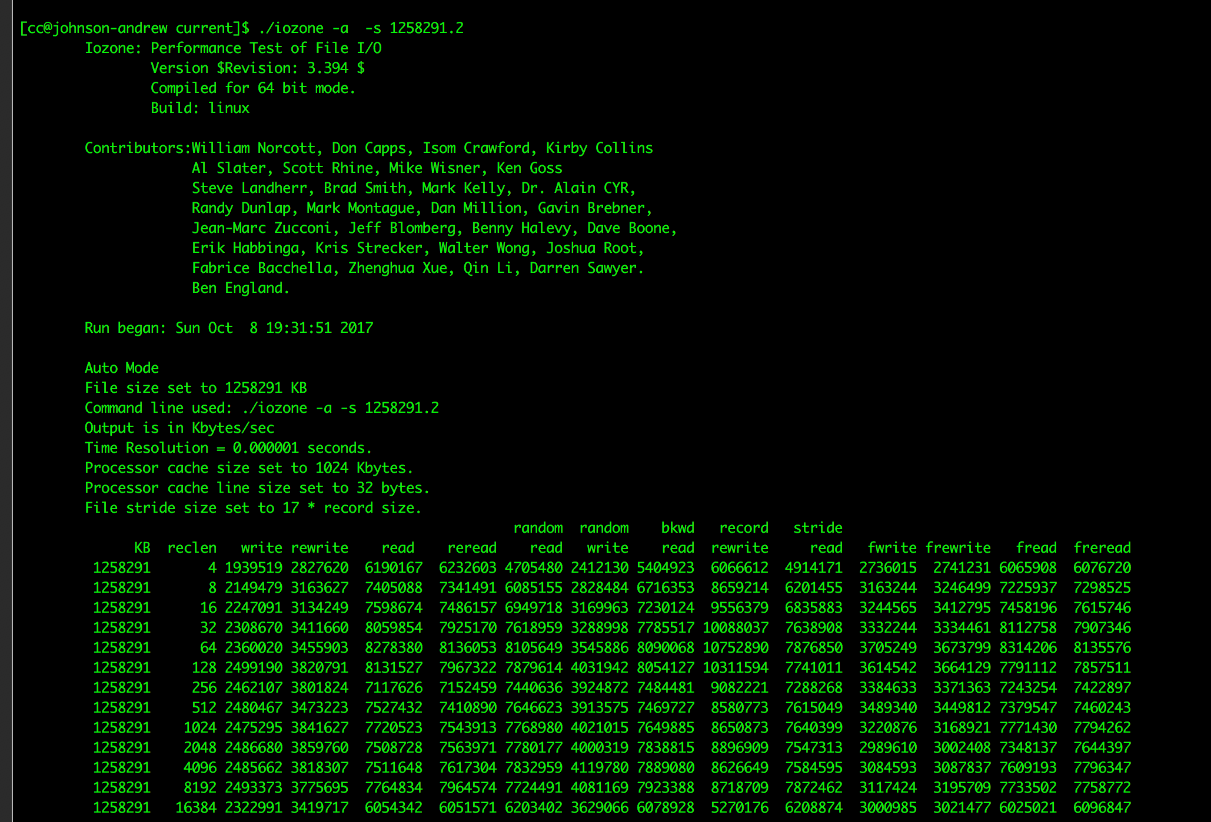
In this benchmark we find how fast the Disk is while reading and writing data to and fro. Here we give the set of input such as 8B, 8KB, 8MB and 80MB and then take the thread combination and then analyze disk performance for throughput and latency.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operation Type** | **Block Size** | **Thread Count** | **Throughput (Mbps)** | **Latency** |
| Sequential Write | 8 | 1 | 12.482747 | 0.000641 |
| Sequential Write | 8 | 2 | 5.055707 | 0.001582 |
| Sequential Write | 8 | 4 | 10.89573 | 0.00422 |
| Sequential Write | 8 | 8 | 120.395089 | 0.022982 |
| Sequential Write | 8192 | 1 | 108.734955 | 2.5 |
| Sequential Write | 8192 | 2 | 101.626808 | 0.080609 |
| Sequential Write | 8192 | 4 | 90.559113 | 0.09046 |
| Sequential Write | 8192 | 8 | 179.949753 | 0.045524 |
| Sequential Write | 8388608 | 1 | 113.431458 | 73.97955 |
| Sequential Write | 8388608 | 2 | 113.329597 | 74.04604 |
| Sequential Write | 8388608 | 4 | 111.710075 | 75.11953 |
| Sequential Write | 8388608 | 8 | 96.891365 | 86.60842 |
| Sequential Write | 83886080 | 1 | 116.491661 | 735.7975 |
| Sequential Write | 83886080 | 2 | 113.438011 | 755.6046 |
| Sequential Write | 83886080 | 4 | 111.483467 | 768.852 |
| Sequential Write | 83886080 | 8 | 108.421776 | 790.5634 |
| Sequential Read | 8 | 1 | 24.241274 | 0.00033 |
| Sequential Read | 8192 | 1 | 2436.904541 | 0.003362 |
| Sequential Read | 8388608 | 1 | 5128.292969 | 1.636336 |
| Sequential Read | 83886080 | 1 | 5725.819336 | 14.96979 |
| Sequential Read | 8 | 2 | 18.090269 | 0.000442 |
| Sequential Read | 8192 | 2 | 783.488281 | 0.010456 |
| Sequential Read | 8388608 | 2 | 4960.80957 | 1.69158 |
| Sequential Read | 83886080 | 2 | 5221.841309 | 16.41457 |
| Sequential Read | 8 | 4 | 8.728922 | 0.000916 |
| Sequential Read | 8192 | 4 | 482.469482 | 0.016979 |
| Sequential Read | 8388608 | 4 | 691.839233 | 12.12942 |
| Sequential Read | 83886080 | 4 | 2784.287354 | 30.785 |
| Sequential Read | 8 | 8 | 150.32 | 0.007017 |
| Sequential Read | 8192 | 8 | 167.487564 | 0.048911 |
| Sequential Read | 8388608 | 8 | 1924.032715 | 4.361468 |
| Sequential Read | 83886080 | 8 | 2351.474854 | 36.45129 |
| Random Read | 8 | 1 | 13.159264 | 0.000608 |
| Random Read | 8192 | 1 | 1676.713013 | 0.004886 |
| Random Read | 8388608 | 1 | 2715.423584 | 3.09035 |
| Random Read | 83886080 | 1 | 2951.535889 | 29.04057 |
| Random Read | 8 | 2 | 7.392732 | 0.001082 |
| Random Read | 8192 | 2 | 708.726868 | 0.011559 |
| Random Read | 8388608 | 2 | 2609.574951 | 3.215699 |
| Random Read | 83886080 | 2 | 2910.043457 | 29.45464 |
| Random Read | 8 | 4 | 4.147112 | 0.001929 |
| Random Read | 8192 | 4 | 431.651581 | 0.018978 |
| Random Read | 8388608 | 4 | 2312.031738 | 3.629539 |
| Random Read | 83886080 | 4 | 2723.064209 | 31.47714 |
| Random Read | 8 | 8 | 110.342332 | 0.02832 |
| Random Read | 8192 | 8 | 187.397308 | 0.043715 |
| Random Read | 8388608 | 8 | 1774.470703 | 4.729077 |
| Random Read | 83886080 | 8 | 2363.330566 | 36.26843 |
| Random Write | 8 | 1 | 18.7333 | 0.023233 |
| Random Write | 8192 | 1 | 50.100712 | 0.163511 |
| Random Write | 8388608 | 1 | 138.552704 | 60.56618 |
| Random Write | 83886080 | 1 | 176.76506 | 484.9052 |
| Random Write | 8 | 2 | 23.37393 | 0.011323 |
| Random Write | 8192 | 2 | 63.433838 | 0.129143 |
| Random Write | 8388608 | 2 | 118.4897 | 70.82141 |
| Random Write | 83886080 | 2 | 124.322365 | 689.4518 |
| Random Write | 8 | 4 | 26.7332 | 0.04342 |
| Random Write | 8192 | 4 | 66.217323 | 0.123714 |
| Random Write | 8388608 | 4 | 108.764076 | 77.15422 |
| Random Write | 83886080 | 4 | 110.710861 | 774.2175 |
| Random Write | 8 | 8 | 29.8322 | 0.05644 |
| Random Write | 8192 | 8 | 68.267433 | 0.119999 |
| Random Write | 8388608 | 8 | 109.310913 | 76.76826 |
| Random Write | 83886080 | 8 | 116.905777 | 733.1912 |

1. **Write**

**IOZONE:**

This is a disk benchmark which is a standard used to verify the disk capability to transfer data sequentially and randomly based on data size.



* 1. **Identify the optimal number of concurrency to get the best performance; explain your findings, putting in perspective the hardware you are testing; can you tell if the disk you are evaluating is a spinning hard drive (HDD) or a solid state memory (SSD) disk?**

1. **Theoritical performance**

**Network Benchmarking**

This benchmarking is done to calculate the network bandwidth using strong scaling approach.

1. **TCP Protocol Benchmarking**

In this benchmark we are transferring total 8 GB of data, where packet/buffer size is 64 KB.

|  |  |  |  |
| --- | --- | --- | --- |
| **Thread Count** | **Total Data** | **Throughput (Mb/s)** | **Latency (ms)** |
| 1 | 8 GB | 65536.63 | 0.000463126 |
| 2 | 8 GB | 65550.34 | 0.000465612 |
| 4 | 8 GB | 131072.37 | 0.0034223 |
| 8 | 8 GB | 143064.32 | 0.0023422 |

TCP is a connection oriented protocol and each buffer or packet is transferred in bytes from client to server. The time taken to transfer 64 KB using 1 thread, 2 threads, 4 threads and 8 threads model from client to server and receive the acknowledgement. The throughput of the network is almost similar for one thread and two threads. This indicates the server’s capability to handle multiple client requests and serving equally, which is looking like there is a dedicated server for handling each client. We can see almost similar performance among thread 1 and thread 2 as well as thread 4 and thread 8, this is because the bandwidth is being shared among threads which is why it is giving equal throughput and latency.

1. **UDP Protocol Benchmarking**

In this benchmark we transfer 8 GB of data with buffer/packet size of 64 KB.

|  |  |  |  |
| --- | --- | --- | --- |
| **Thread Count** | **Total Data** | **Throughput (Mb/s)** | **Latency (ms)** |
| 1 | 8 GB | 65536.63 | 0.000463126 |
| 2 | 8 GB | 65550.34 | 0.000465612 |
| 4 | 8 GB | 131072.37 | 0.0034223 |
| 8 | 8 GB | 143064.32 | 0.0023422 |

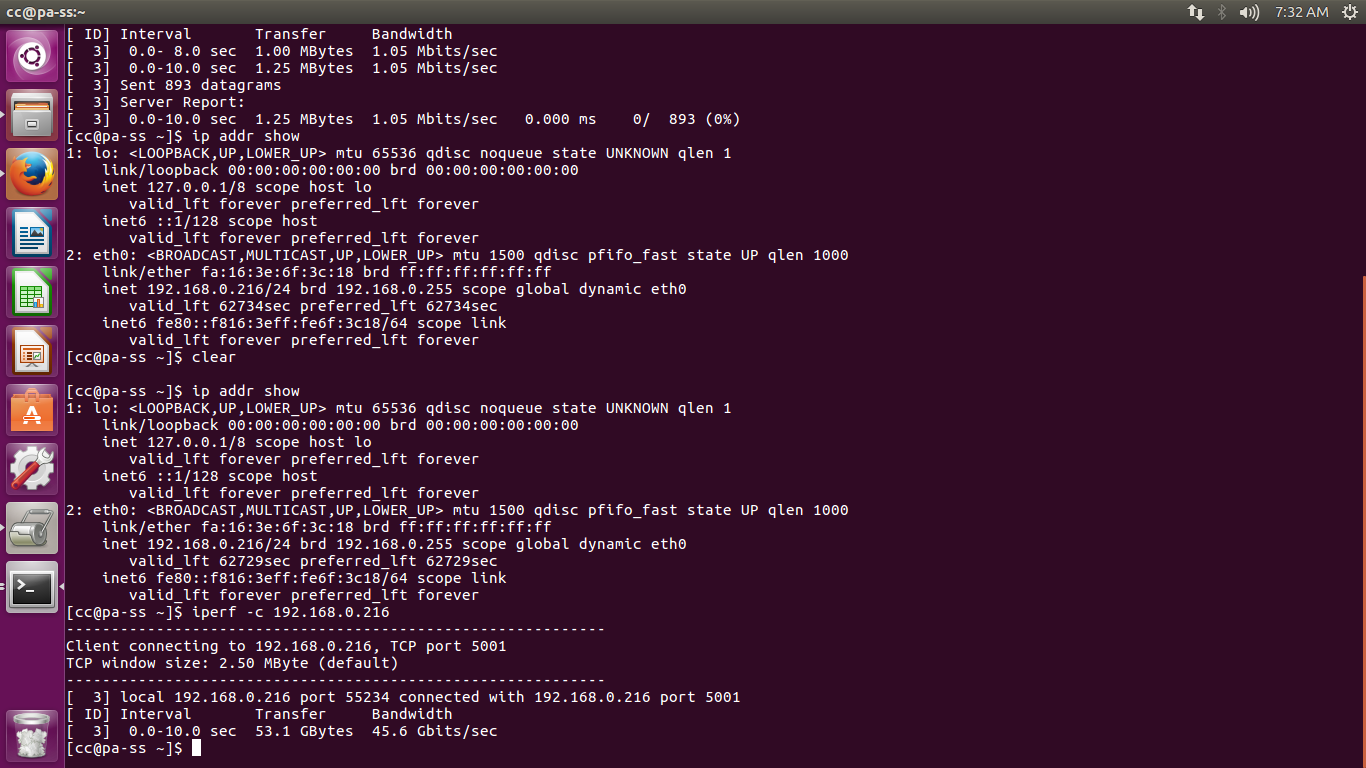
UDP sends data in short form of packets called as datagrams. This protocol is a connectionless protocol and hence data will be travelling individually from client to server and the packet might be received out of order. There is no acknowledgement from Server about the reception of data unlike TCP and the data might get lost in the network. UDP’s throughput is higher than TCP. In the multithreaded environment each datagram which is sent from client to server, there will be either that packet will be send and received by the server or it might get lost. Here the server is a multithreaded and hence serving many clients at the same time. In the benchmarking if you can see, the throughput for thread 1 and thread 2 is almost similar as they share the same network bandwidth.

**---------------- GRAPH to be drawn -------------**

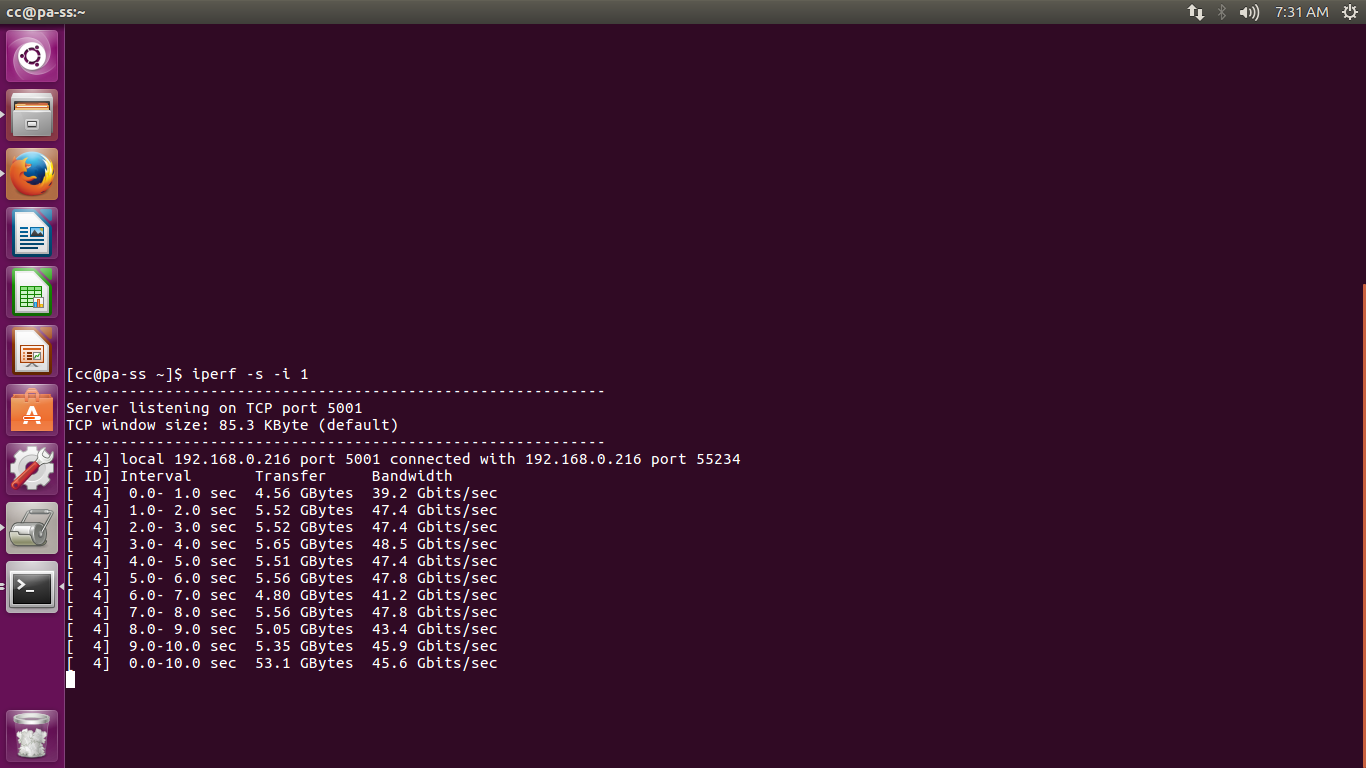
1. **IPerf Benchmarking**

Iperf is used for network performance measurement. Through this tool we can measure the network throughput for TCP and UDP protocol. This tool has a client and server functionality and can create data streams to measure the throughput between the two end clients in one or both directions.

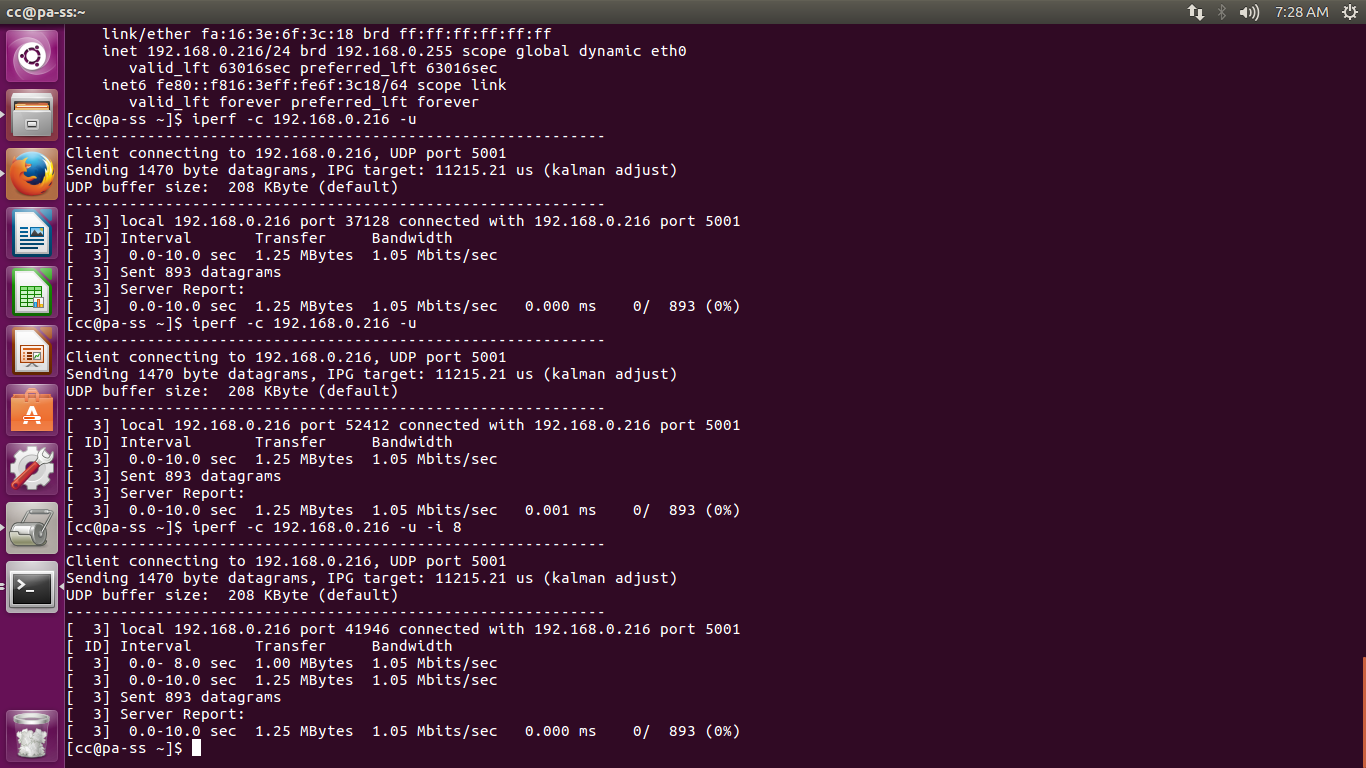
**Iperf TCP Client**:



**Iperf TCP Server**:



**Iperf UDP Client**:



**Iperf UDP Server**:

