# CS 553: Cloud Computing - HA1


Saptarshi Chatterjee
CWID: A20413922


March 28, 2018

## 1 Hardware Specification

Experiments are run on 2 different clusters -

- **Hyperion** : Intel Xeon E312xx (Sandy Bridge). MemTotal: 4046452 kB . Ethernet interface product: Virtio network device
- **Promethus** Disk- is Micron 5100 PRO 2.5"480GB,SATA,6Gb/s,3D NAND,7mm,1.5DWPD2, NIC- Super Micro Computer Inc Ethernet Controller 10-Gigabit X540-AT2, CPU - Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz , 8GiB DIMM Synchronous 2400 MHz (0.4 ns, 64bit)

## 2 CPU Benchmarking (Run on Hyperion)

We were asked to perform 1 trillion arithmetic (quarter precision, half precision, single precision, double precision) using 1, 2 and 4 threads.

$$\begin{aligned}
\text{FLOPS } &= sockets \times \frac{cores}{socket} \times \frac{cycles}{second} \times \frac{FLOPs}{cycle} \\
&= 2 * 1 * 2.299 * 8 \\
&= 36.78 (ForDP) (\text{Sandy Bridge -8 DP FLOPs/cycle}) \\
&= 73.56 (forSP) (\text{16 SP FLOPs/cycle: 8-wide AVX addition + 8-wide AVX multiplication}) \\
&= 147.136 (forQP) \\
&= 147.136 (forHP)
\end{aligned}$$

## Processor Spec

```
schatterjee@bluecompute-7:~$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                2
On-line CPU(s) list:   0,1
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             2
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 42
Model name:            Intel Xeon E312xx (Sandy Bridge)
Stepping:              1
CPU MHz:               2299.996
BogoMIPS:              4599.99
Hypervisor vendor:     KVM
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              4096K
NUMA node0 CPU(s):     0,1
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx
 fxsr sse sse2 ss syscall nx pdpe1gb rdtscp lm constant_tsc rep_good nopl eagerfpu pni pclmulqdq ssse3 fma
 cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_
lm abm invpcid_single retpoline kaiser fsgsbase bmi1 avx2 smep bmi2 erms invpcid xsaveopt
schatterjee@bluecompute-7:~$
```

## Linpack output performance

```
Precisions- DP, Threads 4, TotalTime - 58.147861, GigaOps- 17.197537
schatterjee@bluecompute-7:~$ /exports/home/schatterjee/l_mklb_p_2018.1.009/benchmarks_2018/linux/mkl/benchmarks/linpack/xlinpack_xeon64 /exports/home/schatterj
/l_mklb_p_2018.1.009/benchmarks_2018/linux/mkl/benchmarks/linpack/lininput_xeon64
Sample data file lininput_xeon64.

Current date/time: Sun Mar 25 04:51:25 2018

CPU frequency:    2.751 GHz
Number of CPUs: 2
Number of cores: 2
Number of threads: 2

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=3202964416, at the size=20000

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

Size   LDA    Align. Time(s)    GFlops   Residual     Residual(norm) Check
1000   1000   4      0.015      43.6410  9.394430e-13 3.203742e-02   pass
1000   1000   4      0.013      49.9420  9.394430e-13 3.203742e-02   pass
1000   1000   4      0.013      50.0427  9.394430e-13 3.203742e-02   pass
1000   1000   4      0.014      49.3992  9.394430e-13 3.203742e-02   pass

2000   2000   4      0.092      57.7486  4.085732e-12 3.554086e-02   pass
2000   2000   4      0.085      62.9289  4.085732e-12 3.554086e-02   pass
5000   5008   4      1.278      65.2580  2.262585e-11 3.154992e-02   pass
5000   5008   4      1.243      67.1011  2.262585e-11 3.154992e-02   pass
10000  10000  4      10.064     66.2655  9.187981e-11 3.239775e-02   pass
10000  10000  4      9.543      69.8812  9.187981e-11 3.239775e-02   pass
```

## *CPU performance Table

| Workload | Concurrency | MyCPUBench Measured Ops/Sec | HPL Measured Ops/Sec (GigaOPS) | Theoretical Ops/Sec (GigaOPS) | MyCPUBench Efficiency (%) | HPL Efficiency (%) |
|---|---|---|---|---|---|---|
| QP | 1 | 45.393182 | N/A | 73.568 | 61.70234613 | N/A |
| QP | 2 | 90.078258 | N/A | 147.136 | 61.22108661 | N/A |
| QP | 4 | 85.05064 | N/A | 294.272 | 28.9020498 | N/A |
| HP | 1 | 46.200567 | N/A | 73.568 | 62.79981378 | N/A |
| HP | 2 | 86.669638 | N/A | 147.136 | 58.90444079 | N/A |
| HP | 4 | 85.924472 | N/A | 294.272 | 29.19899685 | N/A |
| SP | 1 | 23.579637 | N/A | 36.784 | 64.1029714 | N/A |
| SP | 2 | 46.652203 | N/A | 73.568 | 63.41371656 | N/A |
| SP | 4 | 43.583872 | N/A | 147.136 | 29.6214876 | N/A |
| DP | 1 | 8.508647 | 11.1738 | 18.392 | 46.26276098 | 60.75358852 |
| DP | 2 | 16.862083 | 24.9512 | 36.784 | 45.8408085 | 67.83166594 |
| DP | 4 | 17.197537 | 69.218 | 73.568 | 23.37638239 | 94.08710309 |

## Synopsis for CPU performance

- Getting best result for QP with 2 threads . 90.07 GIOPS.
- GFLOPS is lowest for DP with 1 thread as we are not optimally using the available CPU power.
- I'm getting around 23% to 69% efficiency varying between different pre-

cision and concurrency using AVX. Where as Linpack gets 60%-94% efficiency

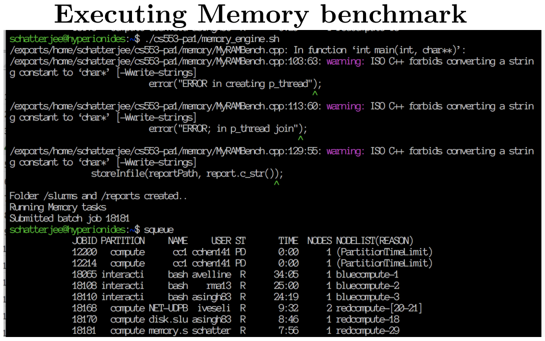
- It's not a Quad-core machine so using 4 threads actually hurts performance than 2 threads as a lot of time is spent in context switch .
- Linpack gets a best performance of 69.2180 GFLOPS for problem size 20000.

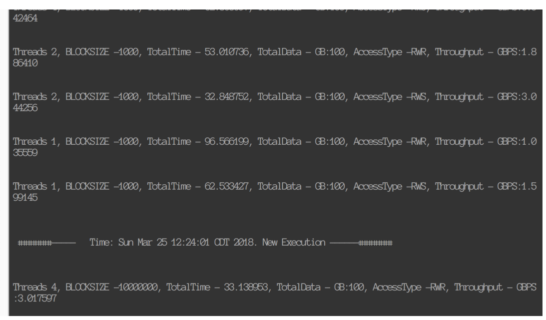
# 3 Memory Benchmarking (Run on Hyperion)

We were asked to perform random and Sequential Read/Write Ops on memory 1GB Workload. Operated over it 100X times with various access patterns (RWS, RWR) , various block sizes (1KB, 1MB, 10MB) and Concurrency (1 thread, 2 threads, 4 threads)

$$
\begin{aligned}
\text{Theoratical Bandwidth } &= \text{Base DRAM clock frequency} \times \text{Number of data transfers per clock} \\
&\times \text{Memory bus (interface) width} \times \text{Number of interfaces} \\
&= 2133,000,000 * 2 * 64 * 1/(8 * 1000 * 1000 * 1000) \\
&\ldots \text{Assuming clock frequency 2133MHz} \\
&= 34.128 GBps
\end{aligned}
$$

**Executing Memory benchmark**




**pmbw - Parallel Memory Bandwidth Benchmark / Measurement**

```
Allocating 2048 MiB for testing.
Running nthreads=1 factor=1073741824 areasize=1024 thrsize=1152 testsize=1152 repeats=932068 testvol=1073742336 testaccess=134217792
run time = 0.0973188 -> rerunning test with repeat factor=16549869477
Running nthreads=1 factor=16549869477 areasize=1024 thrsize=1152 testsize=1152 repeats=14366207 testvol=16549870464 testaccess=2068733808
run time = 0.961803 -> rerunning test with repeat factor=25810688405
Running nthreads=1 factor=25810688405 areasize=1024 thrsize=1152 testsize=1152 repeats=22405112 testvol=25810689024 testaccess=3226336128
run time = 1.49148 -> next test with repeat factor=25958136952
RESULT  datetime=2018-03-25 19:08:01   host=bluecompute-5   version=0.6.2   funcname=ScanWrite64PtrSimpleLoop   nthreads=1   areasize=1024   threadsize=1152 testsize=1152   repeats=22405112   testv
l=25810689024   testaccess=3226336128   time=1.4914796699886210263   bandwidth=17305424635.252933502 rate=4.6228279101011915329e-10
Running nthreads=1 factor=25958136952 areasize=2048 thrsize=2176 testsize=2176 repeats=11929291 testvol=25958137216 testaccess=3244767152
run time = 1.37553 -> next test with repeat factor=28306989248
RESULT  datetime=2018-03-25 19:08:02   host=bluecompute-5   version=0.6.2   funcname=ScanWrite64PtrSimpleLoop   nthreads=1   areasize=2048   threadsize=2176 testsize=2176   repeats=11929291   testv
l=25958137216   testaccess=3244767152   time=1.3755332820001058028   bandwidth=18871326165.409355164 rate=4.2392357218984378163e-10
Running nthreads=1 factor=28306989248 areasize=3072 thrsize=3200 testsize=3200 repeats=8845935 testvol=28306992000 testaccess=3538374000
run time = 1.61965 -> next test with repeat factor=26215814064
RESULT  datetime=2018-03-25 19:08:04   host=bluecompute-5   version=0.6.2   funcname=ScanWrite64PtrSimpleLoop   nthreads=1   areasize=3072   threadsize=3200 testsize=3200   repeats=8845935 testvol=28306
92000   testaccess=3538374000   time=1.619651707005687058   bandwidth=17477209376.287593842 rate=4.5773898039203518802e-10
Running nthreads=1 factor=26215814064 areasize=4096 thrsize=4224 testsize=4224 repeats=6206396 testvol=26215816704 testaccess=3276977088
run time = 1.41459 -> next test with repeat factor=27798606260
RESULT  datetime=2018-03-25 19:08:06   host=bluecompute-5   version=0.6.2   funcname=ScanWrite64PtrSimpleLoop   nthreads=1   areasize=4096   threadsize=4224 testsize=4224   repeats=6206396 testvol=26215
16704   testaccess=3276977088   time=1.4145934039843268692   bandwidth=18532404173.638053894 rate=4.3167631814224234948e-10
Running nthreads=1 factor=27798606260 areasize=6144 thrsize=6272 testsize=6272 repeats=4432176 testvol=27798607872 testaccess=3474825984
run time = 1.46042 -> next test with repeat factor=28551935931
RESULT  datetime=2018-03-25 19:08:07   host=bluecompute-5   version=0.6.2   funcname=ScanWrite64PtrSimpleLoop   nthreads=1   areasize=6144   threadsize=6272 testsize=6272   repeats=4432176 testvol=27798
07872   testaccess=3474825984   time=1.4604232759447768331   bandwidth=19034623954.494651794 rate=4.2028673742782073416e-10
Running nthreads=1 factor=28551935931 areasize=8192 thrsize=8320 testsize=8320 repeats=3431724 testvol=28551943680 testaccess=3568992960
run time = 1.38113 -> next test with repeat factor=31009329963
RESULT  datetime=2018-03-25 19:08:09   host=bluecompute-5   version=0.6.2   funcname=ScanWrite64PtrSimpleLoop   nthreads=1   areasize=8192   threadsize=8320 testsize=8320   repeats=3431724 testvol=28551
43680   testaccess=3568992960   time=1.3811299880035221577   bandwidth=20672886642.098735809 rate=3.8698030606468950462e-10
Running nthreads=1 factor=31009329963 areasize=12288 thrsize=12416 testsize=12416 repeats=2497530 testvol=31009332480 testaccess=3876166560
run time = 1.48354 -> next test with repeat factor=31353435853
RESULT  datetime=2018-03-25 19:08:10   host=bluecompute-5   version=0.6.2   funcname=ScanWrite64PtrSimpleLoop   nthreads=1   areasize=12288  threadsize=12416     testsize=12416  repeats=2497530 testv
l=31009332480   testaccess=3876166560   time=1.4835375279653817415   bandwidth=20902290569.304424286 rate=3.8273317335604425165e-10
Running nthreads=1 factor=31353435853 areasize=16384 thrsize=16512 testsize=16512 repeats=1898828 testvol=31353447936 testaccess=3919180992
run time = 1.52678 -> next test with repeat factor=30803510218
RESULT  datetime=2018-03-25 19:08:12   host=bluecompute-5   version=0.6.2   funcname=ScanWrite64PtrSimpleLoop   nthreads=1   areasize=16384  threadsize=16512     testsize=16512  repeats=1898828 testv
l=31353447936   testaccess=3919180992   time=1.5267796289990656078   bandwidth=20535673479.318599701 rate=3.8956599149556845237e-10
Running nthreads=1 factor=30803510218 areasize=20480 thrsize=20608 testsize=20608 repeats=1494736 testvol=30803519488 testaccess=3850439936
run time = 1.62444 -> next test with repeat factor=28443820394
RESULT  datetime=2018-03-25 19:08:13   host=bluecompute-5   version=0.6.2   funcname=ScanWrite64PtrSimpleLoop   nthreads=1   areasize=20480  threadsize=20608     testsize=20608  repeats=1494736 testv
l=30803519488   testaccess=3850439936   time=1.6244399870047345757   bandwidth=18962546929.663963318 rate=4.2188425582669172485e-10
Running nthreads=1 factor=28443820394 areasize=24576 thrsize=24704 testsize=24704 repeats=1151386 testvol=28443839744 testaccess=3555479968
run time = 1.53197 -> next test with repeat factor=27850179214
RESULT  datetime=2018-03-25 19:08:15   host=bluecompute-5   version=0.6.2   funcname=ScanWrite64PtrSimpleLoop   nthreads=1   areasize=24576  threadsize=24704     testsize=24704  repeats=1151386 testv
l=28443839744   testaccess=3555479968   time=1.5319743290031072557   bandwidth=18566786143.205066681 rate=4.3087694005285775267e-10
Running nthreads=1 factor=27850179214 areasize=28672 thrsize=28800 testsize=28800 repeats=967021 testvol=27850204800 testaccess=3481275600
```

corresponding results re kept in /cs553-pa1/memory/pmbw.txt

**Theoratical Latency**

$Latency(ns) = clockcycletime(ns) x number of clockcycles =$

**Memory benchmark Table**

| Workload | Concurrency | Block Size | MyRAMBench Measured Throughput (GB/sec) | pmbw Measured Throughput (GB/sec) | Theoretical Throughput (GB/sec) | MyRAMBench Efficiency (%) | pmbw Efficiency (%) |
|---|---|---|---|---|---|---|---|
| RWS | 1 | 1KB | 1.63132 | 17.3 | 34.128 | 4.78000469 | 50.6915143 |
| RWS | 1 | 1MB | 1.69056 | 16.22 | 34.128 | 4.9535865 | 47.52695734 |
| RWS | 1 | 10MB | 1.616496 | 15.59 | 34.128 | 4.73656821 | 45.68096578 |
| RWS | 2 | 1KB | 3.151194 | 29.77 | 34.128 | 9.2334564 | 87.23042663 |
| RWS | 2 | 1MB | 3.269755 | 29.92 | 34.128 | 9.58085736 | 87.66994843 |
| RWS | 2 | 10MB | 2.97559 | 30.67 | 34.128 | 8.71891116 | 89.86755743 |
| RWS | 4 | 1KB | 3.150432 | 26.42 | 34.128 | 9.23122363 | 77.41443976 |
| RWS | 4 | 1MB | 3.234842 | 34.08 | 34.128 | 9.4785572 | 99.85935302 |
| RWS | 4 | 10MB | 3.008679 | 30.71 | 34.128 | 8.81586674 | 89.98476324 |
| RWR | 1 | 1KB | 1.137239 | 17.3 | 34.128 | 3.33227555 | 50.6915143 |
| RWR | 1 | 1MB | 1.53018 | 16.22 | 34.128 | 4.48364979 | 47.52695734 |
| RWR | 1 | 10MB | 1.566664 | 15.59 | 34.128 | 4.59055321 | 45.68096578 |
| RWR | 2 | 1KB | 1.975065 | 29.77 | 34.128 | 5.7872275 | 87.23042663 |
| RWR | 2 | 1MB | 2.928384 | 29.92 | 34.128 | 8.58059072 | 87.66994843 |
| RWR | 2 | 10MB | 3.030498 | 30.67 | 34.128 | 8.87979958 | 89.86755743 |
| RWR | 4 | 1KB | 2.065833 | 26.42 | 34.128 | 6.05319093 | 77.41443976 |
| RWR | 4 | 1MB | 2.978313 | 34.08 | 34.128 | 8.72688994 | 99.85935302 |
| RWR | 4 | 10MB | 3.017597 | 30.71 | 34.128 | 8.84199777 | 89.98476324 |

**Memory Latency**

| Workload | Concurrency | Block Size | MyRAMBench Measured Latency (us) | pmbw Measured Latency (us), | Theoretical Latency (us) | MyRAMBench Efficiency (%) | pmbw Efficiency (%) |
|---|---|---|---|---|---|---|---|
| RWS | 1 | 1B | 0.000613001 | 5.78035E-05 | 0.004 | 84.67498713 | 98.55491329 |
| RWS | 2 | 1B | 0.00031734 | 3.35909E-05 | 0.004 | 92.06649924 | 99.16022842 |
| RWS | 4 | 1B | 0.000317417 | 3.78501E-05 | 0.004 | 92.06458035 | 99.05374716 |
| RWR | 1 | 1B | 0.000879323 | 5.78035E-05 | 0.004 | 78.016934 | 98.55491329 |
| RWR | 2 | 1B | 0.000506312 | 3.35909E-05 | 0.004 | 87.34218874 | 99.16022842 |
| RWR | 4 | 1B | 0.000484066 | 3.78501E-05 | 0.004 | 87.89834415 | 99.05374716 |

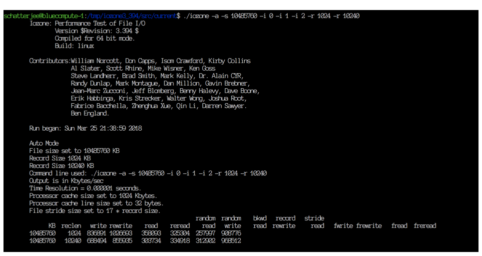
**Synopsis for Memory Benchmark**

- Benchmark works best 1MB Block Size and 2 cores. For block Size 10MB It's closer but not optimal.
- 10MB block , Single thread Random Access gives worst performance .
- Benchmark Provide better performance for Sequential memory access than Random Access. Sequential access is around 12% better than random for the best case ($\frac{3.26-2.92}{2.92}$ case 1MB Block , 2 thread )

- Generally a CPU fetches a batch of RAM positions at once. It then keeps those in a cache while it?s working on this. So if the data in RAM is sequential, the CPU has to wait on such fetching a lot less than if the data it needs to work on is scattered throughout RAM.
- It's not a Quad-core machine so using 4 threads actually hurts performance than 2 threads as a lot of time is spent in context switching.
- Interestingly sequential access for 10MB data 4 threads takes more time than Random Access 4 thread 10 MB . As these are first 2 operations done on the machine looks like cache is not warmed up , so this perticular test case results in anomaly.
- pmbw also gives optimal output for block size of 1MB , but instead of 2 threads , result of 4 thread is better . Which is a deviation from my code result.

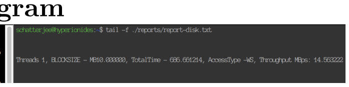
# 4 DISK Benchmarking (Run on Hyperion)

We were asked to Write and Read 10GB data. With various access patterns (RWS, RWR) and various block sizes (1MB, 10MB, 100MB) using 1 thread, 2 threads and 4 threads.
**IOZone benchmark**



corresponding results re kept in /cs553-pa1/disk/iozone.sh

**Running Disk Program**

**Disk benchmark Table**

| Work- load | Concurrency | Block Size | MyDiskBench Measured Throughput (MB/sec) | IOZone Measured Throughput (MB/sec) | Theoretical Throughput (MB/sec) | MyDiskBench Efficiency (%) | IOZone Efficiency (%) |
|---|---|---|---|---|---|---|---|
| RS | 1 | 1MB | 24.240029 | 35[illegible]93 | 1200 | 2.020002417 | 29.84108333 |
| RS | 1 | 10MB | 22.722776 | 303.734 | 1200 | 1.893564667 | 25.31116667 |
| RS | 1 | 100MB | 17.949336 | 341.998 | 1200 | 1.495778 | 28.49983333 |
| RS | 2 | 1MB | 26.432186 | 358.093 | 1200 | 2.202682167 | 29.84108333 |
| RS | 2 | 10MB | 30.5996 | 303.734 | 1200 | 2.549966667 | 25.31116667 |
| RS | 2 | 100MB | 31.167425 | 341.998 | 1200 | 2.597285417 | 28.49983333 |
| RS | 4 | 1MB | 43.434794 | 358.093 | 1200 | 3.619566167 | 29.84108333 |
| RS | 4 | 10MB | 50.746256 | 303.734 | 1200 | 4.228854667 | 25.31116667 |
| RS | 4 | 100MB | 38.324509 | 341.998 | 1200 | 3.193709083 | 28.49983333 |
| WS | 1 | 1MB | 43.196197 | 836.891 | 1200 | 3.599683083 | 69.74091667 |
| WS | 1 | 10MB | 36.822835 | 688.494 | 1200 | 3.068569583 | 57.3745 |
| WS | 1 | 100MB | 38.265487 | 546.319 | 1200 | 3.188790583 | 45.52658333 |
| WS | 2 | 1MB | 43.337197 | 836.891 | 1200 | 3.611433083 | 69.74091667 |
| WS | 2 | 10MB | 45.127668 | 688.494 | 1200 | 3.760639 | 57.3745 |
| WS | 2 | 100MB | 45.08773 | 546.319 | 1200 | 3.757310833 | 45.52658333 |
| WS | 4 | 1MB | 48.600181 | 836.891 | 1200 | 4.050015083 | 69.74091667 |
| WS | 4 | 10MB | 38.434767 | 688.494 | 1200 | 3.20289725 | 57.3745 |
| WS | 4 | 100MB | 38.099094 | 546.319 | 1200 | 3.1749245 | 45.52658333 |
| RR | 1 | 1MB | 11230.61736 | 257.997 | 1200 | 935.8847798 | 21.49975 |
| RR | 1 | 10MB | 3317.251299 | 312.902 | 1200 | 276.4376083 | 26.07516667 |
| RR | 1 | 100MB | 3015.231274 | 337.239 | 1200 | 251.2692728 | 28.10325 |
| RR | 2 | 1MB | 1019.43294 | 257.997 | 1200 | 84.952745 | 21.49975 |
| RR | 2 | 10MB | 382.129398 | 312.902 | 1200 | 31.8441165 | 26.07516667 |
| RR | 2 | 100MB | 106.100706 | 337.239 | 1200 | 8.8417255 | 28.10325 |
| RR | 4 | 1MB | 7148.759976 | 257.997 | 1200 | 595.729998 | 21.49975 |
| RR | 4 | 10MB | 5366.675414 | 312.902 | 1200 | 447.2229512 | 26.07516667 |
| RR | 4 | 100MB | 527.133318 | 337.239 | 1200 | 43.9277765 | 28.10325 |
| WR | 1 | 1MB | 232.433107 | 908.776 | 1200 | 19.36942558 | 75.73133333 |
| WR | 1 | 10MB | 850.706823 | 968.512 | 1200 | 70.89223525 | 80.70933333 |
| WR | 1 | 100MB | 841.7931621 | 851.317 | 1200 | 70.14943018 | 70.94308333 |
| WR | 2 | 1MB | 270.136348 | 908.776 | 1200 | 22.51136233 | 75.73133333 |
| WR | 2 | 10MB | 169.15893 | 968.512 | 1200 | 14.0965775 | 80.70933333 |
| WR | 2 | 100MB | 193.636534 | 851.317 | 1200 | 16.13637783 | 70.94308333 |
| WR | 4 | 1MB | 244.584262 | 908.776 | 1200 | 20.38202183 | 75.73133333 |
| WR | 4 | 10MB | 172.471967 | 968.512 | 1200 | 14.37266392 | 80.70933333 |
| WR | 4 | 100MB | 235.074646 | 851.317 | 1200 | 19.58955383 | 70.94308333 |

**Synopsis for Disk Benchmark**

theoretical throughput = one lane rate * 2 lane per port * 1 port per stack = 600 MB/sec * 2 * 1 = 1200 MB/sec.

- Divided all 36 different disk throughput operations in 9 groups and parallely run them on 9 nodes.
- Disk gives highest throughput when in Sequential Read , using 1 thread and read in 100 MB chunk.
- Creating Disk Write first followed by Read so no need to copy temp files .
- 

# 5 Network Benchmarking

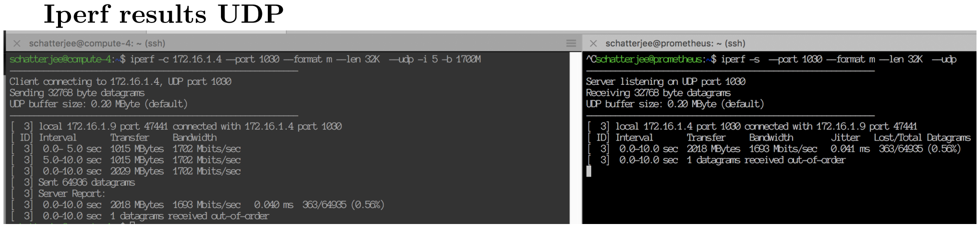
We were asked to implement Server-Client software using both TCP and UDP protocols with Workload: 1GB data. We operated over it 100X times with various block sizes (1KB, 32KB) with Concurrency: 1 thread, 2 threads, 4 threads and 8 threads.

**Theoretical network bandwidth calculation**

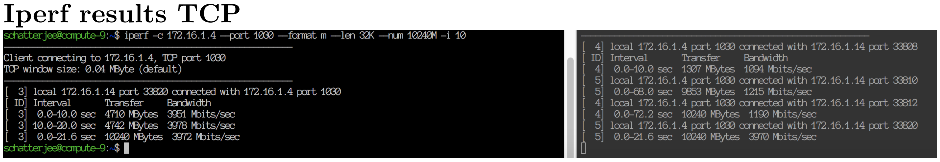
$$\text{Theoretical Throughput} = \frac{RWIN}{RTT}$$

Where RWIN is the TCP Receive Window and RTT is the round-trip time for the path. The Max TCP Window size in the absence of TCP window scale option is 65,535 bytes.

**Iperf results UDP**

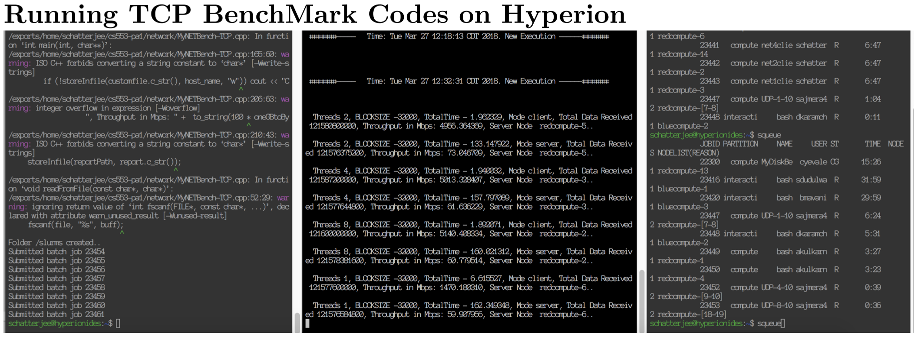



**Iperf results TCP**




Rest of the Iperf commands are detailed in iperf.sh

**Running TCP BenchMark Codes on Hyperion**



**Throughput calculation Table**

| Protocol | Concurrency | Block Size | MyNETBench Measured Throughput (Mb/sec) | iperf Measured Throughput (Mb/sec) | Theoretical Throughput (Mb/sec) | MyNETBench Efficiency (%) | iperf Efficiency (%) |
|---|---|---|---|---|---|---|---|
| TCP | 1 | 1KB | 1.2389 | 564 | 1323.939394 | 0.09357679 | 42.6001373 |
| TCP | 1 | 32KB | 1.6784 | 3626 | 1137.266811 | 0.1475819 | 318.834592 |
| TCP | 2 | 1KB | 1.2389 | 566 | 1323.939394 | 0.09357679 | 42.7512016 |
| TCP | 2 | 32KB | 1.6784 | 3666 | 1137.266811 | 0.1475819 | 322.351797 |
| TCP | 4 | 1KB | 1.2389 | 562 | 1323.939394 | 0.09357679 | 42.449073 |
| TCP | 4 | 32KB | 1.6784 | 3,524.00 | 1137.266811 | 0.1475819 | 309.865721 |
| TCP | 8 | 1KB | 1.2389 | 563 | 1323.939394 | 0.09357679 | 42.5246052 |
| TCP | 8 | 32KB | 1.6784 | 3585 | 1137.266811 | 0.1475819 | 315.229458 |
| UDP | 1 | 1KB | 0.9765 | 559 | 1323.939394 | 0.07375715 | 42.2224765 |
| UDP | 1 | 32KB | 0.9218 | 1693 | 1137.266811 | 0.08105398 | 148.865682 |
| UDP | 2 | 1KB | 0.9765 | 892 | 1323.939394 | 0.07375715 | 67.3746853 |
| UDP | 2 | 32KB | 0.9218 | 1761 | 1137.266811 | 0.08105398 | 154.84493 |
| UDP | 4 | 1KB | 0.9765 | 987 | 1323.939394 | 0.07375715 | 74.5502403 |
| UDP | 4 | 32KB | 0.9218 | 1198 | 1137.266811 | 0.08105398 | 105.340276 |
| UDP | 8 | 1KB | 0.9765 | 959 | 1323.939394 | 0.07375715 | 72.4353399 |
| UDP | 8 | 32KB | 0.9218 | 1399 | 1137.266811 | 0.08105398 | 123.014229 |

**Synopsis for bandwidth calculation**

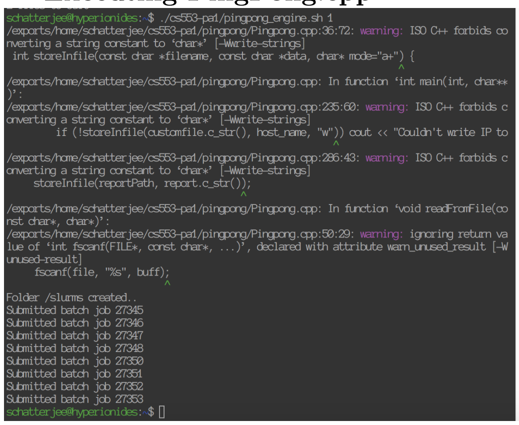
- Increasing No of thread does have significant +ve effect on Measured Throughput . Looks like 1 thread cant max out TCP buffer , If we use 8 threads then TCP throughput is maximal . For UDP it's 2 threads
- for TCP best result is obtained when block size is 32kb and thread count is 8.
- TCP is giving effectively higher Thoughput than UDP . Each frame goes through several buffers we send it: The application buffer, The Protocol Buffer, The Software interface buffer and the Hardware interface buffer. As we start stressing the stack by sending high speed data you will fill up these buffers and TCP is performing better as TCP is optimized for high speed bulk transfers.
- Throughput is dependent on Block size as higher block size yeilds in greater utilization of the available bandwidth and less no of iteration.

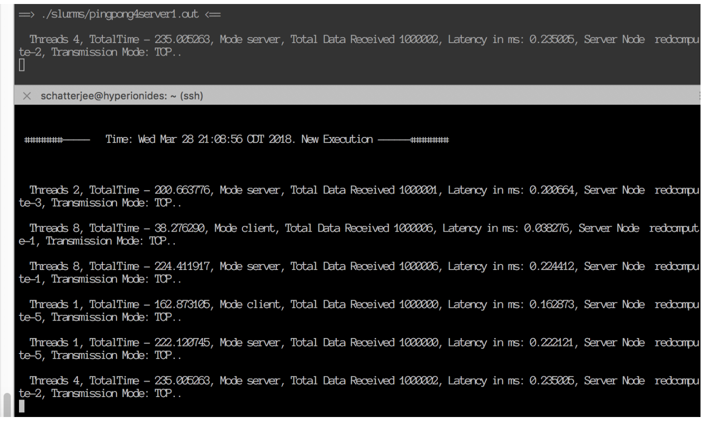
**Latency calculation using Pingpong.cpp and 'PING' utility tool**

Implemented similar Server-Client software to check latency, where server and client need to acknowledge the reciept of 1B data. And iterated it 1 million times.

Theoratical Latency: Ultimately response time over a network is limited by the speed of light. In a vacuum, light travels with a speed of 299 792 458 m / s. So, if we assume that a ?ping? travels with the speed of light, which are the best possible response times we can get ? Assuming machines are 10m apart . Latency should be $\frac{10*2*1000}{299792458} MiliSecond = 0.000066ms$

### Executing PingPong.cpp




### Latencty calculation Table

| Protocol | Concurrency | Message Size | MyNETBench Measured Latency (ms) | ping Measured Latency (ms) | Theoretical Latency (ms) | MyNETBench Efficiency (%) | Ping Efficiency |
|---|---|---|---|---|---|---|---|
| TCP | 1 | 1B | 0.186475 | 0.00396 | 0.000066 | 0.035393484 | 1.6666667 |
| TCP | 2 | 1B | 0.100287 | 0.00396 | 0.000066 | 0.065811122 | 1.6666667 |
| TCP | 4 | 1B | 0.168867 | 0.00396 | 0.000066 | 0.039084013 | 1.6666667 |
| TCP | 8 | 1B | 0.224412 | 0.00396 | 0.000066 | 0.029410192 | 1.6666667 |
| UDP | 1 | 1B | 0.051648 | 0.00396 | 0.000066 | 0.127788104 | 1.6666667 |
| UDP | 2 | 1B | 0.163044 | 0.00396 | 0.000066 | 0.04047987 | 1.6666667 |
| UDP | 4 | 1B | 0.107404 | 0.00396 | 0.000066 | 0.061450225 | 1.6666667 |
| UDP | 8 | 1B | 0.057742 | 0.00396 | 0.000066 | 0.114301548 | 1.6666667 |

### Synopsis for Latencty calculation

- UDP is faster than TCP, and the simple reason is because its nonexistent acknowledge packet (ACK) that permits a continuous packet stream, instead of TCP that acknowledges a set of packets, calculated by using the TCP window size and round-trip time (RTT).
- UDP latencies are less compared to TCP
- For TCP we get best latenct with 2 threads. And for UDP it's 1 thread.

## References


[1] Measuring network throughput
$https://en.wikipedia.org/wiki/Measuring_network_throughput$

[2] UDP Latency
$https://stackoverflow.com/questions/47903/udp-vs-tcp-how-much-faster-is-it$

[3] TCP Bandwidth
$https://serverfault.com/questions/432101/why-is-udp-slower-than-tcp-on-ubuntu-server$