**Design Document of Performance Benchmarking**

Sudipta Swarnakar

Sohan Samant

Benchmarking is the process of measuring some units against its standard. Here the benchmarking of following have been done.

1. CPU
2. GPU
3. Memory
4. Disk
5. Network

Programming languages used are

1. CPU – C++
2. GPU – CUDA C++
3. Memory – C++ and CUDA C++
4. Disk – C++
5. Network – Java

Test Bed –

1. Chameleon cloud KVM virtual machine m1.medium(2 virtual processors with 4GB RAM and 40GB disk)
2. Chameleon cloud Bare Metal (48 CPU Cores with Nvidia Tesla P100 as GPU Device which has 3584 CUDA Cores and CUDA 8 configured on the instance)

**CPU Benchmarking**

CPU benchmarking has done using C++ Programming language

The idea of calculating the FLOPS and IOPS, of a CPU is achieved by executing the number of instructions inside the loop, with certain amount of time. Now to achieve the parallelism Pthreads are used with number of threads varying from 1, 2, 4 and 8. This experiment is done with strong scaling and calculation of FLOPS and IOPS are done on many iteration of Floating point operations and Integer operations, GFLOPS are calculated from this result of FLOPS and IOPS.

We have used Linpack to test our benchmarking.

**GPU**

GPU benchmarking has done using CUDA C++ Programming language

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

**Memory Benchmarking**

Memory has been implemented in C ++ Programming language.

The CPU’s memory part has been tested on OpenStack KVM Cloud which has 2 devices each of having 2 CPU cores.

We are going to perform reading and writing operation. To be more precise we are going to perform Random Read, Random Write, Sequential Read and Sequential Write Operations. Here the block size of memory that we will consider is 8B, 8KB, 8MB and 80MB and combination of different threads (1,2,4,8) . For this benchmark we will be calculating the throughput and latency. Here we have implemented strong scaling. This means we have fixed size of Memory which is 1GB from which we will be reading and writing. Then we will vary the block size and threads to get the performance or desired result.

We also have performed operation on GPU device to find out the GPU memory bandwidth using CUDA C++. This part of the operations has been tested on BareMetal Instance with CUDA8 which has Nvidia Tesla P100 as GPU device.

To compare our result, we have used stream benchmarking tool.

**Disk Benchmarking**

Disk Benchmarking has been implemented in C++ programming language.

The Disk benchmarking has been tested on BareMetal Instance which has CentOS7 as a distro.

The reason that this benchmarking has been done on Baremetal is because we couldn’t get any floating point IPs through the day and we had to test our code at any cost so we chose to do it on Baremetal Instannce.

Here we are going to perform reading and writing operation to desk how fast is the disk. To be more precise we are going to perform Random Read, Random Write, Sequential Read and Sequential Write Operations. Here the block size of memory that we will consider is 8B, 8KB, 8MB and 80MB for varying number of threads of 1,2,4 and 8. For this benchmark we will be calculating the throughput and latency. Here we have implemented strong scaling. This means we have fixed file size of 1.2GB on the disk from which we will be reading and writing. Then we will vary the block size and threads to get the performance or desired result.

To compare our result, we have used IOZONE benchmarking tool.

**Networking Benchmarking**

Network Benchmarking has been implemented in Java programming language.

In this benchmarking we are testing the network bandwidth and latency based on two main Protocols – TCP and UDP. Here we are transferring 64 KB of packet at one time from client to serve and then change the thread counts as you fill in. Here we have created a multi-threaded environment both at client and server side. First, we are testing for TCP Protocol using a fixed size of packet and then sending packets according to threads. So here the client is sending packets to server and then gets an ack from server. For UDP being a connectionless protocol there is no guarantee of message delivery. The client sends the data to the server but there might be drop in the packets and then server sends an acknowledgement. Then we measure the latency and throughput for both UDP and TCP.

To compare our result we have used IPERF benchmarking tool. Here we are testing for TCP first and then we are testing for UDP by giving different bandwidth, because by default IPERF limits the bandwidth for UDP clients to 1Mbits per second by default.

**References:**

<http://www.geeksforgeeks.org/working-udp-datagramsockets-java/>

<http://www.mysamplecode.com/2011/12/java-multithreaded-socket-server.html>

<https://www.tutorialspoint.com/javaexamples/net_multisoc.htm>

<http://net-informations.com/java/net/multithreaded.htm>

https://developer.nvidia.com/computeworks

<https://stackoverflow.com/questions/36196829/fp16-support-in-cuda-thrust>

<http://www.cs.virginia.edu/stream/ref.html>

<http://www.thegeekstuff.com/2011/05/iozone-examples/13/>

<https://stackoverflow.com/questions/29747404/my-program-was-killed>

<https://devblogs.nvidia.com/parallelforall/>

<https://www.tutorialspoint.com/cplusplus/cpp_multithreading.htm>

<http://www.bogotobogo.com/cplusplus/multithreading_pthread.php>

<http://docs.nvidia.com/cuda/cuda-samples/index.html>

<https://devblogs.nvidia.com/parallelforall/easy-introduction-cuda-c-and-c/>

<https://www.karlrupp.net/2016/01/gpu-memory-bandwidth-vs-thread-blocks-cuda-workgroups-opencl/>

<https://devblogs.nvidia.com/parallelforall/how-optimize-data-transfers-cuda-cc/>

<http://www.novatte.com/our-blog/197-how-to-calculate-peak-theoretical-performance-of-a-cpu-based-hpc-system>

https://stackoverflow.com/questions/31227418/c-calculate-gflops