CS553 – Cloud Computing

Programming Assignment 1

Design Document

Karan Jeet Singh; Kanwal Preet Singh
9-23-2014

Design Document

1 OBJECTIVE

This document aims at giving an overview of how benchmarking of different components is being performed, how the programs work, what short comings were anticipated, and how we can improve those systems.

2 CPU

The CPU benchmarking is done in respect to calculating integral operations, and float point operations done by CPU per second.

2.1 DESIGN OVERVIEW:

This program is prepared with the help of CDT Eclipse using C++ compiler.

In this program we calculate the IOPS and FLOPS for the CPU. For both the IOPS and FLOPS there 100,000,000 for single thread, Max. No. of threads it can run for is 16

- For IOPS, for every single iterations 10 operations take place. In case of multithreading the as the no. of threads increase the no. of operations also increases, i.e. if threads are 2 then operations will be 200,000,000, and If threads are 4 then operations will be 400,000,000, so on so forth. The operations involve addition, subtraction and multiplication of various integer in parallel. In the MeasureIOPs() method the for every iteration 10 integral operations takes place. Before running the method which performs the benchmarking, timestamp is noted and the same is done after the method has been completely executed. When all the iterations are complete, we calculate the total no. of operation performed for a given period of time. From these we can get the IOP per second of the CPU.
- For FLOPS, for every single iterations 7 operations take place. In case of multithreading the as the no. of threads increase the no. of operations also increases, i.e. if threads are 2 then operations will be 200,000,000, and if threads are 4 then operations will be 400,000,000, so on so forth. The operations involve addition, subtraction and multiplication of various integer in parallel. In the MeasureFLOPs() method the for every iteration 7 floating operations takes place. Before running the method which performs the benchmarking, timestamp is noted and the same is done after the method has been completely executed. When all the iterations are complete, we calculate the total no. of operation performed for a given period of time. From these we can get the FLOPS per second of the CPU.

2.2 TRADEOFFS

- Using multiple variable to perform same set of operation per iterations gives degraded performance as compared to using single variable with same set of operations as the compiler is unable to handle multiple variable for billions of iterations.
- Unable to use Register as registers has been deprecated in C++11 (Even if we were able use register key word that would not have guaranteed the results as it is mere hint to the compiler rather than a forced command to the compiler)
- Decreasing the number of iterations or number of operations resulted in inconsistent results (As the execution time becomes very less causing the results to be inconsistent)
- If we use the assembly level languages for the same code then it will give better result as it is more closely bound to the hardware which would result in less dependency on compiler related issues.

2.3 POSSIBLE IMPROVEMENTS AND EXTENSIONS

- If the same logic is implemented in any assembly level language then it will give better results.
- The program can be further extended to solve linear equations and matrix problem to utilize the CPU more effectively.

3 GPU

In this program we calculate the IOPS and FLOPS for the GPU in two different experiments

- Using only a single thread
- Measuring the processor speed with full concurrency i.e. 1 thread per core

In the first case for both the IOPS and FLOPS there are 10240 iterations for single thread, for IOPS there are 9 IOP for every single iteration.

For calculation of the IOPS the calculation are performed on the device using the below mention kernel method:

vectorADD << <1, 1 >> >(d_a, d_b, d_c)

The timestamp for the operation is taken before the operation and as well as after the operations and the IOPS are calculated using the values of the timestamp and the iterations, for FLOPS there is only 1 FLOP for every single iteration.

For calculation of the FLOPS the calculation are performed on the device using the below mention kernel method:

vectorFADD << <1, 1 >> >(d_a, d_b, d_c)

The timestamp for the operation is taken before the operation and as well as after the operations and the FLOPS are calculated using the values of the timestamp and the iterations.

For Maximum concurrency, for both the IOPS and FLOPS there are 10240 iterations. For IOPS there are 9 IOP for every single iterations and for FLOPs there are 6 operations. Block size and threads per block are mentioned in the kernel call to achieve max concurrency

For calculation of the IOPS, the calculation are performed on the device using the below mention kernel method

vectorADD << <N, M >> >(d_a, d_b, d_c)

The timestamp for the operation is taken before the operation and as well as after the operations, and the IOPS are calculated using the values of the timestamp and the iterations

For calculation of the FLOPS, the calculation are performed on the device using the below mention kernel method

vectorADD << <N, M >> >(d_a, d_b, d_c)

The timestamp for the operation is taken before the operation and as well as after the operations, and the FLOPS are calculated using the values of the timestamp and the iterations.

3.1 TRADEOFFS

- The threads created were more than the max.no. Of cores on the machine to get more stable results.
- Decreasing the number of iterations or number of operations resulted in inconsistent results. (As the execution time becomes very less causing the results to be inconsistent)

3.2 POSSIBLE IMPROVEMENTS AND EXTENSIONS

- If the same logic is implemented in any OpenCL this program would support a wide range of hardware as CUDA restricts it to NVidia devices.
- By hit and trail method the program can be optimized for specific devices

4 MEMORY

The memory benchmarking aims at calculating the latency and throughput of writing and reading data of variable size to the memory, either sequentially or randomly.

4.1 DESIGN OVERVIEW

The program works by reading an array of characters and then writing it to the memory, and then we calculate the time taken to do so and hence, calculate the latency and throughput.

- The memcpy method is used to do the reading and writing for us, this method takes in the source stream, destination stream, and the block size as its parameter.
- The block size can be specified to the memcpy, ranging from 1B, 1KB, to 1MB.
- The block size specifies the size of data that is picked up from the source stream and sent to the destination stream. The data can either be picked up sequentially, and as well as randomly.

- The program is also capable of handling multiple threads to do the same task simultaneously, this gives the advantage of using up all the cores of a multi-core CPU to do the reading and writing.

4.2 Tradeoffs

- The program is hardcoded to be optimized for a system with 1GB RAM only. So, even though the program would give more accurate results if it is able to use whole of available memory, it is not able to do so.
- Multiple executions of the test explicitly would give different result than running multiple tests through code as some time is wasted during initialization of test.

4.3 Possible Improvements and Extensions

- The program would give much better results if the test is ran multiple times, approximately 10 times, within the code, rather than executing the program 10 times.
- The program only reads and writes character arrays, but using arrays of bigger data types would give much better result. This can be achieved by changing the data type used depending upon the block size given.

5 Disk

The disk benchmarking aims at calculating latency and benchmark of the disk by writing and reading files to the disk.

5.1 Design Overview

The program works by writing some data into a file and then reading it. The time taken to do so can be used to calculate the latency and the throughput.

- The program use the fstream methods to handle the input and output operations to the file, and the read/write methods are used to read and write the data. The read and write tasks can be performed separately upon user's control
- The data is written in blocks, and this block size is user controlled, ranging from 1B, 1KB, to 1MB.
- The data being written to and read from a file can either be selected sequentially, as well as, randomly.
- The program also allows user to specify number of threads, which can be used to read and write to multiple files separately and independently.

5.2 Tradeoffs

- The program is optimized for a system with space less than 2 GB, hence when writing a file with 1 MB blocks, the number of iterations of writing the block are not large enough to be able to perfectly predict the speed of disk.

5.3 Possible Improvements and Extensions

- The program can be optimized if the space available is more than 2GB.
- The program can be made to detect the available free space and then create the files accordingly.

6 Network

Network benchmarking is done by sending a block of data through a call back loop and then noting the time taken to do so.

6.1 Design Overview

The program creates a server and client system on the same machine and sends data packets via call back loop, a timer is ran to calculate the time taken to do so and hence we calculate the latency and throughput of doing so.

- The program can create both server and client depending upon the arguments passed.
- The server and client are both given a port to send and receive data through.
- The client can only send data once server is up and running.
- The program can take in different size of data packets, which are sent through the network.
- The program can also handle multiple threads as specified by the user.

6.2 Tradeoffs

- The program measures the speed of callback loop in a unidirectional route, rather than a bidirectional route
- The timer stops when an acknowledgment is received from server, i.e. when the latency is calculated, the time noted is a little bit more than the time taken to send the data.

6.3 Possible Improvements and Extensions

- If the server was to note down the time at which it receives the data, then that time can be used with the start time of the client to know the exact time it took to transfer the data without the acknowledgement overheads.