CS 553 Cloud Computing
Programming Assignment 1
Name: Kedar Kaushikkar (A20355218)

Design Document

The document includes the design and architecture for the below benchmarking modules,

- CPU
- DISK
- MEMORY

CPU Benchmarking

Language: C

The basic aim of the program is to benchmark the CPU Performance on the basis of Giga FLOPS and Giga IOPS.

Design:

The program performs several arithmetic operations on Integer constants and floating point constants to calculate the Integer operations per cycle and floating point operations per cycle.

The number of operations for Integer and Floating Point constants performed per second are stored in a structured array and the time taken to perform those operations are also stored in one structured time array.

Various functions are defined in the program to calculate the Giga FLOPS and Giga IOPS using the below steps,

- To calculate the IOPS and FLOPS, the program is designed such that 30 arithmetic operations are performed for 10^9 times and the time required to perform those operations is recorded for each concurrency level.
- A Timer is maintained at the start of the operation and at the end of the operation and the time difference is recorded.
- If concurrency level is 1, then the total time elapsed is the time required to perform the operations.
- If the concurrency level is 2, then the total time of both the threads is recorded but the maximum time is used for the calculation.
  Ex: if using two threads running concurrently and thread_1 took x secs and thread_2 took y secs. Then the total time value is the one which is greater as the total time for both the threads. This is the time when the processor executed all the instructions from both the threads.
- The IOPS and FLOPS is calculated by the number of times all the operations performed times the total number of arithmetic operations and the product is divided by total time elapsed to perform those operations.
- The program will output the number of GFLOPS and GIOPS for 1,2 and 4 levels of concurrency.

CPU IOPS & FLOPS – 600 Samples

As another experiment, the above benchmark is ran on floating point and Integer instructions at concurrency level of 4 for 10 mins. The aim of this benchmark experiment is get 600 samples each second and calculate number of instructions achieved in each second.

Design:

To obtain the number of instructions per second this experiment uses the same design and logic implemented in the main CPU Benchmarking experiment described previously.

- Using the calculations for previous experiment at concurrency level 4, Time taken for executing all instructions is achieved.
- Based upon this time the total number of instructions(x) that can be ran in 10 minutes is calculated.
- This experiment of calculating GFLOPS and GIOPS is ran for those x number of instructions thus achieving 600 samples by populating number of instructions per second.
- The calculation for this experiment is explained in the Evaluation report and the data chart for those 600 samples is drawn.

Improvements & Extensions

- The accuracy of the benchmarking can be increased if the usage of system cache is bypassed and also the operations need to be dynamic each time so that the system cannot serve the result from the cache memory.

Memory Benchmarking

Language: C

This benchmarking aims to measure the performance of the memory by performing memory operations like read and write and accessing the memory sequentially and randomly at different concurrency levels.

Design:

The program performs several memory operations to fetch and write data of various block sizes to the memory.

Accessing memory sequentially:

- Benchmarking on sequentially accessed memory is done using 'memmove' which takes reading a block of memory and writing it to different location of the memory.
- This operation is performed for several number of instructions and the time to perform those total instructions is recorded.
- To perform this memory operation for many number of times, two strings are used both having size greater the block size accessed at a time (1B ,1Kb, 1MB).
- Then the memory operations (memmove) is performed on those strings and sequentially each block is read and written to the other string.
- The time for completing all the instructions is recorded.
- Based on this time the throughput for the memory is calculated based on the formula

Throughput = no of times instructions performed * total instructions

---

Total Time Elapsed

Latency = 1/ Throughput

- At different concurrency level, throughput & Latency of each thread is calculated and average of the total throughput based on concurrency level is calculated.

Accessing memory randomly:

- Benchmarking on randomly accessed memory is done using 'memmove' which takes reading a block of memory and writing it to different location of the memory.
- This operation is performed for several number of instructions and the time to perform those total instructions is recorded.
- To perform this memory operation for many number of times, two strings are used both having size greater the block size accessed at a time (1B ,1Kb, 1MB).
- Then the memory operations (memmove) is performed on those strings and randomly each block is read and written to the other string.
- For randomly accessing any block, the program moves the file pointer randomly within the file and then performs the memory operation on its location.
- The time for completing all the instructions is recorded.
- Based on this time the throughput for the memory is calculated based on the formula
  Throughput = no of times instructions performed * total instructions

  ---

  Total Time Elapsed

  Latency = 1/Throughput
- At different concurrency level, throughput & Latency of each thread is calculated and average of the total throughput based on concurrency level is calculated.

Improvements & Extensions

- The accuracy of the benchmarking can be increased by handling the cache memory of the system. The memory block accessed randomly should always be different and should not be accessed from the system cache if already accessed earlier.
- Also adjusting the sampling frequency rather than checking system performance on periodic times may give accuracy in range of throughput and latency.

DISK Benchmarking

Language: C

This benchmarking aims to measure the performance of the disk by performing file operations like read and write and accessing the file on the disk sequentially and randomly at different concurrency levels.

Design:

- This program performs several operations on the disk using file read and file write and accessing the file sequentially and randomly for various block sizes.
- The program creates a text file of 40 Mb which contains random characters.
- To perform file operations on the disk sequentially, reading and writing the files for certain block size is achieved by creating a readbuffer and writebuffer having size equal to size of the block to be accessed at a time (1B, 1Kb, 1Mb).

- The read buffer reads single characters from the file and stores it in itself. The writebuffer takes data from the file and writes in to the writebuffer.
- For Block sizes of 1B ,1KB, 1Mb, the total number of operations performed are 4000000, 40000 ,40 so that each time a text file of 40Mb is read completely.
- For accessing the file sequentially, the file pointer is set to start and is incremented sequentially by block size. Random access of file is done by generating a random number and indexing the file pointer to that number and accessing block from that file pointer location.
- The benchmarking results are evaluated over the throughput and latency parameters and calculated using the below formula

  Throughput = Number of instructions * Num of Bytes
  ---------- Mb/Sec
  Total Time Elapsed * 1024 * 1024

  Latency = 1/Throughput

Improvements & Extensions

- The accuracy and exact behavior of the benchmarks can be studied by running the benchmarks over different environments like virtual machine.
- Also accessing corrupt or bad memory blocks may affect the throughput and latency results.
- Also the rate of data transfer is limited to particular cores i.e. results for benchmarking on single core may vary from working on multiple cores. Thus extending the benchmarking to multiple cores may give more accurate results.

REFERENCES

https://www.microway.com/knowledge-center-articles/detailed-specifications-intel-xeon-e5-2600v3-haswell-ep-processors/

http://en.wikipedia.org/wiki/LINPACK

http://www.cs.virginia.edu/stream/

http://www.iozone.org/

http://cpuboss.com/cpus/Intel-Xeon-E5-2670-v3-vs-Intel-Xeon-E5-2670-v2