CS 553 PROGRAMMING ASSIGNMENT-1

EVALUATION DOCUMENT

ROHIT SINGH

A20361198

1. CPU Benchmarking

In this benchmarking I have measured the speed of the Amazon EC-2 CPU, through GFLOPS (Giga Floating Operation Per Second) and GIOPS (Giga Integer Operation Per Second) and later the efficiency of the CPU compared the theoretical value of the CPU.

I have used *gettimeofday()* function to find the time required by the threads to do the fixed number of operations. I have a loop for calling 3 cases of threads, i.e., 1,2 and 4. After each thread is created I call the respective function by *pthread_join().* After the work assigned to the threads have been done, the control returns to the main function, where I calculate the GFLOPS and the GIOPS. The GFLOPS and GIOPS are directly dependent on the Number of Time the loop is called (which is *INT_MAX* in my case), the number of instructions each iteration of the loop has to do and the number of threads doing the job. Then I have divided the result obtained by time taken and 10^9 (to convert it into Giga).

The functions iops() and flops() are responsible for running the loop with integer and floating point instructions for *INT_MAX* times.

I have also done a sampling of the two operation (GFLOPS and GIOPS) every second for 10mins, for 4 threads. This helped in giving a better insight to the CPU performance. I called the 4 threads along with a start timer to do a fixed number of instructions for a fixed number of Loops (i.e., *INT_MAX*). However, when the timer hits 1 second I store the number of instructions completed on an array and then reset the start time. This continues for 600 such operations. After the 20mins operation (10 for IOPS and 10 for FLOPS), I save the result in two files namely *Sample IOPS.txt* and *Sample FLOPS.txt*. The functions *iopsfor10()* & *flopsfor10()* are responsible for creating 4 threads and then calling *iopscal()* & *flopscal()* respectively, where the actual computaion is done.

I have also implemented the LINPACK Benchmark, which is a well know benchmark for studying the CPU performance.

One improvement which I feel can help is to tune the LINPACK benchmark such that a 90% efficiency can be achieved compared to the theoretical value

2. Memory Benchmarking

In this benchmark I have measured the Latency and the Throughput of the memory for read+write operations, both for Sequential and Random scenarios, for 3 different block sizes of 1B,1KB,1MB. The result was then compared with the theoretical value.

I used memcpy() function for doing the read and write operations on the memory, and malloc() to reserve a fixed amount of memory. During my execution I have taken care that the latency I got was for the memory and not the cache. I reserved a 10MB space by malloc() so that the memory is bigger than the cache and the memory has to be refreshed every time.

The memory benchmark also look into two scenarios Sequential and Random memory accesses. The Sequential access was more of reading and writing the memory space one after another according to the block-size. Whereas the Random access was to read and write the same

fixed memory for a certain block-size randomly. I used a rand() to find the offset (i.e., the position from where the memory needs to be read).

I have used *gettimeofday()* function to find the time required by the threads to do the fixed number of operations. I have a loop for calling 3 cases of threads, i.e., 1,2 and 4. After each thread is created I call the respective function by *pthread_join().* After the work assigned to the threads have been done, the control returns to the main function, where I calculated the Latency and the Throughput. The latency is the execution time in millisecond and the throughput is memory transfered per execution time in Mbps.

I have used two functions *random_access()* & *sequential_access()* to do the above mentioned memcpy() operation. To prevent the program from getting a segmentation error, I have used (rand())%(memory-k), where memory is 10Mb and k is the block-size. This make sure that the memcpy() never goes out of the 10Mb space.

I have also implemented the STREAM Benchmark, which is a well know benchmark for studying the Memory performance.

One improvement which I feel is to measure the page hit and miss rate of memory; and later provide an average access time for the memory. Tune the STREAM Benchmark so that we get a 90% efficiency over the theoretical value.

3. Disk Benchmarking

In this benchmark I have measured the Latency and the Throughput of the disk for read and write operations, both for Sequential and Random scenarios. The result was then compared with the theoretical value.

During my execution I have taken care that the latency I got was for the disk and not the memory. I flushed the file pointer after every execution. I initially wrote a reserved a 10MB file so and then reused it for reading for both the scenarios.

The disk benchmark looks into four scenarios Read Sequential, Write Sequential, Read Random and Write Random. The Sequential access was more of reading or writing the file one after another according to the blocksize. Whereas the Random access was to read or write the same file for a certain blocksize randomly. I used a *rand()* to find the offset (i.e., the position from where the memory needs to be read), and then *fseek()* to set the offset position of the file pointer.

I have used *gettimeofday()* function to find the time required by the threads to do the fixed number of operations. I have a loop for calling 3 cases of threads, i.e., 1,2 and 4. After each thread is created I call the respective function by *pthread_join().* After the work assigned to the threads have been done, the control returns to the main function, where I calculated the Latency and the Throughput. The latency is the execution time in millisecond and the throughput is memory transfered per execution time in Mbps.

I have used four functions r*andom_write(), random_read(), sequential_write()* and *sequential_read()* to do the above mentioned file operations. I have first done the write operation and then used the File.txt created to do the read operation for both scenarios.

I have also implemented the IOZONE Benchmark, which is a well know benchmark for studying the Disk performance.

One improvement which I feel is to measure the seek time, the rotational speed of the disk, and find the average access time for the disk. Tune the IOZONE Benchmark so that we get a 90% efficiency over the theoretical value.