Benchmarking Major Components of Computer System

Sandeep Palur Swathi Rajasurya Manoj Meenakshi Dattatreya Babu
Department of Computer Science, Illinois Institute of Technology, Chicago IL, USA
psandeep@hawk.iit.edu srajasur@hawk.iit.edu mmeenaks@hawk.iit.edu

Abstract - A benchmark is the act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object, normally by running a number of standard tests and trials against it. In computer system, we run benchmarks to understand the performance of components and how each of the components behaves under different loads. As a result, we can program on the computer system by using the maximum performance out of it. In this work, we have run many benchmarking experiments on various components of computer system: CPU, GPU, Memory, Disk, and Network and analyzed the performance of these components under different workloads. We have also found the optimal workloads under which these components perform efficiently.

I. INTRODUCTION

1.1 CPU

A central processing unit (CPU) is the hardware within a computer that carries out the instructions of a computer program by performing the basic arithmetical, logical, and input/output operations of the system. A computer can have more than one CPU; this is called multiprocessing. All modern CPUs are microprocessors, meaning contained on a single chip. Some integrated circuits (ICs) can contain multiple CPUs on a single chip; those ICs are called multi-core processors. An IC containing a CPU can also contain peripheral devices, and other components of a computer system; this is called a system on a chip (SoC). Two typical components of a CPU are the arithmetic logic unit (ALU), which performs arithmetic and logical operations, and the control unit (CU), which extracts instructions from memory and decodes and executes them, calling on the ALU when necessary.

1.2 GPU

A graphics processing unit (GPU), also occasionally called visual processing unit (VPU), is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display. GPU-accelerated computing offers unprecedented application performance by offloading compute-intensive portions of the application to the GPU, while the remainder of the code still runs on the CPU. From a user's perspective, applications simply run significantly faster.

1.3 CPU VERSUS GPU

A simple way to understand the difference between a CPU and GPU is to compare how they process tasks. A CPU consists of a few cores optimized for sequential serial processing while a GPU has a massively parallel architecture consisting of thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously.

1.4 MEMORY

Random-access memory is a form of computer data storage. A random-access memory device allows data items to be read and written in roughly the same amount of time regardless of the order in which data items are accessed. RAM is considered "random access" because you can access any memory cell directly if you know the row and column that intersect at that cell. Similar to a microprocessor, a memory chip is an integrated circuit (IC) made of millions of transistors and capacitors. In the most common form of computer memory, dynamic random access memory (DRAM), a transistor and a capacitor are paired to create a memory cell, which represents a single bit of data. The capacitor holds the bit of information -- a 0 or a 1 (see How Bits and Bytes Work for information on bits). The transistor acts as a switch that lets

the control circuitry on the memory chip read the capacitor or change its state. There are two different types of RAM: DRAM (Dynamic Random Access Memory) and SRAM (Static Random Access Memory).

The two types of RAM differ in the technology they use to hold data, with DRAM being the more common type. In terms of speed, SRAM is faster. DRAM needs to be refreshed thousands of times per second while SRAM does not need to be refreshed, which is what makes it faster than DRAM.

1.5 DISK

There are basically two different types of storage that is widely used: SSD and HDD.

The traditional spinning hard drive (HDD) is the basic nonvolatile storage on a computer. That is, it doesn't "go away" like the data on the system memory when you turn the system off. Hard drives are essentially metal platters with a magnetic coating. That coating stores your data, whether that data consists of weather reports from the last century, a high-definition copy of the Star Wars trilogy, or your digital music collection. A read/write head on an arm accesses the data while the platters are spinning in a hard drive enclosure.

An SSD does much the same job functionally (saving your data while the system is off, booting your system, etc.) as an HDD, but instead of a magnetic coating on top of platters, the data is stored on interconnected flash memory chips that retain the data even when there's no power present. The chips can either be permanently installed on the system's motherboard (like on some small laptops and ultrabooks), on a PCI/PCIe card (in some high-end workstations), or in a box that's sized, shaped, and wired to slot in for a laptop or desktop's hard drive (common on everything else). These flash memory chips differ from the flash memory in USB thumb drives in the type and speed of the memory. That's the subject of a totally separate technical treatise, but suffice it to say that the flash memory in SSDs is faster and more reliable than the flash memory in USB thumb drives. SSDs are consequently more expensive than USB thumb drives for the same capacities.

1.6 NETWORK

A computer network or data network is a telecommunications network that allows computers to exchange data. In computer networks, networked computing devices pass data to each other along data connections. Data is transferred in the form of packets. The connections (network links) between nodes are established using either cable media or wireless media. The best-known computer network is the Internet. Computer networks support applications such as access to the World Wide Web, shared use of application and storage servers, printers, and fax machines, and use of email and instant messaging applications. Computer networks differ in the physical media used to transmit their signals, the communications protocols to organize network traffic, the network's size, topology and organizational intent.

II. EXPERIMENTS & RESULTS

This section is going to talk about the various benchmarking experiments ran on the above described components. We also discuss the results of the experiments with cleanly plotted graphs. We have done all our benchmark programs in C, since it is close to the hardware.

1. CPU Benchmarking Experiment- In this experiment wrote a multithreaded program that executes a lot of arithmetic operations. We ran 30 billion arithmetic operations and we were able to achieve 9.2 GFLOPS and 9.5IOPS with one thread and at the most 25.545 GFLOPS and 26.7 IOPS with 64 threads. The theoretical peak was 40GFLOPS. We were able to achieve close to 60% of the theoretical peak. We were able to achieve this performance because of neglecting memory writes in arithmetic operations. We made the processor only perform the arithmetic operation without writing back the result to the memory. The optimal number of threads was 4. The performance did not scale properly beyond

4. There was a very minor increase of performance with increase in threads beyond 4.

The reason why it did not scale beyond 4 threads is the processor. It had only 2 cores with hyper threading enabled. As we increase the threads beyond 4, the performance dropped because of context switching.

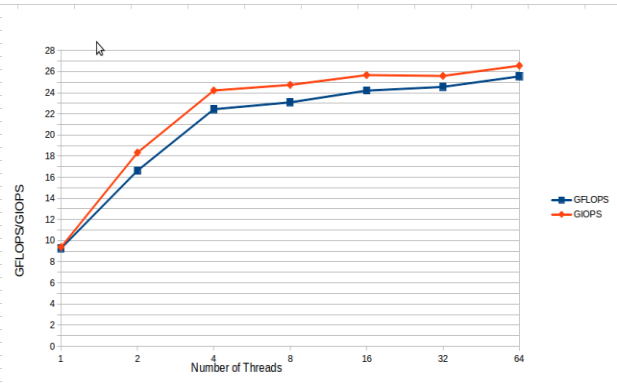

Figure1: Number of Threads versus GFLOPS/IOPS

Test Bed

Processor: Intel® Core™ i3 CPU M 330 @ 2.13GHz × 4
Memory: 4.0GB
OS type: 64 bit

2. DISK Benchmarking Experiment- In this experiment we developed a multithreaded program that does a lot of random and sequential reads to the disk with varying concurrency levels. We ran 1 billion sequential and random operations with varying block size (1B, 1KB and 1MB). The results of the experiments showed that the sequential reads are faster than random reads. Similarly, sequential writes are faster than random writes. As seen from the figure2 which shows the throughput for 1B block size, the sequential reads and writes are faster than random reads and writes. The sequential operations throughput was scaling properly but the random operations throughput was not scaling for more than 2 threads. One KB block size seemed to be good the random and sequential throughput was increasing with the number of threads. **The optimal number of threads for 1KB block size is 4** and for the **block size of 1MB the optimal number of threads were 2**. The latency for 1B block size is **0.000016ms** and the latency for 1KB and 1MB block size is 0.0000. May be the latency is in range of nano seconds.

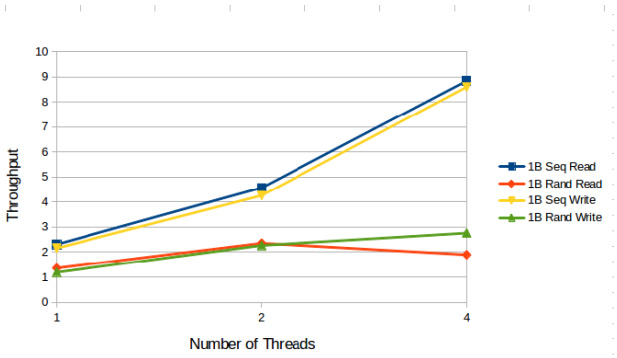

Figure2: Number of threads versus Throughput (MB/s) for 1B block size.

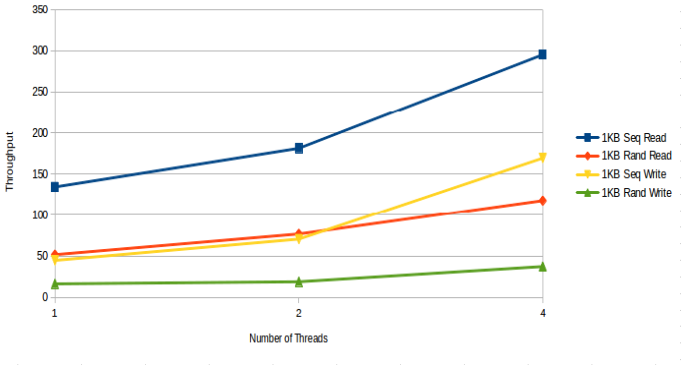

Figure3: Number of threads versus Throughput (MB/s) for 1KB block size.

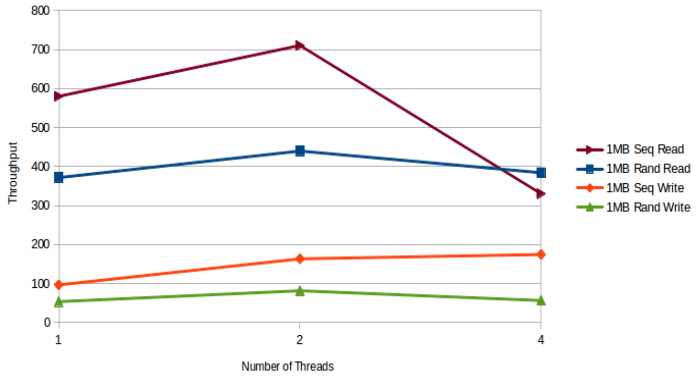

Figure4: Number of threads versus Throughput (MB/s) for 1MB block size

Test Bed
Processor: AMD FX-8120 Eight-Core x 8
Memory: 15.7GiB
OS type: 64 bit
Disk: 957GB (HDD)

3. GPU Benchmarking Experiment- In this experiment, we have used Jarvis cluster that has the configuration detailed in the test bed section. We used CUDA C for this experiment. The program works as follows: we do a device query and find out the available number of SM, SP, grid dimensions, block dimensions, maximum number of threads per block. Having all these information we dynamically decide the number of blocks and threads. In that way we were able to maximize the utilization of the cores by mapping one thread per core. We ran 1.1 billion floating point and arithmetic operations and we were able to achieve **460.4 FLOPS and 459.4 IOPS.** The theoretical peak was 1.1 PFLOPS. We were able to achieve close to 40% of the theoretical peak. For the memory bandwidth tests we used *cudamemcopy* to read and write 2 GB of memory in blocks of 1B, 1 KB and 1MB.

We got obvious results for this experiment. The read/write operations were faster when the block size was 1MB and better for 1KB and bad for 1B. The results we got are 20% close to the theoretical peak because we ran our tests with 2 GB memory. We found out that we would get high throughput if we had ran our tests on 30 to 40 GB of memory. So that we would have ended up getting around 60% of theoretical peak. The latency for 1B was **0.000020ms** and the latency for 1KB and 1MB were negligible. The values were 0.0000ms and 0.0000ms. May be we can find out the latency if we try to calculate it in nano seconds.

Test Bed
Major revision number: 2
Minor revision number: 0
Name: GeForce GTX 480
Total global memory: 1609760768
Total shared memory per block: 49152
Total registers per block: 32768
Warp size: 32
Maximum memory pitch: 2147483647
Maximum threads per block: 1024
Maximum dimension 0 of block: 1024
Maximum dimension 1 of block: 1024
Maximum dimension 2 of block: 64
Maximum dimension 0 of grid: 65535
Maximum dimension 1 of grid: 65535
Maximum dimension 2 of grid: 65535
Clock rate: 1401000
Total constant memory: 65536
Texture alignment: 512
Concurrent copy and execution: Yes
Number of multiprocessors: 15
Kernel execution timeout: No

Throughput(GB)	Block size(Bytes)
0.05679	1
0.20776	1024
3.76	1048576

Figure5: Throughput versus Block Size

4. Memory Benchmarking Experiment- In this experiment, we have developed a multithreaded program that does a lot of sequential and random read/write operations on memory with varying block sizes (1B, 1KB and 1MB). The results were obvious that the sequential read/writes were faster than random read/writes. The concurrency degree also plays a major role in throughput. The below figure shows the graph for sequential reads and writes. The sequential operations are faster than random operations on memory. The throughput increases with the increase in the block size. The optimal number of threads for sequential read/writes differs with block sizes (2 threads for 1B block, 1 thread for 1KB and 1MB block).

The random operations on memory are slower than sequential operations. The throughput doesn't increase with increase in threads. The optimal number of threads was always 1. **The latency is 0.000018.** The below graph shows the throughput values for random operations on memory with varying block sizes and varying concurrency levels.

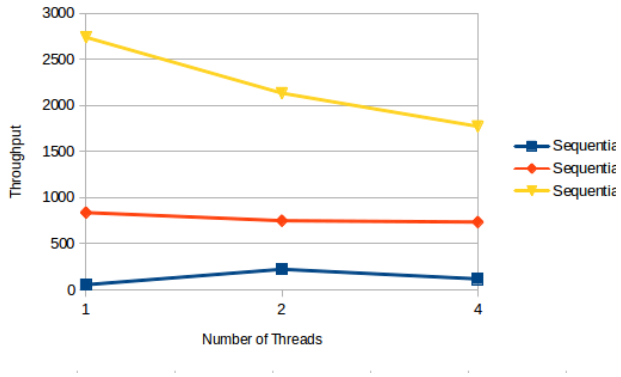

Figure6: Number of Threads versus Throughput (MB/s) for sequential reads/writes

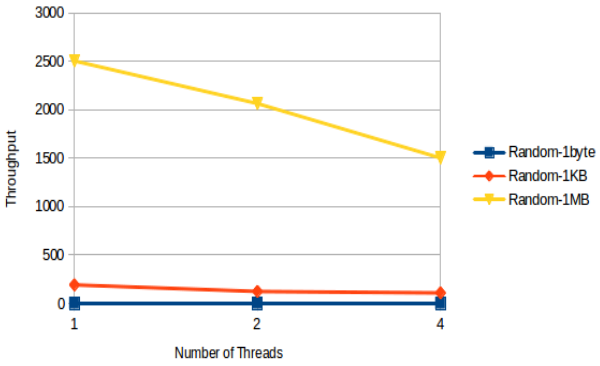

Figure7: Number of Threads versus Throughput (MB/s) for random reads/writes

Test Bed

Processor: Intel® Core™ i3 CPU M 330 @ 2.13GHz × 4
Memory: 4.0GB
OS type: 64 bit

5. Network Benchmarking Experiment- In this experiment we developed multithreaded client and server for both TCP and UDP. We ran experiments by sending a maximum of 10 million packets of varying block sizes (1B, 1KB and 64KB). The results were again obvious the TCP version was faster than the UDP. The graph explains more information about the results.

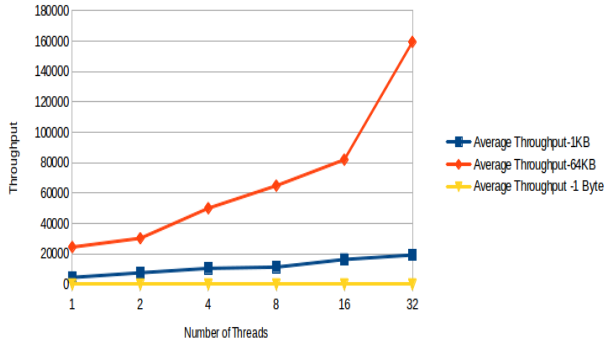

Figure8: Number of Threads versus Throughput (Mb/s) for TCP

The above figure shows the graph for TCP. It is very clear that the throughput increases with the increase of the packet size. Throughput for 64KB is greater than 1KB which in turn is greater than 1B. The same results hold well with increase in the level of concurrency. **For TCP the optimal number of threads for all block sizes is 32**. The results for UDP show the same results that the block size of 64KB has a better throughput than 1KB and 1B. Similarly increasing the concurrency level increases the throughput. But **UDP's throughput is not as good as TCP** which is clearly evident from the graph. The main reason is a lot of packets may be lost in case of UDP. The maximum allowable packet size in UDP is 65535 which include 20 bytes for IP header and 8 bytes for UDP header. So the maximum data size is 65507. The latency for TCP is 0.000345 and UDP is 0.00262.

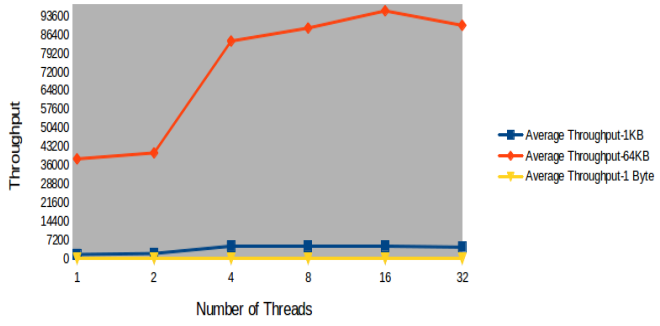

Figure9: Number of Threads versus Throughput (Mb/s) for UDP

Test Bed

Processor: Intel® Core™ i3 CPU M 330 @ 2.13GHz × 4
Memory: 4.0GB
OS type: 64 bit

III. BENCHMARKING WITH TOOLS (EXTRA CREDITS WORK)

CPU Benchmarking (Linpack):
For the Linpack test, given the threads as 4, cores =2, by varying the block size by 4 kb each time, and number of times to solve the equation is 2048, time taken is 0.5532 secs with GFlops 10.7721. The total number of tests performed is 32. Hence, the time for one test is 0.0176 secs.

Memory Benchmarking: (STREAM):
Considering 8 bytes per array item, array size given as 10000000 and the memory 228.9 Mib, upon performing the memcpy operation for 10 times, the observed output is -- Each test will take on the order of 18095 microseconds (18095 clock ticks). The total time taken is 180950.The best throughput got is 9326.1 MB/s, with Avg time = 0.022495 secs

Network Benchmarking: (Iperf):
The throughput got by running the application over TCP for 10 secs is 225 Mbits/sec having the window size as 85.0KB. The total bytes transferred are 268 Mbytes. The throughput got by running the application over TCP for 20Secs is 240 Mbits/sec and the total bytes transferred are 573 Mbytes. The window size is 144KB. By running the UDP over the client and server, 2.39 Mbytes is transferred by giving the bandwidth as 2.0Mbits/sec.

For more extra credits we have computed the standard deviation and average value which is attached as a separate document (please check the zip file)

IV CONCLUSION

In this work, we have studied and analyzed the performance of various computer system components by varying various input metrics and concurrency degree. We have learned more about how to use the components efficiently and effectively to get the maximum out of it. We see that there are different bottlenecks that prohibit each component to be utilized to the core. On the whole, now we can use this knowledge to design and develop any application or software based on the hardware it is going to run on.

V. REFERENCES

[1] http://en.wikipedia.org/wiki/LINPACK
[2] http://www.iozone.org/
[3] http://en.wikipedia.org/wiki/Iperf