CS553 Programming Assignment #1

Benchmarking

Instructions:

- ***Assigned date: Monday, 09/11/2017***
- ***Due date: 11:59PM on Sunday, 10/08/17***
- ***Maximum Points: 100% + 10% Extra Credit***
- *This programming assignment can be done individually, or in groups of two*
- *Please post your questions to the Piazza forum*
- *Only a softcopy submission is required; it must be submitted to "Digital Drop Box" on Blackboard*
- *For all programming assignments, please submit just the softcopy; please zip all files (report, source code, compilation scripts, and documentation) and submit it to BB.*
- *Name your file as this rule: "PA1_LASTNAME1_LASTNAME2.{zip|tar|pdf}". E.g. "PA1_Raicu_Peng.tar".*
- *Late submission will be penalized at 20% per day (beyond the 4-day late pass).*

1 Your Assignment

This project aims to teach you how to benchmark different parts of a computer system, from the CPU, GPU, memory, disk, and network.

You can be creative with this project. You are free to use any of the following programming languages (C, C++, Java, Python, Perl) and abstractions (PThreads, Sockets) that might be needed; other programming languages will not be allowed due to the increased complexity in grading; do not write code that relies on complex libraries (e.g. boost), as those will simplify certain parts of your assignments, and will increase complexity in grading. You must use the Chameleon testbed (https://www.chameleoncloud.org) for this assignment; you must use KVM virtual machine m1.medium (2 virtual processors with 4GB RAM and 40GB disk). You must use CentOS 7 Linux for the OS, as your TAs will use it to evaluate your programming assignments. The TAs will compile and test your code on the same Chameleon environment. If your code does not compile and the TAs cannot run your project, you will get 0 for the assignment.

In this project, you need to design a benchmarking program that covers the five components listed below in a series of 164 separate experiments:

1. **CPU (2*4+1*2+1 = 11 experiments):**
    a. For CPU benchmarks, use strong scaling experiments; this means you will set the amount of work (e.g. number of instructions you want to run in your benchmark), and reduce the amount of work per thread as you increase the number of threads
    b. Measure the processor speed, in terms of double precision floating point operations per second (Giga FLOPS, 10^9 FLOPS) and integer operations per second (Giga IOPS, 10^9 IOPS); ***hint:*** *modern processors can do multiple instructions per cycle, so make sure to give your benchmark good code to allow it to run multiple instructions concurrently; also look into AVX instructions to get even better performance; also make sure to turn on compiler optimizations; full credit can be received without the use of AVX instructions; implementation of code with AVX instruction and adequate evaluation using bare metal provisioning will be given extra credit 2%*
    c. Measure the processor speed at varying levels of concurrency (1 thread, 2 threads, 4 threads, 8 threads)

d. Compute the theoretical peak performance of your processor in flops/sec
e. What efficiency do you achieve compared to the theoretical performance?
f. As a separate experiment, run your benchmark on double precision floating point and integer instructions and 8 threads for a 10-minute period for each one, and take samples every second on how many instructions per second were achieved during the experiment; plot the data for the two experiments (FLOPS and IOPS) with time (0 to 10 min) on the x-axis and FLOPS/IOPS on the y-axis, with 1-second samples (you will have 600 samples for FLOPS and 600 samples for IOPS to plot); this item will be considered as extra credit 1%
g. Run the Linpack benchmark (http://en.wikipedia.org/wiki/LINPACK) and report the best performance achieved using double precision floating point; make sure to run Linpack across all cores and to use a problem size that is large enough to consume ¾ of the available memory of your VM; what efficiency do you achieve compared to the theoretical performance? How does this compare to the results you achieved? Linpack makes use of AVX instructions to achieve high efficiency in FLOPS; you may want to use them as well in your own benchmarks to improve your efficiency.

2. **GPU (1*4+1 = 5 experiments):**
   a. This entire benchmark is optional and extra credit 3%
   b. Measure the GPU speed, in terms of double precision (64-bit) floating point operations per second (GFLOPS), integer (32-bit) operations per second (GIOPS), half-precision (16-bit) operations per second (GHOPS), and quarter-precision (8-bit) operations per second (GQOPS); ***Hint:*** *You will have to use either CUDA or OpenCL to do this assignment; timing might also be tricky, compared to measuring timing on the host system*
   c. Measure the GPU speed with full concurrency (mapping 1 thread per core); use any available GPU on Chameleon; make sure to specify which GPU you used
   d. Compare the measured GPU speed with the theoretical compute speed, and explain your results
   e. Run the Linpack benchmark (http://en.wikipedia.org/wiki/LINPACK) and report the best performance achieved for double precision floating point; make sure to run Linpack across all cores and to use a problem size that is large enough to consume ¾ of the available memory of your GPU; what efficiency do you achieve compared to the theoretical performance? How about compared to the performance you achieved in your own benchmark?

3. **Memory (3*4*4+1+3*4 = 61 experiments):**
   a. For memory benchmarks, use strong scaling experiments
   b. Measure the memory speed of your host; ***hint:*** *you are unlikely going to be able to do this benchmark in Java, while C/C++ is a natural language to implement this benchmark*
   c. Your parameter space should include read+write operations (e.g. memcpy), sequential write access (e.g. memset), random write access, varying block sizes (8B, 8KB, 8MB, 80MB), and varying the concurrency (1 thread, 2 threads, 4 threads, and 8 threads); you should allocate a large piece of memory (e.g. 1GB), and perform read+write or write operations on either sequential or random access within this 1GB of memory. For the latency experiments, you can limit the number of 1B~8B operations to a small number, ensuring that the experiment runs for at least 10s of seconds.
   d. The metrics you should be measuring are throughput (Megabytes per second, MB/sec) and latency (microseconds, us); note that the 8B block case can be used to measure latency, and the 8KB, 8MB, and 80MB cases can be used to measure throughput
   e. Vary the number of threads and find the optimal number of concurrency to get the best performance
   f. Compute the theoretical memory bandwidth of your memory, based on the type of memory and the speed
   g. Run the Stream benchmark (http://www.cs.virginia.edu/stream/) and report the best performance achieved; what efficiency do you achieve compared to the theoretical performance?

h. Measure the GPU memory performance using sequential read+write and sequential write at full concurrency with large block sizes; compare the measured memory bandwidth with the theoretical memory bandwidth, and explain your results; compare the host memory performance to that of the GPU memory; the GPU memory experiment is extra credit 2%

4. **Disk (3*4*4+1 = 49 experiments):**
   a. For disk benchmarks, use strong scaling experiments
   b. Measure the disk speed; ***Hint:*** *there are multiple ways to read and write to disk, explore the different APIs, and pick the fastest one out of all them*
   c. Your parameter space should include read+write operations, sequential read access, random read access, varying block sizes (8B, 8KB, 8MB, 80MB), and varying the concurrency (1 thread, 2 threads, 4 threads, 8 threads)
   d. The metrics you should be measuring are throughput (MB/sec) and latency (ms); note that the 8B block case can be used to measure latency, and the 8KB, 8MB, and 80MB cases can be used to measure throughput; you should allocate a large piece file (e.g. 10GB), and perform read+write or read operations on either sequential or random access within this 10GB file. For the latency experiments, you can limit the number of 1B~8B operations to a small number, ensuring that the experiment runs for at least 10s of seconds.
   e. Identify the optimal number of concurrency to get the best performance; explain your findings, putting in perspective the hardware you are testing; can you tell if the disk you are evaluating is a spinning hard drive (HDD) or a solid state memory (SSD) disk?
   f. Run the IOZone benchmark (http://www.iozone.org/) and report the best performance achieved; what efficiency do you achieve compared to the theoretical performance? Hint: The theoretical performance is generally advertised by the manufacturer.

5. **Network (2*2*1*4*2 + 2*2+2= 38 experiments):**
   a. For network benchmarks, use strong scaling experiments
   b. Measure the network speed over the loopback interface card (1 node, between 2 processes on the same node); for extra credit 2%, measure the network speed between two nodes
   c. Your parameter space should include the TCP protocol stack, UDP, fixed packet/buffer size (64KB), and varying the concurrency (1 thread, 2 threads, 4 threads, 8 threads); note that the multi-threaded support must exist at both client and server
   d. The metrics you should be measuring are throughput (Megabits per second, Mb/sec) and latency (ms); you should fix the amount of data you want to transfer (e.g. 8GB, 8K ping-pong messages), and perform your experiments accordingly using strong scaling
   e. Run the IPerf benchmark (http://en.wikipedia.org/wiki/Iperf) and report the best achieved throughput performance over the loopback interface; what efficiency do you achieve compared to the theoretical memory performance? Repeat the benchmark across two nodes; what efficiency do you achieve compared to the theoretical network speed? For the latency, use the ping utility to measure latency across the loopback, and across the network. How does it compare to the theoretical latency? Test the following configurations: TCP, UDP, loopback, network, throughput, and latency.

Other requirements:

- You must write all benchmarks from scratch. You can use well known benchmarking software to verify your results, but you must implement your own benchmarks. Do not use code you find online, as you will get 0 credit for this assignment. If you have taken other courses where you wrote similar benchmarks, you are welcome to start with your codebase as long as you wrote the code in your prior class.
- All of the benchmarks will have to evaluate concurrency performance; concurrency can be achieved using threads. Use strong scaling in all experiments, unless it is not possible, in which case you need to explain why a strong scaling experiment was not done. Be aware of the thread synchronizing issues to avoid inconsistency or deadlock in your system.

- All of these required benchmarks could be done on a single machine. The multi-node experiments are optional and extra credit. All experiments can be done on KVM on 2 virtual core instances with 4GB of RAM and 20GB disk.
- Experiments should be done in such a way that they take multiple seconds to minutes to run, in order to amortize any startup costs of the experiments; that means that for some of the operations that are really small (e.g. 1B), you might need to do many thousands or even millions of them to run long enough to amortize the costs of the benchmark overheads.
- Not all timing functions have the same accuracy; you must find one that has at least 1ms accuracy or better, assuming you are running the benchmarks for at least seconds at a time.
- Since there are many experiments to run, find ways (e.g. scripts) to automate the performance evaluation.
- Make sure your machine is idle when running the benchmarks as it would help to improve reliability and consistency of the results. For the best reliability in your results, repeat each experiment 3 times and report the average and standard deviation. This is not mandatory, but it will help you get more stable results that are easier to understand and justify.
- No GUIs are required. Simple command line interfaces are expected.

2 What you will submit

When you have finished implementing the complete assignment as described above, you should submit your solution to 'digital drop box' on blackboard. Each program must work correctly and be detailed in-line documented. You should hand in:

1. **Design Doc (10%):** A separate (typed) design document (named pa1-report.pdf) of approximately 1-3 pages describing the overall program design, and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made).
2. **Manual (10%):** A detailed manual describing how the program works. The manual should be able to instruct users other than the developer to run the program step by step. The manual should contain example commands to invoke each of the five benchmarks. This should be included as readme.txt in the source code folder.
3. **Source code (30%):** All of the source code; in order to get full credit for the source code, your code must have in-line documents, must compile, and must be able to run the sample benchmarks from #2 above. You must have a makefile for easy compilation. Please specify which student contributed to what code.
4. **Performance (50%):** Since this is an assignment aimed at teaching you about benchmarking, this is one of the most important part; you must evaluate the five benchmarks with the entire parameters space mentioned in Section 1, and put as a sub-section to your design document mentioned in (1) above. You must produce graphs/tables to showcase the results. Please combine data and plot on the same graph wherever possible, for either compactness reasons, or comparison reasons. Don't forget to plot the average and standard deviation (if you do multiple runs of an experiment). Also, you need to explain each graph's results in words. Hint: graphs with no axis labels, legends, well defined units, and lines that all look the same, are likely very hard to read and understand graphs. You will be penalized if your graphs are not clear to understand. Please specify which student contributed to what benchmark experiments.
5. **Extra credit (10%):** GPU related evaluations, AVX instructions on Intel CPUs, and multi-node networking evaluation are optional and extra credit. Any evaluations you completed from the original writeup on 8 virtual core VMs and bare metal systems will receive full credit, so there is no need to redo anything that you have done from the original assignment; but if you are still working on it, the new assignment should hopefully be simplified and easier to complete in its current form.

Please put all of the above into one .zip or .tar file, and upload it to 'digital drop box' on blackboard'. The name of .zip or .tar should follow this format: *PA1_LastName1_LastName2.{zip|tar}.* Please do NOT email your files to the professor and TA!! You do not have to submit any hard copy for this assignment, just a soft copy via Black Board.

Grades for late programs will be lowered 20% per day late beyond the 4-day late pass.