CS553 Homework #3

Benchmarking CPU

Instructions:

- *Assigned date: Wednesday February 27th, 2019*
- *Due date: 11:59PM on Friday March 8th, 2019*
- *Maximum Points: 100%*
- *This programming assignment must be done individually*
- *Please post your questions to the Piazza forum*
- *Only a softcopy submission is required; it will automatically be collected through GIT after the deadline; email confirmation will be sent to your HAWK email address*
- *Late submission will be penalized at 10% per day; an email to cs553-s19-group@iit.edu with the subject "CS553: late homework submission" must be sent*

1 Your Assignment

This project aims to teach you how to benchmark the processor. You can be creative with this project. You are free to use any of the following programming languages (C, C++) and abstractions (PThreads) 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 can use any Linux system for your development, but you must use the Chameleon testbed [https://www.chameleoncloud.org]; more information about the hardware in this testbed can be found at https://www.chameleoncloud.org/about/hardware-description/, under Standard Cloud Units. Even more details can be found at https://www.chameleoncloud.org/user/discovery/, choose "Compute", then Click the "View" button. You have been created accounts on this virtual cluster for this assignment; if you have not received your accounts information, reach out to the TAs for this information.

In this project, you need to design a benchmarking program that computes matrix multiplication of square matrices of arbitrary size (within the constraints of available memory). You will perform strong scaling studies, unless otherwise noted; this means you will set the amount of work (e.g. the number of instructions or the amount of data to evaluate in your benchmark), and reduce the amount of work per thread as you increase the number of threads. The TAs will compile (with the help of make, ant, or maven) and test your code on the same Chameleon testbed. If your code does not compile and the TAs cannot run your project, you will get 0 for the assignment.

1. CPUBench benchmark:
    a. Implement CPUBench benchmark
    b. Workload: 1 trillion arithmetic (single precision, double precision) operations
        i. SP: single precision operations compute on 4-byte float data types
        ii. DP: double precision operations compute on 8-byte double data types
    c. Concurrency: 1 thread, 2 threads, 4 threads, 8 threads
    d. Measure: processor speed, in terms of operations per second; report data in GigaOPS, giga operations (10^9) per second
2. MatrixBench benchmark:

   a. Implement matrix multiplication (https://en.wikipedia.org/wiki/Matrix_multiplication) supporting square matrices, of arbitrary size. For example, you should be able to support matrices of size 2, where 2x2 matrix is multiplied by another 2x2 matrix, and result in a 2x2 matrix. A matrix of size 1000 would allow a 1000x1000 matrix to be multiplied by another 1000x1000 matrix to result into a new 1000x1000 matrix. You need to implement the matrix multiplication to support multiple cores, as you will evaluate its performance in a multi-core environment. You need to use compiler optimization flags, matching the flags that HPL uses during compilation.
   b. Workload: You should also support matrices of size N, where N denotes the matrix size and follow these constraints: N*N*SIZEOF(element) <= 0.75*RAMSIZE. The element you should support are float and double. Each element should be randomly set to a float (or double) value between -1 and +1.
      i. SP: single precision operations compute on 4-byte float data types
      ii. DP: double precision operations compute on 8-byte double data types
   c. Concurrency: 1 thread, 2 threads, 4 threads, 8 threads
   d. Measure: processor speed, in terms of operations per second; report data in GigaOPS, giga operations (10^9) per second
3. HPL benchmark:
   a. Run the HPL benchmark from the Linpack suite (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 testing node (for this benchmark, you do not have to compute 1 trillion arithmetic computations, but the problem size you set to fill ¾ of the available memory will dictate the workload size; you should use the same problem size as in the matrix multiplication you implemented and evaluated in sub-section 2 above). You can download the Modified HPL Project from http://www.nics.tennessee.edu/sites/default/files/HPL-site/home.html (you need to download it, and compile it). You are going to run shpl (single precision) and xhpl (double precision). Make sure to tune the HPL benchmark in HPL.dat (see http://www.netlib.org/benchmark/hpl/tuning.html for more information on how to interpret the HPL.dat configuration). You will need a working MPI installation; you can install MPICH or OpemMPI with your package manager. You will need to install a BLAS implementation (shared library, not static) using the package manager (e.g. BLAS, LAPACK, or ATLAS). You will need to update the options file with the path to the BLAS shared library.
   b. Compute the theoretical peak performance of your processor. What efficiency do you achieve compared to the theoretical performance? Compare and contrast your performance to that achieved by your benchmark, HPL, and the theoretical peak performance.

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.
- Most benchmarks could be run on a single machine.
- 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.
- For the best reliability in your results, repeat each experiment 3 times and report the average and standard deviation. This will help you get more stable results that are easier to understand and justify.
- No GUIs are required. Simple command line interfaces are expected.

Fill in the table 1 below for the Processor Performance:

Work -load	**Con-cur-rency**	**CPUBench Measured Ops/Sec (GigaOPS)**	**MatrixBench Measured Ops/Sec (GigaOPS)**	**HPL Measured Ops/Sec (GigaOPS)**	**Theoretical Ops/Sec (GigaOPS)**	**CPUBench Efficiency (%)**	**MatrixBench Efficiency (%)**	**HPL Efficiency (%)**
SP	1							
SP	2							
SP	4							
SP	8							
DP	1							
DP	2							
DP	4							
DP	8							

2 Where you will submit

You will have to submit your solution to a private git repository created for you at git@gitlab.com:cs553-2019/<student email id>.git. A directory structure for all the repositories will be created that looks like this: hw1/, hw2/, … You will have to firstly clone the repository. Then you will have to add or update your source code, documentation and report. Make sure this assignment's source code can be found in your hw3 folder. Your solution will be collected automatically after the deadline. If you want to submit your homework later, you will have to push your final version to your GIT repository and you will have let the TAs know of it through email cs553-s19-group@iit.edu. There is no need to submit anything on BB for this assignment. If you cannot access your repository contact the TAs. You can find a git cheat sheet here: https://www.git-tower.com/blog/git-cheat-sheet/

3 What you will submit

When you have finished implementing the complete assignment as described above, you should submit your solution to your private git repository. Each program must work correctly and be detailed in-line documented. You should hand in (points will be given for each of the items below):

1. **Source code:** 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.
2. **Readme:** 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. **Report / Performance:** Must have working code that compiles and runs on the specified cluster to receive credit for report/performance; furthermore, the code must match the performance presented in the report. A separate (typed) design document (named hw3-report.pdf) of approximately 1 page long 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). Since this is an assignment aimed at teaching you about benchmarking, this is one of the most important parts; you must evaluate the benchmark 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 1 table to showcase the results.

Submit code/report through GIT.

Grades for late programs will be lowered 10% per day late.