Benchmarking Homework Assignment #1

Mike Aiello

I. Introduction

A. Assignment

This project aims to teach the student how to benchmark their computer's computational performance. In this project, the student needs to design a benchmarking program that measures the CPU speed, in terms of floating point operations per second (Giga FLOPS, FLOPS) and integer operations per second (Giga IOPS, IOPS); as well as measure the processor speed at varying levels of concurrency (1 thread, 2 threads, 4 threads, and 8 threads).

B. Requirements

- Benchmarks must be written from scratch. Once can use well known benchmarking software to verify your results, but one must implement their own benchmarks. Do not use code online, as one will get 0 credits for this assignment.
- All of the benchmarks will have to evaluate concurrency performance; concurrency can be achieved using threads. Be aware of the thread synchronizing issues to avoid inconsistency or deadlock in one's system.
- 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.
- Not all timing functions have the same accuracy; one must find one that has at least 1 ms accuracy or better.
- Since there are many experiments to run, one must find ways (e.g. scripts) to automate the performance evaluation.
- All benchmarks must be run 3 times, and the reported values should be the average and standard deviation.
- No GUIs are required. Simple command line interfaces will suffice.

II. Design

The assignment while specific in its requirements, the design of the program is still relatively open-ended and some things need to be looked at and contemplated before development. In order to achieve the goal of the program in measuring CPU speed in specific terms of floating point operations per second and integer operations per second, these terms need to be more fully understood.

A. Floating point operations

Floating point operations per second refers to the measure of a computer performance based on the number of floating-point arithmetic calculations a specific processor would be able to perform within 1 second. The use of floating-point numbers allows for the computer to handle long numbers with varying magnitudes of precision, hence why Giga Flops is measured in 10^9 Flops.

With all this in mind, a single floating point operation is a basic arithmetic operation such as computing a sum, doing subtraction, multiplying to a product, or dividing to a quotient. To measure the number of floating point operations a CPU can handle within a given time period is dependent on the operation being done and the algorithm in use.

For multiple use case examples and increased complexity of design for the program, there will be 4 functions created specific to floating point operations, in that for each benchmark test , there will be 4 results of floating point operations per second. The first algorithm will utilize the addition operation, the second will use the subtraction operation, the third will use multiplication, and the last one will use division.

The program will perform a specified number of iterations of each operation, and the time will be measured to see how long it took to do all the iterations. In order to find the floating point operations per second, the number of iterations will be divided by the time it took to complete them.

B. Integer operations

As it goes for integer operations, it is similar yet slightly different. Instead of dealing with floating point numbers or decimal values, integers refer to all whole, real numbers excluding any decimal or fractional values. The same operations will be used, such as addition, subtraction, multiplication, and division.

For multiple use case examples and design of the program, there will also be 4 functions created

specific to integer operations, in that each benchmark test, there will be 4 results of integer operations per second. The first algorithm will utilize the addition operation, the second will use the subtraction operation, the third will use multiplication, and the last one will use division.

The program will perform a specified number of iterations of each operation, and the time will be measured to see how long it took to do all the iterations. In order to find the integer operations per second, the number of iterations will be divided by the time it took to complete them.

C. Number of threads

Another important consideration for the program is the number of threads being used, also referred to as the level of concurrency. Simply the program will calculate the FLOPS and IOPS for 1 thread, 2 threads, 4 threads, and 8 threads. This means the algorithm or calculation and work load being done to find the floating point operations and integer operations per second will be split amongst 1, 2, 4 and 8 threads. The idea we want to explore is wether more threads makes its harder or easier for the computer to do its processing. Another way to look at it is will the FLOPS and IOPS be higher or lower when using more or less threads. That is the goal of the experiment. The design of the experiment also takes into account that the benchmark test will have to be run 3 times in order to produce an average and standard deviation value.

D. Potential Other Designs

A thing to consider in the development of this benchmark program is the actual number of operations per second will not be exact to the CPU's real number it would be able to achieve. This is due to the simpler nature of the code, and how it doesn't take into consideration the time it might take to handle the inherent other parts of the program and computer instructions happening behind the scenes other than the loop with counter structure that is used to calculate the number of operations per second. On top of this in considering other potential designs and improvements, the main thing that could change and be implemented is including other operational algorithms. For example, in this benchmark instead of just using 4 of the main arithmetic operations, there could have been a function or algorithm that incorporated all 4 of these operations. There could also have been an array filling operation that tests the number of operations per second based on the time it takes to append values to a list or matrix.

III. Manual

As slightly alluded to in the design section, the way the program will work and be used is relatively self-explanatory. For simplicity, no user input is required in the use cases of this program considering the goal is to acquire values of FLOPS and IOPS, and the operations needed to do so are automated relative to computing done by the processor. While providing an option for the user to choose the number of threads (1,2,4,8) or the specific operation done within the algorithm (add, subtract, multiply, divide) could be available, since we are looking to conduct experiments on the processor of the computer running the program, it would be best to simply calculate all cases and then organize the data into graphs and tables to compare and contrast after.

How to load and run program

1. Download code
   The program consists of 3 python files, main.py, BenchmarkThreading.py, and BenchmarkData.py

2. install Matplotlib
   type "pip install matplotlib" in terminal in project code folder to make sure your environment can support its functionality

3. Run main.py
   Users upon starting the program , the user will be told what the program is going to do, as well as be prompted to click enter multiple times before proceeding to benchmark. Once a user clicks enter to proceed with benchmark program, there will be no more user input needed, as everything will be automated from this point on

   How the program works

   Once benchmark begins in main.py, an instance of a data object from BenchmarkData.py is first initialized and created to prepare for all the calculated values to be organized into lists, and 2D matrices. This Data object passes as a parameter to a function in BenchmarkThreading.py that starts the benchmark. Once triggered, a function within BenchmarkThreading.py will create the threads and start them all simultaneously. Most of the computation will take place in this function as there are calls to other respective functions needed to perform the base operation for the benchmark. This is an iterative process where depending on the number of threads, this process will need to be repeated several times within 1 round of the benchmark. The actual computation that happens in each iteration to find the number of floating point operations and

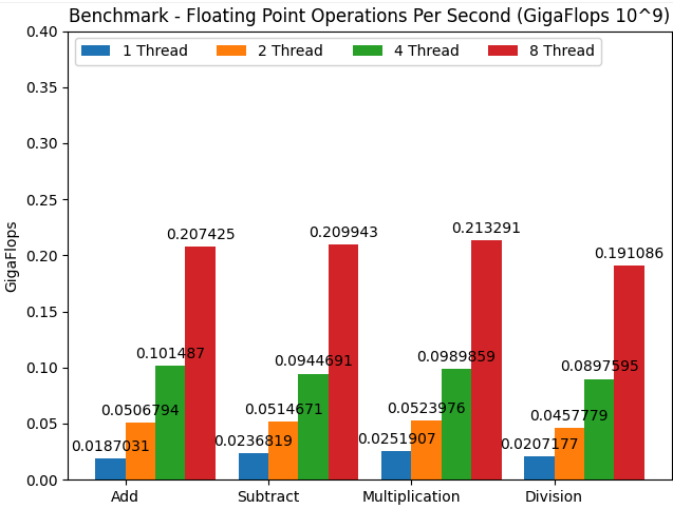
integer operations within a second consists of a for loop that will iterate a specific number of iterations with a timer starting before and ending once the loop is finished to result in time it takes to do the given number of operations. As a side note the program took into account being able to evenly split the workload/number of iterations between the different threads by dividing this number of iterations by the number of threads. This number of iterations as deduced through prior testing will have to be at least 10,000,000. The reason for this is because anything below this number will result in the timing of the operation being too fast to calculate using the time methods and functions supported by python. This is the case with an average computer. Hypothetically, a user's computer can be above average in processing and 10,000,000 might still be too small. Thats the purpose of the program in general however, to determine cpu speed, which is relative to its computing power. This variable can be changed in the code, and does not necessarily change results, it simply just takes longer or slower to complete the benchmark. Regardless, as for the operation being computed in the iterative loops, as stated before is either addition, subtraction, multiplication, and division. For addition and subtraction there is a sum, for multiplication there is a product, and division has a quotient. For simplicity, each sum, product, or quotient is incremented by 1 or by a factor of 1. In program design, these algorithms can be different, and the more complex, the longer each iteration would take. This program design is comparatively aimed at calculating FLOPS and IOPs with the most simple of operations to serve as a baseline for the benchmark. After all this, once the FLOPs and IOPs are calculated, this number is divided by 10^9 to result in Gigaflops and Giga IOPs as specified in the program requirements. As each computation is finished it is immediately stored into the respective data object's list or matrix variable that had been passed into the function. Once all the computations are complete for all the operations within each of the threads, then the first round of the benchmark has been completed. This whole process will need to reoccur 2 more times, with the data being stored into 2 separate data objects to ensure there is no crossover.

As the program runs, after each round of the benchmark is complete, 2 graphs will appear displaying the data for that round of the benchmark. One is for the floating point operations and another is for the integer point operations. Each graph will display the number of operations keycoded / relative to the number of threads and the class of operation, such as addition, subtraction, multiplication, division. These graphs visibly show the relationship between level of concurrency and total number of operations.

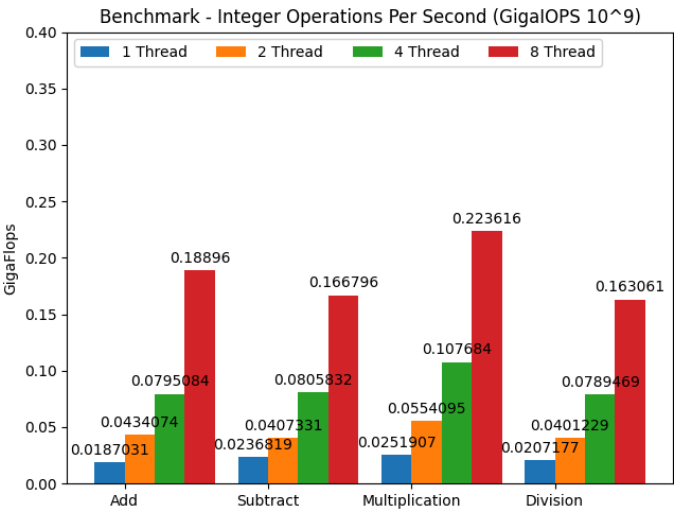
Lastly, once all 3 rounds of the benchmark are complete, 4 more graphs are produced. 2 of them are the total averages across all rounds of benchmarks, one for floating point operations per second, and the other one for integer point operations per second. The other 2 are for the standard deviation values,again, one for floating point point operations and the other for integer point operations, respectively.

IV. Performance

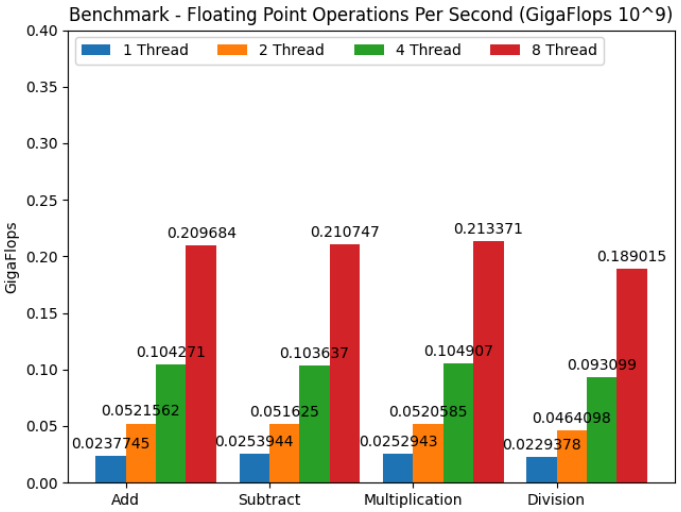
Graph 1: Round 1 FLOPS



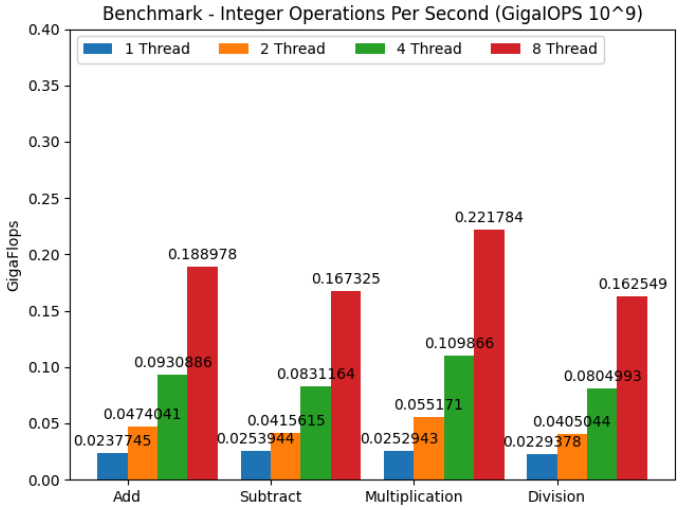
Graph 2: Round 1 IOPS



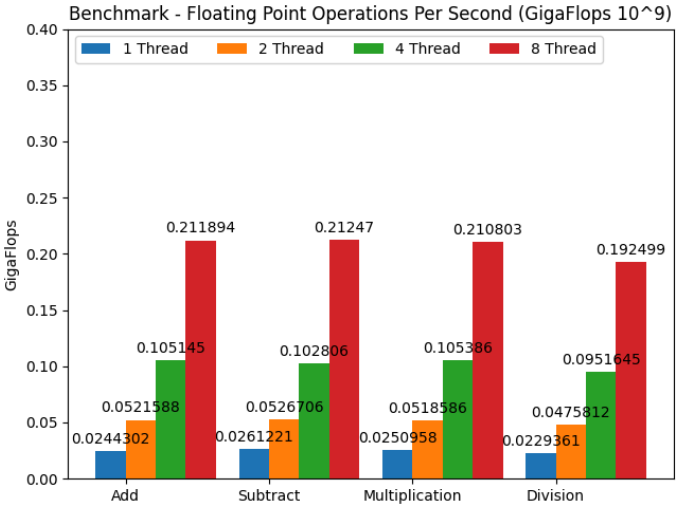
Graph 3: Round 2 FLOPS



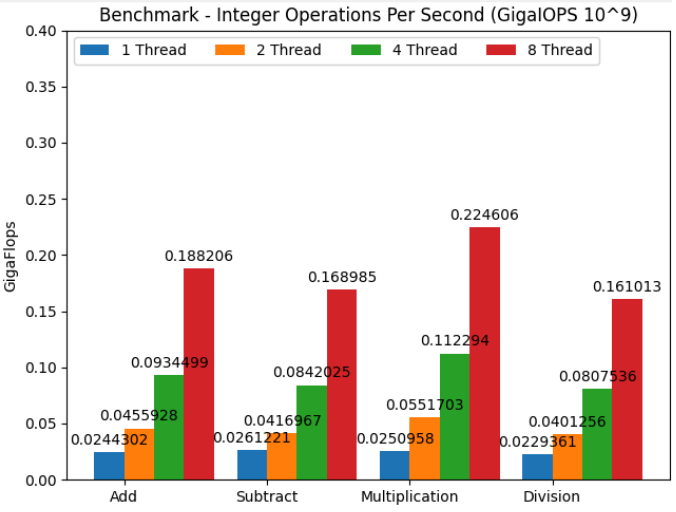
Graph 4: Round 2 IOPS



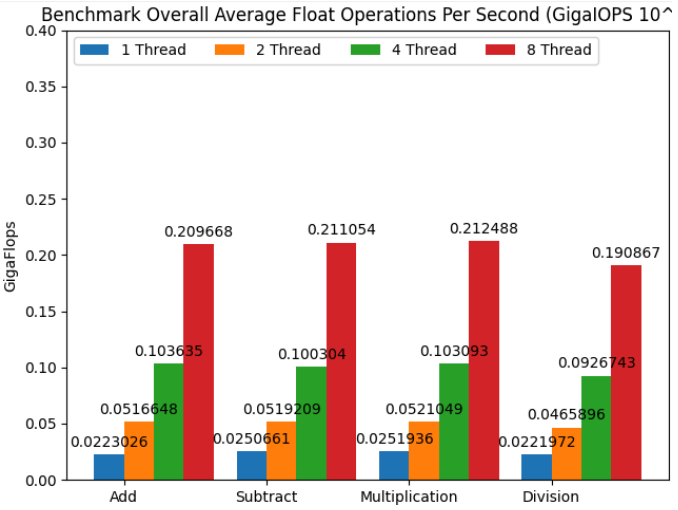
Graph 5: Round 3 FLOPS



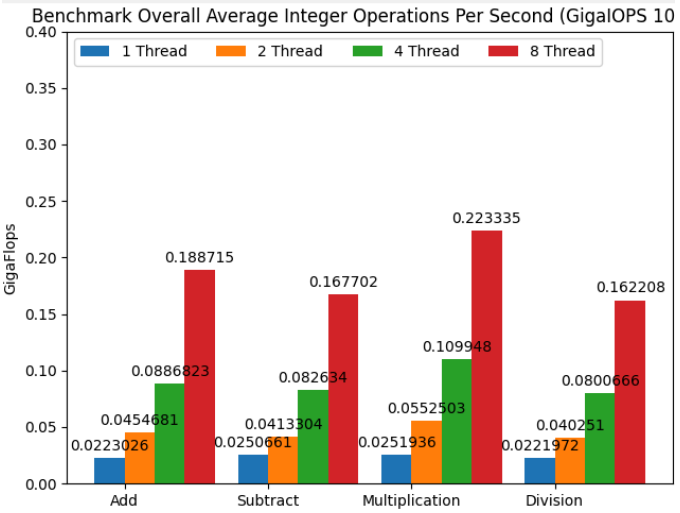
Graph 6: Round 3 IOPS



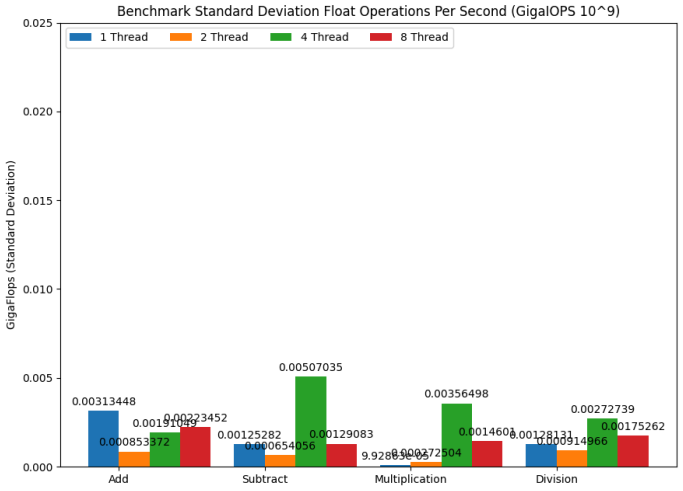
Graph 7: Overall FLOPS Average



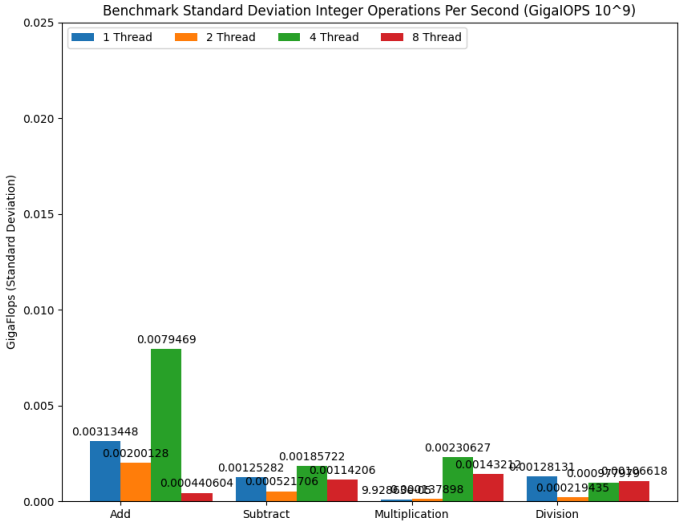
Graph 8: Overall IOPS Average



Graph 9: FLOPS Standard Deviation



Graph 10: IOPS Standard Deviation



As a performance analysis, the graphs should display the general patterns amongst the data, and as depicted in graphs 1 to 8, one can clearly see that the pattern remains consistent across the number of threads. This variable was the main one being studied and it seems to have worked as expected or desired. This means that as the number of threads increases, so does the number of both floating point operations per second and the integer operations per second. Each operation such as addition, subtraction, multiplication, and division have 4 averages, 1 from each of the thread tests and based on the graph there is gradual increase in FLOPS and IOPS, starting lower with the blue line, increasing more to the orange line, increasing more with the green line, and then finally the largest number with the red line. Arguably as a rough estimate across the data, one can say the number of Giga Flops or Giga IOPs doubled with the increase of threads. For example, from the overall averages, we see that 1 thread was able to do about .02-.025 GigaFlops and GigaIops. 2 threads was able to do about double this between .04-.05. 4 threads were able to do double that from .08-.1. Lastly 8 threads were able to do double that from around .16 - .25.

In addition the difference between FLOPs and IOPS seemed to be very minimal as the averages were relatively the same, meaning that the CPU computing operations involving floats vs integers did not necessarily make that large of a difference in this benchmark test.

Another observation is that while all operations seemed to produce relatively consistent results, as in they all had similar values respective to each thread test, meaning it doesn't really matter to much what operation was being done, however, after graph analysis, multiplication might of had slightly more operations per second comparatively to addition, subtraction, and division.

When looking at the standard deviations for all the averages, the results seem relatively more random, however this is expected considering the nature of the standard deviation calculation which gauges the spread of the values comparatively to its mean average. Based on the graphs, one can see that the 4 thread tests had a tendency to deviate from its mean value more so than the other thread tests.