ASTR 415

Evan Shipley-Friedt

September19, 2022

1 Problem Set #1

CPU Expense of Various Mathematical Operations

Operand Type	Starting Value	Operation	CPU time
integer	0	add 5	0.000154
integer	0	subtract 5	0.000153
integer	1	multiply by 1	0.000264
integer	1	divide by 1	0.000225
double-precision	0.0	add 5.0	0.000243
double-precision	0.0	subtract 5.0	0.00027
double-precision	1.0	multiply by 1.000001	0.000164
double-precision	1.0	divide by 1.000001	0.000148
double-precision	1.1	take the root (sqrt(x))	0.000244
double-precision	1.1	raise to 0.5 (pow(x,0.5))	0.002383

1.a Problem Statement Summary

Table of execution times for double-precision addition and multiplication as a function of n. Including timings for the square root and power functions for comparison.

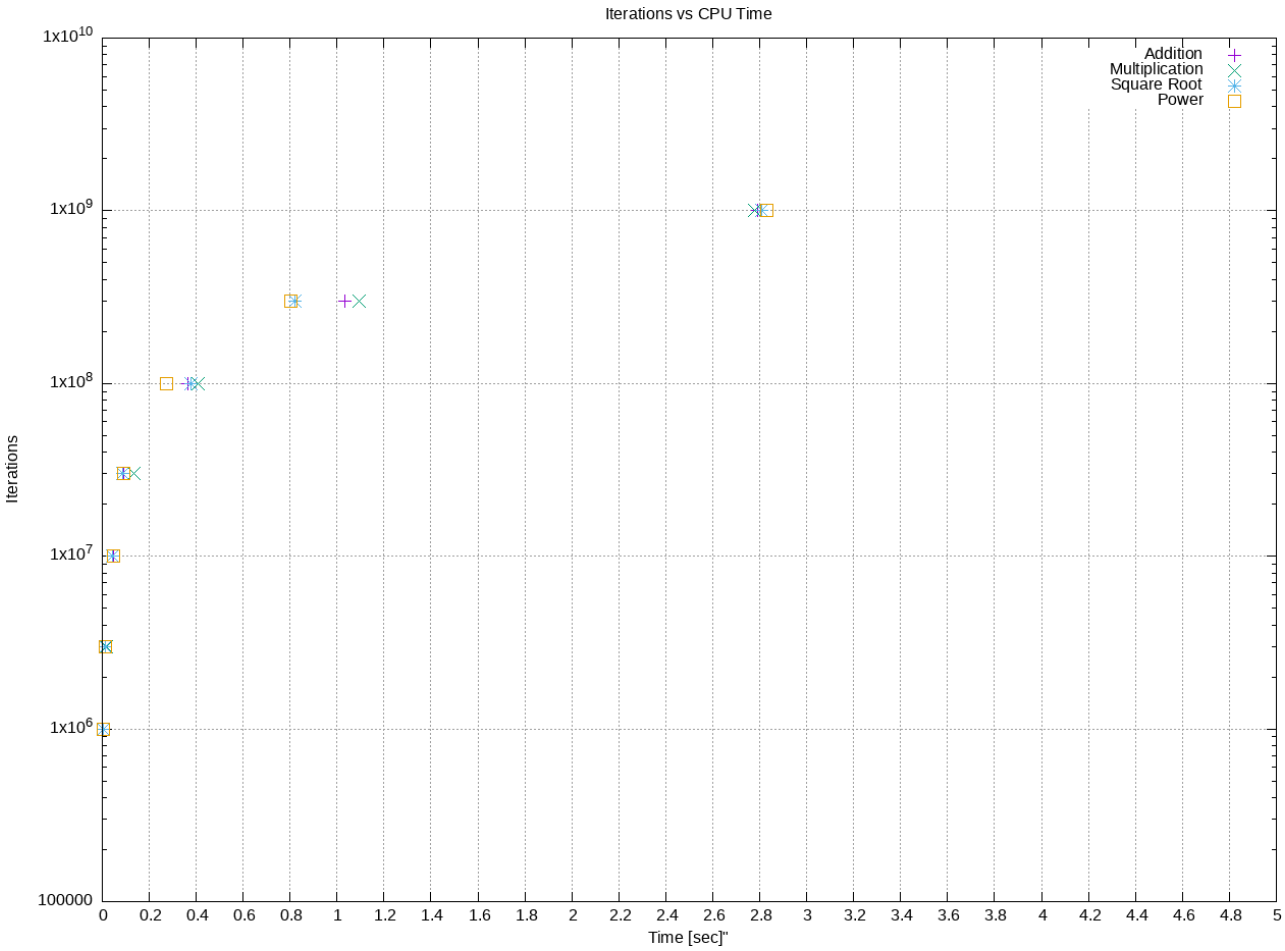
CPU Expense Table using cc

n	Value Counted	Operation	CPU time	MFFLOPs
1e+06	5.0e+06	+	0.004564	2.19e+08
3e+06	1.5e+07	+	0.015257	1.97e+08
1e+07	5.0e+07	+	0.055914	1.79e+08
3e+07	1.5e+08	+	0.126566	2.37e+08
1e+08	5.0e+08	+	0.337663	2.96e+08
3e+08	1.5e+09	+	1.210816	2.48e+08
1e+09	5.0e+09	+	2.799651	3.57e+08
1e+06	2.7e+00	x	0.002840	3.52e+08
3e+06	2.0e+01	x	0.012238	2.45e+08
1e+07	2.2e+04	x	0.050281	1.99e+08
3e+07	1.1e+13	x	0.113824	2.64e+08
1e+08	2.7e+43	x	0.299601	3.34e+08
3e+08	1.9e+130	x	0.900617	3.33e+08
1e+09	inf	x	2.802892	3.57e+08
1e+06	1.0e+00	s	0.003466	2.89e+08
3e+06	1.0e+00	s	0.013667	2.20e+08
1e+07	1.0e+00	s	0.045767	2.18e+08
3e+07	1.0e+00	s	0.092581	3.24e+08
1e+08	1.0e+00	s	0.282555	3.54e+08
3e+08	1.0e+00	s	0.850420	3.53e+08
1e+09	1.0e+00	s	2.852211	3.51e+08
1e+06	1.0e+00	^	0.002915	3.43e+08
3e+06	1.0e+00	^	0.011883	2.52e+08
1e+07	1.0e+00	^	0.080797	1.24e+08
3e+07	1.0e+00	^	0.186115	1.61e+08
1e+08	1.0e+00	^	0.301135	3.32e+08
3e+08	1.0e+00	^	0.849116	3.53e+08
1e+09	1.0e+00	^	2.824868	3.54e+08

1.b Problem Statement Summary

Plot the data in the table, execution time as a function of n. Estimate the number of MFLOPS on average the computer carried out.

CPU Expense Table using gcc



Average MFLOPS for standard cc compiler:

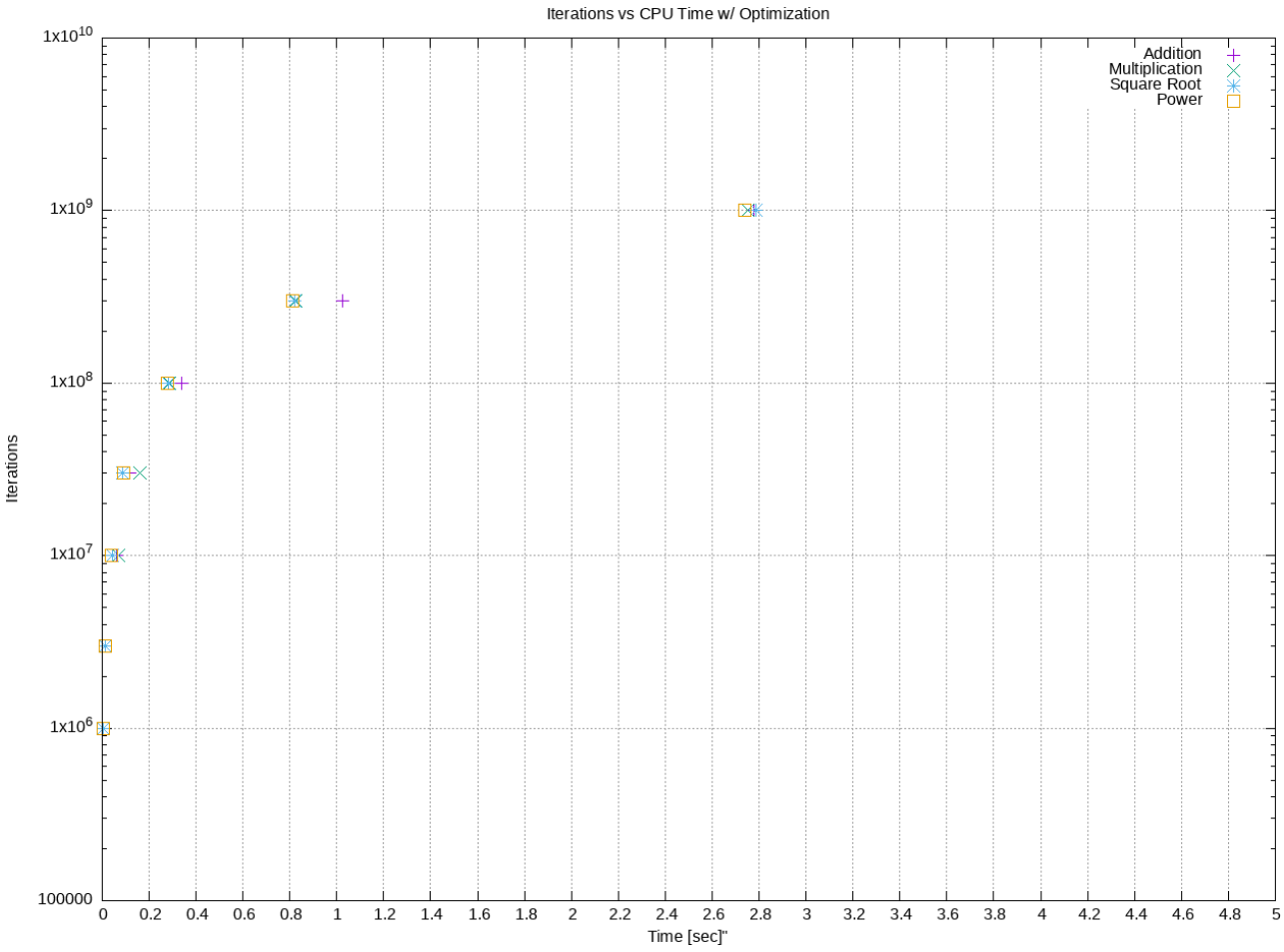
```
MFLOPS average = 2.63e+08
```

1.c Problem Statement Summary

Redo one or more plots in question 1b after recompiling your code with optimizations turned on (e.g., try the "-O2" compiler flag) and plot the results. Also try a different compiler if possible, with and without optimizations (e.g., if you used gcc, try icc next.) Comment on the results and suggest why differences exist.

CPU Expense Table using gcc

n	Value Counted	Operation	CPU time	MFFLOPs
1e+06	5.0e+06	+	0.003669	2.73e+08
3e+06	1.5e+07	+	0.014595	2.06e+08
1e+07	5.0e+07	+	0.058118	1.72e+08
3e+07	1.5e+08	+	0.106730	2.81e+08
1e+08	5.0e+08	+	0.348419	2.87e+08
3e+08	1.5e+09	+	0.999560	3.00e+08
1e+09	5.0e+09	+	3.878606	2.58e+08
1e+06	2.7e+00	x	0.004769	2.10e+08
3e+06	2.0e+01	x	0.013576	2.21e+08
1e+07	2.2e+04	x	0.058578	1.71e+08
3e+07	1.1e+13	x	0.129324	2.32e+08
1e+08	2.7e+43	x	0.333362	3.00e+08
3e+08	1.9e+130	x	1.174547	2.55e+08
1e+09	inf	x	3.506024	2.85e+08
1e+06	1.0e+00	s	0.003017	3.31e+08
3e+06	1.0e+00	s	0.012182	2.46e+08
1e+07	1.0e+00	s	0.053595	1.87e+08
3e+07	1.0e+00	s	0.117966	2.54e+08
1e+08	1.0e+00	s	0.298567	3.35e+08
3e+08	1.0e+00	s	1.097270	2.73e+08
1e+09	1.0e+00	s	3.587376	2.79e+08
1e+06	1.0e+00	^	0.003918	2.55e+08
3e+06	1.0e+00	^	0.013217	2.27e+08
1e+07	1.0e+00	^	0.057697	1.73e+08
3e+07	1.0e+00	^	0.128153	2.34e+08
1e+08	1.0e+00	^	0.346834	2.88e+08
3e+08	1.0e+00	^	1.076634	2.79e+08
1e+09	1.0e+00	^	3.591722	2.78e+08



Average MFLOPS for gcc compiler with no optimization setting:

```
MFLOPS average = 2.56e+08
```

Average MFLOPS for gcc compiler with -O1:

run 1
```
MFLOPS average = 2.84e+08
```
run 2
```
MFLOPS average = 2.53e+08
```

Average MFLOPS for gcc compiler with -O2:

run 1
```
MFLOPS average = 2.85e+08
```
run 2
```
MFLOPS average = 2.83e+08
```

I did not use the icc compiler at this time because it would have required me to install 15 GB worth of libraries and my computer's hard drive is already about 80% full.

In conclusion it appears that running the floating point operations for addition and subtraction tend to perform slower for addition specifically for the 1e8 and 3e8 numbers of iterations, but in all other n values over many tests they sometimes perform faster and sometimes slower.

Overall, it first appeared as though the simpler operations (multiplication and addition) perform the fastest with the regular cc compile method. However, using gcc causes the square root and power functions to regularly perform even more quickly and with the highest MFLOPS. Using -O1 and -O2 with gcc only drastically slowed down the average MFLOPS.

I would say the basic operations performed second fastest with the regular cc method because it will perform the operation in the executable in an efficient way through code and on the computer's processor. I trust that gcc sped up the square root and power functions because the gnu compiler either has a very efficient loop method in calculating them or more likely that it spreads out the calculations across more cores compared to the cc compiler. This allows for more operations to be done at once. Perhaps it could even have a method of telling the CPU the more intensive parts of the executable so that there are no cores idling or waiting on another core to finish a calculation before the former core can move on with it. In otherwords gcc might be better at preventing the calculation from bottle-necking.

I spent over 5-10 hours a day for 5 days working on this assignment. I felt I had a lot of catching up to do compared to some students who already know C, C++, Fortran, and Assembly. I really like coding and get absorbed in learning all kinds of advanced techniques that can help me code more efficiently in the future. Perhaps I also pay attention to details a little too closely when it comes to anything coding related.