Yonathan Daniel
HPML HW1

Miscellaneous for C1-C5

Use harmonic mean to compute average execution time listed in $< T >$: X.
For the Bandwidth calculations I used the formula below.

$$Bandwidth = \frac{2 * sizeof(float) * N * Measurements}{totaltime}$$

This was intuitive as there were two float arrays of size N allocated.
For the throughput I used a similar equation but it had more to do with the number of floating point operations. There were 2 in our case for the loop.

$$Throughput = \frac{floatingPointOps * N * Measurements}{totaltime}$$

I just wanted to explicitly state my intuition behind the calculations as I needed a couple sanity checks after doing this a couple times. I did ran all the code I provided via a virtual machine using GCP.

C1

```
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
N: 1000000, <T>: 0.001526 sec, B: 5.234219 GB/s, F: 1.308555 GFLOP/sec
yd2696@instance-1:~/HPML_dotproducts_benchmarks$ ./dp1 1000000 1000
```

(a) dp1 100000 10 input

```
yd2696@instance-1:~/HPML_dotproducts_benchmarks$ ./dp1 300000000 20
The arguments supplied are 300000000  20
TA advice or declare volatile? 16777216.000000
TA advice or declare volatile? 16777216.000000
TA advice or declare volatile? 16777216.000000
TA advice or declare volatile? 16777216.000000
TA advice or declare volatile? 16777216.000000
TA advice or declare volatile? 16777216.000000
TA advice or declare volatile? 16777216.000000
TA advice or declare volatile? 16777216.000000
TA advice or declare volatile? 16777216.000000
TA advice or declare volatile? 16777216.000000
TA advice or declare volatile? 16777216.000000
TA advice or declare volatile? 16777216.000000
TA advice or declare volatile? 16777216.000000
TA advice or declare volatile? 16777216.000000
TA advice or declare volatile? 16777216.000000
TA advice or declare volatile? 16777216.000000
TA advice or declare volatile? 16777216.000000
TA advice or declare volatile? 16777216.000000
TA advice or declare volatile? 16777216.000000
N: 300000000, <T>: 0.514997 sec, B: 1.454898 GB/s, F: 1.279944 GFLOP/sec
```


(b) dp1 300000000 20 input

Figure 1: dp1 results for both benchmarks

C2

```
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
N: 1000000, <T>: 0.000490 sec, B: 16.263874 GB/s, F: 4.065969 GFLOP/sec
yd2696@instance-1:~/HPML_dotproducts_benchmarks$ ./dp2 1000000 1000
```

(a) dp2 100000 10 input

```
yd2696@instance-1:~/HPML_dotproducts_benchmarks$ ./dp2 300000000 20
The arguments supplied are 300000000  20
TA advice or declare volatile? 67108864.000000
TA advice or declare volatile? 67108864.000000
TA advice or declare volatile? 67108864.000000
TA advice or declare volatile? 67108864.000000
TA advice or declare volatile? 67108864.000000
TA advice or declare volatile? 67108864.000000
TA advice or declare volatile? 67108864.000000
TA advice or declare volatile? 67108864.000000
TA advice or declare volatile? 67108864.000000
TA advice or declare volatile? 67108864.000000
TA advice or declare volatile? 67108864.000000
TA advice or declare volatile? 67108864.000000
TA advice or declare volatile? 67108864.000000
TA advice or declare volatile? 67108864.000000
TA advice or declare volatile? 67108864.000000
TA advice or declare volatile? 67108864.000000
TA advice or declare volatile? 67108864.000000
TA advice or declare volatile? 67108864.000000
TA advice or declare volatile? 67108864.000000
N: 300000000, <T>: 0.263505 sec, B: 2.859163 GB/s, F: 2.515345 GFLOP/sec
```

(b) dp2 300000000 20 input

Figure 2: dp2 results for both benchmarks

C3

```
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
TA advice or declare volatile? 1000000.000000
N: 1000000, <T>: 0.338877 sec, B: 23.607389 GB/s, F: 5.901847 GFLOP/sec
Segmentation fault
```

(a) dp3 100000 10 input

```
yd2696@instance-1:~/HPML_dotproducts_benchmarks$ ./dp3 300000000 20
The arguments supplied are 300000000  20
TA advice or declare volatile? 300000000.000000
TA advice or declare volatile? 300000000.000000
TA advice or declare volatile? 300000000.000000
TA advice or declare volatile? 300000000.000000
TA advice or declare volatile? 300000000.000000
TA advice or declare volatile? 300000000.000000
TA advice or declare volatile? 300000000.000000
TA advice or declare volatile? 300000000.000000
TA advice or declare volatile? 300000000.000000
TA advice or declare volatile? 300000000.000000
TA advice or declare volatile? 300000000.000000
TA advice or declare volatile? 300000000.000000
TA advice or declare volatile? 300000000.000000
TA advice or declare volatile? 300000000.000000
TA advice or declare volatile? 300000000.000000
TA advice or declare volatile? 300000000.000000
TA advice or declare volatile? 300000000.000000
TA advice or declare volatile? 300000000.000000
TA advice or declare volatile? 300000000.000000
TA advice or declare volatile? 300000000.000000
N: 300000000, <T>: 1.187021 sec, B: 11.491170 GB/s, F: 10.109340 GFLOP/sec
```

(b) dp3 300000000 20 input

Figure 3: dp3 results for both benchmarks

C4

```
measurement 989, res:1000000.0
measurement 990, res:1000000.0
measurement 991, res:1000000.0
measurement 992, res:1000000.0
measurement 993, res:1000000.0
measurement 994, res:1000000.0
measurement 995, res:1000000.0
measurement 996, res:1000000.0
measurement 997, res:1000000.0
measurement 998, res:1000000.0
measurement 999, res:1000000.0
N: 1000000, <T>: 0.4035093310860002 sec, B: 19.82605948285978 GB/S, F: 4.9565148
70714945 GFLOP/sec
(hpml_hw1) yd2696@instance-1:~/HPML_dotproducts_benchmarks$ python3 dp4.py 10000
00 1000
```

(a) dp4 100000 10 input

```
measurement 11, res:300000000.0
measurement 12, res:300000000.0
measurement 13, res:300000000.0
measurement 14, res:300000000.0
measurement 15, res:300000000.0
measurement 16, res:300000000.0
measurement 17, res:300000000.0
measurement 18, res:300000000.0
measurement 19, res:300000000.0
N: 300000000, <T>: 132.4236071542 sec, B: 0.3624731347493551 GB/S, F: 0.09061828
368733878 GFLOP/sec
(hpml_hw1) yd2696@instance-1:~/HPML_dotproducts_benchmarks$ python3 dp4.py 30000
0000 20
```

(b) dp4 300000000 20 input

Figure 4: dp4 results for both benchmarks

C5

```
measurement 990, res:2000000.0
measurement 991, res:2000000.0
measurement 992, res:2000000.0
measurement 993, res:2000000.0
measurement 994, res:2000000.0
measurement 995, res:2000000.0
measurement 996, res:2000000.0
measurement 997, res:2000000.0
measurement 998, res:2000000.0
measurement 999, res:2000000.0
0.322269016974009
N: 1000000, <T>: 0.322269016974009 sec, B: 24.82398114195756 GB/S, F: 6.20599528
548939 GFLOP/sec
(hpml_hw1) yd2696@instance-1:~/HPML_dotproducts_benchmarks$ python3 dp5.py 10000
00 1000
```

(a) dp5 100000 10 input

```
measurement 6, res:300000000.0
measurement 7, res:300000000.0
measurement 8, res:300000000.0
measurement 9, res:300000000.0
measurement 10, res:300000000.0
measurement 11, res:300000000.0
measurement 12, res:300000000.0
measurement 13, res:300000000.0
measurement 14, res:300000000.0
measurement 15, res:300000000.0
measurement 16, res:300000000.0
measurement 17, res:300000000.0
measurement 18, res:300000000.0
measurement 19, res:300000000.0
106.22468999969992
N: 300000000, <T>: 106.22468999969992 sec, B: 0.4518723471928758 GB/S, F: 0.1129
6808679821894 GFLOP/sec
(hpml_hw1) yd2696@instance-1:~/HPML_dotproducts_benchmarks$ python3 dp5.py 30000
0000 20
```

(b) dp5 300000000 20 input

Figure 5: dp5 results for both benchmarks

Q1

The reason the first half of the measurements are not taken into account for the computation of the harmonic mean is because of optimizations or caching done by the memory hierarchy. By taking advantage of the optimizations that occur in later runs we can see a more consistent set of execution times. Examples of this come from caches using a LRU (Least Recently Used) design. This keeps more frequently used values nearby, so the benchmark can eventually converge to a more consistent set of results compared to earlier runs

Q2

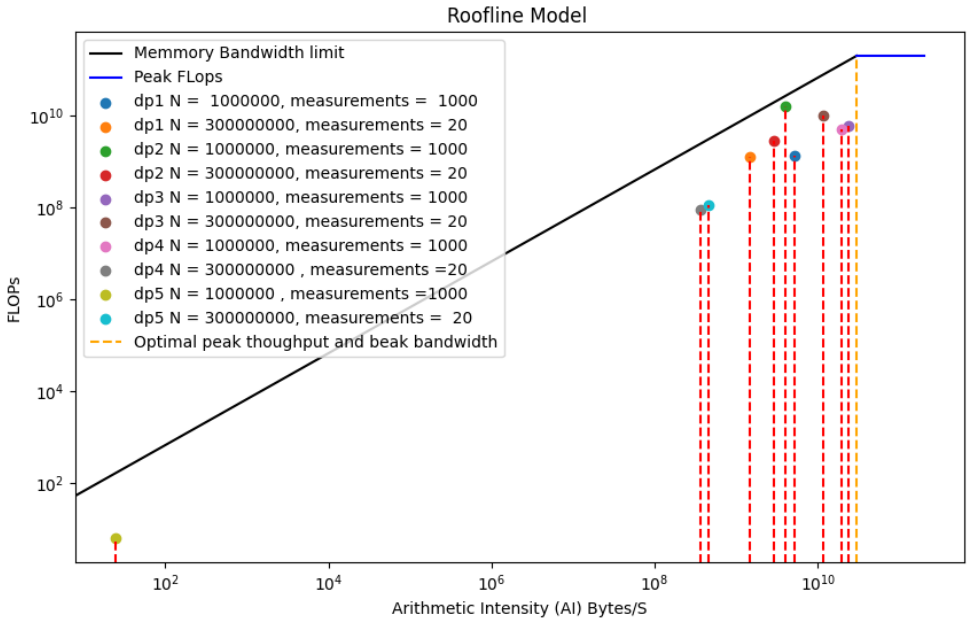

Figure 6: Larger Example Image A

This is what I got for my plots. I used the Bandwidth and Throughput for the x and y coordinates. The x values should have been the same, but I am unsure what happened especially for the one outlier towards the bottom.

Q3

I expected the python implementations to be the slowest overall performance wise. C removes a lot of abstractions from the user, and it is one of the closest languages a developer can get to the hardware excluding assembly. Python sacrifices performance for nicer syntax and other features that I am not as familiar with to go into extreme detail, but I do know it is not able to perform the same as C from a pure performance perspective. For N = 300000000 the python implementations were significantly slower than the C implementations dp4 and dp5 were at 132.4236071542 seconds and 10 6.2246899996992 seconds respectively. For the C benchmarks dp1, dp2, and dp3 it was 0.514997,0.263505, and 1.187021 seconds. A gap was expected, but this was outrageous in my opinion. The numpy benchmark ie dp5 being faster than it's counterpart as the library takes advantage of C's performance to help handle the speed issues.

Q4

For the second benchmark with N = 300000000 and measurements = 20 the values varied the most for the final results. This is because floating point numbers are not exact. The two options are Single precision or Double precision for floating point numbers. Single precision is 4 Bytes (32 bits) and double is 8 Bytes (64 bits) which gives 7 and 15 significant digits of accuracy respectively. From my results the reuslts for dp1 and dp2 are correct for the first, but vary for the second. On dp3 the results seems fine overall, but this one seems intuitive as the it seems more complex than the other libraries. The implementation using the python list data structure had the correct results, but performance was horrific in comparison. For the final benchmark using numpy's dot product method the N = 1000000 was incorrect, but the other one was. It makes sense to an extent as numpy is done via c in the backend, so to see it have similar behavior makes sense in my opinion. It just goes to show the impact a small error can make in some cases.