HPML Assignment 1

Name: Jayraj Pamnani
NetID: jmp10051

Coding Questions:

C1:
Results:

```
Host: b-8-5
[jmp10051@b-8-5 ~]$ gcc -O3 -Wall -o dp1 dp1.c
[jmp10051@b-8-5 ~]$ ./dp1 1000000 1000 > dp1_N1e6_rep1000.txt
[jmp10051@b-8-5 ~]$ ./dp1 300000000 20 > dp1_N3e8_rep20.txt
[jmp10051@b-8-5 ~]$ ls
dp1  dp1.c  dp1_N1e6_rep1000.txt  dp1_N3e8_rep20.txt  ondemand
[jmp10051@b-8-5 ~]$ cat dp1_N3e8_rep20.txt
N: 300000000 <T>: 0.47981026 sec B: 5.00198 GB/sec F: 1.25049 GFLOPS R: 16777216.000
[jmp10051@b-8-5 ~]$ cat dp1_N1e6_rep1000.txt
N: 1000000 <T>: 0.00158536 sec B: 5.04619 GB/sec F: 1.26155 GFLOPS R: 1000000.000
[jmp10051@b-8-5 ~]$
```

C2:
Results:

```
Host: b-8-5
[jmp10051@b-8-5 ~]$ cat dp2_small.txt
N: 1000000 <T>: 0.00047484 sec B: 16.84770 GB/sec F: 4.21192 GFLOPS R: 1000000.000
[jmp10051@b-8-5 ~]$ cat dp2_large.txt
N: 300000000 <T>: 0.23194619 sec B: 10.34723 GB/sec F: 2.58681 GFLOPS R: 67108864.000
[jmp10051@b-8-5 ~]$
```

C3:
Results:

```
[jmp10051@b-8-5 ~]$ cat dp3_N3e8_rep20.txt
N: 300000000 <T>: 0.18532616 sec B: 12.95014 GB/sec F: 3.23754 GFLOPS R: 300000000.000
[jmp10051@b-8-5 ~]$ cat dp3_N1e6_rep1000.txt
N: 1000000 <T>: 0.00032336 sec B: 24.74009 GB/sec F: 6.18502 GFLOPS R: 1000000.000
[jmp10051@b-8-5 ~]$
```

C4:
Results:

```
(pyenv_hpml) [jmp10051@b-10-130 ~]$ cat dp_py_N3e8_rep20.txt
N: 300000000 <T>: 55.84084962 sec B: 0.04298 GB/sec F: 0.01074 GFLOPS R: 16777216.0
(pyenv_hpml) [jmp10051@b-10-130 ~]$ cat dp_py_N1e6_rep1000.txt
N: 1000000 <T>: 0.18616341 sec B: 0.04297 GB/sec F: 0.01074 GFLOPS R: 1000000.0
(pyenv_hpml) [jmp10051@b-10-130 ~]$
```

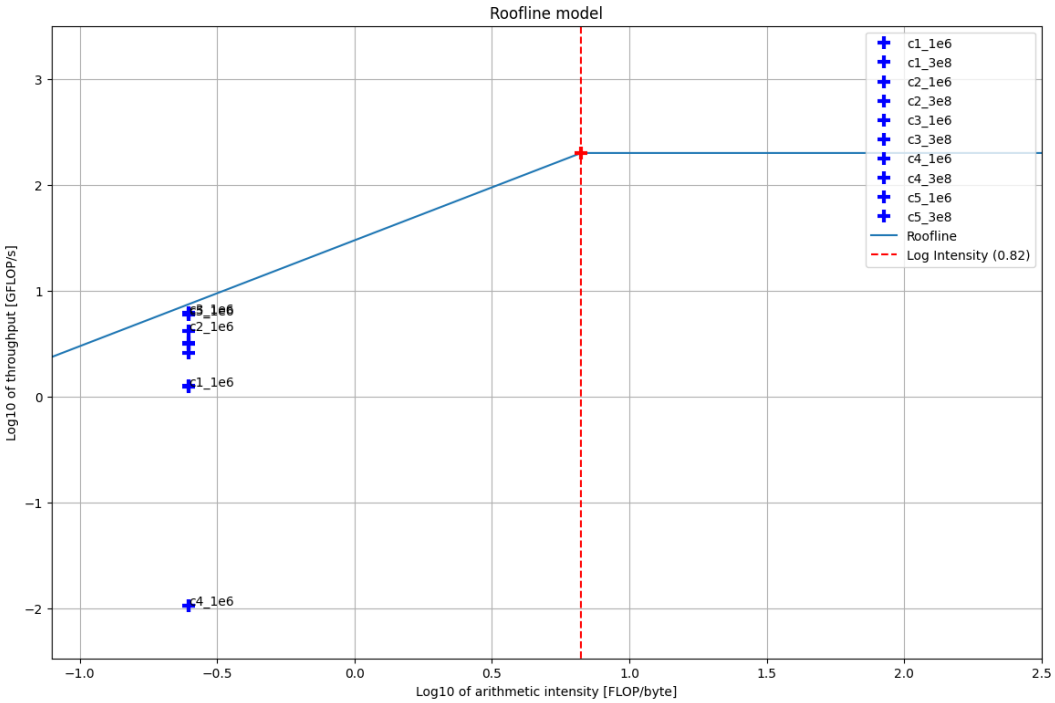
C5:
Results:

```
(pyenv_hpml) [jmp10051@b-8-49 ~]$ python3 dp_py_dot.py 1000000 1000 > dp_py_dot_N1e6_rep1000.txt
(pyenv_hpml) [jmp10051@b-8-49 ~]$ python3 dp_py_dot.py 300000000 20 > dp_py_dot_N3e8_rep20.txt
(pyenv_hpml) [jmp10051@b-8-49 ~]$ cat dp_py_dot_N1e6_rep1000.txt
N: 1000000 <T>: 0.00033579 sec B: 23.82434 GB/sec F: 5.95608 GFLOPS R: 1000000.000
(pyenv_hpml) [jmp10051@b-8-49 ~]$ cat dp_py_dot_N3e8_rep20.txt
N: 300000000 <T>: 0.19050336 sec B: 12.59820 GB/sec F: 3.14955 GFLOPS R: 300000000.000
(pyenv_hpml) [jmp10051@b-8-49 ~]$
```

Theoretical Questions:

Question 1: Explain the consequence of only using the second half of the measurements for the computation of the mean.
Answer: To obtain the mean calculation, utilizing iterations from the latter portion provides more consistent values because caching processes have stabilized during the earlier phase. This pattern is demonstrated in the C1 example, where early iterations show considerable variation. Including only the initial portion would result in an elevated mean due to pre-caching instabilities. Consequently, determining the average from the latter iterations produces more reliable and precise outcomes.

Question 2: Draw a roofline model based on 200 GFLOPS and 30 GB/s. Add a vertical line for the arithmetic intensity.
Draw points for the 10 measurements for the average results for the microbenchmarks.
Answer:



AI = 200 / 30 = 6.66666
Log(6.66666) = 0.8

Question 3: Using the N = 300000000 simple loop as the baseline, explain the difference in performance for the 5 measurements in the C and Python variants.
My Answer:
For N = 300,000,000, we observe that:

- C1 (naive loop):
  <T>: 0.47981026 sec B: 5.00198 GB/sec F: 1.25049 GLOPS

- C2 (loop unrolling):
  <T>: 0.23194619 sec B: 10.34723 GB/sec F: 2.58681 GFLOPS
- C3 (MKL):
  <T>: 0.18532616 sec B: 12.95014 GB/sec F: 3.23754 GFLOPS

Explanation:
Each of the three implementations executes an **identical count of floating-point calculations** (2 operations per element: multiplication plus addition). • Performance variations arise from **optimization efficiency** in hardware utilization:

- C1 processes individual elements sequentially with substantial loop-related overhead.
- C2 minimizes loop costs through unrolling techniques, enabling enhanced **instruction-level parallelism**.
- C3 employs MKL, a sophisticated optimized library utilizing **SIMD instructions, cache optimization strategies, and parallel processing**, achieving superior bandwidth utilization and computational throughput. Therefore, although floating-point operation counts remain identical, **processing time reduces** while both **data transfer rates and computational performance increase** as implementations become more refined.

Question 4:
Check the result of the dot product computations against the analytically calculated result. Explain your findings.
(Hint: Floating point operations are not exact.)
My Answer: For **N = 300,000,000**:

- **C1 (naive loop):**
  <T>: 0.47981026 sec B: 5.00198 GB/sec F: 1.25049 GLOPS
- **C5 (NumPy dot):**
  <T>: 0.19050336 sec B: 12.59820 GB/sec F: 3.14955 GFLOPS

Explanation:
• Theoretically, the dot product calculation between two unit vectors should yield N as the precise result. • Practically, both C1 and C5 employ floating-point computations, which introduce inherent imprecision. Minor rounding discrepancies may emerge, particularly with larger N values.
• The distinction lies **not in** computational accuracy (both face identical floating-point constraints) but in execution efficiency:

- C1 performs sequential element addition through Python/C iteration structures.
- C5 (np.dot) utilizes optimized BLAS libraries implementing vectorized computations and memory-optimized accumulation, delivering significantly higher processing rates. Hence, actual results may deviate minimally from theoretical values, but the essential observation is that C5 produces equivalent results with dramatically improved speed**.**