**High Performance Machine Learning – HW1**

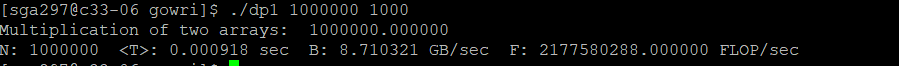
**Name: Sree Gowri Addepalli**

**Net ID: sga297**

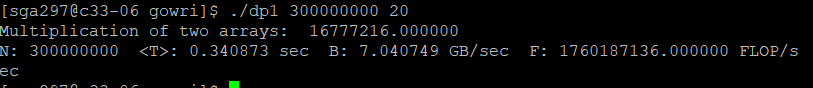
**Date - 9th October 2019**

**C1**

**Exp 1:**



**Exp 2:**

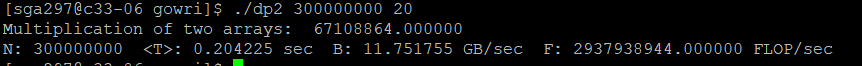


**C2**

**Exp 1:**

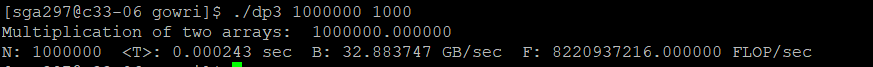


**Exp 2:**

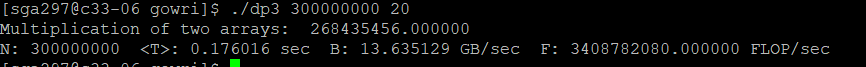


**C3**

**Exp 1:**

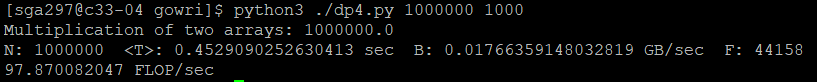


**Exp 2:**

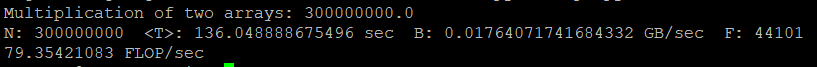


**C4**

**Exp 1:**

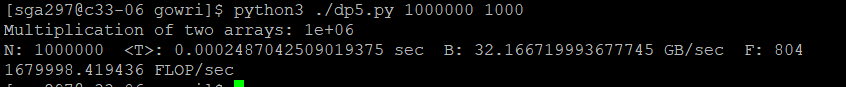


**Exp 2:**

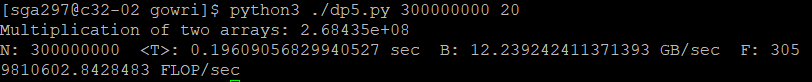


**C5**

**Exp 1:**



**Exp 2:**



**Q1 (3 points):**

**Explain the consequence of only using the second half of the measurements for the computation of the mean.**

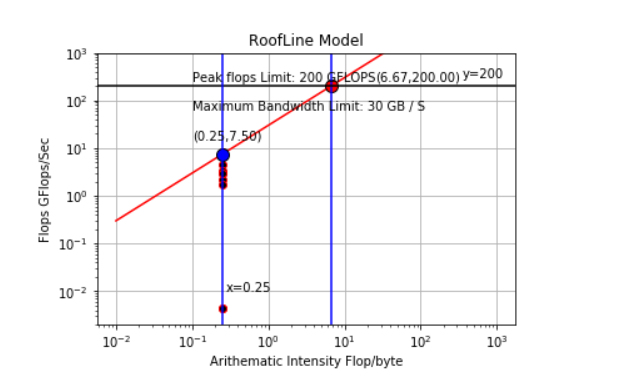
I feel the second half of the measurements are used for computation of mean as they are more consistent with time unlike the first half of the measurements where values can change rapidly due to initial run time overheads like caching and various optimization tasks. So, the first half of the measurements would consist of these times which when taken into consideration for mean computations would not give consistent measurements, which is why the second half of the measurements are taken into consideration for the computation of the mean.

**Q2 (6 points):**

**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.**



**Code:**

flops = [2.1777580288,1.760187136,4.5148641,2.937938,8.220937216,3.40878,0.0044101,0.004410179,8.0416799,3.059810]

bandwidth = [8.710321,7.040749,18.0594569,11.751755,32.883747,13.635129,0.01776635,0.017640717,32.166799936,12.232424]

intensity = [None] \* len(flops)

for i in range(len(flops)):

intensity[i] = flops[i]/bandwidth[i]

import matplotlib.pyplot as plt

import numpy as np

import sys

x = np.logspace(-2, 3,num = 10)

y = 30\*x

plt.loglog(x, y, label='y=30\*x',color='red')

plt.annotate('y=30\*x', xy=(50,20000))

plt.plot(intensity, flops,'o', ms=6,color = 'black',markeredgecolor='red',)

plt.annotate('(%.2f,%.2f)' %(6.67,200), xy=(25,250))

plt.annotate('(%.2f,%.2f)' %(0.25,7.5), xy=(0.1,15))

plt.plot(6.67, 200, 'co', ms=10, label='(6.67,200.00)', color = 'red',markeredgecolor='black')

plt.plot(0.25, 7.5, 'co', ms=10, label='(0.25,7.5)', color = 'blue',markeredgecolor='black')

plt.axvline(x=6.67, color = 'blue')

plt.axhline(y=200, color = 'black')

plt.axvline(x=0.25, color = 'blue')

plt.annotate('y=200', xy=(350,300))

plt.annotate('Peak flops Limit: 200 GFLOPS', xy=(0.1,250))

plt.annotate('Maximum Bandwidth Limit: 30 GB / S', xy=(0.1,60))

plt.annotate('x=0.25', xy=(0.28,0.01))

plt.title('RoofLine Model')

plt.xlabel('Arithematic Intensity Flop/byte', color='black')

plt.ylabel('Flops GFlops/Sec', color='black')

plt.ylim(0, 1000)

plt.grid()

plt.show()

**Q3 (5 points):**

**Using the N=300000000 simple loop as the baseline, explain the difference in performance for the other 5 measurements in the C variants.**

**With** **N = 300000000,**

**T = 0.3408735 sec**

**B = 7.040749 GB/sec**

**F = 1.760187136.000000 GF/S**

1. dp1, N = 1000000, T = 0.0009185 sec, B = 8.710321 GB/sec, F = 2.1777580288.000000 GF/sec

For this, compared to the baseline – it takes more bandwidth and flops. Size is same.

1. dp2, N = 300000000, T = 0.204225 sec, B = 11.7517555 GB/sec, F = 9.37938.000000 GF/sec

For this, compared to the baseline – it takes less time as it moves the array in a step of 4 times faster speed. It takes more bandwidth and flops compared to the baseline. Array size varies by a factor of 300 compared to the baseline.

1. dp2, N = 1000000, T = 0.0004435 sec, B = 18.059456 GB/sec, F = 4.5144864128.000000 GF/sec

For this, compared to the baseline – it takes less time as it moves the array in a step of 4 times faster speed. It takes more bandwidth and flops compared to the baseline. This should run nearly 0.25 times of the baseline as it moves ahead with that speed but due to the increased number of operations, it takes slightly more time.

1. dp3, N = 300000000, T = 0.176016 sec, B = 13.63512 Gb/sec, F = 3.4087828080.000000 GF/sec

For this, compared to the baseline – it takes less time with the usage of MKL library for dot product which speeds up the processor giving better performance. It takes more bandwidth and flops compared to the baseline. Array size varies by a factor of 300 compared to the baseline.

1. dp3, N = 1000000, T = 0.000243 sec, B = 32.883747 Gb/sec, F = 4.78220937216.000000 GF/sec

For this, compared to the baseline – it takes less time with the usage of MKL library for dot product which speeds up the processor giving better performance. It takes more bandwidth and flops compared to the baseline.

**Q3 (6 points):**

**Check the result of the dot product computations against the analytically calculated result.**

**Explain your findings. (Hint: Floating point operations are not exact.)**

There is a difference between the dot product computations and the analytical calculated results for c1 to c3 (dp1 to dp3). This difference is due to how C handles float values and compared to python. C has single precision floating point and python has double precision floating points. So, in single precision floating point, the maximum value of the product it can hold is 16777216.000000 after which any number further gets rounded to this maximum number which is why there is a difference in the computations of the dot product and the analytical calculated results which is not the case in dp4 as python uses double precision which is why it can hold values and the dot product is equal to the analytical calculated result. This is again not the same in dp5 as it uses NumPy which internally uses C and hence single precision value which is why the results of the analytical and dot products do not match.

**Running the code on Prince cluster:**

**1. srun --pty /bin/bash**

**2.**

**dp1 and dp2:**

gcc -O3 -Wall -o dp1 dp1.c ./dp1 1000000 1000

**dp3:**

module load intel/19.0.1 gcc -O3 -Wall -o dp3 dp3.c -I /share/apps/intel/19.0.1/mkl/include -L /share/apps/intel/19.0.1/mkl/lib -lmkl\_rt ./dp3 1000000 1000

**dp4 and dp5:**

module load python3/intel/3.6.3 python dp4.py 1000000 1000