High Performance Machine Learning

Homework 1

Manikanta Srikar Yellapragada
msy290@nyu.edu

06 October 2019

C1: **Experiment 1**: Dot product result is 1000000.000000
N: 1000000
$< T\_avg >$: 0.000917 sec
Bw: 8.725402 GB/s
F: 2181350400.000000 FLOP/s

Experiment 2: Dot product result is 16777216.000000
N: 300000000
$< T\_avg >$: 0.340162 sec
Bw: 7.055452 GB/s
F: 1763863040.000000 FLOP/s

C2: **Experiment 1** Dot product result is 1000000.000000
N: 1000000
$< T\_avg >$: 0.000425 sec
Bw: 18.838442 GB/s
F: 4709610496.000000 FLOP/s

Experiment 2 Dot product result is 67108864.000000
N: 300000000
T avg: 0.217682 sec
Bw: 11.025272 GB/s
F: 2756318208.000000 FLOP/s

C3: **Experiment 1** Dot product result is 1000000.000000
N: 1000000
T avg: 0.000233 sec
Bw: 34.401234 GB/s
F: 8600307712.000000 FLOP/s

Experiment 2 Dot product result is 268435456.000000
N: 300000000
T avg: 0.223257 sec
Bw: 10.749940 GB/s
F: 2687484928.000000 FLOP/s

C4: **Experiment 1** Dot product result is 1000000.0
N: 1000000
T avg: 0.37146557584404943 sec
Bw: 0.021536315934046604 GB/sec
F: 5384078.983511652 FLOP/s

Experiment 2 Dot product result is 300000000.0
N: 300000000
T avg: 145.95954111292957 sec
Bw: 0.016442912753083464 GB/sec
F: 4110728.1882708664 FLOP/s

C5: **Experiment 1** Dot product result is 1000000.0
N: 1000000
T avg: 0.0002443319633603096 sec
Bw: 32.7423391110013 GB/sec
F: 8185584777.750324 FLOP/s

Experiment 2 Dot product result is 300000000.0
N: 300000000
T avg: 0.22834564708173274 sec
Bw: 10.510382092551813 GB/sec
F: 2627595523.1379533 FLOP/s

Q1: During the first half, the program might try to do other tasks like optimization and caching, and there might be Transition look up table misses and page faults, due to which the iterations in the beginning might have abnormal run time. So the consequence of only using the second half of the iterations is that Average time calculated will be accurate unlike the Average time calculated using all the iterations which might include the time spent in the above mentioned tasks.

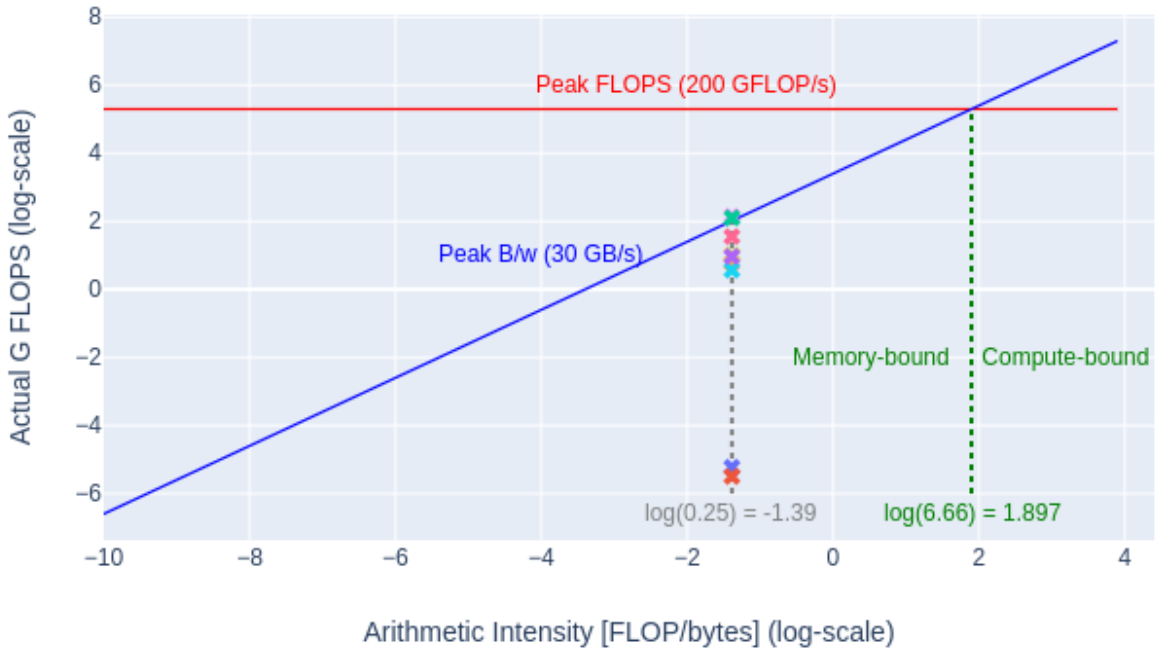

Figure 1: Roofline plot

Q2: Plotted points (x marks) from top to bottom are:

Question	Experiment	Bandwidth
3	1	34.4
5	1	32.74
2	1	18.83
2	2	11.02
3	2	10.75
5	2	10.51
1	1	8.72
1	2	7.05
4	1	0.02
4	2	0.01

I also included a html file which has the interactive version of the above plot (cursor can be hovered over the point (x mark) to know which program and experiment it belongs to).

Q3:

- **dp1, N = 1M** : Since the size of the array is reduced by a factor of 300, running time also differs by a similar amount. Since bandwidth is the rate of data transferred, bandwidth will be higher in N = 1M.
- **dp2, N = 1M** : The implementation of dpunroll contains lesser number of loop runs compared to the baseline addition to a factor of 300. We would expect a decrease in run time by an approximate factor of 1200 (300 due to array size, 4 due to lesser no of loops), we get slightly lesser speed up due to a increase in number of operations per loop.
- **dp2, N = 300M** : As mentioned above, there is a decrement in run time but the speed up obtained is less than 4 times the baseline because of the increase in operations per loop.
- **dp3, N = 1M and 300M**: Due to optimized implementation of the dot product with MKL library, performance is boosted by a huge amount resulting in much better run time.
- **Memory efficiency**: In all the experiments, 300M is consistently slower than 1M, and as a consequence has lesser bandwidth and memory efficiency than 1M.

Q4: Single precision floting point in C cannot hold some values and whenever they occur in the computations, they are rounded down to the limit of this floating point.

- **dp1** The answer in dp1 is 16777216.000000 as the floating point single precision cannot handle the next value so rounds it down to the same value. When 1 is added to 16777216.000000, error is -1 so the result remains the same. Even in the subsequent loops, answer remains the same (16777216.000000).
- **dp2** In dp2, 4 is added to the answer in every iteration. Adding 4 to 67108864.000000 has an error of -4, so the result remains the same and will stay the same even in the consequent iterations of the loop.
- **dp3**: In dp3 error of -8 occurs, so the result is 2687484928.000000.
- **dp4**: In dp4 python uses double precision floating point which can handle all the values in our experiments, hence computed value is equal to the analytically calculated value.
- **dp5**: In dp5 and 300M, the output varied depending on the version of numpy installed in the compute node, and the specification of compute node obtained in prince. The answers in my case were correct in all the runs of this code.
- **Note to the grader**: The output of dp5 could be 268435456.000000 depending on the version of numpy and compute node.