Note: GCP VM used exactly as specified in instructions.
dp1:
N: 1000000 <T>: 0.001538 sec B: 4.843593 GB/sec F: 1.300192 GFLOP/sec
N: 300000000 <T>: 0.468355 sec B: 4.772391 GB/sec F: 1.281079 GFLOP/sec

Result:
1000000.000000
16777216.000000

dp2:
N: 1000000 <T>: 0.000489 sec B: 15.242188 GB/sec F: 4.091544 GFLOP/sec
N: 300000000 <T>: 0.237426 sec B: 9.414210 GB/sec F: 2.527108 GFLOP/sec

Result:
1000000.000000
67108864.000000

dp3:
N: 1000000 <T>: 0.000097 sec B: 76.848416 GB/sec F: 20.628840 GFLOP/sec
N: 300000000 <T>: 0.055105 sec B: 40.562300 GB/sec F: 10.888360 GFLOP/sec

Result:
1000000.000000
300000000.000000

dp4:
N: 1000000 <T>: 0.4123653273582458 sec B: 0.01806791236463747 GB/sec F:
0.004850068294569497 GFLOP/sec
N: 300000000 <T>: 128.88165698051452 sec B: 0.017342841731271985 GB/sec F:
0.004655433628469824 GFLOP/sec

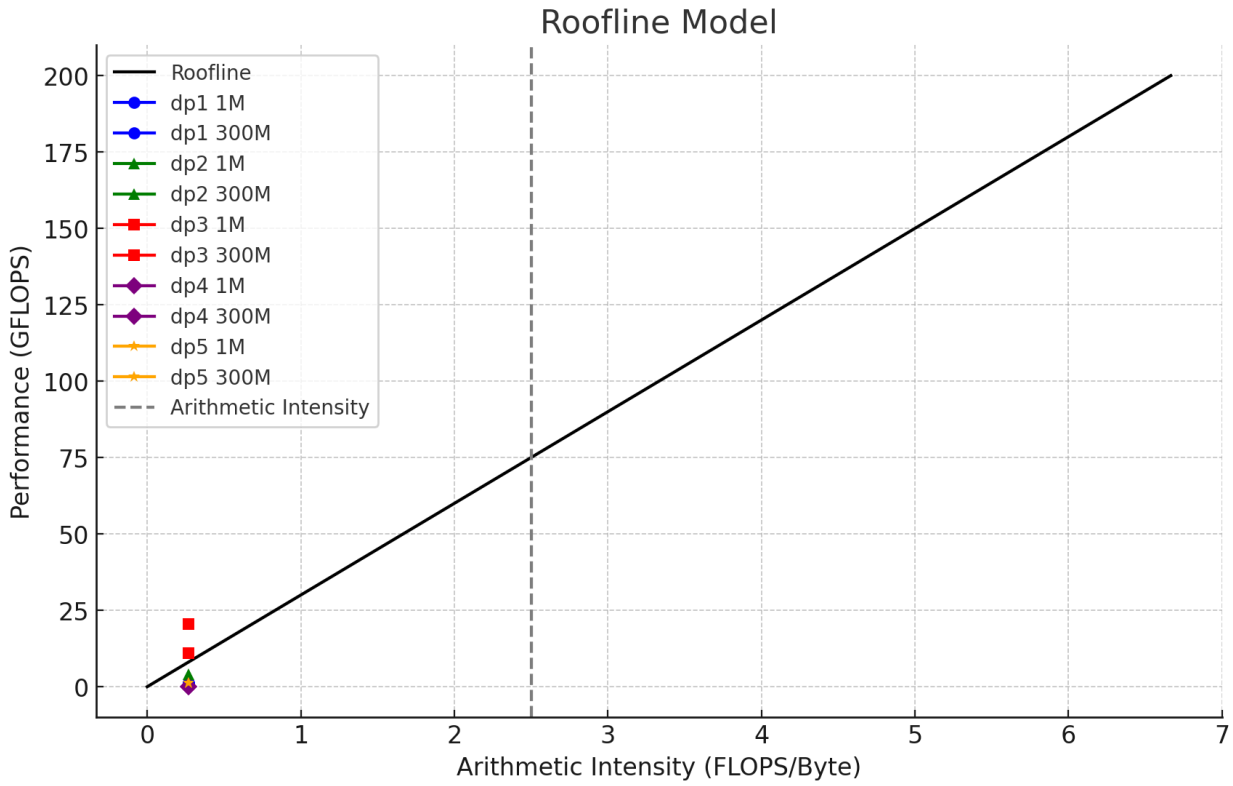
Result:
1000000.0
300000000.0

dp5:
N: 1000000 <T>: 0.001518354892730713 sec B: 4.907008652979803 GB/sec F:
1.317215105358579 GFLOP/sec
N: 300000000 <T>: 0.46211609840393064 sec B: 4.836823877802688 GB/sec F:
1.2983750232296527 GFLOP/sec

Result:
1000000.0
16777216.0

Q1:
The initial executions of the computations may be slower due to various startup costs, such as optimization routines and establishing stable thermal conditions. The consequence of using only the second half of the measurements is that we only focus on the performance under a steady state i.e. one that is more representative of ongoing operation.

Q2:



Q3:
Dp1: Baseline C implementation, moderate efficiency in terms of bandwidth and FLOPS due to direct memory access and the simplicity of operations, worse performance than optimized C but better than Python
Dp2: Significant improvement over dp in terms of bandwidth and FLOPS compared to dp1, as loop unrolling allows for more computations done per loop iteration. Doubles in performance from dp1
Dp3: Even more of an improvement from dp2, dp3. Highest performance in terms of bandwidth and FLOPS/sec among the C implementations. MKL seems to be highly optimized for the architecture (presumably intel), causing the dramatic improvement.
Dp4: Baseline Python, significantly slower than any C implementation, as Python is interpreted. Lowest bandwidth and FLOPS/sec

Dp5: Uses np.dot which has wrapped C code for operations, allowing for comparable performance to basic C implementation.

Q4:
Analytically, we would expect the computation to be 1000000.0 for N = 1000000, and 300000000.0 for N = 300000000. The results above show that the calculations are correct in every case for N = 1000000. For N = 300000000, results vary. The calculation is correct in dp3 and dp4, suggesting that the accumulation of floating point errors are minimized somehow. For dp1 and dp5, we get 16777216.0. This number is equivalent to 2^24, and could arise due to an accumulation of rounding errors over many iterations. This is probably also the case for dp2, where the loop unrolling potentially causes larger increments in the summation (which results in 67108864.0), which leads to the loss of precision being different than the above case.