Homework 1

Vinay Swamy

```
# compile software
gcc -Wall -O3 -o dp1 dp1.c
gcc -Wall -O3 -o dp2 dp2.c
. /opt/intel/oneapi/setvars.sh
gcc -Wall -O3 -o dp3 \
  -I /opt/intel/oneapi/mkl/2022.2.0/include dp3.c \
  -L /opt/intel/oneapi/mkl/2022.2.0/lib -lmkl_rt
```

C1

```
float dp(long N, float *pA, float *pB) {
    float R = 0.0;
    int j;
    for (j=0;j<N;j++)
        R += pA[j]*pB[j];
    return R;
}
```

FLOPs: 2 per iteration(add, multiply), so 2N flops

traffic: 2 accessing vectors @ 4 bytes each so 8N bytes(assuming R is kept in the cache)

Arithmetic Intensity: 2N/8N = 1/4

```
./dp1 1000000 1000
```

N: 1000000 T: 0.000753441500 sec B: 10.617944458860 GB/sec F: 2.654486114715 GFLOP/s dp_out: 1000000.000000000000

```
./dp1 300000000 10
```

N: 300000000 T: 0.209115047600 sec B: 11.476935914200 GB/sec F: 2.869233978550 GFLOP/s dp_out: 16777216.000000000000

C2

```
float dpunroll(long N, float *pA, float *pB) {
  float R = 0.0;
  int j;
  for (j=0;j<N;j+=4)
    R += pA[j]*pB[j] + pA[j+1]*pB[j+1] \
           + pA[j+2]*pB[j+2] + pA[j+3] * pB[j+3];
return R; }
```

FLOPS: 4 mults, 4 adds, so 8N/4 =2N ops traffic: 8 float reads, so 32N/4 =8N bytes Arithmetic Intensity: 2N/8N = 1/4

```
./dp2 1000000 1000
```

N: 1000000 T: 0.000359006544 sec B: 22.283716365906 GB/sec F: 5.570929091476 GFLOP/s dp_out: 1000000.000000000000

```
./dp2 300000000 10
```

N: 300000000 T: 0.131516230800 sec B: 18.248698167527 GB/sec F: 4.562174541882 GFLOP/s dp_out: 67108864.000000000000

C3

```
#include <mkl_cblas.h>
float bdp(long N, float *pA, float *pB) {
  float R = cblas_sdot(N, pA, 1, pB, 1);
return R; }
```

FLOPS/traffic/AI should be the same as before

```
./dp3 1000000 1000
```

N: 1000000 T: 0.000047370568 sec B: 168.881234440761 GB/sec F: 42.220308610190 GFLOP/s dp_out: 1000000.000000000000

```
./dp3 300000000 10
```

N: 300000000 T: 0.052179587800 sec B: 45.994997300458 GB/sec F: 11.498749325114 GFLOP/s dp_out: 300000000.000000000000

C4

```
python dp4.py 1000000 1000
```

N: 1000000 T: 0.283628928714 sec B: 0.028205867561791904 GB/sec F: 0.007051466890447976 GFLOP/s

```
python dp4.py 300000000 10
```

N: 300000000 T: 69.98567886660001 sec B: 0.03429273015375974 GB/sec F: 0.008573182538439935 GFLOP/s

C5

```
python dp5.py 1000000 1000
```

N: 1000000 T: 0.00019073346599999999 sec B: 41.94334726764731 GB/sec F: 10.485836816911828 GFLOP/s

```
python dp5.py 300000000 10
```

N: 300000000 T: 0.1432159034 sec B: 16.757915448096806 GB/sec F: 4.189478862024202 GFLOP/s

Q1

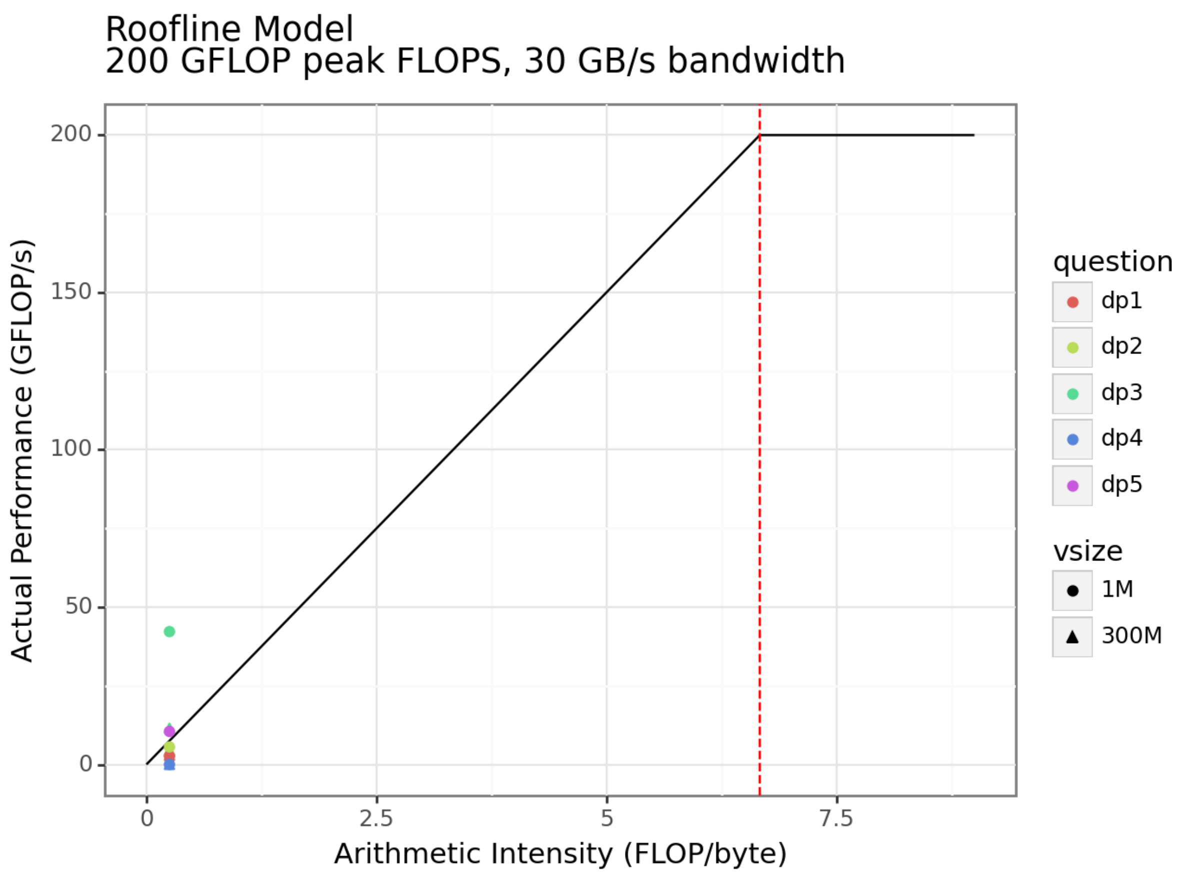
One reason would be for would be for when running the python code, the interpreter must compile the code to byte code before running it. Another reason, is that on the initial run, there may already be data in the cache, so that will need to be cleared before the actual computation can be done.

Q2

```
import pandas as pd
import plotnine as pn
line_df = pd.concat([pd.DataFrame().assign(x = list(range(10))).assign(x = lambda x: x.x.a
flop_df =pd.DataFrame().assign(
  x = [.25]*10,
  y= [2.654486114715,2.869233978550,
      5.570929091476, 4.562174541882,
      42.220308610190, 11.498749325114 ,
      0.007051466890447976, 0.008573182538439935,
      10.485836816911828, 4.189478862024202 ],
  question = ["dp1", "dp1", "dp2", "dp2", "dp3", "dp3", "dp4", "dp4", "dp5", "dp5"],
  vsize = ["1M", "300M", "1M", "300M", "1M", "300M", "1M", "300M", "1M", "300M"]

)

line_df["y"][line_df["y"] >=200]=200
(
  pn.ggplot() +
  pn.geom_line(data = line_df,mapping = pn.aes(x = 'x', y = 'y')) +
  pn.geom_point(data = flop_df, mapping = pn.aes(x = 'x', y = 'y', color = 'question', sha
  pn.geom_vline(xintercept = 6.666666, linetype = "dashed", color = "red") +
  pn.ggtitle("Roofline Model\n200 GFLOP peak FLOPS, 30 GB/s bandwidth") +
  pn.xlab("Arithmetic Intensity (FLOP/byte)") +
  pn.ylab("Actual Performance (GFLOP/s)")+
  #pn.xlim([0,1.1])+
  # pn.scale_x_log10() +
  # pn.scale_y_log10() +
  pn.theme_bw()
)
```

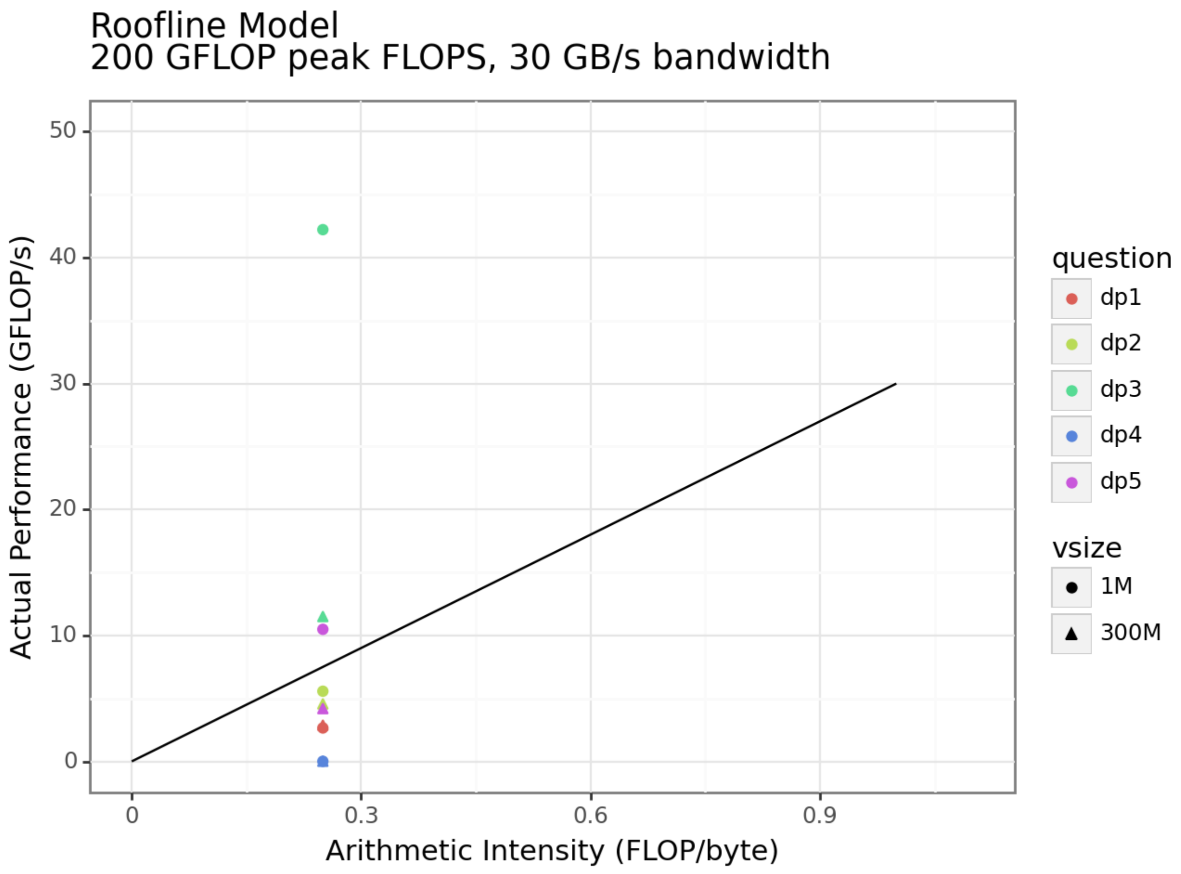


```
<Figure Size: (640 x 480)>
```

Zoomed in

```
(
  pn.ggplot() +
  pn.geom_line(data = line_df,mapping = pn.aes(x = 'x', y = 'y')) +
  pn.geom_point(data = flop_df, mapping = pn.aes(x = 'x', y = 'y', color = 'question', sha
  pn.geom_vline(xintercept = 6.666666, linetype = "dashed", color = "red") +
  pn.ggtitle("Roofline Model\n200 GFLOP peak FLOPS, 30 GB/s bandwidth") +
  pn.xlab("Arithmetic Intensity (FLOP/byte)") +
  pn.ylab("Actual Performance (GFLOP/s)")+
  pn.xlim([0,1.1])+
  pn.ylim([0,50])+
  # pn.scale_x_log10() +
  # pn.scale_y_log10() +
    pn.theme_bw()
)
```

```
/data/vss2134/miniconda/lib/python3.9/site-packages/plotnine/geoms/geom_path.py:98: Plotnine
/data/vss2134/miniconda/lib/python3.9/site-packages/plotnine/layer.py:364: PlotnineWarning: 
```



```
<Figure Size: (640 x 480)>
```

Q3

C2 - Given that the computation is memory bound, the unrolling of the loop allows more memory to be loaded to the CPU at once, and thus the computation is faster.

C3 - The MKL library is able to use SIMD/vectorized instructions to perform the computation faster than the baseline.

C4 - The slowest of all, the overhead of compiling and interpreting the loop over and over again leads to huge slowdowns.

C5 - This uses numpy, and since numpy is written in C and fortran, it circumvents the overhead of the python interpreter, and may have similar optimization to the MKL library.

Q4

Across the runs, the calculations are fine for the 1M run, but for the 300M, looks like there's an overflow error for C1 and C2; C3 seems to work fine, so the MKL libary must be doing something to prevent overflow errors.