Lab 1

Theoretical questions are identified by Q while coding exercises are identified by C.
Submit a tar-archive named with your nyu-email (e.g. .tar => uf5.tar) that unpacks to
/dp1.c
/dp2.c
/dp3.c
/dp4.py
/dp5.py
<email-id/results.pdf
to NYU classes.

The pdf contains the outputs of the programs and the answers to the questions.

C1 (6 points):

Write a micro-benchmark that investigates the performance of computing the dot-product in C that takes two arrays of 'float' (32 bit) as input. The dimension of the vector space and the number of repetitions for the measurement are command line arguments, i.e. a call 'dp1 1000 10' performs 10 measurements on a dot product with vectors of size 1000. Initialize fields in the input vectors to 1.0.

```
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;
}
```

Name the program dp1.c and compile with

gcc -O3 -Wall -o dp1 dp1.c

Make sure the code is executed on a Prince node with Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00GHz or similar and enough RAM, the 300000000 size runs should not be killed by the system!

Measure the execution time of the function with clock_gettime(CLOCK_MONOTONIC).
Measure the time for $N = 1000000$ and $N = 300000000$.
Perform 1000 repetitions for the small case and 20 repetitions for the large case.
Compute the appropriate mean for the execution time for the second half of the repetitions.

For the average times, compute the bandwidth in GB/sec and throughput in FLOP/sec,
print the result as

N: 1000000 : 9.999999 sec B: 9.999 GB/sec F: 9.999 FLOP/sec
N: 300000000 : 9.999999 sec B: 9.999 GB/sec F: 9.999 FLOP/sec

C2 (3 points):

Perform the same microbenchmark with

```
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;
}
```

C3 (3 points):

Perform the same microbenchmark with MKL (Intels library), you may need to install a 'module' on prince to access MKL.

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

C4 (6 points):

Implement the same microbenchmark in python, using numpy arrays as input

```
A = np.ones(N,dtype=np.float32)
B = np.ones(N,dtype=np.float32)
```

for a simple loop

```
def dp(N,A,B):
    R = 0.0;
    for j in range(0,N):
        R += A[j]*B[j]
    return R
```

C5 (4 points):

Perform the same measurements using 'numpy.dot'.

Q1 (3 points):

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

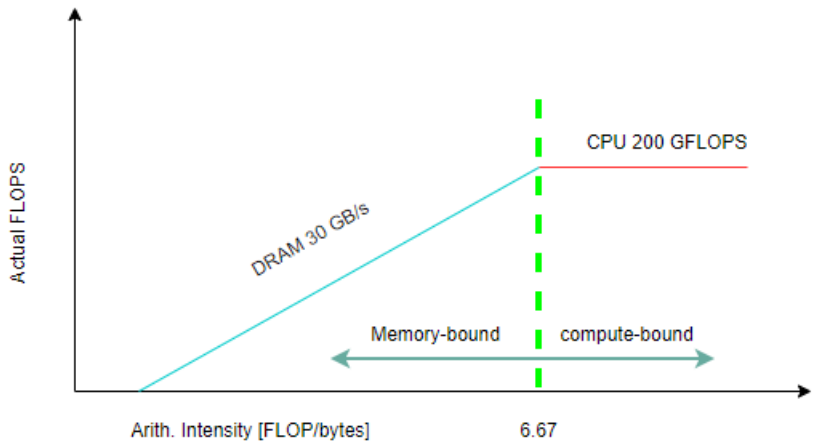
We can show the results

	N	T (s)	B (GB/s)	GFLOPS
dp1	1000000	0.0002883	27.752	6.938
dp1	300000000	0.1004336	23.893	5.974
dp2	1000000	0.0003279	38.372	9.893
dp2	300000000	0.0563335	42.603	10.651
dp3	1000000	0.000341	47.329	11.397
dp3	300000000	0.0479339	50.0689155	12.517
dp4	1000000	0.486923	0.0164	0.0423
dp4	300000000	145.3278	0.0167	0.0412
dp5	1000000	0.000207	26.890	7.342
dp5	300000000	0.13897	25.893	6.936

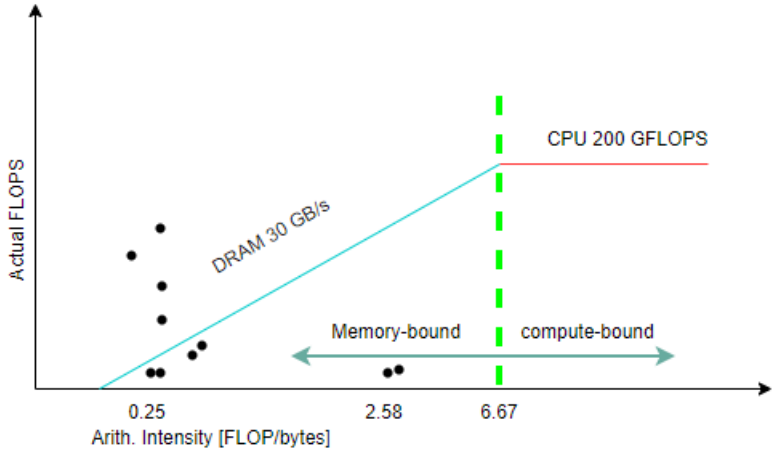
At the beginning, it probably has some uncertainties. The second half measurement will become stable because at the beginning, the CPU may not reach the full power. Another reason is that we need to measure the latency and Bandwidth and GFLOPS, the second half will give us more accurate result.

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.



Q3 (5 points):

Using the $N = 300000000$ simple loop as the baseline, explain the difference in performance for the other 2 measurements in the C variants.

	N	T (s)	B (GB/s)	GFLOPS
baseline	300000000	0.1004336 s	23.893 GB/s	5.974
dp2	300000000	0.0563335s	42.603 GB/s	10.651
dp3	300000000	0.0479339 s	50.0689155GB/s	12.5172289

When we compare baseline and dp3, we can notice that using MKL library can dramatically reduce computation time. The bandwidth and GFLOP increase too. MKL library can process operation very quick. When we compare baseline and dp2, we can notice that dp2 is faster due to the different function. The baseline need to execute 3,000,000,000 "+" and "*". dp2 is faster than baseline because dp2 executes 1/4 loops less than dp1. And for loop instruction, it needs more time to execute.

Q4 (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.)

From the above result, we can notice that dot product is much faster than simple for loop. The np.dot does the actual arithmetic operation. However, in Python, when we write the simple for loop, it is a compiled code which is very slow. Simple for loop is a element-wise multiplication and no.dot is vectorization calculation. So, np.dot is very fast.