High Performance Machine Learning
Homework Assignment 1

Dr. Kaoutar El Maghraoui

Due Date: February 16 2026 Spring 2026 Max Points: 100

Use the Google Cloud platform (GCP) or your own machine. Make sure that your Google VM or your machine has at last 32GB of RAM to be able to complete the assignments. GCP coupons will be shared with you.

Instructions:
Theoretical questions are identified by Q<number> while coding exercises are identified by C<number>. Submit a tar-archive named with your Columbia UNI (e.g. <UNI>.tar) that unpacks to

```
- /dp1.c
- /dp2.c
- /dp3.c
- /dp4.py
- /dp5.py
- /results.pdf
```

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

Important Note on Benchmark Correctness (Dead Code Elimination):
When implementing the microbenchmarks in this assignment, be aware of Dead Code Elimination (DCE) performed by modern optimizing compilers. If the result of the dot-product computation is not used, the compiler may partially or completely remove the computation, leading to unrealistically high bandwidth and FLOP/s measurements.

To ensure correct and meaningful performance measurements, the result of the dot-product computation must be consumed in a way that prevents the compiler from optimizing it away. Acceptable approaches include:

1. Declare the result as `volatile`

```
volatile float dot_product = dp(N, pA, pB);
```

This tells the compiler not to eliminate or optimize away the variable.

2. Print the result

```
float dot_product = dp(N, pA, pB);
printf("%f\n", dot_product);
```

Printing forces the computation to happen.

C1 20 points

Write a micro-benchmark that investigates the performance of computing the dot-product 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 platform that has 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, and print the result as

```
N: 1000000  <T>: 9.999999 sec  B: 9.999 GB/sec   F: 9.999 FLOP/sec
N: 300000000  <T>: 9.999999 sec  B: 9.999 GB/sec   F: 9.999 FLOP/sec
```

C2 15 points

Perform the same microbenchmark using loop unrolling.

```
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 15 points

Perform the same microbenchmark with MKL (Intel library), you may need to install a 'module' 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 — 10 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 — 10 points

Perform the same measurements using **'numpy.dot'**.

Q1 — 5 points

Explain the rationale and expected consequence of only using the second half of the measurements for the computation of the mean execution time. Moreover, explain what type of mean is appropriate for the calculations, and why.

Q2 — 15 points

Draw a roofline model assuming a peak compute performance of 200 GFLOP/s and a peak memory bandwidth of 30 GB/s, and plot the model using matplotlib or an equivalent package. Each microbenchmark refers to one of the five coding tasks (C1–C5). For each microbenchmark, collect performance results at two problem sizes (N = 1,000,000 and N = 300,000,000), yielding a total of 10 benchmark configurations. Each configuration should be run multiple times; report the average performance results (e.g., FLOP/s and memory bandwidth) across repetitions, and plot one point per configuration on the roofline model using these averaged values, along with a vertical line indicating arithmetic intensity

Based on your plotted measurements, explain clearly whether the computations are compute or memory bound, and why. Discuss the underlying reasons for why these computations differ or don't across each microbenchmark.

Lastly, identify any microbenchmarks that underperform relative to the roofline, and explain the algorithmic bottlenecks responsible for this performance gap.

Q3 — 5 points

Using the N = 300000000 simple loop as the baseline, explain the the difference in performance for the 5 measurements in the C and Python variants. Explain why this occurs by considering the underlying algorithms used.

Q4 5 points

Check the result of the dot product computations against the analytically calculated result. Explain your findings, and why the results occur. (Hint: Floating point operations are not exact.)