CSCI-GA.3033-022 - Lab1

This lab is intended to be performed **individually**, great care will be taken in verifying that students are authors of their own submission and that the exam is different from previous courses.

Exercises need to be executed on the **Prince NYU cluster** in the standard compute nodes.

Theoretical questions are identified by **Q<number>** while coding exercises are identified by **C<number>**.

C1 Array Reduction – C code [6 points]

Write a serial microbenchmark to estimate the peak memory bandwidth and FLOP/s from actual experimental measurements. The code should do the following:

- Initialize the array with:
  ```
  for ( i = 0; i < ARRAY_SIZE; ++i)
          array[i] = (double)i/3;
  ```
  Where *i* is an integer, *sum* is a double and *array* is an array of doubles.
- Set the size of the array to several GB in order to obtain peak bandwidth.
- The operation to measure is the following:
  ```
  for (i = 0; i < ARRAY_SIZE; i++)
          sum += array[i]*2;
  ```
- Use `clock_gettime(CLOCK_MONOTONIC, … )` to measure the elapsed time, but do not include the initialization phase.
- Use a loop to measure the best performance out of 10 runs.
- Compile the code with the following:
  - `gcc -W -Wall -O3 ./lab1-c1-c2.c -o lab1-c1-c2 && ./lab1`
- Make sure there are not compilation warnings.

C2 Loop Unrolling – C code [6 points]

Add a different version of the main operation in the same binary.

- This version is optimized using the loop unrolling technique:

```
for (i = 0; i < ARRAY_SIZE; i+=8) {
            sum +=array[i]*2;
            sum +=array[i+1]*2;
            sum +=array[i+2]*2;
            sum +=array[i+3]*2;
            sum +=array[i+4]*2;
            sum +=array[i+5]*2;
            sum +=array[i+6]*2;
            sum +=array[i+7]*2;
      }
```

- Take the lowest elapsed time for C1 and for C2 (10 trials each) without initialization.
- Report separate time, bandwidth and flops for C1 and for C2. The output of your *./lab1* executable should look like this:

```
*C1*

Time: 999.99 secs

Bw: 9999.99 GB/s

FLOPS: 9999.99 GFLOP/s


*C2*

Time: 999.99 secs

Bw: 9999.99 GB/s

FLOPS: 9999.99 GFLOP/s
```

Q1: Question [2 points]

- Why is the loop unrolling version faster?

Q2: Question [2 points]

- What can be done to obtain a higher memory BW using all the resources of the CPU?

Q3: Roofline model [6 points]

Make sure to run all experiments on a system with the following CPU (NYU Prince cluster node):

```
Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00GHz
```

Draw a roofline model including:

- CPU peak GFLOPS line (Intel specs)
- DRAM BW line in GB/s (Intel specs)
- A.I. vertical line
- Two dots for your best C1 and C2 GFLOP/s measurements

Inferencing in Python and C

In this part of the lab we explore the performance of an algorithm that does inferencing of a single sample using the first two layers of a fully connected neural network.

For each layer $l \in [0,1]$, denote with W_l the matrix of the weights, with x_l the vector of the input and with z_l the vector of the output (after the activation function has been computed). Assume the bias to be a vector of zeroes that can be ignored.

The dimension of the network are as follows:

1) The first layer takes as input an image of size 256 x 256 pixel (`65536 features`) and is composed of 4,000 neurons with a ReLU activation function.
2) The second layer takes as input the 4,000 outputs of the first layer and is composed of 1000 neurons with a ReLU activation function.

Q4: Question [2 points]

- What are the dimensions of W_l, x_l, and z_l for each layer l?

Q5: Question [2 points]

- Assuming double precision operations, what are W_l, x_l, and z_l sizes in memory for each layer l?

[Hint: refer to a C implementation of the code.]

C3: Python inferencing [6 points]

Implement the inferencing for the network described above in plain Python (no Pytorch).

- Use the following initialization function for vectors or matrices x, z or W:

  x[i, j] = 0.4 + ((i+j) % 40 - 20)/40.0

- Report the execution time excluding the initialization of the vectors and matrices.
- Report the checksum value *S* as the sum of the elements in the output vector of the last layer (z_1)

C4: Numpy inferencing [6 points]

Use Numpy to speed up the dense computation of the output of each layer.

- Report the execution time excluding the initialization of the vectors and matrices.
- Report the speedup with respect to the execution time of **C3.**
- Verify that the checksum value *S* is the same as in **C3** (excluding the floating point rounding error)

[Hint: use numpy.clip() and numpy.dot().]

C5: C inferencing [6 points]

Write the equivalent of the reference implementation in C and compare the performance and the results with the Python versions. Initialize all the elements of the matrices W_l and the input vector x_0 with the same function described in C2.

- Compile with the makefile in `/scratch/am9031/CSCI-GA.3033-022/lab1/Makefile`
- Report the speedup with respect to the execution time of **C3**
- Verify that the checksum value *S* is the same as in **C3** (excluding the floating point rounding error)

C6: Intel MKL inferencing [6 points]

Same problem as in C5 but this time use a math library.

- Use the Intel MKL library to perform the matrix vector multiplication
- To load the MKL library run the command:

```
module load intel/17.0.1
```

- Compile with the makefile in `/scratch/am9031/CSCI-GA.3033-022/lab1/Makefile`
- Report the execution time excluding the initialization of the vectors and matrices.
- Report the speedup with respect the execution time of **C3**
- Verify that the checksum value *S* is the same as in **C3** (excluding the floating point rounding error)

[Hint: https://software.intel.com/en-us/mkl-developer-reference-c-cblas-gemv]

Grading rules

- Total is 50 points
- Late submission is -3 points for every day, maximum 3 days.
- Non-compiling code: 0 points

Appendix – Submission instructions

Submission through NYUClasses.

Please submit a targz archive with file name *<your-netID>.tgz* with a folder named as your netID (example: am9031/) containing the following files:

- A file named lab1-c1-c2.c containing exercises C1 and C2 executed in sequence
- A file named lab1-c3-c4.py containing exercises C3 and C4 executed in sequence
- A file named lab1-c5-c6.c containing exercises C5 and C6 executed in sequence
- A file named lab1.pdf with answers to questions Q1 to Q5 and all the outputs of the exercises C1 to C6
- Failing to follow the right directory/file name specification is -1 point. Failing to have programs executed in sequence is -1 point.