Image Processing Library

Navneel Mandal & Mayank Singh Kushwaha

Feb 17th, 2019

1 Introduction

It is the first Assignment of the COP290 course. The whole Assignment is sub divided into 3 sub-tasks, each of which have different objectives. The first sub-task aims at creating some functions to use for the second and the third sub-task. The second sub-task aims at using different libraries for doing matrix multiplication, while the third sub-task implements the LeNet Architecture to identify handwritten digits from the MNIST database.

2 Creating Functions for later use

For the first sub-task we create many functions which are to be used later in the subsequent sub tasks. The table below shows the function and the arguments it takes when running.

2.1 Running this sub-task

To run this sub-task independently, we first compile the necessary files by running the command

```
make main
```

Once compiled, we get our .out file as main.out,which can be used to execute the functions with correct parameters. A list of all the commands has been provided in Table 3.

A very detailed description of each of the functions in provided in the README.md. It is encouraged to go through that file.

2.2 Output

The default output comes in the terminal itself but the display may be illegible for bigger size matrix. Therefore there is a Output.txt file, where every output of the main.out file is stored.

3 Comparing Matrix Multiplication

This is the second sub-task, where we use different libraries to compare the speeds of various libraries for matrix multiplication. We compare 3 libraries:

1. Pthreads implementation of Matrix Multiplication
2. OpenBlas Implementation
3. Intel MKL Implementation

3.1 Installation

3.1.1 Pthreads

Pthreads comes installed in UNIX systems, so there is no need for further installation. We, for this assignment, settled on 5 as the number of threads after running for various values and seeing that this one gave the lowest time for the majority of the sizes we tested for. For compilation purposes, though the command

```
<your_command> -lpthreads
```

must be used. Although in this case, just running

```
make plot_pthreads
```

will suffice.

3.1.2 openBlas

To install openBlas, one can directly execute the shell script provided

```
./install_openblas.sh
```

For compilation, similarly

```
<your_command> -lopenblas
```

must be used. Although in this case, just running

```
make plot_openblas
```

will suffice.

3.1.3 MKL

To install MKL, run the script [1]

```
./install_mkl.sh
```

Do note that the job isn't finished, and one must follow further steps by setting the environmental variables as written in the README.md [3]. To compile, one must use the INTEL Link Advisor[2]. One must change the contents of the 'intel' variable. Then just run

```
make plot_mkl
```

to compile the program.

3.2 Performance Comparison

On running the libraries with suitable arguments, all of which can be found in this table, we find that MKL and OpenBlas do extremely well in comparison to our Pthreads(Figure 1).

On close comparison of openBlas and MKL, we see that MKL in fact does better than openBlas, and that too with a good margin(Figure 2.)

Table 1 contains few values which give time comparison v/s matrix size for the 3 libraries.

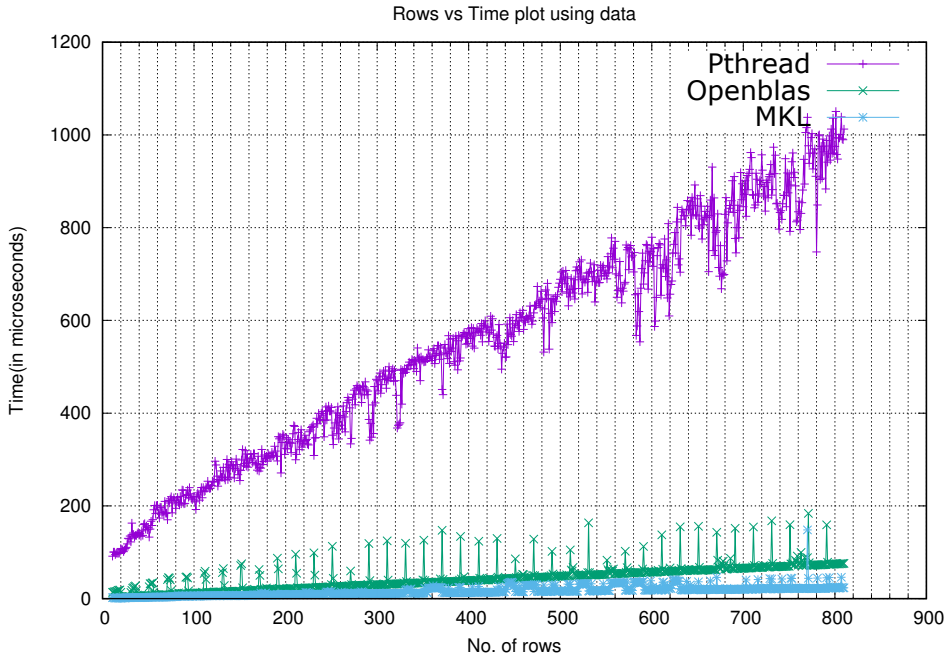

Figure 1: Comparing Pthreads, openBlas and MKL

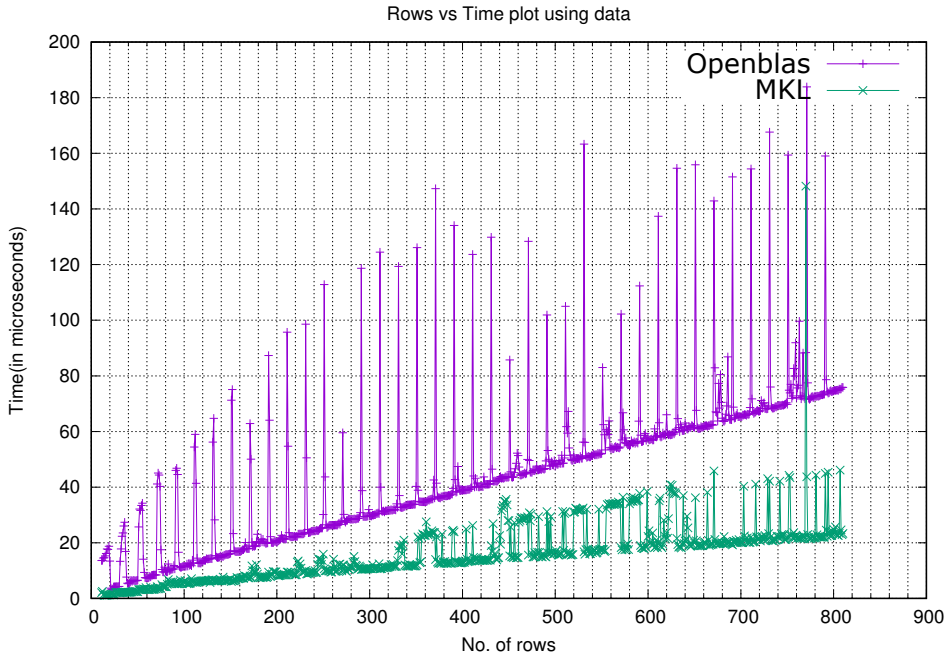

Figure 2: Comparing openBlas and MKL

3.3 Some observations of including the libraries

We found Pthreads and openBlas to be extremely user-friendly in terms of their installation. On the other hand, it was quite inconvenient to actually understand which package for the Intel MKL library would suffice. Once that was done, it was seen that manually setting up the environmental variables was in fact no good, and instead for the best experience, one must make changes to the .profile to restore the changes after shut down of the machine.

	Time (in microseconds)		
Matrix Size	Pthreads	openBlas	MKL
100 x 50	219.729	11.578	5.898
110 x 50	239.548	12.6369	6.037
120 x 50	252.257	13.35	6.313
150 x 50	254.686	16.199	7.034
200 x 50	324.321	20.116	8.728
300 x 50	469.859	29.596	10.786
400 x 50	582.349	38.894	12.939
500 x 50	688.748	47.577	16.65
600 x 50	779.342	57.452	22.320

Table 1: Time taken for different implementations

In addition, using the MKL Link Advisor requires the user to have a good understanding of the architecture, which served as a bottleneck for us for quite some time too. In all, we would say that though, MKL surely was the fastest of the 3 but it came with its own set of demons in its difficulty to install. There were also some fluctuations seen when using Openblas and MKL as you may see in Figure 2. Occasional spikes can be due to the fact that MKL and openBlas are restarted after every 20 iterations. It was mandatory to do so, else deadlocks took place hanging the whole system.

4 LeNet Architecture

In the third Sub-Task, we construct our version of the LeNet architecture by using 2 Convolution layers, 2 Max-Pooling Layer and 2 Fully Connected layers.

4.1 Details of the layers

The first step is to convert the given MNSIT image to an array of size 28x28. Then we store the array in the .txt file. The normalised form(values from 0-1) will be stored in the text file rather than the pixel values. Formula to for normalization is

$$NPV = 1 - \frac{1}{255} \times pixelValues \tag{1}$$

where NPV is the Normalized Pixel Values Matrix.

4.2 Preprocessing and Running the data

To preprocess the data, run

```
python3 preprocess.py <image_file_path>
```

This will output inputData.txt. Compile the code, and run it with

```
make lenet
./lenet.out inputData.txt
```

This will output the probabilities of each of the image being each of the digits from '0' to '9'. In addition, also the message for which is the most likely digit will output.

Layer	Input Dimensions	Input Channels	Output Channels	Kernel Size	Padding	Output Dimensions	Acti. Fun.
Conv 1	28 x 28	1	20	5 x 5	0	24 x 24 x 20	None
Pool 1	24 x 24	20	20	2 x 2 x 20	0	12 x 12 x 20	None
Conv 2	12 x 12	20	50	5 x 5	0	8 x 8 x 50	None
Pool 2	8 x 8	50	50	2 x 2 x 50	0	4 x 4 x 50	None
Fully Conn. 1	4 x 4	50	500	4 x 4	0	1 x 1 x 500	Relu
Fully Conn. 2	1 x 1	500	10	1 x 1	0	1 x 1 x 10	Softmax

Table 2: Layer Inputs and Outputs

5 Automating the Process

For the convenience of the user a bash script automate.sh has been provided. Executing it by

```
./automate.sh
```

A UI will start in the terminal. Follow its instructions to continue.

Table 3: Design structure with the functions

Executable	Function	Arguments							
./main.out	Sigmoid	vector file							
	Softmax	vector file							
	Tanh	matrix file	num rows						
	Relu	matrix file	num rows						
	Padding	matrix file	num rows	pad num					
	Pooling	matrix file	num rows	kernel size	pad num	pool type			
	Convolution	matrix file	num rows	kernel path	kernel rows	pad type	stride value	conv type	*[mat_mul type]*
./plot_[opt].out		rows of matrix	rows of kernel	iters					
./lenet.out		input file							

Legend:
vector/matrix/kernel file: string File path referencing to the txt file containing vector/matrix/kernel in column major order
num rows: integer specifying number of rows of the previous file
pad num: integer specifying the number of padding units
newline: string specifying the type of padding either 'same' or 'valid'

stride value: integer specifying the stride for the kernel to take
conv type: string specifying the type of convolution to use either "Convolution" or "Matrix".
mat_mul_type: optional string. Only specify if using "Matrix" as conv type. Can be either "mkl", "openblas" or "pthreads". Leaving it blank causes it to use pthreads.

References

[1] eddelbuettel. Mkl installation for debian. `https://github.com/eddelbuettel/mkl4deb/blob/master/script.sh`.

[2] Intel. Intel mkl link advisor. `https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor`.

[3] Intel. Setting environmental variables. `https://software.intel.com/en-us/mkl-linux-developer-guide-automating-the-process-of-setting-environment-variables`.