Efficient Matrix Multiplication

Garg, Poorva
poorva98@gmail.com

Senthilnathan, Aditya
aditsep99@gmail.com

February 2019

Objective of the Assignment

Taking convolution of two matrices can be done very efficiently if we employ matrix multiplication which itself is a very expensive thing to do in terms of execution time. So, beside the brute force algorithm, we tried 3 different techniques to make matrix multiplication efficient - used linear algebra libraries like mkl and openBlas, used multithreading in c++. Analyzing the outcomes of these new codes was the objective of this assignment

Commands for Different Matrix Multiplication Algorithms

To compile the code, run following commands on terminal

```
$ export LD_LIBRARY_PATH=/path_to_your_openblas_installation_directory/lib/
$ export MKLROOT=/path_to_your_mkl_installation_directory/intel/mkl/
$ make
```

To delete the compiled code

```
$ make clean
```

To use convolution employing different matrix multiplications, below are the instructions:

1. To use brute force algorithm
   ```
   $ ./output.out convolve_with_padding input_matrix.txt number_of_rows_of_input_matrix
   matrix2.txt number_of_rows_of_kernel
   $ ./output.out convolve_without_padding input_matrix.txt number_of_rows_of_input_-
   matrix matrix2.txt number_of_rows_of_kernel
   ```
2. To use MKL Library
   ```
   $ ./output.out convolve_with_padding_matrixmult_with_mkl input_matrix.txt
   number_of_rows_of_input_matrix matrix2.txt number_of_rows_of_kernel
   $ ./output.out convolve_without_padding_matrixmult_with_mkl input_matrix.txt
   number_of_rows_of_input_matrix matrix2.txt number_of_rows_of_kernel
   ```
3. To use OpenBlas Library
   ```
   $ ./output.out convolve_with_padding_matrixmult_with_openblas input_matrix.txt
   number_of_rows_of_input_matrix matrix2.txt number_of_rows_of_kernel
   $ ./output.out convolve_without_padding_matrixmult_with_openblas input_matrix.txt
   number_of_rows_of_input_matrix matrix2.txt number_of_rows_of_kernel
   ```
4. To use multithreading
   ```
   $ ./output.out convolve_with_padding_matrixmult_with_pthreads input_matrix.txt
   ```

```
number_of_rows_of_input_matrix matrix2.txt number_of_rows_of_kernel
$ ./output.out convolve_without_padding_matrixmult_with_pthreads input_matrix.txt
number_of_rows_of_input_matrix matrix2.txt number_of_rows_of_kernel
```

Linear Algebra Libraries

Let us first describe the two linear algebra libraries which perform the matrix multiplication with utmost efficiency

Intel® MKL Library

Intel® Math Kernel Library [1] is a computing math library of highly optimized, extensively threaded routines for applications that require maximum performance. The library provides Fortran and C programming language interfaces. Intel MKL C language interfaces can be called from applications written in either C or C++, as well as in any other language that can reference a C interface.

OpenBLAS Library

OpenBLAS [2] is an optimized BLAS library based on GotoBLAS2 1.13 BSD version. In scientific computing, OpenBLAS is an open source implementation of the BLAS (Basic Linear Algebra Subprograms) API with many hand-crafted optimizations for specific processor types. It is developed at the Lab of Parallel Software and Computational Science, ISCAS. OpenBLAS adds optimized implementations of linear algebra kernels for several processor architectures, including Intel Sandy Bridge and Loongson. It claims to achieve performance comparable to the Intel MKL.

Performance Comparison of MKL and OpenBLAS

We implemented matrix multiplication using linear algebra libraries MKL and OpenBLAS and recorded the execution time they take in two different situations

1 Multiple Execution on Same Size Matrices

We ran our code to multiply matrices of 50X50 500 times and collected the data of execution time taken to find mean and standard deviation and found that both the libraries give similar performance on small sized matrices.

Data of 500 times execution		
Data	MKL	OpenBLAS
Mean	0.00019351	0.00019351
Standard Deviation	5.9966E-05	5.6906E-05

2 Executing Once on Different Size Matrices

We ran our code to multiply matrices of various sizes ranging from size 1X1 to 300X300 and collected the data of execution time taken to find mean and standard deviation and found that both the libraries give similar performance. But since this time, large sized matrices were also there, MKL emerged out to perform slightly better.

Data of execution on different sized matrices		
Data	MKL	OpenBLAS
Mean	2.79E-03	3.98E-03
Standard Deviation	0.00235683	0.00354337

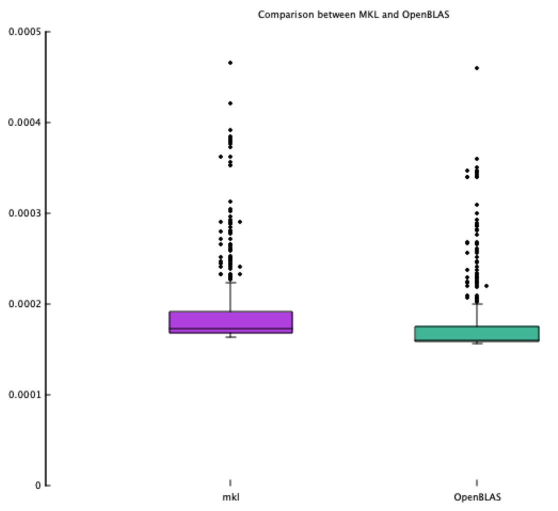

Figure 1: Comparison between MKL and OpenBLAS for same sized matrices

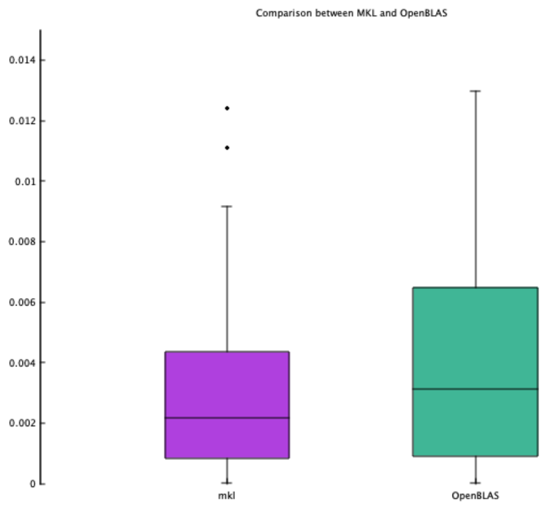

Figure 2: Comparison between MKL and OpenBLAS on Different sized matrices

Multithreading with Pthreads

MKL and OpenBLAS use threading routines to make matrix multiplication efficient. We tried multi-threading ourselves to improve the performance of our brute force algorithm. Since our machine had only two cores and only two threads could be run per core, we initiated four threads. We divided our matrix into four parts and multiplied each part separately to the latter matrix to get the final output. Following data of mean and standard deviation and graphs tell us about the performance of our pThreads implementation.

It is clearly visible that pThreads implementation was able to perform better than brute force implementation due to division of labour. Though, in case of small sized matrices, the overhead of creting a thread may sometimes, cause it to take more time in execution.

Data of execution on same sized matrices		
Data	Brute Force	pThreads
Mean	0.00364517	0.00252567
Standard Deviation	0.00085551	0.00064802

Data of execution on different sized matrices		
Data	Brute Force	pThreads
Mean	2.03E-01	1.84E-01
Standard Deviation	0.206982321	0.18303028

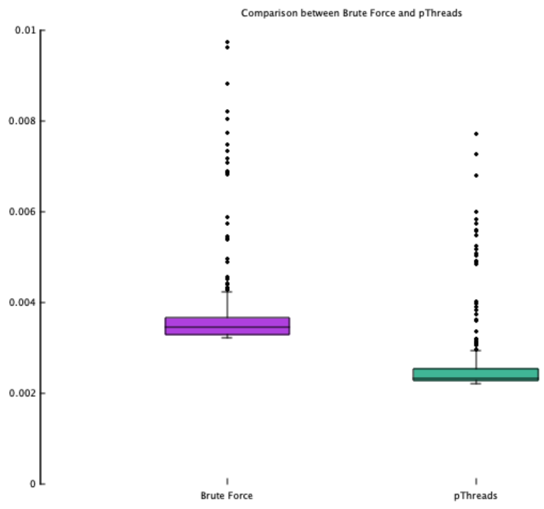

Figure 3: Comparison between Brute Force and pThreads on same sized matrices

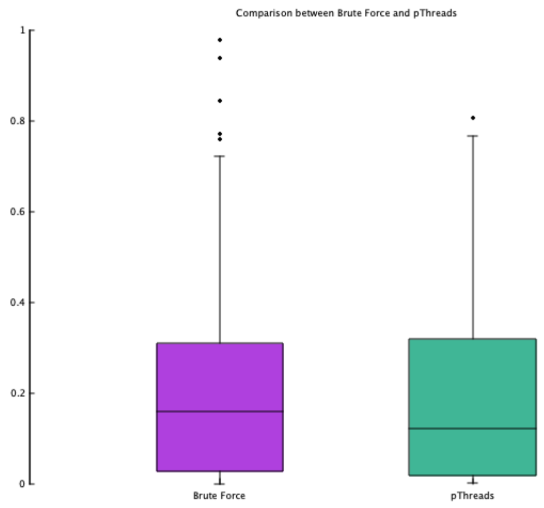

Figure 4: Comparison between Brute Force and pThreads on different sized matrices

Image Processing Library and LeNet Architecture

Along with carefully analyzing the various implementations for matrix multiplication, we also used them to implement our convolution function. We also coded some other image processing functions which include sigmoid, softmax, tanh, relu, maxpool and average pool and further used these functions to implement LeNet architecture to identify digits from MNIST images.

In the terminal, go to the directory which contains all the files, and then run following commands to run the code

```
$ python preprocess.py image.png
$ make
$ ./output.out
```

To delete the compiled code, run

```
$ make clean
```

References

[1] Intel® MKL Library. `https://software.intel.com/en-us/mkl/choose-download/linux`.

[2] OpenBLAS Library. `https://github.com/xianyi/OpenBLAS/wiki/Installation-Guide`.