Brandon Trahms

# Program Write Up

**Problem Description:**

**Program Description** - This program is based on a sequential starter program which builds and trains a Convolutional Neural Network or CNN on the MNIST handwritten digits dataset. Taking this starter code, my aim was to achieve speed up for the training loop of this CNN by making parallel various SIMD operations.

**Value Proposition** - This kind of problem is valuable for the growing size of Neural Networks and their training time. A parallel solution can help speed up the time of training and can increase the throughput for Industry use and machine learning research.

**Numerical Methods** - This program uses Vector/Matrix operations and Transformations when applying 2-D Convolution, Bias offsetting, Activation function to the layers and applying derivative adjustments to the Weighted edges.

**Parallel Methods** - This program uses OpenMP to achieve parallelism which is a compiler directive driven method which generates parallel shared memory code for specific portions of the program.

**Sequential Program:**

**Build and Run:**

```
bctrahms@o244-29:~/CSCI551/MNIST1$ make
g++  -fopenmp   -o Source Source.cpp
Source.cpp: In destructor 'CMNISTData::~CMNISTData()':
Source.cpp:170:12: warning: deleting 'void*' is undefined [-Wdelete-incomplete]
  170 |   delete[] m_labelData;
      |            ^~~~~~~~~~
Source.cpp:171:12: warning: deleting 'void*' is undefined [-Wdelete-incomplete]
  171 |   delete[] m_imageData;
      |            ^~~~~~~~~~
bctrahms@o244-29:~/CSCI551/MNIST1$ ./Source
Training Data Accuracy: 17.88%
Test Data Accuracy: 18.16%

Training epoch 1 / 1...

Training Time:  70.54 seconds

Final Training Data Accuracy: 90.93%
Final Test Data Accuracy: 90.90%

sh: 1: pause: not found
bctrahms@o244-29:~/CSCI551/MNIST1$
```

**Run Times:**

| Run # | Seconds |
|---|---|
| 1 | 70.56 |
| 2 | 70.52 |
| 3 | 70.52 |
| 4 | 70.54 |
| 5 | 70.53 |
| Mean | 70.534 |

These are the times to completion of a single training loop or 1 Epoche. Timing was done using the chrono library's high resolution clock and ran on the ECC cluster Node o244-29.

## Parallel Program:

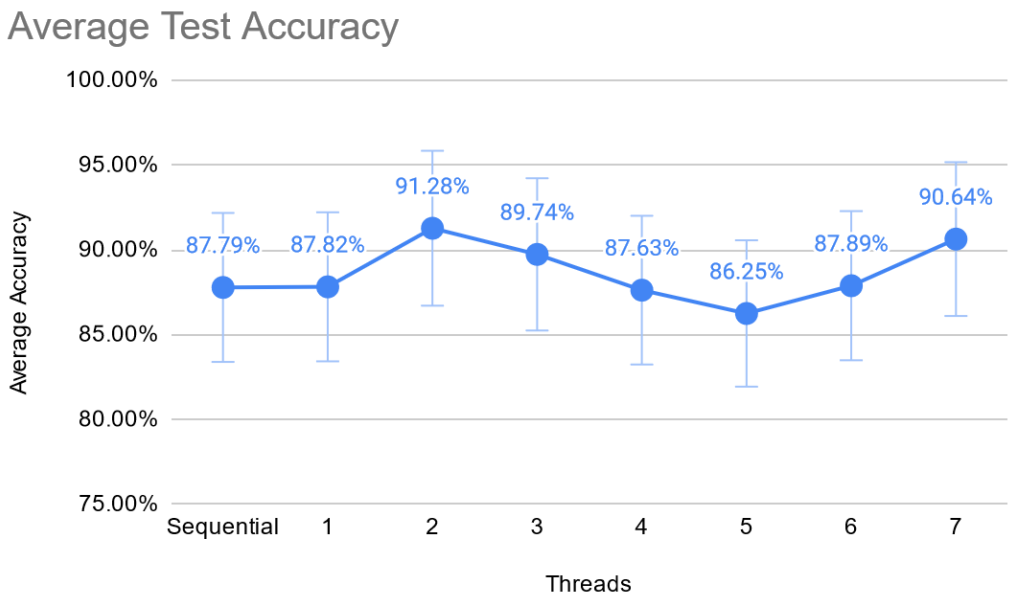
**Build and Run:**

```
bctrahms@o244-29:~/CSCI551/MNIST1-OMP$ make
g++  -fopenmp   -o Source Source.cpp
Source.cpp: In destructor 'CMNISTData::~CMNISTData()':
Source.cpp:173:12: warning: deleting 'void*' is undefined [-Wdelete-incomplete]
  173 |    delete[] m_labelData;
      |             ^~~~~~~~~~~
Source.cpp:174:12: warning: deleting 'void*' is undefined [-Wdelete-incomplete]
  174 |    delete[] m_imageData;
      |             ^~~~~~~~~~~
bctrahms@o244-29:~/CSCI551/MNIST1-OMP$ ./Source
Threads: 4
Training Data Accuracy: 10.18%
Test Data Accuracy: 9.76%

Training epoch 1 / 1...

Training Time:  37.92 seconds

Final Training Data Accuracy: 91.63%
Final Test Data Accuracy: 91.65%

sh: 1: pause: not found
bctrahms@o244-29:~/CSCI551/MNIST1-OMP$
```

**Run Times:**

| Run # \Thread # | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 69.46 | 38.56 | 43.34 | 36.75 | 55.49 | 56.03 | 58.26 |
| 2 | 69.47 | 38.65 | 43.33 | 37.53 | 55.54 | 55.95 | 58.43 |
| 3 | 69.48 | 38.56 | 43.58 | 37.65 | 55.45 | 56.18 | 58.32 |
| 4 | 69.48 | 38.66 | 43.32 | 37.56 | 55.5 | 56.4 | 58.23 |
| 5 | 69.47 | 38.58 | 43.59 | 36.77 | 55.5 | 55.91 | 58.31 |
| Mean | 69.472 | 38.602 | 43.432 | 37.252 | 55.496 | 56.094 | 58.31 |

**Accuracy Check:**




As a check of correctness, plotted the average ending Test Accuracy at each thread number. Based on the close clustering of the accuracies, the parallel version of the program is still working correctly.

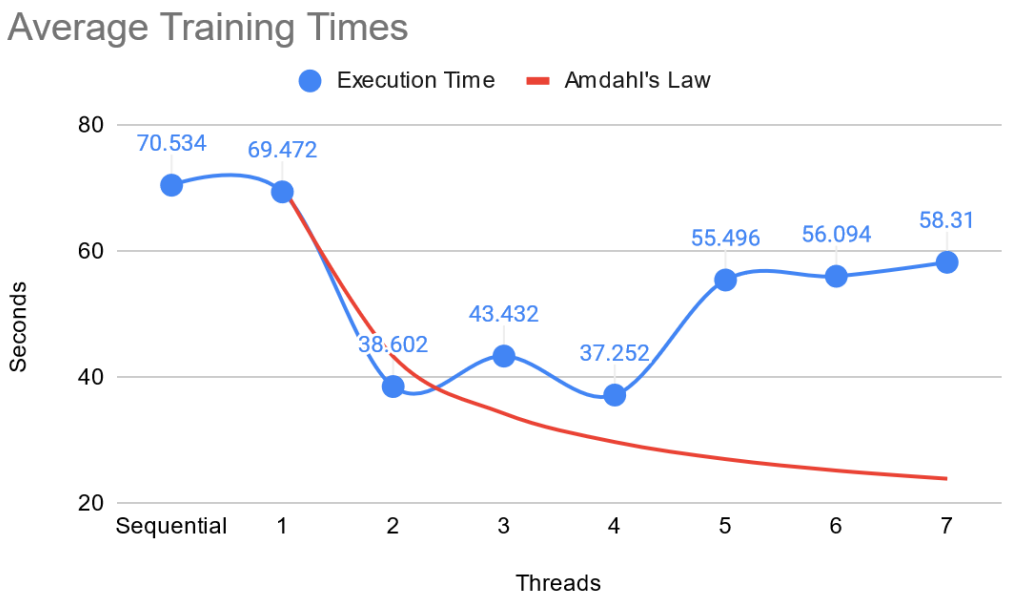
**Speed Up Analysis:**

**Amdhal's Law:**

| Amdhal's Law Parameter | How I derived it | Description |
|---|---|---|
| Parallel Portion(P) | I timed the overall training loop then timed the specific portions of the code that were going to be made parallel. I then divided the sum of the future parallel sections by the overall which gave me the proportion of the program I occupied. | P varied slightly from run to run given it was based on the clock time, but it averaged to be roughly .77. |
| Sequential Portion(1 - P) | Given that I know P, I can calculate the Sequential Portion by subtracting P from 1. | P = .77<br>1 - P = .23 |
| Number of Shared Memory Cores | Using the command htop on a node of the ECC cluster, I | I observed there to be 4 cores. |

| | | |
|---|---|---|
| | could view the number of cores and their usage. | |
| Scaling factor(S) | The optimal scaling factor is based on the number of cores that can be saturated before reaching starvation. Since we have 4 cores, I concluded the optimal threads count would be 4. Thus making our scaling factor to be 4. | S = 4 |

**Graph:**




As the number of threads or S increases, we can see the ideal speed up calculated from Amdahl's Law shown in red lowers the execution time. As seen in blue the time execution time does stay relatively competitive with the ideal speed up, even exceeding it at points. But after the program exceeds 4 threads it experiences a significant slow down which can most likely be attributed to the over saturation of the cores and increased time to switch between threads.