CSE 625 Parallel Programming
Project 3
By Caleb Klenda

Machine Specifications

The project was all performed on my home computer with the following specifications:

- **CPU**
  Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
  AVX2 (256-bit MM registers)
  10 cores / 20 threads
  20 MB Intel Smart Cache (L3-cache)

- **RAM**
  32 GB DDR4 RAM

- **GPU**
  TUF RTX3080 (Ampere GPU)
  8704 CUDA cores
  5 MB of L2-Cache
  10GB GDDR6X

Problem 1

```python
import numpy as np
import matplotlib.pyplot as plt
```
[1] ✓ 4.4s Python

```python
mis_match = [(115, 8111), (195, 47020), (241, 9732), (268, 47938), (300, 49308), (320, 33406), (321, 10485), (341, 34266), (358, 19138), (381,
```
[2] ✓ 0.6s Python

```python
len(mis_match)
```
[3] ✓ 0.4s Python

```
309
```

Read in Data

\+ Code \+ Markdown

```python
# Read train images (60,000x28x28 float32)
train_images = np.fromfile('./data/train-images.bin', dtype = np.float32)
train_images = train_images.reshape(60000, 28, 28)

# Read train images (60,000 ubyte)
train_labels = np.fromfile('./data/train-labels.bin', dtype = np.ubyte)

# Read test images (10,000x28x28 float32)
test_images = np.fromfile('./data/test-images.bin', dtype = np.float32)
test_images = test_images.reshape(10000, 28, 28)

# Read test images (10,000 ubyte)
test_labels = np.fromfile('./data/test-labels.bin', dtype = np.ubyte)
```
[4] ✓ 0.5s Python

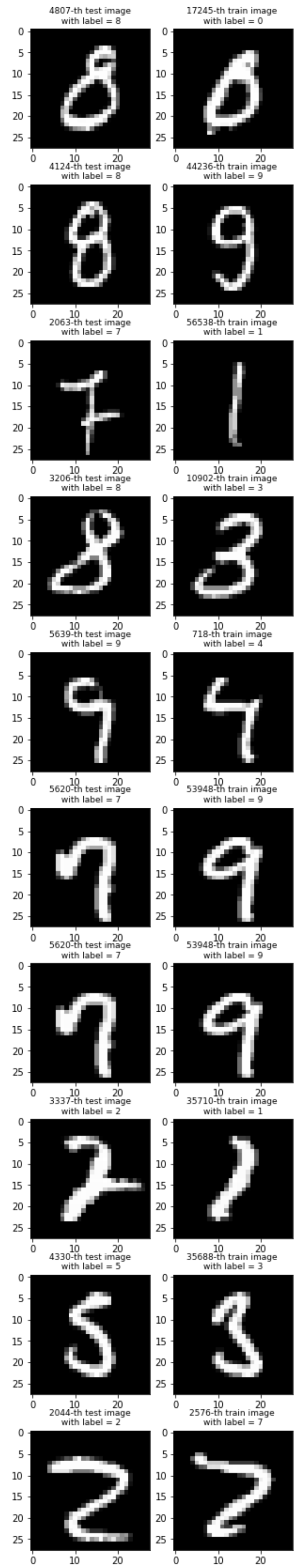
Plot

```python
#utility for plotting
def pairPlot(indexes, i):
    test_idx, train_idx = indexes
    plt.subplot(10, 2, (2*i)+1)
    plt.imshow(test_images[test_idx], cmap='gray')
    plt.title(str(test_idx) + "-th test image\n with label = " + str(test_labels[test_idx]), fontsize = 9)
    plt.subplot(10, 2, (2*i)+2)
    plt.imshow(train_images[train_idx], cmap='gray')
    plt.title(str(train_idx) + "-th train image \nwith label = " + str(train_labels[train_idx]), fontsize = 9)

def pickandPlotRandomPairs(numberOfPairs):
    plt.figure(figsize=(5,30)) # width, height in inch of the plot
    for i in range(0, numberOfPairs):
        randomPair = mis_match[np.random.choice([k for k in range(0,len(mis_match))])]
        pairPlot(randomPair, i)

pickandPlotRandomPairs(10)
```

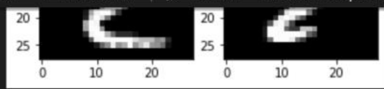
[8] ✓ 1.9s Python


4807-th test image
with label = 8
17245-th train image
with label = 0
4124-th test image
with label = 8
44236-th train image
with label = 9
2063-th test image
with label = 7
56538-th train image
with label = 1
3206-th test image
with label = 8
10902-th train image
with label = 3
5639-th test image
with label = 9
718-th train image
with label = 4
5620-th test image
with label = 7
53948-th train image
with label = 9
5620-th test image
with label = 7
53948-th train image
with label = 9
3337-th test image
with label = 2
35710-th train image
with label = 1
4330-th test image
with label = 5
35688-th train image
with label = 3
2044-th test image
with label = 2
2576-th train image
with label = 7

Mis_match.ipynb   CSE625H3-20.   200   Dataset.ipynb

mismatch_klenda.ipynb > M↓Plot > #usi   Enter an index between 0-308: (Press 'Enter' to confirm or 'Escape' to cancel)   choice = int(i

+ Code   + Markdown   Run All   Clear Outputs of All Cells   Go To Running Cell   Restart   Interrupt   Variables   Outline   ···




```python
#using utility from before
def pairAndPlotUser():
    choice = -1
    while(choice < 0 or choice > 308):
        choice = int(input('Enter an index between 0-308: '))
        if choice > 0 and choice < 308:
            print("You chose: ", mis_match[choice])
            pairPlot(mis_match[choice], 0)

plt.figure(figsize=(4,14))
pairAndPlotUser()
```

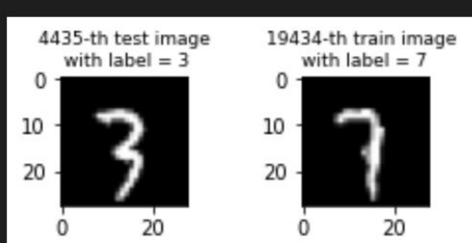
[9]   25.2s

```python
#using utility from before
def pairAndPlotUser():
    choice = -1
    while(choice < 0 or choice > 308):
        choice = int(input('Enter an index between 0-308: '))
        if choice > 0 and choice < 308:
            print("You chose: ", mis_match[choice])
            pairPlot(mis_match[choice], 0)

plt.figure(figsize=(4,14))
pairAndPlotUser()
```

[9]   ✓ 31.9s

```
You chose:  (4435, 19434)
```



Problem 2

In the CodeBlocks project, `All_Pair_distance`, it implements four functions to compute the pair-wise distance matrix of MNIST train images (loaded from `train-images.bin`). These four methods are:

1- `sequential_all_pairs` (sequential computing)
2- `block_all_pairs` (C++ multi threads - block work distribution)
3- `block_ cyclic_all_pairs` (C++ multi threads - block cyclic work distribution)
4- `dynamic_all_pairs` (C++ multi threads - dynamic work distribution)

2 .1

Matrix Size	400	800	10,000	20,000	30,000	60,000
Method 1	0.0503091	0.200686	32.9804	136.509	322.867	1560.27
Method 2 12 threads	0.0086379	0.0312908	6.41424	34.1735	97.6546	425.77
Method 3 12 threads Chunk size 2	0.0052788	0.0184834	2.78006	11.524	28.0694	112.191
Method 4 12 threads Chunk size 2	0.0050414	0.0183471	2.77072	11.5129	27.8863	111.889

2.2 (8 points) In the report, explain the key ideas of the function, `dynamic_all_pairs`, of its work distribution implementation and how and why the `std::mutex` object is used.

`dynamic_all_pairs` assign chunks to threads at runtime allowing it to adapt the the problem it is solving. global_lower which allows access to the first row of the currently processed chunk, which, whenever a thread runs out of work, it will reference to determine what it should do next. Because multiple threads are accessing and modifying global_lower, a mutex is needed so that only one thread can use that resource at a time

Problem 3

Work amount (i.e., the number of outmost iterations) done by each thread for 2 threads on `block_all_pairs` for m=60,000 mxm matrix

We know that for `block_all_pairs` T(i) = i + 1. When using three threads, we split the work into the following:

$$W(1) = \sum_{i=0}^{\frac{m}{3}-1} \mathrm{T(i)} = \sum_{i=0}^{\frac{m}{3}-1} (\mathrm{i}+1) = \sum_{i=0}^{\frac{m}{3}-1} (\mathrm{i}) + \frac{\mathrm{m}}{3} = \frac{\left(\frac{\mathrm{m}}{3}-1\right)\left(\frac{m}{3}\right)}{2} + \frac{m}{3} = \frac{m^2}{18} - \frac{m}{6} + \frac{m}{3}$$

$$= \frac{m^2}{18} + \frac{m}{6}$$

$$W(2) = \sum_{i=\frac{m}{3}}^{\frac{2m}{3}-1} \mathrm{T(i)} = \sum_{i=\frac{m}{3}}^{\frac{2m}{3}-1} (\mathrm{i}+1) = \sum_{i=\frac{m}{3}}^{\frac{2m}{3}-1} (\mathrm{i}) + \frac{\mathrm{m}}{3} = \sum_{i=0}^{\frac{m}{3}-1} \left(\frac{\mathrm{m}}{3} + \mathrm{i}\right) + \frac{\mathrm{m}}{3} =$$

$$= \sum_{i=0}^{\frac{m}{3}-1} \left(\frac{\mathrm{m}}{3}\right) + \sum_{i=0}^{\frac{m}{3}-1} (\mathrm{i}) + \frac{\mathrm{m}}{3} = \frac{m}{3}\left(\frac{m}{3}\right) + \frac{\left(\frac{m}{3}-1\right)\left(\frac{m}{3}\right)}{2} + \frac{m}{3}$$

$$= \frac{m^2}{9} + \frac{\frac{m^2}{9} - \left(\frac{m}{3}\right)}{2} + \frac{m}{3} = \frac{m^2}{9} + \frac{m^2}{18} - \frac{m}{6} + \frac{m}{3} = \frac{m^2}{6} + \frac{m}{6}$$

$$W(3) = \sum_{i=\frac{2m}{3}}^{m-1} \mathrm{T(i)} = \sum_{i=2\frac{m}{3}}^{m-1} (\mathrm{i}+1) = \sum_{i=\frac{2m}{3}}^{m-1} (\mathrm{i}) + \frac{\mathrm{m}}{3} = \sum_{i=0}^{\frac{m}{3}-1} \left(\frac{2\mathrm{m}}{3} + \mathrm{i}\right) + \frac{\mathrm{m}}{3} =$$

$$= \sum_{i=0}^{\frac{m}{3}-1}\left(\frac{2\mathrm{m}}{3}\right) + \sum_{i=0}^{\frac{m}{3}-1}(\mathrm{i}) + \frac{\mathrm{m}}{3} = \frac{m}{3}\left(\frac{2m}{3}\right) + \frac{\left(\frac{m}{3}-1\right)\left(\frac{m}{3}\right)}{2} + \frac{m}{3}$$

$$= \frac{2m^2}{9} + \frac{\frac{m^2}{9} - \left(\frac{m}{3}\right)}{2} + \frac{m}{3} = \frac{2m^2}{9} + \frac{m^2}{18} - \frac{m}{6} + \frac{m}{3} = \frac{m^2}{2} + \frac{m}{6}$$

$$= \frac{5m^2}{18} + \frac{m}{6}$$

So, in the general case the threads work as follows:

$W(1) = \frac{m^2}{18} + \frac{m}{6}, \mathrm{W}(2) = \frac{m^2}{6} + \frac{m}{6}, W(3) = \frac{5m^2}{18} + \frac{m}{6}$

- We plug in 60,000 for m to achieve $\mathrm{W}(1)_{60000}$= 2,000,010,000
- We plug in 60,000 for m to achieve $\mathrm{W}(2)_{60000}$= 600,010,000
- We plug in 60,000 for m to achieve $\mathrm{W}(3)_{60000}$= 1,000,010,000

From this we can observe that as $m \rightarrow \infty$ that W(1) accounts for 11% of the work, W(2) accounts for 33% of the work, and the W(3) accounts for the remaining 55% of the work.

Problem 4

Work amount (i.e., the number of outmost iterations) done by each thread for 2 threads on `block_cyclic_all_pairs`,

Using the formula given $F(i0, c) = a\left(\frac{c^2+2ic+c}{2}\right)$ we compute the following for each thread:

- $\mathrm{W}(1) = 1\left(\frac{\left(\frac{m}{3}\right)^2 + \frac{2(0)m}{3} + \frac{m}{3}}{2}\right) = \frac{m^2}{18} + \frac{m}{6} = \frac{\frac{1}{3}m^2 + m}{6}$
- $\mathrm{W}(2) = 1\left(\frac{\left(\frac{m}{3}\right)^2 + \frac{2\left(\frac{m}{3}\right)m}{3} + \frac{m}{3}}{2}\right) = \frac{m^2}{18} + \frac{2m^2}{18} + \frac{m}{6} = \; = \frac{m^2+m}{6}$

- $W(3) = 1\left(\frac{\left(\frac{m}{3}\right)^2 + \frac{2\left(\frac{2m}{3}\right)m}{3} + \frac{m}{3}}{2}\right) = \frac{m^2}{18} + \frac{4m^2}{18} + \frac{m}{6} = \; = \frac{5m^2}{18} + \frac{m}{6}$

We can see that these exactly match the results from Problem 3 so:

- We plug in 60,000 for m to achieve $W(1)_{60000}$= 2,000,010,000
- We plug in 60,000 for m to achieve $W(2)_{60000}$= 600,010,000
- We plug in 60,000 for m to achieve $W(3)_{60000}$= 1,000,010,000