

## Marshall Lanning

### CECS 625 Parallel Programming Homework Assignment #4 October 21, 2019 (100 points)

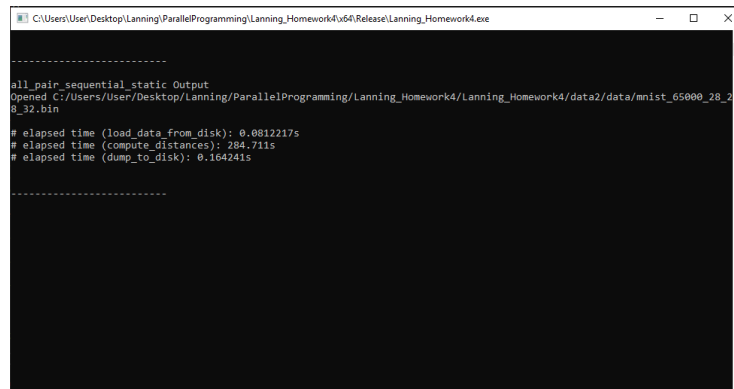
Due: November 6 (Wed) midnight

(Submit your project report and any required VS 2015 project to the Blackboard.)

#### Assignment Description OpenMP-related problems

- 1 (20 points) Consider the source file, `all_pair.cpp`, in Section 4.4. Parallelize the sequential version, `sequential_all_pairs`, using OpenMP with three different scheduling policies, `static`, `dynamic`, and `s` (see page 190 of the textbook). Compare timing performance of these three OpenMP scheduling implementations and the C++ multi-threading version, `dynamic_all_pairs`. Show and discuss the timing results in the project report.

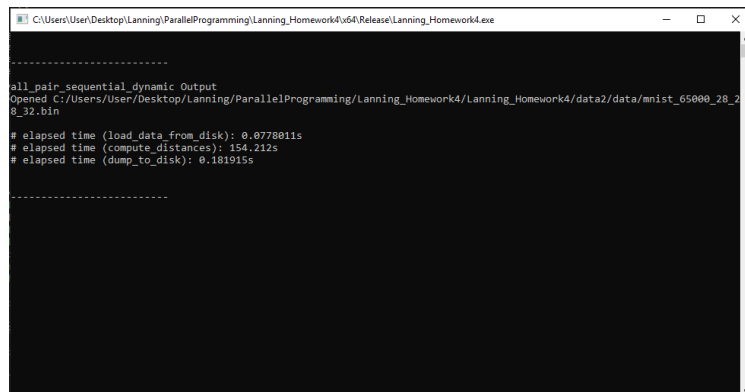
#### Static Scheduling



```
C:\Users\User\Desktop\Lanning\ParallelProgramming\Lanning_Homework4\Release\Lanning_Homework4.exe

-----
all_pair_sequential_static Output
Opened C:/Users/User/Desktop/Lanning/ParallelProgramming/Lanning_Homework4/Lanning_Homework4/data2/data/mnist_65000_28_2_8_32.bin
# elapsed time (load_data_from_disk): 0.0812217s
# elapsed time (compute_distances): 284.711s
# elapsed time (dump_to_disk): 0.164241s
-----
```

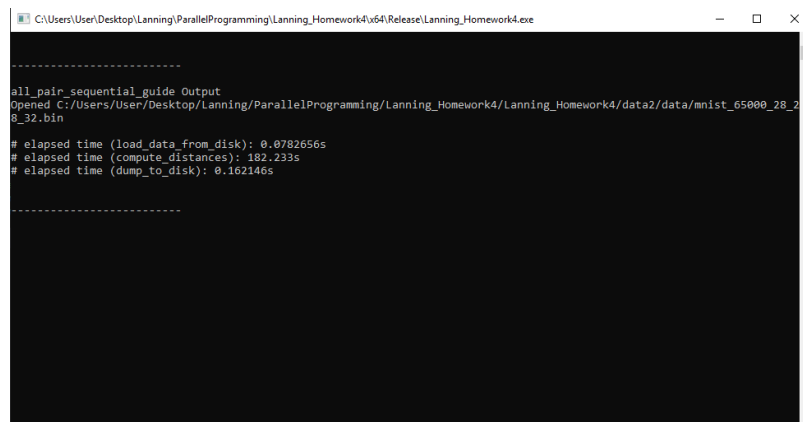
#### Dynamic Scheduling



```
C:\Users\User\Desktop\Lanning\ParallelProgramming\Lanning_Homework4\Release\Lanning_Homework4.exe

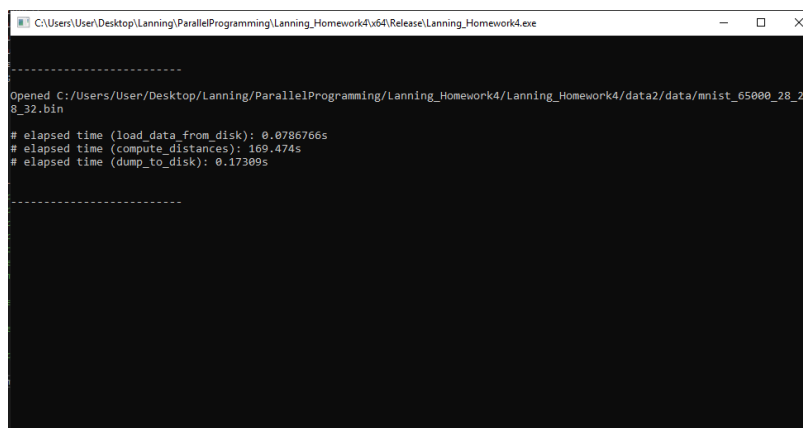
-----
all_pair_sequential_dynamic Output
Opened C:/Users/User/Desktop/Lanning/ParallelProgramming/Lanning_Homework4/Lanning_Homework4/data2/data/mnist_65000_28_2_8_32.bin
# elapsed time (load_data_from_disk): 0.0778011s
# elapsed time (compute_distances): 154.212s
# elapsed time (dump_to_disk): 0.181915s
-----
```

## Guided Scheduling



```
-----  
all_pair_sequential_guide Output  
Opened C:/Users/User/Desktop/Lanning/ParallelProgramming/Lanning_Homework4/Lanning_Homework4/data2/data/mnist_65000_28_2_8_32.bin  
# elapsed time (load_data_from_disk): 0.0782656s  
# elapsed time (compute_distances): 182.223s  
# elapsed time (dump_to_disk): 0.162146s  
-----
```

## Multi-Threaded Dynamic



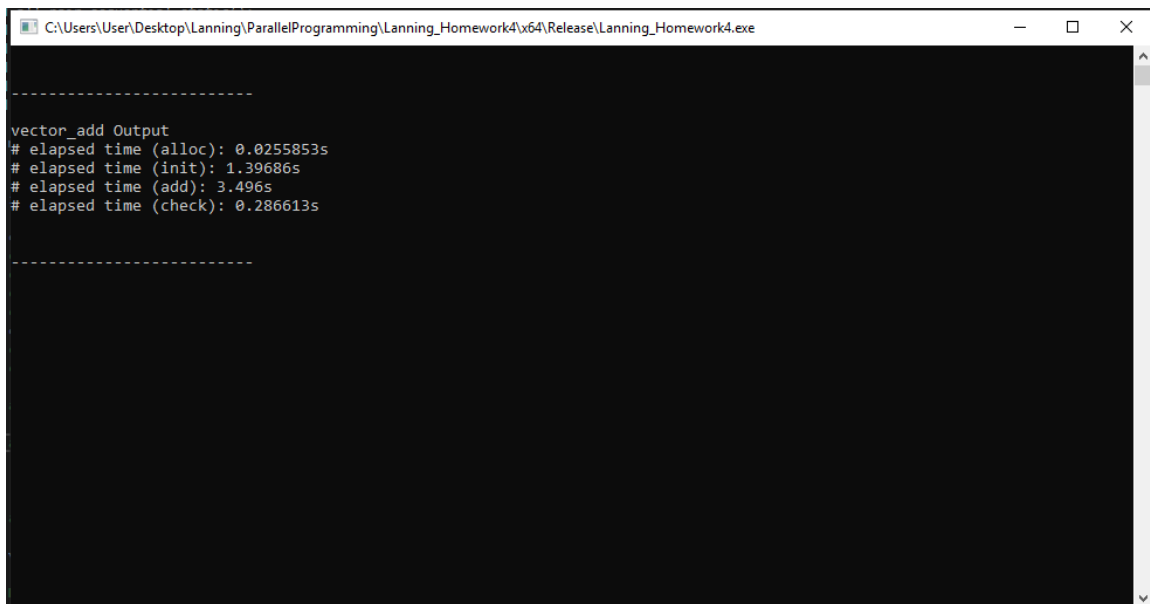
```
-----  
Opened C:/Users/User/Desktop/Lanning/ParallelProgramming/Lanning_Homework4/Lanning_Homework4/data2/data/mnist_65000_28_2_8_32.bin  
# elapsed time (load_data_from_disk): 0.0786766s  
# elapsed time (compute_distances): 169.474s  
# elapsed time (dump_to_disk): 0.17389s  
-----
```

As you can see the best performer here was sequential\_all\_pair with OpenMP scheduling set to dynamic. The next best was the Multi-Threaded Dynamic implementation then guided then static. Static does the worst here because it divides the work for all\_pair up in many chunks and causes idling at the end when there are threads still doing work. Dynamic scheduling does best here because there is basically no idle time. With dynamic scheduling, work is distributed evenly and work is given to the threads as soon as they are ready for more work. Guided does a decent job here and divides the work load up into chunks where each successive chunk decreases in size. You can really visualize the workload when viewing the CPU usage in the Diagnostic Tools when running the program.

Similar to the Project 3, create an OpenMP C++ project (called chapter6) in VS 2015 and copy the indclud folder (as in Project 3) and the data folder (downloaded from the

Blackboard; note this data folder is not the same as the data folder used in Project 3) to the new project home folder. Use this new project to do problems 2-4.

- 2 (10 points) Add the source file, `vector_add.cpp` (listed and explained in Section 6.2), from the textbook's source code website. Modify `vector_add.cpp` so that it will run in VC++ and OpenMP 2.0. Explain your modifications and discuss the timing results in your project report.



```
-----  
vector_add Output  
# elapsed time (alloc): 0.0255853s  
# elapsed time (init): 1.39686s  
# elapsed time (add): 3.496s  
# elapsed time (check): 0.286613s  
-----
```

The only modifications needed to be made for this are to replace any `uint64_t` declarations with `int64_t`. When this is ran without OpenMP parallelization the time to add is 10.47 seconds, when it is ran with OpenMP parallelization the time to add is 3.49 seconds. This proved the effectiveness of OpenMP's parallelization features of for loops.

- 3 (25 points) Add the source file, `1NN.cpp` (listed and explained in Section 6.3), from the textbook's source code website. Modify `1NN.cpp` so that it will run in VC++ and OpenMP 2.0.

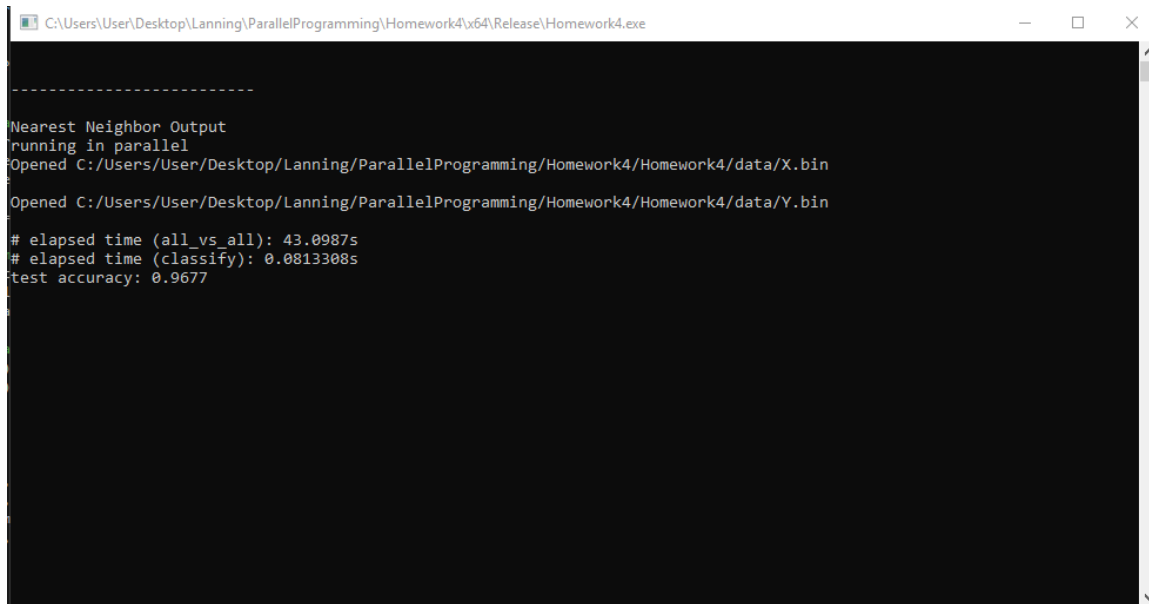
(a) Explain your modifications in your project report.

The modifications I made for 1NN were to first replace any `uint64_t` declarations with `int64_t`. I then removed any `collapse()` calls made when using OpenMP because it is not supported in OpenMP 2.0. I then replaced `main` with my own function declaration to be called in my own `main` function. I then replaced the file path calling the X and Y binary files to match exactly where I put them on the PC.

(b) Show and explain the outputs from `1NN.cpp`.

The output shows the timing result of an `all_vs_all` method implementation with OpenMP parallelization applied. As you can see it ran in under a minute. The

all\_vs\_all method is used to calculate the distance between a given input X and 65,000 images from the mnist data set. Parallelization is very effective here because of the need to iterate through all 65,000 images. The accuracy output shown below is to display the result of how accurate the Nearest Neighbor algorithm guess what a number was according to how big the distance was from the guessed number in terms of probability.



```
-----
Nearest Neighbor Output
running in parallel
Opened C:/Users/User/Desktop/Lanning/ParallelProgramming/Homework4/Homework4/data/X.bin
Opened C:/Users/User/Desktop/Lanning/ParallelProgramming/Homework4/Homework4/data/Y.bin
# elapsed time (all_vs_all): 43.0987s
# elapsed time (classify): 0.0813308s
test accuracy: 0.9677
```

- 4 (45 points) Add the source file, `softmax.cpp` (listed and explained in Section 6.5), from the textbook's source code website. Modify `softmax.cpp` so that it will run in VC++ and OpenMp 2.0.
- (a) Explain your modifications in your report.

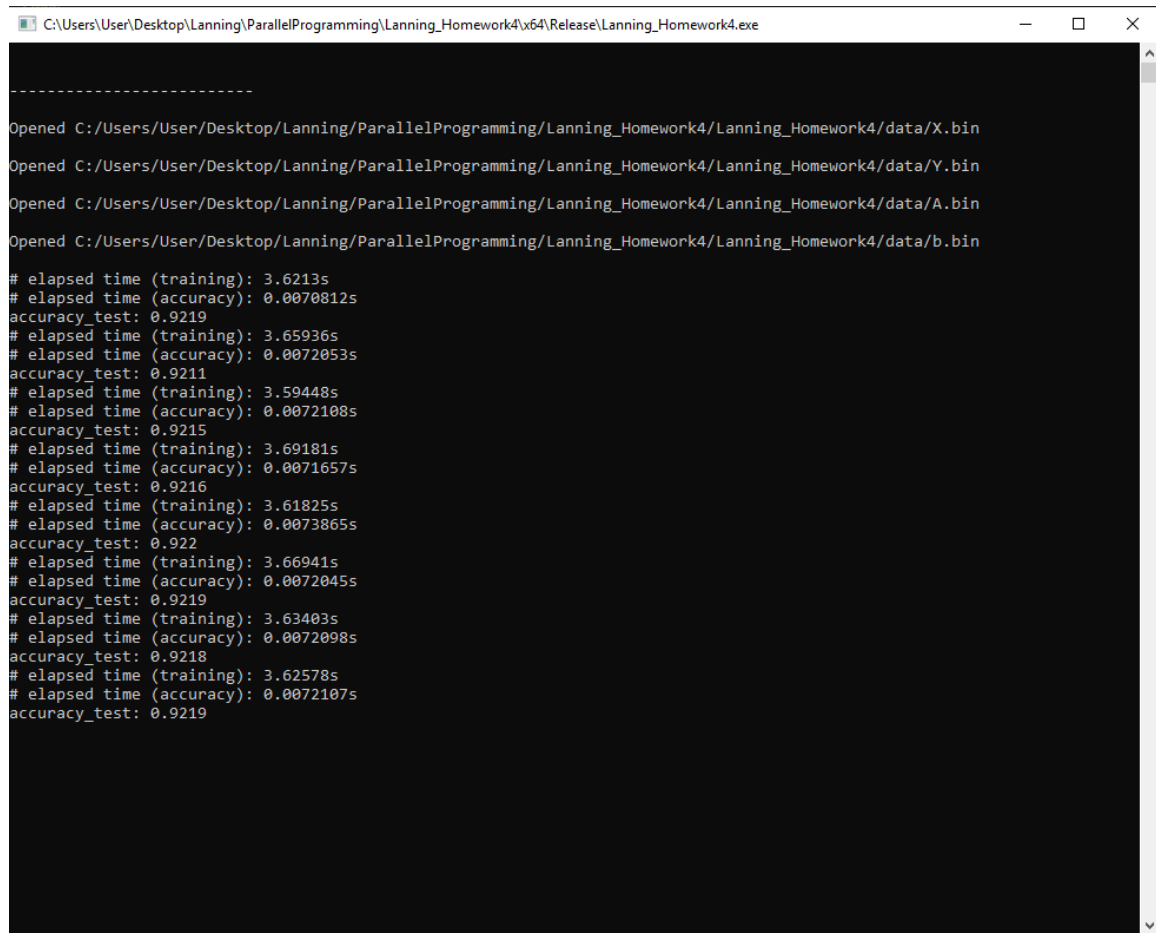
To get this working I first replaced any `uint_64` declarations with `int_64`. I then renamed `main` to match my own function calls in my own `main`. I then needed to include the algorithm library for the `std::max` functions that are called. In the accuracy function I needed to replace `value_t output[num_classes];` with `value_t output[10];` because of the error it threw about the array size being declared as not a constant. I chose 10 because the number of classes will always be 10 in the case of the dataset being numbers from 0 – 9. I then needed to make the function calls to `softmax_regression` and `argmax` in accuracy to have this `<float, int64_t>` after the function name. This was because of the way the functions were created with templates. The `value_t` and `index_t` items were to be floats and `int64_t` values when making those calls. I then needed to change the filepath to the X, Y, A, and b files.

(b) Show and explain the outputs from softmax.cpp.

```
C:\Users\User\Desktop\Lanning\ParallelProgramming\Lanning_Homework4\x64\Release\Lanning_Homework4.exe
-----
Opened C:/Users/User/Desktop/Lanning/ParallelProgramming/Lanning_Homework4/Lanning_Homework4/data/X.bin
Opened C:/Users/User/Desktop/Lanning/ParallelProgramming/Lanning_Homework4/Lanning_Homework4/data/Y.bin
# elapsed time (training): 3.69493s
# elapsed time (accuracy): 0.0072505s
accuracy_test: 0.8359
# elapsed time (training): 3.60478s
# elapsed time (accuracy): 0.0111253s
accuracy_test: 0.8591
# elapsed time (training): 3.59382s
# elapsed time (accuracy): 0.0072483s
accuracy_test: 0.8693
# elapsed time (training): 3.6063s
# elapsed time (accuracy): 0.0075306s
accuracy_test: 0.8774
# elapsed time (training): 3.57169s
# elapsed time (accuracy): 0.0072037s
accuracy_test: 0.8814
# elapsed time (training): 3.63702s
# elapsed time (accuracy): 0.0146217s
accuracy_test: 0.8851
# elapsed time (training): 3.58301s
# elapsed time (accuracy): 0.0081276s
accuracy_test: 0.8875
# elapsed time (training): 3.58023s
# elapsed time (accuracy): 0.0072229s
accuracy_test: 0.8899
# elapsed time (training): 3.57956s
# elapsed time (accuracy): 0.0072923s
accuracy_test: 0.8919
# elapsed time (training): 3.59805s
# elapsed time (accuracy): 0.0084229s
accuracy_test: 0.8941
# elapsed time (training): 3.5795s
# elapsed time (accuracy): 0.0150191s
accuracy_test: 0.8962
# elapsed time (training): 3.60831s
# elapsed time (accuracy): 0.0072188s
accuracy_test: 0.8981
# elapsed time (training): 3.59208s
# elapsed time (accuracy): 0.0081488s
accuracy_test: 0.8985
# elapsed time (training): 3.59135s
# elapsed time (accuracy): 0.0073955s
accuracy_test: 0.8996
# elapsed time (training): 3.58198s
# elapsed time (accuracy): 0.0072344s
accuracy_test: 0.9
# elapsed time (training): 3.62876s
# elapsed time (accuracy): 0.0075824s
accuracy_test: 0.9011
# elapsed time (training): 3.65s
# elapsed time (accuracy): 0.007838s
accuracy_test: 0.9022
# elapsed time (training): 3.61135s
# elapsed time (accuracy): 0.0084586s
accuracy_test: 0.9035
# elapsed time (training): 3.64276s
# elapsed time (accuracy): 0.0072214s
accuracy_test: 0.9044
# elapsed time (training): 3.6512s
# elapsed time (accuracy): 0.0096423s
accuracy_test: 0.9052
# elapsed time (training): 3.67544s
# elapsed time (accuracy): 0.0079714s
accuracy_test: 0.9058
# elapsed time (training): 3.64808s
# elapsed time (accuracy): 0.0072149s
accuracy_test: 0.9069
# elapsed time (training): 3.6452s
# elapsed time (accuracy): 0.0122555s
accuracy_test: 0.907
# elapsed time (training): 3.60033s
# elapsed time (accuracy): 0.007994s
accuracy_test: 0.9073
# elapsed time (training): 3.68033s
# elapsed time (accuracy): 0.0091452s
accuracy_test: 0.9076
# elapsed time (training): 3.57733s
# elapsed time (accuracy): 0.0072236s
accuracy_test: 0.908
```

What this function is doing is making 32 iterations at a time through the mnist data set to train the machine to know the handwritten numbers. The more it trains the more accurate it becomes, hence the increase in accuracy each time.

- (c) Use the pre-trained softmax (as given in data/A.bin and data/b.bin) and get its prediction accuracy. (Note A and b are the required parameters to run softmax regression.) Compare and discuss the accuracy with those obtained in (b).



```
-----  
Opened C:/Users/User/Desktop/Lanning/ParallelProgramming/Lanning_Homework4/Lanning_Homework4/data/X.bin  
Opened C:/Users/User/Desktop/Lanning/ParallelProgramming/Lanning_Homework4/Lanning_Homework4/data/Y.bin  
Opened C:/Users/User/Desktop/Lanning/ParallelProgramming/Lanning_Homework4/Lanning_Homework4/data/A.bin  
Opened C:/Users/User/Desktop/Lanning/ParallelProgramming/Lanning_Homework4/Lanning_Homework4/data/b.bin  
  
# elapsed time (training): 3.6213s  
# elapsed time (accuracy): 0.0070812s  
accuracy_test: 0.9219  
# elapsed time (training): 3.65936s  
# elapsed time (accuracy): 0.0072053s  
accuracy_test: 0.9211  
# elapsed time (training): 3.59448s  
# elapsed time (accuracy): 0.0072108s  
accuracy_test: 0.9215  
# elapsed time (training): 3.69181s  
# elapsed time (accuracy): 0.0071657s  
accuracy_test: 0.9216  
# elapsed time (training): 3.61825s  
# elapsed time (accuracy): 0.0073865s  
accuracy_test: 0.922  
# elapsed time (training): 3.66941s  
# elapsed time (accuracy): 0.0072045s  
accuracy_test: 0.9219  
# elapsed time (training): 3.63403s  
# elapsed time (accuracy): 0.0072098s  
accuracy_test: 0.9218  
# elapsed time (training): 3.62578s  
# elapsed time (accuracy): 0.0072107s  
accuracy_test: 0.9219
```

The pretrained softmax weights and biases show a consistent and effective accuracy throughout each iteration because the weights and bias values are from a pre-trained data set. With the dataset already trained we are expected to have consistent and mostly accurate results when making guesses at numbers.

- (d) Compare the advantages and disadvantages of using 1NN vs softmax as the classifier for MNIST data.

Softmax is better due to its speed and ability to be well trained. Softmax mimics a neural network for machine learning that can literally learn things like a human can. In this case it learns the values of handwritten digits. Another common image

recognition application this can be used for is differentiating between cats and dogs, and other various animals. It is a powerful achievement in computing and a very interesting topic that we have been able to go over.