Darrin Willis - dswillis
Bohan Li - bohanl

15-440 Writeup
Lab 4: K-Means Clustering

https://github.com/darrinwillis/distributedSystems

Overview

For this project we implemented a sequential version of the K-Means data cluster analysis algorithm. We then used the sequential version as a baseline to parallelize it and distribute the work using OpenMPI for Java.

Sequential Version

The sequential version of this program performs fairly simply operations. It calculates initial centroids, then reallocates each data point to a cluster, checks for convergence based on the “distance” the centroids have shifted since last iteration. This user defines a maximum allowed travel distance for the centroids, and calculation continues until no centroids move more than this distance.

Parallelized Version

Algorithmically, the parallelized version is identical to the sequential version. The dataset is partitioned based off of how many processors there are, and each process receives its pairwise disjoint, random dataset, which it performs all computation on. The master calculates initial centroids, and then informs all slaves. The slaves then each allocate their own partition of the data to each cluster. The master then receives all of the data, and combines each of the lists. The master then checks for convergence, recalculates new centroids, and repeats. Note that the master calculates new centroids NOT in parallel. We discovered that it is more efficient for the master to perform ALL computations which require the complete set of data. The only part of this algorithm which benefits from parallelization is datapoint allocation.

Cartesian Pairs and DNA

Our implementation was designed to be blind to the type of data it was performing K-means on, only require that the data class implemented a small number of functions to function. The distance function for DNA was simply a count of the number of bases which were different, whereas the distance function for Cartesian pairs was mean squared distance. The averaging function for DNA was just the most common base in each position. We discussed other methods of averaging for DNA, such as taking all bases with a weighted probability, but decided that simply taking the most common was a better approach, given that our dna was totally random.

Darrin Willis - dswillis
Bohan Li - bohanl

Compiling / Running

Simply run `make` to compile all necessary code. `make clean` to clean out all `.class` files.

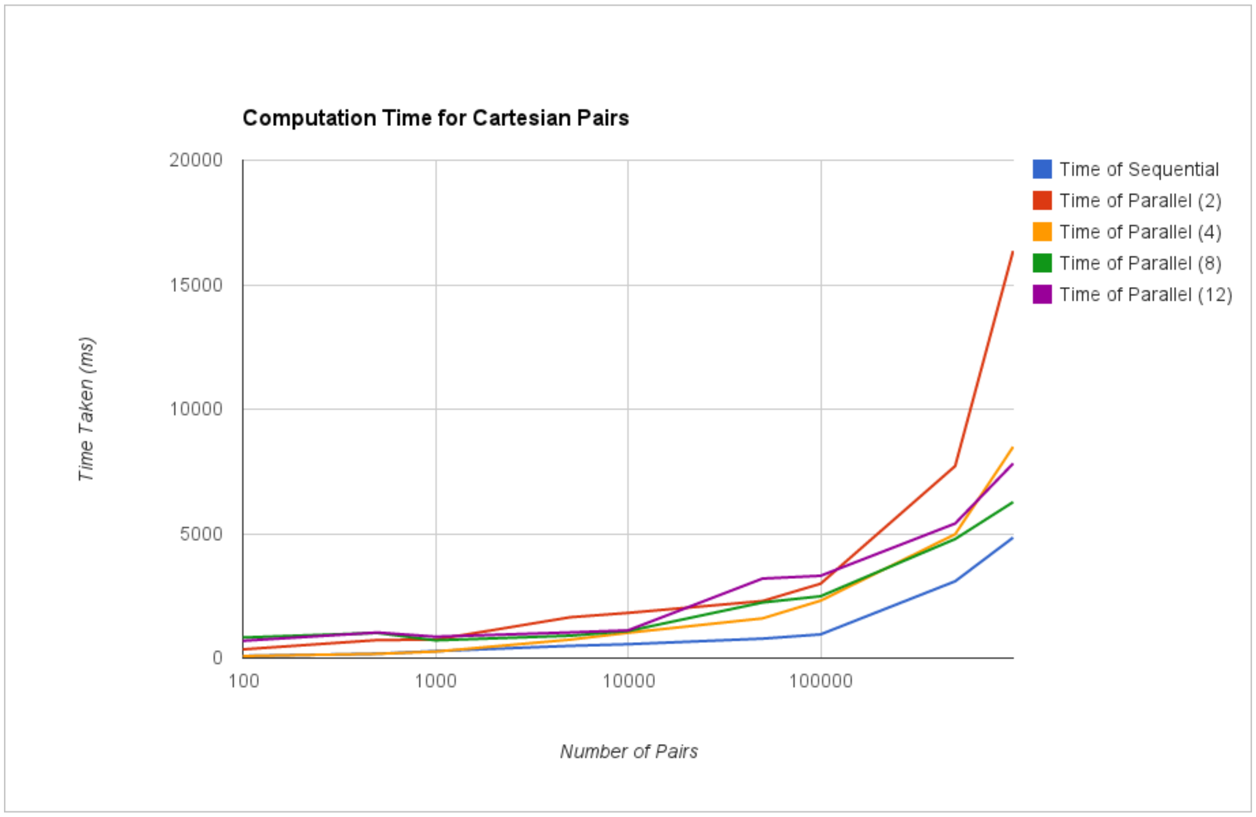
To run a test:

- Sequential:
  - ./seqIntegerTest.sh
    - This tests using cartesian pairs as data points. To modify numbers, see RandomInts.py
  - ./seqDNATest
    - This tests using DNA as data. To modify see RandomDNA.py
- Parallel:
  - ./parallelIntegerTests.sh
    - Test using cartesian pairs. To modify see RandomInts.py
  - ./parallelDNATest.sh
    - Test using DNA. To modify see RandomDNA.py

Also, the scripts used to generate the data for the charts are variations of
./sequentialDataCollection.sh which calls ./sequentialTemp.sh

Darrin Willis - dswillis
Bohan Li - bohanl

Data

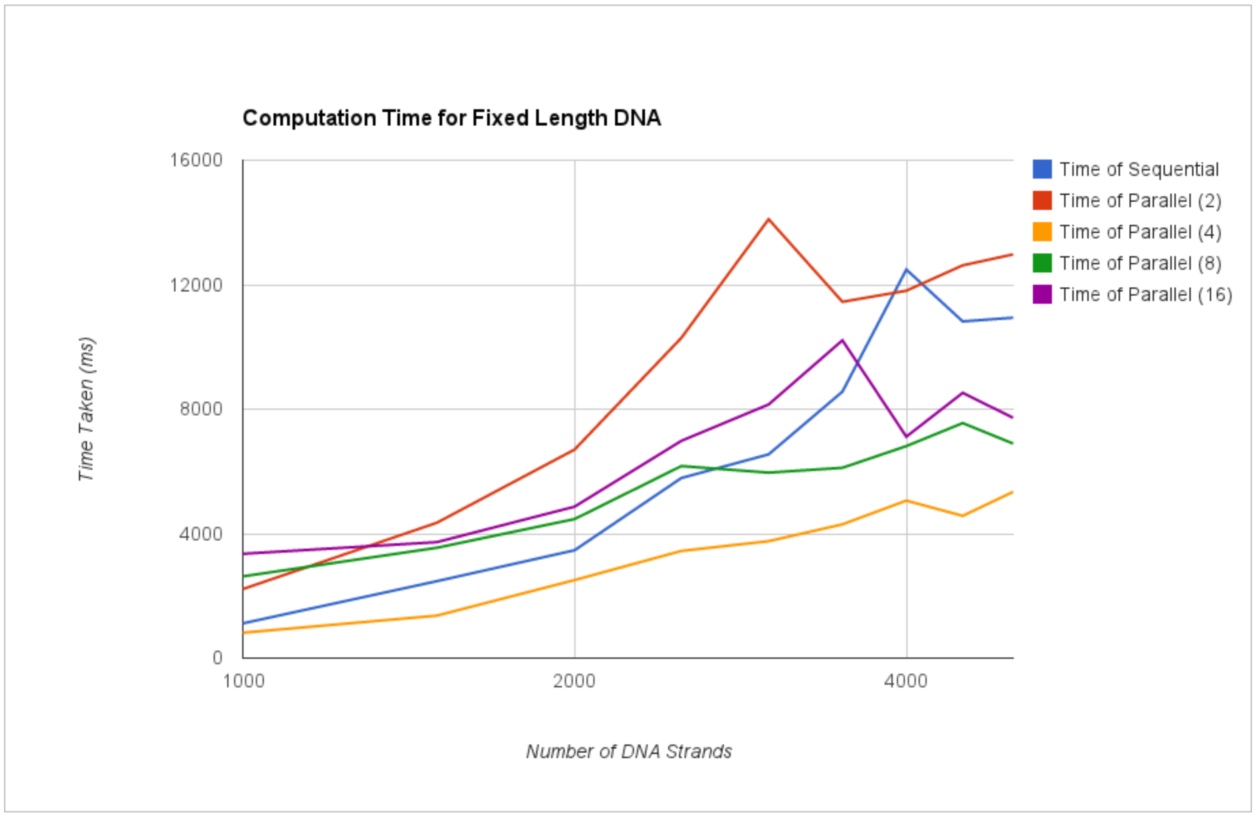


Data Analysis

The sequential time is strictly less than the parallelized time in this case. The reason is simply due to the fact that the communication overhead is too great to result in significant improvement. Also, the java implementation of MPI necessarily requires a bit of hoop-jumping to function correctly, and this results in some additional overhead of manipulating data. However, more predictably, the computation time for the parallelized versions decreases as a function of the number of processes, until 12, which then rises again. This indicates that parallelizing our algorithm does in fact improve its run time.

Darrin Willis - dswillis
Bohan Li - bohanl

Data



Data Analysis

The DNA shows a similar general trend to the Cartesian Pairs. One thing to note about the DNA K-Means processing is that it tended to have a much greater variability in the length of run time, due to variability in the number of K-means iterations it would take to converge. This value varied by about +- 50%. This caused our data to be slightly less reliable, but still showed a distinctive trend. Also, the DNA complexity had to be analyzed differently, because it simply took too long to plot on a logarithmic scale of length. This is just a result of DNA being a quaternary system, rather than a binary system, and also because it involves processing strings rather than ints.