15-440 Project #4

5 December 2013

Vinay Balaji vbalaji@andrew.cmu.edu
Ankit Chheda achheda@andrew.cmu.edu

Parallelization Approach

---

1) Randomly select k centroids from the initial data set, where k is the requested number of clusters.

2) Partition the initial data set into p splits, where p is the requested number of processes to use for parallelization.

3) Master node sends each worker, and itself, a unique split of the data set, as well as all of the current centroids.

4) Each worker, and the master, maps the data points in its split to the nearest centroid.

5) All workers submit their results to the master, which aggregates all of the data points.

6) Once all workers have submitted their work to the master, the master then computes the true mean of each cluster.

7) If, across all the clusters, the difference between a cluster's true mean and its centroid value is within some tolerance threshold ϵ, then we terminate. Else, we repeat steps 2 through 7.

Pseudocode

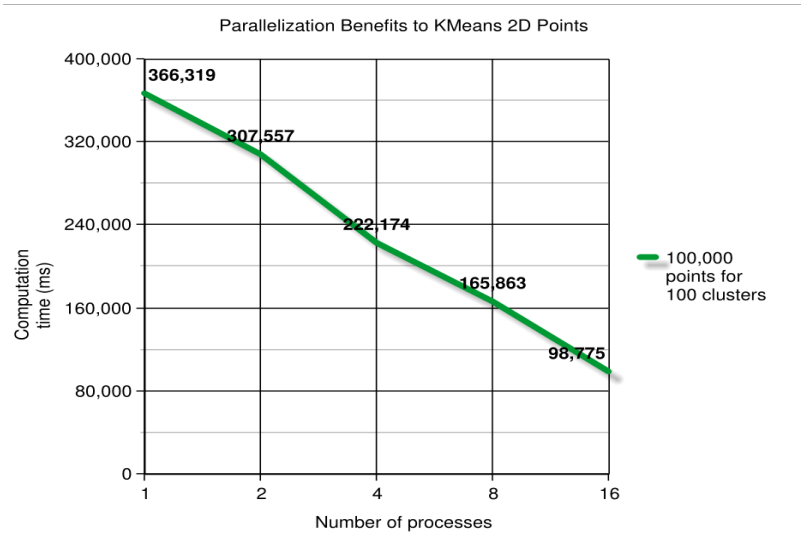
```
centroids = select_random(k, data) // selects k random points from the data
                                   // k is number of clusters
results = null
done = false

do {
    splits = partition_data(p, data) // partition data into p splits
                                     // p is number of processes
    for (split : splits) {
        assign_split_to_worker(split) // distribute splits among workers
    }

    mapped_points = map_points_to_centroids(mysplit, centroids) // calculate nearest centroid for
                                                                // each data point in our split
    if (self_is_worker()) send_to_master(mapped_points) // workers send results back to master
    if (self_is_master()) {
        results = collect_worker_results() // master collects results
        new_centroids = recalculate_centroids(results) // recalculate actual means (new centroids)

        done = true // we may be done
                    // but let's check to be sure
        for (<centroid, new_centroid> : <centroids, new_centroids>) {
            if (abs(centroid - new_centroid) > e) {
                done = false // centroids are not stable
                break
            }
        }
    }
} while (!done)

return results
```

Experimental Results

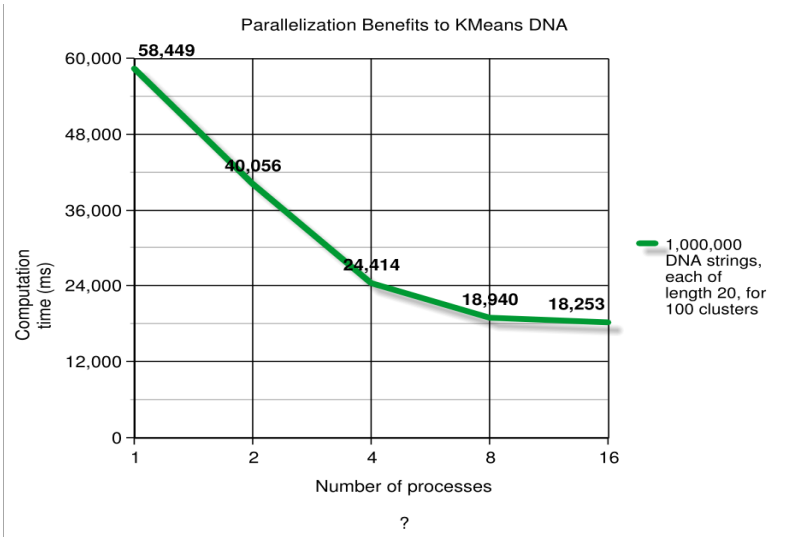
Note

We could only get MPI running across two GHC machines at once, in particular only the GHC machines that prompted the user for a password upon ssh. When trying to add a third machine, since the only machines that seemed to work required passwords, the simultaneous password request ended up with neither machine receiving the correct password. Hence, all of our results are based on running MPI with exactly two machines.

2D Parallelization



DNA Parallelization



?

Clearly, our parallel solution induces a run-time speedup. We feel that the reason that DNA seems to benefit less from parallelization is that computing differences and averages for DNA strands is more computationally intensive than for comparing two 2d-points.

How to Run

To compile our project, do the following:

```
$> ./compile.sh
```

To run a particular test program, do the following:

```
$> ./run.sh <# of processes (1 if serial)> <name of test program> <test program args>
```

Test Programs

Serial:

`TestSerialKMeansDNA` `TestSerialKMeans2D`

Parallel:

`TestParallelKMeansDNA` `TestParallelKMeans2D`

DNA tests require `<number of strands>` `<size of each strand>` `<num clusters>` `<centroid threshold>`
2D tests require `<number of points>` `<num clusters>` `<centroid threshold>`

Examples

To run DNA in serial with 1000 strands of length 20, finding 10 clusters, with threshold 0:
`$> ./run.sh 1 TestSerialKMeansDNA 1000 20 10 0`

To run DNA in parallel with 1000 strands of length 20, finding 10 clusters, with threshold 0 across 4 processes:
`$> ./run.sh 4 TestParallelKMeansDNA 1000 20 10 0`

To run 2D in serial with 1000 points finding 10 clusters, with threshold 0:
`$> ./run.sh 1 TestSerialKMeans2D 1000 10 0`

To run 2D in parallel with 1000 points finding 10 clusters, with threshold 0 across 4 processes:
`$> ./run.sh 4 TestParallelKMeans2D 1000 10 0`

Notes

1) You may be prompted to enter your password (GHC machines are being weird)

2) Input is not checked. If invalid (eg: negative) values are entered, program might crash. Also, all arguments are required.

3) For additional information about MPI as we have used it in our code, please visit:
http://www.open-mpi.org/faq/?category=running
http://www.open-mpi.org/faq/?category=java