CSS 534 Program 5 Report

Spatial Area Skyline Query

Vedang Parasnis, Atul Ahire, Manpreet Kaur

11th of December 2023

Application Overview:

Grid Area Skyline Query:
Our program focuses on parallel implementation of Grid Area Skyline Algorithm (GASKY) to recommend the best location in a large grid denoting a spatial region, while each cell resembles some spatial location on the map. We consider two main types of facilities (Favorable, Unfavourable) and different types of facilities which are geographically distributed across different grid cells. Considering larger size grids, euclidean distance across each pair is computationally heavy and unscalable, we implemented an efficient approach using Voronoi Polygons, and Min Max distance algorithm.

We use two algorithms, Dominance relation finding among points using Voronoi Polygons (Local Skyline respective to each facility type done in parallel), Min Max distance algorithm for each cell in the grid to filter out points that are much closer to unfavorable facilities (Global Skyline Objects). To implement these algorithms, we went with following parallel algorithm design across all parallel implementations.

Finding Concurrency: Task Parallelism → Each facility is computed independently, embracingly parallel).
Decomposition Pattern: Task Driven Data Decomposition → Each Facility Type determines the task that internally decomposes the spatial grid covering respective facility coordinates.
Algorithmic Structure: Divide and Conquer → Divide Facility to compute Local Skyline, conquer them to find global Skyline
Distributed Data Structure: Spatial Grid resembling a multi-dimensional spatial region.

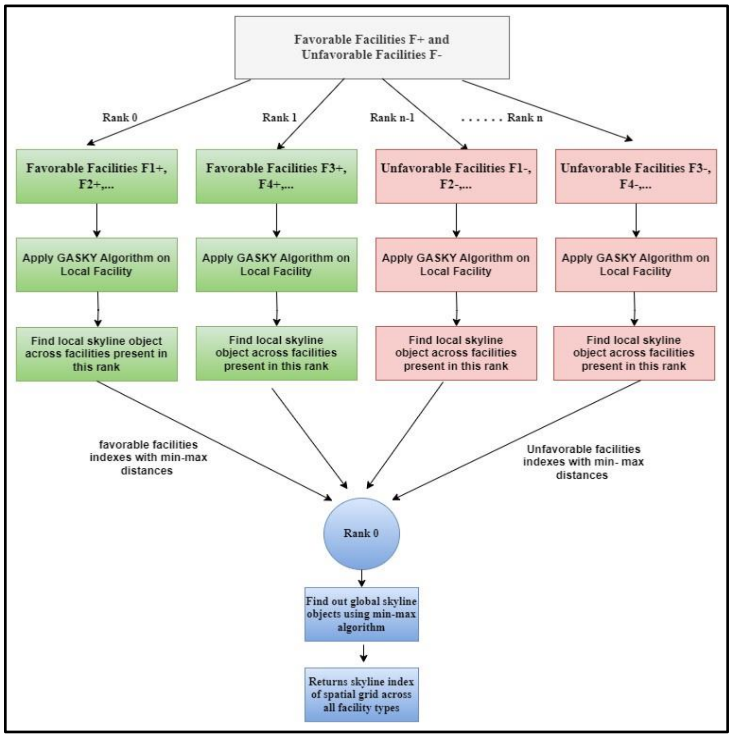
Source Code:

All the source code is included in the root directory of the submission , with respective maven modules for each of them mpi, mapreduce, spark and mass.

Documentation:

1. MPIJava

Parallelization strategy used for MPI is task parallelism where computational tasks are divided into subtasks which can be executed concurrently by different processes.



Favorable Facilities F+ and Unfavorable Facilities F-
Rank 0
Rank 1
Rank n-1
. . . . . . Rank n
Favorable Facilities F1+, F2+,...
Favorable Facilities F3+, F4+,...
Unfavorable Facilities F1-, F2-,...
Unfavorable Facilities F3-, F4-,...
Apply GASKY Algorithm on Local Facility
Find local skyline object across facilities present in this rank
favorable facilities indexes with min-max distances
Unfavorable facilities indexes with min- max distances
Rank 0
Find out global skyline objects using min-max algorithm
Returns skyline index of spatial grid across all facility types

Figure 1. MPI Parallelization

Following are the steps taken to achieve parallelization -

1. Input is read from a file where a favorable and unfavorable facility's binary matrix is present.
2. Ranks are divided into 2 sets in which half of ranks will process favorable facilities and remaining ones will process unfavorable facilities. The reason behind keeping favorable and unfavorable facility computation on different ranks is because the local skyline computation follows task parallelism (embracingly parallel) and has no data dependency among ranks.
3. Each rank knows what kind of facility and number of facilities to be processed. This has been achieved by calculating the number of facilities to be processed by each rank and keeping track of facility count and its type into separate arrays.
4. Each rank now will read its assigned facility data into a 3D array where the first dimension is used to track different facilities and the remaining dimension is used to track x and y coordinates of the facility.
5. Once a facility grid is initialized and loaded into memory, Grid based Area Skyline(GASKY) algorithm is applied on each facility grid present in that rank.
6. Each rank returns the calculated local skyline objects gathered at rank 0. This collective communication is needed as a Min-max algorithm which computes global skyline points based on favorable and unfavorable distance grids which are present across different ranks.
7. Finally, Global skylines are calculated on Rank-0.

2. MapReduce:

Our MapReduce Implementation follows Task Parallelism, we apply Map Reduce based Grid Area Skyline Algorithm (MRGASKY) to compute skyline objects. The following diagram resembles the flow we use.

Job1: Processes the binary grid input from HDFS (we group the input based on the facility type and binary values resembles spatial locations in the region).

1. Map: Compute the distance across each row, emit K/V as ((Facility-Type, Column), Distance)
2. Shuffle Sort: Custom comparator to compare the Facility type in key, if equal sort based on Column Number
3. Reducer: Identify the dominant points respective to a specific column (key extracted) and calculate the distance from it across all the x row projections (Iterable<Values>) lying in the same proximity interval (voronoi polygons used).

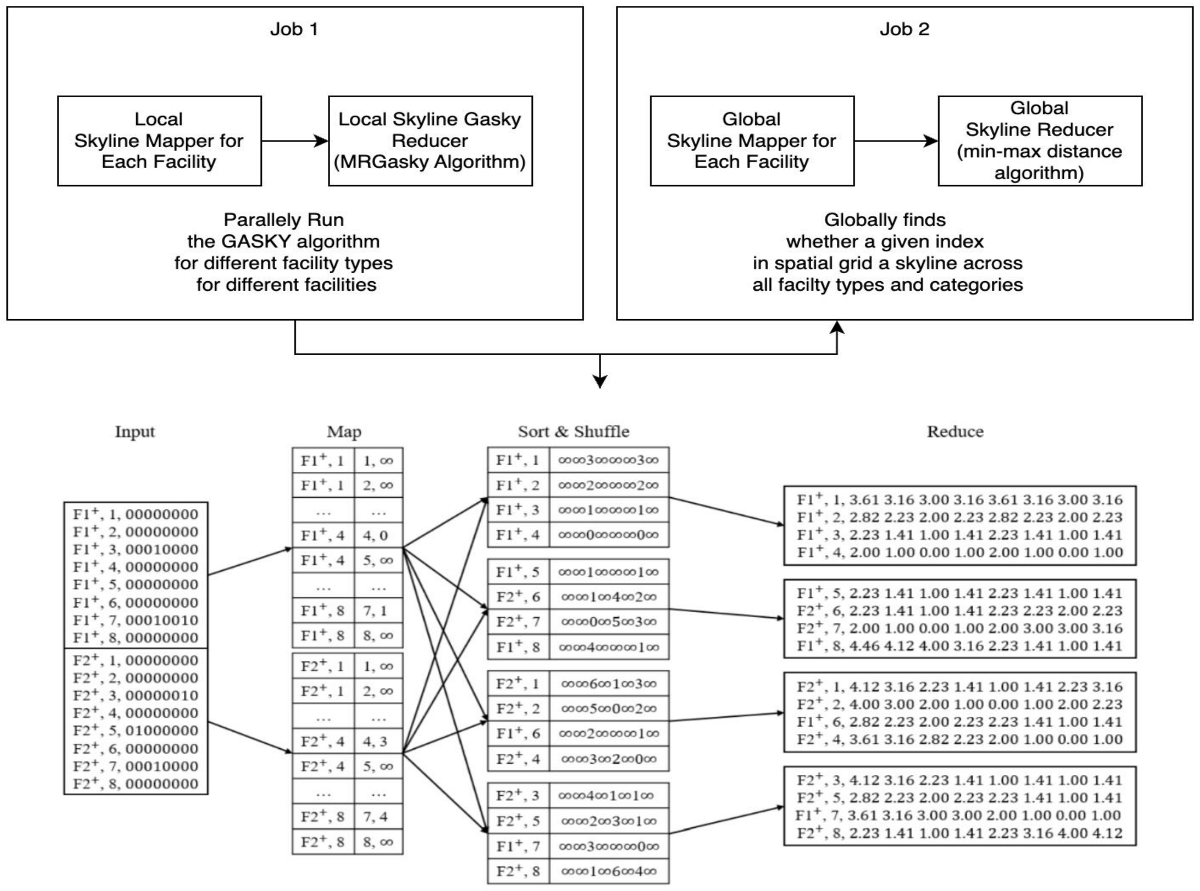

Job 1
Local Skyline Mapper for Each Facility
Local Skyline Gasky Reducer (MRGasky Algorithm)
Parallely Run the GASKY algorithm for different facility types for different facilities
Job 2
Global Skyline Mapper for Each Facility
Global Skyline Reducer (min-max distance algorithm)
Globally finds whether a given index in spatial grid a skyline across all facilty types and categories
Input
Map
Sort & Shuffle
Reduce
$F1^{+}$, 1, 00000000
$F1^{+}$, 2, 00000000
$F1^{+}$, 3, 00010000
$F1^{+}$, 4, 00000000
$F1^{+}$, 5, 00000000
$F1^{+}$, 6, 00000000
$F1^{+}$, 7, 00010010
$F1^{+}$, 8, 00000000
$F2^{+}$, 1, 00000000
$F2^{+}$, 2, 00000000
$F2^{+}$, 3, 00000010
$F2^{+}$, 4, 00000000
$F2^{+}$, 5, 01000000
$F2^{+}$, 6, 00000000
$F2^{+}$, 7, 00010000
$F2^{+}$, 8, 00000000
$F1^{+}$, 1 | 1, ∞
$F1^{+}$, 1 | 2, ∞
... | ...
$F1^{+}$, 4 | 4, 0
$F1^{+}$, 4 | 5, ∞
... | ...
$F1^{+}$, 8 | 7, 1
$F1^{+}$, 8 | 8, ∞
$F2^{+}$, 1 | 1, ∞
$F2^{+}$, 1 | 2, ∞
... | ...
$F2^{+}$, 4 | 4, 3
$F2^{+}$, 4 | 5, ∞
... | ...
$F2^{+}$, 8 | 7, 4
$F2^{+}$, 8 | 8, ∞
$F1^{+}$, 1 | ∞∞3∞∞∞∞3∞
$F1^{+}$, 2 | ∞∞2∞∞∞∞2∞
$F1^{+}$, 3 | ∞∞1∞∞∞∞1∞
$F1^{+}$, 4 | ∞∞0∞∞∞∞0∞
$F1^{+}$, 5 | ∞∞1∞∞∞∞1∞
$F2^{+}$, 6 | ∞∞1∞4∞2∞
$F2^{+}$, 7 | ∞∞0∞5∞3∞
$F1^{+}$, 8 | ∞∞4∞∞∞∞1∞
$F2^{+}$, 1 | ∞∞6∞1∞3∞
$F2^{+}$, 2 | ∞∞5∞0∞2∞
$F1^{+}$, 6 | ∞∞2∞∞∞∞1∞
$F2^{+}$, 4 | ∞∞3∞2∞0∞
$F2^{+}$, 3 | ∞∞4∞1∞1∞
$F2^{+}$, 5 | ∞∞2∞3∞1∞
$F1^{+}$, 7 | ∞∞3∞∞∞∞0∞
$F2^{+}$, 8 | ∞∞1∞6∞4∞
$F1^{+}$, 1, 3.61 3.16 3.00 3.16 3.61 3.16 3.00 3.16
$F1^{+}$, 2, 2.82 2.23 2.00 2.23 2.82 2.23 2.00 2.23
$F1^{+}$, 3, 2.23 1.41 1.00 1.41 2.23 1.41 1.00 1.41
$F1^{+}$, 4, 2.00 1.00 0.00 1.00 2.00 1.00 0.00 1.00
$F1^{+}$, 5, 2.23 1.41 1.00 1.41 2.23 1.41 1.00 1.41
$F2^{+}$, 6, 2.23 1.41 1.00 1.41 2.23 2.23 2.00 2.23
$F2^{+}$, 7, 2.00 1.00 0.00 1.00 2.00 3.00 3.00 3.16
$F1^{+}$, 8, 4.46 4.12 4.00 3.16 2.23 1.41 1.00 1.41
$F2^{+}$, 1, 4.12 3.16 2.23 1.41 1.00 1.41 2.23 3.16
$F2^{+}$, 2, 4.00 3.00 2.00 1.00 0.00 1.00 2.00 2.23
$F1^{+}$, 6, 2.82 2.23 2.00 2.23 2.23 1.41 1.00 1.41
$F2^{+}$, 4, 3.61 3.16 2.82 2.23 2.00 1.00 0.00 1.00
$F2^{+}$, 3, 4.12 3.16 2.23 1.41 1.00 1.41 1.00 1.41
$F2^{+}$, 5, 2.82 2.23 2.00 2.23 2.23 1.41 1.00 1.41
$F1^{+}$, 7, 3.61 3.16 3.00 3.00 2.00 1.00 0.00 1.00
$F2^{+}$, 8, 2.23 1.41 1.00 1.41 2.23 3.16 4.00 4.12

Figure 2. Map reduce parallelization strategy

Job2: Reads the input from Job, apply the second phase of the map reduce algorithm.

1. Map: For each value splitted by , emit K/ V ((Row, Column), (Facility Category, Distance)),
2. Sort and shuffle: Custom Comparator to sort them according to the key (Row, Column).
3. Reducer: Apply the min max distance algorithm considering values ((Iterable<Facility Category, Distance>), don't write the points keys (Row, Column) for which the problem constraints are not satisfied and min-max distance algorithm is not satisfied.

Note:
Both the algorithms are implemented in respective reducers, mappers with custom comparators provide the required input format required to execute the algorithm.

3. Spark

The Spark implementation utilizes several parallelization techniques to enhance the efficiency of computation..

- The algorithm starts by reading input data from HDFS using Spark's textFile method. Each line of the input corresponds to a binary matrix representing the spatial configuration of favorable and unfavorable facilities.
- The input data is partitioned using the repartition method. This ensures that the workload is evenly distributed across the Spark cluster nodes, enhancing parallel processing.
- Local Skyline Computation:
    - Map Phase:
      The flatMapToPair transformation is used to parse and convert the input data into key-value pairs. Each key represents a facility, and the values are tuples containing row numbers and corresponding binary values.The data is grouped by facility and row number, and the distance from the favorable/unfavorable axis is calculated for each row. These distances are emitted as key-value pairs ((Facility, Column), Distance).
    - Shuffle & Sort Phase:
      A custom comparator is applied during the shuffle and sort phase. It ensures that the data is sorted based on facility type and column number.
    - Reduce Phase:
      The reducer identifies dominant points for each column, considering the calculated distances. The result is a list of tuples containing column numbers and their corresponding skyline distances.
- The calculated skyline distances are processed to determine proximity intervals. These intervals represent regions where the dominance relationship changes.
- Global Skyline Computation:
    - For each proximity interval, the algorithm calculates the minimum Euclidean distance from each point to the dominant points in the interval.
    - The global skyline points are obtained by considering the minimum distance from each point to the dominant points across all proximity intervals.
- The computed global skyline points are filtered to retain only those points that satisfy the problem constraints using min max distance algorithm.
- The final ordered coordinate projections of skyline points are generated and saved using the saveAsTextFile method.
- The Spark implementation is rigorously tested on a Spark cluster, demonstrating scalability and efficiency in handling large-scale spatial datasets.
- The algorithm leverages key Spark transformations such as flatMapToPair, mapValues, reduceByKey, groupByKey, sortByKey, filter, and coalesce to achieve task parallelism, data partitioning, parallel sorting, parallel reductions, and efficient data shuffling.

The implementation was tested on a Spark cluster, and experimental results demonstrate the scalability and efficiency of the parallelized MrGasky algorithm. The distributed computation effectively handles large-scale datasets, providing skyline points with reduced processing time.

4. MASS

For Mass the primary parallelization strategy is to use only places and no agents. Since we have task parallelism, at the start each task is almost embarrassingly parallel. However, switching parallelism from task to data parallelism is also not achievable because the tasks have partitioned the data relative to its requirement (task driven data decomposition). Since, as mentioned earlier, each task resembles parallel local skyline computation on a facility type, however, for global skyline we need to collect the data across each cell in the spatial grid to determine global skyline.

In MASS we went with the following approach.

1. Each place resembles a type of facility (Favorable / Unfavorable).
2. Each place extends the base place class, to add more features to that place to be handled in parallel by the MASS core library using threads.Each place hold respective information about
   a. Spatial region coordinates.
   b. Spatial region size
   c. Spatial region distance based on its coordinates.
3. Since, we don't use agents we use place.callAll n times until all stages of algorithm are completed..
4. Overall these are the calls we perform across places to achieve parallelization across places.

```
 spatialGrid.callAll(INIT, fx);  // use call all to configure all places
 spatialGrid.callAll(INIT_BINARY_MATRIX, tokens); // parallely to fill the binary matrix for each pplace
 spatialGrid.callAll(COMPUTE_BEST_ROW_DISTANCE);  // to compute row wise distance parallely for each grid in the place
 spatialGrid.callAll(COMPUTE_PROXIMITY_POLYGONS); // run the dominanance relation algorithm parallelly for all places.
     for (int i=0; i < finalFilteredDistancesIndex.length; i++){
         Object[] data = spatialGrid.callAll(COMPUTE_GLOBAL_SKYLINE) // global skyline computation call all with specified
index across places / facilities
```

Execution Output:

We have tested the execution for a grid size of 512 * 512 with 16 facility count

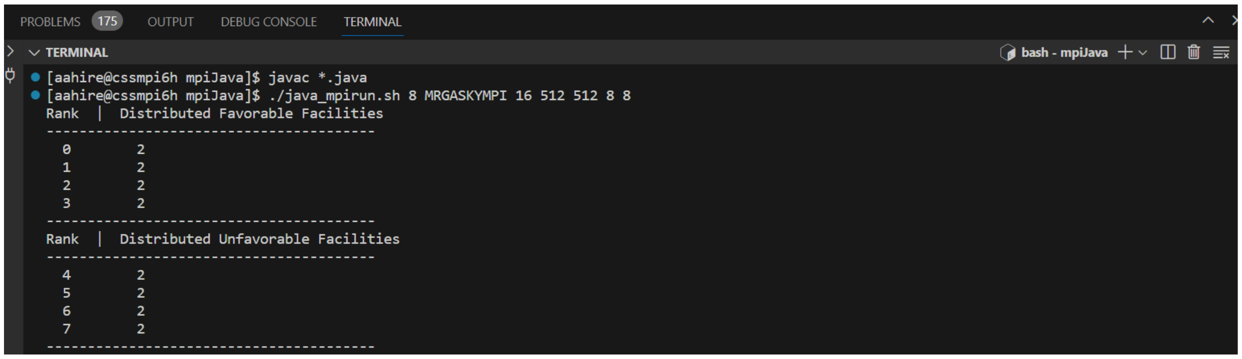
8 favorable and 8 unfavorable, the output skyline objects are too high to be displayed in single output attached the excel sheet covering output:
Output Results **Sheet 1**

MPI :

To execute the MPI program execute the following command:

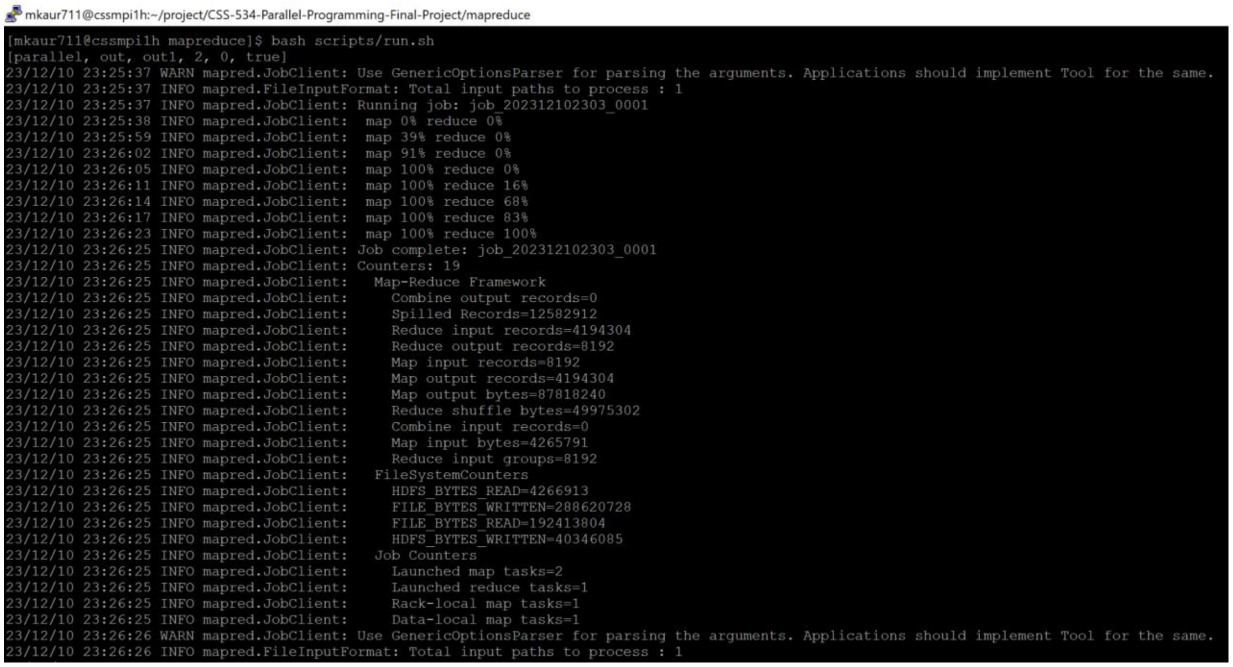
./java_mpirun.sh 8 MRGASKYMPI 16 512 512 8 8

Usage: java MRGASKYMPI numberOfFacilities size size favourable unfavourable

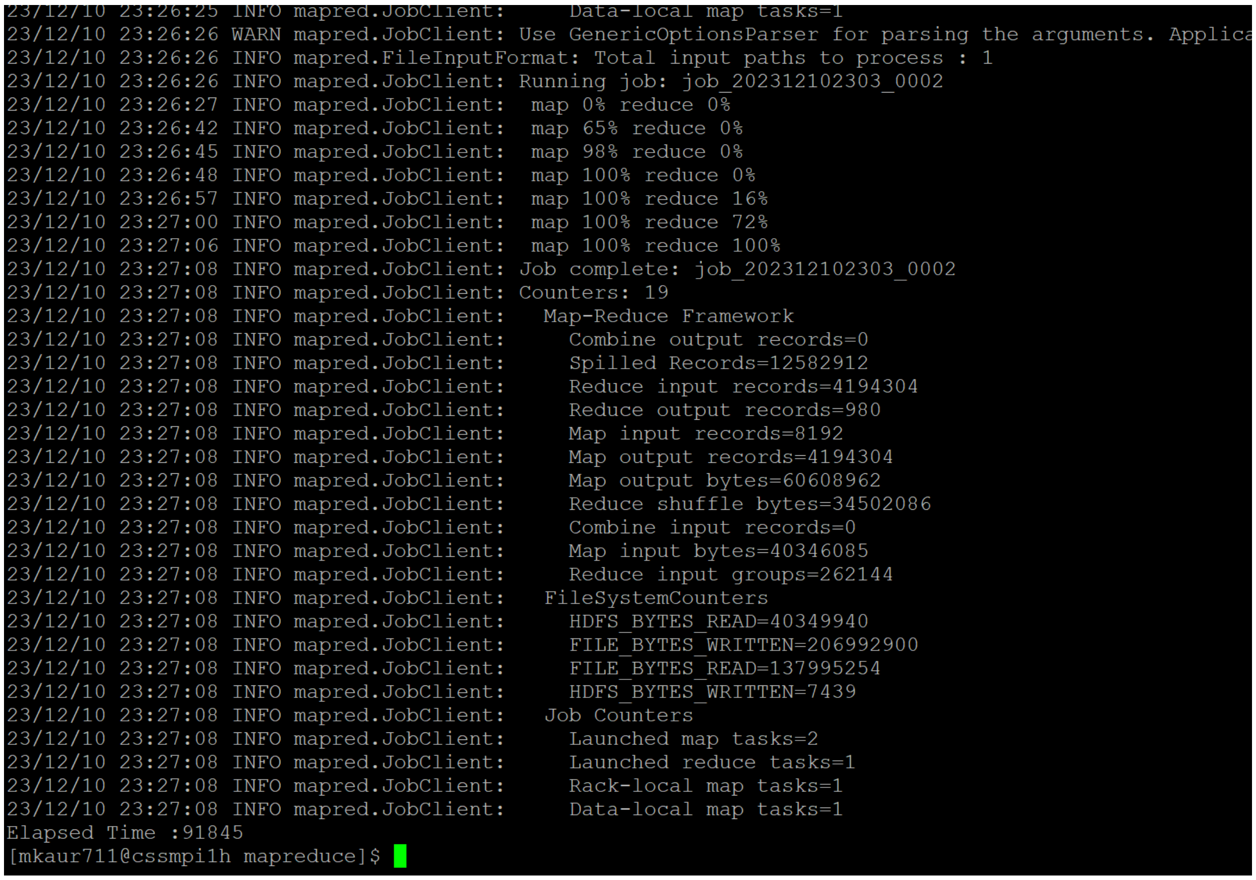


```
[aahire@cssmpi6h mpiJava]$ javac *.java
[aahire@cssmpi6h mpiJava]$ ./java_mpirun.sh 8 MRGASKYMPI 16 512 512 8 8
Rank  |  Distributed Favorable Facilities
----------------------------------------
  0        2
  1        2
  2        2
  3        2
----------------------------------------
Rank  |  Distributed Unfavorable Facilities
----------------------------------------
  4        2
  5        2
  6        2
  7        2
----------------------------------------
```

MapReduce:



```
mkaur711@cssmpi1h:~/project/CSS-534-Parallel-Programming-Final-Project/mapreduce
[mkaur711@cssmpi1h mapreduce]$ bash scripts/run.sh
[parallel, out, out1, 2, 0, true]
23/12/10 23:25:37 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applications should implement Tool for the same.
23/12/10 23:25:37 INFO mapred.FileInputFormat: Total input paths to process : 1
23/12/10 23:25:37 INFO mapred.JobClient: Running job: job_202312102303_0001
23/12/10 23:25:38 INFO mapred.JobClient:  map 0% reduce 0%
23/12/10 23:25:59 INFO mapred.JobClient:  map 39% reduce 0%
23/12/10 23:26:02 INFO mapred.JobClient:  map 91% reduce 0%
23/12/10 23:26:05 INFO mapred.JobClient:  map 100% reduce 0%
23/12/10 23:26:11 INFO mapred.JobClient:  map 100% reduce 16%
23/12/10 23:26:14 INFO mapred.JobClient:  map 100% reduce 68%
23/12/10 23:26:17 INFO mapred.JobClient:  map 100% reduce 83%
23/12/10 23:26:23 INFO mapred.JobClient:  map 100% reduce 100%
23/12/10 23:26:25 INFO mapred.JobClient: Job complete: job_202312102303_0001
23/12/10 23:26:25 INFO mapred.JobClient: Counters: 19
23/12/10 23:26:25 INFO mapred.JobClient:   Map-Reduce Framework
23/12/10 23:26:25 INFO mapred.JobClient:     Combine output records=0
23/12/10 23:26:25 INFO mapred.JobClient:     Spilled Records=12582912
23/12/10 23:26:25 INFO mapred.JobClient:     Reduce input records=4194304
23/12/10 23:26:25 INFO mapred.JobClient:     Reduce output records=8192
23/12/10 23:26:25 INFO mapred.JobClient:     Map input records=8192
23/12/10 23:26:25 INFO mapred.JobClient:     Map output records=4194304
23/12/10 23:26:25 INFO mapred.JobClient:     Map output bytes=87818240
23/12/10 23:26:25 INFO mapred.JobClient:     Reduce shuffle bytes=49975302
23/12/10 23:26:25 INFO mapred.JobClient:     Combine input records=0
23/12/10 23:26:25 INFO mapred.JobClient:     Map input bytes=4265791
23/12/10 23:26:25 INFO mapred.JobClient:     Reduce input groups=8192
23/12/10 23:26:25 INFO mapred.JobClient:   FileSystemCounters
23/12/10 23:26:25 INFO mapred.JobClient:     HDFS_BYTES_READ=4266913
23/12/10 23:26:25 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=288620728
23/12/10 23:26:25 INFO mapred.JobClient:     FILE_BYTES_READ=192413804
23/12/10 23:26:25 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=40346085
23/12/10 23:26:25 INFO mapred.JobClient:   Job Counters
23/12/10 23:26:25 INFO mapred.JobClient:     Launched map tasks=2
23/12/10 23:26:25 INFO mapred.JobClient:     Launched reduce tasks=1
23/12/10 23:26:25 INFO mapred.JobClient:     Rack-local map tasks=1
23/12/10 23:26:25 INFO mapred.JobClient:     Data-local map tasks=1
23/12/10 23:26:26 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applications should implement Tool for the same.
23/12/10 23:26:26 INFO mapred.FileInputFormat: Total input paths to process : 1
```

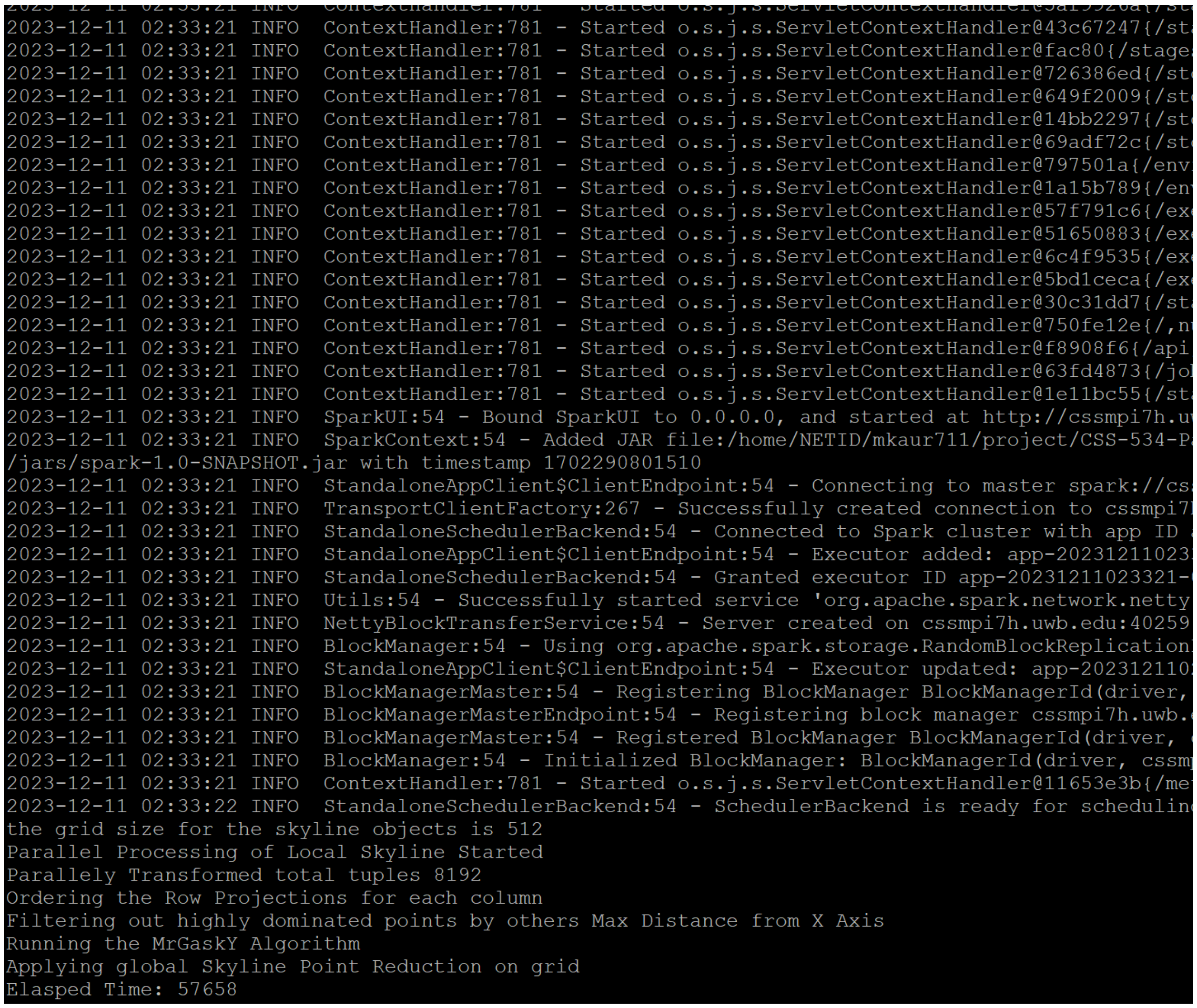


```
23/12/10 23:26:25 INFO mapred.JobClient:     Data-local map tasks=1
23/12/10 23:26:26 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applica
23/12/10 23:26:26 INFO mapred.FileInputFormat: Total input paths to process : 1
23/12/10 23:26:26 INFO mapred.JobClient: Running job: job_202312102303_0002
23/12/10 23:26:27 INFO mapred.JobClient:  map 0% reduce 0%
23/12/10 23:26:42 INFO mapred.JobClient:  map 65% reduce 0%
23/12/10 23:26:45 INFO mapred.JobClient:  map 98% reduce 0%
23/12/10 23:26:48 INFO mapred.JobClient:  map 100% reduce 0%
23/12/10 23:26:57 INFO mapred.JobClient:  map 100% reduce 16%
23/12/10 23:27:00 INFO mapred.JobClient:  map 100% reduce 72%
23/12/10 23:27:06 INFO mapred.JobClient:  map 100% reduce 100%
23/12/10 23:27:08 INFO mapred.JobClient: Job complete: job_202312102303_0002
23/12/10 23:27:08 INFO mapred.JobClient: Counters: 19
23/12/10 23:27:08 INFO mapred.JobClient:   Map-Reduce Framework
23/12/10 23:27:08 INFO mapred.JobClient:     Combine output records=0
23/12/10 23:27:08 INFO mapred.JobClient:     Spilled Records=12582912
23/12/10 23:27:08 INFO mapred.JobClient:     Reduce input records=4194304
23/12/10 23:27:08 INFO mapred.JobClient:     Reduce output records=980
23/12/10 23:27:08 INFO mapred.JobClient:     Map input records=8192
23/12/10 23:27:08 INFO mapred.JobClient:     Map output records=4194304
23/12/10 23:27:08 INFO mapred.JobClient:     Map output bytes=60608962
23/12/10 23:27:08 INFO mapred.JobClient:     Reduce shuffle bytes=34502086
23/12/10 23:27:08 INFO mapred.JobClient:     Combine input records=0
23/12/10 23:27:08 INFO mapred.JobClient:     Map input bytes=40346085
23/12/10 23:27:08 INFO mapred.JobClient:     Reduce input groups=262144
23/12/10 23:27:08 INFO mapred.JobClient:   FileSystemCounters
23/12/10 23:27:08 INFO mapred.JobClient:     HDFS_BYTES_READ=40349940
23/12/10 23:27:08 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=206992900
23/12/10 23:27:08 INFO mapred.JobClient:     FILE_BYTES_READ=137995254
23/12/10 23:27:08 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=7439
23/12/10 23:27:08 INFO mapred.JobClient:   Job Counters
23/12/10 23:27:08 INFO mapred.JobClient:     Launched map tasks=2
23/12/10 23:27:08 INFO mapred.JobClient:     Launched reduce tasks=1
23/12/10 23:27:08 INFO mapred.JobClient:     Rack-local map tasks=1
23/12/10 23:27:08 INFO mapred.JobClient:     Data-local map tasks=1
Elapsed Time :91845
[mkaur711@cssmpi1h mapreduce]$
```

Spark:

To execute the spark program execute the following command:

Spark-submit --class com.css534.parallel.Main --master spark://cssmpi7h.uwb.edu:58140 --executor-memory 4G --total-executor-cores 5 target/spark-1.0-SNAPSHOT.jar inputs.txt 512



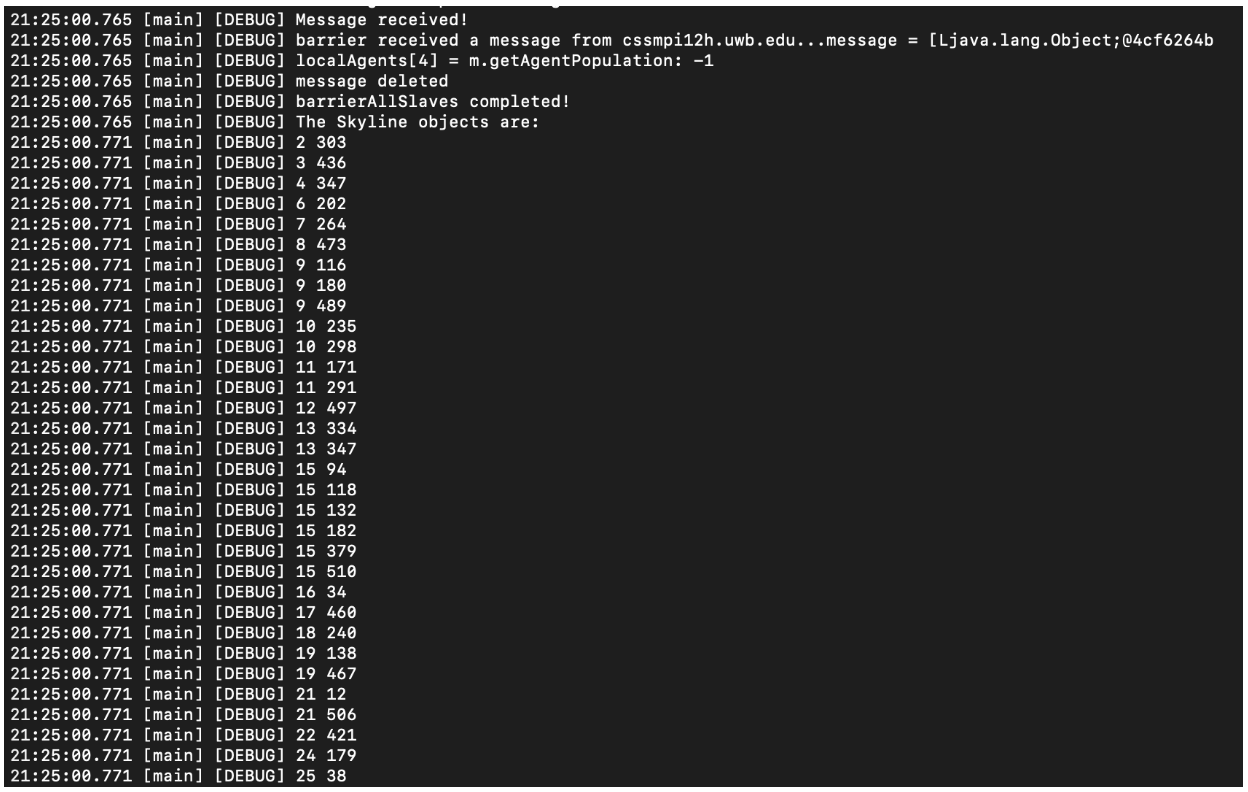
```
2023-12-11 02:33:21 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@5af9926a{/sta
2023-12-11 02:33:21 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@43c67247{/sta
2023-12-11 02:33:21 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@fac80{/stage
2023-12-11 02:33:21 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@726386ed{/st
2023-12-11 02:33:21 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@649f2009{/st
2023-12-11 02:33:21 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@14bb2297{/st
2023-12-11 02:33:21 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@69adf72c{/st
2023-12-11 02:33:21 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@797501a{/env
2023-12-11 02:33:21 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@1a15b789{/en
2023-12-11 02:33:21 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@57f791c6{/ex
2023-12-11 02:33:21 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@51650883{/ex
2023-12-11 02:33:21 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@6c4f9535{/ex
2023-12-11 02:33:21 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@5bd1ceca{/ex
2023-12-11 02:33:21 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@30c31dd7{/st
2023-12-11 02:33:21 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@750fe12e{/,n
2023-12-11 02:33:21 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@f8908f6{/api
2023-12-11 02:33:21 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@63fd4873{/jo
2023-12-11 02:33:21 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@1e11bc55{/st
2023-12-11 02:33:21 INFO  SparkUI:54 - Bound SparkUI to 0.0.0.0, and started at http://cssmpi7h.u
2023-12-11 02:33:21 INFO  SparkContext:54 - Added JAR file:/home/NETID/mkaur711/project/CSS-534-P
/jars/spark-1.0-SNAPSHOT.jar with timestamp 1702290801510
2023-12-11 02:33:21 INFO  StandaloneAppClient$ClientEndpoint:54 - Connecting to master spark://cs
2023-12-11 02:33:21 INFO  TransportClientFactory:267 - Successfully created connection to cssmpi7
2023-12-11 02:33:21 INFO  StandaloneSchedulerBackend:54 - Connected to Spark cluster with app ID
2023-12-11 02:33:21 INFO  StandaloneAppClient$ClientEndpoint:54 - Executor added: app-2023121102
2023-12-11 02:33:21 INFO  StandaloneSchedulerBackend:54 - Granted executor ID app-20231211023321-
2023-12-11 02:33:21 INFO  Utils:54 - Successfully started service 'org.apache.spark.network.netty
2023-12-11 02:33:21 INFO  NettyBlockTransferService:54 - Server created on cssmpi7h.uwb.edu:40259
2023-12-11 02:33:21 INFO  BlockManager:54 - Using org.apache.spark.storage.RandomBlockReplication
2023-12-11 02:33:21 INFO  StandaloneAppClient$ClientEndpoint:54 - Executor updated: app-20231211
2023-12-11 02:33:21 INFO  BlockManagerMaster:54 - Registering BlockManager BlockManagerId(driver,
2023-12-11 02:33:21 INFO  BlockManagerMasterEndpoint:54 - Registering block manager cssmpi7h.uwb.
2023-12-11 02:33:21 INFO  BlockManagerMaster:54 - Registered BlockManager BlockManagerId(driver,
2023-12-11 02:33:21 INFO  BlockManager:54 - Initialized BlockManager: BlockManagerId(driver, cssm
2023-12-11 02:33:21 INFO  ContextHandler:781 - Started o.s.j.s.ServletContextHandler@11653e3b{/me
2023-12-11 02:33:22 INFO  StandaloneSchedulerBackend:54 - SchedulerBackend is ready for schedulin
the grid size for the skyline objects is 512
Parallel Processing of Local Skyline Started
Parallely Transformed total tuples 8192
Ordering the Row Projections for each column
Filtering out highly dominated points by others Max Distance from X Axis
Running the MrGaskY Algorithm
Applying global Skyline Point Reduction on grid
Elasped Time: 57658
```

MASS:

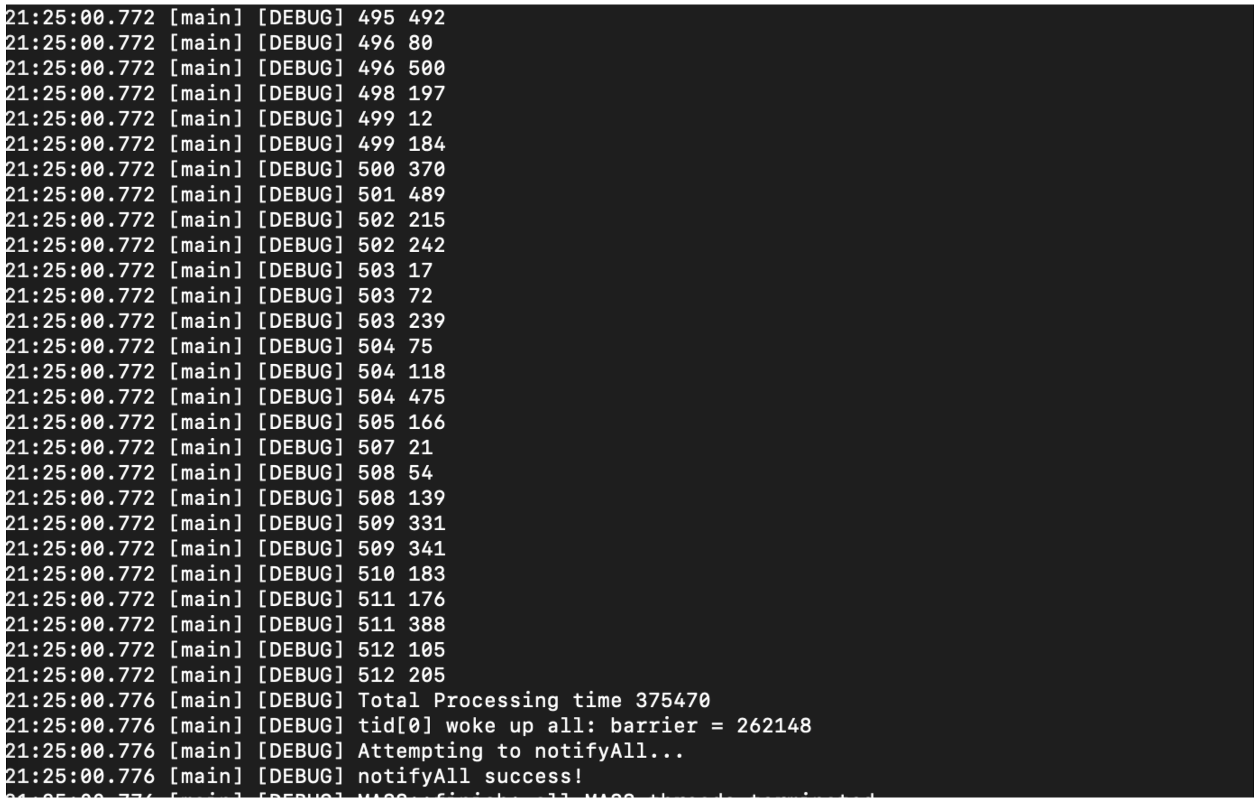
To execute the MASS program execute the following command:

the format for the args are <Facility-Count> GridX GridY FavCount UnFavCount

java -jar mass-skyline-1.0.0-SNAPSHOT.jar 16 512 512 8 8



```
21:25:00.765 [main] [DEBUG] Message received!
21:25:00.765 [main] [DEBUG] barrier received a message from cssmpi12h.uwb.edu...message = [Ljava.lang.Object;@4cf6264b
21:25:00.765 [main] [DEBUG] localAgents[4] = m.getAgentPopulation: -1
21:25:00.765 [main] [DEBUG] message deleted
21:25:00.765 [main] [DEBUG] barrierAllSlaves completed!
21:25:00.765 [main] [DEBUG] The Skyline objects are:
21:25:00.771 [main] [DEBUG] 2 303
21:25:00.771 [main] [DEBUG] 3 436
21:25:00.771 [main] [DEBUG] 4 347
21:25:00.771 [main] [DEBUG] 6 202
21:25:00.771 [main] [DEBUG] 7 264
21:25:00.771 [main] [DEBUG] 8 473
21:25:00.771 [main] [DEBUG] 9 116
21:25:00.771 [main] [DEBUG] 9 180
21:25:00.771 [main] [DEBUG] 9 489
21:25:00.771 [main] [DEBUG] 10 235
21:25:00.771 [main] [DEBUG] 10 298
21:25:00.771 [main] [DEBUG] 11 171
21:25:00.771 [main] [DEBUG] 11 291
21:25:00.771 [main] [DEBUG] 12 497
21:25:00.771 [main] [DEBUG] 13 334
21:25:00.771 [main] [DEBUG] 13 347
21:25:00.771 [main] [DEBUG] 15 94
21:25:00.771 [main] [DEBUG] 15 118
21:25:00.771 [main] [DEBUG] 15 132
21:25:00.771 [main] [DEBUG] 15 182
21:25:00.771 [main] [DEBUG] 15 379
21:25:00.771 [main] [DEBUG] 15 510
21:25:00.771 [main] [DEBUG] 16 34
21:25:00.771 [main] [DEBUG] 17 460
21:25:00.771 [main] [DEBUG] 18 240
21:25:00.771 [main] [DEBUG] 19 138
21:25:00.771 [main] [DEBUG] 19 467
21:25:00.771 [main] [DEBUG] 21 12
21:25:00.771 [main] [DEBUG] 21 506
21:25:00.771 [main] [DEBUG] 22 421
21:25:00.771 [main] [DEBUG] 24 179
21:25:00.771 [main] [DEBUG] 25 38
```



```
21:25:00.772 [main] [DEBUG] 495 492
21:25:00.772 [main] [DEBUG] 496 80
21:25:00.772 [main] [DEBUG] 496 500
21:25:00.772 [main] [DEBUG] 498 197
21:25:00.772 [main] [DEBUG] 499 12
21:25:00.772 [main] [DEBUG] 499 184
21:25:00.772 [main] [DEBUG] 500 370
21:25:00.772 [main] [DEBUG] 501 489
21:25:00.772 [main] [DEBUG] 502 215
21:25:00.772 [main] [DEBUG] 502 242
21:25:00.772 [main] [DEBUG] 503 17
21:25:00.772 [main] [DEBUG] 503 72
21:25:00.772 [main] [DEBUG] 503 239
21:25:00.772 [main] [DEBUG] 504 75
21:25:00.772 [main] [DEBUG] 504 118
21:25:00.772 [main] [DEBUG] 504 475
21:25:00.772 [main] [DEBUG] 505 166
21:25:00.772 [main] [DEBUG] 507 21
21:25:00.772 [main] [DEBUG] 508 54
21:25:00.772 [main] [DEBUG] 508 139
21:25:00.772 [main] [DEBUG] 509 331
21:25:00.772 [main] [DEBUG] 509 341
21:25:00.772 [main] [DEBUG] 510 183
21:25:00.772 [main] [DEBUG] 511 176
21:25:00.772 [main] [DEBUG] 511 388
21:25:00.772 [main] [DEBUG] 512 105
21:25:00.772 [main] [DEBUG] 512 205
21:25:00.776 [main] [DEBUG] Total Processing time 375470
21:25:00.776 [main] [DEBUG] tid[0] woke up all: barrier = 262148
21:25:00.776 [main] [DEBUG] Attempting to notifyAll...
21:25:00.776 [main] [DEBUG] notifyAll success!
```

Performance Analysis:

In order to analyze the performance of parallelization we categorized the benchmarking across all 5 implementations as

1. Spatial Capability/ Improvements:

All the output results for this with more benchmarking is attached in the sheet Output Results **Sheet 2.**

1. We keep the underlying hardware the same (Same node count, cpu cores and memory)

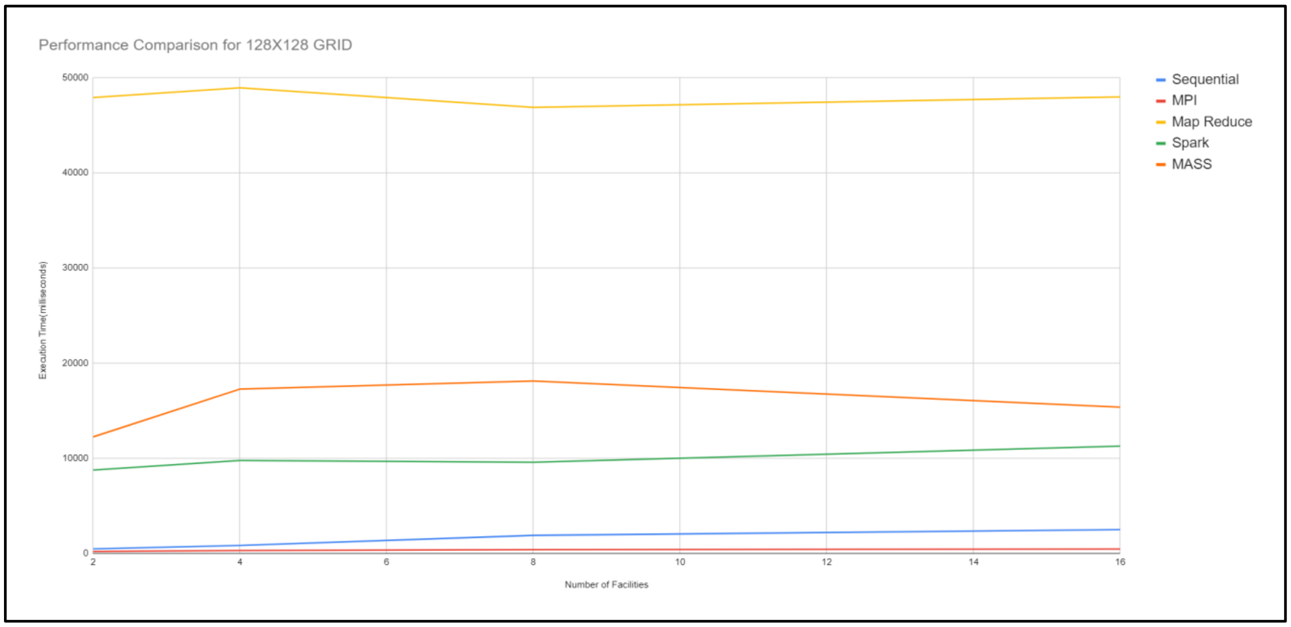


Figure 3. Grid Size 128 * 128

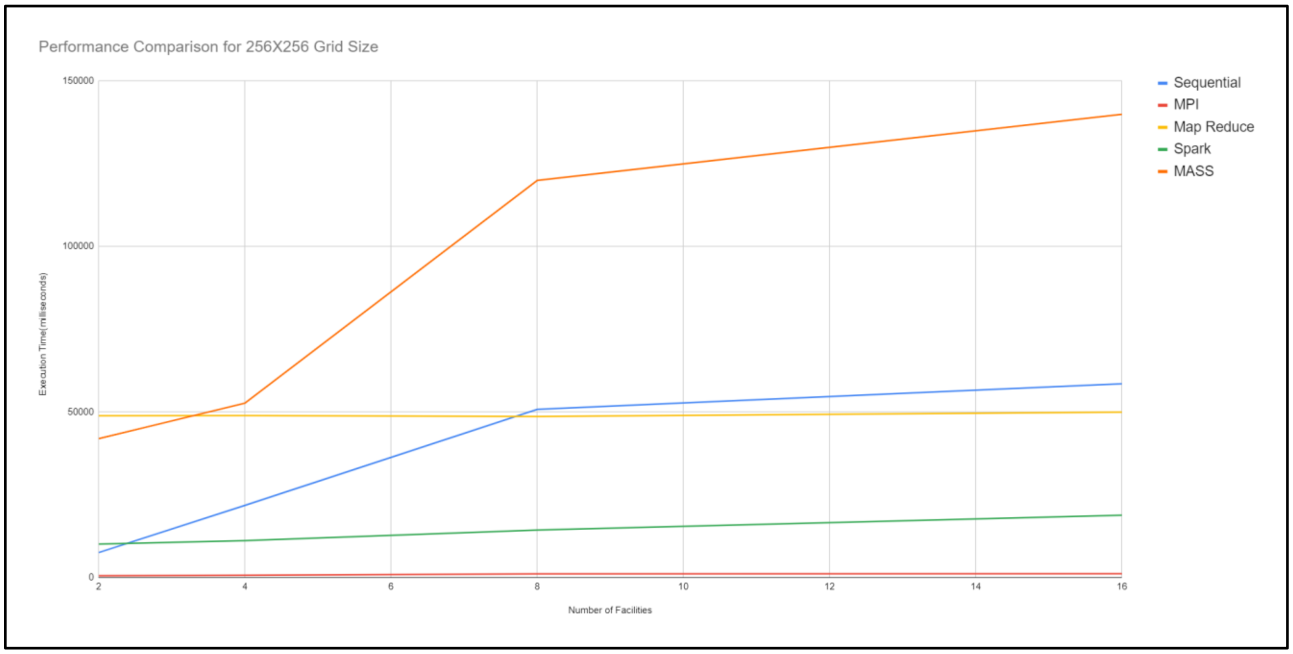


Figure 4. Grid Size 256 * 256

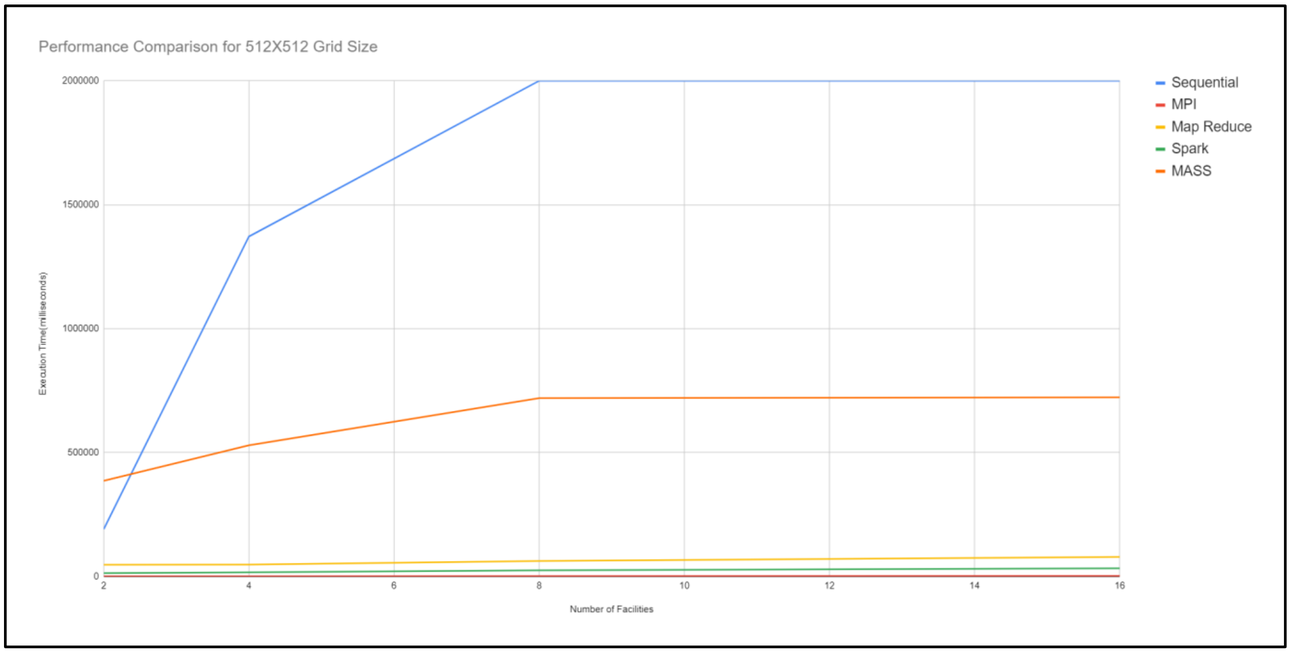


Figure 5. Grid Size 512 * 512

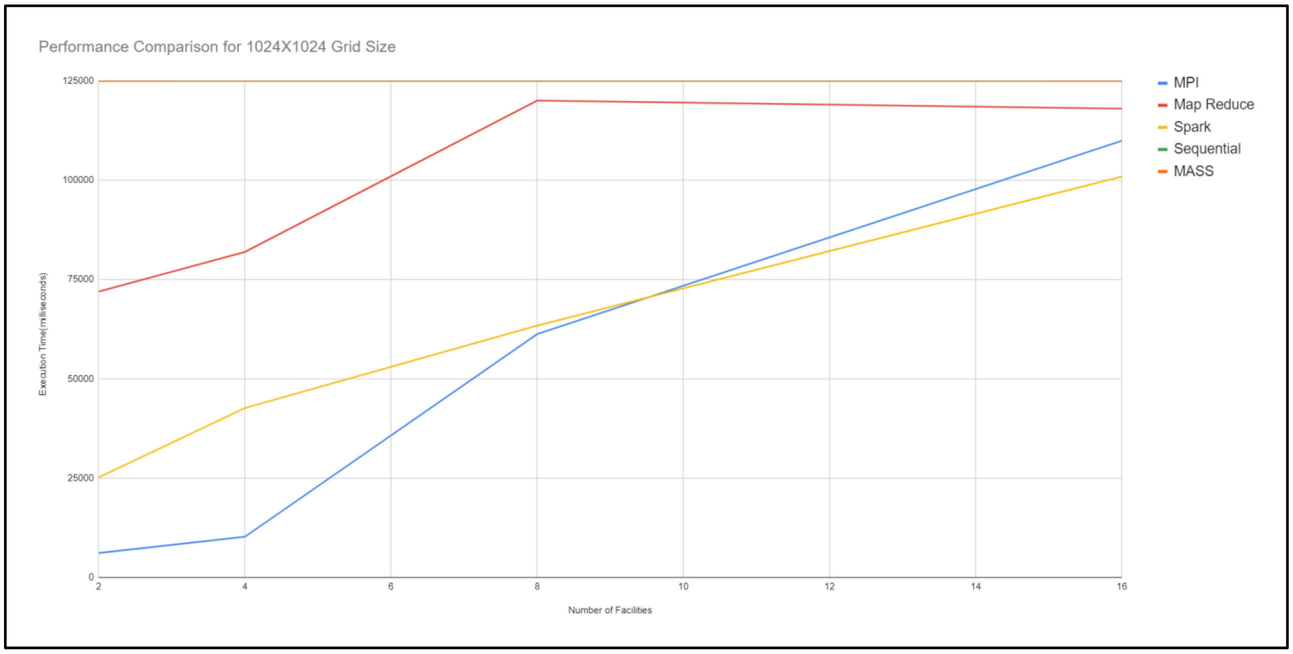


Figure 6. Grid Size 1024 * 1024

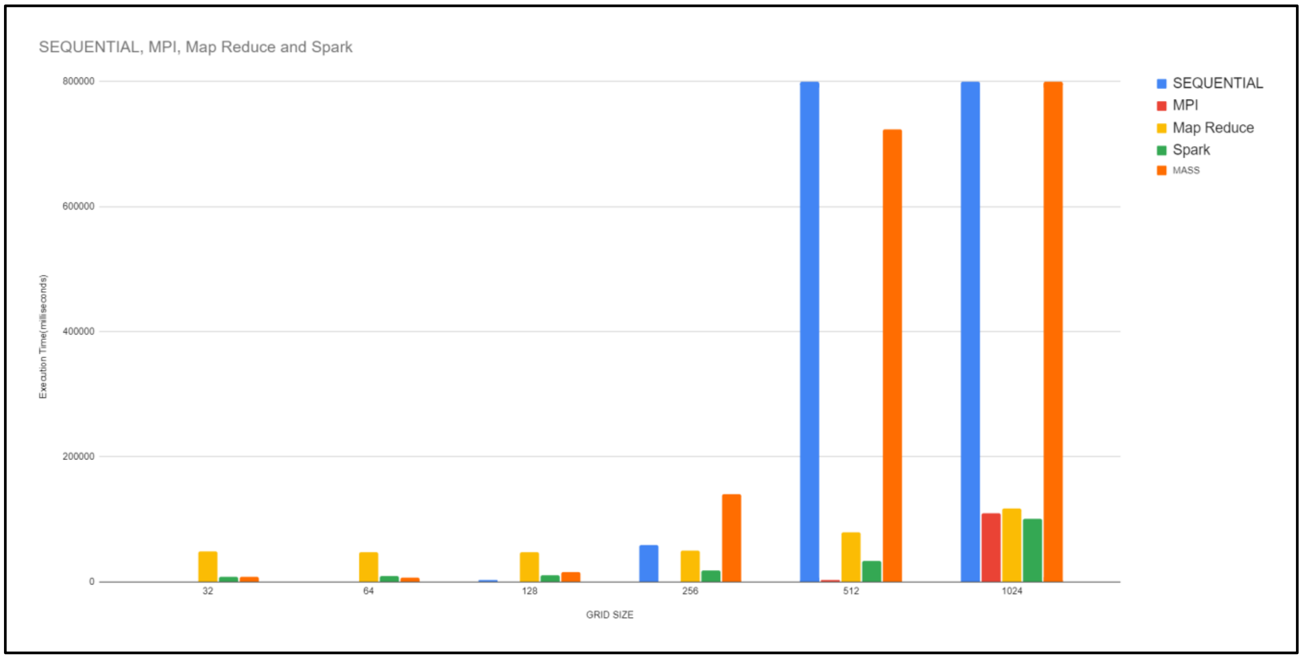


Figure 7. Bar Chart comparing performance for different implementations vs grid sizes.

From the graphs as mentioned, since the skyline query is an application of GIS, it is meant to solve problems with larger spatial regions or bigger grid sizes. Hence from our analysis and benchmarking we found the following results. Performance conclusion for each of the parallel implementations.

MPI: (Best consistent performance for smaller size grids)

1. We found MPI performed the best quite as fast as sequential for smaller size input / grid size. And the performance remained consistent for moderate size input but degraded for larger size due the memory overheads of storing large size grid arrays in the heap space of JVM for each rank.

Map Reduce: (Best consistent performance for larger size grids).

1. For Smaller size inputs, map reduce performance was worst due to presence of multiple jobs, and a lot of I/O operations eventually degrading the overall performance.
2. Map Reduce shows a constant performance with an increase in the input size, due to inherent support for HDFS and task parallelism.

MASS: (Best consistent performance for smaller size grids).

1. The reason for MASS being slow is due to us only using places and no agents.
2. Since as mentioned before in the parallelization strategy for mass the local skyline computation is task parallelism, where only one call to the places.callAll is enough for each place to compute the skyline objects internally.

3. However, for the global skyline computation is a real bottleneck because in this step we invoke places.callAll n * n times (size of the grid) because we need the distances at a given index across all the places to filter out the specific grid position.
4. Hence with an increase in the grid size the performance degrades, however for smaller size grid the performance is as fast as MPI, but much faster than spark and map reduce.

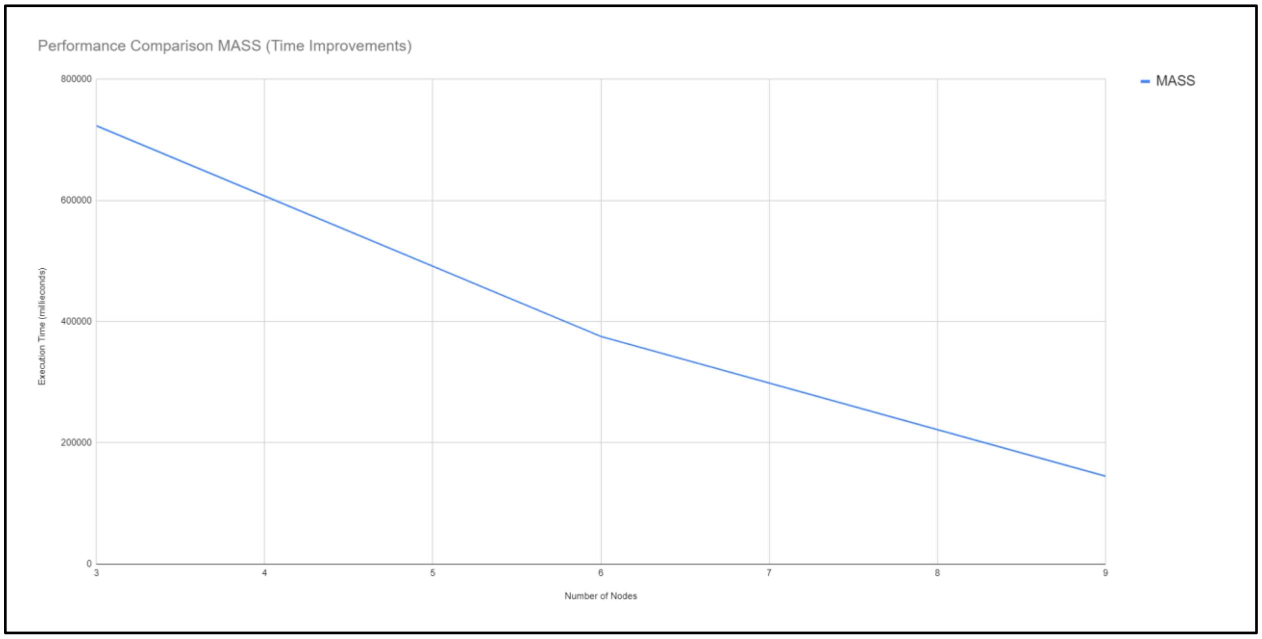
Spark: (Best performance for moderate size inputs).

1. Spark performance was relatively slower as compared to MPI and MASS due to the overhead of RDD partitioning and re-shuffling internally.
2. However for moderate size input between 64 to 128 512 grid size we found spark performance improved and removed consistent, but started degrading for larger size grid inputs. Due to larger size RDD there was more overhead to read it partition or re shuffle in between intermediate operations, resulting in a degradation of performance.

2. Time Improvements

1. We increase the resources to analyze the time improvements (more cpu cores/ nodes).

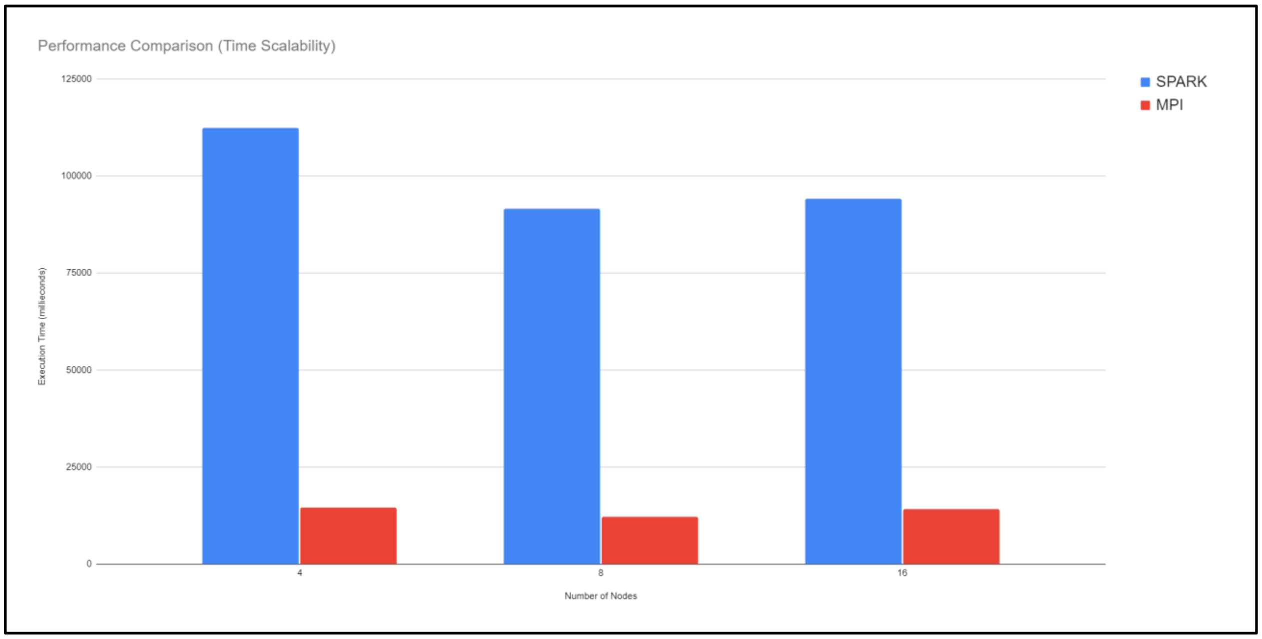
MASS:



Master	Worker	NODES	MASS	%TIME IMPROVEMENT
1	2	3	723002	1
1	5	6	375470	1.925591925
1	8	9	144943	4.988181561

MPI:/ Spark:

After allocating more cpu resources we found the performance improved, however we have kept the memory used the same across all while testing the result .



				% TIME IMPROVEMENT	% TIME IMPROVEMENT
Cluster Nodes	**Ranks / Cpu Cores**	**SPARK**	**MPI**	**MPI**	**Spark**
4	4	112403	14490	1	1
4	8	91571	12112	1.196334214	1.227495605
4	16	94191	14112	1.026785714	1.193351806

After increasing the computing nodes, we found significant improvements in the performance and execution time. In case of mass more cpu nodes provides higher cpu core count allowing MASS core to map more threads based on the cpu core count, thereby enhancing the parallel performance.

Programmability Analysis:

From our programmability analysis, we found that mapreduce had the highest amount of boiler plate code because of the more number classes and Java POJO's. MPI has highest Cyclomatic complexity due to MPIs complex communication logic and nested loops.

Detailed analysis of each implementation:

1. MPI:

There is not much boilerplate code available for MPI apart from MPI init and finalize. Boilerplate % is very low for MPI. But, due to complex parallelism and communication logic as well as the GASKY algorithm, cyclomatic complexity is high.
Major areas where complexity is high -

- Facility initialization where each rank reads the input from file, parses it.
- MRGASKY Algorithm in which we have more nested loops and conditional statements.
- Global skyline and min-max algorithm for favorable and unfavorable facilities is complex.

This Cyclomatic complexity can be reduced by refactoring the code and breaking down complex methods into smaller ones.

2. Map Reduce:

The most boilerplate code is present in the Job configuration, since our implementation uses 2 jobs, each job having its own internal configurations to set the associated mapper, reducer, combiner, partioner's and output formats.
There are multiple 16 classes available in Map reduce implementation. We have considered only reducers for cyclomatic complexity calculation as other classes have complexity negligible. Again, as mentioned in the MPI section, the MRGASKY algorithm is complex and contains nested loops and conditions which is making cyclomatic complexity of map reduce code as 52.

3. Spark:

Spark doesn't have much boilerplate code unlike mapreduce.

Since Spark is based out of Scala functional paradigm with using functional programming in Java, the boilerplate code contributed by the core Spark RDD transformations is quite less.
However, for each of these transformations, we require to run 2 main complex algorithms namely mrGaskyAlgorithm and filterGlobalDominantPoints which internally finds proximity intervals using voronoi projections and uses min-max algorithm.
These implementations have multiple nested loops and conditional statements which are making numerous paths through the code. Moreover, these loops also require some custom non primitive data types like Java objects, hence adding more java POJO's and associated getter setters which overall contribute to increasing the boilerplate percentage in the code.

4. MASS:
Mass implementation has a relatively less boilerplate code as compared to spark, and mpi, Since the only Code required to create a MASS application is MASS.Init() and MASS.finish() similar to MPI, however t The core library specific boiler code is far less in size.
However, for the application requirement of the Skyline grid, we created several java pojo's and associated getter setters to denote the actual entity while performing the algorithmic computation.
Along with algorithmic complexity, our MASS implementation has a callMethod with multiple switch statements which is significantly increasing the cyclomatic complexity.

	Classes	LOC	Boilerplate code LOC	Boilerplate %	Cyclomatic Complexity
Sequential	1	250	0	0	28
MPI	4	600	2	0.005%	99
Mapreduce	16	936	187	20.15%	52
Spark	1	520	22	4.23%	79
MASS	5	720	25	0.02	73