10 Parallel Computing Project paper , 5, MAY 2018

A Parallel Clustering Algorithm with MPI – MKmeans

Tulshi Chandra Das-BSSE0811, Maloy Kanti Sarkar-BSSE0834

Institute of information Technology, University of Dhaka

Email: bsse0811@iit.du.ac.bd,bsse0834@iit.du.ac.bd

***Abstract*— The simplex method is frequently the most efficient method of solving linear programming (LP) problems. We review previous attempts to parallelize the simplex method in relation to efficient serial simplex techniques and the nature of practical LP problems. For the major challenge of solving general large sparse LP problems, there has been no parallelization of the simplex method that offers significantly improved performance over a good serial implementation. However, there has been some success in developing parallel solvers for LPs that are dense or have particular structural properties.**

***Index Terms*— linear programming, simplex method, parallel computing**

1. INTRODUCTION

Simplex method is an algorithm which is used to solve the linear programming problems efficiently. We consider the computation of econometric estimators given by the optimization problem: minx z = 8x1 +9x2, where x1,x2 are the variables.

2x1 + 3x2 ≤ 50

2x1 + 6x2 ≤ 80

3x1 + 3x2 ≤ 70

Simplex method works on standard form of the equations. So, we have to trans-form the equations to standard form:

z = = 8x1 +9x2 + s1 + s2 + s3 2x1 + 3x2 + s1 = 50

2x1 + 6x2 + s2 = 80

3x1 + 3x2 + s3 = 70

Here s1,s2 and s3 are slack variable.

The Starting Tableau

Table1: Initial tableau of optimization

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Basic | z | X1 | X2 | S1 | S2 | S3 | sol |
| Z | 1 | 8 | 9 | 0 | 0 | 0 | 0 |
| S1 | 0 | 2 | 3 | 1 | 0 | 0 | 50 |
| S2 | 0 | 2 | 6 | 0 | 1 | 0 | 8 |
| S3 | 0 | 3 | 3 | 0 | 0 | 1 | 70 |

For optimization problems without closed form solutions, an iterative procedure is typically used. These algorithms successively evaluate the objective function z at different trial parameter vectors until a parameter vector achieves a convergence criterion. Let bi be the RHS of the ith row. Let aij be the coefficient of the entering variable xj in the ith row. The following “minimum ratio test” decides the leaving variable.

The optimization steps are:

1. Determining the pivot row and pivot element using aij and xj
2. Make a non-basic variable to basic variable
3. change the pivot row
4. Then change all other element in the table

The step 1 and 2 and 3 are serialized process and not costly. The step 4 is more costly and time consuming. We will parallelize the step 4.

There are two main sources of computation time in these numerical procedures. First, for each trial vector of parameters, there is the computational cost of evaluating the objective function at this vector of parameters. Where the objective function involves simulation or the solution of a complex behavioral model, such as a dynamic programming model, the objective function level computation costs can be large. A second source of computational costs is that for a high dimensional and continuous parameter space, a large number of vectors of parameters may need to be tried before the convergence criterion is achieved. This parameter level computation cost is increasing in the number of parameters, which is generally related to the complexity of the underlying behavioral model.

[Bsse0811@iit.du.ac.bd](mailto:Bsse0811@iit.du.ac.bd), bsse0834@iit.du.ac.bd

|  |  |
| --- | --- |
| Parallel Computing Project paper, 5, MAY 2018 | 11 |

1. Background

*A. Simplex algorithm*

Suppose, Objective function, maximise z = ax1 + bx2 Subject to constraints,

ax1 + bx2 ≤ c1

ax1 + bx2 ≤ c2

ax1 + bx2 ≤ c3

x1, x2 ≥ 0

Introducing slack variables, the LPP is same as, Objective function,maximise z = ax1 + bx2 + s1 +s2 + s3

ax1 + bx2 + s1 = c1

ax1 + bx2 + s2 = c2

ax1 + bx2 + s3 = c3

x1, x2, s1, s2, s3 ≥ 0

Basic Feasible Solution

Let:

* x1,x2 as non basic variable
* Then BFS can be got immediately as (0,0,c1,c2,c3)

This is possible because in the LHS of the constraint equations (\*) the coefficients of the basic variables s1, s2, s3 form an Identity matrix (of size 3 x 3).

**Closer Look (1):**

We note that the z-row is the objective function row. The remaining 3 rows are the basic variable rows. Each row corresponds to a basic variable.

* the leftmost variable denotes the basic variable corresponding to thatrow
* the coefficients of the basic variables are zero.
* In each column corresponding to a basic variable basic variable has a non-zero coefficient, namely 1
* all the other basic variables have zero coefficients.
* the objective function, z, has value zero.
* the leftmost variable denotes the basic variable corresponding to thatrow.
* We now seek to make one of the nonbasic variables as basic (and so) one of the basic variableswill become nonbasic (that is will drop down to zero)
* the nonbasic variable that will become basic is chosen such that theobjective function will “improve”:

The nonbasic variable that will become basic is referred to as “entering” vari-able and the basic variable that will become nonbasic is referred to as “leav-ing” variable.

**Criterion for “leaving” variable (Feasibilty Condition)**

Let while bi be the RHS of the ith row. Let ai j be the coefficient of the entering variable xj in the ith row. The following “minimum ratio test” decides the leaving variable: Choose xk as the leaving variable where k is given as that row index i for which the ratio {bi ÷ ai j : ai j ≥ 0} is least (break the ties arbitrary)

Pivot element:

* the entering variable column is called the pivot column
* The leaving variable row is called the pivot row
* The coefficient in the intersection of the two is referred to as the pivot element.

Operations: We apply elementary row operations to modify the simplex tableau so that the pivot column has 1 at the pivot element and zero in all other places.

The elementary Row operations are as follows:

* The fuzzy C-means clustering is a soft version of K-means, where each point has a degree of belonging to cluster, as in fuzzy logic.
* The New pivot row = old pivot row pivot element
* Any other new row = corresponding old row – (old coefficient of the entering variable in that row \* New pivot row)
* We shall also change the legend of the new pivot row only as the entering variable.

The last tableau is the optimal tableau as all entries in the objective function row

are ≥ 0 and the LPP is a maximization problem.

*B. The Concept of MPI*

Most popular high-performance parallel architectures used in the parallel programming environment are divided into two classes: message passing and shared storage. The cost of message passing parallel processing is larger, suitable for large-grain process-level parallel computing. Compared with other parallel programming environment, message passing has good portability, supported by almost all parallel environments. Meanwhile, it has good scalability and complete asynchronous communication function, which can well decompose tasks according to the requirements of users, organize data exchange between different processes and is applied to scalable parallel algorithms.

MPI is an interface mode widely used in various parallel clusters and network environments based on a variety of reliable message passing libraries.

MPI is a message passing parallel programming standard used to build highly reliable, scalable and flexible distributed applications, such as workflow, network management, communication services.

MPI is a language-independent communications protocol. FORTRAN, C and C++ can directly call the API library. The goals of MPI are high performance, scalability and portability.

MPI is a library specification for message passing, not a language. Message Passing Interface is a standard developed by the Message Passing Interface Forum (MPIF). MPI is a standard library specification designed to support parallel computing in a distributed memory environment. The first version (MPI-1) was published in 1994 and the second version (MPI-2) was published in 1997 [16]. Both point-to-point and collective communication are supported.

[Bsse0811@iit.du.ac.bd](mailto:Bsse0811@iit.du.ac.bd), bsse0834@iit.du.ac.bd

12 Parallel Computing Project paper 5, MAY 2018

MPICH is an available and portable implementation of MPI, a standard for message passing used in parallel computing. MPI has become the most popular message passing standard for parallel programming. There are several MPI implementations among which MPICH is the most popular one. MPICH1 is the original implementation of MPICH that implements the MPI-1standard. MPICH2 is a high-performance and widely portable implementation of the MPI standard (both MPI-1 and MPI-2). The goals of MPICH2 are to provide an implementation of MPI that efficiently supports different computation and communication platforms including commodity clusters, high-speed networks and proprietary high-end computing systems.

The standards of MPI are as follows [17]:

* Point-to-point communication
  + Collective operations Process groups.
  + Communication contexts
  + Process topologies
  + Bindings for FORTRAN 77 and C
  + Environmental management and inquiry
  + Profiling interface

*C. MPI Functions*

MPI is a library with hundreds of function-calling interfaces, and FORTRAN, C language and C++ can directly call these functions. Many parallel programs can be written with just six basic functions, almost complete all of the communication functions.

Table I illustrates the basic functions. MPI\_Init() initializes the MPI environment and assigns all spawned processes; MPI\_Finalize() terminates the MPI environment; MPI\_Comm\_size() finds the number of processes in a communication group; MPI\_Comm\_rank() gives the identification number of a process in a communication group; MPI\_Send() sends message to the destination process of rank *dest* and MPI\_Recv() receives message from the specified process of rank *source*

.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | TABLE I | |  |
|  |  | MPI BASIC FUNCTIONS | | |  |
|  |  |  |  |  |  |
|  | ***Function*** |  | ***Functionalities*** |  |  |
|  | MPI\_Init |  | Initialization |  |  |
|  | MPI\_Finalize |  | Termination |  |  |
|  | MPI\_Comm\_size |  | Access to the number of processes |  |  |
|  | MPI\_Send |  | Send |  |  |
|  | MPI\_Recv |  | Receive |  |  |
|  | MPI\_Comm\_rank |  | Access to the identification number | |  |
|  |  | of a process | |  |
|  |  |  |  |

*D. The Messing Passing Process of MPI*

MPI is a parallel programming standard based on message passing, whose function is to exchange information, coordinate and control the implementation steps with the definition of program grammar and semantics in the core library by sending messages between the concurrent execution parts.

First all of the MPI programs contain “mpi.h” header file, and then complete the initialization of the program by MPI\_Init(), after that, establish process topology structure and new communicator and call the functions and applications to be used for each process, finally use MPI\_Finalize() to terminate each process.

The parallel program design flow of message passing process is shown in Fig. 1.

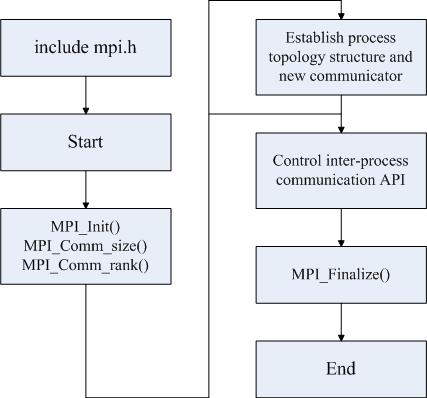


Figure 1. Flow of message passing process

1. Related Work

Several research paper were published on parallel simplex method. Each of those showed a positive effect of parallel implementation of simplex method. To prepare my paper I have taken help of those paper published previous. The paper from which some knowledge are taken are referred at reference part. Each of these research papers showed a slow motion of performance for small dataset and less clustering node. In my project this action is also noticed.

IV. Methodology

In this section, we will propose an implementation of a parallel Simplex method based on MPI. Before introducing our Simplex algorithm, we first give a short description of the Sequential Simplex algorithm in Section *3.1.*

[Bsse0811@iit.du.ac.bd](mailto:Bsse0811@iit.du.ac.bd), bsse0834@iit.du.ac.bd

|  |  |
| --- | --- |
| Parallel Computing Project paper 5, MAY 2018 | 13 |

*A. Serial Simplex Method*

In general, simplex method is implemented with serial code. Each step of update table or data is sequential and straightforward. The steps are given below:

**Step 0: (Initialization):**

Start with a feasible basic solution and construct the corresponding simplex tableau.

**Step 1: (Choice of entering variable):**

If a0j ≥ 0, j=1, 2,.…,*n*, STOP. The current solution is optimal. Otherwise, choose the entering variable using the following pivoting rule: a0s = min {a0j : a0j < 0, j = 1,2, …, n}

**Step 2: (Choice of leaving variable):**

Let I = {i : ais> 0}. If I=∅, STOP. The problem (LP.1) is unbounded. Otherwise, choose the leaving variable using the minimum ratio test:

**Step 3: (Pivoting):**

The pivoting element is the variable ars. Construct the next simplex tableau as follows:

Let, arj

And

aij , j=1,2,….n,n+1

Go to step1.

The simplex algorithm uses the Gauss-Jordan transformation of the tableau to move from one basic feasible solution to another. Each iteration of the simplex algorithm is

relatively expensive. This can be seen by examining the previous formal description of the simplex algorithm. More precisely, the number of multiplications and additions at each iteration is approximately equal to *m*(*m-n*)*+n+1* and *m*(*nm+1*) respectively, where *m* is the number of constraints and *n* is the number of variables.

This happens whenever *n* and *m* are large, in which case nearly 100% of the cpu-time is spent in Step 3 (Pivoting). In this third Step, a multiple of row *r* is added to row I, (this is the only double nested loop executed at each iteration).[3].

*B. Parallel Simplex Method with MPI*

We implemented the simplex method using a straightforward application of the C language tools and the Message Passing Interface (MPI). The storage of the simplex tableau was carried out using a (0:*m*)×(0:*m*+*n*) dimensioned array. Also, in order to allocate the work across multiple processors, MPI was used. The coefficients of the objective function are represented using the *q* vector, the number of processors is denoted by NPRS, the current tableau is denoted by TABL and finally the rank of processor is denoted by ITSK.[4]

**Parallel Simplex Algorithm Begin**

**1-** for 0 ≤ *i* < *m*+ *n* do

In processor 0: Set *q[i]:=TABL[0][i]*

for 0 ≤ *k* < *NPRS* pardo

{for 0 ≤ *i* < *m*/ *NPRS* do

for 0 ≤ *j* < *m*+ *n* do

Set *C[i][j]:=TABL[k\*m/ NPRS+i]*

for 0 ≤ *j* < *m*/ *NPRS* do

Set *b[j] := TABL[j][M+N]*

}

**2**- In processor 0

for 0 ≤ *i* < *n* do

search *i* ( where *q[i]<0* ) Set *column := i*

if failure Goto (10)

**3-** for 0 ≤ *k* < *NPRS* pardo

*X:=min(b[j]/C[j][column],* 0 ≤ *j* < *m*/ *NPRS )*

*Set row\_no:=j*

**4-** for 0 ≤ *k* < *NPRS* pardo

Send (*X,ITSK*) to processor 0

In processor 0

Search *ITSK* which corresponds to *min (X)*

*min\_L:=ITSK*

**5-** From processor *min\_L*

Send row *g:=C[row\_no][:]* to all processor

Send variable *h:=b[[row\_no]* to all processor

**6-** for 0 ≤ *k* < *NPRS* pardo

for 0 ≤ *i* < *m*/ *NPRS* do{

Set *a:=C[i][column]/g[column]*

for 0 ≤ *j* < *m*+ *n* do

Set *C[i][j]:=C[i][j]-a\*g[j]*

Set *b[i]:=b[i]-a\*h*

}

**7-** In processor *min\_L*

for 0 ≤ *j* < *m*+ *n* do{

set *C[row\_no][j] := g[j]/g[column]*

set *b[row\_no] := h / g[column]*

}

**8-** In processor 0

*a:=q[column]/g[column]*

For 0 ≤ *i* < *m*+ *n* do {

Set q[i] := q[i]-a\*g[i]

Set q[n+m] := q[m+n] - a\*h

}

**9-** Goto (2)

**10- F**or 0 ≤ *i* < *m*+ *n* do

In processor 0

Set *TABL[0][i] := q[i]*

for 0 ≤ *k* < *NPRS* pardo{

for 0 ≤ *i* < *m*/ *NPRS* do

For 0 ≤ *j* < *m* + *n* do

Set *TABL[k\*m/NPRS+i][j] := C[i][j]*

for 0 ≤ *j* < *n* / *NPRS* do

Set *TABL[j][m+n] := b[j]*

}

**End.**

[Bsse0811@iit.du.ac.bd](mailto:Bsse0811@iit.du.ac.bd), bsse0834@iit.du.ac.bd

14 Parallel Computing Project paper 5, MAY 2018

V. EXPERIMENTS

In this section, we first give the experimental environment in Section *A*. Second, we give the description of experimental data sets in Section *B*

*A. Experimental Environment***:**

The hardware platform in this paper uses four PC. The software environment uses the following configuration.

**1. Ubuntu 14.04 LTS**:

Memory: 3.8 GiB

Processor: Intel ® CoreTM i5-3470 CPU @ 3.20GHz × 4

OS type: 64-bit

Disk: 107.1 GB

Graphics: Intel ® Ivybridge Desktop

**2. Ubuntu 16.04 LTS**:

Memory: 3.7 GiB

Processor: Intel ® CoreTM i3 CPU 540@ 3.07 GHz × 4

OS type: 64-bit

Disk: 44.2 GB

Graphics: Intel ® Ironlake Desktop

**3. Ubuntu 16.04 LTS**:

Memory: 3.8 GiB

Processor: Intel ® CoreTM i5-3470 CPU @ 2.20GHz × 4

OS type: 64-bit

Disk: 51.5 GB

Graphics: Intel ® Ivybridge Desktop

**4. Ubuntu 14.04 LTS**:

Memory: 3.8 GiB

Processor: Intel ® CoreTM i3-3470 CPU @ 2.20GHz × 4

OS type: 64-bit

Disk: 51.5 GB

Graphics: Intel ® Ivybridge Desktop

In terms of aforementioned platform, sublime text 3 is used to develop procedures. Considering the fairness of comparison, the configuration of MPI parallel development platform is based on open resource project and the experimental platform has a gcc complier based on MinGW (Minimalist GNU for Ubuntu).

*B. Data Sets*

Experimental data sets are selected from the github.com Dataset Repository.

Table 3: Description of Dataset

|  |  |  |  |
| --- | --- | --- | --- |
| Code number of Data Set | Name of Data Set | Size(KB) | Number of instance |
| 1 | afiro | 4 | 28 |
| 2 | adlittle | 17 | 57 |
| 3 | agg | 98 | 489 |

Table 4: Comparison results within processes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset  No. | Execution Time(microsecond) | | | | |
| Serial | Clustered with 2 node(computer) | | | |
| process | | | |
| 4 | 8 | 12 | 16 |
| 1 | 1593 | 530696 | 569165 | 10018335 | 9687363 |
| 2 | 83822 | 10432230 | 11364384 | Waiting state | Waiting state |
| 3 | 1060958 | 8809797 | 9699020 | Waiting state | Waiting state |

Table 5: Comparison results within processes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dataset  No. | Execution Time(microsecond) | | | |
| Clustered with 3 node(computer) | | | |
| process | | | |
| 4 | 6 | 8 | 12 |
| 1 | 807983 | 870251 | 783543 | 1824435 |
| 2 | 16918492 | 17174850 | 15764538 | 38181190 |
| 3 | 12692963 | 13821337 | 12305146 | 25850273 |

Table 6: Comparison results within processes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset  No. | Execution Time(microsecond) | | | | |
| Clustered with 4 node(computer) | | | | |
| process | | | | |
| 4 | 8 | 12 | 16 | 20 |
| 1 | 1150373 | 1076964 | 1675929 | 1144053 | 10053268 |
| 2 | 19987390 | 19157692 | 32246914 | 20105473 | Waiting |
| 3 | 15608005 | 14547605 | 22298906 | 15017849 | waiting |

[Bsse0811@iit.du.ac.bd](mailto:Bsse0811@iit.du.ac.bd), bsse0834@iit.du.ac.bd

|  |  |
| --- | --- |
| Parallel Computing Project paper 5, MAY 2018 | 15 |

In the table3 we show results of processes with cluster of 2 pc and the serialized process. In table4 we show the experiment result in cluster of 3 pc and the table5 show the result of experiment clustering of 4 pc. We have tested the result with several processes. We have calculated the speed up factor of each process using the rule:

Speed up =

Table 7: Comparison of speed of processes corresponding to serial process

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset  No. | Speed up | | | | |
| Clustered with 2 node(computer) | | | |
| process | | | |
| 4 | 8 | 12 | 16 |
| 1 | 0.003 | 0.0027 | 0.00018 | 0.00016 |
| 2 | 0.008 | 0.0007 | Waiting state | Waiting state |
| 3 | 0.12 | 0.1 | Waiting state | Waiting state |

Table 8: Comparison of speed of processes corresponding to serial process

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dataset  No. | Speed up in 3 node cluster | | | |
| Clustered with 3 node(computer) | | | |
| process | | | |
| 4 | 6 | 8 | 12 |
| 1 | 0.02 | 0.018 | 0.02 | 0.008 |
| 2 | 0.004 | 0.004 | 0.005 | 0.002 |
| 3 | 0.08 | 0.07 | 0.086 | 0.04 |

Table 9: Comparison of speed of processes corresponding to serial process

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dataset  No. | Speed up | | | | |
| Clustered with 4 node(computer) | | | | |
| process | | | | |
| 4 | 8 | 12 | 16 | 20 |
| 1 | 0.001 | 0.001 | 0.009 | 0.001 | 0.0001 |
| 2 | 0.004 | 0.004 | 0.002 | 0.004 | Waiting |
| 3 | 0.06 | 0.07 | 0.047 | 0.07 | waiting |

We find that in each process, speed factor is influenced by data set, number of process, communication overhead and number of clustering node. We find the increase of speed up according to size of data set. Speed up also increases according to clustering nodes. For small dataset number of clustering node don’t contribute to speed up enough. We see for larger dataset clustering affect a positive sense to speed up. If the number of process increases upto a bound, processes tend to enter in a waiting state. If our dataset is larger and the number of clustering is enough then we can achieve a better speed up

Figure1: Comparison of speed up in different dataset with processes

[Bsse0811@iit.du.ac.bd](mailto:Bsse0811@iit.du.ac.bd), bsse0834@iit.du.ac.bd

16 Parallel Computing Project paper 5, MAY 2018

Figure2: Comparison of speed up between clustering nodes with 12 processes

VI. DISCUSSION

From the performance analysis figures given in experiment and analysis part, we see that speed up is influenced by process number and clustering node and dataset size. The more dataset is large enough the more speed is acquired by the clustering. The execution time of serial code is for dataset1 is 1593 microsecond. For 2 nodes the execution time of MPI program is 530696 for 4 process. So, we see that there is no performance gain. Speed up increases for larger data set respective to clustering node. But if number of processes is not enough respective to clustering node performance don’t increase enough. In figure 2, see the most speed up for 16 processes in cluster of 4 nodes in dataset 3. In dataset1 and dataset2 clustering doesn’t affect enough. In some cases, program enter into waiting state. It happens when there are more processes than the clustering nodes. In this case, some process can’t get process to run, so it get in waiting state.

VII. CONCLUSION

We proposed a parallel Simplex method based on MPI in this paper. Meanwhile, the configuration of MPI parallel development platform based on open resource MPICH in Ubuntu is implemented, whose ideas and methods can be ported to Windows or other platforms. Experimental results show that the parallel implementation of the Simplex method is efficient in the clustering on large data sets.

ACKNOLEDGMENT

At first, we thank almighty God to help us to complete the project. We wish to thank to our course teacher Dr. B M Mainul Hoss.ain, Associate Professor, Institute of Information Technology, University of Dhaka for his valuable direction and help to complete our paper.

REFERENCES

1. B. Hayes, “Cloud computing,” *Communications of the* *ACM*, vol. 51, no. 7, pp. 9-11, July 2008.
2. E. Lusk, N. Doss, A. Skjellum, “A High-Performance, Portable Implementation of the MPI Message Passing Interface,” *Parallel Computing*, vol. 22, pp. 789-828, 1996.
3. B. Hayes, “Cloud computing,” *Communications of the* *ACM*, vol. 51, no. 7, pp. 9-11, July 2008.
4. World Academy of Science, Engineering and Technology 23 2006
5. S. Kantabutra, A. Couch, “Parallel K-means Clustering Algorithm on NOWs,” *Technical Journal*, vol. 1, no. 6, 2000.
6. A. Jain, R. Dubes, “Algorithms for Clustering Data,” *Prentice Hall*, 1988.
7. D. Arthur, S. Vassilvitskii, “k-means++: the advantages of careful seeding,” *Proceedings of the eighteenth annual* *ACM-SIAM symposium on Discrete algorithms*, pp. 1027-1035, 2007.
8. G. B. Dantzig, Linear Programming and Extensions. NJ: Princeton,
9. K. H. Borgwardt, “Some distribution independent results about the
10. I. Maros, and G. Mitra, “Investigating the sparse simplex method on a distributed memory multiprocessor”, Parallel Computing, vol. 26, pp. 151-170, 2000.method”, Mathematics of Operations Research, vol. 7, no. 3, pp. 441-462, 1982..
11. G. Frahling, C. Sohler, “A Fast K-means Implementation Using Coresets,” *Proceedings of the Twenty-second* *Annual Symposium on Computational Geometry*, pp. 135-143, 2006.
12. D. Klabjan, L. E. Johnson, and L. G. Nemhauser, “A parallel primal-dual simplex algorithm”, Operations Research Letters, vol. 27, no. 2, pp. 47-55, 2000.
13. J. Eckstein, I. Bodurglu, L. Polymenakos, and D. Goldfarb, “Data-Parallel Implementations of Dense Simplex Methods on the Connection Machine CM-2”, ORSA Journal on Computing, vol. 7, no. 4, pp. 402-416, 1995.
14. S. P. Bradley, U. M. Fayyad, and O. L. Mangasarian, “Mathematical Programming for Data Mining: Formulations and Challenges”, INFORMS Journal on Computing, vol. 11, no. 3, pp. 217-238, 1999.
15. I. Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley, 1995.
16. *W. Gropp, E. Lusk, and A. Skjellum, Using MPI: Portable Parallel Programming with the Message Passing- Interface. MIT Press, 1994.*

[Bsse0811@iit.du.ac.bd](mailto:Bsse0811@iit.du.ac.bd), bsse0834@iit.du.ac.bd