A parallel implementation of an exterior point algorithm for linear programming problems

N. Ploskas[1], N. Samaras[1] and A. Sifaleras[2]
[1] University of Macedonia/Applied Informatics, Thessaloniki, Greece
[2] University of Macedonia/Technology Management, Naousa, Greece

Abstract—**The simplex method is perhaps the most widely used method for solving linear programming (LP) problems. The computation time of simplex type algorithms depends on the basis inverse that occurs in each iteration. Parallelizing simplex type algorithms is one of the most challenging problems. The aim of this paper is to present a parallel implementation of the primal exterior point simplex algorithm. In this approach the basis inverse is computed in parallel. The matrix that holds the basis is distributed among different workers and the computation is performed faster in large-scale LP problems. Apart from the parallel implementation, this paper presents a computational study that shows the speedup among the serial and parallel version in large-scale randomly generated full dense LP problems.**

Index Terms—**Computational Study, Exterior Point Simplex type Algorithm, Linear Programming, Parallel Programming.**

I. INTRODUCTION

Linear Programming (LP) is a significant area in the field of mathematical optimization, where a linear function (1) is optimized.

$$z = \sum_{i=1}^{n} c_i x_i \tag{1}$$

Several methods are available for solving LP problems, among which the simplex algorithm [9] is the most widely used due to its simplicity and speed. Although it is well known that the computational complexity of the simplex method is not polynomial in the number of equations, in practice it can quickly solve many LP problems. Applications of the simplex method can be found in various areas, such as operations research and nonlinear constrained structural optimization. We assume that the problem is in its general form. Formulating the linear problem in mathematical terms now, we can describe it as shown below:

$$\begin{array}{ll} \min & c^T x \\ \textit{subject to} & Ax = b \\ & x \geq 0 \end{array} \tag{2}$$

where $A \in R^{m\times n}$, $(c, x) \in R^n$, $b \in R^m$, and T denotes transposition. We assume that A has full rank. The simplex method searches for an optimal solution by moving from one feasible solution to another, along the edges of the feasible set.

Exterior Point Simplex Algorithm (EPSA) differs radically from Primal Simplex Algorithm (PSA) because its basic solutions are not feasible [26] and [28]. EPSA relies on the idea [27] that making steps in directions that are linear combinations of attractive descent directions can lead to faster practical convergence than that achievable by PSA. For all known pivoting rules [24, 30] sequences of examples have been constructed, such that the number of iterations is exponential in m+n. None of the existing simplex type algorithms admits polynomial complexity.

As in the solution of any large scale mathematical system, the computational time for large LP problems is a major concern. The requirement for faster and more efficient computations in scientific applications has been considerably increased in the last years. Natural restrictions and high costs render impossible the increase of speed of processors beyond concrete limits. In order to overcome these difficulties new architectures have been developed importing the parallel processing [3], [7], [12], [33]. Nowadays the clustering of many processors has led to the development of various types of high-performance machines.

Networked computers have become a common infrastructure in most organisations, especially large ones. Additionally, the speed of the network has also improved significantly. Nowadays, it is very common for local area networks to transfer multiple types of data such as voice and video. High performance computing has also benefited and today, distributed computing is quickly gaining popularity [23].

Parallel programming is a good practice for solving computationally intensive problems in various fields. In operations research, for instance, solving maximization problems with simplex method is an area where parallel algorithms are being developed [19], [20], [22], [32]. Parallelizing simplex type algorithms is one of the most challenging problems. Due to very dense matrices and very heavy communication, the ratio of computation to communication is extremely low. It becomes necessary to carefully select parallel techniques, partitioning patterns and communication optimization in order to achieve a speedup. A popular approach for the implementation of parallel algorithms is to configure a cluster or a network of personal computers. With the advances made in computer hardware and software, it is now quite a simple matter to configure a computer network.

The primary reasons for using parallel computing are to:

(i). reduce execution time

(ii). solve larger problems

(iii). provide concurrency

(iv). take advantage of non-local resources - using available computer resources on a wide area network and finally

(v). overcome memory constraints

The application of parallel processing to the simplex method for linear programming has been considered since the early 1970's. However, only since the beginning of the 1980's attempts have been made to develop implementations. Although few experiments have been made, using shared memory machines, the vast majority of implementations used distributed memory multiprocessors and Ethernet-connected clusters.

One of the earliest parallel tableau simplex methods on a small-scale distributed memory Multiple-Instruction Multiple-Data (MIMD) machines is by Finkel [11]. His study showed that the overhead for distributing matrix elements for pivot operations, and for inter-process synchronization did not allow for any significant speedup. Wu and Lewis [36] presented two parallelization of the revised simplex algorithm with explicit form of the basis inverse on a shared memory MIMD machine. Their implementations presented good scalability only on very small problem sizes. Stunkel [34] studied the performance of the tableau and the revised simplex on the iPSC/2 hypercube computer. Helgason et al. [17] proposed an algorithm to implement the revised simplex using sparse matrix methods on shared memory MIMD computer. Furthermore, Shu and Wu [32] and Shu [31] parallelized the explicit inverse and the LU decomposition of the basis simplex algorithms. Both methods are very efficient, with the first being more suitable to LP problems with dense constrained matrices, while the second being more suitable to LP problems with sparse matrices. The LU decomposition of the basis attained some speedup only in large and sparse problems.

Simplex algorithms for general LP problems on Single Instruction Multiple Data (SIMD) have been reported by Agarwal et al. [1], and by Eckstein et al. [10]. The implementation in [1] had a disappointing performance. Eckstein et al. in [10] implemented a parallelization of dense simple and interior-point algorithms in a CM2 machine. They reported that interior point algorithms are relatively easy to implement on SIMD machines with commercially available library software.

Hall and McKinnon [16] worked on parallel revised methods and obtained a speedup of between 2.5 and 4.8. Thomadakis and Liu [35] worked on the standard method utilizing the MP-1 and MP-2 MasPar. Yarmish [37] describes a coarse grained distributed simplex method, dpLP, that efficiently solved all LP problems in the Netlib repository [14]. Cvetanovic et al. [8] report a speedup of twelve when solving two small size problems with standard simplex. Recently, Badr et al. [2] presented results for an implementation on eight computers, achieving a speedup of five when solving small random dense LP problems. Lentini et al. [21] developed a parallel implementation of the standard simplex method with the tableau stored as a sparse matrix. When solving medium sized Netlib problems on four transputers they achieved a speedup of between 0.5 and 2.7, with a super-linear speedup of 5.2 on one problem with a relatively large column-row ratio. Finally, in [4, 6, 29] computational results for parallelizing the network simplex method are reported.

This paper presents a parallelization of the exterior-point simplex algorithm on a multicore machine. The focus of this parallelization is on the basis inverse. An outline of the rest of the paper is as follows. In Section II, the exterior-point simplex algorithm is described and presented. In Section III, the parallel matrix multiplication technique which was used to compute the basis inverse is introduced. Section IV presents the parallel exterior-point simplex algorithm and section V gives the computational results. Finally, the conclusions of this paper are outlined in section VI.

II. EXTERIOR-POINT SIMPLEX ALGORITHM

The algorithm starts with a primal feasible basic partition (B, N). The variables corresponding to B will be called basic. The others will be referred to as non-basic. Later on, the following sets of indexes are computed:

$$P = \{ j \in N : s_j < 0 \} . \tag{3}$$

$$z = \{ j \in N : s_j \geq 0 \} . \tag{4}$$

If P = ∅ then the current basis B and the corresponding solution $x^T = (x_B, x_N)$ is optimal for the primal problem. EPSA firstly defines the leaving and afterwards the entering variable. The leaving variable $x_{B[r]} = x_k$ is computed as follows:

$$a = \frac{x_{B[r]}}{-d_{B[r]}} = \min\left\{ \frac{x_{B[i]}}{-d_{B[i]}} : d_{B[i]} < 0 \right\} \tag{5}$$

where d is an improving direction. This direction is constructed in such way that the ray {x + td : t > 0} crosses the feasible region of (2). The notation d_B denotes those components from d which correspond to the basic variables. The d_B is computed as following:

$$d_B = -\sum_{j \in P} h_j \tag{6}$$

Where $h_j = B^{-1} A_{.j}$. If $d_B \geq 0$, then the problem is unbounded.

In order to compute the entering variable x_l, the following rations must first be computed:

$$\theta_1 = -\frac{-s_Q}{H_{rQ}} = \min\left\{ \frac{-s_j}{H_{rj}} : H_{rj} > 0 \ \wedge \ j \in P \right\} \tag{7}$$

and

$$\theta_2 = -\frac{-s_Q}{H_{rQ}} = \min\left\{ \frac{-s_j}{H_{rj}} : H_{rj} < 0 \ \wedge \ j \in Q \right\} \tag{8}$$

If $\theta_1 \leq \theta_2$ then l = p, otherwise (e.g., $\theta_1 > \theta_2$) l = q. The non-basic variable x_l enters the basis. A formal description of the EPSA is given below.

EPSA algorithm

Step 0. (*Initialization*).

Start with a feasible partition (B, N). Compute B^{-1} and vectors x_B, w and s_N. Find the sets of indices P and Q using relations (3) and (4). Define an arbitrary vector $\lambda = (\lambda_1, \lambda_2, ..., \lambda_{|P|}) > 0$ and compute s_0 as follows:

$$s_0 = \sum_{j \in P} \lambda_j s_j \tag{7}$$

and the direction d_B from (5).

Step 1. (*Termination test*).

i) (*Optimilaty test*). If $P = \varnothing$, STOP. The problem is optimal.

ii) (*Leaving variable selection*). If $d_B \geq 0$, STOP. If $s_0 = 0$ the problem is optimal. If $s_0 < 0$ the problem is unbounded. Otherwise choose the leaving variable $x_{B[r]} = x_k$ using (5).

Step 2. (*Entering variable selection*).

Compute the row vectors:

$$H_{rP} = (B^{-1})_{r.} A_P \text{ and } H_{rQ} = (B^{-1})_{r.} A_Q \tag{9}$$

Compute the ratios θ_1 and θ_2 using relations (7) and (8). Determine the indices t_1 and t_2 such that $P[t_1] = p$ and $Q[t_2] = q$. If $\theta_1 \leq \theta_2$, set l = p, otherwise (e.g., $\theta_1 > \theta_2$) l = q. The non-basic variable xl enters the basis.

Step 3. (*Pivoting*)

Set B[r] = l. If $\theta_1 \leq \theta_2$, set $P \leftarrow P \backslash \{l\}$ and $Q \leftarrow Q \cup \{k\}$. Otherwise, set $Q[t_2] = k$. Using the new partition (B, N) where N = (P, Q), update the matrix B-1 and the vectors x_B, w and s_N. Also update $\overline{d_B}$ as follows:

$$\overline{d_B} = E^{-1} d_B \tag{10}$$

If $l \in P$ set $d_{B[r]} \leftarrow d_{B[r]} + \lambda_l$. Go to step 1.

Proof of correctness of the above algorithm can be found in [25, 28]. In order to solve general linear optimization problems we applied a hybrid algorithm. This version of algorithm embodies unchanged the phase I of PSA and actually applies EPSA in phase II. Thus, in phase I this algorithm works exactly as PSA, and when a feasible partition (solution) is found EPSA is applied. Therefore this algorithm is the union of PSA in phase I and EPSA in phase II and as a result it's a hybrid algorithm.

III. PARALLEL MATRIX MULTIPLICATION

Multiplication of two matrices, A and B, produces the matrix C, whose elements $c_{i,j}$ ($0 \leq i < n$, $0 \leq j < m$), can be computed as follows:

$$c_{i,j} = \sum_{k=0}^{l-1} a_{i,k} b_{k,j} \tag{11}$$

where A is an nxl matrix, and B is an lxm matrix. Each element of the *i*th row of A is multiplied by an element of the *j*th column of B, and the products are summed together to obtain the value of the element in the *i*th row and the *j*th column of C, as illustrated in Figure 1.

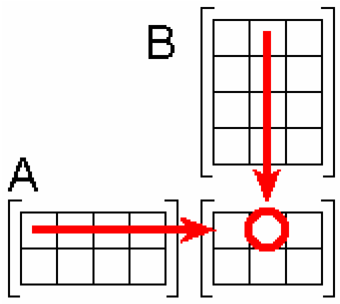

Figure 1. Matrix multiplication scheme

For convenience, let us assume that the matrices are square (nxn matrices). From the definition of matrix multiplication given above, the sequential code to compute AxB could simply be:

```
for (i = 0; i < n; i++)
  for (j = 0; j <n; j++) {
    C[i][j] = 0;
    for (k = 0; k < n; k++)
         C[i][j] = C[i][j] + A[i][k] * B[k][j];

  }
```

The algorithm requires n^3 multiplications and n^3 additions, leading to a sequential time complexity of $O(n^3)$.

Parallel matrix multiplication is usually based upon the direct sequential matrix multiplication algorithm. Even a superficial look at the sequential code reveals that the computation in each iteration of the two outer loops is not dependent upon any other iteration, and each instance of the inner loop could be executed in parallel. Hence, with p = n processors, we can expect a parallel time complexity of $O(n^2)$.

Usually, we want to use far fewer than n processors with nxn matrices because of the size of n. Then each processor operates upon a group of data points. Each matrix can be divided into blocks of elements called submatrices. These submatrices can be manipulated as if they were single matrix elements [13]. Let us select mxm submatrices and s = n/m, that is s rows and columns of mxm submatrices. Then there are s^2 submatrices in each matrix and s^2 processors.

Communication. Each of the s^2 slave processors must receive one row and one column of submatrices, each consisting of m^2 elements. In addition, each slave processor must return a submatrix C to the master processor (m^2 elements), giving a communication time of:

$$
\begin{aligned}
t_{comm} &= s^2 \left\{ 2(t_{startup} + nmt_{data}) + (t_{startup} + m^2 t_{data}) \right\} \\
&= \left(\frac{n}{m} \right)^2 \left\{ 3t_{startup} + (m^2 + 2nm) t_{data} \right\}
\end{aligned} \qquad (12)
$$

Computation. Each slave performs in parallel s submatrix multiplications and s submatrix additions. One sequential submatrix multiplication requires m^3 multiplications and m^3 additions. A submatrix addition requires m^2 additions. Hence:

$$t_{comp} = s\left(2m^3 + m^2\right) \qquad (11)$$

Hence, the time complexity for this computation is $O(sm^3) = O(nm^2)$.

The block matrix multiplication algorithm suggests a recursive divide-and-conquer solution, as described in [15] and [18]. This method has significant potential for parallel implementations, especially shared memory implementations.

Furthermore, matrix multiplication can be implemented with a two-dimensional mesh. There are several ways that matrix multiplication can be developed for a mesh organization. The most well-known are: Cannon's algorithm [5] and the systolic approach. Another matrix multiplication algorithm was devised by Fox; details can be found in Fox et al. [13].

IV. PARALLEL EXTERIOR-POINT ALGORITHM

Since the parallelization of all individual steps of the revised simplex method is limited and very hard to achieve, it is important to consider how the method itself can be modified to allow the maximum degree of independence between the computational steps in different iterations. However, it is also essential that any algorithm performs basis inverse in parallel with simplex iterations, otherwise basis inverse will then become the dominant step and limit the possible speed-up.

Our parallel implementation focus on the reduction of the time taken to perform the basis inverse. The basis inversion is done with the Product Form of the Inverse (PFI) scheme. The parallel implementation is based on a master – slave architecture. The master performs all the steps of the exterior-point simplex algorithm and the slaves only compute their portion of the new basis.

Let us assume that the matrices E (eta matrix) and B (basis) are square (n x n matrices). Furthermore, we have s processors. The E matrix is splitted into s submatrices with m = n/s rows and n columns each, as shown in Figure 2. The master broadcast the B matrix to each processor. Each processor will compute n/s rows of the new basis. The elements of these rows have to be sending back to the master. Finally, the master processor joins the submatrices to obtain the new basis.

A. Pseudo-code for the master task

Table I presents the pseudo-code for the master task used for the basis inverse.

TABLE I.
MASTER TASKS' PSEUDO-CODE

```
1. B = labBroadcast(1, B);
2. for i=2:Numoflabs
3.     labSend(E((i-1)*d/Numoflabs+1:i*d/Numoflabs,:),i,2);
4. end
5. new_B=E(1:d/Numoflabs,:)*B;
6. for i=2:Numoflabs
7.     d2=labReceive(i,3);
8.     B=[B; d2];
9. end
```

where Numoflabs is the number of workers used and d is the rows of the instance.

Initially, the master broadcasts the matrix B to the workers. In lines 2 to 4, the distribution of matrix E is being performed. The master holds the first d/Numoflabs rows and sends the other to workers. Each node has equal number of rows. Next, the master computes its' portion of the new basis. In lines 6 to 9, the master receives the results from other workers and joins the submatrices to form the new basis.

B. Pseudo-code for the slave tasks

Table II introduces the pseudo-code for the slave tasks used for the basis inverse.

TABLE II.
SLAVE TASKS' PSEUDO-CODE

```
1. B=labBroadcast(1);
2. E=labReceive(1,2);
3. new_B=E*B;
4. labSend(B,1,3);
```

Initially, the slave receives the whole matrix B from the master process (line 1) and its' portion of the matrix E (line 2). Next, the slave process performs its' computation (line 3) and sends back to the master the submatrix of the new basis.

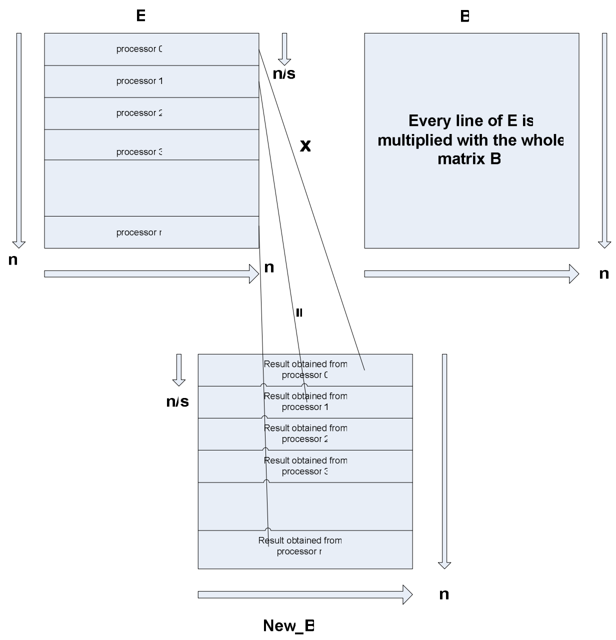
E
B
processor 0
processor 1
processor 2
processor 3
processor n
n/s
X
Every line of E is multiplied with the whole matrix B
n
n
n
=
n/s
Result obtained from processor 0
Result obtained from processor 1
Result obtained from processor 2
Result obtained from processor 3
Result obtained from processor n
n
New_B

Figure 2. Block Matrix multiplication scheme

V. COMPUTATIONAL EXPERIMENTS

The three more usual approaches to analyzing algorithms are i) worst-case analysis, ii) average-case analysis, and iii) experimental analysis. Computational studies have proven useful tools in order to examine the practical efficiency of an algorithm, or even compare algorithms by using the same problem sets.

A. *Computing environment*

The comparative computational study has been performed on an Intel Core 2 Duo T5550 1.83 GHz, with 3Gb RAM running under Windows Professional XP 32-bit Edition SP3. The algorithms have been implemented using MATLAB R2009a 32-bit Professional Edition.

The parallel implementation uses the Parallel Computing Toolbox of the MATLAB R2009a x86 environment. Parallel Computing Toolbox enables us to solve computationally and data-intensive problems using MATLAB and Simulink on multicore and multiprocessor computers. Parallel processing constructs, such as parallel for-loops and code blocks, distributed arrays, parallel numerical algorithms, and message-passing functions let you implement task- and data-parallel algorithms in MATLAB at a high level, without the restriction for programming for a specific hardware and network architectures. The toolbox can be used to execute applications on a single multicore or multiprocessor desktop. Without changing the code, the same application can run on a computer cluster (using MATLAB Distributed Computing Server).

Parallel MATLAB applications can be distributed as executables or shared libraries that can access MATLAB Distributed Computing Server. Facilities are offered for using high-level constructs such as distributed arrays, parallel algorithms, and functions for exchanging information between processes using message passing procedures like broadcast, send, receive and others. Parallel Computing Toolbox has the ability to run eight workers locally on a multicore desktop and can be integrated with MATLAB Distributed Computing Server for cluster-based applications that use any scheduler or any number of workers.

B. *Problem instances*

The instances are listed in Table III. For each problem type of a particular size, 10 instances were generated, using a different seed number. All instances are randomly generated optimal problems. For each instance, we averaged times over 5 runs. All instances are on average 98% dense.

C. *Results*

In this section, computational results regarding to our implementations are reported. The results given in Table IV show the average times of the sequential exterior-point simplex algorithm for 5 executions. All times are displayed in seconds.

TABLE III.
DIMENSIONS OF PROBLEMS

Problem name	Number of rows	Number of columns
pr1	500	500
pr2	1000	1000
pr3	1500	1500
pr4	2000	2000
pr5	2500	2500
pr6	3000	3000

Table IV shows that the computation time of simplex type algorithms depends on the basis inverse that occurs in each iteration.

TABLE IV.
TOTAL TIME OF THE SEQUENTIAL EXTERIOR-POINT SIMPLEX ALGORITHM

Dimension	Total time	Time of basis inverse
pr1	21.81	8.01
pr2	228.10	89.84
pr3	861.37	340.59
pr4	1659.14	640.15
pr5	3731.81	1231.62
pr6	7474.44	2567.89

Tables V and VI present the results from the execution of the parallel implementation with 2 and 4 workers, respectively. In both tables, columns 2, 3, 4 and 5 show the total time, the computational time, the communication time and the time of the basis inverse, respectively. These results are also graphically illustrated in Figures 3 and 4.

TABLE V.
TOTAL, COMPUTATION, COMMUNICATION AND BASIS INVERSE TIME OF THE PARALLEL IMPLEMENTATION USING 2 WORKERS

Dimension	Total Time	Computation Time	Communication Time	Time of basis inverse
pr1	37.72	14.24	23.48	3.88
pr2	424.59	160.96	263.63	42.20
pr3	1384.70	595.92	788.80	167.51
pr4	2548.40	1123.80	1424.60	355.07
pr5	6061.60	2540.50	3521.10	786.71
pr6	12536.70	4839.10	7697.10	1403.90

TABLE VI.

TOTAL, COMPUTATION, COMMUNICATION AND BASIS INVERSE TIME OF THE PARALLEL IMPLEMENTATION USING 4 WORKERS

Dimension	Total time	Computation time	Communication time	Time of basis inverse
pr1	85.02	14.20	70.82	0.37
pr2	586.32	148.78	437.52	17.96
pr3	1577.10	484.62	1092.50	76.80
pr4	2957.50	859.38	2098.20	126.58
pr5	6131.90	1905.10	4226.80	273.84
pr6	11789.45	3985.54	7803.91	526.54

Table VII presents the time taken to perform the basis inverse using 1, 2 and 4 workers, respectively. Figure 5 is the graphical representation of the results shown in Table VII. Table VII shows that we have a significant reduction of the computation time needed by the basis inverse. The speed-up gained from the parallelization of the basis inverse is of average 4.72. Due to very dense matrices and very heavy communication, the ratio of computation to communication is extremely low. As a result the parallel implementation does not offer any speed-up to the total time.

TABLE VII.
BASIS INVERSE TIME USING 1, 2 AND 4 WORKERS, RESPECTIVELY

Dimension	1 worker	2 workers	4 workers
pr1	8.00	3.88	0.37
pr2	89.84	42.20	17.96
pr3	340.59	167.51	76.80
pr4	640.15	355.07	126.58
pr5	1231.60	786.71	273.84
pr6	2567.89	1403.90	526.54

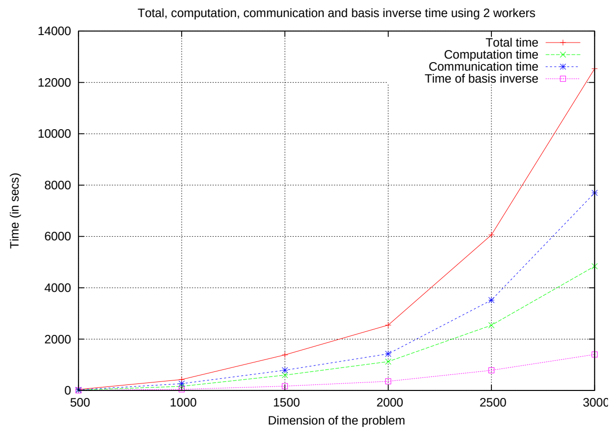

Figure 3. Total, computation, communication and basis inverse times of the parallel implementation using 2 workers

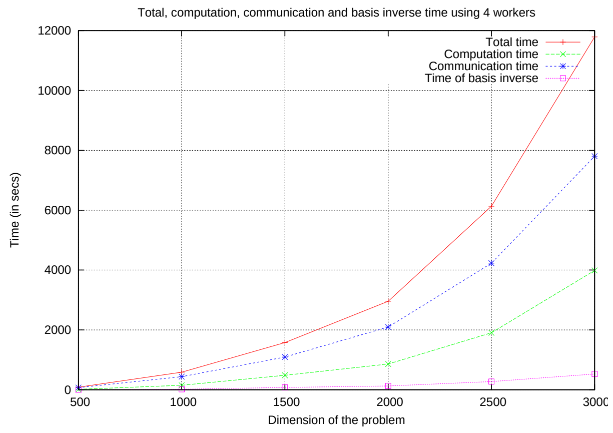

Figure 4. Total, computation, communication and basis inverse times of the parallel implementation using 4 workers

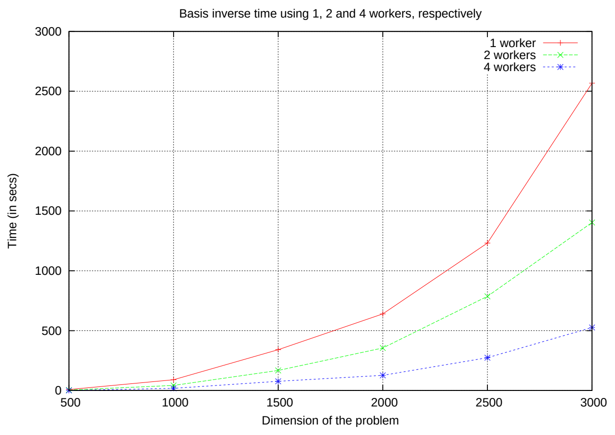

Figure 5. Basis inverse time using 1, 2 and 4 workers, respectively

VI. CONCLUSIONS

A parallel algorithm for the exterior-point revised simplex method has been described and a speed-up of average 4.72, regarding the basis inverse procedure, was demonstrated by the computational results. Due to very dense matrices and very heavy communication, the ratio of computation to communication is extremely low. As a result the parallel implementation does not offer any speed-up to the total time. However, these results could be further improved by performance optimization.

In summary, parallelizing simplex algorithms is not an easy task. Due to heavy communication, the computational results show that it is hard to achieve a linear speed-up even with carefully selected partitioning patterns and communication optimization.

REFERENCES

[1] A. Agrawal, G. E. Blelloch, R. L. Krawitz, and C. A. Phillips, “Four vector-matrix primitives”, In *ACM Symposium on Parallel Algorithms and Architectures*, pp. 292-302, 1989.

[2] E. S. Badr, M. Moussa, K. Papparrizos, N. Samaras, and A. Sifaleras, “Some computational results on MPI parallel implementations of dense simplex method”, *Transactions on Engineering, Computing and Technology*, Vol. 17, pp. 228-231, 2006.

[3] M. Baker, R. Buyya and D. Laforenza, “Grids and Grid technologies for wide-area distributed computing”, *Software-Practice and Experience*, Vol. 32(15), pp. 1437-1466. 2002.

[4] R. S. Barr and B. L. Hickman, “Parallel Simplex for Large Pure Network Problems: Computational Testing and Sources of Speedup”, *Operations Research*, Vol. 42(1), 1994, pp. 65-80.

[5] L. E. Cannon, *A Cellular Computer to Implement the Kalman Filter Algorithm*, Ph.D. Thesis, Montana State University, Bozman, MT, 1969.

[6] M. D. Chang, M. Engquist, R. Finkel and R. R. Meyer, “A Parallel Algorithm for Generalized Networks”, *Annals of Operations Research*, Vol. 14(1-4), pp. 125-145, 1988.

[7] J. C. Cunha, O. F. Rana and P. D. Medeiros, “Future trends in distributed applications and problem-solving environments”, *Future Generation Computer Systems*, Vol. 21(6), pp. 843-855, 2005.

[8] Z. Cvetanovic, E. G. Freedman and C. Nofsinger, “Efficient decomposition and performance of parallel PDE, FFT, Monte-Carlo simulations, simplex, and sparse solvers”, *Journal of Supercomputing*, Vol. 5(2-3), pp. 219-238, 1991.

[9] G. B. Dantzig, “Programming in a Linear Structure”, Report of the September 9, 1948 meeting in Madison, Econometrica, Vol. 17, pp. 73-74, 1949.

[10] J. Eckstein, I. Boduroglu, L. Polymenakos and D. Goldfarb, "Data-Parallel Implementations of Dense Simplex Methods on the Connection Machine CM-2," *ORSA Journal on Computing*, Vol. 7(4), pp. 402-416, 1995.

[11] R. A. Finkel, "Large-Grain Parallelism: Three Case Studies", in *the Characteristics of Parallel Algorithms*, Ed. L. H. Jamieson, The MIT Press, 1987.

[12] I. Foster, C. Kesselman and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *The International Journal of High Performance Computing Applications*, Vol. 15(3), pp. 200-222, 2001.

[13] G. Fox, G. Johnson, S. Otto, J. Salmon and D. Walker, "Solving Problems on Concurrent Processors", Volume 1, Prentice Hall, Englewood Cliffs, NJ, 1988.

[14] D. M. Gay, "Electronic mail distribution of linear programming test problems". *Mathematical Programming Society COAL Newsletter*, Vol. 13, pp. 10-12, 1985.

[15] J. F. Hake, "Parallel Algorithms for Matrix Operations and Their Performance in Multiprocessor Systems", in Advances in *Parallel Algorithms*, L. Kronsjo and D. Shumsheruddin, eds., Halsted Press, New York, 1993.

[16] J. A. J. Hall and K. I. M. McKinnon, "ASYNPLEX an asynchronous parallel revised simplex algorithm," *Annals of Operations Research*, Vol. 81(0), pp. 27-50, 1988.

[17] R. V. Helgason, J. L. Kennington, H. A. Zaki, "A parallelization of the simplex method", *Annals of Operations Research,* Vol. 14(1-4), pp.17-40, 1988.

[18] E. Horowitz and A. Zorat, "Divide-and-Conquer for Parallel Processing", *IEEE Trans. Comput*., Vol. C-32(6), pp. 582-585, 1983.

[19] G. Karypis and V. Kumar, *Performance and Scalability of the Parallel Simplex Method for Dense Linear Programming Problems an Extended Abstract*, Technical Report, Computer Science Department, University of Minessota, 1994.

[20] A. Kilgore, *Very Large-scale Linear Programming: A Case Study Exploiting Both Parallelism and Distributed Memory*, MSc Thesis, Center for Research on Parallel Computation, Rice University, 1993.

[21] M. Lentini, A. Reinoza, A. Teruel and A. Guillen, "SIMPAR: a parallel sparse simplex". *Computational and Applied Mathematics*, Vol. 14(1), pp. 49-58, 1995.

[22] I. Maros and G. Mitra, "Investigating the sparse simplex algorithm on a distributed memory multiprocessor", *Parallel Computing*, Vol. 26(1), pp. 151-170, 2000.

[23] A. K. Noor, "New Computing Systems and Future High-Performance Computing Environment and their Impact on Structural Analysis and Design", *Computers and Structures*, Vol. 64(1-4), pp. 1-30, 1997.

[24] K. Paparrizos, "Pivoting rules directing the Simplex method though all feasible vertices of Klee-Minty examples", *OpSearch*, Vol. 26, pp.77-95, 1989.

[25] K. Paparrizos, "Exterior point simplex algorithms, simple and short proof of correctness", Proceedings of *SYMOPIS '96*, pp. 13-18, 1996.

[26] K. Paparrizos, "Pivoting algorithms generating two paths", presented in ISMP '97, Lausanne, EPFL Switzerland, 1997.

[27] K. Paparrizos, N. Samaras and G. Stephanides, "An efficient simplex type algorithm for sparse and dense linear programs", *European Journal of Operational Research*, vol. 148(2), pp. 323-334, 2003.

[28] K. Paparrizos, N. Samaras, K. Tsiplidis, "Pivoting algorithms for (LP) generating two paths" in : M.P Pardalos, A.C. Floudas (eds.) Encyclopedia of Optimization, Vol. 4, Kluwer Academic Publishers, pp. 302-306, 2001.

[29] J. Peters, "The Network Simplex Method on a Multiprocessor", *Networks*, Vol. 20(7), pp. 845-859, 1990.

[30] C. Roos, "An exponential example for Terlaky's pivoting rule for the Criss-cross Simplex method", *Mathematical Programming*, Vol. 46(1), pp. 79-84, 1990.

[31] W. Shu, "Parallel implementation of a sparse simplex algorithm on MIMD distributed memory computers". *Journal of Parallel and Distributed Computing*, Vol. 31(1), pp. 25-40, 1995.

[32] W. Shu and M. Wu, "Sparse Implementation of Revised Simplex Algorithms on Parallel Computers", Sixth SIAM Conference on Parallel Processing for Scientific Computing, March 22-24, Norfolk, 1993.

[33] B. Skillicorn and D. Talia, "Models and Languages for Parallel Computation", ACM Computing Surveys, Vol. 30(2), pp. 123-169, 1998.

[34] C. B. Stunkel, "Linear optimization via message-based parallel processing", In *International Conference on Parallel Processing*, Vol. 3, pp. 264-271, 1988.

[35] M. E. Thomadakis and J. C. Liu, "An Efficient Steepest-Edge Simplex Algorithm for SIMD Computers," Proc. Of *the International conference on Super-Computing*, ICS '96, pp. 286-293, 1996.

[36] Y. Wu and T. G. Lewis, *Performance of Parallel Simplex Algorithms*, Tech. Report, Dep't of Computer Science, Oregon State U., 1988.

[37] G. Yarmish, *A Distributed Implementation of the Simplex Method*, Ph.D thesis, Polytechnic University, Brooklyn, NY, 2001.

AUTHORS

N. Ploskas is with the Applied Informatics Department, University of Macedonia, Greece (e-mail: mai084@uom.gr).

N. Samaras is with the Applied Informatics Department, University of Macedonia, Greece (e-mail: samaras@uom.gr).

A. Sifaleras is with the Technology Management Department, University of Macedonia, Greece (e-mail: sifalera@uom.gr).

Manuscript received 20 July 2009.

Published as submitted by the author(s).