A parallel implementation of the revised simplex algorithm using OpenMP: some preliminary results

Ploskas Nikolaos
PhD Candidate
Department of Applied Informatics,
University of Macedonia,
156 Egnatia Str.,
54006 Thessaloniki, Greece

Samaras Nikolaos
Assistant Professor
Department of Applied Informatics,
University of Macedonia,
156 Egnatia Str.,
54006 Thessaloniki, Greece

Margaritis Konstantinos
Professor
Department of Applied Informatics,
University of Macedonia,
156 Egnatia Str.,
54006 Thessaloniki, Greece

Abstract

Linear Programming (LP) is a significant area in the field of operations research. The simplex method is the most widely used method for solving LP problems. The aim of this paper is to present a parallel implementation of the revised simplex algorithm. Our parallel implementation focuses on the reduction of the time taken to perform the basis inverse, due to the fact that the total computational effort of an iteration of simplex type algorithms is dominated by this computation. This inverse does not have to be computed from scratch at any iteration. In this paper, we compute the basis inverse with two well-known updating schemes: (i) The Product Form of the Inverse (PFI) and (ii) A Modification of the Product Form of the Inverse (MPFI); and incorporate them with revised simplex algorithm. Apart from the parallel implementation, this paper presents a computational study that shows the speedup among the serial and the parallel implementations in large-scale LP problems. The test set used in the computational study is the Netlib set of linear problems. The parallelism is achieved using OpenMP in a shared memory multiprocessor architecture.

KEYWORDS

Linear Programming, Revised Simplex Method, Basis Inverse, Parallel Computing, OpenMP.

1. INTRODUCTION

Linear Programming (LP) is the process of minimizing or maximizing a linear objective function $z = \sum_{i=1}^{n} c_i x_i$ to a number of linear equality and inequality constraints. Several methods are available for solving LP problems, among which the simplex algorithm is the most widely used. We assume that the problem is in its general form. Formulating the linear problem, we can describe it as shown below:

$$\begin{aligned} &\min && c^T x \\ &\textit{subject to} && Ax = b \qquad (1) \\ & && x \geq 0 \end{aligned}$$

where $A \in R^{mxn}$, $(c, x) \in R^n$, $b \in R^m$, 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. The dual problem associated with the linear problem in equation (1) is shown below:

$$\begin{aligned} &\min && b^T w \\ &\textit{subject to} && A^T w + s = c \qquad (2) \\ & && s \geq 0 \end{aligned}$$

where $w \in R^m$ and $s \in R^n$. As in the solution of any large scale mathematical system, the computational time for large LP problems is a major concern. Parallel programming is a good practice for solving computationally intensive problems in operations research. The application of parallel processing for linear programming has been introduced in the early 1970s. However, only since the beginning of the 1980s attempts have been made to develop parallel implementations.

One of the earliest parallel tableau simplex methods on a small-scale distributed memory Multiple-Instruction Multiple-Data (MIMD) machines is the one introduced by Finkel (1987). Stunkel (1988) implemented both the tableau and the revised simplex method on a 16-processor Intel hypercube computer, achieving a speedup of between 8 and 12 for small problems from the Netlib set (Gay, 1985). Helgason, Kennington and Zaki (1988) proposed an algorithm to implement the revised simplex using sparse matrix methods on shared memory MIMD computer. Furthermore, Shu and Wu (1993) and Shu (1995) parallelized the explicit inverse and the LU decomposition of the basis simplex algorithms. Hall and McKinnon (1996; 1998) have implemented two parallel schemes for the revised simplex method. The first of Hall and McKinnon's parallel revised simplex implementations was ASYNPLEX (1996). In this implementation one processor is devoted to the basis inversion and the remaining processors perform simplex iterations. ASYNPLEX was implemented on a Cray T3D, achieving a speedup of between 2.5 and 4.8 for four modest Netlib problems. The second of Hall and McKinnon's parallel revised simplex implementations was PARSMI (1998). PARSMI was tested on modest problems from the Netlib set, resulting in a speedup of between 1.7 and 1.9. Hall (2005) implemented a variant of PARSMI on a 8-processor shared memory Sun Fire E15k, leading in a speedup of between 1.8 and 3.

Simplex algorithms for general LP problems on Single Instruction Multiple Data (SIMD) have been reported by Agrawal et al. (1989). Luo and Reijns (1992) presented an implementation of the revised simplex method, achieving a speedup of more than 12, when solving modest Netlib problems on 16 transputers. Eckstein et al (1995) implemented a parallelization of standard and revised simplex method in a CM2 machine. Lentini et al (1995) worked on the standard simplex method with the tableau stored as a sparse matrix, resulting in a speedup of between 0.5 and 2.7, when solving medium sized Netlib problems on four transputers. Thomadakis and Liu (1996) worked on the standard simplex method on MasPar MP-1 and MP-2 machines, achieving a speedup of up to three, when solving large randomly-generated problems. Badr et al (2006) implemented a dense standard simplex method on eight computers, leading in a speedup of five when solving small random dense LP problems.

The use of GPUs for general purpose computations is a quite recent topic, which was applied to linear programming. Greeff (2004) implemented the revised simplex method on a GPU using OpenGL and Cg and was able to achieve a speedup of up to 11.4 over an identical CPU implementation. Jung and O'Leary (2008) and Owens et al (2008) also presented an implementation using Cg and OpenGL. Spampinato and Elster (2009) proposed a GPU implementation of the revised simplex method, based on the CUDA architecture and achieved a speedup of up to 2.5. Recently, Bieling, Peschlow and Martini (2010) also presented an implementation of the revised simplex algorithm and achieved a speedup of up to 10.

Finally, computational results for parallelizing the network simplex method are reported in (Chang et al, 1988; Barr and Hickman, 1994; Peters, 1999).

This paper presents a parallelization of the revised simplex algorithm on a shared memory multiprocessor architecture. The focus of this parallelization is on the basis inverse. The structure of the paper is as follows. In Section 2, the revised simplex algorithm is described and presented. In Section 3, two methods that have been widely used for basis inversion are analyzed. Section 4 presents the parallel revised simplex algorithm and section 5 gives the computational results. Finally, the conclusions of this paper are outlined in section 6.

2. REVISED SIMPLEX ALGORITHM

The linear problem in equation (1) can be written as:

$$\begin{array}{ll} \min & c_B^T x_B + c_N^T x_N \\ \textit{subject to} & A_B x_B + A_N x_N = b \qquad (3) \\ & x_B, x_N \geq 0 \end{array}$$

In the above equation, B is a mxm non-singular sub-matrix of A, called basic matrix or basis. The columns of A which belong to subset B are called basic and those which belong to N are called non basic. The solution of the linear problem $x_B = B^{-1}b, x_N = 0$ is called a basic solution. A solution $x = (x_B, x_N)$ is feasible if x > 0. Otherwise the solution is infeasible. The solution of the linear problem in equation (2) is computed by the relation $s = c - A^T w$, where $w = (c_B)^T B^{-1}$ are the simplex multipliers and s are the dual slack variables. The basis B is dual feasible if $s \geq 0$.

In each iteration, simplex algorithm interchanges a column of matrix B with a column of matrix N and constructs a new basis $\overline{B}$. Any iteration of simplex type algorithms is relatively expensive. The total work of an iteration of simplex type algorithms is dominated by the determination of the basis inverse. This inverse however, does not have to be computed from scratch during each iteration. Simplex type algorithms maintain a factorization of basis and update this factorization in each iteration. There are several schemes for updating basis inverse. The most well-known schemes are (i) the Product Form of the Inverse (PFI) and (ii) a Modification of the Product Form of the Inverse, developed by Benhamadou (2002). These methods, in order to compute the new basis, use only information about the entering and leaving variables along with the current basis. A formal description of the revised simplex algorithm (Dantzig, Orden and Wolfe, 1953) is given below.

Table 1. Revised Simplex Algorithm.

Step 0. (*Initialization*).

Start with a feasible partition (B, N). Compute B^{-1} and vectors x_B, w and s_N.

Step 1. (*Test of optimality*).

if $s_N \geq 0$ then STOP. The linear problem is optimal.

else

Choose the index l of the entering variable using a pivoting rule.

Variable x_l enters the basis.

Step 2. (*Minimum ratio test*).

Compute the pivot column $h_l = B^{-1} A_{.l}$.

if $h_l \leq 0$ then STOP. The linear problem is unbounded.

else

Choose the leaving variable $x_{B[r]} = x_k$ using the following equation:

$$x_{B[r]} = \frac{x_{B[r]}}{h_{il}} = \min\left\{\frac{x_{B[i]}}{h_{il}} : h_{il} < 0\right\} \quad (4)$$

Step 3. (*Pivoting*).

Swap indices k and l. Update the new basis inverse $\overline{B}^{-1}$, using PFI or MPFI.

Go to Step 1.

3. METHODS USED FOR BASIS INVERSION

The revised simplex algorithm differs from the original method. The former uses the same recursion relations to transform only the inverse of the basis in each iteration. It has been implemented to reduce the computation time of the basis inversion and is particularly effective for sparse linear problems. In this section, we will review two methods that have been widely used for basis inversion: (i) the Product Form of the Inverse and (ii) a Modification of the Product Form of the Inverse.

3.1 Product Form of the Inverse

The PFI scheme, in order to compute the new basis, uses information only about the entering and leaving variables along with the current basis. The new basis inverse can be updated at any iteration using the equation below:

$$\overline{B}^{-1} = (BE)^{-1} = E^{-1}B^{-1} \quad (5)$$

where E^{-1} is the inverse of the eta-matrix and can be computed by:

$$E^{-1} = I - \frac{1}{h_{rl}}(h_l - e_l)e_l^T = \begin{bmatrix} 1 & & -h_{1l}/h_{rl} & & \\ & \ddots & \vdots & & \\ & & 1/h_{rl} & & \\ & & \vdots & \ddots & \\ & & -h_{ml}/h_{rl} & & 1 \end{bmatrix}$$

If the current basis inverse is computed using regular multiplication, then the complexity of the PFI is $\Theta(m^3)$.

3.2 A Modification of Product Form of the Inverse

MPFI updating scheme has been presented by Benhamadou (2002). The key idea is that the current basis inverse $\overline{B}^{-1}$ can be computed from the previous inverse B^{-1} using a simple outer product of two vectors and one matrix addition. Namely:

$$(\overline{B})^{-1} = (\overline{B})^{-1}_{r.} + v \otimes (B)^{-1}_{r.} \qquad (6)$$

The updating scheme of the inverse is shown in Fig. 1.

Figure 1. A modification of the PFI scheme.

$$B^{-1} : \begin{vmatrix} b_{r1} & \dots & b_{rr} & \dots & b_{rm} \end{vmatrix}$$

$$\overline{B}^{-1} = \begin{vmatrix} b_{11} & \dots & b_{1m} \\ \dots & \dots & \dots \\ 0 & 0 & 0 \\ \dots & \dots & \dots \\ b_{ml} & \dots & b_{mm} \end{vmatrix} + \begin{vmatrix} -\frac{h_{l1}}{h_{rl}} \\ \dots \\ -\frac{1}{h_{rl}} \\ \dots \\ -\frac{h_{m1}}{h_{rl}} \end{vmatrix}$$

The outer product requires m^2 multiplications and the addition of two matrices requires m^2 additions. The total cost of the above method is $2m^2$ operations (multiplications and additions). Hence, the complexity is $\Theta(m^2)$.

4. PARALLEL REVISED SIMPLEX ALGORITHM

The parallelization of all the individual steps of the revised simplex method is limited and very hard to achieve. However, it is also essential for any algorithm to perform basis inverse in parallel with simplex iterations, otherwise basis inverse will become the dominant step and limit the possible speedup. Our parallel implementation focuses on the reduction of the time taken to perform the basis inverse. The basis inversion is done with the Product Form of the Inverse and a Modification of the Product Form of the Inverse, as described in the previous section.

Both methods take as input the previous basis inverse (B^{-1}), the pivot column (h_l), the index of the leaving variable (k) and the number of the constraints (n).

The most time-consuming step of PFI scheme is the matrix multiplication of relation (5). Our parallel algorithm uses the block matrix multiplication algorithm for this step. This algorithm suggests a recursive divide-and-conquer solution, as described in (Hake, 1993; Horowitz and Zorat, 1983). This method has significant potential for parallel implementations, especially on shared memory implementations.

Let us assume that we have *p* processors. We use the following steps to compute the new basis inverse $\overline{B}^{-1}$ with the PFI scheme:

Table 2. Parallel Product Form of the Inverse.

Step 0.
Compute the column vector:

$$v = \left[-\frac{h_{1l}}{h_{rl}} \quad \cdots \quad \frac{1}{h_{rl}} \quad \cdots \quad -\frac{h_{ml}}{h_{rl}} \right]^T$$

Each processor computes in parallel n/p elements of v.
Step 1.
Replace the r^{th} column of an identity matrix with the column vector v. Each processor assigns in parallel n/p elements to the identity matrix. This matrix is the inverse of the Eta-matrix.
Step 2.
Compute the new basis inverse using relation (5) with block matrix multiplication. Each processor will compute n/p rows of the new basis.

We use the following steps to compute the new basis inverse $\overline{B}^{-1}$ with the MPFI scheme:

Table 3. Parallel Modification of Product Form of the Inverse.

Step 0.
Compute the column vector:

$$v = \left[-\frac{h_{1l}}{h_{rl}} \quad \cdots \quad \frac{1}{h_{rl}} \quad \cdots \quad -\frac{h_{ml}}{h_{rl}} \right]^T$$

Each processor computes in parallel n/p elements of v.
Step 1. (The following steps are computed in parallel)
Step 1.1. Compute the outer product $v \otimes B^{-1}_r$ with block matrix multiplication.
Step 1.2. Copy matrix B^{-1} to matrix $\overline{B}^{-1}$. Set the r^{th} row of $\overline{B}^{-1}$ equal to zero. Each processor computes in parallel n/p rows of $\overline{B}^{-1}$.
Step 2.
Compute the new basis inverse using relation (6). Each processor computes in parallel n/p rows of the new basis.

5. COMPUTATIONAL EXPERIMENTS

The three most usual approaches to analyzing algorithms are i) worst-case analysis, ii) average-case analysis and iii) experimental analysis. Computational studies have been proven useful tools in order to examine the practical efficiency of an algorithm or even compare algorithms by using the same problem sets. The computational comparison has been performed on a quad-processor Intel Xeon 3.2 GHz with 2 Gbyte of main memory running under Ubuntu 10.10 64-bit and performed on GCC 4.5.2. The algorithms have been implemented using C++ and OpenMP.

5.1 Problem Instances

Below there are some useful information about the data set, which was used in the computational study. The first column of the table includes the name of benchmarks, the second the number of constraints, the third the number of variables, the fourth the non-zero elements of matrix A and the fifth the sparsity of matrix A.

Table 4. Statistics of the benchmarks.

Problem	Constraints	Variables	Non-Zeros A	Sparcity A
agg	488	163	2410	3.03%
agg2	516	302	4284	2.75%
agg3	516	302	4300	2.76%
bandm	305	472	2494	1.73%

brandy	220	249	2148	3.92%
e226	223	282	2578	4.10%
fffff800	524	854	6227	1.39%
israel	174	142	2269	9.18%
lotfi	153	308	1078	2.29%
sc105	105	103	280	2.59%
sc205	205	203	551	1.32%
scfxm1	330	457	2589	1.72%
scfxm2	660	914	5183	0.86%
scfxm3	990	1371	7777	0.57%
scrs8	490	1169	3182	0.56%
share1b	117	225	1151	4.37%
share2b	96	79	694	9.15%
ship04l	402	2118	6332	0.74%
ship04s	402	1458	4352	0.74%
ship08l	778	4283	12802	0.38%
ship08s	778	2387	7114	0.38%
ship12l	1151	5427	16170	0.26%
ship12s	1151	2763	8178	0.26%
stocfor1	117	111	447	3.44%
klein2	477	54	4585	17.80%
klein3	994	88	12107	13.84%

5.2 Results

Table 5, presents the results from the execution of the serial and parallel implementations of the above mentioned updating schemes. For each implementation, the table shows the basis inverse and the total time. All times are displayed in seconds.

Table 5. Basis inverse and total time of the serial and parallel implementations.

PROBLEM	**Serial implementations**				**Parallel implementations**			
	PFI		**MPFI**		**PFI**		**MPFI**	
	Time of basis inverse	**Total time**	**Time of basis inverse**	**Total time**	**Time of basis inverse**	**Total time**	**Time of basis inverse**	**Total time**
agg	2.83	4.58	2.28	4.05	1.52	3.32	1.13	2.95
agg2	3.54	5.65	2.78	4.97	1.81	3.96	1.52	3.69
agg3	3.28	5.61	2.62	5.03	1.85	4.05	1.48	3.78
bandm	1.01	1.62	0.74	1.41	0.84	1.52	0.61	1.29
brandy	1.30	2.76	1.06	2.55	1.04	2.48	0.76	2.22
e226	1.66	3.34	1.38	3.09	1.22	2.82	0.85	2.50
fffff800	6.10	12.77	4.86	11.64	5.18	11.58	4.31	10.89
israel	0.82	1.73	0.63	1.65	0.53	1.48	0.45	1.31
lotfi	0.33	0.85	0.29	0.80	0.25	0.78	0.21	0.68
sc105	0.08	0.09	0.03	0.07	0.01	0.07	0.02	0.06
sc205	0.55	0.91	0.51	0.85	0.40	0.74	0.20	0.56
scfxm1	3.72	7.11	2.96	6.38	3.17	6.31	2.49	5.80
scfxm2	30.76	62.34	24.26	56.40	26.34	58.56	21.22	52.54
scfxm3	109.06	244.48	83.97	219.22	91.67	224.76	72.93	209.23
scrs8	11.17	22.20	8.69	19.81	9.56	20.20	7.34	17.90
share1b	0.15	0.29	0.09	0.26	0.11	0.25	0.08	0.23
share2b	0.05	0.11	0.04	0.10	0.01	0.10	0.03	0.09
ship04l	5.20	14.53	4.18	13.65	4.43	13.65	3.56	12.90
ship04s	1.76	4.55	1.52	4.31	1.54	4.22	1.21	4.10
ship08l	33.78	94.70	26.30	86.45	30.02	91.90	22.10	82.50
ship08s	7.43	19.07	5.99	17.49	6.50	18.09	5.01	16.95

ship12l	120.21	335.98	89.94	305.95	100.05	317.80	75.64	292.00
ship12s	17.20	43.52	13.45	39.16	12.99	42.84	10.77	36.60
stocfor1	0.04	0.06	0.03	0.05	0.02	0.06	0.02	0.04
klein2	17.36	33.01	13.53	28.80	9.85	25.30	7.55	22.74
klein3	192.50	413.33	148.69	368.50	106.92	326.70	75.30	295.01

Table 6, presents the speedup obtained by the parallel implementations regarding the basis inverse and the total time, for both PFI and MPFI schemes.

Table 6. Speedup obtained by the parallel implementations.

PROBLEM	Speedup			
	PFI		MPFI	
	Basis inverse	Total	Basis inverse	Total
agg	1.86	1.38	2.02	1.37
agg2	1.96	1.43	1.83	1.35
agg3	1.77	1.39	1.77	1.33
bandm	1.20	1.07	1.21	1.09
brandy	1.25	1.11	1.39	1.15
e226	1.36	1.18	1.62	1.24
fffff800	1.18	1.10	1.13	1.07
israel	1.55	1.17	1.40	1.26
lotfi	1.32	1.09	1.38	1.18
sc105	8.00	1.29	1.50	1.17
sc205	1.38	1.23	2.55	1.52
scfxm1	1.17	1.13	1.19	1.10
scfxm2	1.17	1.06	1.14	1.07
scfxm3	1.19	1.09	1.15	1.05
scrs8	1.17	1.10	1.18	1.11
share1b	1.36	1.16	1.13	1.13
share2b	5.00	1.10	1.33	1.11
ship04l	1.17	1.06	1.17	1.06
ship04s	1.14	1.08	1.26	1.05
ship08l	1.13	1.03	1.19	1.05
ship08s	1.14	1.05	1.20	1.03
ship12l	1.20	1.06	1.19	1.05
ship12s	1.32	1.02	1.25	1.07
stocfor1	2.00	1.00	1.50	1.25
klein2	1.76	1.30	1.79	1.27
klein3	1.80	1.27	1.97	1.25
Average	1.79	1.15	1.44	1.17

From the above results, we observe: (i) the MPFI scheme is in most problems faster than PFI both in serial and in parallel implementation, (ii) using PFI scheme, the speedup gained from the parallelization is of average 1.79 for the time of basis inverse and 1.15 for total time, and (iii) using MPFI scheme, the speedup is of average 1.44 for the time of basis inverse and 1.17 for total time.

6. CONCLUSIONS

A parallel algorithm for the revised simplex method has been described in this paper. Some preliminary computational results on Netlib problems have reported a speedup of average 1.79 and 1.44 regarding the basis inverse procedure, using PFI and MPFI updating schemes respectively. These results could be further improved by performance optimization. In future work, we plan to implement our parallel algorithm combining the Message Passing Interface (MPI) and OpenMP programming models to exploit parallelism beyond a single level. Furthermore, we intend to port our algorithm to a GPU implementation based on the CUDA architecture.

REFERENCES

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

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

Barr R.S. and Hickman B.L., 1994. Parallel Simplex for Large Pure Network Problems: Computational Testing and Sources of Speedup, *Operations Research*, Vol. 42, No. 1, pp. 65-80.

Benhamadou M., 2002. On the simplex algorithm 'revised form', *Advances in Engineering Software*, Vol. 33, pp. 769-777.

Bieling J., Peschlow P. and Martini P., 2010. An efficient GPU implementation of the revised simplex method, *Proceedings of IPDPS Workshops*, pp.1-8.

Chang M.D., Engquist M., Finkel R. and Meyer R.R., 1988. A Parallel Algorithm for Generalized Networks, *Annals of Operations Research*, Vol. 14, No. 1-4, pp. 125-145.

Dantzig G.B., Orden A. and Wolfe P., 1953. The Generalized Simplex Method. *RAND P-392-1*, August 4.

Eckstein J., Boduroglu I., Polymenakos L. and Goldfarb D., 1995. Data-Parallel Implementations of Dense Simplex Methods on the Connection Machine CM-2, *ORSA Journal on Computing*, Vol. 7, No. 4, pp. 402-416.

Finkel R.A., 1987. Large-Grain Parallelism: Three Case Studies, *Proceedings of Characteristics of Parallel Algorithms*, Ed. L. H. Jamieson, The MIT Press.

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

Greeff G., 2004. *The revised simplex method on a GPU*, Stellenbosch University, South Africa, Honours Year Project.

Hall J.A.J. and McKinnon K.I.M., 1996. PARSMI, a parallel revised simplex algorithm incorporating minor iterations and Devex pricing. In J. Wasniewski, J. Dongarra, K. Madsen, and D. Olesen, editors, *Applied Parallel Computing*, volume 1184 of Lecture Notes in Computer Science, pages 67-76. Springer.

Hall J.A.J. and McKinnon K.I.M., 1998. ASYNPLEX an asynchronous parallel revised simplex algorithm, *Annals of Operations Research*, Vol. 81, No. 0, pp. 27-50.

Hall J.A.J., 2005. SYNPLEX: a task-parallel scheme for the revised simplex method. *In Contributed talk at the second international workshop on combinatorial scientific computing (CSC05)*.

Hake J.F., 1993. 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.

Helgason R.V., Kennington J.L., Zaki H.A., 1988. A parallelization of the simplex method, *Annals of Operations Research*, Vol. 14, No. 1-4, pp.17-40.

Horowitz E. and Zorat A., 1983. Divide-and-Conquer for Parallel Processing, *IEEE Trans. Comput.*, Vol. C-32, No. 6, pp. 582-585.

Jung J.H. and O'Leary D.P., 2008. Implementing an interior point method for linear programs on a CPU-GPU system, *Electronic Transaction on Numerical Analysis*, vol. 28, pp. 174–189.

Lentini M., Reinoza A., Teruel A. and Guillen A., 1995. SIMPAR: a parallel sparse simplex, *Computational and Applied Mathematics*, Vol. 14, No. 1, pp. 49-58.

Luo J, Reijns G.L., 1992. Linear programming on transputers, *In van Leeuwen J (ed) Algorithms, software, architecture. IFIP transactions A (computer science and technology)*, Elsevier, Amsterdam, pp 525–534.

Owens J.D., Houston M., Luebke D., Green S., Stone J.E., Phillips J.C., 2008. GPU Computing, *Proceedings of the IEEE*, Vol. 96, No. 5, pp. 879 – 899.

Peters J., 1990. The Network Simplex Method on a Multiprocessor, *Networks*, Vol. 20, No. 7, pp. 845-859.

Shu W., 1995. Parallel implementation of a sparse simplex algorithm on MIMD distributed memory computers, *Journal of Parallel and Distributed Computing*, Vol. 31, No. 1, pp. 25-40.

Shu W. and Wu M., 1993. Sparse Implementation of Revised Simplex Algorithms on Parallel Computers, *Proceedings of Sixth SIAM Conference on Parallel Processing for Scientific Computin*g, March 22-24, Norfolk.

Spampinato D.G. and Elster A.C., 2009. Linear optimization on modern GPUs, *Proceedings of the 2009 IEEE International Symposium on Parallel and Distributed Processing (IPDPS '09).*

Stunkel C.B., 1988. Linear optimization via message-based parallel processing, *Proceedings of International Conference on Parallel Processing*, Vol. 3, pp. 264-271.

Thomadakis M.E. and Liu J.C., 1996. An Efficient Steepest-Edge Simplex Algorithm for SIMD Computers, *Proceedings of the International conference on Super-Computing (ICS '96)*, pp. 286-293.