On the Parallel and VLSI Implementation of the Interior Point Algorithm for Linear Programming

Mehran Mesbahi and George Papavassilopoulos

Department of Electrical Engineering—Systems
and Center for Applied Mathematics
University of Southern California
Los Angeles,CA 90089-2563

Tel: (213) 740-2351; email: mesbahi@nyquist.usc.edu

Sixth IASTED-ISMM International Conference

Abstract

We present various aspects of the parallel and VLSI implementation of the Karmarkar interior point algorithm for the Linear Programming problem. Various architectures, based on different partitioning of the input data, is proposed and the complexity of their implementation is discussed. An economic interpretation of the partitioning scheme is then presented which might prove useful for further improvements in the efficient parallelization of the interior point methods.

Keywords: Numerical Algorithms, Optimization, Parallel Computing, VLSI.

1 Introduction

The linear programming problem (LP) is to find vector $x \in R^n$ to,

$$\min c^T x \tag{1}$$

subject to:

$$Ax = b; \tag{2}$$

$$x \geq 0. \tag{3}$$

where $A \in R^{m \times n}$, $b \in R^m$, $c \in R^n$ and $m \leq n$. LP is one of the most important problems in optimization theory. This is mainly because many problems can be formulated easily and

[0]This work was supported in part by the National Science Foundation under Grant CCR-9222734.

rather naturally as LP. In addition, linearity of LP makes it a good candidate to approximate the solution to the wider class of nonlinear optimization problems. Despite its importance in optimization theory, LP has not received adequate attention in parallel and VLSI literatures.

In this paper we present and discuss parallel and VSLI implementation of the Karmarkar interior point algorithm for solving LP. Various partitioning schemes are considered in connection with different architectures and the complexity of these implementations is also analyzed. In addition we give economic interpretations of parallel schemes presented.

1.1 Related Works

The two main computational methods employed to solve LP are the Simplex method, developed by Dantzig in 1963 and Karmarkar's interior point method proposed by Karamarker in 1984. The Simplex method, although highly efficient in practice, was proved to run in exponential time in the worst case. This result has had little effect on the wide acceptance of Simplex as the main method used to solve LP. LP was later proved to be in class P by Khachiyan, but the method developed by him, known as the Ellipsoid method, was shown to be inefficient in practice. Interior point algorithm was developed by Karmarkar as a competitor to Simplex, while being also polynomial in the worst case.

The Simplex algorithm for LP is an inherently sequential algorithm and not easily parallelizable. This is due to the fact that during the course of the algorithm the solution is improved by introducing one basic variable at a time. Nevertheless, Simplex was implemented on an $O(mn \log m \log^3 n)$ chip (in the bit model) using the mesh-of-trees architecture in [2]. In [11], the dual affine version of the interior point method was studied for parallel implementation. In this work, after presenting the algorithm, the authors noted that the main computational task of the method lies in the solving of a system of equations with positive definite symmetric coefficient matrix. In [9], the authors provide certain insights pertaining to the solution of positive definite symmetric (PSD) linear equations arising in the interior point algorithms. As both papers pointed out, the efficient implementation of any version of the interior point method lies in an efficient scheme to solve the PSD system of linear equation in each iteration.

1.2 The Interior Point Algorithm

In this section we briefly describe the original Karmarkar algorithm for the linear programming problem. We then present a simplified version of the algorithm which will be analyzed and implemented in this paper. The motivation for the algorithm and why it works is explained in the original 1984 paper of Karmarkar [6].

Karmarkar first reformulated the general LP, equations 1 and 2 to,

$$\min f(x) = \sum_{j=1}^{n} \log(c^T x / x_j) \tag{4}$$

subject to:

$$Ax = 0; e^T x = n; x \geq 0. \tag{5}$$

where $e = (1, 1, \ldots, 1)^T \in R^n$. He also requires that $Ae = 0$ so that e is a feasible solution. Starting from the initial solution e, the algorithm changes the solution x^k at time k according to the following steps:

1. Construct diagonal matrix D such that $D^{-1}x^k = e$.
2. With this D construct x^k by applying a projective transformation $x^{k+1} = nD^{-1}x^k / e^T D^{-1} x^k$.
3. Go to step 1.

The original interior point algorithm was proposed to prove convergence in polynomial time. In this paper we consider and implement a variation of the interior point method called the primal affine version or the Rescaling algorithm. This is the most direct method among the various versions of the algorithm and brings out the most important computational requirements of the general method without being too concerned with having polynomial worst case running time, although in practice it runs very well.

Let us briefly describe the primal affine version of the interior point method which will be implemented and discussed in this paper. Consider the original formulation of LP in equations 1 and 2. As in the original interior point method, one can formulate the problem such that $x^0 = e$ is an initial feasible solution. At time k the algorithm proceeds as follows:

1. Construct diagonal matrix D from the components of x^k such that $D^{-1}x^k = (1, 1, \ldots, 1) = e$.

2. With this D, compute the projection (not a projective transformation used in the original method) PDc by solving for y in

$$AD^2A^T y = AD^2 c \tag{6}$$

and letting,

$$PDc = Dc - DA^T y \tag{7}$$

3. Determine the number $\tilde{s}$ such that $e - \tilde{s}PDc$ has a zero component.
4. Reduce $\tilde{s}$ by a factor α (usually taken to be 0.96) and call it s, i.e., $s = \alpha\tilde{s}$.
5. Let $x^{k+1} = x^k - sDPDc$.
6. Go to step 1.

Note that $D = diag(d_1, \ldots, d_n)$. As mentioned previously, step number 2, which involves the solution to equation 6, is the most time consuming part of the algorithm and its efficient parallel implementation is the key to an overall efficiency of the parallel interior point method.

2 Architecture and Analysis

In this section we present various architectures that can be employed for the parallel implementation of the primal affine variant of the interior point method. First we discuss the case

where $O(mn)$ processors are available. Mesh-of-Trees architecture [13] has been employed for this case. We will also discuss the case where $O(m)$ processors are available and provide some economic insights for behavior of the parallel scheme.

2.1 Implementation on $O(mn)$ PE's

In this section we present the implementation of the primal affine version of the interior point method discussed in the previous section. The architecture that is used is the mesh-of-trees (MOT) which has certain important features such as an $O(\log n)$ diameter. It has been known that MOT is a powerful configuration for implementing various basic operations like sorting, Jacobi iteration, vector-matrix multiplication, etc., in $O(\log n)$ time [8]. What is new about our presentation is that MOT is also a suitable architecture for the *sequence* of basic operations involved in the primal affine algorithm.

In all subsequent sections it will be assumed that real-valued messages can be transferred along the connections. One can get around this unrealistic assumption by working with a finite precision arithmetic which can be chosen for any LP problem [10], [6]. We will also assume, without loss of generality, that $m \leq n$ (for problems with $m > n$, one can consider the dual of the original problem with $m \leq n$).

Consider an $m \times n$ MOT corresponding to the constraint matrix $A \in R^{m \times n}$. A 2x4 MOT is shown in Figure 1. We will assume that m and n are both powers of two.
For ease of referencing we will use the following notations. $A_{i.}$ and $A_{.j}$ will denote the i-th row and the j-th column of the matrix A, respectively. For referencing the nodes of the MOT we define the following sets. Let $R_i^c(j)$ denote the set of nodes which are the roots at level j in column i. Similarly let $R_i^r(j)$ denote the set of nodes which are the roots at level j in row i. By this notation all the leaf nodes in row i belong to $R_i^r(0)$ and all leaf nodes in column i belong to $R_i^c(0)$. The i-th row root is the only element in $R_i^r(\log n)$ and similarly the i-th column root is the only element in $R_i^c(\log m)$. We note that the cardinality of the set $R_i^r(j)$ is $n/2^j$ and that of $R_i^c(j)$ is $m/2^j$ for all i.

Let us now present the implementation of primal affine algorithm on an $m \times n$ MOT. Initially, x_i, its inverse d_i, and c_i are stored in $R_i^c(\log m)$ (the i-th column root). Then d_i is sent down to the i-th column leaves $R_i^c(0)$. This will take $O(\log m)$ time. To find the projection $(AD)(AD)^T$ and AD^2c we proceed as follows. At the beginning of the step we have $(AD)_{ij}$ stored in $R_i^r(0) \cap R_j^c(0)$. We proceed to form $(AD^2A^T)_{i.}$ on $R_i^r(\log n/m)$. Since $(AD^2AT)_{ii}$ is found by $(AD)_{i.}(AD)_{i.}^T$, one can simply square the entries and sum the square in $O(\log n)$. For $(AD^2A^T)_{ij}$, $i \neq j$, one can sum along the i-th column in $O(\log m)$ time and then sum all the corresponding term in $O(\log n)$ time. If one uses the fact that AD^2AT is symmetric a more efficient algorithm can be found, although not asymptotically better. Similarly AD^2c can be found and be stored in $R_i^r(\log n)$.

Now $c_i d_i$, which is originally stored in $R_i^c(\log m)$ is sent down the i-th column leaf nodes in $O(\log m)$ time. Summing along the row roots we obtain $(AD^2c)_i$ at $R_i^r(\log n)$ in $O(\log n)$

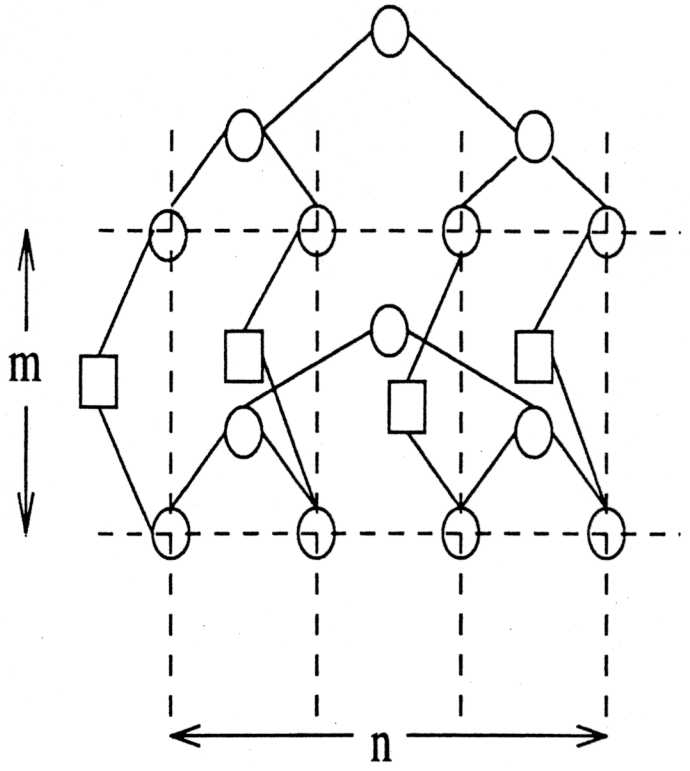

Figure 1: A 2x4 Mesh of Trees

time.

Up to this step of the algorithm, we have obtained the system of equation $(AD^2A^T) = AD^2c$ in $O(\log n)$ time and place them on an $m \times m$ subset of the original $m \times n$ MOT. It can be verified that the diameter of this new subset can be $O(\log n)$. Now one solve the system $AD^2ATy = AD^2c$ in $O(m) + O(\log n)$ by using Gaussian elimination followed by back-substitution using pipelining [8]. We also note that since AD^2A^T is positive definite symmetric matrix, no pivoting strategy is necessary for the LU decomposition. After $O(m)+O(\log n)$ steps, y_i will be stored in $R_i^r(\log n)$ and the computation proceeds as follows.

Having found $y \in R^m$ in $O(m) + O(\log n)$ time and place it such that y_i is in $R_i^r(\log n)$, we find $(AD)^T y$ in $O(\log n)$ steps by sending down y_i to the i-th leaf nodes in $O(\log n)$ and then sum the entries in the i-th column along the i-th column tree in $O(\log m)$ time. Then after $O(\log n)$ steps , the i-th column root $R_i^c(\log m)$ having access to $d_i c_i$ and $((AD)^T)_i$ can obtain PDc_i.

The next step involves determining $\tilde{s}$ such that $e - \tilde{s}(PDc)$ has a zero component. Therefore on can obtain $\tilde{s} = \min_i 1/(PDc)_i$ in $O(\log n)$ time using the fact that sorting on MOT can be done in $O(logn)$ time. Having found $\tilde{s}$, each column root calculates $s = \alpha\tilde{s}$, where α can be taken to be 0.96. Then each column root performs the iteration $x_i^{k+1} = x_i^k - sd_i(PDc)_i$ in $O(1)$.

From the above discussion the total running time for each iteration of the primal affine version of the interior point method is found to be $O(m) + O(\log n)$ using the $O(mn)$ PE's of the $m \times n$ MOT. An examination of the serial algorithm, assuming that only conventional algorithms are available for matrix multiplication (i.e. modolu Strassen's type algorithms), shows that the serial running time is $O(m^2n) + O(n^3)$. Therefore the efficiency is $\Theta(1)$ and our implementation is asymptotically cost-optimal.

Let us now briefly compare our implementation with that obtained for the Simplex method. Each iteration of the primal affine algorithm in our implementation takes $O(m) + O(\log n)$ which is more time consuming than the corresponding pivoting in the Simplex algorithm, which was done in [2] in time $O(\log n)$. But since exponential worst case behavior of the Simplex is absent in the case of the interior point method, our total worst case running time is better than that of obtained in [2], in the worst case.

In view of the fact that the computational efficiency of the iteration of the Karmarkar interior point method is determined by the efficiency of solving the equation 6, which is a system of m equations in m unknowns, it seemed natural to implement the algorithm on an $m \times m$ MOT. We have implemented the algorithm on an $m \times m$ MOT, using block partitioning schemes. Since the implementation is similar to that described above we do not present the details in this paper.

2.2 Implementation on $O(m)$ PE's

It is not always realistic or feasible to assume having access to an $O(mn)$ PE's, especially when the size of the problem is very large. In these situations, one might consider a partitioning of the constraint matrix among $O(m)$ PE's using row partitioning or $O(n)$ PE's using column partitioning. In this section we present this issue for the implementation of the primal affine algorithm on $O(m)$ processors. This discussion also provides a frame work for implementing the algorithm using $O(n)$ PE's by applying the same scheme to the dual of the original LP, which has the transpose of the original matrix A as its constraint matrix. At the end of the section we also provide certain economic insights into the behavior of the parallel version of the interior point algorithm when row partitioning is employed.

2.2.1 Row Partitioning and Its Economic Interpretation

In this section we consider row partitioning of the constraint matrix among m processors. Extensions to block-row partitioning on m/k processors, for some positive divisor k of m, would also be clear from this discussion.

Consider $A \in R^{m \times n}$, and suppose that A_{i*} is known by processor P_i. P_i also has in its memory, a copy of the current feasible solution and a copy of the vector c. At each iteration the parallel algorithm proceeds as follows.

Having access to $(AD)_{i.}$, P_i would like to calculate $(AD^2AT)_{i.}$. First, all processors calculate $(AD^2AT)_{ii}$ which requires no communication and takes $O(n)$ computations. Next, in order to find $(AD^2A^T)_{ij}$, $i \neq j$, the contents of the row j has to be communicated to P_i. To find this inner product, one cannot really do better than sending the vector $(AD)_{j.}$, by the result of Abelson [1]. To find this inner product for all i, we require an all-to-all broadcast of n values, which takes $O(mn)$ on ring, mesh or hypercube [7].

Having found $(AD^2A^T)_{i.}$ in $O(mn)$ steps and store it in P_i we proceed to find $(AD^2c)_i$. This can be done locally at P_i with $O(n)$ arithmetic operations. At this stage, the system of m equations in m unknowns can be solved which has been row partitioned. This system has a positive definite, symmetric coefficient matrix and therefore can be solved using LU or Cholesky factorization and does not require any pivoting strategy. This can be done in $O(n^2)$ time by pipelining the computation and communication on the linear array of processors. Since the linear array can be embedded into a mesh or a hypercube, similar time bound can also be achieved on these architectures [7].

At the end of the previous stage of the algorithm, the distribution of data is shown in Figure 2. We now proceed to compute the projection $PDc = Dc - DA^Ty = Dc - (AD)^Ty$.

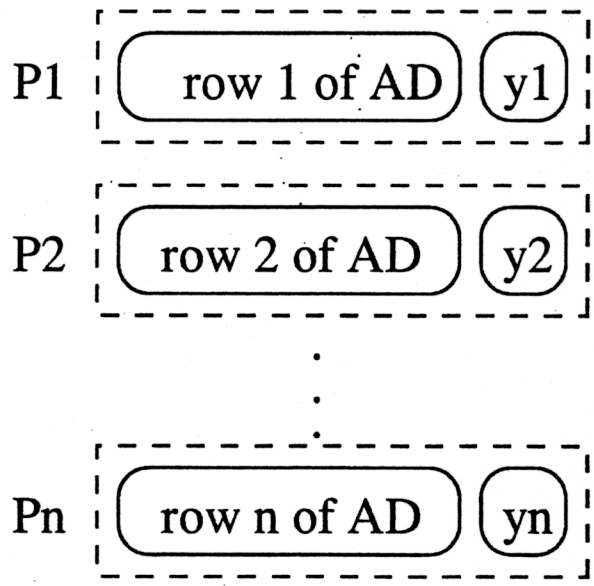

Figure 2: Distribution of the input and y_i's

As Figure 2 demonstrates this step is equivalent to vector-matrix multiplication problem with columns partitioned among the processors (it is rather interesting that we started with a row partitioning and ended up with column partitioning; the reverse situation also holds). Using multi-node accumulation of n/m blocks, taking $O(n)$ time on linear array, mesh and hypercube, each P_i ends up with a block of size n/m of the vector $(AD)^Ty$. Since each P_i has access to Dc, PDc can be found in $O(n/m)$ time with no inter-processor communication necessary. To find the minimum of $(PDc)_i$ (among the positive components only), each P_i finds the minimum in its block (in time $O(n/m \log n/m)$ using for example merge sort) and

then compares the block minimums with others using all-to-all broadcast in $O(m)$. This takes $O(n)$ time. Having $\tilde{s}$ and subsequently s, each block of x^k of size n/m, is updated. At the end of the iteration an all-to-all broadcast of n/m blocks puts us back to where we started at the beginning of the iteration. The total running time of the algorithm on linear array, and subsequently on mesh and hypercube can therefore be bounded by $O(n^2)$. Since m processors are used this parallel implementation is also cost-optimal.

The linear programming problem was originally studied because many problems in production and management could be formulated as LP. There are economic interpretations of the behavior of the Simplex algorithms and terms such as "prices" and "marginal profits" are common in LP literature. Motivated by the ideas related to decomposition principle of Dantzig and Wolfe [5], [4], we now give a brief description of the behavior of the parallel version of the interior point method described above from an economic stand point.

Let us consider P_i as the coordinator of a subdivision of a multi-divisional corporation. P_i only has knowledge about one of the corporate constraints $A_{i.}$ (by constraint we mean something like a minimum production level to keep the overall corporation running or the maximum expenditure allowed on any given day). Each component x_i can be considered as the production level of item i, and the corresponding component of the vector c, i.e., c_i, can be considered as the cost (in which case we want to minimize) or profit (in which case we want to maximize) of producing one unit of item i. Each coordinator can suggest a production level by looking at his/her own constraint. But since there are $m-1$ more constraints on the problem which the coordinator has no knowledge of, each coordinator has to take into account other coordinators' constraints by projecting them into his own constraint set.

More specifically, consider an iteration of the parallel implementation of the interior point method. Starting from the current production level x_k, each coordinator initially scales the constraint set by the current production level. Then through a tele-conference, each coordinator suggests a change of policy in the direction of his/her own constraint vector. How this suggestion is used in the final decision depends on the coefficient y_i that each coordinator attaches to his/her suggestion. This corresponds to the equation 7. To come up with the coefficient y_i, the coordinator P_i not only relies on his/her own constraint $AD_{i.}$, but also on other coordinators constraints $AD_{j.}$, $j \neq i$, by projecting them onto $AD_{i.}$.

At the end of the above tele-conferencing between the coordinators, a mutual direction of change is found and each division updates the production level. Finally, the tele-conferencing stops when an optimal production level is reached.

A more elaborate description of the behavior of the algorithm can also be given for the column partitioning scheme. We shall present this in the future work.

3 Conclusion

We have presented various implementations and the related issues pertaining to the parallel and VLSI implementation of the interior point method for solving the linear programming problem. Implementing the algorithm on mesh-of-trees architecture with $O(mn)$ processors was described. We also discussed the problem of decomposition along the rows of the constraint matrix and the efficiency of this implementation. Certain economic interpretation of the behavior of the parallel version of the interior point method was given which might be useful for designing more efficient parallel algorithms for solving LP.

There are many directions along which this research can be continued. One promising direction is to employ iterative methods for solving the positive symmetric system of equation 6 and their efficient parallel implementation on VLSI. Karmarkar himself pointed out that since an exact projection is not required for the algorithm to converge, one might be able to use iterative methods such as Jacobi or Gauss-Seidel, or algorithms such as the conjugate gradient method, to improve the efficiency of the implementation [6]. In these cases the stopping criterion for the iteration to guarantee convergence has to be studied in greater detail.

On the more theoretical side, it is of interest to obtain certain bounds on the communication complexity of solving system of linear inequalities. In particular, one can consider the following problem: Having two processors, each having access to two different inequalities, $Ax \geq b_1$ and $Bx \geq b_2$, determine in a distributed manner whether or not the set determined by both inequalities is empty with minimum amount of communication.

References

[1] Abelson, H., "Lower Bounds on Information Transfer in Distributed Computations," in *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, 1978.

[2] Bertossi, A.A., Bonuccelli, M.A., "A VLSI Implementation of the Simplex Algorithm, *IEEE Transactions on Computers*, Vol. C-36, No.2, Feb. 1987, pp. 241-247.

[3] Bertsekas, D.P., Tsitsiklis, J.N., *Parallel and Distributed Computation*, Prentice Hall, 1989.

[4] Chvatal, V., *Linear Programming*, Freeeman, New York, 1983.

[5] Dantzig, G.B., Wolfe, P., "Decomposition Principle for Linear Programs," *Operations Research*, 8, 1960, pp.101-111.

[6] Karmarkar, N., "A New Polynomial-Time Algorithm for Linear Programming", *Combinatorica*,4, pp. 373-395, 1984.

[7] Kumar, V., Grama, A., Gupta, A., Karypis, G., *Introduction to Parallel Computing*, The Benjamin-Cummings Publishing Co, Inc., 1994.

[8] Leighton, F.T., *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann, San Mateo, California, 1992.

[9] Lustig, I.J., Marsten, R.E., Shanno, D.F., "The Interaction of Algorithms and Architectures for Interior Point Methods," in *Advances in Optimization and Parallel Computing*, P.M. Pardalos (ed.), North-Holland, 1992.

[10] Papadimitriou, C.H., Steiglitz, K., *Combinatorial Optimization*, Prentice-Hall, 1982.

[11] Saltzman, M.J., Subramanian, R., Marsten, R.E., "Implementing an Interior Point LP Algorithm on a Supercomputer," in *Impacts of Recent Computer Advances on Operations Research*, R. Sharda, B. Golden, E. Wasil, O. Balci, W. Stewart (eds.), Elsevier Science, New York, 1989.

[12] Strang, G., *Linear Algebra and its Applications*, Third edition, Harcourt Brace Jovanovich, 1988.

[13] Ullman, J., *Computational Aspects of VLSI*, Computer Science Press, 1984.