ON THE PARALLEL AND VLSI IMPLEMENTATION OF THE INTERIOR POINT ALGORITHMS

M. Mesbahi*
Jet Propulsion Laboratory
California Institute of Technology

G. P. Papavassilopoulos† and V. K. Prasanna‡
University of Southern California
Department of EE–Systems

Abstract

We discuss several aspects of the parallel and the VLSI implementation of the interior point algorithms for solving the linear programming problem. Various architectures, based on the different partitioning of the input data, are proposed and the complexity of these implementations are discussed. An economic interpretation of a partitioning scheme is then presented which might prove useful for the further improvements in the efficient parallelization of the interior point methods.

Keywords: Interior Point Methods; Linear Programming; Parallel Computing; VLSI

1 Introduction

We consider certain issues pertaining to the parallel and the VLSI implementation of the interior point methods (ipms) for solving the linear pro-

*Corresponding author; E-mail: mesbahi@hafez.jpl.nasa.gov; address: 4800 Oak Grove Dr., M/S: 198-326; Pasadena, CA 91109-8099

†E-mail: yorgos@bode.usc.edu; address: 3740 McClintock Ave., Los Angeles, CA 90089-2563.

‡E-mail:prasanna@halcyon.usc.edu; address: 3740 McClintock Ave., Los Angeles, CA 90089-2562.

gramming problem (LP):

$$\min_x c^T x \tag{1.1}$$

$$\text{subject to: } Ax = b, \tag{1.2}$$

$$x \geq 0, \tag{1.3}$$

where the matrix $A \in R^{m \times n}$ is of full row rank ($m < n$). LP is one of the most important problems in the optimization theory. This is mainly due to the wide array of problems in operations research and combinatorial optimization which can be formulated as an LP. Moreover, solving an LP comes up as a "subroutine" in several algorithms proposed for solving nonlinear programming problems. Indeed, fast LP solvers are of immense importance in the development of fast computational procedures for many optimization problems.

The simplex algorithm, developed by Dantzig [5] more than forty years ago, is still a widely used algorithm for solving LPs. Two other methods for solving LPs include the Ellipsoid algorithm due to Khachian [10] and the interior point method, originated by the paper of Karmarkar [9]. Both methods were developed in the search of *polynomiality* of the worst case running time, a property which is absent in the case of the simplex algorithm. The Ellipsoid method has not yielded an efficient algorithm in practice, where as, it is now clear, that the Karmarkar's algorithm can be made to be very efficient in practice, especially when the problem size increases above some thousands of variables. Motivated by the work of Karmarkar, many interior point algorithms, most of which differ significantly from that of Karmarkar's, have been proposed for solving LPs and the more general nonlinear convex programming problems [14]. We shall focus on a variant of ipms, known as the affine scaling algorithm (ASA) and discuss various issues pertaining to its parallel and VLSI implementation. The ASA is one of the simplest version of the family of ipms that readily brings out the main computational tasks involved in the more general ipms.

Related to the present note are the works of Bertossi and Bonuccelli [2], Lustig *et al.* [13], and Saltzman *et al.* [16]. In [2] a VLSI implementation of the simplex algorithm was presented on a $O(mn \log m \log^3 n)$ chip (in the bit model) using the mesh-of-trees architecture. In [13] and [16] the dual affine version of the ipm was studied, where the authors concentrate on parallelization of the cholesky factorization for solving the positive semi-definite system of linear equation that comes up at each iteration of most LP interior point solvers.

The organization of the paper is as follows. We shall first describe the

affine scaling algorithm (ASA). The VLSI implementation of the ASA is then presented on the mesh-of-trees (MOT) architecture, along with the corresponding complexity analysis. Implementation of the ASA is then discussed on $O(m)$ processors, in addition to some economic interpretation of the behavior of this implementation. The conclusion contains some suggestions along which this work can be continued.

2 Algorithm and Implementation

We shall present the ASA variant of the ipms, along with various architectures that can be employed for its parallel implementation. First we discuss the case where $O(mn)$ processors are available. The mesh-of-trees architecture (MOT) [18] has been employed in this context. We also discuss the case where $O(m)$ processors are available. Finally we proceed to provide an economic insight for the behavior of the proposed parallel scheme.

The algorithm (ASA) which will be implemented in this paper can be described as follows. We shall assume that an initial point $x^1 > 0$ is available such that $Ax^1 \geq b$. For $k \geq 1$, the algorithm proceeds as below:

The Affine Scaling Algorithm (ASA)

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

2. Compute the projection PDc by solving for y in the equation:

$$AD^2A^Ty = AD^2c \tag{2.1}$$

   and letting,

$$PDc = Dc - DA^Ty \tag{2.2}$$

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 ($s = \alpha\tilde{s}$).

5. Let $x^{k+1} = x^k - sDPDc$.

6. Let $k = k + 1$ and go to Step 1.

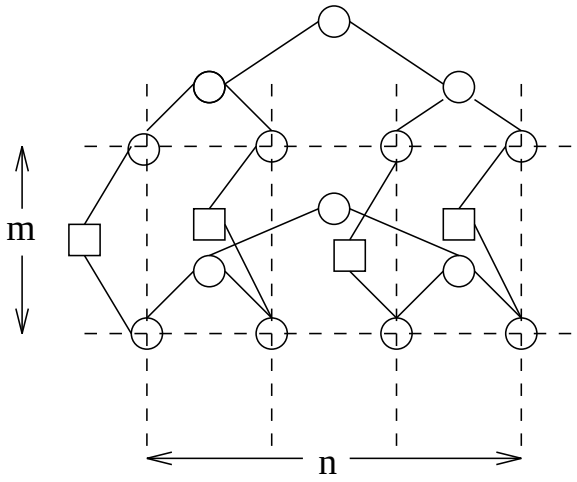

Figure 1: A 2x4 Mesh of Trees

2.1 Implementation on $O(mn)$ Processing Elements

In this section we present an implementation of the ASA described above. The architecture that is employed 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 [12]. What is novel about our implementation is that it demonstrates the suitability of the MOT architecture for the *sequence* of basic operations involved in the ASA.

For the purpose of this paper we shall assume that real-valued messages can be transferred between the processing elements. One can get around this assumption by working with finite precision arithmetic which can be chosen in advance for an instance of LP [9], [15]. We shall also assume, without loss of generality, that both m and n are powers of two.

Consider an $m \times n$ MOT corresponding to the constraint matrix $A \in R^{m \times n}$. A 2×4 MOT is shown in Figure 1. 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, for all i, the cardinality of the set $R_i^r(j)$ is $n/2^j$, and that of $R_i^c(j)$ is $m/2^j$.

Let us now present the implementation of the ASA on an $m \times n$ MOT. Initially, x_i^k, its inverse d_i^k, 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 one proceeds 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 squares 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 the product $c_i d_i^k$, 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 $O(\log n)$ time.

We have now obtained the system of linear equations $(AD^2A^T) = AD^2c$ in $O(\log n)$ time and placed 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 is $O(\log n)$. Now we proceed to solve the system $AD^2A^Ty = AD^2c$ in $O(m) + O(\log n)$ by using Gaussian elimination followed by back-substitution using pipelining [12]. 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 $R_i^r(\log n)$.

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 $(AD)^Ty$ 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. Subsequently, after $O(\log n)$ steps , the i-th column root $R_i^c(\log m)$ having access to d_ic_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 $O(\log n)$ 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 time $O(1)$.

From the above discussion the total running time for each iteration of the ASA is found to be $O(m) + O(\log n)$ using the $O(mn)$ PEs of the $m \times n$

MOT. An examination of the serial algorithm, assuming that only conventional algorithms are available for matrix multiplication (i.e. those requiring $O(n^3)$ arithmetic operations), 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.

In view of the fact that the computational efficiency of each iteration of the ASA is determined by the efficiency of solving the equation (2.1), which is a system of m equations in m unknowns, it seemed natural to implement 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 the present paper.

2.2 Implementation on $O(m)$ PEs

It is not always realistic or feasible to assume having access to an $O(mn)$ PEs, especially when the size of the problem is very large. In this case, one might consider a partitioning of the constraint matrix A among $O(m)$ PEs using row partitioning, or $O(n)$ PEs using column partitioning. In this section we consider this issue for the implementation of the ASA on $O(m)$ processors. This discussion also provides a framework for implementing the algorithm on $O(n)$ PEs 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.

We consider the row partitioning of the constraint matrix among m processors. Extensions to the block-row partitioning of the constraint matrix A on the m/k processors, for some positive divisor k of m, will also be evident from this discussion.

Consider the constraint matrix $A \in R^{m \times n}$, and suppose that $A_{i.}$ is known by the 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 $(AD^2A^T)_{i.}$. First, all processors calculate $(AD^2A^T)_{ii}$ which requires no communication and takes $O(n)$ computations. Next, in order to find $(AD^2A^T)_{ij}$, $i \neq j$, the contents of row j have to be communicated to P_i. To find this inner product, one cannot really do better than sending the vector $(AD)_{j.}$, due to a 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 a ring, mesh or hypercube architectures [11].

Having found $(AD^2A^T)_{i.}$ in $O(mn)$ steps and stored it in P_i we proceed to find $(AD^2c)_i$. This can be done locally at each P_i with $O(n)$ arithmetic

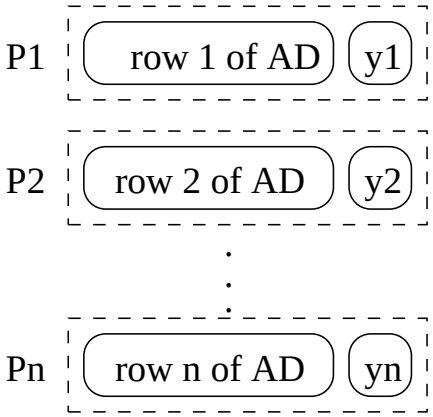

Figure 2: Distribution of the input and y_i's

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 without requiring 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 in a mesh or a hypercube, similar time bounds can also be achieved on these architectures [11].

At the end of the previous stage of the algorithm, the distribution of data can be depicted as in Figure 2. We now proceed to compute the projection $PDc = Dc - DA^T y = Dc - (AD)^T y$. As shown in Figure 2, this step is equivalent to the vector-matrix multiplication problem with a 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 the n/m blocks, taking $O(n)$ time on the linear array, mesh or hypercube, each P_i ends up with a block of size n/m of the vector $(AD)^T y$. Since each P_i has access to Dc, PDc can be found in $O(n/m)$ time with no inter-processor communication. To find the minimum of $(PDc)_i$ (among the positive components only), each P_i finds the minimum in its own block (in time $O((n/m)\log(n/m))$ using for example merge sort) and then compares the block minimums with the minimum of the other blocks by employing an all-to-all broadcast in $O(m)$ time. The complete process takes $O(n)$ time. Having $\tilde{s}$ and subsequently s, each block of x^k of size n/m is then 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.

2.3 An Economic Interpretation

The origins of LP are problems in the production planning and management sciences [5], [7], [8]. There are various economic interpretations of the behavior of the simplex algorithm and terms such as "prices" and "marginal profits" are common in the LP literature. Motivated by the ideas related to decomposition principle of Dantzig and Wolfe [4], [6], we now discuss an economic interpretation of the behavior of the parallel version of the ASA on the $O(m)$ processors discussed earlier.

Let us consider the i-th processor, P_i, as the coordinator of a subdivision of a multi-divisional corporation. P_i only has knowledge of one of the corporate constraints $A_{i.}$ (by a constraint we have in mind 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 the 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 the item i. Each of the coordinators can suggest a production level by examining his/her own constraint. But since there are $m-1$ more constraints on the problem, which an individual coordinator has no knowledge of, each coordinator has to take into account the other coordinators' constraints by "projecting" his/her decision on the set defined by the others.

More specifically, consider an iteration of the parallel implementation of the ASA. 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 "corporate" decision depends on the weight y_i which each coordinator attaches to his/her suggestion. This corresponds to the equation (2.2). To come up with the weight y_i, the coordinator P_i not only relies on his/her own constraint $AD_{i.}$, but also on the other coordinators constraints $AD_{j.}$, $j \neq i$.

At the end of the tele-conference between the coordinators, a mutually agreed upon direction of change is found and each division updates the

corresponding production level. Finally, the tele-conference is terminated when an optimal production level is reached.

3 Conclusion

We have presented various issues pertaining to the parallel and VLSI implementation of the affine scaling version of the interior point methods for solving the linear programming problem. Implementation of the algorithm on the 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. An economic interpretation of the behavior of the parallel version of the interior point method was also presented which might be useful for designing more efficient parallel LP solvers.

There are several directions along which this work can be continued. One promising direction is to employ iterative methods for solving the linear system of equations (2.1) and their efficient parallel implementation on the VLSI.

Acknowledgments

The research of M. Mesbahi was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautic and Space Administration. The research of G. P. Papavassilopoulos was supported in part by the National Science Foundation under Grant CCR-9222734.

Biographies

M. Mesbahi obtained his Ph.D. from USC's Department of EE-Systems in July of 1996. Since then, he has been with the Jet Propulsion Laboratory, California Institute of Technology. His research interests are system and control theory, spacecraft dynamics, optimization theory and algorithms, and parallel computation.

G. P. Papavassilopoulos is a Professor of Electrical Engineering in the Department of EE-Systems at USC. His research interests include control theory, optimization theory and algorithms, and game theory.

V. K. Prasanna (V. K. Prasanna Kumar) is a Professor of Electrical Engineering in the Department of EE-Systems at USC and serves as the Director of the Computer Engineering Division. His research interests include parallel computation, computer architecture, VLSI computations, and high performance computing for signal and image processing, and vision. Dr. Prasanna is a Fellow of the IEEE.

References

[1] H. Abelson, "Lower Bounds on Information Transfer in Distributed Computations," *Journal of ACM*, *27* (2), 384–392, 1980.

[2] A. A. Bertossi, M. A. Bonuccelli, "A VLSI Implementation of the Simplex Algorithm," *IEEE Transactions on Computers*, *C–36* (2), 1987, 241–247.

[3] D. P. Bertsekas, J. N. Tsitsiklis, *Parallel and Distributed Computation*, (Englewood Cliffs, New Jersey: Prentice Hall, 1989).

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

[5] G. B. Dantzig, *Linear Programming and Extensions*, (Princeton, New Jersey: Princeton University Press, 1963).

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

[7] J. H. Greene, K. Chatto, C. R. Hicks and C. B. Cox, "Linear Programming in the Packing Industry," *Journal of Industrial Engineering*, *10*, 364-372, 1959.

[8] L. V. Kantrovich, "Mathematical Methods in Organization and Planning of Production," *Management Science*, *6*, 366-422, 1960.

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

[10] L. G. Khachian, "A polynomial algorithm in Linear Programming," *Soviet Mathematics Doklady*, *20*, 191-194, 1979.

[11] V. Kumar, A. Grama, A. Gupta and G. Karypis, *Introduction to Parallel Computing*, (Redwood City, California: The Benjamin-Cummings Publishing Co, Inc., 1994).

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

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

[14] Y. Nesterov, A. Nemirovskii, *Interior-Point Polynomial Algorithms in Convex Programming*, (Philadelphia: SIAM, 1994).

[15] C. H. Papadimitriou, K. Steiglitz, *Combinatorial Optimization*, (Englewood Cliffs, New Jersey: Prentice Hall, 1982).

[16] M. J. Saltzman, R. Subramanian and R. E. Marsten, "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 (editors), (New York: Elsevier Science, 1989).

[17] G. Strang, *Linear Algebra and its Applications*, (San Diego, California: Harcourt Brace Jovanovich, 1988).

[18] J. Ullman, *Computational Aspects of VLSI*, (Rockville, Maryland: Computer Science Press, 1984).