TR/15/91 **October 1991**

Adapting the Interior Point Method for the Solution of Linear Programs on High Performance Computers

J. Andersen, R. Levkovitz and G. Mitra

z1577434

CONTENTS

Adapting the Interior Point Method for the Solution of Linear Programs on High Performance Computers

J. Andersen, R. Levkovitz and G. Mitra
Brunel - The University of West London, U.K.

1.Abstract

In this paper we describe a unified algorithmic framework for the interior point method (IPM) of solving Linear Programs (LPs) which allows us to adapt it over a range of high performance computer architectures. We set out the reasons as to why IPM makes better use of high performance computer architecture than the sparse simplex method. In the inner iteration of the IPM a search direction is computed using Newton or higher order methods. Computationally this involves solving a sparse symmetric positive definite (SSPD) system of equations. The choice of direct and indirect methods for the solution of this system and the design of data structures to take advantage of coarse grain parallel and massively parallel computer architectures are considered in detail. Finally, we present experimental results of solving NETLIB test problems on examples of these architectures and put forward arguments as to why integration of the system within sparse simplex is beneficial.

2. Sparse Simplex and Interior Point Method: Hardware Platforms

Progress in the solution of large LPs has been achieved in three ways,namely hardware, software and algorithmic developments. Most of the developments during the 70's and early 80's in the Sparse Simplex method were based on serial computer architecture. The main thrust of these developments were towards exploiting sparsity and finding methods which either reduced simplex iterations or reduced the computational work in each iteration [BIXBY91, M1TAMZ91]. In general these algorithmic and software developments of the sparse simplex method cannot be readily extended to parallel computers. In contrast the interior point methods which have proven to be robust and competitive appear to be better positioned to make use of newly emerging high

performance computer architecture. The relative advantages of using IPM over sparse simplex in exploiting these architectures are summarised below. A few researchers [FORTOM90.PARPRS90] have identified difficulties involved in adapting sparse simplex algorithms for parallel computers. Although a number of implementations have been reported [STUNRO88,CHNLNS90], the only credible and robust implementation is that due to Forrest and Tomlin [FORTOM90]. Our profiling information Fig 1.1 and 1.2 for some well known test problems from the Netlib collection show that the main computational work is spread over a number of algorithmic sub-components such as PRICE, BTRAN, FTRAN etc

The relative computational efforts in these procedures vary from model to model. Through some ingenuity and data reorganisation the PRICE procedure has been redesigned for parallelism [FORTOM90] and shows good speed up. The speed up in the other algorithmic procedures are not of the same order. If we take into account AMDAHL'S law [AMDHL67] then we can appreciate how the significant computational effort of the serial part of the logic imposes a fairly modest limit on the scope of speed up. Essentially we cannot easily adapt the simplex method for parallel computation because of the indirect address list structure used to represent sparse matrices and vectors. Whereas in serial machines this representation reduces total number of operations, in parallel machines it markedly slows down processing. Even hardware scatter and gather instructions do not fully cope with the problem of representing sparse data on parallel machines. Parallel machine architectures in general are well suited for dense matrix and vector processing. All variants of IPM share the same computational characteristics: the number of iterations is usually very low, typically less than 100, and the algorithmic steps require a repeated construction and factorization of a Sparse Symmetric Positive Definite (SSPD) system of equations with a fixed non-zero structure. Our profiling information Fig 1.3 clearly illustrates that most of the computational work takes place in the construction of an SSPD matrix and the solution of the resulting system by a direct method such as Choleski factorization or an indirect method such as conjugate gradient. This concentration of computational effort makes IPM well suited for exploiting parallel algorithmic paradigms.

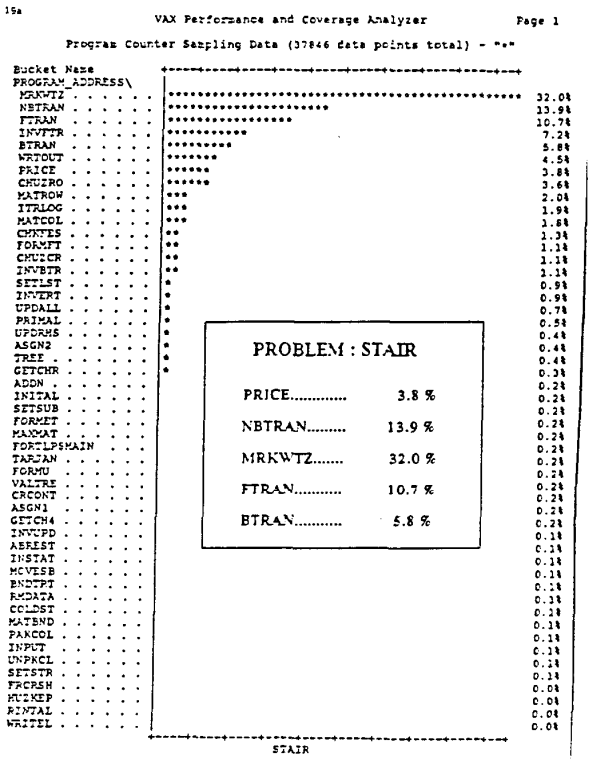


Fig 1.1

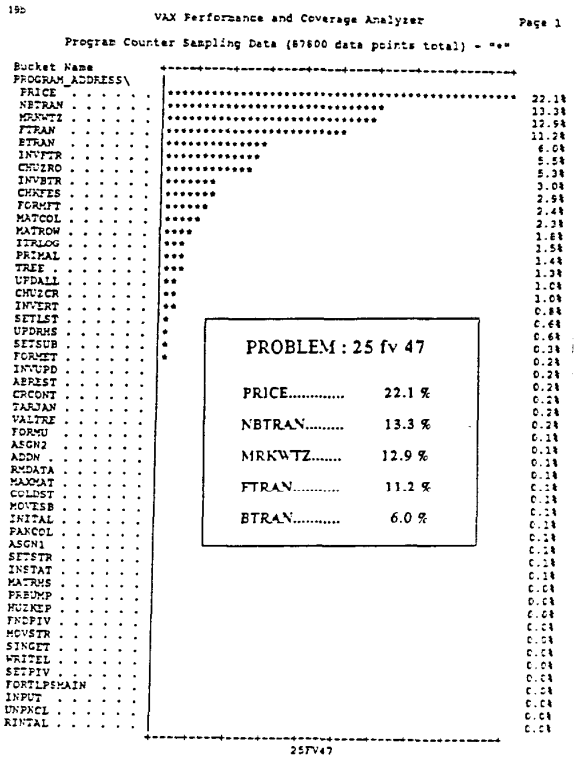


Fig 1.2

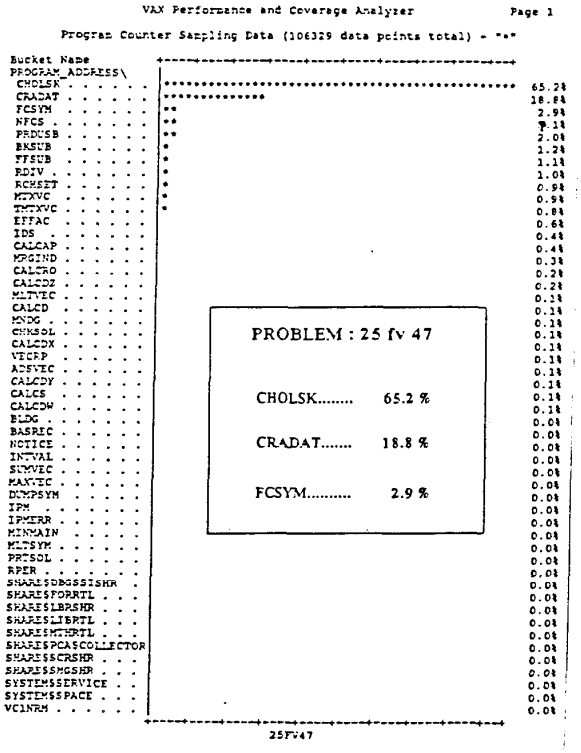


Fig 1.3

The specialists in sparse matrix computation have sharpened the computational methods for solving SSPD systems on parallel computers [DFERED86, GEOLIU81, ASGRLW87] and this has also added to the advantage of adapting IPM on parallel machines. For instance, the use of elimination trees, identification of supernodes and loop unrolling for vector (parallel) machines are well established and well understood [LIU89,LUMASH91]. It is therefore no coincidence that high performance IPM optimization systems incorporate software design which exploit their respective hardware platforms. For instance KORBX system is designed especially for the Alliant 8 processor parallel computer, IBM's OSL is designed for the RS6000 and 3090 computers only; even OBI, otherwise a portable system, is specially tuned for the Cray YMP [BXGLMS91]. Our research interests on the other hand lie in adapting IPM for a range of parallel computing architectures and finding efficient ways of integrating these algorithms with our simplex solver. For our hardware platforms, we have chosen the transputer based Distributed Memory Computer (DMC) and an array processor (AMT-DAP). In this report, we focus on the adaptation of the SSPD solver to these hardware platforms. The rest of the paper is set out as follows: in section 3 we describe the IPM algorithm, in sections 4 and 5 we discuss the DMC and the DAP implementations with the corresponding experimental results. Finally, in section 6, we analyze the computational results and consider the cross-over to simplex strategy.

3.Choice of Interior Point Method.

Among the various IPMs that were suggested and implemented recently, the group of primal-dual type algorithms have emerged as most promising. The framework for the primal-dual path following IPM was introduced by Megiddo in 1986 [MONADL89]. This algorithm solves the following primal and dual problems simultaneously.

$$\begin{aligned} &\textit{Primal:}\quad \min c^T x && \textit{Dual:}\quad \max b^T y \\ &\text{s.t. } Ax = b,\ x \geq 0 && \text{s.t. } A^T y + z = c,\ z \geq 0 \qquad (3.1) \end{aligned}$$

$$A \in R^{m*n}, b, y \in R^m, c, z, x \in R^n$$

The primal-dual algorithm converges to the optimal solution in at most $O(n^{1/2}L)$ iterations [MONADL89] where n denotes the dimension of the problems and L the input size. It

computes both primal and dual intermediate solutions at any stage; this ensures that the retrieval of an optimum extreme point from the optimal primal and dual solutions can be done in strongly polynomial time [MGIDD091]. Three variants of the primal-dual algorithm were implemented namely, the primal-dual affine [MONADL89], primal dual barrier [LUMASH90] and recently the primal dual power series algorithm (predictor corrector)[LUMASH90,BXGLMS91]. All three variants solve the LP problems by minimizing the complementarity gap (optimization step), but while the affine algorithm computes an optimizing step only, the barrier method calculates a combined optimizing and centralizing step which also keeps the solution away from the boundaries. The power series algorithm computes an optimizing step as in the affine algorithm (predictor step) and then the centralizing steps (correcting steps). In algorithm 3.1 we present a pseudo code of the primal dual barrier algorithm.

Algorithm 3.1: Primal-Dual Barrier

PD1. Construct the phase 1 extended problems. Find initial solution for x, y, z.

PD2. Let X be a diagonal matrix of x, Z be a diagonal matrix of z, set. $D = XZ^{-1}$.

PD3. Let $\rho(\mu)$ *be a compound (centralising and advancing) function,* μ *the centralising parameter.*

PD4. Find the new search direction for y: $\dot{y}$

compute : $M = ADA^T$

compute : $\dot{y} = M^{-1}AD\ \rho(\mu)$

use $\dot{y}$ *to compute the search direction for x,z :* $\dot{x}$, $\dot{z}$.

PD5. Make a step in the computed direction $x,y,z \leftarrow x + \alpha(x)x, \dot{x} + \alpha(y)\dot{y}, z, + \alpha(z)\dot{z}$

PD6. If end conditions are met, stop. Else go to step PD2.

Although the predictor corrector algorithm performs better than the other two variants, all primal dual algorithms are computationally dominated by the calculation of the affine trajectory in which a system involving a new SSPD matrix M is created and solved (step PD4).In the subsequent sections we discuss the implementation of this step first on the DMC and then on the DAP.

4.Parallel SSPD Solver Kernel on a Distributed Memory Computer

Our parallel SSPD solver kernel is implemented on a transputer based DMC. The DMC

computer is formed by a grid of independent powerful processors, each one having a local memory and communication channels. As there is no shared memory, all communication between processors is broadcast through these channels. We use the transputer based DMC because transputer hardware is relatively compact, cheap, well supported and can be configured to different topologies easily. For the algorithm stated below, we chose the binary tree grid topology as the most suitable one. To solve the SSPD system of equations in step PD4 we employ the Choleski distributed parallel algorithm (CDP), an extension of the well known sparse Choleski factorization algorithm [GEOLIU81] and presented in algorithm 4.1. The CDP algorithm analyses the sparsity structure of the symmetric matrix and uses the row dependencies to create parallel elimination sequences. In designing this algorithm we have taken advantage of the special IPM property that the non zero structure of the symmetric matrix remains invariant throughout the iterative steps. Thus, structuring and allocating sets of rows taken from the matrix and distributing to different processors are done once whereas only the remaining steps are repeated in every IPM iteration (steps CDP7-CDP11). As the structuring phase occurs only once, the overhead of computational effort - mostly invested in analyzing M to identify a sequence of semi independent sets of rows - proves to be worthwhile.

Algorithm 4.1: Choleski Distributed Parallel Factorization

*CDP*1. *Find a permutation matrix P to minimize the fill in..* $M' = PMP^T$ *,*$t' = Pt$

*CDP*2. *Find sets of indistinguishable rows.*

*CDP*3. *Build elimination tree for the rows and rebalance it.*

*CDP*4. *Partition the rows of the matrix into k subsets,* $\mathrm{R}_1, \mathrm{R}_2, ..., \mathrm{R}_k$ *, and allocate them to the k processors* $\mathrm{P}_1, \mathrm{P}_2, ..., \mathrm{P}_k$ *respectively.*

*CDP*5. *Broadcast the A matrix and processor allocation table over the transputer network.*

*CDP*6 *Factorize partitions of the symmetric matrix M' on the transputers such that*

$$M'\left(\bigcup_{i=1}^{k} R_i\right) = U'^{T}\left(\bigcup_{i=1}^{k} R_i\right) U'\left(\bigcup_{i=1}^{k} R_i\right),\ M'(R_i), U'(R_i) \in R^{|R_i| x m}.$$

*CDP*7 *Broadcast the diagonal matrix D and the vector r over the transputer network.*

*CDP*8 *Compute the numeric factorization of the matrix* $\mathrm{U}'(R_i)$ *using local and communicated data.*

*CDP*9. *Set* $U'^{T}(R_i)\dot{y}'(R_i) = d'(R_i)$

*CDP*10. *Solve for* $d'(R_i) : U'(R_i)d'(R_i) = t'(R_i)$ *(using backward substitution).*

*CDP*11. *Solve for* $\dot{y}'(R_i) : U'^{T}(R_i)\ddot{y}(R_i) = d'(R_i)$ *(using forward substitution).*

The analysis of the symmetric matrix is based on five main concepts broadly concerned with sparsity preservation and data mapping. Sparsity preservation is achieved by a symmetric permutation (P^TMP) which reorders the rows and columns of the matrix M (step CDP1). This reordering is carried out by using the minimum degree heuristic [GEOLIU81]. The ordering of the matrix determines the sequence which in turn fixes the elimination hierarchy. Next, we make use of the properties of indistinguishable rows [GEOLIU81]. These rows become indistinguishable by having the same non zero structure during some stages of the elimination process. These rows are identified, collected together as super nodes and later assigned to the same processor (step CDP2). After determining the super nodes we identify the parallel hierarchy structure of the elimination process by constructing the elimination tree (step CDP3) [LIU89]. The elimination tree T(U) of the Choleskifactor U of the SSPD matrix M is defined in the following way:

Elimination Tree T(U)

$$\textit{A row } r_i \textit{ is the parent of row } r_j, i > j \textit{ iff } i = \min \{k, u_{jk} \neq 0 \; k > j\} \qquad (4.2)$$

A row r_j is a root if no such i exists (hence r_j cannot have a parent)

The elimination tree can be interpreted as a communication tree for the rows of the matrix. All communication during the CDP factorization is done strictly through the branches of the elimination tree. We use the elimination tree to map row subsets of the matrix to the binary tree transputer grid. This mapping is achieved by a simple visiting heuristic which travels through the elimination tree in a top to bottom fashion and identifies the branches where the elimination workload can be divided into roughly equal parts (step CDP4). Finally, the algorithm determines the life span of each row (with respect to the partitioning). The life span of a row is defined below:

Let r_s denote the s^{th} row of the ordered matrix M. we define the Home Processor $HP(r_s)$ and the End Processor $EP(r_s)$ respectively as :

$$\textit{Home Processor : } HP(r_{s)} = P_i, r_s \in R_i, \; R_i \textit{ is allocated to } P_i \textit{ (see CDP4)} \qquad (4.3)$$

$$\textit{End Processor : } EP(r_{s)} = P_{j,} \textit{ where } j = \min \{l \mid r\xi \in R_l, u_{sq} \neq 0, \; s < q, \; l = 2,....,k\} \; @$$

A row r_s and all related information (backward and forward substitution) is communicated between $HP(r_s)$ and $EP(r_s)$ only. We define the life span of the row r, as the tree path between $HP(r_s)$ and $EP(r_s)$. All communication involving the row r_s is limited to this path, hence the length of this path is a useful tool to control and analyze the communication during the elimination process.

After partitioning the matrix, we broadcast the original problem data over the transputer grid (step CDP5). Each processor P_i retains only the necessary information for the row subset R_i, the symbolic factorization is then carried out on each transputer individually (step CDP6). In the iterative stage (steps CDP7-CDP11), the new diagonal matrix and the right hand side vector are broadcast globally at every iteration; the local solutions are gathered and transmitted to the root processor which in turn checks the termination criteria and computes the values for the next step if necessary.

The IPM using the CDP kernel was implemented on a 16 transputer DMC by using the Top-Express Fortran compiler. In table 4.1 we set out six NETLIB [GAY85] test problems and their derived characteristics. Relevant statistics covering tree information, solution time on single processor and 15 processors configurations are also summarized. The tree average path length is defined as the sum of lengths of all paths from the leaves to the root divided by the number of leaves. The ratio (*average path/number of rows*) gives a good indication of inherent parallelism, as the worst case tree structure is a simple list (see problem GROW22). For a more detailed description of the algorithm, the reader is referred to our extended report on the subject.

Table 4.1

								TREE		OPTIMAL SOLUTION			SPEED-UP
				SYMMETRIC	U FACTOR	INDISTIN.			AVERAGE		TIME (sec)		
PROBLEM	ROWS	COLS	NZ	ELEMENTS	ELEMENTS	SETS	ROWS	LEAVES	PATH	ITER.	1PC	15PC	1PC/15PC
GANGES	1310	1681	6912	16621	31664	43	739	351	158	35	910	210	4.33
25FV47	822	1571	10400	22697	35053	53	607	174	188	43	1892	559	3.38
SCTAP3	1481	2480	8874	16240	18811	32	294	620	107	25	300	75	4.00
SHIP12L	1152	5427	16170	23338	12219	27	137	828	35	27	324	54	6.00
CRE_A	3517	4067	19054	44835	36185	60	348	1356	88	35	805	175	4.60
GROW22	440	946	8252	5040	9030	0	0	1	440	27	97	122	0.80

5. The SSPD Solver Kernel on the DAP Computer

The array processor used for this study is the massively parallel AMTDAP610.This model has 4096 simple 1 bit processors organised in a 64 by 64 grid. Each processor has local memory, and can be enhanced by an 8 bit floating point processor. The computational grid defines a fixed communication pattern of rows and columns along which the inter-processor communication is most effective and faster than floating point computations. The processors can either execute a single common instruction in parallel or remain idle. As the speed is achieved through a large number of processors, an effective parallel algorithm must distribute the data over the processor grid uniformly. This computational regime is also known as "fine grain" parallel processing and its application to unstructured sparsity problems presents a special challenge. The explicit data dependencies of the direct method requires representation by list structures and the corresponding algorithm channels most of the numerical computation in a narrow stream. We have decided to use an alternative data representation to avoid this problem. This consideration has promoted us to apply an iterative scheme for solving the SSPD system of linear equations, namely, the preconditioned Conjugate Gradient (CG) method in which the preconditioner is based on the iterative splitting scheme as detailed in [ANMTPK91,GOLOAN83,LAILID88].

An important consideration for the parallel implementation of any iterative solver is the design of a data structure which supports general unstructured sparse matrix-vector multiplication. We have developed a special data structure in which the sparse matrix is condensed by the overlaying of blocks stored into stacks of memory planes. Furthermore, a heuristic which exploits redundancies in the choice of memory planes is employed. This heuristic positions elements from different blocks of the matrix in separate memory planes wherever possible thus enhancing the parallelism in the matrix-vector multiplication kernel [ANMTPK91].

As previously stated, we wish to solve the SSPD system of equations: $My = t$ by a suitable preconditioned CG method [ALMITZ90,GOLOAN83,LAILID88] The system is recast in a normalised form by re-scaling the matrix:

$$M' = diag(M)^{-1/2}\, M\, diag(M)^{-1/2} \qquad (5.1)$$

Given that M has a special algebraic form $(M = ADA^T)$ we can deduce that

$$\left|m'_{ij}\right| < 1. \quad for\ i \neq j \qquad (5.2)$$

Although the traditional preconditioned CG method is employed [GOLOAN83], a particular issue for the massively parallel computer is the preconditioning step of this scheme. Given the original matrix, the object of the preconditioning is to find a good approximation matrix which can be easily inverted. Here, the word "easily" also implies an efficient parallel inversion. The incomplete Choleski factor, a commonly used preconditioner on serial and vector computers, is less efficient on massively parallel computers due to the high dependency of the backward substitution stage. The Jacobi-line diagonal approximation matrix for the splitting scheme is used by Lai and Liddell [LAILID88] for the solution of finite element problems on the DAP. We have adapted a natural extension of this idea by using a pre-conditioner based on a tridiagonal approximation matrix, as it is more stable than the diagonal one and a powerful parallel algorithm (the cyclic reduction algorithm) for solving such a system is available [GOLOAN83]. Unstructured sparse systems however, can present the added problem of having too many zero elements in the subdiagonals, so that the tridiagonal matrix can degenerate into the diagonal approximation matrix. For our implementation we have developed an efficient reordering heuristic which moves numerically large elements of the normalised SSPD matrix into the subdiagonals. This algorithm requires only O(nz) operations where nz is the number of non-zeros in M, since it uses an approximate sorting of matrix elements into size groups.

The preconditioning iteration step is described below

We define the following splitting of M' (after M' was reordered using subdiagonal reordering heuristic).

$$M' = P - Q \quad , \quad P = T + \delta\, I \qquad (5.3)$$

where T is the tridiagonal part of M' and $\delta \in [0,1]$.

If $\delta = 1$ then we know from (5.2) that P must be a diagonally dominant matrix and hence positive definite. For $\delta < 1$ the matrix P may be positive definite.

The splitting scheme leads to the following sequence:

$$Pq_{k+1} = Qq_k + r \quad , \quad k = 0,\ldots,K \qquad (5.4)$$

where r is a residue in one of the CG iterations,

K = number of preconditioning steps,

q is the desired solution to the "easy" preconditioning problem:

Bq = r for a preconditioning matrix B.

The result of these preconditioning iterations becomes:

$$q = P^{-1}\sum_{k=0}^{k-1}(Q\ P^{-1})^k\ r = B^{-1}\ r \qquad (5.5)$$

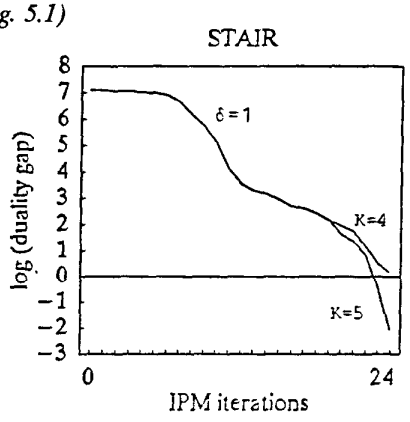
Although the above splitting scheme does not guarantee convergence, we obtained sufficient accuracy in practice.

An experimental test system was set up by replacing the direct solver for the Newton iteration step on a VAX host computer with the iterative CG scheme interfacing to the DAP for each outer iteration of IPM. The special data structure for the massively parallel matrix-vector multiplication as well as the subdiagonal ordering heuristics was computed on the host system before transferring the data into the DAP memory.

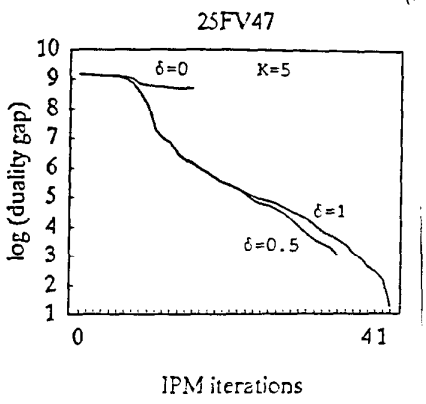
The CG iterations on the DAP were stopped when either the relative error of the solution vector was below the set tolerance (10^{-6}), or when a maximum allowance of CG iterations had been used up. The CG was restarted at 100 iterations to preserve the conjugate property for the direction vectors. A test run was performed using the NETLIB set of problems. In the graphs below we summarize the results for two problems (STAIR,25FV47). Fig. 5.1 and 5.2 show the reduction of the duality gap as a function of the outer IPM iteration. The parameter K is the number of preconditioning steps. The program terminates if either the duality gap is reduced below the set tolerance or increased in a subsequent iteration, indicating a poor solution for the Newton direction. The effect of varying the δ parameter is shown for the problem 25FV47 (Fig 5.2). Setting δ=0 is clearly insufficient for achieving a good preconditioning. Using δ=1 is safer, but reducing δ to 0.5 leads to faster convergence initially although the IPM iterations terminate prematurely due to increasing duality gap at the last step. A particular IPM problem is revealed by studying the relative CG solution error at each IPM iteration step (Fig 5.3) and the number of CG iterations used (Fig 5.4). After 9 IPM iterations the CG scheme reaches the maximum allowance of 400 iterations. Due to this early termination of the CG scheme, the relative error grows dramatically yet the IPM algorithm manages to carry on reducing the duality gap. The best LP solution (K=5) shows a gap of 0.01 corresponding to 4 digits precision in the objective function. The source of this difficulty lies in the changing part of the SSPD matrix, $M = ADA^T$, where $D = XZ^{-1}$. As some variables quickly approach their optimal values while μ is decreased, the approximate complementarity $XZe = \mu e$ -0 0 is gradually enforced, hence the corresponding elements of D can take very large or very

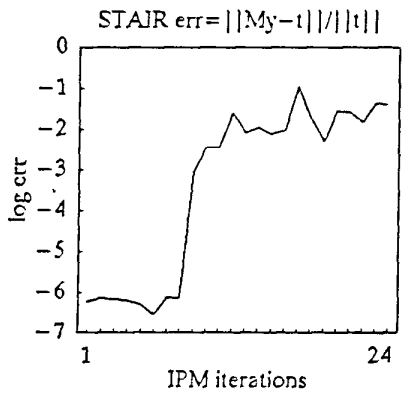
small values. This increases the condition number for the SSPD matrix M thus creating numerical problems for the CG method.

(Fig. 5.1)

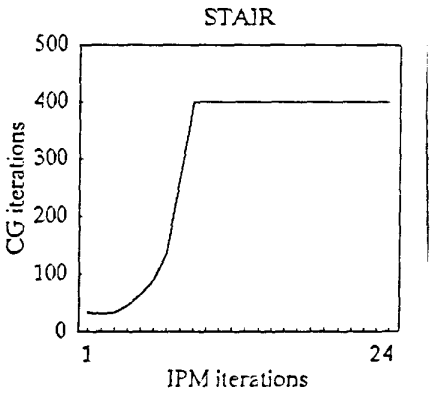


(Fig 5.2)





(Fig. 5.3)



(Fig 5.4)

6. Discussion and conclusions

Our tests show that parallel implementation on the DMC is stable, but an effective speed up can be achieved only on SSPD matrices that have wide and balanced elimination trees. Different reorderings of the SSPD matrix and balancing techniques used for the elimination tree can improve the performance substantially. The DAP implementation is especially relevant for SSPD matrices whose Choleski factor is very dense. The CG numerical problems experienced in the final iterations of IPM can be largely avoided; our experiments in cross-over to simplex indicate that the best results were achieved by terminating IPM prior to reaching the optimal solution [MTLKTZ91]. Also, flagging and removing variables converging to zero can improve the conditioning of the D matrix and in turn increase the stability of the CG solver.

7. Acknowledgements

The research reported in this paper has been partly supported by Digital Equipment Corporations, European External Research Program. We also thank PARSYTEC (Germany) GmbH for their interest in our research and for their software support. Professor Denis Parkinson of AMT Ltd has worked closely with us and we have benefited greatly from his advice regarding the DAP implementation. The support of the UK Science and Engineering Research Council (SERC) is also gratefully acknowledged, who together with AMT Ltd have supported Mr. J. Andersen's CASE studentship.

8. References

[ALMITZ90] J. Andersen, R. Levkovitz, G.. Mitra, M. Tamiz, Adapting IPM For The Solution Of LPs On Serial, Coarse Grain Parallel And Massively Parallel Computer, Brunel University, 1990.

[AMDHL67] G..M. Amdahl, Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities, AFIPS Conference Proceedings, Vol. 20, 483-485, 1967.

[ANMTPK91] J. Andersen, G.. Mitra, D. Parkinson, The Scheduling Of Sparse Matrix-Vector Multiplication On a Massively Parallel DAP Computer, Brunei University, 1991, presented to ICIAM Congress Washington, 91 and submitted to Parallel Computing.

[ASGRLW87] C.C. Ashcroft, R.G. Grimes, J.G. Lewis, etal, Progress in Sparse Matrix Methods for Large Linear Systems on Vector Supercomputers, International Journal of Supercomputer Appl, 10-30, 1987

[BIXBY91] R.E. Bixby, The Simplex Method - It Keeps Getting Better, Presented to the 14th International MPS Symposium, 1991, Holland

[BXGLMS91] R.E. Bixby, J.W. Gregory, Lustig I. J., Marsten, R.E., Shanno D.F.,Very Large Scale Linear Programming: A Case Study In Combining Interior Point And Simplex Methods, Department of Mathematical Science, Rice University, Texas, 1991

[CHNLNS90] G.H. Chen, H.F. Lin, J.P. Sheu, Data Mapping of Linear Programming on Fixed Size Hypercubes, Parallel Computing, Vol. 13, 235-243, 1990.

[DFERED86] I.S. Duff, A.M. Erisman, J.K. Reid, Direct Method For Sparse Matrices, Oxford University Press, 1986.

[FORTOM90] J.J.H. Forrest, J.A. Tomlin, Vector Processing in Simplex and Interior Methods for Linear Programming, Annals of OR, Vol. 22, 071-100, 1990.

[GAY85] D.M. Gay, Electronic Mail Distribution of Linear Programming Test Problems, COAL Newsletter MP Society, Vol 13, 10-12, 1985.

[GEOLIU81] J.A. George, J.W. Liu, Computer Solution Of Large Sparse Positive Definite Systems, Prentice Hall,1981

[GOLOAN83] G.H. Golub, C.F. Van-Loan, Matrix Computation, North Oxford Academic, 1983.

[KARMAR84] N. Karmarkar, A New Polynomial Time Algorithm For Linear Programming, Combinatorica, vol 4,pp 373-379, 1984

[LAILID88] C.H. Lai C.H, H.M. Liddell, Preconditioned Conjugate Gradient Methods On The DAP, Proceedings of The Mathematics Of Finite Elements & Applications, Vol 4. pp 147-156,1988

[LIU89] W.H. Liu, Reordering Sparse Matrices For Parallel Elimination, Parallel

Computing, Volume 11, pp73-91, 1989.
[LUMASH90] Lustig, J.I. Marsten, E. R. Shanno, D.F, On Implementing Mehrotra's Predictor-Corrector Interior Point Method For Linear Programming, Technical Report SOR 90-03, Department of Civil Engineering and Operational Research, Princeton University,1990
[LUMASH91] I.J. Lustig, R.E. Marsten, D.F. Shanno, The Interaction of Algorithms and Architectures for Interior Point Methods, Research Reports, RUTCOR, 1991
[MGIDD091] N. Megiddo, On Finding Primal-Dual and Dual-Optimal Bases. ORSA Journal on Computing No2, Winter 1991.
[MTTAMZ91] G. Mitra, M. Tamiz, Alternative Methods for Representing the Inverse of Linear Programming Basis Matrices, in, Developments in Mathematical Programming, 1975-1989, ASOR special issue, Edited by S. Kumar, Gordon-Breach, 1991.
[MONADL89] D.C. Monteiro, I. Adler, Interior Path Following Primal-Dual Algorithm, Mathematical Programming 44, 1989
[MTLKTZ91] G. Mitra, R. Levkovitz, M. Tamiz, Integration of IPM Within Simplex, Experiments In Feasible Basis Recovery, Brunel University, Presented to 14th MPS Symposium ,1991, Holland.
[PARPRS90] P.M. Pardalos, A.T. Phillips, J.B. Rosen, Topics in Parallel Computing in Mathematical Programming, Report CS-90-22, Department of Computer Science, Pennsylvania State University. Also in SIAM frontiers in Applied Mathematics, 1990.
[STUNRD88] C.B. Stunkel, D.A. Reed, Hypercube Implementation of the Simplex Algorithm, Proceedings on Hypercube Concurrent Computers and Applications, IAACM Publication, 1473-1482, 1988.