TR/01/90 January 1990

ADAPTING THE INTERIOR POINT METHOD FOR THE SOLUTION OF LPs ON SERIAL, COARSE GRAIN PARALLEL AND MASSIVELY PARALLEL COMPUTERS

J Andersen, R Levkovitz,
G Mitra and M Tamiz

ADAPTING THE INTERIOR POINT METHOD FOR THE SOLUTION OF LPs ON SERIAL, COARSE GRAIN PARALLEL AND MASSIVELY PARALLEL COMPUTERS

J Andersen

Brunel University

R Levkovitz
Brunel University

G Mitra
Brunel University

M Tamiz
NAG Limited

Presented at the International Symposium on Interior Point Methods for Linear Programming: Theory and Practice, January 18-19, 1990, Europe Hotel, Schevenigen, Holland.

0. ABSTRACT

In this paper we describe a unified scheme for implementing an interior point algorithm (IPM) over a range of computer architectures. In the inner iteration of the IPM a search direction is computed using Newton's method. 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 serial, coarse grain parallel and massively parallel computer architectures, are considered in detail. We put forward arguments as to why integration of the system within a sparse simplex solver is important and outline how the system is designed to achieve this integration.

1. INTRODUCTION AND BACKGROUND

Over the last thirty years computational algorithms for the solution of linear programs (LP) using the sparse simplex (SSX) method have made remarkable and sustained progress [GILMSW84, GOLTOD88, MURTAG81, ORCHAY68, TAMIZ86] and these advances are certainly set to continue. In an equally remarkable development the innovative projection algorithm put forward by Karmarkar in 1984 [KARMAR84] has deeply influenced theoretical and applied research into LP solution techniques. Taking into account the geometric properties of the starting point and the sequence of search directions, this genre of algorithms [GOLTOD88] have been labelled interior point algorithms (IPM). Most IPMs not only possess the theoretical property of low order (worst case) polynomial complexity, they also perform extremely well in medium to large problems [MARSHN89]. This two fold impact (resulting from the contribution of Karmarkar and the subsequent IPM theory and applications) has considerably stimulated further research into linear and convex programming.

Research and developments in the solution techniques for large sparse unsymmetric and symmetric systems of linear equations, have also influenced directly and indirectly the SSX and IPM computational schemes. Computationally the most demanding steps in the SSX algorithms are concerned with basis inversion, update, and deriving solution values [MITAMZ90, GILMSW84]. Well established sparse unsymmetric (SU) equation solving techniques are used in these steps. Computationally the most burdensome step in the IPM requires the solution of a sparse symmetric positive definite (SSPD) system of equations (see next section). Efficient methods of solving SU and SSPD linear equations have been widely studied and reported for established as well as for newly emerging computer architectures [GEOLIU81, DUFERD86, SPCONF89]. We believe exploitation of these research results in the context of SSX and IPM methods are going to provide major computational developments in the forthcoming years.

The main focus of the work reported in this paper is the efficient adaptation of the IPM in relation to different computer architectures. A secondary but nontheless important objective is to embed the IPM within a sparse simplex (SSX) based LP system. Our main motivation is to take advantage of a suboptimal solution derived at an intermediate stage of the IPM algorithm and use it for a "warm start" of the SSX algorithm [MITAMY88, GLMSTW86, MARBAL88]. The SSX method also has other attractions such as its use in post-optimal analysis as a descriptive algorithm and its well established use in mixed integer programming as the reoptimisation algorithm for the sub-problems of the search tree. Recently the challenge of the IPM has also spurred a new wave of developments by the practitioners of the SSX method. Thus new strategies for PRICE and BASIS INVERSE representation have been introduced [BIXBY89, FORTOM89, ATSUHL88] leading to competitive solution times for large industrial test problems [CHKENW89] on serial and vector computers. These features provide us with compelling reasons to construct computing kernels with exchangable data structures and integrate the IPM and SSX algorithms.

It is well known to researchers in computational linear programming and developers of LP software systems that the solution methods for large scale LP have progressed in three major directions covering algorithmic developments, introduction of novel software techniques and developments in hardware. Our research interests embrace all three aspects and in particular focus on how the recent development in IPMs can be exploited in conjunction with software techniques for newly emerging computer architectures. The use of these hardware platforms for computationally intensive technical and scientific application areas is now well accepted [BERTSK88, PERROT89]. We have classified the present range of computer architectures which are used in scientific computing in a tree structure which is illustrated in diagram 1.

COMPUTER HARDWARE CLASSES

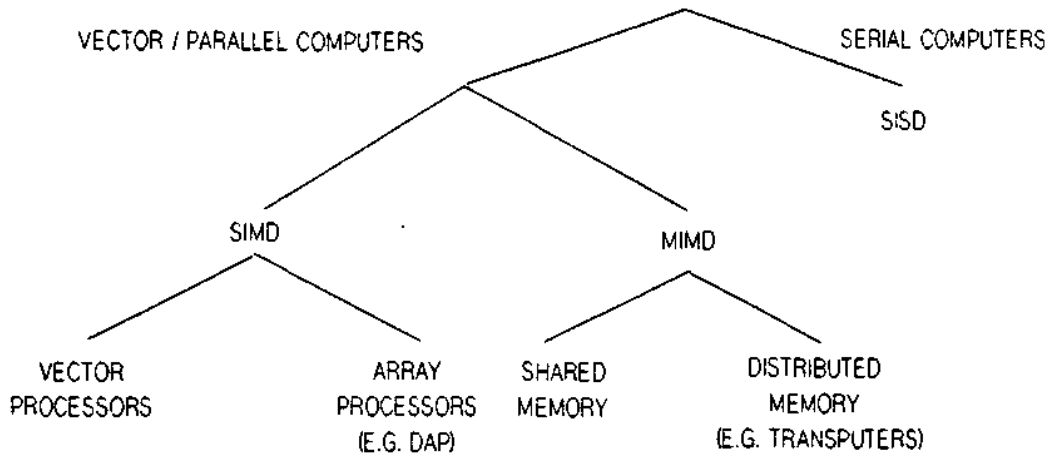

Diagram 1

In diagram 1 the traditional computers are labelled as single instruction single data (SISD) machines. These are also known as serial computers with the original von-Neumann architecture. All other types of computers are grouped as vector and parallel computers. The single instruction multiple data (SIMD) and multiple instruction multiple data (MIMD) are the other two of the three main classes of operational machines described by Flynn [FLYNN72] who provided the earliest taxonomy of serial and parallel computers. Within SIMD class, vector computers use a vector of registers, which are processed by the same instruction in a pipeline. The distributed array processor (DAP) is an example of the other SIMD machine type with an array of processors arranged in a plane, and which can process data held in corresponding arrays of elements [HUNT89] (also see section 4). In contrast with SIMD machines the main feature of the MIMD machines is that each processor may be executing different instructions at any one time on the different items of data. If all the processors can access the same memory location then it is a shared memory machine. Alternatively, data may be distributed (communicated) to the private memory of local processors which may operate on these data items concurrently. These are called distributed memory machines. Our classification is in no way exhaustive, but is fairly representative of the current status of the established architectures. A comprehensive discussion of the hardware classification, software techniques, principles of algorithm design, for these new machines, can be found in [HOCJES88, PERROT89, BERTSK88]. We have only considered three of these machines namely SISD, SIMD (DAP) and MIMD (shared memory) machines. These machines are, however, sufficiently varied to support the algorithm and software design principles discussed in this paper.

The contents of the rest of this paper are organised in the following way. In section 2 we briefly review the range of IPMs taking into account complexity and computational issues. We then outline the primal-dual barrier function approach and justify our choice of this algorithm. The computational implication of the inner iteration which involves solution of an SSPD system for different computer architectures are considered. In section 3 we introduce two parallel computational models for MIMD machines. We review a number of well established algorithms and then describe and justify the adoption of a novel algorithm which is an extension of Toczylowski's method. In section 4 we describe the computational model of the SIMD (DAP) machine. We also discuss the issues of sparsity, choice of data structures and the choice of iterative solution algorithms. We present the strategy and the algorithmic techniques for basis recovery and integration of the IPM computing kernels within a SSX system in section 5 and our conclusions in section 6.

2. CHOICE OF THE INTERIOR POINT ALGORITHM

2.1 A Brief Review of the IPMs

The interior point method can be broadly classified into three main groups, namely, the projective algorithm of Karmarkar and its variations, the affine scaling algorithm and the path following algorithm.

- Projective Algorithms

The original method due to Karmarkar [KARMAR84] has some restrictive assumptions; such as that the constraints are cast in a system of homogeneous equations and a simplex. Furthermore the optimum solution value must be estimated in advance. Anstreicher [ANSTRC86], De Ghellinck and Vial [GHVIAL86], Gonzaga [GONZAG87], Ye and Kojima [YEKOJM87] have also studied this class of algorithms. The outer iterations of these algorithms are shown to be of order O(nL) and the original algorithm of Karmarkar has an overall complexity $0(n^{3.5}L)$. The practical implementation of this class of algorithms for industry standard test problems is no longer considered worthwhile although they continue to provide considerable theoretical interest in terms of complexity issues and convergence properties.

- Affine Scaling Algorithms

In the affine scaling algorithms the projective transformation originally proposed by Karmarkar is replaced by an affine transformation. It was later observed that Dikin had proposed such an algorithm in 1967 [DIKIN67]. Algorithms of this class have also been studied by Meggido and Shub [MEGSHB86], Vanderbei, Meketon and Freedman [VANMKF86] amongst others. In general there are no known polynomial complexity bounds for these algorithms, except for the primal-dual affine algorithm for which Adler et al [MONADR88] derived a polynomial bound. The dual affine algorithm was introduced as a practical implementation of the interior point method by Adler, et al [ADKARV89], and was also investigated by Marsten et al [MARSHN89]. The primal affine algorithm is one of the alternative computational methods implemented within the AT&T's KORBX system [CHKENW89].

- Path Following Algorithms

Karmarkar [KARMAR84] in his original paper also discussed the concept of a central trajectory of the solution points. Mathematically the trajectory can be defined by suitable differential equations taking into account a nonlinear programming (NLP) log barrier function formulation of the LP problem. Gill et al [GLMSTW86] were the first to establish the equivalence of the projective method with Newton barrier method for LP. Renegar [RENEGR88], Kojima et al [KOJMY087], Roos and Vial [ROOVAL88] amongst many others, (see [ROOHRT89]) have studied the path following approach. In general

these algorithms take $O(n^{0.5}L)$ outer iterations. Monteiro and Adler [MNADLR87] have derived the best known complexity bound $0(n^3L)$ for the primal-dual algorithm.

2.2 Primal-Dual Barrier Method

There are two major attractions of the primal-dual barrier method as an algorithm for computational implementation. In this method the primal and dual solutions are feasible at each step. (The complementarity is achieved only at the final step). Thus intermediate solutions from this method can be easily introduced in other methods such as primal simplex. The simple upper bound restrictions and free variables which are often found in industrial test problems can be easily introduced and implicitly represented in this approach.

Consider the primal problem Pr in the standard form in which the simple upper bounds $(0 \le x \le u)$ have been turned into equality using the slack variables y.

$$\begin{aligned} \text{Pr}: \text{Min} \quad & c^T x + 0y, \ (x,y) \in p \cap R_+^{2n} \\ & p = \{(x,y) |\ Ax = b,\ Ix + Iy - u\} \\ & A \in R^{mxn}, b \in R^m, u \in R_+^n \end{aligned} \tag{2.1}$$

Its associated dual Du is stated in the variables v,w,z

$$\begin{aligned} \text{Du}: \text{Max} \quad & b^T v - u^T w,\ v \in R^m, (z,w) \in D \cap R_+^{2n} \\ & D = \{(z,w)\} |\ A^T v + Iz - Iw = c\} \end{aligned} \tag{2.2}$$

The barrier function formulations of these two problems are stated as:

$$\textbf{Barrier Pr:} \quad \text{Min}\, c^T x - \mu \sum_{j=1}^{n} [\ell n(x_j) + \ell n(y_j)],\ (x,y) \in p \tag{2.3}$$

$$\textbf{Barrier Du:} \quad \text{Max}\ b^T v - u^T w + \mu \sum_{j-1}^{n} [\ell n(w_j)] + [\ell n(z_j)], (w,z) \in D \tag{2.4}$$

Forming the Lagrangian functions for these two problems and applying the first order optimality conditions to any of these lead to the following set of equations [MCMNSH89, LUSMSH89]

$$\begin{aligned} & Ax = b \\ & Ix + Iy = u \\ & A^T v + Iz - Iw = c \\ & XZe = \mu e \\ & YWe = \mu e \end{aligned} \tag{2.5}$$

If we apply Newton's method to solve this system of equations with two sets of nonlinear (bilinear) relations XZe $=\mu$ e and YWe = μ e, then the following equations for the Newton directions $\Delta x, \Delta y, \Delta v, \Delta w, \Delta z$ can be derived.

$$\begin{bmatrix} A & o & o & o & o \\ I & I & o & o & o \\ o & o & A^T & -I & I \\ Z & o & o & o & X \\ o & W & o & Y & o \end{bmatrix} \begin{bmatrix} \Delta X \\ \Delta Y \\ \Delta V \\ \Delta W \\ \Delta Z \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ \mu e - xZe \\ \mu e - yWe \end{bmatrix} \qquad (2.6)$$

The directions can now be computed in the following way:

$$\Delta v = (ADA^T)^{-1} q, \qquad (2.7)$$

where $D = (Y^{-1}W + X^{-1}z)^{-1}$

$q = ADp(\mu)$

$p(\mu) - \mu(Y^{-1} - X^{-1})e - (W - Z)e$

$$\Delta x - D(A^T \Delta v - p(\mu)) \qquad (2.8)$$

$$\Delta z - \mu x^{-1} e - ze - x^{-1} z \Delta\Delta \qquad (2.9)$$

The last equation (row) give us:

$$W\Delta y + y\Delta W = \mu e - YWe \qquad (2.10)$$

Using

$\Delta y = -\Delta x$, we finally obtain : (2.11)

$$\Delta z = \mu x^{-1} e - ze - x^{-1} z \Delta\Delta \qquad (2.12)$$

These results are given in [MCMNSH89, LUSMSH89] and also derived in [ANDRSN90-1].

2.3 Computational Implementation of the Algorithm

In order to describe the framework of the algorithm we need to consider four major aspects, namely, (a) Starting point, (b) Search direction, (c) Step size, (d) Termination criterion.

(a) Starting point

In the simplex method there are two distinct phases. The Phase I objective is first investigated and after feasibility is attained the Phase II is investigated. Here it is necessary to use composite Phase I, Phase II form, both for the primal and the dual. We also need to start from a feasible interior point for the primal as well as the dual. The modified primal and the dual incorporating the artificial primal and dual variables $(x_a, x_b), (v_a, z_a, z_b)$ are written as

$$\begin{aligned}
&\text{Min} && c^T x + c_a x_a \\
&\text{subject to} && Ax + (b - Ax^0)Tx_a = b \\
& && d_D^T\, x + x_b = b_a \\
& && x + y = u \\
& && x, y, x_a, x_b > 0
\end{aligned}$$

The dual problem:

$$\begin{aligned}
&\text{Max} && b^T v - u^T u + b_a v_a \\
&\text{subject to} && A^T v + d_D^T v_a + z - w - c \\
& && (b - Ax^0)^T v + z_a - c_a \\
& && v_a + z_b = 0 \\
& && w, z, z_a, z_b > 0
\end{aligned}$$

$(x^0, x_a = 1)$ is taken as the primal starting solution $\left(v^o, z^o w^o, v_a = -1\right)$ as the dual starting solution. As discussed by Lustig [LUSMSH89], the artificial row d_D can be determined for improved computational efficiency of the Phase I Newton step.

$$d_D = \begin{cases} \beta e & [\text{ADKARV86,MCMNSH89}] \\ A^T v^0 + z^0 - w^0 - c & [\text{LUSMSH89}] \end{cases}$$

where β is a scalar chosen to make z° feasible.

Typically $c_a = 3n^2 \max \left|c_j\right|$

$b_a = 3n^2 \max \left|b_j\right|$

For a detailed discussion of these see [LUSMSH89.ANDRSN90-1]

(b) Choice of barrier parameter μ

When advancing through the interior domain, it is necessary to have a balance between progress in the value of the objective function and corrective move towards the central path. This is determined by the choice of μ at each step in relation to the step size as computed by the Newton method and what fraction of it is used in the actual step. See [ADKAR89,MCNANSH89].

(c) Step size

Full step sizes are computed by the relations shown in the last section. In the computational algorithms the step sizes are attenuated by the factors $\alpha_p \alpha_d$ for the primal and dual variables.

$$\alpha_p = \gamma\overline{\alpha}_p$$

$$\alpha_d = \gamma\overline{\alpha}_d$$

where $\overline{\alpha}_p, \overline{\alpha}_d$ are deduced by minimum ratio test for preserving primal and dual feasibility of the new solution and $\gamma = 0.995$ [LUSMSH89].

(d) Stopping Criterion

The two alternative stopping criteria, namely.

$$\frac{Ic^T x - b^T v + u^T WI}{Ic^T XI} < \in, \qquad \text{where } \left|c^T x\right| >> 0$$

and $\in = 10^{-8}$

and $\left|c^T x - b^T v + u^T w\right| < \varepsilon$,

have been successfully used. Since we complete the solution with full basis recovery we do not consider this any further.

The primal-dual log barrier algorithm is set out below.

begin

A1. Initialise

begin

1. Set up Phase I objective function
2. $x_a = 1,\ v_a = -1$
3. Initialise $x^0, y^0, v^0 z^0, w^0$
4. Initialise SSPD solver (see sections 3 and 4)

end

A2. Solve for Newton directions

b**egin**

1. Update ADA^T matrix
2. Solve $\left(ADA^T\right)\Delta v = q$
3. $\Delta x = D\left(A^T \Delta v - p(\mu)\right)$
4. Compute $\Delta z, \Delta y, \Delta w$ by simple scaling operations

end

A3. Update primal and dual variables

begin

1. determine step size factors α_p, α_d

2. $x' = x + \alpha_p \Delta x$ (also x_a if Phase I)

$y' = y + \alpha_p \Delta y$

$v' = v + \alpha_d \Delta v$ (also v_a if Phase I)

$z' = z + \alpha_d \Delta z$

$w' = w + \alpha_d \Delta w$

end

A4. Monitor convergence

A5. If x_a and v_a close to 0 then switch to Phase II.

A6. If the primal solution is nonfeasible or unbounded or optimum
then Finalise (A7)
else repeat A2 to A6

A7. Finalise
begin
1. basis recovery procedure (see section 5)
2. output
end

end

Alternative approaches for the solution of the SSPD system in step A2 are discussed in section 3 and section 4.

3. DIRECT SOLUTION OF INNER STEP SSPD EQUATIONS ON A SERIAL AND A SHARED MEMORY MULTIPROCESSOR COMPUTER

3.1 MIMD Hardware and the Underlying Computational Model

A shared memory multiprocessor computer has a number of processors which can access and share a common memory. Additionally each processor can have it's own private memory which cannot be accessed by the other processors. In general these processors are powerful and can perform all the normal computational tasks. In these machines waiting for memory access is a known problem as two or more processors attempt to access different parts of the memory at the same time using a common bus. The memory access is essentially a serial process often delayed by such bottlenecks. This problem can be partially alleviated by using a very fast bus, by separate memory segments controlled by complex switching systems and by using special hardware devices such as cache memory [HOCJES89].

Shared memory multiprocessor computers use the common memory as a communication pool whose main function is to transfer data and to synchronise the processors. Unlike other computer architectures where the method of connecting the processors mainly determines the computational model, a shared memory computer is flexible, and can be

adopted to fit the algorithm. The basic entity of the parallel computation is a *process.* A *process* is allocated to a processor by the user, or automatically by the system software. The size of a *process* plays a crucial part in the overall performance of a parallel algorithm on the multiprocessor shared memory computer. Due to communication and synchronisation overheads, a task with less than a thousand execution instruction should not be divided into smaller processes [VMSRTL88]. On the other hand, dependency between processes and unpredictability of execution time prevents us from having too large processes. Ideally, the processes have several hundred to several thousand basic instructions and a *process* can be lightweight or heavyweight, and the former is defined as one with less than a thousand instructions. Two principal parallel computational models namely, MIMDS1, MIMDS2, are considered. See diagram 2.

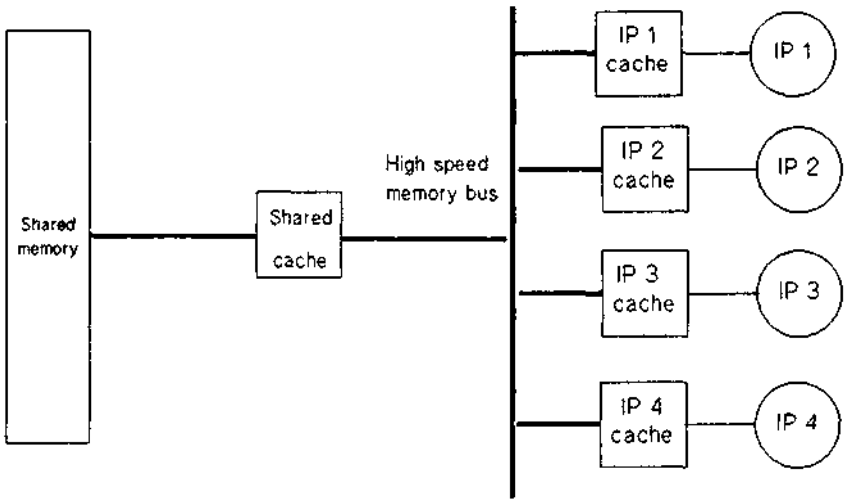

Diagram 2

Principal MIMD architecture using cache
and high speed bus

In MIMDS1 p processors are used and all the processors are considered equivalent. The algorithm stream is sequential and the parallelism is exploited by separating a single task to small, lightweight processes. One of the processors is used to execute the main sequential algorithm. The parallel processes in a single task are executed by all processors. The processes and the processors are anonymous, the next available processor executes the next process in the processes queue. When the task is finished the control returns to the single processor.

In MIMDS2 p processors are used and as before the memory is shared. Part of the memory, however, may be allocated as private to a processor if neccesary. In contrast to the first model a master processor allocates distinct tasks to the slave processors and to itself. This model exploits the parallelism in the algorithm itself and the streaming of the algorithm can be parallel. Task scheduling and synchronisation of control flow are required. The processes are usually heavyweight processes.

3.2 Discussion of Algorithms Suitable for (SISD) Serial Machines

Most successful implementations of the IPM method have made use of proven techniques for solving the SSPD system $(ADA^T)\Delta v = q$. It is well known that a few dense columns of A may produce an extremely dense (ADA^T) matrix. As a result a method of dealing with the dense windows by partitioning the A matrix and using the Schur complement has evolved. We have adopted this approach as well and followed the general strategy suggested by Marsten et a1 [MARSHN89].

ALGORITHM 1

Step A1.4 expanded; initialise SSPD solver.

begin

1. Find the permutation matrix Q for A that corresponds to mimimum degree ordering of AA^T.
2. $\bar{A} = QA$
3. Column partition the matrix to sparse and dense parts

$$\bar{A} = \left[\overline{A}_s \mid \overline{A}_d\right]$$

4. Initialise the data structure for $\overline{A}_s D_s \overline{A}_s^T$

end

Step A2.2 expanded; solve $(\overline{A}D\overline{A}^T)\Delta v = q$

begin

1. Compute $v = \overline{A}_d D_d$
2. Factorise $LFL^T = \overline{A}_s D_s \overline{A}_s^T$

3. Set up:

$$\begin{bmatrix} LFL^T & -V \\ V^T & I \end{bmatrix} X \begin{bmatrix} \Delta v \\ \delta \end{bmatrix} = \begin{bmatrix} q \\ o \end{bmatrix}$$

4. Solve for δ by dense Cholesky factorisation:

$$\left[I + V^T \left(L^F L^T \right)^{-1} V \right] \delta - - V^T \left(LFL^T \right)^{-1} q$$

5. Compute $\Delta \upsilon = \left(LFL^T \right)^{-1} (q + V\delta)$

end

Step A2.2 is executed repeatedly each time a direction of search is computed. This comprises the inner iteration.

3.3 Review of Alternative Methods for MIMD Machine

There are three different approaches towards implementing an algorithm to run on a parallel machine. The first is to make use of the manufacturer supplied precompiler which automatically transforms an existing serial code to a partially parallelised code for the target machine. Although this may lead to a good speed up in many cases we do not consider this any further. The second approach is to analyse a well established algorithm and to discover inherent parallelism within it. Algorithms with proven properties of stability, convergence, etc., are attractive candidates and often adopted in this way. In this approach the main effort goes towards designing new task scheduling schemes. The third approach is to design altogether new parallel algorithms which exploit a given parallel architecture. Algorithms which were introduced but never investigated in detail on serial machines may become candidates for consideration again on parallel machines. For a system implementor this poses a minor disadvantage that the experience in the use of such algorithms may be limited. The Cholesky factorisation schemes which take into account sparsity through matrix reordering, are well established and good candidates for the second type of parallelisation. The algorithms [GEOLIU81, DUFERD86] not only do well on serial machines [MARSHN89] they have also been adopted for different parallel architectures [LIU89, GHLIUN89]. Here we consider the algorithm proposed by Liu in a summary form.

ALGORITHM 2 [LIU89]

Reordering a matrix towards minimum fill-in creates a sequence of eliminations. This sequence reveals that some columns are not dependent because they do not affect each other in the elimination process. These columns can be collected in independent groups. When implementing a parallel algorithm, columns in the same independent group can eliminate in parallel. Liu [LIU88] represents the parallel structure of the elimination sequence by building an elimination tree through simple transformations of the elimination tree, the largest independent groups are identified.

Let P be mxm symmetric positive definite non-reducible matrix; let G(P) be the corresponding matrix graph. Let E = L+L^T be the Cholesky factorisation matrix and let G(E) be the corresponding graph representation. Assume that the matrix P is already ordered towards reducing fill-in and let $v_1,.....,v_n$ be the corresponding nodes sequence in the graph the T(P). The elimination tree is defined as follows:

The elimination tree T(P) has the same node set as the graph G(P).

A node v_i is the parent of v_j $(i > j)$ if and only if

$$i = \min \{r \mid l_{rj} \neq 0\}$$

l_{rj} is the j'th entry in the r'th row of the Cholesky factor L.

In other words, a node is a leaf only if it represents a column that only eliminates with no operations done on it. A node v_j is a son of v_i if it eliminates the latter directly. Liu shows that parts of the tree can be transformed, and while preserving the reduced fill in property of the sequence, a permuted tree with lower height can be found. It pays to "widen" the tree as this creates groups of tasks that can be done in parallel. Hence algorithm No 2 is essentially a scheduling algorithm. Another algorithm which attempts to schedule the Cholesky factorisation work has been suggested by Alvarado [ALVARD86]. This algorithm schedules the basic numeric elimination operations.

Algorithms in the third group have been only recently investigated as new machine types have emerged. For Cholesky factorization for instance Zmijewski [ZMWSKI89] suggests a new algorithm which was designed specially for a message passing multiprocessor computer with a view to reducing the communication overheads which create bottlenecks in these types of computers.

3.4 A New Approach for the Solution of SSPD on Parallel Computers

Tockzylowski had proposed an innovative algorithm [TCWSKI83] for computing the inverse of basis matrices. The method involves representing the inverse in a hierarchical product form. We have extended this algorithm from the SU to the SSPD systems and in particular we exploit the inherent parallelism of this algorithm.

ALGORITHM 3

Let $P \in R^{mxm}$ be an SSPD matrix and consider its 2 partitions (m = k+p), with 4 submatrices.

$$P = \left[\begin{array}{c|c} S & G^T \\ \hline G & B \end{array}\right]; S \in R^{kxk}, G \in R^{pxk}, B \in R^{pxp} \qquad (3.1)$$

Then P can be written in a factored form as

$$P - \begin{bmatrix} S & 0 \\ 0 & I_P \end{bmatrix} x \begin{bmatrix} I_k & 0 \\ G & I_P \end{bmatrix} x \begin{bmatrix} I_k & 0 \\ 0 & Q \end{bmatrix} x \begin{bmatrix} I_k & R \\ 0 & I_P \end{bmatrix} \tag{3.2}$$

where $R = S^{-1} G^T$, $Q = B - GR$, I_k , I_p, are kxk and pxp unit matrices. The unique P^{-1} can then be written in the multiplicative form as

$$P^{-1} - \begin{bmatrix} I_k & -R \\ 0 & I_P \end{bmatrix} x \begin{bmatrix} I_k & 0 \\ 0 & Q^{-1} \end{bmatrix} x \begin{bmatrix} I_k & 0 \\ -G & I_p \end{bmatrix} x \begin{bmatrix} S^{-1} & 0 \\ 0 & I_P \end{bmatrix} \tag{3.3}$$

In (3.3) Q^{-1} is the ubiquitous Schur-Complement [COTTLE74] of S in (3.1). P^{-1} in the explicit form may be written as

$$P^{-1} = \begin{bmatrix} S^{-1} - RQ^{-1}R^T & -RQ^{-1} \\ -Q^{-1}R^T & Q^{-1} \end{bmatrix} \tag{3.4}$$

The partition and the factorisation may be applied recursively to ℓ partitions of P in the following way. Let T = $\{t_1, t_2 \ldots t_\ell\}$, denote a set of indices such that t_i , $(i-1,...,\ell)$ denote the dimensions of the principal submatrices S_{ti} of P . Thus P = S_{t_ℓ}, $t_\ell = m$, $P \in R^{mxm}$.

The computation of the inverse of P is defined by the recurrent factorisation relations (as in (3.3)). (See diagram 3)

$$St_{i+1}^{-1} - \begin{bmatrix} I & -R_{t_i} \\ 0 & I \end{bmatrix} x \begin{bmatrix} I & 0 \\ 0 & Q_{t_i}^{-1} \end{bmatrix} x \begin{bmatrix} I & 0 \\ -G_{t_i} & I \end{bmatrix} x \begin{bmatrix} S_{t_i}^{-1} & 0 \\ 0 & I \end{bmatrix} \tag{3.5}$$
$$i = 1,...,(\ell - 1)$$

The agenda of operations set out below define and specify the scheme for computing the corresponding submatrices

OP1	$R_{t_i} = S_{t_i}^{-1} Q_{t_i}^T$
OP2	$Q_{t_i} = B_{t_i} - Q_{t_i} R_{t_i}$
OP3	Compute $Q_{t_i}^{-1}$ explicitly.
OP4	Compute $W_{t_i} = R_{t_i} Q_{t_i}^{-1} R_{t_i}^T$
OP5	Compute the off diagonal block of $S_{t_{i+1}}^{-1}$, namely $-Q_{t_i}^{-1} R_{t_i}^T$
OP6	Update the inverse additively with W_{t_i}

The algorithm is naturally suited for parallel implementation within the first computational model. All the operations OP1 to OP6 with the exception of OP3 require matrix operation. Parallel algorithms for these multiplicative operations are well known. An efficient implementation of this method on a shared memory MIMD machines depends on the chosen dimensions of the block diagonal kernel (BDK) matrices. The BDKs are p_i x p_i in dimension where $p_i = t_{i+1} - t_i$. The choice of the p_i for the BDKs are governed by two considerations. p_i should not be too small as it would fail to exploit MIMD parallelism, neither should it be too large as this would impinge on the available storage. The use of BDKs explicitly also reduces the memory access bottlenecks. The OP3 (Qt_i nverse computation) is carried out for a p_i x p_i matrix explicitly. The Qt_i inverses are all held in a dense data structure, whereas the other intermediate sub matrices are held in a sparse data structure. This form is an extension of the block bordered lower triangular (BBLT) form [TCWSKI83] and the diagonal elements instead of being 1 x 1 element are p_i x p_i symmetric BDKs. In the SU system the matrix is reordered to obtain a fully reduced lower block triangular form. In our method we order the matrix towards minimum band width using the reverse Cuthill-McKee heuristic [GEOLIU81]. The BDKs are then identified using a greedy heuristic, based on a simple band density. Since P is positive definite all the BKDs must be non singular. In our implementation we have used the generalised envelope data structure whereby all the off BDK matrices are stored in sparse rectangular matrix structures [LEVANM89-2]. A bound on the number of operations and complexity of the algorithm is also derived in this report.

4 PARTITION OF THE MATRIX $\mathbf{P} = S_{t4}$

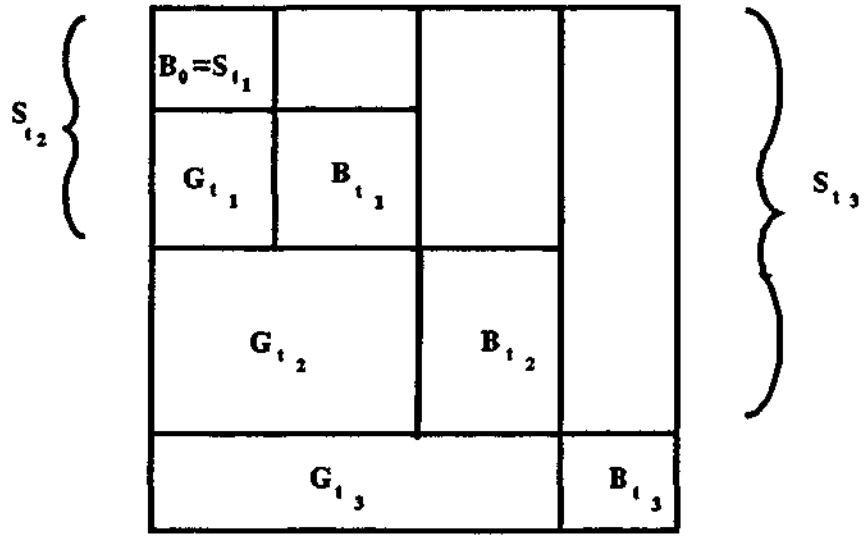

Diagram 3

Finally, we present the algorithm based on the above considerations. (ALGORITHM 3)
Step A1.4 expanded; initialise SSPD solver.

begin

1. Find a permutation matrix Q for A that corresponds to a reduced bandwidth ordering of AA^T
2. $\bar{A} = QA$
3. Column partition the matrix to sparse and dense parts

$$\bar{A} = [\bar{A}_s \mid \bar{A}_d]$$

4. Identify the block diagonal kernels
5. Analyse the size of the expected inverse
6. Initialise the data structure for $\bar{A}_s D_s \bar{A}^T$

end

Step A2.2 expanded; solve $(\bar{A}D\,\bar{A}^T)\Delta v = q$

begin

1. Compute $V = \bar{A}_d D_d$
2. Compute the inverse $P^{-1} = (\bar{A}_s D_s \bar{A}_s^T)^{-1}$ using extended Tockzylowski method.
3. Set up:

$$\begin{bmatrix} P & -V \\ V^T & I \end{bmatrix} x \begin{bmatrix} \Delta v \\ \delta \end{bmatrix} = \begin{bmatrix} q \\ o \end{bmatrix}$$

3. Solve for δ by dense Cholesky factorisation:

$$[I + V^T P^{-1} V]\,\delta = -V^T P^{-1} q$$

4. Compute $\Delta v = P^{-1}(q + V\delta)$

end

4. ITERATIVE SOLUTION OF INNER STEP SSPD EQUATIONS ON THE DAP COMPUTER

4.1 SIMD (DAP) Hardware and the Underlying Computational Model

The SIMD machine we are using for this project is an example of a massively parallel computer: the AMT DAP610. It has 4096 processing elements (PE's) organised in a 64 by 64 grid. Each PE is a fairly simple 1 bit processor. A floating point operation takes on average 1000 clock cycles and a cycle takes 100 ns. The potential performance for this computer is thus 40 MEGAFLOPS. The memory bandwidth for this system is very high (5.1 Gbyte/s), hence the DAP can also be viewed as computer with a very wide 4096 bit bus, this makes the sustained maximum performance realistic for dense matrix problems, which fits onto the DAP 64x64 grid.

The memory is organised as planes of 64 by 64 items of variable word length. A matrix location in memory is equivalent to the third index of a FORTRAN array, this can be visualised as the downward direction shown in diagram 4. The first and second array indices are left vacant, except for explicit scalar or vector operations, which are less efficient. For instance, the parallel addition of two 64 by 64 matrices, indicated as plane-elements of 3-dimensional arrays, would be performed by the following piece of FORTRAN code:

```
REAL A(,,100), B(,,100), C(,,100)
……..
C(,,97) = A(,,97) + B(,,97)
```

Inter-process communication is possible by shift operations in each direction of the grid. The longest communication path in one direction is 32 nodes or cycles, this time is still short compared with the time for a FLOP, hence the communication overhead is relatively modest.

The SIMD machine does not have independent streams, all computations on the DAP are performed in a lock-step fashion according to a single instruction which is decoded by the Master Control Unit (MCU). This simplification of the hardware allows for the "massive" propagation of PE's into a feasible system in terms of engineering constraints. A detailed description of the DAP system is given by [HUNT89].

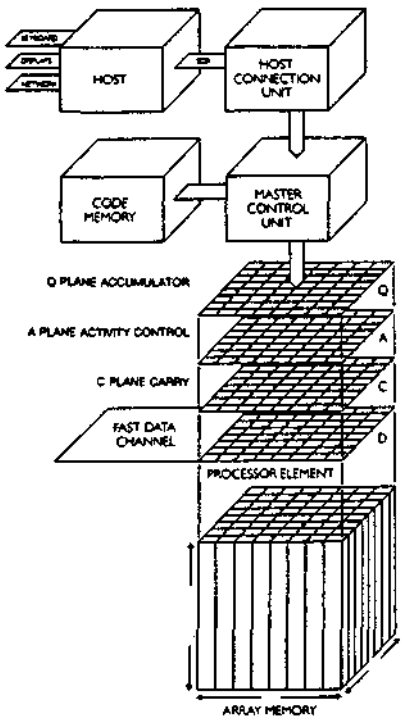

Diagram 4

1.2 Choice of iterative solver for the SSPD

The direct method for solving the SSPD problem involves invariably the Cholesky factorisation of the matrix, with a strategy for limiting the fill-in of non-zero elements, and the computational work involved is highly list-orientated. It is therefore difficult to transfer these algorithms effectively to the DAP computer considered here. This and the fine grained nature of this machine architecture led us to choose an iterative solver, in which the computational steps can be reduced to basic linear algebra operations.

- Iterative solvers

There are many well established iterative solvers, such as Jacobi, Gauss-Seidel, SOR, SSOR, Conjugate Gradient and its many derivatives. Most of the applications are for problems relating to discretisation of PDE's, e.g. finite elements. These problems have generally a banded structure, that is a structured sparsity which is exploited in parallel computations.

An iterative solver which is well adapted for the SSPD system of equations is the preconditioned Conjugate Gradient algorithm. We detail the basic steps of a solver with an ℓ-step preconditioner, according to Golub and Van Loan [GOLOAN83]. This algorithm was also considered by Lai and Liddell [LAIDL88] for the solution of finite element equations on the DAP computer.

Solve the system $P\Delta v = q$, $P = ADA^T$

Step A2.2 expanded

begin {iterative solver}

1. $\Delta v_o = 0$, $r_o - q$ {$r_o = q - P\Delta v$}
2. Solve $Mh_o = r_o$ {preconditioning}
3. $\pi_o = h_o$
4. for k = 1 to m do

begin

1. $\alpha = h_{k-1}^T r_{k-1} / \pi_{k-1} P \pi_{k-1}$

2. $\Delta v_k = \Delta v_{k-1} + \alpha \pi_{k-1}$
3. $r_k = r_{k-1} - \alpha P \pi_{k-1}$

4. if $\| r_k \| < \in$ *then*
   stop
   else

begin

1. **solve** $Mh_k = r_k$

2. $\beta = h_k^T r_k / h_{k-1}^T r_{k-1}$

3. $\pi_k = h_k + \beta \pi_{k-1}$

end

end

end.

The matrix M is the resulting preconditioner of an ℓ-step relaxation scheme. Following [LAILDL88] we start by considering a splitting P = G - H, resulting in the relaxation sequence:

$$h^{(\ell)} = G^{-1} H h^{(\ell-1)} + G_r^{-1}$$

After (ℓ) steps we will have solved the system Mh = r with

$$M = G \left(\sum_{i=0}^{\ell-1} \left(G^{-1} H \right)^i \right)^{-1}$$

The simplest form of preconditioning is the Point Jacobi type corresponding to the splitting: G = diag(P), but the convergence is only assured if P is diagonal dominant.

The Gauss-Seidel preconditioning corresponds to the splitting: G = diag(P) + low(P), in this case the relaxation is always stable for the SSPD, but the necessary back substitution is slow on the DAP. [LAILDL88] found that the Point Jacobi preconditioner worked well when it was combined with a diagonal scaling of the original system in the form:

$$\widehat{P}_{ij} = P_{ij} / (P_{ii} P_{jj})^{\frac{1}{2}}$$

On the DAP there is a very fast implementation of the square root, it takes less time than the multiplication.

Iterative methods have also been used for the SSPD for the Primal Newton barrier IPM [GLMSTW86]. The authors showed that the SSPD problem is equivalent to the least squares problem (stated in their notation):

$$\min : \| X(r - A^T q) \|_Z$$

X: is the diagonal matrix of the previous primal variables.
r: is the right hand side "error signal".
q: is the direction vector of the dual variable.

They solve the least squares problem by using the LSQR algorithm by Page and Saunders 1982 [PAGSAU82], which is similar to a Conjugate Gradient (CG) method as applied to the LS problem. A triangular matrix preconditioner was used which they derived from a partial Cholesky factorisation of the SSPD matrix : AX^2A^T . The back substitution which follows from this preconditioning would not be very efficient for the parallel architecture under consideration. However, their work point toward the assumption that: 1) An interative solution to the SSPD is indeed feasible; 2) The least squares algorithm is more reliable for ill-conditioned problems. It is clear from the Newton step of the IPM that it is the directional information in the solution which matters most.

The general Primal-Dual Newton barrier IPM with bounded variables does not readily lead to same least squares problem, but the CG iteration which avoids the forming of the normal equations of the SSPD is still possible, as shown for the algorithm CGLS in [PAGSAU82].

4.3 Design of Data structures for Sparse Matrix Operations

While the DAP is well suited for the commonly used basic linear algebra kernels such a dense matrix-vector and matrix-matrix multiplication, the exploitation of sparsity is not a straightforward matter. Several schemes have been suggested for efficient matrix-vector multiplication. Efficient schemes for structured sparsity have been well exploited, [BARESH84], but an efficient system for large scale LP problems should be able to handle unstructured sparsity.

The handling of sparse matrices always call for considerable computational overhead, especially on a SIMD machine. This means that there often is a threshold density where it is worthwhile to change over to a dense mode of operation. for most large scale LP problems it is meaningless to use a dense mode. A natural idea would be to try to isolate empty blocks in the problem matrix, however the remaining data structures would still be sparse.

These problems have been tackled by Parkinson [PARKIN81] and by Morjaria and Makinson [MORMAK84]. Parkinson's scheme "Long-Vector" uses low level permutation software for GATHER/SCATTER type of operation. The other scheme "2D ARRAY" maps matrix elements onto the same block if indexes are equal modulo 64 (DAP grid size), and if there is a free space. If the location is occupied, a new block is allocated by pointing to the next plane, as in a stack operation. The algorithm for the matrix-vector multiplication scans all the planes in the stack and make the multiplications when the appropriate indices matches. This matching operation is done by using logical masks, an operation which is ideally suited for the bit-sliced architecture of the DAP, [ANDRSN90-2].

Both these schemes have advantages and disadvantages. The "Long-Vector" scheme uses the memory more efficiently, but there are some communication cost due to the many shift operations involved. In the "2D ARRAY" scheme the elements are always placed under the correct PE hence no horizontal shift are neccessary, but the memory could be used inefficiently. If, for instance, one attempts to store a dense row, column or diagonal using the latter scheme, a totally unaceptable situation would arise, where a new plane (4096) words are allocated for each 64 elements. It is therefore a requirements that dense columms, rows and diagonals are taken out of the SSPD matrix and operated on separately.

We have adopted the "2D ARRAY" scheme for unstructured matrix-vector multiply, because of the scope for solving LP's of a very high order, n > 100000 with the advent of the IPM.

5. EMBEDDING INTERIOR SEARCH WITHIN SPARSE SIMPLEX LP SYSTEMS

The system is designed to recover a basis if the IPM is terminated either at an intermediate point or at the optimum solution. The variables which take positive values (between upper and lower bounds) corresponding to a feasible primal solution may be classified in two sets. The first is a set of basic variables which have the corresponding system of nonsingular basis matrix and the second set is made of remaining positive variables called superbasic variables [MURSND78]. If there is one or more superbasic variables in a feasible solution we have the interior of a face, etc., and not an extreme point. FORTLP [MITAMZ88], MINOS [MUTRSU83] and XMP [MARSTN81] allow inclusion of superbasic variables and nonbasic but feasible solutions can be represented in the system together with a reference basis set.

- Recovery of an optimum basis

In order to recover a basis from the (near) optimum solution we construct a set of basis variables and the corresponding basis matrix. If $u_j - \in_{rhs} > x_j > \in_{rhs}$ and $z_j < \in_{rcost}$ then the variable x_j is in the basis set. All the remaining variables are made non basic and for $u_k > x_k > u_k - \in_{rhs}$ the variables x_k are held at their upper bound (and also nonbasic). Inverting to the corresponding basis and recomputing z_j we obtain a corresponding SSX basis restart point.

- Recovery of a basis from an intermediate point

Since we wish to mix and match computational kernels we have designed a basis recovery procedure which takes an intermediate (interior point) solution and constructs a superior extreme point solution. We take all the positive variables within bounds

$u_j - \epsilon_{rhs} > x_j > \epsilon_{rhs}$ where $x_j = x^i_j$ are the solution values of an intermediate (IPM) solution. We flag $x^i_k < \epsilon_{rhs}$ to non-basic variables at zero value and $x^i_k > u_k - \epsilon_{rhs}$ as nonbasic variables in their upper bound. A basis is constructed out of the remaining variables using a CRASH procedure. Finally, the superbasic variables are given their x^i_j values. We now apply the purification procedure to compute a (basic solution) with a superior objective function value. The purification steps are described in detail in [MITAMY88].

Basis recovery is considered to be an important issue in the exploitation of LP solvers in practice and is also discussed in [MARBAL89, GLMSTW86].

6. DISCUSSION AND CONCLUSIONS

In this paper we have considered the issues of algorithm design and implementation of the IPM for a range of computer architectures. We have indicated how such an IPM solver is integrated within an established SSX LP system. Results of our experimental tests involving industry standard test problems [GAYM85, CHKENW89] will be supplied in a forthcoming report.

7. ACKNOWLEDGEMENTS

Mr Johannes Andersen is supported by the Science and Engineering Research Council (UK) CASE studentship with Active Memory Technology (AMT) as the industrial sponsor. Mr Roni Levkovitz is supported by a grant awarded by the Digital Europe External Research Agreement, Digital Equipment Corporation (DEC). The authors gratefully acknowledge the financial and equipment facilities offered by SERC, AMT and DEC. Finally, we thank NAG Ltd for their continued interest and encouragement in the research of our group and for their financial, administrative and technical support sustained over a number of years.

8. REFERENCES

[ADKARV86] Adler, I., Karmarkar, N., Resende, M.G. and Veiga, G., An implementation of Karmarkar's algorithm for linear programming, Report 86-8, Operations Research Centre, University of California, Berkerley, CA., USA, 1986.

[ADKARV89] Adler, I., Karmarkar, N., Resende, M.G. and Veiga, G., Data structures and programming techniques for the implementation of Karmarkar's algorithm, ORSA Journal on Computing, vol 1, No 2, pp 84-106, 1989.

[ALVARD86] Alvardo, F.L., and Betancourt, R., Parallel inversion of sparse matrices, IEEE publication, pp 74-81, Feb.1986.

[ANSTRC86] Anstreicher K.M., A combined phase I - phase II projective algorithm for linear programming, Technical Report, Yale School of Organisation and Management, New Haven, CT, USA, 1986.

[ANDRSN90-1] Andersen, J., Primal-dual interior point method, Internal Report, Dept of Mathematics and Statistics, Brunel University, Uxbridge, Middlesex, UK, 1990.

[ANDRSN90-2] Andersen, J., Sparse Data Structure for the DAP, Internal Report, Dept of Mathematics and Statistics, Brunel University, Uxbridge, Middlesex, UK, 1990.

[ATSUHL87] Aittoniemi, L., and Suhl, U.H., Computing sparse LU-factorization of large scale linear programming bases, ISBN-3-88398-058-7, Arbeitspapier Nr 58/87, Freie Universitaet, Berlin, W. Germany, 1987.

[BARESH84] Barlow, R.H., Evans, D.J., Shanehchi, J., Sparse Matrix Vector Multiplication on the DAP, Super Computers and Parallel Computation, Clarendon Press, Oxford, ppl49-155,1984.

[BERTSK89] Bertsekas, D.P., and Tsitsiklis, J.N., Parallel and Distributed Computation - Numerical Methods, Prentice Hall,1989.

[BIXBY89] Bixby, R., Discussion of price strategy within the CPLEX system, private communication, ORSA/TIMS meeting, New York, USA, October 1989.

[CHKENW89] Carolan, W.J., Hill, J.E., Kennington, J.L., Niemi, S., and Wichmann, S.J., An empirical evaluation of the KORBX algorithms for military airlift applications, Technical Report 89-OR-06, Dept. of Computer Science and Engineering, Southern Methodist University, Dallas, 1989.

[COTTLE74] Cottle, R.W., Manifestation of the Schur complement, Linear Algebra Applications, vol 8, pp 189-211, 1974.

[DIKIN67] Dikin, I.I., Iterative solution of problems of linear and quadratic programming, Soviet Mathematics, Doklady, vol 8, pp 634-675, 1967.

[DUFERD86] Duff, I.S., Erisman, A.M., and Reid, J.K., Direct method for sparse matrices, Oxford Science Publications, Clarendon Press, Oxford, 1986.

[FLYNN72] Flynn, M.J., Some computer organisations and their effectiveness, IEEE Transactions on Computers, C-21(q), pp 948-960, 1972.

[FORTOM89] Forrest, J.J.H. and Tomlin, J.A., Mathematical programming with a library of optimisation subroutines, presented at ORSA/TIMS, New York, USA, 1989.

[GAYM85] Gay, D.M., Electronic mail distribution of linear programming test problems, Mathematical Programming Society, COAL Newsletter, 1985.
[GEOLIU81] George, J.A. and Liu, J.W., Computer solution of large sparse positive definite systems, Prentice Hall, 1981.
[GHLIUM89] George, J.A., Heath, M., Liu, J.W., Ng, E., Solution of sparse positive definite systems on a Hypercube, Journal of Computational and Applied Mathematics, pp 129-156, 1989.
[GHVIAL86] De Ghellinck, G. and Vial, J.F., A polynomial Newton method for linear programming, Algorithmica, vol 1, pp 425-454, 1986.
[GILMSW84] Gill, P.E., Murray, W., Saunders, M.A., and Wright, M.H., Sparse matrix methods in optimization, SIAM Journal of SCI STAT COMPUT, vol 5, No 3, pp 562-589, 1984.
[GLMSTW86] Gill, P.E., Murray, W., Saunders, M.A., Tomlin, J.A., and Wright, M.H., On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method, Mathematical Programming, vol 36, pp 183-209, 1986.
[GOLOAN83] Golub, G.H., and Van Loan, C.F., Matrix Computations, North Oxford Academic,1983.
[GOLTOD88] Goldfarb, D. and Todd, M.J., Linear Programming, Technical Report No. 777, School of Operations Research, Cornell University, Ithaca, N.Y., USA, 1988.
[GONZAG87] Gonzaga, C.C., A conical projection algorithm for linear programming, Technical Report, Dept of Electrical Engineering and Computer Science, University of California, Berkley, USA, 1987.
[HOCJES88] Hockney, R.W. and Jesshope, C.R., Parallel Computers 2: Architecture, Programming and Algorithms, 2nd edition, Adam Hilger, 1988.
[HUNT89] Hunt, D.J., AMT DAP - a processor array in a workstation environment, Computer Systems Science and Engineering, vol 4, No 2, pp 107-114, 1989.
[KARMAR84] Karmarkar, N., A new polynomial time algorithm for linear programming, Combinatorica, vol 4, pp 373-394, 1984.
[KOJMY087] Kojima, M., Mizuno, S., and Yosishe, A., A primal-dual interior point algorithm for linear programming, Technical Report, Dept. of Information Sciences, Tokyo, Japan, 1987.
[LAILDL88] Lai, C.H., Liddell, H.M., Preconditioned Conjugate Gradient Methods on the DAP, Proceedings from The Mathematics of Finite Elements & Applications vol 4, pp 147-156, 1988.
[LEVANM89-1] Levkovitz, R., Andersen, J., Mitra, G., Algorithms for inverting a Sparse, symmetric positive definite matrix on parallel computers, Internal Report, Dept of Mathematics and Statistics, Brunel University, Uxbridge, Middlesex, UK, 1989.
[LEVANM89-2] Levkovitz, R., Andersen, J., Mitra, G., An algorithm for inverting a sparse, symmetric positive definite matrix with a fixed non zeros structure, Internal Report, Dept of Mathematics and Statistics, Brunel University, Uxbridge, Middlesex, UK, 1989.

[MURSND78] Murtagh, B.A., and Saunders, M.A., Large scale linearly constrained optimization, Mathematical Programming, vol 14, pp 41-72, 1978.
[MURTAG81] Murtagh, B.A., Advanced linear programming computation and practice, McGraw Hill, New York, 1981.
[MURTSU83] Murtagh, B.S., and Saunders M.A., MINOS 5.1 users guide, Technical Report SOL 83-20R, Dept of Operational Research, Stanford University, USA, 1983.
[ORCHAY68] Orchard-Hays, W., Advanced computing techniques in linear programming, McGraw-Hill, New York, USA, 1968.
[PAGSAU82] Paige, C.C., and Saunders, M.A., LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares, ACM Transactions on Mathematical Software, vol 8, pp 43-71, 1982.
[PARKIN81] Parkinson, D., Sparse matrix vector multiplication on the DAP, Technical Report, DAP Support Unit - Queen Mary College, London, UK, 1981.
[PERROT89] Perrot, R., Editor, Software for parallel machines, Proceedings of UNICOM seminar, London, UK, June, 1989.
[RENEGR88] Renegar, J., A polynomial-time algorithm based on Newton's method for linear programming, Math. Programming, vol 40, pp 59-93, 1988.
[ROOHRT89] Roos, C., and Hertog, D., A survey of search directions in interior point methods for linear programming, Report 89-65, Faculty of Technical Mathematics and Informatics, Delft University, Holland, 1989.
[ROOVAL88] Roos, C., and Vial, J.P., A polynomial method of approximate centres for linear programming, Report No. 88-68, Faculty of Technical Mathematics and Informatics, Delft University, Holland, 1988.
[SPCONF89] SIAM Conference on sparse matrices, Salishan College, Oregon, USA, May 1989.
[TAMIZ86] Tamiz, M., Design implementation and testing of a general linear programming system exploiting sparsity, PhD Thesis, Brunel University, UK, 1986.
[TCWSKI83] Toczylowski, E., A hierarchical representation of the inverse for sparse matrices, SIAM J. Alg. Disc Methods, 1983.
[VANMKF86] Vanderbei, R.J., Meketon, M.S., and Freedman, B.A., A modification of Karmarkar's algorithm, Algorithmica, vol 1, pp 395-409, 1986.
[VMSRTL88] 1. VAX VMS RTL Parallel processing (PPL$) manual, 2, Digital Equipment Corp., April 1988.
[YEKOJM87] Ye, Y., and Kojima, M., Recovering optimal dual solutions in Karmarkar's polynomial algorithm for linear programming, Mathematical Programming, vol 39, pp 305-317, 1987.
[ZMWSKI89] Zmijewki, E., Limiting communication in parallel sparse Cholesky factorisation, UCSB, June 1989.

[LIU89] Liu, J.W.H., Reordering sparse matrices for parallel elimination, Parallel computing, vol 11, No. 1, 1989.
[LUSMSH89] Lustig, I.J., Marsten. R.E., and Shanno, D.F., Computational experience with a primal-dual interior point method for linear programming, Technical Report SOR 89-17, School of Industrial and Operations Research, Georgia Institute of Technology, Atlanta, USA, 1989.
[MARBAL88] Marsten, R.E., Saltzman, M.J., Shanno, D.F., Pierce, G.S. and Ballintijn, J.F., Implementation of a dual affine interior point algorithm for linear programming, ORSA Journal on Computing, vol 1, No. 4, pp 287-297, 1989.
[MARSHN89] Marsten, R.E., and Shanno D.F., Interior point methods for linear programming: ready for production use, Practitioner Workshop, ORSA/TIMS New York, USA, October, 1989.
[MARSTN81] Marsten, R.E., The design of XMP linear programming library, ACM Transactions on Mathematical Software, 1981.
[MCMNSH89] McShane, K.A., Monma, C, and Shanno, D.F., An implementation of a primal-dual interior point method for linear programming, ORSA Journal on Computing, vol 1, No 2, pp 70-82, 1989.
[MEGSHB86] Megiddo, N., and Shub, M., Boundary behaviour of interior point algorithms in linear programming, Technical Report, RJ5319, IBM, York Town Heights, New York, USA,1986.
[MITAMY88] Mitra, G. Tamiz, M. and Yadegar, J., Experimental investigation of an interior search method within a simplex framework, Communications of the ACM, vol 31, No 12, pp 1474-1482, 1988.
[MITAMZ88] Mitra, G. and Tamiz, M, (1988), FORTLP user manual, Brunel University and NAG Ltd., 1988.
[MITAMZ90] Mitra, G. and Tamiz, M., Alternative methods for representing the inverse of linear programming basis matrices, to appear in the Progress in Mathematical Programming 1975-1989, Special publication, Australian Society of Operational Research, Editor S. Kumar, 1990.
[MNADLR87] Monteiro, R.D.C., and Adler, I., Interior path following primal-dual algorithms - Part 1 linear programming, Technical Report, Dept. of Industrial Engineering and Operations Research, University of California, Berkley, CA 94720, USA, 1987.
[MONADR88] Monteiro, R.D.C., Adler, I., and Resende, M.G.C., A polynomial time primal-dual affine scaling algorithm for linear and convex programming, and for power series expansion, Technical Report ESRC 88-8, University of California, Berkerley, USA, 1988.
[MORMAK84] Morjaria, M., and Makinson, G.J., Unstructured Sparse matrix Vector Multiplication on the DAP, Super Computers and Parallel Computation, Clarendon Press, Oxford, pp 157-166, 1984.

APPENDIX 1. ACRONYMS USED

BBLT: Block bordered lower triangular (form)

BDK: Block diagonal kernels (of a matrix)

DAP: Distributed array processor (computer)

IPM: Interior point method

MIMD: Multiple instruction multiple data (machine)

PE: Processing elements (of the DAP computer)

SISD: Single instruction single data (machine)

SIMD: Single instruction multiple data (machine)

SU: Sparse unsymmetric (system of equations)

SSPD: Sparse symmetric positive definite (system of equations)

SSX: Sparse simplex (method)