EFFICIENT PARALLEL LINEAR PROGRAMMING

Victor PAN *
Computer Science Department, State University of New York, Albany, NY 12203, USA

John REIF **
Aiken Computation Laboratory, Division of Applied Sciences, Harvard University, Cambridge, MA 02138, USA

Received October 1985
Revised June 1986

Linear programming and least squares computations are accelerated using authors' parallel algorithms for solving linear systems. The implications on the performance of the Karmarkar and the simplex algorithms for dense and sparse linear programs are examined. The results have further applications to combinatorial algorithms.

linear programming • least squares • parallel algorithms

1. Introduction

Numerous practical computations require to find a least squares solution to an *overdetermined* system of linear equations, $Ax = b$, that is, to find a vector x of dimension n that minimizes $\| Ax - b \|$ given an $m \times n$ matrix A and a vector b of dimension m where $m \geq n$. (Here and hereafter we apply the Euclidean vector norm and the associated 2-norm of matrices (Golub and van Loan [5]).) Such a problem is called the *linear least squares problem*, l.l.s.p. In particular solving a linear system $Ax = b$ in the usual sense is a simplification of the l.l.s.p. where the output is either the answer that $\min_x \| Ax - b \| > 0$ or, otherwise, a vector x^* such that $Ax^* - b = 0$.

The first objective of this paper is to re-examine the time-complexity of the l.l.s.p. and to indicate the possibility of speeding up its solution using the parallel algorithms of Pan and Reif [18] combined with the techniques of *blow up transformations* and of *variable diagonals* and with the Sherman–Morrison–Woodbury formula. As a major consequence (which may become decisive for determining the best algorithm for the *linear programming problem* (l.p.p.), at least over some important classes of instances of that problem), we will substantially speed up the algorithm of Karmarkar [7] for the l.p.p., because solving the l.l.s.p. constitutes the most costly part of every iteration of that algorithm. Furthermore we will modify Karmarkar's algorithm and will solve an l.p.p. with a dense $m \times n$ input matrix using $O(Lm \log^2 m)$ parallel arithmetic steps and $m^{2.5}$ processors where the parameter L (defined in [7]) represents the input size of the problem. Applying fast matrix multiplication algorithms we may decrease the above processor bound in the dense case, as well as the asymptotic sequential time bound of [7], by a factor of $m^{0.335}$ (preserving the best asymptotic parallel time). In fact, improving (see Strassen [23]), or even combining the known fast matrix multiplication methods (see Gartenberg [3], Lotti and Romani [10], Pan [13]), may lead to further decrease of processor bounds, but the latter decrease, as well as the improvement by a factor of $m^{0.335}$, would hardly have any practical value due to a huge overhead of the asymptotically fast matrix multiplication methods and their inability to preserve sparsity. Our acceleration of Karmarkar's, however,

* Supported by NSF Grants MCS 8203232 and DCR 8507573.
** This work was supported by Office of Naval Research Contract N00014-80-C-0647.

is practical and most significant in the important case (arising, for instance, in the optimization of economy consisting of several branches weakly connected to each other and in the multicommodity flow problem in a planar network for a fixed number of commodities (see Gondran and Minoux [6] or Murty [11, p. 391]), where the input matrix of l.p.p. is large and sparse and is associated with graphs having a family of small separators (see the formal definitions below, in Section 3).

Our work has several further impacts. Similarly to the case of the algorithm of [7], we may immediately improve the performance of several known algorithms, in particular of algorithms for systems of linear inequalities (Pan [14]), for mathematical programming (Shor [22]), and for sparse non-symmetric systems of linear equations, because (as we indicated above) solving a system of linear equations constitutes a particular case of the l.l.s.p. where $\min_x \|Ax-b\| = 0$. The latter observation leads to a very wide range of applications of our results, including in particular the acceleration of *the simplex algorithms* for a sparse l.p.p. (compare Chvatal [2], Murty [12]). Further applications may include several combinatorial computations. This is demonstrated in Pan and Reif [19], where, relying on the latter improvement of the algorithms for sparse non-symmetric systems of linear equations, we extend the parallel nested dissection algorithm of [18] to the path algebra computations.

We organize the paper as follows. In the next section we recall two known representations of the l.l.s.p., using normal equations and their blow-up transformations. In Section 3 we re-examine the computational cost of sequential algorithms for l.l.s.p.; in particular, we recall the sequential nested dissection algorithm of Lipton, Rose and Tarjan [8] and adjust it to the case of l.l.s.p. In Section 4 we estimate the cost of performing our parallel algorithm for the same problem. In Section 5 we consider one of the major applications of our results, that is, to the acceleration of Karmarkar's algorithm. In the appendix we will briefly comment on the current estimates for the computational cost of solving the l.p.p.

2. Linear least squares problem (l.l.s.p.)

We will use the known fact (see [5]) that the l.l.s.p. can be reduced to computing solution x to the system of normal linear equations

$$A^TAx = A^Tb, \tag{1}$$

which can be reduced to the following system of linear equations in s and y:

$$D_1D_1^Ts + D_1AD_0y = D_1b, \quad D_0^TA^TD_1^Ts = \theta,$$

or, equivalently,

$$Hv = d, \tag{2}$$

where

$$H = \begin{bmatrix} D & D_1AD_0 \\ D_0^TA^TD_1^T & O \end{bmatrix}, \quad v = \begin{bmatrix} s \\ y \end{bmatrix}, \quad d = \begin{bmatrix} D_1b \\ 0 \end{bmatrix}, \quad x = D_0y, \quad D = D_1D_1^T,$$

D_0 is an $n \times n$ matrix, D_1 is an $m \times m$ matrix, D_0, D_1 are non-singular. Here and hereafter I, W^T, v^T, O and θ denote the identity matrix, the transposes of a matrix W and of a vector v, the null matrix and the null vector of appropriate sizes, respectively. Hereafter W^{-T} will denote the inverse of W^T.

If we need to solve the linear system $Ax = b$ in the usual sense, then that system can be equivalently rewritten as $GAx = Gb$, for any non-singular matrix G. The latter system is equivalent to (2) where in that case D can be any $m \times m$ matrix, not necessarily $D_1D_1^T$.

Remark 1. Even though the systems (1) and (2) are equivalent to each other, it is more convenient to apply some algorithms to system (2) than to (1) where H is more sparse and/or better structured than A^TA. (We will call the transition from (1) to (2) and similar transformations *the blow-up transformations* of linear systems.) The simplest and the most customary choice for D_0 and D_1 in (2) is the identity matrices I;

however, choosing appropriate diagonal matrices for D_0 and D_1 we may scale the rows and columns of the three blocks of H in order to stabilize some special algorithms for sparse linear systems (2), such as the nested dissection algorithm; see below. That stabilization can be combined with the customary techniques of threshold pivoting, used in sparse matrix computations at the stage of determining the elimination ordering (Pissanetsky [21]; see also [20] on the stabilization based on variable diagonal techniques and Sherman–Morrison–Woodbury formula).

Remark 2. In some cases further equivalent transformations of the systems (1), (2) are effective; see [20].

3. Sequential computational complexity of the l.l.s.p.

For an l.l.s.p. with a dense matrix A, its solution can be obtained from (1) using $O(m/n)M(n)$ arithmetic operations where $M(n)$ is the cost of $n \times n$ matrix multiplication, $M(n) \leq 2n^3 - n^2$. Theoretically $M(n)$ is at least as small as $o(n^{2.48})$, [23], but that bound is not practical due to the huge overhead constants hidden in that 'o', [13], and because the underlying algorithms do not exploit sparsity of input matrices.

If the matrix A is sparse, the solution can be accelerated using some special methods; see Björck [1]. In particular applying the conjugate gradient method or the Lanczos method (see [1,5]) we may reduce the cost of solving both the system (1) and (consequently) an l.l.s.p. to $O(mN(A))$ arithmetic operations where $N(A)$ is the number of non-zero entries of A, provided that the multiplication by 0 and the addition of 0 are cost-free operations.

We will single out a more specific case encountered in many practical instances of the l.l.s.p., that is, in the instances where the matrix A is sparse and where furthermore the graph $G = (V, E)$ associated with the matrix H has an $s(m+n)$-separator family with $s(m+n) = o(m+n)$. (Hereafter we will assume that $s(k) \geq \sqrt{k}$.) Here and hereafter we apply the two following definitions, which we reproduce from [18, Sect. 1.2] (compare also [8]).

Definition 1. Let C be a class of undirected graphs closed under the subgraph relation, that is, if $G \in C$ and G' is a subgraph of G, then $G' \in C$. The class C is said to have a *dense family of $s(n)$-separators* or, simply, *an $s(n)$-separator family* if there exist constants $n_0 > 0$ and α, $0 < \alpha < 1$, such that for each graph $G \in C$ with $n \geq n_0$ vertices there is a partition V_1, V_2, S of the vertex set of G such that $|V_1| \leq \alpha n$, $|V_2| \leq \alpha n$, $|S| \leq s(n)$, and G has no edge from a vertex of V_1 to V_2 (then S is said to be an *$s(n)$-separator* of G). An undirected graph is said to *have an $s(n)$-separator family* if the class of all its subgraphs has an $s(n)$-separator family.

Binary trees obviously have a 1-separator family. A d-dimensional grid (of a uniform size in each dimension) has an $n^{1-(1/d)}$-separator. Lipton and Tarjan [9] show that the planar graphs have a $\sqrt{8n}$-separator family and that every n-vertex finite element graph with $\leq k$ boundary vertices in every element has a $4\lfloor k/2 \rfloor \sqrt{n}$-separator.

Definition 2. Given a $k \times k$ symmetric matrix $W = [w_{ij}]$, we define $G(W) = (V, E)$ to be the undirected graph with vertex set $V = \{1, \ldots, k\}$ and edge set $E = \{\{i, j\} \mid w_{ij} \neq 0\}$.

The very large linear systems $Ax = b$ that arise in practice often have graphs $G(A)$ with small separators. Important examples of such systems can be found in circuit analysis (e.g., in the analysis of the electrical properties of a *VLSI* circuit), in structural mechanics (e.g., in the stress analysis of large structures), and in fluid mechanics (e.g., in the design of airplane wings and in weather prediction). Similarly the associated graphs $G(A^TA)$, $G(H)$ of the systems (1), (2) frequently have small separators. In all such cases highly effective algorithms can be used. In particular, when the associated graph G of the matrix H of (2) has an $s(m+n)$-separator family, the application of the techniques of nested dissection (see [1, p. 182], George [4], [8]) decreases the cost of the solution of the system (2), and consequently of the original l.l.s.p., to $O(|E|+M(s(m+n)))$ arithmetic operations where $|E|$ is the cardinality of the edge set of G [8]. This is the cost of computing the LDL^T-factorization of H; this cost is much lower than $M(m+n)$, the cost in case of dense H [5]. The subsequent evaluation of the vectors v satisfying (2) costs $O(|E|+(s(m+n))^2)$ arithmetic operations [8], so the approach is particularly effective for solving several systems (1) with fixed A and variable b. Similarly for the linear systems (1) whose graph $G(A^TA)$ has an $s(n)$-separator family, computing the LDL^T-factorization of A^TA costs $O(|E|+M(s(n)))$ operations and the subsequent solution of (1) costs $O(|E|+(s(n))^2)$; in that case stability of the nested dissection algorithm is guaranteed and using system (1) should be preferred; compare Remark 3 below. To see the potential advantage of using the nested dissection algorithm, assume that $m=O(n)$ and that the associated graph G of H is planar. Then G has $O(\sqrt{n})$-separators [8], and $|E|=O(n)$, so computing the LDL^T-factorization of H costs $O(n^{1.5})$ arithmetic operations, and the subsequent solution of (1) costs only $O(n)$ for every fixed vector b, to compare with $O(n^3)$ arithmetic operations, required for the solution if the sparsity is not exploited, and with $O(n^2)$ arithmetic operations required by the conjugate gradient and Lanczos methods.

Remark 3. The system (2) is not positive definite, so the nested dissection algorithm may involve destabilizing elimination steps, characterized by small magnitudes of pivot elements; see Remark 1 and [20] for partial remedies.

4. Parallel algorithms for l.l.s.p.

For large input matrices A, the sequential algorithms for the l.l.s.p. can be prohibitively slow. Their dramatic acceleration that preserves their efficiency can be obtained using the recent parallel algorithms of [18], where in each step every processor may perform one arithmetic operation. Specifically, before [18] appeared, the best algorithms for solving a linear system with an $n\times n$ dense matrix A either (i) were unstable and required $O(\log^2 n)$ parallel steps and $\geq\sqrt{n}\,M(n)$ processors, or (ii) involved $\geq n$ steps and n^2 processors. (Here and hereafter the numbers of processors are defined within constant factors for we may save processors using more steps. Practically this means that the user, having, say, k times less processors than in our subsequent estimates, may still use our algorithms; the parallel time, even increased by a factor of k, may still be attractively small for that user.) The stable iterative algorithm of [18], based on Newton's iteration for the matrix equation $I-XA=0$, requires only $O(\log^2 n)$ steps, $M(n)/\log n$ processors to compute the solution of such a dense system (with the relative error norm bounded, say, by $1/2^{n^{100}}$), provided that the system has a well-conditioned or a strongly diagonally dominant matrix. (In fact the algorithm even inverts the matrix of the given system for the above parallel cost.) That algorithm successively computes $t=\|A\|_1\|A\|_\infty$, $B_0=(1/t)A^T$, $B_{k+1}=2B_k-B_kAB_k$, $k=0,1,\ldots,q$. B_q is shown to be a very high precision approximation to B^{-1} already if $q=O(\log n)$ and if cond(A) is bounded by a polynomial in n (similarly if B is strongly diagonally dominant, that is, if $\|I-XA\|_1<1-1/n^c$ or if $\|I-XA\|_\infty<1-1/n^c$ for a positive constant c). The desired estimates for the parallel complexity of solving dense linear systems immediately follow. Applying the cited algorithm to the system (1), we solve the original l.l.s.p. using $O(\log m+\log^2 n)$ steps, $(M(n)/\log n)(1+m/(n\log n))$ processors. These are the bounds in the case where A is a general (dense) matrix.

Another parallel algorithm of [18] is applied to the systems (2) in the cases of practical interest, where A is sparse and the graph $G=(V,E)$ of H has an $s(m+n)$-separator family; see Definitions 1 and 2. In that case the parallel nested dissection algorithm of [18] computes a special recursive $s(m+n)$-factorization of the matrix H of (2) using $O(\log m\log^2 s(m+n))$ parallel steps and $|E|+M(s(m+n))/\log s(m+n)$ processors, the observations of Remarks 1 and 3 are still applied. (We should apply the same algorithm to the system (1), with even slightly better results, that is, we may replace $m+n$ by n in the cost estimates and improve the stability, provided that the associated graph of (1) has a small separator family.) Following [18, Definition 4.1], we define such a recursive $s(m+n)$-factorization of H as a sequence of matrices $H_0, H_1, \ldots, H_d$ such that $H_0 = PHP^{\mathrm{T}}$, P is an $(m+n)\times(m+n)$ permutation matrix,

$$H_g = \begin{bmatrix} X_g & Y_g^{\mathrm{T}} \\ Y_g & Z_g \end{bmatrix}, \quad Z_g = H_{g+1} + Y_g X_g^{-1} Y_g^{\mathrm{T}} \tag{3}$$

for $g = 0, 1, \ldots, d-1$, and X_g is a block diagonal matrix consisting of square blocks of sizes at most $s(\alpha^{d-g}(m+n)) \times s(\alpha^{d-g}(m+n))$ where $\alpha^d(m+n) \le n_0$ for constants n_0 and α of Definition 1. The latter inequality implies that the factorization (3) has length $d = O(\log m)$, so the computation of (3) is reduced to $O(\log m)$ parallel steps of matrix multiplication and inversion versus $m+n$ such steps in the sequential nested dissection algorithms, required to compute the LDL^{T}-factorization. The dense blocks of X_g (of sizes $s(\alpha^{d-g}n \times \alpha^{d-g}n)$) are inverted by the cited parallel algorithm of [17] for matrix inversion. This enables us to keep the total cost of computing the recursive factorization (3) as low as stated.

Observe that the definition of a recursive $s(n)$-factorization implies the following identities for $g = 0, \ldots, d-1$:

$$H_g = \begin{bmatrix} I & O \\ Y_g X_g^{-1} & I \end{bmatrix} \begin{bmatrix} X_g & O \\ O & H_{g+1} \end{bmatrix} \begin{bmatrix} I & X_g^{-1} Y_g^{\mathrm{T}} \\ O & I \end{bmatrix}$$

and hence

$$H_g^{-1} = \begin{bmatrix} I & -X_g^{-1} Y_g^{\mathrm{T}} \\ O & I \end{bmatrix} \begin{bmatrix} X_g^{-1} & O \\ O & H_{g+1}^{-1} \end{bmatrix} \begin{bmatrix} I & O \\ -Y_g X_g^{-1} & I \end{bmatrix}.$$

This reduces solving linear systems with matrix H_g to solving linear systems with matrices X_g and H_{g+1} and finally implies that, although the recursive factorization (3) is distinct from the more customary LDL^{T}-factorization used in the sequential algorithms, both have similar power, that is, when the recursive factorization (3) is available, $O((\log m)(\log s(m+n)))$ parallel steps and $|E| + (s(m+n))^2$ processors suffice in order to solve the system (2) and consequently the original l.l.s.p. In [18], the partition of H_g in (3) for $g = 0, 1, \ldots, d-1$ is defined by appropriate enumeration of the vertices of the graph G. The enumeration, the study of the block diagonal structure of the matrices X_g and the complexity estimates rely on extensive exploitation of the properties of the graph G stated in Definitions 1 and 2.

Comparing the cost bounds of [8] and [18], we can see that the parallelization is efficient, that is, the product of the two upper bounds on the numbers of steps and processors of [18] is equal (within a polylogarithmic factor) to the bound on the number of arithmetic operations in the current best sequential algorithm of [8] for the same problem. The same efficiency criterion is satisfied in the algorithms of [18] inverting an $n \times n$ dense matrix in $O(\log^2 n)$ parallel steps using $M(n)/\log n$ processors. Consequently all our parallel algorithms for an l.l.s.p. are also efficient.

The complexity estimates of [18] have been established in the case of well-conditioned input matrices; the algorithms of [18] output the approximate solutions with a sufficiently high precision. On the other hand, all the estimates have been extended to the case of an arbitrary integer input matrix A in Pan [15,17] by using some different techniques, in particular using variable diagonals. In that case the solutions are computed exactly, although that computation generally involves larger numbers, such as the determinant of the input matrix, det A. That algorithm exactly computes at first det A and adj A and then $A^{-1} = \text{adj } A/\det A$ and $A^{-1}b = (\text{adj } A)b/\det A$. If only a system $Ax = b$ with an $n \times n$ integer matrix A must be solved, only (adj A)b (rather than adj A) should be computed. The evaluation of det A in [15,17] is reduced to computing the Krylov matrix, $K = [v, Av, \ldots, A^{n-1}v]$, and the vector $A^n v$, $v = [1, 0, \ldots, 0]^{\mathrm{T}}$, and to the exact solution of the linear system, $Ky = A^n v$. The solution vector $y = [y_i]$ is the coefficient vector of the characteristic polynomial of A, $\det|\lambda I - A|$, so $y_0 = \det A$, and all the entries of y are integers. At those stages, $O(\log^2 n)$ parallel steps and $M(n)$ processors suffice, provided that K is strongly diagonally dominant, because we may use the algorithm of [18] in order to compute the integer solution vector y with the absolute error norm bound, say 1/3; then we may obtain y exactly by rounding-off. To make the matrix K strongly diagonally dominant, at first we replace A, say by $A + pI$ or, more generally, by a matrix W such that $W = A \bmod p$, so $\det W = \det A \bmod p$ is computed. Then using Newton–Hensel's lifting, we compute $\det W = \det A \bmod p^s$, where $2|\det A| < p^s$, $s = 2^h$, $h = O(\log n)$, so we may recover $\det A$. Similarly we compute adj A or (adj $A)b$. In the worst case this construction requires to choose p as large as n^n. However, with probability $1 - \epsilon(n)$, $\epsilon(n) \to 0$ as $n \to \infty$, it suffices to choose p being a prime of an order of $O((n\|A\|)^{3.1})$ and to define both the Krylov matrix and the vector $W^n v$ modulo p^s. Another approach leads to slightly inferior (with the time bound increased by a factor of $\log n$) but deterministic estimates. It relies on computing LU-factors of A; see [17].

5. Karmarkar's algorithm, parallelization, application to sparse l.p.

In this section we will examine the cost of Karmarkar's linear programming algorithm [7], and of its modifications that use the blow-up transformations, the nested dissection and parallelization. At first we will reproduce that algorithm, which solves the problem of the minimization of the linear function $c^{\mathrm{T}}y$ subject to the constraints

$$A^{\mathrm{T}}y = 0, \quad \sum_j y_j = 1, \quad y \geq 0, \tag{4}$$

where $y = [y_j, j = 0, 1, \ldots, m-1]$ and c are m-dimensional vectors, A^{T} is an $n \times m$ matrix, $m \geq n$, y is unknown. This version is equivalent to the canonical linear programming problem of the minimization of $c^{\mathrm{T}}y$ subject to $A^{\mathrm{T}}y \leq b$, $y \geq 0$; see [7] and compare [2,12]. We will designate $e = [1, 1, \ldots, 1]^{\mathrm{T}}$,

$$\begin{aligned} &y(i) = [y_0(i), y_1(i), \ldots, y_{m-1}(i)]^{\mathrm{T}}, \\ &D(0) = I, \quad D(i) = \mathrm{diag}(y_0(i), y_1(i), \ldots, y_{m-1}(i)), \\ &B^{\mathrm{T}} = B^{\mathrm{T}}(i) = \begin{bmatrix} A^{\mathrm{T}}D(i) \\ e^{\mathrm{T}} \end{bmatrix}, \qquad i = 0, 1, \ldots \end{aligned} \tag{5}$$

(All the diagonal matrices $D(i)$ encountered in the algorithm of [7] are positive definite.) The algorithm proceeds as follows:

```
**initialize** CHOOSE $\epsilon > 0$ (prescribe tolerance) and a parameter $\beta$ (in particular, $\beta$ can be set equal to
$\frac{1}{4}$). Let $y(0) = (1/n)e$, $i = 0$.
**recursive step** While non-optimal ( $y(i)c^{\mathrm{T}}y(i) > \epsilon$) and while the infeasibility tests fail **do**
  Compute the vector $y(i+1) = \gamma(y(i))$, increment $i$.
  Given vector $y(i)$, the vector $y(i+1)$ is computed as follows:
    (1) Compute the matrix $B = B(i)$ of (5), that is, compute the matrix $A^{\mathrm{T}}D(i)$ and augment it by
        appending the row $e^{\mathrm{T}}$.
    (2) Compute the vector $c_p = [I - B(B^{\mathrm{T}}B)^{-1}B^{\mathrm{T}}]D(i)c$.
    (3) Compute the vector $z(i) = y(i) - \beta r c_p/\|c_p\|$ where $r = 1/\sqrt{m(m-1)}$.
    (4) Compute the vector $y(i+1) = D(i)z(i)/e^{\mathrm{T}}D(i)z(i))$.
```

The algorithm includes the checks for infeasibility and optimality (see [7]), but it is easy to verify that their computational costs, as well as the computational cost of the reduction of the problem from the canonical form to (4), are dominated by the cost of computing the vector $\gamma(y(i))$ at the recursive steps, which is, in turn, dominated by the cost of computing $(B^{\mathrm{T}}B)^{-1}$ given $B^{\mathrm{T}} = B^{\mathrm{T}}(i)$ for all i. [7] shows that $B^{\mathrm{T}}B$ can be represented as follows:

$$B^{\mathrm{T}}B = \begin{bmatrix} A^{\mathrm{T}}D^2(i)A & 0 \\ 0^{\mathrm{T}} & m \end{bmatrix},$$

so the inversion of $B^{\mathrm{T}}B$ is reduced to the inversion of $A^{\mathrm{T}}D^2(i)A$, which in turn is reduced to the inversion of the matrix H of (2) where A is replaced by $D(i)A$. Furthermore we can see that it suffices to compute the product $(B^TB)^{-1}BD(i)c$, and this amounts to matrix-by-vector multiplications and to solving a blown-up linear system of the form (2) with the matrix

$$H = H(i) = \begin{bmatrix} D^{-2}(i) & A \\ A^T & O \end{bmatrix}. \tag{6}$$

This algorithm of [7] requires $O(Lm)$ recursive steps in the worst case, so the total computational cost is $O(LmC)$ where L is the input size of the problem and C is the cost of computing $\gamma(y)$ given by y. The algorithm for the incremental computation of the inverse of B^TB of [7, Sect. 6] implies that $C = O(m^{2.5})$ for the dense A. It is rather straightforward to perform these $O(m^{2.5})$ arithmetic operations in parallel using $O(\sqrt{m} \log m)$ steps and $m^2/\log m$ processors (and using $O(m)$ steps, m^2 processors for the initial inversion of A^TA). Applying the matrix inversion algorithms of [18], we may perform every evaluation of $\gamma(y)$ using $O(\log m + \log^2 n)$ parallel arithmetic steps and $(M(n)/\log n)(1 + m/(n \log n))$ processors, so we arrive at the following trade-off for the estimated total arithmetic cost of Karmarkar's algorithm: $O(m^{1.5}L)$ steps, m^2 processors, that is, $O(m^{3.5}L)$ arithmetic operations (via the straightforward parallelization) or $O(mL(\log m + \log^2 n))$ steps, $(M(n)/\log n)(1 + m/(n \log n))$ processors, that is, $O(mLM(n)$ $(\log m + \log^2 n)(1 + m/(n \log n))/\log n)$ arithmetic operations (via the parallel matrix inversion algorithms of [18]).

In both cases the sparsity of A is not exploited. In particular the algorithm for the incremental computation of the inverse suggested in [7, Sect. 6] does not preserve the sparsity of the original input matrix. This causes some difficulties for practical computations, because the storage space increases substantially. Thus the special methods of solving sparse l.l.s.p., such as the conjugate gradient, the Lanczos and the nested dissection methods (see [1,5] and this paper) become competitive with (if not superior to) the latter algorithm of [7, Sect. 6]. If the matrix A is such that the graph $G = (V, E)$ of the matrices H of (6) has an $s(m+n)$-separator family and $s(m+n) = o(m+n)$, then the nested dissection method can be strongly recommended. Specifically, in that case we arrive at the estimates of $O(Lm(|E| + M(s(m+n))))$ arithmetic operations for solving the l.p.p. by combining [7] and [8] (see Section 3), and of $O(Lm \log m \log^2 s(m+n))$ parallel arithmetic steps and $O(|E| + M(s(m+n))/\log s(m+n))$ processors, by combining [7] and the parallel algorithm of this paper. The reader could better appreciate this improvement due to the application of the nested dissection if we recall that $s(m+n) = \sqrt{8(m+n)}$ where the graph G is planar (as occurs in many operations research applications, for instance in the problem of computing the maximum flow in a network having an $s(m+n)$-separator family). Then the processor bound for computing the recursive factorization (3) is less than $2s^3(m+n) = 8\sqrt{8}(m+n)^{1.5}$ and the total number of arithmetic operations is $O(mL(m+n)^{1.5})$ in that case. The premultiplications of A by the non-singular matrix $D(i)$ do not change the separator sets for the graph G, so these sets are precomputed once and for all, which is an additional advantage of using the nested dissection in this case.

Finally we apply fast matrix multiplication algorithms in order to decrease the known theoretical upper bounds on the complexity of solving l.p.p. with a dense input matrix from $O(m^{3.5}L)$ arithmetic operations of [7] to $O(m^{\beta}L)$ where $\beta < 3.165$. Recall that iteration i of [7] can be reduced to inverting the matrix $H(i)$ of [9]; furthermore the diagonal matrix $\Delta(i) = D^{-2}(i) - D^{-2}(i-1)$ has at most $j = O(\sqrt{m})$ non-zero entries for each i, $i = 1, 2, \ldots$ Now we will apply the Sherman–Morrison–Woodbury formula [5, p. 3],

$$(S - UV)^{-1} = S^{-1} + S^{-1}U(I - VS^{-1}U)^{-1}VS^{-1}, \tag{7}$$

which holds for arbitrary matrices S, U and V of appropriate sizes such that $I + VS^{-1}U$ is a non-singular matrix. We will let $U = V$ be a diagonal matrix with at most j non-zero entries, such that $UV = \left[\begin{smallmatrix} \Delta(i) & O \\ O & O \end{smallmatrix}\right]$. Then the computation of $H^{-1}(i)$ given $H^{-1}(i-1)$ is reduced to the inversion of a $j \times j$ submatrix of $I - VH^{-1}(i-1)V$ and to two rectangular matrix multiplications, of the sizes $m \times j$ by $j \times j$ and $m \times j$ by $j \times m$; see [7]. Thus the entire arithmetic cost of one iteration of [7] is dominated by the arithmetic cost, $M(m, j, m)$, of the $m \times j$ by $j \times m$ matrix multiplication. Respectively, the cost of solving the l.p.p. is $O(LmM(m, j, m))$ arithmetic operations or $O(Lm \log^2 m)$ parallel steps, $M(m, j, m)$ processors, where

Table 1

	Arithmetic operations	Parallel steps	Processors
1st iteration of [7]	$O(m^3)$	$O(\log^2 m)$	$m^3/\log m$
Average over n iterations of [7]	$O(m^{2.5})$	$O(\log^2 m)$	$m^{2.5}$
Any iteration of revised simplex algorithms	$O(m^2)$	$O(m)$	m

$j = O(\sqrt{m})$, $M(m, j, m) = O(m^{\beta})$. Surely $\beta \le 2.25$, for $M(m, j, m) = M(j)(m/j)^2 = o(m^{2.25})$, if $j = O(\sqrt{m})$, but in fact β is upper bounded by 2.165 ([3, p. 108], also compare [10]). This implies the sequential time bound $O(Lm^{3.165})$ and consequently (see [18, App. A]) the processor bound $M(m, j, m) = O(m^{2.165})$ (with the parallel time $O(Lm \log^2 m)$) on the complexity of the l.p.p.). As we have mentioned, large overhead makes fast matrix multiplication algorithms non-practical. Note, however, that even with the straightforward matrix multiplication $M(m, j, m) = m^2(2j-1)$, which implies decreasing the parallel cost of one iteration of [7] to $O(\log^2 m)$ parallel steps, $m^{2.5}$ processors (using as many iterations as in [7], that is, $O(Lm)$).

Remark 4. The latter asymptotic complexity estimates (but with double overhead) could be deduced relying on the inversion of $A^{T}D(i)A$.

Appendix: Current computational cost of solving the l.p.p.

In Table 1 we display the estimates for the computational cost of one iteration of the simplex and Karmarkar's algorithms for the l.p.p. having a dense $m \times n$ input matrix A; compare Pan [16]. We will restrict our analysis to the cases where $n \le m = O(n)$. As in [16], we will not use the possible accelerations based on fast matrix multiplication, but now we will apply the results of [15,18] and the improvement of Karmarkar's from the end of the previous section.

There is a certain controversy about the current upper estimates for the number of iterations in the two cited algorithms. The worst case upper bounds, $O(Lm)$ for [7] and 2^m for the simplex algorithms, greatly exceed the number of iterations required where the same algorithms run in practice or use random input instances. This uncertainty complicates the theoretical comparison of the effectiveness of the two algorithms. However, some preliminary comparison can be based on the partial information already available. In particular let us assume the empirical upper bound $O(m \log m)$ on the number of iterations (pivot steps) of the simplex algorithms, cited by some authors who refer to the decades of practical computation; see [2, pp. 45–46], [12, p. 434]. The bound implies that a total of $O(m^3 \log m)$ arithmetic operations suffice in the simplex algorithm vs. $O(m^3)$ used already in the first iteration of [7]. Moreover there are special methods that efficiently update the triangular factorization of the basis matrices used in the simplex algorithms, which further simplifies every iteration of the simplex algorithms in the case of sparse input matrices; see [2, Chs. 7, 24], [12, Ch. 7]. On the other hand, if appropriate *modifications* of Karmarkar's original algorithm indeed run in a sublinear number of iterations (as he reported on at the TIMS/ORSA meeting, Boston, May, 1985, and at the 12th International Symposium on Mathematical Programming, Boston, August, 1985), this would immediately imply a substantial acceleration of the simplex algorithms at least in the cases of (i) parallel computation and dense input matrices (see Table 1), and (ii) both parallel and sequential computations where the graph associated with the matrix H of (2) has an $s(m+n)$-separator family with $s(m+n) = O((m+n)^q)$, $q < 1$ (see the estimates of Section 5).

References

[1] A. Björck, "Methods for sparse linear least squares problems", in: J.R. Bunch and D.J. Rose, eds., *Sparse Matrix Computations*, Academic Press, New York, 1976.

[2] V. Chvatal, *Linear Programming*, Freeman, San Francisco, CA, 1983.

[3] P.A. Gartenberg, "Fast rectangular matrix multiplication", Ph.D. Thesis, Department of Mathematics, University of California, Los Angeles, CA, 1985.

[4] J.A. George, "Nested dissection of a regular finite element mesh", *SIAM J. on Numerical Analysis* 10 (2), 345–367 (1973).

[5] G.H. Golub and C.F. van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, 1983.

[6] M. Gondran and M. Minoux. *Graphs and Algorithms*, Wiley-Interscience, New York, 1984.

[7] N.K. Karmarkar, "A new polynomial time algorithm for linear programming", *Combinatorica* 4 (4), 373–395 (1984).

[8] R. Lipton, D. Rose and R.E. Tarjan, "Generalized nested dissection", *SIAM J. on Numerical Analysis* 16 (2), 346–358 (1979).

[9] R.J. Lipton and R.E. Tarjan, "A separator theorem for planar graphs", *SIAM J. on Applied Math.* 36, 177–189 (1979).

[10] G. Lotti and F. Romani, "On the asymptotic complexity of rectangular matrix multiplication", *Theoretical Computer Science* 23, 171–185 (1983).

[11] K.G. Murty, *Linear and Combinatorial Programming*, Wiley, New York, 1976.

[12] K.G. Murty, *Linear Programming*. Wiley, New York, 1983.

[13] V. Pan, "How to multiply matrices faster", *Lecture Notes in Computer Science*, Vol. 179, Springer, Berlin, 1984.

[14] V. Pan, "Fast finite methods for a system of linear inequalities", *Computers and Mathematics (with Applics.)* 11 (4), 355–394 (1985).

[15] V. Pan, "Fast and efficient algorithms for the exact inversion of integer matrices", *Proc. Fifth Conference on Foundations of Software Engin. and Theor. Computer Science*, Indian Institute of Technology and Tata Institute of Fundamental Research, New Delhi, Dec. 1985.

[16] V. Pan, "On the complexity of a pivot step of revised simplex algorithm", *Computers and Mathematics (with Applics.)* 11 (11), 1127–1140 (1985).

[17] V. Pan, "Parallel complexity of polylogarithmic time matrix computations", Technical Report 86-14, Computer Science Department, SUNY, Albany, NY, April 1986.

[18] V. Pan and J. Reif, "Fast and Efficient parallel solution of linear systems", Technical Report TR-02-85, Center for Research in Computer Technology, Aiken Computation Laboratory, Harvard University, Cambridge, MA, 1985. Short version in *Proc. 17-th Ann. ACM STOC*, Providence, RI, 143–152.

[19] V. Pan and J. Reif, "Extension of the parallel nested dissection algorithm to the path algebra problems", Technical Report 85-9, Computer Science Department, SUNY, Albany, NY, June 1985.

[20] V. Pan and J. Reif, "Efficient parallel linear programming", Technical Report 86-15, Computer Science Department, SUNY, Albany, NY, 1986.

[21] S. Pissanetsky, *Sparse Matrix Technology*, Academic Press, New York, 1984.

[22] N.Z. Shor, "New development trend in nondifferentiable optimization", *Kibernetika* 13 (6), 87–91 (1977). Translation in *Cybernetics* 13 (6), 881–886 (1977).

[23] V. Strassen, "The asymptotic spectrum of tensors and the exponent of matrix multiplication" (extended abstract), submitted for *Proc. 27th Ann. IEEE FOCS Symp.*, Toronto, 1986.