FAST AND EFFICIENT PARALLEL SOLUTION OF SPARSE LINEAR SYSTEMS*

Victor Pan† John Reif‡

Abstract

This paper presents a parallel algorithm for the solution of a linear system $A\mathbf{x} = \mathbf{b}$ with a sparse $n \times n$ symmetric positive definite matrix A, associated with the graph $G(A)$ that has n vertices and has an edge for each nonzero entry of A. If $G(A)$ has an $s(n)$-separator family and a known $s(n)$-separator tree, then the algorithm requires only $O(\log^3 n)$ time and $(|E| + M(s(n)))/\log n$ processors for the evaluation of the solution vector $\mathbf{x} = A^{-1}\mathbf{b}$, where $|E|$ is the number of edges in $G(A)$ and $M(n)$ is the number of processors sufficient for multiplying two $n \times n$ rational matrices in time $O(\log n)$. Furthermore, for this computational cost the algorithm computes a recursive factorization of A such that the solution of any other linear system $A\mathbf{x} = \mathbf{b}'$ with the same matrix A requires only $O(\log^2 n)$ time and $(|E|/\log n) + s(n)^2$ processors.

keywords: parallel algorithms, sparse linear systems, nested dissection, parallel complexity, graph separators

AMSMOSS: 68Q25, 68Q22, 65Y05, 65Y20, 65F05, 65F50

*Received by the editors March 16, 1987; accepted for publication (in revised form) June 9, 1992. The results of the preliminary version of this paper were presented at the 17th Annual ACM Symposium on Theory of Computing, 1985.

†Department of Mathematics, Lehman College, City University of New York, New York, New York 10468, and Department of Computer Science, State University of New York at Albany, Albany, New York 12222. (vpan@lcvax.bitnet). The work of this author was supported by National Science Foundation grants MCS-8203232, DCR-8507573, CCR-8805782, and CCR-9020690 and by Professional Staff Congress–City University of New York awards 661340, 668541, and 669290.

‡Department of Computer Science, Duke University, Durham, North Carolina 27706. The work of this author was supported by National Science Foundation grant CCR-83-09911, Office of Naval Research contracts N00014-87-K-0310 and N00014-88-K-0458, U.S. Air Force Office of Scientific Research contract AFOSR-87-0386, Defense Advanced Research Projects Agency contracts DAAL03-88-K-1095 and N0014-88-K-0458, and National Aeronautics and Space Administration CESDIS subcontract 550-63 5-30428 URSA.

Preprint of paper appearing in SIAM Journal on Computing, Vol 22, No. 6, pp. 1227-1250, December 1993.

1 Introduction

Recently, it has become feasible to construct computer architectures with a large number of processors. We assume the parallel RAM machine model of [BGH] (see also [EG] and [KR]), where in each step each processor can perform a single addition, subtraction, multiplication, or division over the rationals. It is natural to study the efficient use of this parallelism for solving fundamental numerical problems such as the following.

(1) MULT: Given a pair of $n \times n$ rational matrices A and B, output AB.

(2) INVERT: Given an $n \times n$ rational matrix $A = (a_{ij})$, output A^{-1} within a prescribed error bound ϵ if A is well conditioned (see Definition 2.1 below); else output: ILL-CONDITIONED.

(3) LINEAR-SOLVE: Given a well-conditioned $n \times n$ matrix A and a column vector $\mathbf{b}$ of dimension n, find $\mathbf{x} = A^{-1}\mathbf{b}$ within a prescribed error bound ϵ.

Here and hereafter we assume that $\tilde{A}^{-1}$ and $\tilde{\mathbf{x}}$ approximate A^{-1} and $\mathbf{x}$ within ε if $\|\tilde{A}^{-1} - A^{-1}\| \leq \varepsilon \|A^{-1}\|$ and $\|\tilde{\mathbf{x}} - \mathbf{x}\| \leq \varepsilon \|A^{-1}\| \, \|\mathbf{b}\|$ for a fixed pair of consistent matrix and vector norms.

Our main result in this paper is a numerically stable parallel algorithm for LINEAR-SOLVE for a sparse symmetric positive definite linear system that can be solved by a sequential algorithm based on the generalized nested dissection of the associated graph. In the remainder of this introduction and, more formally, in §3 we compare our algorithm with the previous results, specify the complexity estimates, show our improvement, and list some applications. In particular, in §3 we comment on the classes of the LINEAR-SOLVE instances for which our algorithm is effective.

The nested dissection techniques were first proposed in [Ge] for grid graphs and were then extended to graphs with small separators in [LRT] (see also [R] and the excellent text [GeL] and an alternative version of the nested dissection algorithm in [GT]). Many applications to the sciences and engineering require the solution of such large linear systems; such systems are frequently so large that parallel implementation of the (generalized) nested dissection algorithms is necessary in order to make the solution feasible. (We recall some examples of such problems in §3.)

Work on parallel sparse matrix algorithms can be traced back, at least, to [Ca]. The extension of the idea of nested dissection from the sequential to the parallel case was not immediate since many sets of separators must be eliminated in each parallel step. Linear-time parallel algorithms based on the nested dissection of grids were first described in [Li1] and [Ga]. The survey paper [OV] gives references to early attempts at parallelizing the LINEAR-SOLVE algorithms by nested dissection. Here and hereafter, by "parallel nested dissection" we mean a parallel algorithm for solving sparse linear systems and not a parallel algorithm for computing a dissection ordering. The subsequent literature on the parallel implementation of the nested dissection algorithms includes the papers [GHLN] and [ZG], which give a parallel time bound of

$O(\sqrt{n})$ for grid graphs.

In the proceedings version of our paper [PR1], nested dissection was applied for the first time to yield a numerically stable and processor efficient parallel algorithm for sparse LINEAR-SOLVE with poly-log time bounds, thus reaching (within poly-log factors) the *optimum* bounds for both time and the number of processors. Furthermore, our nested dissection parallel algorithm has been applied to a much larger class of graphs than grid graphs, including planar graphs and $s(n)$-separatable graphs (see Definition 3.1 below) with $s(n) = o(n)$, whereas in the previous literature the parallel nested dissection was restricted to grid graphs. Such an enhanced generality required us to exploit the intricate construction of [LRT] (rather than the simpler constructions of the earlier nested dissection papers, more familiar to the numerical analysis audience); to devise the desired processor efficient version of this approach, we had to elaborate the construction of [LRT] by including the recursive factorization of the input matrix and by proving several properties of the associated graphs. This paper assumes the reader has some exposure to graph techniques. These generalizations of the nested dissection algorithms, including the recursive factorization techniques, are required in several important applications, particularly path-algebra computation in graphs (see [T1], [T2], [PR1], [PR4], and §3 below).

Remark 1.1. Some readers may agree to sacrifice the generality of the results in order to simplify the graph techniques involved. Such readers may replace our Definition 3.1 of separators in graphs by the definition from [GT]. The difference between these approaches is that our definition requires the inclusion of the separator subgraph S into both subgraphs G_1 and G_2, otherwise separated from each other by S in the original graph G, whereas the definition of [GT] requires the elimination of all of the vertices of S and all of the edges adjacent to them from both subgraphs G_1 and G_2. The resulting construction of [GT] is a little simpler than ours, and its application decreases by a constant factor the complexity of the performance of the algorithm in the case of planar graphs G, but the results are applied to a class of graphs that is strictly more narrow than the class we address. As in our proceedings paper, [PR1], we extend to parallel computation the more general algorithm of [LRT], rather than one of [GT], but we demonstrate the vertex elimination construction of [GT] for a 7×7 grid graph in our Figs. 1–5 below (in this case, using the version of [GT], rather than ours, made our display simpler and more compact).

[Li2] and [OR] describe two recent implementations of the parallel nested dissection algorithm on massively parallel SIMD machines. The first implementation is very general and applies to any $s(n)$-separatable graph; it runs on the CONNECTION MACHINE, which is a hypercube-connected parallel machine with 65,536 processors; the second implementation is restricted to grid graphs and runs on the MPP, which is a grid-connected parallel machine with 16,384 processors. Both implementations represent a version of the parallel nested dissection algorithm using $O(s(n))$ time and $s(n)^2$ processors (see the end of

§3).

In the papers [PR2]–[PR4], [PR6], [PR7] we extend our parallel nested dissection algorithm to the linear least-squares problem, to the linear programming problem, and to path-algebra computation in graphs; in all these papers the resulting algorithms are ultimately reduced to application of the algorithm of the present paper.

The rest of the paper is organized as follows: In §2 we briefly recall the known parallel algorithms for MULT and INVERT, and we review their complexity estimates. In §3 we recall some definitions and then state our estimates for the complexity of LINEAR-SOLVE. In §4 we present the parallel nested dissection algorithm for LINEAR-SOLVE for the case of sparse symmetric positive definite systems. In §5, we state our main theorem, which provides bounds on the complexity of the nested dissection algorithm. In §§6–8 we prove these bounds. In Remark 6.1 in §6 we comment on the extension of our results to the nonsymmetric sparse linear systems associated with directed graphs.

2 Auxiliary results on matrix multiplication and inversion

Our algorithm for LINEAR-SOLVE recursively reduces the original problem of large size to a sequence of problems of MULT and INVERT of smaller sizes. Let us recall the complexity of the solution of the two latter problems.

Let $M(n)$ denote an upper bound on the number of processors that suffice to multiply a pair of $n \times n$ matrices in $O(\log n)$ time. Here and hereafter the numbers of processors are defined within a constant factor (we assume Brent's (slowdown) scheduling principle of parallel computations, according to which we may decrease the number of processors from P to $\lceil P/s \rceil$ by using s times as many parallel steps for any natural $s \leq P$). By the upper bound of [Ch], obtained by the straightforward parallelization of the algorithm of [Stra1], we may chose $M(n) \leq n^{2.81}$. In [PR1] and [Pan10] we show that if $k \times k$ matrices can be multiplied in $O(k^\beta)$ arithmetic operations and $\beta < \gamma$ for some γ, then we may choose $M(n) \leq n^\omega$ for some $\omega < \gamma$ and for all n. The current best upper bound on β and ω is 2.375 [CW1]; for surveys of the exciting history of the asymptotic acceleration of matrix multiplications, see also [Pan6], [Pan7], or the original works [Stra1] (the first and justly celebrated breakthrough in this area), [Pan1]–[Pan5], [BCLR], [Bi], [Scho], [CW2], [Stra2]. In practice, however, even for matrices of reasonably large sizes, we should only count on $M(n) = n^3/\log n$ or, at best, on $M(n) = O(n^{2.78})$ because of the considerable overhead of the known asymptotically faster algorithms for matrix multiplication (see [Pan7]).

Let us next estimate the complexity of INVERT.

DEFINITION 2.1. We call an $n \times n$ matrix W well conditioned if $\log \operatorname{cond} W = O(\log n)$, where $\operatorname{cond} W = \|W\| \, \|W^{-1}\|$ for a fixed matrix norm (this definition is invariant in l for all l-norms of matrices).

Now we may recall the following estimate from [PR5] based on the algorithm of [Be] (compare [PaS]):

Fact 2.1. The problem INVERT for an $n \times n$ well-conditioned matrix A and for a positive $\varepsilon < 1$ such that $\log\log(1/\epsilon) = O(\log n)$ can be solved within error bound ε by using $O(\log^2 n)$ parallel time and $M(n)$ processors.

Furthermore, matrix multiplication can be reduced to matrix inversion (see [BM, p. 51] or [Pan7]), so that the processor bound, as well as the parallel and sequential time bounds attained this way, are optimal or nearly optimal (to within a factor of $O(\log n)$). In [Pan8] the above results for dense matrices are extended to the exact evaluation of the inverse of A, of the determinant of A, and of all the coefficients of the characteristic polynomial of A in $O(\log^2 n)$ steps by using $M(n)$ processors in the case for which A is an arbitrary matrix filled with integers and such that $\log \|A\| = n^{O(1)}$ (see proceedings papers [GP1, Part 1] and [Pan9], which cite and (partly) reproduce [Pan8], and see also its extensions in [Pan10], [Pan11], and [KS]).

In §3 we state our estimates for the complexity of sparse LINEAR-SOLVE by using the above estimates for the complexity of MULT and INVERT.

Let us point out two alternatives. In the current applications of our algorithm (see the end of §3) we apply Gaussian elimination for matrix inversion, which for an $n \times n$ matrix means $O(n)$ steps and n^2 processors. On the other hand, theoretically, we may rely on the exact evaluation of the inverse of an $n \times n$ matrix over rationals. This problem has interesting combinatorial applications (see [Lo], [GP1], [GP2], [MVV]). The known parallel algorithms for its solution use $O(\log^2 n)$ steps and $n^\alpha M(n)$ processors, where α varies from 1 in [Cs] to $\frac{1}{2}$ in [PrS] and to slightly less than $\frac{1}{2}$ in [GP3]; furthermore, $\alpha = 0$ even for INVERT over the real matrices if we allow randomized Las Vegas algorithms, because of combining [KS] and [Pan11] (see also [KP], [BP]), although the problem of numerical stability arises with all of these matrix inversion algorithms. The parallel cost of solving sparse linear systems varies, respectively, with the change of matrix inversion algorithms.

3 Some definitions and the complexity of sparse LINEAR-SOLVE

To characterize the linear systems $A\mathbf{x} = \mathbf{b}$ that our algorithm solves, we will need some definitions.

DEFINITION 3.1. A graph $G = (V, E)$ is said to have an $s(n)$-*separator family* (with respect to two constants, $\alpha < 1$ and n_0) if either $|V| \leq n_0$ or, by deleting some separator set S of vertices such that $|S| \leq s(|V|)$, we may

partition G into two disconnected subgraphs with the vertex sets V_1 and V_2 such that $|V_i| \leq \alpha|V|$, $i = 1, 2$, and if, furthermore, each of the two subgraphs of G defined by the vertex sets $S \cup V_i$, $i = 1, 2$, also has an $s(n)$-separator family (with respect to the same constants α and n_0). The resulting recursive decomposition of G is known as the *$s(n)$-separator tree*, so that each partition of the subgraph G defines its children in the tree. The vertices of the tree can thus be interpreted as subgraphs of G or as their vertex sets (we will assume the latter interpretation), and the edges of the tree can be interpreted as the separator sets. Then the vertex set V equals the union of all the vertex subsets in V associated with the edges of the $s(n)$-separator tree and with its leaves. We call a graph *$s(n)$-separatable* if it has an $s(n)$-separator family and if its $s(n)$-separator tree is available.

The above definition of a separator tree follows [LT] and includes a separator in each induced subgraph, unlike the definition of [GT] (see Remark 1.1).

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 family. [LRT] shows that the planar graphs have a $\sqrt{8n}$-separator family and that every n-vertex finite element graph with at most k boundary vertices in every element has a $4\lfloor k/2 \rfloor \sqrt{n}$-separator family. An improved construction due to [D] gives a $\sqrt{6n}$-separator family for planar graphs. (Similar small separator bounds have also been derived by Djidjev for bounded genus graphs and for several other classes of graphs.)

DEFINITION 3.2. Given an $n \times n$ symmetric matrix $A = (a_{ij})$, define $G(A) = (V, E)$ to be the undirected graph with the vertex set $V = \{1, \ldots, n\}$ and the edge set $E = \{\{i, j\} | a_{ij} \neq 0\}$. (We may say that A is *sparse* if $|E| = o(n^2)$.)

The very large linear systems $A\mathbf{x} = \mathbf{b}$ that arise in practice are often sparse and, furthermore, 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). These problems require the solution of (nonlinear) partial differential equations, which are then closely approximated by very large linear differential equations whose graphs are planar graphs or three-dimensional grids. Certain weather prediction models, for example, consist of a three-dimensional grid of size $H_1 \times H_2 \times H_3$ with a very large number $n = H_1 H_2 H_3$ of grid points, but this grid has only a constant height H_3, and hence it has an $s(n)$-separator family for which $s(n) \leq \sqrt{H_3 n}$.

Our algorithm for LINEAR-SOLVE is effective for the systems whose associated graphs have $s(n)$-separator families for which $s(n) = o(n)$ and for which $s(n)$-separator trees are readily available. Thus our result can be viewed as a reduction of sparse LINEAR-SOLVE to the problems of (1) computing an $s(n)$-separator tree in parallel and (2) solving dense linear systems of $s(n)$ equations with $s(n)$ unknowns.

Efficient parallel computation of $s(n)$-separator trees is not simple in general, but it is rather straightforward in the practically important cases of grid graphs (see Figs. 1–5); similarly, such computation is simple for many finite element graphs (see [Ge]). The recent $O(\log^2 n)$-time, $n^{1+\varepsilon}$ processor (for any $\varepsilon > 0$) randomized parallel algorithm of [GM1] gives $O(\sqrt{n})$-separator trees for all the planar graphs.

Many very large sparse linear systems of algebraic equations found in practice, such as linear systems arising in the solution of two-dimensional linear partial differential equations with variable coefficients, have associated graphs that are not grid graphs but that have $s(n)$-separators for $s(n) = o(n)$. The correctness of the generalized parallel nested dissection algorithm applied in the case of graphs with $s(n)$-separators (and thus already in the important case of planar graphs) requires a considerably more complex substantiation and a more advanced proof technique than does the case of grid graphs; in particular, a sophisticated inductive argument is required (see §§7 and 8). This occurs because grid graphs are more regular than are general graphs with $s(n)$-separators.

Let us state the complexity estimates for our solution of sparse LINEAR-SOLVE. Our main result is the decrease of the previous processor bound (supporting the poly-log parallel time) from $M(n)$ to $(|E| + M(s))/\log n$, whereas the time bound increases from $O(\log^2 n)$ to $O(\log^3 n)$. (Since in all the known applications of interest $s(n)$ exceeds cn^{δ} for some positive constants c and δ, we will write $O(\log n)$ rather than $O(\log s(n))$ to simplify the notation. Also note that $2|E|$ is roughly the number of nonzero input entries, so that we cannot generally count on decreasing the sequential time (and therefore also the total work, that is, parallel time times the processor bound) below $|E|$.) It follows, for example, that our improvement of the previous processor bound is by a factor of n if $s(n) = n^{1/2}$, $|E| = O(n^{3/2})$. Because practical implementations of the algorithms would be slowed down to satisfy processor limitations of the actual computers (see discussion at the end of this section), we will decrease the processor bound of $M(n)\log^2 n/T(n)$ to $(|E| + M(s(n)))\log^2 n/T(n)$ in our algorithm, provided that it runs in $T(n)$ time, where $T(n) > c\log^3 n$, $c = O(1)$.

Let us comment further on how we arrive at our estimates. In general, the inverse A^{-1} of a sparse matrix A (even of one with small separators) is dense, and, in fact, if $G(A)$ is connected, A^{-1} may have no zero entries. Therefore, it is common to avoid computing the inverse matrix and instead to factorize it. Our algorithm for LINEAR-SOLVE follows this custom: It computes a special recursive factorization of A. For sparse matrices with small separators, our poly-log-time algorithm yields processor bounds that are of an order of magnitude lower than the bounds attained by means of other poly-log-time parallel algorithms, which compute the inverse matrix. Specifically, let an $n \times n$ positive definite symmetric well-conditioned matrix A be given, such that $G(A)$ has an $s(n)$-separator family, its $s(n)$-separator tree is known, and $s(n)$ is of the form αn^{σ} for two constants $\sigma < 1$ and α. Then we first compute a

special recursive factorization of A (within the error bound 2^{-n^c} for a positive constant c) in $O(\log^3 n)$ time by using $M(s(n))/\log n$ processors (see Theorem 5.1 below), and finally we compute the desired solution vector $\mathbf{x} = A^{-1}\mathbf{b}$. The complexity of this final stage is lower than the complexity of computing the recursive factorization.

For comparison the inversion of an $s(n) \times s(n)$ dense matrix is one of the steps of computing the recursive factorization of A, and the current parallel cost of this step alone is at best $O(\log^2 n)$ time and $M(s(n))$ processors (by using the parallel algorithm of [Be], [PR1].) When our special recursive factorization has been computed, the solution of $A\mathbf{x} = \mathbf{b}$ (for any given $\mathbf{b}$) requires only $O(\log^2 n)$ time and $(|E|/\log n) + s(n)^2$ processors. It is interesting that by multiplying our parallel time and processor bounds we arrive at the sequential complexity estimate of $O(M(s(n)) \log^2 n) = o(s(n)^{2.4})$ arithmetic operations, which matches the theoretical upper bound of [LRT].

Let us demonstrate some consequences of the complexity bounds of our algorithm. We will first assume the weak bound $M(n) = n^3/\log n$ for matrix multiplication. It is significant that already under this assumption our parallel nested dissection algorithm, for poly-log time bounds, has processor bounds that substantially improve the previously known bounds. Let G be a fixed planar graph with n vertices given with its $O(\sqrt{n})$-separator tree. (For example, G might be a graph with a $\sqrt{n} \times \sqrt{n}$ grid.) Then, for any $n \times n$ matrix A such that $G = G(A)$, our parallel nested dissection algorithm takes $O(\log^3 n)$ time and $n^{1.5}/\log^2 n$ processors to compute the special recursive factorization of A and then $O(\log^2 n)$ time and n processors to solve any linear system $A\mathbf{x} = \mathbf{b}$ with A fixed. We have the time bounds $O(\log^3(kn))$ and $O(\log^2(kn))$ and the processor bounds $k^3 n^{1.5}/\log^2(kn)$ and kn, respectively, if $G(A)$ is an n-vertex finite element graph with at most k vertices on the boundary of each face. In yet another example, $G(A)$ is a three-dimensional grid, so that it has an $n^{2/3}$-separator family. In this case we have the same asymptotic time bounds as for planar graphs and our processor bounds are $n^2/\log^2 n$ and $n^{1.33}$, respectively. Furthermore, if we use the theoretical bounds for matrix multiplication, say $M(n) = n^{2.4}$, then our processor bounds for computing the special recursive factorization are further decreased to $n^{1.2}$ in the planar case, to $k^{2.4} n^{1.2}$ in the case of the n-vertex finite elements graphs with at most k vertices per face, and to $n^{1.6}$ for the three-dimensional grid.

In the current practical implementations of our algorithm ([LMNOR] and [OR]) we have not reached the poly-log-time bounds because we have simply used the Gaussian elimination rather than Ben-Israel's algorithm at the stages of matrix inversions, and so we achieve time $s(n)$ with $s(n)^2$ processors. The reason for this choice is the limitation on the number of processors that we could efficiently use on the available computers. It is certain, however, that the future parallel computers will have significantly more processors, and then the application of Ben-Israel's algorithm may be preferred; we cannot exactly

estimate the threshold number of processors that would in practice give the edge to Ben-Israel's algorithm over Gaussian elimination, but according to our theoretical estimates the former algorithm improves the parallel time bounds of the latter one if more than $s(n)^2$ processors are available.

4 Outline of the parallel generalized nested dissection algorithm

In this section we fix an undirected graph G having an $s(n)$-separator family (with respect to constants n_0 and α) (see Definition 3.1). Let A be an $n \times n$ real symmetric positive definite matrix with graph $G = G(A)$ (see Definition 3.2). We will describe an efficient parallel algorithm that computes a special recursive factorization of A. With such a factorization available it will become very simple to solve the system of linear equations $A\mathbf{x} = \mathbf{b}$ for any given vector $\mathbf{b}$ (see the last part of Theorem 5.1 below).

DEFINITION 4.1. A *recursive $s(n)$-factorization* of a symmetric matrix A (associated with a graph $G = G(a)$ having an $s(n)$-separator family with respect to two constants α, $\alpha < 1$, and n_0) is a sequence of matrices, $A_0, A_1, \ldots, A_d$, such that $A_0 = PAP^T$, where P is an $n \times n$ permutation matrix, A_h has size $n_{d-h} \times n_{d-h}$,

$$A_h = \begin{bmatrix} X_h & Y_h^T \\ Y_h & Z_h \end{bmatrix}, \qquad Z_h = A_{h+1} + Y_h X_h^{-1} Y_h^T, \qquad h = 0, 1, \ldots, d-1, \quad (1)$$

and X_h is a symmetric block diagonal matrix corresponding to the separators or, for $h = d$, to the vertex sets in V associated with the leaves of the $s(n)$-separator tree and consisting of square blocks of sizes at most $s(n_{d-h}) \times s(n_{d-h})$, where

$$n_d = n, \qquad n_{h-1} \le \alpha n_h + s(n_h), \qquad h = 1, \ldots, d. \tag{2}$$

Here and hereafter, W^T denotes the transpose of a matrix or vector W. (Note that the constant n_0 used in Definition 3.1 for the separator family is also the order of the diagonal blocks of the matrix X_d.) This recursive factorization is said to be *numerically computed* if the computerized approximants of the induced matrices $A_1, \ldots, A_d$ satisfy (4.1) within the error norm 2^{-n^c} (relative to $\|A\|$) for a positive constant c.

Our definition of a recursive $s(n)$-factorization relies on the matrix identities

$$A_h = \begin{bmatrix} I & O \\ Y_h X_h^{-1} & I \end{bmatrix} \begin{bmatrix} X_h & O \\ O & A_{h+1} \end{bmatrix} \begin{bmatrix} I & X_h^{-1} Y_h^T \\ O & I \end{bmatrix} \tag{3}$$

and

$$A_h^{-1} = \begin{bmatrix} I & -X_h^{-1} Y_h^T \\ O & I \end{bmatrix} \begin{bmatrix} X_h^{-1} & O \\ O & A_{h+1}^{-1} \end{bmatrix} \begin{bmatrix} I & O \\ -Y_h X_h^{-1} & I \end{bmatrix}. \tag{4}$$

Here the matrix A_{h+1} defined by (4.1) is known in linear algebra as Schur's complement of X_h, h ranges from 0 to $d-1$, I denotes the identity matrices, and O denotes the null matrices of appropriate sizes. We show in §7 that A_{h+1} also has certain sparsity properties.

The recursive decomposition (4.1)–(4.4) is intimately related to the recursive decomposition of the associated graph $G = G(A)$ defined by (and defining) the $s(n)$-separator tree. Specifically, we will see that the matrices X_h are block diagonal matrices whose blocks for $h = 0, 1, \ldots, d-1$ are associated with the separator sets of level h from the root of the tree, whereas the blocks of X_d are associated with the leaves of the tree.

Given a symmetric $n \times n$ matrix A associated with an $s(n)$-separable graph $G(A)$, we may compute the recursive $s(n)$-factorization (4.1)–(4.4) by performing the following stages:

Stage 0. Compute an appropriate permutation matrix P, matrix $A_0 = PAP^T$, and the decreasing sequence of positive integers $n = n_d, n_{d-1}, \ldots, n_0$ satisfying (4.2) and defined by the sizes of the separators in the $s(n)$-separator family of $G = G(A)$ (as specified below in §7). The permutation matrix P and the integers $n_d, n_{d-1}, \ldots, n_0$ completely define the order of the elimination of the variables (vertices), so that first we eliminate the vertices of G corresponding to the leaves of the $s(n)$-separator tree, then we eliminate the vertices of G corresponding to the edges adjacent to the leaves of the tree (that is, the vertices of the separators used at the final partition step), then we eliminate the vertices of G corresponding to the next edge level of the tree (separators of the previous partition step), and so on; we formally analyze this in §§7 and 8.

Stage $h+1$ ($h = 0, \ldots, d-1$). Compute the matrices X_h^{-1}, $-Y_h X_h^{-1}$ (which also gives us the matrix $-X_h^{-1} Y_h^T = (-Y_h X_h^{-1})^T$) and $A_{h+1} = Z_h - Y_h X_h^{-1} Y_h^T$ satisfying (4.1), (4.3), (4.4) and such that A_{h+1} has size $n_{d-h-1} \times n_{d-h-1}$. (Each of these stages amounts to inversion, two multiplications, and subtraction of some matrices.)

When the recursive factorization (4.4) has been computed, it will remain to compute the vector $\mathbf{x} = A^{-1}\mathbf{b} = P^T A_0^{-1}(P\mathbf{b})$ for a column vector $\mathbf{b}$. This can be done by means of recursive premultiplications of some subvectors of the vector $P\mathbf{b}$ by the matrices

$$\begin{bmatrix} I & O \\ -Y_h X_h^{-1} & I \end{bmatrix}, \quad \begin{bmatrix} X_h^{-1} & O \\ O & A_{h+1}^{-1} \end{bmatrix}, \quad \begin{bmatrix} I & -X_h^{-1}Y_h^T \\ O & I \end{bmatrix}$$

for $h = 0, 1, \ldots, d$. At the stage of the premultiplication by the second matrix above, the premultiplication by X_h^{-1} is done explicitly and the premultiplication by A_{h+1}^{-1} is performed by means of recursive application of (4.4), so that (4.4) defines a simple recursive algorithm for computing $A^{-1}\mathbf{b}$ for any column vector $\mathbf{b}$ of length n, provided that a recursive $s(n)$-factorization (4.1) is given.

It is instructive to compare the recursive $s(n)$-factorization (4.1)–(4.4) with the Cholesky factorization of A_h used in [LRT]. The notations in [LRT] are distinct from ours, but for the sake of making the comparison we assume that the notation is adjusted to the same format. Then we may say that both factorizations rely on the matrix identities (4.3), (4.4) which, in fact, just represent the block Jordan elimination algorithm for a 2×2 block matrix A_h of (4.1). The Cholesky factorization $PAP^T = LDL^T$ is obtained in [LRT] by the application of the Jordan elimination to the matrix PAP^T, which is equivalent to the recursive application of (4.3) to *both* submatrices X_h and A_{h+1}. (This defines L and L^T in factorized form, but the entries of the factors do not interfere with each other, so that all the entries of $Y_h X_h^{-1}$ coincide with the respective entries of L.) Efficient parallelization of this recursive algorithm (yielding $O(\log^3 n)$ parallel time) is straightforward, except for the stage of the factorization of the matrices X_h (which, by Lemma 7.2 below, are block diagonal with dense diagonal blocks of sizes of the order of $s(n_h) \times s(n_h)$). However, for the purpose of solving the systems $A\mathbf{x} = \mathbf{b}$, we do not have to factorize the matrices X_h. It suffices to invert them, and this can be efficiently done by using the techniques of [Be], provided that A is a well-conditioned matrix. Thus we arrive at the recursive $s(n)$-factorization (4.1)–(4.4), where we recursively factorize only the matrices A_{h+1} in (4.3) and A_{h+1}^{-1} in (4.4) but not the matrices X_h and X_h^{-1}. This modification of the factorization scheme is crucial in some important combinatorial computations (see [PR4], [PR6]).

5 Parallel generalized nested dissection: The main theorem

Hereafter, we will assume that c and σ are constants such that

$$s(n) = cn^{\sigma}, \qquad \tfrac{1}{2} \le \sigma < 1. \tag{1}$$

Equation (5.1) holds in all the interesting applications (such as planar graphs, grid graphs, and finite element graphs) for which $s(n)$-separator families are defined.

For simplicity, we will also assume hereafter that

$$M(n) = n^{\omega^*} \quad \text{for some constant } \omega^* > 2 \geq \tfrac{1}{\sigma} \tag{2}$$

and consequently that

$$M(ab) = M(a)M(b). \tag{3}$$

Theorem 5.1 *Let $G = (V, E)$ be an $s(n)$-separatable graph for $s(n)$ satisfying* (5.1). *Then, given an $n \times n$ symmetric positive definite matrix A such that* cond $A = n^{O(1)}$ *and $G = G(A)$, we can numerically compute a recursive $s(n)$-factorization of A in time $O(\log^3 n)$ with $M(s(n))/\log n$ processors (provided that $M(s(n))$ processors suffice to multiply a pair of $s(n) \times s(n)$ matrices in time $O(\log n)$ and that $M(n)$ satisfies* (5.2)). *Whenever such a recursive $s(n)$-factorization of A is available, $O(\log^2 n)$ time and $(|E|/\log n) + s(n)^2$ processors suffice to solve a system of linear equations $A\mathbf{x} = \mathbf{b}$ for any given vector $\mathbf{b}$ of dimension n.*

Remark 5.1. It is possible to extend Theorem 5.1 to the case for which (5.1) does not hold by using [LRT, Thms. 7–9]. On the other hand, the restriction to the class of symmetric positive definite input matrices A in the statements of Theorem 5.1 is needed only to support numerical stability of the factorization (4.3), (4.4).

Remark 5.2. The product of our parallel time and processor bounds is the same, TP = (PARALLEL TIME) $*$ PROCESSOR $= O(M(s(n))\log^2 n)$, both for computing the whole recursive factorization (4.3), (4.4) and for its proper stage of inverting X_{d-1}.

Remark 5.3. As was noted by Gazit and Miller [GM2], the recursive $s(n)$-factorization can be computed by using $O(\log^2 n \log\log n)$ time and $M(s(n))$ processors. Moreover, their approach can be extended to reach the bounds of $O(\log^2 n \log(1/\epsilon))$ time and simultaneously $O(M(s(n))^{1+\epsilon})$ processors for a positive parameter $\in$ [AR]. One may try to improve the processor efficiency of the latter estimates by applying a relatively minor super effective slowdown of the computations [PP].

6 Outline of the proof of the main theorem

We will show that the parallel algorithm uses only $d = O(\log n)$ stages. Let $\delta = \delta(n)$ denote $cn^{\sigma-1} = s(n)/n$, and let n_0 be large enough, so that

$$\alpha k + s(k) = (\alpha + \delta)k = \beta k, \qquad \beta = \alpha + \delta < 1 \quad \text{if } k > n_0. \tag{1}$$

Equations (4.2) and (6.1) together imply that

$$n_h \le (\alpha+\delta)^{d-h} n = \beta^{d-h} n, \qquad h = 0, 1, \ldots, d. \tag{2}$$

Lemma 6.1 *$d = O(\log n)$ for fixed n_0 and $\alpha < 1$.*

Proof: Relation (6.2) for $h = 0$ implies that $d \le \log(n/n_0)/\log(1/(\alpha+\delta)) = O(\log n)$. □

The next lemma shows that for all h the auxiliary matrices A_h and X_h are positive definite (and therefore nonsingular) and have condition numbers not exceeding the condition number of A. This implies that our parallel algorithm is numerically stable. The lemma follows from the observation that A_{h+1}^{-1} is a principal submatrix of A_h^{-1} and from the interlacing property of the eigenvalues of a symmetric matrix (see [GoL] or [Par]).

Lemma 6.2 *The matrices A_{h+1} and X_h are symmetric positive definite if the matrix A_h is symmetric positive definite, and, furthermore,* $\max\{(\text{cond } A_h)_2, (\text{cond } X_h)_2\} \le (\text{cond } A)_2$ *for all h.*

In the remainder of this paper we further specify our parallel nested dissection algorithm and estimate its complexity. We observe that all its arithmetic operations (except those needed in order to invert X_h for all h) are also involved in the sequential algorithm of [LRT]. (As in the latter paper, we ignore the arithmetic operations for which at least one operand is zero; we assume that no random cancellations of nonzero entries takes place, for if there are such cancellations, we would only arrive at more optimistic bounds; we will treat both X_h and X_h^{-1} as block diagonal matrices having nonzero blocks in the same places.)

For each h we group all the arithmetic operations involved to reduce these operations to a pair of matrix multiplications $U_h = Y_h X_h^{-1}$ (which also gives $X_h^{-1} Y_h^T = U_h^T$) and $W_h = U_h Y_h^T$ and to a (low-cost) matrix subtraction $A_{h+1} = Z_h - W_h$ (it is assumed here that the matrix X_h^{-1} has been precomputed). Below, Theorem 7.1 provides a bound on the complexity of numerically computing the inverse of the auxiliary matrices $X_0, \ldots, X_d$, and Theorem 8.1 provides a bound on the cost of parallel multiplication of the auxiliary matrices and implies the time bound of $O(\log^2 n)$ for the entire computation, excluding the stage of the inversion of the matrices X_h. The number of processors is bounded above by the number of arithmetic operations used in the algorithm of [LRT], that is, by $O(s(n)^3)$ (see [LRT, Thm. 3] and Remark 5.2).

The estimates of Theorem 5.1 for the cost of computing the recursive factorization (4.1)–(4.4) immediately follow from Theorems 7.1 and 8.1 below.

Next we discuss the parallel complexity of back solving the linear system $A\mathbf{x} = \mathbf{b}$, given the recursive factorization. As we have already pointed out, when the recursive factorization (4.1)–(4.4) has been computed, we evaluate

$\mathbf{x} = A^{-1}\mathbf{b}$ by means of successive premultiplications of some subvectors of $\mathbf{b}$ by the matrices $Y_h X_h^{-1}$, X_h^{-1}, and $X_h^{-1}Y_h^T$ for h ranging between 0 and d. The parallel time bounds are $O(\log n)$ for each h and $O(\log^2 n)$ for all h. The obvious processor bound is $(|E|+|F|)/\log n$, where $|E|+|F|$ denotes the number of entries of an $n \times n$ array corresponding to the nonzeros of at least one of the submatrices $X_h^{-1}Y_h^T$, $Y_h X_h^{-1}$, and X_h^{-1} of (4.4) for $h = 0, 1, \ldots, d-1$ (for each h, $X_h^{-1}Y_h^T$, $Y_h X_h^{-1}$, and X_h^{-1} occupy the upper-right, lower-left, and upper-left corners of the array, respectively). The nonzeros of A_0 form the set E of the edges of $G(A)$; other nonzeros form the set F called *fill-in* (associated with the nonzeros introduced in the process of computing the $s(n)$-factorization (4.1)–(4.4)).

Finally, we must discuss the space bounds of the algorithm. By [LRT, Thm. 2], $|F| = O(n + s(n)^2 \log n)$, and this bound can be applied to our algorithm as well. The proofs in [LRT] are under the assumption that $s(n) = O(\sqrt{n})$, but the extension to any $s(n)$ satisfying (5.1) is immediate. Likewise, Lipton, Rose, and Tarjan estimated only the number of multiplications involved, but including the additions and subtractions would increase the upper estimates yielded in their and our algorithms by only a constant factor.

Remark 6.1. The algorithms and the complexity estimates of this paper will be immediately extended to the case of nonsymmetric linear systems with directed graphs if for all h we replace the matrices Y_h^T by matrices W_h (which are not generally the transposes of Y_h) and remove the assumption that the matrices X_h are symmetric. Of all the results and proofs, only Lemma 6.2 and the numerical stability of our $s(n)$-recursive factorization (4.1)–(4.4) are not extended. This lack of extension of Lemma 6.2 surely devalues the resulting numerical algorithm, but the algorithm remains powerful for the computations over the semirings (dioids), with interesting combinatorial applications (see [PR4], [PR6], [PR7]).

7 Cost of parallel inversion of the auxiliary matrices $\mathbf{X_h}$

In this section we specify Stage 0 of computing the recursive factorization (4.1)–(4.4) (see §4) and prove the following result:

Theorem 7.1 *Let A be an $n \times n$ well-conditioned symmetric positive definite matrix having a recursive $s(n)$-factorization. Then $O(\log^3 n)$ parallel time and $M(s(n))/\log n$ processors suffice to numerically invert the auxiliary matrices $X_0, \ldots, X_d$ that appear in the recursive $s(n)$-factorization* (4.1)–(4.4).

Proof. We first reexamine the well-known correlations between the elimination of the variables and of the associated vertices of $G = G(A)$, which we

will derive from the previous analysis of nested dissection in [R] and [GeL]. We observe that the elimination of a vertex (variable) v is associated with the replacement of the edges in the graph G as follows: (1) First, for *every* pair of edges $\{u_1, v\}$ and $\{v, u_2\}$, the fill-in edge $\{u_1, u_2\}$ is to be added to the set of edges (unless $\{u_1, u_2\}$ is already in the graph); (2) then every edge with an end point v is deleted.

Adding an edge such as $\{u_1, u_2\}$ to the edge set corresponds to four arithmetic operations of the form $z - y_1 x^{-1} y_2$, where x, y_1, y_2, z represent the edges $\{v, v\}$, $\{u_1, v\}$, $\{v, u_2\}$, $\{u_1, u_2\}$, respectively (see Figs. 1–5 and the end of Remark 1.1). If a block of variables is eliminated, then a set S, representing this block, should replace a vertex in the above description, so that, at first, for every pair of edges $\{u_1, s_1\}$, $\{u_2, s_2\}$ with the end points s_1 and s_2 in S, the edge $\{u_1, u_2\}$ is added to the set of edges, and then, when all such pairs of edges have been scanned, all the edges with one or two end points in S are deleted. This corresponds to the matrix operations of the form $Z - Y_1 X^{-1} Y_2^T$, where X, Y_1, Y_2^T, Z represent the blocks of edges of the form $\{s_1, s_2\}$, $\{u_1, s_1\}$, $\{s_2, u_2\}$, $\{u_1, u_2\}$, respectively, where $s_1, s_2 \in S$ and where u_1, u_2 denote two vertices connected by edges with S. For symmetric matrices we may assume that $Y_1 = Y_2 = Y$. Of course, the objective is to arrange the elimination so as to decrease the fill-in and the (sequential and parallel) arithmetic cost. This objective is achieved in the nested dissection algorithm, in which the elimination is ordered so that every eliminated block of vertices is connected by edges with only relatively few vertices (confined to an $s(n_h)$-separator, separating the vertices of the eliminated block from all other vertices).

To ensure the latter property we exploit the existence of an $s(n)$-separator family for the graph $G = G(A)$ and order the elimination by using a separator tree T_G defined for a graph G as follows:

DEFINITION 7.1. See Fig. 6. Suppose that the graph $G = (V, E)$ has n vertices. If $n \le n_0$ (see Definition 3.1), let T_G be the trivial tree with no edges and with the single leaf (V, S), where $S = V$. If $n > n_0$, we know an $s(n)$-separator S of G, so that we can find a partition $V(1)$, $V(2)$, S of V such that there exists no edge in E between the sets $V(1)$ and $V(2)$ and, furthermore, $|V(1)| \le \alpha n$, $|V(2)| \le \alpha n$, and $|S| \le s(n)$. Then T_G is defined to be the binary tree with the root (V, S) having exactly two children that are the roots of the subtrees T_{G_1}, T_{G_2} *of* T_G, where G_j is the subgraph of G induced by the vertex set $S \cup V(j)$ for $j = 1, 2$. (Note that T_G is *not* equivalent to the elimination trees of [Schr], [Li2], and [GHLN] or to the separator trees of [GT], since the latter trees do not include the separator in the induced subgraphs.)

The following definitions are equivalent to the usual ones, as, for example, given in [LRT].

DEFINITION 7.2. Let the *height* of a node v in T_G equal d minus the length of the path from the root to v, where d, the height of the root, is the maximum length of a path from the root to a leaf. Let N_h be the num-

ber of nodes of height h in T_G. Since T_G is a binary tree, $N_h \leq 2^{d-h}$. Let $(V_{h,1}, S_{h,1}), \ldots, (V_{h,N_h}, S_{h,N_h})$ be a list of all the nodes of height h in T_G, and let $S_h = \bigcup_{k=1}^{N_h} S_{h,k}$.

Let n be the number of vertices of G. Since G has an $s(n)$-separator family, $|V_{h,k}| \leq n_h$ and $|S_{h,k}| \leq s(n_h)$ for each $h \geq 1$ and for $k = 1, \ldots, N_h$ (see (4.2) for the definition of n_h); furthermore, $|V_{0,k}| \leq n_0$ and $S_{0,k} = V_{0,k}$ for $k = 1, \ldots, N_0$ by the definition of the tree T_G.

DEFINITION 7.3. For each $k = 1, \ldots, N_h$ let $R_{h,k}$ denote the set of all the elements of $S_{h,k}$ that are not in S_{h^*} for $h^* > h$, so that $R_{h,k} = S_{h,k} - \cup S_{h^*,k^*}$, where the union is over all the ancestors $(V_{h^*,k^*}, S_{h^*,k^*})$ of $(V_{h,k}, S_{h,k})$ in T_G. Let $R_h = \bigcup_{k=1}^{N_h} R_{h,k}$.

Observe that, by the definition of the sets $R_{h,k}$ and R_h and of an $s(n)$-separator family, $R_{h,k_1} \cap R_{h,k_2} = \emptyset$ if $k_1 \neq k_2$, $R_h \cap R_{h^*} = \emptyset$ if $h \neq h^*$, and $V = \bigcup_{h=0}^{d} R_h$. Also observe that for distinct k the subsets $R_{h,k}$ of R_h are not connected by edges with each other; moreover, the vertices of each set $R_{h,k}$ can be connected by edges only with the vertices of the set $R_{h,k}$ itself and of the separator sets $S_{h+g,q}$ in the ancestor nodes of $(V_{h,k}, S_{h,k})$ of the tree T_G. Now we are ready to describe the order of elimination of vertices and of the associated variables. We will eliminate the vertices in the following order: first the vertices of R_0, then the vertices of R_1, then the vertices of R_2, and so on. (For each h we will eliminate the vertices of R_h in parallel for all the disjoint subsets $R_{h,1}, R_{h,2}, \ldots, R_{h,N_h}$.) This way all the vertices of $V = \bigcup_h R_h$ will be processed. In particular, the rows and columns of A_h associated with the vertices of $R_{h,k}$ form an $|R_{h,k}| \times |R_{h,k}|$ diagonal block of X_h for $k = 1, 2, \ldots, N_h$; X_h is the block diagonal submatrix of A_h with these N_h diagonal blocks, and $n_{h+1} = n_h - |R_h|$.

Let us now formally define the desired permutation matrix P and set of integers $n_d, n_{d-1}, \ldots, n_0$, which we need in Stage 0 (see §4). Let $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ be any enumeration of the n vertices of G such that $\pi(v) < \pi(v^*)$ if $v \in R_h$, $v^* \in R_{h^*}$, $h^* > h$, and, furthermore, π consecutively orders the vertices of $R_{h,k}$ for each h and k. Thus the elements of $\pi(R_h)$ are in the range $\delta_h + 1, \ldots, \delta_{h+1}$, where $\delta_h = \sum_{g<h} |R_g|$. Such an enumeration can be easily computed directly from the separator tree in parallel time $O(\log^2 n)$ with $n/\log n$ processors by first numbering the vertices of $R_d = S_d$ as $n, n-1, \ldots$ and then numbering (also in the decreasing order) all the previously unnumbered vertices of R_h of height h for each h, where $h = d-1, d-2, \ldots, 0$.

We define the permutation matrix $P = [p_{ij}]$ such that $p_{ij} = 1$ if $j = \pi(i)$ and $p_{ij} = 0$ otherwise. This gives us the initial matrix $A_0 = PAP^T$. Recursively, for $h = 0, 1, \ldots, d-1$, let $n_h = n - \delta_h$, and this completes Stage 0.

Now we define

$$A_h = \begin{bmatrix} X_h & Y_h^T \\ Y_h & Z_h \end{bmatrix},$$

the $(n-\delta_h)\times(n-\delta_h)$ symmetric matrix, where X_h is the $|R_h|\times|R_h|$ upper-left submatrix of A_h, Y_h is the $(n-\delta_h-|R_h|\times|R_h|)$ lower-left submatrix of A_h, and Z_h is the $(n-\delta_h-|R_h|)\times(n-\delta_h-|R_h|)$ lower-right submatrix of A_h. We then define $A_{h+1}=Z_h-Y_hX_h^{-1}Y_h^T$. Thus in Stage $h+1$ of computing the recursive factorization (see §4) we have eliminated the variables (vertices) associated with R_h.

We now claim that for a fixed h we can compute A_{h+1} from X_h, Z_h, and Y_h in time $O(\log^2 s(n))$ with at most $M(s(n))$ processors. To prove this we will investigate the sparsity of A_h and the connectivity of G_h.

Let $A_h=(a_{ij}^{(h)})$. We define the associated graph $G_h=(V_h,E_h)$ with the vertex set $V_h=\{\delta_h+1,\delta_h+2,\dots,n\}$ and the edge set $E_h=\{\{i+\delta_h,j+\delta_h\}|a_{ij}^{(h)}\neq 0\}$; that is, G_h is derived from $G(A_h)$ by adding δ_h to each vertex number (see Figs. 1–5). Note that $i,j\in V_h$ if the edge $\{i,j\}$ belongs to E_h. (The fill-in in stage h is the set of edges that are in E_h but not in E_{h-1}.)

Now we are ready to establish a lemma that provides some useful information about the fill-in, about the connectivity of G_h, and, consequently, about the sparsity of X_h. By usual arguments (see [GeL], [Li2], [GT]) we arrive at the following lemma:

Lemma 7.1 *Let $h\geq 0$. Then the following hold*:

(a) *If p is a path in G_h between two vertices $i\notin V_{h^*,k}$ and $j\in R_{h^*,k}$ for some $h^*\geq h$ and some k, then p visits some vertex v such that $\pi(v)>\delta_{h^*+1}$, that is, $v\notin R_q$ for $q\leq h^*$.*

(b)

$$E_{h+1}=E_h^-\cup F_h,$$

$$E_h^-=\{\{i,j\}\in E_h|i,j\notin R_h\},$$

$$F_h=\cup\{\{i,j\}|\exists k\exists\{i,j_1\},\{j_1,j_2\},\dots\{j_{l-1},j_l\},\{j_l,j\}\in E_h\}$$

provided that $j_1,\dots,j_l\in R_{h,k}$ and that $\pi(i)>\delta_{h+1}$, $\pi(j)>\delta_{h+1}$ in the definition of F_h.

Lemma 7.1 defines the desired restriction on the edge connections in $G(A_h)$. In particular, part (a) of Lemma 7.1 implies that E_h contains no edge between $R_{h,k}$ and R_{h,k^*} for $k\neq k^*$.

Since π groups the vertices of $R_{h,k}$ together and since $\max_k|R_{h,k}|\leq\max_k s(|V_{h,k}|)\leq s(n_h)$, we immediately arrive at the following lemma:

Lemma 7.2 *X_h is a block diagonal matrix consisting of $N_h\leq 2^{d-h}$ square blocks of sizes $|R_{h,k}|\times|R_{h,k}|$, so that each block is of size at most $s(n_h)\times s(n_h)$.*

Lemma 7.2 implies that for $h>0$ the numerical inversion of X_h can be reduced to $N_h\leq 2^{d-h}$ parallel numerical inversions of generally dense matrices, each of size at most $s(n_h)\times s(n_h)$ (that is, one dense matrix is associated with

each $R_{h,k}$, so that its size is at most $|R_{h,k}| \times |R_{h,k}|$). By Fact 2.1 such inversions can be performed in $O(\log^2 n)$ time with $N_h M(s(n_h)) \le 2^{d-h} M(s(n_h))$ processors.

The next lemma is from [LT]. Its proof is simplified in [PR1] for the case in which $\alpha^* > \frac{1}{2}$, and we need only this case. Both proofs also show simple transformations of the respective $s(n)$-separator tree into $s^*(n)$-separator trees.

Lemma 7.3 *For any triple of constants α, α^*, and n_0 such that $\frac{1}{2} \le \alpha^* < \alpha < 1$ and $n_0 > 0$, if a graph has an $s(n)$-separator family with respect to α and n_0 (see Definition 3.1), then this graph has an $s^*(n)$-separator family with respect to α^* and n_0, where $s^*(n) \le \sum_{h=0}^{d} s(n_h)$; in particular, $s^*(n) \le cn^\sigma/(1-\beta^\sigma)$ if $s(n) \le cn^\sigma$ for some positive constants c, β, and σ, where $\beta < 1$ (see (5.1), (6.1), (6.2)).*

Lemma 7.4 *$2^{d-h} M(s(n_h)) \le M(s(n)) \eta^{d-h}$ for some $\eta < 1$.*

Proof. Equation (5.1) and relation (6.2) imply that $s(n_h) \le c(\alpha+\delta)^{\sigma(d-h)} n^\sigma$, so that $M(s(n_h)) \le c^{\omega^*}(\alpha+\delta)^{\sigma\omega^*(d-h)} n^{\sigma\omega^*}$ (see (5.2)). We may choose n_0 sufficiently large so as to make δ sufficiently small and then apply Lemma 7.3 to make sure that $\alpha + \delta$ lies as close to $\frac{1}{2}$ as we like. Since $\sigma\omega^* > 1$ (see (5.2)), we may assume that $(\alpha+\delta)^{\sigma\omega^*} < \frac{1}{2}$, so that $\eta = 2(\alpha+\delta)^{\sigma\omega^*} < 1$. Then

$$2^{d-h} M(s(n_h)) \le \eta^{d-h} c^{\omega^*} n^{\sigma\omega^*} = \eta^{d-h} M(s(n)). \qquad \square$$

¿From Fact 2.1 and Brent's slowdown principle of parallel computation, we may invert X_h by using $O(k \log^2 n)$ steps and $\lceil 2^{d-h} M(s(n_h))/k \rceil \le \lceil M(s(n)) \eta^{d-h}/k \rceil$ processors for some $\eta < 1$ and for any k such that $1 \le k = k(h)$. Choosing the minimum $k = k(h) \ge 1$ such that $M(s(n))\eta^{d-h}/k(h) \le M(s(n))/\log n$ (so that $k(h) = \eta^{d-h} \log n$ if $h > d + \log\log n/\log\eta$ and $k(h) = 1$ otherwise), we simultaneously obtain the time bound $O(\log^3 n)$ (see Lemma 6.1) and the processor bound $M(s(n))/\log n$, required in Theorem 7.1. $\square$

8 Estimating the cost of parallel multiplication of auxiliary matrices

Theorem 8.1 *All the $2d$ matrix multiplications*

$$U_h = Y_h X_h^{-1}, \qquad W_h = U_h Y_h^T, \qquad h = 0, 1, \ldots, d-1 \tag{1}$$

involved in the recursive $s(n)$-factorization (4.1)–(4.4) can be performed by using $O(\log^2 n)$ parallel time and $M(s(n))$ processors or (if we slow down the computations by a factor of $\log n$) by using $O(\log^3 n)$ parallel time and $M(s(n))/\log n$ processors.

Proof. We will prove Theorem 8.1 by estimating the cost of parallel evaluation of the matrix products of (8.1) (given Y_h and X_h^{-1}) for $h = 0, 1, \ldots, d-1$. First we will arrange the matrix multiplications of (8.1) by reducing them to several matrix multiplications of the form

$$U_{h,k} = Y_{h,k} X_{h,k}^{-1}, \qquad W_{h,k} = U_{h,k} Y_{h,k}^T, \qquad k = 1, 2, \ldots, N_h, \quad h = 0, 1, \ldots, d-1. \tag{2}$$

To arrive at such a rearrangement, partition Y_h into N_h submatrices $Y_{h,k}$ having columns associated with the row sets $R_{h,k}$ and having the sizes $m_{h,k} \times |R_{h,k}|$, where $m_{h,k} \le n - \delta_h$ for $k = 1, \ldots, N_h$. The dense diagonal blocks of X_h^{-1} are denoted $X_{h,k}^{-1}$, respectively. By the definition of G_h and T_G and by virtue of Lemma 7.1, the matrix $Y_{h,k}$ may have nonzero entries only in rows i such that i lies in one of the sets $R_{h+g,q}$ (for $1 \le g \le d-h$, $q = q(g,h,k)$) corresponding to an ancestor $(V_{h+g,q}, S_{h+g,q})$ of the node $(V_{h,k}, S_{h,k})$ in T_G.

To deduce the desired complexity estimates, examine the cost of all the latter matrix multiplications (8.2), grouping them not in the above *horizontal* order (where k ranges from 1 to N_h for a fixed h) but in the *vertical* order of Definition 3.1, that is, going from the root of the tree T_G to its leaves.

By slightly abusing the notation, denote $n = |R_{h,k}|$, $m = m_{h,k}$ for a fixed pair h and k, and consider the matrix multiplications of (8.2) associated with the node $(V_{h,k}, S_{h,k})$ and with its descendents in the tree T_G. These matrix multiplications can be performed in $O(\log^2 n)$ time (this is required in Theorem 8.1); let $P(n,m)$ denote the associated processor bound. For the two children of the node $(V_{h,k}, S_{h,k})$ the two associated numbers of processors will be denoted by $P(n_1, m_1)$ and $P(n_2, m_2)$, where, by virtue of Lemma 7.2 and Definition 3.1 (see also [LRT]),

$$\begin{gathered} m_1 + m_2 \le m + 2s(n), \\ n \le n_1 + n_2 \le n + s(n), \\ (1-\alpha)n \le n_i \le \alpha n + s(n) \quad \text{for } i = 1, 2. \end{gathered} \tag{3}$$

Let $M(p,q,r)$ hereafter denote the number of processors required in order to multiply $p \times q$ by $q \times r$ matrices in $O(\log(pqr))$ parallel steps, so that $M(p,q,r) \le M(q)\lceil p/q \rceil \lceil r/q \rceil$ (all the processor bounds have been defined up to within constant factors). For fixed h and k (and, therefore, for a fixed separator $S_{h,k}$) the matrix multiplications (8.2) can be performed by using $O(\log n)$ parallel steps and $M(s(n)+m, s(n), s(n)+m) \le \lceil (1+m/s(n)) \rceil^2 M(s(n))$ processors. Therefore, recursively,

$$P(n,m) \le \left(1 + \left(1 + \frac{m}{s(n)}\right)^2\right) M(s(n)) + P(n_1, m_1) + P(n_2, m_2) \tag{4}$$

for some n_1, n_2, m_1, m_2 satisfying (8.3).

Using (8.4), we will prove the following claim, which in its special case for $m = 0$ amounts to Theorem 8.1 (recall that we already have the parallel time bound $O(\log^2 n)$ of this theorem):

CLAIM. *$P(n,m) \le (c_0 + c_1(m/s(n)) + c_2(m/s(n))^2)M(s(n))$ for all m and n and for some constants c_0, c_1, c_2.*

Proof: If $n \le n_0$, then $P(n,m) \le M(n) \le c_0$ provided that $c_0 \ge M(n_0)$. Thus let $n \ge n_0$ and prove the claim by induction on n. We may assume that n_0 is large enough, so that (8.3) implies that $n_i < n$ for $i = 1, 2$. Then by the induction hypothesis the claim holds if n is replaced by n_i for $i = 1, 2$, so that

$$P(n_1,m_1) + P(n_2,m_2) \le \sum_{i=1}^{2} \left(c_0 + c_1 \frac{m_i}{s(n_i)} + c_2 \left(\frac{m_i}{s(n_i)} \right)^2 \right) M(s(n_i)).$$

Therefore,

$$\sum_i P(n_i,m_i) \le c_0 \sum_i M(s(n_i)) + c_1 \sum_i m_i \frac{M(s(n_i))}{s(n_i)} + c_2 \sum_i m_i^2 \frac{M(s(n_i))}{s(n_i)^2}. \quad (5)$$

Next we deduce from (8.3) that for $g = 1$ and $g = 2$

$$\begin{aligned} \textstyle\sum_i m_i^g \frac{M(s(n_i))}{s(n_i)^g} &\le (\textstyle\sum_i m_i^g) \max_i \left(\frac{M(s(n_i))}{s(n_i)^g} \right) \\ &\le (m + 2s(n))^g \tfrac{M(s(\alpha n + s(n)))}{s(\alpha n + s(n))^g} \\ &\le (m + 2s(n))^g \tfrac{M(s(\beta n))}{s(\beta n)^g} \quad \text{for } \beta < 1 \end{aligned}$$

(apply (6.1) to deduce the last inequality). Applying here (5.1) and (5.2), we obtain that

$$\sum_i m_i^g \frac{M(s(n_i))}{s(n_i)^g} \le \gamma (m + 2s(n))^g \frac{M(s(n))}{s(n)^g}, \quad (6)$$

where $\gamma = \beta^{(\omega^* - g)\sigma}$ is a constant, $\gamma < 1$, $g = 1, 2$, $g < \omega^*$.

Furthermore, (5.1) and (8.3) imply that the sum $M(s(n_1)) + M(s(n_2))$ takes on its maximum value where one of n_1, n_2 is as large as possible (that is, equal to $\alpha n + s(n)$), which makes the other as small as possible (that is, equal to $(1-\alpha)n$). Therefore,

$$\begin{aligned} M(s(n_1)) + M(s(n_2)) &\le M(s(\alpha n + s(n))) + M(s((1-\alpha)n)) \\ &\le M(s((\alpha+\delta)n)) + M(s((1-\alpha)n)) \\ &= M(cn^\sigma(\alpha+\delta)^\sigma) + M(cn^\delta(1-\alpha)^\sigma) \end{aligned}$$

(see (5.1) and (6.1)). Applying here (5.3) and then (5.1) and (5.2), we deduce that

$$\begin{aligned} M(s(n_1)) + M(s(n_2)) &\le (M((\alpha+\delta)^\sigma) + M((1-\alpha)^\sigma)) M(cn^\sigma) \\ &\le ((\alpha+\delta)^{\omega^*\sigma} + (1-\alpha)^{\omega^*\sigma}) M(s(n)), \end{aligned}$$

where $\omega^*\sigma > 1$. The positive δ can be assumed to be arbitrarily close to 0, and so we deduce that

$$M(s(n_1)) + M(s(n_2)) \leq vM(s(n)) \tag{7}$$

for a constant $v < 1$.

Combining (8.4)–(8.7), we obtain that

$$\begin{aligned} P(n,m) \leq\ & (2 + vc_0 + 2\gamma c_1 + 4\gamma c_2)M(s(n)) \\ & +(2 + \gamma c_1 + 4\gamma c_2)M(s(n))\tfrac{m}{s(n)} \\ & +(1 + \gamma c_2)M(s(n))\left(\tfrac{m}{s(n)}\right)^2 \end{aligned} \tag{8}$$

for two constants $\gamma < 1$, $v < 1$. We choose c_2 large enough, so that $1 + \gamma c_2 \leq c_2$, we then choose c_1 large enough so that $2 + \gamma c_1 + 4\gamma c_2 \leq c_1$, and, finally, we chose c_0 large enough, so that $2 + vc_0 + 2\gamma c_1 + 4\gamma c_2 \leq c_0$. Then (8.8) implies the claim and, consequently, Theorem 8.1. □

Acknowledgments. The authors thank the referees for numerous helpful suggestions. The paper was typed by Sally Goodall (with assistance from Joan Bentley, Bettye Kirkland, and Chris Lane).

References

[AR] D. Armon and J. Reif, *Space and time efficient implementation of a parallel nested dissection*, in Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures, Association for Computing Machinery, New York, 1992, pp. 344–352.

[BCLR] D. Bini, M. Capovani, G. Lotti, and F. Romani, $O(n^{2.7799})$ *complexity for matrix multiplication*, Inform. Process. Lett., 8 (1979), pp. 234–235.

[Be] A. Ben-Israel, *A note on iterative methods for generalized inversion of matrices*, Math. Comput., 20 (1966), pp. 439–440.

[BGH] A. Borodin, J. von zur Gathen, and J. Hopcroft, *Fast parallel matrix and* GCD *computation*, Inform. and Control, 52 (1982), pp. 241–256.

[Bi] D. Bini, *Relations between* EC*-algorithms and* APA*-algorithms: Applications*, Calcolo, 17 (1980), pp. 87–97.

[BM] A. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier, New York, 1975.

[BP] D. Bini and V. Pan, *Numerical and Algebraic Computations with Matrices and Polynomials*, Birkhäuser-Verlag, Boston, 1993.

[Ca] D. A. Calahan, *Parallel solution of sparse simultaneous linear equations*, in Proc. 11th Allerton Conference, 1973, pp. 729–738.

[Ch] A. K. Chandra, *Maximal Parallelism in Matrix Multiplication*, Report RC-6193, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1976.

[Cs] L. Csanky, *Fast parallel matrix inversion algorithms*, SIAM J. Comput., 5 (1976), pp. 618–623.

[CW1] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, in Proc. 19th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1987, pp. 1–6; J. Symbolic Comput., 9 (1990), pp. 251–280.

[CW2] D. Coppersmith and S. Winograd, *On the asymptotic complexity of matrix multiplication*, SIAM J. Comput., 11 (1982), pp. 472–492.

[D] H. N. Djidjev, *On the problem of partitioning planar graphs*, SIAM J. Alg. Discrete Meth., 3 (1982), pp. 229–240.

[EG] D. Eppstein and Z. Galil, *Parallel algorithmic techniques for combinatorial computation*, Annual Rev. Comput. Sci., 3 (1988), pp. 233–283.

[Ga] D. A. Gannon, *A note on pipelining mesh-connected multiprocessor for finite element problems by nested dissection*, in Proc. International Conference on Parallel Processing, 1980, pp. 197–204.

[GeL] J. A. George and J. W. H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Ge] J. A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–367.

[GHLN] A. George, M. T. Heath, J. W. H. Liu, and E. G. Y. Ng, *Sparse Cholesky factorization on a local-memory multiprocessor*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 327–340.

[GM1] H. Gazit and G. L. Miller, *A parallel algorithm for finding a separator in planar graphs*, in Proc. 28th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1987, pp. 238–248.

[GM2] H. Gazit and G. L. Miller, private communication, 1992.

[GoL] G. H. Golub and C. F. van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, 1989.

[GP1] Z. Galil and V. Pan, *Improving processor bounds for algebraic and combinatorial problems in* RNC, in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1985, pp. 490–495.

[GP2] Z. Galil and V. Pan, *Improved processor bounds for combinatorial problems in* RNC, Combinatorica, 8 (1988), pp. 189–200.

[GP3] Z. Galil and V. Pan, *Parallel evaluation of the determinant and of the inverse of a matrix*, Inform. Process. Lett., 30 (1989), pp. 41–45.

[GT] J. R. Gilbert and R. E. Tarjan, *The analysis of a nested dissection algorithm*, Numer. Math., 50 (1987), pp. 377–404.

[KP] E. Kaltofen and V. Pan, *Processor efficient solution of linear systems over an abstract field*, in Proc. 3rd Annual ACM Symposium on Parallel Algorithms and Architecture, Association for Computing Machinery, New York, 1991, pp. 180–191,

[KR] R. Karp and V. Ramachandran, *A Survey of Parallel Algorithms for Shared Memory Machines*, in Handbook of Theoretical Computer Science, North-Holland, Amsterdam, 1990, pp. 869–941.

[KS] E. Kaltofen and M. Singer, *Size Efficient Parallel Algebraic Circuits for Partial Derivatives*, Tech. Report 90-32, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, 1990.

[Li1] J. W. H. Liu, *The Solution of Mesh Equations on a Parallel Computer*, Tech. Report CS-78-19, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1978.

[Li2] J. W. H. Liu, *A compact row storage scheme for Cholesky factors using elimination trees*, ACM Trans. Math. Software, 12 (1986), pp. 127–148.

[LMNOR] C. E. Leiserson, J. P. Mesirov, L. Nekludova, S. Omahundro, and J. Reif, *Solving sparse systems of linear equations on the connection machine*, in Proc. Annual SIAM Conference, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1986, p. A51.

[Lo] L. Lovász, *Connectivity algorithms using rubber-bands*, in Proc. 6th Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science 241, Springer-Verlag, Berlin, 1986, pp. 394–412.

[LRT] R. J. Lipton, D. Rose, and R. E. Tarjan, *Generalized nested dissection*, SIAM J. Numer. Anal., 16 (1979), pp. 346–358.

[LT] R. J. Lipton and R. E. Tarjan, *A separator theorem for planar graphs*, SIAM J. Appl. Math., 36 (1979), pp. 177–189.

[MVV] K. Mulmuley, U. Vazirani, and V. Vazirani, *Matching is as easy as matrix inversion*, Combinatorica, 7 (1987), pp. 105–114.

[OR] T. Opsahl and J. Reif, *Solving sparse systems of linear equations on the massive parallel machine*, in Proc. 1st Symposium on Frontiers of Scientific Computing, National Aeronautics and Space Administration, Goddard Space Flight Center, Greenbelt, MD, 1986, pp. 2241–2248.

[OV] J. M. Ortega and R. G. Voight, *Solution of partial differential equations on vector and parallel computers*, SIAM Rev., 27 (1985), pp. 149–240.

[Pan1] V. Pan, *On schemes for the evaluation of products and inverses of matrices*, Uspekhi Mat. Nauk, 27 (1972), pp. 249–250.

[Pan2] V. Pan, *Strassen's algorithm is not optimal. Trilinear technique of aggregating, uniting, and canceling for constructing fast algorithms for matrix multiplication*, in Proc. 19th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1978, pp. 166–176.

[Pan3] V. PAN, *Fields extension and trilinear aggregating, uniting and canceling for the acceleration of matrix multiplication*, in Proc. 20th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1979, pp. 28–38.

[Pan4] V. PAN, *New fast algorithms for matrix operations*, SIAM J. Comput., 9 (1980), pp. 321–342.

[Pan5] V. PAN, *New combinations of methods for the acceleration of matrix multiplications*, Comput. Mach. Appl., 7 (1981), pp. 73–125.

[Pan6] V. PAN, *How can we speed up matrix multiplication?* SIAM Rev., 26 (1984), pp. 393–415.

[Pan7] V. PAN, *How to Multiply Matrices Faster*, Lecture Notes in Computer Science 179, Springer-Verlag, Berlin, 1984.

[Pan8] V. PAN, *Fast and Efficient Parallel Algorithms for the Exact Inversion of Integer Matrices*, Tech. Report 85-2, Department of Computer Science, State University of New York, Albany, NY, 1985.

[Pan9] V. PAN, *Fast and efficient parallel algorithms for the exact inversion of integer matrices*, in Proc. 5th Conference on the Foundation of Software Technology, Lecture Notes in Computer Science 206, Springer-Verlag, Berlin, 1985, pp. 504–521.

[Pan10] V. PAN, *Complexity of parallel matrix computations*, Theoret. Comput. Sci., 54 (1987), pp. 65–85.

[Pan11] V. PAN, *Parametrization of Newton's Iteration for Computations with Structured Matrices and Applications*, Tech. Report CUCS-032-90, Department of Computer Science, Columbia University, New York, 1990; Comput. Math. Appl., 24 (1992), pp. 61–75.

[Par] B. N. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[PaS] V. PAN AND R. SCHREIBER, *An improved Newton iteration for the generalized inverse of a matrix, with applications*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 1109–1131.

[PP] V. PAN AND F. P. PREPARATA, *Supereffective slowdown of parallel computations*, in Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures, Association for Computing Machinery, New York, 1992, pp. 402–409.

[PR1] V. PAN AND J. REIF, *Efficient parallel solution of linear systems*, in Proc. 17th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1985, pp. 143–152.

[PR2] V. PAN AND J. REIF, *Efficient parallel linear programming*, Oper. Res. Lett., 5 (1986), pp. 127–135.

[PR3] V. PAN AND J. REIF, *Fast and efficient algorithms for linear programming and for the linear least squares problem*, Comput. Math. Appl., 12A (1986), pp. 1217–1227.

[PR4] V. Pan and J. Reif, *Parallel nested dissection for path algebra computations*, Oper. Res. Lett., 5 (1986), pp. 177–184.

[PR5] V. Pan and J. Reif, *Fast and efficient parallel solution of dense linear systems*, Comput. Math. Appl., 17 (1989), pp. 1481–1491; preliminary version in Proc. 17th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1985, pp. 143–152.

[PR6] V. Pan and J. Reif, *Fast and efficient solution of path algebra problems*, J. Comput. System Sci., 38 (1989), pp. 494–510.

[PR7] V. Pan and J. Reif, *The parallel computation of minimum cost paths in graphs by stream contraction*, Inform. Process. Lett., 49 (1991), pp. 79–83.

[PrS] E. P. Preparata and D. V. Sarwate, *An improved parallel processor bound in fast matrix inversion*, Inform. Process. Lett., 7 (1978), pp. 148–149.

[R] D. J. Rose, *A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations*, in Graph Theory and Computing, R. Read, ed., Academic Press, New York, 1972, pp. 183–217.

[Scho] A. Schönhage, *Partial and total matrix multiplication*, SIAM J. Comput., 19 (1981), pp. 434–456.

[Schr] R. Schreiber, *A new implementation of sparse Gaussian elimination*, Trans. Math. Software, 8 (1982), pp. 256–276.

[Stra1] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.

[Stra2] V. Strassen, *Relative bilinear complexity and matrix multiplication*, in Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, 1986, pp. 49–54.

[T1] R. E. Tarjan, *Fast algorithms for solving path problems*, J. Assoc. Comput. Mach., 28 (1981), pp. 594–614.

[T2] R. E. Tarjan, *A unified approach to path problems*, J. Assoc. Comput. Mach., 28 (1981), pp. 577–593.

[ZG] E. Zmijewski and J. R. Gilbert, *A parallel algorithm for sparse Cholesky factorization on a multiprocessor*, Parallel Comput., 7 (1988), pp. 199–210.