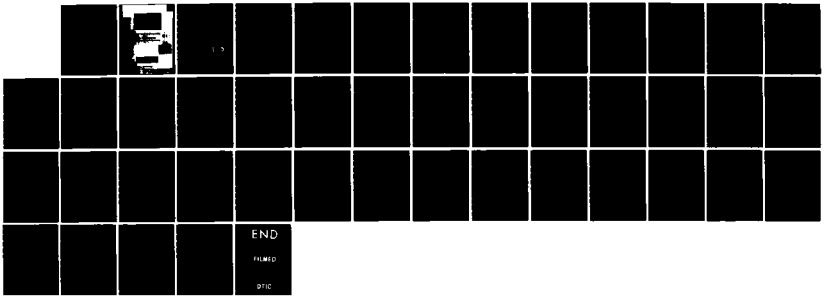
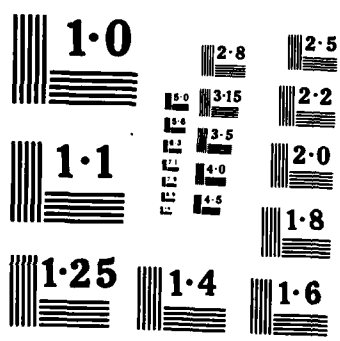
AD-A152 663 EFFICIENT PARALLEL SOLUTION OF LINEAR SYSTEMS(U) 1/1
HARVARD UNIV CAMBRIDGE MA CENTER FOR RESEARCH IN
COMPUTER TECHNOLOGY V PAN ET AL. MAR 85 TR-02-85
UNCLASSIFIED N00014-80-C-0647 F/G 12/1 NL



END
FILMED
DTIC


1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
5·0

EFFICIENT PARALLEL SOLUTION OF

LINEAR SYSTEMS

Victor Pan
John Reif

TR-02-85

Harvard University

Center for Research in Computing Technology

Aiken Computation Laboratory
33 Oxford Street
Cambridge, Massachusetts 02138

EFFICIENT PARALLEL SOLUTION OF LINEAR SYSTEMS

Victor Pan
John Reif

TR-02-85

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A152663	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) EFFICIENT PARALLEL SOLUTION OF LINEAR SYSTEMS		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER TR-02-85
7. AUTHOR(s) Victor Pan John Reif		8. CONTRACT OR GRANT NUMBER(s) N00014-80-C-0647
9. PERFORMING ORGANIZATION NAME AND ADDRESS Harvard University Cambridge, MA 02138		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research 800 North Quincy Street Arlington, VA 22217		12. REPORT DATE March 85 13. NUMBER OF PAGES 41
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same as above		15. SECURITY CLASS. (of this report) 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

unlimited

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

unlimited

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

linear systems, parallel algorithms, matrix inverse, Newton's method, nested dissection, sparse linear systems.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See reverse side.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Abstract

The most efficient known parallel algorithms for inversion of a nonsingular $n \times n$ matrix A or solving a linear system $Ax = b$ over the rationals require $O(\log n)^2$ time and $M(n)\sqrt{n}$ processors (where $M(n)$ is the number of processors required in order to multiply two $n \times n$ rational matrices in time $O(\log n)$.) Furthermore, all known polylog time algorithms for those problems are *unstable:* they require the calculations to be done with perfect precision; otherwise they give no results at all.

This paper describes parallel algorithms that have good numerical stability and remain efficient as n grows large. In particular, we describe a quadratically convergent iterative method that gives the inverse (within the relative precision $2^{-n^{O(1)}}$) of an $n \times n$ rational matrix A with condition $\leq n^{O(1)}$ in $O(\log n)^2$ time using $M(n)$ processors. This is the optimum processor bound and a $\sqrt{n}$ improvement of known processor bounds for polylog time matrix inversion. It is the first known polylog time algorithm that is numerically stable. The algorithm relies on our method of computing an approximate inverse of A that involves $O(\log n)$ parallel steps and n^2 processors.

Also, we give a parallel algorithm for solution of a linear system $Ax = b$ with a sparse $n \times n$ symmetric positive definite matrix A. If the graph G(A) (which has n vertices and has an edge for each nonzero entry of A) is $s(n)$-separable, then our algorithm requires only $O((\log n)(\log s(n))^2)$ time and $|E| + M(s(n))$ processors. The algorithm computes a recursive factorization of A so that the solution of any other linear system $Ax = b'$ with the same matrix A requires only $O(\log n \log s(n))$ time and $|E| + s(n)^2$ processors.

Efficient Parallel Solution of Linear Systems

Victor Pan *

Computer Science Department
State University of New York at Albany
Albany, New York

and

John Reif **

Aiken Computation Lab.
Division of Applied Sciences, Harvard University
Cambridge, MA
and
Laboratory of Computer Science
Mass. Inst. of Technology
Cambridge, MA

Abstract

The most efficient known parallel algorithms for inversion of a nonsingular $n \times n$ matrix A or solving a linear system $Ax = b$ over the rationals require $O(\log n)^2$ time and $M(n)\sqrt{n}$ processors (where M(n) is the number of processors required in order to multiply two $n \times n$ rational matrices in time $O(\log n)$.) Furthermore, all known polylog time algorithms for those problems are *unstable:* they require the calculations to be done with perfect precision; otherwise they give no results at all.

This paper describes parallel algorithms that have good numerical stability and remain efficient as n grows large. In particular, we describe a quadratically convergent iterative method that gives the inverse (within the relative precision $2^{-n^{O(1)}}$) of an $n \times n$ rational matrix A with condition $\leq n^{O(1)}$ in $O(\log n)^2$ time using M(n) processors. This is the optimum processor bound and a $\sqrt{n}$ improvement of known processor bounds for polylog time matrix inversion. It is the first known polylog time algorithm that is numerically stable. The algorithm relies on

* Supported by NSF Grant MCS 8203232.

** This work was supported by Office of Naval Research Contract N00014-80-C-0647.

our method of computing an approximate inverse of A that involves O(log n) parallel steps and n^2 processors.

Also, we give a parallel algorithm for solution of a linear system $Ax = b$ with a sparse $n \times n$ symmetric positive definite matrix A. If the graph G(A) (which has n vertices and has an edge for each nonzero entry of A) is s(n)-separable, then our algorithm requires only $O((\log n)(\log s(n))^2)$ time and $|E| + M(s(n))$ processors. The algorithm computes a recursive factorization of A so that the solution of any other linear system $Ax = b'$ with the same matrix A requires only $O(\log n \log s(n))$ time and $|E| + s(n)^2$ processors.

1. Introduction

Recently it has become feasible to construct computer architectures with a large number of processors. We assume the parallel machine model of [Borodin, von zur Gathen, and Hopcroft, 82], where on each step each processor can do a single rational addition, subtraction, multiplication, or division. In this paper we are concerned about the efficient use of this parallelism for solving some fundamental numerical problems such as

(1) INVERT: given an $n \times n$ rational matrix $A = (a_{ij})$, then output A^{-1} within a prescribed accuracy ϵ if A is well-conditioned, else output ill-conditioned.

(2) LINEAR-SOLVE: given a well-conditioned $n \times n$ matrix A and a column vector b of length n, find $x = A^{-1}b$ within a prescribed accuracy ϵ.

We say that matrix A is *well-conditioned* if cond $A < C$ for a certain parameter C. We say that we have *computed* A^{-1} *within accuracy* ϵ if the norm of the error matrix divided by the norm of A^{-1} does not exceed ϵ.

We are interested in parallel algorithms that have good numerical stability, polylog time bounds and small processor bounds for INVERT and LINEAR-SOLVE. Certainly LINEAR-SOLVE can be immediately reduced to INVERT by

the multiplication of A^{-1} by b. Such a reduction of LINEAR-SOLVE to INVERT is particularly appropriate if several linear systems $Ax = b$ with the same A and different b must be solved.

We will present the algorithms for INVERT and for LINEAR-SOLVE that can be applied for any choice of C and ϵ. On the other hand, the complexity estimates of our algorithms depend on the choice of C and ϵ. In this paper we will be primarily concerned about the complexity estimates in the case where $C = n^c$, $\epsilon = 2^{-n^{c'}}$ for some positive constants c and c', say c=100, c' =10. This covers all instances of practical interest. For the sake of completeness, we will also supply the estimates in the case of arbitrary C and ϵ. To simplify the estimates, we will assume that the arithmetic operations are performed with infinite precision. In addition, we will present the error estimates in the case of finite precision computation; this will demonstrate that our iterative algorithms are stable, moreover, two of them are self-correcting.

1.1. Previous Work On Parallel Matrix Inversion

Parallel algorithms with simultaneous polylog time and polynomial processor bounds are known to exist for these problems, but their practical utility is limited due to their large processor bounds. Furthermore these known algorithms have numerical stability problems; if the calculations are not taken in exact arithmetic, their outputs may substantially differ from A^{-1}, so as not to constitute an approximate inverse.

[Csanky, 76] gave the first proof that INVERT, over fields of characteristic 0, can be done in polylog time. The result was at that time surprising, and the proof is quite elegant. Csanky used the Cayley-Hamilton theorem to reduce the problem of matrix inversion to essentially the problem of computing O(n) products of $n \times n$ matrices. Let $M(n) \geq n^2$ be the number of processors sufficient in order to multiply two $n \times n$ matrices in O(log n) time. By the upper bounds of

[Chandra, 76], $M(n) \leq n^{2.81}$. We easily extend that result as follows. If $n \times n$ matrices can be multiplied involving $O(n^{\omega})$ arithmetic operations for some $\omega > 2$, then $M(n) = n^{\omega}$, see Appendix A. The current best upper bound on ω is 2.495...; however, for matrices of smaller sizes we should count only on $M(n)=n^3$ due to the considerable overhead of the known asymptotically fast algorithms for matrix multiplication, see [Coppersmith and Winograd, 82] and compare [Pan, 84]. Csanky's algorithm takes simultaneous time $O(\log n)^2$ and $nM(n)$ processors. [Preparata and Sarwate, 78] reduced the processor bounds to $\sqrt{n}\, M(n)$. If nonexact arithmetic is used, as this occurs in practice, both of these methods suffer from numerical instability due to the use of the Cayley-Hamilton theorem (see discussion in [Wilkinson, 61,65]).

Two methods for parallel matrix inversion over arbitrary fields of constants are known. [Borodin, von zur Gathen, and Hopcroft, 82] observe that the results of [Valiant, Skyum, Berkowitz, and Rackoff, 83] and [Strassen, 73] combined can be used to parallelize the sequential Gaussian elimination algorithm for matrix inversion. This yields a simultaneous $O(\log n)^2$ time and (very large) polynomial processor algorithm for INVERT over any field. (As a matter of fact, the straightforward parallelization of the Gaussian elimination only yields O(n) time, n^2 processor algorithms for both INVERT and LINEAR-SOLVE.) [Berkowitz, 84] has another such matrix inversion algorithm over arbitrary fields. Neither of these two polylog time algorithms can be viewed as practical in the case of real, complex, or rational inputs.

1.2. Our Results for INVERT.

We have two main results:

(1) a parallel iterative method for INVERT in the case of well-conditioned matrices, and

(2) a parallel method for LINEAR-SOLVE in the case of symmetric positive definite sparse matrices.

Both algorithms run in polylog time and give significant decrease of the known processor bounds.

The iterative methods that we describe in Section 2 and Appendix C are known ones, namely, the Newton iteration and its extensions. The first use of the Newton iteration to invert a matrix is attributed by Householder to [Schultz, 33] but frequently goes by the name of [Hotelling, 43a,b] and [Bodewig, 59]. (See [Householder, 64], [Isaacson and Keller, 66], and [Newman, 82] for more recent descriptions of such methods.) [Bojańczyk, 84] made still another rediscovery of Newton's method and proposed it for parallel computation.

The Newton method is not normally recommended for matrix inversion in textbooks on numerical analysis, since an initial approximate inverse B of a given matrix A is required such that $||I - BA||$ is substantially less than 1. The problem of efficiently (i.e., without inverting A) finding such an approximate inverse of a well-conditioned matrix A was an open problem in numerical analysis for the last 50 years since [Schultz, 33]. For a strictly diagonally dominant matrix A an approximate inverse B of A is an easily computable diagonal matrix, but no known techniques give such B for a general matrix A.

We observe that it suffices to define an approximate inverse B such that $||I - BA|| \leq 1 - (1/n^{O(1)})$. Then, since the Newton method is quadratically convergent, O(log n) iterations are sufficient for numerical computation of A^{-1} (up to accuracy $2^{-n^{O(1)}}$). Each iteration takes only O(log n) time and M(n) processors. In that situation we managed to make the Newton iteration work by contributing a method for computing an approximate inverse B of any given well-conditioned matrix A such that $||I - BA|| < 1 - (1/n^{O(1)})$, see Lemma 2.4 in Section 2 and its substantiation in Section 3. The evaluation of such B involves only $3n^2-2n+1$

arithmetic operations and 2n-2 comparisons that can be implemented using O(log n) parallel steps and n^2 processors, see Lemma 2.5 in Section 2. Our resulting algorithm computes A^{-1} using only a relatively few (only O(log n)) matrix multiplications, that is, using $O((\log n)^2)$ time and M(n) processors. This makes the algorithm efficient for both sequential and parallel computations. Thus we have resolved the key problem that previously made the use of the Newton iteration impractical for computation of A^{-1}. Furthermore, our processor bound is optimal because matrix multiplication can be reduced to matrix inversion, see [Borodin and Munro, 75], p. 51. Our parallel and sequential time bounds are optimal o' nearly optimal (up to within the factor O(log n)). We present the asymptotic complexity estimates but the reader can see from our analysis that our algorithms *have small overhead, so they are efficient and practical already for moderate* n.

1.3. Our Results for LINEAR-SOLVE in the Case of Sparse Symmetric Positive Definite Systems.

Our another contribution is a method for LINEAR-SOLVE for sparse symmetric positive definite linear systems, which in many practical cases provides a further order of magnitude improvement in processor bounds without significant additional time cost. Our algorithm drastically differs from the known methods for polylog time solution of linear systems, which make no use of the sparsity structure of the matrix A. To characterize the linear systems Ax=b that our algorithm solve, we will use the two following definitions.

Definition 1.1. Let **C** be a class of undirected graphs closed under the subgraph relation, that is, if $G \in \mathbf{C}$ and G' is a subgraph of G, then $G' \in \mathbf{C}$. The graphs of that class **C** are *s(n)-separable* if there exist constants $n_0 > 0$ and α, $0 < \alpha < 1$, such that for each graph $G \in \mathbf{C}$ 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 (hence S is said to be an *s(n)-separator* of G and the class C is said to *have the* s(n)*-separator property).*

Binary trees are obviously 1-separable. A d-dimensional grid (of a uniform size in each dimension) is $n^{1-(1/d)}$-separable. [Lipton and Tarjan, 79] show that the planar graphs are $\sqrt{8n}$-separable and that every n-vertex finite element graph with $\leq$ k boundary vertices in every element is $4\lfloor k/2 \rfloor \sqrt{n}$-separable.

Definition 1.2. Given an $n \times n$ symmetric matrix $A = (a_{ij})$, we define $G(A)=(V,E)$ to be the undirected graph with vertex set $V=\{1,...,n\}$ and edge set $E=\{\{i,j\} \mid a_{ij} \neq 0\}$. A is *sparse* if $|E| = o(n^2)$.

The very large linear systems $Ax=b$ that arise in practice often are 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 difference equations whose graphs are generally planar graphs or 3-dimensional grids. The standard weather prediction model for the U.S.A., for example, consists of a 3-dimensional grid with a very large number n of grid points, but this grid has only a constant height h between 7 and 20, and hence it is at most $\sqrt{hn}$-separable.

In general, the inverse of a sparse matrix A (even of one with small separators) is dense. Our algorithm for LINEAR-SOLVE avoids computing the inverse matrix, and instead computes a special factorization of A. For sparse matrices with small separators, our polylog time algorithm yields processor bounds that are an order of magnitude lower than the bounds attained by other polylog time parallel algorithms, which compute the inverse matrix. Specifically, given an

$n \times n$ positive definite symmetric matrix A such that G(A) is s(n)-separable and s(n) is of the form n^{σ} for a constant σ, then we can compute the special recursive factorization of A in $O((\log n)(\log s(n))^2)$ time using $|E| + M(s(n))$ processors. Then, given this recursive factorization, the solution of $Ax=b$ for any given b requires only $O(\log n \log s(n))$ time and $|E| + (s(n))^2$ processors.

The key idea is to use a nested dissection of G(A) in order to reduce the problem to inverting dense matrices of sizes at most $s(n) \times s(n)$. The idea of nested dissection is well known for sequential methods. It was first proposed by [George, 73] for grid graphs and later generalized by [Lipton, Rose, and Tarjan, 79] to graphs with small separators. We are the first to apply a nested dissection to yield a parallel algorithm with polylog time bounds. The extension of the idea of nested dissection from the sequential case to the parallel case was not immediate since many sets of separators must be eliminated at each parallel step. Furthermore, in the parallel case we need a special recursive factorization of A, distinct from the LDL^T-factorization used in sequential nested dissection.

Let us comment on the complexity of our algorithm. At first we will assume the practical bound $M(n) = n^3$ for matrix multiplication. It is significant that our parallel nested dissection algorithm has processor bounds that are substantial practical improvements over the previously known bounds.

Let G be a fixed planar graph such that $O(\sqrt{n})$-separators are known for G and its subgraphs. (For example, G might be a $\sqrt{n} \times \sqrt{n}$ grid graph). Then for any $n \times n$ matrix A such that $G = G(A)$ our parallel nested dissection algorithm takes $O(\log n)^3$ time and $n^{1.5}$ processors to compute the special recursive factorization of A, and then $O(\log n)^2$ time and n processors to solve any linear system $Ax=b$ with A fixed. We have the same time bounds and the processor bounds $n^{1.5}k^3$ and nk^2, respectively, if G(A) is an n-vertex finite element graph with $\leq k$ vertices on the boundary of each face. In yet another example, if G(A) is a 3-

factored as LL^T where L is a linear triangular matrix, see [Wilkinson, 65], [Golub and van Loan, 83]. Furthermore such a decomposition $A_0 = LL^T$ is unique if L has positive diagonal entries. Then (4.2) and (4.3) imply similar decompositions for the matrices A_h, X_h for all h, so that all of those matrices are symmetric positive definite. Let us designate $A_h = L_h L_h^T$, $X_h = \tilde{L}_h \tilde{L}_h^T$. Then $A_h^{-1} = L_h^{-1}(L_h^{-1})^T$, $X_h = \tilde{L}_h(\tilde{L}_h^{-1})^T$ and (4.2) and (4.3) imply that

$$L_h = \begin{bmatrix} \tilde{L}_h & 0 \\ Y_h X_h^{-1} & L_{h+1} \end{bmatrix}, \quad L_h^{-1} = \begin{bmatrix} \tilde{L}_h^{-1} & -X_h^{-1} Y_h^T \\ 0 & L_{h+1}^{-1} \end{bmatrix},$$

h=0,1,...,d-1. Let $\tilde{v}$ be a column-vector such that $||\tilde{v}|| = 1$, $||\tilde{L}_h \tilde{v}|| = ||\tilde{L}_h||$, see (2.1). Then, padding $\tilde{v}$ with zeros at the bottom, we will obtain a vector v such that $||v|| = 1$ and $||L_h v|| \geq ||\tilde{L}_h \tilde{v}|| = ||\tilde{L}_h||$, so that $||L_h|| \geq ||\tilde{L}_h||$. Similarly $||L_h|| \geq ||L_{h+1}||$, $||L_h^{-1}|| \geq ||L_{h+1}^{-1}||$, $||L_h^{-1}|| \geq ||\tilde{L}_{h+1}^{-1}||$.

By Lemma 3.4, all of those four bounds combined imply that

$$||A_h|| \geq ||X_h||, \ ||A_h|| \geq ||A_{h+1}||,$$
$$||A_h^{-1}|| \geq ||X_h^{-1}||, \ ||A_h^{-1}|| \geq ||A_{h+1}^{-1}|| \text{ for all } h=0,1,...,d-1.$$

Surely cond A = cond A_0, so these inequalities immediately imply Lemma 4.4. Q.E.D.

Remark 4.1. The algorithm of this section can be extended to the cases where A is an arbitrary matrix with small separators and such that all "intermediate" submatrices X_h are well-conditioned. The latter property, however, may not hold even for some symmetric matrices A such as $\begin{bmatrix} \epsilon I & I \\ I & \epsilon I \end{bmatrix}$ where $|\epsilon|$ is small.

Lemma 4.2(b) implies

Lemma 4.5. *The matrix product* $Y_h X_h^{-1} Y_h^T$ *can be decomposed into* N_h *products of pairs of matrices of sizes at most* $n_{h,k} \times n_{h,k}$ *for* $k=1,...,N_h$, *where* $n_{h,k} = |S_{h,k}|$. *These products can be computed in time* $O(\log s(n))^2$ *using a total*

some $0 < l < h-1$, unless $j^* = j^{**}$ (we will omit the latter trivial case). Thus there exists a path $\{j^*+\delta_h = j_1,j_2\},\{j_2,j_3\}, \ldots, \{j_{l-1},j_l = j^{**}+\delta_h\}$ in E_h visiting only vertices $j_1,\ldots,j_l$ in R_h; but the induction hypothesis (a) implies that there can be no edge in E_h between $R_{h,k}$ and R_{h,k^*} for $k \neq k^*$, so $j_1,\ldots,j_l \in R_{h,k}$ for some unique k. Thus we have established the case (b) of the lemma for E_{h+1}.

Next suppose the case (a) of Lemma 4.2 does not hold for h+1; so there exists a path p in G_{h+1} between some $i \notin V_{h^*,k}$ and $j \in R_{h^*,k}$ where $h^* \geq h+1$ but p visits no vertex v with $\pi(v) > \delta_{h^*+1}$. Case (b) for h implies that $\{i,j\} \in E_{h+1} - E_h$ only if there is a path in G_h between i and j containing only vertices $j_1,\ldots,j_l \in R_h$ with $\pi(j_r) \leq \delta_{h+1}$, for $r=1,\ldots,l$. Thus we can construct from p a path p' in G_h between i and j that visits no vertex v with $\pi(v) > \delta_{h^*+1}$, violating the induction hypothesis for (a), a contradiction. Thus we have shown that (a) holds for G_{h+1}, establishing the induction hypothesis for h+1. Q.E.D.

Lemma 4.2(a) implies that E_h contains no edge between $R_{h,k}$ and R_{h,k^*} for $k \neq k^*$. Since $\max_k |R_{h,k}| \leq \max_k s(|V_{h,k}|) \leq s(\alpha^{d-h}n)$, we immediately arrive at

Lemma 4.3. *X_h is a block-diagonal matrix consisting of $N_h \leq 2^{d-h}$ square blocks, where each block is of size at most $s(\alpha^{d-h}n) \times s(\alpha^{d-h}n)$.*

Lemma 4.3 implies that the inversion of X_h^{-1} can be reduced to $N_h \leq 2^{d-h}$ parallel inversions of dense matrices (i.e., one dense matrix is associated with each $R_{h,k}$), each of size at most $s(\alpha^{d-h}n) \times s(\alpha^{d-h}n)$. By Corollary 2.1, this can be done in $O(\log s(n))^2$ time, $N_h M(s(\alpha^{d-h}n)) \leq 2^{d-h} M(s(\alpha^{d-h}n))$ processors. By our assumptions, $\alpha^{\omega\sigma} \leq 1/2$, so this processor bound is $O(M(s(n)))$. In fact, we need to prove the following lemma in order to be able to apply Corollary 2.1.

Lemma 4.4. cond $X_h \leq$ cond A *for all* h.

Proof. $A_0 = PAP^T$ is a symmetric positive definite matrix, so it can be

entry at the i-th row and the j-th column of A_i is nonzero provided that here the rows are counted bottom-up and the columns from right to left.

The *fill-in* at stage h is the set of edges that are in E_h but not in E_{h-1}. The following technical lemma upper bounds this fill-in and also provides some useful information about the connectivity structure of G_h.

Lemma 4.2. *Let* $h \geq 0$. *Then*

a) *if* p *is a path in* G_h *between* $i \notin V_{h^*,k}$ *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) $E_{h+1} \subseteq \{\{i,j\} \in E_h \mid i,j \notin R_h\}$

$$\cup\{\{i,j\} \mid \exists k \exists \{i,j_1\},\{j_1,j_2\}, \ldots, \{j_{l-1},j_l\}, \{j_l,j\} \in E_h\}$$

where $j_1, \ldots, j_l \in R_{h,k}$ *and* $\pi(i) > \delta_h$, $\pi(j) > \delta_h$}.

Proof (by induction on h). We first consider the case (a) with h=0. Observe that if there is a path p in G_0 between some $i \notin V_{h^*,k}$ and $j \in R_{h^*,k}$ for any $h^* \geq 0$, then p must contain a vertex, say j^*, of the separator set S_{h^*+1,k^*}, where $(V_{h^*+1,k^*},S_{h^*+1,k^*})$ is the parent node of $(V_{h^*,k},S_{h^*,k})$ in T_G. This vertex j^* has number $\pi(j^*) > \delta_{h^*+1}$, as required.

Suppose that Lemma 4.2(a) holds for some fixed h and for all $h^* \geq h$. By definition, $A_{h-1} = Z_h - Y_h X_h^{-1} Y_h^T$. Let $X_h = (x_{ij})$, $X_h^{-1} = \bar{X}_h = (\bar{x}_{ij})$, $Y_h = (y_{ij})$, $Z_h = (z_{ij})$, and $W_h = Y_h X_h^{-1} Y_h^T = (w_{ij})$. If $\{i,j\} \in E_{h+1}$, then $i,j \notin R_h$, by the definition of R_h, so $\pi(i) > \delta_{h+1}$, $\pi(j) > \delta_{h+1}$. Furthermore we have $z_{i-\delta_h,j-\delta_h} \neq 0$ or otherwise $w_{i-\delta_h,j-\delta_h} \neq 0$. If $z_{i-\delta_h,j-\delta_h} \neq 0$, then $\{i,j\} \in E_h$ as claimed. On the other hand, if $w_{i-\delta_h,j-\delta_h} \neq 0$, then $\exists\, j^*+\delta_h, j^{**}+\delta_h \in R_h$ such that $y_{ij^*} \neq 0$, $\bar{x}_{j^*,j^{**}} \neq 0$ and $y_{jj^{**}} \neq 0$. It follows that $\{i,j^*+\delta_h\},\{j,j^{**}+\delta_h\} \in E_h$. Furthermore the Cayley-Hamilton Theorem gives $X_h^{-1} = c_0 I + c_1 X_h + \ldots + c_{h-1} X_h^{h-1}$ for some scalars $c_0, c_1, \ldots, c_{n-1}$. Hence $\bar{x}_{j^*j^{**}} \neq 0$ implies that the (j^*,j^{**}) entry of X_h^l is not 0, for

Observe that, by definition, $R_h \cap R_{h^*} = \phi$ if $h \neq h^*$ and that $V = \bigcup_{h=0}^{d} R_h$.

Let π: $\{1,...,n\} \rightarrow \{1,...,n\}$ be any enumeration of the n vertices of G such that, if $v \in R_h$, $v^* \in R_{h^*}$ for $h^* > h$, then $\pi(v) < \pi(v^*)$. Thus the elements of $\pi(R_h)$ are in the range from $\delta_h+1,...,\delta_{h+1}$ where $\delta_h = \sum_{g<h} |R_g|$. Such an enumeration can be easily computed in total time $O(\log n)^2$ using $n/\log n$ processors by first numbering the vertices of $R_d = S_d$ by n, n-1,... and then numbering (also in the decreasing order) all previously unnumbered vertices of R_h of height h for each h=d-1,d-2,...,0.

We now define the permutation matrix P such that $P_{ij} = 1$ if $j = \pi(i)$ and $P_{ij} = 0$ otherwise. Then we define the initial matrix $A_0 = PAP^T$. Recursively, for h=0,1,...,d-1,

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

is 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 $A_{h+1} = Z_h - Y_h X_h^{-1} Y_h^T$. Thus at stage h we have eliminated the elements associated with R_h. Note that A_{h+1} is symmetric positive definite if A_h is, compare the proof of Lemma 4.4 below. We now claim that we can compute A_{h+1} from X_h, Z_h and Y_h in time $O(\log s(n))^2$ using at most $M(s(n))$ processors. To prove this, we must investigate the sparsity structure of the latter submatrices of A_h.

Let $A_h = (a_{ij}^{(h)})$. We define an associated graph $G_h = (V_h,E_h)$ with vertex set $V_h = \{\delta_h+1, \delta_h+2,...,n\}$ and edge set $E_h = \{\{i+\delta_h, j+\delta_h\} \mid a_{ij}^{(h)} \neq 0\}$; that is, G_h is derived from $G(A_h)$ by adding δ_h to each vertex number, see Figures 1,2,3,4 and 5 below. This enumeration implies that $\{n-i,n-j\} \in E_n$ if and only if the

Efficient parallel computation of s(n)-separators is not simple in general but it is rather straightforward in the practically important cases of grid graphs (see illustrative Figures 1,2,3,4,5 below); similarly such computation is simple for many finite element graphs, compare [George,73].

To prove Theorem 4.1, we will begin with defining a binary tree T_G for each graph $G \in \mathbf{C}$, see Figure 6. Suppose G=(V,E) has n vertices. If $n \leq n_0$, where n_0 is a given constant, (see Definition 1.1), we let T_G be the trivial tree with no edges and with the single leaf (V,S) where S=V. Else, if $n > n_0$, we recall that 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 is no edge in E between the sets V_1 and V_2, and furthermore $|V_1| \leq \alpha n$, $|V_2| \leq \alpha n$, and $|S| \leq s(n)$. 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 two subtrees T_{G_1}, T_{G_2} of T_G where G_h is the subgraph of G induced by the vertex set $S \cup V_h$ for h=1,2.

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 number of nodes of height h in T_G. Since T_G is a binary tree, $N_h \leq 2^{d-h}$. Let $(V_{h,1}, S_{h,1})$, ..., (V_{h,N_h}, S_{h,N_h}) be a list of these nodes of height h and let $S_h = \bigcup_{k=1}^{N_h} S_{h,k}$. Observe that by the s(n)-separable property of $\mathbf{C}$, $|V_{h,k}| \leq \alpha^{d-h} n$ and $|S_{h,k}| \leq s(\alpha^{d-h} n)$ for each $h \geq 0$ and $k = 1,...,N_h$. Thus $d \leq c^* \log n$, $c^* = 1/\log(1/\alpha)$ for $\alpha^d n \leq n_0$, $|V_{0,k}| \leq n_0$ and $S_{0,k} = V_{0,k}$ for all k, by the definition of the tree T_G.

For each $k = 1,...,N_h$ such that $S_{h,k} \neq V_{h,k}$, let $R_{h,k}$ denote the set of all elements of $S_{h,k}$ that are not in S_{h^*} for $h^* > h$. (Actually $R_{h,k} = S_{h,k} - \cup S_{h^*,k^*}$ where the union is over all ancestors $(V_{h^*,k^*}, S_{h^*,k^*})$ of $(V_{h,k}, S_{h,k})$.) The s(n)-separable property implies that $R_{h,k_1} \cap R_{h,k_2} = \phi$ if $k_1 \neq k_2$. Let $R_h = \bigcup_{k=1}^{N_h} R_{h,k}$.

symmetric positive definite matrix A such that cond $A \leq n^{O(1)}$ *and* $G=G(A)$, *we can numerically compute a recursive* $s(n)$*-factorization in time* $O(\log n(\log s(n))^2)$ *using* $|E| + M(s(n)) + n/\log n$ *processors (where* $M(s(n))$ *is the number of processors sufficient in order to multiply two* $s(n) \times s(n)$ *matrices in time* $O(\log s(n))$). *Furthermore that recursive* $s(n)$*-factorization satisfies Lemmas 4.5 and 4.6 below. Whenever such a recursive* $s(n)$*-factorization of* A *is available,* $O(\log n \log s(n))$ *time and* $|E| + s(n)^2$ *processors suffice to solve a system of linear equations* $Ax=b$ *for any given column vector* b *of length* n.

Corollary 4.1. *Fix an n-vertex planar graph G such that* $O(\sqrt{n})$*-separators for G and for its subgraphs have been precomputed. If* A *is an* $n \times n$ *symmetric positive definite matrix with* $G=G(A)$ *and if* cond $A \leq n^{O(1)}$, *then we can numerically compute a recursive* $O(\sqrt{n})$*-factorization of* A *in time* $O(\log n)^3$ *using* $M(\sqrt{n}) \leq n^{1.25}$ *processors. Whenever such a recursive factorization is available,* $O(\log n)^2$ *time and* n *processors suffice to solve* $Ax=b$ *for a given* b.

Corollary 4.2. *Fix an n-vertex finite element graph G with at most k vertices on the boundary of every face. Let* $O(k\sqrt{n})$*-separators for G and for its subgraphs have been precomputed. If A is an* $n \times n$ *symmetric positive definite matrix with* $G=G(A)$ *and if* cond $A+\leq n^{O(1)}$, *then we can numerically compute a recursive* $O(k\sqrt{n})$*-factorization of A in time* $O(\log n)^3$ *using* $M(k\sqrt{n}) \leq k^{2.5}n^{1.25}$ *processors. Whenever such a recursive factorization is available,* $O(\log n)^2$ *time and* nk^2 *processors suffice to solve* $Ax=b$ *for a given b.*

Corollary 4.3. *If A is an* $n \times n$ *symmetric positive definite matrix with a d-dimensional grid graph (for any* $d \geq 2$) *and if* cond $a \leq n^{O(1)}$, *then we can numerically compute a recursive* $(n^{1-(1/d)})$*-factorization of A in time* $O(\log n)^3$ *using* $M(n^{1-(1/d)}) < n^{2.5-(2.5/d)}$ *processors. Whenever such a recursive factorization is available,* $O(\log n)^2$ *time and* $n^{2-(2/d)}$ *processors suffice to solve* $Ax=b$ *for a given b.*

and X_h is a block-diagonal matrix consisting of square blocks of sizes at most $s(\alpha^{d-h}n) \times s(\alpha^{d-h}n)$ where $n_0 \geq \alpha^d n$, n_0 is a constant. (Here and hereafter W^T denotes the transpose of a matrix W.) Such a recursive s(n)-factorization has length at most $d \leq O(\log n)$ (whereas a customary LDL^T factorization has length n). This recursive factorization is said to be *numerically computed* if the computed approximants of the induced matrices $A_1, \ldots, A_d$ satisfy (4.1) within error norm 2^{-n^c}, for a positive constant c.

Observe that, by definition of a recursive s(n)-factorization, we have for h=0,...,d-1 the identity

$$A_h = \begin{bmatrix} I & 0 \\ Y_h X_h^{-1} & I \end{bmatrix} \begin{bmatrix} X_h & 0 \\ 0 & A_{h+1} \end{bmatrix} \begin{bmatrix} I & X_h^{-1} Y_h^T \\ 0 & I \end{bmatrix} \tag{4.2}$$

and hence

$$A_h^{-1} = \begin{bmatrix} I & -X_h^{-1} Y_h^T \\ 0 & I \end{bmatrix} \begin{bmatrix} X_h^{-1} & 0 \\ 0 & A_{h+1}^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -Y_h X_h^{-1} & I \end{bmatrix} . \tag{4.3}$$

Thus given a recursive s(n)-factorization (4.1), it is easy to recursively compute $A^{-1}b$ for any column vector b of length n.

Lemma 4.1. For all constants α, α^*,c such that $1/2 \leq \alpha^* < \alpha < 1$, if $s(n) \leq cn^\sigma$ and if **C** is a class of s(n)-separable graphs with respect to α, then **C** is $s^*(n)$-separable with respect to α^*, where $s^*(n) \leq cn^\sigma/(1-\alpha^\sigma)$.

Lemma 4.1 can be proven following [Lipton and Tarjan, 79], see the end of this section. Without loss of generality, in the following we will assume that $\alpha = 1/2$ (see Lemma 4.1), that $M(n) = n^\omega$ and that $s(n) = O(n^\sigma)$ for fixed constants ω and σ such that $0 \leq \sigma < 1$, and $2 \leq \omega \leq 3$. Furthermore we will assume that $\sigma \geq 1/\omega$ such that $\alpha \leq 1/2^{\omega\sigma}$ (see Lemma 4.1), so $\alpha^{\omega\sigma} \leq 1/2$.

Theorem 4.1. *If* **C** is a class of s(n)-*separable graphs and if the* s(n)-*separators of a graph* $G \in \mathbf{C}$ *and of its subgraphs are known, then given an* $n \times n$

Lemma 3.5. *Let* B *and* t *be defined by (2.6). Let* μ *be an eigenvalue of* $R(B) = I - BA$. *Then* $0 \leq \mu \leq 1 - 1/((\text{cond } A)^2 n)$.

Proof. Let $R(B)v = \mu v$ for $v \neq 0$. Then $(I - tA^H A)v = v - tA^H Av = \mu v$. Therefore $A^H Av = \lambda v$ for $\lambda = (1-\mu)/t$ so λ is an eigenvalue of $A^H A$. Corollary 3.2 implies that $1/||A^{-1}||^2 \leq \lambda = (1 - \mu)/t \leq ||A||^2$. It immediately follows that

$$1 - t||A||^2 \leq \mu \leq 1 - t/||A^{-1}||^2. \tag{3.1}$$

It remains to recall (2.6) and to apply the inequalities of Lemma 3.3. Q.E.D.

Since μ is an arbitrary eigenvalue of R(B), $\rho(R(B)) \leq 1 - 1/((\text{cond } A)^2 n)$. On the other hand, $||R(B)|| = \rho(R(B))$ since $R(B) = I - tA^H A$ is Hermitian. This completes the proof of Lemma 2.4. Q.E.D.

4. Parallel Generalized Nested Dissection

In this section, we fix a class **C** of undirected graphs, which are s(n)-*separable (with respect to constants* n_0 *and* α*), see* Definition 1.1. Let A be an $n \times n$ symmetric positive definite matrix with graph $G=G(A)$ in class **C** (see Definitions 3.1 and 1.2). Even if A is sparse, and if G(A) has small separators, A^{-1} may not be sparse (and in fact if G(A) is connected, A^{-1} may have no zero entries). We will describe an efficient parallel algorithm that computes a special recursive factorization of A. With that factorization available, it will become very easy to solve the system of linear equations of the form $Ax=b$ for any given vector b.

Definition 4.1. A *recursive* s(n)*-factorization of a matrix* A *with respect to* α, $0<\alpha<1$, is a sequence of matrices $A_0, A_1, \ldots, A_d$ such that $A_0 = PAP^T$, P is an $n \times n$ permutation matrix and for $h=0,1,\ldots,d-1$,

$$A_h = \begin{bmatrix} X_h & Y_h^T \\ Y_h & Z_h \end{bmatrix}, \quad Z_h = A_{h+1} + Y_h X_h^{-1} Y_h^T, \tag{4.1}$$

symmetric positive definite. (W^HW is Hermitian since $(W^HW)^H = W^H(W^H)^H = W^HW$, compare Definition 2.2.)

We will use the 3 following well known results, see [Atkinson, 78], pp. 416-431, or [Wilkinson, 65]. (Recall that $||W||$ denotes the 2-norm of a matrix W.)

Lemma 3.1. $||W|| = ||W^H||$ *for all* W; $||W|| = \rho(W)$ *if* $W = W^H$.

Lemma 3.2. *All eigenvalues of a Hermitian positive semidefinite matrix are nonnegative.*

Lemma 3.3. *Let* $W = (w_{ij})$. *Then* $||W^HW|| = \rho(W^HW) = ||W||^2 \leq ||W^HW||_1 \leq \max_i \sum_j |w_{ij}| \max_j \sum_i |w_{ij}| \leq n||W^HW||$.

Applying Lemma 3.3 to $W = A^H$ we obtain that $||A^H||^2 \leq 1/t$ where t is defined by (2.6). Therefore, by Lemma 3.1, $||A^H|| \leq 1/(t||A||)$. Taking into account that $||A|| \geq \max_{i,j} |a_{ij}|$ we derive the first 2 inequalities of Lemma 2.4.

Corollary 3.1. *Let* $A = (a_{ij})$, B *be defined by (2.6). Then* $||B|| \leq 1/||A|| \leq 1/\max_{i,j} |a_{ij}|$.

It remains to prove the last inequality of Lemma 2.4.

Lemma 3.4. *Let* λ *be an eigenvalue of a nonsingular matrix* W. *Then* $1/\lambda$ *is an eigenvalue of* W^{-1}.

Proof. Surely $\lambda \neq 0$ for nonsingular W. Let $Wv = \lambda v$ for a vector $v \neq 0$. Then $Wv \neq 0$ and $W^{-1}(Wv) = v = (1/\lambda)\lambda v = (1/\lambda)Wv$. Q.E.D.

Corollary 3.2. *Let* λ *be an eigenvalue of* A^HA *and* A *be nonsingular. Then* $1/||A^{-1}||^2 \leq \lambda \leq ||A||^2$.

Proof. $\lambda \leq \rho(A^HA) = ||A||^2$ by Definition 3.1 and Lemma 3.3. On the other hand, $(A^HA)^{-1} = A^{-1}(A^{-1})^H$, so $(A^HA)^{-1}$ is a Hermitian positive definite matrix. By the virtue of Lemma 3.4, $1/\lambda$ is an eigenvalue of $(A^HA)^{-1}$. Consequently $1/\lambda \leq \rho((A^HA)^{-1}) = \rho(A^{-1}(A^{-1})^H) = ||A^{-1}||^2$. Q.E.D.

respectively, compare (2.1). Let $(\text{cond } W)_s = ||W||_s||W^{-1}||_s$ *if* W *is nonsingular,* $s = \infty$ *or* s=1.

Remark 2.1. Using Definition 2.3 we may rewrite (2.6) as

$$B = tA^H,\ t = 1/(||A||_\infty\ ||A||_1).$$

Substituting this into (3.1) and using Lemma 3.3 of the next section, we deduce that

$$||R(B)|| \le 1-1/(||A||_\infty||A||_1||A^{-1}||^2) \le \quad (2.9)$$
$$1-1/((\text{cond } A)_\infty\ (\text{cond } A)_1).$$

Furthermore we may choose

$$B = \tilde{t}A^H,\tilde{t} = 1/||A^HA||_1 \ge 1/(||A^H||_1||A||_1) = 1/(\ |\ A||_\infty||A||_1)$$

and deduce from (3.1) and Lemma 3.3 that, for this choice of B,

$$||R(B)|| \le 1-1/(||A^HA||_1||A^{-1}||^2), \quad (2.10)$$

which is a further small improvement over (2.8) and (2.9). (2.9) and (2.10) lead to the respective improvements of Corollary 2.2.

Remark 2.2. Lemma 2.4 can be proven for any (nonsingular) matrix A. For certain classes of matrices A there exist other options for choosing an approximate inverse B. In Appendix B we indicate such options for some important classes of matrices including Hermitian (real symmetric) positive definite, diagonally dominant and triangular matrices.

3. Proof of Lemma 2.4.

Definition 3.1, $\rho(W)$, the *spectral radius of a matrix* W, is the maximum magnitude of the eigenvalues of W.

Definition 3.2. A matrix that can be represented as W^HW for some matrix W is called a *Hermitian positive semidefinite matrix (or a Hermitian nonnegative definite matrix).* A nonsingular Hermitian positive semidefinite matrix is called *Hermitian positive definite.* A real Hermitian positive definite matrix is called

$$||B|| \leq 1/||A|| \leq 1/\max_{i,j} |a_{ij}|, \quad ||R(B)|| \leq 1-1/((\text{cond } A)^2 n). \tag{2.8}$$

We also immediately verify the following estimate.

Lemma 2.5. *Computing B by (2.6) costs only* $3n^2-2n+1$ *arithmetic operations and* $2n-2$ *comparisons that can be performed in O(log n) time using* n^2 *processors.*

Combining Theorem 2.1, Lemmas 2.4 and 2.5, and the inequality of (2.7) we get

Corollary 2.1. *Let* $A = (a_{ij})$, c *be arbitrary constant,* $\text{cond } A \leq n^{O(1)}$. *Then* $O(\log^2 n)$ *time,* $M(n)$ *processors suffice to compute a matrix* $\tilde{A}^{-1}$ *such that*

$$||A^{-1} - \tilde{A}^{-1}|| \leq 2^{-n^c} / ||A|| \leq ||A^{-1}||2^{-n^c} / \text{cond } A \leq ||A^{-1}||2^{-n^c}.$$

Although Corollary 2.1 covers all instances A of practical interest, we will also state the following immediate generalization of that corollary, compare (2.8) and Theorem 2.1.

Corollary 2.2. *For any number* k *and for any nonsingular* $n \times n$ *matrix A, it is sufficient to use* $O(k \log n)$ *parallel steps and* $M(n)$ *processors in order to compute a matrix* $\tilde{A}^{-1}$ *such that*

$$||\tilde{A}^{-1}|| - ||A^{-1}|| \leq (1-1/(n(\text{cond } A)^2))^{2^k} / ||A|| \leq$$
$$||A^{-1}||(1-1/(n(\text{cond } A)^2))^{2^k}/\text{cond } A \leq ||A^{-1}||(1-1/(n(\text{cond } A)^2))^{2^k}.$$

In particular if $\text{cond } A \leq C$, *then the precision* $||A|| \cdot ||\tilde{A}^{-1} - A^{-1}|| < \epsilon < 1$ *can be assured using* $O(\log n\,(1+|\log(nC^2 \log(1/\epsilon)|))$ *parallel steps and* $M(n)$ *processors, if C and* ϵ *are arbitrary positive constants,* $\epsilon < 1$.

In the next remark and in Appendix B we will use the following definition, compare [Atkinson, 78], [Golub and van Loan, 83], [Wilkinson, 65].

Definition 2.3. $||W||_\infty = \max_i \sum_j |w_{ij}|$, $||W||_1 = \max_j \sum_i |w_{ij}|$ *are the operator norms of a matrix* $W = |w_{ij}|$ *associated with the maximum norm* $||v||_\infty = \max_i |v_i|$ *and with the 1-norm* $||v||_1 = \sum_i |v_i|$ *of a vector* $v = (v_i)$,

binary numbers of the form $a2^{-k}$ where a and k are integers, $|a| < 2^m$, $0 \le k < m$, $m = n^{O(1)}$.

Lemma 2.3 immediately implies the following

Theorem 2.1. *Let (2.2) and (2.4) hold and let* c *be a constant. Then* $O(\log^2 n)$ *time and simultaneously* M(n) *processors suffice in order to compute a matrix* $\tilde{A}^{-1}$ *satisfying (2.5).*

Here (and frequently hereafter) we exploit the possibility to reduce the number of processors by a constant factor k by the price of slowing down the computation k times.

It remains to choose B satisfying (2.4) for a nonsingular A. Hereafter in all expressions $\Sigma_i, \Sigma_j, \max_i, \max_j, \max_{i,j}$ the integer parameters i and j range from 1 to n. Let us specify the vector norm to be *the Euclidean norm,* $||v|| = (\Sigma_i |v_i|^2)^{1/2}$, and let us extend that vector norm to the matrix norm by (2.1). The resulting matrix norm is called the *2-norm.* Let us choose

$$B = tA^H,\ t = 1/(\max_i \sum_j |a_{ij}| \max_j \sum_i |a_{ij}|). \tag{2.6}$$

Definition 2.2. Here and hereafter W^H designates the *Hermitian transpose of a matrix* $W = (w_{ij})$. $W^H = (w_{ji}^*)$, w_{ji}^* being the complex conjugate of w_{ji}. If $W^H = W$, W is called a *Hermitian matrix.* If W is real, then W^H is the transpose of W (which we will designate W^T) and Hermitian W means *symmetric* W, such that $w_{ij} = w_{ji}$ for all i,j.

$$\text{cond } W = ||W|| * ||W^{-1}|| \ge ||I|| = 1 \text{ if W is nonsingular,} \tag{2.7}$$
$$\text{cond } W = \infty \text{ otherwise.}$$

We will prove the next lemma in the next section, compare also Remark 2.1 below and Remark B.1 in Appendix B.

Lemma 2.4. *Let* $A = (a_{ij})$ *be nonsingular, let* B *be defined by (2.6) and* R(B) *be defined by (2.2). Then*

$$R(B) = I - BA, \quad ||R(B)|| = q < 1. \tag{2.2}$$

(Actually (2.2) implies that A and B are nonsingular, see [Atkinson, 78], p. 465.) Note that $||(R(B))^i|| \le q^i \to 0$ as $i \to \infty$ if (2.2) holds.

Lemma 2.1. Let $||(R(B))^i|| \to 0$ as $i \to \infty$. Then $A^{-1} = \sum_{i=0}^{\infty} (R(B))^i B$.

Q.E.D.

Proof. $A^{-1} = (A^{-1}B^{-1})B = (BA)^{-1}B = (I-R(B))^{-1}B = \sum_{i=0}^{\infty} (R(B))^i B$. Q.E.D.

Lemma 2.1 immediately implies

Lemma 2.2. *Let* $B_k^* = (\prod_{h=0}^{k-1} (I+(R(B))^{2^h}))B$. *Then* $A^{-1}-B_k^* = \sum_{i=2^k}^{\infty} (R(B))^i B$.

The relations (2.2) and Lemma 2.2 imply that

$$||A^{-1} - B_k^*|| \le \sum_{i=2^k}^{\infty} q^i ||B|| = q^{2^k} ||B|| \,/\, (1-q). \tag{2.3}$$

The first algorithm for INVERT successively computes $(R(B))^{2^h}$ and B_h^* for $h=0,1,\ldots,k-1$. Recall that, by the virtue of (2.2), $||(R(B))^i|| \le q^i$, so the computation by that algorithm is stable. In Appendix C we will also consider two other algorithms that converge to A^{-1} with the same speed. Those algorithms are superior over the first algorithm for they are not only stable but also self-correcting.

(2.3) implies

Lemma 2.3. *Let (2.2) hold and*

$$q = 1 - 1/n^{O(1)} \text{ as } n \to \infty . \tag{2.4}$$

Let c be a constant. Then $O(\log n)$ *iterations of the algorithm of this section suffice in order to compute a matrix* $\tilde{A}^{-1}$ *such that*

$$||A^{-1} - \tilde{A}^{-1}|| \le 2^{-n^c} ||B||. \tag{2.5}$$

The latter precision suffices for all practical purposes and cannot be reduced further if $|(\log ||B||)| \le n^{O(1)}$ and if the entries of A^{-1} are to be represented by

dimensional grid, so is $n^{2/3}$-separable, then we have the same time bounds as for planar and finite element graphs, and our processor bounds are n^2 and $n^{1.33}$, respectively. Furthermore, if we use more theoretical bounds for matrix multiplication, say $M(n) = n^{2.5}$, then our processor bounds, for computing the special recursive factorization are further decreased to $n^{1.25}$ in planar case, to $n^{1.25}k^{2.5}$ in the case of n-vertex finite element graphs with $\leq k$ vertices per face and to $n^{1.67}$ for the 3-dimensional grid.

1.4. Contents

In Section 2 we present our iterative parallel algorithm for INVERT and its complexity estimates. In Section 3 we prove the main lemma of Section 2. In Section 4 we present our parallel algorithm for LINEAR-SOLVE in the case of sparse symmetric positive definite systems and bound its complexity. In Appendix A we bound the asymptotic parallel complexity of matrix multiplication. In Appendix B we consider the simplified methods for defining an approximate inverses of some special matrices. In Appendix C we consider two alternatives to the iterative algorithm of Section 2. In Appendix D we analyze the errors of our algorithms.

2. Iterative Methods for Parallel Matrix Inversion

Definition 2.1. Let a vector norm be fixed. Then we extend it to the associated operator norm of matrices so that

$$||W|| = \max_{v \neq 0} ||Wv|| \; / \; ||v|| \tag{2.1}$$

for all nonzero vectors v and for all matrices W.

In the following $n^{O(1)}$ denotes the values bounded by a polynomial in n as $n \to \infty$, $A = (a_{ij})$ is a fixed $n \times n$ nonsingular complex matrix (so A^{-1} exists), I denotes the $n \times n$ identity matrix, R(X) denotes $I - XA$, and B is called an *approximate inverse* of A if

of $\sum_{k=1}^{N_h} M(n_{h,k})$ *processors.*

To estimate this number of processors, we need

Lemma 4.6. *For any* $\gamma > 1$, $\sum_{k=1}^{N_h} n_{h,k}^{\gamma} = O(s(n)^{\gamma})$.

Proof. Observe that, by the definition of the tree T_G, if the nodes $(V_{h-1,k_1},S_{h-1,k_1})$ and $(V_{h-1,k_2},S_{h-1,k_2})$ are the children of the node $(V_{h,k},S_{h,k})$, then $n_{h-1,k_1} + n_{h-1,k_2} \leq n_{h,k} + |S_{h,k}| \leq n_{h,k} + s(\alpha^{d-h}n)$ since $|S_{h,k}| \leq s(\alpha^{d-h}n)$. This recursive inequality implies that $\sum_{k=1}^{N_h} n_{h,k}^{\gamma}$ is maximized when only one term is nonzero; in this case $\sum_{k=1}^{N_h} n_{h,k}^{\gamma} \leq \sum_{h^*=h+1}^{d} s(\alpha^{d-h^*}n)^{\gamma} = O(s(n)^{\gamma})$ because we have assumed that $s(n) = O(n^{\sigma})$. Q.E.D.

Since we have also assumed that $M(n) = n^{\omega}$, Lemma 4.6 implies that $\sum_{k=1}^{N_h} M(n_{h,k}) = O(M(s(n)))$. By slowing the parallel computation down by a constant factor, we can reduce the processor bounds to $M(s(n))$ thus arriving at

Lemma 4.7. $A_{h+1} = Z_h - Y_h X_h^{-1} Y_h$ *can be computed in time* $O(\log s(n))^2$ *using* $M(s(n))$ *processors.*

Since there are only $d=O(\log n)$ stages, each taking $O(\log s(n))^2$ time and using $M(s(n))$ processors, we conclude that $O(\log n(\log s(n))^2)$ is the total time required in order to numerically compute the recursive $s(n)$-factorization of A using $M(s(n))$ processors.

Thereafter, we can solve $Ax=b$ for any given column-vector b of length n by computing $x=A^{-1}b$ by recursive application of equation (4.2) for $h=0,1,...,d-1$. The stage h of this "backsolving" computation requires parallel multiplication of a matrix of size $n_{h,k} \times n_{h,k}$ times a column-vector for each $k=1,2,...,N_h$. Thus stage h can be done in time $O(\log s(n))$ using $\sum_{k=1}^{N_h} n_{h,k}^2$ processors. By Lemma 4.6,

this bounds the number of processors to $O(s(n)^2)$, and a slowdown by a constant factor reduces this processor bound to $s(n)^2$. The total time for all $d=O(\log n)$ stages of such backsolving computation is $O(\log n \log s(n))$. This completes the proof of Theorem 4.1, except it remains to justify our assumption that $\alpha^{\omega\sigma} \leq 1/2$ by proving Lemma 4.1.

Proof of Lemma 4.1. Let us first assume that $\alpha^* > 1/2$. Given any graph $G \in \mathbf{C}$, where $G = (V,E)$ and $n = |V| \geq n_0$, we can partition V into disjoint sets V_1^*, V_2^*, S^* such that $|V_1^*| \leq \alpha^* n$, $|V_2^*| \leq \alpha^* n$, $|S^*| \leq O(s(n))$ and V_1^* and V_2^* are not connected by E. That partition can simply be found by appropriately combining the $N_h \leq (\alpha^* - 1/2)1/\log_2\alpha = O(1)$ vertex subsets found at depth $h = d - \log((\alpha^* - 1/2)/\log \alpha)$ of T_G. The separator S_h^* is defined to be the union of all separators of depths $\geq h$ in T_G, and since there is only a constant number of such separators, $|S_h^*| \leq O(s(n))$. Then V_1^* can be any maximal collection of the vertex subsets of depth h in T_G where $n/2 \leq |V_1^*| \leq \alpha^* n$. Letting $V_2^* = V - V_1^* - S_h^*$, we have $|V_2^*| \leq n-(n/2) \leq n/2 \leq \alpha^* n$, so $\mathbf{C}$ is $s^*(n)$-separable with respect to α^* where $s^*(n) = O(s(\ ))$ provided that $\alpha^* > 1/2$. This already justifies the assumption that $\alpha^{\omega\sigma} \leq 1/2$ if $\sigma > 1/\omega$, which is actually sufficient for our purpose. The extension to the case $\alpha^* = 1/2$ and the decrease of the constant hidden in $O(s(n))$ to $1/(1-\alpha^\sigma))$ are obtained similarly to the proof of Corollary 3 of [Lipton and Tarjan, 79]. Q.E.D.

Figure 1. The 7×7 grid graph G_0, with elimination numbering

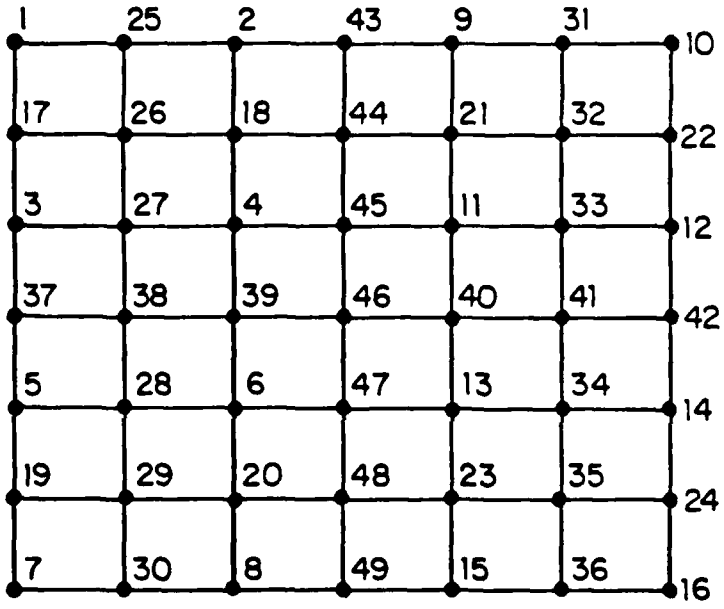


Figure 2. The graph G, derived from G_0 by simultaneous elimination of $R_0 = \{1,2,\ldots,16\}$

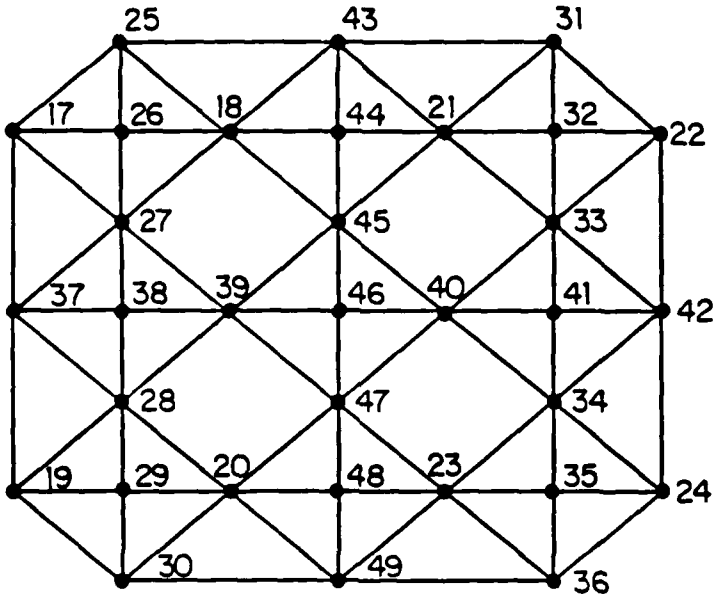


Figure 3. The graph G_2 derived from G_1 by simultaneous elimination of $R_1 = \{17,18,...,24\}$.

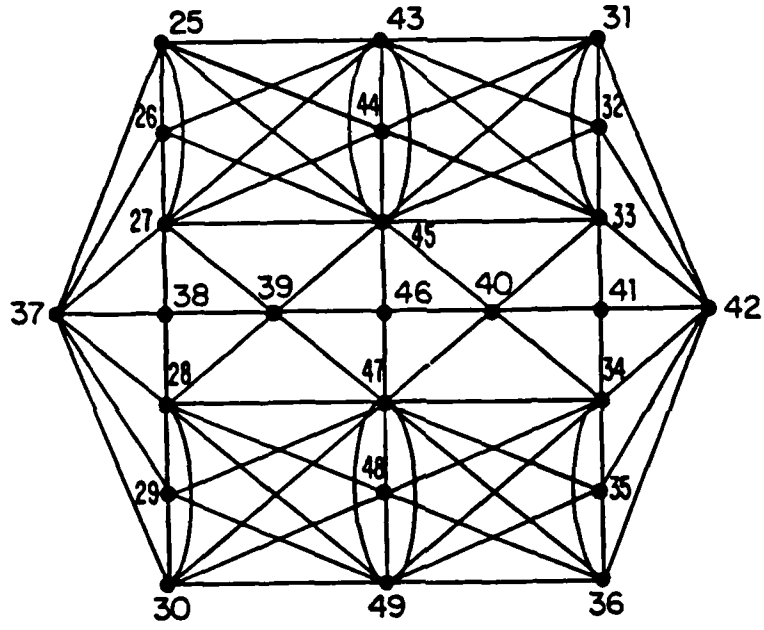


Figure 4. The graph G_3 derived from G_2 by simultaneous elimination of $R_2 = \{25,26,...,36\}$.

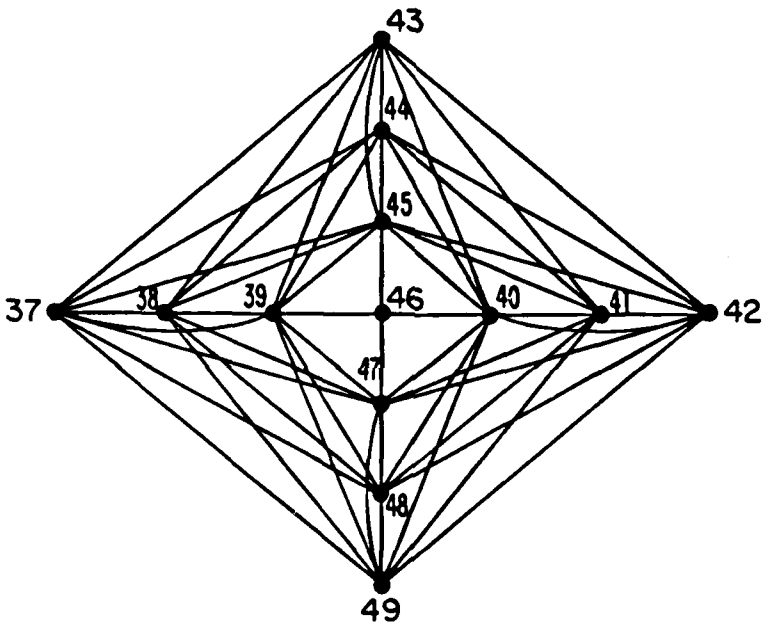


Figure 5. The graph G_4 derived from G_3 by simultaneous elimination of $R_3 = \{37,38,\ldots,42\}$.

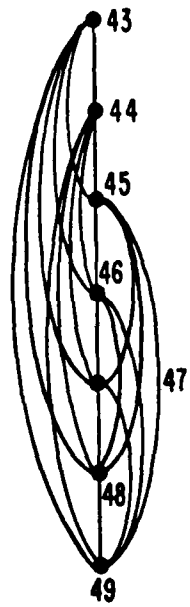
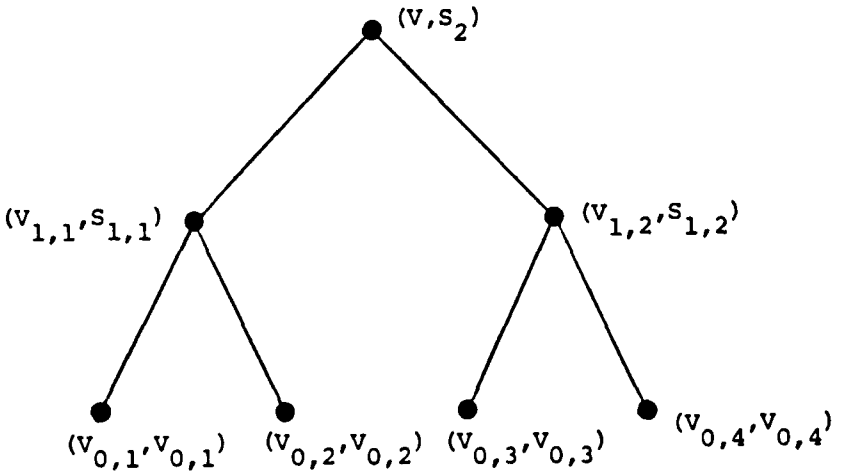


Figure 6. Tree T_G in the case d=2.



References

Atkinson, K.E., *An Introduction to Numerical Analysis*, Wiley, New York (1978).

Berkowitz, S., "On Computing the Determinant in Small Parallel Time Using a Small Number of Processors", *Information Processing Letters* 18, 147-150 (1984).

Bodewig, E., *Matrix Calculus*, Second Edition, Interscience Publishers, Inc., New York; North-Holland Company, Amsterdam (1959).

Bojańczyk, A., "Complexity of Solving Linear Systems in Different Models of Computation", *SIAM J. on Numerical Analysis* 21(3), 591-603 (1984).

Borodin, A., J. von zur Gathen, and J. Hopcroft, "Fast Parallel Matrix and GCD Computations", *Information and Control* 52(3), 241-256 (1982).

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

Chandra, A.K., "Maximal Parallelism in Matrix Multiplication", Report RC-6193, *I.B.M. Watson Research Center*, Yorktown Heights, New York (October 1976).

Coppersmith, D. and S. Winograd, "On the Asymptotic Complexity of Matrix Multiplication", *SIAM J. on Computing* 11(3), 472-492 (1982).

Csanky, L., "Fast Parallel Matrix Inversion Algorithms", *SIAM J. on Computing* 5(4), 618-623 (1976).

George, J.A., "Nested Dissection of a Regular Finite Element Mesh", *SIAM J. on Numerical Analysis* 10(2), 345-367 (1983).

Golub, G.H. and C.F. van Loan, *Matrix Computation*, The Johns Hopkins University Press, Baltimore (1983).

Hageman, L. and D. Young, *Applied Iterative Methods*, Academic Press, New York (1981).

Heller, D., "A Determinant Theorem With Applications to Parallel Algorithms", Tech. Report, *Comp. Sci. Dept., Carnegie-Mellon University* (March 1973).

Hotelling, H., "Some New Methods in Matrix Calculation", *Ann. Math. Statist.* 14, 1-34 (1943a).

Hotelling, H., "Further Points on Matrix Calculations and Simultaneous Equations", *Ann. Math. Statist.* 14, 440-441 (1943b).

Householder, A., *The Theory of Matrices in Numerical Analysis*, Blaisdell Publishing Co., New York (1964).

Isaacson, E. and H.B. Keller, *Analysis of Numerical Methods*, Wiley, New York (1966).

Lipton, R., D. Rose, and R.E. Tarjan, "Generalized Nested Dissection", *SIAM J. on Numerical Analysis* 16(2), 346-358 (1979).

Lipton, R.J. and R.E. Tarjan, "A Separator Theorem for Planar Graphs", *SIAM J. on Appl. Math.* 36, 177-189 (1979).

Newman, M., "Matrix Computation", *Survey of Numerical Analysis*, 222-255, (S. Todd, Ed.), McGraw Hill, New York (1982).

Pan, V., *How to Multiply Matrices Faster*, Lecture Notes in Computer Science, 179, Springer-Verlag, Berlin (1984).

Preparata, F.P. and D.V. Sarwate, "An Improved Parallel Processor Bound in Fast Matrix Inversion", *Information Processing Letters* 7(3), 148-149.

Schultz, G., "Iterative Berechnung der Reziproken Matrix", *Z. Angew. Math. Mech.* 13, 57-59 (1933).

Strassen, V., "Vermeidung von Divisionen", *J. Reine Angew. Math.* 164, 184-202 (1973).

Valiant, L., S. Skyum, S. Berkowitz, and C. Rackoff, "Fast Parallel Computation of Polynomials Using Few Processors", *SIAM J. on Computing* 12(4), 641-644 (1983).

Varga, R.S., *Matrix Iterative Analysis*, Prentice-Hall (1962).

Wilkinson, J.H., "Error Analysis of Direct Methods of Matrix Inversion", *J.AMC* 8, 281-330 (1961).

Wilkinson, J.H., *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford (1965).

Young, D.M., *Iterative Solution of Large Linear Systems*, Academic Press, New York (1971).

Appendix A. Asymptotic Parallel Complexity of Matrix Multiplication

Theorem A.1. *Suppose that the product of two* $N \times N$ *matrices can be computed in* $O(N^\omega)$ *arithmetic operations for* $\omega > 2$. *Then such a product can be computed in parallel time* $O(\log N)$ *involving at most* N^ω *processors.*

Proof. We recall the well known class of bilinear algorithms of rank M for the evaluation of an $n \times n$ matrix product $XY = (\Sigma_j x_{ij} y_{jk})$. Such algorithms first compute the 2M linear functions

$$L_q = \sum_{i,j} f(i,j,q)x_{ij},\ L_q^* = \sum_{j,k} f^*(j,k,q)y_{jk},\ q=1,\ldots,M, \tag{A.1}$$

then the M products $L_qL_q^*$, and finally $\Sigma_j x_{ij} y_{jk}$ as the linear functions in those products,

$$\sum_j x_{ij} y_{jk} = \sum_{q=0}^{M-1} f^{**}(k,i,q)\, L_qL_q^*. \tag{A.2}$$

Here $f(i,j,q)$, $f^*(j,k,q)$ and $f^{**}(k,i,q)$ are constants. Such a bilinear algorithm can be applied recursively, substituting $n \times n$ matrices for the variables x_{ij} and y_{jk} in (A.1), (A.2) and applying the same algorithm in order to multiply L_q and L_q^*, which become $n \times n$ matrices in that case. (Such recursive bilinear algorithms compute $n^h \times n^h$ matrix product involving $O(M^h)$ arithmetic operations.) It is well known, see [Strassen, 73], [Coppersmith and Winograd, 82], that for large n there exist the above recursive bilinear algorithms of rank $M \leq n^\omega$ that evaluate an $n \times n$ matrix product, provided that the assumption of Theorem A.1 holds.

Let us estimate the parallel complexity of such recursive bilinear algorithms for $n^h \times n^h$ matrix multiplication, that is, the time T(h) and the number of processors P(h) involved in such algorithms.

(A.1),(A.2) immediately imply that simultaneously

$$T(h) \leq T(h-1) + 2\log n + \log M + 5,$$
$$P(h) \leq \max\{2n^{2h}, Mn^{2h-2}, P(h-1)M\}.$$

Choosing $h \to \infty$ we immediately deduce the desired estimates of Theorem A.1 in the case where N is a power of n. The extension to arbitrary $N \times N$ matrices is also immediate via embedding those matrices into $n^h \times n^h$ matrices banded with zeros for $h = \lceil \log N / \log n \rceil$.

Appendix B. Approximate Inverses in Some Special Cases.

In this section we will improve the choice (2.6) of an approximate inverse of A for certain classes of matrices A.

Modification B.1, compare [Isaacson and Keller, 66], p. 84. Let $\lambda_1, \lambda_2, \ldots, \lambda_n$ be the eigenvalues of $A^H A$ such that

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n > 0.$$

(As follows from Lemmas 3.3 and 3.4,

$$\lambda_1 = ||A||^2, \ \lambda_n = 1/||A^{-1}||^2.) \tag{B.1}$$

If for a given matrix A we may precompute λ_1 and λ_n, then we may improve the choice (2.6) for the approximate inverse B of A by setting

$$B = t^* A^H, \ t^* = 2/(\lambda_1 + \lambda_n). \tag{B.2}$$

(B.1) and (B.2) imply that $||R(B)|| = (\lambda_1 - \lambda_n)/(\lambda_1 + \lambda_n)$, and this leads to the following equations, which improve over the bounds of Lemma 2.4,

$$q = ||R(B)|| = \frac{(\text{cond } A)^2 - 1}{(\text{cond } A)^2 + 1} = 1 - \frac{2}{(\text{cond } A)^2 + 1}. \tag{B.3}$$

(Indeed, $||R(B)|| = \rho(R(B))$, see Lemma 3.1, and the eigenvalues of R(B) are just $\mu_i = 1 - t^* \lambda_i$, $i = 1, \ldots, n$, see the proof of Lemma 3.5, so that $1 - t^* \lambda_1 \leq \mu_i \leq 1 - t^* \lambda_n$, $i = 1, \ldots, n$. It remains to recall (B.1),(B.2) in order to arrive at (B.3).)

The improvement over the estimate of Lemma 2.4 is limited, however, that is, $1 - ||R(B)||$ increases less than 2n times even if we may precompute λ_1 and λ_n satisfying (B.1) and then replace (2.6) by (B.2).

Modification B.2. If the matrix A is Hermitian positive definite, (see

Definition 3.1), which is frequent in practical computations, see [Golub and van Loan, 83], [Hageman and Young, 81], [Young, 71], then we may choose

$$B = t' \, I, \, t' \; = 1/\max_j \sum_i \, | \, a_{ij} \, | \; \leq 1/||A|| \leq 1/\max_{i,j} | \, a_{ij} \, | \, .$$

Extending the proof of Lemma 2.4, we derive in this case that

$$||R(B)|| \leq 1 - t' \; /||A^{-1}|| \leq 1 - 1/(n^{1/2} \; \text{cond A}). \tag{B.4}$$

Remark B.1. Actually we may reduce the inversion of an arbitrary non-singular matrix A to the inversion of the Hermitian positive definite matrix $A^H A$ since $A^{-1} = (A^H A)^{-1} A^H$. Then we will arrive at the bound (B.4) (where $A^H A$ replaces A) if we compute the 1-norm $||A^H A||_1$ of the matrix $A^H A$ and choose $B = I/||A^H A||_1$, see Definition 2.3. The resulting estimate (B.4) will be close to (2.8) and (2.10).

Modifications B.1 and B.2 can be combined together if A is a Hermitian positive definite matrix and if $||A||$ and $1/||A^{-1}||$ can be precomputed, in which case setting $B = 2I/(||A|| + 1/||A^{-1}||)$ we would yield $||R(B)|| = 1 - \dfrac{2}{\text{cond A}+1}$.

Modification B.3. Case 1. Let $A = (a_{ij})$ be *strongly diagonally dominant,* that is, let for a constant c

$$(2-1/n^c) \, | \, a_{ii} \, | \; > \sum_j \, | \, a_{ij} \, | \;\; \text{for all i} \tag{B.5}$$

or

$$(2-1/n^c) \, | \, a_{jj} \, | \; > \sum_i \, | \, a_{ij} \, | \;\; \text{for all j.} \tag{B.6}$$

The class of diagonally dominant matrices is very important in applications, see [Varga, 62] and [Hageman and Young, 81]. (B.5) implies that A is *strongly row-diagonally dominant;* (B.6) implies that A is *strongly column-diagonally dominant.* In both cases we choose

$$B = \text{diag}\{1/a_{11}, 1/a_{22}, \cdots, 1/a_{nn}\}. \tag{B.7}$$

Lemma B.1, compare Definition 2.3. *(B.5) and (B.7) imply that*

$$||R(B)||_\infty \leq 1 - 1/n^c. \tag{B.8}$$

(B.6) and (B.7) imply that

$$||R(B)||_1 \leq 1 - 1/n^c. \tag{B.9}$$

Proof. For all i the row-vector i of the matrix R(B)=I-BA has the 1-norm equal to $\sum_{j \neq i} |a_{ij}| / |a_{ii}|$ which is not greater than $1-1/n^c$ if (B.5) holds. This immediately implies (B.8). (B.6) and (B.9) turn into (B.5) and (B.8) if A replaces A^H.

Thus in the cases where A is strongly diagonally dominant we choose B defined by (B.7) and again arrive at the bounds of Corollary 2.1 provided that one of the two norm of Definition 2.3 substitutes for the 2-norm. The computation of B by (B.7) involves only n arithmetic operations that can be performed using 1 parallel step and n processors.

Modification B.3. Case 2. Let $A = (a_{ij})$ be a triangular matrix. Then define B by (B.7) and apply the algorithm of Section 2 for computing B_k^* for $k = \lceil \log n \rceil$. Although B does not satisfy (2.2) in this case, the method works simply because R(B) is an $n \times n$ triangular matrix whose diagonal is filled with zeros and therefore $R(B)^i = 0$ if $i \geq n$. Thus $A^{-1} = B_k^* = \sum_{i=0}^{n-1} (R(B))^i B$, so that the convergence of the algorithm to A^{-1} in $\lceil \log n \rceil$ steps immediately follows for any triangular nonsingular matrix A even if cond A is exponentially large, compare [Csanky, 76] and [Heller, 73]. (Note that the inversion of a matrix A can be reduced to the inversion of triangular marices if the QR-factors of A or LUP-factors of A are available.)

Appendix C. Refinement of the Approximate Inverse of a Matrix by Newton's Iteration and by the Residual Correction Method.

Next we will consider two alternatives to the first iterative algorithm for INVERT described in Section 2. As in the case of the first algorithm, we will assume that an approximate inverse B of A has been precomputed such that (2.2) holds.

The second algorithm performs Newton's iteration for the matrix equation $R(X) = 0$ by successively computing

$$\tilde{B}_h = (2I - \tilde{B}_{h-1}A)\tilde{B}_{h-1} = (I + R(\tilde{B}_{h-1}))\tilde{B}_{h-1} \qquad (C.1)$$

for $h=1,2,...$ and letting $\tilde{B}_0 = B$. It is readily verified that $R(\tilde{B}_h) = (R(\tilde{B}_{h-1}))^2$ for $h=1,2,...$, so that $I - \tilde{B}_kA = R(\tilde{B}_k) = (R(\tilde{B}_0))^{2^k}$ for all k. We postmultiply the latter equations by A^{-1}, note that (2.2) and Lemma 2.1 imply that

$$||A^{-1}|| \leq ||B||/(1-q), \qquad (C.2)$$

and arrive at the bound

$$||A^{-1} - \tilde{B}_k|| \leq q^{2^k}||B||/(1-q), \qquad (C.3)$$

compare (2.3).

The third algorithm relies on the well known residual correction method, see [Atkinson, 78], p. 469. We successively compute

$$X_{h+1,g} = X_{h,g} + B_g(I - AX_{h,g}),\ X_{0,g} = B_g \text{ for } h=0,1,...,s-1,s\geq 2 \qquad (C.4)$$

where $g=0$ and $B_0 = B$ satisfies (2.2). Then we recursively perform the iteration sweep (C.4) for $g=1,2,...$, choosing $B_g = X_{s,g}$. Within each iteration sweep (C.4) $I - X_{h+1,g}A = (I - X_{h,g}A)(I - B_gA)$, $h=0,1,...,s-1$, see [Atkinson, 78], so that $||I - B_{g+1}A|| \leq ||I - B_gA||^s$. Therefore after k iteration sweeps (C.4) we arrive at an approximation B_k to A^{-1} such that $||I - B_kA|| \leq q^{s^k}$. We postmultiply $I - B_kA$ by A^{-1}, recall (C.2) and deduce that

$$||A^{-1} - B_k|| \leq q^{s^k}||B||/(1-q). \qquad (C.5)$$

For $s=2$, (C.5) is similar to (2.3) and (C.3).

Unlike the algorithms of Section 2, both algorithms of this appendix are

self-correcting; the errors of computing $\tilde{B}_k$ and X_{hg} for any k,h,g do not propagate, that is, they are automatically corrected at the next iterations provided, of course, that the iterations are not too much contaminated by the errors (it is necessary that in spite of the errors the computed matrices $\tilde{B}_k$ and X_{hg} remain to be approximate inverses of A), see the next appendix.

Appendix D. Some Error Estimates.

Let us assume that the computation of B by (2.6) and the subsequent Newton's iteration (C.1) have been performed with finite precision chopping to d binary digits and let us estimate the total round-off errors of the computation. Let $\Delta(u)$ and $\delta(u) = \frac{||\Delta(u)||}{||u||}$ designate the absolute and the relative errors of computing u where u can be a matrix or a scalar (in the latter case $||u|| = |u|$, $||\Delta u|| = |\Delta u|$). As this is customary, we will assume that $d \to \infty$ and *will ignore the values of smaller orders of magnitudes.*

We immediately derive from (2.6) that $\delta(1/t) \le 2^{1-d}$, $\delta(B) \le 3 \cdot 2^{-d}$.

Since $||B|| \le 1/||A||$, see (2.8), it follows that

$$||I-(B + \Delta B)A|| \le ||I-BA|| + 3 \cdot 2^{-d} \le 1+3 \cdot 2^{-d}-1/(n(\text{cond } A)^2).$$

Therefore it is sufficient to choose $d = O(\log n)$ in order to assure that, say

$$||I-(B+ \Delta B)A|| < 1-0.6/n^c \qquad (D.1)$$

provided that $||I-BA|| < 1-1/n^c$. Therefore if we compute B by (2.6), it is sufficient to choose $d = O(\log n)$ in order to assure that the computed approximation matrix $B + \Delta B$ will remain practically as good as the approximate inverse B, that is, the total number of iterations sufficient for computing A^{-1} with required precision may increase at most by 1 if $B + \Delta B$ substitutes for B. (Note that the first iteration squares the residual norm and turn $1-0.6/n^c$ into $1-1.2/n^c+0.36/n^{2c}$, which is less than $1-1/n^c$ for large n, compare (D.1).)

Next let us assume that $\tilde{B}_{h-1}$ and A are given and that $\tilde{B}_h$ has been computed by (C.1) using d binary digits. If the straightforward algorithm for matrix multiplication is used, then the error bounds $||\Delta(\tilde{B}_{h-1}A)||$ and $||\Delta(\tilde{B}_h)||$ can be immediately estimated applying the backward error analysis, see [Wilkinson, 65]. The error analysis is not too hard even if the matrix multiplications in (C.1) are performed by asymptotically fast algorithms. In [Pan, 84], pp. 117-127, the error analysis is presented for such fast sequential algorithms; the analysis does not change in the case of parallel computation. In particular, Corollary 23.4, p. 121, and Theorem 24.2, p. 127, of [Pan, 84] imply that

$$||\Delta(\tilde{B}_{h-1}A)|| \leq 2^{-d}n^{O(1)}||\tilde{B}_{h-1}|| * ||A||. \tag{D.2}$$

Let $||I-\tilde{B}_0A|| = q$, so that $||I-\tilde{B}_gA|| \leq q^{2^g}$ for all g. Then (C.1) implies that

$$||\tilde{B}_h|| \leq (1+||I-\tilde{B}_{h-1}A||)||\tilde{B}_{h-1}|| \leq (1+q^{2^g})||\tilde{B}_{h-1}|| \leq (1+q+q^2+\cdots+q^h)||\tilde{B}_0||,$$

compare Lemma 2.2. Therefore

$$||\tilde{B}_h|| * ||A|| \leq (1/(1-q))||\tilde{B}_0|| * ||A|| \leq 1/(1-q),$$

since $B = \tilde{B}_0$ is defined by (2.6), compare (2.8). Substitute this into (D.2) and deduce that

$$||\Delta(\tilde{B}_{h-1}A)|| \leq 2^{-d}n^{O(1)}/(1-q) \leq 2^{-d}n^{O(1)}$$

if $1-q \geq 1/n^{O(1)}$.

Similarly we extend this bound to the following estimate.

$$||\Delta(\tilde{B}_h)|| \leq 2^{-d}n^{O(1)}||I+R(\tilde{B}_{h-1})|| * ||\tilde{B}_{h-1}|| \leq 2^{-d}n^{O(1)}/||A||$$

so that

$$||I-(\tilde{B}_h + \Delta(\tilde{B}_h))A|| \leq ||I-\tilde{B}_hA|| + 2^{-d}n^{O(1)}.$$

If, say $||I-\tilde{B}_hA|| \leq 2^{-n^c-1}$, then it is sufficient to choose $d=n^c+O(\log n)$ in order to assure that $||I-(\tilde{B}_h + \Delta(\tilde{B}_h))A|| \leq 2^{-n^c}$. Then again $\tilde{B}_h + \Delta(\tilde{B}_h)$ serves practically as well as $\tilde{B}_h$ in the subsequent iterations (C.1), (that is, 1-2 extra iterations shall guarantee at least the original precision of the output if $\tilde{B}_h + \Delta(\tilde{B}_h)$ replaces

$\tilde{B}_h$ in (C.1)).

Similar error estimates can be obtained if the third algorithm (using the residual corrections (C.4)) is applied.

END

FILMED

5-85

DTIC