TECHNICAL REPORT NO. 328

Undulant-Block Pivoting and $(L + U)$, D' Decomposition

by

David S. Wise

Revised: September 1991

COMPUTER SCIENCE DEPARTMENT
INDIANA UNIVERSITY
Bloomington, Indiana 47405-4101

Undulant-Block Pivoting and $(L + U),\ D'$ Decomposition

David S. Wise[1]
Computer Science Department
Indiana University
Bloomington, Indiana 47405–4101
dswise@iuvax.cs.indiana.edu

revised: September, 1991

[1]Research reported herein was sponsored, in part, by the National Science Foundation under Grant Number DCR 90-027092.

Abstract

A new formulation for LU decomposition allows efficient representation of intermediate matrices when pivoting on blocks of various sizes, *i.e.* during "undulant-block" pivoting. Its efficiency arises from block encapsulization that is implicit in the data structure or that is used by the process scheduler, and defers row/column permutations that destroy such encapsulizations. It is useful, therefore, for parallel or distributed processing on matrix representations that encapsulate submatrices as substructures.

A given matrix, A is decomposed into two matrices, plus two permutations. The permutations, P and Q, are the row/column rearrangements usual to full pivoting. The principal results are $(L+U)$ and $(QDP)^{-1}$, where L and U are proper lower—and respectively upper—triangular, D is quasi-diagonal following the zero blocks in $L+U$, and $PAQ = (I+L)D(U+I)$.

An example of a motivating data structure, the quadtree representation for matrices, is reviewed. Candidate pivots for Gaussian elimination under that structure are subtrees, both constraining and assisting the pivot search; block operations there decompose accordingly. Finally, an integer-preserving version is presented.

1991 Mathematics Subject Classification: 65F05 primary; 68Q22, 65F50 secondary.
CR categories and Subject Descriptors:
G.1.3 [**Numerical Linear Algebra**]: Linear systems, Matrix inversion, Sparse and very large systems; C.1.2 [**Multiple Data Stream Architectures (Multiprocessors)**]: Parallel processors. E.1 [**Data Structures**]: Trees; D.1.1 [**Applicative (Functional) Programming Techniques**]; I.1.2 [**Algebraic Manipulation**]: Algebraic algorithms.
General Term: Algorithms.
Additional Key Words and Phrases: Gaussian elimination, block algorithm, LU factorization, quadtree matrices, quaternary trees, exact arithmetic.

0 Introduction

In the context of matrix computations, *block algorithms* are defined in terms of operations on submatrices instead of on elements. This paper relaxes the constraint of uniform size, so that blocks of various sizes, including elements as 1×1 submatrices, can be operands. Such a block algorithm is *undulant* when it can alter the sizes of its submatrix operands during a run.

Block algorithms have attracted much attention lately because they are well suited to high-performance computing. Non-trivial blocks use hierarchical [8, 7] and distributed memory more effectively than row-based methods, and manipulations on blocks raise the granularity of parallel processing, ameliorating the problem of process scheduling. Moreover, these algorithms offer better locality on machines where processor-memory access time varies according to relative addresses of processor and memory. The efficiency of parallel processing improves considerably as more independent, large-submatrix operations are scheduled onto processors closer to their respective operands. This desirability of block operations is well known, with matrix-matrix operations often called Level 3 (BLAS) operations, where matrix-vector operations are ranked at Level 2.

Staticly sized blocks are used by block algorithms for Gaussian elimination (GE) under one of two philosophies. They can be used without constraint in cases when it is safe not to pivot, for example when the problem is diagonally dominant or symmetric positive definite. Alternatively, they can be used heuristically for pivoting, against the rare possibility of collapse, because a nonsingular residual matrix has no nonsingular and stable-pivot block of the required size and orientation. Upon such a failure, the partial solution is abandoned and the problem is reordered and solved again, even though reversion to scalar pivoting—the simplest of undulant-block strategies—could complete it. Because such failures are less likely with smaller blocks, the heuristic approach constrains the need of high-performance for larger operands.

George [9] observes that block pivoting offers a middle ground between the complicated programming necessary for sparse matrix techniques, and the fill-in that results from straightforward code using a band or band-like ordering. Undulant block-sizing is, therefore, developed here in an effort also to marry his unified approach to sparse/banded problems to the increased throughput of parallelism, even though the steps just after pivoting on a large block may be forced onto smaller ones. Nevertheless, a larger pivot

may become practical again later.

GE either with partial pivoting (GEPP) or even full pivoting (GEFP), remains the matrix algorithm of greatest interest, for both numeric and symbolic problems. Demmel and Higham [5] present new and generally favorable results on numeric stability of block algorithms using BLAS3. They consider error analysis on GE and GEPP, iterative refinement on GEPP, block triangular factorizations on GE, and orthogonal transformations. Of their results, the one closest to this work is on block-triangular factorizations (GEPP on staticly sized blocks), which they show can be unstable.

However, the decomposition algorithm described here differs in two respects: it uses undulant blocking and it accommodates full pivoting. Moreover, it uses efficiently any block-oriented matrix representation. Neither analytical nor experimental results on its stability, fill-in, or speed are included here, but strategies are suggested for improving these attributes by careful selection of pivot blocks.

This algorithm is designed also to accommodate special representations for sparse and symbolic matrices. Both cases suggest that a zero submatrix have a trivial representation, and that row/column permutations would be very expensive; therefore, one concession to full pivoting is to defer permutations. An example of such a representation, as well as the algorithms that use $(L + U), D'$ decomposition to solve linear systems and to invert matrices, is offered in Section 3. Section 4 extends it to division-free algorithms necessary for exact and symbolic arithmetic.

This formulation of classic LU decomposition helps computations subsequent to decomposition, encapsulating those intermediate results according to whatever block structure is provided by the matrix representation. Block decomposition of the results here retains the relative "geography" in the parameters; thus, block pivots in the result D' are represented just as conveniently as they were in input A.

The remainder of this paper is in five parts. The first presents a generalization of LU decomposition: $(L + U), D'$ decomposition; and dependent algorithms for solving linear systems and matrix inversion that are intended to use undulant blocking. The second section presents the quadtree representation for numeric and permutation matrices. The third shows how such a representation, which constrains its candidate-pivot blocks, needs this decomposition. Section 4 shows how to adapt it for exact arithmetic. Finally, the last section offers conclusions.

1 $(L+U), D'$ decomposition

Definition. *A matrix A is* proper *lower (upper) triangular if $a_{ij} = 0$ for $i \leq j$ (respectively, $i \geq j$).*

Notation. *I denotes the* identity *matrix of any order. Similarly, 0 denotes the* zero *matrix of any order.*

Definition. *Two matrices, A and B, are said to be* disjoint *if*

$$\forall i, j (a_{ij} = 0 \vee b_{ij} = 0).$$

Definition. *A square matrix A is* quasi-diagonal *[6] if it has square submatrices (cells) along its main diagonal with its remaining elements equal to zero.*

This more obscure term is used instead of *block-diagonal* to emphasize that the blocks along the diagonal can differ in size, as Faddeeva illustrates. "The determinant of a quasi-diagonal matrix is equal to the product of the determinants of the diagonal cells, on the strength of a notable theorem by Laplace [6]."

If D is an $n \times n$ quasi-diagonal matrix with b nonzero blocks then, following the block decomposition, one can partition the basis of the underlying vector space, decomposing it into b mutually complementary subspaces. Thus, problems on D can be decomposed into b small, independent problems: one in each subspace.

The proper lower triangular matrix, L, associated below with a quasi-diagonal matrix, D, will have zero submatrices exactly where D has nonzero matrices. Therefore, the above decomposition on the vector space underlying D can be applied to $I+L$, as well. The same can be said for associated upper triangular matrices, U.

Definition. *An $(L+U), D'$* decomposition *of A, nonsingular, is the quadruple, $\langle P, Q, L+U, (QDP)^{-1} \rangle$ where*

- *P and Q are permutations;*
- *L is proper lower triangular and U is proper upper triangular;*
- *D is quasi-diagonal and disjoint from both L and U;*

- $PAQ = (I + L)D(U + I)$.

It is trivial to separate $(L + U)$ into unit upper and unit lower triangular matrices, $I + L$ and $U + I$, but this is never necessary.

Notation. *Subscripts* n, m, s, *should be read as "north, middle, south;" and* w, c, e *as "west, central, east."*[1]

Algorithm 1 (Pivoting nonsingular A to $\langle P, Q, A', D' \rangle$.) Full, undulant-block pivoting is assumed, although no strategy for selecting pivot blocks is yet addressed. The quadruple of results from repeated pivoting on a matrix A is described recursively as follows.[2] If A is void then the result is $\langle I, I, 0, 0 \rangle$.

Otherwise, let the block decomposition of A, isolating the $k \times k$ pivot block, A_{mc}, be labeled

$$A = \begin{array}{c} \\ i \\ k \\ n-k-i \end{array} \begin{array}{c} \begin{array}{ccc} j & k & n-k-j \end{array} \\ \begin{pmatrix} A_{nw} & A_{nc} & A_{ne} \\ A_{mw} & A_{mc} & A_{me} \\ A_{sw} & A_{sc} & A_{se} \end{pmatrix} \end{array}$$

where A is $n \times n$ and A_{ne} is $i \times j$. The trivial case has $i = 0 = j$ and $k = n$. Then

$$\langle P, Q, A', D' \rangle = \left\langle \begin{pmatrix} 0 & I & 0 \\ P_w & 0 & P_e \end{pmatrix}, \begin{pmatrix} 0 & Q_n \\ I & 0 \\ 0 & Q_s \end{pmatrix}, \begin{pmatrix} A'_{nw} & A'_{nc} & A'_{ne} \\ A'_{mw} & 0 & A'_{me} \\ A'_{sw} & A'_{sc} & A'_{se} \end{pmatrix}, \begin{pmatrix} D'_{nw} & 0 & D'_{ne} \\ 0 & D'_{mc} & 0 \\ D'_{sw} & 0 & D'_{se} \end{pmatrix} \right\rangle$$

where P_w is $(n-k) \times i$, P_e is $(n-k) \times (n-k-i)$, Q_n is $j \times (n-k)$, and Q_s is $(n-k-j) \times (n-k)$. These values can be computed as follows: first

$$D'_{mc} = A_{mc}^{-1}$$

is solved recursively (with good accuracy); then the pivot row and pivot column are completed using Level 3 operations,

$$\begin{pmatrix} A'_{mw} & A'_{me} \end{pmatrix} = D'_{mc} \begin{pmatrix} A_{mw} & A_{me} \end{pmatrix}$$

[1] Using these enumerated types, instead of integers, for block indexing dodges debates about zero-based vs. one-based indexing, obscures ordering among blocks, and avoids any implication that, say, 5 could be a useful block index.

[2] Versions of these algorithms for quadtrees (cf. Section 2) have been programmed in C and in Haskell, the international functional programming language [11], but code would add little to them and Figure 2.

$$\begin{pmatrix} A'_{nc} \\ A'_{sc} \end{pmatrix} = \begin{pmatrix} A_{nc} \\ A_{sc} \end{pmatrix} D'_{mc};$$

finally,

$$\left\langle \begin{pmatrix} P_w & P_e \end{pmatrix}, \begin{pmatrix} Q_n \\ Q_s \end{pmatrix}, \begin{pmatrix} A'_{nw} & A'_{ne} \\ A'_{sw} & A'_{se} \end{pmatrix}, \begin{pmatrix} D'_{nw} & D'_{ne} \\ D'_{sw} & D'_{se} \end{pmatrix} \right\rangle$$

results from recursive pivoting on the $(n-k) \times (n-k)$ problem

$$\begin{pmatrix} A_{nw} & A_{ne} \\ A_{sw} & A_{se} \end{pmatrix} - \begin{pmatrix} A_{nc} \\ A_{sc} \end{pmatrix} \begin{pmatrix} A'_{mw} & A'_{me} \end{pmatrix},$$

that can also be derived using Level 3 operations. □

The subtrahend in the last expression can alternatively be computed as

$$\begin{pmatrix} A'_{nc} \\ A'_{sc} \end{pmatrix} \begin{pmatrix} A_{mw} & A_{me} \end{pmatrix}.$$

A permutation, P or Q, can be efficiently represented in structures other than the matrix used here, *e.g.* a list of indices for its I entries. The next section defines a general matrix representation that is especially cheap for permutation matrices [17], as well as a scheme for indexing of both trivial and non-trivial I submatrices.

In the special case that $i = 0 = j$ at every level in the recurrence, the algorithm might be called "undulant-block Gaussian elimination." Figure 1 illustrates Algorithm 1 in this case. In this case it is easy to see that $P = I$, $Q = I$, $A' = L + U$, $D' = D^{-1}$, and that these form the $(L+U), D'$ decomposition of A, by conventional GE.

However, a permutation of the same example in Figure 2 can retain the same pivots in the same order, (R, S, T), to yield the pivoting shown under GEFP. The (now nontrivial) permutations, P and Q, are shown separately in Figure 3.

Theorem 1 . *Let $\langle P, Q, A', D' \rangle$ be the result from Algorithm 1 on nonsingular input A. Then the $(L+U), D'$ decomposition of A is $\langle P, Q, PA'Q, D' \rangle$.*

Proof: If A is nonsingular then the recursive pivoting is defined and P and Q are permutation matrices, by construction. If an oracle had provided P and Q before Algorithm 1 began, then we could have permuted A, for instance

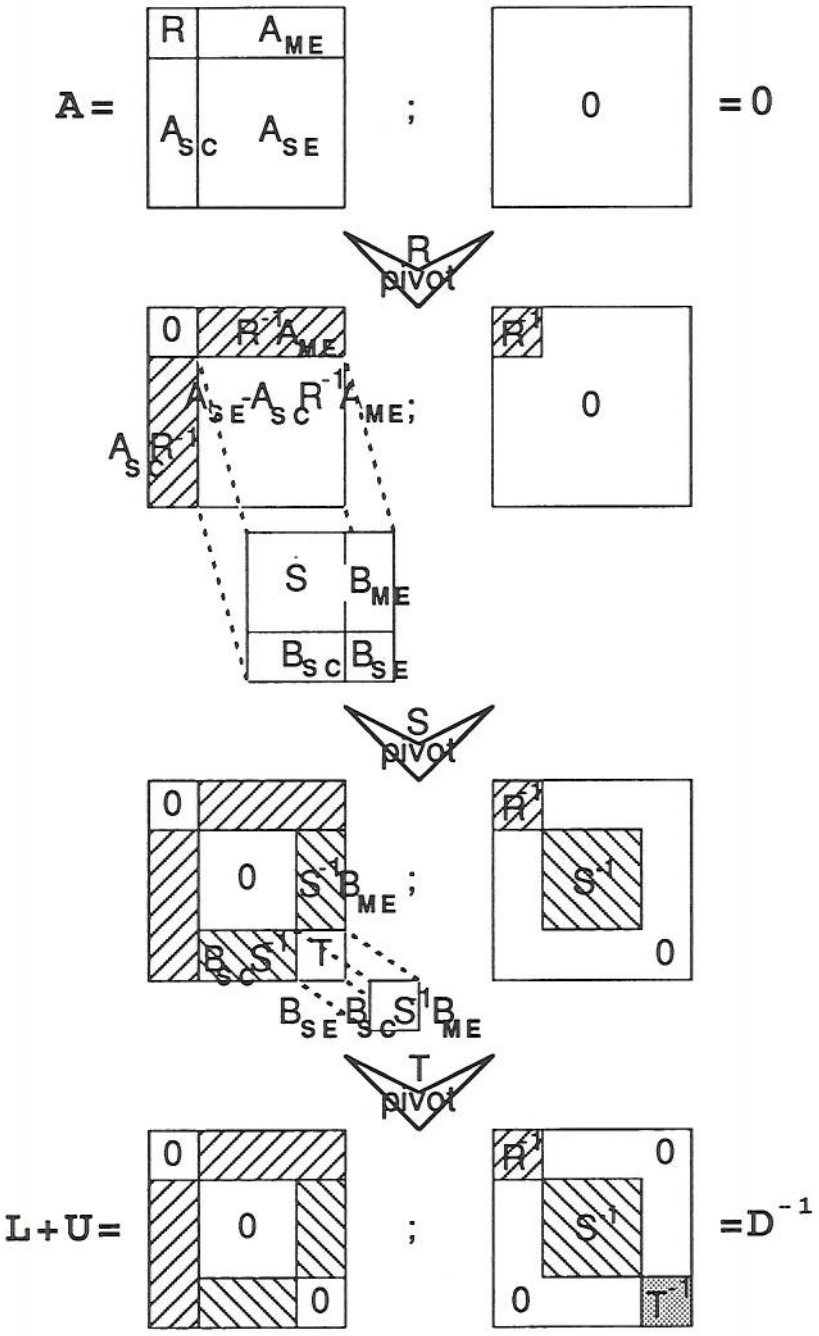

Figure 1: Undulant-block elimination.

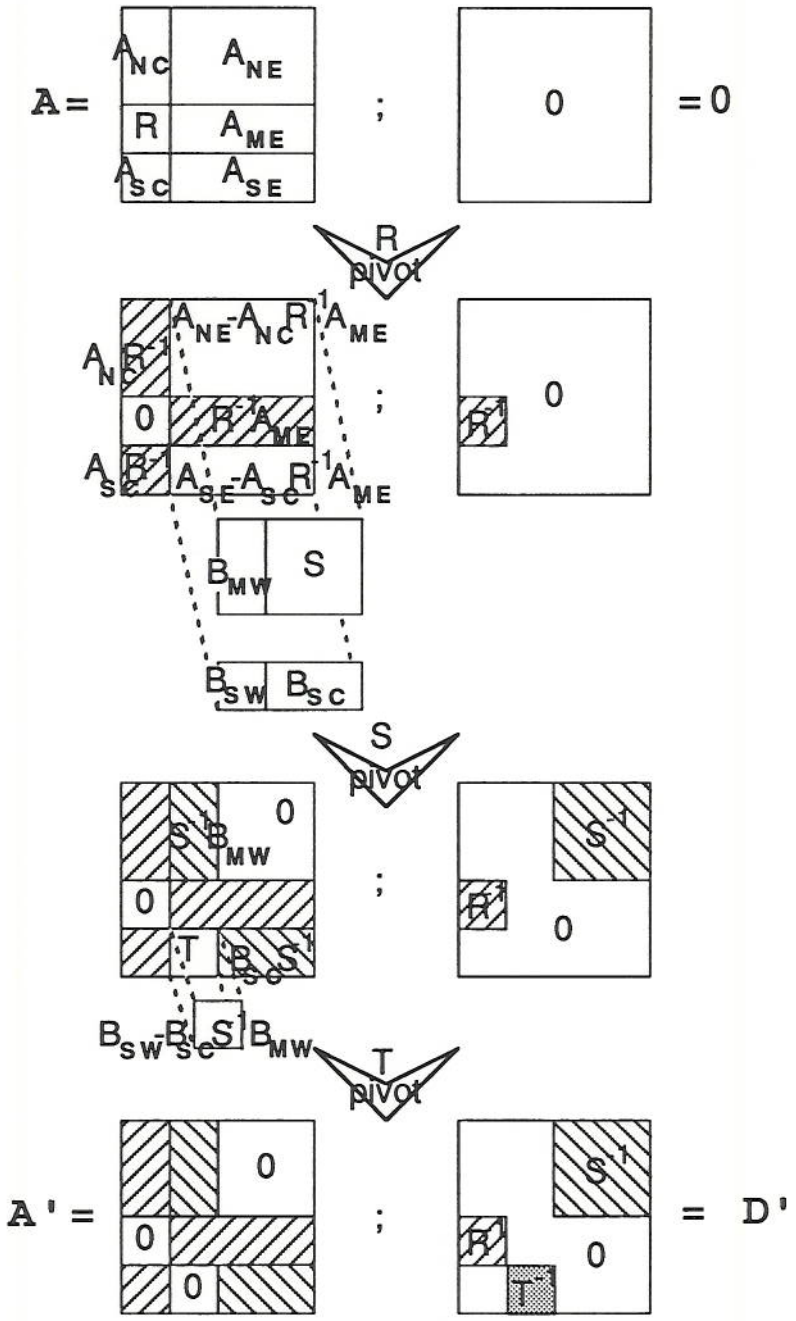

Figure 2: Full, undulant-block pivoting like Figure 1's.

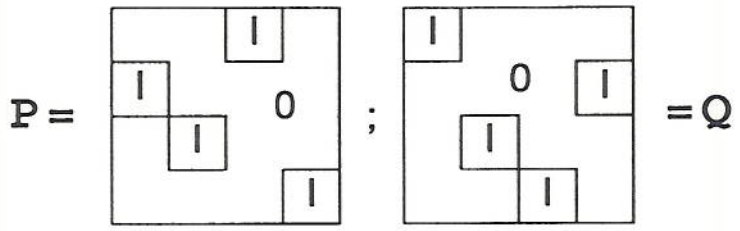

Figure 3: Permutation matrices from pivoting in Figure 2.

from Figure 2, to PAQ, and the same pivoting on it would have followed the main diagonal, as in Figure 1, to yield $(L+U)$ and D^{-1}. Thus, L (U) is proper lower (respectively, upper) triangular; D is quasi-diagonal, disjoint from both L and U; and

$$PAQ = (I+L)D(U+I).$$

However, because no permutations occur in Algorithm 1, permuting A before it is equivalent to permuting both its A' and D' results afterwards

$$\begin{aligned} PA'Q &= L+U; \\ PD'Q &= D^{-1}. \end{aligned}$$

Therefore, $D' = (QDP)^{-1}$. □

Corollary 1 . *D' in Algorithm 1 is a rearrangement of a quasi-diagonal matrix to fit the structure of the pivots, as selected.*

Algorithm 2 ($(L+U), D'$ decomposition.) Use Algorithm 1 to compute $\langle P, Q, A', D' \rangle$ from A, and apply the permutations *once* to return $\langle P, Q, PA'Q, D' \rangle$. □

There are three reasons that the block pivoting illustrated in Figure 2 is favored over traditional pivoting strategies (*e.g.* GEPP). First, the block structure of A' and D' in Figure 2 exactly follows the geography of the chosen pivot blocks, to save intermediate storage under *any* matrix representation that uses block decomposition. Secondly, the permutations in Algorithm 2

reorder *both* L and U. Since their sum is disjoint, it suffices to permute A' just once; only two other permutations are required in Algorithm 3 or 4. Last, it demonstrates full pivoting, desirable to improve the chances for finding a large pivot among candidates of various sizes.

Section 3 presents a block-oriented data structure that compactly represents the decomposition of D' in Figure 2, but not of D^{-1} in Figure 1. Moreover, it can easily represent and manipulate $L + U$, computed from Figure 2 by Algorithm 2 to appear exactly as in Figure 1, because the only special blocks there are zeroes, and so represented especially compactly. The following two algorithms are particularly fast when large diagonal blocks of $L + U$ are zero, and when the associated nonzero blocks of D' are packed.

Algorithm 3 (Solving a linear system.) Solve $A\vec{x} = \vec{b}$ using this reformulation:

$$PAQ = (I + L)D(U + I)$$

implies

$$P^{-1}(I + L)Q^{-1}(QDP)P^{-1}(U + I)Q^{-1}\vec{x} = \vec{b}$$

1. Compute the $(L + U), D'$ decomposition of A using Algorithm 2.
2. [forward substitution] Solve $(I + L)\vec{y} = P\vec{b} = \vec{c}$.
   - If $L + U = 0$ then $\vec{y} = \vec{c}$.
   - Otherwise, partition[3]

$$L + U = \begin{pmatrix} L_{nw} + U_{nw} & E \\ W & L_{se} + U_{se} \end{pmatrix}; \quad \vec{y} = \begin{pmatrix} \vec{y_n} \\ \vec{y_s} \end{pmatrix}; \quad \vec{c} = \begin{pmatrix} \vec{c_n} \\ \vec{c_s} \end{pmatrix}.$$

   - Recursively solve $(I + L_{nw})\vec{y_n} = \vec{c_n}$.
   - Recursively solve $(I + L_{se})\vec{y_s} = \vec{c_s} - W\vec{y_n}$.
3. [backward substitution] Similarly, solve $(U + I)\vec{z} = P(D'(Q\vec{y}))$.
4. Permute $\vec{x} = Q\vec{z}$. □

[3] E and W should be chosen for compact representation under the matrix representation; secondarily, one might choose E and W to be of about equal size, or to force either $L_{nw} + U_{nw}$ or $L_{se} + U_{se}$ to be entirely zero.

Algorithm 4 (Matrix inversion.) Invert

$$A = P^{-1}(I + L)[Q^{-1}(QDP)P^{-1}](U + I)Q^{-1}.$$

1. Compute the $(L + U), D'$ decomposition of A using Algorithm 2.
2. If $L + U = 0$ then $A^{-1} = (QP)D'(QP)$;
3. Otherwise compute $L' = (I + L)^{-1}$ and $U' = (U + I)^{-1}$ recursively; then $A^{-1} = Q(U'(PD'Q)L')P$.

   - Partition $L + U$ as in the previous algorithm.
   - Recursively compute

$$L'_{nw} = (I + L_{nw})^{-1}; L'_{se} = (I + L_{se})^{-1};$$
$$U'_{nw} = (U_{nw} + I)^{-1}; U'_{se} = (U_{se} + I)^{-1}.$$

   - Then

$$L' = \begin{pmatrix} L'_{nw} & 0 \\ -L'_{se}WL'_{nw} & L'_{se} \end{pmatrix}; \quad U' = \begin{pmatrix} U'_{nw} & -U'_{nw}EU'_{se} \\ 0 & U'_{se} \end{pmatrix}.$$

□

A single recurrence can yield $(L' - I) + (U' - I)$ directly from $L + U$ using fewer processes; the program is left as an exercise for the reader. A second exercise is to modify Algorithm 2 to return D^{-1} in place of D', specifically to accelerate Step 3 of Algorithm 4; does Step 2 improve similarly?

A design criterion of these algorithms—to avoid permutations on intermediate matrices—has been met. Algorithm 1 uses none none; Algorithms 2, 3, and 4 require minimal permutations. Moreover, in all algorithms only two permutations, P and Q, are needed; under partial pivoting one of them becomes I. After the serious row/column rearrangement in Algorithm 2, Algorithm 3 permutes only vectors. After the Algorithm 2 rearrangement, Algorithm 4 trivially rearranges a quasi-diagonal matrix and then uses a second serious row/column rearrangement to complete the inverse.

The primary goal of these algorithms, to implement LU decomposition with variously large Level 2 and 3 (block) operations, is achieved in Algorithm 1. Figures 1 and 2 might suggest that the disjoint results, A' and D', be summed to form a compact result. They are held separate, however, because that sum is not easily separated after the rearrangement (because of undulant block sizing); some large Level 2 and 3 (block) operations would also be splintered in Algorithms 3 and 4.

2 Quadtrees and undulant-block pivoting

Algorithm 1 is a generalization of generic LU decomposition. If it is restricted so that k is fixed at all levels of the recursion, then it implements conventional (non-undulant) block pivoting; if $k = 1$ everywhere, it implements scalar pivoting. If it is further restricted with $j = 0$, then $P = I$ and it uses partial pivoting. If, furthermore, $i = 0$ then $Q = I$, pivoting disappears, and Algorithm 1 implements traditional GE.

The generalization was necessary in order to discuss block pivoting with undulant (varying) k. In order to take best advantage of efficient Level 2 and 3 (block) operations, we want k to be large at some steps even though it may be forced smaller at others. However, if the choice of i, j, k is unrestricted then pivot selection can become more difficult than the underlying linear problem. For instance, P in Figure 3 has only one (of nine) contiguous 2×2 block that is nonsingular, and hence a pivot candidate; a search for it that also tested eight other candidates overwhelms the effort actually to eliminate it. That is, complete searches like this may actually be counterproductive, especially in sparse matrices and when undulance permits default to scalar pivots.

This section reviews the quadtree representation of matrices [14], which motivated the generalized Algorithm 1. It decomposes full matrices as balanced quaternary trees. Sparse matrices are accommodated by providing a distinguished null representation for an entirely zero (sub)matrix (*cf.* hypermatrices [4]). Moreover, the pivot search only considers those blocks that coincide with subtrees as pivot candidates..

Postulate. *A d-dimensional* array *is represented as a* 2^d*-ary tree.*

In all of the following, the orders of matrices and sizes of vectors are taken to be powers of two. Arrays of other sizes can be embedded efficiently into these using zero padding, which Algorithm 1 will treat as "eliminated" *ab initio.*

Data Structure [Binary tree representation of vectors]. *A* vector *of size* 2^p*, represented as a binary tree of depth* p*, is*

- *homogeneously zero and represented as* 0*;*
- *represented by the appropriate non-zero scalar when* $p = 0$*;*

- *otherwise represented as a pair of subvectors,* (north, south), *each of which is size* 2^{p-1}, *at least one of which is non-zero.*

Definition. *The* Ahnentafel index *[3] of an entire vector is* 1. *If the Ahnentafel index of a subvector is* i, *then the Ahnentafel index of its north son is* $2i$, *and the Ahnentafel index of its south son is* $2i + 1$.

This is the familiar "level order" indexing of a binary tree, where the 2^i nodes at Level i are indexed left-to-right from 2^i up to $2^{i+1} - 1$. In the following, all indices[4] are Ahnentafel indices.

Data Structure [Quadtree representation of matrices]. *A* matrix *of order* 2^p, *represented as a quaternary tree of depth* p, *is*

- *homogeneously zero and represented as* 0;
- *represented by the appropriate non-zero scalar when* $p = 0$;
- *otherwise represented as a quadruple of submatrices,* (northwest, northeast, southwest, southeast), *each of which is order* 2^{p-1}, *at least one of which is non-zero.*

An alternate definition of this structure allows the elementary non-zero items to be uniformly larger than 1×1. For instance, a system of time-dependent differential equations does well with an elementary block of size 3×3 [4, p. 5]. Memory bandwidth may make it practical to use even larger, sequentially allocated "elementary" blocks.

Definition. *The* index *of an entire matrix is* $\langle 1, 1 \rangle$. *If the index of a submatrix is* $\langle i, j \rangle$, *then the* index *of its northwest quadrant is* $\langle 2i, 2j \rangle$, *of its northeast quadrant is* $\langle 2i, 2j + 1 \rangle$, *of its southwest quadrant is* $\langle 2i + 1, 2j \rangle$, *and of its southeast quadrant is* $\langle 2i + 1, 2j + 1 \rangle$. *The first index of each index pair is the* row index; *the second is the* column index.

[4] In practice, the significant bits in each Ahnentafel index are reversed. That is, the node at Level 0 is indexed 1; the left son of Node i at Level j is indexed $i + 2^j$, and its right son by $i + 2^{j+1}$. The reversal allows a bottom-up index to be build easily, using doubling and addition at each node, and to be discharged simply by descending the binary tree using integer `quotient_remainder` on 2 (*i.e.* `shift` and `even?`) at each level.

Data Structure. *A* permutation matrix *of order* 2^p, *represented as a quaternary tree of depth* p, *is*

- *homogeneously zero and represented as* 0;
- *the identity matrix and represented as* I;
- *otherwise represented as a quadruple of submatrices,* (northwest, northeast, southwest, southeast), *each of which is order* 2^{p-1}.

Moreover, if indices, $\langle i, j\rangle$ *and* $\langle m, n\rangle$, *of two* I *blocks in a permutation matrix satisfy either equation* $i = 2^k m + s$ *or* $j = 2^k n + s$ *for* $0 \leq s < 2^k$, *then* $m = i$ *and* $n = j$.

The last clause establishes the "Eight Rooks Problem" orientation of I blocks in permutation matrices.

Corollary 2 . *No nonterminal node in a quadtree matrix has* 0 *as all four of its quadrants. Similarly, no permutation matrix has both northeast and southwest quadrants* 0 *while both its northwest and southeast are simultaneously either* 0 *or* I.

The indices of the pivot blocks are sufficient to determine Permutations P and Q. For example, R, S, and T in Figure 2, are respectively indexed $\langle 6, 4\rangle$, $\langle 2, 3\rangle$, and $\langle 7, 5\rangle$. An I entry at $\langle 2, 3\rangle$ is further refined to $\langle 4, 6\rangle$ and $\langle 5, 7\rangle$ (its northwest and southeast sons are I), yielding the index sequence:

$$\langle 6, 4\rangle,\ \langle 4, 6\rangle,\ \langle 5, 7\rangle,\ \langle 7, 5\rangle.$$

Projecting on the first components yields the sequence, $6, 4, 5, 7$, appearing in the index sequence of I entries in P of Figure 3:

$$\langle 4, 6\rangle,\ \langle 5, 4\rangle,\ \langle 6, 5\rangle,\ \langle 7, 7\rangle.$$

Projecting on the second components yields the sequence, $4, 6, 7, 5$, which appears in the index sequence of I entries there in Q:

$$\langle 4, 4\rangle,\ \langle 6, 5\rangle,\ \langle 7, 6\rangle,\ \langle 5, 7\rangle.$$

A Haskell [11] declaration for a *Matrx* that unifies both permutation and ordinary matrices appears below. It is a constructed data type that has four alternatives: 0, a 1×1 scalar matrix, a list of four quadrants, and I for permutation matrices. Appendix A exhibits code for simple operations on it that take advantage of the algebraic properties of 0 and I.

```
type Quadrants a = [Matrx a]                    --list of *four* submatrices.
data Matrx  a =   ZeroM | ScalarM a | Mtx  (Quadrants a)
                        | IdentM             --used only in permutations.
```

3 Applying the decomposition to quadtrees

Convention. *When applying Algorithm 1 to the quadtree representation, the values of i, j, k, n there are restricted. The second two must be powers of 2; $2^q = k \leq n = 2^p$. Furthermore, the first two must be multiples of k; e.g. $0 \leq i = ck < n$.*

Theorem 2 . *The pivot block selected from an $n \times n$ quadtree matrix by Label $\langle i, j, k \rangle$ in Algorithm 1, has an Ahnentafel index $\langle (n+i)/k,\ (n+j)/k \rangle$.*

Proof: The number of larger quadrants in the matrix, nodes above that block in the tree, is $\sum_{0 \leq i < \lg(n/k)} 2^i = n/k - 1$. The number of equivalently sized blocks in its row but to its west is i/k. So its row index, including itself, is $(n/k - 1) + i/k + 1$. □

Theorem 3 . *When A in Algorithm 1 is represented as a quadtree and split into four subtrees, A_{mc} intersects exactly one of the subtrees.*

Proof: The convention on i, j, k, and n implies either that $k = n$ and $A_{mw} = A$, or that $k \leq n/2$ while i, j are multiples of k; then A_{mw} is located within one of A's four quadrants. □

Theorem 4 . *Algorithm 1 can be programmed recursively following the quadtree decomposition, as in Appendix A.*

Proof: The key to the program is Theorem 3. Once the pivot has been identified, it either coincides with the matrix A (see Algorithm 3), or it is *within* one quadrant, and the algorithm can be applied recursively. The following modification to Algorithm 1 elucidates the recurrence with two changes: the size, n, is maintained as an invariant power of 2 by padding zeroes in place of (middle and central) eliminated blocks, and the quadrant recursion is sketched, using the new indices h and v, read "vertical" and "horizontal". Since the pivot, A_{mc}, is entirely within one quadrant, it orients the other

three: horizontal vertical, and diagonal from it. The diagonal quadrant is, thus, A_{vh} and the vertical (and horizontal) block is further decomposed: west, central, east (respectively north, middle, south). In the following sketch, the pivot appears northwest and the diagonal block is southeastern; with full pivoting any of three other orientations are also possible.

Let the block decomposition of nontrivial A, isolating the pivot block A_{mc} and diagonal block A_{vh}, be labeled

$$A = \begin{array}{r} \\ i \\ k \\ n/2-i-k \\ n/2 \end{array} \begin{array}{c} \begin{array}{cccc} j & k & n/2-k-j & n/2 \end{array} \\ \left(\begin{array}{ccc|c} A_{nw} & A_{nc} & A_{ne} & A_{nh} \\ A_{mw} & A_{mc} & A_{me} & A_{mh} \\ A_{sw} & A_{sc} & A_{se} & A_{sh} \\ \hline A_{vw} & A_{vc} & A_{ve} & A_{vh} \end{array}\right) \end{array}$$

where A is $n \times n$, A_{ne} is $i \times j$, A_{mc} is $k \times k$, and A_{vh} is $n/2 \times n/2$. Then

$$\langle P, Q, A', D' \rangle = \left\langle \left(\begin{array}{ccc|c} 0 & I & 0 & 0 \\ P_w & 0 & P_e & P_h \end{array}\right), \left(\begin{array}{cc} 0 & Q_n \\ I & 0 \\ 0 & Q_s \\ \hline 0 & Q_v \end{array}\right), \right.$$

$$\left. \left(\begin{array}{ccc|c} A'_{nw} & A'_{nc} & A'_{ne} & A'_{nh} \\ A'_{mw} & 0 & A'_{me} & A'_{mh} \\ A'_{sw} & A'_{sc} & A'_{se} & A'_{sh} \\ \hline A'_{vw} & A'_{vc} & A'_{ve} & A'_{vh} \end{array}\right), \left(\begin{array}{ccc|c} D'_{nw} & 0 & D'_{ne} & D'_{nh} \\ 0 & D'_{mc} & 0 & 0 \\ D'_{sw} & 0 & D'_{se} & D'_{sh} \\ \hline D'_{vw} & 0 & D'_{ve} & D'_{vh} \end{array}\right) \right\rangle$$

where P_w is $(n-k) \times i$, P_e is $(n-k) \times (n/2-k-i)$, P_h is $(n-k) \times (n/2)$, Q_n is $j \times (n-k)$, Q_s is $(n/2-k-j) \times (n-k)$, and Q_v is $(n/2) \times (n-k)$. These values can be computed as follows: first

$$D'_{mc} = A_{mc}^{-1}$$

is solved recursively; then the pivot row and pivot column are completed:

$$\left(\begin{array}{ccc|c} A'_{mw} & 0 & A'_{me} & A'_{mh} \end{array} \right) = D'_{mc} \left(\begin{array}{ccc|c} A_{mw} & 0 & A_{me} & A_{mh} \end{array} \right);$$

$$\left(\begin{array}{c} A'_{nc} \\ 0 \\ \hline A'_{sc} \\ \hline A'_{vc} \end{array}\right) = \left(\begin{array}{c} A_{nc} \\ 0 \\ \hline A_{sc} \\ \hline A_{vc} \end{array}\right) D'_{mc}.$$

Then

$$\left\langle \left(\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ P_w & 0 & P_e & P_h \end{array}\right), \left(\begin{array}{cc} 0 & Q_n \\ 0 & 0 \\ \hline 0 & Q_s \\ 0 & Q_v \end{array}\right), \left(\begin{array}{ccc|c} A'_{nw} & 0 & A'_{ne} & A'_{nh} \\ 0 & 0 & 0 & 0 \\ \hline A'_{sw} & 0 & A'_{se} & A'_{sh} \\ A'_{vw} & 0 & A'_{ve} & A'_{vh} \end{array}\right), \left(\begin{array}{ccc|c} D'_{nw} & 0 & D'_{ne} & D'_{nh} \\ 0 & 0 & 0 & 0 \\ \hline D'_{sw} & 0 & D'_{se} & D'_{sh} \\ D'_{vw} & 0 & D'_{ve} & D'_{vh} \end{array}\right) \right\rangle$$

results from recursive pivoting on the $n \times n$ problem

$$\left(\begin{array}{ccc|c} A_{nw} & 0 & A_{ne} & A_{nh} \\ 0 & 0 & 0 & 0 \\ \hline A_{sw} & 0 & A_{se} & A_{sh} \\ A_{vw} & 0 & A_{ve} & A_{vh} \end{array}\right) - \left(\begin{array}{c} A_{nc} \\ 0 \\ \hline A_{sc} \\ A_{vc} \end{array}\right) \left(\begin{array}{ccc|c} A'_{mw} & 0 & A'_{me} & A'_{mh} \end{array}\right).$$

The bars in this sketch illustrate how recursive computations in the pivot quadrant (to the northwest here) can proceed independently of the other quadrants. Then, with minimal transfer of information: D'_{mc} and $\left(\begin{array}{ccc} A'_{mw} & 0 & A'_{me} \end{array}\right)$ to the vertical quadrant and similarly D'_{mc} and $\left(\begin{array}{c} A_{nc} \\ 0 \\ A_{sc} \end{array}\right)$ to the horizontal quadrant, compute $\left(\begin{array}{ccc} A'_{vw} & A'_{vc} & A'_{ve} \end{array}\right)$ and $\left(\begin{array}{c} A'_{nh} \\ A'_{mh} \\ A'_{sh} \end{array}\right)$, perhaps simultaneously; and propagate A_{vc} and $A'mh$ to the diagonal quadrant to compute A'_{vh} as a block (or quadrant) "flop." □

Ahnentafel indices are useful here to locate padding that represents eliminated elements in this version of Algorithm 1. A similar mechanism was used in the code for scalar Jordan elimination [15].

Corollary 3 . *Algorithms 2 and 3 can be programmed recursively following the quadtree decomposition.*

The code for Algorithm 1 is symmetric with respect to orientation of the recurrence. In order to maintain the structure of the matrix, eliminated portions of the matrix are retained within A' in place of zero padding, but are tagged—much like the shading of areas in Figure 2.

For the pivoting illustrated in Figure 2, the results from $(L+U), D'$ decomposition are illustrated by $L+U$ at the bottom of Figure 1 and D' at the

bottom of Figure 2. Both of them fit the quadtree representation well. D' is preferable to D^{-1} of Figure 1 because the block decomposition exactly fits the quadrant representation. Thus, subsequent multiplications on in in Algorithms 3 and 4 are localized to subtrees coincident with represented blocks of quasi-diagonal D. $L+U$ is represented as efficiently as A' in Figure 1 because the only special blocks that cross quadrant boundaries are zero anyway; their representation is compact whether or not they are further decomposed. Thus, when Algorithms 3 and 4 cleave L and U by halves, rather than according to the pivot geography, there is minimal additional overhead for these zeroes; it is a small price to obtain the benefits in numeric accuracy (binary-tree association of addition) and in parallelism (subprocesses of equal size) from a balanced decomposition of the recursions that follow Algorithm 2.

The search for the next pivot can be distributed across the traversal that is implicit in each pivot step [15]. The idea is to use the tree as a search tree with decorations at each node pointing toward the preferred pivot within that submatrix; as each node is visited the decoration results from a tournament round indicating which of its four sons (or itself) is the locally preferred pivot. Zero and eliminated submatrices are disqualified from the local tournament.

Because this decomposition is targeted toward block operations (like those in Appendix A), it is important that the matrix *not* be permuted *during* decomposition, so that uneliminated blocks retain their integrity, locality, and—as often as possible—their decorations.[5] Therefore, the candidates pivot blocks must, themselves, be subtrees. For example, S cannot be a pivot candidate in Figure 1 because it crosses subtree boundaries, but in Figure 2 it is. Algorithm 1 assures that the S^{-1} and 0 blocks resulting from Pivot S in Figure 2 land at exactly the same geography (indexing) in A' and D' as that of S. Thus, the identification of S as a block assures the efficient representation of the resulting eliminations.

Moreover, the payoff from S's integrity in Figure 2 continues in Algorithm 3, where the matrix-vector (respectively, matrix-matrix) product decomposes so that S multiplies a single subvector (submatrix); both suboperands are already represented as subtrees. In contrast, rearranging D' to D^{-1} as in Figure 1 splits S across four quadrants, the Algorithm 3 product no longer has such nicely encapsulated factors, and so it loses its locality.

[5] Not to mention that row/column permutations on these tree structures are deceptively expensive and, therefore, to be avoided.

Issues to be considered in selecting pivot candidates are arithmetic (sustaining stability of floating-point computation, or avoiding large intermediate results when using big integers), minimizing fill-in (for sparse matrices), and increasing the size of the chosen pivot block (for parallelism), but these criteria are not always consistent. Demmel's and Higham's [5] example of unstable block LU factorization demonstrates one conflict: between good arithmetic and larger pivots. Fortunately, there are fewer subtrees in a quadtree representation than there are subblocks in the matrix of corresponding size, so the search for a good pivot is confined. A heuristic is now presented to illustrate how the quadtree structure can be used to identify larger pivot blocks.

Definition. *A matrix of order* 2^p *is* nown-singular *if* $p < 2$ *and it is nonsingular, or if* $p = q + 1$, *and one of its four quadrants of order* 2^q *is* 0, *and the two adjacent quadrants are nown-singular.*

It is intended that nown-singular blocks have determinants that are easily computed bottom-up, and easily screened as pivot candidates. This definition provides base cases of 1×1 and 2×2. Alternatively, any 3×3 block might also be included as basic because its determinant is so easily computed.

The term, "nown-singular," is chosen to suggest that, under the quadtree matrix representation, such blocks are easily *known* to be *non*-singular. However, not every uneliminated, non-singular quadrant will be nown-singular and so a pivot candidate; therefore, this is a fast, but partial, filter for pivot blocks. Even when a non-zero matrix is not nown-singular, some submatrix of it must be; undulant-block pivoting will treat it as a pivot candidate.

Theorem 5 . *Every nown-singular matrix is non-singular.*

Theorem 6 . *Every non-singular triangular matrix of order* 2^p *is nown-singular.*

Corollary 4 . *Every non-singular,* 2×2, *block-triangular matrix of order* 2^p *is nown-singular.*

Theorem 7 . *The inverse of a nown-singular matrix is nown-singular.*

Definition. *The* known-determinant *of a matrix is the magnitude of its determinant if it is nown-singular, and zero otherwise.*

A non-zero known-determinant not only identifies a candidate pivot block, but it also measures on whether it is a stable one. Under undulant-block

pivoting some ordering is needed to compare two blocks of different sizes. Something like Wilkinson's results on scalar pivots should extend to undulant block algorithms [5]. The ability to compute a block's determinant so cheaply suggests that computation of other scaled measures are also tractable: for instance, the geometric mean of a block's eigenvalues or the harmonic mean[6] of its scalar pivots.

4 Exact-arithmetic decomposition

Integer solutions for linear systems are important in rational arithmetic and symbolic computation. The usual algorithm is due to Bareiss [1], whose solutions are optimal in the sense that the denominator is the determinant, d, of the input matrix, A. That such a denominator is necessary can be observed by considering (the much less efficient) Cramer's rule. A solution that derives solutions exclusively using integer addition and multiplication up to this final quotient, is said to be *division-free*. If the denominator there is no larger than that determinant, then the division-free solution is considered to be optimal. While Bareiss's solution is division-free and optimal, it seems to require intensive, nonlocal computation that is more typical of algorithms for non-sparse matrices. The problem of adapting his algorithm to sparse quadtree representation is left open.

This section extends the $(L+U), D'$ decomposition to exact arithmetic.[7] Any use of Gaussian elimination for such problems is likely sub-optimal, because the implicit, independent solutions on L and U from an $(L+U), D'$ decomposition can each introduce a denominator of d to the solution; d occurs as a denominator in both L and U. That is, we can expect the denominator common to such a division-free solution to be d^2 at least.

If all pivot blocks are nown-singular, then the cumulative denominator is the product of all their known-determinants. In order to restrain its growth, therefore, one should seek a candidate pivot block with a *small*[8] known-

[6]Suggested by Yugo Kasiwagi.

[7]At this writing the algorithms reported in this section have not yet been implemented.

[8]An strange antithesis is that pivoting under exact, an floating-point, arithmetic requires "smaller" pivots to constrain growth of that denominator, rather than "larger" ones to assure stability. Nevertheless, the algorithms for pivot search and their goal of on-time accuracy remain the same.

determinant—one, if possible. Moreover, it is desirable common divisors among the elements of a matrix be factored out.

Algorithm 5 (Greatest common divisor of a matrix's elements.) If A is `ZeroM` then $\gcd A = 0$; if A is `ScalarM` i then $\gcd A = |i|$. If A is `Mtx` $[nw, ne, sw, se]$ then

$$\gcd A = \gcd(\gcd(\gcd nw, \gcd ne), \gcd(\gcd sw, \gcd se)).\square$$

That is, the gcd of a matrix can be computed as the matrix is built. It might be stored as a decoration at the root of a quadtree, for use when the matrix is borrowed.

In this section, all matrices L, D, U, F, A are integer matrices. They correspond to real matrices of the same shapes from Section 1, (which are temporarily renamed $\overline{L}, \overline{D}, \overline{U}$, *etc.* here.) Following the naming of Section 1, we seek a decomposition:

$$PAQ = (I + LF^{-1})(DF^{-1})(F^{-1}U + I)$$

where $D' = (PDF^{-1}Q)^{-1}$, P, Q are permutation matrices, L, U are proper lower, upper triangular matrices, D is block diagonal whose blocks correspond to successive pivots, and F is a diagonal matrix of factors, the determinants of those pivots.

Algorithm 6 (Pivoting nonsingular A to $\langle P, Q, L + U, D', F\rangle$.) The input parameters are $\langle d, A\rangle$, where d is the cumulative denominator. The quintuple $\langle P, Q, A', D', F\rangle$ results from recursive pivoting on a factor d (initially 1) and an integer matrix A, described recursively as follows. If A is void then the result is $\langle I, I, 0, d, 0\rangle$.

(Optionally use Algorithm 5 to eliminate the factor $\gcd(d, \gcd A)$ from input; however, Algorithm 6 is no longer division-free.)

Otherwise, let the block decomposition of A, isolating the pivot block A_{mc}, be labeled as in Algorithm 1, and let $(\det A_{mc}) = f$ so that $J = fA_{mc}^{-1}$ is also an integer matrix.

$$A = \begin{array}{c} \\ i \\ k \\ n-k-i \end{array} \begin{array}{c} \begin{array}{ccc} j & k & n-k-j \end{array} \\ \begin{pmatrix} A_{nw} & A_{nc} & A_{ne} \\ A_{mw} & A_{mc} & A_{me} \\ A_{sw} & A_{sc} & A_{se} \end{pmatrix} \end{array};$$

Then

$$\langle P, Q, A', D', F \rangle = \left\langle \begin{pmatrix} 0 & I & 0 \\ P_w & 0 & P_e \end{pmatrix}, \begin{pmatrix} 0 & Q_n \\ I & 0 \\ 0 & Q_s \end{pmatrix}, \begin{pmatrix} A'_{nw} & A'_{nc} & A'_{ne} \\ A'_{mw} & 0 & A'_{me} \\ A'_{sw} & A'_{sc} & A'_{se} \end{pmatrix}, \right.$$

$$\left. \begin{pmatrix} D'_{nw} & 0 & D'_{ne} \\ 0 & D'_{mc} & 0 \\ D'_{sw} & 0 & D'_{se} \end{pmatrix}, \begin{pmatrix} f & 0 \\ 0 & F' \end{pmatrix} \right\rangle$$

where P_w is $(n-k) \times i$, P_e is $(n-k) \times (n-k-i)$, Q_n is $j \times (n-k)$, Q_s is $(n-k-j) \times (n-k)$, and f is interpreted as a $k \times k$ scalar matrix. These values can be computed as follows: first J and f are solved recursively and D'_{mc} is defined as dJ.

$$D'_{mc} = dJ = (df)A_{mc}^{-1};$$

then the pivot row and pivot column are completed,

$$\begin{pmatrix} A'_{mw} & A'_{me} \end{pmatrix} = D'_{mc} \begin{pmatrix} A_{mw} & A_{me} \end{pmatrix}$$

$$\begin{pmatrix} A'_{nc} \\ A'_{sc} \end{pmatrix} = \begin{pmatrix} A_{nc} \\ A_{sc} \end{pmatrix} D'_{mc};$$

finally,

$$\left\langle \begin{pmatrix} P_w & P_e \end{pmatrix}, \begin{pmatrix} Q_n \\ Q_s \end{pmatrix}, \begin{pmatrix} A'_{nw} & A'_{ne} \\ A'_{sw} & A'_{se} \end{pmatrix}, \begin{pmatrix} D'_{nw} & D'_{ne} \\ D'_{sw} & D'_{se} \end{pmatrix}, F' \right\rangle$$

results from recursive pivoting on the $(n-k) \times (n-k)$ problem: df and

$$f \begin{pmatrix} A_{nw} & A_{ne} \\ A_{sw} & A_{se} \end{pmatrix} - \begin{pmatrix} A_{nc} \\ A_{sc} \end{pmatrix} \begin{pmatrix} A'_{mw} & A'_{me} \end{pmatrix};$$

(optionally using Algorithm 5 within the subtraction.) Then $L + U = PA'Q$ as in Algorithm 2. □

Theorem 8 . *Algorithm 6 is weakly correct.*

Proof: If Algorithm 6 halts, then it yields permutations P and Q. The rest of this argument presumes that they have been provided by an oracle *a priori*, and is cast as if the input, 1 and PAQ, were provided in place of

1 and A. Thus, the north and west blocks $A_{nw}, A_{nc}, A_{ne}, A_{mw}, A_{sw}$ would be void, and we may limit ourselves to the simpler block decomposition

$$A = \begin{pmatrix} A_{mc} & A_{me} \\ A_{sc} & A_{se} \end{pmatrix}$$

at each step of the recurrence. No pivoting is required and Figure 1 applies.

Now suppose that Algorithms 1 and 6 are run simultaneously on an integer matrix, pivoting on geographically corresponding blocks. A "bar" notation is used to identify data for Algorithm 1; "unbarred" variables denote corresponding data for Algorithm 6. It can be shown by simple induction on the number of pivot steps that Algorithm 1 and Algorithm 6 correspond in the following way:

$$d\overline{A} = A$$

$$f\overline{D'}_{mc} = D'_{mc}$$

$$f\overline{A'}_{sc} = A'_{sc}; \quad f\overline{A'}_{me} = A'_{me};$$

$$fd\overline{A'}_{se} = A'_{se}.$$

From these equations, and the arrangement of F and A' during the recurrence, the following equations hold after pivoting:

$$\overline{L} = LF^{-1}; \overline{U} = F^{-1}U; \overline{D^{-1}} = D'F^{-1} = F^{-1}D'.$$

□

In the following applications of the integer $(L+U), D'$ decomposition, it will be useful to precompute the greatest common divisors of collections of adjacent elements in F. When F is stored as a binary vector or as a quadtree matrix, this is best done with one bottom-up traversal of F, storing the `gcd` and `lcm` of each subtree at its root.

Algorithm 7 (Exact solution for a linear system.) Given an integer matrix, A, and an integer vector, $\vec{b}$, solve $A\vec{x}/f = \vec{b}$ for integer vector, $\vec{x}$, and integer denominator, f.

1. Apply Algorithm 6 to $\langle 1, A\rangle$, yielding $\langle P, Q, L+U, D', F\rangle$, and use this reformulation:

$$A(\vec{x}/f) = P^{-1}(I+LF^{-1})Q^{-1}(QDF^{-1}P)P^{-1}(F^{-1}U+I)Q^{-1}(\vec{x}/f) = \vec{b}.$$

2. [forward substitution] Solve for integer $\langle \vec{y}, e \rangle$ from $\langle \vec{c}, d \rangle$ to satisfy:

$$(I + LF^{-1})\vec{y}/e = P\vec{b} = \vec{c}/d$$

with d initially 1.

- If $L + U = 0$ then $\langle \vec{y}, e \rangle = \langle \vec{c}, d \rangle$.
- Otherwise, partition

$$L + U = \begin{pmatrix} L_{nw} + U_{nw} & E \\ W & L_{se} + U_{se} \end{pmatrix};$$

$$F = \begin{pmatrix} F_{nw} & 0 \\ 0 & F_{se} \end{pmatrix}; \quad \vec{c} = \begin{pmatrix} \vec{c_n} \\ \vec{c_s} \end{pmatrix}.$$

- Recursively solve $(I + L_{nw}F_{nw}^{-1})\vec{y_n}/e_n = \vec{c_n}/d$ to obtain $\langle \vec{y_n}, e_n \rangle$ and let $e = \operatorname{lcm}(d, e_n \operatorname{lcm} F_{nw})$.
- Recursively solve $(I + L_{se}F_{se}^{-1})\vec{y_s} = \vec{c_s}/d - WF_{nw}^{-1}\vec{y_n}/e_n$. The right-hand side is rewritten $[r\vec{c_s} - W\vec{y_n}(sF_{nw}^{-1})]/e$ where $r = e/d$; $s = e/e_n$, making the numerator integral. Then the solution is

$$\left\langle \vec{y} = \begin{pmatrix} s\vec{y_n} \\ \vec{y_s} \end{pmatrix}, e \right\rangle.$$

3. [backward substitution] Similarly, solve for integer $\langle \vec{z}, f \rangle$:

$$(F^{-1}U + I)\vec{z}/f = P(D'(Q\vec{y}))/e.$$

4. Permute $\vec{x} = Q\vec{z}$ to get the result $\langle \vec{x}, f \rangle$.

□

When binary vectors are used to represent the intermediate solutions, $\vec{y}$ and $\vec{z}$, and if minimization of f or elementwise reduction of $\vec{x}/f$ to lowest terms is desired, then each subtree (subvector) should be decorated at its root with the greatest common divisor of its two components (*cf.* Algorithm 5). This is best done bottom-up as a vector is built.

Algorithm 8 (Exact matrix inversion.) Similar to Algorithm 7; see Appendix B.□

5 Conclusions

Not so long before the serial addressing of a Fortran array fixed our attention on row/column operations, undulant-block decomposition was recognized [6] as an important way to divide-and-conquer matrix problems. More recently, block organization re-emerged [10] for sparse matrices [9] and as a tactic to improve locality, a problem that manifested itself earlier as page faults [7], and lately as block transfers among local memories of a multiprocessor [8].

The idea of distributing the search across the preceding pivot [15] is not new [13, p. 154], but it is interesting that its disuse there was not attributed to limitations of architecture, but rather (at that time, before RISC) to the limitations of the programming language. Not only can contemporary programming styles better handle functions that return multiple results [11], they are also better designed to use tree structures that can stash intermediate results at internal nodes.

We have implemented several kinds of undulant-block pivoting on quadtrees using the following global strategy: compute local measures and run a tournament to bubble the best choice up the quadtree as it is rebuilt during the preceding pivot step. Since a good sparse-matrix algorithm avoids visiting too many elements, it is useful to store some of these measures as decorations at interior nodes to enhance the impact of sparseness: much of the matrix is not traversed at each step. Several different attributes identify a favorable candidate for a pivot block: large magnitude for floating point stability, small magnitude for integer arithmetic, large size for parallelism, and minimal Markovitz measures to reduce fill-in.

Repeated reorderings of the matrix during elimination is avoided because the quadtree matrix representation makes them expensive. However, a good *a priori* ordering (before elimination begins) remains an open problem. Certainly such an ordering is desirable to assemble dependencies as much as possible into one quadrant, rather than allowing them to remain distributed randomly across the matrix.

Quadtree matrices offer a *uniform* representation that allows a single algorithm to perform reasonably well on both sparse and dense matrices, on both uniprocessors and on parallel processors. However, it requires a heap resource (that is, a memory full of nodes for linked allocation, as well as a storage manager for it) that is unusual in algebraic languages like Fortran, but is conventional to languages like Lisp, Scheme, and Smalltalk, and even in

operating systems. The heap sometimes becomes a system-level resource in such environments, where multitasking or hardware support has sometimes been provided to enhance heap management.

As parallel architectures become more available, new multiprocessor algorithms like these will help create a demand for heap-based multiprocessing. Architectural support for storage management will be necessary in order to ameliorate the interprocessor synchronization required to manage it. The algorithms presented here represent a large number of problems that build and abandon large intermediate structures, and that have already motivated the architectures of huge machines. Yet all the structures discussed here are trees or dags, rather than arbitrary graphs; so a strategy as simple as reference counting, for example, is sufficient to manage a multiprocessor heap for this large class of problems. These algorithms are being used to benchmark such heap-managing hardware [16].

This paper shows how to implement undulant-block pivoting in general, and shows that it fits well the quadtree representation of matrices. Quadtrees restrict the search for pivot blocks, even under full pivoting, and admit a speedy heuristic for identifying non-trivial, non-singular pivot blocks of various sizes, as well as a strategy for managing arithmetic on them. Thus, they illustrate one way to implement undulant-block efficiently. Tests on its performance are in progress and early results are most encouraging [2].

Acknowledgements

Many people have made suggestions related to this material, but I am particularly indebted to S. Kamal Abdali and Tektronix Labs, where this work began.

References

[1] E. R. Bareiss. Sylvester's identity and multistep integer-preserving Gaussian elimination. *Math. Comp.* **22**, 103 (July 1968), 565-578.

[2] P. Beckman. Static measures of quadtree prepresentation of the Harwell-Boeing sparse matrix collection. Tech. Rept. 324, Computer Science Dept., Indiana University (Jan. 1991).

[3] H. G. Cragon A historical note on binary tree. *ACM Computer Architecture News* **18**, 4 (Dec 1990), 3.

[4] I. S. Duff, A. M. Erisman, & J. K. Reid. *Direct Methods for Sparse Matrices.* Clarendon Press, Oxford (1989).

[5] J. W. Demmel & N. J. Higham. Stability of block algorithms with fast Level 3 BLAS. LAPACK Working Note 22, Computer Science Dept., The Univ. of Tennessee (revised: July 1991). *ACM Trans. Math. Software* (to appear).

[6] V. N. Faddeeva *Computational Methods of Linear Algebra.* Dover Publications, New York (1959). Translated from Russian, originally published Moscow (1950).

[7] P. C. Fischer & R. L. Probert. Storage reorganization techniques for matrix computation in a paging environment. *Comm. ACM* **22**, 7 (July 1979), 405–415.

[8] K. A. Gallivan, R. J. Plemmons, & A. H. Sameh. Parallel algorithms for dense linear Algebra computations. *SIAM Review* **32**, 1 (March 1990), 54-135.

[9] A. George. On block elimination for sparse linear systems. *SIAM J. Numer. Anal.* **11**, 3 (June 1974), 585–603.

[10] G. H. Golub & C. F. Van Loan. *Matrix Computations* 2nd edition. The Johns Hopkins University Press, Baltimore (1989).

[11] P. Hudak & P. Wadler (eds.) Report on the Programming Language Haskell, a Non-strict, Purely Functional Language. Technical Report YALEU/DCS/RR777, Department of Computer Science, Yale University. (April 1990).

[12] O. Østerby & Z. Zlatev. *Direct Methods for Sparse Matrices. Lecture Notes in Computer Science* **157**. Berlin, Springer (1983).

[13] G. W. Stewart *Introduction to Matrix Computations.* Academic Press, New York (1973).

[14] D. S. Wise. Representing matrices as quadtrees for parallel processors (extended abstract). *ACM SIGSAM Bulletin* **18**, 3 (August 1984), 24–25.

[15] D. S. Wise. Parallel decomposition of matrix inversion using quadtrees. *Proc. 1986 International Conference on Parallel Processing* (IEEE Cat. No. 86CH2355–6), 92–99.

[16] D. S. Wise. Design for a multiprocessing heap with on-board reference counting. In J.-P. Jouannaud (ed.), *Functional Programming Languages and Computer Architecture, Lecture Notes in Computer Science* **201**, Berlin, Springer (1985), 289-304.

[17] D. S. Wise & J. Franco. Costs of quadtree representation of non-dense matrices. *J. Parallel Distrib. Comput.* **9**, 3 (July 1990), 282–296.

A Quadtree representation in Haskell.

```
type Quadrants a = [Matrx a]                --list of *four* submatrices.
data Matrx  a =   ZeroM | ScalarM a | Mtx  (Quadrants a)
                        | IdentM            --used only in permutations.
instance (Num a) => Num (Matrx a) where
    fromInteger 0          = ZeroM
    fromInteger 1          = IdentM
    negate 0               = 0
    negate (ScalarM x)     = ScalarM (negate x)
    negate (Mtx quads)     = Mtx (map negate quads)
    x          + 0         = x         --NB:addition won't handle IdentM
    0          + y         = y
    ScalarM x + ScalarM y = case x+y of  0 -> 0
                                         z -> ScalarM z
    Mtx x      + Mtx y     = case  zipWith (+) x y  of [0,0,0,0] -> 0
                                                       quads  -> Mtx quads
    x          - 0         = x
    0          - y         = negate y
    ScalarM x - ScalarM y = case x-y of  0 -> 0
                                         z -> ScalarM z
    Mtx x      - Mtx y     = case  zipWith (-) x y  of [0,0,0,0] -> 0
                                                       quads  -> Mtx quads
    0          * _         = 0
    _          * 0         = 0
    1          * y         = y      --NB: multiplication accepts IdentM
    x          * 1         = x
    ScalarM x * ScalarM y = ScalarM (x*y)
       --Except with infinitesimal floats:  case x*y of  0->0; z->ScalarM z
    Mtx x      * Mtx y     = (case zipWith (+)
                                  (zipWith (*) (colExchange x)(offDiagSqsh y))
                                  (zipWith (*)       x        (prmDiagSqsh y))
                              of     [0,0,0,0] -> 0
                                     quads     -> Mtx quads
                            ) where colExchange [nw,ne,sw,se] = [ne,nw,se,sw]
                                    prmDiagSqsh [nw,ne,sw,se] = [nw,se,nw,se]
                                    offDiagSqsh [nw,ne,sw,se] = [sw,se,sw,se]
```

B Exact matrix inversion.

Algorithm 8 (Exact matrix inversion.) Invert

$$A = A/1 = P^{-1}(I + LF^{-1})[Q^{-1}(QDF^{-1}P)P^{-1}](F^{-1}U + I)Q^{-1}.$$

The solution is a ratio of integer matrix and a scalar denominator, f.

1. Compute the exact $(L+U), D'$ decomposition of A using Algorithm 6: $\langle P, Q, L+U, D', F\rangle$,.

2. If $L + U = 0$ then let $f = \operatorname{lcm} F$ and

$$A^{-1} = QF^{-1}PD'QP = (Q(fF^{-1})P)D'(QP)/f;$$

3. Otherwise compute $L'/d = (I + LF^{-1})^{-1}$ and $U'/e = (F^{-1}U + I)^{-1}$ recursively; then $A^{-1} = Q(U'(PD'Q)L')P/(de)$.

   - Partition $L + U$ as in the previous algorithm.
   - Recursively compute

$$L'_{nw}/d_n = (I + L_{nw}F_{nw}^{-1})^{-1}; L'_{se}/d_s = (I + L_{se}F_{se}^{-1})^{-1};$$

$$U'_{nw}/e_n = (F_{nw}^{-1}U_{nw} + I)^{-1}; U'_{se}/e_s = (F_{se}^{-1}U_{se} + I)^{-1}.$$

   - Let $d = d_n d_s \operatorname{lcm} F_{nw}$; $e = e_n e_s \operatorname{lcm} F_{nw}$
   - Then the following are integer matrices:

$$\begin{aligned} L' &= \begin{pmatrix} (d/d_n)L'_{nw} & 0 \\ -L'_{se}W[(\operatorname{lcm} F_{nw})F_{nw}]^{-1}L'_{nw} & (d/d_s)L'_{se} \end{pmatrix}; \\ U' &= \begin{pmatrix} (e/e_n)U'_{nw} & -U'_{nw}[(\operatorname{lcm} F_n w)F_{nw}]^{-1}EU'_{se} \\ 0 & (e/e_s)U'_{se} \end{pmatrix}. \end{aligned}$$

   - When $f = (\det A)$ is known *a priori*, divide every element of $U'(PD'Q)L'$ by $h = de/f$ to get B. Otherwise let $f = de; h = 1$.
   - The final result is $\langle PBQ, f\rangle$. (Further reductions may yet be possible using Algortihm 5.)

□

Reduction to lowest terms is particularly important when Algorithm 8 is applied to non-trivial pivots in Algorithm 6. In that case, some reduction is assured without any gcd computation at its last step because the determinant is known. If no reduction is performed, then $e = d$ at every step of the recurrence and f can grow rapidly. Therefore, it becomes the more important to choose pivots to minimize the magnitude of elements of F, especially to the northwest, *i.e.* at the earlier pivots. Thus arises a preference to pivot on blocks with unit determinant, which would be limited to pivots on 1, itself (or its equivalences under symbolic computation), if scalar pivots were the only choice. However, undulant-block pivoting admits other candidates: for instance $\begin{pmatrix} 7 & 8 \\ 8 & 9 \end{pmatrix}$ is an excellent nown-singular pivot. With luck, larger "unitary" pivots will be found.