Costs of Quadtree Representation of Non-dense Matrices

David S. Wise * & John Franco †

Computer Science Department, Indiana University
101 Lindley Hall, Bloomington, IN 47405-4101
dswise@iuvax.cs.indiana.edu franco@iuvax.cs.indiana.edu

TECHNICAL REPORT NO. 229

Costs of Quadtree Representation Of Non-dense Matrices

by

David S. Wise & John Franco

Revised: February, 1989

* Research reported herein was sponsored, in part, by the National Science Foundation under Grant Number DCR 84-05241.

† Research reported herein was sponsored, in part, by the U.S. Air Force under a grant numbered AFOSR 84-0372.

Costs of Quadtree Representation of Non-dense Matrices

David S. Wise * & John Franco †

Computer Science Department, Indiana University
101 Lindley Hall, Bloomington, IN 47405-4101
dswise@iuvax.cs.indiana.edu franco@iuvax.cs.indiana.edu

CR categories and Subject Descriptors:
E.1 [**Data Structures**]: Trees; G.1.3 [**Numerical Linear Algebra**]: Sparse and very large systems; F.2.2 [**Nonnumerical Problem Complexity**]: Computations on discrete structures; D.1.1 [**Applicative (Functional) Programming Techniques**].
General Term: Measurement.

Abstract

The quadtree representation of matrices is a uniform representation for both sparse and dense matrices which can facilitate shared manipulation on multiprocessors. This paper presents worst case and average case resource requirements for storing and retrieving familiar families of patterned matrices: packed, symmetric, triangular, Toeplitz, and banded. Using this representation it compares resource requirements of three kinds of permutation matrices, as examples of non-dense, unpatterned matrices. Exact values for the shuffle and bit-reversal permutations (as in the fast Fourier transform), and tight bounds on the expected values from purely random permutations are derived. Two different measures, *density* and *sparsity*, are proposed from these values. Analysis of quadtree matrix addition relates density of addends to space bounds on their sum, and relates their sparsity to time bounds for computing that sum.

---

* Research reported herein was sponsored, in part, by the National Science Foundation under Grant Number DCR 84-05241.

† Research reported herein was sponsored, in part, by the U.S. Air Force under a grant numbered AFOSR 84-0372.

Section 1. Introduction

Recent papers [12, 13] have proposed a uniform quadtree representation for both dense and sparse matrices which provides a graceful decomposition of algorithms suitable for scheduling on multiprocessors. Most of the material there is about algorithms for manipulating quadtree matrices, focusing on the pattern of decomposing $n \times n$ matrices gracefully onto p processors for $p \ll n$. Some of these results are outlined at the end of the next section.

This paper presents measures for the quadtree representation that show how it compares with alternative sparse representations. Section 2 defines the quadtree representation of matrices, structuring dense and sparse matrices *indistinguishably.* This uniform representation comes at some price, of course, because a sizable tree of nonterminal nodes is above all the scalar entries in a matrix. In the usual matrix representation this tree is supplanted by hardware that directly implements the bijection from index to memory address; here the mapping is represented by levels of structure that can be explicitly shared. This sharing can be used on a multiprocessor to decompose data and processes among memories and processors to use parallel resources coherently.

Although nontrivial in comparison with the constant-time access and zero-space overhead of sequentially stored matrices, the additional overhead for the nonterminal nodes is an artificial concern for three reasons. First of all, it may be irrelevant in the appropriate algorithms (*vide infra*) because they typically use recursive descent. That is, rather than accessing elements of a matrix from the root of the tree (analogously to indexing from a distinguished memory address), these algorithms recurse to nested, successively shallower subtrees, so that only rarely is an entire path from the root traversed to manipulate just *one* element.

Secondly, even if the complete path from the root were traversed upon every probe into an array, some time spent in such a traversal might be offset by effective memory speed, improving with depth; this is purely an architectural phenomenon that is motivated by the rigors of multiprocessing. For instance, accelerated access time is realized on the later of repeated probes into a subtree that is sufficiently small to fit within cache. Something similar happens when subtrees are stored locally on distributed processors: random manipulation within a local subtree is very fast, compared to the interprocessor contention

arising from global access. Under such architectures, the improved efficiency of repeated accesses within a subtree counteracts the increased cost to isolate it.

Thirdly, heap memory, which directly supports trees and other linked structures, is an attractive architecture [6] for avoiding "hot spots" in a multiprocessor where each processor accesses shared memory through a packet switch [9]. When arrays are mapped sequentially across address space, their usual traversals generate a regular addressing pattern. Then several processors, even if running different algorithms on disjoint data but using the same address map, tend to collide repeatedly in the switch. A first collision between two might occur randomly, but the addressing patterns of those two synchronize and follow one-another through their traversals; that couple presents a larger obstacle to a third, *etc.*, and shortly a gaggle of these traversals would be driving one "hot spot" along that regular pattern through address space. Experienced Cray programmers use different "strides" in setting up coordinated pipelines to avoid just this problem.

Many problems that justify parallel computation also exhibit sparseness. Therefore, these results on space and access efficiency are of interest in designing both parallel algorithms and parallel computers [3]. It is the purpose of this paper to measure analytically the costs of storing and retrieving large matrices in the quadtree format for comparison with alternative representations. Some empirical results are already available elsewhere [1]. However, truly competitive results may require further developments in hardware and algorithms [3] that, perhaps, will be encouraged by the results presented below.

The main results are summarized in Table 1. Half of the table reports values for sparsity and density, but they are only defined and related to resource bounds on quadtree operations in Section 5. At this point the reader can readily understand the entries in the first two columns. The second column presents *exact* worst-case space required to represent an $n \times n$ matrix in quadtree format (n is a power of two.) As expected, the space for a packed, symmetric, or triangular matrix remains $O(n^2)$, but that for a Toeplitz, diagonal, or banded matrix is $O(n)$. The constants of proportionality are slightly higher than those from the usual representations of such matrices (*e.g.* sequential or vector-compressed structures), but hardly outrageous when one allows that *only one convention is used to represent all.*

Pattern	Space	Density*	Expected Path	Sparsity*
Packed	$\frac{4}{3}(n^2 - \frac{1}{4})$	1	$\lg n + 1$	0
Symmetric	$\frac{2}{3}(n+2)(n-\frac{1}{2})$	$\frac{1}{2} + \frac{0.75}{n}$	$\lg n + 1$	0
Hankel/Toeplitz	$4n - \lg n - 3$	$\frac{1}{3n} - \frac{0.75 \lg n}{n^2}$	$\lg n + 1$	0
Triangular	$\frac{2}{3}(n+2)(n-\frac{1}{2})$	$\frac{1}{2} + \frac{0.75}{n}$	$\frac{\lg n}{2} + \frac{3}{2} - \frac{1}{2n}$	$\frac{1}{2} - \frac{1}{\lg n}$
fFT permutation	$\frac{n \lg n}{2} + \frac{4n}{3} - \frac{1}{3}$	$\frac{0.375 \lg n + 1}{n}$	$\frac{\lg n}{2} + \frac{4}{3} - \frac{1}{3n}$	$\frac{1}{2} - \frac{0.83}{\lg n}$
Random permutation	$\frac{n \lg n}{2} + 0.9n - \frac{4}{3}$	$\frac{0.375 \lg n + 0.7}{n}$	$\frac{\lg n}{2} + 0.9$	$\frac{1}{2} - \frac{0.4}{\lg n}$
Diagonal	$2n - 1$	$\frac{1.5}{n}$	$2 - \frac{1}{n}$	$1 - \frac{2}{\lg n}$
Tridiagonal	$6n - 2\lg n - 5$	$\frac{4.5}{n}$	$\frac{10}{3} - \frac{3}{n} + \frac{2}{3n^2}$	$1 - \frac{3.33}{\lg n}$
Pentadiagonal	$8n - 2\lg n - 9$	$\frac{6}{n}$	$\frac{10}{3} - \frac{1}{n} - \frac{10}{3n^2}$	$1 - \frac{3.33}{\lg n}$
Heptadiagonal	$11n - 2\lg n - 19$	$\frac{8.25}{n}$	$\frac{10}{3} + \frac{5}{n} - \frac{76}{3n^2}$	$1 - \frac{3.33}{\lg n}$
Enneadiagonal	$13n - 2\lg n - 27$	$\frac{9.75}{n}$	$\frac{10}{3} + \frac{7}{n} - \frac{100}{3n^2}$	$1 - \frac{3.33}{\lg n}$
Shuffle permutation	$3(n-1)$	$\frac{2.25}{n}$	$3(1 - \frac{1}{n})$	$1 - \frac{3}{\lg n}$
Zero	0	0	0	1

Table 1. Measures of patterned and unpatterned matrices as quadtrees.

**Density* is accurate within a term of $\Theta(n^{-2})$. *Sparsity* is accurate within a term of $\Theta((\lg n)^{-2})$.

Section 2 offers the definitions and normal forms for quadtree representation; it also indicates how some parallel algorithms fit this representation. Sections 3 and 4 compare the costs of representing familiar matrices [10]; many of them are characterized by strong patterning. Section 3 presents the expected path and exact space measures (worst-case) for familiar patterns of matrices: packed, symmetric, triangular, and finally banded matrices, the most familiar of sparse matrices. Section 4 deals with very sparse matrices, patterned after permutation matrices that only have n non-zero entries: the "shuffle" and "bit-reversal" permutations, encountered in the fast Fourier transform, and random permutation matrices. The foregoing analyses support the definitions in Section 5 for the measures, *sparsity* and *density*, easily computed for any matrix representation. An analysis of matrix addition there shows that sparsity of addends relates directly to time to add them, and that their density relates to the space ocupied by their sum. Conclusions and future work appears as Section 6.

Section 2. Quadtree Representation and Algorithms

Dimension refers to the number of subscripts on an array. The *size* of a vector is the cardinality of a basis for its space, *i.e.* the number of its elements when written in the conventional tuple notation. *Order* of a square matrix, similarly, is the cardinality of a basis for its underlying vector space, the number of its rows or columns when it is written as the conventional tableau. In this paper, matrices will be square, and orders and sizes will mostly be powers of two; a convention for padding arrays of other sizes is presented after some definitions.

Thesis. *Any d-dimensional array is decomposed by blocks and represented as a 2^d-ary tree.*

Here only vectors and matrices are considered, where $d = 1$ suggests binary trees and $d = 2$ suggests quaternary trees—or quadtrees. A vector is said to be *homogeneous* if all of its scalar elements have the same value.

Binary Vector Representation. *A vector of size 2^p is either homogeneous and is represented by that scalar value, or it is not homogeneous and is represented by a binary tree whose subtrees each represent vectors of size 2^{p-1}. These subtrees are identified as left and right, isomorphic to the left and right halves of the conventional tuple notation.*

A vector of size 1 is trivially homogeneous, and is always represented by a scalar. Of particular note is the scalar 0, which is well implemented to be the pointer *NIL* in *Pascal* notation. Such an implementation recognizes the importance of quickly recognizing zero elements, because in any heap-based system all meaning of *NIL* is apparent without a memory reference. Therefore, the "zero vector" would be represented by *NIL*, as is the "zero matrix" in the following representation.

Quadtree Matrix Representation. *A matrix of order 2^p is either homogeneously zero, in which case it is represented NIL; or $p = 0$ and its element is a non-zero scalar, in which case it is represented by that scalar; or else it is represented by an ordered quaternary tree [7] whose subtrees each represent matrices of order 2^{p-1}. These subtrees are respectively identified as northwest, northeast,*

southwest, and southeast, isomorphic to the four quadrants of a block decomposition of the conventional tableau, in the order that those names suggest.

Tagged Quadtree Matrix Representation. *A quadtree matrix representation is tagged if each non-zero, non-scalar subtree is decorated by a boolean value, labeled transposed. When transposed is* **false** *the isomorphism between subtrees and subblocks remains the trivial one used in the definition of quadtree matrix representation. If transposed is* **true**, *then the isomorphism at every level of the tree exchanges the northeast and southwest subtree in that order.*

That is, if the matrix is transposed then the first subtree maps to the northwest block, the second to the southwest, the third to the northeast, and the fourth to the southeast at every level in the tree (unless deeper subtrees are also transposed.) The effect of this is to transpose the matrix from its conventional orientation.

So that this recursive cleaving works smoothly even when the order, n, is not a power of two, an $n \times n$ matrix is embedded in a $2^{\lceil \lg n \rceil} \times 2^{\lceil \lg n \rceil}$ matrix, justified at the lower, right (southeast) corner with zero padding to the north and west. Then it can be represented efficiently as a quaternary tree; padding with *NIL* minimizes the space consumed in padding. (The matrix is justified to the southeast, rather than to the northwest, in anticipation of the recurrence for eliminants [2].)

This prescribes a *normal form* for the representation of matrices as quaternary trees (henceforth "quadtrees"): no scalar entry is ever 0 and four quadrants cannot all be *NIL*. Scalars can only occur at Level $2^{\lceil \lg n \rceil}$ in the quadtree representation of an order n matrix. (This normal form for representation of any matrix should be distinguished from a normal form for that matrix, itself.) Together, the order and the quadtree representation of a matrix comprise a unique representation for the matrix.

Theorem 1. *If a matrix of order n is padded as described, then it has a unique quadtree representation. If two matrices of the same order share the same quadtree representation, then they are the same matrix.*

Proof by a simple induction on $\lceil \lg n \rceil$. There is only one way to pad the matrix out to order $2^{\lceil \lg n \rceil}$.

Of course, there can be several representations of a matrix as a tagged quadtree, accordingly as different subblocks are transposed. Knowledge of order is necessary in order to interpret *NIL* uniquely, and it becomes critical upon output. It must be acknowledged that the I/O conversions are non-trivial algorithms, but they require comparatively little processing and are also restrained by communication bandwidth whenever they are necessary. Like floating-point number conversions, however, they remain an irritating impediment to one who would experiment with the algorithms discussed below.

Similarly, a vector of arbitrary size n is embedded in one of size $2^{\lceil \lg n \rceil}$, with zero padding to the left, which then is represented as a binary tree. Thus, we obtain a normal form for representation of vectors as binary trees, which is unique when size, as well, is specified.

Corollary 1. *If a vector of size n is padded as described, then it has a unique binary tree representation. If two vectors of the same size share the same binary tree representation, then they are the same vector.*

The algorithm for matrix addition [12] decomposes naturally into four quadrant additions which are independent processes. Because of their mutual independence, these four are naturally computed in parallel within a shared memory, or distributed to independent processors with private memory. The decomposition extends naturally to 16, 64, 256, *etc.* processors, or—by splitting the sums in half, rather than in quarters—to 2, 8, 128 *etc.* as well. Whenever either addend is *NIL*, addition immediately returns a shared reference to the other addend.

The algorithm for matrix multiplication may be decomposed two ways (again treating the product as two halves), four ways (the four quadrants of the answer), and eight ways (the eight quadrant products in the block or cellular decomposition [5] of Gaussian matrix multiplication.) Whenever either factor is *NIL*, their matrix product is annihilated to *NIL*. Moreover, the resulting product matrix (of order n) is more stable than that from pipelined dot-products, because each element is computed as a sum over a tree of addends, rather than as

a serial sum of n addends; each addend participates in at most $\lg n$, rather than up to n, additions. Thus, quadtree multiplication of sparse matrices improves both on the time of Strassen's [11] (when a factor's quadrant is *NIL* only six recursive multiplications are necessary, instead of eight), and on the stability of pipelined vector processing.

The problems of solving linear systems and of matrix inversion can both be reduced to the Pivot Step problem [7, 13]. The important problem there of identifying a pivot is mitigated by the search space already being structured as a tree. The quadtree representation not only allows pivoting to occur on arbitrary elements with equal facility, but also it provides for propagation of the computation from the "pivot block" to its siblings in parallel [13], providing parallelism at a higher level in the problem's decomposition than is available using only vector operations. For instance, pivoting on an entire quadrant uses the same code as pivoting on a scalar [14].

Other conventions of representation (besides *NIL*) and other annotations on nonterminal nodes (beyond the *transposed* tag) can enhance various quadtree algorithms. For instance, any scalar, x, might be used to represent a scalar matrix $[x\delta_{i,j}]$; then the unit matrix, 1, accelerates multiplicative operations [12]. Alternatively, that scalar x might always represent a homogeneous matrix $[x]$, all of whose elements are x; under a Boolean algebra the matrix 1 could then be programmed as the additive annihilator, more useful there than a multiplicative identity. In addition to the *transposed* tag, each interior node in the quadtree might also be decorated by the magnitude of its largest element to assist in selecting a stable pivot [13]; other decorations there assist in pivot selection meeting other criteria. Theorem 1 suggests that a quadtree representation always be decorated at its root with the size of the represented matrix.

The remainder of this paper addresses static measures of the (tagged) quadtree representation. Use of the algorithms mentioned above is only incidental. Even though implicitly shared references are likely to result from many programs, none of the analyses that follow consider any sharing beyond that explicitly stated and indicated in the figures. All these measures, therefore, are conservative.

Section 3. Matrices with Elementary Patterns

In order to measure the relative cost of quadtree representation, we compute in this section the total space and path length in various familiar patterned matrices. Each scalar and each interior node are counted as one unit of memory, and each such node accessed during the fetch of a random element contributes one unit to the expected path length.

For comparison, an $n \times n$ matrix stored sequentially in memory (in the conventional manner as under *Fortran*) occupies n^2 space and has expected path length 1. In special cases *ad hoc* representations reduce this space and some of these improvements will be noted.

Definition. *A matrix,* $[a_{i,j}]$ *is zero if* $a_{i,j} = 0$ *for all* i *and* j.
A matrix, $[a_{i,j}]$ *is packed if* $a_{i,j} \neq 0$ *for all* i *and* j.
It is symmetric if $a_{i,j} = a_{j,i}$ *for all* i *and* j.
It is lower (upper) triangular if $a_{i,j} = 0$ *for* $i < j$ *(respectively* $i > j$*).*
It is diagonal if $a_{i,j} = 0$ *for* $i \neq j$.
It is Hankel if $a_{i+1,j} = a_{i,j+1}$ *for all* $i, j > 0$.

Whenever an element is not constrained to be zero in these definitions, we require it to be non-zero in order to attain the worst-case measures. That is, any unconstrained region of a matrix is assumed to be packed with (pairwise) different values, so that there is no sharing of common quadrants.

Hankel matrices are quite similar to Toeplitz matrices: a Toeplitz matrix has $a_{i+1,j+1} = a_{i,j}$ for all $i, j > 0$; so the pattern of sharing across quadrants is the same. Table 2 lists type indices for various patterns of matrices, not all yet defined, that are used as subscripts below.

Definition. *Let* $n = 2^p$, *for* p *an integer and let* t *be a type index* D, F, HT, P, R, S, SD, T, *or* Z *from Table 2. The function* S_t *maps* p *to the number of nodes (worst case space) necessary to represent a* $n \times n$ *matrix of type* t. *The function* P_t *maps* p *to the worst case expected path length in a* $n \times n$ *matrix of type* t.

Thus, S_P and P_P specify space and expected path length for packed matrices and S_S maps p to the space necessary to represent a symmetric $n \times n$ matrix, and P_S maps p to the expected *path* length in a symmetric $n \times n$ matrix.

Index	Matrix Type	Index	Matrix Type
B	Banded	R	Random Permutation-pattern
C	Clip	SD	Shuffle/Deal Permutation-pattern
D	Diagonal	S	Symmetric
F	fFT (bit reversal) Permutation-pattern	T	Triangular
HT	Hankel/Toeplitz	Z	Zero
P	Packed		

Table 2. Type indices identifying patterned matrices.

Theorem 2.

$$\begin{aligned} S_Z(p) &= 0; & P_Z(p) &= 0; \\ S_P(p) &= \tfrac{4}{3}(4^p - \tfrac{1}{4}); & P_P(p) &= p+1; \\ S_S(p) &= \tfrac{2}{3}(4^p - \tfrac{1}{4}) + 2^p; & P_S(p) &= p+1; \\ S_T(p) &= \tfrac{2}{3}(4^p - \tfrac{1}{4}) + 2^p; & P_T(p) &= \tfrac{1}{2}(p+3-2^{-p}); \\ S_D(p) &= 2^{p+1} - 1; & P_D(p) &= 2 - 2^{-p}; \\ S_{HT}(p) &= 2^{p+2} - p - 3; & P_{HT}(p) &= p+1. \end{aligned}$$

Proof. Zero matrices are a trivial exception; *NIL* takes no space and contributes nothing to path length. The other values are derived from recurrence equations. The space for any of the other matrices of order one is just 1, for the non-zero scalar. Similarly, the expected path lengths are all just 1.

$$S_P(0) = S_S(0) = S_T(0) = S_D(0) = S_{HT}(0) = 1;$$

$$P_P(0) = P_S(0) = P_T(0) = P_D(0) = P_{HT}(0) = 1.$$

The recurrences for the first three follow Figure 1. The quadtree representation of a packed matrix decomposes into four packed quadrants.

$$\begin{aligned} S_P(p+1) &= 1 + 4S_P(p); \\ P_P(p+1) &= 1 + \frac{4}{4}P_P(p). \end{aligned}$$

Thence,

$$S_P(p) = \sum_{i=0}^{p} 4^i = \frac{4^{p+1} - 1}{3}$$
$$P_P(p) = p + 1.$$

The representation of a symmetric matrix appears much like a packed matrix; indeed their expected path lengths are the same since both "trees" seem to be complete upon traversal. The difference is that a symmetric matrix can make use of the tagged quadtree representation, sharing southwest quadrants, being a directed acyclic graph (*dag*) rather than a tree. That is, its northeast quadrant is really a transposed reference to its southwest, which—although packed—contributes to the space but once. Its northwest and southeast quadrants are just symmetric matrices (Figure 1).

$$P_S(p) = P_P(p) = p + 1;$$
$$S_S(p+1) = 1 + 2S_S(p) + S_P(p) + 0.$$

Thus,

$$S_S(p) = \sum_{i=0}^{p} 2^i + \frac{4^p}{3} \sum_{i=0}^{p-1} \frac{2^i}{4^i} - \frac{1}{3}\sum_{i=0}^{p-1} 2^i = \frac{2}{3}(4^p - 1) + 2^p$$

In the worst case, the quadtree representation of a triangular matrix occupies the same space as that of a symmetric matrix of the same order. The reason is illustrated in Figure 1 for lower triangular matrices; the northeast quadrant is *NIL* and thus adds nothing to the required space. It is analogous to the shared, tagged northeast quadrant of a symmetric matrix which also contributes nothing. The expected path is one for the root of the tree, plus 25% of the expected path length in each of the quadrants, which are respectively triangular, zero, packed, and triangular.

$$S_T(p+1) = 1 + 2S_T(p) + S_P(p) + S_Z(p) = S_S(p+1)$$
$$P_T(p+1) = 1 + \tfrac{2}{4}P_T(p) + \tfrac{1}{4}P_P(p) + \tfrac{1}{4}P_Z(p).$$

Solving this recurrence [7: §1.2.3-16],

$$\begin{aligned} P_T(p) &= \sum_{i=0}^{p} 2^{-i} + \frac{p}{4}\sum_{0}^{p-1} 2^{-i} + \frac{1}{4}\sum_{0}^{b-1} i2^i \\ &= \tfrac{1}{2}(p + 3 - 2^{-p}). \end{aligned}$$

The quadtree representation of a diagonal matrix has two quadrants that are diagonal and two that are *NIL*.

$$\begin{aligned} S_D(p+1) &= 1 + 2S_D(p) + 2S_Z(p); \\ P_D(p+1) &= 1 + \tfrac{2}{4}P_D(p) + \tfrac{2}{4}P_Z(p). \end{aligned}$$

Solving these recurrences establishes the values in the theorem.

The quadtree representation of a packed Hankel matrix shares some features of that for symmetric matrices and that for diagonal matrices. The efficiencies of its representation come from its representation as a dag. Just as for symmetric matrices, the northeast quadrant is a shared (but untransposed) reference to the southwest. The northwest and southeast quadrants are quite similar, heavy with shared references, both internally and to the southwest. In fact, the structure of shared references, which contribute no additional space, is much like the *NIL* pointers in the representation of a diagonal matrix. There is only one column (row) of "unshared" scalar values to be counted.

$$\begin{aligned} S_{HT}(p+1) &= 1 + S_{HT}(p) + 2S_D(p) + 0; \\ P_{HT}(p) &= P_P(p). \end{aligned}$$

Solving these recurrences again establishes the values for S_{HT} and P_{HT} stated in the theorem. ∎

The values from Theorem 2 appear in the second and fourth column of Table 1, expressed in terms of $n = 2^p$. It is useful to compare this space to that for alternative representations. For instance, a $n \times n$ Hankel or Toeplitz matrix can be compressed to a sequential vector of size $2n - 1$; the quadtree representation takes less than twice this space. Similarly, a diagonal matrix can be represented as a vector of size n; again the quadtree representation is under twice as big,

about the size of its representation with all the non-zero entries in each row linked [7] (linked-row). Representing packed matrices as quadtrees costs a 33% space overhead beyond the sequential n^2 space. This overhead applies as well to that for symmetric and triangular matrices, which occupy just over half the space of a packed matrix of the same order, just as they do in sequential storage schemes.

Since the space efficiency is nearly realized without special accessing algorithms, there must be a penalty. The tradeoff is manifest in the path length, which is logarithmic in n for the dense patterns. Nevertheless, the expected path length in a diagonal matrix is under two, and that for a triangular matrix is almost half that of a packed matrix, just as under a compressed sequential scheme (where almost half of the possible pairs of indices can be discharged without any memory access.) Of course, no memory fetches are required to probe the quadtree representation of a zero matrix, *NIL*.

As the purpose of this paper is to obtain analytical measures on sparse matrices, we consider the most familiar family of them: the banded matrices. In order to get tight bounds for banded matrices, however, a new kind of pattern for matrices, the clip matrix, must first be introduced. A "clip" matrix looks similar to a triangular matrix, with the role of the main diagonal replaced by one closer to a corner. See Figure 2; the name, "clip" suggests the clipped corner where lies its dense fill.

Definition. *A matrix, $[a_{i,j}]$ of order n is banded with bandwidth b if $|i-j| > b$ implies $a_{i,j} = 0$*
It is lower (upper) clip with bandwidth b if $n-i+j > b$ (respectively $n+i-j > b$) implies $a_{i,j} = 0$

Any unconstrained portion of the matrix is again presumed to be packed. As before, we take n and b as powers of two: $n = 2^p; b = 2^w$ for non-negative integers p and w ("width"). When $p = w$, then, a banded matrix is packed and a clip matrix is triangular.

Definition. *Let $2^p = n \geq b = 2^w$, for p, w non-negative integers and let t be a type index, B or C, from Table 2. The function S_t maps p to the number of*

nodes (worst case space) necessary to represent a $n \times n$ matrix of type t with bandwidth b. The function P_t maps p to the worst case expected path length in a $n \times n$ matrix of type t with bandwidth b.

Theorem 3.

$$S_C(p,w) = \tfrac{4}{3} + \tfrac{1}{2}\left(\tfrac{2^w}{2^p}\right)^2[\tfrac{1}{3} + w - 2^{-w}]$$
$$P_C(p,w) = p - w + \tfrac{2}{3}(4^w - 1) + 2^w$$
$$S_B(p,w) = \tfrac{4}{3}(2^{p+w+1} + 2^{p-w} - 2^{2w}) + 2[(2^p - 2^w) - (p-w)] - \tfrac{5}{3}$$
$$P_B(p,w) = \tfrac{10}{3} + \tfrac{2^w}{2^p}[2(w-1) + (2^{-p} - 2^{-w}) - \tfrac{2^w}{2^p}(w + \tfrac{1}{3})]$$

Proof. For clip matrices,

$$S_C(p,p) = S_T(p);$$
$$S_C(p+1,w) = 1 + S_C(p,w) + 3S_Z(p).$$
$$P_C(p,p) = P_T(p);$$
$$P_C(p+1,w) = 1 + \tfrac{1}{4}P_C(p,w) + \tfrac{3}{4}P_Z(p).$$

Solving these, we obtain the values stated in the theorem, to be used in the following recurrences for banded matrices.

Banded matrices are composed of clip quadrants and banded quadrants as shown in Figure 2.

$$S_B(p,p) = S_P(p);$$
$$S_B(p+1,w) = 1 + 2S_B(p,w) + 2S_C(p,w).$$
$$P_B(p,p) = P_P(p);$$
$$P_B(p+1,w) = 1 + \tfrac{2}{4}P_B(p,w) + \tfrac{2}{4}P_C(p,w).$$

The exact solutions appear in the statement of the theorem. ∎

Corollary 2. *Let $2^p = n \geq b = 2^w$, for p, w non-negative. The quadtree representation of an worst-case $n \times n$ banded matrix of bandwidth b requires space*

$$\tfrac{4}{3}(2nb - b^2 + \tfrac{n}{b} - 1) + 2[n - b - \lg(\tfrac{n}{b})] - \tfrac{1}{3}$$

and has expected path length

$$\frac{10}{3} + \frac{b}{n}[2(\lg b - 1) + (\frac{1}{n} - \frac{1}{b}) - \frac{b}{n}(\lg b + \frac{1}{3})].$$

Definition. *A banded matrix with bandwidth* 1 *is said to be tridiagonal; with bandwidth* 2, *pentadiagonal; with bandwidth* 3, *heptadiagonal; with bandwidth* 4, *enneadiagonal.*

The space and expected path length for $n \times n$ tri-, penta-, and enneadiagonal matrices can be read from Theorem 3 with $w = 0, 1, 2$, or from Corollary 2.

Corollary 3. *Let* $n = 2^p$. *The quadtree representations of an* $n \times n$ *tri-, penta-, and enneadiagonal matrix (worst-case) occupy space, respectively,* $6n - 2\lg n - 5$, $8n - 2\lg n - 9$, $13n - 2\lg n - 27$; *and have expect path length, respectively,* $\frac{10}{3} - \frac{3}{n} + \frac{2}{3n^2}$, $\frac{10}{3} - \frac{1}{n} - \frac{10}{3n^2}$, $\frac{10}{3} + \frac{7}{n} - \frac{100}{3n^2}$.

Alternatively, one can define bandwidth, b, of the form $b = 2^v - 1$ and rewrite the recurrences on v. This exercise will verify the results above for tridiagonal matrices ($v = 1$), and fills in the following values for heptadiagonal matrices ($v = 2$).

Theorem 4. *Let* $n = 2^p$. *The quadtree representation of an* $n \times n$ *heptadiagonal matrix (worst-case) occupies space* $11n - 2\lg n - 19$ *and has expect path length* $\frac{10}{3} + \frac{5}{n} - \frac{76}{3n^2}$.

Thus, a tridiagonal matrix that occupies $3n$ cells if represented in sequential storage requires just under twice that as a quadtree, although that proportion falls as bandwidth increases. However, expected depth, a reflection of access cost, remains remarkably small over various bandwidths. Compared to a banded matrix stored in a more conventional scheme, for instance the linked-row structure [7], this 10/3 expected path length is encouraging; a pentadiagonal matrix has a longer expected path-length in linked-row representation.

Section 4. Permutation Matrices

Nevertheless, there is something artificial about the space and path length results of the previous section. The patterns of the matrices are so regular that the measures are almost predictable, at least within a coefficient. They do not convince one that the quadtree representation is really useful—only that it behaves well. In an effort to obtain analytic results of extremely sparse matrices, the family of permutation matrices is considered.

Definition. *A permutation-patterned matrix is square and has exactly one non-zero entry in each row and column. A permutation matrix is a permutation-patterned matrix in which each non-zero entry is* 1.

Because any permutation matrix can be represented as a vector of integers (regardless of whether a vector is represented using sequential memory or as a binary tree), it seems wasteful even to consider them expanded in a quadtree matrix representation. They are studied here because they initially seem to be *really* sparse, but turn out to be surprisingly expensive. Permutation matrices, particularly that familiar one associated with the fast Fourier transform (fFT) algorithm, have little patterning of zeroes that would allow a collapse of the quadtree representation. Since, as argued elsewhere [4], patterning is necessary to sparseness, it is interesting to see how these measures of space and path length compare with those of the highly patterned matrices, already presented.

Another reason for obtaining measures on permutation matrices is that permuting the indexing on a quadtree representation is a surprisingly expensive operation. (In contrast, most presentations of Gaussian elimination trivialize that the cost of permuting indices.) Whereas permuting the rows/columns of a sequentially stored matrix is simple (involving an indirect indexing and an increase of one in path length), it can require a severe twist on a quadtree representation. As a result, multiple permutations of quadtree matrices should be avoided, these results teach us to seek algorithms that accumulate repetitious index permutations into a single one, or that avoid them entirely.

Because we are more interested in the patterning of zeroes in the permutation matrices than in any space compression possible from sharing within them, all space analyses ignore the possibility of sharing submatrices. Thus, the space

analyzed is that required for quadtree representation of permutation-*patterned* matrices.

Two permutations, $\mathbf{D}_p$ and $\mathbf{S}_p$ occur naturally in developing the fFT, which are here called *deal* and *shuffle*, respectively [14].

Definition. *Let p be a non-negative integer. A permutation matrix, $[a_{i,j}]$ of order 2^p is $\mathbf{D}_p$ (deal) if $a_{i,j} = 1$ iff $j = 2i - 1$ or $j = 2i - n$.*
$\mathbf{S}_p$ (shuffle) is the transpose of $\mathbf{D}_p$.

The typical layout of $\mathbf{D}_p$ appears below.

$$\mathbf{D}_p = \left(\begin{array}{ccccccc|ccc} 1 & 0 & 0 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \cdots & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & \cdots & 1 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \cdots & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & \cdots & 0 & 1 \end{array}\right)$$

Pease [8] names $\mathbf{S}_p$ as $\mathbf{P}$, without specifying its order. Multiplying a vector of size 2^p by $\mathbf{D}_p$ on the left has the effect of reordering its elements as if the vector had been a deck of cards, which has been dealt into two full hands of $n/2$ cards that were then stacked. Multiplying it on the left by $\mathbf{S}_p$ has the effect of performing a perfect riffle-shuffle on the elements of the vector. Multiplying on the right by these permutations reorders the columns similarly.

The fFT permutation (p-bit-reversal), arises as a direct consequence of Pease's recurrence [8], as shown by Wise [14] who calls it F_p.

Definition. *The fFT permutation (p-bit-reversal) is defined by the recurrence*

$$F_0 = 1;$$
$$F_{p+1} = \left(\begin{array}{c|c} F_p & 0 \\ \hline 0 & F_p \end{array}\right) D_{p+1}.$$

Theorem 5 [14].

$$F_{p+1} = S_{p+1} \left(\begin{array}{c|c} F_p & 0 \\ \hline 0 & F_p \end{array} \right)$$

Recall the definition of the functions S_t and P_t for t in Table 2, given at the beginning of Section 3.

Theorem 6.

$$\begin{aligned} S_{\mathrm{SD}}(p) &= 3(2^p - 1); \\ P_{\mathrm{SD}}(p) &= 3(1 - 2^{-p}); \\ S_F(p) &= p2^{p-1} + \frac{2^{p+2} - 1}{3}; \\ P_F(p) &= \frac{p}{2} + \frac{4 - 2^{-p}}{3}. \end{aligned}$$

Proof. Figure 2 illustrates how $\mathbf{D}_{p+2}$ can be decomposed at Level 2 into the quadrants of $\mathbf{D}_{p+1}$ by duplicating and restructuring the quadrants of the latter. The top *two* levels of the tree are rearranged there and counted explicitly in the following two recurrences:

$$\begin{aligned} S_{\mathrm{SD}}(0) &= 1; \\ S_{\mathrm{SD}}(1) &= 3; \\ S_{\mathrm{SD}}(p+2) &= 5 + 2(S_{\mathrm{SD}}(p+1) - 1). \\ P_{\mathrm{SD}}(0) &= 1; \\ P_{\mathrm{SD}}(1) &= \tfrac{3}{2}; \\ P_{\mathrm{SD}}(p+2) &= 2 + \tfrac{8}{16}(P_{\mathrm{SD}}(p+1) - 1) + \tfrac{8}{16}P_Z(p). \end{aligned}$$

Solving these recurrences, we find that for $p > 0$

$$\begin{aligned} S_{\mathrm{SD}}(p) &= 3\Big(\sum_{i=0}^{p-3} 2^i\Big) + 2^{p-2}S_{\mathrm{SD}}(2) = 3(2^p - 1); \\ P_{\mathrm{SD}}(p) &= 3(1 - 2^{-p}). \end{aligned}$$

F_p is well known as the "bit-reversal" permutation because it exchanges x_i and $x_{b(i)}$ in permuting $\vec{x}$, where b is a function on natural numbers less than 2^p that reverses the p-bit strings that represent them [8]. That is, the element of $\vec{x}$

indexed $i = \sum_{j=0}^{n} b_j \cdot 2^j$ is indexed $\sum_{j=0}^{n} b_{n-j} \cdot 2^j$ after that permutation. (This fact, as well as Theorem 5, can be established by a simple induction on p.)

Figure 3 illustrates the quadtree representation of F_{2k}, the $2^k \times 2^k$ fFT permutation matrix, with its 4^{2k} entries gridded as a $2^k \times 2^k$ matrix of blocks, each of which is a $2^k \times 2^k$ matrix. Each of those blocks has *exactly* one element that is non-zero. This gridding can be inferred from the bit-reversal function, b; if one considers the grid block of F_{2k} indexed by $i, j \leq 2^k$ and the bitwise concatenation of i and j, $c = i2^k + j$, then the only non-zero entry in that grid block is in its position indexed $\langle c, b(c) \rangle$. That is, if $0 < i, j \leq 2^k$ and the $\langle i2^k + b(q),\ j2^k + b(r) \rangle^{\text{th}}$ element of F_{2k} is 1, then $i = r$ and $j = q$.

Redrawing the above block decomposition as a tree, also in Figure 3, we find that the quaternary tree is complete for the first k levels, down to the 4^k *intermediate* nodes that root these grid blocks. The trees rooted there are metalinear; that is, each node has at most one son, along the path toward the unique 1 entry. All other pointers in those subtrees are *NIL*.

Thus,

$$\begin{aligned} S_F(0) &= P_F(0) = 1; \\ S_F(1) &= 3; \qquad P_F(1) = 3/2; \\ S_F(p+2) &= 1 + 4S_F(p) + 2^{p+2}. \\ P_F(p+2) &= 1 + P_F(p) + \frac{2^{p+2}}{4^{p+2}}. \end{aligned}$$

The last recurrence arises by adding a root above four copies of the quadtree representation for F_p, and extending each metalinear chain by one level at the bottom. Solving these two recurrences yields the worst-case results stated in the theorem. ∎

If sharing were allowed—that is if we consider permutation matrices, rather than that permutation-patterned ones—then $S_{\text{SD}}(p)$ collapses to $4p - 2$ and *all* $2^i \times 2^i$ shuffle/deal permutation matrices for integers $i \leq p$ can be represented in shared space $5p - 3$. Similarly, because the bit-reversal function, b, is its own inverse, F_p is also, and so a symmetric permutation matrix; therefore, the space measure, S_F, can be halved by sharing of transposed quadrants.

Theorem 6 shows that the space needed for an $n \times n$ fFT permutation matrix grows with $(n \lg n)$; this is significantly more than the linear space we obtain with the vector-of-indices representation. Also, the average path length is surprisingly poor, essentially $\frac{\lg n}{2}$, reflecting the completeness of the tree to that level, but already more than half that of a packed matrix. Theorem 7 should then be no surprise, because it shows that F_p measures to be the worst of the permutation-patterned matrices.

Theorem 7. *The quadtree representation of an fFT permutation-patterned matrix, F_p, realizes the worst case space and expected-path-length measures of any $2^p \times 2^p$ matrix with only 2^p non-zero entries.*

Proof: Let $n = 2^p$ and, for $0 \leq m \leq 4^p$, define $S(p,m)$ to be the worst-case *space* for an $n \times n$ matrix with m non-zero entries, and $P(p,m)$ to be the worst expected *path length* for an $n \times n$ matrix with m non-zero entries.

Lemma 1. *$S(p,0) = 0 = P(p,0)$ and, for $0 < m \leq 4^p$,*

$$S(p,m) \leq mp - m\log_4 m + \tfrac{4}{3}m - \tfrac{1}{3};$$
$$P(p,m) \leq \log_4 m + \tfrac{1}{3}(4 - m4^{-p}).$$

Lemma 1 is proved in the Appendix. When $m = 2^p$, it gives the following upper bounds:

$$S(p,2^p) \leq 2^p \cdot p - 2^p \cdot \tfrac{p}{2} + \tfrac{4}{3}2^p - \tfrac{1}{3} = p2^{p-1} + \tfrac{2^{p+2}-1}{3};$$
$$P(p,2^p) \leq \tfrac{p}{2} + \tfrac{1}{3}(4 - 2^p \cdot 4^{-p}) = \tfrac{p}{2} + \tfrac{4-2^{-p}}{3}.$$

However, Theorem 6 shows that these upper bounds are, in fact, *realized* by fFT permutation matrices. So they measure the worst of all permutation matrices. ∎

Now we consider tight bounds for the average time and space required by quadtree representation of random permutation-patterned matrices. The results show that a random permutation matrix requires about as much resource as the worst case fFT permutation matrix. Thus, the surprisingly poor resource characteristics of the fFT permutation matrix are also typical for randomly chosen permutation matrices.

Definition. *Let $n = 2^p$, for p an integer. The function S_R maps p to the average number of nodes necessary to represent a random $n \times n$ permutation-patterned matrix. The function P_R maps p to the expected path length in a random $n \times n$ permutation-patterned matrix.*

Theorem 8. *If p is even*

$$S_R(p) \leq \frac{2^p}{1+2^{-p}}\left(\frac{p}{2} + \frac{2^{-p/2}}{1+2^{-p}} + \frac{2 \cdot 2^{-p}}{3 \cdot (1+2^{-p})^2}\right) + 2^p\left(\frac{4}{3} - e^{-\frac{2^{p/2}}{2^{p/2}-2}}\right) - \frac{4}{3}$$

and

$$S_R(p) \geq 2^p\left(\frac{p}{2} + .78\right) - \frac{4}{3}.$$

If p is odd

$$S_R(p) \leq \frac{2^p}{1+2^{-p}}\left(\frac{p}{2} + \frac{1}{2} + \frac{2^{-p/2+1/2}}{1+2^{-p}} + \frac{4 \cdot 2^{-p}}{3 \cdot (1+2^{-p})^2}\right) + 2^p\left(\frac{2}{3} - \frac{1}{2}e^{-\frac{2^{(p+1)/2}}{2^{(p+1)/2}-2}}\right) - \frac{4}{3}$$

and

$$S_R(p) \geq 2^p\left(\frac{p}{2} + .768\right) - \frac{4}{3}.$$

Proof is in the Appendix.

Corollary 4. *For n a large power of 2, the average space required for $n \times n$ permutation matrices is between $(n \lg n)/2 + .768n - 4/3$ and $(n \lg n)/2 + .983n - 4/3$.*

Proof: From Theorem 8 the average space gets trapped between the lower bound of
$2^p(p/2 + .768) - 4/3$ or $2^p(p/2 + .783) - 4/3$ and the upper limit of $2^p(p/2 + 4/3 - e^{-1}) - 4/3$ or $2^p(p/2 + 7/6 - e^{-1}/2) - 4/3$. The result follows. ∎

Theorem 9. *If p is even, then*

$$P_R(p) \leq \frac{p}{2} - e^{-\frac{2^{p/2}}{2^{p/2}-2}} + \frac{1}{1+2^{-p}}\left(\frac{1}{3} + \frac{2}{7}\frac{2^{-p}}{1+2^{-p}}\right) + 1$$

and

$$P_R(p) \geq \frac{p}{2} + .862 - \frac{2^{-p}}{3}.$$

If p is odd, then

$$P_R(p) \leq \frac{p}{2} + \frac{1}{2} - e^{-\frac{2^{(p+1)/2}}{2^{(p-1)/2}-2}} + \frac{1}{1+2^{-p}}\left(\frac{2}{3} + \frac{2^{-\frac{p-5}{2}}}{7(1+2^{-p})}\right)$$

and

$$P_R(p) \geq \frac{p}{2} + .764 - \frac{2^{-p}}{3}.$$

Proof is in the Appendix.

Corollary 5. *For n a large power of 2 the average path length in an $n \times n$ permutation matrices is between $\frac{1}{2}\lg(n) + .763$ and $\frac{1}{2}\lg(n) + .966$.*

Proof: From Theorem 9 the average path length tends to between a lower bound of $p/2 + .862$ or $p/2 + .763$ and an upper bound of $p/2 + 4/3 - e^{-1}$ or and $p/2 + 7/6 - e^{-1}$. The result follows from $n = 2^p$. ∎

The bounds in both Corollaries 4 and 5 are tight with respect to their leading coefficients of $\frac{1}{2}$, coinciding with those from the fFT permutation. Their second coefficients lie between 55% and 75% of the corresponding coefficients, $\frac{4}{3}$, from the fFT permutation. While looser, the two sets of bounds for those second coefficients bracket the same range.

Section 5. Measures of Sparsity and Density

Duff states in his authoritative survey, "In quantitative terms, the density of a matrix is defined as the percentage of the number of nonzeros to the total number of entries in the matrix. The term sparsity for the complement of this quantity is rarely used. [4, p. 500]" Rather, he suggests that sparsity of a matrix has as much to do with the distribution of zero elements as with their relative population.

This section presents an alternative definition of *sparsity* [14] that contrasts with density, but one which does reflect patterning of zero entries, as well as a plethora of them. Furthermore, a theorem ties the measure of sparsity to time bounds on matrix addition, just as density is already tied to space bounds on the matrix sum. Both these measures are easily computed and are scaled between zero and one.

We already have closed-form results for worst-case total space and expected depth from various patterned and permutation matrices, as summarized in Table 1. Based on Duff's *caveat* and these numbers, we propose measures for both density and sparsity. Although they were motivated by the results on quadtrees, they are defined *independently of any particular matrix representation.* Values for the quadtree matrix representation are included with the cases in Table 1 .

Definition. *Density of a matrix is the ratio between the space it occupies, and the space occupied by a packed matrix of the same order. Non-sparsity of a matrix is the ratio between the expected time to access a random element and the expected access time within a packed matrix of the same order. Sparsity is the difference between one and the non-sparsity ratio.*

Both density and sparsity are measured on a scale from zero to one, and we shall see that they are not complementary. For the conventional row-major, sequential representation of matrices, however, the density measure corresponds precisely with Duff's [4], and constant access time yields zero sparsity. So they are inversely related on that common model.

When rows (or columns) are linked according to a bit-map, however, non-sparsity becomes half the expected row (respectively, column) length. Triangular matrices, for instance, measure 1/2 in both sparsity and density under this representation, but still complementary; diagonal matrices also measure complementary: density $2/n$ and density $1 - 2/n$. Measures of common patterns under other representations are left as problems. Complementary density and sparsity measures (that is, equal density and non-sparsity ratios) seem to be common; is this equivalence an indicator for good sparse representations?

In reviewing the results for these measures on quadtrees, once again we restrict the order of matrices to be powers of two (or four). For the quadtree representation of $2^p \times 2^p$ Type t matrices (t according to Table 2) density is given by the ratio $S_t(p)/S_P(p)$ and sparsity is $1 - P_t(p)/P_P(p)$. Table 1 presents results for space, density, expected path length (root to terminal node), and sparsity for $n \times n$ matrices of the types in Table 2.

The remarkable lines in Table 1 are those for symmetric matrices, Hankel matrices, and for random and fFT permutation-patterned matrices; they measure out to be neither dense nor sparse. Quadtree representations of Toeplitz/Hankel matrices measure out to almost 0 in both density and sparsity. This result is due to its heavy sharing of quadrants in a dag, rather than a tree, structure. Similarly, symmetric matrices measure to be only 50% dense, but only because the tagged quadtree allows heavy sharing, like a dag. In both cases their sparsity suggests that a traversal will find them full, unless the traversing algorithm can take advantage of reference equality to attenuate its work.

These permutation-patterned matrices have sparsity and density measures that only sum to slightly over $1/2$, in contrast with their n non-zeroes of n^2 entries; aren't they sparse? This measure, however, is consistent with Duff's observation that patterning is essential to sparseness; both the FFT (bit-reversal) and random permutations are characterized by lack of local patterning.

Also interesting is that triangular and random permutation matrices strangely share the same sparsity, about $1/2$, while the other sparsities are close to 0 or nearly 1. These are strange kin, because their trees are of radically different shapes.

The utility of these measures must be demonstrated in analytical or in experimental studies of the behavior of algorithms on arguments with various measures. An example worst-case analysis for the matrix addition algorithm follows.

Theorem 10. *The quadtree sum of two $n \times n$ matrices, respectively of density d_1 and d_2 and of sparsity s_1 and s_2, requires uniprocessor (proportional) time and space within the bounds:*

$$n^2(\lg n+1)\max[0, 1-(s_1+s_2)] \leq \textbf{uniprocessor time} \leq n^2(\lg n+1)[1-\max(s_1, s_2))],$$

$$\tfrac{4}{3}n^2|d_1 - d_2| \leq \textbf{space}_{\textbf{sum}} < \tfrac{4}{3}n^2 \min(1, d_1 + d_2);$$

and, itself has sparsity and density measures within the bounds,

$$\max(0, s_1 + s_2 - 1) \leq \textbf{sparsity}_{\textbf{sum}} \leq 1 - |s_1 - s_2|;$$

$$|d_1 - d_2| \leq \mathbf{density}_{\mathbf{sum}} < \min(1, d_1 + d_2).$$

Proof: These results follow from the following observations. The sum will be, at worst, packed:

$$\mathbf{space}_{\mathbf{sum}} \leq S_P(\lg n) \leq \tfrac{4}{3}n^2 - \tfrac{1}{3} < \tfrac{4}{3}n^2,$$

and is no larger than the sum of the space occupied by the addends:

$$\mathbf{space}_{\mathbf{sum}} \leq space_1 + space_2 \leq S_P(\lg n)(d_1 + d_2).$$

If one addend is the negative of the other, then the space for the sum, *NIL*, is zero;

$$\mathbf{space}_{\mathbf{sum}} \geq |space_1 - space_2| = 0 = \tfrac{4}{3}n^2|d_1 - d_2|;$$

otherwise the sum-tree must, at least contain a root:

$$\begin{aligned}\mathbf{space}_{\mathbf{sum}} &\geq |space_1 - space_2| + 1 \\ &\geq \tfrac{4}{3}(n^2 - \tfrac{1}{4})|d_1 - d_2| + 1 > \tfrac{4}{3}n^2|d_1 - d_2|\end{aligned}$$

Let (proportional) time be measured by the number of nodes visited in computing the sum quadtree; only the portions near the roots of both trees, common to both addends are traversed during summation, because unshared periphery are merely borrowed in the sum. The number of nodes visited during an addition is, at best, zero when both addends are *NIL*. That number is otherwise proportional to the path common to the two addends:

$$\begin{aligned}\mathbf{uniprocessortime} &\geq 0; \\ &\geq \mathbf{totalpath}_1 + \mathbf{totalpath}_2 - \mathbf{totalpath}_{\mathrm{Packed}}; \\ &\geq n^2(\lg n + 1)\max(0, 1 - (s_1 + s_2)).\end{aligned}$$

It is less than the lesser of the two total paths:

$$\begin{aligned}\mathbf{uniprocessortime} &\leq \min(\mathbf{totalpath}_1, \mathbf{totalpath}_2) \\ &\leq n^2(\lg n + 1)(1 - \max(s_1, s_2)).\end{aligned}$$

Bounds on density follow from dividing the bounds from the space analysis, above, by $S_P(\lg n)$. In considering sparsity, however, we must consider the entire, unshared path length within the sum:

$$\begin{aligned}\mathbf{totalpath_{sum}} &\geq 0;\\ &\leq n^2(\lg n + 1);\\ &\geq |\mathbf{totalpath_1} - \mathbf{totalpath_2}|;\\ &\leq \mathbf{totalpath_1} + \mathbf{totalpath_2}.\end{aligned}$$

The first two bounds arise from the extreme cases when the sum is *NIL* or completely packed, respectively. The second two bounds account for the cases where the addends are additive inverses, or have no coincident non-zero elements, respectively. Since, for $i = 1, 2$:

$$\mathbf{totalpath}_i = n^2(\lg n + 1)(1 - s_i),$$

$$\max(0, s_1+s_2-1) = 1-\min(1, 2-(s_1+s_2)) \leq \mathbf{sparsity} = 1-\frac{\mathbf{totalpath}}{n^2(\lg n + 1)} \leq 1-|s_1-s_2|.$$

∎

Theorem 10 shows that the bounds, at least, relate sparsity to time and density to space. In general, however, analytical results like these are difficult and so the utility of these measures ultimately must be established or denied by experimentation on real data [1]. Results on multiplicative operations, or average case results of any sort would help compare the quadtree representation to other representations for matrices.

Section 6. Conclusions

Measures of space and expected depth of quadtree representations of various kinds of "sparse" matrices have been presented. All measures exceed those for conventional representations, in most cases by only a constant factor. Quadtree representation, however, offers a natural facility for process decomposition and effective scheduling, so the increased costs may be recovered on a multiprocessor. Since heap memory is not addressed regularly, the chance of "hot spots" in parallel memory access into a quadtree is eliminated, and available bandwidth to memory will, therefore, be better used.

The exercise of comparing these measures revealed that permutation matrices are remarkably expensive under the quadtree representation. By a simple extrapolation, it was easy to see that permuting indices on quadtree matrices was an expensive operation, corresponding to a multiplication [12] by one of those expensive matrices. This motivated a rule-of-thumb for developing algorithms on quadtrees: permutations should be accumulated or entirely avoided. Fortunately, it is indeed possible to find algorithms (e.g. for Gaussian elimination) that avoided permutations, which were suited for implementation under other representations, as well.

In retrospect, we realize that, even under another representation, repeated permuting of indices is a bad strategy under multiprocessing because sharing the permuted structure across multiple processes requires additional synchronization that is best avoided. (Synchronization is, of course, free on uniprocessors, so it's easy to see why permutations have been taken for granted for such a long time.)

If these measures are a harbinger of difficulty, however, they also point to a likely alternative to heavy use of permutations. It will be desirable to avoid wild permutations (like the fFT) on parallel processors, in favor of alternatives like factoring simpler permutations from partial results. Although the fFT permutation measures out to be bad, it admits a factorization of Shuffle Permutations (or Deals; Theorems 5 and 6), each of which measures out to be comparatively tame. Perhaps permutations can be distributed out from a parallel process to a serial process where they would not create bandwidth-consuming, chaotic access, or perhaps, like the permutations in convolution, these accumulated permutations will simplify or cancel themselves on other postponed permutations.

The original purpose of these measures was to compare the relative costs of different patterns of matrices *within* the quadtree representation. Now that the measures of space and path length have been extended to representation-independent measures for density and sparsity, other questions arise. Do these measures form a basis for comparisons among different representations? That is, can we characterize the general utility of a matrix representation in these terms? For instance, representations yielding sparsity and density whose sum is less than one appear to be efficient; is this always true? Is it possible for their sum to exceed one under some rational representation?

Theorem 10 offers resource bounds for addition under quadtree representation. Bounds for other operations: for matrix inversion and solutions to linear systems, or for matrix multiplication, are needed. Average case analyses, of course, would be even more useful. Of more interest would be measures on algorithms that are independent of a particular representation, say, in terms of density and sparsity. That is, can we find a variant of these definitions for density and sparsity that would be useful measures of important matrix algorithms, independent of any particular representation of matrices?

Acknowledgement: An earlier version of this paper was assembled while the first author was a visitor at Tektronix Labs. Toeplitz matrices were measured at the suggestion of E. Kaltofen.

References

1. S. K. Abdali & D. S. Wise. Experiments with quadtree representation of matrices *Proceedings 1988 Intl. Symp. on Symbolic and Algebraic Computation, Lecture Notes in Computer Science* Berlin, Springer (to appear).

2. S. K. Abdali. & D. D. Saunders. Transitive closure and related semiring properties via eliminants. *Theoret. Comp. Sci.* **40**, 2,3 (1985), 257-274.

3. P. J. Denning Parallel computing and its evolution. *Comm. ACM* **29**, 12 (Dec. 1986), 1163-1167.

4. I. S. Duff. A survey of sparse matrix research. *Proc. IEEE* **65**, 4 (Apr. 1977), 500-535.

5. V. N. Faddeeva. Computational Methods of Linear Algebra. Dover, New York, 1959, §1.4.

6. S. D. Johnson. Storage allocation for list multiprocessing. Technical Rep. 168, Computer Science Dept. Indiana University, Bloomington, Indiana, Mar. 1985.

7. D. E. Knuth. *The Art of Computer Programming* **I**, *Fundamental Algorithms*, 2nd Ed., Addison-Wesley, Reading, MA, 1975, §1.2.3-16, 2.2.6, §2.3; p. 401.

8. M. C. Pease. An adaptation of the fast Fourier transform for parallel processing. *J. ACM* **15**, 2 (Apr. 1968), 252-264.

9. R. Rettberg & R. Thomas. Contention is no obstacle to shared-memory multiprocessing. *Comm. ACM* **29**, 12 (Dec. 1986), 1202-1212.

10. J. R. Rice. *Matrix Computations and Mathematical Software,* McGraw-Hill, New York, 1981, Chapter 4.

11. V. Strassen. Gaussian elimination is not optimal. *Numer. Math.* **13**, 4 (Aug. 1969), 354-356.

12. D. S. Wise. Representing matrices as quadtrees for parallel processors. *Inform. Process. Lett.* **20** (May, 1985), 195-199.

13. D. S. Wise. Parallel decomposition of matrix inversion using quadtrees. In Hwang, K., Jacobs, S. J., and Swartzlander E. E. (Eds.), *Proc. 1986 International Conference on Parallel Processing*, IEEE Computer Society Press, Washington, 1986, pp. 92-99.

14. D. S. Wise. Matrix algebra and applicative programming. In Kahn, G. (Ed.), *Functional Programming Languages and Computer Architecture, Lecture Notes in Computer Science* **274**, Springer, Berlin, 1987, pp. 134-153.

Appendix

$S(p,m)$ is the worst-case *space* for an $n \times n$ matrix with m non-zero entries, and $P(p,m)$ is the worst expected *path length* for an $n \times n$ matrix with m non-zero entries.

Lemma 1. *$S(p,0) = 0 = P(p,0)$ and, for $0 < m \leq 4^p$,*

$$S(p,m) \leq mp - m\log_4 m + \tfrac{4}{3}m - \tfrac{1}{3};$$
$$P(p,m) \leq \log_4 m + \tfrac{1}{3}(4 - m4^{-p}).$$

Proof: The proof is trivial for $m = 0$. For $0 < m \leq 4^p$ the proof follows by simple induction on p:

Basis: For $p = 0$, necessarily $m = 0$ or $m = 1$. The latter case is also easy:

$$S(0,1) = 1 = 1 \cdot 0 - 1 \cdot \log_4 1 + \tfrac{4}{3} \cdot 1 - \tfrac{1}{3};$$

$$P(0,1) = 1 = \log_4 1 + \tfrac{1}{3}(4 - 1 \cdot 4^{-0}).$$

Induction Step: Assume that Lemma 1 holds for $p = q$, and consider $q + 1$.

$$S(q+1,m) = 1 + \max_{\substack{i+j+k+l=m \\ 0 \leq i,j,k,l \leq 4^q}} [S(q,i) + S(q,j) + S(q,k) + S(q,l)];$$
$$P(q+1,m) = 1 + \tfrac{1}{4}\Big\{ \max_{\substack{i+j+k+l=m \\ 0 \leq i,j,k,l \leq 4^q}} [P(q,i) + P(q,j) + P(q,k) + P(q,l)] \Big\}.$$

Establishing the bounds from these equations decomposes into four cases: respectively when four, three, two, or only one of i,j,k,l are non-zero. Only the first and last cases will be expanded here; the cases for three and two are similar.

When all four are non-zero, m must be 4 or larger:

$$\begin{aligned}
S(q+1,m) \leq & 1 + \max_{\substack{i+j+k+l=m \\ 0<i,j,k,l\leq 4^q}} [(i+j+k+l)(q+\tfrac{4}{3}) - \tfrac{4}{3} \\
& \qquad - (i\log_4 i + j\log_4 j + k\log_4 k + l\log_4 l)]; \\
= & 1 + m(q+\tfrac{4}{3}) - \tfrac{4}{3} - \min_{\substack{i+j+k+l=m \\ 0<i,j,k,l\leq 4^q}} [i\log_4 i + j\log_4 j + k\log_4 k + l\log_4 l]; \\
\leq & m(q+\tfrac{4}{3}) - \tfrac{1}{3} - 4[\tfrac{m}{4}\log_4 \tfrac{m}{4}] \\
& \qquad \text{because minimum occurs at } i=j=k=l=\tfrac{m}{4}; \\
= & mq + \tfrac{4}{3}m - \tfrac{1}{3} - m\log_4 m + m \\
= & m(q+1) - m\log_4 m + \tfrac{4}{3}m - \tfrac{1}{3}; \\
P(q+1,m) \leq & 1 + \tfrac{1}{4} \max_{\substack{i+j+k+l=m \\ 0<i,j,k,l\leq 4^q}} [(\log_4 i + \log_4 j + \log_4 k + \log_4 l) \\
& \qquad + 4(\tfrac{4}{3}) - \tfrac{1}{3}(i+j+k+l)4^{-q}]; \\
= & 1 + \tfrac{4}{3} - \tfrac{1}{3}m4^{-(q+1)} + \tfrac{1}{4} \max_{\substack{i+j+k+l=m \\ 0<i,j,k,l\leq 4^q}} [\log_4 i + \log_4 j + \log_4 k + \log_4 l]; \\
\leq & 1 + \tfrac{4}{3} - \tfrac{1}{3}m4^{-(q+1)} + \tfrac{1}{4}[4\log_4 \tfrac{m}{4}] \\
& \qquad \text{because maximum occurs at } i=j=k=l=\tfrac{m}{4}; \\
= & \log_4 m + \tfrac{4}{3} - \tfrac{1}{3}m4^{-(q+1)}.
\end{aligned}$$

When only one is non-zero, $m \geq 1$ and the recurrence is straightforward:

$$\begin{aligned}
S(q+1,m) \leq & 1 + \max_{\substack{i+j+k+l=m \\ 0=i=j=k<l\leq 4^q}} [lq + l\log_4 l + \tfrac{4}{3}l - \tfrac{1}{3}]; \\
= & 1 + mq + m\log_4 m + \tfrac{4}{3}m - \tfrac{1}{3}; \\
\leq & m(q+1) - m\log_4 m + \tfrac{4}{3}m - \tfrac{1}{3}; \\
P(q+1,m) \leq & 1 + \tfrac{1}{4} \max_{\substack{i+j+k+l=m \\ 0=i=j=k<l\leq 4^q}} [\log_4 l + \tfrac{1}{3}(4 - l4^{-q})]; \\
= & 1 + \tfrac{1}{4}[\log_4 m + \tfrac{1}{3}(4 - m4^{-q})]; \\
= & \tfrac{1}{4}\log_4 m + \tfrac{4}{3} - \tfrac{1}{3}m4^{-(q+1)}; \\
\leq & \log_4 m + \tfrac{1}{3}(4 - m4^{-(q+1)}). \quad \square
\end{aligned}$$

Lemma 2. *If $b \geq 1$ and $|a| \leq 1$ then $(1-a)^b \geq 1 - ab$.*

Proof: Compare the Taylor Series expansion of the logarithm of both sides.□

Lemma 3. *If $a \geq 1$ then $(1-x)^a \leq 1 - ax + (ax)^2/2$.*

Proof: Using Taylor's Series expansion (around zero) with remainder [7, p. 150] we can write

$$(1-x)^a = 1 - ax + \frac{a^2(1-c)^a x^2}{2}$$

where $0 \leq c \leq x$. Setting $c = 0$ proves the lemma. □

Lemma 4. *For all $x, y \geq 0$, $(1-x)^y \geq e^{-xy/(1-x)}$.*

Proof: It suffices to show that $-\ln(1-x) \leq x/(1-x)$. But

$$-\ln(1-x) = x + \frac{x^2}{2} + \frac{x^3}{3} + \frac{x^4}{4}...$$

and

$$\frac{x}{1-x} = x + x^2 + x^3 + x^4 + ...$$

Straightforward comparison proves the lemma. □

Lemma 5. *For all $x, y \geq 0$, $(1-x)^{xy} \leq e^{-x^2 y}$.*

Proof: Take the logarithm of both sides, apply the Taylor Series expansion to the left side and compare terms. □

Definition. *$P(p)$ is an $n \times n$ permutation matrix where $n = 2^p$ and p is an integer.*

Definition. *$T_P(p)$ is a complete quadtree representing a $2^p \times 2^p$ packed matrix and $T(p)$ is the quadtree that results from representing a given $2^p \times 2^p$ permutation matrix $P(p)$.*

Theorem 8. *If p is even*

$$S_R(p) \leq \frac{2^p}{1+2^{-p}}\left(\frac{p}{2}+\frac{2^{-p/2}}{1+2^{-p}}+\frac{2\cdot 2^{-p}}{3\cdot(1+2^{-p})^2}\right)+2^p\left(\frac{4}{3}-e^{-\frac{2^{p/2}}{2^{p/2}-2}}\right)-\frac{4}{3}$$

and

$$S_R(p) \geq 2^p\left(\frac{p}{2}+.78\right)-\frac{4}{3}.$$

If p is odd

$$S_R(p) \leq \frac{2^p}{1+2^{-p}}\left(\frac{p}{2}+\frac{1}{2}+\frac{2^{-p/2+1/2}}{1+2^{-p}}+\frac{4\cdot 2^{-p}}{3\cdot(1+2^{-p})^2}\right)+2^p\left(\frac{2}{3}-\frac{1}{2}e^{-\frac{2^{(p+1)/2}}{2^{(p+1)/2}-2}}\right)-\frac{4}{3}$$

and

$$S_R(p) \geq 2^p\left(\frac{p}{2}+.768\right)-\frac{4}{3}.$$

Proof: Only the case where p is even is presented in detail. The case for odd p may be established in similar fashion. Associate with each node in $T_P(p)$ a pair of integers (i,j) where i specifies the level of the node in $T_P(p)$ and j is the left-to-right order of the node at that level. Let $N(i,j)$ be an indicator variable which has value 1 if node (i,j) exists in $T(p)$ and 0 otherwise. The expectation of $N(i,j)$ is the probability that node (i,j) is in $T(p)$. Then,

$$S_R(p) = \sum_{i,j} E(N(i,j)) = \sum_{i,j} pr(\text{node } (i,j) \text{ is in } T(p)). \tag{1}$$

But node (i,j) is in $T(p)$ if and only if the submatrix corresponding to that node is not zero. Let $pr(p,i)$ denote the probability that the submatrix corresponding to a node at level i is zero. Therefore, the probability that node (i,j) is in $T(p)$ is $1-pr(p,i)$.

Figure 4 illustrates the calculation of $pr(p,i)$. The $2^{p-i}\times 2^{p-i}$ submatrix corresponding to node (i,j) has been placed at the lower right corner of matrix P for simplicity; this matrix must be entirely zero. In order for P to be a permutation matrix exactly 2^{p-i} rows of submatrix B must contain 1's. This requires

2^{p-i} columns of submatrix A to be all zero. There are $\binom{2^p-2^{p-i}}{2^{p-i}}$ ways to choose 2^{p-i} zero columns in A. For each way to choose 2^{p-i} zero columns in A there are $(2^p - 2^{p-i})!$ ways to place 1's in submatrices A and C such that no two are in the same row or column. Finally, for each way to place 1's in A and C there are $2^{p-i}!$ ways to place 1's in B (these must be placed in the zero columns of A). Thus, for a node at level i,

$$pr(p,i) = \frac{\binom{2^p-2^{p-i}}{2^{p-i}} 2^{p-i}!(2^p - 2^{p-i})!}{2^p!} = \frac{(2^p - 2^{p-i})!(2^p - 2^{p-i})!}{(2^p - 2 \cdot 2^{p-i})!2^p!}$$

$$= \frac{(2^p - 2^{p-i} - 0)}{(2^p - 0)} \cdot \frac{(2^p - 2^{p-i} - 1)}{(2^p - 1)} \cdot \frac{(2^p - 2^{p-i} - 2)}{(2^p - 2)} \cdots \frac{(2^p - 2^{p-i+1} + 1)}{(2^p - 2^{p-i} + 1)}$$

$$= \prod_{k=0}^{2^{p-i}-1} \left(1 - \frac{2^{p-i}}{2^p - k}\right). \tag{2}$$

Using (2), the fact that the number of nodes on Level i is 4^i, and that the node on Level 0 is always present, rewrite (1) as follows:

$$S_R(p) = 1 + \sum_{i=1}^{p} 4^i \left(1 - \prod_{k=0}^{2^{p-i}-1} \left(1 - \frac{2^{p-i}}{2^p - k}\right)\right). \tag{3}$$

Now split the sum in (3) into two subsums by cleaving the range of k and derive upper and lower bounds for each. These will later be added to establish the theorem.

a. Consider the sum

$$Upper = \sum_{i=p/2+1}^{p} 4^i \left(1 - \prod_{k=0}^{2^{p-i}-1} \left(1 - \frac{2^{p-i}}{2^p - k}\right)\right)$$

$$\leq \sum_{i=p/2+1}^{p} 4^i \left(1 - \left(1 - \frac{2^{p-i}}{2^p - 2^{p-i} + 1}\right)^{2^{p-i}}\right)$$

From Lemma 2, $2^{p-i} \geq 1$ implies

$$\left(1 - \frac{2^{p-i}}{2^p - 2^{p-i} + 1}\right)^{2^{p-i}} \geq 1 - \frac{2^{2(p-i)}}{2^p - 2^{p-i} + 1}.$$

Therefore, we can bound *Upper* from above as follows:

$$Upper \leq \sum_{i=p/2+1}^{p} 4^i \left(\frac{4^{p-i}}{2^p - 2^{p-i} + 1} \right) = 2^p \sum_{i=p/2+1}^{p} \frac{1}{1 - 2^{-i} + 2^{-p}}$$

$$= \frac{2^p}{1 + 2^{-p}} \sum_{i=p/2+1}^{p} \frac{1}{1 - \frac{2^{-i}}{1+2^{-p}}}.$$

Making use of the fact that $1/(1-x) \leq 1 + x + 2x^2$ if $0 \leq x \leq 1/2$ we can write

$$Upper \leq \frac{2^p}{1 + 2^{-p}} \sum_{i=p/2+1}^{p} \left(1 + \frac{2^{-i}}{1 + 2^{-p}} + \frac{2 \cdot 2^{-2i}}{(1 + 2^{-p})^2} \right)$$

$$\leq \frac{2^p}{1 + 2^{-p}} \left(\frac{p}{2} + \frac{2^{-p/2}}{1 + 2^{-p}} + \frac{2 \cdot 2^{-p}}{3 \cdot (1 + 2^{-p})^2} \right).$$

Upper is bounded from below by

$$Upper \geq \sum_{i=p/2+1}^{p} 4^i \left(1 - \left(1 - \frac{2^{p-i}}{2^p} \right)^{2^{p-i}} \right).$$

From Lemma 3, $2^{p-i} \geq 1$ implies

$$\left(1 - \frac{2^{p-i}}{2^p} \right)^{2^{p-i}} \leq 1 - \frac{2^{2(p-i)}}{2^p} + \frac{2^{4(p-i)}}{2^{2p+1}}.$$

Therefore,

$$Upper \geq \sum_{i=p/2+1}^{p} 4^i \left(2^{p-2i} - 2^{2p-4i-1} \right) \geq \frac{p 2^p}{2} - \frac{2^p}{6}.$$

b. Now consider the sum

$$Lower = \sum_{i=1}^{p/2} 4^i \left(1 - \prod_{k=0}^{2^{p-i}-1} \left(1 - \frac{2^{p-i}}{2^p - k} \right) \right)$$

$$\leq \sum_{i=1}^{p/2} 4^i \left(1 - \left(1 - \frac{2^{p-i}}{2^p - 2^{p-i} + 1} \right)^{2^{p-i}} \right)$$

$$\leq \sum_{i=1}^{p/2} 4^i \left(1 - \left(1 - \frac{1}{2^i - 1} \right)^{2^{p-i}} \right)$$

Lemma 4 asserts that $(1-x)^y \geq e^{-xy/(1-x)}$ for all positive x and y. Therefore,

$$\begin{aligned} Lower &\leq \sum_{i=1}^{p/2} 4^i - \sum_{i=1}^{p/2-1} 4^i \left(1 - \frac{1}{2^i - 1}\right)^{2^{p-i}} - 2^p \left(e^{-\frac{2^{p/2}}{2^{p/2}-2}}\right) \\ &\leq \frac{2^{p+2}}{3} - \frac{4}{3} - 2^p \left(e^{-\frac{2^{p/2}}{2^{p/2}-2}}\right). \end{aligned}$$

Lower is bounded from below by

$$\begin{aligned} Lower &\geq \sum_{i=1}^{p/2} 4^i \left(1 - \left(1 - \frac{2^{p-i}}{2^p}\right)^{2^{p-i}}\right) \\ &= \sum_{i=1}^{p/2} 4^i \left(1 - \left(1 - \frac{1}{2^i}\right)^{2^{p-i}}\right). \end{aligned}$$

Lemma 5 asserts that $(1-x)^{yx} \leq e^{-x^2 y}$ for all positive x and y. So

$$\begin{aligned} Lower &\geq \sum_{i=1}^{p/2} 4^i - 2^p \left(e^{-1} + \frac{1}{4}e^{-4} + \frac{1}{16}e^{-16}\right) - \sum_{i=1}^{p/2-3} 4^i e^{-16} \\ &\geq \frac{2^{p+2}}{3} - \frac{4}{3} - .38 \cdot 2^p. \end{aligned}$$

Putting the bounds of Cases *a* and *b* together proves the theorem. ∎

Theorem 9. *If p is even, then*

$$P_R(p) \leq \frac{p}{2} - e^{-\frac{2^{p/2}}{2^{p/2}-2}} + \frac{1}{1+2^{-p}}\left(\frac{1}{3} + \frac{2}{7}\frac{2^{-p}}{1+2^{-p}}\right) + 1$$

and

$$P_R(p) \geq \frac{p}{2} + .862 - \frac{2^{-p}}{3}.$$

If p is odd, then

$$P_R(p) \leq \frac{p}{2} + \frac{1}{2} - e^{-\frac{2^{(p+1)/2}}{2^{(p-1)/2}-2}} + \frac{1}{1+2^{-p}}\left(\frac{2}{3} + \frac{2^{-\frac{p-5}{2}}}{7(1+2^{-p})}\right)$$

and

$$P_R(p) \geq \frac{p}{2} + .764 - \frac{2^{-p}}{3}.$$

Proof: Again only the case for even p is extended here; the case in which p is odd is similar. A path contains one node from each of levels 0,1,2... in $T_P(p)$ until a node at some level, say i, representing an all zero submatrix is reached. Therefore,

$$P_R(p) = \sum_{i=0}^{p} pr(\text{random path is at least } i \text{ in length}).$$

But, a path is at least i in length if and only if the level i submatrix is not zero. The probability that the level i submatrix is zero was found in the previous proof. Using that probability we have

$$P_R(p) = 1 + \sum_{i=1}^{p} \left(1 - \prod_{k=0}^{2^{p-i}-1} \left(1 - \frac{2^{p-i}}{2^p - k} \right) \right). \tag{4}$$

As before, break the sum of (4) into two subsums so that the upper limit of the first subsum is $p/2$ and the lower limit of the second subsum is $p/2+1$ and find bounds for each subsum. Proceeding as in the proof of Theorem 2, we find that the second subsum is bounded from above by

$$\begin{aligned} Upper &\leq \frac{2^p}{1+2^{-p}} \sum_{i=p/2}^{p} \frac{4^{-i}}{1 - \frac{2^{-i}}{1+2^{-p}}} \\ &\leq \frac{2^p}{1+2^{-p}} \sum_{i=p/2+1}^{p} 4^{-i} \left(1 + \frac{2 \cdot 2^{-i}}{1+2^{-p}} \right) \\ &\leq \frac{2^p}{1+2^{-p}} \left(\frac{1}{3} 2^{-p} + \frac{2}{7} \left(\frac{4^{-p}}{1+2^{-p}} \right) \right). \end{aligned}$$

The second subsum is bounded from below by

$$\begin{aligned} Upper &\geq \sum_{i=p/2+1}^{p} \left(2^{p-2i} - 2^{2p-4i} \right) = 2^p \sum_{i=p/2+1}^{p} 4^{-i} - 4^p \sum_{i=p/2+1}^{p} 16^{-i} \\ &\geq \frac{1}{3} - \frac{1}{3 \cdot 2^p} - \frac{1}{15}. \end{aligned}$$

The first subsum is bounded from above:

$$Lower \leq \sum_{i=1}^{p/2} \left(1 - \left(1 - \frac{1}{2^i}\right)^{2^{p-i}}\right) \leq \sum_{i=1}^{p/2} \left(1 - e^{-\frac{2^{p-i}}{2^i - 2}}\right) \leq \frac{p}{2} - e^{-\frac{2^{p/2}}{2^{p/2}-2}}.$$

The first subsum is bounded from below:

$$Lower \geq \sum_{i=1}^{p/2} \left(1 - e^{-2^{p-2i}}\right) \geq \frac{p}{2} - e^{-1} - 2 \cdot e^{-4}.$$

Assembling all the bounds proves the theorem. ∎

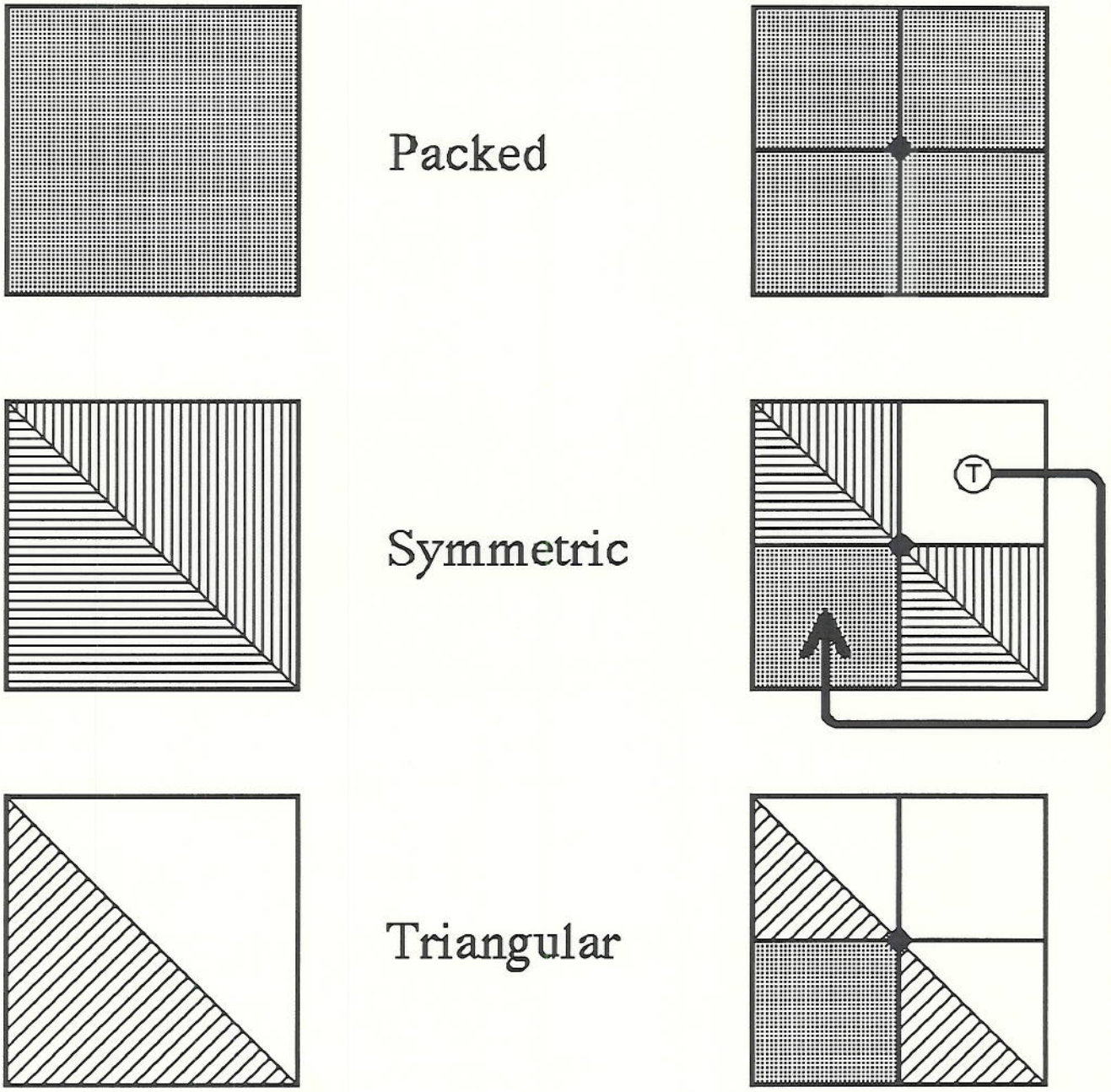

Figure 1. Recursive decomposition of packed, symmetric, and triangular matrices.

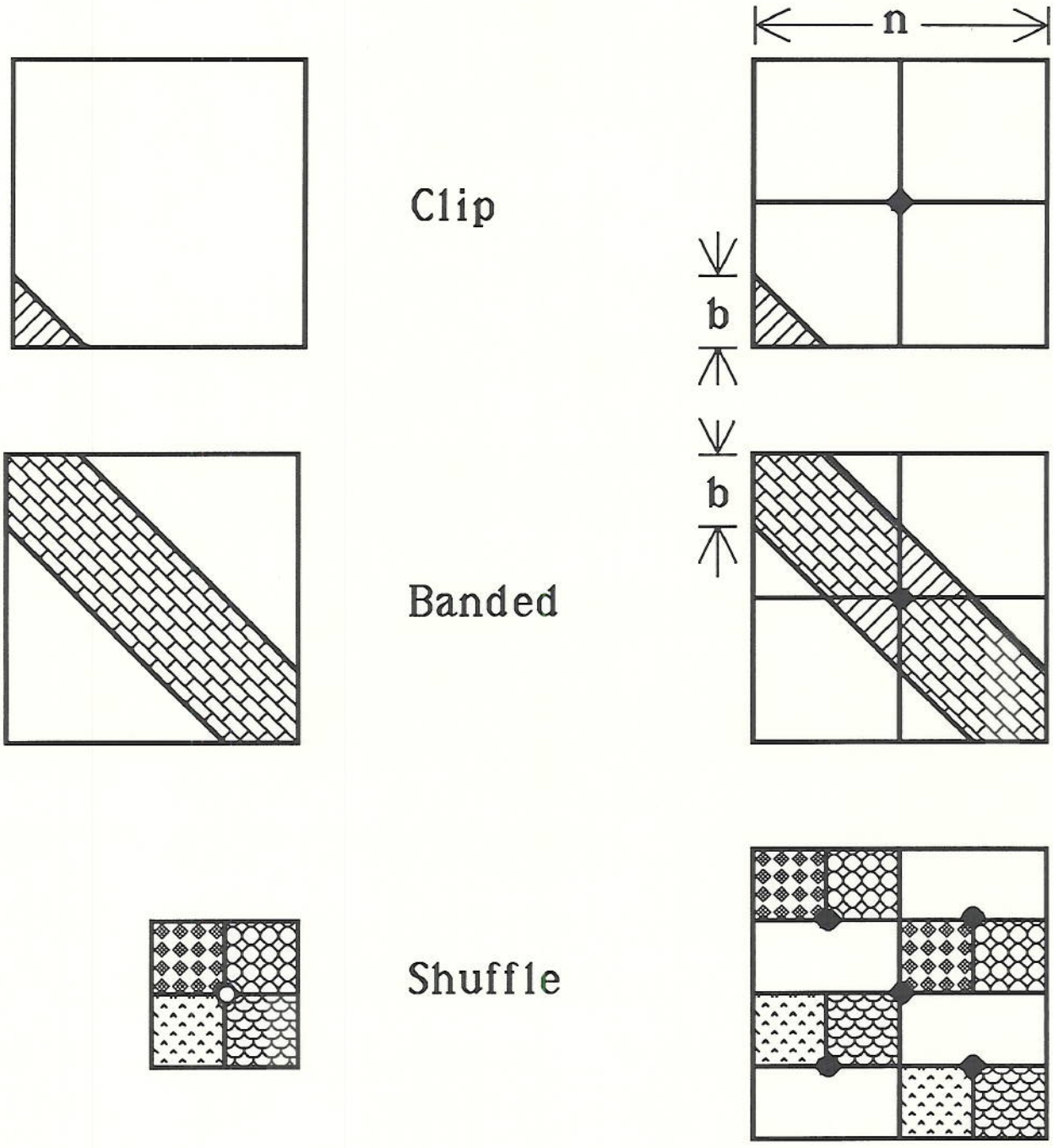

Figure 2.. Recursive decomposition of clip, banded, and shuffle matrices.

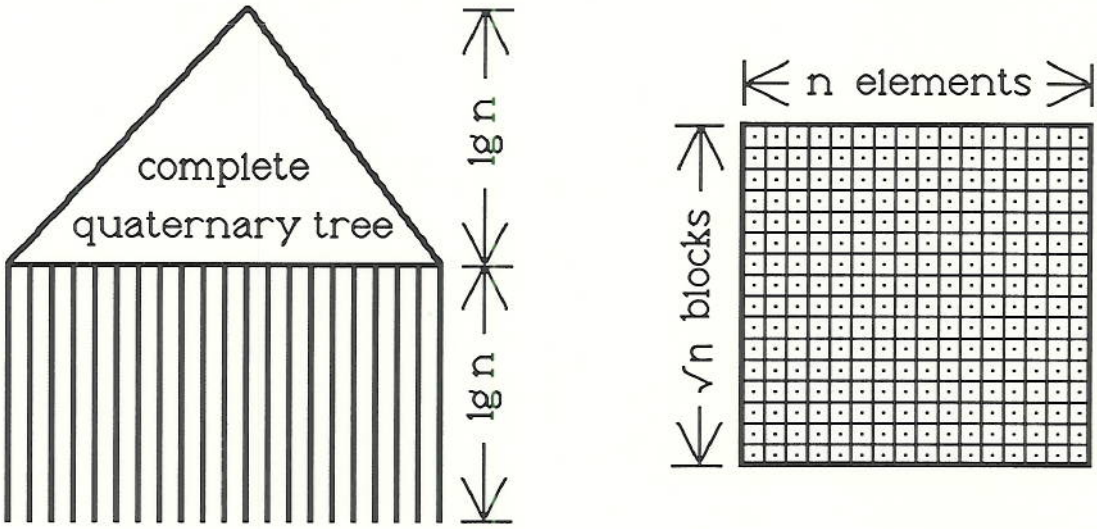

Figure 3. fFT permutation.

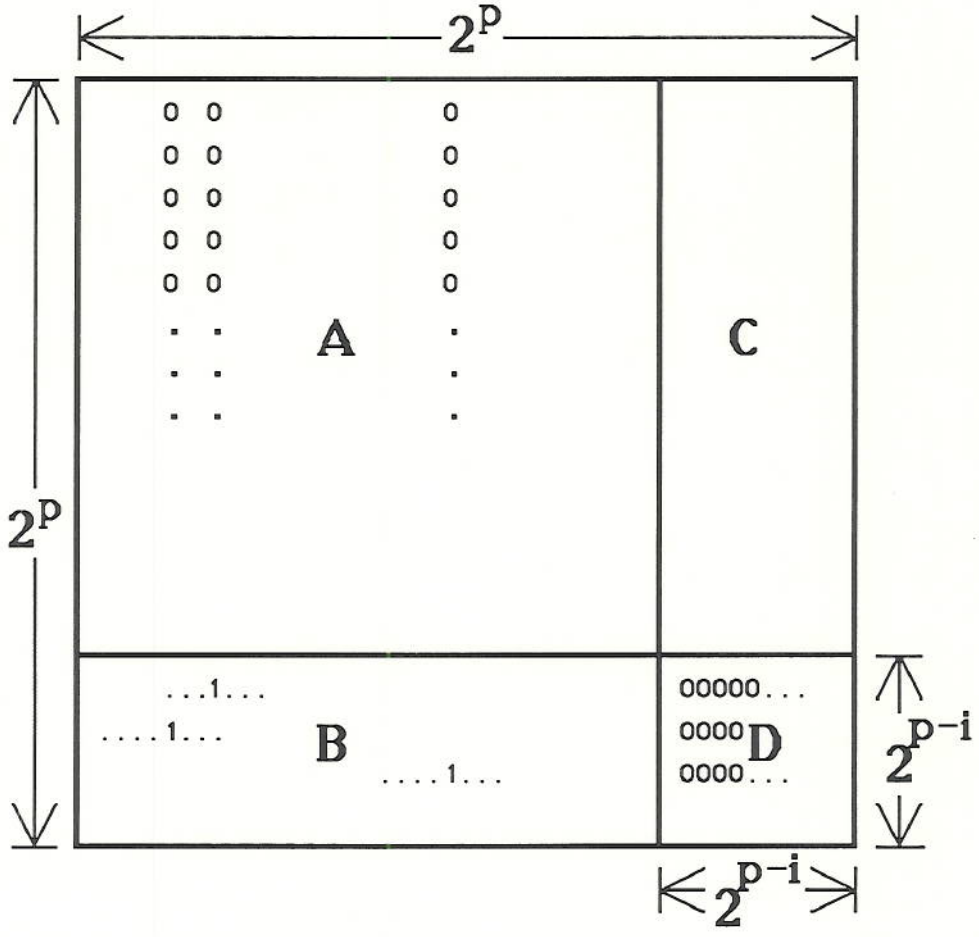

Figure 4. Calculation of the number of $2^p \times 2^p$ permutation matrices such that the last 2^{p-i} columns of the last 2^{p-i} rows are all zero.