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.*