TABLE X
OVERLAPPED-SCATTER STORAGE REQUIREMENTS

Matrix	# Nonzeros	Bound on Overlap Storage
DSP	35712	47543
64K	50539	57329
256K	52512	83338

and solution (*SOLVE*) as the computational bulk of the algorithm. In Section IV, we described the parallelization of *LOAD*. We identified parallel *LOAD* as a sequence of lock-synchronized parallel loops to provide a model of their behavior. We then used this analysis to predict the performance of our parallel implementation of *LOAD* and compared it to experimental measurements on up to six processors.

In Section V, we considered parallel sparse matrix solution. We first explored the direct parallelization of the efficient sequential scatter-gather algorithm and concluded on the basis of experimental results that it was not very promising. We then used a fine-grained model of parallelism and reported an efficient implementation using task clustering to minimize scheduling overheads. Very good utilization of concurrent parallelism was observed, but a performance penalty had to be paid due to increased indirect operand accesses necessitated by the fine-grained model. Finally, we described the implementation of a new parallel medium-grained sparse matrix solver. The key idea here was the use of an auxiliary indirect destination pointer to facilitate efficient source-target matching. Although the fine-grained approach provided slightly better load balancing and lower scheduling overheads, the medium-grained solver achieved superior overall performance due to significantly lower operand access costs and better vectorization. The storage needed for the medium-grained solver, although only approximately one-third that for the fine-grained solver, was, however, on the order of the number of arithmetic operations. We therefore concluded the section by describing some work currently in progress on a parallel solver that is much more space efficient without any loss in performance.

ACKNOWLEDGMENT

We would like to thank D. Berkley for graciously allowing us to use his machine, P. Subramaniam for numerous technical discussions, and H. Nham for his support and encouragement.

REFERENCES

[1] G. Alghband and H. F. Jordan, "Multiprocessor sparse L/U decomposition with controlled fill-in," Tech. Rep. 85-48, ICASE, NASA Langley Res. Center, Hampton, VA, 1985.
[2] P. R. Benyon, "Exploiting vector computers by replication," *Comput. J.*, vol. 28, no. 2, pp. 138-141, 1985.
[3] R. Betancourt, "Efficient parallel processing technique for inverting matrices with random sparsity," *IEE Proc.*, vol. 133, pt. E, pp. 235-240, July 1986.
[4] G. Bischoff and S. Greenberg, "CAYENNE: A parallel implementation of the circuit simulator SPICE," in *Proc. Int. Conf. Comput.-Aided Design*, Santa Clara, CA, Nov. 1986, pp. 182-185.
[5] P. Cox, R. Burch, and B. Epler, "Circuit partitioning for parallel processing," in *Proc. Int. Conf. Comput.-Aided Design*, Santa Clara, CA, Nov. 1986, pp. 186-189.
[6] T. A. Davis and E. S. Davidson, "PSOLVE: A concurrent algorithm for solving sparse systems of linear equations," in *Proc. 1987 Int. Conf. Parallel Processing*, St. Charles, IL, Aug. 1987, pp. 483-490.
[7] J. J. Dongarra, F. G. Gustavson, and A. Karp, "Implementing linear algebra algorithms for dense matrices on a vector pipeline machine," *Siam Rev.*, vol. 26, no. 1, pp. 91-112, 1984.
[8] I. S. Duff, "Parallel implementation of multifrontal schemes," *Parallel Comput.*, vol. 3, pp. 193-204, 1986.
[9] ——, "Multiprocessing a sparse matrix code on the Alliant FX/8," Rep. CSS210, Comput. Sci. Syst. Division, Harwell Lab., 1987.
[10] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*. London, England: Oxford University Press, 1986.
[11] FX/Architecture Manual, Alliant Comput. Corp., 1986.
[12] B. Greer, "Converting SPICE to vector code," *VLSI Syst. Design*, Jan. 1986.
[13] J. W. Huang and O. Wing, "Optimal parallel triangulation of a sparse matrix," *IEEE Trans. Circuits Syst.*, vol. CAS-26, pp. 726-732, Sept. 1979.
[14] G. K. Jacob, A. R. Newton, and D. O. Pederson, "Parallel linear-equation solution in direct-method circuit simulators," in *Proc. Int. Symp. Circuits Syst.*, Philadelphia, PA, May 1987, pp. 1056-1059.
[15] J. W. H. Liu, "Computational models and task scheduling for parallel sparse Cholesky factorization," *Parallel Comput.*, vol. 3, pp. 327-342, 1986.
[16] R. Lucas, T. Blank, and J. Tiemann, "A parallel solution method for large sparse systems of equations," *IEEE Trans. Comput.-Aided Design*, vol. CAD-6, pp. 981-991, Nov. 1987.
[17] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Memo. ERL-M520, Electron. Res. Lab., Univ. California, Berkeley, May 1975.
[18] L. W. Nagel, AT&T Bell Labs., unpublished work.
[19] B. R. Penumalli, AT&T Bell Labs., unpublished work.
[20] J. Perry, Alliant Comput. Syst., unpublished work.
[21] P. Sadayappan and V. Visvanathan, "Circuit simulation on a multiprocessor," in *Proc. Custom Integrated Circuits Conf.*, Portland, OR, May 1987, pp. 124-128.
[22] ——, "Parallelization and performance evaluation of circuit simulation on a shared memory multiprocessor," in *Proc. 1988 Int. Conf. Supercomput.*, St. Malo, France, July 1988, pp. 254-265.
[23] D. Smart and J. White, "Reducing the parallel solution time of sparse circuit matrices using reordered Gaussian elimination and relaxation," in *Proc. Int. Symp. Circuits Syst.*, Helsinki, Finland, 1988.
[24] R. E. Tarjan and A. C. Yao, "Storing a sparse table," *Commun. ACM*, vol. 22, pp. 606-611, Nov. 1979.
[25] J. Vlach and K. Singhal, *Computer Methods for Circuit Analysis and Design*. New York: Van Nostrand Reinhold, 1983.
[26] W. T. Weeks, A. J. Jimenez, G. W. Mahoney, D. Mehta, H. Qassemzadeh, and T. R. Scott, "Algorithms for ASTAP—A network analysis program," *IEEE Trans. Circuit Theory*, vol. CT-20, pp. 628-634, Nov. 1973.
[27] F. Yamamoto and S. Takahashi, "Vectorized *LU* decomposition algorithms for large-scale circuit simulation," *IEEE Trans. Comput.-Aided Design*, vol. CAD-4, pp. 232-238, July 1985.

Simulating Essential Pyramids

RUSS MILLER AND QUENTIN F. STOUT

Abstract—**Pyramid computers, and more generally pyramid algorithms, have long been proposed for image processing. They have the advantage of being a regular structure with a base naturally identified with an input image, and a logarithmic height which enables rapid reduction of information. Unfortunately, when the image contains multiple objects of interest, it is often difficult to use the pyramid efficiently since the most straightforward algorithms try to simultaneously use the apex for each object, creating a severe bottleneck. Furthermore, algorithms which are efficient in such settings tend to be**

Manuscript received February 17, 1988; revised July 16, 1988. R. Miller is supported by the National Science Foundation under Grants DCR-86-08640 and IRI-8800514. Q. F. Stout is supported by the National Science Foundation under Grant DCR-85-07851 and by an Incentives for Excellence Award from Digital Equipment Corporation.

R. Miller is with the Department of Computer Science, State University of New York at Buffalo, Buffalo, NY 14260.

Q. F. Stout is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109.

IEEE Log Number 8824090.

0018-9340/88/1200-1642$01.00 © 1988 IEEE

significantly more complicated than is desirable, limiting the appeal of pyramids for more complicated analyses and images. However, this paper shows that one can systematically simulate the effect of having a separate "essential" pyramid over each object, greatly simplifying algorithm development since algorithms can be written assuming that there is only a single object. This approach can yield optimal or nearly optimal algorithms for the pyramid computer, and can also be used on nonpyramid architectures such as the hypercube, mesh-of-trees, mesh, mesh with row and column buses, mesh with reconfigurable buses, and PRAM. For several of these architectures, the simulated essential pyramids can all simultaneously perform an algorithm nearly as fast as a pyramid computer over a single object.

Index Terms—**Hypercube computers, image processing, mesh-of-trees, meshes with buses, parallel algorithms, pyramid computers, vision architectures.**

I. INTRODUCTION

Pyramid-like computers [38] and pyramid-like data structures [26] have long been proposed for image processing. Many of these proposals have been based on biological considerations, noting that the retina has an extremely large number of simultaneous inputs, and yet in a small number of neural steps animals can perform a variety of recognition tasks [39]. While the steps in this process are not fully understood, one thing that is clear is that there must be a rapid funneling of data into a region in which the decision is made.

The (standard) pyramid computer was proposed as a model of this behavior, having a large square base in which the image input starts (as an array of pixels), and only a logarithmic number of stages which reduces the data to a single apex. Throughout this paper, the base of the pyramid is an $n \times n$ mesh, called level 0. In general, *level i* is a mesh of size $(n/2^i) \times (n/2^i)$, where each processing element (PE) is the parent of four children on the level below. (See Fig. 1.) While pyramids have been studied for some time, only recently have advances in VLSI technology made it possible to build pyramids of interesting size, and several have recently been constructed or are in the process of being constructed [2], [3], [7], [14], [27], [36]. Along with this ability to construct pyramids has been a rapid expansion of interest in them (see [3], [24], [35], [40] and the extensive references therein.)

In a pyramid with an $n \times n$ base, the hope is that a given problem can be solved in logarithmic time, that is, in time $\Theta(\log n)$. (Θ denotes "order exactly," Ω denotes "order at least," O denotes "order at most," and o denotes "order strictly less than.") Any nontrivial problem must take $\Omega(\log n)$ time since it takes that long for information to travel from the base to the apex. If one has an algorithm in which data proceed strictly upward, with each level taking a fixed amount of time before finishing by passing information up, then indeed the problem will be completed in $\Theta(\log n)$ time. For example, if the input is a graytone image stored one pixel per base processor, and the goal is to determine the average gray level, then the base PE's send their values up to their parents. As the information proceeds upwards, each PE in the receiving level receives four values from its children. The PE then averages these values and passes up the result. In exactly $\lg(n)$ steps the information reaches the apex, which computes the answer. ($\lg$ denotes $\log_2$.) This approach can also be used to compute higher order least squares polynomial approximations of the image [1]. Other problems that can also be solved with bottom-up techniques, using a constant amount of time per level, include detecting features such as spots and streaks [25] and finding the closest pair of "marked" pixels [32]. These latter algorithms involve communication between adjacent PE's on each level, as well as child-parent communication.

Unfortunately, many problems cannot be solved in a single pass, nor even in a fixed number of up or down passes. For example, to rotate the image through a quarter-turn will take $\Omega(n)$ time, as can be shown by a simple counting argument similar to that used in [31] to show that sorting also requires $\Omega(n)$ time. Namely, consider any quadrant of the base, together with the subpyramid over it. This subpyramid has $\Theta(n)$ wires leaving it, and $\Theta(n^2)$ values which must leave over these wires, so $\Omega(n)$ time is required. (Furthermore, rotation or sorting can be completed in $\Theta(n)$ time using the base mesh alone [37].)

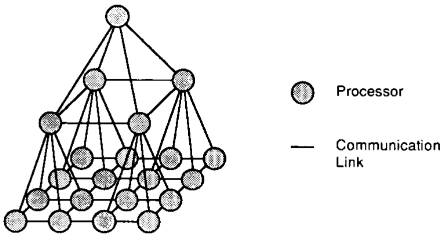

Fig. 1. A pyramid computer of size 16.

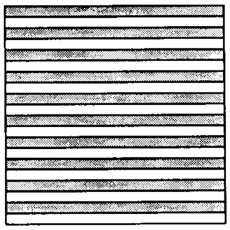

Fig. 2. A difficult image.

Even problems which can be solved in logarithmic time for a single object may not be solvable in logarithmic time if there are multiple objects in the image and a separate answer is desired for each. For example, if one has an input as in Fig. 2, and wants to compute the average gray level for each of the dark stripes, then at least $\Omega(n^{1/2})$ time is necessary [19]. Intuitively, the reason for this delay is that each stripe needs to use the apex (or at least, the upper levels of the pyramid) to rapidly combine data from opposite ends of the base. This creates a bottleneck, and one can show that it is best to move data only halfway up the pyramid. Simpler techniques such as solving the problem for one object at a time can lead to far worse performance. For example, finding the average gray level of each stripe in Fig. 2 would take $\Theta(n \log n)$ time with this approach.

While the techniques in [17] and [19] show how to attain the optimal worst case performance for many problems involving multiple objects, the algorithms are sufficiently complex so as to discourage their use. In this paper, we take a different approach which we believe to be much more appealing, and which can be applied to many other models. For both the pyramid and numerous nonpyramid machines we show how to simulate an "essential" pyramid (defined in Section II) over each object in the base image, where all of the simulated pyramids are logically disjoint. Using them, when one has a problem involving calculating a property of each object, one merely writes an algorithm for the problem on an essential pyramid with a single object, executing this in all the simulated pyramids simultaneously.

Besides providing a systematic solution to determining properties of multiple objects, using simulated essential pyramids has two additional advantages. The first is that the resulting algorithms are often optimal or nearly optimal, where by nearly optimal we mean that the time is within one or two logarithmic factors of being optimal. Thus, it is far more efficient than merely running an algorithm on the objects one at a time. The second is that the approach can be used on a wide range of parallel computers, not just pyramids. In fact, when used on more powerful models, such as the hypercube and mesh-of-trees, the time to run an algorithm on all of the simulated pyramids is nearly the same as the time to run the same algorithm on a pyramid with a single object.

In the second section, the various machine models, and the notion of essential pyramids, are defined. The third section shows how to simulate essential pyramids over each object for a variety of machines. It also analyzes the time required to simulate one step, or one upward or downward sweep, of an algorithm for an essential pyramid with a single object. The fourth section uses this to give algorithms for images with multiple objects on a variety of different machines. The last section contains additional comments.

II. Models and Definitions

In all of the models of parallel computers considered in this paper, it is assumed that the PE's have a uniformly bounded amount of memory, the word size is large enough to contain n, all operations such as add or compare take constant time, and it takes a constant amount of time to transmit a single word to a neighboring PE. (Two PE's are said to be ***neighbors*** if there is a direct communication link between them.)

Machine Models

The pyramid computer, shown in Fig. 1 and defined in the Introduction, is said to be of *size* n^2 if the base is an $n \times n$ mesh. Notice that a pyramid of size n^2 actually has $(4/3)n^2 - 1/3$ PE's. The input is assumed to be an $n \times n$ image, stored in the natural fashion, one pixel per base PE. Thus, the input to a pyramid of size n^2 is an image of n^2 pixels. Also notice that n must be an integral power of 2. This restriction will be used throughout this paper.

A *mesh-of-trees of size* n^2 also starts with a base which is an $n \times n$ mesh. Above each row there is a binary tree which has the PE's in the row as its leaves, and above each column there is a similar tree. These trees are disjoint except for their leaves. (See Fig. 3.) Like the pyramid, the image is assumed to be initially stored in the base. The mesh-of-trees was originally studied as a useful model in VLSI layouts [11], [12], [41], but recently it has been receiving more attention as a parallel computer [10], [18], [20], [22].

Notice that, like the pyramid, the mesh-of-trees has a communication diameter of $\Theta(\log n)$. Furthermore, it can provide logarithmic time solutions to some problems that cannot be solved even in polylogarithmic time on a pyramid. (A function is *polylogarithmic* if it is $O(\log^k n)$ for some $k > 0$.) For example, by using only the row trees, in $\Theta(\log n)$ time the mesh-of-trees can find the average gray level of each of the stripes in Fig. 2. However, the mesh-of-trees is similar to the pyramid in that rotating an image by a quarter-turn, or sorting items, takes $\Omega(n)$ time, as can be shown by using the same argument given for the pyramid.

A ***hypercube of size*** n^2 has exactly n^2 PE's, indexed by the binary numbers from 0 to $n^2 - 1$. Two PE's are neighbors if and only if their indexes differ by an integral power of 2. (See Fig. 4.) Hypercube computers have been proposed for quite some time [30], but only in the last few years have any nontrivial ones been built [28]. Currently several corporations market hypercube computers, and there is extensive interest in such machines [8], [9], [43].

Images are mapped onto hypercubes using Gray codes G_d, where G_d maps $\{0, \cdots, 2^d - 1\}$ onto the d-bit binary strings, with the property that $G_d(i)$ and $G_d((i + 1) \bmod 2^d)$ differ by exactly one bit. For our purposes it is best to use the reflexive Gray codes defined by $G_1(0) = 0$, $G_1(1) = 1$, and $G_{d+1}(i) = 0G_d(i)$ for $i < 2^d$, $G_{d+1}(i) = 1G_d(2^{d+1} - 1 - i)$ for $i \geq 2^d$. Using this, the pixel at (i, j) is mapped to the PE with coordinates the concatenation of $G_{\lg(n)}(i)$ and $G_{\lg(n)}(j)$.

The communication diameter of a hypercube of size n^2 is $2\lg(n)$, and, unlike the pyramid or mesh-of-trees, the hypercube can route many items nearly as rapidly as it can route a single item. For example, an image can be rotated a quarter-turn in $\Theta(\log n)$ time, and n^2 items can be sorted in $\Theta(\log^2 n)$ time using bitonic sort. (It is an interesting open question whether the hypercube can sort in $o(\log^2 n)$ time.)

Many different variations on adding buses to mesh computers have been studied. Here only two will be considered. In a *mesh of size* n^2 *with row and column buses*, there is a base mesh of size n^2. There is a bus for each row and each column, with all of the PE's in a row or column attached to the bus. For each bus, only one PE at a time can transmit a value, which is simultaneously received in unit time by all the other PE's attached to the bus. This model has been studied in [23] and [33]. It has a communication diameter of $\Theta(1)$, but again a simple wire-counting argument shows that it takes $\Omega(n)$ time to rotate an image by a quarter-turn.

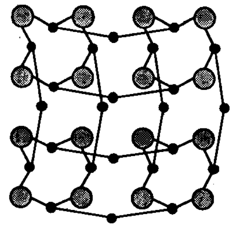

Fig. 3. A mesh-of-trees of size 16. (Mesh connections omitted for clarity.)

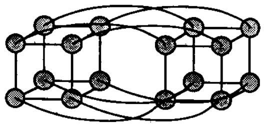

Fig. 4. A hypercube of size 16.

In a *reconfigurable mesh of size* n^2, or *mesh of size* n^2 *with reconfigurable buses*, imagine a grid of horizontal and vertical wires, each of length n and spaced unit distance apart. A PE is attached to each intersection, and the PE can control a switch on each of the four sides of the intersection. Each switch can disconnect the wire from the intersection. If all of the switches are set to the "connected" positions, then the entire system is linked as a single bus in which only one PE can broadcast a value which is received in unit time by all other PE's. If, for example, all of the column switches are set to "disconnect," and all row switches to "connect," then the system is a collection of rows, each with its own bus. These can all be used simultaneously, just as in a mesh with row and column buses. The possibility of having connection patterns which are neither rows nor columns makes this model significantly more powerful than the mesh with row and column buses [16], although it still takes $\Omega(n)$ time to rotate an image by a quarter-turn. Many models similar to a reconfigurable mesh have been proposed and constructed [5], [13], [16], [29], [42].

A *parallel random access machine (PRAM) of size* n^2 has n^2 PE's, each with their own storage of fixed size, and a global memory of size $\Theta(n^2)$. In a PRAM, PE's do not communicate directly with each other, but instead communicate by leaving messages in the global memory. We assume that each memory cell can be accessed by at most one PE at a time. This is known as the "exclusive read exclusive write" (EREW) model of PRAM, and is the most restrictive PRAM model usually studied. We also assume that the PRAM is synchronous, although with extra "handshaking" communication all of the algorithms used herein could be run in the same time (measured in O-notation) on the asynchronous model.

For the PRAM we assume that the image is initially stored in the global memory, in a standard row-major order. The PRAM is the most powerful model we will study since any collection of up to $n^2/2$ pairs of PE's can communicate (through the global memory) in $\Theta(1)$ time. For example, image rotation can be accomplished in $\Theta(1)$ time, and sorting can be accomplished in $\Theta(\log n)$ time [6].

Essential Pyramids

Throughout, we assume that the pixels corresponding to an object of interest form a connected set, and that no two objects overlap. We also assume that the pixels have already been labeled, where each pixel in an object receives the label of that object, objects have distinct labels, and each pixel not in any object receives a "null" label.

Many pyramid algorithms which compute a property of a single object use only the PE's above the object to do useful work. Motivated by this, [20] introduced the notion of an *essential*

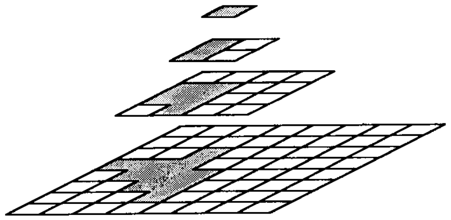

Fig. 5. Gray nodes are in the essential pyramid.

pyramid, defined as follows. To help distinguish essential pyramids from the simulating machine, PE's in the essential pyramid will be called *nodes*. Given an image stored in the base of a pyramid P of size n^2, let O be a single object in the image. Delete from P's base all nodes which do not contain a pixel in O, and then, in a bottom-up fashion, delete from P all nonbase nodes which no longer have any children. When this is completed, the remaining portion P' of P contains exactly those nodes which are above some part of O. However, this is still not quite what is desired, since, for example, an object consisting of a single pixel will have a P' consisting of lg n nodes, each with a single parent and a single child. In this situation, nodes above the base cannot combine any information not directly available to the base, and hence the base could act as the apex. Therefore, we add a second pruning pass to P'. This pass starts at the apex of P', deleting the apex if it has only one child. If the apex is deleted, then the process is recursively applied to its child (stopping at the base if the object is a single pixel). The result of this second pass is the essential pyramid of O, denoted ess(O). (See Fig. 5.) Notice that the ess(O) depends upon P and O's position in the base of P, but as this dependency will be true for all objects it will not be explicitly mentioned further.

Even with all of the pruning, ess(O) can be significantly larger than O. For example, consider an object O which is the four center squares of the base. At each level of the pyramid, there are exactly four nodes above O, except for the top level in which there is only the apex. At each level, the mesh connections between these four nodes form a square, so ess(O) is an obelisk of height lg(n).

Within a given pyramid, two disjoint objects may have essential pyramids which are not disjoint. For example, if each gray strip in Fig. 2 is an object, then all the essential pyramids contain the apex. This shows that the apex can be contained in $\Theta(n)$ essential pyramids, and that in general a node at level i can be contained in $\Theta(2^i)$ essential pyramids. Since the objects must be connected, the top-down pruning ensures that if a node p in ess(O) is not the apex, then O passes through the border of the base square beneath p. Since a node at level i is above a base border of $\Theta(2^i)$ pixels, and since the objects are disjoint, a node at level i is in $O(2^i)$ essential pyramids. Adding up all of the worst case possibilities shows that the total number of nodes in the essential pyramids is $O(n^2)$, i.e., at most linear in the size of the input. These facts will be used repeatedly when constructing the simulated essential pyramids.

III. SIMULATING ESSENTIAL PYRAMIDS

For all of the models considered here, the essential pyramids will be simulated so that there is a fixed constant C, independant of n and of the image, so that each real PE simulates at most C different nodes in the essential pyramids. Using this, the local computations by any real PE are at most C times the local computations done by any node in an essential pyramid. Furthermore, the real PE's must simulate the communication between neighbors in the essential pyramids. Since the worst case information transmission time will be at least as large as the local computation time, it is the only component that will be discussed in the following.

The approach we take in the following constructions is to concentrate on those for the pyramid and mesh-of-trees, and then relate the other architectures to these two. This greatly shortens our presentation, although in practice one would write a more direct construction for each architecture, rather than simulating an architecture simulating essential pyramids.

Pyramids

To construct the essential pyramids for an image on a pyramid, we follow techniques from [19]. Consider the base as being partitioned into $k \times k$ squares, where k is a power of 4. Pick such a square, and let Q denote the subpyramid with that square as its base. The apex of Q, denoted q, is at level i = lg n. By the comments at the end of Section II, q is in at most $4k - 4$ essential pyramids, and each of q's children is in at most $2k - 4$ essential pyramids. There are exactly k PE's in Q at level $i/2$, each of which will be used to simulate at most four replicas of q and at most eight replicas of children of q. Each PE simulating a replica of q will also replicate q's children (at least, all the ones in the same essential pyramid). There may also be children of q which are the apex of an essential pyramid not including q, in which case the child is replicated all by itself. In [19], the PE's in Q at level $i/2$ were called the "data square" corresponding to the base square of Q. We now describe how the replication and simulation is accomplished.

The construction begins bottom-up. First, each PE containing a pixel of an object determines which of its neighbors also contains a pixel of the same object. It then creates a record with the object's label, the PE's coordinates, and the information about the neighbors. These records are used to create the simulated essential pyramids. If none of the neighbors are in the same object, then the base PE is the apex of the essential pyramid, and the record is finished. Otherwise, the records are moved up one level. Each PE at level 1 receives at most four records. Within each 2×2 data square at level 1 (corresponding to a 4×4 square in the base), the records are sorted by the object's label, using snake-like or proximity ordering. This groups all the information concerning the same object together within each data square, and from this information it can be decided which PE's at levels 1 and 2 of the original pyramid above the 4×4 square are in the essential pyramid for the object. The PE holding the first record (in sorted order) for an object will simulate these nodes in the essential pyramid.

Each record corresponding to a pixel in the interior of the base 4×4 square, or to a pixel on the border where the pixel on the opposite side of the border is not in the object, is now finished. Otherwise, there must be nodes above the current simulated level which are in the essential pyramid of the object. Each such record is then passed up, and the process is repeated. Each step corresponds to base squares 4 times larger in both dimensions, and simulated essential nodes 1 and 2 levels higher, using data squares 2 times larger in both dimensions.

When the bottom-up pass is completed, all of the nodes in the simulated essential pyramids have been identified. A real PE at level i which is simulating a node q can determine if q has an essential parent by sending a record up to the data square directly above, it can determine which of q's children are essential by sending records to the data squares directly below, and it can determine which of q's neighbors on the same level are essential by sending records to the neighboring data squares. All of this can be accomplished in $\Theta(2^i)$ time for all real PE's at level i, using techniques as in [19]. Furthermore, by exactly the same techniques, in $\Theta(2^i)$ time, a single step involving essential nodes at levels $2i$ and $2i - 1$ can be simulated by the real PE's at level i. If the simulated algorithm involves nodes at all levels in the essential pyramids, then by simulating first those at the base, then at level 1, then level 2, etc., a single simulation step can be completed in $\Theta(n^{1/2})$ time. This same technique also can be used to simulate upward or downward passes of an algorithm. Summarizing, we have the following.

Theorem 3.1: In a pyramid of size n^2, simulating a single step of an algorithm on all essential pyramids simultaneously can be completed in $\Theta(n^{1/2})$ time, and can be completed in $\Theta(2^{i/2})$ time if the step involves only simulated nodes at levels i, $i - 1$, and $i + 1$. Simulating an entire upward or downward pass, where at each step only nodes at one level are computing before sending results to the next level (each level finishing in a constant amount of time), can be completed in $\Theta(n^{1/2})$ time.

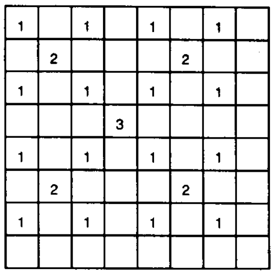

Fig. 6. A pyramid mapped onto its base. (Numbers indicate levels.)

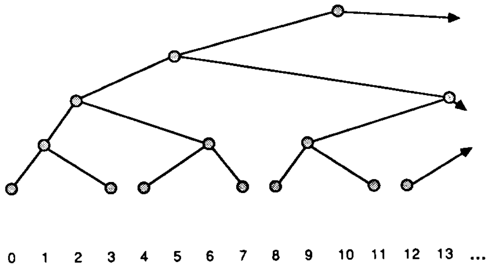

Fig. 7. Tree nodes above the base nodes they are mapped onto.

Meshes

Two-dimensional mesh-connected computers, or meshes, form the base of pyramids and the mesh-of-trees. It is also well known that the pyramid layers above the base can be projected onto the base mesh so that no base PE has more than one nonbase PE projected onto it, and so that for each PE p above the base, p is projected into the square that is the base of the subpyramid with p as apex. (See Fig. 6.) This can be used to easily construct the simulated essential pyramids. Namely, project a pyramid onto the mesh, and have each mesh PE simulate the essential nodes that it and the pyramid PE projected onto it (if any) would simulate in the pyramid. Using this, we obtain the following.

Theorem 3.2: In a mesh of size n^2, simulating a single step of an algorithm on all essential pyramids simultaneously can be completed in $\Theta(n)$ time, and can be completed in $\Theta(2^i)$ time if the step involves only simulated nodes at levels, i, $i - l$, and $i + 1$. Simulating an entire upward or downward pass can be completed in $\Theta(n)$ time.

Mesh-of-Trees

Simulating essential pyramids on the mesh-of-trees provides times significantly faster than those possible on the pyramid itself, and it seems that the mesh-of-trees is nearly ideal for this task. A node p at level i in an essential pyramid will be simulated by a PE q at level i in the mesh-of-trees, and q will be above the square which is the base of the subpyramid with p as its apex. To achieve rapid simulations, first the simulators of the nodes in the essential pyramids are identified, then each simulator determines the locations of the simulators of each neighbor of each node it simulates. The details of these steps appear in [20].

Theorem 3.3: In a mesh-of-trees of size n^2, simulating a single step of an algorithm on all essential pyramids simultaneously can be completed in $\Theta(\log n)$ time, and can be completed in $\Theta(i)$ time if the step involves only simulated nodes at levels i, $i - 1$, and $i + 1$. Simulating an entire upward or downward pass can be completed in $\Theta(\log^2 n)$ time.

Hypercube

Borrowing from [18], Fig. 7 illustrates one way in which a row tree in a mesh-of-trees can be mapped onto the base row so that no base PE has more than one nonleaf PE mapped onto it. Using this, the mesh-of-trees of size n^2 can be embedded into its base so that no base PE has more than one (nonleaf) PE from the row trees and one PE from the column trees mapped onto it. Furthermore, when the base mesh is mapped into a hypercube of size n^2 using the reflexive Gray code, the method illustrated in Fig. 7 has the additional property that two neighboring nodes in a row or column tree are mapped to base nodes which are in turn mapped to PE's at most distance 2 apart in the hypercube. This mapping of a mesh-of-trees of size n^2 into a hypercube of size n^2 results in each hypercube PE simulating one base PE, and at most one row tree PE and at most one column tree PE, and neighbors in the mesh-of-trees are mapped to PE's at most 2 apart in the hypercube. Using this, and the simulation results for the mesh-of-trees, gives the following result.

Theorem 3.4: In a hypercube of size n^2, simulating a single step of an algorithm on all essential pyramids simultaneously can be completed in $\Theta(\log n)$ time, and can be completed in $\Theta(i)$ time if the step involves only simulated nodes at levels i, $i - 1$, and $i + 1$. Simulating an entire upward or downward pass can be completed in $\Theta(\log^2 n)$ time.

Meshes with Buses

The mesh with row and column buses can simulate the mesh-of-trees by first mapping the mesh-of-trees onto its base (as in Fig. 7), and then using the row and column buses to help simulate communication within the row and column, respectively, trees. Within any single row or column, to simulate communication between a PE and one of its children, where the parent is at level $i \leq (\lg n)/2$, the communication will just use the standard mesh links. The bus is not used because such communication can occur in time $\Theta(2^i)$, while if the bus was used then the large number of such PE's would require at least this many messages (and hence at least this much time). For communication involving parents at higher levels, the buses are used, sending at most $n/2^i$ messages. Using this, we obtain the following.

Theorem 3.5: In a mesh of size n^2 with row and column buses, simulating a single step of an algorithm on all essential pyramids simultaneously can be completed in $\Theta(n^{1/2})$ time, and if the step involves only simulated nodes at levels i, $i - 1$, and $i + 1$, then the simulation can be completed in $\Theta(2^i)$ time if $i \leq (\lg n)/2$, and in $\Theta(n/2^i)$ time otherwise. Simulating an entire upward or downward pass can be completed in $\Theta(n^{1/2})$ time.

For a reconfigurable mesh, the mesh-of-trees is again mapped to its base in the same manner. In any row or column, for any level i, the bus can be partitioned into disjoint subbuses so that each simulated tree PE at level i, along with its children, is on its own subbus. Using this, the reconfigurable mesh can simulate all row or column tree PE's at level i communicating with their children in a constant amount of time. Furthermore, in the mesh-of-trees, an essential node at level i may have children simulated in different row or column trees, so $\Theta(i)$ time is needed to simulate this communication. The reconfigurable mesh can accomplish this in a constant amount of time [16], giving the following results which are slightly better than those for the mesh-of-trees.

Theorem 3.6: In a reconfigurable mesh of size n^2, simulating a single step of an algorithm on all essential pyramids simultaneously can be completed in $\Theta(\log n)$ time, and can be completed in $\Theta(1)$ time if the step involves only simulated nodes at levels i, $i - 1$, and $i + 1$. Simulating an entire upward or downward pass can be completed in $\Theta(\log n)$ time.

PRAM

The PRAM can simulate a single step of the mesh-of-trees in constant time. Furthermore, communication between arbitrary PE's can occur in constant time. Using this, one obtains the following.

Theorem 3.7: In a PRAM of size n^2, simulating a single step of an algorithm on all essential pyramids simultaneously can be completed in $\Theta(1)$ time. Simulating an entire upward or downward pass can be completed in $\Theta(\log n)$ time.

IV. SAMPLE USES OF SIMULATED ESSENTIAL PYRAMIDS

Several algorithms have been developed for pyramids with a single object in their base. While authors did not write these in terms of essential pyramids, the vast majority of them actually run with little or no modification on essential pyramids. Illustrative, nonexhaustive examples of this are mentioned in the following theorem. In the theorem, an upward or downward pass will always take $O(1)$ time per level.

Theorem 4.1: In an essential pyramid over a single object in an image of size n^2,

a) In a single upward pass, the average gray level, standard deviation of the gray level, number of pixels, number of pixels on the perimeter, and centroid of the object can be determined.

b) For any fixed k, in a single upward pass, the least squares polynomial of degree k can be fit to the gray levels of the object, assuming that pixels not in the object have value 0.

c) In a single upward pass, it can be determined if the object has any horizontal or vertical concavities.

d) In a single upward pass, it can be determined if the object is a "spot," "streak," "streak end," edge of specified size, "blob," or "ribbon."

e) In a single upward pass, it can be determined if the object is the digitization of a straight line.

f) In $\Theta(\log n)$ time, it can be determined if the object is convex.

g) If the object is convex, then in $\Theta(\log n)$ time the extreme points of its convex hull can be determined.

h) For an arbitrary object, in $\Theta((\log^2 n)/\log \log n)$ time, the extreme points of its convex hull can be determined.

Proof: Parts a) and b) are quite obvious, with [1] apparently the first to note b). For c) and e), see [15]. For d) see [25]. For f), g), and h) see [17].

This theorem can be combined with those of the previous section to give a very large number of problem/machine times. This is too many to analyze in depth for optimality, so only a few will be analyzed to show that the resulting algorithm is optimal or nearly optimal for each machine examined. Even for the simplest problems, one obtains optimal or nearly optimal results.

Theorem 4.2: Once the simulated essential pyramids have been constructed, then for each object, in

a) $\Theta(n^{1/2})$ time on a pyramid of size n^2,
b) $\Theta(n)$ time on a mesh of size n^2,
c) $\Theta(\log^2 n)$ time on a mesh-of-trees of size n^2,
d) $\Theta(\log^2 n)$ time on a hypercube of size n^2,
e) $\Theta(n^{1/4})$ time on a mesh of size n^2 with row and column buses,
f) $\Theta(\log n)$ time on a reconfigurable mesh of size n^2,
g) $\Theta(\log n)$ time on a PRAM of size n^2,

one can determine the average gray level, standard deviation of the gray level, number of pixels, length of the perimeter, centroid, kth degree polynomial fit, detect horizontal or vertical concavities, or decide if the object is a "spot," "streak," "streak end," edge of specified size, "blob," "ribbon," or digitization of a straight line. Furthermore, the times for the pyramid, mesh, mesh with row and column buses, reconfigurable mesh, and PRAM are (worst case) optimal, and the times for the mesh-of-trees and hypercube are within a factor of log n of being optimal.

Proof: The times obtained are immediate consequences of Theorem 4.2 and the theorems in Section III. The optimality of the times for the reconfigurable mesh and PRAM are obvious. For the mesh, the optimality follows from the fact that it takes $\Theta(n)$ time to transmit information across the mesh, and the near optimality of the times for the hypercube and mesh-of-trees follows similarly. To see optimality for the other models, consider the stripes in Fig. 2. For these objects, a lower bound proof in [19] shows that the time in a) is optimal for the pyramid, and the analysis in [33] shows that the time in e) is optimal for the mesh with row and column buses.

Slightly faster algorithms are known for determining average gray level (and all of the other properties as well) on the mesh-of-trees and hypercube. An algorithm for the mesh-of-trees which takes $\Theta((\log^2 n)/\log \log n)$ time, and an algorithm for the hypercube which takes $\Theta(\log n)$ time, are given in [20]. The latter is clearly optimal, while the optimal time for the mesh-of-trees is still an open problem.

Theorem 4.3: Once the simulated essential pyramids have been constructed, then for all objects simultaneously, in

a) $\Theta(n^{1/2} \log^2 n/\log \log n)$ time on a pyramid of size n^2,
b) $\Theta(n \log^2 n/\log \log n)$ time on a mesh of size n^2,
c) $\Theta(\log^3 n/\log \log n)$ time on a mesh-of-trees of size n^2,
d) $\Theta(\log^3 n/\log \log n)$ time on a hypercube of size n^2,
e) $\Theta(n^{1/4} \log^2 n/\log \log n)$ time on a mesh of size n^2 with row and column buses,
f) $\Theta(\log^3 n/\log \log n)$ time on a reconfigurable mesh of size n^2,
g) $\Theta(\log^2 n/\log \log n)$ time on a PRAM of size n^2,

one can find the extreme points of the convex hull of each object.

Again it is true that some of these architectures can find extreme points faster than the times given in Theorem 4.3. For example, on the PRAM and hypercube, all extreme points can be determined in $\Theta(\log n)$ time by incorporating techniques in [21]. However, the existence of slightly faster algorithms for a few of these models does not diminish the significance of the use of simulated essential pyramids. Simulated essential pyramids provide a systematic technique for exploiting pyramid algorithms to determine properties of multiple objects, and the programmer/algorithm designer time saved by using a systematic approach may be far more valuable than the extra machine time incurred.

V. Final Remarks

Simulating essential pyramids provides a systematic way of extending pyramid algorithms for single objects to images with multiple objects, on a variety of parallel computers. The software necessary to create simulated essential pyramids, and to carry out stepwise simulations, needs to be written only once for each machine. Once this is in place, a programmer needs only to label the objects, and to write an algorithm for an essential pyramid with a single object in its base. This is then simulated over all objects. While simulations of one architecture by another are common in parallel computing, it is rare that one simulates multiple copies of one architecture on another. Simulating multiple copies of the "essence" of another architecture, where the essence is defined by the data, is even rarer, as this is the first example that we know of.

This approach has obvious advantages in reducing programmer time, and in enabling algorithms to be ported between different architectures. Less obvious is the fact that it often produces algorithms that are worst case optimal, or within a logarithmic factor of being optimal. This was shown in Theorems 4.2 and 4.3. Furthermore, for all of the architectures studied, the number of processors needed to obtain these times grows only linearly with the size of the input image.

The machine architectures discussed herein are not an exhaustive list of architectures for which this approach can be profitably used. For example, one could also use this approach on a mesh in which each row and each column is a complete graph, or has a multistage interconnection network. It can also be used on a two-level mesh where the base level is an $n \times n$ mesh and the level above is a $k \times k$ mesh, where $k < n$. Here each PE in the second level may be connected to all the base PE's in a $(n/k) \times (n/k)$ square, or to only one of the PE's in such a square. It can also be used on the enhanced pyramids discussed in [34], on cube-connected cycles, and on shuffle-exchange machines.

Besides varying the target architectures, one can also vary many aspects of the basic technique. For example, on the mesh-of-trees one may prefer to group levels of essential pyramids into layers of lg lg n levels thick, simulating a node at the top of a layer, and all of its essential descendants in the same layer, together in one PE. The internal time for a PE to simulate a single step increases, but in some cases this can be more than made up for by the decrease in the communication required [20]. Another possibility is to consider algorithms which alter the shape of an essential pyramid. For example, suppose the convex hulls of the objects are all disjoint, and the task is to "fill in" the convex hull of each object. The extreme points of the objects can be determined using the essential pyramids, and the most natural way to fill in the convex hulls is to use a top-down approach, filling in the essential pyramid of the convex hull of each object.

REFERENCES

[1] P. J. Burt, "Hierarchically derived piecewise polynomial approximations to waveforms and images," TR-838, Comput. Vision Lab., Univ. Maryland, 1979.

[2] P. J. Burt, C. H. Anderson, J. O. Sinniger, and G. van der Wal, "A pipelined pyramid machine," in *Pyramidal Systems for Computer Vision, NATO ASI Series F: Computer and System Sciences, Vol. 25.* New York: Springer-Verlag, 1986, pp. 133-152.

[3] V. Cantoni and S. Levialdi, Eds., *Pyramidal Systems for Computer Vision, NATO ASI Series F: Computer and Systems Sciences*, vol. 25. New York: Springer-Verlag, 1986.

[4] V. Cantoni and S. Levialdi, "PAPIA: A case history," in *Parallel Computer Vision.* New York: Academic, 1987, pp. 3-13.

[5] D. M. Champion and J. Rothstein, "Immediate parallel solution of the longest common subsequence problem," in *Proc. 1987 Int. Conf. Parallel Processing*, pp. 70-77.

[6] R. Cole, "Parallel merge sort," in *Proc. 27th Symp. Foundations Comput. Sci.*, 1986, pp. 511-516.

[7] G. Fritsch, "General purpose pyramidal architectures," in *Pyramidal Systems for Computer Vision, NATO ASI Series F: Computer and System Sciences*, vol. 25. New York: Springer-Verlag, 1986, pp. 41-58.

[8] M. T. Heath, Ed., *Hypercube Multiprocessors 1986.* Philadelphia, PA, SIAM, 1986.

[9] ——, *Hypercube Multiprocessors 1987.* Philadelphia, PA, SIAM, 1987.

[10] M.-D. A. Huang, "Solving some graph problems with optimal or near-optimal speedup on mesh-of-trees networks," in *Proc. 26th Symp. Foundations Comput. Sci.*, 1985, pp. 232-240.

[11] F. T. Leighton, "New lower bound techniques for VLSI," in *Proc. 22nd Symp. on Foundations Comput. Sci.*, 1981, pp. 85-98.

[12] ——, "Parallel computation using meshes of trees," in *Proc. 1983 Int. Conf. Graph Theoret. Concepts Comput. Sci.*

[13] H. Li and M. Maresca, "Polymorphic-torus network," in *Proc. 1987 Int. Conf. Parallel Processing*, pp. 411-414.

[14] A. Merigot, P. Clermont, J. Mehat, F. Devos, and B. Zavidovique, "A pyramidal system for image processing," in *Pyramidal Systems for Computer Vision, NATO ASI Series F: Computer and System Sciences*, vol. 25. New York: Springer-Verlag, 1986, pp. 109-124.

[15] R. Miller, "Pyramid computer algorithms," Ph.D. dissertation, State Univ. New York, Binghamton, 1985.

[16] R. Miller, V. K. Prassana Kumar, D. Reisis, and Q. F. Stout, "Meshes with reconfigurable buses," in *Proc. 5th MIT Conf. Advanced Res. VLSI.* Cambridge, MA: MIT Press, 1988, pp. 163-178.

[17] R. Miller and Q. F. Stout, "Convexity algorithms for pyramid computers," in *Proc. 1984 Int. Conf. Parallel Processing*, pp. 177-184.

[18] ——, "Some graph- and image-processing algorithms for the hypercube," in *Hypercube Multiprocessors 1987.* Philadelphia, PA: SIAM, 1987, pp. 418-425.

[19] ——, "Data movement techniques for the pyramid computer," *SIAM J. Comput.*, vol. 16, pp. 38-60, 1987.

[20] ——, *Parallel Algorithms for Regular Architectures.* Cambridge, MA: MIT Press, 1988.

[21] ——, "Efficient convex hull algorithms," *IEEE Trans. Comput.*, this issue, pp. 1605-1618.

[22] V. K. Prassana Kumar and M. Eschoghian, "Parallel geometric algorithms for digitized pictures on mesh of trees," in *Proc. 1986 Int. Conf. Parallel Processing*, pp. 270-273.

[23] V. K. Prassana Kumar and C. S. Raghavendra, "Image processing on enhanced mesh connected computers," in *Proc. Comput. Architecture Pattern Anal. Image Database Man.*, 1985, pp. 243-247.

[24] A. Rosenfeld, Ed., *Multiresolution Image Processing and Analysis.* New York: Springer-Verlag, 1984.

[25] ——, "Some pyramid techniques for image segmentation," in *Pyramidal Systems for Computer Vision, NATO Series F: Computer and System Sciences*, vol. 25. New York: Springer-Verlag, pp. 261-271.

[26] H. Samet, "A tutorial on quadtree research," in *Multiresolution Image Processing and Analysis.* New York: Springer-Verlag, 1984, pp. 212-223.

[27] D. H. Schaefer, P. Ho, J. Boyd, and C. Vallejos, "The GAM Pyramid," in *Parallel Computer Vision.* New York: Academic, 1987, pp. 15-42.

[28] C. L. Seitz, "The Cosmic cube," *Commun. ACM*, vol. 28, pp. 22-23, 1985.

[29] L. Snyder, "Introduction to the configurable, highly parallel computer," *Computer*, vol. 1, pp. 47-56, 1982.

[30] J. S. Squire and S. M. Palais, "Programming and design considerations for a highly parallel computer," in *Proc. AFIPS Conf.*, vol. 23, 1963 SJCC, pp. 395-400.

[31] Q. F. Stout, "Sorting, merging, selecting, and filtering on tree and pyramid machines," in *Proc. 1983 Int. Conf. Parallel Processing*, pp. 214-221.

[32] ——, "Pyramid computer solutions of the closest pair problem," *J. Algorithms*, vol. 6, pp. 200-212, 1985.

[33] ——, "Meshes with multiple buses," in *Proc. 27th Symp. Foundations Comput. Sci.*, 1986, pp. 264-273.

[34] ——, "Pyramid algorithms optimal for the worst case," in *Parallel Computer Vision.* New York: Academic, 1987, pp. 147-168.

[35] S. L. Tanimoto and A. Klinger, Eds., *Structured Computer Vision: Machine Perception Through Hierarchical Computation Structures.* New York: Academic, 1980.

[36] S. L. Tanimoto, T. J. Ligocki, and R. Ling, "A prototype pyramid machine for hierarchical cellular logic," in *Parallel Computer Vision.* New York: Academic, 1987, pp. 43-83.

[37] C. D. Thompson and H. T. Kung, "Sorting on a mesh-connected parallel computer," *Commun. ACM*, vol. 20, pp. 263-271, 1977.

[38] L. Uhr, "Layered 'recognition cone' networks that preprocess, classify, and describe," *IEEE Trans. Comput.*, vol. C-21, pp. 758-768, 1972.

[39] ——, "Parallel, hierarchical software/hardware pyramid architectures," in *Pyramidal Systems for Computer Vision, NATO ASI series F: Computer and System Sciences*, vol. 25. New York: Springer-Verlag, 1986, pp. 1-20.

[40] ——, *Parallel Computer Vision.* New York: Academic, 1987.

[41] J. D. Ullman, *Computational Aspects of VLSI.* Rockville, MD: Computer Science Press, 1984.

[42] C. C. Weems, S. P. Levitan, A. R. Hanson, E. M. Riseman, J. G. Nash, and D. B. Shu, "The image understanding architecture," COINS Tech. Rep. 87-76, Univ. Massachusetts, 1987.

[43] *Proc. Third Conf. Hypercube Concurrent Comput. Appl.*, SIAM, 1988.

Pairwise Reduction for the Direct, Parallel Solution of Sparse, Unsymmetric Sets of Linear Equations

TIMOTHY A. DAVIS AND EDWARD S. DAVIDSON

Abstract—**PSolve is a concurrent algorithm for solving sparse systems of linear equations on a shared-memory parallel processor. Each autonomous process uses pairwise pivoting and synchronizes with only a few others at a time. On the Alliant FX/8, PSolve is faster than Gaussian elimination and two common sparse matrix algorithms.**

Index Terms—**Asynchronous algorithms, concurrent data structures, gather/scatter vector hardware, pairwise pivoting, parallel algorithms, parallel architectures, sparse matrices, sparse vector hardware, synchronization.**

Manuscript received February 17, 1988; revised July 16, 1988. This work is supported in part by a fellowship awarded by the American Electronics Association (with funds from Digital Equipment Corporation) which supports graduate students who intend to pursue careers in academia, and by DOE Grant DE-FG02-85ER25001, NSF Grants MIP-84-10110 and DCR-85-09970, and a donation from IBM. This work has been presented in part at the 1987 International Conference on Parallel Processing [11].

T. A. Davis is with the Center for Supercomputing Research and Development, University of Illinois, Urbana, IL 61801.

E. S. Davidson is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109.

IEEE Log Number 8824091.

0018-9340/88/1200-1648$01.00 © 1988 IEEE