Efficient Parallel Convex Hull Algorithms

RUSS MILLER, MEMBER, IEEE, AND QUENTIN F. STOUT, MEMBER, IEEE

Abstract—**In this paper, we present parallel algorithms to *identify* (i.e., detect and enumerate) the extreme points of the convex hull of a set of planar points using a hypercube, pyramid, tree, mesh-of-trees, mesh with reconfigurable bus, EREW PRAM, and a modified AKS network. It is known that the problem of identifying the convex hull for a set of planar points given arbitrarily cannot be solved faster than sorting. For the situation where the input set of n planar points is given *ordered* (by *x-coordinate*) one per processor on a machine with $\Theta(n)$ processors, we introduce a worst case hypercube algorithm that finishes in $\Theta(\log n)$ time, a worst case algorithm for the pyramid, tree, and mesh-of-trees that finishes in $\Theta(\log^3 n/(\log \log n)^2)$ time, and a worst case algorithm for the mesh with a reconfigurable bus that finishes in $\Theta(\log^2 n)$ time. Notice that for ordered data the sorting bound does not apply. We also show that our $\Theta(\log n)$ time hypercube algorithm for ordered data extends to yield an optimal time and processor $\Theta(\log n)$ worst case time EREW PRAM algorithm for the case where the set of planar points is distributed *arbitrarily* one point per processor. We also show that this algorithm can be extended to run in worst case $\Theta(\log n)$ time on a modified AKS network, giving the first optimal $\Theta(\log n)$ time algorithm for solving the convex hull problem for arbitrary planar input on a fixed degree network.**

Index Terms—**AKS network, computational geometry, convex hull, EREW PRAM, hypercube, mesh, mesh-of-trees, parallel algorithms, pyramid, reconfigurable mesh.**

I. Introduction

THE CONVEX hull is a geometric structure of primary importance that has been well studied for the serial model of computation [8], [47], [52], [56], [60]. It has applications to normalizing patterns in image processing, obtaining triangulations of sets of points, topological feature extraction, shape decomposition in pattern recognition, and testing for linear separability, to name a few. Some general references which describe such problems, show some of their uses, and provide some serial algorithms solving them are [46], [47], [56].

Compared to the number of serial algorithms for solving such problems, the number of parallel algorithms is quite small. A number of parallel algorithms have been presented which computed geometric properties of digitized pictures (c.f., [17], [26], [35]–[37], [40], [42], and the references contained therein). However, problems that involve digitized pictures are significantly different from the problems that arise when the figures are represented as sets of points, which is a much more general form of input, and one that allows for larger problems to be solved on existing machines. In the early 1980's, parallel algorithms for convex hull problems using point data began to appear [3], [13], [44]. In 1984, the authors published a preliminary version [34] of [38] that included parallel algorithms for several problems involving geometric properties, such as convexity, proximity, area, intersection, and containers on a mesh computer, and Chazelle published a paper using a one-dimensional systolic computer to solve some problems involving convexity, proximity, and intersection [12]. Subsequently, additional papers with parallel algorithms for point data input have appeared [1], [6], [7], [19], [24], [30], and it can be expected that this trend will continue. Parallel computers provide the possibility of substantial improvements in the running time of algorithms, allowing larger problems to be solved in a feasible amount of time.

Elegant serial solutions to many problems are based on being able to efficiently construct the planar Euclidean Voronoi diagram of a set of planar points, or use sophisticated data structures specifically designed for geometric problems [52]. However, it is not clear that manipulating data structures for constructing Voronoi diagrams is as useful in parallel computers, since operations such as following a pointer may be very efficient on a serial computer but less so on a parallel one. Some of our algorithms are for local-memory parallel computers where information must be exchanged as messages between processors. In such a setting, the distance information must travel and the communication patterns that govern multiple messages become dominant considerations.

In this paper, we give algorithms to identify (i.e., detect *and* enumerate) the extreme points of the convex hull for a set of planar points on a hypercube, pyramid computer, tree machine, mesh-of-trees, mesh with reconfigurable bus, EREW PRAM, and a modified AKS network. It is known that the problem of identifying the convex hull for a set of planar points given arbitrarily cannot be solved faster than sorting [47]. Therefore, given a set S of n planar points, distributed arbitrarily one point per processor on a network M with n processors, sorting bounds dictate that solutions to the convex hull problem require $\Omega(T_M(n))$ worst case time, where $T_M(n)$ represents the worst case time to sort n items, stored one item per processor on network M.

The concentration of this paper is on developing polylogarithmic (i.e., $O(\log^c n)$, for $c \geq 1$ a constant) time algorithms for a variety of parallel machines. Since the convex hull is a geometric structure of importance to solving many other problems, there will be occasions when one is concerned

Manuscript received February 17, 1988; revised July 15, 1988. This work was supported by National Science Foundation Grants DCR-8507851, DCR-8608640, IRI-8800514 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, 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 8824087.

0018-9340/88/1200-1605$01.00 © 1988 IEEE

with identifying the extreme points of a set of planar points as an intermediate step in solving computationally intensive problems. In this situation, it is reasonable to assume that preprocessing of the data has been, or can be, performed to order the data. Most of the algorithms presented in this paper assume the input set S of planar points is initially distributed in an ordered fashion so that the x-coordinate of the point in processor P_i is less than the x-coordinate of the point in processor P_j, for $i < j$. (This ordered data assumption was used independently in [19] for obtaining an optimal $O(\log n)$ time algorithm for the CREW PRAM.)

Generally, the worst case running times of our convex hull algorithms that assume ordered data input are far superior to the worst case running times of algorithms possible for arbitrary input since these are bounded by sorting. In fact, for some of the mesh-based models discussed in this paper, for a set of planar points distributed arbitrarily, a best possible worst case lower bound on the time to solve the convex hull problem is $\Omega(n^{1/2})$. For the pyramid and mesh-of-trees, in which this $\Omega(n^{1/2})$ time bound applies, optimal $\Theta(n^{1/2})$ worst case time solutions to this problem are known [38]. A contrast in the running times of the algorithms based on differences in input can be seen by the fact that the algorithm we introduce for input size n on a pyramid with n base processors finishes in worst case $\Theta(\log^3 n/(\log \log n)^2)$ time when the data are ordered. This algorithm is also extended to give a worst case $\Theta(\log^3 n/(\log \log n)^2)$ time algorithm for ordered input on a tree and mesh-of-trees architecture, and to give a worst case $\Theta(\log^2 n)$ time algorithm for ordered input on a mesh with a reconfigurable bus. A somewhat different approach is used to give an optimal worst case $\Theta(\log n)$ time algorithm for ordered input on a hypercube. Lower bounds are discussed more fully in Section II-D.

As a byproduct of the algorithms that we develop for ordered input on a hypercube, we also introduce an optimal worst case $\Theta(\log n)$ time and n-processor EREW PRAM algorithm for solving the convex hull problem for planar point data input distributed arbitrarily one per processor. This improves on the optimal worst case $\Theta(\log n)$ time CREW PRAM algorithm given in [1] and [6] in that we use the weakest PRAM model, the EREW PRAM, which forbids concurrent reads and concurrent writes. We also give an expected $\Theta(\log n)$ time algorithm for the hypercube to solve the convex hull problem for planar point data input distributed arbitrarily one per processor. Finally, we give the first worst case $\Theta(\log n)$ time algorithm to solve the convex hull problem for planar point data input distributed arbitrarily one point per processor on a fixed degree network, specifically, a modified AKS network.

On parallel machines, the convex hull is often a fundamental step in determining minimum-area containers into which points can fit, the diameter of the set of points, linear separability of two sets of objects, and so forth. Since its applications are so wide ranging, it makes sense to consider running times of algorithms for different machines and different types of input.

In Section II, the notation, models of computation, and definitions that are used throughout the paper are defined. Furthermore, lower bounds are given for solving the convex hull problem, and different assumptions regarding the form of the input data are discussed. In Section III, a general parallel algorithm is given for solving the convex hull problem for ordered data. Worst case $\Theta(\log^3 n/\log \log n)$ time implementations of this algorithm are then given for the pyramid, tree, and mesh-of-trees, while an efficient $\Theta(\log^2 n)$ implementation is given for the mesh with a reconfigurable bus. The algorithm given in Section IV for solving the convex hull problem on a hypercube is different from the algorithms given in Section III, and has a worst case running time of $\Theta(\log n)$. In this section, we also give an expected $\Theta(\log n)$ time algorithm to solve the convex hull problem for planar data initially distributed in an arbitrary fashion. In Section V, we show how to combine ideas developed in Section IV with the algorithms given in Section III to derive worst case $\Theta(\log^3 n/(\log \log n)^2)$ time algorithms for the pyramid, tree, and mesh-of-trees. In Section VI, we show that the hypercube algorithm that is developed in Section IV can be modified to yield an optimal worst case $\Theta(\log n)$ time, n-processor, EREW PRAM algorithm for solving the convex hull problem assuming an arbitrary ordering of the planar point data input. In Section VII, we show that the hypercube algorithm that is developed in Section IV can be modified to yield an optimal worst case $\Theta(\log n)$ time algorithm for a fixed degree network assuming arbitrary planar point data input. This is the first algorithm presented that solves the convex hull problem in worst case $\Theta(\log n)$ time on a fixed degree network. Section VIII is the conclusion.

II. Preliminaries

A. Order Notation

Throughout this paper, O, Θ, and Ω notation are used, where Θ is used to mean "order exactly," O is used to mean "order at most," and Ω is used to mean "order at least." That is, given nonnegative functions f and g defined on the positive integers, we write $f = \Theta(g)$ if and only if there are positive constants C_1, C_2, and a positive integer N such that $C_1 * g(n) \leq f(n) \leq C_2 * g(n)$, whenever $n > N$. We write $f = O(g)$ if and only if there is a positive constant C and an integer N such that $f(n) \leq C * g(n)$, for all $n > N$, and we write $f = \Omega(g)$ if and only if there is a positive constant C and an integer N such that $C * g(n) \leq f(n)$, for all $n > N$.

B. Models of Computation

In this section, the models of computation that are used in this paper are defined and some standard terminology is reviewed.

The *communication diameter* of a machine is defined to be the maximum of the minimum distance (number of communication links) between any two processors in the network. Therefore, the communication diameter of a machine gives a lower bound on the running time for problems where data need to be exchanged between processors at maximum distance.

Another method of determining lower bounds for problems that require extensive data movements is by a *wire-counting* (or *wire-cutting* or *cut-set*) argument. For instance, suppose one is concerned with the minimum time necessary to sort or

route data on a particular machine and it can be shown that in the worst case all of the data from one "half" of the machine must be exchanged with all of the data from the other "half" of the machine. If there are w wires that connect the two halves of the machine, then in one unit of time only $2w$ elements can cross these w bidirectional communication wires. Therefore, if each half of the machine has $n/2$ pieces of data, then $\Omega(n/w)$ time is required simply to move data between the two halves of the machine.

1) Mesh Computer: An optimal mesh algorithm for identifying the extreme points of the convex hull is given in [34], [38], and [40]. The description of the mesh is presented in this section for convenience since a number of the other architectures of interest in this paper are mesh-based, e.g., the pyramid, mesh-of-trees, and mesh with reconfigurable bus.

The *mesh computer (mesh) of size n* is a machine with n simple *processing elements* (PE's) arranged in a square lattice. To simplify exposition, it is assumed that $n = 4^c$ for some integer c. For all $i, j \in [0, \cdots, n^{1/2} - 1]$, PE $P_{i,j}$, representing the PE in row i and column j, is connected via bidirectional unit-time communication links to its four *neighbors*, PE's $P_{i\pm1,j\pm1}$, assuming they exist. (See Fig. 1.) Each PE has a fixed number of registers (words), each of size $\Theta(\log n)$, and can perform standard arithmetic and Boolean operations on the contents of these registers in unit time. Each PE can also send or receive a word of data from each of its neighbors in unit time. Each PE contains its row and column indexes, as well as a unique identification register, the contents of which are initialized to the PEs' proximity order index, as discussed below.

Several of the algorithms presented in this paper rely on ordering data with respect to the proximity ordering of processors (which is based on the concept of space-filling curves). There is no single natural ordering of a two-dimensional mesh, so many orderings of processors have been used. Some of the more useful and popular orderings are given in Fig. 2. Notice that snake-like ordering has the useful property that PE's with consecutive numbers in the ordering are adjacent in the mesh, while shuffled row-major ordering has the property that the first quarter of the PE's form one quadrant, the next quarter form another quadrant, etc., with this property holding recursively within each quadrant. This property of shuffled row-major ordering is useful in many applications of a divide-and-conquer approach. Proximity ordering combines the advantages of snake-like and shuffled row-major order. Some properties of proximity ordering follow. Given row and column coordinates of a PE P, in $O(\log n)$ time a single processor can compute the proximity order of P by a binary search technique. Similarly, given a positive integer i, the row and column coordinates of the PE with i as its proximity number can be determined in $O(\log n)$ time by a single processor. Given any positive integers $i < j$, the shuffled row-major property of recursively dividing indexes among quadrants gives the property that the distance from PE number i to PE number j is $O((j - i)^{1/2})$, and that a path of length $O((j - i)^{1/2})$ can be achieved using only PE's numbered from i to j. Furthermore, the PE's numbered from i through j contain a subsquare with more than $(j - i)/8$ PE's.

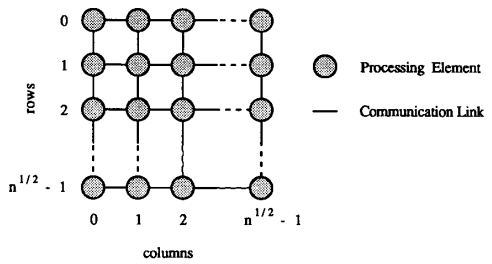


Fig. 1. A mesh computer of size n.

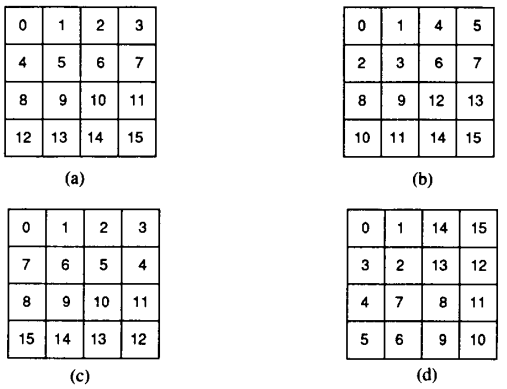


Fig. 2. Indexing schemes for the processors of a mesh. (a) Row-major. (b) Shuffled row-major. (c) Snake-like. (d) Proximity.

The communication diameter of a mesh of size n is $\Theta(n^{1/2})$ as can be seen by examining the distance between PE's in opposite corners of the mesh. This means that if a PE in one corner of the mesh needs data from a PE in another corner of the mesh at sometime during an algorithm, then a lower bound on the running time of the algorithm is $\Omega(n^{1/2})$.

2) Pyramid Computer: A *pyramid computer (pyramid) of size n* is a machine that can be viewed as a full, rooted, 4-ary tree of height $\log_4 n$, with additional horizontal links so that each horizontal *level* is a mesh. It is often convenient to view the pyramid as a tapering array of meshes. A pyramid of size n has at its base a mesh of size n, and a total of $\frac{4}{3}n - \frac{1}{3}$ PE's. The levels are numbered so that the base is level 0 and the apex is level $\log_4 n$. A PE at level i is connected via bidirectional unit-time communication links to its nine neighbors (assuming they exist): four siblings at level i, four children at level $i - 1$, and a parent at level $i + 1$. (A sample pyramid is given in Fig. 3.) It is assumed that each PE has a fixed number of words (registers), each of size $\Theta(\log n)$, and that arithmetic, Boolean, and communication operations with a neighbor take unit time. Each PE contains registers with its row, column, and level coordinates, the concatenation of which provides a unique label for the PE.

Notice that the communication diameter of a pyramid computer of size n is $\Theta(\log n)$. This is true since any two processors in the pyramid can exchange information through the apex. (The reader is referred to [40] for $\Theta(\log n)$ time algorithms that solve a variety of problems on a pyramid of size n.) Of course, if too much data are trying to be passed

Ncube [21], FPS [20], and Ametek [5]. communication diameter. A PRAM is often described in terms

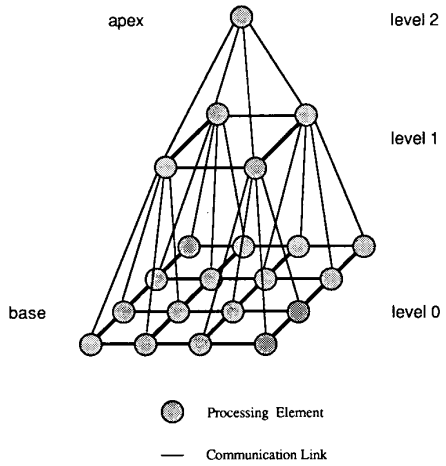


Fig. 3. A pyramid computer of size 16.

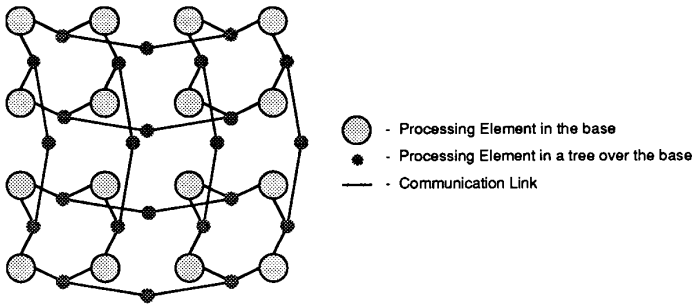


Fig. 4. A mesh-of-trees of base size $n = 16$. Note: The mesh connections have been omitted for clarity.

through the apex, then the apex becomes a bottleneck. In [39] and [40], it is shown that for a variety of problems on a pyramid computer of size n, the $\Omega(\log n)$ lower bound is overly optimistic and can be replaced by a bound closer to $n^{1/4}$. Notice that efficient pyramid algorithms must avoid operations that require extensive data movement, such as sorting or routing all of the data in the base, since a simple wire-counting argument shows that $\Omega(n^{1/2})$ time is required. To see this, consider the number of wires crossing the middle of the pyramid versus the number of items that potentially must move from one half to the other. In the base of the pyramid, there are $n^{1/2}$ wires crossing the middle of the pyramid, in the next level there are $n^{1/2}/2$ such wires, and so on, giving the total number of wires crossing the middle of a pyramid of size n to be $\sum_{i=0}^{\log_4(n)-1} n^{1/2}/2^i$, which is $2n^{1/2} - 2$. Since all n pieces of data that initially reside in the base of the pyramid may need to cross from one side of the base mesh to the other, then $\lceil n/(2n^{1/2} - 2) \rceil$ time units, or $\Omega(n^{1/2})$ time, is required just to get data across the middle of the pyramid. Pyramid computer projects have been proposed [57] and several are under construction [10], [11], [14], [18], [51], [55].

3) Mesh-of-Trees Architecture: A *mesh-of-trees of base size* n, where n is an integral power of 4, has a total of $3n - 2n^{1/2}$ PE's. n of these are base PE's arranged as a mesh of size n. Above each row and above each column of the mesh is a perfect binary tree of PE's. Each row (column) tree has as its leaves an entire row (column) of base PE's. All row trees are disjoint, as are all column trees. Every row has exactly one leaf PE in common with each column tree. Fig. 4 shows a sample mesh-of-trees. Each base processor is connected to six neighbors (assuming they exist): four in the base mesh, a parent in its row tree, and a parent in its column tree. Each PE in a row or column tree that is neither a leaf nor a root is connected to exactly three neighbors in its tree: a parent and two children. Each root in a row or column tree has its two children as neighbors. It is assumed that each PE has a fixed number of words (registers), each of size $\Theta(\log n)$, and that all arithmetic, Boolean, and communication operations with a neighbor take unit time. Each PE contains identity registers with its row, column, and level coordinates (the base being level 0), the concatenation of which provides a unique label for the PE.

Like the pyramid, the mesh-of-trees also has a communication diameter proportional to the logarithm of the number of base PE's. Also, like the pyramid, a simple wire-counting argument shows that for operations that require extensive data movement, such as sorting or routing, $\Omega(n^{1/2})$ time is required since only $2n^{1/2}$ wires cross the middle of the mesh-of-trees. However, due to the multiple paths available between proces-

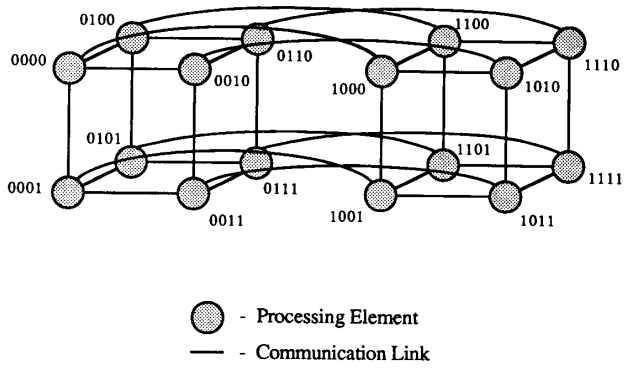


Fig. 5. A hypercube of size $n = 16$ with the processors labeled using a binary representation.

sors, the mesh-of-trees is often able to provide solutions to problems that are more efficient than those possible for the pyramid [40]. In fact, while it is of no use for the convex hull problem considered in this paper, [40] has shown that the mesh-of-trees can sort a restricted amount of data given in certain configurations in $\Theta(\log n)$ time. The mesh-of-trees is a very useful architecture in VLSI because it embeds nicely into the plane [58], although no significant mesh-of-trees has been built yet. It should be noted that the mesh-of-trees is not always defined to include the connections between base PE's, but it is easy to show that these additional connections do not change the planar embedding properties of the mesh-of-trees.

4) Hypercube: A *hypercube of size n*, where n is an integral power of 2, has n PE's indexed by the integers $\{0, \cdots, n-1\}$. Viewing each integer in the index range as a $\log_2 n$ bit string, two PE's are connected via a bidirectional communication link (i.e., they are neighbors) if and only if their indexes differ by exactly one bit. A hypercube of size n is created recursively from two hypercubes of size $n/2$ by labeling each hypercube of size $n/2$ identically and independently with the indexes $\{0, \cdots, (n/2) - 1\}$, and then appending a 1 in front of the bit strings of one of the cubes and a 0 in front of the other, which "creates" a new link from each PE in one cube to the corresponding PE in the other cube. See Fig. 5. It is assumed that each PE has a fixed number of words (registers), each of size $\Theta(\log n)$, and that all arithmetic, Boolean, and communication operations with a neighbor take unit time.

It is easy to see that like the mesh-of-trees and pyramid, the communication diameter of a hypercube of size n is $\Theta(\log n)$. However, unlike the mesh-of-trees or pyramid, a wire-counting argument only shows that $\Omega(1)$ time is required for operations that require extensive data movement, since there are $n/2$ wires that connect two subhypercubes of size $n/2$ in a hypercube of size n. This is encouraging, and in [40] it is shown that many problems can be solved much more efficiently on a hypercube than on other machines. A variety of hypercubes are marketed commercially, including fine-grained machines such as the Connection Machine [22], and medium-grained machines by companies such as Intel [23], Ncube [21], FPS [20], and Ametek [5].

5) Mesh with Reconfigurable Bus: For many applications, it is desirable to have an interconnection scheme that may be reconfigured during an algorithm. We consider a reconfigurable array of processing elements that combines the advantages of a number of architectures including the mesh, pyramid, mesh-of-trees, and meshes with broadcast buses. (For descriptions of a mesh with multiple broadcast buses, the reader is referred to [27] and [54].) The *mesh with reconfigurable bus (reconfigurable mesh) of size n* consists of an $n^{1/2} \times n^{1/2}$ array of processors connected to a grid-shaped reconfigurable broadcast bus, where each processor has four locally controllable bus *switches*, as shown in Fig. 6. Other than the buses and switches, the reconfigurable mesh is similar to the standard mesh in that it has $O(n)$ area, under the assumption that processors, switches, and individual links have constant size. In one unit of time, each processor can perform standard arithmetic and Boolean operations on its own data, can set any of its four switches, and can send and receive a piece of data from the bus. The restriction is that within any maximally connected subbus, simultaneous writes to the subbus are only allowed if the same piece of information is being written, otherwise simultaneous writes to a subbus are prohibited. This differs from the model used in [31]–[33] in which simultaneous writes are strictly prohibited.

Notice that the switches allow the broadcast bus to be divided into subbuses, including row and column buses, a bus within each disjoint submesh, and so forth, where each subbus can function as a smaller mesh with a reconfigurable bus. This architecture is much more general than the mesh augmented with row and column broadcasts, and still requires only $O(n)$ area to layout an n-processor mesh with reconfigurable bus. Furthermore, except for small differences, the reconfigurable bus is used as an interconnection network in the *polymorphic-torus network* [29] and in the latest version of the *content addressable array parallel processor (CAAPP)*, which is the lowest level of the *image understanding architecture (IUA)* [59] There are also similarities between the reconfigurable mesh and the CHiP architecture [53].

6) PRAM: A *parallel random access machine (PRAM)* is an idealized parallel model of computation, with a unit-time communication diameter. A PRAM is often described in terms

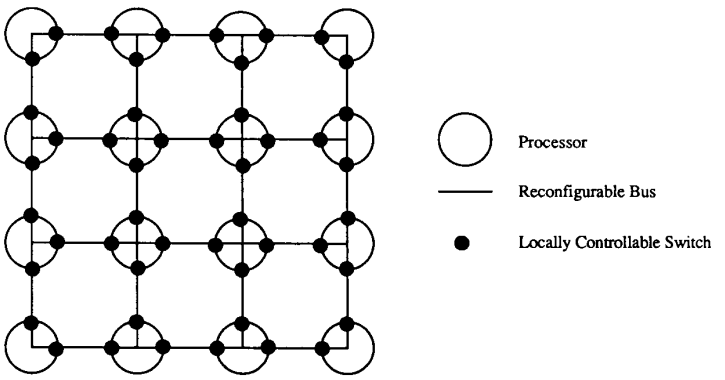


Fig. 6. A reconfigurable mesh of size $n = 16$.

of a machine consisting of identical processors and a global memory, where all processors have unit-time access to any memory location.

Three of the common variations of the PRAM are now described. A *concurrent read, exclusive write (CREW) PRAM* permits multiple processors to read data from the same memory location simultaneously, but permits only one processor at a time to attempt to write to a given memory location. A *concurrent read, concurrent write (CRCW) PRAM* permits concurrent reads as above, but allows several processors to attempt writing to the same memory location simultaneously, with some tie-breaking scheme used so that only one of the competing processors succeeds in the write. An *exclusive read, exclusive write (EREW) PRAM* is the most restrictive version of a PRAM in that only one processor can read and write from a given memory location at a given time. Some bus-based machines with a small number of processors, such as those marketed by Alliant, ELXSI, Encore, and Sequent, to name a few, are conceptually similar in design to a PRAM.

7) Modified AKS Network: Given n items distributed no more than one per processor, [2] gives an n-processor $\Theta(\log n)$ degree network and an $n \log n$ processor bounded degree network that sorts the items in worst case $\Theta(\log n)$ time. In [28], Leighton gives a construction that when combined with this AKS network gives an $O(n)$ processor bounded degree network that sorts n items, distributed no more than one per processor, in worst case $O(\log n)$ time. This processor organization will be referred to as *LeiAKS* in this paper.

The *cube-connected cycles network (CCC) of size n* [48] is a fixed degree network with n processors capable of simulating with constant delay a hypercube of size n for a wide variety of algorithms, including $\Theta(\log n)$ time bitonic merge and broadcasting. For $n = 2^k$, k an integer of the form $k = r + 2^r$, the processors of a CCC of size n are grouped into 2^{k-r} cycles, where each cycle consists of 2^r circularly connected processors, and where the cycles are interconnected as a $(k - r)$-dimensional hypercube. One drawback of a cube-connected cycles network is that when trying to simulate recursive divide-and-conquer hypercube algorithms, CCC's do not partition into smaller CCC's. The problem is that cycles get partitioned into paths. This can be remedied by introducing additional interconnection links to form the desired cycles. Furthermore, these additional interconnection links can be added so that this *modified cube-connected cycles network* will remain a fixed degree network.

Constructing a processor organization that gives each of n processors LeiAKS connections, as well as modified cube-connected cycles connections, gives a processor organization that can sort in $\Theta(\log n)$ time and perform hypercube operations and divide-and-conquer techniques that are used in Section IV in the required time. This organization will be referred to as a *modified AKS network*, and is defined more fully in [41].

C. Convex Hull

In this paper, we give efficient algorithms for identifying the extreme points that represent the convex hull of a set S of n or fewer planar points, initially distributed one point per processor. The *convex hull* of a set S of points, denoted *hull*(S), is defined to be the minimum convex set containing S. A point $P \in S$ is an *extreme point* of S if $P \notin$ *hull*($S - P$). For several of the algorithms presented in this paper, it will be useful to impose an ordering on the extreme points of S. The ordering will be in a *counterclockwise* fashion, starting with the easternmost point. Since the number of points under consideration is finite, and we assume that no two points have the same x-coordinate, then notice that there must be a unique easternmost point.

In general, if S is finite, then *hull*(S) is a convex polygon, and the extreme points of S are the corners of this polygon. The edges of this polygon will be referred to as the *edges of the hull*(S). (See Fig. 7.) We say that the *extreme points of S have been identified*, and hence *hull*(S) has been identified, if

1) for each PE P_i containing a point of S, P_i has a Boolean variable "extreme" that is true if and only if the point contained in P_i is an extreme point of S, and

2) for each PE P_i containing an extreme point of S, P_i contains the position of its point in the counterclockwise ordering, the total number of extreme points, and its adjacent extreme points in the counterclockwise ordering.

That is, the term *identify* is used throughout this paper to mean *detect (mark)* and *enumerate (number in counter-*

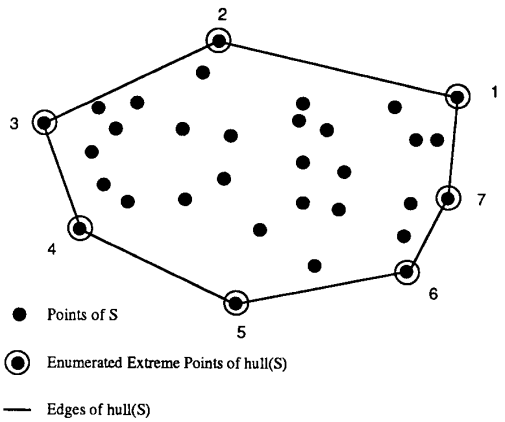


Fig. 7. Convex hull of S.

clockwise order). Finally, we define the *convex hull problem* to be the problem of identifying the extreme points of a given set of planar points.

D. Lower Bounds

In [47], it is shown that sorting n elements can be reduced to the problem of identifying the extreme points of the convex hull of a set S of n planar points. Therefore, given a set S of n planar points distributed arbitrarily, a serial machine will require $\Omega(n \log n)$ time in the worst case to identify (i.e., mark and enumerate) the extreme points of hull(S). Furthermore, this result shows that any reasonable $O(n)$ processor parallel machine requires $\Omega(\log n)$ worst case time to identify the extreme points of hull(S) when the points of S are distributed in an arbitrary fashion one per PE.

Proposition 1: Given a set S of n planar points stored arbitrarily one per processor on machine M that takes $\Omega(T_M(n))$ worst case time to sort n elements, a lower bound on the worst-case time to identify the extreme points of hull(S) on M is $\Omega(T_M(n))$. ■

Therefore, for mesh-based machines with $\Theta(n)$ processors, such as the mesh, mesh-of-trees, and pyramid, $\Omega(n^{1/2})$ worst case time is required to solve the convex hull problem for a set of planar inputs arbitrarily distributed throughout the base processors of the machine. These worst case lower bounds are easy to derive through wire-counting arguments, as discussed previously in this section. Furthermore, [38] and [40] give optimal $\Theta(n^{1/2})$ worst case time algorithms for solving the convex hull problem on these machines with arbitrary planar point data input. Notice also that an easy wire-counting argument shows that a tree machine with n pieces of data stored one per leaf processor must take $\Omega(n)$ time to sort the data in the worst case. Therefore, the worst case running time to solve the convex hull problem for planar point data that is initially distributed in an arbitrary fashion one per leaf processor on a tree with n leaf processors is $\Omega(n)$. Finally, the only provable lower bound on the worst case running time to solve the convex hull problem for planar point data distributed arbitrarily among the n processors of a hypercube is $\Omega(\log n)$, based on the communication diameter. However, currently the best worst case time to sort on a hypercube of size n is $\Theta(\log^2 n)$ [9], which means that currently all algorithms bounded by sorting take $\Omega(\log^2 n)$ worst case time on a hypercube of size n.

An interesting result is presented in [16], which shows that $\Omega(\log n)$ is the best possible worst case running time for a machine to solve the convex hull problem even if arbitrarily many processors are allowed.

Proposition 2 [16]: Given a set S of n planar points stored arbitrarily one per processor on a parallel machine with arbitrarily many processors, a lower bound on the worst case time to identify the extreme points of hull(S) is $\Omega(\log n)$. ■

It should be noted that this proposition applies to the problem of *identifying* (i.e., detecting *and* enumerating) the extreme points on a parallel machine. For the problem of simply detecting the extreme points, it is easy to construct a $\Theta(1)$ time CRCW PRAM algorithm that requires only polynomially many processors.

An identification algorithm for the CREW PRAM is given in [6] that is optimal with respect to both time and space, and is used as a point of reference for the algorithms given throughout this paper.

Proposition 3 [6]: Given a set S of n planar points stored arbitrarily one per processor on a CREW PRAM with n processors, the extreme points of hull(S) can be identified in $\Theta(\log n)$ time. ■

III. Parallel Algorithms for Ordered Input

In this section, we describe a general parallel algorithm for identifying the extreme points of the convex hull of a set S of n planar points distributed in an ordered fashion throughout the processors of the machine. As mentioned previously, the points are assumed to be ordered so that the x-coordinate of the point in PE P_i is less than the x-coordinate of the point in PE P_j, for $i < j$. After describing the general algorithm, we give efficient implementations of it on the pyramid, tree, mesh-of-trees, and mesh with reconfigurable bus. It should be noted that in Section V, we show how to incorporate techniques from Section IV into the algorithms presented in this section to further improve the running times for the pyramid, tree, and mesh-of-trees.

The general parallel algorithm that we use to identify the extreme points of a set S of n planar points is given below in Algorithm Identify_Hull.

Algorithm Identify_Hull:

1. Divide the set S of n planar points into two subsets S_1 and S_2, each of size $n/2$, so that all points of S_1 have x-coordinates less than those of S_2, and such that $S = S_1 \cup S_2$.
2. Recursively identify the extreme points of hull(S_1) and hull(S_2).
3. Identify the upper and lower common tangent lines between hull(S_1) and hull(S_2). See Fig. 8. It should be noted that an extreme point p_k of S_1, with p_{k-1} and p_{k+1} as its preceding and succeeding extreme points, respectively, with respect to the counterclockwise ordering of the extreme points of S_1, is the left endpoint of the upper common tangent line between S_1 and S_2 if and only if no points of S_2 lie above the line $\overline{p_{k+1}p_k}$, while at least one point of S_2 lies above the line $\overline{p_kp_{k-1}}$. (Recall that the extreme points are labeled in

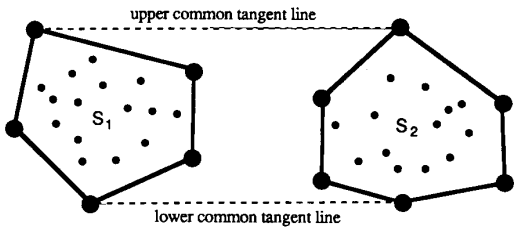


Fig. 8. Upper and lower tangent lines between linearly separable sets S_1 and S_2.

counterclockwise fashion.) Similar remarks can be made about the other three endpoints.

4. Eliminate all extreme points between the common tangent lines (i.e., all extreme points of S_1 and S_2 that are inside the quadrilateral formed by the four endpoints representing the common tangent lines) and renumber the remaining extreme points.

The remainder of this section is concerned with developing efficient implementations of this algorithm for a variety of models of computation, some of which will be improved further in Section V.

A. Pyramid and Tree

Assume the points of S initially reside in the base of a pyramid of size n ordered by x-coordinate with respect to the proximity order indexing scheme of the base mesh. We now give the implementation details corresponding to each step of Algorithm Identify_Hull.

1. Due to the ordering of points in the base of the pyramid, those points in processors labeled 0, 1, $\cdots$, $(n/2) - 1$ (conceptually corresponding to the left half of the base of the pyramid) will represent S_1, while those points in processors labeled $n/2$, $(n/2) + 1$, $\cdots$, $n - 1$ (conceptually corresponding to the right half of the base of the pyramid) will represent S_2.

2. Recursively identify the extreme points of hull(S_1) and hull(S_2).

3. One of the keys to an efficient pyramid algorithm is in the identification of the upper and lower common tangent lines between hull(S_1) and hull(S_2). Assume that there are n_1 extreme points in S_1 and n_2 extreme points in S_2. Further, assume the extreme points of S_1 are labeled 1, 2, $\cdots$, n_1. Since the extreme points of S_1 and S_2 have been identified in Step 2, each processor that is responsible for an extreme point of S_1 (S_2) knows

a) its number in the counterclockwise ordering of the extreme points of S_1 (S_2),

b) the total number of extreme points, n_1 (n_2), of S_1 (S_2), and

c) the points and processors that correspond to its preceding and succeeding extreme points with respect to the counterclockwise ordering of the extreme points of S_1 (S_2).

We now describe an algorithm to determine the left endpoint of the upper common tangent line between S_1 and S_2.

Pipeline through the apex of the pyramid a logarithmic number of equidistantly placed extreme points from S_1, with respect to the counterclockwise ordering, down to the base processors of the right side of the pyramid that are responsible for S_2. Let $i = \lfloor n_1/\log_2 n_1 \rfloor$. Specifically, in $\Theta(\log n)$ time send copies of the extreme points, along with their preceding and succeeding extreme points, labeled $1, 1 + i, 1 + 2i, \cdots, 1 + i\lfloor \log_2 (n_1) - 1 \rfloor$ through the apex and down to the right side of the pyramid where the extreme points of S_2 reside in the base. As each extreme point p_k sent from S_1, with p_{k-1} and p_{k+1} its preceding and succeeding extreme points, arrives at a base processor on the right side of the pyramid, the processor receiving the extreme point will determine if the point of S_2 that it is maintaining is above the line $\overline{p_{k+1}p_k}$. The processor will also determine if its point is above the line $\overline{p_kp_{k-1}}$. Each of these results are recorded with a single bit in the processor. So, each base processor on the right side of the pyramid maintains a $\Theta(\log n)$-bit vector to keep track of the intervals of extreme points that it considers candidates for containing the left extreme point of the upper tangent line between S_1 and S_2.

After all $\lfloor \log_2 n_1 \rfloor$ points have reached the base processors on the right side of the pyramid, the base processors on the right side of the pyramid pipeline the bits of their vectors up to the apex in pairs (each pair corresponds to a single extreme point of S_1), logically ORing corresponding entries along the way, until the apex knows the single interval of extreme points of S_1 corresponding to possible candidates for the left endpoint of the upper common tangent line between S_1 and S_2.

The apex broadcasts to the base processors on the left side of the pyramid the indexes of the two extreme points that determine the single interval of $O(n_1/\log n_1)$ extreme points that are candidates for the left endpoint of the upper common tangent line. This binary search process continues recursively in search of the left endpoint of the upper common tangent line, where at each iteration of the search, the number of extreme points under consideration is reduced by a factor of $\Theta(\log n_1)$.

A similar operation is performed to find the right extreme point of the upper common tangent line, and both extreme points that correspond to the lower common tangent line between S_1 and S_2. The worst case running time for this algorithm to find an endpoint of a common tangent line between S_1 and S_2 is given by $T(n) = T(n/\log n) + \Theta(\log n)$, which is $\Theta(\log^2 n/\log \log n)$.

4. Once the four extreme points corresponding to the endpoints of the two tangent lines are known, they are broadcast, along with their counterclockwise ordering number restricted to either S_1 or S_2 and their preceding and succeeding extreme points restricted to S_1 or S_2, to all base processors. This is accomplished by a straightforward bottom-up top-down tree-like report and broadcast in $\Theta(\log n)$ time. All PE's containing an extreme point of S_1 or S_2 can now compute in $\Theta(1)$ time whether or not they remain an extreme point of S, and if so they may determine their possibly new number and total number of extreme points of S, as well as their possibly new preceding and succeeding extreme points.

Therefore, the worst case running time for this algorithm to identify the extreme points of n planar points is given by the recurrence $T(n) = T(n/2) + \Theta(\log^2 n/\log \log n)$, which is $\Theta(\log^3 n/\log \log n)$. Since this pyramid algorithm only makes

use of the child–parent links of the pyramid and not of the mesh links, the algorithm is easily extended to a tree machine.

Theorem 4: Given a set S of n planar points ordered by x-coordinate in the base of a pyramid (leaves of a tree) of size n, the implementation of Algorithm Identify_Hull given above will identify the extreme points of hull(S) in $\Theta(\log^3 n/\log\log n)$ time. ■

B. Mesh-of-Trees

An algorithm to identify the extreme points of the convex hull of a set S of n points, initially distributed ordered by x-coordinate with respect to the proximity ordering in the base mesh of a mesh-of-trees of base size n, follows directly from the algorithm just described in Section III-A for the pyramid and tree architectures. Data movement operations that perform standard report and broadcast operations, as well as techniques for pipelining data throughout the mesh-of-trees, are straightforward and can be found in [40]. Simply perform the abstract data movement operations implied in the algorithm of the preceding section to arrive at a $\Theta(\log^3 n/\log\log n)$ time mesh-of-trees algorithm for solving the convex hull problem.

Theorem 5: Given a set S of n planar points ordered by x-coordinate in the base of a mesh-of-trees of base size n, the implementation of Algorithm Identify_Hull given in this section will identify the extreme points of hull(S) in $\Theta(\log^3 n/\log\log n)$ time. ■

C. Mesh with Reconfigurable Bus

In this section, we describe an algorithm to identify the extreme points of the convex hull of a set S of n points, initially distributed on point per processor on a reconfigurable mesh of size n, where the points are ordered by x-coordinate with respect to the proximity ordering of the mesh. The algorithm presented in this section follows the spirit of Algorithm Identify_Hull given in Section III. However, the implementation is tailored to this interesting architecture so as to obtain an algorithm that finishes in $\Theta(\log^2 n)$ worst case time. This improves on the running times of the convex hull algorithms given for the pyramid, tree, and mesh-of-trees by a factor of $\Theta(\log n/\log\log n)$.

Using Algorithm Identify_Hull as a template, we give the detailed implementation of an algorithm to identify the extreme points of a set S of n planar points on a mesh with reconfigurable bus of size n.

1. In order to obtain a proper subdivision of points into subreconfigurable meshes, the switches of the reconfigurable bus are set so as to partition the reconfigurable mesh with respect to the proximity ordering of the processors. This partitioning will have processors labeled $0, 1, \cdots, (n/2) - 1$ (conceptually corresponding to the left half of the mesh with a reconfigurable bus) represent S_1, while those points in processors labeled $n/2, (n/2) + 1, \cdots, n - 1$ (conceptually corresponding to the right half of the mesh with a reconfigurable bus) will represent S_2. This step is complete in $\Theta(1)$ time.

2. The problem can now be solved recursively within each submesh with reconfigurable bus of size $n/2$.

3. The endpoints of the upper and lower common tangent lines between hull(S_1) and hull(S_2) can be identified in $\Theta(\log n)$ time by a binary search technique for each of the four required points. For instance, in order to determine the extreme point that corresponds to the left endpoint of the upper tangent line between S_1 and S_2, do the following.

a) Suppose there are n_1 extreme points of S_1 labeled $1, 2, \cdots, n_1$. Let $l = 1$ and $r = n_1$.

b) Let $k = \lfloor (l + r)/2 \rfloor$. Use the bus to broadcast extreme point p_k of S_1 to the processors on the right side of the reconfigurable mesh that are responsible for S_2.

c) Using simultaneous writes to the bus, all processors on the right side of the reconfigurable mesh that maintain a point above the line $\overline{p_{k+1}p_k}$, broadcast an identical message on the bus to all n processors.

d) If there is at least one such point of S_2 that is above $\overline{p_{k+1}p_k}$, then the binary search continues on the points labeled $k + 1, k + 2, \cdots, r$. [i.e., set $l = k + 1$ and return to Step 3b)].

e) If there are no points of S_2 above $\overline{p_{k+1}p_k}$, then using simultaneous writes to the bus, all processors on the right side of the reconfigurable mesh that maintain a point above the line $\overline{p_kp_{k-1}}$, broadcast an identical message on the bus to all n processors.

i) If there are no such points of S_2 above $\overline{p_kp_{k-1}}$, then the binary search continues on the set of points labeled $l, l + 1, \cdots, k - 1$. [i.e., set $r = k - 1$ and return to Step 3b)].

ii) If there is at least one point of S_2 above $\overline{p_kp_{k-1}}$, then p_k is the left endpoint of the upper tangent line between S_1 and S_2. Notice that p_k has the property that all points of S_2 lie below the line $\overline{p_{k+1}p_k}$, while at least some points of S_2 lie above the line $\overline{p_kp_{k-1}}$.

After no more than $\lceil \log_2 n_1 \rceil$ iterations, an extreme point of S_1 will be found that is the left endpoint of the upper tangent line between S_1 and S_2. Since each of the $O(\log n)$ broadcasts takes $\Theta(1)$ time, this step is complete in $\Theta(\log n)$ time in the worst case. The algorithm is trivially modified to find each of the other three common tangent points, so that in $\Theta(\log n)$ worst case time, the endpoints of the upper and lower tangent lines between S_1 and S_2 will be known.

4. Once the four extreme points corresponding to the endpoints of the two tangent lines are known, they are broadcast to all PE's in $\Theta(1)$ time. All PE's containing an extreme point of S_1 or S_2 can now compute in $\Theta(1)$ time whether or not they remain an extreme point of S, and if so they may determine their possibly new number and total number of extreme points of S, as well as their possibly new preceding and succeeding extreme points. This step takes $\Theta(1)$ time.

Therefore, the worst case running time for this algorithm to identify the extreme points of n planar points is given by the recurrence $T(n) = T(n/2) + \Theta(\log n)$, which is $\Theta(\log^2 n)$.

Theorem 6: Given a set S of n planar points ordered by x-coordinate in the processors of a mesh with reconfigurable bus of size n, the extreme points of S can be identified by the above algorithm in $\Theta(\log^2 n)$ time. ■

IV. Hypercube Algorithms for Ordered and Unordered Input

This section introduces an optimal $\Theta(\log n)$ time algorithm for a hypercube of size n to identify the extreme points of a set

of n planar points, distributed one point per processor in an ordered fashion. This algorithm is then extended to give an expected $\Theta(\log n)$ time hypercube algorithm to solve the convex hull problem for a set of planar input points that are initially distributed in an arbitrary fashion, one point per processor.

Given an input set of size n distributed one element per base processor on a mesh-of-trees of base size n, [40] shows that any algorithm that runs in $T(n)$ time on the mesh-of-trees of base size n can be simulated to run in $cT(n)$ time, c a constant, on a hypercube of size n. This follows from the mapping given in [37] that shows how to embed a mesh-of-trees of base size n into a hypercube of size n so that adjacent processors of the mesh-of-trees are mapped to hypercube processors that are at most two communication links apart, and so that no hypercube processor is responsible for more than three mesh-of-trees processors. In this section, we show that by exploiting the hypercube topology, we are able to solve the convex hull problem substantially faster on a hypercube than by direct simulation of a mesh-of-trees algorithm.

The hypercube algorithm presented in this section gives a solution to the convex hull problem for ordered input that conceptually differs only slightly from the algorithms given in Section III for the pyramid, tree, mesh-of-trees, or mesh with reconfigurable bus. One change is that the hypercube network allows us to do more than one search at a time. Another change is that the endpoints of upper and lower common tangent lines are determined by a comparison of slopes of hull edges, which replaces the bottleneck-restricted binary search technique used in Section III. Specifically, suppose $\overline{p_iq_j}$, $p_i \in S_1$, $q_j \in S_2$, is the upper tangent line between convex sets S_1 and S_2, as in Fig. 8. The algorithm presented in this section uses the fact [45] that the slope of $\overline{p_iq_j}$ is between

1) the slope of $\overline{p_ip_{i-1}}$ and the slope of $\overline{p_{i+1}p_i}$, and

2) the slope of $\overline{q_{j-1}q_j}$ and the slope of $\overline{q_jq_{j+1}}$.

That is, in this section we just make comparisons to the specific points that will answer the query, rather than with all points. The algorithm also relies heavily on data movement operations that exploit properties of the hypercube. The algorithm follows.

1. The set S, which consists of n planar points, is divided into $n^{1/4}$ subsets, $S_1, S_2, \cdots, S_{n^{1/4}}$, each of size $n^{3/4}$, such that $S = \cup_{i=1}^{n^{1/4}} S_i$ and the x-coordinates of all points in S_i are less than the x-coordinates of all points in S_j, for $i < j$. Define *region* R_i to be the processors responsible for set S_i. It is assumed that the set S_i is stored in the ith subhypercube of size $n^{3/4}$ and that this ordering holds recursively within each such region of size $n^{3/4}$. It should be noted that the idea of partitioning into regions of size square root in the size of the input has been used previously in conjunction with PRAM algorithms (e.g., [6]). However, to the best of our knowledge, data division techniques such as this have only recently been explored with respect to the hypercube (c.f., [17], [40]).

2. For each region R_i, $1 \le i \le n^{1/4}$, recursively identify the extreme points of hull(S_i).

3. In $\Theta(\log n)$ time, each region R_i, $1 \le i \le n^{1/4}$, can determine p_{w_i} and p_{e_i}, the westernmost and easternmost extreme points, respectively, of S_i. This is accomplished in $\Theta(\log n)$ time by performing a report and broadcast operation within R_i so that all processors of R_i know p_{w_i} and p_{e_i}. Each processor of R_i that contains an extreme point of S_i can now decide in $\Theta(1)$ time whether its extreme point lies on, above, or below $\overline{p_{w_i}p_{e_i}}$.

Steps 4–6 are performed twice, once for those extreme points in R_i that lie on or above $\overline{p_{w_i}p_{e_i}}$, and once for those points in R_i that lie on or below $\overline{p_{w_i}p_{e_i}}$, $1 \le i \le n^{1/4}$. The description that follows in Steps 4–6 will be concerned only with those extreme points that lie on or above their respective lines, with the description being similar for the extreme points that lie on or below their line.

4. The goal of this step is to determine for each region R_i, the upper (and when performed a second time, the lower) tangent line between its set of points S_i, and every other set of points S_j, for $i \neq j$. This is similar to the approach used in a PRAM algorithm given in [6] in which $n^{1/2}$ regions, each containing $O(n^{1/2})$ points, are used. However, the number and sizes of the regions, as well as the specific implementation given here, are very dependent on the properties and data movement operations available for the hypercube. The method for determining the tangent lines also differs from the binary search technique used in [6]. Furthermore, a direct simulation of the algorithm given in [6] would not yield an efficient hypercube algorithm.

a) For each region R_i, *mark* $n^{1/4}$ extreme points that are equidistantly spaced with respect to their number in the counterclockwise ordering of the extreme points of S_i. For instance, if region R_i has n_i extreme points, and $l = \lfloor n_i/n_i^{1/4} \rfloor$, then mark points $1, 1 + l, 1 + 2l, \cdots, 1 + l\lfloor n_i^{1/4} - 1 \rfloor$ in region R_i. For each marked extreme point p_k, with p_{k-1} and p_{k+1} as its preceding and succeeding extreme points, respectively, with respect to the counterclockwise ordering of extreme points in R_i, create two *slope records*. One record consists of the slope of line $\overline{p_{k+1}p_k}$, p_{k+1}, p_k, and the index of the processor that contains the point, while the other record consists of the slope of line $\overline{p_kp_{k-1}}$, p_k, p_{k-1}, and the index of the processor.

b) Define a *source region* to be a region that *sends* slope records to other regions, and a *host region* to be a region that *receives* such records from source regions. During Step 4 of this algorithm, every region will act as both a source and a host region.

Every region R_i acts as a source region and sends its $O(n^{1/4})$ slope records to every other region R_j, $i \neq j$. Every region R_i acts as a host region and receives the slope records from every source region R_j, $i \neq j$. Since each of the $n^{1/4}$ regions is sending $O(n^{1/4})$ records to each of the other $n^{1/4} - 1$ regions, there are only $O(n^{3/4})$ records being routed in the hypercube of size n. Using data movement operations given in [43], the necessary routing can be accomplished in $\Theta(\log n)$ time. The routing is performed so that within each host region, all $O(n^{1/2})$ records arrive ordered together by the slope field.

c) Each host region creates two slope records for each of its $O(n^{3/4})$ extreme points. Using a bitonic merge within each host region, in $\Theta(\log n)$ time merge the $O(n^{3/4})$ ordered slope records of the host with the $O(n^{1/2})$ ordered slope records sent by the source regions.

d) Perform a $\Theta(\log n)$ time parallel prefix operation within each host region so that each source record determines the

largest slope of a hull edge of the host region, and its associated points, that is smaller than its slope. Similarly, perform a $\Theta(\log n)$ time parallel "postfix" operation (a backwards prefix operation) so that each source record determines the smallest slope of a hull edge of the host region, and its associated points, that is larger than its slope.

In $\Theta(1)$ time, every processor residing in host region R_j that contains a source record representing an extreme point p from source region R_i, $i \neq j$, can determine

i) the tangent line to S_j passing through point p, and

ii) whether p is on, to the left of, or to the right of (with respect to the counterclockwise ordering of extreme points in S_i) the upper common tangent line between S_i and S_j.

e) Using a $\Theta(\log n)$ time routing step, the source records return to their originating regions ordered by destination region by slope. Within each source region, by comparing neighboring elements, it is determined for each destination region the interval of points, delimited by consecutive marked extreme points, that contain an endpoint of an upper common tangent line between the source region and the destination region.

It is important to note that our algorithm compares marked extreme points of S_i with *all* extreme points of S_j, for all $i \neq j$, in order to determine the subset of points in S_i that contains the endpoint of the upper common tangent line between S_i and S_j. This avoids a possible pitfall [7] that would arise if one tried determining the subset based on comparing marked extreme points of S_i with only the marked extreme points of S_j, $i \neq j$.

f) Perform Steps 4a)–4e) twice more, where the returning information is used to determine which sets of points are to be compared between regions. Notice that during each of the two additional applications of Steps 4a)–4e), a source region will send a possibly distinct set of $O(n^{1/4})$ points to each of the other regions. However, there will still be only $O(n^{3/4})$ pieces of data being routed in the hypercube of size n, since each of the $n^{1/4}$ regions is sending $O(n^{1/4})$ records to each of the other $n^{1/4} - 1$ regions. Therefore, the routing can still be accomplished in $\Theta(\log n)$ time. After a total of three applications of Steps 4a)–4e), all processors of region R_i, $1 \leq i \leq n^{1/4}$, know the extreme points representing the upper common tangent line (and lower common tangent line after Step 4 is performed a second time) between S_i and all S_j, $i \neq j$.

The running time of this step is $\Theta(\log n)$, and is dominated by the time to perform data movement operations such as routing, broadcasting, reporting, parallel prefix, and bitonic merge.

5. In $\Theta(\log n)$ time each of the regions can now decide how many, and which of its extreme points, are extreme points of S. In order for region R_i, $1 \leq i \leq n^{1/4}$, to determine the interval of its extreme points between its westernmost and easternmost extreme points that are extreme points of S, the following is performed within R_i.

a) Determine the minimum slope of a tangent line between R_i and R_j, for $j < i$ (i.e., those regions to the left of R_i). Let p_l be the extreme point of R_i that is an endpoint of this common tangent line.

b) Determine the maximum slope of a tangent line between R_i and R_j, for $j > i$ (i.e., those regions to the right of R_i). Let p_r be the extreme point of R_i that is an endpoint of this common tangent line.

c) If p_r is to the left of p_l, or $p_r = p_l$ and the angle open to the top, formed by these two line segments, is less than 180°, then no points of R_i are extreme points of S. Otherwise, those extreme points of R_i between p_l and p_r are extreme points of S. See Fig. 9.

Broadcasts and reports within regions complete this step in $\Theta(\log n)$ time.

6. Each region creates a record with the total number of points that remain in the final upper (lower) hull. A $\Theta(\log n)$ time prefix operation will inform each region as to the new set of labels for its points. The relabeling of points and identification of preceding and succeeding points can be completed in $\Theta(\log n)$ time.

For further explanations and implementations of some of the $\Theta(\log n)$ time data movement operations, such as broadcast, report, parallel prefix, and so on, the reader may wish to refer to [40]. The worst case running time for this algorithm to identify the extreme points of n planar points distributed in an ordered fashion one point per processor on a hypercube of size n is given by the recurrence $T(n) = T(n^{3/4}) + \Theta(\log n)$, which is $\Theta(\log n)$.

Theorem 7: Given a set S of n planar points ordered by x-coordinate one point per processor in a hypercube of size n, the algorithm given above will identify the extreme points of hull(S) in $\Theta(\log n)$ time. ■

An expected $\Theta(\log n)$ time algorithm is given in [50] for sorting data on a hypercube of size n. This operation may be used in conjunction with the algorithm given in this section to derive an expected $\Theta(\log n)$ time hypercube algorithm to solve the convex hull problem for an arbitrarily distributed set of planar points. Sort the initial data into order by x-coordinate in expected $\Theta(\log n)$ time. This performs the initial preprocessing step, after which the algorithm given in this section can be performed as stated.

Theorem 8: Given a set S of n planar points distributed in an arbitrary fashion one point per processor in a hypercube of size n, the extreme points of hull(S) can be identified in expected $\Theta(\log n)$ time. ■

If a $\Theta(\log^2 n)$ time deterministic sort is used instead of an expected $\Theta(\log n)$ time sort to order the initial set of planar points by x-coordinate, then the following is obtained.

Corollary 9: Given a set S of n planar points distributed in an arbitrary fashion one point per processor in a hypercube of size n, the extreme points of hull(S) can be identified in $\Theta(\log^2 n)$ time. ■

V. Improving the Hierarchical Algorithms

Ideas used in the hypercube algorithm of Section IV can be combined with the algorithms given in Section III to produce slightly faster algorithms for the pyramid, tree, and mesh-of-trees. Instead of merging two regions at a time, as in Section III, one can merge $\Theta(\log^{1/3} n)$ regions at a time. However, if each region pipelines $\Theta(\log n)$ probes to each other region, then the apex would become a bottleneck. This can be circumvented by having each region send only $\Theta(\log^{1/3} n)$ probes at a time to each other region, resulting in $\Theta(\log n)$ probes passing through the apex at each step. With this

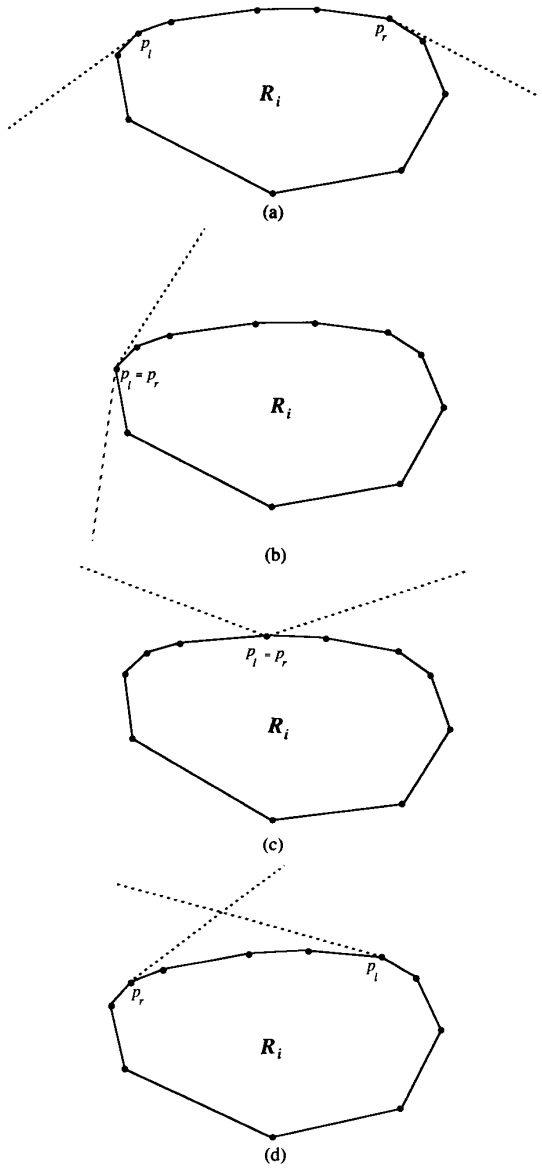


Fig. 9. Using p_l and p_r to determine extreme points. (a) p_l is to the left of p_r and the angle open to the top is $> 180°$. Those extreme points of R_i that are between p_l and p_r are extreme points of S. (b) p_l is equal to p_r and the angle open to the top is $> 180°$. p_l $(= p_r)$ is an extreme point of S. (c) p_l is equal to p_r and the angle open to the top is $\leq 180°$. No extreme points of R_i are extreme points of S. (d) p_r is to the left of p_l. No extreme points of R_i are extreme points of S.

modification, the time of the algorithms obey the recurrence $T(n) = T(n/\log^{1/3} n) + \Theta(\log^2 n/\log\log n)$, which is $\Theta(\log^3 n/(\log\log n)^2)$.

Theorem 10: Given a set S of n planar points ordered by x-coordinate in base of a pyramid, base of a mesh-of-trees, or leaves of a tree, the extreme points of S can be identified in $\Theta(\log^3 n/(\log\log n)^2)$ time. ■

VI. EREW PRAM Algorithm for Unordered Input

The hypercube algorithm for ordered data given in Section IV can be modified in a straightforward manner to solve the convex hull problem for planar point data stored in an arbitrary fashion on an EREW PRAM in optimal $\Theta(\log n)$ time using optimal $\Theta(n)$ processors. Previously, such time and processor bounds were obtainable for algorithms on the CREW PRAM [1], [6], but it is not clear that these algorithms could be modified to run in the same time using a linear number of processors on an EREW PRAM.

The modifications to the hypercube algorithm of Section IV consist of sorting the initial data into order by x-coordinate on the EREW PRAM, and then simply following the spirit of the algorithm given in Section IV on the EREW PRAM. It should be noted that deterministic sorting or routing of n elements distributed one per processor on an n processor EREW PRAM can be completed in $\Theta(\log n)$ time [15], as can parallel prefix [25].

Theorem 11: Given a set S of n planar points distributed arbitrarily one per processor on a EREW PRAM with n processors, in optimal $\Theta(\log n)$ time the extreme points of hull(S) can be identified. Furthermore, this result is both time and processor optimal. ■

VII. A Modified AKS Network Algorithm for Unordered Input

Recall from Section II-B-7 that a modified AKS network (a bounded degree network) with $O(n)$ processors can sort n items, distributed no more than one per processor, in worst case $O(\log n)$ time by restricting the interconnections between processors to those present in the LeiAKS network. Other operations required in the hypercube algorithm for arbitrary input, given in Section IV, can be accommodated on the modified AKS network by exploiting the interconnections available through the modified cube-connected cycles. Therefore, a straightforward adaptation of the expected $\Theta(\log n)$ time hypercube algorithm for arbitrary input given in Section IV will yield an optimal worst case $\Theta(\log n)$ time algorithm for solving the convex hull problem on a modified AKS network, under the assumption that the set of planar points is initially distributed in an arbitrary fashion, one element per processor of the network. This is the first $\Theta(\log n)$ time algorithm given to solve the convex hull problem for arbitrary planar point input on a fixed degree network. Recall from Section II-D that any parallel algorithm to solve the general convex hull problem for planar point data input must take $\Omega(\log n)$ worst case time, regardless of how many processors are available.

Theorem 12: Given a set S on n planar points distributed arbitrarily one per processor on a modified AKS network, in optimal $\Theta(\log n)$ time the extreme points of hull(S) can be identified. Furthermore, this result is both time and processor optimal. ■

VIII. Conclusion

In this paper, poly-logarithmic time parallel algorithms were given to determine the convex hull of a set of planar points using a hypercube, pyramid, tree, mesh-of-trees, mesh with reconfigurable bus, EREW PRAM, and a modified AKS network. All algorithms are new and more efficient than any previously developed algorithms for the same machine and input set. It was discussed that given a set S of n planar points,

distributed arbitrarily one per processor on a network with n processors, optimal worst case solutions to the convex hull problem on networks of processors such as the pyramid and mesh-of-trees require $\Omega(n^{1/2})$ time, and in fact $\Theta(n^{1/2})$ time solutions to this problem were cited. For the case where the n input points are ordered on a machine with $\Theta(n)$ processors, our hypercube algorithm finishes in worst case $\Theta(\log n)$ time, our pyramid, tree, and mesh-of-trees algorithms finish in worst case $\Theta(\log^3 n/(\log \log n)^2)$ time, and our mesh with reconfigurable bus algorithm finishes in worst case $\Theta(\log^2 n)$ time.

We showed that our $\Theta(\log n)$ time hypercube algorithm can be modified to give an optimal time and processor worst case $\Theta(\log n)$ time EREW PRAM algorithm for planar point data given in an arbitrary fashion. This improves upon the CREW PRAM algorithm of [1] and [6] in that the same optimal worst case running time is achieved, but on the weakest PRAM model. By using an expected $\Theta(\log n)$ time routing algorithm, the hypercube algorithm for ordered data can be modified to give an expected $\Theta(\log n)$ time algorithm for arbitrary planar point data input. Finally, we gave the first optimal worst case $\Theta(\log n)$ time algorithm for solving the convex hull problem for arbitrary planar input on a fixed degree network.

We should note that the worst case $\Theta(\log^2 n)$ time algorithm that we presented for the reconfigurable mesh can be modified using data movement operations developed in [33] to work even if the model is restricted so that only one message is allowed to be broadcast onto a maximally connected bus at any time. Furthermore, we conjecture that $\Theta(\log^2 n)$ time is suboptimal for this architecture, and leave the problem of lower bounds and/or faster algorithms as an open problem. Similarly, we conjecture that a running time of $\Theta(\log^3 n/(\log \log n)^2)$ is not the best possible for the pyramid, tree, and mesh-of-trees, and leave this as an open problem.

Acknowledgment

The authors would like to express their appreciation to the anonymous reviewers for their comments.

References

[1] A. Aggarwal, B. Chazelle, L. Guibas, C. O'Dunlaing, and C. Yap, "Parallel computational geometry," in *Proc. 1985 Symp. Foundations Comput. Sci.*, pp. 468-477.

[2] M. Ajtai, J. Komlos, and E. Szemeredi, "An $O(n \log n)$ sorting network," *Combinatorica*, vol. 3, pp. 1-19, 1983.

[3] S. Akl, "Parallel algorithms for convex hulls," Dep. Comput. Sci., Queens Univ., Kingston, Ont., Canada, 1983.

[4] H. M. Alnuweiri and V. K. Prasanna Kumar, "Efficient image computations on VLSI architectures with reduced hardware," in *Proc. IEEE 1987 Workshop Comput. Architecture, Pattern Anal. Mach. Intell.*, pp. 192-199.

[5] Ametek, Inc., *Ametek System 14 User's Guide*, Aug. 1986.

[6] M. J. Atallah and M. T. Goodrich, "Efficient parallel solutions to some geometric problems," *J. Parallel Distributed Comput.*, vol. 3, pp. 492-507, 1986.

[7] ——, "Parallel algorithms for some functions of two convex polygons," in *Proc. 24th Allerton Conf. Commun., Contr., Comput.*, 1986, pp. 758-767.

[8] D. Avis, "On the complexity of finding the convex hull of a set of points," Tech. Rep. FOCS 79.2, School of Comput. Sci., McGill Univ., 1979.

[9] K. E. Batcher, "Sorting networks and their applications," in *Proc. AFIPS Spring Joint Comput. Conf.*, vol. 32, 1968, pp. 307-314.

[10] P. J. Burt and G. S. van der Wal, "Iconic image analysis with the pyramid vision machine (PVM)," in *Proc. IEEE 1987 Workshop Pattern Anal. Mach. Intell.*, pp. 137-144.

[11] V. Cantoni, M. Ferretti, S. Levialdi, and R. Stefanelli, "Papia: Pyramidal architecture for parallel image analysis," in *Proc. 1985 Comput. Arithmetic Conf.*

[12] B. Chazelle, "Computational geometry on a systolic chip," *IEEE Trans. Comput.*, vol. C-33, pp. 774-785, 1984.

[13] A. Chow, "A parallel algorithm for determining convex hulls of sets of points in two dimensions," in *Proc. 19th Allerton Conf. Commun., Contr., Comput.*, 1981, pp. 214-233.

[14] P. Clermont and A. Merigot, "Real time synchronization in a multi-SIMD massively parallel machine," in *Proc. IEEE 1987 Workshop Pattern Anal. Machine Intell.*, pp. 131-136.

[15] R. Cole, "Parallel merge sort," in *Proc. 27th IEEE Symp. Foundations Comput. Sci.*, 1986, pp. 511-517.

[16] S. Cook and C. Dwork, "Bounds on time for parallel RAMs to compute simple functions," in *Proc. 14th ACM Symp. Theory Comput.*, pp. 231-233.

[17] R. Cypher, J. L. C. Sanz, and L. Snyder, "Hypercube and shuffle-exchange algorithms for image component labeling," in *Proc. IEEE 1987 Workshop Pattern Anal. Mach. Intell.*, pp. 5-9.

[18] G. Fritsch, W. Kleinoeder, C. U. Linster, and J. Volkert, "EMSY85—The Erlanger multiprocessor system for a broad spectrum of applications," in *Proc. 1983 Int. Conf. Parallel Processing*, pp. 325-330.

[19] M. T. Goodrich, "Finding the convex hull of a sorted point set in parallel," *Inform. Processing Lett.*, vol. 26, 1987/1988, pp. 173-179.

[20] J. L. Gustafson, S. Hawkinson, and K. Scott, "The architecture of a homogeneous vector supercomputer," in *Proc. 1986 Int. Conf. Parallel Processing*, pp. 649-652.

[21] J. Hayes, T. N. Mudge, Q. F. Stout, S. Colley, and J. Palmer, "A microprocessor-based hypercube supercomputer," *IEEE Micro*, vol. 6, pp. 6-17, 1986.

[22] D. Hillis, *The Connection Machine*. Cambridge, MA: MIT Press, 1985.

[23] Intel Corporation, *iPSC System Overview*, Jan. 1986.

[24] C. S. Jeong and D. T. Lee, "Parallel geometric algorithms on a mesh connected computer," Tech. Rep. 87-02-FC-01 (revised), Dep. EECS, Northwestern Univ.

[25] C. P. Kruskal, L. Rudolf, and M. Snir, "The power of parallel prefix," in *Proc. 1985 Int. Conf. Parallel Processing*, pp. 180-185.

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

[27] V. K. P. Kumar and C. S. Raghavendra, "Array processor with multiple broadcast," in *Proc. 1985 Symp. Comput. Architecture.*

[28] T. Leighton, "Tight bounds on the complexity of parallel sorting," *IEEE Trans. Comput.*, vol. C-34, pp. 344-354, 1985.

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

[30] M. Lu and P. Varman, "Solving geometric proximity problems on mesh-connected computers," in *Proc. 1985 Workshop Comput. Architecture Pattern Anal. Image Database Management*, pp. 248-255.

[31] R. Miller, V. K. Prasanna Kumar, D. Reisis, and Q. F. Stout, "Meshes with reconfigurable buses," in *Proc. Fifth MIT Conf. Adv. Res. VLSI*, 1988, pp. 163-178.

[32] ——, "Image computations on reconfigurable VLSI arrays," in *Proc. IEEE Comput. Soc. Conf. Comput. Vision Pattern Recognition*, 1988, pp. 925-930.

[33] ——, "Data movement operations and applications on reconfigurable VLSI arrays," in *Proc. 1988 Int. Conf. Parallel Processing Vol. I: Architecture*, 1988, pp. 205-208.

[34] R. Miller and Q. F. Stout, "Computational geometry on a mesh-connected computer," in *Proc. 1984 IEEE Int. Conf. Parallel Processing*, pp. 66-73.

[35] ——, "Geometric algorithms for digitized pictures on a mesh-connected computer," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-7, pp. 216-228, 1985.

[36] ——, "Varying diameter and problem size in mesh-connected computers," in *Proc. 1985 Int. Conf. Parallel Processing*, pp. 697-699.

[37] ——, "Graph and image processing algorithms for the hypercube," in *Proc. 1986 SIAM Conf. Hypercube Multiprocessors*, 1987, pp. 418-425.

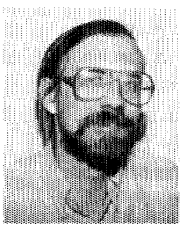
[38] ——, "Mesh computer algorithms for computational geometry," Tech. Rep. 86-18, State Univ. New York, Buffalo, Dep. Comput. Sci., July 1986; *IEEE Trans. Comput.*, to be published.

[39] ——, "Data movement techniques for the pyramid computer," *SIAM J. Comput.*, vol. 16, pp. 38–60, Feb. 1987.
[40] ——, *Parallel Algorithms for Regular Architectures*. Cambridge, MA: MIT Press, 1988, to be published.
[41] ——, "Augmenting Leighton's modification to the AKS network," Univ. Michigan Tech. Rep., 1988.
[42] D. Nassimi and S. Sahni, "Finding connected components and connected ones on a mesh-connected parallel computer," *SIAM J. Comput.*, vol. 9, pp. 744–757, 1980.
[43] ——, "Parallel permutation and sorting algorithms and a new generalized connection network," *J. ACM*, vol. 29, pp. 642–667, July 1982.
[44] D. Nath, S. N. Maheshwari, and P. C. P. Bhatt, "Parallel algorithms for the convex hull in two dimensions," in *Proc. Conf. Anal. Problem Classes Programming Parallel Comput.*, 1981, pp. 358–372.
[45] F. P. Preparata and S. J. Hong, "Convex hulls of finite sets of points in two and three dimensions," *Commun. ACM*, vol. 2, pp. 87–93, 1977.
[46] F. P. Preparata and D. T. Lee, "Computational geometry—A survey," *IEEE Trans. Comput.*, vol. C-33, pp. 1072–1100, 1984.
[47] F. P. Preparata and M. I. Shamos, *Computational Geometry*. Berlin, Germany: Springer-Verlag, 1985.
[48] F. P. Preparata and J. Vuillemin, "The cube-connected cycles: A versatile network for parallel computation," *Commun. ACM*, vol. 5, pp. 300–309, 1981.
[49] H. Raynaud, "Sur l'enveloppe convexe des nuages de points aleatoires dans R_n," *J. Appl. Probability*, vol. 7, pp. 35–48, 1970.
[50] J. H. Reif and L. G. Valiant, "A logarithmic time sort for linear size networks," *J. ACM*, vol. 34, pp. 60–76, 1987.
[51] D. H. Schaefer *et al.*, "The PMMP-A pyramid of MPP processing elements," in *Proc. 18th Annu. Hawaiian Int. Conf. Syst. Sci.*, vol. 1, 1985, pp. 178–184.
[52] M. I. Shamos, "Computational geometry," Ph.D. dissertation, Yale Univ., 1978.
[53] L. Snyder, "Introduction to the configurable, highly parallel computer," *Computer*, pp. 47–56, Jan. 1982.
[54] Q. F. Stout, "Meshes with multiple buses," in *Proc. 1986 IEEE Symp. Foundat. Comput. Sci.*, pp. 264–273.
[55] S. L. Tanimoto, "Programming techniques for hierarchical parallel image processors," in *Multicomputers and Image Processing Algorithms and Programs*, K. Preston and L. Uhr, Eds. New York: Academic, 1982, pp. 421–429.
[56] G. T. Toussaint, "Pattern recognition and geometrical complexity," in *Proc. 5th Int. Conf. Pattern Recognition*, 1980, pp. 1324–1347.
[57] L. Uhr, *Algorithm-Structured Computer Arrays and Networks*. New York: Academic, 1984.
[58] J. D. Ullman, *Computational Aspects of VLSI*. Rockville, MD: Computer Science Press, 1984.
[59] 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, Amherst.
[60] A. Yao, "A lower bound to finding convex hulls," Dep. Computer Sci., Stanford Univ., 1979.

Russ Miller (S'82–M'85) was born in Flushing, NY, on January 8, 1958. He received the B.S., M.A., and Ph.D. degrees in computer science/mathematics from the Department of Mathematical Sciences, State University of New York, Binghamton.

Since 1985 he has been an Assistant Professor in the Department of Computer Science at the State University of New York, Buffalo. Since 1988 he has also been Associate Director for the graduate group in Advanced Scientific Computing at the University of Buffalo. His primary research interests are parallel algorithms, parallel computing, and parallel architectures. He recently coauthored (with Q. F. Stout) the book *Parallel Algorithms for Regular Architectures* (Cambridge, MA: MIT Press, 1988).

Dr. Miller is a member of the IEEE Computer Society, the Association for Computing Machinery, the Society of Photo-Optical Instrumentation Engineers, and Phi Beta Kappa.



Quentin F. Stout (M'82) received the B.A. degree from Centre College, Danville, KY, and the Ph.D. degree from Indiana University.

Since 1984 he has been an Associate Professor in the Department of Electrical Engineering and Computer Science of the University of Michigan, Ann Arbor. From 1976 to 1984 he was in the faculty of the Mathematical Sciences Department of the State University of New York, Binghamton. His primary research interests are in parallel algorithms, parallel computing, and parallel architectures. He recently coauthored (with R. Miller) the book *Parallel Algorithms for Regular Architectures* (Cambridge, MA: MIT Press, 1988).

Dr. Stout is a member of the Association for Computing Machinery, the American Mathematical Society, and the Mathematical Association of America, and serves on the editorial board of the *Journal of Parallel and Distributed Computing*.