A Fast Adaptive Convex Hull Algorithm on Two-Dimensional Processor Arrays with a Reconfigurable BUS System[1]

S. Olariu
J. Schwing, and J. Zhang
Department of Computer Science
Old Dominion University
Norfolk, VA 23529-0162
U.S.A.

Abstract- **A bus system that can change dynamically to suit computational needs is referred to as reconfigurable. We present a fast adaptive convex hull algorithm on a 2-dimensional processor array with a reconfigurable bus system (2-d PARBS, for short). Specifically, we show that computing the convex hull of a planer set of n points taken $O(\log n/\log m)$ time on a 2-d PARBS of size $mn \times n$ with $3 \leq m \leq n$. Our result implies that the convex hull of n points in the plane can e computed in $O(1)$ time in a 2-d PARBS of size $n^{1.5} \times n$.**

1 Introduction

Recent advances in VLSI have made it possible to build massively parallel machines featuring many thousands of cooperating processors. This increase in computational power does not, however, translate into increased performance of the same order of magnitude. One of the reasons seems to be that interprocessor communications and simultaneous memory accesses often act as bottlenecks in parallel machines.

To alleviate the inefficiency of long distance communication among processors, bus systems have been recently added to a number of parallel machines [2-4,5,6,11]. If such a bus system can be dynamically changed, under program control, to suit communication needs among processors, it is referred to as *reconfigurable*. Examples include the *bus automaton* [11], the *reconfigurable mesh*, and the *polymorphic torus* [2,3], among others.

The computational model used throughout this work is the *reconfigurable mesh* [5]. An $m \times n$ reconfigurable mesh (also called a PARBS [13]) consists of $m \times n$ identical processors positioned on a rectangular array (refer to Figure 1). The processor at (i,j), $(1 \leq i \leq m; 1 \leq j \leq n)$ is identified by $P(i,j)$. Every processor has 4 ports denoted by N, S, E, and W. There are also implicit *north*, *south*, *east*, and *west* directions (refer to Figure 1). In each processor, ports can be dynamically connected in pairs to suit computational needs. In the absence of these local connections, the PARBS is functionally equivalent to the mesh connected computer.

[1]This work was supported by NASA under grant NCC1-99 by the National Science Foundation under grant CCR-8909996 is gratefully acknowledged

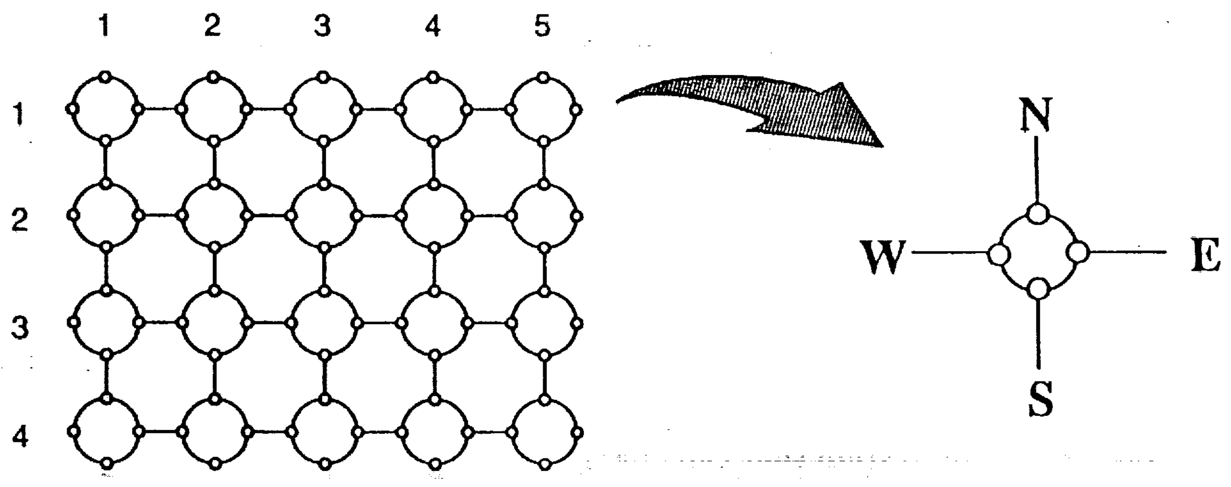

Figure 1: A 4x5 PARBS

We assume that each processor has a small number of registers of size $O(\log n)$ bits and that a processor can perform in unit time standard arithmetic and boolean operations. We assume a single instruction stream: in each time unit the same instruction is broadcast to all processors, which execute it and wait for the next instruction. Each instruction can consist of setting local connections (as explained later), performing an arithmetic or boolean operation, broadcasting a value on a bus, or receiving a value from a specified bus. The regular structure of the PARBS makes it suitable for VLSI implementation. In fact, it has been argued [5] that the PARBS can be used as a universal chip capable of simulating any equivalent-area architecture without loss of time.

By adjusting the local connections within each processor several subbuses can be established. We assume that the setting of local connection is destructive in the sense that setting a new pattern of connections destroys the previous one. At any given time, only one processor can broadcast a value onto a bus. Processors, if instructed to do so, read the bus. If no value is being transmitted on the bus, the read operation has no result. It is assumed [5,6] that communications along buses take $O(1)$ time. This seems to be a reasonable assumption in the light of recent experiments with the YUPPIE system [4].

A number of problems have been solved in $O(1)$ time on PARBS. Very recently, Wang et al [13] have proposed $O(1)$ algorithms for the transitive closure and some related graph problems; Olariu, Schwing, and Zhang [9] have proposed an adaptive sorting algorithm; specifically, they show that sorting a sequence of n reals takes $O(\frac{\log n}{\log m})$ time on a 2-d PARBS of size $nm \times n$ with $3 \leq m \leq n$. In particular, their result implies a constant-time sorting algorithm on an $n^{1.5} \times n$ 2-d PARBS.

The convex hull of a set of points in the plane is defined as the smallest area convex set that contains the original set. The problem of computing the convex hull of points in the plane is central in a variety of problems in pattern recognition, computer graphics, statistics, and image processing [1,7,8,10].

To the best of our knowledge, no convex hull algorithm has been reported in the literature on a 2-d PARBS. The purpose of this paper is to propose a fast adaptive convex hull algorithm for a set of n points in the plane. We reduce the problem of computing the convex hull of a set of planar points to the problems of sorting and computing the prefix maximum of n real numbers. To begin, we show that the problem of computing the maximum of n real numbers can be solved in time $O(\frac{\log n}{\log m})$ on a 2-d PARBS of size $m \times n$ with $2 \leq m \leq n$. We also use the fast adaptive sorting algorithm of [9]. What results is a fast adaptive algorithm that computes the convex hull of a set of n points in the plane in $O(\frac{\log n}{\log m})$ time on a 2-d PARBS of size $nm \times n$ with $3 \leq m \leq n$. In particular, for $m=n^{0.5}$ we obtain an $O(1)$ time convex hull algorithm on a 2-d PARBS of size $n^{1.5} \times n$.

2 The stepping stones

Our convex hull algorithm relies on a number of intermediate results that we present next. To begin, we consider the problem of computing the prefix maximum of n reals on a $n \times n$ PARBS. Specifically, given n real numbers $a_1, a_2, \ldots, a_n$ with processor $P(1,j)$ storing a_j, the problem is to compute $\max_{1 \leq i \leq j}\{a_i\}$ for all $1 \leq j \leq n$. Our algorithm involves establishing a number of subbuses and broadcasting values along them. The details of our algorithm are spelled out by the following sequence of steps.

```
Algorithm Prefix-Maximum;

Step 1. every processor $P(i,j)$ $(2 \leq i \leq n-1; 1 \leq j \leq n)$ connects its ports $N$ and $S$;

Step 2. every processor $P(1,j)$ $(1 \leq j \leq n)$ broadcasts $a_j$ southbound along the vertical
    subbus in column $j$;

Step 3. every processor $P(i,j)$ $(2 \leq i \leq n-1; 2 \leq j < i)$ connects its ports $W$ and $E$;

Step 4. every processor $P(j,j)$ $(2 \leq j \leq n)$ broadcasts $a_j$ westbound along the horizontal
    subbus in row $j$;

Step 5. every processor $P(j,i)$ $(2 \leq j \leq n-1; 1 \leq i < j)$ compares $a_i$ and $a_j$;
    if $a_i > a_j$ then
        $P(j,i)$ disconnects the horizontal subbus;
        marks itself;

Step 6. every marked processor $P(i,j)$ broadcasts a "0" along the horizontal subbus
    eastbound;

Step 7. every processor $P(j,j)$ $(1 \leq j \leq n)$ stores in its own memory a "0" or a "1"
    depending on whether or not it has received a "0" in Step 6;

Step 8. every processor $P(i,j)$ $(2 \leq i < j \leq n)$ connects its ports $N$ and $S$;
```

Step 9. every processor $P(j,j)$ $(2 \le j \le n)$ broadcasts on the vertical subbus northbound the value it has stored in Step 7;

Step 10. every processor $P(1,j)$ $(2 \le j \le n)$ that has received a "0" in Step 9 connects its ports W and E;

Step 11. every processor $P(1,j)$ $(1 \le j \le n)$ that stores a "1" broadcasts a_j eastbound along the horizontal subbus in row 1;

Theorem 1. Algorithm Prefix-Maximum correctly computes the prefix maximum of n real numbers in O(1) time on an $n \times n$ PARBS.
Proof. To begin, note that in Step 5, every processor $P(i,j)$ $(2 \le i \le n-1; 1 \le j < i)$ knows a_i and a_j. Further, it is easy to see that at the end of Step 7 processor $P(j,j)$ $(1 \le j \le n)$ stores a "1" if, and only if, a_j is as least as large as a_i with $i < j$.

Consequently, every processor $P(1,j)$ in row 1 that at the end of Step 9 stores a "0" knows that a_j cannot be the prefix maximum of a_i for $i \le j$. In fact the prefix maximum of the first j real numbers $a_1, a_2, \ldots, a_j$ is stored by the first processor to the left of $P(1,j)$ that stores a "1". The conclusion follows. □

Next, we show how to compute the maximum of n real numbers $a_1, a_2, \ldots, a_n$ on an $m \times n$ PARBS with $2 \le m \le n$. Again, we assume that the numbers are stored one per processor such that for all j $(1 \le j \le n)$, $P(1,j)$ stores a_j. The idea of our algorithm is to partition the original $m \times n$ PARBS into subPARBS of size $m \times m$. To avoid tedious but inconsequential housekeeping details we assume that n is a power of m.

We partition the n columns into contiguous groups of m columns each and let the k-th subPARBS, M_k, $(0 \le k \le n/m - 1)$ consist of the columns $km+1$, $km+2, \ldots, km+m$. As a preprocessing step, for all j $(2 \le j \le n)$ we move the data contained in $P(1,j)$ to the "diagonal" processor of its $m \times m$ subPARBS, $P((j-1) \bmod m + 1, j)$. The main loop of this algorithm applies the (prefix) maximum algorithm described above to specified $m \times m$ subPARBS. This process proceeds iteratively, determining the maxima of groups of size m, m^2, m^3, and so on. Clearly, in $\log_m n = \frac{\log n}{\log m}$ iteration we have computed the maximum of the n numbers.

We omit the details of bus-construction steps which are similar to those in the previous algorithm. The reader can easily fill in the details.

Algorithm Maximum;

```
Step 1. {preprocessing}
    for all $j$ $(1 \le j \le n)$ in parallel
        establish a vertical subbus from $P(1,j)$ to $P((j-1) \bmod m + 1, j)$;
        $P(1,j)$ broadcasts $a_j$ on this subbus to $P((j-1) \bmod m + 1, j)$;
        $P((j-1) \bmod m + 1, j)$ marks itself
    endfor;
```

```
Step 2. {main loop}
    for $k \leftarrow 1$ to $\frac{\log n}{\log m}$ do
        for all $j$ $(1 \leq j \leq \frac{n}{m^k})$ in parallel
            all processors connect ports $W$ and $E$;
            all processors $P(i,(j-1)m^k+1)$ split the horizontal subbus in row $i$;
            all marked processors broadcast the value they hold
            along the horizontal subbus westbound;
            all marked processors unmark themselves;
            $M_{(j-1)m^{k-1}}$ computes the maximum of the values
            in column $(j-1)m^k+1$;
            let the result be stored in $P((j-1) \bmod m+1,(j-1)m^k+1)$;
            all processors $P((j-1) \bmod m+1,(j-1)m^k+1)$ mark themselves
        endfor
    endfor;
```

Theorem 2. Algorithm Maximum correctly computes the maximum of n real numbers in $O(\frac{\log n}{\log m})$ time on an $m \times n$ PARBS with $2 \leq m \leq n$.

Proof. The correctness is implied by the following result: at the end of the t-th iteration $(0 \leq t \leq \frac{\log n}{\log m})$, for all j $(1 \leq j \leq \frac{n}{m^k})$, processor $P((j-1) \bmod m+1,(j-1)m^t+1)$ contains the maximum in columns $(j-1)m^t+1$ through jm^t.

The proof of the above statement is by induction on t. The basis is easy: at the end of the 0-th iteration the conclusion is guaranteed by the preprocessing step.

Assume the above statement satisfied at the end of the t-th iteration. We only need show that it also holds at the end of the $(t+1)$-st iteration. For this purpose, it is instructive to follows the $(t+1)$-st iteration: here, after all processors connect their ports W and E thus establishing horizontal subbuses in each row, the processors $P(i,(j-1)m^{t+1}+1)$ split the horizontal subbus in row i; next, all marked processors broadcast the value they hold along the horizontal subbus westbound. By the induction hypothesis, these are processors $P((j-1) \bmod m+1,(j-1)m^t+1)$. Therefore, when the subPARBS $M_{(j-1)m^{t+1}}$ compute the maximum of the values in column $(j-1)m^{k+1}+1$, the induction hypothesis guarantees that the resulting value is the maximum in columns $(j-1)m^{t+1}+1$ through jm^{t+1}, a total of m^{t+1} columns.

To argue for the running time, note that by Theorem 1, the inner for loop runs in O(1) time. The conclusion follows. □

3 The Algorithm

We are now in a position to present our planar convex hull algorithm. Let $S=\{p_1, p_2,\ldots,p_n\}$ be a planar set of points; for $1 \leq i \leq n$, p_i is represented by its Cartesian coordinates (x_i, y_i). To avoid tedious details we assume, without loss of generality, that the points in S are in *general* position, with no three collinear and no two having the same x or y coordinate. The output of the convex hull algorithm is a linked list CH that contains all the points on the convex hull starting with the one with the largest x coordinate and proceeding counterclockwise. Our algorithm consists of the following sequence of steps.

```
Algorithm Convex-Hull;

Step 1. find the four extremal points in $S$, and let them be, without loss of generality, $p_1$,
        $p_2$, $p_3$, and $p_4$. Specifically, $x_1=\max_{1\le j\le n}\{x_j\}$, $y_2=\max_{1\le j\le n}\{y_j\}$, $x_3=\min_{1\le j\le n}\{x_j\}$,
        and $y_4=\min_{1\le j\le n}\{y_j\}$.

Step 2. compute the sets
            $S_1 = \{p_i | x_2 \le x_i \le x_1; y_1 \le y_i \le y_2\}$,
            $S_2 = \{p_i | x_3 \le x_i \le x_2; y_3 \le y_i \le y_2\}$,
            $S_3 = \{p_i | x_3 \le x_i \le x_4; y_4 \le y_i \le y_3\}$,
            $S_4 = \{p_i | x_4 \le x_i \le x_1; y_4 \le y_i \le y_1\}$.
        Note: For simplicity, we deal with $S_1$ only, the others being perfectly similar.

Step 3. sort the points in $S_1$ by increasing $y$ coordinate, and let $L_1=(p_1=q_1, q_2,\ldots,q_t=p_2)$
        be the resulting sorted sequence;

Step 4. for all $j$ $(1 \le j \le t)$ in parallel
            find the subscript $d_j$ $(j < d_j \le t)$ such that the angle determined by $q_{d_j}$,
            $q_j$, and the negative direction of the $x$ axis is as large as possible;

Step 5. compute the prefix maximum of the values $d_j$ in $L_1$, and set $m(j) \leftarrow \max_{1\le t\le j-1}\{d_t\}$;

Step 6. $CH_1 \leftarrow L_1$;
        for all $j$ $(2 \le j \le t-1)$ in parallel
            remove $q_j$ from $CH_1$ whenever $d_j \le m(j)$;
```

Before giving the proof of correctness of our algorithm, we need to take note of the following simple observation. The sorted sequence L_1 of points obtained at the end of Step 3 can be viewed as determining a *polygonal line* (termed a *chain* in [10]) joining p_1 and p_2. It is easy to see that the convex hull CH of the set S of points is exactly the convex hull of the simple polygon P obtained by concatenating the polygonal lines L_1, L_2, L_3, and L_4, in this order.

The following result argues for the correctness of our algorithm.

Theorem 3. At the end of Step 6, CH_1 contains the portion of the convex hull contained in S_1.

Proof. By the previous observation we only need show that the linked list CH_1 obtained at the end of step 6 contains the restriction of the convex hull of P between p_1 and p_2. This follows from the following claim

a point q_j $(2 \le j \le t-1)$ of L_1 belongs to CH if, and only if, $d_j > m(j)$.

First, let $q_j (2 \leq j \leq t-1)$ in L_1 belongs to the convex hull and let q_i and q_k $(i < j < k)$ be its immediate neighbors on the convex hull. (We note that since q_1 and q_t trivially belong to the convex hull, the points q_i and q_k are well defined.) Clearly, $d_i = j$ and so $m(j) = j < d_j = k$, as claimed.

Conversely, if some point q_j in L_1 does not belong to the convex hull then let q_i and q_k $(i < k)$ be the closest points on the convex hull, with q_j lying on the chain from q_i to q_k. Since q_i and q_k are neighbors on the convex hull, we have $d_i = k$; furthermore, $d_j \leq k = m(j)$, and the conclusion follows. □

Next, we propose to show how Steps 1–6 above can be efficiently implemented on a 2-d PARBS. More precisely, we assume a 2-d PARBS of size $nm \times n$ with $3 \leq m \leq n$. Some of the Steps 1–6 in our algorithms need the whole PARBS while others can run on a subPARBS, as specified; the data movement necessary to conform to the input requirements of a specific step are ignored here; the reader can easily work out all the details.

Step 1 can be implemented to run in O(1) time on an $n \times n$ subPARBS since we only need compute $\max_{1 \leq j \leq n}\{z_j\}$ and $\min_{1 \leq j \leq n}\{z_j\}$ with $z = x$ and $z = y$.

Step 2 is demonstrated for S_1 only; computing S_i with $i = 2, 3, 4$ is similar. All that is needed is to establish a subbus running through the whole of row 1. The processors storing p_1 and p_2 broadcast, in two computational steps, their Cartesian coordinates to all processors in row 1; every processor that stores a point in S_1 marks itself. Thus Step 2 runs in O(1) time.

Step 3 can be implemented as follows. First, all unmarked processors change the y coordinate of the point that they store to $+\infty$. Now the sorting algorithm in [9] is invoked: this runs in $O(\frac{\log n}{\log m})$ and uses the whole PARBS. Note that at the end of Step 3, processors $P(1,1)$, $P(1,2)$,...,$P(1,t)$ contain L in sorted order.

Step 4 can be implemented to run in O(1) time on an $mn \times n$ subPARBS as follows. Recall from Step 3 that, initially, for all $1 \leq j \leq t$ $P(1,j)$ stores q_j. For further reference, this subPARBS is further subdivided into subPARBS of size $m \times n$ as follows. The first $m \times n$ subPARBS involves the first m rows, the second the next m rows and so on. We establish vertical subbuses in each column and let $P(1,j)$ broadcast the Cartesian coordinates of q_j along the subbus in column j $(1 \leq j \leq t)$. Next, establish horizontal subbuses running from $P(m(j-1)+1, j)$ to $P(m(j-1)+1, t)$ $(1 \leq j \leq t)$. Note that these are precisely the first rows of our $m \times n$ subPARBS. For all j, $P(m(j-1)+1, j)$ broadcasts the Cartesian coordinates of q_j eastbound on the horizontal subbus in row $m(j-1)+1$. Every processor $P(m(j-1)+1, k)$ with $j < k \leq t$ computes the angle specified in Step 4. Actually, computing the angle itself is not necessary, the tangent of the angle can be readily computing using two subtractions and a division. Now the maximum of all values in the first rows of these subPARBS can be computed in $O(\frac{\log n}{\log m})$ time using Algorithm Maximum developed in Section 2. It is easy to arrange for the maximum in row $m(j-1)+1$ to be sent back to $P(1,j)$. This, clearly takes O(1) time since only the appropriate subbuses have to be established and the information broadcast along them.

Step 5 can be implemented to run on an $n \times n$ subPARBS by using Algorithm Prefix-Maximum discussed in Section 2.

Step 6 involves marking every $P(1,j)$ that contains a point of the convex hull. After this is done, a horizontal subbus is established in row 1. Every marked processor splits this bus and broadcasts its identity westbound on its own subbus. This, in fact creates the list CH_1 as desired. Clearly, the running time of this step is O(1).

To summarize our discussion we state the following result.

Theorem 4. The convex hull of a planar set of n points can be computed on an PARBS of size $nm \times n$ with $3 \leq m \leq n$ in $O(\frac{\log n}{\log m})$ time. □

In particular, if $m = n^{0.5}$ then we have the following result.

Corollary 4.1. The convex hull of a planar set of n points can be computed in O(1) time on an PARBS of size $n^{1.5} \times n$. □

4 Conclusion

A bus system that can be dynamically altered to suit communicational needs among co-operating processors is referred to as *reconfigurable.* In this paper we a fast adaptive algorithm to solve the planar convex hull problem.

Specifically, we showed that computing the convex hull of a set of n points in the plane takes $O(\frac{\log n}{\log m})$ on a 2-d PARBS of size $nm \times n$ with $3 \leq m \leq n$. In particular, our result implies that the same problem can be solved in O(1) time on a 2-d PARBS of size $n^{1.5} \times n$.

References

[1] J. A. Holey and O. H. Ibarra, Iterative algorithms for planar convex hull on mesh connected arrays, *Proc. 1990 International Conference on Parallel Processing*, III-102-III-109.

[2] H. Li and M. Maresca, Polymorphic-torus network, *IEEE Transactions on Computers*, vol. C-38, no. 9, (1989) 1345-1351.

[3] H. Li and M. Maresca, Polymorphic-torus architecture for computer vision, *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 11, no. 3, (1989) 233-243.

[4] M. Maresca and H. Li, Connection autonomy and SIMD computers: a VLSI implementation, *Journal of Parallel and Distributed Computing*, vol. 7 (1989) 302-320.

[5] R. Miller, V. K. Prasanna-Kumar, D. Reisis, and Q. F. Stout, Meshes with reconfigurable buses, *Proceedings of the International Conference on Parallel Processing*, vol. 1, (1988) 205-208.

[6] R. Miller, V. K. Prasanna-Kumar, D. Reisis, and Q. F. Stout, Data movement operations and applications on Reconfigurable VLSI arrays, *Proceedings of the fifth MIT Conference on Advanced Research in VLSI*, (1988) 163-178.

[7] R. Miller and Q. Stout, Mesh Computer Algorithms for Computational Geometry, *IEEE Trans. on Computers*, 38 (1989), 321-340.

[8] R. Miller and Q. Stout, Efficient Parallel Convex Hull Algorithms, *IEEE Trans. on Computers*, 37 (1988), 1605-1618.

[9] S. Olariu, J. L. Schwing, and J. Zhang, A Fast Adaptive Sorting Algorithm on a Two Dimensional Processor Array with a Reconfigurable Bus System, submitted.

[10] F. P. Preparata and M. I. Shamos, Computational Geometry, An Introduction, Springer-Verlag, New York, Berlin, 1988.

[11] J. Rothstein, Bus automata, brains, and mental models, *IEEE Trans. on Systems Man Cybernetics*, 18, (1988).

[12] Q. F. Stout, Meshes with Multiple Buses, *Proc. 27th IEEE Symp. on the Foundations of Comp. Science*, (1986) 264-273.

[13] B. F. Wang, C. J. Lu, and G. H. Chen, Constant Time Algorithms for the Transitive Closure Problem and its Applications, *Proceedings of the International Conference on Parallel Processing*, vol. 3, (1990) 52-59.