A New Adaptive Sorting Algorithm on the Reconfigurable Mesh, an Image Understanding Architecture

M. Manzur Murshed
Computer Sciences Laboratory
The Australian National University
Canberra ACT 0200, Australia

Richard P. Brent
Oxford University Computing Laboratory
Oxford University
Oxford, OX1 3QD, U.K.

Abstract

Recently we have introduced the idea of adaptive *algorithms which runs on reconfigurable meshes of variable sizes and aspect ratios without compromising AT^2 optimality. We have also supported our idea by developing adaptive algorithms for sorting and computing maxima. In this paper we develop a new adaptive sorting algorithm which has lower constant associated with the highest order term in the complexity order than our previously published adaptive sorting algorithm.*

1 Introduction

It is well known that interprocessor communications and simultaneous memory accesses often act as bottlenecks in present-day parallel machines. To address this problem, dynamically reconfigurable bus systems have been introduced recently to a number of parallel architectures [7] of which the *reconfigurable mesh* [12] has drawn considerable interest. Among many applications, the reconfigurable mesh can be viewed as a part of a powerful image understanding architecture for supporting real time image understanding applications and research in knowledge-based computer vision [13].

The virtual communication diameter of a reconfigurable mesh of any size is $O(1)$ which is exploited by many researchers to develop constant time algorithms [7]. To realise these constant time algorithms we need to use more processors than we usually use to solve the same problems on ordinary meshes. In fact, we can easily observe that the ratio of the number of processors used in a constant time algorithm to the number of processors used in an ordinary mesh algorithm solving the same problem is polynomial in problem size. Ben-Asher *et al.* [1] have presented the idea of self-simulation where the existing reconfigurable mesh algorithms are executed with slowdown on a reconfigurable mesh of size smaller than intended for those algorithms. A few self-simulation techniques appear in [1, 6] with optimal slowdown for various models of reconfigurable mesh.

In [5] we have pointed out that self-simulation, even with optimal slowdown, compromises the AT^2 [11] optimality of the resultant algorithm. To overcome this limitation of self-simulation, we have presented a new idea of developing algorithms on reconfigurable meshes which will be adaptive in the sense that these algorithms can be executed on reconfigurable meshes of variable sizes and aspect ratios while keeping the AT^2 measures unaffected. To illustrate our idea we have developed adaptive AT^2 optimal algorithms for sorting items and computing the contour of maximal elements of a set of planar points in [5].

In this paper, we develop a new adaptive sorting algorithm based on Schnorr and Shamir's efficient sorting algorithm (see Section 2.2) on ordinary meshes. The constant factor of the highest order term in the complexity order of Schnorr and Shamir's algorithm is much lower than that of Marberg and gafni's *rotatesort* [4] algorithm based on which our previous adaptive sorting algorithm was developed in [5]. We, therefore, claim that the adaptive sorting algorithm developed here is more efficient than the adaptive sorting algorithm presented in [5].

The paper is organised as follows. The computational model of the reconfigurable mesh and the sorting algorithm of Schnorr and Shamir are presented in Section 2. In Section 3 we formally define the idea of adaptive algorithms and then present a generic adaptive algorithm. In the framework of this generic adaptive algorithm, we further develop a new efficient adaptive sorting algorithm in Section 4.

2 Preliminaries

For the sake of completeness, we briefly describe the reconfigurable mesh in Section 2.1 and present Schnorr and Shamir's sorting algorithm on ordinary meshes in Section 2.2.

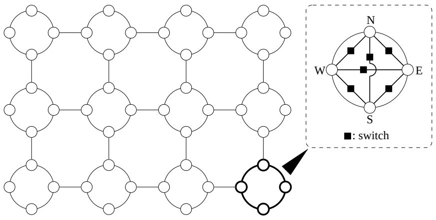

Figure 1: A reconfigurable mesh of size 3×4.

2.1 The Reconfigurable Mesh

The reconfigurable mesh [12] is primarily a two-dimensional mesh of processors connected by reconfigurable buses. In this parallel architecture, a processor is placed at the grid points as in the usual mesh connected computers. Processors of the reconfigurable mesh of size $X \times Y$ are denoted by $PE_{i,j}$, $0 \leq i < X$, $0 \leq j < Y$ where processor $PE_{0,0}$ resides in the south-western corner. Each processor is connected to at most four neighbouring processors through fixed bus segments connected to four I/O ports **E** & **W** along dimension x and **N** & **S** along dimension y. These fixed bus segments are building blocks of larger bus components which are formed through switching, decided entirely on local data, of the internal connectors (see Figure 1) between the I/O ports of each processor. The fifteen possible interconnections of I/O ports through switching are shown in Figure 2. Like all reconfigurable bus systems, the behaviour of reconfigurable mesh relies on the key assumption that the transmission time of a message along a bus is independent of the length of the bus.

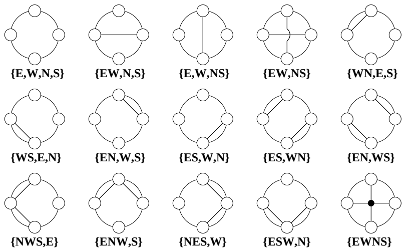

Figure 2: Possible interconnections between the four I/O ports of a processor.

A reconfigurable mesh operates in the single instruction multiple data (SIMD) mode. Besides the reconfigurable switches, each processor has a computing unit with a fixed number of local registers. Other than the buses and switches the reconfigurable mesh of size $p \times q$ is similar to the standard mesh of size $p \times q$ and hence it has $\Theta(pq)$ area in VLSI embedding [11], under the assumption that processors, switches, and links between adjacent switches occupy unit area.

2.2 Schnorr and Shamir's Sorting Algorithm

Consider sorting MN items on an ordinary mesh of size $M \times N$ where each processor has exactly one item (in scrambled order) at the start. Also suppose that, after sorting, each processor will again contain exactly one item. Schnorr and Shamir [10] have assumed the order of processors in the final output to be the snake-like-row-major order. To keep the algorithm correct for the entire range of values of M and N, it is also assumed that $N \leq M^2$. We further assume that $M = 2^{4s}$, $N = 2^{4t}$, and $4s \geq 3t$, which implies $M \geq N^{3/4}$, for the sake of simplicity in presentation.

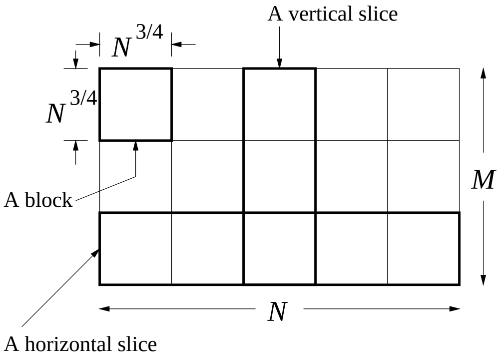

Figure 3: Definition of blocks, vertical slices, and horizontal slices in the sorting algorithm of Schnorr and Shamir.

Algorithm 1 Schnorr and Shamir's Algorithm [10]

1 *Sort all the blocks (Figure 3) in snake-like-row-major order;*

2 *Permute the columns so that the $N^{3/4}$ columns in each block are distributed evenly among the $N^{1/4}$ vertical slices (Figure 3);*

3 *Sort all the blocks in snake-like-row-major order;*

4 *Sort all the columns of the mesh downwards;*

5 Collectively sort blocks 1 and 2, blocks 3 and 4, etc., of each vertical slice in snake-like-row-major order;

6 Collectively sort blocks 2 and 3, blocks 4 and 5, etc., of each vertical slice in snake-like-row-major order;

7 Sort all the rows of the mesh into alternating left-to-right and right-to-left order;

8 Perform $2N^{3/4}$ steps of the odd-even transposition sort [3] along the snake;

---

3 Adaptive Algorithms

Let a problem $\mathcal{P}$ of size n have $I(n)$ *information content* [11, pp. 51–54]. If this problem $\mathcal{P}$ is realized in a VLSI circuit with aspect ratio $\alpha \geq 1$ then, by Ullman [11, pp. 57], the AT^2 lower bound of $\mathcal{P}$ will be $\Omega(\alpha I^2(n))$. Now, consider a reconfigurable mesh of size $p \times q$ where $pq = kI(n)$, $1 < p \leq q \leq I^2(n)$, and $k \geq 1$.

Throughout this paper we assume that initially each item of $I(n)$ information content is contained in a distinct processor. In such case $k = \frac{pq}{I(n)}$ represents how much of the mesh is filled with data.

Let $\mathcal{P}$ be solved, AT^2 optimally, on a reconfigurable mesh of size $p \times q$ in $O(T)$ time. Then

$$pqT^2 = \frac{q}{p} I^2(n) \ .$$

This implies

$$T = \frac{I(n)}{p} = \frac{q}{k} \ . \tag{1}$$

We now present the formal definition of *adaptive* algorithms developed in our paper [5] without including the analytical support for space constraint.

Definition 1 *Consider an arbitrary problem $\mathcal{P}$ of size n with information content $I(n)$. An algorithm $\mathcal{A}$ is called* adaptive *if $\mathcal{A}$ takes $O\left(\frac{I(n)}{p}\right) \equiv O\left(\frac{q}{k}\right)$ time to solve $\mathcal{P}$ on a reconfigurable mesh of size $p \times q$, where $\sqrt{I(n)} \leq p \leq q \leq I(n)$ and $k = \frac{pq}{I(n)}$.*

Now, from equation (1) we find that $\frac{p}{k} = \frac{I(n)}{q}$. From Definition 1 we also find that $q \leq I(n)$. We thus can conclude that

$$k \leq p \tag{2}$$

which is an important relation assumed in Algorithm 2 below.

The study of adaptive algorithms on reconfigurable meshes reveals that the discussion of optimality issues in mesh-connected networks should not be limited to any specific class or model. It is obvious that existing AT^2 optimal algorithms on reconfigurable mesh will play significant role in the development of future adaptive algorithms but we must consider optimal algorithms on linear arrays and ordinary meshes as well.

Adaptive algorithms can be developed from scratch. But we are interested in designing adaptive algorithms mainly by threading optimal algorithms on linear arrays, ordinary meshes, and reconfigurable meshes. We use the following algorithmic structure in developing adaptive algorithms for specific problems:

Algorithm 2 Generic Adaptive Algorithm

Principal Module: *Here $\mathcal{P}$ of size n is solved on a reconfigurable mesh of size $p \times q$ where $p \leq q$ and $pq = kI(n)$. It is assumed that $I(n)$ items of information are contained in the first $\frac{I(n)}{p}$ columns where each processor $PE_{i,j}$, $0 \leq i < p$ and $0 \leq j < \frac{I(n)}{p}$, contains exactly one item of information.*

1p Divide the mesh of size $p \times q$ into $\frac{q}{k}$ submeshes of size $p \times k$ each.

2p Distribute the $I(n)$ information content equally among the submeshes in such a way that each processor $PE_{i,jk}$ of the main mesh, $0 \leq i < p$ and $0 \leq j < \frac{q}{k}$, receives one item of information. This ensures that each submesh of size $p \times k$ now contains exactly p items of information in its first column.

3p Solve the subproblem with p items of information in each submesh of size $p \times k$ using the algorithm of the supporting layer in parallel.

4p Merge the solutions of the $\frac{q}{k}$ subproblems using the entire mesh of size $p \times q$.

Supporting Module: *Here $\mathcal{P}$ of size at most p is solved on a reconfigurable mesh of size $p \times k$ where $k \leq p$. It is assumed that p items of information are contained in the first column where each processor contains exactly one item of information.*

1s Divide the mesh of size $p \times k$ into $\frac{p}{k}$ submeshes of size $k \times k$ each.

2s Distribute the p items of information equally among the submeshes in such a way that each submesh of size $k \times k$ contains exactly k items of information in its top row.

3s Solve the subproblem with k items of information in each submesh of size $k \times k$ in parallel.

4s Merge the solutions of the $\frac{p}{k}$ subproblems using the entire mesh of size $p \times k$.

The above generic Algorithm 2 assumes $\frac{p}{k}$ and $\frac{q}{k}$ to be integers for the sake of simplicity.

As mentioned in Definition 1, the main goal of Algorithm 2 is to achieve $O\left(\frac{q}{k}\right)$ run time. Phases 1p and 1s of Algorithm 2 need no inter-processor communication and thus these can be done in constant time.

As $\frac{I(n)}{p} = \frac{q}{k}$ by equation (1), phase 2p can be considered as a problem of transferring column j to column kj for all $j : 0 \leq j < \frac{q}{k}$. Now each of the column transfer can be done in constant time by p row broadcasts in parallel. So, phase 2p can be done in $O\left(\frac{q}{k}\right)$ time.

Phase 1s not only divides the mesh of size $p \times k$ into $\frac{p}{k}$ submeshes of size $k \times k$ but also distributes p items of information equally among these submeshes such that every submesh contains k items of information in its first column. Hence, phase 2s can be done in two steps. In the first step, the item of information in processor $PE_{i,0}$ of each submesh of size $k \times k$ is transferred to processor $PE_{i,i}$ in parallel by row broadcasts for all $i : 0 \leq i < k$. In the second step, the item of information in each processor $PE_{j,j}$ is transferred to processor $PE_{0,j}$ in parallel by column broadcasts for all $j : 0 \leq j < k$. So, it can be concluded that phase 2s takes $O(1)$ time.

If we can fairly assume that a constant time algorithm exists to solve phase 3s (possibility of such algorithm can be derived from equation (1)) then the only challenge that remains in designing an adaptive algorithm for any specific problem is to solve phases 4p and 4s in $O\left(\frac{q}{k}\right)$ time as phase 3s degenerates to phases 1s–4s.

Phases 3p and 4p and phases 3s and 4s should not necessarily be simple implementations of the many-way divide-and-conquer strategy as mentioned in Algorithm 2. In many cases, it is more efficient to replace these phases by complex iteration if the motivation is to solve a large problem instance with the solution of smaller problem instances.

Using the framework of generic adaptive Algorithm 2 we developed an adaptive sorting algorithm based on Marberg and Gafni's rotatesort [4] algorithm on ordinary meshes in [5]. In the next section, the generality of Algorithm 2 is again illustrated by developing a new efficient adaptive sorting algorithm based on Schnorr and Shamir's sorting algorithm on ordinary meshes.

4 A New Adaptive Sorting Algorithm

In this section we develop a new adaptive sorting algorithm, using the framework of generic adaptive Algorithm 2, to sort n items AT^2 optimally on a reconfigurable mesh of size $p \times q$, $p \leq q$ and $pq = kn$, based on the efficient AT^2 optimal sorting Algorithm 1 of Schnorr and Shamir on ordinary meshes.

Algorithm 1 of Schnorr and Shamir uses a constant number of linear transformations in the form of sorting and cyclic rotation along rows or columns. Besides these, Algorithm 1 also recursively degenerates into sorting problems on smaller meshes. We further assume $M \geq N^{3/4}$ as mentioned in Section 2.2. On the contrary, if $M \ngeq N^{3/4}$ then $N > M^{4/3} \geq M^{3/4}$ then Algorithm 1 can easily be adapted by considering blocks (Figure 3(a)) of size $M^{3/4} \times M^{3/4}$ instead and transposing all row(column) operations into column(row) operations.

The following two lemmas play important roles in the development of adaptive sorting algorithm presented in this section:

Lemma 1 *Let s items are stored in some s processors of a reconfigurable linear array of $m \geq s$ processors. Then these items can be sorted in $O(s)$ time.*

Proof. A straightforward emulation of odd-even transposition sorting algorithm [3] solves the problem. ■

Lemma 2 *Sorting m items in the first row of a reconfigurable mesh of size $m \times m$ can be done in $O(1)$ time.*

Proof. See in [2, 8, 9]. ■

In this section we further assume $k = 2^{4r}$, $p = 2^{4s}$, and $q = 2^{4t}$ where r, s, and t are integers and $r \leq s \leq t$ for the sake of simplicity.

The principal module and the supporting module of this adaptive algorithm are presented in Sections 4.2 and 4.1 respectively.

4.1 The Supporting Module

In this section we develop an adaptive algorithm to sort p items on a reconfigurable mesh of size $p \times k$, $k \le p$, according to the supporting module of Algorithm 2. We assume that phases 1s and 2s of Algorithm 2 are already completed, i.e., the mesh is divided into $\frac{p}{k}$ submeshes of size $k \times k$ each and the given p items in the first column are distributed in such a way that each processor $PE_{ik,j}$, $0 \le i < \frac{p}{k}$ and $0 \le j < k$, receives an item. This distribution of items transformed the column of p items into an array of $\frac{p}{k} \times k$ items where each row is separated by k rows. Now, phases 3s and 4s of Algorithm 2 emulate Algorithm 1 of Schnorr and Shamir. As phase 2 of Algorithm 1 can be obtained by cyclic rotation of each row in parallel, this emulation requires only the following basic operations in various phases of Algorithm 1:

If $k \ge \left(\frac{p}{k}\right)^{3/4}$

1. Sort $\left(\frac{p}{k}\right)^{3/4} \times \left(\frac{p}{k}\right)^{3/4}$ items using a submesh of size $k\left(\frac{p}{k}\right)^{3/4} \times \left(\frac{p}{k}\right)^{3/4}$.
2. Rotate $\frac{p}{k}$ items using a submesh of size $p \times 1$.
3. Sort k items in a row using a submesh of size $k \times k$.
4. Sort $\left(\frac{p}{k}\right)^{3/4} \times 2\left(\frac{p}{k}\right)^{3/4}$ items using a submesh of size $k\left(\frac{p}{k}\right)^{3/4} \times 2\left(\frac{p}{k}\right)^{3/4}$.
5. Sort $\frac{p}{k}$ items using a submesh of size $p \times 1$.
6. Perform $2\left(\frac{p}{k}\right)^{3/4}$ steps of the odd-even transposition sort.

Else $(\Rightarrow \frac{p}{k} > k^{3/4})$

7. Sort $k^{3/4} \times k^{3/4}$ items using a submesh of size $kk^{3/4} \times k^{3/4}$.
8. Rotate k items in a row using a submesh of size $k \times k$.
9. Sort $\frac{p}{k}$ items using a submesh of size $p \times 1$.
10. Sort $2k^{3/4} \times k^{3/4}$ items using a submesh of size $2kk^{3/4} \times k^{3/4}$.
11. Sort k items in a row using a submesh of size $k \times k$.
12. Perform $2k^{3/4}$ steps of the odd-even transposition sort.

The problem of rotation can always be transformed into a sorting problem without any slowdown. A rotation, therefore, takes no more time than it does to sort.

Now operations 3, 8, and 11 can be done in $O(1)$ time by Lemma 2. Using Lemma 1, operations 2, 5, and 9 can be done in $O\left(\frac{p}{k}\right)$ time.

Algorithm 1 of Schnorr and Shamir can be applied, by treating the reconfigurable mesh as an ordinary mesh, to solve operation 1 and 4 in $O\left(\left(\frac{p}{k}\right)^{3/4}\right)$ time, operation 7 and 10 in $O\left(k^{3/4}\right) = O\left(\left(\frac{p}{k}\right)^{3/4}\right)$ time.

Theorem 3 *Given p items in the first column of a reconfigurable mesh of size $p \times k$, $k \le p$, these items can be sorted in $O\left(\frac{p}{k}\right)$ time, which is AT^2 optimal.* ■

4.2 The Principal Module

In this section we develop an adaptive algorithm to sort n items on a reconfigurable mesh of size $p \times q$, $p \le q$ and $pq = kn$, according to the principal module of Algorithm 2. We assume that phases 1p and 2p of Algorithm 2 are already completed, i.e., the mesh is divided into $\frac{q}{k}$ submeshes of size $p \times k$ each and the given n items in the first $\frac{n}{p}$ columns are distributed in such a way that each processor $PE_{i,jk}$, $0 \le i < p$ and $0 \le j < \frac{q}{k}$, receives an item. This distribution of items transformed the given n items into an array of $p \times \frac{q}{k}$ items where each column is separated by k columns. Now, phases 3p and 4p of Algorithm 2 emulate Algorithm 1 of Schnorr and Shamir which requires only the following basic operations in various phases of Algorithm 1:

If $p \ge \left(\frac{q}{k}\right)^{3/4}$

1. Sort $\left(\frac{q}{k}\right)^{3/4} \times \left(\frac{q}{k}\right)^{3/4}$ items using a submesh of size $\left(\frac{q}{k}\right)^{3/4} \times k\left(\frac{q}{k}\right)^{3/4}$.
2. Rotate $\frac{q}{k}$ items using a submesh of size $1 \times q$.
3. Sort p items in a column using a submesh of size $p \times k$.
4. Sort $2\left(\frac{q}{k}\right)^{3/4} \times \left(\frac{q}{k}\right)^{3/4}$ items using a submesh of size $2\left(\frac{q}{k}\right)^{3/4} \times k\left(\frac{q}{k}\right)^{3/4}$.
5. Sort $\frac{q}{k}$ items using a submesh of size $1 \times q$.
6. Perform $2\left(\frac{q}{k}\right)^{3/4}$ steps of the odd-even transposition sort.

Else $(\Rightarrow \frac{q}{k} > p^{3/4})$

7. Sort $p^{3/4} \times p^{3/4}$ items using a submesh of size $p^{3/4} \times kp^{3/4}$.
8. Rotate p items in a column using a submesh of size $p \times k$.

9. Sort $\frac{q}{k}$ items using a submesh of size $1 \times q$.
10. Sort $p^{3/4} \times 2p^{3/4}$ items using a submesh of size $p^{3/4} \times 2kp^{3/4}$.
11. Sort p items in a column using a submesh of size $p \times k$.
12. Perform $2p^{3/4}$ steps of the odd-even transposition sort.

From equation (2) we get $k \leq p$. Hence, operations 3, 8, and 11 can be done in $O\left(\frac{p}{k}\right)$ time by Theorem 3. Using Lemma 1, operations 2, 5, and 9 can be done in $O\left(\frac{p}{k}\right)$ time.

Algorithm 1 of Schnorr and Shamir can be applied, by treating the reconfigurable mesh as an ordinary mesh, to solve operation 1 and 4 in $O\left(\left(\frac{q}{k}\right)^{3/4}\right)$ time, operation 7 and 10 in $O\left(p^{3/4}\right) = O\left(\left(\frac{q}{k}\right)^{3/4}\right)$ time.

Theorem 4 *Given n items in the first $\frac{n}{p}$ columns of a reconfigurable mesh of size $p \times q$, $p \leq q$ and $pq = kn$, these items can be sorted in $O\left(\frac{q}{k}\right)$ time, which is AT^2 optimal.* ■

To sort MN items on an ordinary mesh of size $M \times N$, Schnorr and Shamir's Algorithm 1 takes $M + 2N + o(M + N)$ steps [10], while Marberg and Gafni's rotatesort algorithm takes $7M + 7N$ steps [4]. As both the adaptive sorting algorithms, developed in this paper and in [5], use the same framework of generic adaptive Algorithm 2, the adaptive sorting algorithm developed in this paper (based on Algorithm 1) is more efficient than the adaptive sorting algorithm in [5] (based on rotatesort) as the constant factor of the highest order term in the complexity order of Algorithm 1 is lower than that of rotatesort.

5 Conclusion

In [5] we introduced the idea of *adaptive* algorithms which runs on reconfigurable meshes of variable sizes and aspect ratios without compromising the AT^2 optimality. We supported our idea by developing adaptive algorithms for sorting and computing maxima in [5]. In this paper we have developed a new adaptive sorting algorithm which has lower constant associated with the highest order term in the complexity order than our adaptive sorting algorithm in [5].

References

[1] Y. B. Asher, D. Gordon, and A. Schuster. Efficient self-simulation algorithms for reconfigurable arrays. *J. Paral. and Dist. Comp.*, 30:1–22, 1995.

[2] J.-W. Jang and V. K. Prasanna. An optimal sorting algorithm on reconfigurable mesh. *J. Paral. and Dist. Comp.*, 25:31–41, 1995.

[3] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, U.S.A., 1992.

[4] J. M. Marberg and E. Gafni. Sorting in constant number of row and column phases on a mesh. *Algorithmica*, 3:561–572, 1988.

[5] M. M. Murshed and R. P. Brent. Adaptive AT^2 optimal algorithms on reconfigurable meshes. In *Proc. of PDCS'98*, U.S.A., 190–195, 1998.

[6] M. M. Murshed and R. P. Brent. Algorithms for optimal self-simulation of some restricted reconfigurable meshes. In *Proc. of ICCIMA'98*, Australia, 734–744, 1998.

[7] K. Nakano. A bibliography of published papers on dynamically reconfigurable architectures. *Paral. Proc. Lett.*, 5:111–124, 1995.

[8] M. Nigam and S. Sahni. Sorting n numbers on $n \times n$ reconfigurable meshes with buses. *J. Paral. and Dist. Comp.*, 23:37–48, 1994.

[9] S. Olariu and J. L. Schwing. A novel deterministic sampling scheme with applications to broadcast-efficient sorting on the reconfigurable mesh. *J. Paral. and Dist. Comp.*, 32:215–222, 1996.

[10] C. P. Schnorr and A. Shamir. An optimal sorting algorithm for mesh connected computers. In *Proc. of the Eighteenth Annual ACM Symp. on Theory of Computing*, 255–263, 1986.

[11] J. D. Ullman. *Computational Aspects of VLSI.* Computer Science Press, U.S.A., 1984.

[12] B.-F. Wang and G.-H. Chen. Constant time algorithms for the transitive closure and some related graph problems on processor arrays with reconfigurable bus systems. *IEEE Trans. on Paral. and Dist. Sys.*, 1:500–507, 1990.

[13] C. C. Weems et al. The image understanding architecture. *Int. J. of Comput. Vision*, 2:251–282, 1989.