THE AUSTRALIAN NATIONAL UNIVERSITY

TR-CS-97-06

RMSIM: a Serial Simulator for Reconfigurable Mesh Parallel Computers

M. Manzur Murshed and Richard P. Brent

April 1997

Joint Computer Science Technical Report Series

Department of Computer Science
Faculty of Engineering and Information Technology

Computer Sciences Laboratory
Research School of Information Sciences and Engineering

This technical report series is published jointly by the Department of Computer Science, Faculty of Engineering and Information Technology, and the Computer Sciences Laboratory, Research School of Information Sciences and Engineering, The Australian National University.

Please direct correspondence regarding this series to:

Technical Reports
Department of Computer Science
Faculty of Engineering and Information Technology
The Australian National University
Canberra ACT 0200
Australia

or send email to:

Technical.Reports@cs.anu.edu.au

A list of technical reports, including some abstracts and copies of some full reports may be found at:

http://cs.anu.edu.au/techreports/

Recent reports in this series:

TR-CS-97-05 Beat Fischer. *Collocation and filtering — a data smoothing method in surveying engineering and geodesy*. March 1997.

TR-CS-97-04 Stephen Fenwick and Chris Johnson. *HeROD flavoured oct-trees: Scientific computation with a multicomputer persistent object store*. February 1997.

TR-CS-97-03 Brendan D. McKay. *Knight's tours of an 8 $\times$ 8 chessboard*. February 1997.

TR-CS-97-02 Xun Qu and Jeffrey X. Yu. *Mobile file filtering*. February 1997.

TR-CS-97-01 Peter Arbenz and Markus Hegland. *The stable parallel solution of general narrow banded linear systems*. January 1997.

TR-CS-96-09 Ralph Back, Jim Grundy, and Joakim von Wright. *Structured calculational proof*. November 1996.

RMSIM: a Serial Simulator for Reconfigurable Mesh Parallel Computers

M. Manzur Murshed*
Richard P. Brent
Computer Sciences Lab, Research School of Information Science & Engg.
Australian National University, Canberra ACT 0200, Australia
e-mail: murshed@cslab.anu.edu.au

April 15, 1997

Abstract

There has recently been an interest in the introduction of reconfigurable buses to existing parallel architectures. Among them Reconfigurable Mesh (RM) draws much attention because of its simplicity. This paper presents the RMSIM (Reconfigurable Mesh SIMulator), a serial simulator written in C, which permits to study algorithms for restricted 3-dimensional RM, known as mesh of meshes, in a monoprocessor environment. RMSIM is an easy-to-use simulator capable of simulating any algorithm in different axis-orientations within restricted regions. To enhance the debugging facilities RMSIM is equipped with a snapshotter to generate LaTeX pictures of any planar segment of the simulated mesh in any step of program execution.

*Corresponding author.

1 Introduction

It is well-known that interprocessor communications and simultaneous memory accesses often act as bottlenecks in present-day parallel machines. Bus systems have been introduced recently to a number of parallel machines to address this problem. Examples include the *Bus Automaton* [6], the *Reconfigurable Mesh (RM)* [5], the *content addressable array processor* [9], and the *Polymorphic torus* [3]. A bus system is called *reconfigurable* if it can be dynamically changed according to either global or local information.

We have developed a serial simulator for reconfigurable mesh parallel computers, RMSIM (Reconfigurable Mesh SIMulator). This simulator, written in ANSI C, can simulate a subnetwork of a 3-dimensional reconfigurable mesh known as *mesh of meshes* (Section 2). Memory management of the simulator is dynamic and the achievable size of the simulated network is dependent on the memory available to the system. This simulator has an in-built interpreter to execute user written programs. The interpreter is capable of executing a program in different axis-orientations within restricted regions. In defining the programming language we have concentrated mainly on making the effort of transforming the algorithms to equivalent programs straightforward and easy. To aid in debugging, RMSIM is capable of generating LaTeX picture of any planar segment of the mesh in any step while executing a program.

This paper is organized as follows. In the next section we present the basic issues of RM. Section 3 describes the software organization of RMSIM. In Section 4 we present the programming facilities supported by the interpreter of RMSIM. Various facilities for reusing programs are discussed in Section 5. In Section 6 we describe the debugging facilities of RMSIM including the snapshotter. Sections 7-9 describe future development, conclusions, and technical reference respectively.

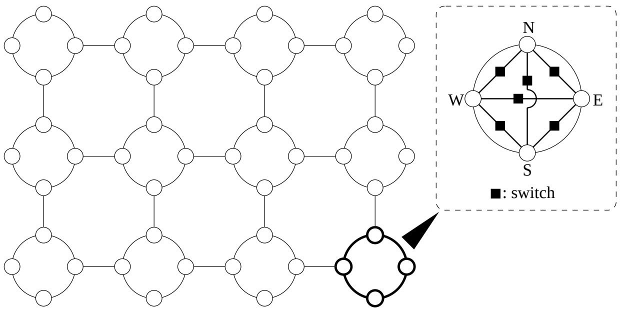

Figure 1: A 3×4 reconfigurable mesh

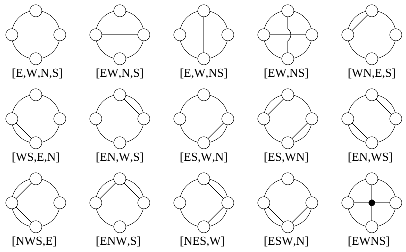

Figure 2: Possible internal connections between the four I/O ports of a PE

2 Preliminaries

The reconfigurable mesh is primarily a two-dimensional mesh of processors connected by reconfigurable buses. In this parallel architecture, a processor element (PE) is placed at the grid points as in the usual mesh connected computers. Each PE is connected to at most four neighbouring PEs 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 PE. The fifteen possible interconnections of I/O ports through switching are shown in Figure 2. Like all bus systems, the behaviour of RM relies on the assumption that the transmission time of a message along a bus is independent of the length of the bus [1].

A reconfigurable mesh operates in the single-instruction multiple-data (SIMD) mode. Besides the reconfigurable switches, each PE has a computing unit with a fixed number of local registers. A single time step

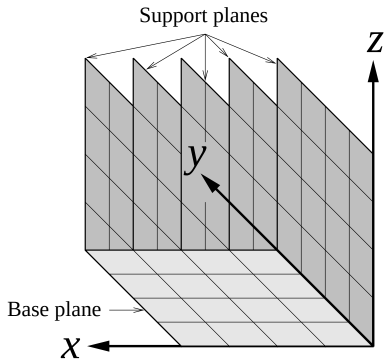

Figure 3: $5 \times 5 \times 5$ *mesh of meshes*

of an RM is composed of the following four substeps:

BUS substep. Every PE switches the internal connectors between I/O ports by local decision.

WRITE substep. Along each bus, one or more PEs on the bus transmit a message of length bounded by the bandwidth of the fixed bus segments as well as the switches. These PEs are called the *speakers*. It is assumed that a collision between several speakers will be detected by all the PEs connected to the bus and the transmitted message will be discarded.

READ substep. Some or all the PEs connected to a bus read the message transmitted by a single speaker. These PEs are called the *readers*.

COMPUTE substep. A constant-time local computation is done by each PE.

Reconfigurable meshes of higher dimension can be constructed in a similar way. A number of interesting algorithms [1, 2, 4, 7, 8] have appeared in the literature for these meshes. In most of the cases [1, 2, 4, 8], 3-dimensional RM with two additional ports U and D along dimension z was considered because of a subnetwork of it, known as *mesh of meshes*, where only planar algorithms are applied on the planes shown in Figure 3. To overcome the implementation issues, optimal simulation of higher dimensional RMs by 2-dimensional ones is given in [7].

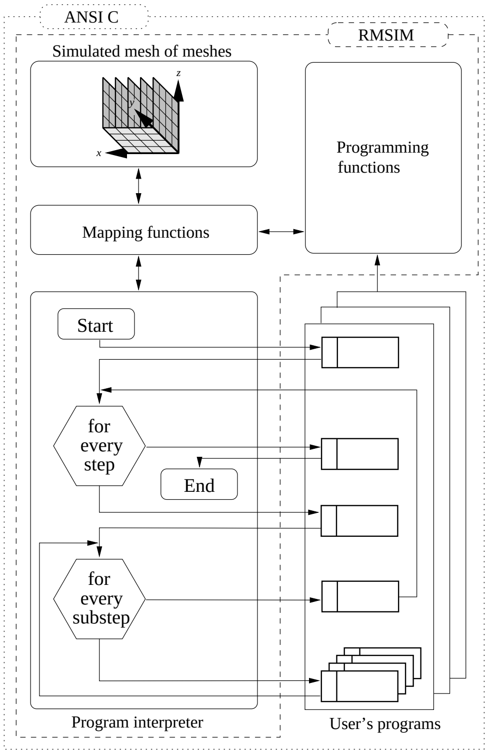

Figure 4: RMSIM software structure

3 Software Structure

RMSIM is primarily an ANSI C library. Although written in ANSI C, the library preserves object oriented programming principles by data hiding. RMSIM provides a simulator to simulate a mesh of meshes, an interpreter to execute programs on the simulated RM, and a set of functions to aid in programming. Because reconfigurable meshes are SIMD machines, a single program is written by the user and this program is executed in all PEs in parallel. In RMSIM, a user is required to write separate functions for each step of the algorithm through the call of specific functions from the library. Inbuilt functions are available for reconfiguring local ports' interconnections, writing along a bus and reading from a bus. Any reference to the simulated RM is carried through some mapping functions to enable the interpreter to execute programs on any possible axis-orientations (Section 5.1). Mapping also allows the interpreter to execute a program within specific region rather than using the whole area of the simulated RM. RMSIM allows a user to take a snapshot of any planar segment of the simulated mesh in LaTeX picture format. Figure 4 shows the general software structure of RMSIM.

4 Programming Facilities

As already stated in the previous section, a user is required to write C functions for each step of the algorithm. Conversion of an algorithm to a program is an easy task in RMSIM. Three functions, *Bus()*, *Write()*, and *Read()*, are available to the user to convert the BUS, WRITE, and READ substeps in the algorithm in a straight forward manner. Functions *GetReg()* and *SetReg()* are available to manipulate any register of a PE. To convert other elements of an algorithm like arithmetic and logical expressions, comparisons etc., the user can use any constructs available in C. Figure 5 shows an example of the natural resemblness of the converted program with the original algorithm.

As shown in Figure 4, the execution loop in the interpreter invokes user defined functions at the start and the end of each step as well as the whole program. The use of these functions is entirely dependent on the user's need. These functions may be suitable for initializing, generating LaTeX picture of a plane of simulated mesh of meshes in some specific step, gathering statistics, debugging etc. As an example, Figure 6, 7 are generated at the end of steps 0 and 1 respectedly while executing the program of Figure 5 on an RM of size 5×6.

5 Program Reusability

Like many other programming languages, RMSIM is capable of using previously written programs into a new program through *subroutine calls*. This enables the user to divide a long algorithm into small pieces. Many RM algorithms appeared in the literature containing references to other published algorithms. This simplified the description of those algorithms significantly. The capability of reusing

1.
- Every processor connects port N with S;
- Every processor at row 0 writes register r_0 to port N;
- Every processor reads port N into register r_0;

2.
- Every processor does the following:
  if register $r_0 = 0$ **then**
  connects port E with W
  else
  connects port W with N and E with S;
- Only the processor $P(0,0)$ does the following:
  if register $r_0 = 0$ **then**
  writes '#' to port E
  else
  writes '#' to port N;
- Every processor reads port E into register r_1;

3.
- Every processor connects port N with S;
- Every processor $P(x,y)$ does the following:
  if register r_1 = '#' **then**
  writes y to port S;
- Every processor at row 0 reads port S into register r_1;

Algorithm: PrefixSum

$\Longrightarrow$

```
#include ⟨stdio.h⟩
#include "rm.h"

static BYTE Step0( void ) {
  switch( MicroStep ) {
  case BUS:   if( Bus( "NS", "E", "W", "U", "D", "" ) )
                return 1;
              break;
  case WRITE: if( y == Sy )
                if( Write( N, GetReg( 0 ) ) )
                  return 1;
              break;
  case READ:  if( Read( N, 0 ) ) return 1;
              break;
  }
  return 0;
}

static BYTE Step1( void ) {
  switch( MicroStep ) {
  case BUS:   if( !GetReg( 0 ) ) {
                if( Bus( "EW", "N", "S", "U", "D", "" ) )
                  return 1;
              }
              else {
                if( Bus( "WN", "ES", "U", "D", "", "" ) )
                  return 1;
              }
              break;
  case WRITE: if( x == Sx && y == Sy )
                if( !GetReg( 0 ) ) {
                  if( Write( E, 99 ) ) return 1;
                }
                else if( Write( N, 99 ) ) return 1;
              break;
  case READ:  if( Read( E, 1 ) ) return 1;
              break;
  }
  return 0;
}
.
.
.
```

Program: PrefixSum (*partial*)

Figure 5: An RM algorithm with the converted program to compute the prefix sums of a binary sequence

programs by RMSIM enables a user to convert those algorithms into programs in similar straightforward fashion.

The capability of reusing programs is embedded into RMSIM by the implementation of an internal subroutine calling system. Besides the housekeeping of the internal stack, this system decides appropriate mapping functions to be applied on while accessing a PE of the simulated RM. the *Axis-orientation mapping* and the *region mapping* are the two mapping classes used by RMSIM.

5.1 Axis-orientation Mapping

To locate a PE into the simulated mesh of meshes, 3-dimensional Cartesian coordinates are used. From now on $PE_{x,y,z}$ will denote the PE at the coordinate (x, y, z). The XY-plane is assumed to be the base plane while the planes perpendicular to it act as the supporting planes. Let this axis-orientation be

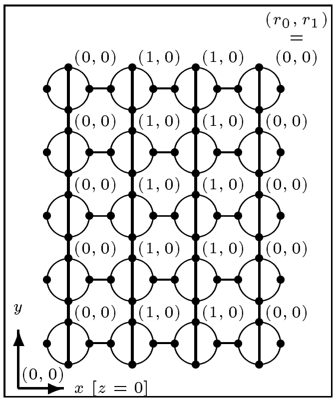

Figure 6: PROG: PrefixSum, STEP: 0

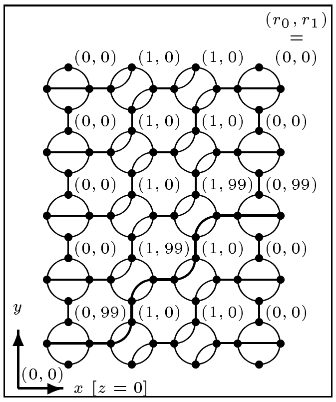

Figure 7: PROG: PrefixSum, STEP: 1

called *XY_Z*. RMSIM allows an user to use different axis-orientation while executing a program e.g. a program can be executed considering *YZ*-plane be the base plane. The six possible axis-orientations are shown in Figure 8. RMSIM automatically selects proper mapping functions while reusing nested programs. Suppose the current and requested axis-orientations be *YZ_X* and *XZ_Y*, RMSIM will then select *YX_Z*.

The problem of computing the ranks of N distinct numbers on an $N \times N \times N$ mesh of meshes exemplifies the axis-orientation mapping. Let the number n_i be stored in $PE_{i,0,0}$, $0 \leq i < N$. Now a row broadcast after a column broadcast can be done to distribute the numbers in the *XY*-plane so that $PE_{i,j,0}$, $0 \leq i, j < N$, receives the pair (n_i, n_j) and then produces 1, if $n_i > n_j$, or 0, otherwise. Ranks can now be computed by executing the program *PrefixSum* in *YZ_X* axis-orientation in every *YZ*-planes to add the comparison values along each column. A few steps of computing the ranks of 4 numbers on an $4 \times 4 \times 4$ mesh of meshes are shown in Figures 9 to 14. Figures 11 to 14 represents the execution of program *PrefixSum* in a different axis-orientation.

5.2 Region Mapping

Enhancement in the power of program reusability through axis-orientation mapping cannot be realised completely if no way is allowed to execute a program in a restricted area of the mesh of meshes e.g. in the previ-

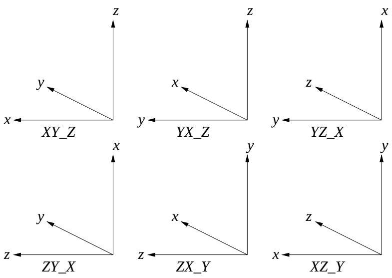

Figure 8: Six possible axis-orientations

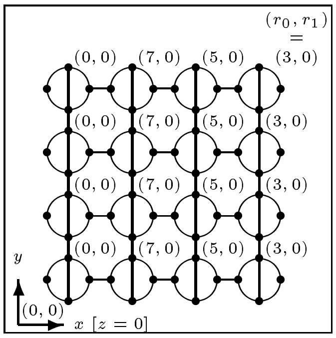

Figure 9: PROG: Rank, STEP: 0

ous example of computing ranks each execution of the program *PrefixSum* uses a specific YZ-plane rather than using the entire mesh of meshes. Executing a program in an enclosed area through *region mapping* can itself be a very useful tool. Suppose, for example, in solving a problem we need to compute the prefix sums of k segments of length m of a sequence of mk numbers. If $mk \times m$ mesh is available, then these k prefix sums can be computed by executing the program *PrefixSum* in k different enclosed areas where each

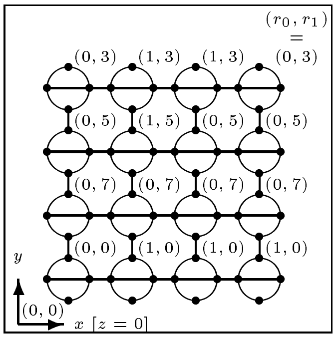

Figure 10: PROG: Rank, STEP: 1

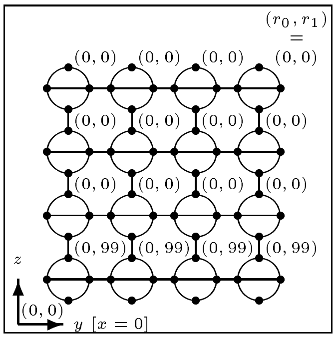

Figure 11: PROG: PrefixSum, STEP: 1

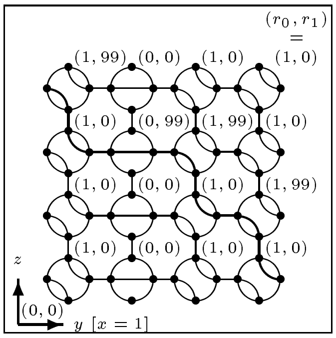

Figure 12: PROG: PrefixSum, STEP: 1

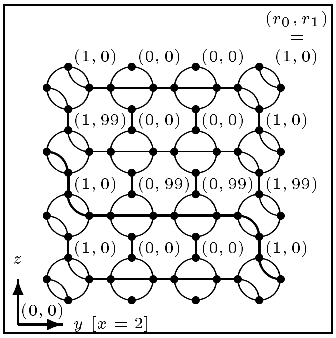

Figure 13: PROG: PrefixSum, STEP: 1

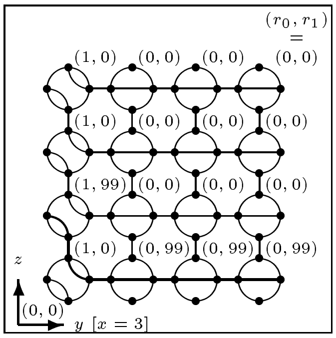

Figure 14: PROG: PrefixSum, STEP: 1

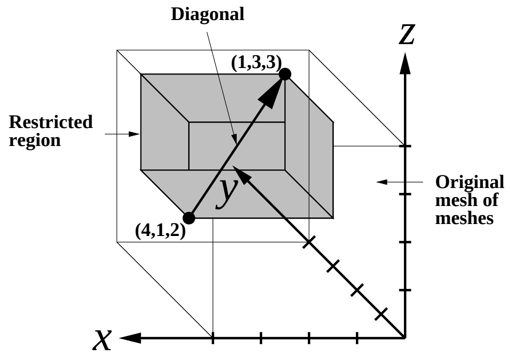

Figure 15: A restricted region in a $5 \times 5 \times 5$ mesh of meshes

area consists of m consecutive columns of the original mesh.

In RMSIM, while writing a program, a user never assumes the length of each dimension. So, while executing a program the user must explicitly define the area of execution by giving the coordinates of the two PEs along the diagonal. In the program interpreter each step is executed in the PEs of the restricted area in the following order starting from the starting PE along the diagonal:

Loop along 3rd axis
    Loop along 2nd axis
        Loop along 1st axis

Actual axes to be considered in place of 1st, 2nd, and 3rd axes are resolved according to the current axis-orientation. In *ZX_Y* axis-orientation 1st, 2nd, and 3rd axes are taken as z, x, and y axes. The step of each loop is either 1 or -1 depending on coordinates of the PEs along the diagonal. For example, the region defined in Figure 15 will generate the following loop structure if the current axis-orientation is *XY_Z*:

for $z = 2$ to 3 step 1
    for $y = 1$ to 3 step 1
        for $x = 4$ to 1 step -1

Axis-orientation mapping and region mapping are transparent to the user. While writing a program a user should always assume that the acting axis-orientation is *XY_Z* and x, y, and z axes are the 1st, 2nd, and 3rd axes respectively. These assumptions make the task of programming extremely easy. While executing a program a user should only be careful about the mapped locations of *data/source* PEs and *result/destination* PEs. In a region, making any of the loop step -1 may produce interesting results. For example, if the program *PrefixSum* is executed in a region where the loop step of the 1st axis is -1, then the program will compute the sums in opposite direction.

6 Debugging Facilities

RMSIM generates error codes while executing a program if problem occurs. Besides this standard technique, RMSIM is equipped with a snapshotter. The snapshotter can be used to generate LaTeX picture of any planner portion of the simulated mesh of meshes in any step of program execution. The generated pictures are scalable and can show the content of at most two registers of each PE. Figures 6, 7, 9, ..., 14 are all generated by the snapshotter.

7 Future Development

The following extensions and enhancements of RMSIM may be implemented in the future:

- Extension of the capability of RMSIM to handle higher dimensional meshes. This can be done in two ways. Either RMSIM will simulate higher dimensional meshes or it will resolve the extra dimensions into the current mesh of meshes through another layer of general purpose mapping.
- RMSIM should provide some self simulation techniques so that a program requiring mesh of meshes of size $n_x \times n_y \times n_z$ can be executed on a simulated mesh of meshes of size $p_x \times p_y \times p_z$ where $p_x \leq n_x$, $p_y \leq n_y$, and $p_z \leq n_z$.
- Enhancement to the basic RM such as nonmonotonic RM [1] should be handled by RMSIM.
- Besides the snapshotter, RMSIM should be equipped with a 3D-viewer.
- Finally, a library of predefined programs from various application classes should be added to the RMSIM.

8 Conclusion

RMSIM is an easy-to-use serial simulator which can simulate a restricted form of 3-dimensional reconfigurable mesh known as mesh of meshes. Besides simulating, RMSIM provides a program interpreter which can execute programs in any possible axis-orientation within an enclosed region. RMSIM can generate snapshots of any planar segment of the simulated mesh in LaTeX picture format. RMSIM is being used in the study of algorithms for RM and is being evaluated and improved.

9 Technical Reference

This software is available by anonymous ftp: cslab.anu.edu.au in the directory /pub/Manzur/RMSIM (free distribution).

References

[1] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster. The power of reconfiguration. *Journal of Parallel and Distributed Computing*, 13:139–153, 1991.

[2] Yen-Cheng Chen and Wen-Tsuen Chen. Constant time sorting on reconfigurable

meshes. *IEEE Transactions on Computers*, 43:749–751, 1994.

[3] Massimo Maresca. Polymorphic processor arrays. *IEEE Transactions on Parallel and Distributed Systems*, 4:490–506, 1993.

[4] Mark S. Merry and Johnnie Baker. A constant time sorting algorithm for a three dimensional reconfigurable mesh and reconfigurable network. *Parallel Processing Letters*, 5:401–412, 1995.

[5] Russ Miller, V. K. Prasanna Kumar, Dionisios I. Reisis, and Quentin F. Stout. Data movement operations and applications on reconfigurable VLSI arrays. In *Proc. International Conference on Parallel Processing*, pages 205–208, 1988.

[6] J. Rothstein. Bus automata, brains, and mental models. *IEEE Trans. Syst. Man Cybern*, 18:522–531, 1988.

[7] Ramachandran Vaidyanathan and Jerry L. Trahan. Optimal simulation of multidimensional reconfigurable meshes by two-dimensional reconfigurable meshes. *Information Processing Letters*, 47:267–273, 1993.

[8] Biing-Feng Wang, Gen-Huey Chen, and Ferng-Ching Lin. Constant time sorting on a processor array with a reconfigurable bus system. *Information Processing Letters*, 34:187–192, 1990.

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