THE AUSTRALIAN NATIONAL UNIVERSITY

TR-CS-97-16

Algorithms for Optimal Self-Simulation of Some Restricted Reconfigurable Meshes

M. Manzur Murshed and Richard P. Brent

July 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-97-15 Peter Strazdins. *Reducing software overheads in parallel linear algebra libraries*. July 1997.

TR-CS-97-14 Michael K. Ng and William F. Trench. *Numerical solution of the eigenvalue problem for Hermitian Toeplitz-like matrices*. July 1997.

TR-CS-97-13 Michael K. Ng. *Blind channel identification and the eigenvalue problem of structured matrices*. July 1997.

TR-CS-97-12 Michael K. Ng. *Preconditioning of elliptic problems by approximation in the transform domain*. July 1997.

TR-CS-97-11 Richard P. Brent, Richard E. Crandall, and Karl Dilcher. *Two new factors of Fermat numbers*. May 1997.

TR-CS-97-10 Andrew Tridgell, Richard Brent, and Brendan McKay. *Parallel integer sorting*. May 1997.

Algorithms for Optimal Self-Simulation of Some Restricted Reconfigurable Meshes

M. Manzur Murshed*
Richard P. Brent

Computer Sciences Lab, Research School of Information Sciences & Engg.
The Australian National University, Canberra ACT 0200, Australia
e-mail: {murshed, rpb}@cslab.anu.edu.au
Tel: +61 6 279 8636, Fax: +61 6 279 8651

July 15, 1997

Abstract

There has recently been an interest in the introduction of reconfigurable buses to existing parallel architectures. Among them the Reconfigurable Mesh (RM) draws much attention because of its simplicity. However the wide acceptance of RM depends on its scalability through self-simulation. This paper presents a simple self-simulation algorithm which can self-simulate the monotonic RM model optimally and the piecewise-monotonic RM model asymptotically optimally. We claim here that our algorithm preserves the essence of configurational computation and uses less broadcasts than simulation by the contraction and linear-connected component computation methods [1].

Keywords: Reconfigurable mesh; Simulation; Parallel algorithms; Parallel architectures

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* [15], the *Reconfigurable Mesh (RM)* [9], the *content addressable array processor* [17], and the *Polymorphic torus* [8]. A bus system is called *reconfigurable* if it can be dynamically changed according to either global or local information.

*Corresponding author.

Can these models be the basis for the design of next generation of massively parallel computers? Perhaps the answer depends on the most fundamental related issue of *virtual parallelism* or *self-simulation*: Given an algorithm which is designed for a large RM, can it be executed efficiently on a smaller RM?

In [1] Ben-Asher et al. present optimal self-simulation algorithms for the HV-RN and LRN models (defined in Section 2). They also present a self-simulation algorithm for the RN model with an extra slowdown which is polylogarithmic in the size of the simulated mesh. In self-simulating the HV-RN model they apply a standard simulation technique, known as the contraction method, where a single processing element (PE) simulates a submesh. This method destroys the beauty of *configurational computation* [16], a key strength of RM-algorithms, as most of the bus segments are configured virtually in a single PE. Self-simulation of the LRN model is done by windowing the simulating mesh over the simulated mesh in a snakelike order while computing linear-connected components. This introduces extra broadcasts in addition to the necessary windowing broadcasts.

In this paper we present a self-simulation algorithm SIMPLE where the simulating mesh is used as a window over the simulated mesh. The key issue in Algorithm SIMPLE is to determine a suitable sequence of windowing so that after a finite number of steps correct self-simulation is achieved. We show that Algorithm SIMPLE can simulate the monotonic RM model optimally. We also show that Algorithm SIMPLE can self-simulate the piecewise-monotonic RM model and the optimal slowdown can be achieved if the simulating mesh is small compared to the simulated mesh which is a desirable property of self-simulation.

This paper is organized as follows. In the next section we present the basic issues of RM and present the monotonic and piecewise-monotonic RM models. The section also includes definitions associated with the self-simulation problem. In Section 3.1 mapping of the simulated mesh into the simulating mesh is described in detail. The Algorithm SIMPLE is presented in Section 3.2. In Section 3.3 we show that Algorithm SIMPLE can self-simulate the monotonic and piecewise-monotonic RM models optimally. Section 4 concludes the paper.

2 Preliminaries

For the sake of completeness, we briefly define the reconfigurable mesh and give definitions of the problem of self-simulation.

2.1 Reconfigurable Mesh

The reconfigurable mesh is primarily a two-dimensional mesh of PEs connected by reconfigurable buses. In this parallel architecture, a 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 $\mathcal{N}$ & $\mathcal{S}$ along dimension x and $\mathcal{E}$ & $\mathcal{W}$ along dimension y. These fixed bus segments

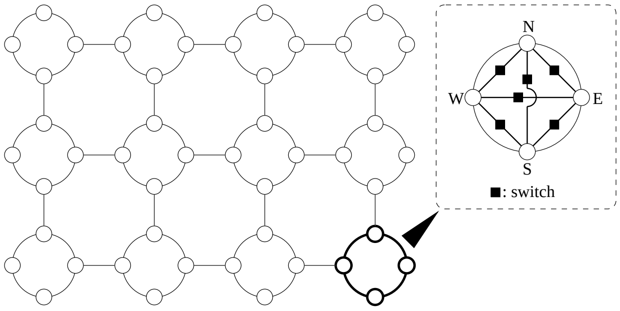

Figure 1: A reconfigurable mesh of size 3×4.

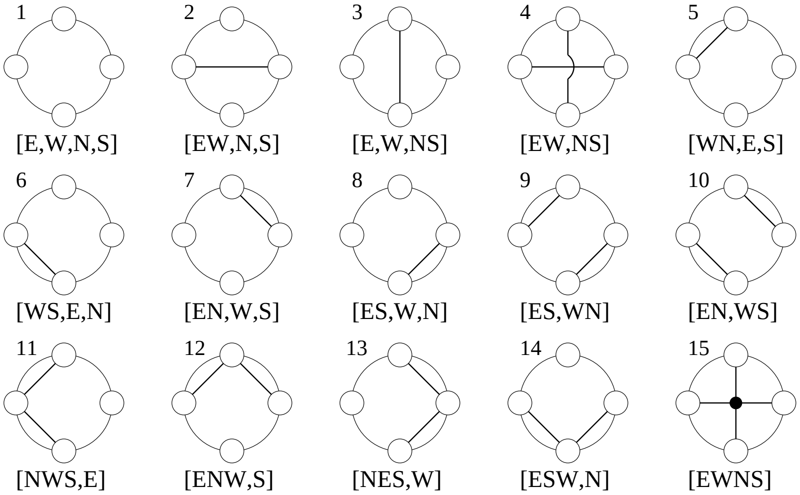

Figure 2: Possible fifteen local configurations of a PE.

are building blocks of larger bus components which are formed through switching, determined entirely by 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, also known as *local configurations*, 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 [2].

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 of an RM is composed of the following four substeps:

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

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

READ substep. Some or all the processors connected to a bus read the message transmitted by a single speaker. These processors are called the *readers*. In this paper we assume that each reader can detect whether the designated port carries any signal or not. A reader is allowed to read only when it detects a signal in the associated port.

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

The general reconfigurable mesh model, as presented above, does not specify the exact operation of the switches. The following basic variants are proposed in [1]:

Horizontal-Vertical RM (HV-RN Model). Buses are formed along either rows or columns, but may not contain building blocks from both dimensions. This model supports local configurations 1 to 4 in Figure 2.

Linear RM (LRN Model). A bus may consist of any connected path of edges, not only vertical or only horizontal. The buses are however only *linear*, i.e., a fixed bus segment is attached to at most one other fixed bus segment at each end. Local configurations 1 to 10 in Figure 2 are supported by this model.

General RM (RN Model). A configuration of buses is any partition of the network into edge disjoint subgraphs, so buses are not necessarily linear. This model supports all the fifteen local configurations in Figure 2.

The variants defined above depends solely on local configurations of ports. Now we present two additional models where restrictions are imposed over the global characteristics of the buses.

Definition 1 *A function $f(x)$ is called* positive monotonic *w.r.t. x if $f(x_1) \geq f(x_2)$ whenever $x_1 \geq x_2$. Similarly a function $f(x)$ is called* negative monotonic *w.r.t. x if $f(x_1) \leq f(x_2)$ whenever $x_1 \geq x_2$. A function $f(x)$ is* monotonic *w.r.t. x if it is either positive or negative monotonic w.r.t. x.*

Definition 2 *A function $f(x)$ is called* piecewise-monotonic *w.r.t. x if axis-x can be divided into successive ranges such that $f(x)$ is positive monotonic in alternate ranges and negative monotonic in the rest of ranges.*

Monotonic RM Model. Each bus represents a monotonic function w.r.t. either row and/or column index within a range.

Piecewise-Monotonic RM Model. Every bus represents a piecewise-monotonic function w.r.t. either row or column index within a range. Moreover in any step all buses represent functions w.r.t. same index.

Observe that both the models are included in the LRN model. Also observe that the HV-RN model is included in the monotonic RM model which is again included in the piecewise-monotonic RM model.

We believe that the monotonic and piecewise-monotonic RM models are defined here for the first time but many published algorithms for the LRN models can readily be used in these models without any modifications or with very small modifications. Among them PARITY algorithms [7, 14], conversion between number representations algorithms [6], prefix-sums algorithm [11], sorting algorithms [12, 13] etc. can be adapted into the monotonic as well as the piecewise monotonic RM models and prefix-remainders algorithm [11], integer summing algorithms [6, 11], integer multiplication algorithm [5], sorting algorithm [6], algorithms based on function decomposition [3], HISTOGRAM algorithm [4] etc. can be applied into the piecewise-monotonic RM model only. Moreover it is quite obvious that all the algorithms suitable for the HV-RN model are applicable to both the models.

2.2 Problem Definition

Let $RM_C^{A\times B}$ denote a reconfigurable mesh of A rows and B columns with each PE having C registers.

Definition 3 *The self-simulation problem of RM is to step-by-step simulate $RM_R^{M\times N}$ by $RM^{P\times Q}_{\Theta\left(R\lceil\frac{M}{P}\rceil\lceil\frac{N}{Q}\rceil\right)}$ where $P \leq M$, $Q \leq N$, and the computing power of the PEs and the bus bandwidth (not less than $\log MN$) are assumed to be equivalent in both the meshes.*

To simplify the exposition $\frac{M}{P}$ and $\frac{N}{Q}$ are assumed to be integers. If the memory requirement of the simulating RM is bounded as defined in the above definition then the *slowdown* remains as the key issue.

Definition 4 *We say that reconfigurable mesh R_1 is simulated by R_2 with slowdown S if the result for any algorithm A_1 on R_1 is achieved through the execution of a step-by-step simulation algorithm A_2 on R_2 in which each step of A_1 is simulated with slowdown at most S.*

Obviously the self-simulation of $RM_R^{M\times N}$ by $RM^{P\times Q}_{\Theta\left(R\frac{M}{P}\frac{N}{Q}\right)}$, $P \leq M$ and $Q \leq N$, is said to be optimal if the slowdown is $\Theta\left(\frac{M}{P}\frac{N}{Q}\right)$. A smaller slowdown would lead to a serial algorithm contradicting lower bound.

3 Self-Simulation Algorithm

Let the PEs of the simulated mesh $RM_R^{M\times N}$ and the simulating mesh $RM^{P\times Q}_{\Theta\left(R\frac{M}{P}\frac{N}{Q}\right)}$ be denoted by the matrices $R[0:M-1, 0:N-1]$ and $S[0:P-1, 0:Q-1]$ respectively. Let $R(x, y)$, $0 \leq x < M$ and $0 \leq y < N$, denote the PE at the intersection of row

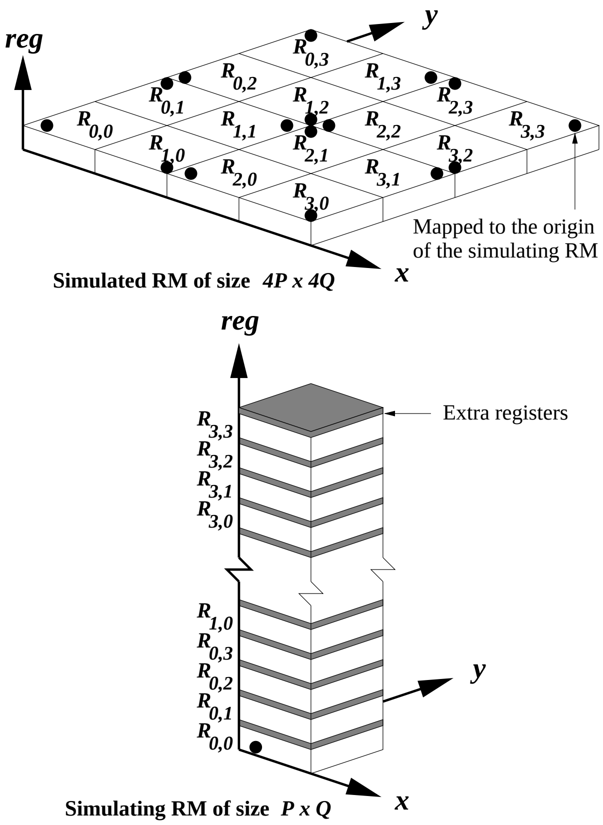

Figure 3: Mapping of the simulated RM into the simulating RM.

x and column y of the simulated mesh. Similarly let the PE at the intersection of row x and column y of the simulating mesh be denoted by $S(x, y)$, $0 \le x < P$ and $0 \le y < Q$.

We first develop necessary mapping techniques for the simulation. An algorithm SIMPLE is then presented and finally restrictions are imposed on the general RM to make the algorithm SIMPLE optimal.

3.1 Mapping of the Simulated RM into the Simulating RM

In this paper the following two functions play important role in mapping meshes:

$$FOLD(a, b) = \begin{cases} a \bmod b & \text{if } a \text{ div } b \text{ is even} \\ b - 1 - (a \bmod b) & \text{otherwise} \end{cases}$$

$$UNFOLD(a, b, c) = \begin{cases} bc + a & \text{if } c \text{ is even} \\ b(c+1) - 1 - a & \text{otherwise} \end{cases}$$

Let the simulated RM be divided into $\frac{M}{P}\frac{N}{Q}$ nonoverlapping submeshes $R_{i,j}$ of size $P \times Q$ containing the processing elements $R[iP : (i+1)P - 1, jQ : (j+1)Q - 1]$ for $0 \le i < \frac{M}{P}$ and $0 \le j < \frac{N}{Q}$. Now the simulated mesh is mapped into the simulating mesh in such a way that the processing element $R(x,y)$ is simulated by $S(FOLD(x,P), FOLD(y,Q)$ for $0 \leq x < M$ and $0 \leq y < N$. This ensures one-to-one PE mapping of each submesh $R_{i,j}$ into the simulating RM and whenever the simulating RM simulates the submesh $R_{i,j}$ the processing element $S(x,y)$ simulates the processing element $R(UNFOLD(x,P,i), UNFOLD(y,Q,j))$ for $0 \leq i < \frac{M}{P}$, $0 \leq j < \frac{N}{Q}$, $0 \leq x < M$ and $0 \leq y < N$. The same benefits can also be achieved through straightforward mapping without using any folding techniques. But the mapping presented here has its unique characteristics - the external neighbours of a boundary PE, p of the submesh $R_{i,j}$ are mapped in the same simulating PE where p is also mapped, $0 \leq i < \frac{M}{P}$ and $0 \leq j < \frac{N}{Q}$. This keeps the broadcasts of simulation data low in the expense of the introduction of a mapping of the ports due to the change in the direction of x- and/or y-axes in some of the mapped submeshes. For the submesh $R_{i,j}$, $0 \leq i < \frac{M}{P}$ and $0 \leq j < \frac{N}{Q}$, the ports are mapped as follows:

$$MAPPORT(\mathcal{E}, i, j) = \begin{cases} \mathcal{E} & \text{if } i \text{ is even} \\ \mathcal{W} & \text{otherwise} \end{cases}$$

$$MAPPORT(\mathcal{W}, i, j) = \begin{cases} \mathcal{W} & \text{if } i \text{ is even} \\ \mathcal{E} & \text{otherwise} \end{cases}$$

$$MAPPORT(\mathcal{N}, i, j) = \begin{cases} \mathcal{N} & \text{if } j \text{ is even} \\ \mathcal{S} & \text{otherwise} \end{cases}$$

$$MAPPORT(\mathcal{S}, i, j) = \begin{cases} \mathcal{S} & \text{if } j \text{ is even} \\ \mathcal{N} & \text{otherwise} \end{cases}$$

Now each PE of the simulating mesh simulates $\frac{M}{P}\frac{N}{Q}$ PEs of the simulated mesh. We assume that the k-th register of the simulated processing element $R(x,y)$ is mapped into the $\left(\left(x\frac{N}{Q} + y\right)(R+\varepsilon) + k\right)$-th register of the corresponding simulating processing element $S(x \bmod P, y \bmod Q)$ where $0 \leq x < M$, $0 \leq y < N$, $0 \leq k < R$, and ε is a small integer. If *register* is considered as the third axis then the above register mapping stacks the submeshes $R_{i,j}$ over the simulating RM in column-major order (Figure 3) and each submesh is alloted an extra ε registers per PE for simulation purpose.

3.2 SIMPLE: a Self-Simulation Algorithm

Let B denote the set of all the boundary PEs of the simulating RM, i.e., $B = \{S(x,y) \mid x = 0 \vee x = P-1 \vee y = 0 \vee y = Q-1\}$. Let a port, t of a boundary PE, p be called **port* if t is not connected to any port external to p. Every boundary PE has exactly one *port except $S(0,0)$, $S(0,Q-1)$, $S(P-1,0)$, and $S(P-1,Q-1)$ which have two *ports each. Whenever the submesh $R_{i,j}$ is simulated, for each boundary PE, two registers from the ε extra registers are allocated for each *port. Let these special registers be called **reg1* and **reg2*.

Let for each step s of an RM-algorithm A, b(s), r(s), w(s), and c(s) denote the BUS, READ, WRITE, and COMPUTE substeps respectively. In the reminder whenever we mention that some steps or substeps are executed in the simulating RM while simulating a specific submesh, it is assumed that the references to any register, to any port and to the coordinates of any PE are mapped accordingly.

We now present a self-simulation algorithm without considering any specific model in mind. In Section 3.3 we show that this algorithm can optimally self-simulate some classes of RM where restrictions are imposed over the global characteristics of bus reconfigurations.

ALGORITHM: SIMPLE(RM-algorithm: A **)**

1 For each step $s \in A$ do the following

1.1 For each boundary PE $\in B$ do the following in parallel

For each *port t do the following

For each mapped submesh $R_{i,j}$, $0 \leq i < \frac{M}{P}$ and $0 \leq j < \frac{N}{Q}$, set *reg1 to 0;

1.2 Generate a finite sequence of pairs $(i_1, j_1), (i_2, j_2), \ldots, (i_L, j_L)$ of length L where $\forall K : 0 \leq i_k < P$ and $0 \leq j_k < Q$.

1.3 For each pair (i_k, j_k) do the following on the mapped submesh R_{i_k, j_k}

1.3.1 Execute b(s);

1.3.2a Execute w(s);

1.3.2b For each boundary PE $\in B$ do the following in parallel

For each *port, t do the following

if *reg1 $= 1$ then write *reg2 to port t;

1.3.3a Execute r(s);

1.3.3b For each boundary PE $\in B$ do the following in parallel

For each *port, t do the following

if t senses signal then set *reg1 to 1 else set *reg1 to 0;

if *reg1 $= 1$ then read port t into *reg2;

1.3.4a Execute c(s);

1.3.4b For each boundary PE $\in B$ do the following in parallel

For each *port, t do the following

Copy *reg1 and *reg2 into the similar registers, allocated for t, of the neighbouring mapped submeshes;

Step 1.2 is the most crucial part of the above algorithm. Generating a sequence of length L which leads to correct self-simulation depends on many factors which are discussed in the next section.

The order of step 1.1 of Algorithm SIMPLE is $\Theta\left(\frac{M}{P}\frac{N}{Q}\right)$. Let the order of step 1.2 be $O_{1.2}$. Steps 1.3.2a and 1.3.2b can be done in a single WRITE substep. Similarly steps 1.3.3a and 1.3.3b can be done in a single READ substep while steps 1.3.4a and 1.3.4b can be done in a single COMPUTE substep. So all the substeps of step 1.3 can be executed in order $O(1)$. Hence the order of step 1.3 is $O(L)$.

Lemma 1 *Slowdown of Algorithm SIMPLE is* $\max\left(\Theta\left(\frac{M}{P}\frac{N}{Q}\right), O_{1.2}, O(L)\right)$.

3.3 Optimal Self-Simulation of Some Restricted RM

Let $RSEQ(i)$ and $CSEQ(j)$ denote the sequences of pairs $(i,0)$, $(i,1)$, ..., $(i,\frac{N}{Q}-1)$ and $(0,j)$, $(1,j)$, ..., $(\frac{M}{P}-1,j)$ respectively. Let $\overline{S}$ denote the sequence S in reverse order, $S_1+S_2+\cdots+S_n$ denote the concatenation of sequences S_1, S_2, ..., S_n in order and S^k denote the sequence $\underbrace{S+S+\cdots+S}_{k \text{ times}}$.

Theorem 1 *Any monotonic RM-algorithm can be self-simulated optimally by the Algorithm SIMPLE.*

Proof. Let us consider the sequences of pairs, $S_+ = CSEQ(0)+CSEQ(1)+\cdots+CSEQ(\frac{N}{Q}-1)$ and $S_- = \overline{CSEQ(0)}+\overline{CSEQ(1)}+\cdots+\overline{CSEQ(\frac{N}{Q}-1)}$. Let $S = S_+ + S_-$. Let A be any arbitrary monotonic RM-algorithm written for the simulated mesh. We now show that the sequence $S+\overline{S} = S_+ + S_- + \overline{S_-} + \overline{S_+}$ of length $4\frac{M}{P}\frac{N}{Q}$ which is $\Theta\left(\frac{M}{P}\frac{N}{Q}\right)$ can be used in the step 1.2 of Algorithm SIMPLE for each step of A to achieve a correct self-simulation.

First consider only the positive monotonic buses. Let any arbitrary positive monotonic bus u terminate in the submeshes $R_{a,b}$ and $R_{c,d}$. Now assume $a \leq c$ and this implies $b \leq d$ as u is positive monotonic. Based on the characteristics of positive monotonic bus we can say that the trajectory of the bus through various submeshes $R_{i,j}$ follow the sequence of pairs $S_u = (a,b)$, $(a+1,b)$, ..., (k_b,b), $(k_b,b+1)$, $(k_b+1,b+1)$, ..., $(k_{b+1},b+1)$, ..., (k_{d-1},d), $(k_{d-1}+1,d)$, ..., (c,d) where $0 \leq k_b \leq c$ and $k_{l-1} \leq k_l \leq c$, $b < l < d$.

It is very easy to show that S_u and $\overline{S_u}$ are contained in S_+ and $\overline{S_+}$ respectively preserving the order.

Suppose, the processor that writes on the bus u resides on the submesh $R_{p,q}$, $(p,q) \in S_u$. Then after using the the sequence S_+ completely in the step 1.2 of Algorithm SIMPLE it can be claimed that the portion of the bus u residing in the submeshes $R_{i,j}$, $(i,j) \in S_u$ and the index of the pair (i,j) in S_u is greater than or equal to that of the pair (p,q), is simulated completely. Similarly after using the sequence $\overline{S_+}$ completely we can claim that the rest portion of the bus u is also completely simulated.

The sequences of pairs S_- and $\overline{S_-}$ play similar role in simulating negative monotonic buses correctly.

Now the generation of the sequence of pairs $S+\overline{S}$ is independent of any step of RM-algorithm A. So the order of step 1.2 of Algorithm SIMPLE, $O_{1.2}$ can be considered as $O(1)$ and thus by Lemma 1 the slowdown of Algorithm SIMPLE in self-simulating any monotonic RM-algorithm is $\Theta\left(\frac{M}{P}\frac{N}{Q}\right)$ which is optimal. □

Theorem 2 *Any piecewise-monotonic RM-algorithm can be self-simulated by the Algorithm SIMPLE.*

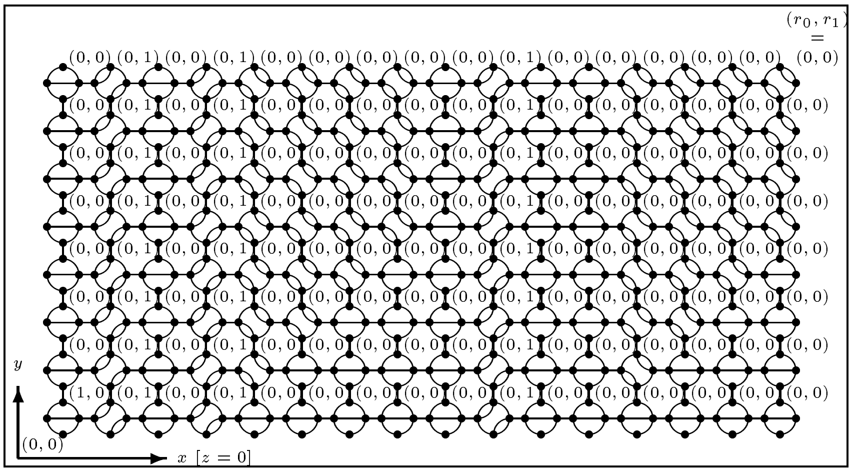

Figure 4: A step in the algorithm [6] of adding k integers of n bits each on an RM of size $2n \times 2nk$ where $\Delta = n$ (*figure generated by the serial simulator RMSIM* [10]).

Proof. Let A be any arbitrary piecewise-monotonic RM-algorithm written for the simulated mesh. Without any loss of generality we assume that the buses of any particular step of A be piecewise-monotonic w.r.t. column index.

Let Δ_u^+ denote the minimum of the minimum PE distance along the row axis of any two successive positive monotonic segments of bus u. Similarly let Δ_u^- denote the minimum of the minimum PE distance along the row axis of any two successive negative monotonic segments of bus u. Let $\Delta = \min_{\forall u}(\min(\Delta_u^+, \Delta_u^-))$ and

$$K = \begin{cases} 1 + \left(2\left\lceil \frac{Q}{\Delta} \right\rceil + 1\right) & \text{if } \Delta \text{ can be computed} \\ 3 & \text{otherwise.} \end{cases}$$

An example of Δ is given in Figure 4.

Now consider the sequence of pairs

$$S = \begin{cases} \sum_{j=0}^{\frac{N}{Q}} \left(CSEQ(j) + \overline{CSEQ(j)}\right)^{K \text{ div } 2} & \text{if } K > 3 \\ \sum_{j=0}^{\frac{N}{Q}} \left(CSEQ(j) + \overline{CSEQ(j)} + CSEQ(j)\right) & \text{otherwise.} \end{cases}$$

We now show that the sequence $S + \overline{S}$ of length $2K\frac{M}{P}\frac{N}{Q}$ can be used in the step 1.2 of Algorithm SIMPLE for each step of A to achieve a correct self-simulation.

Let any arbitrary piecewise-monotonic bus u terminates in the submeshes $R_{a,b}$ and $R_{c,d}$. Now assume $a \leq c$. Let the trajectory of the bus through various submeshes $R_{i,j}$ follow the sequence of pairs S_u. As this trajectory S_u passes through any submesh $R_{i,j}$ at most $K - 1$ times it is easy to show that S_u and $\overline{S_u}$ are contained in S and $\overline{S}$ respectively preserving the order.

Suppose, the processor that writes on the bus u resides on the submesh $R_{p,q}$, $(p, q) \in S_u$. Then after using the the sequence S completely in step 1.2 of Algorithm SIMPLE it can be claimed that the portion of the bus u residing in the submeshes $R_{i,j}$, $(i, j) \in S$ and the index of the pair (i, j) in S_u is greater than or equal to that of the pair (p, q), is simulated completely. Similarly after using the sequence $\overline{S}$ completely we can claim that the rest portion of the bus u is also completely simulated. $\square$

In Theorem 2 nothing is stated about the slowdown of the self-simulation. In general cases the slowdown will not be optimal. However we can achieve optimal slowdown for instances where the following conditions are met:

1. For each step of piecewise-monotonic RM-algorithm A, Δ is known *a-priori*.
2. $\frac{N}{Q}$ is large, a desirable property in self-simulation, so that K can be considered as a constant.

4 Conclusion

In this paper we have presented a self-simulation algorithm SIMPLE for monotonic and piecewise-monotonic RM model. Through Algorithm SIMPLE we have achieved optimal self-simulation for monotonic RM model and asymptotically optimal self-simulation for piecewise monotonic RM model. We believe that Algorithm SIMPLE preserves the essence of configurational computation and uses less broadcasts than the algorithms in [1].

References

[1] Yosi Ben Asher, Dan Gordon, and Assaf Schuster. Efficient self-simulation algorithms for reconfigurable arrays. *Journal of Parallel and Distributed Computing*, 30:1–22, 1995.

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

[3] Gen-Huey Chen, Biing-Feng Wang, and Hungwen Li. Deriving algorithms on reconfigurable networks based on function decomposition. *Theoretical Computer Science*, 120:215–227, 1993.

[4] Ju-Wook Jang, Heonchul Park, and Viktor K. Prasanna. A fast algorithm for computing a histogram on reconfigurable mesh. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17:97–106, 1995.

[5] Ju-Wook Jang, Heonchul Park, and Viktor K. Prasanna. An optimal multiplication algorithm on reconfigurable mesh. *IEEE Transactions on Parallel and Distributed Systems*, 8:521–532, 1997.

[6] Ju-Wook Jang and Viktor K. Prasanna. An optimal sorting algorithm on reconfigurable mesh. *Journal of Parallel and Distributed Computing*, 25:31–41, 1995.

[7] Philip D. Mackenzie. A separation between reconfigurable mesh models. *Parallel Processing Letters*, 5:15–22, 1995.

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

[9] 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.

[10] M. Manzur Murshed and Richard P. Brent. RMSIM: a serial simulator for reconfigurable mesh parallel computers. Technical Report TR-CS-97-06, Joint Computer Science Tech. Report Series, The Australian National University, April 1997.

[11] Koji Nakano. Prefix-sums algorithms on reconfigurable meshes. *Parallel Processing Letters*, 5:23–35, 1995.

[12] Madhusudan Nigam and Sartaj Sahni. Sorting n numbers on $n \times n$ reconfigurable meshes with buses. *Journal of Parallel and Distributed Computing*, 23:37–48, 1994.

[13] Stephan Olariu and James L. Schwing. A novel deterministic sampling scheme with applications to broadcast-efficient sorting on the reconfigurable mesh. *Journal of Parallel and Distributed Computing*, 32:215–222, 1996.

[14] Stephan Olariu, James L. Schwing, and Jingyuan Zhang. On the power of two-dimensional processor arrays with reconfigurable bus systems. *Parallel Processing Letters*, 1:29–34, 1991.

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

[16] B. F. Wang. *Configurational computation: A new algorithm design strategy on processor arrays with reconfigurable bus systems.* PhD thesis, National Taiwan University, 1991.

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