Improved Scaling Simulation of the General Reconfigurable Mesh

José Alberto Fernández-Zepeda, Ramachandran Vaidyanathan, and Jerry L. Trahan

Department of Electrical & Computer Engineering
Louisiana State University, Baton Rouge, LA 70803, USA

Abstract. The reconfigurable mesh (R-Mesh) has drawn much interest in recent years, due in part to its ability to admit extremely fast algorithms for a large number of problems. For these algorithms to be useful in practice, the R-Mesh must be scalable; that is, any algorithm designed for a large R-Mesh should be able to run on a smaller R-Mesh without significant loss of efficiency. This amounts to designing a "scaling simulation" that simulates an arbitrary step of an $N \times N$ R-Mesh on a smaller $P \times P$ R-Mesh in $O\left(\frac{N^2}{P^2} f(N,P)\right)$ steps; $f(N,P)$ is a non-decreasing function representing the *simulation overhead*. The aim is to minimize this overhead, ideally to a constant.
In this paper, we present a scaling simulation for the general (unconstrained) R-Mesh. This simulation has an overhead of $\log N$ (smaller than the $\log P \log \frac{N}{P}$ overhead of the previous fastest scaling simulation), using a CREW LRN-Mesh (a weaker version of the General R-Mesh) as the simulating model; prior simulations needed concurrent write.

1 Introduction

Reconfigurable bus-based models use dynamically alterable connections between processors to create various, changing, bus configurations. This allows efficient communication, and further, allows faster computation than on conventional "non-reconfigurable" models. This paper deals with the ability of a reconfigurable mesh (R-Mesh) [3, 6, 7, 10] to adapt an algorithm instance of an arbitrary size to run on a given smaller model size without significant loss of efficiency. This ability holds particular significance for flexibility in algorithm design and for running algorithms with various input sizes (designed for an arbitrary sized model) on an available model of given size. More formally, for any $P < N$, let a $P \times P$ R-Mesh simulate a step of an $N \times N$ R-Mesh, in $O\left(\frac{N^2}{P^2} f(N,P)\right)$ time, where $f(N,P)$ is a non-decreasing function that represents the *simulation overhead*. If the simulation overhead, $f(N,P)$, is a constant, then the model is said to have an *optimal scaling simulation*. On the other hand, if $f(N,P)$ depends only on P and is independent of N, then the model is said to have a *strong scaling simulation* [5]. The goal of designing a scaling simulation is to reduce

$f(N,P)$, ideally to a constant. Throughout this paper we assume the simulated (resp., simulating) R-Mesh to be of size $N \times N$ (resp., $P \times P$).

Reconfigurable models possess a large body of fast algorithms, yet only a handful of results exist for scaling these models. Previously, Maresca [8] established that the Polymorphic Processor Array (PPA) has an optimal scaling simulation. The PPA restricts the pattern of buses that can be created, severely curtailing the power of the model [14]. Ben-Asher *et al.* [1] proved that the LRN-Mesh (a restriction of the R-Mesh) also has an optimal scaling simulation. The LRN-Mesh admits only certain patterns of buses, making it unsuitable for fundamental problems such as graph connectivity [2]. Trahan and Vaidyanathan [13] have shown that, for certain restrictions of local connections, the RMBM (a reconfigurable bus-based model) has a strong scaling simulation. Recently, the authors [4, 5] have proved that the FR-Mesh (another restriction of the R-Mesh) has a strong scaling simulation with a simulation overhead of $\log P$. These results impose restrictions on the model scaled (though, allowing increase in number of processors, the FR-Mesh is as "powerful" as the R-Mesh [12]).

The best results, so far, on scaling the general (unrestricted) R-Mesh are due to the authors [5] and Matias and Schuster [9]. We developed a scaling simulation for the unrestricted R-Mesh with a simulation overhead of $\log P \log \frac{N}{P}$. Matias and Schuster [9] designed a randomized scaling algorithm for the unrestricted R-Mesh simulated on the LRN-Mesh. This method has a constant (with high probability) simulation overhead, when $P \leq \frac{N}{\log N \log\log N}$; the simulating LRN-Mesh, however, uses the ARBITRARY rule to resolve concurrent writes, a rule not easily implementable on a bus.

In this paper, we construct a deterministic scaling simulation for the unrestricted R-Mesh, which improves on the best previous simulation overhead of $\log P \log \frac{N}{P}$. The key component of this scaling simulation is a novel method to convert a general R-Mesh algorithm to run on an LRN-Mesh of the same size. The conversion identifies structures that comprise a spanning forest of reduced bus configurations, then emulates these with linear buses following Euler tours of the trees. This approach has a $\log N$ simulation overhead. Further, the simulating LRN-Mesh uses only exclusive writes, whereas prior scaling simulations needed concurrent writes. We next extend the algorithm to a lower-overhead simulation of an FR-Mesh and to a simulation by a model using pipelined optical buses (PR-Mesh). Table 1 summarizes the results of this paper.

Table 1. Summary of results for the general R-Mesh and FR-Mesh scaling simulations

$N \times N$ Simulated Model	$P \times P$ Simulating Model	Simulation overhead
CRCW R-Mesh	CREW LRN-Mesh or PR-Mesh	$\log N$
CRCW FR-Mesh	CREW LRN-Mesh or PR-Mesh	$\log P$

Section 2 describes the R-Mesh and some of its versions. Sections 3.1 and 3.2 describe the scaling simulations of an R-Mesh and FR-Mesh using a CREW LRN-Mesh. Section 3.3 presents scaling simulations of an R-Mesh and FR-Mesh using an optical model. Finally, Section 4 identifies some open problems.

2 The Reconfigurable Mesh

An $R \times C$ *Reconfigurable Mesh* (*R-Mesh*) is a two-dimensional array of processors connected in an $R \times C$ grid. Each processor in the R-Mesh has direct "external connections" to adjacent processors through a set of four input/output ports. A processor can internally partition its set of ports so that ports in the same block of a partition are fused. These partitions, along with external connections between processors, define a global bus structure that connects the ports of processors. All ports that are part of the same bus are said to be in the same *component* of the R-Mesh. Figure 1 shows a component in bold. Also, Fig. 1 shows a 3×5 R-Mesh, depicting the fifteen possible port partitions of a processor. The R-Mesh is a synchronous model that may change its bus configurations at each step. It also assumes negligible delay on buses [7, 8, 10].

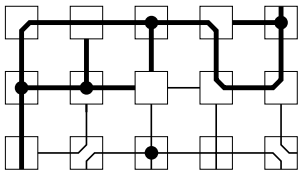

Fig. 1. Port partitions of an R-Mesh. Processor (0,0) is placed on the upper left corner.

The *LRN-Mesh* is a restricted version of the R-Mesh that allows processors to fuse only pairs of ports together or leave ports unfused. (That is, the LRN-Mesh does not permit the connections in processors (0,2), (0,4), (1,0), (1,1), and (2,2) of Fig. 1.) The *FR-Mesh* is another restriction of the R-Mesh that permits only the "fusing" and "cross-over" connections shown in processors (2,2) and (2,3), respectively, of Fig. 1. A component is *linear* iff it connects its ports only as allowed by the LRN-Mesh; otherwise the component is *non-linear*.

In most of this paper, we assume the concurrent read concurrent write (CRCW) model. Let the term *reading* (*writing*) *port* to refer to a port through which a processor is reading (writing). The R-Mesh will resolve concurrent writes by any of the following rules: COMMON (where all writing ports of a component must write the same value), COLLISION (where concurrent writes result in a distinguished collision symbol being written on the component), or COLLISION$^+$ (that behaves like the COMMON rule when all processors attempt to write the same value on a component, and like COLLISION otherwise). In this paper, we will also discuss the PRIORITY rule (in which the lowest indexed writing port

of a component succeeds in writing) and the ARBITRARY rule (in which an arbitrary port succeeds in writing). PRIORITY and ARBITRARY rules are difficult to implement and will be used only as algorithmic tools. Regardless of the write rule used, the final value on the bus is called the *bus data*.

3 Scaling Simulation via LRN-Mesh

We now design a scaling simulation of the R-Mesh by an LRN-Mesh, where the sizes of both machines have the same order, and then (optimally) scale down the simulating LRN-Mesh. We will later refine this result so that the simulating LRN-Mesh uses only exclusive writes. The simulation allows the powerful and flexible algorithm design permitted by the CRCW model, while using the more feasible bus implementation of the CREW model.

3.1 Simulation Description

We will describe the simulation through the following phases:
1. Simulation of an $N \times N$ CRCW R-Mesh on a $2N \times 2N$ CRCW LRN-Mesh
2. Simulation of an $N \times N$ CRCW R-Mesh on a $P \times P$ CRCW LRN-Mesh
3. Simulation of an $N \times N$ CRCW R-Mesh on a $P \times P$ **CREW** LRN-Mesh

Phase 1. This phase establishes the following result.

> • For any $P < N$, any step of an $N \times N$ COMMON, COLLISION, COLLISION$^+$, or PRIORITY CRCW R-Mesh can be simulated on a $2N \times 2N$ COMMON, COLLISION, COLLISION$^+$, or PRIORITY CRCW LRN-Mesh in $O(\log N)$ time.

We first describe the result for the COMMON write rule, then include the other write rules. We now construct a simulation of an $N \times N$ COMMON CRCW R-Mesh, $\mathcal{Q}$, on a $2N \times 2N$ COMMON CRCW LRN-Mesh, $\mathcal{Z}$, in $O(\log N)$ time.

A group of four processors of $\mathcal{Z}$ simulates a processor of $\mathcal{Q}$, where Fig. 2 shows selected configurations of a processor of $\mathcal{Q}$ and the corresponding configurations assumed by a group of $\mathcal{Z}$. Assign to each bus of $\mathcal{Z}$ a direction (in or out) as shown in Fig. 2; this assignment is mainly for ease of explanation and does not require the "more powerful" directional model [2]. Allow each processor in the group to write only to outgoing buses and read only from incoming buses.

Since an LRN-Mesh can simulate linear components directly, we focus our attention on non-linear components. Let the *non-linear graph* of $\mathcal{Q}$ be a graph in which, for each nonlinear component, the processors are nodes and the links between them are edges. The idea in this simulation is to iteratively grow a spanning tree for each component in the non-linear graph of $\mathcal{Q}$. During the spanning tree construction, we simultaneously construct an Euler tour of each tree. The Euler tour enables the LRN-Mesh to handle each tree as a linear bus. For clarity, we describe the algorithm acting on only one component on the non-linear graph.

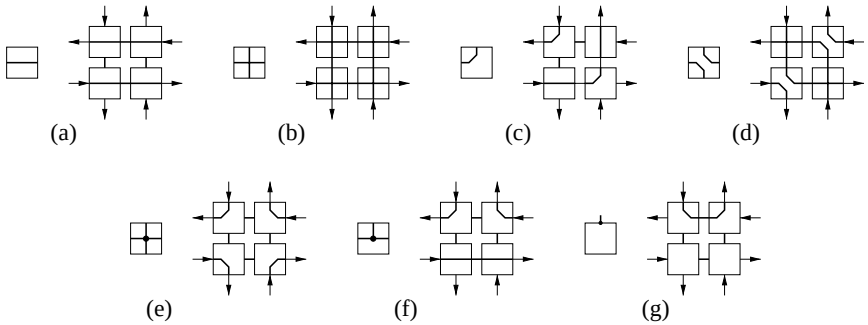

Fig. 2. Group configurations for processors with special types of connections: a-d) linear; e,f) non-linear; g) end-point bus.

We now describe the simulation. For brevity, we omit details.

Step 1 - Bus contraction: This step partially contracts the non-linear graph. It removes each chain that connects a leaf (degree 1 node) to an internal node (degree 3 or 4 node) to construct a *raked graph* (Fig. 3). Before removing these chains, $\mathcal{Z}$ simulates each corresponding bus segment independently and obtains their resulting bus data; these values will be used later during Step 7.

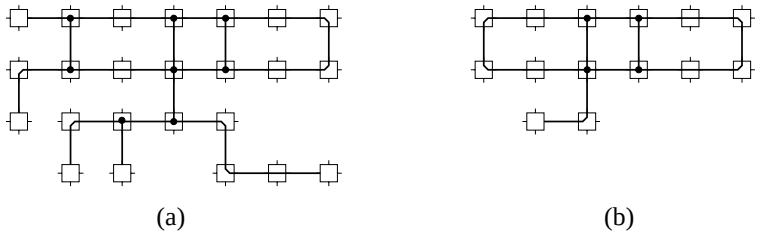

Fig. 3. a) Initial non-linear graph of the component (before the bus contraction of Step 1); b) raked graph (after Step 1).

Step 2 - Initiating spanning tree construction: This step further reduces the raked graph to a *distilled graph* and starts constructing a spanning tree. Let each internal node (degree 3 or 4) and leaf (degree 1) of the raked graph be a node of the distilled graph. Let a chain of degree-2 nodes in the raked graph be an edge in the distilled graph. Denote the corresponding groups of processors in $\mathcal{Z}$ as node groups and edge groups, respectively. Each node group chooses its neighboring node group with smallest index as its parent. Add the edge between each parent and child into a forest (that eventually will be a spanning tree). Denote as a *root group* a group with index smaller than any of its neighbors.

Step 3 - Initial Euler tour construction: This step constructs an Euler tour in each tree of the forest obtained in Step 2 using the configurations shown in

Fig. 2a-d for edge groups and Fig. 2e-g for node groups. The root group of each tree acts as a leader and broadcasts its index to all the node groups in its tree.

Repeat Steps 4 and 5 $2 \log N$ times to complete a spanning forest of the distilled graph.

Step 4 - Grafting trees: This step combines trees in the forest by a *graft operation*, in which each tree chooses a neighboring tree with a smaller label and grafts onto it. This operation basically changes the internal connections of a node group in each of the two trees to incorporate the edge between them into the Euler tour.

Step 5 - Grafting rejected trees: It is possible that a tree has not been subject to a graft operation in Step 4, despite having a neighbor in another tree via an unselected edge. Force each such tree to graft onto some neighboring tree.

Step 6 - Adding unselected edges to the spanning tree: This step extends the spanning tree of the distilled graph to a spanning tree of the raked graph (Fig. 4).

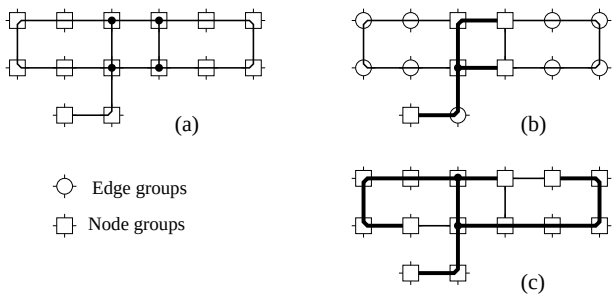

Fig. 4. a) Raked graph (after Step 1); b) Spanning tree of the distilled graph (before Step 6); c) Spanning tree of the raked graph (after Step 6).

Step 7 - Simulating bus communication: This step simulates the communication through the Euler tours of the spanning tree obtained in Step 6. Since an Euler tour connects all the ports of the same component to the same bus, the writer processors of that component can simulate the write rule of $\mathcal{Q}$. Finally, broadcast the resulting bus data to the bus segments contracted in Step 1. This step completes the simulation of the general R-Mesh by the LRN-Mesh. ■

Example: Figure 4a shows a raked graph after the contraction step (Step 1). The bold bus in Fig. 4b represents the spanning tree of the distilled graph generated after the iterative process of Steps 4 and 5. Notice that all the node groups are part of the spanning tree, but not all of the edge groups. Figure 4c shows the spanning tree of the raked graph after Step 6.

All of the above steps run in constant time. The running time of the algorithm is due to the iterative process on Steps 4 and 5. After Step 3, it is possible to have $O(N^2)$ trees in a component. By grafting these trees in Steps 4 and 5, the number of trees reduces by at least half, so the algorithm runs in $O(\log N)$ iterations to complete the construction of the spanning tree. This algorithm readily adapts to the COLLISION and COLLISION$^+$ rules. When $\mathcal{Q}$ uses the PRIORITY rule, solve concurrent writes in Steps 1 and 7 using priority resolution in $O(\log N)$ time.

Phase 2. This phase uses the scaling simulation of Ben-Asher *et al.* [1] to scale the $N \times N$ COLLISION$^+$ CRCW LRN-Mesh down to a $P \times P$ COLLISION$^+$ CRCW LRN-Mesh. By extending this result to other write rules, we obtain the following.

> • For any $P < N$, any step of an $N \times N$ COMMON, COLLISION, COLLISION$^+$, or PRIORITY CRCW R-Mesh can be simulated on a $P \times P$ COMMON, COLLISION, COLLISION$^+$, or PRIORITY CRCW LRN-Mesh in $O\left(\frac{N^2}{P^2} \log N\right)$ time.

Phase 3. This phase simplifies the simulating machine $\mathcal{R}$ to use only exclusive writes. Trahan *et al.* [14] proved for the RMBM reconfigurable model that the CREW version can simulate the CRCW version (for the write rules considered here) in constant time, utilizing the ability to perform neighbor localization (a procedure that finds the nearest marked processor to a reference processor). We extend this idea to the simulation via LRN-Mesh expressed in Phase 2. It is easy to prove that a CREW linear bus that can be segmented can simulate a COMMON, COLLISION, or COLLISION$^+$ CRCW linear bus in constant time by applying neighbor localization.

Since a $P \times P$ **CREW** LRN-Mesh simulates a $P \times P$ COMMON, COLLISION, or COLLISION$^+$ LRN-Mesh in constant time, then by using the result of Phase 2 we obtain the following theorem.

Theorem 1. *For any $P < N$, any step of an $N \times N$* COMMON, COLLISION, COLLISION$^+$, *or* PRIORITY *CRCW R-Mesh can be simulated on a $P \times P$* **CREW** *LRN-Mesh in $O\left(\frac{N^2}{P^2} \log N\right)$ time.* ■

Although Matias and Schuster [9] also simulated the general R-Mesh via the LRN-Mesh, their simulation is randomized and quite different from the one proposed in this paper. Their simulation computes connected components in two stages. In the first stage, they obtained the connected components in each $P \times P$ sized window of $\mathcal{Q}$. In the second stage, they completed the process, calculating the connected components on a graph with $O\left(\frac{N^2}{P}\right)$ nodes and $O\left(\frac{N^2}{P}\right)$ edges. The connected components algorithm in the second stage is the LRN-Mesh simulation of a randomized PRAM algorithm. On the other hand, our connected components algorithm (spanning forest construction) is deterministic and the input is a graph with $O(N^2)$ nodes and $O(N^2)$ edges. Another important difference is that, to attain the stated overhead in the simulation of Matias and

Schuster, the write rule for the simulating machine must be ARBITRARY, which is difficult to implement in a bus. Our simulation uses only exclusive writes.

3.2 Scaling Simulation of FR-Mesh using LRN-Mesh

The authors [5] previously developed a scaling simulation for the FR-Mesh. By applying Theorem 1 in some portions of this FR-Mesh scaling simulation, an LRN-Mesh can simulate an FR-Mesh with simulation overhead of $O(\log P)$.

Corollary 2. *For any $P < N$, any step of an $N \times N$* COMMON, COLLISION, COLLISION$^+$, *or* PRIORITY *CRCW FR-Mesh can be simulated on a $P \times P$* **CREW** *LRN-Mesh in $O\left(\frac{N^2}{P^2} \log P\right)$ time.*

3.3 Simulation of R-Mesh by PR-Mesh

The Pipelined Reconfigurable Mesh or PR-Mesh [12] is a special type of reconfigurable mesh that uses optical buses. It can configure its port connections to form linear buses as the LRN-Mesh can. Each optical bus can perform several one-to-one communications in constant time by using pipelining. An optical bus consists of two bus segments (upper and lower) connected at one of the ends, forming a directional U-shaped bus. Processors use the upper bus to write and the lower bus to read.

A $P \times P$ PR-Mesh can simulate each step of a $P \times P$ CREW LRN-Mesh with no cycles in constant time as follows. Scale the LRN-Mesh down to a $\frac{P}{2} \times \frac{P}{2}$ LRN-Mesh. Then, replicate the connections of this new LRN-Mesh on the PR-Mesh, creating a double bus structure to decide which of the two end-processors connects the upper and lower segments. Finally, use only one of these two buses to broadcast the written value. (This simulation corresponds to those in earlier papers [11, 12], though they did not explicitly address the leader election problem.) If the LRN-Mesh has cycles, the PR-Mesh can find a leader within the cycle in an additional $O(\log P)$ time, then break the cycle, and proceed as if the LRN-Mesh was acyclic.

Corollary 3. *For any $P < N$, any step of an $N \times N$* COMMON, COLLISION, COLLISION$^+$, *or* PRIORITY *CRCW R-Mesh can be simulated on a $P \times P$ PR-Mesh in $O\left(\frac{N^2}{P^2} \log N\right)$ time.*

Corollary 4. *For any $P < N$, any step of an $N \times N$* COMMON, COLLISION, COLLISION$^+$, *or* PRIORITY *CRCW FR-Mesh can be simulated on a $P \times P$ PR-Mesh in $O\left(\frac{N^2}{P^2} \log P\right)$ time.*

Remark: The $O(\log P)$ time to break cycles of the LRN-Mesh is an additive overhead in the simulation overhead of Corollaries 3 and 4 because the PR-Mesh performs this operation only once per each of the $\left(\frac{N^2}{P^2}\right)$ windows.

4 Open Questions

Open problems include investigating whether or not the General R-Mesh has an optimal scaling simulation or at least a strong scaling simulation. Another research direction is to find configurations (other than LRN-Mesh and FR-Mesh) that are useful computationally and have optimal or strong scaling simulations. Finally, is priority resolution in $o(\log P)$ time possible on a $P \times P$ FR-Mesh using COMMON or COLLISION rules? The answer to this could immediately improve simulation overheads for the R-Mesh and FR-Mesh.

Acknowledgments. This material is based upon work supported in part by the National Science Foundation under Grant No. CCR-9503882.

References

1. Ben-Asher, Y., Gordon, D., Schuster, A.: Efficient Self Simulation Algorithms for Reconfigurable Arrays. J. Parallel Distrib. Comput. **30** (1995) 1–22
2. Ben-Asher, Y., Lange, K.-J., Peleg, D., Schuster, A.: The Complexity of Reconfiguring Network Models. Info. and Comput. **121** (1995) 41–58
3. ElGindy, H., Wetherall, L.: A Simple Voronoi Diagram Algorithm for a Reconfigurable Mesh. IEEE Trans. Parallel Distrib. Systems **8** (1997) 1133–1142
4. Fernández-Zepeda, J.A., Trahan, J.L., Vaidyanathan, R.: Scaling the FR-Mesh under Different Concurrent Write Rules. World Multiconf. on Systemics, Cybernetics, and Informatics (1997) 437–444
5. Fernández-Zepeda, J.A., Vaidyanathan, R., Trahan, J.L.: Scaling Simulation of the Fusing-Restricted Reconfigurable Mesh. IEEE Trans. Parallel Distrib. Systems **9** (1998) 861–871
6. Jang, J.-w., Nigam, M., Prasanna, V. K., Sahni, S.: Constant Time Algorithms for Computational Geometry on the Reconfigurable Mesh. IEEE Trans. Parallel Distrib. Systems **8** (1997) 1–12
7. Li, H., Stout, Q. F.: Reconfigurable SIMD Massively Parallel Computers. IEEE Proceedings **79** (1991) 429–443
8. Maresca, M.: Polymorphic Processor Arrays. IEEE Trans. Parallel Distrib. Systems **4** (1993) 490–506
9. Matias, Y., Schuster, A.: Fast, Efficient Mutual and Self Simulations for Shared Memory and Reconfigurable Mesh. 7th IEEE Symp. Par. Distrib. Processing (1995) 238–246
10. Miller, R., Prasanna-Kumar, V. K., Reisis, D., Stout, Q.: Parallel Computations on Reconfigurable Meshes. IEEE Trans. Comput. **42** (1993) 678–692
11. Pavel, S., Akl, S. G.: On the Power of Arrays with Optical Pipelined Buses. Int'l. Conf. Par. Distr. Proc. Techniques and Appl. (1996) 1443–1454
12. Trahan, J. L., Bourgeois, A. G., Vaidyanathan, R.: Tighter and Broader Complexity Results for Reconfigurable Models. Parallel Processing Letters, special issue on Bus-based Architectures **8** (1998) 271–282
13. Trahan, J. L., Vaidyanathan, R.: Relative Scalability of the Reconfigurable Multiple Bus Machine. Proc. Workshop Reconfigurable Arch. and Algs. (1996)
14. Trahan, J. L., Vaidyanathan, R., Thiruchelvan, R. K.: On the Power of Segmenting and Fusing Buses. J. Parallel Distrib. Comput. **34** (1996) 82–94