Constant Time Simulation of an R-Mesh on an LR-Mesh

Carlos Alberto Córdova-Flores[1], José Alberto Fernández-Zepeda[1], and Anu G. Bourgeois[2]

[1]CICESE
Department of Computer Science
Km. 107 Carretera Tijuana-Ensenada
Ensenada, B.C. 22860, Mexico
{cordovaf, fernan}@cicese.mx

[2]Georgia State University
Department of Computer Science
Atlanta, GA 30303, USA
abourgeois@cs.gsu.edu

Abstract

Recently, many parallel computing models using dynamically reconfigurable electrical buses have been proposed in the literature. The underlying characteristics are similar among these models, but they do have certain differences that can take form of restrictions on configurations allowed. This paper presents a constant time simulation of an R-Mesh on an LR-Mesh (a restricted model of the R-Mesh), proving that in spite of the differences, the two models possess the same complexity. In other words, the LR-Mesh can simulate a step of the R-Mesh in constant time with a polynomial increase in size. This simulation is based on Reingold's algorithm to solve USTCON in log-space. The simulation is also the first to be executed in constant time.

1. Introduction

Reconfigurable bus architectures have the ability to create different interconnection topologies allowing processors to exchange information efficiently. These architectures use their buses as a computational resource, allowing the execution of fast parallel algorithms. Reconfigurable models are computationally more powerful than non-reconfigurable models like the Parallel Random Access Machine (PRAM) [4]. Nevertheless, there is skepticism with respect to some of the assumptions of these models, such as the propagation delay in buses, which is assumed to be constant [7].

Researchers have proposed a number of reconfigurable bus-based models, such as the Reconfigurable Mesh (R-Mesh) [7, 10], the Reconfigurable Network (RN) [1], the Polymorphic Processors Array (PPA) [8], Reconfigurable Multiple Bus Machine (RMBM) [13]. Among these models, the R-Mesh is possibly the most widely studied model. Researchers have also designed a great number of fast algorithms for these models. Vaidyanathan and Trahan [14] wrote a comprehensive book on dynamically reconfigurable computer architectures and algorithms.

An R-Mesh is a two-dimensional array of processors connected in a grid such that each processor has fixed connections with its four neighbors through its N, S, E, and W ports. In addition, each processor has the capacity to select one of fifteen internal connections between its ports at the beginning of each machine cycle. The squares in Figure 1 represent processors with the fifteen possible internal connections of an R-Mesh. Each circle represents an input/output port.

The Linear Reconfigurable Mesh (LR-Mesh) is a restricted version of the R-Mesh. Each port in the LR-Mesh can connect to at most one other port in the same processor, so the LR-Mesh allows ten of the fifteen connections of the R-Mesh. The type of buses allowed by an LR-Mesh are called linear buses. Figures 1a to 1j show the ten possible connections for the LR-Mesh. The LR-Mesh and the R-Mesh use undirected buses.

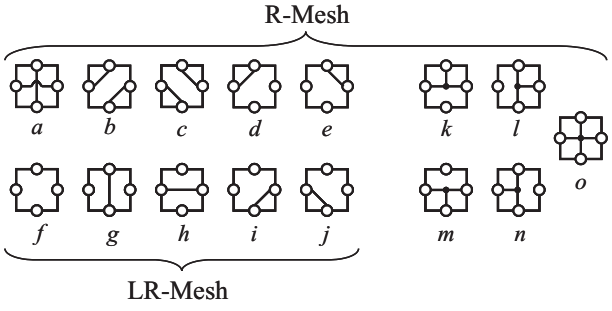

Figure 1. The possible internal connections of the R-Mesh and LR-Mesh.

1-4244-0910-1/07/$20.00 ©2007 IEEE.

The class of languages accepted by an LR-Mesh (R-Mesh) in constant time with polynomial number of processors is equivalent to the class L (SL) of languages accepted in deterministic logarithmic space (symmetric logarithmic space) on a Turing machine [11]. For many years it was widely conjectured that $L \subset SL$, but in 2005, Reingold [12] proved that $L = SL$. The result of Reingold implies that the LR-Mesh is computationally as powerful as the R-Mesh and that the LR-Mesh can simulate an R-Mesh in constant time, allowing the LR-Mesh a polynomial increase in the number of processors. However, the amount of resources required by the LR-Mesh to perform this simulation has not been clear.

Fernández-Zepeda *et al.* [2] demonstrated, using the bus linearization technique, that an LR-Mesh of $N \times N$ processors can simulate an R-Mesh of the same size in $O(\log N)$ time. This simulation is optimal in the number of processors, and before this paper, it was the fastest simulation of this type. Until now, it was remained a challenge to reduce the time required for the simulation.

In this paper, we present an algorithm to perform this simulation in constant time. The drawback of our simulation is that the number of processors in the simulating model is extremely high. However, at this time, ours is the only simulation available of an R-Mesh on an LR-Mesh in constant time. By establishing the equivalence between the two models, we can now easily map an algorithm designed for the R-Mesh to an LR-Mesh.

We present the simulation in three phases, with each one running in constant time on the LR-Mesh.

1. *Transform G into an expander.* Given the rotation map of the R-Mesh graph G, the LR-Mesh algorithm transforms this graph into an expander G_γ. The output of this step is the rotation map of G_γ. We use the algorithm of Reingold [12] to accomplish this step.

2. *Solve USTCON.* Using the rotation map of G_γ, we solve the undirected $s - t$ connectivity in G_γ. We employ this solution to create the transitive closure graph G^* of G and its adjacency matrix A^*.

3. *Handle non-linear buses as linear buses.* We use the information of A^* to handle non-linear buses accepted by the R-Mesh, as linear buses accepted by the LR-Mesh. By performing this transformation, the R-Mesh on LR-Mesh simulation is straightforward.

We organize the remaining sections of this paper as follows. Section 2 provides some basic concepts and definitions. Section 3 describes the transformation of a graph G into an expander G_γ. Section 4 details the algorithm to solve USTCON in the expander. Section 5 presents the simulation of the R-Mesh on the LR-Mesh. Finally, Section 6 provides some concluding remarks.

2. Preliminaries

This section explains basic notation and some concepts related with Reingold's algorithm [12]. A graph is x-regular if the sum of all entries in each row of its adjacency matrix is x. One useful form to represent a graph is the rotation map. Let $G = (V, E)$ be an x-regular graph, for some x, and assume that each vertex assigns sequential labels to all the edges that are incident to it. Given two arbitrary vertices $u, v \in V$ that are connected through edge $e \in E$, if e is both, the i^{th} edge of u and the j^{th} edge of v, then the rotation map of the pair (u, i) is the pair (v, j) and is denoted by $\mathrm{Rot}_G(u, i) = (v, j)$. We use the term Rot_G when we refer to the rotation maps of all possible pairs in G.

Given an undirected graph $G = (V, E)$ and two vertices $s, t \in V$, the undirected $s - t$ connectivity problem (USTCON) is the process of determining whether there exists a path in G that connects s to t. Reingold [12] designed a deterministic log-space algorithm to solve USTCON (that is complete for the class SL). With this result he proved that $L = SL$. To solve USTCON, he basically designed a log-space transformation of the input graph G into an expander graph G_γ that maintains the same structure in the connected components. Since the diameter of any expander is logarithmic in the number of vertices, it is straightforward to check all possible paths of logarithmic size that leave s to determine if at least one of them reaches t. An expander, roughly speaking, is a sparse graph with high connectivity. Researchers have designed methods to generate families of expanders [3, 6, 9].

The transformation of the input graph G into an expander is the medullar part of Reingold's algorithm. This transformation requires the generation of a small expander H. Expander H is a regular graph, its degree and number of vertices depends on the degree of G. This transformation is described in Definition 1 and was obtained from [12].

Definition 1. Let G be a D^{16}-regular graph with N vertices and H a D-regular graph with D^{16} vertices, where D is a constant. Given the rotation maps of graphs G and H, the transformation $\mathcal{T}(G, H) = G_\gamma$ generates the rotation map of the expander G_γ as follows:

- The value of γ is the smallest integer that satisfies: $(1 - 1/DN^2)^{2^\gamma} < 1/2$.
- Compute G_γ recursively by the rule: $G_\gamma = (G_{\gamma-1} \textcircled{z} H)^8$, where $G_0 = G$. □

The symbol $\textcircled{z}$ represents the zig-zag product. The zig-zag product betwen G and H, $G \textcircled{z} H$, produces a graph that is D^2-regular with ND^{16} vertices. When this graph is elevated to the power eight, the number of edges incident to each vertex increases to D^{16}. The successive combination of the zig-zag product and the exponentiation operation generates a family of graphs $(G_1, G_2, \ldots, G_\gamma)$ that are

D^{16}-regular and gradually improves its connectivity properties. The expander G_γ has $ND^{16\gamma}$ vertices. The value γ is $O(\log N)$ [12]. If $\gamma = C \log N$, where C is a constant, then $ND^{16\gamma} = N^\alpha$, where $\alpha = (16\, C \log D) + 1$, which is polynomial in N.

Let $[X]$ be the set $\{0, 1, \ldots, X-1\}$. Denote the label of each vertex in the expander G_γ as $(v, a_0, \ldots, a_{\gamma-1})$, where $v \in [N]$ and $a_i \in [D^{16}]$, for $0 \leq i < \gamma$. Additionally, we can decompose each a_i in a sequence of 16 digits in base D, denoted by $k_{i,0}, \ldots, k_{i,15}$.

The pseudo-code of Algorithm 1 [12] computes the transformation $\mathcal{T}(G, H) = G_\gamma$.

Algorithm 1. Reingold's algorithm.
input: Rotation maps of G (Rot_G) and H (Rot_H)
output: Rotation map of G_γ (Rot_{G^γ})
begin
for $j = 0$ to 15 **do**
Set $(a_{\gamma-1}, k_{\gamma,j}) \leftarrow \mathrm{Rot}_H\,(a_{\gamma-1}, k_{\gamma,j})$
if j is even, recursively set
$((v, a_0, ..., a_{\gamma-2}), a_{\gamma-1}) \leftarrow \mathrm{Rot}_{G^{\gamma-1}}((v, a_0, ..., a_{\gamma-2}), a_{\gamma-1})$
if $j = 15$, reverse the order of individual labels in a_γ:
Set $k_{\gamma,0}, \ldots, k_{\gamma,15} \leftarrow k_{\gamma,15}, \ldots, k_{\gamma,0}$.
end for
end

Notice that Algorithm 2 is recursive. To compute Rot_{G^γ}, this algorithm first needs to compute $\mathrm{Rot}_{G^{\gamma-1}}$, and so on. Remember that $\mathrm{Rot}_{G_0} = \mathrm{Rot}_G$. For convenience, in Algorithm 2 we make index j to vary from 0 to 15; in [12] index j varies from 1 to 16.

3. Transformation of G into an expander

This section presents the implementation of the algorithm of Reingold [12] on an LR-Mesh. This algorithm transforms an arbitrary non-bipartite D^{16}-regular graph $G = (V_G, E_G)$, where $|V_G| = N$, into an expander G_γ. We assume that we know the rotation map of graph G and each $\mathrm{Rot}_G(u, i)$ is stored in a different processor of a row or column of processors in the LR-Mesh. The output for this transformation is the rotation map of G_γ.

Each step of Algorithm 1 can be viewed as a permutation that an LR-Mesh can compute in constant time. To implement all the steps of Algorithm 1 in the LR-Mesh, connect as many permutation stages as the number of operations in the algorithm. These types of connections in the LR-Mesh basically form a multistage interconnection network (MIN), where stage i of the MIN computes the operation that Algorithm 1 computes in time i.

The pseudo-code of Algorithm 2 (that runs in the LR-Mesh) is the parallel implementation of Algorithm 1. (Algorithm 2 includes the generation of H as a step; in Algorithm 1, H is an input). Steps 1 to 5 generate the MIN and Step 6 computes the rotation map of G_γ. The LR-Mesh executes all these steps in $O(1)$ time. Next, we briefly explain each step and discuss the resources required by the LR-Mesh.

Algorithm 2. Generate expander G_γ
input: Rotation maps of G (Rot_G)
output: Rotation map of G_γ (Rot_{G^γ})
begin
1. Generate expander H
2. Compute the number, r, of permutations in the MIN
3. Find the permutation sequence $p_0, p_1, \ldots, p_{r-1}$
4. Assign rotation maps to each vertex
5. Set the permutations in the MIN
6. Compute the rotation map of G_γ

end

Step 1. Generate H. Generate the rotation map of a D-regular expander $H = (V_H, E_H)$, where $|V_H| = D^{16}$. Since the expander H has a constant number of vertices and edges, a single processor of the LR-Mesh can generate H in constant time by using some of the methods mentioned in the literature to generate expanders. This processor also sends each $\mathrm{Rot}_H(u, i)$ to different processors in a single row or column of the LR-Mesh, so the LR-Mesh can move them easily if necessary.

Step 2. Compute r. Let r be the total number of permutations in the MIN. Let S_γ be the total number of operations to calculate the expander G_γ in Algorithm 1. Since Algorithm 2 maps each operation from Algorithm 1 to one permutation in the MIN, r is equal to S_γ. From Algorithm 1, we conclude that $S_\gamma = (S_{\gamma-1} + 2) \times 8 + 1$, where $S_0 = 1$. An LR-Mesh can compute S_γ in constant time with a network that consists of γ stages in cascade. Each stage i consists of an adder (performs $S_{i-1} + 2$), a multiplier (multiplies the previous result by eight), and another adder (adds one to the previous result), all of them also connected in cascade. This network is very similar to the network presented in [5]. The value of r is $O(N^3)$, thus it is necessary at least an LR-Mesh of $O(N^3) \times O(N^3)$ processors to compute r.

Step 3. Find $p_0, p_1, \ldots, p_{r-1}$. The goal of this step is to find the sequence of permutations $p_0, p_1, \ldots, p_{r-1}$ for the MIN that computes the operations of Algorithm 1. There are three types of permutations in the MIN, $\mathcal{H}$, $\mathcal{G}$ and $\mathcal{X}$. Permutations $\mathcal{H}$ are based on the rotation map of graph H. Permutations $\mathcal{X}$ corresponds to the last operation in each recursion level of Algorithm 1. Finally, permutations $\mathcal{G}$ are based on the rotation map of graph G. There are some subclasses for these permutations, denoted by $\mathcal{H}_s^t$, $\mathcal{X}^t$ and $\mathcal{G}^t$. The indices t and s indicate the depth of the recursion and the operation number inside the recursion, respectively, where $0 \leq t \leq \gamma$ and $0 \leq s \leq 15$.

For each permutation number x, where $0 \leq x \leq r-1$, we find its type of permutation and its indices t and s. Figure 2 shows the structure of the order that the permutations

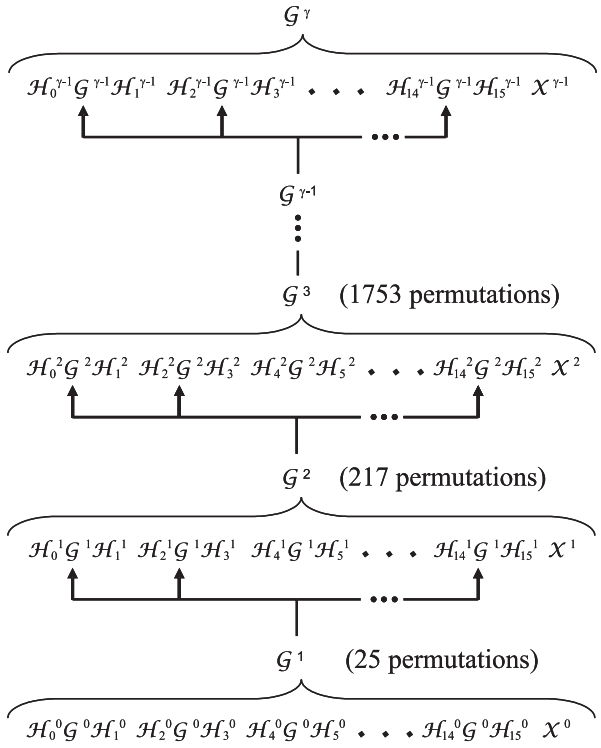

Figure 2. Structure of the order of the permutations in the MIN.

follow in the sequence. To generate the sequence of permutations $p_0, p_1, ..., p_{r-1}$, we built a label of γ digits in base 25 for each permutation in the structure of Figure 2.

For each permutation $\mathcal{H}_s^{\gamma-1}$, $\mathcal{X}^{\gamma-1}$ and $\mathcal{G}^{\gamma-1}$ (top row in the structure of Figure 2) the most significant digit of its label is its position (0 to 24) in its row; the $\gamma - 1$ remaining digits of its label are zeros. The permutations of the types $\mathcal{H}_s^{\gamma-2}$, $\mathcal{X}^{\gamma-2}$ and $\mathcal{G}^{\gamma-2}$ form eight rows (each one of 25 permutations). Each row replaces one permutation $\mathcal{G}^{\gamma-1}$. The label for each permutation is the following. The most significant digit of the label is the most significant digit of the permutation label its row is replacing. The second most significant digit of its label is its position in its row. The $\gamma - 2$ remaining digits of its label are zeros. We continue with this labeling for all permutations. This labeling is exactly the same as the one most editors use to label chapters, sections, and subsections of a book. Additionally, each permutation keeps its row position, so it knows its index s and also maintains the level of recursion t.

The algorithm stores the 25 permutation labels of each row of permutations in the 25 first processors of a row of processors. So the algorithm requires $O(N^3)$ rows of processors for this purpose. To facilitate the movement of labels, the LR-Mesh assigns 24 empty rows of processors between each row of processors with labels. So the LR-Mesh can move all the permutation labels to a single column in constant time.

To generate the sequence of permutations, we just eliminate all the permutations $\mathcal{G}^i$, except for $i = 0$, and sort the remaining permutations with respect to their labels using the constant time algorithm of Jang and Prasanna [5]. An LR-Mesh of $O(N^3) \times O(N^3)$ performs Step 2 in constant time.

Step 4. Assign rotation maps. Each stage of the MIN contains $D^{16}N^{\alpha}$ vertices. Each vertex has a unique label $(\bar{v}, \bar{a})$, where $\bar{v} = (v, a_0, a_1, \ldots, a_{\gamma-1})$ is the label of a vertex of the expander G_γ and $\bar{a} = a_\gamma$ is the label of one of its edges.

• If $p_i = \mathcal{G}^0$, then each vertex $(v, a_0, a_1, \ldots, a_\gamma)$ of stage i in the MIN stores $\mathrm{Rot}_G(v, a_0)$.

• If $p_i = \mathcal{H}_s^t$, then each vertex $(v, a_0, a_1, \ldots, a_\gamma)$ of stage i stores $\mathrm{Rot}_H(a_0, w)$, for all w. Also, each vertex $(v, a_0, a_1, \ldots, a_{t-2}, a_{t-1}, a_t, \ldots, a_\gamma)$ exchanges the value $\mathrm{Rot}_H(a_0, w)$ with the value $\mathrm{Rot}_H(a_{t-1}, w)$ of vertex $(v, a_{t-1}, a_1, \ldots, a_{t-2}, a_0, a_t, \ldots, a_\gamma)$, for all w.

• If $p_i = \mathcal{X}^t$, there is nothing to assign.

It is possible to embed the connections between two adjacent stages of the MIN on an LR-Mesh of $2D^{16}N^{\alpha} \times D^{16}N^{\alpha}$ processors. The LR-Mesh requires $O(N^{\alpha}) \times O(N^{\alpha+3})$ processors to embed the whole MIN.

Since the values of Rot_H and Rot_G are distributed in a single row or column of processors of the LR-Mesh, the LR-Mesh can broadcast each of these values to its respective destination in constant time. Notice that there are enough processors between each stage of the MIN to exchange the required data in constant time.

Step 5. Set permutations in the MIN. For all $0 \leq i < \gamma$, each vertex of stage i in the MIN (called source vertex) determines to which vertex it connects in stage $i + 1$ (called destination vertex). Each vertex of stage i knows permutation p_i.

• If $p_i = \mathcal{G}^0$ and the source vertex is $(v, a_0, a_1, \ldots, a_\gamma)$, then its destination vertex is $(v', a_0', a_1, \ldots, a_\gamma)$ where $(v', a_0') = \mathrm{Rot}_G(v, a_0)$.

• If permutation $p_i = \mathcal{H}_s^t$, the label of the source vertex is $(v, a_0, \ldots, a_t, a_{t+1}, \ldots, a_\gamma)$, and the 16 digits in base D of a_{t+1} are $(k_{t+1,0}, \ldots, k_{t+1,s}, \ldots, k_{t+1,15})$, then its destination vertex is $(v, a_0, a_1, \ldots, a_t', a_{t+1}', \ldots, a_\gamma)$, where the value of $a_{t+1}' = (k_{t+1,0}, \ldots, k_{t+1,s}', \ldots, k_{t+1,15})$ and $(a_t', k_{t+1,s}') = \mathrm{Rot}_H(a_t, k_{t+1,s})$.

• If $p_i = \mathcal{X}^t$ and the label of the source vertex is $(v, a_0, \ldots, a_t, \ldots, a_\gamma)$ then the label of its destination vertex is $(v, a_0, \ldots, \breve{a}_t, \ldots, a_\gamma)$, where $\breve{a}_t$ is a_t when all its digit in base D are reverse.

Once each source vertex knows its destination vertex, we

can create all the connections between stages of the MIN simultaneously. The LR-Mesh needs two rows of processors to embed one row of the MIN; assume that the source vertex is j and the destination vertex is k. We can set a connection between vertices j and k in an LR-Mesh of $2D^{16}N^{\alpha} \times D^{16}N^{\alpha}$ as follows. Start in j, follow the upper row of the pair j until the main diagonal, then turn up (if $j < k$) or down (if $j > k$) and follow the vertical bus until the lower row of pair k, finally turn right (if $j < k$) or left (if $j > k$) to reach vertex k. This procedure takes constant time. The factor of two is necessary to avoid collisions.

Step 6. Compute rotation map of G_γ. We compute the rotation map of each vertex of G_γ as follows. All the vertices of stage γ transmit their complete label leftwards through the MIN. Each vertex of stage 0 reads the label.

If vertex with label $(v, a_0, a_1, \ldots, a_\gamma)$ receives label $(v', a'_0, a'_1, \ldots, a'_\gamma)$ then $\mathrm{Rot}_{G\gamma}((v, a_0, \ldots, a_{\gamma-1}), a_\gamma) = ((v', a'_0, \ldots, a'_{\gamma-1}), a'_\gamma)$.

Notice that all the values of $\mathrm{Rot}_{G\gamma}$ are distributed and stored in a single column of the LR-Mesh.

Lemma 2. Given the rotation map of a non-bipartite undirected D^{16}-regular graph $G = (V_G, E_G)$, where $|V_G| = N$, an LR-Mesh of $O(N^{\alpha}) \times O(N^{\alpha+3})$, where α is a constant, computes the rotation map of the expander G_γ in $O(1)$ time. □

4. Solving USTCON

In this section, we solve the undirected $s-t$ connectivity problem for selected vertices of graph G_γ on the LR-Mesh in constant time. Then, we employ this solution to create the adjacency matrix A^* of the transitive closure G^* of G. We assume that the input for this phase is the rotation map of G_γ.

4.1 Basic notation and definitions

Let $G_\gamma = (V_\gamma, E_\gamma)$ be an expander, where $|V_\gamma| = N^{\alpha}$ and α is a constant. We use graph G_M (that is basically a MIN) to represent how a path of G_γ evolves in time. It is useful to think of stage i of graph G_M as a photograph of the position, in time i, of a traveler when he/she follows a path in G_γ. Figure 3a shows an example of a graph G_γ and Figure 3b shows graph G_M.

Remark 1. The graph of Figure 3a is not an expander, but we use it to give an idea of the relation between G_γ and G_M.

Graph G_M has $\delta + 1$ stages, denoted by $s_0, s_1, \ldots, s_\delta$, where $\delta = O(\log N^{\alpha})$ is the diameter of G_γ. Since G_γ is D^{16}-regular, there are D^{16} edges between each vertex of stage i and the vertices of stage $i+1$, for $0 \leq i < \delta$. Notice

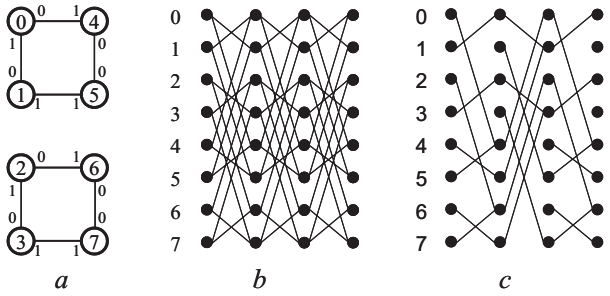

Figure 3. a) example of a 2-regular graph with N=8; b) Graph G_M obtained from the left graph; c) Graph G_R for the path 0,1,0.

that each vertex of stage i, $0 < i < \delta$, has D^{16} incoming edges from stage $i-1$ and D^{16} outgoing edges to the stage $i+1$. Each vertex of stage 0 has only D^{16} outgoing edges, and each vertex of stage δ has only D^{16} incoming edges. A path in graph G_M is a sequence of edges $e_1, e_2, \ldots, e_\delta$, where e_i is an edge between stages $i-1$ and i. In an arbitrary path of length δ, edges e_i and e_{i+1} are adjacent, for all $1 < i < \delta$. Each outgoing edge of a vertex in G_M has a label from the set $[D^{16}]$. We can think of any arbitrary path of length δ as a number of δ digits in base D^{16}. If $\delta = k \log N$, starting from vertex s, there are $T = D^{16\delta} = N^{\beta}$ different paths of length δ, where $\beta = (16k \log D)$ and k is a constant. Denote these paths by $q_0, q_1, \ldots, q_{T-1}$.

Consider the following restricted version of USTCON. Given an arbitrary path p_x of G_γ that starts in s, determine if the path reaches vertex t. To solve this problem is more convenient to use graph G_R. This graph is a subgraph of G_M that has the same vertices but only $1/D^{16}$ of its edges. For example, if path q_x uses edge $e_i = c$, then all the edges except edges with label c are useless between stages $i-1$ and i. Graph G_R eliminates useless edges in all the stages. Figure 3c shows graph G_R for the path $0, 1, 0$. Moreover, since we are just interested in the connectivity among vertices of G rather than vertices of G_γ, we are considering s and t to be sets of vertices rather than individual vertices. In this variation, our goal is to determine if one vertex of set s is connected to one vertex of set t. Notice that we check simultaneously the same path but starting in many different vertices of G_γ. Notice that graph G_R is a forest since each vertex in the stage i has at most one neighbor on stage $i+1$. We can construct graph G_{TE} by replacing each edge of G_R by a pair of edges. Since we can directly embed graph G_{TE} in an LR-Mesh, we use this graph to solve this version of USTCON in the LR-Mesh.

4.2 LR-Mesh USTCON algorithm

In this section, we present an algorithm that solves USTCON in the LR-Mesh. We briefly describe each main step and discuss the number of processors required by the LR-Mesh. Algorithm 3 shows its pseudo-code.

Step 1. Transform x to radix D^{16}. As we can see in the pseudo-code of Algorithm 3, for each path q_x, we need to transform number x from base 10 to base D^{16} to obtain the labels of edges $e_1, e_2, \ldots, e_\delta$ in the path. Remember that $0 \leq x \leq N^\beta$. This transformation starts with a division of x by D^{16}; then divide the quotient of this operation by D^{16}, and repeat this process until the quotient is zero. We can use an LR-Mesh of $O(N^\beta) \times O(N^\beta)$ to accomplish this task for each transformation by using an algorithm similar to the one presented in [5]. Once the set of quotients are available, the LR-Mesh computes concurrently the reminders of each division. The remainders are the edges $e_1, e_2, \ldots, e_\delta$. The LR-Mesh completes the transformation in $O(1)$ time. The size of the required LR-Mesh to perform this step is $O(N^{2\beta}) \times O(N^\beta)$, since we require N^β transformations.

Algorithm 3. USTCON on an LR-Mesh
input: Rotation map of G_γ ($\mathrm{Rot}_{G\gamma}$)
output: Adjacency matrix A^* of the transitive closure G^* of G.
begin
  for each path q_x, where $0 \leq x < T$ **pardo**
    1. Transform x to radix D^{16} to obtain $e_1, e_2, \ldots, e_\delta$
    for each pair $s, t \subset V_G$ **pardo**
      2. Create copies of G_{TE}
      3. Generate the adjacency matrix A^*
    end for
  end for
end

Step 2. Creating graph G_{TE}. We can create all the stages of graph G_R simultaneously. To generate the set of connections between stage $i-1$ and i, each vertex $\bar{v} = (v, a_0, a_1, ..., a_{\gamma-1})$ of stage $i-1$ needs the value of edge e_i and the rotation map $\mathrm{Rot}_{G\gamma}(\bar{v}, e_i) = (\bar{v}', e_i')$, both are available. Perform the connection between source vertex $\bar{v}$ and destination vertex $\bar{v}'$ in the same way as explained in Step 5 of Section 3.

We can generate graph G_{TE} from G_R by increasing the size of the LR-Mesh by a factor of four to create a double bus structure and generate a pseudo-tour of Euler. We can accomplish this task by using a procedure similar to the one presented in [2]. It is possible to embed graph G_{TE} in a LR-Mesh of $O(N^\alpha) \times O(N^\alpha \log N)$.

The above procedure generates a graph to check connectivity for only one specific path. So we perform the same process simultaneously for all the N^β different paths. Even more, this step generates all the entries of row v of the adjacency matrix A^*. To compute the remaining rows, generate N copies of each of the above graphs and compute the entries concurrently. The size of the required LR-Mesh to perform this step is $O(N^{\beta+\alpha+1}) \times O(N^\alpha \log N)$.

Step 3. Generate A^*. Once the connections of the pseudo-tour of Euler are set in all N^β graphs, each vertex of stage zero whose label starts with v writes v on its bus. If a vertex whose label starts with u in any stage of the N^β graphs reads v, then there exists a path that goes from v to u and $A^*(v,u) = 1$; otherwise, $A^*(v,u) = 0$. An LR-Mesh uses an OR operation to compute this step in constant time.

The LR-Mesh stores each row of the adjacency matrix A^* in a different row of the LR-Mesh. We allow $N-1$ empty rows of processors between each row of processors with information. In this way, the LR-Mesh can move all the entries of A^* to a single row or column in constant time.

Lemma 3. Given the rotation map of an expander $G_\gamma = (V_\gamma, E_\gamma)$, where $|V_\gamma| = N^\alpha$, and sets of vertices $s, t \subset V_\gamma$, an LR-Mesh of $O(N^{\beta+\alpha+1}) \times O\left(N^\alpha \log N\right)$ solves a variation of USTCON in $O(1)$ time. □

By combining Lemmas 2 and 3, we obtain Corollary 4.

Corollary 4. Given the rotation map Rot_G of a non-bipartite undirected D^{16}-regular graph $G = (V_G, E_G)$, where $|V_G| = N$, an LR-Mesh of $O(N^{\beta+\alpha+1}) \times O\left(N^{\alpha+3}\right)$ computes adjacency matrix A^* of the transitive closure of G in $O(1)$ time. □

5. Simulation of the R-Mesh on the LR-Mesh

In this section, we present a simulation of an R-Mesh of $M \times M$ processors on an LR-Mesh that runs in constant time. For this purpose, we just need to show that an LR-Mesh can simulate an arbitrary machine cycle of the R-Mesh in constant time. A machine cycle of an R-Mesh consists of the following four sub-cycles:

- Port configuration
- Bus write
- Bus read
- Computation

Before describing the simulation, we define the graph, G, of an R-Mesh configuration. This graph is a representation of the connections between the ports of the R-Mesh. Each block in the port partition of a processor generates a vertex of G. Thus, one can view each vertex of G as a set of ports internally connected within a processor of the R-Mesh. Let v_1 and v_2 be vertices of G. An edge exists in G between v_1 and v_2 iff an edge exists in the R-Mesh between ports p_1 and p_2, where ports p_1 and p_2 are elements of the partition blocks that generate vertices v_1 and v_2, respectively. Notice from Figure 1, that the internal connections of a single R-Mesh processor can generate up to four nodes of G, so the number of vertices, N, of G is at most $4M^2$.

Since Corollary 4 requires the input graph G to be regular, add as many self-loops to each vertex as necessary.

The inputs for the simulation are the port configuration of each R-Mesh processor, the bus data that each processor is going to write in its bus or buses, and the computation to be executed by each processor. Store this information on the first $4M^2$ processors of the top row of the LR-Mesh, such that a group of four processors of the LR-Mesh stores the information of a single processor of the R-Mesh. This arrangement of the inputs facilitates the movement of information in the LR-Mesh and reserves enough space to assign each node of graph G to a single processor of the LR-Mesh.

Next, we describe how the LR-Mesh simulates each of the four sub-cycles in constant time.

Port Configuration. Based on the port configuration of each R-Mesh processor, the LR-Mesh can determine the number of vertices of the graph G associated to each R-Mesh processor. The LR-Mesh can assign a sequential label to each vertex of graph G by performing a prefix sum [10] in constant time. Then, by exchanging indices between processors that store the information of neighboring processors of the R-Mesh, the LR-Mesh obtains the rotation map of each vertex of G in constant time. The LR-Mesh requires $N \times N$ processors to generate (Rot_G).

Since (Rot_G) is known, by using Corollary 4, the LR-Mesh obtains the adjacency matrix A^* of the transitive closure of G in constant time using $O(N^{\beta+\alpha+1}) \times O\left(N^{\alpha+3}\right)$ processors.

The ones in each row of matrix A^* represent the set of vertices that belong to same connected component of G. Using neighbor localization on linear buses [10] and prefix sum, the LR-Mesh can assign to each vertex of G a connected component label. Then the LR-Mesh can sort [5] all the vertices by its component number in constant time and fuse them together to create a single linear bus per component. This step takes constant time in an LR-Mesh of $O(N) \times O(N)$ processors. for each row of A^*

Bus write simulation. Each vertex writes their bus data to its bus and let the bus to decide the final bus data based on the write rule assumed by the LR-Mesh. This step takes constant time.

Bus read simulation. The bus data is available at the bus at the end of the bus write sub-cycle. Each vertex reads its bus and stores the resulting value in some variable.

Computation. A single processor of the LR-Mesh can perform this operation based on the vertices of graph G it handles. This operation takes constant time since at most, a processor handles four vertices of G.

Theorem 5 summarizes the result of this paper.

Theorem 5. A Common or Collision CRCW R-Mesh of $M \times M$ processors can be simulated by a CREW LR-Mesh of $O(M^{2(\beta+\alpha+1)}) \times O\left(M^{2(\alpha+3)}\right)$ processors in $O(1)$ time, where β and α are constants. □

6. Concluding remarks

We have shown that an LR-Mesh can simulate an R-Mesh in constant time. This is the fastest simulation of this type. The drawback of the simulation is the great amount of resources required by the LR-Mesh. We think it is possible to optimize portions of the operations and reduce the size of the required LR-Mesh; however, the great number of processors is due mainly to the large number of vertices in the expander and the paths generated to solve USTCON. Notice that Reingold's algorithm is very efficient in space in the sequential version, but it requires an extremely high number of operations. Notice that the LR-Mesh does not require concurrent writes to perform the simulation. Since the graph G of the R-Mesh has at most degree four, we therefore conjecture that it may be possible to reduce the number of processors in the simulating model.

References

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

[2] J. A. Fernández-Zepeda, R. Vaidyanathan and J. L. Trahan. Using Bus Linearization to Scale the Reconfigurable Mesh. *Journal of Parallel and Distributed Computing*, 62(4):495–515, April 2002.

[3] O. Gabber and Z. Galil. Explicit Constructions of Linear-sized Superconcentrators *Journal Computer and System Sciences*, 22(3):407–420, June 1981.

[4] J. JáJá, Introduction to Parallel Algorithms. Addison Wesley, 1992.

[5] J.-w. Jang and V. K. Prasanna. An Optimal Sorting Algorithm on Reconfigurable Mesh. *Journal of Parallel and Distributed Computing*, 25(1):31–41, Feb. 1995.

[6] S. Jimbo and A. Maruoka. Expanders Obtained from Affine Transformations. *Combinatorica*, 7(4):343–355, 1987.

[7] H. Li and Q. F. Stout. Reconfigurable SIMD Massively Parallel Computers. *IEEE Proceedings*, 79(4):429–443, 1991.

[8] M. Maresca. Polymorphic Processor Arrays. *IEEE Transactions on Parallel and Distributed Systems*, 4(5):490–506, May 1993.

[9] G. A. Margulis. Explicit Construction of Expanders. *Problemy Peredači Informacii*, 9(4):71–80, 1973.

[10] R. Miller, V. K. Prasanna-Kumar, D. Reisis and Q. Stout. Parallel Computations on Reconfigurable Meshes, *IEEE Transactions on Computers*, 42(6):678–692, June 1993.

[11] N. Nisan and A. Ta-Shma. Symmetric Logspace is Closed under Complement. *In 27th ACM Symp. Theory of Computing*, pages 140–146, 1995.

[12] O. Reingold. Undirected ST-connectivity in Log-space. *In Symposium on the Theory of Computing (STOC)*, pages 376–385, 2005.

[13] J. L. Trahan, R. Vaidyanathan and R. K. Thiruchelvan. On the Power of Segmenting and Fusing Buses. *Journal of Parallel Distributed Computing*, 34(1):82–94, April 1996.

[14] R. Vaidyanathan and J. Trahan. Dynamic Reconfiguration: Architectures and Algorithms. Kluwer Academic / Plenum Publishers, New York, 2004.