The Well-Connected Processor Array

Dan Gordon

Abstract—A new theoretical model for reconfigurable processor arrays is introduced. Most of the models considered in the literature are similar to the reconfigurable mesh (RMESH), in which each processing element (PE) is connected to its four neighbors by reconfigurable buses. In the new model, called the "well-connected processor array" (WECPAR), every PE is connected to each neighbor by $k \geq 1$ point-to-point lines, and it also controls the switching between those lines. k is called the connectivity of the WECPAR. Any line entering the PE can either be connected to the PE itself, or it can be connected by the PE to another line, thus enabling complex switching configurations. This model is suitable for arrays in which the computation and memory areas of a PE are very much larger than a switch area. The concept of a *burden* placed on a PE by the lines connected to or passing through it is introduced. This is used to derive a lower bound of $k = \Omega(d^{3/2}/e)$ on the connectivity required by a WECPAR to embed any graph of degree $\geq d$ with expansion e; for $e = 1$, this result is sharp. Various other issues are examined: graph embeddings, algorithms, broadcasting, routing, and self-simulation. A novel transportation-type routing method utilizes the connectivity for efficient routing. A sample algorithmic result is that an n-point FFT can be done in logarithmic time on a WECPAR of n PEs with a connectivity of $\sqrt{n/2}$.

Index Terms—Graph embeddings, multi-connected processing elements, parallel computing, point-to-point communications, reconfigurable processor array, routing, self-simulation, WECPAR

1 Introduction

RECONFIGURABLE processor arrays are regular interconnected arrays of processors in which the connections can be reconfigured during the execution of some program. For some of the earlier papers on this topic, see [1]–[8].

Most of the literature on reconfiguration considers the *reconfigurable mesh* (RMESH), which consists of an array of PEs connected by reconfigurable buses. Each processor is usually connected to its four neighbors by a single bus, and it can control the configuration of the buses passing through it. Many aspects of reconfiguration have been considered, such as graph embeddings, algorithms, routing, and fault tolerance. For an introduction to this topic, see, for example [9], [10].

In this paper, we consider a different theoretical model, called "the well-connected processor array" (WECPAR). It differs from the RMESH in two ways. Firstly, the communication lines are only point-to-point (p2p), and secondly, each PE has $k \geq 1$ lines on each side connecting it to its four neighbors. For each incoming line, a PE can either connect to that line (and to others), or it can connect the line to another line, thus forming long communication lines between distant PEs. We also assume that the PEs on the boundaries are connected to some external host.

In one sense, this model is more limited than an RMESH because a PE does not listen to the lines passing through it. On the other hand, the ability of a PE to connect to or configure more than four lines offers several advantages. The capabilities we attribute to a WECPAR apply only to the basic model, and this does not preclude additional capabilities, such as a common bus or horizontal and vertical buses, or access to a shared memory. The purpose of this introductory study is to investigate the possibilities offered by the point-to-point multiple connections of the WECPAR. This model is suitable for processor arrays in which the processing and memory areas of each PE are very significantly larger than that of a single switching element, so all area required for the switches can be covered by the PE area.

One motivation for this study stems from [8], which considered a model with just one communication line between adjacent PEs. It was shown in [8] that in this model, a complete binary tree can be embedded in a 2D array of PEs so that (asymptotically) 100% of the PEs are utilized as tree nodes. By comparison, the well-known H-tree scheme utilizes only 50% of the PEs, and previous schemes for hexagonal grids achieve 70% [6]. Binary tree embeddings have been studied very extensively from various perspectives; besides the above-mentioned papers, see also [11]–[16]. The WECPAR is a natural extension of the model considered in [8].

The feasibility of the WECPAR model is based on the assumption that in today's technology, there are sufficiently many VLSI layers so that horizontal and vertical communication lines can be laid out in two layers, and this would still leave enough layers for the processing and switching elements. The WECPAR can also be viewed as a fusion between a mesh of processors and a network-on-chip; see [17]–[21] for some references to the latter concept. The idea of having multiple communication lines has been studied before by ElGindy et al. [22], where multiple parallel buses are used to connect a circular array of processors. Since the WECPAR model is theoretical, we do not go into the details of implementation. For some recent relevant possibilities, see, for example, [23], and for some optical alternatives, see [24], [25].

The WECPAR idea raises the question of how it compares with the well known RMESH from a theoretical and a practical point of view. Since it is a different concept, it cannot be

• *D. Gordon is with the Department of Computer Science, University of Haifa, Haifa 31905, Israel. E-mail: gordon@cs.haifa.ac.il.*

Manuscript received 05 Apr. 2012; revised 29 Oct. 2012; accepted 01 Nov. 2012; published online 25 Nov. 2012; date of current version 29 Apr. 2014.
Recommended for acceptance by K. Li.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2012.280

easily compared. For example, broadcasting is obviously faster (theoretically) on an RMESH, but a WECPAR is clearly more suitable for simulating an evolving neural network. From an implementation point of view, the RMESH buses require more power to drive a signal to all the nodes in their path, but point to point lines can potentially be implemented with light signals which require less power. The literature on RMESH algorithms and applications is very extensive, so a comprehensive comparison between the two models is beyond the scope of this paper.

This paper examines several aspects of the WECPAR: theoretical properties (Section 3), graph embeddings (Section 4), broadcasting, routing and self-simulation (Section 5), and algorithms (Section 6). The basic switching configurations are explained in Sections 2 and 7 concludes with some further research directions.

2 Switching Configurations

We formally define an (m, n, k)-WECPAR W as an $m \times n$ array of PEs, with k data paths between every adjacent pair of PEs. Connections between PEs run only horizontally or vertically. k is called the *connectivity* of W, and we will also refer to W as a k-WECPAR. The paths at the boundary PEs are assumed to be connected to an external host.

At any given time, every path entering a PE is one of three states: it is directly connected to the PE through a port, or it is connected to one (and only one) other path entering the PE, or it is simply disconnected. The PE is in direct control of the state of each data path, and it is assumed that once two paths are connected by the PE, they function as one continuous path without interfering with other processes or connections of the PE. Note that throughout the paper, we use the following three terms interchangeably: (communication) path, (data) line and wire.

Fig. 1 shows a schematic diagram of PE with connectivity 3 and the possible switch configurations inside the PE. Each incoming line can be connected directly to the PE through a port, shown schematically as a circular switch. Alternately, if the line is not connected to the PE, it can be connected to another line via one or more wire switches, shown as squares. The figure also shows all possible configurations of such wire switches.

Note that two types of loops are theoretically possible in this model. One type is when one port of a PE is connected (via switches in other PEs) to another port of the same PE. This situation can be easily detected by the PE as soon as a signal is sent from one port to the other. Another type of loop can occur when a line forms a closed circuit without any PE connected to the line. This situation does not affect any PE in the WECPAR.

3 Theoretical Results

This section presents two theoretical results: the NP-completeness of embedding a graph in a WECPAR, and a lower bound on the required connectivity of such an embedding. We recall some standard definitions related to graph embeddings. Let G and H be two graphs. An *embedding* of G (the guest graph) in H (the host graph) is a one-to-one mapping of the vertices of G into the vertices of H such that the edges of G are mapped to edge-disjoint paths of H. The number of vertices of H divided by the number of vertices of G is called the *expansion* of the embedding. We modify these concepts for WECPARS.

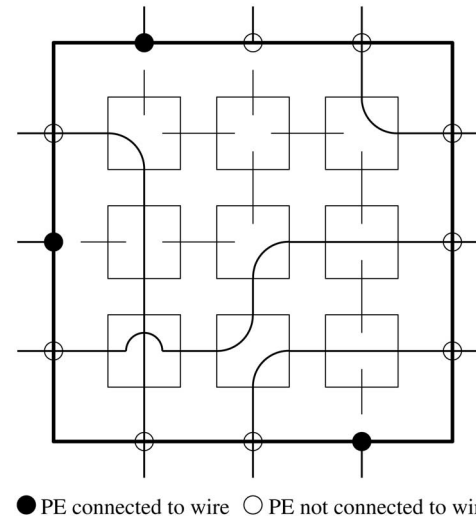

Fig. 1. Example of a configuration of switches in a PE.

Definition 1. *An embedding of a graph $G = (V, E)$ in a WECPAR W is a pair (f, c) such that f is a 1-1 mapping from V to the PEs of W and c is a configuration of switches in W such that for every edge $\{u, v\} \in E$, $f(u)$ and $f(v)$ are connected in W. The*expansion *of the embedding is the number of PEs in W divided by $|V|$. For convenience, we define $f(e)$, for $e = \{u, v\} \in E$, as the path in W from $f(u)$ to $f(v)$. (If there is more than one path from $f(u)$ to $f(v)$, then one of them is chosen arbitrarily to be $f(e)$).*

3.1 NP-Completeness of the Embedding Problem

The following natural question immediately arises: Can a given graph be embedded in a given WECPAR? We formally define the GEW (graph embedding in a WECPAR) problem as follows: Instance: an encoding $\langle G, W \rangle$ of a graph G and a WECPAR W. Question: Can G be embedded in W? Note that formally, we consider GEW to be the language consisting of all the positive instances of this problem. Not surprisingly, we have:

Theorem 1. *The GEW problem is NP-complete.*

Proof. We first show that the problem is in NP. Suppose we are given an instance $\langle G, W \rangle$ of GEW with a graph $G = (V, E)$ and an (m, n, k)-WECPAR W. Nondeterministically, we choose a 1-1 function f from V to the PEs of W, and a configuration of the switches in W. Then we check in polynomial time that every edge $\{u, v\} \in E$ is mapped to a path in W connecting $f(u)$ and $f(v)$.

To show that GEW is NP-hard, we show that the restricted problem with $m = 1$ is NP-hard. We call the restricted problem GEW1. For a given $(1, n, k)$-WECPAR, we label its PEs $P_1, P_2, \ldots, P_n$ by numbering them consecutively from left to right. Let us recall the minimum cut linear arrangement (MCLA) problem [26, p.201]: Instance: an encoding $\langle G, k \rangle$ of a graph $G = (V, E)$ and a positive integer k. Question: is there a one-to-one function $f : V \to \{1, 2, \ldots, |V|\}$ such that for all $1 \leq i < |V|$,

$|\{\{u,v\} \in E | f(u) \leq i < f(v)\}| \leq k$? We shall prove that MCLA is polynomially reducible to GEW1.

We define a mapping from instances of MCLA to instances of GEW1 as follows. Given an instance $\langle G, k\rangle$ of MCLA, where $G = (V, E)$, we map it to an instance $\langle G, W\rangle$ of GEW1 consisting of the same graph G and a $(1, n, k)$-WECPAR W, where $n = |V|$. We shall show that $\langle G, k\rangle \in \text{MCLA} \Leftrightarrow \langle G, W\rangle \in \text{GEW1}$.

Assume $\langle G, k\rangle \in \text{MCLA}$, and let f be a mapping as in the MCLA problem. Let $V = \{v_1, v_2, \ldots, v_n\}$ be a labeling of V such that for all $1 \leq i \leq n$, $f(v_i) = i$. We define a mapping $g : V \to \{P_1, P_2, \ldots, P_n\}$ simply as $g(v_i) = P_i$, for $1 \leq i \leq n$. What remains to be seen is that the edges of G can be mapped to connected paths in W. This is not immediately obvious, because in the MCLA, no accounting is needed for the left and right ports of the PEs.

We begin with mapping all the edges emanating from v_1. Assume that there are ℓ such edges. Clearly, $\ell \leq k$. In the MCLA, these edges can be viewed as line segments connecting the point 1 to some other points between 2 and n. Since $\ell \leq k$, we simply utilize ℓ ports on the right side of P_1, and connect the communication paths from these ports in straight paths to the left ports of the PEs corresponding to the other vertices of the edges from v_1.

We now continue with the edges from v_2 (except for an edge $\{v_1, v_2\}$, if it exists in E). We utilize the free ports on the right of P_2 to map these edges to straight paths to the left ports of PEs numbering 3 or above. Since all the paths from P_1 were straight, all the free ports on the left of P_2 can be connected by straight paths to PEs lying to the right of P_2. Due to the limit of k in the MCLA problem, it is clear that there must be sufficiently many free ports on the left of P_2 for this purpose. We now continue with P_3, P_4, and so on, in a similar manner, until all the edges are mapped to connected paths in the WECPAR. This proves that G can be embedded in W; i.e., $\langle G, W\rangle \in \text{GEW1}$.

Assume now that $\langle G, W\rangle \in \text{GEW1}$; i.e., G is embeddable in W. Let $V = \{v_1, v_2, \ldots, v_n\}$ be a labeling of V such that for $1 \leq i \leq n$, v_i is mapped to P_i. We define the mapping f required for the MCLA problem as $f(v_i) = i$, for $1 \leq i \leq n$. The required bounding condition in the MCLA holds trivially because there are only k ports on each side of a PE. Note that some of the embedded paths may make "turns" back and forth between the PEs, but this is obviously not a problem. □

3.2 Lower Bounds on Connectivity Required for Embedding

We consider the following problem: Given a graph with a minimal vertex degree d, what is the minimum connectivity k required to embed it in a WECPAR? Since the number of ports of a PE is $4k$, k must satisfy $k \geq d/4$. However, unless the graph is a subgraph of a 2-dimensional grid, some ports must be used for paths that pass through the PE. The following concept will be used to obtain a nontrivial lower bound on the required connectivity.

Definition 2. *Let $G = (V, E)$ be a graph, W a WECPAR, and (f, c) an embedding of G in W. For $p \in V$, let $P = f(p)$ and Q some other PE in W. We define the burden that P places on Q, denoted by $b(P, Q)$, as the number of paths with endpoint P and passing through Q or terminating in Q. For a set S of PEs, we define $b(P, S) = \sum_{Q \in S} b(P, Q)$, and $b(P) = \sum_{Q \in W} b(P, Q)$.*

Note that if we have a path with endpoints P, Q (which is the image of an edge of G) then $b(P, Q) + b(Q, p)$ is the exact number of ports used by the path (1 port from P, 1 from Q, and 2 ports from every PE through which the path passes). It follows that $\sum_{p \in V} b(f(p))$ is the total number of ports used by the embedding.

Theorem 2. *Given a graph $G = (V, E)$ with vertex degree $\geq d$, and an N-node square WECPAR W, the connectivity required to embed G in W with expansion e is $k = \Omega(d^{\frac{3}{2}}/e)$. For $e = 1$, this result is sharp.*

Proof. Let (f, c) be an embedding of G in W, $p \in V$ and $P = f(p)$. We will first derive a lower bound for $b(P)$. For an integer $r \geq 1$, denote by $S(P, r)$ the set of PEs in W whose distance from P is exactly r. Let $s_r = |S(P, r)|$ (the size of $S(P, r)$). Assuming that $r \leq \sqrt{N}$, we have $r + 1 \leq s_r \leq 4r$ ($s_r = r + 1$ if P is a corner PE, and $s_r = 4r$ if P is at distance $\geq r$ from all the boundary PEs of W).

Clearly, $b(P, S(P, 1)) \geq d$ because all paths from P either pass through or connect to PEs of $S(P, 1)$. Consider now the PEs of $S(P, 2)$: $b(P, S(P, 2) \geq d - 4$, because with the possible exception of 2 to 4 direct connections between P and $S(P, 1)$, all the other paths from P either pass through or connect to PEs of $S(P, 2)$. In general, for $r \geq 1$, we have

$$b(P, S(P, r)) \geq d - 4 - 8 - \cdots - 4r = d - 2r(r + 1). \quad (1)$$

Naturally, we should only consider radii r for which $d - 2r(r + 1) > 0$. Let R be the maximal integer s.t. $d - 2R(R + 1) > 0$, i.e., $2(R + 1)(R + 2) \geq d > 2R(R + 1)$. Hence, $d = \theta(R^2)$.

We now get our lower bound on $b(P)$ by summing the lower bounds of Eq. (1):

$$\begin{aligned} b(P) &\geq \sum_{r=1}^{R} b(P, S(P, r)) \geq \sum_{r=1}^{R} (d - 2r(r + 1)) \\ &= Rd - \frac{4}{3}R^3 + 2R^2 - \frac{20}{3}R + 4. \end{aligned} \quad (2)$$

The last equality in Eq. (2) follows from the standard summation equations $\sum_{r=1}^{R} r = \frac{1}{2}R(R + 1)$ and $\sum_{r=1}^{R} r^2 = \frac{2}{3}R^3 - \frac{3}{2}R^2 - \frac{23}{6}R - 2$. Using $d > 2R(R + 1)$ in Eq. (2), we get

$$b(P) > 2R^3 - \frac{4}{3}R^3 + [\cdots] = \frac{2}{3}R^3 + [\cdots], \quad (3)$$

where $[\cdots]$ stands for lower order terms of R. Now, $d = \theta(R^2) \Rightarrow b(P) = \Omega(d^{\frac{3}{2}})$.

Note now that the number of vertices of G is N/e, so the total burden, (which is also the number of ports used) is $\frac{N}{e}\Omega(d^{\frac{3}{2}})$ Therefore, the average burden placed on a single PE is $\Omega(d^{\frac{3}{2}}/e)$. The minimal required connectivity k is at least 1/4 of the average burden, so $k = \Omega(d^{\frac{3}{2}}/e)$.

To see that the result is sharp for $e = 1$, we shall show that for infinitely many values of d, there are graphs with vertex degrees $\geq d$ which can be embedded in a WECPAR with connectivity $O(d^{\frac{3}{2}})$. Given an integer R, we consider

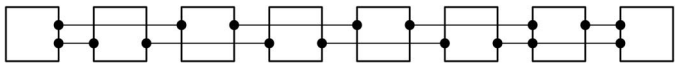

Fig. 2. Circular array embedded in a 2-WECPAR row.

the following graph G: the vertices form a square grid of size at least $(R+1) \times (R+1)$, and two vertices are connected by an edge iff the L_1-distance between them is $\leq R$. By calculations similar to the above, we can see that the minimal vertex degree, obtained at a corner, is $d = 2 + 3 + \cdots + R = R(R+1)/2 + 1$. According to the above result, the required connectivity to embed G in a square WECPAR W of the same size (expansion $e = 1$) is $k = \Omega(d^{\frac{3}{2}}) = \Omega(R^3)$.

Embedding G in a WECPAR W (of the same size) can be done as follows. The vertices of G are mapped in a straightforward manner to the PEs of W, and horizontal and vertical edges are mapped to horizontal and vertical paths. Mapping the diagonal edges is done similarly to the diagonal paths in Fig. 4: all the diagonal paths have just one turn of $90°$, and all the switch configurations are identical. Burden arguments similar to the above can be used to show that this embedding can be done with connectivity $O(d^{\frac{3}{2}})$. □

4 Graph Embeddings

Circular arrays. Fig. 2 shows how a circular array can be embedded into a single row of a 2-WECPAR. This construction can be done in all the rows and columns of a rectangular 2-WECPAR, resulting in a 2D circular array, which is topologically equivalent to a torus.

Binary tree in a WECPAR row. A different type of embedding is that of a binary tree of level ℓ in a single row or column of a WECPAR, as shown in Fig. 3 for a tree of level 3. The required connectivity is ℓ, i.e., it is logarithmic in the row length. If this construction is done in every row and column of a square WECPAR, the resulting graph is known as a *mesh of trees*; see [27, p.120].

6- and 8-connected grids. Fig. 4 shows an embedding of an 8-connected grid in a 3-WECPAR. Note that all PEs have the same switch configurations. An hexagonally-connected grid can also be implemented in a similar manner, requiring a connectivity of 2.

Hypercubes. One of the most useful graphs for parallel applications is that of a hypercube. Such a cube can also be embedded in a WECPAR, as shown in Fig. 5 for a hypercube of dimension 4. Using the short connecting lines, PEs 0–3 form a 2D square, and so do PEs 4–7. The two squares are connected so as to form a 3D cube with PEs 0–7. A similar construction is done with PEs 8-15, and the two 3D cubes are joined to form a hypercube of dimension 4. It is easy to see that a hypercube of even dimension d requires a WECPAR of size $2^{d/2} \times 2^{d/2}$, with connectivity $2^{d/2} - 2$. If d is odd, the required WECPAR is of size $2^{(d+1)/4} \times 2^{(d+1)/2}$.

Binary tree embeddings in a 2-WECPAR. In [8] it was shown how a complete binary tree can be embedded into a 1-WECPAR

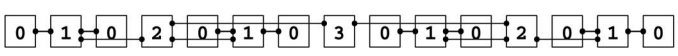

Fig. 3. Embedding of a binary tree in a WECPAR row.

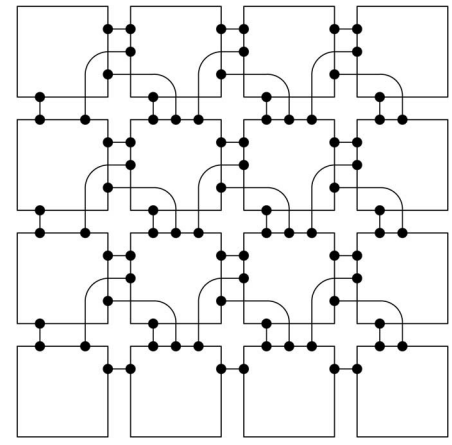

Fig. 4. Eight-connected array embedded in a 3-WECPAR.

with asymptotically 100% utilization of the nodes. In this context, we define a "tile" to be a rectangular array of PEs containing a complete binary tree and one extra PE which is used as a tree node in the recursive construction of higher-level trees. The embedding of [8] is quite complicated, requiring five types of basic "tiles"; five larger tiles, each made up of four basic tiles, can then be laid out recursively to produce the result. Furthermore, in an $(n, n, 1)$-WECPAR, about $2\sqrt{n}$ of the border PEs are not utilized by the construction. Using a 2-WECPAR, we can embed a binary tree using 100% of the PEs, and requiring only two basic tiles, each of size 4×4 PEs.

Fig. 6 shows how a 2-WECPAR can be used efficiently to embed a complete binary tree with 100% utilization. The scheme uses two types of tiles, called **A** and **B**, each containing a complete binary tree. Trees of higher levels are obtained by a recursive construction of lower-level tiles. Each tile also contains a "spare" PE which is used for higher level nodes in the recursive construction. At the lowest level, each tile consists of a 4×4 square of PEs forming a tree of level 3.

Fig. 6(a) shows the recursive construction of an **A**-tile of level 5 made up of one basic **A**-tile (of level 3) in the upper right quadrant, a mirrored **A**-tile (labeled **A1**) in the upper left quadrant, a **B**-tile in the lower left quadrant, and a rotated **B** in the lower right quadrant.

Fig. 6(b) shows the recursive construction of a **B**-tile of level 5, made up of four tiles of level 3. The construction uses

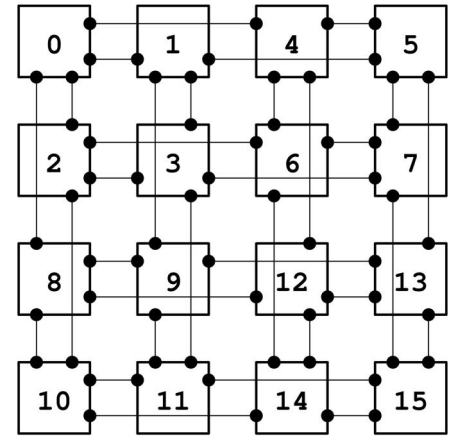

Fig. 5. Four-dimensional hypercube embedded in a 2-WECPAR.

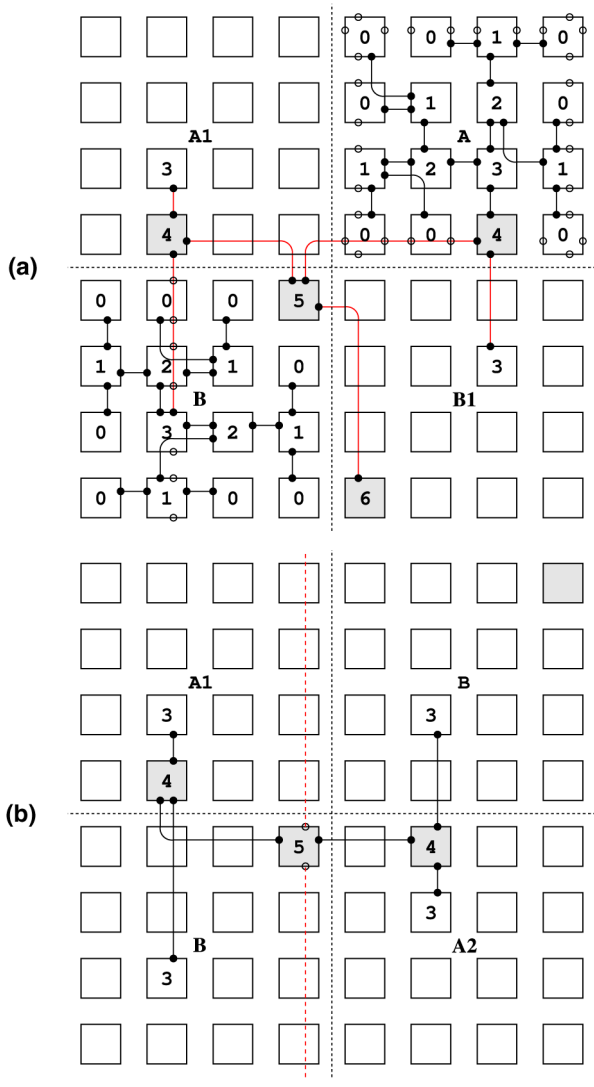

Fig. 6. Recursive layout of an A-tile (a) and a B-tile (b) for laying out a binary tree, showing details of the basic A- and B-tiles. A1 and A2 are mirrored versions of A, and B1 is a rotated B.

two different reflections of an A-tile (A1 and A2), and two B-tiles. The tiles can be characterized as follows.

1) In all the boundary PEs, the ports near the extreme edges are free to be used for edges of higher-level trees. These ports are marked by open circles in the A-tile of Fig. 6 (but not marked in the B-tile).
2) In an A-tile, the extra PE is on the bottom row, immediately to the right of the center, and it can be connected to the root of the A-tile.
3) In a B-tile, the extra PE is in the top right corner.
4) The root of a B-tile (at all levels) is immediately to the left and below the tile's midpoint.
5) The root of a B-tile can be accessed from the top and bottom of the tile by straight lines passing through ports which are free for this purpose; these ports are marked by empty circles in the B-tile of Fig. 6(a). Furthermore, the free paths to the root are immediately to the left of the vertical center line of the tile; these paths are shown by the red dashed lines in the higher level B-tile of Fig. 6(b).

It can be easily verified by inspection that the above properties carry over to higher level trees constructed recursively.

5 Broadcasting, Routing and Self-Simulation

5.1 Broadcasting

A most fundamental step is that of broadcasting. This cannot be done $O(1)$ time as in an RMESH, but we can use the size of the connectivity to speed up the obvious logarithmic time.

Theorem 3. *Let W be an (n, n, k)-WECPAR, and assume $n = 2^\ell$. Then broadcasting from a corner PE to all the others can be done in time $\lceil \log n / \log(k+1) \rceil$.*

Proof. Assume first that $k = 1$. We denote the ith PE in row 0 by $\mathrm{PE}(i)$. Broadcasting can be done in 2ℓ reconfiguration steps as follows. The initial configuration connects $\mathrm{PE}(0)$ with $\mathrm{PE}(n/2)$, and $\mathrm{PE}(0)$ sends the information to $\mathrm{PE}(n/2)$. This scheme now continues recursively in each half of the row, and the total number of configuration steps is clearly $\ell = \log_2(n)$. In the next step, the top PE in each row broadcasts down the column in the same number of steps.

For $k > 1$ we divide the row into $k+1$ spans of (almost) equal size. $\mathrm{PE}(0)$ is connected to the leftmost PE of each other span, and the data is transmitted. This scheme continues recursively in each span. The number of reconfiguration steps is $\lceil \log_{(k+1)} n \rceil = \lceil \log n / \log(k+1) \rceil$. □

5.2 Routing

The problem of routing on reconfigurable meshes has been studied very extensively; see, for example [28] and the references therein. The WECPAR offers a unique approach to this problem by assigning some of the connections to a cyclic reconfiguration scheme which transports the packets in a regular and predictable manner.

The idea is demonstrated in Fig. 7 for connectivity $k = 3$. The figure shows the first two stages of the scheme on a single WECPAR row. In stage 1, the row is divided into spans of $k+1$ PEs, and the leftmost PE of each span connects to all the PEs of

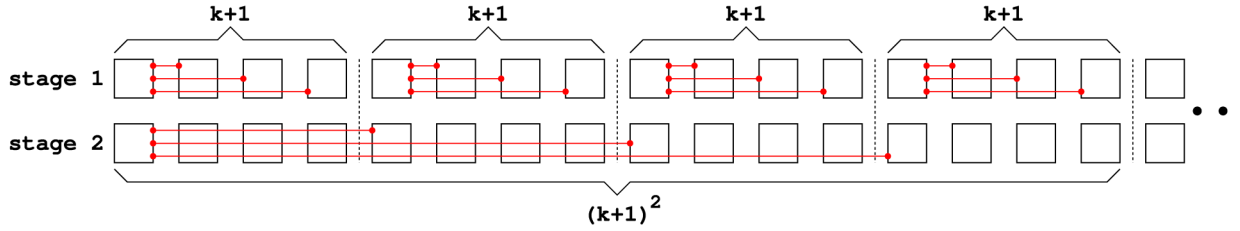

Fig. 7. First two stages of the routing scheme on a 3-WECPAR row.

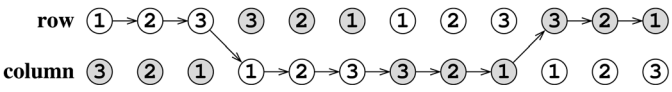

Fig. 8. Path followed by a packet along the row and column configurations.

the span. Every PE that has a packet to transmit sends it to the leftmost PE, and if the destination is within the span, it is sent immediately to its destination.

At stage 2, only the leftmost PE of each span is active and all the others act as connectors. New spans are formed, consisting of $(k+1)^2$ PEs, and the leftmost active PE of each (new) span connects with the k active PEs in its span, and receives the required data. This continues for ℓ stages, where ℓ is the minimal number such that $(k+1)^\ell \geq n$, where n is the row length; i.e., $\ell = \lceil \log n / \log(k+1) \rceil$.

The first ℓ stages will be called "data gathering". After these stages, the configurations are repeated in the reverse order, and in each span, the data is transmitted from the leftmost PE to the active PEs in its span. Note that the $(\ell+1)$st configuration is identical to the ℓth configuration, but the data travels in the opposite direction. This is done until the first configuration is reached again, and then every packet will reach its destination; these stages will be called "data distribution". The cycle is now repeated. The total number of reconfiguration steps in a cycle is 2ℓ. For example, if $8 < n \leq 16$ and $k = 3$, then $\ell = 2$; see Fig. 7.

To transmit data both horizontally and vertically in an (n,n)-WECPAR, we employ the same scheme also on the columns so that at every time step, all row configurations are identical, and the same holds for the columns. However, to reduce latency, stage 1 of the column scheme coincides with the $(\ell+1)$st stage of the row scheme, so a packet waits ℓ stages at most until its transmission begins.

Furthermore, once a packet gets under way, it needs at most two cycles to reach its destination. This is explained with the aid of Fig. 8, which shows an example of the stages along the rows and columns for $\ell = 3$. The numbers represent the configuration types, empty circles represent data gathering, and shaded circles represent data distribution. The figure shows the path taken by a packet starting from stage 1 along its row. After stage 3 (gathering), it takes stage 1 along the column until the end of the cycle, i.e., it reaches its final row. Then, the packet continues along the row at the exact cycle point where it left off, reaching its final destination at the end of the second cycle. These results are summarized in Theorem 4 below.

Theorem 4. *Assuming that k lines per side are allotted to communications, routing on an (n,n)-WECPAR can be done in 5ℓ reconfiguration steps, where $\ell = \lceil \log n / \log(k+1) \rceil$.*

5.3 Self-Simulations

The term "self-simulation" usually refers to the ability of a small reconfigurable network to simulate a large network with a minimal loss of time. In [29], it was shown that if an RMESH is restricted to have only linear buses, then a large mesh can be simulated in optimal (linear) time. Even though a linear bus is similar in shape to a p2p line, the result does not carry over to WECPARS, because it uses a certain connected-component algorithm which relies on the broadcast capabilities of a bus.

The major difference between self-simulation with buses and p2p lines is that with buses, the message has to be delivered to all the PEs connected to the bus, but with p2p lines, the message only has to be delivered from one PE to the PE at the end of a line. Thus, the self-simulation problem for WECPAR s just reduces to a routing problem, and this can be solved by the routing scheme of Section 5.2.

A different type of simulation is sometimes possible when we want to simulate a WECPAR of a given connectivity k by a larger WECPAR with lower connectivity, or vice versa. This is trivially possible for certain values of WECPAR sizes and connectivity: consider Fig. 1, which shows an example of the configuration of switches in a PE with connectivity 3. Every such PE can be replaced by a 3×3 block of PEs of a WECPAR with nine times as many PEs and connectivity 1. Every simulated transmission step is followed by an internal transmission step in the blocks, in which the data is sent to the central PE, which performs the required computation. This is followed by another internal step within the blocks to deliver the data from the central PE to the others, and then the external transfers are done. Clearly, the reverse simulation is also possible.

Self-simulations can also be specific to certain applications, as will be shown in Section 6.2.

6 Algorithms on the WECPAR

6.1 Semi-Group Operations

Semi-group operations, such as computations of sums, can be done on a WECPAR in logarithmic time, similarly to such operations on an RMESH. Can such operations be done any faster on a WECPAR using its connectivity? Theoretically, the answer is negative. However, we can express the time in terms of the time it takes a single PE to compute such an operation on several elements. This could be useful when the PEs have dedicated hardware for this purpose.

Theorem 5. *Let W be an (n,n,k)-WECPAR, and assume that computing a certain semi-group operation on $k+1$ elements on a single PE takes $f(k)$ time. Then computing this operation on all the elements of W takes $O(f(k)\log n/\log(k+1))$ time.*

Proof. The proof utilizes the same configuration steps used for routing, as shown in Fig. 7. In stage 1, all elements transmit their data to the leftmost PE in their span. Each such PE now computes the operation on its $k+1$ elements in time $f(k)$. In stage 2, the spans are increased by a factor of $k+1$, and the scheme continues with the PEs that were leftmost in the shorter span. This continues for ℓ steps, where $\ell = \lceil \log n / \log(k+1) \rceil$, at the end of which the leftmost PEs contain the result for their rows. The scheme is now applied to the first column, and the result follows. □

6.2 Normal Butterfly and Hypercube Algorithms

Many parallel algorithms have been developed for various graphs of PEs. In this section, we will concentrate on some algorithms that utilize the WECPAR's connectivity.

One family of such algorithms consists of the so-called normal butterfly algorithms; see [27, Section 6.2]. To describe these algorithms, we first explain the butterfly network.

The butterfly network of rank r consists of $(r+1)2^r$ nodes, organized as $r+1$ ranks, numbered $0,\ldots,r$, with each rank containing 2^r nodes. Fig. 9 shows the standard butterfly graph

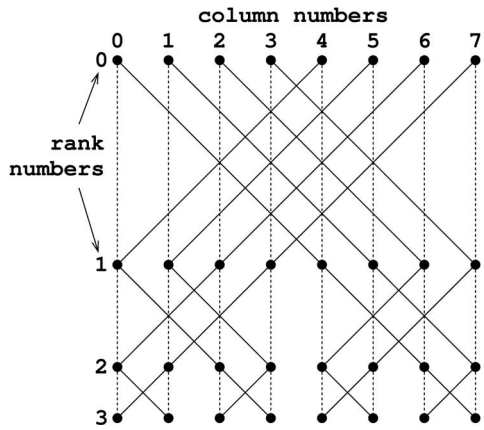

Fig. 9. Butterfly network of rank $r = 3$.

of rank 3. The nodes of rank r are numbered $0, \ldots, 2^r - 1$, and each node of rank r is connected to the nodes of ranks $r - 1$ and $r + 1$ with the same number. Additional connections are as follows: ranks 0 and 1 are divided into two spans of equal size. The nodes of the first span of rank 0 (numbered 0 to $2^{r-1} - 1$) are connected to the second span of nodes of rank 1 (numbered 2^{r-1} to $2^{r-1} - 1$), and the nodes of the second span of rank 0 are connected to the first span of rank 1, as shown in Fig. 9. At the next level, the pattern repeats itself separately and independently in each span: the span is divided into two, and connections with the lower two corresponding spans follow the same pattern. At the last two levels, the nodes are connected in pairs: node $2i$ of level $r - 1$ is connected to node $2i + 1$ of level r, and node $2i + 1$ of level $r - 1$ is connected to node $2i$ of level r. As noted in [27, p. 221], if we "collapse" all the columns of a butterfly of rank r, then we get exactly a hypercube of dimension r.

Normal butterfly algorithms are algorithms which can be run on the butterfly network and which operate as follows: at any give time step, all the data resides in one rank of the butterfly network, and at the next time step, the data either stays in the same rank, or moves one rank up or one rank down. Movement between the ranks is only allowed along the edges of the butterfly. Two important examples of normal butterfly algorithms are the fast Fourier transform (FFT), and a modified odd-even merge-sort [27, p. 225].

A normal butterfly algorithm can be run on the hypercube in the same time as on the butterfly [27, Thm. 6.4], so in order to run it on a WECPAR, we can configure the WECPAR as a hypercube, as in the previous section, and run the algorithm on the hypercube. As noted in § 4, the connectivity required by a hypercube of dimension d is $2^{d/2} - 2$. However, using a WECPAR will require about half the connectivity due to the nature of such algorithms. Similarly to the hypercube simulation, every butterfly column will be simulated by a single PE, so the vertical connections are not needed. Additionally, at any given time step, the diagonal links used by the butterfly are only those between adjacent ranks, and these will be simulated by reconfigurable connections of the WECPAR.

Fig. 10 shows a WECPAR with nodes corresponding to a butterfly of rank 6. Every PE is numbered with the butterfly column number which it simulates. Every quadrant of the WECPAR contains consecutively numbered PEs, and the overall numbering of the PEs goes from the top-left quadrant to the top-right, then bottom-left, then bottom-right. This order is repeated recursively in each quadrant.

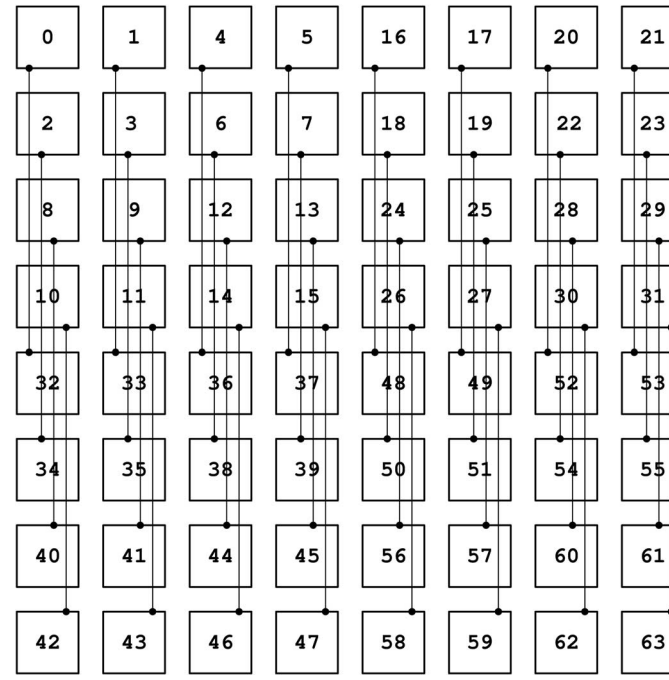

Fig. 10. WECPAR links corresponding to the diagonal edges between ranks 0 and 1 of a butterfly network of rank 6.

The figure shows connections corresponding only to the diagonal butterfly links between ranks 0 and 1: nodes 0–31 are connected respectively to nodes 32–63. For the diagonal butterfly connections between ranks 1 and 2, the WECPAR can be reconfigured so that in the upper half, nodes 0–15 are connected respectively to nodes 16–31, and similarly for the lower half. This can be repeated for any of the diagonal links in a butterfly connecting two adjacent ranks. The following theorem summarizes this simulation.

Theorem 6. *Any normal butterfly algorithm requiring a butterfly network of rank r and time $T(r)$ can be run on a $(2^{r/2}, 2^{r/2}, 2^{r/2-1})$-WECPAR in time $O(T(r))$.*

As a practical example, consider a 1024-point FFT which requires a (32,32)-WECPAR. If we configured the WECPAR as a hypercube and then run the FFT, we would need a connectivity of $k = 2^5 - 2 = 30$, but using the above dynamic reconfiguration, we only need $k = 16$.

A related concept is that of a normal hypercube algorithm, which is defined in [30, Section 3.1.3] as a hypercube algorithm in which at any given time step, only the edges corresponding to one dimension are active. It is obvious that such algorithms can be simulated by a WECPAR similarly to normal butterfly algorithms, and with the same connectivity.

We consider now two self simulation problems related to normal butterfly and hypercube algorithms. The first one is the problem of implementing the above approach on a WECPAR with a smaller connectivity. Clearly, the routing mechanism of Section 5.2 can be used for this purpose, but there is an even simpler method for this problem, as shown by the following theorem.

Theorem 7 (Connectivity reduction for normal butterfly and hypercube algorithms). *Suppose we have a (1,n,k)-WECPAR such that $n = 2^m$ and $k = 2^{m-\ell}$, where $l \geq 1$. Assume that for*

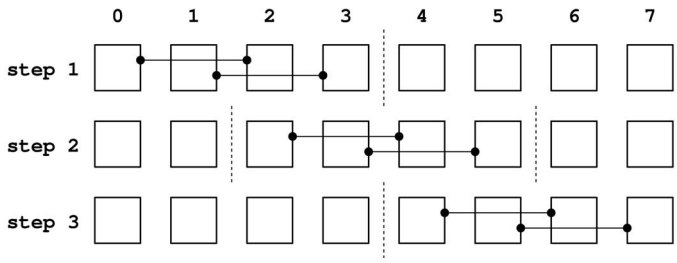

Fig. 11. Three configuration steps required to transmit messages from the first half of a WECPAR row to the second half, shown for row length $n = 8$ and connectivity $k = n/4 = 2$.

$0 \leq i \leq n/2 - 1$, $\mathrm{PE}(i)$ *wishes to transmit a message to* $\mathrm{PE}(i + n/2)$. *Then this transmission can be done in* $3^{\ell-1}$ *reconfiguration steps.*

Proof. Note first that if $k = n/2$, then a single configuration step suffices, as shown in Fig. 10. To simplify matters, we will first explain the proof for $n = 8$ and $k = n/4 = 2$, with the aid of Fig. 11.

Suppose that for $0 \leq i \leq 3$, $\mathrm{PE}(i)$ needs to send a message M_i to $\mathrm{PE}(i + 4)$. In step 1, $\mathrm{PE}(0)$ and $\mathrm{PE}(1)$ send messages M_0 and M_1 to $\mathrm{PE}(2)$ and $\mathrm{PE}(3)$, respectively. In step 2, $\mathrm{PE}(2)$ sends messages M_0 and M_2 to $\mathrm{PE}(4)$, and $\mathrm{PE}(3)$ sends messages M_1 and M_3 to $\mathrm{PE}(5)$. Now, $\mathrm{PE}(4)$ and $\mathrm{PE}(5)$ already have their respective messages, and in step 3 they transmit the remaining messages to their destination.

Thus, any reduction of ℓ by 1, requires three times as many reconfiguration steps, and the result follows by applying this approach recursively. □

Consider again the example of a 1024-point FFT, which requires a connectivity of 16. If the connectivity is reduced to 8, then only the first two stages of the algorithm require the 3-step method of Theorem 7. This theorem can thus be viewed as a type of connectivity-time tradeoff result.

The second type of self-simulation problems for normal butterfly and hypercube algorithms is that of simulating a large WECPAR by a small one. This can be done by using each PE of the small WECPAR to simulate a small square of PEs of the large WECPAR. For example, consider the problem of executing a 256-point FFT on a WECPAR of only 64 PEs (as in Fig. 10). In this case, we assign four consecutive numbers to each PE and the links would follow the same pattern as for a 64-point FFT. However, four values will now be transmitted on each link, and the lowest level operations of the algorithm are done within each PE. Hence, such self-simulations do not present any problem.

6.3 Numerical High-Performance Applications

With the advent of clusters of multi-core CPUs and GPGPUs, high performance computing (HPC) capabilities have increased very significantly in recent years. A natural question, raised by one of the reviewers, concerns the potential application of WECPAR s to HPC. We will consider two such applications: the problem of image reconstruction from projections in biomedical imaging, and certain solution methods for linear systems.

For the first problem, many parallel methods have been proposed in the literature, including methods based on parallel arrays, such as [31]. The WECPAR provides a unique approach to this problem. Assume that the (unknown) image is overlaid by a WECPAR so that every PE covers a square of pixels. In many types of these calculations, computations are done along lines passing through the image pixels; the lines model X-rays (or some other particles) passing through a slice of an object; see, for example, [32].

Assume that we have many sets of such parallel lines, and at each stage, we consider a single such set of parallel lines. A typical computation along a single line involves summation of certain values which are calculated at each pixel through which the line passes. Each PE calculates the relevant values for the pixels which it covers. For the summation, we proceed similarly to the semi-group operations in Section 6.1. To do this, we need to create long links in the horizontal, vertical and diagonal directions. As can be seen from Fig. 4, the first two directions present no problem. Furthermore, it is easy to see from this figure that long diagonal links can also be created by joining, for example, an incoming horizontal link to an outgoing diagonal link.

A somewhat different type of problem is the solution of linear systems of equations. Many parallel algorithms exist for this purpose using clusters of nodes, and in some cases, also utilizing GPGPUs. As an example of one potential application of WECPAR s to this problem, we consider a specific approach called domain decomposition (DD) [33], in which the linear system is divided into blocks of equations, which may overlap. A typical solution method iterates on the following two operations until convergence:

1. Internal operations within each block of equations.
2. Data exchange between blocks of equations.

The data exchange is due to the fact that blocks of equations correspond to neighboring subdomains and there is a need to update each subdomain with information from its neighbors. Consider now the assignment of blocks of equations to the PEs of a WECPAR. If two blocks need to exchange information, we would want to assign them to neighboring PEs. Such an assignment is simple to implement when the nonzero elements are close to the diagonal. In more complicated cases, we may need to use the routing mechanism of Section 5.2 to exchange information between blocks.

7 Discussion and Further Research

This paper presented the concept of a WECPAR, which is a reconfigurable mesh of processors, with multiple, point-to-point communications between processors. Each processor can dynamically reconfigure its switches. Some theoretical results were presented, and several practical concepts were examined, including embedding problems, algorithms, and routing. It was shown that even though certain graphs, such as the hypercube, can be embedded in a WECPAR, its dynamic reconfigurability enables normal hypercube and butterfly algorithms to be run with lower connectivity than required by a fixed embedding of the graphs.

Several research directions can be pursued, such as additional parallel algorithms, fault tolerance, and applications in various areas, such as the simulation of neural networks and cellular automata, for which WECPARs seem to be suitable. A major direction for further study is the actual implementation of a working model of the WECPAR. This, of course, will depend on the state of the technology, and additional hardware can be

added to the WECPAR, such as a bus connecting all the PEs to an external host. A mechanism for synchronization should be included so that switching between configurations is done by all the PEs at the same time.

If the WECPAR model is to become a practical computational tool, some implementation and usage issues will have to be studied. These include programming languages that will facilitate the use of the WECPAR's reconfigurability. Most probably, a WECPAR will serve as a special computational device rather than a general purpose computer. To this end, a two levels of languages are envisioned: a low-level language to support specific WECPAR operations, and a higher-level accommodation of WECPAR operations in a general purpose parallel programming language such as OpenCL [34].

Acknowledgments

The author would like to thank the anonymous reviewers for their very helpful comments and suggestions.

References

[1] I. Koren, "A reconfigurable and fault-tolerant VLSI multiprocessor array," in *Proc. 8th Annu. IEEE Symp. Comput. Architecture*, 1981, pp. 425–442.
[2] L. Snyder, "Introduction to the configurable highly parallel computer," *Computer*, vol. 15, no. 2, pp. 47–56, Feb. 1982.
[3] A. L. Rosenberg, "The Diogenes approach to testable fault-tolerant arrays of processors," *IEEE Trans. Comput.*, vol. C-32, no. 10, pp. 902–910, Oct. 1983.
[4] R. Negrini, M. G. Sami, and R. Stefanelli, "Fault tolerance techniques for array structures used in supercomputing," *Computer*, vol. 19, no. 2, pp. 78–87, Feb. 1986.
[5] F. Lombardi, D. Sciuto, and R. Stefanelli, "An algorithm for functional reconfiguration of fixed-size arrays," *IEEE Trans. Comput.-Aided Des.*, vol. 7, no. 10, pp. 1114–1118, Oct. 1988.
[6] D. Gordon, I. Koren, and G. M. Silberman, "Embedding tree structures in VLSI hexagonal arrays," *IEEE Trans. Comput.*, vol. C-33, no. 1, pp. 104–107, Jan. 1984.
[7] D. Gordon, I. Koren, and G. M. Silberman, "Restructuring hexagonal arrays of processors in the presence of faults," *J. VLSI Comput. Syst.*, vol. 2, pp. 23–35, 1987.
[8] D. Gordon, "Efficient embeddings of binary trees in VLSI arrays," *IEEE Trans. Comput.*, vol. C-36, no. 9, pp. 1009–1018, Sept. 1987.
[9] K. Bondalapati and V. K. Prasanna, "Reconfigurable computing: Architectures, models and algorithms," *Current Sci.*, vol. 78, pp. 828–837, Apr. 2009.
[10] R. Vaidyanathan and J. L. Trahan, *Dynamic Reconfiguration: Architectures and Algorithms*, vol. 13. London: Kluwer Academic/Plenum Publishers, 2004.
[11] H. Y. Youn and A. D. Singh, "On implementing large binary tree architectures in VLSI and WSI," *IEEE Trans. Comput.*, vol. 38, no. 4, pp. 526–537, Apr. 1989.
[12] M. Formann and F. Wagner, "The VLSI layout problem in various embedding models," in *Graph-Theoretic Concepts in Computer Science*, vol. 484, R. Möhring, Ed. Berlin, Germany: Springer-Verlag, 1991, pp. 130–139.
[13] D. Sitaram, I. Koren, and C. M. Krishna, "A random, distributed algorithm to embed trees in partially faulty processor arrays," *J. Parallel Distrib. Comput.*, vol. 12, no. 1, pp. 1–11, 1991.
[14] R. Heckmann, R. Klasing, B. Monien, and W. Unger, "Optimal embedding of complete binary trees into lines and grids," in *Graph-Theoretic Concepts in Computer Science*, vol. 570, G. Schmidt and R. Berghammer, Eds. Berlin, Germany: Springer-Verlag, 1992, pp. 25–35.
[15] Y.-B. Lin, Z. Miller, M. Perkel, D. Pritikin, and I. Sudborough, "Expansion of layouts of complete binary trees into grids," *Discrete Appl. Math.*, vol. 131, no. 3, pp. 611–642, 2003.
[16] A. Matsubayashi, "VLSI layout of trees into grids of minimum width," in *Proc. 15th Annu. ACM Symp. Parallel Algorithms Architectures (SPAA'03)*, 2003, pp. 75–84.
[17] T. Bartic, J.-Y. Mignolet, V. Nollet, T. Marescaux, D. Verkest, S. Vernalde, and R. Lauwereins, "Topology adaptive network-on-chip design and implementation," in *IEE Proc. Comput. Digit. Tech.*, vol. 152, no. 4, pp. 467–472, Jul. 2005.
[18] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of Network-on-Chip," *ACM Comput. Surveys*, vol. 38, no. 1, p. 1, 2006.
[19] J. Y. Hur, S. Wong, and S. Vassiliadis, "Partially reconfigurable point-to-point interconnects in Virtex-II Pro FPGAs," in *Proc. Reconfigurable Comput.: Architectures, Tools Appl.*, vol. 4419, *Third International Workshop, ARC 2007*, P. C. Diniz, E. Marques, K. Bertels, M. M. Fernandes, and J. M. P. Cardoso, Eds. Berlin, Germany: Springer-Verlag, 2007, pp. 49–60.
[20] F. G. Moraes, N. L. V. Calazans, A. V. de Mello, L. H. Moller, and L. C. Ost, "HERMES: An infrastructure for low area overhead packet-switching Networks on Chip," *Integration VLSI J.*, vol. 38, no. 1, pp. 69–93, Oct. 2004.
[21] T. Pionteck, R. Koch, and C. Albrecht, "Applying partial reconfiguration to Networks-on-Chips," in *Proc. 16th Int. Conf. Field Programmable Logic Appl. (FPL)*, Aug. 2006, pp. 155–160.
[22] H. ElGindy, H. Schröder, A. Spray, A. K. Somani, and H. Schmeck, "RMB – a reconfigurable multiple bus network," in *Proc. 2nd IEEE Symp. High-Performance Comput. Architecture (HPCA'96)*, 1996, pp. 108–117.
[23] H. Giefers and M. Platzner, "A triple hybrid interconnect for many-cores: Reconfigurable mesh, NoC and Barrier," in *Proc. Int. Conf. Field Programmable Logic Appl.*, 2010, pp. 223–228.
[24] M. M. Eshaghian-Wilner, A. Khitun, S. Navab, and K. Wang, "A nano-scale reconfigurable mesh with spin waves," in *Proc. 3rd Conf. Comput. Frontiers (CF'06)*, 2006, pp. 65–70.
[25] Z. Yu, F. Luo, B. LI, W. Zhou, L. Zong, and Q. Tao, "Reconfigurable mesh-based inter-chip optical interconnection network for distributed-memory multiprocessor system," *Optik*, vol. 121, no. 20, pp. 1845–1847, 2010.
[26] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: Freeman, 1979.
[27] J. D. Ullman, *Computational Aspects of VLSI*. Rockville, MD, USA: Computer Science Press, 1984.
[28] Z. Zhang, A. Greiner, and S. Taktak, "A reconfigurable routing algorithm for a fault-tolerant 2d-mesh network-on-chip," in *Proc. 45th Annu. Des. Autom. Conf. (DAC'08)*, 2008, pp. 441–446.
[29] Y. Ben-Asher, D. Gordon, and A. Schuster, "Efficient self-simulation algorithms for reconfigurable arrays," *J. Parallel Distrib. Comput.*, vol. 30, no. 5, pp. 1–22, 1995.
[30] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. San Mateo, CA, USA: Morgan Kaufman, 1992.
[31] D. Gordon, "Parallel ART for image reconstruction in CT using processor arrays," *Int. J. Parallel Emergent Distrib. Syst.*, vol. 21, no. 5, pp. 365–380, Oct. 2006.
[32] G. T. Herman, *Fundamentals of Computerized Tomography: Image Reconstruction from Projections*, 2nd ed. Berlin, Germany: Springer-Verlag, 2009.
[33] B. Smith, P. Bjørstad, and W. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Diffferential Equations*. Cambridge, U.K: Cambridge Univ. Press, 1996.
[34] OpenCL – the open standard for parallel programming of heterogeneous systems [Online]. Available: http://www.khronos.org/opencl.

Dan Gordon received the BSc and MSc degrees in mathematics from the Hebrew University of Jerusalem, Israel, and the DSc degree in mathematics from the Technion–Israel Institute of Technology, Haifa. He is a professor of Computer Science with the University of Haifa, Isarel, and has also taught at other universities in Israel and the US. His current research interests include scientific and parallel computing, computer graphics, and geometric computing.