Open access • Journal Article • DOI:10.1145/2629521

Graph-Based Approaches to Placement of Processing Element Networks on FPGAs for Physical Model Simulation — Source link

Bailey Miller, Frank Vahid, Tony Givargis, Philip Brisk

Institutions: University of California, Riverside, University of California, Irvine

Published on: 15 Dec 2014 - ACM Transactions on Reconfigurable Technology and Systems (ACM)

Topics: Graph embedding, Graph (abstract data type), Graph drawing, Physical system and Simulated annealing

Graph-Based Approaches to Placement of Processing Element Networks on FPGAs for Physical Model Simulation

BAILEY MILLER and FRANK VAHID, University of California, Riverside
TONY GIVARGIS, University of California, Irvine
PHILIP BRISK, University of California, Riverside

Physical models utilize mathematical equations to characterize physical systems like airway mechanics, neuron networks, or chemical reactions. Previous work has shown that field programmable gate arrays (FPGAs) execute physical models efficiently. To improve the implementation of physical models on FPGAs, this article leverages graph theoretic techniques to synthesize physical models onto FPGAs. The first phase maps physical model equations onto a structured virtual processing element (PE) graph using graph theoretic folding techniques. The second phase maps the structured virtual PE graph onto physical PE regions on an FPGA using graph embedding theory. A simulated annealing algorithm is introduced that can map any physical model onto an FPGA regardless of the model's underlying topology. We further extend the simulated annealing approach by leveraging existing graph drawing algorithms to generate the initial placement. Compared to previous work on physical model implementation on FPGAs, embedding increases clock frequency by 25% on average (for applicable topologies), whereas simulated annealing increases frequency by 13% on average. The embedding approach typically produces a circuit whose frequency is limited by the FPGA clock instead of routing. Additionally, complex models that could not previously be routed due to complexity were made routable when using placement constraints.

Categories and Subject Descriptors: B.5.2 [**Design Aids**]: Automatic Synthesis; C.3 [**Special-Purpose and Application-Based Systems**]: Real-Time and Embedded Systems

General Terms: Design, Performance

Additional Key Words and Phrases: Field programmable gate array (FPGA), cyber-physical system, physical model, differential equation, graph embedding, placement, simulated annealing

ACM Reference Format:
Bailey Miller, Frank Vahid, Tony Givargis, and Philip Brisk. 2014. Graph-based approaches to placement of processing element networks on FPGAs for physical model simulation. ACM Trans. Reconfig. Technol. Syst. 7, 4, Article 10 (December 2014), 22 pages.
DOI: http://dx.doi.org/10.1145/2629521

1. INTRODUCTION

Fast physical model simulation is required in many domains, such as biomedical engineering, physics, and chemistry. A physical model represents some observable physical phenomena, usually as a set of normal, partial differential, or ordinary differential equations, which can be solved using time-stepping equation solvers. For example,

This work was supported in part by the National Science Foundation (CNS1016792, CPS1136146) and the Semiconductor Research Corporation (GRC 2143.001).
Authors' addresses: B. Miller, F. Vahid, and P. Brisk, Department of Computer Science and Engineering, University of California, Riverside, 900 University Avenue, Riverside CA 92501; T. Givargis, School of Information and Computer Sciences, University of California, Irvine, 6210 Donald Bren Hall, Irvine CA 92697.

DOI: http://dx.doi.org/10.1145/2629521

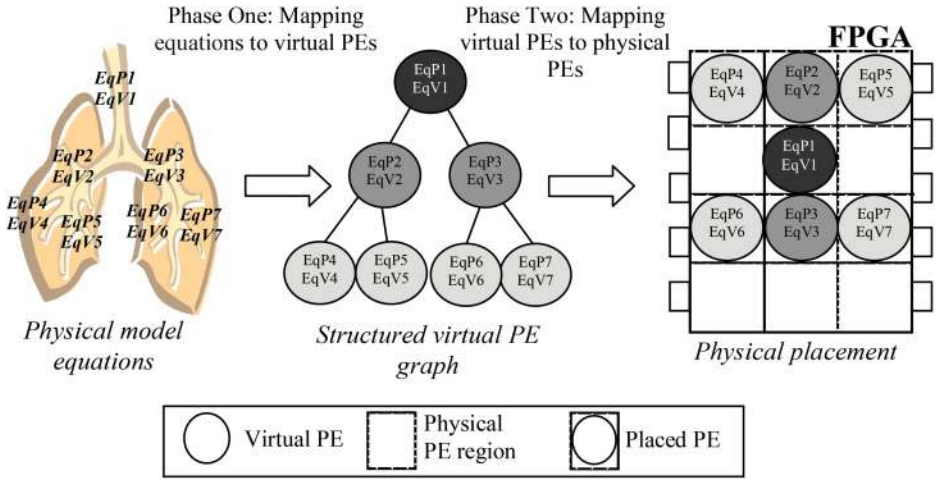

Fig. 1. Two-phase approach of mapping physical model equations onto a structured graph of virtual PEs, and mapping virtual PEs onto a FPGA utilizing graph embedding techniques.

physical models can be used to interact with and test cyber-physical devices like ventilators and pacemakers [Miller et al. 2012]. The use of physical model simulations for testing can be less expensive than testing in a physical environment that is difficult, expensive, or dangerous to create or use. Physical models may also be more accurate than physical analogues; for example, a balloon may capture some of the behavior of a lung but may not be able to accurately model various lung diseases.

Compared to a multicore desktop processor, a field programmable gate array (FPGA) can speed up physical model simulation by up to three orders of magnitude by partitioning the computation across hundreds of processing elements (PEs) [Huang et al. 2012], each of which is optimized to execute time-stepping equation solvers [Huang et al. 2011]. Many physical models share the same natural structure as the corresponding physical system. For example, a Weibel lung model [Weibel 1963] utilizes a binary tree structure that mimics lung physiology in which the trachea is the root and gas exchange occurs at the leaves. Similarly, atrial cell models utilize a three-dimensional (3D) mesh structure to simulate the propagation of electrical signals across tissues of cardiac cells [Zhang et al. 2001]. Physical system equations are naturally grouped; for example, the volume and pressure of a lung branch have data dependencies and thus should ideally be placed within the same PE to minimize communication costs. Applying this structure to a network of PEs synthesized on an FPGA naturally minimizes communication costs.

Our work leverages graph theoretic techniques to embed the topology of a physical model onto a two-dimensional (2D) mesh of PEs on an FPGA. This reduces communication cost and enables higher circuit frequencies, translating to faster execution of physical models. A secondary contribution is a simulated annealing algorithm that generates placements for physical models with irregular topologies that are not compatible with existing graph embedding algorithms.

Figure 1 details a two-phase approach for embedding a physical model onto an FPGA. The first phase maps the physical model equations to a structured virtual PE graph, which is structured in the form of the physical model: each virtual PE node contains a group of equations, along with connections to adjacent PE nodes. The second phase defines physical PE regions on the FPGA, onto which virtual PEs may be mapped, and then finds the actual placement using a graph embedding algorithm or simulated

annealing. In Figure 1, a graph embedding algorithm maps a binary tree to a 2D grid by placing the root in a physical PE region in middle of the grid and expanding child subtrees outward. Connections between PEs in the phase 1 virtual graph remain unchanged once mapped to physical regions; adjacency of physical regions does not imply connection between PEs mapped to those regions.

This work also improves the simulated annealing approach for mapping virtual PEs to physical PE regions [Miller et al. 2013], yielding two key benefits. First, many physical models have irregular topologies for which no embedding algorithms are known; simulated annealing is shown to achieve good-quality results for such topologies. Second, an embedding algorithm must be implemented for each known topology, which can be difficult and time consuming. To address these concerns, simulated annealing is shown to generate good-quality layouts that are applicable to any physical model. Additionally, we show that using graph algorithms to generate a better initial placement prior to simulated annealing results in better final placements. Furthermore, the simulated annealing algorithm has been extended with various optimizations, including modeling the suboptimality of routing across FPGA architectural features, autonormalization of timing and wiring costs, and automated initial temperature configuration.

1.1. Related Work

Our past research efforts on fast execution of physical models on FPGAs have achieved orders of magnitude of acceleration over executing on desktop processors, and several times speedups over graphical processing units (GPUs), with improvements considering time/dollar cost [Huang et al. 2011; Miller et al. 2013]. Speedup was achieved by parallelization of differential equations across hundreds of PEs for complete applications. FPGAs are an ideal substrate for physical model execution because the massively parallel local-neighbor communication inherent in these models is a good match for the spatial parallelism provided by an FPGA. In contrast, the data transfer overhead inherent in shared memory multiprocessors and GPUs is a significant impediment to performance. We introduced an automated design flow that translates a physical model specification into an equation dependency graph, partitions equations into PEs via simulated annealing, schedules computation and point-to-point data transfers, and generates a synthesizable HDL implementation of the system. PEs may be either generic computation units with an ALU and programmable instructions or a custom datapath targeted at a specific equation. Recent work has shown additional speedups by creating heterogeneous networks of general, programmable PEs, and PEs with custom datapaths for solving specific equations [Huang et al. 2012]. The reader is encouraged to refer to our earlier manuscripts for more information on the PE architecture and compiler, which cannot be included here due to space constraints. In this work, we seek to minimize the suboptimal circuit implementations yielded when routing a PE network on an FPGA by guiding the process with placement constraints.

Other relevant work includes the use of an FPGA to accelerate a heart model [De Pimental and Tirat-Gefen 2006], including interfacing of the simulation with a pacemaker via analog-digital converters, and the creation of a custom FPGA for the simulation of gene regulatory networks [Tagkopoulos et al. 2003].

The research efforts described earlier used heuristics to map equations to PEs and relied on commercial tools to place PEs. In contrast, our approach maps equations to PEs that maintain the natural structure of the physical model and embeds the structure onto a 2D grid, which is a natural coarse-grained representation of the spatial parallelism afforded by the FPGA. Our experiments demonstrate that this approach yields faster circuits with higher clock frequencies, which in turns yields faster execution of physical models.

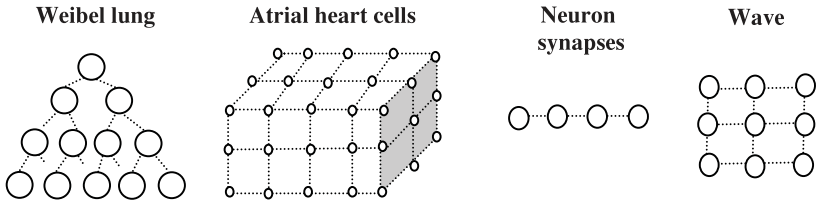

Fig. 2. Various physical models and graphs of their representative structures.

The problem of mapping algorithms with communication structures that differ from the interconnection scheme of the host architecture was first considered in the 1980s. Bokhari [1981] introduced a heuristic for mapping algorithm tasks to adjacent processors in a "finite element machine" array processor. Later, Berman and Snyder [1987] offered a general solution for embedding common structures such as cubes, meshes, linear arrays, and trees. Much of that research has been used in distributed and high-performance computing domains for mapping tasks to processors to minimize communication costs [Bhatelé and Kalé 2008]. Prior work on VLSI design has embedded a binary tree of processors in a square to reduce communication costs [Ullman 1984].

Graph embedding has also been used for FPGA placement as an alternative to iterative improvement heuristics and recursive partitioning methods. One approach is to convert a netlist into a hypergraph, which is then embedded onto a 2D grid using a recursive space-filling curve [Banerjee et al. 2009]; this approach yielded up to 2× faster runtimes for placement but did not improve critical path delay, which is critical to the speed of physical model simulation. CAPRI [Gopalakrishnan et al. 2006] creates an initial placement of a circuit by embedding the netlist into the target FPGA platform. CAPRI models the routing delays of the target FPGA in a metric space and uses matrix projections to minimize distortions between the graph representation of the netlist and the target. Both of these prior works mapped netlists onto low-level FPGA resources (LUTs, CLBs, etc.), whereas our work places networks containing hundreds of individual PEs into larger physical regions that are abstracted onto the FPGA.

2. PHYSICAL MODEL STRUCTURES

Physical models often have a natural structure associated with a corresponding layout in the physical world. Consider a human lung, which begins at the trachea and splits into nearly identical left and right lobes. Each lung contains more than 20 additional splits as the airway passage diameters decrease and eventually are able to support blood–gas exchange alveoli. The lung has thus often been modeled as a binary tree of 20 or more generations such that gas flow at the trachea can be used to compute the pressure and volume of internal branches [Weibel 1963]. Similarly, cell models of electrical activity across atrium walls use a 3D mesh structure to allow neighboring cells to propagate signals. Figure 2 shows some examples of physical models and corresponding structures, described as follows:

—*Weibel lung*: A binary tree–shaped lung model in which an inlet flow at the root of the tree is used to compute volume and pressure at lower branches. Each node of the tree computes the volume V and flow F of the corresponding branch.
—*Atrial heart cells*: A 3D mesh of cells where each cell propagates signals to its neighbors [Zhang et al. 2001].
—*Neuron synapses*: A one-dimensional (1D) array of cells that simulates the firing of neuron synapses [Terman et al. 2008].
—*Wave*: A wave model that has a 2D mesh network structure and is often used to model the propagation of sound, acoustics, and so forth [Motuk et al. 2005].

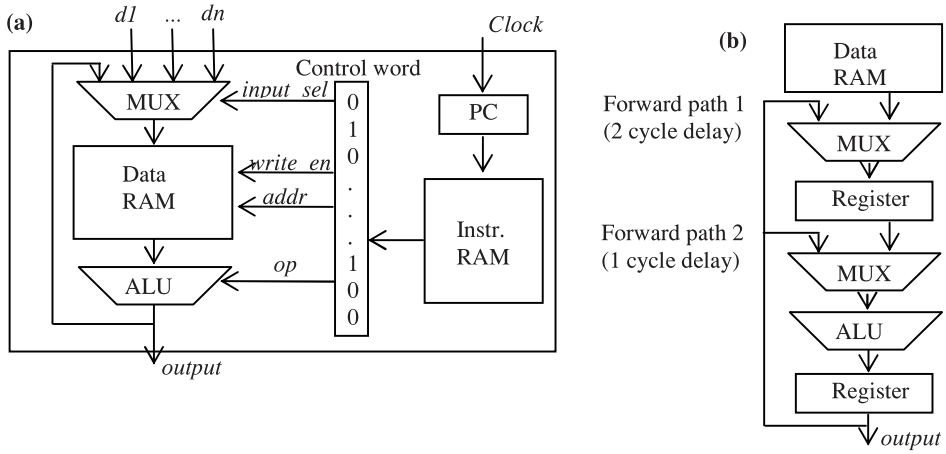

Fig. 3. (a) General PE architecture. (b) Datapath pipeline.

Large physical models such as those described earlier can be partitioned into a network of hundreds of PEs to achieve very fast simulation speeds. By maintaining the structure associated with the physical model during physical placement of PEs on an FPGA, the routing overhead between PEs can be minimized. The natural structure of a physical model typically uses an optimally minimal number and length of wires, because only local communication between cells, lung branches, and so forth, is required. Previous work in physical model simulation attempted to recover the physical model structure via heuristic annealing algorithms after having converted the specification of the physical model's equations to an equation dependency graph [Huang et al. 2011]; however, finding the globally optimal solution for physical models containing thousands of equations and hundreds of PEs is not feasible with this approach. Instead, we propose to preserve the connections as they were modeled to minimize communication cost.

3. BACKGROUND: PROCESSING ELEMENT AND NETWORK ARCHITECTURE

Earlier work proposed an architecture for a PE optimized for solving general ODEs, as described in Figure 3 [Huang et al. 2011]. A PE consists of an output port, an input port connected directly to the output of other PEs in the network, an instruction RAM containing microcoded control words, and a pipelined datapath. The number of inputs and RAM size is adjustable according to the ODEs mapped to a PE.

A program counter driven by a global clock synchronizes execution of each PE. Execution of an iteration consists of an update phase, in which data dependencies across PEs are resolved by forwarding and storing computed values, followed by a compute phase that executes arithmetic operations to solve the model for the next iteration. All computation and communication instructions are compiled statically.

Currently, we manually convert floating-point numbers into 32-bit fixed-point numbers that can be executed efficiently with the integer ALU and shift operator in the PE. We estimate the value range for each variable in the model and scale the variable accordingly. A comparison of double-precision floating point and fixed-point implementations of a Weibel 11-generation model found that the maximal relative error among all variables is within 0.5%. Investigation shows that utilizing floating-point cores would yield a 3× to 8× performance decrease and 3× to 5× increase in size; thus, we use fixed point to minimize resource utilization.

A single three-input PE has a frequency of 312MHz when implemented on a Virtex-6 VSX475T-2ff1156 using Xilinx ISE 14.2, the critical path being the decoding of the

control word output from the instruction memory. However, when a network consists of hundreds of PEs, the frequency can drop to 100MHz or lower, even becoming unroutable if there are too many data dependencies. The bottleneck in many large networks come from critical paths introduced either by the interconnections between PEs or by nonlocal placement of a PE's components. An approach to lessen the impact of inter-PE channels is to add pipeline stages between PEs, at the expense of more cycles per communication step. However, complex designs will still suffer performance degradation due to nonlocal PE component placement.

4. PHASE 1: MAPPING EQUATIONS TO VIRTUAL PROCESSING ELEMENTS

Given the specification of a physical model as a set of ordinary or partial differential equations, a map must be built that groups equations into a structured virtual PE graph G that maintains the structure of the physical model. Initially, the virtual PE graph has unconstrained size; it is then folded to fit into the available resources of the *target platform*, which is an FPGA in our case.

4.1. Partitioning Equations

Let $G = (v, e)$, where $v = \{v_1, v_2 \ldots, v_n\}$ is a set of n vertices and $e = \{e_1, e_2, \ldots, e_k\}$ is a set of k edges between vertices in v. Let $E = \{E_1, E_2, \ldots, E_m\}$ be the set of equations defined in the specification of the physical model. The set of vertices v represents virtual PEs, which may have equations from E allocated to them. The set of edges e represent communication channels between virtual PEs. If an edge $e_i = (v_j, v_k)$ exists, then there exists a dependency between the equations hosted in v_j and v_k.

The graph G and its nodes and edges are defined by the structure of the physical model. For example, a three-level binary tree–shaped Weibel lung model would have the following graph: $v = \{v_1, \ldots, v_7\}$, $e = \{(v_1, v_2), (v_1, v_3), (v_2, v_4), (v_2, v_5), (v_3, v_6), (v_3, v_7)\}$, which is a binary tree rooted at vertex v_1.

Each equation E_i can be allocated to a vertex v_i in G according to a surjective mapping function $f : E \rightarrow v$. The function f depends on the structure of G and maps groups of equations that represent the same physical element (e.g., a lung branch or atrial cell) to a single vertex. The result of applying the map function f to each equation yields a structured virtual PE graph G that maintains the basic structure of the physical model, and where each vertex (virtual PE) contains equations that represent some physical element of the physical model.

4.2. Folding

A physical model may be very large—a Weibel model with 11 generations contains more than 4,000 differential equations. To meet the physical constraints of a realistic platform when mapping virtual PEs to physical PEs, the virtual PE graph G must first be scaled down. We *fold* G by applying a homomorphic folding function φ that maps the larger graph to a smaller, more compact version, G', while preserving the structure of G. In particular, φ maps G to G', where the size n of the vertex set of G' is less than or equal to the number of supportable PEs in the target platform S; $\varphi : \mathrm{G} \rightarrow G' \mid G'_n <$ S. φ must also maintain the topology of G in G' by either maintaining an existing edge of G in G' or by merging the equations of vertex $a \in G$ into vertex $b \in G'$ such that the length of any edge connected to the merged vertices is constant.

It is generally possible to fold symmetric structures by cutting a graph into two subgraphs and merging vertices that share the same position in each subgraph [Aleliunas and Rosenberg 1982; Ellis 1991; Wagner 1990]. For example, Aleliunas and Ellis used folding to reduce the aspect ratio of rectangular graphs into forms that could be embedded onto a 2D grid. Wagner developed algorithms for folding strongly balanced hypertrees to embed them into hypercubes.

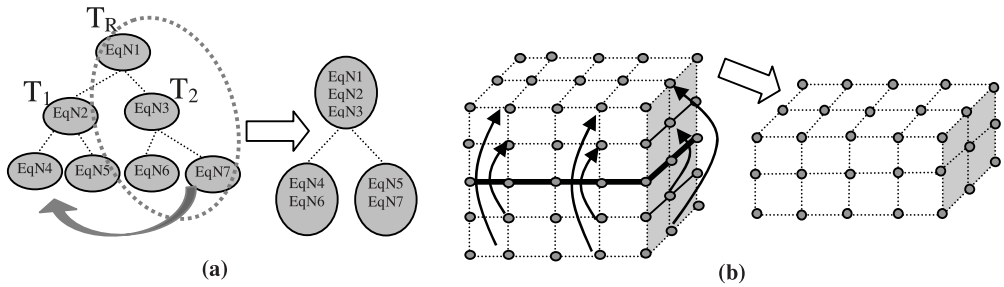

Fig. 4. Contraction of the PE dependency graph by folding: (a) binary tree and (b) 3-dimensional mesh.

The exact definition of φ depends on the physical model structure. Different physical models can reuse the same folding functions as long as their structures match; thus, a folding function for each structure type must be identified. A potential pitfall of folding is that structured virtual PE graph sizes tend to be reduced by halves. In the worst case, folding a graph that is incrementally larger than the target platform could create a situation where almost half of the physical PE regions are empty. One solution is to manually merge the last few virtual PEs in this particular case. The following section provides examples that target binary tree physical models, describing the mapping function f and folding function φ that result in the generation of a structured virtual PE graph.

4.3. Lung Model Example

A small Weibel lung model with three generations of bifurcating airways is structured as a binary tree with $2^3 - 1 = 7$ branches, or 14 interdependent differential equations for computing the pressure and volume of each branch. Let the set of equations E in the specification of the physical model be ordered such that the first l equations compute the volume and pressure of the root node, the next l equations compute the left child of the root, followed by l equations for the right child of the root, and so on. Equations can thus be initially partitioned to vertices in G via $f(e_i) = i/l$. The left side of Figure 4(a) shows a representative structured PE graph, where EqNx represents the equations allocated to each node.

Consider if the target platform for the three-generation Weibel lung model is an FPGA that contains only enough resources for three PEs. Since each vertex in the graph represents a virtual PE that must eventually be physically placed, an excess of four PEs will not fit into the device. The graph can be folded as shown in the right side of Figure 4(a) by merging nodes in such a way as to maintain the graph structure. Let T_R be the root of the graph G, and T_1 and T_2 be the subtrees whose roots are the left and right children of T_R, respectively. We fold T_2 into T_1 by traversing down each subtree simultaneously and moving any equations within the current node of T_2 into the equivalent node of T_1. The root node T_R is also merged into the root node of T_1; otherwise, T_R would contain only a single child. This method maintains the adjacency of vertices in T_2 within T_1, as long as each subtree is symmetrical. Nonsymmetrical structures can still be folded imperfectly by merging the vertices in T_2 that have no corresponding vertex in T_1 such that a minimum of additional edge length is required.

5. PHASE 2: MAPPING VIRTUAL PROCESSING ELEMENTS TO PHYSICAL PROCESSING ELEMENTS

Once a structured graph of virtual PEs has been created, each virtual PE must be mapped to a physical location on the target platform. This mapping must consider

both the average and maximum distances between PEs to reduce congestion and critical paths introduced via inter-PE communication channels. The simple solution to this problem is to let a commercial synthesis tool flatten the design hierarchy and run heuristic algorithms to select an appropriate placement. However, a circuit that contains hundreds of PEs is sufficiently complex such that modern tools cannot find good solutions without having additional constraints specified. Our approach defines a 2D grid of physical PE regions on a target FPGA platform using an XY Cartesian coordinate system. Each physical PE region in the grid contains just enough resources to implement a single PE. Virtual PEs can be mapped to physical PE regions on the grid using topology-specific graph embedding techniques or simulated annealing in our framework.

5.1. Creating the Host Architecture on an FPGA

When performing place and route operations on large PE networks using commercial tools (Xilinx ISE 13.4) and a flattened netlist, we noticed that the critical path most often occurs between memories or logic components that belong to the same PE. Each PE in our design requires two memories (BRAMs), one multiplier (DSP block), and approximately 250 lookup tables (LUTs). Intuitively, one would expect that communication between different PEs would have a much larger impact on delay. When examining the result, the commercial tools are unable to place components within the same PE nearby one another, which is an unfortunate artifact of flattening the netlist up front. Our approach effectively sidesteps this issue by creating a floor plan in which individual PEs are placed contiguously.

Relationally placed macros (RPMs) are used to establish relative distances between PE memories. RPMs have been shown to provide faster circuit designs, even with modern tools [Singh 2011]. On Xilinx FPGAs, a Cartesian coordinate system is used to specify the locations of components like DSPs and BRAMs. BRAM and DSP modules are physically located in homogeneous columns that stretch the height of the FPGA. We create an RPM for a PE using the Xilinx RLOC constraint by specifying that the offset between its instruction and data memories should be $X = 0$, $Y = 1$, and that the offset between the instruction memory and the DSP should be exactly $X = –4$, $Y = 0$. The RPM thus ensures that PE memories are placed in neighboring BRAMs within the same BRAM column, and that the related DSP module is in the closest available location in a neighboring DSP column.

RPMs are useful for ensuring the close locality of BRAM and DSP modules that belong to the same PE, but we still must constrain each PE to specific physical PE regions on the target platform. We utilize the Xilinx AREA_GROUP constraint during place and route to place PEs into physical PE regions. A selection of physical components of the FPGA (BRAM, DSPs, and slices) is first grouped into a *pblock*. We use the Xilinx PlanAhead tool to manually create pblocks in a grid structure. Each pblock contains enough resources for a PE: two BRAMs, multiple DSPs, and more than 300 LUTs. The PEs in the design netlist can then be constrained via the AREA_GROUP constraint to a specific pblock region. The use of pblocks not only designates an exact location to place a PE but also helps the place and route tools by requiring that the components in a PE hierarchy be placed within the pblock area. Since the area of the pblock is roughly what is required of a PE, the resulting PE implementation is densely packed and optimized. The use of placement constraints helps to shift the circuit critical path from internal PE connections to PE network communication channels.

We target a Xilinx XC6VSX475T. The Virtex6 platform contains approximately 297K LUTs, 2K DSP units, and 1K Block RAM (36KB each) memories. The largest grid size that can be constructed is 14×39, yielding a maximum of 504 PEs. For most physical models, 500 PEs is sufficient for much faster than real-time simulation speeds. We

note that the approach is not limited to one specific tool or vendor; all FPGAs consist of a regular, reconfigurable fabric, and most vendors allow blocks of resources to be grouped to create uniform structures. We consider only the specifically denoted FPGA and vendor (Xilinx) discussed earlier to ease the discussion.

5.2. Graph Embedding–Based Placement

Physical models that exhibit common structures are able to take advantage of graph embedding techniques during physical placement. *Graph embedding* is the process of mapping a guest graph of architecture g onto a host graph of a different architecture h. Graph embedding has been studied for at least 30 years by mathematical theorists, and many optimal solutions have been found for the embedding of structures like trees and meshes onto grids and hypercubes [Chen and Stallmann 1995; Ullman and Narahari 1990]. The typical metric by which graph embedding algorithms are evaluated is *maximum dilation*, or the maximum number of nodes that a wire may need to pass through to be completed. Since in a physical model-solving PE network the communication channels are point to point between PEs, the dilation is always exactly one. We alter the metric's definition slightly to be the max wire length between any two connected physical regions. A second important metric is the *average dilation*, or average wire length of all communication channels in the circuit.

By taking advantage of the research on graph embedding techniques to map virtual PEs to physical PEs on the target platform, the resulting physical placement can achieve smaller maximum and average dilation in the circuit. Smaller maximum dilation implies a reduction in the critical path, as once a virtual PE has been constrained to a physical region using RPMs and pblocks, the longest wires for any complex network are typically connected between different PEs (as opposed to internal PE connections). Lower average dilation means that fewer routing resources will be required, which typically results in faster circuits [Xilinx 2010].

The graph embedding problem relates to the general mapping problem, where computational tasks must be placed onto a host architecture such that communication between PEs is minimized [Berman and Snyder 1987]. Let $G_T = (V_T, E_T)$ be the guest graph, where G_T is the structured virtual PE graph (see Section 4). Let $G_H = (V_H, E_H)$, where G_H is a graph that represents the physical PE layout. V_H is a set of all physical PE regions, and E_H is initially empty because no connections exist until virtual PEs are placed. An embedding of G_T onto G_H is a result of applying an injective mapping function $\Psi_V : V_T \rightarrow V_H$ to every vertex in G_T. Once the vertex mapping has been completed and a placement is created, an additional mapping $\Psi_E : E_T \rightarrow E_H$ can be inferred automatically by creating an edge $e = (u, v) \in E_H$ for every edge $p = (l, k) \in E_T$ where $\Psi_V^{-1}(l) = u$ and $\Psi_V^{-1}(k) = v$.

The quality of the graph embedding is denoted by the average and maximum dilation of the result of applying Ψ_V and Ψ_E. Since dilation in the context of PE networks on FPGAs with point-to-point communication is wire length, we use a Euclidean distance measure. Although it is possible to measure dilation using specific FPGA routing architecture characteristics [Gopalakrishnan et al. 2006], at a macro level the distance between physical grid locations will suffice.

Embedding binary trees onto 2D grids is a solved problem [Chen and Stallmann 1995; Lee and Choi 1996; Ullman and Narahari 1990]. Embedding a binary tree onto square grids has an $O(sqrt(n))$ maximum dilation, where n is the number of generations of the tree. We utilize the H-tree construction technique (popular in VLSI) for the layout of tree architectures onto optimally sized square hosts [Ullman 1984]. H-tree construction creates an H-fractal tree where each subsequent branch of the tree alternates between horizontal and vertical tracks and wire length is halved. The graph is split recursively

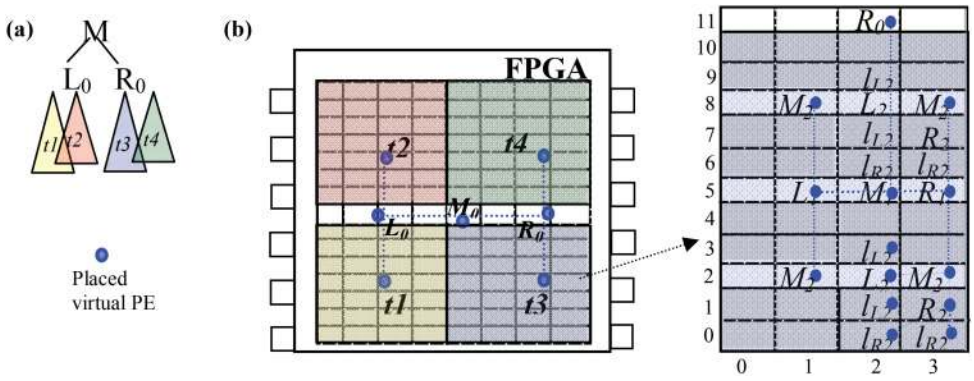

Fig. 5. Embedding 7-level binary tree into a 2D grid: (a) Initial split of 4 subtree; (b) Two additional recursive splits. White rows host root and children of a split branch. For clarity not all branches shown.

into four subtrees until leaf nodes can be placed. Where each split occurs, a track is used to host the root of the split and its two children, which are the roots of the actual four subtrees. Additional horizontal tracks are added for the three relevant parent nodes of each split subtree. Following the second split, leafs can be placed nearby their parents.

For square grids, the H-fractal tree method produces optimal results (in terms of dilation); however, for rectangular-shaped grids such as the 14 × 39 PE grid available on our target FPGA, H-tree construction requires some modifications. For example, the number of vertical tracks required for a seven-generation tree using the H-tree method is 31, or more than twice the number of available columns in the FPGA PE grid. We leverage the fact that our FPGA can route wires between PEs diagonally, as opposed to the strict row-column ordering of previous H-tree considerations [Lee and Choi 1996]. In addition, since the width of the target is the limiting factor to the number of possible recursive splits, it is not possible to maintain the ideal H-fractal shape in a rectangular grid. We therefore define a base case for the bottom k-generations of a tree that can no longer maintain H-fractal shape, such that an optimal placement of lower generations and leaf nodes can be completed. To embed the tree, we first perform placement via recursive splits down to the leaves of the tree, then perform compaction and reordering of rows to further minimize maximum wire length.

Figure 5 illustrates the process. The binary tree is split into four subtrees and assigned to a quadrant of the grid. The blue lines mark connections between physical PE regions that contain a mapped virtual PE, which are marked with blue dots. The graph embedding follows the H-tree fractal shape design until the grid becomes too narrow to maintain the shape when placing the final two generations of the tree. At that point, a base case known placement is utilized to place the remaining virtual PEs into physical PE regions with minimal wire lengths. Rows 4 and 10 contain no mapped virtual PEs, which unnecessarily inflates the maximum wire length. A simple greedy algorithm can be used to compact the graph embedding by moving the row with the longest wire until no improvement can be made.

6. SIMULATED ANNEALING–BASED PLACEMENT

Simulated annealing is a general method that can map any structured virtual PE graph onto physical PEs. This approach is useful when a physical model has no obvious structure for which a graph embedding algorithm could be used, such as unbalanced or asymmetrical trees [Gabryś et al. 2005]. Simulated annealing also yields useful comparisons to embedding by providing reasonable PE placements.

The simulated annealing approach utilizes methods previously described for timing-driven placement in the VPR tool [Marquardt et al. 2000]. We define a cost function that considers FPGA architectural features, wiring cost, and timing cost (critical path length); it is shown experimentally that our cost function correlates linearly with resulting circuit frequency. We also present a neighbor function that swaps virtual PEs based on the relative placement of connected virtual PEs. Our neighbor function provides faster convergence and results in lower cost placements than random swaps.

6.1. Cost Function

The cost function is calculated at each iteration of the simulated annealing algorithm to determine the effect of a perturbation on the current solution state. The *Wiring_Cost* term determines the cost associated with routing all of the inter-PE nets in the design. This term is a summation of all wire lengths in the design that are routed between physical PE regions:

$$Wiring\_Cost = \sum_{n=0}^{N_{NETS}} D(n_{SNK}, n_{SRC}). \tag{1}$$

$D(n_{SNK}, n_{SRC})$ is the distance of the net from source to sink, defined later. The VPlace algorithm of VPR uses a similar equation to calculate routing cost, although it uses a Manhattan distance measure and a factor to compensate for the extra resources required by highly connected nets. Minimizing the wiring cost during simulated annealing produces a placement with less congestion, resulting in faster circuit implementations [Xilinx 2010].

The *Timing_Cost* term considers the impact of the longest wires in the design, which are most likely to form a critical path in the circuit. Similar to the T-VPlace algorithm, wires are assigned a weight depending on their length [Marquardt et al. 2000]. Wires closer to the maximum wire length of the design are weighted heavily, whereas shorter wires have less impact. Note that our current implementation assumes a delay that corresponds linearly to the distance between physical PE regions; future work could achieve more accurate timing cost estimates by modeling the delay between physical PE regions in the target FPGA. Each wire is assigned a weight according to the following equation:

$$Weight(n) = 1 - \frac{Wmax - D(n_{SNK}, n_{SRC})}{Wmax}. \tag{2}$$

Wmax is the maximum PE-to-PE wire length in the design. The *Timing_Cost* for each net is calculated as follows:

$$Timing\_Cost(n) = D(n_{SNK}, n_{SRC}) \cdot Weight(n)^{W\_exp}. \tag{3}$$

W_exp is a user-defined exponent that causes higher-weighted wires to have more impact on timing cost. In T-VPlace, this exponent is called the *criticality exponent*.

FPGAs commonly contain architectural features that prevent placement of logic in specific areas. For example, Figure 9 (shown later) presents a Virtex6 FPGA that has a large gap in the middle for monitoring/programming components. Wires routed across such gaps incur an additional timing cost penalty through exponentiation of the distance by a user-defined constant *Arch_exp*. The exponentiation adds high penalties to wires that are both long and cross a gap. Whether or not a wire crosses a gap is determined by projecting a vector v from source to sink and recording intersecting physical PE regions in a set *route*. Physical PE regions that contain gaps or features restricting logic are annotated and recorded in a set *ignored*. Distance is then calculated

based on whether or not *route* and *ignored* are disjoint:

$$D(n) = \begin{cases} dist(n) & route \cap ignored = \emptyset \\ dist(n)^{Arch\_\exp} & \text{otherwise.} \end{cases} \quad (4)$$

In Equation (4), *dist(n)* is the distance measure—that is, Euclidean or Manhattan distance. The total timing cost for the PE network placement is equal to the summation of the timing cost of every net.

Furthermore, the cost function incorporates previously described autonormalization techniques to ensure that the cost of a single perturbation is related to relative changes in both the wiring and timing costs:

$$\Delta Cost = \lambda \cdot \frac{\Delta Timing\_Cost}{Previous\_Timing\_Cost} + (1 - \lambda)\frac{\Delta Wiring\_Cost}{Previous\_Wiring\_Cost}. \quad (5)$$

λ controls how much weight to give each cost term after each iteration. For all experiments in this article, we use $\lambda = 0.5$ [Marquardt et al. 2000].

Later in the article, Figure 7(a) shows a linear regression representing how the cost function relates to the resulting circuit frequency of a PE network once placed and routed.

6.2. Neighbor Function

In simulated annealing, a *neighbor function* is a local perturbation of a solution that may or may not improve its overall quality. An initial solution, which is likely to be far from optimal, can be computed randomly or use an efficient polynomial-time heuristic. A simple neighbor function in the context of our PE placement problem is to randomly select two PEs within the network and swap their locations; we use this neighbor function as a baseline.

The neighbor function presented here attempts to cluster connected PEs together, with the goal of reducing wire lengths in the process. A random physical PE region P_1 that contains a mapped virtual PEV is selected first. Each connection $e = (P_1, P_p)$ in V is evaluated, where P_p is the physical PE region of the virtual PE connected to V. A Euclidean vector is constructed from P_1 to P_p. An average of all vectors originating from P_1 identifies a physical PE region that minimizes the average wire length of all connections to the PE if the virtual PE were placed there.

If a virtual PE does not already occupy the target physical PE region, then P_1 can be placed immediately on the target. Otherwise, the algorithm examines each of the target PE's neighbors. If any neighbor is unoccupied, then P_1 is moved there. If all neighbors are occupied, the algorithm selects the physical PE region whose average connection vector endpoint is closest to P_1 for swapping.

Figure 6 provides an example. P_1 is randomly selected. An average of the two connections e_1 and e_2 yields an area of the platform where P_1 should be placed to minimize wire lengths. If there is an empty physical PE region, then P_1 is moved there. Otherwise, each physical PE region in the area is evaluated and the best candidate is swapped with P_1. The best candidate is determined by computing the potential cost reduction

6.3. Simulated Annealing Algorithm Strategies

The cooling schedule used during simulated annealing can cause dramatic differences in the obtained solution [Nourani and Andresen 1998]. We experimented with linear $[T(t) = T_0 - -\eta t]$, geometric $[T(t) = T_0/t]$, and exponential $[T(t) = T_0 a^t]$ cooling schedules. We found that both linear and geometric schedules produce a configuration with a similar cost for a given physical model, whereas the exponential schedule

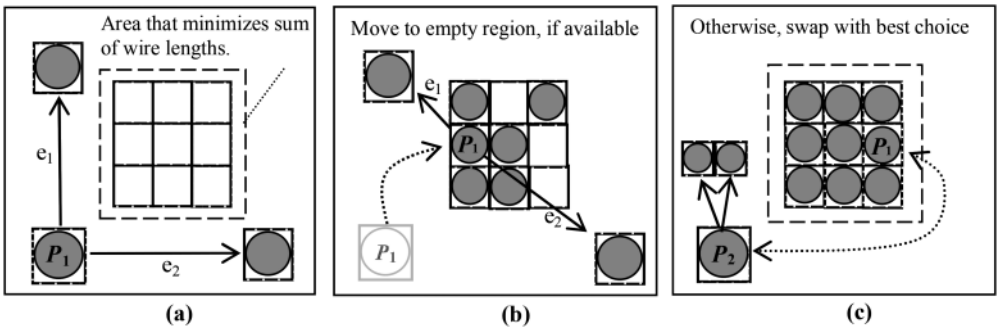

Fig. 6. The neighbor function moves PE1 to minimize wire length. (a) An area to move PE1 to is identified. (b) PE1 is moved to an empty region if available, (c) otherwise, PE1 is swapped with a PE in the area that benefits most from the swap, i.e., PE2 has neighbors near the original location of PE1.

($\alpha = 0.99$) yields a configuration that highly depends on the initial random placement and does not generally produce a good result. This is due to the quickly decaying nature of the exponential function, which makes it difficult to escape local minima in the solution space. All experiments in this article utilize a geometric cooling schedule.

Figure 6(c) shows the effect that modifying the number of solution perturbations performed per iteration has on the average final cost of five different models (Weibel10, neuron1d, neuron2d, asymmetrical tree, and random). More perturbations/iteration implies a longer runtime but with more swaps occurring at a higher temperature. Using 16 perturbations/iteration gives a 19% decrease in the final cost, on average, over 1 perturbation/iteration. The runtime of simulated annealing for a large 500-PE network using 32 perturbations/iteration is less than 10 minutes, whereas the runtime when using 1 perturbation/iteration is approximately 2 minutes.

The initial temperature is determined automatically for each run by performing trial annealing runs and searching for a temperature that results in a given acceptance ratio [Johnson et al. 1989]. In this work, we use an acceptance ratio of 0.9 for all simulated annealing runs. Additionally, a restart functionality resets the current configuration if no perturbation produces a higher-quality configuration after 250 consecutive iterations, hence the oscillations on the right-hand side of Figure 7(b) for the random neighbor function. The reset functionality helps to ensure that the annealing process does not get stuck in a local minima after having accepted a worse solution. When a reset occurs, the best configuration seen thus far is reloaded, but the anneal schedule continues without being reset. Instead of continuing along a worse path indefinitely, the algorithm can reset to a better-known configuration and continue. The number of iterations allowed before a reset is configurable—setting the number too low may ruin the hill-climbing capabilities of the algorithm, but setting too high may result in longer runtimes. The annealing process ends if 1,000 consecutive iterations do not improve the configuration.

7. EXPERIMENTS

To evaluate placement based on graph embedding to accelerate physical model simulation on FPGAs, we implemented a number of physical models of varying size on a Xilinx XC6VSX475T-2ff1156 FPGA. The physical models include a Weibel lung that is structured as a binary tree, a 1D neuron array, and a 2D grid of neurons. Each physical model is implemented using both 256 and 500 PEs. We use Xilinx ISE 13.4 software to synthesize and implement VHDL descriptions of the PE networks for all experiments. Note that to implement the 10-generation Weibel model, we use 500 physical PEs.

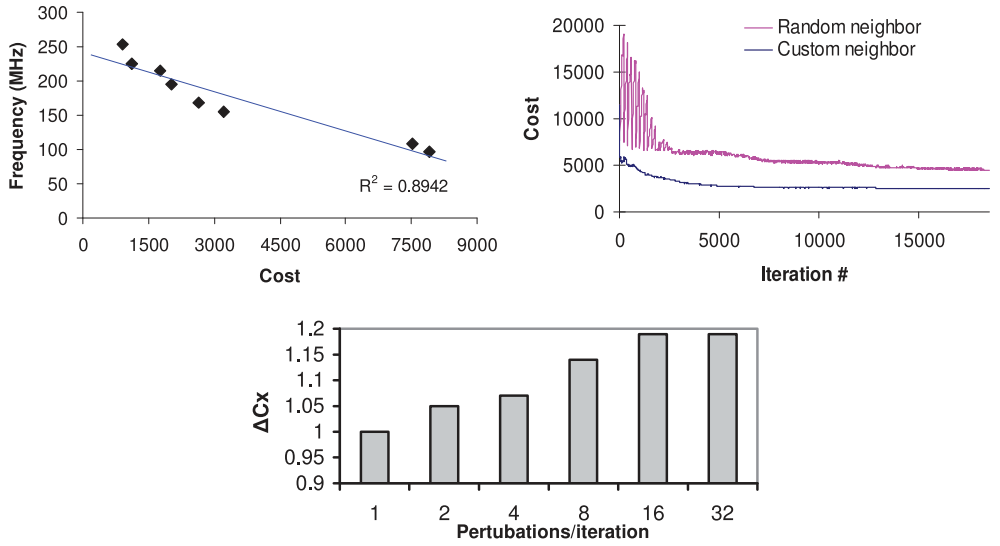

Fig. 7. (a) Cost function correlates with resulting circuit frequency. (b) Comparison of random swaps to the custom neighbor function (cost is equivalent at iteration 0). (c) Average effect of changing the number of perturbations per iteration (runtime) on final cost. ΔCx is the improvement over 1 perturbation/iteration.

Recall that the target platform is constrained to 504 physical PEs. We fold the Weibel model to a structured virtual PE graph of 512 nodes and then manually merge a few of the leaves until the size constraint is met. The alternative would have been to continue folding the structured virtual PE graph until the size constraint is met, which would result in 256 virtual PEs, leaving around 50% of the available resources unused.

For each physical model, we implemented three methods of placement for the PE networks. The first method utilizes the compiler from previous work to partition the physical model equations to PEs and generate a custom communication network [Huang et al. 2012]; no constraints are used to map the PEs to specific physical PE regions, and we rely on the Xilinx tools to place and route PEs onto the target platform. The second method first creates a structured graph of virtual PEs, folds it to fit FPGA platform constraints, and then utilizes the simulated annealing approach of Section 6 to map virtual PEs to physical regions; as discussed previously, we utilize a geometric cooling schedule and run the annealing using $\lambda = 0.5$ and an initial temperature acceptance ratio of 0.9. The third method creates a structured virtual PE graph of the physical model, folds it to fit the FPGA target platform size constraint, and then places the physical model using a topology-specific graph embedding algorithm. The Weibel model uses an H-tree graph embedding as described earlier. The 1D neuron model is a linear array of 6,400 neurons, thus the graph embedding that is used places PEs into rows and connects the rows at the edges to form a Hamiltonian path among all PEs. The 2D neuron model consists of a 2D 64 $\times$ 64 mesh of neurons, where each neuron is connected to at most four neighbors. The embedding for the 2D neuron model is a direct mapping onto the 2D grid of physical PE regions after the original physical model is folded.

7.1. Results

Figure 8 shows the resulting circuit frequencies of implementing PE networks on an FPGA with the four different techniques. The first and second columns use the equation partitioning performed by the PE network compiler. The third and fourth columns use

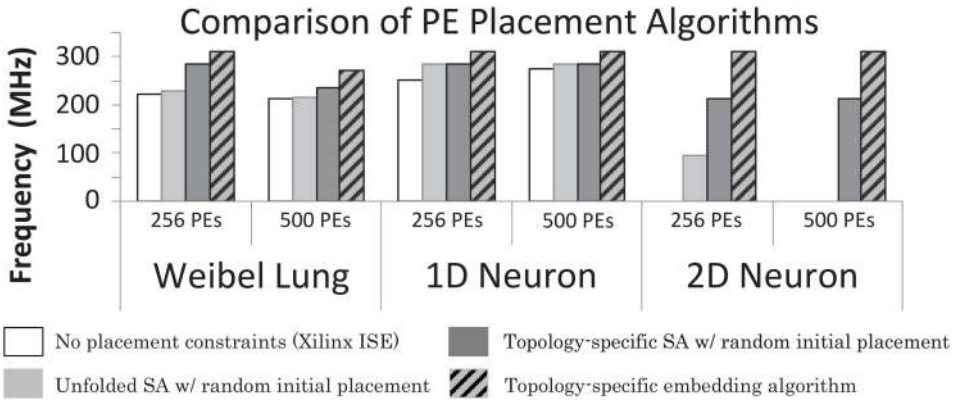

Fig. 8. Frequencies of PE networks. Missing results could not be implemented by Xilinx ISE.

a partitioning based on the structure of the model, and the corresponding graphs were folded to meet target platform constraints.

The first columns do not use physical placement constraints. These data points represent the ability of Xilinx ISE to place and route the PE network compiler-generated circuit without any guidance. The second columns use the same RTL as the first but with placement constraints generated via simulated annealing that map virtual PEs to physical PE regions. The difference between the two columns in each case show that applying placement constraints through an automated simulated annealing process can provide some marginal improvement and is even able to route a network that Xilinx was unable to do without constraints.

The third and fourth columns use a partitioning of equations based on the model's structure. The third column uses simulated annealing and the fourth column uses a topology-specific graph embedding algorithm to generate the placement. The graph embedding approach is almost always able to produce a circuit that tops 300MHz. The ceiling for the circuit frequency in a PE network is approximately 310MHz for the selected platform. We determined the ceiling by placing and routing a circuit with a single PE and evaluating the critical path of the internal datapath. It is not possible for a network of PEs to operate faster than the ceiling, and any decrease in performance can be attributed to critical paths introduced by inter-PE connections. The graph embedding approach is typically able to minimize the critical path length and provide placements that allow the circuit to approach the ceiling. The only embedding example that could not reach the ceiling of 310MHz is the 10-generation Weibel lung model using 500 PEs. Because the 2D grid of the physical PE regions is narrow, an optimal embedding of the tree cannot occur. Wire lengths between successive generations are longer, resulting in longer critical path delays.

Missing columns indicate that the Xilinx tools were not able to place and route the design due to high congestion. The compiler that partitioned the equations and created the communication network could not sufficiently reduce the data dependencies between PEs for these large physical models, resulting in an overwhelming number of wires in the network. Note that these designs are routable if we use either a topology-specific simulated annealing or graph embedding approach.

7.2. A Look Inside the FPGA

Figure 9 depicts the placement of the first few generations of a nine-generation Weibel model on 256 PEs, as captured by the Xilinx PlanAhead tool. An overlay of nodes and

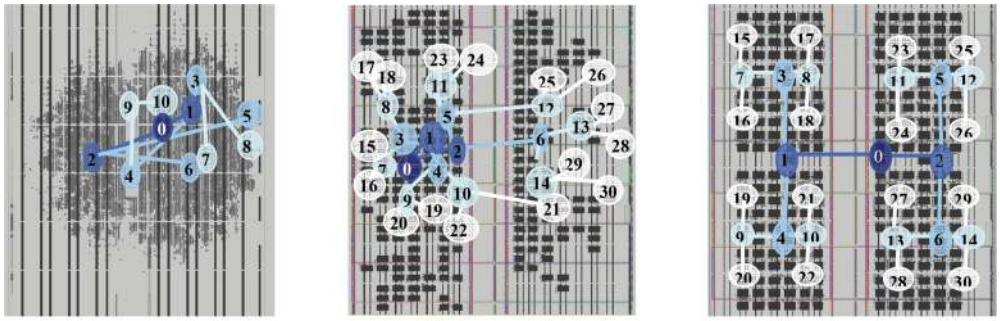

Fig. 9. Different placements of a 256 PE Weibel lung model. (a) Unconstrained placement of PEs performed by Xilinx ISE; (b) Simulated annealing; (c) Graph embedding.

connections shows where virtual PEs have been mapped onto the FPGA. Figure 9(a) shows how Xilinx ISE implements the PE network in the absence of additional constraints that map to specific physical regions. Due to the complexity of the circuit, the resources of a single PE can be spread over a wide area; thus, we have marked only the approximate central location of the first four generations of the left subtree of the graph. Note that if we do not specify placement constraints, the tool places PEs at suboptimal locations, yielding potentially long wire distances between PEs. For example, the wires between node two and its children five and six span more than halfway across the entire design.

Figure 9(b) depicts the placement produced by the simulated annealing algorithm. Each black block indicates a virtual PE that has been mapped to a physical PE location. An empty space is a physical PE region onto which no virtual PE was mapped. As a consequence of simulated annealing, nodes that share connections tend to be grouped together, whereas the overall tree tends to expand outward from the center of the grid toward leaf nodes grouped on the perimeter.

Figure 9(c) shows the tree embedded in the host grid using a topology-specific algorithm. The center of many common (Xilinx) FPGAs contains immutable logic, and minimization of the routing across the center is desired. This embedding requires a single wire across the gap, at the second generation of the tree.

We also measured the static and dynamic power of each case using the Xilinx XPower Analyzer. The unconstrained placement uses approximately 20% less absolute dynamic power on average than both the simulated annealing and embedding constrained placement approaches. The Xilinx ISE options for power reduction during circuit implementation were disabled for every reported experiment, thus timing is the driving optimization goal.

8. PLACEMENT OF NONSTRUCTURED PHYSICAL MODELS

Thus far, the experimental evaluation has been limited to structured physical models with topologies for which embedding algorithms are known. For example, the H-tree embedding for binary trees is not immediately translatable to assymmetric trees, which are representative of lung blood circulatory models [Horsfield et al. 1982; Gabryś et al. 2005]. Furthermore, some physical models are fully irregular. For example, a spiking neural network model, which simulates a human brain cortical network, consists of interconnected groups of neurons with statistically generated connectivity [Nageswaran et al. 2009]. Such models result in random, commonly nonplanar graph structures that make embedding difficult, if not impossible. Furthermore, if the structure is not clear, then a structured virtual PE graph cannot be created. We refer to models that are not easily embedded into the 2D FPGA host grid as *nonstructured models*.

Simulated annealing can generate placement constraints for nonstructured models, thereby improving circuitry frequency compared to placement using commercial tools with unconstrained placement. In this section, we describe improvements to the simulated annealing algorithm that provide better solutions with increased clock frequency for unstructured models.

8.1. Using Graph Drawing Algorithms to Generate Initial Placement

As described previously, our simulated annealing placer uses a randomly generated initial placement as an initial solution. To improve the overall quality of results, we introduce a heuristic to generate a lower cost initial solution compared to a random placement. One of the benefits of simulated annealing algorithms is that good solutions can be found even from random, high entropy initial configurations. However, by starting with a good placement that is assumed to be close to an approximately global optimal solution, the annealing process can focus on performing target-specific optimizations and find a good solution in less time. Decades of research into graph drawing techniques has produced algorithms that attempt to minimize edge distances, edge intersections, and the area of drawing size [Fruchterman et al. 1991]. Such techiques can be leveraged to generate an initial layout of network PEs on the FPGA's 2D grid, which has substantially less cost compared to a random initial layout. Previous research has shown that, on average, a better-than-random starting solution can yield better final solutions [Johnson et al. 1989].

We extended our previously introduced PE network compiler to use graph drawing algorithms to generate an initial placement. The approach uses the graph drawing and visualization tools Graphviz and/or Tulip to generate a layout of an existing, possibly folded, virtual PE network. Graphviz and Tulip are freely available open source projects. Given a DOT-formatted input file that describes the nodes (PEs) and edges (wires) of the graph, the output of a graph drawing tool is the virtual PE network with *X, Y* coordinates assigned to each node. The placed graph is then transposed onto the FPGA 2D grid by normalizing the aspect ratio of the graph with that of the grid. Graph nodes are placed onto physical PE regions based on their normalized *X, Y* coordinates from the placed graph output of Graphviz. Figure 10(a) illustrates the process of mapping Graphviz output to the FPGA.

The result of the initial mapping may be an invalid configuration, which may occur if more than one virtual PE is mapped to the same physical PE region. The normalization process shrinks a graph layout down to fit the FPGA; sometimes, several virtual PEs may overlap. In such scenarios, a process, shown in Figure 10(b), can legalize the configuration. Our solution evaluates each physical PE region in turn to determine if it contains multiple virtual PEs. If so, then all but one must be relocated to empty physical PE regions. Ideally, they will be relocated to nearby empty regions to maintain the placement generated by the drawing algorithm as much as possible. Our approach is to search an expanding radius of neighbor regions until an empty region is found. Neighbors with a distance of 1 are searched first (shaded lightly in Figure 10(b)), then a distance of 2 (dark shading), and so forth. Thus, overlapping virtual PEs are moved a minimum distance from their original location, and an empty region is guaranteed to be found, provided that $num\_PEs <= num\_regions$. Other approaches besides the presented greedy algorithm are possible.

8.2. Graph Drawing Algorithms

Many graph drawing algorithms exist, and each may produce a different layout of the same graph. Various categories of graph drawing algorithms exist, including force-directed, orthogonal or planar, layered, and *tree* layout strategies.

As shown in Figure 11, different graph drawing algorithms give different layouts. Force-based algorithms, like neato and Frutcherman-Reingold, give reasonable layouts

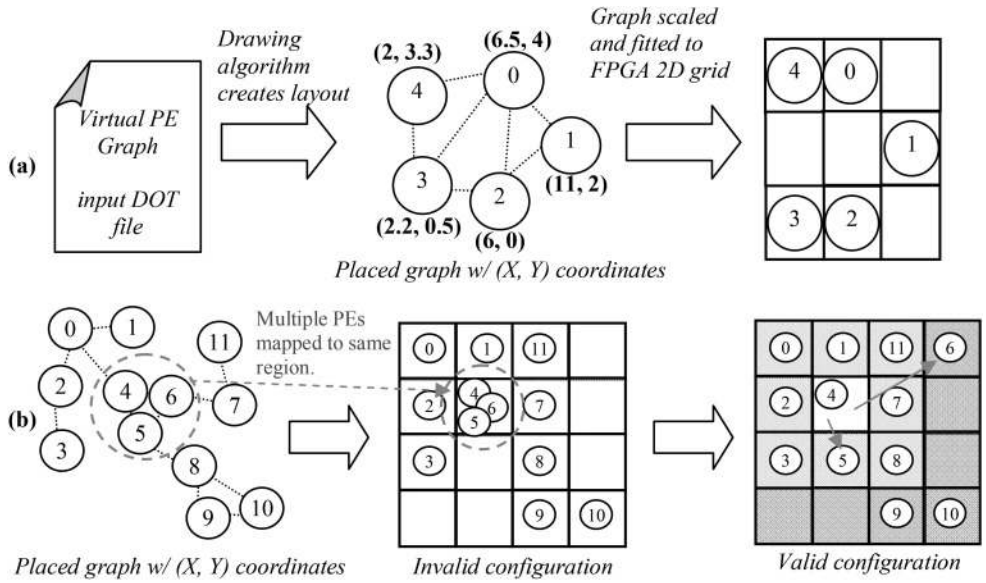

Fig. 10. (a) Using Graphviz to generate an initial layout of a non-structured graph, and (b) creating a valid configuration by moving overlapping virtual PEs to empty physical PE regions by checking increasingly distant neighboring regions.

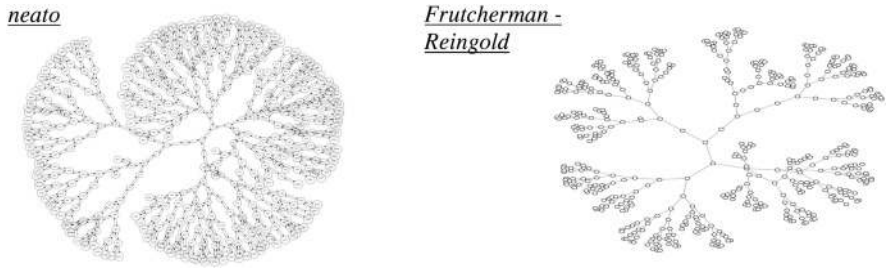

Fig. 11. Result of applying two different graph drawing algorithms to the same graph.

for almost any model. Such algorithms utilize their own heuristics to converge on a good solution that may not be optimal but will at least be within some local minima with reduced edge lengths and better layout than a random approach.

In contrast to force-based drawing algorithms, other algorithms are specific to certain structures or graph types. Heirarchical and tree-based algorithms perform best if the model has a specific structure that matches the algorithm. Often, such algorithms work *only* on graphs of a specific structure; for example, a tree drawing algorithm may fail when trying to draw a graph with cycles.

8.3. Placement Using Graph Drawing Evaluation

Table I shows initial and final costs of five different graph drawing algorithms applied to structured and nonstructured models. The cost is calculated by the equation presented in Section 6, which considers the wire length total, critical path, and architectural features of the target platform (Xilinx Virtex 6). We implement and compare the force-based algorithms neato, fdp, and Frutcherman-Reingold (FR); the heirarchical algorithm Sugiyama (Sg); and the radial layout algorithm circo. Each reported

Table I. Initial and Final Cost of Various Graph Drawing Algorithms for Both Structured and Unstructured Models
Initial cost is based on the drawing algorithm output. Final cost is calculated after mapping PEs to physical regions using simulated annealing, with the drawing algorithm output as the initial configuration. "N/A" means that the graph drawing algorithm could not produce a layout.

Structure (No. of PEs)	Initial Cost						Final Cost after Simulated Annealing					
Drawing algorithm	rand	neato	fdp	FR	Sg	circo	rand	neato	fdp	FR	Sg	circo
	Structured graphs											
bitree(255)	*11500*	*1433*	***1128***	*1389*	*20256*	*4847*	*1028*	*944*	***920***	*1052*	*9028*	*990*
bitree(500)	19496	*9005*	***8073***	*10282*	*11782*	*9006*	*1917*	*1702*	***1606***	2848	*2455*	*1746*
linear(256)	*10335*	***899***	*1045*	*1280*	*1785*	*2529*	*861*	***704***	*759*	*1019*	*905*	*837*
linear(500)	*19366*	*6047*	*8278*	*8415*	***4784***	*8157*	*1600*	*1355*	*1366*	*2651*	*1582*	***1319***
mesh(256)	*29832*	***8386***	*8466*	*9082*	*19496*	*18204*	*8778*	*7728*	***7715***	*8329*	*9996*	*8629*
mesh(500)	*96K*	*73K*	*69K*	*67K*	*N/A*	***65K***	*63K*	*61K*	***58K***	*60K*	*N/A*	*63K*
	Unstructured graphs											
a_tree(500)	*19806*	*6836*	*9242*	*7833*	*13079*	***5931***	*2697*	*1981*	*2423*	*2353*	*2664*	***1875***
noStrc(300)	*9440*	***2329***	*2379*	*2490*	*3854*	*5249*	*2290*	*1950*	***1905***	*2060*	*2153*	*2117*
noStrc(500)	*13135*	*6454*	*6124*	*7574*	*8007*	***5355***	*3346*	***2713***	*2865*	*2923*	*3500*	*2894*
Avg. impr. over random (X)	–	***4.31***	*4.22*	*3.70*	*2.40*	*2.41*	–	***1.17***	*1.14*	*0.95*	*0.85*	*1.12*

result is the average of five simulated annealing runs for each configuration. We establish initial temperatures using an acceptance ratio of 0.9, although a lower acceptance ratio might better preserve the initial placement seed yielded by the graph drawing algorithm.

Runtimes of the simulated annealing algorithm vary with network size and connectivity. The minimum runtime was 36 seconds for the 256 PE binary tree model; the maximum was about 6 minutes for the 2D neuron model. The runtime of the Xilinx tools is not affected by the placement constraints, requiring 1 to 3 hours depending on the model. In contrast, implementing an embedding algorithm can take hours to days depending on the complexity of the structure. The simulated annealing approach thus provides a faster and more generalized method for creating faster circuits automatically without requiring structure-specific implementations.

Every drawing algorithm, except for the heirarchical algorithm Sugiyama (Sg), produces a layout with lower cost than a random mapping. Sugiyama creates very wide layouts with large aspect ratios with most nodes at the bottom, which makes for poor starting points to the annealing process.

The force-based algorithms neato and fdp generate the best initial layouts, reducing the initial cost compared to random by an average of 4.31× and 4.22×. In two cases, circo produces the best initial layout. Circo produces layouts similar to H-tree embedding for tree-like acyclic graphs. The PE network compiler may generate networks that recapture some of the tree structure of the original model, thus the initial layout resembles an H-tree.

The higher-quality initial layouts that are generated by drawing algorithms result in a lower final costs on average for the neato, fdp, and circo algorithms. The fdp and neato algorithms produce final costs that, on average, are 17% and 14% lower.

The placement constraints generated by the best-performing drawing algorithm for each model were used during an implementation of the circuit; the results are reported in Figure 12. The first three columns do not use a graph folding technique to capture the model structure. Instead, a PE network compiler produced the architecture. In contrast, the topology-specific columns had equations partitioned according to the model structure, which typically reduces the number of wires in the circuit and results in less congestion during place and route. The embedding approach achieves close to optimal implementations because the embedding placement migrates the

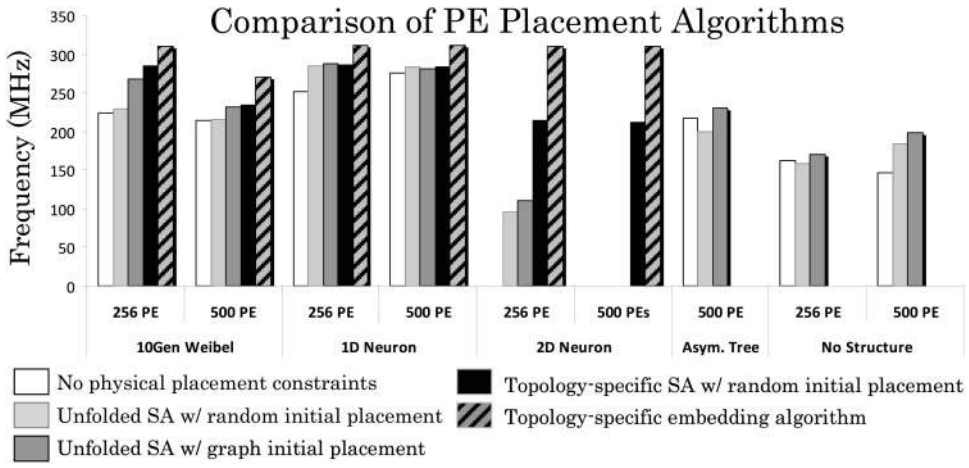

Fig. 12. Frequency of FPGA circuits for various models and techniques. Missing results could either not be routed by Xilinx ISE or are not applicable (e.g., topology-specific results for unstructured models).

critical path from intranetwork PE-to-PE communication channels to intra-PE logic. The three right-most models, which are nonstructured models (asymetrical tree counts as nonstructured because there is no simple embedding), lack folded or embedded results because such models cannot use those approaches. The largest model, neuron2d(500), could not not be implemented by any unfolded approach due to the high number of wires in the design. The neuron2d(256) results could only be routed if placement constraints were used. In some cases where the connectivity of the network is low, Xilinx is able to perform place and route effectively without placement constraints. The described approaches have the most benefit for larger, more connected structures.

The third column in Figure 12 shows the frequency of the circuit when using the graph drawing algorithm approach on an unfolded graph. The second column shows a random initial layout approach. Comparing these two results for each model yields the improvement in circuit frequency due to better initial placement using graph drawing algorithms, which is an average of 9% higher frequency. The left-most columns show no placement constraints. Comparing "No physical placements constraints" with "Unfolded SA w/graph initial placement" yields a 13% average improvement in circuit frequency due to placement constraints and initial graph drawing layouts, even when not considering model-specific structure.

9. CONCLUSION

We have presented an approach for fast physical model simulation on FPGAs that makes use of the physical model's structure to improve performance. The first phase of the approach maps physical model equations to a structured virtual PE graph and groups related equations. The second phase of the approach maps the structured virtual PE graph to a 2D grid of FPGA physical regions by using either a graph embedding or a simulated annealing technique. For models with an embeddable structure, the graph embedding and simulated annealing techniques provide 25% and 13% average improvements in circuit frequencies compared to placements that do not map to specific physical regions. Additionally, some complex circuits that could not previously be implemented without placement constraints are made routable. Nonstructured models can also utilize the simulated annealing approach to generate placement constraints and achieve better timing. We also introduced a technique that utilizes graph drawing algorithms to generate the initial annealing placement. Utilizing only the graph

drawing and simulated annealing approach without considering model structure yields an average of 13% improvement in circuit frequency. Future work includes examining other embedded processing platforms to examine trade-offs among performance, cost, power, and design time.

REFERENCES

Romas Aleliunas and Arnold Rosenberg. 1982. On embedding rectangular grids in square grids. *IEEE Transactions on Computers* 31, 9, 907–913. DOI:http://dx.doi.org/10.1109/TC.1982.1676109

Pritha Banerjee, Susmita Sur-Kolay, Arijit Bishnu, Sandip Das, Subhas C. Nandy, and Subhasis Bhattacharjee. 2009. FPGA placement using space-filling curves: Theory meets practice. *ACM Transactions on Embedded Computing Systems* 9, 2, Article No. 12. DOI:http://doi.acm.org/10.1145/1596543.1596546

Francine Berman and Lawrence Snyder. 1987. On mapping parallel algorithms into parallel architectures. *Journal of Parallel and Distributed Computing* 4, 5, 439–458. DOI:http://dx.doi.org/10.1016/0743–7315(87)90018–9

Abhinav Bhatelé and Laxmikant V. Kalé. 2008. Benefits of topology aware mapping for mesh interconnects. *Parallel Processing Letters* 18, 4, 549–566.

Shahid H. Bokhari. 1981. On the mapping problem. *IEEE Transactions on Computers* 30, 3, 207–214.

Woei-Kae Chen and Matthias F. M. Stallmann. 1995. On embedding binary trees into hypercubes. *Journal of Parallel and Distributed Computing* 24, 2, 132–138. DOI:http://dx.doi.org/10.1006/jpdc.1995.1013

John A. Ellis. 1991. Embedding rectangular grids into square grids. *IEEE Transactions on Computers* 40, 1, 46–52.

Thomas MJ Fruchterman and Edward M. Reingold. 1991. Graph drawing by force-directed placement. *Software: Practice and Experience* 21, 11, 1129–1164.

Elżbieta Gabryś, Marek Rybaczuk, and Alicja Kędzia. 2005. Fractal models of circulatory system. Symmetrical and asymmetrical approach comparison. *Chaos, Solitons & Fractals* 24, 3, 707–715.

Padmini Gopalakrishnan, Xin Li, and Lawrence Pileggi. 2006. Architecture-aware FPGA placement using metric embedding. In *Proceedings of the 43rd Annual Design Automation Conference (DAC'06)*. ACM, New York, NY, 460–465. DOI:http://doi.acm.org/10.1145/1146909.1147033

Keith Horsfield, Wendy Kemp, and Sally Phillips. 1982. An asymmetrical model of the airways of the dog lung. *Journal of Applied Physiology* 52, 1, 21–26.

Chen Huang, Bailey Miller, Frank Vahid, and Tony Givargis. 2012. Synthesis of custom networks of heterogeneous processing elements for complex physical system emulation. In *Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'12)*. ACM, New York, NY, 215–224. DOI:http://doi.acm.org/10.1145/2380445.2380483

Chen Huang, Frank Vahid, and Tony Givargis. 2011. A custom FPGA processor for physical model ordinary differential equation solving. *IEEE Embedded Systems Letters* 3, 4, 113–116.

David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch, and Catherine Schevon. 1989. Optimization by simulated annealing: An experimental evaluation. Part I: Graph partitioning. *Operations Research* 37, 6, 865–892.

Alexander Marquardt, Vaughn Betz, and Jonathan Rose. 2000. Timing-driven placement for FPGAs. In *Proceedings of the ACM/SIGDA 8th International Symposium on Field Programmable Gate Arrays (FPGA'00)*. ACM, New York, NY, 203–213. DOI:http://doi.acm.org/10.1145/329166.329208

Sang-Kyu Lee and Hyeong-Ah Choi. 1996. Embedding of complete binary trees into meshes with row-column routing. *IEEE Transactions on Parallel and Distributed Systems* 7, 5, 493–497. DOI:http://dx.doi.org/10.1109/71.503774

Bailey Miller, Frank Vahid, and Tony Givargis. 2012. Digital mockups for the testing of a medical ventilator. In *Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium (IHI'12)*. ACM, New York, NY, 859–862. DOI:http://doi.acm.org/10.1145/2110363.2110473

Bailey Miller, Frank Vahid, and Tony Givargis. 2013. Embedding-based placement of processing element networks on FPGAs for physical model simulation. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (*FPGA'13).* ACM, New York, NY, 181–190.

Erdem Motuk, Roger Woods, and Stefan Bilbao. 2005. Implementation of finite difference schemes for the wave equation on FPGA. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'05)*. IEEE, Los Alamitos, CA, 237–240. DOI:http://dx.doi.org/10.1109/ICASSP.2005.1415690

Jayram Moorkanikara Nageswaran, Nikil Dutt, Jeffrey L. Krichmar, Alex Nicolau, and Alexander V. Veidenbaum. 2009. A configurable simulation environment for the efficient simulation of large-scale

spiking neural networks on graphics processors. *Neural Networks* 22, 5, 791–800. DOI:http://dx.doi.org/10.1016/j.neunet.2009.06.028

Yaghout Nourani and Bjarne Andresen. 1998. A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General* 31, 41, 8373–8385.

Julio C. G. De Pimentel, and Y. G. Tirat-Gefen. 2006. Hardware acceleration for real time simulation of physiological systems. In *Proceedings of the 28th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS'06)*. IEEE, Los Alamitos, CA, 218–223. DOI:10.1109/IEMBS.2006.260298

Satnam Singh. 2011. The RLOC is dead—long live the RLOC. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'11)*. ACM, New York, NY, 185–188.

Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. 1981. Methods for visual understanding of hierarchical system structures. *IEEE Systems, Man, and Cybernetics* 11, 2, 109–125. DOI:10.1109/TSMC.1981.4308636

Ilias Tagkopoulos, Charles Zukowski, German Cavelier, and Dimitris Anastassiou. 2003. A custom FPGA for the simulation of gene regulatory networks. In *Proceedings of the 13th ACM Great Lakes Symposium on VLSI (GLSVLSI'03)*. ACM, New York, NY, 132–135. DOI:http://doi.acm.org/10.1145/764808.764843

David Terman, Sungwoo Ahn, Xueying Wang, and Winfried Just. 2008. Reducing neuronal networks to discrete dynamics. *Physica D: Nonlinear Phenomena* 237, 3, 324–338.

Jeffrey D. Ullman. 1984. *Computational Aspects of VLSI*. W. H. Freeman & Co., New York, NY.

Stuart Ullman and Bhagirath Narahari. 1990. Mapping binary precedence trees to hypercubes and meshes. In *Proceedings of the 2nd IEEE Symposium on Parallel and Distributed Processing*. 838–841. DOI:10.1109/SPDP.1990.143655

Alan S. Wagner. 1990. *Embedding All Binary Trees in the Hypercube*. Technical Report. University of British Columbia, Vancouver, BC, Canada.

Ewald Weibel. 1963. *Morphometry of the Human Lung*. Springer-Verlag, Berlin, Germany.

Xilinx Inc. 2010. *Virtex-6 FPGA Routing Optimization Design Techniques*. Retrieved Sept 1, 2013, from http://www.xilinx.com/support/documentation/white_papers/wp381_V6_Routing_Optimization.pdf.

Henggui Zhang, Arun V. Holden, and Mark R. Boyett. 2001. Gradient model versus mosaic model of the sinoatrial node. *Circulation* 103, 4, 584–588.

Received September 2013; revised January 2014; accepted February 2014