# Embedding-Based Placement of Processing Element Networks on FPGAs for Physical Model Simulation

Bailey Miller
Dept. of Computer Science and Engineering
University of California, Riverside
bmiller@cs.ucr.edu

Frank Vahid
Dept. of Computer Science and Engineering
University of California, Riverside
Also with CECS, UC Irvine
vahid@cs.ucr.edu

Tony Givargis
Center for Embedded Computer Systems (CECS)
University of California, Irvine
givargis@uci.edu

## ABSTRACT

Physical models utilize mathematical equations to model physical systems like airway mechanics, neuron networks, or chemical reactions. Previous work has shown that physical models can execute fast on FPGAs (field-programmable gate arrays). We introduce an approach for implementing physical models on FPGAs that applies graph theoretic techniques to make use of a physical model's natural structure—tree, ring, chain, etc.—resulting in model execution speedups. A first phase of the approach maps physical model equations to a structured virtual PE (processing element) graph using graph theoretic folding techniques. A second phase maps the structured virtual PE graph to physical PE regions on an FPGA using graph embedding theory. We also present a simulated annealing approach with custom cost and neighbor functions that can map any physical model onto an FPGA with low wire costs. Average circuit speedup improvements over previous works for various physical models are 65% using the graph embedding and 35% using the simulated annealing approach. Each approach's more efficient use of FPGA resources also enables larger models to be implemented on an FPGA device.

## Categories and Subject Descriptors

B.5.2 [**Design Aids**]: Automatic synthesis
C.3 [**SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS**]: Real-time and embedded systems

## Keywords

Real-time emulation, field-programmable gate array (FPGA), ordinary differential equations, physical models, cyber-physical systems, differential equation synthesis, high-level synthesis, system-level synthesis, processing elements, PE networks, graph embedding, placement, simulated annealing, emulation

## 1. INTRODUCTION

Fast physical model simulations are required in various domains, including biomedical engineering, physics, chemistry, and much more. A physical model represents some observable physical phenomena, usually as a set of normal, partial differential, or ordinary differential equations. The set of equations can be solved using time-stepping equation solvers.


*FPGA'13,* February 11–13, 2013, Monterey, California, USA.


In the cyber-physical system domain, previous work uses physical models to interact with and test devices such as ventilators [16], pacemakers [13], and unmanned aerial vehicles [9]. Using physical model simulations for testing can be preferable over the actual physical environment when such an environment is difficult, expensive, or dangerous to create or use. Physical models may also be more accurate than physical analogs, e.g., a balloon may capture some of the behavior of a lung, but may not be able to accurately model various lung diseases.

Our previous research has been able to speed up physical model simulation up to three orders of magnitude versus multicore desktop processors, by partitioning physical model computation across hundreds of processing elements (PEs) on an FPGA [12], each PE optimized to execute time-stepping equation solvers [11].

Many physical models share the same natural structure as the corresponding physical system. For example, a Weibel lung model [27] utilizes a binary tree structure because the lung physiology itself is a tree in which the trachea is the root and where gas exchange occurs at the leaves. Similarly, atrial cell models utilize a three-dimensional mesh structure to simulate the propagation of electrical signals across tissues of cardiac cells [29]. Equations of the physical system are grouped naturally, e.g., 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. Generally, the natural structure of a physical model provides an optimal grouping of equations that minimizes communication costs.

A key contribution of this work is utilizing the natural structure of simulated physical model during placement of a PE network onto an FPGA. By using graph embedding techniques that have been

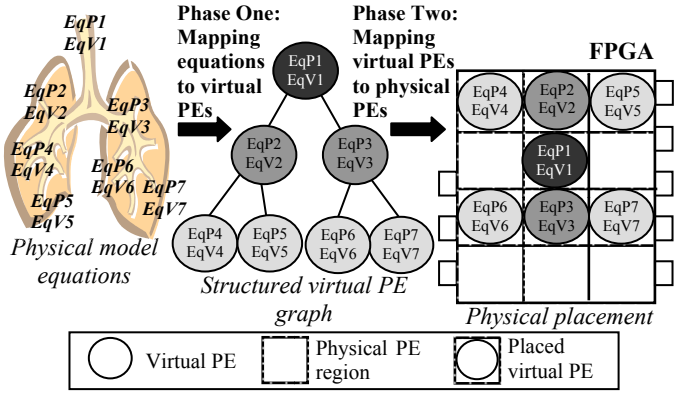


**Figure 1: Two-phase approach of mapping physical model equations onto a structured graph of virtual PEs, and mapping virtual PEs onto a FPGA utilizing graph embedding techniques.**

extensively researched in graph theory literature, the structure of the physical model can be embedded onto a two-dimensional grid of PE elements on an FPGA. By performing graph embeddings, the resulting circuit incurs less communication cost and enables higher circuit frequencies, translating to faster execution of physical models. A secondary contribution is the definition of a simulated annealing approach that provides cost and neighbor functions for minimizing distances between PEs placed on a grid of physical regions on a FPGA, used for unknown model structures and also for evaluating the first contribution.

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. A structured virtual PE graph has virtual PE nodes that contain groups of equations, have connections to other virtual PE nodes, and is structured in the form of the physical model. Physical placement can then be performed by defining physical PE regions where virtual PEs may be mapped, and then either applying the appropriate graph embedding algorithm or using a general simulated annealing approach to perform the mapping. In the right side of Figure 1, a graph embedding algorithm maps a binary tree to a two-dimensional grid by placing the root in a physical PE region in middle of the grid, and expanding the child subtrees out in different directions.

The rest of the paper is structured as follows. Section 2 describes past work on accelerating physical models, as well as other applications of graph embedding theory. Section 3 describes some example physical models with specific structures. Section 4 describes the process of partitioning equations to virtual PEs. Section 5 describes the mapping of virtual PEs to physical PE regions, either by using a graph embedding or simulated annealing approach. Section 6 discusses experiments showing circuit frequency speedups when using different placement strategies.

## 2. RELATED WORK

Our past research efforts on fast execution of physical models on FPGAs [11][12] have achieved orders of magnitude of acceleration over executing on desktop processors and several times speedups over graphical processing units, with improvements even when considering time/dollar-cost. Speedup was achieved by parallelization of differential equations across hundreds of PEs, for complete applications and not just kernels. FPGAs excel at executing physical models because the massively-parallel local-neighbor communication of physical models represents an excellent match for FPGA fabrics, avoiding common memory or external input/output bottleneck problems. An automated flow was presented that translates a specification of the physical model into an equation dependency graph, partitions equations into PEs via simulated annealing, schedules computations and custom point-to-point communications, and finally generates HDL for commercial tool synthesis. 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 [12]. de Pimentel also utilized an FPGA to accelerate a heart model on an FPGA [19], and interfaced the simulation with a pacemaker via analog-digital converters. Tagkopoulos built a custom FPGA for the simulation of gene regulatory networks [22].

While the above past efforts used heuristics to map equations to PEs and have relied on commercial tools to place PEs, we propose that a mapping of equations to PEs that maintains physical model natural structure and performs placement based on the structure can yield faster circuits and faster execution of physical models.

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 summarized the issue and offered a heuristic for mapping algorithm tasks to adjacent processors in a "finite element machine" array processor [5]. Later, Berman and Snyder offered a general solution for embedding common structures such as cubes, meshes, linear arrays, and trees [3]. Much of that research has been used in distributed and high-performance computing domains for mapping tasks to processors to minimize communication costs [4]. VLSI design has also utilized graph embedding techniques, including minimizing communication between a binary-tree structured processor network implemented on an optimally sized square [23].

The general problem of placing logic into a programmable FPGA fabric has previously been considered as a graph embedding problem, as opposed to the typical approach of iterative heuristics and recursive partitioning. Banerjee proposed converting netlists into hypergraphs and embedding the hypergraphs onto the two-dimensional grid of FPGA resources using a recursive space-filling curve [2]. This approach can yield up to 2x faster runtimes for placement, but yields little improvement to the critical path delays needed for faster physical model simulations. Gopalakrishnan proposed a new approach called CAPRI to create an initial placement of a design based on the embedding of a netlist into the target FPGA platform [10]. CAPRI models the routing delays of the target FPGA platform in a metric space and uses matrix projections to minimize distortion between graph abstractions of the netlist and platform. These previous works have focused on mapping to low-level FPGA resources like CLBs, whereas our work focuses on the best placement of a network of hundreds of individual PEs in abstracted FPGA physical regions.

## 3. 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 twenty additional splits as the airway passage diameters decrease and

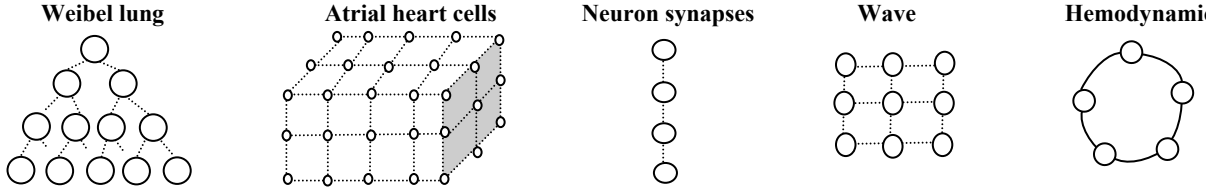


**Figure 2: Various physical models and graphs of their representative structures.**

eventually are able to support blood-gas exchange alveoli. The lung has thus often been modeled as a binary tree of twenty or more generations such that gas flow at the trachea can be used to compute the pressure and volume of internal branches [27]. Similarly, cell models that simulate electrical activity across heart atrium walls utilize a three-dimensional mesh structure to allow neighboring cells to propagate signals. Figure 2 shows some examples of physical models and their corresponding structures, which are described below.

*Weibel lung*: The classical 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 [27]. Each node of the tree computes the volume $V$ and flow $F$ of the corresponding branch:

$$\frac{dV_i}{dt} = F_{parent} \cdot C_1 + V_i \cdot C_2 + F_i$$

$$\frac{dF_i}{dt} = V_i \cdot C_3 - F_i \cdot C_4 - V_{R\_child} \cdot C_5 - V_{L\_child} \cdot C_6$$

*Atrial heart cells:* A 3-dimensional mesh of cells, where each cell propagates signals to its neighbors [29]. $v_i$ is the membrane potential of cell $i$ and is computed by the following equation.

$$\frac{dv_i}{dt} = (c_1 + (v_{x1} + v_{x2} + v_{y1} + v_{y2} + v_{z1} + v_{z2} - c_3 \cdot v_i) \cdot c_2) \cdot c_3$$

*Neuron synapses:* A 1-dimensional array of cells that simulates the firing of neuron synapses. $s$ is the synaptic variable, $v$ is the membrane potential, and $w$ is a channel gating variable [21].

$$\frac{dv_i}{dt} = c_1 \cdot v_i + w_i - c_2 \cdot (v_i - c_3) \cdot (s_{i-1} + s_{i+1})$$

$$\frac{dw_i}{dt} = c_1 \cdot w_i - v_i$$

$$\frac{ds_i}{dt} = c_1 \cdot (1 - s_i) \cdot (v_i - c_2) - c_3 \cdot s_i$$

*Wave:* A wave model has a two-dimensional mesh network structure and is often used to model the propagation of sound, acoustics, etc [17]. The amplitude of the signal at node $i$ is given by:

$$\frac{du_i}{dt} = c_1 \cdot (u_{x1} + u_{x2} + u_{y1} + u_{y2}) + c_2 \cdot u_i$$

*Hemodynamics*: A model that simulates the circulation of the human body, and includes submodels for the left/right heart ventricle and pulmonary/systemic tissues [25]. The hemodynamic model is arranged in a circular structure. Since there are many different types of equations to model this system, we omit the detailed descriptions here.

Large physical models such as those described above can be partitioned to hundreds of PEs in a network to achieve very fast simulation speeds. By maintaining the structure associated with the physical model during physical placement of PEs onto 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, etc. 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 [11]. However, finding the globally optimal solution for physical models containing thousands of equations and hundreds of PEs is not feasible with this approach. Instead of attempting to recover structures with heuristics, we propose to preserve the connections as they were modeled so as to minimize communication cost.

# 4. PHASE 1: MAPPING EQUATIONS TO VIRTUAL PEs

Given the specification of a physical model that enumerates the physical model equations, a map must be built that groups equations into a structured virtual PE graph $G$ that maintains the structure of the physical model. Equations must first be partitioned to a structured virtual PE graph of unconstrained size. Second, the graph must be reduced in size via folding to fit into available resources of the *target platform*. The target platform, which is typically an FPGA but could be an ASIC, is the device that the circuit will be placed on. There are limited resources on the target platform, thus folding is necessary for physical models whose structured virtual PE graphs exceed the size of the target platform.

## 4.1 Partitioning equations

Let $G=(v,e)$, where $v=\{v_1,v_2...,v_n\}$ is a set of $n$ vertices and $e=\{e_1,e_2,...,e_k\}$ is a set of $k$ edges between vertices in $v$. Let $E=\{E_1,E_2,...,E_m\}$ be the set of equations defined in the specification of the physical model. The set of vertices $v$ represent 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 $e_i=(v,u)$ exists, then there exists dependencies between the equations hosted in $v$ and $u$. The graph $G$ and its nodes and edges are defined by the structure of the physical model; a three-level binary-tree shaped Weibel lung model thus would have a graph that contains:

$G_v=\{v1,v2,v3,v4,v5,v6,v7\}$

$G_e=\{(v1,v2),(v1,v3),(v2,4),(v2,v5),(v3,v6),(v3,v7)\})$

Each equation $E_i$ can be allocated to a vertex $v_i$ in $G$ according to a surjective mapping function $f : E \rightarrow G_v$. 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 $G$ which 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 4000 differential equations. In order to meet the physical constraints of using a real platform when mapping virtual PEs to physical PEs, the virtual PE graph $G$ must first be scaled down. We perform *graph folding* on $G$ by applying a homomorphic folding function $\varphi$ that maps the larger graph to a smaller, more compact version $G'$ while preserving the structure of $G$. In particular, $\varphi$ 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 $S$; $\varphi : G \rightarrow G' \mid G'_n < S$. $\varphi$ must also maintain the topology of $G$ in $G'$ by either maintaining an existing edge of

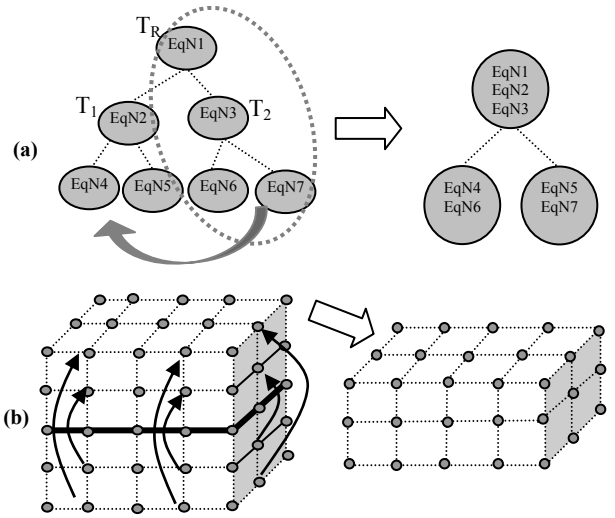


**Figure 3: Contraction of the PE dependency graph by folding: (a) binary tree (b) 3-dimensional mesh.**

*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. Informally, structures that are symmetric can generally be folded by cutting the graph into two subgraphs, and merging vertices that share the same position in each subgraph. Folding of graphs has been previously explored in graph theory literature [1][7][26]. Aleliunas [1] and Ellis [7] utilized folding techniques in order reduce the aspect ratio of rectangular graphs into forms that could be embedded onto a two-dimensional grid. Other work has developed algorithms for folding strongly balanced hypertrees in order to embed them into hypercube structures [26].

The exact definition of $\varphi$ 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 tended to be reduced by halves, potentially creating a situation where almost half of the physical PE regions of the target platform are empty. One solution is to simply manually merge the final few virtual PEs if the size constraint of the target platform is only slightly less than the size of the structured virtual PE graph. The following section provides examples that target binary tree physical models, describing the mapping function *f* and folding function $\varphi$ which 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 fourteen 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 3(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 3(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. Non-symmetrical 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 PEs TO PHYSICAL PEs

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 two-dimensional grid of physical PE regions on a target FPGA platform. Each physical PE region in the grid contains just enough resources to implement a single PE. Physical PE regions are defined at specific locations to create a two-dimensional grid that can be addressed using a XY Cartesian coordinate system. Whether or not the physical PE region actually contains a physical PE depends on the subsequent mapping. Virtual PEs can be mapped to physical PE regions on the grid using either structure-specific graph embedding techniques that place a guest graph into a host graph algorithmically, or by a generic simulated annealing approach with custom cost functions to reduce wire length.

## 5.1 FPGA platform two-dimensional grid

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 manifests between memories or logic components that belong to the same PE. Each PE in our design requires two memories (BRAMs), one multiplier (DSP), and approximately 250 lookup-tables (LUTs). We expected that communication channels between different PEs would be the primary cause of delay. Because of the complexity of large PE networks, the tools are not able to always place components of the same PE nearby each other. This problem can be addressed via the use of placement constraints during synthesis and place and route.

We first utilize Relationally Placed Macros (RPMs) to establish relative distances between PE memories. RPMs have been shown to provide faster circuit designs, even with modern tools [20]. On Xilinx FPGAs, a Cartesian coordinate system is used to specify the locations of components like DSPs and BRAMs (Figure 4). BRAM and DSP modules are physically located in homogeneous

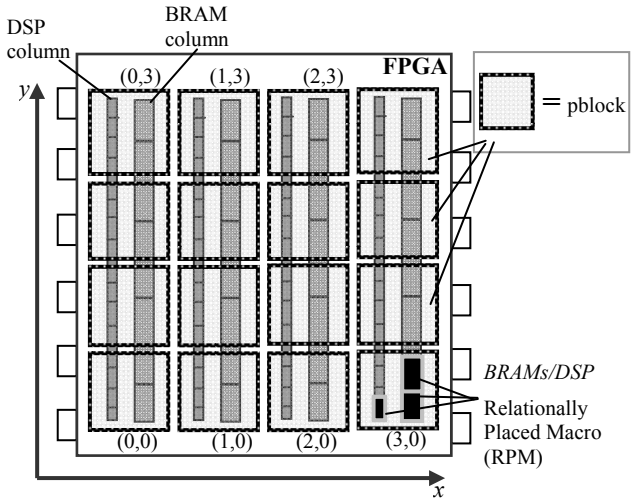


**Figure 4: A 4x4 grid of physical PEs on a FPGA. Physical PEs are constrained to specific areas using pblocks.**

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 grid size that can be constructed is 14x39, yielding a maximum of 504 PEs. For the vast majority of physical models, 500 PEs is sufficient for much faster than real-time simulation speeds. We note that our approach is not limited to one specific tool, platform 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) above 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 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 [6][15][24]. The typical metric that graph embedding algorithms are evaluated by is *maximum dilation*, or the maximum number of nodes that a wire may need to pass through to be completed. Since in physical model-solving PE networks the communication channels are point-to-point between PEs, the dilation is always exactly one. We thus alter the metric's definition slightly to be the maximum wire length between any two PEs. A second important metric is the *average dilation*, or the 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, since once a PE has been constrained using RPMs and pblocks the longest wires for any complex network is typically connected between different PEs (as opposed to internal PE connections). Lower average dilation means that less routing resources will be required, which typically results in faster circuits [28]. In the next sections, we first define the graph embedding problem. We then show how to utilize a graph embedding technique called H-tree construction to embed a binary tree structured physical model into a 2D grid of PEs.

### 5.2.1 *Graph embedding*

The graph embedding problem relates to the general mapping problem [3], where computational tasks must be placed onto a host architecture such that communication between PEs is minimized. 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 the 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, then 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 $\psi_V$ and $\psi_E$. Since dilation in the context of PE networks on FPGAs with point-to-point communication is wire length, we use a basic Euclidean distance measure $D = sqrt( (y2-y1)^2 + (x2-x1)^2 )$. While possible to measure dilation using specific FPGA routing architecture characteristics [10], at a macro level the simple distance between physical grid locations will suffice.

### *5.2.2 Example: Binary tree embedding onto 2D grid*

Embedded binary trees onto two-dimensional grids is a thoroughly researched area [6][14][24]. It has been proven that the graph embedding of a binary tree onto optimally-sized square grids have an $O(sqrt(n))$ maximum dilation, where $n$ is the number of generations of the tree. We utilize the H-tree construction technique that is used in VLSI for the layout of tree architectures onto optimally sized square hosts [23][30]. H-tree construction creates an H-fractal tree shaped liked that of Figure 5, where each subsequent branch of the tree alternates between

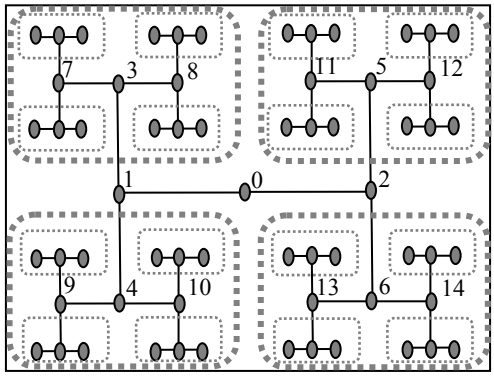


**Figure 5: Six level binary-tree placed on a square 2-dimensional mesh. Dashed boxes indicate recursive splits into subtrees.**

horizontal and vertical tracks and wire length is halved. This process is done by splitting the graph recursively 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 4 subtrees. In Figure 5, the tree is labeled by breadth-first ordering, such that the root is '0', the left child is '1', the right child is '2', and so on. Leaf nodes are not labeled for figure clarity. The thick dashed boxes represent the subtrees of the first recursive split; the row of vertices '0', '1', and '2' have a horizontal track allocated to them. The thin dashed boxes represent the subtrees created by a second recursive split of each of the first four subtrees. Additional horizontal tracks are added for the three relevant parent nodes of each split subtree. Following the second split, leaf nodes can be placed nearby their parents.

For optimally-sized square grids, the method demonstrated in Figure 5 produces optimal results (in terms of dilation). However, for rectangular-shaped grids such as the 14x39 PE grid available on our target FPGA, H-tree construction can not be immediately applied without some modifications. For example, the number of vertical tracks required for a 7-generation tree using the H-tree method is 31, or more than twice the number of available columns in the FPGA PE grid. We can take advantage of the fact that our FPGA can route wires between PEs diagonally, as opposed to the strict row-column ordering of previous H-tree considerations [14]. Also, since the width of the target is the limiting factor to the number of possible recursive splits, it's not possible to maintain the nice H-fractal shape of the graph embedding 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, than perform compaction and reordering of rows to further minimize maximum wire length.

1. Separate the grid into 4 quadrants to host the initial split of the tree.
2. Place the root node $M_0$ and its children $L_0$, $R_0$ in the center row of the grid. $M_0$ is placed in a column in the center of the grid. $L_0$ and $R_0$ are placed in a middle column of the neighboring upper and lower quadrants
3. Place each child of $L_0$ and $R_0$ onto the same vertical track as its parent, and onto the center row of a quadrant (Figure 6a).

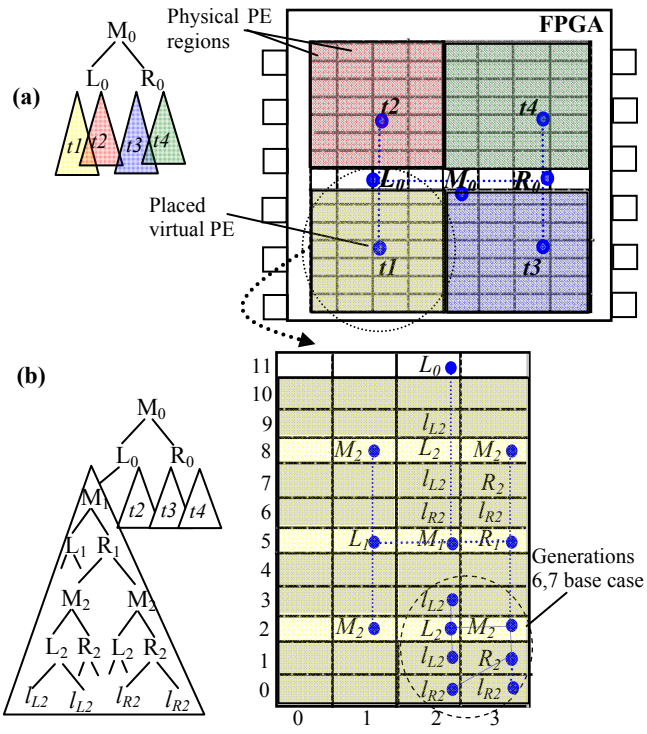


**Figure 6: Embedding 7-level binary tree into a rectangular 2D grid: (a) Initial split of 4 subtrees (not to scale), (b) Two additional recursive splits. White rows host root and children of a split branch. For clarity, not all branches are shown.**

4. Recursively split each subtree by placing the children of the subtree's root on the same row, and allocating additional rows to host new subtrees (Figure 6b).
5. At generation N-1, utilize a known placement to place the final levels (non-fractal shape).

The process described in the steps above can be seen in Figure 6. 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. Note that rows four and ten 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.

## 5.3 Simulated annealing based placement

This section provides a general method for mapping a structured virtual PE graph to physical PEs by using a simulated annealing approach. Such a general method can be useful when a physical model has no obvious structure for which a graph embedding algorithm could be used, such as an unbalanced or asymmetrical tree [8]. Simulated annealing also yields useful comparisons to the graph embedding approach by providing reasonable PE placements. We define a cost function that considers FPGA architectural features, critical path length, and wire congestion; it is shown experimentally that our cost function correlates linearly

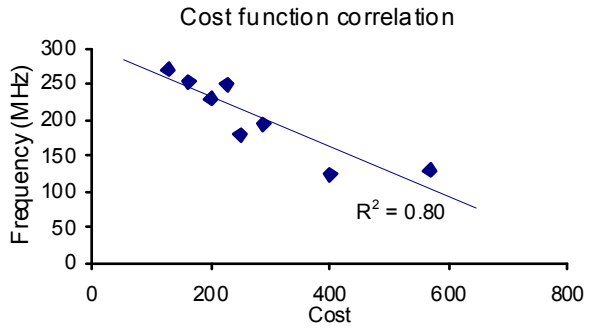


**Figure 7: Simulated annealing cost function correlates with resulting circuit frequency. A variety of different physical models are represented.**

with resulting circuit frequency. We also present a neighbor function that swaps PEs using vectors based on the placement of connected PEs. Our neighbor function provides faster convergence and results in lower cost placements than performing random swaps of PEs.

### 5.3.1 Cost function

The cost function of the simulated annealing based placement approach is defined as:

*Cost = w1*Sum + w2*Max + w3*Gaps*

*Sum* is the total of all the wire lengths in the design. By minimizing the sum of the wire lengths, the wire congestion in the design is reduced, which impacts critical path timing less. *Max* is the maximum wire length in the design. Minimizing *Max* is the key goal during simulated annealing, because it will likely represent the critical path in the circuit. *Gaps* is the number of wires that cross an area on the FPGA that must be routed through or around. For example, on most Xilinx Virtex 6 chips there is a large gap in the middle for monitoring or programming components and where user design logic can not be placed (see Figure 11). Wires through such gaps incur extra routing delays and thus we strive to reduce the amount of those types of connections. The constants *w1*, *w2*, and *w3* are weighting coefficients that can be used as tuning knobs for the algorithm. Typical values of *w1*, *w2*, and *w3* are 0.1, 10, and 1 respectively. *w2* is the most critical parameter, and should be selected based on the total number of wires in the design. If there are many wires, the *w1*Sum* factor may be very high, and the maximum wire length M*ax* factor may not contribute much to the cost of the current solution – in such cases *w2* should be increased to offset this effect. Figure 7 shows a linear regression representing how the cost function relates to the resulting circuit frequency of a PE network placed using simulated annealing.

### 5.3.2 Neighbor function

The neighbor function in a simulated annealing algorithm moves the current state of the design in order to explore the solution space of the problem. The neighbor function presented here attempts to cluster connected PEs together, hopefully reducing wire lengths in the process. A random physical PE region $P_1$ that contains a mapped virtual PE $V$ is first selected to be moved. 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 vector $v = (r,\theta)$ is built such that $r = sqrt(dx^2 + dy^2)$ and $\theta = tan^{-1}(dy/dx)$, where $dx$ and $dy$ are the differences in the x and y coordinates between $P_1$ and $P_p$. An average of all the connection vectors yields a target

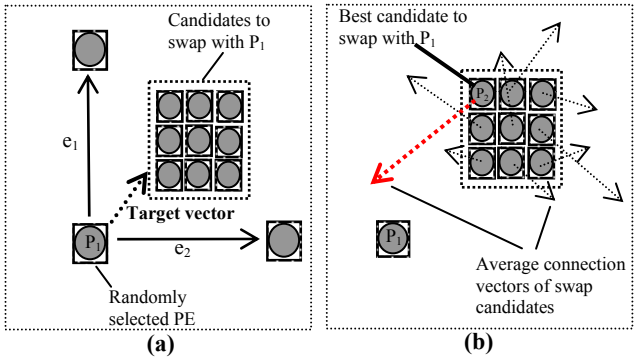


**Figure 8: The neighbor function picks two PEs (P1,P2) by randomly selecting P1, (a) finding candidates for P2 by averaging P1's connections (e1, e2), and (b) picking P2 based on the distance between P1 and a candidate's average connection vector endpoint.**

vector that identifies a physical PE region that would reduce the average wire length of the connections to the PE if the virtual PE were placed there. If the target physical PE region does not have a virtual PE mapped, than the virtual PE is moved onto the target physical PE region. If the target physical PE region does have a virtual PE mapped, than an evaluation of the target physical PE region and each of its neighbors in the grid takes place to determine the best candidate for a swap. The target physical PE region and its neighbors have their connections' vectors averaged in turn. The region that has an average connection vector endpoint closest to $P_1$ is selected to be swapped. If any of the neighbors do not contain mapped virtual PEs, than the empty neighbor is automatically selected to be swapped. Figure 8 shows how the neighbor function works. A random PE $P_1$ is first selected. An average of the two connections of $P_1$, $e_1$ and $e_2$, yields a target vector that denotes an area of the platform where $P_1$ should be placed to minimize the wire lengths of $e_1$ and $e_2$. Each candidate physical PE region in the area has its connections averaged (Figure 8b). The candidate physical PE region that has an average connection vector closest to $P_1$ is in the top left, thus a swap would occur with the top left PE ($P_2$) and $P_1$.

Figure 9 shows the convergence of the design cost towards a final solution for 50K iterations of the simulated annealing algorithm while implementing a neuron model utilizing 256 PEs. Using our custom neighbor function, the resulting cost is 50% less than given by the random alternative.

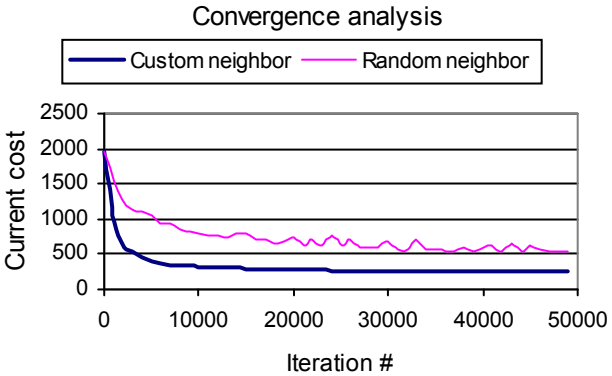


**Figure 9: Convergence of custom neighbor function compared to random swaps.**

### 5.3.3 *Annealing temperature schedule*

The cooling schedule used during simulated annealing can cause dramatic differences in the obtained solution [18]. To verify that we chose the correct schedule for this problem, we have experimented with linear, geometric, and exponential type cooling schedule functions. We found that both linear and geometric schedules produce a solution with a similar cost for a given physical model, while the exponential schedule ($\alpha = 0.99$) yields a solution that is highly dependent 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 paper utilize a geometric cooling schedule.

## 6. EXPERIMENTS

To evaluate graph embedding as a technique for accelerating physical model simulations 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 one-dimensional neuron array, and a two-dimensional 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, with flags '-ol high' and '-xe normal' to encourage the tool to work hard at achieving timing closure. Note that to implement the 11-generation Weibel model, we use 500 physical PEs. 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 leafs until the size constraint is met. The alternative is to continue folding the structured virtual PE graph until the size constraint is met, which would result in 256 virtual PEs and almost 50% of the available resources unutilized.

For each physical model, we implemented three methods of placement for the PE networks. The first method utilizes the compiler from previous work [11] to partition the physical model equations to PEs and generate a custom communication network. No constraints are used to map the PEs to specific physical PE regions; 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 5.3 to map virtual PEs to physical regions. For the simulated annealing algorithm in all cases we utilize a geometric cooling schedule, and let the algorithm run for 50K iterations to reach a steady state. The weighting constants (*w1*,*w2*,*w3*) are (0.08,10,1). 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 uses a graph embedding algorithm specific to the selected physical model. The Weibel model uses a H-tree graph embedding as described previously. The one-dimensional neuron model is a linear array of 6400 neurons, thus the graph embedding that is used places PEs into rows and connects the rows at the edges to form a Hamiltonian path amongst all PEs. The two-dimensional neuron model consists of a two-dimensional 64x64 mesh of neurons, where each neuron is connected to at most 4 neighbors. The graph embedding for the two-dimensional neuron model is a direct mapping onto the two-dimensional grid of FPGA physical regions, after folding the original physical model.

### 6.1 Results

Figure 10 shows the resulting circuit frequencies of implementing PE networks on an FPGA with the above three techniques. The 'NoPhys_XlnxPlcmt' columns do not use physical placement constraints. 'Phys_SimAnnlPlcmt' columns use simulated annealing to map virtual PEs to physical PE regions. 'Phys_EmbedPlcmt' uses an embedding approach appropriate to the implemented model. For the same model, all three approaches use the same RTL description of the circuit. The graph embedding approach is almost always able to produce a circuit that tops 300 MHz. The ceiling for the circuit frequency in a PE network is approximately 310 MHz for the selected platform. The ceiling can be determined by implementing 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 go 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 thus provide placements that allow the circuit to approach the frequency ceiling. The only embedding example that could not reach the ceiling of 310 MHz is the 11-generation Weibel lung model using 500 PEs. Because the two-dimensional grid of the physical PE regions is narrow, an optimal embedding of the tree cannot occur. Wire lengths between successive generations are much longer, resulting in longer critical path delays.

Some data points of the method using no physical placement constraints are marked 'N/A'. This indicates that the Xilinx tools were not able to place and route the design due to high

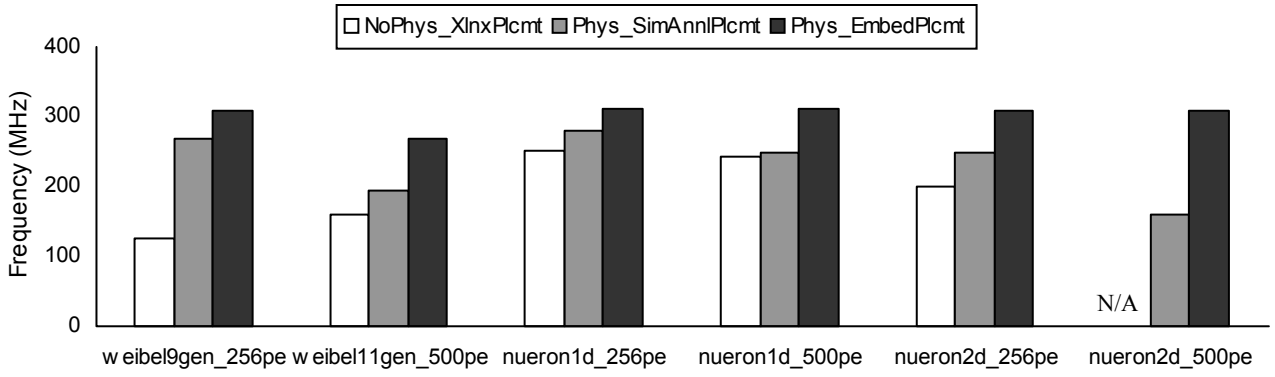


**Figure 10: Frequencies of PE network implementations simulating a Weibel lung and 1D/2D neuron networks. Each PE network was placed with (i) no physical region constraints, (ii) physical regions selected by simulated annealing, and (iii) physical regions selected by embedding the model structure onto the FPGA grid. Points marked 'N/A' could not be routed because of high complexity. (i), (ii), and (iii) all use the same RTL description during synthesis, but (ii) and (iii) use region constraints.**

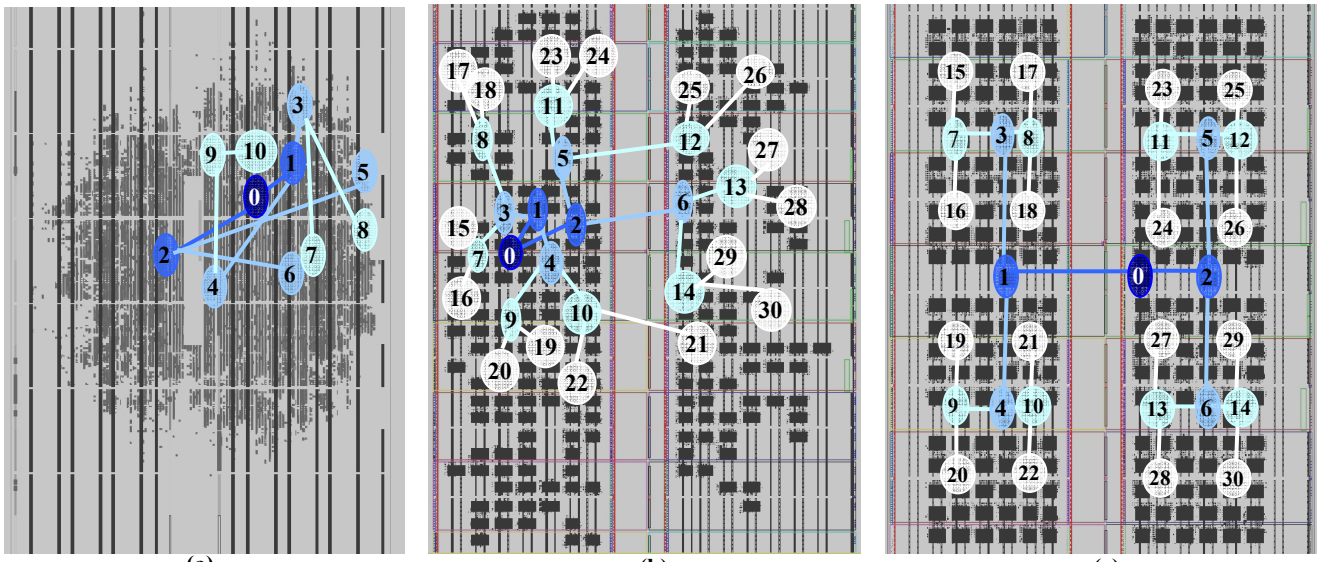


**Figure 11: Different placements of a 256 PE Weibel lung model. The virtual PEs of the physical model are labeled in breadth-first order, where 0 is the root. Dark nodes are closer to the root. (a) Unconstrained placement of PEs performed by Xilinx tools. (b) PEs mapped to physical PE regions by simulated annealing. (c) PEs mapped to physical PE regions via a graph embedding algorithm. Empty (lighter) space in the circuits could be used to implement other logic, such as tracing or debug support.**

congestion. The compiler that partitioned the equations and created the communication network could not adequately reduce the data dependencies between PEs for these large physical models, resulting in an overwhelming number of wires in the network. Note that the designs are routable if we use either a simulated annealing or graph embedding approach, which indicates that graph embedding or a simulating annealing approach enable implementation of physical models on an FPGA that previously could not be implemented on the FPGA.

## 6.2 A look inside the FPGA

Figure 11 shows a graphical depiction of 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 connections shows where virtual PEs have been mapped onto the FPGA. Figure 11(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 non-optimal locations such that the wire distances between PEs can be very long. For example, the wires between node two and its children five and six span more than halfway across the entire design.

Figure 11(b) depicts a typical result of using the simulated annealing algorithm. Each black block indicates where a virtual PE was mapped to a physical PE location. An empty space in the grid means that no virtual PE was mapped to the grid at that physical region. The effect of the simulated annealing algorithm can be seen by evaluating the placement of the virtual PEs onto the grid. Nodes that share connections tend to be grouped together, while overall the tree tends to expand outward from the center of the grid. Leaf nodes are grouped towards the outside of the grid. Figure 11(c) shows an embedding of the tree onto the host grid using the graph embedding approach. Recall that the center of many common (Xilinx) FPGAs contains immutable logic, and thus minimization of the routing across the center is desired. The embedding requires only 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 power on average than both the simulated annealing and embedding constrained placement approaches.

# 7. CONCLUSION

We presented an approach for fast physical model simulation on FPGAs that makes use of the physical model's structure to improve performance. The approach's first phase maps physical model equations to a structured virtual PE graph and groups related equations. The approach's second phase maps the structured virtual PE graph to a two-dimensional grid of FPGA physical regions by using either a graph embedding or simulated annealing technique. The graph embedding and simulated annealing techniques provide 65% and 35% average increases in circuit frequencies, respectively, compared to placements that do not map to specific physical regions.

# 8. ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation (CNS1016792, CPS1136146), the Semiconductor Research Corporation (GRC 2143.001), and a U.S. Department of Education GAANN fellowship.

# 9. REFERENCES

[1] Aleliunas, R., and Rosenberg, A.L. 1982. On Embedding Rectangular Grids in Square Grids. Computers, IEEE Transactions on , vol.C-31, no.9, pp.907-913, Sept. 1982.

[2] Banerjee, P., Sur-Kolay, S., Bishnu, A., Das, S., Nandy, and S.C., Bhattacharjee, S. 2009. FPGA placement using space-filling curves: Theory meets practice. ACM Trans. Embed. Comput. Syst. vol. 9, no. 2, Oct. 2009.

[3] Berman, F., and Snyder, L. 1987. On mapping parallel algorithms into parallel architectures, Journal of Parallel and Distributed Computing, vol. 4, no.5, Oct. 1987, pp 439-458.

[4] Bhatelé, A., and Kalé, L.V. 2008. Benefits of Topology Aware Mapping for Mesh Interconnects. Parallel Processing Letters, vol.18, no.4, pp.549-566, 2008.

[5] Bokhari, S.H. 1981. On the Mapping Problem. Computers, IEEE Transactions on , vol.C-30, no.3, pp. 207-214, March 1981.

[6] Chen, W.K., and Stallmann, M. 1995. On embedding binary trees into hypercubes. J. Parallel Distrib. Comput. 24, 2 (February 1995), 132-138.

[7] Ellis, J.A. 1991. Embedding Rectangular Grids Into Square Grids. IEEE Transactions on Computers, pp. 46-52, Jan. 1991.

[8] Gabryś, E., Rybaczuk, M.,and Kędzia, A. 2005. Fractal models of circulatory system. Symmetrical and asymmetrical approach comparison, Chaos, Solitons Fractals, vol. 24, no. 3, May 2005, pp 707-715.

[9] Gholkar, A., Isaacs, A., and Arya, H. 2004. Hardware-In-Loop Simulator for Mini Aerial Vehicle, Sixth Real- Time Linux Workshop, NTU, Singapore, Nov. 2004.

[10] Gopalakrishnan, P., Li, X., and Pileggi, L. 2006. Architecture-aware FPGA placement using metric embedding. In Proceedings of the 43rd annual Design Automation Conference (DAC '06). ACM, New York, NY, USA, pp. 460-465.

[11] Huang, C., Vahid, F., and Givargis, T. 2011. A Custom FPGA Processor for Physical Model Ordinary Differential Equation Solving. Embedded Systems Letters, IEEE , vol.3, no.4, pp.113-116, Dec. 2011.

[12] Huang, C., Miller, B., Vahid, F., and Givargis, T. 2012. Synthesis of custom networks of heterogeneous processing elements for complex physical system emulation. In Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS '12). ACM, New York, NY, USA, pp. 215-224.

[13] Jiang, Z., Pajic, M., and Mangharam, R. 2011. Model-Based Closed-Loop Testing of Implantable Pacemakers. Cyber-Physical Systems (ICCPS), 2011 IEEE/ACM International Conference on, pp.131-140, April 2011.

[14] Lee, S.K., and Choi, H.A. 1996. Embedding of complete binary trees into meshes with row-column routing. Parallel and Distributed Systems, IEEE Transactions on , vol.7, no.5, pp.493-497, May 1996.

[15] Matic, S. 1990. Emulation of hypercube architecture on nearest-neighbor mesh-connected processing elements. Computers, IEEE Transactions on , vol.39, no.5, pp.698-700, May 1990.

[16] Miller, B., Vahid, F., and Givargis, T. 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, USA, pp. 859-862.

[17] Motuk, E., Woods, R., and Bilbao, S. 2005. Implementation of finite difference schemes for the wave equation on FPGA. ICASSP.

[18] Nourani, Y., and Andresen, B. 1998. A comparison of simulated annealing cooling strategies. Journal of Physics A: Mathematical and General, vol. 31, no. 41, 1998.

[19] de Pimentel, J.C.G., and Tirat-Gefen, Y.G. 2006. Hardware Acceleration for Real Time Simulation of Physiological Systems. Engineering in Medicine and Biology Society, 2006. EMBS '06. 28th Annual International Conference of the IEEE , vol., no., pp. 218-223, Aug. 2006.

[20] Singh, S. 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, USA, pp. 185-188.

[21] Terman, D., Ahn, S., Wang, X., and Just, W. 2008. Reducing neuronal networks to discrete dynamics, Physica D: Nonlinear Phenomena, vol. 237, no. 3, March 2008.

[22] Tagkopoulos, I., Zukowski, C., Cavelier, G., and Anastassiou, D. 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, USA, pp. 132-135.

[23] Ullma, J.D. 1984. Computational Aspects of VLSI. W. H. Freeman & Co., New York, NY, USA.

[24] Ullman, S., and Narahari, B. 1990. Mapping binary precedence trees to hypercubes and meshes. Parallel and Distributed Processing, 1990. Proceedings of the Second IEEE Symposium on , pp. 838-841, Dec. 1990.

[25] van Meurs, WL. 2011. Modeling and Simulation in Biomedical Engineering: Applications in Cardiorespiratory Physiology. McGraw-Hill Professional.

[26] Wagner, A.S. 1991. Embedding all binary trees in the hypercube. Parallel and Distributed Processing, Proceedings of the Third IEEE Symposium on , pp. 104-111, Dec 1991.

[27] Weibel, E.R. 1963. Morphometry of the Human Lung. Berlin, Germany: Springer-Verlag 1963.

[28] Xilinx, 2010. Inc. Virtex-6 FPGA Routing Optimization Design Techniques. http://www.xilinx.com/support/documentation/white_papers/wp311.pdf

[29] Zhang, H., Holden, A.V., and Boyett, M.R. 2001. Gradient model versus mosaic model of the sinoatrial node. Circulation. vol. 103, pp. 584-588.

[30] Zienicke, P. 1990. Embeddings of Treelike Graphs into 2-Dimensional Meshes. In Proceedings of the 16th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '90). London, UK, pp. 182-192.