Neural Network-Based Thermal Simulation of Integrated Circuits on GPUs

Arvind Sridhar, *Student Member, IEEE,* Alessandro Vincenzi, *Student Member, IEEE,* Martino Ruggiero, *Member, IEEE,* and David Atienza, *Member, IEEE*

***Abstract*—With the rising challenges in heat removal in integrated circuits (ICs), the development of thermal-aware computing architectures and run-time management systems has become indispensable to the continuation of IC design scaling. These thermal-aware design technologies of the future strongly depend on the availability of efficient and accurate means for thermal modeling and analysis. These thermal models must have not only the sufficient accuracy to capture the complex mechanisms that regulate thermal diffusion in ICs, but also a level of abstraction that allows for their fast execution for design space exploration. In this paper, we propose an innovative thermal modeling approach for full-chips that can handle the scalability problem of transient heat flow simulation in large 2-D/3-D multiprocessor ICs. This is achieved by parallelizing the computation-intensive task of transient temperature tracking using neural networks and exploiting the computational power of massively parallel graphics processing units. Our results show up to 35× run-time speedup compared to state-of-the-art IC thermal simulation tools while keeping the error lower than 1 °C. Speedups scale with the size of the 3-D multiprocessor ICs and our proposed method serves as a valuable design space exploration tool.**

***Index Terms*—2-D/3-D integrated circuits (ICs), graphics processing unit (GPU), neural networks (NNs), thermal modeling.**

I. Introduction

AN IMPORTANT side effect of the continued scaling and miniaturization of CMOS technology is the ever increasing device power density. The resulting difficulties in managing temperature, especially local hot spots, have been recognized as one of the major challenges for designers of electronic circuits in the latest technology nodes [1]. Although energy consumption is one of the main sources of temperature increase, a purely power-aware control and design is not an optimal solution to the thermal issues [2]. Power and energy are indeed regarded as cumulative quantities, whereas thermal issues appear because chips tend to warm-up in a nonuniform way, which leads to hot-spots and spatial gradients that can cause timing errors and physical damage. In fact, both spatial and temporal distribution of power consumption in integrated circuits (ICs) are relevant in the relationship between power/energy and temperature. As a consequence, thermal modeling and analysis has developed into a distinct area of study (with respect to power and energy modeling) that has gained considerable attention over the last decade due to the large power density increase in ICs [3]–[7].

In addition, the situation will be exacerbated by upcoming 3-D IC stacking [1]. 3-D stacking also brings disruptive and distinctive challenges related to thermal dissipation, which in this case involves cooling a volume instead of just cooling a planar IC surface [8]. In fact, neglecting thermal information during the design of 3-D ICs would imply excessive over-design, due to the extremely conservative constraints that designers may impose in order to guarantee correct circuit operation under all possible running conditions and resulting temperature transients.

As a result, new thermal modeling technologies that can capture the transient thermal behavior of hardware elements and their interaction with the software running on them are required to enable thermal-aware chip design and validation of dynamic thermal management strategies. In addition to possessing sufficient accuracy to capture the complex mechanisms that regulate thermal diffusion and radiation, these novel thermal modeling approaches also need to have a sufficiently high level of abstraction that allows for fast execution [8], [9].

In recent years, a number of full-chip 2-D/3-D thermal simulators, based on finite-element and compact models, have been proposed that provide detailed temperature distributions [6], [9], [10], [11]. However, thermal analysis using conventional methods is a resource-consuming and time-consuming task that can be strongly influenced by accuracy requirements, as well as the complexity of the run-time 2-D/3-D IC execution scenario under study, making them unfeasible for large-scale IC design.

Nevertheless, recently graphics processors have become increasingly competitive in speed, programmability, and price. Graphics processing units (GPUs) have already been used to implement many computationally intensive algorithms in various areas, including high-performance computing, scientific computation, and image processing [12]–[14]. Indeed, the main advantage of GPUs over central processing units

Manuscript received February 7, 2011; revised June 29, 2011 and September 27, 2011; accepted October 6, 2011. Date of current version December 21, 2011. This work was funded in part by the Nano-Tera RTD Project CMOSAIC (ref. 123618), which is financed by the Swiss Confederation and scientifically evaluated by SNSF, as well as by the PRO3D STREP Project (ref. FP7-ICT-248776) financed by the EC in the 7th Framework Program. This paper was recommended by Associate Editor P. Li.

The authors are with the Embedded Systems Laboratory, École Polytechnique Fédérale de Lausanne, Lausanne 1015, Switzerland (e-mail: arvind.sridhar@epfl.ch; alessandro.vincenzi@epfl.ch; martino.ruggiero@epfl.ch; david.atienza@epfl.ch).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2011.2174236

(CPUs) is the high computational throughput at a relatively low cost, which is achieved through their massively parallel architecture. Hence, efficient GPU usage and exploitation have already been identified as the next breakthrough in electronic design automation (EDA) tools [15].

However, it is difficult to structure algorithms to take advantage of massively parallel processors, such as GPUs. Even if existing thermal models of ICs can potentially exhibit a high level of parallelism, it is difficult to translate this parallelism into an efficient software implementation on commodity GPUs—only one attempt has been proposed until today [8]. Even though this approach has not yet been validated for real 3-D ICs with liquid cooling support, this paper outlines the potential capabilities of GPUs for thermal modeling. Therefore, novel GPU-friendly modeling technologies and capabilities are needed to support accurate and fast 2-D/3-D full-chip thermal analysis and simulation.

In this context, neural networks (NNs) can represent an optimal parallelizable solution to this problem. NN is an information processing method that was inspired by the way biological nervous systems, such as the brain, function to process information [16]. It is composed of a large number of highly interconnected processing elements (neurons) working in unison, and can be trained to solve complex problems, such as thermal modeling of specific IC layouts [17]. On one hand, NNs are very flexible and can be trained to mimic the behavior of any physical system with relative ease of implementation. On the other hand, a major drawback of NNs is their potentially long training time (several hours or even days) because they are very computation and data-intensive [18]. Nevertheless, their capacity to execute a large number of operations simultaneously and with relatively low data transfer makes them a potentially attractive concept for GPU computing [19].

Therefore, in this paper our goal is to utilize and exploit the power of GPUs, in combination with NNs, to develop a fast and accurate thermal modeling approach for transient thermal analysis of 2-D/3-D ICs. The main contributions of this paper are the following.

1) We present a new full-chip thermal modeling technology based on NNs and GPUs. The model is transient and can accurately predict the temporal evolution of both planar and 3-D chip temperatures. We have validated the accuracy of the model by comparing our results with 3D-ICE [9], a state-of-the-art thermal simulation tool for 3-D ICs.
2) We propose a methodology for the optimal training of the proposed NNs for thermal modeling. The one-time training is performed using state-of-the-art (but slow) thermal modeling tools. This step enables the removal of the unnecessary state variables in our presented NN-based thermal model (such as the temperatures of layers in which the user is not interested), which, in turn, drastically improves computational efficiency.
3) We introduce an innovative approximation, namely, the proximity-based reduction, which further reduces the computational complexity of our thermal model, while negligibly affecting model accuracy (an error of less than 1 °C).
4) Our NN-based thermal model specifically targets GPU platforms because they are architectures on which the matrix-vector multiplication is easily parallelizable. On the contrary, state-of-the-art thermal simulators like HotSpot [6] or 3D-ICE [9] are based on sequential operations that cannot be ported efficiently on massively parallel architectures. Thus, GPU exploitation in our proposed NN-based thermal modeling approach resulted in considerable time-savings (35× run-time speedup), especially for large IC problem sizes and detailed layout models, while incurring an error lower than 1 °C in comparison to state-of-the-art IC thermal simulation tools.
5) Once trained, our NN-based thermal simulator on GPU platforms can be reused any number of times with different 2-D/3-D IC floorplan configurations as long as the area of the dies remains constant—which is indeed the case in most thermal-aware placement algorithms [20], [21]. Hence, our proposed thermal modeling approach enables fast thermal-aware design space and floorplanning exploration based on GPU technology.

The rest of this paper is organized as follows. First, Section II reviews the previous research done on thermal modeling of ICs and GPU acceleration. Then, Section III describes the compact thermal model, which forms the basis for the training of the proposed NN-based simulator. Section IV presents the proposed new NN-based simulator, the training methodology, and the paradigms for accurate and optimal training. Section V presents the implementation scheme of the proposed model on a GPU platform. Finally, Section VI describes our experimental results, and Section VII summarizes the main conclusions of this paper.

II. Related Work

A considerable amount of research has gone into developing thermal models for planar ICs with conventional heat-sink based cooling. In [22], the authors presented different and detailed full-chip thermal models. These models provide detailed temperature distribution information across the silicon die and can be solved efficiently. Unfortunately, a limitation of the above model lies in the fact that they rely on some over-simplifications, in particular for the package thermal model. In comparison, there are also several package-level thermal models [11], [23] leveraging compact modeling techniques. These thermal models consist of networks of thermal resistances, whose values are extracted by data-fitting from the results of accurate but time-consuming detailed numerical package thermal model simulations (e.g., finite element method). Therefore, they are not fully parameterized and cannot be easily used to explore new system designs.

The finite difference (FD) method is the traditionally preferred approach used in the thermal simulation of ICs because it guarantees the best accuracy in modeling the complex 3-D structures of 3-D ICs, while retaining a sufficiently high level of abstraction. However, accurate 3-D thermal analysis

using FD method can be very expensive, which requires solving a huge system of linear equations with multimillion unknowns [10]. Other methods present simplified thermal models for steady-state simulations and provide no information about the transient thermal behavior of the ICs [3]. The methods in [6] and [7] use a finite-difference based method to generate a compact thermal model for the IC. In addition, while a high-level interconnect model is developed in [6] to simulate the effects of interconnect self-heating, [7] applies the alternative direction implicit technique for obtaining fast and stable transient results.

None of the above thermal models takes advantage of the computational power provided by modern GPUs. It must be noted that GPU-based parallel computing has also been employed in various EDA tools, such as analysis of large scale power distribution networks [13], physical synthesis [24], and gate-level simulation [25].

The only paper that proposes GPU-based full-chip thermal simulation methods for 3-D ICs with integrated microchannel cooling is [8]. This paper proposes an iterative methodology that uses a two-step relaxation based preconditioner for a conventional multigrid conjugate gradient method. This paper relies on modifying established simulation methodologies to suit the GPU architecture for faster and more efficient computing. However, being an iterative technique brings with it the typical limitations of slow convergence and sensitivity to initial conditions and initial guess. In addition, the accuracy of their thermal model has not been validated with respect to real measurements or other tools.

Our approach is based on NN modeling. Our NN model is a direct-method based simulator and hence, contrary to iterative techniques, the results are immediate. The authors in [17] already used a very different NN-based method for thermal modeling of a planar chip. However, their model is not applicable to 3-D ICs and has been designed for run-time thermal management. It uses online readings from on-chip temperature sensors for continuous training of the NN, and thus, it is not suited for design space exploration.

The enormous benefits that NN can bring while running on GPUs have been widely demonstrated in the literature. During the last few years, many different methods have been investigated to improve the performance of NNs on GPUs. Generic NNs have been implemented on GPUs in the recent past using shaders to modify the GPU's rendering pipeline [12]. This is a significantly less convenient approach, requiring the programmer to formulate the algorithm in terms of pixels, textures, vertices, and other graphics primitives. GPU programmability has considerably improved thanks to CUDA, which offers a much more flexible and intuitive programming platform. In [26], one of the first NNs for the unified shader architecture using CUDA was implemented.

III. Compact Thermal Modeling

Thermal simulation of ICs is conventionally performed using compact modeling [6], [27]. This makes use of the analogy between heat diffusion in solids and current flow in electrical systems governed by parabolic differential equations, like an RC circuit. In this paper, we will use the compact modeling method to characterize the thermal behavior of an IC, and then use the results to train the NN.

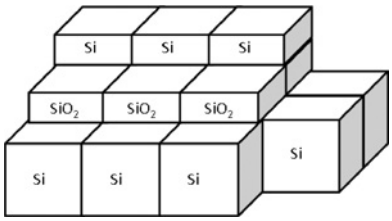

Fig. 1. Discretization of an IC into "thermal cells."

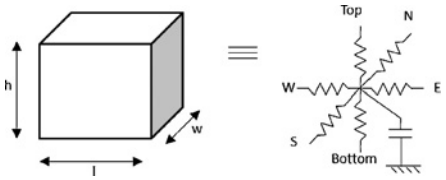

Fig. 2. Equivalent circuit of a solid thermal cell.

The conventional compact modeling for heat conduction in solids is done by applying the finite-difference approximation to the governing equations of heat transfer in solids [27]. This involves dividing the different layers in an IC into cuboidal "thermal cells" as shown in Fig. 1. Next, the well-known analogy between heat and electrical conduction is invoked here with the temperature represented as voltage and the heat flow represented as electric current [6] to convert each thermal cell into an equivalent electrical circuit as shown in Fig. 2. Finally, the nodes of these thermal cells are connected to the nodes of their neighboring cells through the interfaces by computing the equivalent conductances between them. The boundary conditions are given by convective resistances connecting the top layers of the IC to the ambient temperature. Although strictly speaking the convective resistance to the ambient is temperature dependent, in most early-stage thermal modeling and thermal-aware designs it is customary to assume a constant convective resistance to the ambient [6]. This results in a linear time invariant (LTI) system for thermal analysis and the following system of ordinary differential equations is generated:

$$\mathbf{G}\mathbf{X}(t) + \mathbf{C}\dot{\mathbf{X}}(t) = \mathbf{U}(t) \tag{1}$$

where $\mathbf{X}(t)$ is the vector of all node temperatures (as a function of time), $\mathbf{C}$ is a diagonal matrix of all cell capacitances, and $\mathbf{U}(t)$ is a vector of inputs (heat sources as a function of time) wherever they exist. $\mathbf{G}$ is a sparse symmetric block tri-diagonal conductance matrix, where the nonzero non-diagonal elements represent the connections between neighboring nodes. The formulation of heat flow equations, as described above, can be extended to structures containing multiple layers of thermal cells. The boundary conditions are given as source terms in the $\mathbf{U}$ vector. For example, if the top layer is connected to the ambient via some silicon-to-air thermal resistance, then the nodes on the top layer are grounded to the ambient temperature via that conductance term. This method can be used to generate a compact thermal model for any general heterogeneous structure like an IC die, and the 3-D temporal evolution of heat inside the 3-D IC can be accurately modeled.

For more information about the compact thermal modeling used in this paper, please refer to [9].

A "thermal grid" matrix is hence generated by connecting individual thermal cells in the entire IC. Cell dimensions used for the discretization of the heterogeneous IC structure are dependent upon the accuracy and speed requirements of the designer. In our experiments, we found that cell sizes of few hundred micrometers are sufficient for a typical IC thermal modeling accuracy [2], as they can incur errors of less than 1 °C.

The next step is the formulation of equations for the simulation of the thermal grid. For this, (1) is integrated numerically using the backward Euler method as follows:

$$\left(\mathbf{G} + \frac{1}{h}\mathbf{C}\right)\mathbf{X}(t_{n+1}) = \frac{1}{h}\mathbf{C}\mathbf{X}(t_n) + \mathbf{U}(t_{n+1})$$
$$\Rightarrow \mathbf{A}\mathbf{X}(t_{n+1}) = \mathbf{B}\mathbf{X}(t_n) + \mathbf{I}\mathbf{U}(t_{n+1}) \quad (2)$$

where h is the time-step used for the numerical integration, $\mathbf{A} = \mathbf{G} + \frac{1}{h}\mathbf{C}$, $\mathbf{B} = \frac{1}{h}\mathbf{C}$, and $\mathbf{I}$ is the identity matrix. Here, t_n denotes the nth time point during the transient simulation. The system of linear equations in (2) can be solved using direct or iterative solvers. However, there are inherent disadvantages in using either approach. Direct solvers typically involve some form of sparse factorization of matrix $\mathbf{A}$, such as LU decomposition [28], and the subsequent execution of forward-backward substitutions for each time point in the simulation time interval to obtain the temperatures of the IC as a function of time. Both the factorization and the forward-backward substitutions are essentially serial operations and cannot be implemented on parallel computing platforms to speed up the simulations. On the other hand, iterative techniques such as Gauss–Jacobi relaxation are memory efficient and highly parallelizable but suffer from very slow convergence when applied to problems containing smooth spatial distribution of state variables such as temperatures in an IC [29]. In addition, these methods are very sensitive to changes in the inputs to the system and hence, would become very slow if power inputs change drastically or if the floorplan configuration of the IC is changed, as is commonly done during design space exploration.

We propose an alternative to these approaches, which is a thermal simulator based on artificial NNs. In the next section, the theory and the implementation of the proposed NN-based thermal simulator will be described and its advantages over direct and iterative solvers will be demonstrated. For this paper, the thermal grids for the test cases were built and the training data for the NN was generated using 3D-ICE [9], an open source thermal simulation software based on the compact modeling technique.

IV. NN-Based Thermal Simulation

Artificial NNs are multi-input multi-output operators, which can be trained to mimic the behavior of any mathematical function through learning the input-output dependencies of that function from some test data. Fig. 3 shows a simple 4-input 3-output linear NN. The various inputs of this NN are connected to the outputs via weighted links. Hence, the outputs

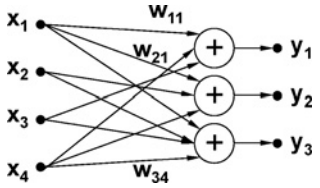

Fig. 3. Simple linear NN.

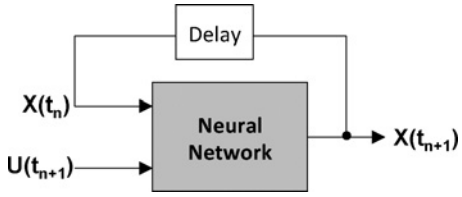

Fig. 4. Linear NN for thermal simulation of ICs.

can be expressed as a weighted sum of the inputs as follows:

$$y_i = \sum_j w_{ij} x_j \quad (3)$$

where w_{ij} connects input x_j to output y_i. A zero weight indicates the absence of a connection, meaning the input under consideration does not influence the output.

The thermal model for an IC structure is a deterministic system. More specifically, we assume that the thermal properties of the materials in an IC do not vary with temperature, as it occurs in large ranges of operating conditions of ICs. These thermal properties could be calculated for a worst-case IC operating temperature. Therefore, the equations in (1) represent a LTI system that can be represented using a linear single layer NN. The goal of our NN-based simulator, as shown in Fig. 4, is to approximate the following function derived from (2):

$$\mathbf{X}(t_{n+1}) = \mathbf{P}\mathbf{X}(t_n) + \mathbf{Q}\mathbf{U}(t_{n+1}) \quad (4)$$

where $\mathbf{P} = \left(\mathbf{G} + \frac{1}{h}\mathbf{C}\right)^{-1}\frac{1}{h}\mathbf{C}$ and $\mathbf{Q} = \left(\mathbf{G} + \frac{1}{h}\mathbf{C}\right)^{-1}$. The NN here is trained using 3D-ICE. Once trained, the weights of this NN reflect the coefficient of these matrices, and the NN is able to run as a stand-alone simulator on massively parallel computing platforms, such as GPUs, to give significant speedups when compared to the conventional techniques. The accuracy and the computational advantages of the proposed NN-based simulator come from various aspects of our model and will be discussed in the ensuing subsections.

A. Training of the NN

Learning in NN essentially consists of finding the correct set of weights for each connection in the NN, such that the input-output relationship of the system is emulated accurately for all possible cases. In other words, in a fully connected NN for the thermal simulation of an IC, the weights correspond to the elements of matrices $\mathbf{P}$ and $\mathbf{Q}$ in (4). Training is performed by supplying to the NN a finite set of inputs and outputs from 3D-ICE over a specific simulation interval (N_{Train} time points) as shown in Fig. 5. During each training iteration, the outputs from NN and the target output from 3D-ICE are compared and

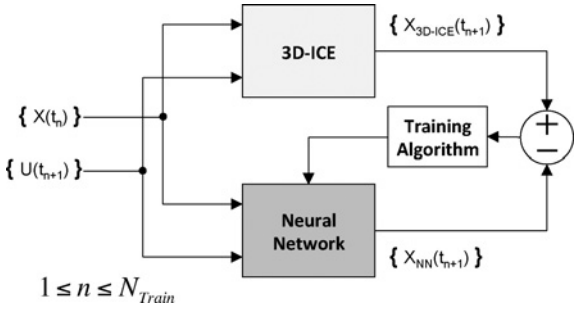

Fig. 5. NN-based thermal simulator trained using 3D-ICE.

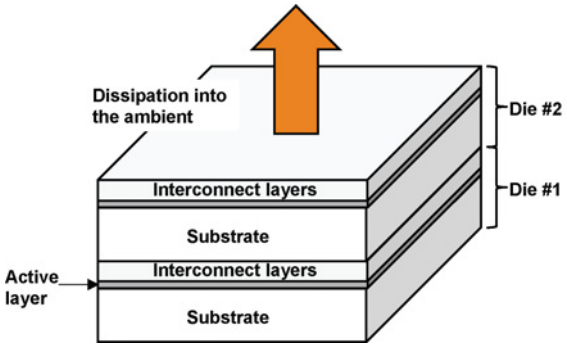

Fig. 6. Layers in a 3-D IC and the path of heat dissipation.

based on the error incurred, the weights of the NN are updated using some training algorithm and the new set of outputs from NN are compared again with 3D-ICE. These iterations are continued until convergence is reached based on some predefined error tolerance. Once the training is finished, these weights can be stored and can be reused for future simulations.

ICs are heterogeneous structures and consist of layers of different materials. The composition of a given IC affects its thermal characteristics. Hence, all the structural and material heterogeneity of the IC must be taken into account in the construction of the compact model (Section III) for accurate simulation. However, designers who want to study the design-time or run-time thermal characteristics of an IC are typically interested only in the temperatures of the active regions of the IC, that are the layers of silicon where the devices are fabricated and where bulk of the power is dissipated, as shown for a two-die 3-D IC stack in Fig. 6. Moreover, during the design space exploration, typically only the floorplan configuration and the routing parameters are varied while keeping all the technological parameters, such as the materials, the thickness of substrate, the number of interconnect layers, and so on constant. Hence, it would be sufficient to train and run the NN to simulate temperatures for only the active layers. In addition, since the thermal grid is assumed to be an LTI system, the sources representing the ambient can be eliminated by considering a zero ambient temperature, invoking the principle of superposition, and later adding the actual ambient temperature to the results of the simulation. Hence, the number of neurons (outputs) in the reduced NN would equal the number of thermal cells in the active layers of the IC, and the number of inputs would be equal to twice that number, namely, the temperatures of the cells in the active layer from the previous time point plus the sources representing the power dissipation for the same cells. This results in the minimum possible problem size, greatly reducing the computational complexity of the NN.

B. Training Length and Method

Since the intended application for the NN is to replace a conventional thermal simulator, the training process must ensure that the NN is capable of predicting temperatures for all possible power inputs and initial temperature states. Hence, a minimum number of time points of temperature data (N_{Train}) must be supplied to the training algorithm for a comprehensive training. This minimum number of training points ($N_{\text{Train,min}}$) depends upon the nature and the size of the thermal grid being solved. The method to compute it is as follows.

In a NN consisting of l neurons (outputs) $\{y_1, y_2..., y_i..., y_l\}$ and m inputs $\{x_1, x_2..., x_j..., x_m\}$, there are a total of lm weights to be computed, assuming a fully connected network. However, as it can be seen from (3), each neuron y_i is affected by only m weights $w_{ij}, 1 \leq j \leq m$. Hence, the training of these m weights is dependent only upon the output y_i and the corresponding target output t_i from 3D-ICE. That is, each set of m weights is trained simultaneously within the set, and is independent of the other sets. Each training time point provides information about how a particular combination of known inputs $\{x_1, x_2..., x_j..., x_m\}$ results in a known target output t_i. These form the coefficients in the training algorithm. Since there are m unknown variables (weights) per neuron, at least m equations, or training time points, are needed to find a unique solution. In other words

$$N_{\text{Train,min}} = m. \tag{5}$$

In our thermal simulations, given n_{cells} number of thermal cells in an active layer of an IC, there are $2n_{\text{cells}}$ inputs to the NN (past temperatures plus the power inputs for each cell), related to outputs as follows:

$$\mathbf{X}(t_{n+1}) = \mathbf{W} \cdot \begin{bmatrix} \mathbf{X}(t_n) \\ \mathbf{U}(t_{n+1}) \end{bmatrix} \tag{6}$$

where $\mathbf{W}$ is a $n_{\text{cells}} \times 2n_{\text{cells}}$ matrix containing the weights of the NN. Hence, $N_{\text{Train,min}} = 2n_{\text{cells}}$. However, it is recommended to train the NN using a higher number of data points to hasten the convergence of the training algorithm and to obtain a much more accurate solution when simulating. In our experiments we used a value of N_{Train} which is 20%–50% higher than $N_{\text{Train,min}}$.

Since a simultaneous training of the weights over the entire training data set must be performed to obtain the correct solution, a batch training algorithm (as opposed to an incremental training algorithm) must be used. In all our experiments we used the RPROP batch training algorithm [30], which is one of the fastest training algorithms available.

C. Proximity Based Model Reduction

A significant reduction in computational and memory complexity can be achieved by relaxing the requirement of a fully connected NN for thermal simulation. Given the diffusive

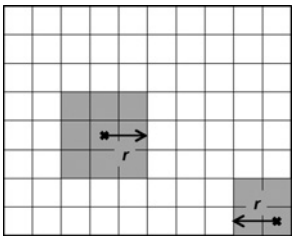

Fig. 7. "Neighborhoods" of neurons in a floorplan.

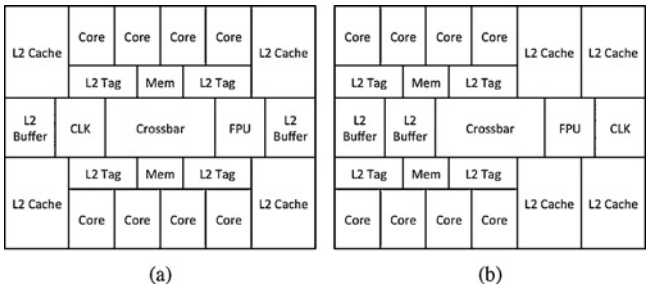

Fig. 8. Floorplan of the UltraSPARC T1 (Niagara-1) chip. (a) Original and (b) alternate.

nature of the heat flow in an IC, as shown in Fig. 6, much of the heat flows vertically upward from the source to the ambient, following the path of least resistance, with little lateral spreading of heat. In other words, there is very little heat flow/interaction between thermal cells that are far apart within the same layer. Hence, the connection of individual neurons in the network can be limited to neighbors lying within some distance r from each other, as defined by the user.

In our experiments, given the floorplan of an IC discretized into thermal cells for compact modeling as shown in Fig. 7, rectangular regions of "neighborhood" were defined around each neuron based on r, and only the cells lying within this region were connected to the neuron in the network. Two such neurons (cells on the floorplan) and their corresponding neighborhoods are highlighted in Fig. 7. Hence, the $\mathbf{W}$ matrix in (6) would no longer be a full-matrix but very sparse, leading to considerably lower memory consumption, and faster training and simulation using the NNs. However, this approximation still leads to a certain error, which we quantitatively analyze in Section VI.

D. Randomization of Training Input

One of the major advantages of the proposed NN-based approach is the reusability of the NN, to enable fast design space exploration of ICs for thermal reliability. Hence, the NN, once trained, must be capable of simulating with accuracy a variety of floorplan configurations. This, in turn, means that the learning of the NN must incorporate uncertainty of future floorplan configurations and power dissipation patterns. If the NN is trained for a particular floorplan, say that of the UltraSPARC T1 (Niagara-1) architecture from Sun Microsystems as shown in Fig. 8(a) [2], [31], then all thermal cells which lie within one of its elements would always be fed with identical power values at all times. As a result, the weights associated with these inputs would be identical

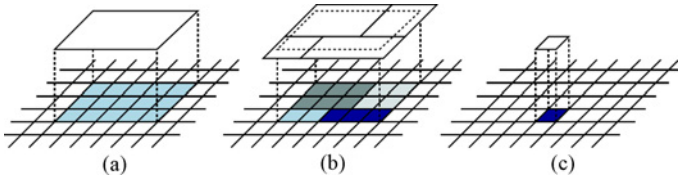

Fig. 9. Thermal cells receiving the same power inputs while (a) training and different power inputs while (b) simulating. The solution to handle the worst case: (c) a different floorplan element for every thermal cell.

to each other at the end of the training since any training algorithm, in response to identical input values from multiple sources, would act in the same manner upon each of them.

To illustrate this, consider Fig. 9(a), which shows the relation between a floorplan element and the thermal cells inside it. The individual floorplan elements, such as the one shown in this figure, are projected onto a grid of thermal cells in the discretized IC structure. At any time point during the preparation of the training set, the heat dissipated by this floorplan element is divided equally between all the highlighted cells in this figure. The range of inputs given to this group of cells is then limited by the maximum power density (heat dissipation per unit area) of the floorplan element. If the training is performed using this particular floorplan configuration, then all the highlighted thermal cells receive identical power values during training and hence, the weights connecting these inputs to the outputs in the NN would turn out to be identical. However, if this NN is used to simulate another floorplan configuration (even if the new configuration is obtained by rearranging the same floorplan elements), the thermal cells that had previously belonged to a specific floorplan element used for training might now receive inputs belonging to a different range of values pertaining to a different floorplan element. This scenario is illustrated in Fig. 9(b), where the same group of thermal cells is split into four different blocks. Large errors would result during run-time because the NN training process was blind to this scenario of nonidentical power values amongst these sets of thermal cells. In addition, if the maximum power density of one of the four new floorplan elements in this location is higher than the power density of the floorplan element in Fig. 9(a) used for training, then the neurons that compute the temperature of these cells might receive as input a value that has not been considered for them while generating the training set. This would also result in significant errors during the simulation.

One straightforward way to solve this problem is to randomize the inputs during training. That is, once the discretization size of the thermal cells is fixed, each thermal cell in all the active layers must be given different and random input power values in order to train the NN for the worst-case situation where every thermal cell receives inputs from a different floorplan element. This solution is represented in Fig. 9(c). Once the NN is trained in this manner, virtually any floorplan configuration can be simulated with the error being bounded purely by the training algorithm. In addition, each of these floorplan elements must be given the entire range of power density values that could be encountered in future floorplan configurations. For this, in our experiments, the maximum possible power density (heat dissipation per unit area) for

any floorplan element was used as the benchmark. During the training process, an input power density that is higher than this value was applied to the entire active layer, and the input power value for each thermal cell was randomized between zero (to account for idle times of different floorplan elements when they do not dissipate any significant energy) and the selected power density. This ensures that the training is comprehensive and covers all scenarios. For the Niagara chip described above, the maximum possible power density for any floorplan element was found to be about $37\,\mathrm{W/cm^2}$. Hence, a power density higher than this value was used in all our trainings.

E. Convergence of the NN Training

For the proposed NN model to work as an independent thermal simulator, it is essential that the training process converges under all scenarios. This subsection provides the proof of the proposed model's convergence. For this, let us reconsider (6) describing the NN-based modeling of the thermal behavior of the system. For simplicity of illustration, let there be only two temperature nodes in the system. Hence, the equations for the proposed NN-based model can be written as follows:

$$\begin{bmatrix} x_1(t_{n+1}) \\ x_2(t_{n+1}) \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{bmatrix} \begin{bmatrix} x_1(t_n) \\ x_2(t_n) \\ u_1(t_{n+1}) \\ u_2(t_{n+1}) \end{bmatrix}. \quad (7)$$

As described in Section IV-B, the training of each row of weights in this system is independent of the other. Hence, it is sufficient to consider the convergence of one of the rows. For instance, given four training (consecutive) input data $\{x_1(t_{1-4}), x_2(t_{1-4}), u_1(t_{2-5}), u_2(t_{2-5})\}$, and the corresponding four training target data, $\{x_1(t_{2-5})\}$, the NN equations for the first row can be written as follows:

$$\begin{aligned} x_1(t_1)w_{11} + x_2(t_1)w_{12} + u_1(t_2)w_{13} + u_2(t_2)w_{14} &= x_1(t_2) \\ x_1(t_2)w_{11} + x_2(t_2)w_{12} + u_1(t_3)w_{13} + u_2(t_3)w_{14} &= x_1(t_3) \\ x_1(t_3)w_{11} + x_2(t_3)w_{12} + u_1(t_4)w_{13} + u_2(t_4)w_{14} &= x_1(t_4) \\ x_1(t_4)w_{11} + x_2(t_4)w_{12} + u_1(t_5)w_{13} + u_2(t_5)w_{14} &= x_1(t_5). \end{aligned} \quad (8)$$

Hence, the NN training process is akin to solving the above system of linear equations for the unknown vector $\mathbf{w}$ written as follows:

$$\mathbf{Mw} = \mathbf{N} \quad (9)$$

where $\mathbf{w} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \end{bmatrix}^T$.

The physics and the deterministic nature of the system require the weights $\mathbf{w}$ to be unique, i.e., at the end of the training, the weights converge to the elements of the matrices $\mathbf{P}$ and $\mathbf{Q}$ in (4). Once so trained, these weights should mimic the LTI thermal system *exactly*. These unique weights can be calculated by ensuring sufficient variance in the training set such that the equations in (8) are linearly independent and well-conditioned. This is part of the motivation behind randomizing the training data set in Section IV-D.

Now, given that a system of linear equations is linearly independent, it is well known that a method like least squares can be used to solve them since there is a unique local (and global) minimum for the residual of the system, which is equal to zero. Hence, for any training method which minimizes the gradient of the error function at each step of solving (such as the gradient descent algorithm), convergence is guaranteed for such a system. The NN training process essentially does this. Since the training algorithm RPROP [30] used in our model works toward minimizing the gradient of the error, convergence in our model is also guaranteed.

However, it must be noted that the quality of the output (i.e., the error in the final solution during run-time) and the speed of convergence during the training process depends upon: 1) the approximations applied to the structure of the system and the quality of the training sample, and 2) the nature of the training algorithm. First, depending upon the approximations applied to the relationship between inputs and outputs in the NN (refer to Section IV-C), it is possible that the unique local minimum (the error function during the training process) might not approach zero, but some finite number. This is not an artifact of the training process but a limitation imposed by the physics of the problem and the structure/complexity of the NN used to represent it. This can be solved by increasing the complexity of the NN, by increasing the "neighborhood distance" r (see Section IV-C) depending upon the accuracy/speed requirements of the user. In addition, sufficient variability in the training samples is required for faster convergence and accuracy. This is because diverse input power values result in a well-conditioned set of coefficients in the system of equations to be solved [rows and columns of $\mathbf{M}$ in (9)], aiding the training process. This can be ensured by applying a very wide range of random power input values for the system during the training process. Second, the nature of the training algorithm (essentially the algebraic space in which it looks for a solution of weights for the NN) might limit its capacity to approach the exact solution vector indefinitely, and instead be constricted to some finite distance away from it (i.e., the solution vector is a non-Cauchy sequence, with the error function oscillating between two values around the minimum). This can be solved by combining multiple training algorithms in the training process, each complementing the other for speed and accuracy. It must be remembered that, in either scenario described above, there is still a convergence.

A detailed analysis of the numerical stability of the proposed NN-based model is provided in the appendix, while a study of the final error as a function of the approximations introduced in the model is presented using experimental results in Section VI. Our results indicate that the rate of the convergence depends upon the quality of the training inputs, the size of the problem, and the number of coefficients that need to be computed.

V. Implementation of the Proposed NN-Based Simulator on GPUs

The proposed NN for thermal modeling of complex integrated circuits has been implemented for running on modern

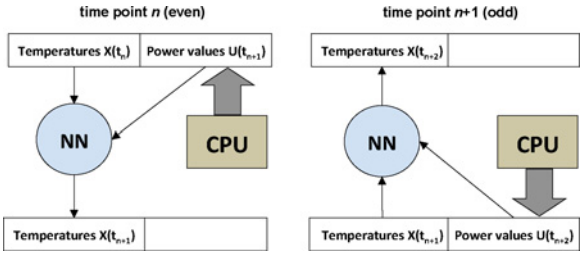

Fig. 10. Memory usage in the GPU implementation of the NN-based simulator.

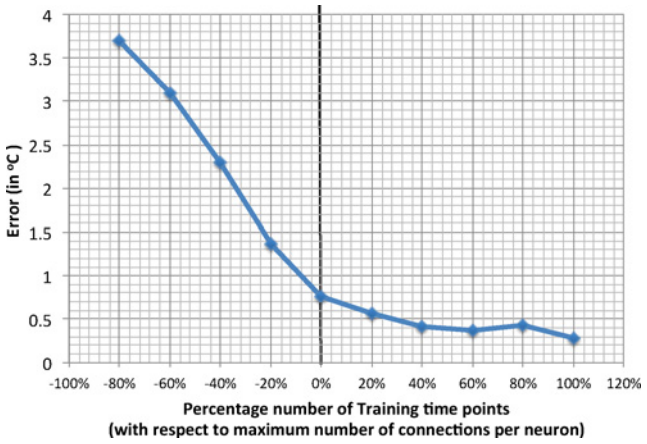

Fig. 11. Run-time error of the NN-based simulator as a function of the number of training time points of data supplied.

GPUs. Since the training phase is needed just once for a given IC, we focused more on the porting of the NN execution on GPU. Clearly, our ultimate goal is to speed up the thermal-aware design time for complex ICs, which can require several NN runs with different floor plan configurations.

A. GPU Architecture and CUDA Programming Model

The Fermi-based GPU used in this paper is an NVIDIA GeForce GTX 480 [32], a parallel computing machine comprising two levels of shared memory and 480 streaming processors (SPs) organized in 15 streaming multiprocessors (SMs). Streaming multiprocessors manage the execution of programs using "warps"—groups of 32 threads. Each SM features two warp schedulers and two instruction dispatch units, allowing two warps to be issued and executed concurrently. All instructions are executed in a single instruction multiple data fashion, where one instruction is applied to all threads in the warp. This execution method is called single instruction multiple threads. All threads in a warp execute the same instruction or remain idle (different threads can perform branching and other forms of independent work). Warps are scheduled by special units in SMs in such a way that, without any overhead, several warps execute concurrently by interleaving their instructions.

From the point of view of writing application codes, it is important to take into account the organization of the work, i.e., use 32 threads simultaneously. The code that does not break into 32 thread units can have a lower performance. The hardware chooses which warp to execute during each cycle, and it switches between them without penalties. If we compare with CPUs, this process is similar to the simultaneous execution of 32 programs that (can) switch at every cycle without penalties. CPU cores can indeed execute only one program at a time and switching to other programs costs hundreds of cycles.

B. Runtime Execution of NN on GPU

Once the NN has been trained, the entire NN is described by means of the weight matrix $\mathbf{W}$. This sparse matrix is stored in the GPU global memory according to the compressed sparse row format as required by the cuSparse library [33]. Every time-step of the thermal simulation corresponds to a matrix-vector multiplication between matrix the $\mathbf{W}$ and the input vector, which consists of the temperature in the previous step and power input for the different cells. The matrix-vector multiplication is performed using the function cusparseDcsrmv of cuSparse library [34]. It computes the operation $y = \alpha Ax + \beta y$. α will be set to 1 while β will be 0 to erase the previous content of y and overwrite the result. This function call requires three main parameters: the matrix $\mathbf{A}$ (l rows $\times$ m columns), an input vector x (m elements), and an output vector y (l elements).

Every input vector stores all the temperatures $\mathbf{X}(t_n)$ in the first half and all the power values $\mathbf{U}(t_n)$ in the second half (see Fig. 10). The matrix-vector multiplication between the weights matrix and this input vector gives the temperatures $\mathbf{X}(t_{n+1})$ in one simulation step. This set of values represents the thermal state to be used as input again in the following simulation step. Therefore, to avoid one memory copy at each step, we allocate on GPU two vectors with as many elements as the number of inputs and we swap their address at every iteration.

Before starting the simulation, we download the initial thermal state of the chip at the beginning of the first input vector in the GPU global memory. When the simulation starts, at every step, the CPU computes and downloads to GPU the new power values, writing them in the second half of the current input vector. CPU then calls the cusparseDcsrmv function to execute the matrix-vector multiplication and to write the result at the beginning of the other input vector. Before starting the next iteration, we swap the addresses of the two vectors in such a way that the vector used as output and containing the updated thermal state will be filled with the new set of power values and used as input. The vector used as input in the previous step will behave, as a consequence, as the output vector in the next step.

VI. Experimental Results

Since the results for 2-D ICs can be extrapolated to 3-D ICs, in the ensuing experimental studies, we used a 2-D IC as an illustrative example for the sake of simplicity (henceforth referred to as the *Test IC*). Test IC contains two layers: one made of silicon with the UltraSPARC Niagara floorplan [Fig. 8(a)] fabricated on it and the other an interconnect layer conducting heat to the ambient. As explained in Section IV-A,

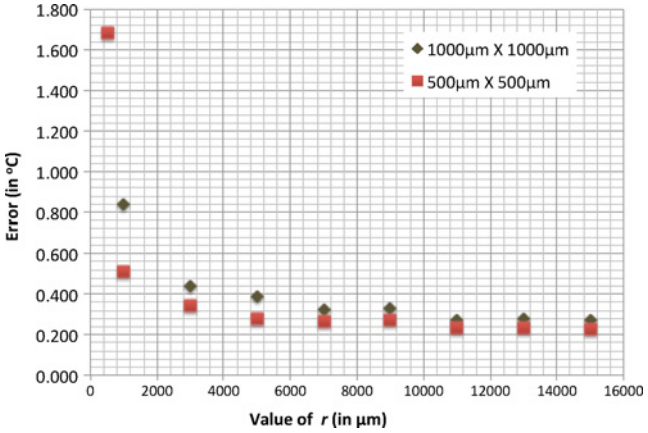

Fig. 12. Run-time error versus neighborhood distance r.

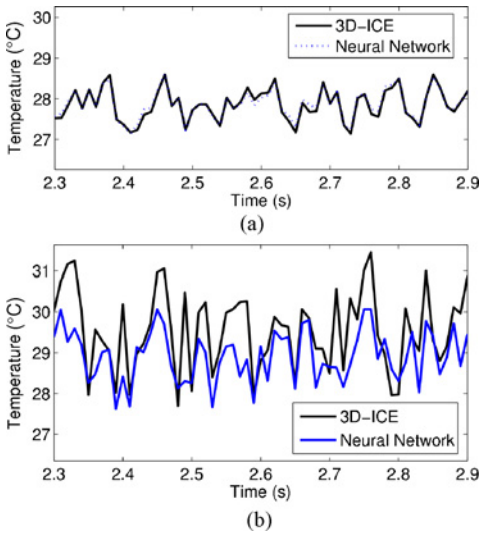

Fig. 13. Comparison of temperature waveforms for Case (a): the same training-time and run-time floorplans, and Case (b): different training-time and run-time floorplans. (a) Training with the floorplan Fig. 8(a) and run with Fig. 8(a). (b) Training with the floorplan Fig. 8(a) and run with Fig. 8(b).

the NN in each experiment was trained using the 3D-ICE simulator [9]. Each of the following experiments demonstrates the basis of the various aspects of our implementation, and highlights the resulting advantages in computational complexity and accuracy of the proposed NN-based thermal simulator.

A. Training Length

The length of the training data set and the use of a batch training scheme are fundamental to the accuracy and reliability of the proposed NN-based thermal simulator. As discussed in Section IV-B, temperature and input data for a minimum number of training time points ($N_{\text{Train,min}}$) that equals the maximum number of weights (or connections) for any neuron in the NN must be supplied for a complete learning of the weights (note that every neuron in a floorplan has a different number of neighbors as shown in Fig. 7), namely

$$N_{\text{Train,min}} = \max_i m_i. \tag{10}$$

TABLE I
ERROR AND NUMBER OF WEIGHTS VERSUS NEIGHBORHOOD DISTANCE r

Distance r	1000 μm × 1000 μm		500 μm × 500 μm	
	Error (K)	# Weights	Error (K)	# Weights
500 μm	–	420	1.684	14k
1000 μm	0.841	3440	0.512	38k
3000 μm	0.441	16k	0.345	224k
5000 μm	0.387	33k	0.278	497k
7000 μm	0.325	52k	0.263	794k
9000 μm	0.333	68k	0.270	1.07M
11 000 μm	0.272	80k	0.233	1.27M
13 000 μm	0.279	87k	0.231	1.39M
Fully connected	0.270	88k	0.225	1.41M

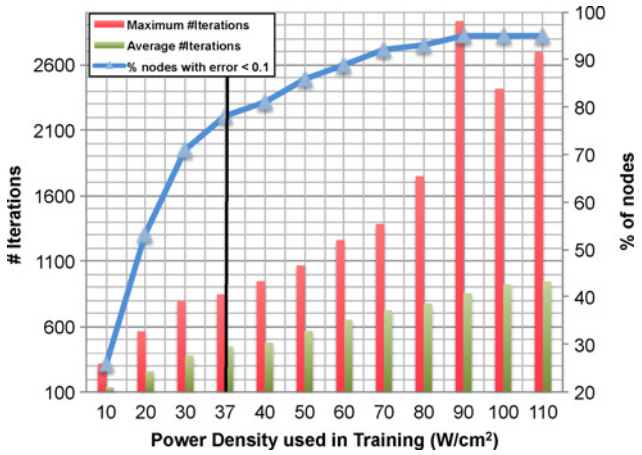

Fig. 14. Maximum temperature error, maximum number of training iterations, and average number of training iterations versus maximum training-time power density applied.

To illustrate this, the Test IC was discretized into thermal cells of dimensions 1000 μm × 1000 μm and simulated using 3D-ICE for approximately 400 time points, and the temperature and input data were stored.

Next, a NN was created for this experiment, as described in Section IV-C with a neighborhood distance r = 2000 μm. Next, a series of training routines were run using N_{Train} data points from 3D-ICE, where N_{Train} was either lower than, equal to or higher than $N_{\text{Train,min}}$. Post training, the NN was run for the remaining inputs in the data and the resulting maximum run-time temperature error with respect to the output from 3D-ICE (over all time points and all neurons) was recorded. These results are plotted in Fig. 11. Here, a negative percentage value in the x-axis, for example -80%, means the N_{Train} is 80% less than $N_{\text{Train,min}}$. 0% indicates that $N_{\text{Train}} = N_{\text{Train,min}}$, and a value of +40% indicates $N_{\text{Train}} = 1.4N_{\text{Train,min}}$. As Fig. 11 shows, the error drops quickly as N_{Train} approaches $N_{\text{Train,min}}$ and then remains fairly constant, indicating the significance of this training criteria.

B. Effect of Proximity Based Model Reduction

As described in Section IV-C, exploiting the physics of heat transfer in an IC to reduce the connectivity in the proposed NN-based simulator greatly reduces memory consumption and computational complexity. However, its effect on the accuracy of the results must be first studied before using it as an effective simulation strategy. For this purpose, two different cases

TABLE II
SIMULATION TIME COMPARISON FOR 54K TIME POINTS FOR VARIOUS NEIGHBORHOOD DISTANCES r

Discretization	Distance r	Simulation Time (s)		
		3D-ICE	NN-CPU	NN-GPU
1000 μm × 1000 μm	1000 μm	21.21	5	1.92
	3000 μm		7	1.89
	5000 μm		9	2.08
	7000 μm		11	2.24
	9000 μm		13	2.44
	11 000 μm		15	2.53
	13 000 μm		16	2.5
	Fully connected		16	2.53
500 μm × 500 μm	1000 μm	98.45	12	2.88
	3000 μm		34	3.34
	5000 μm		44	4.79
	7000 μm		70	6.37
	9000 μm		96	7.61
	11 000 μm		102	8.61
	13 000 μm		107	9.12
	Fully connected		107	9.06

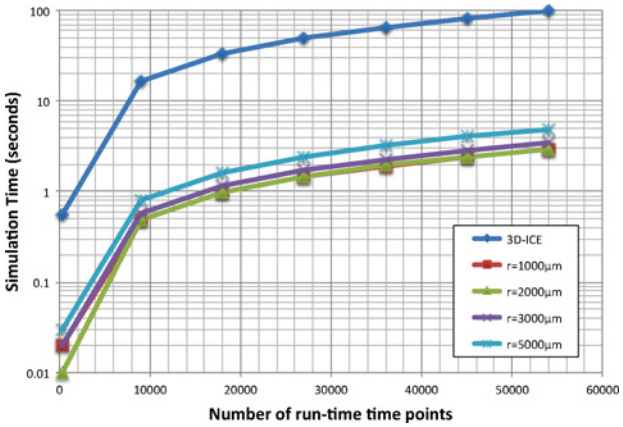

Fig. 15. Simulation time comparison between 3D-ICE (run on CPU) and the various cases of NN-based thermal simulators (run on GPU).

were studied, namely: 1) Test IC discretized into thermal cells of dimensions 1000 μm × 1000 μm, and 2) Test IC discretized into thermal cells of dimensions 500 μm × 500 μm. In each case, the neighborhood distance r was increased from 500 μm up to an area covering the full chip (i.e., a fully connected NN, where each output depends upon all the inputs in the IC).

In each case, training was performed using 3D-ICE and the NN-based simulator was then run independently for 10 000 time points. Results from run-time were compared with the corresponding output from 3D-ICE and the maximum temperature error between the two were recorded. These results are shown in a scattered plot in Fig. 12. The results are also tabulated in Table I along with the corresponding number of NN weights to be computed for each case. As our results outline, even a very loosely connected network, such as r = 3000 μm, results in negligible error (<0.5 °C) while resulting in large computational and memory savings. The case of 1000 μm × 1000 μm with r = 500 μm is not shown to preserve the scale of the plot, since the run-time error began to explode as soon as the simulation started.

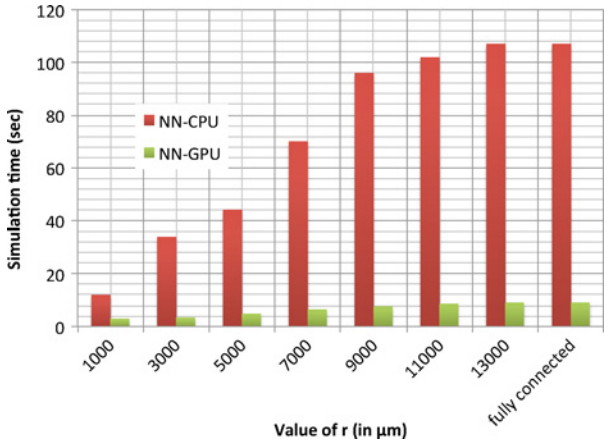

Fig. 16. Simulation time comparison between NN run on CPU and GPU.

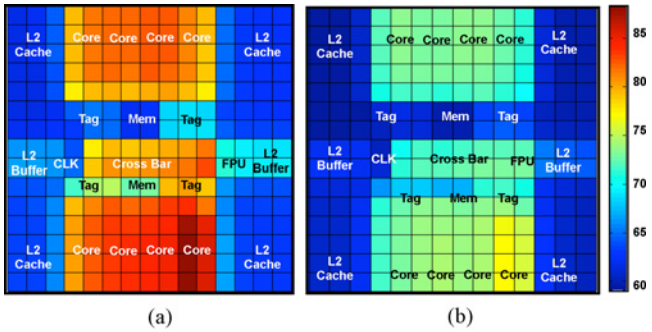

Fig. 17. Thermal maps of two UltraSPARC T1 (Niagara) chips stacked one on the top of the other (Case A in Section VI-E). (a) Bottom die. (b) Top die.

C. Effect of Randomization of Training Input

As described in Section IV-D, for the NN-based simulator to be reusable (i.e., able to work on any floorplan configuration), the training must be performed on an input data set that covers all the possible scenarios during run-time. To illustrate this problem, the NN-based simulator was trained for the Test IC with the floorplan configuration as shown in Fig. 8(a), with $N_{\text{Train}} = N_{\text{Train,min}}$. Once trained, the NN-based simulator was used to simulate two cases: 1) a Test IC with the same floorplan configuration and power dissipation functions as Fig. 8(a), and 2) another Test IC, which contains the same floorplan elements and the power dissipation functions for the elements as the original floorplan but having a slightly different floorplan configuration as shown in Fig. 8(b).

The resulting run-time temperature behavior comparison between 3D-ICE and the NN-based simulator for both cases are shown in Fig. 13(a) and (b), respectively. The maximum run-time temperature error for case 1) was 0.26 °C while that for case 2) was 2.51 °C, about an order of magnitude difference.

Next, the training was performed, as described in Section IV-D, using distinct and random input power dissipation functions for each thermal cell (neuron) in the floorplan, and then the simulator was used to simulate the original Niagara floorplan in Fig. 8(a). To see the effect of the maximum amplitude of the training power values on the run-time accuracy, different power densities (power dissipated per unit area) were

applied to the Test IC and randomly distributed among the thermal cells during the training. The maximum power density for any floorplan element in the Niagara floorplan (computed from the steady state power dissipation data for UltraSPARC T1 [2], [31]) was found to be 37 W/cm^2. This means that the run-time power density would not exceed this value anywhere in the IC. Hence, the above experiments were run with a set of training-time power densities lower than, equal to, and higher than this value.

The resulting run-time error distribution for each case is plotted in Fig. 14. This figure shows the percentage of nodes in the system which show an error less than 0.1 °C. In addition, the maximum and average number of RPROP training iterations required for each experiment is plotted in the same graph. The case of the training-time maximum power density equaling the run-time maximum power density has been highlighted. Our results indicate that, while the percentage of nodes which show an error less than 0.1 °C increases and converges to 100%, the number of training iterations (and hence the computational effort) also increases with increasing training-time power density. Hence, an optimal point, depending upon the requirements of the user, can be attained given that the run-time maximum power density of an IC is known.

D. Speedups Using the Proposed NN-Based Thermal Simulator

To illustrate the simulation time savings of the proposed approach compared to the conventional techniques, the NN-based simulator was trained and then run on the GeForce GTX 480 GPU platform with 480 CUDA cores running at 1.4 GHz with 1.5 GB GDDR5, and the simulation times for various numbers of time points of simulation were compared with the corresponding simulation using 3D-ICE run on Intel(R) Core(TM) i7 920 2.67 GHz processor with four cores and 6 GB RAM. The resulting simulation times recorded for the case of 500 μm × 500 μm discretization with various neighborhood distances r are plotted in Fig. 15 on a semilog scale, as a function of the number of time points in the simulation. In our experiments, the NN-based simulators were found to be up to 35× faster than 3D-ICE during runtime. This speedup is primarily achieved due to the extreme parallelizability of the NN-based simulator as opposed to the 3D-ICE run-time operations (forward-backward substitutions of matrix factors), which are serial in nature.

Next, to highlight the need for GPUs in the proposed approach, the same NN-based simulators trained for Test IC with 500 μm × 500 μm discretization were run on both the CPU platform [four threads running on the Intel(R) Core(TM) i7 920 2.67 GHz processor] and the GPU platform (the GeForce GTX 480 processor), and the simulation times were compared. The results (for a simulation of 54k time points) are plotted in Fig. 16 as a function of the neighborhood distance r. In addition, the simulation times from the above experiments are tabulated in Table II.

E. Simulation of a 3-D IC

1) *Case A:* As mentioned earlier, the analytical study and the implementation of NN-based simulator as applied to 2-D ICs above can be extrapolated in a straightforward manner to 3-D ICs. In order to demonstrate this, a test 3-D IC was considered by stacking two Test ICs one on top of the other [both containing the UltraSPARC T1 floorplan configuration as shown in Fig. 8(a)]. Heat dissipation values were obtained from measurements of the switching activity of the individual elements [31]. A neighborhood distance of $r = 4000\,\mu$m was found to be sufficient to generate accurate results. The NN-based simulator was trained with a data set containing 20% more points than $N_{\text{Train,min}}$. Post training, the simulator was run for 400 time points and the results were compared with 3D-ICE. The resulting temperature distribution map for the two active layers of Test 3-D IC is shown in Fig. 17. The results from the NN-based simulator matched well with 3D-ICE and at the end of the simulation, 88.2% of the neurons were found to show an error less than 1 °C.

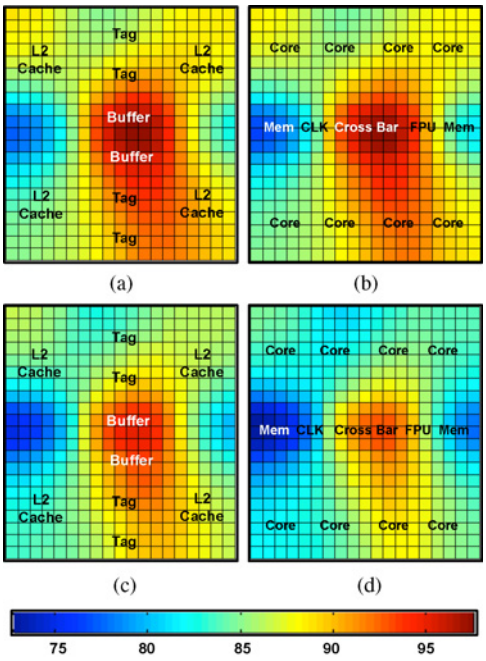

Fig. 18. Thermal maps of four dies (two cores and two memories with floorplan architecture from [35]) stacked one on the top of the other (Case B in Section VI-E). (a) Die #1. (b) Die #2. (c) Die #3. (d) Die #4.

2) *Case B:* Another test stack was created with 4-dies. This time, the floorplan was changed to the one discussed in [35]—splitting the cores and the memories of the Sun UltraSPARC Niagara architecture into separate dies. The test stack contained two dies made up of cores and the other two dies made up of caches. Each die in this example is 1 cm × 1.1 cm in size. Power traces were obtained, again, from the measurements of the switching activity. As in the previous example, a neighborhood distance of $r = 6000\,\mu$m was found to be sufficient to generate accurate results. The NN-based simulator was trained with a data set containing 20% more points than $N_{\text{Train,min}}$. Post training, the simulator was run for a simulation time interval of 2 h of switching activity, containing 36 000 time points and the results were compared with 3D-ICE. The resulting temperature distribution map for the four active layers of this 3-D IC is shown in Fig. 18 (dies numbered

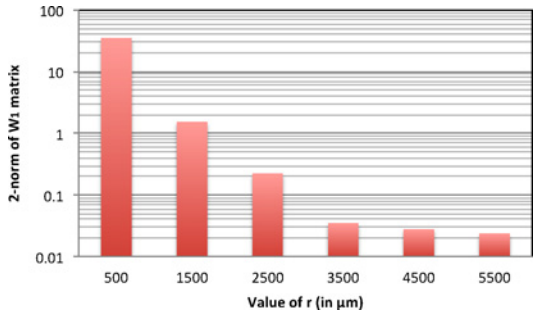

Fig. 19. 2-norm of the $\mathbf{W}_1$ matrix for the Test IC, against increasing proximity distance r.

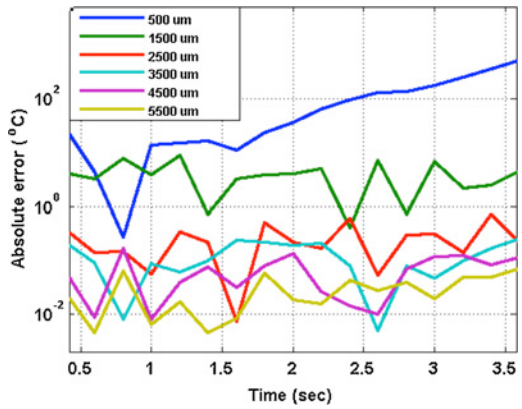

Fig. 20. Temporal evolution of the absolute value of error at a single node when the NN-models for Test IC built using various proximity distance values are used to simulate the temperatures for some random sequence of input heat fluxes.

from bottom to top). At the end of the simulation 94% of the nodes were found to show an error less than 1 °C.

VII. Conclusion

In this paper, we presented an innovative full-chip thermal modeling approach that exploits NNs and the computational power of modern GPUs. This approach tackles the scalability problem of transient heat flow simulation in large 2-D/3-D multiprocessor ICs by parallelizing a NN-based temperature tracking model on GPUs. Our experiments with realistic multicore IC designs show that the proposed approach achieves up to 35× run-time speedups, while keeping the error below 1 °C, in comparison to state-of-the-art thermal simulation tools.

Appendix
Numerical Stability of the NN-Based Model

The proposed NN-based thermal model, like all numerical techniques used for transient system analyses, incurs an error that accumulates over time. This is because the temperatures at a given time point depends partly on the temperatures at the previous time point. Therefore, part of the error incurred at the very first time point of solving is *propagated* to any future time point. This is called the *propagation error* ϵ_p. The propagation error from all previous time points together with the *local truncation error* ϵ_l incurred at each time point t_{n+1} due to the numerical approximation in the current computation, gives the *global error* ϵ_g of the simulator. Since the exact solution is usually not known, it is customary to estimate the global error as the local truncation error at a given time point, multiplied by the number of time points preceding it.

As an example, it is well known that the backward Euler method—the method used in 3D-ICE as shown in (2)—has a local truncation error of $\epsilon_{l,BE} = \mathbf{O}(h^2)$, where h is the step-size used [36]. Hence, decreasing the step-size h, the local truncation error at each time step decreases in a quadratic fashion. However, the estimate of the global error at the end of an interval of simulation of length T seconds is as follows:

$$\epsilon_g = \frac{T}{h}\mathbf{O}(h^2) = \mathbf{O}(h). \tag{11}$$

Hence, the global error at the end of simulation using the backward Euler method decreases linearly with smaller and smaller step-sizes. Using extremely small step-sizes can help reduce this error. But it comes at a huge computational overhead. Thus, what is practical—and more important—in these problems is to ensure that the accumulated error is bounded and does not increase indefinitely. In other words, it must be ensured that the method used for solving the equations is *numerically stable*.

Backward Euler method is one of the very few unconditionally stable methods. Usage of this method in 3D-ICE ensures that increasing the step-size h to any value however large (provided the resulting error is acceptable to the user) does not lead to the blowing up of the simulation results and the error is still bounded. Since the proposed NN-based model is trained using 3D-ICE, which is numerically stable, the proposed model also retains this feature. This can be shown using the following analysis.

The conditions that determine the numerically stability of a method can be phrased as follows: at each step of simulation, the contribution of error from a previous step must be progressively diminished. In other words, the propagation error term from a previous time step in the global error estimate of the current time step must approach zero as we move farther and farther away from that time step. For the proposed NN-based model, this can be illustrated by considering (6) and rewriting it by expanding the weight matrix $\mathbf{W}$ as follows:

$$\mathbf{X}(t_{n+1}) = [\mathbf{W}_1|\mathbf{W}_2] \cdot \begin{bmatrix} \mathbf{X}(t_n) \\ \mathbf{U}(t_{n+1}) \end{bmatrix} \tag{12}$$

where $\mathbf{W}_1$ and $\mathbf{W}_2$ are square matrices, containing the NN weights pertaining to the temperatures from the previous step and the heat sources, respectively. Let us assume that starting from the given initial state $\mathbf{X}_0$, the estimated temperature states for the next $n+1$ time points ($\{t_1, t_2 ... t_n, t_{n+1}\}$) are $\{\mathbf{X}_1, \mathbf{X}_2 ... \mathbf{X}_n, \mathbf{X}_{n+1}\}$ and the corresponding solution from 3D-ICE are $\{\mathbf{T}_1, \mathbf{T}_2 ... \mathbf{T}_n, \mathbf{T}_{n+1}\}$, respectively. For the purposes of the following analysis, let us consider the temperatures from 3D-ICE to be the reference. The global error, with respect to the results from 3D-ICE, at the last time point $n+1$ is thus given by

$$\epsilon_{g,n+1} = \mathbf{T}_{n+1} - \mathbf{X}_{n+1}. \tag{13}$$

From (12) and (4), the above equation can be written as

$$\epsilon_{g,n+1} = (\mathbf{P}\mathbf{T}_n + \mathbf{Q}\mathbf{U}_{n+1}) - (\mathbf{W}_1\mathbf{X}_n + \mathbf{W}_2\mathbf{U}_{n+1}). \tag{14}$$

Rearranging the terms on the right-hand side we get

$$\epsilon_{g,n+1} = \mathbf{P}\mathbf{T}_n - \mathbf{W}_1\mathbf{X}_n + \mathbf{A}_{n+1} \tag{15}$$

where $\mathbf{A}_{n+1} = (\mathbf{Q} - \mathbf{W}_2)\mathbf{U}_{n+1}$. Writing $\mathbf{P} = \mathbf{W}_2 + \mathbf{R}$, for some matrix $\mathbf{R}$, we can simplify the above as follows:

$$\begin{aligned}\epsilon_{g,n+1} &= \mathbf{W}_1(\mathbf{T}_n - \mathbf{X}_n) + \mathbf{R}\mathbf{T}_n + \mathbf{A}_{n+1} \\ &= \mathbf{W}_1\epsilon_{g,n} + \mathbf{B}_{n+1}\end{aligned} \tag{16}$$

where $\mathbf{B}_{n+1} = \mathbf{R}\mathbf{T}_n + \mathbf{A}_{n+1}$. The error at time t_n, i.e., ϵ_n in the above equation can be written similarly in terms of the error at the previous time point, ϵ_{n-1}, and so we get

$$\begin{aligned}\epsilon_{g,n+1} &= \mathbf{W}_1(\mathbf{W}_1\epsilon_{g,n-1} + \mathbf{B}_n) + \mathbf{B}_{n+1} \\ &= \mathbf{W}_1^2\epsilon_{g,n-1} + \mathbf{W}_1\mathbf{B}_n + \mathbf{B}_{n+1}.\end{aligned} \tag{17}$$

This equation can be recursively rewritten until we find the error at the last time point t_{n+1} in terms of the error at the first time point of solving t_1 as follows:

$$\epsilon_{g,n+1} = \mathbf{W}_1^n\epsilon_{g,1} + \sum_{i=0}^{n-1}\mathbf{W}_1^i\mathbf{B}_{n+1-i}. \tag{18}$$

In the above equation, the summation term on the right-hand side depends purely upon the reference solution and the given input vector. Hence, only the first term represents the accumulation of error from the first step of solving until the current step. Taking the euclidean norm on both sides and using the triangular inequality, we can write

$$\begin{aligned}\|\epsilon_{g,n+1}\| &\leq \|\mathbf{W}_1^n\epsilon_{g,1}\| + \left\|\sum_{i=0}^{n-1}\mathbf{W}_1^i\mathbf{B}_{n+1-i}\right\| \\ &\leq \|\mathbf{W}_1^n\|\|\epsilon_{g,1}\| \\ &\leq \|\mathbf{W}_1\|^n\|\epsilon_{g,1}\|.\end{aligned} \tag{19}$$

For this global error to be bounded the accumulated error term must progressively diminish with increasing n, as discussed above. In other words, the stability of the NN-model can be guaranteed if the weight matrix satisfies $\|\mathbf{W}_1\|^n \rightarrow 0$ as $n \rightarrow \infty$, or

$$\|\mathbf{W}_1\| < 1. \tag{20}$$

This sufficient condition for numerical stability is satisfied by the matrix $\mathbf{P}$ in (4), because by definition, it is an operator for a numerically stable backward Euler method. In our proposed NN-based model, the goal of the training is to match the matrix $\mathbf{W}_1$ with this matrix as closely as possible (in other words, $\mathbf{R}$ must be ideally equal to zero or negligible). The idea of proximity-based model reduction (described in Section IV-C) approximates this matching process giving memory and computational savings. That is, when a certain distance of proximity is applied, the effect on the training is that certain non-diagonal elements of the $\mathbf{P}$ matrix (a dense matrix), which are relatively small (due to the small contribution of the corresponding temperatures to the evolution of temperature at a given location owing to the large distances between them) are neglected. Hence, the numerical stability of the resulting NN-based model depends purely on the errors introduced by this approximation, assuming no other errors are introduced in the training process. We have studied the relationship between the stability of the NN-model and the proximity-based model reduction using some example as described below.

We trained and test-ran an NN-based model for a single die Test IC with a discretization size of $1000\,\mu\text{m} \times 1000\,\mu\text{m}$, increasing the proximity distance (r) from $500\,\mu\text{m}$ until $5500\,\mu\text{m}$, and observed the error at a single node in the IC as a function of time for each case. The entire test simulation run was for a simulation time interval of 40 s, with a step-time of 0.2 s (a total of 200 time points of simulation). We also computed the 2-norms of the corresponding $\mathbf{W}_1$ matrices in each case [as required by the stability criterion described in (20)]. The 2-norms and the evolution of the absolute value of error with time at a single node for each of these cases are shown in semi-log scale in Figs. 19 and 20, respectively. These figures indicate that the case of $r = 500\,\mu\text{m}$ is clearly unstable, as indicated by its 2-norm value. The case of $r = 1500\,\mu\text{m}$ is right on the border and shows large errors. Note that the errors in the temperatures reach very high values (tens of °C) even before the stability threshold is breached. Hence, in all realistic scenarios, proximity distances chosen by designers are large enough to negate the possibility of instability in the model.

REFERENCES

[1] ITRS. (2009). *International Technology Roadmap for Semiconductors* [Online]. Available: http://www.itrs.net/Links/2009ITRS/Home2009.htm

[2] A. Coskun, T. Rosing, K. Whisnant, and K. Gross, "Static and dynamic temperature-aware scheduling for multiprocessor SoCs," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 16, no. 9, pp. 1127–1140, Sep. 2008.

[3] Y. Cheng, P. Raha, C. Teng, E. Rosenbaum, and S. Kang, "ILLIADS-T: An electrothermal timing simulator for temperature-sensitive reliability diagnosis of CMOS VLSI chips," *IEEE Trans. Comput.-Aided Des. Integr. Circuits*, vol. 17, no. 8, pp. 668–681, Aug. 1998.

[4] Y. Tal and A. Nabi, "A simple analytic method for converting standardized IC-package thermal resistances (thetasja, thetasjc) into a two-resistor model (thetasjb, thetasjt)," in *Proc. ISTMM*, 2001, pp. 134–144.

[5] W. Huang, E. Humenay, K. Skadron, and M. Stan, "The need for a full-chip and package thermal model for thermally optimized IC designs," in *Proc. ISLPED*, 2005, pp. 245–250.

[6] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan, "HotSpot: A compact thermal modeling methodology for early-stage VLSI design," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 14, no. 5, pp. 501–513, May 2006.

[7] T. Wang and C. Chen, "3-D thermal-ADI: A linear-time chip level transient thermal simulator," *IEEE Trans. Comput.-Aided Des. Integr. Circuits*, vol. 21, no. 12, pp. 1434–1445, Dec. 2002.

[8] Z. Feng and P. Li, "Fast thermal analysis on GPU for 3D-ICs with integrated microchannel cooling," in *Proc. IEEE/ACM ICCAD*, Nov. 2010, pp. 551–555.

[9] A. Sridhar, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza, "3D-ICE: Fast compact transient thermal modeling for 3-D ICs with inter-tier liquid cooling," in *Proc. IEEE/ACM ICCAD*, Nov. 2010, pp. 463–470.

[10] P. Li, L. Pileggi, M. Asheghi, and R. Chandra, "IC thermal simulation and modeling via efficient multigrid-based approaches," *IEEE Trans. Comput.-Aided Des.*, vol. 25, no. 9, pp. 1763–1776, Sep. 2006.

[11] P. Li, L. Pileggi, M. Asheghi, and R. Chandra, "Efficient full-chip thermal modeling and analysis," in *Proc. IEEE/ACM ICCAD*, Nov. 2004, pp. 319–326.

[12] D. Steinkraus, I. Buck, and P. Y. Simard, "Using GPUs for machine learning algorithms," in *Proc. ICDAR*, 2005, pp. 1115–1120.

[13] Z. Feng and P. Li, "Multigrid on GPU: Tackling power grid analysis on parallel SIMT platforms," in *Proc. ICCAD*, Nov. 2008, pp. 647–654.

[14] Z. Feng and Z. Zeng, "Parallel multigrid preconditioning on graphics processing units (GPUs) for robust power grid analysis," in *Proc. IEEE/ACM DAC*, Jun. 2010, pp. 661–666.

[15] J. Croix and S. Khatri, "Introduction to GPU programming for EDA," in *Proc. IEEE/ACM ICCAD*, Nov. 2009, pp. 276–280.
[16] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Upper Saddle River, NJ: Prentice-Hall, 1994.
[17] P. Kumar and D. Atienza, "Neural network based on-chip thermal simulator," in *Proc. ISCAS*, May–Jun. 2010, pp. 1599–1602.
[18] T. Ho, P. Lam, and C. Leung, "Parallelization of cellular neural networks on GPU," in *Proc. Conf. Patt. Recog.*, 2008, pp. 2684–2692.
[19] K.-S. Oh and K. Jung, "GPU implementation of neural networks," in *Proc. Conf. Patt. Recog.*, 2004, pp. 1311–1314.
[20] P. Ghosal, T. Samanta, H. Rahaman, and P. Dasgupta, "Thermal-aware placement of standard cells and gate arrays: Studies and observations," in *Proc. ISVLSI*, Apr. 2008, pp. 369–374.
[21] K. Chao and D. Wong, "Thermal placement for high-performance multichip modules," in *Proc. ICCD: VLSI Comput. Processors*, Oct. 1995, pp. 218–223.
[22] J. Parry, H. Rosten, and G. Kromann, "The development of component-level thermal compact models of a C4/CBGA interconnect technology: The motorola PowerPC 603 and PowerPC 604 RISC microproceesors," *IEEE Trans. Compon. Packag. Manuf. Technol.*, vol. 21, no. 1, pp. 104–112, Mar. 1998.
[23] E. Bosch, "Thermal compact models: An alternative approach," *IEEE Trans. Compon. Packag. Technol.*, vol. 26, no. 1, pp. 173–178, Mar. 2003.
[24] Y. Liu and J. Hu, "GPU-based parallelization for fast circuit optimization," in *Proc. IEEE/ACM DAC*, Jul. 2009, pp. 943–946.
[25] D. Chatterjee, A. DeOrio, and V. Bertacco, "Event-driven gate-level simulation with GPUs," in *Proc. DAC*, Jul. 2009, pp. 557–562.
[26] S. Lahabar, P. Agrawal, and P. J. Narayanan, "High performance pattern recognition on GPU," in *Proc. Nat. Conf. Comput. Vision, Patt. Recog., Image Process. Graph.*, 2008, pp. 154–159.
[27] F. Incropera, D. Dewitt, T. Bergman, and A. Lavine, *Fundamentals of Heat and Mass Transfer*. New York: Wiley, 2007.
[28] J. Demmel, S. Eisenstat, J. Gilbert, X. Li, and J. Liu, "A supernodal approach to sparse partial pivoting," *SIAM J. Matrix Anal. Applicat.*, vol. 20, no. 3, pp. 720–755, 1999.
[29] K. Stxben and U. Trottenberg, *Multigrid Methods: Fundamental Algorithms, Model Problem Analysis and Applications*. New York: Springer, 1981.
[30] M. Riedmiller, "Rprop-description and implementation details," Inst. Logic Complexity Syst. Deduct., Univ. Karlsruhe, Karlsruhe, Germany, Tech. Rep. W-76128, Jan. 1994.
[31] A. Leon, J. L. Shin, K. W. Tam, W. Bryg, F. Schumacher, P. Kongetira, D. Weisner, and A. Strong, "A power-efficient high-throughput 32-thread SPARC processor," in *Proc. ISSCC*, Feb. 2006, pp. 295–304.
[32] *NVIDIA GeForce GTX 480* [Online]. Available: http://www.nvidia.com/object/product_geforce_gtx_480_us.html
[33] *CUDA Sparse Library* [Online]. Available: http://developer.download.nvidia.com/compute/cuda
[34] N. Bell and M. Garland, "Efficient sparse matrix-vector multiplication on CUDA," NVIDIA Corporation, Santa Clara, CA, Tech. Rep. NVR-2008-004, Dec. 2008.
[35] A. Coskun, D. Atienza, T. Rosling, T. Brunschwiler, and B. Michel, "Energy-efficient variable-flow liquid cooling in 3-D stacked architectures," in *Proc. DATE Conf. Exhib.*, 2010, pp. 111–117.
[36] J. Butcher, *Numerical Methods for Ordinary Differential Equations*. New York: Wiley, 2003.

Arvind Sridhar (S'07) received the B.Eng. degree in electronics and communication engineering from the College of Engineering Guindy, Anna University, Chennai, India, in 2006, and the M.A.Sc. degree in electronics from Carleton University, Ottawa, ON, Canada, in 2009. He is currently pursuing doctoral studies with the Embedded Systems Laboratory, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.

He was a Research Scholar with the Computer-Aided Design Laboratory, Carleton University from 2006 to 2009, and has interned with the Advanced Thermal Packaging Group, IBM Research, Zurich, Switzerland, in 2011. He is the author of 3D-ICE, the first compact transient thermal simulator for 2-D/3-D ICs with liquid cooling, which is currently being used by researchers in more than 50 universities and laboratories worldwide.

Alessandro Vincenzi (S'11) received the B.S. and M.S. (summa cum laude) degrees in computer science from the University of Parma, Parma, Italy, and the University of Verona, Verona, Italy, in 2007 and 2010, respectively. In 2010, he joined the Embedded Systems Laboratory Group, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, where he is currently working toward the Ph.D. degree in electrical engineering.

His current research interests include thermal modeling of electronic devices, as well as programming on parallel and high performances architectures.

Mr. Vincenzi received the Best Student Award at the end of his first year of studies from the University of Parma in 2004.

Martino Ruggiero (M'11) received the M.S. degree in electrical engineering and the Ph.D. degree from the University of Bologna, Bologna, Italy, in 2004 and 2007, respectively.

He is currently with the Electronics, Computer Sciences, and Systems Department, University of Bologna, and with the Embedded Systems Laboratory, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, where he holds a post-doctoral position. His current research interests include embedded system architecture and software (SW) design, with particular emphasis on low-power architecture design and SW for ultraportable devices, distributed and parallel computing, development of a simulation environment at different levels of abstraction for multiprocessor systems-on-chip, application partitioning for parallel architectures, and complete algorithmic techniques for mapping and scheduling.

David Atienza (M'05) received the M.Sc. and Ph.D. degrees in computer science and engineering from the Complutense University of Madrid (UCM), Madrid, Spain, and the Inter-University Micro-Electronics Center, Heverlee, Belgium, in 2001 and 2005, respectively.

Currently, he is a Professor and the Director of the Embedded Systems Laboratory (ESL), École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, and an Adjunct Professor with the Computer Architecture and Automation Department, UCM. His current research interests include system-level design methodologies for low-power embedded systems and high performance systems-on-chip (SoCs), including new thermal-aware design for 2-D and 3-D multiprocessor SoCs, design methods, and architectures for wireless body sensor networks, dynamic memory management and memory hierarchy optimizations, as well as novel architectures for logic and network-on-chip interconnects. In these fields, he is a co-author of more than 160 publications in peer-reviewed international journals and conferences, several book chapters, and two U.S. patents.

Dr. Atienza received the Best Paper Award at the VLSI-SoC 2009 Conference, and three Best Paper Award Nominations at the WEHA-HPCS 2010, ICCAD 2006, and DAC 2004 Conferences. He is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF CIRCUITS AND SYSTEMS and Elsevier *Integration*. He has been a member of the Executive Committee of the IEEE Council on Electronic Design Automation since 2008, and a GOLD member of the Board of Governors of the IEEE Circuits and Systems Society since 2010.