Fast Thermal Simulation of 2D/3D Integrated Circuits Exploiting Neural Networks and GPUs

Alessandro Vincenzi, Arvind Sridhar, Martino Ruggiero, David Atienza
Embedded Systems Laboratory (ESL), École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
{alessandro.vincenzi, arvind.sridhar, martino.ruggiero, david.atienza}@epfl.ch

***Abstract*—Heat removal is one of the major challenges faced in developing the new generation of high density integrated circuits. Future design technologies strongly depend on the availability of efficient means for thermal modeling and analysis. These thermal models must be also accurate and provide the most efficient level of abstraction enabling fast execution.**

We propose an innovative thermal simulation method based on Neural Networks that is able to solve the scalability problem of transient heat flow simulation in large 2D/3D multi-processor ICs by exploiting the computational power of massively parallel graphics processing units (GPUs).

I. Introduction and Related Work

Continued scaling and miniaturization of CMOS technology is drastically increasing device power density. Managing temperature, especially local hot spots, has been indeed recognized as one of the major challenges for designers in the latest technology nodes [1]. As a consequence, thermal modeling and analysis is a novel and distinct area of study that has gained major attention due to the large power density increase of the latest integrated circuits (IC) [2]. In addition, this power density increase will be exacerbated by upcoming 3D IC stacking, which involves cooling a volume instead of just cooling a planar IC surface [3], [4].

New thermal modeling technologies that can capture the transient thermal behavior of hardware elements and their interaction with the software running on them need to be developed. These novel thermal modeling approaches must guarantee a sufficient degree of accuracy to capture the complex mechanisms that regulate the thermal diffusion and radiation. At the same time, they must be defined with a level of abstraction that allows for fast execution [3], [5].

In recent years, a few full-chip 2D/3D thermal models have been proposed to provide detailed temperature distributions [1], [2], [5], [6]. However, thermal analysis is a resource and time consuming task that can be strongly influenced by its accuracy requirement, as well as the complexity of the run-time 2D/3D IC execution scenario under study. Thus, finite-element models [1], [4], as well as compact models [2], can impose long simulation times that turn out to be unfeasible for large IC designs.

Nevertheless, in recent times graphics processors have become increasingly competitive in speed, programmability and price. Indeed the main advantage of GPUs over 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 EDA tools [7] [12].

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

In this context, Neural Networks (NNs) [8] represent a promising parallelizable solution to this problem [9]. While one of their main drawbacks is their potentially long execution time because they are a highly computationally and data intensive solution [10], their large number of operations and relatively low data transfer makes them also a potentially attractive concept for GPU computing [11].

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 2D/3D full-chips. The main contributions of this paper are the following:

1) We present a new full-chip thermal modeling methodology based on NNs and GPUs. The model is transient and can accurately predict the temporal evolution of both planar and 3D chip temperatures. We have validated the accuracy of the model comparing our results with 3D-ICE [5], a state-of-the-art thermal simulation tool for 3D ICs.

2) We propose a methodology for the training of the proposed NNs for thermal modeling. The one-time training enables removing 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 drastically improves computational efficiency.

3) We introduce an innovative approximation, namely, the proximity-based reduction, that further reduces the computational complexity of our thermal model, while negligibly affecting model accuracy.

4) Our NN-based thermal model specifically targets GPU platforms. Utilizing a GPU in our proposed NN-based thermal modeling approach resulted in considerable time-savings, es-

This research has been partly supported by the Swiss NSF (SNF), grant number 200021-130048, and the Nano-Tera RTD project CMOSAIC (ref. 126618), which is financed by the Swiss Confederation and scientifically evaluated by SNSF.

978-1-61284-660-6/11/$26.00 © 2011 IEEE

pecially for large IC problem sizes and detailed layout models. 5) Once trained, our NN-based thermal simulator can be reused any number of times with different 2D/3D IC floorplan configurations. Hence, our proposed thermal modeling approach enables fast and accurate thermal-aware design space and floorplanning explorations using GPUs.

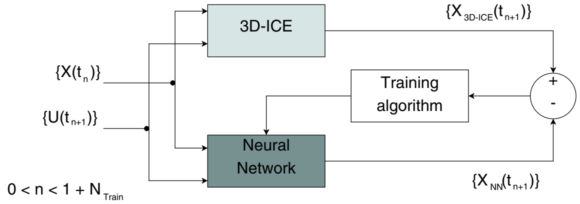

Fig. 1. NN-based thermal simulator trained using 3D-ICE.

II. Neural Network-based Thermal Simulation

Thermal simulation of ICs is conventionally performed using compact modeling [2], [13]. The conventional compact modeling for heat conduction in solids is done by applying finite-difference approximation to the governing equations of heat transfer in solids [13]. This involves dividing the different layers in an IC into cuboidal ”Thermal Cells”. Next, the well-known analogy between heat and electrical conduction is invoked with the temperature represented as voltage and heat flow represented as electric current [2] to convert each thermal cell into an equivalent electrical circuit [5]. 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. Hence, the following system of ordinary differential equations are 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.

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 accuracy [5].
The next step is the formulation of equations for the simulation of the Thermal Grid. For this, Eq (1) is integrated numerically using the backward Euler method as follows:

$$\begin{aligned}\left(\mathbf{G} + \frac{1}{h}\mathbf{C}\right)\mathbf{X}(t_{n+1}) &= \frac{1}{h}\mathbf{C}\mathbf{X}(t_n) + \mathbf{U}(t_{n+1}) \\ \Rightarrow \mathbf{A}\mathbf{X}(t_{n+1}) &= \mathbf{B}\mathbf{X}(t_n) + \mathbf{I}\mathbf{U}(t_{n+1}),\end{aligned} \tag{2}$$

where h is the time-step used for the numerical integration and t_n denotes the n^{th} time point during the transient simulation. The system of linear equations in (2) can be solved using direct or iterative solvers. We propose a thermal simulator based on Artificial Neural Networks (NNs) as an alternative to these approaches.

Artificial Neural Networks are multi-input multi-output operators, which can be trained to mimic the behavior of any mathematical function through learning the input-output dependence of that function from some test data. The various inputs of this neural network are connected to the outputs via weighted links. Hence, the outputs can be expressed as a weighted sum of the inputs as follows:

$$y_i = \sum_j w_{ij} x_j, \tag{3}$$

where w_{ij} connects input x_j to output y_i. In the case of thermal simulation, assuming that the thermal properties of the materials in an IC do not vary with temperature, (1) represents a linear time-invariant (LTI) system. Thanks to this property, a linear single layer neural network should be sufficient for our purposes. In addition, 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.

The goal of our neural network-based simulator is to approximate the following function derived from (2):

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

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 first trained using samples generated with 3D-ICE and then run as a stand-alone simulator on GPUs to give significant speedups when compared to the conventional techniques.

A. Training of the Neural Network

Learning in NN essentially consists of finding the correct set of weights for each connection in the NN. Training is performed by supplying to the NN a set of inputs and outputs taken from 3D-ICE over a specific simulation interval (N_{Train} time points or train samples) as shown in Fig. 1. During each training iteration, the outputs from the NN and the target output form 3D-ICE are compared and the weights of the NN are updated according to the error incurred. These iterations are repeated 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 many layers of different materials but 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 the bulk of the power is dissipated. Hence, it would be sufficient to train and run the neural network to simulate temperatures for only the active layers. Consequently, the number of neurons (outputs) in the reduced neural network would equal the number of thermal cells in the active layers of the IC.

B. Training length and method

The training process must ensure that the neural network 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 ($N_{Train,min}$) depends upon the nature and the size of the thermal grid being solved. The method to compute this value is the following.

In a neural network 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 difference between the output y_i and the corresponding target output t_i. 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. And 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, $N_{Train,min} = m$. In our thermal simulations, given n_{cells} number of thermal cells in an active layer of an IC, there are $2n_{cells}$ inputs to the neural network (past temperatures plus the power inputs for each cell), related to outputs follows:

$$\mathbf{X}(t_{n+1}) = \mathbf{W} \cdot \left[\begin{array}{c} \mathbf{X}(t_n) \\ \mathbf{U}(t_{n+1}) \end{array} \right], \qquad (5)$$

where $\mathbf{W}$ is a $n_{cells} \times \ 2n_{cells}$ matrix containing the weights of the NN. Hence, $N_{Train,min} = 2n_{cells}$. However, it is recommended that a higher number of data points be used for the training to hasten the convergence of the training algorithm and to a much more accurate solution. 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 [14], which is one of the fastest training algorithms available.

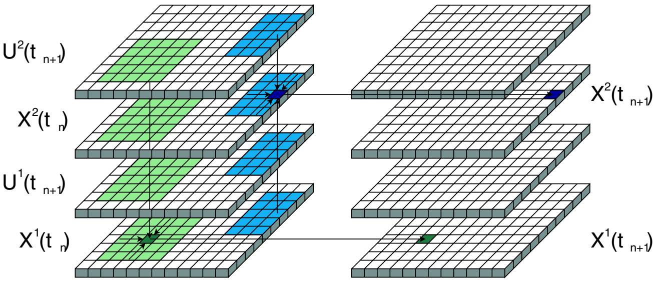

Fig. 2. "Neighborhoods" and input/output dependency of two neurons between two consecutive train samples.

C. Proximity based model reduction

A significant reduction in computational and memory complexity can be achieved by relaxing the requirement of a fully connected neural network for thermal simulation. Given the diffusive nature of the heat flow in an IC, much of the heat flows vertically upwards from the source to the ambient, following the path of least resistance. Hence, 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 here defined as *proximity*.
In our experiments, rectangular regions of "neighborhood" were defined around each neuron based on r, and only cells lying within this region were connected to the neuron in the network. Every neuron must also be connected vertically to all the remaining active layers. Two such neurons and their corresponding neighborhoods are highlighted in Fig. 2. This image refers to a 3D IC with two dies and shows how neurons are connected to their neighbours (input) and how their outputs match the thermal state of the same thermal cell in the following time point, as expressed in 5. Hence, the $\mathbf{W}$ matrix in (5) would no longer be a full matrix but very sparse, leading to considerably lower memory consumption, and faster training and simulation using the NNs.

D. Randomization of training input

To enable fast design-space exploration of ICs for thermal reliability, our proposed model must be able to simulate different floorplan configurations without loss of accuracy. If the neural network is trained with a given configuration of floorplan elements then all the thermal cells that lie within one of them would always be fed with identical power values at every time step. After the training, if the neural network is used to simulate a different distribution of floorplan elements, the thermal cells might receive power dissipation values that were not recorded in the training set, leading to large errors (up to $2.5°$C) if comapred with the requested precision.

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 the floorplan must be given different and random input power values in order to train the neural network for the worst case situation where every thermal cell belongs to a different floorplan element. Once the NN is trained in this manner, virtually any kind of floorplan configuration can be simulated with the error being bounded purely by the training process. Also, it must be ensured that the entire range of input power magnitudes has been covered by the training process. For this, in our experiments, each thermal cell receives a range of random values between zero and a value higher than the maximum heat dissipation per unit area for any floorplan element. This value can be determined repeating several trainings until the average magnitude of all the errors measured post training reaches the target precision of the training.

E. Implementation of the NN-based simulator on GPUs

The presented NN for thermal modeling of complex IC has been implemented for training and running on GPUs. In both the implementations, single precision floating point values can be used since our NNs are trained to compute the variation of the temperatures in the active layers and ony a few bits are needed to represent the integer parts of these numbers.

1) Training phase on GPU: The training is based on an iterative process. At each iteration, called *epoch*, the weights are used to process all the N_{Train} train samples and then updated, epoch by epoch, to reduce the error measured against the desired output that the neural network is supposed to learn. The training stops when the errors produced by all the neurons on all the train samples are found to be lower than a desired degree of precision, or when the maximum error error does not change more than a given threshold after two consecutive epochs.

As mentioned before (see Section II-B), every neuron (a row in the $\mathbf{W}$ matrix) can be trained independently from the others since it is connected with the input layer by its own set of weights. As a consequence, each neuron can process its own N_{Train} number of samples. Training each neuron independently speeds up the training and give a homogeneous level of precision to all neurons since otherwise the duration of the entire training and the resulting precision will be dominated by the neuron with the slowest convergence rate. Moreover, this gives the possibility to exploit the thread hierarchy and organization proposed by CUDA by simply creating a block of threads for each neuron. Then, within a block, every thread can be assigned to one or more connections with the inputs (the columns in the $\mathbf{W}$ matrix).

The whole training set is stored into the global memory and organized as a 3D object. Every training sample can be seen as a set of couples of matrices storing temperatures and power inputs for each thermal cell in each die (see Fig. 2). Therefore, the whole training set corresponds to a `cudaExtent` structure where the depth is the number of train samples and the width and the height match the effective number of columns (times the double of the number of dies) and rows in which the simulated IC is divided according to the dimensions of the thermal cells. The same memory organization can be aplpied to the data stored in shared memory. To execute coalesced accesses to global memory, temperatures and power values from all the dies can be interleaved to build a unique matrix: Fig. 3 shows how to store and align a training sample for a single die problem, divided into 8 rows and 5 columns. Thanks to this memory organisation, all accesses to the inputs placed in the same line of the training sample are done at consecutive locations of memory.

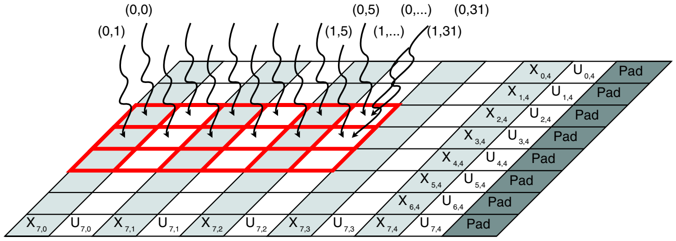

Fig. 3. Memory alignment and accesses done by the threads.

To increase the occupancy (the number of thread blocks running on the same multiprocessor) introducing more threads per block, the algorithm can read more than one line of memory per time. Threads can indeed be organized into two dimensional blocks as shown in Fig. 3, rounding the x dimension to a multiple of the warp size. This will give two significative improvements: the long latency to access global memory is hidden by multiple accesses done by more warps running on the same multiprocessor and different warps will access a different line of global memory avoiding non coalesced accesses. In our experiments we found that the most suitable number of lines (the y dimensions of the block) corresponds to using a number of threads such that the occupancy of each multiprocessor given by the CUDA occupancy calculator is equal to 50%.

The threads in a block must also cooperate to compute the output y_i of the neuron as in 3 to compare it with the corresponding target t_i and get a measure of the error to update the weights. This parallel reduction can be implemented as suggested in [18].

The only part that slows the performances of the GPU implementation of RPROP is the update of the weights done at the end of every epoch. This operation depends on the runtime values of the weights and this causes divergences in the execution that cannot be avoided.

2) Runtime execution of NN on GPU: Once the training is complete, the entire NN is described by means of the weight matrix $\mathbf{W}$, stored on the GPU global memory according to the Compressed Sparse Row format (CSR) as required by the cuSparse library [16]. Every time step of the thermal simulation corresponds to a matrix-vector multiplication between $\mathbf{W}$ and a vector containing temperatures and power inputs for all the cells in the active layers. The matrix-vector multiplication $\mathtt{y} = \alpha\mathtt{A} * \mathtt{x} + \beta\mathtt{y}$ is performed using the function `cusparseScsrmv` of cuSparse library [17].

Every input vector $\mathtt{x}$ 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 while the output vector $\mathtt{y}$ contains the resulting temperatures $\mathbf{X}(t_{n+1})$. This last set of values represents the thermal state to be re-used as input 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. This improvement can be applied only if the weight matrix is permuted after the training to separate, die by die, the weights connecting neurons to temperatures from weights connecting power inputs. Each time step is finally computed with three single statements: the download of each $\mathbf{U}(t_n)$ from CPU to GPU, the matrix-vector multiplication and the swap of the pointers.

III. Experimental Results

All the ICs tested contain two layers for each die. The active layer has the UltraSPARC Niagara floorplan [15] on top

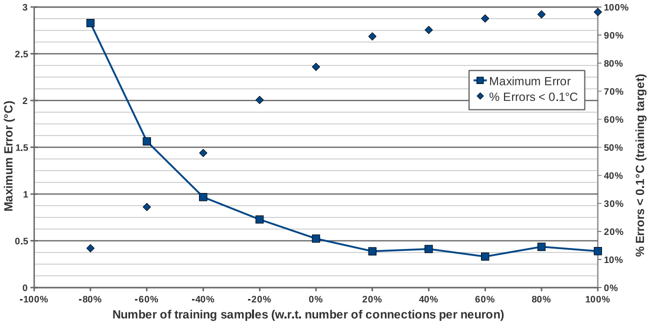

Fig. 4. Run-time error of the NN-based simulator as a function of the number of training time points used in the training.

of it while the other one is an interconnect layer conducting heat to the ambient. All the trainings are done using $0.1°\mathrm{C}$ as target error and $90\mathrm{W/cm}^2$ as the maximum value of power density for the range of power inputs. To measure the precision after the training, the NNs are used to simulate two hours of execution and the maximum run-time temperature error with respect to the output from 3D-ICE is measured and recorded before every power switching activity (five times per second). During this phase, random power inputs are generated using the peak power value declared by the designers for any floorplan element as upper bound.

Each experiment that follow will serve to demonstrate the basis of the various aspects of our implementation and to highlight the resulting advantages in computational complexity and accuracy of the proposed 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 II-B, temperatures and input data for a minimum number of training time points ($N_{Train,min}$) 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. 2). That is, $N_{Train,min} = \max_i m_i$. To illustrate this, an IC with two dies was discretized into thermal cells of dimensions $500\mu\mathrm{m} \times 500\mu\mathrm{m}$ and simulated using 3D-ICE to prepare the training set. Next, a neural network was created with a neighborhood distance $r = 5000\mu m$ and series of trainings were run using different number of train points. The maximum error measured against 3D-ICE and the percentage of errors lower than the target precision requested in the training phase are plotted in Fig.4. The value 0% in the x axis indicates that $N_{Train} = N_{Train,min}$ while the value $+40\%$ indicates $N_{Train} = 1.4N_{Train,min}$. As can be seen, the error drops quickly as N_{Train} approaches $N_{Train,min}$ and then remains fairly constant, indicating the significance of this training criteria. On the other side, the percentage of neurons that produce an error in the output less than $0.1°\mathrm{C}$ keeps increasing and settles at 98%.

B. Effect of Proximity based model reduction

As described in Section II-C, exploiting the physics of heat transfer in an IC to reduce the connectivity in the proposed NN-based simulator considerably 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, we studied the error produced by the neural network when simulating IC made with one, two and three dies discretized into thermal cells of dimensions $1000\mu\mathrm{m} \times 1000\mu\mathrm{m}$. In each case, the number of train samples used to train was $1.2N_{Train,min}$ while the neighborhood distance r was increased to get a surface from $(1000\mu\mathrm{m})^2$ up to an area covering the full chip (i.e., a fully connected neural network, where each output depends upon all the inputs in the IC). The run time errors are shown in the plot in Fig. 5. The case at three dies with $r = 1000\mu m$ is not shown since the NN cannot complete the training converging to the desired target precision. The results show that a full connected network can generate a maximum error equal to $0.25°\mathrm{C}$ while a loosely connected network, such as $r = 5000\mu m$ produces a maximum error equal to $0.35°\mathrm{C}$. On the other side, reducing the proximity decreases the percentuage of errors lower than the training target but this side effect can be balanced increasing the length of the training set used to train the NN.

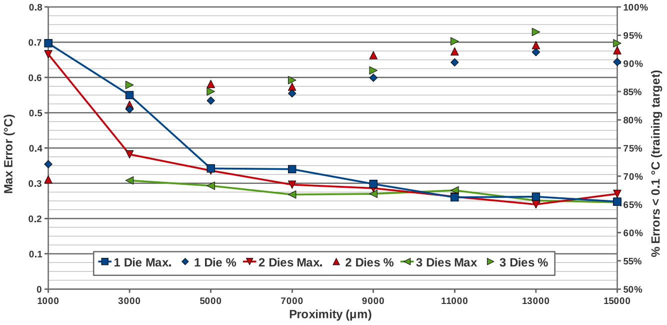

Fig. 5. Run-time error of the NN-based simulator as a function of the proximity r.

C. Speed ups 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 (480 CUDA cores running at 1.4 GHz and 1.5 GB GDDR5). Then 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 (4 cores and 6GB of RAM). The ratios between the simulation times of 3D-ICE and the neural network run on GPU are compared in term of speedup and plotted in Fig. 6 as a function of the proximity r. In our experiments, the NN-based simulations were found to be up to 100x faster than 3D-ICE during run time (considering 3 dies and $500\mu m \times 500\mu m$ discretization with neighborhood

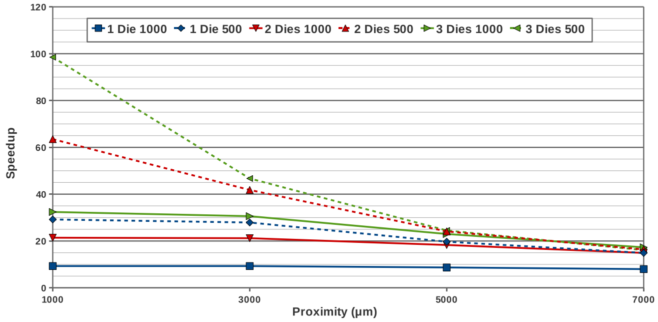

Fig. 6. Simulation time comparison between 3D-ICE run on CPU and NN run on GPU.

distances $r = 1000\mu m$). The plot shows that the speed-ups depend upon the complexity of the problem, i.e. the number of dies, the discretization and the neighborhood distance between cells. This speed-up 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 were run on both the CPU platform (4 threads running on the Intel(R) Core(TM) i7 920 2.67 GHz processor) and the GPU. The ratio between the times taken to train the NN are plotted in Fig. 7 while Fig. 8 shows the times of the thermal simulations. Both results are plotted as a function of the neighborhood distance r, the discretization and the number of dies. Again, the speed-ups depend upon the complexity of the problem: for simple cases GPU and CPU performances are almost the same while for complex ones the GPU is faster (up to 6x for the training and 9x when simulating). The reported speed-ups are measured with respect to a multi-threaded CPU version. If we consider the single-thread CPU serial execution of the NN, these speedups become much higher (i.e. 24x for the training and 36x for the simulation). Table I reports the time taken by 3D-ICE and by our neural networks to simulate two hours of activity.

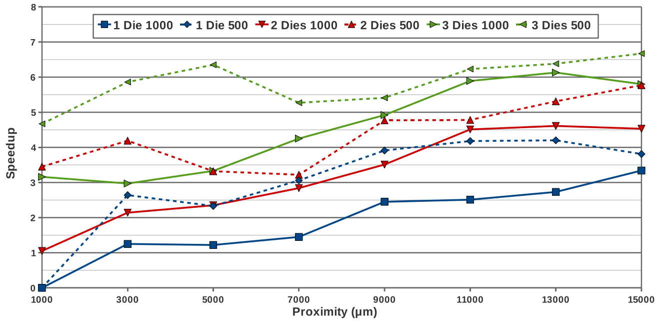

Fig. 7. Simulation time comparison between training NN on CPU and GPU.

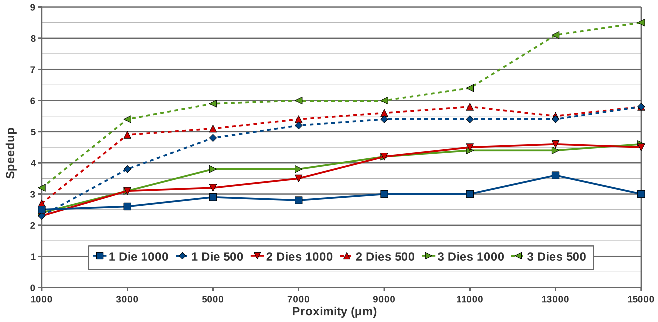

Fig. 8. Simulation time comparison between NN run on CPU and GPU.

TABLE I

SIMULATION TIMES (SECONDS) USING PROXIMITY $r = 5000\mu m$.

Dies	$1000\mu m \times 1000\mu m$			$500\mu m \times 500\mu m$		
	3D-ICE	NN-CPU	NN-GPU	3D-ICE	NN-CPU	NN-GPU
1	11.7	4.2	1.4	56.6	13.6	2.9
2	32.1	6.4	1.8	174.9	37.3	7.2
3	57.9	9.1	2.5	347.4	83.7	14.1

IV. CONCLUSION

This paper presented the design and validation of an innovative full-chip thermal modeling approach exploiting neural networks and the computational power of modern GPUs. Our experiments with realistic multi-core IC designs show that the proposed approach achieves relevant run-time speed-ups, while keeping a negligible error.

REFERENCES

[1] P. Li *et al.*, “IC thermal simulation and modeling via efficient multigrid-based approaches”, *IEEE Trans. on Computer-Aided Design*, 25(9), pp.1763-1776, 2006.

[2] W. Huang *et al.*, “HotSpot: A compact thermal modeling methodology for early-stage VLSI design”, *IEEE Trans. VLSI Sys.*, 2006.

[3] Z. Feng, P. Li; , “Fast thermal analysis on GPU for 3D-ICs with integrated microchannel cooling,” *Proc. IEEE/ACM ICCAD, 2010.*

[4] C. Xu *et al.*. “Fast 3D Thermal Analysis of Complex Interconnect Structures Using Electrical Modeling and Simulation Methodologies”, *Proc. IEEE/ACM ICCAD, 2009.*

[5] A. Sridhar *et al.*, “3D-ICE: Fast compact transient thermal modeling for 3D ICs with inter-tier liquid cooling”, *Proc. IEEE/ACM ICCAD, 2010.*

[6] P. Li *et al.*, “Efficient Full-Chip Thermal Modeling and Analysis”, *Proc. IEEE/ACM Conference on Computer-Aided Design*, pp. 319-326, November 2004.

[7] J. Croix and S. Khatri ”Introduction to GPU Programming for EDA”, ICCAD 09

[8] S. Haykin, *N Networks: A Comprehensive Foundation* (1st ed.), 1994 Prentice Hall PTR, Upper Saddle River, NJ, USA.

[9] P. Kumar, D. Atienza, “Neural network based on-chip thermal simulator”, *Proc. IEEE Sym. Circuits and Systems (ISCAS)*, 2010.

[10] T.Y. Ho, P.M. Lam, and C.S. Leung, “Parallelization of cellular neural networks on GPU”, *Conference on Pattern Recognition*, 2008.

[11] K.-S. Oh and K. Jung, “GPU implementation of neural networks”, *Conference on Pattern Recognition*, vol. 37, no. 6, pp. 1311-1314, 2004.

[12] S. Raghav *et al.*, “Scalable instruction set simulator for thousand-core architectures running on GPGPUs” *Proc. High Performace Computing and Simulation Conference 2010*, pp. 459-466.

[13] J. Lienhard-IV and J. Lienhard-V, *A heat transfer textbook*. Cambridge, Massachusetts: Phlogiston Press, 2006.

[14] M. Riedmiller, “Rprop- description and implementation details”, *University of Karlsruhe Technical Report*, January 1994.

[15] A. Leon *et al.*, “A power-efficient high-throughput 32-thread SPARC processor”, *Proc. International Solid-State and Circuits Conference (ISSCC) 2007.*

[16] CUDA Sparse Library, http://developer.download.nvidia.com/compute /cuda/.

[17] N. Bell and M. Garland, ”Efficient sparse matrix-vector multiplication on CUDA”, *NVIDIA Corporation Technical Report*, Dec, 2008.

[18] Mark Harris, ”Optimizing Parallel Reduction in CUDA”, URL: http://developer.download.nvidia.com/compute/cuda/sdk/website/C/src /reduction/doc/reduction.pdf