Accelerating Thermal Simulations of 3D ICs with Liquid Cooling using Neural Networks

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

Vertical integration is a promising solution to further increase the performance of future ICs, but such 3D ICs present complex thermal issues that cannot be solved by conventional cooling techniques. Interlayer liquid cooling has been proposed to extract the heat accumulated within the chip. However, the development of liquid-cooled 3D ICs strongly relies on the availability of accurate and fast thermal models.

In this work, we present a novel thermal model for 3D ICs with interlayer liquid cooling that exploits the neural network theory. Neural Networks can be trained to mimic with high accuracy the thermal behavior of 3D ICs and their implementation can efficiently exploit the massive computational power of modern parallel architectures such as graphic processing units. We have designed an ad-hoc Neural Network model based on pertinent physical considerations of how heat propagates in 3D IC architectures, as well as exploring the most optimal configuration of the model to improve the simulation speed without undermining accuracy. We have assessed the accuracy and run-time speed-ups of the proposed model against a 3D IC simulator based on compact model. We show that the proposed thermal simulator achieves speed-ups up to 106x for 3D ICs with liquid cooling while preserving the maximum absolute error lower than 1.0 °C.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids—*Simulation*

Keywords

Thermal Modeling, Thermal Simulation, 3D ICs, Liquid Cooling, Neural Networks

1. INTRODUCTION

With conventional scaling of CMOS devices fast running into theoretical walls, vertical integration of IC dies seems to be the most viable solution to meet the ever rising demands for more compact and faster electronic products in the short- and medium-term [1]. However, 3D integration brings with it aggravated thermal issues due to compounded heat fluxes and larger thermal resistances. In this context, interlayer microchannel-based liquid cooling of 3D ICs has come to be accepted as the most promising solution enabling vertical integration of ICs [2].

Nevertheless, interlayer liquid cooling still remains a niche technology and has not yet been deployed for large-scale production by the electronics industry. One of the main reasons which limits interlayer liquid cooling technology is the lack of Electronic Design Automation (EDA) tools that can provide IC designers with efficient simulation of thermal behavior of ICs cooled using microchannel heat sinks. In particular, this situation becomes a major bottleneck for the development of thermal-aware design and run-time approaches for liquid cooled 3D ICs.

In the last few years, several authors have attempted to address this issue by using various modeling methodologies [3, 4, 5]. Unfortunately, most of these modeling and simulation solutions are based on either fine-grained finite-element/finite-difference methods [3] or compact modeling techniques [4, 5]. Both methods have a superlinear computational complexity with respect to size of the problem. As more and more number of IC-dies are stacked with interlayer microchannel cooling, the computational times of these solutions would become too long to be practical for effective design-space explorations. In this paper, we address the problem of computational complexity of existing thermal simulation methodologies by proposing two key innovations.

The first contribution is the use of modern Graphic Processor Units (GPUs). GPUs have become in the recent times a computing mainstay in many different applications other than graphics processing. Their massively parallel architecture makes them a much faster and cheaper alternative compared to CPUs. This has resulted in the recent years in their exploitation for a myriad of EDA tools requiring high-performance computing [6, 7]. However, it is difficult to structure algorithms in order to fully exploit the GPU architecture and any new GPU-based EDA tool must be designed keeping in mind the aspects of GPU-based programming [8].

The second contribution of this work is the exploitation of Neural Networks (NNs) to provide a well-parallelizable thermal modeling approach. Neural Network theory provides a structurally straightforward programming tool that can be trained to mimic any mathematical function. In this work, we create and train a NN using 3D-ICE [4, 5], a conventional compact model-based thermal simulator for liquid-cooled 3D ICs. Once trained, the properties of large-scale parallel operations and low data transfer overhead of this NN-based simulator make it an appropriate candidate for implementation on GPUs. Recently, a NN-based simulator for 3D ICs has been presented by the authors of [9]. However, they tackle only the thermal modeling issue of conventional air-cooled ICs, without considering liquid cooling. In this paper, we create a NN-based thermal simulator for liquid-cooled 3D ICs that efficiently runs on GPUs. In a nutshell, the main contributions of this paper are the following:

1. We present a new transient thermal simulation framework for liquid-cooled 3D ICs based on neural networks and GPUs.

2. We exploit the physical insights inspired by how heat flows propagate in microchannels heat sinks to reduce the complexity of our NN-based simulator, by introducing an innovative *proximity-based reduction*. Thus, the temperature calculation speed is significantly improved while the thermal estimation accuracy is preserved.

3. We further improve the reduction techniques to find the optimal configuration of the NN-based simulator to maximize the speed-ups. A detailed analysis on the search for the NN-model configuration is also presented.

4. The NN-model, once trained, can be reused with different floorplan configurations of the 3D ICs. This feature is crucial for the acceleration of design-space exploration of liquid-cooled 3D ICs, where a very large number of different floorplans must be comparatively evaluated for thermal and electrical characterization.

The rest of the paper is organized as follows. Section 2 briefly describes the compact modeling tool that is used to train the NN-based simulator while Section 3 reviews the basics about the thermal model of conventional air-cooled ICs based on neural networks. The description of proposed NN-based thermal simulator is introduced in Section 4 and its structure and implementation details are discussed in various subsections. Experimental results are presented in Section 5. Finally, Section 6 summarizes the work with the conclusions.

2. THERMAL MODELING FOR LIQUID-COOLED 3D ICS

Interlayer liquid cooling of 3D ICs is accomplished using microchannels that are etched behind each die that is stacked in the IC. The cross-section of a typical interlayer liquid-cooled 3D IC is shown in Figure 1. Cold fluid is injected via a reservoir from one end (the inlet) and warm fluid exits into another reservoir at the other end (the outlet).

For such a structure, a compact and transient thermal simulation methodology of liquid-cooled 3D ICs typically involves the analogy between heat transfer in materials and electric current. The equivalent RC circuit that models the liquid-cooled 3D IC is represented by the following ordinary differential equations:

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

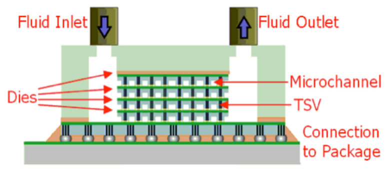

Figure 1: Cross-sectional view of a liquid-cooled 3D IC (taken from [4]).

where $\mathbf{G}$ and $\mathbf{C}$ are, respectively, the conductance and capacitance matrices. $\mathbf{X}(t)$ is the vector of temperature responses in the 3D IC, and $\mathbf{U}(t)$ is the vector of power traces. Once the system has been formulated in this manner, it is solved via numerical integration using backward Euler method. The solution at the $(n+1)^{th}$ time point is written as follows:

$$\mathbf{X}(t_{n+1}) = \mathbf{P}\mathbf{X}(t_n) + \mathbf{Q}\mathbf{U}(t_{n+1}), \tag{2}$$

where,

$$\mathbf{P} = \left(\mathbf{G} + \tfrac{\mathbf{C}}{h}\right)^{-1} \cdot \tfrac{\mathbf{C}}{h}, \quad \text{and} \quad \mathbf{Q} = \left(\mathbf{G} + \tfrac{\mathbf{C}}{h}\right)^{-1}.$$

Here, h is the step size using to discretize time. In order to reduce the problem size, a modified version of the above thermal model based on porous mediums [5] was used in this work. As in [5], Eq. (2) encloses the boundary conditions of the modeled ICs with microchannels. In our work, we consider Eq. (2) as the basis for the development of the proposed NN-based thermal simulator further described in Section 4.

3. REVIEW OF NN-BASED SIMULATOR FOR CONVENTIONAL ICS

The NN-based thermal simulator proposed in this paper is based on the ability of neural networks to learn the thermal behavior of a liquid-cooled 3D IC and then, based on this training, to act as a stand-alone thermal simulator. A NN-based thermal simulator for conventional air-cooled 2D/3D ICs was proposed in [9]. The authors proposed to train a neural network to reproduce the dependence in Eq. (2): each neuron in the model represents the thermal state of a given point (node or thermal cell) in the volume of the IC. In particular, it is sufficient to define a neuron for each node in the layers of the IC where the heat dissipation occurs (active layers). Therefore, the resulting NN will have a single layer of neurons sharing the same inputs.

Each neuron in the NN expresses the output as a weighted sum of the inputs:

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

where w_{is} are coefficients, called *weights*, that represent the contribution of the input to the output. To replicate the thermal evolution in Eq. (2), a neural network performing the following operation is constructed:

$$\mathbf{X_i}(t_{n+1}) = \sum_j w_{ij}\mathbf{X_j}(t_n) + \sum_k w_{ik}\mathbf{U_k}(t_{n+1}) \tag{4}$$

The weights in the above equation are trained to represent the coefficients of the corresponding matrices $\mathbf{P}$ and $\mathbf{Q}$ in Eq. (2). For this training, the neural network must be given training samples that describe, for each neuron, the dependencies between the inputs and the corresponding

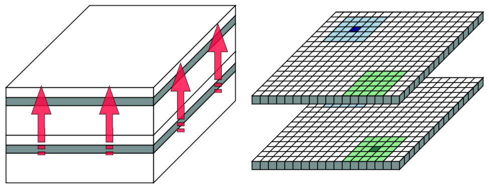

Figure 2: Heat flow and neuron connections for conventional 2D/3D ICs

output. Since the output of each neuron represents a continuous range of temperatures, all the neurons have a linear activation function. Once the neural network is trained to replicate correctly all the training samples, it can be used as stand alone thermal simulator. The accuracy of the NN-based simulator depends upon the training algorithm, the quality of inputs, the approximations involved in the construction of the NN and the error tolerance imposed on the training process.

Furthermore, the NN-based simulator proposed in [9] exploits the predominantly vertical heat flow that characterizes a conventional air-cooled IC as shown in Figure 2. This implies that there is little lateral spreading of heat, meaning temperatures of nodes in a given layer of IC that are far apart do not affect each other significantly. This model then introduces a parameter, called *proximity* that can be used to reduce the number of inputs for each neuron and hence the overall complexity of its implementation. The key idea in this process is to remove from Eq. (4) all the terms in both the summations on the RHS that do not contribute significantly to the output.

Another innovation in this simulator was that instead of using a training set made with samples representing a real scenario where a floorplan is used, power traces with random values between zero and the maximum power density of the design is fed to the neural network during training. These samples represent the worst-case scenario, and each neuron is trained for all possible heat flux distributions and heat flux levels. Hence, the run-time error of a neural network so trained will be independent of the configuration of the floorplan. This enables the NN-based simulator to be used for design space exploration (floorplanning).

4. PROPOSED NN-BASED SIMULATOR FOR LIQUID-COOLED 3D ICS

The NN-based simulator for liquid-cooled 3D ICs proposed in this work is trained in a similar manner to the model described in the preceding section. Thus, neurons represent the temperatures of nodes in the active layers. The neural network is constructed as mentioned before and trained to mimic the behavior of Eq. (2). However, the nature of heat flow in liquid-cooled 3D ICs is fundamentally different from conventional ICs and the model must be adapted to capture these different heat dissipation paths. The aspects of the proposed NN-based simulator are discussed in the ensuing subsections.

4.1 Training methodology

There are several algorithms available in the literature for training neural networks. Since the objective here is to train the NN weights in Eq. (4) to mimic the behavior of the deterministic system in Eq. (2) based on a set of inputs and outputs, a batch training algorithm like RPROP, that was used in [9], must be used. Also, given that there are as many unknown weights to be computed per neuron as there are inputs for it, a minimum number of training samples must be provided to the training algorithm for accurate estimation of the weights. As in [9], a randomized heat flux distribution with random power traces bounded by the maximum heat flux levels expected during run-time was used to generate the training data. The use of a random training set is justified by the need of a model that is independent by the distribution of power density in the various layers of a 3D IC.

One major difference of the proposed NN-based simulator from the NN-based simulator for conventional ICs is that the flow rates of coolant affects the behavior of the system. The higher the flow rate, the lower the overall thermal resistance of the system to the heat sink, which in turn means that the coefficients of the matrices in Eq. (2) are changed. Since it is common to design a liquid-cooled 3D IC for variable flow rates, training must be performed for each flow rate value intended in the final design.

To eliminate errors introduced by the training of the network and to be able to isolate the effect of the various reduction techniques applied to the proposed model, we replace the iterative training algorithms with a direct method to compute the weights as the solution of a linear system. Each training sample provides one additional equation in the computation of the unknown weights. Since we provide a higher number of samples than unknowns, we have an overdetermined system. To solve this, we deploy the least-squares method using QR decomposition in this work.

4.2 Proximity-based reduction

The heat flow patterns in liquid-cooled 3D ICs is fundamentally different from conventional air-cooled ICs. Hence, when proximity-based reduction is applied to the proposed simulator, these patterns must be taken into account. In liquid-cooled 3D ICs, there are indeed *two* major paths of heat flow as shown in Figure 3: one vertical, from the active layers towards the microchannel heat sinks, and one along the microchannels, the path along with heat is carried by the coolant. As a consequence, the temperature of a given point on the surface of the IC will be influenced by the thermal state and the heat dissipation at nodes lying upstream or downstream along the direction of the channel, in addition to those nodes which are directly above and below it in the other active layers. Hence, *proximity regions* are defined as rectangular regions in each active layer along the channel as highlighted in Figure 3.

The width of this rectangle (the "proximity distance") defines the complexity of our model. In other words, it signifies how large an area of the IC is assumed to influence the temperatures in the node under consideration. If this proximity distance is small, then the complexity of the re-

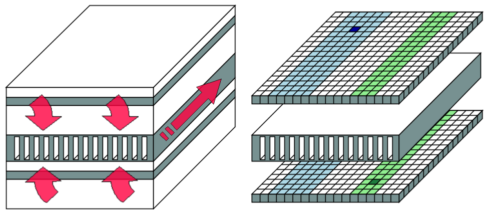

Figure 3: Heat flow and neuron connections for 2D/3D ICs with liquid cooling

Algorithm 1 Solving with constant proximity

```
1: for $c_i = 1 ... N_C$ do
2:    $I \leftarrow$ ExtractInput ($TrainSet, Proximity, [1, 1, c_i]$)
3:    $(Q, R) \leftarrow$ QRdecompose ($I$)
4:    for $r_j = 1 ... N_R$ do
5:       for $l_k = 1 ... N_L$ do
6:          $O \leftarrow$ ExtractOutput ($TrainSet, [l_k, r_j, c_i]$)
7:          $(W[l_k, r_j, c_i], Residual) \leftarrow$ lssolve ($O, Q, R$)
8:       end for
9:    end for
10: end for
```

sulting neural network is lower, at the expense of potential loss of accuracy. Two cases are shown in Figure 3: for one neuron in the bottom-right corner of the IC, the proximity region is much smaller than the proximity of the neuron in the top-left corner.

The least squares method used for the calculation of the weights, as described in Section 4.1, gives as one of the outputs the final residual during the calculation of weights. This residual value serves as an indicator of the run-time accuracy of the NN-based simulator that has been constructed. Hence, depending upon the accuracy and performance requirements of the designer, one can fix a desired proximity distance during the training of the NN. This is illustrated in Algorithm 1, where neurons are indexed using a tuple $[layer, row, column]$ as they belong to a three dimensional structure having dimensions $[N_L, N_R, N_C]$ (see Figure 3).

Algorithm 1 extracts from each sample in the training set a matrix containing the training input for the given neuron (line 2) and executes its QR decomposition (line 3). This operation is done only for neurons that belong to the first row and to the first layer because neurons in the same row on all the layers share the same proximity region. Then, for all the remaining neurons, it extracts the target training output (line 6) and computes their weights as the unknowns of a linear system (line 7). The residual is discarded or reported as a measure of the accuracy of the NN over the training set.

4.3 Optimal proximity profile

Due to the sensible heat absorption in microchannel heat sinks, temperatures increase as the distance from the inlet increases. Hence, there is a strong thermal gradient form the inlet to the outlet. In addition to causing nodes (neurons) near the outlet to be considerably hotter than those near the inlet, this phenomenon causes unequal lateral spreading of heat in the ICs from inlet to outlet.

This means that neurons representing temperatures at nodes closer to the outlet would require a larger proximity distance than those near the inlet to attain the same level of accuracy. In other words, for the same value of proximity used for all neurons in a 3D ICs, the training residuals (or the run-time errors) for neurons near the outlet would be larger than those near the inlet.

Using a small constant proximity distances to reduce the simulator complexity would result in large errors near the outlet. On the other hand, using extremely large constant proximity distances to compensate for these errors near the outlet increases the complexity of the entire simulator, while being an overkill for neurons near the inlet. Hence, it is possible to find an *optimal* proximity distance profile for the neurons in a 3D IC as a function of the distance from the inlet, which minimizes the NN complexity, while distributing the errors fairly uniformly. This proximity distance profile would reflect the amount of lateral heat spreading that occurs at a particular point in the 3D IC along the microchannel. The residual resulting form the training algorithm can be used to estimate this optimum proximity distance for each row of neurons along the microchannel. This modified training algorithm for optimal proximity is shown in Algorithm 2.

The idea behind this algorithm is to use the given proximity to compute the weights only for the neurons in the last row and save the maximum value of the residual found in this first part (lines 1-10). Then, the proximity is set to a minimal value (line 11) and weights are computed processing one row (in all the layers) per time. If for any neuron a residual higher than the maximum found in the last row is found, then the proximity value is increased and the training of the row is restarted (line 19-21). This approach allows to identify, for each row, the minimum proximity value that will guarantee a run-time error lower than the error in the last row of neurons.

The value $MinLength$ used in line 11 and 20 to find the optimal proximity, corresponds to the minimum distance such that the operation done by `ExtractInput` extracts a larger set of inputs. For instance, using Figure 3 as reference, $MinLength$ is the value that turns the proximity region of the neuron in the bottom layer into a region as large as the one that belongs to the neuron in the top layer. In the setup of our model, we chose to set $MinLength$ as the length of the thermal cell in the compact model used to generate the training samples.

4.4 Running the NN-based simulator

Once the proposed NN-based simulator is trained, it can be used to simulate the temperatures of the target liquid-cooled 3D ICs using matrix vector multiplications. Essentially, the weights computed during the training are stored in the form of a sparse matrix in the GPU global memory. Then, the power traces and the thermal states are sent from the CPU memory whenever needed. These form the vector

Algorithm 2 Solving for optimal proximity

```
1: $MaxRes \leftarrow 0$
2: for $c_i = 1 ... N_C$ do
3:    $I \leftarrow$ ExtractInput ($TrainSet, Proximity, 1, N_R, c_i$)
4:    $(Q, R) \leftarrow$ QRdecompose ($I$)
5:    for $l_k = 1 ... N_L$ do
6:       $O \leftarrow$ ExtractOutput ($TrainSet, l_k, N_R, c_i$)
7:       $(W[l_k, N_R, c_i], Residual) \leftarrow$ lssolve ($O, Q, R$)
8:       $MaxRes \leftarrow$ max ($MaxRes$, max($|Residual|$))
9:    end for
10: end for
11: $Proximity \leftarrow MinLength$
12: for $r_i = 1 ... N_R - 1$ do
13:    for $c_i = 1 ... N_C$ do
14:       $I \leftarrow$ ExtractInput ($TrainSet, Proximity, 1, 1, c_i$)
15:       $(Q, R) \leftarrow$ QRdecompose ($I$)
16:       for $l_k = 1 ... N_L$ do
17:          $O \leftarrow$ ExtractOutput ($TrainSet, l_k, r_i, c_i$)
18:          $(W[l_k, r_i, c_i], Residual) \leftarrow$ lssolve ($O, Q, R$)
19:          if max($|Residual|$) $> MaxRes$ then
20:             $Proximity \leftarrow Proximity + MinLength$
21:             restart $c_i$ loop
22:          end if
23:       end for
24:    end for
25: end for
```

Table 1: Structural parameters of the Test 3D IC

Parameter	Value	Parameter	Value
Layers per die	2	Channel height	$100\mu m$
Die size	14×15 mm^2	Channel width	$50\mu m$
Die height	$65\mu m$	Channel pitch	$100\mu m$

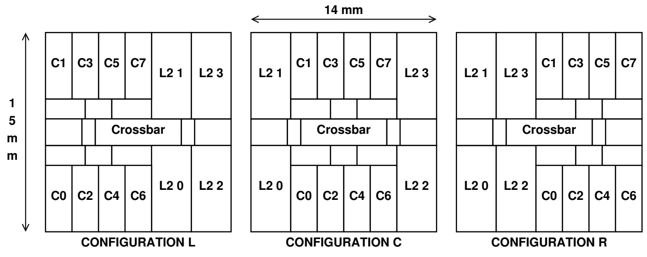

Figure 4: Niagara floorplan configurations used for testing

of inputs, which get multiplied by the weights to obtain the thermal state in the next time step. In our implementation, we used the cuSPARSE library [11] for these computations on our GPU platform.

5. EXPERIMENTAL RESULTS

To illustrate the accuracy and the performance of the proposed NN-based simulator we define a *Test 3D IC* as a stack made up of three dies interleaved by two cavities. The structural properties of the stack are shown in Table 1. To study the effect of cooling effort on the simulator, all experiments were performed using the three flow rates: 24ml/min, 36ml/min, and 48ml/min. A cell size of $500\mu m \times 500\mu m$ was used to discretize the structure and the training data for the NN-based simulator were generated by simulating it using 3D-ICE. This fixes the number of neurons in the network to 2520, as we define a neuron for each thermal cell in the three active layers.

During the training phase, data was generated using a randomized floorplan with random power traces bounded by a maximum heat flux of 90W/cm^2. The number of training samples generated was set to 20% more than the minimum number required. On the other side, during the running and measurement phase, the UltraSPARC Niagara floorplan [10] is assigned to the active layer of each die. This floorplan configuration was modified to generate three different test cases as shown in Figure 4. In the following experiments, "LLL" refers to the case when all the dies have the floorplan configuration "L" in Figure 4. Similarly, "LRL" represents a stack with the floorplans "L" in the bottom and top die, and "R" in the middle die, etc.

The NN-based simulator is run on the NVIDIA Tesla^{TM} c2070 (448 CUDA Cores running at 1.15 GHz and 6GB GDDR5) while the corresponding simulations on CPU based on 3D-ICE are run on Intel$^{®}$ Core^{TM} i7 920 (4 cores running at 2.67 GHz and 6GB of RAM).

5.1 Computation of Optimal Proximity

We tested the ability of Algorithm 2 to compute the optimal proximity by first training the Test 3D IC using Algorithm 1 with a series of different constant proximity values. During each training, the maximum value of residuals for each neuron was stored. Then, the resulting NN was run to simulate the floorplan configuration CCC and the maximum run-time error w.r.t. 3D-ICE was measured. Figure 5

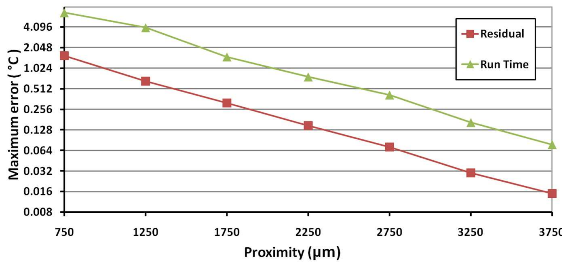

Figure 5: Max. residual and max. run-time error when training with Algorithm 1

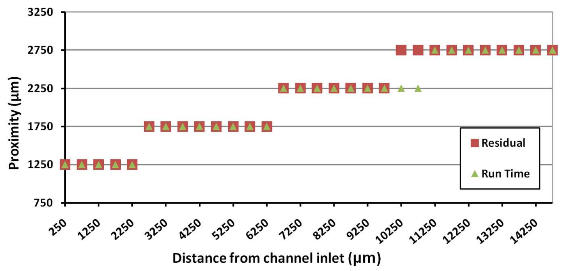

Figure 6: Profile of the optimal proximity along the channel

shows the maximum residual and corresponding the maximum run-time error for each proximity value.

As can be seen from this figure, the maximum residual from training and the maximum run-time error show the same trend. This means that the residual can be effectively used to compute the optimal profile of the proximity along the direction of the channel. Figure 6 shows the optimal proximity profile computed both using Algorithm 2 and using run-time error. To obtain the optimal proximity using the run-time error in this figure, we first train the NNs using Algorithm 1 with different proximity values and 36ml/min as flow rate. In each case, we measure the maximum runtime error as a function of distance along the channel. Then we fix, for each neuron, the minimum proximity value such that the run-time error remains lower than 0.5°C. The minimum proximity found for the neurons representing the last row was $2750\mu m$ and therefore this value has been given as input to Algorithm 2 to verify its capability to generate the same profile.

As Figure 6 shows, the two proximity profiles show the same trend except for a few values. Even when mismatches occur, the proximity value obtained using residuals from training is higher than the one obtained using run-time measurements. Hence, using the former methods results only in reduced error at run-time at a marginal additional cost of computation. Hence, we can conclude that Algorithm 2 is a reliable method to compute the optimal proximity profile during training and the run-time errors resulting from it is less than the intended tolerance.

5.2 Error analysis

Figure 7 shows the evolution of the maximum run-time error incurred by the proposed NN-based simulator (trained using Algorithm 2), when it is used to simulate Test 3D IC with CCC configuration. Errors in this figure are reported for the three different flow rates. As can be seen from these plots, the error decreases in each case with increasing proximity values. However, given the same proximity value, simulations with different flow rates result in different er-

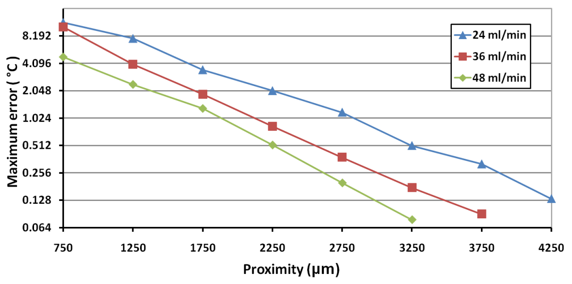

Figure 7: Maximum run-time error for the floorplan configuration CCC

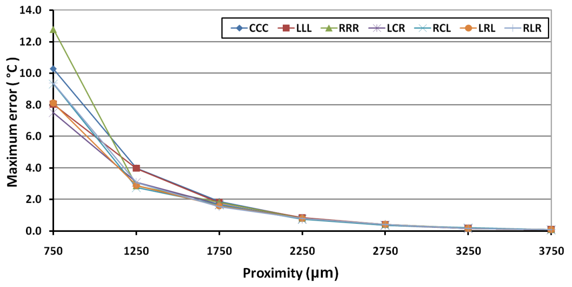

Figure 8: Maximum run-time error for different floorplan configurations

rors. Higher flow rates result in smaller errors than lower flow rates because coolant at high flow rates carries heat from the IC quicker, leading not only to lower IC temperatures, but also prevent heat spreading within the silicon by lowering the net thermal resistance to the heat sink. Hence, for a given proximity value, the NN-simulator trained for a higher flow rate better captures the heat spreading effects than an NN-simulator trained for lower flow rates.

Figure 8 reports the maximum run-time error obtained when NN-based simulator was used to simulate the Test 3D IC for the same flow rate (36ml/min) but with different floorplan configurations. As can be seen, the run-time independent of the floorplan configuration.

5.3 Performance analysis

To compare the performances of the two training techniques (Algorithms 1 and 2) simulation speed ups of NN-based simulators trained using both these techniques running on GPU, against 3D-ICE running on CPU. This experiment was repeated for different maximum proximity values and flow rates. Figure 9 shows the increase (in percentage) of speedup obtained with the introduction of the training Algorithm 2 over the Algorithm 1. As can be seen from this figure, improvements of up to 25% can be obtained by using the optimal proximity profiles.

Finally, to illustrate the scalability of our neural network-based thermal simulator, Figure 10 shows how the GPU speedup changes with increasing number of dies (in other words, the problem size), increasing flow rate and trained for various error tolerances (1.0°C, 0.5 °C and 0.1 °C). As can be seen from this figure, speedups increase with increasing problem size and also with increasing flow rate. In all these experiments, the dies are interleaved with channel cavities, similar to the Test 3D IC.

6. CONCLUSIONS

In this work we presented a Neural Network-based thermal model that can be run on massively parallel architectures

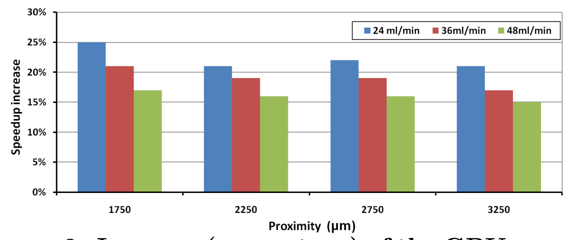

Figure 9: Increase (percentage) of the GPU speedup using Algorithm 2

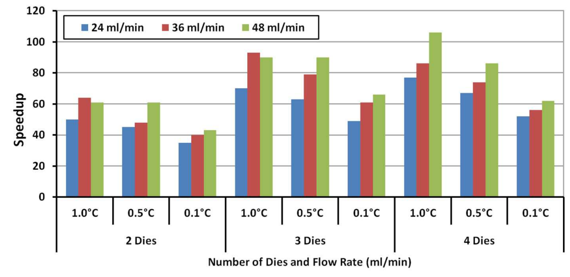

Figure 10: GPU speedup for different problem sizes, flow rates and maximum run-time error

like GPUs to accelerate thermal simulations of 3D ICs with liquid cooling. We also introduced a new training technique that exploits the horizontal heat flow due to the coolant passing through the channels to reduce the simulation time without worsening the run time error. Results show that the speedups obtained comparing the simulation times against a compact model running on CPU ranges from 35x, to limit the run-time error under 0.1°C, up to 106x if errors lower than 1.0°C are accepted.

7. ACKNOWLEDGMENTS

This work was funded in part by the Swiss NSF grant number 200021-130048 and by the Nano-Tera RTD Project CMOSAIC (ref. 123618), which is financed by the Swiss Confederation and scientifically evaluated by SNSF.

8. REFERENCES

[1] "International technology roadmap for semiconductors (ITRS)," *2009 Edition-ERD.*

[2] T. Brunschwiler *et al.*, "Interlayer cooling potential in vertical integrated packages," *Microsystem Technologies: MNSISPS*, vol. 15, no. 1, 2009.

[3] H. Mizunuma *et al.*, "Thermal modeling for 3D-ICs with integrated microchannel cooling," *Proc. ICCAD, 2009.*

[4] A. Sridhar *et al.*, "3D-ICE: Fast compact transient thermal modeling for 3D ICs with inter-tier liquid cooling", *Proc. ICCAD, 2010.*

[5] A. Sridhar *et al.*, "Compact transient thermal model for 3D ICs with liquid cooling via enhanced heat transfer cavity geometries", *Proc. THERMINIC, 2010.*

[6] Z. Feng and Z. Zeng, "Parallel Multigrid Preconditioning on Graphics Processing Units (GPUs) for Robust Power Grid Analysis", *Proc. DAC, 2010.*

[7] Y. Liu and J. Hu, "GPU-based parallelization for fast circuit optimization", *Proc. DAC, 2009.*

[8] J. Croix and S. Khatri "Introduction to GPU Programming for EDA", *Proc. ICCAD 09.*

[9] A. Sridhar *et al.*, "Neural Network-Based Thermal Simulation of Integrated Circuits on GPUs", *Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 31, no. 1, 2012.

[10] A. Leon *et al.*, "A power-efficient high-throughput 32-thread SPARC processor", *Proc. ISSCC 2007.*

[11] CUDA Sparse Library.