Full-chip Thermal Analysis of 3D ICs with Liquid Cooling by GPU-Accelerated GMRES Method

Xue-Xin Liu, Zao Liu, Sheldon X.-D. Tan, and Joseph Gordon
Dept. Electrical Engineering, University of California, Riverside, CA 92521

***Abstract*—Cooling and related thermal problems are the principal challenges facing 3D integrated circuits (3D-ICs). Active cooling techniques such as integrated inter-tier liquid cooling are promising alternatives for traditional fan-based cooling, which is insufficient for 3D-ICs. In this regard, fast full-chip transient thermal modeling and simulation techniques are required to design efficient and cost-effective cooling solutions for optimal performance, cost and reliability of packages and 3D ICs. In this paper, we propose an efficient finite difference based full-chip simulation algorithm for 3D-ICs using the GMRES method based on GPU platforms. Unlike existing fast thermal analysis methods, the new method starts from the physics-based heat equations to model 3D-ICs with inter-tier liquid cooling microchannels and directly solves the resulting partial differential equations using GMRES. To speedup the simulation, we further develop a preconditioned GPU-accelerated GMRES solver, GPU-GMRES, to solve the resulting thermal equations on top of some published sparse numerical routines. Experimental results show the proposed GPU-GMRES solver is up to $4.3\times$ faster than parallel CPU-GMRES for DC analysis and $2.3\times$ faster than parallel LU decomposition and one or two orders of magnitude faster than the single-thread CPU-GMRES for transient analysis on a number of thermal circuits and other published problems.**

I. Introduction

Three dimensional stacked integrated circuits (3D-ICs) are valuable for their massive bandwidth improvements while reducing effective chip footprint [1]. 3D stacked integration enables heterogeneous integration of cores, memories, and analog devices, and overcomes the barriers in interconnect scaling. However, it also introduces new challenges due to the high power density resulting from the placement of computational units on top of each other. Among these challenges, cooling and thermal problems are of high priority and are hot research topics in this area.

To remove the excessive heat in 3D chips, traditional fan-based cooling techniques are not sufficient due to their limited heat removal capabilities [2]. Active cooling techniques, such as embedded microchannel cooling, are a promising alternative. Microchannel based liquid cooling technique can remove up to 200–400 W/cm^2 and has the potential to reach 1000 W/cm^2 [3], [4]. In this regard, fast full-chip thermal modeling and simulation techniques [5], [6] are required to design efficient and cost effective cooling solutions for optimal electrical performance and reliability.

This research was supported in part by NSF grants under No. CCF-1116882, No. CCF-1017090, No. OISE-1130402, No. OISE-0929699. and in part by a UC MEXUS-CONACYT collaborative research grant (2011-2013).

Traditional thermal analysis solves the partial thermal diffusion equation directly using numerical approaches such as finite difference method, finite element method, and computational fluid dynamics. This process is computationally intensive, especially for large-scale 3D-ICs, as it requires solving a large number of linear equations given by the equivalent thermal circuit. Thus, fast linear solvers are crucial in transient thermal simulation.

Graphics Processing units (GPU), with their massively parallel architecture, are perfect for taking advantage of the inherent parallelism in linear algebra [7], [8]. Currently, GPUs or GPU-clusters can easily deliver terascale computing, which was only available on super-computers in the past, for solving many scientific and engineering problems. To date, dense linear algebra support on the GPU is well developed, with its own BLAS implementation [9], but sparse linear algebra support is still limited. In [6], a CPU based sparse LU solver was used to simulate the thermal systems. However, sparse LU solvers have complicated forms of parallelism if compared to dense solvers, and are tough to be implemented on today's high performance GPUs. On the other hand, iterative solvers such as Generalized minimum residual method (GMRES) [10] have a significant amount of parallelism. And when combined with a good preconditioner to improve convergence, GMRES can be faster than a sparse LU solver especially for large problems.

In this paper, a preconditioned GMRES solver is implemented on GPU platforms to speedup the transient thermal simulation of large-scale 3D-ICs with integrated liquid cooling inter-tier microchannels. Unlike existing fast thermal analysis methods, where approximation or compact models are used for liquid cooling microchannels, the new method starts from the channels' physical heat equations to model 3D-ICs, and directly solves the resulting partial differential equations using GMRES. To mitigate the excessive computational cost of solving large linear systems, we explore the massive parallelism in general purpose GPUs. Experimental results show that our GPU-GMRES solver is up to $4\times$ faster than parallel LU solvers [11] and parallel CPU versions of GMRES [12] over a number of thermal circuits and other published problems.

This paper is organized as follows. Section II reviews 3D-ICs with liquid cooling structure and their features from a computing perspective. Section III introduces modeling and analysis method for 3D ICs with integrated microchannels. Section IV describes the proposed GPU-GMRES parallel algorithm, followed by several numerical examples in section V.

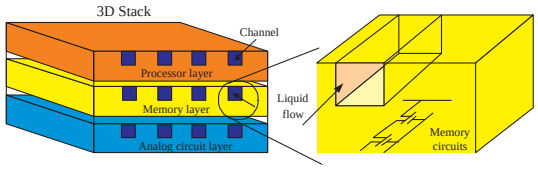

Fig. 1: 3D stacked IC with inter-tier liquid cooling

Last, Section VI concludes the paper.

II. Background

A. 3D-ICs with integrated inter-tier microchannels

Fig. 1 shows a 3D system consisting of a number of stacked layers (with cores, L2 caches, crossbar, memory controllers, buffers, etc.) and microchannels built in-between the vertically stacked layers for liquid cooling. Forced inter-layer convective cooling with water is applied [13], and the microchannels are distributed uniformly, and fluid flows through each channel at the same flow rate. The liquid flow rate provided by the pump can be dynamically altered at runtime.

Laminate liquid flows in the microchannels make the resulting heat equations more complicated as heat is removed by both heat sinks and laminate liquid flows. To mitigate this problem, some simple models were proposed as an add-on to the existing thermal models for package and chips at the cost of the accuracy. In [14], [6], the liquid cooling effects are modelled by simplified RC networks with voltage-controlled current sources to model the dominant convective heat flow (in flow direction). In [15], a simple resistor model is proposed for the liquid cooling microchannels. In this paper, we consider both conductive heat flow in the solid (chips and package) and the convective heat flow in the coolant flow, and directly solve the resulting partial differential equations without any approximation using finite different method.

B. GPU and GPU-enabled numerical libraries

GPU has been widely used for scientific computation in the past few years, thanks in part to NVIDIA CUDA (Compute Unified Device Architecture) [16]. GPU can easily perform $10\times$ faster than multi-core CPU and is very affordable. The latest CUDA GPU architecture, Fermi, has 3.0 billion transistors and up to 512 CUDA cores, while Intel's latest Xeon CPU has 2.3 billion transistors and can handle up to 16 threads. CPU and GPU have different design philosophies. CPU dedicates a large percentage of available transistors to cache and have only a few ALUs (arithmetic logic unit). As a result, CPU can quickly run complex serial programs with low memory access latency. On the contrary, GPU posesses several hundreds of ALUs in the parallel streaming multiprocessors, while it only uses a small percentage of available transistors to cache. Hence, GPU lends itself to many linear algebra problems, since they commonly are data intensive and have strong parallel nature. To support tbe linear algrbra computation, NVIDIA has released two libraries, CUBLAS and CUSPARSE [9], [17]. Additionally, the potential for fast linear algebra on the GPU has been recognized by third parties who have released such libraries as MAGMA [7], CULA [8] and CUSP [18].

C. Some relevant existing works

Recently there have been several works published on GPU based circuit analysis and simulation. In [19] a multigrid solver is used for DC analysis of power grids, while [20] used multigrid as a preconditioner for the conjugate gradient method for DC analysis of power distribution networks. The same group also applied their multigrid/conjugate gradient solver to the same problem addressed in this paper by using a purely resistance based symmetric model [15]. While this model has the benefit of being symmetric, it is unclear how to control the rate of flow through the microchannels by adjusting the resistor values. Instead, we base our work on the model proposed in [6], which allows specific rates of flow through the microchannels by modeling convection along the microchannels as a controlled current source.

There are also several papers implementing GMRES on GPU. In [21], GMRES with a block ILU preconditioner is parallelized on GPU. However, it only makes a small part of GMRES parallel, leaving many operations in serial. A more thorough GPU implementation of GMRES is proposed in [22], but it uses an inefficient sparse matrix format [23] and does not mention what precision was used for testing, making comparison difficult. The most recent paper [24] measures precision by comparing their GPU-GMRES to a customized CPU based solver, rather than to a validated solver, leaving open a question about the actual accuracy of their implementation.

III. Modeling of 3D-ICs with integrated microchannels

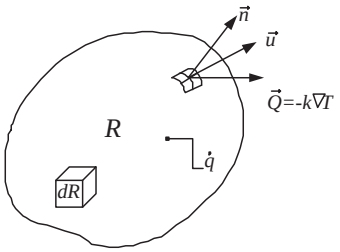

Fig. 2: Energy conservation for a control volume

In this section, we develop a thermal model for 3D stacked ICs with embedded microchannels and heat sinks from the basic heat equations. Using energy conservation in a control volume as shown in Fig. 2, the heat transfer equation of incompressible material can be written as [25]

$$\frac{d}{dt}\int_R \rho\hat{u}dx = -\int_S (\rho\hat{h})\vec{u}\cdot\vec{n}dS - \int_S (-k\nabla T)\cdot\vec{n}dS + \int_R \dot{q}dR, \quad (1)$$

where R is the control volume, S is its surface, $\vec{n}$ is normal to the surface, ρ is the density of the material,$\hat{u}$ is the specific

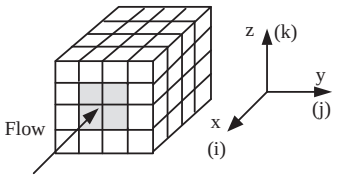

Fig. 3: Meshed chip cell with coolant channel

internal energy, $\hat{h}$ is the convection coefficient, $\vec{u}$ is the flow rate, k is the thermal conductivity of the material, T is the temperature, and $\dot{q}$ is the volumetric rate of the heat generation inside R. By applying Gauss' theorem to Eq. (1) and using $du = c_p dT$, the general form of heat equation can be written as

$$\rho c_p \frac{\partial T}{\partial t} = -\rho c_p \vec{u} \cdot \nabla T + k\nabla^2 T + \dot{q}, \tag{2}$$

where c_p is the specific heat. Applying finite difference method and assuming the channel is along x-axis, as illustrated in Fig. 3, the discretized form of Eq. (2) is

$$\begin{aligned}\rho c_p \frac{dT}{dt} = &\frac{k_{xx}}{\Delta x^2}(T_{i+1,j,k} - 2T_{i,j,k} + T_{i-1,j,k}) \\ &+ \frac{k_{yy}}{\Delta y^2}(T_{i,j+1,k} - 2T_{i,j,k} + T_{i,j-1,k}) \\ &+ \frac{k_{zz}}{\Delta z^2}(T_{i,j,k+1} - 2T_{i,j,k} + T_{i,j,k-1}) \\ &+ u_{xx}\frac{\rho c_p}{2\Delta x}(T_{i+1,j,k} - T_{i-1,j,k}) + g(\vec{v}, t),\end{aligned} \tag{3}$$

where $T_{i,j,k}$ is the temperature on meshed grid (i, j, k), k_{xx}, k_{yy}, and k_{zz} are thermal conductivities along x, y, and z axes, respectively, u_{xx} is the flow rate in x direction, $g(\vec{v}, t)$ is the heat generation in volume $\Delta v = \Delta x \Delta y \Delta z$. Therefore, Eq. (3) models the thermal behavior of 3D-IC stack as a thermal circuit, and it can be rearranged into the form of ordinary differential equation,

$$GT(t) + C\frac{dT(t)}{dt} = BU(t) \tag{4}$$

where G and C are the coefficient matrices that represent the thermal conductivities and capacitance, B is the input position matrix of the heat sources, $T(t)$ is the matrix of on-chip temperature, and $U(t)$ is the matrix of input power sources.

At the sidewall of the channel, boundary conductances $g_{side} = h_{side}S$, $side = top/bottom/left/right$, can be used to model the heat exchange from the channel sidewall to the coolant as shown in Fig.5, where h_{side} is the convection coefficient at the sidewall of microchannel [26] and S is the area of the convective surface.

Notice that in microchannels, each cell has the terms

$$I_c = \rho C_p u_{xx} \frac{T_{i+1,j,k} - T_{i-1,j,k}}{2\Delta x} \tag{5}$$

which represent the convective heat transfer along the flow direction and are the dominant terms (compared to all the other conductive terms in (3)). The conductive heat flow linearly depends on the temperature difference between the two boundaries of the cell along the flow direction. As a result, they can be viewed as temperature-controlled heat sources (or voltage controlled current sources in circuits). Since we have the controlled sources, the resulting G matrices no longer symmetric. For cells in solids (chips or packages), the flow rate $\vec{u}_{xx}$ is 0 and thus there is no convection term.

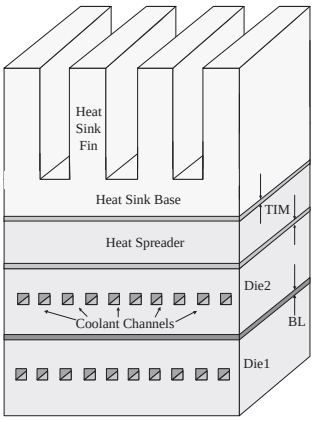

Fig. 4: A view of 3D-IC stack

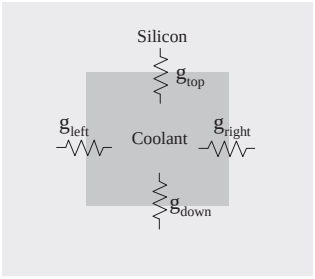

Fig. 5: Model of heat transfer between coolant and the sidewall of the channel

We remark that our model is different from the simplified circuit model proposed in [15] that uses very small thermal resistors R_f in coolant flow direction to model the convective heat exchange as shown in Fig. 6 (b). While the model in this paper uses the heat-controlled temperature flow (voltage-controlled current sources) naturally derived from the energy equation (3) to model the heat convection in the direction of the channel without macro-modeling based approximations process. This is also in contrast with the macro-modeling based works in [14], [6].

Without loss of generality, the example we used in our testing case is a 3D-IC stack IC shown in Fig. 4. It consists of 2 active layer stacked IC with a heat sink on the top, and macrochannels embedded in the active silicon layers. The geometrical and material properties of the test stack are listed in Table I, where BL represents the bounding layer between two dies, and TIM represents the thermal interface materials. Inside the channel we assume the coolant flows at a constant rate.

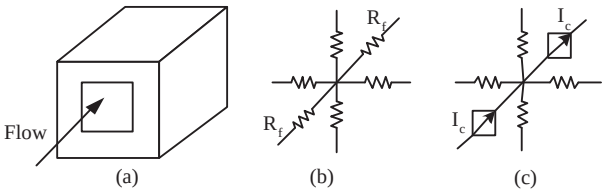

Fig. 6: (a) Coolant flow inside a channel; (b) Modeling heat convection in the direction of the channel using thermal resistor R_f; (c) Modeling heat convection in the direction of the channel using current sources I_c derived from the energy equation.

TABLE I: Geometrical and material information of the 3D structure.

Layers	Geometry (mm)	Material
Die	$8 \times 8 \times 1$	Silicon
TIM	$8 \times 8 \times 0.25$	Indium
BL	$8 \times 8 \times 0.25$	Silicon Nitride
Heat spreader	$8 \times 8 \times 0.5$	Copper
Heat sink base	$8 \times 8 \times 0.5$	Aluminium
Heat sink fin	$8 \times 8 \times 3$	Aluminium

IV. GPU-ACCELERATED GMRES SOLVER

The generalized minimum residual (GMRES) method is an iterative method for solving large-scale systems of linear equations ($Ax = b$), where A is sparse in our case. Algorithm 1 shows the standard Krylov-subspace based GMRES method with left preconditioning [27], which uses projection method to form the m-th order Krylov-subspace [10], [27], e.g.,

$$\mathcal{K}_m = \text{span}(\mathbf{r_0}, MAr_0, (MA)^2 r_0, \ldots, (MA)^{m-1} r_0), \quad (6)$$

After orthogonalization and normalization, the orthonormal basis of this subspace is V_m. To generate the Krylov subspace in GMRES, Arnoldi iteration is employed to form the V_m. Each Arnoldi iteration generates a new basis vector and is appended to the previous Krylov subspace basis $\mathcal{K}_j$ to obtain the augmented subspace $\mathcal{K}_{j+1}$. The Arnoldi iteration also creates an upper Hessenberg matrix $\tilde{H}_m$ used to check the solution at the current iteration. As a result, the approximated solution x becomes the linear combination of $x_m = x_0 + V_m y_m$, where y_m is calculated in Line 12 of Algorithm 1.

Algorithm 1 GMRES with Left Preconditioning

Input: $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$, $x_0 \in \mathbb{R}^n$ (initial guess), $M \in \mathbb{R}^{n \times n}$(preconditioner), m (restart)
Output: $x \in \mathbb{R}^n$: $Ax \approx b$
1: $r_0 \leftarrow M(b - Ax_0)$
2: $\beta \leftarrow \|r_0\|_2$, $w \leftarrow r_0/\beta$
3: **for** $j = 1$ to m **do**
4: $\quad w = MAv_j$
5: $\quad$ **for** $i = 1$ to j **do**
6: $\quad\quad h_{i,j} \leftarrow w_i^T v_j$
7: $\quad\quad w \leftarrow w - h_{i,j} v_i$
8: $\quad$ **end for**
9: $\quad h_{j+1,j} \leftarrow \|w\|_2$, $v_{j+1} \leftarrow w/h_{j+1,j}$
10: **end for**
11: $V_m \leftarrow [v_1, \ldots, v_m]$, $\tilde{H}_m \leftarrow \{h_{i,j}\}_{1 \le i \le j+1; 1 \le j \le m}$
12: $y_m \leftarrow \text{argmin}_y \left\| \beta e_1 - \tilde{H}_m y \right\|_2$, $x_m \leftarrow x_0 + V_m y_m$
13: If satisfied So, else $x_0 \leftarrow x_m$ and GoTo 1

A. Parallelization on GPU platforms

To parallelize the GMRES algorithm, we need to identify several computation intensive steps in Algorithm 1. There exist many GPU-friendly computing operations in GMRES, such as the vector addition (`axpy`), 2-norm of vector (`nrm2`), and sparse matrix-vector multiplication (`csrmv`) SpMV. Based on the examples we focused on, we found that SpMV takes up 50% of the overall runtime to build the Krylov subspace shown in (6). Those routines have been parallelized in generic parallel algorithms for sparse matrix computations library (CUSP) [18].

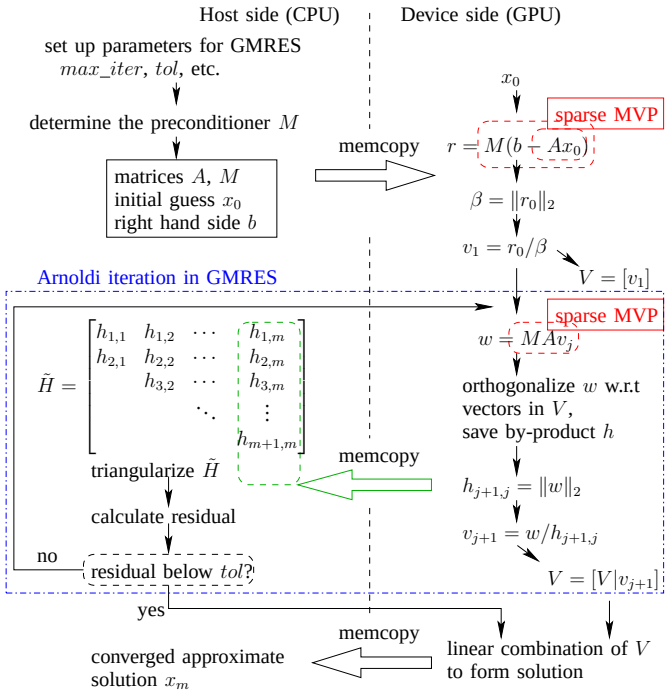

Fig. 7: GPU parallel solver for envelope-following update.

GPU programming is typically limited by the data transfer bandwidth as GPU favors computationally intensive algorithms [28]. Hence how to wisely partition the data between CPU memory and GPU memory to minimize the data traffic is crucial for GPU programming. Let us first make a rough sketch of the quantities in the GMRES Algorithm 1. Although GMRES tends to converge quickly for most circuit examples, i.e., the iteration number $m \ll n$, the space for storing all the subspace basis V_m of n-by-m, i.e., m column vectors with n-length, is still big. In addition, every newly generated matrix-vector product needs to be orthogonalized with respect to all its previous basis vectors in the Arnoldi processes. To utilize the data intensive capability, we keep all the vectors of v_m in GPU global memory allows GPU to handle those operations, such as inner-product of basis vectors (`dot`) and vector subtraction (`axpy`), in parallel.

On the other hand, it is better to keep the Hessenberg matrix $\tilde{H}$, where intermediate results of the orthogonalization are stored, at the host side. This comes with the following reasons. First, its size is $(m+1)$-by-m at most, rather small if compared with circuit matrices and Krylov basis. Besides, it is also necessary to triangularize $\tilde{H}$ and check the residual regularly in each iteration so the GMRES can return the approximate solution as soon as the residual is below a preset tolerance. Hence, in consideration of the sequential nature of the triangularization, the small size of Hessenberg

matrix, and the frequent inspection of values by host, it is preferable to allocate $\tilde{H}$ in CPU (host) memory. As shown in Algorithm. 1, the memory copy from device to host is called each time when Arnoldi iteration generates a new vector and the orthogonalization produces a new vector h, which is the $j+1$ column of $\tilde{H}$, and is transferred to the CPU, where a least square minimization (a series of Givens rotations, in fact) is performed to see if the desired tolerance of our residual has been met. Our obsevation shows that the data transfer and subsequent CPU based computation takes up less than 0.1% of the run time.

Fig. 7 illustrates the compuation flow, and the memory access behavior of CPU and GPU during the operations we mentioned above.

B. Preconditioners

Preconditioners are used to increase the rate of convergence for GMRES. A well chosen perconditioner will potentially making GMRES much faster than the one without preconditioner.

The asymmetric Bridson Approximate Inverse (AINV) preconditioner [18], [29], [30], with tolerance dropping is used as the preconditioner for our 3D IC thermal analysis. This method is chosen for our GPU implementation, since its works in a way that is friendly to GPU parallel operation and hence is easily embedded into the aformentioned GMRES algorithm flow of GPU computation. In addition, according to [31], AINV is simpler and more robust than ILU(0), but its behavior is very similar to ILUT. It is noteworthy that AINV requires a sparse incomplete LU factorization with level 0 of fill in, ILU(0), and hence is carried out on CPU for its serial computation nature. The resulted AINV preconditioner is then transferred to GPU and applied in parallel. The sparse matrix format we use to store the preconditioner is the same as the one used in the framework of [18].

V. Numerical Results

The proposed algorithm has been implemented in NVIDIA CUDA and run on the Tesla C2070 GPU card with 448 cores running at 1.15 GHz with 5 GB of global memory. This card has a double precision floating point peak performance of 515 GFLOPS and 1.03 TFLOPS single precision peak performance.

Although the CPU results were tested on a dual Xeon E5620 machine (8 cores total) at 2.40 GHz with 36 GBytes memory, only one quad-core processor was used. For a thorough comparison, serial GMRES and parallel GMRES on CPU [12] and GPU are directly compared, and a parallel LU solver, superLU_MT [11], and GPU-GMRES with an AINV preconditioner are compared for the full thermal simulation on a number of packages. SuperLU_MT is a commonly preferred and publicly available parallel LU-based solver [6]. The parallel GMRES on CPU [12] is not used for the transient simulation because it has a convergence issue when the initial value for x is close to the correct value, and this scenario only happens during transient simulation. Instead, a regular GMRES algorithm is used for comparison.

TABLE II: Statistics for thermal circuits

Thermal Ckts	Dimension	nnz	% Nonzero
ckt1	7,168	47,360	9.1919e-04
ckt2	57,344	390,144	1.1864e-04
ckt3	114,688	781,824	5.9439e-05
ckt4	172,032	1,174,528	3.9687e-05

TABLE III: GMRES: CPU vs GPU (unsymmetric, GotoBLAS, restart$=32$, tol$=1e{-}9$, double precision)

Dataset name	n No. of rows	nnz No. of nonzeros	CPU serial time	CPU 4-core time	CPU iter	GPU time	GPU iter	speed up
FEM_3D _therm2	147,900	3,489,300	3.0	0.315	53	0.142	45	$2.22\times$
cage13	445,315	7,479,343	2.7	0.341	18	0.100	18	$3.41\times$
cage15	1,505,785	27,130,349	8.9	1.347	16	0.315	16	$4.28\times$

For testing, our testcases consist of four 3D-ICs with package systems. The thermal circuits have varied sizes after the finite difference discretization and they are simulated over 1600 time-steps. More information on these examples can be found in Table II, where nnz is the number of nonzeros in the matrices.

First, we show the motivation for running the thermal simulation using GPU-GMRES. GMRES on the CPU and GPU (using a Jacobi preconditioner) were tested on several large matrices from the University of Florida Sparse Matrix Collection [32]. The names of the matrices along with the results are listed in Table III. From the table we notice that GPU-GMRES is faster than the parallelized CPU GMRES. A $3.41\times$ speedup at 7.5 million nonzeros (in matrices) and $4.3\times$ at 27 million nonzeros were obtained. As the problem sizes become larger, GPU-GMRES becomes faster.

Next, we perform the comparison in the transient thermal simulation. Since the matrix A does not change on the transient time steps, the preconditioner M only needs to be calculated once and repeatedly used for several thousand times. Therefore, it worths the time to calculate a preconditioner which is set to a more accurate, even though it is computationally expensive. In addition, as the transient simulation goes on, the right hand side b does not change a lot in the nearby time steps and usually the new x_t is quite close to its previous state x_{t-1}. Hence, to speedup GMRES, the initial guess to solve for x_t is set to be the solution of the previous time step, x_{t-1}. This strategy reduces the average number of iterations significantly. On the examples we tested, our solver only takes 18 iterations for the smallest circuit and 47 for the largest one. The results are shown in Table IV. All time measurements consider every procedure except the initial file reading of the data (G, C, and B matrices) into memory.

Fig. 8 shows the thermal profile of the 3D circuit with two active layers and microchannels and heat sink with the convective interface. The flow rate inside the microchannels are fast enough to remove heat instantaneously, thus, the

TABLE IV: Transient runtime in seconds (4 cores, GotoBLAS, restart$=32$, tol$=1e{-}9$, single precision)

Thermal circuit	superLU_MT	serial CPU GMRES	GPU GMRES	Speedup
ckt1	10.45	1166	67.59	$0.15\times$
ckt2	292.77	19422	266.29	$1.10\times$
ckt3	1123.46	43921	520.02	$2.16\times$
ckt4	1928.14	91694	860.37	$2.24\times$

temperature distribution inside the microchannels is almost constant (difference is $0.005\,^{\circ}\mathrm{C}$) as Fig. 9 shows.

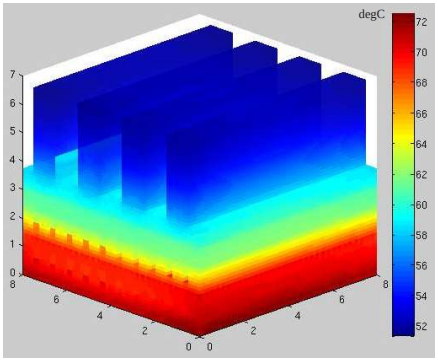

Fig. 8: Temperature profile of the 3D IC with two active layers

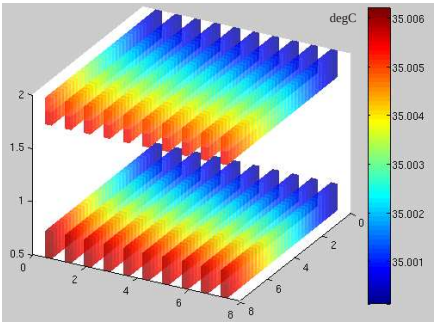

Fig. 9: Temperature profile inside the coolant channels

VI. Conclusion

An efficient finite difference based full-chip simulation algorithm for 3D-ICs based on GPU is proposed. Unlike existing fast thermal analysis methods, our method starts from the heat equations to model 3D-ICs with inter-tier liquid cooling microchannels, and directly solves the resulting PDE using GMRES. We gain further speedup by developping a preconditioned GPU-accelerated GMRES solver, Experimental results show that the proposed GPU-GMRES solver is up to $4.3\times$ faster than parallel CPU-GMRES for DC analysis and $2.3\times$ faster than parallel LU decomposition and one or two orders of magnitude faster than the single-thread CPU-GMRES for transient analysis on a number of thermal circuits and other published problems.

References

[1] B. Black, M. Annavaram, *et al.*, "Die stacking (3d) microarchitecture," in *Proceedings of MICRO-39*, 2006.

[2] J. L. Ayala, A. Sridhar, and D. Cuesta, "Thermal modeling and analysis of 3D multi-processor chips," *Integration, the VLSI Journal*, vol. 43, pp. 327–341, September 2010.

[3] "IBM Interlyaer cooling technology for 3D packages." http://www.zurich.ibm.com/st/cooling/integrated.html.

[4] A. K. Coskun, D. Atienza, *et al.*, "Energy-efficient variable-flow liquid cooling in 3D stacked architectures," in *Proc. European Design and Test Conf. (DATE)*, pp. 111–116, IEEE Press, 2010.

[5] X. Wei and Y. Joshi, "Optimization study of stacked micro-chanel heat sinks for micro-electronic cooling," *IEEE Transaction on Components and Packaging Technologies*, vol. 26, pp. 441–448, 2003.

[6] A. M. Sridhar, A. Vincenzi, *et al.*, "3D-ICE: Fast compact transient thermal modeling for 3D-ICs with inter-tier liquid cooling," in *Proc. Int. Conf. on Computer Aided Design (ICCAD)*, pp. 463–470, IEEE Press, 2010.

[7] R. Nath, S. Tomov, and J. Dongarra, "An improved magma gemm for fermi gpus," 2010.

[8] J. R. Humphrey, D. K. Price, K. E. Spagnoli, A. L. Paolini, and E. J. Kelmelis, "CULA: hybrid GPU accelerated linear algebra routines," in *SPIE Defense and Security Symposium (DSS)*, April 2010.

[9] NVIDIA, "CUBLAS library," February 2011. http://developer.nvidia.com/cuBLAS.

[10] Y. Saad and M. H. Schultz, "GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. on Sci and Sta. Comp.*, pp. 856–869, 1986.

[11] J. W. Demmel, J. R. Gilbert, and X. S. Li, "An asynchronous parallel supernodal algorithm for sparse gaussian elimination," *SIAM J. Matrix Analysis and Applications*, vol. 20, no. 4, pp. 915–952, 1999.

[12] E. Chow, "Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns," *Int. J. High Perform. Comput. Appl.*, vol. 15, pp. 56–74, February 2001.

[13] T. Brunschwiler, B. Michel, *et al.*, "Interlayer cooling potential in vertically integrated packages," *Microsyst. Technol.*, vol. 15, pp. 57–74, October 2008.

[14] A. M. Sridhar, A. Vincenzi, *et al.*, "Compact transient thermal model for 3D ICs with liquid cooling via enhanced heat transfer cavity geometries," in *16th International Workshop on Thermal Investigation of ICs and Systems*, October 2010.

[15] Z. Feng and P. Li, "Fast thermal analysis on gpu for 3d-ics with integrated microchannel cooling," in *Proc. Int. Conf. on Computer Aided Design (ICCAD)*, pp. 551 –555, November 2010.

[16] "NVIDIA's next generation CUDA compute architecture: Fermi," 2009. White paper.

[17] NVIDIA, "CUSPARSE library," January 2011. http://developer.nvidia.com/cuSPARSE.

[18] N. Bell and M. Garland, "Cusp: Generic parallel algorithms for sparse matrix and graph computations," 2010. Version 0.2.0.

[19] Z. Feng and P. Li, "Multigrid on GPU: tackling power grid analysis on parallel SIMT platforms," in *ICCAD '08*, pp. 647–654, IEEE Press, 2008.

[20] Z. Feng, Z. Zeng, and P. Li, "Parallel on-chip power distribution network analysis on multi-core-multi-gpu platforms," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 10, pp. 1823–1836, 2011.

[21] M. Wang, H. Klie, *et al.*, "Solving sparse linear systems on NVIDIA Tesla GPUs," in *Proc of the 9th Intl. Conf. on Computational Science*, pp. 864–873, 2009.

[22] R. Li and Y. Saad, "GPU-accelerated preconditioned iterative linear solvers," 2010.

[23] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, (New York, NY, USA), pp. 18:1–18:11, ACM, 2009.

[24] R. Couturier and S. Domas, "Sparse systems solving on GPUs with GMRES," *The Journal of Supercomputing*, pp. 1–13–13, 2011.

[25] J. H. L. IV and J. H. L. V, *A Heat Transfer Textbook*. Phlogiston Press, 2003.

[26] R. Shah and A. London, *Laminar Flow Forced Convection in Ducts*. New York, USA: Academic Press, 1978.

[27] Y. Saad, *Iterative methods for linear systems*. PWS publishing, 2000.

[28] D. B. Kirk and W.-M. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*. San Francisco, CA: Morgan Kaufmann Publishers Inc., 2010.
[29] M. Benzi, C. D. Meyer, and M. Tuma, "A sparse approximate inverse preconditioner for the conjugate gradient method," *SIAM J. Sci. Comput.*, vol. 17, pp. 1135–1149, September 1996.
[30] R. Bridson and W.-P. Tang, "Refining an approximate inverse," *J. Comput. Appl. Math.*, vol. 123, pp. 293–306, November 2000.
[31] C. Ke, M. J. Ablowitz, S. H. Davis, E. J. Hinch, A. Iserles, J. Ockendon, and P. J. Olver, *Matrix Preconditioning Techniques and Applications; electronic version*. Cambridge: Cambridge Univ. Press, 2005.
[32] T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," *ACM Transactions on Mathematical Software*, 2011.