Transactions Briefs

Parallel Thermal Analysis of 3-D Integrated Circuits With Liquid Cooling on CPU-GPU Platforms

Xue-Xin Liu, Kuangya Zhai, Zao Liu, Kai He, Sheldon X.-D. Tan, and Wenjian Yu

***Abstract*— In this brief, we propose an efficient parallel finite difference-based thermal simulation algorithm for 3-D-integrated circuits (ICs) using generalized minimum residual method (GMRES) solver on CPU-graphic processing unit (GPU) platforms. First, the new method starts from basic physics-based heat equations to model 3-D-ICs with intertier liquid cooling microchannels and directly solves the resulting partial differential equations. Second, we develop a new parallel GPU-GMRES solver to compute the resulting thermal systems on a CPU-GPU platform. We also explore different preconditioners (implicit and explicit) and study their performances on thermal circuits and other types of matrices. Experimental results show the proposed GPU-GMRES solver can deliver orders of magnitudes speedup over the parallel LU-based solver and up to 4× speedup over CPU-GMRES for both dc and transient thermal analyzes on a number of thermal circuits and other published problems.**

***Index Terms*— 3-D-integrated circuit (IC), finite difference, generalized minimum residual method (GMRES), graphic processing unit (GPU) parallel computing, thermal analysis.**

I. Introduction

Three-dimensional integrated circuits (3-D ICs) technology has been viewed as a necessary driving force to maintain the trend described by Moores law [17], [23]. To remove the excessive heat in 3-D 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 promising alternatives. Microchannel-based liquid cooling technique can remove up to 200–400 W/cm^2 and has the potential to reach 1000 W/cm^2 [1], [8]. To design efficient 3-D IC structures and packages with advanced cooling solutions, accurate and fast detailed transient thermal analysis techniques are required [21], [25].

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 3-D ICs, as it requires solving a large number of linear equations given by the equivalent thermal circuit.

To significantly improve the simulation efficiency, exploiting the parallelism of simulation algorithms on multicore and many-core computing platforms becomes a viable solution. The family of graphic processing units (GPU) are among the most powerful multi-core computing systems in mass-market use [16]. On the other hand, iterative solvers such as generalized minimum residual (GMRES) method [18], which depend on many matrix-vector multiplications and other relatively cheap matrix operations, are more amicable for parallelization, especially on GPU platforms.

Manuscript received November 10, 2012; revised July 13, 2013 and November 11, 2013; accepted February 23, 2014. Date of publication April 7, 2014; date of current version February 20, 2015. This work was supported in part by NSF under Grant CCF-1017090 and Grant CCF-1255899, in part by the Semiconductor Research Corporation under Grant 2013-TJ-2417, in part by NSFC under Grant 61076034, and in part by Tsinghua University Initiative Scientific Research Program.

X.-X. Liu, Z. Liu, K. He, and S. X.-D. Tan are with the Department of Electrical Engineering, University of California at Riverside, Riverside, CA 92521 USA (e-mail: xliu@ee.ucr.edu; stan@ee.ucr.edu).

K. Zhai and W. Yu are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100090, China.

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

Digital Object Identifier 10.1109/TVLSI.2014.2309617

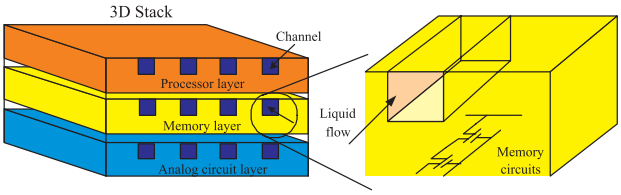

Fig. 1. 3-D stacked IC with intertier liquid cooling.

In this brief, we propose an efficient GPU-accelerated GMRES iterative solver for finite difference thermal analysis of 3-D ICs. First, the new method starts from fundamental physics principles to model 3-D ICs with intertier liquid cooling microchannels and directly solves the resulting partial differential equations, which consist of both convection of liquid flows and heat diffusions. Second, we develop a new parallel GMRES iterative solver to solve the resulting large thermal equation systems on CPU-GPU platforms. We carefully partition the computing tasks among CPU and GPU in the GMRES method to minimize the communication traffic between GPU and CPU and thus boost the overall performance. Third, we explore and compare different preconditioners (implicit and explicit) for GMRES on typical thermal matrices with liquid cooling channels and other types of matrices. We observe that the implicit preconditioners like incomplete LU (ILU) decomposition show better performance for thermal analysis. Experimental results show the proposed solver, called GPU-GMRES solver can deliver orders of magnitudes speedup over the parallel LU based solver and up to 4× speedup over CPU-GMRES solver for both dc and transient thermal analyzes on a number of thermal circuits and other published problems.

II. Background

A. 3-D-ICs With Integrated Intertier Microchannels

Fig. 1 shows a 3-D system consisting of a number of stacked layers (with cores, L2 caches, crossbars, memory controllers, buffers, etc.) and microchannels built in-between the vertically stacked layers for liquid cooling. The microchannels are distributed uniformly. Forced interlayer convective cooling with water is applied [7], 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 via convection effect. To mitigate this problem, some simple models were proposed as an add-on to the existing thermal models for packages and chips at the cost of accuracy. In [22] and [21], the liquid cooling effects are modeled by simplified RC networks with simplified voltage-controlled current sources to model the dominant convective heat flow (in flow direction). In [12], a simple resistor model is proposed for the liquid cooling microchannels. In this brief, we consider both conductive heat flow in the solid (chips and packages) and the convective heat flow in the coolant flow, and directly solve the resulting partial differential equations without any approximation.

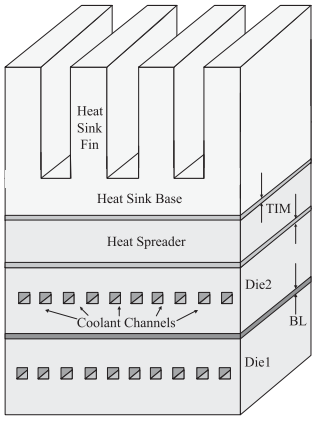

Fig. 2. View of 3-D-IC stack.

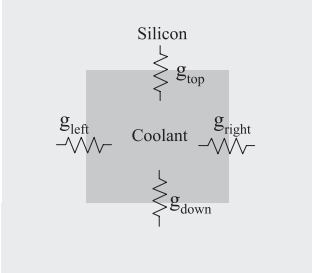

Fig. 3. Model of heat transfer between coolant and the sidewall of the channel.

B. Relevant Previous Arts of Linear Solvers

Recently, there have been several works published on GPU-based circuit analysis and simulation. Work in [11] uses multigrid as a preconditioner for the conjugate gradient method in dc analysis of power distribution networks. The work was later extended consider liquid cooling by using a purely resistance-based symmetric model [12]. While this model generates symmetric matrices, which are easy to solve and parallelize, its approximation to the fundamental physics of heat flow will compromise the accuracy.

There are also several papers implementing GMRES on GPU. In [24], GMRES with a block ILU preconditioner, which conducts ILU only along the main diagonal (DIAG) is parallelized on GPU. The block ILU preconditioner can easily provide some levels of parallelism to GPU. However, it drops the entries outside the DIAG blocks, and hence tends to be inaccurate and incur more iterations to converge. A more thorough GPU implementation of GMRES is proposed in [14], but it uses JAD and DIA sparse matrix formats, which are inefficient compared with NVIDIA-supported Compressed Sparse Row format [3]. In addition, [14] does not mention what precision is used for testing, making comparison difficult. Recently a GPU-accelerated preconditioned GMRES solver was proposed for solving power grid networks [15].

III. Modeling of 3-D-ICs With Integrated Microchannels

The thermal systems with both heat diffusion and convection due to the liquid flow (incompressible material) can be written in the following:

$$\rho c_p \frac{\partial T}{\partial t} = -\rho c_p \vec{u} \cdot \nabla T + k \nabla^2 T + \dot{q} \tag{1}$$

where c_p is the specific heat, ρ is density of the material, $\dot{q}$ is the rate of the heat generation, $\vec{u}$ is fluid's velocity field, and k is thermal conductivity of the material.

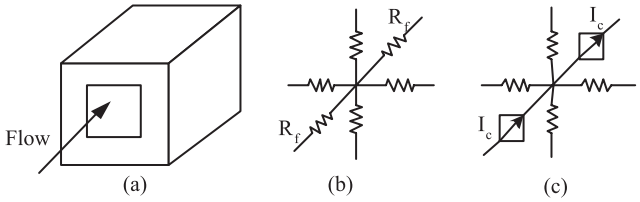

Fig. 4. (a) Coolant flow inside a channel. (b) Modeling heat convection in the direction of the channel using thermal resistor R_f [12]. (c) Modeling heat convection in the direction of the channel using current sources I_c derived from the energy equation.

TABLE I
Geometrical and Material Information of the 3-D 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	$8 \times 8 \times 0.5$	Aluminum
Heat sink fin	$8 \times 8 \times 3$	Aluminum

Applying finite difference method and assuming the channel is along x-axis, the discretized form of (1) will lead to

$$\mathbf{GT}(t) + \mathbf{C}\frac{\mathrm{d}\mathbf{T}(t)}{\mathrm{d}t} = \mathbf{BU}(t) \tag{2}$$

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

Boundary conductance is used to model the heat exchange between channel sidewall and the coolant, as shown in Fig. 3. It is calculated as $g_{\text{side}} = h_{\text{side}} S$, side $= \{\text{top/bottom/left/right}\}$, where h_{side} is the convection coefficient and S is the area of the convective surface [20].

Notice that in microchannels, each cell has the convective term (using central differencing)

$$I_c = A_{\text{cv}}(T_{i+1,j,k} - T_{i-1,j,k}) = A_{\text{ac}} T_{i+1,j,k} - A_{\text{ac}} T_{i-1,j,k} \tag{3}$$

where $A_{\text{cv}} = \rho c_p u_{xx}/2\Delta x$ represents the convective constant. $T_{i,j,k}$ is the temperature on meshed grid (i, j, k), u_{xx} is the flow rate which goes along x-direction. As we can see, the convective item I_c is essentially modeled by two voltage controlled current source terms, as shown in (3)-one representing convective heat transport from the previous cell to the current cell along the channel (controlled by the temperature $T_{i+1,j,k}$ at the front surface of the cell) by and the other representing the heat transport away from the current cell to the next cell (controlled by the temperature $T_{i-1,j,k}$ at the end surface of the cell).

The conductive heat flow linearly depends on the temperature difference between the two boundaries of the cell along the flow direction. Thus, they can be viewed as temperature-controlled heat sources (or voltage controlled current sources in circuit model). Since we have the controlled sources, the resulting $\mathbf{G}$ matrices will no longer be symmetric. Since cells of solids (chips or packages) has no liquid flow inside, there is no convection term.

We remark that our model is different from the simplified circuit model proposed in [12], which uses very small thermal resistors R_f in coolant flow direction to model the convective heat exchange, as shown in Fig. 4(b). Our model uses the heat-controlled temperature flow naturally derived from the energy (1) to model the unidirectional heat convection in the channel, as shown in Fig. 4(c).

Without loss of generality, the example used in our test case is a 3-D-IC stack IC shown in Fig. 2. It consists of two active layer stacked IC with a heat sink on the top, and microchannels embedded in the active silicon layers. The geometrical and material properties of the test stack are listed in Table I, where BL represents the

Algorithm 1 GMRES With Left Preconditioning. (Note: $\mathbf{e}_1$ Is the Unit Vector $[1, 0, \ldots, 0]$, and Vector $\mathbf{y}_m$ Is Calculated in the LS Problem, Which Minimizes the 2-Norm Residual)

Input: $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$, $\mathbf{x}_0 \in \mathbb{R}^n$ (initial guess), $\mathbf{M} \in \mathbb{R}^{n\times n}$ (preconditioner), m (restart)
Output: $\mathbf{x} \in \mathbb{R}^n$: $\mathbf{Ax} \simeq \mathbf{b}$

1: $\mathbf{r}_0 = \mathbf{M}(\mathbf{b} - \mathbf{Ax}_0)$
2: $\beta = \|\mathbf{r}_0\|_2$ *// Initial residual*
3: $\mathbf{v}_1 = \mathbf{r}_0/\beta$ *// First basis vector*
4: $\mathbf{a} = \beta \cdot \mathbf{e}_1$ *// RHS of LS problem in GMRES*
5: **for** $j = 1$ to m **do**
6: $\quad \mathbf{w} = \mathbf{MAv}_j$
7: $\quad$ **for** $i = 1$ to j **do**
8: $\quad\quad h_{i,j} = \mathbf{w}^{\mathrm{T}}\mathbf{v}_j$ *// Hessenberg matrix element*
9: $\quad\quad \mathbf{w} = \mathbf{w} - h_{i,j}\mathbf{v}_i$
10: $\quad$ **end for**
11: $\quad h_{j+1,j} = \|\mathbf{w}\|_2$ *// Hessenberg matrix element*
12: $\quad \mathbf{v}_{j+1} = \mathbf{w}/h_{j+1,j}$ *// New basis vector*
13: $\quad \mathbf{V}_m = [\mathbf{v}_1, \ldots, \mathbf{v}_m]$, $\tilde{\mathbf{H}}_m = \{h_{i,j}\}_{1\le i\le j+1,\ 1\le j\le m}$
14: $\quad$ Apply Givens rotations to solve LS problem: $\mathbf{y}_m = \mathrm{argmin}_{\mathbf{y}_m}\|\mathbf{a} - \tilde{\mathbf{H}}_m\mathbf{y}_m\|_2$
15: $\quad$ **if** a_{j+1} < preset tolerance **then** *// After Givens rotations, the updated residual is* a_{j+1}
16: $\quad\quad$ Return $\mathbf{x}_m = \mathbf{x}_0 + \mathbf{V}_m\mathbf{y}_m$
17: $\quad$ **end if**
18: **end for**
19: $\mathbf{x}_0 = \mathbf{x}_m$ and go to Line 1 *// Restart*

bounding layer between the two dies, and TIM represents the thermal interface materials. Inside the channel we assume the coolant flows at a constant rate.

IV. Parallel GMRES Solver on GPU-CPU Platform

The GMRES method is an iterative method for solving large-scale systems of linear equations ($\mathbf{Ax} = \mathbf{b}$), where $\mathbf{A}$ is sparse in our case. Algorithm 1 shows the standard Krylov subspace-based GMRES with left preconditioning, which uses projection method to form the mth order Krylov subspace [19].

A. Parallelization on GPU-CPU Platforms

To parallelize GMRES, we need to identify several computation intensive steps in Algorithm 1. There exist many GPU-friendly operations in GMRES, such as vector addition (axpy), 2-norm of vectors (nrm2), and sparse matrix-vector (SpMV) multiplication (csrmv). Based on the examples we focus on, we have noticed that SpMV takes up to 50% of the overall runtime to build the Krylov subspace. Those routines are GPU-friendly and efforts have been made already to accelerate them in generic parallel algorithms for sparse matrix computations library [4].

GPU programming is typically limited by the data transfer bandwidth as GPU favors computationally intensive algorithms [13]. Hence, how to wisely partition the data between CPU memory (host side) and GPU memory (device side) to minimize data traffic is crucial for GPU computing. In the sequel, we make some detailed analysis first for GMRES in Algorithm 1. First, we consider about storage of Krylov subspace vectors. In GMRES with restart mechanism, the maximum iteration number before the restart is m, $m \ll n$, and the memory cost of subspace $\mathbf{V}_m$ is n-by-m, i.e., m column vectors with n-length. This is still big. Hence, transferring the memory of the subspace vectors between the CPU and GPU memories is not an efficient choice. In addition, every newly generated matrix-vector product needs to be orthogonalized against to all its previous basis vectors in the Arnoldi processes. To use the data intensive capability

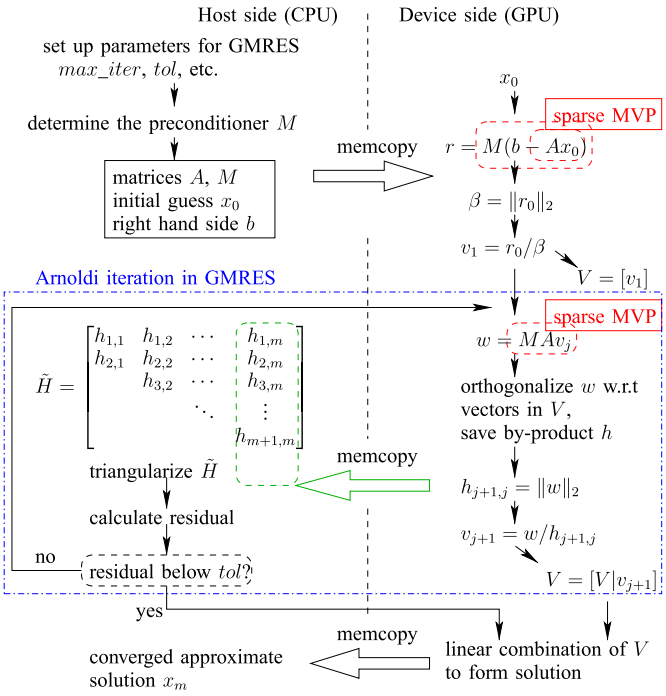

Fig. 5. Proposed GPU-accelerated parallel preconditioned GMRES solver. We also show the partitioning of the major computing tasks between CPU and GPU here.

of GPU, we keep all the subspace vectors in GPU global memory, thus allows GPU to handle 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 $\tilde{\mathbf{H}}$, where intermediate results of the orthogonalization are stored, at the CPU 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 vectors. Besides, it is also necessary to triangularize $\tilde{\mathbf{H}}$ and check the residual in each iteration so GMRES can return the approximate solution as soon as the residual is below a preset tolerance. Hence, it is preferable to allocate $\tilde{\mathbf{H}}$ in 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 $\mathbf{h}$, which is the $(j + 1)$th column of $\tilde{\mathbf{H}}$, and is transferred to the CPU, where a least squares (LS) minimization (a series of Givens rotations, in fact) is performed to see if the desired tolerance of residual has been met. Our observation shows that the data transfer and subsequent CPU-based computation takes up less than 0.1% of the total run time.

Fig. 5 shows the computation flow, the partitions of the major computing steps, and the memory accesses between CPU and GPU during the operations we mentioned above.

B. GPU-Friendly Implementation of Preconditioners

Most of the existing preconditioners can be broadly classified as being either explicit or implicit [5]. A preconditioner is implicit if its application within each step of the chosen iterative method requires to solve a linear system. With implicit preconditioner, we choose a nonsingular matrix $\mathbf{M}$, where $\mathbf{M} \approx \mathbf{A}$, and $\mathbf{M}$ should satisfy the requirement that solving a system with matrix $\mathbf{M}$ is easier than solving the original system of $\mathbf{A}$. In contrast, for explicit preconditioner, the approximate inverse (AINV) form of $\mathbf{A}$ is calculated first and will be known to iteration solvers. Hence, the preconditioning operations become one or more matrix-vector products.

The most common and widely used implicit preconditioner is based on ILU decomposition. For ILU preconditioner, $\mathbf{M} = \tilde{\mathbf{L}}\tilde{\mathbf{U}}$, where $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{U}}$ are the sparse triangular matrices, which approximate the $\mathbf{L}$ and $\mathbf{U}$ factors of $\mathbf{A}$, respectively. Applying ILU preconditioner

TABLE II
STATISTICS FOR THERMAL CIRCUITS

circuit name	dim	nnz	density
G2_circuit (UFL)	150,102	438,388	2.91
G3_circuit (UFL)	1,585,478	4,623,152	2.90
mc2rm (3D-ICE)	20,000	109,000	5.44
mc4rm (3D-ICE)	50,250	329,140	6.53
therm1	57,344	390,144	6.76
therm5	262,144	1,795,840	6.84

requires the solving of two sparse triangular systems with forward and backward substitutions, which is the bottleneck limiting the performance due to its serial nature. In our work, we use ILU0 with the fewest fill-ins. We let ILU decomposition be performed on CPU. Then, the two triangular matrices are transferred to GPU. The preconditioning with ILU0 is to apply GPU parallel triangular solves on the newly formed basis vector, which is also calculated on GPU.

We also use AINV preconditioners with tolerance dropping in [5] and [6] as our explicit preconditioner. The AINV preconditioner $\mathbf{M}$ is an approximate to $\mathbf{A}^{-1}$, and $\|\mathbf{I}-\mathbf{MA}\|$ is used to define the similarity of $\mathbf{M}$ and $\mathbf{A}^{-1}$. The approximation to $\mathbf{A}^{-1}$ is attained using three matrices [26]

$$\mathbf{A}^{-1} \approx \mathbf{M} = \mathbf{Z}\mathbf{D}^{-1}\mathbf{W}^{\mathrm{T}} \tag{4}$$

where $\mathbf{Z}$ and $\mathbf{W}$ are the two unit upper triangular matrices, $\mathbf{D}$ is a DIAG matrix. $\mathbf{Z}$ and $\mathbf{W}$ are similar to $\mathbf{U}^{-1}$ and $\mathbf{L}^{-1}$ triangular parts in LDU decomposition of $\mathbf{A} = \mathbf{LDU}$, respectively. They can be directly obtained by means of a biconjugation process [6]. If we define $\mathbf{M}_l = \mathbf{W}^{\mathrm{T}}$ and $\mathbf{M}_r = \mathbf{Z}\mathbf{D}^{-1}$, the preconditioning operations are two matrix-vector product of $\mathbf{M}_l$ and $\mathbf{M}_r$, respectively, which is easily to be parallelized. In our work, $\mathbf{M}_l$ and $\mathbf{M}_r$ are computed on CPU first, and then transferred to GPU for fast matrix-vector multiplications used in the preconditioned GPU-GMRES.

V. NUMERICAL RESULTS AND DISCUSSION

The proposed algorithm has been implemented using NVIDIA CUDA and run on Tesla C2070 GPU, which has 448 cores of 1.15 GHz and 5-GB global memory. The CPU results were tested on a quad-core Xeon E5620 machine at 2.00 GHz with 28-GB memory. For a thorough comparison, serial GMRES on CPU, parallel GMRES on CPU-GPU platform with preconditioners, and a parallel LU solver, superLU_MT. SuperLU_MT is a commonly preferred and publicly available parallel LU-based solver [10]. GPU-GMRES with AINV and ILU0 preconditioners are compared on a number of thermal and other matrices.

Our test cases consist of two thermal circuit models generated by our own finite difference thermal analysis tool based on the realistic package structures, thermal boundary conditions and materials. In addition, two circuit examples from the University of Florida matrix collection [9] and four 3-D-ICE testcases with package systems [21] are also added in our experiments. More detailed information on these thermal matrices and published matrices can be found in Table II, where dim is the matrix size, i.e., number of rows and number of columns, nnz is the number of nonzero elements in the matrix, and density is the ratio defined as nnz/dim, i.e., the average number of nonzero elements per row.

The performance of GMRES solver with preconditioners is summarized in Table III. For comparison, the same equation $\mathbf{Ax} = \mathbf{b}$ is solved as follows.

1) SuperLU_MT, i.e., multithreaded direct LU method, whose factorization routine psgstrf runs on our server with four threads in parallel.
2) GMRES solver using only serial single-thread CPU.
3) The proposed GMRES solver with GPU parallel computation. The GMRES solvers of both CPU and GPU versions take the parameters such as restart number, maximum iteration number, and residual tolerance. In our experiments, these three parameters are set as 32, 10^6, and 10^{-6}, accordingly. We remark that since the LU-based method employs four threads and can be viewed as a parallel LU-based solver, its comparison with our GPU GMRES solver is very relevant. It will show the parallelization benefits on two different multi/many core computing platforms.

TABLE III
COMPARISON OF SOLVERS ON $\mathbf{Ax} = \mathbf{b}$

1	2	3	4	5	6	7	8
circuit	SuperLU		GMRES			speedup	
name	fact.	sol.	precond	CPU	GPU	$\frac{C2}{C4+C6}$	$\frac{C2}{C6}$
G2	0.8	0.03	NON 0	8.9	2.2	0.4×	0.4×
			DIAG 0.003	0.3	0.04	19×	20×
			ILU0 0.7	0.03	0.06	1×	13×
			AINV 4.1	0.4	0.1	0.2×	8×
G3	30.7	1.9	NON 0	211	16.5	2×	2×
			DIAG 0.02	7.0	.024	697×	1023×
			ILU0 7.5	0.8	0.14	4×	219×
			AINV 42	8.4	0.68	0.7×	45×
mc2rm	0.33	0.01	ILU0 0.14	0.11	0.15	1.2×	2×
			AINV 0.18	0.14	0.15	1×	2×
mc4rm	1.63	0.05	ILU0 0.3	0.5	0.4	2×	4×
			AINV 5.0	0.5	0.2	0.3×	8×
therm1	16.0	0.15	ILU0 0.36	1.17	0.73	15×	22×
			AINV 3.02	1.18	0.63	4×	25×
therm5	554.1	1.45	ILU0 1.62	8.53	2.12	148×	261×
			AINV 11.9	17.0	3.34	36×	165×

Note that we also test the three different preconditioners, such as DIAG, ILU0, and AINV in the two GMRES implementations. The time spent on preconditioner constructions and GMRES solving with preconditioners are all recorded in Table III. As can be observed by the runtime measurements, GMRES performs better with the help of preconditioner. Especially for large matrices, GMRES with no preconditioner or with DIAG preconditioner fails to converge within the given limit of maximum iteration number. In these cases, we remove them from the table.

In general, as the problem sizes become larger, the GPU-GMRES will show more speedup over the parallel SuperLU_MT (ranging from one to two orders of magnitudes). ILU0 preconditioner usually gives better performance over AINV preconditioner. Compared with the CPU version of preconditioned GMRES solver, the GPU-GMRES solver can deliver about 2–4× speedup for large cases. Also with larger thermal circuits, we expect more speedups as indicated by the table.

We have further comments on ILU0 and AINV preconditioners. Both of them show better robustness than the simple DIAG preconditioner, and they succeed on all the demonstrated examples. Their robustness comes from the better knowledge obtained from matrix $\mathbf{A}$ during the preconditioner construction phase. The AINV preconditioner is more expensive than ILU0, since it better approximates $\mathbf{A}^{-1}$ and tends to have more nonzero elements than ILU0 preconditioner, which approximates $\mathbf{A}$. This increased cost of AINV is reflected in both the construction phase and the GMRES iteration phase, as can be observed from Table III.

It is noteworthy that we use single-precision floating-point representation, i.e., 32 bits, for real numbers in all the calculations. This implementation comes from two reasons. One is because NVIDIAs current generation GPUs support single-precision better than double precision, in terms of speed and memory space. The other reason, which is also from the perspective of our application here, is that thermal simulation results, such as temperatures, do not require very high precision. We have compared our single-precision results with double-precision ones, and the temperature difference between the two is at most 0.1°, which is acceptable to most of the IC thermal analyzes.

For even larger circuit matrices, whose data cannot be holded by the current generation GPUs memory (about 3–6 GBs), some partitioning strategies have to be employed to distribute the computing tasks and data into different GPUs on the same node or different

computing nodes (GPU clusters). In other words, distributed parallel computing techniques are required to consider such large problems, which is not the focus of this brief. Note that, traditional CPU-based computing will also face the similar issue as the memory for any CPU is always limited. Also from thermal analysis perspective, we can always trade the accuracy with thermal circuit complexities (thus simulation efficiency) using different grid sizes.

VI. Conclusion

An efficient finite difference-based full-chip simulation algorithm for 3-D-ICs with liquid cooling based on CPU-GPU platform is proposed. Unlike existing fast thermal analysis methods, the new method starts from the basic heat equations to model 3-D-ICs with intertier liquid cooling microchannels, and directly solves the resulting Partial Differential Equation using iterative GMRES solver. To speed up the analysis process, we further developed a preconditioned GPU-accelerated GMRES solver. We also studied different preconditioners for GPU platforms and compared their performances. Experimental results showed that the proposed solver can lead to orders of magnitudes speedup over the parallel LU-based solver and up to 4× speedup over CPU-GMRES for thermal circuits and other published problems.

References

[1] (2008). *IBM Interlyaer Cooling Technology for 3D Packages* [Online]. http://www.zurich.ibm.com/st/cooling/integrated.html

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

[3] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in *Proc. Conf. High Perform. Comput. Netw., Storage Anal.*, 2009, pp. 18:1–18:11.

[4] N. Bell and M. Garland. (2010). *Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations* [Online]. Available: http://cusp-library.googlecode.com

[5] 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, Sep. 1996.

[6] M. Benzi and M. Tuma, "Numerical experiments with two approximate inverse preconditioners," *BIT Numerical Math.*, vol. 38, no. 2, pp. 234–241, 1998.

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

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

[9] T. Davis. (2012). *The University of Florida Sparse Matrix Collection* [Online]. Available: http://www.cise.ufl.edu/research/sparse/

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

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

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

[13] D. B. Kirk and W.-M. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*. San Francisco, CA, USA: Morgan Kaufmann, 2010.

[14] R. Li and Y. Saad, "GPU-accelerated preconditioned iterative linear solvers," *J. Supercomput.*, vol. 63, no. 2, pp. 443–466, 2010.

[15] X. Liu, H. Wang, and S. X.-D. Tan, "Parallel power grid analysis using preconditioned gmres solvers on cpu-gpu platforms," in *Proc. ICCAD*, Nov. 2013, pp. 561–568.

[16] (2011). *NVIDIA Corporation* [Online]. Available: http://www.nvidia.com

[17] R. Patti, "3D integration: New opportunities for advanced packaging," in *Proc. EPEPS*, Oct. 2011, pp. 1–41.

[18] Y. Saad and M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Statist. Comput.*, vol. 19, pp. 856–869, Oct. 1986.

[19] Y. Saad, *Iterative Methods for Linear Systems*. Boston, MA, USA: PWS, 2000.

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

[21] A. M. Sridhar *et al.*, "3D-ICE: Fast compact transient thermal modeling for 3D-ICs with inter-tier liquid cooling," in *Proc. ICCAD*. 2010, pp. 463–470.

[22] A. M. Sridhar *et al.*, "Compact transient thermal model for 3D ICs with liquid cooling via enhanced heat transfer cavity geometries," in *Proc. 16th Int. Workshop Thermal Invest. ICs Syst.*, Oct. 2010, pp. 1–6.

[23] S. Swarup, S. X.-D. Tan, and Z. Liu, "Thermal characterization of TSV based 3D stacked ICs," in *Proc. IEEE Conf. EPEPS*, Oct. 2012, pp. 335–338.

[24] M. Wang, H. Klie, M. Parashar, and H. Sudan, "Solving sparse linear systems on NVIDIA Tesla GPUs," in *Proc. 9th Int. Conf. Comput. Sci.*, 2009, pp. 864–873.

[25] X. Wei and Y. Joshi, "Optimization study of stacked micro-chanel heat sinks for micro-electronic cooling," *IEEE Trans. Components Packag. Technol.*, vol. 26, no. 1, pp. 441–448, Mar. 2003.

[26] S. Xu, W. Xue, K. Wang, and H. Lin, "Generating approximate inverse preconditioners for sparse matrices using CUDA and GPGPU," *J. Algorithm, Comput. Technol.*, vol. 5, no. 3, pp. 475–500, 2010.