A Structured Parallel Periodic Arnoldi Shooting Algorithm for RF-PSS Analysis based on GPU Platforms*

Xue-Xin Liu†, Hao Yu‡, Jacob Relles†, and Sheldon X.-D. Tan†
† Department of Electrical Engineering, University of California, Riverside, CA 92521
‡ School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore
xliu@ee.ucr.edu, haoyu@ntu.edu.sg, jrelles@ee.ucr.edu, stan@ee.ucr.edu

ABSTRACT

The recent multi/many-core CPUs or GPUs have provided an ideal parallel computing platform to accelerate the time-consuming analysis of radio-frequency/millimeter-wave (RF/MM) integrated circuit (IC). This paper develops a structured shooting algorithm that can fully take advantage of parallelism in periodic steady state (PSS) analysis. Utilizing periodic structure of the state matrix of RF/ MM-IC simulation, a cyclic-block-structured shooting-Newton method has been parallelized and mapped onto recent GPU platforms. We first present the formulation of the parallel cyclic-block-structured shooting-Newton algorithm, called *periodic Arnoldi shooting* method. Then we will present its parallel implementation details on GPU. Results from several industrial examples show that the structured parallel shooting-Newton method on Tesla's GPU can lead to speedups of more than 20× compared to the state-of-the-art implicit GMRES methods under the same accuracy on the CPU.

1. INTRODUCTION

The recent advance in radio-frequency (RF) and millimeter-wave (MM) integrated circuits (ICs) operating at 60 GHz [1–4] can provide much higher data-rate than today's mobile devices for future smart-mobile applications. However, the design for RF/MM-IC front-end circuits at this scale is very challenging [5–11]. At the scale of 60 GHz, all active and passive devices are closely coupled and the resulting post-layout circuit model has a drastically increased complexity. Moreover, in order to follow the high-frequency carrier, traditional transient analysis requires using small time-steps and hence results in a long simulation time.

The analysis of RF/MM-IC systems is notoriously difficult for speedup as accuracy can not be compromised for a precision design. Worse, its design complexity increased drastically since all active and passive devices are no longer separated but coupled. There are three approaches to numerically find the PSS solution [5–10]: finite difference method (FDM), harmonic balance (HB) method, and shooting-Newton method. Compared to the first two approaches, shooting method has a better convergence for strongly nonlinear circuits because the underlying transient analysis has an adaptive time-step control. However, the resulting Jacobian (sensitivity matrix) during the shooting-Newton method is usually a large-scale dense matrix. The iterative GMRES with the use of a standard Krylov-subspace [12, 13] and an implicit matrix formulation [8] partially alleviate the analysis cost of shooting-Newton method.

As such, the performance and capacity required to successfully verify a full-chip design strains the limit of the existing single-core based computing approach. Fortunately, for the sake of harvesting high-performance scalability from parallelism, modern computer architecture has shifted towards designs that employ multiple processor cores on a chip, so called multi-core processors or chip-multiprocessors (CMP) [14, 15]. For example, the graphic processing unit (GPU) is one of the most powerful many-core computing systems in use. The general purpose computing has been around for decades and we have many well developed tools to improve the performance scalability of developing new scientific computing for a CPU. Only recently though, has GPU computing become a viable platform for solving large-scale computationally intensive scientific computing problems. In addition to being less of a mature platform, current GPUs have some architectural constraints that make it a little more difficult to exploit their full potential. Fortunately, tools for general-purpose GPU computing such as CUDA [16], Stream SDK [17] and OpenCL [18] allow much easier access to the computing capabilities of the modern graphics processor. A fast RF/MM-IC analysis based on the massive parallelism using GPUs is thereby urgently needed to handle the design complexity and reduce the design cycle.

To harness the power of the underlying hardware parallelism of the recent multi-core and GPU processors, an algorithmic innovation is required to develop an efficient parallel analysis. A simple parallel implementation of the matrix-vector multiplication can be definitely beneficial when it is unclear how to map the matrix-vector multiplication in shooting-Newton to the GPU hardware. In this paper, we further point out that studying problem structure is one effective way to explore parallelism. Since most RF/MM ICs are with periodic inputs and can be characterized as a periodic steady state (PSS) problem, the state matrix generally shows a *periodic-block-matrix structure*, or periodic structure, to be considered when developing algorithms for RF simulation. Without exploration of the structure of state space, the remaining part of the traditional PSS problem can not be fully parallelized. We employ the time-domain shooting-Newton method as an example to demonstrate how to improve the efficiency by identifying parallelism from the RF problem structure, i.e., periodic structure, which brings inherent parallelism from the RF problem structure but was ignored by the existing non-structured algorithms.

In this paper, we formulate the structured shooting-Newton method with the use of one periodic Arnoldi shooting (PAS) algorithm, and further map the related computational operations of PAS on the latest NVIDIA Tesla GPU platform. The new PAS algorithm can identify the periodic structured Krylov-subspace and hence can lead to highly parallel periodic structured operations. The existing study in [19, 20]

*This work is funded in part by NSF grant under No. CCF-0448534, in part by NSF grant under No. OISE-0929699. Hao Yu is supported by a grant under No. NRF2010NRF-POC001-001.

978-1-4244-7514-8/11/$26.00 ©2011 IEEE

has an initial exploration of the structured Krylov-subspace and its parallization. However, [19] has not exposed how to identify the periodic structured Krylov space during the shooting-Newton, and [20] has not discussed the improved iteration convergence and the structured parallization. The primary contribution of this paper is the development of the PAS-based shooting-Newton method on the state-of-art GPU of Tesla-C1060 card with CUDA. Experiment results show superior performance improvement when compared to the sequential non-structured shooting-Newton method such as up to 27× runtime speedup.

2. TRADITIONAL SHOOTING NEWTON

In this paper, we review the shooting-Newton-based periodic steady state (PSS) analysis, which is to be parallelized.

A nonlinear RF/MM-IC circuit can be described in general by a differential-algebra-equation (DAE) shown below,

$$\mathbf{f}(\mathbf{x}(t),t) = \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{q}(\mathbf{x}(t)) + \mathbf{j}(\mathbf{x}(t)) + \mathbf{u}(t) = \mathbf{0}, \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^N$ is the state variable vector including nodal voltages and branch currents, $\mathbf{j}(\mathbf{x}(t))$ is for nonlinear node currents, $\mathbf{q}(\mathbf{x}(t))$ is for nonlinear node charges or fluxes, and $\mathbf{u}(t)$ is for external periodic sources.

A state-transition function $\boldsymbol{\phi}_T(\mathbf{x}_0, t_0)$ is the solution of (1) at $t_0 + T$, starting from a guessed initial state $\mathbf{x}(t_0)$ at t_0, or

$$\mathbf{x}(t_0 + T) = \boldsymbol{\phi}_T(\mathbf{x}(t_0), t_0). \tag{2}$$

Then two-point boundary constraint for PSS problem is

$$\boldsymbol{\phi}_T(\mathbf{x}(0), 0) = \mathbf{x}(0) \tag{3}$$

for one period with $t_0 = 0$. A shooting-sensitivity, or called shooting-Jacobian, can be further defined by

$$J_{\boldsymbol{\phi}_T}(\mathbf{x}(0), 0) = \frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}\mathbf{x}(0)} = \frac{\mathrm{d}\boldsymbol{\phi}_T(\mathbf{x}(0), 0)}{\mathrm{d}\mathbf{x}(0)}. \tag{4}$$

Since (3) is in the nonlinear algebraic form, Newton iteration is deployed to solve for $\mathbf{x}(0)$ or denoted as $\mathbf{x}_0$ (and $\mathbf{x}(t)$ as $\mathbf{x}_T$). Such a method is the so-called shooting-Newton algorithm [5, 7, 8, 10] with the following Newton iteration

$$\mathbf{x}_0^k = \mathbf{x}_0^{k-1} + \left[\mathbf{I} - \mathbf{J}_{\boldsymbol{\phi}_T}(\mathbf{x}_0^{k-1}, 0)\right]^{-1} \left[\boldsymbol{\phi}_T(\mathbf{x}_0^{k-1}, 0) - \mathbf{x}_0^{k-1}\right] \tag{5}$$

where $\mathbf{I}$ is the identity matrix and k is the iteration number.

Specifically, $\mathbf{J}_{\boldsymbol{\phi}_T}$ is obtained by the following manner. One first integrates the DAE (1) in one period, for example, by Backward-Euler method

$$\frac{1}{h_j}\left[\mathbf{q}(\mathbf{x}_j) - \mathbf{q}(\mathbf{x}_{j-1})\right] + \mathbf{j}(\mathbf{x}_j) + \mathbf{u}_j = 0, \tag{6}$$

where h_j is the j^{th} time-step: $h_j = t_j - t_{j-1}$, with $t_0 = 0$ and $t_p = T$ for total p time-steps. As such, the linearized (6) at l^{th} transient-Newton iteration becomes

$$\left[\mathbf{G}(\mathbf{x}_j^{l-1}) + \frac{\mathbf{C}(\mathbf{x}_j^{l-1})}{h_j}\right](\mathbf{x}_j^l - \mathbf{x}_j^{l-1}) = -\frac{1}{h_j}\left(\mathbf{q}(\mathbf{x}_j^{l-1}) - \mathbf{q}(\mathbf{x}_{j-1})\right) - \mathbf{j}(\mathbf{x}_j^{l-1}) - \mathbf{u}_j \tag{7}$$

with $\mathbf{G}(\mathbf{x}_j^{l-1}) = \mathrm{d}\mathbf{j}(\mathbf{x}_j^{l-1})/\,\mathrm{d}\mathbf{x}$ and $\mathbf{C}(\mathbf{x}_j^{l-1}) = \mathrm{d}\mathbf{q}(\mathbf{x}_j^{l-1})/\,\mathrm{d}\mathbf{x}$.

Moreover, when differentiating (6) with respect to $\mathbf{x}_0$, the following relation can be obtained

$$\left[\mathbf{G}(\mathbf{x}_j) + \frac{\mathbf{C}(\mathbf{x}_j)}{h_j}\right]\frac{\mathrm{d}\mathbf{x}_j}{\mathrm{d}\mathbf{x}_0} = \frac{\mathbf{C}(\mathbf{x}_{j-1})}{h_j}\frac{\mathrm{d}\mathbf{x}_{j-1}}{\mathrm{d}\mathbf{x}_0}. \tag{8}$$

When this is applied recursively following the chain-rule for all time steps in one period, one can eventually obtain the shooting-Jacobian by

$$\mathbf{J}_{\boldsymbol{\phi}_T} = \prod_{j=1}^{p}\left[\mathbf{G}(\mathbf{x}_j) + \frac{\mathbf{C}(\mathbf{x}_j)}{h_j}\right]^{-1}\frac{\mathbf{C}(\mathbf{x}_{j-1})}{h_j}. \tag{9}$$

Note that as the transient analysis is first applied, the matrices $\mathbf{G}(\mathbf{x}_j) + \frac{\mathbf{C}(\mathbf{x}_j)}{h_j}$, $j = 1, \ldots, p$, are already available and are stored as LU-factored sparse matrices. Therefore the computational cost is mainly from the n^2 backward and forward evaluations of forming the dense $\mathbf{J}_{\boldsymbol{\phi}_T}$, and the n^3 flops of the direct LU-factorization of (5).

3. GMRES WITH NON-STRUCTURED KRYLOV SUBSPACE

3.1 Conventional Arnoldi Method in GMRES

To reduce the computation cost of solving the shooting-Newton update equation (5), the traditional non-structured Krylov-subspace based GMRES method can be applied. The GMRES method is an iterative method for solving systems of large scale linear equations $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A}$ is dense.

Algorithm 1 shows one standard Krylov-subspace based GMRES [12, 13]. The m-th order orthonormal vector basis $\mathbf{V}^m = [\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_m]$ is needed to span the m^{th} order Krylov-subspace, and hence, to solve the linear equation is equivalent to find the optimal coefficient $\mathbf{y}$ to calculate approximate solution by linear combination $\mathbf{x}^m = \mathbf{V}^m\mathbf{y}$, such that the norm $||\mathbf{b} - \mathbf{Ax}^m||_2$ attains its minimum. As such, the approximate solution $\mathbf{x}^m$ is contained in

$$\mathcal{K}^m = \mathrm{span}(\mathbf{b}, \mathbf{Ab}, \mathbf{A}^2\mathbf{b}, \ldots, \mathbf{A}^{m-1}\mathbf{b}),$$

whose orthonormal basis is $\mathbf{V}^m$.

Algorithm 1 Non-Structured GMRES with conventional Arnoldi method

Input: $\mathbf{A}$, and $\mathbf{b}$

1: Initialize: $\mathbf{r}^0 = \mathbf{b} - \mathbf{Ax}^0$ and $\mathbf{h}^0 = ||\mathbf{r}^0||_2$, $\mathbf{v}^1 = \mathbf{r}^0/\mathbf{h}^0$
2: **while** $\mathbf{h}^{i+1} > tol$ & $i < maxIter$ **do**
3: Set $\mathbf{h}^i = \mathbf{V}^{i-1}\mathbf{A}\mathbf{v}^i$
4: Set $\mathbf{w} = \mathbf{A}\mathbf{v}^i - \mathbf{V}^{i-1}\mathbf{h}^i$
5: Set $\mathbf{g}^i = ||\mathbf{w}||_2$, $\mathbf{H}^i = \begin{bmatrix} \mathbf{H}^{i-1} & \mathbf{h}^i \\ 0 & \mathbf{g}^i \end{bmatrix}$
6: Update: $\mathbf{v}^i = \mathbf{w}/\mathbf{g}^i$, $\mathbf{V}^i = [\mathbf{V}^{i-1}, \mathbf{v}^i]$
7: Minimize: $||\mathbf{h}^0 - \mathbf{H}^i\mathbf{y}^i||_2$ to find $\mathbf{y}^i$
8: $\mathbf{x}^i = \mathbf{x}^0 + \mathbf{V}^i\mathbf{y}^i$
9: **end while**

Output: $\mathbf{x} = \mathbf{x}^k$ ($\mathbf{x} \in \mathbb{R}^N$) such that $\mathbf{Ax} \approx \mathbf{b}$

Moreover, to generate the Krylov-subspace in GMRES, the Arnoldi algorithm [12, 13] can be used to form the orthonormal basis $\mathbf{V}^m$ of the Krylov-subspace. Each Arnoldi iteration generates a new vector and is then appended to the previous Krylov-subspace basis $\mathbf{V}^{m-1}$ to form a new orthonormal basis $\mathbf{V}^m$. The Arnoldi algorithm also creates an upper Hessenburg matrix $\mathbf{H}_m$ of size $(m+1)$-by-m, which is used to check the solution at the current iteration. However, as discussed later in this paper, this standard GMRES procedure ignores the periodic structure of matrix for the PSS RF problem, and hence might result in more iterations and low parallel scalability.

3.2 Matrix-free in GMRES

For the PSS shooting-Newton method, the Krylov-subspace based GMRES can be employed to calculate the state variable $\mathbf{x}$, when the state matrix $\mathbf{A}$ and input $\mathbf{b}$ are given by $\mathbf{A} = \mathbf{I} - \mathbf{J}_{\phi_T}$, and $\mathbf{b} = \phi_T(\mathbf{x}_0^{k-1}, 0) - \mathbf{x}_0^{k-1}$.

GMRES can be embedded into the shooting-Newton algorithm. However, explicitly forming the matrix $\mathbf{A}$ involves n^2 evaluations and n^2 multiplications for $\mathbf{A}\mathbf{v}_k$ for each GMRES iteration. To further reduce the computational cost, one needs to avoid the explicit formulation of matrix $\mathbf{A}$. The work in [8,10] introduced a matrix-free approach to directly calculating the matrix-vector-multiplication (MVP) $\mathbf{A}\mathbf{v}_k$ without forming $\mathbf{A}$.

This is realized in Algorithm 2. The matrix-free method calculates the product of $\mathbf{J}_{\phi_T}\mathbf{v}^k$ instead of $\mathbf{A}\mathbf{v}^k$ by iteratively reusing the pre-factored chain-rule values of each

$$\left(\mathbf{G}(\mathbf{x}_j) + \frac{\mathbf{C}(\mathbf{x}_j)}{h_j}\right), \quad j = 1, \ldots, p. \tag{10}$$

Here, the shift of $\mathbf{I}$ to form $\mathbf{A}$ from $\mathbf{J}_{\phi_T}$ can be easily corrected when updating $\mathbf{v}^k$.

Algorithm 2 Matrix-free to replace Line 4 in Algorithm 1

Input: Previous basis $\mathbf{v}^k$ and prefactorized matrices (10)
1: $\mathbf{u} = \mathbf{v}^k$
2: **for** $j = 1 : p$ **do**
3: $\quad \mathbf{u} = \left(\mathbf{G}(\mathbf{x}_j) + \frac{1}{h_j}\mathbf{C}(x_j)\right)^{-1} \frac{1}{h_j}\mathbf{C}(\mathbf{x}_j)\mathbf{w}$
4: **end for**
5: $\mathbf{u} = \mathbf{v}^k - \mathbf{u}$
Output: $\mathbf{u} = \mathbf{A}\mathbf{v}^k = (\mathbf{I} - \mathbf{J}_{\phi_T})\mathbf{v}^k$

4. GMRES WITH PERIODIC STRUCTURED KRYLOV SUBSPACE

We observe that the Krylov-subspace of a periodic system usually contains a periodic-block-matrix structure (or periodic structure). We show that if one can identify the periodic structure of the Krylov-Subspace, the convergence and efficiency of the GMRES can be improved. As discussed later, the proposed periodic Arnoldi shooting method, *PAS*, can further maximize the parallelism to support the shooting-Newton method on GPUs.

4.1 Periodic Structured Krylov Subspace

Note that the RF/MM-IC system with periodic inputs is in fact a periodic system. Hence the associated Krylov-subspace would also exhibit a periodic structure. As such, it inspires us to exploit the structure-preserved Krylov-subspace method during the GMRES iteration.

Let's first identify the structure of the underlying Krylov-subspace. For $j = 1, \ldots, p$ time-step, defining

$$\mathbf{G}_j = \mathbf{G}(x_j),\ \mathbf{C}_j = \mathbf{C}(x_j),\ \mathbf{A}_j = \left[\mathbf{G}_j + \frac{\mathbf{C}_j}{h_j}\right]^{-1} \frac{\mathbf{C}_{j-1}}{h_j}, \tag{11}$$

the shooting-Jacobian $\mathbf{J}_{\phi_T}$ in (9) becomes

$$\mathbf{J}_{\phi_T} = \mathbf{A}_p \times \mathbf{A}_{p-1} \times \cdots \times \mathbf{A}_1. \tag{12}$$

As shown in [20], the multiplied product J_{ϕ_T} has an invariant subspace identical to the periodic block-structured matrix in 13, which has a periodic structured Krylov-subspace.

$$\mathcal{J} = \begin{bmatrix} 0 & & & \mathbf{A}_p \\ \mathbf{A}_1 & \ddots & & \\ & \ddots & & \\ & & \mathbf{A}_{p-1} & 0 \end{bmatrix} \tag{13}$$

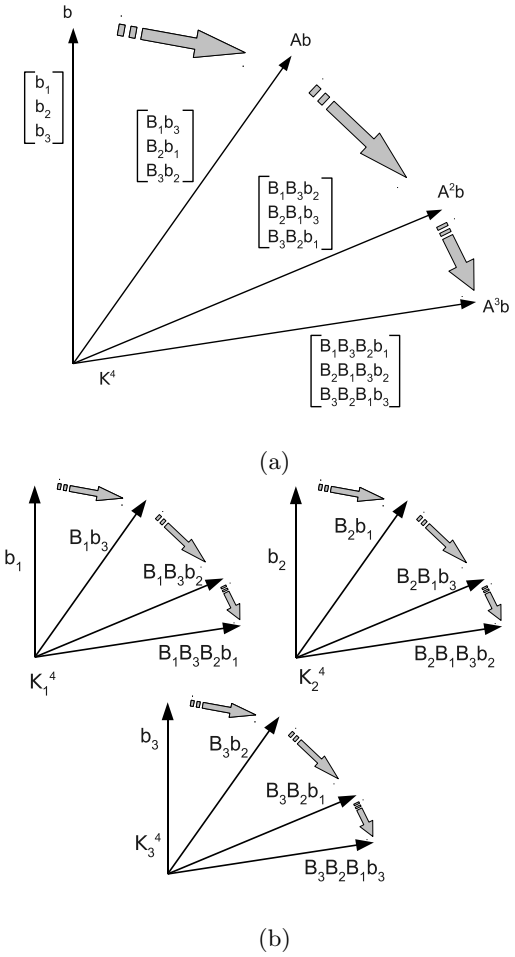

Figure 1: The comparison of standard Krylov-subspace and p-periodic-block-structured Krylov-subspace. Order $m = 4$, and $p = 3$ time-steps.

Accordingly, as shown in Algorithm 3, one can develop a a periodic Arnoldi method [20] based on the periodic Krylov-subspace, i.e., the block matrices $\mathbf{V}_j^m$ and $\mathbf{H}_j^m$, $j = 1, \ldots, p$. Here, the subscript j denotes the index of the periodic blocks ($j = 1, \ldots, p$), and the superscript i denotes the index of the order of the Krylov-subspace ($i = 1, \ldots, m$). Different from Algorithm 1, the periodic structure of the generated Krylov-subspace is preserved in Algorithm 3 because each orthonormalized base $\mathbf{v}_j^i$ is constructed separately for each $\mathbf{A}_j$. In contrast, the base of the Krylov-subspace in Algorithm 1 is generated for the overall $\mathbf{J}_{\phi_T} = \mathbf{A}_1 \times \cdots \times \mathbf{A}_p$ in Algorithm 1. As a result, the periodic structure of the underlying Krylov-subspace is destroyed in Algorithm 1.

4.2 Convergence Improvement

Since the periodic Arnoldi method can identify the Krylov-subspace with the consideration of the periodic structure, in the following, we further explain that a better convergence can be observed in comparison with the non-structured Krylov-subspace based method. The primary observation in Section 4.1 is to illustrate that the underlying Krylov-subspace of shooting-Newton implicitly has the periodic structure as shown in 13). The subspace at each time-step are calculated from each those $\mathbf{A}_i$ and right-hand-side $\mathbf{b}_i$ vectors ($i = 1, \ldots, p$), which preserves the periodic structure in (13).

Take $m = 4$ and $p = 3$ for example, in the traditional

Algorithm 3 A matrix-free periodic Arnoldi method

1: Inputs: $\mathbf{A}_j$ by pre-factorized matrices (10) ($j = 1, \ldots, p$)
2: Initialize $\mathbf{V}_1^0$ by $\mathbf{v}^0$ and $\mathbf{V}_0^i$ by 0
3: **while** $\mathbf{h}^{i+1} > tol$ & $i < maxIter$ **do**
4: **for** $j = 1 : p$ **do**
5: Set $\mathbf{h}_j^i = \mathbf{V}_{j+1}^{i-1}\mathbf{A}_j\mathbf{v}_i^j$,
6: Set $\mathbf{w} = \mathbf{A}_j\mathbf{v}_i^j - \mathbf{V}_{j+1}^{i-1}\mathbf{h}_j^i$
7: Set $\mathbf{g}_j^i = ||\mathbf{w}||_2$, $\mathbf{H}_j^i = \begin{bmatrix} \mathbf{H}_j^{i-1} & \mathbf{h}_j^i \\ 0 & \mathbf{g}_j^i \end{bmatrix}$
8: $\mathbf{v}_{j+1}^i = \mathbf{w}/\mathbf{g}_j^i$, $\mathbf{V}_{j+1}^i = [\mathbf{V}_{j+1}^{i-1}, \mathbf{v}_{j+1}^i]$
9: **end for**
10: **end while**
11: Outputs: Block matrices $\mathbf{V}_j^m$ and $\mathbf{H}_j^m$ ($j = 1, \ldots, p$)

GMRES, the solution is

$$\mathbf{x}^4 \in \text{span}(\begin{bmatrix}\mathbf{b}_1\\ \mathbf{b}_2\\ \mathbf{b}_3\end{bmatrix}, \begin{bmatrix}\mathbf{A}_1\mathbf{b}_3\\ \mathbf{A}_2\mathbf{b}_1\\ \mathbf{A}_3\mathbf{b}_2\end{bmatrix}, \begin{bmatrix}\mathbf{A}_1\mathbf{A}_3\mathbf{b}_2\\ \mathbf{A}_2\mathbf{A}_1\mathbf{b}_3\\ \mathbf{A}_3\mathbf{A}_2\mathbf{b}_1\end{bmatrix}, \begin{bmatrix}\mathbf{A}_1\mathbf{A}_3\mathbf{A}_2\mathbf{b}_1\\ \mathbf{A}_2\mathbf{A}_1\mathbf{A}_3\mathbf{b}_2\\ \mathbf{A}_3\mathbf{A}_2\mathbf{A}_1\mathbf{b}_3\end{bmatrix})$$

while the p-periodic-block-structured method has the solution

$$\mathbf{x}^4 = \begin{bmatrix}\mathbf{x}_1^4\\ \mathbf{x}_2^4\\ \mathbf{x}_3^4\end{bmatrix} \begin{array}{l} \mathbf{x}_1^4 \in \mathcal{K}_1^4 = \text{span}(\mathbf{b}_1, \mathbf{A}_1\mathbf{b}_3, \mathbf{A}_1\mathbf{A}_3\mathbf{b}_2, \mathbf{A}_1\mathbf{A}_3\mathbf{A}_2\mathbf{b}_1) \\ \mathbf{x}_2^4 \in \mathcal{K}_2^4 = \text{span}(\mathbf{b}_2, \mathbf{A}_2\mathbf{b}_1, \mathbf{A}_2\mathbf{A}_1\mathbf{b}_3, \mathbf{A}_2\mathbf{A}_1\mathbf{A}_3\mathbf{b}_2) \\ \mathbf{x}_3^4 \in \mathcal{K}_3^4 = \text{span}(\mathbf{b}_3, \mathbf{A}_3\mathbf{b}_2, \mathbf{A}_3\mathbf{A}_2\mathbf{b}_1, \mathbf{A}_3\mathbf{A}_2\mathbf{A}_1\mathbf{b}_3) \end{array}$$

Their differences can be clearly illustrated by Figure 1. Due to the structure-preservation, the periodic Arnoldi shooting (PAS) method with (p-periodic-block-matrix structure) shows higher freedom to search the approximate solution from a p times larger dimension of the subspace than the traditional one. Therefore, PAS can significantly reduce the iteration number under the same accuracy, which in turn has a reduced runtime as observed by the experiments. In the following part, we further show that such a structured subspace can also lead to a number of structured operations, which can easily mapped on the parallel GPU platform.

4.3 PAS Parallelization on GPU

In this part, we further discuss how to have a highly parallel implementation of PAS-based shooting-Newton method on GPU.

We first describe the processor and memory architectures of recent Nvidia Tesla GPU (T10 GPU). One T10 is a streaming multiprocessor(SM) and has 8 streaming processors (SPs or cores) and 2 special function units (SFUs). It has 16 KB R/W shared memories. It can has 1024 threads (128 threads/SP) and 16K registers. Threads are grouped by thread block and up to 512 threads are allowed per thread block. A SM can hold up to 8 thread blocks. A GPU grid (one T10) consists of many blocks with different kinds of memories such as local memory (local to each thread), shared memory (shared by threads in a block), global memory, constant memory and texture memory shared by threads by all the threads in a GPU grid. Those memories have different latencies. In T10 GPU, treads within a thread block are grouped into *warps* of 32 parallel threads, which is the scheduling unit of coalescing memory access.

Note that tremendous amount of matrix vector multiplications (MVMs) are employed in GMRES. Given sufficient memory, it allows us to keep our values of matrix $\mathbf{A}$ as well as residual vector $\mathbf{v}^i$ and in GPU memory without constantly swapping them in and out of memory for each iteration. To implement such a common MVM operation on GPU, we notice that memory bandwidth is a critical part of performance bottleneck and must be considered when programming applications for GPU computing. One of the key aspects to making the MVM in GMRES performed well is to limit the number of memory transactions, which are required without compromising the amount of work to be offloaded to the GPU. Improving the efficiency of memory transactions within one GPU, the computing application could observe the performance difference between 10% and 50% of the theoretical computing limit of one GPU.

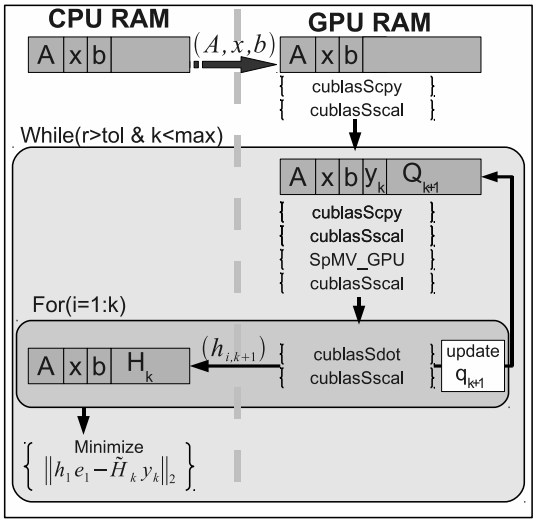

Figure 2: Diagram showing memory transfers and CUDA usage in GMRES implementation.

A diagram on how memory transfers are used efficiently for our GPU-based GMRES implementation is shown in Fig. 2. Once the initial transfer of the matrix $\mathbf{A}$ and vectors $\mathbf{x}$ and $\mathbf{b}$, the only time a memory transfer is made during the main loop of the GMRES implementation is when building the Hessenberg matrix during the orthonormalization process. During this step, the $k+1$ column of $\tilde{H}$ is transferred to the CPU from which a Least-Square minimization is performed to see if the desired tolerance of our residual has been met. As can be seen in Fig. 2, all matrix-vector, vector-vector, and scalar-vector operations are done without any unnecessary memory transfers. Other than the MVM operation, all of these operations can be done with the well optimized CUDA implementation of Basic Linear Algebra Subprograms (CUBLAS).

More importantly, because of the independent calculation of the new basis vector inside each subspace for different time-step j ($j = 1...p$) in Algorithm 3, the structure-preserved Arnoldi iteration can be highly implemented on GPU in parallel. For the initialization part, the memory for basis vectors and Hessenberg matrices are allocated. While it is a wise strategy to keep all basis vectors $\mathbf{V}_i^{m+1}$ in GPU memory, since Hessengberg matrices are needed in solving the the least squares problem. In order to calculate the final solution, because of the serial nature of the least squares problem, $\mathbf{H}_i^m$s are saved in host memory instead of GPU memory. There is another noteworthy detail about the application of MVM to the calculation of new basis vector. Same as the matrix-free GMRES method, we also use the pre-factorized LU matrices from the linearization process of device models, since these matrices are required by SPICE on each time-step. So once they are available, we save them into compressed row major form (CSR) sparse matrix. Hence, it accelerates periodic structured GMRES when using GPU-MVM to generate the new vector.

Table 1: Initial GMRES GPU vs. CPU test

Name	N(dim)	density	Time (ms) CPU	Time (ms) GPU	Speedup
MNA	10,000	4.88%	436.8	140	23.8
Cage11	39,082	0.04%	216.3	56	3.8
laminar	67,173	0.09%	15921	2030	7.84
Torso2	115,967	0.007%	700.8	123	5.66

5. NUMERICAL EXPERIMENTS

The proposed periodic Arnoldi shooting is implemented in C++ within SPICE on GPU and is called GPU-PAS-GMRES. The GPU cards we use for all the experiments here is NVIDIA's Tesla C1060, which contains 30 multiprocessors (totally 240 cores) and works in a 1.30 GHz clock rate with 4GB on-chip memory. For the CPU-server,an Intel core-duo server with 2.4GHZ CPU and 4GB memory is used.

The matrix-free GMRES with the non-structured Krylov-subspace, are implemented for the comparison. The non-structured Krylov-subspace is identified through the least-minimum-square constraint using a template from [21]. The matrix-free GMRES is implemented exactly following the procedure described in [8, 10], and is called CPU-GMRES and GPU-GMRES in the following. The GMRES iteration tolerance is set as 10^{-6} for voltage nodes. We compare the accuracy and runtime with a scalability study using 5 industrial analog/RF examples, including a CMOS-dc-converter, a CMOS-frequency-multiplier, a BJT-mixer, a CMOS-low-noise-amplifier (LNA), and a CMOS-switch-cap. We increase the complexity by adding extracted parasitics.

5.1 GPU-based Matrix-Vector-Multiply

Since the traditional GMRES spends really a long time to generate the sensitivity matrix, and use that dense matrix to calculate new basis vector each time in the Arnoldi iteration, this method is not suitable for large scale RF circuit simulation. On the contrary, the most up-to-date matrix-free method utilizes the prefactorized sparse LU matrices inside the Arnoldi iteration for matrix vector multiplication. Thus there is a evident speedup compared to explicit GMRES.

When the GPU parallelization is considered, we first show the results of the GPU-based GMRES solver vs. the CPU-based GMRES solver, mainly to demonstrate the performance matrix-vector-multiply (MVM). The performance if our current implementation on the GPU achieves the convergence in a similar number of iterations to that of a pure CPU implementation. The benchmark matrices used in the experiment here come from the University of Florida Sparse Matrix Collection [22]. As shown in Table 1, using a non-optimized version of the MVM for the GPU, we obtained 4× speedup on average over the CPU-based GMRES. With further optimization of GPU and CPU communication discussed in Section 4.3, the GPU-based GMRES is capable of 15× speedup on average over the CPU-based GMRES on the same matrices.

5.2 GPU-based PAS-GMRES on GPU

We first show details of one examples with waveform accuracy comparison. The example is a CMOS switch-cap with 654 states and a carrier input frequency of 1GHz. Fig. 3 shows two agreed periodic-steady-states at two nodes. For this example, the GPU-PAS-GMRES method converges in 3 iterations using 0.04s and the CPU-GMRES method converges in 11 iterations using 52.5s.

In the following, we further demonstrate the speedup of

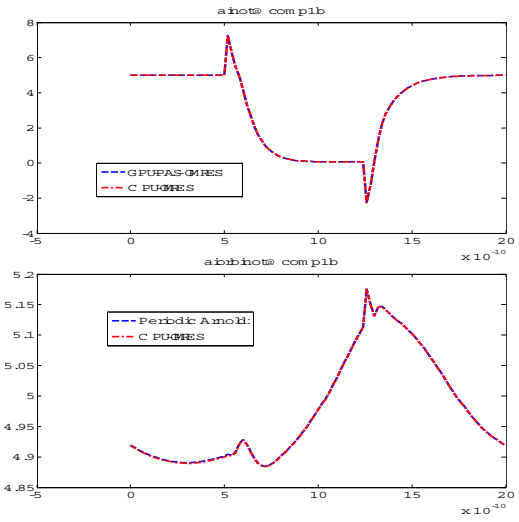

Figure 3: The PSS waveform accuracy comparison at two nodes of a cmos-switch-cap. The red-line is the non-structured CPU-GMRES, and the blue-line is the structured GPU-PAS-GMRES.

the GPU-parallization of the PAS-GMRES. When the GPU parallelization is further applied to the PAS-GMRES solver, we can save the computation time further, and the speedup of parallel GPU-PAS-GMRES over the GPU-GMRES (matrix-free) is up to 27× for these examples. As such, the new parallel GPU-PAS-GMRES solver is more scalable and will yield more speedups when the larger post-layout RF circuits are available.

Table 2: Run time comparison of different GMRES method

Ckt	#Eq	Time (s) CPU	Time (s) GPU	Time (s) GPU-PAS
dc-converter	481	35.8	1.13	0.10
BJT-mixer	504	51.1	2.69	0.56
switch-cap	654	52.5	0.30	0.04
freq-mult	649	139	5.42	0.81
LNA	1055	1238	27.2	1.04

6. CONCLUSION

In this paper, we have developed a structured shooting algorithm that can fully exploit parallelism in periodic steady state (PSS) analysis. The proposed method utilizes the periodic structure of the state matrix of RF/MM-IC, and the periodic Arnoldi based shooting-Newton method, called PAS. PAS leads to the robust parallel shooting-Newton algorithm with improved convergence and parallelism. Due to the structure-preservation, PAS is further mapped on a GPU platform with additional speedup observed. We have showed the implementation detail such as the optimization of memory allocation between CPU and GPU for this purpose. Experimental results from several industrial examples show that when compared to the state-of-the-art implicit GMRES methods under the same accuracy, the GPU-PAS based shooting-Newton method can result in up to 27× speedup.

7. REFERENCES

[1] C. H. Doan, S. Emami, D. A. Sobel, A. M. Niknejad, and R. W. Brodersen. Design considerations for 60

GHz CMOS radios. *IEEE Commun. Mag.*, pages 132–140, 2004.

[2] T. C. Chen. Where CMOS is going: trendy hype vs. real technology. In *International Solid-State Circuits Conference*, pages 22–27, Feb. 2006.

[3] A. Hajimiri. Holistic design in mm-wave silicon ICs. *IEICE Trans. on Electronics*, pages 817–828, 2008.

[4] B. Razavi. Design of millimeter-wave CMOS radios: A tutorial. *IEEE Trans. Circuits Syst. I*, pages 4–16, 2009.

[5] T. J. Aprille and T. N. Trick. Steady-state analysis of nonlinear circuits with periodic inputs. *IEEE Proc.*, pages 108–114, 1972.

[6] S. Skelboe. Computation of the periodic steady-state response of nonlinear networks by extrapolation methods. *IEEE Trans. on Circuits and Systems*, pages 161–175, 1980.

[7] K. S. Kundert, J. K. White, and A. Sangiovanni-Vincentelli. *Steady-State Methods for Simulating Analog and Microwave Circuits*. Kluwer Academic Publishers, 1990.

[8] R. Telichevesky, K. Kundert, and J. White. Efficient steady-state analysis based on matrix-free Krylov-subspace methods. In *Proc. Design Automation Conf. (DAC)*, 1995.

[9] K. Mayaram, D. C. Lee, S. Moinian, D. Rich, and J. Roychowdhury. Computer-aided circuit analysis tools for RFIC simulation: algorithms, features, and limitations. *IEEE Trans. on Circuits and Systems-II*, pages 274–286, 2000.

[10] O. Nastov, R. Telichevesky, K. Kundert, and J. White. Fundamentals of fast simulation algorithms for RF circuits. *IEEE Proc.*, pages 600–621, 2007.

[11] H. Chang and K. Kundert. Verification of complex analog and RF IC designs. *IEEE Proc.*, pages 622–639, 2007.

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

[13] G. H. Golub and C. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.

[14] Intel multi-core processors, making the move to quad-core and beyond (White Paper), 2006. `http://www.intel.com/multi-core`.

[15] Multi-core processors - the next evolution in computing (White Paper), 2006. `http://multicore.amd.com`.

[16] CUDA (compute unified device architecture). `http://www.nvidia.com/object/cuda_home.html`.

[17] AMD Steam SDK. `http://developer.amd.com/gpu/ATIStreamSDK`.

[18] Open computing language (OpenCL). `http://www.khronos.org/opencl`.

[19] Wim Bomhof. *Iterative and parallel methods for linear systems, with applications in circuit simulation*. PhD thesis, Mathematical Institute, Utrecht University, 2001.

[20] X.-X Liu, H. Yu, and S. X.-D. Tan. A robust periodic arnoldi shooting algorithm for efficient large-scale rf/mmic simulation. In *Proc. Design Automation Conf. (DAC)*, pages 573–578, Jun. 2010.

[21] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[22] Tim Davis. The university of florida sparse matrix collection. `http://www.cise.ufl.edu/research/sparse/`.