Transient Analysis of Large Linear Dynamic Networks on Hybrid GPU-Multicore Platforms

Xue-Xin Liu*, Sheldon X.-D. Tan*, Zao Liu*, Hai Wang†*, and Tailong Xu‡
* Dept. Electrical Engineering, University of California, Riverside, CA 92521
† School of Microelectronics and Solid State Electronics, UESTC, Chendu, China, 610054
‡ School of Electronics and Information Engineering, Anhui University, Hefei, China

***Abstract*—A new transient analysis method is proposed for general linear dynamic networks, such as on-chip power grid networks, using hybrid GPU-based multicore platform. The new method, called *ETBR-GPU*, first performs sampling-like reduction on the original circuit matrices where the frequency domain responses at different frequency points can be calculated in parallel on multicore CPU. After the reduction, the reduced circuit matrices, which are dense but well suitable for GPU's data parallel computing, are simulated on GPU. Such reduction based simulation technique is very amenable for parallelization on the hybrid multicore and GPU platforms, where coarse-grained task-level and fine-grained lightweight-thread level parallelism can be both exploited. The proposed method is very general, since it can analyze any linear networks with complicated structures and macromodels, and it does not assume some structure properties in order to build problem-specific preconditioners, as many iterative solvers do. Experiments show that the new method achieves about one or two orders of magnitude speedup when compared to the general LU-based simulation method on some recently published IBM power grid benchmark circuits.**

I. Introduction

The verification of today's large linear global networks such as on-chip large power grid networks remains challenging for chip designers. Fast verification of voltage drops and other noises on power delivery networks is critical for final design closure. The so-called power integrity verification becomes even more challenging due to increasing design complexity and lowered supply voltage as VLSI technologies advance [1].

Intensive studies have been carried out to seek for efficient analysis of large power grid networks in the past decade. Various algorithms have been proposed to improve scalability in computing time and to reduce memory footprints [2], [3], [4], [5]. But most of those techniques are based on the homogeneous single-core architectures. Since the introduction of multicore architectures, hardware designs are going through a renaissance. The leap from single-core to multi-core or many-core technologies has permanently altered the course of computing. Among them, the graphics processing units, GPUs, are one of the most powerful many-core computing systems in mass market use.

In this article, we propose a new transient analysis method for general linear dynamic networks based on hybrid GPU-based multicore platforms, which are the popular parallel computing system. We use on-chip power grid networks as the driving problem to illustrate the effectiveness of the proposed method. The new algorithm, called *ETBR-GPU*, first reduces the circuit complexity in frequency domain before the transient simulation of the reduced circuits. Direct solvers, like LU factorization, are used so that it can solve for any linear networks without dependency on problem-specific preconditioners as required by most of the iterative solvers. As a result, the parallelism and data independency in the ETBR-GPU are exploited in a different way.

We show that ETBR-GPU has coarse-grained task-level and fine-grained light-weight thread level parallelisms, each of them can be exploited by exploring the strengths of the GPU and CPU architectures. ETBR-GPU first performs sampling-like reduction on the original circuit matrices where the frequency domain responses at different frequency points have been calculated in parallel on multicore CPU platforms at the task level. After the reduction, the reduced circuit matrices, which are dense but well suitable for GPU platform computing, are simulated on GPU platforms at fine-grained thread level. ETBR-GPU especially favors transient simulation with long simulation periods as frequency-domain reduction related costs is almost independent of simulation time steps. Experimental results show that the new method can achieve about one or two orders of magnitude speedup when compared to the general LU-based simulation method on some recently published IBM power grid benchmark circuits [6].

This research was supported in part by NSF grants under No. CCF-1017090, No. OISE-1130402, and No. OISE-1051797.

II. Background

A. The problem of power grid simulation

In this section, we review the problem of power grid analysis. We remark that our proposed method is not limited to solve this category of problems, but can be applied to any linear dynamic networks.

An on-chip power grid network can be modeled as RC or RLC networks with known time-variant current sources, which can be obtained by gate-level logic simulations of the circuits. A typical power grid model can contain several million nodes, and up to hundreds of thousands of input current sources. There are some nodes with known voltages in the grid, and modeled as nodes connected with constant voltage sources.

Given the current source vector, $\mathbf{u}(t)$, the node voltages can be obtained by solving the differential equations formulated using modified nodal analysis (MNA),

$$\mathbf{G}\mathbf{x}(t) + \mathbf{C}\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{B}\mathbf{u}(t), \tag{1}$$

where $\mathbf{G} \in \mathbb{R}^{n\times n}$ is the conductance matrix, $\mathbf{C} \in \mathbb{R}^{n\times n}$ is the matrix resulting from charge storage elements, $\mathbf{B} \in \mathbb{R}^{n\times m}$ is the input selector matrix, $\mathbf{x}(t) \in \mathbb{R}^n$ is the vector of time-varying node voltages and branch currents of inductors and

978-1-4673-0859-5/12/$31.00 ©2012 IEEE

Algorithm 1 Extended truncated balanced realization (ETBR) method

Input: MNA matrices of the linear system, $\mathbf{G}$, $\mathbf{C}$, $\mathbf{B}$, input source $\mathbf{u}(t)$, and reduced order (number of samples), q

Output: Reduced system matrices $\hat{\mathbf{G}}$, $\hat{\mathbf{C}}$, $\hat{\mathbf{B}}$

1: Convert all the input signals $\mathbf{u}(t)$ into $\mathbf{u}(s)$ using FFT.
2: Select q frequency points $s_1, s_2, \ldots, s_q$.
3: Compute $\mathbf{z}_k = (s_k\mathbf{C} + \mathbf{G})^{-1}\mathbf{B}\mathbf{u}(s_k)$. *// Multithread*
4: Form the matrix $\mathbf{Z} = [\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_q]$.
5: Perform SVD on $\mathbf{Z}$, such that $\mathbf{Z} = \mathbf{V}\mathbf{S}\mathbf{U}^{\mathrm{T}}$.
6: $\hat{\mathbf{G}} = \mathbf{V}^{\mathrm{T}}\mathbf{G}\mathbf{V}$, $\hat{\mathbf{C}} = \mathbf{V}^{\mathrm{T}}\mathbf{C}\mathbf{V}$, and $\hat{\mathbf{B}} = \mathbf{V}^{\mathrm{T}}\mathbf{B}$,

voltage sources, and $\mathbf{u}(t) \in \mathbb{R}^m$ is the vector of independent power sources.

Suppose the backward Euler method is applied to the integration of this dynamical system, the transient behavior of this power grid can be solved step by step from a given initial condition $\mathbf{x}(0)$ using

$$(\mathbf{G} + \frac{1}{h}\mathbf{C})\mathbf{x}(t+h) = \frac{1}{h}\mathbf{C}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t+h), \qquad (2)$$

where h is the time step length. If a fixed time step h is chosen, then the left-hand side matrix, $\mathbf{G} + \frac{1}{h}\mathbf{C}$, will remain the same along all time steps. Hence, in this case, its LU factors can be factorized only once and reused for the triangular solves in the following time steps.

B. Review of reduction-based simulation methods

In this project, we use ETBR method [5], because it is more amenable for parallel computing as each frequency domain response can be computed independently. Algorithm 1 summarizes the flow of ETBR, which reduces the original system in Eq. (1) into a reduced system of $\hat{\mathbf{G}}$, $\hat{\mathbf{C}}$, and $\hat{\mathbf{B}}$ can be simulated in time domain more efficiently. To retrieve the original waveforms, a left multiplication of the basis matrix $\mathbf{V}$ on the reduced circuit state $\hat{\mathbf{x}}(t)$ is required, i.e., $\mathbf{x}(t) = \mathbf{V}\hat{\mathbf{x}}(t)$. The stability and passivity properties of ETBR can be promised if proper operations are applied during the reduction [5].

We observe that for the ETBR algorithm, computing $\mathbf{z}_k$, shown in Line 3, can be easily parallelized as each $\mathbf{z}_k$ can solved independently. Solving $\mathbf{z}_k$ using direct sparse solvers however is challenging to parallelization in GPU as memory access of sparse matrices are difficult to optimized. As a result, $\mathbf{z}_k$, will be computed on mutlicore CPU platforms. After the reduction, the reduced system matrices are dense. The simulation of dense matrices is difficult for CPU, but is very suitable for GPU computing, as dense matrices based LU factorization and triangular factor solvers on GPU-platforms have been well developed in CUBLAS (for triangular matrices solvers) [7] and in MAGMA library [8].

III. The Proposed ETBR-GPU method

In this section, we first provides the flow of our GPU-accelerated ETBR reduced model based power grid analysis—ETBR-GPU, and discuss our GPU programming details during implementation.

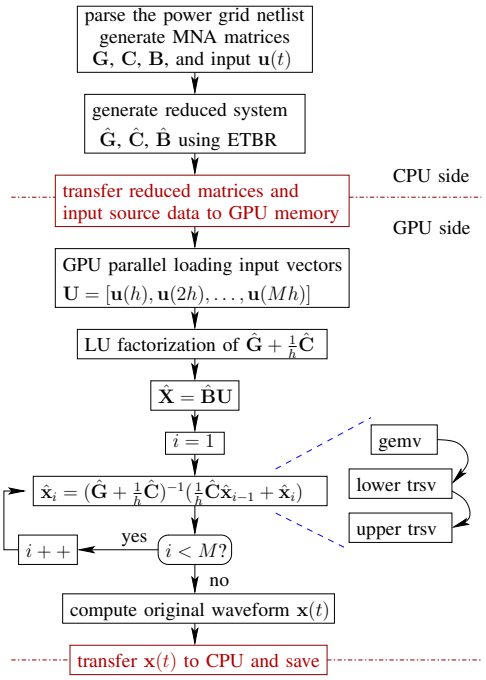

Fig. 1. The flow of GPU accelerated ETBR based power grid analysis.

A. The overall algorithm flow

Fig. 1 gives the overall flow of our new method. The whole algorithm has two main parts, the CPU part (host) and GPU part (device) as clearly marked in the figure. CPU part mainly reads and parses the netlist, generate the reduced system by ETBR, which are sequential codes and difficult to parallelize, while GPU part takes care of the input source evaluation and LU triangular solves of transient integration equations, which are the most time-consuming parts. Once the reduced system matrices are ready, they are transferred to GPU global memory for transient simulation. Meanwhile, the input stimulus information is also transferred to GPU memory so the input source interpolation and evaluation can be run in parallel. This is important as the input source interpolation takes significant amount of time to compute. But fortunately the right-hand side input vectors $\mathbf{u}(t)$ can be computed independently at different time t although the transient time steps have to be calculated one by one in a sequential way. As we will show below, we can compute all vectors of $\mathbf{u}(t)$ very efficiently by harvesting the massive data parallel power of GPU. Also in GPU side, backward Euler equation's left-hand side matrix is formed and factorized in to LU factors, which are saved for repeated use in all time steps.

B. Parallel input source interpolation on GPUs

Since the elements of $\mathbf{u}(t)$ contributed from different sources and at different time steps are independently calculated, this task can be easily accelerated by parallel GPU computing. In our proposed simulator, a parallel GPU kernel will launch a multiple of thread blocks to evaluate the sources. Each thread block is responsible for interpolating one specific source, and the threads inside calculates the source's values at a group of, say, 256, time steps. Fig. 2 illustrates the thread

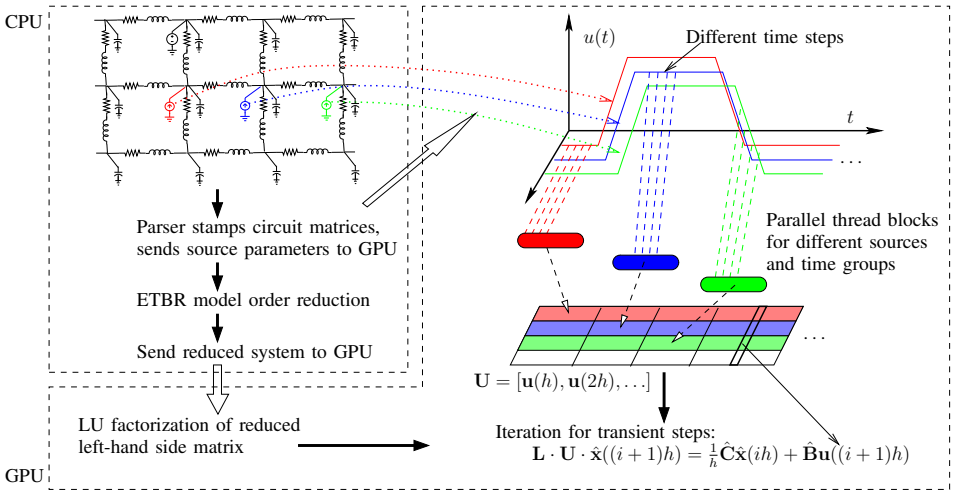

Fig. 2. ETBR-GPU flow and GPU-accelerated Input source interpolation.

Algorithm 2 GPU Parallel evaluation of input sources for transient power grid analysis

1: Configure the kernel grid and block dimensions according to number of sources and time steps.
2: **for** all sources **do** *// launch threads in grids*
3: Specify the block index for corresponding input source.
4: Read PWL or pulse waveform parameters into shared memory.
5: **for** all time steps **do** *// threads in each block calculate values for the same source*
6: Calculate the time t handled by each thread.
7: Use t to judge the location in a PWL or pulse.
8: Linear interpolate PWL or pulse to get source value.
9: Save value element to $\mathbf{u}(t)$.
10: **end for**
11: **end for**

organization for parallel source interpolation. Note that there can be many concurrent blocks working on the same source, but at different time groups.

For GPU computing, the main challenge is to allow fast memory access by threads or reduce memory traffic by reuse via shared memory (or texture memory) within blocks. In this way, GPU cores can be busy all the time without waiting for memory access. In GPU, fast global memory access by threads can be done by coalesced memory access, where a half warp (or a warp) of threads (16 or 32 threads respectively) can read their data from the global memory in one read access. Coalesced memory access requires that data are arranged continuously in memory and consecutive with respective to involved thread indexes. As a result, for the parallel source evaluation, we need to align the values of one source at all time steps in one row. Hence, once all N_{cur} sources and all M time steps are evaluated, we obtain a matrix $\mathbf{U}$ of dimension N_{cur}-by-M. In this way, the i-th column of $\mathbf{U}$ is exactly the input vector $\mathbf{u}(t)$ when $t = i \cdot h$, and can be used in computing the right-hand side of transient simulation. This process of memory allocation and parallel interpolation of sources is demonstrated in Fig. 2.

C. Solving transient steps on GPU

ETBR is based on model order reduction technique to reduce the circuit complexity. As a result, the reduced system matrices are dense due to the nature of the projection based reduction. Solving dense matrices is very expensive for traditional single-core or even multicore CPU architecture. However dense matrix solving is very amenable for GPU computing as dense matrices based LU factorization and triangular factor solvers on GPU platforms have been well studied and implemented in CUBLAS (for triangular matrices solvers) [7] and in MAGMA library [8] due to more regular memory access patterns and simple data dependency of dense or full matrices. For instance, for LU factorization using right-looking method with look-ahead techniques, the row exchanging can be done very efficient in the row-major format as coalescent memory access can be explored [8].

IV. Numerical Results

Our ETBR-GPU is implemented in C, Its GPU part is incorporated into the main program with CUDA C programming interface. The accuracy and efficiency of the program are tested on several RLC benchmark circuits, including two industrial benchmark power grid network circuits from IBM [6]. Their complexities range from 40 thouand to 3 million nodes.

The computer running the these programs is a Linux server with two-sockets, each of which hosts an Intel 2.4 GHz Xeon Quad-Core CPU. In total, there are 16 threads from the 8 CPU cores. The host side has 36 GBytes memory available for the two CPUs. Meanwhile, the GPU card installed on this server is Tesla C2070 containing 448 cores running at 1.15 GHz and up to 5 GBytes global memory.

To put our new simulator's performance into a right perspective, we compare ETBR-GPU with the multicore-CPU version of ETBR and a standard LU-based method based on UMFPACK [9]. We remark that we do not compare ETBR-GPU with any iterative solvers as most of existing iterative solvers are highly tuned to specific problems and are not general enough for commonplace linear dynamic systems. On

TABLE I

PERFORMANCE COMPARISON OF STANDARD LU, ETBR AND ETBR-GPU METHODS. N, AND q ARE ORDERS OF ORIGINAL SYSTEM AND REDUCED SYSTEM, CORRESPONDINGLY. $N_{\rm cur}$ IS THE NUMBER OF CURRENT SOURCES. M IS THE NUMBER OF TRANSIENT TIME STEP. THE AVERAGE AND MAXIMUM ERRORS ARE MEASURED IN VOLTS. ALL TIME RESULTS ARE MEASURED IN SECONDS.

ckt	N	$N_{\rm cur}$	M	$T_{\rm LU}$	Proposed Method—ETBR-GPU							Speedup over LU	Speedup over ETBR
					MOR by ETBR				Transient Simulation				
					q	$T_{\rm ETBR}$	ave err	max err	$T_{\rm CPU}$	$T_{\rm GPU}$	Speedup		
pgckt1	89,200	1,000	20,000	162.1	16	2.7	4e−6	9e−5	12.2	5.8	2.1×	18×	1.8×
pgckt2	366,400	1,000	20,000	656.5	8	3.1	1.3e−5	5e−4	6.5	3.0	2.2×	107×	1.6×
pgckt3	3,064,800	4,000	2,000	714.2	8	37.2	1.4e−4	5e−4	33.5	16.6	2×	13×	1.3×
ibmpg1t	39,680	10,774	10,000	107.6	48	4.7	1e−3	8e−3	158.6	1.66	96×	17×	26×
			20,000	206.8		4.9	1e−3	8e−3	305.2	2.92	104×	25×	39×
			40,000	412.7		5.0	1e−3	8e−3	597.1	5.81	102×	37×	55×
			80,000	805.2		5.0	1e−3	8e−3	1126	10.0	112×	52×	75×
ibmpg2t	164,237	36,838	5,000	324.4	32	54.0	1e−3	6e−3	131.2	13.2	10×	4.5×	3×
			10,000	650.7		55.1	1e−3	6e−3	263.8	13.6	20×	9.5×	5×
			20,000	1127.4		55.9	1e−3	6e−3	491.5	15.2	32×	16×	8×
			40,000	2525.9		60.4	1e−3	6e−3	1046.6	18.3	57×	32×	14×

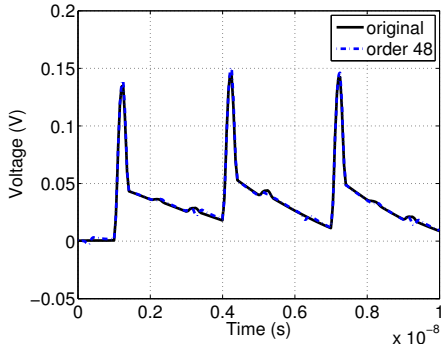

Fig. 3. Transient waveforms of standard LU and ETBR-GPU at one port node of ibmpg1t.

the other hand, ETBR or ETBR-GPU is a general simulator for linear dynamic systems and it does not assume or exploit any structures of the given systems. As a result, it will be more fair to compare ETBR-GPU with the general LU-based simulator. Fig. 3 shows the simulation results of *ibmpg1t*, from IBM. It is a voltage waveform at node `n0_2679_17913`. ETBR-GPU's result matches the golden renference very well.

The statistics of run time and speedup are shown in Table I, where $T_{\rm LU}$ is the time for LU-based method, $T_{\rm ETBR}$ is the CPU time for reduction only. $T_{\rm CPU}$ is the time for transient simulation of reduced systems on CPU and $T_{\rm GPU}$ is the time for transient simulation of reduced systems on GPU. The "Speedup" column is the speedup of the transient simulation (only) of GPU over CPU. "Sspeedup over LU" is the total time of ETBR-GPU over the LU-based method. "Speedup over ETBR" is the ETBR-GPU speedup over ETBR on CPU only.

We have several observations. First, for the first three power grid circuits, ETBR does not need high order to achieve a good accuracy as switching speeds of currents are not very high. The ETBR-GPU speedup over LU is quite impressive (can be $100\times$), although ETBR-GPU does not significantly faster than ETBR as the gain from the simulation of the reduce matrices are not significant due to the small sizes of the reduce matrices. Second, for the two IBM benchmarks *ibmpg1t* and *ibmpg2t*, high order reduced models are used as the currents are swiching faster. In this case, transient simulation of the reduced models become more expensive as the reduced matrices become larger. But ETBR-GPU can still deliver $10\times$ speed up over the LU-based method, since the GPU dense matrix solvers are much more efficient than CPU as the first speedup shows. Third, the longer simulation time is, the better speedups we will have, because GPU portion of the computing will grow more dominant in terms of total costs and GPU raw powers are fully unleashed. Theoretically, the speedup of ETBR-GPU over LU will be up-bounded by the first speedup in the table, which can be $100\times$.

V. CONCLUSION

A new algorithm has been proposed for transient analysis of general linear dynamic networks, such as on-chip power grid networks, on the hybrid GPU CPU multicore platforms. The new algorithm, called ETBR-GPU, is a reduction based simulation and is very amenable for parallelization on the hybrid multicore CPU and GPU platforms. The proposed method is capable to analyze any linear networks with complicated structures and macromodels, and especially favors transient simulation with long simulation periods as reduction costs in frequency domain is less sensitive to the simulation time. Experiments show that the ETBR-GPU can achieve one or two orders of magnitudes speedup when compared to the general LU-based simulation on IBM power grid benchmarks.

REFERENCES

[1] International technology roadmap for semiconductors (ITRS), 2010 update, 2010. http://public.itrs.net.

[2] S. R. Nassif and J. N. Kozhaya. Fast power grid simulation. In *Proc. Design Automation Conf. (DAC)*, pages 156–161, 2000.

[3] J. M. Wang and T. V. Nguyen. Extended Krylov subspace method for reduced order analysis of linear circuit with multiple sources. In *Proc. Design Automation Conf. (DAC)*, pages 247–252, 2000.

[4] H. F. Qian, S. R. Nassif, and S. S. Sapatnekar. Random walks in a supply network. In *Proc. Design Automation Conf. (DAC)*, pages 93–98, 2003.

[5] D. Li et al. ETBR: Extended truncated balanced realization method for on-chip power grid network analysis. In *Proc. DATE*, pages 432–437, 2008.

[6] IBM power grid benchmarks. http://dropzone.tamu.edu/~pli/PGBench/.

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

[8] S. Tomov et al. Dense Linear Algebra Solvers for Multicore with GPU Accelerators. In *Distributed Processing, Workshops and Phd Forum (IPDPSW)*, pages 1–8, 2010.

[9] UMFPACK. http://www.cise.ufl.edu/research/sparse/umfpack/.