Clustering and Parallel Processing on GPU to Accelerate Circuit Transient Analysis

Shital V. Jagtap and Y. S. Rao

Abstract Today, in the age of digital era, electronic circuit is the key component and its design, testing is validated through simulator. But even though use of simulator is cost-effective, large circuit simulation is quite time consuming. Also various iterations in transient analysis might make simulation slow. In this paper, we have addressed parallel computing approach using Graphics Processing Unit (GPU). Forming clusters of executable procedures are very crucial, so that it can be mapped to graphics processor for parallel processing. In every circuit nodal analysis finds current, voltage etc. at various nodes periodically. Matrix operations, linear–nonlinear equations, integration, differential equations, numerical methods are some of the very basic operations required in circuit analysis. Data-code partitioning, parallel data mapping, reductions, fast memory access, parallelizing loops are the strategies adopted for parallel processing on GPU. More than 40% speed gain is achieved on circuit having at least four components along with transient analysis for more than thousand iterations.

Keywords GPU (Graphics processing unit) · Transient analysis · Clustering LU decomposition

1 Introduction

Circuit simulators are used in almost all the electronic industries and plays very crucial role in electronic circuit design, verification and testing. All electronic designs rely truly on simulation software. In academics also, students adopts safe practices and research on simulators. DC, AC, transient, pulse or noise analysis and its graph

S. V. Jagtap (✉)
RAIT, Nerul, Navi Mumbai, India
e-mail: svjagtap@gmail.com

Y. S. Rao
SPIT, Andheri, Mumbai, India
e-mail: ysrao@spit.ac.in

© Springer Nature Singapore Pte Ltd. 2019

R. Kamal et al. (eds.), *International Conference on Advanced Computing Networking and Informatics*, Advances in Intelligent Systems and Computing 870,
https://doi.org/10.1007/978-981-13-2673-8_36

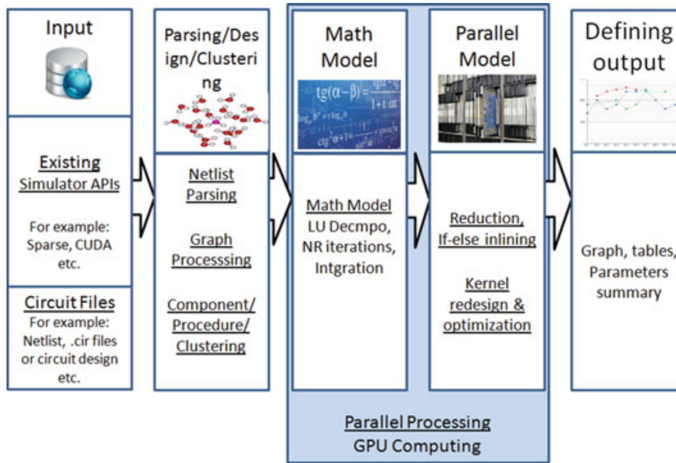

Fig. 1 System architecture

can be derived easily using simulator. But, with the density of the IC at nanoscale, the traditional simulators require more and more time to perform simulation as a whole. Transient analysis is time domain operation in which many Newton Raphson iterations are required which are very compute intensive. The simulation time of transient analysis in SPICE grows super-linearly with the number of equations that describe the circuit. For large circuit transient analysis time may extend to many hours or days. Transient analysis iteration scales as O(N1.2) where N is number of equations, whereas the double precision FLOPS of CPUs only as O(N0.96) where N is the number of transistors in the CPU [1, 2]. It means that as circuit components increases, CPU simulation time also increases exponentially.

Figure 1 shows various steps and execution of simulator in serial and parallel mode. To process circuit parameters various mathematical and load operations are required which process these parameters thousands of time and need millions of clock cycles. Some operations need common processing and common parameters. We applied clustering approach on these parameters and their operations. These clusters are then mapped on GPUs to process operations in parallel on many parameters.

This paper addresses system for GPU accelerated circuit simulation especially for transient analysis. Paper is organized as follows. Section 1 is introduction to topic and Sect. 2 gives details of previous approaches for accelerating circuit simulation. Section 3 elaborates the adopted GPU programming strategies and Sect. 4 explains clustering approach. Section 5 gives performance analysis and results.

2 Previous Work

To accelerate the simulation many approaches are proposed like parallel processing, distributed computing or specialized optimized algorithm. Transistor model is accelerated using GPU in SPICE simulator by Gulati et al. [3]. This acceleration was model-specific that is applicable to transistor model only. If-else in-lining and coalesced memory access is also proposed which helps in accelerating the operations on GPU and is useful to every GPU application. The work started by Gulati was the good start in research of parallel processing in circuit simulation. Chen and Wang used pivoting reduction technique for LU sparse solver [4], which is faster compare to the solvers like NICSLU or PARDISO. NICSLU uses a column-level dependence graph to schedule the tasks [5]. The dependence is extracted from the symbolic structure of the factors. Since the structure of the factors cannot be determined before factorization, the elimination tree (ET) is used to represent the dependence. It follows all the basic steps like numerical update, pruning with proper pivoting. These steps consume extra time for simple dense small size matrix processing. FPGA based techniques are also available for LU decomposition which are better than serial processing [6–8]. Compare to FPGA techniques GPU parallel programming is simple and easy to understand. In some approaches rather than accelerating computations, circuit or gate design is partitioned to find independent components and mapped it in parallel [9, 10]. Every circuit analysis need lots of computation, so some parallel and distributed approaches are available for mathematical operations like LU decomposition, integration [11–14]. Chen and Wang modified it for circuit simulation but for large circuit [15, 16], matrix column dependency causes slower simulation. Davis and Natarajan proposed KLU data structures and algorithm which can be used in circuit data storage and processing for sparse matrices [2]. It gives stability in matrix processing. For small circuit simulation it proves to be slow but adopted by many simulators like NGSPICE. Saol, Vuducl and Xiaoye proposed distributed approach to solve sparse matrix. This is costly approach but can be useful to extend GPU technology also. From all the available approaches, parallel processing using GPU is very cost effective approach. New faster GPUs are coming in the market. So research is still persistent to accelerate simulation on new faster GPUs. This paper focuses on utilizing computational power of GPU for heavy computations of circuit transient iterations.

3 GPU Strategies

GPU processes graphics and includes thousands of SIMD multi-threaded, multi-core processors with inbuilt memory levels having different sizes and access time. For example Kepler K40 graphics card contain 2880 cores. GPU processor do not consume enormous power, heat indulgence is adequate, so can be used with laptops or small systems. Cost of GPU is just some thousand rupees. Access to high end GPU

is available free of cost online through GPU clusters (from GPU Excellence centre). GPU is an emerging parallel processing approach for heavy computations. CUDA is the software platform available for GPU. It supports heterogeneous programming. Due to SIMD nature, sequential code is not directly executable on GPU. Redesign and optimization is needed in memory access-storage, execution configuration, instruction cycles and control flow. Following strategies are adapted to modify and redesign sequential code so that it will execute faster on GPU especially for cluster based circuit simulation.

1. Kernel formation and optimization—Avoid costly operations and replace them using less costly operations in kernel. Proper partitioning of data or operations is required which uses adequate kernel size. Two parameters are considered to decide proper kernel size: a. Maximum GFLOPS obtained from kernel and b. Maximum memory bandwidth used by that kernel.
2. Parallel reduction—Reduction is a generic operation that takes $n > 1$ values and returns a single value. Elements can be re-arranged and combined in any order. Threads need to access results produced by other threads using either shared memory or by synchronisation. Key requirements for a reduction operator $\circ$ are:

   a. Commutative $a \circ b = b \circ a$; b. Associative $a \circ (b \circ c) = (a \circ b) \circ c$

3. Minimize loops by solving data dependency.
4. If-else in-lining-Use minimum if-else statement so that optimum time is used for execution on all threads.
5. Avoid warp and thread divergence in CUDA kernel. If possible interchange the work done by warps.
6. Utilize memory hierarchy: a. Coalesced memory access utilizes optimum time to read or write data items. b. If possible copy data in shared memory or registers for fast access. c. Memory Bandwidth calculation-Choose the kernel giving highest speed up. d. Reduce/Eliminate data transfers between CPU and GPU. Combine multiple device memory allocations and transfers in one step. cudaMalloc() and cudaFree() are costly operations so minimize them by reusing the allocations. e. Use page-locked host memory for data transfers. f. Use asynchronous data copy if possible.

4 Clustering for Circuit Simulation

After compilation of netlist many procedures are required to simulate the circuit. We developed approach for circuit component and procedure partitioning and optimized for a highly-parallel GPU. Moreover, flow is structured to extract the best simulation performance from the given circuit execution. Logic is used to verify designs at the behavioral level, as well as the structural level, ensuring that a synthesized circuit's procedure cluster matches the functionality and timing of the behavioral model.

4.1 Circuit Analysis

Transient analysis is one that finds voltage (or current) versus time. Linearization of non-linear devices, operating point analysis, conductance stamping into the modified nodal analysis (MNA) matrix and linear system solution are time consuming itself. Matrix methods like LU decomposition can be used to solve linear equations. Dense matrix library proves to be time and space consumable if matrix is sparse. Various numerical methods like Newton-Raphson, Runge-Kutta or trapezoidal methods are useful to find roots of equations, derivative or integration. These methods can be parallelized to accelerate the computations.

As the execution of software functions are concerned, execution time varies. All the ‘load’ functions in simulator are compute intensive. It loads default parameters along with calculated parameters into the simulation matrix and its execution time sums up to approximately 54% of total analysis time. Other time consuming component is actual matrix solving to find unknown circuit node parameters which is approximately 36% [1]. In order to complete the analysis of time spent in the transient analysis—5% of time in circuit error, the truncation error calculation, 2% in NI iterations and 2% in NI integration. Then 1% is spent in rest of simulator. MNA matrix is solved using LU decomposition. Left-looking LU decomposition algorithm works columnwise and convenient to make it parallel [11]. Calculation of one column depends on values generated by previous column. Assign one column at a time to GPU and execute in parallel. Launch ‘n’ threads, where ‘n is column size. One thread is executing one element of column. Proper synchronization is required to remove any data dependent operation. Serial algorithm worst case complexity was O(n3) whereas parallel algorithm complexity is O(n2).

4.2 Clustering

54% of simulation time is needed just in load and setup operation even if numbers of components are very few. If component count increased, it increases setup time exponentially. But if we form one load cluster of all components, load time increases by few clock cycles, at least not exponentially. So for large circuit or for transient analysis, cluster formation is better compare to manual parallel execution. Figure 2 shows approximate clusters formation. There are two modes to process complete netlist using clustering approach.

1. Component clusters—This method create clusters of components of same level. Same level means fanin of components is ready in previous level. It helps in execution of one cluster is possible at a time. Graph DFS algorithm is used select component clusters.
2. Method/Procedure clusters—All component have some common execution steps like setup or load etc. Creating clusters of these methods is method clustering.

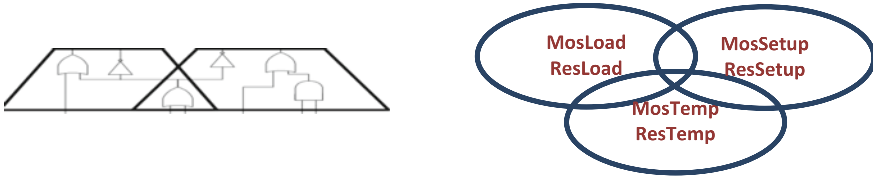

Fig. 2 Component and method clusters

Method clustering is more suitable for GPU computing as many instance of thread creation is possible due to SIMD nature of GPU.

Common operations are either with same component or different component. Same component means circuit may contain many instances of resistors or capacitors. If components are same, sub-operations are exactly same, so cluster operations are exactly same. It helps in increasing speed more compare to different components. For different components like transistor and diode, some of the sub-operations are same. So can be mapped to many threads, but thread processing time may vary. Following are some cluster creation precautions:

4.2.1 Cluster Constraints and Data Set

Data set for circuit simulation with respect to GPU processing are components with all parameters like Resistance, conductance, drain current, base current etc. In determining how to partition the netlist into clusters, we considered several factors: (i) Similarity in process with parameter data type (ii) Time required to execute functions. GPU specific factors are: (i) Minimum dependencies among parameters in various iterations. (ii) Time required to execute process should be more than a. time to load the data in GPU memory. b. overhead of selecting process from process set.

4.2.2 Partitioning

To exploit the parallelism available in the GPU, segment the simulation procedure into several logic blocks. a. Divide every logic block into sub-blocks, containing one operation. b. Compare sub-blocks of one component with other component sub-blocks. Uniform sub-blocks are given same weight. c. Apply k-means algorithm. Define number of clusters and initial mean as basic operation for every cluster. In second step we get the distance vectors. Compare distance vectors and select subblock of minimum distance as an operation in that cluster. Constraints defined above can act as threshold for distance vector and used to determine—in which cluster procedure should be added. d. Distance may vary and there may be overlap also in the operations. In the overlap, we can set flag to indicate operation is executed and don't execute it again.

4.2.3 Procedure Balancing

Cluster parameters are copied in global and shared memory. In some structures only pointers are defined and has to be converted into actual structure before copying. We map one cluster to graphics processor at a time. As fine-level granulaity, we can even divide one load method into many instances. E.g. if there are 4 resistor components, resload is repeated at least 4 times. On GPU, we can create four threads to execute load resistor function in parallel. But we need thread synchronization to manage cyclic dependency among the variables.

4.2.4 Simulation Phase

Sequence for cluster execution is based on following parameters: 1. Sequence and privilege of operation. 2. Data or input/output dependency. 3. Synchronization dependency 4. Data availability in global/shared memory. There are restrictions on GPU memory size and extra time is needed in loading, unloading all the parameters. So if cluster execution time is much more compare to serial time plus loading time, that cluster execution is suitable to make it parallel.

Sufficient memory size is also required to process all the components at a time, as for large circuit millions of components with thousands of parameters are used.

5 Performance Comparisons

Circuit netlist having basic components are tested on GeForce M980 processor with 296 cores and 2 GB graphics card memory. NGSPICE simulator is used on Ubuntu 14.04 version. NGSPICE with KLU version is considered for comparison. Execution time of thousands of transient iterations are considered. One complete iteration involves initialisation, setup, load, LU decomposition, forward- backward substitution. Setup and load involves many small mathematical operations like solving algebraic equations, integration etc. Netlist parsing time is constant for all the circuits. Clustering and parallel processing is used in parameter setup, load and mathematical operations. Speed gain is calculated using the Eq. (1)

$$\text{Percentage speed gain} = 100 * (\text{serial} - \text{parallel time})/\text{parallel time} \quad (1)$$

Average execution time is considered for every circuit execution. Table 1 gives serial and parallel execution time of five example circuits having thousands of transient iterations. Figure 3 shows relative time comparison graph among serial and parallel execution.

It shows that basic GPU strategies and clustering approach accelerate circuit processing at least by 40% and increases subsequently for more iterations.

Table 1 Serial and parallel execution of different netlist

Netlist	Circuit	No. of iterations	Serial execution in (s)	Parallel execution in (s)	Speed gain (%)
1	AC sine wave voltage	10,000	6.38	3.75	70.13
2	RC circuit	1008	0.853	0.58	47.06
3	Full wave bridge rectifier	10,000	12.54	8.23	52.36
4	Common source jfet amplifier	10,000	8.71	4.26	104.46
5	Integrator with square wave input	2520	2.115	1.4875	42.21

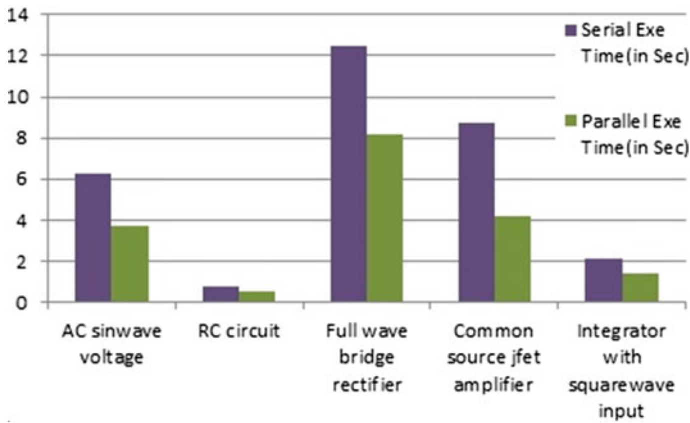

Fig. 3 Execution time comparison graph

References

1. F. Lannnutti, P. Nanzi, M. Olivieri, KLU sparse direct linear solver implementation into NGSPICE, in *19th International Conference on Mixed Design of Integrated Circuits and Systems*, Poland, 24–26 May 2012
2. T. Davis, E. Palamadai Natarajan, Algorithm 907: KLU, a direct sparse solver for circuit simulation problems. ACM Trans. Math. Softw. **37**(3), Article 36, Sept 2010
3. K. Gulati, J.F. Croix, S.P. Khatri, R. Shastr, Fast circuit simulation on graphics processing units (IEEE, 2009), pp. 403–408
4. X. Chen, Y. Wang, H. Yang, A fast parallel sparse solver for SPICE-based circuit simulators, 978-3-9815370-48/DATE15/c2015, EDA
5. X. Chen, Y. Wang, H. Yang, NICSLU: an adaptive sparse matrix solver for parallel circuit simulation. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **32**(2), Feb 2013

6. G. Wu, Y. Dou, G.D. Peterson, Blocking LU decomposition for FPGAs, in *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*
7. Y. Shao, L. Jiang, Q. Zhao, Y. Wang, High performance and parallel model for LU decomposition on FPGAs, 978-0-7695-3932-4/09 $26.00 © 2009 IEEE
8. M.K. Jaiswal, N. Chandrachoodan, FPGA-based high performance and scalable block LU decomposition architecture. IEEE Trans. Comput. **61**(1), Jan 2012
9. D. Chatterjee, A. Deorio, V. Bertacco, Event driven gate-level simulation with GP-GPUs (ACM, 2009), 978-1-60558497-3/09/07
10. D. Chatterjee, A. Deorio, V. Bertacco, Gate-level simulation with GPU computing. ACM Trans. Des. Autom. Electron. Syst. **V**
11. H.M.D.M. Bandara, D.N. Ranasinghe, Effective GPU strategies for LU decomposition, in *IEEE International Conference on High Performance Computing* (2011)
12. L.F. Cupertino, A.P. Singulani, C.P. da Silva, M.A. Pacheco, LU Decomposition on GPUs: the impact of memory access, 978-0-7695-4276-8/10 $26.00 © 2010 IEEE
13. T. Dong, A. Haidar, P. Luszczek, J.A. Harris, S. Tomov, J. Dongarra, LU factorization of small matrices: accelerating batched DGETRF on the GPU (2014)
14. N. Galoppo, N.K. Govindaraju, M. Henson, D. Manocha, LU-GPU: efficient algorithms for solving dense linear systems on graphics hardware (ACM, 2005), 1-59593-061-2/05/0011
15. L. Ren, X. Chen, Y. Wang, C. Zhang, H. Yang, Sparse LU factorization for parallel circuit simulation on GPU (ACM, 2012), 978-1-4503-1199-1/12/06
16. X. Chen, Y. Wang, H. Yang, An adaptive LU factorization algorithm for parallel circuit simulation, 978-1-46730772-7/12/$31.00 ©2012 IEEE