Literature Survey on GPU Accelerated Circuit Simulation

Shital V. Jagtap
RAIT, Nerul
Navi Mumbai,Maharashtra, India

Dr. Y. S. Rao
SPIT, Andheri
Mumbai, India

Abstract— Analysis, Testing and validation of electronic circuit is very crucial in industries of electronics and embedded systems. Instead of actual hardware testing, simulation software is used for this purpose. Very large circuit design like VLSI testing, affects speed and accuracy of such software tools. So there is the need to increase speed of simulation software without compromising quality. GPUs(Graphics Processing Unit) as a many core processors, are used for parallel processing. To get the advantage of parallel processing in circuit simulation, GPUs can be helpful. This paper explores various methods to accelerate electronic circuit simulation by using parallel processing on GPU. Event-driven, gate-level, model driven approaches are discussed here. Parallelism is induced in mathematical methods used for finding various parameters like voltage, current etc. Speed can be increased from 2 to 150 times for various designs.

Keywords –Graphics Processing Unit(GPU), Newton-Raphson (NR) method, transistor model, Look Up Tables(LUTs)

I. INTRODUCTION

Testing of integrated circuits, electronic design and hardware testing is expensive, impractical, and getting the behavior of internal signals are extremely difficult. Therefore almost all IC design relies heavily on simulation. Simulation software allows for modeling of circuit operation and is an importat analysis tool due to its highly accurate modeling capability. Learners can also analyze, synthesize, organize, and evaluate content and result of electronic circuit through simulation. Simulating a circuit's behavior before actually building, can greatly improve design efficiency by removing faulty designs.

For very large circuit design like VLSI development, sequential simulators need large execution time and memory. FPGA have long execution times that are not acceptable, but are used since there are no other superior known approaches to verify circuits. In order to shorten this time period, we must try to the use of distributed or paralle1 methods. Distributed approach is costly compare to parallel processing on GPU. Now a day's cost effective thousand core GPUs are available with sufficient memory. To take advantage of GPU for parallel processing, there is the need to analyze design to find independent execution components, data dependency and memory access.

Various ways are developed to induce parallelism in simulation of electronic circuit. Transistor model consumes most of the time of simulation, so parallelism is tried on BSIM3 model of simulator program[1]. Gate level analysis and clustering is used to identify independent components for mapping to GPU[2]. In TinySPICE approach many uniform components are mapped in parallel[3]. Some mathematical model are implemented on GPU[5][6].But there is the need of parallelizing some more mathematical models. This paper addresses all those methods in detail.

This paper is organized in different sections. Section II describes main architectural features of GPU which are helpful for parallel programming. Section III gives details of various methods used for electronic circuit simulation using GPU. Section IV gives discussions on all methods.

II. GPU ARCHITECTURE

GPU(Graphics Processing Unit) is processor used for graphics processing. It is the many core processor. Compare to CPU which contain maximum 8 to 16 cores, GPU contain thousand of cores with memory about 32GB. GPU contain set of SIMD cores. So GPU is suitable for the problems that can be expressed as data-parallel computations ie. same program is executed on many data elements in parallel with high arithmetic intensity. Now a days research is going on in utilizing high computing power of GPU for non-graphics applications. CUDA is the software platform available by NVIDIA for parallel programming on GPU[7]. It

divides tasks between a CPU host and GPU device, that is called heterogeneous computing. Kernel functions are called by the host, but executed on the device by multiple threads simultaneously.

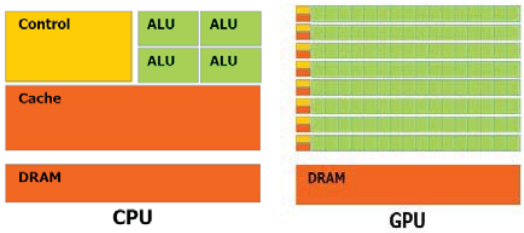

Figure.1. CPU vs GPU architecture

Logically, threads are grouped into 1D, 2D, or 3D thread blocks[7]. When called, a kernel is given an execution configuration, specifying block and grid sizes. Individual threads are free to follow any path through the conditionals and loops of kernel code, but the hardware gives better performance for same type of computations. In hardware, the GPU is made up of a number of multiprocessors. Thread blocks are assigned to multiprocessor at a time. The threads of a block are executed by the multiprocessor in groups of 32 called warps. Warp executes only one instruction at a time. If threads of a warp have taken different paths, performance decreases. Optimal performance is achieved by maximizing the number of concurrently executing threads, constrained by memory resources.

III. CIRCUIT SIMULATION METHODS ON GPU

A. Transistor model evaluation

Gulati suggested the approach for accelerating transistor model acceleration using GPU[1]. The SPICE algorithm and its variants simulate the nonlinear time-varying behavior of a design, by employing the key procedures like-

a. Formulation of circuit equations using Modified Nodal Analysis (MNA) or Sparse Tableau Analysis (STA).
b. Evaluating the time-varying behavior of the design using numerical integration techniques, applied to the non-linear circuit model.
c. Solving the non-linear circuit model using Newton-Raphson (NR) based iterations.
d. Solving a linear system of equations in the inner loop of the engine.

The main time-consuming computation in SPICE simulator is the evaluation of device model equations in different iterations. Gulati and Croix found that transistor model takes nearabout 75% of the SPICE runtime[1]. This is because these evaluations are performed for each device, and possibly repeated for each time step, until the convergence of the NR based non-linear equation solver. The total number of such evaluations can easily run into the billions. Therefore, the speed of the device model evaluation code is a significant determinant of the speed of the overall SPICE simulator.

The key contributions in the implementation are:

a. Explorer the match between parallel device model evaluation and the capabilities of a GPU, a SIMD-based device.
b. The different threads, which perform device model evaluations, are implemented so that there are no data or control dependencies between threads.
c. All device model evaluation threads compute identical instructions, but on different data, which exploits the SIMD architecture of the GPU.
d. The values of the device parameters required for evaluating the model equations are obtained using a device memory lookup, thus exploiting the extremely large memory bandwidth of GPUs.
e. Device model evaluation is implemented in a manner which is aware of the specific constraints of the GPU platform such as the use of texture memory for table lookup, memory coalescing for device memory accesses and the balancing of hardware resources used by different threads.

BSIM3 device model computations can be parallelized in a similar manner.

a. Inlining if-then-else Code: If-else code may cause thread divergence and some thread may remain idle. So inlining of code is required. In this system execute if-then code and if-else code one after another. Now depending on condition select proper result.
b. Partitioning the BSIM3 Code into Kernels: As the GPU is SIMD type of processor, modifications in the code is required to map independent components on thousands of threads. So kernel is partitioned into many small or large kernels.
c. Efficient use of GPU Memory Model: To reduce data transfer time and utilize the features of memory architecture of GPU.

Average speed acceleration in SPICE execution on GPU by above given strategies is 2.36x over CPU execution. For some circuit designs maximum speed up is 4x also.

This acceleration is model specific that is applicable to specific model only. If-else inlining 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.

B. Event-driven Gate level simulation

Chatterjee proposed acceleration through parallel mapping of components[2]. Circuit or gate design is partitioned to find independent components and mapped it in parallel. It is used to verify designs at the behavioral level, as well as the structural level, ensuring that a synthesized circuit's netlist matches the functionality and timing of the behavioral model[2]. Gate-level simulators proceed in two phases: During the first phase, the circuit netlist to be simulated is restructured and optimized. It is also called compilation phase. In the second phase, the netlist is executed. The performance of the simulator is driven by this second phase, since the compilation step is only required once per netlist. In an event-driven simulation, a gate is simulated only if at least one of its input values had changed, while in an oblivious simulator all gates are evaluated with each cycle. It is typical for large designs to only simulate a small fraction of the gates individual execution threads. It would operate on distinct netlist clusters and communicate in an event-driven fashion, with a thread being activated if switching activity was observed at the inputs of its netlist cluster. To emulate a circuit netlist, a compiler partitions the netlist into blocks and then loads each block into separate units. First applies a compilation phase, during which it transforms the netlist to leverage the raw performance of the target architecture. This is followed by a simulation phase where the compiled netlist is uploaded to the GPU co-processor and one or more simulations may be executed with different input testbenches.

Factors important in netlist partitioning are a) Time required to simulate a macro-gate should be greater than overhead of determining which macro-gates to simulate. b) CUDA's multiprocessors can only communicate through device memory, thus macro-gates should not share data. c) To avoid cyclic dependencies between macro-gates, simulate each macro-gate at most once per cycle.

To address the constraint list, segment the netlist by partitioning it into layers. Each layer encompasses a fixed number of the netlist's levels. Macro-gates are then defined by selecting a set of nets at the top boundary of a layer, and including its cone of influence back to the input nets of the layer.

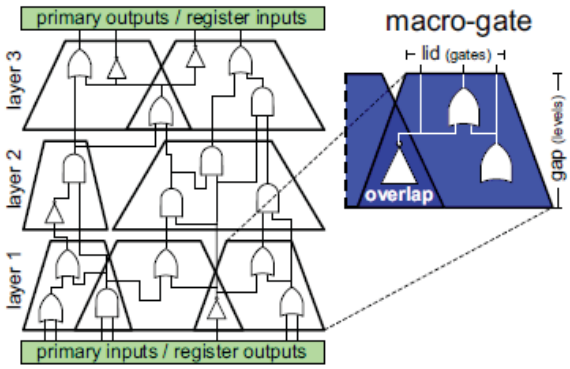

Figure. 2. Macro-gate structure[2]

The combinational netlist is levelized, that is, logic gates are organized into levels as shown in figure 2. With this organization, it is possible to simulate the entire netlist one level at a time, from inputs to outputs, with no backward dependency. Macro-gate means several gates of the original netlist connected to each other. Levelize means logic gates are organized into levels, so that the fanin of all gates in one level is computed in previous levels.

The number of levels within each layer is called the gap and corresponds to the height of the macro-gate. By using this procedure, it is possible that a given logic gate is assigned to two or more macro-gates. In this case, duplicate it, so that each macro-gate can compute the value of its output nets without sharing any data with other macro-gates. Finally the number of output nets used to generate each macro-gate is a variable parameter called lid, whose value is selected so that the number of logic gates in all macrogates are approximately the same. Figure 2 shows a schematic of the segmentation technique. The set of nets that cross the boundary between each pair of layers is monitored during simulation to determine which macro-gates should be activated. In this prototype implementation used is an ALAP (as-late-as-possible) levelization.

Select optimal values for gap and lid, so as to achieve a high-level of parallelism during simulation with little macro-gate overlap and low activation rates. In the prototype implementation, gap and lid are fixing across the

entire netlist. Additional performance could be achieved if each layer had its own associated gap and each macrogate had an associated lid. For large scale industry design having millions of gates this system proves to be better. Compare to other event driven this gate level event driven system gives 13x speedup.

This approach is different from previous, as it works on design level rather than model level. So this is also a new good direction for research. But this strategy is applicable to digital gates design partitioning only. Also for every partition, its data or attributes should be copied to GPU memory for proper usage. Partitioning and data copy time may decrease the overall performance of system.

C. TinySPICE

In this approach SPICE simulator is parallelized for massive uniform data on GPU like memory cell design and this approach is proposed by Han and Zhao[3]. TinySPICE accelerates the entire SPICE simulation computations on GPU without introducing excessive CPU-GPU data communications and device memory accesses. It can analyze small nonlinear circuits in GPU's shared memory. General nonlinear electronic circuit simulation techniques rely on NR method to solve the differential equations.

In order to obtain the stamping locations of nonlinear elements in the system MNA matrix, it is necessary to store the terminal indices of each nonlinear device. Figure 3 shows how to store terminal indices of all transistors into a long index mapping vector. In the Mos_map vector, Idx stands for the corresponding LUT storage index for a transistor and d,g,s, and b represent the indices of each transistor's terminals in the MNA matrix, respectively.

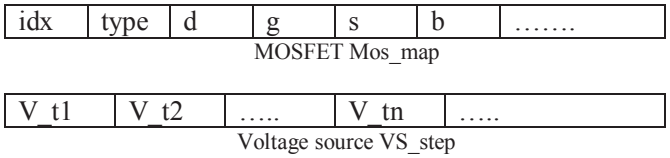

Figure 3. Memory layout of Vectors for storing LUTs

With such mapping information, device evaluation results from the LUTs can be directly written into the system MNA matrix as well as the RHS(Right Hand Side) vector. Since the elements of linear devices such as resistors, capacitors and inductors have fixed values, their corresponding stamped elements in the MNA matrix will not change throughout the entire SPICE simulation.

It can be pre-evaluated and stored into a linear-element matrix. During each NR iteration, linear-element matrix is combined with the nonlinear-element matrix obtained from nonlinear device evaluations, direct solution methods such as LU factorization etc. Then it can be applied to compute the solutions. TinySPICE simulator adopts dense matrix structure.

Newton-Raphson Iterations: Allocate system MNA matrix and RHS in registers for each GPU thread. Load linear element matrix, RHS vectors, index mapping vectors from GPU's texture memory to shared memory. There are iterations of NR method which perform following operations repeatedly-

1. Reset system MNA matrix and RHS vector by loading initial data from shared memory.
2. Evaluate nonlinear devices.
3. Stamp system MNA matrix and compute the RHS vector.
4. Factorize system MNA matrix of each circuit and solve for the solution vector.
5. Apply a damping factor for the solution if needed.

If NR does not converge then perform another n iterations of steps 1-5. Return solution if NR converged else return error flag.

Since each GPU's streaming multiprocessor (SM) has very limited memory resources, the number of circuits to be analyzed at the same time should be carefully determined based on the circuit sizes and on-chip memory usage. The limited memory can impact the number of GPU threads running on each SM. To achieve the best simulation performance, TinySPICE first finds out the optimal thread block sizes and grid sizes by evaluating simple memory cost functions. Subsequently, the proper thread organization and assignment are determined, and the final simulation code can be compiled for a given circuit design. It is worth noting that different circuit analysis problems may result in different GPU thread settings, and therefore get different speedups compared to CPU-based SPICE simulations. This system is very useful circuit design having millions of uniform components like cell or SRAM design. TinySPICE can achieve up to 138x speedups for a variety of circuit analysis problems without loss of accuracy.

TinySPICE is faster compare to other GPU accelerated circuit simulators. But this system applicable to many uniform small components. For the circuit having millions of uniform components, this system is more efficient.

D. Sparse LU factorization

To find different parameters of any electronic design, like nodal analysis or transient analysis, we need mathematical model. We may need to find current, voltage at different nodes at different time instances. Mathematical model involve matrix calculations, solving differential equations, integration techniques etc.
For simple Modified Nodal Analysis(MNA) ie finding current/voltage at different nodes, matrix approach is better. To find unknowns in linear equations, LU decomposition is useful. For very large circuit, matrix size is also very large and need much time for decomposition. This matrix is unsymmetric, highly sparse and irregular. Chen, Wang, Jaiswal or many researchers implemented LU decomposition on large matrix parallel sparse LU solver on GPU to accelerate the operation[5][6][8]. It includes exposing more parallelism for many-core architecture. Expose enough parallelism to make the sparse LU solver efficient on GPU. In Figure 4 colored steps are executed on GPU and non-colored steps on CPU. Parallelism between vector operations alone is not enough for the thousands of threads running concurrently on GPU. It also utilizes the parallelism within the vector operations. To efficiently deal with the two levels of parallelism, partition the workload based on the features of GPU architecture. This strategy minimizes idle threads, saves synchronization costs, and ensures enough threads in total.

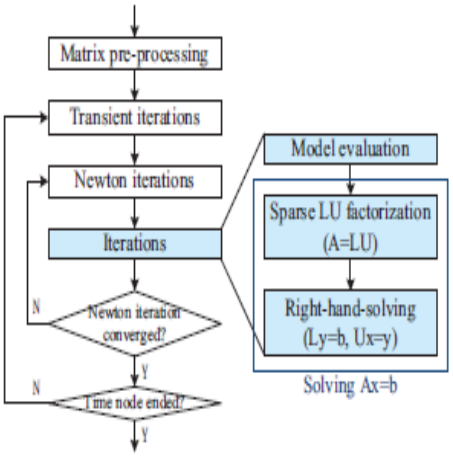

Figure 4. Circuit iterations

Ensuring timing order on GPU involves carefully controlling the number of thread groups. For optimizing memory access pattern a. Design the suitable data format of the intermediate vectors on GPU b. Sort the nonzeros to improve the data locality for more coalesced accesses to global memory. Left looking and right looking algorithm is available for LU decomposition[9]. It need many iterations and many elements of iterations are not dependent on previous iterations. So it is possible to make parallel those operations. For serial execution worst case complexity is more than $O(n^3)$.If we execute it in parallel complexity may scale down to $O(n^2)$. Waiting time for dependent data is very less, So use of left or right looking algorithm is useful to implement these algorithms on GPU.

For very large sparse matrix above given system is very efficient. At the same time for small dense matrix, system is time consuming. Better dynamic system is needed which differentiates between CPU intensive and GPU intensive matrix processing and apply it on required processor.

Other than LU decomposition, many other operations like integration, differential equations are used in circuit simulation. Different numerical methods are used to implement those methods like Trapezoidal and Runge kutta method. Parallel version for some methods are available[13]. These parallel algorithms proves to be efficient for many iterations and large function. But these parallel methods are not yet fully utilized for circuit simulation. For very large circuit operations these methods would prove to be efficient.

IV. DISCUSSIONS AND CONCLUSION

To accelerate circuit simulation, different methods are proposed for GPU execution. In every method one common point is use of Shared memory. Shared memory in GPU is faster than device memory. To accelerate processing, data is frequently copied on shared memory. But it should not minimize arithmetic intensity.

The event driven algorithm provided by mixed-mode simulators is general purpose and supports non-digital types of data. For example, elements can use real or integer values to simulate DSP functions or sampled data filters. As event driven algorithm is faster than the standard SPICE matrix solution, simulation time is greatly reduced for circuits that use event driven models in place of analog models.

Some issues should be taken into consideration for parallel programming on GPUs like communication bottlenecks, conditional control flow, Kernel invocation overheads, Data structures to store circuit elements, Data size and compatibility with SIMT program flow.

Many parameters of circuit like current, voltage etc are calculated using mathematical methods, For example nodal analysis using matrix method. LU factorization, newton raphson iterations, Runge Kutta Integrators are some more operations. NR method and RK method need many iterations for accurate result. These methods gives good acceleration if we execute it on GPU. For that data dependency on previous iterations should be removed and independent components should be identified. Some researchers are using graph structures to find independent components.

Transistor model acceleration, Gate partitioning methods, TinySPICE approaches and mathematical model acceleration are discussed in this paper. All methods are implemented at specific place in circuit simulation. But the combination of all these methods are required and dynamic selection strategy is required which will give good balance between memory utilization and speed.

Use of GPU in circuit simulation is still under research on different GPUs available and newly developed GPUs.

REFERENCES

[1] Kanupriya Gulati, John F. Croix, Sunil P. Khatri and Rahm Shastr, "Fast Circuit Simulation on Graphics Processing Units", IEEE, 2009. Pages- 403 - 408

[2] Debapriya Chatterjee, Andrew DeOrio and Valeria Bertacco, "Event-Driven Gate-Level Simulation with GP-GPUs", ACM, 2009.

[3] Lengfei Han, Xueqian Zhao, Zhuo Feng, "TinySPICE: A Parallel SPICE Simulator on GPU for Massively Repeated Small Circuit Simulations", IEEE, 2013, ISSN 0738-100X, Page 1-8

[4] Xiaoming Chen, Yu Wang, Huazhong Yang,"Parallel Circuit Simulation on Multi/Many-core Systems", IEEE, 2012.

[5] Xiaoming Chen, Yu Wang, Huazhong Yang, "A Fast Parallel Sparse Solver for SPICE-based Circuit Simulators", 978-3-9815370-4-8/DATE15/c2015 EDAA

[6] Manish Kumar Jaiswal, NitinChandrachoodan, "FPGA-Based High-Performance and Scalable Block LU Decomposition Architecture", IEEE TRANSACTIONS ON COMPUTERS, VOL. 61, NO. 1, JANUARY 2012

[7] NVIDIA, "NVIDIA_CUDA_ProgrammingGuide_2.3", cuda2.3 toolkit/docs/NVIDIA_CUDA_ProgrammingGuide_2.3.pdf.

[8] Xiaoming Chen, Yu Wang, Huazhong Yang,"GPU-Accelerated Sparse LU Factorization for Circuit Simulation with Performance Modeling", IEEE, 2013.

[9] H. M. D. M. Bandara, D. N. Ranasinghe "Effective GPU Strategies for LU Decomposition", hipc2011

[10] Youngsok Kim, Jaewon Lee, Donggyu Kim, and Jangwoo Kim, "ScaleGPU: GPU Architecture for Memory-Unaware GPU Programming", IEEE Computer Architectures Letters, Vol. 13, No 2. 2014

[11] Ling Ren, Xiaoming Chen, Yu Wang, Chenxi Zhang, "Sparse LU factorization for parallel circuit simulation on GPU", IEEE, 2012.

[12] Alper Sen · Baris Aksanli · Murat Bozkurt, "Speeding Up Cycle Based Logic Simulation Using Graphics Processing Units", Springer, 2011, DOI 10.1007/s10766-011-0164-7

[13] M. Al-Turany, "Runge-Kutta algorithm for track propagation on GPUs", GSI Scientific Report 2010