YAPSIM: Yet Another Parallel Logic Simulator using GP-GPU

Takuya Hashiguchi, Yuichiro Mori, Masahiko Toyonaga, Michiaki Muraoka

Department of Information Science, Graduate School of Science,
Kochi University
e-mail:{ hashiguc, ymori, toyonaga, muraoka }@is.kochi-u.ac.jp

Abstract

In this paper, a new high-speed logic simulator YAPSIM based on a parallel logic simulation methodology using GP-GPU is presented. It consists of three acceleration methods for simulation performance, a fan-out cone grouping method, a LUT method and a GPU internal memory access method. The experimental comparison result shows that YAPSIM executed 29 times faster than a high speed commercial simulator for a combinational circuit of 75,000 gates, and 5.7 times faster for a sequential circuit of 84,000 gates respectively.

I. Introduction

In late years, newer high-speed circuit validation methods have been reported using parallel processing techniques for logic circuit simulation [1,2,3,4] and system-level simulation [5,6]. These simulators utilize the *CUDA* programming environment of GP-GPU (General Purpose Graphic Processing Unit) as the parallel processing environment.

Literatures of [1, 2, 3] propose logic simulation methods using GP-GPU based on the event-driven method. Literature of [1] presents a dividing method of large logic circuits into several logic cones and an execution method of logic cones in parallel. Literature [2] presents a dividing method of large logic circuits into several logic macros and an execution method of them in parallel. Literature of [3] proposes a logic simulation method by assigning logic gates to GP-GPU THREADs individually. These literatures claim that the speed of their GPU simulators is twice to 40 times higher than that of conventional sequential simulators which are the base of their GPU simulators, however the comparison results of those with the high speed commercial simulators are not shown. Their evaluation results using benchmark LDPC [8] show that their speed is 10 to 100 times slower than that of commercial simulators, that means the speed of their paralleled GPU simulators is not faster than the commercial simulators.

The literature of [4] claims their GPU based simulator is 2.9 to 138 times faster than the "baseline CPU simulator" which is 1.4 to twice faster than the commercial simulator, thus the direct comparison between the GPU simulator and the commercial simulator is not described.

In this paper, a new high-speed parallel logic simulator, Yet Another Parallel logic SIMulator using GP-GPU, named YAPSIM, is described. YAPSIM is based on a levelized model for parallel logic simulation which assigns the logic gates of a circuit on their signal propagation levels to GP-GPU THREADs and simulate them in parallel simply. There are three key techniques of YAPSIM for higher speed simulation as follows. First, YAPSIM uses a fan-out cone grouping method for efficient usage of scalable GP-GPU SMs. Second, it uses a LUT based logic gate calculation method. Third, it brushes up simulation data unit for GPU memory access minimization after the analysis of the internal memory structure.

According to the comparative experimental results between YAPSIM and a commercial simulator, it is found that YAPSIM executed 29 times faster for a combinational circuit of 75,000 gates, and 5.7 times faster for a sequential circuit of 84,000 gates than that by the commercial simulator.

In section II, we introduce a GP-GPU environment and a basic parallel logic simulation method based on the logic levelized model. In section III, the three keys of YAPSIM are described. In section IV, the comparison results between YAPSIM and a commercial simulator with the sequential version of YAPSIM are described. We summarize our proposal in section V.

II. GPU Environment and a Parallel Logic Simulation

A. CUDA/GPU Environment

CUDA is one of the parallel programming environment provided for NVIDIA's GP-GPU [7]. Program module of *CUDA* environment consists of two sub modules, i.e. one is *HOST* as the computer module for CPU and the other is *DEVICE* module for GP-GPU. *DEVICE* module runs in parallel as numerous *THREAD*s allocated in cores on SMs (the streaming multiprocessors of GPU). Here, *THREAD* is one processing unit name. *THREAD*s can use two kinds of memories of *GLOBAL* and *SHARED*. *CUDA* provides multi-thread execution environment for the parallel processing program, while the processing time of it depends on the number of SMs in GP-GPU Products.

BLOCK is a group of *THREADs* in computation logical structure in *CUDA*. The environment assigns all *THREAD*s into available *BLOCK*s automatically. A *BLOCK* is corresponding to one SM.

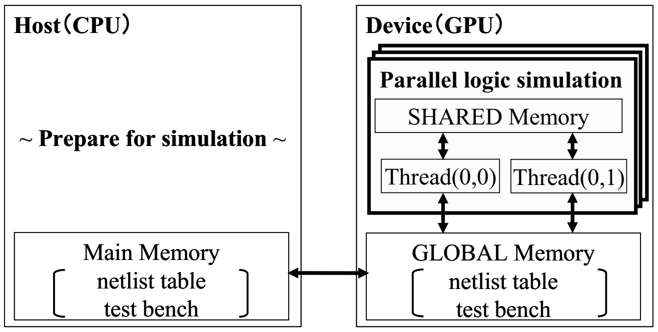


Fig.1 Overall configuration of a *CUDA* program module

Fig.1 shows the overview of *CUDA* program module configuration with *HOST* module and *DEVICE* module with *GLOBAL* and *SHARED* memories.

Input data of *CUDA* program is firstly loaded into *HOST* module, then *HOST* transferred them to *Global* memory in *DEVICE* module when it needs to be executed in parallel. If *DEVICE* module needs to refer the data in *HOST* module, then *HOST* module transfer them to *GLOBAL* memory firstly and *DEVICE* module copy them to *SHARED* memory secondly.

THREAD can access to *SHARED* memory faster and to *GLOBAL* memory slower, while *SHARED* memory capacity is as small as 48KB and *GLOBAL* memory is as large as 1.5GB in case of Geforce GTX480.

Thus, a key to speed up execution in *CUDA* environment is the proper utilization of *SHARED* and *GLOBAL* memories.

B. A Basic Levelized Simulation Method

A simple and basic logic simulation method based on a logic levelized model is illustrated in Fig.2. Here, squares indicate logic gates, the thick arrow is a logical operation order along the logic levels.

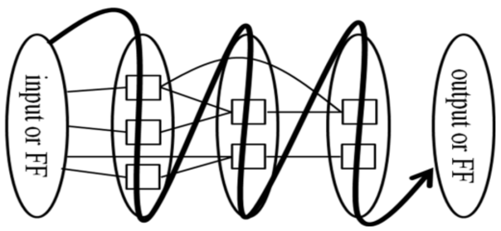


Fig.2 A basic levelized simulation method

Here, a logic circuit is divided into several levels, according to the logic signal propagation sequences from the primary inputs to the primary outputs of the circuit. The primary inputs are assigned to the first level, and sequentially the next logic gates assigned to the next level respectively and finally the primary outputs assigned to the final level.

The logic simulation results of the circuit are obtained by evaluating all logic gates on all levels, i.e. evaluating logic gates on the first level to the final level. If all test patterns are evaluated, then the logic simulation is completed.

The basic algorithm of the simulation is shown in Fig.3.

The states of gate instances are stored in *SHARED* memory of the GPU and the probe points assigned to instances by the designer are transferred and saved in *GLOBAL* memory in YAPSIM. And then those states of probe points are transferred to the host CPU and to be displayed to the circuit designer. However, *GLOBAL* memory access and the state value transfer should be reduced to speed up the simulation of a large logic circuit, because it requires a longer time. So, software simulators which are flexible for circuit debugs are recommended instead of a GPU simulator like YAPSIM, which is more suitable for large scale circuits or systems.

III. High-speed parallel logic simulator: YAPSIM

In this section, the three key improvements of YAPSIM for faster logic simulation are described.

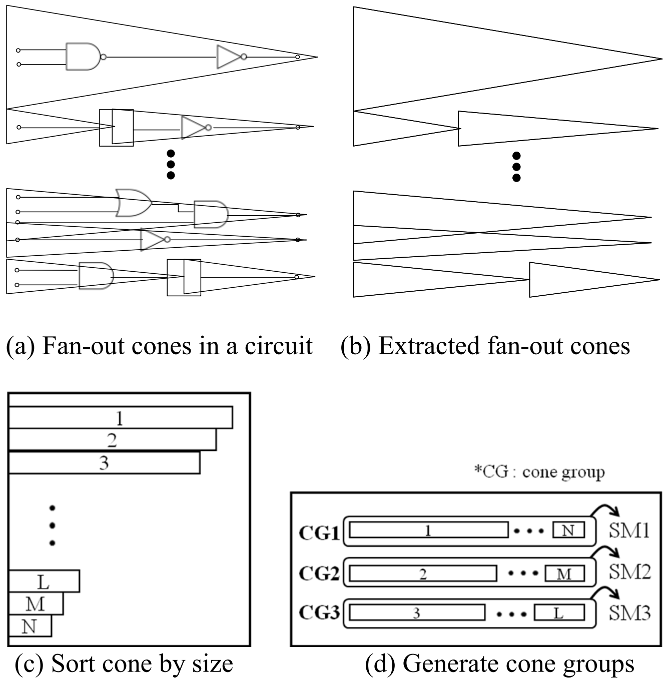
A. Fan-out Cone Grouping and Scalable SM usage

YAPSIM is implemented based on the simulation method in Fig.3, though it cannot afford the larger logic circuit because of the number of cores and *SHARED* memory in a single SM is restricted. It cannot afford when the number of gates on a level of the circuit becomes more than the available number of *THREAD*s of a *BLOCK*

We introduce the fan-out grouping method to YAPSIM that can handle multiple SMs. A fan-out cone is a group of logic gates determining the value of a primary output or an input port of a flipflop.

```
Step0(HOST): Calculate the number of levels from the primary
    inputs to the primary outputs of the circuit net-list.
Step1(HOST): Transfer the net-list to DEVICE
Step2(HOST): Transfer test vectors to DEVICE
Step3(HOST): Call DEVICE module
Step4(DEVICE): Execute logic simulation
        for all test vectors {
      for i=1 to number of levels
                execute logic gates in i-th level in parallel
                }
        }
Step5 (DEVICE): Transfer logic simulation outputs to HOST
```

Fig.3 A basic levelized simulation algorithm



(a) Fan-out cones in a circuit (b) Extracted fan-out cones

(c) Sort cone by size (d) Generate cone groups

Fig.4 Overview of the cone group generation

Fig.4 (a), (b), (c) and (d) illustrate the fan-out cone grouping procedures. Fig.4 (a) shows the fan-out cones in a circuit, determining the primary output values, and Fig.4 (b) illustrates extracted fan-out cones from the circuit.

As the gate counts of fan-out cones are different, the fan-out cones are sorted by the gate count in descending order as shown in Fig. 4 (c). Finally the logic gate number balanced groups as shown in Fig.4 (d) are generated by using the order in Fig.4 (c). The algorithm is shown in Fig.5.

STEP1. Extract fan-out cones of the logic circuit.
STEP2. Sort the fan-out cones by their number of gates in descending order.
STEP3. Assign the fan-out cones to groups in descending order evenly.

Fig.5 Cone Group Generation Algorithm

YAPSIM is added further extension by using multiple GP-GPUs to make YAPSIM scalable for the gigantic gate size circuits.

B. High Speed Logic Evaluation using LUT

The gate evaluations, which are done corresponding to the various gate types are carried out by using the conditional statements such as, "if", "case" or "switch" statements. For example, when a case statement is executed on the GPU, all processes for all conditions are executed and then the processing time of a case statement requires longer than that a general CPU. To minimize the processing time, the number of branches is reduced as shown Fig. 6.

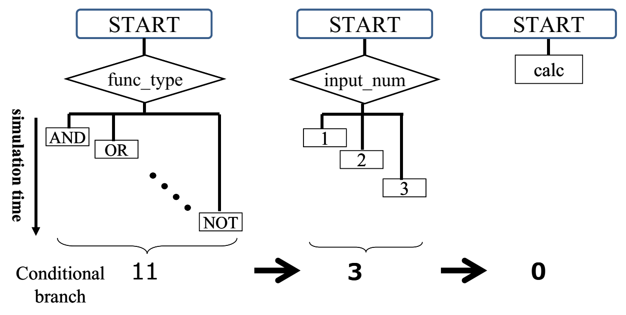


Fig.6 Reduction of conditional branches

For higher speed gate evaluation as the further improvement of YAPSIM, a LUT method for logic gate evaluation is employed. Each LUT_index which is the concatenation of type (3 bits), i1 (1 bit), and i2 (1 bit) is 5 bit code. The output ("out" in the table) value of the gate is determined by referencing LUT using the index as shown in Fig. 7. LUT_index is generated by the concatenation as expressed in Eq. 1.

LUT_index = {function_type, input1, input2}, (Eq. 1)

The logic gate evaluation can be done as follows. First calculate LUT_index in Eq.1 by using a function type number and its inputs, then access data in LUT by LUT_index. This direct access memory makes the logic gate evaluation very fast. An example of the LUT is shown in Fig.7.

	index			out
	type	i1	i2	
	000	0	0	0
AND	000	0	1	0
(000)	000	1	0	0
	000	1	1	1
	001	0	0	0
OR	001	0	1	1
(001)	001	1	0	1
	001	1	1	1
	010	0	-	1
NOT	010	0	-	1
(010)	010	1	-	0
	010	1	-	0
	011	0	0	0
XOR	011	0	1	1
(011)	011	1	0	1
	011	1	1	0

Fig.7 Logic Evaluation LUT

C. Continuous memory access improvement

The high usage data in calculation in the structure data of the netlist are transferred into the array type memory.

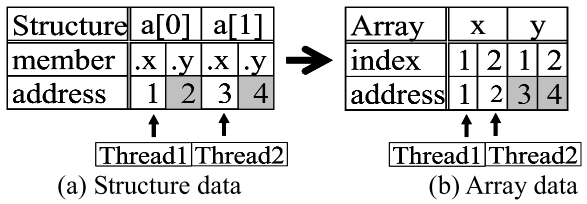


Fig.8 Cone Group Generation Algorithm

inst#	level
1	3
2	4
3	2
4	1
5	6
6	2
7	5
8	7
9	1
10	2

(a) Original table

inst#	level
1	1
2	1
3	2
4	2
5	2
6	3
7	4
8	5
9	6
10	7

(b) Sorted table

Fig.9 Sorting netlist table

Fig. 8 shows an example of netlist data access which is placed continuously in an array. The array is generated by sorting the gate instance table of netlist by the level in Fig. 8 (a). The other example shows the replacement of the distributed data items in the structured table with the continuous data, which is in the order of thread ID of GPU, in an array in Fig. 8 (b). This continuous data arrangement realizes faster memory access drastically.

IV. Performance Evaluation

We compare the three simulators, i.e. they are YAPSIM with three GTX480, a levelized model based sequential logic simulator SeqSim without a GPU engine and a commercial logic simulator C-SIM which is a high speed simulator based on the event driven method.

A. Experimental Environment

The experimental platforms are as follows.

1) Desktop platform: Windows 7 and Intel Core i7-950 CPU with the clock 3.07GHz
The main memory is 3GB with the clock 3.07GHz.
2) *CUDA* platform: *CUDA* 5.0.
3) GPU engine: Three boards of Geforce GTX480 which has 480 cores with the clock 1.4GHz.
The global memory is 1.5GB with the clock 1.8GHz.

YAPSIM execution time on GP-GPU is obtained by the NVIDIA *CUDA* Compute Visual Profiler, while the SeqSim and C-SIM execution times are obtained as the elapse time of the Desktop.

Table I Evaluation Circuits

Circuit	#Input	#Output	#FF	#Gate	#Logic Level
4bit-adder x 640	9	3200	0	12800	9
LDPC encoder	1723	2048	0	75035	12
CPU01	19	18	173	2111	56
CPU20	19	360	3460	42220	56
CPU40	19	720	6920	84440	56

NOTE: # is "number of "

The evaluation circuits are adder4x640 including 640x 4-bit-adders and LDPC encoder [8] as combinational circuits, and the processors of CPU01, CPU20 and CPU40 with single, twenty and forty CPU cores as sequential circuits. Table I shows the overview of evaluation circuits. The test vectors for the two combinational circuits are generated as random number sequences and the test vector length is 100,000 cycles. The test vector for sequential circuits is a set of CPU instructions consists of load, store, add and other operations with the length of 100,000 cycles.

B. Experimental Result

The speed comparison results of the execution times and the speed ratios of C-Sim, SeqSim and YAPSIM are shown in Table II. In Table II, YAPSIM is up to 29 times faster than that of C-Sim for the combinational circuit, LDPC encoder. YAPSIM is up to 5.67 times faster than that of C-Sim for the sequential circuit, CPU40.

The comparison between YAPSIM and SeqSim, YAPSIM is up to 25.3 times faster than that of SeqSim for the combinational circuit, LDPC encoder. YAPSIM is up to 7.5 times faster than that of SeqSim for the sequential circuit, CPU40.

Consequently, experimental results show that YAPSIM based on parallel algorithms have been able to take advantage of the performance of the GPU is efficient in a larger circuit.

Table II Performance Comparison

Circuit	Execution time [sec]			Ratio	
	C-Sim	SeqSim	YAPSIM	S / Y	C / Y
4bit-adder x 640	15.30	9.2	1.24	7.42	12.34
LDPC encoder	51.80	44.8	1.77	25.31	29.27
CPU01	1.20	1.22	3.29	0.37	0.36
CPU20	20.10	24.97	5.45	4.58	3.69
CPU40	38.30	50.62	6.75	7.50	5.67

NOTE: S/Y is "SeqSim/YAPSIM", C/Y is "C-Sim/YAPSIM"

V. Conclusion

We propose YAPSIM parallel logic simulator using GP-GPU. YAPSIM is based on a logic simulation method with allocating logic gates to GP-GPU threads individually. Furthermore, it consists of three improvement methods, a fan-out cone grouping method for effective use of available SMs, a LUT method and a GPU internal memory access method. The experimental results of the comparisons between a high speed commercial simulator (C-Sim) and YAPSIM implementation shows that YAPSIM executed 29 times faster than that by the high speed commercial simulator for a combinational circuit (LDPC encoder) of 75,000 gates, and 5.7 times faster for a sequential circuit (CPU40) of 84,000 gates respectively.

Further evaluation of the practical large-scale circuits is the future work.

Acknowledgement

This work is supported by VLSI Design and Education Center (VDEC), The University of Tokyo with the collaboration with Mentor Graphics Corporation.

References

[1] Debapriya Chatterjee, Andrew DeOrio, Valeria Bertacco, "Event-Driven Gate-Level Simulation with GP-GPUs," In Proc. of ACM/IEEE DAC, pp. 557-562, 2009.
[2] Debapriya Chatterjee, Andrew DeOrio, Valeria Bertacco, "GCS: High Performance Gate Level Simulation with GPGPUs," In Proc. of ACM/IEEE DATE, pp.1332-1337, 2009.
[3] Alper Sen, Baris Aksanli, Murat Bozkurt, Melih Mert, "Parallel Cycle Based Logic Simulation using Graphics Processing Unit," In Proc. of IEEE ISPDC, pp. 71-78, 2010.
[4] Bo Wang, Yuhao Zhu, Yangdong Deng, "Distributed Time, Conservative Parallel Logic Simulation on GPUs," Proceedings of the 47th Design Automation Conference, pp. 761-766, June 2010.
[5] Sara Vinco, Debapriya Chatterjee, "SAGA: SystemC Acceleration on GPU Architectures," Proceedings of the 49th Design Automation Conference, pp. 115 - 120, June 2012.
[6] Rohit Sinha, Aayush Prakash, and Hiren D. Patel, "Parallel Simulation of Mixed-abstraction SystemC Models on GPUs and Multicore CPUs," Proceedings of ASP-DAC2012, pp. 455-460, January 2012
[7] "NVIDIA CUDA Compute Unified Device Architecture," http://www.nvidia.com/object/cuda_home_new.html
[8] Open Cores. htpp://www.opencores.org