GPGPU による組み合わせ論理回路の入力パタン並列論理シミュレーション

森　　裕紀
平石　裕実

（平成 24 年 9 月 21 日提出
平成 24 年 11 月 30 日修正
平成 24 年 12 月 17 日再修正）

要　旨

大規模論理回路の設計検証やテスト生成において，大規模組み合わせ論理回路の論理シミュレーションが必要となる．これを高速化するために，数百の演算ユニット（コア）を利用できる GPGPU 技術を用いて大量の入力パタンを並列に処理する論理シミュレーション手法を提案する．ISCAS のベンチマーク回路の論理シミュレーションで提案手法により約 12〜18 倍高速化することができた．

キーワード：GPGPU，論理シミュレーション，組み合わせ論理回路，並列処理，論理設計

1. はじめに

半導体技術の発展に伴い，大規模論理回路が開発されているが，その設計の正しさを確かめる設計検証や，回路の故障を発見するためのテスト入力パタンの生成（テスト生成）に，論理シミュレーションが用いられている．回路の大規模化に伴い，論理シミュレーションにかかる時間も増大し，その高速化が望まれている．一方で，シミュレーションを行うコンピュータの性能も向上してきており，特に，グラフィック処理を行うGPU (Graphics Processing Units) では，数百のコア（演算ユニット）を用いた並列処理が普通に行われるようになってきている．近年，GPU を汎用的な計算にも利用しやすいように機能が追加され，開発環境も整備されつつあり，多くの科学技術計算の分野でGPGPU (General Purpose computing on Graphics Processing Units) が用いられるようになってきている．

このような背景のもと，本研究では論理シミュレーションの高速化を目的とし，GPGPU による論理シミュレータの開発を行った．GPGPU を用いた論理シミュレーションとして，対象の回路を複数の部分回路に分割し，各部分回路に一つのスレッドを割り当て，これらのスレッドを GPGPU 上で並列に実行する方式が提案されている[1–3]．これらの方式では，順序回路のシミュレーションも可能であるが，次のような問題点がある．

① 順序回路のクロックサイクル毎に出力と次状態を計算する必要があるため，一つの入力パタンに対して出力と次状態を求めてからでないと次の入力パタンに対するシミュレーションが出来ない．対象が組み合わせ回路であっても，順序回路の場合と同じアルゴリズムが用いられるので，入力パタン毎に回路の出力を計算することになり，複数の入力パタンに対する出力を並列に求めることは出来ない．

② 一つのスレッドが担当する部分回路は，入力端子からのレベルごとに 1 ゲートまたは複数のゲートからなるマクロゲートになっている．例えば，図 2 において，スレッド 1 が G11, G21, G31 の評価を行い，スレッド 2 が G12, G22, G32 の評価を担当する．一般に，レベル $i+1$ のゲートの評価を行うためにはレベルiのゲートの評価が終了している必要があるため，レベル毎に各スレッドのバリア同期が必要となる．

③ GPGPU では，最大で 512 個のスレッドが一つのブロックとして並列に実行される．512 個よりも多くのスレッドが必要な場合は，複数のブロックを並列に実行することになるが，異なるブロックに属するスレッド間ではバリア同期を行うことが出来ない．このため，複数ブロックを用いて実行する場合，部分回路のコピーを作成する必要がある．例えば，図 1 において，出力 A と出力 B の値を求めるのに，入力付近で，700 スレッド必要であるとすると，二つのブロックに分けて実行する必要がある．この時，出力A を計算する部分回路と出力 B を計算する部分回路の共通部分はコピーを作成して両方のブロックで計算する必要がある．

④ 一般に，出力付近の計算では，必要なスレッド数が少なくなるため並列度が低下する．

そこで，我々は，対象を組み合わせ論理回路に限定することにより，複数の入力パタンに対して並列にシミュレーションすることを可能とし，上記 ① の問題点を解決している．また，多くの入力パタンに対する論理シミュレーションを行うことを想定し，一つのスレッドで回路全体をシミュレーションすることにし，入力パタン数に応じて多数の同一スレッドを並列に実行

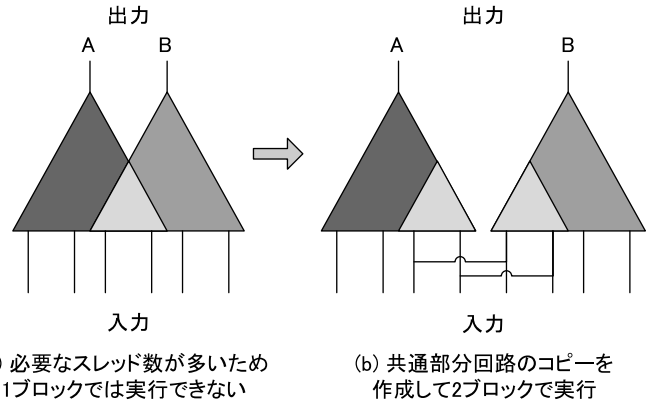

図 1　複数ブロックでのシミュレーション

する．これらのスレッドは互いに独立して実行することが可能なため，上記 ② のようなスレッドのバリア同期は不要となる．また，一つのスレッドで回路全体をシミュレートするため，上記 ③ のような部分回路のコピーは不要であり，並列に実行するスレッド数は並列にシミュレートする入力パタン数に依存するため，上記 ④ のような問題点も発生しない．我々の提案手法は SIMD (Single Instruction Multiple Data) 型の並列処理になっており，効率向上が期待できる．一方，我々の提案手法では，順序回路のシミュレーションは困難であるが，大規模論理回路はスキャン設計を用いることが多い[4] ため，組み合わせ回路の論理シミュレーションに対する需要も高いと考えられる．実際に提案手法をGPGPUを用いて実装し，ISCAS'85 ベンチマーク回路 [5] の論理シミュレーションを行った結果，約 12〜18 倍の高速化を実現できた．

以下，第 2 章では本研究で使用した NVIDIA 社の GPU Tesla M2090 [6, 8] の概要と GPU の統合開発環境 CUDA (Computer Unified Device Architecture) [7, 9] の概要を説明する．第 3 章では，並列論理シミュレーションアルゴリズムを提案する．第 4 章では，ISCAS のベンチマーク回路の論理シミュレーションに提案手法を適用した結果を示し，第 5 章でまとめを行う．

2. GPU と CUDA

本研究で用いた GPU Tesla M2090 は，Fermi コア・アーキテクチャ [8] を採用しており，32 個のコアを持つストリーミング・マルチプロセッサ 16 個で構成されており，GPU 当たり合計 512 個 (32×16) のコアを持っている．また，この GPU は 6 GB のデバイスメモリを持っており，ホスト・コンピュータとは，PCI-Express x16 で接続されている．GPU では，32 個のスレッドを warp と呼び，warp 単位でストリーミング・プロセッサに割り当てて実行される．

CUDA は NVIDIA 社の GPU を用いて汎用的なプログラムを動かすためのプラットフォーム・統合開発環境である[9]．GPGPU では，CPU と GPU の両方を用いて計算を行う．CPU は「ホスト」と呼ばれ，GPU は「デバイス」と呼ばれる．ホストで実行されるプログラムは「ホストプログラム」，GPU に実行させるプログラムは「カーネル（プログラム）」と呼ばれる．これらは，互いに協調して動作する．ホストプログラムが開始されると，まずデバイスにカーネルプログラムがロードされる．次に，ホストで必要なデータを準備してから，そのデータをデバイスに転送してカーネルプログラムを起動する．このカーネルプログラムにより，デバイス側でデータ処理が行われ，処理結果がデバイスメモリに格納されるので，それをホスト側に転送する．

この CUDA プログラミングにおいて注意する点は，メモリ転送である．カーネルプログラムを起動する前にメモリ上のデータをホスト側からデバイス側に転送する必要がある．また，カーネルプログラムの終了後，結果をホスト側に転送する．処理の前後にメモリ転送の時間がかかるので，その時間をGPU の並列処理により短縮できるだけの効率が必要である．

CUDA プログラミングモデルでは，「スレッド」，「ブロック」，「グリッド」という概念がある．

ブロックは，複数のスレッドからなり，グリッドは複数のブロックからなる．1 つのブロックには最大 512 スレッドを格納することが出来る．また，1 つのグリッドには最大 65535 × 65535 のブロックを配置することが出来る．

このプログラミングモデルにおいて，スレッドは GPU の一つのコアで実行され，ブロックは GPU の一つのストリーミング・マルチプロセッサで実行される．カーネルプログラムはグリッドとして起動される．

3. 入力パタン並列論理シミュレーション

まず，論理シミュレーションを行う回路のネットリストを用いて，回路内の各ゲートの入力側からのレベルを求める（図 2 参照）．入力端子のレベルを 0 とし，各ゲートのレベルは，そのゲートの入力信号を生成しているゲートのレベルの最大値+1 と定義する．これにより，レベル i のゲートの出力の値が求まれば，レベル $i+1$ のゲートの出力を計算することが出来る．入力端子に与えられる入力信号から，レベルの昇順に各ゲートの出力を計算していくことにより，すべての出力端子の信号値を求めることができる．この操作を一つの「スレッド」で実現する．図

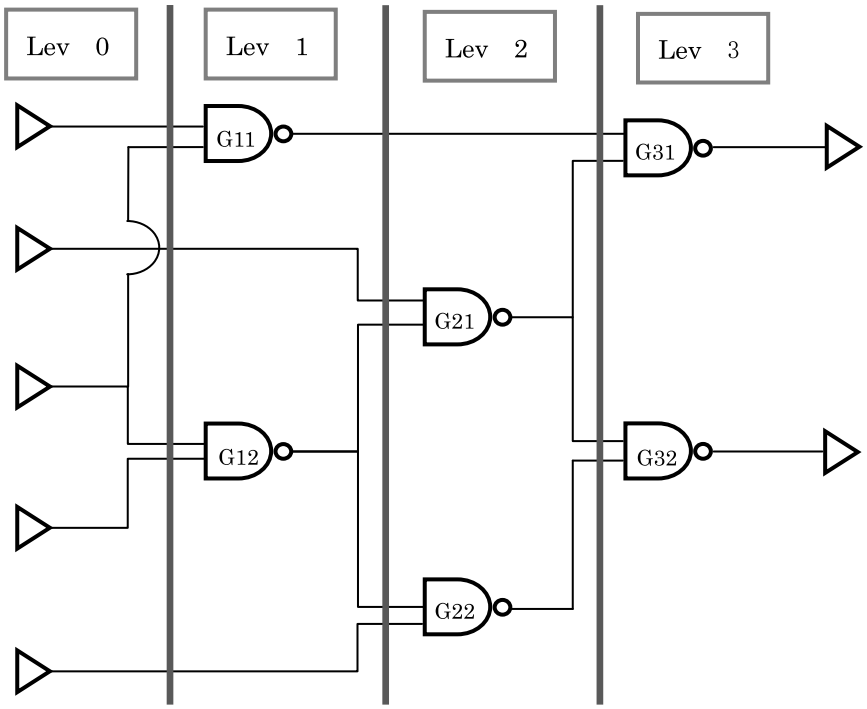

図 2　各ゲートのレベル

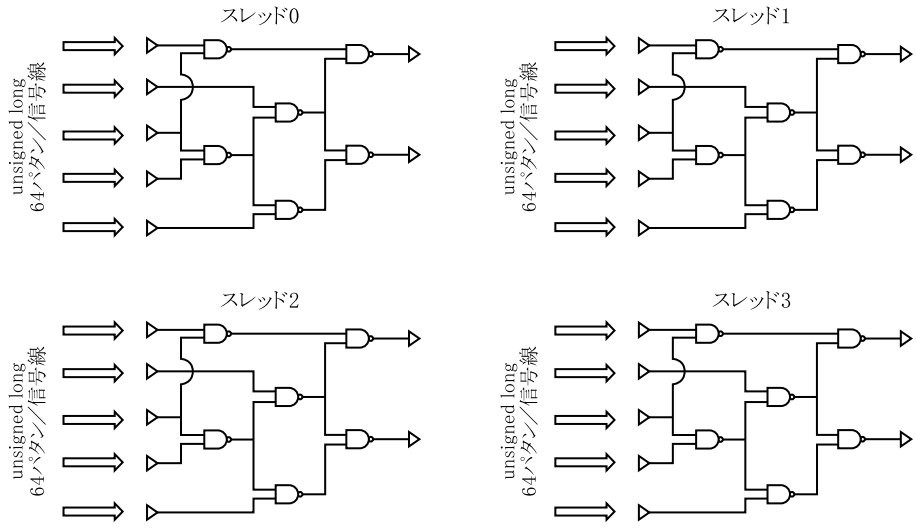

図 3　4スレッドを用いた256パタン並列シミュレーション

2 の例では，一つのスレッドが G11, G12, G21, G22, G31, G32 の順にゲートの出力を計算する．

スレッドで計算する信号値は，64 bit の符号なし整数の各ビットに一つの 0/1 パタンを割り当てることで 64 パタンを表現する．GPU の 64 bit 論理演算命令を使って各ゲートの出力を順に計算することにより，1 スレッドで $64 = 2^6$ 個のパタンを並列に計算することが出来る．したがって，例えば図 3 に示すように，4 スレッド並列に実行すると，$64 \times 4 = 256$ パタンを並列に計算することが出来る．

1 ブロックには最大 512 個のスレッドを格納することが出来るので，1 ブロックで，$64 \times 512 = 32,768 = 2^{15}$ 個のパタンを並列にシミュレーションできる．これより多くの入力パタンに対してシミュレーションする場合は，複数のブロックを用いたグリッドを構成することで対応する．

4. 実験結果

ISCAS'85 のベンチマーク回路 [5] に対して 2^{24} 個の入力パタンに対する論理シミュレーションを行った結果を表 1 に示す．

表 1 において，「入力数」欄は回路の入力信号数（ビット），「出力数」欄は回路の出力信号数（ビット），「ゲート数」欄は回路を構成するゲートの数を表している．CPU 欄は，ホスト・コンピュータ (Xeon X5675, 3.07 GHz) で逐次型プログラムにより 2^{24} 個の入力パタンに対する論理シミュレーションを行うのに要した時間（秒）である．この逐次型プログラムでは，GPU のスレッドと同様に 64 bit 符号なし整数の各ビットに一つの 0/1 パタンを割り当てることで 64 パ

表 1　実験結果

回路名	入力数	出力数	ゲート数	CPU	GPU	比率
c432	36	7	160	3.56	0.28	12.7
c880	60	26	383	9.71	0.65	14.9
c1355	41	32	546	14.69	0.93	15.8
c2670	233	140	1193	25.26	2.05	12.4
c3540	50	22	1669	35.98	2.72	13.2
c5315	178	123	2307	52.24	4.07	12.9
c6288	32	32	2416	73.78	4.17	17.7
c7552	207	108	3512	80.79	5.29	15.3

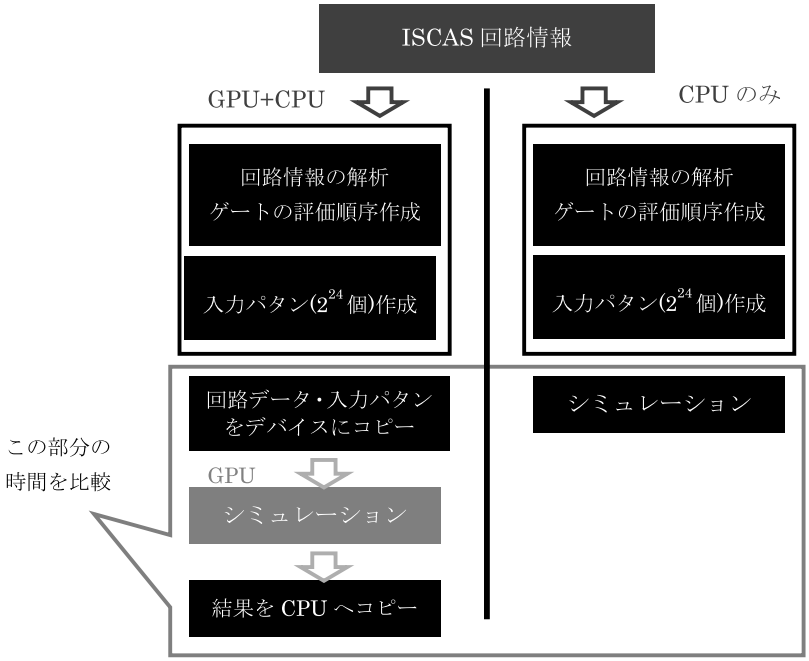

図 4　GPUとCPUの比較

タンを表現し，64 bit 論理演算命令を用いて各ゲートの出力を順に計算することにより 64 個のパタンを並列に計算している．GPU 欄は，提案手法により 2^{24} 個の入力パタンに対する論理シミュレーションを行うのに要した時間（秒）である．回路データと入力パタンを GPU に転送したり，シミュレーション結果を CPU に転送する時間も含んでいる．なお，CPU 欄，GPU 欄の

時間は，回路データを解析してゲートの評価順序を決めたり入力パタンを作成する部分の時間は含んでいない（図 4 参照）．比率は両者の比を表している．これより，12.4〜17.7 倍提案手法の方が高速であることがわかる．

まとめ

本論文では，組み合わせ回路を対象とし，複数の入力パタンに対する論理シミュレーションをGPGPU を用いて並列に行う方法を提案した．実際にこの手法を NVIDIA Tesla M2090 を用いて実装し，ISCAS'85 のベンチマーク回路に適用した結果，約 12〜18 倍高速化できることを示した．提案手法では，回路全体のシミュレーションを 1 スレッドで実現しているため，回路規模がより大きくなると，GPU のメモリが不足する可能性がある．このため，今後より大規模な回路に適用し本手法の限界をチェックする必要がある．また，文献 [1–3] に示された手法との性能比較も必要である．さらに，近年コア数が増えつつあるマルチコア CPU での並列シミュレーションとの比較も今後の課題としたい．

参　考　文　献

[1] Zhang Yuxuan, Wei Tingcun, Kai Yaowen, Fan Xiaoya, Zhang Meng, and Zhao Lili, "Logic Simulation Acceleration Based on GPU", MIXDES 2011, 18th International Conference "Mixed Design of Integrated Circuits and Systems", June 16–18, 2011, Gliwice, Poland

[2] Yuhao Zhu, Bo Wang and Yangdong Deng, "Massively Parallel Logic Simulation with GPUs", ACM Trans. Design Automation of Electronic Systems, Vol. 16, No. 3, Article 29, June 2011.

[3] Alper Sen, Baris Aksanli, Murat Bozkurt, and Melih Mert, "Parallel Cycle Based Logic Simulation using Graphics Processing Units", 9th International Symposium on Parallel and Distributed Computing, 2010.

[4] 畠山，"設計者に必要なテスト容易化設計の基礎知識"，デザインウェーブマガジン，pp. 47–54, March 2011.

[5] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinatorial benchmark circuits and a target translator in FORTRAN", In Int. Symposium on Circuits and Systems, Special Session on ATPG and Fault Simulation, 1985

[6] NVIDIA,"サーバ用 TESLA GPU コンピューティングソリューション"，http://www.nvidia.co.jp/object/tesla-servers-jp.html

[7] NVIDIA，"CUDA 並列コンピューティングプラットフォーム"，http://www.nvidia.co.jp/object/cuda-parallel-computing-platform-jp.html

[8] G-DEP，"GPU の構造"，http://www.gdep.jp/page/view/252

[9] G-DEP，"CUDA プログラミングモデル"，http://www.gdep.jp/page/view/253

Input Pattern Parallel Logic Simulation of Combinatorial Circuits Using GP-GPU

Hiroki MORI
Hiromi HIRAISHI

Abstract

In the area of logic verification and test generation for large logic circuits, logic simulations of large combinatorial circuits become essential task. In this paper, we propose a new logic simulation method that processes large number of input patterns in parallel by using GP-GPU approach, where we can use, in general, hundreds of processing units (cores). We applied our method to logic simulation of ISCAS benchmark circuits, and obtained around 12 to 18 times speed-up.

Keywords: GP-GPU, Logic Simulation, Combinatorial Circuits, Parallel Processing, Logic Design