Vol.2015-SLDM-169 No.12
2015/1/29

GP-GPUを用いた高速並列論理シミュレータ

橋口 拓哉[†1]　青野　寛之[†2]　豊永 昌彦[†1]　村岡 道明[†1]
[†1]高知大学大学院　理学専攻　情報科学分野
[†2]高知大学　理学部　情報科学分野
〒780-8520 高知県高知市曙町 2-5-1
{hashiguc, toyonaga, muraoka}@is.kochi-u.ac.jp

概要　本論文は，GPGPU を使った高速論理シミュレーションの高速化手法について述べる．本シミュレーション高速化手法の構成は，論理素子を GP-GPU スレッドを割り付ける並列論理シミュレーション法を基本アルゴリズムとし，GPU 内部メモリアクセス高速法，条件分岐削減法および SM 演算時間平坦法である．商用論理シミュレータと本シミュレータのプロトタイプの比較実験の結果，7.5 万ゲートの組合せ回路で 29 倍，および，8 万ゲートの順序回路で 5.7 倍の高速性を示すことができた．

A High-Speed Parallel Logic Simulater Using GP-GPU

Takuya Hashiguchi, Masahiko Toyonaga, Michiaki Muraoka
Graduate School of Science, Kochi University
2-5-1 Akebono-cho, Kochi, 780-8520, Japan

Abstract: In this paper, a new high-speed logic simulator based on a parallel logic simulation methodology using GP-GPU is presented. It is based on a fan-out cone grouping method, and consists of three acceleration methods for simulating performance, a GPU internal memory access method, a branch reduction method and an SM execution time balancing method. The experimental comparison result shows that the proposed simulator 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.

1　はじめに

近年，システムの大規模化や半導体の微細化技術の進歩により，設計の規模や複雑性が急激に増加しており，その検証がますます重要になってきている．大規模な設計のハードウェア / ソフトウェア協調検証には膨大な時間がかかるため一般的にエミュレータが用いられている．

しかし，大規模回路ではエミュレータは書き換え（再構成）に長時間を要し，デバック性が低いため，ハードウェア部の高速なシミュレーションが必要である．エミュレータに替わる高速回路検証法として，並列論理シミュレーション法[1][2][3][9][10]やシステムレベルの並列シミュレーション法[3][4]が報告されている．これらの手法は，並列処理機構としてプログラミング環境 CUDA の GP-GPU(General Purpose Graphic Processing Unit)を利用している．

[1,2,3]の文献はイベントドリブン法に GP-GPU を利用した論理シミュレーションである．[1]の文献は，論理回路をロジックコーンに分割して GP-GPU を利用した並列シミュレーションである．[2]の文献は，論理回路を論理マクロに分割して GP-GPU を利用する並列シミュレーションしている．[3]の文献は，論理回路を論理ゲート毎に GP-GPU スレッドに割り付けて並列化している．

本論文は，論理回路シミュレーションや GPU の特徴を十分考慮し高速化を行ったのでその手法について述べる．

まず基本アルゴリズムとして，論理シミュレーションを単純化するために論理回路のレベル（論理段）付けを行い，レベル上の各論理ゲートを GP-GPU スレッドに割り付けて，論理レベル上の論理ゲートを並列演算させる論理シミュレーション法がある．

第 1 に，GPGPU の内部構造の分析から，論理回路情報を高速にメモリアクセス可能なデータ構造に変換を行う．

第 2 に，処理の分岐を削減し，高速化した論理回路シミュレーションを構成している．

第 3 に，大規模回路のシミュレーションを高速に行うためにファンアウトコーンを用い論理回路をコーンに分割し，コーンの処理時間が均等になるようグループ化を行っている．

商用論理シミュレータと本並列シミュレータのプロトタイプの比較実験の結果，7.5 万ゲートの組合せ回路を 29 倍高速化，さらに 8 万ゲートの順序回路を 5.7 倍高速化することができた．

2 GPU 環境と並列論理シミュレーション

2.1 CUDA / GP-GPU 環境

CUDA は NVIDIA 社の GP-GPU を使う並列プログラミング環境である[11].

CUDA プログラミングは, 2 階層のプログラムで構成されている． 1 つは CPU 側で実行するプログラムで，HOST と呼ばれる．もう 1 つは，GP-GPU 側で実行するプログラムで，DEVICE と呼ばれる．DEVICE は，多数の SM (Streaming Multiprocessor)を持ち，全 SM 共有の 1GB 程の大局メモリ（Global メモリ）で構成されている. 各 SM は内部に SM 内でのみ共有できるシェアードメモリ（Shared メモリ）を持っている．図 1 は, HOST と DEVICE を含む全体構成を示す．並列処理の入力データは，HOST から DEVICE の Global メモリに転送される．HOST から各 Shared メモリへデータを転送するには, まず HOST から Global メモリにデータを転送し，その後，Global メモリのデータを Shared メモリへ転送する．

CUDA プログラム環境おける最小の処理単位は，Thread である. 同時に実行される Thread 数は GPU のプロセッサ数と CUDA 仕様に依存する. ブロックは，幾つかの Thread の集合である．ブロックは，自動的に各 SM に割り付ける．

GP-GPU で処理を高速化するには, Shared メモリと Global メモリの使い分けがカギとなる. なぜならば，Shared メモリはアクセス時間が短く，容量が小さい (Geforce GTX480 などで 48KB)，その反面，Global メモリは容量が大きく，アクセス時間が長い．

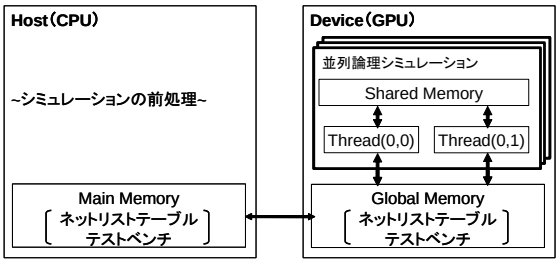


図 1 Host (CPU) と Device (GPU)のメモリ構成

2.2 レベルソート法による基本シミュレーション

先行研究[6][7][8]ではレベルソート法に基づき並列論理シミュレーションアルゴリズムを提案した. 図 2, 3 にレベルソート法の処理方法および並列論理シミュレーションの処理手順を示す.

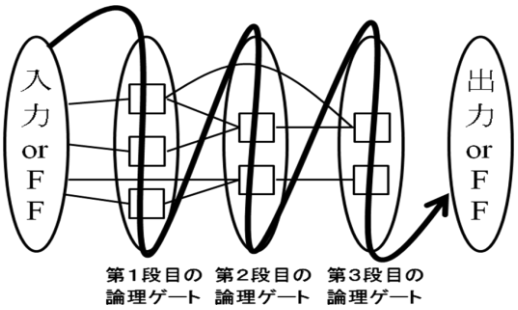


図 2 レベルソート法による論理シミュレーション

図 2 では、四角は論理ゲートを示し，太線は論理回路の演算順序を示す．レベルソート法において、外部入力値を最初の論理段(level)の入力として，論理段内の論理ゲートの出力の計算を繰り返す．そうして，外部出力の値が得られる．すべてのテストパターン長に対して，この処理を繰り返してシミュレーションが完了する．

図 3 は，レベルソート法を使った並列論理シミュレーションアルゴリズムである．

Step0: 論理回路における外部入力端子 FF から外部出力端子 FF までの段数計算
Step1 ネットリストの並列処理用変換と DEVICE への転送
Step2 テストベクタの DEVICE への転送
Step3 DEVICE の処理(カーネル関数)の呼び出し
Step4 並列論理回路シミュレーション(DEVICE)
　Step4.1 テストベクタ長分繰返したなら Step5 へ.
　Step4.2 段数分繰返したら Step4.1 へ.
　Step4.2.1 1 サイクルのテストパターンの外部入力端子設定.
　Step4.2.2 同一レベルの論理ゲートの並列演算し, Step4.2 へ.
Step5 回路出力を HOST への転送

図 3 レベルソート法による並列シミュレーション

手順

3 高速化手法

並列化アルゴリズムの高速化手法として３つの手法を提案しプロトタイプを改良した. 高速化前を Ver. 1 とし, 各高速化手法に対応したバージョンを Ver. N と示す.

3.1 メモリアクセスの高速化

メモリアクセスの高速化を行うため，以下の２方法でネットリスデータの連続化を行い，GPU 内部のキャッシュのヒット率向上を図った.

i) 構造体を配列に置換し連続アクセスを行う.

図 4 の左部分が構造体，右部分が配列のメモリ配置を表わす．構造体ではメンバ毎に離散的にメモリ領域が割り当てられるため，データアクセスに時間がかかるが，配列を使用すると連続した領域にアクセスすることになりキャッシュのヒット率が向上し，アクセス速度の高速化が期待できる．(ver. 2)

評価回路として 4bit の Adder を並列に 640 個並べた回路（adder4x640）で ver.2 と高速化前のシミュレータ ver.1 で処理時間を比較すると約 1.5 倍（ver.1：35.7sec → ver.2：22.7sec）の高速化が確認できた．また，GPU 内部のキャッシュのヒット率を CUDA の提供する visual profiler を用いて計測したところ ver.1 が約 50%であったのに対し，ver.2 では約 69%とヒット率が向上しているのを確認できた.

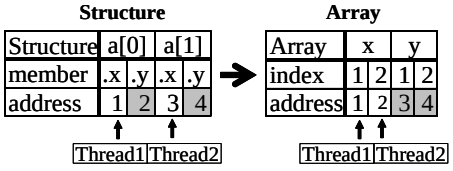


図 4 構造体の配列への置換

ii) ネットリストテーブルをレベル順にソートし論理ゲートの連続アクセス速度を行う.

図 5 はネットリストテーブルのソートを図示したもので，左部分がソート前，右部分がソート後のネットリストテーブルを表している.

ソート前のテーブルではインスタンスの論理段(レベル) が順番になっていないのに対し，ソート後のテーブルではインスタンスがレベル順に整列するため，キャッシュのヒット率が向上し，アクセス速度の高速化が期待できる．(ver.3)

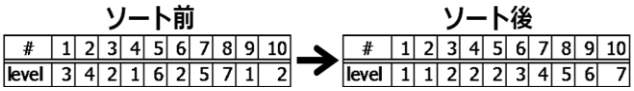


図 5 レベル順にソート

評価回路 adder4x640 で ver.1 と ver.3 で処理時間を比較したところ約 6.3 倍（ver.1：35.7sec → ver.3：5.64sec）の高速化が確認できた. visual profiler を用いてキャッシュのヒット率を計測したところ ver.1 が約 50%, ver.2 が約 69%であったのに対し，ver.2 では約 80%とヒット率が向上しているのを確認できた.

3.2 条件分岐削減による高速化手法

次に条件分岐を削減することによる高速化について述べる．kernel 関数 (GPU 側のプログラム) 内の CUDA スレッドは SIMD 方式で実行されるため，条件分岐が存在すると下の図 6 に示すようにすべての分岐に対応する処理が実行されるため，処理性能が低下する．これを避けるため，kernel 関数内の条件分岐数を極力削減する.

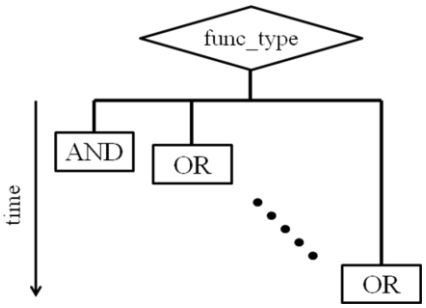


図 6 CUDA での条件分岐処理例

本研究では条件分岐を削除するため，論理ゲートの入力ピン数に依存しない固定サイズの LUT を使用した論理演算を行うことで条件分岐をなくす．LUT によるゲートの論理演算時には，未使用の入力には固定値を設定して 2 入力でも 3 入力でも同じ大きさの LUT を使用することで入力数に依存した条件分岐をなくす． (ver.4)

LUT を使用した論理ゲートの出力値の高速演算法について述べる．演算対象となる論理ゲートの機能タイプと論理ゲートの入力値（in1, in2）を連結し LUT の index を計算する． 以下に計算式を 式 (1) 示す.

$$LUT\_index = \{func\_type, input1, input2\}, \quad (1)$$

この index が指す LUT の値を出力値として得ることにより，高速に論理演算を行う．図 7 に LUT の例を

示す.

	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

図 7　LUT による論理演算高速化

評価回路 adder4x640 で ver.1 と ver.4 で処理時間を比較したところ約 13 倍（ver.1 : 35.7sec → ver.4 : 2.78sec）の高速化が確認できた.

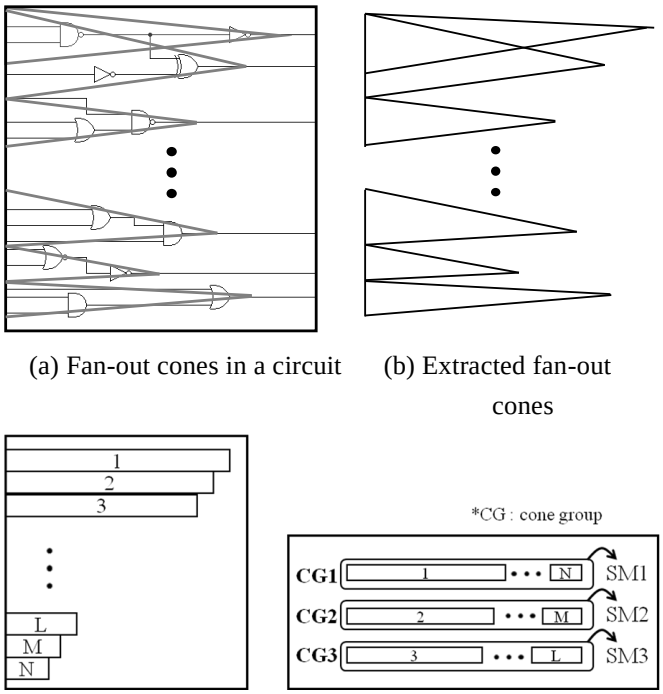
3.3　大規模回路対応の並列化手法

図 3 に示したレベルソート法の並列論理シミュレーションは，単体 SM に適しており，シェアドメモリの容量を超える多数のゲートで構成された大規模な論理回路のシミュレーションを扱えない.

そこで，複数 SM を利用して大規模回路のシミュレーションを高速化する大規模並列シミュレーション手法を提案する．本手法は，論理回路をいくつかのファンアウトコーンに分割し，各ファンアウトコーンで図 3 の手法を適用する．1 つのコーンは，他のコーンの信号の影響を受けないため，1 つのブロック内で並列計算が可能である.

しかし，コーンの数は，外部出力数と同数であるため，単純に適用することは困難である．例えば大規模回路はコーン数が 1,000～数 100,000 以上となる，一方で CUDA のブロック数と SM 数を 1:1 で対応させると, 例えば GTX480 は SM 数が 15 個であるため，大規模回路の全てのコーンをブロックへ割り当てることは不可能である.

提案手法は，複数のコーンを 1 つのグループにまとめ，ブロックに割り当てて演算する方法を採用する．ブロックに割り当てるコーングループ生成の概要と方法は，図 8 と 9 にそれぞれ示す.



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

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

図 8　コーンのグループ化の概要

STEP1. 論理回路からファンアウトコーンを抽出.
STEP2. インスタンス数毎にファンアウトコーンを降順ソート.
STEP3. ファンアウトコーンの段数が均等になるようにファンアウトコーンのグループ割り付け.

図 9　コーングループの作成手順

図 9 に示した手順で複数のコーンから GPU の SM 数と同数のコーングループを作成する．各コーングループは CUDA のブロックに 1:1 で対応させ，図 3 の手法を適応し, 各ブロック（SM）で並列演算を行う.

4　性能評価

すべての高速化手法を実装し高速並列論理シミュレータのプロトタイプを開発した.

開発したプロトタイプシミュレータの GPU 実行時間と市販の高速な論理シミュレータの CPU 実行時間を比較し，高速並列論理シミュレーションプロトタイプの処理時間の比較を行い，高速化手法の有効性を評価した.

4.1　評価環境

実験環境は以下のとおりである.

・ コンピュータ環境

- PC 環境： Windows XP SP3, Intel Core i7-950 3.07GHz
- GPU 環境: CUDA 4.1, Geforce GTX480 (480 コア), 1.15GHz (プロセッサクロック), 1.5GHz（メモリクロック）

・ シミュレータ
・ シーケンシャル論理シミュレータ
- SEQSIM: シーケンシャルなレベルライズドシミュレータ
・ 並列論理シミュレータ
- GPUSim: GPUSim (Geforce GTX480)

・ 商用シミュレータ (VDEC 提供)
- C-Sim (ModelSim SE 10.2c) イベント駆動シミュレータ

4.2 実験内容および評価データ

複数 GPU を使用する並列論理シミュレータ GPUSim のシミュレーション実行時間をレベルソート法によるシーケンシャルシミュレータ(SEQSIM)および高速な市販シミュレータ (C-Sim) と比較した．

実行時間の測定方法は，GPU 環境では NVIDIA CUDA Compute Visual Profiler が示すエラプス時間を用い，PC 環境では市販シミュレーションのエラプス時間を用いた．評価用の論理回路は，組合せ回路として，4bit Adder を 640 個並列に並べた adder4x640 と低密度パリティビットエンコーダである LDPC エンコーダ[12] を用い，順序回路として，CPU コアをそれぞれ 1, 20, 40 個内蔵する 3 種のプロセッサ (CPU01, CPU20，CPU40) を用いた．表 1 に評価回路の一覧を示す.

表 1 評価回路

評価回路	入力ピン数	出力ピン数	FF数	ゲート数	論理段数
adder4x640	9	3200	0	12800	10
LDPC エンコーダ	1723	2048	0	75035	12
CPU01	19	18	173	2111	56
CPU20	19	360	3460	42220	56
CPU40	19	720	6920	84440	56

シミュレーション条件として，テストベクタ長は 100,000 テストサイクルとし，テストベクタとして，2 つの組合せ回路用はランダム生成したもの，順序回路である 3 つのプロセッサ用はロード，ストア，加算などの命令列を使用した.

4.3 評価結果

提案するレベルソート法を用いたシーケンシャルアルゴリズムに基づくシーケンシャルシミュレータ(SEQSIM)と同法を用いた GPU 向きの並列アルゴリズムに基づく並列シミュレータ(GPUSim)を開発した．表 2 には GPUSim と SEQSIM および高速な市販論理シミュレータ (以下 C-Sim) との速度の比較結果を示す．また，評価用回路対応に実行時間[sec]および速度比率が示されている．

表 2 より, GPUSim は高速市販シミュレータ C-Sim と比べ，約 0.36～29 倍の高速化率となった．

GPUSim と C-sim の比較において，特に，組合回路である LDPC エンコーダについては，約 29 倍と最も高い高速化率となった．

一方，順序回路では，ゲート数が約 7.5 万ゲートの順序回路(CPU40)においては GPUSim が C-Sim に比べ, 5.7 倍高速であったが，一番小さい順序回路 (CPU01) では C-Sim より低速であった．

LDPC エンコーダが最も高い高速化率であった理由として，この回路は全ゲート数が 75,000 であり、また論理段数が 12 段であるため，1 つの論理段上のゲート数が多くなりことにより論理ゲートの並列度が高くなるためと考えられる．次に順序回路について，ゲート規模が最大の CPU40 で GPUSim が C-Sim よりも高速であった理由として，大規模回路では並列度が高いため, 並列アルゴリズムに基づく GPUSim は GPU の性能を活かすことができたが，小規模回路では並列効果を発揮することができないためと考える．

表 2 シミュレータの性能比較

評価回路	処理時間[sec]			ratio	
	C-Sim	SeqSim	GPUSim	C / S	C / G
4bit-adder x640	15.3	9.2	1.24	1.66	12.34
LDPCエンコーダ	51.8	44.8	1.77	1.16	29.27
CPU01	1.2	1.22	3.29	0.98	0.36
CPU20	20.1	24.97	5.45	0.80	3.69
CPU40	38.3	50.62	6.75	0.76	5.67

5 まとめ

本研究では GP-GPU 向きの並列論理シミュレーションアルゴリズムを基に GPU アーキテクチャを効率的に利用することによる並列シミュレーションの高速化手法を提案した．その高速化手法は，メモリアクセスの高速化，条件分岐の削減による論理演算の高速化および大規模回路向けのシミュレーション手法から構成される．

本高速化手法に基づき高速並列論理シミュレータ

GPUSim を開発した．GPUSim の性能を市販の高速論理シミュレータ (C-Sim) と比較した結果，組合回路で最大 29 倍 (LDPC エンコーダ)，順序回路では最大 5.7 倍 (CPU40)の高速化率を得ることができた。また, 本評価結果より, 回路大規模になるほど, GPUSim は C-SIM と比較して高速化率が高くなる傾向が見られ，数十万ゲートでは本プロトタイプは市販高速シミュレータと比べ 10 倍以上の高速性を得る見通しを得た．大規模なシステムの評価において，本プロトタイプを GPU スーパーコンピュータ上で動作させると，市販高速シミュレータの数十～数百倍程度の高速性が得られる可能性がある．

今後の課題として，実用的でさらに大規模な回路を用いた評価や GPU クラスタを使用した高速な論理シミュレーションを行っていく必要がある．

またそれらの結果を今後の改良にフィードバックし，並列シミュレーションのさらなる高速化を図りたいと考える．

謝辞

本研究は東京大学大規模集積システム設計教育研究センターを通し，メンター・グラフィックス・ジャパン株式会社の協力で行われたものである．

参考文献

[1] Debapriya Chatterjee, Andrew DeOrio, Valeria Bertacco, “Event-Driven Gate-Level Simulation with GP-GPUs”, DAC2009, July 26-31, 2009

[2] Debapriya Chatterjee, Andrew DeOrio, Valeria Bertacco “GCS: HighPerformance Gate Level Simulation with GPGPUs”, 978-3-9810801-5-5/DATE09 © 2009 EDAA

[3] Bo Wang, Yuhao Zhu, Yangdong Deng, “Distributed Time, Conservative Parallel Logic Simulation on GPUs”, DAC2010, June 12-18, 2010

[4]Sara Vinco, Debapriya Chatterjee, “SAGA:SystemC Acceleration on GPU Architectures”, DAC2012, 2012

[5] Rohit Sinha, Aayush Prakash, and Hiren D. Patel, “Parallel Simulation of Mixed-abstraction SystemC Models on GPUs and Multicore CPUs”, 5C-2, PP.455-460, ASP-DAC2012, January 31- February 2, 2012

[6] 大菊祥子，橋口拓哉，豊永昌彦，村岡道明 ”GP-GPU を用いた並列論理シミュレーションアルゴリズムの評価”, DA シンポジウム 2012 論文集, pp.109-114, 2012 年 8 月 29 日

[7]橋口拓哉，豊永昌彦，村岡道明 ”GP-GPU を用いた並列論理シミュレーション手法”, DA シンポジウム 2013 論文集, pp.97-102, 2013 年 8 月 22 日

[8] 橋口拓哉, 豊永雅彦, 村岡道明, "GP-GPU を用いた高速並列論理シミュレーション手法", ETNET2014, No.19 2014 年 3 月

[9] 松本夏樹, 村岡道明, "FPGA を用いた論理シミュレーション手法", デザインガイア 2013, 2013 年 11 月

[10] 竹内勇矢，トウブンチク，村岡道明 ”並列化アルゴリズムによる論理シミュレーションの高速化手法の提案”, DA シンポジウム 2013 論文集, pp.91-96, 2013 年 8 月 22 日

[11] NVIDIA CUDA Compute Unified Device Architecture

[12] Open Cores. htpp://www.opencores.org