Vol.2013-SLDM-163 No.10
2013/11/27

マルチコアプロセッサを用いた並列論理シミュレーション手法

とう 文竹　　竹内 勇矢　　豊永 昌彦　　村岡 道明

高知大学大学院　理学専攻　情報科学分野
〒780-8520 高知県高知市曙町 2-5-1
E-mail: {bunchiku, ytakeuch, toyonaga, muraoka}@is.kochi-u.ac.jp

あらまし　本稿では、ソフトウェアの実行時間を簡易に見積もる手法を用いて、論理シミュレーションアルゴリズムに適用し、効率の良い並列アルゴリズムを提案する。論理ミュレーションアルゴリズムの中のボトルネック部分である論理演算処理部分について、マルチコアプロセッサを前提とした論理シミュレーションアルゴリズムの並列化により高速化を考える。論理回路を並列処理可能な回路へ分割する方法として、相互に通信が生じないロジックコーンを用いた。また、並列なロジックコーンの処理時間を均等にするために「畳込み法」を提案した。本並列アルゴリズムを組合せ回路と順序回路で評価した結果、両者とも並列化前よりソフトウェア実行時間を 30%以上削減できる見通しを得た。

キーワード　マルチコアプロセッサ、論理シミュレーション、ロジックコーン、並列処理

A Parallel Logic Simulation method based on Multi-core Processor

Wenzhu Dou　Yuya Takeuchi　Masahiko Toyonaga　Michiaki Muraoka
Information Science Division, Graduate School of Science, Kochi University
2-5-1 Akebono－Cho, Kochi, 780-8520 Japan

Abstract　In this paper, the estimation method of software execution time is applied to the logic simulation algorithm to propose an efficient parallel algorithm. This algorithm parallelizes the logic evaluation which is a bottleneck part in the logic simulation algorithm to accelerate the speed of the processing using multi-core processors. The logic cones which have no communication with each other are used to divide the circuit for parallel processing. And, the "Logic cone folding method" is proposed using logic cones to equalize the parallel processing time. The proposed parallel algorithm was evaluated by applying to combinational circuit and sequential circuits, and a prospect that both can be reduced by 30% or more software execution time than before parallelization was obtained.

Keywords　Multi-core processor, Logic simulation, Logic cone, Parallel processing

1. はじめに

現在、組込みプロセッサは数多くの分野で使用され、低消費電力かつ高性能な組込みシステムに用いられている。ソフトウェアの性能を向上する技術として、並列処理を実現すること、いわゆる並列化プログラミングの重要性が高まっている。GPU を利用した並列処理[1]の研究が進行中であり、マルチコアプロセッサの実用化が進みつつあり、これからアルゴリズムの並列化が期待する。

従来では、PC 上の C 言語で記述されたアルゴリズムの実行時間を測ることは簡単にできるが、対象プロセッサを変更した場合(例えばARM946E-S を使用した場合)の実行時間の予測は困難であった。先行研究[2]では、ソフトウェアのプロファイリングを行うことにより、実行時間を簡単に見積もる手法を提案した。本研究では、その手法を論理シミュレーションアルゴリズムに適用し、効率の良い並列化アルゴリズムを提案する。

2. ソフトウェアの実行時間の見積もり手法

ソフトウェアの実行時間の見積もり手法は、図 1 に示す 5 ステップより構成される。

(1) 対象アルゴリズムの時間精度付きモデル[3]を生成する。時間精度付きモデルとは、C 記述のソフトウェアに基本ブロックの実行時間を算出し、プロセッサの実行サイクル数として付与したものを SpecC 言語[4]で記述を行うものである。

(2) (1)で生成した時間精度付きモデルのシミュレーショ

ンとプロファイリング[5]を行う。時間精度付きモデルで算出したシミュレーション結果のデータを用いて、ターゲットプロセッサにおけるアルゴリズムの内部動作やテーブルの動作回数、アクセス頻度、サイクル数などを求める。

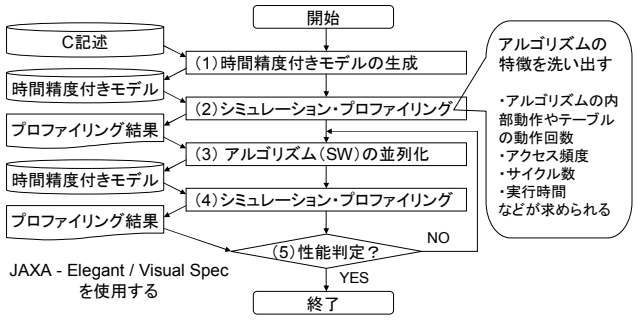

図 1. 見積もり手法のフローチャート

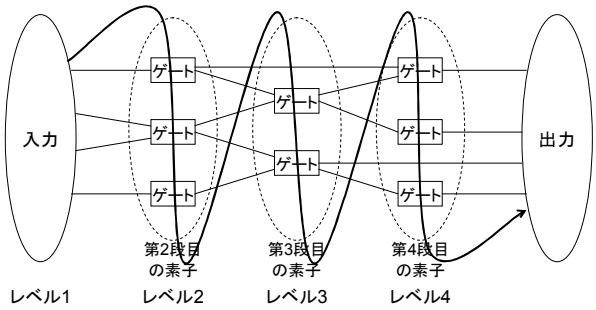

図 2. レベルソート法の演算順番のイメージ図

(3) アルゴリズム（ソフトウェア）の並列化により高速化を図る。
(4) 並列化アルゴリズムのシミュレーションとプロファイリングを行う。(3)で作成した並列アルゴリズムの時間精度付きモデルを生成し、(2)と同様にシミュレーションとプロファイリングを行う。
(5) 性能判定:評価した結果が性能を満たせば終了となるが、アルゴリズムの並列化が適切でない場合には、(3)に戻る。

3. 適用対象（論理シミュレーションアルゴリズム）

論理シミュレーションは論理回路の設計検証の一手法として広く用いられている。近年、半導体技術の進歩により、回路が大規模になって、処理の高速化に対する要望はますます高まっている。

現在、広く普及している論理シミュレータは、イベント・ドリブン型である。これは、信号の変化（イベント）のある素子（論理ゲートなど）に着目する。すなわち、イベントの発生した素子（または端子）の次段の素子のみ演算処理を行う。その素子の出力が変化していると、それを次段へと次々に伝えていく。イベント・ドリブン型では、信号の変化と伝搬遅延時間を考える必要があるので、並列化の実現は難しいと考える。

本論文では、並列化に着目し、ゲートレベル論理回路（ネットリスト）の伝搬遅延時間を考えず、全論理ゲートを順次に演算するレベルソート法を用いる論理シミュレーションアルゴリズムを適用対象とする。

図 2 では、レベルソート法の演算順番を示している。入力を持たない論理素子いわゆる外部入力端子のレベルを 1 とする。論理素子はその入力の論理素子のレベルより必ず大きなレベルが割り当てられる。各素子が属される段とレベルを対応する。図 2 に示したように、入力信号の変化の有無に関係なく、入力ピンに繋がる段から順に段内のすべての論理ゲートを演算する。

今回用いた論理シミュレーションプログラムの処理手順を以下に示す。
(i) ネットリストファイルを読み込む。
(ii) 回路の論理ゲート段数を数える。
(iii) テストベクタを設定する。
(iv) 入力ピンに繋がる段から順に段内のすべての論理ゲートを演算する。
(v) 論理ゲートの演算結果をシミュレーション結果として保存する。

4. 並列論理シミュレーションアルゴリズムの検討

ソフトウェアの実行時間の見積もり手法を論理シミュレーションアルゴリズムに適用し、プロファイリングを行う。

4.1 評価環境

4 章、5 章、6 章でのプロファイリング評価は、以下の環境とする。
・使用シミュレータ：
　Elegant / Visual Spec (vesion4.1.6)
・ターゲットプロセッサ：ARM946E-S
・コンパイラ：arm-elf-gcc
・クロック周波数（最大）：200MHz
・シミュレーションパターン長：10000 サイクル
・評価データ：16bit adder と 8bit マイコン(cpu)

4.2 プロファイリング

今回の評価データは、組み合わせ回路（16bit adder）と順序回路（8bit マイコン）を使用する。8bit マイコン

は評価用回路としては小さいが、順序回路の特徴を持つため、複数個並列に並べると大きい順序回路としての評価が可能と考える。

表 1 では、論理シミュレーションアルゴリズムのプロファイリング結果を示す。その結果、シミュレーションパターン長は 10000 サイクルの場合、論理ミュレーションアルゴリズムの 8 割以上の実行サイクル数を占めた論理演算処理部分がボトルネック部分であることが判明した。

表 1. 論理ミュレーションアルゴリズムの
プロファイリング結果

ファンクション名	16bit adder	8bit マイコン（cpu）
	累積サイクル数（各ファンクションが占める割合）	累積サイクル数（各ファンクションが占める割合）
前処理（ネットリストを読み込む処理など）	8,927,292.5（15.91%）	20,521,126.7（2.07%）
論理ゲート段数を数える処理	27,575.5（0.05%）	5,981,508（0.60%）
論理ゲートを段ごとに演算する処理	**47,155,035（84.04%）**	**967,393,035(97.33%)**
合計（サイクル数）	56,109,903（100.00%）	993,895,669.7（100.00%）

論理演算処理部分について、マルチコアプロセッサを前提とした論理シミュレーションアルゴリズムの並列化により高速化を考える。

4.3 ロジックコーンによる回路の分割方法

並列化プログラム間では、データの転送時間が短くなると、並列効果が顕著になると考えられる。そのため、相互にデータ通信がない回路の分割方法が期待される。今回、論理回路をロジックコーンに変換し、ロジックコーン間のデータ通信をなくし、ロジックコーンを並列実行する演算方法について検討する。

ロジックコーンによる回路の分割はフリップフロップ（FF）や外部入出力端子を境界として各コーン間で通信がないように論理回路を分割する。イメージとして、図 3 に示す。論理回路の入力数が 6 で、出力数が 4 の場合、出力に着目すると、論理回路は 4 つのロジックコーン（図 3 の右側に示す）に分けられる。 コーンとコーンの間には重複する部分があるが、互いに通信することがない。以上のようにロジックコーンを用いて分割すると、論理回路を並列に実行することができる。

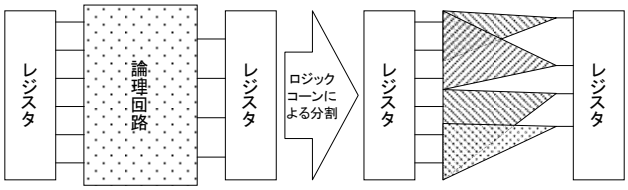

図 3. ロジックコーンのイメージ図

4.4 ロジックコーンのプロファイリング

ロジックコーンによる回路の分割方法を論理シミュレーションアルゴリズムに適用し、プロファイリングを行う。

表 2 では、ロジックコーンの論理シミュレーションアルゴリズムのプロファイリング結果を示す。

表 2. ロジックコーンの論理シミュレーション
アルゴリズムのプロファイリング結果

ファンクション名	16bit adder	8bit マイコン（cpu）
	累積サイクル数（各ファンクションが占める割合）	累積サイクル数（各ファンクションが占める割合）
前処理（ネットリストを読み込む処理など）	9,218,438.5（3.07%）	119,986,385.5（0.24%）
論理ゲート段数を数える処理	27,575.5（0.01%）	5,981,508（0.01%）
ファンアウトコーン処理	330,597.5（0.11%）	127,081,048.5（0.25%）
ロジックコーンごとに論理ゲートを演算する処理	289,835,035（96.80%）	50,415,461,500（99.50%）
合計（サイクル数）	299,411,646.5（100.00%）	50,668,510,442（100.00%）

表 2 のプロファイリング結果と表 1 の結果を比較すると、ロジックコーンの論理演算処理部分で演算するトータルゲート数が増えるため、論理演算処理部分のトータル実行サイクル数が増加した。しかし、1 つのロジックコーンで演算するゲート数は元の論理演算処理部分で演算するトータルゲート数より少なくなる。そのため、ロジックコーンの論理演算処理部分について、並列化を検討し、高速化を考える。

5. ロジックコーンの論理シミュレーションアルゴリズムの並列化評価および考察

ロジックコーンの論理シミュレーションアルゴリズムを用いて、よりよい並列効果を得るための並列化を検討する。

5.1 処理時間の平坦化（ロジックコーンの畳込み法）

論理回路をロジックコーンに変換すると、図 4 の(1)に示すように、各ロジックコーンの演算するゲート数が違い、つまり、各ロジックコーンの処理時間の差が大きくなる。たとえば、順序回路（8bit マイコン）の場合、最大のロジックコーンのサイクル数は 618,370,000 で、最小のサイクル数は 370,000 である（シミュレーションパターン長が 10000 サイクルの場合）。マルチコアプロセッサを用いて並列処理を行う場合、マルチコアの数が限られ、1 つのマルチコアを 1 つのコーングループに対応

させ、複数のロジックコーンをコーングループにまとめることになる。よりよい並列効果を得るために、各マルチコアの処理時間を平坦にすることが必要である。処理時間を平坦にするためにロジックコーンの分割方法“ロジックコーンの畳込み法”を提案し、その手順を以下に示す。図 4 では、処理手順と対応するイメージ図を示す。

(1) 各ロジックコーンの実行サイクル数を調べる。(2 章で説明したソフトウェアの実行時間の見積もり手法(1)、(2)で計測する)

(2) 全てのロジックコーンを実行サイクル数の降順にソートし、サイクル数の合計を求める。

(3) 1 つのコーングループを 1 つのマルチコアプロセッサに対応させ、並列化させるマルチコア数を指定し、コーングループの平均サイクル数(合計サイクル数/マルチコア数)を求める。

(4) 畳込み法を用いて、ロジックコーンをコーングループに割り付ける。

a. 最大コーンのサイクル数がコーングループの平均サイクル数より小さい場合、図 4 の (4) a に示すように、降順にソートしたロジックコーンを順に畳み込む。

b. 最大コーンのサイクル数がコーングループの平均サイクル数より大きい場合、図 4 の (4) b に示すように、以下の処理を行う。

(i) コーングループの平均サイクル数より大きいロジックコーンをそれぞれ 1 つのコーングループに割り付ける。

(ii) マルチコアを有効に使用するため、残りの必要なマルチコアの数(残りのコーンの実行サイクル数の和/最大コーンの実行サイクル数+1)を求める。

(iii) 残りの降順にソートしたロジックコーンを畳み込む。

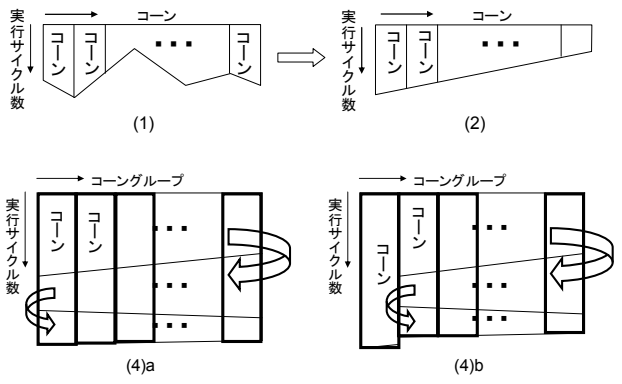

図 4. 畳込み法のイメージ図

(5) より良い結果を得るため、最適化を行う。

最大コーンのサイクル数を減らせないため、(4)の b の場合、最適化する必要がない。

(4)の a の場合、以下の流れで最適化を行う。

a. 得られたコーングループの標準偏差を求める。

b. 最適化処理を実行するたびに標準偏差を更新する。

c. 標準偏差が最小になるまで最適化を行う。

b の最適化処理の手順を以下に示し、イメージとして、図 5 に示す。

(i) 一番大きいコーングループの合計サイクル数(cgmax)と一番小さいコーングループの合計サイクル数(cgmin)の差(cgsub)を求める。式で示すと、cgsub = cgmax - cgmin となる。

(ii) 一番小さいコーングループの中で一番小さいロジックコーンを探索する。

(iii) 見つかったロジックコーンのサイクル数(cmin)とコーングループのサイクル数の差(cgsub) / 2 の和(csum)を求める。式で示すと、csum = cmin + cgsub/2 となる。

(iv) 求めた和(csum)を一番大きいコーングループの中の全てのコーンと比較し、サイクル数が一番近いコーンを探索する。

(v) 一番大きいコーングループの中で見つかったロジックコーンと一番小さいコーングループの中での一番小さいロジックコーンを入れ替える。

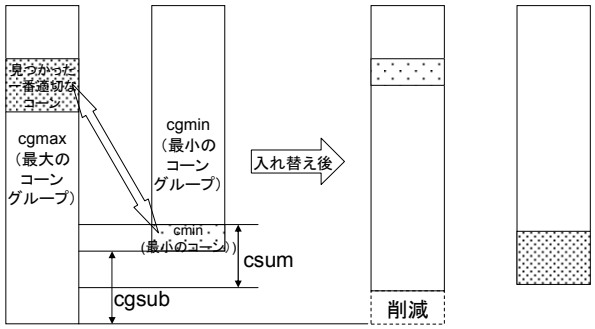

図 5. 畳込み法の最適化のイメージ図

5.2 ロジックコーンの畳込み法の評価

ロジックコーンの畳込み法をロジックコーンの論理シミュレーションアルゴリズムに適用し、評価を行う。

評価データは、組み合わせ回路(16bit adder)と順序回路(8bit マイコン)で、マルチコア数を 8 とする。以下の評価結果は並列のオーバーヘッドなどを考えず、並列処理時間を見通すと、次のような結果となった。今回の評価では、論理演算処理部分のみ並列を行うので、論理演算処理部分の実行サイクル数と並列効果を以下の図 6 と図 7 で示す。

16bit adder の場合、図 6 に示したように、ほぼ均等に

分割され、実行サイクル数を約 87.4%削減することができた。

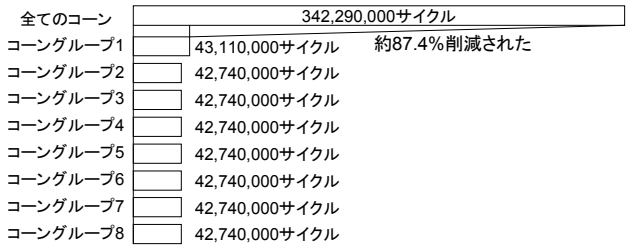

図 6. 16bit adder 並列後のサイクル数

8bit マイコン（cpu）の場合、図 7 に示したように、ほぼ均等に分割され、実行サイクル数を約 87.5%削減することができた。

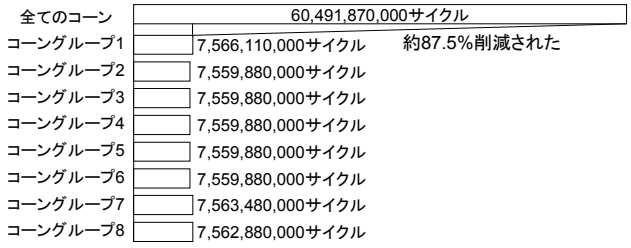

図 7. 8bit マイコン（cpu）並列後のサイクル数

8bit マイコン（cpu）の場合、ロジックコーンの数は 191 であり、仮にマルチコアの数を増やすと、ある限界まで、増やすほど各コーングループのサイクル数が減り、高速化率が向上する。図 8 に示したように、マルチコア数は 99 の場合、最大のロジックコーンを一つのコーングループに割り付けられ、8bit マイコン（cpu）の実行速度が一番高く、実行サイクル数が一番低い。

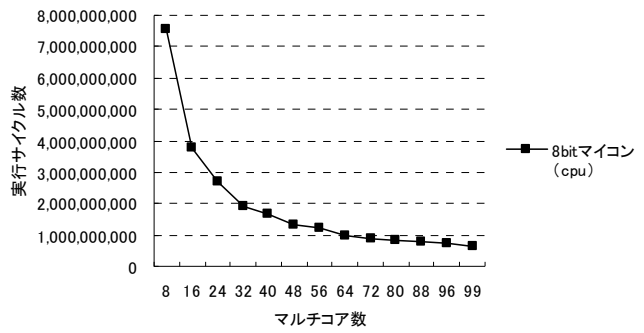

図 8. 8bit マイコン（cpu）の並列度と実行速度の関係図

5.3 考察

ロジックコーンの畳込み法をロジックコーンの論理シミュレーションアルゴリズムに適用し、組み合わせ回路（16bit adder）と順序回路（8bit マイコン）を評価回路として、評価を行った。

その結果、組み合わせ回路（16bit adder）の論理演算処理部分が 8 並列の場合、ロジックコーンの論理演算処理部分より約 87.4%削減することができ、元の論理演算処理部分より 9%削減することができた。

順序回路（8bit マイコン）の論理演算処理部分が 8 並列の場合、元の論理演算処理部分より処理時間が大きかったが、ロジックコーンの論理演算処理部分より約 87.5%削減することができた。

論理シミュレーションアルゴリズムの並列化で高速の効果を得たが、ロジックコーンの論理演算処理部分のコーングループ内で重複に演算するゲート数が多くため、高速率が低く、更なる高速化を検討する必要がある。

6. 並列論理シミュレーションアルゴリズムの高速化の検討と評価

論理回路をロジックコーンにするとき、ロジックコーン間で互いに影響がないため、同じゲートを各ロジックコーンでそれぞれに演算することになる。そうすると、同じゲートを複数回演算することがある。

マルチコアプロセッサを用いて並列処理を行う場合、全てのロジックコーンを並列に演算することが難しくて、複数のロジックコーンをコーングループにまとめ、コーングループを並列に演算することになり、コーングループの中で同じゲートを複数回に演算することなる。コーングループの中で同じゲートを重複に演算しないようにすると、さらに高速化になると考える。

16bit adder と 8bit マイコン(cpu)を 8 並列する場合、重複演算を削除する前と後の比較は以下の表 3 で示す。

表 3. プロファイリング結果の比較

	16bit adder	8bit マイコン(cpu)
	累積サイクル数	累積サイクル数
元の論理演算処理部分	47,155,035	967,393,035
ロジックコーンの論理演算処理部分（トータル、重複演算あり）	289,835,035	50,415,461,500
ロジックコーンの論理演算処理部分（8並列、重複演算あり）	43,110,000	7,566,110,000
ロジックコーンの論理演算処理部分（8並列、重複演算なし）	31,680,000	656,060,000

その結果、組み合わせ回路（16bit adder）の論理演算処理部分が 8 並列の場合、重複演算を削除する後は削除する前より約 26.5%削減することができ、元の論理演算処理部分より 32.8%削減することができた。

順序回路（8bit マイコン）の論理演算処理部分が 8 並列の場合、重複演算を削除する後は削除する前より約

91.3%削減することができ、元の論理演算処理部分より32.2%削減することができた。

組み合わせ回路（16bit adder）と順序回路（8bit マイコン）を高速化した結果、8 並列にしたが、1.5 倍の高速率しか得られなかった。以下にその理由を述べる。

16bit adder の回路では、トータルのインスタンス数は 130 で、ロジックコーンにすると、最大のロジックコーンのインスタンス数は 64 である。インスタンスの数と実行時間が比例することを考えると、並列度を増やすと、最大 2 倍の高速率が得られる。

8bit マイコン(cpu)の回路では、トータルのインスタンス数は 2148 で、ロジックコーンにすると、最大のロジックコーンのインスタンス数は 1030 である。並列度を増やすと、最大 2 倍の高速率が得られる。

今回の評価ではオーバーヘッドなどを考えなかったが、重複演算を削除すると、コーングループの均等が崩れ、ほぼ平坦に実行することができないため、最大の高速率が得られなかった。今後、重複演算を削除する後の平坦化も考える必要がある。

7. まとめと今後の課題

7.1 まとめ

本研究では、論理シミュレーションアルゴリズムのボトルネック部分である論理演算処理部分について、並列化アルゴリズムを提案し高速化の可能性を見積もった。

今回の評価で用いた論理回路データの並列度が低く、1.5 倍の高速率が得られた。複数の組み合わせ回路（16bit adder）と順序回路（8bit マイコン）を並べる回路で、並列度が高く、もっと良い高速率が得られると予測される。

今回高速化した論理シミュレーションアルゴリズムの実行時間と市販シミュレータを比較すると、以下の表 4 で示す。

評価環境を以下となる。

・動作環境：Intel(R) Core(TM) i5-2500 CPU @ 3.3GHz
・クロック周波数：3.30GHz
・シミュレーションパターン長：10000 サイクル
・市販シミュレータ：ModelSim SE 6.2e （VDEC 提供）

表 4. 市販シミュレータとの比較表

	adder 16X4 [sec]	adder 16X8 [sec]	cpu [sec]	cpu X2 [sec]	cpu X4 [sec]	cpu X8 [sec]
元の論理演算処理部分	0.031	0.063	0.187	0.374	0.765	1.622
ロジックコーンの論理演算処理部分（並列、重複演算なし）	0.030 （8並列）	0.058 （16並列）	0.251 （2並列）	0.368 （4並列）	0.534 （8並列）	0.945 （16並列）
市販シミュレータ	0.108	0.160	0.137	0.225	0.407	0.795

今回高速化した論理シミュレーションアルゴリズムの実行時間と市販シミュレータを比較すると、組み合わせ回路の場合は 2 倍以上の高速率を得た。順序回路については、市販シミュレータよりやや遅かったが、論理回路の規模を大きくするとより高い高速率が得られると見通すことができた。

7.2 今後の課題

本研究では、論理シミュレーションアルゴリズムについて、並列化アルゴリズムを提案し、高速化の可能性を見積もる評価を行った。

今後の課題として、アルゴリズムのブラッシュアップによりさらなる高速化ができると考えられる。また、ゲートの複合化による高速化方法[6]や論理シミュレーションアルゴリズムの重複演算を削除後の平坦化も考えられる。さらに、大規模な論理回路や多種の回路による評価を行うとともに、マルチコアを用いた並列アルゴリズムの高速化効果を実証したいと考える。

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, “GCS: High-Performance Gate-Level Simulation with GP-GPUs”, 978-3-9810801-5-5/DATE09 © 2009 EDAA

[2] 松永惇弥，村岡道明，“タイミングを考慮したハードウェア/ソフトウェア分割手法の評価”，デザインガイア 2009，pp31-36，2009 年 12 月

[3] Michiaki Muraoka, Noroyoshi Itoh, Rafael K. Morizawa, Hiroyuki Yamashita, Takao Shinsha, “Software Execution Time Back-annotation Method for High Speed Hardware-Software Co-simulation”, Proc. Of SASIMI2004, pp.169-175, October. 2004

[4] Andreas Gerstlauer, Rainer Dömer, Junyu Peng, Daniel D. Gajski, “システム設計：SpecC による実現”，2001 年

[5] 荒木大，“ELEGANT でのシミュレーションについて”，システム設計環境セミナーーELEGANT の研究開発成果発表ー前刷集，pp.43-59，2008 年 3 月

[6] 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 2013，2013 年 8 月