情報処理学会論文誌　コンピューティングシステム　Vol.7　No.1　46–56 (Mar. 2014)

LSI論理シミュレーションにおける SIMD並列化手法の提案

甲斐 夏季[1,a)]　小出 洋[2,b)]

受付日 2013年7月25日, 採録日 2013年12月25日

概要：大規模集積回路（LSI）の設計時には，ある入力を行ったときの回路の出力が設計者の意図するものであるかを確認する必要があり，その工程を論理シミュレーションと呼ぶ．LSI 論理素子の集積度は増加の一途をたどっており，それにともない論理シミュレーションに要する時間が増大し，高速なシミュレーション手法の必要性が高まっている．本研究の目的は LSI 論理シミュレーションの高速化である．手法として，ネットリストをタスクグラフに変化し，得られたタスクグラフに対して SIMD 並列化処理を静的に行い，SIMD 演算を用いて高速化することを提案する．SIMD 演算用のプロセッサとして今回は Cell Broadband Engine（Cell/B.E.）を用いた．提案手法を評価するために，元のタスクグラフと SIMD 並列化後のタスクグラフを比較して，タスク数をどれだけ減少させることが可能であるか実験を行った．また，2 つのタスクグラフを Cell/B.E. 上の実行プログラムで実行し，シミュレーション実行時間を比較した．その結果，タスク数を最大で 87%減少させることができ，シミュレーション実行時間は最大で 86%短縮することができた．

キーワード：並列分散処理，SIMD 並列化，SIMD 演算，Cell/B.E.，LSI 論理シミュレーション

A SIMD Parallelization for an Application for LSI Logic Simulation

Natsuki Kai[1,a)]　Hiroshi Koide[2,b)]

Received: July 25, 2013, Accepted: December 25, 2013

Abstract: This paper proposes to SIMD parallelization and a task scheduling method in order to make LSI logic simulation run faster. LSI logic simulation is a confirming process if the output is much as the designer expects or not. The number of elements of LSI has been increasing dramatically in these days. The simulator has to spend much more time to simulate because of it, and logic simulation occupies most of whole LSI development time. These backgrounds urge us to make LSI logic simulation run faster. In our proposal method, we convert a netlist into a task graph, and then make that task graph SIMD parallelized as a preparation. Task graph we got are executed by SIMD instructions as experiments to evaluate our proposal method, tasks in a SIMD parallelized task graph are executed by Cell Broadband Engine (Cell/B.E.) with SIMD arithmetic logical units, and we measure and compare that elapsed time. In the results of experiments, 87% tasks are SIMD parallelized and the simulation time on Cell/B.E. decreases by 86%, at the most.

Keywords: parallel and distributed processing, SIMD parallelization, SIMD instruction, Cell/B.E., LSI logic simulation

[1] 九州工業大学大学院情報科学専攻
Graduate School of Computer Science and Systems Engineering, Kyushu Institute of Technology, Iizuka, Fukuoka 820–8502, Japan
[2] 九州工業大学大学院情報科学研究院
Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology, Iizuka, Fukuoka 820–8502, Japan
[a)] kai@klab.ai.kyutech.ac.jp
[b)] koide@ai.kyutech.ac.jp

1. はじめに

本研究では，大規模集積回路（Large Scale Integration, LSI）設計の論理シミュレーションにおける組合せ回路のテスト計算を SIMD（Single Instruction stream Multiple Data stream）[1] 並列化し，タスクスケジューリング手法を用いて高速化することを提案する．

© 2014 Information Processing Society of Japan

論理シミュレーションとは，LSI の設計段階において設計した回路の出力検証を行う技術であり，ある入力テストパターンに対する出力が設計者の意図するものであるか否かの検証を行うものである．LSI の回路素子の集積度は増加の一途をたどっており，それにともない論理シミュレーションに要する時間は増大し，設計時間の多くを占めるようになっている．

従来の手法ではネットリストを参照し，入力としてテストパターンを読み込み，信号線追跡を行いながら回路素子の論理演算を行うことで出力を求める，という工程をテストパターンの数だけ行っていた．そのため信号線追跡が計算時間の多くを占めており [2], [3]，また，信号線追跡は逐次的に行う必要があるため回路の分割による並列化や SIMD 並列化が難しい．そのため従来手法の多くは，高速化のために入力データによる並列化を行うが，素子数の増加による 1 つのテストパターンあたりの計算時間が増大するという問題に対処できない．

提案手法では，回路素子の論理演算を処理単位（タスク），回路素子間の信号線接続をタスク間の依存関係としたタスクグラフとして回路を表現する．このタスクグラフに SIMD 並列化を含む並列化に関する手法を適用し，計算資源を適切に利用してタスクを実行する．タスク実行では，一度に複数のデータに対して処理を行う SIMD 演算を用いることで高速化できる．

この手法には以下のような利点がある．

- 事前にネットリストを静的に処理するため，実際の出力計算時に信号線を追跡しない．
- 事前処理で SIMD 並列化を行うことで同時に処理できる演算をまとめ，SIMD 演算により高速にタスク実行できる．
- 将来的にはタスクスケジューリング手法や，タスクグラフの並列化に関する既存の手法を適用できる．

今回は提案手法の評価のため，タスクグラフを単純な方法で SIMD 並列化し，元のタスクグラフのタスク数と比較して，並列化によってタスク数をどれほど減らすことが可能か試みる実験を行った．また，SIMD 並列化効率を上げるための改良プログラム（以下，改良 SIMD 並列化）を実装し，単純な手法の SIMD 並列化（以下，単純 SIMD 並列化）と比較してどれだけタスク数を減少させることができるか，またどれだけ計算時間を高速化できるか調べた．単純 SIMD 並列化と改良 SIMD 並列化については 5 章で説明する．そのうえで，元のタスクグラフと単純 SIMD 並列化後のタスクグラフ，改良後の SIMD 並列化タスクグラフを Cell Broadband Engine [4]（Cell/B.E.）上で SIMD 演算を行う実行プログラムで実行し，実行時間を比較した．その結果，単純 SIMD 並列化と改良 SIMD 並列化について並列化効率の明確な差は見られなかったが，SIMD 並列化処理によって最大でタスク数を 80%以上削減（表 **2** 参照）することが可能となり，SIMD 演算によってシミュレーション時間は最大で 90%以上短縮（図 **17** 参照）させることが可能であると判明した．

本論文では，2 章において LSI の論理シミュレーションについて説明し，3 章では従来のシミュレーション手法について説明する．そして，4 章で提案手法についての説明を行い，5 章で SIMD 並列化についての説明を行う．そして，6 章で提案手法の評価実験を行い，最後に本研究のまとめと今後の課題について述べる．

2. LSI の論理シミュレーション

LSI の設計時に，ある入力に対する出力が開発者の意図するものか確認する技術は論理シミュレーション（テスト計算）と呼ばれている [5]．論理シミュレーションの概略を図 **1** に示す．

LSI のテスト計算は，内部状態が存在することで出力が過去の状態に依存する順序回路と，内部状態を持たず出力が入力のみで決定する組合せ回路とに切り分けて行われるが，本研究では組合せ回路のテスト計算を高速化する．順序回路については，内部状態保持のためにラッチを使用することで仮想的に組合せ回路の組合せ（図 **2** 参照）と考えることが可能であり，本研究の適応範囲内であると考える．

組合せ回路は論理積（AND）や論理和（OR），否定（NOT）などの論理素子から構成されるループのない回路であり，

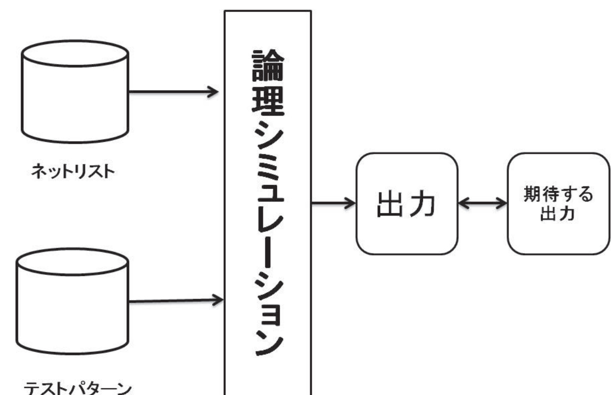

図 1　論理シミュレーションの概略
Fig. 1　Overview of logic simulation.

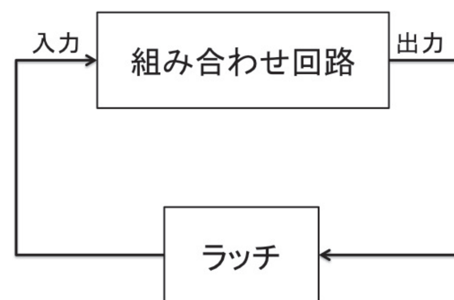

図 2　順序回路適用概略図
Fig. 2　Overview of adaptation to sequential circuit.

ⓒ 2014 Information Processing Society of Japan

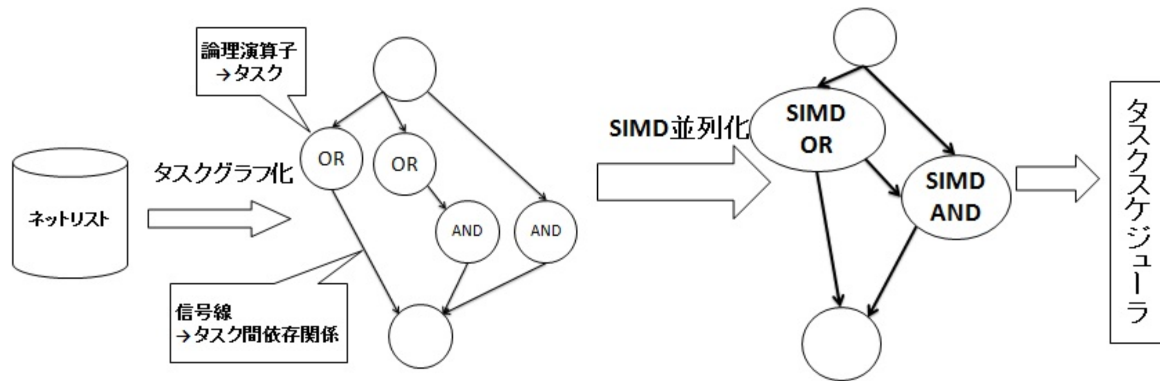

図 3　提案手法
Fig. 3　Proposal method.

そのテスト計算は，テストパターンを入力したときの回路の出力と，開発者の期待する出力とを比較することで行われる．このテスト計算において，回路は素子の種類や信号線接続情報からなるリスト（ネットリスト）で表現され，従来の手法ではこのリストを参照して信号線を追跡し，回路の出力を計算する．

3. 従来のシミュレーション手法

3.1 計算手順

組合せ回路のテスト計算では回路を表すネットリストを参照して出力を計算し，設計者の期待値と比較する．以下の手順でテスト計算を行う．

(1) 組合せ回路のネットリストを読み込む．
(2) 入力信号のテストパターンを読み込む．
(3) 信号線追跡，論理演算によって出力信号を求める．
(4) 求めた出力と期待する出力とを比較する．
(2)–(4) の手順をテストパターンの数だけ繰り返す．

3.2 高速化方法

この手法を高速化する方法として，テストパターンを分割してそれぞれのパターンを異なる複数の計算機に割り当て，シミュレーションを行う方法がある．この方法は提案手法においても適用可能であり，提案手法を実装したシステムを複数用意し，テストパターンを分割してシステムそれぞれで並列にシミュレーションを行うことが可能である．

3.3 従来の手法の問題点

この手法の問題点として，信号線追跡に多くの時間を要することがあげられる．この手法ではテストパターンごとに毎回信号線追跡を行っており，シミュレーション時間の大部分を占めているため，この時間を短縮することで高速化が可能である．

次に，並列化が不十分である点があげられる．回路規模が大きくなると 1 つのテストパターンあたりの計算時間が大きくなるため，テストパターンによる分割のみでは対処できない．この手法では手順 (3) で逐次的に計算を行っているため，より粒度の細かい並列化や SIMD 並列化，分散化が難しいことに原因がある．

この手法に加えて，事前にネットリストを処理し，論理演算の計算順序を指定してプログラム化するコンパイラ方式を用いる [6] ことがあるが，この場合も論理演算を逐次的に行うため，やはり並列化が難しい．

提案手法では計算前の前処理として回路をタスクグラフで表現して並列分散化，SIMD 並列化を行い，SIMD 演算器を用いて複数のテストパターンを同時計算することでこれらの問題を解決し，シミュレーションを高速化する．

4. 提案手法

4.1 計算手順

提案手法の概略図を**図 3** に示す．提案手法では以下のような手順でテスト計算を行う．

(1) 組合せ回路のネットリストを静的に処理し，回路を表すタスクグラフを生成する．
(2) タスクグラフに対し，静的 SIMD 並列化処理を施す．
(3) スケジューリングを用いて，動的にタスクを計算機に割り当てる．
(4) SIMD 演算を用いてタスクの論理演算を行う．
(5) 求めた出力と期待する出力とを比較する．

これらの手順のうち，(1) と (2) は計算前に静的に行い，(3) と (4) で動的にタスクスケジューリングを行ってタスク実行することで，テスト計算を高速化する．本研究では手順 (2) を実装することで，実行タスク数を減少させることを目的としている．また動的タスクスケジューリングについては未実装のため今後の課題とする．

4.2 タスクグラフによる論理回路図の表現

タスクとは，プログラム内のループやサブルーチンなどのまとまった処理単位のことであり，入力と処理内容，出力からなる．一般的に，プログラムのそれぞれのタスク間には依存関係が存在し，タスクと依存関係は非循環有向グ

© 2014 Information Processing Society of Japan

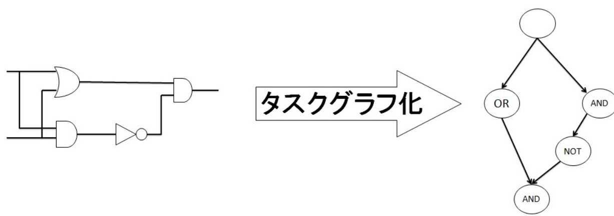

図 4　タスクグラフの例
Fig. 4　Example of task graph.

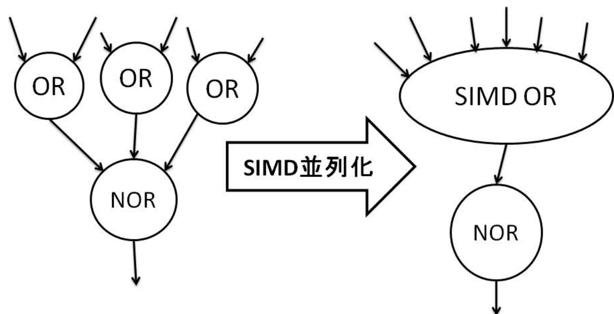

図 5　SIMD 並列化の例
Fig. 5　Example of SIMD parallelization.

ラフで表すことが可能であり，これをタスクグラフと呼ぶ．このグラフの情報を用いてタスクを実行する．

手順 (1) では，組合せ回路を並列プログラムと見なし，ネットリスト中の回路素子の論理演算をタスク，素子間の信号線接続を依存関係としたタスクグラフを生成する．タスクグラフの例を**図 4** に示す．

タスクグラフとして回路を表現することで並列分散化が容易となり，タスクを並列分散実行できる．また，タスクグラフに対して SIMD 並列化を含む様々な最適化を行うことで，タスク実行を高速化できる．

4.3　SIMD 並列化

LSI は論理演算を行う多数の素子から構成され，中には同じ論理演算を行うものも多く含まれる．SIMD 並列化とは，複数のデータを同時に処理する SIMD 命令を活用するための最適化作業であり，同時に実行可能なタスクをまとめることで，SIMD 命令により一度に処理する．SIMD 並列化の例を**図 5** に示す．

タスクグラフとして表現した回路において，同じ論理演算を行うタスク間に依存関係がなければ，まとめて 1 つのタスクとすることでタスク割当てに要する時間を削減でき，SIMD 命令を用いて論理演算を行うことで高速にタスクを実行できる．SIMD 並列化については 5 章で詳しく説明する．

4.4　SIMD 演算を用いたタスク実行

本手法では，割り当てられたタスクを実行する際，SIMD 命令を用いて論理演算を行うことで多くのデータを一度に処理し，実行を高速化する．また，SIMD 演算の得意なプロセッサでタスクを実行することで高速に SIMD 演算を行う．SIMD 演算が得意なプロセッサは，画像処理を行う Graphics Processing Units（GPU）を汎用計算に利用する GPGPU（General-Purpose computing on GPU）や，Cell/B.E. のようなヘテロジニアスマルチコアプロセッサがあげられる．後述する評価実験では Cell/B.E. を用いてタスクを実行する実験を行った．

5.　SIMD 並列化

5.1　SIMD 並列化の要件

SIMD 並列化を行うにあたって 3 つの条件がある．1 つ目は，並列化を施すタスクがすべて同一の論理演算子を持つことである．SIMD 演算は複数データを単一命令で実行可能であるが，異なる演算は不可能といった制限がある．そのため OR タスクと OR タスク，AND タスクと AND タスクのように同一の論理演算子を持ったタスクどうしは SIMD 並列化可能であるが，OR タスクと AND タスクのように異なる論理演算子を持つタスクは SIMD 並列化不可能とする．

2 つ目の条件は SIMD 並列化を施すタスク間に依存関係がないことである．あるタスクを実行する前に実行しておかなければならないタスクを先行タスクと呼んでいる．並列化を試みる 2 つのタスクにおいて，片方のタスクの先行タスクがもう一方のタスクである場合，もしくは先行タスクを通してもう一方のタスクに到達可能である場合，我々はこの 2 つのタスクは互いに依存関係を持つタスクとし，SIMD 並列化不可能とする．

3 つ目の条件として，1 つのタスクにおける最大 SIMD 並列数を 16 とする．SIMD 並列数の制限は今回実験で使用する Cell/B.E. のアーキテクチャの制限 [7] となっている．よって，SIMD 並列化が可能な場合においても，すでにどちらか一方のタスクが 16 並列していた場合，そのタスクは SIMD 並列化不可能とする．

5.2　手順

SIMD 並列化は 3 つのステップから構成されている．

(1) 各タスクに実行順序番号を割り振る．**図 6** に示されている右端の数字が実行順序番号であり，丸の中の数字は各タスクの番号，文字列はそれぞれの論理演算子とする．
(2) 同一実行順序内で同一の論理演算子を持つタスクを SIMD 並列化する．**図 7** の太枠で囲まれた部分が SIMD 並列化されたタスクである（単純 SIMD 並列化）．
(3) まだ最大 SIMD 並列数（今回の実験では 16）まで並列化されていないタスクは，実行順序番号をずらしてさらに SIMD 並列化していく（改良 SIMD 並列化）．

© 2014 Information Processing Society of Japan

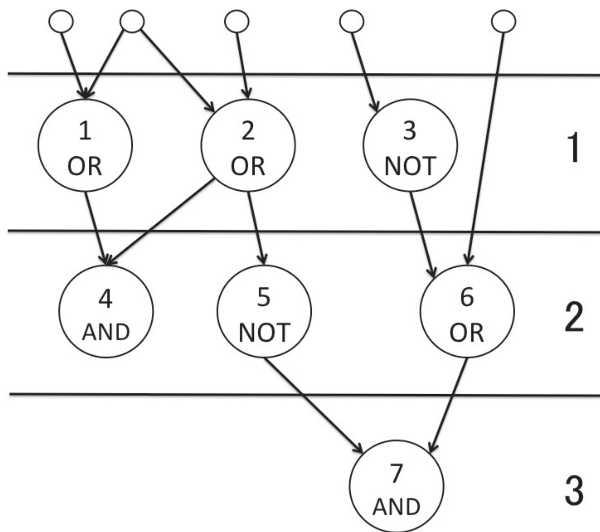

図 **6**　手順 (1)
Fig. 6　Step (1).

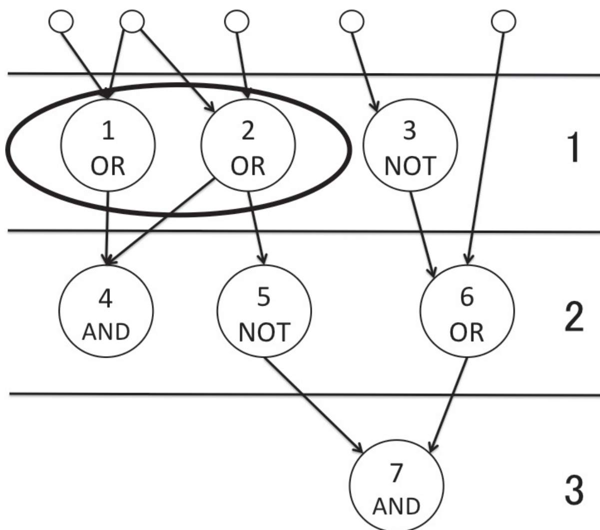

図 **7**　手順 (2)
Fig. 7　Step (2).

手順 (1) では，初期値を持つ入力タスク（図 6 中の小丸）が存在する状態で，先行タスクがすべて値を持っている状態となっているタスクを実行可能なタスクとして実行順序番号 1 を割り振る．そして実行順序番号 1 が終了した状態で，次に実行可能なタスクに実行順序番号 2 を割り振る，といったサイクルを最後まで続けることで全タスクにそれぞれの実行順序番号を持たせる．図 6 の場合，タスク番号 1，2，3 のタスクは先行タスクがすべて入力タスクであるため実行順序番号 1 を割り振られている．しかしタスク番号 6 の OR タスクは先行タスクに，入力タスクとタスク番号 3 のタスクを持つため，タスク番号 3 のタスクが実行されてからでないと実行可能とならない．そのためタスク番号 6 は実行順序番号 2 が割り振られている．初期状態であるこのときのクリティカルパスは 3 で，タスク数は入力タスクを除いて 7 である．

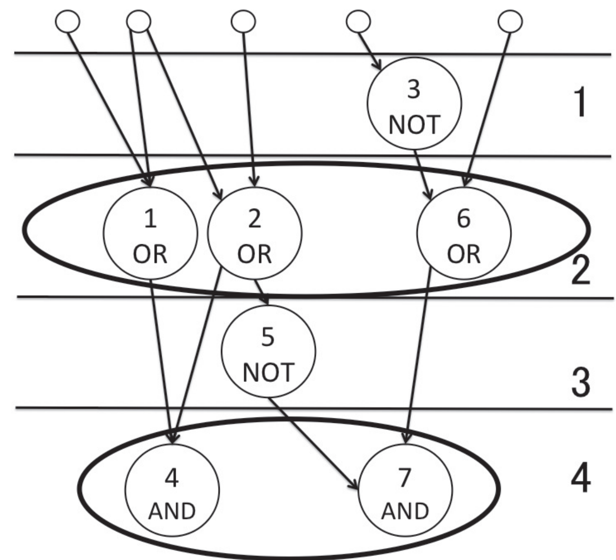

図 **8**　手順 (3)
Fig. 8　Step (3).

手順 (2) では，同一実行順序で SIMD 並列化の要件を満たしたタスクを SIMD 並列化していく．図 7 でタスク番号 1 とタスク番号 2 の OR タスクが SIMD 並列化の要件を満たしているため並列化されているが，そのほかのタスクは要件を満たしていないためそのままとなっている．タスク番号 4 とタスク番号 7 の AND タスクや，タスク番号 1，2，6 の OR タスクは SIMD 並列化可能ではあるが，まだこの段階では SIMD 並列化を行わない．その理由は，むやみに実行順序番号の違うタスクと SIMD 並列化をすることによってクリティカルパスが冗長となってしまう可能性があるからである．手順 (2) が終了すると，クリティカルパスを本来のタスクグラフのものと変えることなく実行タスク数のみを減少させることが可能となっている．ここまでの処理を “単純 SIMD 並列化（simple SIMD）” と呼ぶことにする．この時点でのクリティカルパスは手順 (1) と変わらず 3 で，タスク数は 1 つ減少して 6 である．

手順 (3) では，手順 (2) から実行順序番号をずらしてさらに SIMD 並列化していく．これにより前述したとおりクリティカルパスが長くなってしまう可能性があるが，実行タスク数を減らすことが可能である．図 8 では，タスク番号 1，2 の実行順序を 1 から 2 に遅らせることでタスク番号 6 の OR タスクと SIMD 並列化が可能となっている．その結果，タスク番号 4，5 のタスクは実行順序が遅れ，タスク番号 5 を先行タスクとして持つタスク番号 7 の AND タスクの実行順序が 3 から 4 となっている．そこからさらにタスク番号 4 の AND タスクの実行順序を 3 から 4 に遅らせることでタスク番号 7 の AND タスクと SIMD 並列化されている．手順 (3) まで実装することでクリティカルパスが 3 から 4 に増加しているが，実行タスク数を 7 から 4 まで減少させることが可能となっている．ここまで処理することを “改良 SIMD 並列化（improved SIMD）” と呼ぶことに

© 2014 Information Processing Society of Japan

する．この場合，どれだけ実行順序を遅らせることを許可するかが非常に重要な要素となってくる．なぜならば仮に実行順序を無限に遅らせることを許可すれば，確かにより多くのタスクをSIMD並列化可能であると考えられる．しかしそれでは最悪の場合，クリティカルパスがタスクの数と等しくなってしまい，タスク実行の際に大きな障害となる．このことから実行順序遅延許可範囲については熟考が必要であるが，すべてのタスクグラフに対して明確な答えとなるものを導き出すことが大変困難であるため，今回の実験は試験的に実行順序遅延許可範囲を “2” 以内，つまり実行順序番号の差が2より大きい場合，SIMD並列化不可能と見なす，という条件で行った．この許可範囲については6.6節で詳しく述べる．次に，手順(3)における実行順序遅延タスク決定アルゴリズムについて説明する．実行順序番号の若い順にタスクを探していき，まだ最大SIMD並列数（今回は16）まで並列化されていないタスク（以下タスクAとする）を見つけたら，タスクAの実行順序を1つだけ遅延させ，遅延後の実行順序番号でタスクAが他のタスクとSIMD並列化可能かどうかをチェックする．SIMD並列化可能だった場合SIMD並列化を行い，SIMD並列化不可能だった場合タスクAの実行順序は元に戻し，また別のタスクをチェックしていく．同一実行順序内でタスクAを見つけ出す方法について，現在はSIMD並列化について可能かどうか確かめるために演算子ごとに順番にチェックを行うという簡潔なアルゴリズムを用いている．チェックを行う順番は，OR，AND，NOR，NAND，NOTの順となっている．

6. 評価実験

今回，本研究の提案手法を評価するために3つの実験を行った．また，追加実験として先行研究との実行時間比較およびインテルプロセッサを用いた実行時間比較を行った．今回の実験に用いたネットリストはISCAS89 [8], [9], [10]で紹介されており，シミュレーション回路としての妥当性は十分保証されている．

6.1 SIMD並列化実験

まず最初にSIMD並列化実験である．この実験ではネットリストをタスクグラフ化し，得られたタスクグラフに対してSIMD並列化処理を施し，SIMD並列化実行前のタスク数と実行後のタスク数を比較してどれだけ減少させることが可能か調べた．**表1**に実験に用いた各タスクグラフのタスク数，クリティカルパス長および入力数を示す．

次に表2に，表1のタスクグラフを用いてSIMD並列化実験を行った結果を示す．表中の “simple SIMD” および “improved SIMD” については5章で述べたため説明を省略する．

表2によると，s208の元々のタスク数は134であり，単純SIMD並列化を施したところタスク数が78まで減少し，並列化前のタスク数（134）と比較するとその減少率は41%となる．**図9**にSIMD並列化実験によるクリティカルパスの変化を示す．

表1　実験に用いたタスクグラフの詳細
Table 1　Information about task graphs.

	タスク数	クリティカルパス長	入力数
s208	134	16	19
s1196	685	28	32
s1494	874	21	14
s5378	3650	30	214
s9234	6441	62	247

表2　SIMD並列化実験によるタスク数の変化
Table 2　Difference of tasks with SIMD parallelization.

	SIMD 並列化前の タスク数	simple SIMD 後のタスク数 （タスク減少率 [%]）	improved SIMD 後のタスク数 （タスク減少率 [%]）
s208	134	78 (41)	60 (55)
s1196	685	178 (74)	152 (86)
s1494	874	122 (86)	114 (87)
s5378	3650	679 (81)	667 (81)
s9234	6441	994 (84)	952 (85)

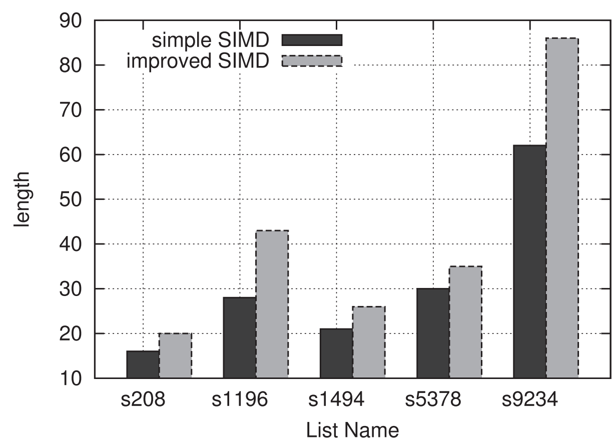

図9　SIMD並列化実験によるクリティカルパス長の変化
Fig. 9　Difference of length of critical path with SIMD parallelization.

図9より，どのタスクグラフも改良SIMD並列化を施すことによってクリティカルパスが長くなっていることが分かる．一方で表2より，どのタスクグラフも80%前後のタスクを減少させていることが分かる．s208のみが単純SIMD並列化，改良SIMD並列化を施しても50%程度のタスク減少率にとどまっている理由は，元のタスクグラフサイズが他と比較して小さいことがあげられる．表2より，s208はクリティカルパスが16に対してタスク数が134しかない．これは同一実行順序内におけるタスク数が平均して8程度しかないことを示している．一方で80%前後のタスク減少率を示したs1196，s1494，s5378，s9234に関して

© 2014 Information Processing Society of Japan

表 3　各タスクグラフにおける SIMD 並列数の詳細
Table 3　Detailed information about each SIMD parallelized task graph.

		最大 SIMD 並列化タスク数	SIMD 並列数 9 以上のタスク数	その他のタスク数	SIMD 並列化不可タスク数割合 [%]
s208	simple SIMD	1	0	48	2.040
	improved SIMD	1	0	30	3.225
s1196	simple SIMD	11	15	88	22.807
	improved SIMD	13	19	55	36.782
s1494	simple SIMD	34	12	12	53.488
	improved SIMD	34	15	26	65.333
s5378	simple SIMD	164	34	39	83.544
	improved SIMD	167	33	26	88.496
s9234	simple SIMD	299	61	137	72.435
	improved SIMD	302	67	89	80.568

は，同一実行順序内に平均して 20 以上のタスクが存在している．これらのことから，サイズの小さいタスクグラフは SIMD 並列化，特に単純 SIMD 並列化には不向きであると考えられる．

s1494，s5378，s9234 について，単純 SIMD 並列化と改良 SIMD 並列化の間に明確な差が見られないが，この理由は大きく分けて 2 つあると考えられる．1 つ目は単純 SIMD 並列化を施したことによりタスク間の依存関係が増加してしまったことである．タスク数が比較的多いタスクグラフにおいては，同一実行順序内に多くの SIMD 並列化可能タスクが存在する．それらを可能な限り SIMD 並列化することにより 1 つのタスクがより多くの依存関係を持ってしまう．それにより実行順序を遅らせ，さらなる SIMD 並列化を試みても，どのタスクとも依存関係を持ってしまい SIMD 並列化できない状況にあると考えられる．2 つ目の理由は，SIMD 並列化要件の 3 番目である最大 SIMD 並列数である．今回の実験で用いた Cell/B.E. の仕様上，最大 SIMD 並列数を “16” としてある．つまり，単純 SIMD 並列化を施すことにより多くのタスクが最大 SIMD 並列数に達していたということである．**表 3** に各タスクグラフにおける最大 SIMD 並列数に達したタスク数，SIMD 並列数が 9 以上のタスク数および平均 SIMD 並列数を示す．SIMD 並列数が 9 以上のタスクというのは，そのタスクどうしでは SIMD 並列化が不可能なタスクの数を示し，最大 SIMD 並列数に達しているタスクは除外されている．SIMD 並列化不可タスク数割合は，そのタスクどうしでは SIMD 並列化を行うことができないタスクの割合を示す．表 3 より，s1494，s5378，s9234 の SIMD 並列化不可タスク数割合は s208，s1196 と比較して倍以上高くなっている．s5378 を見てみると，80%以上のタスクが SIMD 並列化不可な状態になるまで SIMD 並列化が施されているということである．これらのことが原因で s1494，s5378，s9234 において，単純 SIMD 並列化と改良 SIMD 並列化の間に明確な差が生まれなかったと考える．

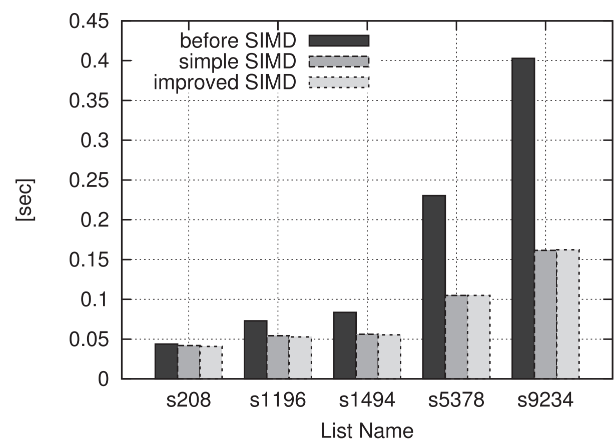

図 **10**　実行時間の比較
Fig. 10　Compare the elapsed time.

6.2　実行時間比較

2 番目の実験として 6.1 節で得られたタスクグラフを用いて Cell/B.E. 上でシミュレーションを行い，実行時間を計測した．Cell/B.E. は 2 種類のプロセッサを持っている．1 つは PowerPC Processor Element（PPE），もう 1 つは Synergistic Processor Element（SPE）である．SPE は全部で 8 基あり（実際は 1 基が冗長性確保用，1 基が OS 占有のためユーザが使用できるのは 6 基），SIMD 演算に特化された演算用コアである．PPE は汎用コアで，SPE の管理を行っている．今回の実験では SIMD 並列化効率を測るために SPE は 1 基のみを使用するものとする．また，シミュレーションは 1 テストパターンのみの実行とする．**図 10** に実行時間の比較を示す．

図 10 では，左の before SIMD グラフが元のタスクグラフを用いて計測された実行時間であり，真ん中の simple SIMD グラフが単純 SIMD 並列化後のタスクグラフを用いて計測された実行時間であり，右の improved SIMD グラフが改良 SIMD 並列化後のタスクグラフを用いて計測された実行時間である．図 10 より，どのタスクグラフにおいても単純 SIMD 並列化グラフと改良 SIMD 並列化グラフ

© 2014 Information Processing Society of Japan

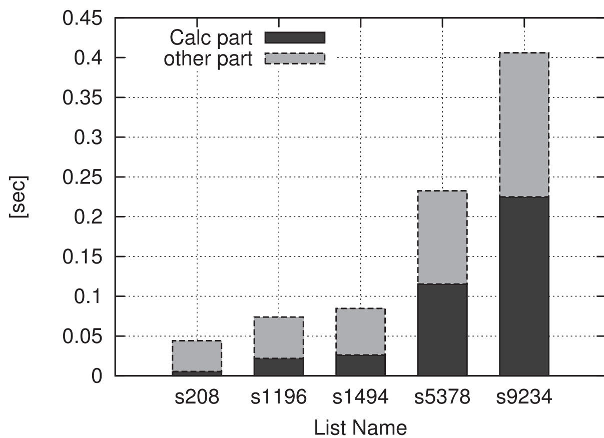

図 11　元のタスクグラフ実行時間の内訳
Fig. 11　Analysis of the elapsed time with original task graph.

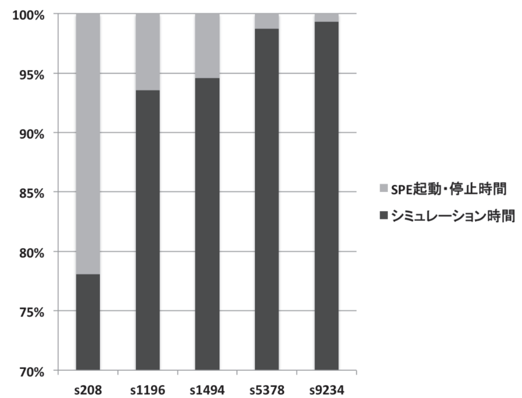

図 12　元のタスクグラフの Calc part 内訳
Fig. 12　Analysis of Calc part with original task graph.

に明確な違いは見られない．この理由は表 2 から，タスク減少率に大きな違いが見られなかったことであると考えられる．しかしながら，元のタスクグラフの実行グラフとSIMD 並列化後の実行グラフにおいても表 2 のタスク減少率ほどの違いが見られない．本来ならば実行タスク数が減少した分だけ実行時間も短縮されると考えられたが，実行時間のみを計測した結果，期待されるような結果は得られなかった．そこで我々は原因を究明するためにさらなる実験を行った．

6.3　実行時間の内訳

3 つ目の実験として実行時間の内訳を詳細にわたって計測した．実行プロセスは大きく分けて以下の 4 つである．
(1) 入力データ（タスクグラフ）読み込み
(2) SPE 起動
(3) シミュレーション
(4) SPE 終了

このうち，実際の計算部分となるのは (2)–(4) までであるため，(2)–(4) までを計算部分（Calc part），(1) をその他の部分（other part）とする．さらに計算部分（Calc part）の (2) および (4) を SPE 起動・停止時間，(3) をシミュレーション時間とする．元のタスクグラフの実行時間内訳を**図 11** に示す．

図 11 において，グラフ下の Calc part が実行プロセス (2)–(4) の計算時間，グラフ上の other part が (1) の実行時間である．次に図 11 中の Calc part の時間内訳を**図 12** に示す．

図 12 において，グラフ下のシミュレーション時間が実行プロセス (3) の実行時間，グラフ上の SPE 起動・停止時間が (2) と (4) の時間である．同様に単純 SIMD 並列化後のタスクグラフについてそれぞれ示す．改良 SIMD 並列化後のタスクグラフは表 2 および図 10 より，単純 SIMD 並列化後の実行結果と明確な差異が見られないため省略する．**図 13** に単純 SIMD 並列化後のタスクグラフの実行時間の内訳を示す．

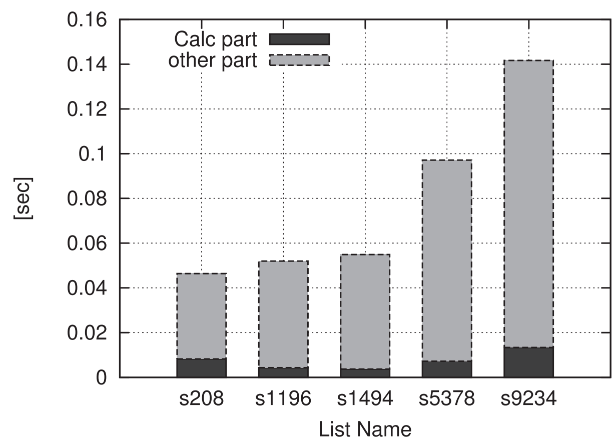

図 13　単純 SIMD 並列化後のタスクグラフ実行時間の内訳
Fig. 13　Analysis of the elapsed time with simple SIMD parallelized task graph.

図 13 より，図 11 と比較して Calc part が占める割合が減少していることが分かる．これは手順 (1) のデータ読み込みが実行時間に大きな影響を受けないことと，実行タスク数が減少したことによる手順 (3) のシミュレーション時間が減少したことが原因であると考えられる．データ読み込み部分の実行時間については入力ファイルの形を変更することでさらなる高速化の余地が残されているが，実際の論理シミュレーションでは全テストパターンを行う必要があり，SIMD 並列化・SIMD 演算によって十分にオーバラップ可能な部分であると考えられるため今回は改良の余地を残したままとした．次に，**図 14** に単純 SIMD 並列化後のタスクグラフの実行時間計算部分の内訳を示す．

図 14 より，予想どおりシミュレーション時間の割合が図 12 と比較して減少している．相対的に SPE 起動・終了時間の占める割合が増えているが，SIMD 演算に必要不可欠な処理のため省略は不可能である．またデータ読み込み時間と同様に，全テストパターンのシミュレーションで十分オーバラップ可能な時間であると考えられる．

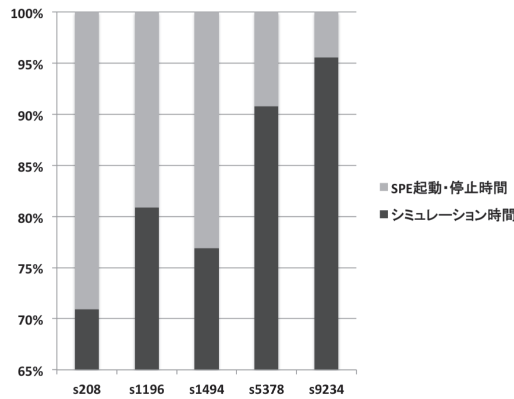

図 **14**　単純 SIMD 並列化後のタスクグラフ Calc part 内訳
Fig. 14　Analysis of Calc part with simple SIMD parallelized task graph.

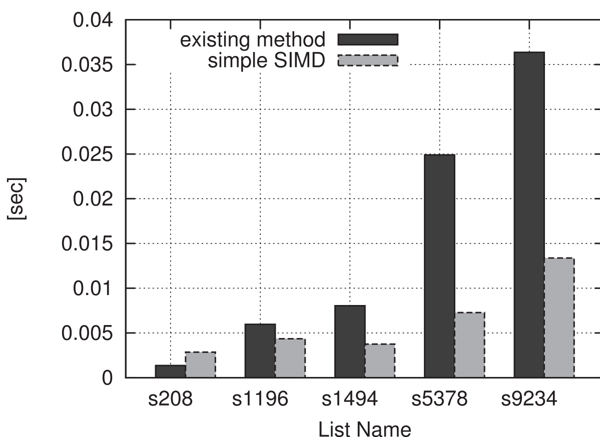

図 **15**　先行研究と提案手法の実行時間比較
Fig. 15　Compare two elapsed time, existing method with our proposal method.

6.4　先行研究との実行時間比較

追加実験として 3 章で紹介した先行研究のシミュレーション時間を計測し，本研究の実行時間と比較した．比較する本研究の実行時間は，6.1 節で得られた単純 SIMD 並列化後のタスクグラフを読み込み，Cell/B.E. 上で 1 テストパターンの実行を行ったうちの Calc part（6.2 節参照）の時間とする．先行研究の実行時間も同様に 1 テストパターンの実行を行い，シミュレーションプログラムのうち，ネットリスト構造の読み込みおよびシミュレーションに要した時間とする．なお，先行研究の実行は平等性を期するために Cell/B.E. 上で行った．それぞれの実行時間を比較した結果を図 **15** に示す．

図 15 より，s208 に関しては先行研究の方が高速であることが，その他のネットリストに関しては本研究の提案手法のほうが高速であることが分かる．特にネットリストサイズが大幅に上がる s5378 および s9234 に関しては 3 倍近く高速化が可能となっている．本提案手法においてネットリストサイズが比較的小さい s208 では，SPE 制御・通信のオーバヘッドが全体時間の 3 割を占めている．これが原因となって先行研究の実行時間と比較して遅くなってしまったと考えられる．s1196 程度のネットリストサイズを超えるネットリストならば本提案手法の方が高速であることが分かった．

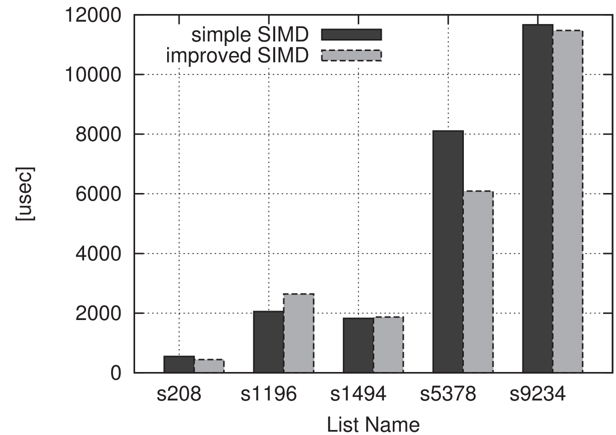

図 **16**　インテルプロセッサを用いた実行時間
Fig. 16　The elapsed time with Intel processor.

6.5　インテルプロセッサを用いた実行時間比較

インテルプロセッサを用いて 6.1 節で得られたタスクグラフを読み込み，実行時間を計測した．実行環境は Macbook Air，Intel Core i7 1.8 GHz，メモリ 4 GB DDR3 モデルである．単純 SIMD 並列化後タスクグラフ（図中 simple SIMD グラフ）および改良 SIMD 並列化後タスクグラフ（図中 improved SIMD グラフ）をインテルプロセッサを用いて実行した結果を図 **16** に示す．図 16 における縦軸は時間を表すが，単位は μ 秒である．実行には SSE などの SIMD 演算を用いず，SIMD 並列化されたタスクはループ実行を行うようにした．また，テストパターンは同様に 1 テストパターンの実行のみとした．図 10 と図 16 を比較すると単位を見て分かるが，インテルプロセッサを用いた実行がはるかに速かった．これは，1 テストパターンのみの実行では SPE 制御・通信のオーバヘッドが生じてしまい，SIMD 演算でオーバラップすることができていないことが原因の 1 つであると考えられる．他にもプロセッサの性能の差が出てしまったことなども原因としてあげられる．しかしこの実験結果から，Cell/B.E. 以外のプロセッサを用いての実行に対する期待が高まった．インテルプロセッサを用いれば SSE を使用することも可能であり，さらなる高速化が見込めると考えられる．

6.6　実験考察

ここでは 3 つの事柄について述べる．まず 1 つ目に単純 SIMD 並列化と改良 SIMD 並列化についてである．改良 SIMD 並列化は，単純 SIMD 並列化後にある一定の実行順序遅延許可範囲を定め，その範囲内でさらなる SIMD 並列化を試みる手法である．今回の実験では実行順序遅延許

© 2014 Information Processing Society of Japan

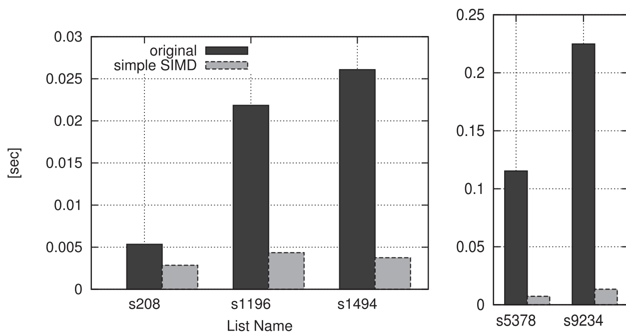

図 17　元のタスクグラフと単純 SIMD 並列化後タスクグラフにおける Calc part 実行時間の比較

Fig. 17 Compare the elapsed time of Calc part with original and simple SIMD parallelized task graph.

可範囲を 2 として行っているが，表 2 および図 10 より，単純 SIMD 並列化と改良 SIMD 並列化の間に明確な差が見られないという結果である．この理由は 6.1 節で述べたが，最大 SIMD 並列数を 16 と定めていることがあげられる．またその理由により，仮に実行順序遅延許可範囲を 2 より増やしていったとしても表 2 の改良 SIMD 並列化タスク減少率と比較して明らかに良い結果が得られるとは考えにくい．これらのことから我々は経験的に実行順序遅延許可範囲を 2 と定めた．しかし，単純 SIMD 並列化と比較して改良 SIMD 並列化がまったく改善されていないわけではない．単純 SIMD 並列化では元のタスクグラフのタスク数が比較的多い場合において高いタスク減少率を示したが，タスク数が少ない場合は同一実行順序内のタスク数が相対的に少なくなるため，タスク減少率も低くなってしまうという問題点が存在した．一方で改良 SIMD 並列化ではタスク数が少ない場合において，単純 SIMD 並列化と比較して，高いタスク減少率を示した．この理由は単純 SIMD 並列化を終えた時点では，先に述べた最大 SIMD 並列数まで達していないタスクが多数存在したからであると考えられる．そのため改良 SIMD 並列化は単純 SIMD 並列化が苦手とする，タスク数が少ないケースに対して有効な手法である．

2 つ目に 6.2 節で述べた，タスク減少率と比較して実行時間の減少率が低いことについて述べる．表 2 より単純 SIMD 並列化による s9234 のタスク減少率は 84%となっているが図 10 より s9234 の実行時間減少は 60%にとどまっている．s1494 については，単純 SIMD 並列化によりタスク減少率が 86%となっているのに対し，実行時間は 30%程度しか減少していない．6.3 節の図 11 および図 13 より，図 10 の実行時間の大部分は実行プロセス (1) のデータ読み込みが占めていたことが判明した．純粋なシミュレーション部分である実行プロセス (2)–(4) にあたる計算部分（Calc part）の実行時間を比較したグラフを図 17 に示す．図 17 より，計算部分実行時間を比較するとほとんどのタスクグラフにおいてタスク減少率と同等の実行時間短縮が見られ，期待するような結果となった．s208 のみ期待するような実行時間短縮が見られなかった理由は，図 12 および図 14 より，実行タスク数が少なすぎるため SPE 起動・終了時間のオーバヘッドを隠蔽することができなかったことがあげられる．

表 4　SIMD 並列化処理時間

Table 4 The elapsed time with SIMD parallelization.

	単純 SIMD 並列化 [sec]	改良 SIMD 並列化 [sec]
s208	0.213	3.485
s1196	4.597	198.866
s1494	6.941	254.112
s5378	104.668	16,814.423
s9234	343.252	153824.706

3 つ目に 6.1 節の SIMD 並列化処理時間を示し，実行時間と照らし合わせながら本提案手法の有用性について述べる．SIMD 並列化処理に要した時間およびその入力数を**表 4** に示す．同一実行順序内の演算子をチェックするだけの単純 SIMD 並列化と違い，依存関係を調べる必要のある改良 SIMD 並列化のほうが処理時間が長くなっていることが分かる．タスクグラフの構造に応じて処理時間が異なるため，2 つの処理時間に相関関係などは見られなかった．今回の実験で用いたネットリストの中で最も SIMD 並列化処理が長くなったものは s9234 であり，改良 SIMD 並列化では 15 万秒以上の時間がかかっていることが分かる．反対に単純 SIMD 並列化では 343 秒程度の処理時間となっている．図 17 より，s9234 においては SIMD 並列化前のタスクグラフ実行時間と比べて，単純 SIMD 並列化後のタスクグラフ実行時間が 80%以上短縮（短縮時間はおよそ 0.2 秒）されている．ここで注目すべき点は，今回の実験で行った実行は 1 テストパターンのみの実行で 0.2 秒の時間を短縮可能である，という点である．表 1 より，s9234 は非常に多くの入力ゲートを持っており，このことからテストパターンの数が膨大なものとなることは容易に予想がつく．SIMD 並列化処理によって発生する計算時間の短縮は，テストパターン数 × 短縮時間（秒）となり，SIMD 並列化処理によって生じるオーバヘッド時間をオーバラップするには十分であると考えられる．

7. まとめ

本研究では，LSI 論理シミュレーションの高速化を目的とした SIMD 並列化手法の提案を行った．提案手法ではネットリストをタスクグラフと見なすことで，回路素子を単位とした並列分散化や複数のデータを高速に計算するための SIMD 並列化が可能になり，従来の手法で問題となっていた信号線追跡の時間を削減することが可能となった．

提案手法である SIMD 並列化を単純 SIMD 並列化，改良 SIMD 並列化と並列化レベルを分けることで様々なサイズ

© 2014 Information Processing Society of Japan

のネットリストに対応可能となった．実験結果から，単純SIMD並列化はある一定以上のサイズを持つネットリストに対して効果を発揮し，単純SIMD並列化が不得意とする，比較的小さいサイズのネットリストに対しては改良SIMD並列化が効果を発揮することが判明した．ある一定以上のサイズを持つネットリストに対してSIMD並列化を施すことで，実行タスク数を87%減少させることが可能であり，実際のシミュレーション時間についてもタスク減少率と同等である86%の短縮が可能であると判明した．これらのことからSIMD並列化およびSIMD演算はLSI論理シミュレーションに対して十分な効果を発揮するといえる．

提案手法ではSIMD並列化によって得られたタスクグラフを，将来的にはタスクスケジューラを用いて実行することでさらなる高速化が期待される．タスクスケジューラとの連携は未実装であるため早急に連携可能な状態にしたいと考えている．また，改良SIMD並列化における実行遅延許可範囲についてもさらなる議論が必要である．今回実験で使用したプロセッサはCell/B.E.であるが，別のSIMD演算可能な手法（GPGPU [11]，SSE [12] など）についても実装し，Cell/B.E.による実行時間と比較したいと考えている．

謝辞　本研究の一部は科研費（基盤研究（C）課題番号：24500043）の助成を受けている．

参考文献

[1] Flynn, M.J.: Some Computer Organizations and Their Effectiveness, *IEEE Trans. Comput.*, Vol.C-21, No.9, pp.948–969 (1972).

[2] 西ノ原亮司，松浪拓海，小出　洋：大規模集積回路の論理シミュレーションのSIMD並列化手法の提案，IPSJ第52回Programming Symposium予稿集，pp.153–160 (2011).

[3] Kai, N., Nishinohara, R. and Koide, H.: A SIMD Parallelization Method for an Application for LSI Logic Simulation, *2012 41st International Conference on Parallel Processing Workshops*, pp.375–381, ICPPW (2012).

[4] S.C.E. Inc.: Cell Broadband Engine, Sony Computer Entertainment Inc. (online), available from ⟨http://cell.scei.co.jp/⟩ (accessed 2013-07-22).

[5] Hirakawa, K., Shiraki, N. and Muraoka, M.: Logic simulation for LSI, *Design Automation Conference*, pp.755–761 (online), DOI: 10.1109/DAC.1982.1585580 (1982).

[6] Taniguchi, K., Fujii, H., Kajihara, S. and Wen, X.: Hybrid fault simulation with compiled and event-driven methods, *Proc. IEEE International Conference on Design & Test of Integrated Systems in Nanoscale Technology*, pp.240–243 (2006).

[7] S.C.E. Inc.: PLAYSTATION3 Linux Information Site, Sony Computer Entertainment (online), available from ⟨http://cell.fixstars.com/ps3linux/⟩ (accessed 2013-07-22).

[8] Venkataramani, P. and Agrawal, V.D.: ATE Test Time Reduction Using Asynchronous Clocking, *2013 IEEE International Test Conference* (*ITC*), pp.1–10, IEEE (2013).

[9] Sauer, M., Kim, Y.M., Seomun, J., Kim, H.-O., Do, K.-T., Choi, J.Y., Kim, K.S., Mitra, S. and Becker, B.: Early-life-failure detection using SAT-based ATPG, *2013 IEEE International Test Conference* (*ITC*), pp.1–10, IEEE (2013).

[10] Ye, J., Huang, Y., Hu, Y., Cheng, W.-T., Guo, R., Lai, L., Tai, T.-P., Li, X., Changchien, W., Lee, D.-M., et al.: Diagnosis and Layout Aware (DLA) scan chain stitching, *2013 IEEE International Test Conference* (*ITC*), pp.1–10, IEEE (2013).

[11] Sen, A., Aksanli, B., Bozkurt, M. and Mert, M.: Parallel Cycle Based Logic Simulation Using Graphics Processing Units, *International Symposium on Parallel and Distributed Computing*, pp.71–78 (online), DOI: 10.1109/ISPDC.2010.26 (2010).

[12] Juan, C. and Canqun, Y.: Optimizing SIMD Parallel Computation with NonConsecutive Array Access in Inline SSE Assembly Language, *International Conference on Intelligent Computation Technology and Automation*, (online), DOI: 10.1109/ICICTA.2012.70 (2012).

甲斐 夏季

1989年生．2012年九州工業大学情報工学部知能情報工学科卒業．2014年同大学大学院情報科学専攻修士課程修了予定．

小出 洋　（正会員）

九州工業大学大学院情報工学研究院准教授．1997年電気通信大学大学院電気通信学研究科博士後期課程修了．日本原子力研究所計算科学推進センター研究員，九州工業大学大学院工学研究科講師を経て，2003年より現職．博士（工学）．並列分散処理，脅威トレースに関する研究に従事．

© 2014 Information Processing Society of Japan