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

西ノ原　亮司[†]　松浪拓海[†]　小出　洋[††]

本研究では，大規模集積回路の論理シミュレーション (LSI テスト計算) の計算時間の短縮を目的とした SIMD 並列化を提案する．LSI テスト計算における回路はネットリストで表現されるが，そのデータ構造は複雑であるため，並列化や SIMD 並列化は難しく，従来は入力テストデータでの分割による並列化のみ行われることが多かった．一般にネットリストは大規模になり信号線の追跡に計算時間を要するため，純計算時 (実行インスタンス) における入力テストデータ数が少なければ多くの計算時間を要してしまうことになる．提案する手法では，ネットリストで表現される回路を並列分散プログラムと考え，事前に SIMD 並列化を行う．1) 静的にネットリストを処理することから純計算時には信号線の追跡を行う必要がない，2) 同時に処理できる演算を静的にまとめることにより，SIMD 演算の効率を良くすることができる，3) タスクスケジューリング手法を含む既存のワークフロー型並列分散プログラムの効率化のさまざまな手法を応用することができる，等の利点がある．現在は Cell/B.E. や GPGPU といった SIMD 型アクセラレータを含むハードウェア・プラットホーム上に本手法を評価するためのプロトタイプシステムを実装中であるが，それらの計算時間に関する評価と既存の LSI のテスト計算との比較について議論を行う．

A Proposal of SIMD Parallelization for a Logic Simulation of Large Scale Integration

Ryoji Nishinohara,[†] Takumi Matsunami[†] and Hiroshi Koide[††]

In this paper, we propose a SIMD parallelization approach to reduce the computation time of a logic simulation of Large Scale Integrated Circuit (a test calculation for LSI). This curcuit is expressed as a net list. The simulator receives a set of test data to the circuit and compares the result with the expected value. The parallelization approach is only done by the input data set in this kinds of simulation and any SIMD parallelization approaches is not usually employed, because the net list is expressed by very complicated data structures. Generally speaking, the net list is usually very large, and it takes a long computation time to chase signal lines. If a set of input data set is relatively small at a execution time instance, it needs a long computation time. The proposed method thinks the net list which expresses circuit as a parallel and distributed program and it processes the net list based on a SIMD parallelization approach. The proposed method has various merits like following. 1) It does not need to chase signal lines at the computation time instances, because it processes the net list statically beforehand. 2) It can collect the same type of logical calculation to improve the efficiency of the SIMD parallelism. 3) We can apply many ordinary techniques to improve the elapse time of parallel and distributed work flow programs. We are implementing a prototype system on a hardware platform contains Cell/B.E. and/or GPGPU SIMD accelerators to evaluate the proposed method. We discuss the comparison between the proposed method and the ordinary LSI test computation about the elapse time.

1. はじめに

本研究では，大規模集積回路 (Large Scale Integration, LSI) の設計における，論理シミュレーションの組み合わせ回路テストを，SIMD (Single Instruction Multiple Data) 並列化し，さらにタスクスケジューリングの手法を用いることで高速化することを提案する．

タスクスケジューリングの手法は，一般に，互いに依存関係があるようなタスクグラフによって表現できる並列分散アプリケーションにおいて，その計算時間の短縮を目的に利用される．提案する手法では，ネットリストによって表現される LSI の組み合わせ回路を並列分散プログラムとみなす．すなわち，組み合わせ回路内で行われる論理演算のそれぞれをタスク，論理回路素子間の信号線接続をタスク間の依存関係とみなすことで，タスクスケジューリングの手法を用いて高速化する．

† 九州工業大学 情報工学部
†† 九州工業大学大学院 情報工学研究院

この手法には，

- 事前の SIMD 並列化によって静的にネットリストを処理するため，純計算時 (実行時インスタンス) では信号線を追跡する必要がない，
- 静的にネットリストを処理する際，同時に処理することができる演算をまとめておくことによって SIMD 並列化の効率を高められる，
- タスクスケジューリング手法を含む，ワークフロー型並列分散アプリケーションの並列化に関する既存の効率化手法を適用できる，

という利点がある．

本論文では，現在実装中のプロトタイプシステムについて，

- Cell/B.E. 上で SIMD 並列化した論理演算と，従来の方法における論理演算の部分とを比較し，
- 動的にネットリストを処理している従来の方法において，信号線の追跡に要する計算時間がどのくらいの割合を占めているかを確かめる，

ということによって，提案する手法により論理シミュレーションが高速化できることを見積もった．前者については，論理演算を Cell/B.E. 上の SPE を用いて SIMD 並列化した場合，同じ計算を PPE 上で行った場合に比べて 2 倍から 3 倍高速化することができることが分かった．後者については，従来の手法においては，論理演算がシミュレーション全体のおよそ 4 割を占めるものの，信号線追跡にも多くの時間が割かれており，信号線追跡を削減する提案手法によって高速化が見込めることが分かった．

本論文の構成は以下の通りである．第 2 章では，本研究の背景となる，LSI の論理シミュレーションとその既存の手法について述べる．第 3 章では，SIMD 演算と SIMD 並列化についての説明をした後，SIMD 演算に強いプロセッサについて紹介する．第 4 章では，今回の実験で用いた Cell/B.E. について詳しく述べる．また，実装したプログラムで用いた，ループ展開とダブルバッファリングに関するプログラミング技法についても述べる．第 5 章では，提案する手法について，従来の手法と比べて述べる．第 6 章では，行った評価実験について説明する．まず従来の手法について，信号線の追跡が計算時間のオーバーヘッドの一つとなることを確かめ，次に SIMD 並列化の有効性について確かめる．第 7 章では，本論文のまとめと，今後の取り組みについて述べる．

2. 本研究の背景

昨今の LSI に集約される論理回路素子の数は，集積技術の進歩によって増加の一途をたどっている．新たな LSI を開発するにあたっては，その LSI の設計段階において，ある入力に対する LSI の出力が，開発者の意図するものであることを確かめる必要がある．この確かめの作業は計算機を用いて行われることが多く，LSI の論理シミュレーションと呼ばれる．

LSI の論理シミュレーションは，内部状態を持ち，過去の状態に依存して出力が変化する順序回路と，そうでない組み合わせ回路とに切り分けて行われる．このうち，組み合わせ回路のテスト計算は，論理積 (AND)，否定論理積 (NAND)，論理和 (OR)，否定論理和 (NOR)，否定 (NOT) 等の論理素子で構成されたループのない回路に，テストデータ☆を入力したときの回路の出力値と，開発者の期待する出力値とを比較することにより行われる．テストの対象となる回路は，それぞれの信号線と素子の関係を表すリスト (ネットリスト) によって表現されることがあるが，ネットリストを表すデータ構造は複雑で，並列化による処理の高速化を行うことは難しい．このため，従来の論理シミュレーションの並列化手法においては，入力するテストデータの範囲を分割することで並列化する等の手法をとることが多かった．

しかし，一般にネットリストは，LSI が大規模になるほど信号線の追跡に多くの計算を必要とする．このため，前述の方法による並列化は，昨今の LSI の論理シミュレーションにおいては，追跡する信号線とテストデータがどちらも増加しているという問題に効果的に対処できない．また，昨今の LSI に集約される論理回路素子の数が非常に多いことから，論理シミュレーションの実行時には，大量の論理演算を高速に実行する必要がある．

本研究の提案する手法では，事前にネットリストを静的に処理し，SIMD 並列化することによって，LSI の大規模化に伴う信号線追跡の繰り返しをなくすことで，この問題に対処する．また，タスクスケジューリングの手法を用いることで，複数の計算機を動的スケジューリングにより活用し，計算の効率化を図る．

3. SIMD 並列化

SIMD 演算は，一つの命令でいくつかの数のデータに対して一度に処理を行う演算の一方式である．(SIMD 命令でない) 汎用な命令は，汎用のソースレジスタと汎用のデスティネーションレジスタにそれぞ

☆ テストデータは，検証対象の論理回路に対して外部から入力する信号列のシミュレーションパターン集合である．

れ与えられる単一のデータに対して，定められた処理を行った後，デスティネーションレジスタに結果を格納するという処理を行う．それに対して SIMD 命令では，ソースレジスタとデスティネーションレジスタとして専用の SIMD レジスタを持ち，レジスタに格納された連なる複数のベクトルデータに対して，定められた処理を一度に行う．旧来の汎用な命令による演算を，SIMD 演算に置き換えることを SIMD 並列化といい，SIMD 演算に適した処理を SIMD 並列化することによって，効率化を図ることができる．

3.1 SIMD 演算に適した処理

SIMD 演算の特色は，長いベクトル長を持つ SIMD レジスタに格納されたデータを，一命令で一度に，画一的に処理できるということにある．演算するデータの長さが SIMD レジスタ長に満たない場合は，その分だけ演算時間にロスが生じてしまう．このため，演算するデータは，出来るだけまとめて SIMD 演算するというのが普通である．特に条件分岐や繰り返しは，その条件の判定に逐次的な処理を含み，SIMD 演算の効率を下げてしまうため，これらを取り除く (可能な限り必要のない形にする) ことによって，さらに効率化を図ることができる．

3.2 ヘテロジニアスマルチコアプロセッサ

昨今のマイクロプロセッサは，回路の複雑化，高密度高集積化，高クロック化のために，消費電力と発熱量の増大が問題視されており，単一のプロセッサコアの性能向上はすでに頭打ちの状況にある．このため現在では，一つのプロセッサに複数個のコアを搭載し，並列処理を行うことによって全体としての処理能力向上を狙う，マルチコア構成のプロセッサが主流となっている．

特に，ヘテロジニアスマルチコアと呼ばれる新しいプロセッサの構成は，単体で高性能化された制御用のプロセッサコアを 1 個から数個，SIMD 演算などの特定の処理を高性能化した多数の演算用のコアを同時に含むことによって，多様なタイプのプロセッサコアを，スケジューラにより適材適所で有効利用することが出来る．また，複雑化し過ぎたプロセッサコアの構造を単純化することによって，高クロック化，パイプライン化と，省電力と発熱量の低減の両立を実現している．ヘテロジニアスマルチコアプロセッサは，異なるアーキテクチャのプロセッサコアを組み合わせて一つのプロセッサを構成することにより，アーキテクチャ毎の弱点を補い合って，アプリケーション全体の処理効率を高めることができる．

例えば，ソニー・コンピュータエンタテインメント社が開発したヘテロジニアスマルチコアプロセッサである Cell Broadband Engine (Cell/B.E.) は，PowerPC アーキテクチャのプロセッサコア 1 個と，SIMD 演算の性能に優れた演算用のコア 8 個を有している．これにより，消費電力や発熱量を抑えながらも，従来のプロセッサに比べて高速に加減算や論理演算のような基本演算処理を，複数同時に行うことが可能である．

3.3 GPGPU

GPGPU (General-Purpose computing on Graphics Processing Units, グラフィックス・プロセッシング・ユニットによる汎目的計算) とは，本来は画像処理を行うための補助演算装置である GPU の演算能力を，画像処理以外の汎用な目的に利用できるようにするための技術のことである．最近のデスクトップアプリケーションでリアルタイムに行われる画像処理は，3 次元描画を含むレンダリングアルゴリズムの高度化と，画面の高精細化・高解像度化，高フレームレート化によって，非常に負荷が高くなっている．

これらのアプリケーションの要求に応えるために，GPU には高性能なメモリと強力な演算器を集積されている．また，デスクトップアプリケーションで行われる 3 次元グラフィックスの描画には，行列演算を中心とした SIMD 演算や，浮動小数点の演算が非常に多く含まれ，描画性能の向上のために，GPU は SIMD 演算と浮動小数点演算の性能が特に高められている．また，GPU ベンダーから GPGPU のための技術情報，プログラミング環境，フレームワーク等が公開され，計算負荷の高いさまざまな実用アプリケーションにも応用され始めている．[1] 特に，ハイパフォーマンスコンピューティング (HPC) の分野で注目され，科学技術計算を行う PC クラスタに搭載される機会が多くなっており，最近はスーパーコンピュータにおいても計算ノードの構成に含まれることが多い．

4. Cell Broadband Engine

この章では，Cell/B.E. のアーキテクチャ及び Cell/B.E. 上のプログラミングで有効な技法について述べる．

4.1 Cell/B.E. のアーキテクチャ

ソニー・コンピュータエンタテインメント社が開発，販売を行っている，PLAYSTATION 3 に搭載されている Cell/B.E. は，汎用プロセッサである PowerPC Processor Element (PPE) を 1 基，演算用のプロセッサである Synergistic Processor Element (SPE) を 8 基持つ，ヘテロジニアスマルチコアプロセッサである．各プロセッサはメインメモリや入出力デバイス，

グラフィックデバイスと，Element Interconnect Bus (EIB) と呼ばれる高速なバスで接続されており，各プロセッサは EIB を経由してデータアクセスを行う．Cell/B.E. の内部構成を **図 1** に示す．

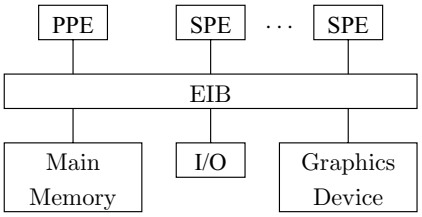

図 1 Cell/B.E. の内部構成
Fig. 1 Structure of Cell/B.E.

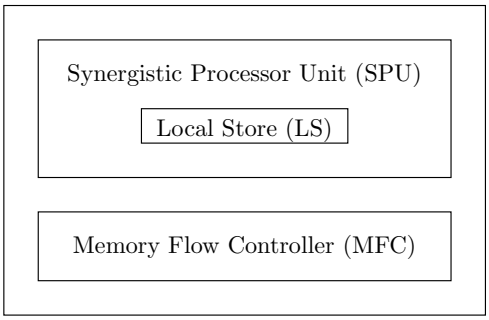

図 2 SPE の構成
Fig. 2 Structure of SPE.

4.1.1 PowerPC Processor Element

PPE は 64 bit Power アーキテクチャの汎用的なプロセッサであり，従来のプロセッサと同様に OS の駆動等を行うことが可能だが，特に Cell/B.E. においては，演算を行う SPE を制御する役割を担っている．

後述する実験では，Power アーキテクチャ上の Time Base Register (TBR)[2] を用いて計算時間の計測を行った．

4.1.2 Synergistic Processor Element

SPE は SIMD 型アーキテクチャを採用しており，SIMD 演算に特化している．SPE は 演算装置である Synergistic Processor Unit (SPU) と Memory Flow Controller (MFC) から構成されている．SPE の構造を **図 2** に示す．SPE はメインメモリに直接アクセスできないが，それぞれの SPE が SPU 内に 256 KiB の Local Store (LS) と呼ばれる専用のメモリを持つ．そのため，SPE がメインメモリを参照するときは，Direct Momory Access (DMA) を用いてメインメモリと LS との間でデータ転送を行う．この転送処理は MFC が行う．

SPU にはプロファイリングのために利用できる，SPU Decrementer[3] というレジスタがあり，後述の実験ではこのレジスタを用いて計算時間の計測を行ったため，ここでこのレジスタについて説明する．

4.2 プログラミング技法

ここでは，実験で用いた，Cell/B.E. 上のプログラミングで特に有効な技法として，ループ展開，ダブルバッファリングについて説明する．

ループ展開

ループ展開とは，繰り返しの処理を展開してループ回数を減らすことで，ループ制御によるオーバーヘッドを減らし，プログラムの実行を高速化する手法である．SPE は SIMD 演算に特化しているため，プログラムの制御を得意としていない．ループ展開を行うことで，その制御を減らすことができ，SIMD 演算を効率よく行うことができる．そのため，SPE 上の計算で特に有効な技法である．

ダブルバッファリング

ダブルバッファリングとは，データを格納するバッファを 2 個用意し，バッファを入れ替えながら処理とデータ転送を並列に行うことで入出力の待ち時間を削減し，実行を高速化する手法である．PPE と SPE との間の DMA 転送は，SPE 内の MFC が処理を行うため，データ転送中に SPU は他の処理を行うことが可能である．ダブルバッファリングを用いて，DMA 転送と演算を並列に行うことで，SPE の通信待ち時間を削減することができる．

5. 提 案 手 法

この章では，本研究で提案する手法について，従来の手法との比較を行いながら説明する．

5.1 従来の手法

LSI の論理シミュレーションにおける組み合わせ回路のテスト計算では，ネットリストによって表現される信号線と論理回路素子との関係を参照しながら，組み合わせ回路の入力部分にいくつかのテストパターンを入力することによって得られる出力と，設計者が期待する組み合わせ回路の出力を比較し，それらが合致することを確かめる．従来の手法では，

(1) 組み合わせ回路のネットリストを参照する，
(2) 入力信号のテストパターンを読み込む，
(3) 論理演算によって出力信号を求める，

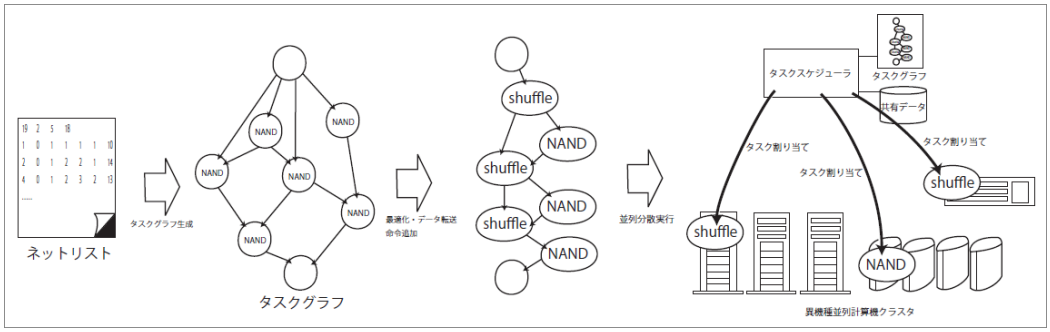

図 3 提案システムの概略図
Fig. 3 Overview of Proposed System

(4) 求まった出力と期待する出力とを比較する，

という処理を逐次的に繰り返すことによって，論理シミュレーションを行っている．

この処理の並列化に対するこれまでのアプローチとして，まったく同じ論理シミュレーションを行える計算機を複数台用意し，入力信号のテストパターンをいくつかに分割して，それぞれの分割部分を異なる計算機に割り当てるというものがある．このアプローチでは，組み合わせ回路の規模が大きくなったとき，例えば回路に含まれる論理素子の数が 2 倍になったときに，入力信号のテストパターンの数は 4 倍となるため，計算時間を据え置きたい場合には，同じ性能の計算機が 4 倍の数必要になる．しかし，このアプローチは提案手法においても応用可能で，入力信号のテストパターンを与えて計算させることが出来るシステムを複数用意し，これに対して範囲を分けたテストパターンを割りつけることによる並列分散化も考えられる．

5.2 提 案 手 法

本研究では，従来の手法の，組み合わせ回路の大規模化により計算時間が大きく増加するという問題を改善する手法を提案する．提案手法の概略図を **図 3** に示す．提案手法では，

(1) 組み合わ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) タスクグラフを SIMD 並列化する，
(3) 動的スケジューリングを用いてタスクを計算機に割り当てる，
(4) タスクを SIMD 演算を用いて計算する，
(5) 求まった出力と期待する出力とを比較する，

という処理を行う．タスクグラフの生成と SIMD 並列化は事前に静的に行い，タスクスケジューリングを動的に行うことで，論理シミュレーションの高速化を行う．以下，この手順を詳しく説明する．

5.2.1 タスクグラフの生成

タスクとは，プログラム内のループやサブーチン等のまとまった処理単位のことである．プログラムのそれぞれのタスクの間には，データ依存等の実行順序に関する制約があり，プログラムはこの依存関係を用いてタスク間を繋いだグラフとして表すことができる．これをタスクグラフと呼ぶ．このように，プログラムをタスク単位に分割し，ネットワークで接続された計算機資源にタスクを割り当てることで，並列分散計算を行うことができる．

提案手法では，組み合わせ回路を，その論理演算をタスク，論理回路素子間の信号線を依存関係としたタスクグラフと見なす．従来の手法では，シミュレーション時にすべての計算機が信号線の追跡を行うが，この方法では，信号線の追跡はタスク割り当てを行うより以前の一回のみでよい．そのため，事前にネットリストを静的に処理し，タスクグラフを生成することで，シミュレーション時の信号線の追跡を削減することができる．また，回路をタスクグラフとして扱うことで，以下で説明するタスクスケジューリングの手法を用いて並列分散計算を効率よく行うことができる．

5.2.2 SIMD 並列化

同じ論理演算を行うタスクは，互いに依存関係が無ければ，それらをまとめて処理することで，タスク割り当ての時間を削減できる．また，まとめたタスクは，SIMD 演算によって効率よく計算することができる．

5.2.3 タスクスケジューリング手法を用いた高速化

前述のように，タスクを計算機資源に割り当て，並列分散計算を行う際，そのタスクの割り当て方によって計算時間が大きく変化する．この計算時間を短くするため，タスクをどの計算機に割り当てるか，スケジューラが適切に選択する必要がある．このようなタスクの割り当てをタスクスケジューリングと呼び，計

算時間の短縮化等のため，様々なスケジューリング手法に関する研究が行われている．

提案手法では，タスク割り当てを動的にスケジューリングすることで，論理シミュレーションの計算時間を短縮する．動的スケジューリングとは，タスク割り当ての際に，手法ごとに決めたポリシーに従って，タスクを割り当てる計算機を決定するスケジューリング手法である．動的スケジューリングでは，タスクを割り当てる計算機を決定する際に計算時間が必要になるが，この時間は経験から短く済ますことが可能である．以下に動的スケジューリングの例を挙げる．

クリティカルパス (CP) 法

クリティカルパス (CP) 法[4)5)] とは，クリティカルパス上のタスクを優先的に割り当てるスケジューリング手法である．このスケジューリング手法では，タスクを割り当て時に，その時点でのクリティカルパスを計算し，タスク割り当てを動的に行う．

5.2.4 SIMD 演算による高速化

提案手法では，割り当てられたタスクを処理する際，SIMD 演算を用いることで論理演算の高速化を行う．SIMD 演算を用いることで，多くのテストパターンを効率よく計算することができる．

5.2.5 処理の最適化

以上のような高速化の手法に加えて，提案手法では，計算機に合わせて SIMD 最適化や並列化コンパイラによる最適化等，様々な最適化を行うことができ，タスクの処理を高速化することができる．以下に，例として挙げた 2 つの最適化を説明する．

SIMD 最適化

SIMD 最適化とは，計算時に積極的に SIMD 命令を用いる，メモリのアラインメントを調整する等，SIMD 演算を効率よく行えるようにすることである．プログラムのソースコード自体に変更を加える手法，また，コンパイラが自動的に行うものがある．

並列化コンパイラ

並列化コンパイラ[6)] とは，複数のプロセッサを持つ計算機等，同時に複数の演算処理を行える環境で，並列に処理を行うように，自動的にコードを変換するコンパイラである．並列に処理を行うプログラムでは，複数の処理が同じデータ領域を用いる等，排他制御が必要となる．このようなプログラムのコードを書くことは非常に手間がかかり，並列処理を最適に行うことは難しい．並列化コンパイラを用いて自動的にコードを変換することで，この手間を無くし，最適に並列処理を行うことができる．

6. 評 価 実 験

提案手法の評価のため，以下の 2 つの実験を行った．

(1) 従来の手法を用いた，論理シミュレーションに要する計算時間の計測，

(2) 提案手法における，論理演算の計算時間の計測．

前者は信号線追跡の時間に関する評価，後者は SIMD 演算の高速化の評価を目的として実験を行った．

6.1 従来の手法の計算時間測定

この実験では，従来の手法における計算に必要な時間を計測した．従来の手法では，計算を行う際に毎回信号線の追跡を行っており，何度も同じ処理を繰り返すという無駄がある．この実験では従来の手法の処理全体に要する時間と，論理演算に要する時間を計測し，信号線追跡が処理時間に占める割合を調べた．

6.1.1 実 験 内 容

実験は**表 1** に示す環境で行った．この実験では，ネットリストを読み込み，表として管理し，この表を参照することで信号線の追跡を行った．プログラム実行後，gprof[7)] を用いてプロファイリングを行い，処理全体に要した時間，論理演算部分に要した時間をそれぞれ計測した．

表 1 実験環境 (従来の手法)
Table 1 Experiment Environment of Ordinary Method

プラットフォーム	Sun SPARC Enterprise T5440
OS	Solaris 10 OS 10/08
CPU	UltraSPARC T2 Plus 1.2 GHz (8 core), 4 CPU
メインメモリ	128 GiB

6.1.2 実 験 結 果

この実験の結果を**表 2** に示す．従来の手法においては，論理演算は実験全体のうち約 35 % を占めていることが確認できた．論理演算部分以外に要した時間 1.28 秒のうち，I/O 関係の処理 (ネットリストやテストパターンの読み込み，結果の出力) を除いた時間は，主に信号線の追跡に要した時間である．この時間は，提案手法を用いることで削減できる．また，本実験で論理演算に要した時間 0.69 秒について，提案手法では SIMD 並列化して論理演算を行うため，この時間を短縮することができる．

6.2 提案手法の計算時間測定

提案手法では，全体の処理のうち，論理演算の部分を SIMD 並列化し，さらに SIMD 演算に特化したプロ

表 2　従来の手法の計算時間
Table 2　Computation Time of Ordinary Method

	実験全体の計算時間	論理演算の計算時間
時間 [sec.]	1.97	0.69

セッサを用いて，SIMD 演算を高速に行うことで全体の計算時間を短縮する．この実験では，SIMD 演算に特化したプロセッサである SPE を搭載した Cell/B.E. を用いて，提案手法の計算時間を計測した．

6.2.1 実験内容

実験は**表 3** に示す環境で行った．実験では，PPE と SPE のそれぞれで SIMD 命令セットを用いた AND 演算を行うため，32 MiB のテスト用ベクトルデータ 2 つを用意する．この 2 つのテスト用ベクトルデータの AND をとる演算を行い，その計算時間を計測した．SPE の計算ではダブルバッファリングを用いたため，処理全体の計算時間と計算部分のみの両者を計測し，比較することでダブルバッファリングの効果を調べた．同様に，ループ展開に関して，展開を行う場合と行わない場合とを比較し，ループ展開による計算時間の減少を調べた．それぞれ計算時間の計測には，PPE では TBR，SPE では SPU Decrementer をそれぞれ用いた．

表 3　実験環境 (提案手法)
Table 3　Experimental Environment of Proposed Method

プラットフォーム	PLAYSTATION 3 (CECHB00)
OS	Yellow Dog Linux 6.2
Kernel	Version 2.6.29-3.ydl61.4
CPU	Cell/B.E. 3.2 GHz
メインメモリ	256 MiB

6.2.2 実験結果

この実験の結果を**表 4** に示す．SPE の処理全体の時間と計算部のみの時間とを比較すると，その差は 1 ミリ秒以下であり，ダブルバッファリングを用いることで，PPE と SPE との間の DMA 転送に要する時間は計算時間でオーバーラップが可能であることが確認できた．また，PPE と SPE の計算の速さを比較すると，ループ展開を行う場合では SPE は PPE の約 3.1 倍，ループ展開を行わない場合は約 2.6 倍の速さとなっている．このことから，SPE のような SIMD 演算を高速に行うことができるプロセッサを用いることで，論理演算を高速に行え，提案手法の SIMD 並列化が有効であることが確認できた．SIMD 演算では，ループ展開により SIMD 演算器を有効に利用すると効果的であるから，本実験で実際にループ展開を行う場合と行わない場合を計測し，比較したところ，PPE，SPE はどちらも計算時間が短くなり，ループ展開が SIMD 演算の高速化に効果的であることが確認できた．

表 4　PPE，SPE の計算時間 (ミリ秒)
Table 4　Computation Time of PPE and SPE (msec.)

	ループ展開有	ループ展開無
PPE	151.9	189.2
SPE (処理全体)	48.6	73.8
SPE (計算部)	47.9	73.7

7. おわりに

本研究では，LSI テスト計算の高速化を目的とした SIMD 並列化手法を提案し，その評価を行った．従来の手法の計算時間を測定した結果から，従来の手法は信号線追跡が計算時間の多くを占めていることがわかり，提案手法ではこの信号線追跡の時間が不要になるため，論理シミュレーションの高速化が可能であることがわかった．また，提案手法の計算時間測定の結果から，Cell/B.E. をうまく用いることで論理演算を高速に行うことが可能であり，提案手法で論理シミュレーションの計算時間を減らすことが可能であることの見積もりができた．提案手法では動的スケジューリングの計算時間が必要になるが，1 スケジューリングタイムインスタンスあたりに必要な時間は，経験的に 1 ミリ秒程度であり，SIMD 演算とのオーバーラップも可能である．

今回の実験では，Cell/B.E. を用いて提案手法の評価を行ったが，今後は SIMD 型アクセラレータを含む他のプラットフォームとして，GPGPU について実験を行い，提案手法の評価を行う予定である．また，タスク割り付け時のデータ通信の時間について実験を行い，タスクスケジューリングを含めた議論を行いたい．

謝　辞

本研究は文部科学省科学研究費補助金 (課題番号 21500039) および総務省の助成を受けている．

参考文献

1) NVIDIA Corporation: NVIDIA CUDA C Programming Guide 2.3.1，pp.1-5 (2009).
2) International Business Machines Corporation: 64 ビット Linux での Power Architecture の Time Base Register，2007.

http://www.ibm.com/developerworks/jp/linux/library/pa-timebase/, 2010 年 11 月 29 日確認.
3) Sony Computer Entertainment, Inc.: Cell Broadband Engine アーキテクチャ Ver. 1.02, pp.146-147 (2007).
4) 諏訪和徳, 立川賢治, 曽禰元隆: 通信時間を考慮に入れた CP 法を用いたタスクスケジューリング手法, 電子情報通信学会総合大会講演論文集. 情報・システム, Vol.1, p.81 (1996).
5) KWOK Y.: Dynamic critical-path scheduling: An effective technique for allocating task graphs multiprocessors, IEEE Trans. Parallel and Distributed Systems, Vol.7, No.5, pp.506-521 (1996).
6) 笠原博徳: 最先端の自動並列化コンパイラ技術, 情報処理, Vol.44, No.4, pp.384-392 (2003).
7) Jay Fenlason, Richard Stallman: GNU gprof (1998)
http://ftp.gnu.org/old-gnu/Manuals/gnuprof-2.9.1/, 2010 年 11 月 29 日確認.