DESCARTES:順序回路を対象とした実数値シミュレーションに基づく並列テスト生成システム

伊達 博　中尾 教伸　畠山 一実
(株) 日立製作所日立研究所

概要

順序回路中の単一縮退故障に対し，並列処理により高速にテスト生成するシステム DESCARTES を提案する．DESCARTES の特徴は，多くの並列性をもつ実数値シミュレーションを用いた並列テスト生成法にある。本論文では，システム構築上の課題と並列化方式とについて述べ，ワークステーションネットワーク上で動作するプログラムに関する，ISCAS89 ベンチマークデータによる評価結果を示す。

Abstract

This paper presents DESCARTES, a test generation system for synchronous sequential circuits using a genral purpouse parallel computer. The feature of DESCARTES is to parallelize the simulation-based test generation method, based on 'real-valued logic simulation'. The program runs on a network of workstations connected through ethernet. Experimental results for ISCAS'89 benchmark sequential circuits illustrate the efficiency of this approach.

1　はじめに

LSI の大規模·高機能化と共に設計期間が増大し，大規模データを扱える高速な設計自動化システムが望まれている。従来，逐次型計算機上のシステムでは，これらの要求に対し，解くべき問題を階層的に扱うなどして対処してきたが，解の品質，処理速度ともに限界に近づきつつある。LSI テストの分野でも，論理 LSI の大規模化，高機能化に伴い，検査用テストパターンの作成は困難さを増しており，ゲートアレイ等の ASIC にとっては，設計期間を短縮するうえで大きな障害となると予想される。一般に論理 LSI は，組合せ回路部分と記憶素子をもつ順序回路であり，テスト生成をする場合，時刻に依存して各ゲートにおける入出力値の変化を考慮しなければならない。そのため順序回路に対する自動テストパターン生成 (ATPG) は，VLSI 設計における最も計算パワーを必要とする処理の一つとなっている。

現在，順序回路のテスト生成を容易にするため，論理 LSI の多くはスキャン回路設計等のテスト容易化設計を取り入れ，順序回路のテスト生成問題を組合せ回路の問題に置き換えて扱っている。しかしながら性能重視の RISC プロセッサやマイコン，ゲートアレイ等の ASIC においてはスキャン回路のオーバヘッド削減の要望があるため，高速に高品質なテストパターンを生成できる順序回路用テスト生成システムが望まれている。

順序回路を対象とした ATPG としては，大きく分けて二種類ある。一つは，組合せ回路のテスト生成法である D アルゴリズム [21] や PODEM [10] を時間軸展開の概念を導入して拡張し，ある一つの故障に関して，それを検出するためのテストパターンを決定論的に構成して行く方法 (algorithmic 方式) である。もう一つは，適当な初期パターンを与え，シミュレーションに基づき，それをテストパターンとなるように変形して行く方法 (simulation-based 方式) である [2, 5, 11, 12]。しかしながら，いずれの方式も大規模回路に適用するためには，高速化が必要なため，これまで多くの研究者によって，並列処理利用したテスト生成法の研究が進められてきた。プログラムの並列化方式としては，故障並列，探索空間並列，方策並列，機能並列などが提案されている。

故障並列では，テストの対象となる故障のリストを

"DESCARTES:A Parallel Test Generation System for Sequential Circuits Based on Real-valued Logic Simulation" by Hiroshi DATE, Michinobu NAKAO and Kazumi HATAYAMA Hitachi Research Laboratory, Hitachi Ltd.

分割し，各プロセッサに分担させてテスト生成を行う。Patil と Banerjee [19] は，故障には，検出しやすい故障と検出しにくい故障とがあり，複数の検出しにくい故障が同じプロセッサに割り付けられる傾向にあると，負荷バランスが崩れることを指摘している。この問題に対して，アイドル状態になったプロセッサは，もっとも多くの故障リストを担当しているプロセッサに要求を出し，故障リストの半分を分担することにより動的負荷バランスを実現している。また，Fujiwara と Inoue [9] は，並列処理システムにおける故障分割に基づくテスト生成をモデル化し，一回の通信でプロセッサに割り当てられる部分問題の最適粒度と台数効果について解析している。Agrawal 等 [1] は，GENTEST [6] を故障並列により並列化し，ワークステーションネットワーク上で評価した結果を報告している。

探索空間並列では，テスト生成処理における木探索を複数のプロセッサで行う。Patil と Banerjee [20] は，分割統治法に基づく PODEM を実装し，検出しにくい故障に対する有効性を示している。

故障並列と探索空間並列とを組合せた順序回路用テスト生成システムとして，ProperHITEC [18] がある。このシステムでは，並列プログラムの開発効率についても言及している。オブジェクト指向プログラミングを採用することにより，逐次プログラムと並列プログラムとで共有できる部分をなるべく多くすることで，逐次プログラムに加えられた改良点が，すぐに並列プログラムに反映されるようにしている。

方策並列では，異なるアルゴリズムに基づき，各プロセッサでテスト生成を行う。Chandra と Patel [4] は，いずれか一つのプロセッサが，テスト生成に成功したら他のプロセッサに処理を終了させるメッセージを送信する方式について検討し，そのオーバヘッドについて評価している。

機能並列 (AND 並列) では，独立に動作しうる複数の処理にテスト生成処理を分割し，それらを各プロセッサで実行する。Motohara [16] は，PODEM プログラムと故障シミュレーションを別々のプロセッサで実行させるシステムについて評価を行っている。

以上の並列化方式に関するは報告では，大規模回路への対処法に関する言及はされていない。回路規模が大きくなると，ハードウエアの制限から回路情報を各プロセッサのメモリ上に分担して持たせる必要が出てくる。

トポロジカル並列では，回路情報を分割し，各プロセッサに分配して持たせ，プロセッサ間で通信しながら，テスト生成を行う。回路分割の方法としては，論理シミュレーションに関して検討されている [13, 22]。テスト生成に関しては，Kramer [14] がコネクションマシンの各プロセッサをゲートに対応させた方式について評価を行っている。

以上述べた，これまでのテスト生成処理の並列化に関する研究の多くは，algorithmic 方式に分類されるテスト生成法に関するもので，simulation-based 方式に関する研究は，ほとんど報告されていない。

本論文では，順序回路中の単一縮退故障を高速に検出するために，simulation-based 方式に分類される実数値シミュレーションに基づくテスト生成法を基本アルゴリズムとし，多くの並列性を抽出することを可能にしたテスト生成システム DESCARTES を提案する。

以下では，2. で実数値シミュレーションに基づくテスト生成法について述べ，3. で，並列テスト生成システム DESCARTES における課題とその並列化方式について述べる。4. で，本システムの評価項目にまとめ，並列処理用システムソフトウエア Express を用いた並列プログラムをワークステーションネットワーク上で動作させ，ISCAS'89 のベンチマークデータにより評価した結果を示す。

2 実数値シミュレーションに基づくテスト生成

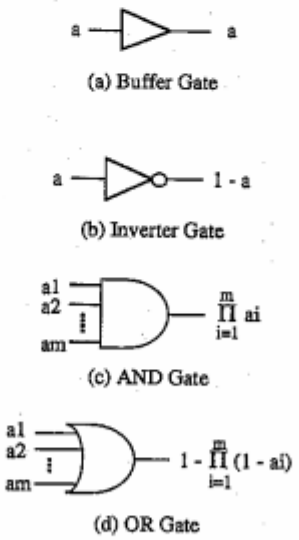

図 1: 実数値シミュレーションの例

実数値シミュレーションに基づくテスト生成 [11, 12] では，図 1に示すように，論理回路における各ゲートの演算処理を実数値に拡張する。そしてある故障を検出するためのテストパターンと任意に与えられた入力パターンが，どの程度違うのかを表すコストを定義する。そして実数値の収束計算処理を繰り返すことにより，与

えられた入力パターンをテストパターンへと近づけて行く。

以下では，収束計算処理とそれを用いた順序回路テスト生成処理の概要について述べる。

2.1 収束計算処理

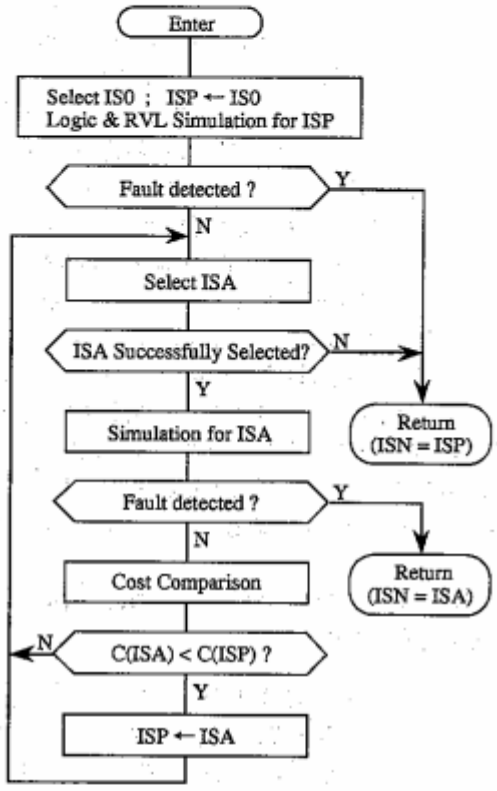

図 2: 収束計算処理

検出すべき故障 fが与えられると，図2に示す手順で収束計算処理が行われる。最初に，収束計算処理の対象となるパターン系列 ISPとして，$IS0$が選ばれると，パターン系列 ISPに対して，正常時の論理シミュレーションと故障時の論理シミュレーションが実行される。その結果，故障 fが検出されなければ，以下の処理を繰り返す。パターン系列 ISPに対して，1ビットだけ値を反転させたパターン系列が存在すれば，それを ISAとして選択する。そして，ISA に対して，故障シミュレーションを行い故障が検出されない場合は，各パターン系列のコストを計算する。ここで，パターン ISのコスト $C(IS)$ とは，組合せ回路の場合は，正常時の論理シミュレーションと故障時の論理シミュレーションを行ったときの各外部出力端子における値の差を全て加えたもの DPOの逆数で定義される。

$$C(IS) = \frac{1}{DPO} \tag{1}$$

順序回路の場合は，時間軸展開した記憶素子 jにおける出力値の差 DFO_jに重み W_jをかけたものを全ての記憶素子について加え，さらに DPOを加えた値の逆数をコストとする。この重み W_jを制御することで故障信号を外部出力に近い記憶素子へと誘導している。

$$C(IS) = \frac{1}{DPO + \sum_j W_j DFO_j} \tag{2}$$

このようにして計算された ISA のコストが減少する場合は，それを対象パターン系列 ISPとして処理を繰り返し，テストパターン ISNを出力する。

2.2 順序回路テスト生成処理

順序回路テスト生成の処理手順を図 3に示す。未検出故障が選択されると収束計算処理により検出されたテストパターン ISNをテストパターン系列に加える。その後，故障シミュレーションを行い，同時に検出される故障を対象故障から除く。回路状態を記憶し，論理シミュレーションの結果から過去に同じ回路状態があったかを調べることで，故障検出における不要な処理を回避している。

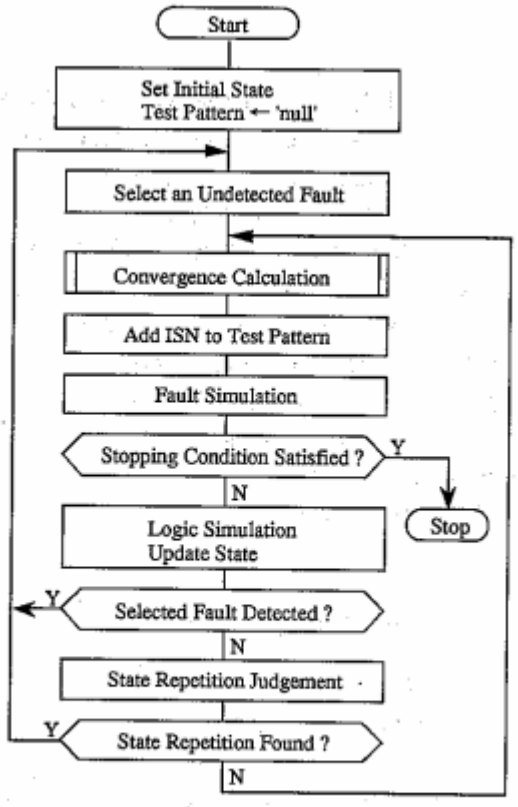

図 3: 順序回路テスト生成処理概要

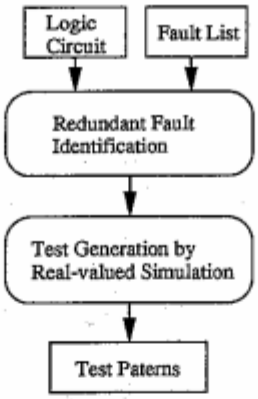

図 4: DESCARTES 概略処理フロー

3 並列テスト生成システム DESCARTES

並列テスト生成システム DESCARTES (Distributed processing Environment oriented System using Concurrent AcceleRative TEst generation methods for Sequential circuits) の目的は，並列処理を応用することにより，スキャン回路を持たない順序回路に対しても，高い故障検出率を実現するテストパターン系列を実用時間内に生成することである。そのためには，効率的な並列化方式が重要となる。ここでは，DESCARTES の処理概要，システム構築上の課題及び並列化方式について述べる。

3.1 処理概要

並列テスト生成システム DESCARTES の概略処理フローを図4に示す。最初に，論理 LSI の論理回路データと検出すべき故障のリストが与えられる。ここで故障は，ゲートの入出力信号線を対象とした単一縮退故障 (0 レベルまたは 1 レベルに固定する故障) であるとする。最初に，これらの故障の中から，冗長故障と呼ばれる回路出力値が正常時と故障時とで変わらない故障を検出する。容易に判定できる冗長故障をテスト生成前に指摘することは，テスト生成の無駄な処理を削減でき，システム全体の処理時間短縮につながる。その結果残った故障を対象として，実数値シミュレーションに基づくテスト生成法を用いてテストパターンを生成する。

3.2 システム構築上の課題

一般に，高速な並列プログラムを開発するには次の三つの課題を解決する必要がある。複数のプログラムが並行に動作すると，その実行順序で処理結果が異なることがある。よって，解の品質を落とさずに並列度の高い並列アルゴリズムを設計する必要がある (課題 1)。また，プロセッサ間通信コストが比較的大きな並列処理環境では，プロセッサ間の通信オーバヘッドがなるべく小さくなるようなプログラム間のメッセージ設計を行う必要がある (課題 2)。そして各プロセッサのアイドル状態がなるべく少なくなるように，各プロセッサの負荷均等化も考慮しなければならない (課題 3)。

また，大規模な回路データを扱えるシステムにするには，テスト生成処理をトポロジカル並列に基づき並列化する必要がある。そのためには，さらに以下の課題を解決しなければならない。回路分割を行うと他のプロセッサ上のプログラムに処理を依頼しなければならないので，プロセッサ間のメッセージ通信が必ず生じる。よって，プログラム間の通信オーバーヘッドが，なるべく少なくなるような回路分割方式が必要となる (課題 4)。そして非同期的なメッセージ交換を必要とする複雑な分散アルゴリズムに基づいて，並列プログラムを実現しなければならないので，効率良くプログラム開発が可能な，並列プログラム設計方法が必要となる (課題 5)。最後に並列プログラムを実際に動作させて詳細な解析を行い，必要となるメモリ量と高速化とのトレードオフを見極めることが重要である (課題 6)。

(課題 1) から (課題 6) に沿って，DESCARTES における課題をまとめる。

(1) (課題 1) から (課題 3) に関しては，テスト生成の処理フローから適切な並列性を抽出し，通信頻度がなるべく少なくなり，かつ負荷が均等化するように並列化する必要がある。

(2) (課題 4) に関しては，基本アルゴリズムと整合性のよい回路分割法を提案しなければならない。

(3) (課題 5) に関しては，効率よく並列プログラムを開発するためのツールの整備と効果的なプログラム設計手法とが必要である。

(4) (課題 6) に関しては，並列プログラムを実現し，その詳細な評価を行う必要がある。

(3) の課題における並列プログラム開発環境の現状について簡単に述べる。並列処理用システムソフトウエアが多数提案されており，種々の並列計算機への実装が進められている。FORTRAN や C 言語などの並列処理ライブラリを提供するものとして，PVM, Express, Linda などがある。このように，汎用並列計算機上で動作する各種の言語処理系が提案されており，それとともに並列プログラム開発ツールも充実しつつある。ま

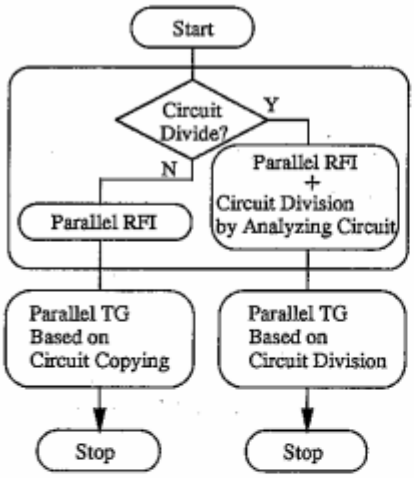

図 5: 回路解析を利用した回路分割

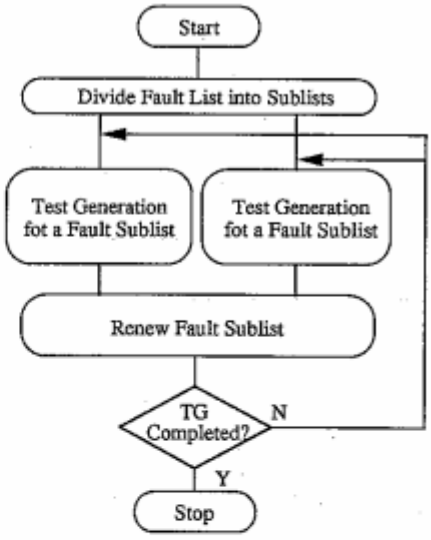

図 6: 故障並列によるテスト生成処理の並列化

た並列プログラム設計手法としては，並列オブジェクトモデルに基づくものがあり，LSI-CAD における論理シミュレーション [15] や配線プログラム [7, 8] が設計されている。これらの動向を踏まえ，(3) の課題に対処できる環境を整備して行く必要がある。

以下では，(1),(2) の課題に対処するための並列化方式を提案する。

3.3 全処理における並列化

DESCARTES では，テスト生成の前に回路解析を行い冗長な故障を検出する。この回路解析結果を回路分割を行うときの情報として用いトポロジカル並列を実現する。図5に示すように，回路分割が必要な場合は，並列簡易冗長故障検出処理における回路解析処理を応用し，部分回路に分割する。そして各ワークステーションのメモリ上には，部分回路情報のみをもたせ，プログラム間でメッセージを交換しながらテスト生成を行う。また，回路分割が必要ない場合は，全回路を各ワークステーションの主メモリにコピーして並列にテスト生成を行う。

3.4 冗長故障判定処理における並列化

冗長故障判定処理では，並列学習操作に基づく方式 [17] を既に提案している。ここでの並列化方式は，二つある。一つは，論理ゲートにおける信号線の値が，0 または 1 に固定するするかどうかを判定するのに複数の信号線を同時に評価している。また，複数の故障に対して同時に冗長判定を行う故障並列も採用している。

3.5 テスト生成処理における並列化

テスト生成処理では，故障並列，探索空間並列，機能並列に基づき並列アルゴリズムを設計している。以下では，各並列化方式について詳しく述べる。

テスト生成処理全体における並列化方式として，図6に示すような**故障並列** を採用する。故障リストを分割し，各プロセッサに分担させて同時にテスト生成を行う。このとき，各プロセッサでのテスト生成処理において故障シミュレーションにより他のプロセッサが担当している故障リストに含まれる故障も検出されることがある。そのため，同期をとって各故障リストを更新する処理を行い，無駄なテスト生成処理を削減する。

テスト生成処理における収束計算処理では，二つの方式に基づき並列化を行っている。収束計算処理全体の並列化として探索空間並列の一種と考えられるコスト計算の並行処理を行っている。これは，コスト改善の方式により並列度が異なる。逐次プログラムの収束計算におけるコスト改善処理では，二つのパターンのコストを比較し，コストの小さい方を採用する。並列プログラムでは，図 7に示すように異なる入力パターンに関する収束計算処理を複数のプロセッサで同時に行う。各プロセッサで正常時の論理シミュレーションと故障時の実数値シミュレーションが終了すると，対象とする故障が検出されるかどうかを調べ，どのプロセッサでも検出されない場合は，全てのプロセッサで同時に計算したコストをマスタプロセッサが集計する。その結果，コストの最も小さなパターンを採用し，次の対象パターンとして処理を繰り返す。複数パターンの選択方法としては，ランダムに生成した一つの入力

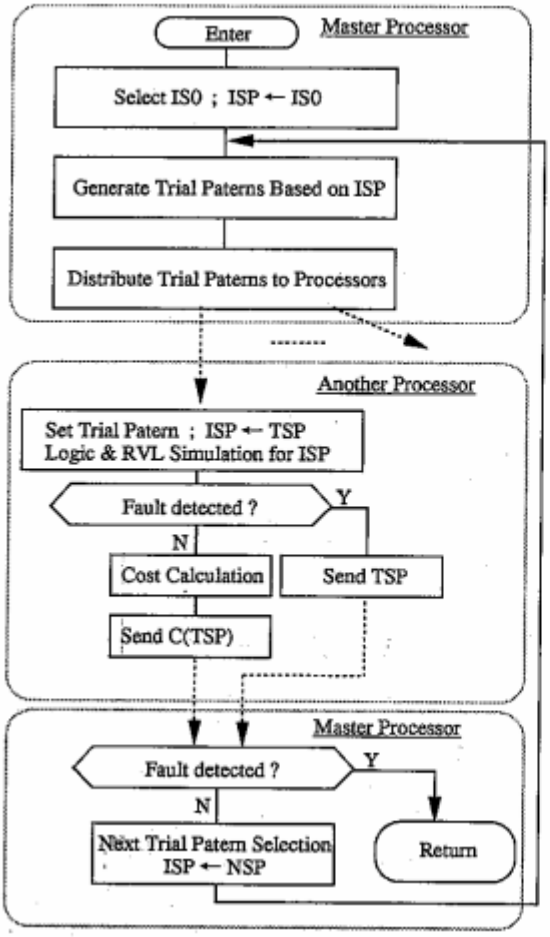

図 7: 探索空間並列による収束計算処理の並列化

パターンについて n ビットだけ変更したパターンを選ぶ。ここで, nは, 入力パターンビット数以下の自然数である。n を変更することにより, 一度に比較対象とする探索空間を変えることができる。

収束計算処理におけるコスト計算では, 最初に正常時の論理シミュレーションと故障時の論理シミュレーションを行う。これらの処理は, 独立に実行できるので, 機能並列により並列処理を行う。

4 評価実験

4.1 評価項目

DESCARTES を構成する個々の並列プログラムについて, 高速化, 大規模データへの適用という観点から以下の評価項目を設定する。

(1) 並列化方式の高速性に関する評価： プログラムの並列化により, 解の品質を落とさず, どの程度の高速化が達成でき, オーバヘッドの要因は何かを並列化方式毎に調べる。

(2) 回路分割方式の有効性に関する評価： 大規模データへの適用可能性を評価するため, 回路分割方式に基づくテスト生成プログラムを実現し, その有効性を評価する。

(3) システムの実用性に関する評価： 大規模順序回路データを実用時間内でテスト生成可能かどうかを実データを用いて評価するとともに, プログラム変更に対して迅速に対応可能かどうかを評価する。

これらの評価事項の中で, 本論文では, (1) に関して, DESCARTES における故障並列による並列化方式についての評価結果を示す。

4.2 実験環境

並列処理の環境としては, 汎用並列計算機と複数のワークステーションをネットワークで接続したものとが考えられる。現在の LSI 設計環境から容易に実現できるということからワークステーションを ether ネットで接続し, 同時に最大 16 台まで使用できる環境下で, 評価実験を行うこととした。ワークステーションは, 全て HP9000 715/33 を使用した。また, 汎用並列計算機上へのプログラムの移植性を考慮し, 種々の並列処理環境をサポートしている並列処理用システムソフトウエア Express を採用した。

4.3 実験結果

冗長故障判定プログラムとテスト生成プログラムを故障並列に基づき並列化した並列プログラムを ISCAS89 ベンチマークデータを用て評価した結果を示す。各プログラムに関して, プロセッサ数と台数効果の関係を調べた。台数効果とは, 逐次プログラムの実行時間を並列プログラムの実行時間で割った値で, 並列処理効果を示している。表 1 は, ISCAS89 ベンチマークデータの諸元 [3] である。

表 1: ISCAS89 ベンチマークデータ

circuit name	primary inputs	primary outputs	FFs	AND/OR/ NOT gates
s208	11	2	8	96
s838	35	2	32	390
s5378	35	49	179	2779
s9234	19	22	228	5597
s13207	31	121	669	7951

(1) 冗長故障判定プログラムの評価

冗長故障判定プログラムに関して, 検出した冗長故障数 (Nrf), プロセッサ数 (Np), 実際にプログラムを実

行するのにかかった時間 (Ttat), 台数効果 (speed up) を測定した結果を表 2に示す。並列冗長故障判定プログラムは, プロセッサ数に依存せず全てのデータにおいて逐次プログラムと同じ冗長故障検出率を示しており, 並列化による解の品質の低下はない。

表 2: 冗長故障判定数と処理時間

circuit name	Nrf	Np	Ttat (sec.)	speed up
s208	64	1	0.6	1
		4	0.4	1.5
		8	0.5	1.2
s838	182	1	3.9	1
		4	1.4	2.7
		8	1.3	3.0
s5378	774	1	46.3	1
		4	18.3	2.5
		8	8.7	5.3
s9234	3050	1	403.1	1
		8	63.4	6.3
		16	34.3	11.8
s13207	390	1	508.1	1
		8	81.0	6.3
		16	42.3	12.0

また台数効果 (speed up) に関しては, 回路規模の小さなデータに関しては, 少ないプロセッサで飽和するが, 回路規模が増加すると台数効果は向上することがわかり, s9234, s13207 では, 約 12 倍の台数効果を達成した。

(2) テスト生成プログラムの評価

テスト生成プログラムに関する実験結果について考察する。最初に全てのプロセッサが一つの故障に対するテスト生成が終了するのを同期をとって待ち, 故障リストを更新し, 新たなテスト生成を行った場合のプロセッサ数 (Np), 故障検出率 (test cov.), テスト系列長 (Npat), 実際にプログラムを実行するのにかかった時間 (Ttat), 台数効果 (speed up) を測定した結果を表 3に示す。この表から, 他のプロセッサの終了を待つと, 回路規模が大きくなっても, 台数効果は向上しないことがわかる。これは, 各プロセッサの負荷が均等でないことによる。

表 3: 同期式テスト生成の処理結果

circuit name	Np	test cov. (%)	Npat	Ttat (sec.)	speed up
s208	1	88.1	592	48	1
	4	90.1	832	30	1.6
	8	81.5	1088	24	2.0
s838	1	81.9	1328	1353	1
	4	82.9	1656	521	2.6
	8	81.9	1808	301	4.5
s5378	1	91.6	3630	100109	1
	8	91.6	4480	20224	5.0
	16	90.8	4960	14238	7.0

表 4は, 各プロセッサは, 他のプロセッサの終了を待たずに非同期的に故障リストを更新し, 残った故障リストを再分配した結果である。この場合, 回路規模が増加すると台数効果が向上する。

これらの結果から, 実数値シミュレーションに基づくテスト生成プログラムを故障並列により並列化する場合でも, Patil と Banerjee[19] が algorithmic 方式のテスト生成法に関して指摘しているのと同様に, 一つの故障に対するテスト生成処理の負荷にばらつきがあることがわかった。

また, 非同期的に各プロセッサを動作させた並列プログラムでは, 台数効果も向上し, 故障検出率も乱数系列の違いによるばらつきが若干あるものの逐次プログラムとほぼ同等なものが得られることがわかる。s5378 では, 約 12 倍の台数効果を達成している。

表 4: 非同期式テスト生成の処理結果

circuit name	Np	test cov. (%)	Npat	Ttat (sec.)	speed up
s208	1	88.1	592	48	1
	4	86.1	586	24	2.0
	8	86.8	744	19	2.5
s838	1	81.9	1328	1353	1
	4	80.1	1366	479	2.8
	8	80.1	1452	211	6.4
s5378	1	91.6	3630	100109	1
	8	90.2	4610	17508	5.7
	16	90.1	4650	8343	12.0

5 むすび

並列処理に基づいた実用的なテスト生成システムを構築する上での課題を明確にし, 実数値シミュレーションに基づく並列テスト生成システム DESCARTES を提案した。その課題と並列化方式について述べた。

そして並列化方式の高速性に関する評価として, ワークステーションネットワークによる並列処理環境で故障並列による並列プログラムを評価した。

冗長故障判定プログラムに関する実験では, 逐次プログラムと同等の冗長故障検出率を示し, 16 台のワークステーションを用いて最大 12 倍の高速化を達成した。

テスト生成プログラムに関する実験では, 実数値シミュレーションを用いたテスト生成も algorithmic 方式のテスト生成法と同様に一つの故障に対するテスト生成処理時間が不均一であることがわかった。非同期的に故障リストを更新し, 各プロセッサに故障を分担させテスト生成させるプログラムでは, 逐次プログラムと同等の故障検出率が得られ, 16 台のワークステーションを用いて最大 12 倍の高速化を達成した。

基本アルゴリズムに関する今後の課題として，故障検出率の向上，テスト系列長の削減などがあげられる。

また，並列テスト生成システムとしての課題としては，以下のものがある。

(1) DESCARTES における他の並列化方式の高速性に関する評価。

(2) 回路分割方式の有効性に関する評価。

(3) システムの実用性に関する評価。

参考文献

[1] P. Agrawal, V. D. Agrawal, J. Viloldo, "Sequential Circuit Test Generation on a Distributed System," *Proc. 30th DA Conf.*, pp. 107-111, 1993.

[2] V. D. Agrawal, K.- T. Cheng and P. Agrawal, "CONTEST : A Concurrent Test Generator for Sequential Circuits," *Proc. 25th DA Conf.*, pp. 84-89, 1988.

[3] F. Brglez, D. Bryan and K. Kozuminski, "Combinational Profiles of Sequential Benchmark Circuits," *Proc. ISCAS'89*, pp. 1929-1934, 1989.

[4] S. J. Chandra and J. H. Patel, "Experimental Evaluation of Testability Measures for Test Generation," *IEEE Trans. Computer-Aided Design*, Vol. 8, No. 1, pp. 93-97, Jan. 1989.

[5] K.- T. Cheng and V. D. Agrawal, "A Sequential Circuit Test Generator Using Threshold-Value Simulation," *Dig. FTCS-18*, pp. 24-29, 1988.

[6] W.- T. Cheng and T.J. Chakraborty, "Gentest – An Automatic Test-Generation System for Sequential Circuits," *Computer*, Vol. 22, pp. 43-49, April 1989.

[7] 伊達博, 大嶽能久, 瀧和男, "並列オブジェクトモデルに基づく LSI 配線プログラム", 情報処理学会論文誌, Vol. 33, No. 3, pp. 378-386, Mar. 1992.

[8] H. Date, K. Taki, "A Parallel Lookahead Line Search Router with Automatic Ripup-and-reoute", *EDAC-EUROASIC93*, pp. 117-121, Feb. 1993.

[9] H. Fujiwara and T. Inoue, "Optimal Granularity of Test Generation in a Distributed System," *IEEE Trans. Computer-Aided Design*, Vol. 9, No. 8, pp. 885-892, Aug. 1990.

[10] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Trans. Comput.*, Vol. C-30, No. 3, pp. 215-222, 1981.

[11] K. Hatayama, K. Hikone, M. Ikeda and T. Hayashi, "Sequential Test Generation Based on Real-Valued Logic Simulation," *Proc. ITC'92*, pp. 41-48, 1992.

[12] K. Hikone, M. Ikeda, K. Hatayama and T. Hayashi, "Sequential Test Generation by Real Number Simulation," *Systems and Computers in Japan*, Vol. 24, No. 9, pp. 64-75, 1993.

[13] F. Hirose, K. Takayama, and N. Kamato, "A Method to Generate Tests for Combinational Logic Circuits Using an Ultra High Speed Logic Simulator," *Proc. ITC'88*, pp. 102-107, 1988.

[14] G.A. Kramer, "Employing Massive Parallelism in Digital ATPG Algorithms," *Proc. ITC'83*, pp. 108-114, 1983.

[15] 松本幸則, 瀧和男, "バーチャルタイムによる並列論理シミュレーション", 情報処理学会論文誌, Vol. 33, No. 3, pp. 387-395, Mar. 1992.

[16] A. Motohara, et al., "A Parallel Scheme for Test Pattern Generation," *Proc. IEEE ICCAD*, pp. 156-159, 1986.

[17] 中尾教伸, 伊達博, 宮崎政英, 畠山一実, "並列学習操作による冗長故障判定手法," 信学技報 FTS93-43, pp. 33-40, 1993.

[18] S. Parkes, P. B. Banerjee and J. Patel, "PROPER-HITEC : A Portable, Parallel, Object-Oriented Approach to Sequential Test Generation," *Proc. 31st DA Conf.*, pp. 717-721, June 1994.

[19] S. Patil and P. Banerjee, "Fault Partitioning Issues in an Integrated Parallel Test Generation/Fault Simulation Environment," *Proc. ITC'89*, pp. 718-726, 1989.

[20] S. Patil and P. Banerjee, "A Prallel Branch-and-Bound Algorithm for Test Generation," *Proc. 26th DA Conf.*, pp. 339-343, June 1989.

[21] J. P. Roth, W. G. Bouricious and P. R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish between Failures in Logic Circuits," *IEEE Trans. Electron. Comput.*, Vol. EC-16, No. 5, pp. 567-579, 1967.

[22] S.P. Smith, B. Underwood, and M.R. Mercer, "An Analysis of Several Approaches to Circuit Partitioning for Parallel Logic Simulation," *Proc. ICCD : VLSI in Computers and Processors.*, 1987, pp. 664-667.