設計自動化　68－17
（1993. 10. 29）

並列学習操作による冗長故障判定手法

中尾　教伸　　伊達　博　　宮崎　政英　　畠山　一実

（株）日立製作所　日立研究所

茨城県日立市大みか町七丁目１番１号

あらまし　本稿では，順序回路を対象とした並列冗長故障判定手法を提案する．その特徴は，論理値が固定される信号線を検出する処理において，複数の信号線を同時に学習操作する点にある．また，さらに並列性を高めるために，個々の故障の冗長性を判定する処理において，複数の故障を同時に判定する故障並列法を採用した．本手法に基づく並列プログラムを複数のワークステーションからなるネットワーク上で実現し，ISCAS'89 ベンチマーク回路を用いて評価した結果を示す．

和文キーワード　　テスト生成，冗長故障，学習操作，並列処理，故障並列

A Redundant Fault Identification Method Based on Parallel Learning Procedure

Michinobu NAKAO　Hiroshi DATE　Masahide MIYAZAKI　Kazumi HATAYAMA

Hitachi Research Laboratory , Hitachi , Ltd.

7-1-1 Omika-cho Hitachi-shi Ibaraki-ken

Abstract　In this paper, we present a redundant fault identification method for sequential circuits on a distributed system. The key is to learn concurrently for different signals to find fixed signal logic values. We also adopted fault parallelism that deals concurrently with different faults in judgement of redundancy for each fault. We implemented the parallel program based on proposed method and evalueated experimental results on a network of workstations using the ISCAS'89 benchmark circuits.

英文 key words　　Test generation, Redundant faults, Learning procedure, Parallel processing, Fault parallel

1．はじめに

論理LSIの大規模化，高機能化と共に，回路の故障検査に必要なテストパターンの作成は困難さを増している．これは論理LSIの設計期間を短縮するうえで大きな障害となることから，テスト生成処理の高速化は重要な課題である．

一般に論理LSIは組合せ回路部分と記憶素子からなる順序回路である．順序回路のテスト生成を容易にするために，現在，論理LSIの多くはスキャン回路設計を取り入れ，順序回路のテストパターン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，マイコン，ゲートアレイ等においてはスキャン回路のオーバーヘッド削減の要望がある．しかし，スキャン回路を持たない順序回路に対して，従来のテストパターン生成手法では非常に多くの時間を要し，大規模回路になると，実用的な時間での処理が不可能になる．

テスト生成高速化のために，自動テスト生成システムSOCRATES[7]やESSENTIAL[8]では，テスト生成の前処理として学習と呼ばれる操作を行い，テスト生成を効率化している．この学習操作は，テスト生成におけるバックトラックを減らすとともに，冗長故障と呼ばれる回路出力値が正常時と故障時で変わらない故障の指摘もすることができる．容易に判定できる冗長故障をテスト生成手法適用前に指摘することは，テスト生成システム全体の処理時間短縮につながる．特に，冗長判定のできない，シミュレーションを基本にしたテスト生成の場合は有効である．

さらに高速なテスト生成を実現する1つの方法として，複数のプロセッサを用いた分散並列処理がある．テスト生成の並列処理について，探索並列法[1]，回路並列法[2]，方策並列法[3]，故障並列法[4,5,6]などが提案されている．特に，故障並列法は故障集合を分割して各プロセッサに割り当て，複数の故障を同時に処理するもので，汎用コンピュータの疎結合型ネットワーク上で，テスト生成を効率良く高速化できることが報告されている．しかし，これらの並列化手法はテスト生成手法に対するものであり，前処理に対する並列化は提案されていない．テスト生成システム全体の処理時間をさらに短縮するためには，テスト生成本体だけでなく，前処理についても並列化する必要がある．

本稿では，順序回路を対象とした冗長故障判定を効率良く並列処理する手法を提案する．そして，本手法に基づき，複数のワークステーションから構成されるネットワーク上で実現した並列プログラムの性能を，ISCAS'89ベンチマーク回路を用いて評価する．

2．学習操作を用いた冗長故障判定

2.1 学習操作

SOCRATES[7]やESSENTIAL[8]に組み込まれている**学習操作**は，論理回路の1つの信号線に論理値を設定し，含意操作を行った結果得られた命題及びその対偶命題を記憶する操作である．ここで，含意操作とは，信号線の論理値設定に対し，一意的に論理値が決まる信号線を順次決定する操作である．この含意操作の段階で矛盾が生じる場合，即ち，同一信号線に異なる論理値設定の要求が生じる場合がある．図1にその例を示す．例えば，信号線s1に論理値1を設定し含意操作を行う．AND2の出力s1を1に設定するにはその入力s3,s4を1に設定する必要がある．しかし，s4を1に設定することはs6が0であることを意味するため，s6を入力の1つとするAND3の出力s3は0となり，矛盾を生じる．従って，s1を論理値1に設定できないことから，論理値0に固定される信号線と判定できる．この信号線s1のように正常時の論理値が固定される信号線を**固定値信号線**と呼ぶ．同様にs0も論理値0の固定値信号線となる．

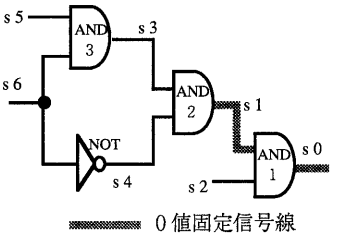


図1．固定値信号線及び冗長故障の例

この時，s0の0縮退故障とs1の0縮退故障は，正常時に1に設定できないため，冗長故障となる．また，s2の0縮退故障及び1縮退故障は，その影響をs0に伝播しなければ検出できないが，固定値信号線s1によってs0が故障時でも0になるため，s0に故障信号を伝播できず，冗長故障となる．

2.2 冗長故障判定の概要

並列化を行った冗長故障判定手法の逐次アルゴリズムについて述べる．なお，順序回路を対象としているので，多時刻の回路を扱わなければならない．そのため，順序回路のフィードバックループの情報を時間軸展開し，全体としてループのない組み合わせ回路として考える手法を用いている．図2に逐次版冗長故障判定の処理全体の流れを示す．その各処理について，以下の(1)，(2)，(3)で詳細を説明する．

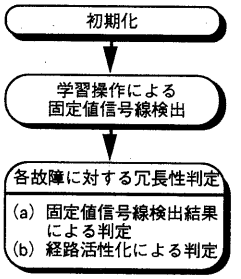


図2．冗長故障判定手法

(1) 初期化

回路データ・故障リストを入力し，計算処理に用いるテーブルの設定や変数・定数の初期化を行う．さらにゲートの深さに関するソーティングを行って信号線の処理順序を決定する．

(2) 学習操作による固定値信号線判定処理

処理手順を図3に示す．各信号線を選択し，論理値1及び0の初期設定で含意操作を行う．含意操作の過程で矛盾が発生した場合，初期設定が不可能であるので，その信号線を初期設定の否定値に固定される固定値信号線と判定する．固定した論理値に対して含意操作を行い，関連する固定値信号線部分を明らかにする．これを**固定値含意操作**と呼ぶ．ここでの学習操作は固定値信号線を検出することが目的であるため，含意操作で矛盾が生じない場合はその結果から導かれる命題やその対偶命題を記憶することはせず，次の信号線を判定する．

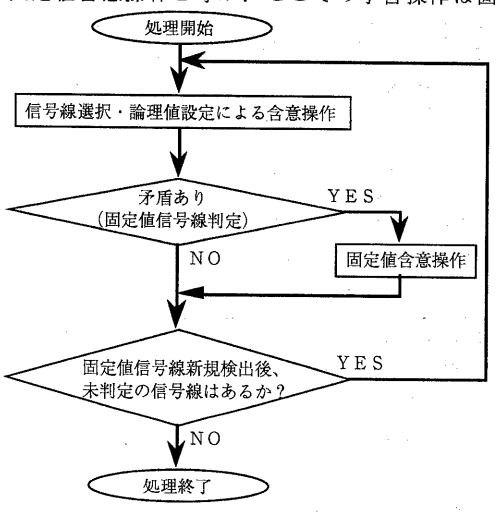


図3．固定値信号線検出の処理手順

なお，新規に固定値信号線が発見されると，回路についての知識を更新することになり，発見以前の学習操作を再試行する価値が生じる．図4に，学習操作を再試行する必要のある回路例を示す．例えば，信号線s3，s4より先に信号線s0に対する学習操作を行うとする．信号線s0を論理値1に設定して含意操作を行った場合，ORゲート，NANDゲートの入力信号線の論理値を決定することができないため，矛盾を生じない．しかし，信号線s3，s4を学習操作によって固定値信号線と判定した後，再度信号線s0に論理値1を設定し含意操作すると，信号線s7で矛盾を生じ，s0を0値固定信号線と判定できる．この信号線s0のように，他の固定値信号線が検出されないと固定値信号線と判定できない信号線があることから，発見可能な固定値信号線を数え尽くすために，新規検出が無くなるまでこのプロセスを繰り返す．

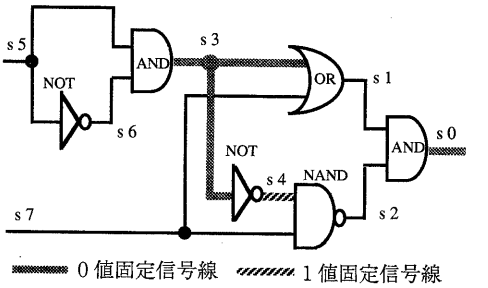


図4．学習操作を再試行する必要のある回路例

(3) 各故障に対する冗長性判定

次の2つのフェイズに分けることができる．

(a) 固定値信号線検出結果による判定

固定値信号線の検出結果から容易に冗長と判定できる故障を，故障リストから冗長故障リストに移行する．

(b) 経路活性化による判定

さらに冗長故障の指摘数を増やすため，(a)で冗長故障と判定されなかった故障に対して，冗長かどうか判定を行う．例えば，図5の回

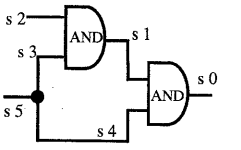


図5．冗長故障例

路には固定値信号線がないため，(a) では冗長故障を指摘できないが，信号線 s3の１縮退故障と s4 の１縮退故障は冗長故障である．このような冗長故障を判定するため，各故障を仮定し，経路活性化，即ち，故障信号を外部出力へ伝搬させる操作を試みる．この過程で，外部出力又は記憶素子への伝搬経路が明らかに存在しない故障について，冗長と判定する．

3．冗長故障判定手法の並列化

学習操作による固定値信号線検出と，経路活性化による冗長性判定のそれぞれの処理で，異なる並列化方式を用いる．前者では，複数の信号線を同時に処理する**並列学習操作**を提案し，後者では複数の故障を同時に判定する**故障並列法**[4]を採用する．以下では，その詳細について述べる．なお，各プロセッサには１つのプログラムが対応し，その中で並列処理全体を管理するプログラムを master プログラム，それ以外のプログラムを slave プログラムと呼ぶことにする．

3.1 初期化

master プログラムは回路データと故障リストを読み込む．次にゲートの深さに関するソーティングを行った後，これらの回路情報をslave プログラムにブロードキャストする．したがって，master 及び全 slave プログラムは同じ回路情報を持っている．そして，各プログラムは逐次処理の場合と同様の初期化を行う．

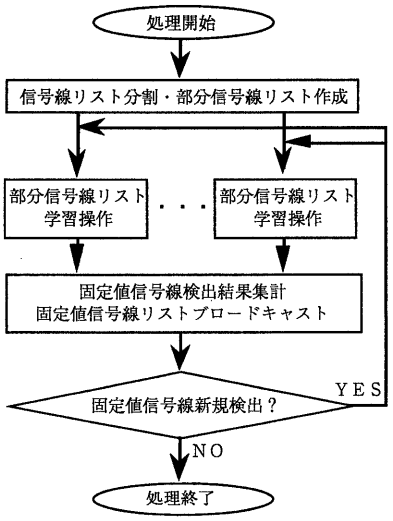


図６．並列学習操作による固定値信号線検出の処理手順

3.2 並列学習操作による固定値信号線検出

図６に処理手順を示す．まず，信号線リストを各プログラム用に分割する．本稿では深さの小さい信号線から順に，サイクリックに各プログラムに割り当てる．以下では，分割した信号線リストの１つ１つを**部分信号線リスト**と呼ぶ．

各プログラムは部分信号線リストの信号線について，学習操作を行って固定値信号線かどうかの判定を行う．新規に固定値信号線を発見すると，固定値含意操作を行って，固定値信号線検出結果を管理する**固定値信号線リスト**に書き込む．固定値含意操作では回路全体の信号線に対して含意操作を行うので，部分信号線リスト以外の信号線を固定値信号線と判定する場合もある．以下では，各プログラムの部分信号線リストに対する処理を，**部分信号線リスト学習操作**と呼ぶ．但し，１回の部分信号線リスト学習操作では，各信号線に対する学習操作・固定値信号線判定は１回のみとする．

学習操作の結果，固定値信号線を新規検出した場合，2.2(2)で述べたように検出以前の学習操作を再試行しなければならない．そのためには，各プログラムは他のプログラムの検出結果を知る必要がある．そこで，部分信号線リスト学習操作終了後，masterプログラムはslaveプログラムから受け取った新規検出信号線を集計し，更新された固定値信号線リストを全てのslaveプログラムにブロードキャストする．すべてのプロ

(a) 部分信号線リスト学習操作１回目

信号線名	1	2	3	4	5	6	7	8	9	10	11	12	....
mater	1*	0*	X	X	X	X	X	X	X	0*	1*	1*	....
slave1	1*	0*	X	X	X	1*	X	0*	X	0*	1*	1*	....
slave2	X	X	X	X	X	X	X	X	X	0*	X	1*	....

(b) 集計・ブロードキャスト後

信号線名	1	2	3	4	5	6	7	8	9	10	11	12	....
mater	1	0	X	X	X	1	X	0	X	0	1	1	....
slave1	1	0	X	X	X	1	X	0	X	0	1	1	....
slave2	1	0	X	X	X	1	X	0	X	0	1	1	....

(c) 部分信号線リスト学習操作２回目

信号線名	1	2	3	4	5	6	7	8	9	10	11	12	....
mater	1	0	X	0*	1*	1	X	0	X	0	1	1	....
slave1	1	0	X	X	X	1	X	0	X	0	1	1	....
slave2	1	0	X	X	X	1	X	0	X	0	1	1	....

(d) 集計・ブロードキャスト　各プログラムが判定する信号線

* 新規検出固定値信号線

図７．固定値信号線リストの例

グラムが固定値信号線を新規に検出しなくなるまで，部分信号線リスト学習操作を繰り返す．

図7はプログラム数が3個の場合について，上記処理を行った際の固定値信号線リストの例である．例えば，masterプログラムの動作を考える．部分信号線リストは信号線1，4，7，10，…である．(a)の部分信号線リスト学習操作1回目では信号線1，10について固定値信号線と判定している．固定値含意操作により関連する固定値信号線として，信号線2，11，12を指摘する．masterプログラムは(b)の集計・ブロードキャストによって，slave 1プログラムが検出した固定値信号線6，8を知り，固定値信号線リストに書き込む．固定値信号線の新規検出があるため，(c)の2回目の部分信号線リスト学習操作を行う．masterプログラムの場合，固定値信号線判定を行うのは信号線4，7である．このうち，新たに信号線4が固定値信号線であることが判明し，関連して信号線5も固定値信号線と指摘する．さらに，集計・ブロードキャスト後，固定値信号線の新規検出がなくなるまで，部分信号線リスト学習操作を繰り返す．

3.3 故障並列法による冗長性判定

2.2(3)(b)の経路活性化による判定では，各故障に対する冗長性判定が他の故障の判定結果に依存しないので独立に処理できる．そこで，経路活性化による判定を故障並列法を用いて処理する．まず，2.2(3)(a)の固定値信号線検出結果による判定を行った後，まだ冗長と判定されていない故障の集合を等分割し，各プログラムに割り当てる．各プログラムは割り当てられた故障について経路活性化による判定を行い，全故障を判定した後，結果をmasterプログラムに報告する．

4．試行結果及びその検討

以上示した手法を用いて実験プログラムを作成し，評価した．我々が並列処理を行った環境は，

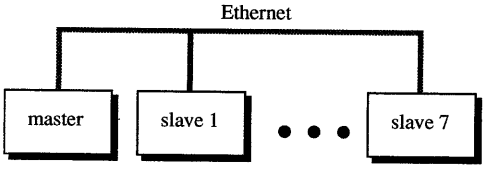


図8．8台のワークステーションから構成されるネットワーク

図8のように，最大で8台のワークステーションとそれをつなぐイーサネットから構成されるネットワークである．今回はネットワーク管理ソフトとして，メッセージパッシングを基本とした通信用ライブラリを提供するP4 (Portable Programs for Parallel Processors)[9]を採用した．ワークステーションはHP9000/715 (41MIPS, 32Mbyte)を使用し，試行データとしてISCAS'89国際標準ベンチマーク回路を用いた．

4.1 処理時間の計測

初期化，固定値信号線検出，冗長性判定の処理時間 T_{init}，T_{fix}，T_{rf} を次のように定義する．

$$T_{init} = t_{init}(0) \quad \cdots (1)$$

$$T_{fix} = \Sigma_i \max_p t_{fix}^i(p) \quad \cdots (2)$$

$$T_{rf} = \max_p t_{rf}(p) \quad \cdots (3)$$

但し，$t_{init}(p)$，$t_{rf}(p)$ はそれぞれ，初期化，冗長性判定におけるプログラムpのCPU処理時間を表す(p = 0はmasterプログラム)．$t_{fix}^i(p)$ は，固定値信号線検出処理で，プログラムpが第i回目の部分信号線リスト学習操作及び結果集計・固定値信号線リスト更新に要したCPU時間を表し，Σ_i は部分信号線リスト学習操作の繰り返し回数についての和をとるものとする．T_{fix} を式(2)で定義するのは，1回の部分信号線リスト学習操作毎に各プログラムが同期をとるので，全プログラムがその操作を終了するまでの時間を計測する必要があるためである．以上から，冗長故障判定全体の処理時間の下限は，

$$T_{min} = T_{init} + T_{fix} + T_{rf} \quad \cdots (4)$$

で表せる．このときの台数効果 SP_{min} を，逐次アルゴリズムによるCPU時間 T_{seq} を用いて，

$$SP_{min} = T_{seq} / T_{min} \quad \cdots (5)$$

で定義する．固定値信号線検出，冗長性判定の処理における台数効果 SP_{fix}，SP_{rf} も，同様に定義する．また，実際にかかるシステム全体の処理時間（Turn Around Time）を T_{tat} とする．さらに各プログラムのCPU時間の合計 T_{cpu} を

$$T_{cpu} = \Sigma_p(t_{init}(p) + \Sigma_i \max_p t_{fix}^i(p) + t_{rf}(p))$$

で定義する．

表1に本手法を用いた実験で指摘した冗長故障数と，T_{min}，T_{tat}，T_{CPU} を示す．T_{tat} は処理時間の下限 T_{min} より常に大きい．その差 $T_{tat} - T_{min}$ は，全てのプログラムがCPUを使っていない時間の合計を表し，プログラム間通信やワークステーション上で動いている他のプロ

表1　冗長故障指摘数と処理時間

Circuit	Gates	Faults	N fix	N rf	Nprg	T min	T tat	T cpu
s208	120	215	36	56	1	1.4		1.4
					4	0.6	0.7	1.8
					8	0.4	0.6	2.2
s298	161	308	0	0	1	0.6		0.6
					4	0.4	0.5	0.9
					8	0.3	0.5	1.3
S344	205	342	0	0	1	0.6		0.6
					4	0.4	0.5	1.0
					8	0.4	0.5	1.5
s349	206	350	1	1	1	0.8		0.8
					4	0.5	0.6	1.2
					8	0.4	0.6	1.7
s382	221	399	0	0	1	0.8		0.8
					4	0.4	0.6	1.2
					8	0.4	0.5	1.7
s386	177	384	0	0	1	0.9		0.9
					4	0.4	0.6	1.2
					8	0.4	0.5	1.8
s400	227	424	1	1	1	1.1		1.1
					4	0.6	0.8	1.7
					8	0.5	0.7	2.2
s420	244	430	132	211	1	3.3		3.3
					4	1.2	1.3	4.0
					8	0.9	1.0	4.7
s444	244	474	1	4	1	1.7		1.7
					4	0.8	0.9	2.4
					8	0.6	0.8	3.0
s526	256	555	0	0	1	1.2		1.2
					4	0.6	0.7	1.6
					8	0.5	0.7	2.1
s641	436	467	0	0	1	2.1		2.1
					4	0.9	1.0	2.6
					8	0.7	0.9	3.7
s713	450	581	16	17	1	3.0		3.0
					4	1.2	1.5	3.5
					8	0.9	1.2	4.5
s820	304	850	0	0	1	2.8		2.8
					4	1.1	1.4	3.5
					8	0.8	1.0	4.5
s832	302	870	0	0	1	2.9		2.9
					4	1.1	1.2	3.5
					8	0.9	1.1	4.6
s838	486	857	336	527	1	10.5		10.5
					4	3.3	3.6	11.4
					8	2.7	3.6	14.2
s1196	583	1242	0	0	1	3.9		3.9
					4	1.6	1.7	4.6
					8	1.3	1.4	6.5
s1238	562	1355	0	6	1	4.6		4.6
					4	1.9	1.9	5.6
					8	1.5	1.6	7.1
s1423	879	1515	0	4	1	3.5		3.5
					4	1.7	1.8	4.6
					8	1.6	1.7	8.2
s1488	671	1486	0	0	1	7.9		7.9
					4	2.7	3.8	9.0
					8	2.1	2.9	11.3
s1494	665	1506	0	1	1	8.0		8.0
					4	2.8	2.9	9.3
					8	2.0	2.2	11.2
s5378	3316	4603	423	576	1	68.8		68.8
					4	23.4	24.9	84.0
					8	14.7	15.0	93.8

・Tmin, Ttat, Tcpu の単位は秒
・Nfix, Nrf はそれぞれ指摘した固定値信号線数と冗長故障数

セスによるオーバーヘッドと考えられる．T_{tat} はネットワークやワークステーションの状態に強く依存するが，T_{min} とほぼ同様にふるまうことから，処理時間の評価に T_{min} を用いることが適切であると考える．また，Nprg =1 の T_{cpu} と Nprg = 4, 8 の T_{cpu} の差は，システム全体の並列処理によるオーバーヘッドを示す．Nprg = 8 の場合と Nprg =1 の場合の T_{cpu} の比は，回路規模が小さいときは2倍前後だが，s5378 の場合は 1.4 倍である．回路規模が大きくなるにつれて，並列処理によるオーバーヘッドの総処理時間に占める割合が減ると考えられる．

4.2　処理時間と台数効果についての考察

(1) プログラム数と台数効果

図9に，s5378 の処理時間の内訳と台数効果を示す．他の回路もほぼ同様の傾向である．

T_{init} については，プログラム数の増加に伴い，漸増している．この T_{init} の増加分は，回路データ及び故障リストのブロードキャストに要した時間である．2分木状にデータを伝達しているので，プログラム数Nprgに対し，T_{init} の増加分は log Nprgに比例する筈で，実験結果もそれを裏付けている．さらにプログラム数を増加させた場合，T_{init} の影響により式(4)で定義された T_{min} も増加に転じると予想される．即ち，総処理時間を最小にするプログラム数が存在する．最適なプログラム数に関しては，テスト生成の場合について文献[5,6]で論じてられている．

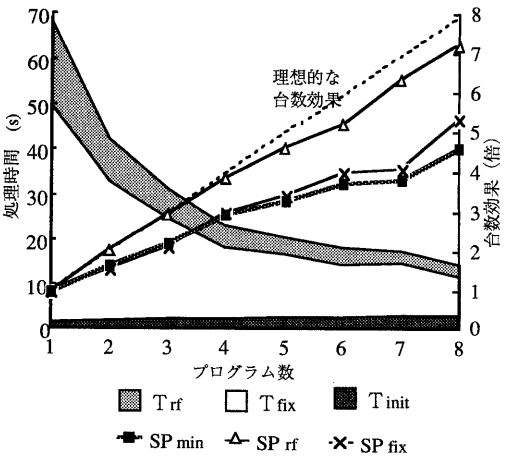


図9．s5378 の処理時間の内訳と台数効果

固定値信号線検出処理と冗長性判定処理の台数効果は，Nprg=8 の場合でそれぞれ，SP_{fix} = 5.4, SP_{rf} = 7.5 である．冗長性判定処理の台数効果

は理想的な値である8にかなり近い．これはプログラム間通信が処理の最後に結果を集計するときのみであることと，負荷分散がうまくいっていることが考えられる．後者については，単純な負荷分散であるが，1つの故障に対する処理時間が均一なためと思われる．

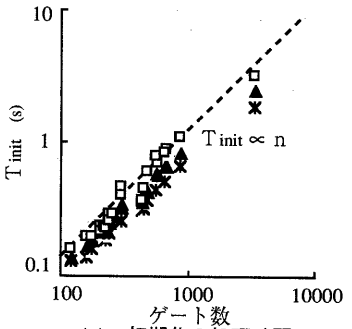
逆に，固定値信号線検出処理の台数効果があまりよくない理由として，次の3つが考えられる．第1に各プログラムが部分信号線リスト学習操作を終了するまで，他のプログラムの新規検出情報を知ることができないこと，第2に1回の部分信号線リスト学習操作で1つでも新規検出があれば，全プログラムが再度部分信号線リスト学習操作を行うため，判定する信号線の延べ回数が逐次アルゴリズムの場合に比べて増える可能性があること，第3に新規検出信号線の集計・ブロードキャストに要する時間がプログラム数に応じて増えることである．第2の理由に関して，部分信号線リスト学習操作の繰り返し回数は，今回実験を行った回路では3回以下で，逐次アルゴリズムの場合と同様だったことから，台数効果への影響はあまりないと思われる．

(2) 処理時間の計算量

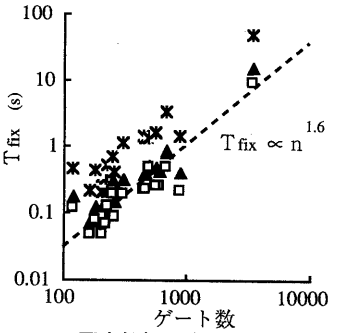
回路ゲート数と各部分の処理時間との関係を図10に示す．初期化での主な処理はソーティングとデータのブロードキャストである．逐次アルゴリズムの場合，図10(a)よりソーティングを中心とした初期化がゲート数nに対しO(n)で処理されることがわかる．ブロードキャストに要する時間は，理論的には回路データ及び故障リストのサイズに比例する筈である．図10(a)より，Nprg = 4，8の場合も，初期化の処理時間がO(n)であることが実験で確認できた．

固定値信号線検出，冗長性判定の計算量は，図10(b)(c)より，並列処理の場合も逐次処理の場合もほぼ同じオーダーで，ゲート数nに対し，$O(n^{1.5})$から$O(n^{1.6})$程度であることがわかる．T_{fix}とT_{rf}が単純にゲート数に依存せず，いくらかばらつきがあるのは，固定値信号線数など回路の性質が影響するためと思われる．

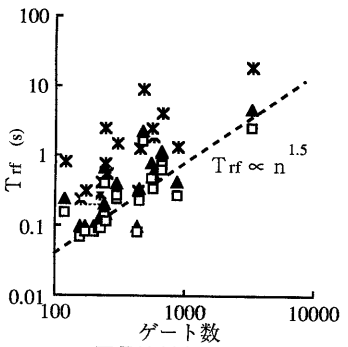
総処理時間の計算量に関しても，図10(d)が示すように，並列・逐次処理共に$O(n^{1.6})$程度である．また本手法による並列化を行えば，逐次処理の場合の計算量を変えずに，プログラム数に応じて処理時間を短縮できることが分かる．



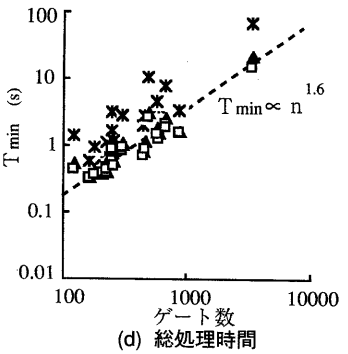
(a) 初期化の処理時間



(b) 固定値信号線検出の処理時間



(c) 冗長性判定の処理時間



(d) 総処理時間

✱ Nprg=1 ▲ Nprg=4 □ Nprg=8

図10．回路ゲート数と処理時間との関係

(3) 逐次処理時間と台数効果

処理時間はゲート数だけでなく回路の性質も影響するので，台数効果もゲート数だけには依存しない場合もあると考えられる．そこで，逐次処理の場合の処理時間と台数効果の関係を図１１に示した．

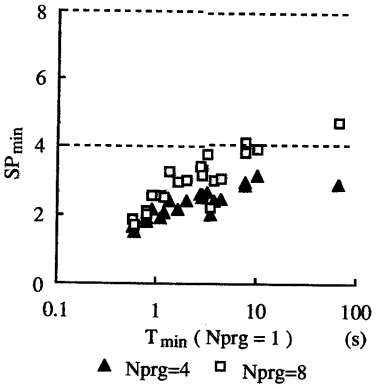


図１１．逐次処理時間と台数効果との関係

これにより，台数効果は逐次処理時間に強く依存しており，処理に時間のかかる回路ほど並列処理による高速化が有効であることがわかる．その理由として，本プログラムで並列化している部分の計算量がゲート数に対し$O(n^{1.6})$程度，そうでない部分が$O(n)$であることなどが挙げられる．

5．おわりに

本稿では，順序回路に対する冗長故障判定を並列に処理する手法として，並列学習操作に基づいた固定値信号線検出処理を提案した．さらに並列性を高めるため，冗長性判定処理では故障並列法を採用した．その有効性を確かめるため，複数のワークステーションからなるネットワーク上で，ISCAS'89 ベンチマーク回路を用いて評価した．その結果，並列化した部分は，処理時間の計算量がゲート数 n に対し 逐次処理と同じ$O(n^{1.6})$程度であり，プログラム数に応じて高速化できることがわかった．また，本手法を用いると回路規模の増加に従って台数効果が向上することを確認した．

参考文献

[1] S.Patil and P.Banerjee, "A parallel branch-and-bound algorithm for test generation," IEEE Trans. Computer-Aided Design, Vol.9, No3, pp.313-322, March 1990.

[2] G.A.Kramer, "Employing massive parallelism in digital ATPG algorithm," Proc. 1983 Int'l Test Conf., pp.108-114, 1983.

[3] S.J.Chandra and U.H.Patel, "Test generation in a parallel processing environment," Proc. IEEE Int'l Conf. Computer Design, pp.11-14, 1988.

[4] A.Motohara, K.Nishimura, H.Fujiwara and I.Shirakawa, "A parallel scheme for test pattern generation, "Proc. IEEE Int'l Conf. Computer-Aided Design, pp.156-159, 1986.

[5] 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.

[6] 井上智生，米澤友紀，藤原秀雄，"テスト生成における並列処理の最適なシステム構成について，" 信学技報 FTS92-17, pp.17-24, 1992.

[7] M.H.Schulz and E.Auth, "Advanced automatic test pattern generation and redundancy identification techniques," Dig. Papaers, FTS-18, pp.30-35, June 1988

[8] E.Auth & M.H.Schulz :A Test - Pattern - Generation Algorithm for Sequential Circuits: IEEE DESIGN & TEST OF COMPUTERS , pp.72-86, June 1991.

[9] Ralph Butler & Ewing Lusk : User's Guide to the p4 Parallel Programming System : Argonne National Laboratory, Oct. 1992.