計算機アーキテクチャ　83－5
（１９９０．７．１８）

並列システムによるテスト生成

蓑原 隆　　上野洋一郎　　小夏建司　　当麻喜弘

東京工業大学 情報工学科

論理回路の検査入力生成を並列システムで行なう方法を新しく提案する．本方式は，回路の出力で誤りを観測するための回路の入力を求める問題を，回路を出力から入力へ逆向きに探索することによって部分問題に分割し，各部分問題を並列に処理する．また，本方式の効果を評価するために，ハイパーキューブ接続，メッシュ接続等の並列システム上での本方式の処理のシミュレーションを行なった．その結果，どちらのシステムでも検査入力生成処理の高速化が得られることを確認した．

Automatic Test Pattern Generation using Parallel Processing System

Takashi MINOHARA　Yoichiro UENO　Kenji KONATSU　Yoshihiro TOHMA

Department of Computer Science, Tokyo Institute of Technology

2-12-1, Ookayama, Meguro-ku, Tokyo, 152, Japan

In this paper, a new ATPG method for parallel processing system is described. The problem to find patterns of primary inputs for observing errors at primary outputs is divided into subproblems by backward tracing a circuit from outputs to inputs, and each subproblem is solved in parallel. In order to estimate an efficiency of this method, its behavior on a hypercube connected computer and a mesh connected computer is simulated. The result shows it can speed up test generation on both systems.

1 はじめに

論理回路の大規模複雑化により，テスト入力生成問題はますます困難かつ重要になってきている．従来のテスト生成アルゴリズムの大部分は，目標とする単一の故障に対するテスト入力を生成するもの[1),2)]で，回路全体の故障に対するテスト入力集合を求めるためには，通常，故障シミュレーションと組み合わせて処理を行なう．これまで，テスト生成処理の高速化手法として，発見的手法を用いて解の探索を有利に導く方法[3),4)]が報告される一方，並列処理による高速化を行なうために，故障集合を分割して各部分の故障に対するテスト生成を並列に処理する方法[5),6)]が提案されている．

テスト生成処理は対象回路の出力で誤りを観測するための入力の条件を求める問題と考えられる．本報告で我々は，回路を出力から入力に向かって逆向きに探索することにより問題を部分問題に分割し，並列に処理する方法を新しく提案する．本方式は，目標故障を設定してテスト入力を求める方法でないため故障シミュレーションを必要としないこと，処理結果として得られた各テストパターンで検出可能な故障の情報が得られることなどの特徴を持つ．また，本方式の処理をモデル化し，計算機シミュレーションによってハイパーキューブ接続，メッシュ接続等の並列システム上でのテスト生成処理時間短縮の効果を評価する．

2 並列テスト生成法

以下，本方式が対象とする回路は，AND，OR，NOT，NOR，NAND，XOR，XNORから構成された組合せ回路であり，対象とする故障は，信号線の単一縮退故障(s-a-0またはs-a-1)である．また，信号線の状態を表すシンボルとして**0**，**1**，**D**，$\overline{\mathbf{D}}$および*を用いる．**0**(**1**)は，信号線の値が0(1)であることを，**D**($\overline{\mathbf{D}}$)は，信号線が1から0(0から1)に誤る誤りを，*は，信号線の値が不定であることを表す．

2.1 問題の分割による並列処理

回路の故障は回路にテスト入力を与え，出力で誤りを観測することによって検査される．したがって，対象回路のPrimary Output(以下POとする)で誤りが観測されるためのPrimary Input(以下PIとする)の条件を作成することによってテスト入力を生成できる．

POの誤りは，その出力をドライブしているゲートの出力に誤りが現れるときに観測される．これはゲートの出力信号線に故障がある場合，あるいはゲートの入力の誤りが出力まで伝搬した場合のいずれかであり，前者は正常時の出力を得るための入力をゲートに与えればよいことから，ゲート出力に誤りが現れるためのPIの条件を求める問題は，ゲート入力を適切な状態するためのPIの条件を求める問題に変換できる．このときのゲート入力の状態はただ一つ定まるわけではなく，誤りの伝搬のしかたなどによって複数の状態が考えられ，そのいずれもが故障を検査している状態になり得る．例えば，図1の回路の出力L1で誤り$\overline{\mathbf{D}}$

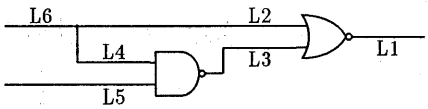

図1: 回路例1

が観測されるのは，NORゲートの入力に誤りが伝搬する{ L2 = **D**, L3 = **0**}，{ L2 = **0**, L3 = **D**}，{ L2 = **D**, L3 = **D**}およびNORゲートの出力自身に故障がある場合{ L2 = **1**, L3 = *}，{ L2 = *, L3 = **1**}の5通りの状態に変換される．

ゲートの入力の状態は，すなわち，そのゲートをドライブしているゲートの出力，あるいはそのゲートの入力がPIとなっている場合はPIの状態であるから，ゲートの入力を特定の状態にするためのPIの条件を求める問題は，そのゲート入力をドライブするゲート出力またはPIを適切な状態にするためのPIの条件を求める問題に変換できる．ただし，複数の入力信号線を持つゲートの場合，各入力信号線が同時にその状態になることが要求される．したがって，各入力信号線それぞれを与えられた状態にするためのPIの条件をすべて満たすPIの条件が得られたとき，そのPIの条件が当該ゲートの入力を与えられた状態にするためのPIの条件になる．図1のNORゲートの入力が{ L2 = **D**, L3 = **0**}となるためのPIの条件は，L2を**D**とするPIの条件として{ L6 = **1**}，L3を**0**とするPIの条件として{ L5 = **1**, L6 = **1**}が得られたら，{ L5 = **1**, L6 = **1**}で与えられる．

このように，POで誤りが観測されるためのPIの条件を求める問題を入力側の問題に順次変換していくことにより，最終的に各PIを与えられた状態にするためのPIの条件を求めるという問題にまで分割できる．この問題の解は，与えられたPIの状態の無故障時の値をPIに入力するという条件で与えられ，この解を使って今度は出力側に向かって問題を解いていくことにより，目的のテスト入力を作成できる．したがって，テスト生成処理は，ゲート出力を与えられた状態にするためのPIの条件を求める処理(以下この処理をOR処理と呼ぶ)およびゲート入力を与えられた状態にするためのPIの条件を求める処理(以下この処理をAND処理と呼ぶ)を次々に起動していくことによりな

される.

OR処理, AND処理の内容は, それぞれ次のようになる.

- OR処理
  1. 出力の状態とゲートの種類から入力の状態を生成する.
  2. 入力の各状態ごとに, AND処理を起動し, 結果を得る.
  3. 各状態についての処理結果を統合する.
- AND処理
  1. 入力の状態から各入力信号線をドライブしているゲートの出力の状態を得る.
  2. 各信号線ごとに, OR処理を起動し, 結果を得る.
  3. 各信号線から得られた結果を同時に満足するPIの条件を作成する.

図1の回路に対する各処理の起動をグラフで表すと図2のようになる. 図において上から下に向かって処理の起動

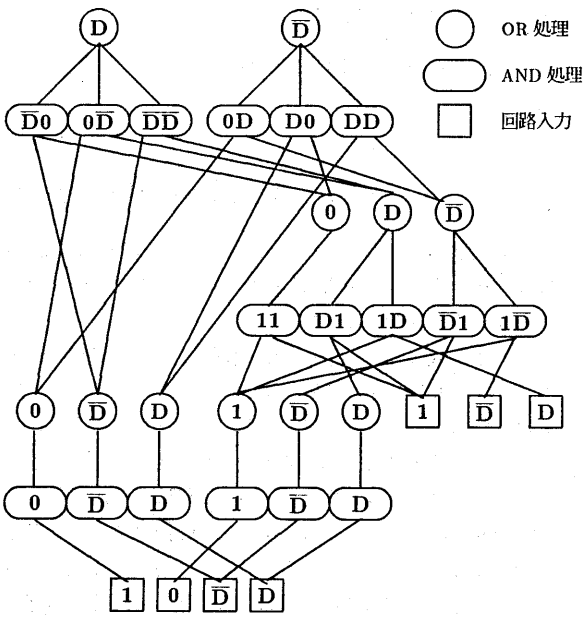

図2: 回路例1のテスト生成処理

が進み, 逆に下から上に向かって処理結果が渡される. 親子関係にない処理は結果の受渡しを行なわないため, 並列に処理することが可能である.

2.2 テスト候補と故障情報

ある故障を検査できるテストが求まったとき, そのテストはその故障の影響である誤りが伝搬する経路上の故障を同様に検査可能である. したがって, 本テスト生成法において回路中のゲートの入出力を与えられた状態にするためのPIの条件すなわちテスト候補を作成する際に, 誤りが伝搬する経路上の故障の情報を付加していくことにより, 最終的に求められたテストで, 回路中のどの故障を検査できるかの情報を得ることができる. ただしファンアウト, 再収斂を含む回路では, 故障の影響が複数の信号線を伝搬することによるセルフマスキングや多重経路による故障検出等を考慮しなければならない.

セルフマスキングの例として, 図1の回路で, L2に誤り ($1 \rightarrow 0$) が現れ, L3に正常値0が現れることによりPOに誤りが伝搬する状態の処理を考える. L2に誤り ($1 \rightarrow 0$) が現れるための条件(1), L3に正常値0が現れる条件(2)はそれぞれ次のようになるが,

回路入力:(L6=1)	故障:(L6 s-a-0, L2 s-a-0)	(1)
回路入力:(L6=1, L5=1)	故障:()	(2)

L6のs-a-0による誤りはL6からL2へと伝搬するだけでなく, L4, L3と伝搬するため, L3に正常値が得られず, L6のs-a-0は検出できない. したがって故障の情報からL6のs-a-0を取り除く必要が生じる.

このようなセルフマスキングや多重経路による故障検出等に対応するため, 本方式では, 故障情報に fanout stem(FOS) の状態を加え, FOSの状態を考慮して故障情報の更新を行なっている. 例えば上記の例では, 条件(1)でL6のs-a-0はFOSであるL6に誤りが伝搬してもよいときに誤りを伝えること, 条件(2)で誤りが伝搬してこないためにはFOS L6に誤りが伝搬してはいけないことを情報として持たせ. L6のs-a-0は2つの条件を同時に満足しないので故障情報から削除する. セルフマスキング, 多重経路伝搬を処理するためのFOSの情報を含めたテスト候補の定義および表記法を図3に示す.

故障情報はAND処理において複数のテスト候補からそれらを同時に満たす条件を求めるときに更新される, 故障情報の更新処理は誤りがいずれか1つの信号線に伝搬することを要求している場合(単一伝搬)と, 複数の信号線に同時に誤りが伝搬することを要求している場合(多重伝搬)とで異なりそれぞれ次のような処理を行なう.

- 単一伝搬の場合
  - 新しい直接伝搬故障情報は, 各テスト候補の直接伝搬故障情報に含まれる故障で構成する.
  - 新しいファンアウト故障情報は, 各テスト候補の同一のFOSに対するファンアウト故障情報ごとに,
    * すべてのFOSが誤りの伝搬を示している場

< テスト候補 > ::= '{' < テスト入力パターン > '}''[' < 故障情報 > ']'

< テスト入力パターン > ::= < 入力 > | < 入力 >< テスト入力パターン >

< 入力 > ::= < 信号線名 > ' → ' < 信号値 >

< 故障情報 > ::= '(' < 直接伝搬故障情報 > ')' < 故障情報' >

< 故障情報' > ::= *empty* | '(' < ファンアウト故障情報 > ')' < 故障情報' >

< 直接伝搬故障情報 > ::= *empty* | < 故障 >< 直接伝搬故障情報 >

< 故障 > ::= < 信号線名 > ' → ' < 誤り >

< ファンアウト故障情報 > ::= '<' < ファンアウト情報 > '>'

|'<' < ファンアウト情報 > '>' < ファンアウト故障情報' >

< ファンアウト故障情報' > ::= *empty* | < 故障 >< ファンアウト故障情報' >

図 3: テスト候補

合は，ファンアウト情報としてそのFOSの誤りを示し，各ファンアウト故障情報に含まれる故障でファンアウト故障情報を構成する．

* すくなくとも1つのFOSが正常値の伝搬を示している場合は，ファンアウト情報としてその正常値を示し，故障を含まないファンアウト故障情報を構成する．

- 多重伝搬の場合
  - 新しい直接伝搬故障情報は，空にする．
  - 新しいファンアウト故障情報は，各テスト候補の同一のFOSに対するファンアウト故障情報ごとに，
    * 同時に誤りが伝わるとしているすべての信号線のテスト候補にそのファンアウト故障情報が含まれ，かつすべてのFOSが誤りの伝搬を示している場合，ファンアウト情報としてそのFOSの誤りを示し，各ファンアウト故障情報に含まれる故障でファンアウト故障情報を構成する．
    * 同時に誤りが伝わるとしている信号線のうち一部のテスト候補に含まれるファンアウト故障情報，およびすくなくとも1つのFOSが正常値の伝搬を示しているファンアウト故障情報は，ファンアウト情報として故障のないときの正常値を示し，故障を含まないファンアウト故障情報を構成する．

例えば，図1の回路で，L2に誤り ($1 \rightarrow 0$) が現れるためのテスト候補として，

{ L6→1 } [(L2→D) (< L6→D > L6→D)]

L3に正常値0が現れるためのテスト候補として，

{ L6→1 L5→1} [()(< L6→1 >)]

が得られたとき，NORゲートの入力が{ L2 = **D**, L3 = **0**}となるためのテスト候補は，

{ L6→1 L5→1 } [(L2→D) (< L6→1 >)]

になる．

3 シミュレーションによる効率評価

並列処理によりどのような速度向上が得られるかを評価するため，本テスト生成法の並列システム上での動作のシミュレーションを行なった．

テスト生成のAND処理，OR処理に要する時間は，処理するテスト候補によって変化するため，実際にテスト候補作成を行なわなければ評価できない．しかし，シミュレーションの対象とする並列システムのパラメータを変化させるたびにテスト生成処理を行なって処理時間を決定した場合，1回のシミュレーションにかかる時間が大きくなってしまう．そこで，我々はテスト生成の処理をモデル化し，単一プロセッサ上での処理時間の記録を用いて，並列処理のシミュレーションを行なった．

3.1 テスト生成処理のモデル

OR処理，AND処理は，親子関係にある処理とのデータの受渡しに注目すると，いずれも，

1. 親処理から受けとった信号線の状態パターンから子処理へ渡す状態パターンを作成する．
2. 子処理を起動し，処理終了を待つ．

3. 子処理から受けとった処理結果から親処理へ渡すテスト候補を作成する.

という処理になる. ここで, 子処理の起動は一度に行ない, 子処理からの結果がすべて揃ってからテスト候補作成を行なうとし, 処理のモデルを,

1. 処理が起動されてT_1時間後に子処理の起動を行なう.
2. 子処理の処理結果がすべて得られてからT_2時間後に処理を終了し, 処理結果を親処理に送る.

と定める. 処理時間T_1, T_2は単一プロセッサ上で実際にテスト生成処理を行ない, 各処理にかかった時間を記録することにより定める. また, 処理の起動は親処理からの固定長のメッセージの到着により行なわれるとし, 処理結果の受渡しのメッセージ長は, 実際のテスト生成処理で記録したテスト候補の大きさに比例するものとする.

3.2 並列処理システムのモデル

シミュレーションの対象とする並列処理システムのモデルを次のように定める.

- システムの構成は, 2^n個$(n = 1, 2, 3\cdots)$のPEがハイパーキューブ接続されたもの, あるいは, 2^{2m}個$(m = 1, 2, 3\cdots)$のPEが$2^m \times 2^m$のメッシュ(同一行, 同一列でのラップアラウンドを持つ)接続されたものである.
- 各PEは独立した処理を行なうことができる.
- PE間の通信路は双方向である.
- 各PEでの処理と並行して, 通信を行なうことができる.
- 1つのPEに複数の処理が割り当てられた場合, ラウンドロビン式のマルチタスクで処理される.

3.3 プロセッサへの処理の割り当て

本テスト生成法では, 複数のAND処理において, 同一の信号線の同一状態に対するOR処理の処理結果を必要とする場合がある. このとき, それぞれについてひとつづつOR処理を起動すると, 処理の数が指数的に増えることになる. そこで, 各信号線の一状態に対するOR処理はただ一つだけ起動し, 処理結果の共有を行なう.

OR処理の処理結果を共有するためには, AND処理は同一のPEに対してOR処理の起動を要求しなければならない. しかし, AND処理間でPEへのOR処理の割り当ての統一をとることは困難である. ここで, OR処理の祖父となるOR処理に注目すると, ファンアウトステム(FOS)以外の処理を行なうOR処理は同一の祖父を持つことから, OR処理のPEへの割り当ては祖父のOR処理により決定可能である. 一方, FOSの処理を行なうOR処理は複数の祖父を持ち, 祖父による割り当てができない. このようなOR処理は, 全処理の起動前にあらかじめ割当を決定しておくことにする. また, AND処理については処理結果の共有を行なわないため, 親であるOR処理がPEへの割り当てを決定する.

並列処理システムがハイパーキューブ接続の場合, 各処理のPEへの割り当ては次のようにして決定する. ただし総PE数は2^nとし, 各PEのアドレスを$(a_{n-1}a_{n-2}\cdots a_1a_0)$とする.

- OR処理の割り当て:

  まず, レベルiのPE:$(a_{n-1}a_{n-2}\cdots a_j\cdots a_1a_0)$が自分自身とPE:$(a_{n-1}a_{n-2}\cdots \overline{a_j}\cdots a_1a_0)(j = i \bmod n)$に接続された2進木を考える. FOSおよびPOの処理を行なうOR処理は, その数が$k(2^{l-1} < k \leq 2^l)$個であれば, レベルlのPEに割り当てる. それ以外のOR処理は割り当てを行なうPEのレベルがi, 割り当てるOR処理の数が$k(2^{l-1} < k \leq 2^l)$個のとき, 割当を行なうPEの2レベル上$(i-2$レベル$)$のPEから分岐するレベル$(i-2+l)$のPEに割り当てる.

- AND処理の割り当て:

  AND処理は, その子供であるOR処理への経路上に割り当てる,

また, 並列処理システムがメッシュ接続の場合, 各処理のPEへの割り当ては次のようにして決定する. ただしPEは2^n行2^n列のメッシュを構成し, アドレスは$(x, y)(0 \leq x, y \leq (2^n - 1))$で与えられるものとする.

- OR処理の割り当て:

  各信号線につき1つの処理の割り当てを次のように行ない, 同一信号線の他状態の処理は, 割り当てられたPEのアドレスが(x, y)のとき$(x+1, y)$, $(x, y+1)$, $(x+1, y+1)$に割り当てる.

  まず, POおよびFOSの処理をPE間の距離が最大になるように割り当てる. つぎに割り当てられたPEをルート(レベル0)として, レベルiのPE:(x, y)が, iが偶数の場合 PE:$(2x+1, y)$および PE:$(2x-1, y)$, iが奇数の場合 PE:$(x, 2y+1)$および PE:$(x, 2y-1)$に接続された2進木を考える. 各PEが自分の孫の処理として割り当てるOR処理の対象とする信号線数が$k(2^{l-1} < k \leq 2^l)$個のとき, そのPEから分岐しlレベル下のPEにOR処理を割り当てる.

- AND処理の割り当て:

  AND処理は, 親であるOR処理のPEに隣接する4つのPEに割り当てる.

3.4 シミュレーション結果

7483(4bit full adder)，7485(4bit magnitude comparator)，74181(4bit ALU)，および74630(ECC)のパリティジェネレータ部の4つの回路を対象としてテスト生成処理を行ない，次式で表されるPEn台の相対処理速度をシミュレーションにより求めた結果を図4，5に示す．なお，シミュレーションにあたって，PE間の通信速度は，PE1台の処理速度を処理時間計測に使用したワークステーション(SUN3/60)と同程度とすると，約500KByte/secに相当するように設定した．

$$\text{PE}n\text{台の相対処理速度} = \frac{\text{PE1台の処理時間}}{\text{PE}n\text{台の処理時間}}$$

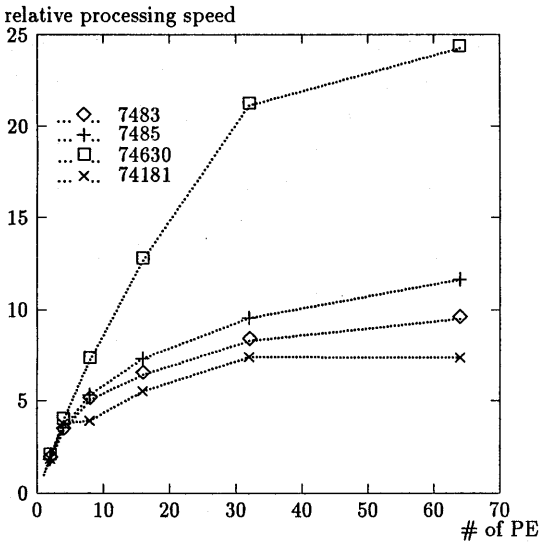

図4: ハイパーキューブ接続での相対処理速度vs PE数

ハイパーキューブ接続，メッシュ接続のいずれのシステムでも並列処理により処理速度の向上が得られることが確認される．74630は特に処理速度向上が大きいが，この回路は回路の大きさに対してPIからPOまでのゲート段数が少なく，並列に処理できる信号線が多いためだと考えられる．

また，74181ではハイパーキューブ接続よりもメッシュ接続の方が大きい相対処理速度を得ている場合があり，現行のPEへの処理の割り当て方法に問題があることが分かる．

4 まとめ

並列システム上での論理論理回路の検査入力生成法として，回路に故障の影響が伝搬する条件を回路出力から入力に向かって逆向きにたどることにより問題の分割を行なう方法を提案した．またハイパーキューブ接続，メッシュ接続の並列システム上での処理時間短縮の効果を計算機シミュレーションにより確認した．また，現在のPEへの処理の割り当て方法に改善の余地があることも分かった．今後は，より効率的な処理の割り当て方法について考えていきたい．

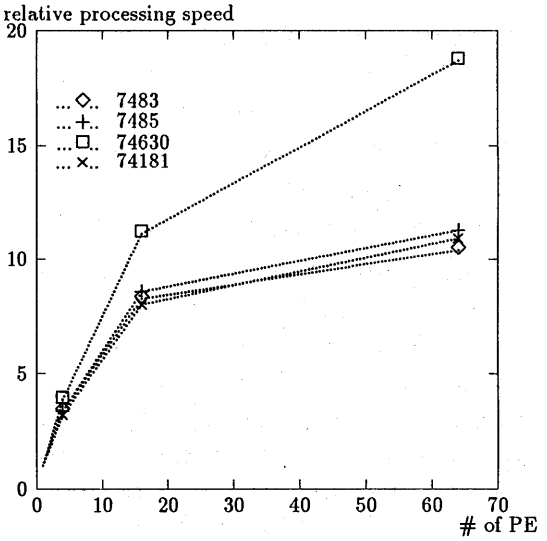

図5: メッシュ接続での相対処理速度vs PE数

参考文献

1) J. P. Roth: "Diagnosis of automata failures: a calculus and a method", IBM J. Res. & Dev., **10**, 4, pp. 278–291 (July 1966).

2) P. Goel: "An implicit enumeration algorithm to generate tests for combinational logic circuits", In Proc. 10th Int. Symp. Fault-Tolerant Comput., pp. 145–151 (Oct. 1980).

3) H. Fujiwara and T. Shimono: "On the acceleration of test generation algorithms", IEEE Trans. Comput., **C–32**, 12, pp. 1137–1144 (Dec. 1983).

4) M. H. Schulz and E. Auth: "Advanced automatic test pattern generation and redundancy identification techniquies", In 18th Int. Symp. Fault-tolerant Comp., pp. 30–35, IEEE (June 1988).

5) H. Fujiwara and A. Motohara: "Fast test pattern generation using a multiprocessor system", Trans. IEICE, **E 71**, 4, pp. 441–447 (April. 1988).

6) S. Patil and P. Banerjee: "Fault partitioning issues an intergrated parallel test generation/fault simulation environment", In Proc. Int. Test Conf., pp. 718–726, IEEE (1989).