設計自動化　68－13
（1993. 10. 29）

並列テスト生成におけるテストパターン数について

井上 智生　　藤原 秀雄

奈良先端科学技術大学院大学　情報科学研究科
奈良県生駒市高山町8916-5

あらまし　本稿では，並列テスト生成において生成されるテストパターン数について考察する．複数のプロセッサで同時にテストパターンを生成する並列テスト生成では，一台のプロセッサによる処理に比べて多くのテストパターンを生成する傾向にある．本稿では，生成されるテストパターン数とそのパターンによって検出される故障数との関係を定式化し，並列テスト生成によって生成されるテストパターン数を解析する．さらに，プロセッサ間通信の生成されるテストパターン数への影響について解析し，効果的な通信方法を示す．

和文キーワード　テスト生成，並列処理，マルチプロセッサ，故障並列法，テストパターン数

On the Test Set Size in Parallel Test Generation

Tomoo INOUE　　Hideo FUJIWARA

Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara

Abstract　In this paper, we consider the test set size produced in parallel test generation for logic circuits. Since in a multiple processor system, each processor generates test-patterns independently and simultaneously, the total set of test-patterns tends to be larger than that generated by a single processor system. In this paper, we formulate the relation between the number of generated test-patterns (test set size) and the number of faults detected by the patterns, and analyze the total number of test-patterns produced in parallel test generation. Further, we analyze the influence of communication among processors on the number of test-patterns, and show an effective method of communication which can obtain a small size of test set in parallel test generation.

英文 key words　Test generation, Parallel processing, Multiple processor, Fault parallelism, Test set size

1. はじめに

回路規模が増大し，ますます困難となる論理回路のテスト生成処理を高速化する一つの手段として並列処理がある．並列テスト生成は，問題の分割方法や処理方法によって，故障並列法，探索並列法，回路並列法，方策並列法に分類できる．

故障並列法については，藤原と井上が[1]において，一台のクライアント・プロセッサと複数台のサーバ・プロセッサから成る疎結合分散型ネットワークにおける並列テスト生成を提案している．また，Patilと Banerjee[2] は，すべてのプロセッサが対等な処理を行うマルチプロセッサシステムを用いた故障並列法を提案している．これら２つの方法は，いずれも処理の高速化を第一の目的としており，生成されるテストパターン数については十分検討されていない．

これらの並列テスト生成では，各プロセッサでテストパターン生成と故障シミュレーションが行われる．複数のプロセッサが同時にテストパターンを生成するとき，それぞれのプロセッサは他のプロセッサが生成するテストパターンとは無関係にテストパターン生成を行う．そのため，たとえテストパターン生成の目標となった故障が他のプロセッサで生成されたパターンで検出可能であったとしても，別のパターンを生成してしまう．この処理は生成されるテストパターンの増加をもたらす．テストパターンの増加は並列処理の効果を下げ，高速化を妨げるだけでなく，テスト実行時間を増大させる原因ともなる．本稿では，上記の２つの故障並列法[1],[2]において生成されるテストパターン数について解析を行う．Goelが[3]で示した，シングルプロセッサでのテストパターン数と検出故障数との関係をもとに，それぞれの故障並列テスト生成におけるテストパターン数と検出故障数との関係を定式化し，解析を行う．また，並列テスト生成処理で行われるプロセッサ間の通信が生成されるテストパターン数にもたらす影響を解析し，テストパターン数を小さくするための通信方法について考える．

2 故障並列法によるテスト生成方式

与えられた故障の集合を分割して，各プロセッサに割り当てる並列テスト生成の方式を**故障並列法**という．故障並列法によるテスト生成は，藤原，井上[1]とPatil and Banerjee[2] によって報告されている．ここでは，本稿で解析するこれらの並列テスト生成の方式を説明する．

2.1 クライアント・サーバ方式

（CS方式：藤原，井上 [1]）

一台のクライアント・プロセッサとN台のサーバ・プロセッサが一本のネットワークを介して接続されている（図１）．クライアントは故障集合を故障表の形で管理する．クライアントは故障表の中から未処理の故障を複数個取り出し，これらを**ターゲット故障**として故障表とともにサーバへ転送する．サーバは，クライアントから受け取ったターゲット故障の中から，一個の故障を取り出し，その故障に対するテストパターン生成を行う．そして，生成されたパターンを用いて，ターゲット故障だけでなく故障表中の未処理の故障すべてを対象に故障シミュレーションを行う．サーバはクライアントから受け取ったターゲット故障がすべて処理されるまでこれを繰り返し，結果をクライアントへ返送する．

クライアントはサーバからの結果をもとに故障表を更新し，新たなターゲット故障をサーバへ転送する．クライアントは，故障表中のすべての故障が処理されるまでこれを繰り返す．

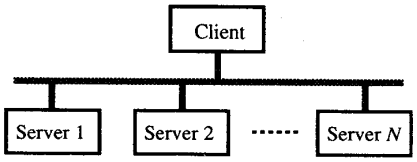

図１． CS方式

2.2 Patil and Banerjeeの方式

（PB方式：Patil and Banerjee [2]）

N台のプロセッサから成り，すべてが互いに通信可能な経路を持つネットワークで構成される（図２）．各プロセッサにはあらかじめ，全故障をプロセッサ数Nで等分した故障が割り当てられる．各プロセッサは割り当てられた故障集合から一個の故障を取り出し，その故障に対するテストパターン生成を行う．そして，割り当てられている故障のうちの未処理の故障を対象に，その生成されたパターンを用いて故障シミュレーションを行う．生成したパターンは他のすべてのプロセッサに転送される（**パターン・ブロードキャスト**）．パターンを受け取ったプロセッサは，そのパターンを用いて，自分に割り当てられている故障を対象に故障シミュレーションを行う．プロセッサは，割り当てられている故障の検出率があらかじめ決められた値p_c（**通信遮断係数**）に達するまで，このパターン・ブロードキャストを行う．

プロセッサは，割り当てられた故障の処理をすべて終えると，他のまだ処理を終えていないプロセッサから，それぞれに残存する負荷の情報（故障数）を収集する．その情報をもとに，もっとも大きい残存負荷を持つプロセッサからその負荷の半分を受け取り，処理を行う（**動的負荷平衡**）．各プロセッサについて以上の処理が終了することで，全体の処理は完了する．

本稿では，通信遮断係数p_cが次の３つの場合について解析を行う．

(1) $\mathrm{PB}_{\mathrm{nb}}$方式（$p_c = 0$）

パターンの送受信は行われない．各プロセッサが故障シミュレーションで用いるテストパターンは，それぞれで生成されたテストパターンのみである．

(2) $\mathrm{PB}_{\mathrm{brd}}$方式（$p_c = 1$）

プロセッサは生成したすべてのパターンを他のプロセッサへブロードキャストする．すなわち，各プロセッサはシステム全体で生成されるすべてのテストパターンを用いて，自分に割り当てられている故障に対して故障シミュレーションを行う．

(3) $\mathrm{PB}_{\mathrm{pc}}$方式（$0 < p_c < 1$）

プロセッサは，割り当てられている故障の検出率が

通信遮断係数p_cに達するまで生成したテストパターンを他のプロセッサへブロードキャストし，検出率がp_cを越えた後はブロードキャストを行わない．

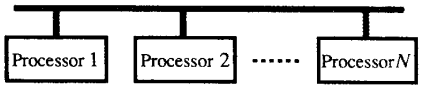

図２．PB方式

3. テストパターン数と検出故障数との関係

Goelは[3]において，生成されるテストパターン数とそのパターンによって検出される故障数との関係を解析している．本稿ではこの解析結果をもとに，マルチプロセッサシステムで生成されるテストパターンの数について考察する．

3.1 シングルプロセッサシステムにおけるテストパターン数

図３にGoel[3]の解析結果を示す．横軸は生成されたテストパターンの数，縦軸はそれらのパターンでは検出できない，すなわち，まだ検出されずに残っている故障の数を表している．この故障を残存故障と呼ぶ．Goelはこの図に示すように，テストパターン数と残存故障数との関係を，指数関数（フェーズI）と一次関数（フェーズII）との組み合わせによって表現し，それらの関数を

$$F = Me^{-at/T_I} \qquad (0 \le t \le T_I) \tag{1}$$

$$F = \alpha t + \beta \qquad (T_I \le t \le T_{\text{single}}) \tag{2}$$

としている．tは生成されたテストパターン数，Fはそのときの残存故障数を表し，Mは与えられた回路の全故障数を表す．a, α, βは与えられた回路から決まる定数である．また，T_{single}はすべての故障を検出するのに必要なテストパターン数，T_Iは残存故障数の変化がフェーズIからフェーズIIへ切り換わるテストパターン数を表す．T_I本のテストパターンが生成されたときの残存故障数をμMとすると，各定数は，

$$a = -\log \mu \tag{3}$$

$$\alpha = -\frac{\mu M}{T_{\text{single}} - T_I} \tag{4}$$

$$\beta = \frac{\mu M T_{\text{single}}}{T_{\text{single}} - T_I} \tag{5}$$

と表すことができる．

テストパターン数に対する残存故障数の変化を式(1),(2)のように考えたとき，それぞれのフェーズは次のように考えることができる．

（フェーズI）新たに検出される故障数は残存する故障の数に比例する．故障シミュレーションの効果が反映される区間．

（フェーズII）残存故障数が小さく，故障シミュレーションの効果が小さい．残存する故障数とは無関係と考えられる区間．

式(1)において

$$q = e^{-a/T_I} \tag{6}$$

とおくと，

$$F = Mq^t \qquad (0 \le t \le T_I) \tag{7}$$

と書き換えることができる．これより，フェーズIからフェーズIIへ切り換わるときの全故障数に対する残存故障数の比μは，

$$\mu = q^{T_I} \tag{8}$$

と表すことができる．本稿では，回路のテストパターン数と残存故障数との関係は３つの定数q, μ, αによって表されるものとし，式(7),(2)をもとに，マルチプロセッサシステムで生成されるテストパターン数と残存故障数との関係を解析する．

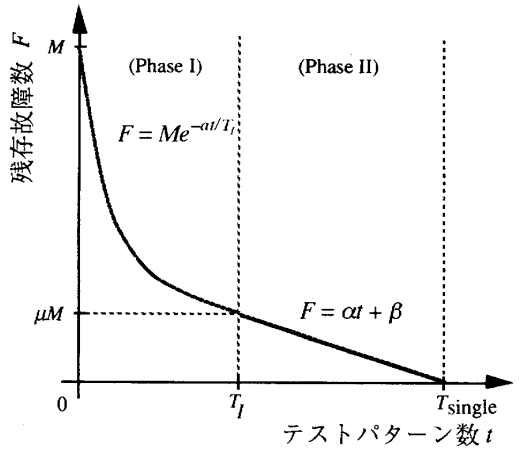

図３．テストパターン数と残存故障数との関係

3.2 マルチプロセッサシステムにおけるテストパターン数

マルチプロセッサシステムで生成されるテストパターンの数は，各プロセッサで生成されるテストパターンの数の合計と考えることができる．マルチプロセッサシステムで生成されるテストパターン数と残存故障数との関係を定式化する上で，以下の２つを仮定する．

仮定１：与えられた回路の全故障の集合と，その集合から無作為に故障を取り出して得られる任意の部分集合を考えたとき，その部分故障集合に対するテストパターン数と残存故障数との関係は，全故障集合に対するテストパターン数と残存故障数との関係と同様であるものとする．すなわち，全故障集合に対するテストパターン数と残存故障数との関係が定数q, μ, αによって表されるとき，任意の部分故障集合に対してテスト生成を行ったときの残存故障数の変化もまた，qを底とする指数関数フェーズIと傾きαの一次関数フェーズIIによって表され，２つのフェーズが切り換わる点はその部分故障集合の残存故障の比がμのときとする．

仮定２：各プロセッサに割り当てられる負荷は均等でかつ均質であるとする．すなわち，各プロセッサに割り当てられる故障の数は等しく，かつ，テスト容易故障もテスト困難故障も等しい割合で分配されるものとする．したがってこの仮定のもとでは，PB方式の動的負荷平衡は発生しないものと考えることができる．

以上の仮定より，N個のプロセッサから成るマルチプロセッサシステムにおけるプロセッサあたりの残存故障数は，

$$F = F_0 q^n \qquad \left(0 \le n \le T_I'\right) \tag{10}$$

$$F = \alpha n + \beta' \qquad \left(T_I' \le n \le T'\right) \tag{11}$$

となる．ただし，F_0はプロセッサに割り当てられる故障数，nは処理されるパターン数，β'は定数とする．また，T_I'は割り当てられた故障の残存故障数がμF_0になるのに必要なテストパターン数，T'は割り当てられた故障がすべて検出されるまでに必要なテストパターン数を表す．

したがって，N個のプロセッサから成るマルチプロセッサシステム全体で生成されるテストパターン数は，

$$T_{\text{multi}} = NT' \tag{12}$$

となる．

3.3 PB_{nb}方式によるテストパターン数

PB_{nb}方式ではパターン・ブロードキャストは行われない．各プロセッサが行うテストパターン生成及び故障シミュレーションの対象は，いずれも初めに割り当てられた故障のみである．したがって，全故障数をMとしたとき，一台のプロセッサに割り当てられる故障数F_0は，

$$F_0 = \frac{M}{N} \tag{13}$$

となる．また，プロセッサが生成するテストパターン数と故障シミュレーションに用いるパターン数とは等しいので，プロセッサが処理の対象とするテストパターン数nは，

$$n = t \tag{14}$$

とおくことができる．したがって，PB_{nb}方式におけるフェーズIの残存故障数は，式(10)に式(13),(14)を代入して，

$$F = \frac{M}{N} q^t \tag{15}$$

と表すことができる．プロセッサに割り当てられた故障数に対する残存故障数の比がμになる，すなわちフェーズIが終了するテストパターン数をT_{Inb}で表せば，

$$\mu \frac{M}{N} = \frac{M}{N} q^{T_{Inb}} \tag{16}$$

が成り立つので，

$$T_{Inb} = \frac{\log \mu}{\log q} \tag{17}$$

$$= T_I \tag{18}$$

となる．すなわち，一台のプロセッサがフェーズIで生成するテストパターン数は，シングルプロセッサシステムが全故障を対象にフェーズIで生成するテストパターン数に等しい．

プロセッサが割り当てられた故障をすべて検出するのに必要なテストパターン数をT'_{nb}とする．フェーズIIにおける残存故障数は，式(11)に式(14)を代入して，

$$F = \alpha t + \beta' \tag{19}$$

が得られる．ここで，

$$\mu \frac{M}{N} = \alpha T_{Inb} + \beta' \tag{20}$$

$$0 = \alpha T_{nb}' + \beta' \tag{21}$$

が成り立つ．これらの2式(20),(21)に式(4)を代入して，

$$\mu \frac{M}{N} = \frac{\mu M}{T_{\text{single}} - T_I}\left(T_{nb}' - T_{Inb}\right) \tag{22}$$

が得られる．これより，フェーズIIで生成されるテストパターン数は，

$$T_{nb}' - T_{Inb} = \frac{T_{\text{single}} - T_I}{N} \tag{23}$$

と表すことができる．このように，フェーズIIで各プロセッサが生成するテストパターン数は，シングルプロセッサシステムで生成されるテストパターン数に比べて$1/N$倍になる．

また，プロセッサが割り当てられた故障をすべて検出するのに必要なテストパターン数T'_{nb}は，

$$T_{nb}' = \frac{T_{\text{single}} - T_I}{N} + T_I \tag{24}$$

となる．したがって，PB_{nb}方式で生成されるテストパターンの総数をT_{nb}で表すと，式(12)より，

$$T_{nb} = NT_{nb}' \tag{25}$$

$$= T_{\text{single}} + (N-1)T_I \tag{26}$$

が得られる．このようにPB_{nb}方式の並列テスト生成では，フェーズIで生成されるテストパターン数は，シングルプロセッサシステムで生成されるテストパターン数T_Iに比べてN倍に増加する．これは，自分に割り当てられた故障を他のプロセッサで生成されたテストパターンを用いてテストすることができないためと考えることができる．一方フェーズIIでは，テストパターン生成の目標となる故障以外はそのパターンによって検出されることはないので，生成されるテストパターンの増加は起こらない．

3.4 PB_{brd}方式によるテストパターン数

プロセッサは一本のテストパターンを生成し，そのパターンを用いて故障シミュレーションを行うが，それと同時に他のプロセッサへその生成したテストパターンを送付する．もし各プロセッサへの負荷が均等かつ均質に分配され，同時にこの処理が進行しているとすれば，$N-1$台のプロセッサからそれぞれ一本ずつテストパターンを受け取ることになることになる．プロセッサは受け取った$N-1$本についても故障シミュレーションを行う．すなわち，プロセッサは一本のテストパターンを生成した後，N本のパターンを用いて故障シミュレーションを行うものと考えることができる．

しかしこれらのN本のテストパターンは，それぞれが他の$N-1$本のテストパターンが検出する故障とは無関係に生成されたものであるため，一台のプロセッサで逐次的に生成されたN本のテストパターンに比べて，検出できる故障数は小さくなると思われる．この検出できる故障数の減少度は，並列度Nが大きくなるにつれて高くなるものと考えられる．この性質を定数r $(0 < r < 1)$を用いて表し，並列に生成されたN本のテストパターンが検出できる故障数は一台のプロセッサで逐次的に生

成されたN^r本のテストパターンが検出できる故障数と等しいものとする．

この仮定に基づき，一台のプロセッサが処理するテストパターン数は，

$$n = N^r t \tag{27}$$

となる．また，

$$F_0 = \frac{M}{N} \tag{28}$$

であるので，フェーズIにおけるt本のテストパターンを生成したときの残存故障数は，

$$F = \frac{M}{N} q^{N^r t} \tag{29}$$

となる．フェーズIの終了するテストパターン数をT_{Ibrd}で表せば，式(16)-(18)と同様にして，

$$\mu \frac{M}{N} = \frac{M}{N} q^{N^r T_{Ibrd}} \tag{30}$$

$$T_{Ibrd} = \frac{1}{N^r} \frac{log\ \mu}{log\ q} \tag{31}$$

$$= \frac{1}{N^r} T_I \tag{32}$$

が得られる．すなわち，生成したテストパターンをブロードキャストすることで，フェーズIで生成されるテストパターンを$1/N^r$にすることができる．

一方フェーズIIでは，生成されたテストパターンが，そのテストパターン生成の目標となった故障以外の故障を検出されることはない．すなわち，他のプロセッサからテストパターンを受け取って故障シミュレーションを行っても，新たに検出される故障はない．したがって，フェーズIIで生成されるテストパターン数は，プロセッサに割り当てられた故障がすべて検出されるのに必要なテストパターン数をT'_{brd}とすると，式(23)と同様に，

$$T'_{brd} - T_{Ibrd} = \frac{T_{\text{single}} - T_I}{N} \tag{33}$$

となる．よって，PB_{brd}方式において生成されるテストパターンの総数は，

$$T_{brd} = N\left(\frac{1}{N^r} T_I + \frac{T_{\text{single}} - T_I}{N}\right) \tag{34}$$

$$= T_{\text{single}} + \left(N^{1-r} - 1\right) T_I \tag{35}$$

となる．このように，パターン・ブロードキャストを行うことによって，パターン・ブロードキャストを行わないPB_{nb}方式に比べて，フェーズIで生成されるテストパターン数の増加をN倍からN^{1-r}倍に抑えることができる．しかし言い換えれば，$N \geq 2$のとき，

$$\left(N^{1-r} - 1\right) > 0 \tag{36}$$

なので，テストパターンは少なくともN^{1-r}倍は増加し，必ず

$$T_{brd} > T_{\text{single}} \tag{37}$$

となる．すなわち，複数のプロセッサで同時にテストパターン生成が行われるテスト生成方式では，逐次的にパターン生成を行うシングルプロセッサに比べて，生成されるテストパターン数は必ず増加するものと考えることができる．またその増加率は，並列度Nが大きくなるにつれて高くなる．

3.5 PB_{pc}方式によるテストパターン数

前節で述べたように，フェーズIIで行われるパターン・ブロードキャストは生成されるテストパターン数に影響しない．プロセッサ間の通信費用をできるだけ小さくして生成されるテストパターン数を小さくするためには，フェーズIのときのみパターン・ブロードキャストを行えばよい．しかし，フェーズIが終了する点μは回路によって決まる値であり，これをあらかじめ予測することは困難である．本節では，$p_c \leq 1 - \mu$，すなわちフェーズIが終了する前に，パターン・ブロードキャストを終了したときのテストパターン数について考察する．

故障検出率がp_cに達するときのテストパターン数をT_pとすると，テスト生成開始からパターン・ブロードキャストが終了するまでの残存故障数は式(29)より，

$$F = \frac{M}{N} q^{N^r t} \qquad \left(0 \leq t \leq T_p\right) \tag{38}$$

となるから，パターン・ブロードキャスト終了時の残存故障数

$$\left(1 - p_c\right) \frac{M}{N} = \frac{M}{N} q^{N^r T_p} \tag{39}$$

より，

$$T_p = \frac{1}{N^r} \frac{log\left(1 - p_c\right)}{log\ q} \tag{40}$$

となる．また，パターン・ブロードキャスト終了後のフェーズIの残存故障数は，式(10)より，

$$F = \left(\frac{M}{N} q^{N^r T_p}\right) q^{t - T_p} \tag{41}$$

$$= \frac{M}{N} q^{\left(N^r - 1\right) T_p + t} \qquad \left(T_p \leq t \leq T_{Ipc}\right) \tag{42}$$

となる．よって，フェーズI終了時のテストパターン数をT_{Ipc}で表せば，

$$\mu \frac{M}{N} = \frac{M}{N} q^{\left(N^r - 1\right) T_p + T_{Ipc}} \tag{43}$$

$$T_{Ipc} = \frac{log\ \mu}{log\ q} - \left(N^r - 1\right) T_p \tag{44}$$

$$= T_I - \left(N^r - 1\right) T_p \tag{45}$$

となる．一方，フェーズIIで生成されるテストパターン数は，式(23),(33)と同様にして求められるので，プロセッサに割り当てられた故障をすべて検出するの必要なテストパターン数T'_{pc}は，

$$T'_{pc} = T_I - \left(N^r - 1\right) T_p + \frac{T_{\text{single}} - T_I}{N} \tag{46}$$

となる．したがって，PB_{pc}方式による並列テスト生成で生成されるテストパターンの総数T_{pc}は，

$$T_{pc} = N T'_{pc} \tag{47}$$

$$= T_{\text{single}} + (N-1) T_I - N\left(N^r - 1\right) T_p \tag{48}$$

で表される．この式を式(26)を用いて書き換えると，

$$T_{pc} = T_{nb} - N\left(N^r - 1\right) T_p \tag{49}$$

と表すことができる．このT_{pc}はパターン・ブロードキ

ャストの回数T_pが大きくなるにつれて小さくなる．すなわち，PB_{pc}方式は，PB_{nb}方式で増加するテストパターン数をパターン・ブロードキャストによって抑えることができる．そして，設定した通信遮断係数p_cがちょうどフェーズIの終了時になったとき，すなわち，$p_c = 1-\mu$のとき，式(40)より，

$$T_p = \frac{1}{N^r}\frac{log\,\mu}{log\,q} \tag{50}$$

$$= \frac{T_I}{N^r} \tag{51}$$

となるので，

$$T_{pc} = T_{single} + \left(N^{1-r}-1\right)T_I \tag{52}$$

となり，式(35)と等しくなる．これは，PB方式で得られる最小のテストパターン数であり，また，これを得るにはすべての生成したテストパターンをブロードキャストする必要はなく，式(51)に示すテストパターン数をブロードキャストすればよいことがわかる．

3.6 CS方式によるテストパターン数

CS方式による並列テスト生成では，N台のサーバ・プロセッサによってテストパターン生成と故障シミュレーションが行われる．クライアント・プロセッサは，各サーバがテストパターン生成の目標となるターゲット故障の決定と各サーバで処理された検出故障の集計を行う．

クライアントからサーバへの一回の通信で送られるターゲット故障数をmとする．サーバは，m個のターゲット故障を検出するテストパターンを生成するが，ある一個のターゲット故障に対して生成されたテストパターンが，他のターゲット故障も検出することがある．したがって，m個のターゲット故障に対して生成されるテストパターン数は，実際にはmより小さくなるものと思われる．そこで，m個のターゲット故障に対して生成されるテストパターン数をm^sとする．ただし，$0<s<1$とする．

サーバは，ターゲット故障を受け取ってからその故障の処理をすべて終えるまで通信を行わない．したがって，この間のテストパターン数と残存故障数との関係は，PB_{nb}方式と同様になる．また，m個のターゲット故障の処理を終えたあとはクライアントと通信を行い，他のプロセッサの処理の情報を受けるので，このときは，PB_{brd}方式と同様と考えられる．

CS方式では，各サーバがすべての故障を対象として故障シミュレーションを行うので，一回目の通信を行うまでのフェーズIでの残存故障数は，

$$F = Mq^t \qquad \left(0 \le t \le m^s\right) \tag{53}$$

となる．m^s本のテストパターンを生成した後，サーバはクライアントと通信を行う．このとき，もし各プロセッサへの負荷が均等かつ均質に分配され，同時にこの処理が進行しているとすれば，他の$N-1$台のサーバもまたm^s本のテストパターンを生成し故障を検出していると考えることができる．このように，一台のサーバがm^s本のテストパターンを生成したとき，全体ではm^sN本のテストパターンが生成されるが，これらのパターンは並列に生成されるため，3.4節で述べたように，一台のプロセッサで逐次的に生成されたテストパターンに比べて検出できる故障数は小さくなると思われる．ここではPB_{brd}方式での仮定と同様に，並列に生成されたm^sN本のテストパターンが検出できる故障数は，一台のプロセッサで逐次的に生成された$(m^sN)^r$本のテストパターンが検出できる故障数と等しいものとする．

この仮定をもとにサーバの処理の流れを言い換えれば，サーバはm個のターゲット故障に対する処理を終えクライアントと通信を行うことで，$(m^sN)^r$本のテストパターンを生成し，そのパターンを用いて故障シミュレーションを行ったことになるといえる．よって，サーバがm個のターゲット故障の処理を終えてからクライアントと通信を行うまでを一回と数えれば，k回の通信を行った後のフェーズIでの残存故障数は，

$$F = Mq^{k\left(m^sN\right)^r} \tag{54}$$

と表すことができる．したがって，式 (53), (54) より，サーバがt本のパターンを生成したときの残存故障数は，フェーズIの終了するテストパターン数をT_{Ics}とすると，

$$F = Mq^{\left(t\,\%\,m^s\right)+k\left(m^sN\right)^r} \quad \left(0 \le t \le T_{Ics}\right) \tag{55}$$

$$k = \frac{t-\left(t\,\%\,m^s\right)}{m^s} \tag{56}$$

ただし，%は，剰余を表す演算子とする．

k_I回目の通信で，ちょうどフェーズIが終了する，すなわち，

$$T_{Ics}\,\%\,m^s = 0 \tag{57}$$

$$k_I = \frac{T_{Ics}}{m^s} \tag{58}$$

となると仮定すれば，

$$\mu M = Mq^{k_I\left(m^sN\right)^r} \tag{59}$$

$$k_I\left(m^sN\right)^r = \frac{log\,\mu}{log\,q} \tag{60}$$

$$= T_I \tag{61}$$

となる．したがって，フェーズIの終了するテストパターン数T_{Ics}は，

$$T_{Ics} = k_I m^s \tag{62}$$

$$= \frac{m^{s(1-r)}}{N^r}T_I \tag{63}$$

となる．

次に，フェーズIIで生成されるテストパターン数について考える．

各サーバがテストパターン生成の対象とする故障は，ターゲット故障としてクライアントによって決定される．またフェーズIIでは，故障シミュレーションによる新たな故障検出はない．よって，テストパターン数を考える上では，フェーズIIにおけるテストパターン生成処理は一台のクライアントで行われているものと考えても問題はない．したがって，フェーズIIにおいて全体で生成されるテストパターン数をT_{IIcs}とすると，その残存故障数は，

$$F = \mu M - \alpha t_2 \qquad \left(0 \le t_2 \le T_{IIcs}\right) \tag{64}$$

であり，これは，シングルプロセッサシステムによるフェーズIIのものと等しいので，

$$T_{IIcs} = T_{\text{single}} - T_I \tag{65}$$

となる．したがって，CS方式による並列テスト生成で生成されるテストパターン数T_{CS}は，

$$T_{cs} = NT_{Ics} + T_{IIcs} \tag{66}$$

$$= T_{\text{single}} + \left(\left(m^s N\right)^{(1-r)} - 1\right) T_I \tag{67}$$

となる．このように，CS方式による並列テスト生成では，m個のターゲット故障を処理するごとに通信を行い全体の残存故障の情報を得るため，一本のテストパターンを生成するごとに通信を行うPB_{brd}方式（式(35)）に比べて，フェーズIで生成されるテストパターン数は$m^{s(1-r)}$倍増加する．

また，CS方式で生成されるテストパターン数が最小になるのは，$m = 1$のときであり，

$$T_{cs} = T_{\text{single}} + \left(N^{1-r} - 1\right) T_I \tag{68}$$

となり，式(35)と同様に表される．

4 CS方式とPB方式との比較

前章で述べたように，並列テスト生成で生成されるテストパターン数の変化は，故障シミュレーションの効果がないフェーズIIにおける処理には無関係で，故障シミュレーションの効果が得られるフェーズIにおける処理方法に依存する．本章では，フェーズIにおける処理方法に注目し，CS方式とPB方式とを比較することで，並列テスト生成の性質を考察する．

4.1 CS方式とPB方式との関係

CS方式による並列テスト生成において，フェーズIの処理で残存する故障数は式(55),(56)で示したように，

$$F = Mq^{\left(t \,\%\, m^s\right) + k\left(m^s N\right)^r} \qquad \left(0 \le t \le T_{Ics}\right) \tag{69}$$

$$k = \frac{t - \left(t \,\%\, m^s\right)}{m^s} \tag{70}$$

で表される．ここで，最初にすべての故障をターゲット故障として各サーバに割り当てる，すなわち，$m = M / N$とすると常に$t < m$なので，

$$t \,\%\, m^s = t \tag{71}$$

$$k = 0 \tag{72}$$

となる．よって式(69)は，

$$F = Mq^t \qquad \left(0 \le t \le T_{Ics}\right) \tag{73}$$

と書き換えることができる．したがって，フェーズIが終了するのに必要なテストパターン数T_{ICS}は，

$$\mu M = Mq^{T_{Ics}} \tag{74}$$

$$T_{Ics} = \frac{\log \mu}{\log q} \tag{75}$$

$$= T_{Inb} \tag{76}$$

となる．このように，初めにすべてのターゲット故障を割り当てるCS方式は，あらかじめ全故障を分割して各プロセッサに割り当て，通信を行わないPB_{nb}方式を表現している．

また，式(69),(70)において$m = 1$とすると，常に，

$$t \,\%\, m^s = 0 \tag{77}$$

$$k = t \tag{78}$$

となるので，式(69)は，

$$F = Mq^{tN^r} \qquad \left(0 \le t \le T_{Ics}\right) \tag{79}$$

と書き換えることができる．したがって，フェーズIが終了するのに必要なテストパターン数T_{Ics}は，

$$T_{Ics} = \frac{1}{N^r} \frac{\log \mu}{\log q} \tag{80}$$

$$= T_{Ibrd} \tag{81}$$

となる．このように，一回の通信で一個のターゲット故障を送信するCS方式は，一本のテストパターンを生成するごとにパターン・ブロードキャストを行うPB_{brd}方式を表している．

以上のように，CS方式による並列テスト生成で生成されるテストパターン数と残存故障数との関係を表す式は，PB_{nb}方式，PB_{brd}方式のテストパターン数と残存故障数との関係も表現していることがわかる．またこのことから，PB方式によるテスト生成として，m本のテストパターンを生成するごとに他のプロセッサへパターン・ブロードキャスト行う方式も考えることができる．この方式のフェーズIにおける残存故障数の関数は式(69),(70)と同様に，

$$F = \frac{M}{N} q^{(t \,\%\, m) + k(mN)^r} \qquad \left(0 \le t \le T_{Imb}\right) \tag{82}$$

$$k = \frac{t - (t \,\%\, m)}{m} \tag{83}$$

と書き表すことができる．ただしT_{Imb}は，この方式で並列テスト生成を行ったときにフェーズIで生成されるテストパターン数を表す．

4.2 テストパターン数と通信量

並列テスト生成では，並列度Nが高くなるにつれて生成されるテストパターンは増加する．テストパターンの増加を防ぐためには，通信回数を多くする，すなわちPB_{pc}方式では通信遮断係数p_cを大きくとる（最大で$p_c = 1 - \mu$），また，CS方式ではターゲット故障数mを小さくとる（最小で$m = 1$）ことが必要である．しかし，通信回数を多くすると通信のオーバーヘッドが増加し，並列処理の効果は下がってしまう．並列テスト生成では，処理時間と生成されるテストパターン数とはトレードオフの関係にあり，両方の目標を同時に達成することできない．ここでは，目標とするテストパターン数が与えられたときに必要な通信回数について考える．

フェーズIで生成されるテストパターン数のシングルプロセッサに対する増加率をω（ただし，$\omega > 1$）とする．すなわち，フェーズIで生成されるテストパターン数の目標をωT_Iとする．

PB_{pc}方式のフェーズIにおいて，一台のプロセッサで生成されるテストパターン数は，式(45)に示すように，

$$T_{Ipc} = T_I - \left(N^r - 1\right) T_P \tag{84}$$

で与えられる．一台あたりの目標テストパターン数

$$\omega\frac{T_I}{N}=T_I-\left(N^r-1\right)T_p \tag{85}$$

を達成するのに必要な通信時間K_{pc}は，

$$K_{pc}=T_p \tag{86}$$

$$=\frac{N-\omega}{N\left(N^r-1\right)}T_I \tag{87}$$

となる．

一方，CS方式のフェーズIにおける一台のサーバで生成されるテストパターン数の式(63)は，

$$T_{Ics}=\frac{T_I^{\frac{1}{r}}}{Nk_I^{\frac{1}{r}-1}} \tag{88}$$

のようにも表すことができるので，一台あたりの目標テストパターン数ωT_Iを達成する通信回数K_{cs}は，

$$K_{cs}=\omega^{-\frac{r}{1-r}}T_I \tag{89}$$

となる．

ここで，この２つの通信回数の差を考えると，

$$K_{\mathrm{diff}}=K_{pc}-K_{cs} \tag{90}$$

$$=\frac{T_I}{N\left(N^r-1\right)}\left(N-\omega-N\left(N^r-1\right)\omega^{-\frac{r}{1-r}}\right) \tag{91}$$

となる．これをωで偏微分して，

$$\frac{\partial}{\partial\omega}K_{\mathrm{diff}}=\frac{T_I}{N\left(N^r-1\right)}\left(\frac{r}{1-r}N\left(N^r-1\right)\omega^{-\frac{1}{1-r}}-1\right) \tag{92}$$

が得られる．生成されるテストパターンの最小は，$\mathrm{PB}_{\mathrm{pc}}$方式，CS方式，いずれも

$$\omega=N^{1-r} \tag{93}$$

のときであり，そのときの通信回数は，

$$K_{pc}=K_{cs}=\frac{T_I}{N^r} \tag{94}$$

なので，これを式(92)に代入して，

$$\frac{\partial}{\partial\omega}K_{\mathrm{diff}}=\frac{T_I}{N\left(N^r-1\right)}\left(\frac{r}{1-r}\left(N^r-1\right)-1\right) \tag{95}$$

が得られる．また式(94)より，

$$K_{\mathrm{diff}}=0 \tag{96}$$

がいえる．したがって，

$$N^r>\frac{1}{r} \tag{97}$$

が成り立てば，

$$\frac{\partial}{\partial\omega}K_{\mathrm{diff}}>0 \tag{98}$$

となるので，$\omega>N^{1-r}$のとき，

$$K_{\mathrm{diff}}>0 \tag{99}$$

となるωが存在することがわかる．この条件式(97)は，rが１に近いときに成り立ち，実際にもrは１に近い値をとると考えられる．よって，並列テスト生成において生成されるテストパターン数をできるだけ小さくしたいとき，CS方式は$\mathrm{PB}_{\mathrm{pc}}$方式に比べて，より少ない通信回数で目標とするテストパターン数を達成できるといえる．

また，以上の解析から，$\mathrm{PB}_{\mathrm{brd}}$方式，$\mathrm{PB}_{\mathrm{pc}}$方式で行われるパターン・ブロードキャストにおいても，一本のテストパターンを生成するごとにブロードキャストせずに，複数のテストパターンをまとめてブロードキャストすることで，より効率の良い処理を達成することができると考えられる．

5 まとめ

本稿では，並列テスト生成において生成されるテストパターン数について考察した．シングルプロセッサシステムで生成されるテストパターンの数とそのパターンによって検出される故障数との関係をもとに，マルチプロセッサシステムで生成されるテストパターンの数とそのパターンによって検出される故障数との関係を導き，並列テスト生成で得られるテストパターン数について解析した．並列テスト生成で生成されるテストパターン数は，シングルプロセッサシステムで得られるテストパターン数に比べて大きくなることを示し，処理方式の違いによるテストパターン数の変化を示した．

さらに，マルチプロセッサシステムで行われる通信の，生成されるテストパターン数への影響を解析し，並列テスト生成によって発生するテストパターン数の増加を抑える方法について述べた．

参考文献

[1] 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.

[2] S.Patil and P.Banerjee, "Performance trade-offs in a parallel test generation/fault simulation environment," *IEEE Trans. Computer-Aided Design*, Vol.10, No.12, pp.1542-1558, Dec. 1991.

[3] P. Goel, "Test generation cost analysis and projections," *Proc. 17th Ann. Design Automation Conf.*, pp.77-84, 1980.

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

[5] 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.

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

[7] F.Hirose, K.Takayama, and N.Kawato,"A method to generate tests for combinational logic circuits using an ultra high speed logic simulator," *Proc. 1988 Int'l Test Conf.*, pp.102-107, 1988.

[8] S.Patil and P.Banerjee, "A parallel branch-and-bound algorithm for test generation," *IEEE Trans. Computer-Aided Design*, Vol.9, No.3, pp.313-322, March 1990.

[9] S.Patil, P.Banerjee and J.H.Patel,"Parallel test generation for sequential circuits on general-purpose multiprocessors," *Proc. 28th ACM/IEEE Design Automation Conf.*, pp.155-159, 1991.