設計自動化49－13
(1989.10.25)

テスト生成における並列処理の最適スケジューリング

Optimal Schedule in Parallel Processing for Test Pattern Generation

井上 智生　　藤原 秀雄
Tomoo INOUE and Hideo FUJIWARA
明治大学 理工学部
Department of Computer Science, Meiji University

あらまし　論理回路のテスト生成問題は，ＮＰ困難な問題として知られており，その処理にはバックトラック手法を用いるため，高速化は非常に困難である。本稿では，高速化の一方法として，汎用コンピュータの疎結合分散型ネットワークを用いたテスト生成並列処理を提案する。そして，各プロセッサへの部分問題の割り当ての効果，部分問題の粒度，シングル・プロセッサ・システムに対するマルチ・プロセッサ・システムのスピードアップ率を解析することにより，その性能を評価する。

Abstract　The problem of test generation for logic circuits is known to be NP-hard, and hence it is very hard to speed up the test generation process due to its backtracking mechanism. This paper presents an approach to parallel processing of test generation for logic circuits in a loosely-coupled distributed network of general purpose computers, and analyze the effects of the allocation of target faults to processors, the optimal granularity (grain size of target faults) and the speedup ratio of the multiple-processor system to a single processor system.

１．はじめに

一般にテスト生成処理には，テスト・パターン生成と故障シミュレーションとが合わせて用いられる。効率の良いテスト・パターン生成アルゴリズムとして，ＰＯＤＥＭ［１］，ＦＡＮ［２］，ＳＯＣＲＡＴＥＳ［３］などが報告されている。しかし，テスト・パターン生成問題はＮＰ困難であるために，一般にその計算量は回路の大規模化に伴って指数関数的に増大する。また故障シミュレーションについては演繹または同時故障シミュレーションが有効ではあるが，これらのシミュレーションの計算複雑度は$O(G^2)$～$O(G^3)$と考えられる。従って，１台の汎用コンピュータ上での大規模な回路に対するテスト生成処理には限界がある。

ＶＬＳＩ回路に対するＣＡＤシステムについても，その高速性について同様に問題があるが，それを軽減する方法として，ＩＢＭのYorktown Simulation Engine（ＹＳＥ）［９］，ＮＥＣのHardware Accelerator（ＨＡＬ）［１０］，Zycad，Silicon Solutions，Daisy，etc.［１１，１２］などの専用ハードウェア・アクセラレータの適用が報告されている。これらをテスト生成処理に応用する試みもある。

Kramerは，テストパターン生成にＭＩＴ　Connection Machineを用いたが［１４，１５］，その高速化の有効性は，そのアルゴリズムの指数関数的な性質のために，小さな回路に対してしか得られていない。また，Elziqら［１６］は，論理検証と故障シミュレーションについて報告しており，その中で，専用エンジンを用いたエキスパート・システムまたは知識ベース・システムによるテスト生成システムを提案しているが，その具体的なテスト生成エンジンは示されていない。

専用ハードウェア・エンジンは，非常に高速のシミュレーションを供給するものであるが，そのハードウェアのコストは非常に高価でまた汎用性は低い。それに代わる比較的安価で汎用性の高いものとして，汎用コンピュータの疎結合分散型ネットワークが考えられる。汎用コンピュータの疎結合ネットワークを用いた分散型故障シミュレータが報告され，線形に近い高速性が得られている［１７］。テスト生成については，多重の発見的手法を用いて並列に処理する分散型システムについての提案も発表されている。

本原ら［１８］と藤原［１９］は，密結合マルチ・プロセッサ・システムＬＩＮＫＳ-１を用いたテスト生成の並列処理で，１）故障並列：異なる故障を同時に処理，２）探索並列：一故障に対する（分岐限定における）決定木の異なる節を同時に処理，の２つの並列性を示し，その効果を報告している。

本稿では，汎用コンピュータの疎結合分散型ネットワークで，［１８，１９］の故障並列を拡張したテスト生成並列処理を考察する。［１８，１９］では考えられていなかった最適粒度（１回の処理にプロセッサへ割り当てられる故障数）について解析し，「静的タスク割り当て」と「動的タスク割り当て」における最適粒度を求める。また，シングル・プロセッサ・システムに対するマルチ・プロセッサ・システムのスピードアップ率を導き，その性能を評価する。

２．分散型システムの構造

本稿で取り扱う疎結合のマルチ・プロセッサ・システムの構造を図１に示す。クライアントとサーバは通信用バスによって結合されている。クライアントはサーバにタスクの実行を要求する。サーバはその割り当てられたタスクの処理し，それを終えるとその結果をクライアントに返送し，次の新しいタスクを要求する。クライアントはその結果を保存し，新しいタスクをそのサーバに用意する。この分散処理では，全体問題はいくつかの部分問題すなわちタスクに分割され，それぞれのタスクはサーバに割り当てられる。

本稿における問題はテスト生成である。よって目標は全故障に対するテスト・パターンの生成であり，問題の領域は故障の集合となる。故障は各サーバに分配され，それらに対するテスト・パターンが生成される。この故障をターゲット故障と呼び，この故障群の大きさすなわちターゲット故障数を粒度と呼ぶ。従って，サーバに割り当てられる部分問題すなわちタスクは故障群に対するテスト生成問題である。与えられた故障群に対して各サーバはテスト・パターン生成と故障シミュレーションを行う。その結果は，生成されたテスト・パターンによって検出可能な故障の集合，冗長故障の集合，そしてバックトラック数の制限を越えたために打ち切られた故障の集合である。この情報はクライアントに送られ，そこで故障表が更新される。そしてクライアントは，未処理の故障群を取り出しサーバに送る。サーバはその新しい故障群に対するテスト生成を行う。この処理は故障表中の全故障が処理されるまで続けられる。

並列度を上げるために計算粒度を小さくすれば，サーバはより短時間に処理を終えるが，そのために，クライアントへの要求回数が増すようになる。よって通信のオーバーヘッドが増加し，逆に処理の効率は低下する。この性質は様々な要因が複雑に関係し合っているため，与えられた問題およびシステムに対して，最高の性質を引き出すような最適粒度を予測することは非常に困難である。

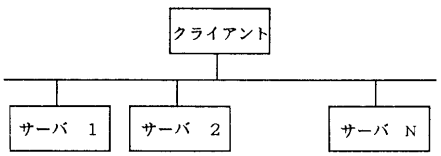

図１．分散型システムの構造

３．問題の形式化

本稿では，１台のクライアントとN台のサーバから成る分散型ネットワークを考える（図１）。

与えられた回路の全故障数をMとする。故障 f_i に対するテスト・パターン生成処理を「故障 f_i に対する処理」と呼ぶ。１つの故障に対する処理の結果は，１）その故障は生成されたテスト・パターンによって検出される，２）その故障は冗長である，３）その処理はバックトラックの制限を越えたため打ち切られた，のいずれかとなる。サーバ j での故障 f_i に対する処理時間，すなわちサーバ j が故障 f_i に対するテスト生成処理を完了するのに必要な計算時間を τ_{ij} とする。故障 f_i に対する処理がサーバ j に割り当てられる確率を δ_{ij} とする。故障 f_i に対する処理の後，サーバ j がクライアントと通信する確率を λ_{ij} とする。クライアント・サーバ間のデータ転送に要する，競合による待ち時間を含めた平均通信時間を τ_c とする。

以上の定義より，サーバ j が割り当てられた全処理を完了するのに必要な平均時間は，

$$T_j = \sum_{i=1}^{M} \delta_{ij} \left(\tau_{ij} + \lambda_{ij} \tau_c \right) \qquad (1)$$

システム全体の処理が完了するのに必要な平均時間は T_j の最大値によって定義され，

$$T = \max \{ T_j \} \qquad (2)$$

この様にして，Tを最小にするタスク割り当てのスケジュールを求めることが問題となる。

４．均質問題に対する静的タスク割り当て

４．１最適粒度

まず初めに，クライアントからサーバへの１回の通信で送られるターゲット故障数を一定にする「静的タスク割り当て」を考える。

最適なタスク割り当てのスケジュールを得るには，各サーバへの負荷が均等に分配されること，例えば，困難故障も容易故障と同様に均等に分配することが重要である。しかし，テスト生成処理に先立って故障が困難か容易かを正確に知ることは困難である。（一つの方法として，可制御性／可観測性尺度［２５］を用いたテスト容易性の解析による評価がある。）ここでは以下に示す様な仮定によって，すべての故障が均等に分配されるものとする。

（１）すべてのサーバの処理性能は等しい。（$\tau_{ij} = \tau_i$）

（２）故障 f_i がサーバ j に割り当てられる確率は j に無関係でどの故障に対しても等しい。（$\delta_{ij} = \delta_i$）

これらの仮定をおいた問題を「均質な」問題と呼ぶ。

クライアントからサーバへの１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 j はm個のターゲット故障に対する処理をすべて完了した後にクライアントと通信する。従って，故障 f_i に対する処理の後，サーバ j がクライアントと通信する確率は $\lambda_{ij} = 1/m$ となる。

各故障する処理時間の平均を τ とする。すなわち，

$$\tau = \frac{\sum_{i=1}^{M} \tau_i}{M} \qquad (3)$$

I　故障シミュレーションを適用しない場合

まず初めに，全故障に対してそれぞれテスト・パターン生成を行い，故障シミュレーションを実行しない場合を考える。よって故障 f_i がサーバ j に割り当てられる確率は，$\delta_{ij} = 1/N$ となる。以上の式を式（１），（２）に代入して，次式が得られる。

$$T_j = \sum_{i=1}^{M} \delta_{ij}(\tau_{ij} + \lambda_{ij}\tau_c)$$
$$= \sum_{i=1}^{M} \frac{1}{N}\left(\tau_i + \frac{1}{m}\tau_c\right)$$
$$= \frac{M}{N}\left(\tau + \frac{1}{m}\tau_c\right) \quad (4)$$

この式はmの単調減少関数であり，$1 \leqq m \leqq M/N$である。よって，Tの最小値は$m = M/N$のときに得られる。

$$T_{min} = \frac{M}{N}\tau + \tau_c \quad (5)$$

均質問題において，全故障のそれぞれに対してテスト・パターンを生成し，故障シミュレーションを適用しない場合，各サーバがクライアントから１回で全ターゲット故障を受け取るとき最高の性能が得られることがわかる。しかし，ある故障に対して生成されたテスト・パターンは他の故障を検出可能なことが多い。従って，テスト・パターン生成処理の後故障シミュレーション処理を適用すれば，全故障のそれぞれに対してテスト・パターン生成処理を行う必要はない。

II　故障シミュレーションを適用する場合

次に，テスト・パターン生成の後，故障シミュレーションを適用する場合を考える。クライアントは故障表からm個の故障を取り出し，ターゲット故障としてサーバに送る。そのサーバは，m個の故障のうちの１個に対するテスト・パターンを生成し，そのテスト・パターンを用いて，m個の故障集合だけでなく，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。この処理はターゲット故障がすべて処理されるまで繰り返される。m個のターゲット故障に対する処理によって，新たにρm個の故障が検出可能または冗長という結果が得られると仮定する。「新たに」とは，今まで検出可能か冗長かわからなかった故障が新たにそれがわかることを言う。この故障を「新たに処理される故障」と呼ぶ。

ターゲット故障数に対する新たに処理される故障の比を次のように定義する。

$$\rho = \frac{\text{サーバあたりの新たに処理される故障数}}{\text{サーバあたりのターゲット故障数}} \quad (6)$$

このρm個の新たに処理される故障のうち，いくつかはそれ自身に対するテストパターン生成処理によって検出可能か冗長かわかる故障であり，その他は故障シュミレーション処理によって検出可能とわかる故障である。よって，故障 f_i がいずれかのサーバで処理される確率は，

$$\frac{m}{\rho_i m} = \frac{1}{\rho_i} \quad (7)$$

すなわち，

$$\sum_{j=1}^{N} \delta_{ij} = \frac{1}{\rho_i} \quad (8)$$

一方，仮定$\delta_{ij} = \delta_i$より，

$$\sum_{j=1}^{N} \delta_{ij} = \sum_{j=1}^{N} \delta_i = N\delta_i \quad (9)$$

よって式（８），（９）より，

$$\delta_{ij} = \delta_i = \frac{1}{N\rho_i} \quad (10)$$

サーバ j に割り当てられたすべての処理が完了するのに必要な平均時間は，式（１）に式（１０）及び$\tau_{ij} = \tau_i$，$\lambda_{ij} = 1/m$より得られる。

$$T_j = \sum_{i=1}^{M} \frac{1}{N\rho_i}\left(\tau_i + \frac{\tau_c}{m}\right) \quad (11)$$

式（１１）の右辺は j に無関係である。すなわち，平均処理時間は，どのサーバにおいても等しくなる。これは，サーバはすべて同性能で，かつどの故障も等確率で各サーバに割り当てられるという均質問題を仮定したことによる。よって，通信時間を含めた全処理時間 T は，

$$T = \sum_{i=1}^{M} \frac{1}{N\rho_i}\left(\tau_i + \frac{\tau_c}{m}\right) \quad (12)$$

通信時間：τ_c

次の２つを仮定する。(1)クライアント・サーバ間で転送されるデータの大きさは一定，よってそれに要する時間は一定。(2)サーバからクライアントへの通信要求の起こる総数はサーバ数Nに比例，よってクライアント・サーバ間での通信待ち時間はNに比例。以上より次式が得られる。

$$\tau_c = t_0 + t_1 \quad (13)$$

（ただし，t_0，t_1は定数）

新たに処理される故障の比：ρ

図２は，新たに処理される故障数の変化の様子を調べるために，ISCAS'85ベンチマーク回路C7552［２４］について実験した結果を示したものである。テスト・パターン生成の後，故障シミュレーションを行うことを一処理として横軸に処理数，縦軸に新たに処理された故障数を表している。この結果から，新たに処理される故障数はその処理数が増加するにつれて急激に減少するものと思われる。そこ

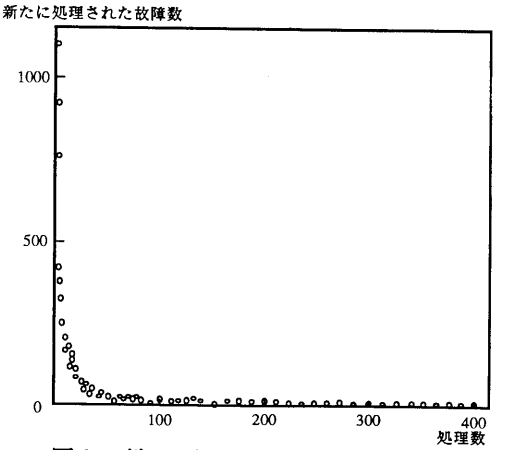

図２．新たに処理される故障数 (C7552)

でターゲット故障数に対する新たに処理される故障数の比ρ(x)を次式の様に仮定する。

$$\rho(x) = \frac{1}{r_0 + r_1 x} \tag{14}$$

（ただし，xは処理された故障数，r_0，r_1 は定数）

クライアントはサーバからの結果を受け取ると故障表を更新する。この時多数のサーバが同時に働いてるために，異なるサーバ間で同じ故障を処理することがあり得る。このオーバーラップした処理は無駄と考えられる。この数はターゲット故障数あるいはサーバ数の増加に伴って大きくなり，従って新たに処理される故障の比は小さくなると思われる。よって次式の様に仮定する。

$$\rho(x) = \frac{1}{r_0 + r_1 x + r_2 mN} \tag{15}$$

（ただし，r_2は定数）

全体から見て，i 番目に処理される故障を$f_{\pi(i)}$とする。ただし，π は故障処理数から故障番号への順列である（π : $\mathcal{M} \rightarrow \mathcal{M}$，$\mathcal{M} = \{1, 2, 3, \ldots, M\}$）。よって上式（１５）は故障$f_{\pi(i)}$が処理されるときの新たに処理される故障の比として，次の様に書き換えることができる。

$$\rho_{\pi(i)} = \frac{1}{r_0 + r_1 i + r_2 mN} \tag{16}$$

順列$\mathcal{M}$の集合を$\mathcal{P}$とする。このとき，順列$\mathcal{M}$と故障処理の順序は一対一に対応しており，その数はM!通りである。

ここで，すべての順列に対する全処理時間の平均を考えると，式（１２）と式（１６）より，

$$T = \frac{\sum_{\pi \in \mathcal{P}} \sum_{i=1}^{M} \frac{1}{N}(r_0 + r_1 i + r_2 mN)\left(\tau_{\pi(i)} + \frac{\tau_c}{m}\right)}{M!} \tag{17}$$

一方，

$$\sum_{\pi \in \mathcal{P}} \sum_{i=1}^{M} i\tau_{\pi(i)} = \sum_{i=1}^{M} i\left((M-1)! \sum_{i=1}^{M} \tau_i\right) = \sum_{i=1}^{M} i\,(M!\,\tau) \tag{18}$$

従って式（１７），式（１８）より，

$$T = \sum_{i=1}^{M} \frac{1}{N}(r_0 + r_1 i + r_2 mN)\left(\tau + \frac{\tau_c}{m}\right) \tag{19}$$

$$= \frac{M}{N}\left(r_0 + r_1 \frac{M+1}{2} + r_2 mN\right)\left(\tau + \frac{\tau_c}{m}\right) \tag{20}$$

Tをmで偏微分して，

$$\frac{dT}{dm} = \frac{M}{N}\left(r_2 N\tau - \frac{\left(r_0 + r_1 \frac{M+1}{2}\right)\tau_c}{m^2}\right) \tag{21}$$

よってTの最小値は，

$$T_{min} = \frac{M}{N}\left(\sqrt{\left(r_0 + r_1 \frac{M+1}{2}\right)\tau} + \sqrt{r_2 N \tau_c}\right)^2 \tag{22}$$

それを得られるターゲット故障数は，

$$m = \sqrt{\frac{\left(r_0 + r_1 \frac{M+1}{2}\right)\tau_c}{r_2 N \tau}} \tag{23}$$

図３に，通信時間を含めた全処理時間をサーバあたりのターゲット故障数の関数として示す。曲線（ａ）と曲線（ｂ）より，通信時間が小さくなると最適粒度，全処理時間の両方が小さくなることがわかる。曲線（ａ）と曲線（ｃ）より，サーバ間の処理のオーバーラップが小さくなると，全処理時間は小さくなり，最適粒度は大きくなることがわかる。

図４に，ISCAS'85ベンチマーク回路C7552［２４］に対する実験結果を示す。実験には，イーサネットで結合され

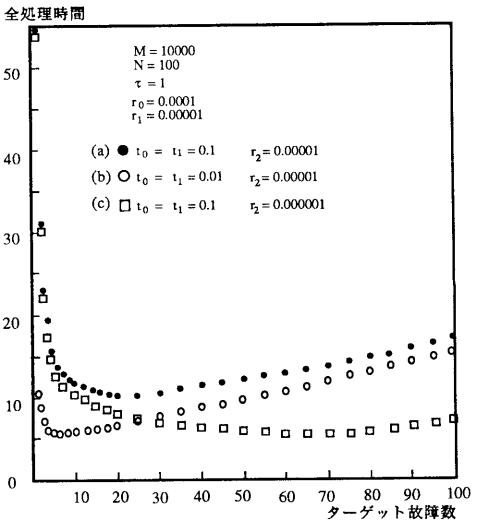

図３．全処理時間

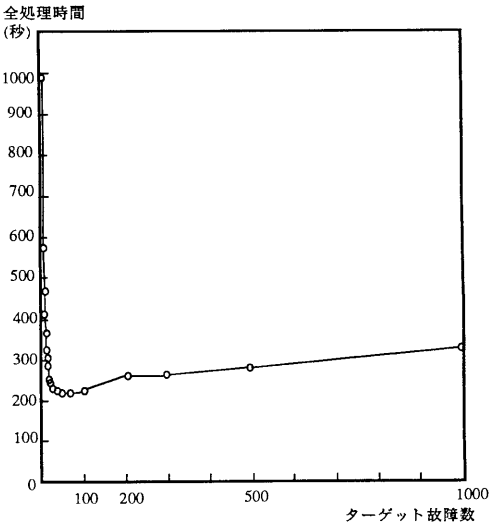

図４．実験結果(サーバ数3,C7552)

た4台のSUN3/50（クライアント1台，サーバ3台）にFANアルゴリズム［2］を用いて行った。この図は均質問題について解析した上述の図3とかなり一致していると言える。

4.2 マルチ・プロセッサ・システムのスピードアップ率

マルチ・プロセッサ・システムのスピードアップ率をN台のマルチ・プロセッサ・システム上での全処理時間に対するシングル・プロセッサ・システム上での全処理時間の比で定義する。すなわち，

$$S = \frac{T_{single}}{T_{multi}} \tag{24}$$

シングル・プロセッサ・システム上での処理時間は，

$$T_{single} = \sum_{i=1}^{M}(r_0 + r_1 i)\tau = M\left(r_0 + r_1\frac{M+1}{2}\right)\tau \tag{25}$$

式（22），（24），（25）より，マルチ・プロセッサ・システムの最大スピードアップ率S_{max}は，

$$S_{max} = \frac{T_{single}}{T_{min}} = \frac{N}{\left(1+\sqrt{\left(\frac{r_2 N}{r_0 + r_1\frac{M+1}{2}}\right)\left(\frac{t_0+t_1 N}{\tau}\right)}\right)^2} < N \tag{26}$$

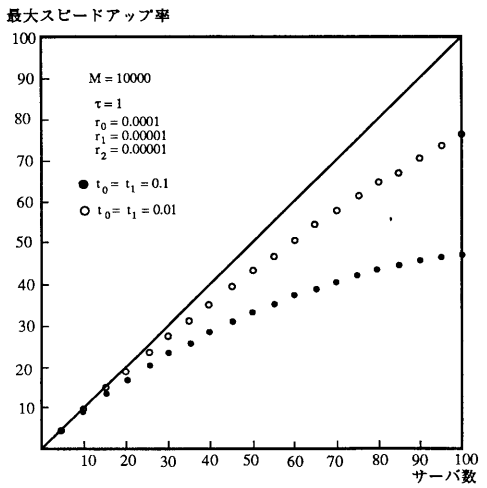

図5．最大スピードアップ率

従って，次の2式

$$r_2 mN << r_0 + r_1\frac{M+1}{2} \tag{27}$$

$$\frac{t_0+t_1 N}{m} << \tau \tag{28}$$

が成り立つとき，すなわちオーバーラップ処理のために起こる新たに処理される故障の比の減少が非常に小さく，かつ故障あたりのデータ転送時間とその待ち時間が故障の処理時間に比べて非常に小さいとき，S_{max}はNに近づく。

図5に，最大スピードアップ率をサーバ数の関数として，通信時間の異なる2つの場合について示した。

5．均質問題に対する動的タスク割り当て

この章では，クライアントからサーバへの1回の通信で送られるターゲット故障の数を，時刻の変化にともなって変化させる「動的タスク割り当て」について考える。

前章と同様に，均質問題を仮定する。すなわち，すべての故障f_i，サーバjに対して，$\tau_{ij} = \tau_i$，$\delta_{ij} = \delta_i$と置く。

動的タスク割り当てにおける全処理時間は，静的タスク割り当てにおけるターゲット故障数 m を m_iに置き換えることによって得られる。すなわち，

$$T = \sum_{i=1}^{M}\frac{1}{N\rho_i}\left(\tau + \frac{\tau_c}{m_i}\right) \tag{29}$$

よって式（17）より，

$$T = \frac{\sum_{\pi\in P}\sum_{i=1}^{M}\frac{1}{N}\left(r_0 + r_1 i + r_2 m_{\pi(i)} N\right)\left(\tau_{\pi(i)} + \frac{\tau_c}{m_{\pi(i)}}\right)}{M!} \tag{30}$$

$$= \sum_{i=1}^{M}\frac{1}{N}\left(r_0 + r_1 i + r_2 m_i N\right)\left(\tau + \frac{\tau_c}{m_i}\right) \tag{31}$$

上式を m_iで偏微分して，

$$\frac{dT}{dm_i} = \frac{M}{N}\left(r_2 N\tau - \frac{(r_0 + r_1 i)\tau_c}{m_i^2}\right) \tag{32}$$

よって動的タスク割り当てにおけるTの最小値は，

$$T_{dynamic} = \sum_{i=1}^{M}\frac{1}{N}\left(\sqrt{(r_0 + r_1 i)\tau} + \sqrt{r_2 N\tau_c}\right)^2 \tag{33}$$

これを得る最適なターゲット故障数は，すべての i について，

$$m_i = \sqrt{\frac{(r_0 + r_1 i)\tau_c}{r_2 N\tau}} \tag{34}$$

上式より時刻 t の最適粒度は，

$$m(t) = \sqrt{\frac{(r_0 + r_1 x_t)\tau_c}{r_2 N\tau}} \tag{35}$$

ただし x_tは，時刻 t までに全サーバによって処理された全故障を表す。すなわち，クライアントは現在時刻 t までに処理された全故障数 x_tを数え上げ，式（35）からターゲット故障数m(t)を計算し，そのm(t)個のターゲット故障を待機中のサーバに割り当てることによって，最高の性能を得ることができる。ここでは式（35）は連続関数を表しているが，m(t)は整数と定義される。

次に，静的タスク割り当てに対する動的タスク割り当ての効果について考察する。
静的タスク割り当てにおける最小値T_{static}は式（２２）より，

$$T_{static}=\frac{M}{N}\left(\sqrt{\left(r_0+r_1\frac{M+1}{2}\right)\tau}+\sqrt{r_2N\tau_c}\right)^2 \qquad (36)$$

よって，T_{static}と$T_{dynamic}$との差は，

$$T_{static}-T_{dynamic}=\frac{2\sqrt{r_2N\tau_c\tau}}{N}\sum_{i=1}^{M}\left(\sqrt{\left(r_0+r_1\frac{M+1}{2}\right)\tau}-\sqrt{r_0+r_1 i}\right)$$

$$>0 \qquad (37)$$

この式はM ＞ 1のとき常に成立する。よって動的タスク割り当ては静的タスク割り当てに比べて常に効果的と言える。

６．まとめ

本稿では，汎用コンピュータの疎結合分散型ネットワークを用いた論理回路のテスト生成並列処理を提案した。そして，各プロセッサへのターゲット故障の割り当ての効果，最適粒度（ターゲット故障数），シングル・プロセッサ・システムに対するマルチ・プロセッサ・システムのスピードアップ率を解析した。

すべてのサーバ・プロセッサが同性能で，どの故障も各サーバへ割り当てられる確率が均等である均質問題において，もし故障シミュレーションを行わずにそれぞれの故障に対してテスト・パターンを生成するならば，通信を含めた全処理時間 T はターゲット故障数 m の単調減少関数になる。この場合，各サーバがクライアントから一度にターゲット故障を受け取るとき，すなわちm ＝ M／Nのとき，最高の性能が得られる。しかし，ある故障に対して生成されたテスト・パターンを用いて故障シミュレーションを行えば，すべての故障に対してテスト・パターン生成を行う必要はない。その場合について解析するため，ターゲット故障に対する新たに処理される故障の比を導き，動的，静的の両方のタスク割り当てにおける最適粒度の式を導いた。また動的タスク割り当てについては，静的タスク割り当てに比べて効果的であることも示した。

さらに，シングル・プロセッサ・システムに対するマルチ・プロセッサ・システムの性能を評価するために，そのスピードアップ率の式を導き，クライアント・サーバ間の通信時間とサーバ間でのオーバーラップ処理数が十分小さいとき，スピードアップ率はNに近づくことを示した。

従って，ハードウェアから成る係数が前もって与えられた場合，比ρの中のオーバーラップ処理を示す項が最小になるようにタスク割り当てを行えば，マルチ・プロセッサ・システムにおける最高の性能を引き出すことが出来る。

参考文献

[1] P.Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," IEEE Trans. Comput., Vol. C-30, No.3, pp.215-222, March 1981.

[2] H.Fujiwara and T. Shimono, "On the acceleration of test pattern generation algorithms," IEEE Trans. Comput., Vol. C-32, No.12, pp.1137-1144, Dec. 1983.

[3] M.H.Schulz and E.Auth,"Advanced automatic test pattern generation and redundancy identification techniques," Dig. of Papers, FTCS-18, pp.30-35, June 1988.

[4] O.H.Ibarra and S.K.Sahni, "Polynomially complete fault detection problems," IEEE Trans. Comput., Vol. C-24, No.3, pp.242-249, March 1975.

[5] H.Fujiwara and S.Toida, "The complexity of fault detection problems for combinational logic circuits," IEEE Trans. Comput., Vol. C-31, No.6, pp.555-560, June 1982.

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

[7] T.W.Williams and K.P.Parker, "Design for testability - A survey," Proc. IEEE, Vol.71, pp.98-112, Jan. 1983.

[8] W.A.Rogers, J.F.Guzolek, and J.A.Abraham, "Concurrent hierarchical fault simulation: A performance model and two optimizations," IEEE Trans. Computer-Aided Design, Vol. CAD-6, No.9, pp.848-862, Sept. 1987.

[9] G.Pfister, "The Yorktwon simulation engine: Intorduction," Proc. Ann. 19th Design Automation Conf., pp.51-54, 1982.

[10] T.Sasaki, N.Koike, K.Ohmori, and K.Tomita, "HAL: A block level hardware logic simulator," Proc. Ann 20th Design Automation Conf., pp.150-156, 1983.

[11] T.Blank, "A survey of hardware accelerators used in computer-aided design," IEEE Design and Test, pp.21-39, Aug. 1984.

[12] B.Milne, "Put the pedal to the metal with simulation accelerators," Electronic Design, pp.39-52, Sept. 1987.

[13] L.T.Smith and R.R.Rezac, "Methodology for and results from the use of hardware logic simulation engine for fault simulation," Proc. Int. Test Conf., pp.224-228, 1984.

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

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

[16] Y.M.Elziq, H.H.Butt and A.K.Bhatt, "An automatic test pattern generation machine," Proc. IEEE Int. Conf.on Computer-Aided Design, pp.257-259, 1984.

[17] P.A.Duba, R.K.Roy, J.A.Abraham, and W.A.Rogers, "Fault simulation in a distributed environment," Proc. 25th Design Automation Conf., pp.686-691, 1988.

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

[19] H.Fujiwara and A.Motohara, "Fast test pattern generation using a multi-processor system," Trans. of IEICE, Vol. E-71, No.4, pp.441-447, April 1988.

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

[21] 下条,宮原,高島, "通信競合を含めたマルチプロセッサにおけるプロセス割り当て問題," 電子情報通信学会論文誌, Vol. J68-D, No.5, pp.1049-1056, 1985年5月.

[22] Y-T. Wang and R.J.T. Morris, "Load Sharing in Distributed Systems," IEEE Trans. Comput., Vol. C-34, No.3, pp.204-217, March 1985.

[23] Z. Cvetanovic, "The effects of problem partitioning, allocation, and granularity on the performance of multiple-processor systems," IEEE Trans. Comput., Vol. C-36, No.4, pp.421-432, April 1987.

[24] F. Brglez and H. Fujiwara, "A neutral netlist of ten combinational benchmark circuits and a target translator in FORTRAN," Proc. IEEE Int. Symp. on Circuits and Systems, Kyoto, Japan, June 5-7 1985.

[25] L. H. Goldstein, "Controllability/observability analysis of digital circuits," IEEE Trans. Circuits and Systems, Vol. CAS-26, No. 9, pp. 685-693, Sept. 1979.

盛光印刷所