設計自動化　64－3
(1992.10.22)

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

井上 智生　　米澤 友紀　　藤原 秀雄

明治大学　理工学部
神奈川県川崎市多摩区東三田１－１－１

あらまし　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テスト生成の並列処理を提案する．分散型システムにおけるプロセッサ数と処理時間の関係について考察し，最小の処理時間を得るための最適なプロセッサ数について解析する．さらに，ISCASベンチマーク回路を用いた，ワークステーションから成るネットワーク上での実験結果を示す。

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，最適プロセッサ数

An Optimal Scheme of Parallel Processing for Test Generation in a Distributed System

Tomoo INOUE　　*Tomonori* YONEZAWA　　*Hideo* FUJIWARA

Department of Computer Science
Meiji University
1-1-1 Higashimita Tama-ku
Kawasaki-shi Kanagawa-ken

Abstract　In this paper, we present an approach to parallel processing of test generation for logic circuits in a loosely-coupled distributed network of general purpose computers. We consider the relation between the total processing time and the number of processors, and analyze the optimal number of the processors which minimizes the total processing time in the system. Further, we show experimental results based on an implementation on a network of SUN workstations using the ISCAS benchmark circuits.

英文 key words　Test generation, Parallel processing, Multiple processor, Fault parallel, Optimal number of processors

1 はじめに

論理回路のテスト生成問題は，組み合わせ回路でさえNP困難であるために[1][2]，その処理には多くの時間を要する．テスト生成のアルゴリズムとして，PODEM[3]，FAN[4]，SOCRATES[5]などの効率の良いものが報告されているが，今日の大規模化された論理回路に対しては，その効果に限界がある．

より高速に処理を行う方法の一つとして，マルチプロセッサシステムを用いた並列分散処理がある．並列処理では，与えられた問題をどのように分割して各プロセッサに割り当てるかによって，その処理方法をいくつかに分類することができる．

テスト生成の並列処理についてもいくつかの方法が報告されている[6]-[13]．[11]では，一つの故障に対するテストパターンを生成するときに作られる決定木を分割して各プロセッサへ割り当てる**探索並列法**について，[6],[9]では，与えられた回路を部分回路に分割して各プロセッサへ割り当て，テスト生成を行う**回路並列法**について報告されている．また，[8]では，異なる発見的手法をそれぞれのプロセッサで用いながら，一つの故障についてテスト生成を行う**方策並列法**も報告されている．筆者らは[10]において，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，故障集合を分割して各プロセッサへ割り当てる**故障並列法**によるテスト生成の並列処理を提案し，最適粒度（一回の通信でプロセッサに割り当てられる部分問題の大きさ）とスピードアップ率について解析を行っている．

本稿では，[10]で示したテスト生成の並列処理システムにおけるプロセッサ数と総処理時間の関係について考察する．[10]で提案されている分散型システムにおける最適なプロセッサ数について解析し，より多くのプロセッサを用いてより高速の処理を可能にするシステムの構成方法について解析する．また，これらの解析を実験によって確認する．

2 クライアント・サーバモデルにおけるテスト生成処理

2.1 CSモデルの構成

クライアント・サーバモデル（Client-Serverモデル，以下，CSモデルと呼ぶ）の構成を図2.1に示す．一台のクライアント・プロセッサとN台のサーバ・プロセッサは，一本の疎結合分散型ネットワークによって接続されている．それぞれのプロセッサは汎用コンピュータから成る．

ここで取り扱う問題はテスト生成である．したがって目標は与えられた回路に定義された故障に対するテストパターン生成であり，問題の領域はその故障集合となる．CSモデルにおけるテスト生成処理の流れは次のようになる．

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，故障表とともにサーバへ転送する．この故障を**ターゲット故障**と呼ぶ．

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．

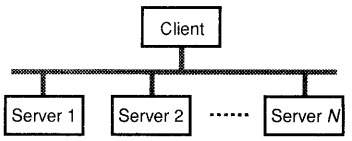

図2.1 CSモデルの構成

2.2 CSモデルにおける問題の定式化

ここではCSモデルにおけるテスト生成問題を定式化する．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．故障f_iに対する処理の結果は，(1)故障f_iは生成されたテストパターンによって検出される，(2)故障f_iは冗長である，(3)故障f_iに対する処理は，バックトラックの制限を越えたために打ち切られた，のうちのいずれかとなる．

サーバS_jにおける故障f_iに対する処理時間，言い換えれば，サーバS_jが故障f_iに対するテストパターン生成処理を完了するのに要する時間をτ_{ij}とする．故障f_iに対する処理がサーバS_jに割り当てられる確率をδ_{ij}とする．故障f_iに対する処理が終了した後，サーバS_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時間と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．

これらの定義により，サーバS_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，

$$T_j = \sum_{i=1}^{M} \delta_{ij}(\tau_{ij} + \lambda_{ij}\tau_c) \tag{2.1}$$

すべての故障に対する処理が完了するのに必要な平均処理時間は，サーバS_jの処理時間の最大値

$$T = max\ \{T_j\} \tag{2.2}$$

で表される．

均質問題の仮定

CSモデルにおいて最小の処理時間を得るためには，各サーバへの負荷が均等になることが重要であるが，それぞれの故障に対するテスト生成処理の困難さを前もって知ることは難しい．ここでは，[10]と同様に，以下に示すような問題の均質性を仮定する．

(1) すべてのサーバの処理性能は等しい．

$$\tau_{ij} = \tau_i \tag{2.3}$$

(2) 故障f_iに対する処理がサーバS_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確率である．

$$\delta_{ij} = \delta_i \tag{2.4}$$

通信確率：λ_{ij}

クライアントからサーバS_j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S_jはm個のターゲット故障に対する処理を終えた後，クライアントと通信する．すなわち，故障f_iに対する処理の後，サーバS_jとクライアントの間で通信が行われる確率は，

$$\lambda_{ij} = \frac{1}{m} \tag{2.5}$$

となる．

故障f_iに対する処理がサーバS_jに割り当てられる確率：δ_{ij}

クライアントは故障表中の処理されていない故障表から未処理の故障をm個取り出し，それをターゲット故障としてサーバへ転送する．サーバはクライアントから受け取ったm個のターゲット故障のうちから一個を取り出し，その故障に対するテストパターンを生成する．そして生成されたテストパターンを用いて，m個のターゲット故障だけでなく，クライアントから受け取った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m個のターゲット故障に対する処理がすべて終了するまでこれを繰り返す．

このとき，m個のターゲット故障に対する処理によって新たにρm個の故障が処理される（検出される，または冗長と判定される）と仮定する．このρを**新たに処理される故障の比**と呼ぶ．すなわち，

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

この比ρは処理が進むにつれて変化する．i番目の処理が行われるときの新たに処理される故障の比をρ_iと表す．

故障f_iに対する処理がN台のうちのいずれかのサーバで行われる確率は，

$$\sum_{j=1}^{N} \delta_{ij} = \frac{1}{\rho_i} \tag{2.6}$$

一方，均質問題の仮定 δ_{ij}=δ_i より，

$$\sum_{j=1}^{N} \delta_{ij} = \sum_{j=1}^{N} \delta_i = N\delta_i \tag{2.7}$$

したがって，故障f_iに対する処理がサーバS_jに割り当てられる確率は，

$$\delta_{ij} = \delta_i = \frac{1}{N\rho_i} \tag{2.8}$$

となる．

新たに処理される故障の比：ρ_i

i番目に処理される故障f_iが処理されると新たにρ_iの故障が処理される．このρ_iは，故障シミュレーションの性質から，テスト生成処理の開始直後（iが小さいとき）は大きい値をとるが，処理が進むにつれて（iが大きくなるにつれて）急速に減少していく．

また，クライアントはサーバから結果を受け取ると故障表を更新する．このとき，複数のサーバによって同時に複数の処理が進められているため，異なるサーバ間で同一の故障を処理することがあり得る．この処理を**オーバーラップ処理**と呼ぶ．このオーバーラップ処理は，サーバ数が1の時は発生しないが，サーバ数が2,3,...と増加するほど，また，ターゲット故障数mが大きいほど増加すると考えられる．

これらを考慮して，ρ_iは，

$$\rho_i = \frac{1}{r_0 + r_1 i + r_2 m(N-1)} \tag{2.9}$$

（r_0, r_1, r_2 は定数）

のように表すことができる．

通信時間：τ_c

クライアント・サーバ間で行われる通信一回に要する時間は，次に示す要素から成り立つと考えられる．

(1) クライアント・サーバ間で転送されるデータは故障表であり，そのサイズは不変である．よって転送されるデータ量は一定で，それに要する時間は一定である．

(2) CSモデルにおける通信はクライアントとサーバの間でのみ行われる．一台のサーバがクライアントと通信を行っている間は，その通信が終了するまで，他のサーバはクライアントと通信することはできずに待ち状態となる．このサーバ間の通信競合は，接続されているサーバ数Nに比例して大きくなる．

これらの仮定により，クライアント・サーバ間で行われる通信一回あたりの平均時間は，

$$\tau_c = t_0 + t_1 N \tag{2.10}$$

（t_0, t_1 は定数）

となる．

総処理時間：T

以上の定義および仮定より，一台のクライアントとN台のサーバから成るCSモデルにおけるテスト生成の総処理時間は，

$$T = T_j = \sum_{i=1}^{M} \frac{1}{N}(r_0 + r_1 i + r_2 m(N-1))\left(\tau_i + \frac{1}{m}(t_0 + t_1 N)\right) \tag{2.11}$$

すべての故障に対する処理の平均

$$\tau = \frac{1}{M}\sum_{i=1}^{M} \tau_i \tag{2.12}$$

を用いて表せば，

$$T = \frac{M}{N}\left(r_0 + r_1\frac{M+1}{2} + r_2 m(N-1)\right)\left(\tau + \frac{1}{m}(t_0 + t_1 N)\right) \tag{2.13}$$

となる．総処理時間Tをサーバ数Nの関数として書き換えれば，

$$T = \frac{M}{N}(c_0 N + c_1)(c_2 N + c_3) \tag{2.14}$$

$$c_0 = r_2 m,\quad c_1 = r_0 + r_1\frac{M+1}{2} - r_2 m,$$

$$c_2 = \frac{t_1}{m},\quad c_3 = \tau + \frac{t_0}{m}$$

この式(2.14)をNで偏微分すると，

$$\frac{\partial T}{\partial N} = M\left(c_0 c_2 - \frac{c_1 c_3}{N^2}\right) \tag{2.15}$$

これよりサーバ数が，

$$N_{\mathrm{opt}} = \sqrt{\frac{c_1 c_3}{c_0 c_2}} \tag{2.16}$$

$$= \sqrt{\frac{\left(r_0 + r_1\frac{M+1}{2} - r_2 m\right)\left(\tau + \frac{t_0}{m}\right)}{r_2 t_1}} \tag{2.17}$$

のとき，CSモデルにおける最小総処理時間

$$T_{\min} = M\left(\sqrt{c_0 c_3} + \sqrt{c_1 c_2}\right)^2 \tag{2.18}$$

$$= M\left(\sqrt{r_2 m\left(\tau + \frac{t_0}{m}\right)} + \sqrt{\left(r_0 + r_1\frac{M+1}{2} - r_2 m\right)\left(\frac{t_1}{m}\right)}\right)^2 \tag{2.19}$$

が得られる．

図2.2に，CSモデルにおける総処理時間Tをサーバ数Nの関数として示す．

サーバ数Nすなわちプロセッサ数を増加させていくと，総処理時間Tは次第に減少していき，サーバ数がN_{opt}のとき総処理時間は最小となる．しかし，サーバ数がN_{opt}より大きくなると，総処理時間Tは増加する．

CSモデルでは，N_{opt}より多くのプロセッサが与えられても，より小さい総処理時間を得ることはできない．式(2.17)，(2.19)より，より多くのサーバ数でより小さい総処理時間を得るには，通信時間を表す項のうちの定数t_1を小さくすればよいことがわかる．この定数t_1は，クライアント・サーバ間の通信におけるサーバどうしの通信競合を表している．

サーバ間の通信競合は次の２つからなると考えられる．

(1) 通信経路の競合：一台のクライアントとN台のサーバは一本の通信経路によって接続されているために，一台のサーバがクライアントと通信を行っている間は，通信経路がふさがれているために，他のサーバとは通信することができない．

(2) クライアントの競合：クライアントはサーバから結果を受け取ると，その結果をもとに故障表を更新し，次の新しいタスクをサーバへ転送する．サーバは，クライアントへ結果を送ってから次のタスクを受け取るまでは待ち状態となる．またこの間，他のサーバもクライアントと通信を行うことはできない．

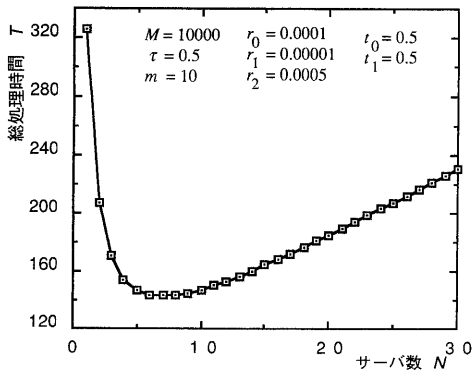

図2.2 サーバ数Nに対する総処理時間T

これらはいずれもシステムのハードウェアに依存するものであり，特に(1)は，通信経路やオペレーティングシステムの性能から決まるために，容易に変化させることはできない．(2)は，サーバ数が大きくなることによって，クライアントへの負荷（故障表の管理などのタスク管理作業）が増大し，各サーバへの応答が追いつかない状況が発生すると思われる．したがって，クライアントへの負荷を軽減するようなシステムを構成することで，より多くのプロセッサを用いて，より小さい総処理時間を得ることができると考えられる．次章では，このシステム構成について考察する．

3 クライアント・エージェント・サーバモデルにおけるテスト生成処理

3.1 CASモデルの構成

クライアント・エージェント・サーバモデル（Client-Agent-Serverモデル，以下CASモデルと呼ぶ）の構成を図3.1に示す．

一台のクライアント・プロセッサには，N_a台のエージェント・プロセッサが，また，それぞれのエージェント・プロセッサにはN_s台ずつのサーバ・プロセッサが接続されている．CSモデルと同様に，すべてのプロセッサは汎用コンピュータから成り，すべてのプロセッサは一本の疎結合分散型ネットワークによって接続されている．

CASモデルにおけるテスト生成処理の流れは以下のようになる．

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，それらをエージェントへのターゲット故障として，故障表とともにエージェントへ転送する．

ターゲット故障と最新の故障表を受け取ったエージェントは，その故障表をもとにエージェント自身の故障表を更新する．そして，そのターゲット故障の中から複数個の故障を選び出し，それらをサーバへの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．

ターゲット故障を受け取ったサーバは，その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を生成処理を行い，その生成されたテストパターンを用いて，エージェントから受け取った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，エージェントから受け取ったターゲット故障に対する処理をすべて終えるまでこれを繰り返し，その結果をエージェントへ返送する．

エージェントはサーバからの結果を受け取ると，その結果をもとに自分の故障表を更新する．そして，クライアントから受け取ったターゲット故障の中から新たなターゲット故障を取り出し，サーバへ転送する．エ

ージェント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，その結果をクライアントへ返送する．

クライアントは，エージェント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エージェントへ転送する．クライアントは，故障表中の故障がすべて処理されるまでこれを繰り返す．

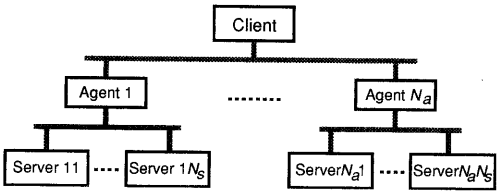

図3.1 CASモデルの構成

3.2 CASモデルにおける問題の定式化

CSモデルにおける問題の定式化と同様に，与えられた回路の全故障数をMとする．j番目のエージェントA_jに接続されたk番目のサーバをサーバS_{jk}と表す．サーバS_{jk}における故障f_iに対する処理時間をτ_{ijk}とする．故障f_iに対する処理がサーバS_{jk}に割り当てられる確率をδ_{ijk}とする．故障f_iに対する処理の後，サーバS_{jk}がエージェントA_jと通信する確率をλ_{sijk}，エージェントA_jがクライアントと通信する確率をλ_{aij}とする．クライアント・エージェント間の，競合による待ち時間を含めたデータ転送に要する平均時間をτ_{ca}，エージェント・サーバ間の，競合による待ち時間を含めたデータ転送に要する平均時間をτ_{cs}とする．

これらの定義により，サーバS_{jk}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，

$$T_{jk} = \sum_{i=1}^{M} \delta_{ijk}\left(\tau_i + \lambda_{aij}\tau_{ca} + \lambda_{cijk}\tau_{cs}\right) \tag{3.1}$$

すべての故障に対する処理が完了するのに必要な平均処理時間は，サーバS_{jk}の処理時間の最大値

$$T = max\{T_{jk}\} \tag{3.2}$$

で表される．

均質問題の仮定

CSモデルと同様に均質問題を仮定する．すなわち，

(1) すべてのサーバの処理性能は等しい．

$$\tau_{ijk} = \tau_i \tag{3.3}$$

(2) 故障f_iに対する処理がサーバS_{jk}に割り当てられる確率は，j, k に無関係で，どの故障に対しても等確率である．

$$\delta_{ijk} = \delta_i \tag{3.4}$$

通信確率：λ_{aij}，λ_{sijk}

クライアントからエージェントA_jへの一回の通信で送られるターゲット故障数をm_aとする．エージェントはm_a個に対する処理を接続されているサーバを利用して行った後，クライアントと通信する．すなわち，故障f_iに対する処理の後，エージェントA_jがクライアントと通信する確率は，

$$\lambda_{aij} = \frac{1}{m_a} \tag{3.5}$$

エージェントA_jからサーバS_{jk}への一回の通信で送られるターゲット故障数をm_sとする．サーバはm_s個のターゲット故障に対する処理を終えた後，エージェントA_jと通信する．すなわち，故障f_iに対する処理の後，サーバS_{jk}がエージェントA_jと通信する確率は，

$$\lambda_{sijk} = \frac{1}{m_s} \tag{3.6}$$

となる．

故障f_iがサーバS_{jk}に割り当てられる確率：δ_{ijk}

クライアントは故障表中のまだ処理されていない故障の中からm_a個を取り出し，それらをターゲット故障としてエージェントへ転送する．エージェントは，クライアントから受け取ったm_a個のターゲット故障の中からm_s個（$m_a \geqq m_s$）を取り出し，それをターゲット故障としてサーバへ転送する．m_s個のターゲット故障を受け取ったサーバは，CSモデルと同様に，それらの故障に対してテストパターン生成と故障シミュレーションの処理を行う．このときの新たに処理される故障の比をCSモデルにおける処理と同様にρで表し，i番目の処理が行われるときの新たに処理される故障の比をρ_iで表す．

故障f_iに対する処理がいずれかのサーバで行われる確率は，

$$\sum_{k=1}^{N_s}\sum_{j=1}^{N_a} \delta_{ijk} = \frac{1}{\rho_i} \tag{3.7}$$

均質問題の仮定，$\delta_{ijk} = \delta_i$ より，

$$\sum_{k=1}^{N_s}\sum_{j=1}^{N_a} \delta_{ijk} = \sum_{k=1}^{N_s}\sum_{j=1}^{N_a} \delta_i = N_a N_s \delta_i \tag{3.8}$$

したがって，故障f_iがサーバS_{jk}に割り当てられる確率は，

$$\delta_{ijk} = \delta_i = \frac{1}{N_a N_s \rho_i} \tag{3.9}$$

となる．

新たに処理される故障の比：ρ_i

CASモデルにおける故障の比ρ_iは，CSモデルにおいて考慮されたサーバ間のオーバーラップ処理のほかに，エージェント間のオーバーラップ処理を考慮する必要がある．CASモデルでは，複数のエージェントが異なるサーバの処理の結果を，それぞれが所有する故障表によって管理しているために，エージェント間でのオーバーラップ処理が発生する．この数は，エージェント数N_aが大きくなるにつれて，また，クライアントからエージェントへのターゲット故障数m_aが大きくなるにつれて大きくなると考えられる．

したがって，CSモデルにおける新たに処理される故障の比（式(2.9)）をもとに，CASモデルにおける新たに

処理される故障の比は，

$$\rho_i = \frac{1}{r_0 + r_1 i + r_2 m_s (N_s - 1) + r_3 m_a (N_a - 1)} \quad (3.10)$$

(r_0, r_1, r_2, r_3 は定数)

となる．

通信時間：τ_{ca}, τ_{cs}

通信時間について，CSモデルにおける通信時間に関する仮定をもとに，以下を仮定する．

(1)クライアント・エージェント間またはエージェント・サーバ間で一回に転送されるデータ量は一定．よってそれに要する時間は一定．

(2)すべてのエージェントからクライアントへの通信要求はエージェント数に比例する．また，一台のエージェントとそれに接続されているサーバとの通信頻度はそのサーバ数に比例する．したがって，すべてのプロセッサを接続する一本の通信経路がふさがれることによって生じるクライアント・エージェント間，エージェント・サーバ間の通信待ち時間は，総エージェント数N_aと総サーバ数N_aN_sとの和に比例する．

(3)クライアントが行うタスク管理作業はエージェント数N_aに比例する．また，エージェントが行うタスク管理作業は，それぞれに接続されているサーバ数N_sに比例する．よってクライアントが作業中であるために生じるエージェント間のクライアント競合による待ち時間はエージェント数N_aに比例．また，エージェントが作業中であるために生じるサーバ間のエージェント競合による待ち時間はサーバ数N_sに比例．

したがって，待ち時間を含めたクライアント・エージェント間の一回あたりの平均通信時間は，

$$\tau_{ca} = t_{a0} + t_{a1} N_a (N_s + 1) + t_{a2} N_a \quad (3.11)$$

(t_{a0}, t_{a1}, t_{a2} は定数)

また，待ち時間を含めたエージェント・サーバ間の一回あたりの平均通信時間は，

$$\tau_{cs} = t_{s0} + t_{s1} N_a (N_s + 1) + t_{s2} N_s \quad (3.12)$$

(t_{s0}, t_{s1}, t_{s2} は定数)

$t_{a0} = t_{s0} = t_0$, $t_{a1} = t_{s1} = t_1$, $t_{a2} = t_{s2} = t_2$ を仮定すれば，

$$\tau_{ca} = t_0 + t_1 N_a (N_s + 1) + t_2 N_a \quad (3.13)$$

$$\tau_{cs} = t_0 + t_1 N_a (N_s + 1) + t_2 N_s \quad (3.14)$$

(t_0, t_1, t_2 は定数)

となる．

総処理時間：T

以上の定義，仮定および(2.12)より，CASモデルにおけるテスト生成処理時間Tは，

$$T = T_{jk} = \frac{M}{N_a N_s} \left(r_0 + r_1 i + r_2 m_s (N_s - 1) + r_3 m_a (N_a - 1)\right) \cdot \left(\tau + \frac{1}{m_a}(t_0 + t_1 N_a (N_s + 1) + t_2 N_a) + \frac{1}{m_s}(t_0 + t_1 N_a (N_s + 1) + t_2 N_s)\right) \quad (3.15)$$

となる．この式をエージェント数N_aの関数として書き換えると，

$$T = \frac{M}{N_a N_s}(c_{a0} N_a + c_{a1})(c_{a2} N_a + c_{a3}) \quad (3.16)$$

$$c_{a0} = r_3 m_a$$

$$c_{a1} = r_0 + r_1 \frac{M+1}{2} + r_2 m_s (N_s - 1) - r_3 m_a$$

$$c_{a2} = \left(\frac{1}{m_a} + \frac{1}{m_s}\right) t_1 (N_s + 1) + \frac{t_2}{m_a}$$

$$c_{a3} = \tau + \left(\frac{1}{m_a} + \frac{1}{m_s}\right) t_0 + \frac{t_2}{m_s} N_s$$

となり，CSモデルにおける，サーバ数Nについての総処理時間の関数Tと同様の形になる．

式(3.16)を，横軸にエージェント数N_a，縦軸に総処理時間Tのグラフとして，図3.2に示す．CSモデルと同様に，エージェント数N_aを増加させていくと総処理時間Tは減少していき，エージェント数が，

$$N_{a\mathrm{opt}} = \sqrt{\frac{c_{a1} c_{a3}}{c_{a0} c_{a2}}} \quad (3.17)$$

のとき，Tは最小となる．そして，$N_a$optを越えるとTは次第に増加してゆく．

このように，CASモデルにおけるクライアントとエージェントは，CSモデルにおけるクライアントとサーバの関係と同様の関係が成り立っていると考えられる．

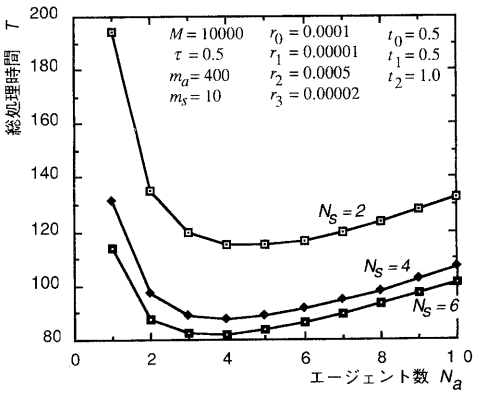

図3.2 エージェント数N_aに対する総処理時間T

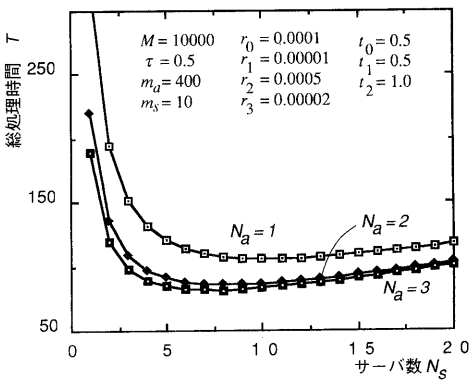

図3.3 サーバ数N_sに対する総処理時間T

すなわち，CASモデルは，それぞれのサーバがN_s台のプロセッサによって高速化されたCSモデルと考えることができる．

また，式(3.15)をエージェント一台当たりのサーバ数N_sの関数として書き換えると，

$$T = \frac{M}{N_a N_s}(c_{s0}N_s + c_{s1})(c_{s2}N_s + c_{s3}) \tag{3.18}$$

$$c_{s0} = r_2 m_s$$

$$c_{s1} = r_0 + r_1\frac{M+1}{2} - r_2 m_s + r_3 m_a(N_a - 1)$$

$$c_{s2} = \left(\frac{1}{m_a} + \frac{1}{m_s}\right)t_1 N_a + \frac{t_2}{m_s}$$

$$c_{s3} = \tau + \left(\frac{1}{m_a} + \frac{1}{m_s}\right)(t_0 + t_1 N_a) + \frac{t_2}{m_a}N_a$$

となり，CSモデルにおける，サーバ数Nについての総処理時間の関数Tと同様の形で表される．

式(3.18)を，横軸にエージェント一台あたりのサーバ数N_s，縦軸に総処理時間Tのグラフとして，図3.3に示す．CSモデルと同様に，サーバ数N_sを増加させていくと総処理時間Tは減少していき，サーバ数が，

$$N_{s\mathrm{opt}} = \sqrt{\frac{c_{s1}c_{s3}}{c_{s0}c_{s2}}} \tag{3.19}$$

のとき，Tは最小となる．そして，$N_{s\mathrm{opt}}$を越えるとTは次第に増加してゆく．

このように，CASモデルにおけるエージェントとサーバとは，CSモデルにおけるクライアントとサーバの関係と同様の関係が成り立っていると考えられる．すなわち，CASモデルは，複数のCSモデルが一台のクライアントによって管理されていると考えることができる．

3.3 総プロセッサ数と総処理時間の関係

CASモデルにおける総プロセッサ数は，

$$\begin{aligned} N_{\mathrm{total}} &= 1 + N_a + N_a N_s \\ &= 1 + N_a(N_s + 1) \end{aligned} \tag{3.20}$$

で表される．

横軸に総プロセッサ数N_{total}，縦軸に総処理時間Tをとり，エージェント数を変化させたときの曲線を図3.4に示す．総プロセッサ数を増加させていくと，図3.3に示したように，それぞれのエージェント数について最小総処理時間を描きながら，エージェント数が増加するにつれて総処理時間は減少していく．そして，総プロセッサがN_{opt}の時，総処理時間は最小となり，N_{opt}を越えると総処理時間は増加してゆく．すなわち，総プロセッサ数Nに対する総処理時間の変化の様子もまた，CSモデルにおけるサーバ数に対する総処理時間の変化と同様の形を描いている．このようにエージェントを設定することで，CSモデルに比べて，より多くのプロセッサ数でより小さい総処理時間を得られることがわかる．

またこの図より，総プロセッサ数N_{total}が与えられた場合，最小の処理時間を得るための最適なエージェント数，およびサーバ数を得ることができる．例えば，図3.4において，与えられた総プロセッサ数$N_{\mathrm{total}} = 25$のとき，エージェント数$Na = 3$，エージェントあたりのサーバ数$N_s = 7$のシステムを構成すれば最小の処理時間を得ることができる．

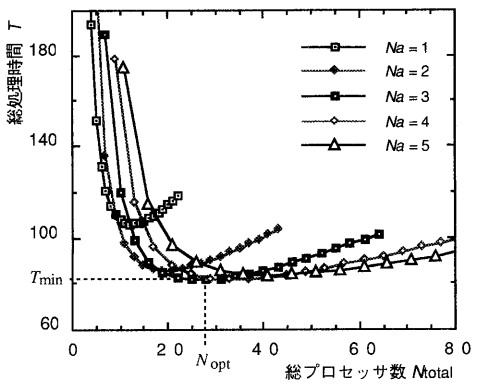

図3.4 総プロセッサ数N_{total}に対する総処理時間T

4 実験結果

CASモデルを実現し，実験を行った．クライアント，エージェント，サーバ，それぞれのプロセッサには，ワークステーションSUN4/LC（12.5MIPS，8MByteメモリ）を使用し，すべてが一本のイーサネット(Ethernet)ネットワークによって接続されている．テストパターン生成アルゴリズムには，FAN[4]を用いた．実験には，ISCAS'89ベンチマーク回路[15]のS15850及びS9234をフルスキャン設計によって組み合わせ回路に変換したものを使用した．

クライアントからエージェントへの一回の通信で送られるターゲット故障数m_a，エージェ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m_sを一定にして，接続するエージェント数Na，サーバ数N_sの違いによる，総処理時間Tの変化を調べた．

2つの回路の結果は同様の傾向を示したので、ここではS15850についてのみ示す。

エージェント数と総処理時間の関係：エージェント数に対する総処理時間の変化の様子を図4.1に示す．解析で示したように，いずれのサーバ数についても，最小の総処理時間を得る最適なエージェント数が存在している．

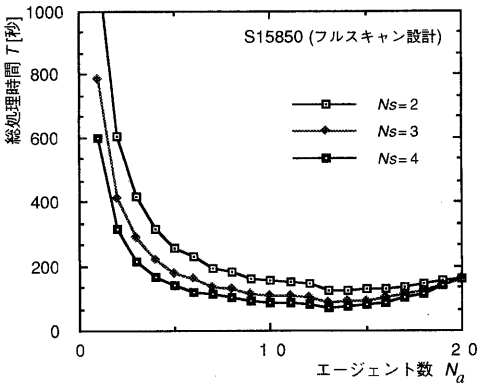

図4.1 エージェント数Naに対する総処理時間T

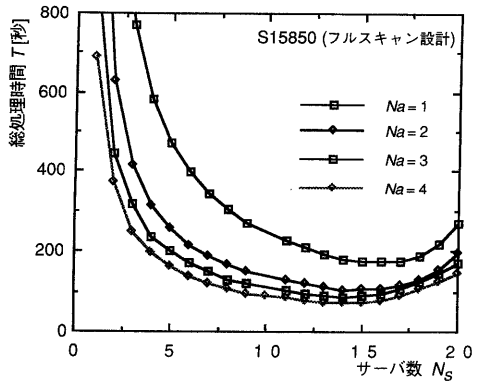

図4.2 エージェント数N_Sに対する総処理時間T

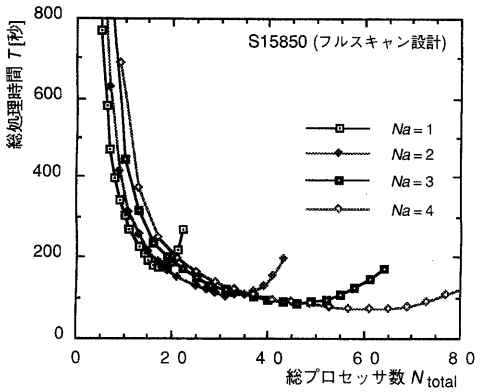

図4.3 総プロセッサ数N_{total}に対する総処理時間T

サーバ数と総処理時間の関係：サーバ数に対する総処理時間の変化の様子を図4.2に示す．解析で示したように，いずれのエージェント数についても，最小の総処理時間を得る最適なサーバ数が存在している．

総プロセッサ数と総処理時間の関係：総プロセッサ数に対する総処理時間の変化の様子を図4.3に示す．それぞれのエージェント数について最適なサーバ数で最小総処理時間を示す曲線を描きながら，全体でまた，最小総処理時間を得る最適総プロセッサ数を示す曲線を描いているといえる．

5 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故障並列法によるテスト生成の並列処理について述べた．

クライアント・サーバモデルにおけるテスト生成処理問題を定式化し，サーバ数（プロセッサ数）に対する総処理時間の変化について解析し，最小の総処理時間を得る最適なサーバ数が存在することを確かめた．

より多くのプロセッサを用いてより小さい総処理時間を得るためのシステム構成としてクライアント・エージェント・サーバモデルを提案した．

クライアント・エージェント・サーバモデルにおけるテスト生成処理問題を定式化し，エージェント数，エージェントあたりのサーバ数，総プロセッサ数，それぞれに対する総処理時間の変化について解析した．そして，与えられたプロセッサ数に対する最適なシステム構成が存在することを示し，実験によって確かめた．

参考文献

[1] O. H. Ibarra and S. K. Sahni, "Polynomially complete fault detection problems," *IEEE Trans. Comput.*, vol. C-24, pp. 242-249, Mar. 1975.

[2] H. Fujiwara and S. Toida, "The complexity of fault detection problems for combinational logic circuits," *IEEE Trans. Comput.*, vol. C-31, pp. 555-560, June 1982.

[3] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Trans. Comput.*, vol. C-30, pp. 215-222, Mar. 1981.

[4] H. Fujiwara and T. Shimono, "On the acceleration of test pattern generation algorithms," *IEEE Trans. Comput.*, vol. C-32, pp. 1137-1144, Dec. 1983.

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

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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当問題，" 電子情報通信学会論文誌，vol. J68-D, No.5, pp.1049-1056, 1985年5月．

[15] F. Brglez, D. Bryan and K.Kozminski, "Combinational profiles of sequential benchmark circuits," *Proc. Int'l Symp. Circuits and Systems, IEEE Press*, New York, 1989, pp. 1929-1934.