論　文

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

正　員　井上　智生[†]　　非会員　米澤　友紀[††]　　正　員　藤原　秀雄[†]

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

Tomoo INOUE [†], *Member*, Tomonori YONEZAWA [††], *Nonmember*
and Hideo FUJIWARA [†], *Member*

あらまし　本論文では，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手法を提案する．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．更に，ISCAS '89 ベンチマーク回路に対して，140 台のワークステーションからなるネットワーク上での実験結果を示す．

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

1. まえがき

論理回路のテスト生成問題は，組合せ回路でさえ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，今後ますます大規模化される論理回路に対しては，その効果に限界がある．

処理を高速化する方法の一つに，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おいても，いくつかの方法が報告されている[(6)~(13)]．テストパターンを探索する空間を分割して部分空間を各プロセッサへ割り当てる探索並列法[(11)]，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る回路並列法[(6),(9)]，異なる発見的手法をそれぞれのプロセッサに割り当て，一つの故障に対して同時にテスト生成を行う方策並列法[(8)]，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[(10)]などがある．筆者らは文献(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上で，クライアントサーバモデルによる故障並列法を提案し，最適粒度（1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．そこでは，プロセッサ数と総処理時間の関係については述べなかった．

本論文では，まず文献(10)で提案したクライアントサーバモデル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．総処理時間を最小にする最適なプロセッサ数が存在することを理論的解析により示す．従って，この最適プロセッサ数より多くのプロセッサを使ってもこのクライアントサーバモデルでは総処理時間を更に小さくすることはできない．更に，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，プロセッサ数と総処理時間の関係を解析する．また，これらの解析をISCAS '89 ベンチマーク回路を用いて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，

† 奈良先端科学技術大学院大学情報科学研究科，生駒市
Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma-shi, 630-01 Japan
†† 松下電器産業株式会社，門真市
Matsushita Electric Industrial Co., Ltd., Kadoma-shi, 570 Japan

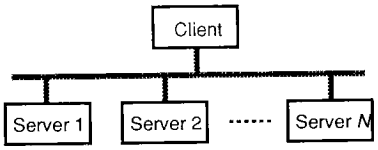

図1　CSモデルの構成
Fig. 1　Architecture of the Client-Server model.

以下，CSモデルと呼ぶ)の構成を図1に示す．1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，1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．ここで取り扱う問題はテスト生成である．従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，故障表と共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，1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．

2.2　CSモデルにおける問題の定式化

ここではまず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超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，

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

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Tは，サーバS_jの処理時間の最大値で表される．

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

[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，文献(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．

$$\tau_{ij}=\tau_i \qquad (3)$$

(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．

$$\delta_{ij}=\delta_i \qquad (4)$$

各故障に対する処理時間τ_iの平均をτとする．すなわち，

$$\tau=\frac{1}{M}\sum_{i=1}^{M}\tau_i \qquad (5)$$

とする．

[総処理時間]　T

クライアントからサーバS_jへの1回の通信で送られるターゲット故障数をmとすると，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，

$$\delta_{ij}=\frac{1}{N}(r_0+r_1 i+r_2 mN)\quad (r_0,\ r_1,\ r_2 \text{は定数}) \qquad (6)$$

故障f_iに対する処理の後，サーバS_jがクライアントと通信する確率は，

$$\lambda_{ij}=\frac{1}{m} \qquad (7)$$

また，その通信に要する平均時間は，

$$\tau_c=t_0+t_1 N\quad (t_0,\ t_1 \text{は定数}) \qquad (8)$$

と表すことができる．

従って，CSモデルにおけるテスト生成の総処理時間Tは，

$$T=\frac{M}{N}\left(r_0+r_1\frac{M+1}{2}+r_2mN\right)\left(\tau+\frac{1}{m}(t_0+t_1N)\right) \tag{9}$$

となる(文献(10)参照).

総処理時間 T をサーバ数 N の関数として書き換えれば,

$$T=\frac{M}{N}(c_0N+c_1)(c_2N+c_3) \tag{10}$$

$$c_0=r_2m,\quad c_1=r_0+r_1\frac{M+1}{2},\quad c_2=\frac{t_1}{m},\quad c_3=\tau+\frac{t_0}{m}$$

となる. この式(10)を N で偏微分すると,

$$\frac{\partial T}{\partial N}=M\left(c_0c_2-\frac{c_1c_3}{N^2}\right) \tag{11}$$

となり, これよりサーバ数が,

$$N_{\mathrm{opt}}=\sqrt{\frac{c_1c_3}{c_0c_2}} \tag{12}$$

$$=\sqrt{\frac{\left(r_0+r_1\frac{M+1}{2}\right)\left(\tau+\frac{t_0}{m}\right)}{r_2t_1}} \tag{13}$$

のとき, CS モデルにおける最小総処理時間

$$T_{\min}=M(\sqrt{c_0c_3}+\sqrt{c_1c_2})^2 \tag{14}$$

$$=\left(\sqrt{r_2m\left(\tau+\frac{t_0}{m}\right)}+\sqrt{\left(r_0+r_1\frac{M+1}{2}\right)\left(\frac{t_1}{m}\right)}\right)^2 \tag{15}$$

が得られる.

図2に総処理時間とサーバ数の関係を示す.

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(13), (15)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同士の通信競合を表している.

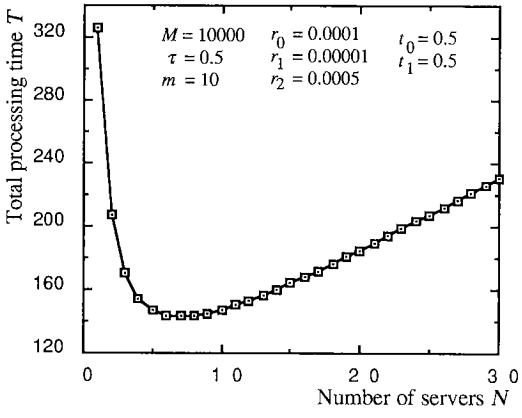

図2 サーバ数 N に対する総処理時間 T
Fig. 2 Total processing time T vs. number of servers N.

サーバ間の通信競合は次の二つからなると考えられる.

(1) 通信経路の競合: 1台のクライアントと N 台のサーバは1本の通信経路によって接続されているために, 1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, ほかのサーバもクライアントと通信を行うことはできない.

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, 容易に改善することはできない. (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. 従って, 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に示す. 1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

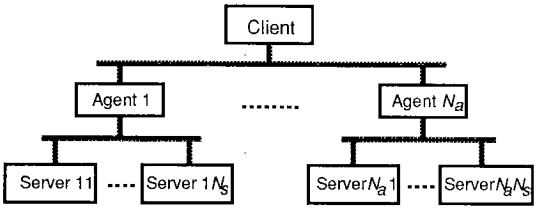

図3 CAS モデルの構成
Fig. 3 Architecture of the Client-Agent-Server model.

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なり，すべてのプロセッサは1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，故障表と共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，故障表と共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1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．

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，

$$T_{jk}=\sum_{i=1}^{M}\delta_{ijk}(\tau_i+\lambda_{aij}\tau_{ca}+\lambda_{cijk}\tau_{cs}) \tag{16}$$

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 T は，サーバ S_{jk} の処理時間の最大値で表される．

$$T=\max\{T_{jk}\} \tag{17}$$

[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．

$$\tau_{ijk}=\tau_i \tag{18}$$

（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．

$$\delta_{ijk}=\delta_i \tag{19}$$

各故障に対する処理時間 τ_i の平均を τ とする．すなわち，

$$\tau=\frac{1}{M}\sum_{i=1}^{M}\tau_i \tag{20}$$

とする．

[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1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，

$$\lambda_{aij}=\frac{1}{m_a} \tag{21}$$

エージェント A_j からサーバ S_{jk} への1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，

$$\lambda_{sijk}=\frac{1}{m_s} \tag{22}$$

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，

$$\sum_{j=1}^{N_a}\sum_{k=1}^{N_s}\delta_{ijk}=\frac{1}{\rho_i} \tag{23}$$

均質問題の仮定，$\delta_{ijk}=\delta_i$ より，

$$\sum_{j=1}^{N_a}\sum_{k=1}^{N_s}\delta_{ijk}=\sum_{j=1}^{N_a}\sum_{k=1}^{N_s}\delta_i=N_aN_s\delta_i \tag{24}$$

従って，故障 f_i がサーバ S_{jk} に割り当てられる確率は，

$$\delta_{ijk}=\delta_i=\frac{1}{N_aN_s\rho_i} \tag{25}$$

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ラップ処理（文献(10)参照）のほかに，エージェント間のオーバ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．

従って，CSモデルにおける新たに処理される故障の比（文献(10)参照）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，

$$\rho_i=\frac{1}{r_0+r_1i+r_2m_sN_s+r_3m_aN_a} \tag{26}$$

(r_0, r_1, r_2, r_3 は定数)

と表すことができる．

[通信時間] τ_{ca}, τ_{cs}

CASモデルにおける通信について次のことが言える．

（1） クライアント・エージェント間およびエージェント・サーバ間で1回に転送されるデータはいずれも故障表であり，その大きさは一定．

（2） クライアント・エージェント間およびエージェント・サーバ間での通信は，いずれも同一の通信経路を介して行われる．また，通信が行われるプロセッサ対の数は，エージェント数と総サーバ数の和である．

（3） エージェントはクライアントに結果を転送し，次の新しいタスクを受け取るまで待ち状態となる．またこの間，他のエージェントは通信することはできない．

（4） サーバはエージェントに結果を転送し，次の新しいタスクを受け取るまで待ち状態となる．またこの間，そのエージェントに接続されたほかのサーバは通信することができない．

（5） クライアントが行うタスク管理作業とエージェントが行うタスク管理作業は，故障表の更新とターゲット故障の決定からなり，それらの管理作業は同じである．

これら(1)～(5)に基づき，以下を仮定する．

（1） クライアント・エージェント間とエージェント・サーバ間の1回のデータ転送に要する時間は等しく，また一定である．

（2） 1本の通信経路を競合することによって生じる通信の待ち時間は，エージェント数 N_a と総サーバ数 N_aN_s の和に比例する．

（3） クライアントに対するエージェントの待ち時間は，エージェント数 N_a に比例する．

（4） エージェントに対するサーバの待ち時間は，サーバ数 N_s に比例する．

（5） 上で述べた仮定(3)，(4)の待ち時間に関する比例定数は等しい．

これらの仮定をもとに，待ち時間を含めたクライアント・エージェント間の1回当りの平均通信時間として，

$$\tau_{ca}=t_0+t_1N_a(N_s+1)+t_2N_a \tag{27}$$

が得られる．また，待ち時間を含めたエージェント・サーバ間の1回当りの平均通信時間は，

$$\tau_{cs}=t_0+t_1N_a(N_s+1)+t_2N_s \tag{28}$$

と表すことができる．ここで，t_0, t_1, t_2 は定数である．

[総処理時間] T

式(9)と同様に，以上の定義と仮定より，CASモデルにおけるテスト生成処理時間 T は，

$$T=\frac{M}{N_aN_s}\left(r_0+r_1\frac{M+1}{2}+r_2m_sN_s+r_3m_aN_a\right)$$

$$\cdot\left(\tau+\frac{1}{m_a}(t_0+t_1N_a(N_s+1)+t_2N_a)\right.$$
$$\left.+\frac{1}{m_s}(t_0+t_1N_a(N_s+1)+t_2N_s)\right) \quad (29)$$

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，

$$T=\frac{M}{N_aN_s}(c_{a_0}N_a+c_{a_1})(c_{a_2}N_a+c_{a_3}) \quad (30)$$

$$c_{a_0}=r_3m_a$$

$$c_{a_1}=r_0+r_1\frac{M+1}{2}+r_2m_sN_s$$

$$c_{a_2}=\left(\frac{1}{m_a}+\frac{1}{m_s}\right)t_1(N_s+1)+\frac{t_2}{m_a}$$

$$c_{a_3}=\tau+\left(\frac{1}{m_a}+\frac{1}{m_s}\right)t_0+\frac{t_2}{m_s}N_s$$

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(30)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4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，

$$N_{a\mathrm{opt}}=\sqrt{\frac{c_{a_1}c_{a_3}}{c_{a_0}c_{a_2}}} \quad (31)$$

のとき，T は最小となる．そして，$N_{a\mathrm{opt}}$ を超えると T は次第に増加していく．

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．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(29)をエージェント1台当りのサーバ数 N_s の関数として書き換えると，

$$T=\frac{M}{N_aN_s}(c_{s_0}N_s+c_{s_1})(c_{s_2}N_s+c_{s_3}) \quad (32)$$

$$c_{s_0}=r_2m_s$$

$$c_{s_1}=r_0+r_1\frac{M+1}{2}+r_3m_aN_a$$

$$c_{s_2}=\left(\frac{1}{m_a}+\frac{1}{m_s}\right)t_1N_a+\frac{t_2}{m_s}$$

$$c_{s_3}=\tau+\left(\frac{1}{m_a}+\frac{1}{m_s}\right)(t_0+t_1N_a)+\frac{t_2}{m_a}N_a$$

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(32)を，横軸にエージェント1台当りのサーバ数 N_s，縦軸に総処理時間 T のグラフとして，図5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，

$$N_{s\mathrm{opt}}=\sqrt{\frac{c_{s_1}c_{s_3}}{c_{s_0}c_{s_2}}} \quad (33)$$

のとき，T は最小となる．そして，$N_{s\mathrm{opt}}$ を超えると T は次第に増加していく．

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1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．

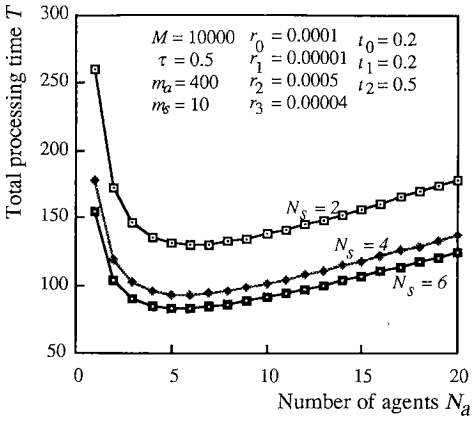

図4　エージェント数 N_a に対する総処理時間 T
Fig. 4　Total processing time T vs. number of agents N_a.

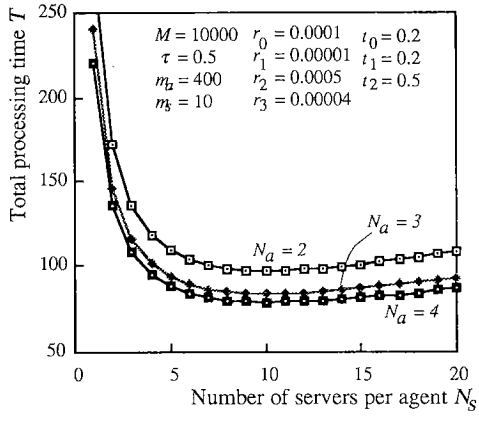

図5　サーバ数 N_s に対する総処理時間 T
Fig. 5　Total processing time T vs. number of servers per agent N_s.

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,

$$N_{total}=1+N_a+N_aN_s$$
$$=1+N_a(N_s+1) \quad (34)$$

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6に示す. 総プロセッサ数を増加させていくと, 図5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} のとき, 総処理時間は最小となり, N_{opt} を超えると総処理時間は増加していく. 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6において, 与えられた総プロセッサ数 $N_{total}=19$ のとき, エージェント数 $N_a=3$, エージェント当りのサーバ数 $N_s=5$ 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.

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 (12.5 MIPS, 8 MByte メモリ) を使用し, すべてが1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 S9234, S13027, および S15850 をフルスキャン設計によって組合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1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1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 N_a, 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.

[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7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.

[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8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9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.

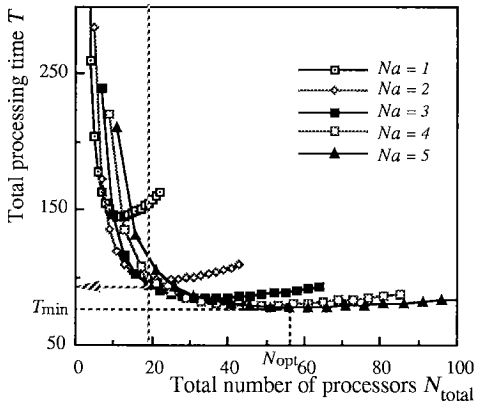

図6 総プロセッサ数 N_{total} に対する総処理時間 T
Fig. 6 Total processing time T vs. total number of processors N_{total}.

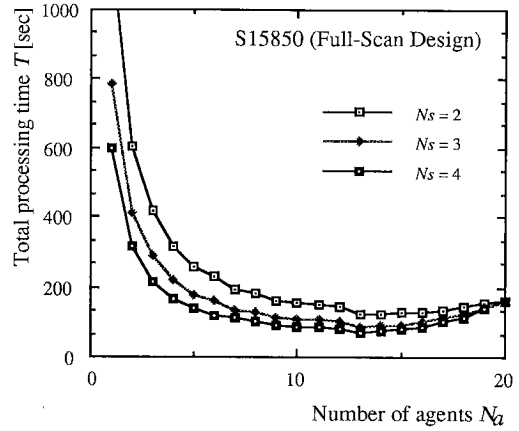

図7 エージェント数 N_a に対する総処理時間 T
Fig. 7 Total processing time T vs. number of agents N_a.

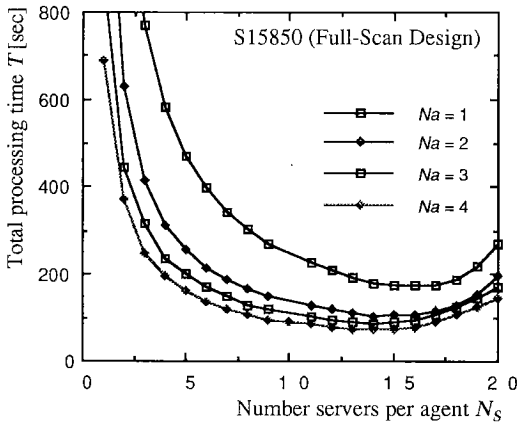

図8　サーバ数 N_s に対する総処理時間 T
Fig. 8　Total processing time T vs. number servers per agent N_s.

この結果において，$N_a \geqq 2$ のときに得られる最小処理時間は，$N_a=1$ のときに得られる最小処理時間に比べて，いずれも小さくなっている．$N_a=1$ の CAS モデルは CS モデルと同様の働きをするので，これを CS モデルとみなすことができる．従って，CAS モデル ($N_a \geqq 2$) によって，CS モデルでの最小処理時間を更に小さくできることが確かめられた．

5. むすび

本論文では，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，エージェント当りのサーバ数，および総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た．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理論的解析により示し，また，これらの解析結果を ISCAS '89 ベンチマーク回路を用いて実験によって確かめた．ここでは，与えられた回路に対する最適なシステム構成を求める問題はまだ解決されていない．最適なシステム構

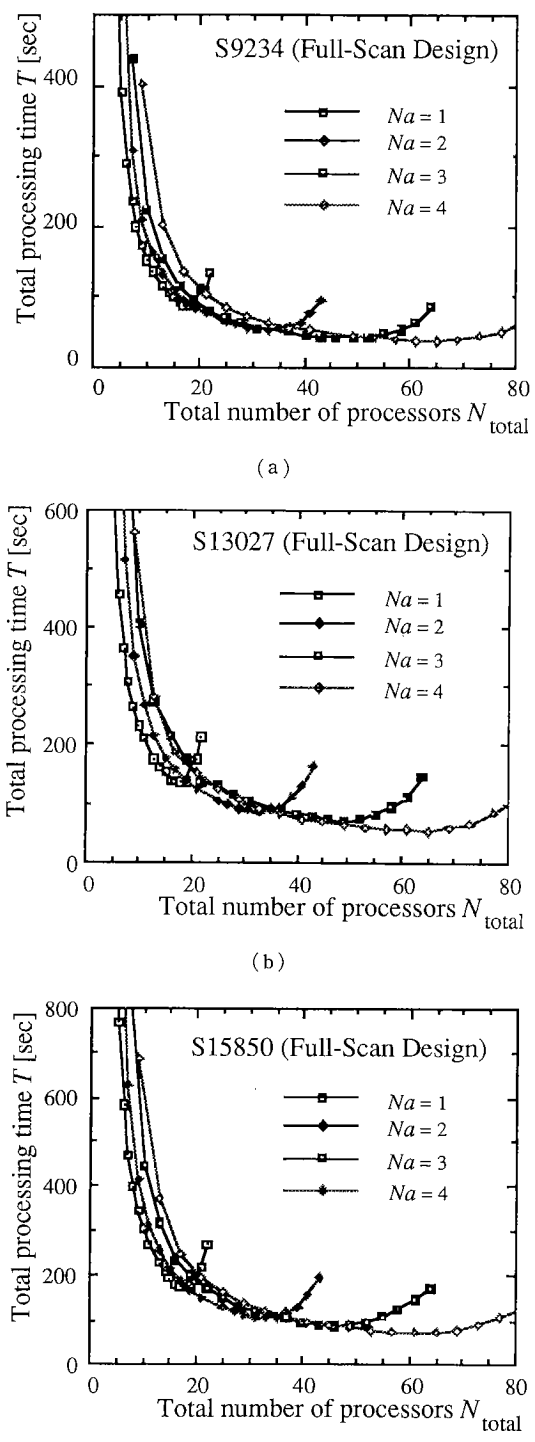

図9　総プロセッサ数 N_{total} に対する総処理時間 T
Fig. 9　Total processing time T vs. total number of processors N_{total}.

成をあらかじめ予測するために，ここでの理論的解析に用いられた種々のパラメータを推定することが課題として残されている．

文　献

(1) Ibarra O. H. and Sahni S. K.: "Polynomially complete fault detection problems", IEEE Trans. Comput., **C-24**, pp. 242-249 (March 1975).

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

(3) Goel P.: "An implicit enumeration algorithm to generate tests for combinational logic circuits", IEEE Trans. Comput., **C-30**, pp. 215-222 (March 1981).

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

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

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

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

(8) Chandra S. J. and Patel J. H.: "Test generation in a parallel processing environment", Proc. IEEE Int'l Conf. Computer Design, pp. 11-14 (1988).

(9) Hirose F., Takayama K. and Kawato N.: "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) Fujiwara H. and Inoue T.: "Optimal granularity of test generation in a distributed system", IEEE Trans. Computer-Aided Design, **9**, 8, pp. 885-892 (Aug. 1990).

(11) Patil S. and Banerjee P.: "A parallel branch-and-bound algorithm for test generation", IEEE Trans. Computer-Aided Design, **9**, 3, pp. 313-322 (March 1990).

(12) Patil S. and Banerjee P.: "Performance trade-offs in a parallel test generation/fault simulation environment", IEEE Trans. Computer-Aided Design, **10**, 12, pp. 1542-1558 (Dec. 1991)

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

(14) 下条真司，宮原秀夫，高島堅助："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"，信学論(D)，**J68-D**, 5, pp. 1049-1056 (1985-05).

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

(平成5年3月24日受付，6月21日再受付)

井上　智生

昭63明治大・工・電子通信卒．平2同大大学院博士前期課程了．同年松下電器産業(株)に入社，明治大大学院博士後期課程を経て，現在奈良先端科学技術大学助手．松下電器産業(株)においてマイクロプロセッサの研究開発に従事．明治大，奈良先端大において，テスト生成，並列処理，テスト容易化設計の研究に従事．

米澤　友紀

平3明治大・工・電子通信卒．平5同大大学院博士前期課程了．同年松下電器産業(株)に入社．明治大在学中，主として並列処理，テスト容易化設計の研究に従事．

藤原　秀雄

昭44阪大・工・電子卒．昭49同大大学院博士課程了．同大・工・電子助手，明治大・工・電子通信助教授，情報科学教授を経て，現在奈良先端科学技術大学教授．昭56ウォータールー大客員助教授．昭59マッギル大客員準教授．論理設計論，高信頼性設計，設計自動化，テスト容易化設計，テスト生成，並列処理，計算複雑度に関する研究に従事．著書「Logic Testing and Design for Testability」(MIT Press)など．情報処理学会会員．IEEE Fellow．