Vol. 27 No. 2　　情報処理学会論文誌　　Feb. 1986

多数個マルチプロセッサの時間評価のためのシミュレータ†

相原玲二††　栄藤稔††
松本裕††　阿江忠††

マルチプロセッサなど並列処理の手法を用いることによりシステムの高速化を図る試みはすでに多く行われている．半導体技術等の進歩によりますます高多重の並列処理システムが実現可能となりつつあり，今後さらにこの方向の発展が期待される．しかし，マルチプロセッサにおいて，一般にプロセッサ数を増加しても速度向上はある点で飽和する傾向にあり，期待どおりにならないことも多い．特に，多数個のプロセッサをもつシステムを製作する場合，事前に行う性能評価が重要となる．本論文では，多数個（百個以上を想定）マルチプロセッサ上の処理をシミュレーションにより性能評価するのに適したモデルを提案し，さらに実際に製作したシミュレータについて述べる．モデルおよびシミュレータは，特にプロセッサ間の通信時間の影響を考慮に入れて作られている．今回のインプリメントでは，扱い得る処理フローに一種の制約を設けているが，処理フローの記述の容易さや会話的実行などによる操作性の良さが主な特長となっている．

1. まえがき

半導体技術の進歩に伴う素子の高集積化，低価格化により，比較的小規模のプロセッサを多数使用した高速コンピュータシステムを構築することが注目されている[1]．今後さらに集積度が上がり1チップ上に多くのプロセッサおよびメモリが格納可能となれば，より一層性能対価格比の良い高多重並列処理システムの実現が期待される．

一方，並列処理システムに関する研究は多くの人々によって行われているが今なおいくつもの問題点を残している[2]．問題点の一つであるプロセッサ間の結合(形状)は，プロセッサ間の通信量に応じシステムの性能，とりわけ処理速度に大きな影響を及ぼす．しかもこの傾向はプロセッサ数が多くなるほど大きい．多数個のプロセッサからなる並列処理システムを設計・実現する場合，プロセッサの結合と通信量に注目した性能予測を事前に，可能な限り正確に行う必要がある．

処理時間に注目した並列処理システムの性能評価についてすでにいくつかの提案があるが，それらは(1)与えられたプロセッサ結合に対し最適な（またはそれに近い）タスク配置を求めるアプローチと(2)与えられたプロセッサ結合，与えられたタスク配置に対し性能を予測するアプローチとに大別できる．(1)はストレートフォワードな設計が期待でき，その手法としてはグラフ理論の最小カットアルゴリズムを用いる解法[3]やヒューリスティックな解法[4]などが提案されている．しかし，このアプローチは主に少数（数個～数十個程度）のプロセッサを対象にしたもので，多数（数十個以上）について求めることは，たとえヒューリスティックな方法を用いても現実的でない．(2)のアプローチの場合は性能評価の結果をフィードバックすることにより設計を繰返し，要求仕様が満足されるまで続ける[5]．このアプローチでは待行列モデルを用いる方法とモンテカルロ法などによるシミュレーションを行う方法が代表的なものとして挙げられる．

本論文では多数個（百個以上）マルチプロセッサの処理時間に関する性能評価を対象としており，その場合の評価システムに要求される事項として

（ⅰ）各タスクにおける制御方式の違いの評価
（ⅱ）動的性能の評価
（ⅲ）システム中の特定の部分に関する評価

などが挙げられる．(ⅰ)はデータの分岐・合流点における制御方式の違いなどを意味しており，データの分配順序などが与える影響について評価可能となる．(ⅱ)はデータの到着間隔にばらつきがあるような場合の評価を意味する．(ⅲ)は実時間処理システムなどにおいて動作時間に制限の与えられた部分（全処理中の一部分）がある場合，果たして条件を満たすか否かを評価することである．そのためには，評価対象となるシステム中の任意の部分を指定して動作を観測できることが望ましい．(ⅱ)(ⅲ)は実時間処理システムを対象とする場合特に重要な要素となる．(ⅰ)～(ⅲ)の条

† A Many Processor Simulator for Performance Evaluation by Reiji Aibara, Minoru Etoh, Hiroshi Matsumoto and Tadashi Ae (Cluster II (Electrical and Industrial Engineering), Faculty of Engineering, Hiroshima University).

†† 広島大学工学部第二類（電気系）

件およびプロセッサが多いことなどを考慮して，本論文では(2)のフィードバックのアプローチのうちシミュレーションによる性能評価方法を採用する．以下では，多数個マルチプロセッサ用シミュレータのための評価モデルを示し，さらに作成したシミュレータについて述べる．

2. 評価モデル

性能評価を行う場合，採用するモデルが妥当なものか否かが重要である．単一プロセッサにおけるモデリングの手法およびシミュレータは多く提案されており[6]，すでに確立していると考えられる．一方，論理ゲートレベルでシステム(回路)の動作を評価する論理シミュレータが製作されており[7]，LSI の設計等に利用されつつある．単一プロセッサの性能評価は一般にモデル化のレベルが高く，論理ゲートの場合モデル化のレベルは非常に低い．本論文の対象とする並列処理システムの場合はこれらの中間に位置すると考えられる．正確な性能評価のためには，マシンインストラクションまたは基本演算やシステムコールなどのレベルを取り扱う必要がある[8]が，評価に要する時間は大きくなる．これに対し，本論文では，1プロセッサにおいて1ないし数個程度のタスクが動作するものとし，タスクのレベルを単位として取り扱う．高度の並列処理を行う場合個々のプロセッサでの処理はさほど複雑にはならないと考えられるため，上記の設定は適当であろう．

並列処理システムの性能評価のためのモデルとしては拡張されたペトリネット[9]などが提案されているが，適応可能なシステムが制限されており，シミュレーションを前提としたモデルとしては適当とは言えない．以下では，シミュレーションにより多数個マルチプロセッサの実行時間評価を行うため，その上で動作するプログラムのモデルを提案する．

2.1 モデルの定義

提案するモデルの基本要素は**タスク**と**リンク**である．タスクはいくつかの逐次処理の集合であり，一度実行を開始すると終了するまで他のタスクとのコミュニケーションはないものとする．タスクは一つまたは複数のタスクからの**トークン**（データまたは制御信号）により実行を開始し，実行終了後一つまたは複数のタスクへトークンを送る（後で述べる特殊なタスクを除く）．一般にプロセスは他のプロセスとコミュニケーションをしながら実行が進むが，ここではいくつかのタスクによりプロセスが構成されると考える．リンクは二つのタスク間でのトークンの伝達を実現し，伝達は一方向に行われる．

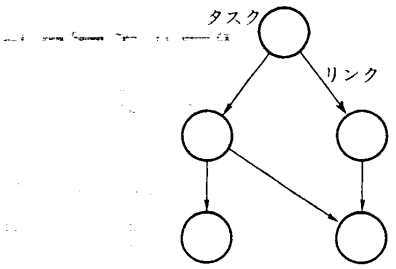


図 1 シミュレーションモデルの例
Fig. 1 An example of simulation model.

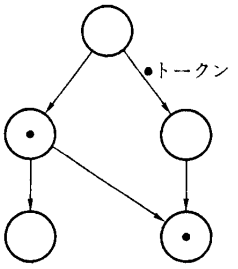


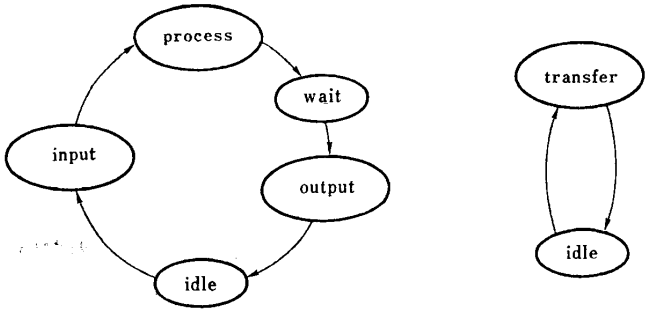
図 2 トークン付きモデル
Fig. 2 Simulation model with tokens.

(a) ソース・タスク (b) シンク・タスク

図 3 特殊なタスク
Fig. 3 Special tasks.

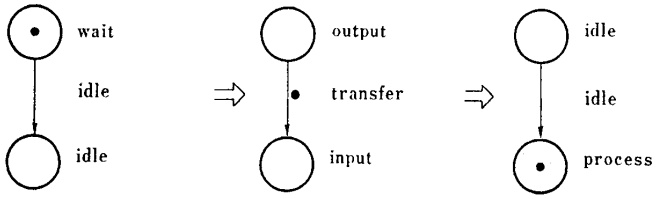
タスクを円，リンクを有向枝に対応させることによりプログラムを**図1**のようにグラフで表現できる．また，トークンをドットで表すことによりプログラムの実行状態を示すこともできる（**図2**）．トークンはタスク上およびリンク上に位置することができ，トークンがタスク上にあるときは処理中（待ち状態も含む）を示し，リンク上にあるときはトークンが伝達中であることを示している．タスクには2種類の特殊なタスクが存在し，ソース・タスク，シンク・タスクと呼ばれる．前者はトークンを発生して送出し，後者はトークンを受け取り消滅させる（**図3**）．

タスクおよびリンクの状態遷移を**図4**に示す．タスクにおいては入力リンク数，出力リンク数が複数である場合があるが基本的には同様の遷移を示す．トークンがタスクからタスクへリンクを介して伝達されるときの各タスクおよびリンクの状態を**図5**に示す．トー



(a) タスク　　(b) リンク

図 4 タスクとリンクの状態遷移
Fig. 4 State transitions of task and link.



(a) 転送直前　　(b) 転送中　　(c) 転送直後

図 5 トークンの転送
Fig. 5 Token transfer.

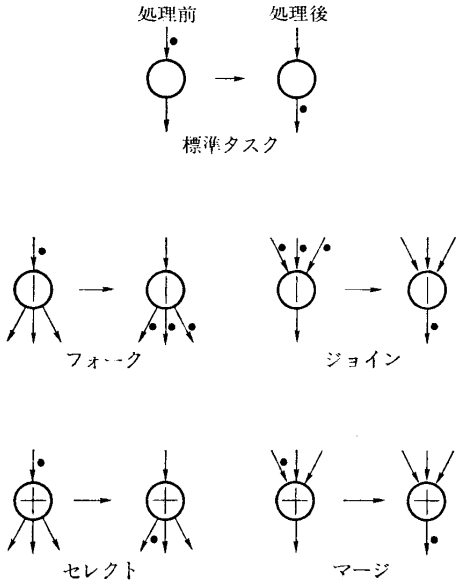


図 6 用意されたタスク
Fig. 6 Given tasks.

クンの伝達は図5(a)の状態になるまで開始されることはなく，伝達中は(b)の状態，さらに伝達終了直後は(c)となる．状態遷移から明らかなように，このモデルでは，タスクおよびリンク上にトークンはたかだか1個しか存在しない．

2.2 タスクにおけるトークン制御

以上の定義においては，入力リンク数，出力リンク数が複数の場合のタスクの動作が明確でない．例えば入力リンク数が複数の場合タスクが処理を開始する条件はどれか一つのトークンの到着とするかあるいは全入力リンクからのトークン到着とするか，また，出力リンク数が複数の場合タスクの処理終了後どれか一つの出力リンクへのみトークンを出すかあるいは全出力リンクへトークンを出すかなどである．シミュレーションにおいてはこれらトークンの制御に関する情報を各タスクに与える必要があるが，どのような動作をするタスクを用意すべきかということはシミュレーションの目的などにより異なる．もとのプログラムの動作を正確にトレースするような情報を与えれば精度の高いシミュレーションが期待できるがシミュレーションに必要な時間は増大する．以下では，後述するシミュレータで用意したタスクの動作を示す．(製作したシミュレータにおいてタスクの制御情報はあらかじめ与えられており，ユーザによる定義は認められていない．)

今回用意したタスクの処理前後の様子を**図6**に示す．

標準タスク

入力リンク1，出力リンク1であるタスク．入力リンクよりトークン到着後直ちに処理を開始し，処理終了後出力リンクへトークンを送出する．送出する場合は図5に示したように，一度 wait 状態に入りリンクおよび次のタスクが idle 状態になるまで待ち，これら条件がそろえば直ちに伝達が開始される．

フォーク

入力リンク1，出力リンク2以上のタスク．トークン到着後直ちに処理を開始し，処理終了後すべての出力リンクへトークンを送出する．送出は，出力リンクごとに標準タスクにおける出力と同様の動作が行われ，それらは並行に処理されるものとする．

セレクト

入力リンク1，出力リンク2以上のタスク．トークン到着後直ちに処理を開始し，処理終了後どれか一つの出力リンクを選択しトークンを送出する．

ジョイン

入力リンク2以上，出力リンク1のタスク．すべての入力リンクからトークンが到着した時点で処理を開始する．入力は各入力リンクごと並行に処理されるものとする．処理終了後出力リンクへトークンを一つだけ送出する．

マージ

入力リンク2以上，出力リンク1のタスク．入力リンクのうちどれか一つからトークンが到着すると直ちに処理を開始し，処理終了後出力リンクへトークンを送出する．一つの入力トークンに対し一つの出力トークンが対応する．

2.3 評価量

ここでは，システムの性能を表す二つの評価量，伝達時間と通過間隔について述べる．

伝達時間

あるリンクから別のあるリンクまでトークンが通過するのに要する時間を伝達時間と呼ぶ．この評価量はシステムの応答特性を示しており，実時間処理システムにおいては特に重要な要素となる[10]．伝達時間を求めるにはあるトークンに注目してその動きを追跡する必要があるが，2.2 節で述べたフォークやジョインのようなタスクを通過する場合，トークンの対応が1対1でなくなり追跡は難しい．ここでは，例えばトークン i がフォークを通過することにより複製された場合出力されるすべてのトークンをトークン i に対応するトークンと呼ぶ．伝達時間は，注目したトークンに対応するトークンのうち最初に到着したものの時間とする．

定義

リンク a を i 番目 ($i=1, 2, \cdots, n$) に通過するトークンをトークン i とする．トークン i がリンク a を通過開始する時刻を S_i，トークン i に対応するトークンのうちリンク b を最初に通過開始する時刻を O_i とすると，リンク a からリンク b への伝達時間 T_i は

$$T_i=O_i-S_i$$

である．

実際にシステムの性能を表現する場合，

初期伝達時間 T_0,

最大伝達時間 $T_{max}=\max_i T_i$

などが用いられる．

通過間隔

あるリンクに注目し，そこを通過するトークンとトークンの時間間隔を通過間隔と呼ぶ．伝達時間は一つのトークンに対する処理時間を示しているが，これに対し通過間隔は，システム全体または一部において平均どのくらいのトークンが処理されているかを示すための評価量と考えられ，これもシステムの性能を示す重要な要素である．

定義

リンク a を i 番目 ($i=1, 2, \cdots, n$) に通過するトークンをトークン i とする．トークン i がリンク a を通過開始する時刻を S_i とすると，リンク a における通過間隔 I_i は，$i\geqq 2$ について

$$I_i=S_i-S_{i-1}$$

である．ただし，I_i は $i=1$ について定義されない．

実際にシステムの性能を表現する場合，

$$\text{平均通過間隔}\quad I_{ave}=\frac{1}{n-1}\sum_{i=2}^{n} I_i$$

などが用いられる．

3. シミュレータ

前章で示したモデルに基づき製作したシミュレータ[11]について述べる．シミュレータの構成を**図7**に示す．シミュレータは三つの部分からなりそれぞれの主な機能は以下のとおりである．

インタプリタ部

シミュレータの核にあたる部分で，タスク間の接続情報，タスクの処理時間，通信時間，トークンの入力時間間隔などを入力すると各タスクの動作をシミュレートし，伝達時間，通過間隔などを出力する．

トランスレータ部

インタプリタ部への入力情報は，インタプリタ部の実行に都合が良いよう設計されており非常に複雑な構造をしている．いわば機械語のような入力情報をユーザが直接書くことは，記述の効率が悪く，確認や修正のときにも不要な手間を必要としてしまう．トランスレータ部は，スクリプトと呼ばれるいわば高級言語のようなユーザ記述用入力形式をインタプリタ部入力形式に変換する．

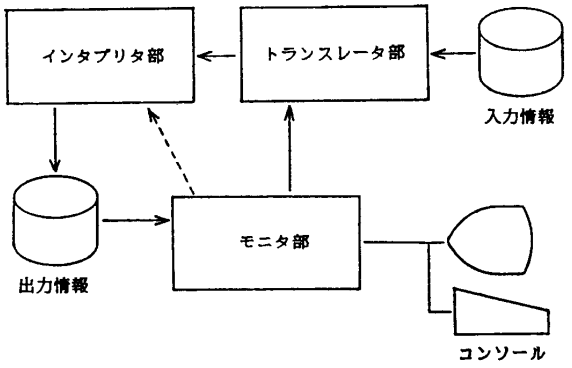


図7 シミュレータの構成
Fig. 7 Configuration of simulator.

モニタ部

シミュレータの会話的使用を可能にするユーザインタフェース．トランスレータ部，インタプリタ部の起動を行い，さらにインタプリタ部に対し会話的に実行の制御を行う．シミュレーションの中断・再開，タスクの状態の読み出し・変更などを行うことができる．

3.1 直並列フロー

今回製作したシミュレータで扱い得るタスクの接続構造は直並列フロー[12]に限っている．これはトランスレータ部による制約であり，インタプリタ部は任意の構造を許している．したがってトランスレータ部のみの変更によりこの制約を除くこともできる．しかし，タスクの接続構造を直並列フローに制限することは

（1） 記述が構造化され，また容易である，

（2） ハードウェアへの対応が容易である[13]，

などの特長を持つ．

本稿で用いる直並列フローとは，以下に示す2種類の分解の再帰的適用により定義される．

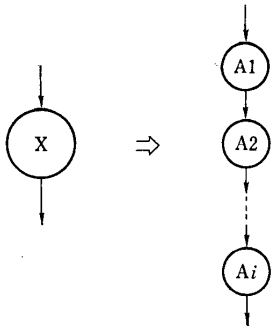
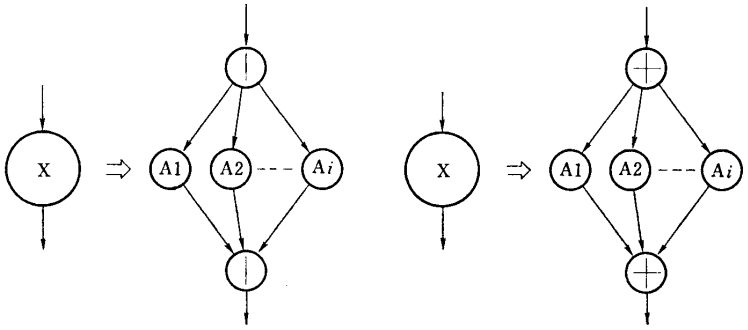


図 8 直 列 化
Fig. 8 Sequential decomposition.



（a） フォーク・ジョイン型　　（b） セレクト・マージ型

図 9 並 列 化
Fig. 9 Parallel decomposition.

直列化

一つの処理モジュール（いくつかのタスクの集まりであり，全体としては1入力1出力．以下，単にモジュールと呼ぶ．）を複数のより小さなモジュールの直列接続に分解する．これはパイプライン処理などに対応する．この分解をモジュールの直列化と呼ぶ（図8）．

並列化

一つのモジュールを（より小さな）モジュールの並列接続に分解する．このとき新たに分岐（1入力多出力）と合流（多入力1出力）のタスクが必要となる．（狭義の）並列処理などに対応する．この分解をモジュールの並列化と呼ぶ（図9）．モジュールの並列化には次の二つの型が存在する．

（a） フォーク・ジョイン型

フォークに入力されたトークンはフォークに接続されている全モジュールに分配され，それらのトークンは各モジュールで処理された後，ジョインで回収され出力される（図9（a））．

（b） セレクト・マージ型

セレクトに入力されたトークンはセレクトに接続されているモジュールの中から一つを選択しトークンを渡す．トークンの渡されるモジュールはラウンドロビン式に順次選択されるとする．各モジュールで処理されたトークンはマージに渡され出力される（図9（b））．

トランスレータ部への入力形式は次のようになる．

直列化

モジュール X を直列化して A1…Ai の並びに変換する場合，以下のように記述する．

X＝A1－A2－…－Ai

並列化

モジュール X を並列化して A1…Ai の並列接続に変換する場合，フォーク・ジョイン型では，

X＝A1|A2|…|Ai

セレクト・マージ型では，

X＝A1＋A2＋…＋Ai

と記述する．

最初の変換を明示するために，開始モジュール（仮想モジュール）を次のように宣言する必要がある．

@（モジュール名）

このモジュールの入力はソース・タスクへ，出力はシンク・タスクへそれぞれ自動的に接続される．直並列フローの記述例を以下に示し，対応するフローのグラ

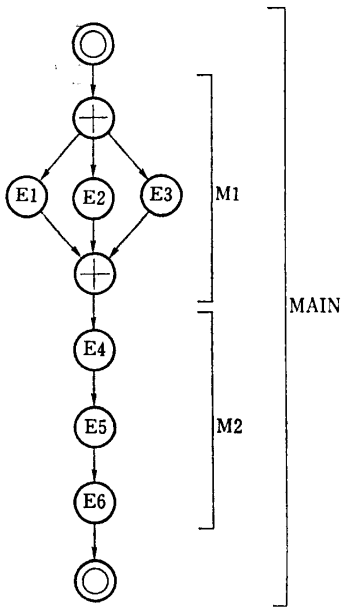


図 10 直並列フローの例
Fig. 10 An example of series-parallel flow.

フを**図 10** に示す.

```
@MAIN
MAIN=M1−M2
M1=E1+E2+E3
M2=E4−E5−E6
```

3.2 時間記述

本シミュレータでは，各タスクの処理時間，リンクの転送時間はトークンの種類等によらず一定として扱う．処理時間は各タスクごとに1トークンあたりの時間を与える．転送時間は直接リンクには与えず，各タスクに対し1トークンあたりの入力時間，出力時間として与える．タスクaからタスクbへリンクが存在する場合，リンクの転送時間はタスクaの出力時間とタスクbの入力時間の大きい方の値で与えられる．

例えば，タスク E1 に入力時間，処理時間，出力時間を与える場合は次のように記述する．

```
#E1（入力時間，処理時間，出力時間）
```

また，セレクト・マージ型によって並列化されたモジュール M1 の場合，

```
#M1（入力時間，セレクト処理時間
                +マージ処理時間，出力時間）
```

と記述する．

システムに対するトークン入力の時間情報も与えることができる．つまり，ソースタスクにおけるトークン発生間隔を記述する．例えば，1番目のトークンが入力され，その t_2 後に2番目のトークン，さらにその t_3 後に3番目のトークンが入力される場合，

```
%t2, t3%
```

と記述する．一定間隔 t で5個のトークンが入力される場合は，

```
%t, t, t, t%  または  %4(t)%
```

と記述する．なお，入力トークンが1個の場合は，

```
%%
```

とすればよい．

3.3 その他特徴

シミュレーション結果

出力可能な情報の主なものは以下のとおり．ただし評価量とは 2.3 節で述べたものを指す．

（1） システム全体に関する各評価量
（2） 指定したモジュールに関する各評価量
（3） トークンごとの動作の詳細

シミュレーションの中断

指定したタスクが特定の条件（トークン入力状態など）を満たすときシミュレーションの中断ができる．さらに，シミュレーションの論理時刻を指定して中断することや，1単位時間ごとに中断・再開するステップ動作も可能である．

ハードウェアの共用

同一のプロセッサに複数タスク，また同一の通信路に複数リンクを割当てた場合をシミュレートすることができる．同一プロセッサに割当てられた複数タスクのうち1タスクのみが処理状態に入ることができ，その処理が終了するまで他のタスクは処理状態へ入れない．リンクについても同様の動作をする．

スクリプト

トランスレータ部への入力文法を**図 11** に示す．

4. シミュレータ動作例

製作したシミュレータの動作を確認するため，すでにマルチプロセッサ上で動作している処理についてシミュレートを行った．この処理は2次元図形のパターンマッチであり，

処理1：データの特徴抽出（GA）
処理2：パターンマッチ（REC）

の二つの処理のシーケンシャルなフローを基本とする．詳細なタスクのフローは図 10 に示した構造をしており，前半の並列化部分が GA，後半の直列化部分が REC に相当する．マルチプロセッサへのインプリメントでは，セレクトとマージを除いて，タスクはプロセッサへ1対1対応をしている．使用したハードウ

```
スクリプト ::= メインモジュール宣言部
             | 構造記述部
             | 時間記述部
             | 入力データ系列記述部
             | ハードウェア制限記述部
             | 観測点記述部
             | ブレークポイント記述部
メインモジュール宣言部 ::= '@' モジュール名
構造記述部 ::= モジュール名 '=' シリーズ型
             | モジュール名 '=' フォーク・ジョイン型
             | モジュール名 '=' セレクト・マージ型
シリーズ型 ::= モジュール名
             | シリーズ型 '-' モジュール名
             | 反復数 '-' モジュール名
フォーク・ジョイン型 ::= フォーク・ジョイン型 '|' モジュール名
                       | 反復数 '|' モジュール名
セレクト・マージ型 ::= セレクト・マージ型 '+' モジュール名
                     | 反復数 '+' モジュール名
時間記述部 ::= '#' モジュール名 '(' 時間 ',' 時間 ',' 時間 ')'
             | '#' モジュール名 '(' 時間 ',' 時間 '|' 時間 ',' 時間 ')'
入力データ系列記述部 ::= '%{' 系列表現式 '}%'
系列表現式 ::= 間隔時間
             | 系列表現式 ',' 間隔時間
             | 反復数 '(' 系列表現式 ')'
ハードウェア制限記述部 ::= 'extask' '{' タスク列 '}'
                         | 'exlink' '{' リンク列 '}'
タスク列 ::= タスク指定
           | タスク列 ',' タスク指定
リンク列 ::= リンク指定
           | リンク列 ',' リンク指定
モジュール指定 ::= '/' モジュール名
                 | '/' モジュール名 '.' 指示子
                 | モジュール指定 '/' モジュール名
                 | モジュール指定 '/' モジュール名 '.' 指示子
タスク指定 ::= モジュール指定
             | モジュール指定 '/' 'fork'   | モジュール指定 '/' 'join'
             | モジュール指定 '/' 'select' | モジュール指定 '/' 'merge'
リンク指定 ::= タスク指定 '(o)' | タスク指定 '(i)'
観測点記述部 ::= 'observer' '{' モジュール指定 '}'
ブレークポイント記述部 ::= タスク指定 'input' '(' トークン番号 ')'
                         | タスク指定 'procs' '(' トークン番号 ')'
                         | タスク指定 'output' '(' トークン番号 ')'
                         | 'at' 停止時刻
                         | 'interval' 停止時間間隔
                         | 'queue' 制限長
                         | 'term' 処理済トークン数
                         | タスク指定 'waittime' '(' 制限時間 ')'
モジュール名 ::= 英字
               | モジュール名 英字
               | モジュール名 数字
```

図 11　スクリプトの文法（一部）
Fig. 11　Syntax of script.

ェアは32台のプロセッサからなるマルチプロセッサ UNIP[14] であり，GA, REC それぞれのタスク数を i, j とすると，$i+j=29$ という条件の下で各々のタスク数を変化させてインプリメントおよび処理時間の測定を行っている．この実験では，セレクトとマージが同一のプロセッサに，一部のリンクが同一の通信路（バス）に割当てられている．

上記の条件をスクリプトで記述し，シミュレーションを行った．各々のタスクの処理時間，通信時間は実測結果から求めて与えている．**図 12** に結果を示す．

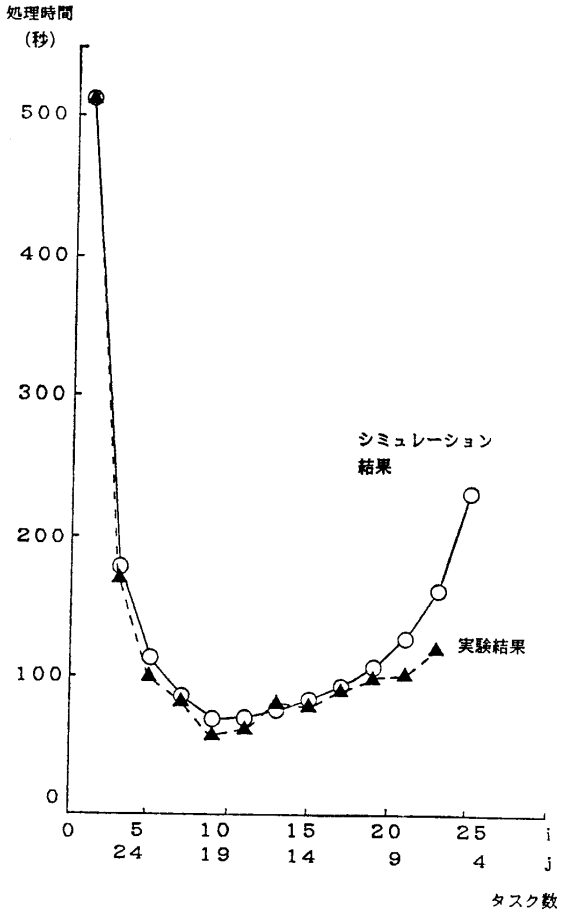


図 12 実験結果とシミュレーション結果
Fig. 12 Results of experiment and simulation.

5. むすび

多数個マルチプロセッサ上の処理をシミュレーションにより評価するためのモデルを示し，それに基づき製作したシミュレータについて述べた．また，すでにマルチプロセッサ上へインプリメントされている処理をシミュレートすることによりシミュレータの動作確認を行った．本シミュレータはすべてC言語で記述されており（ソーステキストは全体で約6,000行）Unixシステム上に実現されている．扱い得るタスク数は最大600（主記憶830キロバイト，仮想記憶のサポートなし）である．シミュレーション時間を短縮するよう最適化されているため，前述の実験のシミュレーションに要する時間はGA数（REC数）の選び方によらずほぼ一定であり，1サンプル点を求めるのに要する時間はMC 68000 (8MHz) CPUのシステムにおいて約610秒（VAX 11/750では約220秒）であった．この場合，シミュレータの走行時間比（モデル上で経過した時間に対するシミュレーション実行に要した時間）は1.2～10.2（VAX 11/750の場合0.5～3.7）となる．さらに，本シミュレータでは指定したタスクまたはモジュールにおける処理状況を細かく観測でき，また，トークンの発生頻度を自由に指定できるため実時間処理システムの評価においても有効である．

本稿で示したシミュレータの動作例ではシミュレーション結果と実験結果がよく一致している．これはプロセッサ内処理時間とプロセッサ間通信時間の正確な値が求まっていたこと，処理フローが比較的単純であったことなどが要因として挙げられる．直並列フローについてはタスク数が百を越えるような場合にも適用可能と考えているが，その精度についてはさらに検討の必要があろう．また，直並列フローの拡張や自由なフローを許す場合の適用可能性については今後の検討を要す．

謝辞 最後に，VAX 11/750の利用は本学市川忠男教授のご好意によるもので，お礼申し上げます．

参考文献

1) Hoshino, T., Kawai, T., Shirakawa, T., Higashino, J., Yamaoka, A., Ito, H., Sato, T. and Sawada, K.: PACS: A Parallel Microprocessor Array for Scientific Calculations, *ACM Trans. Comput. Syst.*, Vol. 1, No. 3, pp. 203-221 (1983).
2) Special Issue on Performance Evaluation of Multiple Processor Systems, *IEEE Trans. Comput.*, Vol. C-32, No. 1 (1983).
3) Stone, H.S.: Multiprocessor Scheduling with the Aid of Network Flow Algorithms, *IEEE Trans. Softw. Eng.*, Vol. SE-3, No. 1, pp. 85-93 (1977).
4) Ma, P.R., Lee, E.Y.S. and Tsuchiya, M.: A Task Allocation Model for Distributed Computing Systems, *IEEE Trans. Comput.*, Vol. C-31, No. 1, pp. 41-47 (1982).
5) Ae, T., Aibara, R. and Etoh, M.: Design and Realization of Many Processor System Using Parallel Flow Graph, Proc. 1984 IEEE Workshop Language for Automation, pp. 7-12 (1984).
6) 三上，箱崎，関野：計算機システムの性能評価技術［I］,［II］，電子通信学会誌，Vol. 59, No. 10/No. 12, pp. 1083-1090/pp. 1369-1376 (1976).
7) 小池，大森，佐々木：論理シミュレーションマシンのアーキテクチャ，情報処理学会論文誌，Vol. 25, No. 5, pp. 864-872 (1984).
8) Ae, T., Chunha, W. C. and Yamasaki, H.: Evaluation Language for Real-Time Controller, Proc. 1984 IEEE Workshop Language for

Automation, pp. 67-71 (1984).
9) Ramamoorthy, C. V. and Ho, G. S.: Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets, *IEEE Trans. Softw. Eng.*, Vol. SE-6, No. 5, pp. 440-449 (1980).
10) Ma, R. P.: A Model to Solve Timing-Critical Application Problems in Distributed Computer Systems, *Computer*, Vol. 17, No. 1, pp. 62-68 (1984).
11) 阿江，相原，栄藤，松本，三保：直並列フローの実行評価とプロセッサ・アレイ上への実現について，信学技報，CAS 84-208, pp. 21-28 (1985).
12) 阿江，相原，栄藤，森福：直並列フローで記述される並列パイプライン処理のためのマルチプロセッサ，信学技報，EC 84-2, pp. 13-24 (1984).
13) 相原，栄藤，阿江：制限されたフローに対するマルチプロセッサのためのネットワーク—コストと性能評価—，「アーキテクチャワークショップインジャパン '84」シンポジウム論文集，pp. 151-158 (1984).
14) 相原，阿江：マルチマイクロプロセッサによるソート／サーチエンジンの試作，情報処理学会論文誌，Vol. 26, No. 2, pp. 349-355 (1985).

(昭和 60 年 5 月 15 日受付)
(昭和 60 年 9 月 19 日採録)