計算機アーキテクチャ　71－17
（1988. 7. 8）

並列推論マシンに於ける負荷分散方式の評価

An Evaluation of the Load Balancing Mechanism for a Parallel Inference Machine

山内　宗、　田中　英彦

Tsukasa YAMAUCHI* and Hidehiko TANAKA**

*日本電気株式会社　　**東京大学工学部
NEC Corp.　　University of Tokyo

あらまし　我々は、並列推論マシンで、Committed choice言語を実行する際の負荷分散方式についての評価・検討を行うために、ソフトウェア・シミュレーションを行った。シミュレーションでは、単一化プロセッサとメモリで構成される推論ユニットが２種の多段ネットワークによって多数並列に接続されたモデルを仮定した。本稿では、シミュレーション・モデルと処理方式、負荷分散方式について述べた後、シミュレーション結果について報告する。

Abstract　There are many important problems in executing committed choice languages in a Parallel Inference Engine PIE. One of them is the load balancing problem. We made a simulation to evaluate more effective load balancing mechanism. In this paper, first we describe the simulation model, the execution mechanism and the load balancing mechanism, then we present the simulation results.

1．はじめに

現在我々は、高並列推論エンジンＰＩＥの開発を進めている。ＰＩＥは、ゴール書換えモデルに基づき、論理型言語を高並列に実行する推論マシンである。現在迄のところ、ＰＩＥに於ける並列実行は、ＯＲ並列を主体に検討が進められてきた。従って、従来の処理方式のままでは、ＧＨＣ、ＦＬＥＮＧ等のCommitted choice言語を効率良く実行することは、難しい。並列推論マシン上で、Committed choice言語を実行する際に問題となるのは、次の２点である。

①共有変数のアクセス競合と負荷分散

②実行制御とスケジューリング

ここでは、まず上記の①について、ソフトウェア・シミュレーションにより、評価、検討を行うことにする。

具体的には、

・負荷分散の方式
・相互結合網の構成

についての評価検討を行う。

シミュレーションの対象とする言語は、ＧＨＣとよく似たCommitted choice言語であるＦＬＥＮＧを用いることにする。

2. 論理型言語ＦＬＥＮＧ

ＦＬＥＮＧは、ＧＨＣと非常によく似た言語であり、次に挙げる点がＧＨＣとの主な相違点である。

①ガード部が存在せず、定義節のヘッド・リテラルの単一化が成功すれば、すぐにコミットされる。

②ボディ部の各リテラルは、論理的ＡＮＤ関係にないので、どれか一つのゴールが失敗しても、他のゴールの実行には影響を与えず、全てのゴールは必ず実行される。ゴール間にＡＮＤ関係を持たせたい場合には、プログラム中に明示的に書く必要がある。

また、ＧＨＣと同様にＦＬＥＮＧに於いてもヘッド・リテラルの単一化の際には、親ゴール中の変数を具体化することはできず、その場合は単一化はサスペンドする。

この様に、共有変数の扱いに関してはＦＬＥＮＧはＧＨＣにかなり似ているが、実行制御に関する部分はかなり簡略化されている。この様にＦＬＥＮＧはＧＨＣに比べて低レベルの言語であるが、ほとんどのＧＨＣプログラムは容易にＦＬＥＮＧプログラムにコンパイルすることが可能であり、また、そのための処理系もすでに完成している。

3. ＰＩＥのシミュレーション・モデル

今回用いたＰＩＥのシミュレーション・モデルを図１、２に示す。図１は全体構成を示し、推論ユニット（ＩＵ）が相互結合網によって並列に接続されて、ＰＩＥを構成している。図２は、各ＩＵの内部構成を示し、一つの推論ユニットは、単一化の処理を行う単一化プロセッサ（ＵＰ）と定義節やゴール・フレーム（ＧＦ）を格納するメモリ・モジュール（ＭＭ）、共有変数等を格納する共有メモリ（ＳＭ）で構成されている。そして、ＵＰは自分と対になっているＭＭ、ＳＭへはネットワークを介さずに高速にアクセスできる。

各ＩＵは基本処理単位であるゴール・フレームをやり取りするための分配網（ＤＮ）と、共有変数の値、構造データ等をやり取りしたり、サスペンドしたゴール・フレームのアクティベイト等の制御を行うための共有メモリ網（ＳＭＮ）の２つの相互結合網で結合されている。共有メモリへのアクセスとゴール分配の際の通信特性が大きく異っていることが、二種のネットワークに分けた理由である。

ＳＭＮとＤＮは、どちらも４×４のクロスバー・スイッチング・ユニット（ＳＵ）を用いた多段ネットワーク（オメガ網）である。特に、ＤＮは、自動負荷分散機能を持った、ネットワークとなっている。

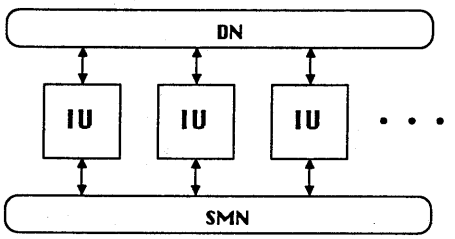
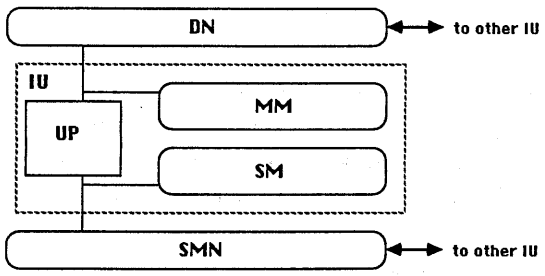


図１　システムの全体構成



IU: Inference Unit
UP: Unify Processor
MM: Memory Module
SM: Shared Memory
DN: Distribution Network
SMN: Shared Memory Network

図２　ＩＵの内部構成

4. シミュレーションに於ける処理方式

4. 1　ゴール・フレームの表現

ＰＩＥの基本処理単位はゴール・フレームであるが、今回のシミュレーションでは、そのゴール・フレームをリテラル単位で構成した。従って、コミット後に生成される、ボディ部の複数のリテラルは、それぞれ別のゴール・フレームとして別のＩＵに送られる。その際、ＳＭＮへのアクセスを減らすために、ローカルなＳＭ中へのポインタは、デリファレンスした結果がＧＦ中に書かれる。

4．2　共有変数の配置と単一化

ＵＰがＧＦの処理を行う際には、まず定義節中の全ての変数の領域をローカルなＳＭ中にとり、undef に初期化する。そして、ヘッド・リテラルの単一化によって生じた結合は、その変数セルに書き込まれる。また、新ＧＦを生成する（この処理をリダクションと呼ぶ）際には、変数セルのうち値がバインドされているものについてはその値を、undef のままのセルについてはそのセルへのポインタをＧＦ中に書き込む。

4．3　ヘッド・リテラル単一化時のサスペンド・アクティベイト

ＦＬＥＮＧでは、次の２通りの場合にヘッド・ユニフィケーションのサスペンドが起こる。

①親ゴール中のundef の変数に値をバインドしようとした時

②定義節のヘッド・リテラル中のbind-to-non-variable annotation のある変数にundef の変数をバインドしようとした時

この様な場合にサスペンドしたＧＦは、そのサスペンドの要因となった変数セルが具体化された時に実行可能となり、アクティベイトされる。ＧＦのサスペンド、アクティベイト処理は次のように行われる。

①ヘッド・ユニフィケーション中に、サスペンドが起こるようなバインドをしようとすると、ＭＭ中のＧＦのアドレスがサスペンドの要因となったundef セルの所に記憶される。

②ＧＦの単一化において、１つの単一化も成功せず、１つ以上の単一化がサスペンドした場合は、ＧＦはアクティブ・キューから外され、サスペンド・キューにつながれる。

③undef セルが具体化されると、ＳＭはサスペンドしていたＧＦをＳＭＮを介してアクティブ・キューにつなぎ直す。

5．負荷分散方式

5．1　負荷分散の目的

ＧＨＣやＦＬＥＮＧの様なCommitted choice言語を並列推論マシン上で高速に効率良く実行させるためには、どの様な負荷分散方式を目指すべきであるか。並列マシンである以上、プログラムの並列度を十分引き出せる様な負荷分散方式を目指すのはもちろんであるが、前述の様にCommitted choice言語は、共有変数のアクセス競合や、実行のサスペンド等の特徴が有り、これらが頻繁に生ずると処理速度に低下をもたらすのは明かであるので、本稿では以下の事項を目標として負荷分散方式を検討する。

①プログラムの並列度を十分に引き出す。
②共有変数へのアクセス競合を減らす。
③実行のサスペンドを減らす。

5．2　自動負荷分散ネットワーク

前述のように、ＰＩＥにはゴール分配網と共有メモリ網の２種の相互結合網がある。特に、ゴール分配網は、自動負荷分散の機能をサポートしている。これは、ゴールの転送とは、逆の向きに行き先ＩＵの負荷情報を伝送し、負荷の量が最小の行き先を選択して、そこにゴールを送りつけることにより、効率的な負荷の分配をするものである。

自動負荷分散ネットワークは、４入力４出力クロスバー・スイッチを基本とするスイッチング・ユニット（ＳＵ）を構成要素としたオメガ網であり、各ＳＵは、次段のＳＵから送られてくる負荷情報を比較し、その中で最少の負荷情報を前段ＳＵに伝送する。そして、負荷分散する時は、最少の負荷情報を伝えて来ている次段ＳＵへの経路を設定する。

5．3　負荷分散のパラメータ

負荷分散をするためには、まず、負荷量というものを何らかの形で評価する必要がある。ＰＩＥの基本処理単位はゴールであるので、負荷情報としてゴールの数というものが考えられる。但し、Committed choice言語においては、サスペンド状態であるゴールの数が非常に多く、これらのゴールの数を考慮に入れずにゴールの分配を行うと、それらのサスペンド状態のゴールが実行可能状態になった時にＩＵの負荷に大きな偏りを生じてしまう可能性がある。そこで、アクティブ（実行可能）なゴール数だけではなく、サスペンド状態のゴールの数も負荷情報の考慮に入れる必要がある。

負荷量の評価の次に考えなければならないの

は、リダクションによって新たに生成されたゴールを自分で実行するのか、あるいは、ＤＮを通じて他のＩＵに送りつけるべきなのかを決定するための負荷量の閾値である。この閾値は、原則的には、ＧＦを自分で処理するコストと他のＩＵに送りそのＩＵで処理してもらうコストの差で決まるのであるが、スケジューリングのこと迄考えると、それほど単純には決めることが出来ない。

負荷分散は全て自動的にするのか、あるいは、ある程度は、プログラマやコンパイラの指示も付け加えるべきなのかということは議論の分かれる点であるが、重要なことである。ここでは、プログラマやコンパイラが付け加えた指示があれば、それを尊重し、そのような指示がなくてもそれなりの性能を出すということをポリシーとする。

以上の点をまとめると、負荷分散のパラメータとしては、以下の３つが挙げられる。

①負荷量を何で評価するか。
②ＧＦを分配する時の閾値
③プログラマやコンパイラによる負荷分散の指示（但し、具体的にどの様な指示を与えるかについては、後述する。）

６．シミュレーションの仮定

シミュレーションをする際に以下のことを仮定した。

①ＵＰはＰＩＥの試作ＵＰとほぼ同程度の機能のハードウェアを仮定し、マイクロ命令レベルまでシミュレートする。
②共有メモリ網は、４入力４出力クロスバーのスイッチング・ユニット（ＳＵ）を用いたオメガ網とし、１段あたりの接続時間は、２クロックとする。
③ＳＭ、ＭＭは、外部からのネットワークとローカルなＵＰから同時読みだし可能であり、書き込みを行う場合には、ロックがかけられる。
④ＩＵ台数は１６台、６４台とする。

シミュレータはＵＮＩＸ上のＣ言語で実装し、約１１０００行である。

シミュレーションに用いた例題は、以下のプログラムである。

・nreverse　（リストの反転）
・qsort　（リストのソート）
・primes　（素数を求める）
・perm　（リストの順番を入れ換えた結果を全て求める）

７．シミュレーション結果

７．１　ボディ部のゴール・リテラルの順番

ＧＨＣやＦＬＥＮＧの様なCommitted choice言語に於いては、ボディ部のゴール・リテラルは並列に実行されることを仮定しているので、本来は、その順番は処理に影響を与えないはずである。しかし、ＰＩＥは、新ゴールを生成するリダクションという処理を逐次的に行うので、その順番は処理に大きな影響を与える。前述の様にサスペンドを減らし、スムーズに実行が行われる様に負荷分散をするためには、データの流れる通りに実行されることが望ましいと考えられる。

図３にnreverseのＦＬＥＮＧプログラムを示す。このプログラムは、nrevの実行結果が返って来ない限りappendはすぐサスペンドしてしまうので、ボディ部の順番としては、nrev、appendの順にして、少しでも早くnrevの実行が行われるようにすることが望ましい。また、nreverseの場合は、最初はappendはサスペンドしてしまい、nrevだけが走り、それが終わって初めてサスペンドしていたappendが動き出す。そして、nrevが走っている間は、並列度が全く無く、その後で、appendが最高でリストの長さ分の並列度で動き始める。従って、理想的な形としては、図４の様に、各ＩＵがパイプ・ライン状に動いてなるべく早くnrevの実行を終了させ、appendの実行に移ることが望ましい。

```
append([H|X],Y,Z):-append(X,Y,ZZ),unify(_,[H|ZZ],Z).
append([],X,Y):-unify(_,X,Y).

nreverse([H|X],R):-nreverse(X,XX),append(XX,[H],R).
nreverse([],R):-unify(_,R,[]).
```

図３　ＦＬＥＮＧプログラムの例（nreverse）

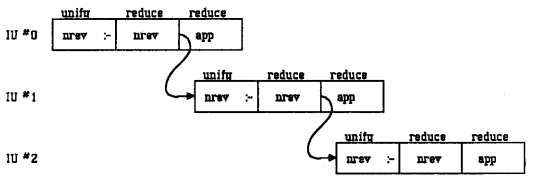


図４　nreverseの理想的実行

７．２　負荷量の評価

負荷量の評価基準として、アクティブＧＦの数とサスペンドＧＦの数を用いるということは前にも述べた。そして、１台のＩＵがアクティブＧＦを多く抱えているという状態は、プログラムの並列度を十分に出し切っていないということであり、あまり望ましくない。従って、アクティブＧＦの個数を重視する必要があると考えられるが、前述の様にサスペンドＧＦを無視するわけにも行かない。そこで、負荷量の評価基準として、

負荷量　＝　アクテイブＧＦ数　×　２５６
　　＋　サスペンドＧＦ数

という式を用いることにした。

７．３　負荷分散の閾値

リダクションによって新しく生成されたＧＦを他のＩＵに送るか、それとも自分で処理するかという判断をするために負荷分散に閾値を設けるということを前述したが、これは、他のＩＵへＧＦを送ってそのＩＵで実行してもらうコストと自分で実行するコストの差で決まる。しかし、ＰＩＥのＤＮは、接続時間、スループット共に高速であり、通信コストは低い。よって、他のＩＵに実行してもらうのと自分で実行するのとでは、あまりコストの差が無い。

従って、閾値はほとんどゼロに設定することが望ましいが、他のＩＵとのＧＦの差が一つしかない時にＧＦの無駄な転送キャッチ・ボールをしてしまうおそれがあるので、アクティブなＧＦの数の差が１以下の場合は他のＩＵにＧＦを送りつけないことを原則とする。

７．４　プログラマやコンパイラによる負荷分散の指示

負荷分散の制御をあまり複雑にすると、そのためのオーバーヘッドが無視できないものとなるので、負荷分散を全く自動のみで行うことには、少し無理があると考えられる。そこで、ある程度は、プログラマやコンパイラがプログラムに書き加えた付加情報に従って、負荷分散を制御するということが考えられる。特に、急激な負荷の変動が起きた時（ＧＦの数が急に増えるのではなく、サスペンドＧＦが急に次々とアクティブになる場合等）には、全自動の負荷分散方式では分散に乱れが生じる事がある。そのような場合に負荷分散をスムーズに行うために、付加情報は有効である。また、サスペンド、アクティベイトが頻繁に起きるプログラム（素数生成etc.）では、付加情報によってサスペンドの頻度を減らすことにより、実行をスムーズにすることが出来る。

７．４．１　負荷分散の付加情報

具体的には、付加情報として次の事項が考えられる。

①ＧＦを他のＩＵへは送らず、自分のＭＭに残す。

②ＤＮからの負荷情報と自分の負荷量を比較し閾値を見て、他のＩＵへ送るかどうかを判断する。（全自動の負荷分散方式と同じ）

③負荷量の比較、閾値との比較等はせずに、ＧＦをＤＮへ放出する。

７．４．２　付加情報を与える目安

実際のＦＬＥＮＧプログラムにどの様な基準で前述の３種の付加情報を割り当てるかについてであるが、これは以下の方針に従うことにする。

①ボディ部の各リテラルはデータの流れ通りに並んでいると仮定し、ジェネレータあるいはサスペンドを解除することが可能と考えられるシステム述語（unify等）等は、なるべく他のＩＵへ送ることが望ましい（③の指示を与える）

②ボディ部の最後尾のリテラルは、システム述語ならば自分のＭＭに残す（①の指示）がそうでない場合は、②の指示にする。その理由は、自分のＭＭに残すという指示は、かなりきびしいものであり、使い方を誤ると１台のＩＵへの負荷の集中が起こりかねず、新しいゴールを生

成することが無い、システム述語に限った方が良いと考えられたからである。また、unify述語は、SMへのアクセスが多いのでその点を考慮しても、やはりローカルMMに残すべきである。

③GHCのガード部をFLENGに書き直した場合は、そのためにシステム述語（比較等の）が付加されることになるが、これは、真っ先に実行してもらいたいものであるから、やはり、③の指示を与える。

④リテラルの性質のよくわからないものに対しては、従来通りの指示（②の指示）を与える。

7．4．3　付加情報の効果

自動負荷分散ネットワークの効果、付加情報の効果を比較するために、シミュレーション結果を図5～7に示す。図5は、比較のためにランダムに負荷分散を行った場合についてのシミュレーション結果である。図6は自動負荷分散ネットワークの機能のみを使って負荷分散を行った場合のシミュレーション結果である。また図7は、ユーザによる付加情報を考慮して負荷分散を行った場合のシミュレーション結果である。図5と図6を比較すると、図6の方が実行速度が一様に速く、Committed choice言語に於いても自動負荷分散ネットワークが有効であることがわかる。また、図6と図7を比較すると、nrev,primes は図7の方が実行が早く終了している。これは、稼働プロセッサ数の時間変化を見てもわかるように、平均並列度及び並列度の立ち上がりの両方が向上したためであると考えられる。特に、nrevの場合は、付加情報を与えた事によって、appendの立ち上がりがスムーズになっている事がわかる。また、permに対しては、付加情報の効果がほとんど無いが、これは、付加情報が主にサスペンドを減らすことを目的としているので、permの様にサスペンドをしないプログラムに対しては意味が無いためと考えられる。

8．おわりに

以上のシミュレーション結果より、ユーザによる付加情報を用いて負荷分散を行う場合については以下のことがわかった。

・付加情報を用いることにより、ゴールがサスペンドする頻度、時間等を減少させることが出来るようになり、実行速度が速くなる。

・サスペンドを起こさないような例題においては、あまり付加情報の効果は見られない。

・Committed choice言語では、パイプライン並列で各プロセスが動いていることが多いので、そのパイプをなるべく乱さないような付加情報が効果的である。

〈参考文献〉

[1] Moto-oka, T., Tanaka, H., et al, "The Architecture of a Parallel Inference Engine -PIE-", FGCS '84, ICOT.

[2] 山内、小池、野田、田中、"高並列推論エンジン実験環境ＰＩＥＥＥ　－自動負荷分散ネットワーク－"、第34回情処全大、4P-3、1987.

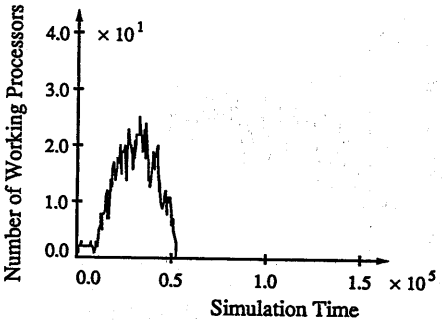
[3] 坂井、小池、田中、元岡、"動的負荷分散を行う相互結合網の構成"、情処論、Vol.27、No.5、1986.

[4] 垂井、田中、"並列推論マシンにおけるストリーム並列言語の実行方式の評価"、Proc. of Logic Programming Conference '87、ICOT、1987.

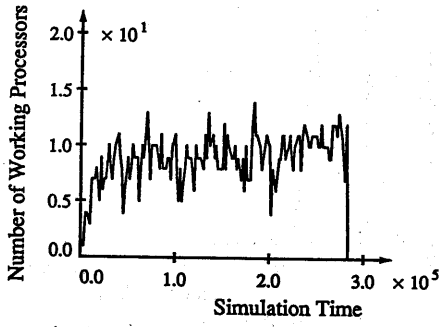
[5] 山内、田中、"ＰＩＥにおけるストリーム並列言語を指向したネットワーク構成法"、第35回情処全大、3C-8、1987.

[6] Nilsson, M., Tanaka, H.,"- FLENG Prolog - The Language which turns Supercomputers into Parallel Prolog Machines ",The Logic Programming Conference '86, ICOT

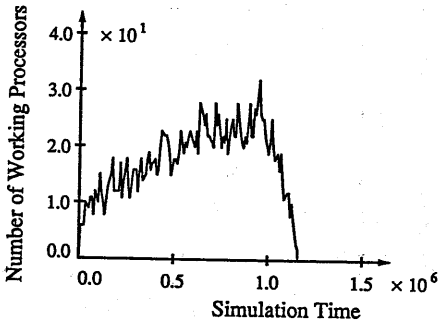
[7] 宮崎、瀧、"Multi-Psi におけるFlat GHCの実行方式"、Proc. of Logic Programming Conference '86、ICOT、1986.



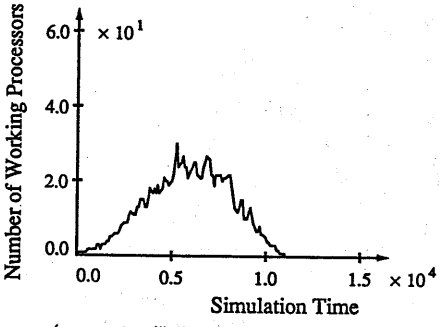
(nrev-60,推論ユニット６４台)



(qsort-256,推論ユニット１６台)

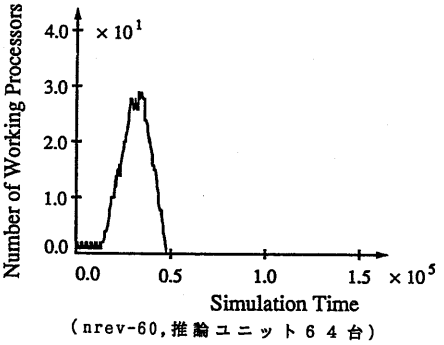


(primes-1000,推論ユニット６４台)

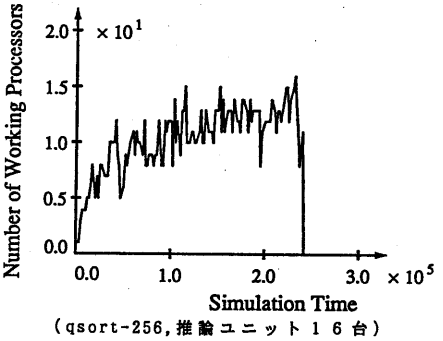


(perm-4, 推論ユニット６４台)

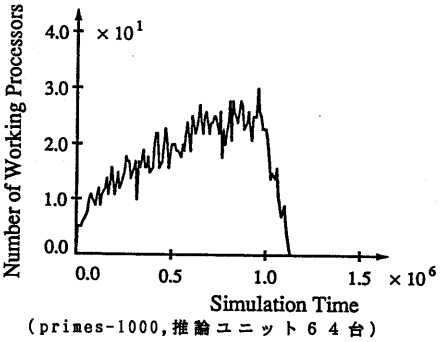
図５　ランダム負荷分散の場合のシミュレーション結果



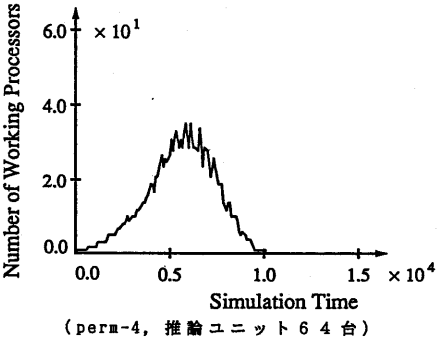
(nrev-60,推論ユニット６４台)



(qsort-256,推論ユニット１６台)

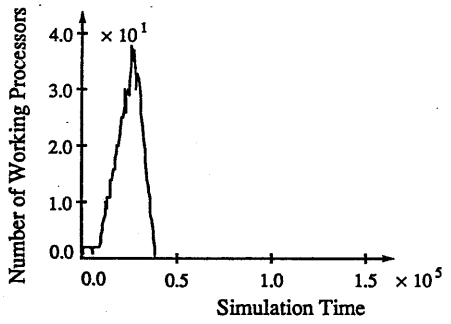


(primes-1000,推論ユニット６４台)

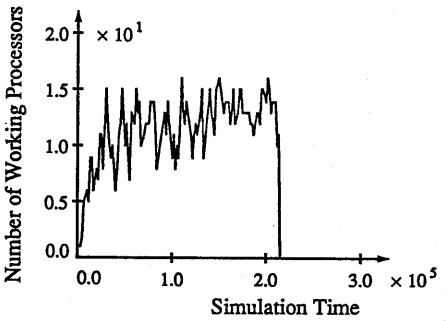


(perm-4, 推論ユニット６４台)

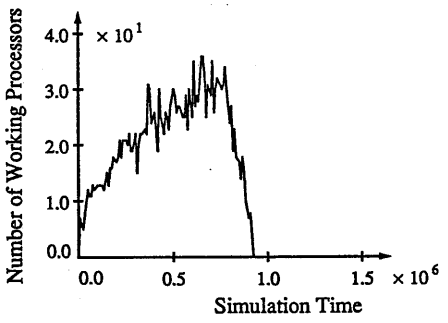
図６　全自動の負荷分散のみの場合のシミュレーション結果



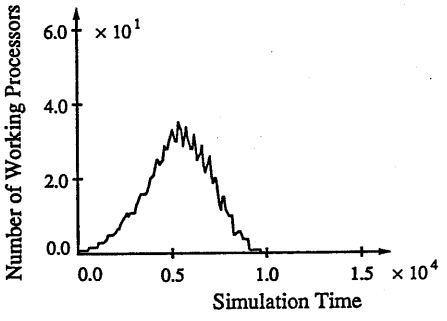
(nrev-60,推論ユニット６４台)



(qsort-256,推論ユニット１６台)



(primes-1000,推論ユニット６４台)



(perm-4,推論ユニット６４台)

図７　付加情報を用いて負荷分散を行った場合