(13)

EC83－39

高並列推論エンジンＰＩＥ～並列度のシミュレーションとその評価

丸山　勉・湯原雅信・相田　仁
後藤厚宏・田中英彦・元岡　達
（東　大）

１９８３年１２月１６日

社団法人　電　子　通　信　学　会

EC83－39

高並列推論エンジンＰＩＥ
～ 並列度のシミュレーションとその評価 ～

A HIGHLY PARALLEL INFERENCE ENGINE : PIE

-- An Evaluation of Effective Parallelism and Performance --

丸山　勉，　湯原　雅信，　相田　仁，　後藤　厚宏，　田中　英彦，　元岡　達
T. Maruyama, M. Yuhara, H. Aida, A. Goto, H. Tanaka, T. Moto-oka

東京大学　　工学部
Faculty of Engineering,　University of TOKYO

1．　はじめに

我々はゴール書き換えモデル［１］に基づいた高並列推論エンジンＰＩＥ（Parallel Inference Engine）の設計を進めている。現在、ＰＩＥの基本処理要素である単一化プロセッサを試作中であり［２］、その設計データを基にＰＩＥにおける並列度のシミュレーションを行なったので、その結果について報告する。

2．　ＰＩＥ第一次モデルの概要

ＰＩＥ第一次モデルの主な特徴は次の２つである。
（１）ＰＩＥの処理単位であるゴールフレームの論理的な独立性を充分に活かすことにより、高並列処理が実現できる。
（２）ゴールフレームの導出機構（論理）とアクティビティ制御機構（制御）をアーキテクチャ上において分離することにより、柔軟な問題解決を進めることができる。

現在設計を進めているＰＩＥ第一次モデルの全体像を図１に示す。

主なモジュールの機能は以下の通りである。

▶ 定義節メモリ（ＤＭ）
定義節を保持するメモリであり、単一化が行なわれるリテラルの引数の状況によって動的な定義節の絞り込みを行なう。各定義節メモリがすべての定義節を持つことを考えている。

▶ 単一化プロセッサ（ＵＰ）
ゴールフレームの対象リテラルとそれに対応する定義節のすべての選択肢との間で単一化および縮退を行なう［２］。新しく導出したゴールフレームを自分と対になったメモリモジュールへ、またはゴールフレーム分配網を経て他のメモリモジュールへ送り出す。

▶ メモリモジュール（ＭＭ）
ゴールフレームを蓄えるメモリである。

▶ アクティビティ・コントローラ（ＡＣ）
ゴールフレーム間の関係を表わす関係木を保持し、ＡＣ間でコマンド［３］を送受することによって関係木の操作を行ないゴールフレームを管理する。

▶ ゴールフレーム分配網（Distribution Network）
ゴールフレームの分配に用いられる網である。

▶ コマンド通信網（Command Network）
ＡＣ間でのコマンドの送受に用いられる網である。

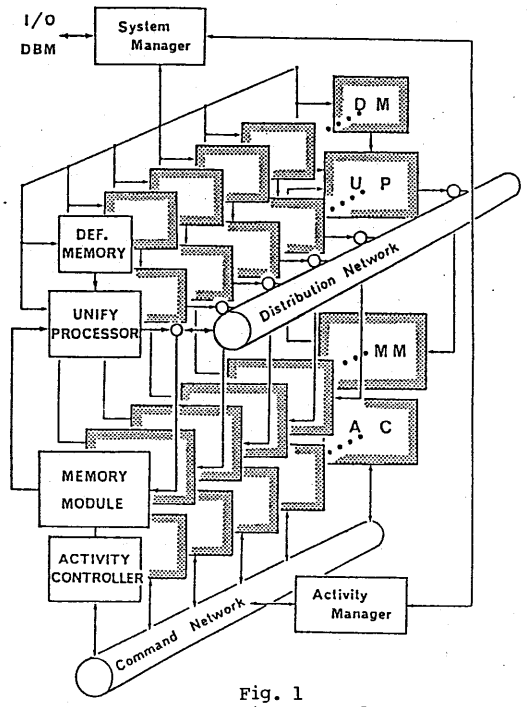

Fig. 1
Global Architecture of PIE

3. シミュレータ

3.1 プロセスコントロール機能

本シミュレータはUnix 上のC言語で記述した。SIMULAにみられるようなプロセスコントロールの機能をCから呼び出せるアセンブラサブルーチンの形でサポートしたので、プロセスのコンカレントなシミュレーションを比較的容易に行なうことができる[4]。

プロセスは、そのプロセスが実行されるべき時刻に応じてEvent List に挿入され、Event List の先頭のプロセスから順次処理される。

シミュレータに実装したプロセスコントロール関数の代表的なものを以下に示す。

- New (function, stacksize) : 関数 function を実行するプロセスを生成する。生成されたプロセスは stacksize で指定された大きさのスタックを持つ。
- Passivate () : プロセスをEvent List からはずし passive 状態にする。
- Activate (process) : passive 状態にあるプロセス processを active 状態にする。
- Hold (time) : time 時間後にそのプロセスの実行を再開する。プロセスはEvent List の time 時間後に相当する位置に挿入される。
- Cancel (process) : Event List につながれているプロセス process の実行を中断し、passive 状態にする。

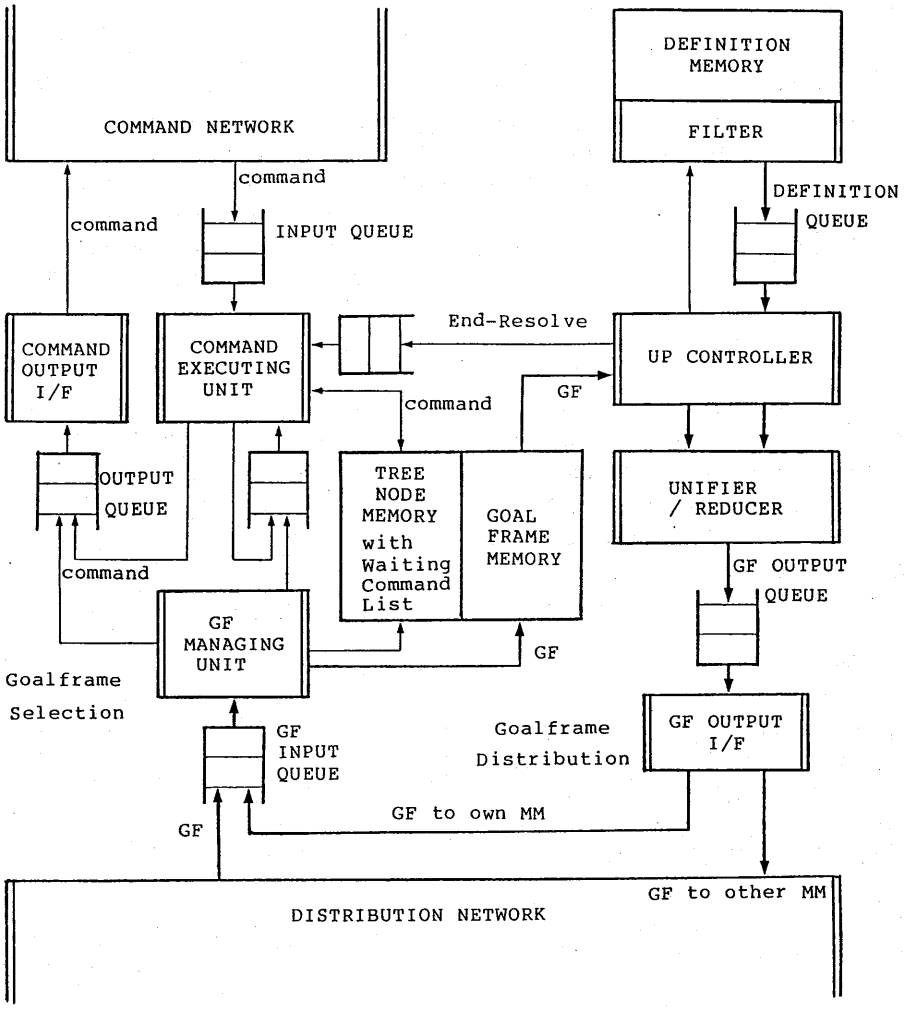

Fig. 2
Simulation Model

3．2　シミュレーションモデル

ＰＩＥの第一次モデルの１ユニット分のシミュレーションモデルを図２に示す。

図２において、左右の枠が二重になっているものがシミュレータ上でのプロセスである。これらのプロセスが、シミュレータ上でコンカレントに実行される。図２に示した各モジュールと、第２章で述べたＰＩＥの各モジュールとの対応を表１に示す。

シミュレータ上のプロセスは、それぞれ次のような役割りを果たす。

▶ Filter

UPからリテラルの引数に関する情報を受け取り、定義節の絞り込みを行なう。

▶ UP　Controller

ゴールフレームのヘッダ部処理等の単一化に必要な準備、およびシミュレータ上のプロセスFilter と　Unifier／Reducerの制御を行なう。

▶ Unifier／Reducer

入力ゴールフレームとそれに対応する定義節との間で単一化および縮退を行なう。図２に示したモデルでは、ゴールフレームに対応する定義節が複数ある場合には、それらは逐次的に処理される。

▶ GF　Output　Interface

Unifier／Reducerからゴールフレームを受け取りDistribution Network内のプロセスにゴールフレームを渡す。

▶ Distribution Network

ゴールフレームを運ぶプロセスの集合であり、新しいゴールフレームが導出されるごとに、ネットワーク中の１つのプロセスがそのゴールフレームを受けとり、指定された遅延時間後に目的地にそのゴールフレームを送り届ける。

▶ GF　Managing Unit

ゴールフレームを受けとり、その優先度に応じて　Goal Frame　Memory に格納するとともに、そのゴールフレームに対応する関係木のノードをTree Node Memory の中につくる。

▶ Command　Execution　Unit

コマンドを実行する。コマンドに応じてTree Node Memory 中のノードを操作し、ゴールフレームの管理をおこなう。

▶ Command　Output　Interface

Command　Network　内のプロセスにコマンドを渡す。

▶ Command　Network

Distribution Networkと同様にプロセスの集合でありコマンドの転送を行なう。

尚、シミュレータに於いてQueue長はすべて無限大を仮定している。

3．3　組み込み述語の連続実行

UPが、その結果が１通りである組み込み述語を実行する場合には、縮退をせずに続けて次のリテラルの単一化を行なうことができる。図３は、書き換えの結果生ずる組み込み述語を連続して処理する例を示している。さらに、

（１）定義節の絞り込みによって、ただ１つの定義節が選ばれたとき、

（２）複数選ばれた定義節の中で最後の定義節との単一化を行なうとき、

などの場合にも次の単一化を引き続き行なうことができるが、今回のシミュレータには実装していない。

Table 1.　Correspondence between Modules in Fig.1 and Processes in Fig.2

ＰＩＥ第１次モデル	Simulation Model
定義節メモリ	FILTER DEFINITION MEMORY
単一化プロセッサ	UP CONTROLLER UNIFIER/REDUCER GF OUTPUT I/F
メモリモジュール	GOAL FRAME MEMORY TREE NODE MEMORY
	GF MANAGING UNIT
アクティビティ・コントローラ	COMMAND EXECUTION UNIT COMMAND OUTPUT I/F
ゴールフレーム分配網	DISTRIBUTION NETWORK
コマンド通信網	COMMAND NETWORK

3．4　シミュレータにおけるプロセスの実行

シミュレータにおける各プロセスの動きを図4に示す。

UP Controller はゴールフレームの単一化の準備を行なう。準備が終了したならば、Filter を起動し、リテラル名とその引数の状況をFilter に渡す。Filter はその情報によって定義節の絞り込みを行ない、それを通過した定義節をUP Controller に送る。

UP Controller は定義節を受け取ったならば入力ゴールフレームと定義節をUnifier／Reducerに送り、単一化および縮退を実行させる。今回のモデルでは、単一化プロサッサ中のUnifier／Reducerが1台なので、複数定義節がある間、UP Controller はUnifier／Reducerが処理を終了するのを待ち、次々と単一化および縮退を実行させる。定義節がなくなったら、UP Controller はUnifier／Reducer の終了を待たずに、次のゴールフレームの単一化の準備を始める。

定義節の絞り込みにより、1つの定義節も選ばれなかったときには、UP Controller は直ちに次のゴールフレームの処理に移る。

ゴールフレームが十分多い状況下では、単一化の準備と定義節の絞り込みは、上記のように単一化および縮退と並行して動作するため、全体の処理時間にはあまり影響しない。

3．5　シミュレーションクロック

今回のシミュレーションでは、シミュレーションクロックとして、現在試作を進めている単一化プロセッサ［2］のマイクロプログラムのクロックを想定している。試作単一化プロセッサのマイクロプログラム用シミュレータを用いた測定から、単一化および縮退にかかるマイクロステップ数は、両者ともおおよそ、1セル当り5～8ステップであることがわかった。そこで、今回のシミュレーションでは、その値に基づいて各プロセスの処理時間を表2のように決めた。

一般に、組み込み述語の実行速度は普通のリテラルの単一化に比べて速いと考えられる。しかし、最悪の場合を考慮して単一化と同じ処理時間がかかるものとした。

ヘッダ部の処理等単一化の準備に必要な時間、および定義節の絞り込みに必要な時間については、まだハードウェアの具体的な検討が進んでいないので、適当な値を用いた。

Distribution Networkについてもハードウェアの詳細は検討中であるで、その遅延時間を表2のように定めた。ただし、宛先の衝突による影響は考慮していない。

表に実行時間を示さなかったプロセスは時間0で処理を行なうものとしている。

Table 3.　Delay Time of Distribution Network

number of UPs	delay time
16	2 X Reduction Time(Tr)
64	3 X Tr
128	4 X Tr
256	4 X Tr

```
<input Goal Frame>
?-nodiag(2,1,1),plus(1,1,V0),safe(2,[],V0),queens([3,4,5,6],[2,1],V1),print(V1).
<Definition>
  nodiag(U,P,N):-plus(P,N,T1),minus(P,N,T2),neq(U,T1),neq(U,T2).
<New Goal Frame>
?-safe(4,[],1),queens([1,2,3,5,6],[4],V0),print(V0).
```

Fig. 3
An Example of Continuous Execution

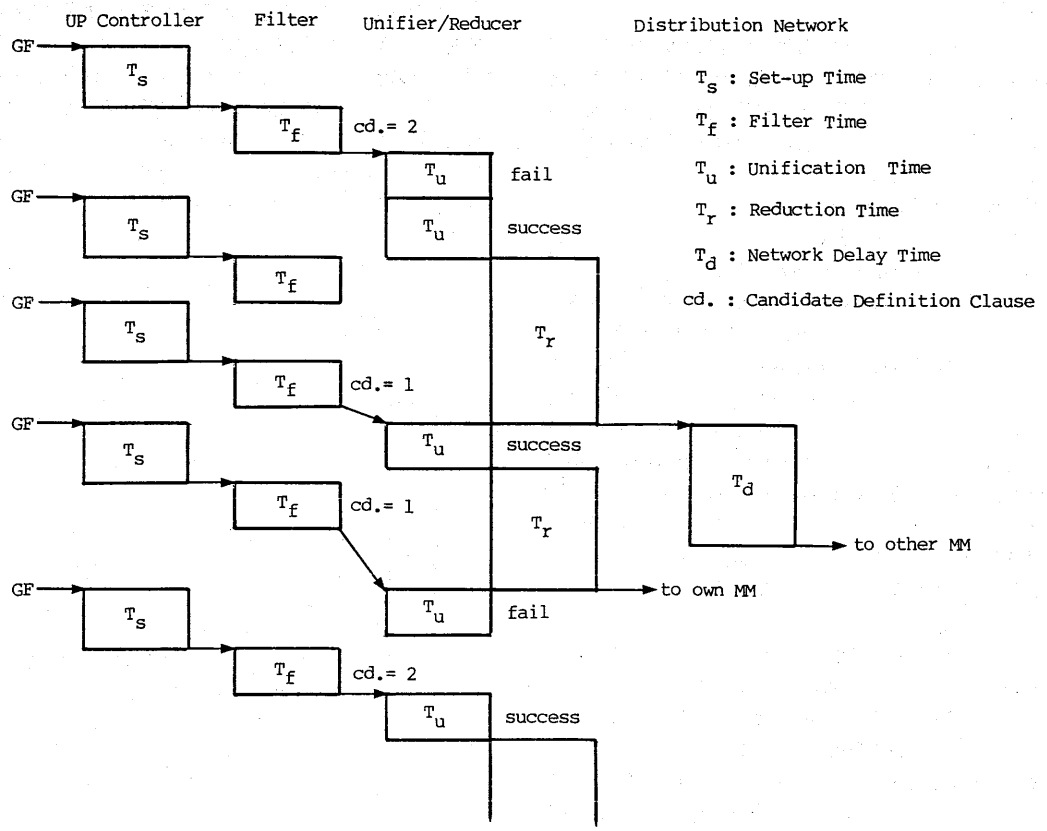

Fig. 4
An Execution Feature of Processes

Table 2. Simulation Time Setting in Simulator

	処理時間	実行するプロセス
単一化の準備	100 ＊ p 1	UP Controller
定義節の絞り込み	50 ＊ p 2	Filter
単一化	単一化されたセル数 ＊ 7	Unifier／Reducer
組み込み述語の実行	計算されたセル数 ＊ 7	Unifier／Reducer
縮退	縮退されたセル数 ＊ 7	Unifier／Reducer
ネットワークの遅延	・他のメモリモジュールへ ゴールフレーム長 ＊ 縮退時間 ＊ p 3 ・自分自身のメモリモジュールへ 0	Distribution Network

ただし、p 1, p2, p3はシミュレーションのパラメータである。

4　シミュレーション結果

ＰＩＥ第一次モデルでは、原則的にOR関係にある多数のゴールフレームの並列処理を、多数台のUPを用いて実現する。従って、ＰＩＥの処理速度はUPの台数およびその稼働率に強く依存する。UPの稼働率に影響を与える要因として以下のものが考えられる。

（１）問題の持つ並列性とUP台数
（２）ゴールフレームの分配ストラテジ
（３）Distribution Networkの遅延時間

UP台数に比べて問題の並列性が大きく、すでに十分多くのゴールフレームが存在する状況では、上記の（２），（３）ともそれほど処理速度に影響を与えないと考えられる。これは、各メモリモジュール内のゴールフレームの数が十分大きければ、メモリモジュール内のすべてのゴールフレームの処理を行なう時間に比べネットワークの遅延時間のほうが短くなるためである。さらにゴールフレームの分配ストラテジとしても、各UPの入力ゴールフレームがなくならない程度の分散を行なえば十分であるからである。

以下に述べるシミュレーションでは、

◆UP台数
◆ゴールフレームの分散ストラテジ
◆Distribution Network　の遅延時間
◆単一化の準備および定義節の絞り込みの処理時間
◆ゴールフレームの選択ストラテジ

を変化させてその影響について測定を行なった。ただし、各々のシミュレーションにおいては他の事項を次のように設定した。

① ゴールフレームはメモリモジュールに到着した順に処理される。
② ゴールフレームの分配方法はEmpty Self（4．2参照）である。
③ Distribution Networkの遅延時間はおおよそUP台数の対数に比例するものとし、表3に示した時間とする。
④ 単一化の準備および定義節の絞り込みの時間はそれぞれ１００，５０クロックとする。

また、ネットワークの負荷による遅延時間の変化は考慮していない。

シミュレーションに用いたプログラムは［6-queens ］，［8-queens ］，［ＬＬ2Ｐ］（覆面算）［５］である。

以後の説明ではUPとはシミュレータのUP Controller，Filter およびUnifier／Reducerのことをさすものとし、メモリモジュールとはGoal Frame Memory をさすものとする。

4．1　UPの台数と稼動率

UP台数を4～２５６まで変えたときの、処理速度および達成される最大の並列度についてのシミュレーション結果を図5～8に示す。図5，6はそれぞれ、UP台数に対するスループット、および達成される最大の並列度であり、図7，8はUP２５６台のときの、各シミュレーション時刻におけるUP Controller，Filter，Unifier／Reducerの稼働数、およびメモリモジュール内のゴールフレームの総数である。

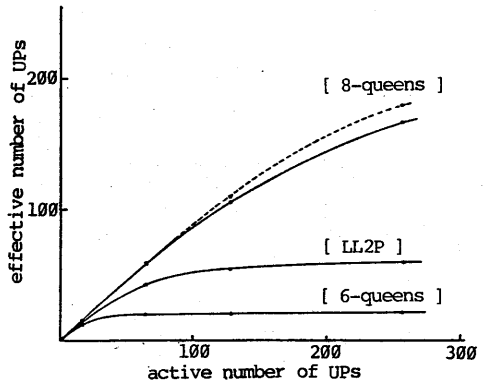

Fig. 5
Throughput

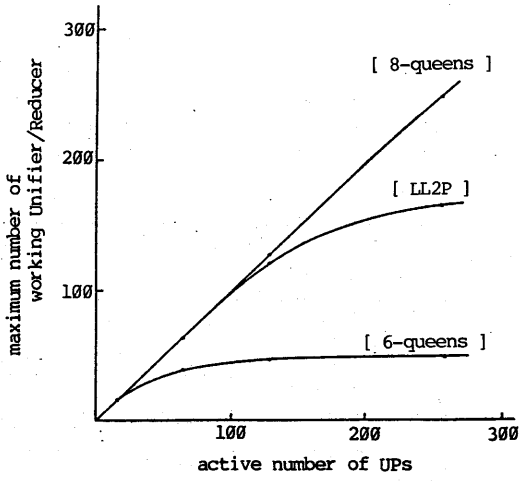

Fig. 6
Maximum Effective Number of UPs

図5にみられるように、UP台数の増加に伴って、次第にスループットが飽和してゆく。このスループットの飽和の原因としては、一般に次の2つが考えられる。

① UP台数に対する問題の並列性の不足

② UP台数の増加に伴なうネットワークの遅延時間の増加

図6より、[6-queens], [LL2P]におけるスループットの飽和の主な原因は、問題の並列性がUP台数に比較して小さいことにより、UPの稼動数が制限されてしまうためであることが解る。[8-queens]の場合は、UPの稼働率は一見高いようであるが、UP台数256台のときのメモリモジュール内のゴールフレーム数は、総数で最大100程度であり、平均は1に満たない(図8)。その結果、単一化の準備, 定義節の絞り込み, および単一化/縮退の各処理が、逐次化されてしまいスループットが低下してしまう。このことは図5の破線に示したようにネットワークの遅延を零にしても、それほどネットワークのスループットが向上しないことからも確認される。従って、もっと大きな並列性を持つ問題ならば、スループットはその並列性に応じて伸びると考えられる。

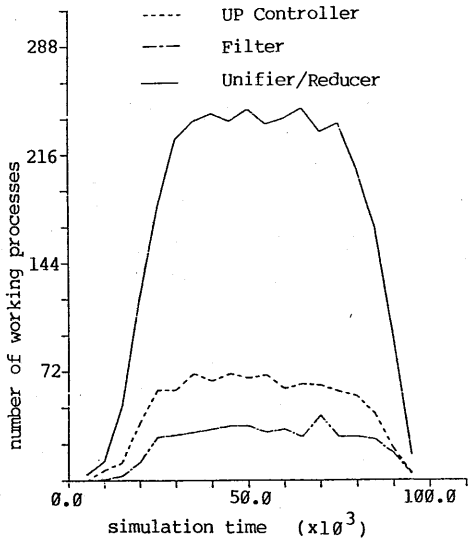

Fig. 7
Number of Working Processes
[8-queens] UP = 256

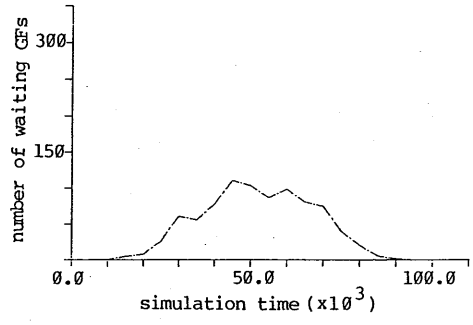

Fig. 8
Effect of Network Delay
[8-queens] UP = 256

以上のことにより、PIEにおいてより高速な処理を実現するためには、問題の持つ並列性が十分に引き出されるまでの間に、

① ネットワークの高速な転送

② UPを最も効率的に使用できるようなゴールフレームの分配

を行なうことが重要であるということがわかる。

4.2 ゴールフレームの分配ストラテジ

PIEにおいてゴールフレーム間並列性を有効に利用するには、UPを最も効率的に使用できるようなゴールフレームの分配を行なうことが重要である。ゴールフレームの分配を行なう場合に、いたずらに他のUPにゴールフレームを送出するのでは、ネットワークの遅延時間の分だけ処理速度が落ちることになる。従って、導出したゴールフレームを自分自身のメモリモジュールへ戻すか、他のメモリモジュールへ送り出すかを、いかにして決定するかが問題となる。そこで、以下に示すような分配ストラテジについてシミュレーションを行なってみた。

(1) First Self

入力ゴールフレームに対する最初の導出ゴールフレームを自分と対になったメモリモジュールに戻し、それ以降の導出ゴールフレームは他のメモリモジュールにランダムに分配する。常に導出したゴールフレームの1つを自分と対になったメモリモジュールに戻すことにより、UPの効率的な使用を行なうことができると考えられる。

(2) Empty Self

ゴールフレームの導出を行なった際、自分と対のメモリモジュールが空ならば、自分と対のメモリモジュールへ、そうでなければ他のメモリモジュールへ送る。

(3) ランダム

全くランダムに分配する。(1), (2)と比べてUPの使用効率、ネットワークの遅延共に不利であると考えられる。(1), (2)との比較の意味でシミュレーションを行なった。

シミュレーション結果を図9～12に示す。

図9, 10はそれぞれ、UP台数を256台としたときのシミュレーション時刻におけるUnifier/Reducerの稼動数、

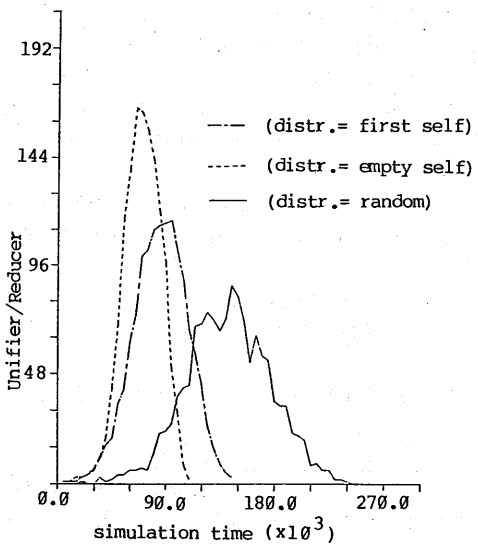

Fig. 9
Effect of Distribution Strategy on Working UPs
[LL2P] UP = 256

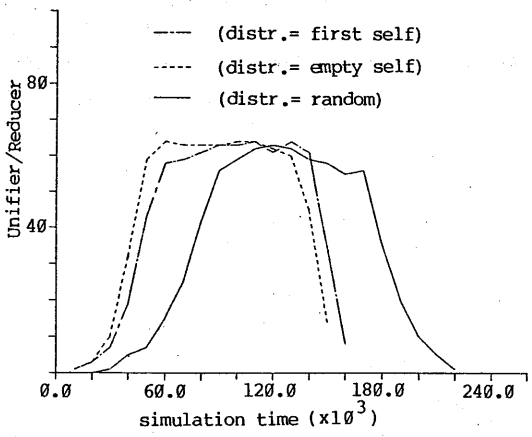

Fig. 11
Effect of Distribution Strategy on Working UPs
[LL2P] UP = 64

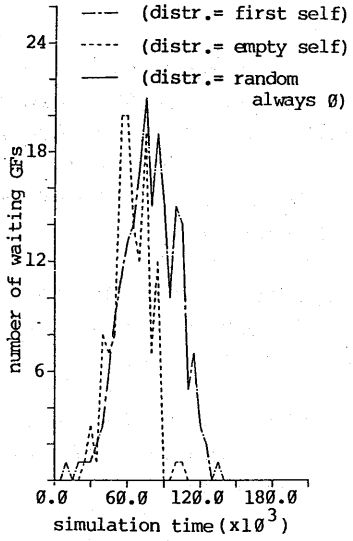

Fig. 10
Effect of Distribution Strategy on Waiting GFs
[LL2P] UP = 256

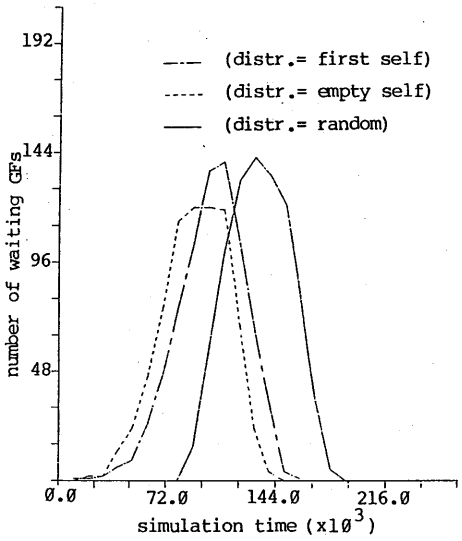

Fig. 12
Effect of Distribution Strategy on Waiting GFs
[LL2P] UP = 64

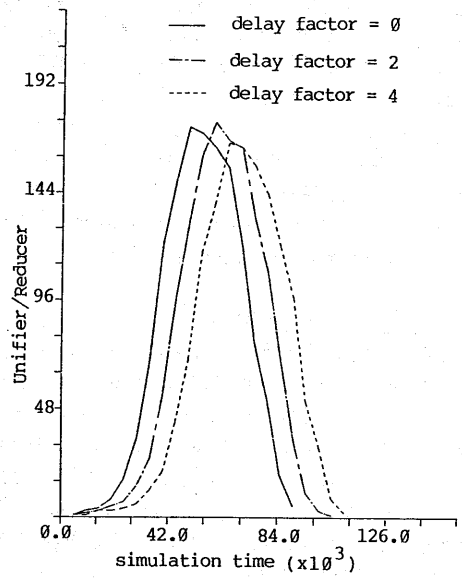

Fig. 13
Effect of Network Delay on Working UPs
[LL2P] UP = 256

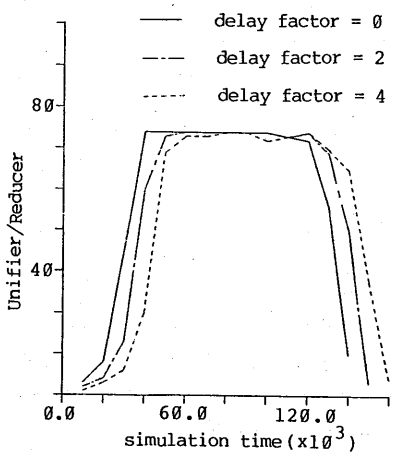

Fig. 15
Effect of Network Delay on Working UPs
[LL2P] UP = 64

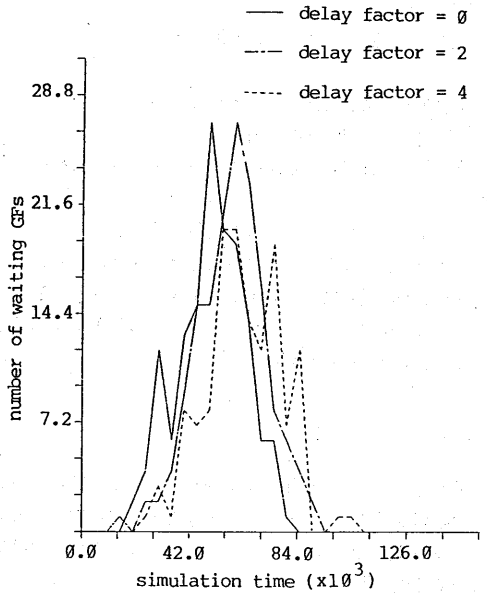

Fig. 14
Effect of Network Delay on Waiting GFs
[LL2P] UP = 256

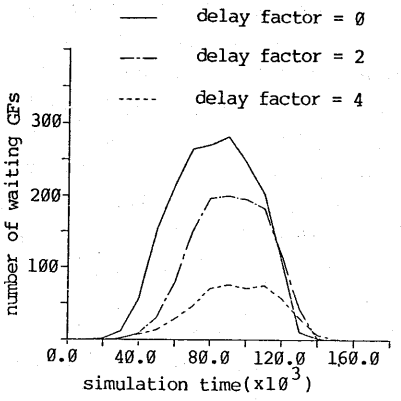

Fig. 16
Effect of Network Delay on Waiting GFs
[LL2P] UP = 64

およびメモリモジュール内のゴールフレームの総数を示している。図11，12はUP台数を64台としたときのものである。UP台数が256台のときは、UP台数に比べて問題のもつ並列性が小さい場合に、UP台数が64台のときはその逆の場合に相当している。

図9，11から解るように、Empty Self が処理時間が最も短い。さらに図9からは、Empty Self が最も高いUnifier／Reducerの稼働率を与えることが解る。しかし、Empty Self では、UP内部の処理はかならずしも効率的であるとはいえない。一方、First Self では、自分と対のメモリモジュール内に常に1つのゴールフレームを残すので、UP内部の処理は効率的に行なわれるが、残しすぎることによってゴールフレームの偏りを生じ、処理速度が落ちてしまう。以上のことより、今後、UP内部の効率的な利用を考慮しつつ均等なゴールフレームの分配を行なうことができるストラテジを探ることが重要であることがわかる。

さらに、図11より、UP台数に比べてゴールフレーム数の方が多い状態（グラフの線が平坦になっている部分）の長さは、分配ストラテジにかかわらず一定であり、処理時間はほぼその状態に達するまでの時間によって決まっていることが解る。このことは、ゴールフレーム数がUP台数に比べて少ない状況下において、ゴールフレームの効率的な分配を行なうことが重要であることを示している。

4．3 Distribution Networkの遅延時間

ネットワークによるゴールフレームの転送時間を縮退時間の0，2，4倍としたときの、Unifier／Reducerの稼動数およびメモリモジュール内のゴールフレームの総数について、UP台数を64台，256台としてシミュレーションを行なった。その結果を図13～16に示す。

図13，15から解るように、全体の処理時間はゴールフレームの分配ストラテジの場合と異なり、UP台数の影響は少なく、ゴールフレーム数がある程度まで増加するまでの時間（UPの稼動数がUP台数が256台のときでも20台程度になるまでの時間）によって決まっていることがわかる。これは、分散ストラテジがEmpty Self であることによるものと考えられる。Empty Self では、一度動き始めたUPはそのまま動き続ける。ゴールフレーム数が増加し、ある程度の数のUPが動き出してしまえば、それらによる処理速度に影響を与えるほどネットワークによる遅延時間は大きくはないからである。

この結果より、ネットワークの遅延はゴールフレームが増加し始めるときに大きく影響し、それ以降はあまり関係しないことが解る。4．2で問題の並列性が十分引き出されるまでの間のネットワークの高速な転送が重要であることを述べたが、以上の結果より、ネットワーク遅延についてもその範囲としてゴールフレームが増加し始めるわずかの範囲を考えれば十分であることが解る。

4．4 単一化の準備および定義節の絞り込みの実行時間

ここでは、単一化の準備および定義節の絞り込みにかかる時間に関する評価結果について述べる。単一化の準備にかかる時間、および定義節の絞り込みにかかる時間を変えたときのシミュレーション結果を図17に示す。UP台数は32台とした。この結果より、問題の並列度が十分に大きい状況下では、これらの処理時間はほとんど影響しないことがわかる。ただし、並列度が少ない状況下ではUP内部の処理が逐次化されるので、その分だけ処理速度は低下する。

4．5 ゴールフレームの選択ストラテジ

問題の持つ並列度が十分に大きい状況では、メモリモジュールのオーバーフローを回避する手段が必要である。また、全解探索ではなく、1つの解をより早く見つけることも重要である。それらに対する1つの解決方法として、メモリモジュールからのゴールフレームの選択ストラテジ［5］がある。文献［5］において、すでにゴールフレームの選択ストラテジの特性について簡単な評価結果を報告した。今回のシミュレータでは、時間管理を導入し、またゴールプールを多数台のMMに分割したため、より実際的である。

選択ストラテジとしては、

（1）FIFO
　メモリモジュールに到着した順番に処理を行なう。
（2）Breadth First
　探索木上で深さの浅いものを優先する。
（3）Depth First
　探索木上で深さの深いものを優先する。
（4）Proposed Method ［5］
　最も早く決着がつくと思われるゴールフレームを優先する。

の4つについてシミュレーションを行なった。

シミュレーション結果を図18に示す。図18より、ゴールフレームの選択方法を変えることによって、メモリモジュール内のゴールフレーム数も少なくなり、最初の解を早く求めることができることが解る。

5． おわりに

本報告では、PIEの第一次モデルについて、達成される最大の並列度（スループット）、およびそれに影響を及ぼすいくつかの要因についてシミュレーションを行ないその結果について評価を行なった。

評価結果を以下にまとめておく。

（1）達成される並列度

ゴールフレームの分配ストラテジに強く依存するが、その

ストラテジとしてEmpty Self のようなものをとった場合には、問題の持つ並列性の許す限りで、最大の並列度を容易に実現し得る。

（2）スループット

個々のモジュールの処理速度以外に、ＰＩＥ第一次モデルの処理速度に決定する要因は、

① 問題の並列性が引き出され始めた時点、すなわちゴールフレームが増加し始めた時点で、ＵＰを最も効率的に使用できるようなゴールフレームの分配

② 同時点までにおけるネットワークの遅延時間

の2つである。いったん並列性が引き出された後では、ゴールフレームの分配やネットワークの遅延時間はほとんど影響を与えない。

今後、今回の結果を基にしてゴールフレームの分配ストラテジおよびネットワーク構成の具体的な検討を行なってゆく予定である。また、今回シミュレーションの対象としなかった部分も含めて、より詳細なシミュレーションを進めてゆく予定である。

《参考文献》

［1］ 後藤 他，“高並列推論エンジンＰＩＥ～ゴール書き換えモデルとアーキテクチャ”，第２７回情処全大，４Ｐ－９（1982）．

［2］ 湯原 他，“高並列推論エンジンＰＩＥの単一化プロセッサと縮退アルゴリズム”，信学技報ＥＣ８３－３０（1983）．

［3］ 丸山 他，“推論向き高並列計算機システムのアクティビティ制御機構”，第２６回情処全大，４Ｎ－５（1983）．

［4］ 後藤 他，“高並列推論エンジンＰＩＥについて”，The Logic Programming Conference ’８３，ＩＣＯＴ（1983）．

［5］ 後藤 他，“高並列推論エンジンＰＩＥにおける並列処理の効率化手法について”，信学技報ＥＣ８３－９（1983）．

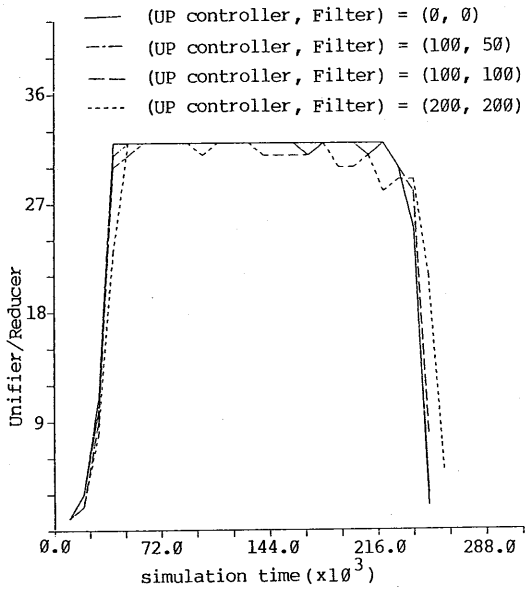

Fig. 17
Effect of Setup/Filtering time
[LL2P] UP = 32

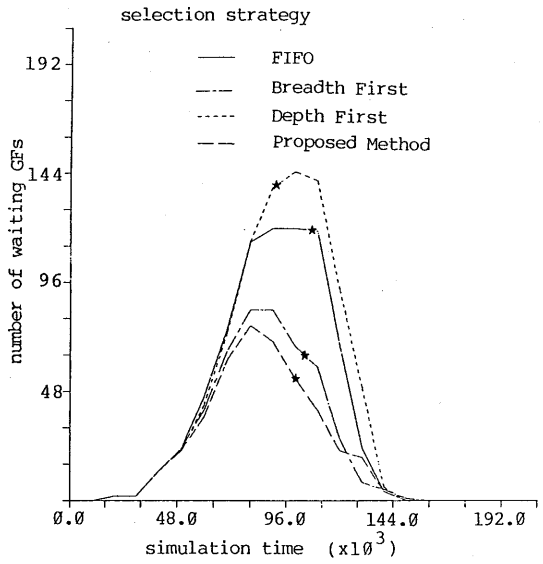

Fig. 18
Effect of Goal Frame Selection
[LL2P] UP = 64