EC83－38

高並列推論エンジンＰＩＥにおける構造データの効率的な処理方式について

An Efficient Processing Method of Structure Data on the Highly Parallel Inference Engine PIE

平田　圭二　，　相田　仁　，　後藤　厚宏　，　田中　英彦　，　元岡　達

K. Hirata , 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

高並列推論エンジンＰＩＥは、並列性の高い大きな問題に対する高速な処理を目指しており、将来の知識情報処理の中核となる。ＰＩＥの応用プログラムは主として、エキスパートシステム、CAD、知識ベースシステム、機械翻訳、自然言語処理など、人工知能、知識工学関連のものが多く、マシン稼動時には、大きな構造データが頻繁に出現する。現在、我々はＰＩＥ第１次モデルを基に並列処理の制御方式［１］、単一化プロセッサの試作［２］、構造データの効率的な処理方式［３］等、それぞれの立場から検討を進めている。一方高並列マシンにおける構造データの効率的な処理方式については、データフローマシンにおいて検討が進められている。しかし並列推論マシンにおいては一部で検討が始まったばかりであり、提案のみにとどまっている［６］。そこで本報告では、ＰＩＥ第２次モデルに向け、並列推論マシンにおける大きな構造データ、特にリンクによって結ばれているデータの効率良い処理方式に焦点をあて、処理の高速化、資源の節約が可能な構造データの共有方式についてシミュレーション評価に基づいた検討を行なう。

第２章では、本研究の出発点となるＰＩＥ第一次モデルについて簡単に述べる。第３章では、構造データの処理方式の分類と、各方式の特徴について述べる。第４章では、構造データについての特性を調べる為に作成したシミュレータと、その処理の様子について述べる。第５章では、シミュレーションによって得られた結果についての解析、検討を行なう。第６章では、構造データの共有方式向きのアーキテクチャについて考察する。

２．　ＰＩＥ第１次モデルの概要

ＰＩＥの基本設計方針は、次のようなものである。

（１）　ゴール書き換えモデルの上で中間ゴール節（ゴールフレーム、以下ＧＦと略す）を次々と書き換えながら処理を進める。

（２）　各ＧＦは論理的に高い独立性を保っており、これにより並列処理が可能となる。

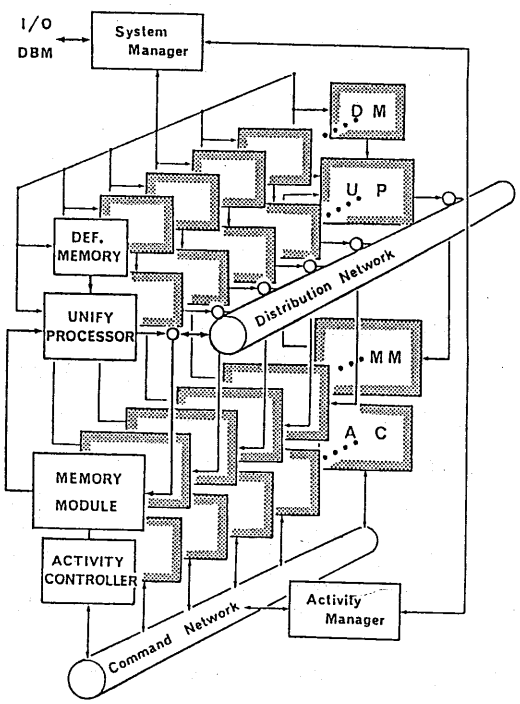


図１．　ＰＩＥ第一次モデル

ＰＩＥ第１次モデル（図１）ではＧＦの書き換えを行なう毎に縮退操作を行ないＧＦ長を短く保つ。ＧＦをＵＰ（ユニファイプロセッサ）ーＭＭ（メモリモジュール）間で転送しながら処理を進める。上記の基本方針を素直な形で実現した結果、全体構成を簡明にすることができた。

以下、プロセッサとメモリの関係に注目する。中間ゴール節を表現する個々のＧＦをゴールプール（ＭＭ）より取り出しＵＰに送る。ＵＰ内で単一化が実行され、新たに生成されたＧＦがゴールプールに格納される。この処理を並列に行なうためにＵＰを複数個、ＭＭも複数個用意し、その間をネットワークで結合した。このとき、１つのＵＰと１つのＭＭは直結しており、ＵＰは自分と直結しているＭＭに対するＧＦの読み出しや書き込みをネットワークを介さずに直接行なう。従ってマシン構成は、このＵＰーＭＭペアがＧＦを分配するためのネットワークに結合しているイメージになる。さらに、現在ＧＦ分配に用いられるネットワークを階層化し、ローカルとグローバルの２階層とすることを検討中である。ローカルなネットワークはＵＰーＭＭペアを密に結合し、ＵＰの稼働率を均等にし、局所的に最適な状態に保つ。グローバルなネットワークは、ローカルネットワークの範囲だけでは重過ぎる仕事を分散するために用いる。

３．　構造データの処理方式

３．１　構造データの処理方式の分類

構造データに対する処理方式という面から見たＰＩＥ第１次モデルの位置付けを明確にし、次期モデルを考える指針とするため、構造データの処理方式を大きく２つの面から分類した。１つはＧＦのＵＰ内書き換え／ＭＭ内書き換え、もう１つはＧＦ生成時の構造データのコピー／共有である（表１、図２）。

★　ＧＦのＵＰ内書き換え　：
ＧＦの書き換えをＵＰ内で行なう。ＧＦをＭＭより取り出しＵＰに取り込み、その中で単一化、縮退操作を施し、再びＭＭに戻す。

★　ＧＦのＭＭ内書き換え　：
ＧＦの書き換えをＭＭ内で行なう。単一化における照合操作をＵＰで行ない、縮退による子ＧＦの生成はＭＭ内の親ＧＦを用いて直接行なわれる。

★　構造データのコピー　：
ＰＩＥ第１次モデルで行なわれているように、子ＧＦが親ＧＦの持っていた構造データ全体をコピーして持つ。

表１．構造データの処理方式の分類

書換え ＼ 構造データ	コピー	共有
MM内	①	②
UP内	③ ＰＩＥ第１次モデル	④

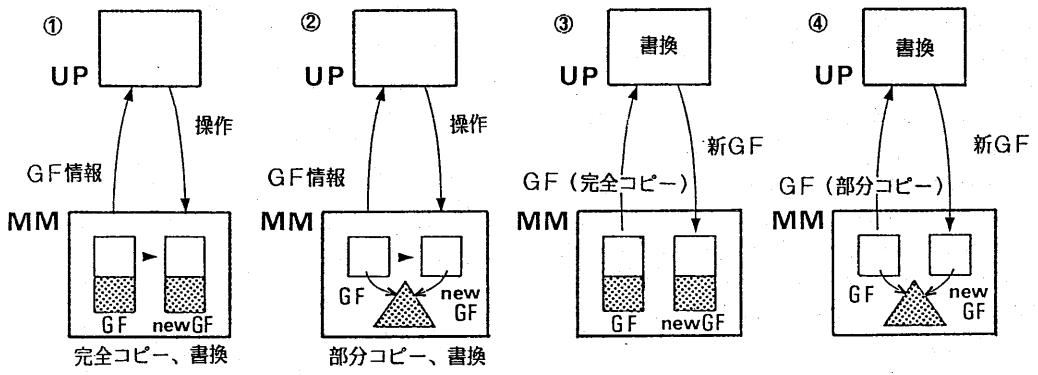


図２．　構造データの処理方式の分類

★ 構造データの共有 ：

各ＧＦ間で共有可能な構造データのGround Instance（以下ＧＩと略す）の部分を共有する。このとき、ＯＲ関係にある子ＧＦ間での共有と、親ＧＦから子ＧＦを生成する時の共有の両方を含む。

各処理方式のイメージを以下に述べる。

①及び② （MM内書き換え／コピー、共有） ：

下記の③、④方式では処理の中心がＵＰであるのに対しＭＭにも処理を分担させる方式である。処理機能の分担の方法によって様々のものが考えられる。ここでは、ＵＰへ単一化に必要な情報のみ送りつけ、それに対してＵＰは子ＧＦ生成（縮退）に必要な操作をMMに送り出すものとする。ＧＦはMM内で縮退操作を施される。この場合、ネットワークを通るデータ量は①～④の内で最も少ないであろう。しかしMMには高度な機能（例えば単一化に必要なデータの抽出、子ＧＦ生成に必要な基本操作の実装等）が要求される。これはデータフローマシンにおける構造化メモリのイメージに近い。①と②の違いは、MM内で子ＧＦを作る時共有できる構造データを実際に共有するか否かだけである。

③ （ＵＰ内書き換え／コピー） ：

ＰＩＥ第１次モデルで採用している。MM中より親ＧＦ全体がコピーされてＵＰに送り込まれる。ＵＰでＧＦに単一化、縮退操作を施して子ＧＦの全体がMMに送り戻される。簡明な方式であるが、ＧＦサイズが大きくなった場合、子ＧＦ生成時の書き換え操作の手間やＧＦ分配時におけるネットワークの転送量が増大する。

④ （ＵＰ内書き換え／共有） ：

今回シミュレータで採用したモデルである。③と同様にＧＦの単一化、縮退操作をＵＰ内で行なうが、単一化、縮退に必要なＧＦの一部分のみMMからＵＰに送る。親ＧＦと一緒に送られてこない構造データ部分はＧＦからのポインタによって参照される。単一化時、MM中の情報がさらに必要になった場合はポインタを１段ずつたぐり、その情報をＵＰに取り込む（追加読み出し、Lazy Fetchと呼ぶ。以下ＬＦと略す）。単一化、縮退が終了し、子ＧＦを作る時、親ＧＦの持っている構造データはポインタによって受け渡され、子ＧＦはMMに戻される。その結果、親ー子、子ー子ＧＦ間で構造データを共有することになる。共有された構造データはMMに置き去りになったままで処理は終了する。ネットワークの転送、縮退操作の手間、メモリ容量を考慮すると、かなりの効率化が期待できる。ただし以下のような問題点が生じる可能性はある。

・追加読み出し操作による待ち時間
・単一化、縮退に必要な情報の目印付けと切り分け
・MMのガーベジコレクションによる効率低下

```
?- P(...),R( [[a,b|*L],m,n] ).
?- Q(...),R( [[a,b|*L],m,n] ).
```

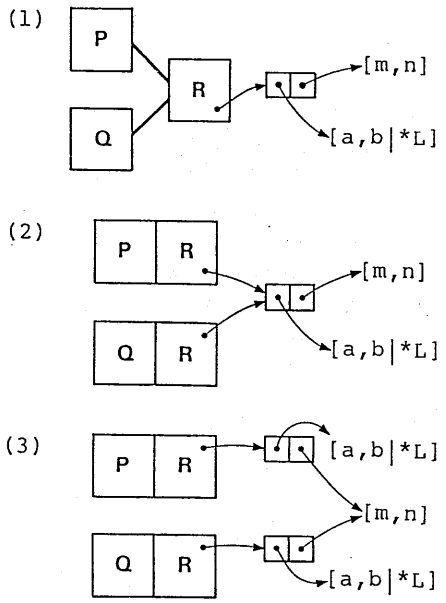


図３．構造データの共有レベル

３．２ 構造データの共有レベル

以上、述べたように、共有可能な構造データをMM内に置き去りにすることによって、処理の効率化が図れる。構造データの共有のレベルはＧＦの内部表現と密接に関連しており、ここでは以下の３つに大別した。

（１）リテラルのレベル、
（２）構造データ全体のレベル、
（３）構造データ中のGround Instance （ＧＩ）のレベル、

この３つの違いを具体例で示す。表記上では図３のように見えていても、３つの共有レベル毎に内部表現は異なったものになる。

（１）の場合、リテラルＲ全体を共有している。表現は最

もコンパクトになる。ただし単一化、縮退の時、共有している未定義変数を1つのGF中だけで識別するのは困難であるため、高並列処理に必要なGF相互の独立性は損われる。また縮退の時は、値の代入や変数名の書き換えのために構造データ全体を調べなければならない。

（2）の場合、リテラルR自体は別々だが構造データはすべて共有している。（1）と同様、未定義変数を共有することによってGF相互の独立性は損われ、並列性を取り出すことが難しくなる。

（3）の場合、共有度は最も低いが、未定義変数を含まない構造データ部分（GI）のみの共有であるため、単一化時における1つのゴール節の中での変数の識別は容易であり、縮退時の値の代入、変数の書き換えも、構造データを全く共有しない場合と同様に行なえる。

以上のように3つの共有レベルを各々比較した結果、GFの独立性が十分にあり並列度を損わないという点で（3）をシミュレータで採用した。

4．　シミュレータでの処理方式

大きな構造データを操作するプログラムでは構造データをGF間で共有することによって処理効率の大幅な向上が期待できる。潜在的な効率化の可能性、及びその理想的な場合の上限を確認するため、第3章までの検討より、シミュレーションモデルとして④ー（3）（UP内書き換え／GIのみの共有）方式を採用し、シミュレーション評価を行なった。シミュレータはUNIX上のC言語、約4000行で書かれており、VAX11／730上で稼動している。

4．1　構造データ共有方式とGFの内部表現

シミュレータでのGFは、文献［4］のものとは内部表現が一部異なっている（図4）。

ヘッダ部とリテラル部は（a）、（b）ともに同じものである。ヘッダ部はこのGF全体に関する情報、例えばGF長、所有している未定義変数の個数等を持っている。リテラル部はリテラル個々に関する情報、例えばリテラル名、引数の情報等を持っている。

（a）のストラクチャ部は、そのGFが持つ全ての構造データであり、未定義変数もGIも共に含む。（b）において、（a）のストラクチャ部に相当する部分が未定義変数を含む部分と含まない部分に分割されている。（b）でのGFの上の部分（ヘッダ部、リテラル部、未定義変数を含む構造データ部分）をGFの暫定読み出し部分と呼ぶ。（b）において、GIからGFの暫定読み出し部分へのポインタは存在しない。他のGFの暫定読み出し部分や他のGIからのポインタ、他のGIへのポインタは存在する。

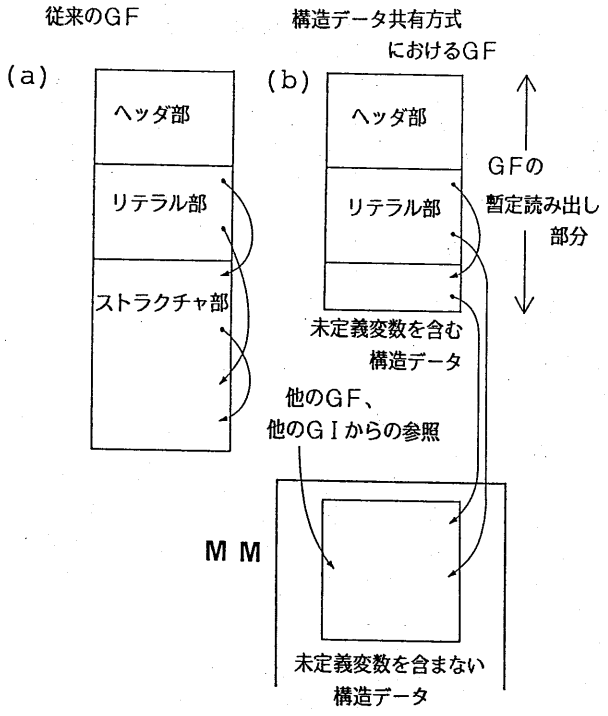


図4．GFの内部表現

4．2　構造データ共有方式の処理例

ここではシミュレータの中で実際に行なわれる処理の様子を示す。但し、GF全体の内部処理と直接関係ない所は省略した。またここでいうセルとは、GFを構成する最小単位のことである。ポインタやシンボル等は1セルで表現でき、リストのノードは2セルで表現できる。

```
Goal        : ?- P( [b,c], *L ),Q( [a|*L] ).
Definition  : P( *X, *X ).
```

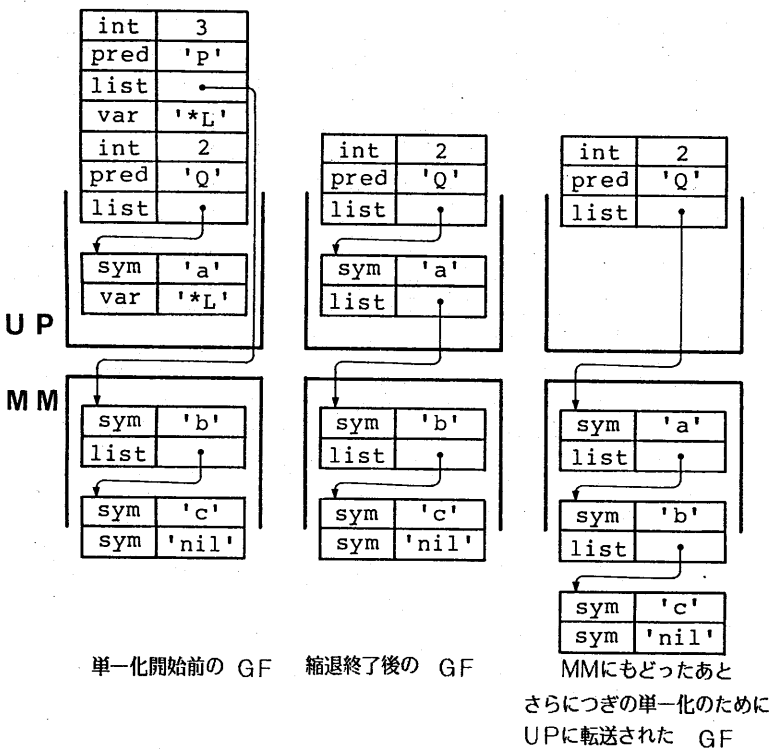


図5. 処理例1.

処理例1.（図5）

この例では、追加読み出しを1回も行なわずに単一化、縮退が終了する。構造データを共有した分だけネットワーク転送負荷、縮退の手間が軽減されている。但し、子GFが次にMMからUPに転送される際、構造データの切り分けが必要になる。

処理例2.（図6）

この例では追加読み出しが1回生じ、2セルだけMMからUPに転送される。しかし［b，c］というリストは、MMに置き去りにしたままで単一化、縮退は終了する。

処理例3.（図7）

（2つの［a，b］は実体が異なるとする）

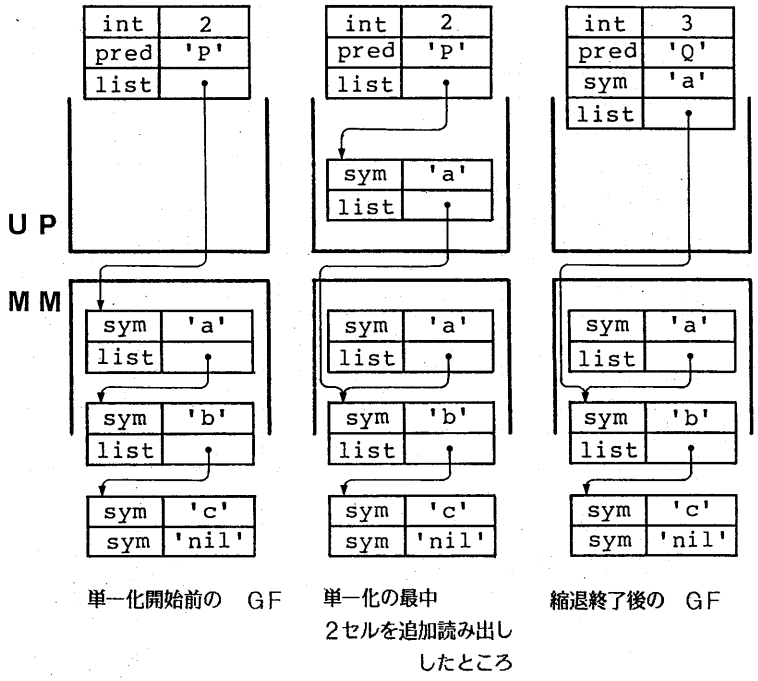
これは単一化のため、すべての構造データが必要となる。さらにUPに取り込まれた構造データはMMに戻った時、全く同一でも異なる構造データとして扱われる。このような場合はコピー方式より効率が低下するであろう。

尚、現在のシミュレータではGFの暫定読み出し部分のみUPにバッファリングしている。即ち、1度UPに送り込めば複数個の定義節と単一化が可能である。これに対し追加読み出しにより取り込まれたセルはUP中にバッファリングされないので、複数の定義節との単一化、縮退を行なう場合は毎回読み出して来なくてはならない。次期シミュレータでは追加読み出しセルのバッファリングを行なう予定である。

4．3 構造データの共有方式の特徴

ここで構造データの共有方式の特徴を簡単にまとめてみる。

```
Goal       : ?- P( [a, b, c] ).
Definition : P( [*H|*L] ) :- Q( *H, *L ).
```



単一化開始前の　GF　　単一化の最中 2セルを追加読み出したところ　　縮退終了後の　GF

図6．処理例2．

◇ UP−MM間ネットワーク転送
- ネットワークを通るデータには2通りのタイプ（GFの暫定読み出し部分と追加読み出しセル）がある。
- GF長はGIの部分だけ短縮される。
- 追加読み出しの分だけアクセス回数が増大する。転送量自体は比較的少ない。

◇ 単一化、縮退
- 単一化に要するステップ数は変化しないが、追加読み出しセルの到着を待つ間が無駄時間となる。
- 縮退時にMMに置き去りにした構造データをコピーしない分だけ高速になる。

◇ メモリ
- 構造データを共有しているので小容量で済む。よってハードウェアコスト減少、資源の有効利用につながる。
- MM中のGF密度が上昇するのでメモリアクセス競合が起こり易くなる。

5．検討

5．1 シミュレーション結果について

シュレーションによって得られたデータを以下に示す（図8〜図10）。テストプログラムには
<1>Equiv2　（数式簡単化、文献［4］）、
<2>ペントミノ、
<3>LL2P　（覆面算、文献［4］）、
の3つを用いた。ペントミノのプログラムは、文献［5］を参考にして書いた。シミュレータの解探索ストラテジが横型探索であるため、データは横軸を関係木の深さにとった。得られたデータの分散は小さかったので平均値のみを表示し、最大最小値は表示しなかった。グラフ中の各々の番号は次のような意味である。

［A］必要なデータセルの総数
① GFの暫定読み出し部分の全体量、
② 従来の方式でのGIの全体量、
③ 構造データ共有方式でのGIの全体量、
従って
②／③ … 構造データの共有度、
①+② … 従来方式で必要なメモリ総容量、
①+③ … 共有方式で必要なメモリ総容量、

```
Goal       : ?- P( [a,b], [a,b] ).
Definition : P( *X, *X ) :- Q( *X ).
```

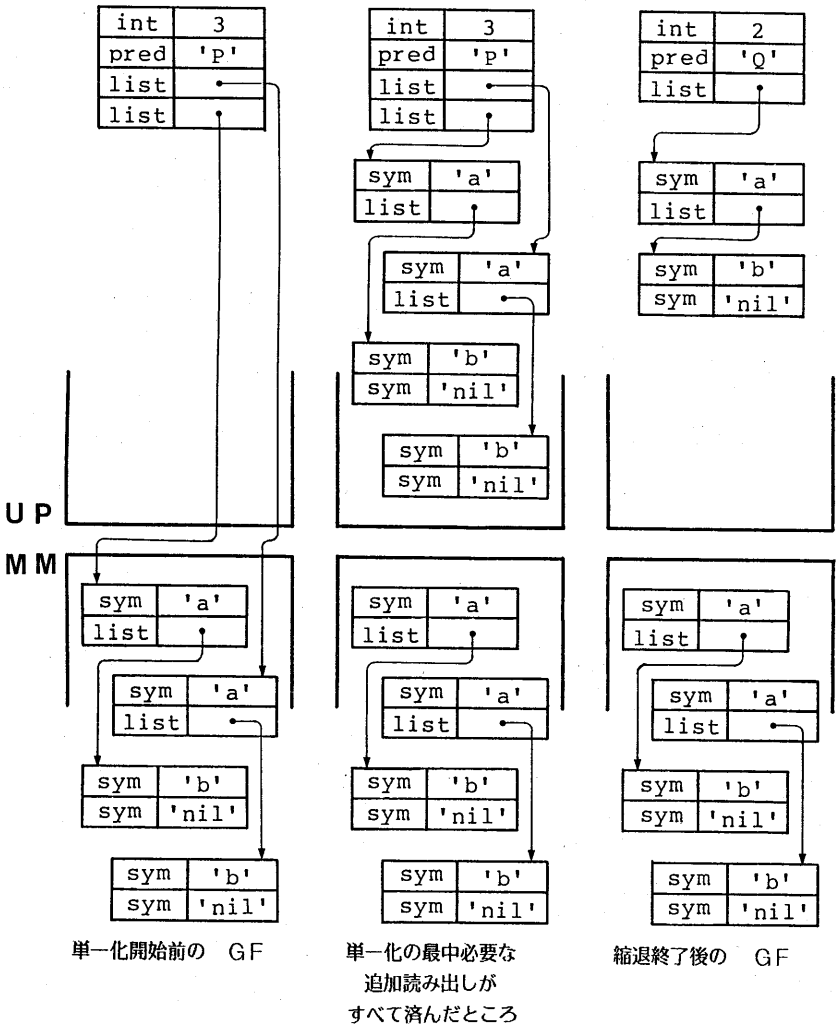


図7．処理例3．

である。

［B］GFの各部分を構成するセル数の平均（1GF当たり）

④ GFの暫定読み出し部分の長さ、

⑤ （イ）従来方式でのGFのストラクチャ部全体、

（ロ）（イ）の内、GIの占める部分、

従って

（イ）－（ロ） … 共有方式の暫定読み出しGF内に存在する未定義変数を含む構造データ部のセル数、

④ ＋（イ） … 従来方式で1つのGFを構成するのに必要なセル数、

④ ＋（ロ） … 共有方式で1つのGFを構成するのに必要なセル数

である。

［C］追加読み出しされたセル数の平均（1単一化当たり）

⑥成功、失敗に拘わらず、1単一化当たり追加読み出しによってUPに持って来られたセル数である。

［D］UP－MM間ネットワーク転送量

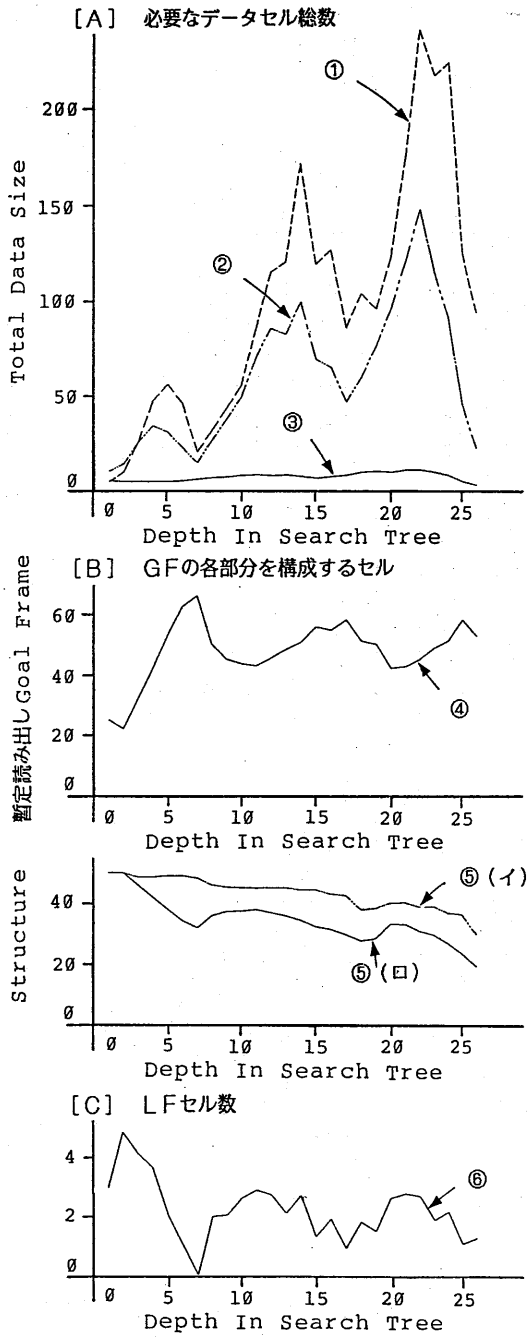


図8. シミュレーションデータ（<1>Equiv2）

全UP−MM間で転送されたデータサイズの分布を表わすヒストグラムである。従来方式（コピー）の場合、MMは単なるメモリとしての役割しか持たない為、MMに書き込んだ

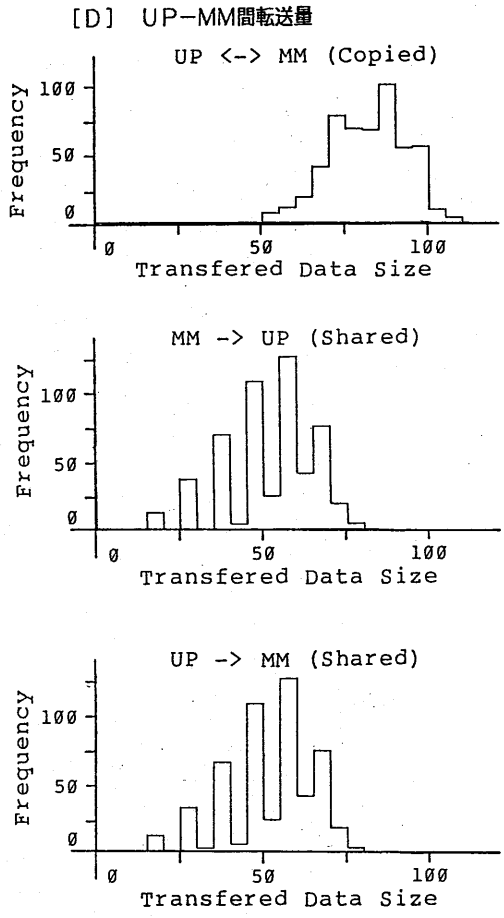


図8．シミュレーションデータ（<1>Equiv2）

り、読み出したりするGFの個数及びその大きさは等しい。共有方式の場合はUP内で作られたGIをかかえたGFが存在するので、MMから取り出す時、GIの切り分けを行なわなければならない。追加読み出しセルを考慮しなければ転送量はMMからUPの方が若干少ない。3つのヒストグラムの内、上の1つがコピーした場合、下2つが共有した場合である。追加読み出しの回数は表2にまとめた。4．2節でも述べたように、追加読み出しの回数はバッファリングしていない時のものである。また0セルというのは、成功、失敗を含めて、一度も追加読み出しせずに単一化が終了した回数である。

5．2 データの解析

5．1節の<1>〜<3>のプログラムでシミュレーションを行ない、［A］〜［D］の各項目について解析を行なう。

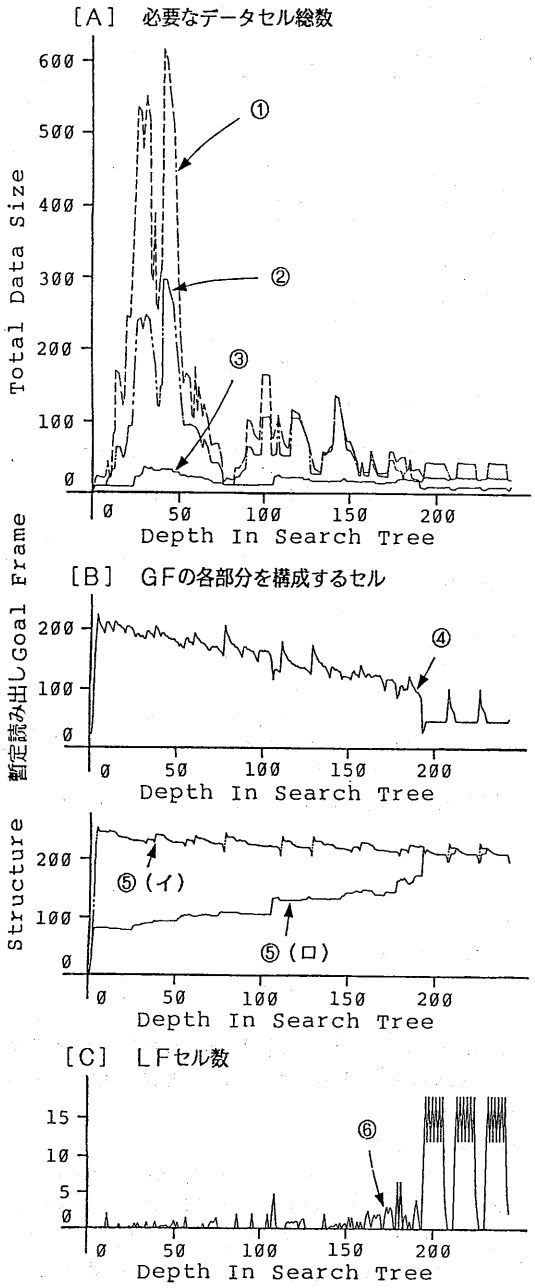


図9．シミュレーションデータ（<2>ペントミノ）

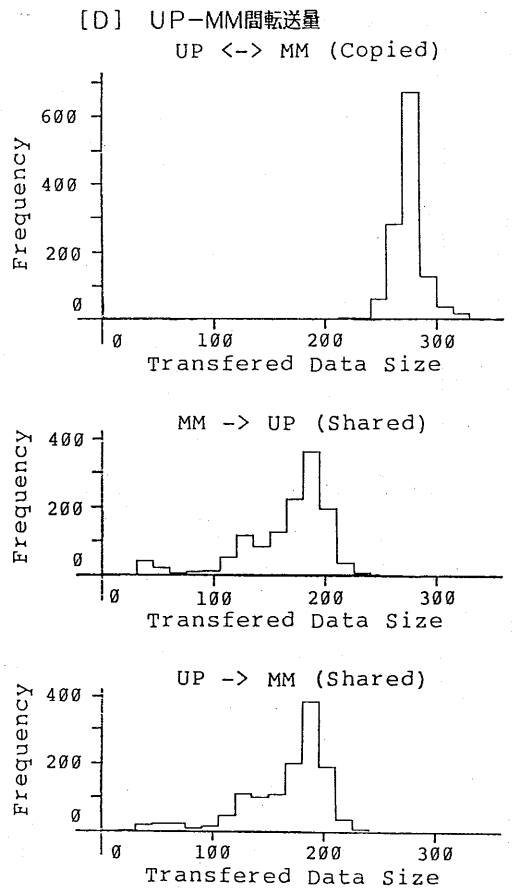


図9．シミュレーションデータ（<2>ペントミノ）

<1>Equiv2（図8）

[A]①②③

このプログラムは1つのゴールに対して成功する定義節の数がかなり多い為、同じ構造データを共有している子GFが多く存在していることを示している。このようなプログラムでは、メモリを爆発的に使い切ってしまう危険性が減少する。

[B]④⑤

GFの暫定読み出し部分の大きさと、ストラクチャ部（⑤（ロ））の大きさが3：2程度なので、GF長は約3／5に減少することが予想され、それは[D]のネットワーク転送量のデータとも符合する。

[C]⑥

処理全体に渡って平均的に約2セルが追加読み出しされている。本シミュレータでは追加読み出しセルをバッファリングしていないが、単一化で必要とされる追加読み出しセルがその対象となっているGFに対して同一である可能性は高い。

[D]

従来方式（コピーした場合）と共有方式では[B]の結果からの推察通り、かなり大きな差が出ている。この差はネットワーク転送量だけでなく、縮退に要する時間にも影響を与える。つまり減少したセル数分だけ縮退の手間も減少している筈である。この差はすべてGFの持つ構造データのGI部

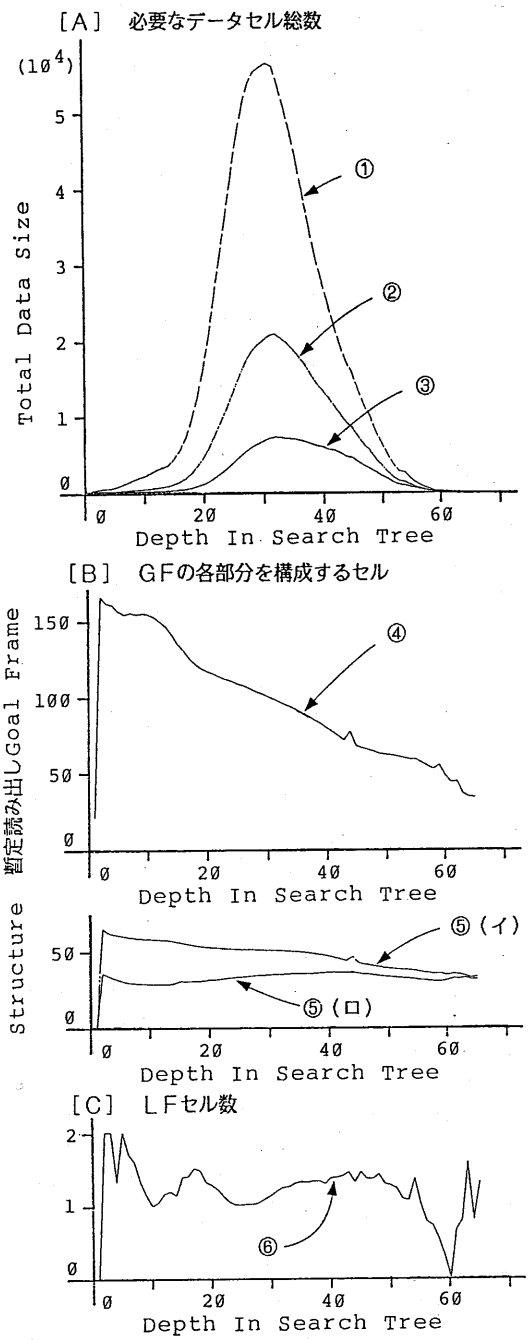


図10. シミュレーションデータ (<3>LL2P)

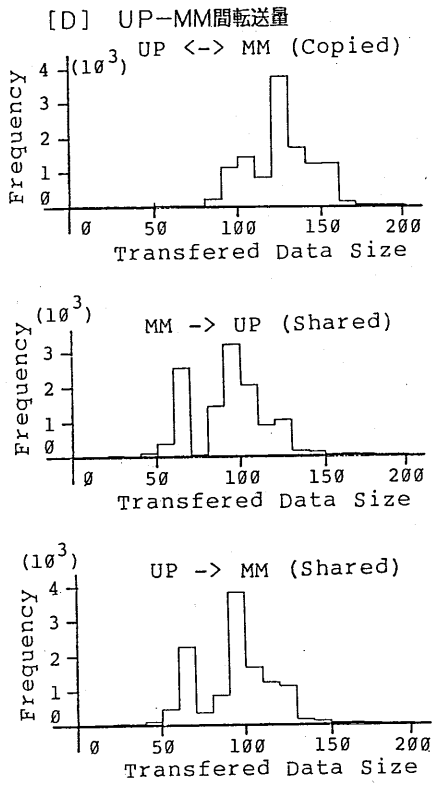


図10.シミュレーションデータ (<3>LL2P)

分に起因している。よってGI部分が大きくなればなる程、共有方式の効率は向上することになる。

<2>ペントミノ (図9)

[A] ①②③

従来方式では①②のように必要なメモリ容量が大きく変動するにも拘わらず、共有方式では少ない変動で押えることが可能となっている。

[B] ④⑤

このプログラムでは全構造データの総セル数はあまり変化せずに、その中に含まれるGIの量が徐々に増加していく。全体として見ると、GFの暫定読み出し部分の大きさとストラクチャ部の大きさは同程度であるからGF長は約半分に減少することが予想され、それは[D]のデータにも現われている。

[C] ⑥

全く追加読み出しを行なわない時期もあり、かなり不規則な動きをしている。保持している構造データが100〜200と大きく、比較的深いレベルまで追加読み出ししていることがわかる。最後の方での急激な増加は、変数に代入が行なわれ、GIになった構造データに対してパターンマッチが施されている為であろう。

表2．　追加読み出し（ＬＦ）セル数とその回数
（ＬＦセルをＵＰでバッファリングしない時）

プログラム	セル数	回数
Equiv2	0	4098
	3	623
	4	3391
ペントミノ	0	5499
	2	724
ＬＬ2Ｐ	0	7164
	2	12320

［D］
この場合も共有方式の方が従来方式に比べて転送時間はおよそ半減することがわかる。同様に、縮退に要する時間もほぼ半減していると考えられる。

＜3＞ＬＬ2Ｐ（図10）

［A］①②③
このプログラムでは、親ＧＦ１個に対して子ＧＦが高々２個しかできず、あまり共有度は高いとはいえない。また大きな構造データは生成されていない。
［B］④⑤
ＧＦの暫定読み出し部分の長さが構造データに比べてかなり大きく、さらに構造データの中には未定義変数を含む部分が多いため構造データ共有方式の利点はあまり活用されていない。
［C］⑥
これはEquiv2の例と似ている。全体に渡っておよそ１セルの追加読み出しが生じている。
［D］
他の２つのプログラムと比較して、それ程ネットワーク負荷、縮退の手間は減少していない。これは、あまり構造データ共有方式にとって有利でない例題である。

このように処理効率はプログラムの特性に大きく依存することが分かった。具体的には構造データ共有方式に有利なプログラムとは次のような特徴を持つものである。

▽ 大きな構造データが多く出現し、その時、未定義変数は構造データの根ノード近く、できるだけ浅いレベルに存在している。
▽ 構造データの共有度が高い、即ち、ある親ＧＦ対して子ＧＦが多く存在し、その間で構造データを共有している。
▽ 追加読み出しが少くなるように構造データはポインタで受け渡しをする。またできる限り構造データの上部で処理を済ます。

これより構造データ共有方式に有利なプログラムの書き方も存在するであろう。例えば構造データを作る時は上にデータを積み上げるようにして作る方が良いとか、未定義変数を含む構造データはフラットな形ではなく入れ子の形にした方が良い等である。こういう点を考慮して従来のプログラムを効率の良いものへ書き換えることもできる。実際の応用を考えると、今回のテストプログラムは
・小さすぎる、
・共有度が低い、
・大きな構造データが出現しない、
という点で不適当ではある。しかし、この程度の条件でも大幅な効率化が図れることは、容易に推察できる。

６．　構造データ共有方式に関する検討

これまでの議論、シミュレーションデータの解析により構造データ共有方式による効率化は可能であるとの結論がでた。ここでは、構造データ共有方式（④方式）に適したアーキテクチャについて議論する。ＰＩＥでは、ネットワーク転送、単一化、縮退といった操作がパイプライン的に円滑に進むようにアーキテクチャを構築せねばならない。

６．１　構造データ共有方式におけるＵＰ

ＵＰは、複数の定義節との単一化、縮退を行なうので、取り込んだＧＦをバッファリングしておかねばならないが、追加読み出しをしたセルに対しても同様、バッファリングするとよいであろう。また追加読み出し中の待ち時間をどうするかが一番大きな問題である。追加読み出しセル待ちをしている間、他の定義節との単一化を行なうなど単一化レベルでの並列処理化を考えなければならない。また、生成した子ＧＦを次にMMから取り出す時に、構造データの切り分けを行なうため、まず子ＧＦの持つＧＩに印付けをしなければならない。縮退後出力バッファに書き出す時、この操作を兼ねさせることが可能である。

６．２　構造データ共有方式におけるネットワーク

ここでは、ネットワーク階層におけるローカルなネットワークの方についてのみ考える。この時、次の２つの網が必要となる。

◎ ＵＰ−MMペア間でＧＦを分配する網（ＧＦ分配網）は、負荷分散の為に用いる。ＧＦは他のＵＰ−MMペアに任意に分配する。
◎ 他のＵＰと直結しているMMから散らばったセルを追加読み出しする為に、ＬＦ網を用いる。

ＧＦ分配網は高スループットが要求され、ＬＦ網は高速なレスポンスが要求される。

このように密な結合のもとでのUP−MMペアの最適台数は今後の検討課題である。これらを検討する為の次期シミュレータのネットワーク構成の概略は、図11のようになるであろう。

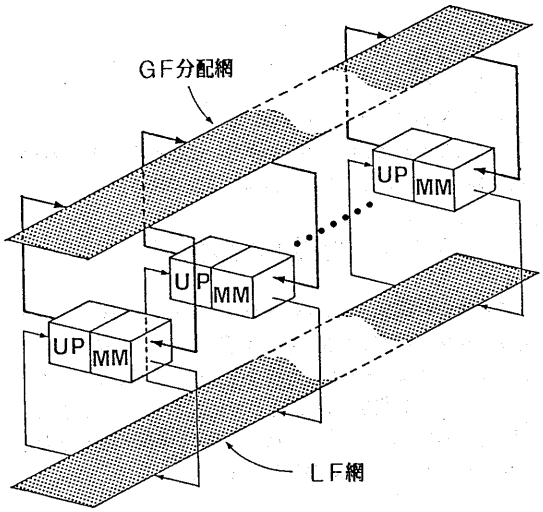


図11. ローカルなネットワーク構成

6.3 構造データ共有方式におけるMM

従来方式（PIE第1次モデル）のMMは単なるキューだと考えても差し支えないが、構造データの共有を行なうと、ガーベジコレクション（以下GCと略す）が必要になる。システムとして、GCによって長時間処理が中断するのは好ましくない。従って実時間GCでなければならないと考える。PIEモデルでは、他のUP−MMペアに分配した子GFが、自分のMMを参照したり、GCが行なわれているMMを参照しているGFが別のUP内に存在していることもある。従って、GCがその参照先を勝手に変更できない。現在検討している方式としては

☆ 参照カウンタを用いる方法、

☆ 2つのセミスペースを同時に使用する方法［7］、

がある。これら2つの方法は、各MMごとに行なうのであるが、MMは複数個あることを利用して、

☆ 逐次型GCでMMをバンクごとにGCしていく方法、

☆ 以上の方法を組み合わせた方法、

なども考えている。

7. おわりに

知識情報処理、特に自然言語処理などで頻繁に出現する大きな構造データを効率良く処理するためには、本報告で述べてきたように、構造データを分割し、GF間で共有するという方法が有効であることが分かった。プログラムの特性によっては大幅な処理効率の向上が期待できる。その有効性を最大限に引き出すためのアーキテクチャに関する考察を行なった。第3章の②方式はPIE第2次モデルの候補の1つであり、その場合、MMの高機能化が重要な課題となる。構造データを効率良く処理するために採用した構造データの共有方式がMMの高機能化への糸口となった。今後、次期シミュレーションモデルを構築し、構造データの共有方式を取り入れた場合、システム全体としてどの程度の性能が達成可能なのか、シミュレーションによる評価を進める。

謝辞

本研究を行なうにあたり、いつも深夜まで熱心に討論し、貴重な意見をして下さるSIGIE†のメンバー諸氏に感謝いたします。

<< 参考文献 >>

［1］ 丸山 他，“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”，信学技報EC83−39（1983）.

［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］ 平田 他，“高並列推論エンジンPIEにおけるゴールメモリ構成法に関する一考察”，第27回情処全大4P−13（1983）.

［4］ 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）.

［5］ 古川，近藤，“Two Dimensional Programming in Prolog”，The Logic Programming Conference ´83，ICOT（1983）.

［6］ Ciepielewski，A. and Haridi，S.，“A Formal Model for OR−Parallel Execution of Logic Programs”，IFIP，1983.

［7］ Baker，H.，“List Processing in Real Time on a Serial Computer”，CACM 21，4（April 1978）.

†SIGIE®
(Special Interest Group on Inference Engine)
Privately Founded in 1982,
at Moto-oka Tanaka Lab.,
University of Tokyo