ＰＩＥにおける構造メモリの構成について

平田　圭二，　田中　英彦，　元岡　達

（東京大学　工学部）

１．はじめに

現在我々は高並列推論エンジンＰＩＥの設計・試作を進めている。ＰＩＥはゴール書き換えモデルに基づきProlog に代表される論理型プログラミング言語を、ＯＲ並列に直接実行するマシンである。ゴールは互いに独立であり、ゴールプール中に格納する。ゴールフレーム間の論理的独立性を保ち、ＯＲ並列性を最大限に抽出する為に、新しいゴールは基本的にコピーすることによって生成される。我々はＰＩＥを設計する第一段階として、完全コピーによりゴールを生成するモデルＰＩＥ－Ｉの検証を行った［１］。

一方、論理型言語の応用プログラムは主として自然言語処理、エキスパートシステム、ＣＡＤなど人工知能、知識工学関連のものが多く、ＰＩＥの稼動時には大きな構造データを持ったゴールが頻繁に出現すると考えられる。しかしＰＩＥ－Ｉは完全コピー方式を採用したのでその処理にはかなりのオーバーヘッドが予想される。構造データ共有方式を用いるとそのオーバーヘッドは大幅に減少することがシミュレーションにより確認され、その結果１桁程度の高速化が達成可能であることが分かった［２］。これを踏まえて我々は構造データ共有方式を採用したＰＩＥ－Ⅱを設計した。ＰＩＥ－Ⅱではネットワークを階層化し、構造データを格納する構造メモリを設けた［３，９］。

本論文では、ＰＩＥ－Ⅱにおける構造メモリの機能、構成法などについて述べる。第２章ではＰＩＥに構造データ共有方式を導入する場合の処理の様子や、構造メモリの基本機能について述べる。第３章ではＰＩＥにおける構造メモリを設計する上で重要であると思われる点について述べ、４章ではＰＩＥ－Ⅱの処理速度に関する簡単な評価を行う。

２．ＰＩＥにおける構造データ共有方式

マシン内でゴールは、ＡＮＤ関係で結ばれたゴールリテラルの集り（ゴールフレーム，Goal Frame，ＧＦ）として表現される。ＰＩＥ－Ｉのアーキテクチャでは、複数の推論ユニット（Ｉnference Unit，ＩＵ）がネットワークで結合されている（図１）。ＩＵは定義節メモリ（Definition Memory，ＤＭ）、単一化プロセッサ（Unify Processor，ＵＰ）、メモリモジュール（Memory Module，ＭＭ）、アクティビティコントローラ（Activity Controller，ＡＣ）から成る。ＤＭには単一化を行う定義節、即ちプログラムが格納される。ＵＰでは単一化・縮退が行われる。縮退操作によって不要な変数や参照されなくなったセルが取り除かれ、親ＧＦ及び定義節の持っている必要な構造データをコピーすることによって新しい子ＧＦが生成される。単一化と縮退はパイプライン的に実行される。ＭＭはゴールプールの役割を果たし、ＵＰは必ず同一ＩＵ内のＭＭからＧＦを取り出す。新しいＧＦは負荷分散の為、スイッチングネットワークで他のＩＵ内のＭＭに送り出されることがある。各ＧＦは推論木上の各ノードと対応付けられており、ＡＣはそのノード操作を行う。このようにＰＩＥではゴールの処理とその制御を分離すること

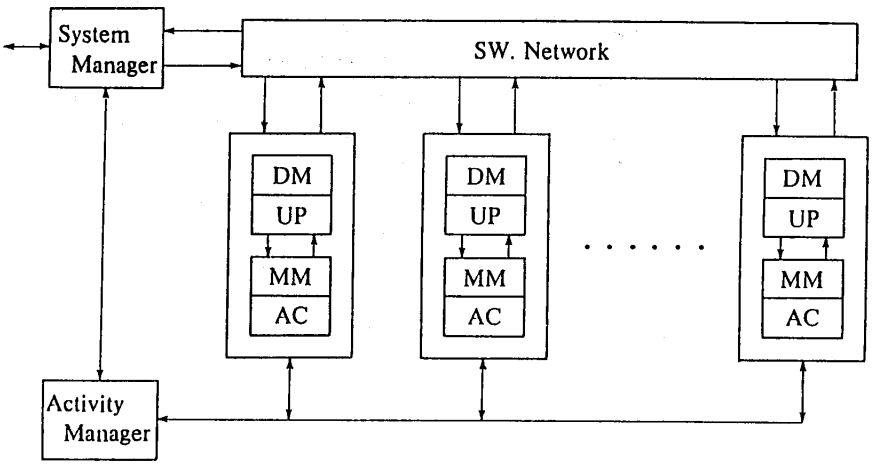

図１．　ＰＩＥ－Ｉアーキテクチャ

によって、NOTやGUARD等の高階機能や、様々な解探索ストラテジを容易に実現できる。

2．1　構造データ共有方式の機構

ＰＩＥ－ＩのＧＦは、ヘッダ部、リテラル部、構造部の3つから成る。ヘッダ部にはＧＦ全体に関する情報（ＧＦid.、リテラル数等）が格納されており、リテラル部にはＧＦ中の個々のリテラルに関する情報（リテラル名、引数の個数等）が格納されている。リテラルの引数として構造データがくる場合は、その構造データは構造部にしまわれ、リテラル部からのポインタによって指される。これに対して、構造データ共有方式でのＧＦはヘッダ部、リテラル部までは同じであるが、ＰＩＥ－ＩのＧＦの構造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けられる。未定義変数を含まない構造データ部分（Ground Instance，ＧＩ）はＧＦ間で共有され、構造メモリ（ＳＭ）に格納される（図2）。共有の対象とならないヘッダ部、リテラル部、未定義変数を含む構造データ部はＧＦの暫定的に読み出される部分を形成し、ＵＰ－ＭＭ間で転送される。ＧＩに対しては決して書き換えが起こらず読み出す一方であるから、ＧＦ間の論理的独立性が失なわれることはない。ＵＰ－ＳＭ間で転送されるコマンドは単なるメモリへの読み書きのレベルではなく、構造データに対してより直接的な高レベルのコマンドに設定し、効率の良いコマンド転送や実行を可能とすることができる。

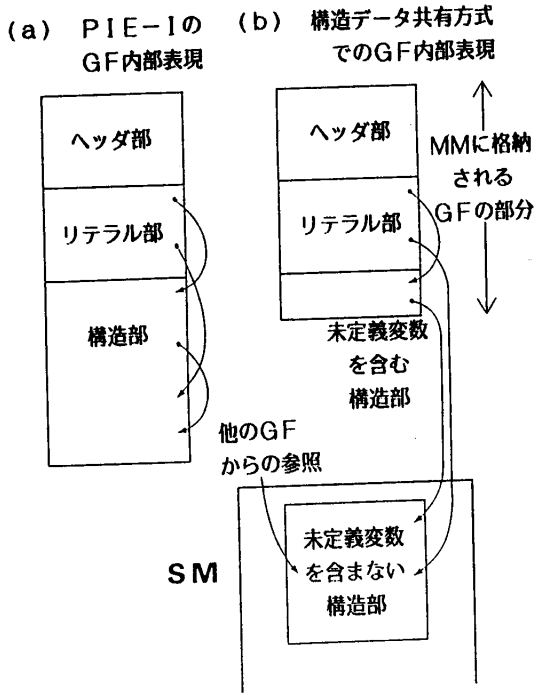

図2．
ＰＩＥ－Ｉ及び構造データ共有方式でのＧＦの内部表現

ＰＩＥ－Ｉでは縮退の時、親ＧＦの持つ構造データをコピーしていたが、構造データ共有方式ではその構造データへのポインタを新しい子ＧＦに受け渡すだけで済む。従って縮退の手間が大幅に省けるだけでなく、ＧＦ長の短縮によりネットワークの転送コストが減少し、必要なメモリ容量も少くなる。

単一化・縮退に要する時間を試作ＵＰを用いて測定した［４］。その結果、各々にかかった時間は関与したセル数に比例し、テストプログラムでは縮退に要する時間の方が、単一化に要する時間の４～１３倍かかることが分かった。従って大きな構造データを抱えたＧＦの処理においては縮退操作が全体の実行速度を決めてしまう。しかし構造データ共有方式を採用しシミュレーションを行ったところ、ＧＦ長は約１／２に短縮し、縮退に要する時間も約１／２に減少することを確認した［２］。実際の応用プログラムではＧＦが更に大きな構造データを抱えることが予想されるので、さらに大幅な高速化が可能となる。従ってＰＩＥ－ＩＩではＧＩだけの共有方式を採用した。

2．2　構造データの処理

構造データ共有方式を導入することでＰＩＥ－ＩＩでは追加読み出し、ＧＩの切り分けという操作が必要になる。以下これらを説明する。

（１）　追加読み出し（Lazy Fetch，ＬＦ）

単一化の最中にＧＩ中の情報が必要となった時は、ポインタをたぐりＳＭ中の情報を取りだしてくる必要がある。この操作を追加読み出し（Lazy Fetch，ＬＦ）と呼ぶ。ＬＦは単一化時に行われる操作である為、高速な応答が要求される。現在、ＬＦの応答時間を短縮し、頻度を低く抑えるための対策を3つ考えている。

まず最初に各ＵＰごとに追加読み出しバッファ（Lazy Fetch Buffer，ＬＦＢ）を設ける。一般に、1つのＧＦは複数の定義節と単一化されることが多く、この時ＬＦされるセルは同一であることが多い。従って、毎回ＳＭへアクセスするのではなく、1回ＬＦされたセルをＬＦＢにバッファリングしておけば、2回目以降のＬＦはＳＭではなくＬＦＢにセルを取りに行くだけで済む。シミュレーションによると、ＬＦＢを設けた場合、ＬＦを行わずに済む単一化は全単一化回数の内、７０～９６％を占めた［９］。

次にＬＦをある程度まとめて行う。シミュレータではＬＦを１回行う度にポインタを１段たぐり、構造データの１ノード分をフェッチすることにした。この時ＳＭへのアクセスを必要としない単一化を除けば、１単一化当り平均１～２回のＳＭアクセスが生じた［９］。ＳＭがＵＰ側で必要としているノードをまとめて転送する為に、そのポイン

タを何段分たぐれば良いかの情報をＬＦコマンドでSMに送ることによって平均SMアクセス回数をほぼ１にすることができる。

最後にＧＦの暫定読み出し部分に構造データを１段たぐったレベルのノードを含ませる。つまりＧＦをMMからＵＰに転送する時に、１段分のＬＦ相当のことを行う。定義節の形を見ればＬＦが生じるかどうかの判定は予め可能なことが多い。実際、半分以上のＬＦがポインタを１段しかたぐらないので、これでＬＦ回数は大幅に減少する。

（２）　Ground Ｉnstance の切り分け

ＧＩが生成されるのは、一番最初のＧＦがメモリ中に投入された時と、ＵＰにおける１回の単一化・縮退が終了した時である。この時、ＧＦのＧＩ部分はSMに、それ以外の部分はMMに格納しなければならない。初期ＧＦの切り分けは入力時に行えば良い。単一化・縮退によって生成された場合は、新ＧＦの持つ構造データを調べ、その中のＧＩ部分を切り分けなければならない。シミュレーションによれば、生成されるＧＩの量は１ＧＦ当り数セル程度である［２］。先に印付けだけを行って、異なるタイミングで切り分けを行うことも可能であるが、その場合は印付けの結果を切り分け時まで保持しておかねばならない。切り分けの操作は構造データをコピーしたり、生成したりする時に副次的に実行できる。そのような時点としては、

（ａ）　ＵＰ内でＧＦに縮退操作を行う時、
（ｂ）　MMに新ＧＦを格納する時、
（ｃ）　MMからＧＦを読み出す時、

がある。MMから親ＧＦを取り出し、ＵＰで子ＧＦを生成した時にはじめて親ＧＦの持つＧＩが共有の対象となり得るので、（ｃ）のような遅いタイミングでも十分である。どのタイミングが最良かはマシンの他の部分をどう構築するかにも関係するので後で議論する。

２．３　構造データ共有方式での構造メモリ

構造データの共有方式を導入した時の処理過程はすでに述べた。ここで、SMで行われるデータ操作についてまとめる。SMの持つべき基本機能は、

（１）　転送されてくる新ＧＩ部分の格納、
（２）　ＬＦコマンドに対するセル送出、
（３）　ガーベジコレクション、

である。新しく生成されたＧＩ部分は、切り分けのタイミングによってＵＰあるいはMMから送られてくる。SMはそれを受け取り格納する。このＧＩを格納するコマンドに対してはSMのどのアドレスにＧＩを格納したかの情報をＧＦに送り返す必要がある。またＵＰから送られてくるＬＦコマンドにはポインタを何段たぐれば良いかの情報が含まれているので、SMはその段数分だけＬＦしたセルをＵＰに送り出す。このようにSMで実行される命令は２種類しかなく、比較的単純なものである。SMに対する書き込みは新しく生成されたＧＩを格納することであり、必ずセル領域の割り当てを伴う。読み出しはＬＦを行うことである。SM中での構造データの書き換えや、コピーを実行するような操作はない。つまりＰＩＥのSMでは他のＧＦに対して副作用を伴うような操作は行わずに済むので、この点では非常に有利である。ガーベジコレクションについては３．３節で述べる。

３．構造メモリの構成

ＰＩＥにおけるSMの設置方法には２通りある。１つは全体で１台のSMを共有する方法であり、もう１つは各ＩＵ毎にSMを設ける方法である（図３）。

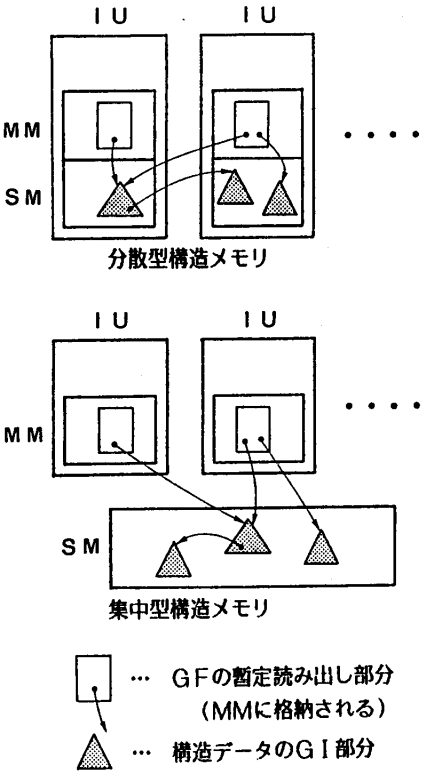

図３．　分散型構造メモリと集中型構造メモリ

現在までに提案されているリダクションマシンやデータフローマシンの多くは、メモリアクセス競合の回避、並列処理の為にメモリを多バンク化しネットワークで結合している。しかし、分散化すれば全体の処理効率をある程度犠牲にして、アクセスの均等化等のユニット間の協調をとらねばならないので、アーキテクチャの複雑化は避けられない。従ってＰＩＥのSMを分散型にするか、集中型にするかは

慎重に検討する必要がある。本章ではSMの構成法を様々な観点から議論する。

3．1　追加読み出しのたぐり

GFは負荷分散を目的としてあらゆるIUに送出される可能性があるので、分散型SM方式では、UPで生成されたGIがポインタによって連結され、複数のSM内に散在することになる。1回の単一化で生成されるGIの量は比較的少いので、1つの構造データはいくつものSMバンク間にまたがって存在する。LFによるポインタのたぐりを何段分かまとめて行おうとすれば複数のSMバンクにアクセスを行わざるを得なくなる。単一化の最中にネットワークアクセスが何度も生じるのは非常に不利である。よってLFの立場からは集中型SMの方が好ましい。しかし集中型方式では複数IUからのアクセスが集中し、SMがシステム性能のボトルネックとなる可能性があり、共有の数には自ずから上限が存在する。

3．2　GIの切り分け操作とGI格納コマンド

2．3節では、SMに格納したGIへのポインタを、GIの格納コマンドに対する応答としてGFに返す必要があると述べた。分散型SMでGIを切り分けるタイミングは、新しいGFが各IUに配られMMに格納される時点、もしくはMMから読み出す時点が良い。何故ならGIの部分をこのMMと同一IU内のSMに格納すれば応答が即座に得られ、LFの時も同一IU内のSMへのアクセスで十分だからである。

集中型SMではSMとIUが分離されている為、切り分けを副次的にどのタイミングで行っても大差はない。この時はSM中のGIへのポインタを手に入れるまでの時間と、その為のネットワークトラヒックの増大が問題となる。これを解決する為に、SMの空きアドレスを分配するリングバスと、各IUに空きアドレスを受け取る入力バッファを設けることが考えられる（図4）。

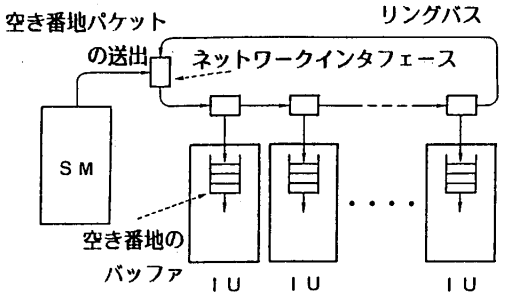

図4．　構造メモリの空き番地分配機構

SM中の空き番地をパケットに乗せてリングバスに送り出す。GIを格納する為にSMへのポインタが欲しい時は、リングバスからパケットを取り込み、そのアドレスを使用する。入力バッファを設ければ、待ち時間は無視できる。一周してSMに戻ってきたパケットが空の時はSMが新しい空き番地を再びパケットに乗せて送り出す。この機構を用いた場合、IUからSMへ送られるGIの格納コマンドは、格納すべきGIとSM内での格納番地から成り、GIの書き込みに対する応答は省略することができる。

3．3　ガーベジコレクション（GC）

まずGCをUPの処理と逐次的に行うか、並列に行うかの選択がある。PIEは複数のIUが並列に処理を行い、IUの中ではパイプライン的に処理が進む。逐次的にGCを行い、長時間処理が中断するとマシンパフォーマンスが著しく低下してしまう。従って並列GCが望ましい。

PIEで取り扱う構造データのノードは可変長であるからメモリ領域の圧縮が必要となる。また圧縮した後、ノードの移転先をMMやUP内のGFに知らせてGFの持つポインタの内容を新しい移転先に書き換えなければならないが、GFからSM中の構造データへのポインタの数は莫大であるからこれは非現実的である。そこでアドレス変換テーブル（Address Transfer Table, ATT）を用意し、ノードの参照は必ずそのテーブルを介して行うこととし、可変長セルメモリ（Vari-sized Cell Memory, VCM）にノードの本体を格納する（図5）。

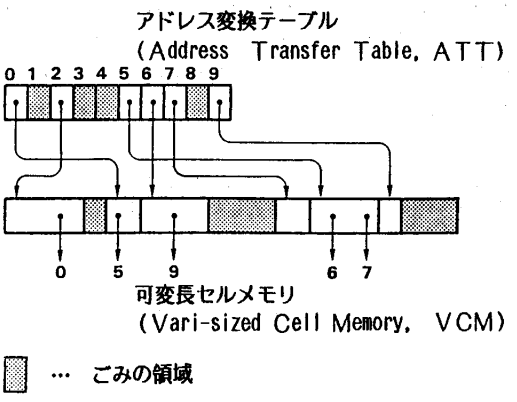

図5．　アドレス変換テーブルと可変長セルメモリ

同様の手法はSmalltalk－80のオブジェクトメモリにおいても用いられている［6］。ATTのエントリは固定長であるから、これは自由リストで管理すれば良い。

ガーベジコレクション（GC）の方式は、ノードが到達可能かどうかの判定によりたぐりを用いる方式と参照カウ

ンタを用いる方式に分類され、また空き領域を自由リストで管理するか、コピーによって圧縮しヒープとして管理するかに分類される［５］。

まず、たぐりを用いた場合について考える。SM内にＶCMを２組用意し、たぐりと印付けで不要なセルの判定を行い、同時にＶCMの生きているノードをもう一方のＶCMにコピーすることで圧縮を行う。ＰＩＥでは根ノードが莫大な数にのぼるので、それらをどう扱うかが問題となる。また分散型SMではSM間にまたがる構造データのたぐりをどのように制御するかも問題になる。

参照カウンタ方式を採用した場合、ノードの参照数が変化するのはＧＦが生成、消滅した時と、ＬＦを行う時である。これらはかなり頻繁に生じるので、ＵＰ中で参照カウンタに対するコマンドを予め最適化しておく等の対策を講じなければならない。またＬＦコマンドは何段分かまとめてたぐりを行うので、ＵＰ側が本当に参照しているセルをSM側では知ることができない。よって参照カウンタに対するコマンドは必ずＵＰ側から発信されねばならない。分散型SMでは参照カウンタに対するコマンドの同時実行制御も問題になる。現在まで参照カウンタ方式を実用化する為の工夫が提案されているが、それはＬＩＳＰマシンでの環境を強く意識したものである［７］。従ってそのまま適用するわけにはいかない。実際、ＬＩＳＰマシンでは大多数のセルの参照数が１だが、ＰＩＥでは参照数がその構造データを必要とするＧＦ数に等しいなど、処理環境は大きく異なっている。またデータフローマシンにおける複数SM間のコマンドトラヒックや参照カウンタ操作の負荷についての報告がある［８］。それによるとセルのアロケートが頻繁になるにつれて、参照カウンタ操作がボトルネックになるとされている。

たぐり、参照カウンタいずれの方式においてもSMを分散型にすると、アーキテクチャが複雑化してしまい、制御のオーバーヘッドが大きくなるので、ＧＣの立場からは集中型SMが望ましい。

３．４　構造メモリに要求される処理能力

以上の議論より、我々はＰＩＥのSMはできるだけ１台に集中した方が良いと考えた。しかしＩＵ台数としては、かなり大きな値（１００～１０００）を想定しているので、１台のSMですべてのＩＵを賄うのは、アクセス競合、SMの処理能力からみて非現実的である。まずSMに要求される処理速度と組み合わせ可能なＩＵ台数の概算を行う。

図６はＩＵとSMにおけるコマンドやデータの流れを表している。ΔＧは新しく生成されたＧＩ部分を指す。ＬＦは追加読み出しによるコマンド、データの流れであることを表す。表１にトラヒックの推定値を掲げる。これらの値はシミュレーションによって得られたデータを基に算出したものであり、①～⑦の各経路を流れるデータのサイズとその生起間隔を示している。ここでクロックとは時間の単位であり、試作ＵＰのマイクロステップに等しくとった（＝２００ｎＳ）［４］。サイズの単位は１セル（＝５バイト）であり、１セルによって１つのポインタや１つのシンボル等が表現できる。例えばリストの１ノードを表すには２セル必要である。⑦のトラヒックの数値は１つのＧＦがおよそ４００クロックで処理されていくことを示してい

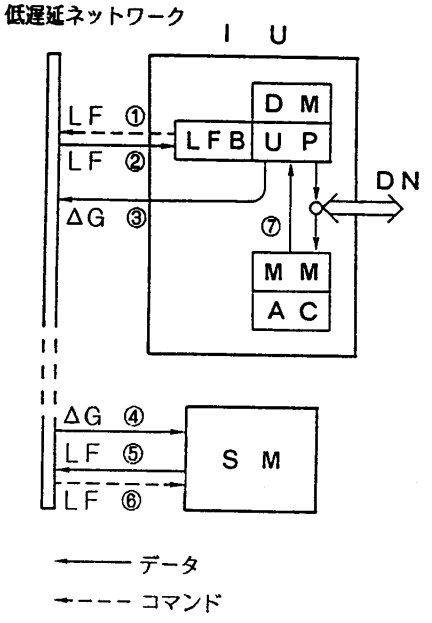

図６．構造データ処理に関するデータ転送

	トラヒックの生起間隔（クロック）	データサイズ（セル）
①	３００～４０００	２
②	３００～４０００	２～１０
③	４００～８００	２～４
④	２５～５０	２～４
⑤	１７～２５０	２～１０
⑥	１７～２５０	２
⑦	４００	十数～百数十
④+⑤+⑥	７～３６	２～１０

構造メモリ全体に要求されるスループット
＝ 低遅延ネットワークのトラヒック
＝ ④+⑤+⑥

表１．　トラヒックに関するデータ

る。

①～③は1台のＩＵ当りのトラヒックであり、④～⑥はＩＵ16台につき1台のSMを設置した場合のSMに対するトラヒックである。表1からSMに要求されるスループットは最繁時1セル／1クロック程度であるので、SM1台にはＩＵ16台程度が限界であることが分かる。

3．2節で述べたように、SMは空き番地をパケットの形でリングバスを通してＩＵに分配すると仮定して、リングバスのトラヒックを概算する。そのパケットは幅32ビット程度のアドレスを格納している。単一化・縮退が1GF当り400クロックかかって完了し、新しい根ノードが1．5個／1GF含まれると仮定すると、ＩＵ16台では24パケット／400クロック必要になる。つまりリングバスのトラヒックは約1パケット／17クロックである。

4．性能評価

ここではテストプログラムについて行ったシミュレーション［2］［4］［9］の結果を基に、簡単な性能評価を行う。以下1クロック＝200nS、1セル＝5バイトとする。ＰＩＥ－ＩではGFの単一化・縮退に平均700クロック要し、GFの平均サイズはおよそ100セルである。700クロックの内、7～20%が単一化の時間、残りが縮退の時間である。今、単一化・縮退がパイプライン的に進行しても、MMのスループットは100セル／660クロック（1バイト／260nS）であり、縮退の時間がボトルネックとなる。従ってＰＩＥ－ＩのＩＵ1台当たりの処理速度は約8KＩＰSである。　構造データ共有方式を採り入れたＰＩＥ－Ⅱでは、LFによって単一化時間は平均約20クロック増加する。縮退時間は1／2に減少して約300クロックかかる。やはり縮退時間がボトルネックとなり、ＰＩＥ－ⅡのＩＵ1台の処理速度は約16KＩＰSである。

以上を踏まえて、我々はＰＩＦを階層的に構成することにした（図7）。ＩＵ16台につきSMを1台設け、レベル1システムを作る。この中ではGFは構造データを共有する。1つのレベル1システムだけでは負荷が重過ぎる仕事の場合は、レベル2ネットワークを介して他のレベル1システムにGFを分散させる。この時転送されるGFはSM中のGＩ部分をコピーして持っており、物理的に完全に独立した形をしている。従ってレベル1システム間での構造データの共有は起こらない。ＰＩＦの処理能力を上げたい時は、階層レベルを増やしレベル1システムを必要に応じて接続していけば良い。構造メモリを導入したことで、UP側の負荷増大は小さく押えられており、レベル1システム1つで250KＩＰSの処理速度を持つ。

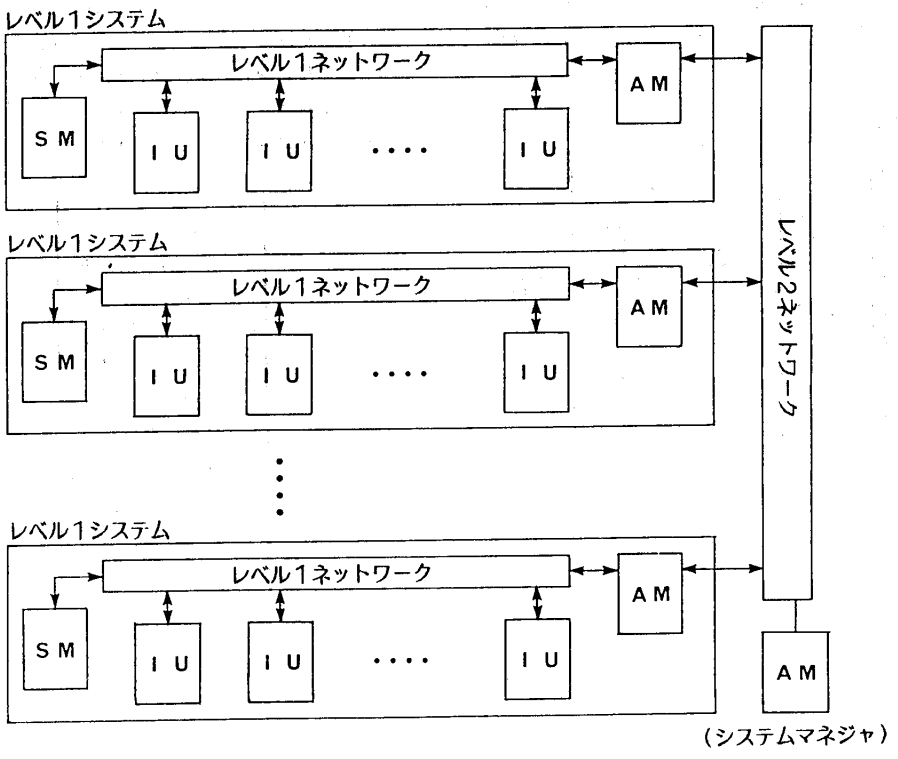

図7．　ＰＩＥ－Ⅱアーキテクチャ

5．おわりに

我々は今後さらにSMの高機能化を考えている。例えば、より使い易い言語環境を提供する為 setofなどの高階機能を持つ述語を実装する場合の支援機構や、更に能率の良い共有方式を実装すること等で、その時は構造メモリの処理負荷が変化するので、アーキテクチャもそれに適合させる必要がある。これについても今後検討を行う予定である。

ＰＩＥに構造データ共有方式を実装した場合、構造メモリをいかに構成するべきかについて述べてきた。現在まで提案されている並列マシンのアーキテクチャには、ＰＩＥ－Ⅱのような集中型構造メモリを持つものは見受けられない。しかし効率と簡潔さを念頭に置けば、ある程度SMを集中化する方が総合性能が向上するものと思われ、その最適共有台数を求めることが課題である。今までのところ、１６台程度が良いという結論を得ているが、今後更に多くのデータを収集する必要があろう。これらを基に内部構成の設計を進めていきたい。

謝辞

本研究を行うにあたり、いつも深夜まで熱心に議論し、貴重な助言をして頂くＳＩＧＩＥのメンバー諸氏に感謝致します。

<< 参考文献 >>

[１] 後藤，相田，丸山，湯原，田中，元岡，
“高並列推論エンジンＰＩＥについて”，The Logic Programming Conference '８３，ＩＣＯＴ，March，１９８３．

[２] 平田，相田，後藤，田中，元岡，
“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”，電子通信学会技術研究報告 ＥＣ８３－３８，１９８３．

[３] 相田，田中，元岡，
“高並列推論エンジンＰＩＥの階層的構成法”，情報処理学会第２９回全国大会 ２Ｂ－４，１９８４．

[４] 小池，相田，田中，元岡，
“ＰＩＥの試作ＵＰの性能評価”，情報処理学会第２９回全国大会 ２Ｂ－６，１９８４．

[５] Cohen, J.,
“Garbage Collection of Linked Data Structures”, ACM Computing Surveys, Vol. 13, 3, Sept.１９８１.

[６] Goldberg, A. and Robson, D.,
“Smalltalk－８０：The Language and Its Implementation”, Addison－Wesley, １９８３.

[７] Deutsch, L. P. and Bobrow, D. G.,
“An Efficient, Incremental, Automatic Garbage Collection”, CACM Vol. １９, ９, Sept. １９７６.

[８] 中村，長谷川，雨宮，
“リスト処理向きデータフローマシン用構造体メモリの設計と評価”，電子通信学会技術研究報告 ＥＣ８１－３２，１９８１．

[９] Moto-oka, T., Tanaka, H., Aida, H., Hirata, K. and Maruyama, T.,
“The Architecture of A Parallel Inference Engine －ＰＩＥ－”, Proc. of Int. Conf. On FGCS, ＩＣＯＴ, Nov, １９８４.