論　文

高並列推論エンジンPIEの構造データ共有方式

正員　平田　圭二†　　正員　田中　英彦†

Structure Data Sharing Method of The Highly Parallel Engine PIE

Keiji HIRATA† *and* Hidehiko TANAKA†, *Members*

あらまし　高並列推論エンジンPIEは，ゴール書き換えモデルに基づき論理型プログラミング言語をOR並列に高速実行するマシンであり，将来の知識処理に於ける汎用機を目指している．ゴール書き換えモデルの実装方式を選定するにあたり，まず我々は基本処理単位を完全コピーによって生成する方式から出発し，そして漸次，適当な共有機構を取り入れて行くこととした．本論文で提案した構造データ共有方式とは，基本処理単位の持つ構造データの内，Ground Instanceの部分だけを基本処理単位間で共有する方式である．本論文では構造データ共有方式をソフトウェアシミュレーションによって評価検討し，その有効性を示す．本方式を導入すると，縮退時間や基本処理単位の転送時間が短縮され，処理速度は向上するが，オーバーヘッドとしてLazy Fetch操作，Ground Instanceの切り分け・格納操作，およびガーベジコレクションが必要となる．以上の各項目に関する詳細なシミュレーションを踏まえて，定量的な考察を行った．

1. まえがき

近い将来，知識処理システムの必要性は飛躍的に増大するので，高速な知識処理専用マシンの開発が急務である．知識処理専用マシンを含め，一般的に，あるプログラミング言語を高速に実行する専用計算機を構築する際に考慮せねばならない点の一つとして，そのプログラミング言語の持つ基本演算や基本操作を効率良く実行できるアーキテクチャを持つことが挙げられる．特に頻繁に用いられる基本演算や，データ構造に対しては十分考慮すべきである．しかし応用プログラムはプログラミングスタイルに大きく左右されることが考えられ，従って全体的な傾向を把握するためには多くの経験を積まねばならない．実用に耐え得るマシンとは，いかなるプログラミングスタイルを持つプログラムに対しても一定以上の性能を達成できるマシンであるとも言えよう．

現在我々の開発している高並列推論エンジンPIE[8]もこういった視点に立ち，アーキテクチャの構築を進めて来た．PIEはゴール書き換えモデルに基づき，論理型プログラミング言語に内在するOR並列性を最大限に活用しつつ高速に処理を行うマシンである．論理型プログラミング言語は一階述語論理に基礎を置き，記述力，検証能力，可読性等の点において従来の手続き型言語に優ると言われている[7]．この論理型プログラミング言語をより高速に実行するには，内在する並列性を最大限に抽出し，活用できる並列推論マシンが必要である．しかし，従来までに論理型プログラミング言語によって書かれた応用プログラムが，並列推論マシン上でどのような静的，動的挙動を示すかという観点から捉えた研究は，文献(1)，(14)に見られる程度で，いまだあまり多くない．例えば文献(1)では，OR関係数，述語の引数の個数といったPrologプログラムの静的な統計情報を測定している．

Lispに代表される関数型言語も人工知能システムの記述言語として良く用いられてきており，古くから逐次型，最近に至って並列型の専用マシンの研究が開始されている．Lispの場合はリスト構造の演算というレベルで考えると，実行モデルは言語仕様から比較的一意に決定できるという利点がある．Lispプログラムの静的・動的解析としてはClarkらの代表的な研究[2],[3]がある．この結果はLispシステムに適したガーベジコレクション方式の提案[4]等へと結びついている．

PIEのような論理型言語の並列リダクションマシンの研究にはICOTのPIM-R[15]，PIM-D[16]，富士通研の株分けモデル[17]，ALICE[18]，OR-Parallel Token Machine[19]等があり，世界的に見ても端緒に就

† 東京大学工学部電気工学科，東京都
Faculty of Engineering, The University of Tokyo, Tokyo, 113 Japan

いたばかりであり，応用プログラムとしてどのような種類のものが走り，どのような特性を持っているのか完全には把握されていないのが現状である．

我々は，実際のマシンアーキテクチャを仮定した上で，論理型プログラミング言語によって表現された応用プログラムの実行をシミュレートし，特に構造データのふるまいに着目し処理負荷やデータの流れ等を評価した．以下“構造データ”とは，ポインタによって結ばれたデータ一般のことを指す．本論文では，その評価・検討結果について述べ，更にそれらを処理モデルおよびアーキテクチャにいかに反映すべきかについても議論する．ここで行った対象プログラムの特性評価や議論等は他の推論マシンを開発して行く際にも有効なものである．以下の章では並列推論マシンにより論理型プログラミング言語を実行した際の構造データの動的ふるまいについて述べるが，論理型プログラミング言語にはpure Prologを仮定し，並列推論メカニズムのモデルには，現在我々が開発中のＰＩＥで採用した処理方式を仮定する．**2.**では並列推論エンジンＰＩＥの処理モデル，アーキテクチャの必然性について説明し，並列推論マシンにおける高速化手法の一つである構造データの共有を導入した経緯について述べる．そしてマシンアーキテクチャを構成して行く上でどのようなデータを収集すべきかを検討する．**3.**ではシミュレーションに使用したいくつかのテストプログラムが本質的に持っている性質を示した後，今回採用した実行モデル上で，構造データに関する振る舞いを実際にソフトウェアシミュレータを用いて調べ，その測定データを解析する．**4.**では収集したデータを踏まえて，高速化の可能性がどの部分に内在しているかを議論し，それを引き出すのに適したアーキテクチャを示唆する．

2. 高並列推論エンジンＰＩＥのアーキテクチャと構造データの共有

2.1 ゴール書き換えモデル

高並列推論エンジンＰＩＥで採用されているゴール書き換えモデルのＯＲ並列処理について概観する．論理型プログラミング言語の実行を論理式の導出過程に忠実な書き換え操作として捉えたモデルがゴール書き換えモデルである[5]．ゴール書き換えモデルでは一つの中間ゴールから次の中間ゴールを導出する過程が，パターンマッチおよび変数の束縛を行う単一化と，それ以降の処理にとって不要なデータを取り除く縮退という二つの操作から成る．従ってこのモデルでの中間ゴールは次の単一化に必要十分な環境しか持ち合わせておらず，論理型プログラミング言語固有の解探索機構を実現するためのバックトラック処理に必要な情報は存在しない．しかし逆に，一つのゴールとそれに対応する複数の定義節から複数の中間ゴールを同時に導出すること，すなわちＯＲ並列処理を行うことで非決定性が実現できる．つまり，このモデルの実行過程に縮退操作を取り入れることで，ＯＲ並列性との整合性が明確化する．各中間ゴール同士は論理的には独立であるから，並列に処理することに関しては何の副作用も生じない．

2.2 ＰＩＥのアーキテクチャ

ＰＩＥは上で述べたような処理モデルを効率良く実現するためのアーキテクチャを持っている（図１）．マシン内においては，中間ゴールが基本処理単位となり，ＡＮＤ関係で結ばれたゴールリテラルの集合（ゴールフレーム，ＧＦ）として表現される．複数の推論ユニット（Inference Unit，ＩＵ）はネットワークで結合されている．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，すなわちプログラムを格納する．ＵＰでは単一化・縮退を行う．縮退操作は，不要となった変数や参照されなくなったデータ部分を除き，親ＧＦや定義節の持つ必要な構造データ部分のみをコピーすることにより，新しいＧＦを生成する．ＭＭはゴールプールの役割をし，ＵＰは必ず自ＩＵ内のＭＭからＧＦを取り出す．新しいＧＦは負荷分散のために，ネットワークを通して他のＩＵに送出されることがある．各ＧＦは実際の推論過程を制御するために推論木上の各ノードに対応付けられており，ＡＣはその管理を行う．このようにＧＦ自体の処理と制御を分離することで柔軟な解の探索ストラテジの実現や，NotやGuardといった高階機能の実現が可能となっている．

2.3 基本処理単位と実装モデル

このようにＰＩＥでは，各ＩＵ内で親ＧＦから子ＧＦが作られ，ＩＵ間で転送されて処理が進む．ここで処理速度を大きく左右する実装上の問題点として，次の二つが挙げられる[6]．

(a) 基本処理単位のレベル

(b) 基本処理単位の内部表現

まず(a)の点については，上で述べたモデルをＰＩＥ上で

実現する場合，各IU間で転送される基本処理単位としてはリテラルや定義節など様々なものが考えられることを意味している．例えば基本処理単位を初期ゴールからのすべての情報というように非常に高くとれば，基本処理単位の制御は簡単になるかわりに，データ長が大きくなり，コピーや転送のオーバーヘッドが増大してしまうであろう．逆の場合はスタックの一部を転送するイメージに近く，制御のためのオーバーヘッド増加を招くであろう．さらに特徴的なのは，非決定性により，基本処理単位のレベルに応じた共有が生じることである．ここでいう共有とは定義節中のリテラルや環境の一部を共有することを意味し，この種の共有が生じるとアクセス競合や負荷の集中により処理速度の低下を引き起こす可能性があり好ましくない．(b)の点については，基本処理単位どうしが論理的に独立であっても，それを構成する構造データの物理的共有が考えられることを意味している．これによってOR並列処理によって生成された複数の処理単位中に出現した同じ構造データを，あるメモリ（PIEの場合はMM）に格納・共有し，基本処理単位の短縮を図り，コピーや転送のオーバーヘッドを軽減できる．しかしこの場合も構造データの切り分けやアクセスのオーバーヘッド等とのトレードオフを十分に考慮せねばならない．

実装モデルの選定は，直接マシン性能に影響をおよぼすだけに慎重に行わなければならない．まず逐次型推論マシン[7]においては，複数のスタックを用いて論理型プログラミング言語を実行する方式が代表的である．しかしスタックが伸長し，その一部を転送するというイメージをいきなりはじめから並列マシンに持ち込んでも，いたずらに制御を複雑にするだけで得策ではないと考える．従って我々は知識処理向き並列推論マシンの出発点において，まず制御の容易さを念頭に据え，基本処理単位として初期ゴールからの全ての情報という極端なものを設定した．この場合，縮退時には構造データの完全コピーを行うことになり，処理に必要な情報は完全に局所化されている．これに様々な共有メカニズムを導入して行くと，処理に必要な情報は分散化される．我々はこの構造データの完全コピーという点から出発し，先に述べた共有メカニズムによる利点とのトレードオフを考慮しつつ，それらを漸次導入して行き，最良の点に到達するという方針を取った．

完全コピー方式をとるとGFの中にはすべての環境および構造データが含まれる．このために解くべき問題によってはGFのサイズが非常に大きくなる可能性がある．GF長を短縮し効率良い処理を実現するために，まず先に述べた(b)の内部表現の共有について考える．一般に論理型プログラミング言語で取り扱う構造データ中には変数が存在する．変数に値の束縛が生じると，その変数を含む構造データは全く異なるものを表現することになる．変数を含む構造データをポインタで参照することにより共有すると，逐次型推論マシンで行われているようなmolecule（skeletonへのポインタと環境へのポインタの対）による表現が必要になる．書き込みの場合，すなわち変数に対して何らかの値の束縛が生じた場合，副作用を避けるためには必然的にその構造データをコピーしなければならない．また読み出しの場合，変数の本当の値を得るために「たぐる」という操作が必要である．しかし完全コピー方式では，変数には未定義のものしか存在しないので，

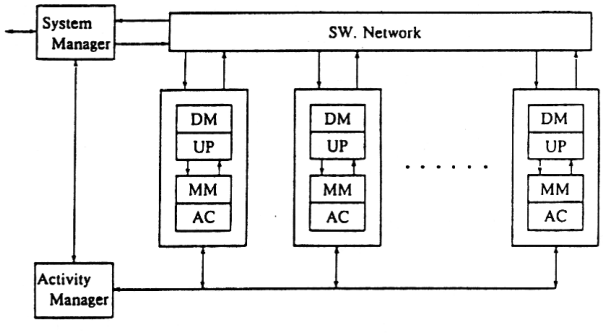


図1　PIE-Iのアーキテクチャ
Fig.1 Global architecture of PIE-I.

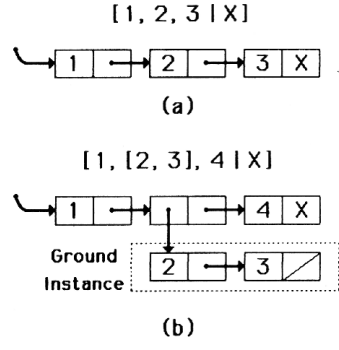


図2　Ground Instanceの例
Fig.2 Examples of Ground Instance.

たぐる操作は不要である．

2.4 構造データ共有方式

共有への第1段階として，未定義変数を含まない構造データ部分，すなわちGround Instance（GI）のみ共有することを提案する．未定義変数を含まない構造データには書き換えが生じないので，この部分を切り分け，あるメモリ中に置き去りにし，コピーや転送の対象から外せば処理の効率が図れるであろう．つまり単一化・縮退という本質的な処理には関係のない部分を選択的に持ち運ばないということである．GIを切り分け，ポインタで参照すると，OR並列処理の結果としてGIの共有が生じる．以降，この方式を構造データ共有方式と呼ぶ．ここでGIの例を示す．図2(a)のリスト構造にはGIが含まれないが，図2(b)のリスト構造には2ノード分GIが含まれている．構造データ共有方式を実装するためには二つの特別な操作が必要である．一つは単一化時におけるGIの読み出しであり，Lazy Fetch（LF）と呼ぶ（“Lazy”という名前は，PIEが基本的に完全コピー方式から出発しているため，必要なデータは“Eager”に持ち運ぶところに由来している）．もう一つは縮退時におけるGIの格納である．以下，例を挙げて説明する．図3のような単一化が行われる時，ゴール側のリテラルappの第1引数にはメモリ中の実体を指すポインタしかない．単一化を行う定義節側を見ると，ポインタを1段たぐり，そのcar部分とcdr部分を取って来る必要が生じる．この操作は単一化の最中に生じ，そのオーバーヘッドが直接マシン性能に影響する．PIEでは一つのゴールと複数の定義節間で次々と単一化を行うが，その時同じノードに対して何度もLFを行うことが予想される．そこで，1回当たりのLF操作のコストを低く抑えるため，実体の格納してあるメモリと，プロセッサの間にキャッシュのような役割りを果たすバッファを設け，LFしたノードは次の単一化に備えてここに蓄えることとし，これをLFバッファと呼ぶ．図4は，縮退時に新たな構造データが生成され，それがメモリ中に格納されるところを示している．リテラルprintの引数の構造データの変数Yに値が代入されGIとなるので，新ゴールの引数であるリストの1段目が切り分け・格納の対象となる．縮退に要する時間は単一化に要する時間よりも通常大きく，そこがボトルネックとなり易いので，GIの切り分け・格納のオーバーヘッドも極力抑えなければならない．

構造データ共有方式を導入することによりLF操作，GI切り分け・格納操作のオーバーヘッドは増加するが，コピーや転送のオーバーヘッドは減少する．従ってこれらオーバーヘッドをシミュレーションにより計量し，トレードオフとなる点を見い出せば良いことになる．

3. 構造データの動的ふるまい

3.1 シミュレーションプログラム

テストプログラムとしては実際の応用プログラムに見られる特徴をできるだけ良く反映しているものが望ましい．我々の用いたテストプログラムは覆面算（LL2P），8Queens（8Q），ペントミノ（Pent），数式簡単化（Equiv2）である．LL2Pや8Qは問題の規模が大きく並列度も高いがGIはそれ程多くは出

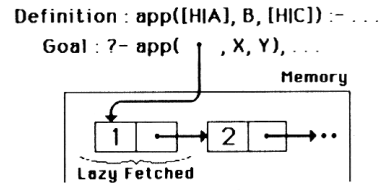


図3 Lazy Fetchについて
Fig.3 Lazy Fetch.

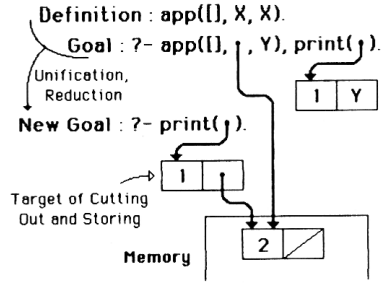


図4 Ground Instanceの切り分け・格納
Fig.4 Cutting out and storing of Ground Instance.

表1 シミュレーションで用いたテストプログラム

プログラム名	全GF数	推論木の深さ	静的OR関係数
LL2P	12155	65	1.33
8Q	127020	239	1.67
Pent	1270	244	8.63
Equiv 2	515	26	7.80

現しない．Pentの並列度はあまり高くないが比較的大きなGIが出現する．Equiv 2は問題の規模は小さいものの，並列度は高く，大きなGIが出現する．このような各プログラムの特徴は表1，図5に現れている．全GF数，推論木の深さで問題の規模や並列度が分かり，動的OR関係数で構造データの共有度が分かる．ここで静的OR関係数[1]とは，同じヘッドリテラルで始まる定義節が平均何個あるかを表し，動的OR関係数とは一つのGFが平均何個の子GFを生成したか（但し，子GFが0個のものは除いた）を表す．以上

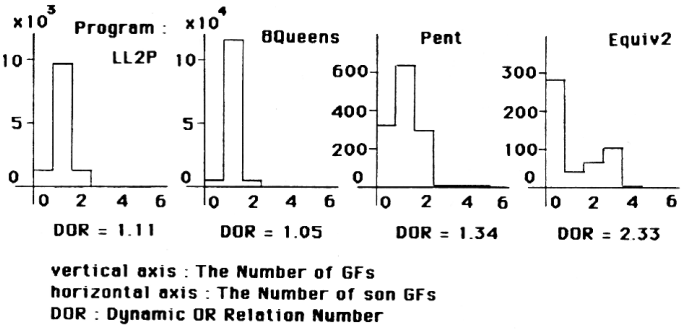


図5　動的OR関係数と分布
Fig.5 Distribution of dynamic OR relation number.

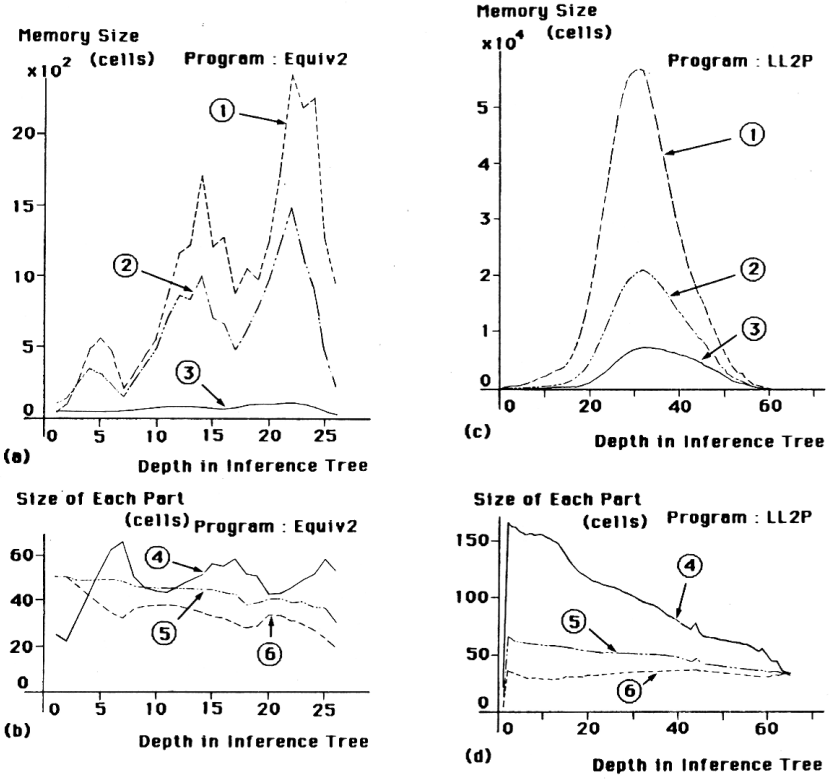


図6　必要メモリ量とGFの各部分のサイズ
Fig.6 The number of cells for each part.

四つのプログラムは並列度および共有率の評価用テストプログラムとしては適当であろう．

3.2 構造データの動的ふるまい

構造データ共有方式の有効性を確認するためソフトウェアシミュレータを作成し，種々のデータを収集した[20]．まず，実際にGIがどの程度共有されGF長が短縮しているかを調べる．図6，図7，表2，表3にその結果の一部を示す．作成し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図6のグラフは横軸を推論木の深さにとった．シミュレータでは1語1セルとしてあり，例えばアトムは1セルで，リストノードは2セルで表現できる．図6での①～⑥の各番号の意味は次のとおりである．①はGFの内，GI以外の部分を格納するために必要なメモリ量，②は構造データの共有を行わない場合のGIの総量，③は構造データ共有方式の場合のGIの総量，④はGFの内，GI以外の部分の大きさ，⑤は構造データの共有を行わない時にGFの持つ構造データ全体の大きさ，⑥は⑤の内GIの占める部分，である．図7はUPとMM間で転送されるGF長のヒストグラムである．(b)，(d)が構造データ共有方式を採用した場合である．各々⑥の分だけ転送量が減少している．表2は全て平均の値である．ここでGIの共有率とは“見かけの延べセル数／実際に必要なセル数”と定義した．Equiv2はGF中でGIの占める割り合いが大きいので，構造データ共有方式による利得も大きい．図6においても共有率が高いので，②に対して③が非常に少なく，並列度の高い問題でもメモリを爆発的に使い切るようなことがない．⑥の分だけGFが短縮すると，GF長はおよそ1／2になる．LL2Pの場合は表2からGF中でGIの占める割り合いが比較的低く，共有率も低いことが分かる．従って構造データの共有を行っても多少の利得はあるものの，Equiv2程の大きな効果は得られていない．

こうして得られたデータと，文献(9)による試作単一化プロセッサのマイクロステップ数から，実際に処理速度の向上を評価した（表3）．処理速度は縮退時間とGFの転送時間がボトルネックとなって決まるものと仮定している．Equiv2の場合はセル数自体の数値はそれ程大きくないが，GIのGF全体に占める割り合いが非常に大きいため短縮率が大きく，処理速度はより向上する．これらの結果は，構造データの共有により処理速度が上昇したと言うより，大きなGIの出現する問題でも一定の性能を維持できると換言することもできよう．

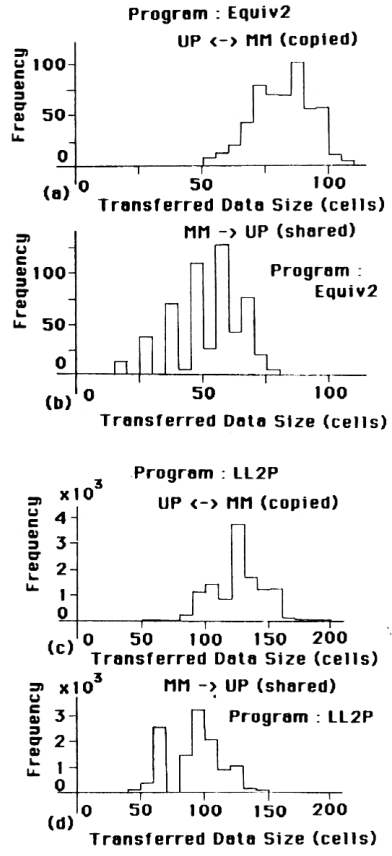


図7 UP-MM間データ転送量
Fig.7 Transferred data size between UP and MM.

表2 GF長とメモリ使用量

プログラム名	GFの内GIを除いた部分	GIのためのメモリ使用量	1GF当たりのGIの量	GIの共有率
LL2P	111	4240	22.69	2.61
8Q	48.5	3990	7.50	2.15
Pent	164	182	34.97	3.14
Equiv2	50.4	74.5	3.86	8.36

単位は全てセル数

表3 縮退操作に要する時間の短縮

プログラム名	1セル当たりの縮退μステップ	短縮したセル数	縮退時間の短縮率(%)
LL2P	5.27	52.4	54.5
8Q	5.40	15.7	39.6
Pent	6.29	109	43.8
Equiv2	4.61	37.7	65.1

3.3 Lazy Fetch操作

LF操作は単一化時に生じるので，できる限り高速に実行する必要がある．従ってメモリまでGIの一部を取りに行くようなコストの高いLF操作はできるだけ避けるために，LFバッファ（LFB）を設けた．このLFB導入の効果を評価したのが表4である．セル数0とは，LFが生じずに終了した単一化の回数のことである．問題によるばらつきはあるものの，1回の単一化当たりでおよそ1～2ノードしかフェッチして来ないことが分かる．LFBを設けたことでLF回数はおよそ1／10と著しく減少し，LF操作のコストは十分に低く抑えられる．

3.4 Ground Instanceの切り分けと格納

GIの切り分けは縮退時に行わなければならない．先にも述べたように完全コピー方式では一般に単一化より縮退に時間がかかり，ボトルネックとなっている．従ってGIの格納は後からでも実行可能であることを考えると，切り分けだけは高速に実行しなくてはならない．GIの切り分けはpostオーダでしか行えないので比較的コストは高い．どれ位の速度でGIが生成されるかを測定した（表5）．またGIをメモリに格納する際にはGFがその格納番地を知らなくてはならない．従って，その格納アドレスを転送する手間が必要となるので1GF当たりいくつの格納アドレスを必要としているかも同様に測定した．表5から，GIの生成速度は比較的遅いことが分かる．これはGIの切り分けがある程度まとめて行えることを示唆している．1GF当たり生成するセル数というのは，逆に言えば1GF当たり生成されるゴミのセル数でもある．

3.5 ガーベジコレクション

LispやSmalltalkシステム等の構造データを扱うシステムでは，ガーベジコレクション（GC）にCPU時間の25～40％が消費されているという報告がある[10,11]．GCを行うとシステムの効率が落ちるので，慎重にその方式を決定しなければならない．このメモリにおけるGC方式としては印付けとコピー（Mark & Copy）による方法[12]と参照カウンタ[4]を用いる方法が候補として挙げられる．前者の方式を採用した場合はGF中の根ノードからたぐりを開始することになり，根ノードの検出方法およびその個数が問題になるであろうし，後者の方式を採用した場合は，参照カウンタ命令の転送が問題になるであろう．従ってこれらの計量を行った（表6）．この数値を見る限りでは参照カウンタ方式の方がやや有利に思えるが，メモリ内の処理負荷も考慮に入れねばならない．GCに関してはさらに詳細なマシンアーキテクチャを仮定した上で評価を行う必要があろう．構造データ共有方式によって生成されたGIは，単調増加的にその量が増えて行く性質を持つので，オブジェクトの寿命に基づくGC方式[13]との相性も考察の対象に入ると考え，構造データの各ノードの寿命を測定した（表7，図8）．どのテストプログラムについても，測定データから描いた寿命の分布はおおよそ指数的に減少していた．図8にその一例を示す．共有率の高い問題ほど，相対的に寿命は長くなる傾向にあるので，仮想記憶を導入した際には積極的に利用すべき性質である．

3.6 議　論

構造データ共有方式を用いない場合，縮退時間は成

表4　Lazy Fetch操作

プログラム名	セル数	LF回数	LFB†→メモリの平均アクセス回数††	LFせずに済む単一化の割合(%)
LL2P	0	13325	1.00	68.4
	2	6160		
8Q	0	133191	1.00	85.4
	2	22712		
Pent	0	5611	1.93	95.7
	2	490		
Equiv2	0	7666		
	3	43	1.20	96.4
	4	289		

全てLFB有りのデータ
† LFB：Lazy Fetch Buffer
†† LFの生じる単一化当たり

表5　Ground Instanceの格納

プログラム名	生成セル数	必要な格納アドレス数†
LL2P	1.81	0.62（3.00）
8Q	0.19	0.07（1.00）
Pent	1.05	0.27（6.20）
Equiv2	0.30	0.06（0.50）

カッコ内はピーク値
全て1GF当たりのデータ
† 1回の単一化当たり

表6　ガーベジコレクションのためのデータ

プログラム名	根ノード数	最適化無の参照カウント命令数	最適化有の参照カウント命令数
8Q	2.81	5.45	0.53
Pent	5.80	11.32	2.80
Equiv2	2.77	5.23	1.36

全て1GF当たりのデータ

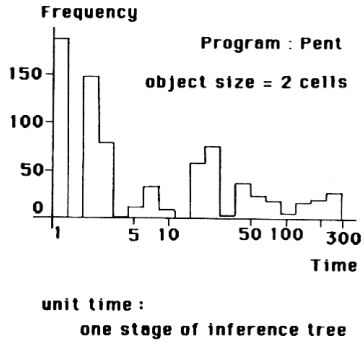


図8　オブジェクト（ノード）の寿命分布
Fig.8 Distribution of node lifetime.

表7　オブジェクト（ノード）の寿命

プログラム名	ノードの寿命	相対的寿命†
LL2P	12.5	19.2
8Q	39.2	16.4
Pent	29.1	11.9
Equiv2	11.0	42.3

推論木1段が単位時間
†推論木の全深さに対する割合（単位：%）

功した単一化時間の3～14倍である[9]．例えば単一化と縮退をパイプライン的に実行しようとすれば，1台の単一化ハードウェア，複数台の縮退ハードウェア，およびそれらの複雑な制御が必要になる．本方式を導入すると，表3より縮退時間が約1／2に減少するので，それだけハードウェアコストは減少し，制御も簡単になる．またGFサイズもプログラムの特性に依存しにくくなり，常に低いデータ転送量を維持できる．LFについては，LFバッファを設ければコストは十分低く抑えられることを確認した．単一化に要する時間は多少増加しても，縮退とのパイプライン処理に隠れてしまう可能性があるのに対し，GIの切り分けはマシンの処理速度に直接影響する．しかしGIの生成速度が比較的低いことを利用し，切り分け・格納を単一化数回ごとに1回行った場合，GFサイズは5セル程度（約10%）しか増加しない（表5，図7）．従ってこのトレードオフは動的に変化させることが望ましい．

4. むすび

我々が現在開発中の高並列エンジンPIEは，将来の知識処理における汎用機を目指している．従って種種の応用プログラムの特性を十分に把握した上で，それに適したPIEのアーキテクチャの構築を進めなくてはならない．本論文ではPIEにおいて大規模な構造データが出現した時でも高速な処理が維持できるように構造データ共有方式の導入を提案した．四つの異なる性質を持つプログラムに対して構造データ共有方式の理想的な場合での効率化の上限値を求め，次にオーバーヘッドの要因であるLF操作，GI切り分け・格納操作に関して各項目を測定した．その結果，LFバッファを設ければLF操作のコストはかなり下がること，GIの切り分け・格納は頻繁に行わずとも十分であること等を確認した．最後に構造データを扱うシステムでは避けて通れない問題であるGCについても，印付けとコピーおよび参照カウンタの2通りの方法を仮定し，簡単な比較やコマンドトラヒックの見積り等を行った．以上のシミュレーション結果より構造データ共有方式の有効性を示した．

次にこの方式を支援するハードウェアについて少し触れる．LF操作やGI切り分け・格納操作をUP側ですべて管理するのはあまりに負担が大きい．そこでGI格納専用メモリ（Structure Memory：SM）を導入し，上記操作を行えば良いであろう．SMを各IUごとに分散的に設置するか集中的に設置するか，およびUPやSMとのインタフェースについてもさらに今後シミュレーションを行い，議論を重ねる必要がある．

またGFの持つ構造データだけでなく定義節の持つ構造データにも着目すれば，本方式により一層の効果が期待できよう[15]．但し，効率良く定義節中の構造データを取り扱うためには，さらに別のアーキテクチャ上の工夫が必要となろう．

基本処理単位のレベルや，処理方式の異なる他の並列推論マシンにおいても，プロセッサ間で転送される基本処理単位中のGIを共有することができ，容易に効率化が図れる．その場合も本論文で述べたようなLF操作，GIの切り分け・格納操作が必要となり，同様の議論が適用できる．

謝辞　本研究を行うにあたり，常日頃より親身に御指導賜わった故元岡達教授に深謝いたします．またPIE研究プロジェクトSIGIEのメンバー諸氏には貴重なコメントを頂いたり，有益な議論をして頂いたりしたことを感謝いたします．データの収集には坂本光弘氏に協力して頂きました．

文　献

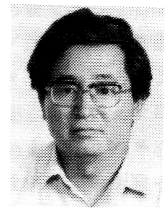
(1) 尾内，益田，麻生：“Prolog プログラムの静的解

析について”，第28回情処学前期全大，4H-3，pp.401-402（昭59-03）．
(2) D.W. Clark and C.C. Green：“An Empirical Study of List Structure in Lisp”, Commun. ACM, 20, 2, pp.78-87 (Feb. 1977).
(3) D.W. Clark：“Measurements of Dynamic List Structure Use in Lisp”, IEEE Trans. Software Eng., SE-5, 1, pp.51-59 (Jan. 1979).
(4) L.P. Deutsch and D.G. Bobrow：“An Efficient, Incremental, Automatic Garbage Collector”, Commun. ACM, 19, 9, pp.522-526 (Sept. 1976).
(5) 後藤，相田，田中，元岡：“ゴール書き換えモデルに基づく論理型プログラムの並列処理方式”，情処学論，25，3，pp.413-419（昭59-05）．
(6) 平田，丸山，田中，元岡：“高並列推論マシンにおける基本処理単位及び構造記憶に関する考察”，Proc. LPC'85 ICOT, pp.27-38 (July 1985).
(7) D.H.D. Warren：“Implementing Prolog - Compiling Predicate Logic Programs Vol.1, 2”, D.A.I. Research Report No.39, 40 (May 1977).
(8) T. Moto-oka, H. Tanaka, H. Aida, K. Hirata and T. Maruyama：“The Architecture of a Parallel Inference Engine-PIE-”, Proc. Intl. Conf. FGCS, ICOT, pp.479-488 (Nov. 1984).
(9) 小池，相田，田中，元岡：“PIEの試作UPの性能評価”，第29回情処学後期全大，2B-6，pp.89-90（昭59-09）．
(10) D.M. Unger：“Generation Scavenging：A Non-disruptive High Performance Storage Reclamation Algorithm”, SIGPLAN Notices, 19, 5, pp.157-167 (May 1984).
(11) D.M. Unger and D.A. Patterson：“Berkeley Smalltalk：Who Knows Where The Time Goes?”, Smalltalk-80：Bits of History, Words of Advice, Glenn Krasner (Ed), pp.189-206, Addison-Wesley (1983).
(12) H.G. Baker Jr.：“List Processing in Real Time on a Serial Computer”, Commun. ACM, 21, 4, pp.280-294 (April 1978).
(13) H. Lieberman and C. Hewitt：“A Real-Time Garbage Collector Based on The Lifetimes of Objects”, Commun. ACM, 26, 6, pp.419-429 (June 1983).
(14) A. Goto, H. Tanaka and T. Moto-oka：“Highly Parallel Inference Engine PIE-Goal Rewriting Model and Machine Architecture-”, New Generation Computing, ICOT, 2, 1, pp.37-58 (1984).
(15) R. Onai, M. Aso, H. Shimizu, K. Masuda and A. Matsumoto：“Architecture of a Reduction Based Parallel Inference Machine：PIM-R”, ICOT Technical Report：TR-105 (March 1985).
(16) N. Ito, H. Shimizu, M. Kishi, E. Kuno and K. Rokusawa：“The Dataflow-Based Execution Mechanisms of Parallel and Concurrent Prolog”, New Generation Computing, ICOT, 3, 1, pp.15-41 (1985).
(17) 久門，板敷，佐藤，増沢，相馬：“並列推論処理システム-改良型節単位処理方式”，第30回情処学前期全大，7C-8，pp.217-218（昭60-03）．
(18) S. Gregory：“Implementing PARLOG on the Abstract PROLOG Machine”, Research Report DOC 84/23, Dept. of Comp., Imperial College, London (Aug. 1984).
(19) S. Haridi and A. Ciepielewski：“An OR-parallel Token Machine”, TRITA-CS-8303, Royal Inst. of Tech., Stockholm (May 1983).
(20) 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”，信学技報，EC83-38（1983-12）．

（昭和61年1月31日受付）

平田　圭二

昭57東大・工・金属卒．昭59同大学院情報工学修士課程了．現在，同大学院博士課程在学中．並列推論マシンのアーキテクチャの研究に従事．情報処理学会会員．



田中　英彦

昭40東大・工・電子卒．昭45同大学院博士課程了．同年同大学講師．以来，ネットワークアーキテクチャ，分散処理，計算機アーキテクチャ，知識処理などの研究に従事．現在，同大助教授．工博．45年度本会米沢賞受賞．著書「情報通信システム」，「計算機システム技術（共著）」，「計算機アーキテクチャ（共著）」，「ソフトウェア指向アーキテクチャ（共著）」など．