ICOT Technical Report: TR-165

TR-165

並列推論マシン PIM-D の評価

伊藤徳義，来住晶介，久野英治
(沖電気)
六沢一昭(ICOT)

April, 1986

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

並列推論マシンＰＩＭ－Ｄの評価

The Evaluation Results of the Parallel Inference Machine PIM-D

伊藤　徳義*　　来住　晶介*　　久野　英治*　　六沢　一昭**
Noriyoshi Ito, Masasuke Kishi, Eiji Kuno, Kazuaki Rokusawa

* 沖電気
Oki Electric Industry Co., Ltd.

** (財)新世代コンピュータ技術開発機構
Institute for New Generation Computer Technology

概要

データフローモデルをベースとしたデータフロー方式並列推論マシンPIM-Dの実験機を試作した．そのデバッグを一通り完了し、いくつかの典型的なプログラムについて評価を行った．本稿ではこの実験機のアーキテクチャ、目標言語である論理型言語の処理モデル、及び評価結果について述べる．この結果によると、プログラムに内在する並列性をマシン上で実現できることが確かめられた．

1.　はじめに

第５世代コンピュータプロジェクトの一環として、並列推論マシンPIM-D(Parallel Inference Machine based on the Dataflow model)の研究開発を行っている．PIM-Dは、論理型言語をベースとした第５世代コンピュータ核言語の並列処理システムであり、処理方式の有効性の確認のため専用プロセッサを複数台接続した実験機を試作し、その評価を行った．PIM-Dの主な特徴は以下の通りである．

①　データフローモデルをベースとしており、論理型プログラムを本マシンの機械語に相当するデータフローグラフにコンパイルし、実行する．このため、プログラムに内在する並列性を容易に取り出すことができる．

②　２つの論理型言語（ＯＲ並列型言語、及び，ＡＮＤ並列型言語）をストリームベースの処理モデルで統一的にサポートする．

③　階層型バス接続網の採用により局所性を生かした並列処理を実現できると共にシステムの拡張性にも優れている．

④　構造データを格納する構造メモリをロジック内蔵のインテリジェントメモリとして実現し、機能分散化した．

実験機は複数の処理要素(PE)、及び複数の構造メモリ(SM)から構成されており、これらを階層型バス構造ネットワークにより相互接続している．各階層バスには最大８台までのユニットを接続することができ、バスに接続するユニットの数や階層レベルを変化することによってシステムの容易な拡張が可能である．各PEは内部に２台のマイクロプログラム制御演算ユニットを内蔵したパイプライン構造プロセッサであり、データフローグラフを並列処理する．SMはPE間で共有される構造データの格納、操作、管理を行い、PEからの構造データアクセス要求を処理し、その結果をPE側に返す．

実験機ハードウェア及びマイクロプログラムのデバッグを一通り完了し、いくつかの典型的なサンプルプログラムについて評価を行った．PE４台SM４台までの構成のマシンの性能評価結果によると、プログラムに内在する並列性をマシン上で実現でき、並列性の高いプログラムでは台数の増加に従って性能がほぼリニアに向上することが確かめられた．以下、PIM-Dの目標言語、処理方式、実験機のアーキテクチャ、及びその評価結果について述べる．

2.　目標言語

PIM-Dの目標言語は並列論理型プログラミング言語である．論理型プログラムに内在する並列性には大別してＯＲ並列性、及びＡＮＤ並列性がある．プログラムはゴール（質問）が与えられたときに起動される．このとき、与えられたゴールに対する解を求めるユニフィケーション処理は、ゴールと同一述語を持つ候補節（このような候補節の集合を定義と呼ぶ）との間で試行される．各候補節の間は論理和（ＯＲ）の関係にあり、ＯＲ並列性はこのユニフィケーションを候補節毎に並列

に実行することによって得られる．起動されたそれぞれのユニフィケーション処理をＯＲプロセスと呼ぶ．ＯＲプロセス群から得られる解の順序は非決定的である．一般には最も早く求められたものから順に、ゴール側へ返される．これに対して、ＡＮＤ並列性は論理積（ＡＮＤ）関係にあるサブゴールを並列実行することによって得られる．このとき、起動されたサブゴール間で共有変数を持つ場合はそれに具体化(Instantiate)された値の無矛盾性検査が必要となる．

PIM-Dでは、これらの並列性に対応した２つの言語をサポートする．１つはＯＲ並列型Prologであり、もう１つはＡＮＤ並列型Prologである．このうちＯＲ並列型Prologは、主として解の非決定的探索処理を並列に行う場合に使用される．即ち、いくつかの解の候補を並列に探索することによって処理の高速化を図る．このとき並列に実行される処理の間に相互干渉はなく独立に実行できる．

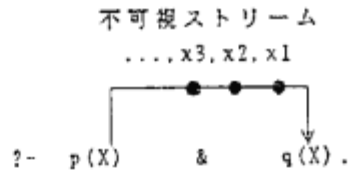
これに対して、ＡＮＤ並列型Prologはホーン節にDijkstraのガード付きコマンド［７］の機能を付与することによって、ＡＮＤゴール間で共有される変数を介したプロセス間非同期通信機能を提供している．このような言語としてConcurrent Prolog［13］,PARLOG［６］,GHC(Guarded Horn Clauses)［14］等が提案されている．このうちGHCは、Concurrent Prologよりも明快なセマンティクスと効率良いインプリメンテーションを提供し、しかもPARLOGよりも強力な記述力を持っており、ICOTで研究開発を進めている第５世代コンピュータの並列型核言語KL1(Kernel Language#1)のベース言語として選択された．以下、ＯＲ並列型Prolog及びGHCの処理方式について述べる．

３．　処理方式

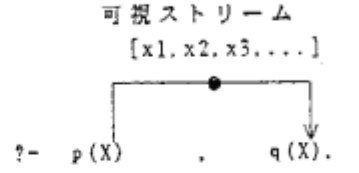
ＯＲ並列型Prolog及びGHCプログラムではいずれも、ゴールが与えられたときにそのゴールと同一述語を持つ定義が起動される．PIM-Dでは、各定義を１つの手続きとしてデータフローグラフにコンパイルする．データフローグラフは、ノードと、それらを結ぶ有向アークで表現される．ノードは、実行すべきオペレーションを示しており、有向アークはノード間で受け渡されるオペランドの従属関係を示している．各ノードはその入力アークに全オペランドが揃ったときに起動され、オペレーションコードで指定された演算を行って、結果を出力アークの指すノードに渡す．これによって次のノードが起動され同様な操作を繰り返す．このように、演算の結果がオペレータの入力オペランドにのみ依存するという関数型の特性を持っているために、オペランドの揃ったオペレーションが複数個存在すればそれらの並列実行が可能となり、並列処理の実現が容易となる．

PIM-Dでは上述２つのタイプの論理型言語を統一的にサポートしている．いずれの言語においてもプロセス間通信は、ストリームを介して行われる．

ＯＲ並列型Prologでは、各ゴール呼出しは解の候補列を生成する．このとき解をゴール側へ返す操作は非決定的に行われる．即ち、ＯＲプロセス群を並列に実行しながら、得られた順に解を返すことができる．このような制御を実現するために、解の集合をストリームとして表現し、その操作を行うためのストリームマージ・オペレータを用意した［10］．ここでのストリームはプログラマには直接見えないことから“不可視ストリーム”と呼ばれる．図１(a)に、ゴールリテラルp(X)の実行によって変数Xに具体化された値（インスタンス）の列を次のリテラルq(X)に渡す場合の様子を示す．p(X)側はストリームの生産者プロセスであり、その実行によって得られた解（変数Ｘのインスタンス）は非決定的にストリームに追加される．q(X)側はストリームの消費者プロセスであり、ストリームから次々とＸのインスタンスを取りだして処理を行う．これによって、２つのプロセスの間のパイプライン処理が実現できる．



(a)　ＯＲ並列型Prologにおけるプロセス間通信



(b)　GHCにおけるプロセス間通信

図１　ストリームを介したプロセス間通信

これに対して、GHCではプログラマがストリームの内容を意識しながらコードを書く（ストリームの内容はプログラマに陽に見える）．従って、このようなストリームを“可視ストリーム”と呼ぶ．プログラマはストリームを、未定義変数を含む構造データとして表現し、ストリームの生産プロセスや消費プロセスを記述する．図１(b)に、ゴールリテラルp(X)によって起動されるプロセス

がリスト表現のストリームを生成し、リテラルq(X)によって起動されるプロセスがこのストリームを消費する場合の例を示す。この場合、後述するようなガード機構を用いた排他的制御により、変数Xには高々１つのインスタンスしか具体化されない。

以下、これらの論理型言語の処理の詳細について述べる。

（１）　ＯＲ並列型Prologの処理方式

ユニフィケーション処理において問題となるのは構造データの処理方式である。PIM-Dの目標とするような知識情報処理の応用においては複雑な知識（構造データ）の処理が必要である。このような処理においてゴール呼出しの度に構造をコピーする方式は構造コピー操作のためのオーバヘッドやコピー領域のためのメモリ使用量が大きくなり現実的でない。従って、PIM-Dでは基本的にプロセス間で構造を共有し、必要に応じてコピーする遅延コピー方式を採用した。このような処理をマルチプロセッサ環境で実現すると、場合によっては共有構造データ参照のための遠隔アクセスが頻繁に発生する恐れがある。このとき、遠隔アクセスの応答時間はネットワークの遅延時間や構造データ側のプロセッサの混み具合の影響を受け、必ずしも短時間で結果が返るとは限らない。従って、各プロセッサは構造参照の結果を待つことなくプロセスを切り換えて他のプロセスの実行を開始する必要がある。

このような処理を従来の逐次型プロセッサで実現するとプロセスの頻繁な切り換えによって極端な性能低下をもたらす恐れがある。前述のようにデータフローマシンは、命令の間の独立性を保証しておりこのような分散処理環境に適している［１］［３］。従って、PIM-Dでは基本的にプロセス間で構造を共有し、必要に応じてコピーする遅延コピー方式を採用した。

各手続きはゴールが与えられたときに起動され、ユニフィケーション処理を開始する。このとき、図２に示すようにゴール引数を手続きに渡すことによって手続きに含まれる全ての節のユニフィケーションを並列に起動する。各々の節のユニフィケーションは引数毎に独立に実行される。即ち、ＯＲ並列処理を実現すると共に、引数毎の並列ユニフィケーションを実現している。このとき、問題となるのは各引数毎のユニフィケーションの間で共有される変数の取り扱いである。PIM-Dではこのような共有変数が具体化されたときのみその無矛盾性検査(consistency checking)を行う。このため、共有変数と非共有変数を明確に区別して取り扱っており、ユニフィケーションを行うオペレータは共有変数が具体化されたときのみその結合環境を生成する。以下、具体例を示す。

例えば、以下のような節が与えられたとする（シンタックスはDEC-10 Prolog［５］に準拠する）。

```
p(X,a,b) :- ...
```

ここで、記号':-'は含意を示しており、その左辺p(X,a,b)を頭部リテラル、右辺を本体と呼ぶ。pは述語名であり、Ｘは変数、a及びbは定数である。この節は図３に示すように、ゴール引数と頭部リテラル引数の間のユニフィケーションを行うデータフローグラフにコンパイルされる。

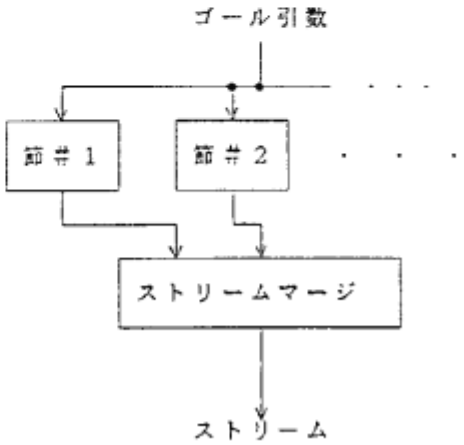


図２　ＯＲ並列処理の実現

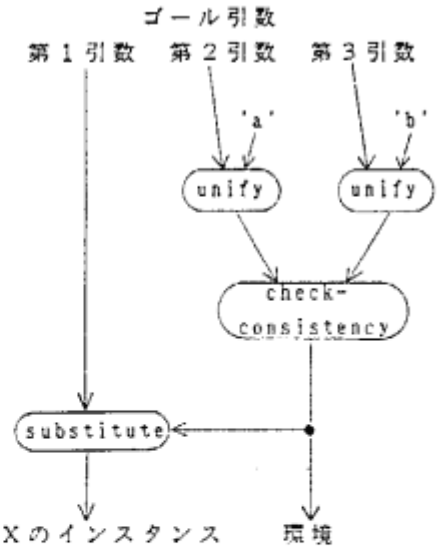


図３　ユニフィケーションのグラフ表現

図３で示される２つのunifyオペレータは、それぞれ、第２引数及び第３引数のユニフィケーションを行う。このオペレータはゴール引数が共有変数であるときのみその結合環境を生成する。それ以外のとき、ユニフィケーションが成功すれば

空の結合環境を生成するし、失敗すればその旨を示す特殊記号'fail'を出力する。第１引数に関しては、変数と任意の項とのユニフィケーションは常に成功することから、unifyオペレーションは省略可能であり、ゴール引数は直接次の処理へ渡される。unifyオペレータの結果は、次のcheck-consistencyオペレータへ渡され、２つのユニフィケーションの間の無矛盾性検査に使用される；即ち、双方の環境が共有変数の結合情報を持つときのみ、共有変数に具体化された値の間に矛盾がないか否かを調べ、新しい環境を生成する。生成された環境はゴール引数に含まれる共有変数の置換のためにsubstituteオペレータへ渡される。

この手続きに対して、以下のような共有変数を持つゴールが与えられたとする。

```
?- p(f(...Y...),Y,Z).
```

この例ではゴール第１引数と第２引数の間で変数Ｙを共有している。従って、ゴール呼出しを行う前に変数Ｙは共有変数Ysに変更される。ここで添字sは共有変数であることを示している。前述のように構造の遅延コピー方式を採用しているために、引数として現われる構造 f(...Ys...)は、プロセス間で共有される。即ち、図３の手続きに渡されるゴール第１引数はこの構造へのポインタとなる。

ゴール第２引数に対するunifyオペレータは、共有変数Ysが定数項aに具体化されたことを示す環境[Ys=a]を生成する。これに対して第３引数のユニフィケーションを行うunifyオペレータはゴール引数が非共有変数であるため、空の結合環境[ ]を生成する。check-consistencyオペレータはこれらの環境をもとに無矛盾性検査を行う。この場合の処理は単純であり、単に頭部ユニフィケーションで得られた環境[Ys=a]をsubstituteオペレータに渡す。substituteオペレータはこの環境をもとにゴール第１引数の共有変数Ysを置換し、新しい構造 f(...a...) を生成する（このとき構造のコピー操作が行われる）。ここで注意すべきであるのは、この置換操作が行われるのは第１引数に共有変数が含まれ、しかもunifyオペレータにより生成された環境が空でない場合のみである。このため、PIM-Dでは構造を示すポインタに共有変数を持つ構造とそうでない構造の間の区別を行うためのタグを用意している。subustituteオペレータはこのタグがオンで、しかも環境が空でないときのみ置換操作を行い、それ以外の場合は構造へのアクセスを行わない。

ゴール側へ値を返す処理はストリームを介して行われる。このとき各ストリーム要素（解）は頭部リテラル引数のインスタンスと節で得られた環境から成るリストとして表現される。例えば、前述の節が

```
p(X,a,b).
```

のように本体の存在しない単位節(unit clause)であれば、解を返すためのグラフは図４で示される。

同図においてＸのインスタンスは図３で示した頭部ユニフィケーションで得られた結果であり、環境はcheck-consistencyオペレータによって得られた結果である。cons-listオペレータはその２つの入力をもとにリストセルをSMに生成し、そのポインタを返す。append-streamオペレータは得られたリストをストリーム要素としてストリームに追加する。

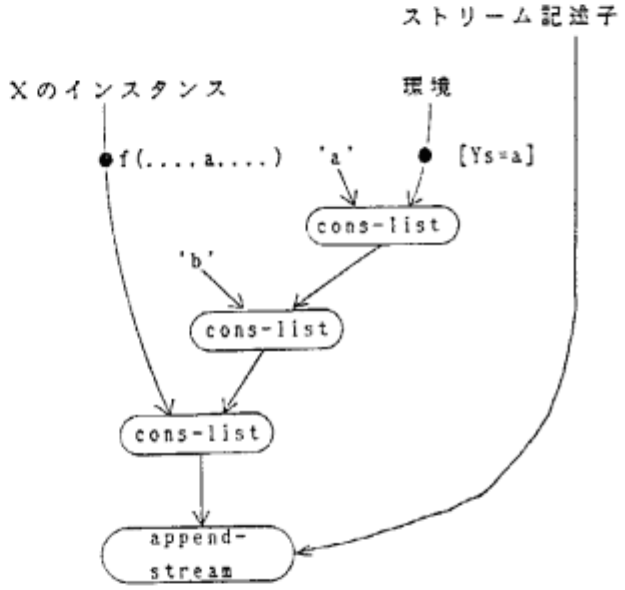


図４　ゴールへの解を返す操作

ストリームの制御はI-structure［３］と同様に行われる。ゴール呼出し時に、起動されるＯＲプロセス群によって得られる解の集合を格納するストリームの記述子が生成される。ストリーム記述子は、ストリームの先頭を指すポインタ SHP (Stream Head Pointer)と最後尾を指す STP (Stream Tail Pointer)から構成される［11］。生産者側（起動されたＯＲプロセス）は、上記append-streamの実行によって得られた解をSTPの指す位置に追加してSTPの値を更新する。一方、ストリームの消費者側（結果を待つゴール）は、SHPの指す先頭位置から順次、解を取り出す。

（２）GHCの処理方式

GHCプログラムの場合もほぼ同様なデータフローグラフにコンパイルされる。GHCの場合、ガード付きコマンドの機構を用いてプロセス間の非同期通信機能を提供している。プログラムは以下の

例に示すように、ガード節から構成される。

```
app(X,Y,Z) :- X=[] | Z=Y.
app(X,Y,Z) :- X=[H|U] | Z=[H|V],app(U,Y,V).
```

ここで、"|"は一種のＡＮＤ制御オペレータであり、コミットオペレータと呼ばれる。その左辺をガード、右辺を本体と呼ぶ。コミットオペレータはガード条件が成立した節の中から排他的に１つの節を選択する役割を果す。一般には、最も早くガードの成立した節が選ばれる。選択された節はその本体を新たなゴールとして実行する。GHCにおいては、このガード機構を用いてプロセス間の非同期通信手段を提供している。即ち、ガードにおけるゴール変数に非変数項を具体化するユニフィケーションは禁止される。このようなユニフィケーションは、ゴール変数が変数を共有する他のプロセスで具体化されるまで待ち状態となる。例えば、上記のガードにおけるユニフィケーションX=[]やX=[H|U]は、変数Ｘが具体化されるまで待たされる。変数Ｘが具体化されると、それぞれ空リストであるか否かのテストを行う。これによって、いずれかの節が排他的に選択される。

これに対して本体におけるユニフィケーションではこのような制約はない。例えば、上記の例で第２節が選択されたとすると、その本体におけるユニフィケーションZ=[H|V]は、Ｚが未定義であれば未定義変数Ｖを含むリスト[H|V]に具体化する。この操作と並行して変数Ｖを第３引数として手続きの再帰呼出しを行う（これによって、リストの残りを示す変数Ｖがさらに具体化される）。このようにし、このプログラムは第１引数（変数Ｘ）で示されるリストが空になるまで消費し、第３引数（変数Ｚ）で示されるリストを生成する。このときリストは可視ストリームを実現する。

PIM-Dでは、上例の定義を図５で示すデータフローグラフにコンパイルする。このグラフはゴール第１引数の値に応じて節インデクシングを行って節の選択を行う場合を示している。この例ではインデクシングによって選択される節が常に１つに限定されるので排他制御を行うセマフォオペレータは省略されている。

４．　マシン・アーキテクチャ

PIM-D実験機は図６に示すように、階層構造ネットワークにより相互接続された１６台の処理要素(PE)、１５台の構造メモリ(SM)、及び１台のホストコンピュータから成っている[９]。各階層の間はバス（T-BUSと呼ぶ）で接続されている。T-BUSはNN(Network Node)内のBA(Bus Arbitor)で制御され、各T-BUSには最大８台までの下位ノードを接続することができる。低遅延接続網を提供するために、T-BUSのバンド幅は十分大きくとっており（112本の信号線から成る）、通常のパケットを１転送サイクルで送信可能である。T-BUSを介したパケット転送は非同期に行われる。このため、各NNには上位方向用及び下位方向転送用の２つのFIFO(First-In First-Out)メモリを用意した。

各PEは内部に２台のマイクロプログラム制御演算ユニットを内蔵したパイプライン構造プロセッサであり、データフローグラフを並列処理する。SMはPE間で共有される構造データの格納、操作、管理を行い、PEからの構造データアクセス要求を処理し、その結果をPE側に返す。

４台のPE及び４台のSMをT-BUSで接続したモジュールをクラスタと呼び、このクラスタをさらに上位T-BUSで接続し、階層化している。従って、階層レベルやT-BUSに接続するクラスタの数を変化することによってシステムの容易な拡張が可能である。

PEはさらに、いくつかの機能ユニットから構成される。PQU(Packet Queue Unit)は、PE内部のT-BUSを経由して送られてくるパケットのバッファであり、FIFOメモリ（容量16Ｋパケット分）を内蔵している。パケットは、その宛先命令アドレスとオペランドを持つ。ICU(Instruction Control Unit)は、PQUから受け取ったパケットをもとに命令の発火制御（オペランドが揃ったか否かの判定）を行う。発火した命令はI-BUS経由で次段の２つのAPU(Atomic Processing Unit)のいずれかに送られ実行される。この発火制御のためにハードウェア・ハッシュ回路を内蔵している。APUは専用タグ判定ハードウェアを備えたマイクロプログラム制御プロセッサであり、ICUから受け取った命令を解釈・実行し必要に応じて結果パケットや構造データ操作パケットを生成する。結果パケットはその宛先に応じて、T-BUS経由で自PEや他PEのPQUに送られる。構造データ操作パケットはSMに送られる。この他、PE内で局所的なデータを格納するためのメモリLM(Local Memory)があり、APUからアクセス可能である。

SMは、構造データを格納するためのメモリ及びマイクロプログラム制御プロセッサSPU(Structure Processing Unit)から成っており、構造データ操作パケットを解釈し実行する。構造データの読み出しのような応答を必要とする操作パケットの場合は、結果を要求元のPEに送るための結果パケットを生成する。

プロセスの割付け戦略には、プロセス識別子予約方式を採用している。即ち、各PEのLMには使用可能プロセス識別子テーブルがあり、プロセス（手続き）起動命令が実行されたとき、APUはこのテーブルからプロセス識別子を取り出し新プロセスに割付ける。この識別子はPE番号とPE内の局所番号から成っており、新プロセスには指定され

たPEが割付けられる．従って，テーブルに格納する全プロセス識別子のPE番号の比率を変化することによってプロセスの分散比率を可変にできる．例えば，テーブルの全識別子のPE番号を自PE番号に設定すれば新プロセスは全て自PE内で局所的に実行される．このような比率はシステム初期化時に設定可能であり，PE間の距離をもとにして決定される．テーブルはFIFOメモリとして実現されており，新プロセスの割付けられたPEからプロセスを起動したPE宛へ新たな識別子が補充される．この方式の利点は自動的な負荷バランス制御が可能なことである．即ち，実行可能プロセスを多く抱える忙しいPEからのプロセス識別子の補充は遅れる傾向にあるため，そのPEへの新プロセスの割付けは抑制される．

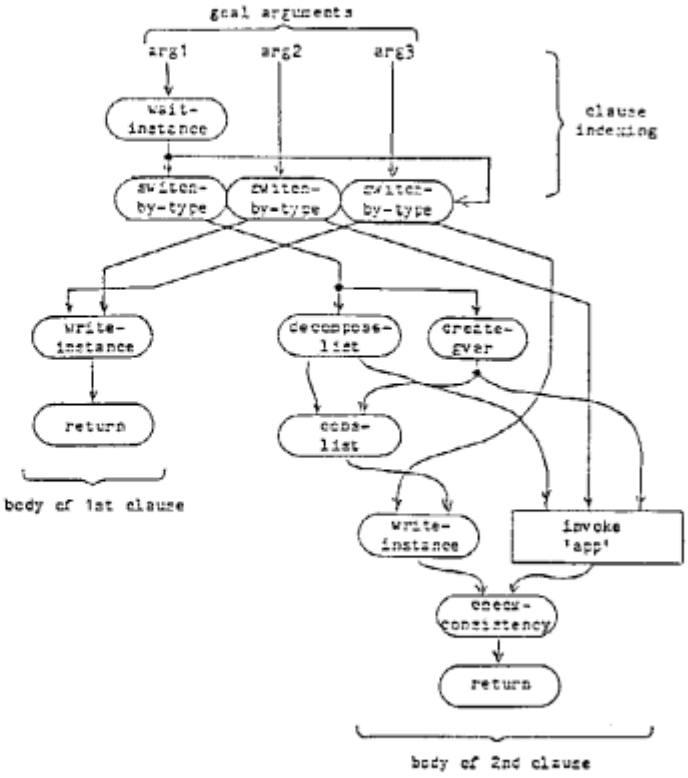


図5　GHCプログラムのグラフ表現

PE: Processing Element
  PQU: Packet Queue Unit
  ICU: Instruction Control Unit
  APU: Atomic Processing Unit
  LMU: Local Memory Unit
SM: Structure Memory
  SPU: Structure Processing Unit
  SMU: Structure Memory Unit
NN: Network Node
  BA: Bus Arbitor
  FIFO:First-In First-Out Memory

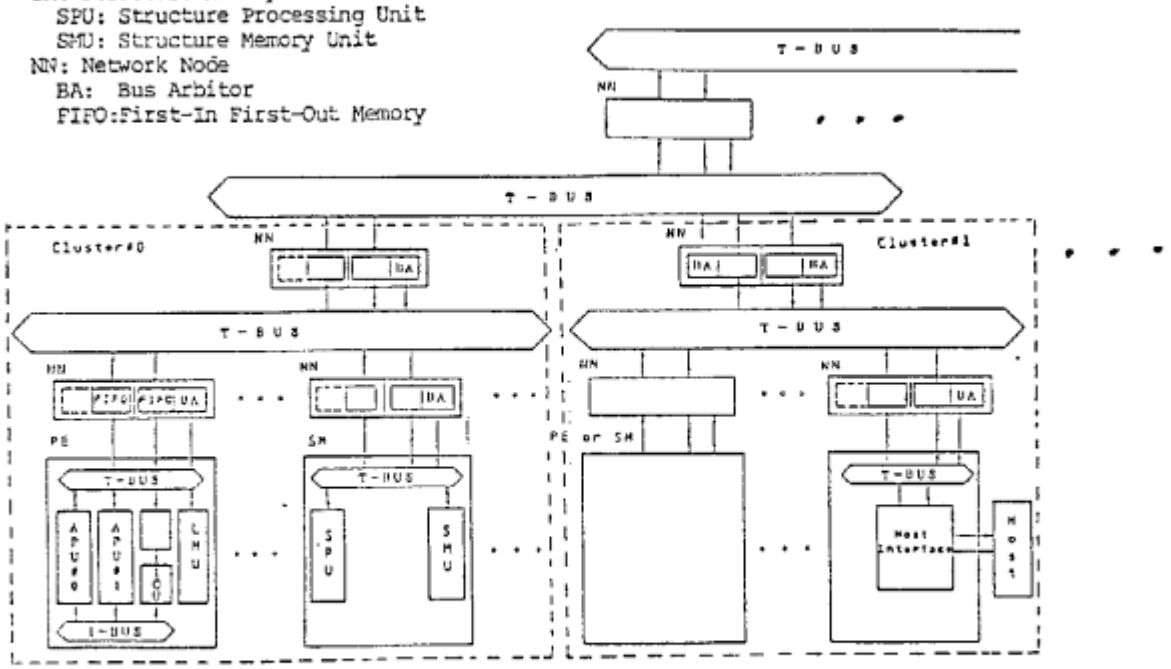


図6　実験機の構成

5. 評価結果

実験機の評価を行うためにいくつかのベンチマーク・プログラムを走らせモジュール(PE及びSM)の台数を変化しながら、各機能ユニットの稼働率、PEやSM台数を変化したときの台数効果等の計測項目について測定した。ベンチマーク・プログラムの一覧を表1に示す。

表1 ベンチマーク・プログラム一覧

言語	プログラム
OR並列型	7-queens
OR並列型	quick-sort
OR並列型	BUP
OR並列型	DCG
GHC	7-queens
GHC	quick-sort

このうち quick-sort は２５５個の数値要素から成るリストの分類を行うプログラムであり、 DCG (Definite Clause Grammar)は簡単な日本語文の解析木を作るプログラムである。 BUP(Bottom Up Parser)はこの解析をボトムアップ的に行うプログラムである。

表２は7-queensプログラムを実行したときの、PEの主要構成要素であるICU及びAPUの稼働状況をを示す。同表に示すICU及びAPUの入力待ちは、それぞれ、PQUからの命令待ち時間及びICUからの結果パケット待ち時間の割合を示したものである。実行時間はこれらのユニットにおける内部処理時間の割合を示す。ICUの出力待ちはAPUへの命令転送待ち時間の割合を示す。APUの出力待ちは、結果パケット生成やLMアクセスの際のT-BUS占有時間、及びT-BUSアクセス待ち時間である。

表２． ICU及びAPUの稼働率

ユニット	入力待ち	実行時間	出力待ち
PQU	１５％	３５％	５０％
APU	１４％	５１％	３６％

同表によるとICUの稼働率はAPUのそれに比して小さい。これは、APU（２台分）の処理能力が相対的にICUに比して小さいためにICUからAPUへの命令出力待ち時間が大きくなるためである。このため、APUの単体能力を増強するかAPU台数を増加する等の検討が必要である。APUでは出力待ち（T-BUS時間）の割合が比較的大きい。この理由は、T-BUSの占有権の制御を非同期に行っていること、及びAPUの内部バスの幅がT-BUSに比して小さくT-BUSへのパケット転送に数サイクル要することであり、改善の余地がある。

図７はPE及びSMの台数を変化したときの処理時間及び性能の推移を示す。性能は単位時間当りに成功したゴール呼出し回数RPS(Reductions Per Second)で示している。ソフトウェア・シミュレーション結果［12］で示したように内在する並列性の高いプログラム程、台数効果が顕著に現われる。例えば7-queensプログラムは台数の増加に伴ってほぼリニアに性能が向上するのに対して、並列性の小さいquick-sortプログラムは測定範囲内で既に性能がやや飽和する傾向にある。

6. おわりに

データフロー方式並列推論マシンPIM-Dにおける２つのタイプの並列論理型言語 （ＯＲ並列型Prolog及びGHC）の処理方式を示し、マシン・アーキテクチャを示した。マイクロプログラム制御の専用演算ユニット内蔵のパイプライン構造プロセッサを複数台接続した実験機を試作し、いくつかのベンチマーク・プログラムを実行させて、評価データを収集した。それによると、PIM-Dではプログラムに内在する並列性を十分引き出すことができ、プロセッサ台数を増加することにより性能がほぼリニアに向上することが確かめられた。今後、PE16台、SM15台規模の実験機の評価を行うとともに、大規模システム構築に向けての検討を進めていく予定である。

最後に、日頃御指導戴くICOT第４研究室の内田室長、御討論戴く後藤研究員、及びPIMグループ関係各位に深謝する。

〈参考文献〉

[1] Amamiya, M., Hasegawa, R., Nakamura, O., and Mikami, H., "A List-processing oriented Data Flow Architecture," Proceedings of AFIPS '82, pp. 143-151, June, 1982.

[2] Arvind, Gostelow, K.P., and Plouffe, W.E. "An Asynchronous Programming Language and Computing Machine," TR 114a, Dept. of ICS, University of California, Irvine, Dec. 1978.

[3] Arvind and Thomas, R.E. "I-Structures: An Efficient Data Type for Functional Languages," TM-118, Laboratory of Computer Science, MIT, 1980.

[4] Arvind and Innucci, R.A., "A Critique of Multiprocessing von Neumann Style," Proceedings of 10th International Symposium of Computer Architecture, June, 1983.

[5] Bowen, D.L., "DEC System-10 PROLOG User's Mannual," Dept. of Artifical Intelligence,

University of Edinburgh, 1982.
[6] Clark, K. and Gregory, S., "PARLOG: Parallel Programming in Prolog," Research Report DOC 84/4, Imperial College of Science and Technology, April 1984.
[7] Dijkstra, E.M., "A discipline of Programming," Prentice-Hall, 1976.
[8] Gurd, J.R. and Watson, I., "Data Driven System for High Speed Parallel Computing," Computer Design, June and July 1980.
[9] 来住、久野、六沢、伊藤、「データフロー方式並列推論マシン実験機のアーキテクチャ」 第３０回情報処理学会全国大会予稿集、１９８５．
[10] Ito, N. and Masuda, K., "Parallel Inference Machine Based on the Data Flow Model," Proceedings of International Workshop on High Level Computer Architecture 84, Los Angeles, California, May 1984.
[11] Ito, N., Shimizu, H., Kishi, M., Kuno, E., and Rokusawa, K., "Data-flow Based Execution Mechanisms of Parallel and Concurrent Prolog," New Generation Computing, Vol.3, No.1, 1985.
[12] Ito, N., Kishi, M., Kuno, E., and Rokusawa, K., "The Dataflow-based Parallel Inference Machine To Support Two Basic Languages in KL1," Proceedings of IFIP Working Conference on Fifth Generation Computer Architecture, UMIST(Manchester), July, 1985.
[13] Shapiro, E.Y., "A Subset of Concurrent Prolog and Its Interpreter," TR-003, ICOT, Jan. 1983.
[14] Ueda, K., "Guarded Horn Clauses," TR-103, ICOT, 1985.

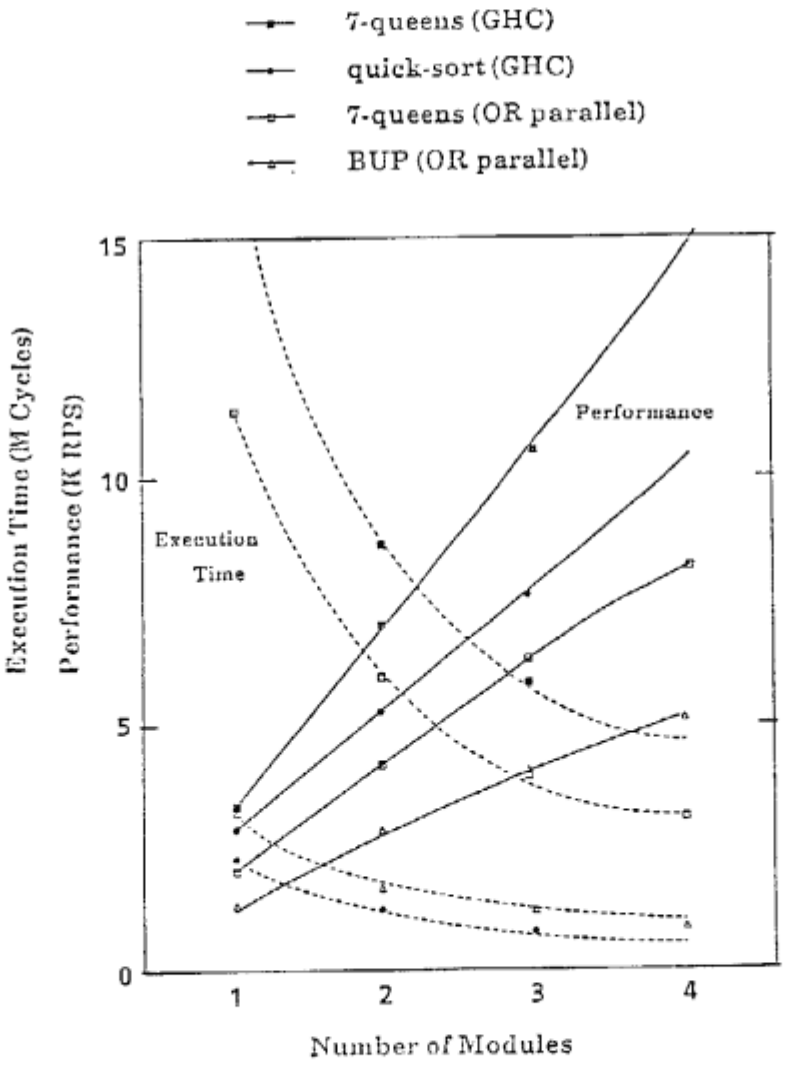


図７　実験機における性能の台数効果