ICOT Technical Report: TR-187

TR-187

並列推論マシン PIM-D における GHC 処理方式と実験機による評価

伊藤徳義、久野英治、(沖電気)
佐藤正俊 (ICOT)

June, 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における GHC処理方式と実験機による評価

佐藤 正俊* 伊藤 徳義** 久野 英治**

*新世代コンピュータ技術開発機構 **沖電気工業株式会社

ABSTRACT *A Paralell Inference Machine based on the Dataflow model (PIM-D) is investigated and its experimental system is constructed from multiple processing elements and structure memories interconnected by a hierarchical network. An execution model and its support mechanism for Guarded Horn Clauses (GHC), which is a stream AND-parallel logic programming language, are described. The preliminary evaluation results of the experimental machine are presented. The evaluation results show that the machine can exploit parallelism in program.*

1．はじめに

知識情報処理を目標とする第五世代コンピュータプロジェクトの一環として，ＩＣＯＴを中心に並列推論マシンの研究を行っており[Goto 85]，これまでいくつかの並列推論マシンの処理方式の検討を進めてきた[Onai 85],[Ito 84-1],[Kuno 85]。著者等はデータフロー方式に基づく並列推論マシン PIM-D (Parallel Inference Machine based on the Dataflow model)のアーキテクチャ及び処理方式を検討し，ソフトウェア・シミュレータ及び実験機による評価を行ってきた[Ito 85]。 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いる。
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つのタイプの論理型言語を統一的に実現している。
ＯＲ／ＡＮＤ並列型言語をデータフローグラフをベースとして統一的にサポートしている。
(3) ネットワークは階層型バスを使用している。
このためシステムの拡張性が高い。またプロセッサ間の距離を考慮した（局所性を生かした）負荷分散制御が可能となる。

これまで，ＯＲ並列を主体とした並列実行型Prolog（以下OR並列型Prologと呼ぶ）とストリームＡＮＤ並列に主眼を置いた Concurrent Prolog（ＣＰ）[Shap 83]の処理方式を検討してきた。本稿では新たにＡＮＤ並列型言語の一種である Guarded Horn Clauses（ＧＨＣ）[Ueda 85-2] をPIM-D 上にサポートしたので，実現上の特徴をこれまでサポートしている言語と比較して示す。さらにPIM-D 実験機による評価結果を示す。この結果から，PIM-D 実験機の処理方式の確認，並列処理の有効性の検証ができ，またこれらと同時に今後の課題も整理できたので示す。

以下，第２節でＡＮＤ並列型言語ＧＨＣの概略を述べ，さらに第３節でOR並列型PrologとＣＰとを比較しながらＰＩＭ－ＤにおけるＧＨＣの処理方式について整理する。第４節では実験機のアーキテクチャを示し，第５節で実験機についての評価を行ない，最後に改良と今後の課題について考察する。

2．ＧＨＣ

1984年の段階でOR並列型Prolog及びＣＰをベースに並列記述言語（核言語第一版KL1-84）の処理方式を検討し，評価してきた[Kuno 85]が，これらの評価を通して論理型プログラミングの強力な問題解決能力を示すことができたと同時に，核言語の記述能力と実行効率をより一層強化する必要があることもわかってきた。OR並列型Prologの場合，非決定的な探索問題の記述性に優れているが，対象とする問題によっては，積極的に実行制御を記述することが必要である。一方，ＣＰは並行プロセスの記述やその制御の記述に優れているが，例えば多環境の管理等が効率的な実装の点で難しいことがわかった[Ito 84-2],[Ueda 85-1]。

ＧＨＣはＣＰと同様，ＡＮＤ並列型言語に分類され，プログラムはガードつきHorn節から構成される。両者の違いは同期のためのメカニズムにあり，後述するようにＧＨＣではＣＰにおける多環境の問題が解決されている。またＧＨＣは，ＯＲ並列処理機能を言語のプリミティブとして持たないのでOR並列型言語に比べて機械語に近いレベルの言語であると考えてよい。このため，探索処理のようなＯＲ並列処理についても，積極的な探索の制御戦略とともに実効的にＯＲ並列処理をプログラミングすることができる。さらに，ある制限のもとで書かれたPrologの全解探索プログラムのようなＯＲ並列型言語のプログラムは，ＧＨＣのようなＡＮＤ並列実行型言語のプログラムに機械的に変換可能である[Ueda 85-3]ことも分かっている。

これらの理由より1985年の段階で核言語第一版のベースとしてＧＨＣを位置付けることにし(KL1-85)，さらに，これまで検討してきたKL1-84の検討結果をKL1-85にまとめるために新たにPIM-D 上にＧＨＣをサポートした。

以下，ＧＨＣのシンタックスを示す。
ＧＨＣのプログラムは，次の形をしたガードつきのHorn節(clause)の集まりである。

```
H   :-   G1,...,Gm   |   B1,...,Bn.
                                (m,n>=0)
```

ヘッド　　ガード部　トラスト ボディ部

受動部　　能動部

Hをヘッド(頭部)といい，G1,...,Gm をガード部(guard-part)と呼ぶ。ガード部はＡＮＤ関係','で結ばれたゴール列である。ヘッドとガード部をあわせて受動部 (passive-part)という。B1,...,Bn はボディ部(body-part) または能動部(active-part) と呼び，ＡＮＤ関係で結ばれたゴール列である。'|' はトラスト(trust) と呼び，論理的には',' と同様ＡＮＤ関係を表わす。また，ヘッドの名前と引数個数が同じであるような節の集合を手続きと呼び，その各々の節をゴールに対する候補節(candidate-clause)と呼ぶ。

３．実行方式

PIM-D で実行する論理型言語はデータフローグラフにコンパイルする。PIM-D はこのデータフローグラフを解釈実行する。データフローグラフはノードと有向アークからなる。ノードは実行すべきオペレータを示し，有向アークはデータ（オペランド）の流れる方向を示している。各オペレータはその入力アーク上に全オペランドが揃うと実行可能となり，その実行結果は出力アーク上に出力される。これらのオペレータの実行はその入力オペランドのみに依存するという関数型の特性を持っており（オペレータ間の独立性が保証されており），オペレータ群を並列に実行できる(Arvi 78)。

３．１　データフローグラフ上の特徴

PIM-D ではデータフローグラフにより論理型言語を統一的に扱っているが，論理型言語の持つ特性によりデータフローグラフ上の特徴が異なる。この特徴について，これまでサポートしてきたOR並列型Prolog，ＣＰと新たにサポートしたＧＨＣを比較し整理する。この特徴とはユニフィケーションと非決定性処理であり，ユニフィケーションではゴールの並列実行の際のそれぞれのゴールの環境の扱いであり，非決定性処理では言語による非決定性の違いより要求される制御の違いである。

３．２　ユニフィケーション

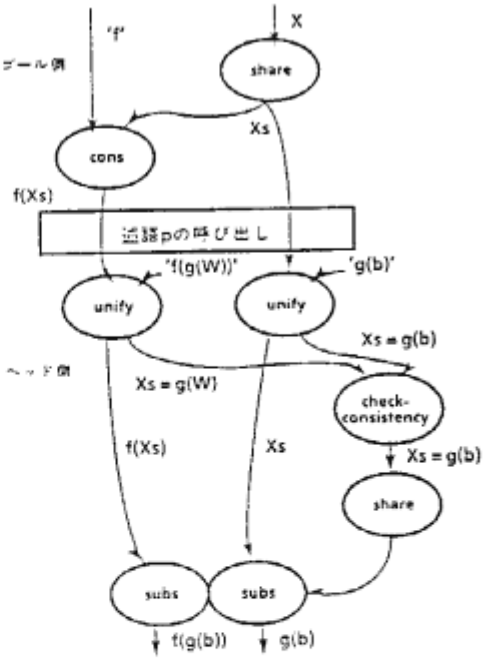
並列に実行するユニフィケーション間で共通な変数（例えば候補節をＯＲ並列で実行するときゴール引数側の変数は並列に実行されるユニフィケーション間で共通な変数であり，ゴール／引数をＡＮＤ並列に実行するときゴール／引数間にまたがる同一変数は共通な変数である）が含まれる場合，ゴール引数を全てコピーするか，部分的に共有して共有できない部分を管理する方法がある。PIM-D では後者の方法を採用し共有できない部分にそれぞれにローカルな結合環境を持たせている。またそのローカルな結合環境を持つ変数を共有変数として区別している。

OR並列型PrologやＣＰでは共有変数を含むユニフィケーションを並列に実行する場合，それぞれのユニフィケーションで共有変数に対する結合環境を生成し，このユニフィケーション間の関係がＡＮＤ関係であればこれらの結合環境間の無矛盾性のチェックを行なう必要がある。例えば以下のゴールと定義節が与えられたとする。

```
?- p(f(X),X).
p(f(g(W)),g(b)) :- ...
```

ここでゴールはその第一引数と第二引数の間で変数Ｘを共有している。PIM-D では引数間の並列処理を実現しており，まずゴール呼び出し前に変数Ｘを共有変数Xsに更新する。定義節ではその第一引数及び第二引数のユニフィケーションの結果として，それぞれ結合環境Xs=g(W) 及びXs=g(b) を得る。引数間の並列処理の場合，これらユニフィケーションはＡＮＤ関係にあり無矛盾性のチェックが必要である。つまりこれらの結果よりXsのインスタンス間の無矛盾性のチェックを行ない，環境Xs=g(b) を得る。このような処理は，ＡＮＤゴール間の並列実行においても同様である。

この例について概念的なデータフローグラフを図１に示す。




```
?- p(f(X),X)).
p(f(g(W)),g(b)) :- ...
```

図１ 共有変数を持つユニフィケーション
(OR並列型Prolog)

まずshare オペレータにより共有変数を生成しユニフィケーションを実行する．その結果各々の結合環境が生成されcheck-consistency オペレータにより結合環境間のユニフィケーションを行ない最終的な結合環境を得る．さらにこの共有変数に対する最終インスタンスが非共有変数を含む場合share オペレータにより共有変数を生成する．最終インスタンスが非共有変数を含まない場合はshare オペレータの処理は単純であり入力インスタンスをそのまま出力する．check-consistency で得られた最終結合環境によりsubstituteオペレータは結合環境に示される置換を行ない最終のインスタンスを得る．

これに対して，ＧＨＣにおいては受動部におけるゴール側変数の具体化は禁止されている．このような具体化を行なうユニフィケーションは，変数を共有する他ゴールにおいて非変数項に具体化されるまで待たされる．従って上述のような結合環境は生成しない．このようにＧＨＣでは複雑な処理を必要とする多環境の生成は不要であり効率良く実現できる．

ＧＨＣの場合を上記と同じ例で図２に示す．まず受動部におけるゴール変数と非変数項のユニフィケーションと待ち合せの機能をwait-and-unifyオペレータにより実現している．このオペレータはユニフィケーションの際，入力オペランドが具体化されているかチェックする．具体化されていればユニフィケーションを行ない，具体化されていなければそのオペランドが具体化されるまで待つ．check-consistency オペレータは単にユニフィケーションの成功／失敗のチェックのみとなる．

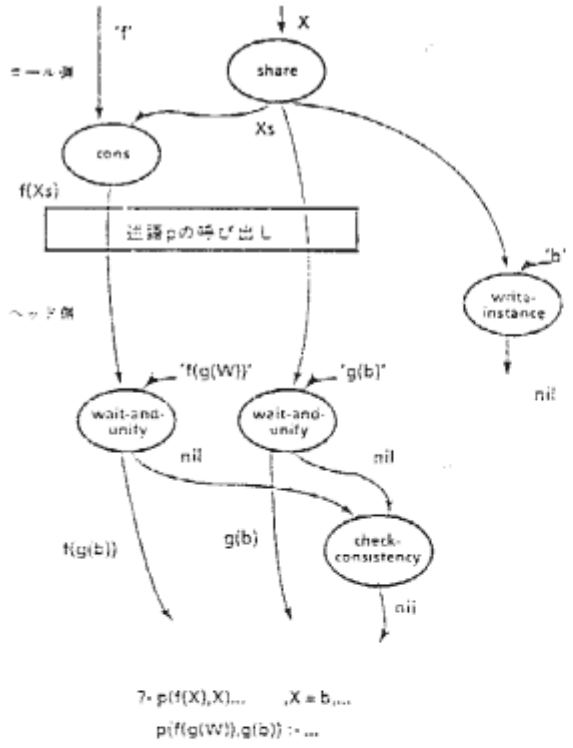


図２ 共有変数を持つユニフィケーション
(GHC)

３．３　非決定性処理

ＯＲ並列型PrologのようにＯＲ並列を主体として実行を行なう場合，１つのゴールに対する解が複数存在する可能性があり，それらを非決定的にマージする制御が必要となる．即ち，定義節側で非決定的に求められた解を，求められた順にゴール側で取り出し次の処理を進める．この非決定性を"don't know nondeterminism" と呼び，PIM-D ではこのような解の集合をストリームとして実現している．このストリームはプログラムには直接現れないことから invisible streamと呼び non-strictなデータ構造を用いて実現している．

ＣＰやＧＨＣは前述のトラストオペレータと呼ばれるDijkstraのGuarded command [Dijk 75] と同様な排他的制御機能を持つ．この排他的制御機能は非決定的であり最初にガード部を通過したＯＲプロセスのみが以降の処理を継続できる．この非決定性を"don't care nondeterminism" と呼び，特に探索型のプログラムの場合，プログラマは解の操作のためにストリームを陽に意識しなければならない．このストリームをvisible streamと呼ぶ．またＣＰにおいては節の頭部及びガード部の処理で得られた共有変数に関する結合環境はトラストオペレータの実行を契機に他プロセスに対して公開される．つまりＣＰのトラストオペレータは排他的制御の他に結合環境の公開も持つ[Ito 85]．

ＧＨＣにおけるトラストオペレータは上述のように受動部でのユニフィケーションがローカルな結合環境を生成しないため，ＣＰの場合のような結合環境の公開の機能は必要なく排他的制御機能のみを持つ．

４．マシンアーキテクチャ

データフロー・メカニズムに基づく並列推論マシンのアーキテクチャを検討する上でまず図３のような概念構成を考えた．この構成はデータフローグラフを解釈実行する処理要素ＰＥ（Processing Element）群と構造データの格納や操作を行なう構造メモリＳＭ（Structure Memory）群及びこれらを結ぶネットワークから構成される密結合マルチプロセサ・システムである．このように，構造データ処理機能を分離してＳＭとして実現した理由は我々の目指す応用分野が記号処理であり構造データを効率よく処理できると考えたためである．本マシンでは構造データ共有方式を採用しており，プロセス間で構造データを共有し必要に応じて（オンデマンドで）構造データの内容の必要な部分を参照するようにしている．この方式のメリットは以下の通りである．

- 構造データコピーのための処理オーバヘッドを軽減でき，構造データの重複格納によるメモリオーバヘッドを回避できることにある．
- ＰＥとＳＭとを機能分散することにより両者の並列実行が可能となり，処理時間の短縮が期待できる．
- ＡＮＤ並列実行時必要な共有メモリセルをＳＭのメモリセルとして実現できる．
- プロセスの割当てと構造データの割当てについてそれぞれ独立に行なうことができ，これらの割当て戦略がきめ細かく実現できる．

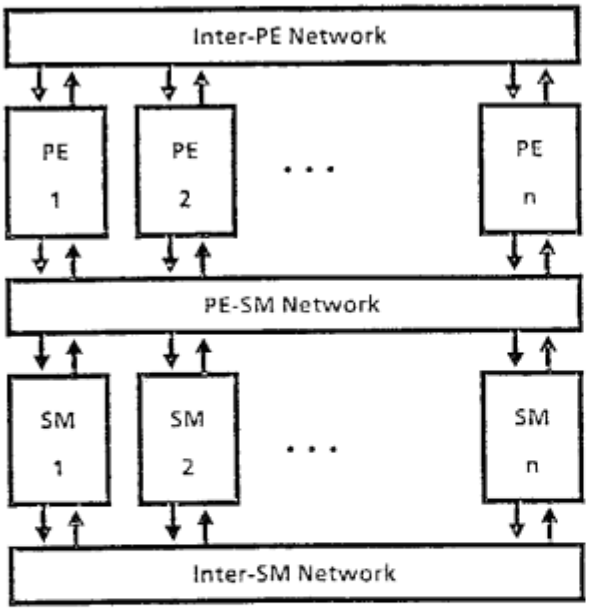


図３概念構成

４．１　構成の検討

概念構成を実現するためにソフトウェア・シミュレータにより
(i)　構成，処理方式の確認，
(ii)　並列処理の有効性の検証，
(iii) 実験機や言語仕様へのフィードバック，
を行なってきた。

ＰＥはPIM-D の主構成要素であり，オペランドの待ち合せ制御や命令実行制御を行ない必要に応じてＳＭに指令パケットを転送する。ＰＥは複数の機能ユニットから構成されるパイプライン構造プロセッサである。ＳＭは指令パケットを解釈し，構造データの読み書きや，メモリ割付け，不用メモリの回収等を行なうインテリジェントメモリである。

ネットワークはモジュール間のパケット交信に使用され，３つのネットワーク（ＰＥ間ネットワーク，ＰＥ－ＳＭ間ネットワーク，ＳＭ間ネットワーク）から成る。ＰＥ間ネットワーク，ＳＭ間ネットワークは二次元メッシュネットワークであり，ＰＥ－ＳＭ間ネットワークは等距離多段ネットワークである。このように３つのネットワークを仮定した理由はモジュール間のパケット交信のトラフィックが主に３種類（ゴール割当てパケット，構造データ読み出しパケット，ガーベッジコレクション制御パケット）考えられ，それぞれのトラヒック特性を考慮したためである。

その結果，構成・処理方式の確認，並列処理の有効性をソフトウェア・シミュレータ上で検証できた。特にネットワーク・トラヒックにおいて構造データ制御のトラヒックが支配的であり，その他のトラヒックはほぼ無視できる程度であることが得られ，実験機では実装上の問題を考慮してこれらネットワークを１つに統合し実現した。

４．２　実験機の構成

このソフトウェア・シミュレータでの評価を基に実験機を試作した。実験機の規模はＰＥが１６台，ＳＭが１５台とさらに１台のホストコンピュータから成り，ＰＥ４台，ＳＭ４台を基本構成単位（クラスタ）としてそれぞれが階層構造ネットワーク（T-BUS ）により相互接続している。その構成を図４に示す。全体の構成はソフトウェア・シミュレータでの評価を反映し，実装面からシンプルで拡張性に富む構成として実現している。

各クラスタは上位ネットワークと同じT-BUS で相互接続した最大８台までの構成要素から構成される。各T-BUS はバス・アービタを備えたＮＮ（Network Node）により制御される。低遅延通信網を実現するためにこの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），構成要素間のパケット転送を１転送サイクルで行なうことができる。このため負荷分散制御の負担を減少できる。またこのような階層型構成はクラスタ内の構成要素数を自由に変化させることで多くの実験環境を設定でき等距離ネットワークでの負荷分散制御や距離の異なる構成要素間のネットワークでの負荷分散制御等のデータ収集／評価が可能である。

以下この構成要素を順に示す。

(i) ＰＥ

ＰＥは以下のような機能ユニットから成る。

ＰＱＵ（Packet Queue Unit）：　パケットのバッファリング機能を持つFIFO（First-In First-Out）メモリである。

ＩＣＵ（Instruction Control Unit）：　ＰＱＵからのパケットを基に命令の発火制御を行なう。

ＡＰＵ（Atomic Processing Unit）：ＩＣＵから受け取った命令の解釈・実行を行なう。

ＬＭＵ（Local Memory Unit）：　ＰＥ内で局所的に使用されるデータを格納するメモリである。

それぞれの機能ユニットはパケットの送受によりパイプライン動作をする。ＩＣＵからの命令パケットはI-BUS経由で各ＡＰＵに送られ，ＡＰＵは必要に応じてT-BUSを介してＬＭＵをアクセスし，その結果をＰＱＵまたは他のＰＥへ結果パケットとして送る。また構造データのアクセスが必要な場合は，ＡＰＵはそのアクセス要求を指令パケットとして生成しT-BUS を介してＳＭに送る。このときアクセス要求に対する応答が必要であればＳＭはその応答を結果パケットとしてＰＥに送る。

(ii) ＳＭ

ＳＭはＰＥからの指令パケットを解釈・実行するＳＰＵ（Structure Processing Unit ）及び構造データを格納するＳＭＵ（Structure Memory Unit）から構成され，ＰＥからの構造データのアクセス要求に従い構造データの読み書きや，メモリ割付け，不用メモリの回収等を行なう。

各構造データ語には，語の内容が有効であるか否かを示すタグビットが存在し構造データの読み出し要求と書き込み要求との非同期待ち合せ機能が提供されている。タグビットは以下の状態を表わす。

empty　：語の内容は空である。
ready　：語に対する書き込みがすでに行なわれた。
pending：書き込みが行なわれる以前に読み出し要求

があった。
タグがempty である語に対して読み出し要求があるとタグは pending に設定され，読み出し要求は待ち状態となる。この語に対して書き込みがあった時点で読み出し要求は実行され，語のタグはready 状態にセットされる。このほかに，２語を単位とした各メモリセル毎に参照数フィールドを持ち構造メモリの資源管理は，これを用いて行なわれる。

(iii) ＮＮ
T-BUS のアクセス権を制御し，各パケットの送受を行なう。アクセス権を制御するＢＡ（Bus Arbitor ）及びパケットのバッファリングを行なうＦＩＦＯの２つの機能ユニットから構成される。

(iv) ＳＶＰ(Service Processor)
ＳＶＰはプログラムやデータのロードや結果の収集／格納の機能を持ち制御ホスト（VAX-11/730）とＶＩＭ（ＶＡＸ Interface Module ）からなる。
機能ユニットのハードウェア諸元を付録に示す。

５．評価

現在，実験機のハードウェア／ファームウェアデバッグをほぼ終了し１クラスタについて簡単なプログラムを実行しデータを収集している[Kuno 86],[Ooha 86],[Roku 86]。ここでは PIM-D 実験機の並列処理の有効性の検証と問題点の洗い出しのためにこれらのデータの評価を行なう。

５．１　性能及び台数効果

実験機の性能及び台数効果を図５に示す。この図はOR並列型PrologとＧＨＣのプログラムに対して，１クラスタ内のＰＥ及びＳＭを同数にして１から４台まで変化させたときのグラフである。ＣＰのプログラムについては現在データを収集していないがその性能についてはソフトウェア・シミュレータの結果によりＧＨＣに比べて1/2 ～1/3と予想される[Sato 86]。評価プログラムはOR並列型Prologプログラムとして７queens，Bottom Up Parser（ＢＵＰ），ＧＨＣプログラムとして７queens，quick-sortを選んだ。ここでＢＵＰは文法規則数が約３０程度からなる日本語文解析プログラムであり，quick-sortは整数 256 個を昇順に並びかえるプログラムである。性能はプログラムを実行する（全解を求める）のに要したマシンサイクル数と単位時間（１秒）当たりに成功したゴール呼び出し(Reduction）の回数であるＲＰＳ（Reductions Per Second）で示している。この図より性能が台数の増加に伴ってほぼリニアに向上していることが確認できる。また７queensプログラムにおいて，OR並列型Prolog版とＧＨＣ版を比較するとOR並列型Prolog版の方がＧＨＣ版より実行時間が短い（高速である）にもかかわらず見掛け上の性能（ＲＰＳ）が低い結果となっている。この理由は２つあり，第一にＧＨＣではOR の操作をユーザがユーザ定義述語として明示的にプログラムするため，OR の操作（ストリーム処理とそのゴール呼び出し）のオーバヘッドがOR並列型Prologに比べ大きく，このため共にデータフローグラフとして統一的に実現しているプログラムにもかかわらずOR並列

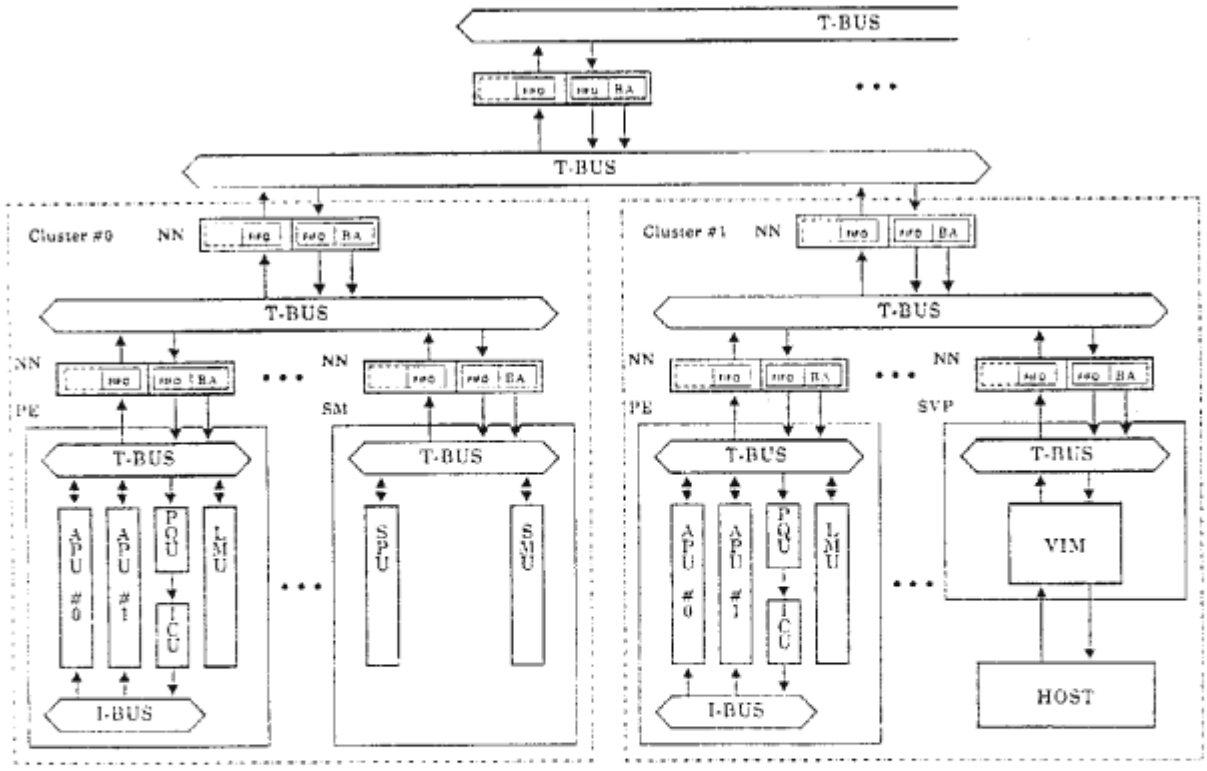


図４ 実験機の構成

型Prolog版の実行時間が短くなっている。これについてはOR並列型Prologのようにシステムプリミティブレベルのサポートの導入で実行時間の短縮が期待できる。第二にＧＨＣではOR並列型Prologののように言語レベルで全解探索の機能を持たず（OR並列型Prologよりゴールの実現する機能が低く），同じ処理を記述する場合ゴールの処理は軽い処理となる。これらにより見掛け上の性能（ＲＰＳ）的にはＧＨＣ版が高く現れる。

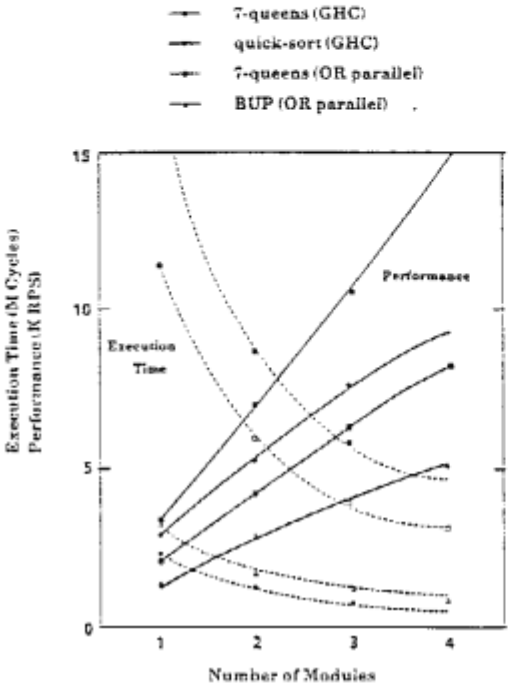


図５実機上の性能と台数効果

５．２　動特性

５．２．１　稼働バランス

ＰＥ台数を１～３まで変化させた場合のＩＣＵとＡＰＵについて稼働時間の変化の内分けを図６に示す。このデータはOR並列型Prolog版７queensを実行した結果である。この図よりＰＥ台数１～３の範囲では台数による稼働時間の割合はほぼ一定であり，この傾向はほぼ一般的なものと考えても良いと思われる。ＩＣＵは実稼働率がそれほど高くなく，次段のＡＰＵにおける入力待ち時間が多い割にはＡＰＵへの命令出力待ち率が高いという結果が出ている。これはＰＱＵ，ＩＣＵ，ＡＰＵで構成されるサーキュラ・パイプライン上ＡＰＵの負荷が大きく偏るためであると思われる。この対策としてＡＰＵ台数の増加による負荷分散とＡＰＵ自身の性能向上が考えられるが，ＡＰＵの実行時間とアクセス待ち時間から考えるとＡＰＵ台数増加による負荷分散での対策よりＡＰＵ自身の性能向上が必要である。現在の構成からＰＥ内のT-BUS のアクセスネックが予想されるが（現在上位ネットワークと同じものを使用している），実際に命令実行頻度の高い命令（プロセス制御命令が高く，特にこの命令はＬＭにプロセス制御用のデータを置いているためT-BUS アクセスが多い）についてそのダイナミック・ステップを考察して見るとバス権獲得のために費やしているステップは全体のステップのほぼ半分を占めている。つまりこのネックを解消することでＡＰＵの性能を向上できる。

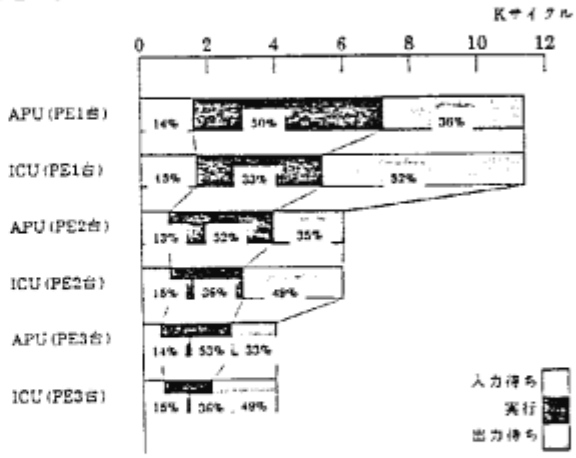


図６ ICUとAPUの稼動時間

５．２．２　発火制御

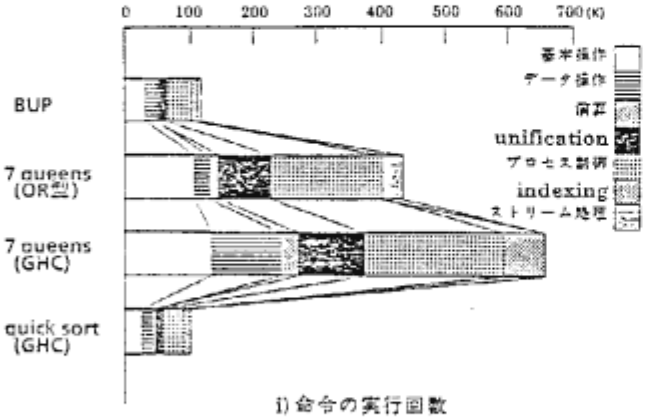
発火制御ではオペランドメモリの検索／書き込みにハッシングを用い，ハッシュ衝突時は再ハッシュを行ない前のエントリに連鎖する方式を取っている。OR並列型Prolog版７queensプログラムの場合，オペランドメモリは最大使用時１２％である。オペランド検索のためのハッシュの衝突は１８％であり，チェインの長さは０～１である。またトークンキューの使用率は最大使用時１０％である。さらに大規模なプログラムについての評価が必要であるが，これらの結論として現在のところ発火制御についてはハッシュ関数に関して改良は続けるが，ほぼその機能を満足していると考えられる。

５．２．３　実行制御

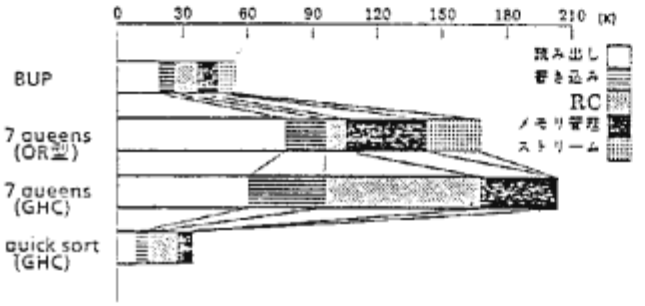
実行制御に関して命令の実行回数とＳＭコマンドの実行回数の特性を考察する。ここではOR並列型PrologにおけるＢＵＰと７queens，ＧＨＣにおける７queensとquick-sortの各命令実行回数とＳＭコマンド実行回数を測定しその結果を図７に示す。この図より命令に関してはプロセス制御命令と基本操作命令（コピー命令）が実行頻度が高く，これらの最適化が速度向上には有効であることが言える。言語の特徴としてはＧＨＣは解の操作をユーザ定義述語として明示的に実現しなければならずその操作がプロセス制御命令とその他の命令の増加に現れている。データ操作命令の増加はＧＨＣの特徴である同期命令と共有変数の生成のための命令をここに含めたためである。

ＳＭコマンドに関してはOR並列型PrologとＧＨＣでは傾向が異なる。OR並列型 Prologにおいて読み出しコマンドが全コマンドの半数近くを占めているのに対し，ＧＨＣにおいては読み出しコマンドは多少減少しリファレンスカウント制御用コマンドが大幅に増加している。これは変数に対して以下のような扱い方の違いがあるため

である。つまりOR並列型 Prologでは変数は単に識別子であり，トークンとして扱っており，解についてはストリームの操作によって実現している。一方ＧＨＣでは変数はＳＭ上のメモリセルへのポインタであり変数に対する操作は全てＳＭへの操作として実現され，解はこの変数により受け渡ししている。これよりＳＭコマンドの実行回数についても解の操作におけるストリームの実現の違いの影響を確認できた。



i)命令の実行回数



ii)SMコマンドの実行回数

図７ 命令/コマンドの実行回数

５．３　決定的宣言の効果

OR並列型Prologは決定的宣言により受け渡される解に従ったストリームの最適化ができる。最適化の概要は以下のとおりである。まずinvisible streamの実現方法はＡＮＤ関係にあるゴール間でnon-strictなデータ構造を共有させ，このデータ構造により解を受け渡す方法である。このとき生産者側ゴールが解を高々一つしか生成しない（決定的である）ならばnon-strictなデータ構造を使っての受け渡し（解のマージのための仕掛け）は必要なく，このストリームは直接的に値を受け渡すことによって代用できる。またこの代用により無駄なnon-strictなデータ構造に対する操作を無くすことができる。OR並列型Prologでこのような最適化を実現するためにはゴールが決定的であるか否か区別する必要があり，これをコンパイラに対して宣言させることで実現している。７queensプログラムのPE台数１台の場合のデータについてこの最適化による効果を表１に示す。またこの決定的宣言を行なっている述語は述語／４程度を占めている。この表より命令実行数は２５％減少し，ＳＭコマンド実行数は５０％減少し，性能として３０％向上している。

表１　決定的宣言の効果（OR並列型Prolog：７queens）

	命令数	コマンド数	実行サイクル
宣言無し	440050	190163	11409796
宣言有り	336829	96949	8125864

以上より評価の結果をまとめると以下のようになる。

(1)　１クラスタでの性能及び台数効果を確認した。
(2)　ＧＨＣはOR並列型Prologに比べてゴールの処理のレベルが低いため見掛け上の性能が高い。
(3)　ＧＨＣはストリームの効果的なシステムサポートにより性能向上が期待できる。
(4)　現在構成上ネックとなっているのはＡＰＵの性能であり特にT-BUS アクセスネックが問題である。
(5)　OR並列型Prologにおける決定的宣言の効果が確認できた。

６．改良と今後の課題

上記の評価より単体性能，負荷分配方式について改良と今後の課題を整理する。

６．１　単体性能の向上

演算部（ＡＰＵ）での処理ネックは前節で示したようにT-BUS アクセスの集中にある。このネックは構成上T-BUS をＬＭアクセス用とパケット転送用に共有しているためのバス権獲得待ちのためであり，これらの用途毎にバスを分離（ＬＭアクセス用バスを新たに設置）することで解消されよう。現在のＡＰＵの内部バス幅は主として実装上の問題から３２ビットに制限されており，T-BUS へのパケット転送に数ステップを要している。このバス幅を大きくすることによってマイクロプログラムステップの大幅な減少が期待でき，例えば出力データのレジスタへの格納は現在の４ステップを１ステップで実現できる。

このＡＰＵの処理性能の向上に伴い発火制御部（特にＩＣＵ）の処理性能の向上が必要である。現在のＩＣＵはＡＰＵがReady になるまで次のトークンの処理を開始できない。これはＩＣＵをトークン入力及びオペランドメモリ検索と命令パケットの送出の２段のパイプライン構成を実現することにより解決できる。

６．２　負荷分配方式

実験機ではマシンアーキテクチャの節で述べたようにプロセスの割り当てと構造データの割り当てを独立に行なうことが可能であり，現在それぞれの負荷分配はバランステーブル方式を採用して行なっている。バランステーブルとは分散環境にある資源（ここでは空プロセス識別子または空構造データセル）の管理テーブルであり，各ＰＥに資源単位に１つ割り当てられる。各ＰＥは全ＰＥの空プロセス識別子や全ＳＭの空構造データセル（以下，利用可能な資源と呼ぶ）をこのテーブルで管理し，プロセス割り当てや構造メモリの割り当て要求が発生したとき（実行時）にこのテーブルを用いて資源割り当てを行

なう．このテーブルへの新たな利用可能な資源の補給は割り当てられた資源で指定されるＰＥやＳＭにおいてその資源が利用された後に行なわれる．ここで他ＰＥ／ＳＭの負荷が重い場合はこの補給のための処理は遅れ，実行元のテーブルに含まれる負荷の重いＰＥ／ＳＭの利用可能な資源の割合は減少する．この結果資源の割り当ては負荷に従ってバランスする．またこのバランステーブルに含まれる利用可能な資源の割合をダイナミックに変えることによって問題解決時の実行特性に合った負荷分配が可能である．例えば問題解決の初期はクラスタ間に渡るようなゴール割り当てを中心に行ない，クラスタ内の負荷の増加に従いクラスタ内の局所的なゴール割り当ての割合を増加するという戦略が考えられる（これによりゴール間の通信をクラスタ内に局所化できる）．このためには最適な負荷の定義が必要であり，この負荷に従い利用可能な資源の補給の割合を変えることで実現できる．

現在，実験機ではバランステーブル方式の有効性を評価するために割り当て比をスタティクに与えそのバランスの様子を評価している．その結果，ゴール割り当てについては問題解決後の各ＰＥでの命令実行数及びゴール実行数によりほぼバランスしていることが確認され，構造データについても各ＳＭでのコマンド実行数より負荷のバランスが確認されている．

今後は最適な負荷の定義の決定とダイナミックな負荷バランスの実現を行なっていく予定である．

7．おわりに

データフローモデルに基づく並列推論マシンPIM-D 上でのＧＨＣの実現方式について，これまでPIM-D がサポートしてきたOR並列型PrologとＣＰとの実現方式と比較して，整理を行なった．また実験機についてその構成の概要を述べ，１クラスタについてのデータを評価した．これにより実験機の処理方式及び並列処理の有効性の確認ができ，またこれらと同時にハードウェア及び言語処理系における今後の課題を抽出できた．今後はこれらについてさらに検討していく予定である．

最後に，日頃ご指導をいただく内田俊一第４研究室長はじめ並列推論マシングループ諸氏に感謝の意を表する．

〈参考文献〉

[Arvi 78] Arvind et al.,"An Asynchronous Programming Language and Computing Machine," TR-114a, Dept. of ICS, Univ. of California, Irvine, Dec., 1978.

[Dijk 76] Dijkstra,"A Discipline of Programming," Prentice-Hall, 1976.

[Goto 85] A.GOTO et al. "Current Research Status of PIM: Parallel Inference Machine", Third Japan-Sweden workshop on Logic Programming, Tokyo pp.14, (Also ICOT TM-140). (1985-11).

[Ito 84-1] 伊藤他，"データフロー方式並列推論マシンのアーキテクチャ"，Proceedings of THE LOGIC PROGRAMMING CONFERENCE '84, Tokyo, 7-1,(1984-3).

[Ito 84-2] 伊藤他，"データフローマシン上でのConcurrent Prolog機能の実現"，第28回情処全大，5F-4,(1984-3).

[Ito 85] N.Ito et al. "The Dataflow-Based Parallel Inference Machine to Support Two Basic Languages in KL1",IFIP TC-10 Working Conference on Fifth Generation Computer Architecture,(1985-7).

[Kuno 85] 久門他，"並列推論処理システム　－改良型節単位処理方式－"，第30回情処全大，7C-8,(1985-3).

[Kuno 85] 久野他，"データフロー方式並列推論マシンにおけるＯＲ／ＡＮＤ並列効果のシミュレーションによる測定"，第30回情処全大，6C-3,(1985-3).

[Kuno 86] 久野他，"データフロー方式並列推論マシン－実験機の性能評価－"，第32回情処全大，2R-3,(1986-3).

[Onai 85] 尾内他，"リダクション方式並列推論マシンＰＩＭ－Ｒのアーキテクチャ"，PROCEEDINGS OF THE LOGIC PROGRAMMING CONFERENCE '85, Tokyo, 2.1, p.1-14, (1985-7).

[Ooha 86] 大原他，"データフロー方式並列推論マシン－実験機のプロセッサ割付け方式－"，第32回情処全大，2R-4,(1986-3).

[Roku 86] 六沢他，"データフロー方式並列推論マシン－実験機の動特性－"，第32回情処全大，2R-2,(1986-3).

[Sato 86] 佐藤他，"データフロー方式並列推論マシン－ソフトウェアシミュレータによるＧＨＣプログラムの評価－"，第32回情処全大，2R-1,(1986-3).

[Shap 83] E.Y.Shapiro, "A subset of Concurrent Prolog and Its Interpreter", ICOT TR-003 (1983-1).

[Ueda 85-1] K.Ueda, "Concurrent Prolog Re-examined," ICOT TR-102, 1985.

[Ueda 85-2] K.Ueda, "Guarded Horn Clause," ICOT TR-103, 1985.

[Ueda 85-3] 上田，"全解探索論理プログラムの決定的論理プログラムへの変換"，日本ソフトウェア科学会第２回大会，5A-2,p145-148,1985

付録　ハードウェア諸元

・ＰＥ（１台あたり）
- トークンメモリ容量：　１６Ｋトークン　×　８６ビット
- 命令メモリ容量：　９６Ｋ語　×　５９ビット
- オペランドメモリ容量：３２Ｋトークン
- 演算ユニット：　２台
  - マイクロプログラム制御
  - ビットスライス・マイクロプロセッサ使用
  - タグ判定ハードウェア内蔵
  - マシンサイクル：　３３３ｎ秒
- 局所メモリ容量：　５１２Ｋ語　×　３２ビット

・ＳＭ（１台あたり）
- 演算ユニット：　１台
  - マイクロプログラム制御
  - ビットスライス・マイクロプロセッサ使用
  - タグ判定ハードウェア内蔵
  - マシンサイクル：　３３３ｎ秒
- 構造メモリ容量：　１Ｍ語　×　３２ビット

・ＮＮ（１台あたり）
- 内部バッファ容量：　１２８トークン　×　８６ビット
- 転送サイクル：　５００ｎ秒　／　トークン