ICOT Technical Memorandum: TM-0717

TM-0717

# 並列推論マシンの組み立て

後藤厚宏

May, 1989



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

# 並列推論マシンの組み立て

後藤 厚宏
新世代コンピュータ技術開発機構 • 第4研究室

## 1 はじめに

ICOTの第五世代コンピュータプロジェクトでは，数[10]台またはそれ以上の要素プロセッサ(PE)を結合した並列推論マシンの研究開発を進めている[2]。並列推論マシンを作る第一の目的は知識情報処理の応用問題に対して十分な処理能力を提供することである。この意味で並列処理の技術は高性能化のための基本的枠組となる。

並列処理では，解くべき問題が持つ並列性，言い換えれば問題を解くアルゴリズムの並列性を，プログラムにおいて制限されない形で表現することから始まる。並列推論マシンの役割は，このような並列プログラムをいかにして効率良く実行するかにある。このためには，プロセッサやネットワークのハードウェア技術は勿論のこと，KL1で記述されたプログラムを並列に実行するためのメカニズム，コンパイル技術，さらには，実用システムにおいて必須となるガーベジコレクション等のメモリ管理機構を作り上げなければならない。

本稿では，核言語KL1プログラムを実行する並列推論マシンの枠組と鍵となる技術について紹介する。まず，並列推論マシンを作る上での課題について述べ，それが，KL1のコンパイラとその抽象機械語の設計，およびKL1の並列処理系の設計においてどのように対処されているかを述べる。続いて，現在開発を進めている並列推論マシンのハードウェアの概略を紹介する。

## 2 並列推論マシンをいかにして組み立てるか

ICOTの並列推論マシン研究においては，プログラミングからアーキテクチャまでを並列論理型言語KL1によって一貫性を持って実現することを一つの目標としている。このような一貫性は言語とアーキテクチャの整合性を高め，システムとしての見通しを良くするための重要な条件である。例えば，論理に基づくメタ推論，学習，知識の獲得/管理といった高度な人工知能ソフトウェアまでもが性能の向上の手段としてシステムの中に取り込みやすくなる。この意味で，KL1は並列推論マシンPIMの機械語(ハードウェアとの接点)からユーザ言語までの各レベルをカバーする核言語として，並列性，記述力，効率という3つの要求を満たすことができる言語の一つであると考えている。

では，KL1のプログラムを並列に処理する並列推論マシンを作る上での特徴的な課題と並列推論マシンを作る上での役割分担について考えてみよう。

### 2.1 並列推論マシンの課題

並列推論マシン(PIM)を作り上げるためには，多くの課題を解決しなければならない。その中で特に重要な2つの課題を取り上げてみる。

#### (1) 並列推論マシンが利用できる並列性

並列処理の単位(粒度)を小さくすると並列性は高まる。しかし，実際のプログラムでは，細かい粒に問題を分割すると，並列処理の効果に対する並列処理のオーバヘッド，つまり並列に実行されているもの同士の同期や通信のコストが高くなりがちである。また，細かい粒に分割しなくても与えられたプログラムがハードウェアとして用意されたプロセッサ数を大きく越える並列性を持つ場合もある。つまり，PIMを作る上で考えるべき並列性は処理性能を得るためのものであり，PIMを使って解こうとしている問題やプログラムの性質としての並列性[1]とは必ずしも一致しないことに注意しなければならない。このような性能を得るための並列度は，"並列処理のオーバヘッドを小さく抑えられるような粒の数"であり，利用できるハードウェア技術と処理方式の両面から適切な処理単位を選ぶ必要がある。

#### (2) 局所性の高いメモリ管理

PIMが対象とする知識情報処理では，扱うデータ構造がリストのように動的にメモリに割り当てられるものがほとんどである。特に，KL1はデータの破壊的書き換えを許さないため，単純に実装すると急速にメモリを消費してしまうため，PIMのメモリ管理の役割が従来の計算機にも増して重要となる。

PIMのメモリ管理は幾つかの難しい問題を含んでいる。通常の一括型ガーベジコレクション(GC)は，メモリ参照の局所性が悪く，キャッシュメモリのように参照の局所性を利用するハードウェアが有効に働かないという欠点を持つ。PIMにおいては，一般に，局所性のないメモリ管理操作はコストの高い通信となるため，全体性能が大きく低下してしまう。さらに，プロセッサ

[1] コンカレンシ(Concurrency)と呼ぶほうがよいかもしれない。

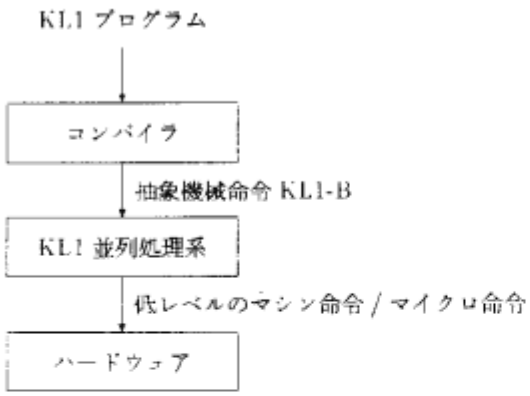


図 1: 並列推論マシンを作るための役割分担

をネットワークで結合した疎結合システムでは，一つのノードがGCを起こすと他のノードとの間の通信ができなくなるという問題も生じる。このため，PIMにおいては，メモリの割り当て / 回収を，実行時にできる限り局所的な操作によって行うことができるメモリ管理機構を設計する必要がある。

## 2.2 並列推論マシンを作る上での役割分担

並列推論マシン (PIM) が対象とするプログラムはKL1で記述され，最終的には，ハードウェアが提供する低レベルのマシン命令またはマイクロ命令によって実行される。PIMの設計においては前述のような課題を図1に示す3つの部分がうまく役割分担をしあって解決することが重要である。

KL1は抽象度が高い言語であるため，コンパイラによってKL1プログラムをKL1-Bと呼ぶ抽象的な機械語のプログラムに翻訳する。個々のKL1-B命令は，KL1のユニフィケーションを，KL1の並列実行の枠組の中で，クローズのガード部やボディ部，または引数の型に従って分類して操作を単純化したものである。コンパイラの役割は，KL1プログラムの静的な解析によってユニフィケーションだけでなく，メモリ管理に必要な情報[2]を抽出すると共に，重複した操作や判定を除くことによって，ハードウェアにとって効率良く実行できるオブジェクトプログラムに変換することである。

KL1-Bに変換されたプログラムは，ハードウェアが提供する低レベルのマシン命令またはマイクロ命令によるKL1の処理系によって並列に実行される。コンパイラによる解析を行っても，個々のKL1-B命令には実行時の動的な判定に基づく動作が多く残っている。また，KL1-B命令には，KL1の荘園機能，メモリ管理機能，および，ネットワークを介した通信プロトコルが含まれている。並列処理系の役割は，上記のように，KL1の処理系の基本操作であるにも関わらず，ハードウェアによって直接実装することが難しい複雑な操作を実現することにある。これらの機能の実現においては，大域的な操作を局所的な操作に置き換えることに主眼を置く。

ハードウェアの重要な役割の一つは，"処理全体において比較的単純であるにもかかわらずその頻度が多いため性能を抑えている要因" を解決することにある。PIMにおいては，KL1のデータ型判定のためのタグ操作，メモリ管理情報[2]の操作，共有メモリのデータ参照，および，ネットワークを介した通信機能を支援することがハードウェアの役割である。

## 2.3 マルチ PSI システム

並列アルゴリズムやPIMOSの開発には，従来型の計算機上でのクロスシステムだけでなく，実際の並列マシン上でプログラミングを繰返していくことが重要である。ICOTでは，そのような並列ソフトウェアの開発環境として，マルチPSIシステムを開発した。マルチPSIの要素プロセッサは，ICOTで開発した逐次型推論マシンPSIのCPUおよびメモリ部と同じものであり，それらがメッシュ型のネットワークによって最大64台まで接続される。これまでに，以下で述べるようなKL1の並列処理系とPIMOSが実装し，その上でのプログラムを積み重ねを進めている。

# 3 KL1 の抽象機械語とコンパイラ

KL1-B[4]は，コンパイラによってリダクションの制御や単一化を最適化することを目指したKL1の抽象機械語であり，PrologにおけるWAM (Warren Abstract Machine)[5]に相当するものである。KL1-Bでは，KL1の並列処理の単位をゴールリダクションとしている。コンパイラは，ゴールリダクション中のユニフィケーション操作をその意味毎に専用化したKL1-B命令に変換すると共に，メモリ管理に必要な情報を抽出する。

## 3.1 ゴール単位の並列実行

KL1のプログラムは，別稿で述べられているように，並列プロセスを表現するゴールを候補クローズで示されたユニフィケーションによって書き換えていくゴールリダクションによって実行される。ここで，全てのユニフィケーションは同期の規則を守っている限り並列に実行可能である。しかし，2.1節で述べたように並列処理の単位を小さくすることが性能の点では必ずしも得策ではない。そこで，PIMのKL1ゴール実行における同期や負荷分散の単位をゴールリダクションとし，ゴールリダクション中のユニフィケーション間の並列実行を行わず，ユニフィケーション間の関係をコンパイラで解析してレジスタを利用した最適化を行うことにする。

[2]3.3節のMRB操作の情報

### 3.2 インデキシングによる重複処理の除去

ゴールの実行では，与えられたプログラムの候補クローズ群のそれぞれについて，ヘッドとガード部のユニフィケーションを行い，一つのクローズを選択する。KL1では，ゴールの実行のために試みる候補クローズの選択の順番や一つのクローズ内でのユニフィケーションの順番が言語として規定されていないため，処理系が自由に決めて良い。このため，コンパイル時に複数の候補クローズのユニフィケーションの関係を解析し，それらを適切に並び換えることによって，個々のクローズが選択されるための決定木を求めることができる。これをクローズインデキシングと呼ぶ。

例えば，図2のような候補クローズがあった場合，(1)と(2)は第1引数がリストの時にのみ，また(3)はnilの時にのみ，選択される可能性がある。さらに，(1)と(2)のどちらを選択するかは，ガード部の $X \bmod P$ の結果によって一意に決めることができる。

PIMでは，クローズインデキシングのためのKL1-B命令を用意し，コンパイラがクローズ間の共通した操作を重複して実行しないようなKL1-B命令列に変換して実行する。

### 3.3 MRBによるガーベジコレクション

KL1プログラムの並列実行において，使用済みのメモリ領域を実行時に再利用するためには，メモリ中に表現されたデータ構造または未定義変数セルへの参照が，いつ増えたか，またはいつ不要になったか，の情報が必要である。KL1では，コンパイラの解析によって，このようなデータへの参照数の増減をクローズ毎に容易に知ることができる。例えば，

$$p(X) :- true \mid true.$$

では変数$X$を通して参照しているデータへの参照数が一つ減少し，

$$p(X) :- true \mid q(X),\ r(X).$$

では，参照数が一つ増える。

PIMでは，コンパイル時に取り出した参照数の増減情報をKL1-Bの命令として表現する。一方，KL1プログラムの実行で操作するデータ構造，または，未定義変数セルのポインタに，そのポインタが指すものへの参照数を示す1ビットのフラグ(MRB)を付加する。MRBは，図3に示すように，そのON●/OFF○によって，そのデータが複数のものから参照されている(多重参照)か，またはただ一つから参照されている(単一参照)かを示す[3]。これにより，KL1-Bのユニフィケーションにおいて MRB が OFF(単一参照) であるデータへの参照数が減る時は，そのデータのメモリ領域をその場で回収することができる。

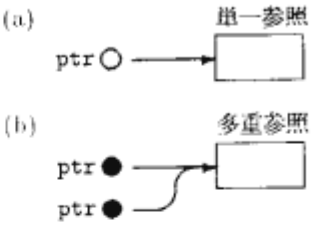


図3. MRB方式による参照数の表現

この方式では，一旦参照数が2以上になったデータのメモリ領域の回収することができないが，大部分のデータ構造への参照数がひとつであることが分かっており，KL1プログラムのメモリ消費スピードを抑えることができる。

### 3.4 KL1-Bのコード例

コンパイラは，前述(1)～(3)のようなKL1の候補クローズ群を図4のようなKL1-B命令列に変換する。ユニフィケーションに対応した命令(図4の read_car, get_list_value 等)のオペランドはユニフィケーションの対象となるデータやそれを持つレジスタ番号である。 jump_on_non_listはクローズのヘッドやガード部のインデキシング命令であり，オペランドとして決定木の枝に相当する飛び先ラベルを持つ。 execute, suspend, proceed等は4.1節で述べるゴール単位の並列実行を制御する命令であり，collect_list, collect_valueはMRBのGC用命令である。

## 4 KL1の並列処理

KL1の並列処理では，ユニフィケーションのためのデータ参照やゴールの実行制御を分散管理し，大域的な通信を節約することが重要なポイントの一つである。

### 4.1 ゴールの実行制御

システム内に存在する多数のゴールは，概念上並列に処理可能である。ただし，各ゴールをいつでも実行できるわけではない。それをリダクションするために必要な入力引数がプログラムの候補クローズのヘッドやガード部に指定されているためである。

PIMでは，図5に示す構造でゴールの実行制御を行う。まず，システム内のゴールはレディ状態かサスペンド状態にある。レディゴール(RG)はそのゴールに指定された優先度[4]のスタックに繋がれる。プロセッサはレディゴールをスタックから取り出し(CG)，ゴールの述語に対応する図4に示したようなKL1-Bコードに

[3] ただし，ポインタの先が未定義変数の時は，最大2までの参照をMRB-OFF とする．詳細は文献[1]参照．

[4] 別稿，" 並列論理型言語 KL1" 参照．

(1) $filter([X|X_{s1}], P, Y_{s0}) :- X \bmod P = \backslash = 0 \mid Y_{s0} = [X|Y_{s1}], filter(X_{s1}, P, Y_{s1})$.
(2) $filter([X|X_{s1}], P, Y_{s0}) :- X \bmod P =:= 0 \mid filter(X_{s1}, P, Y_{s0})$.
(3) $filter([], P, Ys) \quad :- true \mid Ys = []$.

図 2: KL1 プログラム例

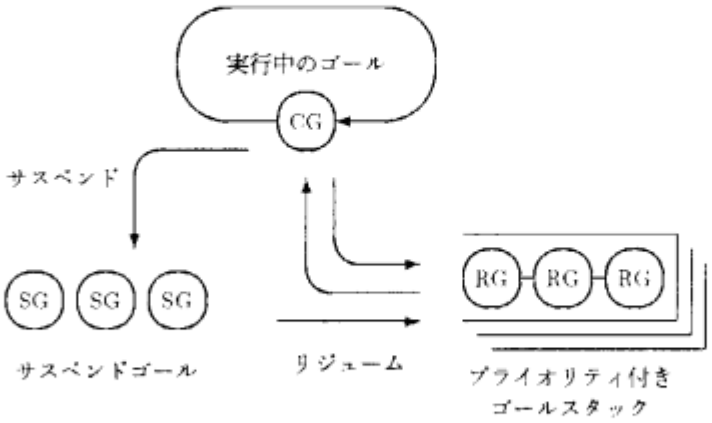


図 5: ゴールの実行制御

```
filter/3:    try_me_else  filter/3/1
             jump_on_non_list  A1, filter/3/3
             read_car  A1,  X4
             integer  X4
             read_cdr  A1,  X5
             integer  A2
             modulo  X4,  A2,  X6
             try_me_else  filter/3/6
             put_constant  0,  X7
             not_equal  X6,  X7
             collect_list  A1
             put_list  X1
             write_car_value  X1,  X4
             write_cdr_variable  X1,  X4
             get_list_value  X1,  A3
             put_value  X5,  A1
             put_value  X4,  A3
             execute  filter/3
filter/3/6:  collect_list A1
             put_value X5,  A1
             execute  filter/3
filter/3/3:  check_constant  [], A1, filter/3/1
             collect_value  A2
             get_constant  [],  A3
             proceed
filter/3/1:  suspend  filter/3
```

図 4: KL1-B コードの例

従ってリダクションを試みる。まず、ヘッドとガード部のユニフィケーションにおいて、KL1-B コードが指定したゴール引数が具体化していなかった場合は、サスペンドゴール (SG) としてそのゴールが待つ変数セルにリンクされる。一方、ゴールがリダクションされた場合は、選択されたクローズのボディ部に従って、未定義変数を具体化したり新たなゴールがフォークする。具体化した未定義変数にサスペンドゴールがリンクされていた場合は、そのサスペンドゴールをレディゴールとしてスタックに入れる (リジューム)。またフォークしたゴールの一つはレジスタ上のゴールのコンテキストを用いて次のリダクションを試みることにより、メモリからゴールのコンテキストをロードする手間を省く。

PIM では、ゴールの実行制御を分散化するために、各要素プロセッサがゴールスタックを持つ。この場合、プロセッサへの負荷の分散が重要な研究課題となる。そこで、各種の問題において適切な負荷分散方式を試みることができるように、KL1 プログラミングにおける負荷分散指定と、並列処理系による負荷分散の両者を用意している。5節で述べる PIM/p においては、前者をネットワークで結合されているクラスタ間に、後者を共有メモリで密に結合されているクラスタ内のプロセッサ間に適用する予定である。

## 4.2 ゴールの分散管理

KL1 の荘園機能は、メタレベルからオブジェクトレベルのプログラムの実行を監視・制御するための機能である。荘園機能は、それに属するゴール群の論理的なグループであるため、荘園とそれに属するゴールを結

ぶ木構造を設けて管理することができる。一方，個々のゴールは負荷分散によって別々のプロセッサに分散して実行される。荘園機能のための木構造が，ネットワーク結合されたプロセッサのノード[5]間にまたがると，荘園情報を管理するプロセッサに制御が集中したり，プロセッサ間通信が増えてしまう。そこで，PIMでは，プロセッサ，または，メモリを共有するプロセッサ群（クラスタ）毎に荘園の支店に相当するもの（単親と呼ぶ）を設け，荘園に属するゴールを分散管理する。これにより，ゴールリダクション毎に荘園の状態を問い合わせるような操作が局所化できる。

### 4.3 分散データ管理

ネットワークで結合されたPIMにおいて全体を一つのアドレス空間とすると，ノード毎に独立したガーベジコレクション（GC）が難しくなってしまう。なぜなら，あるノードのGCによってデータのアドレスが変わった場合，そのデータを参照している他のノードに新しいアドレスを教えて回らなければならないため，結局のところノード毎の局所的なGCが大域的な操作になってしまうからである。PIMではこれを避けるために，各ノードに輸出表と呼ぶ間接アドレス表を設け，ノードの外から内部のデータを参照しているパスを分離する。さらに，ノードから他のノードのデータを参照するパスのために輸入表と呼ぶ間接アドレス表を設けている。このようにノードの中と外のアドレス系を分離することにより，ノードの外部のデータへの参照の一つ一つの手間は増えるが，ノード毎の局所的なGCが可能になる。さらに，ノード間のデータ転送において，データを読み込んだことや，データを問い合わせにいったことを輸出表／輸入表に覚えておくことにより，重複したデータ転送を避けることができる。

## 5 並列推論マシンハードウェア

ICOTでは，現在，抽象機械語KL1 Bを共通の仕様とする幾つかのPIMハードウェアを開発中である。以下では，その内の一つであるPIM/pのアーキテクチャについて紹介する。

### 5.1 クラスタによるプロセッサの階層的接続

PIMのハードウェアでは，並列処理の効果を引き出すために，通信コストに局所性のある接続方式を用いて要素プロセッサ（PE）を接続し，PE内での局所的な処理，および比較的通信コストの小さい近傍のPE群による局所的処理を利用することが重要である。

PIM/pにおいては，図6に示すように，PEをクラスタによって階層的に接続し，通信コストに局所性のある接続方式を実現する。各クラスタは，共有バス／共有メモリによって密に結合された8台のPEからなる。PE数を128とした場合，PIM/pの本体は，ネットワークによって結合された16台のクラスタで構成されることになる。クラスタ内では一つのアドレス空間を共有し，レスポンスが高速で通信コストが小さいプロセッサ間の並列処理が可能である。一方，クラスタ間に渡る分散ユニフィケーションはネットワークを介した非同期メッセージ通信によって行う。

各PEはフロントエンドプロセッサ（FEP）または入出力機器を接続するポートを持っている。これにより，PIM/pの応用がデータベースのように大きい入出力バンド幅を必要とする時は，そのバンド幅にみあうだけの入出力機器を各PEに分散して接続することができる。

### 5.2 RISCと高機能命令を融合した命令セット

PIM/pのPEは，RISCの利点とマイクロプログラムの利点を融合した命令セットを持つ[2]。RISC指向の命令によりマシンサイクルの短縮とハードウェアコストの節約が可能となる。そこで，PIM/pは，MRBによる実行時のメモリ再利用やデレファレンスを支援するRISC指向の命令（単命令と呼ぶ）を持ち，4段のパイプラインを用いて，最高1マシンサイクル[6]に1命令の割合で実行できる。

KL1-Bの高機能命令はPIM/pのマクロ命令を利用して実装する。マクロ命令は引数として指定されたレジスタのデータ型判定に基づき，小さいコストでマクロの本体の条件呼び出しができる。マクロの本体は通常の単命令とほぼ同様の内部命令によって記述し，各PEの内部命令メモリから供給されるため，マクロ本体の実行中は共有メモリからの命令フェッチが節約できる。

このようなKL1向き命令セットと短いマシンサイクルにより，MRBによる実行時のGCコストを含めた要素プロセッサの単体性能として，appendにおいて600K rps程度，通常のプログラムにおいても200-500K rpsの性能を見込んでいる。

### 5.3 KL1向き一貫性キャッシュと排他制御機構

キャッシュメモリはCPUのメモリアクセスを高速化する基本技術である。PIMでは，クラスタ内での共有メモリアクセスにおいて一貫性が保持できるライトバック型のキャッシュを各PEに設けている。ここで，各クラスタにおいてPE台数に比例した並列処理性能を得るためには，共有バスネックの回避が必要である。そこで，バスの転送能力を高くするとともに，KL1の

[5] 5章で述べるPIMではクラスタがノードに相当する。

[6] マシンサイクルとしては50ナノ秒を目標としている。

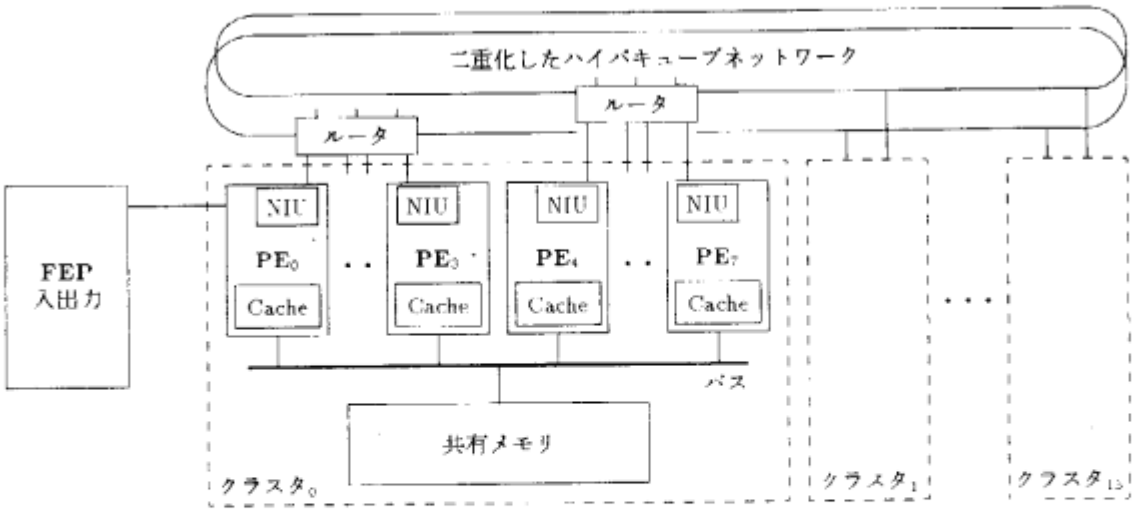


図 6: PIM/p の全体構造

並列処理の特性を活かしてバストラヒックを節約するキャッシュコマンドを用意している[3]。

また，共有メモリ上の変数を介した通信ではメモリアクセスの排他制御機構が重要である。そこで，各 PE のキャッシュのブロック状態を利用したロック機構を設け，低コストの排他制御を可能としている。

### 5.4 要素プロセッサ毎のクラスタ間メッセージ処理

PIM/pのクラスタ間に渡るメッセージ通信においては，パケットの組み立て / 読み取りをハードウェアで支援することが重要である。そこで，ネットワークへの接続口に NIU (Network Interface Unit) を用意し，CPU のコプロセッサとしてバイト / ワード変換機能とルータとのパケット送受を代行させる。さらに，クラスタ間に渡る並列処理性能をクラスタ内の高い並列処理性能に見合うだけ向上させるために，NIU をクラスタ内の各 PE に設け，各 PE がクラスタ間通信を必要とした時点で自らクラスタ間通信を行うことができるようにしている。

### 5.5 ハイパキューブ網を用いたクラスタ間の接続

クラスタ間ネットワークにおいては，実効的なスループットを高めるとともに，ハードウェア実装上の問題を解決する必要がある。そこで，ネットワークのトポロジとしてノード間距離が小さいハイパキューブ網を採用し，ルーティングの自由度を高めている。

ハイパキューブの次元数は 6 次元であり，最大 64 クラスタ (512 PE) をハイパーキューブのトポロジで結合することができる。さらに，各クラスタ性能に対応するために，ネットワーク全体を 2 重化し，各クラスタでは 4 個の PE(NIU) 毎に一つのネットワークルータに接続する。ネットワークの通信路は 1 バイト幅であり，クラスタ当たりの通信容量は最大 40M バイト / 秒である。

## 6 おわりに

並列知識情報処理を考えると，処理の汎用性 / 柔軟性をより強く認識する必要がある。例えば，扱うデータ構造はリストのように動的にメモリを使うものがほとんどであり，より強力なメモリ管理やガーベジコレクションの機能が要求される。これまで述べてきたような並列推論マシンを作る上での鍵となる技術について振り返ると，知識情報処理向きマシンを作るということは特殊な計算機を作ることではない，と言えるだろう。逆に，これまで“汎用”と呼ばれていた計算機は真に汎用なものではなく，実際にはその応用範囲が限られてきたと考えるべきであろう。つまり，計算機の適用範囲を広げるという意味において，知識情報処理向き計算機の開発は，より汎用な，または，より強力な計算機を作ることと考えてよい。知識情報処理向き計算機とは，単にゲーム木の探索を行うだけの計算機ではなく，これまでの計算機が持っている多くの機能も兼ね備えていなければならないからである。このように，知識情報処理向き並列計算機の研究開発は“計算機システム全体の機能と性能を強化する努力”と考えるべきであろう。

## 参考文献

[1] T. Chikayama and Y. Kimura. Multiple Reference Management in Flat GHC. In *Proceedings of the Fourth International Conference on Logic Programming*, pages 276-293, 1987.

[2] A. Goto et al. Overview of the Parallel Inference Machine Architecture (PIM). In *Proc. of*

*the International Conference On Fifth Generation Computing Systems 1988*. Tokyo, Japan, November 1988.

[3] A. Goto, A. Matsumoto, and E. Tick. Design and Performance of a Coherent Cache for Parallel Logic Programming Architectures. In *16th Annual International Symposium on Computer Architecture*, Jerusalem, May 1989.

[4] Y. Kimura and T. Chikayama. An Abstract KL1 Machine and its Instruction Set. In *Proceedings of the 1987 Symposium on Logic Programming*, pages 468–477, 1987.

[5] D.H.D. Warren. An Abstract Prolog Instruction Set. Technical Note 309, Artificial Intelligence Center, SRI, 1983.