TR-0774

並列推論マシンPIM

瀧　和男

May, 1992

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

並列推論マシン PIM

瀧 和男

第一研究室
(財) 新世代コンピュータ技術開発機構
〒108 東京都港区三田 1 丁目 4-28
taki@icot.or.jp

概要

並列推論マシン PIM は、第五世代コンピュータ (FGCS) プロジェクトの最終成果の一つであるプロトタイプ・ハードウェア・システムである。FGCS プロジェクトは、21 世紀の高度知識情報処理に必要とされる基礎技術の確立を目指しているが、PIM システムはそのような技術のうち、知識処理向きの大規模並列計算機アーキテクチャと、効率の良い核言語処理系の実現を目的としている。同時に PIM システムは、並列処理のための基本ソフトウェアおよび知識処理を始めとする応用ソフトウェアの研究開発環境の提供を目指している。

並列推論マシン PIM は、分散メモリ構造を持つ大規模 MIMD 型並列計算機であり、並列論理型言語 KL1 (核言語) を効率良く実行する機構を持つ。KL1 の言語仕様、その実現方式、マシンアーキテクチャはいずれも、知識処理に限らず、より広い問題領域として、動的で均質さの低い大量計算問題を効率良く実行するのに極めて優れた性質を持つ。このような問題領域は、ベクタープロセッサや SIMD 型並列計算機が得意としない分野であるとともに、市販されはじめた数値計算用の大規模 MIMD 型計算機システムも、満足にカバーできていない分野である。すなわち PIM システムは、これまでにほとんど開拓されていない新しい領域の大量計算問題を効率良く実行するための、ソフトウェアとハードウェアの技術を指向しているのである。

本論文では、2 つの重要な事項を述べる。第一は、並列推論マシン PIM と、その上の KL1 言語処理系について、研究開発の概要を報告し、技術内容を要約することである。第二は、開発したシステムが、特に動的で均質さの低い大量計算問題の高効率実行に優れているという観点から、言語、その実現方式、マシンアーキテクチャのそれぞれについて、特徴を列挙し他の並列計算機システムにない優れた点を明らかにすることである。これらにより、PIM システムの全体イメージを鮮明に描写することを試みる。

1 はじめに

第五世代コンピュータ (FGCS) プロジェクトは、21 世紀の高度知識情報処理に必要とされる、ソフトウェアとハードウェアの基礎技術を確立するために進められてきた。並列推論マシン PIM は、そのなかのプロトタイプ・ハードウェア・システムであり、知識情報処理を支える巨大な計算能力提供のための技術を目指している。本論文で扱う PIM システムは、ハードウェアとその上での核言語 KL1 の言語実装までを含める。

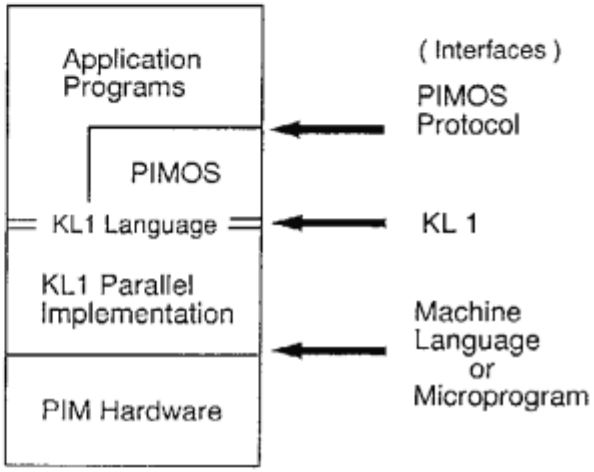

図 1: PIM のシステム構成の概要

プロジェクトの開始に当たり、多様な研究開発を並行して進めるための共通の技術的基礎として、論理型言語が選ばれた。この選択は、プロジェクトの創始者らの洞察に基づく以下のような作業仮説によっている。すなわち、将来の情報処理にとって問題となりそうな数々の事項、例えば並列処理の実行効率、言語の記述能力、ソフトウェアの生産性等々が、論理プログラムの枠組に基づいてシステム全体を再構築することにより、劇的に改善されるというものである。

この作業仮説に基づき、システムのすべて、すなわちハードウェア、システムソフトウェア、中核となる言語、応用ソフトウェア、プログラム方法論まで、すべてを論理プログラムの枠組の元に、ゼロから構築する努力を開始した。図 1 に、システム構成の概要を示す。

まず最初に、並列プログラムの記述と並列実行の双方の効率的実現を目指して、核言語 KL1 の言語設計を行った。続いて、PIM のアーキテクチャおよびハードウェア

設計と、KL1 言語の実装設計を並行して進めた。これを行うにあたっては、KL1 の効率の良い並列実行の実現を目指した。

PIM は分散メモリ構造を持つ大規模な MIMD 型並列計算機を指向し、基本的な目標としておおよそ 1000 台の要素プロセッサの接続を目指した。これらの要素プロセッサは高速のプロセッサで、記号処理の効率向上のためのハードウェアを備えている。PIM の開発は、知識処理を効率良く実行できるハードウェアアーキテクチャの実現とともに、並列ソフトウェアの研究開発環境の提供を目的としている。並列ソフトウェアの技術は、並列計算機の計算能力を最大限に引き出すために無くてはならないものであり、並列ソフトウェアの研究開発環境を整備することは、プロジェクトの運営理念の一つでもあった。

KL1 は並列論理型言語であり、まず最初は、知識処理を効率良く実現できる言語を目指した。けれども、多様な知識処理のための共通の基本言語である必要性から、特定の推論 (reasoning) メカニズムや知識表現機能を片寄ってサポートすることはせず、より汎用的な並列記号処理言語として仕様が固まった。

同時に KL1 の仕様は、知識処理のみならず、より広範囲の問題領域として、動的で均質さの低い大量計算問題の効率的処理に適するものとなった。このような問題領域は、市販され始めた数値計算用の大規模 MIMD 型計算機およびそのソフトウェアシステムが、カバーしきれていない問題領域であり、知識処理は、そのような問題の代表ともいえる。PIM のハードウェアと言語実装は、KL1 言語の持つ上記の問題領域向き機能をとりわけ効率良く実現するものである。

本論文は、次の 2 種類の事項に焦点を当てる。一つは、PIM ハードウェアと KL1 言語実装の研究開発状況と技術概要の報告である。もう一つは、KL1 言語、言語実装方式、およびハードウェアアーキテクチャの特徴と他の並列計算機システムに比べての優位性について、特に動的で均質さの低い大量計算問題に適するという観点から、整理し論じることであり、PIM システムの全体像の明確化を目的とする。

以下では、まず第 2 章で、FGCS プロジェクトにおける並列処理研究の歴史を概観するとともに、幾つかのキーワードの説明を行う。第 3 章は本論文の主題の一つである。KL1 言語、言語実装方式、ハードウェアアーキテクチャの特徴づけを行うとともに、他のシステムにない優れた点を列挙し論じる。続いて第 4 章では、PIM のマシンアーキテクチャとハードウェア仕様について技術の概要を報告する。アーキテクチャ研究用とソフトウェア開発支援用に PIM の 5 種類のモデルを開発しており、それらの各々について紹介する。第 5 章では、KL1 言語の実装方式、テクニックについて、基本的な重要事項を概観し、分散メモリ構造のハードウェアにおける、より具体的な言語実現イメージを与える。第 6 章では、幾つかの計測評価結果を紹介する。小粒度の並行プロセスの効率の良い実現と、遠隔プロセス間同期の効率の良い実現に焦点を当てる。また、ベンチマークプログラムを用いた性能評価と PIM 上の最新計測結果も紹介する。最後に、第 7 章では本論文の主張をまとめる。

すでに述べたように、ソフトウェアの技術は効率の良い並列処理を実現するためになくてはならない。並列ソフトウェアに関する幾つかの重要事項は、以下の関連論文に収録されている。並列オペレーティングシステムについては [Chikayama 92] に、またソフトウェアで制御する負荷バランス方式については [Nitta *et al.* 92] に報告されている。

2 研究開発の歴史

本章では、FGCS プロジェクトのなかでの並列処理に関わる研究開発の歴史を概観する。合わせて、重要なキーワードについて説明する。研究開発の経緯や、プロジェクト運営上の意思決定などについて、さらに多くの情報が以下の論文に記されている [Uchida 92] [Uchida *et al.* 88]。

2.1 並列処理研究の本格化

FGCS プロジェクトの中で、並列処理に関する研究開発が本格化したのは、プロジェクトの中期の始め、1985 年である。その少し前に、並列論理型言語 GHC [Ueda 86] が設計され、研究開発を進めるうえでの中核となる言語 (核言語) として選ばれた。言語機能については、3.4 節で述べる。

GHC (正確にはフラット GHC) をインターフェースとして、小規模のハードウェアおよびソフトウェア開発を開始した。試作ハードウェアは、並列ソフトウェアの研究道具として使用し、研究開発経験と評価結果を次の一回り規模の大きいシステムの開発に反映させる方針をとった。こうして、試作システムの規模と機能を次第に拡大してゆく一種のブートストラップ方式を取った。

マルチ PSI [Taki 88] が、このような研究開発の開始点となった。ハードウェア開発の目的は、マシンアーキテクチャの研究であるとともに、効率の良い言語実装方式の試験台を提供することでもあった。そしてさらに、並列ソフトウェアの研究開発環境として使えるものを実現すべく、ハードウェアの信頼性、短期開発、計測やデバッグ機能の充実などにも注意を払った。マルチ PSI/V1、マルチ PSI/V2 のハードウェア開発、言語実装、そしてそれらの上でのソフトウェア研究を経て、多くの並列ソフトウェアの技術やノウハウを蓄積するとともに、各種の並列応用プログラムを開発することができた [Nitta *et al.* 92] [Chikayama 92]。こうしてマルチ PSI は、最終的には PIM で使用するための並列ソフトウェアの開発環境として、十二分の働きをした。

2.2 Multi-PSI/V1

最初のハードウェアはマルチ PSI/V1 であり、1986 年春に稼働した [Taki 88] [Masuda *et al.* 88]。要素プロセッサとして、プロジェクト前期の成果であるパーソナル逐次型推論マシン PSI [Taki *et al.* 84] を使用した。6 台の PSI を 2 次元格子ネットワークで接続し、ネットワーク上は wormhole routing と呼ばれるメッ

セージ転送方式をとった。フラットGHC の始めての分散実装(分散メモリ計算機上の並列実装の意味)を試みた [Ichiyoshi *et al.* 87]。実行速度は、インタープリタシステムを ESP (PSI のシステム記述言語)で記述していたため、1 KLIPS (Kilo Logical Inferences Per Second) と遅かったが、フラットGHC を分散実装するときの基本方式やテクニックの研究開発にたいそう役立った。小規模の並列プログラムをその上で開発し、性能評価や負荷分散実験に使用した。

2.3 GHC から KL1 へ

GHC は、核言語の持つべき基本機能を規定したが、大規模なシステム上でのより本格的な利用を考えるとき、次に示すような言語の拡張が必要となった。すなわち、負荷割付とスケジューリングの制御機能、メタコントロールと資源管理等の OS 支援機能、実行効率を高めるための種々の組み込み機能などを拡張して、核言語 KL1 が誕生した [Ueda *et al.* 90] [Chikayama 92]。GHC が論理的な並行性 (concurrency) だけを取り扱うのに対し、KL1 ではさらに、それを物理的な並列実行 (parallelism) に結び付け(並列計算機への負荷割付を扱い)、実行効率の調整を可能とし、OS の実現を容易にするなどの点が強化され、ソフトウェアによる負荷分散の実験や本格的な応用プログラム開発に適用可能となった。

KL1 を効率良く実現するために、KL1 の実行メカニズムを分解して、仮想ハードウェア上のより低レベルの操作として規定した中間言語 KL1-B [Kimura *et al.* 87] が、並行して開発された。KL1 プログラムは、コンパイラにより KL1-B の命令列に翻訳される。システムにより、KL1-B の2進コードを直接マイクロプログラムで解釈実行するものと、KL1-B の命令列をさらに低レベルの機械語命令列に変換してから実行するものがある。

2.4 Multi-PSI/V2

二つ目の、より本格的なハードウェアとして開発したのがマルチ PSI/V2 であり、規模の面でも性能面でも、始めての並列推論マシン実験機と呼べるものになった [Takeda *et al.* 88] [Nakajima 92]。マルチ PSI/V2 は、1988 年から稼働を始め、FGCS'88 国際会議においても展示実演された。

マルチ PSI/V2 は 64 台の要素プロセッサを 2 次元格子ネットワークで接続したもので、各プロセッサは、改良版の逐次型推論マシン PSI-II [Nakashima *et al.* 87] の CPU と同一のハードウェアである。ネットワークは再設計し、転送性能が向上した(チャネルあたり 10MB/sec, full duplex)。KL1-B はマシン語そのものであり、それを解釈実行するマイクロプログラムを新規開発した。メモリ管理、通信、スケジューリング、例外処理等々の実行時機能も、ほとんどマイクロプログラムで実現している。処理性能、メッセージプロトコル、メモリ管理方式など、多くの点でマルチ PSI/V1 に比べて大きな改良を行い、単一プロセッサの性能は 130 KLIPS (KL1 の append プログラムにて) に達した。マルチ PSI/V2 の仕様は、表1〜4 に含まれている。1988 年以来、64 プロセッサの最大構成システムから 16 プロセッサの小構成まで、合計 15 システム以上が稼働し、ICOT を始めとして関連の研究機関で並列ソフトウェアの研究開発に使用されている。

マルチ PSI/V2 の強力なシミュレータを PSI-II 上に開発した。これをシュード・マルチ PSI と呼んでいる。大きな特徴は、CPU ハードウェアが同じであることを利用して、一旦シミュレーションが始まると、PSI-II 用のマイクロプログラムの代わりに、KL1 用のマイクロプログラムを動的に読み込んできて、マルチ PSI/V2 とまったく同じバイナリコードをマルチ PSI/V2 の単一プロセッサと同じ速度で実行することである。このシステムにより、PIMOS を含めた応用プログラムの並列実行のシミュレーションが効率良く行え、性能チューニングを除くプログラムのデバッグにたいそう貢献している。

2.5 Multi-PSI/V2 上のソフトウェア開発

並列オペレーティングシステム PIMOS の最初の版と、4 種の小規模並列プログラム (評価用プログラム) が、FGCS'88 国際会議に合わせて開発された [Ichiyoshi 89]。PIMOS の開発では、応用プログラムの並列性を阻害しないための徹底した分散資源管理と、デバッグや計測評価機能等の並列プログラム開発環境の充実に努力が払われた。小規模並列プログラムに関しては、並列ソフトウェアの重要な研究課題である以下のような項目の研究を兼ねて、開発を進めた。並列度が大きく計算の複雑さ (complexity) は増加しない効率の良い並列アルゴリズム、KL1 によるプログラミングに適した問題の設計技法とプログラムテクニック、動的で均質さの低い問題に適した負荷バランス方式、などである。

PIMOS は多くの改良を重ねながら、1992 年春までに PIM の各モデルに移植された。詰め込みパズル (pentomino) [Furuichi *et al.* 90] や最短経路問題 (bestpath) [Wada *et al.* 90] をはじめとする小規模並列プログラムについては、1989 年頃まで改良と計測評価を行い、その後も PIM のテストおよび評価用に継続使用している。

これらのソフトウェア開発を通して得た感触は、マルチ PSI/V2 上の KL1 言語処理系と PIMOS は、実際的な応用プログラムでの使用に耐えられる性能水準に達したとともに、複雑な並列プログラムの記述やソフトウェアによる負荷分散実験を行うのに十分な機能を備えるものとなったということである。

このことを踏まえて、大規模な並列応用プログラムの研究開発が、1989 年後半頃から始まり、現在も継続中である [Nitta *et al.* 92]。それらの幾つかはすでに PIM に移植され、高い処理性能を示している。

2.6 並列推論マシン PIM

2.6.1 PIM の 5 つのモデル

並列推論マシン PIM の設計は、マルチ PSI/V2 の製造と並行して始められた。マルチ PSI では、ソフトウェア開発に使用する関係上、短期に十分信頼性の高いシステムを開発するという必要性から、マシンアーキテクチャ

上の研究要素を十分に盛り込みにくかった面がある。そこで PIM では、アーキテクチャ上の研究要素も十分取り込みつつ、並列ソフトウェアの研究にも使用するという、欲張った開発計画を立てた。

アーキテクチャ上の最も顕著な違いは、マルチクラスタ構成である。10 台前後の少数のプロセッサが、共有バスと共有メモリを介して密に結合しているのがクラスタである。そして数多くのクラスタを高速のネットワークで結合して、プロセッサ台数数百以上の大規模システムを構成する。このようなアーキテクチャの利点については 3.7節で述べる。

新しいシステムを実現するにあたり、頻繁なプロセッサ間通信に適する並列キャッシュメモリ、より高性能の記号処理向き要素プロセッサ、ネットワークの改良など、多くの要素技術の研究開発が必要であった。そこで異なる要素技術または要素技術間の組み合わせを試すことができるよう、5 つの PIM モデルの開発を計画し、各々のモデルに対して異なる研究テーマを与えた。

2 つのモデル、PIM/p [Kumon *et al.* 92] と PIM/c [Nakagawa *et al.* 92] は、マルチクラスタ構造を持ち、それぞれ 512 プロセッサと 256 プロセッサの最大構成をとる。ともにアーキテクチャ研究用とソフトウェア開発用を兼ね、プロセッサやネットワーク構造が互いに異なる。

PIM/k [Sakai *et al.* 91] と PIM/i [Sato *et al.* 92] は、クラスタ内アーキテクチャの研究用マシンである。キャッシュメモリの方式やプロセッサ構造に特徴を持つ。

PIM/m [Nakashima *et al.* 92] は、マルチクラスタ構造を取らないかわり、マルチ PSI との互換性を最大限に重視したソフトウェア開発用のマシンである。マルチ PSI/V2 に比べて単一プロセッサの性能は、約 1.5 〜 2.5 倍高く、プロセッサ数は 4 倍の 256 台である。要素プロセッサには、最新の逐次型推論マシンである PSI-UX [Nakashima *et al.* 90] の CPU を使用しており、PSI-UX 上でシミュレータのシュード PIM/m が利用できる。

それぞれのマシンアーキテクチャとハードウェアの仕様については、4章にまとめる。

PIM の開発では、一部の LSI の試作を 1989 年頃から始め、最終設計を 1990 年にほぼ完了し、1991 年には製造を行い、その後 1992 年春にかけて、組み立て・試験とソフトウェアの実装を進めてきた。

2.6.2 ソフトウェア互換性について

KL1 言語はすべての PIM モデルに共通であり、実行効率を問題にしなければ、PIMOS を含めあらゆる KL1 プログラムがどのマシンでも実行可能である。

ハードウェアアーキテクチャは 2 つのグループに分かれる。マルチクラスタ構造を取らない PIM/m およびマルチ PSI のグループと、それ以外の PIM モデルのグループである。構造は 2 つのグループ間で異なるが、プログラマから見える抽象化されたアーキテクチャは、以下の点で共通である。

PIM/m およびマルチ PSI では、プロセッサへの仕事の割当はすべてプログラム制御で行う。これはプログラマが直接記述する場合もあれば、負荷割付のライブラリプログラムが行うこともある。一方クラスタ構成を持つ PIM では、クラスタ内の負荷バランスはプログラム制御で行うのではなく、KL1 処理系の中のスケジューラが行う。すなわち、プログラマはクラスタ内負荷バランスについては考える必要がなく、クラスタへの負荷割付だけを指定すればよい。つまりプログラマから見れば、クラスタ 1 個が PIM/m やマルチ PSI のプロセッサに対応しており、構成の違うモデル間でのプログラムの移植を容易にしている。

2.7 PIM のための KL1 実装

KL1 は、大規模な記号処理の用途向けに、効率の良い本格的な言語処理系を実現したおそらく世界で最初の例であろう。KL1 の実行メカニズムのうち、分散メモリ構造のハードウェアに対応する部分は、マルチ PSI/V2 上の KL1 実装の段階で、効率の良い方式の追及を十分に行うことができた。PIM 用の KL1 実装を行うにあたって、主にマルチクラスタ構成に対応するための拡張を行った。

拡張項目としては、共有メモリの更新を行う場合の排他制御、クラスタ内の自動負荷バランス、メッセージの追い越しに対処するためのメッセージプロトコルの改良、クラスタ内の並列塵集め、などである [Hirata *et al.* 92]。

KL1 処理系をクラスタ構造を持つ 4 種の PIM モデルに効率良く移植するための方法も工夫した。共通の処理系仕様として、仮想ハードウェア上の実行アルゴリズムを実行可能な仕様として記述した。これを VPIM (バーチャル PIM) と呼んでいる。VPIM を記述するために、C 言語に近い仕様記述言語 PSL を設計して用いた。VPIM は、C 言語に変換してシミュレータ上で実行することができ、これにより実行アルゴリズムを検証する。検証が終わると、それぞれの PIM モデル用に、VPIM を自動または手作業により機械語命令やマイクロプログラムに変換してそれぞれの PIM 用の KL1 実装とする。

PIM/m に関しては、マルチ PSI/V2 の KL1 実装を厳密に引き継ぎ、マルチ PSI のマイクロプログラムを手作業と自動変換を織り交ぜて PIM/m 用マイクロプログラムに変換した。PIM/m とマルチ PSI は、KL1 プログラムのオブジェクトコードに関して、2 進コードレベルで互換性を持つ。

VPIM の開発は、1988 年頃の仕様記述言語の準備から始まり、クラスタ構造に対応するための実行メカニズムの拡張設計が 1991 年まで続き、VPIM 仕様として完成した。各 PIM モデルへの移植は 1990 年秋頃から部分的に始まり、1992 年に完了した。現在、PIMOS と合わせて、各 PIM 上で KL1 が利用可能である。PIM/m に関しては、他の PIM より一歩早く、1991 年夏から PIMOS を含めて稼働している。

2.8 性能評価とシステム評価

計測、分析、評価は、以下に示すようなシステムの色々なレベルで実施しなければならない。

1. ハードウェアアーキテクチャとハードウェア実装

2. KL1 の実行メカニズムと実装テクニック

3. 応用問題の解法に関する並列アルゴリズムとその実装テクニック

4. 負荷割付方式

5. ある問題をあるアルゴリズムと実装手法と負荷割付方式を用いて あるマシン上に実現した時の総合性能

マルチ PSI では、上記に関する各種の研究が行われてきた。1～2 に関する報告 [Masuda *et al.* 88] [Nakajima 92] や 3～5 に関する報告 [Nitta *et al.* 92] [Furuichi *et al.* 90] [Ichiyoshi 89] [Wada *et al.* 90] がある。

PIM に関しても、初期の計測が始まっており、速報が [Nakashima *et al.* 92] [Kumon *et al.* 92] に収録されている。PIM の総合評価はまだこれからであるが、初期計測と簡単な考察について 6章で扱う。

3 PIM と KL1 システムの特徴付け

PIM と KL1 システムは、大規模な知識処理を効率良く実行するのに適した数々の優れた点を有する。このような処理は、動的な実行時特性や不均質な計算をしばしば発生するが、その性質は、ベクタープロセッサや SIMD 型計算機が得意とするような科学技術計算の性質とは著しく異なっている。

本章ではまず、はじめの2節でシステムの特徴の概略をまとめ、続く本論の理解を助ける。つぎに PIM と KL1 システムが処理対象としている問題領域の性質を明らかにする。そして、そのような問題領域における効率の良い並列プログラミングと並列実行を実現するために、KL1 言語、言語実装方式、マシンアーキテクチャが備えている数々の優れた点について論じる。これらにより、システムの全体像を明確に描き出すことを目指す。これらはまた、PIM と KL1 システムが、最近市販されはじめている数値計算用の大規模 MIMD 計算機と比べて、いかなる点で違って(優れて)おり、いかなる点が共通しているかを知るための、よい助けとなることであろう。

3.1 システムの特徴のまとめ

本節では、PIM と KL1 システムの全体イメージと主な特徴を簡単にまとめ、以下の節への導入部とするとともに、あとから読み返すときのためのコンパクトなサマリーとする。

分散メモリ型 MIMD マシン： PIM の構成を大まかにいうと、分散メモリ構造を持つ MIMD 型並列計算機であり、数百の計算ノードが高速のネットワークで接続されている。プロセッサ台数の拡張性と実現の容易さを重視している。計算ノードは、1台のプロセッサか密結合のクラスタであり、大容量のメモリを備える。プロセッサは記号処理に適した機能を持つ。

論理型言語： 核言語 KL1 は並列論理型言語であり、ソフトウェアとハードウェアの唯一のインターフェースである。KL1 は、並列プログラミングに適した汎用言語であり、特に並列の記号処理に適しているが、ある種の論理型言語が持つような特定の推論 (reasoning) メカニズムや知識表現機能を実現しようとするものではない。KL1 は、以下に示す問題領域のプログラムを効率良く実現するための数多くの優れた機能を有する。

対象問題領域： 大規模な知識処理および記号処理が、第一義的応用問題領域であるが、より広範囲の領域として、動的で均質さの小さい大量計算問題 (非データ並列の大量計算問題) も対象とする。

言語実装： 分散メモリ構造のハードウェア上に、一つの KL1 システムが実装されており、計算ノードごとに独立の KL1 システムが載っているのではない。論理変数やプログラムコードについて大域名前空間を実現している。プロセス間の通信や同期は、たとえ別々の計算ノード間であっても暗黙の内に行われる。小粒度の並行プロセスを効率良く実現している。

これらの実装方式は、KL1 言語の持つ上述の問題領域向き機能を効率良く実現することを目指している。

負荷バランスの基本方針： 計算ノード間の負荷割付および負荷バランス制御は、プログラム制御を基本とする。プログラムで制御できないような自動負荷バランス機構は設けない。言語システムは、ソフトウェアによる負荷分散実験に必要な、機能と処理効率を実現する。

3.2 基本方式の選択

(1) 論理型言語： 論理型言語を核言語に使用することが、最初の選択であった。これは1章に述べたとおり、プロジェクトの作業仮説に基づくものであるが、論理型言語が知識処理と並列処理の両方に適するという、プロジェクト創始者たちの洞察があった。具体的な KL1 言語仕様決定に至る過程は [Chikayama 92] に報告されている。

(2) 言語仕様先決めのアプローチ： システム中の幾つもの異なる階層について並行開発する必要性から、始めに中間層である核言語の仕様を定め、その仕様に基づいて、下層のハードウェアおよび言語実装方式と、上層の OS および並列プログラムの各々を並行して設計、試作していった。

(3) MIMD マシン： a. MIMD 型並列計算機は、標準的な構成要素を用いて最も実現が容易なこと、b. 1980 年代前半のデータフロー計算機は、投入するハードウェア量に対して得られる性能が必ずしも十分でないことが、プロジェクト前期の研究から判断されたこと、c. SIMD 型計算機は非データ並列の大量

計算問題に対して効率を発揮しにくいと考えられたこと、の3点により MIMD 型の選択となった。

(4) 分散メモリ構造： 選択の第一の理由は、実現の容易さとプロセッサ数に関するの拡張性の容易さである。

第二として、共有メモリ構造を基本としてプロセッサ台数の拡張性を持たせた以下のようなシステムは、KL1 プログラムのように小量の遠隔データ転送を頻繁に起こすようなプログラムに適さないのではないかと考えられたことである。例えば、階層的なキャッシュメモリを用意しディレクトリ管理等を用いて規模の拡張性を実現しているシステムがあるが、小量のデータ転送に適するようメモリの管理単位を小さくすると、ディレクトリメモリの容量が現実的でないほど増大する欠点がある。

第三として、共有メモリ構成のシステムを使ったとしても、単一代入言語を実装しようとすると塵集め等のメモリ管理をなるべく局所的に効率良く実現したくなり、これは共有メモリ構成で提供される単一アドレス空間の利点を使わない方向になるからである。つまり、計算ノード単位の塵集めを行うためには、遠隔参照とローカルな参照を分離する必要性が出てくるからである。

3.3 対象問題領域の特徴付け

(1) 対象問題領域と特徴： 大規模な知識処理や記号計算の性質は、ベクタープロセッサや SIMD 型計算機が得意とするような種類の数値計算問題の性質とは、大きく異なっている。これらの数値計算問題は、普通はデータ並列性を備える問題として定式化されており、均質なデータに対する同期的な計算が行われる。

それに対して、大規模な知識処理や記号計算を並列計算機で実行する場合、しばしば動的で均質さの低い計算を発生する。たとえば発見的知識を用いる探索問題を解く場合、各計算ノードにおける計算負荷は、探索木の拡大や枝刈りの発生により劇的に変動する。また知識処理システムが、多くの性質の異なるオブジェクトから出来ている場合、それぞれのオブジェクトはしばしば非同期的に振る舞い、均質でない計算負荷を発生する。これらの問題の計算負荷は、実行する前から正確に予測することは困難な場合が多い。

このような性質を持つ問題を本論文では、動的で均質さの小さい大量計算問題と呼ぶことにする。知識処理や記号計算の多くの問題がこの中に含まれるとともに、データ並列性の小さい大規模な数値計算問題なども、並列処理で高い効率を実現することの難しさにおいて、同類と考えられる。PIM と KL1 システムは、これらの問題を対象として、効率の良い並列処理の実現を目指している。

(2) システムへの要求事項： 動的で均質さの小さい大量計算問題は、データ並列性の大きい問題に比べ、プログラム構造や通信・同期の仕様が複雑であったり、実行時の負荷バランスが難しかったりしがちである。そのような問題をプログラム化し効率良く並列実行しようとする場合、以下に挙げる事項が、データ並列処理の場合に比べてより強くシステムに対して要求される。

1. 複雑な構造を持つ並列プログラムが記述し易いこと
2. バグが除去できること (特に通信と同期に関するバグ)
3. 動的負荷バランスが実現できること
4. 負荷の発生が予想と異なることも多いため、負荷割付けとスケジューリングの変更に対して柔軟性が大きいこと

3.4 KL1 言語の特徴付け

本節では、KL1 言語の持つ幾つもの優れた特徴を列挙し解説する。特にそれらの言語機能が、前節に上げた要求事項を満たし、動的で均質さの低い大量計算問題におけるプログラミングと効率の良い並列処理の実現に、なくてはならないものであることを述べる。KL1 の設計思想と特徴に関する詳しい記述は [Chikayama 92] に含まれる。

はじめの3項目は、論理的並列性 (concurrency) の記述能力に関するもので、KL1 の基本機能にあたる GHC の仕様に含まれるものである。これらは、複雑な構造を持つ並列プログラムの記述能力を高めている。

(1) データフロー同期： KL1 プロセス間の通信と同期は、データフローモデルに基づきまったく暗黙のうちに行われる (プログラム中では単なるユニフィケーションである)。暗黙の通信・同期機能は、遠隔プロセス間でも同様に利用できる。この機能は、専用のプリミティブを用いて通信や同期を陽に記述する場合に比べ、通信と同期に関するバグ混入の機会を減少させるのに絶大な効果を発揮している (PIMOS の開発等で実績が示されている)。暗黙の通信・同期の実現は、論理変数の持つ単一代入の性質を利用している。

(2) 小粒度並列プロセス： KL1 プログラムにおける並列実行の単位は、節定義の中のボディゴールの1個1個 (GHC では更に細かい) である。各ゴールは、小粒度のプロセスと見なすことができ、KL1 の実行は多数の小粒度プロセスの並列実行であるともいえる。KL1 プログラムは、ボディゴールの数だけの並列性を自然に内在することができる。

(3) 非決定性 (indeterminacy)： 論理変数に値が定まるまで自動的に待つのがデータフロー同期のメカニズムであるが、KL1 の1個のゴール (プロセス) は、複数の論理変数の何れかに値が定まるのを一時に待つことができる (データフロー同期と非決定性を組み合わせることによって実現している)。一つの変数に値が定まると実行が再開され、ガードの実行に成功した候

補節の一つ (通常はその変数を待っている節) が選択されて、ボディへ実行が進む。この場合の節選択に関する非決定性を indeterminacy と呼んでいる。

この機能は、非決定的な動きをさせたいプログラムの記述には無くてはならない。例えば、並列処理効率を高めるために見込み計算 (specurative computation) を許したプログラムの記述や、動的負荷分散プログラムの記述であり、これらは実用的な並列処理に必須のものである。この種のプログラムは、非決定性を持たない単一代入言語では記述が困難である。

次に挙げる 3 つの特徴は、GHC から KL1 へ言語拡張したときの拡張機能に関するものである。(4) はプロセッサへの仕事の割付けやスケジューリングの制御に関するものであり、論理的な並列性 (concurrency) を並列計算機上の実行時の並列性 (parallelism) に対応付けるための機能ともいえる。(5) は OS の実現を支援するための機能、(6) は実用プログラムの記述に必要な効率向上のための機能である。

(4) **プラグマ (pragma)**： プラグマは、計算ノードへのゴールの割付けとゴールの実行優先度を指定するための記法である。プラグマはプログラムの意味には影響を与えずに、実行時の並列性と実行効率を制御する。プラグマは通常、問題を解くためのアルゴリズムに係わるプログラム記述およびデバッグが完了した後に、ゴールに対して付加するものであり、シンタクスがプログラムの他の部分の記法と明らかに異なるため、付加および変更が容易である。

負荷分散プラグマ： ゴールのプロセッサへの割付けは、@node(X) というプラグマで指定する。X はプログラム中で計算可能である。(1) と (2) の機能とあいまって、負荷分散プラグマにより、計算ノードへの負荷割付けに関する大きな柔軟性を実現している。つまり、(1) があるために負荷割付けを変えても通信や同期の記述を変えずに済み、(2) のおかげで負荷割付け量を細かく制御することができる。また (3) との組み合わせにより、単一代入言語の良い性質を壊さないままで、動的負荷分散プログラムを記述することができる (単一代入の性質を壊すと (1) の実現が難しくなる)。

実行優先度制御プラグマ： ゴールの実行優先度は、@priority(Y) というプラグマで指定する。現状で 4096 の優先度レベルが実現されており、きめ細かい優先度指定ができる。このプラグマは、ゴール間の厳格ではない実行順を制御し、並列処理効率の調整に有効である。

(3) の機能と組み合わせることにより、見込み計算を効率良く制御することができる。例えば、発見的知識を用いた探索問題を並列に実行する場合、知識を適用して得られた評価値の高い枝ほど高い優先度で実行することにより、並列性を維持しながら無駄な見込み計算を抑えることができる。

(5) **荘園機能 (ゴールの集合を制御する機能)**： 荘園は、ゴールの集合をタスクとして取り扱うための機能である。タスクとはプログラム実行および資源管理の単位である。荘園機能は主に PIMOS の中でタスク管理に使われている。タスクの起動・停止や資源消費の上限を設定することができ、またタスク内でのエラーや例外は、凍結されて荘園の外に報告される。

(6) **効率向上のための組込み機能**： KL1 は、効率向上のための組込み述語やデータ型を用意している。これらは、単一代入規則を始めとする論理型言語の (GHC のレベルの) 性質から逸脱しない範囲で導入されている。これらの機能は、逐次プログラムに比べて同じ計算の手間 (complexity) でプログラムが実現できるためのものや、単一代入言語であるがための効率の悪さを救うためのもの等を含んでいる。

3.5 言語実装方式の特徴付け

前節で述べた KL1 言語の優れた特徴は、動的で均質さの低い問題の効率の良い並列処理に必要とされる 3.3 節に挙げた要求事項に合致するものであった。本節で述べる言語実装方式の特徴のほとんどは、上記の言語の特徴を最大限に生かしながら、現実に高い実行効率を実現することを目指したものである。以下では特徴を列挙することに重点をおき、実現方式の技術的な説明は 5 章に示す。

(1) **暗黙の通信** 並列プロセス間の通信と同期は、プロセス間で共有された論理変数に対するユニフィケーションとして暗黙のうちに実行される。これらは計算ノード内でも、プロセスが別々の計算ノードに置かれた場合でも、共通である。遠隔プロセス間での同期が暗黙のうちに行われることは、プログラマにとってとりわけ利益が大きい。

プロセス (ゴール) は、負荷分散プラグマ @node(X) を付加するだけで計算ノード間を移動する。このときプロセスが参照ポインタを持っていたならば、異なるノード間での遠隔参照が自動的に生成される。これらの遠隔参照を用いて、遠隔プロセス間での通信と同期が実現されている。

この方式により、ハードウェアの分散メモリ構造は、プログラマにとってほとんど見えないものとなっている。プログラマは、プロセス間の通信と同期を設計するのに、それらのプロセスがどこに置かれるか (同一計算ノード内か別ノードか) を意識する必要がなくなる。

すなわち、論理的並列性の記述の段階 (コンカレント・プログラミング) では、プログラマはハードウェア構造を意識しないでよく、プラグマを用いて負荷割付けやスケジューリングなどの実行時の並列性制御を記述する段階 (パラレル・プログラミング) でだけ、プロセッサの接続構造等のハードウェア構造を意識すれば良い。

これらの機能を実現するための実装方式の特徴を以下にまとめる。

- 分散メモリハードウェア上の大域名前空間 — 論理変数、構造体データ、プログラムコード等に対して、計算ノード間をまたがった遠隔参照の管理が暗黙のうちに行われる。
- (計算ノード間での) ユニフィケーションおよびゴール (プロセス) 移動により起こる暗黙のデータ転送
- データ転送やゴール移動時の暗黙のメッセージ送受、メッセージの組み立て分解
- メッセージ数の削減を目指したメッセージプロトコルと、ネットワーク上のメッセージ追越しに対処できるメッセージプロトコル

(2) 小粒度並列プロセス： 小粒度の並列プロセスを効率良く実現しており、それらの間での通信と同期も低いコストで実装した。実行優先度管理を取り入れたプロセススケジューリングなども効率良く実現している。効率の良い実現が出来たことで、多数の小粒度プロセスを活用した並列性の高いプログラミングを現実の応用プログラムで用いることが可能となった。プロセス個数が多いと、負荷バランスの制御も容易となる。

クラスタ内の自動負荷バランス機能も、上記スケジューリング機構として実現した。

(3) メモリ管理： 以下の塵集め機構を実装した。

- クラスタ内における、実行時塵集め (参照カウント法のサブセットを使用) と、停止回収型のコピー方式による塵集めの組み合わせ
- 遠隔参照ポインタの実行時におけるインクリメンタルな回収 (重み付け参照カウント法を使用)

塵集めを含む動的メモリ管理を効率良く実現することは、記号処理言語を実現する上で無くてはならないが、同時に実用的な単一代入言語を実現するという意味でも不可欠である。

単一代入の性質は、すでに述べたように暗黙の通信・同期などに大きく役立っており、動的で均質さの低い問題のプログラミングにとって重要である。こう考えると、塵集めの機能は、記号処理でなくても動的で均質さの低い大量計算問題一般の並列処理にとって、なくてはならないものと言えそうである。

(4) 荘園の実装： 荘園は、ゴールの集合を1個のタスクとして一括管理・制御するための機能である。複数の計算ノードに分散されたゴールを効率良く一括管理・制御するために、数々のメカニズムを実現している。

(5) OS 核を取り込んだ言語実装： 図2に、KL1 の実装と OS 機能との関係を示す。KL1 の言語実装は、いわゆる OS 核の機能を中に取り込んだものとなっている。メモリ管理、プロセス管理とスケジューリング、通信と同期、大域名前空間、通信メッセージの組み立て・送受などの機能を、言語実装の中で実現している。一方 PIMOS は、プログラミング環境やユーザーインターフェースを始めとする上層の OS 機能を実現している。

言語実装のなかに OS 核機能を取り込んだ理由は、それらの機能をなるべく低いコストで使いたいためである。それらは、動的で均質さの低い大量計算問題向けに KL1 言語が備える優れた特徴を、真に効率良く実現するのに不可欠だからである。例えば、小粒度の並列プロセス、それらの間の通信・同期、さらに暗黙の遠隔通信・同期、優先度付きスケジューリングなど、すべて OS 核機能に関係するものである。

OS 核機能を低いコストで利用できるようにすることは、われわれの対象とする動的で均質さの低い問題の並列処理にとって、きわめて重要なことと言える。たとえ数値計算であっても、データ並列性の低い問題の場合には同様に重要である。

市販され始めている大規模 MIMD 型計算機の上で、同様の問題領域の処理を効率良く行おうとすると、現在の標準的なオペレーティングシステムのインターフェースでは、処理オーバーヘッドが大きすぎる心配がある。インターフェースの切り直しや OS 核の再設計が必要になるかも知れない。

3.6 負荷バランスの基本方針

計算ノード間の負荷バランスは、プログラム制御で行う方針を取った。すなわち、ハードウェアや言語実装中のスケジューラが完全自動で行うという方針は取らなかった。その理由は二つある。

一つは、KL1 言語が、ソフトウェアで負荷分散やスケジューリング制御を効率良く記述できる機能を備えており、また実行時においても十分高い効率が実現できているためである。負荷バランス等については、特に動的な問題に関して多くの研究課題が残されており、応用ソフトウェアのレベルで、効率良くそれらの実験が出来るメリットは大きい。

二つめは、分散メモリアーキテクチャの計算機では、計算の局所性の維持が特に重要になるためで、完全自動の負荷バランスを指向するよりも、より良い負荷バランスの方針を何らかの方法でプログラムが示す方が現実的であるとの判断による。

応用問題領域によって、良好な負荷バランス方式が確立したものについては、それを負荷バランスユーティリティ・プログラムとして提供することにより、プログラマがいつも負荷バランスのコードを記述する必要はなくなる。

3.7 ハードウェアアーキテクチャの特徴付け

PIM ハードウェアのアーキテクチャ上の特徴を以下にまとめる。効率の良い記号処理を指向した部分、動的で

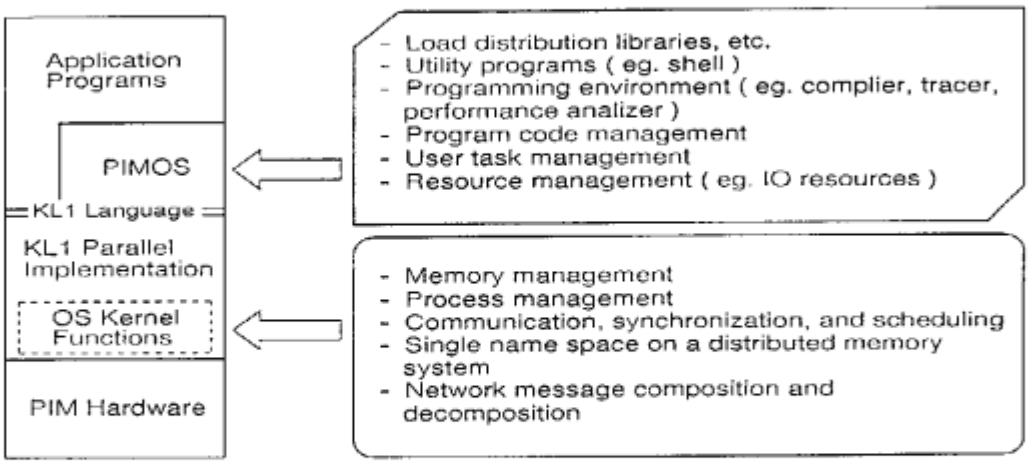

図 2: KL1 の実装と OS 機能の関係

均質さの低い大量計算問題向けの部分などがあるが、標準的構成要素技術を用いた部分もある。

(1) 分散メモリ型 MIMD マシン： PIM の構造をマクロに見るならば、分散メモリ構造を持つ大規模 MIMD 計算機である。最大数百の計算ノードが高速のネットワークで接続された構造を持つ。基本的な方式の選択の一つとして分散メモリ構造を選んだが、規模の拡張性の良さ、実現の容易さ、局所的なメモリ管理と大域的管理を分離する場合に適した構造であること、などが主な選択理由である。

(2) クラスタ構造： 8 台のプロセッサが共有バスと共有メモリにより密結合されたのがクラスタである。多数のクラスタが高速のネットワークで接続され、マルチ・クラスタ構造により全体システムを構成する。一つのクラスタの中では、言語システムの持つスケジューリング機能によって自動的な負荷バランスを実現しているため、プログラマは一つのクラスタを大きな計算能力と大容量メモリを持った一つの計算ノードとして取り扱うことが出来る。

クラスタ構造は、クラスタ内のプロセッサ間通信に関して、遅れ時間が小さくバンド幅が広いという特徴を提供している。クラスタ構造にはつぎのような利点がある。一つめは、クラスタ内では局所性の低い問題でも効率良く扱えることである。この種の問題については、クラスタという部分構造を持つ方が、全体が分散メモリ構造からなるシステムに比べ、通信オーバーヘッドが低減できること、相対的に小さな問題でも等しい並列処理効率が得られることの 2 点で優れる。

二つ目は、メモリの利用効率を高め、分散メモリ接続に比べて少ないメモリで等しい効率を得られることである。プロセッサがバンド幅の広い通信路で接続されているために、プロセッサ当たりの必要メモリ量を小さくできる。またプロセッサ毎の必要メモリ量が動的に変動しても、8 プロセッサでメモリ共有するため全体の必要量は平均化される。メモリのコストがシステム中に占める割合は極めて大きいため、メモリサイズの低減はシステムコストの削減に効果が大きい。

マルチ・クラスタ構造の欠点といえば、一階層の分散メモリ接続に比べ構造が複雑なことであるが、共有バス共有メモリによる密結合プロセッサは次第に標準的なシステム構成要素になりつつあり、この問題は早晩解消されるであろう。

(3) プロセッサ性能に対し大容量のメモリ： 我々の対象とする問題は、遠隔プロセス間で非同期的な通信を発生することが多いが、このような問題では計算ノード当たりに必要なメモリ量は、データ並列性を持つ問題の同期的処理の場合に比べ大きくなる。すなわちプロセスが通信の応答を待つ平均時間は同期的処理に比べて長く、プロセッサの稼働率を維持するためには、待ち時間に処理するための余分な仕事が必要になる。その分大きなメモリを必要とする。

(4) 高速ネットワーク： MIMD 型並列計算機における高速の接続ネットワークは、すでに標準的な技術が確立されつつある。しかしながら、我々の対象とする問題では、データ並列性を備えた問題の同期的処理と比べ、プロセッサ負荷とネットワーク負荷の重さの比率が異なっているようである。分散メモリハードウェア上に大域名前空間を実現するような処理は、ネットワークメッセージの送受の度に名前の管理テーブル参照などのプロセッサ負荷を発生する。つまり単純なデータ転送だけの場合に比べプロセッサの相対負荷が大きく、ネットワーク通信のバンド幅はしばしばプロセッサ性能で制限される。ネットワークバンド幅への要求はその分小さくなっている。

他方 KL1 で書かれた応用では、遠隔プロセス間の同期のために少量のデータ転送が高い頻度で発生する。

この場合、バンド幅より応答時間の短さが重要になる。

(5) 並列キャッシュメモリ： クラスタ内の各プロセッサは、ライトバック型の並列キャッシュメモリを備える。基本的なキャッシュ制御方式は、標準になりつつある技術の一つを用いるが、幾つかの最適化と工夫を備える。プロセッサごとに独立のタクスを実行する場合に比べ、一つの問題を複数プロセッサが協力して解く場合には、プロセッサ間通信に起因するバス負荷が極めて大きく、バス負荷の低減が課題となる。共有データの参照パターンの特徴を生かしてバス負荷を低減するバスプロトコル等を工夫している。また排他制御機構として、キャッシュの状態制御を利用した語単位のロック機構を実現している。

(6) 専用プロセッサ： プロセッサは、タグ操作やデータ型判定機構、参照ポインタを自動的に手繰り寄せる機構など、KL1 を始めとする記号処理言語の実現に適した機構を備える。タグ操作機構は、単一代入言語で暗黙の同期を実現する場合にも役立つ。

プロセッサは、中間言語 KL1-B にコンパイルされたプログラムを高率良く実行するために設計された機械語命令を備える。

パイプライン実行制御や RISC 命令の技術も用いているが、これらは標準的技術である。

4 マシンアーキテクチャとハードウェア

システムの全体像について前節で述べてきたが、本節では、各 PIM モデルのアーキテクチャとハードウェア仕様を紹介する。

4.1 5つの PIM モデルの概要

5種の PIM モデルを開発してきたが、それぞれ異なるアーキテクチャおよび要素技術の組み合わせで実現しており、モデル毎にそれぞれ異なる研究開発上の役割を与えている。

PIM/p：PIM/p は最も規模の大きいモデルで、最大512台の要素プロセッサを接続できる。PIM/p は、アーキテクチャの研究用とソフトウェアの研究開発環境実現の両方の目的を持つ。

PIM/p は図3 のマルチクラスタ構成を取る。最大64クラスタが接続できる。接続ネットワークはハイパーキューブ構造であり、2組の同仕様のネットワークを各クラスタに接続してネットワークバンド幅を広げている。

クラスタは8台のプロセッサを含み、それらが共有バスと共有メモリで密結合されている。プロセッサは並列キャッシュ、ネットワークインターフェース NIU、I/O デバイスインターフェース (SCSI バス) を含む [Kumon *et al.* 92]。

PIM の要素プロセッサはどのモデルでも SCSI バスを持ち、フロントエンドプロセッサ (FEP) とハードディスクが接続される。FEP は PSI-UX [Nakashima *et al.* 90] であり、高機能のマン・マシン・インターフェース装置として機能する。

PIM/m：PIM/m はソフトウェアの研究開発マシンであり、マルチ PSI/V2 とのあいだに、オブジェクトコード レベルでの互換性を備えている。図4 に示すように、最大で 256 台の要素プロセッサが 2 次元格子ネットワークで接続される。最大 20 GB (32台) のハードディスクと、たくさんの FEP が接続できる [Nakashima *et al.* 92]。

PIM/c：PIM/c もマルチクラスタ構造を取り、最大 32 クラスタ、256 台のプロセッサが、クロスバー・ネットワークで接続される [Nakagawa *et al.* 92]。アーキテクチャ研究と、一部のソフトウェア開発に使用する。

PIM/k：PIM/k はクラスタ内のアーキテクチャを研究するためのシステムである。階層構成を持つキャッシュを実現しており、クラスタ構造を取りながらより多くのプロセッサを接続することを目指している [Sakai *et al.* 91]。4台のプロセッサがローカルバスとセカンド・キャッシュを共有し、ミニ・クラスタを構成する。4つのミニ・クラスタが共有のメモリバスを介して共有メモリに接続されるている (図5)。

PIM/i：PIM/i も、クラスタ内のアーキテクチャ研究用のシステムである。LIW 型の機械語命令を持つプロセッサや、放送型の並列キャッシュを試す (他の PIM の並列キャッシュは、無効化型である)[Sato *et al.* 92]。

各 PIM モデルの全体的構造について表1 にまとめる。プロセッサ、ネットワーク、キャッシュシステムなどの構成要素の仕様について、以下の各章で述べる。

4.2 要素プロセッサ

KL1 言語を実行するときは、実行時のデータ型の判定を頻繁に行う必要があるため、PIM の要素プロセッサはいずれもタグ・アーキテクチャをとっている。これはマルチ PSI も同様である。

PIM/p、PIM/i、PIM/k の要素プロセッサは RISC 型の命令セットを備え、PIM/m と PIM/c はマイクロプログラム制御により CISC 型の命令を実現している (表2)。前者の機械語命令は、KL1-B よりさらに低いレベル (単純) であり、後者は、KL1-B がほとんどそのまま命令セットになっている。

PIM/p のプロセッサは、マクロ・コールと呼ぶ特徴のある機能を備えており、低いコストのサブルーチン呼び出しが可能である [Kumon *et al.* 92] [Shinogi *et al.* 88]。これによりコンパイル結果のオブジェクトコードサイズを小さく抑えながら、実行時オーバーヘッドも増加させ

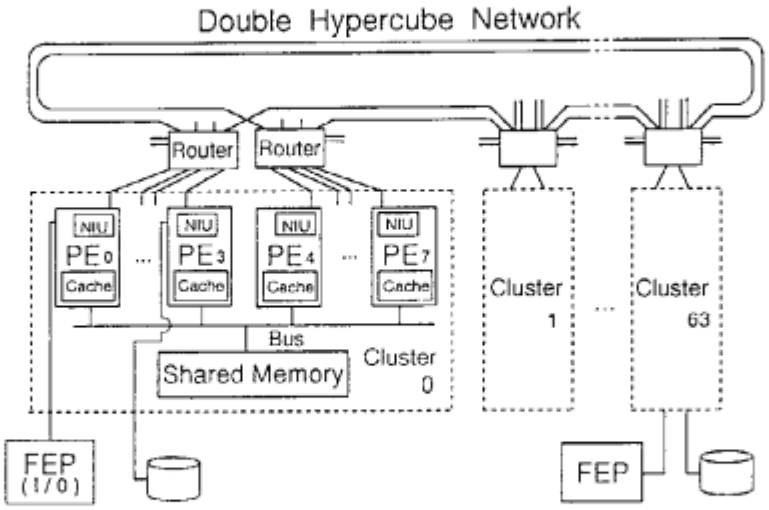

図 3: PIM/p の全体構造

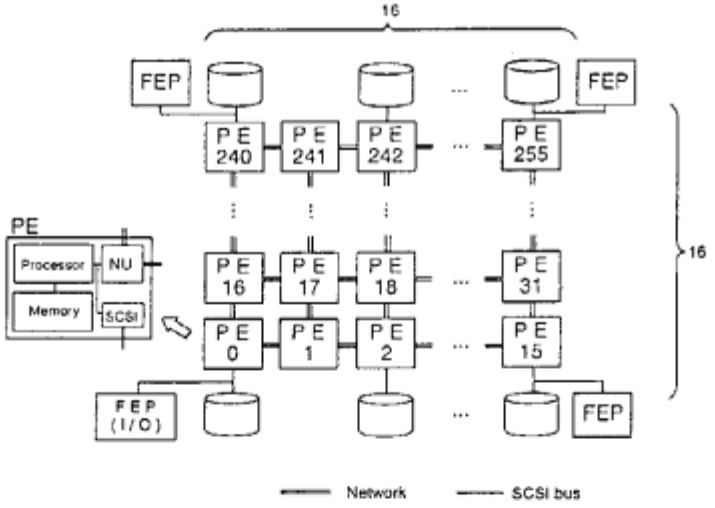

図 4: PIM/m の全体構造

表 1: 各 PIM の全体構造のまとめ

	Topology	Number of Clusters	Total Number of PEs	Memory Size/Cluster
PIM/p	hypercube × 2	64	512	256 MB
PIM/m	mesh	256	256	80 MB
PIM/c	crossbar	32	256	160 MB
PIM/k	—	1 †	16	1 GB
PIM/i	—	2	16	320 MB
Multi-PSI/V2	mesh	64	64	80 MB

(† : four mini-clusters included)

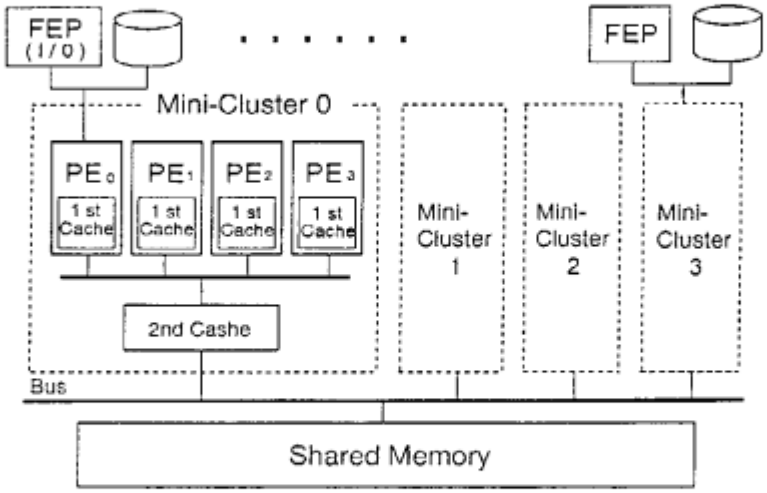

図 5: PIM/k の全体構成

表 2: 要素プロセッサの仕様

	Instruction set	Cycle time	LSI fabrication	Line interval
PIM/p	RISC + macro instruction	60 nsec †	standard-cell	0.96 μm
PIM/m	CISC (micro programmable)	65 nsec	standard-cell	0.8 μm
PIM/c	CISC (micro programmable)	50 nsec †	gate-arrays	0.8 μm
PIM/k	RISC	100 nsec	custom	1.2 μm
PIM/i	RISC	100 nsec †	standard-cell	1.2 μm
Multi-PSI/V2	CISC (micro programmable)	200 nsec	gate-arrays	2.0 μm

(† are design specifications. They are under testing with longer cycle time.)

ないで済む。PIM/p は他にも KL1 実装用の命令として、ポインタを自動的に手繰る命令や、実行時塵集めの MRB 方式 [Chikayama *et al.* 87] をサポートする命令などを備える。PIM/p のプロセッサは 4 段のパイプライン実行を行っている。

PIM/m [Nakashima *et al.* 92] のプロセッサは、マイクロプログラム制御で 5 段のパイプライン実行を行う。命令セットは KL1-B そのもので、マルチ PSI/V2 とオブジェクトコード レベルでの互換性が有る。高度なデータ型判定機構や、ポインタを自動的に手繰る機能を持つ。

PIM/i のプロセッサは、LIW(long instruction word) 型の命令セットを実現している。

4.3 ネットワーク

各 PIM のネットワークの仕様を表 3 にまとめる。

PIM/p では各プロセッサが NI を持ち、4 個の NI がネットワークノードを成すルータに接続されている。二組のハイパーキューブ・ネットワークを使用して広いバンド幅を実現している。

PIM/m はマルチ PSI と同じく 2 次元格子ネットワークを用いる。PIM/p と PIM/m は wormhole routing と呼ばれるメッセージ転送方式を取り、可変長の行き先アドレス付きメッセージが、ネットワークノード毎に動的に経路制御を受けて、目的プロセッサに届けられる。

PIM/c はクラスタに 1 台ずつ、クラスタ・コントローラと呼ばれる特別のプロセッサを持つ。このプロセッサは共有バスに接続され、ネットワークインターフェースの役割を果たす。

4.4 キャッシュシステム

KL1 プログラムはプロセッサ間で非同期的な通信を頻繁に発生し、これが共有バスの負荷をたいへん重くする。これを軽減するために、並列キャッシュのプロトコルを最適化した [Goto *et al.* 89] [Matsumoto *et al.* 87]。バスの使用率低減に効果がある [Nishida *et al.* 90]。すべての PIM は、ライト・バック型のキャッシュプロトコル

表 3: ネットワーク

	# PEs in a cluster	# NIs in a cluster	Transfer Rate †
PIM/p	8	8	33 MB/sec ‡ ×2
PIM/m	1	1	8 MB/sec
PIM/c	8	1	40 MB/sec ‡
PIM/k	16	—	—
PIM/i	8	1	—
Multi-PSI/V2	1	1	10 MB/sec

(PE = processing element, NI = network interface)
(†: per channel, full duplex ‡: design specifications)

表 4: キャッシュシステムの仕様

	Coherence Control		Mapping	Cache Size	
	Protocol	# States †		Instruction	Data
PIM/p	invalidation	4	4 way	64 KB	
PIM/m	—	—	direct-map	5 KB	20 KB
PIM/c	invalidation	5	2 way	80 KB	
PIM/k	hierarchical invalidation	4	(1st) direct-map	128 KB	256 KB
			(2nd) 4 way	1 MB	4 MB
PIM/i	broadcasting	6	direct-map	160 KB	160 KB
Multi-PSI/V2	—	—	direct-map	20 KB	

(† does not include *locking* state.)

を使用している (Table 4)。またキャッシュブロックの状態制御を活用した、語単位のロック機構 (排他制御機構) を効率良く実現している。

5 KL1 言語実装方式

KL1 言語は、動的で均質さの小さい大量計算問題を効率良くプログラム化し実行するのに適した、多くの優れた機能を持っている。このことは 3.4 節で述べたとおりである。KL1 の言語実装は、それらの特徴のある機能を効率良く実現することに最大の努力を払っている。

本章では、言語の実装方式およびテクニックについて、主要技術の中身の一つ一つを簡単に紹介してゆく。それらは、3.5 節で述べた言語実装方式の特徴に対応するものであるが、本章では、言語実現のための幾つかのキーとなる方式について、より具体的なイメージを与えることを目的としている。より詳細な情報は、別稿に報告されている [Hirata *et al.* 92] [Nakajima 92]。

5.1 KL1 の実行モデル

言語実装方式の理解を助けるため、KL1 の実行モデルについて簡単に紹介しておく。KL1 プログラムは、次のような形をした節の集合として実現される。

$$\underbrace{H : -G_1, ..., G_m}_{guard\ part} \mid \underbrace{B_1, ..., B_n.}_{body\ part}$$

ここで H はヘッド、G_i はガード・ゴールであり、これらをまとめてガード部と呼ぶ。B_i はボディ・ゴール、"|" をコミットメント・オペレータと呼ぶ。

実行されるべきゴールがあり、それとマッチするヘッドを持った節があると、以下の実行が行われる。

節のガード部は、パターンマッチと条件テストを行う。複数の候補節がある場合、それらのガード部は逐次にテストされる。ある候補節がパターンマッチと条件テストに成功すると、その節はコミット (選択) される。このとき元のゴールが、選択された節のボディ・ゴールと置き換えられる。これらのボディ・ゴールは並列に実行される (AND 並列)。KL1 の節は、実行しようとするゴールを節中のボディ・ゴールに書き替える書き替え規則と見なすことができる。

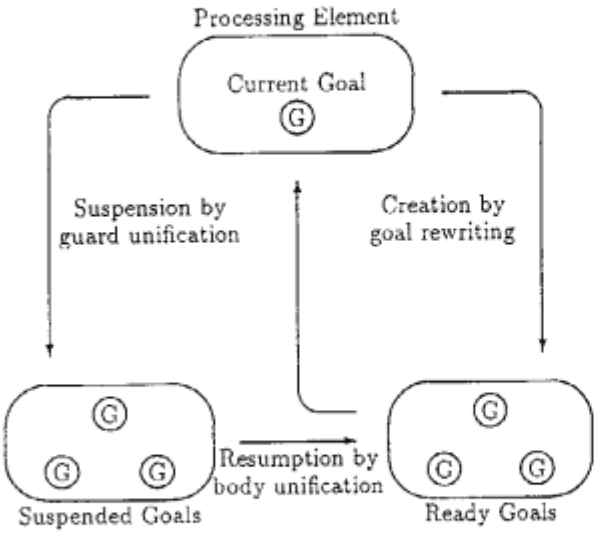

図 6: KL1 の実行モデル

KL1 の実行モデルを図6 に示す。これから書き替えられるべき実行可能ゴール (ready goal) はまとめて管理されている。その中から一つが実行のために取り出される (current goal)。そのゴールとマッチする節が見つかり、条件テストに成功して選択されると、ボディ・ゴールへの書き替えが起こる。書き替えられたゴールは、実行可能ゴールの集合に戻される。

複数のゴールは、論理変数を共有することができ、これらの変数を用いてゴール間の通信や同期を行う。ここで、変数を共有する二つのゴールを考えよう。一方のゴールの書き替えの中で生まれたボディ・ユニフィケーションは、変数に値を決める。もう一方のゴールの実行で起こるガード・ユニフィケーションは、変数の値をテストする。これが二つのゴール間の通信である。

もしガード・ユニフィケーションの前に変数に値が決まっていないと (そして他に選択される節がないと)、実行は変数に値が決まるまで中断される。これが同期のメカニズムである [Ueda *et al.* 90]。

5.2 暗黙の通信の実現

計算ノード間の暗黙の通信を実現するために、いくつかの重要なメカニズムが用いられている。

一つの計算ノードの中で、変数を共有している二つのゴールがあるとしよう。それぞれのゴールは、変数への参照ポインタを持っている。ゴールの一つが別の計算ノードに移される場合、元の変数への遠隔参照ポインタが自動的に生成される。別の計算ノードに置かれたゴール間の暗黙の通信は、この遠隔参照ポインタに沿って行われる。

暗黙の通信を実現するための主な技術要素を以下に示す。

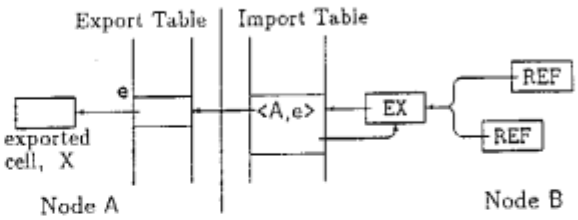

図 7: 輸出入表

5.2.1 大域名前空間

暗黙の遠隔参照管理が、論理変数、構造体データ、プログラムコードに対して実現されている。これは、分散メモリハードウェアの上の、仮想的な大域名前空間の実現と表現することもできる。

この機能は、輸出入表によって実現されている。輸出入表は、一種の間接参照を集めたテーブルであり、計算ノード内のローカルなアドレス系と、外部からその計算ノードを指す遠隔参照とを分離する働きを持つ (図 7)。遠隔参照 (外部参照) は、⟨A,e⟩ として表現される。ここで A は、参照しようとする変数の存在しているノードの番号、e は、輸出表の登録番号である。輸出表への登録は、新しい遠隔参照の生成が必要になった時点で、動的に行われる [Ichiyoshi *et al.* 87]。

もし、ローカルな塵集めが起きて参照されている論理変数のセルアドレスが変化したとしても、輸出表の登録番号 e は変わらない。

同一の論理変数に対して繰り返し輸出入が発生した場合、輸出入表への再登録を省略して、以前の登録番号を用いる最適化を行っている。これにより、変数の参照に関するノード間のデータ転送回数が削減できている [Ichiyoshi *et al.* 88]。

5.2.2 暗黙のデータ転送

ユニフィケーションに起因するデータ転送： 計算ノード間のデータ転送は、普通はユニフィケーションにより引き起こされる。

ガード・ユニフィケーションは、論理変数の値を確かめようとする。このときそれが外部参照 (図 7中の EX) であると、読出し要求メッセージ `%read(X,ReturnAddress)` が、ノード A に送出される[1]。ここで X は、外部参照 ⟨A,e⟩ であり、ReturnAddress は、ノード B の輸出表に新しく登録された登録番号である。

上記のガード・ユニフィケーションの元となったゴール実行は、(他に選択される節がなければ) 中断される。

読出し要求メッセージが到着したとき、参照されている変数が値 V を持つ場合、その値は直ちに `%answer_value( ReturnAddress, V )` メッセージによって返される。値 V が返ってくると、中断中のゴール実行は再開される。

もし読出し要求メッセージ到着時に、被参照変数が値をまだ持っていない場合、値が決まるまで、読出し要求が待たされる (ゴールの中断と同様)。

[1]正確には、直ちに選択される節が他にないことが分かってからメッセージ送出が行われる。

ボディ・ユニフィケーションによって、遠隔の論理変数 X とあるターム Y をユニファイしようとした場合、メッセージ %unify(X, Y) が参照先の計算ノードに送られる。ここで Y がアトミックなデータである場合には、単なるデータ転送となる。

もし Y が別の計算ノードへの外部参照であり、さらに X にも Y の参照先にも値が決まっていない場合、つまり別々の計算ノードに置かれた、値を持たない論理変数どうしのユニフィケーションの場合、計算ノード間で外部参照のループができる危険性がある。これを解決するために、ノード番号の大小を利用して、外部参照ポインタの向きを制御する方法を取っている [Ichiyoshi *et al.* 88]。

遅延転送 (Lazy Transfer)： 構造体データをノード間で転送する時、1 レベル転送を実施している。構造体の要素は、アトミックなデータか入れ子になった構造体である。アトミックなデータは、コピーしてそのまま送られる。一方入れ子になった構造体は、すぐには送られず、外部参照ポインタの形で渡される。これを 1 レベル転送と呼んでいる。この方針は、「大きいデータの転送はそれが本当に必要になるまで、なるべく遅らされるべきであり、それにより、データを使わないかも知れない計算ノードへの不要な転送を避ける」という考え方に基づいている。

プログラム・コードの転送 プログラム・コードは、一種の大きな構造体として扱われる。プログラム・コードは、最初はローダ・プログラムにより、どこか 1 か所の計算ノードにロードされ、計算はそのノードで開始される。すべての KL1 のゴールは、対応するプログラム・コードへの参照ポインタを内蔵している。もし、あるゴールが別の計算ノードに移され、そのノードに対応するプログラム・コードが無い場合、ゴール実行は中断される。そしてそのゴールが持つコードへの外部参照ポインタを利用して、プログラム・コードは動的に取ってこられる。

コードに関しては、繰り返しの輸出入に対して同一の輸出入表登録番号を用いる方法を必ず実施している。これを守る限り、外部参照識別子をコードの識別に利用できる。

5.3 小粒度並列プロセス

5.3.1 プロセスのグループ管理

KL1 のゴールは、一種の**軽いプロセス** (lightweight process) と考えることができる。効率の良い並列処理のためには、十分な並列性が必要であり、利用者のタスクは十分な数の軽いプロセスを含む必要がある。一方オペレーティングシステムは、これらの軽いプロセスの集合を一つのまとまったタスクとして、一括管理、制御できなければならない。これを支援するのが荘園であり、ゴールの集合に対する実行制御、資源管理、状態監視などのメタ機能を実現している。

荘園と里親： すべての KL1 のゴールは、いずれかの荘園に属さねばならない。里親とは、荘園機能をローカルに実現する代理であり、ある荘園所属のゴールが存在する計算ノードには、かならず代理としての里親が生成されている。

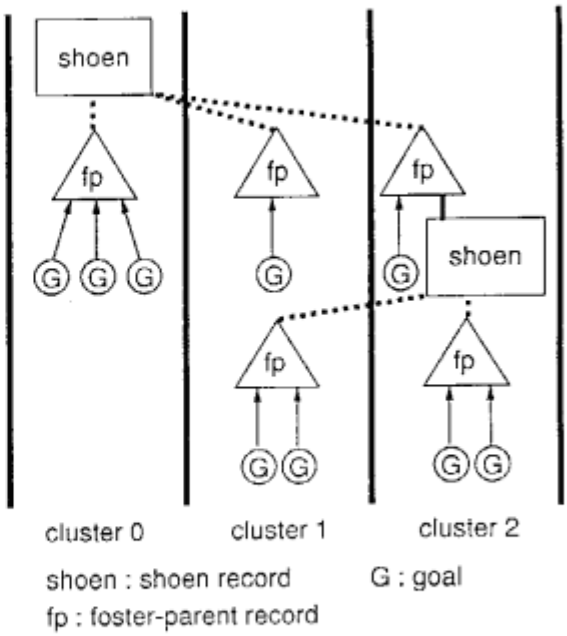

図 8: 荘園と里親とゴールの関係

すべてのゴールは、同一ノード内の里親を指すポインタを持ち、然るべき周期で (例えばゴール書き替えの度に)、メタ制御の要求の有無を里親に確認する。図 8 に、荘園、里親、ゴールの間の関係を示す。

荘園と里親はそれぞれ、タスクの実行制御、資源量、ゴール数などに関する状態を保持する。もし、各ゴールが直接に荘園を参照すると、頻繁な通信が起こると共に、荘園参照のボトルネックが発生し得る。里親はこれを回避するために置かれている。

終了検出： ゴールの集合の終了を検出することは、並列処理の中でも実現に注意を要するものの一つで、ネットワーク上を転送途中のメッセージがある場合には特に難しい。たとえばすべての里親が、それぞれの管理するゴールの終了を荘園に報告してきても、もしネットワーク上を移動中のゴールが存在するならば、荘園は終了してはならない。

一つの解は、Weighted Throw Counting (重み付け送出カウント法 = WTC) であり、これは重み付け参照カウント法 (Weighted Reference Counting = WRC) の一つの応用である [Rokusawa *et al.* 88] [Watson *et al.* 87]。

5.3.2 ゴール・スケジューリング

ここで述べるゴール・スケジューリングは、荘園によるゴール集合の管理とは別の概念である。ゴール・スケジューリングは、ゴールの状態遷移の実現のことであり、実行可能 (ready)、実行中 (execution)、中断中 (suspension) の間での状態移動が管理される。実行優先度についても合わせて管理される。

スケジューリングの基本方式： 実行可能ゴールは、実行可能ゴール・スタック (ready goal stack) と呼ばれるLIFO 型のキューで管理される。スタックの一番上から実行可能ゴールが取り出され、書き替えが行われ、書き替えられたゴールは再びスタックに戻される。すなわちゴールの書き替え順に関して深さ優先のスケジューリングになっている。

論理変数が値を持たないためにゴール実行が中断すると、そのゴールは中断の原因となった論理変数にリンクされて、変数が値を持つのを待つ。その間は、別のゴール実行が行われる。変数に値が決まると、中断していたゴールは実行可能となり、実行可能ゴール・スタックに戻される。

ゴールの実行優先度が、プラグマによって指定できる。実行可能ゴール・スタックは、優先度の数の分だけ用意される。

クラスタ内負荷分散： クラスタ内については、自動負荷バランスを試みている。クラスタにつき一本のゴール・スタックだけを用意すると、プロセッサ間でゴール・スタック更新の競合が起こるので、最高優先度ゴール用のゴール・スタックをプロセッサ毎に用意した。最高優先度のゴールが、プロセッサ間の負荷を均等化するようにクラスタ内に分散される [Sato *et al.* 87] [Hirata *et al.* 92]。

別クラスタへのゴール送出： プラグマ付きのボディ・ゴール goal@node(CL) は、%throw メッセージによりクラスタ CL へ送出される。受け取ったクラスタ (正確にはクラスタ中の一つのプロセッサ) は、受け取ったゴールを実行可能ゴール・スタックに入れるとともに、里親とリンクする。このときもし里親がなかったなら、新たに生成する。

5.4 メモリ管理

動的なメモリ割り付け、解放、塵集め等のメモリ管理は、記号処理言語の実現にも、並列処理に適した単一代入言語の実現にも無くてはならない。

5.4.1 MRB による実行時塵集め

MRB 方式は、参照カウント法のサブセットであり、ポインタ・タグの中の 1 ビットで参照数が 1 かそれより多いかを区別する [Chikayama *et al.* 87] [Inamura *et al.* 88]。塵になってしまうセルの中で、参照数が 1 のものについては、実行時に逐一回収できる。論理変数の使い方の特徴として単一参照は多い。

MRB は構造体データの要素更新時の最適化にも使用している。KL1 は単一代入言語であるため、構造体の一部を更新したい場合、意味上はすべての要素をコピーして新しい構造体データとしなければならない。しかしながら構造体の参照数が 1 の場合だけは、同構造体を参照するゴールが他にはないので、直接書き替えをしたうえで表面上は新たな構造体を生成したように見せている。これにより、特に、大きな構造体の部分更新のコストが削減できた。

5.4.2 クラスタ内塵集め

MRB 方式の実行時塵集めと合わせて、クラスタ内では停止回収型のコピー方式に基づく塵集めを併用している。MRB 方式では、取りきれない塵が出るからである。このコピー方式に基づく塵集めについては、並列処理を試みている [Imai *et al.* 91]。

5.4.3 WEC によるクラスタ間塵集め (輸出表の解放)

遠隔参照を実現する輸出表のエントリのうち、不要になったものを実行時に逐一回収するために、重み付け輸出カウント (Weighted Export Count = WEC) 法を実現した [Ichiyoshi *et al.* 88]。これは、既に述べた WRC 法の応用の一つである。この方式の利点の一つは、完全な参照カウント法を用いるのに比べて、カウント数更新に要するメッセージ数が少ないこと、もう一つは、一旦輸入したデータを再輸出する時のメッセージ数も削減できることである。

5.5 抽象命令セット KL1-B

KL1-B は KL1 プログラムを効率良く実行するための中間言語 (抽象命令セット) で、各 PIM に共通である。KL1-B の役割は、Prolog の場合の WAM [Warren 83] と同様である。KL1-B の命令セットについては他に報告されている [Hirata *et al.* 92] [Kimura *et al.* 87]。

これまで述べてきた KL1 実装に関する数々の方式は、そのほとんどが、KL1 言語システムの中の実行時ルーチンとして実現されており、KL1-B 命令の途中または命令と命令の隙間で、然るべき条件が成立すると暗黙の内に呼び出される。

PIM 用の KL1 コンパイラは次の 2 段階の処理を含む。はじめに KL1 プログラムを KL1-B にコンパイルする。次に KL1-B コードを機械語に再変換するとともに、実行時ルーチンとリンクする。

6 計測と評価

本節では、並列推論マシンと言語システムに関する、幾つかの計測評価結果について述べる。計測は、小粒度の並列プロセスおよび遠隔の通信・同期が、低いコストで実現できている点に焦点を当てる。ベンチマークプログラムを使った性能測定の結果も紹介する。これには PIM/m 上の最新の計測も含まれる。

6.1 マルチ PSI/V2 上の計測と評価

KL1 の言語実装は、いわゆる OS 核の機能を包含していることは、3.5 節で述べたとおりである。実際に 5 章で示した実装方式のほとんどは、OS 核の機能に関係している。これらが効率良く実現できていることは、動的で均質さの低い大量計算問題を KL1 を用いて本当に効率良く並列処理するための必要条件である。

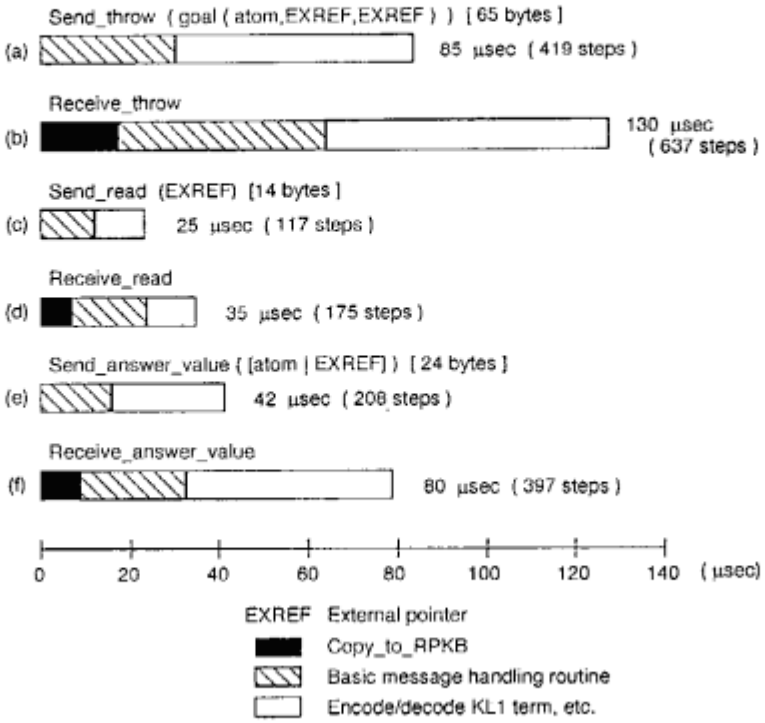

図 9: メッセージ処理コスト

表 5: メッセージ頻度とリダクション数

Pentomino (39.3 KRPS on 1 PE)

Num of PEs	4 PEs	16 PEs	64 PEs
execution time (sec)	54.63	14.62	4.35
total reductions (×1000)	8,317.	8,332.	8,340.
reductions/sec (KRPS)	152.2	570.1	1,919.4
reductions/msg	221.	108.	88.
msg bytes/sec (×1000)	14.5	108.1	440.5

Bestpath (23.4 KRPS on 1 PE)

Num of PEs	4 PEs	16 PEs	64 PEs
execution time (sec)	10.655	4.062	1.691
total reductions (×1000)	987.7	1213.6	1,505.2
reductions/sec (KRPS)	92.7	298.8	890.1
reductions/msg	21.9	11.7	6.2
msg bytes/sec (×1000)	114.0	692.5	3,854.3

(KRPS: Kilo Reductions Per Second)

本節では、OS 核に関わる言語実装の幾つかが、実際にマルチ PSI/V2 上で効率良く実現できていることを測定データを用いて示す。計算ノード内でのゴールスケジューリング・コスト、ノード間での通信コスト、ベンチマークプログラムにおける通信オーバーヘッドについて以下に述べる。

マルチ PSI/V2 は 64 台のプロセッサを 2 次元格子ネットワークで接続したマシンである。計算ノードはクラスタではなく、1 台のプロセッサであることを念のために記しておく。

6.1.1 ノード内のゴールスケジューリング・コスト

プロセッサ内の、ゴールスケジューリングと同期のコストが報告されている [Onishi *et al.* 90]。

エンキュー / デキュー： 実行可能ゴール・スタックに対して、最も単純なゴールをエンキュー / デキューするコストは、5.4 μs (27 マイクロ命令ステップ) であった。これは、ゴールをスタックに入れるコストと取り出すコストの和である。

節が選択されてゴールの書き替えが起こると、書き替えられたゴールは実行可能ゴール・スタックに入れられ、実行のために取り出されるのを待つ。一般的な計算機システムの言葉で言えば、上記のコストは、プロセス生成 (フォーク) のコストの一部といえる。

単一待ち： 一つの論理変数に値が決まるのを待つためにゴールが中断するのを単一待ち (single-suspension) と呼んでいる。このコストは、14 μs (70 ステップ) である。

このコストは、中断されたゴールが、中断原因となった論理変数にリンクされるコスト、変数の値が決まってリンクから外され実行可能ゴール・スタックに入れられるコスト、取り出されるコストの合計である。

単一待ちのコストは、プロセッサ内におけるプロセス間同期のコストといえる。

多重待ち： 一つのゴールが、複数の論理変数のいずれかに値が決まるのを待つために中断することを多重待ち (multiple-suspension) と呼ぶ。二つの論理変数に対する多重待ちのコストは、28 μs (140 ステップ) である。

単一待ちに比べてのコストの増加分は、非決定性 (indeterminacy) を実現するためのコストといえる。

これらの低いコストが実現できたことにより、小粒度の非常に多数のプロセスを現実の応用に利用することが可能となった。次次節に紹介するベンチマークプログラムの最短経路問題は、その良い例である。測定されたゴールスケジューリング・コストは、プロセッサ内におけるプロセスの最小粒度 (性能的に許容できる粒度の下限) の目安を与える。

6.1.2 ノード間の通信コスト

プログラムから見た時の、異なる計算ノード間での通信コストについて、いくつかの基本処理の重さをマルチ PSI/V2 上で測定した [Nakajima *et al.* 90]。

別の計算ノードへのゴール送出は、`%throw_goal` メッセージにより実現される。これは軽いプロセスの遠隔呼び出しに当たる。ノードをまたがっての論理変数の値の読出しは、`%read` と `%answer_value` メッセージの組み合わせで実現される。これは、遠隔プロセス間の通信および同期に対応する。

図9 は、これら3種類の基本的なメッセージ処理にかかる時間を送出側、受取り側双方で測定したものである。処理時間はさらに、処理の内容により三つに分けて示してある。

Encode/decode KL1 term, etc. は、KL1 処理系が管理する内部データ表現からネットワークメッセージを組み立て、あるいはその逆を行うことである。輸出入表の参照・更新処理および到着したゴールを里親とリンクする処理なども含まれ、メッセージ処理機能の中心をなす。

Basic message handling routine は、1語40ビットのタグ付きデータと8ビット直列転送のネットワークメッセージデータとの間の、単純な語長変換を行う。ハードウェアバッファとの間の転送時間も含まれる。これらの処理の時間は、適当なハードウェア支援により削減可能なものである。

Copy_to_RPKB は、ハードウェアバッファが溢れそうな時だけ実行される処理で、ハードウェアバッファとソフトウェアバッファの間のデータの移し替えを行う。

ハードウェアの転送レートは、0.2 μs/byte(チャネル当たり、片道の速度) である。ネットワークノードを一つ越える時間は 1 μs 以下である。この値と図9中の処理時間を比べることにより、通信時間の殆どんどを上記のメッセージ処理が占めることが分かる。

メッセージ処理コスト： `Send_throw` (a) は65バイトの `%throw_goal` メッセージ送出処理の時間を示す。メッセージは3引数のゴールを含む。処理時間は419マイクロ命令ステップ、約85 μs (サイクルタイム = 200 ns) である。`Receive_throw` (b) は同じメッセージの受取り処理時間を示す。実行可能ゴール・スタックへ格納する時間を含む。

グラフの (c)、(d)、(e)、(f) はそれぞれ、`%read` メッセージと `%answer_value` メッセージの送出、受取り処理の時間を示す。

`%throw_goal` メッセージの送出、受取り処理時間の和は、合計 215 μs (1056 ステップ) である。これらは別の計算ノードに対して、軽いプロセスを遠隔呼び出しするコストと考えることができる。`%read` と `%answer_value` メッセージの送出、受取り処理時間の全ての和は、182 μs (897 ステップ) である。これは遠隔プロセス間で同期をとるためのコストである。

これらの値を前節の計算ノード内での処理コストと比べてみよう。遠隔プロセス間での同期のコストは、ローカルなプロセス間での同期コストに比べて、約10倍大きい。プロセスの遠隔呼び出しコストとローカルな呼び出しコストとの比は、10倍よりは大きそうであるが、40倍以下であることが分かっている (ローカルなプロセスフォークのコストの一部が示されているため)。このように遠隔処理 / ローカルな処理のコスト比が 10 〜 40 倍という程度に遠隔処理が軽いことにより、別の計算ノードにまたがって比較的粒度の小さい、または通信比率の大きいプロセスの間の並列処理をすることが、ある程度可能になったと言える。

遠隔処理のコストが示されたため、遠隔処理の発生頻度から処理オーバーヘッド (通信オーバーヘッド) を求めることが可能となる。逆に、処理オーバーヘッドの上限を決めてやると、遠隔処理の許容発生頻度の上限ないしは異なるノード間での粒度の下限の目安が与えられる。つまり負荷の均等化のためにはなるべく小さい粒度を用い

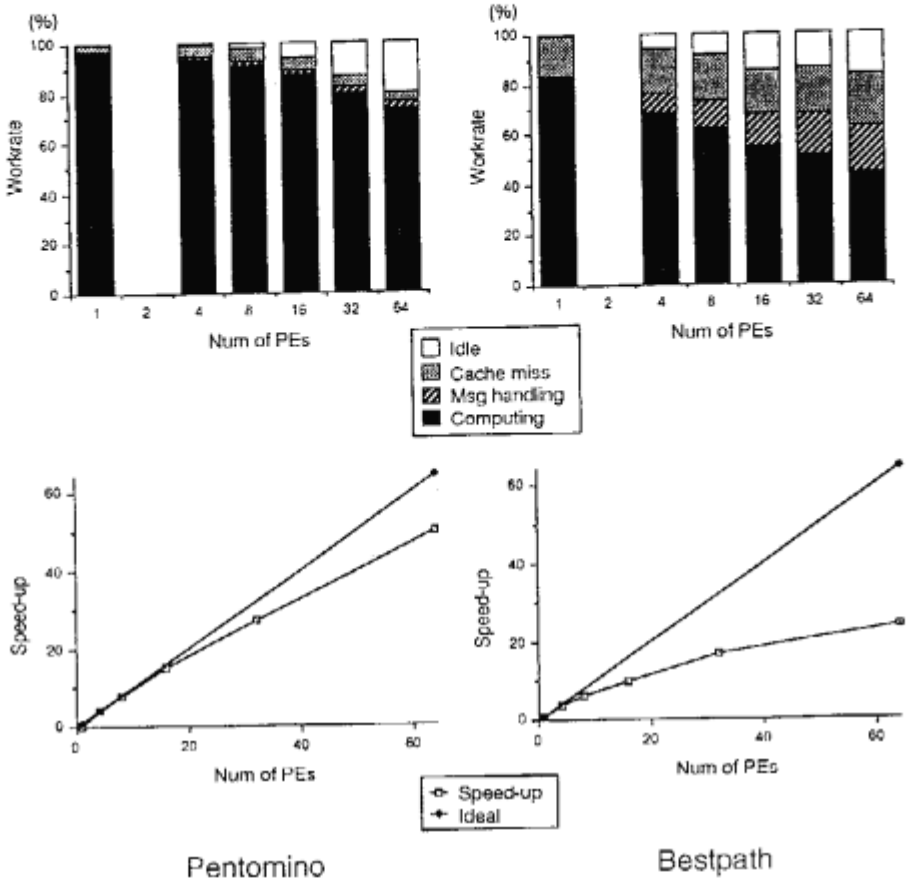

図 10: プロセッサ稼働率とスピードアップ

たいが、どの程度まで小粒度が許されるのかを考えることができる。

6.1.3 ベンチマークプログラムによる計測

ベンチマークプログラム： 使用した2種類のベンチマークプログラムについて簡単に説明する。

- **詰込みパズル (ペントミノ)：** 詰込みパズル (ペントミノ) の全解を探索する問題である。異なる詰込み方に対応する探索木のすべての枝を動的に拡張しながら並列探索する。兄弟の枝の間では通信の必要がなく、負荷バランスはとりやすい。2階層の動的負荷分散により良好な負荷バランスを実現している [Furuichi *et al.* 90]。

- **最短経路問題 (Bestpath)：** 160 × 160 の格子状のグラフが与えられ、グラフの各辺には非負のコストが付与されている。プログラムは、グラフ上の与えられた開始点から、他の全ての点の各々に至る最小コストの経路を同時並列的に探索する。最短経路探索の分散アルゴリズムを実現している [Wada *et al.* 90]。グラフの点の各々は、KL1 のプロセスとして実現されており、それらがメッセージを交換しながら経路探索を行う。プロセスの接続構造は静的であるが、計算負荷は動的に発生する。25,600 の小粒度のプロセスが協調しながら解を求める。

メッセージとリダクション： 表5 は、実行時間、リダクション (ゴールの書き替え) 数と頻度、通信メッセージの頻度等を示す [Nakajima *et al.* 90]。1回のリダクションの平均時間は、KRPS の逆数で与えられる。ここで KRPS は KLIPS と等価と考えて良い。1リダクションの時間は、詰込みパズルで 25 μs (127 ステップ)、最短経路問題で 43 μs (214 ステップ) である。これらはほぼ、計算ノード内におけるプロセスの平均粒度と考えることができる。64 プロセッサ使用時のメッセージの発生頻度は、詰込みパズルでは 88 リダクションに1回、最短経路問題では 6 リダクションに1回である。

これらの計測結果からネットワーク負荷が計算されている [Nakajima *et al.* 90]。10 Mbytes/s (full duplex のチャネル当たり) のネットワークバンド幅に対して、ネットワーク利用率の平均値は非常に小さく、詰込みパズルで 0.08 %、最短経路問題で 0.3 % であった。

表 6: PIM/m の単体プロセッサ性能

benchmark	condition	PIM/m	Multi-PSI/V2	$\frac{Multi-PSI/V2}{PIM/m}$
append	1,000 elements	1.63 msec	7.80 msec	4.8
best-path	90,000 nodes	142 sec	213 sec	1.5
pentomino	8 × 5 box	107 sec	240 sec	2.2
15-puzzle	5,885 K nodes	9,283 sec	21,660 sec	2.3

表 7: ペントミノ (詰込みパズル) の処理性能 (8 × 5 box)

No. of PEs	PIM/m		Multi-PSI/V2		$\frac{Multi-PSI/V2}{PIM/m}$
	Time	Speedup	Time	Speedup	
256 PE	1,124 ms	95.41			
128 PE	1,290 ms	83.13			
64 PE	2,162 ms	49.60	4,679 ms	51.20	2.16
32 PE	3,694 ms	29.03	8,278 ms	28.94	2.24
16 PE	6,910 ms	15.52	15,686 ms	15.27	2.27
1 PE	107,238 ms	1.00	239,545 ms	1.00	2.23

表 8: ペントミノ (詰込みパズル) の大きな問題における処理性能 (10 × 6 box)

No. of PEs	PIM/m		Multi-PSI/V2		$\frac{Multi-PSI/V2}{PIM/m}$
	Time	Speedup	Time	Speedup	
256 PE	103,655 ms	234.20			
128 PE	188,452 ms	128.87			
64 PE	359,268 ms	67.60	886,325 ms		2.47
32 PE	694,553 ms	34.96	1,729,430 ms		2.49
16 PE	1,367,240 ms	17.76			
1 PE	24,285,015 ms	1.00			

通信オーバーヘッド： 二つのベンチマークプログラムに対して、プロセッサの実行特性が計測された [Nakajima *et al.* 90]。

図 10 の中では、プロセッサ稼働率の内訳が次の 4 項目についてなされている。リダクション処理 (Computing)、メッセージ処理 (Msg handling)、キャッシュミスによる損失 (Cache miss)、アイドル時間 (Idle) である。全てのプロセッサの間の平均値が棒グラフに示されている。測定結果から得られるスピードアップのグラフも合わせて示されている。

詰込みパズルでは、2 階層の動的負荷分散を行っている。64 プロセッサを用いた場合、実行時間 4.35 秒の間に、数千個の小粒度プロセスが、プロセッサの稼働率を維持するために動的・適応的 (adaptive) に分散される。効率 78% (スピードアップ 50 倍) を得ている。64 プロセッサ使用時の稼働率の低下は、プロセッサに仕事を供給することの遅れ時間と、プロセス粒度のばらつきに因っている。

最短経路問題では、25,600 の小粒度プロセスが 64 プロセッサ上に静的に配置され、メッセージ交換を行いながら分散アルゴリズムを実行する。メッセージはグラフの開始点から 2 次元格子のグラフ上を波面状に広がるため、計算負荷は動的に発生する。負荷の均等化のために、グラフをプロセッサ数より十分多い断片に分断して、ランダムにプロセッサへ割り付けている。プロセッサ数が多い時の効率 (スピードアップ) の低下は、負荷のばらつき (アイドル時間の増加)、通信オーバーヘッド、キャッシュミスによる損失、見込み計算 (speculative computation) の増加に因っている。前 3 者は棒グラフに現れている。

表 5 と 図 10 から、最短経路問題では、6 リダクションに一度という高い頻度で通信を行っているが、その時の

通信オーバーヘッドは約 15% で、高くない値であることが分かる。プログラマは、現実にこの程度の頻度の通信を使用可能だといえる。この状態でのネットワーク使用率は 0.3% であり、通信が増加すると、ネットワーク負荷ではなくプロセッサ負荷の増大が性能を制限することになる。

6.2 PIM 上の計測結果の速報

6.2.1 単体プロセッサ性能

表 6 に、4 種のベンチマークにおける PIM/m の単体性能と、マルチ PSI/V2 との性能比を示す。PIM/m の単体性能は、マルチ PSI/V2 の単体性能に比べて、append を除き 1.5 〜 2.3 倍大きい。append で顕著に高いのは最適化の効果である。ワーキングセットの大きい問題の方が、性能比が小さくなる傾向を示している [Nakashima *et al.* 92]。

6.2.2 システム性能

表 7 と表 8は、詰込みパズルを実行したときの、PIM/m の処理性能の速報である。

表 7 におけるスピードアップの頭打ちは、問題サイズが小さいためである。表 8 の大きい問題では、256 プロセッサで 234 倍 (効率 91%) という高い性能を示している。表 7 の小さい問題の場合でも、多段の動的負荷分散を用いて、約 1.1 秒の実行時間の間に数千個の小粒度プロセスを分散させ、95 倍のスピードアップを得ている。これも注目すべき数値かも知れない。この結果からも、小粒度の多数のプロセスを効率良く扱える言語システムが実現されていると言えるであろう。

7 おわりに

本論文は二つのテーマについて述べた。一つは、並列推論マシン PIM と核言語 KL1 の言語実装に関する研究開発概要の報告である。もう一つは、我々のシステムが、知識処理を始めとする動的で均質さの低い大量計算問題の並列処理を効率良く実現するのに、極めて優れた特徴を持っていることを示し、それらの特徴の一つ一つを明らかにすることである。このような性質の問題は、市販されている大規模並列マシンやそのソフトウェアシステムではカバーされていない。すなわち PIM と KL1 システムは、このような新しい問題領域における大規模並列処理を指向しているといえる。

PIM は、グローバルには分散メモリ構造をとる MIMD 型並列マシンであり、最大で 512 台のプロセッサを接続する。複数のアーキテクチャのうち主要なものは、共有メモリのクラスタをネットワークで多数接続する階層構成である。対象問題領域における効率の良い並列処理を実現するための、多くの要素技術を内蔵し、特に記号処理に対して高い効率を示す。

KL1 言語は、動的で均質さの低い大量計算問題の並列処理に適した、多くの特徴のある機能を持つ。主なものは、(1) 小粒度並列プロセス、(2) 暗黙の通信と同期、(3) 論理的な並列性の記述と物理的な並列性の制御 (負荷割付など) の分離、などである。それらにより、並列性は高いが構造の複雑なプログラムの記述を容易にするとともに、負荷バランスに対する大きな柔軟性を実現している。効率の良い言語実装によって、KL1 の持つ特徴的な機能を現実のプログラム開発に適用できるようになった。PIM と KL1 システムは、対象問題領域の並列ソフトウェアの研究と開発のために、強力な環境を提供している。

計測評価結果によって、小粒度の並列プロセスや、それらの間の遠隔の通信・同期が、きわめて小さいコストで実現できていることが示された。またベンチマークプログラムの実行では、数多くの小粒度プロセスを動的に取り扱う問題において大きなスピードアップを実現している。これらの結果により、われわれのシステムが、動的で均質さの低い大量計算問題に対して高い効率を示すとともに、対象問題領域の並列ソフトウェア研究開発に有用であることが、おおむね確認されたと言える。

さらに進んだ計測評価が続けられており、結果は近いうちに発表されることになろう。一方で、並列ソフトウェアに関する研究課題は山積しており、動的で均質さの低い問題を対象とした大規模並列処理をあらゆるケースについて効率良く実現するには、特にソフトウェア面を強化した研究の継続・進展が是非とも必要である。そしてこのことこそが、プロジェクトが目指してきた 21 世紀の高度知識情報処理実現のための、一つの重要な鍵になることであろう。PIM と KL1 システムが、ますます重要性を増しつつある並列ソフトウェア研究を進めるための研究環境として、最大限に威力を発揮し貢献することを期待する。

謝辞

PIM と KL1 システムの研究開発は、ICOT 第一研究室の研究員と、関係各社の研究所・工場の人々を中心に、第二研究室、第七研究室を始め ICOT の各研究室および研究計画部の人々の協力と、PIM WG 諸兄の側面援助のもとに進められた。著者は、これらの人々のたゆまぬ努力と協力に、最大限の敬意と感謝の念を捧げる。また終始 指導と励ましを続けてくださった内田俊一研究部長、渕一博研究所長に心から感謝する。

参考文献

[Chikayama *et al.* 87] T. Chikayama and Y. Kimura. Multiple Reference Management in Flat GHC. In *Proc. of the Fourth Int. Conf. on Logic Programming*, 1987, pp.276-293.

[Chikayama 92] T. Chikayama. Operating System PIMOS and Kernel Language KL1. In *Proc. of the Int. Conf. on Fifth Generation Computer Systems*, 1992.

[Furuichi *et al.* 90] M. Furuichi, K. Taki and N. Ichiyoshi. A multi-level load balancing scheme for or-parallel exhaustive search programs on the Multi-PSI. In *Proc. of PPoPP'90*, pp. 50-59, 1990.

[Goto *et al.* 88] A. Goto, M. Sato, K. Nakajima, K. Taki and A. Matsumoto. Overview of the Parallel Inference Machine Architecture (PIM). In *Proc. of the Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, 1988, pp.208–229.

[Goto *et al.* 89] A. Goto, A. Matsumoto and E. Tick. Design and Performance of a Coherent Cache for Parallel Logic Programming Architectures. In *Proceedings of 16th Annual International Symposium on Computer Architecture*, pages 25 – 33, Jerusalem, Israel, 1989.

[Hirata *et al.* 92] K. Hirata, R. Yamamoto, A. Imai, H. Kawai, K. Hirano, T. Takagi, K. Taki, A. Nakase and K. Rokusawa. Parallel and Distributed Implementation of Concurrent Logic Programming Language KL1. In *Proc. of the Int. Conf. on Fifth Generation Computer Systems*, 1992.

[Ichiyoshi *et al.* 87] N. Ichiyoshi, T. Miyazaki and K. Taki. A Distributed Implementation of Flat GHC on the Multi-PSI. In *Proceedings of Fourth International Conference on Logic Programming*, pages 257–275, University of Melbourne, MIT Press, 1987.

[Ichiyoshi *et al.* 88] N. Ichiyoshi, K. Rokusawa, K. Nakajima and Y. Inamura. A New External Reference Management and Distributed Unification for KL1. *New Generation Computing*, Ohmsha Ltd. 1990, pp.159–177.

[Ichiyoshi 89] N. Ichiyoshi. Parallel logic programming on the Multi-PSI. ICOT Technical Report TR-487, ICOT, 1989. (Presented at the Italian-Swedish-Japanese Workshop '90).

[Imai *et at.* 91] A. Imai, K. Hirata and K. Taki. PIM Architecture and Implementations. In *Proc. of Fourth Franco Japansese Symposium*, ICOT, Rennes, France, 1991.

[Imai *et al.* 91] A. Imai and E. Tick. Evaluation of Parallel Copying Garbage Collection on a Shared-Memory Multiprocessor. *ICOT Technical Report*, TR-650, 1991. (To appear in IEEE Transactions on Parallel and Distributed Systems)

[Inamura *et al.* 88] Y. Inamura, N. Ichiyoshi, K. Rokusawa and K. Nakajima. Optimization Techiniques Using the MRB and Their Evaluation on the Multi-PSI/V2. In *Proc. of the North American Conf. on Logic Programming*, 1989, pp. 907–921 (also *ICOT Technical Report*, TR-466, 1989).

[Kimura *et al.* 87] Y. Kimura and T. Chika-yama. An Abstract KL1 Machine and its Instruction Set. In *Proc. of Symposium on Logic Programming*, 1987, pp.468–477.

[Kumon *et al.* 92] K. Kumon, A. Asato, S. Arai, T. Shinogi, A. Hattori, H. Hatazawa and K. Hirano. Architecture and Implementation of PIM/p. In *Proc. of the Int. Conf. on Fifth Generation Computer Systems*, 1992.

[Masuda *et al.* 88] Y. Masuda, Y. Ishizuka, Y. Iwayama, K. taki and E. Sugino. Preliminary Evaluation of the Connection Network for the Multi-PSI System. In *Proc. Europian Conference on Artificial Intelligence 1988 (ECAI-88)*, August 1988.

[Matsumoto *et al.* 87] A. Matsumoto, T. Nakagawa, M. Sato, K. Nishida and A. Goto. Locally Parallel Cache Design Based on KL1 Memory Access Characteristics. ICOT Technical Report 327, 1987.

[Nakagawa *et al.* 89] T. Nakagawa, A. Goto and T. Chikayama. Slit-Check Feature to Speed Up Interprocessor Software Interruption Handling. In *IPSJ SIG Reports*, 89-ARC-77-3, 1989 (*In Japanese*).

[Nakagawa *et al.* 92] T. Nakagawa, N. Ido, T. Tarui, M. Asaie and M. Sugie. Hardware Implementation of Dynamic Load Balancing in the Parallel Inference Machine PIM/c. In *Proc. of the Int. Conf. on Fifth Generation Computer Systems*, 1992.

[Nakajima *et al.* 89] K. Nakajima, Y. Inamura, N. Ichiyoshi, K. Rokusawa and T. Chikayama. Distributed Implementation of KL1 on the Multi-PSI/V2. In *Proc. of the Sixth Int. Conf. on Logic Programming*, 1989, pages 436–451.

[Nakajima *et al.* 90] K. Nakajima and N. Ichiyoshi. Evaluation of Inter-processor Communication in the KL1 Implementation on the Multi-PSI. In *ICOT TR-531*, 1990.

[Nakajima 92] K. Nakajima. Distributed Implementation of KL1 on the Multi-PSI. In *Implementation of Distributed Prolog*, edited by P. Kacsuk and M. Wise, John Wiley & Sons, Ltd., 1992.

[Nakashima *et al.* 87] H. Nakashima and K. Nakajima. Hardware Architecture of the Sequential Inference Machine : PSI-II. In *Proceedings of 1987 Symposium on Logic Programming*, Sept. 1987, pp 104–113.

[Nakashima *et al.* 90] H. Nakashima, Y. Takeda, K. Nakajima, H. Andou and K. Furutani. A Pipelined Microprocessor for Logic Programming Languages. In *Proc. 1990 Intl. Conf. on Computer Design*, Sept. 1990, pp. 355-359.

[Nakashima *et al.* 92] H. Nakashima, K. Nakajima, S. Kondo, Y. Takeda, Y. Inamura, S. Onishi and K. Masuda. Architecture and Implementation of PIM/m. In *Proc. of the Int. Conf. on Fifth Generation Computer Systems*, 1992.

[Nishida *et al.* 90] K. Nishida, Y. Kimura, A. Matsumoto and A. Goto. Evaluation of MRB Garbage Collection on Parallel Logic Programming Architectures. In *Proc. of the Seventh Int. Conf. on Logic Programming*, 1990, pages 83–95.

[Nitta *et al.* 92] K. Nitta, K. Taki and N. Ichiyoshi. Experimental Parallel Inference Software. In *Proc. of the Int. Conf. on Fifth Generation Computer Systems*, 1992.

[Onishi *et al.* 90] S. Onishi, Y. Matsumoto, K. Nakajima and K.Taki. Evaluation of the KL1 Language System on the Multi-PSI. In *Proc. of Workshop on Parallel Implementation of Languages for Symbolic Computation*, July 30–31, 1990, Oregon, USA. Also ICOT TR-585.

[Rokusawa *et al.* 88] K. Rokusawa, N. Ichiyoshi, T. Chikayama and H. Nakashima. An Efficient Termination Detection and Abortion Algorithm for Distributed Processing Systems. In *Proc. of the 1988 Int. Conf. on Parallel Processing*, Vol. 1 Architecture, 1988,pp.18–22.

[Rokusawa *et al.* 92] K. Rokusawa and N. Ichiyoshi. A Scheme for State Change in a Distributed Environment Using Weighted Throw Counting. In *Proc. of Sixth Int. Parallel Processing Symposium*, IEEE, 1992.

[Sakai *et al.* 91] H. Sakai, A. Nakase, T. Takewaki and S. Asano. Applying Inter-cluster Shared Memory Architecture to a Parallel Inference Machine. In *Parallel and Distributed Computing in Engineering Systems*, edited by S. Tzafestas, P. Borne and L. Grandinetti, Elsevier Science Publishers, 1991.

[Sato *et al.* 87] M. Sato, A. Goto, et al. KL1 Execution Model for PIM Cluster with Shared Memory. In *Proceedings of the Fourth International Conference on Logic Programming*, pages 338–355, 1987.

[Sato *et al.* 88] M. Sato and A. Goto. Evaluation of the KL1 Parallel System on a Shared Memory Multiprocessor. In *Proc. of IFIP Working Conf. on Parallel Processing*, 1988, pp. 305–318.

[Sato *et al.* 92] M. Sato, K. Takeda and T. Ohara. Design of the Parallel Inference Machine PIM/i Processor. In *Trans. of IPSJ*, Vol.33, No.3, 1992, pp. 278–287 (*In Japanese*).

[Shinogi *et al.* 88] T. Shinogi, K. Kumon, A. Hattori, A. Goto, Y. Kimura and T. Chikayama. Macro-call Instruction for the Efficient KL1 Implementation on PIM. In *Proceedings of the International Conference on Fifth Generation Computing Systems 1988*, Tokyo, Japan, pages 953–961, 1988.

[Takagi *et al.* 91] T. Takagi and A. Nakase, Evaluation of VPIM: A Distributed KL1 Implementation – Focusing on Inter-cluster Operations –, In *IPSJ SIG Reports*, 91-ARC-89-27, 1991 (*In Japanese*).

[Takeda *et al.* 88] Y. Takeda, H. Nakashima, K. Masuda, T. Chikayama and K. Taki. A Load Balancing Mechanism for Large Scale Multiprocessor Systems and its Implementation. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, ICOT, Tokyo, 1988.

[Taki *et al.* 84] K. Taki, M. Yokota, A. Yamamoto, H. Nishikawa, S. Uchida, H. Nakashima and A. Mitsuishi. Hardware Design and Implementation of the Personal Sequential Inference Machine (PSI). In *Proc. of the Int. Conf. on Fifth Generation Computer Systems 1984*, pp.398–409, Tokyo, Nov. 1984.

[Taki 88] K. Taki. The Parallel Software Research and Development Tool: Multi-PSI System. In *Programming of Future Generation Computers*, K.Fuchi and M.Nivat (Editors), pages 411–426, Elsevier Science Publishers B.V., North Holland, 1988.

[Uchida *et al.* 88] S. Uchida, K. Taki, K. Nakajima, A. Goto and T. Chikayama. Research and Development of the Parallel Inference System in the Intermediate Stage of The FGCS Project. In *Proc. of the Int. Conf. on Fifth Generation Computer Systems 1988*, pp.16–36, Tokyo, Nov. 1988.

[Uchida 92] S. Uchida. Summary of the Parallel Inference Machine and its Basic Software. In *Proc. of the Int. Conf. on Fifth Generation Computer Systems*, 1992.

[Ueda 86] K. Ueda. Guarded Horn Clauses: A Parallel Logic Programming Language with the Concept of a Guard. ICOT Technical Report 208, 1986.

[Ueda *et al.* 90] K. Ueda and T. Chika-yama. Design of the Kernal Language for the Parallel Inference Machine. *The Computer Journal*, (33)6, 1990, pp.494–500.

[Wada *et al.* 90] K. Wada and N. Ichiyoshi. A study of mapping locally message exchanging algorithms on a loosely-coupled multiprocessor. ICOT Technical Report TR-587, 1990.

[Warren 83] D. H. D. Warren. An Abstract Prolog Instruction Set. Technical Note 309, Artificial Intelligence Center, SRI, 1983.

[Watson *et al.* 87] P. Watson and I. Watson. An Efficient Garbage Collection Scheme for Parallel Computer Architectures. In *Proc. of Parallel Architectures and Languages Europe*, LNCS 259, Vol.II, 1987, pp.432–443.