TM-0675

並列推論マシンPIM

由吉伸行

January, 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 PIM

市吉 伸行
(財) 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

第5世代プロジェクトは、1990年代の大規模知識処理システムのベースとなる第5世代コンピュータの開発を目指しているが、その特徴は、並列マシンと知識処理システムを結ぶ層として論理型の核言語を設定し、そこから下層および上層を設計して行くインサイド・アウトのアプローチを取っているところにある。このアプローチをとる大前提として、(1) 大規模知識処理システムの要求する計算能力に応えるためには並列実行が必須であること、(2) 複雑な知識処理システムを見通し良く設計するためには明晰な意味論を持つ論理プログラミングが有効であること、(3) 論理プログラムは並列処理に向いていること、という認識がある。

本稿では、第5世代コンピュータシステムのエンジンとなる並列推論マシン (ハードウェア + 言語処理系) とその上の並列オペレーティングシステム PIMOS について解説する。初めに並列論理型言語 KL1 の設計思想および言語仕様を述べ、次に PIM のハードウェアとその上の KL1 処理系の実現方式を概説し、PIMOS については、その機能・特徴と OS 保護の手法を紹介する。最後に、研究開発の現状と課題について触れる。

2 並列論理型言語 KL1

並列論理型言語 KL1 は、ストリーム AND 並列型言語 GHC [8] に制限を加えた Flat GHC (FGHC) をベースにして、OS を支援し、効率的並列実行を可能にするための実行管理、資源管理、優先順位および負荷分散指定の諸機能を追加したものである。

2.1 Flat GHC

Flat GHC (FGHC) は GHC に限られた組込み述語のみの出現をガード部に許すような制限を加えたものである。FGHC ではガードの実行がネストせず、AND/OR 木や変数環境の管理が不要なので、オーバヘッドの小さい処理系が実現できる。ガードにユーザ述語が書けないことにより表面的記述力は落ちるが、ガードにおけるテストをボディ・ゴールの組合わせに変換できるので、本質的記述力は低下しない。

2.2 荘園

荘園とは、実行制御と資源管理を行なうためのメタプログラミング機能である。末尾再帰呼出しされる KL1 のゴールが小粒度プロセスを表わすのに対し、荘園は1つのまとまった計算を定義し、荘園単位に実行の監視や制御が行なえるようになっている。荘園は組込み述語 execute の呼出しによって生成される。

```
execute(Goal, Control, Report, Mask)
```

ここで、 Goalは KL1 データとして表現された KL1 ゴール、 Controlは実行制御用のコントロールストリーム、 Reportは実行監視用のレポートストリーム、 Maskはこの荘園で捕捉する例外事象を指定するマスクである。生成された荘園の下でGoal(およびその子孫ゴール) が実行される。

荘園の下で新たに execute を呼出すことによりネストした荘園 (子荘園と言う) を生成することもできる。

コントロールストリームにstart、stop、abortなどのコマンドを流すことにより、実行の開始 (再開)、中断、放棄などができる[1]。レポートストリームには、荘園内で起きた特定の事象を報告するメッセージが流れる。荘園の終了 (全てのゴールおよび子荘園が終了したこと) の報告、例外事象の発生の報告などである。

[1] KL1 におけるストリームは、メッセージを要素とするリストとして表わされる。メッセージの送り手と受け手の間の同期は KL1 の同期機構によって自然に実現される。

荘園は資源管理の単位でもある。資源とは計算量の目安であり、計算時間やメモリ消費量と大体の比例関係にある。荘園があらかじめ指定された量の資源を消費してしまうと実行中断状態になり、資源の不足がレポートストリームに報告される。荘園の実行管理プログラムは、必要に応じて、コントロールストリーム・コマンドにより消費できる資源の上限を増やすことができる。資源管理機構により、プログラムの暴走を防いだり、荘園単位のフェアなスケジューリングが可能になっている。

KL1では、例外事象は、例外の種類によって決まるタグ値とマッチするマスク値を持っている一番内側の荘園のレポートストリームに次の形で報告される。

```
exception(Type, Goal, NewGoal)
```

ここで、 Typeは例外の種類を表わすデータ、 Goalは例外が発生したゴール、 NewGoalは未束縛変数である。荘園の実行管理プログラムは、上記メッセージを受け取るとTypeおよびGoalを解析し、 NewGoalを適切なゴールで具体化することによって実行を継続したり、場合によってはabortコマンドで単に実行を放棄することもできる。

例外事象には、ゴールの失敗(全ての節のガードが失敗した)、ボディ・ユニフィケーションの失敗、組込み述語例外(不正な入力引数、算術演算例外など)、raise述語の呼出しによるユーザ定義例外などがある。これらはユーザ定義例外を除けば全てFGHCプログラムにおける失敗である。

例外処理機能は、OSサービス提供やKL1の言語機能拡張に使うことができる。例えば、整数の3と構造型データ1+2のユニフィケーションの失敗時に再開ゴールとしてtrueを与えることで、(オーバヘッドはかなり大きいが)ユニフィケーションのセマンティクスを拡張できたりする。

2.3 優先度指定・負荷分散指定

優先順位および負荷分散は、ボディ・ゴールにプラグマと呼ばれる次のようなアノテーションを付けることによって指定する。

$$Goal@\texttt{priority}(Prio)$$

$$Goal@\texttt{processor}(Proc)$$

前者は$Goal$を優先度$Prio$で実行せよという指定であり、後者は$Goal$をプロセッサ$Proc$で実行せよという指定である。プラグマはプログラムの論理的意味を変えない。

荘園毎に実行優先度の上下限が決まっており、実行優先度は、荘園の優先度幅の中の割合で指定したり(絶対指定)、親ゴールの優先度と上下限の間の割合で指定したり(相対指定)できる。物理的には2^{17}程度の細かい優先度レベルを提供するが、応用プログラムの実行効率を上げるのにこの細かい優先度指定が役立つことが示されている。また、ユーザ・プログラムの暴走のためにOSプロセスがなかなかスケジュールされないといった状況は、 OSプロセスの優先度をユーザ・プログラムよりも高くすることで防げる。

MIMD型並列マシンの性能を十分に引き出すためには、実行負荷をよく分散しなければならない。しかし、単純な負荷の均等化はプロセッサ間通信オーバヘッドを大きくするし、 KL1プログラムが対象とする非定型な問題では計算負荷の事前の予想が難しい、という困難がある。種々の負荷分散手法を試すために、現在のところ上記のように低レベルの指定法を提供している。

3 並列推論マシンのハードウェア

KL1を実行する並列マシンがParallel Inference Machine(PIM)[3]である。現在、ハードウェア詳細仕様の少しずつ異なるいくつかのPIMを設計開発中だが、それらはKL1-B[6]と呼ばれる共通の中間言語をサポートする。ここでは、その内のひとつであるPIM/pを紹介する。

PIM/pのアーキテクチャは図1に示すように階層的である。すなわち、下位のレベルとして8台の要素プロセッサがメモリを共有するクラスタがあり、上位のレベルとしてクラスタたちが高速ネットワーク(ハイパキューブ)で結合され、全体として当初128プロセッサの並列マシンが構成されるが、将来的には1000台程度の規模まで拡張可能となっている。

メモリ共有型アーキテクチャは共有バスがボトルネックになるため、要素プロセッサ10数台程度が限界と見られ、大規模並列マシンを実現できない。また、大規模ネットワークにおいては、配線が可能であるためには、何らかの局所性を持つ(つまり、プロセッサ間の距離に遠近がある)ような結合方式とならざるを得ない。これが、上位構造が局所性を持つネットワーク型になっている理由である。

一方、プロセッサ間の通信はメッセージ通信よりも

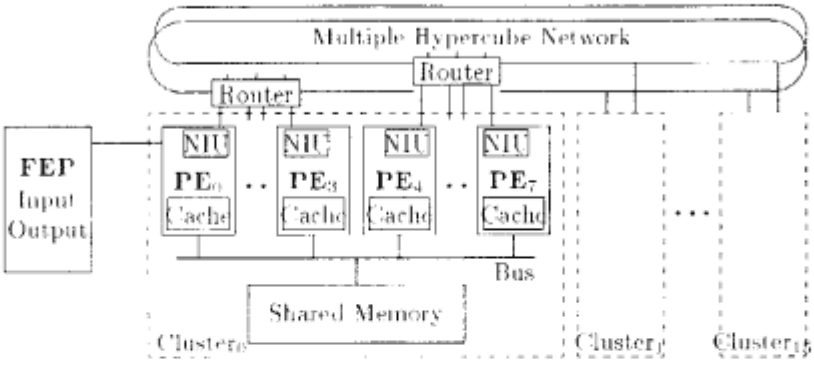

Figure 1: PIM/p のアーキテクチャ

共有メモリを介した通信の方がコストが安い。PIM の典型的アプリケーションは、多くのプロセスが部分を受け持って全体として1つの問題を解くような通信の多い型のものが予想されるので、台数性能比を上げるために通信コストはできるだけ抑えたい。下位レベルに密結合のクラスタを採用した理由は、相互間通信の多いプロセスたちを同一のクラスタに割当てることによって、通信オーバヘッドを低減できることにある。

プロセッサは並列記号処理向きに、タグ処理機能や後述の MRB の支援機能を持ったものを新規設計した。

クラスタ内では、分散したキャッシュ間の一貫性を保つこと、競合資源のロックを低いオーバヘッドで実現することが必要である。そのために、KL1 プログラムの実行特性に合わせたバス・プロトコルを設計した。例えば、プロデューサの書いたデータを別のプロセッサ上のコンシューマが読む場合にキャッシュ間データ移動が起こるが、KL1 ではプロデューサがそのデータに再びアクセスしないので、移動元のキャッシュのデータを無効化することができる。また、共有されていないことの分かるデータはバス・サイクルを使わずにロックできる。

ネットワーク接続装置 (NIU) には、バイト / ワード変換とパケット・バッファリング機能があり、汎用入出力インタフェースも接続されている。

なお、クラスタ内の6プロセッサにフロント・エンド・プロセッサ (FEP) やディスク装置などの入出力機器が直結可能となっている。PIM 上のアプリケーションの多くは計算バウンドだと思われるが、自然言語処理などで、大きな (PIM の大きな実メモリにも入り切らないような) データベースを必要とする可能性もあるので入出力バンド幅は大きめに取る予定である。

4 KL1 処理系

PIM の上で KL1 プログラムの実行を司るのが KL1 処理系である。KL1 処理系の開発において、次のような問題点があった。

- 分散したゴールの管理

  プロセッサをまたがって実行される述語の実現、実行制御、終了検出。

- 分散したデータの管理

  分散ユニフィケーションの実現、クラスタをまたがる参照ループ生成の防止。

- 効率のよいガーベジ・コレクタ

  KL1 は、破壊的書込みの許される言語やバックトラックによりメモリが自動的に開放される Prolog と比べてメモリ消費速度が速いので、トータルな実行性能を高く保つためには効率の良いガーベジ・コレクタ (GC) が必須。クラスタをまたがる GC も必要。

KL1 の基本実行モデルを記述した後、個々の問題点への解決策を紹介する。

4.1 基本実行モデル

我々の処理系は以下のような実行モデルに従っている。AND 関係で結ばれたゴールの集まりであるゴール・プールとプロセッサの集合があり、各プロセッサはゴール・プールからゴールを取り出して、リダクションを試みる。プロセッサの数が増えると、同時に多数のゴールのリダクションが行なえるので、処理系のスループットが上がることになる。

これが基本実行メカニズムであるが、実際には、KL1ではゴール・プールは荘園に対応し、ゴールの失敗は例外の形でゴール・プールの外に伝えられる。

KL1プログラムは、プログラム・モジュール単位にWAMライク[2]な中間言語KL1-Bにコンパイルされる。コンパイラはレジスタ・レベルの最適化やコミットできる節を高速に見つけるクローズ・インデキシングを行なっている。

4.2 荘園里親方式

クラスタをまたがった荘園においては、分散実行制御や分散終了検出が大きな問題となる。荘園には、実行可能状態や中断状態があり、もしその情報を集中化しているとゴールがスケジュールされる度にそのゴールの荘園を管理しているクラスタに状態を問い合わせねばならず、オーバヘッドが大きい。そこで我々の処理系では荘園の管理情報をクラスタ間に分散し、それを里親と呼んでいる[4]。

分散した荘園における実行終了の検出はいわゆる分散計算終了検出の問題であるが、重みつきレファレンス・カウントを応用したWTC方式[7]により終了検出に必要なメッセージ量を低く抑えることに成功した。

4.3 分散データ管理と分散ユニフィケーション

分散処理系で全体を1つのアドレス空間とすると、クラスタ毎の独立したGCが難しくなるという難点がある。クラスタ内GCによりデータのアドレスが変わると[3]、そのデータを参照している全てのポインタを更新しなければならないが、独立して動いている他のクラスタ内のポインタの更新は非常に難しい。

そこでKL1処理系では、クラスタ内外でアドレス系を分け、各クラスタはクラスタ外からクラスタ内への参照を管理する輸出表と、クラスタ内からクラスタ外への参照を管理する輸入表という2つのアドレス変換テーブルを管理することとした。

データがクラスタ間に分散していると、ユニフィケーションにおいてクラスタ外のデータを読んだり、書き込んだりする場合が出てくるが、そのためのreadプロトコルとunifyプロトコルを設計した。また、分散ユニフィケーションを無秩序に行なうと外部参照ポインタからなるループができる可能性があるので、クラスタ番号を比較して番号の大きいクラスタを指す外部参照は自クラスタの変数にバインドしてはいけない(その代わりにunifyメッセージを出す)という規則を設けた[4]。

4.4 ガーベジ・コレクション

KL1処理系におけるGCの重要性から、クラスタ内・クラスタ間の即時および一括型のGCが開発された。

MRB方式[1]は、参照ポインタにMultiple Reference Bit (MRB)と呼ばれる多重参照情報を持たせて、MRB=OFFのポインタが単一参照であることを保証するようにしたものである。KL1プログラムにおいては、大半のデータは単一参照されているが、MRB=OFFのポインタが消費されればそのデータの占めていたメモリ領域は回収できる。ポインタ側にMRBを持たせたために、回収時以外にはポインタの先にアクセスしなくて済むというメリットがある。これは、クラスタにおいてロックの回数を増やさずに済むことにつながる。MRB情報は、ベクタ要素の更新やストリーム・マージの最適化およびデータ輸出入管理の簡略化にも利用している。

分散処理系では、クラスタ外から参照されているデータの回収が問題となるが、重みつき参照カウント原理を用いたWEC方式[5]によって即時GCを実現した。

MRB方式とWEC方式で回収できないデータがあるため、クラスタ内並列GC[10]および一括型大域GCを検討中である。

5 並列オペレーティングシステムPIMOS

並列オペレーティングシステムPIMOS[11, 2]は、並列推論マシン向けの単一ユーザ、マルチタスクのOSである。

オペレーティング・システムの目的を一言で言えば、計算機の持つ物理的な機能をユーザに使い易い形で提供することである。計算機の物理的機能は大雑把に言えば、計算能力と入出力機能だが、それをユーザに使い易い形で提供するとは、一つには計算機資源を明快なモデルの形に抽象化することであり、もう一つにはユーザが、故意または過失により、そのモデルに

[2] WAMは、現在広く採用されているPrologの抽象マシン語である。

[3] データのアドレスを動かさないマーク・アンド・スウィープ型のGCは処理時間やフラグメンテーションの欠点がある。

[4] 実際には間接的輸出の関係でもう少し複雑な規則となっている[5]。

違反した時に、計算機資源および違反していない部分を保護することである。並列オペレーティングシステムPIMOSは次のような基本方針で設計された。

- KL1で記述された純粋な論理型OSであること。

  これには、OSを記述するレベルで既に計算機資源がユニフォームに抽象化されているというメリット(例えば、メモリ保護の問題が生じない)がある。KL1は外部と相互作用するプロセスの記述に向いており、PIMOSでは計算機の資源の多くをKL1プロセスとしてユーザに見せている。入出力デバイスはその典型である[5]。ユーザプログラムとOSはオブジェクトとメタの関係にあり、理想的には、OSはユーザプログラムを1レベル下のデータとして制御するべきだが、これは実行効率上の問題があり、現時点では実用的でない。そこで、前述のようにKL1ではFGHCにメタプログラミング機能を組込んでいる。

- 集中型単一OSであること。

  PIMOSはプロセサ毎に独立して動作するOSの集合体ではなく、全体として一体であるシステムを並列実行可能な部分を並列実行する。非定型並列処理においては物理プロセッサを意識しない方が、自然にプログラムが書けるのである。ただし、PIMが上位層で疎結合マシンであることを考慮して、強い結びつきのあるプロセスたちを同じクラスタにマッピングするなどの措置は講じている(PIMOSの論理的構造への影響はない)。

PIMOSの具体的機能としては、タスクの生成・監視・制御、入出力デバイスの管理、シェル機能、などがある。ユーザ・プログラムはソフト例外を上げる(raise)ことにより、PIMOSへの要求ストリームを獲得し、このストリームにメッセージを流すことによって必要なPIMOS資源を得るようになっている。全てのPIMOS資源はその資源特有のプロトコルを持ったストリームとしてユーザに提供される。

メタプログラミング機能を「理想的に」実現していないために、PIMOSではOSをユーザから保護する必要がある。下の例を見よう。ユーザプログラムがN文字を読むためにgetbというコマンドをファイルに送ったとする。

[5]これに対してPrologにおいて入出力機能がいかにPrologのセマンティクスと合わない汚い形で入っているかを思い起こされたい。

```
..., Req = [getb(N,Str)|NewReq], ...
```

ここで、Reqはファイルへの要求ストリームである。PIMOSのファイルハンドラ側にはgetbに対応して、次のような節がある。

```
file_handler([getb(N,Str)|NewReq],File) :-
    true |
    readFromFile(File,N,StrRead,NewFile),
    Str = StrRead,
    file_handler(NewReq,NewFile).
```

もしここで、文字列が読み込まれる前にユーザ・プログラムがStrに例えば整数の0をユニファイしてしまったとすると、ファイルハンドラで読み込まれた文字列StrReadとStrとのユニフィケーションが失敗してしまう。また、ユーザプログラムが文字数のNを具体化しないまま終了してしまうと、ファイルハンドラ側に終了しないゴールができてしまう。

PIMOSでは保護フィルタというインタフェース・ルーチンを導入することでこれを解決した。保護フィルタはユーザがPIMOS資源へのストリームを獲得する度に、ユーザプログラムとPIMOS資源の間に自動的に挿入されるフィルタ・プロセスであり、次の機能を持つ。

- ユーザプログラムからのメッセージでPIMOS側で読む部分については、それが具体化されるのを待ち、メッセージとして正しい形をしていることを確認した上で、PIMOS側に流す。

- ユーザプログラムからのメッセージでPIMOS側で書く部分については、新たな未束縛変数に置き換えてPIMOS側に渡し、PIMOS側で具体化した後に元の変数とユニファイする。

保護フィルタの導入によって、ユニフィケーションの失敗やデッドロックはユーザタスク内で起きることになり、PIMOSは安全である。

また、ユーザタスクの強制終了時にPIMOSへのストリームを閉じるバルブ機構が用意されている。

6 現状と課題

PIMとPIMOS、およびそれらを結ぶKL1について概説してきた。現在、KL1処理系がマルチPSI[6]に

[6]逐次型推論マシンPSIをメッシュ状のネットワークで結合した疎結合型並列マシン。マルチPSIの要素プロセッサはPIM/pのク

実装されており、PIMOS暫定版が稼働している。また、1台の小型化PSI上でマルチPSIをシミュレートするPseudoマルチPSIシステムも開発されている。両システムとも徐々にICOT外の研究機関にリリースして並列プログラミング研究に役立てて頂く予定である。KL1処理系がPIM/pに実装されるのは来年度半ば頃となろう。

現在のPIMOSでは、入出力機能が全てFEPにおいて実現されており、コンパイラなどの開発環境もPSI上のクロスシステムに置かれているが、それらの機能をPIMOS本体に移行していくつもりである。

本来OSが管理すべき計算機資源の内、クラスタ毎のメモリ使用量などは、現在KL1の下に隠れてしまっており、きれいな形でそれらを言語レベルに持ち上げて管理の対象にすることは、論理OSとしてのPIMOSの今後の課題のひとつであろう。

負荷分散の研究はこれからの分野だが、研究の進展にともない将来的には、実験的な動的負荷分散機能をPIMシステムに組込んで行きたい。

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] T. Chikayama, H. Sato, and T. Miyazaki. Overview of the parallel inference machine operating system (PIMOS). In *Proceedings of the International Conference on Fifth Generation Computer Systems 1988*, pages 230–251.

[3] A. Goto, M. Sato, K. Nakajima, K. Taki, and A. Matsumoto. Overview of the parallel inference machine (PIM) architecture. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1988*, pages 208–229.

[4] N. Ichiyoshi, T. Miyazaki, and K. Taki. A distributed implementation of Flat GHC on the Multi-PSI. In *Proceedings of the Fourth International Conference on Logic Programming*, pages 257–275, 1987.

[5] N. Ichiyoshi, K. Rokusawa, K. Nakajima, and Y. Inamura. A new external reference management and distributed unification for KL1. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1988*, pages 904–913.

[6] 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.

[7] K. Rokusawa, N. Ichiyoshi, T. Chikayama, and H. Nakashima. An efficient termination detection and abortion algorithm for distributed processing systems. In *Proceedings of the 1988 International Conference on Parallel Processing, Vol. I Architecture*, pages 18–22, 1988.

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

[9] D.H.D. Warren. An abstract Prolog instruction set. Technical Note 309, SRI International, 1983.

[10] 佐藤正俊, 後藤厚宏. KL1並列処理系の評価 - メモリ消費特性とGC. 並列処理シンポジウムJSPP'88論文集.

[11] 宮崎敏彦. 並列論理型言語KL1の実現方式と並列OSの記述. 電子情報通信学会論文誌 D Vol. J71-D, No. 8, pages 1423–1432, 1988.

著者紹介

市吉 伸行

1979年東京大学理学部情報科学課卒業。1981年同大学院修士課程修了。1982年(株)三菱総合研究所入社。1984年より1年間米国パテル研究所にてAIを研修。1987年より(財)新世代コンピュータ技術開発機構へ出向。論理型言語処理系、並列プログラミングの研究開発に従事。情報処理学会、AAAI会員。

クラスタに相当するとみなてよい。FEPはシステム全体で最大4台まで接続可能となっている。