ICOT Technical Memorandom: TM-1203

TM-1203

並列論理型言語 KL1 による オペレーティングシステム PIMOS の実現について

藤瀬　哲朗、屋代　寛、近山　隆

August, 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

並列論理型言語 KL1 による
オペレーティングシステム PIMOS の実現について

藤瀬 哲朗　屋代 寛　近山 隆
(財) 新世代コンピュータ技術開発機構

PIMOS (Parallel Inference Machine Operating System) は, 第 5 世代コンピュータプロジェクトの一環として開発された並列推論マシン PIM のオペレーティングシステムである. 我々の開発では, まず, 核言語として並列論理型プログラミング言語 KL1 を設定し, ハードウェアとソフトウェアのインタフェースを固定するアプローチをとった. 一方, PIMOS も KL1 で記述されており, 資源管理の論理的な記述を可能としている. このことにより, 開発環境の違いに影響されることなく, OS の開発が可能となった. 本稿では, PIMOS の実現方式と数種類のマシン上に実装した経験について述べる.

Implementation of PIMOS
in a Concurrent Logic Programming Language KL1

Tetsuro FUJISE Hiroshi YASHIRO Takashi CHIKAYAMA
Institute for New Generation Computer Technology
Mita Kokusai Bldg. 21F, 4-28, Mita 1-chome, Minato-ku, Tokyo 108 JAPAN

The parallel inference machine operating system PIMOS is an operating system for the parallel inference machines developed in the Japanese Fifth Generation Computer Systems project. Bases on a specification of a concurrent logic programming language KL1, design of the hardware and the language implementation started downward, and writing the PIMOS and parallel software started upward. In this approach, PIMOS has been developed without any problem in differences of software development environment for each system. This paper gives an implementation of PIMOS in the KL1 language.

1 はじめに

第5世代コンピュータプロジェクトの一環としてICOTで開発・改良中の並列推論マシンPIM (Parallel Inference Machine) [5] は，並列論理型プログラミング言語KL1 [6] をソフトウェアとハードウェアのインタフェースとして持つ，高級言語専用計算機である．この PIM のオペレーティング・システムが PIMOS (Parallel Inference Machine Operating System) [7] である．

並列論理型プログラミング言語 KL1 は，Prologのようなある特定のアルゴリズムの記述向きのプログラミング言語とは異なり，汎用の並列記号処理言語である．KL1 は，実行順序や同期を意識せずに書けるデータフロー並列性を持つ高水準言語であり，Lisp などと同じレベルの自動記憶域管理機能を持つ．

PIM は，並列論理型プログラミング言語 KL1 でシステム全体が動作するため，そのための最適化がいろいろなされてはいるが，基本的には一般の疎結合型並列計算機[1]である．一般的なマシンとの違いをまとめると以下の通りである．

- 実験機である．第5世代計算機システムのプロトタイプとなるべき計算機であって，すぐに商用に使うようなものではない．
- KL1 の専用計算機である．
- 疎結合型並列計算機である．もちろん常に通信のコストを気にする必要がある．

本稿では，以上のような状況の下，開発された PIMOS とその記述言語 KL1 との関係について述べる．

2 OS と言語処理系

2.1 OS と言語処理系との関係

PIM には，数種類のハードウェアがあり，様々な並列プログラムの性能評価およびチューンを各ハードウェア上，つまり共通 OS である PIMOS 上で行なうことでハードウェア自身の評価を行なっている．

PIMOS は各 PIM の共通 OS であり，共通プログラミング言語である KL1 で記述されている．KL1 プログラムとしての PIMOS を動作させるためには，一般には，それを動作させるための OS 部分を各種ハードウェア上に実現する必要がある．しかもそれらは並列動作機能の primitive である必要がある．並列動作機能の primitive が何であるかは，なかなか良くわからないところである．

OS において，システム内のジョブの動作状況を把握し，必要に応じて強制終了させたりすることを考える．実際に動作しているジョブをデータと見立て，それを制御する OS をプログラムとすると，OS は相手が並列であろうと逐次であろうとメタレベル制御機能を必要とすることが言える．

メタレベル制御機能は，OS だけではなく，複雑な計算を行なう場合にも必要とする場合がある．並列計算の場合には，メタレベル制御の必要性が非常に顕著に現れてくる．計算の負荷分散などが良い例であろう．

我々はこの並列動作のための primitive，すなわちメタレベル制御の多くが OS だけではなく，並列応用プログラム実行のための primitive であると考える立場をとった．

この立場により OS と言語処理系の関係が逐次処理を原則とするシステムと異なったものとなっている．これが PIMOS 開発における状況の特徴的なところである．

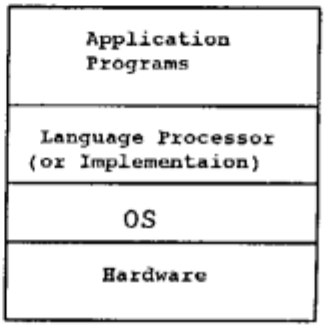

図 1: 従来の階層

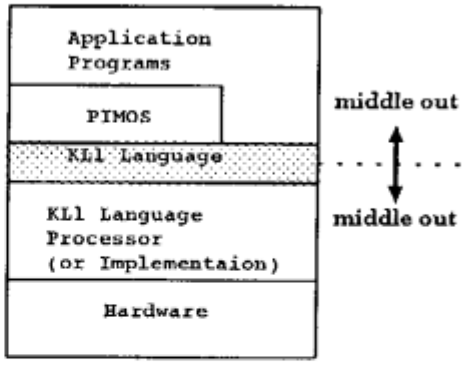

図 2: Middle out アプローチ

一般には，OS と言語処理系の関係は図 1 とする場合が多い．我々のアプローチでは，図 2 にあるように言語処理系上に OS を実現した．オブジェクト指向モデルやデータフローモデルなど，これまでに提案されてきた並列処理モデルの多くが記述できるプログラミング言語 GHC から派生した KL1 を中心に据え，

[1] ひとつのノード内で複数のプロセッサがメモリを共有している場合もある．

一方において KL1 を効率的に動作する言語処理系およびハードウェアを実現し，もう一方においてユーザが KL1 で記述したプログラムの実行環境を提供する OS を実現した．いわゆるプログラミング言語を中心に据えた Middle out アプローチ [1], [2] と呼ばれる方法である．

2.2 Middle out アプローチ

Middle out アプローチを採る場合，並列動作機能の primitive とは，並列論理型プログラミング言語 GHC (実際には KL1) の言語処理系が必要とする機能となる．

実際には，小規模の実験的ハードウェアを組立てて KL1 プログラムの効率的実行を確認しつつ，並列ハードウェア，並列言語処理系[2]，並列 OS の役割分担が決められた．最初から独立に役割が設定されたわけではない．

Middle out アプローチにより，通常の OS の機能を KL1 言語処理系と並列 OS である PIMOS の双方で分担し合う構造となった．PIMOS は並列処理におけるメタな機能を使う側，KL1 言語処理系はメタな機能を提供する側となった．

本アプローチのメリットは，次の通りであった．

- ハードの主たる実現目標を KL1 プログラムの効率的実行に絞ることができる．
- PIMOS を KL1 で記述できることから，様々なシステムの管理モデルを並列計算モデルのまま実現できる．

後者は，一見大した話ではないと思うかもしれないが，今回の開発では絶大な効果を発揮した．これについては後述する．

もちろん次のデメリットも存在した．

- KL1 言語処理系の実現のための負担が大きい．

KL1 言語は，とりわけ簡単さが大きな特徴であるが，それでも各機能を積み上げるとそれなりの量になる．通常 OS がもつメモリ等の計算資源管理も言語処理系が担うことになるし，高級記号処理言語には必須である自動記憶域管理機能もこの処理系で実現しなくてはいけない．実現方式にもよるが，それなりに大変なことには違いない．

KL1 言語処理系の実現は，KL1 プログラムの実行効率を向上させることが目標であるので，負荷がかかるのは当然であるかもしれない．

[2]現存の PIM には RISC タイプ，CISC タイプ双方が存在し，また各ノードの結合方式もそれぞれ異なる．言語処理系もアーキテクチャに合ったそれぞれの実現方式が採用されている．

3 資源管理モデル

並列 OS である PIMOS，特に論理デバイスやタスクを管理する資源管理部が背負った要件は，次の通りである．

- ユーザプログラムの並列性を損なってはいけない．

  PIM は，並列知識情報処理プログラムを走らせるための実験機である．PIMOS のユーザプログラムに対する影響度を低くしたい．OS のサービスも極力ユーザプログラムの並列性を損なわないようにする．

- 並列 OS が動作中にユーザプログラムが停止している保証はない．

  逐次型のマシンではないので，PIMOS 動作中にユーザプログラムが停止している保証はない．停止させるには，ユーザプログラムが動作する全ノード中のユーザプログラム部分を停止する必要がある．ノードが増えてくるとその手間はばかにならない．もちろん意図的に実行中のユーザプログラムを停止させる場合は除く．

- そのような複数のプログラム群の実行を管理しなくてはいけない．

  一つのユーザプログラムの実行管理について上述の要件があるのに加えて，それらの集団を管理しなくてはならない．一つのユーザプログラムの実行管理が他のユーザプログラムの実行管理をできるだけ邪魔しないようにする必要がある．

- ユーザプログラム群の実行を管理することで，ユーザプログラム群の実行を妨げてはいけない．

  ユーザプログラム群を管理するがために，全体のパフォーマンスを落すことは避けたい．

PIMOS の資源管理部の目標はユーザプログラムを邪魔せず，かつ保護を中心とした管理機能を実現することとなった．

以上のような要件を満たすために，局所資源管理モデル [3] を導入した．

3.1 資源木

局所資源管理モデルは，タスク[3]・資源[4]を局所的に管理するためのものである．このモデルを木状に結合することで，PIMOS 上の全タスク・資源の管理構造を実現している．PIMOS では，この構造のことを資源木と呼んでいる．

[3]プログラムの実行単位のことである．

[4]仮想化された入出力のような OS が提供する資源のことである．

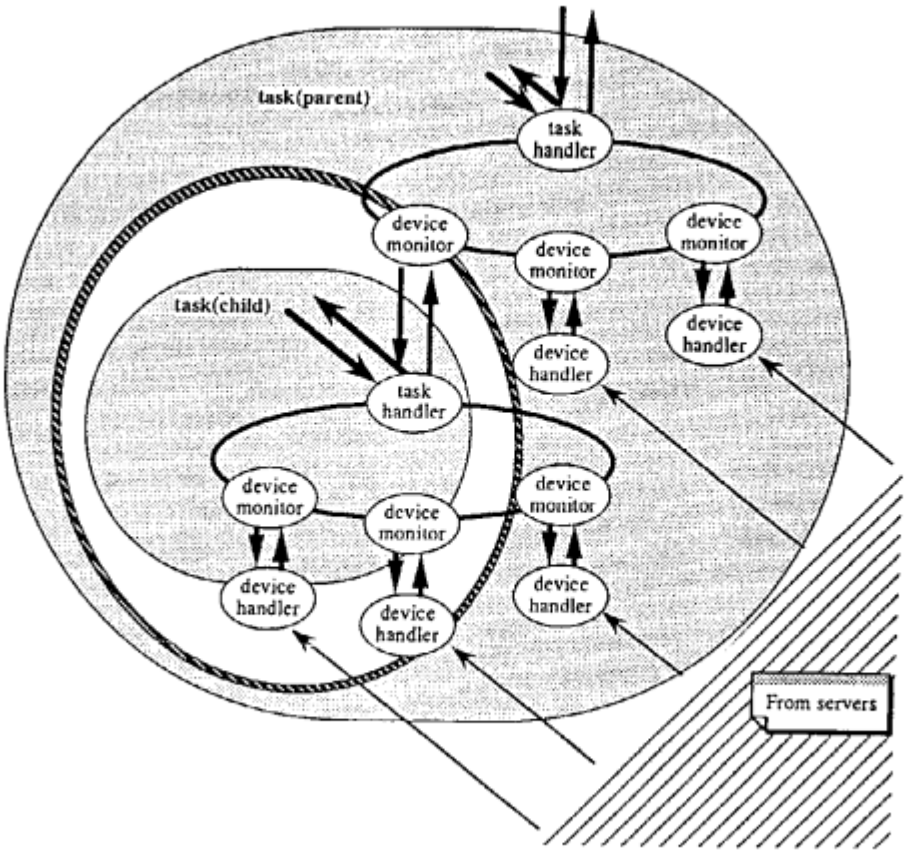

図 3: 資源木

以下このモデルについて説明する.

図 3 に資源木の概念図を示す. 図中の部品を簡単に説明する.

- タスクハンドラ (task handler)

  タスクを管理する OS 機構である.

- デバイスハンドラ (device handler)

  タスク以外の OS 定義資源を管理する OS 機構である.

- デバイスモニタ (device monitor)

  タスクハンドラがタスクが所有するデバイスのデバイスハンドラを管理するための補助機構である.

これらの機構がストリームで結合し, 局所資源管理モデルを実現している.

3.2 局所資源管理モデル

前述したように並列性を保証しつつ他の管理の邪魔をしないために PIMOS では管理を局所化することにした (図 4). 局所管理しつつ階層的に分散しているので, ノード数が増えた場合は効果的である.

モデル化の効果を示すために, 図 5 を利用して簡単にこのモデルを説明する. ここでは資源とはタスクと考えて支障ない. 図 3, 図 5 にあるようにハンドラの (7) とモニタの (1) が接続され, (2) が次の資源のモニタの (1) に接続される. これを繰り返し, 最後に (8) に接続される. 局所資源管理モデル (図 5) における典型的な通信パターンを以下に示す.

- (1) → (2)

  ユーザプログラムから対象資源の利用

- (1) → (3)

  親タスクから対象資源への制御 / 問い合わせ

- (1) → (3) → (7)

  親資源から対象資源の子資源への制御 / 問い合わせ

- (8)

  子資源から対象資源への問い合わせ

- (8) → (4) → (2)

  子資源から対象資源の親資源へ問い合わせ

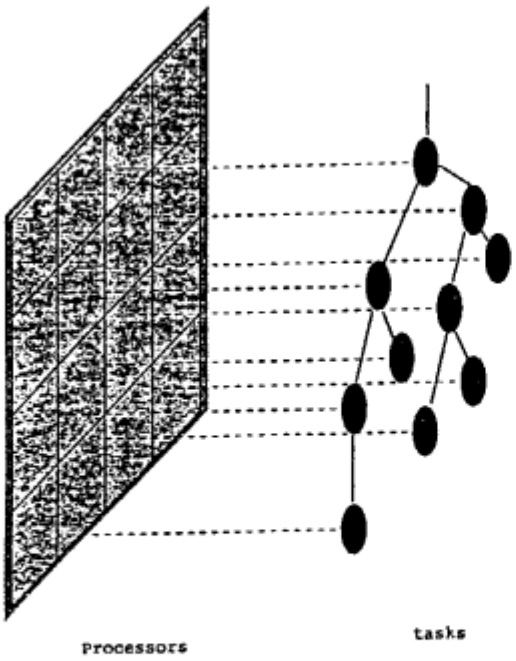

図 4: 資源の階層的管理

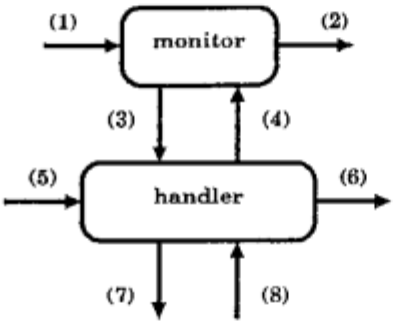

図 5: 局所資源管理モデル

- (5) → (7)
  ユーザプログラムから対象資源の子資源への制御 / 問い合わせ

このモデルでもっとも注意を要するのは，終了処理部分である．例えば，子資源の利用を終了したと同時にユーザタスクが強制終了してしまった場合，子資源の終了処理とユーザタスクの終了処理の競争が起こる．局所資源管理モデルでであるため，処理自身の逐次性が保証されない．単純に管理すると階層下位と階層上位の資源管理が困難な状況になる．そのために局所資源管理モデルに次の基本的な規則を適用した．

規則：

1. 資源は自身を終了するために，親資源から終了許可を得る必要がある．親資源からのストリームを親資源が切り離した時に初めて終了することができる．
2. 資源は自身を終了するためには，子資源の終了を確認する必要がある．
3. 資源は子資源から終了許可要求があった後，子資源へ終了許可を送り，二度と子資源へのストリームへメッセージを送ってはならない．但し，子資源からのメッセージは受信しなくてはいけない．最後に子資源が資源へのストリームを閉じた，すなわち子資源からメッセージが来る可能性が無いことが保証された時，子資源へのストリームを閉じる．
4. 資源は自身の強制終了時，子資源に強制終了命令を別途発するが，資源の終了処理はあくまで上述の処理に従う．

この規則の正当性は，ストリームを流れるメッセージ間に半順序を導入すれば証明できるがここでは割愛する．

この規則から簡単に次の終了手順が導かれる．

終了手順：

1. (5) が閉じられる．強制終了時には子資源に強制終了命令を発する．
2. 子資源の終了を確認するため，(7) に終了確認通知を送る．この通知を受け取った子資源のモニタは，今後対応する子資源にメッセージを送らないこととする．
3. (8) からすべての子資源に終了通知が送られたことを確認した後，(4) に終了許可を要求する．
4. モニタは (4) から終了許可要求を受け取る．そしてもしモニタが (1) からすでに終了確認通知を受け取っている時，子資源に終了許可通知を送る．受け取っていない時には (1) から終了確認通知を受け取った時，子資源に終了通知を送る．
5. (3) から終了許可通知を受け取る．初めて終了できる準備が整ったので，以下の最終処置を行なう．
   (a) (7) の通信路を閉じる．
   (b) 次に (8) の通信路が閉じられたら，(4) の通信路を閉じる．
   (c) モニタが (4) の通信路が閉じると，モニタは (1) と (2) を結合し，(3) の通信路を閉じる．そしてモニタは消滅する．
   (d) (3) の通信路が閉じられたら，最後に (6) 閉じて資源は消滅する．

各ストリームに対してこの手順を遵守して扱うことで，この処理は，必ず終了する．

4 開発経験

前節で述べたように，PIMOS の要件を満たすために我々は局所資源管理モデルを各資源に適用し，かつそれらに処理規則を加えてストリームで結合することで，PIMOS 上の全資源管理機構を実現した．

我々は Middle out アプローチを採ったため，PIMOS は，このモデルを並列論理型プログラミング言語 KL1 でそのまま実現することとなった．KL1 言語のもつストリーム AND 並列といった計算処理のモデルにより非常に簡単に実現できてしまった．

4.1 実行環境について

Middle out アプローチを採用したことが，最も効果的であったのが，開発環境の変化への適応である．

開発環境の遷移順に従って説明する．

1. Unix 上の疑似 KL1 言語処理系上での開発

   PIMOS 開発用の KL1 処理系 PDSS (PIMOS Development Support System) で，まず PIMOS の開発は始められた．この処理系は Unix 上の疑似並列処理系であるので，並列動作する訳ではない．

   PDSS 上では，まず上述の資源管理モデルの検証が行なわれた．実際は，規則がもっと曖昧な状況下で，モデルを検証した段階がまずあった．疑似並列処理系であっても，KL1 プログラムの動作を追うと非常に面白い現象に出合った．優先度を全く導入しない状況で，ユーザプログラムと資源管理プログラムの実行順を保証しないこととした．すると予期しない順でプログラム群が動作をし，モデルの論理的無矛盾性が必要であることを身をもって実感した．

   検証を繰り返した後，モデルが決定した．モデルを基に PIMOS の資源管理部を PDSS 上で開発した．実験と数学的証明を伴っており，なおかつほとんどモデルのまま記述できる言語で管理機構が開発できたので，この段階は非常にスムーズにすすんだ．率直に言えば，簡単に終了してしまった．

   デバイスとの結合については，この段階では疑似デバイスとしか行なわれなかった．PIMOS のデバイスは，ほとんどストリーム・インタフェースで実現されているため，動作確認は非常に簡単であった．

2. Pseudo Multi-PSI 上での開発

   Pseudo Multi-PSI は並列実験機である Multi-PSI とほぼ同様な KL1 言語処理系をもった PSI 上のシステムである．

   本段階の目的は，以下の通りであった．

   - 実際の入出力デバイスとの結合
   - KL1 言語処理系のデバッグ
   - 実機への移植に向けた動作確認

   動作確認のためのツールは，Multi-PSI のハード的サポートを行なう CSP (コンソール・プロセッサ) のもつ機能である．限定トレースといった KL1 言語のレベルでのステップ実行をサポートしていた．もっぱらこのツールを使えれば，動作内容やデータの変化を追うことが可能であった．

   まだレジデントコンパイラが無かったため，クロスシステムでコンパイル，リンクを行ない，できたロードモジュールを IPL によりハードに張り付け実行するといった具合であった．

   入出力デバイスとの結合は，非常にスムーズにいった．ストリーム・インタフェースの効果である．

   KL1 言語処理系のデバッグは，さすがに時間がかかった．ある程度まとまった規模の初めてのプログラムであったためである．

   KL1 言語処理系の仕様等の見直しを進めながら，実機への移植に向けた動作確認を行なった．

   本段階の特徴は，PIMOS 側でほとんどバグがでなかったことである．middle out アプローチのメリットがここで特徴付けられた．

3. Multi-PSI 上での開発

   この段階の目的は次の通り．

   - PIMOS の並列動作の確認

   並列実験機での初めての動作確認である．

   動作環境は Pseudo Multi-PSI と同様であった．

   KL1 言語処理系の並列動作に関するデバッグを行ない，PIMOS の動作確認を行なった．初めて並列で動作する環境での実行であったが，驚いたことにほとんどバグが発生しなかった．同期バグのように如何にも起こると期待できるものが，ほとんど無かったのである．

4. Multi-PSI 上での拡充

   並列知識情報処理技術の研究のためには，たとえ実験機であっても開発環境は必要である．

   PIM が完成するまでの間，レジデントのコンパイル環境，デバッギングツール，チューニングツールなどが次々に完成し，ユーザプログラムの拡充に貢献することとなった．

5. PIM への移植

上述の流れから容易に想像できると思う. Multi-PSI とオブジェクト互換である PIM/m はもちろんスムーズであった. 他の PIM への移植は, PIM/s といった密結合部分を含む並列計算機上のシミュレータから実機に至るまで, 効率的な KL1 処理系を完成させることが移植作業であった. PIMOS のデバッギング作業は, ほとんど行なう必要はなかった. 処理系の開発が適当な段階まで来ると, 開発環境付きで応用プログラム群が動作し始めているといった具合である.

経験談であるため結論付けることはできないが, Middle out アプローチによる効果として, 次のことが挙げられると思う.

- モデルの構築から PIMOS の本体部分の開発に至るまで KL1 といった高級言語を基本としていたため, 各開発環境の変化に影響されることなく論理的に洗練することが可能であった.

4.2 KL1 言語の機能について

KL1 言語もつ機能の中で PIMOS が主に利用したものは次の通りである.

- 暗黙のデータ同期機構

  上述のようなイベント駆動型的規則を宣言型プログラミング言語で記述するとき, 暗黙のデータ同期機構があるために局所的な処理記述の集合で実現した. 本機能があることが, 各段階で対した同期バグが発生せずに PIMOS が実現できた理由のひとつであろう.

- 多重待ち機能

  採用したモデルは, 非常に多くの多重同期処理を必要とする. 通常は非常にいやらしいところだが, Committed-Choice 言語である KL1 には言語自体にこの機能があるため, 非常に簡単に記述することができた. 逆にこの機能があるため, 実現が容易であることを前提として, 抽象度が高い段階のモデル構築の議論が可能であった.

- 多重読み出し機能

  強制終了の通知は, 階層的に資源を管理している以上, この通知の階層的伝達の遅延が非常に問題になる. しかしながら, KL1 における多重読み出し機能と多重待ち機能を利用することで, 非常事態発生をある変数に値を設定するだけで特急伝達が可能であった.

  この処理には, 優先度付き多重待ち機能をもちいることにより, 非常事態の優先的伝達を行なった.

- 優先度指定機能

  もちろん優先度付ける処理を必要とする場合があり, 利用した. KL1 の場合, 優先度幅が適当な大きさあり指定が簡単なため, ユーザタスクへの影響度が少ないものである.

- 実行ノード指定機能

  PIMOS 資源管理部はユーザタスクの並列性をできるだけ損なわないよう, 資源利用の要求を行なったユーザプログラムのいるノードで局所管理コードが実行される.

- 荘園機能

  GHC を拡張した機能で, KL1 独特のものである. ユーザタスクの暴走をメタプログラムで停止させるためには, 必須のものであった.

荘園機能は別として, 以上の機能が言語要素にあるために非常に楽に資源管理を実現した感がある.

5 おわりに

開発環境が揃ってきたため, 様々な問題にチャレンジすることも可能になってきた. 特に並列処理の経験を持たない研究者が, 容易に並列プログラムを組み, チューンできる不思議な情景に出合えるようになった. 海外からも Telnet を利用して, PIMOS にアクセスすることもできるようになった. 1989 年にユーザにリリースして以来, 100 人強 (実際にはもっと多いのかもしれない) の方々に利用して頂き, 様々な応用プログラムが開発されてきた.

またプロセッサの台数も増えることで, 今までにできなかった計算が少しずつ可能になってくる実感も湧いてくるようになった.

以上のような開発経過を振り返り, 環境が整ってきたことで感ずる次の問題を挙げておく.

- 言語処理系の機能を膨らませすぎたこと

仕様を高レベルで大きくしたため言語処理系の開発作業が大変である部分もあった. 逆に PIMOS の方があっけないほど簡単に済んでしまった箇所があった. 別の見方をすると PIMOS がある程度以下のレベルを制御できないがために不便となった部分が目立つようになった.

実験機からさらに発展させる段階でいくつかの機能については, もう一度洗い直す必要があるかもしれない.

実現性が見えなかった OS をより容易に実現するため, 実現イメージがわかり易い KL1 言語処理系に負荷をかけて来たが, もうそろそろ OS 側に機能を寄せるべきかもしれない.

参考文献

[1] 近山: 並列推論マシンのオペレーティングシステム, ICOT ジャーナル No.29, pp.2-12 (1992).

[2] Chikayama,T. : Operating System PIMOS and Kernel Language KL1, *Proc.of FGCS'92*, pp.73-88 (1992).

[3] 藤瀬 : PIMOS の資源管理におけるストリームの扱いについて, *Proc.of KL1 Programming Workshop'90*, pp.57-61 (1990).

[4] 市吉: 並列論理型言語 KL1, 数理科学 第 27 巻 7 号, pp.11-18 (1989).

[5] Taki,K. : Parallel Inference Machine PIM, *Proc.of FGCS'92*, pp.50-72 (1992).

[6] Ueda, K. *et al.*: Design of the Kernel Language for the Parallel Inference Machine, The Computer Journal, vol.33, No.6, pp.494-500 (1990).

[7] Yashiro,H. *et al.*: Resource Management of PIMOS, *Proc.of FGCS'92*, pp.269-277 (1992).