PIMOSの設計方針

佐藤裕幸[1]、越村三幸[1]、近山隆[1]、藤瀬哲朗[2]、松尾正浩[2]、和田久美子[3]
1: (財)新世代コンピュータ技術開発機構 2: (株)三菱総合研究所 3: 沖電気工業(株)

1 はじめに

第5世代コンピュータ・プロジェクトでは、知識情報処理を行うに当たって必要とされる高度な計算能力を提供するために、PIM[1]やマルチPSI [2]などの並列推論マシンの開発を行っている。これらの並列推論マシンの能力を最大限に引き出すためには、高並列に動作するプログラムを効率的に制御するオペレーティング・システム(OS)が必要不可欠である。

PIMOS(Parallel Inference Machine Operating System) [3, 4, 5, 6] は、この目的のために設計された並列推論マシン用のOSである。PIMOSの主な仕事は、ユーザ・プログラムが消費するさまざまな資源を管理することである。これらの資源を並列マシン上で効率良く管理するには、従来のOSのように集中的に管理するのではなく、それぞれの資源を分散して並列に管理しなければならない。

本論文では、この並列性を引き出すためのPIMOSの設計方針について報告する。

2 プロセス

ユーザ・プログラムが消費するCPU時間、メモリ、入出力装置などの資源を、それぞれのプログラムが並列に動作する環境の下で管理するのは、容易なことではない。例えば、OSがあるデータに対する処理を行っている最中に、そのデータの状態をユーザ・プログラムによって変更されることを防ぎたい場合がよくある。そのような場合、従来の逐次計算機上では、「OSがそのデータを処理している最中は、ユーザ・プログラムを動作させない」という実行順序によってデータの変更を防ぐことができた。しかし、PIMOSが動作するような並列環境では、並列性を犠牲にせずに、そのような実行順序を制御することはできない。つまり、OSがそのデータを処理している最中に、全てのユーザ・プログラムの実行を停止させることになり、これは明らかに並列性を犠牲にすることになる。従って、従来の逐次計算機と同様の方式を採用することはできない。

そこでPIMOSでは、「プロセス」[7]と呼ばれるプログラミング・スタイルを導入することにより、この問題を解決している。このプロセスは、管理しているデータと通信用の変数(ストリーム)を保持している。プロセスの外からこのデータにアクセスする場合は、このストリームにメッセージを送る(通信用の変数をメッセージ列に具体化する)という形でしか行えない。従って、そのストリームにアクセスすることができないプログラムは、このデータにアクセスすることはできない。つまり、データへのアクセス・パスを1つにすることで、複数のプログラムからの競合を防いでいる。

PIMOSが管理する全ての資源は、このプロセスを用いて、ストリームを介して通信することによって管理されている。つまり、このシステムでの資源へのアクセスは、従来のシステムのスーパバイザ・コールのような、資源を集中管理しているモジュールへの通信によって行われるのではなく、それぞれの資源を管理しているプロセスへストリームを介して通信することによって行われる。従って、PIMOSでは、これらのストリームを管理することが、資源を管理することに相当している。以下では、このプロセスによる資源の管理方式について説明する。

3 荘園(資源管理, 異常処理)機能

PIMOSの記述言語である核言語第1版(KL1)はFGHC[8]を基本にしており、全てのゴールは同じレベルのAND関係になっているため、プログラムが消費する資源を制御/管理し、異常事態を処理することができない。そこで、KL1では、このようなメタ・プログラミング機能を、以下のような組込述語によって、提供している。

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

ここで、各引数は以下のようなものである。

Goal: 実行されるゴール列であり、このゴールの実行中に消費される資源(CPU時間やメモリ量など)を管理し、実行中の異常を報告する機能が働く。この資源管理/異常処理の単位を「荘園」と呼ぶ。この荘園は、任意にネストすることができる。

Control: 実行の制御(開始, 中断, 放棄など)及び資源を割り当てるためのストリームである。

Report: 荘園の実行状態(実行の終了, 資源の超過使用, 例外事象など)が報告されるストリームである。

Tag: 荘園がネストしている場合に、例外事象を報告する荘園を識別するためのタグである。

このように荘園は、制御ストリームと報告ストリームをインタフェースとした、言語レベルで提供されているプロセスと考えることができる。

Design Principle of the PIMOS

Hiroyuki SATO[1], Miyuki KOSHIMURA[1], Takashi CHIKAYAMA[1], Tetsuro FUJISE[2], Masahiro MATSUO[2], Kumiko WADA[3]
1: Institute for New Generation Computer Technology 2: Mitsubishi Research Institute 3: Oki Electric Industry Co.,Ltd.

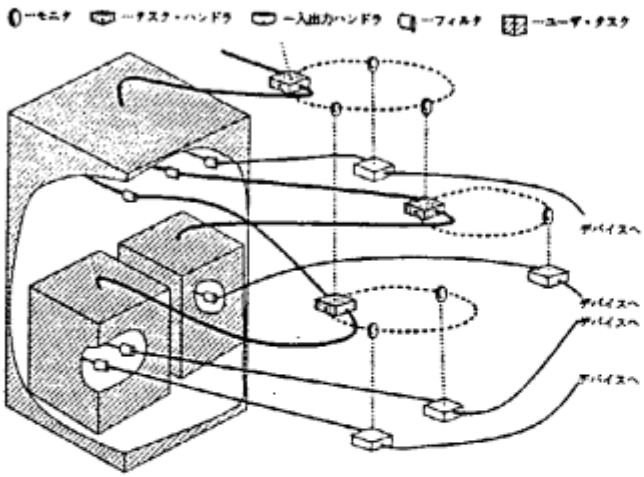

図 1: 資源木とユーザ・タスクの構造

4 資源木による資源の管理

PIMOS が管理する資源には、荘園により割り付けられる CPU 時間やメモリなどの他に、入出力装置などの資源がある。これらの資源を含めた管理単位を「タスク」と呼んでおり、タスク内で更にタスクを生成することができ、それらの子タスクも資源の一つとして管理される。PIMOS では、荘園の機能を利用して、これらすべての資源をその階層構造に沿った「資源木」と呼ばれる木構造によって管理している (図 1)。

この図の右側の部分が資源木であり、各資源を管理するプロセスを構成要素としている。現在の PIMOS では、子供を持つことができる資源は、タスクしかない。従って、資源木の各ノードは、タスクの階層に沿って木構造になっている。また、各タスク内で使用されているそれぞれの資源は、対応する資源ハンドラによって管理され、それらはモニタを介して輪状に結ばれている。

4.1 タスクに対する操作

従来の OS では、タスク (システムによってはプロセスと呼ぶこともある) は、1 つの平坦なテーブルで集中的に管理されている場合が多い。しかしこれでは、タスクに対する操作を行うたびに、このテーブルへのアクセスが集中し、それがネックとなってしまう。そのため、PIMOS では、タスクに対する操作を「タスク・ハンドラへのストリームに操作メッセージを送る」という形で行う。各タスク・ハンドラは、ユーザからのメッセージを受け取ると、タスクを実現している荘園の制御 / 報告ストリーム (図ではユーザ・タスクへの太実線) を使って、それぞれ独立して、タスクに対する操作を行う。

また、「タスク内で使用している全ての資源の状態を返す」といった処理の場合、タスク・ハンドラは、モニタへのストリームにその旨のメッセージを送るだけで、すぐにユーザからの次のメッセージに対する処理を行える。モニタは、そのメッセージを受け取ると、資源ハンドラにメッセージを送ると共に、次のモニタに同じメッセージを送る。このように、メッセージが各資源ハンドラに分散されるため、それぞれ独立して各資源の状態を返すことができる。

一方、従来の OS の場合は、タスク内で使用されている各資源も、1 つの平坦なテーブルで集中的に管理されていることが多い。そのため、上記のような「全ての資源の状態を返す」処理では、それが終了するまで、ユーザからの次のメッセージに対する処理を行うことはできない。

4.2 入出力装置に対する操作

従来の OS での入出力操作は、例えば、「タスク内の入出力資源テーブルの N 番目に登録されている出力装置に出力する」という形で行われることが多い。そのため、入出力操作を行うたびに、そのテーブルへアクセスすることになる。逐次計算機システムの場合は、1 つのタスク内の実行は、所詮、逐次的に行われるため、このテーブルへのアクセス集中は、問題とはならない。しかし、並列計算機システムの場合は、独立した入出力装置への操作は、それぞれ並列に行える。従って、逐次計算機システムと同様の方式を採用してしまうと、資源管理テーブルへのアクセスが集中し、並列性を犠牲にしてしまうことになる。そのため、PIMOS での入出力装置へのアクセスは、タスクの場合と同様に「入出力ハンドラへのストリーム (図では細実線) に直接メッセージを送る」という形になっている。従って、独立した入出力装置への操作は、それぞれ並列に行えるようになっている。

また、実際の物理的な入出力操作は、フロントエンド・プロセッサ (FEP) によって行われるが、それも本体からはプロセスとして扱えるようになっている。従って、物理的な入出力操作も、複数の FEP(マルチ PSI 第 2 版では、最大 4 台までの FEP を接続することができる) によって、並列に行うことができる。

5 おわりに

PIMOS では、プログラムの並列性を犠牲にしないために、従来のシステムのようにそれぞれの資源を集中的に管理するのではなく、プロセスを用いて、それぞれ独立して管理している。今後我々は、この設計方針に基づいて、マルチ PSI 第 2 版上で開発を行って行く。

参考文献

[1] 後藤他: 並列推論マシン PIM– 中期構想 –, 第 33 回情処全国大会,3B-5,1986-10.

[2] 稲村他: Multi-PSI システムの概要, 第 32 回情処全国大会,3Q-8,1986-3.

[3] 佐藤他: PIMOS の概要 – 並列推論マシン用オペレーティング・システムの概要 –, 第 34 回情処全国大会,2P-8,1987-3.

[4] 佐藤他: 並列論理型 OS–PIMOS(1)– 資源管理方式 –, 第 35 回情処全国大会,4D-3,1987-10.

[5] 松尾他: PIMOS のタスク管理方式 – タスク終了時の資源解放 –, 第 36 回情処全国大会,3D-4,1988-3.

[6] ICOT 第 4 研究室編: PIMOS 機能設計書, テクニカル・メモ TM-503, ICOT, 1988.

[7] E. Shapiro and A. Takeuchi, *Object Oriented Programming in Concurrent Prolog*, New Generation Computing, Springer Verlag Vol.1, No.1, pp.25-48, 1983.

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