TR-0840

PIMOS のプログラミング環境

石田　茂、近山　隆

April, 1993

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5

Institute for New Generation Computer Technology

PIMOSのプログラミング環境

石田 茂　近山 隆
(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
{ishida, chikayama}@icot.or.jp

概要

第五世代コンピュータプロジェクトでは, 高度な知識情報処理を行なうにあたって必要とされる計算能力を提供するために, 並列推論マシン (PIM) の開発を進めてきた. PIM は Flat GHC を基本とした並列論理型言語 KL1 が効率良く実行するように設計されている. PIM のオペレーティングシステムである PIMOS は, 高並列に動作するプログラムを効率的に制御し, KL1 言語にとって快適なソフトウェア開発環境を提供している. 本稿では, デバッギング機能に焦点をあて, PIMOS のプログラミング環境を紹介する.

1 はじめに

第五世代コンピュータプロジェクトでは, 高度な知識情報処理を行なうにあたって必要とされる計算能力を提供するために, 並列推論マシン (PIM)[1] の開発を進めてきた. PIMOS[3][4] は PIM の能力を最大限に引き出すために, 高並列に動作する KL1 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である.

本稿では, PIMOS 上での KL1 プログラムの開発の経験をもとに, KL1 プログラムのデバッギングに有効な機能を紹介する. 第 2 章では KL1 言語と KL1 プログラムのデバッグについて述べ, 第 3 章では PIMOS のデバッグ支援機能について紹介し, 第 4 章では PIMOS の性能評価支援機能について紹介する. 最後に第 5 章でまとめと今後の課題を述べる.

2 KL1

2.1 KL1 の概要

KL1[5] は, Flat GHC[6] を基本とした並列論理型言語である.

2.1.1 KL1 の実行モデル

KL1 のプログラムは以下の形をしている.

$$H : -G_1, \ldots, G_m \mid B_1, \ldots, B_n. \quad (m, n \geq 1)$$

ここで, '|'(縦棒) はコミット演算子といい, H をヘッド, 各 G_i をガードゴール, 各 B_i をボディゴールという. '|' より前の部分はガードといい, 後の部分をボディという.

KL1 の実行メカニズムは以下のようになっている.

```
producer(Stream) :- true |
    Stream = [msg|NextStream],    (送信)
    producer(NextStream).
consumer([Msg|Stream]) :- true |  (受信)
    consumer(Stream).

?- producer(S), consumer(S).      (ゴール)
```

図 1: ストリーム通信の例

- Prolog とは異なり，同じ述語のすべての節は並列に試される．
- ヘッドとガードゴールにはその節が選択される条件を記述する．ガードゴールは左側から右側へ逐次に実行され，呼び出し側のゴールの変数が具体化しない理由で，ガードゴールがテストできない場合，そのガードゴールは当該変数が具体化するまで中断する．
- コミット演算子は，ガードが成功した節のうち 1 つだけを選択する．
- 選択された節のボディーゴールは並列に実行される．

このメカニズムによって，プロセス間の通信と同期は，次節で示すように共有変数を用いて行なわれる．

2.1.2 ストリーム通信

ストリーム通信は KL1 の基本的なプログラミング手法である．KL1 では，プロセス間の通信は共有変数により実現するが，継続した通信を行なうために，共有変数をメッセージと新たな共有変数の組にしたデータ構造に具体化する．この通信形態を「ストリーム通信」と呼ぶ．図 1 にストリーム通信の例を示す．通常，KL1 プログラムの実行は，ストリーム通信によりプロセス間でメッセージのやり取りをしながら進んでいく．

2.2 KL1 プログラムのデバッギング

2.2.1 KL1 プログラムでよく見かけるバグ

KL1 プログラムでよく見かけるバグを分類すると以下のようになる．

リダクションの失敗 呼び出されたゴールがどの節のヘッドともマッチしないか候補節のすべてのガードゴールが失敗した場合，リダクションの失敗となる．

ユニフィケーションの失敗 互いに相異なる値をもつ変数同士がボディでユニファイされると，そのユニフィケーションは失敗となる．

意図しない節へのコミット 節の選択条件の記述の間違いにより，意図していない節にコミットすることがある．この場合，ここであげている問題を起こす原因となることがある．

無限ループ ループが何も外部に出力しない場合，無限ループしているプロセスをみつけることは非常に難しい．無限ループしているプロセスは，メモリなど計算資源をすべて消費するかもしれない．プログラムが小さいうちはすべてをトレースすることもできるが，ある程度大きいプログラムになると実際上不可能である．

デッドロック デッドロックとは，2 つ以上のプロセスが互いに他方の計算が終るのを待っている状態をいう．ここでは，デッドロックという言葉を，プロセスが起きることのないイベントを待っている状態という広い意味で使っている．そのようなプロセスは観察できるアクションを起こさないので，見つけるのが難しい．また，このように 1 つのプロセスが実行を中断すると，そのプロセスの出力に依存している他のプロセスも実行を永久に中断する．この依存関係の連鎖は多数のプロセスを含むこともある．したがって，このようなプロセスをすべて報告すると情報量が多くなるので，問題そのものの原因を掴むのが困難になる．

2.2.2 KL1 プログラムの性能向上

KL1 プログラムの性能向上にはいくつかのアプローチがあるが，以下は最もよく用いられるアプローチである．

頻繁に呼び出される述語を見つける 最も頻繁に呼び出されている述語を見つけることで，性能向上のために労力をかけるべき箇所が明確になる．

優先度制御 KL1 のデータフロー同期機構のおかげで，プロセスの実行順序に依存せずに，正しい実行結果が得られる．しかし，スケジューリングによっては，性能が更に向上することもある．KL1 の優先度制御メカニズムは，プログラムの大まかな実行順序の指定を許している．

負荷分散 マルチプロセッサシステム上でプログラムを効率良く実行するためには，計算負荷を各々のプロセッサにうまく分散させることが大切である．一方，分散させすぎると通信量が増え，結果として性能が低下する．このように負荷の均等化と通信の局所性はトレードオフの関係にあり，特に疎結合型のマルチプロセッサシステムではプログラムの性能向上の重要な鍵となる．KL1 では，ゴールの実行するプロセッサを指定する単純な負荷分散機構を提供している．効率のよい負荷分散アルゴリズムを見つけ出すことは，並列処理の最も重要な研究課題の一つである．

3 デバッグ支援機能

3.1 リスナ

リスナは KL1 プログラムの実行を行なうためのトップレベルのユーザインタフェースシステムで，対話的な Prolog システムに似ている．リスナは以下の機能をもっている．

ゴール列の実行 リスナのトップレベルから対話的にゴールを実行することができる．

トレース機能 指定されたゴールのリダクションの過程の詳細を見ることができる．通常のシングルステップ実行の他に，指定した述語のみをトレースするスパイ機能を許している．ゴールの実行を一時的に保留することによって，実行順序を変えることができる．マルチウインドウトレース機能は，トレースしているゴールを複数のグループに分け，トレース情報を別々のウインドウに表示する．

インスペクタ，モニタ機能 インスペクタや変数モニタを用いてデータ構造を調査することができ，データ構造が生成されている最中でも調査することができる．詳細は次節以降で説明する．

プロファイル機能 プログラムの実行情報を集計することができる．計測されたデータは性能評価支援ツール ParaGraph(後述) によって解析され，グラフィック表示される．

デッドロックの検出 ガーベジコレクション中に発見したデッドロック (永久に再開されることのない中断) を報告する. 検出されるゴールは, デッドロックの因果関係の中で原因となっているゴールである.

3.2 インスペクタ

インスペクタは対話的にデータ構造を解析し表示するツールである.

```
@@4096 29    layered: filter([ & , & |E1],1,1,W)
41    * (1)     filter([ & , & |E1],1,1,W)? inspect 1   ← サブゴール 1 を調査対象としてインスペ
filter([3*'[' & ']',4* V1 |T1],1,1,W)> me               ← データの表示              クタを起動
0 : filter
1 : [3*'[' & ']',4* V1 |T1]
2 : 1
3 : 1
4 : W
filter([3*'[' & ']',4* V1 |T1],1,1,W)> 1                ← データを 1 段深く潜る
0 : 3*[ & |Q1]
1 : 4* V1
tail : T1
```

図 2: インスペクタの実行例

3.3 変数モニタ

まだ具体化していない変数の将来の値を調べるには, インスペクタやトレーサからデータ駆動型の監視プロセスを起動する. 監視されている変数の値はその具体化を待って報告される. また, リスト構造のデータがインクリメンタルに具体化していく様子も監視でき, 通信用ストリームのトレースに役立つ.

```
@@4096   1   layered: area(ok,begin,B)
   7     * (1)  colors(G)
   8     * (2)  area1(ok,begin,G,B)? m G     % 変数 G のモニタ
@@4096   1   layered: area(ok,begin,B)
   7     * (1)  colors(G)
   8     * (2)  area1(ok,begin,G,B)?
aq:1 G == [red,yellow,blue,white] ?          % 具体化状況の報告
```

図 3: 変数モニタの実行例

4 性能評価支援機能

4.0.1 Runtime Monitor

Runtime Monitor は, プログラム実行時に各プロセッサの稼働率を表示する. 稼働率は一定時間間隔毎にカラーもしくは白黒の濃淡グラフで表示される. 図 4 に Runtime Monitor の出力画面例を示す. 縦軸はプロセッサ番号, 横軸は時間に対応したサイクル番号を表す. 三角

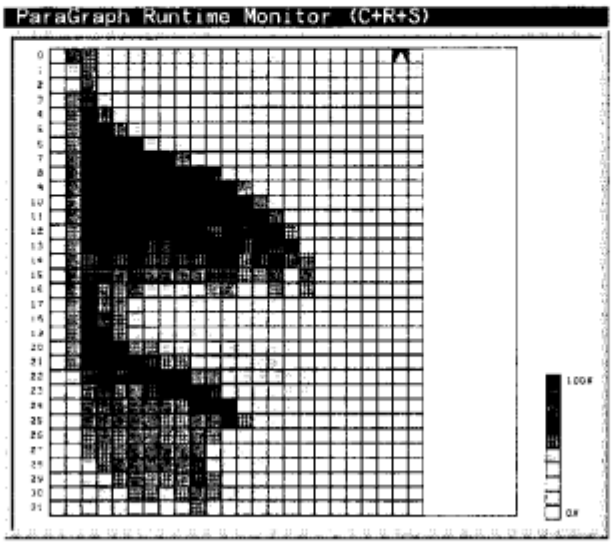

図 4: Runtime Monitor の出力画面例

形 (△) は, ガーベジコレクションの発生を示す. 稼働率以外に, いつ, どのプロセッサで, どんな種類の低レベルのメッセージが送受信されたかという情報を表示することができる. なお, Runtime Monitor のオーバーヘッドは 5% 以下である.

4.1 ParaGraph

ParaGraph[7] は, KL1 プログラムの詳細な実行情報を解析するためのツールである. 計測されたデータは, "What"(どの述語が), "When"(いつ), "Where"(どのプロセッサで) という 3 つの次元をもち, リダクション数やサスペンション数が性能向上に有用な情報である. 逐次プログラムの実行では, "Where" は一定であり, "When" については実行順序が決まっているためそれ程重要ではないが, 並列プログラムの実行では, これら三つの次元の情報は不可欠である.

無限ループの検出

ParaGraph は本来, デバッグのためのツールではなく, プログラムの性能向上を支援するツールであるが, 無限ループの検出にも利用できる. 図 5に ParaGraph を用いて無限ループを検出した例を示す. リダクション数の推移から無限ループしている述語がひと目でわかる.

5 まとめと今後の課題

以上, PIMOS のプログラミング環境について述べた. PIMOS は実験段階であるが, 1988 年以来多数の応用ソフトウェアや PIMOS 自身の開発にも利用されている.

静的解析ツール 現在, 簡易なプログラムの静的解析ツールが提供されているが, クローズ間の入出力関係を解析するような高機能なツール [9] を検討している.

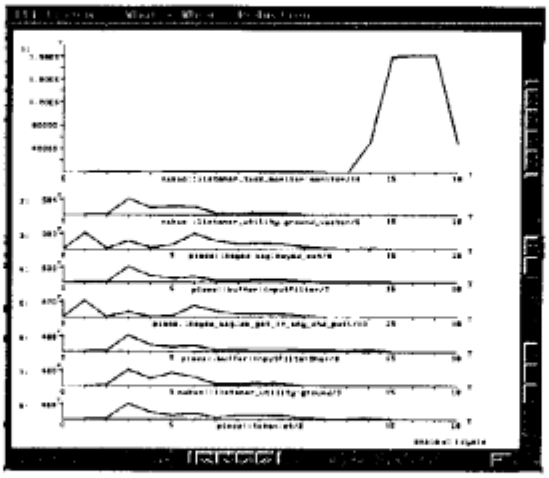

図 5: ParaGraph で無限ループを検出した例

上位言語 KL1 の上位言語として，綾 (AYA)[10] が開発されている．綾 (AYA) はオブジェクト指向プログラミング機能を言語レベルでサポートしており，変数名の綴間違いや引数の順序の間違いといったバグが入りにくいようになっている．綾 (AYA) のプログラムは KL1 プログラムにコンパイルして実行される．

ポータブルな KL1 処理系 KL1 プログラムの移植性を高めるために，KL1 プログラムを C にコンパイルする処理系の検討が進められている．[11]

参考文献

[1] A. Goto, et al. Overview of the Parallel Inference Machine Architecture (PIM). In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pp.208–229, ICOT, 1988.

[2] K. Taki. Parallel Inference Machine PIM. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, pp.50–72, ICOT, 1992.

[3] T. Chikayama, et al. Overview of the Parallel Inference Machine Operating System (PIMOS). In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pp.230–251, ICOT, 1988.

[4] T. Chikayama. Operating System PIMOS and Kernel Language KL1. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, pp.73–88, ICOT, 1992.

[5] K. Ueda and T. Chikayama. Design of the Kernel Language for the Parallel Inference Machine, In *Computer Journal*, Dec. 1990.

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

[7] S. Aikawa, et al. ParaGraph: A Graphical Tuning Tool for Multiprocessor Systems. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, pp.286–293, ICOT, 1992.

[8] Y. Inamura and S. Onishi. A Detection Algorithm of Perpetual Suspension in KL1. In *Proceedings of 7th International Conference on Logic Programming*, 1990.

[9] K. Ueda and M. Morita. A New Implementation Technique for Flat GHC. In *Proceedings of 7th International Conference on Logic Programming*, 1990.

[10] K. Susaki and T. Chikayama. KL1 上の並列プロセス指向言語 AYA. Technical Report TR-652, ICOT, 1991.

[11] T. Chikayama. A Portable and Reasonably Efficient Implementation of KL1. Technical Report TR-747, ICOT, 1992.