PIMOSのセルフ・コンパイラ

佐藤令子[*1]　堀　敦史[*2]　関田大吾[*2]　近山　隆[*3]

[*1]三菱電機㈱情報電子研究所　[*2]㈱三菱総合研究所　[*3]（財）新世代コンピュータ技術開発機構

1．はじめに

新世代コンピュータ技術開発機構においては、第五世代計算機研究開発プロジェクトの中核テーマとして並列推論マシンと並列アプリケーション・ソフトウェアの開発が行われている。このプロジェクトの中期には、並列推論マシン・マルチPSI と、並列論理型OS・PIMOS[1]、その他幾つかのアプリケーション・プログラムなどが開発された。これらのOSやアプリケーション・プログラムの記述は、核言語KL1 でなされている。

核言語KL1 は、並列論理型言語Guarded Horn Clausesのサブセットであるflat GHCに基づいて設計された並列論理型言語である。KL1 で記述されたソース・プログラムは、コンパイラにより、KL1-B[2]と呼ばれる抽象機械語にコンパイルされる。このようなコンパイラとしてはこれまでに、Prolog版、ESP 版などが開発され用いられているが、今回、PIMOS の一部としてマルチPSI 本体上で動作するKL1 セルフ・コンパイラの作成を行った。

KL1 セルフ・コンパイラは、それ自身がKL1 で記述されたコンパイラで、マルチPSI の本体上でコンパイラ自身分散され、並列に実行できる。本稿では、このKL1 セルフ・コンパイラの構造と、その並列性について述べる。

2．コンパイラ

KL1 セルフ・コンパイラは、KL1 プログラムをKL1-B 抽象機械語に変換するプログラムであり、以下の５つの処理フェーズからなる。

①プリ・プロセス
②正規化
③コード生成
④MRB-GC用命令生成
⑤レジスタ割り付け

各フェーズについて説明を行う。

①プリ・プロセス

ファイルから読み込んだプログラム中に現れるマクロ表記を、マクロを含まないKL1 プログラムに展開し、その後、プログラムを節単位に構造化する。

②正規化

コードの生成に先立ち、プログラム中に現れるネストした構造体を、中間的な変数を生成しながら平坦な操作列に展開する。

③コード生成

②で展開されたものを、KL1-B 抽象機械語命令列におとす。ただし、各命令語中の引数は②で変換された中間的な変数表記のままである。

④MRB-GC用命令生成

MRB-GC（単一参照のデータに対するインクリメンタルGC）を行うための命令を生成する。生成する命令語は主にMRB のメンテナンス命令か、ガーベジ回収用の命令である。

⑤レジスタ割り付け

中間表記されている各変数に、レジスタを割り付ける。レジスタの割り付けは、使用するレジスタをできるだけ少なくするように行われる。また、レジスタ転送命令を少なくするために、可能な限り同じデータを扱う変数に同じレジスタを割り付けるよう最適化を行う。

3．アセンブラ

アセンブラの主な仕事は、コンパイラが生成したKL1-B コード列を、対応する機械語命令列に変換し、モジュール内の参照関係を解決することである。

アセンブラでは以下のような構造を生成する。

・GC領域

GC領域は、モジュール外への参照ポインタを格納する領域であり、コード領域から指される。実際に外部へのポインタを張るのはローダの仕事である。

・述語エントリ領域

述語エントリ領域は、外部から参照される述語へのポインタを格納する領域である。

・コード領域

コード領域は、機械語命令列を格納する領域である。

アセンブラではモジュール内の参照関係を解決するために、専用のアドレス・テーブル・プロセスを作って用いる。従来のアセンブラでは、アドレス計算を行う際にラベルなどの登録と検索を２パスで行うのが普通であった。今回用

Implementing KL1 self compiler on PIMOS
Reiko SATO', Atsushi HORI', Daigo SEKITA', Takashi CHIKAYAMA''
'MELCO, 'MRI, ''ICOT

いたテーブル・プロセスは、KL1 で自然にコーディングすると、コード領域全体からの登録が済むまで、検索が自動的に待たされるので、登録要求と検索要求の順序を意識せずに記述することができた。

4. ローダ

ローダは、アセンブラの作成した構造体を基に、KL1 のモジュール構造を生成するプログラムである。変換の必要なデータについては処理を行った後、KL1 のモジュール型データとしてメモリ上に生成する機能を持つ。

ローダでの主な変換処理は、

・アトムの名前からアトム番号への変換

・モジュール間参照の解決

である。

ローダをKL1 で記述する際に注意したこととして、モジュール間参照を解決する時にどのようにしてデッドロックを回避するか、という問題がある。2つのモジュールがお互いの述語を参照しあったときに、どちらも相手のモジュールが完成していないためにデッドロックが起きる。これを回避するために、モジュール内の他のモジュールを参照する部分を未定義（undef）ではなく、一旦ある変数とユニファイして参照（ref）にし、すべてのモジュールがロードされたところで、その変数を相手先モジュールと一気にユニファイするようにした。これらの変数やモジュール定義を管理するテーブルをPIMOS とは別に内部で作成することにより、デッドロックすることなしに、モジュール間述語参照を解決し、未定義モジュールの検出なども可能となった。

5. 並列性

コンパイラ・システム全体としての並列化を考える時、まず考えられるのは、コンパイラ・アセンブラ・ローダといった大まかな処理の分散である。しかし、実際の処理の大部分はコンパイラに集中しており、これらの分散を行っただけでは負荷の均一な分散は難しい。そこで、さらにそれぞれの処理の細分化と、処理されるデータ自身の並列性による、並列化を考えてみる。

・コンパイラ

データの独立性から、モジュール、述語、節を単位とした並列化が可能である。その際、プリ・プロセスなどの各フェーズをパイプライン的に並列化することも考えられる。

・アセンブラ

モジュール単位の並列化が可能である。処理の主な部分がコードの変換であるため、処理の細分化による並列化の効果はあまり期待できない。

・ローダ

モジュール生成のような処理は並列化できる。アトム、モジュール間参照などの解決のため、入力データ全体を対象とする部分は並列化しにくい。

以上の並列性を考慮し、図1のようなプロセス構造をとった。

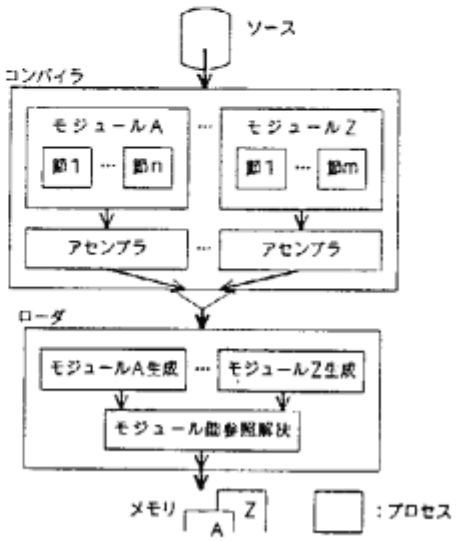

図1. セルフ・コンパイラのプロセス構成

コンパイラは、モジュール単位にサブ・プロセスとアセンブラ・プロセスを生成し、モジュール単位のサブ・プロセス下に、さらに節ごとのサブ・プロセスを作って各節をコンパイルする。ローダは全体でただ一つのプロセスとするが、各モジュールの生成などの処理は並列に実行できる。

6. まとめ

PIMOS 上にKL1 セルフ・コンパイラを作成し、現在、マルチPSI 上で稼動中である。KL1 で作成したため、コンパイラ全体及び各部に、自然な並列性を得ることができた。今後、これらの自然な並列性を生かしたより詳細な分散方式を検討し、性能の向上を目指すと共に、その評価を行っていく予定である。

[参考文献]

[1] T. Chikayama, H. Sato and T. Miyazaki : 'Overview of the Parallel Inference Machine Operating System', Proceedings of the International Conference on Fifth Generation Computer Systems, 1988.

[2] Y. Kimura and T. Chikayama : 'An Abstract KL1 Machine and it's Instruction Set', Proceedings of the Symposium on Logic Programming, 1987.