計算機アーキテクチャ 89−28
（１９９１．　７．　１９）

並列推論マシン PIM/p 用 KL1 処理系の実装

宮崎芳枝　　畑澤宏善

株式会社 富士通ソーシアルサイエンスラボラトリ

平野喜芳

財団法人 新世代コンピュータ技術開発機構

並列論理型言語 KL1 を実行する並列推論マシン PIM には、アーキテクチャの異なるいくつかのサブモジュールがあり、PIM/p は、マクロコールと呼ぶサブルーチンを低コストで実行できるハードウエア機構を持っている。PIM/p 用の KL1 処理系は各 PIM 共通の抽象処理系仕様を基に抽象機械語 KL1-B を解釈実行する部分を展開テンプレートを用いて 3 段階に分けて実装した。すなわち、実行頻度の高く単純で高速を望む処理は展開し、複雑な処理はマクロ命令を呼び出し、さらに複雑で低速でもいい処理はマクロ本体からサブルーチンを呼び出すようにした。本稿では、KL1 コンパイラも含めたこの実装方式を説明し、またシミュレータ上での評価について述べる。

Implementation of KL1 Processing System for Parallel Inference Machine PIM/p

Yoshie MIYAZAKI　　Hiroyoshi HATAZAWA

FUJITSU Social Science Laboratory Ltd.

8, Higashida-machi, Kawasaki-ku, Kawasaki 210 Japan.

Kiyoshi HIRANO

Institute for New Generation Computer Technology

There are some parallel inference machines(PIMs) which execute KL1, having different architecture. PIM/p can execute subroutine called "macro-call" at low cost. We design KL1 processing system for PIM/p based on abstract specification for all PIMs. For implementing its interpreter level, we parted three levels by using templeate. For the simple and frequently executed processing for which high speed is required, we apply machine instruction directly, for the complicated processing we apply high speed subroutine macro-call and for the more complicated processing which does not need high speed, we apply soubroutine called by macro-body. This paper describes its implementation including KL1 compiler, and the evalution result measured on PIM/p simulator.

1 はじめに

現在、ICOTでは異なるアーキテクチャを持ついくつかの並列推論マシンPIMを開発している。PIMは、並列論理型言語KL1を実行するマシンである。

そのうちのPIM/pにKL1処理系を実装した。KL1処理系はPIMの開発に先だって開発された並列推論マシンマルチPSI上ですでに稼働している。マルチPSIは、逐次型推論マシンPSI-IIのCPUを要素プロセッサとして、高速ネットワークで2次元メッシュ状に接続したマシンである。それに対し、PIM/pは、共有バス/共有メモリによって密に結合された8台の要素プロセッサからなるクラスタをハイパキューブ状にネットワークで接続したマシンである。

マルチPSIでは、KL1を実行するための抽象機械語KL1-Bを水平型マイクロプログラムによって解釈実行する方式をとっている。この方式は、オブジェクトコード量は大きくならないが、コンパイラによる最適化がしにくい。処理系の実現方法には、抽象機械命令をより低いレベルのマシン命令に展開する方法もある。この方法は、コンパイラによる最適化がしやすいがコード量が大きくなりがちである。PIM/pでは、両者を合わせた方法を採用している。すなわち、抽象機械命令のうち、高速化が必要で単純な処理はコンパイラが展開し、複雑な処理はマクロ命令と呼んでいる高速なサブルーチンによって実行する。高速化があまり必要でない処理は、サブルーチン呼び出しによって実行する。

本論文では、まずPIM/pについて概要を述べ、KL1処理系の実装方法をKL1コードの生成方式（KL1コンパイラ）も含めて説明する。さらに、PIM/pシュミレータ上で行なった評価結果について報告する。

2 PIM/p

2.1 PIM/pハードウエア

PIM/pは、数百台の要素プロセッサを結合するため、メモリ共有の8台の要素プロセッサからなるクラスタを二重のハイパーキューブ構造のネットワークで結合した2階層の構造をとる（図1）。

要素プロセッサは、命令実行制御ユニット（IPC）、キャッシュ制御ユニット（BSC）、ネットワークインターフェースユニット（NIU）等と、メモリとして、キャッシュメモリ、ローカルメモリ、内部命令メモリ(IIM)から構成される。

命令実行制御ユニットは、1語が8ビットのタグ部と32ビットのデータ部からなるタグアーキテクチャのプロセッサである。キャッシュ制御ユニットは、ライトバック型の一貫性の並列キャッシュである。ネットワークインタフェースユニットは、要素プロセッサ毎にあり、ネットワーク制御部ユニット（Router）と命令実行制御ユニット間のインターフェース制御を行なう。

各クラスタは、要素プロセッサ8台と最大512要素プロセッサをハイパーキューブ構造で結合できるネットワーク制御部ユニット、最大256MB容量の共有メモリ等から構成される。ネットワーク制御部ユニットは、クラスタ間に渡るプロセッサ間通信の非同期制御を行なう。

フロントエンドプロセッサ（FEP）のPSIとは、SCSIインターフェースで結合する。

2.2 PIM/pの命令

PIM/pの特徴的な命令にはタグ判定条件分岐命令とマクロコール命令がある。

2.2.1 タグ判定条件分岐命令

PIM/pでは、レジスタ上のタグ部を調べて分岐するタグ判定条件分岐命令が用意されている。タグ判定によってKL1のデータ型のチェックができる。この命令は、1命令でタグ判定と分岐ができる。PIM/pには、ディレイド分岐命令が用意されていて、これによるとディレイドスロットで実行する命令も含めて、分岐する場合は同じクロック数で実行できる。PIM/pには、コンディションコードを調べる専用の条件分岐命令はなく、コンディションコードもあるレジスタのタグ部に割り付けれていて、このタグ判定条件分岐命令によって調べる。

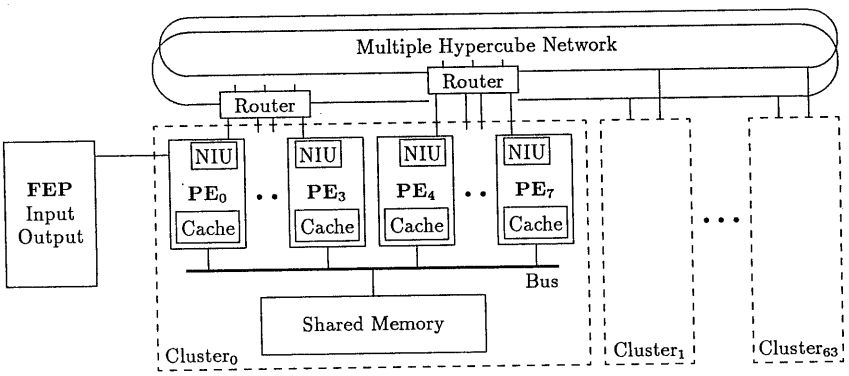

図 1: PIM/p の全体構成

2.2.2 マクロコール命令

PIM/p は、サブルーチンを低コストで実行できるハードウエア機構を持っている。これがマクロコールである。

一般に、サブルーチンを用いるとコード量は小さくなるが、実行時間が多くかかる。このコストは呼び出しと戻る時のコストと、引数の受渡しのコストであり、PIM/p では、引数の受渡しを高速化できる機構が用意されている。

PIM/p ではレジスタの内容を、間接アクセスするための間接レジスタが3個ある。マクロコール命令では、引数として3つのレジスタを間接レジスタに設定できる。マクロ本体では、間接アクセスによって、マクロコール命令のオペランドで指定されたレジスタの内容をアクセスすることができる。間接レジスタを使用することで引数の受け渡しのコストを小さくすることができる。

またマクロ命令本体は、内部命令メモリに格納されているので高速にアクセスできる。

マクロコール命令には、無条件マクロコール命令と条件マクロコール命令がある。条件マクロコール命令の条件は、タグ判定結果を条件とする。

3 ＫＬ１処理系

KL1 プログラムの処理系の全体図は、図2のようになっている。KL1-B は、KL1 を実行するための抽象機械語である。KL1 処理系は、KL1 を KL1-B に翻訳する KL1 コンパイラと KL1-B 実行系の2つからなっている。

この KL1-B を解釈実行する抽象機械は VPIM と呼ばれている。KL1-B は、マシンに依存しない形で設計されているが、マシンにはそれぞれ特徴がある。そこで KL1-B を直接実行するのでなく、KL1 コンパイラのポストコンパイラが展開テンプレートによって KL1-B をマシン依存の言語に変換する。PIM/p では KL1 プログラムも KL1-B 実行系も RISC ふうの命令体系を持ったの PIM/p の機械語に翻訳され、直接実行される。

4 KL1 実行系

4.1 VPIM

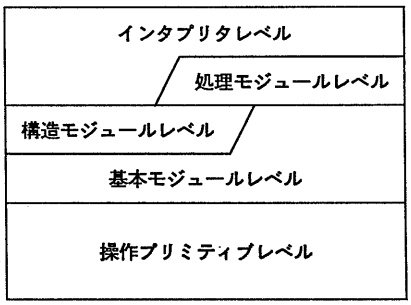

図 3: VPIM の階層

VPIM は、KL1 処理系の各 PIM 共通部分として開発されている。VPIM とは、VPIM 仕様記述言語 PSL で書かれた KL1-B 実行系の仕様である。PSL は、並列推論マシン PIM の機能仕様および実現方式を記述するために開発されたタグアーキテクチャ指向のマクロ言語である。

VPIM は、仕様書であると同時に、コンパイルすることにより PIM 実機のファームウエアプログラムになる。また、PSL を C 言語に変換することにより、VPIM は汎用計算機上でシミュレータ（PIM/s）として動作する。

VPIM は、マシンの詳細部分に依存しないレベルのアルゴリズムを記述することにし、マシン依存の部分はコンパイル時になるべく吸収できるように設計されている。VPIM は、図3に示す5レベルの階層構造をしている。以下に各レベルについて述べる。

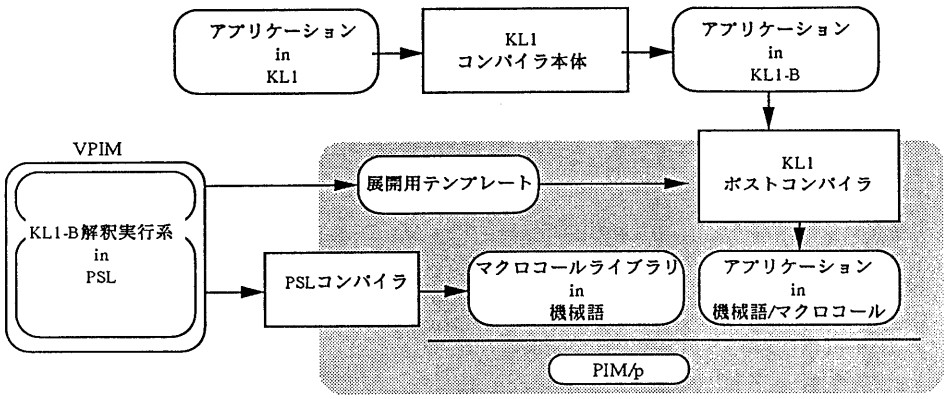

図 2 KL1プログラム実行

- **インタプリタレベル**
  KL1-B の解釈実行とクラスタ間メッセージの解釈を行なう。

- **処理モジュールレベル**
  例外処理等の複雑なインタプリタ部分やクラスタ内負荷分散、クラスタ間メッセージ通信等を行なう。

- **構造モジュールレベル**
  構造体データの定義やクラスタ間処理に使う制御データ構造毎の処理モジュール。

- **基本モジュールレベル**
  メモリ管理、クラスタ間メッセージの送信受信処理を行なう。

- **操作プリミティブレベル**
  上記、各レベルを記述するために用いている共通の低レベル操作ルーチン。

4.2 VPIM から PIM/p へ

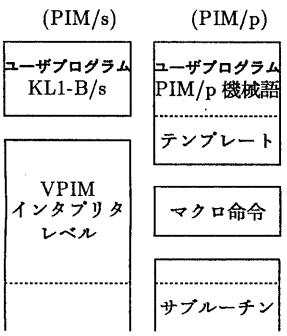

図 4: VPIM から PIM/p へ

VPIM は、PSL によって記述されていて、PSL コンパイラによって PIM/p 機械語に変換され、PIM/p 上で走らせることができる。マシン依存部分や処理効率の向上をはかる部分に関しては VPIM ソースの書換えが必要になる。VPIM のインタプリタレベルとその他に書き換えなければならない部分について次のように実装した。

4.2.1 インタプリタレベル

VPIM のインタプリタレベルは、KL1-B を解釈実行する。PIM/s 上では VPIM は、インタプリタによって、解釈実行されるのに対し、PIM/p は、展開されたコードを実行するので、書き換えなければならない。このインタプリタレベルを KL1 コンパイラの展開テンプレートを用いて実装し、テンプレートで吸収できなかった複雑な部分をマクロ命令、さらに複雑な処理はサブルーチンを用い、3 段階に分けて実装した（図 4）。

基本的な方針としては、マクロ命令やサブルーチンの呼び出しで条件チェックがあるとして、実行頻度の高く単純で高速を望む処理は展開し、複雑な処理はマクロ命令を呼び出し、さらに複雑で低速でもいいものはマクロ本体からサブルーチンを呼び出すようにする。

(1) **展開テンプレート**

展開テンプレートは、KL1 ポストコンパイラにおいて各 KL1-B 命令を機械語に変換する際に参照する。テンプレートは、機械語で 1～6 命令ぐらいで記述できる範囲とし、それを越える複雑な処理については、マクロとして記述し、それを呼び出すマクロコール命令をテンプレート中に置く。テンプレートとマクロ命令の切りわけの基準としては、

- 実行回数の多い KL1-B 命令は、マクロ命令やサブルーチン呼び出すとコストがかかるのでなるべくテンプレート内に収める。
- テンプレートは、一つの KL1-B 命令当たりの機械語が大き過ぎると展開されたコードが大きくなるので 1～6 機械語命令程度とする。
- マクロコールのコストは低いが、何回も呼び出すとコストがかかるので一つの KL1-B からのマクロコールは、なるべく 1 回以内とする。

を考慮した。

(2) **マクロ命令**

マクロ命令は、PIM/p の内部命令メモリ (IIM) 領域に置かれるサブルーチンである。内部メモリは、キャッシュのミスヒットがないため高速アクセスが可能であるが、8K ステップ分の大きさしかないために、ここに大きなルーチンを置くことは出来ない。このため、マクロ命令の基準としては、

- デレファレンスルーチンなど展開するほど小さくはないが比較的小さなルーチンで、使用頻度が高く高速化が望まれるものは、ここに本体を置く。
- それ以外のルーチンについては、外部メモリ上のサブルーチンを呼び出すためのエントリをここに置くものとする。

となっている。

(3) **サブルーチン**

サブルーチンのトップレベルは、IIM 領域に置けなかったインタプリタレベルの処理である。これらは、マクロ命令から呼ばれるサブルーチンの一部として、外部メモリに置かれる。

4.2.2 命令展開の具体例

KL1-B 命令が展開テンプレートによってどのように展開されるか具体例を次に示す。

- 機械語に展開される命令の例

```
is_integer(Src,Lfail) -->
  jnxim(Src,type#integer,Lfail,mask#type)
```

KL1-B 命令 is_integer は一つの機械語 jnxim に展開される。

- マクロコールする命令の例

```
read_wait(Str,ImmPos,Dst,Lsusp) -->
    rtwm(Dst,Str,ImmPosB),
    mcximd(Dst,type#ref,mask#type,
                    m_Deref_Suspend),
    argset(Str,ImmPosB,Dst),
    jxim(Dst,type#ref,Lsusp,mask#type)
```

KL1-B 命令 read_wait はマクロコール命令 mcximd を含む4つの機械語 rtwm, mcximd, argset, jxim になる。

- サブルーチン呼び出しをする命令の例

```
unify(Src1,Src2) -->
      mc(Src1,Src2,arg#dummy,m_Unify)
```

KL1-B 命令 unify はマクロコール命令 mc になる。
マクロ本体は、

```
#MacroBody_define m_Unify()
{
  $CALL( i_ActiveUnify_Sub( I_RegSource1,
                            I_RegSource2));
  $RETURN();
}
```

I_RegSource1,I_RegSource2 は
間接レジスタである。

でサブルーチンを呼び出している。

4.3 最適化

PSL や KL1 をコンパイラした結果の命令列には、人間が記述したのに比べて無駄が多い。また、コード量が大きくなったり、パイプライン・ブレイクが起きて実行速度が遅くなることがある。

そこで次のような PSL コンパイラによる実行系の最適化や KL1 コンパイラによる KL1 プログラムの最適化を行なった。

PIM/p では、命令実行制御ユニットでは各命令を D,A,T,B の4ステージのパイプラインで実行している。

- インタロックを押えるために間に命令を入れる

```
例  rtw  R1 I1 R2     DATB          rtw   DATB
    mcna R1 I2 I3 I4  DDDATB --> Inst1  DATB
                                 Inst2  DATB
                                 mcna   DATB
```

R1,R2 : レジスタ, I1,..,I4 : 即値
Inst1,Inst2 : 命令

rtw でメモリからデータをレジスタに読み込んでマクロコール命令のので型の判定を
しようとすると2クロックのインタロックが生じる。
この場合、間に2命令入れるとパイプラインの乱れがなくなる。

- 命令の順序の入れ換えなどによるディレイドスロットの有効利用

```
例  w8tw   DATB            mcd     DATB
    mc      DATB     -->   w8tw     DATB
    ijnxim  SCDATB         ijnxim   SCDATB
```

マクロコール (mc) では、マクロ本体を実行するまでに1クロック遅れがでる。
mc をディレイド命令 mcd にしてディレイドスロットに適当な命令を入れられれば、空きステージをなくすことができる。

- 結果が使われないような無駄な命令の削除

```
例  li    R3 10
    mvtwt R3 R3 type#int
    add   R2 R2 R3
```

これは、KL1-B 命令 put_integer が展開されたものであるが、li で 10 を R3 にロードし mvtwt でタイプを設定している。add で R2 と R3 を足すが、R3 のタグは参照されないので、mvtwt 命令は不要である。

- 何回も出てくる処理の共通化
  何回も出てくる命令はまとめる。

5 KL1 コンパイラ

5.1 構成と概要

KL1 言語処理系は、KL1 プログラムを機械語に翻訳するためのツール群である。現在は、図5のように、

- KL1 コンパイラ本体
- KL1 ポストコンパイラ
- KL1 アセンブラ
- KL1 リンカ

で構成されている。このうちマシン依存部はポストコンパイラとアセンブラである。以下に、マシン依存部の説明をする。

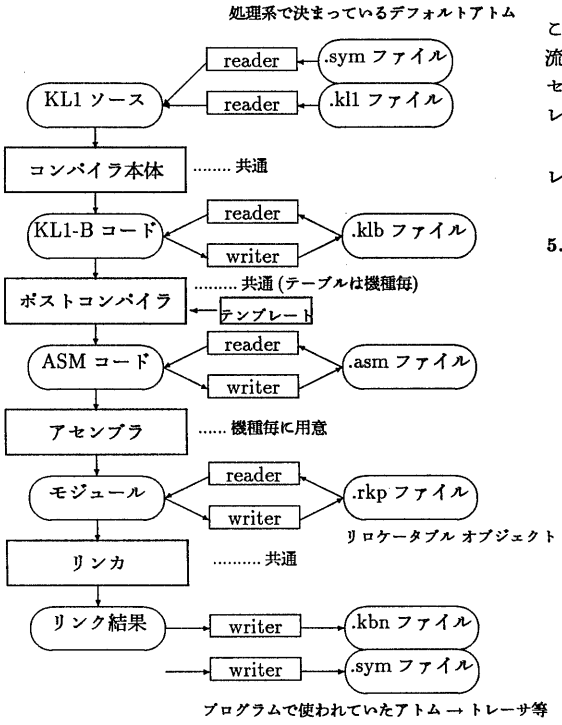

図 5: KL1 コンパイラの全体構成

5.2 KL1 ポストコンパイラ

KL1 ポストコンパイラは、コンパイラ本体の出力する KL1-B 命令を入力とし、アセンブラ制御情報および PIM/p アセンブラ言語を出力する。また、コンパイラ本体の出力した論理レジスタ番号に対し、物理レジスタ番号を割り付ける。ポストコンパイラの処理自体はマシンに依存しないが、参照するテーブル類はマシンに依存する。特に KL1-B 命令を機械語に変換するための展開テンプレートは重要である。

展開テンプレートでは、KL1-B 命令を PIM/p 機械語命令列に変換し、レジスタアロケーションに必要な情報を出力する。これは、マシンに依存する部分である。

テンプレートの元となるのは、VPIM のインタプリタレベルである。テンプレートでは、このインタプリタレベルの処理のうち機械語で簡単に記述できる範囲をサポートし、複雑な処理についてはマクロ命令を呼び出すためのマクロコール命令をテンプレート中に置くことで対応する。コード量が大きくなり過ぎないことや、生成されたコードによる処理の高速化を考えなければならない。設計方針に関しては 4.2.1 を参照。

KL1 コンパイラ本体では、レジスタについては論理レジスタ番号を割り付けている。ポストコンパイラではそれぞれの論理レジスタの寿命を計算し、有限個の物理レジスタに割り付けなければならない。論理レジスタの寿命計算に必要な情報として、PIM/p 機械語命令の種類やレジスタアクセスの種類などを KL1 のモジュールとして与えている。この情報を受けとったポストコンパイラは、命令の実行の流れを表すネットワークプロセスを作り、これに各種メッセージを流すことによって論理レジスタの寿命計算や物理レジスタの割り付けを行う。

KL1 コンパイラは、全て KL1 で書かれており、テンプレートも KL1 で書かれている。

5.3 KL1 アセンブラ

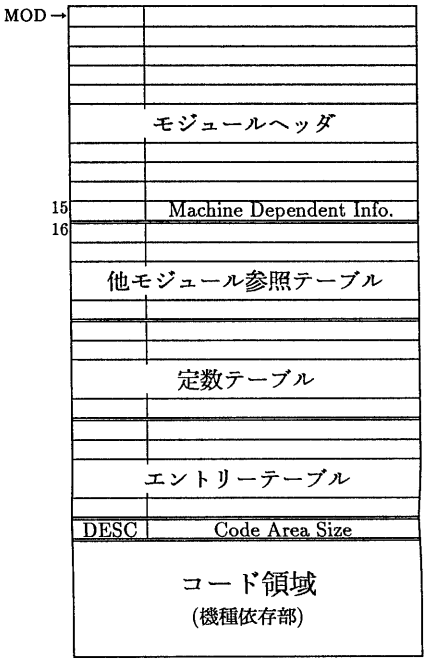

図 6: メモリ上でのコードモジュールの形式

アセンブラは、シンボリックなアセンブラ言語をバイナリに変換する。その結果のバイナリをコードモジュールと呼ばれる構造体に格納している。これは、メモリ管理などのために構造体にしておく必要があるからである。PIM/p の KL1 コードモジュールの形式は、図 6のようにモジュールヘッダ、他モジュール参照テーブル、定数テーブル、エントリテーブル、コードディスクリプタ、コード領域から構成されている。このうち、コード領域は機種依存であるが、それ以外の部分は全てタグ付ワードで各 PIM 共通形式である。

PIM/p 機械語命令は 4 または 6 バイト長で、タグを付けずに詰めてコード領域に格納する。図 6の Machine Dependent info. にはマシン依存の情報が格納されている。PIM/p では、マクロ名とマクロコール位置との対応表をここに格納し、マクロコール時の内部アドレスを決定するリンカに情報を渡している。

表 1: 実行時における命令の状況

プログラム	展開された命令の全実行回数	マクロコール命令の割合 (%)	実際に分岐したマクロコール命令の割合 (%)	一回のマクロコールにより実行したマクロ内の命令個数	
queen	1219006	8.3	81.5	11.6	8 クイーン
prime	90734	11.1	67.2	9.5	素数生成
prime_dd	42607	16.2	78.6	13.7	素数生成
bup3	1235405	11.3	70.9	12.3	構文解析
mastermind	223403	8.0	72.5	10.5	ゲーム
cube	557358	4.9	84.0	11.3	制約充足問題

6 評価

4 章で説明した設計方針による状況を評価するため PIM/p シミュレータ上で以下のような項目について測定を行なった。測定は、全て 1 台の PE で実行したものである。

- 展開された命令の全実行回数に対するマクロコール命令の割合
- 一回のマクロコールにより実行したマクロ内の命令個数
- 展開された命令の全実行回数、マクロ内の命令の全実行回数、サブルーチン内の命令の全実行回数の比較
- マクロコール命令とサブルーチンの比較
- サブルーチンの呼びだし状況

表 1 は、展開された命令の全実行回数、マクロコール命令の割合、実際に分岐したマクロコール命令の割合、一回のマクロコールにより実行したマクロ内の命令個数を示す表である。マクロコールの割合は、展開された命令の全実行回数に対するマクロコール命令（条件付マクロコール＋無条件マクロコール）の割合である。実際に分岐したマクロコール命令の割合は、マクロコール命令のうち実際に呼び出されたマクロコール命令の割合である。

一回のマクロコールにより実行したマクロ内の命令個数は、

（一回のマクロコールにより実行したマクロ内の命令個数）

$$= \frac{(\text{マクロ内の命令の全実行回数})}{(\text{マクロコール実行回数})}$$

で求められる。

どのプログラムもマクロコール命令の占める割合はそれほど多くなく、およそ 10 命令に 1 つがマクロコール命令となっていることがわかる。これは、それほど多くないと考えられる。

マクロコール命令のうち、実際に分岐したマクロコール命令の割合は 67.1 ～ 84.0% である。

一回のマクロコールにより実行したマクロ内の命令個数は、10 程度である。マクロとして実行された命令数がどのくらいの時に効率が良いかも評価したいところである。

表 2: 命令の分布

プログラム	展開された命令 (%)	マクロ内の命令 (%)	サブルーチン (%)
queen	32	31	37
prime	27	29	44
prime_dd	9	21	70
bup3	31	43	26
mastermind	34	28	38
cube	55	31	14

表 2 は、各テストプログラムの展開された命令の全実行回数、マクロ内の命令の全実行回数、サブルーチン内の命令の実行回数の全体の命令実行回数に対する割合を示す表である。

queen,prime,bup3,mastermind では 3 段階でほぼ同数であるが、prime_dd ではサブルーチンが多くなっている。これはサブルーチン呼び出しのある suspend 命令が多く使われているのでこのような結果が出たと思われる。また、cube では約半分が展開された命令である。

マクロ命令はキャッシュのミスヒットがないので多くしてもよいと考えられる。

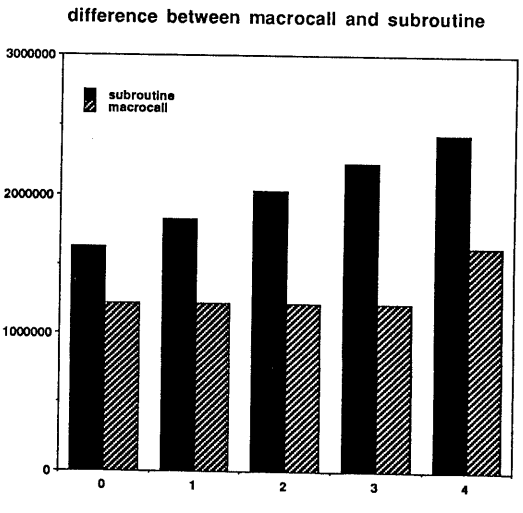

図 7: マクロコール命令とサブルーチン命令の差

図7は、queenについてマクロコール命令とサブルーチン命令の比較をした表である。

マクロコール命令をサブルーチン呼び出しに置き換えた場合を考える。引数なしの場合は5命令に、引数1個の場合は7命令に置き換えられる。そこで、マクロ命令の引数が全てなしの場合、引数が全て1個の場合、... に仮定した。

このようにマクロコール命令によって命令数をかなり減らすことができることがわかる。マクロコール命令は、引数が4個以上になるとサブルーチンと同じように引数の受渡しがあるので、引数3個の場合が命令数の削減率が最大になる。

表3: サブルーチン呼びだし

プログラム	マクロ命令 -> サブルーチン (回)	サブルーチン -> サブルーチン (回)
queen	25534	29514
prime	2359	0
prime_dd	4324	4963
bup3	24044	19833
mastermind	3521	5350
cube	3940	2228

表3はマクロ本体からサブルーチンを呼ぶ回数とサブルーチンがサブルーチンを呼ぶ回数である。サブルーチンがサブルーチンを呼んでネストが深くなるとコストが高くなる。表からは平均して1回だけサブルーチンがサブルーチンを呼んでいることがわかる。primeでは0になっていてマクロ本体から1レベルのサブルーチンが呼ばれるだけになっている。

7 まとめ

PIM/pでは、KL1-B命令が、展開された命令、マクロ命令、サブルーチンの3段階に分かれるが、その分け方は適切でなければならない。

今回初めてPIM/p用にKL1処理系を実装し、シミュレータ上で評価を行なった。マクロコール命令の展開された命令の全実行回数に対する割合はそれほど多くなかった。3段階の命令の分布はほぼ均等なプログラムが多く、マクロの大きさは、10程度であることがわかった。

現在の実装は、マクロ命令を呼びさらにサブルーチンを呼び出しているが、マクロ本体からサブルーチンを呼び出すだけのような場合には、直接サブルーチンを呼び出すように最適化をする予定である。さらに、内部命令メモリ領域の空いたところに別のマクロを置いてマクロの比率を高める予定である。

今後、コード量の縮小と生成されたコードによる処理速度の高速化を図るために正確な評価が必要である。それをもとにして、展開テンプレートの再検討を行なう予定である。

現在は、PIM/pハードウエアの開発も進み、実機でKL1プログラムが動作するようになった。KL1処理系の開発は、まずシミュレータ上で行なわれ、実機上へと移るわけであるが、今後は実機向けに改良も行なう。

謝辞

日頃御指導を頂いているICOT第1研究室のPIMグループを始めとする研究員の方々に感謝します。また、本研究の機会を与えて頂いたICOT第1研究室瀧和男室長に感謝します。

参考文献

[1] D.H.D.Warren : An Abstract Prolog Instruction Set, Technicsl Note,SRI International, 1983

[2] A.Goto,M.Sato,K.Nakajima,K.Taki and A.Matsumoto : Overview of the Parallel Inferrance Machine Architecture (PIM),Proceedings of the FGCS'88, 1988

[3] A.Goto,T.Shinogi,T.Chikayama,K.Kumon and A.Hattori : Processor element Architecture for a Parallel Inferrance Machine PIM/p, Journal of Information Proceedings, Vol.13, No 2, 1990

[4] E.Tick : Performance of Parallel Logic Programming Architectures,Technical Report TR-421, ICOT, 1988

[5] 山本礼己 , 他 : 並列推論マシンPIMにおける抽象機械語 KL1-B の実装 - 高級機械語を実装するための道具立 -, 電機通信学会 コンピュータシステム研究会 並列処理に関する「指宿」ミニシンポジュウム ,1989

[6] 中越靖行, 他 : VPIM及びその開発言語PSLについて, 情報処理学会, 1989

[7] 平野喜芳、後藤厚宏 : 並列論理型言語KL1のコンパイル方式の改良, 並列処理シンポジウム JSPP '90 論文集, 1990

[8] 後藤厚宏 : 並列型推論マシンのアーキテクチャ, 情報処理, Vol.32, No.4, 1991

[9] ICOT 第1研究室 : VPIM処理方式解説書, 1991