ICOT Technical Report: TR-574

TR-574

並列マシンにおける言語処理系の開発環境と実装手法
—PIMのKL1言語処理系を例に—

高木常好，山本礼己，今井　明，
伸瀬明彦，平野喜芳，中越靖行

July, 1990

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

並列マシンにおける言語処理系の開発環境と実装手法
- PIM の KL1 言語処理系を例に -

高木常好　　山本礼己　　今井 明

財団法人 新世代コンピュータ技術開発機構

仲瀬明彦

株式会社 東芝

平野喜芳　　中越靖行

株式会社 富士通ソーシアルサイエンスラボラトリ

並列マシンでは、同期の制御などの並列プログラムを簡単にするため、並列処理機能を有した高級言語が必須である。高級言語の処理系は、言語とマシンが直接実行する命令との間に大きなギャップがあるため、大規模かつ複雑なものになる。しかし、このような言語処理系を効率良く開発する手法には決定的なものが存在しない。我々は並列推論マシン PIM の KL1 言語処理系を開発するにあたって、処理系を記述するための言語 PSL(PIM Specification descriptive Language) を設計した。また、PSL プログラミングを支援するユーティリティ、処理系のシミュレータ、処理系を実機へ実装するためのコンパイラなどを開発した。本稿では、これらの開発環境の内容と現状について報告する。

Developing Environments and Impremention Techniques for Parallel Language Processor
- For example: KL1 Processor in PIM -

Tsuneyoshi TAKAGI　　Reki YAMAMOTO　　Akira IMAI

Institute for New Generation Computer Technology

Mita Kokusai Bldg. 21F, 1-4-28, Mita, Minato-ku, Tokyo 108 Japan.

Akihiko NAKASE

TOSHIBA Coporation

Kiyoshi HIRANO　　Yasuyuki NAKAGOSHI

FUJITSU Social Science Laboratory Ltd.

We are developing a parallel language processor for KL1, a concurrent logic programming language, on parallel inference machines(PIMs). A Language processor for high level parallel programming languages such as KL1 tend to be large and complicated, because there is a wide gap between the parallel programming language and the native machine code. Therefore, it is very important to use sophistcated developing environment for parallel language processors. We provided integrated developing environment, including PIM specification descriptive language(PSL), programming support utilities on an editor, a simulation system, and a compiler to generate native machine codes.

1　はじめに

並列プログラミングでは、同期の制御などの複雑な記述をしなければならない。これをC言語のような我々がよく使っている言語で書いたとしたら、処理が複雑すぎるため、到底まともなプログラミングは望めない。だから、並列マシンにおいては、並列処理機能を有した高級言語が必須となる。こうした高級言語では、言語とマシンが直接実行する命令との間に大きなギャップがあるため、その言語処理系は大規模で複雑なものになってしまう。したがって、高級言語の言語処理系を効率良く開発する手法を考えることは重要である。

図1は、ICOTで開発中の並列推論マシンPIM[1]における並列論理型言語KL1の処理系開発の概要を表したものである。KL1言語処理系においても、言語とマシンのマイクロ命令との間のギャップを埋めるにあたって、それを効率良く実装するために抽象機械語KL1-B[2]を導入した。KL1-B言語は、WAM[3]などに比べると、並列推論機能や、OS機能の一部をサポートするため、その処理は複雑になっている。ICOTでは、これまでにPIMのプロトタイプマシンであるMulti-PSI[4]を開発し、そこでKL1-B言語処理系を実装する実験を行なってきた。このMulti-PSIにおいては、KL1-B言語処理系をC言語で設計し、それをファームウェアに書き直すという方法を採ったが、C言語とMulti-PSIのファームウェア命令の仕様の違いが大きかったため、直接書き直すことは難しいという反省を得た。

そこでPIMのKL1-B言語処理系開発においては、それを効率良く行なうために言語処理系を開発するための言語PSL(PIM Specification descriptive Language)を設計した[5]。PSLは、C言語程度の記述性を保ちながら、PIM実機のファームウェアレベルのアーキテクチャとの親和性が失われないように設計された、タグアーキテクチャ向きのマクロ定義言語である。そして、PIM実機を睨んで設定した仮想マシン上に、このPSLを用いてKL1-B言語処理系を開発している。我々は、この仮想マシン上の処理系をVPIM(Virtual PIM)と呼び、次に挙げる項目を目的とした開発を行なっている。

- 言語処理系の仕様書にする
- 汎用計算機上にシミュレータとして動作させる
- コンパイルして実機に実装する

本稿では、このVPIMの目的を実現するために開発した、PSLプログラミングを支援するユーティリティ、処理系のシミュレータ、処理系を実機へ実装するためのコンパイラなどを紹介する。また、これらの開発環境によるVPIM開発の例を挙げて現状を報告する。

2章で開発環境の全体構成を述べ、3, 4, 5章でVPIMの各目的毎の開発環境を紹介する。また6章ではVPIM開発の現状を述べ、最後に今後課題として残っている開発環境について述べる。

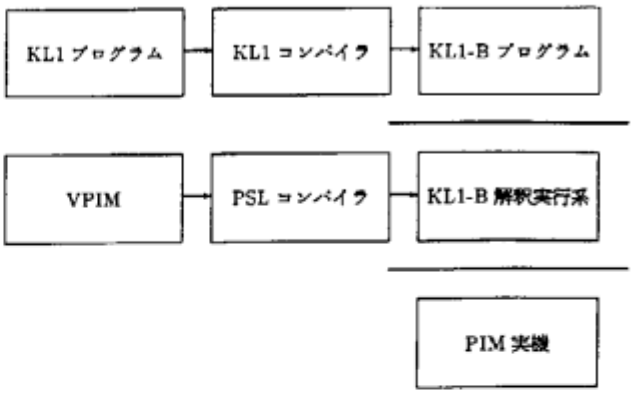

図1. KL1言語処理系の概要

2　VPIMの開発環境の構成

1章で述べたように、VPIMには3つの目的がある。これにしたがって、開発環境はこれらの目的を果たすために、それぞれの面から支援する形で構成される。図2は、VPIMの開発環境の構成を示したものである。

VPIMを仕様書にするという点においては、エディタを中心として、コーディング規約の設定や、クロスリファレンスデータベースなどで支援を行なう。

VPIMを汎用機上でシミュレータとして動作させるという点については、汎用機上にPIMの仮想ハードウェアを構成し、VPIMを汎用機がサポートする言語に変換した後、これらをコンパイル、リンクして動作させ、テストプログラムを用いてテスト、デバッグを行なう。

VPIMを実機に実装するという点においては、VPIMを記述するPSLはマクロ定義言語であることから、そのマクロを展開するツールと、展開後のマクロ定義を実機命令コードに変換するコンパイラから構成される。

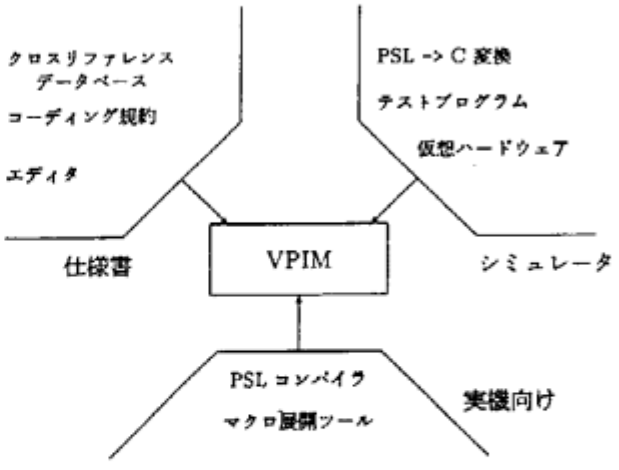

図2. 開発環境の構成

3　仕様書としてのVPIM開発

VPIMはそのまま、あるいは適当な編集をすることによって、PIMの機能仕様、実現方式などの仕様書として読

めることを目的としている。この目的のためにPSLによるマクロの定義に加えて、それに関するコメントをある決められた形式でコーディングしている。

ここでは、VPIMを仕様書とするために設けた規約や、コーディングを支援する環境について述べる。

3.1 コーディング規約

我々はVPIMを複数人でコーディングしていて、各々の好みが異なるため、自由に書いてしまったら仕様書としてまとまりがなくなってしまう。だから、処理系を書く上で、そのコーディングに関する約束を決めることは重要である。次に、我々が採用したコーディング規約のうち主なものを紹介する。

(1) 階層設計

「読み易さ」、「書き直し易さ」を重視して、次に挙げる5つの階層に分類してVPIMを構成した。

- 抽象機械語解釈レベル
  KL1-B命令を解釈実行するトップレベル
- 処理モジュールレベル
  クラスタ内通信処理、例外処理、クラスタ間メッセージ通信処理、ユニフィケーションの基本操作など
- 構造モジュールレベル
  構造体データの定義と操作、ガーベジコレクションなど
- 基本モジュールレベル
  メモリマップの定義、排他制御の基本処理など
- 操作プリミティブレベル
  メモリやレジスタの操作などPSLの基本命令定義

マクロの定義は、この階層の範囲内で自由に書くことができる。ただし、マクロの再帰呼び出しを防ぐために、ある階層の定義ではそれ以上の階層からしか参照してはならないと言う制限を設けている。

(2) マクロ名の規定

マクロ名においては、それから定義している処理内容が一目で想像できるように、各階層を示すプリフィックスをつけること、基本操作を含む場合は統一した操作名を挿入するなどの約束事を決めている。

また名前が長くなるために用いる略語については、その統一をはかるために略語登録ファイルを設けて、そこに登録された略語のみをコーディングに使用していいことにしている。この略語登録ファイルは、開発者全員の了解の下で更新する。

3.2 エディタ

エディタは、専用のものを開発すれば問題ないのであるが、世の中にはすでに優れたエディタがいくつも存在し、これらを使わない手はない。しかし、言語処理系開発特有の機能は、新たに作成しなければならないので、こうして作った機能を容易に付加できるものが良い。例えばEmacsのように、新規に作成したコーディング支援ツールをエディタの中から呼び出すことが簡単なものが望まれる。

現在我々が使用しているエディタにおいても、以下に出てくるVPIM開発支援ツールをエディタ内から簡単に呼び出せるようにしている。

3.3 クロスリファレンス・データベース

クロスリファレンス・データベースは、PSLで記述されたマクロから抽出した定義と参照に関する情報をデータベースとし、マクロの検索、定義と参照の関係の整合性チェックなどを行なう。

(1) 定義情報データ

定義情報は、マクロ、変数、定数がどこで定義されているかを知らせるもので、ファイル名、定義の先頭行、マクロ名(もしくは、変数、定数名)、マクロであればその引き数と言ったものになる。また、そのファイルは誰が担当しているのかという情報も付け加える。実際には、次のような形式の一覧として、データは保持される。

```
マクロ名 引き数 ファイル名 担当者 行番号
```

(2) 参照情報データ

参照情報とは、マクロ、変数がどこで参照されているかを知らせるもので、マクロ名(もしくは、変数、定数名)、ファイル名、参照の行番号、マクロであればその引き数と言ったものになる。また、そのファイルは誰が担当しているのかという情報も付け加える。実際には、次のような形式の一覧として、データは保持される。

```
マクロ名 引き数 ファイル名 担当者 定義マクロ名 行番号
```

この定義参照データベースを使用した検索や整合性チェックツールはエディタから呼び出すことができる。マクロの検索においては、検索したいマクロ名の所にカーソルを持っていき簡単な操作をすると、それが定義の検索の場合にはマクロを定義している部分がエディタのウィンドウに開かれ、参照の検索の場合にはマクロを参照している場所を示す参照情報の一覧が表示される。また、定義と参照関係の整合性チェックにおいては、エディタ上で簡単な操作をすると、その時点で発見されたエラーの一覧が表示される。

このクロスリファレンスデータベースは、VPIM開発者の間では大変好評で、開発効率を向上することにおいて大きく貢献しているものと評価している。また、このツールのために作成する定義参照情報データを用いて、定義や参照の偏りを調べたりする統計情報ツールなども用意している。

4 シミュレータによるVPIMの開発

PIMの研究開発の目的の一つは、並列論理型言語の実行方式をいろいろ試すことである。また、これらの試行はハードウェアの開発とは独立に行ないたい。そこで我々は、既存の並列計算機上にシミュレータを構成することにした。このシミュレータは、PSLで記述されたVPIMを並列計算機のサポートする言語(具体的にはC言語)に変換し、それをコンパイルして動作させるものである。

4.1 シミュレータの構成

PIMは、複数のPE(Processing Element)が共有メモリを介してつながるクラスタ構造と、複数のクラスタがネットワークにより結合された構造との2階層構造を持つため、シミュレータにおいてもこの構造を実現しなければならない。また、VPIMをマシン命令の変わりにC言語に変換したとしても、その周辺を構成するハードウェア群の部分をシミュレータ上に実現する必要がある。

図3にシミュレータの構成を示す。KL1-Bを解釈実行する部分はVPIMをC言語に変換する。ここでは、PSLをC言語に変換するツールを使用する。また、PIMのハードウェアの構成要素をシミュレートする部分は直接C言語で記述した。そして、これらをコンパイル、リンクすることによってシミュレータを構成する。

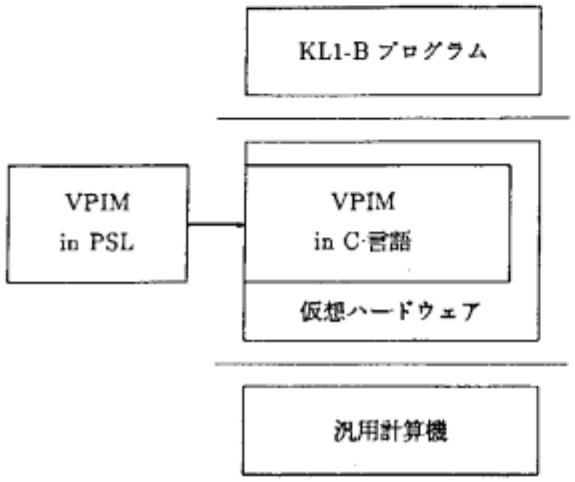

図3. シミュレータの構成

4.2 仮想ハードウェアの実現

C言語に変換したVPIMを汎用並列マシン上で動作するために必要となる仮想ハードウェアの構成要素と、その実現方法について述べる。図4は、PIMの構成を概念的に表したものである。仮想ハードウェアは、この構成をシミュレートする。

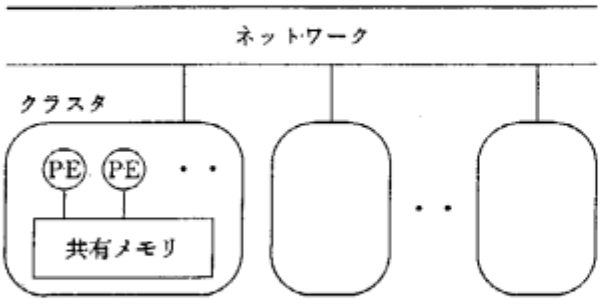

図4. PIMの構成の概念図

(1) タグ付き演算装置

PIMはタグアーキテクチャの推論マシンであるのに対し、汎用並列マシンではタグがサポートされていない。そこで、タグ付きデータの演算処理を、タグ部と値部に分けてシミュレートすることにした。具体的には、タグ付きデータをタグ部と値部とからなるC言語の構造体データとして実現する。

(2) レジスタ

レジスタはタグ付きデータの配列とみなせる。したがって、シミュレータ上では構造体の配列として実現する。また、PE毎に持つレジスタはローカル変数、クラスタ毎に持つレジスタはPE間では共有なので共有変数として実現する。

(3) 共有メモリ

共有メモリもタグ付データの配列とみなせる。ただし、PE間で共有しなければならないので、共有変数として実現する。また複数PEは、あるPEを一つのプロセスとして起動し、他のプロセスをforkすることによって実現する。forkされたプロセスは、共有変数を通して他のプロセスと通信する。

(4) ネットワーク

クラスタ間のネットワークは、複数のシミュレータを起動して、それらを汎用並列マシンが有するプロセス間通信(具体的には、ソケットを利用した通信)を使用して実現する。クラスタ間でやりとりするメッセージは、このソケットを通して送受する。

5 実機向けのVPIM開発

VPIMは、コンパイルすることによって、そのままPIM実機上で動作することを目的としている。一方、PIM実機側では、それぞれ異なる新しい技術要素の実験を目的として5つのハードウェアモジュールの開発をしている。このため、コンパイラは異なるモジュールに対して、そのいづれにも適用できる工夫が必要である。

5.1 マクロ展開

PSLはマクロ定義言語であるため、実機命令にコンパイルする前処理としてVPIMをマクロ展開する必要がある。PSLの構成要素は、マクロ定義の記法、制御構造、および基本マクロのライブラリである。基本マクロの主要操作対象はタグ付ワードの集合からなる二つのデータクラス、レジスタおよびメモリである。PSLで書かれたプログラムは、制御構造、基本マクロ、レジスタ名および定数に展開する。

世の中にはすでにマクロ展開を行なうツールが存在するが、展開途中で得られるデータが欲しいとか、後段のコンパイラのために独自の展開結果を出力したいなどの理由から、PSL専用の展開ツールを作ることにした。以下にこの展開ツールの特徴を挙げる。

(1) 展開終端指定が簡単にできる

この章に始めにも述べたように、PIMは複数のハードウェアモジュールから構成されているため、各モジュールが用意する命令の違いにより、VPIMの基本マクロの切口に違いが生じる。この基本マクロの切口の違いは、いろい

ろなマクロで展開を止めて、それ以下の部分は各モジュール毎に開発することで対応する。このツールでは、展開の止めたいマクロの名前の一覧を与えるだけで良い。

(2) 展開依存関係を出力する

展開依存関係とは、あるマクロを展開した時に、そのマクロ定義と、展開に関わったマクロ定義との関係のことである。具体的には、あるマクロを展開する過程で、どこのどういうマクロを展開して、それがどのファイルに記述されているかを調べて、一覧を出力する。このマクロ展開に関係したファイルの一覧は、PSL ファイルの内容を変更した時、その変更によって展開結果が変更になるマクロの検出に使用する。また、展開の始端となるマクロ全てについて展開依存関係を調べると、展開に関わるマクロ群の偏りが調べられる。

(3) 展開履歴の挿入ができる

これは (2) の展開依存関係に似ているが、あるマクロを展開した時に、そのマクロと展開に関わったマクロ定義の名前と位置を展開結果に挿入するものである。具体的には、あるマクロを展開する過程で、どこのどういうマクロを展開して、それがどのファイルの何行目に記述されているかを調べて、それを一種のラベルのような形で展開結果の中に入れていく。

(4) 自動的にラベル・メンテナンスを行なう

PSL では、ラベルと goto 文をその文法に含んでいるので、マクロ展開をした結果、ラベル名の衝突が生じる可能性がある。この問題を解決するために展開ツールは、ラベルと goto 文の関係を調べて、ラベル名が衝突しないように付け換える機能を有する。

5.2 PSL コンパイラ

マクロ展開ツールの出力は、まだ基本マクロのみで記述された PSL のプログラムである。PSL コンパイラとは、これを入力とし、PIM 実機用の命令コード列を生成するコンパイラである。実際には、コンパイラの出力をアセンブラ、リンカに通して実行形式のモジュールが得られる。

PSL コンパイラでする仕事は大まかに言って、レジスタの割り付け、基本命令の展開、サブルーチンの呼び出しの 3 つである。また、実機は複数のハードウェアモジュールから構成されるため、各モジュールの構成情報を与えると、それぞれに合ったコードを生成する機能を有する。ハードウェアモジュールの構成情報とは、割り付けに使用できるレジスタ数、PSL の基本マクロと実機の命令コード列の対応表などである。

6 VPIM の現状

VPIM の各目的から見た現状について述べる。

(1) 仕様書としての VPIM

現在 VPIM は、PSL で 70K ステップ程であり、その内コメントを除くマクロの定義の部分は 30K ステップ程である。また、コメントに関しても、書く形式を統一したことにより、かなり見やすくまとまったものになっている。

「読み易さ」、「書き直し易さ」という点においては、階層化設計により、大体のところ目的が達成できている。しかし、各階層のコーディング量を比較すると、処理モジュールレベルの割合が他のレベルに比べて大きくなってしまうことが分かった。現在は、このレベルの切り分けの見直しを検討している。

(2) シミュレータとしての VPIM

テストプログラムにより、処理系の動作を確認し、VPIM のデバッグをしている最中である。また、シミュレータ内に統計用のルーチンを入れて、計測した結果から、方式の見直しや効率化を行なっている [2)6)]。

(3) 実機向けの VPIM

PSL コンパイラは KL1 で書かれており、現在は汎用機上で動作する KL1 の逐次処理系上で動作確認を行なっている。その他のツールは、C 言語もしくは UNIX のシェルスクリプトで書かれている。

また、VPIM で KL1-B 命令を全てマクロ展開してしまうと処理系が膨大なものになってしまうことが、展開を実行して分かった。そこで、いろいろな所からよく参照されるマクロ定義に関しては、それをサブルーチンという形で切り出した。現在、KL1-B 命令が 128 個、サブルーチンが 198 個になっている。

図 5, 6, 7 は、あるサブルーチンマクロの展開前、展開後、そしてコンパイル結果の例である。

```
#SUBROUTINE f_Deref_Sub(reg1, reg2)
{
  LOOP(){
    s_DerefReg(reg1, reg2);
    s_IfNotUnbound(reg1) {
      f_ReclaimIfSingleRefPath (reg1, reg2);
      s_IfNotREF(reg1) { break; }
    } else  break;
  }
  $RETURN ();
}
```

図 5. 展開前

```
f_Deref_Sub(reg1, reg2)
{
   LOOP() {
      p_MoveWord(reg1, reg2);
      p_IfMRBEQImmediate(reg1, _MRB_ON) {
         p_Read(reg1, reg1);
         p_MoveWord(reg1, reg1);
         p_SetImmediateMRB(_MRB_1, reg1);
      } else { p_Read(reg1, reg1); }
      TypeSwitch(reg1) {
       case VOID:   case UNDF:   case HOOK:
       case EUNDF:  case ENHOOK:  case RHOOK:
       case WEIREF: case BEIREF: case WEIVAL:
       case RDHOK:  case EXLOCK: case MGHOK:
         p_Compare(D_NULL, D_ONE);
         break;
       default:
         p_Compare(D_NULL, D_NULL);
      }
```

```
        p_IfEQ() {
            p_IfMRBEQImmediate(reg1, _MRB_OFF) {
                p_WriteWithOffset
                        (D_Free_1_PtrReg, reg2, _ZERO);
                p_MoveWord(reg2, D_Free_1_PtrReg);
            }
            p_IfTypeNEImmediate(reg1, REF) { break; }
        } else    break;
    }
    $RETURN();
}
```

図 6. 展開後

```
$LABEL("OO_f_Deref_Sub_2_Label_1")
$LOAD(3,30000)
$RESTORE(3,30000)
deref(4,{0,3})
$SAVE(4,30001)
$SAVE(3,30000)
$VPIM_IF($MACRO(s_IfNotUnbound(3),□),
         "OO_f_Deref_Sub_2_Label_5")
je(3,63,"OO_f_Deref_Sub_2_Label_3")
$LOAD(2,661)
$RESTORE(2,661)
pushtw(2,{0,4})
$SAVE(2,661)
$LABEL("OO_f_Deref_Sub_2_Label_3")
jxim(3,63,"OO_f_Deref_Sub_2_Label_4",63)
$GOTO("OO_f_Deref_Sub_2_Label_2")
$LABEL("OO_f_Deref_Sub_2_Label_4")
$GOTO("OO_f_Deref_Sub_2_Label_6")
$LABEL("OO_f_Deref_Sub_2_Label_5")
$GOTO("OO_f_Deref_Sub_2_Label_2")
$LABEL("OO_f_Deref_Sub_2_Label_6")
$GOTO("OO_f_Deref_Sub_2_Label_1")
$LABEL("OO_f_Deref_Sub_2_Label_2")
$RETURN
```

図 7. コンパイル後

7 今後の開発環境

我々がとった手法による VPIM 開発において、今後の課題として考えているものについて述べる。現在 VPIM はクラスタ内の処理については動作確認が進み、クラスタ間の通信処理方式については、その開発が始まったばかりの状態にある。したがって、今後の課題として中心になるのは、クラスタ間通信処理部のテスト、デバッグ環境である。

7.1 クラスタ間通信処理のデバッグ環境

クラスタ間通信処理のデバッグにおいては、同時に複数のクラスタの様子が一望できることが条件になる。そこで、マルチウィンドウシステムにおいて、複数の VPIM シミュレータを各ウィンドウ毎に起動し、相互のやりとりをウィンドウ間で行なう。また、他のクラスタの内容を別のクラスタから参照できる仕組みが必要であり、VPIM 処理系で実現するメッセージ通信を使用して、データを移動させることを考えている。

現在、クラスタ間のデバッグ環境に関しては、開発者が実際にデバッグした上での意見を聞いて、設計、開発を進めている。

7.2 VPIM のテスト方式

VPIM シミュレータでは、Multi-PSI で使用したテストプログラムを手直ししてテストを行なっている。しかし、PSL コンパイラが本格的に動き出してきた時、コンパイル後の VPIM をシミュレートする必要もあり、これらのテストをどのように行なっていくかは現在検討中である。

8 おわりに

VPIM のコーディングは '89 年頭から始まっており、それと同時に開発環境の整備も始まっている。開発支援ツールに関しては、VPIM 開発者の意見を取り入れて、現在も成長中である。また、VPIM 自身については、これの評価を進め、グレードアップを行なっていく。

コンパイラの手間を考えたため、PSL は C 言語程度の記述性があるとはいえ、まだまだ並列言語の処理系を記述するには低級である。今後は、PSL にオブジェクト指向の概念をとり入れたり、VPIM を KL1 で記述するなどの実験もしたいと考えている。

我々は、言語処理系を記述するための言語を設計し、それによって処理系の開発を行なうという方法を採用した。この手法は、言語処理系開発としては珍しく、今後の言語処理系開発にとって参考になるものと考えている。

謝辞

日頃ご指導頂いている渕和男室長をはじめとする ICOT 第 1 研究室の研究員の方々に感謝致します。また、数々の貴重な助言を下さった PIM 開発グループの皆様に感謝致します。

参考文献

1) A.Goto, et.al., "Overview of the Parallel Inference Machine Architecture (PIM)", In Proceedings. of the International Conference On FGCS'88, pp208-229, 1988.

2) 平野 他, 「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, 1990.

3) D.H.D.Warren, "An Abstract Prolog Instruction Set", Technical Note 309, SRI International, 1983.

4) K.Taki, "The parallel software research and development tool: Multi-PSI system.", In France-Japan Artificial Intelligence and Computer Science Symposium 86, pp365-381, 1986.

5) 山本 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 -」, 信学技報 Vol.89 No.168, pp51-56, 1989.

6) 今井 他, 「共有メモリ結合マルチプロセッサにおける KL1 向き並列実行 GC 方式の評価」, 情処学会研究報告 89-ARC-79-7, pp49-56, 1989.