計算機アーキテクチャ　112－9
（１９９５．　６．　２３）

並列推論マシンPIMの単体プロセッサの評価

久門 耕一

(株)富士通研究所　　マルチメディアシステム研究所

要約

第五世代コンピュータプロジェクトにおいて、並列論理型言語KL1を効率よく動かすための専用並列計算機PIMが開発された。本報告では、KL1高速実行のために導入された専用命令の効果を調べるために、専用命令を持たない汎用計算機のために開発されたKL1処理系であるKLIC処理系と比較する。

比較は、実機での実行時間と、実行される命令のトレースの双方を分析することにより行った。その結果、専用命令を利用する従来の処理系は、最悪時にもある程度の性能を保証する事が出来るが、通常時の性能は、ソフトウェアによる最適化を施したKLIC処理系とほぼ同程度であることが分かった。

Evaluation of an Elementaly Processor of Prarllel Inference Machine

Kouichi Kumon

Multimedia Systems Laboratories, FUJITSU Laboratories LTD.

1015. Kamikodanaka Nakahara-ku, Kawasaki 211, Japan

Email: kumon@flab.fujitsu.co.jp

Abstract

In the FGCS project, dedicated parallel inference machine PIM had been developed for efficient execution of KL1. In this report, we evaluate effects of the dedicated instructions of PIM by comparing both the PIM systems and KLIC which is a KL1 processing system for UNIX based workstations. The comparison is based on the execution time and the execution instruction traces of both processing systems on PIM/p.

The result shows that the former procesing system which utilizes dedicated instructions keeps better performance even in the worst case, but typically, its performance is about the same as that of the KLIC system, because the procesing mechanisms and compilation methods are much optimized and suited for general purpose processors.

1 はじめに

ICOT では、平成 4 年度までの 11 年間の第五世代コンピュータプロジェクトにおいて、並列論理型言語 KL1 を研究するため、および、KL1 を効率よく実行するための適切なハードウェアを研究するため、ハードウェアシミュレータ・マルチ PSI、および、その知見に基づく 5 種類の PIM を作成してきた。

PIM は、KL1 の実行を高速に行なうための専用の命令を持ち、PIM/m は完全疎結合であり、それ以外の PIM は、共有バスで接続された複数の要素プロセッサからなるクラスタと呼ぶ単位を専用ネットワークで接続した並列計算機である。

更に、平成 5 年度から始まった第五世代コンピュータ基盤化プロジェクトにおいては、KL1 プログラムを汎用計算機システム上で実行するための KL1 処理系 KLIC[6] を開発してきた。これは、PIM とは異なり、汎用のハードウェア上で KL1 を動作させるためのシステムであり、KL1 を C にコンパイルし、システムに備わっている C コンパイラを用いてオブジェクトを作成するシステムである。

これら 2 つのシステムでは、設計方針が大きく異なっており、前者は専用ハードウエアによる高速化、後者は、高性能汎用プロセッサ技術とコンパイル技術による高速化を利用したことになる。

PIM システムを評価する際、従来は主に並列時の速度向上、いわゆる台数効果の測定が行なわれてきた。しかし、専用命令を用いたことによる性能向上に関しては評価が難しかった。

専用命令は、ある処理方式で必要とされる処理の一部をハードウェア化したものであり、元となる処理方式を不変のものとした場合、ハードウェア化することで性能が向上するのは当然である。しかし、汎用計算機を用いた場合でも、処理方式の変更やコンパイル技術の向上などにより、専用命令を持つ専用計算機の性能に匹敵する性能を得ることが可能な場合も多い。

本報告においては、KL1 の専用計算機として設計された PIM とその上の KL1 処理系を PIM 上に移植した KLIC 処理系と比較することにより、クロック速度の向上やキャッシュサイズの違いなど、半導体テクノロジによる差異を切り離し、専用命令を持つ PIM の性能について論じる。

2 PIM 上 KLI 処理系と KLIC システム

PIM 上の KL1 処理系と、KLIC 処理系ではその設計方針に大きな違いがある。本節では、PIM を KLIC システムと比較し評価するために、双方のシステムの概略を述べる。

PIM は、記号処理言語である KL1 を効率よく実行するために、次に示すような特徴を持っている。

- タグアーキテクチャをもつ。すなわち、データはその値と独立にデータの型を示すタグを持っている。タグ用には 8 ビットが用意されている。これにより、256 種類のデータ型を表すことが出来る。8 ビットのうちの 1 ビットは、実時間型 GC の一種である MRB 方式 [2] に必要な状態を表すために用いられている。KL1 処理系では、残り 7 ビット、すなわち 128 種類のデータ型のうち、46 種類が使用されている。PIM は、このタグの値を判定し、その結果により分岐するための専用命令を持っており、多くの場合、1 命令でタグの判定、分岐が行われる。

- MRB-GC 処理には、ポインタのデレファレンスと共に、MRB の状態を伝搬させていく必要があるが、PIM には MRB 操作専用命令が用意されており、MRB 操作のための余分なオーバヘッドは生じない。

- KL1 には、型宣言がない。従って、動的な型判定が頻繁に生じ、実行パターンも動的に変化する。しかし、実際にプログラム中で枝分かれして実行されるコードは、全プログラムの一部にすぎない。このため、速度を犠牲にせずに静的なコードサイズを小さくするため、オーバヘッドの小さいサブルーチン呼び出し機構を用意し、比較的呼びだし頻度が小さい KL1 処理に共通する操作を入れることになる。

  このような機構として、PIM/m ではマイクロプログラムアーキテクチャが使用され、PIM/p ではマクロ命令と呼ばれる機構が用意された。

- 複数の要素プロセッサがメモリ空間を共有する、クラスタ構成の PIM では、排他制御のため、メモリの lock/unlock 操作命令が用意されている。これらの命令により、他の汎用計算機にみられるような、compare swap や test and set 等の高機能命令とは異なり、様々

な排他制御プリミティブを低いオーバヘッドで構成することが出来る。

- タグ命令などPIMの持つ特殊命令を利用するために、PSLと呼ぶCに似た言語を開発し、Virtual PIM(VPIM)と呼ぶ処理系をPSLにより作成し、各種PIM向けにはVPIM記述からPSLコンパイラによりKL1処理系のターゲットマシンコードを生成した[1]。

上述のようなPIMの特徴とは異なり、KLICシステムが対象とする計算機システムは、高性能汎用プロセッサを用いたワークステーションなどであり、また可搬性を高くするため、中間言語としてC言語がもちいられた。このため、KL1に必要なタグのために別途領域を用意することは、記憶効率、実行効率の双方の点で好ましくない。そこで、KLICでは、

- WAM[1]で用いられたように、データをアドレスの下位2ビットが常に0となる語境界(32ビットマシンでは4byte境界、64ビットマシンでは8byte境界)に置き、本来0である部分にタグ情報を持たせる方式を取っている。この部分はわずか2ビットしかないため、PIMとは異なり、4種類のタグしか表すことが出来ない。また、タグを判定し分岐する場合には、最低でも汎用命令で2命令が必要となる。
- MRB管理のためのタグが確保できないため、ガーベジコレクションは、一括型のみサポートする。
- 頻繁に実行されるメインパスは、サブルーチンコールを極力行わずに実行できるようにし、それ以外の部分は、コードサイズ削減のためサブルーチンコールにする。
- 複数のプロセッサによるデータ参照の一貫性を保つためのロック/アンロックのオーバヘッドをなくすため、共有メモリを基本とした並列実行は行わない[2]。

これら性格の異なった2つのシステムを対等に比較するためには、同一のハードウェア上で2つのシステムを動かし、比較することが必要である。このため、KLIC/pと呼ぶPIM/p上のKLIC処理系を開発した。これにより、同一のハードウェア上で設計方針の異なる2つのシステムを比較することが可能となった。

[1]ユーザのKL1ソースは、KL1コンパイラによりターゲットマシンコードへと直接コンパイルされる。

[2]共有メモリを通信路として用いたり、共有メモリ上に置くための特殊なオブジェクトなどはある。しかし、これらはKLICの核部分ではない。

3 測定方法

実行環境 KLICシステムは、汎用計算機システム上でKL1言語を実行するために開発されて来た処理系である。KLICはKL1言語をコンパイルしてC言語のプログラムを生成し、ホストシステムのC言語処理系により実行モジュールが作成される。従って、PIM上で動くKLICシステム(KLIC/p)を次のように作成した。

1. PIM/pのためのGNU Cコンパイラ
   KLICシステムに用いるCコンパイラは最適化を十分に行なわなければ性能の良いオブジェクトコードを生成することが出来ない。そのため、最適化コンパイラとしてGNUのCコンパイラを利用し、PIM/pのためのコンパイラを作成した。ここで生成されたコードは通常のワークステーション用のコンパイラに比較して同程度のオブジェクト品質を持っている。
2. KLICクロスコンパイル環境KLIC/p
   本来、KLICシステムはUNIXシステム用に開発された処理系であるため、UNIXを持たないPIM/p上で実行するには、UNIXの持つシステムコールやライブラリ関数などを、PIM/p上で(疑似的にでも)用意する必要がある。このため、KLICシステムを動作させるために最低限必要なライブラリ群を作成した。これらのシステムを用いて、KLICシステムをクロスコンパイラとして使用し、PIM/pシステム用のKLICクロスコンパイル環境を作成した。

この評価を行う上で、PIM/pシステムはKLIC処理系と従来のKL1処理系の両方を実行することが出来るため、マシン構成やオペレーティングシステム(OS)の違いに依存しない評価を行う上で重要な役割を果たす。リスト反転プログラムの他に、リスト反転プログラムとは大きく性格の異なるプログラムとして、lifeプログラムに関しても、詳細に実行を調べた。

ガーベジコレクタとヒープサイズ ガーベジコレクタ(GC)の設計に関しても、双方のシステムでは大きな違いがある。全てのPIM上のKL1処理系では、Multiple Reference Bit(MRB)と呼ばれ

る実時間ガーベジコレクタが実装されており、推論実行中のメモリ消費速度を下げている、MRB で回収できなかったごみでヒープが使い尽くされたときに、一括型のガーベジコレクタが起動される。

これに対して、KLIC システムでは、これは、KLIC システムでは、MRB 実装に必要な 1 ビットのタグビットを持つ余裕がない事と、汎用命例では MRB 操作のためのオーバヘッドが大きく性能の劣化が大きいと予想されたため実時間型でのガーベジコレクションは行なっていない。従って、KLIC システムでは、ごみはコピー方式の一括ガーベジコレクタによってのみ回収される。

PIM システムとは異なり、KLIC システムは、UNIX オペレーティングシステム上での動作を前提としており、UNIX のマルチタスク機構や、仮想記憶機構のため、プロセッサの持つ物理的なメモリを KLIC に直接的に割り当てることは出来ない。つまり、あまりに大きなヒープ領域を割り当てれば、ガーベジコレクタの起動頻度は下がるがヒープがディスクにスワップアウトされ、かえって性能が悪くなることが予想される。

そこで、KLIC システムでは、ガーベジコレクタによりヒープ上に十分な空き容量が確保できない場合に、ヒープを段階的に大きくすると言う方式を用いている。このため、最終的なヒープサイズや、ガーベジコレクタの起動回数は、ヒープの初期初期サイズや実行状況により大きく異なるため、性能も大きく変動する。

このように、KLIC の性能は、ヒープの初期サイズの設定に依存するため、本節の以下の測定では、KLIC の性能は、初期ヒープサイズとして標準値 (24K 語) を使用することにした。また、PIM/p 上の KLIC (KLIC/p) では、10M 語のヒープを初期値とする測定も行った。この値は、ほぼ PIM 上の KL1 処理系のヒープサイズと同じである。

ベンチマークに使用したプログラム群は、append, puz15, pento, bp100x100, waltz, zebra, puzzle, life であり、これらは、従来から PIM/p システムの性能改善のために用いられてきたベンチマーク群である。

4 測定結果

まず、測定したそれぞれのシステムの append 性能を比較する。従来から、append はシステムのリダクションスピードを示す指標として良く用いられてきた。4.1 節で、append 実行の詳細な内容を検討する。

表 1 に PIM/p システムと KLIC システム、そして、他の 4 種類の PIM の計 7 種類のシステムの append Reduction Per Seconds(RPS) を示す。プロセッサ内のキャッシュメモリの影響をみるために、要素数としては、1500 要素と 5000 要素に関して調べた。

表 1: 各マシンでの Append RPS

System	clock (ns)	nrev1500 (RPS×10³)	nrev5000 (RPS×10³)	Ratio
PIM/m	65	600	411	.69
PIM/p	80	305	281	.92
PIM/c	66	55	56	1.02
PIM/i	240	65	65	1.00
PIM/k	100	76	76	1.00
KLIC(1)	28	1,151	1,173	1.01
KLIC/p	80	225	238	1.06
KLIC/p(2)	80	405	386	0.95

(1) SS10 上の時間は CPU 時間で測定
(2) ヒープサイズの初期値 10M 語

KLIC/p の 2 つのヒープサイズ設定での速度の差から、KLIC システムの性能は、ヒープサイズの初期値に大きく依存しており、一口で言い表すことが困難である。ヒープサイズを小さく設定すると、ガーベジコレクションの起動回数が増え、性能が低下し、ヒープサイズを大きくすると、キャッシュミスヒット、更には、仮想記憶によるディスクへのスワップアウトなどにより、性能は低下する。

KLIC システム測定時の OS に依存する不確定性は、PIM/p 上の KLIC システムである、KLIC/p を用いることにより、排除することが出来る。これは、汎用 OS という不確定性をもたない PIM の特徴である。KLIC/p 上のヒープサイズとしては、KLIC の標準値である 24K 語と、PIM/p 上の KL1 システムのヒープサイズにほぼ等しい 10M 語の 2 つに関して調べた。

図 1にみられるように、PIM/m システムでは 1500 要素と 5000 要素のリスト反転では、性能がかなり違う。これは、PIM/m のキャッシュサイズが 4K 語であり、5000 要素のリスト全てを キャッシュ上に保持することが出来ないからと考えられる。append にみられる実行パターンは、ストリーム上へのデータの送出、受け取りを表すものと考えることが出来る。このような操作は、KL1 プログラムに普遍的にみられるものであり、

append を用いることで、システムの性能の一部を知ることが出来る。しかし、append 以外のベンチマークプログラムの構造は、より複雑であり、実際のアプリケーションプログラムの性能に影響するようなサスペンドリジュームを行うものも含まれる。このように、append と他のベンチ

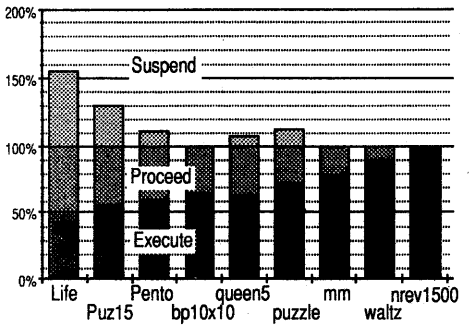

図 1: ベンチマークプログラムの動特性

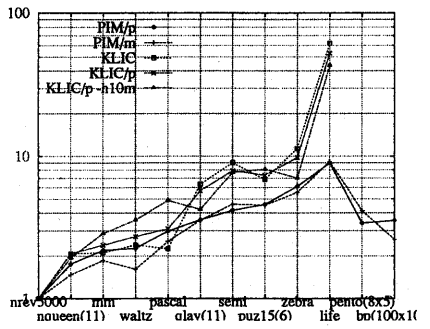

図 2: 正規化したリダクション時間

マークプログラムは性格が異なると考えられるが、その差異を明らかにするため、KL1 プログラムを実行する際の処理を execute, proceed, suspension の 3 つに大きく分類した。

総リダクションに対するそれぞれの処理の割合を、図 1に示す。

ここで測定した全てのシステムは終端最適化を行っているので、execute は最も軽い操作となる。execute と proceed はリダクション処理を構成するので、この 2 つの合計がリダクション数と一致する。ボディー部にユーザ定義ゴールが無い節の実行時に proceed 処理が行われ、それ以外の場合には、execute 処理が行われる。図中の 100%を超える部分は、リダクション数に対するサスペンド処理の割合である。KL1 の特徴である並行実行は、中断/再開処理を繰り返すことによって実現される。しかし append では中断/再開処理は全く行われないため、並行動作の性能は反映されていない。

図 2に各ベンチマークの平均リダクション時間を append のリダクション時間で正規化したものを示す。

このように、ほとんどのプログラムにおいて、1 リダクションに必要な実行時間は、append の 1 リダクション時間よりもかなり大きい。すなわち、一般のプログラムにおける RPS 値は append における RPS 値よりもかなり小さいものとなる。

表 2に様々なベンチマークプログラムにおける、全 PIM の KL1、KLIC および、ヒープサイズ標準設定と 10M 語に設定時の KLIC/p 処理系の比較を示す。この表から明らかなように、life 以外のほとんどのプログラムにおいて、KLIC/p の性能と PIM/p 上の KL1 処理系の性能は、±20%程度の差しかないことが分かる。そこで、Life プログラムの挙動については 4.2 節で詳細に解析する。

4.1 append の実行コードの分析

本節では、KLIC システムがどのようにして専用命令を用いた PIM システムと同等の性能を得ているのかを分析する。ここでは、append を KLIC/p システムで実行した場合の命令トレースと PIM/p 処理系上での命令トレースを比較することにより、実行命令レベルの詳細な比較を行う。

PIM/p と KLIC/p の双方のシステムで実行した命令列トレースを見やすくするために、疑似 C コードで表したものを図 3と図 4に示す。それぞれのの C コード 1 行は、命令トレースの 1 命令に対応している。KLIC/p システムでは、append のメインループ内で、21 命令が実行されている。この命令数は、KLIC を SPARC や DEC Alpha プロセッサ上で実行させたものと同一の命令数である。一方、PIM/p システム上では、同じループが 28 命令で実行されている。このように、KLIC システムの方が短い命令数で append の実行が行われているが、その理由を明らかにするため、実行された命令をその目的に応じて分類したものが表 3である。

この表から、PIM/p システムに関し次のことが分かる。

- 全体の 20%の命令が MRB 処理のために使用されている。
- VPIM 処理系でクラスタ内並列処理を実現のために必要な lock/unlock 処理の割合は大きくない。
- 荘園管理のために必要な命令は、あまり大きくはないが、無視できない量を占める。
- PIM/p システムでは、データのタグ検査を

表 2: 各処理系の相対性能

システム	nrev5000	life	mm	pascal	semi
PIM/p	*44452*	*11458*	*13517*	*3546*	*10832*
PIM/p	1	1	1	1	1
PIM/m	1.46	1.43	1.72	1.73	1.32
KLIC(1)	4.17	0.60	4.33	5.54	1.93
KLIC/p	0.85	0.14	0.78	0.82	0.45
KLIC/p(2)	1.37	0.28	1.05	0.83	0.74
システム	nqueen(11)	qlay(11)	waltz	zebra	puz15(6)
PIM/p	*28486*	*22144*	*9667*	*8782*	*131905*
PIM/p	1	1	1	1	1
PIM/m	1.74	1.47	2.04	1.61	1.48
KLIC(1)	3.53	2.34	3.95	2.25	2.78
KLIC/p	0.72	0.53	0.70	0.53	0.53
KLIC/p(2)	1.24	1.16	0.87	1.20	0.78

イタリック体の数字は PIM/p 上での実行時間 (ms)
(1) SS10 での測定は CPU 時間 (ms)
(2) ヒープサイズの初期値は 10 M 語

表 3: 実行命令の分類

Category	PIM/p		KLIC/p
型判定	6	<	7
MRB 処理	6	≫	–
メモリ書き込み	2	<	3
荘園処理	3	>	–
排他制御	2	>	–
デレファレンス	2	>	1
メモリ割り付け	2	<	3
メモリ読み出し	1	<	2
スリット・チェック	1	<	3
データ移動	1	=	1
その他	2	>	1
計	28	>	21

6 回 6 命令で実行しているが、KLIC/p システムでは、3 回 7 命令で行なっている。

- 荘園管理のために必要な命令は、あまり大きくはないが、無視できない量を占める。

MRB によるごみ集めは、KL1 言語のメモリ消費率を大幅に削減するが、以下に示すようなデメリットも存在する。

デレファレンスと型判断 前述のように、PIM/p システムでは、タグをチェックするために必要な命令は一命令である。それに対し、KLIC/p システムでは、2 個以上の命令が必要である。しかし、PIM/p システムの方がデータ型の判定回数、すなわちタグの検査回数が多いため、タグ判定のために使われた命令数は、PIM/p システムと KLIC/p システムでほぼ同じ命令数になっている。また、MRB によるごみ集めを行うためには、コンスセルなど、構造体の中に変数セルを持たせることが出来ない。append プログラムにおいては、コンスセルの cdr 部分は未定義変数になることが多い。従って、構造対外におかれた変数セルにアクセスするために、間接参照が一段入り、型判断回数が増える。

余分なデータの移動 append のコードにおいては、MRB により、入力として渡されたコンスセルはまず回収され、その後で、再度利用されることになるが、回収時に、コンスセルの中身を保存しておかなければならない。そのため、データの移動回数が多くなっている。

フリーリストの管理 MRB により回収されたセルは、フリーリストにつながれる。そして、新しいセルと必要とする場合には、常にフリーリストから取り出されている。このとき、フリーリストが〜でないことを確認するための命令が常に必要になる。

GC のコスト MRB による GC を行う場合、そのためのオーバヘッドは、ほぼヒープの消費量に比例し、大雑把に言えばリダクション数に比例する。一方、一括型の GC であるコピー方式 GC では、GC のためのコストは、ヒープ内で実際に使われているセル数に比例する。このことから、MRB による GC はヒープの大半が使用中のセルにより占められている場合に有効である。明らかに、ここで調べたベンチマークプログラムでは、MRB による GC が有利な条件ではない。つまり、MRB

```
append(x, y, z)
{
 top:
  if (is_ref(x)) {                          typechk
      t = x; x = deref(x);                  deref
      if (is_undef(x)) goto susp;           typechk
      if (!is_mrb_on(x)) {                  MRB(gc)
          *t = flist; flist = t;}           MRB(gc)
      if (is_ref(x)) goto ...; }            typechk
  if (!is_list(x)) goto ...;                typechk
  d = cdr(x);                               read(cdr)
  if (is_mrb_on(x)) new_list(x);            MRB(gc)
  if (empty(flist)) make_flist();           alloc
  new_w = flist; flist = *new_w;            alloc
  cdr(x) = new_w;                           write(cdr)
  *new_w = UNBOUND;                         MRB(penalty)
  {   /* mcall unify_bound(z, x) */        (mcall-penalty)
    if (is_ref(z)) {                        typechk
        work1 = z; z = deref(z);            deref
        if (!is_undef(z)) goto ...;}        typechk
    work2 = lock_read(work1);               exclusive
    if (is_mrb_on(work2)) {...};            MRB
    if (!is_undef(work2)) goto ...;         exclusive
    z = x;                                  redundant
    unlock(*work1) = x;}                    write(unify)
  x = d;                                    MRB(penalty)
  z = new_w;                                move
  reds =                                    shoen
    reds -                                  shoen
      2;                                    shoen
  if (reds > 0 && !slit_chk()) goto top;    slitchk&loop
}
```

図 3: PIM/p 上 append の実行命令トレース (疑似 C コード)

```
append(x, y, z)
{
 top:
  tag = tagof(x);                   typechk
  if (is_list_tag(tag))             typechk
    goto list;                      typechk
                    delay_slot(redundant)
list:new_word = allocp;             alloc
  *allocp = allocp;                 write(cdr)
  work = car(x);                    read(car)
  *(allocp+1) = work;               write(car)
  new_cons = mkcons(allocp)         alloc
  tag = tagof(z)                    typechk
  allocp += 2;                      alloc
  if (!is_ref(tag)) goto ..;        typechk
  work = *z;                        deref
  if (z == work)                    typechk
    goto L;                         typechk
L:*z = new_cons;                    write(unify)
  x = cdr(x);                       read(cdr)
  z = new_word;                     move
  work = heaplimit;                 slitchk
  if (allocp < work)                slitchk
    goto top;                       slitchk
}
```

図 4: KLIC/p 上 append の実行命令トレース (疑似 C コード)

によるごみ集め適していないと言うことが出来る。

以上のような考察から、我々が調べたベンチマーク群において、KLIC のメモリ管理方式は十分に効率が良いものであるという事が出来る。

4.2 life プログラムの分析

life プログラムは、append プログラムとは対照的な特性を示すプログラムである。PIM/p ハードウェアで測定した結果から、KLIC システムは、ほとんどのプログラムで、PIM 上 KL1 システムと互角あるいは、若干速い場合もあることが分かった。しかし、life プログラムは例外的に遅く、標準のヒープサイズを用いた場合には、従来のシステムの 7 倍も遅い。この性能の低下は以下のように説明することが出来る。

- life プログラム中のゴールは互いに強い依存関係があり、高頻度で中断/再開を繰り返している。KLIC システムにおいては、ゴールレコードや、サスペンションレコードは全てヒープ中に割り当てられる。そして、使用済みの領域はガーベジコレクタによってのみ回収される。しかし、PIM システム上の KL1 処理系では、ゴールレコードや、サスペンションレコードは、使用し終わったとたんにフリーリストに回収される。通常、これらの構造体は、コンスセルよりもかなり大きいため、中断/再開の頻度が高い場合には、ヒープの消費量を劇的に増加させ、ガーベジコレクタをたびたび呼び出すため性能の低下が著しくなる。KLIC/p システムで、ヒープサイズが標準値の場合とと 10m 語の場合の比較を行うと、小さいヒープサイズの場合、ガーベジコレクタがより頻繁に起動され性能が約半分まで低下することからも確認する事が出来る。

- KLIC システムでは、頻繁に実行されると思われるコードは、インライン展開し、頻度の少ないと思われるコードは、実行時ライブラリに集めている。また、実行コードを小さくし、速度を向上させるために、実行時に起きたあらゆる例外事象は一つの例外処理エントリによって扱われる。この例外事象の中には、残りヒープ量不足によるガーベジコレクタの起動や、ゴールの中断が含まれている。

中断要因に対応する処理は、例外を扱う実行時ライブラリが要因分析を行った後に行われるため、比較的オーバヘッドが大きい。このため、中断/再開処理は、PIM/p システムの方がかなり高速である。

5 おわりに

ICOT が開発した 5 種類の PIM のうちの 1 つ PIM/p を用いて、PIM の専用命令による高速化について、ICOT が開発を行なってきた汎用 UNIX ワークステーション上の KL1 処理系 KLIC との比較を行なった。その結果、

- KL1 を効率良く実行する専用命令を備えた PIM は、専用命令を利用することにより、MRB 処理によるオーバヘッドを隠蔽し、専用命令を持たない計算機において、MRB 処理を行なわない場合とほぼ同程度の速度で実行が可能である。
- PIM 上 KL1 処理系の基本部分である VPIM に基づく処理系では、共有メモリによる並列処理を行なうためのオーバヘッドは、append プログラム実行時で、約 7%程度と小さいことが分かった。
- PIM 用 KL1 処理系で採用している実時間ガーベジコレクション MRB は、プログラム中で使用するメモリ量が小さいようなベンチマークプログラムにおいては、かなりのオーバヘッドをもち、一括型 GC の方が有利である。

であることが分かった。

PIM システムは、並列論理型言語を実行する専用計算機として作成され、実際にユーザに使われた数百台規模の並列計算機である。

しかし、本報告でも述べたようにその要素プロセッサの性能は、汎用計算機で行なわれる処理方式の最適化による性能向上が生かされていない点もある。これは、処理系の性能がアーキテクチャの性能だけではなく、処理系開発のための環境 (コンパイラ、デバッガ、パフォーマンスチューニングツール) に依存しており、これらが充実している汎用計算機システムの方が種々の実験が用意に行なえ、最適化したシステムが作りやすいからである。これらが、今後の専用計算機システム作成の課題であると考える

謝辞

本報告をまとめるにあたり、執筆の機会を与えていただいた、(株) 富士通研究所 林 弘 システム研究部門長、川戸 信明 情報網システム処理研究部、有益な討論、助言をいただいた、ICOT 研究所 内田 俊一 所長、元第一研究部 近山 隆 部長 (現在 東京大学工学部 電子工学科 助教授) 並びに元第一研究部の研究員の方々に感謝致します。

参考文献

[1] D. H. D. Warren. An Abstract Prolog Instruction Set. *Technical Report 309, Artificial Intelligence Center*, SRI International, 1983

[2] 木村 康則、近山隆並列論理型言語 KL1 の多重参照管理によるガーベジコレクション、情報処理学会論文誌, Vol.31, No.2, pp.316–327 (1990)

[3] ICOT 第 1 研究室編、VPIM 処理方式解説書、TM-1044, ICOT (1991)

[4] ICOT, 平成 5 年度発電設備診断システムの開発調査研究開発報告書 融合化設計・試作編 (I 融合型基本ソフトウェア技術) (1994)

[5] K. Kumon et al. Architecture and Implementation of PIM/p. In *Proc. of International Conference on Fifth Generation Computer Systems*, pp. 414–424. ICOT, June 1992.

[6] Takashi Chikayama, Teturo Fujise, and Daigo Sekita. A Portable and Efficient Implementation of KL1. In Manuel Hermenegildo and Jean Penjam, editors, *Proceedings of International Symposium on Programming Language Implementation and Logic Programming*, number 884 in Lecture Notes in Computer Science. Springer-Verlag, 1994.

[7] Richard M. Stallman. Using and Porting GNU CC for version 2.6. Free Software Foundation, 1994