ICOT Technical Report: TR-478

TR-478

マルチPSI/V2におけるMRB方式を用いた最適化手法とその評価

稲村　雄、市吉　伸行、
六沢　一昭、中島　克人

May, 1989

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

マルチ PSI/V2 における MRB 方式を用いた最適化手法とその評価

稲村 雄　　市吉 伸行
六沢 一昭　　中島 克人
(財) 新世代コンピューター技術開発機構

概要

マルチ PSI/V2 は、並列ソフトウエアの研究および種々の実装技術のテストのために ICOT で開発された疎結合マルチプロセッサであり、その上には、並列論理型プログラミング言語 KL1(核言語第 1 版) の処理系が実装されている。マルチ PSI/V2 への KL1 処理系実装の試みの初期から、効率的なメモリ管理方式をどのようにして実現するかということが重要な課題の一つであることが認識されていた。その結果、新しく考案された多重参照ビット (MRB) 方式により、効率的な即時式局所ガーベジコレクションが実現された。更に、MRB 情報は、構造体要素の破壊的書き込み、高速のストリーム・マージャの実現、プロセッサ間データ管理の効率化等の最適化にも利用できることが明らかになった。この論文では、これらの最適化手法およびその有効性について述べる。

1　序論

マルチ PSI/V2[6, 7] は、第 5 世代コンピュータシステム (FGCS) プロジェクトで開発された疎結合マルチプロセッサである。その開発の目的は、(1) 並列推論マシン PIM[4] の実現に先行して、大規模並列論理型ソフトウェアを実行可能な強力なシステムを提供すること、および (2)KL1 実装の上での種々のテクニックをテストすること、の二点であった。マルチ PSI/V2 上での並列推論マシンオペレーティングシステム (PIMOS) [2] 初期バージョンの開発はすでに終了しており、現在、その上でいくつかの中規模プログラムが動作している。

マルチ PSI/V2 への KL1 実装方式の検討における主な留意点の一つは効率的なメモリ管理方式の実現であった。KL1 の実行はヒープをベースにしているためメモリの消費が非常に速く、無造作なメモリ管理方式の実装は、システムの全体性能を制限する要素となると予想されるからである。これを解決するために多重参照ビット (MRB) 方式[1] を導入し、性能の良い即時式局所ガーベジコレクションを実現することとした。更に、MRB 情報を用いることで、即時式ガーベジコレクション以外にもメモリ管理に関する種々の最適化の実現が可能であることが判明した。この論文では、これらの最適化手法、およびその性能測定の結果について述べる。性能測定結果によって、最適化手法の効果、および MRB 方式の適用範囲の広さを確認することができた。

2　MRB 方式

MRB は、全ての参照ポインタに付加される 1 ビットの情報であり、あるポインタの MRB がオフならば、そのポインタはそのデータを指す唯一つのポインタであることが保証される。MRB がオフのポインタを使い終わったとき、すなわち参照先のデータを読んでしまい、それ以後その

ポインタが必要でなくなったとき、あるいは単にそのポインタが捨てられたとき、データの占めていたメモリ領域は、プログラムからはアクセス不能となっているため回収することが可能である。

変数に限り、最大2本のMRB オフのポインタが同じデータを参照することが許される。何故ならば、変数はデータの生成プロセスと消費プロセスとで共有されるのが普通であり、この場合にMRBがオンになってしまうのではガーベジコレクションの効率が落ちることになるからである。ポインタの1つから変数に具体値がバインドされ、残りのポインタからその値が読まれれば、変数セルおよび、具体化されたデータを回収することができる。

ポインタの消費や複製は、コンパイル時に判断することができる。コンパイラは、それに従ってゴミとなったデータを再利用するための命令や、ポインタのMRBをオンにする命令を発行する。MRB方式のメリットの1つは、参照データが回収されるときを除いて、データ自身へのアクセスが必要ないということである。PIMのような共有メモリ型マルチプロセッサではメモリのアクセス競合をできるかぎり抑えなくてはならないため、この事実は一層重要であると言える。

記号処理プログラムでは、データの多くが単一参照であることが知られている。KL1もこの点に関しては例外でなく、ベンチマークプログラムによる測定結果から、40%～90%のガーベジデータがMRB方式によって即時的に回収されることが分かっている。

MRB方式の詳細については[1]を参照のこと。

3 MRBによる最適化

3.1 構造体への破壊的書き込み

KL1のような論理型プログラミング言語では、未定義変数にのみ値の代入が可能であり、一旦、具体化されたデータを変更することはできない。そのため、幾つかの要素を除いては既に存在している構造体と同じであるような構造体データが必要なときは、元の構造体の要素をコピーして新しい構造体を作らなければならない。しかし、MRB方式を使えば他にその構造体を参照しているポインタがあるかどうかが分かるので、単一参照である場合には幾つかの要素を破壊的に書き換えることによって構造体の再利用を行なうことが可能である。

以下は、MRBがオフである場合に構造体の再利用が行なえる例である。

```
foo1([X1|X2] :- true | bar([Y1|Y2]), ...
foo2([X1|X2] :- true | bar([X1|Y2]), ...
```

最初の例ではConsセルが再利用され(構造体フレームの再利用)、次の例ではConsセルおよびリストのCAR部が再利用される(構造体要素の再利用)。

3.2 PE間のデータ管理(外部参照)

外部参照ポインタとは、他のPE内のデータオブジェクトを参照するもので、ゴールをPE間で分散した結果として生成される。ゴールを送信するPEはその引数であるデータオブジェクトを輸出し、ゴールを受信するPEはそれを輸入する。局所ガーベジコレクションを可能にするために、外部参照ポインタは輸入表エントリおよび輸出表エントリと呼ばれる二つの間接レコードを通してデータオブジェクトを指す[5]。一つのデータオブジェクトが重複して輸出表エントリもしくは輸入表エントリに登録されるとPE間メッセージの数が増加する可能性があり、好ましくない。これを避けるためにハッシングを用いた単一登録方式が採用されている[6]。

```
merge([], In2, Out) :- true | Out = In2.
merge(In1, [], Out) :- true | Out = In1.
merge([X|In1], In2, Out) :- true | Out = [X|Out2], merger(In1,In2,Out2).
merge(In1, [X|In2], Out) :- true | Out = [X|Out2], merger(In1,In2,Out2).
```

図 1: 述語 "merge/3" の定義

また、白輸出表および白輸入表と呼ばれる、より単純な構造をもった輸入表および輸出表も導入され、MRB がオフであるデータの輸出入に使われている[6]。白輸出入では、ほとんどの場合データ・オブジェクトを指す唯一つのポインタが輸出されることになり、輸出する PE 内部にはそのオブジェクトへの参照はなくなるため、ハッシングによる単一登録方式を用いる必要はない。白輸出入表を用いる場合、輸出および輸入の手続きは、非常に単純になっている。

3.3 ストリーム・マージャ

プロセス間のストリームによる通信はコミッテッドチョイス型言語で非常に良く使われるプログラミング手法である。あるプロセスが不特定数のプロセスからメッセージを受け取る場合、送信側のプロセスのメッセージストリームは受信側のプロセスの一個のストリームにマージされることになる。これは、オブジェクト指向型のプログラムでオブジェクトへの参照 (オブジェクトを表現するプロセスへの入力ストリーム) が複製される可能性のある場合に一般的に使われる手法である。

KL1 で 2 入力のストリーム・マージャ・プロセスを図 1 に示すように、定義することができる。

このプロセスによって、二つの入力ストリームの要素はマージされて出力ストリームに送られる。しかし、このようなストリームのマージ操作の実現方法には次のような二つの大きな欠点がある。

- ストリームの一要素入力毎に、"merge/3" プロセスの中断・再開という手間のかかる処理が必要となる。
- 入力ストリームの数が増えるごとに、マージ処理のコストも増える。

この問題に対して、述語を表すための特殊な構造体を使うという巧妙な方法が提案された[8]。これは非常に一般的なもので、マージャ・プロセスだけでなくガードゴールのない述語には全て適用できるような方法であった。マルチ PSI/V2 において、われわれはマージャ・プロセスにのみ特化された、より単純な方法を採用した。

この方法は、マージャ・プロセスがその具体化を待っている変数とマージャ・プロセス自身のために特別な表現 (マージャ・フック変数: MHV およびマージャ・レコード: MR) を用意することで実現される。MHV は MR を指し、MR はマージャ・プロセスの出力変数に相当するセルを指す。MR はその他に、自分への入力ストリームの数を示す整数値を保持している。この値は、マージャ・プロセスの終了を知るために用いられる (図 2)。

MHV とリスト・データのユニフィケーションが起こると、新しく生成されるリスト・データで出力変数が即時に具体化されることになるため、ユーザ定義によるマージャで問題となるマージャ・プロセス自身の中断・再開によるオーバヘッドを取り除くことができる (図 3)。

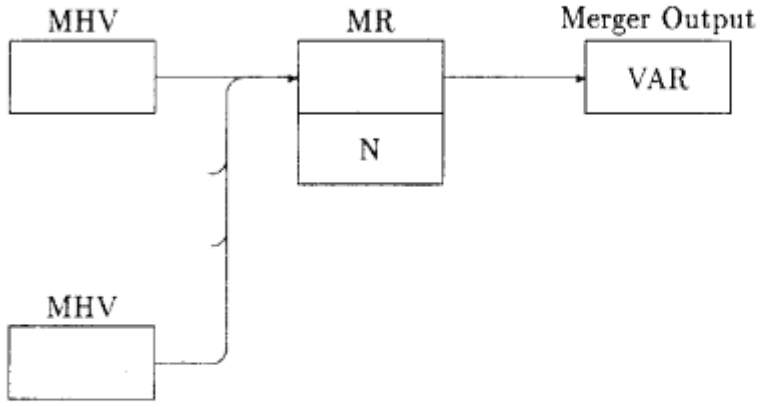

図 2: 組込みマージャ・プロセスの表現形式

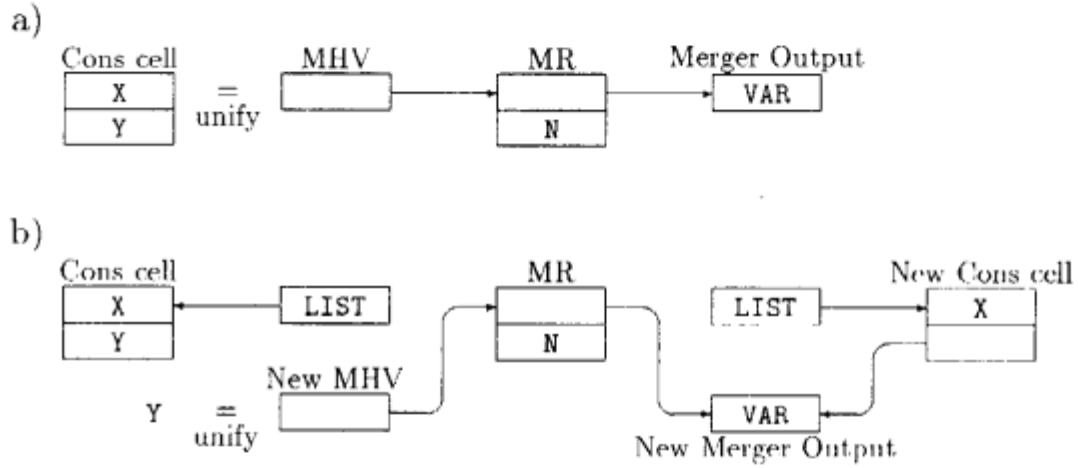

図 3: MHV とリスト・データとのユニフィケーション

MRB 情報を用いると更なる最適化が期待できる。入力される Cons セルが単一参照だった場合、それを新しい出力 Cons セルとして再利用できるため、新しいリスト・データを作る手間を省略することができる。

入力ストリーム数が動的に増加可能であることも、効率的なマージャ・プロセスの実現のためには重要である。 KL1 による 2 入力マージャを用いる場合、これは多段のマージャ・プロセスという非常にコストのかかる処理によって実現されることになる。われわれの採用した方法は、 MHV に関するユニフィケーションをマージャ・プロセスのもともとの定義(図 1) から拡張することである。 MHV とベクタ・データとのユニフィケーションを可能とし、これをマージャ・プロセスへの入力ストリームの追加要求であると解釈する。ベクタが MHV とユニファイされると、ベクタの要素の数だけの入力ストリーム (MHV) がマージャ・プロセスへ追加され、ベクタのそれぞれの要素は新しい入力ストリームのそれぞれにユニファイされる。

このようにユニフィケーションを拡張することにより、マージャ・プロセスは論理的には、図 4に示すような、無限個の節からなり、2 個から無限個の引数を持つような述語であるとみえる。

```
merge([],     Out) :- true | Out = [].
merge([A|In],Out) :- true | Out = [A|NewOut], merge(I,NewOut).
merge([],     In,     Out) :- true | merge(In,Out).
merge(In,     [],     Out) :- true | merge(In,Out).
merge([A|I1],I2,     Out) :- true | Out = [A|NewOut], merge(I1,I2,NewOut).
merge(I1,     [A|I2],Out) :- true | Out = [A|NewOut], merge(I1,I2,NewOut).
merge([A|I1],I2,     I3,     Out) :- true |
                                     Out = [A|NewOut], merge(I1,I2,I3,NewOut).
merge(I1,     [A|I2],I3,     Out) :- true |
                                     Out = [A|NewOut], merge(I1,I2,I3,NewOut).
merge(I1,     I2,     [A|I3],Out) :- true |
                                     Out = [A|NewOut], merge(I1,I2,I3,NewOut).
                  :
                  :
merge({},      Out) :- true | Out = [].
merge({X},     Out) :- true | merge(X,Out).
merge({X,Y},   Out) :- true | merge(X,Y,Out).
merge({X,Y,Z},Out) :- true | merge(X,Y,Z,Out).
                  :
                  :
merge({},    In,    Out) :- true | merge(In,Out).
merge(In,    {},    Out) :- true | merge(In,Out).
merge({X},   In,    Out) :- true | merge(X,In,Out).
merge(In,    {X},   Out) :- true | merge(In,X,Out).
merge({X,Y},In,    Out) :- true | merge(X,Y,In,Out).
merge(In,    {X,Y},Out) :- true | merge(In,X,Y,Out).
                  :
                  :
```

図 1: 組込みストリーム・マージャの論理的な定義

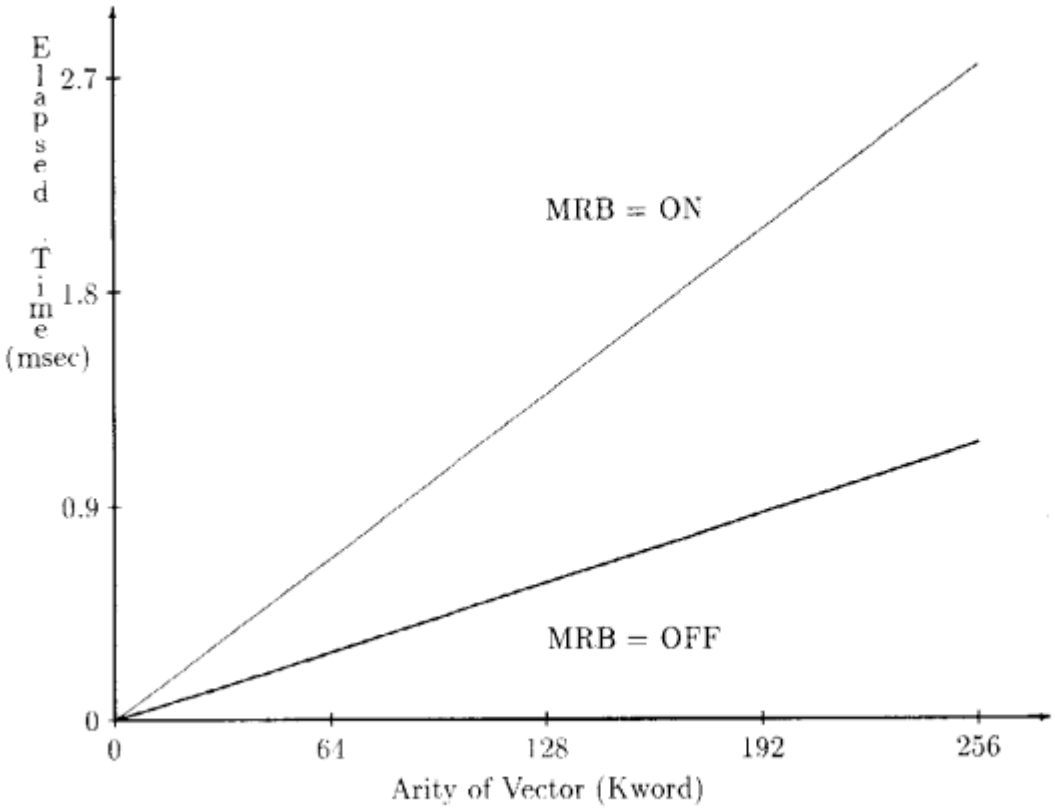

図 5: ベクタ更新処理のコスト (全要素書き換え)

処理系の実装においては複雑化を避けるために、2引数の述語 "merge/2" のみが組込み述語として用意され、複数の入力ストリームを持つマージャ・プロセスは、MHV とベクタのユニフィケーションのみにより実現されることとした。

4 最適化の評価

この章では、前章で述べた最適化手法の性能測定結果を示す。それぞれの項目で、MRB がオンの場合とオフの場合について処理時間を測定した。前者は、MRB 最適化を行っていない処理系での場合に相当すると考えられるため、これらの時間の差が最適化の効果を表していると言える。

4.1 構造体の再利用

4.1.1 ベクタの更新

MRB の効果を確かめるために、色々な大きさのベクタの書き換えのコストを測定した。この測定プログラムは、与えられたベクタの要素全てを書き換えるものであり、MRB がオンである場合とオフである場合それぞれについて処理に要した時間を測定した。

ミュータブルアレイ方式[3] を採用したことにより、MRB がオンである場合でも更新操作のためにベクタ全体をコピーする必要はなく、ベクタ1要素の書き換えのコストは常に一定とすることができた。しかし、それでも MRB がオンのときのベクタの書き換えのコストは、MRB がオフのときの約 2.5 倍になった。

4.1.2 簡単なベンチマークプログラムにおける効果

構造体の再利用の効果を、いくつかの簡単なベンチマークプログラム (*Append, Quick-sort, Prime number generator*) を用いて測定した。それぞれのプログラムを、以下のような3通りの方法でコンパイルして測定を行った。

1. 構造体の再利用を行なわない
2. 構造体フレームを再利用
3. 構造体の要素まで再利用

表 1に測定結果を示す。

表 1: 構造体再利用による性能改善

	No reuse (KRPS)	Frame reuse (KRPS)	Element reuse (KRPS)
Append	110	128	146
Qsort	95.8	108	113
Primes	55.9	59.8	61.2

RPS: reductions per second

これらのベンチマークによる測定結果から、構造体要素の再利用により、 10 ～ 30% 程度の性能改善が得られることが分かった。ここで得られた改善は主として、実行中の命令ステップ数の減少によるものである。しかし、構造体の再利用にはメモリアクセス頻度を減少させる効果もあり、このことが、特に PIM[4] のようなメモリ共用マシンの場合、性能向上に大きく貢献すると期待できる。

4.2 ストリーム・マージャ

組込みマージャによる性能向上の度合いを見るために、組込みマージャおよび KL1 で定義したマージャ (図 1) を用いた場合でマージ処理に要するコストの違いを測定した。 MRB オン / オフのリストを入力した場合に 1 要素のマージに要するコストを図 6に示す。

MRB がオンの場合の KL1 マージャと組込みマージャとのコスト差がマージャ・プロセスのために既述の表現形式を用意したことによる効果と考えることができる。この組込みマージャによって、性能は 4 倍に向上した。 MRB のオン / オフによる組込みマージャ処理のコスト差が、MRB の効果であるといえる。 MRB がオフの場合、実行速度はオンの場合の 2 倍である。

組込みマージャによる性能の改善だけでも十分効果的であるように見えるが、 MRB による更なるスピードアップも、やはり必要だと考えられる。ストリーム・マージ操作というのは KL1 のプログラムでは非常に多く使われるものであり、そのスピードアップはシステムの全体性能に大きな効果があると言えるからである。

4.3 白と黒の輸出の比較

白輸出入表導入の効果を確かめるために、ゴールの分配と、データ転送のコストを、白輸出入表と黒輸出入表を使用した場合それぞれについて測定した。

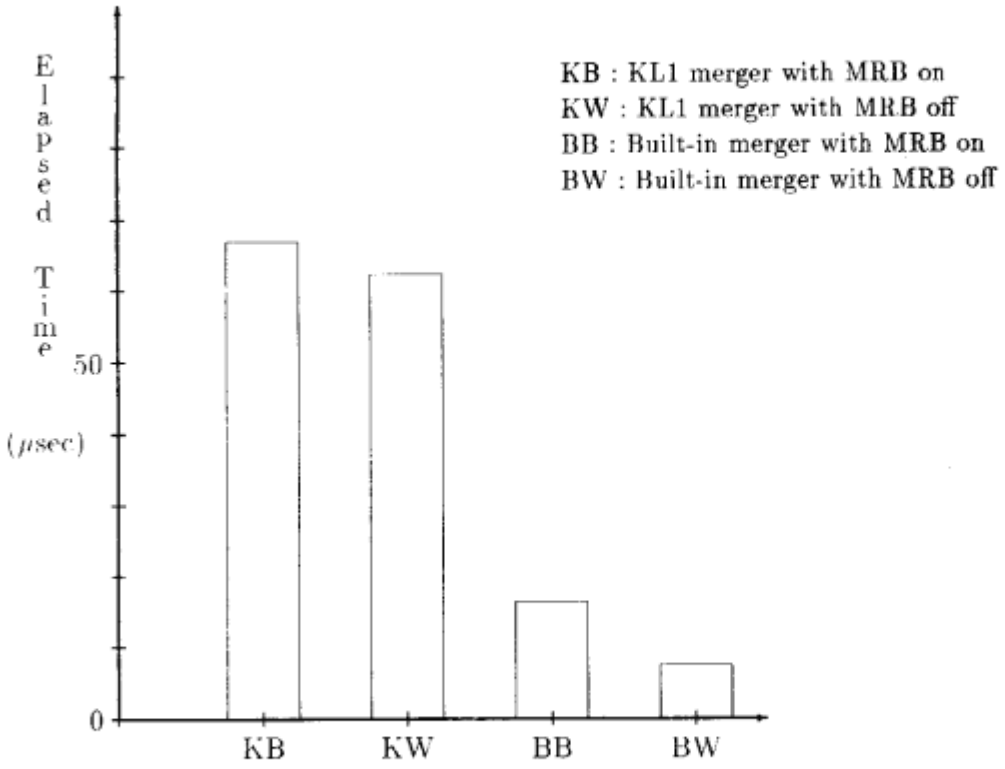

図 6: マージ操作の性能

4.3.1 ゴール分配のコスト

ゴール分配のコストを、いくつかの場合について測定した。ゴールの引数としては、以下の3種類を選んだ。

1. MRB がオフの変数。
2. MRB がオンの変数。
3. アトミック・データ。

1の引数は白輸出表エントリを用いて輸出される。2の引数は黒輸出表エントリを用いて輸出される。3の引数は、ゴールの引数として直接送られる。それぞれの場合についてゴール・リダクション一回に要する処理時間を図 7に示す。

アトミック・データを輸出するのに必要なコストを、ゴール送出操作におけるコンスタント・オーバヘッドであると考えれば、黒輸出におけるコストは、白輸出におけるコストの約3倍である。但し、疎結合マルチプロセッサであるため、図 7からも分かる通り、ゴール送出操作に必要なコンスタント・オーバヘッドは非常に大きく (約 300μsec)、投げられるゴールの引数が少ない場合には、白輸出と黒輸出との差はほとんど問題にならない。ゴールの送出においては、白輸出表は引数の多いゴールを投げるための最適化であると言える。

4.3.2 データ転送のコスト

外部参照されているデータは、PE 間メッセージ `read` と `answer_value` によって転送される。白輸出されたデータと黒輸出されたデータとをこのプロトコルによって転送する場合のコストの

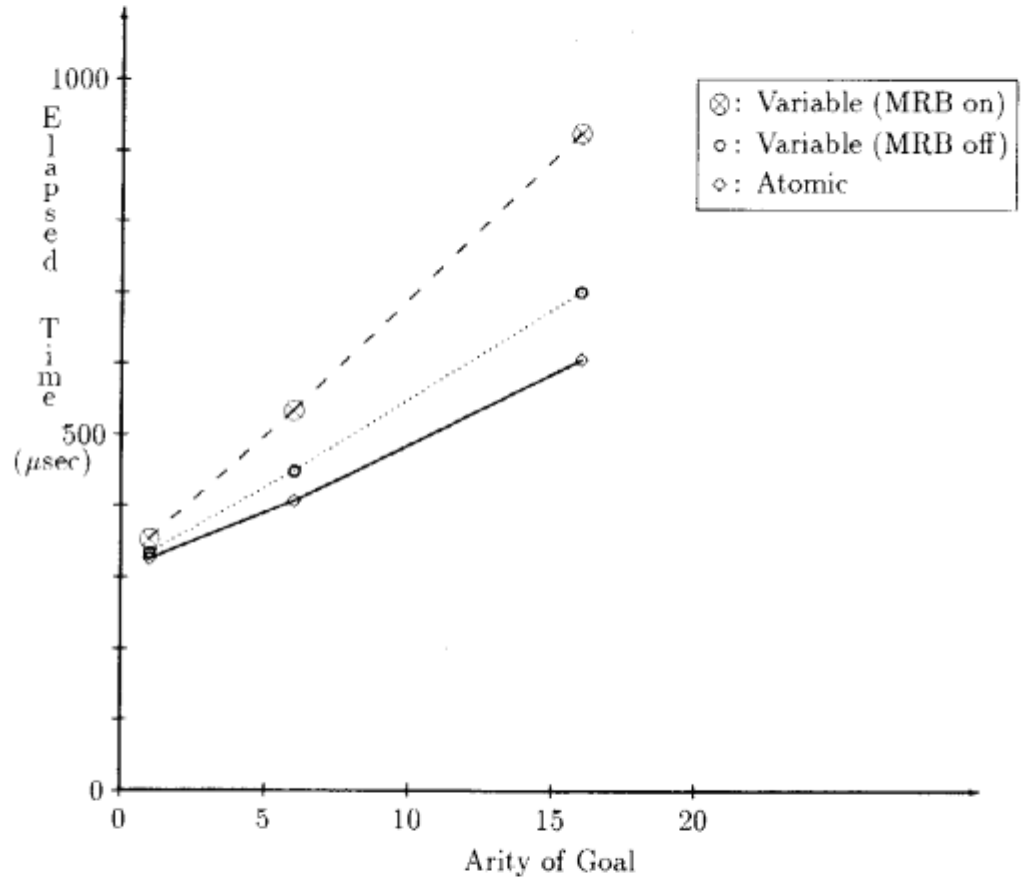

図 7: 1 ゴール・リダクションの所要時間

表 2: Cost of one cons cell transfer

White export	Black export
240 μsec	300 μsec

差を測定した。転送されるデータとしては種々のサイズのリストを用いている。本処理系では要求駆動による PE 間コピー方式[6] を採用しているため、リスト・データが対象である場合、一時に転送されるのは 1Cons セルであり、その CDR 部は外部参照として表現される (CDR 部がリストもしくは変数である場合)。一個の Cons セルの転送に要する時間を表 2に示す。

一個の Cons セルを転送する場合でも、白輸出によって通信コストは 20% 程度減少する。これは、readとanswer_valueによるデータ転送処理におけるコンスタント・オーバヘッドがゴール送出処理におけるそれに比べて十分小さいためであり、データ転送においては白輸出表の効果は大きいと言える。

5 結び

この論文では、構造体の再利用、一定時間で処理できる組込みストリームマージャ、白輸出などの、MRB 情報を用いたいくつかの最適化手法について述べた。これらの最適化手法の効果をそれぞれ測定したところ、10% から 200% を越える性能改善が得られることが分かった。

これにより、MRB 方式は即時的局所ガーベジコレクションを可能にするだけでなく、メモリ管理に関する種々の処理を高速化できることを示した。

個々の最適化に関する測定結果から、それぞれの効果は確かめられたが、これらがシステムの全体性能へ与える影響は、各処理が実際のアプリケーション実行時にどのような頻度で発生するかを測定することで評価されなければならない。今後は、種々のプログラミングスタイルで書かれた実用的な大きさのプログラムを用いて上記の測定を行なう予定である。

謝辞

この研究の機会を与えて下さった、渕 一博 ICOT 研究所長、内田 俊一 同第 4 研究室長に感謝します。また、我々と一緒にマルチ PSI/V2 上の KL1 システムの設計、実装、評価をして下さった ICOT ならびに協力会社の研究員の方々に感謝します。

参考文献

[1] T. Chikayama and Y. Kimura. Multiple Reference Management in Flat GHC. In *Proceedings of the Fourth International Conference on Logic Programming*, 1987.

[2] T. Chikayama, H. Sato and T. Miyazaki. Overview of the Parallel Inference Machine Operating System (PIMOS). In *Proceedings of the International Conference on Fifth Generation Computer Systems*, ICOT, Tokyo, 1988.

[3] L. H. Eriksson and M. Rayner. Incorporating mutable arrays into logic programming. In *Proceedings of the Fourth International Conference on Logic Programming*, Upssala, 1984.

[4] A. Goto, M. Sato, K. Nakajima, K. Taki and A. Matsumoto. Overview of the Parallel Inference Machine Architecture (PIM). In *Proceedings of the International Conference on Fifth Generation Computer Systems*, ICOT, Tokyo, 1988.

[5] N. Ichiyoshi, K. Rokusawa, K. Nakajima and Y. Inamura. A New External Reference Management and Distributed Unification for KL1. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, ICOT, Tokyo, 1988.

[6] K. Nakajima, Y. Inamura, N. Ichiyoshi, K. Rokusawa, T. Chikayama. Distributed Implementation of KL1 on the Multi-PSI/V2. To appear in *Proceedings of the Sixth International Conference on Logic Programming*, 1989.

[7] K. Taki. The parallel software research and development tool: Multi-PSI system. Programming of Future Generation Computers, Elsevier Science Publishers B.V. (North-Holland), 1988.

[8] K. Ueda, and T. Chikayama. Efficient stream/array processing in logic programming language. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, ICOT, Tokyo, 1984.