ICOT Technical Report: TR-412

TR-412

並列推論マシンPIMにおける効率的構造体処理方式
—参照数管理と長男優先方式—

今井　明、木村　康則、稲村　雄、後藤　厚宏

July, 1988

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

並列推論マシンPIMにおける効率的構造体処理方式

-参照数管理と長男優先方式-

Efficient Structure Handling Method for Parallel Inference Machine PIM

- Reference Management and *Chonan* Scheme -

今井 明* 木村 康則 稲村 雄 後藤 厚宏

Akira IMAI Yasunori KIMURA Yu INAMURA Atsuhiro GOTO

新世代コンピュータ技術開発機構†

Institute for New Generation Computer Technology (ICOT)

1 はじめに

ICOTでは第五世代コンピュータプロジェクトの一環として，並列推論マシンPIMの開発を行っている．PIMの核言語はKL1 (Kernel Language 1)と呼ばれるflat-GHC [5]に基づく並列論理型言語である．KL1はメモリセルの破壊的書き換えといった副作用を許さない言語であり，並列処理システムの記述に不可欠な同期・通信などを自然に記述できる．

本稿では，このKL1の実装にあたって，KL1がサポートしているデータタイプであるベクタと呼ぶ構造体の処理方式について述べる．

まず最初に，ベクタ処理を単純に実装した時の問題点を指摘する．次に，この問題を回避するために

- 参照数管理による最適化

が非常に効果的であることを示す．

更に，この最適化が施せないような状況下では，従来提案されているMutable Array[3]の諸方式についての検討を行い，これらの問題点を指摘したうえで，

- 長男優先方式による更新方式

を提案し，その評価を行うことにより有効性を示す．

2 ベクタの更新時に生じる問題

KL1は副作用を持たない言語であるため，構造体の1要素を更新する場合，更新前後の構造体は全く異なる構造体として扱われる．そのため，これを単純に実現しようとすると，更新前の構造体と1要素しか異ならない構造体を新たに作る必要がある．これは，更新後の構造体を作るために新たな構造体の領域を確保して，変更のある1要素を除く全ての要素をもとの構造体からコピーする必要があるため，1要素を変更する毎に，構造体の要素数に比例したメモリアクセスのコストが必要である．

更に，更新前の構造体と1要素しか変わらない構造体が次々とメモリ空間に割り付けられてしまうため，メモリ領域を単調かつ急速に消費してしまうという問題をも引き起こしてしまう．

そこで，以下KL1がサポートしているベクタと呼ばれる構造体の実装にあたって，この問題を解決する方法についての考察を述べる．

3 参照数管理による最適化

3.1 単一参照時の再利用による最適化

ベクタを更新する場合，更新前のベクタに対して他から参照されていないことが保証されていれば，操作は簡略化できる．すなわちこの場合，更新後のベクタとして新たなベクタ領域を確保する代わりに，更新前のベクタを回収し即座に再利用することが可能なため，変更のない要素は流用することができ，変更要素のみを破壊的に上書きするだけで更新操作は完了する．この方法では，無変更要素に対するメモリアクセスは一切不要である．

3.2 MRB

KL1の実装にあたっては，既に多重参照ビット(Multiple Reference Bit: 以後MRBと呼ぶ)によるインクリメンタルGC(Garbage Collection)を提案し [2, 8]，変数セル，リストセルに関してはシミュレーションによって，その有効性を確認している [6, 7]．

MRB方式は，KL1において大多数のデータオブジェクトが単一参照であるという特徴を利用し，あるデータの

*JUNET: imai@icot.junet

†108 東京都 港区 三田 1 丁目 4-28 三田国際ビルヂング 21F., Phone: 03(456)3193 Telex:ICOT J32964

参照数をオブジェクト側ではなくポインタ側に付随した1ビットの情報で管理するというものである．MRB方式によって単一参照であることを保証されたデータを消費したプロセスは，その時点でデータオブジェクトを回収して，再利用可能な状態にすることができる．これがMRB-GCの原理である．

具体化された構造体へのポインタのMRBビットの値は○(白いと呼ぶ)または●(黒いと呼ぶ)で表現され，それぞれ次のような意味を持つ[1]．

○: 参照されるデータへのポインタはこれ一本である．(単一参照)

●: 参照されるデータへのポインタが他にもあるかもしれない．(多重参照)

MRB管理の下では，前節の最適化が適用できるかどうかは，更新前のベクタのMRBの値を読むことで即座に判定できる．ベクタをMRB管理するのためのコストは，MRBがポインタ側に付随した情報であるため，既に評価済みの変数セル，リストセルの場合と同等と考えられ，低コストで実現可能と考えられる．

4 Mutable Array

4.1 Mutable Array の導入

前章で述べたMRB管理による最適化は，MRBが●で，多重参照かもしれないベクタを更新する場合には適用できない．しかし，このような場合であっても，更新前のベクタと同じサイズの領域を割り当てて無変更部分のコピーを行うことなしに新しいベクタを作るMutable Arrayと呼ばれる方法が提案されている[3]．この方式では，更新によって生じた複数のベクタは，デスクリプタと呼ばれる更新前後の差分を記録したセルのチェーンと，複数のベクタによって共有される1つのベクタ本体で表現される．

すなわち，図1の例では，

	要素0	要素1	要素2	要素3	
ベクタA	{	a,	b,	c,	d }
ベクタB	{	a,	b,	e,	d }
ベクタC	{	a,	f,	e,	d }

をそれぞれ意味している．

4.2 従来の Mutable Array 更新方式

Mutable Arrayの更新の方式として，Old Real Schemeおよび，New Real Schemeが提案されている[3]．なお，以下説明のために用いる図2，図3，図5は，

(1) 第i要素がa，第j要素がbのベクタV0が最初に存在している．

[1]未定義変数の場合は，具体化するためとそれを読み出すための2本の参照ポインタがあるのが通常の場合であるため，MRBの扱いが少し異なるが，ここでは省略する．

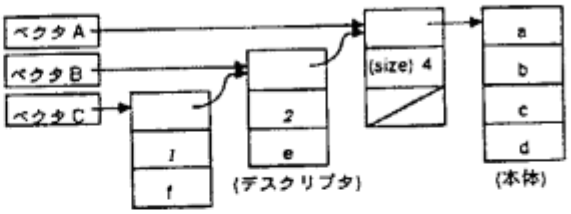

図 1: Mutable Array の例

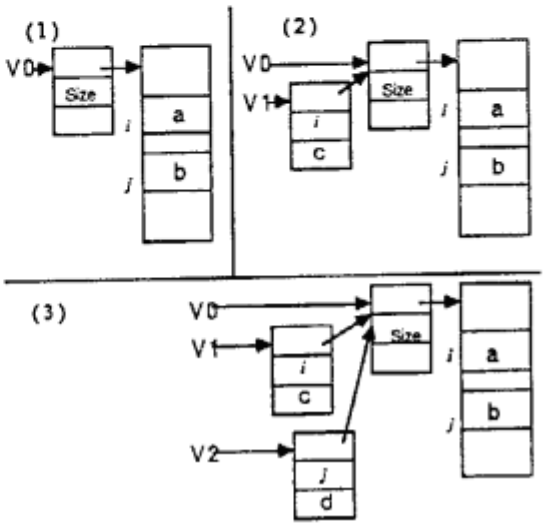

図 2: Old Real Schemeによる更新

(2) V0の第i要素をcに変更したものをベクタV1とする．

(3) 次に，V0の第j要素をdに変更したものをベクタV2とする．

とした時の例を示したものである．

4.2.1 Old Real Scheme

更新前のベクタに対する更新後のベクタの差分をデスクリプタに記録する方法である(図2参照)．更新前のベクタには，何ら変更がない．

- 更新を進めると，更新後のベクタは長いデスクリプタのチェーンを持つようになる．(更新前の方が参照コストが低い)
- 更新時に必要なデスクリプタの数は，常に1である．

4.2.2 New Real Scheme

更新後のベクタに対する更新前のベクタの差分をデスクリプタに記録する方法である(図3参照)．更新前のベクタには変化があるが，ベクタの値そのものは変化がない．

New Real Schemeの特徴として，

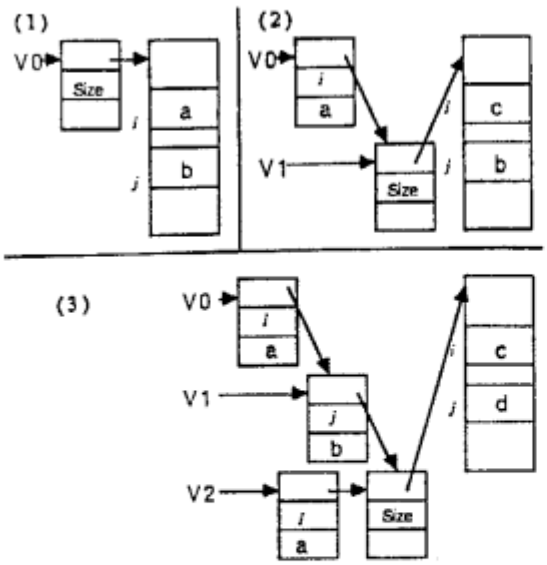

図 3: New Real Schemeによる更新

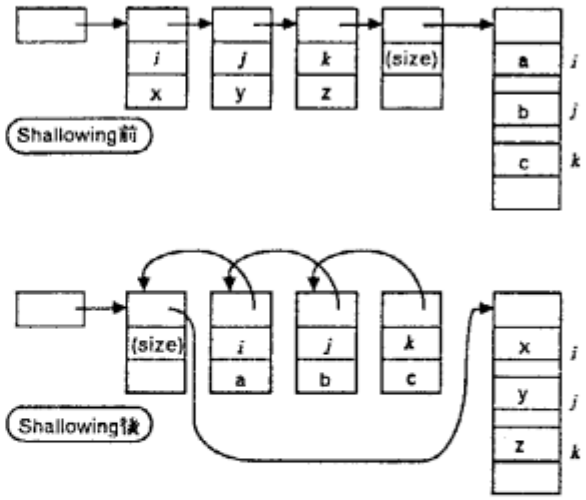

図 4: Shallowing

- 更新を進めると，更新前のベクタのデスクリプタのチェーンが伸びる．(更新後の方が参照コストが低い)
- 全ての要素が本体に格納されているベクタ (リアルなベクタと呼ぶ) でないものを更新する場合，デスクリプタが複数個必要である．
- リアルなベクタでないものを一度でも更新すると，それ以降本体を共有するベクタの更新にはすべて複数個のデスクリプタが必要である．

4.3 Shallowing

更新方式とは別に，Mutable Array の操作としてShallowingがある [1]．これは，デスクリプタのチェーンが長くなって，参照コストが高くなった場合に，新たなデスクリプタを消費することなく，また他のベクタの一貫性を保ったままリアルなベクタにする操作であり，経由していたデスクリプタのポインタを逆転させることによって実現される (図4参照)．

更新時に，更新前のベクタに Shallowing 操作を行ってリアルなベクタにした後に，New Real Schemeで更新を行うという方式も考えられる．この方法では，New Real Schemeでリアルでないベクタの更新を行う際の問題を回避することができる．

5 長男優先方式

5.1 従来の更新方式の問題点

前節で述べたように，従来提案されていたベクタ更新のアルゴリズムにはOld Real SchemeおよびNew Real Schemeがあり，これらは更新 / 参照の頻度が高いベクタの性質によって，ベクタ操作全体のコストは変化する．

このように，一般的には最新のバージョンの更新が多いと考えられることが[3]においても指摘されており [2]，このような条件の下ではNew Real Schemeの方がOld Real Schemeよりも効率的であると考えられる．しかしながら，New Real Schemeでは一度リアルでないものを更新した場合，その後の処理の効率が落ちてしまう．従来の提案は逐次処理系への実装を考慮したもので，ベクタの生成順序をプログラマが意識することによって，この欠点を回避することができる．しかし，並列処理系ではスケジューリングによって，ベクタの生成順序が変化する．KL1 においてはサスペンド (中断) 処理によって，子が親より先に生成されることはないが，New Real Schemeの欠点をプログラマが回避することができない．

そこで，長男優先方式(*Chonan* Scheme) を提案し，この問題を解決することを図る．

5.2 長男優先方式

長男優先方式とは，リアルなベクタを更新する際にはNew Real Schemeを用いて，この方式の特長を受け継ぎ，New Real Schemeによる更新が不利になるリアルでないベクタを更新する際にはOld Real Schemeの方法で更新する方式である (図5参照)．この方式について，以下考察する．

まず，ベクタの更新を行う場合，更新前のベクタに対し更新後のベクタを子供と考える．ある親に対し，最初に生成された子から順に，第1子，第2子.... とする．この更新方式では，参照のコストが小さいリアルなベクタは，常に最初に割り当てられたベクタから見た第1子の系列で一番世代の若いベクタが格納されていることになる．参照のコ

[2]他の言語 (例えば FORTRAN や C) などで配列の更新を行うといえば，最新のバージョンを更新しそれを新たな最新のバージョンにすることと考えられ，特別な配慮 (例えばコピー) をしておかないと古いバージョンの参照は不可能である．

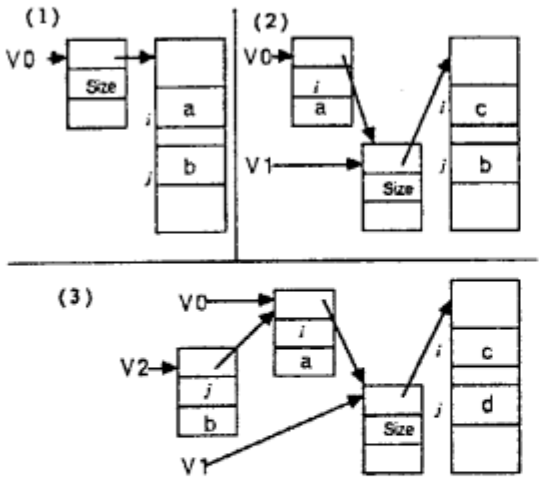

図 5: 長男優先方式による更新

ストは，この第1子の系列の一番若いベクタに対して最小になるということが長男優先方式の名前の由来である．

長男優先方式による更新の特徴として，

- New Real Schemeでリアルになる場合，長男優先方式でも常にリアルである．
- 本体を共有するベクタの中に，必ず一つリアルなものが存在する．
- 更新の際に必要なデスクリプタ数は，常に1である．

が挙げられる．

6 密結合並列マシン上への実装

6.1 PIM のハードウェア構成

ここで，現在我々が開発を行っている並列推論マシン PIM のハードウェア構成を簡単に説明する (図6参照)．

単体で 200-500KLIPS の要素プロセッサ (PE) 8-10 台を共有バス / 共有メモリで密結合したクラスタを，ネットワークで疎結合する．各プロセッサは高速のコヒーレントキャッシュメモリを持っている．同期のための排他制御は，キャッシュのブロック状態を利用したロックによって低コストで実現できる[4]．

6.2 ベクタ更新時の排他制御

PIM のクラスタ内部のような密結合並列マシン上では，排他制御を含めた処理アルゴリズムを考えなければならない．そこで，前章までに述べた処理方式で必要な排他制御について考察する．

まず，Old Real Schemeでは，更新前のベクタが持っているメモリ対して書き込みが不要であるため，更新時の排他制御は一切不要である．

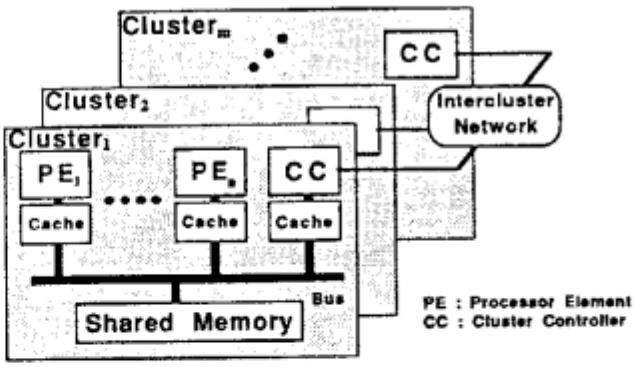

図 6: 並列推論マシン PIM のハードウェア構成

次に，New Real Schemeでは，更新前にベクタ本体を指していたデスクリプタ，及びベクタ本体に対する排他的書き込みの必要があるため，このデスクリプタに対し書き込み時にロックが必要となる．

長男優先方式は以上2方式の組み合わせであるから，リアルなベクタの更新を行う場合には，ベクタ本体を指すデスクリプタの書き込みの際にこのデスクリプタへのロックが必要となる．

Shallowing 操作では，チェーンを構成する全てのデスクリプタに対し，ポインタ付け換えの操作が必要となる．このため，ポインタを逆転しながら，デスクリプタにロックを掛けてゆき，本体に到達したのちに逆転されたポインタを辿りながら，このロックを外してやる操作が必要である．この結果，Shallowing 前に最初に指されていたデスクリプタへのロックは，Shallowing 操作開始時から終了時まで掛けておく必要があり，1つのバージョンの Shallowing 操作中は他のバージョンの更新 / 参照が原則的に不可能となる[3]．バージョン数が増え，長いチェーンを持つベクタを Shallowing する際は，ロック操作の回数も増えるだけでなく，並列処理を制限するものとなってしまうため，密結合並列システムでは Shallowing は不向きであると考えられる．

7 Mutable Array 更新方式の評価

7.1 評価条件

この章では，前章までに述べた Mutable Array の更新方式の評価を行う．ここで，Mutable Array は，デスクリプタを3ワードとし，ベクタサイズを本体を指すデスクリプタの第2ワードに記録することを仮定している．このような条件の下での更新 / 参照の際に必要最低限のメモリアクセス数を，計算によって求める[4]．

[3]他のバージョンがこのチェーン以外の枝を持ち，その枝に参照しようとするインデックスがある場合は参照可能である．

[4]なお PIM のクラスタ内の場合，並列キャッシュ機構によって，あるアドレスに対するメモリアクセスのコストは，アクセスを行う PE のキャッシュ上へのエントリーの有 / 無，他の PE のキャッシュとの共有 / 非共有などの条件によって異なるが，ここでは考慮していない．

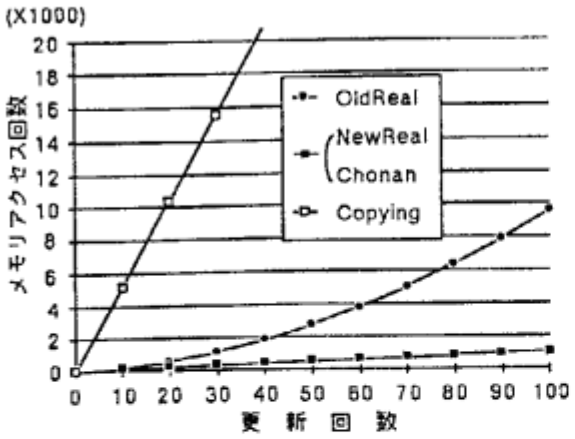

図 7: 例 1 での更新

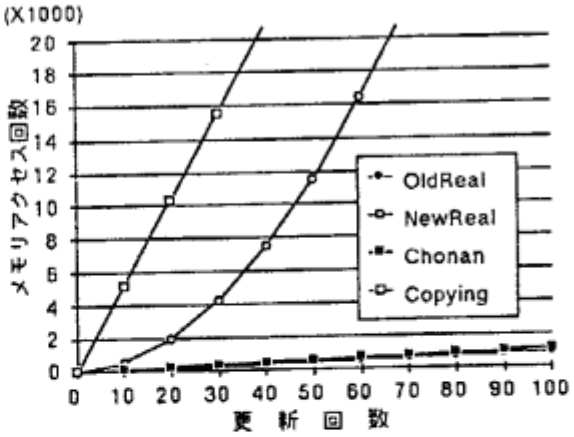

図 8: 例 2 での更新

7.2 ベクタ参照のコスト

リアルなベクタの場合，参照時のメモリアクセス数は

R_0 = デスクリプタ第 1 ワード +
(本体のアドレスの読出し)
デスクリプタ第 2 ワード +
(インデックスあふれ検査のための読出し)
本体 1 ワード
(要素の読み出し)

= 3

である．

チェーンが 1 つ伸びた状態 (これをチェーンの長さ 1 とする) では，最初に読んだデスクリプタに読みたいインデックスがある確率は，ベクタサイズを S とすると，

1/S

であるから，メモリアクセス数の平均は，

$$R_1 = (1/S * 2 + (1 - 1/S)(2 + 2)) + 1$$

である．

チェーンの長さを n とした時，参照の際のメモリアクセス数の平均は，

$$R_n = 1/S * 2 + (1 - 1/S) * ((R_{n-1} - 1) + 2) + 1$$

より，

$$R_n = 2(1 - S)(1 - 1/S)^n + (2S + 1)$$

となる．(これは，$2S + 1$ に収束する．)

7.3 ベクタ更新のコスト

KL1 のベクタ更新のための組込述語は，

```
set_vector_element(
        OldVect,   :(更新前のベクタ)
        Index,     :(更新を行う要素の位置)
        OldElm,    :(更新前のベクタの要素)
        NewElm,    :(更新後のベクタの要素)
        NewVect)   :(更新後のベクタ)
```

のように，更新前の要素を参照して返す仕様になっているこの述語を各処理方式で実現する時のメモリアクセス数を比較する．

- 例 1 長男のみの更新を n 回，1 回の更新につき 1 度，任意の 1 要素を参照

長男を New Real Scheme で更新する場合のメモリアクセス数は，10 である．この例では長男優先方式も同じである．

$$\begin{aligned}
U_{old(n)} &= \sum_{i=1}^{n}(R_i + 4) \\
&= -2S(1 - S)(1 - 1/S)^{n+1} + (2S + 5)n \\
&\quad +2S(1 - S)(1 - 1/S) \\
U_{new(n)} &= 10n \\
U_{chonan(n)} &= 10n \\
U_{copy(n)} &= (2S + 7)n
\end{aligned}$$

ベクタのサイズを 256 とした時のアクセス数 (積算) のグラフを，図7に示す．

- 例 2 生成された初期状態ベクタのみの更新を n 回，1 回の更新につき 1 度，任意の 1 要素を参照

$$\begin{aligned}
U'_{old(n)} &= (4 + R_1)n \\
&= (9 - 2/S)n \\
U'_{chonan(n)} &= (U_{new(1)} + R_0) + (4 + R_2)(n - 1) \\
&= 13 + (11 - 6/S + 2/S^2)(n - 1) \\
U'_{new(n)} &= (U_{new(1)} + R_0) + \sum_{i=2}^{n}((7i + 10) + R_{i-1}) \\
&= (7/2)n^2 + (15/2 + 2S)n \\
&\quad +2S(1 - S)(1 - (1 - 1/S)^n) \\
U'_{copy(n)} &= (2S + 7)n
\end{aligned}$$

ベクタのサイズを 256 とした時のアクセス数 (積算) のグラフを，図8に示す．

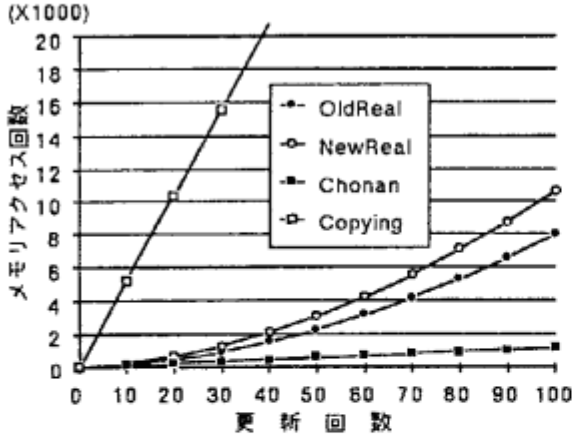

図 9: 例 3 での更新

- 例 3 長男系列の更新を 4 回，次男の更新を 1 回を繰り返す．参照の条件は前 2 例と同じ

ベクタのサイズを 256 とした時のアクセス数 (積算) のグラフを，図9に示す．(式は略)

7.4 評価結果のまとめ

以上の例より，長男優先方式は更新されるベクタの性質が変化しても，従来の更新方式のに比べ，平均して良好な値が得られることが示された．例 2 の場合はOld Real Scheme に比べて，更新回数 100 回の場合 20% 増となっているが，New Real Schemeは約 50 倍となっていることに比べると大幅な改善である．

最後に，あるベクタを別のプロセスで全く独立に更新する場合，フォークさせる時点でコピーを行ってから更新した方が，ボディを共有したまま長男優先方式で更新した場合に比べて効果的な場合があることを示す．これは，コピー後のベクタを単一参照にすることによって最適化ができる場合，コピー後のベクタの更新コストを大幅に下げることができるため特に有効であり，この最適化を含めた例を図10に示す．

8 おわりに

KL1 処理系でのベクタの処理方式として，参照数管理下で単一参照が明らかな場合の最適化の有効性を示した．さらに，多重参照となってこの最適化ができない場合でも，長男優先方式で更新すると，従来のOld Real SchemeやNew Real Schemeに比べて，メモリアクセスを削減できるケースが多いことを示した．

謝辞

日頃より貴重な御意見，御討論を頂く ICOT 第 4 研究室の西田健次研究員をはじめとする PIM グループ /Multi-

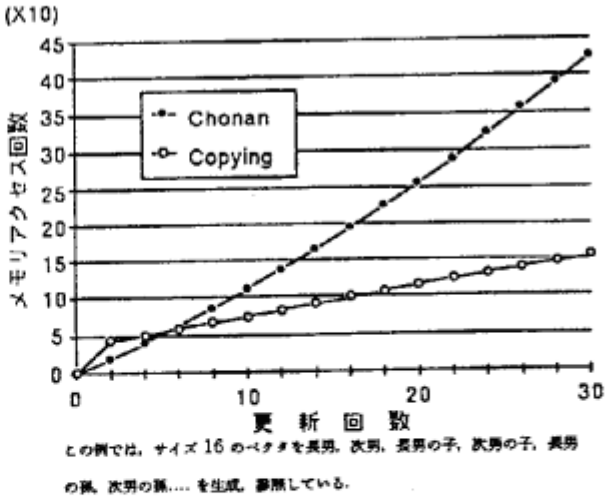

図 10: コピーが有効な例

PSI グループの方々に感謝します．また，本研究の機会を与えて頂いた渕一博研究所長，内田俊一第 4 研究室長に感謝します．

参考文献

[1] H. G. Baker. Shallow Bindings in Lisp 1.5. In *Communications of the ACM*, Vol.21,#7, 1978.

[2] T. Chikayama and Y. Kimura. Multiple Reference Management in Flat GHC. In *Proceedings of the Fourth International Conference on Logic Programming*, pages 276–293, 1987. (also in ICOT Technical Report, TR-248).

[3] L. H. Eriksson and M. Rayner. Incorporating Mutable Arrays into Logic Programming. In *Proceedings of the Second International Logic Programming Conference*, pages 101–114, June. 1984.

[4] A. Goto. Parallel Inference Machine Research in FGCS Project. In *US-Japan AI Symposium 87*, pages 21–36, Nov. 1987.

[5] K. Ueda. Introduction to Guarded Horn Clauses. TR 209, ICOT, 1986.

[6] 西田 他. MRBによる多重参照管理方式. 第 35 回情報処理全国大会 2Q-8, September 1987.

[7] 西田 他. KL1擬似並列処理系における実時間GC方式のキャッシュ特性の評価. 「コンピュータアーキテクチャシンポジウム」予稿集, pages 81–88, May 1988.

[8] 木村 他. KL1の多重参照ビットによるGC方式について. 「データフローワークショップ 1987」予稿集, pages 215–222, October 1987.