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†

木村康則††* 近山 隆††

本論文では，並列論理型言語 KL1 のデータの参照数管理を行うことによるインクリメンタル GC 方式を提案し，その方式の評価結果を報告する．ICOT では，並列推論マシン PIM の研究開発を行っている．PIM 上の言語は，並列論理型言語 KL1 である．KL1 は副作用を持たない言語であるため，単純に実装すると，メモリを単調かつ急激に消費してしまい，ガーベジコレクションを頻繁に引き起こす．そこで，筆者らは，データ自体ではなく，そのデータへのポインタにデータの参照数を示す1ビットのフラグを設け，このフラグをコンパイラと処理系の両方で管理することにより，参照がなくなった時点で積極的にデータを回収するインクリメンタル GC 方式を提案した．この方式は，参照数情報をポインタ側に持つため，参照数更新のための余分なメモリ参照が不要で，マルチプロセッサ上での効率的な実現が容易である．本論文では，この方式を汎用計算機上に作成した処理系を用いて評価した結果を述べる．インクリメンタル GC を行わない場合に比較して，コード量の増加もわずかで，多くの場合，半分程度のヒープ消費量でプログラムを実行できることが分かった．また，回収されるデータの生存期間は非常に短く，キャッシュヒット率の向上も望めることが分かった．実行時間についても全体のメモリ量に対してアクティブセルの割合が大きいほど，一括型 GC の回数を減らせる本方式が有利であることを示した．

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 の研究開発を行っている．PIM 上の言語 KL1[3] は，GHC (Guarded Horn Clauses)[11] に制限を加えたフラット GHC を基本とした言語で，同期通信機構を備えた並列論理型言語である．フラット GHC は副作用を持たないため，プログラミングやデバッギングが楽になる半面，メモリの消費速度が速く，ゴミ集め（ガーベジコレクション，以下 GC）を頻繁に起こす．

GC の実現方式は，pure Lisp に始まる関数型言語の処理系に関して数多く提案されており，メモリ領域を使い切った時点で再利用可能領域を回収する一括方式，データに参照数のカウンタを設け，それが零になった時点で回収するリファレンスカウント方式 (RC方式) などが代表的である[4),5)]．また，一括 GC の際の実行の中断を避けるために，インクリメンタルに GC を行う一括 GC 方式の変形改良型も提案されている[1),17)]．

しかし，これらの方式をマルチプロセッサ上で実現しようとすると，単一プロセッサの場合に比べて，動的なオーバヘッドが大きくなる．すなわち，マルチプロセッサ上の GC は，他プロセッサにあるデータ（疎結合システムの場合）や，共有メモリ上のデータ（密結合システム）を扱わなければならない．これは，一括 GC 方式では実行の中断時間 (GC 時間) を長くし，RC 方式では参照カウンタの1回ごとの更新の手間を大きくする．さらに，一括型 GC はメモリ参照の局所性が乏しく，キャッシュのミスヒットやページフォールトが増える．したがって，マルチプロセッサ上の GC としては，実行時オーバヘッドが少なく，かつシステム全体の GC を行う回数をできるだけ少なくするような方式が望まれる．

そこで，筆者らは，データ自体ではなく，そのデータへのポインタにデータの参照数を示す1ビットのフラグを設け，このフラグをコンパイラと処理系の両方で管理することにより，小さな実行時オーバヘッドで多くの場合に良好な回収効率を実現するインクリメンタル GC 方式を提案した[2),14),18),20)]．構造体データへのポインタのフラグは構造体データの参照数が '1' と分かっているか，'2' 以上（多数）かもしれないかを表し，未定義変数へのポインタのフラグは未定義変数の参照数が '2' 以下か '3' 以上（多数）かを表す．データへの参照の増減は，コンパイラが調べ，参照が増える場合にはフラグを '多数' にし，参照がなくなり回収できる可能性がある場合には回収用の命令をコンパイルコード中に生成する．一方，処理系では，回収用の命令を実行すると，回収対象データのフラグを調べ，

† Incremental Garbage Collection by Multiple Reference Management of Data Objects in KL1 by YASUNORI KIMURA and TAKASHI CHIKAYAMA (The Fourth Laboratory, ICOT Research Center, Institute for New Generation Computer Technology).

†† (財)新世代コンピュータ技術開発機構研究所第四研究室
* 現在 (株)富士通研究所人工知能研究部
Fujitsu Laboratories Ltd.

本当に回収できるかどうかを決める．なお，本方式では，いったん‘多数’参照となったデータに対して参照数を減らす操作は行っていないため，後で参照がなくなっても回収できない．したがって，一括型 GC が別途必要となる．

次章以下では，まずインクリメンタル GC 方式の概要を述べ，次に単一プロセッサ上に作成した処理系エミュレータを用いて行った実験および評価結果について報告する．

2. 並列論理型言語 KL1

2.1 シンタックス

並列論理型言語 KL1 は，フラット GHC に基づいて ICOT で設計された言語である．KL1 プログラムは，次のようなシンタックスを持つ節（またはクローズ）の集合として表される．

$H :- G_1, \cdots, G_m | B_1, \cdots, B_n.$ $(m \geq 0, n \geq 0)$

ここで，H, G_i, B_i は，各々，クローズヘッド，ガードゴール，ボディゴールと呼ばれる．‘|’はコミットメントオペレータと呼ばれ，クローズ中でこれに先立つ部分を受動部（またはガード部），これに続く部分を能動部（またはボディ部）と呼ぶ．ガードゴールには，組込述語しか書けない．これは GHC の言語記述能力を保ちながら，効率的な実現を考慮して採用された制限である．また，ボディゴールには，組込述語とユーザ定義述語の両方のゴールが書ける．

2.2 実行方式

本節では，**図1**に示すような，整数を要素とするリストから，ある与えられた整数の倍数を取り除くプログラムを使って KL1 プログラムの実行方式について説明する．

述語‘filter/3’のクローズヘッドの第一引数は取り除く基になる整数，第二引数は整数のリスト，第三引数は結果のリストが置かれる変数である．プログラムでは，第二引数のリストを‘Car’，‘Cdr’に分解し，‘Car’が第一引数の倍数かどうかをガード部の組込述語で調べる．もし倍数なら，‘Car’を捨て（クローズ(1)），倍数でなければ結果のリストに入れる（クローズ(2)）．クローズ(3)は，終了条件である．この例では，クローズ(1)，(2)のガード部ではリストの分解が行われ，もしこのリストへの参照数が‘1’，すなわちここに引数として渡されたものだけであれば，コンスセル* は分解後捨てられる．一方，クローズ(2)が

```
filter(P, [X|Xs0], Ys0) :-
        X mod P =:= 0|
        filter(P, Xs0, Ys0).        (1)
filter(P, [X|Xs0], Ys0) :-
        X mod P =\= 0|
        Ys0 = [X|Ys1],
        filter(P, Xs0, Ys1).        (2)
filter(P, [], Ys0) :-
        true|Ys0 = [].              (3)
```

図1 フィルタプログラム例
Fig. 1 KL1 program of ‘filter’.

選択された時には，ボディ部では新たなリストの生成が起こる．したがってこの場合には，リダクションごとに1個のコンスセルの解放，割り付けが起こることになる．

2.3 ガーベジコレクション方式

前節（2.2節）で述べたように，KL1 ではメモリの解放，新たな割り付けが頻繁に起こり，メモリを急速に消費する．副作用を許す言語では，プログラマの責任で割り付け済みの領域を更新して再利用することによって，Prolog のようにバックトラックのある言語では，バックトラックにより解放された領域を処理系が暗黙のうちに再利用することによって，この急激なメモリ消費を抑えることができる．しかし，KL1 は副作用もバックトラックも許さない言語であるため，単純に実装するとメモリ領域を単調かつ急速に消費する．たとえば，ある構造体と一部の要素だけが異なる構造体を作ろうとすると，元々の構造体とは別の新しい構造体が生成されてしまう．このようなメモリの急激な消費は GC を頻発させる．さらに，メモリアクセスの局所性が悪くなることからキャッシュのミスヒットやページフォールトも多くなる．そこで，実行時に不要になったメモリ領域を実行時に効率的に回収し，再利用するインクリメンタル GC 機構が必要になる．

インクリメンタル GC 方式としては，各データオブジェクトにそこへの参照数を示すカウンタを設ける参照カウンタ方式が一般的である．しかし，この方式では，

1. 原理的に一語長分の参照カウンタフィールドがデータオブジェクトごとに必要である．
2. 参照カウンタの更新のためにメモリアクセスが必要でオーバヘッドが大きい．
3. 循環構造データを回収できない．

という問題がある．1. は，元々一語長のデータについては使えるメモリ領域がコピー方式の一括型 GC と同じく全体の半分程度になってしまうことを意味してい

* リストを構成する2語長セル．

る．また，マルチプロセッサ上の KL1 処理系では，疎結合の場合にはプロセッサ間を渡るポインタによって，密結合の場合には共有メモリによってデータが共有される．これらの共有データの参照数を管理しようとすると，プロセッサ間で参照数更新のためのメッセージを流したり，共有メモリの排他制御が必要となる．したがって，前記2. の問題点は，マルチプロセッサ上での実現を考えた時には，より顕著になる．

一方，実際の KL1 プログラムの実行の様子を検討すると，データの被参照数は '1' の場合が非常に多いことが予想される．また，KL1 におけるユニフィケーションなどでは，データオブジェクト本体をアクセスする必要がなく，オブジェクトへのポインタの操作で済むことが多い*．そこで，筆者らは，データに対するポインタ側にそのデータの被参照数が '1' かそれ以上 ('多数') かを示す1ビットのフラグ (Multiple Reference Bit, 以下 MRB) を設け，これによってデータの参照数を管理して GC を行う方式について提案した[2]．次章以降ではこの方式の概要と，その評価結果について述べる．

3. MRB 方式

3.1 MRB 方式の考えかた

MRB は，オブジェクトではなく，ポインタに付随する1ビットのフラグである．MRB の値は，基本的には，ポインタの指す先のデータセルの参照数が '1' であるか，'多数' であるかを示す．この結果，MRB 方式は，一般の参照カウンタによる管理に比べて以下のような特徴を持っている．

1. 参照数を管理する情報に必要なメモリ量が少ない（ポインタごとに1ビット）．
2. 参照数の更新操作が，データオブジェクトにアクセスすることなくできる．

これは，2.3 節で述べた一般の参照カウンタ方式の問題点のうちの二つを解決し，マルチプロセッサ上での効率的な実現を容易にするものである．

MRB 方式では，あるデータオブジェクトが参照数 '1' で生成され，その後参照数に変化がなければ，そのデータオブジェクトを消費**したゴールによって回収される．参照数の増減は，コンパイラが KL1 プログラムの個々のクローズを解析することによって検出し，参照数が増加する場合には MRB を '多数' にする命令を，データオブジェクトを回収できる可能性のある場合には，回収のための命令を生成する．また，構造体の一部を書き換えた新しい構造体を生成する場合，元の構造体への参照数が '1' で，新しい構造体の生成後は不要であれば，元の構造体の一部を書き換えてそのまま利用できる．

ただし，参照数がいったん '多数' になると，その後は参照するものがなくなっても検出できないため，不要データの回収は不完全である．また，本方式でも一般の参照管理方式 GC と同じく，循環構造は回収できない．そこで，他方式の GC と併用する必要がある．

3.2 MRB の定義

KL1 プログラムの実行において扱うデータオブジェクトは，構造体*，未定義変数，定数および間接ポインタに大別できる．これらのデータオブジェクトは，KL1 のゴールまたは構造体の要素として，ポインタを通して参照される．MRB は，ポインタに付加された1ビットの参照管理情報である．

あるデータオブジェクトがプログラムから参照される時，一般には何段かのポインタを経由して参照される．このポインタチェインを参照パスと呼び，一つでも MRB が '多数' を示すポインタを含む参照パスを MRP (Multiple Reference Path), MRB が '多数' を示すポインタを全く含まない参照パスを SRP (Single Reference Path) と呼ぶ．

SRP と MRP を○ ('白い' と呼ぶ)，● ('黒い' と呼ぶ) で表現することにすると，その基本的な意味は以下のようになる．

○：この参照パスの指すデータオブジェクトへの参照パスはこれ一本である（単一参照）．

●：この参照パスの指すデータオブジェクトへの参照パスがほかにもあるかもしれない（多重参照）．

未定義変数セルへのポインタが持つ MRB は扱いが少し異なる．KL1 の未定義変数には，基本的に，それを具体化するための参照と，具体化された値を読み出す参照の二つのポインタがあることが多い**．そこで，未定義変数セルへのポインタが持つ MRB の値の意味を以下のように決める．

○：この参照パスの指す未定義変数セルに対する参照パスは，ほかには SRP が一本あるか，MRP が任

---

* たとえば，変数と構造体のボディ部でのユニフィケーションでは，構造体本体へのポインタを変数セルに代入すればよく，構造体本体へアクセスする必要はない．

** 内容データを読み出して，データオブジェクト自体はその後使わなくなること．

* リスト，ベクタおよびストリング．

** 変数は，書き手と読み手があってはじめて意味があるのが普通である．

意本あるかのいずれかである.

●：この参照パスの指す未定義変数セルに対する参照パスがほかに幾つでもあるかもしれない.

単一化によって具体化された変数セルについては，その変数セルへの参照パスと内容データの MRB の組合わせで，以下のいずれかが成り立つ.

○○：参照パスが SRP で内容データの MRB が白なら，この変数セルに対する他の参照パスはない.

それ以外：この変数セルに対する他の参照パスがあるかもしれない.

このような表現形式を実現するためには，ポインタに限らず，変数セルの値となるものすべて，たとえば整数，定数などにも MRB ビットを付加できるようにする必要がある.

このように MRB の値を意味付けすると，構造体，変数セル，および単一化によって具体化した変数セルに対するポインタの MRB の許される組合わせは，**図 2**，**図 3**，**図 4** のようになる.

3.3 MRB の更新操作

KL 1 の実行は以下のような操作をメモリ領域に対して行いながら進む，この時，3.2 節で述べた条件が満たされるように，MRB の管理を行わなければならない.

1. 変数，構造体などの生成
2. 変数，構造体などのゴール間に渡る分配
3. 変数のデリファレンス
4. 変数の具体化

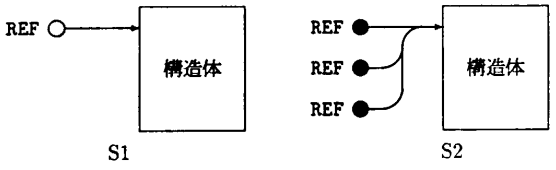


図 2 構造体の管理
Fig. 2 MRB maintenance for structures.

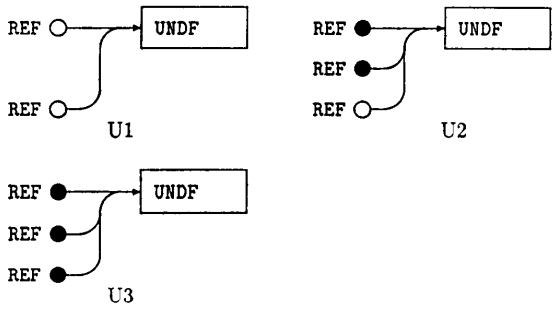


図 3 未定義変数の管理
Fig. 3 MRB maintenance for uninstantiated cells.

5. 構造体要素の取り出し

このうち，1，2はコンパイル時にクローズを解析し MRB 管理命令を生成することによって，3，4は実行時に MRB の値を調べることによって，MRB 管理が行われる．以下に，個々の場合についての MRB 管理の方法を説明する.

3.3.1 変数，構造体などの生成

クローズのボディ部で新たに変数セルを割り付けたり，構造体を作る場合である．構造体はボディゴールやボディユニフィケーションの引数として生成されるが，生成時には他の参照パスはないので MRB は白にする．変数については，ボディ部の出現回数により MRB の値を決める.

p :- true|$q(X), r(X, [\mathrm{car}|\mathrm{cdr}])$. (1)

p :- true|$q(X), r(X, [\mathrm{car}|\mathrm{cdr}]), s(X)$. (2)

クローズ(1)の場合は，ボディ部で新たに割り付ける変数 X の出現回数が2なので，このセルに対するポインタの MRB は白である（上記 U 1 に相当）．一方，クローズ(2)では，3回（以上）現れているので MRB は黒となる（U 3 に相当）．リスト [car|cdr] に対するポインタの MRB はクローズ(1)，(2)とも白である.

3.3.2 変数，構造体などのゴール間に渡る分配

クローズのヘッドで受けた引数をボディ部で分配する場合である.

$p(X)$:- true|$q(X)$. (1)

$p(X)$:- true|$q(X), r(X)$. (2)

クローズ(1)では，ヘッドで受けた引数をそのままボディゴールに転送するだけなので参照数の増減はなく，MRB に変化はない．クローズ(2)では，二つに分配するので，MRB は黒にする（X が構造体であれば S 2，未定義変数であれば U 2 または U 3 に相当する）.

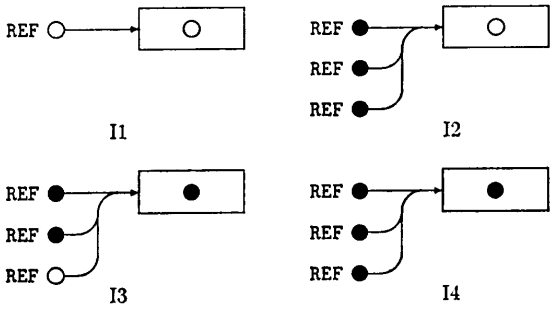


図 4 具体化変数の管理
Fig. 4 MRB maintenance for instantiated cells.

3.3.3 変数のデリファレンス

変数のデリファレンスを行う場合である．デリファレンス結果のポインタの MRB は，デリファレンス途中の間接ポインタのなかで一つでも MRB が黒のものがあれば，すなわち MRP ならば，黒，それ以外の場合は白とする．この操作は，デリファレンスチェインに現れる間接ポインタの MRB の論理和をとることによって実現される．

3.3.4 変数の具体化

変数の具体化の場合の MRB の管理は，まず，二つの変数をデリファレンスし，どちらかの参照パスが MRP ならば黒で具体化，さもなくば白で具体化する．**表 1** に個々の場合を示す．変数 X, Y の参照パスが表の第一行と第一列で表された場合の具体化されたセルの MRB を，対応する行，列位置で示す．表で，☆は，構造体同士のユニフィケーションが行われることを示し，この場合には，要素ごとのユニフィケーションがこの表のルールに従って行われる．また，U2 白および，U2 黒はそれぞれの SRP/MRP を使ってユニフィケーションが行われることを示す．

3.3.5 構造体要素の取り出し

ガード部でベクタの要素を取り出し，その要素とベクタ本体をボディ部に転送する場合である．

$p(V, I)$:- vector_element(V, I, E)|$q(V, E)$.

ガード部の組込述語 'vector_element' は，ベクタ V の I 番目の要素を取り出し，変数 E で受け，両方をボディ部へ転送する．

この場合，読み出したベクタの要素は，ベクタの本体と，変数 E から指されているので，ベクタ V の I 番目の要素の MRB と変数 E の MRB の両者を黒くしなければならない．ベクタ V 全体への参照数は増減がない．

表 1 ユニフィケーション時の MRB 管理
Table 1 MRB maintenance in unification.

X/Y	S1	S2	U1	U2白	U2黒	U3
S1	☆	☆	I1	I1	I3	I4
S2	☆	☆	I3	I4	I3	I4
U1	I1	I3	I1	I2	I3	I2
U2白	I1	I4	I2	I2	I4	I4
U2黒	I3	I3	I3	I4	I3	I3
U3	I4	I4	I2	I4	I3	I4

4. MRB によるガーベジコレクション

4.1 ガーベジコレクションの可能性

3.3 節のように MRB を管理すると，KL 1 の実行中に，あるデータオブジェクトへの参照が最後であることが MRB を用いて判定でき，そのオブジェクトの領域を回収し，再利用できることがある．回収が可能な場合について以下に説明する．

4.1.1 変数のデリファレンス

変数のデリファレンス時の間接ポインタは，その間接ポインタセルへの参照が単一 (SRP) で，この間接語の内容の MRB が白であれば，デリファレンス時に回収できる．**図 5** で，デリファレンス後は，変数 X から指される間接セルと，コンスセルを指す間接セルは回収される．

4.1.2 構造体とのユニフィケーション

受動部で，ゴールの引数の一つとソースプログラム中のヘッドに書かれた構造体との単一化に成功した時，ゴール引数として与えられた構造体が SRP 参照であれば回収できる．たとえば以下のようなクローズがあったとき，第一引数として与えられたコンスセルは SRP 参照であれば回収できる．ベクタが現れた場合も同様である．

p([X|Y]) :- true|$b(X), c(Y)$.

4.1.3 ガード部のみに現れる変数とのユニフィケーション

ゴール引数がガード部のみに現れる変数とのユニフィケーションに成功した時，そのゴール引数が SRP 参照であれば回収できる．以下の例で変数 Void はボディ部に現れないので，このクローズが選択されれば変数 Void から指されているデータが回収の対象となる．

p(Void) :- true|true.

回収対象が構造体の場合，前述のユニフィケーション時の回収と異なり，構造体自体のみならず，構造体

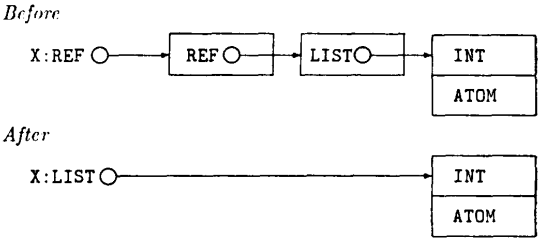


図 5 間接ポインタの回収
Fig. 5 Reclamation of indirect cells whose MRBs are off.

から SRP で参照されているデータオブジェクトすべてについて，再帰的に回収できる．

4.2 回収の方法とタイミング

4.1.2 項や，4.1.3 項で示したようなデータ回収の可能性は，KL1 のソースクローズを解析することによって分かるので，コンパイラによって回収のための命令を陽に出すものとする．また，実際の回収の可否はそのクローズが選択されると分かった時点で行わなければならない．したがって，回収のための命令は概念的には，コミットメントオペレータの位置に生成される．一方，デリファレンス時の回収（4.1.1 項）は動的にしか分からないので実行時に行われる．

4.3 関連する最適化

ユーザプログラム中の KL1 のベクタの要素をボディ部で更新する時，そのベクタが SRP 参照ならば更新する要素をベクタ本体に破壊的に書き込み，ベクタ本体を再利用することができる．

p(Vold, I, C, Vnew) :- true|
　set_vector_element(Vold, I, C, Vnew).

この例では，ボディ部の組込述語 set_vector_element で，ベクタ Vold の I 番目の要素を C に書き換えて新しいベクタ Vnew にしている．もし，Vold が SRP 参照ならば，Vold を再利用し，I 番目の要素を C にして Vnew とすることができる．Vold が MRP 参照ならば，Vnew のために新たにベクタを割り付け，I 番目以外の要素を Vold からコピーし，I 番目の要素を C にしなければならない[15]．

5. 実験処理系と命令セット

5.1 実験処理系

MRB-GC 方式の有効性と問題点を調べるために，PDSS システム[6]を使って実験を行った．PDSS システムは，UNIX マシン上に構築された単一プロセッサ上での KL1 処理系である．

この処理系は，既に提案した KL1 の抽象マシン[7]に準拠している．すなわち，ユニフィケーションを行うためのレジスタ群（引数レジスタと呼ばれる）や，実行環境を保持するための制御レジスタ群を持ち*，メモリには，MRB のフィールドを持った変数やコンスセルに使われるメモリがそのサイズごとにフリーリストとして管理されている．これは，MRB 方式ではデータの割り付け開放が順不同に起こるため連続領域を端から使っていくといったメモリ管理が適さないことから採られた方法である．

また KL1 のゴールは，メモリ上でゴールレコードとして表され，ゴールレコードには実引数を格納するための引数スロットと，このゴールを実行するための命令番地などの制御情報を格納するためのスロットがある．さらに，ゴールの実行が中断した時の処理を行うために使うサスペンションレコードなどがメモリに取られている．KL1 のプログラムは，まず，KL1-B[7]命令列にコンパイルされる．そして，PDSS 処理系が KL1-B 命令列を次々に読み出し，解釈実行することにより，KL1 プログラムの実行が進む．

5.2 構造体中の未定義変数の扱い

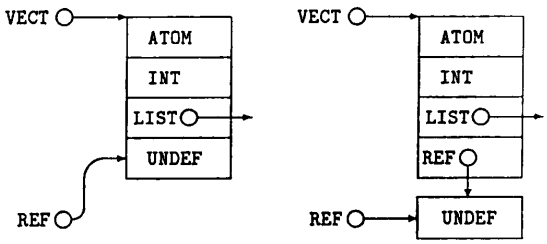
PDSS 処理系では，**図6**の MRB 方式で示すように，WAM[12] とは異なり構造体中の未定義変数は未定義セルを別に取り，構造体にはそれへの参照ポインタを入れている．構造体中に未定義変数を直接取るとその構造体本体を回収できる時でも，構造体中の未定義変数を指している他の参照があるかもしれないので回収できないことがある．未定義変数が具体化され，その値を他の参照パスのすべてから読み出されるまで構造体の回収を遅らせる方式も考えられるが，実行時の処理が複雑になる．

5.3 命令セット

筆者らは，既に KL1 の抽象命令セット[7),19]を提案した．本節では，MRB を管理するために新たに導入した命令について説明する．導入した命令は，3.3 節で述べた MRB 管理のための命令と，4 章で述べたガーベジセル回収用命令である．また，従来からの命令で，デリファレンス処理を伴う命令については，3.3.3 項で示したように MRB を管理し，4.1.1 項で示したように不要な間接セルの回収を行うように仕様を変更した．

5.3.1 MRB 管理のための命令

3.3 節で述べた MRB の更新操作のための命令は，



WAM 方式　　MRB 方式

図 6 構造体中の未定義セル
Fig. 6 Treatment of uninstantiated cells in structures.

* PDSS システムでは，実際にはメモリに割り付けられている．

主としてボディ部でデータへの参照数が増える場合に，そのデータへのポインタのMRBを黒にすることである．**表2**において，(1)～(5)は，各々，レジスタ *Xi* が持つデータをレジスタ *Aj*，ゴールレコードの引数スロットの *Gj* 番目，リストLの 'Car' 部，'Cdr' 部，ベクタ *V* の *I* 番目要素位置に転送すると共に，両者のMRBを黒くする命令である．(6)～(10)は，新たに未定義変数セルを割り付け，オペランドとして与えられたレジスタや，構造体の中からMRB黒で指させるための命令である．(11)は，ベクタ *V* の *I* 番目要素位置のMRBを黒くする命令である．

5.3.2 回収のための命令

4章で述べたガーベジセル回収のための命令である．なお，この命令セットは，デリファレンス命令を陽に持っていないので，デリファレンス時の回収は個々の命令中で必要に応じて行われる．

- collect_list *Ai*

  レジスタ *Ai* から指されるリストがSRP参照であれば，リストセルを回収する．
- collect_vector *Ai*, *Nv*

  レジスタ *Ai* から指される長さ *Nv* のベクタがSRP参照であれば，ベクタセルを回収する．
- collect_value *Ai*

  レジスタ *Ai* から指される木状のデータオブジェクト群がSRP参照である限り再帰的に回収する．
- put_reused_list *Ai*, *Aj*

  レジスタ *Ai* から指されるリストがSRP参照であれば，そのリストセルへのポインタをレジスタ *Aj* に転送する．*Aj* のMRBは白にする．これは，一つのクローズ内で 'collect_list *Ai*' と 'put_list *Aj*' 命令が現れ，*Ai* のMRBが白の時，いったんリストセルを解放せずにそのまま再利用するための最適化の命令である．*Ai* がMRP参照であれば，*Aj* のための新たなリストセルの割り付けが起こる．
- put_reused_vector *Ai*, *Aj*, *Nv*

  これは，*Ai* が長さ *Nv* のベクタを指していることを除けば，操作は 'put_reused_list' 命令の場合と同様である．

表2 MRBの更新操作のための命令
Table 2 Instructions for MRB updating.

	命　令
(1)	put_marked_value *Xi*, *Aj*
(2)	set_marked_value *Xi*, *Gj*
(3)	write_marked_car_value *L*, *Xi*
(4)	write_marked_cdr_value *L*, *Xi*
(5)	write_marked_element_value *V*, *I*, *Xi*
(6)	put_marked_variable *Xi*, *Aj*
(7)	set_marked_variable *Xi*, *Gj*
(8)	write_marked_car_variable *L*, *Xi*
(9)	write_marked_cdr_variable *L*, *Xi*
(10)	write_marked_element_variable *V*, *I*, *Xi*
(11)	mark_element *V*, *I*

```
p([X|Y], X, Z) :- true|Z=[X|ZZ], q(X, ZZ).
p/3: wait_list A1                        % 第一引数はリスト？
     read_car A1, X3                     % car を読む
     read_cdr A1, X4                     % cdr を読む
     wait_value X3, A2                   % 第二引数の単一化
     collect_value A2                    % 第二引数の回収
     collect_value X4                    % ボイド変数の回収
     put_reused_list A1, X5              % リストの再利用
     write_marked_car_value A1, X3       % 変数 'X' の転送とマーク
     write_cdr_variable A1, A2           % 変数 'ZZ' の割り付け
     get_list_value X5, A3               % Z=[X|ZZ]
     put_value X3, A1
     execute q/1
```

図7 KL1プログラムとコンパイル例
Fig. 7 KL1 sample program and its compiled code.

5.3.3 コンパイル例

MRB-GC用命令を生成するコンパイル例を**図7**に示す．第一引数のリストの 'Cdr' 部（変数 'Y'）および第二引数（変数 'X'）に対しては，ボディ部で使われていないので，回収命令が生成される．一方，第一引数のリストの 'Car' 部（変数 'X'）は，ボディ部で2回使われるのでマーク命令（write_marked_car_value）が生成される．さらに，ボディ部でリストが使われているので，第一引数のリストの再利用を試みるための命令（put_reused_list）が生成される．

5.4 一括型GCでのMRB管理

MRBによるGCでは，いったん黒くなったデータやループ状のデータは後で参照がなくなっても回収できない．したがって，別途一括型のGCが必要となる．PDSS処理系では，このためにコピー方式によるGCを備えている[13]．このコピー方式GCは，旧領域から新領域に生きているデータをコピーする時に，間接ポインタをデリファレンスすることと，参照数が '1' のデータに対しては，一括GCの終了後にはMRB白のポインタで指されるようにMRBの付け換えを行うことに特徴を持っている．

6. 実験と評価

実験は，5章で述べた PDSS 処理系を用いて行った．ベンチマークプログラムとしては，**表3**に示すプログラムを使った．表3では，個々のプログラムのリダクション数と機能を簡単に示す．

6.1 命令コードの特性

表4に，表3の10個のプログラムについて MRB-GC を行う場合と行わない場合のコンパイルコードの静的および実行命令数の比('MRB-GC を行う場合/MRB-GC を行わない場合') の平均と標準偏差を示す．この比が，MRB-GC を行うことによる命令数の増加を示す．表より，MRB-GC を行うことによる静的命令数，実行命令数の増加は，各々 5%，7% であり，全体への影響は大きくない．

6.2 使用ヒープ量

表5に，各々のプログラムを実行するのに必要としたヒープ量（変数および構造体のための領域）を語*数で示す．ヒープは，'MRB-GC あり' ではフリーリストにより管理され，'MRB-GC なし' では連続領域を端から使っていく．また，MRB-GC なしの場合については，構造体中の未定義変数を構造体の外側に採る必要がないので，図6の WAM 方式のように扱って MRB-GC に不当に有利にならないようにした．

表より，MRB-GC を行うと MRB-GC を行わない場合に比べて，多くのプログラムで少ないヒープ量で実行できることが分かる．これは，KL1 におけるデータの参照数は多くが '1' であることを実証し，MRB-GC の有効性を示すものである．'Qlay' の場合には，MRB-GC を行ったほうがメモリを多く消費する．これは，構造体の変数セルを構造体の外に採ること（図6）の増分に相当する．このプログラムではネストの深いリストを多く扱うが，ネストの段数分だけ余分に間接セルを割り付けることによるヒープの使用増が顕著に表れた場合と言える．

'Kl1 cmp*' とあるのは，4.3節で述べた，単一参照 (MRB が白) のベクタの再利用をしない時の 'kl1 cmp' の使用ヒープ量である．KL1 コンパイラでは，レジスタ割り付けの際，コンパイル対象のプログラムに現れる変数の生存区間を管理するためのテーブルをベクタで表現している．このテーブルは，レジスタを最適に割り付けるために何度も参照，更新される．したがって，このテーブルの再利用を考えず，破壊的な更新ができないことにすると，更新ごとにベクタ本体をコピーする必要が生じる．この場合では，7倍近くのヒープを必要とすることが分かり，ベクタ本体の再利用の効果が大きいことが分かる．

6.3 回収されるデータの生存期間

MRB-GC によって回収されるデータの生存期間を調べるために，プログラムの実行開始から終了までに渡って，変数，リストなどのためにヒープ領域が割り

* 一語は MRB を含めたタグ部1バイト，データ部4バイトから構成される．

表3 ベンチマークプログラム
Table 3 Benchmark programs.

プログラム	リダクション	機能
prime	5,876	素数生成
queen	38,878	エイトクイーン
qlay	19,419	レイヤード法[9]によるエイトクイーン
bup	34,857	ボトムアップパーサ
kl1 cmp	14,919	KL1 で記述した KL1 コンパイラ
espascal	335,115	パスカルの三角形による係数の計算 (E. Tick[10] による)
etsmall	918,520	詰め込みパズル（同上）
tri	666,235	パズル問題（同上）
semi	292,309	半群の要素の計算（同上）
pax	17,530	並列自然言語パーサ

表4 静的，実行命令数の比
Table 4 Static, dynamic sizes of benchmark programs.

	平均	標準偏差
静的命令数比	1.05	0.028
実行命令数比	1.07	0.041

表5 使用ヒープ量
Table 5 Heap sizes used for execution of benchmark programs.

プログラム	MRB-GC		比 (B/A)
	なし (A)	あり (B)	
prime	10,848	1,503	0.14
queen	817,070	459,919	0.56
qlay	582,146	679,557	1.17
bup	66,655	20,076	0.30
kl1 cmp	34,969	17,873	0.51
kl1 cmp*	231,797	216,312	0.93
espascal	471,760	50,284	0.11
etsmall	3,174,541	671,354	0.21
tri	1,209,532	1,209,529	1.00
semi	493,383	101,265	0.21
pax	24,298	12,784	0.53

付けられてから，回収されるまでの期間（ライフタイム）をリダクション数で数えた．**図8**に結果を示す．図で，横軸はリダクション数で，縦軸が各リダクション間データが生きた後，MRB-GCによって回収されたヒープ領域の，総回収量に対する割合である．

図から，MRB-GCによって回収されるデータの多くは，その生存期間が非常に短く，高々10リダクション程度以内に大多数のデータは回収されていることが分かる．実験に用いたプログラムの1リダクションあたりに実行される命令数の平均は20命令程度であるので，これは，200命令に相当する．MRB-GCを行わない場合には，変数，リストなどはヒープを単調に消費して割り付けられるので，キャッシュのミスヒットの増大を招くと予想される．一方，MRB-GCを行うと，データの生存区間がたとえ長くても，データが回収されフリーリストにつながれた後，すぐに使われればキャッシュのヒット率の向上が望める*．

6.4 命令別の回収効率

ヒープの回収は，5.3.2項で示した命令およびデリファレンス時に行われる．**図9**に，命令別の回収量の内訳を示す．ここでは，'Reuse' 命令（5.3.2項）による再利用もいったん回収された後，再び割り付けられたものとして扱った．

'Prime' では，リストを使ったストリーム通信を行っているため，'collect_list' による回収が多い．'Queen' では，'collect_value' による回収が多い．これは，探索木の終端クローズのヘッド引数の多くがボイド変数で受けていることによるものと考えられる．また，

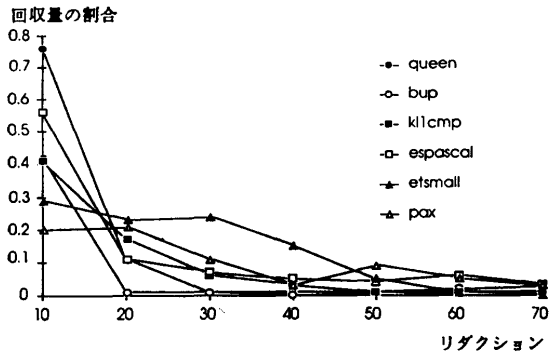


図8 回収されるデータの生存期間
Fig. 8 Lifetimes of garbage collected cells by MRB.

* 多くの場合，回収時にはそのデータは読まれているため．

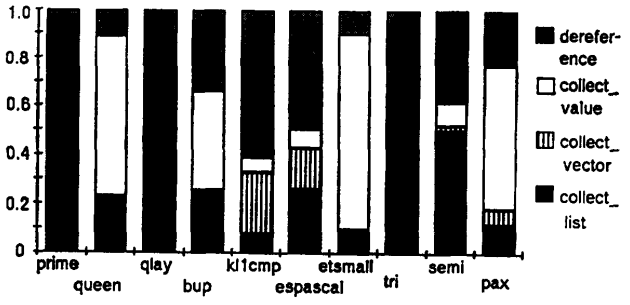


図9 命令別の回収効率
Fig. 9 Reclamation efficiency of MRB-GC instructions.

'qlay' や 'tri' では，ほとんどがデリファレンス時の回収である．これは，表5から分かるように，元々この二つのプログラムは回収率があまり良くなく，わずかに回収された領域がデリファレンスによるものだったことを示している．さらに，どのプログラムでもデリファレンス時の回収の比率が比較的大きい．デリファレンス時の回収は実行時に動的に行われるため，実際のマシンでの本方式の効率的な実現のためにはハードウェアによるサポートが有効である[9]．

6.5 Reuse 命令の効果

MRBによるメモリ管理方式では，不要になった領域の回収，新たな領域の割り付けの手間は単純にヒープを伸ばしていく方式より大きくなる．そこで，同じクローズ内での同じ大きさの領域がMRB-GCにより回収される可能性があり，またボディ部での割り付けが起こることがわかる時には，いったん回収して割り付けし直すことをせずに，そのまま使ってしまうことが考えられる．

このために導入された命令が 'Reuse' 命令であり，コンスセルおよびベクタ本体について 'Reuse' 命令が生成されている（5.3.2項）．

表6に，'prime'，'bup'，'semi' の三つのプログラムについて，'Reuse' 命令を生成しない場合と，する場合のフリーリストへのアクセス回数を示す．この三つのプログラムでは，全体のフリーリストのアクセスの各々，45%，20%，36% が減少しており，メモリ管理

表6 フリーリストアクセス数
Table 6 The number of access counts for free lists.

プログラム	Reuse 命令		比 (B/A)
	なし (A)	あり (B)	
prime	21,497	11,751	0.55
bup	95,257	75,971	0.80
semi	1,056,655	675,045	0.64

オーバヘッドを節約できた．その他のプログラムについては，構造体を 'Reuse' できることがほとんどなく，効果はわずかであった．この最適化は，プログラミングスタイルに依存し，'Reuse' できる場合／できない場合がプログラムによってはっきりする傾向にある．しかし，コンパイラのみの最適化で実現可能で，'Reuse' できない場合の実行時オーバヘッドは小さく*，'Reuse' できる場合の効果は大きい．したがって，この最適化の実用上の価値は大きいと言える．

6.6 一括型 GC の回数

MRB-GC はインクリメンタル GC の一種であるが，いったん黒くなったデータは回収できないため別途一括型 GC が必要になる．**表 7** に，MRB-GC を行う場合と行わない場合の一括型 GC の回数を示す．表で，'メモリ量' は，変数や構造体を割り付けるための領域とゴールレコードなど制御データを置く領域の合計で，大きさは，プログラムを実行できる最小限に近い値に設定した．MRB-GC を行った場合には一括型 GC が起動される回数が減少していることが分かる．これは，一括型 GC の回数をできるだけ減らすという初期の目的を達成するものである．

6.7 アクティブセルと実行時間の関係

MRB-GC を行うと，インクリメンタルなメモリ回収のために通常のプログラム実行の時間が増え，また，MRB-GC を行わないと，一括型 GC の回数が増えるため，一括 GC 時のデータのコピーのための時間が増える．コピー式一括型 GC の時間は，GC 起動時のアクティブセル数に比例する．そこで，一括型 GC の時間も含めた場合の全体の実行時間の比('MRB-GC ありの場合の実行時間'/'MRB-GC なしの場合の実行時間') を，メモリ総量に対するアクティブセルの割合を変化させて測定した*．結果を**図 10** に示す．

図で，横軸がアクティブセルの割合，縦軸が時間比である．MRB-GC を行う場合では，メモリはフリーリストにより管理されているため，データの割り付け，回収時のフリーリスト管理の時間も含んでいる．'Prime', 'tri' や 'espascal' ではアクティブセルの割合を大きくするために総メモリ量を極端に小さくすると，総メモリ量がプログラム実行時に一時に必要とする最大メモリ量より小さくなり，実行を続行できなくなることが起きたために，アクティブセルの割合が大の時のデータが採れていない．このようなアクティブセルの割合が 10% 以下と非常に少ないプログラムの場合には，一括型 GC の時間がわずかで MRB-GC を行うことによるオーバヘッドのほうが顕著になる．'Bup', 'qlay' や 'semi' では，緩やかな右下がりのグラフとなっていることから，アクティブセルの割合が大きくなる（'プログラムの大きさ' に対してメモリサイズが小さくなる）ほど，MRB-GC を行うことによる効果が大きくなっていると言える．アクティブセルの割合が 40〜50% を越えると MRB-GC を行うほうが実行時間が速くなる傾向にある．

本論文での評価では，C言語で書かれた汎用計算機上のエミュレータ (PDSS 処理系) を使っている．エミュレータでは，MRB の管理も命令としてエミュレートされるので，比較的低速である．一方，一括型 GC は，C言語で直接記述されている．このことは，MRB 管理のオーバヘッドが一括型 GC のオーバヘッ

表 7 一括型 GC の回数
Table 7 The number of stop-and-collecting GC.

プログラム	メモリ量 (Kwords)	MRB-GC	
		なし	あり
prime	50	52	0
queen	20	58	27
qlay	400	3	2
bup	20	8	1
kl1 cmp	10	5	2
kl1 cmp*	10	29	2
espascal	20	34	2
etsmall	50	72	30
tri	100	10	10
semi	100	19	2
pax	30	1	0

* 'Reuse' しようとする構造体を指すポインタの MRB の白／黒を調べるだけである．

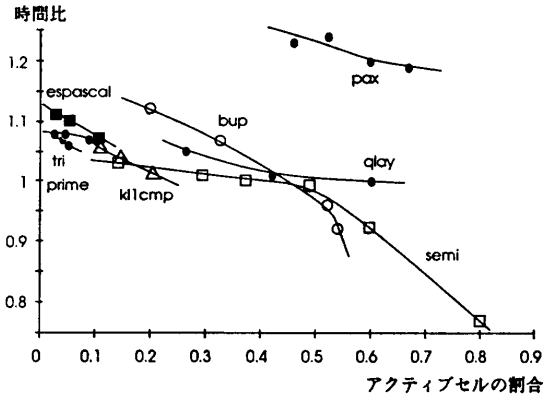


図 10 アクティブセルと実行時間の関係
Fig. 10 Relationship between active cells and execution time.

* 具体的には，総メモリ量を変化させ，一括型GC時にコピーされたデータ量をアクティブセル量として測定した．

ドに対して大きく出やすくしている．また，一括型GC時にはキャッシュのミスヒットが多くなるが，MRB-GCでは，6.3節で示したように，キャッシュのヒット率の向上が望める．さらに，実際のマルチプロセッサ環境では，単一プロセッサ上での同一プログラムの実行の場合よりアクティブセルの量が増える傾向にあることが報告されている[16]．これらの点を考慮に入れれば，MRB-GCをMRB管理のためのハードウェアを持つ並列計算機上に実現するならば，この実行時間の増大は小さくなり，結果として，アクティブセルの割合がそれほど大きくなくてもMRB-GCのほうが有利になることが予想される．したがって，MRB-GCを実際の並列計算機上に実現することは十分実用性があると言える．

7. おわりに

本論文では，データ自体ではなく，データへのポインタに参照数が'1'か'多数'かを表すビットを設け，このビットを管理することにより，参照がなくなった時点でデータの占める領域を回収するインクリメンタルGC方式(MRB-GC)を提案し，評価を行った．その結果，以下のことが分かった．

1. MRB-GCは，これを行わない場合に比べて，コード量の増加もわずかで，多くの場合，消費ヒープ総量は半分程度でプログラムを実行できる（6.1節，6.2節）．
2. 回収されるデータの生存期間は非常に短くキャッシュ機構を持つプロセッサでは，ヒット率の向上も望める（6.3節）．
3. 回収命令別による回収データ量の割合では，デリファレンス時のものが比較的多い．デリファレンス時の回収は動的に行われるため，効率的な実現のためには，ハードウェアによるサポートが有効である（6.4節）．
4. クローズ内で，回収，割り付けが起こる同じ大きさの構造体を再利用することによって，フリーリストへのアクセス回数を減らすことができる（6.5節）．
5. 一括型GCの回数を大幅に減らせ，実行時間についても全体のメモリ量に対してアクティブセルの割合が大きいほど，MRB-GCが有利であることが分かった（6.6節，6.7節）．

筆者らは，現在MRB-GCを支援するハードウェア機能を備えた並列推論マシンPIM/p[9]の開発を進めており，PIM/pのマシンアーキテクチャに合った命令セットの改良などを行っている．MRB-GCを行うことの時間的オーバヘッド，MRB-GCが有利となるアクティブセルの割合，などについては，PIM/p上で改めて測定する必要がある．これらは，今後の検討課題である．

謝辞 日頃御指導頂くICOT第4研究室，内田俊一室長に感謝します．また，PDSS処理系を使って実験を行ってくれた，富士通SSLの武井則雄氏，平野喜芳氏，貴重なコメントを頂いたICOT第4研究室，後藤厚宏氏をはじめとするICOTおよび関連メーカの方々，PIM/MPSIワーキンググループ，PIM/MPSI開発グループ，並列処理検討会のメンバの方々に感謝します．

参考文献

1) Baker, H. G.: List Processing in Real Time on a Serial Computer, *Comm. ACM*, Vol. 21, No. 4, pp. 280-294 (1978).
2) Chikayama, T. and Kimura, Y.: Multiple Reference Management in Flat GHC, *Proc. Int. Conf. on Logic Prog. '87*, pp. 276-293 (1987).
3) Chikayama, T., Sato, H. and Miyazaki, T.: Overview of the Parallel Inference Machine Operating System (PIMOS), *Proc. Int. Conf. on FGCS '88*, pp. 230-251 (1988).
4) Cohen, J.: Garbage Collection of Linked Data Structures, *ACM Comput. Surv.*, Vol. 13, No. 3, pp. 341-367 (1981).
5) Deutsch, L. P. and Bobrow, D. G.: An Efficient, Incremental, Automatic Garbage Collector, *Comm. ACM*, Vol. 19, No. 9, pp. 522-526 (1976).
6) ICOT第四研究室：PDSS—言語仕様と使用手引き—，TM-437, ICOT (1988).
7) Kimura, Y. and Chikayama, T.: An Abstract KL1 Machine and Its Instruction Set, *Proc. Symp. on Logic Prog. '87*, pp. 468-477 (1987).
8) Okumura, A. and Matsumoto, Y.: Parallel Programming with Layered Streams, *Proc. Symp. on Logic Prog. '87*, pp. 224-231 (1987).
9) Shinogi, T., Kumon, K., Hattori, A., Goto, A., Kimura, Y. and Chikayama, T.: Macro-call Instruction for the Efficient KL1 Implementation on PIM, *Proc. Int. Conf. on FGCS '88*, pp. 953-961 (1988).
10) Tick, E.: Performance of Parallel Logic Programming Architectures, TR-421, ICOT (1988).
11) Ueda, K.: Guarded Horn Clauses: A Parallel Logic Programming Language with the Con-

cept of a Guard, TR-208, ICOT (1986).
12) Warren, D. H. D.: An Abstract Prolog Instruction Set, Technical Note 309, SRI International (1983).
13) 宮内, 木村, 近山, 久門: MRB による多重参照管理方式—KL 1 処理系における一括型 GC の特性評価—, 第 35 回情報処理学会全国大会論文集 (昭和 62 年後期), 2Q-7 (1987).
14) 近山, 木村: ポインタタグ (MRB) による多重参照管理方式, 第 35 回情報処理学会全国大会論文集 (昭和 62 年後期), 2Q-5 (1987).
15) 今井, 木村, 稲村, 後藤: 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—, 信学技報, CPSY 88-47 (1988).
16) 佐藤, 後藤: KL 1 並列処理系の評価—メモリ消費特性と GC—, 並列処理シンポジウム JSPP '89 予稿集, pp. 195-202 (1989).
17) 小沢, 林, 服部: 実時間 GC の実現方式と評価, 情報処理学会論文誌, Vol. 29, No. 5, pp. 465-471 (1988).
18) 木村, 近山: MRB による多重参照管理方式—KL 1 処理系における実時間ガーベジコレクション方式—, 第 35 回情報処理学会全国大会論文集 (昭和 62 年後期), 2Q-6 (1987).
19) 木村, 近山, 久門, 中島 (浩): 並列推論マシン PIM—KL 1 の抽象命令仕様とコンパイラ—, 第 34 回情報処理学会全国大会論文集 (昭和 62 年前期), 2P-1 (1987).
20) 木村, 西田, 宮内, 近山: KL 1 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 '87 予稿集, pp. 215-222 (1987).

(平成元年 6 月 14 日受付)
(平成元年 10 月 11 日採録)

木村 康則 (正会員)

昭和 31 年生. 昭和 54 年名古屋工業大学工学部電子工学科卒業. 昭和 56 年東京工業大学大学院修士課程修了. 同年(株)富士通研究所入社. Lisp マシン ALPHA のハードウェア, ファームウェアの開発に従事. 昭和 60 年 6 月(財)新世代コンピュータ技術開発機構に出向. 第四研究室にて KL 1 コンパイラの開発に従事. 平成元年 4 月(株)富士通研究所に復職, 現在に至る. 人工知能研究部第三研究室に所属し, 並列マシン, 並列言語などの開発に従事している. 電子情報通信学会会員.

近山 隆 (正会員)

昭和 28 年生. 昭和 52 年東京大学工学部計数工学科卒業. 57 年同工学系大学院情報工学専門課程博士課程修了. 工学博士. 同年富士通(株)入社. (財)新世代コンピュータ技術開発機構に出向, 現在に至る. この間, 手続き型言語, 関数型言語, 論理型言語, オブジェクト指向言語とこれらの逐次および並列処理系, プログラミング環境, 論理型言語専用計算機とそのオペレーティングシステムの研究開発に従事.

論 文 誌 編 集 委 員 会

委員長	村井 真一		
副委員長	益田 隆司		
委員	浮田 輝彦	小池 誠彦	小谷 善行
	佐藤 輿二	島津 明	戸川 隼人
	永田 守男	原田 紀夫	疋田 輝雄
	松田 晃一	三浦 孝夫	毛利 友治
	吉澤 康文	米崎 直樹	