TR-321

KL1の多重参照ビットによるGC方式について

木村康則、西田健次
宮内信仁、近山　隆

November, 1987

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

ＫＬ１の多重参照ビットによるＧＣ方式について

Realtime GC by Multiple Reference Bit in KL1

木村康則，西田健次，宮内信仁(＊)，近山隆

Yasunori KIMURA, Kenji NISHIDA, Nobuhito MIYAUCHI, and Takashi CHIKAYAMA

（財）新世代コンピュータ技術開発機構，(＊)三菱電機（株）
ＩＣＯＴ，　(＊)MITSUBISHI

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の核言語ＫＬ１は，制限を加えたＧＨＣ（フラットＧＨＣ）に基づく並列論理型言語の一種である．ＫＬ１は，メモリセルの書き替えといった副作用を持たない言語であり，並列処理システムの実現にとって不可欠である同期や通信等の並列処理の基本概念が自然に記述できる．

ただし，ＫＬ１のような副作用を持たない言語を単純に実装すると，メモリ領域を単調，かつ，急速に消費してしまう．例えば，同じ構造体データの一部を書き替えた構造体を何回も生成しようとすると，元々の構造体の一部だけが異なった新しい構造体が次々とメモリ領域に割り付けられてしまう．

このようなメモリの急速な消費は，一括型ＧＣを頻発させてしまう．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機構（実時間ＧＣ）が必要となる．

実時間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 原理的にポインタの一語長分の参照カウンタフィールドが必要である，

② 参照カウンタ更新のためのオーバヘッドが大きい，

という問題がある．

これに対し，実際のプログラムの実行では，データオブジェクトへの参照数が１または２の場合が多いと予想される．このため，前述①の問題は，カウンタのビット幅を小さくすることが可能であろう．

一方，ＫＬ１の処理における単一化では，データオブジェクト本体にアクセスする必要がなく，そこへのポインタを渡すだけの操作が多い．しかし，参照カウンタ方式では，カウンタを更新する必要があれば，常にオブジェクト本体にアクセスしなければならないため，メモリアクセスが増えてしまう．ＰＩＭのように，密結合マルチプロセッサ環境でデータが多数のプロセッサに共有されている場合は，カウンタの更新操作を排他的に行う必要があり，カウンタの更新のコストがさらに大きくなる．

本稿では，並列論理型言語ＫＬ１の実行における，多重参照ビット（Multiple Reference Bit：ＭＲＢと以後呼ぶ）による実時間ＧＣ方式とその評価について述べる．

本方式は，データオブジェクトではなく，ポインタにおいて参照数を管理することによって，参照カウント方式の②の問題を解決することを狙っている．さらに，①の問題については，参照カウンタに相当する情報を１ビットのＭＲＢで表現することによってメモリ資源を節約し，操作を簡素化している．ただし，ＭＲＢは１ビット分の情報しか持たないため，参照数が０となったメモリセルを回収できない場合もある．そこで，シミュレーション評価によって，実用上十分な回収効率が得られることを示す．

2．ＭＲＢ方式

2．1．ＭＲＢ方式の考え方

実時間ＧＣでは，データオブジェクトに対する参照数を実行時に計数することによって，そのデータオブジェクトに対する参照ポインタがなくなったことを検出し，データオブジェクトが使用していたメモリ領域を回収する．

ＫＬ１の並列処理において実時間ＧＣを効率的に実現する場合は，

① 参照数を管理するための情報が簡単であること，
② 参照数の更新操作が，データオブジェクトそのものにアクセスすることなく可能であること，
③ 参照数の更新操作をコンパイル時に抽出し，機械語に埋め込めること，

が重要である．

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．一方，ＫＬ１のプログラムにおいては，ＫＬ１プログラムのクローズ毎に，データオブジェクトへの参照数が増減するかどうかが検出できる．このため，ＫＬ１のゴールをリダクションする時に，ＭＲＢの値を管理することが可能である．

例えば，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無ければ，そのデータを消費したプロセスがデータオブジェクトを回収できる．さらに，構造体のようなデータオブジェクトの一部を書き替えた構造体を生成する場合，その構造体への参照数が"１"であれば，変更の無い部分を流用することができる．

一方，参照数が一旦"多数"になると，その後は参照するものが無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になる．

2．2．ＭＲＢの値の定義

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引数として，ポインタ（または間接ポインタの列）を通して参照される．

ＭＲＢは，ポインタに付加された１ビットの参照カウンタに相当する．ただし，定数の場合は，定数自体もＭＲＢを持つ．

ＭＲＢビットの値を○（'白い'と呼ぶ）または●（'黒い'と呼ぶ）であらわすことにする．その意味は基本的には以下の様になる．

○：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）

ただし，未定義変数セルへのポインタが持つＭＲＢの扱いは少し異なる．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２つのポインタがある場合が多い．そこで，未定義変数セルへのポインタが持つＭＲＢの値の意味は，以下の様になる．

○：参照される未定義セルに対するポインタの内，白いものが他に高々１個あるかもしれない．
　白いものがただ１つのとき，他に黒いものが１個以上あるかもしれない．．
●：参照される未定義セルへのポインタが他にもあるかもしれない．

上記のようにＭＲＢの値を意味付けることにより，構造体，変数セル，及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の許される組み合わせは図１～図３の９種となる．

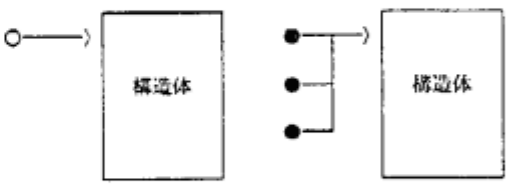
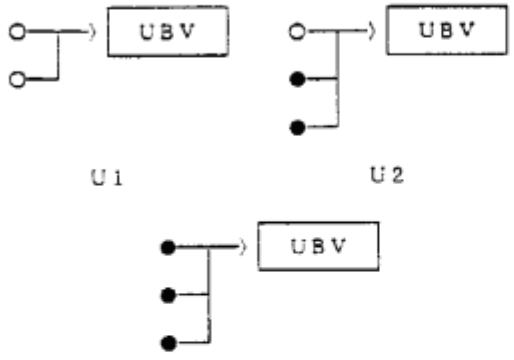
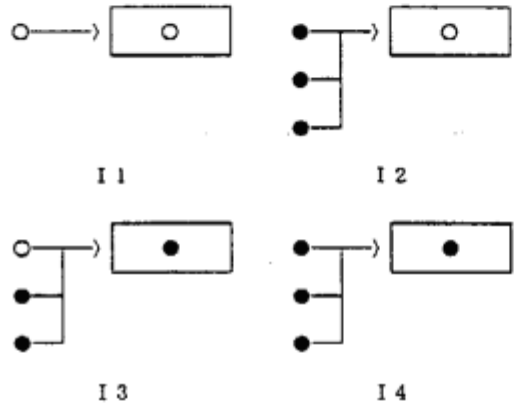


図１．構造体のＭＲＢ



U1　　U2　　U3

図2．未定義セルのＭＲＢ



I1　　I2　　I3　　I4

図3．具体化セルのＭＲＢ

２．３．ＭＲＢの更新操作

ＫＬ１の実行は以下の様な操作をメモリ領域に対して行いながら進む．このとき，前節で述べたＭＲＢの定義が満たされるように各ポインタまたは定数セルの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わたる分配
③ 変数のデレファレンス
④ ユニフィケーション
⑤ 構造体要素の参照

以下に，個々の場合についてのＭＲＢ管理の方法を説明する．

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，構造体を作ったりする場合である．

```
p :- true | q(X), r(X, [Car|Cdr]).        (1)
p :- true | q(X), r(X, [Car|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○である（上記Ｕ１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●となる（Ｕ３に相当）．構造体 [Car|Cdr] に対するポインタのＭＲＢはクローズ(1)，(2) とも○である．

② 変数，構造体などのゴール間にわたる分配

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移すだけなのでＭＲＢに変化はない．クローズ(2) では，２つ（以上）に分配するので，ＭＲＢは●にする（X が未定義変数であればＵ２あるいはＵ３に相当する）．

③ 変数のデレファレンス

変数のデレファレンスを行う場合で，デレファレンス結果のＭＲＢは間接ポインタの連鎖をたぐっていく途中で，間接ポインタのなかで１つでもＭＲＢが●のものがあれば●，それ以外の場合は○とする．

④ ユニフィケーション

ユニフィケーションにおけるＭＲＢの管理は，まず，２つの変数をデレファレンスし，どちらかのＭＲＢが●であれば●で具体化，さもなくば○で具体化する．表１に個々の場合を示す．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，Ｕ２○及び，Ｕ２●はそれぞれ○のポインタ／●のポインタを使ってユニフィケーションが行われることを示す．

表1．ユニフィケーション時のMRB管理

	S1	S2	U1	U2○	U2●	U3
S1	☆	☆	I1	I1	I3	I4
S2	☆	☆	I3	I4	I3	I4
U1	I1	I3	I1	I2	I3	I2
U2○	I1	I4	I2	I2	I4	I4
U2●	I3	I3	I3	I4	I3	I3
U3	I4	I4	I2	I4	I3	I4

⑤ 構造体要素の参照

構造体の要素を参照するとき，取り出した値のMRBは，構造体を指し示すポインタのMRBと要素自体のMRBのどちらかが●であれば●であり，両者が○であれば○である．

2．4．MRBを用いたGC方式

上記の様にKL1のデータをMRB管理すると，KL1の実行中に，あるデータへの参照が最後でかつMRBが○であれば，そのデータを回収し，メモリ領域を再利用できることになる．例えば，以下の場合に回収が可能となる．

(1) 変数のデレファレンス

変数のデレ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で（ポインタMRBは○），この間接ポインタのMRBが○であれば，デレファレンス時に回収できる（図4）．

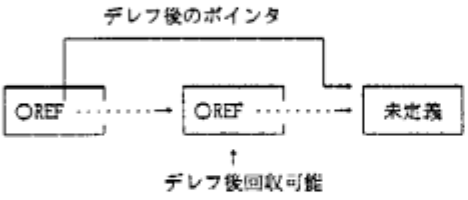


図4．変数のデレファレンス

(2) 構造体のユニフィケーション

受動部で，ゴールの引数の一つが構造体との単一化に成功し，そのクローズを選択することが確定したとき，ゴール引数として与えられた構造体のMRBが○であればその構造体を回収できる．例えば以下の様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MRBが○であれば回収できる．

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

(3) ボディに現われない変数とのユニフィケーション

ゴール引数がボディに現われない変数とユニファイし，そのクローズを選択することが確定したとき，そのゴール引数のMRBが○であれば回収できる．

```
p(foo, X) :- true | true.
             ~
```

(4) 回収される構造体の要素

ある構造体が回収されるとき，その構造体の要素も各々のMRBが○であれば回収してよい．例えば，以下のクローズで，第一，第二引数とも同じ構造体が与えられ，構造体同士のユニフィケーションが行われたとき，その構造体は回収され，さらに，要素も再帰的にたぐって（MRBが○であれば）回収できる．

```
p(X, Y) :- X = Y | true.
           ~~~~~
```

以上の回収操作の内，回収できるタイミングがコンパイル時にわかるものについては，コンパイラで回収のための命令を陽に出すことができる．ただし，実際の回収はそのクローズが選択されるとわかった時点で行なわなければならない．従って，回収のための命令は概念的には，コミットバーを越える時点に生成される．

3．抽象マシンと命令セット

MRBによる実時間GC方式を実現するためには，先に提案したKL1抽象マシンと命令セット［2］を変更，改良する必要がある．本章では，主に変更点について説明する．

3．1．抽象マシン

まず，MRBのフィールドを各KL1データセル

に設けなければならない．これにはタグフィールドの１ビットを充てることとした．次に，変数セル，構造体のためのメモリ領域は，動的な獲得，回収が容易にできるようにそれぞれフリーリストにより管理することにした．また，前章で示したガーベジセルの回収は，そのクローズが選択されるとわかってから行わなければならない．そこで，受動部の実行中に上記２．４（４）で示した回収可能な構造体要素を一時的に保持しておくためのスタックを新たに導入した（これをガーベジスタックと呼ぶ）．更に，構造体中の未定義セルの扱いについて変更を加えた．構造体中に未定義セルを直接割り当てるとその構造体のＭＲＢが○で回収できるときでも，未定義セルを指している他のパスがあるかも知れないので回収できないことになってしまう．これは，回収効率を著しく低下させるものと考えられる．そこで，未定義セルは構造体の外部に割り付け，構造体にはそれへの参照ポインタを入れることにした（図５）．

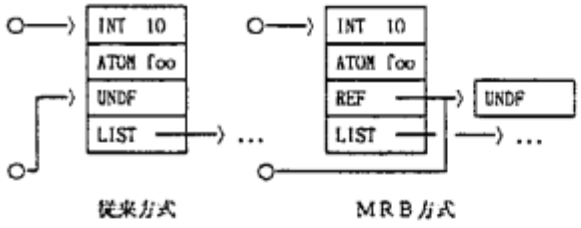


図５．構造体中の未定義セル

３．２．命令セット

ＭＲＢを正しく管理するために個々の命令の仕様を変更し，新たに幾つかの命令を導入した．新たに導入した命令は，２．３節で述べたＭＲＢ管理のための命令と，２．４節で述べたガーベジセル回収用命令である．また，従来からの命令はＭＲＢを正しく管理し，デレファレンス時に回収できる間接セルは回収するように変更した．

(1) 受動部において回収を抑制する命令

受動部の単一化を行う命令は，実行中に回収の条件を満たしたときに積極的にセルを回収しようとする．しかし，そのデータが能動部で使われている場合には，たとえＭＲＢが○であっても回収してはならない．そこで，次のような命令によって回収を抑制する場合がある．つまりレジスタXiが保持しているデータはＭＲＢが○でも回収しない．

```
wait_reused_value  Xi, Aj
read_reused_value  Xi
```

(2) 能動部において多重参照を指定する命令

能動部において，ＭＲＢを●にする命令である．mark は，ヘッド引数として受け取ったデータを能動部で複数のゴールに分配するときに用いる（上記２．３の②に相当する），残りの３つは，能動部で新たに３箇所以上から参照される変数を割り付けるときに用いる（上記２．３の① に相当する）．

```
mark  Xi
set_marked_variable Xi, Gj
put_marked_variable  Xi, Aj
write_marked_variable  Xi
```

(3) 回収のための命令

．２．４節で述べたガーベジセル回収のための命令である．なお，現在の命令セットはデレファレンス命令を陽に持っていないので，デレファレンス時の回収は個々の命令中で行なわれる．

```
collect_list Ai
  -- Ai から指されるリストのＭＲＢが○であれ
     ばリストセルを回収する．
collect_value Ai
  -- Ai から指されるデータのＭＲＢが○である
     限り再帰的に回収する．
collect_stack
  -- ガーベジスタックから指されているデータ
     を回収する．
```

(4) コンパイル例

ＭＲＢ－ＧＣ用命令を生成するコンパイル例を図６に示す．

```
p([X|Y], X) :- true | q(X).

p/2: wait_list  A1          % 第一引数はリスト？
     read_variable X3              % car を読む
     read_variable X4              % cdr を読む
     wait_reused_value X3, A2
                          % 第二引数の単一化
     collect_list  A1           % リストの回収
     collect_value X4       % ボイド変数の回収
     collect_stack     % ガーベジスタックの回収
     put_value  X3, A1
     execute    q/1


p:- true | q([X|Y],Y),r(X,[Y|X]).

p/0: create_goal r/2     % ゴールレコードの生成
     set_list A1, G1  % 第一引数のリストの設定
     write_marked_variable A2      % car を書く
     write_value X3                % cdr を書く
     set_value A1, G2          % 第二引数の設定
     enqueue_goal r/2   % ゴールレコードの待避
     put_list A1
     write_value X3
     write_value A2
     execute q/2
```

図６．ＫＬ１プログラムとコンパイル例

４．ＭＲＢ－ＧＣの評価

前節までで述べた，ＭＲＢによる実時間ＧＣ方式の回収効率を調べるため，コンパイラを改良し，プログラミング言語'C'でＫＬ１処理系のエミュレータを作成して評価を行った．評価プログラムとしては，以下の４種のプログラムを用いた．

① 素数生成プログラム

リストを用いた典型的なストリーム通信によって素数を次々と生成していくプログラムである．整数列を生成するプロセスと，一つの素数に対してその倍数を削除するプロセスから成っている．

② クイーン

有名なクイーン問題で，全解探索プログラムの一例である．

③ レイヤードクイーン

② と同様のクイーン問題であるが，［８］で提案された手法によるものである．構造体を多く共有することに特徴がある．

④ ＢＵＰ

ボトムアップパーサで，４種の中では最も大きく，実用的なプログラムで近い．また，ＡＮＤ並列ＧＨＣに書き換えられた，ＯＲ探索型のプログラムの一例である．

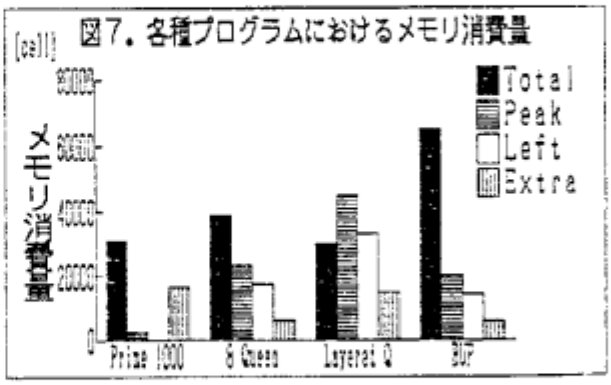
測定項目として，各々のプログラムを実行するために実行された機械語命令の総数，必要とするメモリ量，及び回収用命令の実行回数と，実際に回収されたセルの量を測定した．表２に命令の実行総数，表３に必要としたメモリ量，また図７にそのグラフを示す．

表２．実行命令数の特性

プログラム	総命令数 (A)	WRITE_VAR (B)	ＭＲＢ命令 (C)	B/A	C/A
Prime 1000	298,886	16,684	16,856	.056	.056
8 Queen	667,974	6,544	71,560	.010	.107
Layered Q	702,905	15,112	47,256	.021	.067
ＢＵＰ	613,805	6,136	43,877	.010	.071

表３．メモリ使用率

Cells	Total (A)	Peak (B)	B/A	Left (C)	C/A	Extra
Prime 1000	31,354	2,939	.062	0	.000	—
LIST	15,956	999	.063	0	.000	—
VARIABLE	16,126	1,001	.063	0	.000	16,684
8 Queen	39,433	23,496	.600	17,607	.447	—
LIST	14,208	6,861	.483	6,671	.470	—
VARIABLE	17,561	9,774	.557	4,265	.242	6,544
Layered Q	30,233	15,345	1.50	33,007	1.09	—
LIST	15,112	15,112	1.00	11,000	.728	—
VARIABLE	15,121	15,121	1.00	11,007	.728	15,112
ＢＵＰ	65,264	20,092	.307	14,331	.220	—
LIST	19,162	7,968	.416	5,223	.273	—
VARIABLE	33,076	4,156	.126	3,885	.126	6,136



(1) 命令実行特性

表２は，各々のプログラムを実行した時の命令の

総数，命令'Write_variable'の実行回数，ＭＲＢ用命令の総実行回数を表す．３．１節で説明したように，ＭＲＢによる実時間ＧＣでは構造体の外部に未定義セルを割り当てている．このため，'Write_variable'の数は，ＭＲＢによる実時間ＧＣを実現するにあたって余分に必要とされるメモリ領域の大きさに相当する．ＭＲＢ用命令とは，'Collect_list','Collect_value'，'Collect_stack'，'Mark'命令の実行回数の総和である．表で，Ｃ／Ａが，ＭＲＢ用命令の全実行命令数に占める割合を表わす．これらの４つのプログラムでは，６~１０％程度であった．ＭＲＢ用命令以外の命令もＧＣのために機能拡張が行われているため，これだけで処理速度の低下を判断することはできないが，ＭＲＢ管理のためのハードウエアを用意すれば実行速度の低下は，ＭＲＢによるＧＣを行なわない場合に比べて５割以内に抑えられると考えている．

(2) メモリ使用率

表３は，各々のプログラムを実行するのに必要としたメモリ量を表している．表３で，Total は，ＭＲＢによるＧＣを行わない場合にプログラムを実行するために必要とされる総メモリ量を表わす．Peak は，ＭＲＢによるＧＣを行った場合にプログラム実行のために必要とされるメモリ領域の大きさ，Left は，実行後に残ったメモリ領域の大きさを示す．これらの算出式をまとめると以下の様になる．なおLIST はコンスセルの数，VARIABLE は変数セルの数であり，Extra は，構造体中に変数セルを置けないことによるメモリの余分な使用量を示す．従って，これを差し引かないとＭＲＢによるＧＣを行わない場合の正しいメモリ使用量とならない．

```
Total = LIST * 2 + VARIABLE - Extra
Peak  = LIST * 2 + VARIABLE
Left  = LIST * 2 + VARIABLE
```

表３より，素数生成プログラムではＭＲＢによるＧＣを行わない場合と比較して，７％程度のメモリ量で実行でき，しかも実行後はすべてのメモリ領域が回収されて，ガーベジセルは全く残っていない．ＫＬ１のプログラムがこの素数生成プログラムの様なストリームを多用したプログラミングスタイルで書かれるであろうことを考えれば，これは非常によい傾向である．また，クイーン，ＢＵＰプログラムもそれぞれ，６割，３割程度のメモリ量で実行ができ，実行後ゴミとして，５割弱，２割強のメモリが残っていた．これらも，ほぼ満足のいく値である．

一方，レイヤードクイーンの場合には，ＭＲＢによるＧＣを行ったほうがメモリをより多く消費する．この増分は，構造体の変数セルを構造体の外に採ることのオーバヘッドに等しい．すなわち，メモリの回収は，実行中はほとんど行われず，実行後，解の出力時のデレファレンスによる間接ポインタの回収のみであったことを意味していると考えられる．このプログラムは，先にも述べたように構造体を多く共有する．ＭＲＢによるＧＣでは，一旦多重参照（●）となったものは，後で単一参照となっても回収できない．レイヤードクイーンは，このＭＲＢによるＧＣの欠点が顕著に表われた場合と言える．

(3) 命令別の回収率

表４に，命令別の回収セル数とデレファレンスに依る回収セル数を示す．

表4．回収セルの内訳

	Primes 1000	8 Queens	BUP
collect_list 呼出し	33,024 16,512	6,410 31,221	20,176 14,453
collect_value 呼出し	0 172	13,906 18,307	3,700 13,252
collect_stack 呼出し	0 0	0 0	0 2,493
passive_deref active_deref	16,685 0	2,177 0	25,689 1,819

表４からわかるように，プログラムによって回収される命令にばらつきがあることがわかる．素数生成のプログラムでは，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．クイーンでは，'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．一方，'Collect_stack' では，全く回収していない．これは，ＫＬ１のユニフィケーションの多くが単方向であり，受動部では，複雑なユニフィケーションがあまり行われていないことを示していると考えられる．

5. まとめと今後の課題

ＭＲＢを用いた実時間ＧＣ方式について報告した．テストプログラムを用いた実験では，十分な回収効率を期待できることがわかった．

今後，ＭＲＢを用いた実時間ＧＣを並列マシンに実装するにあたって検討すべき項目をまとめると以下の様になる．

(1) 現在は回収用の命令を陽に出しているので，動的な命令実行数が増えている．テストプログラムでは約２割の増加であった．そこで，実行命令数をさらにおさえるような命令体系の検討が必要である．

(2) ＭＲＢ管理のために個々の命令自身の操作が重くなる．ＭＲＢ管理を高速に行うハードウエアの検討が必要である．

(3) ＭＲＢを用いて実時間ＧＣを行うと，フリーリスト管理の効果によりワーキングセットを縮小でき，キャッシュヒット率の向上を期待できる[7]．このようなメモリアクセスの局所性の向上はマルチプロセッサシステムにおいてより効果が大きい．ＭＲＢの並列キャッシュに及ぼす影響について検討を進める必要がある．

(4) 一方，変数セル，コンスセルをフリーリスト管理するコストは小さくない．ＭＲＢ方式に適したメモリ領域の高速な獲得，解放機構の検討が必要である．

(5) ＭＲＢでは，単一参照（○）のデータは回収できるが一旦多重参照（●）となると，後で単一参照に戻ったとしても回収できない．従って，多重参照データを多く作るプログラムでは，やはり，一括型のＧＣが必要となる．特に，マルチプロセッサシステムに適した一括型ＧＣの検討が必要である．この時，一括型ＧＣを工夫すればＭＲＢが●で単一参照のものを○に変えることができる．筆者らは，既にＧＣ時にＭＲＢの付け替えを行う方式を考案し，エミュレータに組み込み，実験を行っている[6]．この評価結果については，また別の機会に報告したい．

謝辞

日頃御指導頂くＩＣＯＴ４研，内田俊一室長に感謝します．また，貴重なコメントを頂いたＩＣＯＴ，ＰＩＭ／ＭＰＳＩ－ＷＧの方々，４研ＰＩＭ／ＭＰＳＩグループ及び関連メーカの方々に感謝します．とりわけ，ＩＣＯＴ４研，後藤厚宏氏には，ＧＣの評価及び本論文をまとめるにあたって多大な援助を頂いた．特に記して謝意を表わしたいと思います．

参考文献

[1] T.Chikayama and Y.Kimura : "Multiple Reference Management in Flat GHC", Proc. of ICLP'87, also as ICOT-TR248

[2] Y.Kimura and T.Chikayama : "An Abstract KL1 Machine and its Instruction Set", Proc. of SLP'87, also as ICOT-TR246

[3] 木村他：並列推論マシンＰＩＭ－ＫＬ１の抽象命令仕様とコンパイラ－，第３４回情処全大 ２Ｐ－１，（昭和６２年前期）

[4] 近山他：ポインタタグ（ＭＲＢ）による多重参照管理方式，第３５回情処全大 ２Ｑ－５，（昭和６２年後期）

[5] 木村他：ＭＲＢによる多重参照管理方式－ＫＬ１処理系における実時間ガーベジコレクション方式－，第３５回情処全大 ２Ｑ－６

[6] 宮内他：ＭＲＢによる多重参照管理方式－ＫＬ１処理系における一括型ＧＣの特性評価－，第３５回情処全大 ２Ｑ－７

[7] 西田他：ＭＲＢによる多重参照管理方式－ＫＬ１処理系におけるキャッシュ特性の評価－，第３５回情処全大 ２Ｑ－８

[8] 奥村他：レイヤードストリームを用いた並列プログラミング，Proc of LPC'87, ICOT, June 1987.