Vol. 30 No. 9 情報処理学会論文誌 Sep. 1989

FGHC 処理システムのメモリ使用特性と世代別ガーベジ・コレクション†

小沢年弘†† 細井聡†† 服部彰††

並列論理型言語 FGHC は，プログラムの並列性を自然に記述できる言語であるが，その実行のために多くのメモリを消費する．FGHC を効率的に処理するためには，効率的なメモリ管理方式を構築する必要がある．本論文では，FGHC のメモリ使用特性に基づき，この言語向きの世代別ガーベジ・コレクション (GC) 方式を提案し，それを試作，評価した結果を述べる．我々は，FGHC 並列処理系を密結合マルチ・プロセッサ上に作成し，その上で，FGHC プログラムのメモリ使用特性を測定した．その結果，アクティブ・データの比率が低いこと，非常に短いライフ・タイムのデータとより長いライフ・タイムを持つデータの二種類に，はっきり分割できることを明らかにした．これらの性質を利用した効率的なメモリ管理方式として，二つだけに世代を限った二世代 GC 方式を提案している．この GC は，第一世代領域の GC によりライフ・タイムの短いデータをふるいにかけ，長いライフ・タイムを持つデータを第二世代領域に集める方式である．さらに，この方式を試作し性能評価を行った．その結果，第一世代領域を持つことによりライフ・タイムの短いデータを回収できること，第二世代の GC を行わず第一世代の GC だけを行った場合でも，第一世代領域を小さく保ったままで，40～95%のゴミの回収ができることを確かめた．

1. はじめに

計算機の利用範囲をより高度でより大規模な問題へ広げるために，知識処理を並列に行わせる研究が盛んに行われている．

我々は，第5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 の開発を進めている．PIM 上に実装される核言語 KL 1 は，FGHC に OS 記述のための機能を付加したものである．FGHC は，GHC[1] に制限を加えた，AND 並列論理型言語である．

この言語は，論理型と並列実行セマンティクスを持つことを意図して設計され，プログラム中の並列性を細かいレベルで自動的に抽出できる言語である．しかし，その効率的な実現のためにはいくつかの問題がある．その一つとして，並列実行ではスタックによる実現が困難なため，すべての変数をヒープ領域に割り付けざるを得ず，変数への単一代入の性質と相まってメモリを多量に消費する問題がある．そのために，効率的なメモリ管理方式を実現することが大きな課題になっている[2]．

この問題を解決するために，参照カウンタによる即時型ガーベジ・コレクション (GC) である MRB[3] や LRC[4] 方式が提案されている．しかし，参照カウンタによる方式は，カウンタの管理のオーバヘッドや環状になったゴミが回収できないなどの問題点がある．

本論文では，FGHC 並列処理系を使い，FGHC プログラムのメモリ使用特性を明らかにしている．この結果に基づいて，FGHC 向きのメモリ管理方式として二つの世代を持つ二世代ガーベジ・コレクション方式を提案するとともに，その方式の試作，評価について述べる．

2. FGHC

FGHC は，効率的な実現のために GHC に制限を加えた並列論理型言語であり，そのプログラムは，次のような文(節)の集まりである．

Head :-Guard1, Guard2, … | Body1, Body2, ….

Head, Guard n ($n=1, 2, \cdots$), Body n ($n=1, 2, \cdots$) は，それぞれ素命題を表し，Head, Guard n, Body n は，それぞれヘッド，ガード・ゴール，ボディ・ゴールと呼ばれる．また，ヘッドとガード・ゴールをまとめてパッシブ・パート，ボディ・ゴールをまとめてアクティブ・パートと呼ぶ．

GHC の実行は，あるゴールに対して，これとユニフィケーションできるヘッドを持ち，かつそのすべてのガード・ゴールが成立する節が選ばれることから始まる．ただし，このときゴールに含まれるいかなる変数も具体化してはならない．この節を選ぶことをコミットと呼ぶ．

コミットできる節がないときには，二種類の可能性がある．第一は，ゴールの未束縛の変数がある具体値

† The Memory Usage Characteristics of FGHC System and Generation Type Garbage Collection by TOSHIHIRO OZAWA, AKIRA HOSOI and AKIRA HATTORI (Fujitsu Limited).

†† (株)富士通

に決まればコミットできる場合である．このときには，そのゴールの実行は中断される（これをサスペンドと呼ぶ）．後で未束縛変数が具体化されたときに再びゴールの実行が試される（これをリジュームと呼ぶ）．もう一つの可能性は，ゴールの未束縛変数にどんな値が代入されても，コミットする可能性のある節がない場合である．この場合には，ゴールの実行は失敗する．

ある節がコミットされると，その節のボディ・ゴールが並列に実行され，すべてのゴールが処理されるまで実行が続けられる．

ガード・ゴールに組み込み述語しか許さないという制限を持たせたものが FGHC である．この制限によりコミットの判断が簡単化され，効率的な実現が可能になった．

3. FGHC 並列処理系

FGHC 処理系は何件か実現されているが[6)~7)]，我々は，並列実行時におけるメモリ使用特性を測定するために，マイクロ・プロセッサを共有バスに結合した，密結合メモリ共有型並列計算機（Sequent Symmetry CPU 80386）上に FGHC 並列処理系を作成した．この処理系は，FGHC ソース・プログラムを KL 1-b[8)] と呼ばれる中間コード列にコンパイルし，この中間コードを解釈実行するエミュレータ方式である[9)]．処理系の構成を，**図1**に示す．エミュレータを複数発生させ，それぞれを実際のプロセッサ（PE）に割り当てることにより並列実行を行う．コンパイルされたコードは，全 PE により共有され，共有メモリの特定の領域におかれる．

パッシブ・パートは，コンパイラにより KL 1-b コード列に展開され直接実行されるが，ボディ・ゴールは，ゴール・レコードと呼ばれる制御ブロックにより表される．このレコードには，引数や対応する節のコンパイル・コードへのポインタなどが含まれる．ゴール・レコードは，ゴール・レコード用の領域にとられ，フリー・リスト管理により再利用されている．

節がコミットされるとボディ・ゴールに対応したゴール・レコードが生成され，スケジューリング・キューにつなげられる．エミュレータは，スケジューリング・キューからゴール・レコードを取り出し，コミットできる節を探すため，定義節のパッシブ・パートを実行する．コミットする節があればそのボディ・ゴールの生成を行い，失敗すれば全体を失敗させる．

その他の場合には，つまりまだコミットできないが，未束縛変数の具体化によりコミットする可能性が残っているならば，その変数からこのゴール・レコードをポイントさせる（フックする）．将来，その変数が具体化したとき，フックしているゴール・レコードをスケジューリング・キューに戻すことにより，実行を再開させる．

ゴール・レコードは，フリーリスト管理され，ゴール生成時にフリーリストから獲得され，ゴール実行終了時にフリーリストにもどされることにより再利用されるが，変数セルやリスト，アトムなどのデータは，ヒープと呼ばれるすべてのエミュレータがアクセス可能な領域に，動的に割り付けられる．ヒープは，GC により回収され，再利用される．

4. 処理系のスケジューリングと負荷分散方式

FGHC を効率的に実行するためには，ゴール実行のサスペンド回数が少なくなるようなスケジューリングとともに，すべてのエミュレータにゴールをいつも供給できる負荷分散方式が重要となる[10)]．

ここで，実装したスケジューリングと負荷分散は，次のような方式である[11)]（**図2**参照）．

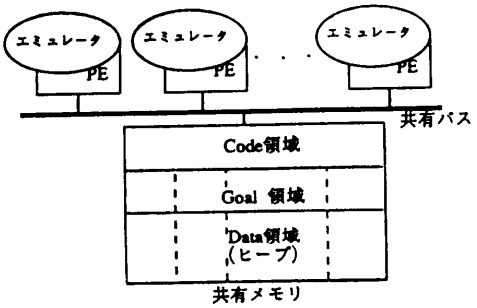

図 1 処理系の構成
Fig. 1 Structure of FGHC processor.

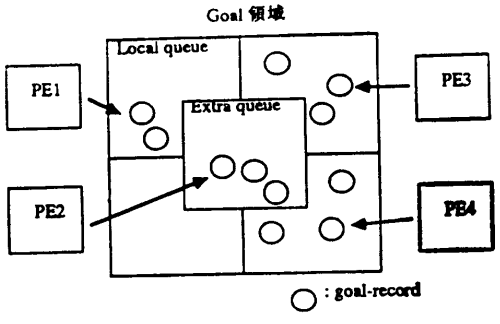

図 2 負荷分散方式
Fig. 2 Load balancing method.

1) 各エミュレータは，それぞれスケジューリング・キューを持つ．このキューは他のエミュレータからはアクセス不可能である．これを，ローカル・キューと呼ぶ．

2) すべてのエミュレータからアクセスできるスケジューリング・キューがシステムに一本ある．このキューをエクストラ・キューと呼ぶ．また，このキューの長さを保持するレジスタ EXTRA-LENGTH を設ける．

3) 負荷分散方式は，次のとおり．

各エミュレータは，各自のローカル・キューからゴールを取り出し実行する．実行に伴って生成されたゴールは各自のローカル・キューの先頭につながれる．各自のローカル・キューにゴールがなくなるとエクストラ・キューにつながれたゴールを取り，実行する．

各エミュレータは，ゴール実行ごとに各自のローカル・キューの長さが，EXTRA-LENGTH/CONSTANT よりも長くないか調べ，長ければ上記の条件を満たすまでローカル・キューの先頭からエクストラ・キューへゴールを移す．

この方式は，基本的には depth-first なスケジューリングであるが，負荷分散のためにエクストラ・キュー経由で移動するゴールがあるために完全な depth-first とはならない．

CONSTANT の実際の値は，実際のプログラムを走らせて実行時間が短かった 4 に設定してある．

5. FGHC のメモリ使用特性

GC を意図的に起こすことにより，FGHC プログラムのメモリ使用特性を調べた．ここで用いた GC 方式は，次のようになっている．

1) あるプロセッサが GC を起動すると，すべてのプロセッサがゴールの処理を中断する．

2) 各プロセッサは，あらかじめ決められた順番で逐次に，それぞれのローカル・キュー，レジスタをルートにしてコピー法 GC を行う[12]．エクストラ・キューのようなシステム共有資源は，始めに GC を行うプロセッサが処理する．

3) すべてのプロセッサの GC が終了すると，ゴールの処理を再開する．

メモリの使用特性として，いくつかのベンチマーク・プログラムを使い，次の項目を測定した．

1) 最大アクティブ・データ率

最大アクティブ・データ量／全割り付けデータ量．

アクティブ・データとは，ある時点で，使用中のヒープ中のデータであり，割り付けデータとは，ヒープに割り付けられたデータである．

2) データのライフ・タイム

A〜$(A+a)$ 番目の間に割り付けられたデータの生存率の時間変化．実行の初めや終りなど，割り付けられる時期により生存率が変わる可能性があるので，いくつかの A の値において測定した．ただし，

生存率＝アクティブ・データ量／
割り付けたデータ量 a．

これらの値は，約 2Kword のデータを新たに割り付けるごとに，GC を起こすことにより測定した．

5.1 ベンチマーク・プログラム

測定に使用したプログラムは，以下のものである．

1) BUP

ボトムアップ・パーザ．OR 並列問題．

2) DB

メタ・インタプリタによるデータベース・サーチ[13]．

3) MAXF

ネットワークの最大流量問題．ネットワークの各ノードをプロセスとし，その間でメッセージ通信をする．

各プログラムの諸性質を**表 1** に挙げる．ただし，台数効果とは，PE 1 台での実行速度に対する比．

表 1 ベンチマーク・プログラムの性質
Table 1 The characteristics of benchmark programs.

		BUP	DB	MAXF
リダクション数		36K	724K	69K
全割り付けデータ量 (word)		73K	1700K	266K
台数効果	1PE	1.0	1.0	1.0
	8PE	5.6	6.6	6.5
	16PE	9.8	11.4	13.5
サスペンド率	1PE	1%	11%	42%
	8PE	38%	26%	42%
	16PE	35%	26%	41%

表 2 最大アクティブ・データ率
Table 2 Maximum active data/Total allocated data.

		BUP	DB	MAXF
最大アクティブ・データ率	1PE	21%	1.5%	29%
	8PE	19%	1.5%	8%
	16PE	18%	1.5%	7%

サスペンド率＝サスペンド回数／
リダクション回数.

5.2 アクティブ・データ

各プログラムの最大アクティブ・データ率を表2に示す.

1台での実行ではデータを生成するゴールが続けて動くようなプログラムでは，複数台で実行することにより最大アクテイブ・データ量は下がるが，OR 並列のようにデータの生成と消費が交互に行われるようなプログラムでは，複数台で実行することの影響はほとんどない.

また，FGHC では，アクティブ・データ率は低いことが分かる．このようにアクティブ・データ率が低いので，ページ・フォールトを減らし高い効率を得るために，FGHC のメモリ管理には，アクティブ・データのコンパクションの機能が必須である.

5.3 データのライフ・タイム

各データに割り付け時期を示す世代フィールドを設

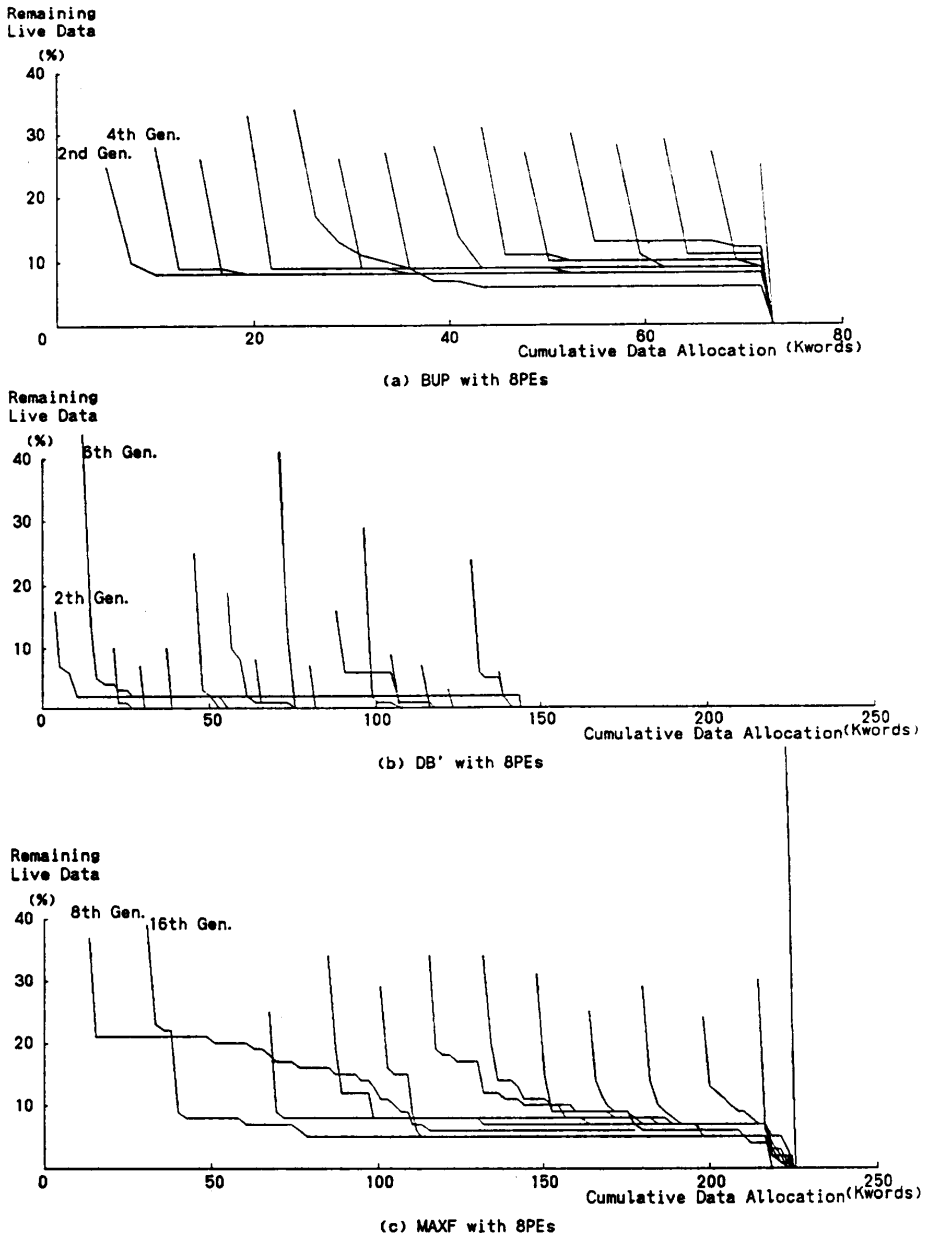

図3 データの生存率の変化
Fig. 3 Life time of data.

け，データのライフ・タイムを測定した．第N世代のデータとは，第$(N-1)$回の GC が終了してから，第N回の GC が始まるまでに割り付けられたデータである．**図3**に実行にともなう生存率の変化を示す．

(b)の DB′ は，DB のデータベースの規模を小さくしたものである（リダクション数 54 K，全割り付けデータ量 144 Kwords）．横軸は，それまでに割り付けたデータ量を目盛としている．各ラインは，各世代ごとのデータの生存率の変化を示す．

どの世代の生存率も，初めのある一定期間で急激に低くなり，その後，低い水準ではあるがほぼ一定値を保っている．これは，FGHC プログラムの次のような性質によるものと考えられる．つまり，多くのデータは，ゴールからゴールへ情報を伝えるためだけに使われるが，そのゴールのライフ・タイムも短いので，多くのデータは，短いライフ・タイムを持つ．その他のデータは，結果を蓄えるものなどで比較的長いライフ・タイムを持っている．

この結果から FGHC のデータは，そのライフ・タイムが非常に短いものとより長いものの二種類にはっきりと分けられることが分かる．ただし，細かく見ると MAXF では，定常状態になるまでの時間が比較的長い．

6. FGHC 向きの世代別 GC 方式

ここで，我々は FGHC のデータ使用特性を考慮した世代別 GC 方式[14)~17)]を提案するとともに，その試作，評価について述べる．

6.1 二世代ガーベジ・コレクション

データの生存率が低いことから，GC の方式として，アクティブ・データ量に比例した時間で済むコピー法に基づく GC が有利である．しかも，ライフ・タイムが，極端に二種類に分かれることから，ライフ・タイムによりデータを二世代に分けた世代別 GC 方式（二世代 GC 方式）が適していると考えられる．世代をそれ以上分けても，世代の管理が繁雑になるだけである．

図4に二世代 GC の構成を示す．第一世代領域は，短いライフ・タイムのデータをふるいにかけるためのもので，第二世代領域は，長いライフ・タイムを持つデータを格納するための領域である．それぞれの世代の領域は，その世代の中でコピー法 GC を行うために，二つの空間（新空間と旧空間）に分けられる．また，第二世代領域から第一世代領域をポイントしているデータを管理するために，それらのデータのアドレスを格納するスタック（GC スタック）を設ける．このスタックは第一世代の GC を行う場合のルートとなる．

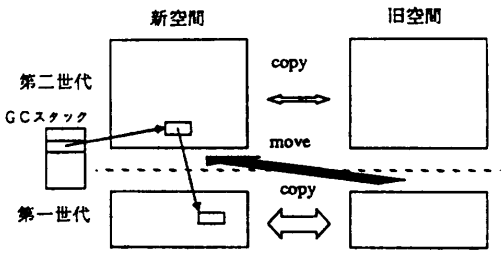

図4 二世代 GC の構成
Fig. 4 Structure of 2 generation GC.

世代の管理や世代別の GC などは次のように行われる．

1) リダクション時や GC 時に第二世代領域から第一世代領域へのポインタが生成されたならば，そのポインタを格納しているデータのアドレスを GC スタックに積む．

2) 新しいデータは第一世代領域に割り付けられる．

3) 第一世代領域が使い尽くされると，第一世代の GC が起動される．第一世代の GC におけるルートは，各スケジューリング・キューと GC スタックである．

4) 第一世代の GC において，第一世代領域に割り付けられた後ある時間以上経っても生きているデータは，第二世代領域に移される．その他のアクティブ・データは，第一世代領域の新空間に移される．

5) 第二世代の GC は，第一世代領域も合わせて行われる．つまり，各スケジューリング・キューをルートとして，普通のコピー法 GC を行う．ただし，コピー先は，データが現在属する世代の新空間である．

第一世代の GC を第二世代の GC よりも頻繁にかけることにより，効率のよい GC を実現する．

割り付け後第二世代に移されるまでの時間は，実際には，その後ある量T以上のメモリ割り付けが行われるまでで測られる．つまり，あるデータ割り付け後，さらにTワード以上割り付けるところまで処理が進んでも，そのデータが生きているならば第二世代へ移される．この最低割り付け量Tを Threshold と呼ぶ．

データ割り付け後，何ワード割り付けたかの管理は厳密に行う必要はなく，例えば次のように行えばよ

い．つまり，データにそのデータが何回目の GC 後に割り付けられたデータであるかを示す世代フィールドを付け，さらに各 GC 間の実行において何ワードの割り付けを行ったかを記録する．それにより各 GC 後割り付けたデータの総量が分かるので，ある GC より以前に割り付けられたデータは第二世代に移せばよい．また，もし第二世代に移すための最低割り付け量 T ごとに GC を起動するならば，データの世代フィールドは必要なくなり，1回 GC を経験したデータを第二世代に移すようにすればよい．GC を経験しているかどうかは，ヒープの空領域を管理するレジスタの直前の GC 終了後の値とデータのアドレスを比較するだけで判定できる．

6.2 二世代 GC の評価

あるデータが割り付けられてから，第二世代へコピーされるまでに新たに割り付ける最低量 T と第二世代にコピーされるデータ量の関係を図5に示す．このとき，第二世代の GC は行わないで測定した．縦軸は，第二世代にコピーされたデータ量の全割り付けデータ量に対する割合である．

第一世代でふるいにかけるほど，第二世代にコピーされる量は減少する．BUP, DB では，T の値がある程度の大きさになるまでの間コピー量は急激に減少し，それ以後はあまり変化しなくなる．これは，第一世代でふるいをかける時間に最適な時間があることを意味する．ある程度以下では，第二世代が早く消費されてしまい，ある程度以上では，第一世代のなかで何回もコピーされるデータを生じさせ GC の効率を落とす．この二つのプログラムの場合には，T の値として 8〜32 Kword 程度が最適である．BUP, DB は，第一世代の GC だけで，それぞれ 80%，95% 以上のゴミを回収している．

しかし，MAXF では，第二世代へのコピー量が比較的大きい．これは，このプログラムのデータのライフ・タイムのばらつきが大きいためであり，第二世代の GC の頻度を上げる必要がある．ただし，$T=20$ Kword とすれば，ガーベジの 40% 以上は，第一世代の GC のみで回収できる．この値は第二世代の GC をまったく行わないときの値である．第二世代の GC を行わないと，一度データが第二世代領域にコピーされるとそれが後にゴミになっても，それからポイントされるすべてのデータが第二世代領域にコピーされてしまう．したがって，第二世代の GC を低い頻度でも行えば，第一世代 GC の回収率はより向上することが見込まれる．

これらのことから，二世代だけを持つ世代別 GC により効率的なメモリ管理が可能になると考えられる．

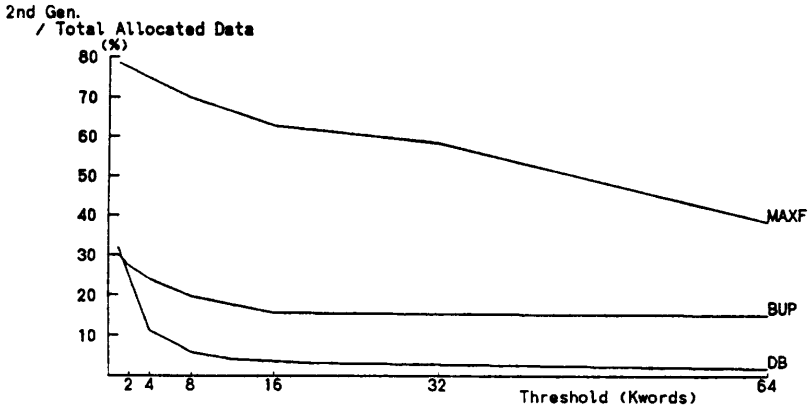

図 5 第二世代へのコピー量
Fig. 5 Amount of data copied to 2nd Generation.

7. まとめ

FGHC 並列処理系を用いて，FGHC プログラムのメモリ使用特性を評価した．その結果，

1) アクティブ・データ率が低いこと

2) データは，そのライフ・タイムが短いものと長いものがはっきり二つに分けられること

が分かった．

これらの性質より，FGHC 向きの GC 方式として，二つの世代だけを持つ二世代 GC を提案し，その評価を行った．その結果，

1) ライフ・タイムの短いデータをふるいにかけるために，世代に分けることが有効である．

2) 第一世代から第二世代への移動を，割り付け後さらに 20 Kword 程度新たにデータを割り付けた以降にすると，第二世代の GC を行わず第一世代の GC だけを行った場合でも，よい場合は 95% 以上，悪い場合でも 40% 程度のゴミの回収が行えた．

さらに，第二世代の GC を低い頻度ででも行えば，第一世代の GC の回収率はより向上することが見込まれる．

今後，さらに正確な評価を行うために，より多くのプログラムでテストする必要がある．また，第二世代の GC をどのくらいの頻

度で起こすべきかなどの課題がある.

謝辞 日頃御指導いただく棚橋部門長，林部長，ならびに，貴重なコメントをいただいた研究室諸兄，ICOT 第四研究室のメンバ諸氏に感謝します.

参考文献

1) Ueda, K.: Guarded Horn Clauses, Technical Report TR-103, ICOT (1985).
2) 佐藤正俊，後藤厚宏：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, pp. 195-202 (1989).
3) Chikayama, T. and Kimura, Y.: Multiple Reference Management in Flat GHC, *Proc. Int. Conf. on Logic Programming '87*, pp. 276-293 (May 1987).
4) Goto, A., Kimura, Y., Nakagawa, T. and Chikayama, T.: Lazy Reference Counting: An Incremental Garbage Collection Method for Parallel Inference Machines, *Proc. Int. Conf. on Logic Programming '88*, pp. 1241-1256 (Aug. 1988).
5) ICOT 第 4 研究室：PDSS—言語仕様と使用手引き—，ICOT TM-437 (1988).
6) Ichiyoshi, N., Miyazaki, T. and Taki, K.: A Distributed Implementation of Flat GHC on the Multi-PSI, *Proc. Fourth Int. Conf. on Logic Programming*, pp. 257-275 (1987).
7) Sato, M. and Goto, A.: Evaluation of the KL1 Parallel System on a Shared Memory Multiprocessor, *IFIP WG 10.3 Working Conference on Parallel Processing in Pisa, Italy*, pp. 305-318 (1988).
8) Kimura, Y. and Chikayama, T.: An Abstract KL 1 Machine and Its Instruction Set, *Proc. 1987 Symp. on Logic Programming*, pp. 468-477 (1987).
9) 細井，小沢，服部：密結合マルチプロセッサ上の KL 1 並列処理系の評価，第 38 回情報処理学会全国大会論文集，pp. 1009-1010 (1989).
10) Sugie, M., Yoneyama, M. and Goto, A.: Analysis of Parallel Inference Machines to Achieve Dynamic Load Balancing, *Proc. Int. Workshop Artif. Intell. for Industrial Applications*, pp. 511-516 (1988).
11) 安里，岸本，小沢，細井，林，服部：GHC 処理系における負荷分散方式の検討，*CPSY*, 88-46 (1988).
12) Backer, H. G.: List Processing in Real Time on a Serial Computer, *Comm. ACM*, Vol. 21, No. 4, pp. 280-294 (1978).
13) 北上，横田，服部：知識処理向き並列推論エンジン，*CPSY*, 88-50 (1988).
14) Lieberman, H. and Hewitt, C.: A Real-Time Garbage Collector Based on the Lifetimes of Objects, *Comm. ACM*, Vol. 26, No. 6, pp. 419-429 (1983).
15) Ballard, S. and Shirron, S.: The Design and Implementation of VAX/Smalltalk-80, *Smalltalk-80: Bits of History, Words of Advice*, Krasner, G. (ed.), pp. 127-150, Addison-Wesley (1983).
16) Ungar, D.: Generation Scavenging: A Non-disruptive High Performance Storage Reclamation Algorithm, *Proc. ACM SIGSOFT/SIGPLAN Software Engineering Symp. on Practical Software Development Environments*, pp. 157-167 (1984).
17) Nakajima, K.: Piling GC—Efficient Garbage Collection for AI Languages—, *Proc. IFIP WG 10.3 Working Conference on Parallel Processing*, pp. 201-204 (1988).

(平成元年 3 月 24 日受付)
(平成元年 6 月 13 日採録)

小沢 年弘 (正会員)

昭和 35 年生. 昭和 57 年横浜国立大学工学部電気工学科卒業. 昭和 59 年同大学院修士課程修了. 同年，(株)富士通研究所入社. 並列計算機，並列言語等に興味を持つ.

細井 聡 (正会員)

昭和 37 年生. 昭和 61 年早稲田大学理工学部電気工学科卒業. 同年富士通研究所入所. 以来並列処理，特に並列言語の研究に従事.

服部 彰 (正会員)

昭和 24 年生. 昭和 47 年大阪大学工学部電気工学科卒業. 昭和 49 年同大学大学院工学研究科修士課程修了. 同年(株)富士通研究所入社. 現在，人工知能研究部第 3 研究室長. 記号処理マシン，並列処理マシンの研究開発に従事. 電子情報通信学会，人工知能学会各会員.