ICOT Technical Report: TR-458

TR-458

FGHCのメモリ使用特性と世代別GC

小沢　年弘、細井　聡、
服部　彰(富士通)

March, 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

ＦＧＨＣのメモリ使用特性と世代別ＧＣ

小沢　年弘　細井　聡　服部　彰
（株）富士通

概要

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、ほぼはっきり分割できることなどが明らかになった。これらの性質をメモリ管理に利用するために、二つだけに世代を限った二世代ＧＣ方式を提案するとともに、その性能評価を行い、第一世代のＧＣにより大部分のゴミを回収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。このＰＩＭ上に実装される核言語ＫＬ１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[10]。

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することにより、ＦＧＨＣ向きのメモリ管理方式を考案し評価している。

２　ＦＧＨＣ

ＦＧＨＣは、効率的な実現のためにＧＨＣに制限を加えた言語であり、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, Guardn (n = 1, 2,...), Bodyn (n = 1,2,...)は、それぞれ素命題を表わし、Head, Guardn, Bodyn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、ゴール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・ゴールが成り立つ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。一つ目は、ゴールの未束縛の変数がある具体値に決まればコミットできる場合であり、そのときには、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時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無い場合である。このときには、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を効率的に行うことができるようになり、効率的な実現が可能になった。

３　ＦＧＨＣ並列処理系

ＦＧＨＣの処理系は、すでに何件か実現されている。

我々は、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ＫＬ１－ｂ[4]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。
処理系の構成は、図１に示すように、エミュレータを複数発生させ、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。エミュレータ間の通信は最終的には共有メモリを通して行われる。

パッシブ・パートは、コンパイラによりＫＬ１－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わさる[2][3]。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。
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。ゴール・レコードは、フリー・リスト管理されており、再利用される。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・レコードを指させる（フックする）。将来、その変数が具体化するとき、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、実行を再開する。
ゴール・レコードは、フリーリスト管理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、ガーベジ・コレクション（ＧＣ）により回収される。

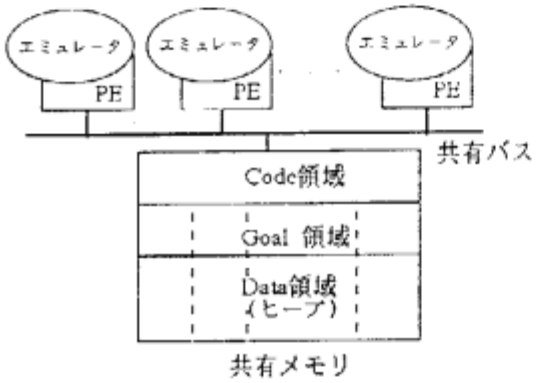


図　１処理系の構成

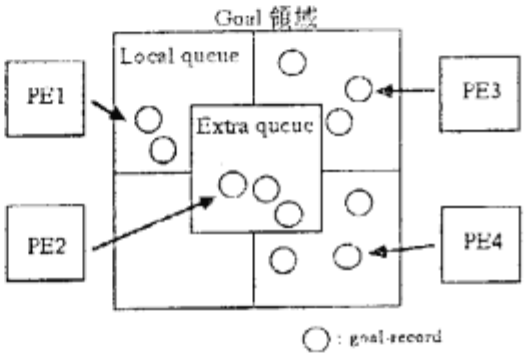


図　２　負荷分散方式

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、全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。[11]
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（図　２）。

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、次の通り。
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、ゴール実行毎に各自のローカル・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・リングであるが、負荷分散でエキストラ・キューに移動するゴールのために完全なｄｅｐｔｈ－ｆｉｒｓｔとはならない。
CONSTANTの実際の値は、４に設定してある。

５　ＦＧＨＣのメモリ使用特性

ガーベジ・コレクション（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、予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。エキストラ・キューのようなシステムで共有している資源は、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
２）データのライフ・タイム
Ａ～（Ａ＋ａ）番目に割り付けられたデータの生存率の時間変化。ただし、生存率　＝　アクティブ・データ量　／　割り付けたデータ量

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を上げる。
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。
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に上げる。

		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
サスペンド率	１ＰＥ	１％	１１％	４２％
	８ＰＥ	３８％	２６％	４２％
	１６ＰＥ	３５％	２６％	４１％

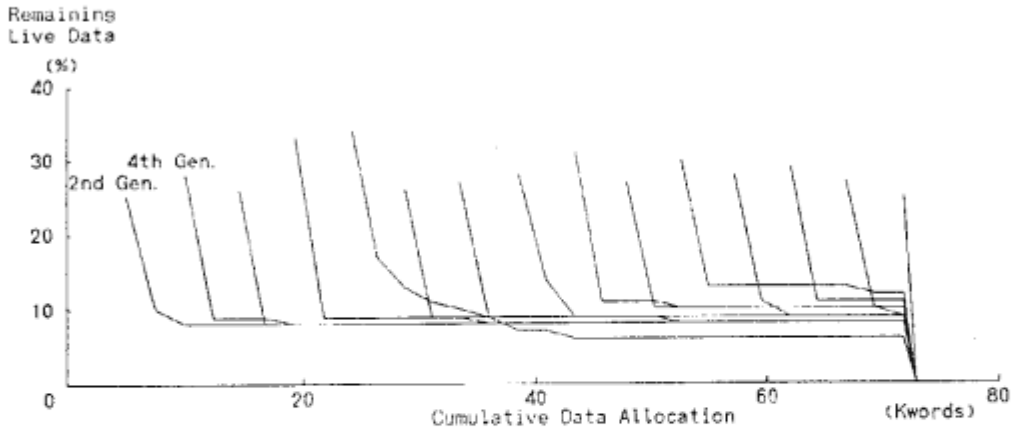
表　１　プログラムの性質

５．２　アクティブ・データ

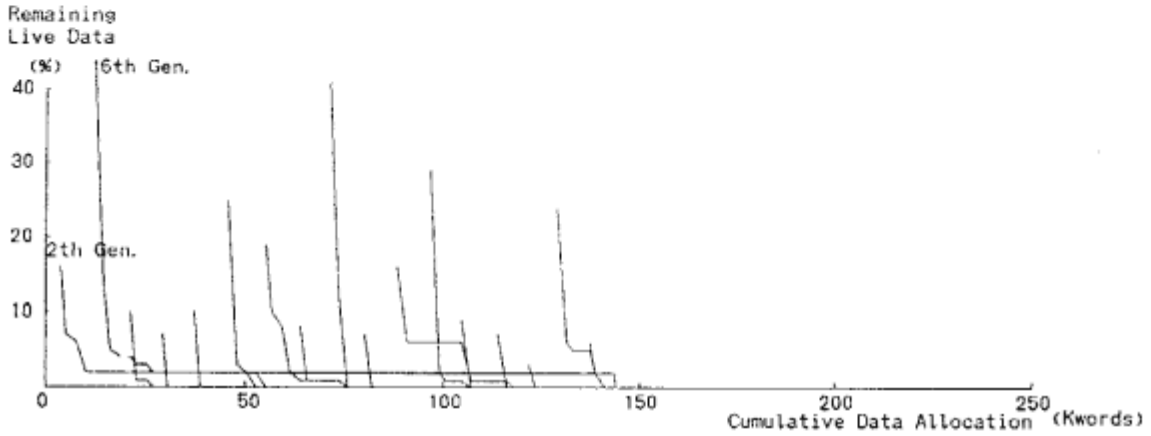
各プログラムの最大アクティブ・データ率を表２に示す。

		ＢＵＰ	ＤＢ	ＭＡＸＦ
最大アクティブ・データ率	１ＰＥ	２１％	１５％	２９％
	８ＰＥ	１９％	１５％	８％
	１６ＰＥ	１８％	１５％	７％

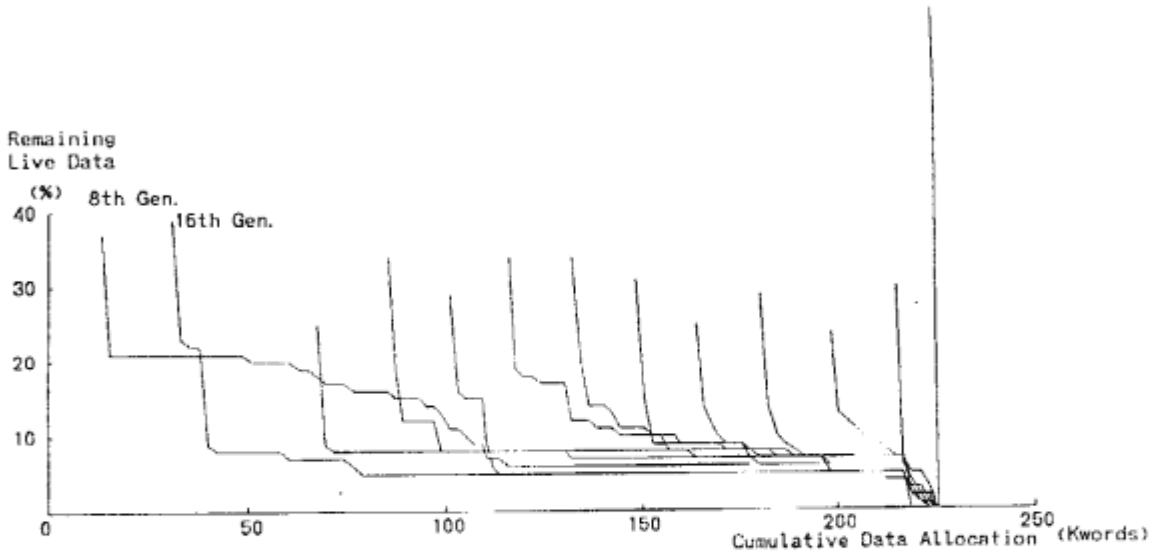
表　２　最大アクティブ・データ率



(a) BUP with 8PEs



(b) DB' with 8PEs



(c) MAXF with 8PEs

図　3　データの生存率の変化

一台での実行ではデータを生成するゴールがずっと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ィブ・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５４Ｋ、全割り付けデータ量１４４Ｋｗｏｒｄｓ）。横軸は、それまでに割り付けたデータ量を単位としている。各ラインは、各世代毎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。この傾向は、使用するＰＥ数に依らない。これは、ＦＧＨＣプログラムの次のような性質に依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、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、大きく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わりと長い。

６　ＦＧＨＣ向きの世代別ＧＣ方式－二世代ＧＣ－

データの生存率が低いことから、ＧＣの方式としては、アクティブ・データに比例した時間で済むコピー法に基づくＧＣが、メモリ管理には有利である。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、世代を二世代に限った世代別ＧＣ方式が適していると考えられる[6][7][8][9]。

第一世代は、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、第二世代から第一世代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（図４）。

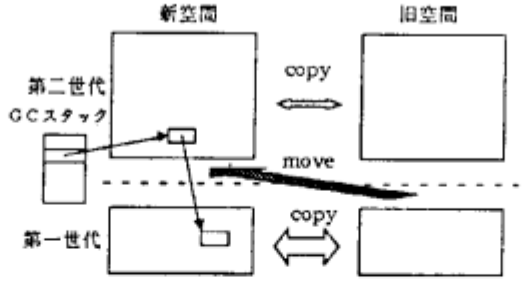


図　４　二世代ＧＣの構成

リダクション時やＧＣ時に第二世代から第一世代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。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。新しいデータは第一世代に割り付けられる。第一世代の中のデータは、割り付けられた後、ある時間以上経っても生きていれば、第二世代に移される。

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代用される。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。

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

ールドをつけ、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管理する。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より以前に割り当てられたデータは第二世代に移せばよい。もし第二世代に移すための最低割り付け量Ｔごとにを起動することが可能ならば、データの世代フィールドは必要なくなり、一回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、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済む。

第二世代のＧＣは、第一世代といしょに行われる。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。

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られる最低量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。縦軸は、第二世代にコピーされたデータ量と最大アクティブ・データ量の比、および第二世代にコピーされたデータ量と全割り付けデータ量との比である。

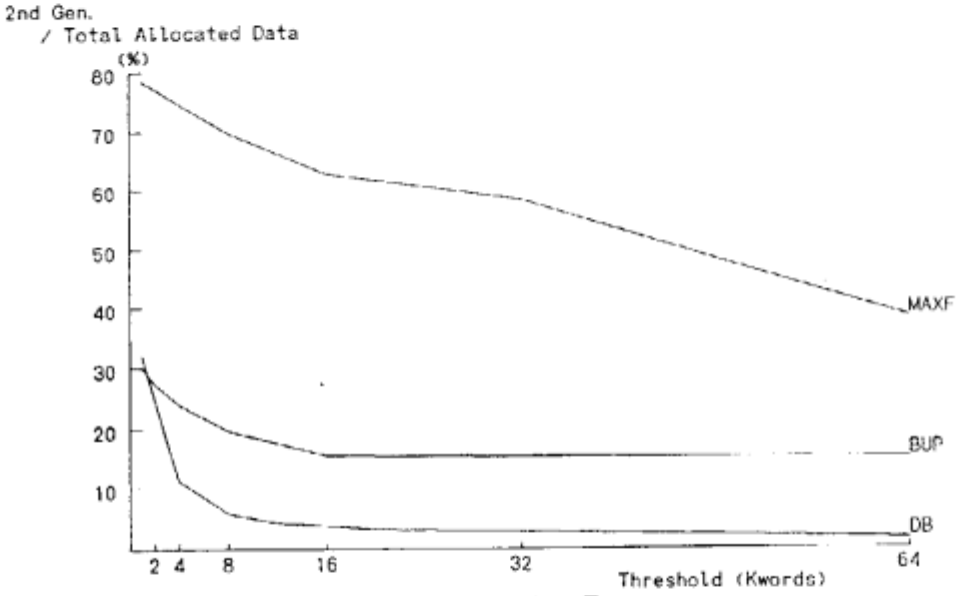


図　５　第二世代へのコピー量

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し、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てしまう。この二つのプログラムの場合には、Ｔの値として８～３２Ｋｗｏｒｄ程度が最適であり、第一世代のＧＣだけで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を頻繁にかける必要がある。ただし、Ｔ＝２０Ｋｗｏｒｄ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。

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比較的はっきり二つに分かれていること

が分かった。

さらに、これらの性質より、ＦＧＨＣ向きのＧＣ方式として、二つの世代だけをもつ二世代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、第一世代を持つことに意味がある。
２）第一世代から第二世代への移動を割り付け後２０Ｋｗｏｒｄ程度以降にすると、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。

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も残っている。

謝辞

日頃御指導いただく渕一博部門長、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] K. Ueda, 'Guarded Horn Clauses', Technical Report TR-103, ICOT, 1985.
[2] N. Ichiyoshi, T. Miyazaki, and K. Taki, 'A Distributed Implementation of Flat GHC on the Multi-PSI', Proceeding of the Fourth International Conference on Logic Programming, 1987.
[3] M. SATO and A. GOTO, 'Evaluation of the KL1 Parallel System on a Shared Memory Multiprocessor', IFIP WG 10.3 Working Conference on Parallel Processing in Pisa, Italy, 1988.
[4] Y. Kimura and T. chikayama, 'An Abstract KL1 Machine and Its Instruction Set', Proceedings of 1987 Symposium on Logic Programming, 1987.
[5] H. G. Backer, 'List Processing in Real Time on a Serial Computer', Comm. ACM, Vol. 21, No. 4, pp. 280-294, 1978.
[6] H. Lieberman and C. Hewitt, 'A Real-Time Garbage Collector Based on the Lifetimes of Objects', Comm. of the ACM, Vol. 26, No. 6, 1983.
[7] S. Ballard and S. Shirron, 'The Design and Implementation of VAX/Smalltalk-80', Smalltalk-80: Bits of History, Words of Advice, G. Krasner(editor), Addison Wesley, pp. 127-150, 1983.
[8] D. Ungar, 'Generation Scavenging: A Non-disruptive High Performance Strage Reclamation Algorithm', Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practival Software Development Environments, pp. 157-167, 1984.
[9] K. Nakajima, 'Piling GC - Efficient Garbage Collection for AI Languages-'
[10] 佐藤正俊、後藤厚宏、'ＫＬ１並列処理系の評価ーメモリ消費特性とＧＣー'，並列処理シンポジウム　ＪＳＰＰ'８９，1989．
[11] M. Sugie, M. Yoneyama, and , A. Goto, 'Analysis of Parallel Inference Machines to Achieve Dynamic Load Balancing', Proceeding of International Workshop Artificial Intelligence for Industrial Applications, 1988.