ICOT Technical Report: TR-496

TR-496

FGHC向き世代別ガーベジ・コレクション

小沢　年弘、細井　聡、服部　彰(富士通)

July, 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向き世代別ガーベジ・コレクション

Generation type Garbage Collection for FGHC

小沢　年弘　　細井　聡　　服部　彰
Toshihiro OZAWA　Akira HOSOI　Akira HATTORI

富士通（株）
FUJITSU LIMITED

１．　はじめに

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上に実装される言語は，ＡＮＤ並列論理型言語ＧＨＣ[1]に効率化のための制限を加えた言語ＦＧＨＣに基づいている．

ＦＧＨＣは，並列実行される複数のプロセスを生成し，変数を通してそれらの間の同期／通信を簡単に記述できる言語である．ＦＧＨＣでは，変数は単一代入であるためにメモリを多量に消費し，かつ動的なメモリ管理が必要とされる．従って，ＦＧＨＣでは，メモリ管理が言語の効率的実現のために重要な課題となっている[2]．

この課題を解決するため，参照カウンタによるガーベジ・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，カウンタ管理のオーバー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プログラムのメモリ使用特性に基づいて，ＦＧＨＣ向きのメモリ管理方式として二つの世代を持つ世代別ガーベジ・コレクション方式（二世代ＧＣ）を提案し，ＦＧＨＣ並列処理系上にその方式を試作し，評価した結果について述べる．

２．　並列言語ＦＧＨＣ

ＦＧＨＣのプログラムは，次のような文（節）の集まりである．

$$Head :- Guard_1, Guard_2, \ldots \mid Body_1, Body_2, \ldots .$$

$Head, Guard_n$ $(n = 1, 2, \ldots)$, $Body_n$ $(n = 1,2,\ldots)$は，素命題を表わし，それぞれヘッド，ガード・ゴール，ボディ・ゴールと呼ばれる。

ＦＧＨＣの実行は，あるゴールとユニフィケーションできるヘッドを持ち，かつすべてのガード・ゴールが成立する節を選らぶことから始まる．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．ゴールに含まれる変数の具体化が必要な場合は，そのゴールの実行は中断される（サスペンド）．

節が選ばれるとそのゴールの実行は終了し，選ばれた節のボディ・ゴールが並列に実行される．ボディ・ゴールの実行においては，変数の具体化が許される．実行を中断されたゴールは，他のゴールの実行により変数の値が決まった時に再び実行される．一つのゴールを実行することをリダクションと呼ぶ．

３．　ＦＧＨＣ並列処理系

ＦＧＨＣ処理系は何件か実現されているが[5) 6) 7) 8)]，我々はメモリ共有型並列計算機（Sequent Symmetry）上にＦＧＨＣ並列処理系を作成し[9]並列実行時におけるメモリ使用特性を測定した．

この処理系は，ＦＧＨＣプログラムをＫＬ１-b[10]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）に割り当て並列実行を行う（図１参照）．コンパイルされたコードは，全ＰＥに共有され，共有メモリの特定の領域（Code領域）におかれる．

ボディ・ゴールは，ゴール・レコードと呼ばれる制御レコードにより表わされる．あるゴール・レコードのライフ・タイムは，表わしているゴールの生成から実行終了までであるので，この処理系では，ゴール・レコード専用領域（Goal-record 領域）を用意し，フリー・リスト管理によりゴール・レコードを再利用している．

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．これらのデータのライフ・タイムを予め知ることは困難であるため，ヒープは，ＧＣにより回収，再利用される．

この処理系では，ゴールの実行は次のように行われる．以下の説明では，ゴールはゴール・レコードで表わされているものとする．

生成されたゴールはスケジューリング・キューにつなげられ，エミュレータは，スケジューリング・キューからゴールを取り出し実行する．ただし，スケジューリング・キューへのアクセス競合を防ぐために，図２に示すように複数のスケジューリング・キューを用意している[11]．各エミュレータは，他のエミュレータからはアクセスできない自身のスケジューリング・キュー（ローカル・キュー）を持つ．また，動的負荷分散を行うために，全エミュレータがアクセスできるスケジューリング・キュー（エキストラ・キュー）を処理系に一本用意している．普段，各エミュレータは，ローカル・キューをスケジューリング・キューとしているが，自身のローカル・キューが長くなりすぎたり，他にローカル・キューが空になったエミュレータが存在する場合には，エキストラ・キューにゴールをつなぐ．ローカル・キューが空になったエミュレータは，エキストラ・キューからゴールを獲得しようとする．ローカル・キューが空になったエミュレータの存在は，共有メモリ上のフラグを立てることにより知らされる．

４．ＦＧＨＣ処理系のメモリ使用特性

この処理系上で，新たにデータを約２Kword割り付けるごとにＧＣを起こし，ＦＧＨＣプログラムの特性を調べた．ここで用いたＧＣは，全プロセッサがゴールの処理を中断して，予め決められた順番で逐次に，コピー法ＧＣ[12]・[13]を行う方式である．

４．１　ベンチマーク・プログラム

測定に使用したプログラムを示す．

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[14]．データ・ベースそのものは，コード領域にあり，ＧＣ対象外である．

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．

表１に各プログラムの諸性質を示す．ただし，台数効果とは，ＰＥ一台での実行速度に対する比である．また，サスペンド率とは，（サスペンド回数）／（リダクション回数）である．

４．２　使用ゴール数とアクティブ・データ

表２に使用ゴール数と最大アクティブ・データを示す．使用ゴール数とは，フリー・リストから取り出され，使用中のゴール・レコードの数である．また，最大アクティブ・データ率は，（最大アクティブ・データ量）／（全割り付けデータ量）である．アクティブ・データとは，ヒープ中の使用中のデータ，全割り付けデータとは，実行終了までにヒープに割り付けられたデータである．

アクティブ・データ率は低いことが分かる．このために，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．

４．３　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

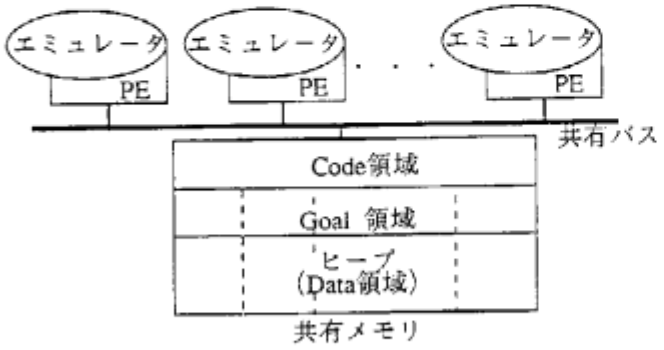

図１．処理系の構成

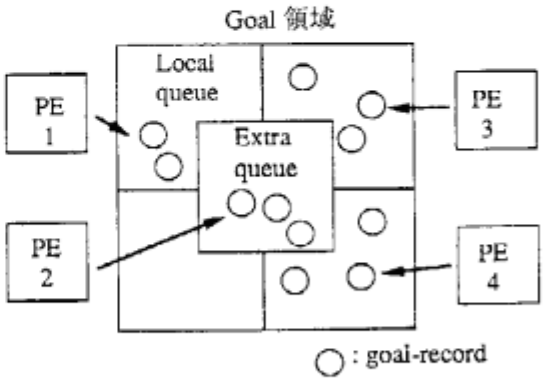

図２．負荷分散方式

ある．図3にプログラムＢＵＰ，ＭＡＸＦにおける実行にともなう生存率の変化を示す．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．ただし，生存率とは，（その世代のアクティブ・データ量）／（その世代の割り付けたデータ量）である．

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，ほぼ一定値を保っている．この性質は、すべてのプログラムが備えていた。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，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．

表１．ベンチマーク・プログラムの性質

		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.使用ゴール・レコード数と最大アクティブ・データ率

		ＢＵＰ	ＤＢ	ＭＡＸＦ
平均使用ゴール数	１ＰＥ	２０	９５７	３４６
	８ＰＥ	７３	２２６	３８７
	１６ＰＥ	１１１	３５３	３８７
最大使用ゴール数	１ＰＥ	２６	１２０３	４１２
	８ＰＥ	１３４	６５３	４３２
	１６ＰＥ	２０１	８１１	３８７
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％

５．　ＦＧＨＣ向きの世代別ＧＣ方式

ＦＧＨＣのデータ使用特性を考慮した世代別ＧＣ方式[15) 16) 17) 18)]を提案するとともに，その試作，評価について述べる．

５．１　二世代ＧＣ

データの生存率が低いことから，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種類に分けた世代別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）を各ＰＥに設ける．

世代の管理や世代別のＧＣは次のように行われる．

１）新しいデータは第一世代の新空間に割り付けられる．

２）ゴール実行時や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．

３）各世代のＧＣは，ＧＣ対象領域が異なるだけで同じアルゴリズムを用いている．

第一世代ＧＣは，各スケジューリング・キューとＧＣスタックをルートとし，第一世代のみフリップし（新空間を旧空間を入れ替える．），第一世代のみの

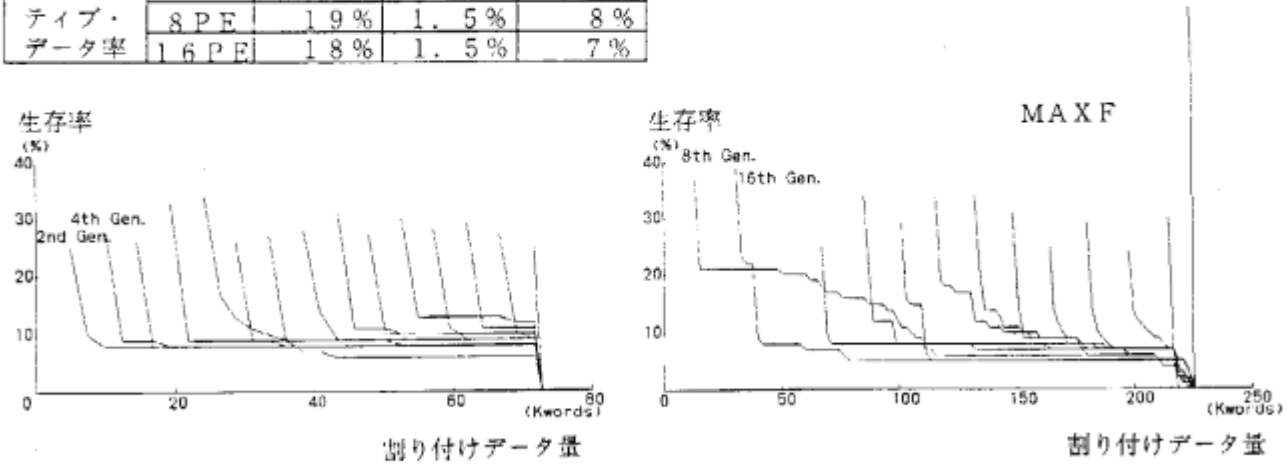

図３　データのライフ・タイム（８ＰＥｓ）

ＧＣを行う．第二世代ＧＣは，ルートを各スケジューリング・キューとし，両世代ともフリップし，両世代のＧＣを行う．

4）各世代のＧＣにおいて，第一世代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，第二世代の新空間に移される．その他のアクティブ・データは，第一世代の新空間に移される．

5）あるＰＥで第一世代の新空間が使い尽くされると，ＧＣ起動要求を共有メモリのフラグを通して他のＰＥに通知する．全てのＰＥは，各ゴール実行ごとにフラグを調べ，ＧＣ起動要求があればゴール実行を中断する．全ＰＥのゴール実行が中断されると，第一世代ＧＣが実行される．

6）　第二世代ＧＣは，第二世代が使い尽くされる時に起動されるのはもちろんであるが，より積極的に起動される場合もある．これは，第二世代に移動された後ゴミになったデータから指されているために第二世代に移動させられる（ゴミ）データを減らし，総移動量を減少させるためである．

7）第一世代ＧＣを第二世代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．

第一世代に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．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，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ごとに，その後割り付けたデータの量が分かるので，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．

また，第一世代をThresholdの大きさを持つ３つの領域に分け，各領域を旧空間（今までのデータ割り付け領域），新空間（ＧＣ後のデータ割り付け領域），第一世代の若いデータの移動先空間（一回ＧＣを経験したデータの領域）という構成にしても，世代別ＧＣが実現可能である．この場合には，新空間を使い切った時，つまりThreshold量のデータを割り付けるごとにＧＣを起動し，フリップする．このフリップでは，今までの新空間，旧空間，移動先空間は，それぞれ旧空間，移動先空間，新空間に変わる．（フリップ後の）旧空間のデータを移動先領域に，（フリップ前の）移動先領域のデータを第二世代の新空間に移動することにより，Threshold以上の割り付け後も生きているデータのみを第二世代に移動させることができる．この構成では，データに世代フィールドを付けることや，ＧＣごとのデータ割り付け量を管理する必要がなく，簡便な実現が可能である．

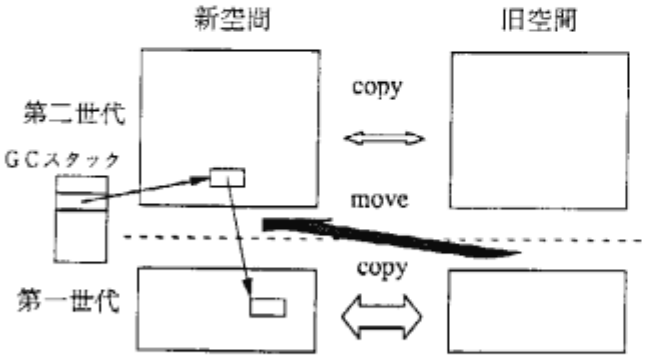

図４．　二世代ＧＣの構成

５．２　ＧＣの並列実行について

ＧＣの並列化は，次のように行った．各ＰＥは，初め自身のローカル・キュー中のゴールをルートとし，それからたどれるデータを移動する．次に，エキストラ・キューからまだたどられていないゴールを獲得し，そのゴールからたどられるデータを移動する．つまり，ＧＣにおける負荷平均化のために動的に分配されるのは，エキストラ・キュー中のゴールに限られ，分散される最小の仕事は，一つのゴールからたどられるすべてのデータを移動することである．

６　二世代ＧＣの評価

６．１　総データ移動量

二世代ＧＣの性能評価のため，先のベンチマーク・プログラムを実行したときのデータの総移動量を，普通のコピー法ＧＣと二世代ＧＣとで比較した．データの総移動量で性能を比較するのは，二世代ＧＣの一回のＧＣ実行時間が100ミリ秒以下と短く，実時間で測ると誤差が大きくなるためである．ＧＣの仕事量は，基本的にデータの総移動量に比例するので，この量で性能を比較することができる．両方式でヒープの大きさが同じという条件の下で，いくつかの大きさのヒープに対して総移動量を測定した．その結果を図５に示す．横軸は，（最大アクティブ・データ量）／（ヒープの大きさ）で示した．また、そのときのＧＣ回数を図６に示す．二世代ＧＣの構成は，前述した第一世代を３分割したものであり，第二世代の大きさは最大アクティブ・データ量に合わせ，第一世代の大きさを変化させた．なお，ＰＥ台数は，８台である．

すべてのプログラムにおいて，ヒープ中にアクティブ・データが多く，ＧＣが頻発する場合には，二世代

ＧＣの効果が大きい．逆に，BUP，MAXFでは，ヒープ領域を大きくすると，二世代ＧＣの移動量がコピー法より多くなる．しかし，これは，これらのプログラムのデータの総割り付け量があまり多くないために，コピー法ＧＣが数回しか起きていないためである．この場合でも，移動量の差は小さく，二世代ＧＣは，アクティブ・データの比率によらず，平均してよい効率が得られている．

また，世代管理のためのオーバーヘッドによるゴール実行速度の低下は，約５％に抑えられている．

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．

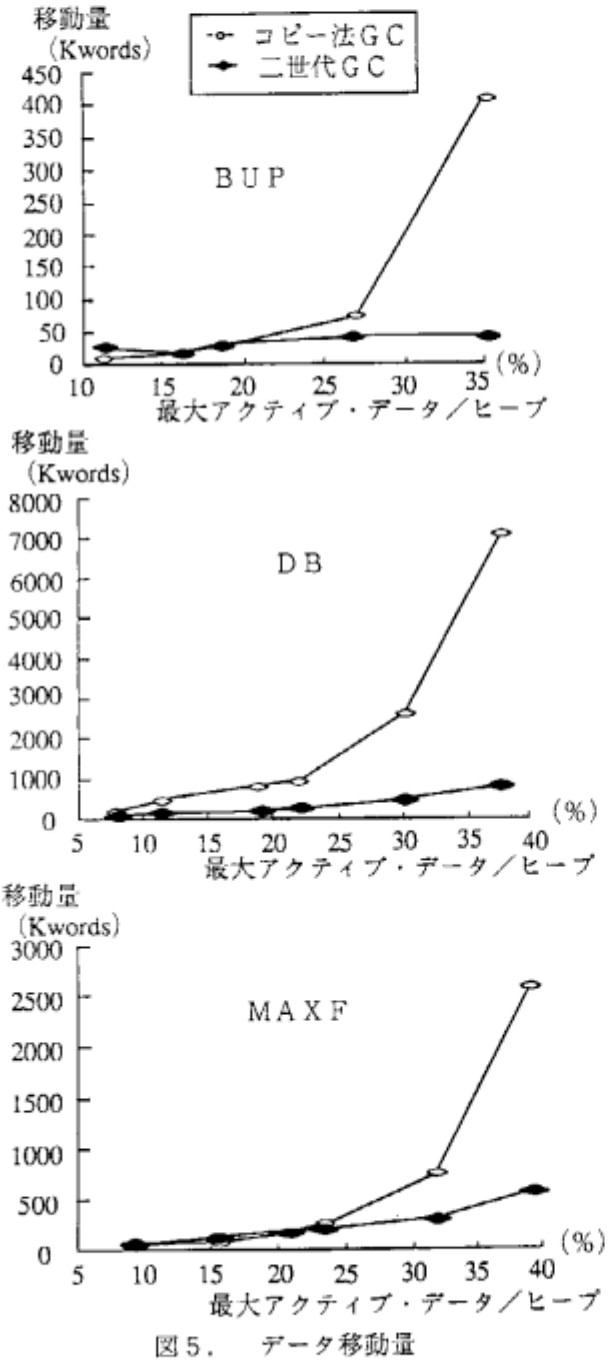

図５．　データ移動量

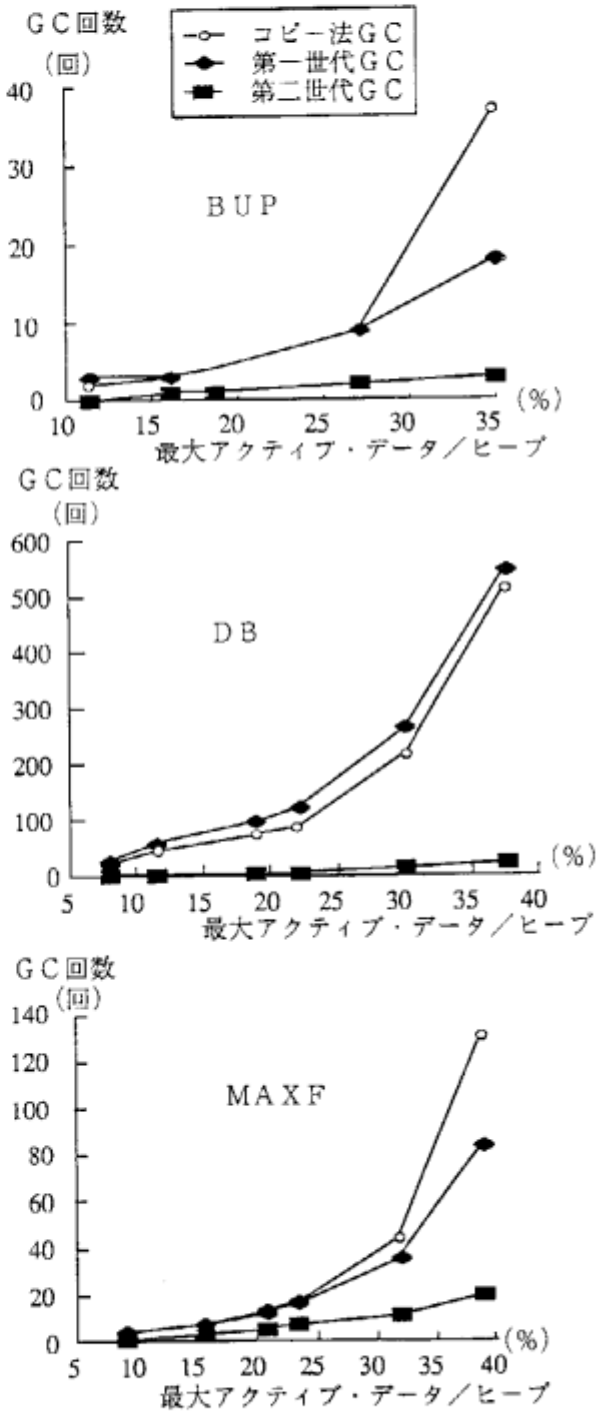

図６．　ＧＣ回数

6．2　並列化の効果

各ＧＣにおいて次の量を測定し，全ＧＣにわたって平均したものを図７に示す．

（最も多くデータを移動したＰＥのデータ移動量）／（そのＧＣで移動した全データの量）

この量は，各ＧＣにおいて，最も仕事をしたＰＥが，全体の何割の仕事をしたかを示すものであり，ＧＣ時間も，この割合に比例して短縮すると考えられる．

この結果より，並列化によるＧＣの実行速度向上は，ＰＥ３台以上において２～３倍程度に留まっていると考えられる．これは，ＧＣの分割単位が，あるゴールからたどれるすべてのデータを移動させることと比較的大きいこと，および移動量がゴールによりかたよることによる．並列化による実行速度向上をさらに上げるには，ＧＣの分割単位をより小さくすることが必要である．例えば，一度に行うデータの移動はポインタを一段たどるだけに限り，さらにデータをたどることは，負荷分散の対象にするなどして，仕事の大きさをより小さくすることが有効であると考えられる．

7．　まとめ

ＦＧＨＣプログラムのデータは，ライフ・タイムによりはっきりと二種類に分けられることを利用して，ＦＧＨＣ向きＧＣを提案し、その試作、評価を行った．その結果，

１）ヒープ領域に対するアクティブ・データの比率に依らず，効率的なＧＣ方式であることを確かめた．特に，ヒープに対するアクティブ・データの比率が高い場合には，普通のコピー法に比べ，１０倍以上の効率を示す．

２）ＧＣの並列実行に置いて，ＧＣをゴールからたどれる全データの移動ごとに分割した場合，ＰＥ台数３台以上で２～３倍の実行速度の向上が得られる．

しかし，より多くのＰＥで処理する場合や，並列化による実行効率の向上を目指すならば，ＧＣをより小さな仕事に分割し，より細かな負荷分散を行う必要がある．

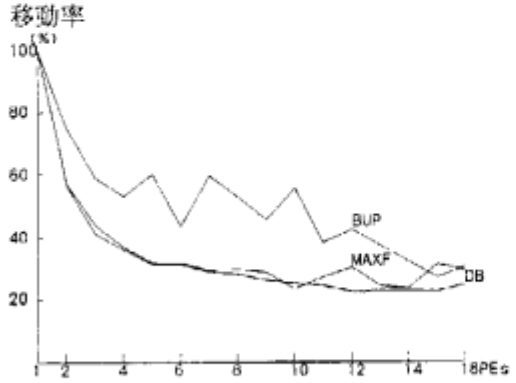

図７　データを最も移動したＰＥのデータ移動率

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．

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) K. Ueda, 'Guarded Horn Clauses', Technical Report TR-103, ICOT, 1985.
2) 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－〟，並列処理シンポジウムＪＳＰＰ〝８９，1989．
3) T. Chikayama, Y. Kimura, 'Multiple Reference Management in Flat GHC', Proc. Int. Conf. on Logic Programming '87, pp. 276-293, May, 1987.
4) A. Goto, Y. Kimura, T. Nakagawa, and T. Chikayama, 'Lazy Reference Counting:An Incremental Garbage Collection Method for Parallel Inference Machines', Proc. Int. Conf. on Logic Programmg '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) 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.
7) M. SATO and A. GOTO, 'Evaluation of the KL1 Parallel System on a Shared Memory Multiprocesor', IFIP WG 10.3 Working Conference on ParallelProcessing in Pisa, Italy,1988.
8) M.Sugie, M.Yoneyama, and, A. Goto, 'Analysisof Parallel Inference Machines to Achieve DynmicLoad Balancing', Proceeding of International Workshop Artificial Intelligenc for Industrial Applications, 1988.
9) 細井，小沢，服部，〝密結合マルチプロセッサ上のＫＬ１並列処理系の評価〟，第３８回情処全大，pp. 1009-1010, 1989.
10) Y. Kimura and T. Chikayama, 'An Abstract KL1 Machine and Its Instruction Set', Proceedings of 1987 Symposium on Logic Programming, 1987.
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〟，CPSY88-46, 1988.
12) Cohen, J., 'Garbage Collection of Linked Data Structures', ACM Comput. Surv., Vol 13, No. 3, pp. 341-367, 1981.
13) Baker, H. G., 'List Processing in Real Time on a Serial Computer', Comm. ACM, Vol. 21, No. 4, pp. 280-294, 1984.
14) 北上，横田，服部，〝知識処理向き並列推論エンジン〟，CPSY88-50, 1988.
15) 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.
16) 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.
17) D. Ungar, 'Generation Scavenging: A Non-disruptive High Performance Strage Reclamation Algorithm', Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, pp. 157-167, 1984.
18) K. Nakajima, 'Piling GC - Efficient Garbage Collection for AI Languages-', Proceeding of the IFIP WG 10.3 Working Conference on Parallel Processing, pp. 201-204, 1988.