圧縮型ガーベッジコレクションの高速化

寺島元章† 佐藤圭史†

この論文では新方式の圧縮型ガーベッジコレクションの設計とその評価を述べる．それは既成の圧縮型ガーベッジコレクションの処理時間を大幅に改善し，リスト処理系全体の高速化を実現することになる．これは，ガーベッジコレクション自体の高速化と対象領域の効果的使用から生み出された成果であり，テストプログラムを用いた実験からも実証されている．

Speedup of Sliding Compaction Garbage Collection

Motoaki Terashima† and Keishi Sato†

The design and analysis of new sliding compaction garbage collection are presented. The new GC largely saves the processing time of conventional sliding conpaction garbage collection, and it may make the Lisp system totally run faster. This results from both speedup of its processing and effective use of a storage space that it manages. They are shown by our experimental data obtained from the execution of Lisp test programs.

1. はじめに

ガーベッジコレクション（以下，略してGCという）は動的データを扱う言語処理系の必須の機能として，これまでに数多くの実装方式の研究が行われてきた．その成果はGC処理の高速化や並列化などに現れている[1]．

GCの一種である圧縮型GC（Silding compaction GC）は，データオブジェクトの作られた順序を保持しながら使用中のものを集めて，1つの塊にする機能を持つ．使用中と使用済データオブジェクトが混在する記憶空間は，使用中データオブジェクトのみの記憶空間に凝縮されることになる．そのため，スワップやキャッシングの効果が最大限に引き出されることになる．

本論で提案する新方式のGCはこうした圧縮型GCを高速化したうえで，そのGC対象領域を限定した結果，従来の圧縮型GCよりも処理速度が格段に向上している．GCを行うことで，ワーキングセットが小さくなり，GC以外の処理（純計算）は高速になる．さらに，GCを含めた処理系全体を高速にすることも可能である．

本GCは異種のワークステーションで稼働中のリスト処理系であるPHL[2),3)]（Portable Hash Lisp）に実装されている．PHLの各データオブジェクトは対象計算機のマシンワードを最小の単位として構成されている．これをフィールドと呼ぶ．64ビットアーキテクチャを持つAlphaマシン上のPHLは1ワード64ビット化にともなう機能改良が行われているが，いずれの処理系でも本GCによる処理全体の効率化が実験結果から示されている．

本論文では，GCの設計方針，実装法，および評価について述べる．

2. 背景

2.1 圧縮型GC

2.1.1 時間計算量

停止回収型GCは複写型（Copying GC）と圧縮型の2つの方式に大別される．典型的な複写型GCの処理は使用中データオブジェクトを移すことであるから，その時間計算量は，

$$O(A) \quad (1)$$

である．ここで，Aは使用中データオブジェクトの総容量である．

これに対して，典型的な圧縮型GCでは，印付け処理で使用中データオブジェクトを識別してから，圧縮とポインタの補正を行うため格納領域全体を操作する必要がある．このため時間計算量が格納領域の総容量に比例するというのが従来の定説であった．

† 電気通信大学大学院情報システム学研究科
Graduate School of Informations Systems, University of Electro-Communications

しかし，これまでの圧縮型 GC の高速化に関する研究の成果として，印付け後に使用中データオブジェクトのクラスタ（使用中データオブジェクトの連続したフィールドの塊）の先頭アドレスをデータとして大小順にソートする方式により，時間計算量を，

$$O(A) + O(n \log n) \qquad (2)$$

にまで短縮することが可能となった[4),5)]．ここで，n はクラスタの総数である．なお，一般的なアプリケーションでは使用中のデータオブジェクトの集まりはクラスタを形成しやすいので，n は A と比べて十分小さいことが予想される．この場合，式 (2) は，

$$O(A) \qquad (3)$$

と近似され，圧縮型 GC は複写型の GC と同じ時間計算量になる．

2.1.2 ポインタ補正

圧縮型 GC では使用中データオブジェクトの移動にともなって，ポインタの補正が必要となる．ポインタの補正を高速に行うアルゴリズムに補正表[6)] を用いる方式がある．

補正表を用いるポインタ補正では，GC 対象領域を 2^m $(m > 0)$ フィールドごとの小区画に分割し，1 回目の走査でその先頭番地の補正値（代表値）が補正表に格納される．2 回目の走査でポインタの補正が行われるが，対象となるポインタの補正値は，その参照先の小区画の代表値（参照先が小区画の後半であれば，次の小区画の代表値）と，小区画内の補正値の和として求められる．小区画内の補正値は対象となるフィールドに近い端より走査を行うことで求まる．これは，平均 $2^m/4$ 個のフィールドを走査することになり，論文 7) によれば，この値が 4 以下であるならば，Morris[8)] の方法よりも高速である．

こうしてポインタ補正が行われた後に，3 回目の走査で GC 対象領域の圧縮が行われる．

2.2 世代別管理

GC 処理を効率化するための戦略に世代別管理[9)] と呼ばれる手法がある．世代別管理は，長期間使用されている古いデータオブジェクトと比較的新たに作成されたデータオブジェクトを区別し，前者に対する回収作業を省くことで GC 処理にかかる時間を短縮するものである．圧縮方式 GC の枠組みの中に世代別管理を取り入れたアルゴリズムとして，MOA[10)] などが提案されている．

世代別管理 GC を実装する際の問題点もある．最大の問題は，Lisp の SETF などによるリスト構造の変更によって作られる古い世代のデータオブジェクトから，新しい世代のデータオブジェクトを指すポインタである．これを「前向きポインタ」と呼ぶことにする．前向きポインタの存在は，単純に古い世代のデータオブジェクトを処理の対象から外すことができないことを意味する．これは世代間のポインタの整合性が失われるためである．この解決方法として，世代間に跨るこのようなポインタを識別して，これを表（remembered set）に登録，根として管理する方法が知られている[9)]．MOA は remembered set を用いずに，単純な世代別管理を行っている．

3. ガーベッジコレクション

前述のように，圧縮型 GC はほぼ $O(A)$ の時間量でその処理を終えることができる．さらに高速な処理は，GC 対象領域を限定することによって可能となる．

本 GC は最も新しく作成されたデータオブジェクトが存在する領域に対象を絞って処理を行う．この GC 対象領域の限定化は，世代別管理方式による限定化よりもさらに狭い範囲となる．このため，GC の対象にならない使用済データオブジェクトが存在することもありうる．

データオブジェクトは格納領域のアドレスの小さい方から順に詰められ，格納領域は十分大きいものとする．GC は格納領域が一定の量（D）だけ使用された際に起動される．GC 終了後，新たなデータオブジェクトは，GC によって圧縮された領域以降に納められていく．GC が順次呼び出されていく様子を図 **1** に示す．*top* は次回の GC が起動されるまでに使用可能な領域の上限を指すと同時に，GC 対象領域の上端を指

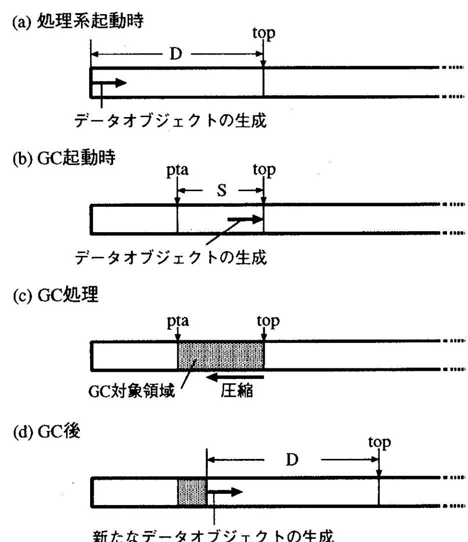


図 **1** GC 処理
Fig. 1 GC processing.

している．また，*pta* は GC 対象領域の下端を示すものであり，*top* より S $(0 < S \leq D)$ だけ戻ったところを指す．

GC は，*top* に近い最新のデータオブジェクトが存在する領域を処理の対象とする．この領域は，計算で使用中のデータオブジェクトと，計算がきわめて短期間に計算が終了して，すでに使用済となったデータオブジェクトが混在している可能性が高いためである．これらよりも古く生成されたデータオブジェクトは，その大部分がすでに使用済となっている可能性が高いと考えられる．最も古いデータオブジェクトが存在する領域はアンカーと呼ばれ，格納領域の最下位部（アドレスが最も小さい）になる．ここには使用中のデータオブジェクトが多く存在（凝縮）していることがある．

このように，本 GC は最も新しく作成されたデータオブジェクトの存在する領域を GC 対象領域とすることで，処理の高速化に直結する使用中データオブジェクトの局所化の効果を得ることを意図している．なお，本 GC で圧縮方式を利用しているのはこのように限られた一部の領域中のデータオブジェクトを，生成順序に依存して選別していることによるものである．

さらに，本 GC はポインタ補正の高速化と補正表のエントリに関する見直しによって GC アルゴリズム自体の高速化も図っている．

3.1 GC 対象領域の管理

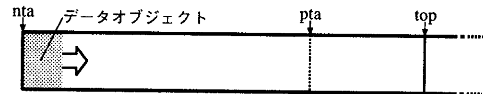
本 GC では，GC 対象領域の限定化によって，世代別管理と同様に前向きポインタの処理に関する問題が生じる．また，回収を効率的に行うために，GC 対象領域に隣接する使用済データオブジェクト群の先頭フィールドを，圧縮される領域の開始点とする．

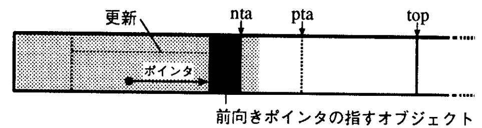
本 GC ではこれらを解決するために，GC 対象領域外で前向きポインタが完結する領域を把握，管理する．また，GC 対象領域の境界を跨る前向きポインタは remembered set に登録し，GC の印付けでの根としている．

GC 対象領域外で前向きポインタが完結する領域は，*nta* が管理している．*nta* は SETF，REPLACE 関数の実行で前向きポインタが発生した場合に更新作業が行われる．この様子を示したのが**図 2** である．その終点が GC 対象領域外を指す前向きポインタである場合は，*nta* の値を更新し（図 2 (b)），終点が GC 対象領域内である前向きポインタが発生した場合は，このポインタを remembered set へ登録する（図 2 (c)）．

3.2 GC 対象領域の確定

GC が起動すると，GC 対象領域の印付け作業を行うが，同時に後向きポインタ（その終点が始点より小さいアドレスであるポインタ）を調べ，その終点のアドレスで最大のものを求める．





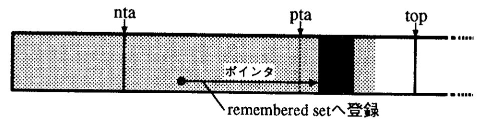


図 2 *nta* の管理
Fig. 2 Administration of *nta*.

最終的な GC 対象領域については，この終点の最大アドレスと *nta* を比べ，大きい値（アドレス）が指すオブジェクトの次が GC 対象領域の下端となる．この下端が *pta* を超えなければ，その間に存在するデータオブジェクトは使用中データオブジェクトから参照されていないので，これらは使用済データオブジェクトとして扱える．*pta* を超える場合は，そのオブジェクトはそのまま残される．

この GC 対象領域決定アルゴリズムは，データオブジェクトの寿命が非常に短いプログラムで有効となることが 4 章で示される．

3.3 圧縮方式アルゴリズムの高速化

本 GC では，ほぼ $O(A)$ の時間計算量で処理を終える GC を実装しているが，いっそうの高速化のためにその圧縮アルゴリズム，補正表のエントリ，ポインタ補正のアルゴリズムに関して，改良を行っている．また，補正表や remembered set などの作業領域は，GC 終了後に新たな格納領域となる部分に確保される．

3.3.1 2 回の走査

本 GC の処理手順は，印付け，補正表の作成，データオブジェクトの再配置の順に行われる．ポインタ補正は，補正表の作成時とデータオブジェクトの再配置のときに行う．補正表はアドレスの小さい方から各クラスタを走査することで作られる．このとき，クラスタ中の後向きポインタ（より小さいアドレスを指すポインタ）が補正される．この補正は，これらのポインタが指すデータオブジェクトの補正値が，（作成中の）補正表から求められることを利用している．データオ

表 1 Morris の方法と補正表を用いる方法の GC 処理時間の比較
Table 1 Comparison of GC processing time.

TPU

格納領域（MB）		0.10	0.15	0.20	0.25	0.30	0.40	0.80
処理時間（sec.）	Morris 法	0.48	0.26	0.19	0.17	0.13	0.08	0.04
	補正表（$m = 5$）	0.25	0.15	0.10	0.08	0.06	0.05	0.02

REDUCE

格納領域（MB）		0.10	0.15	0.20	0.25	0.30	0.40	0.80
処理時間（sec.）	Morris 法	4.33	3.44	1.94	1.60	1.00	0.72	0.33
	補正表（$m = 5$）	2.57	1.90	1.26	0.96	0.77	0.45	0.26

ブジェクトの再配置では，作成済の補正表を用いて前向きポインタの補正が行われる．このように本 GC での格納領域の走査回数は 2 回である．

3.3.2 ポインタ補正

後向きポインタの補正では，現在走査中のクラスタ内を指すポインタの補正値は，補正表を用いずに即座に求めることができる．これは，その値が pta から現クラスタまでの無印フィールド（使用済データオブジェクトのフィールド）の累計値そのものであり，既知であることによる．こうした場合以外や，前向きポインタの補正では補正表が利用される．補正表は GC 対象領域を 2^5 フィールドごとの小区画に分割し，これら小区画の代表値をエントリとしている．

補正表では，データオブジェクトの補正値を表現するのに必ずしも 1 語（マシンワード）を必要としないことに着目し，補正表のエントリ内の余分なビットを別の目的で使用している．PHL では，データオブジェクトのアドレスが 29 ビットのバイトアドレスで表現されていることから，補正値は 27 ビットで十分であり，残りの 5 ビットは他の目的で使用可能である．これらの余剰ビットを利用して各小区画の前半に含まれる無印フィールドの数が記録される．補正値を求める際は，この情報を用いることにより，小区画を 4 分割し，その分割された領域の走査で補正値が求められる．この場合，平均 $2^5/8 = 4$ フィールドの走査となる．また，小区画内にクラスタが 1 つしか存在しないような場合で，小区画内のオブジェクトの補正値がすべて同一であるとき，小区画の走査は不要である．補正表のエントリの MSB はこのことを示す目的で利用している．

こうした改良により，本 GC は Morris[8] に基づいた方法よりも高速であることが**表 1** の実行結果からも示されている．表 1 は，Morris の方式と補正表（$m = 5$）に基づいた 2 つの圧縮型 GC について，それらの処理時間を格納領域を変えて求めたものである．処理系は NEWS-5000（R4400SC/50 MHz）上の PHL である．なお，補正表も格納領域に含まれている．

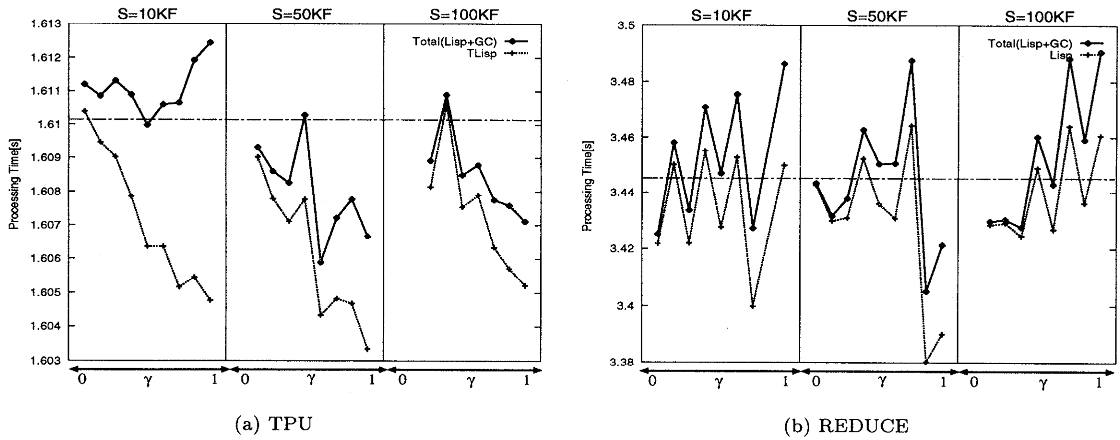
4. 評 価

4.1 処理速度

本 GC の性能評価の実験結果を図 **3**，**4**，**5** に示す．この実験で使用したテストプログラムは TPU[11]，REDUCE[12] である．TPU は定理証明を行う Lisp プログラムであり，REDUCE は数式処理を行う Lisp プログラムである．TPU は比較的データオブジェクトの寿命が短く，REDUCE は setf などの副作用のある関数が多用されており，しかもデータオブジェクトの寿命が比較的長い．両者は世代の観点からは相反するプログラムである．測定は，Visual Technology VT-Alpha 500AXP，Sun Enterprise 450，および NEC PC-9821V13 の 3 種の計算機上で行った．なお，計算機上で稼働しているオペレーティングシステムはそれぞれ，DEC Digital UNIX，日本語 Solaris2.6，FreeBSD2.2.8 である．

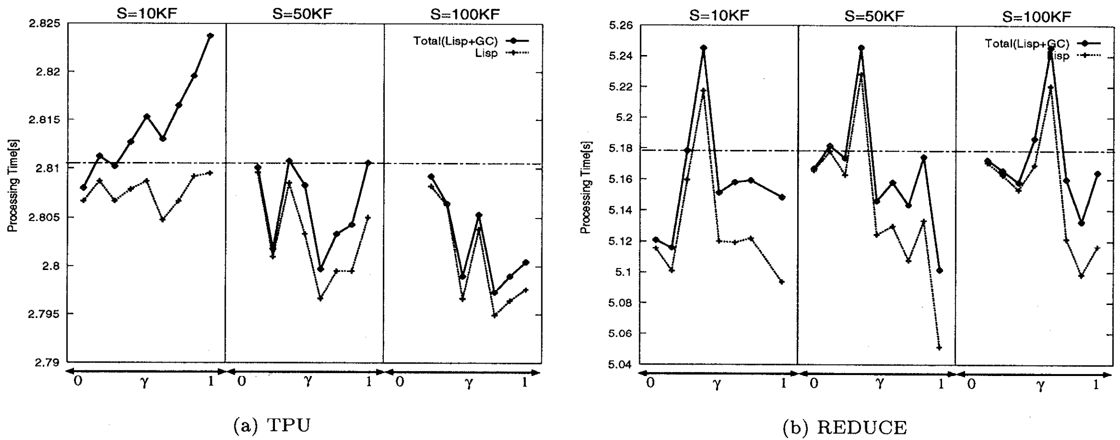
図 3，4，5 に示す結果は，GC 対象領域の大きさ S を固定し，それぞれについて D $(S < D)$ の値を変化させたときの処理時間である．実線が処理系全体を，点線がリスト処理に要した時間を示す．これらの差は GC 処理時間である．GC が呼び出されない場合の処理時間は，水平の破線が示す．TPU の結果では，$\gamma\ (= S/D)$ が大きくなるにつれてリスト処理の時間は減少するが，逆に GC 処理の時間が増大することが視覚的に分かる．これは，ワーキングセットの縮小がリスト処理系のコストに，GC 対象領域の増加が GC 処理のコストに反映したものである．REDUCE に示される結果では，アンカーに対する回収作業が影響し，リスト処理に必要な時間も増大することが示される．GC によるリスト処理時間の減少と GC 処理自体にかかる時間のバランスがとれ，アンカーに対する回収作業が行われないような場合に，本 GC の効果が得られることがこの結果から示される．

TPU，REDUCE ともに，本 GC によるリスト処理系高速化が実現されている．また，計算機のアーキテクチャや，オペレーティングシステムに大きく依存



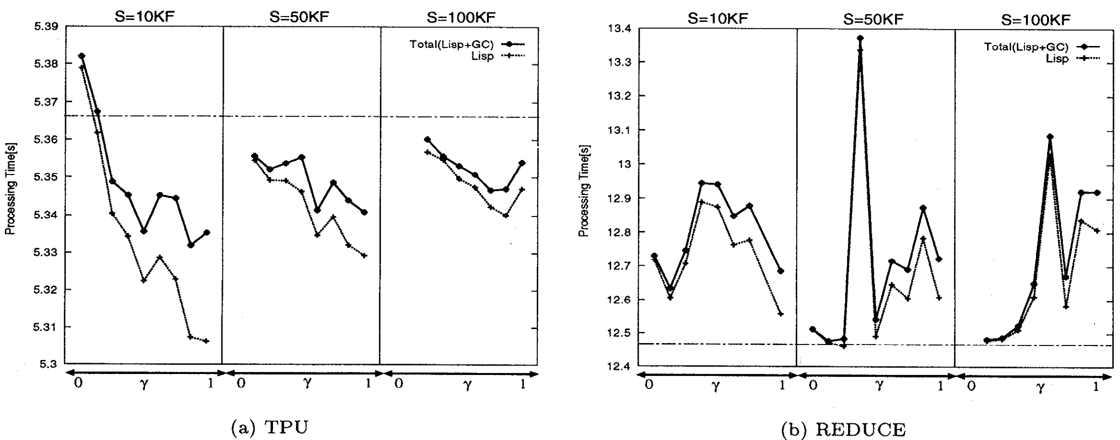
(a) TPU　　(b) REDUCE

図 3 処理時間（Digital UNIX/Alpha）
Fig. 3 Processing time of PHL (Digital UNIX/Alpha).



(a) TPU　　(b) REDUCE

図 4 処理時間（日本語 Solaris2.6/Ultra Sparc）
Fig. 4 Processing time of PHL (Solaris2.6/Ultra Sparc).



(a) TPU　　(b) REDUCE

図 5 処理時間（FreeBSD/Pentium）
Fig. 5 Processing time of PHL (FreeBSD/Pentium).

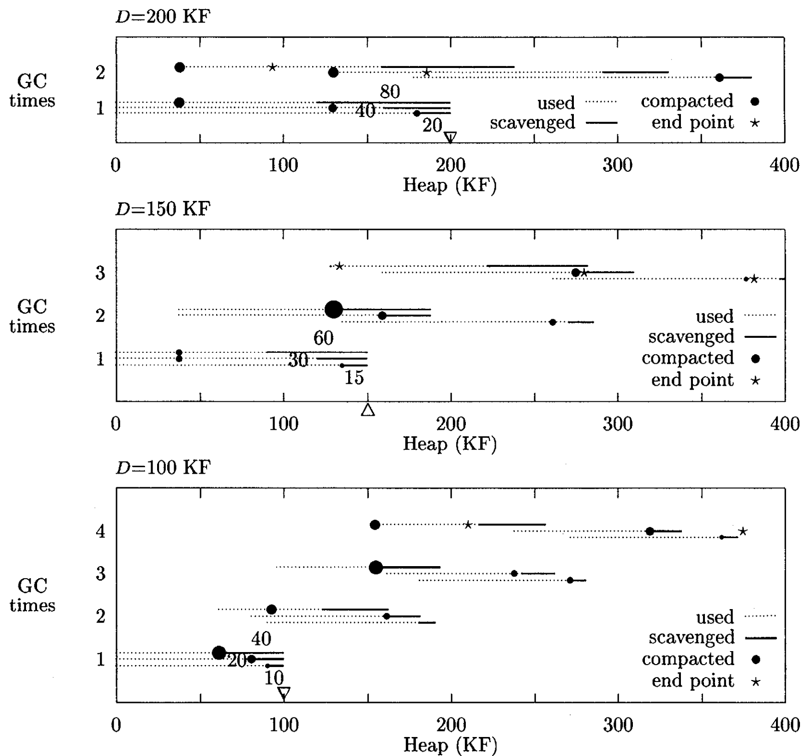


図 6　格納領域の使用状況（TPU プログラム）
Fig. 6　Utilization of a heap (TPU program).

しないことも示される．

4.2　格納領域の様子

TPU プログラムを実行したときの本 GC による格納領域の様子を図 **6** に示す．図中の点線は GC が起動される前までに使用された格納領域である．その中で，実線は GC 対象領域を示す．黒丸の位置は GC により圧縮されたオブジェクトの下端を，その大きさは，それらデータオブジェクトの総量を相対的に示したものである．

一部に，GC 対象領域より下位アドレスに圧縮が行われている場合がある．これらは特に GC 対象領域が大きい場合に顕著な結果である．これらは本 GC が GC 対象領域外の隣接する不使用データオブジェクトを副次的に回収することによるものである．実験を行った TPU のように，データオブジェクトの寿命が短く，使用済データオブジェクトのみが存在する領域が GC 対象領域に隣接する場合に，こうした結果が得られる．このため，必要以上に GC 対象領域を確保しても使用される格納領域の総量はそれほど変化しない．これに反して，GC 対象領域外に最新の使用中データオブジェクトが存在する場合，その部分の回収は行われず，必要な格納領域量は大きなものとなってしまうことがこの図から示される．

4.3　全 面 回 収

実行が進むと GC の対象とならなかった領域が未回収として残されることになる．格納領域のより効果的な使用のために，この領域全体の回収を定期的に行うことはもちろん可能である．本 GC の特徴として，処理時間は使用中データオブジェクトの総容量（A）に比例するため，この全面回収に要する時間は格納領域の大きさには依存しないが，それでも GC の多少のオーバヘッドが生じることになる．

5.　お わ り に

本論文では GC 処理を行うことにより処理全体が高速かつ効率的に働くような新方式の圧縮型 GC を提案した．ワーキングセットを小さくすることでリスト処理系を高速にすることが GC の主目的である現在，本 GC は圧縮方式としてこれに大いに寄与するものと考える．本 GC は比較的小さな記憶空間を GC 対象領域としているにもかかわらず，効率的なデータオブジェクトの回収が高速に行われる．本 GC は計算機やオペレーティングシステムには依存せず，その効果が得られることが実験結果から示されている．

本GCの応用として，GC対象領域の大きさによって処理時間を規定するような実時間GCが考えられる．

謝辞 種々の有益なコメントをいただいた査読者に感謝します．

参 考 文 献

1) Wilson, P.R.: Uniprocessor Garbage Collection Techniques, Technical Report, University of Texas, Tex. (1994).
2) 寺島元章：PHLの新インタプリタ，記号処理研究会資料，SYM 73-5, pp.33–40 (1994).
3) 佐藤圭史，青木 徹，寺島元章：Alpha-chipマシン上のPHL処理系について，電子情報通信学会技術報告，COMP97-94, pp.57–64 (1998).
4) Sahlin, D.: Making Garbage Collection Independent of the Amount of Garbage, SICS Research Report R86008, Sweden (1987).
5) Suzuki, M. and Terashima, M.: Time- and Space-Efficient Garbage Collection Based on Sliding Compaction, Graduate School of Information Science Technical Report, University of Electro-Communications (1993). 情報処理学会論文誌，Vol.36, No.4, pp.925–931 (1995).
6) Terashima, M. and Goto, E.: Genetic order and compactifying garbage collectors, *Information Processing Letters*, Vol.7, No.1, pp.27–32 (1978).
7) 寺島元章，佐藤和美：可変容量セルの効率的なくず集めについて，情報処理学会論文誌，Vol.30, No.9, pp.1189–1199 (1989).
8) Morris, F.L.: Time- and Space- Efficient Garbage Collection Algorithm, *Comm. ACM*, Vol.21, No.8, pp.662–665 (1978).
9) Unger, D.M.: Generarion scavenging: A non-disruptive high performance storage reclamation algorithm. *ACM Conference on Practical Programming Environments*, pp.157–167 (1984).
10) Suzuki, M., et al.: MOA – A Fast Sliding Compaction Scheme for a Large Storage Space, *IWMM95 Memory Management*, Baker, H.G. (Ed.), LNCS, Vol.986, pp.197–210 (1995).
11) Chang, C.L.: The unit proof and the input proof in theorem proving, *JACM*, Vol.17, No.4, pp.698–707 (1970).
12) Hearn, A.C.: *REDUCE User's Manual, version 3.4*, The Rand Corporation, CA (1988).

(平成10年11月30日受付)
(平成11年2月8日採録)

寺島 元章（正会員）

昭和23年生．昭和48年東京大学理学部物理学科卒業．昭和50年同大学院修士課程，昭和53年同博士課程修了．理学博士．昭和53年より電気通信大学計算機科学科勤務．現在，同大学院情報システム学研究科助教授．プログラミング言語とその処理系，記憶管理方式，記号数式処理系等に興味を持つ．AAAI，ACM各会員．

佐藤 圭史

昭和49年生．平成9年電気通信大学電気通信学部電子物性工学科卒業．現在電気通信大学大学院情報システム学研究科博士前期課程在学中．記号処理系における記憶管理方式に興味を持つ．