圧縮方式による世代別ガーベッジコレクションの実装について

上野 真由子[†] 山室 弥久[†] 寺島 元章[†]

使用中データオブジェクトをヒープの一端にそれらの配置順序を保存して再配置する圧縮方式（mark-and-compact）に基づいた世代別ガーベッジコレクションの効率的な実装とその評価について述べる．圧縮方式のガーベッジコレクション（GC）を用いると，古いオブジェクト集団はヒープの一端に蓄積され，アロケーションポイント方向により新しいオブジェクト群が連続的に並び，この序列は永久に変化しない．本 GC は．ヒープの一定量が消費されるときに行われる新しいオブジェクトに対する処理と，より低い頻度で行われる古いオブジェクト群に対する処理からなる．効率上から後者は前者と組み合わせて実行される．GC の処理回数でオブジェクトの世代数を表すと，古いオブジェクト集団はヒープのアドレスで区分される複数の世代から構成される．各世代の量的変化はこれらのアドレスに反映される．そこで，アドレスの変化を追跡することで多世代管理に似た効果的な世代管理が実現できる．また，圧縮方式では，GC の対象領域のオブジェクトの生存分布がミクロ的に得られる．これはマクロ的な占有率より有益な情報であり，これを利用した対象領域の動的変更機能についても述べる．

Implementation of Generational Garbage Collection Based on Mark-and-compact

Mayuko Ueno,[†] Mitsuhisa Yamamuro[†]
and Motoaki Terashima[†]

This paper describes an efficient implementation and evaluation of generational garbage collection based on a sliding compaction (mark-and-compact) scheme that moves used data objects toward an end of a heap preserving their allocated order. The sliding compaction GC makes old data objects locate in an end of the heap and young data objects continuously toward allocation point, and the allocated order of these objects remains the same forever. The generational GC described here is made of two features; young data objects are processed when part of the heap is used, while old data objects are processed less frequently than the former. The latter is done in cooperation with the former for efficiency. The old objects are made up of one or more 'generations' separated by addresses of the heap, provided that the generation corresponds to the number of the GC that processes the object. The object size of each generation affects the address, so that a refined generational scheme similar to multi generational GC is realized by making use of such addresses. The sliding compaction GC gives us not only the load factor but also the distribution of alived objects. The generational GC also adopts dynamic adjustment of the scavenged space size makin g use of the latter that is more valuable than the former.

1. はじめに

ガーベッジコレクション（以下，GC と呼ぶ）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[1),2)]．GC は，ヒープに作られたデータオブジェクトでもう参照されることのない使用済みのオブジェクトを回収し，それらが占めていた領域を再利用する自動記憶管理機構のことである．

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

本稿では圧縮方式（sliding compaction）による世代別 GC の効果的な実装について述べる．世代別 GC は，「データオブジェクトの大多数は短命で，長命なオブジェクトはさらに生き長らえる」という仮説に基づき，各オブジェクトをその寿命によって新旧いくつかの世代の領域に分け，それぞれを個別に扱うことで GC 処理時間の短縮を意図したものである[3),4)]．

世代別 GC はその歴史的な経緯から，複写方式（copying collection）[5)] に基づいたものが大勢である．圧縮方式は複写方式に比べて，移動オブジェクト量あたりの処理時間が長い点で不利である．しかし，ヒープ全体あるいはその一部の領域を連続して使用するこ

とができることや，各オブジェクトが領域中でつねにその作られた順序（これを生成順序[6]と呼ぶ）で並ぶという利点を持つ．それは，GC の処理を経て，古い世代のオブジェクトは領域の一端に，そして新世代のオブジェクトはその他端方向に「自然に」配置されることを意味する．

本稿で提案する世代別 GC はこのような圧縮方式の特徴を利用して，

(1) 段階的
大容量ヒープの一部を効果的に使用する，
(2) 自在性
古い世代への移行を細かに調整する，
(3) 可変性
各世代の領域量は可変である，
(4) 効率的
GC 処理の階層化，

という特徴を持つ．

新世代のオブジェクトを処理する GC（いわゆる，新世代 GC）の対象領域はキャッシングが有効に働く比較的狭い領域☆で，大容量のヒープの一部である．1 回以上の新世代 GC を経たオブジェクトは「殿堂入り」を果たし，古い世代となる．新世代 GC は殿堂入りに際して，副次的に得られる対象領域のオブジェクトの生存分布を利用する．これは 100 分割かそれ以上に細分化された各小区画中の使用中オブジェクト比率（占有率）のことで，殿堂入りするオブジェクト群を効果的かつ緻密に決めることができる．残されたオブジェクト群は再度，新世代 GC の対象となる．多世代 GC☆☆は処理対象となった各世代を 1 つ古くするとともに，容量が変化しない古い世代を今後の GC 処理対象から外すことを行う．

本 GC は圧縮方式の枠組みの中で実現されるものであるが，複写方式に基づく既成の世代別 GC との比較についても述べる．そして，この GC を PHL（Portable Hashed Lisp）翻訳系[7]の実行環境下で実装し，評価を行う．

2. 世代別 GC

世代別 GC は，先に述べたデータオブジェクトの寿命に関する仮説に基づいてオブジェクトをその寿命によっていくつかの世代に分け，新世代 GC よりも低い頻度で古い世代のオブジェクトを処理する GC（いわゆる，旧世代 GC）を起動させることで，各回の GC 処理時間と全体の GC 処理時間の短縮を意図した手法である．

世代別 GC はその歴史的経緯から，複写方式か自由リスト方式☆☆☆に基づくものが大勢である．複写方式は，移動先・移動元という 2 つの半領域のいずれかを一度は空にする必要があり，オブジェクトの移動が位置関係でなく参照関係に基づくという問題がある．前者は必然的にオブジェクトの移動量を増加させるし，後者は古い世代の領域にオブジェクトを移す最適な時期を決める「殿堂入り」（tenuring）問題の解決を難しくする．自由リスト方式は，オブジェクトの再配置を行わないため，局所性や多様性に効果的に対処できないという問題点がある．

2.1 殿堂入り問題

殿堂入り問題とは，ある世代のオブジェクトがそれより（1 つ）古い世代のオブジェクトになる最適な時期をどのように決めるかということである．つまり，

- 殿堂入りを遅らせると ···
該当オブジェクトに対する GC 処理が増え，時間効率を低下させる，
- 殿堂入りを早めると ···
死蔵オブジェクト（使用済みでも回収不能なもの）を増加させ，時間と記憶効率を低下させる，

となり，GC と処理系全体の効率を大きく左右する．この殿堂入り問題を解決するために，殿堂入りする候補（オブジェクト）の量で決める方式[8]や，経過した GC 回数で決める方式[9),10)]，新世代の領域のサイズを動的に変更する方式[11)]など，これまでにいくつかの手法が提案されている．しかし，ある程度の効率の改善はされるものの，すべてのプログラムに対して万能とはいかないのが現実である．

殿堂入り問題の解決を難しくしている理由として，複写方式ではオブジェクトの移動に際して位置関係が保存されないという根本的な原因がある．つまり，オブジェクトが参照関係優先で，GC の経験回数などの寿命に関係なくごっちゃに移されることがあるからである．

GC の経験回数で殿堂入りを決める方式では，1 回目や 2 回目での設定は技術的な問題もなく容易に行えるが，3 回目となると格段に難しくなる．たとえば，2 回目では，1 回目の GC を経験したオブジェクト群とそれ以外の新たなオブジェクト群は半領域中で明確に

☆ 使用する計算機の二次キャッシュ容量に収まることが望ましく，2〜4 MB 程度である．

☆☆ 新世代とその他古い世代のオブジェクトの処理を一括して行う GC のこと．

☆☆☆ オブジェクトの再配置のない GC で，mark–and–sweep とも呼ばれる．

分離しているからである．しかし，3 回目ではこの両者が分離されて移される保証はなにもない．そこで，殿堂入りオブジェクトを識別するには各オブジェクトに経験回数の情報を付けるなどの付加的な処理が必要となる．また，候補となるオブジェクトの量で殿堂入りを決める場合もこうした処理が必要となる．

3. 圧縮型 GC

圧縮方式に基づく GC を圧縮型 GC と呼ぶ．圧縮型 GC の特徴の 1 つは生成順序の保存である．そこでは，生成時に作られたヒープ上での順序関係が恒久に保存され，結果的に，各オブジェクトは作られた順に古いものから新しいものへとヒープに並ぶ．GC 時に行われるオブジェクトの圧縮の始点となるヒープのアドレスを記録すれば，GC の経験回数によるオブジェクト群の識別が容易にできる．圧縮型 GC の下では，各世代を区切るのはこうしたアドレスである．

もう 1 つの特徴はヒープ中のオブジェクトのミクロ的な情報が副次的に得られることである．ポインタ補正値と呼ぶ，使用中オブジェクトが圧縮により移動する量を求めるために，ヒープあるいはクラスタ（使用中オブジェクトの連続した塊）が走査される．この走査過程で，ヒープ中の使用中オブジェクトの分布データを負荷なく得ることができる．なお，停止回収型☆では，この走査回数は 2 回である．この回数は，印付け（marking）をマークビット表のような外部表で行う場合も，各オブジェクトのビット表現の 1 ビットを用いて行う場合も同じである[12]．

3.1 高速圧縮型 GC

最近の圧縮型 GC の高速化に関する研究は，$O(A)$ の時間計算量（time complexity）を持つ高速圧縮型 GC を生み出している[13),14)]．ここで，A は使用中データオブジェクトの総容量である．

この種の GC は，使用中データオブジェクトのクラスタの先頭アドレスをデータとして大小順にソートし，このソート済みデータを利用してヒープ中のクラスタのみを走査する．そのため，時間計算量は，

$$O(A) + O(n \log n) \quad (1)$$

となる．n はクラスタの総数である．

一般的なアプリケーションでは使用中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．そこで，式 (1) は複写型 GC の時間計算量と同じく

$$O(A) \quad (2)$$

と近似できるのである．

表 1 は数式処理システム REDUCE[15)] 上で実行した行列計算（Mat）の実行結果である．そのプログラムは 6 行 6 列の逆行列を求めその検算を行うものである．高速圧縮型 GC と複写型 GC の両者が記載されているが，それぞれの右端の値（Tgc/M）はほぼ同じである．これから，両者とも実行プログラムに関して，移動したオブジェクトの容量（M）に比例した時間で処理を行っていることが分かる．

表 1 Mat の GC 処理時間（1）
Table 1 Processing time of Mat on GC (1).

S	Tgc (times)	Ttotal	M	Tgc/M
高速圧縮型 GC				
3 MB	7.51 (56)	34.53	20.0	0.376
6 MB	4.28 (22)	31.50	10.2	0.421
12 MB	1.20 (10)	28.35	2.6	0.459
複写型 GC				
6 MB	2.75 (56)	29.96	55.3	0.050
12 MB	1.15 (22)	28.96	21.8	0.053
24 MB	0.43 (10)	28.52	7.0	0.061

（注）S：ヒープ容量
Tgc：GC 処理時間, ( ) 内は GC 回数
Ttotal：全処理時間，単位はいずれも秒
M：移動量の総和，単位は MB.

表 1 では，圧縮型 GC のヒープ（S）と複写型 GC の半領域を同じにして対比してある．当然，GC の回数は同じである．複写型 GC が 3 倍ほど高速なのは，複写型 GC の作業がオブジェクトの移動とポインタの付け替えであるのに対し，圧縮型 GC は印付け，補正，移動という多くの作業がともなうからである．

高速圧縮型 GC で世代別管理を行っている GC としては，これまでに便宜的 GC[12)] とハイブリッド GC[16)] が知られている．これらの GC と本稿で提案する GC との相違点は 6 章の関連研究のところで詳しく述べる．

3.2 生成順序の保存

任意の 2 つのデータオブジェクトの間にはそれらが置かれたヒープ中のアドレスで規定される生成順序といわれる順序関係が成立する．たとえば，アドレスの増加する向きにアロケーションポイント（最新のオブジェクト）が移動するならば，新しいオブジェクトは古いオブジェクトに対して必ず「大なり」の関係が成立する．圧縮型 GC の処理下ではこの関係はつねに保存されるが，通常の複写方式ではこの保証はない．

生成順序を利用した実例として，WAM に基づく Prolog 処理系[17),18)] がある．この処理系は任意の 2 つの選択点（choice points）の生成順序が恒久的に保存されることを利用している．この順序関係が保存され

☆ GC がある対象領域に対して一連の処理をしている間に GC 以外の処理が停止するもののこと．

るならば，どちらが新しいかは単にそれらのアドレスを比較すればよいのである．これは処理系の高速化に直結する．

4. 提案手法

この章では，世代別管理を圧縮方式に取り入れたGCの概要とその実現法について述べる．

4.1 着想点

本GCの設計では圧縮方式の特徴を十分に生かすことを考えた．その1つは，最新のオブジェクトが置かれたアロケーションポイントから任意のオブジェクト境界までGCの回収領域にできることである．ただし，回収領域に含まれない（古い）オブジェクトから参照されるオブジェクトのために，リメンバードセット（remembered set）[4]のような仕掛けが必要となる．回収領域中の使用中オブジェクトの総量は圧縮された塊の量としてヒープアドレスの差異で計ることができる．これはGCの対象となった各世代オブジェクトの変動量の算出に利用できる．

次は，回収領域の走査を利用して精緻なオブジェクト分布が得られることである．領域全体での占有率のようなマクロ的な情報ではなく，クラスタ単位で計れるミクロな情報である．この情報を活用すれば，次回の回収領域を精緻に設定することができる．

最後は，最近の計算機アーキテクチャに見られる高速記憶（キャッシュ）の活用である．これは圧縮方式に限ったことではないが，頻繁に使用される新世代の領域がキャッシュに置かれれば，キャッシュヒットの効果により処理の高速化が期待できる．その効果が最も期待される一次キャッシュ容量はたいてい数十KB程度で，これを利用するには新世代領域を小さくとらなければならない．しかし，二次キャッシュは数MBを搭載した計算機もあり，今回はこちらを活用した．

図1は圧縮方式でのヒープの使用法を示したものである．図1(a)は古典的な方法で，ヒープがすべて消費尽くされたらGCが起動して，使用中オブジェクトをその一端に圧縮するものである．図1(b)は今回採用した方法で，ヒープの一部を段階的に消費していくものである．この利点は，consumedと表記された，新たなオブジェクトのための領域をキャッシュ容量に見合うように設定できることである．この領域量はヒープの容量以下ならば任意である．

4.2 GCの構成

本GCは多世代のオブジェクトの管理を統合して行うことができる．機能的には，新世代のオブジェクトに対する処理を行う新世代GCと，それを含む形での多世代GCとからなる．実装上も，新世代GCのルーチンを組み込む形で多世代GCが作られている．

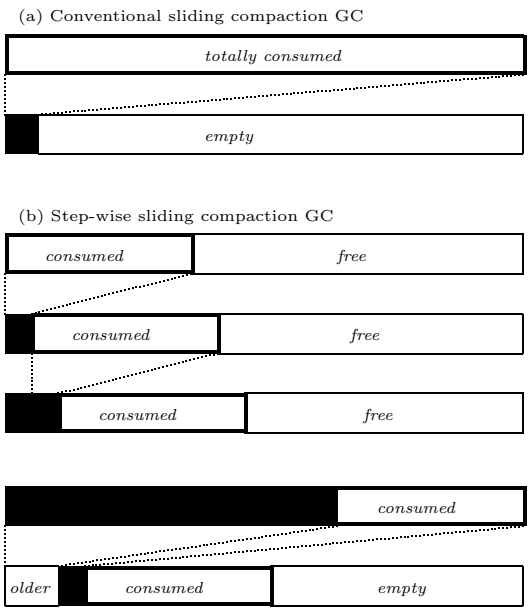


図1 ヒープ使用法
Fig. 1 Heap utilization.

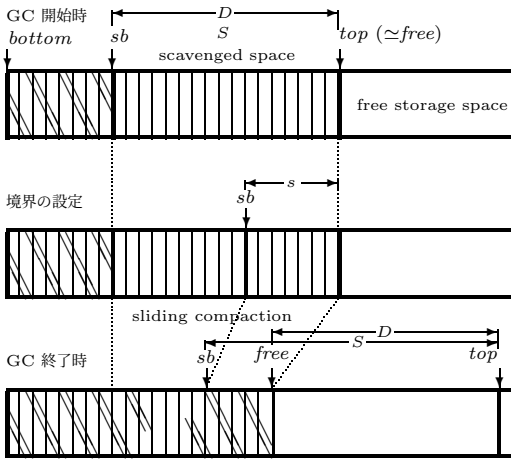
GCが起動されたときに，そのどちらになるかはヒープの消費状況による．多世代GCが起動するのはヒープがすべて消費尽くされたときである．図1(b)では上から4番目である．多世代GCは古いオブジェクトの領域（図中の黒帯）を含む回収領域（図ではヒープ全体）中の使用中オブジェクトを左端に圧縮する．圧縮された古いオブジェクトは1つ古い世代のオブジェクトになり（図中のolder），新しいオブジェクトは古いオブジェクトになる．ここでは，圧縮方式の機能により世代の順に位置が固定することが重要である．

多世代GCは当然，キャッシュサイズを超える領域を処理することになる．この点で時間的負荷が大きくなるが，その起動回数は新世代GCよりも格段に少ない．また，処理時間が使用中オブジェクトの総量に比例する高速圧縮型GCの技法を使用しているため，負担増は大きな問題とならないと考える．

4.3 新世代GC

新世代GCは，高速圧縮型GCと同様に，印付け，クラスタ表の作成，補正表の作成，ポインタ補正とオブジェクトの移動という各段階を進む．この間，ヒープの断片は2回走査される．これらの各段階で使用されている高速化技法については他に詳細な文献[12),13),16)]もあるので，ここでは世代別管理の手法だけを述べる．

新世代GCは新たに作られたオブジェクトと1回以



（注） *free*：アロケーションポイント

図 2 新世代 GC のヒープ処理
Fig. 2 Process of new generation GC.

上（新世代）GC を経験したオブジェクトを対象にする．後者は，いわゆる殿堂入りをしなかったオブジェクトである．新世代 GC はヒープが一定量（**図 2** の D）消費されたときに起動される．その回収対象領域（図 2 の S）は一般に D よりも大きい．S から D を除いた部分は新世代 GC を経験したオブジェクトが存在する領域である．ただし，図では説明の便宜上，最初は $D = S$ としてある．

新世代 GC は補正表を作成するために回収対象領域中の各クラスタを順次走査する．この 1 回目の走査時に，回収対象領域のオブジェクト分布の情報に基づいて *sb* という境界を決める．これは，殿堂入りオブジェクトと次の新世代 GC の対象となるオブジェクトの境，いわゆる新旧世代の境界である．その決定の指標となるのは，

(1) ヒープサイズの比率
(2) 占有率

である．1 番目は回収領域全体に対する新世代領域として残る領域のサイズ比（$|s|/|S|$）である．2 番目は小区画あたりの占有率（使用中オブジェクトの容量比）である．小区画とは回収領域をほぼ 100 個に分けた小さな領域のことである．その容量は厳密には同じでなく，境界となったクラスタのサイズに依存する．この両者を閾値に用いることで，殿堂入りのタイミングの細かな調整ができるのが本 GC の大きな特色である．

新世代 GC は，*sb* をまたぐ旧世代のオブジェクトから新世代のオブジェクトを指すポインタをリメンバードセットに登録する．この更新はオブジェクトを移動する 2 回目の走査時に行われる．この更新を効率的に行うために，最初の走査時で *sb* を確定するのである．あとは，*sb* の補正を行えばよい．次回は，*sb* から *top* までが新世代 GC の回収領域となる．これは，D よりも大きい．

境界（*sb*）は回収領域内ならば任意のオブジェクト境界に設定することができる．それは明らかに，GC 経験回数によらないし，その差異で分断する必要もないのである．このように，殿堂入りに対する GC 回数の設定に関しては既成の世代別 GC よりも多くの自由度を持つのである．

4.4 多世代 GC

多世代 GC は，殿堂入りとなった古い世代のオブジェクトが占める領域（旧世代領域）の回収も行う．古い世代のオブジェクトの移動は新しいオブジェクトの移動と同時に，圧縮処理の枠組みで効率的に行われる．これは両者の領域が隣接していることによる．

新世代 GC と異なるのは，処理の対象となった古い各世代についてそのオブジェクト減少度を調べることである．そして，数回の多世代 GC で変化のない最古参の世代については，これ以後の GC 処理から外すようにする．これらは，「コアデータ」のような永続的なオブジェクトと考えられるからである．

各世代をまたぐポインタの記録に必要なリメンバードセットは 1 つの表で効果的に管理できる．これは，各ポインタの始点が世代順に並ぶからである．ただし，世代が増えると，記録すべきポインタの数も通常，増える．これは，リメンバードセットを用いるどのような方式にもいえることである．

5. 実装と評価

5.1 実験環境

提案手法の性能評価は PHL 翻訳系の実行環境下で行い，言語やその処理系の特性に依存しない普遍的な結果を得ることを目標とした．こうした環境では，純粋に機械語のプログラムが生成するデータオブジェクトのみが GC の対象になるからである．なお，PHL は可搬性のある Lisp 処理系で，これまでに，Sparc, Alpha, MIPS, Pentium, MC68000 などの命令セットアーキテクチャを持つ計算機で稼働している．

今回の実験に使用した計算機は Sun Microsystems 社のワークステーション Enterprise 450（UltraSparc 300 MHz を 4 個搭載）である．その一次キャッシュ容量は 32 KB（命令 16 KB，データ 16 KB），二次キャッシュ容量は 2 MB である．また，主記憶容量は 256 MB（アクセスタイム 60 nsec.）である．

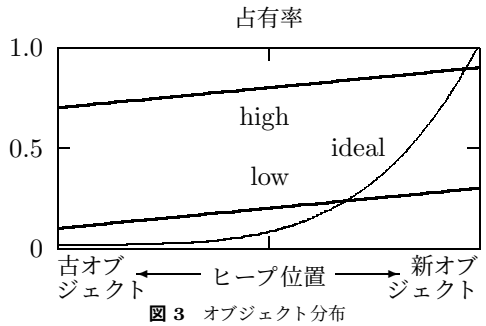


図 3 オブジェクト分布
Fig. 3 Objects distribution.

5.2 殿堂入り閾値の設定

図 3 は回収領域中の使用中オブジェクトの分布，いいかえれば，小区画あたりの占有率を模式的に示したものである．横軸は回収領域上の位置を，縦軸は占有率を表す．この図で示された 3 つの分布曲線は必ずしも実際とは合致しないかもしれないが，世代別 GC で効果的に処理できるのは，ideal で示されるような指数分布に近いオブジェクト分布である．占有率が高止まりする分布に対しては時間的損失が多いし，逆に低すぎる分布は領域的利得が少ないからである．

そこで，新世代と旧世代の領域の境界を，回収領域に対して

(1) 左半分の占有率が 0.1 から 0.4 の間にある，
(2) 左半分の占有率が 0.1 以下で，かつ全体の占有率がその 1.2 倍以上，

といういずれかの条件を満たす場合に，左右の中間の位置に設定した．これは，現回収領域の半分より左にある古いオブジェクトは必ず殿堂入りすることを意味する．これに加えて，(1) の条件は占有率が極端に高い場合（0.4 以上）や低い場合（0.1 以下）にすべての使用中オブジェクトを殿堂入りさせる．しかし，(2) の条件は占有率が低い場合でも指数関数的に増加する場合は除く（回収領域の左半分にあるオブジェクトだけを殿堂入りさせる）ことを規定する．この殿堂入りのタイミングの条件は，GC 経験回数でいわれるところの 1 回以上で 2 回未満となる．

5.3 評　　価

表 2 は表 1 に示した行列計算（Mat）の本 GC 下での実行結果である．ヒープサイズは 5 MB である．「閾値」で示される GC 経験回数で殿堂入りを決めた場合，GC 間での消費量（D）によらず，1 回か 2 回の場合に処理時間の短縮が実現されている．3 回以上の場合はこの表の結果からオブジェクトの移動量の増加にともない GC 処理時間が増加すると予想される．

表 3 は GC 経験回数でいうところの 1 回から 2 回の間をさらに細かく設定した場合の本 GC の結果である．閾値で示される％は収領域のその比率相当分を新世代領域として残す．この部分に存在するオブジェクトはすべて次回の GC の処理対象になる．この比率が大きくなれば，M の値が示すようにオブジェクトの移動は増えて，GC 処理時間を増加させる．しかし，次回の回収領域は大きくなり，結果的にヒープの使用は比較的狭い範囲にとどまることになる．いわゆる局所性が向上し，この表にはないが多世代 GC の回数が減る．

表 2 Mat の GC 処理時間 (2)
Table 2 Processing time of Mat on GC (2).

閾値	D	Tgc	Ttotal	M	E
GC	3	11.85	38.61	13.51	370.7
3 回	2	6.38	33.11	16.40	317.0
	1	5.35	32.12	25.81	266.1
GC	3	5.60	32.93	12.51	317.6
2 回	2	2.98	30.24	16.38	267.6
	1	4.54	32.09	26.71	181.0
GC	3	5.95	33.76	12.93	0
1 回	2	2.83	29.60	16.59	0
	1	3.96	30.84	22.83	0

（注）Tgc：GC 処理時間
Ttotal：全処理時間，単位はいずれも秒
M：移動量の総和，単位は MB
E：回収領域の増分量，単位は KB
D の単位は MB.

表 3 Mat の GC 処理時間 (3)
Table 3 Processing time of Mat on GC (3).

D	閾値	Tgc (times)	Ttotal	M	E
提案手法					
3	25%	5.65 (38)	32.57	12.48	67
	50%	5.58 (38)	32.53	13.62	148
	75%	5.55 (38)	32.46	12.50	245
	50%-α	5.60 (38)	32.52	13.53	111
2	25%	2.80 (57)	29.70	16.12	91
	50%	2.83 (57)	29.78	16.20	183
	75%	2.96 (57)	30.04	16.37	252
	50%-α	2.75 (57)	30.01	16.17	85
1	25%	4.20 (114)	31.07	24.05	43
	50%	4.29 (114)	31.01	25.78	125
	75%	4.68 (114)	31.62	26.84	184
	50%-α	4.10 (114)	30.90	23.61	52
ハイブリッド GC（S は 0.4×D に設定）					
2	GC	2.80 (57)	29.92	13.13	—
1	1 回	3.27 (114)	30.73	16.45	—
世代別複写型 GC（D は新世代領域の総量）					
4	GC	1.57 (66)	28.44	20.00	—
2	2 回	2.78 (138)	29.58	29.70	—
1		3.85 (287)	30.87	38.47	—

（注）Tgc の（）内は GC 回数
他の記号の意味と単位は表 2 と同じ
提案手法の上段の 3 つは閾値（領域比）で境界を決めた場合，下段は占有率を加味して決めた場合．

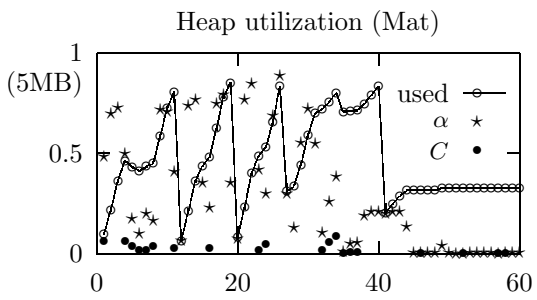


図 4 新世代 GC 下での Mat のヒープ消費
Fig. 4 Heap utilization of Mat program on new generation GC.

表 3 の閾値が 50%−α のものは 5.2 節の条件を適用した場合である．D が 1 MB と 2 MB のときでは，条件を適用した方が明らかに処理時間が短く，殿堂入りの設定が有効に働いていることが分かる．なお，この設定はオブジェクトの GC 経験回数でいえば 2 よりも 1 に近いところである．これが，回数を比較的大きくとると効果のあるといわれる複写方式に基づく世代別 GC と異なる点である．

表 3 にはハイブリッド GC と複写方式の世代別 GC の結果も記載してある．ハイブリッド GC のヒープサイズは同じく 5 MB である．GC 対象領域量（S）は D の 40%である．ハイブリッド GC では S/D の比率の設定に自由度があり，この比率が変われば処理時間も変わる．測定した中ではこの比率のときが全処理時間が最短であった．本 GC は「最速」のハイブリッド GC に対して同等かわずかに劣るだけである．

世代別複写方式は一般的な「新旧各 2 面」方式を採用し，GC を 2 回経験した新世代オブジェクトを殿堂入りさせる．旧世代オブジェクトの領域（半領域）が満たされると，旧世代 GC がその使用中オブジェクトを他方（半領域）に移す．表中の D の値は新世代オブジェクトの領域量（半領域の 2 倍）である．

この実験からは，使用した計算機の二次キャッシュ容量に見合う D=2 MB よりも，D=4 MB の方が全処理時間が短くなるという結果が得られている．GC 処理時間を比較すると，複写方式は本 GC の 60%程度である．なお，この表から全処理時間の両者の差は GC 処理時間の差よりも小さいことが分かる．これは，圧縮方式の特徴であるデータオブジェクトの配置に関する局所化の効果であると考えられる．

図 4 は，D を 1 MB にして Mat を実行したときのヒープ使用の様子を GC の回数（横軸）ごとに表示したものである．図中の used で示される白抜きの折線は GC 終了後のアロケーションポイントの開始点である．各回の占有率は星印が示しているが，プログラムの実行を通して大きく変化する．占有率が高い場合にはアロケーションポイントの開始点の伸びも大きいが，占有率が 0 の近傍であればアロケーションポイントの開始点はほとんど変化しない．こうしたことは通常の圧縮型 GC にもいえることでもある．

表 4 Fourier の GC 処理時間
Table 4 Processing time of Fourier on GC.

D	閾値	Tgc (times)	Ttotal	M	E
提案手法					
2	00%	0.46 (27)	32.77	2.85	0
	50%	0.49 (27)	32.35	3.41	57.6
	50%-α	0.48 (27)	32.09	3.31	42.4
1	00%	0.53 (55)	32.81	3.56	0
	50%	0.66 (55)	32.34	4.32	41.1
	50%-α	0.64 (55)	31.64	4.07	30.2
ハイブリッド GC（S は 0.4×D に設定）					
2	GC	0.52 (27)	33.96	3.90	—
1	1 回	0.63 (55)	34.02	4.36	—
世代別複写型 GC（D は新世代領域の総量）					
4	GC	0.27 (29)	33.48	3.56	—
2	2 回	0.37 (59)	32.97	4.38	—
1		0.51 (120)	33.54	5.24	—

（注）記号の意味と単位は表 3 と同じ．

本 GC ではアロケーションポイントの開始点が後退する（図では右下がりになる）こともある．これは S が D よりも大きくなることによる副作用であるが，一時的にせよヒープの比較的狭い範囲が頻繁に使用され，キャッシングが有効に働くことになる．この現象は図では GC の 7 回目付近と 35 回目付近に現れている．なお，黒丸は両者の差（S−D）である．ただし，D=S の場合（0 の値）は表示されていない．

表 4 は，REDUCE 上で実行したフーリエ解析計算（Fourier）の実行結果である．前述の Mat と異なるのは占有率である．移動量からも分かるように Fourier の方がかなり小さい．このプログラムにおいても，占有率に関する条件を適用した方が良い結果となっている．本 GC は処理時間においてハイブリッド GC より優位にある．世代別複写方式と比較すると，GC 処理時間においては 1.72 倍程度の差があるが，全処理時間ではわずかに優位になる．なお，複写方式では新世代オブジェクトの領域量を使用計算機の二次キャッシュに見合う 2 MB にした方が全処理時間が短くなった．これが行列計算の場合と異なる点である．

今回の設定は個々の小区画の占有率を直接使用したわけではなく，単に 2 つの平均値を利用したにすぎないが，それでも比較的良好な結果が得られている．これは，次回の新世代 GC で領域的な利得が得られるオブジェクト群を 2 種の占有率だけで効果的に選別でき

たことを意味している．要するに，占有率が極端に高いか低すぎるオブジェクトの集団と，小区画の占有率がアロケーションポイントの移動方向に増加するオブジェクトの集団を識別するにはこれらのデータで十分であったのである．

回収領域は新世代 GC が処理したオブジェクト群の特性によって，拡大したり縮小したりというようにその量は動的に変化する．いわば，ある程度分かったオブジェクトの特性に基づいて量を変えるのであって，今後作られるであろう「未知」のオブジェクト群を想定して量を変えるのではない．したがって，仮にそれらの特性が変化したとしても拡大や縮小が過度になることも過大に振動することもない．

6. 関 連 研 究

6.1 便宜的 GC

便宜的 GC は単純な世代別管理を実現した高速圧縮型 GC の一種である．その回収領域は最新に作成されたデータオブジェクトが存在する領域で，D の一部である．本 GC の S を *top* 側に縮めたものと考えればよい（図 2 参照）．利点は，回収領域が小さくかつ単純な世代別管理により GC 処理時間の短縮が実現されること，回収領域に存在する使用中オブジェクトの圧縮にともなうワーキングセットの減少により全処理時間の短縮が図られることである．

世代別管理は 1 回の GC を経たオブジェクトがすべて殿堂入りすることである．いわゆる新世代 GC も旧世代 GC もない単純な仕掛けである．

便宜的 GC の問題点は回収領域に含まれない D の一部（非回収領域）に存在する使用中オブジェクトが未処理で残されることである．両者の境界から非回収領域を進んで初めて使用中オブジェクトが現れるまでの区間を効果的に見つけ，その部分を再利用することは行っている．しかし，非回収領域に存在する使用中オブジェクト群は残りの部分を再利用できない領域にする．最悪の場合は非回収領域すべてが再利用できなくなる．結果として，ヒープが際限なく単調に消費し続けられる．これが便宜的 GC の大きな問題である．

6.2 ハイブリッド GC

ハイブリッド GC は便宜的 GC の問題点の克服を目指して考案された．その基本アイデアは間歇的に全面回収を行うことである．それまでヒープに散在した使用中オブジェクトをヒープの一端に圧縮し，再利用可能な（大きな 1 つの）領域を作ることである．ヒープの全面にわたり回収を行うのは，世代管理に必要なリメンバードセットの構造を単純化するためである．

これを世代別 GC にあてはめると，1 回の新世代 GC を経て「殿堂入りした」オブジェクトが，数回の新世代 GC に対して 1 回の（旧世代）GC 処理の対象になるということである．

本 GC とハイブリッド GC の大きな違いは殿堂入りのタイミングである．本 GC は 1 回以上可変であるのに対して，ハイブリッド GC は 1 回の固定である．オブジェクトの生存分布などプログラムの特性が変わりうる状況では固定より可変の方が効果的である．前述の Fourier での本 GC の優位性はこの例証でもある．もう 1 つは，ユーザが回収領域量（S）を指定する必要がないことである．ハイブリッド GC や便宜的 GC ではユーザが S を決めるが，処理時間の短縮が図られる適切な S を見つけるのは容易ではない．

7. お わ り に

本稿では圧縮方式に基づく世代別 GC の効果的な実装と Lisp プログラムの実行結果に基づいた評価を述べた．圧縮方式の利点である生成順序保存がヒープ上での各世代の序列を一義的に決めること，回収領域の走査で副次的に得られる生存オブジェクトの情報など，複写方式に基づく既成の世代別 GC には見られない特徴を利用して各世代を効果的に管理する手法を提案し，それらを実装した．

本 GC は占有率が低く GC 処理時間の比較的短いプログラムの処理では通常の世代別複写方式とほとんど遜色ないが，それ以外のプログラムにおいて時間性能で劣ることは明らかである．

本 GC を含む圧縮型 GC は，それが動作する処理系でプログラマはデータオブジェクトの生成順序を利用した効率的なプログラムが書けるという意義を持つ．著者らが知る限りこの種のプログラムはまだ数例しかないが，順序関係の利用はアルゴリズム構築の基本であるので，この順序関係を利用した高速アルゴリズム（プログラム）の時間的な利得は GC 処理時間の両者の差を相殺するかもしれないのである．

参 考 文 献

1) Wilson, P.R.: Uniprocessor Garbage Collection Techniques, Technical Report, University of Texas, Tex. (1994).
2) Jones. R. and Lins, R.: *Garbage Collection*, John Wiley & Sons, UK (1996).
3) Liebeman, H. and Hewitt, C.: A Real-time Garbage Collector based on the Lifetimes of Objects, *CACM*, Vol.26, No.6, pp.419–429 (1983).

4) Unger, D.M.: Generarion Scavenging : A Non-disruptive High Performance Storage Reclamation Algorithm, *ACM Conference on Practical Programming Environments*, pp.157–167 (1984).
5) Fenichel, R.R. and Yochelson, J.C.: A Lisp Garbage Collector for Virtual Memory Computer Systems, *CACM*, Vol.12, No.11, pp.611–612 (1969).
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，山本洋司，古川敦司，渡辺美苗：PHL の新コンパイラ，記号処理研究会資料 SYM 78, pp.17–24, 情報処理学会 (1995).
8) Unger, D. and Jackson, F: An Adaptive Tenuring Policy for Generation Scavengers, *ACM Trans. Prog. Lang. Syst.*, Vol.14, No.1, pp.1–27 (1992).
9) Wilson, P.R. and Moher, T.G.: Design of the Opportunistic Garbage Collector, *ACM OOPSLA'89*, *SIGPLAN Notices*, Vol.24, No.10, pp.23–35 (1989).
10) 田中詠子，田中良夫，中西正和：Adaptive Garbage Collection — 実装とその評価，電子情報通信学会誌，Vol.J79-D-1, No.5, pp.253–260 (1996).
11) 吉川隆英，近山 隆：効率のモデルに基づきヒープサイズを自動調整する世代 GC 方式，情報処理学会論文誌：プログラミング，Vol.41, No.SIG9(PRO 8)，pp.78–86 (2000).
12) 佐藤圭史，寺島元章：圧縮型ガーベッジコレクションの高速化，情報処理学会論文誌，Vol.40, No.5, pp.2397–2403 (1999).
13) Suzuki, M., Koide, H. and Terashima, M.: MOA — A Fast Sliding Compaction Scheme for a Large Storage Space, *IWMM95*, Baker, H.G. (Ed.), LNCS 986, pp.197–210, Springer-Verlag（1995).
14) Chung, Y.C., et al: Reducing Sweep Time for a Nearly Empty Heap, *POPL '00*, ACM, pp. 378–389 (2000).
15) Hearn, A.C.: *REDUCE User's Manual, version 3.4*, The Rand Corporation, CA (1988).
16) 宮本崇生，寺島元章：ハイブリッドガーベッジコレクションの実装と評価，情報処理学会論文誌：プログラミング，Vol.40, No.SIG7(PRO 4), pp.24–31 (1999).
17) Appleby, K., et al.: Garbage collection for Prolog based on WAM, *CACM*, Vol.31, No.6, pp.719–741 (1988).
18) Bekkers, Y., Ridoux, O. and Ungaro, L.: Dynamic Memory Management for Sequential Logic Programming Languages, *IWMM92*, LNCS 637, pp.82–102, Springer-Verlag（1992).

付　　録

以下は今回使用したテストプログラムの説明である．

- Mat：
  6 行 6 列の行列（各成分は 1 変数の記号）

$$A = \begin{pmatrix} a_{11}\, a_{12} \ldots a_{16} \\ \cdots \\ a_{61}\, a_{62} \ldots a_{66} \end{pmatrix}$$

  の逆行列 A^{-1} を求め，$A \times A^{-1}$（=E）を行う．
- Fourier：
  cos と sin の 6 次式
  $(a_1 \cos wt + a_3 \cos 3wt + a_5 \cos 5wt + b_1 \sin wt + b_3 \sin 3wt + b_5 \sin 5wt)^6$
  を展開して，cos と sin の 1 次式に変形する．

(平成 13 年 5 月 31 日受付)
(平成 13 年 8 月 31 日採録)

上野真由子
昭和 52 年生．平成 12 年東邦大学理学部情報科学科卒業．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に在学中．記号処理系における記憶管理方式に興味を持つ．

山室　弥久
昭和 51 年生．平成 12 年徳島大学工学部知能情報工学科卒業．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に在学中．プログラミング言語処理系における記憶管理方式に興味を持つ．

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．ACM 会員．