Vol. 35 No. 11 情報処理学会論文誌 Nov. 1994

ショートノート

生成順序の保存に基づくコピー方式世代管理の一方法

小出 洋† 鈴木 貢† 野下浩平†

オブジェクトの生成順序を保存するコピー方式ガーベジコレクションを利用してオブジェクトの世代管理を行う新しい方法を示す．世代管理はコピー方式ガーベジコレクションを使用してよく実現されるが，従来の方法では，純計算時に古い領域から新しい領域を参照する特別なデータ構造と処理が必要であった．本方法では，ガーベジコレクションのためのそうした処理を純計算時に必要としない．その実現の考え方は，古い領域から新しい領域を指すポインタがないように新旧の領域の境界を求め，生成順序保存を利用して最も古いオブジェクトから先に古い領域にプロモートすることである．

An Algorithm of Generational Copying Garbage Collection Based on the Generated-Order-Preserving Property

Hiroshi Koide,† Mitsugu Suzuki† and Kohei Noshita†

We present a new method for generational copying garbage collection. The copying scheme has been widely used for implementing generational garbage collectors. All the previous methods require certain kinds of special tables for handling pointers from old objects to new objects at runtime. Our method requires none of those tables. The basic idea is to compute the boundary between the new region and the old region in order to remove all reverse pointers. By using the generated-order-preserving property, older objects can always be promoted to the old region earlier than younger objects.

1. はじめに

コピー方式ガーベジコレクション[2),3)]（コピー方式GC）は，いままでにオブジェクトの世代管理[1),6),7)]に多く用いられている．本稿では，オブジェクトの生成順序を保存する性質を持つコピー方式GC（生成順序保存型コピー方式GC）を利用してオブジェクトの世代管理を行う新しい方法を示す．純計算時にGCのための特別な処理を行わないという条件のもとで，オブジェクトの世代管理を行うことを目的とする．従来の世代管理を行うGCでは，純計算時にGCのための何らかの処理が必要であった．

本稿の方法では，記録領域を新旧の二つの領域に分割する．新しい領域は，更に同じ大きさの二つの半領域に分割される．本方法の世代管理では，新しい領域の二つの半領域の間で2回か3回コピーされた使用中オブジェクトを古い領域に移動する（プロモーション）．ただし，後で述べる最大ポインタの方が大きい場合は，その最大ポインタが指すオブジェクトとそれよりも古いオブジェクトまでをプロモートする．後者の場合は，一度コピーされると同時にプロモートされる場合もある．

新しい領域の二つの半領域間のコピーは生成順序保存型コピー方式GC[4)]を使用して行う．本方法のプロモーション戦略は生成順序が保存されていることを利用して，最も古い使用中オブジェクトから順に古い領域にプロモートする．

この生成順序保存型コピー方式GCは，印付け段階で使用中オブジェクトのアドレスを表にし，コピー先領域におく（A-領域，**図1**参照）．つぎにA-領域を整数値データとみなして整列する．コピー段階では，A-領域を参照して使用中オブジェクトをコピー先領域にコピーする．ポインタ補正段階では，普通のコピー方式と同様の転送先ポインタを利用してポインタ補正を行う．

本稿の方法では，古い領域から新しい領域を指すポインタ（逆ポインタ）の補正のために，古い領域に対しても印付けを行う．この印付けの際，次に述べる

† 電気通信大学電気通信学部情報工学科
Department of Computer Science, Faculty of Electro-Communications, The University of Electro-Communications

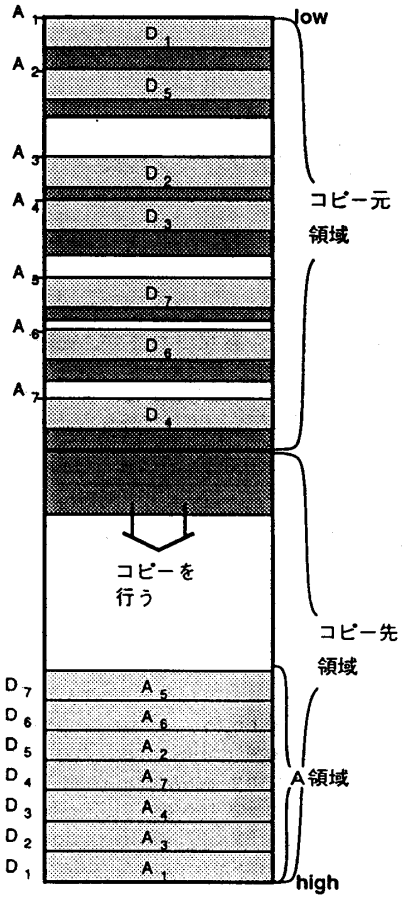


図 1 生成順序保存型コピー方式 GC のメモリ配置
Fig. 1 Memory allocation for the generated-order-preserving copying GC.

"最大ポインタ" を同時に計算する.

最大ポインタとは，新旧の境界をそれよりも大きくしたとき古い領域から新しい領域を指すポインタがなくなるアドレス値である．この最大ポインタが指すオブジェクトとそれよりも古いオブジェクトを古い領域にプロモートすると，GC が終わった時点で逆ポインタが存在しないことが保証される．

コピー方式を使用した世代管理で効率的なものに Appel の方法[1]がある．Appel の方法では，新しい領域で1回コピーされたオブジェクトすべてを古い領域にプロモートする．本方法では，2回か3回の GC を生き残ったオブジェクトと最大ポインタが指すオブジェクトを比較して，より新しいオブジェクトとそれよりも古いオブジェクトまでをプロモートする．

本方法では，新しい領域の任意の二つのオブジェクトのうち，古いものより新しい方のオブジェクトが先にプロモートされることがない．生成順序が保存されない従来の世代管理では，このことは保証されない．

また，Appel の方法では，純計算時にページフォールトなどの割り込みの機構を利用して，逆ポインタを記憶領域とは別領域の表 (assignment table) に登録している．本方法では，そうした純計算時の処理や領域が不要である．しかし，GC 時における計算時間を Appel の方法と比較すると，本方法では古い領域の印付けと生成順序の保存のための整列の手間が余分に必要になる．

2. アルゴリズム

本方法での記憶領域の割り当ては，**図 2** のようになる．アドレスが d_s から d_0 より小さい部分を古い領域，d_0 から d_e より小さい部分を新しい領域とする．新しい領域は，更に二つの半領域に分割される．アドレスが d_0 から d_1 より小さい領域をページ0，d_1 から d_e より小さい領域をページ1とする．使用中オブジェクトは二つの半領域の間で2回か3回コピーされた後，古い領域にプロモーションされる．また，

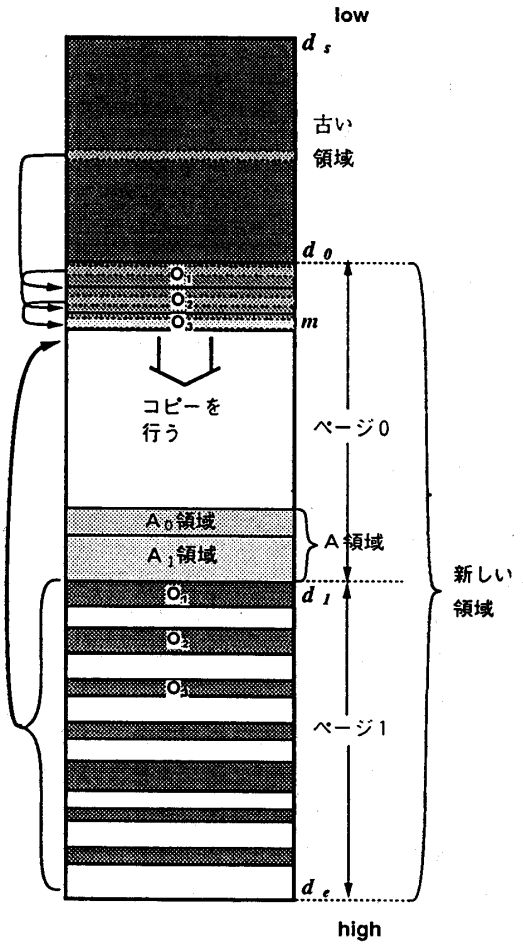


図 2 純計算時に負担のないコピー方式世代管理 GC のメモリ配置
Fig. 2 Memory allocation for the generational GC without the runtime overhead.

変数 m は最大ポインタを計算するために使用される．

本方法では，GC が起動されると，印付け，整列，ポインタ補正，プロモーション，半領域の切替えの各段階を順に実行する．

印付け：言語処理系のレジスタやスタックなどから，すべての使用中オブジェクトのリスト構造をたどって印付けをする．印付けをすると同時に，次のものをコピー先領域の最下位アドレスから順に登録する（A-領域）．

- 新しい領域にある使用中オブジェクトのアドレス
- 逆ポインタの場合はその始点のアドレス

また，ページ1からページ0にコピーを行う場合は逆ポインタの終点の最大値を変数 m の初期値として記録する．

整列：A-領域にある使用中オブジェクトへのポインタの集まりを整数値データとみなして上昇順に整列する．A-領域は d_0 を境に図2のように，逆ポインタのアドレス（A_0-領域）と新しい領域の使用中オブジェクトのアドレス（A_1-領域）に分かれる．

コピー：A_1-領域の指すオブジェクトを，A_1-領域の順序でコピーする．その際，コピー元のオブジェクトの先頭にコピー先オブジェクトへの転送先ポインタを入れる．

ポインタ補正：コピーされたすべてのオブジェクトのポインタを表す部分に対して，転送先ポインタを参照してポインタ補正を行う．A_0-領域の指すオブジェクトについても，ポインタ補正を行う．

プロモーション：コピー段階において，ページ1からページ0にコピーを行った場合は，以下のことを行う．

m の計算：d_0 から，オブジェクト中のポインタを順に調べ，始点が m 以下で終点が m 以上のポインタの終点を新しく m とすることを繰り返す．ただし，調べるオブジェクトのアドレスが m より大きくなるまでとする．

d_0 の更新：2回の GC（ページ0からページ1，ページ1からページ0）または，3回の GC（ページ1からページ0，ページ0からページ1，ページ1からページ0）を生き残ったオブジェクトの次のアドレスで d_0 を置き換える．ただし，これよりも m が大きい場合は，m で指されるオブジェクトの次のアドレスで d_0 を置き換える．

d_1 の更新：d_1 が更新された場合，$d_1=(d_0+d_e)/2$ とする．

半領域の入れ換え：ページ0とページ1の二つの半領域の役割を入れ換える．

A-領域は印付けの段階でその内容が作成される次の整列の段階で，A-領域は A_0-領域と A_1-領域に分かれる．A_0-領域はポインタ補正のために，A_1-領域は新しい領域の生成順序を保存するために，それぞれ使用される．

新しい領域での使用中オブジェクトの占有率が高く，逆ポインタの個数や新しいオブジェクトの個数が多いため A-領域が大きい場合に，コピー段階で，A-領域の一部がコピーするオブジェクトで上書きされてしまう場合がある．その場合には，上書きされる前に，コピー元領域のすでにコピーが終わっている部分で転送先ポインタ以外の部分に，その A-領域の一部を退避する．このように，コピー元領域が利用できるのも，オブジェクトの生成順序が保存されているからである．

古い領域の GC は，古い領域が記憶領域の半分に近付いて来た場合に行われる．それを行うためには，ページ1からページ0にコピーが行われ，d_0 の更新の時に，$d_0=d_e$ とする．次回の GC の時に記憶領域全体の GC が行われる．

3. 考　　察

計算時間を議論するため，使用中オブジェクトの個数を n，使用中オブジェクト全体の大きさを S とする．

世代管理を導入する前の生成順序保存型コピー方式 GC の全計算時間は，整列段階の n 個の整数の整列時間とその他の段階全体の和である．後者は通常のコピー方式 GC と同様に，使用中オブジェクト全体の大きさ S に比例する計算時間で実行できる．この整列は，n 個の整数（例えば 32 ビットの1語）を大きさの順序に並べる単純なものである．そのため，ヒープ整列法やクイック整列法のように高速の整列法を使用すれば，$n\log_2 n$ に比例する計算時間で実行できる．実際的には並べ換える対象のほとんどは，最初からほとんど整列されているのが普通であり，直接挿入法（インサーションソート）やシェルソートのいずれかを使用するとよい．（このことはいくつかの実験により確かめた．）

一方，本方法の全計算時間は，印付け段階における使用中オブジェクトの判別に，使用中オブジェクトの個数 n に比例する計算時間がかかる．しかし，それ以降の各段階では，前回の GC から増えた使用中オブジェクトの分だけ整列やコピーを行えばよい．ここで世代管理により整列すべき整数の個数が少なくなることに注目されたい．したがって，世代管理を行わない生成順序保存型コピー方式 GC より計算時間を減らすことが期待できる．

本方法を Appel の方法と比較すると，以下のようになる．

- 本方法では assignment table を省略でき，Appel の方法で利用されているページフォールトなどの割り込み機構が不要である．しかし，本方法では古い領域の印付けと生成順序保存のために整列の手間が余分に必要になる．
- Appel の方法では，新しい領域で一度コピーされたオブジェクトすべてをプロモートするが，本方法では，2回か3回の GC を生き残ったものか，最大ポインタが指すオブジェクトとそれよりも古いオブジェクトまでをプロモートする．従って，本方法の方が古い領域の増え方は少ないと考えられる．
- 本方法では，新しい領域の任意の二つのオブジェクトのうち，新しい方のオブジェクトが先にプロモートされることがない．生成順序が保存されない従来の世代管理では，これは保証されない．

生成順序保存を利用する場合，GC が起動されたときにコピーされたオブジェクトのポインタを表に記録しておくことにより，さらに複雑な世代別管理に応用できる．例えば，新しい領域の大きさによって，何世代までさかのぼって GC を行うかを変化させることなどが考えられる．

一般に世代管理では，GC 時に古い世代を操作しないことと純計算時の負担をなくすことの二つの条件は両立しないものと思われる．本稿の方法は前者の条件を緩めることによって，後者を実現するものといえる．

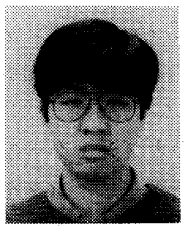
本方法の実際の場での評価は今後の課題とする．

なお，本稿は口頭発表[5]に基づいている．

参考文献

1) Appel, A.: Simple Generational Garbage Collection and Fast Allocation, *Softw. Pract. Exper*, Vol. 19, No. 2, pp. 171-183 (1989).
2) Baker, H.G.: List-Processing in Real Time on a Serial Computer, *Comm. ACM*, Vol. 21, No. 4, pp. 280-294 (1978).
3) Fenichel, R. and Yochelson, J.: A Lisp Garbage-Collector for Virtual-Memory Computer Systems, *Comm. ACM*, Vol. 12, No. 11, pp. 611-612 (1969).
4) 小出 洋，野下浩平：生成順序を保存するコピー方式ガーベジコレクションについて，情報処理学会論文誌，Vol. 34, No. 11, pp. 2395-2400 (1993).
5) 小出 洋，野下浩平：生成順序の保存に基づくコピー方式世代管理法，情報処理学会記号処理研究会報告，SYM 71-1 (1993)；訂正 SYM 72 (1994).
6) 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 (1993).
7) Ungar, D.: Generation Scavenging: a Non-disruptive High Performance Storage Reclamation Algorithm, *SIGPLAN Notices*, Vol. 19, No. 5, pp. 157-167 (1984).

(平成6年3月31日受付)
(平成6年7月14日採録)



小出 洋（学生会員）

1991 年電気通信大学計算機科学科卒業．1993 年同大学大学院情報工学専攻博士前期課程修了．現在，同博士後期課程在学中．記憶管理方式，並列分散処理に関する研究を行っている．

鈴木 貢（学生会員）

1989 年電気通信大学計算機科学科卒業．1991 年同大学院情報工学専攻博士前期課程修了．現在同博士後期課程在学中．記憶管理アルゴリズムに興味を持つ．

野下 浩平（正会員）

1943年生，東京大学工学部計数工学科卒業．現職：電気通信大学情報工学科教授．工学博士（東京工業大学）．専門：アルゴリズム解析，組合せゲームの理論と実験．ACM, EATCS, CSA 等の各会員．