Vol. 35 No. 11　　情　報　処　理　　Nov. 1994

解　説

ごみ集めの基礎と最近の動向

4. 世代別ごみ集め†

中西正和††　田中詠子†††

1. はじめに

リスト構造によるデータ構造は，動的特性に優れメモリ領域の有効利用が可能であるために，人工知能などの複雑なデータの記憶に利用されている．しかし，記憶すべき情報の量には限りがないのにメモリには限りがある．有限のメモリ資源を効率よく，プログラマに負担をかけずに管理することは不可欠である．Lisp などの記号処理言語では，ごみ集め (GC) が必要なときに自動的に起動して，メモリの中を整理し，限られたスペースの中に繰り返し新しい資源を作って再利用するようにしている．

GC は，生物の眠りにたとえられることが多い．人間などの生物は，起きているときは脳の中の状態がどのようになっているかなどおかまいなしに情報を湯水のように流し込む．一定の時間が経つと流し込まれる領域が満杯になって睡魔が襲い，眠っている間にメモリの整理が行われる．目覚めたときにはまた新しく情報を取り込む領域が確保されている．一括型の GC は，生物の眠りの現象とよく似ているというわけである．マーク・スイープ法では，確かに GC 中は処理が中断して眠ったようになり，目覚めたときには眠りによって再び使用可能になったセルのリストが与えられるようになっている．しかしこの単純な方法では，アクティヴなセルが多ければ多いほどマーキングに時間がかかり，メモリ領域が大きくなればなるほど回収の時間がかかる．人間が記憶している情報の量から人間のメモリ領域の大きさを推量すると，人間の GC のアルゴリズムはこのような単純なものではないことが分かる．

マーク・スイープ法の欠点は，上記のように情報と領域の大きさに依存することであった．しかし，情報には「価値」があり，長時間保存すべきものと即時に不要になるものがある．Lisp インタプリタの動作などでは，この双方が共存することが知られている．たとえば，純 Lisp で Lisp の式を評価することを考えよう．関数呼出の式の引数の並びに現れるそれぞれの式は，eval で評価されるが，apply に渡すために evlis によってまとめられて一つのリストとなる．この評価済みの値のリストを形成しているセルは，apply に渡されるとすぐに不要になる．すなわち，これらのセルは情報を一時的に連結するだけのために利用されているだけであり，その寿命は非常に短い．しかし，短寿命だからといって量が少ないわけではない．マーク・スイープ法ではメモリ領域が全部使われるまで寿命の長短に関係なく累積していく．短寿命のセルが多い場合は，不要な情報を長期間メモリに保持することになる．

一方，複写方式の GC では，マーク・スイープ法とは逆の無駄が発生する．すなわち，領域が半分になるため GC の頻度は大きくなるが，回収のフェーズがなく寿命の短い情報が複写されない利点がある反面，寿命の長いデータは毎度複写されなければならない．

2. 世代別 GC の基本

起きている間に取り入れた情報を眠りの間に整理して記憶しなおすのが GC である，と考えると，「整理」には情報の価値を判別する作業が含まれると考えるのが自然である．複写方式の欠点は，寿命の長い情報までも毎回移動させてしまうものであったが，寿命の長短を判別して寿命の長

† Generational Garbage Collection by Masakazu NAKANISHI (Department of Mathematics, Faculty of Science and Technology, Keio University) and Eiko TANAKA (Department of Computer Science, Graduate School of Science and Technology, Keio University).

†† 慶應義塾大学理工学部数理科学科
††† 慶應義塾大学大学院理工学研究科計算機科学専攻

い情報の複写を避けるようにすれば無駄を省くことができる．特に，Lisp などのシステムではある程度生き続けたデータはさらに長期にわたって生き残るという性質があることが分かっている[7]．昔からあった古いデータは複写せず，最近新しく記憶されたデータだけを複写の対象にすればかなりの無駄を省くことができるはずである．この方法が世代別 GC (generational garbage collection)[7),8),14)]である．

2.1 主要な技法

世代別 GC は，寿命の長さを判定し，いくつかの世代 (generation) に分類して異なる領域にわりあてるのが基本的な考え方である (**図-1**)．通常の GC では，最も若い世代だけを複写の対象とする．一度複写されたデータは一度「生き残った」と判定され，何回か生き残ると一つ上の世代とみなされ，上位の領域に移動させるようにする．これを**殿堂入り** (**tenuring**) という (**図-2**)．通常の若い世代だけの GC の繰返しで空き領域が不足したとき，はじめて一つ上の世代を加えて GC を行う．これでも空き領域が不足した場合は，さらに一つ上の世代を加えて行う．

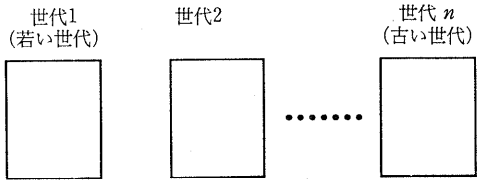


図-1 世代別 GC のメモリ構造

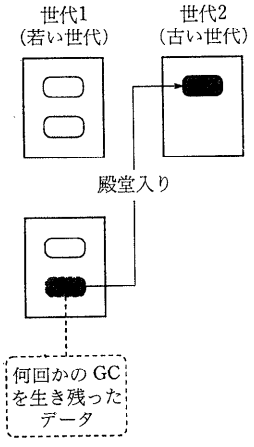


図-2 殿堂入り

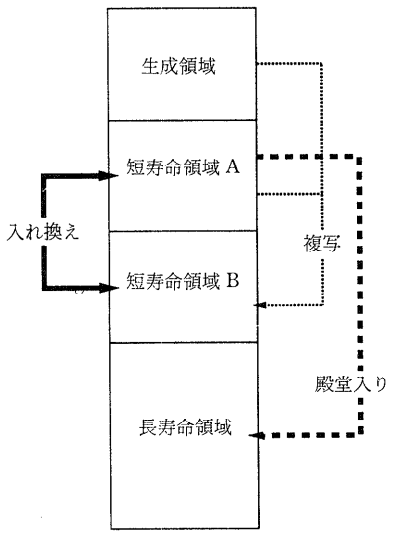


図-3 Generation Scavenging GC のメモリ構造

GC で生き残ったデータは，一つ上の世代へ移行するための**カウント**が増やされる．カウントが一定の数を越えると，上の世代に昇進する．この結果，頻度の多い若い世代の GC の複写の回数を減少させることができる．

世代別 GC の一つである generation scavenging GC[14)] では，メモリ領域を**生成領域**，二つの**短寿命領域**，そして**長寿命領域**に分け，生成領域と短寿命領域を主記憶にとり，長寿命領域を仮想記憶にとることにより，無駄なページフォルトを防ぎ多くのメモリを効率的に使用することができる (**図-3**)．

2.2 古い世代からの参照

世代別 GC は基本的には複写方式であるため，複写のときに，すべての生きているリストを指しているセル，すなわち**ルート**や，セルの中で参照している複写されたセルへのポインタを書き換える作業が当然必要になる．ところが，古い世代のデータが若い世代のセルを参照していたらどうするか，という問題がある．Lisp では，rplaca, rplacd を使用した場合に起こる現象である．このようなことが起こっているならばその若い世代のデータは古いデータと同等の寿命をもっているとみなすべきである．このため，次のような方法が考えられている[7]．

古い世代から若い世代を参照するときは，直接指すのではなく，間接参照のためのテーブルを介する (**図-4**)．このテーブルをルートの一部とみ

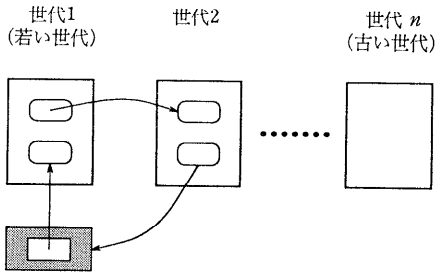


図-4　古い世代から若い世代への間接参照

なして若い世代の GC を行う．この結果，古い世代から指された若い世代は生きているものとみなされて複写の対象となる．この方法は優れているようにみえるが，処理系がポインタを参照するときに直接参照か間接参照かを判定しなければならず，そのオーバヘッドを無視することができない．したがって，汎用機には向いていない．この方法は，専用のハードウェアでオーバヘッド部分を短時間で処理できる Lisp マシンで採用されている[2),4)]．

オーバヘッドを回避できない汎用機では，古い世代から若い世代へは直接参照する方法が考えられている[14)]．この場合，古い世代の中の若い世代を参照している部分の位置を記憶するテーブルを用意する．古い世代のセルの内容を変更するとき，それが若い世代へのポインタに変わるのであれば，そのセルの場所をテーブルに記憶させる(**図-5**)．若い世代の GC が起こったときは，このテーブルに記憶されているセルをルートの一部とみなすようにしてやれば良い．

2.3 世　代　数

世代別 GC の最適な世代数は，その利用目的によって異なる．たとえば，Tektronix の Smalltalk[1)]では，8世代が採用されているが，一般の簡単な Lisp などでは2世代だけでもある程度の効果があると言われている．しかし，当然であるが，同じ Lisp を利用していても，そのアプリケーションによって大きく異なる．基本的な Lisp インタプリタであれば，evlis による短寿命のセルの発生がセル利用の大部分を占めるため，2世代でもかなり大きな効果がある．しかし，セルの消費を抑えた Lisp インタプリタや Lisp コンパイラによってコンパイルされたコードによる利用では3世代以上での効果が検知できる．

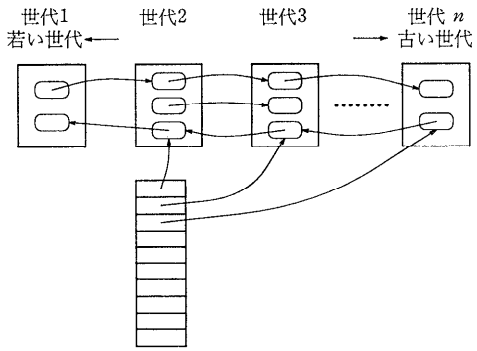


図-5　古い世代からのテーブル参照

2.4 長寿命領域の GC

世代別 GC では通常の GC は若い世代のデータに限定している．しかし，長寿命領域へ殿堂入りしたデータの中にはごみになってしまうものも存在する．時間が経つにつれ，長寿命領域が使い尽くされてしまい処理が続かなくなれば，長寿命領域の GC を行わなければならない．長寿命領域は一般的に短寿命領域に比べて大きな領域が割り当てられる．したがって長寿命領域の GC はめったに起こらないが，一回起こると非常に時間がかかる．しかも，仮想記憶に割り当てられた場合には，ページフォルトが多発するためさらに時間がかかる．長時間の処理の中断は使用者に苦痛を与えるため，システムがアイドル状態になるのを待って，長寿命領域の GC はオフラインの状態で行う[14)]．長寿命のデータを効率良く回収する研究も行われている[5)]．

2.5 殿堂入り問題

殿堂入りとは，短寿命領域中で，複写によってある一定の回数生き残ったデータを，短寿命領域から長寿命領域へ移すことである．何回の生き残りで殿堂入りを起こすかを決めるのが**殿堂入り問題 (tenuring problem)** である[14),15)]．殿堂入りを起こすまでの複写の回数を **advancement threshold** という．この値が小さすぎると，長寿命領域に移された直後にセルが死ぬ可能性がある．長寿命領域の中にごみとなるべきセル（これを**殿堂中のごみ (tenured garbage)** という）を置いておくのは無駄である．

この問題を解決するために GC による中断時間を十分小さくできるようなしきい値を設けて，その値を越えたときに生きているデータの中でカウントの高いものから順に長寿命領域に複写する

方法がある[15]．これによって，人間に耐え得るだけのぎりぎりの中断時間を越えるまで長寿命領域への殿堂入りは起こらないので，その分，殿堂中のごみの出現が抑えられる．この手法は，十分大きな領域を確保することによって殿堂中のごみが抑えられる代わりに，1回の GC による中断時間が大きくなってしまうことを改良するために用いられている．ただし，しきい値を越えるまでは生きているデータは短寿命領域中で繰り返し複写され続ける．

3. 最近の動向

3.1 Opportunistic GC

opportunistic GC (OGC)[16]は，generation scavenging GC に **bucket-brigade** の手法を使用することにより，各データがカウントをもつことなしに advancement threshold を設定できるようにした手法である．OGC では，advancement threshold は1と2の間が良いとされている．それは，次のような理由による．世代数を2に設定した場合，advancement threshold を1に設定すると，生成領域の中で生きているデータはただちに長寿命領域に複写される．しかしこの場合は，領域が GC される直前に生成されたデータは死ぬまでの十分な時間を与えられずに移されてしまう．その結果，長寿命領域に移された寿命の短いデータはすぐにごみとなってしまい，殿堂中のごみの量が増えてしまう．一方 advancement threshold を2に設定すると，前述のような問題は生じない．すなわち，領域が GC される直前に生成されたデータはすぐに長寿命領域に移されるわけではなく，短寿命領域中の別領域に1回移され，次の GC のときに長寿命領域に移されるからである．しかし，この場合は長寿命領域に移されるデータは2度複写されるので，複写のコストがかかってしまう．advancement threshold を2よりも大きくする場合には，advancement threshold を増やすコストに比べて長寿命領域へ移されるデータが減る割合が少ない．したがって，advancement threshold を2よりも大きくする意味はあまりない．よって，1と2の間が良いとされている．

bucket-brigade の手法では，**図-6** に示すように，一つの領域を二つに分け，各領域をさらにいくつかの bucket に分けている．複写が始まると，

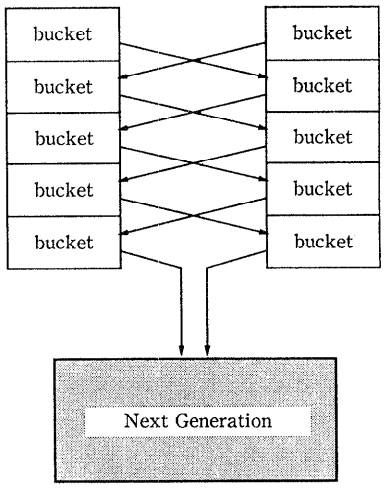


図-6 bucket-brigade

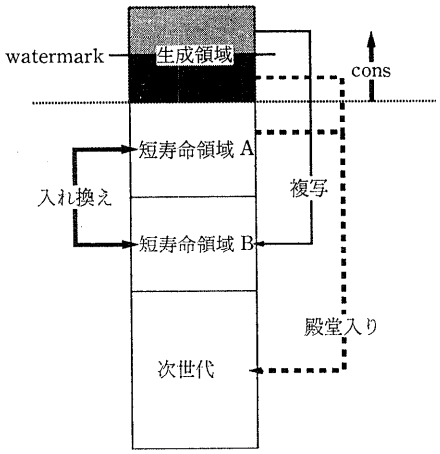


図-7 Opportunistic GC

各 bucket の中の生きたデータはずらされるようにして，もう一つの領域の bucket に移される．bucket の数だけずらされたデータは自動的に次の世代へ移ることができる．したがって，各データがカウントをもつことなしに寿命の管理ができる．

実現方式としては，生成領域に境界 (**watermark**) を設ける (**図-7**)．図では生成領域の下から上に矢印のようにデータが生成されていく．すなわち，下のデータのほうが古いデータである．GC が始まるとコピーが行われるが，watermark より下に位置するデータは，生成されてからかなり時間が経っているので長寿命と判断されて長寿命領域に直接移される．これらのデータの advancement threshold は1になる．watermark より上

に位置するデータは，まだ長寿命と判断するには十分な時間を経過していないので短寿命領域に複写される．これらのデータの advancement threshold は 2 になる．生成領域全体のデータの advancement threshold の平均は 1 と 2 の間になる．この方法では watermark を移動することにより，advancement threshold を 1 から 2 の間に自由に設定することができる．これらは各データにカウントをもたせることなしに，bucket-brigade の手法により実行される．

3.2 Adaptive garbage collection

殿堂入り問題を解決するためには，advancement threshold を適切に設定する必要がある．このとき考慮しなければならないのは，アプリケーションによって適切な advancement threshold は異なるということである．アプリケーションによって，また動かすインタプリタによってセルの消費のペースは変わってくる．セル消費のペースによってセルが長寿命であるための条件が変わるので，advancement threshold の数値のもつ意味もセル消費のペースによって変わってしまう．アプリケーションに応じて，またはセル消費のペースに応じて動的に advancement threshold を決定することによる，効率の良い GC アルゴリズム **adaptive garbage collection** (**AGC**) の研究が行われている[10)~13)]．AGC はセルの消費のペースに応じた GC であり，そのときのセルの消費状態によって advancement threshold を動的に調整して GC を行うことにより，殿堂中のごみを最小限に抑え，長寿命領域を有効に使用することができる．

3.3 長寿命のデータの回収

アプリケーションによっては，大きな構造データが数多くメモリ上に存在する場合がある．このような大きなデータは，ある程度長時間生き残る傾向があるため，何回かの GC を生き残って長寿命領域へ複写される．アプリケーションによってはこのようなデータもやがて不必要となり，長寿命領域に大量の殿堂中のごみが出てきてしまう．このごみは長寿命領域の GC が行われるまで回収されない．この問題を解決するための方法が提案されている[5)]．前述のような大きな構造データのために，短寿命領域や長寿命領域のほかに別の領域 (**keyed area**) を用意し，短寿命領域での GC が終わると殿堂入りの代わりにその領域へ移される．移された構造データには印 (**key object**) が付けられ，それがルートとなる．keyed area に移されたデータの GC は短寿命領域の GC 時に key object を通して行われる．key object がどこからも参照されなくなれば，その key object によって管理されていた部分は回収される．この方法によって，大きな構造データによる大量の殿堂中のごみを減少させることができるが，key object がどこからも参照されなくなった後，構造データの一部が別のデータから参照されている場合の管理などについて考慮する必要がある．

3.4 分散メモリ上の世代別 GC

分散メモリ構成の並列計算機上で記号処理を行うときに生じる問題点を考慮し，効率の良い GC アルゴリズムが開発されている[6)]．このアルゴリズムでは，プロセッサ間のメモリ使用量がばらつくという問題を回避し，各プロセッサがメモリを使い切るまでの時間を均一化することができる．また，プロセッサ間にまたがる大きな構造データが存在するときに，そのデータをたどる処理の直列化が並列性の向上を妨げる原因として考えられる．このような大きなデータの多くが長寿命であることを予想し，世代別 GC を導入することによって大きな構造データをたどることによる弊害を回避することができる．しかし，殿堂入り問題は出現し，並列記号処理言語ではデータの寿命もより不確定になることが予想されるので，考慮する必要がある．

3.5 世代別 GC の並列化

世代別 GC を並列化するという研究も行われている[9)]．on-the-fly GC[3)]は，リスト処理プロセッサと，GC 専用のプロセッサの二つのプロセッサを並行に動かすことによって，GC による中断時間を減らすというものである．世代別 GC の並列化では，リスト処理プロセッサとは別に各世代ごとに GC プロセッサを割り当て，さらに，長寿命領域からの参照のテーブル管理のための別のプロセッサも割り当てて，それらを並行に動かすというものである．マルチ CPU のコンピュータが次々と開発されている今日では，世代別 GC の並列化は，共有メモリのコンピュータ・アーキテクチャをより効果的に使用するための重要なアプローチとなる．

4. まとめ

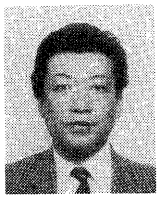
以上のように，世代別 GC は「生成されたほとんどのデータは生成後間もなく死んでしまう」という性質を利用し，GC を生成後間もないデータに対して重点的に行うことにより GC 効率をあげることを意図したものである．特に複写方式の GC に組み込む際には容易に実現することができ，その実行効率もかなり優れたものとなる．

GC の研究の大きなテーマの一つは「GC による中断時間をなくす」という点にあるが，世代別 GC はこの目標を実現するための重要な鍵である．

参考文献

1) Caudill, P. J. and Wirfs-Brock, A.: A Third Generation Smalltalk-80 Implementation, OOPSLA '86 Proceedings, pp. 119-130 (Sep. 1986).
2) Courts, R.: Improving Locality of Reference in a Garbage-Collecting Memory Management System, Comm. ACM, Vol. 31, No. 9, pp. 1128-1138 (Sep. 1988).
3) Dijkstra, E. W., Lamport, L., Martin, A. J., Scholten, C. S. and Steffens, E. F. M.: On-the-Fly Garbage Collection: An Exercise in Cooperation, Comm. ACM, Vol. 21, No. 11, pp. 966-975 (Nov. 1978).
4) Greenblatt, R.: The LISP Machine, MacGraw Hill (1984).
5) Hayes, B.: Using Key Object Opportunism to Collect Old Objects, OOPSLA '91 Proceedings, pp. 33-46 (Oct. 1991).
6) 小池汎平，田中英彦：分散メモリ並列計算機上でのジェネレーションスキャベンジング GC，並列処理シンポジウム JSPP '90, pp. 273-280 (May 1990).
7) 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 (June 1983).
8) Moon, D. A.: Garbage Collection in a Large Lisp System, ACM Symposium on Lisp and Functional Programming, pp. 235-246 (Feb. 1984).
9) Sharma, R. and Soffa, M. L.: Parallel Generational Garbage Collection, OOPSLA '91 Proceedings, pp. 16-32 (Oct. 1991).
10) 高岡詠子，中西正和：セル消費監視プロセスの設置による Generation Scavenging GC，情報処理学会記号処理研究報告，91-SYM-61-5, No. 78, pp. 1-7 (Sep. 1991).
11) 高岡詠子：Adaptive Garbage Collection の提案およびその実現のための研究，Master's thesis，慶應義塾大学 (Feb. 1992).
12) 高岡詠子，中西正和：アダプティヴ・ガーベッジ・コレクションの実現のための実験報告，情報処理学会記号処理研究報告，93-SYM-67-1, No. 8, pp. 1-8 (Jan. 1993).
13) 田中詠子，田中良夫，中西正和：Adaptive Garbage Collection の提案および実験，電子情報通信学会論文誌 (D-I), J77-D-I, 9, pp. 611-618 (Sep. 1994).
14) Ungar, D.: Generation Scavenging: A Non-disruptive High Performance Storage Reclamation Algorithm, ACM SIGPLAN Notices, Vol. 19, No. 5, pp. 157-167 (May 1984).
15) Ungar, D. and Jackson, F.: Tenuring Policies for Generation-Based Storage Reclamation, OOPSLA '88 Proceedings, pp. 1-17 (Sep. 1988).
16) Wilson, P. R. and Moher, T. G.: Design of the Opportunistic Garbage Collector, OOPSLA '89 Proceedings, pp. 23-35 (Oct. 1989).

(平成6年2月15日受付)



中西 正和（正会員）

1966年慶應義塾大学工学部卒業．1969年慶應義塾大学工学部助手．1989年慶應義塾大学理工学部教授．工学博士．1967年，日本初の実用 Lisp 処理系を作成．以後，記号処理言語，人工知能用言語等の研究に従事．1975年，プログラムの性質の自動証明系，1982年 Lisp マシン SYNAPSE の開発など．電子情報通信学会会員．本会プログラミングシンポジウム委員会幹事長．

田中 詠子（正会員）

1968年生．1990年慶應義塾大学理工学部数理科学科卒業．1992年同大学院理工学研究科計算機科学専攻修士課程修了．現在，同博士課程在学中．LISP に興味を持ち，Garbage Collection の研究を行っている．電子情報通信学会会員．