オブジェクトの世代を考慮に入れたインクリメンタルなごみ集め処理

小池龍信† 岩井輝男† 中西正和††

ごみ集め処理（GC）を必要とする言語がいくつか存在し，Lisp言語などのような関数型言語処理系にはGCが不可欠である．これらの処理系は，実時間性に向いてないとされてきた．理由としては，GCを行なっている時に処理が一時中断することがあげられる．この中断時間をできるだけ減らすための研究がなされている．このようなGCを実時間GCと呼び，その中の一つにTreadmill GCがある．Treadmill GCは複写式GCを改良したものである．このGCでは全てのオブジェクトを双方向環状リストで連結し，GCは複写によるオブジェクトの移動ではなく，双方向ポインタのつけかえ（relink）で実現する．これにより複写式GCの利点を保持し，さらにオブジェクトの移動を行なわないため複写式GCで問題となるreadバリアが解消されている．Treadmill GCの時間的コストは生きているオブジェクトの数に比例する．よって生きているオブジェクトが多い場合には実時間性が乏しい．この問題点の解決方法として世代別GCの考えを取り入れた．世代別GCの考え方は，ある程度生き続けたオブジェクトは半永久に生き残り，また生成して間もないオブジェクトの殆んどは寿命が短いというものである．本稿では，世代別GCの考えを取り入れたTreadmill GC（Opportunistic Treadmill GC）の提案及びその実現方法，さらにいくつかのベンチマークアプリケーションを実行し，その実験結果から1回のGC時間，総実行時間を削減したことについて報告を行なう．

Incremental Garbage Collection Considering the Objects' Lifetime

TATSUNOBU KOIKE,† TERUO IWAI† and MASAKAZU NAKANISHI††

Real-time Garbage Collection is a study that makes pause time of GC as short as possible. Treadmill GC, one of the real-time GC algorithms, is improved Copying GC. In this scheme all objects are linked by cyclic doubly-linked lists (treadmill). Objects are not removed by copying, and objects' pointer of treadmill are relinked. This means that advantages of Copying GC are preserved, solving read barrier problem at the same time. We propose the "Opportunistic Treadmill GC", which is a Garbage Collection technique that the collector traces only short-life objects, setting long-life objects out of collector's view. There is a strong evidence that the overwhelming majority of objects die very young, although a small proportion may live for a long time. In Treadmill GC, pause time of GC is mostly the time for relinking alive objects. Especially in the original Treadmill GC, collector has to relink all alive object. However in Opportunistic Treadmill GC, collector only has to relink short-life objects of all alive objects. Hence we can make a pause time of GC shorter and improve effective real-time GC. Then we implemented the original Treadmill GC and Opportunistic Treadmill GC on an incremental garbage collector of the Lisp1.5 based system, and showed how efficient it is by a few experiments. In comparison with the original Treadmill GC, we could decrease average time of one GC execution as well as total execution time. We refined incremental GC so that the real-time systems with our Opportunistic Treadmill GC will be more useful.

1. はじめに

Lispをはじめとする関数型言語処理系やオブジェクト指向言語処理系では，動的に確保されたメモリ領域を再利用するために回収する，ごみ集め処理（以下GC）が必要となる．一般的にGCは，使用中のメモリ領域を誤って回収することを避けるためにアプリケーションの処理を一旦停止させて行なう．ところがインタラクティブやリアルタイムなアプリケーションには，GCによる処理の中断は致命的である．このGCによる処理の中断時間をなるべく短くすることを目的としたGCが実時間GCであり，これまでの研究でさま

† 慶應義塾大学大学院 理工学研究科 計算機科学専攻
Department of Computer Science, Graduate School of Science and Technology, Keio University
†† 慶應義塾大学 理工学部 情報工学科
Department of Information and Computer Science, Faculty of Science and Technology, Keio University

ざまな手法が提案されてきた．

メモリ領域の複写をとることによってGCを行なう複写式GCを実時間化した場合，アプリケーションの実行中に参照したメモリ領域が既に複写されたかどうかを判断しなければならず（readバリア），その判断によって著しく処理効率が低下してしまう．そこで実時間複写式GCを改良したTreadmill GC[1]は処理系内部にGC専用の双方向環状リストを持つ．その双方向環状リストの付け換えを行ない，そのリストの中の場所によって既に再利用可能かどうかを意味的に分類し，メモリ領域自体の複写は行なわない．これにより実時間複写式GCの欠点であるreadバリアを解消しつつ，複写式GCの利点を効果的に得ることができる．

複写式GCにおいては，使用中のメモリ領域を全て複写しなければならない．複写したもののほとんどは，過去のGCにおいて複写を経験したものであることがわかっている．このことからある程度の回数のGCを経験したものは，半永久的に回収することができず，生成されてから間もないものほど回収されやすい，ということがいえる．これが世代の概念であり，半永久的に回収できないと判断されたものをGCの対象外とすることによって，メモリ領域の複写回数を軽減することができる[2]．このように複写式GCに世代の概念をとりいれ，世代によって別々の管理を行なう手法が世代別GCの基本的な考えである．

世代別GCも数多くの研究がされており，そのほとんどが複写式GCを基本としたものである．Treadmill GCも上述したように複写式GCを改良したものである．Treadmill GCにさらに世代の概念をとりいれることによって，GCによる処理の中断時間を短くすることが本研究の目的である．本稿では，世代別GCの手法をTreadmill GC上で実現する**Opportunistic Treadmill GC**の手法の提案を行なう．また，実時間GCの目的である1回のGCによる処理の中断時間を短くする，という点について，通常のTreadmill GCに比べ改良できたことを実験的評価によって示し，その有効性について検証する．

2. 実時間GC

実時間GCではアプリケーションの処理に対し，

- メモリ領域の枯渇による停止を防ぐ
- 毎回のGCによる中断時間を短くする

ことを目的とする．実時間GCは，アプリケーションの処理を行なうmutatorと，GCを行なうcollectorを実際に並列に動作させる並列GCと，mutatorの処理中にcollectorの処理を時間的に分散させ，疑似的にmutatorが停止していないように見せるインクリメンタルGCの2通りの方法で実現される．

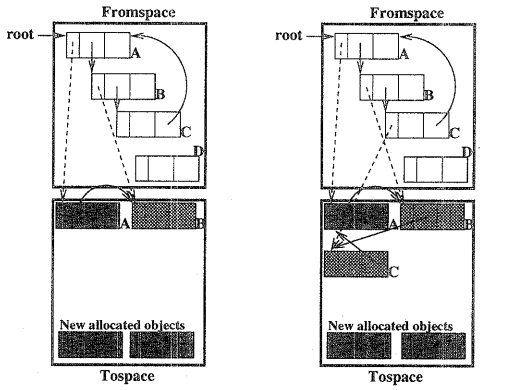


図1 実時間複写式GC
Fig. 1 Incremental copying GC.

実時間GCでは，mutatorとcollectorの間でオブジェクトの参照関係の整合性を保つために，mutatorはcollectorに協調し同期をとらなければならない．

2.1 writeバリア型GC

mutatorがオブジェクト間のポインタを書き換える際に同期をとるもので，大きく2種類に分類される[3]．

- **Snapshot at beginning**[4]
  GC開始時に生きていたものを保持する方針．
- **Incremental update**
  GC終了時に生きているものを保持する方針．

2.2 readバリア型GC

mutatorがオブジェクトを読み込む際に同期をとるもので，H. G. Bakerによって提案された，複写式GCを実時間化した**Incremental copying GC**[5][1]が最も知られている．mutatorはオブジェクトを読み込む際にそれがFrom空間にあるのか，To空間にあるのかを逐一調べ，もしFrom空間ならばそれをTo空間に複写する．また新しいオブジェクトはTo空間に割り当てる．(図1)．

オブジェクトの読み込みの頻度は，ポインタの書き換えの頻度よりもはるかに多い．例えばLispでは，carやcdrなどの頻繁に使用される関数が呼び出されるたびにreadバリアを必要とする．複写の際の時間的負荷はポインタの走査に比べてはるかに大きく，またcollectorの作業時間は生きているオブジェクトが既にTo空間にあるかどうかに依存している．このためIncremental copying GCによるコストは大きく，さらに予想できないことから，厳密な実時間性を要求するアプリケーションには適していない[1][6][3][7]．

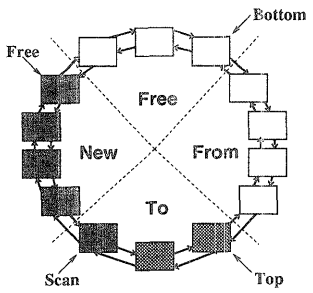


図 2 Treadmill GC
Fig. 2 Treadmill GC.

2.3 Treadmill GC

Baker によって提案された Treadmill GC[1] は，Incremental copying GC を改良したもので，図 2 のように全てのオブジェクトを双方向環状リスト（treadmill）で連結し，GC は複写による移動ではなく双方向ポインタの付け換え（relink）により行なう手法である．これにより複写式 GC による利点を保持しつつ，オブジェクトが移動することがないため，Incremental copying GC の欠点である read バリアを解消している [1)6)3)7)]．

Treadmill GC では，Free ポインタと Bottom ポインタ及び，Scan ポインタと Top ポインタ（図 2 参照）が出会った時点で，*flipping*（From 空間と To 空間の入れ換え），*recoloring*（色の塗り換え，black →white），そして *root insertion* が行なわれ GC が終了する．

3. 世代別 GC

3.1 世代別 GC の基本的な手法

オブジェクトには，ある程度生き続けたものは半永久的に生き残るという性質があること，また生成されたオブジェクトのほとんどは寿命が短く，生成されてからすぐにごみとなってしまうことが知られている．そこでオブジェクトをその寿命に応じていくつかの世代に分け，寿命の短い世代の領域のみを頻繁に複写式 GC を行なう手法が基本的な世代別 GC である．複写式 GC の時間的負荷は生きているオブジェクトの量に比例するため，寿命の短い世代の GC は短時間で終了する．短寿命領域のオブジェクトは，GC を経験した回数がある一定の回数（**Advancement Threshold**）を越えた場合には長寿命領域に複写される．この複写を殿堂入り（tenuring）という．オブジェクトが tenuring される率は低いため，長寿命領域を GC する頻度は極めて低くなり，GC による中断時間を削減する効率の良い手法である．

3.2 殿堂入り問題と Advancement Threshold

世代別 GC では，Advancement Threshold の値をいくつに設定するかが GC の効率に大きな影響を与える．この値が大き過ぎると，長寿命とみなしてもよいはずのオブジェクトが，短寿命領域間で繰り返し複写されなかなか GC の対象外となりえない．また小さ過ぎると，実際には短寿命であるにも関わらず tenuring され，長寿命領域内でごみ（tenured garbage）となってしまい長寿命領域が無駄に消費されてしまう．長寿命領域が全て使用されると長寿命領域をも含めた GC を行なう必要がある．長寿命領域では生きているオブジェクトが多く，この GC には長い時間がかかってしまうため，長寿命領域の GC はなるべく行なわない方がよい．よって，長寿命領域の前の世代においてオブジェクトの大多数はごみとして回収し，長寿命領域でごみとなることを避ける必要がある．

Wilson らは Advancement Threshold の値は 1 から 2 の間が良いとしている [2)]．それは，

- Advancement Threshold が 1 の時，
  GC の直前に生成されたオブジェクトが tenured garbage になりやすい．
- Advancement Threshold が 2 以上の時，
  2 回及び 3 回以上の GC 後に生き残るオブジェクトに大差がない．

という実験結果に基づく主張である．

4. Opportunistic Treadmill GC

複写式 GC を基本とした Treadmill GC でも，オブジェクトの世代毎に別々の管理を行なう手法が有効である [8)]，と考えられる．そこで本稿ではオブジェクトの世代を考慮に入れ，Advancement Threshold を 1 から 2 の間とするインクリメンタル Treadmill GC（Opportunistic Treadmill GC）の手法を提案する．

4.1 Opportunistic Treadmill GC の手法

インクリメンタル GC では，ある一定量のオブジェクトを生成する毎に GC を起動し，その生成量に比例した量のオブジェクトを走査することを繰り返す．そこで本稿では，このように起動した GC を **1 回の GC** と呼び，*root insertion* から次の *root insertion* までを **1 回の GC サイクル**と呼ぶ．Advancement Threshold は，何回の GC サイクルを経験したかを表すものとなる．以下で述べる手法に従うことにより，生きているオブジェクトが 1 回の GC サイクル中で relink されるのは 1 回限りであるので，Advancement Threshold の本来の意味は満たしていることになる．

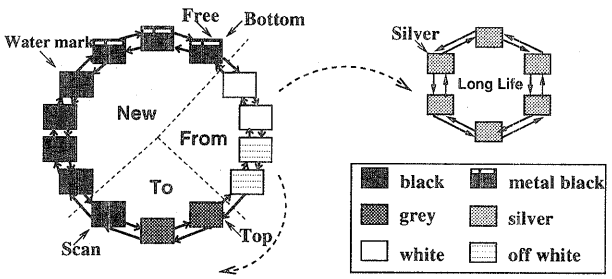


図 3 Opportunistic Treadmill GC
Fig. 3 Opportunistic Treadmill GC

marking color

オブジェクトの色分けは，E. W. Dijkstra のトリカラーマーキング（black, grey, white）の 3 色に加え，以下の色を追加する．

- **silver** ··· 長寿命オブジェクト．
- **metal black** ··· 1 回の GC サイクル終了直前に生成されたオブジェクト．
- **off white** ··· collector に走査されておらず，生成時に metal black であったオブジェクト．

silver，metal black で色づけされたオブジェクトは，black 同様，既に走査済みであるものとし，collector はそのオブジェクトの参照先を走査することはない．

長寿命 treadmill

長寿命オブジェクトは通常の treadmill からは取り除き，長寿命オブジェクト用の treadmill（図 3 参照）に属するものとする．これにより，collector は長寿命オブジェクトを走査の対象外とすることになる．

オブジェクトの生成

watermark（図 3）以前に生成されたオブジェクトは，Treadmill GC 同様に black で割り当てる．しかし watermark 以降は GC サイクルの終了直前であると判断し，watermark 以前に生成されたオブジェクトに比べ生成されてから十分な時間が経過しないまま，次回の GC サイクルを迎えてしまう．従って次回の GC サイクル中で生き残っても長寿命とはいえない．このようなオブジェクトはあらかじめ metal black で割り当てる．

flipping とリカラーリング

1 回の GC サイクルが終了し 4 つのポインタ（Free, Scan, Bottom, Top）の位置を置き換える点については，通常の Treadmill GC と同様である．

オブジェクトのリカラーリングフェーズでは，black であったものは white に色を変えるだけでなく，metal black であったものは off white に色を変える．

生きているオブジェクトの relink

collector は From 空間の生きているオブジェクトを relink して，その位置を置き換える．Scan ポインタの指しているオブジェクトの参照先が，

- white のオブジェクトであった場合
  長寿命 treadmill へ relink して，その参照先を全て走査した後 silver に色づけする．
- off white のオブジェクトであった場合
  To 空間へ relink して，その参照先を全て走査した後 black に色づけする．

基本的に以上の手法により，black で割り当てられたオブジェクトが 1 回の GC を生き残るとすると，black → white → silver という順にリカラーリングされ，長寿命オブジェクトとなる．このようなオブジェクトの Advancement Threshold は 1 である．また，metal black で割り当てられたオブジェクトが 2 回の GC を生き残るとすると，metal black → off white → grey → black → white → silver という順にリカラーリングされ，長寿命オブジェクトとなる．このようなオブジェクトの Advancement Threshold は 2 である．

mutator と collector の同期

Treadmill GC は read バリアを解消した手法で，比較的 mutator の負荷が軽い write バリアで実装することができる．Opportunistic Treadmill GC では，回収効率よりも実時間性を最重要目的とすることから，Snapshot at beginning アルゴリズムを採用する．

4.2 Opportunistic Treadmill GC の正当性

Snapshot at beginning アルゴリズムは，GC が開始された時点で生きているオブジェクトは，GC が終了するまでたとえごみになっても生きているとみなし，次回の GC で回収する，という考え方に基づいて正当性を保証している．また silver のオブジェクトからのみに参照されているオブジェクトは，Ungar が提案した *Remember set* によって保持し，ルート集合の一部とみなす．従って，Opportunistic Treadmill GC の正当性を示すには，tenured garbage を回収できることを示せばよい．

tenured garbage は長寿命 treadmill 内に存在する．

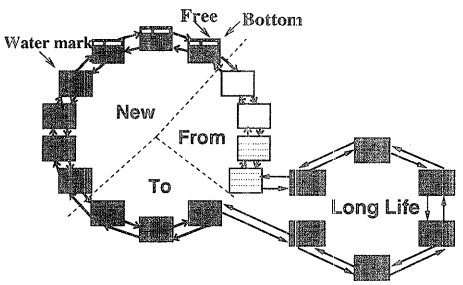


図4 Long Life Insertion
Fig. 4 Long Life Insertion.

また長寿命treadmillは，通常のtreadmillと同様に双方向環状リストであるため，図4のようにTo空間へ挿入（**Long Life Insertion**）しsilverからblackとすれば，次回のGCサイクルでtenured garbageは必ず回収され，かつ生きていたオブジェクトは再び長寿命treadmillを構成できる．

以上のことから，Opportunistic Treadmill GCは正当性を満たしたGCであるといえる．

4.3 Minimum safe relinking rate

Treadmill GCにおける回収作業は，生きているオブジェクトのrelinkである．よってオブジェクトの生成量とrelinkすべき生きているオブジェクトの量との比率を，**Minimum safe relinking rate**と呼び，以下の式で与えられる．

m個のオブジェクトを生成した時，1個のオブジェクトをrelinkするものとする．また，1回のGCサイクルで長寿命を除いて生き残るオブジェクトの個数は，常にNとすると，1回のGCサイクル中でNm個のオブジェクトが生成される．新たに生成されたオブジェクトの中のごみオブジェクトは，次回のGCサイクルまでに回収されることはなく，Nm個のオブジェクトが次回のGCサイクル中に生成される．ここで，N個のオブジェクトが全てoff whiteのオブジェクトであり，*Young*を通常のtreadmillに属しているオブジェクトの個数とすると，

$$(N+Nm)+Nm = N(1+2m) \leq Young$$
$$m \leq \frac{Young - N}{2N}$$

となり，M個のオブジェクトが生成されたら1回のGCを起動するとすると，その時のGCでは，

$$\frac{M}{m} \geq \frac{M}{\frac{Young-N}{2N}} = \frac{2MN}{Young - N}$$

の個数の生きているオブジェクトをrelinkすればよい．

Treadmill GCにおいては，生きているオブジェクトを全てrelinkしなければならない．その個数をN'とし，オブジェクトの総個数を$MAXCELLS$とすると1回のGCでは，

$$\frac{M}{m} \geq \frac{M}{\frac{MAXCELLS-N'}{2N'}} = \frac{2MN'}{MAXCELLS - N'}$$

の個数の生きているオブジェクトをrelinkすることになる．

Opportunistic Treadmill GCで1回のGCサイクルで長寿命を除いて生き残るオブジェクトの個数Nと，Treadmill GCで1回のGCサイクルで生き残るオブジェクトの個数N'とでは明らかに，$N \ll N'$の関係がある．よって，

$$\frac{2MN}{Young - N} \ll \frac{2MN'}{MAXCELLS - N'} \qquad (1)$$

であるといえる．このことからOpportunistic Treadmill GCでは，Treadmill GCに比べて，1回のGCでrelinkすべきオブジェクトの個数が少なく，1回のGC時間を削減することができると考えられる．

GCが進むに伴い，*Young*の値は次第に減少していく．よって式（1）の大小関係が逆転することがありうる．そのような場合になった時点でLong Life Insertionを行なえば良く，この時のGCサイクルのみに限っては，Treadmill GCと同様のMinimum safe relinking rateとなる．

5. 実験結果及び考察

5.1 実験方法

本研究の実験では，Lisp1.5ベースの処理系に，Opportunistic Treadmill GC（OPTMGC）とTreadmill GC（TMGC）を実装し，各種ベンチマークアプリケーションのインタプリタによる実行で比較を行なった．測定項目は以下の通りである．

（1） **1回のGC時間** ... 1回のGC時間の平均値
（2） **総実行時間** ... アプリケーションの総実行時間
（3） **長寿命を含めたGCの頻度と負荷の割合**

ベンチマークアプリケーションには次のものを利用した．

- **boyer**
  ある規則に従って項書き換えを行ない，定理の自動証明を行なうアプリケーションである．多くのセルを消費すると共に，副作用によるポインタの書き換えが頻繁に行なわれる．世代間ポインタが数多く出現する．
- **BIT (bit '(a b c d e f g h i j k l))**
  リストを引数として与えると，リストの各要素をこの順で葉に持つ全ての二進木リストを返すアプリケーションである．生成されたセルの多くが長い間生き続け，最終的に返される二進木リストに使われるセルの数は非

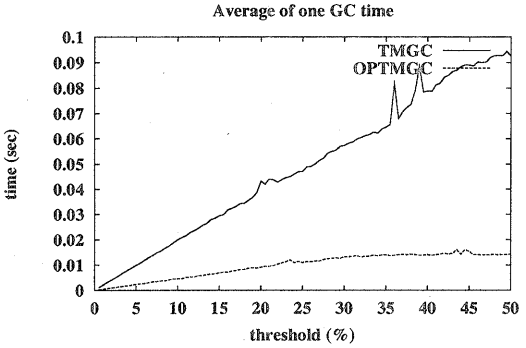


図 5 (boyer) の 1 回の GC 時間
Fig. 5 A GC time of (boyer).

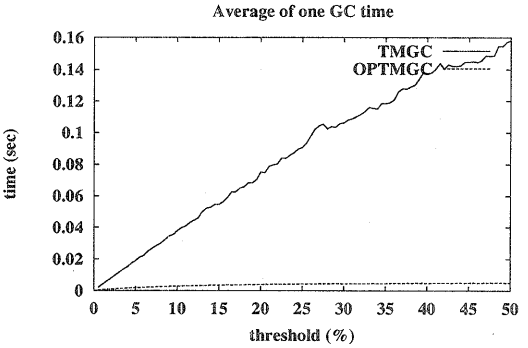


図 6 (bit '(a b c d e f g h i j k l)) の 1 回の GC 時間
Fig. 6 A GC time of (bit '(a b c d e f g h i j k l)).

常に多い.

- **NQUEEN (nqueen 11)**
  探索問題を扱うアプリケーションで, 引数に与える数値によって異なるが多くのセルを消費し, ごみの出る量も多い.

5.2 結果及び考察

4.3 節で述べた Minimum safe relinking rate に従ってGC を行なう. OPTMGC と TMGC について, GC が起動されるまでのセルの生成量を, メモリ領域に対して 1%～50%まで 0.5%ずつ変化させ, それぞれの場合において上述した項目について測定した結果を示す.

5.2.1 1 回の GC 時間の比較

図 5, 図 6 及び図 7 は, 横軸がメモリ領域に対する, GC が起動されるまでのセルの生成量 (M) の割合を表し, 縦軸は 1 回の GC 時間の平均値を表している.

OPTMGC, TMGC 共に GC の処理は Minimum safe relinking rate に従って行なっている. 4.3 節で述べたように, OPTMGC の方が TMGC に比べ 1 回の GC で relink すべきセルの数が極端に少ないことは理論的にも明らかであり, よって図 5 から図 7 に示す通り OPTMGC による 1 回の GC 時間は大幅に削減されている.

TMGC では, GC が起動されるまでに生成されるセルの量が増加すると, 1 回の GC 時間も線形的に増加する. それに対して OPTMGC では, GC が起動されるまでに生成されるセルの量がメモリ領域に対する割合が 30%を越えたあたりからはほぼ横這いとなる. このことから, GC が起動されるまでに生成されるセルの量を変化させて実行しても, OPTMGC では GC 時間に影響しない, といえる. その理由としては, TMGC では GC サイクルを通して relink すべき生きているセルの量が多い場合, GC が起動されるまでに生成されるセルの量が増加すると, 1 回の GC で relink すべきセルの量も大きく増加するが, OPTMGC では relink すべきセルの量が極端に少ないため, GC が起動されるまでに生成されるセルの量が増加しても, 1 回の GC で relink すべきセルの量はわずかしか増加しないためである.

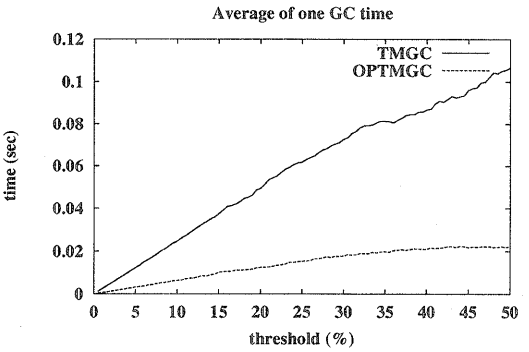


図 7 (nqueen 11) の 1 回の GC 時間
Fig. 7 A GC time of (nqueen 11).

5.2.2 総実行時間の比較

図 8 は, 横軸がメモリ領域に対する, GC が起動されるまでのセルの生成量 (M) の割合を表し, 縦軸は TMGC による総実行時間を 1 とした時の OPTMGC による総実行時間である.

図 8 から, OPTMGC は TMGC に比べ 1 回の GC 時間の削減のみならず, アプリケーションの実行時間をも短縮することができている. また, GC が起動されるまでのセルの生成量を, TMGC と OPTMGC とでそれぞれ別にしても, どんな場合においても OPTMGC によって GC を行なう方が効率が良いこともわかる.

5.2.3 長寿命を含めた GC の頻度と負荷の割合

OPTMGC では 4.3 節で述べたように, TMGC との Minimum safe relinking rate の大小関係が逆転し

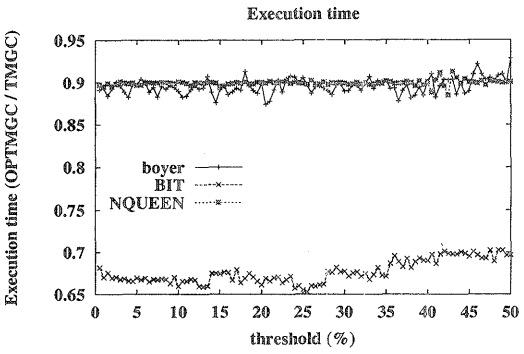


図 8 TMGC に対する OPTMGC の総実行時間の割合
Fig. 8 Execution time rate of OPTMGC to TMGC.

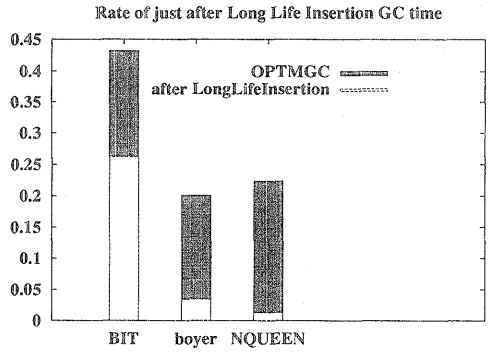


図 9 GC 時間の割合
Fig. 9 Rate of GC time.

表 1 Long Life Insertion 回数
Table 1 Number of times Long Life Insertion.

アプリケーション	回数
boyer	0.935
BIT	1.968
NQUEEN	0.161

た時点で Long Life Insertion を行なう．表 1 はそれぞれのベンチマークアプリケーションの 1 回の実行において，Long Life Insertion を行なった回数を表している．

OPTMGC では，Long Life Insertion によって長寿命のセルは通常の treadmill の To 空間に挿入される．よって Long Life Insertion を行なった直後の GC サイクルで多くのセルを relink しなければならない．図 9 は，Long Life Insertion を行なった直後の GC サイクルにおける GC 時間が，OPTMGC の GC 時間全体に対してどの程度の割合を占めているかを表している．またこの図は TMGC の GC 時間を 1 としたものである．

6. 結論及び今後の展望

6.1 結　　論

複写式 GC をベースとした Treadmill GC は，実時間 GC において read バリアの問題を解消し複写式 GC の利点を効果的に発揮する手法である．しかしオブジェクトの生存率が高い場合には効率は低下する．そこで本稿では，実時間 GC においてもオブジェクトの世代の概念を効果的に反映させるために，双方向環状リスト内の位置を意味的に世代分割して世代ごとに別々の管理を行なう手法である，Opportunistic Treadmill GC を提案した．

Opportunistic Treadmill GC について以下のことが結論として導くことができる．

- 複写式 GC をベースとした Treadmill GC は，オブジェクトの生存率が高い場合に効率は低下する．しかし Opportunistic Treadmill GC では，長寿命と判断したオブジェクトを GC の対象外とすることによって，生き残るオブジェクトのうち，まだ生成されてから間もないものだけを GC の対象として処理すれば良い．これによって GC 時間を削減することが可能となった．
- 生き残ったオブジェクトを長寿命とみなすための条件を，そのオブジェクトが生成された時期が GC の直前直後かで区別し，Advancement Threshold の値を平均的に 1 から 2 の間に設定することにより効率の良い世代昇格方針を実現した．
- 1 回の GC は，Minimum safe relinking rate に従って行なうため，1 回の GC で行なう作業量を制限することにより，GC による中断時間を平均的に時間分散させかつその中断時間も極端に短くした．
- GC を起動するまでに生成するオブジェクトの量が，どのような場合においても Treadmill GC と比較して効率良く実行できる．
- 長寿命を含めた GC を行なうには，Long Life Insertion をすることによって解決している．Long Life Insertion 直後は多くのオブジェクトを GC の対象としなければならなくなるが，その場合は最悪でも Treadmill GC と同等である，という制限ができる．

以上のことから Opportunistic Treadmill GC は有効な手法であると考えられ，実時間システムへの応用が期待できる．

6.2 今後の展望

本稿では，Opportunistic Treadmill GC がインク

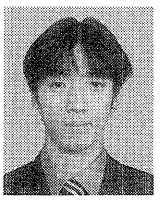
リメンタルGCとしてより有効な特長をもつことを示したが，以下のことを課題として解決することによってより効果的な並列GCへの適用が期待できる．

- 1回のGC時間の上限
  並列GCでは，1回のGC時間を最悪の場合でも一定短時間以内に抑えることを保証しなければならない．Opportunistic Treadmill GCでは，Snapshot at beginningアルゴリズムを採用しているため，GCサイクル開始時にroot集合が直接指すオブジェクトをrelinkするといったroot insertionに費やす時間は，root集合の大きさに依存し一定短時間以内を保証できない．また，Long Life Insertion後のGCサイクルは長寿命オブジェクトを含めたGCとなり，Treadmill GCと同等の手法となることからここでも1回のGC時間を一定短時間以内を保証できない．
- 回収効率の向上
  またOpportunistic Treadmill GCは非常に保守的な手法である．Snapshot at beginningアルゴリズムによるfloating garbageや，長寿命と判断したtenured garbageを即時回収することができない．並列GCでは回収効率の向上をはかり，安定したオブジェクトの提供を保証しなければならない．

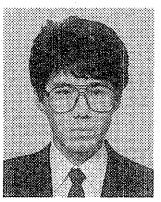
参考文献

1) Baker, H. G.: The Treadmill: Real-Time Garbage Collection Without Motion Sickness, *ACM SIGPLAN Notices*, Vol. 27, No. 3 (1992).
2) Wilson, P. R. and Moher, T. G.: Design of the Opportunistic Garbage Collector, *ACM SIGPLAN Notices*, Vol.24, No.10, pp.23–35 (1989).
3) Wilson, P. R.: Uniprocessor Garbage Collection Techniques, *Proceedings of International Workshop on Memory Management* (Bekkers, Y. and Cohen, J.(eds.)), Lecture Notes in Computer Science, Vol. 637, University of Texas, USA, Springer-Verlag (1992).
4) Yuasa, T.: Real-Time Garbage Collection on General-Purpose Machines, *Journal of Software and Systems*, Vol. 11, No. 3, pp. 181–198 (1990).
5) Baker, H. G.: List Processing in Real-Time on a Serial Computer, *Communications of the ACM*, Vol. 21, No. 4, pp. 280–94 (1978). Also AI Laboratory Working Paper 139, 1977.
6) Jones, R. and Lins, R.: *Garbage Collection: Algorithms for Automatic Dynamic Memory Management*, Wiley (1996). With a chapter on Distributed Garbage Collection by R. Lins.
7) Wilson, P.R. and Johnstone, M.S.: Truly Real-Time Non-Copying Garbage Collection, *OOPSLA/ECOOP '93 Workshop on Garbage Collection in Object-Oriented Systems* (Moss, E., Wilson, P. R. and Zorn, B.(eds.)) (1993).
8) Koike, T., Iwai, T. and Nakanishi, M.: Incremental Garbage Collection Considering the Objects' life time, *Proceedings of the Seventeenth IASTED International Conference on Applied Informatics* (1999). To appear.

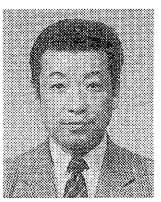
(平成10年12月25日受付)
(平成11年 2 月15日採録)



小池 龍信（学生会員）
平成9年慶應義塾大学理工学部卒業．平成11年同大学大学院理工学研究科修士課程修了．Lisp処理系の研究に興味を持つ．



岩井 輝男（正会員）
平成8年慶應義塾大学大学院理工学研究科博士課程単位取得退学．同年より同大学研究生．Lispに興味をもち，並列Lisp，並列処理の研究を行なう．



中西 正和（正会員）
昭和41年慶應義塾大学工学部卒業．昭和44年同大学工学部助手．平成1年同大学理工学部教授．工学博士．昭和42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．昭和57年Lispマシン SYNAPSEの開発など．