6U－7

# オブジェクトの世代を考慮にいれた Treadmill GC

小池 龍信　中西 正和

慶應義塾大学大学院 理工学研究科 計算機科学専攻

## 1. はじめに

リスト構造の処理を得意とするLispには，「ごみ」と呼ばれる一度使用し不要になったメモリ領域を，自動的に検出して再利用できるようにするgarbage collection (以下GC) 機構が備わっている．そこで本稿では，実時間GCとして考案された手法のひとつである，**Treadmill GC**[1] にオブジェクトの世代の概念を取り入れる手法を提案し，通常のTreadmill GCに比べGC時間の削減など，その有効性の検証を行ないたい．

## 2. Treadmill GC

H.G.Bakerは Incremental Copying GC[1][2][4] を改良し，図1のように全てのobjectを双方向環状リストで連結し，GCは複写による移動ではなくポインタの付け換えにより行なう手法をとるTreadmill GC[1]を考案した．これにより複写法の利点を保持しつつ，Incremental Copying GCの欠点であるreadバリアを解消している．

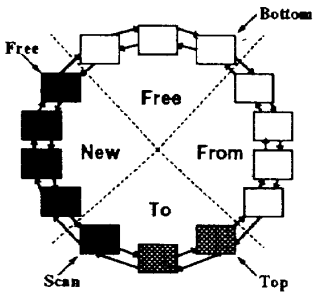


図 1: Treadmill

## 3. 本研究の方針

本研究では，objectの世代の概念を考慮にいれた，インクリメンタルな**Opportunistic Treadmill GC**の手法を提案する．

### 3.1 世代の概念

- 生成されて間もない(短寿命)objectほどごみになりやすい．
- 短寿命objectから長寿命objectに向かうポインタが大多数である．

この点を考慮すると，通常のGCを短寿命objectに限定して行ない，長寿命objectをGCの対象外とすることによって，1回のGC時間を大幅に短縮することができる[3]．通常のGCをある一定の回数以上生き残ると長寿命objectと判断するが，その回数(**Advancement Threshold**)は1から2の間が良いとされている[3]．Advancement Thresholdを1に設定すると，Tenured garbageが増加してしまい，また，2より大きくしても，Tenuringされるobjectが減る割合が少ないためである．

### 3.2 Opportunistic Treadmill GCの提案

Opportunistic Treadmill GCでは通常の3色(black,grey,white)による色分け[2][4]に加え，長寿命objectはsilver，GC直前に生成されたobjectはyounger_blackで色づけを行なう．ここで，silverおよびyounger_blackはblack同様，すでに走査済みであるものとし，collectorはsilver，younger_blackの子objectの走査は行なわない．また，長寿命objectは通常の双方向環状リストからは取り除き，長寿命object用のtreadmillに属するものとする．

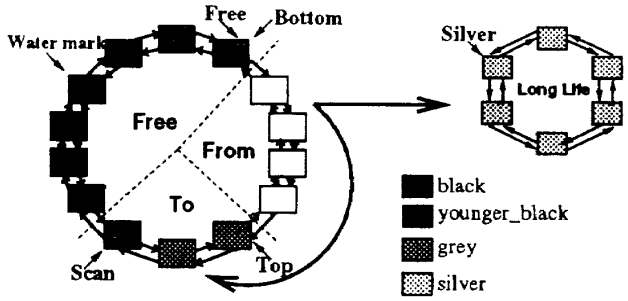


図 2: Opportunistic Treadmill

- objectのアロケイト
  water mark(図2)以前にアロケイトされたobjectはblackで，それ以後はyounger_blackで割り当てる．
- 生きているobjectのrelink(図2)
  - white object

---

Treadmill Garbage Collection Considering the Object's life time

Tatsunobu KOIKE Masakazu NAKANISHI

Department of Computer Science, Graduate School of Science and Technology, Keio University 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Kanagawa 223, Japan

以前 black だった object は長寿命 treadmill へ relink し, collector による走査後 silver に色づけする.
以前 younger_black だった object は to 空間へ relink し, collector による走査後 black に色づけする.

- grey, black, younger_black, silver object
  何もしない.

この手法により, Advancement Threshold を 1 または 2 に設定したことになる.

## 4. 実験とその結果

Lisp1.5 ベースの処理系に一括型の Treadmill GC(以下 TMGC), 及び一括型の Opportunistic Treadmill GC(以下 OPTMGC) を実装し, ヒープ領域には object 30 万個の条件で以下のベンチマークプログラムを実行した.

フィボナッチ数列 (fib 30)
アッカーマン関数 (ack 3 8)
TARAI 関数 (tarai 16 8 4)　　BOYER
BIT (bit '(a b c d e f g h i j k))

図 3 ～ 6 は, 1 回の GC で relink された object の数及びその時点での長寿命 object の総個数も示している. また GC 回数と GC 時間については表 1 に示しているとおりである.

Treadmill GC は複写式 GC をベースにしているため, object の生存率が高い場合には効率が低下する. しかし表 1 が示すように, 比較的実装が容易な一括型 Opportunisitc Treadmill GC によって, 長寿命と判断した object を通常の GC の対象外とし, 無意味なポインタの付け換えをなくして GC 時間を大幅に削減して効率をあげている.

表 1: GC 回数 (回) の比較と GC 時間 (秒) の比較

| ベンチマーク | TMGC | | OPTMGC | |
|---|---|---|---|---|
| | 回数 | GC 時間 | 回数 | GC 時間 |
| フィボナッチ数列 | 53 | 0.07 | 53 | 0.03 |
| アッカーマン関数 | 112 | 1.04 | 114 | 0.92 |
| TARAI 関数 | 514 | 0.55 | 521 | 0.15 |
| BOYER | 33 | 3.02 | 36 | 0.35 |
| BIT | 2 | 0.11 | 2 | 0.14 |

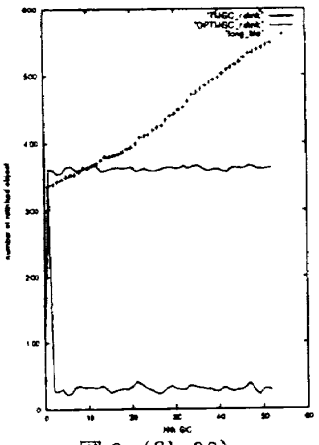
図 3 (fib 30)

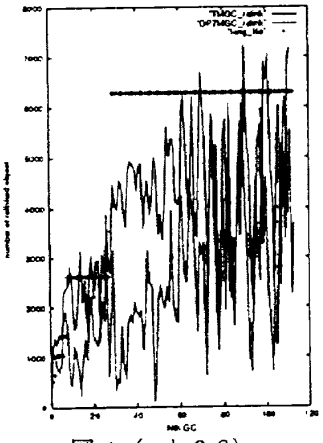
図 4 (ack 3 8)

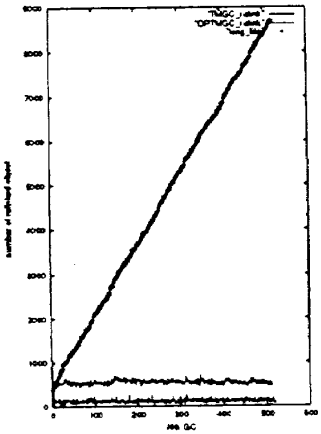
図 5 (tarai 16 8 4)

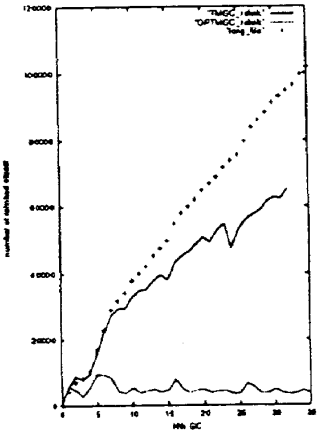
図 6 (boyer)

## 5. 今後の展望

GC の健全性を満たすためには, Tenured garbage も回収しなければならない. どのようなタイミングで, どのような手法で回収するかを今後検討していく. また, Treadmill GC はインクリメンタル GC の手法として考案され, mutator と collector との同期操作も write バリアによる実装が可能である. 4. 章の実験とその結果からも, 1 回の GC 時間を最小限に抑えることを要求するインクリメンタル GC としても, Opportunistic Treadmill GC の手法は効率の向上を期待できる.

## 参考文献

[1] Henry.G.Baker. The treadmill: Real-time garbage collection without motion sickness. *ACM SIGPLAN Notices*, Vol. 27, No. 3, Mar 1992.
[2] R. Jones and R. Lins. *Garbage Collection: Algorithms for Automatic Dynamic Memory Management*. Wiley, 1996. With a chapter on Distributed Garbage Collection by R. Lins.
[3] 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会研究報告, Vol. 91-SYM-61-5, pp. 1-7, September 1991.
[4] Paul R. Wilson. Uniprocessor garbage collection techniques. In Yves Bekkers and Jacques Cohen, editors, *Proceedings of International Workshop on Memory Management*, Vol. 637 of *Lecture Notes in Computer Science*, University of Texas, USA, 16-18 September 1992. Springer-Verlag.