記　号　処　理 75－4
（1994．9．16）

印付けと回収と純計算を並列に実施するごみ集め

鈴木 貢
電気通信大学 情報工学科

寺島 元章
電気通信大学 情報システム学研究科

概要

マークアンドスイープ法によって固定長セルの回収を行なう並列ごみ集め (GC) を提案する。本 GC と従来の並列 GC との大きな違いは、後者は印付けと回収を直列的に実施するのに対し、本 GC はそれらを並列に実施する点にある。本稿の骨子は、印付けを行ないながら回収を行なう「尺取り虫印付け」である。印付けは、スタックを使ったリストたどり法を用いるので、それに要する時間量は生存セルの個数に比例する。また本 GC では, 従来の印付け回収法では必須であった回収時の印の取り外しが不要である。しかも、印付けプロセスと回収プロセスの間に際どい部分がなく、それらは非同期に稼働することができる。

Inchworm Marking and "Mark during Sweep" garbage collection

Mitsugu Suzuki
Department of Computer Science and Information Mathematics
The University of Electro-Communications

Motoaki Terashima
Graduate School of Information Systems
The University of Electro-Communications

1-5-1 Chofugaoka, Chofu-Shi, Tokyo 182 Japan

Abstract

A parallel garbage collector (GC) which is based on a mark and sweep method for fixed sized cells is presented. This parallel GC performs both marking and collection processes in parallel, though conventional parallel GCs do them serially. This is the major difference between the two. A technical point of our parallel GC is use of inchworm marking which can mark active cells during the whole time even if collection task is in process. It uses a push-down stack to traverse list structures, and it requires the time proportional to the number of active cells. The collection task is free from unmarking process which is indispensable to the conventional mark and sweep GCs. There exists no critical region between marking and collection processes, so that they can run asynchronously.

1 はじめに

近年、コヒーレントキャッシュ等の技術が進歩し、共有メモリを持つ密結合型並列プロセッサの効率的な稼働が可能になった。一方、GCを必要とする応用や言語で、純計算の停止が好ましくないものがある。このようなGCの開発という問題に対して、マークアンドスイープ法による並列GC[2],[4]等が提案された。これらでは、2台の並列プロセッサをそれぞれGCと純計算(mutator)に割り当て、両者を並列に実施し、GCのためのmutator停止を回避している。これらでは。GCプロセスは「使用中セルへの印付け」(印付け)と、「ごみセルの回収」(回収)の二つの作業を直列的に行っている。仮に印付けと回収を並列に行なうことが可能であれば、全体としての計算の並列度をより高めることが可能である。本稿で述べるのは、GCの役割を印付けプロセス(marker)と回収プロセス(collector)に分割し、印付けと回収を並列に実施する並列GCである。この点が本稿の並列GCと従来の並列GCとの本質的な相違点である。

本稿のGCはスタックを用いたリストたどり法によって印付けを行なうので、印付けにかかる時間量は「使用中のセルの個数」に比例する。使用中セルの占有率が十分小さい(数%程度)場合は、リストたどり法による印付けが、走査法による印付けに較べてはるかに効率が良いことは明白である。そして、印付けと回収の実行時間を同程度にし得るので、印付けと回収を並列に行なうという利点が生じる。

ところで、従来の並列印付け法は、印付けと回収の並列実行を許さなかった。我々は、この問題の解として「尺取り虫印付け法」を提案する。この方式については、2節で詳述する。本稿では、使用中セルの追跡法(印付け戦略)に湯浅によって提案されたスナップショット印付け法[7]を用いている。しかし、本稿のアルゴリズムは、この印付け戦略に特化しているわけではない。並列印付け戦略間の比較はWilsonが[6]で詳述している。

以下アルゴリズムを詳述している箇所では、拡張された並列Pascal-Sを用いて記述している。この仕様については、[1]で述べられている。そして、それらの箇所では行の先頭に小さい字体で行番号が割り当てられている。その番号はアルゴリズムの説明において$\ell 1$のようにして参照される。また、定数の依存関係を明示するために、定数の定義で式を用いることがある。また、プログラム中および文中での"$\leftarrow$"の記号は、Pascalで代入の金物表現である":="と同じ意味である。

2 尺取り虫印付け法

従来マークアンドスイープ法では、セルの状態は「印付き」と「印なし」の2つのみが定義されていた。GCは次の作業を繰り返す。

- (印付け期間)プログラムの実行環境から到達可能な(以下単に到達可能という)セルの全てに印を付ける。
- (回収期間)ヒープ全域を走査して
  - (回収)印が付いていないセルを再利用可能なセルのリスト(以下フリーリストと呼ぶ)へ挿入する。
  - (印の外し)印がついたセルの印を外す。

ごみであるセルが「回収可能である(ごみである)」と判明するのは、回収期間中のみである。つまり印付け期間中は、回収可能であることが確定しているセルは存在しない。このことが、印付けと回収を直列化していた。別の見方によれば、印を外す作業を「印がないという印」の再利用と見做すことができる。そして、このことが次の印付け期間中のごみセルの回収を不可能にしていた。従って、このような印の再利用を行わないことが、印付けと回収の並列化を可能にすると結論することができる。印を再利用しないことは、具体的には次のようにすることである。

- 印は正の整数であり、各セルの印の初期値は0である。
- 2つの変数、fixedとunfixedを用意する。fixedの初期値は2,unfixedの初期値は1である。セルが「回収可能」であるとは、その印がfixedでもunfixedでもないことであると定義する。
- markerの作業は以下を順番に繰り返すことである。

1. (印付け期間) 到達可能なセルの全てに fixed 印を付ける。
2. (休止期間) 印付けが終わると、unfixed ← fixed とし、印付けの休止期間に入る。
3. (印付け期間への切替え)fixed を 1 増し、collector との同期 (後述) を取る。

- collector はヒープを順番にサーチしながら、回収可能セルをフリーリストへ挿入する。(以下この作業自体を回収と呼ぶ) 印の外しは行わない。

2 つの印 fixed と unfixed の意味は、それぞれ「到達可能であることが確定している」と「到達可能であるのかそうでないかがはっきりしない」ということである。
marker と collector は fixed と unfixed の値を共有している。そして、marker がこれらの値を更新する時、「回収可能でないセルを回収しない」という点での健全性を保証するために、両者の間での同期が必要である。休止期間へ入る時点で印付けは完了していることから、unfixed 印の付いたセルは到達可能でないことが確定している。従って、marker が休止期間に入る前に unfixed ← fixed とすることは、問題無い。問題であるのは、次の印付け期間に入る前に fixed を 1 増すことである。marker が fixed を更新してその値を用いて印つけを開始するが、collector が fixed の値が更新されたことを知らないで、その新しい fixed の値が付いたセルを回収する可能性がある。このことを回避するために、marker は collector が fixed の値の更新に気付いたことを確認する必要がある。collector との同期はこの目的のために行われる。その詳しい実現法は、4節で述べる。結果的に fixed に割り当てられる値は印付け期間毎に 1 づつ増し、Figure 1のように、unfixed は fixed を 1 または 0 の距離をおいて追いかけている。この様子が、この印付け法の名前 (尺取り虫印付け法) の由来である。そして、この印付け法は「印付けをしながら回収を行なう」という命題を達成している。

しかし、印付けが完了する度に fixed(と unfixed) を増すことの無限の繰り返しは、セルのサイズの点から実用的ではない。そして、印をある程度の大きさの数 Mの剰余系としても構わないといえる。その理由は、m を fixed でも unfixed でもない印のひとつであるとして、

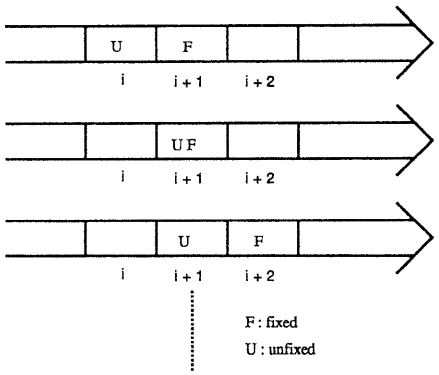

Figure 1: Behavior of the two variables, "unfixed" and "fixed"

m が付いたセルの全てが collector によって回収済みであるならば、m を fixed に割り当てること、つまり印の再利用が出来るからである。Mの大きさの最小値は、marker が印つけにかかる時間を T_m (到達可能なセルの量と一つのセルの印つけにかかる時間の積)、collector がヒープを一順するのにかかる時間を T_cとして、次のようになる。

$$M \geq \lfloor \frac{T_c}{T_m} \rfloor + 2 \ldots (1)$$

仮に m 印の付いた未回収のごみセルがヒープに残っているとしても、m を fixed に割り当てることは、回収してはいけないセルを回収することはないという点で、健全である。それは、このことが単に回収可能セルの回収を一時的に諦めることに過ぎないからである。m は再び fixed でも unfixed でもない印に戻り、回収を諦めたセルは回収の対象になる。このような状況を「早めの割り当て」と呼ぶことにする。早めの割り当ては、Mが 1 式で表された値以下であるときに発生する。そして、等価的に (使用不能セルの) 占有率を増加させるので、GC の効率を悪化させる。

3 印付け戦略

本稿で用いているスナップショット印付け法の主張は、「印付けの完了時に印が付けられてい

るのは、印付けの開始時に使用中であったセルと印付け中に確保したセルである。」ということである。この印付け法では、古荘らの並列GC[9]のようなcopy on write技法を用いた実現法も存在するが、本稿では、湯浅の論文で述べられている実現法を用いる。
印付けのためのスタック(以下単にスタックと呼ぶ)を用意する。印付けは、以下のように実施される。

1. (スナップショット)印付けの開始時のプログラムの実行環境の根(以下Rと呼ぶ)の持つデータで、ポインタであるものをmutatorとは不可分にスタックに積む。
2. (追跡)スタックが空になるまで、スタックへの積み卸し行いながらセルに印を付ける。

またmutatorは、印付け期間中のセルの確保、書き換え時に以下を実行する必要がある。

- セルを確保したら、それに印を付ける。
- セルの書き換え時に、上書きされた値がポインタであり、それで指されているセルに印が付いていないなら、そのポインタをスタックに積む。

セルは、スタックに積まれる時に印を付けられる。mutatorとmarkerは,スタックとRを共有している。一般にRは、言語処理系のスタックやレジスタ等である。スタックへの積みや卸し(後述のmspushやmspop)の際、スタックの首尾一貫性を保証するためにそれらは不可分に行われる必要がある。そのためにセマフォ等が使用されるであろう。あるいは、スタックの代わりに[4]のように双頭のキューを用いることも考えられる。
また、Rのスナップショットを不可分に行うための何らかの方策が必要である。markerがmutatorを一時的に停止させることや、copy on writeの技法の使用が考えられる。あるいは、markerがmutatorに割り込みをかけて、Rの内容をスタックに吐き出させることも考えられる。本稿では、それらについて詳述しない。

4 アルゴリズム

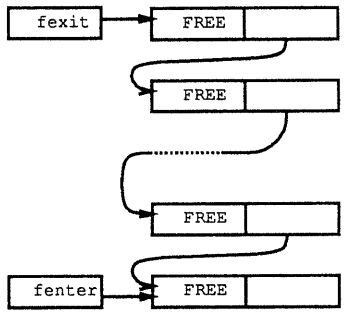

Figure 2: List of free cells

系の初期化の後、3つのプロセスが生成される。

```
1  begin
2    系の初期化;
3    cobegin
4      mutator; { 純計算プロセス }
5      maker; { 印付けプロセス }
6      collector; { 回収プロセス }
7    coend
8  end.
```

「系の初期化」の直後はRから到達可能なセルは存在しないとする。また、unfixed = 1, fixed = 2であるとする。

4.1 共通データ構造

印のデータ型mark_t、セルを指すポインタ型pointer_t、セルのデータ構造cell_t、ヒープのデータ構造heap、確保したセルを指すポインタの格納場所allocを定義する。また、先に述べた二つの変数fixedとunfixedを定義する。セルは、LispのCONSセルのような単純な2つ組のポインタであるとする。また、ひとつのセルのあるフィールドへの書き込みは、アトミックであるとする。ヒープは大きさH + 2のセルの配列であり、ポインタはその配列へのインデクスであると定義する。allocはRに含まれ、スナップショットの対象になる。

```
9  type mark_t = 0..M-1;
10 type pointer_t = 0..H+1;
11 type cell_t = record
12   mark : mark_t;
13   left, right : pointer_t
14 end;
```

```
15 var heap:array[pointer_t]of cell_t;
16 var alloc:pointer_t;
17 var fixed, unfixed: mark_t;
```
応用においては、実現者はセルのデータ構造にタグを付加して、セルにポインタ値だけでなくスカラ等の種々のデータを表現する能力を与えるかも知れない。しかし、本稿では議論の単純化のために、そのような工夫は施さない。
以下のポインタ値を予約する。
```
18 const NULL=H; FREE=H+1;
```
NULLは唯一の「何も指さない」ポインタであり、リスト構造の終端を表現する。アルゴリズムの簡潔化のためにheap[NULL].left=NULLかつheap[NULL].right = NULLであるとする。
car部がFREEであるセルの主張は、自分が回収済みであるということである。これは、collectorが回収済みセルを回収することの防止のための工夫である。collectorが回収可能セルを走査するのに使うポインタfpと、フリーリストを構成するデータ構造を定義する。
```
19 var fp:pointer_t;
20 var fenter,fexit:pointer_t;
```
fenterはフリーリストの末尾のセルを指している。fexitはその先頭のセルを指す。その様子をFigure 2に示す。fenterとfexitに関する主張は、「fenter ≠ fexitならば、fexitで指されているセルを使うことが出来る」ということである。以下のスタックの操作を行う手続きと、Rをスナップショットする手続が定義されているとする。
```
21 procedure mspush(p : pointer_t);
22 function mspop: pointer_t;
23 procedure snapshot;
```
mspushは、markerが印付け期間中であり、pがNULLでもFREEでもなく、またpで指されたセルにfixed印が付いていなければそれにfixed印を付けて、pをスタックに積む。印付け期間中であることは、unfixed≠fixedであることを調べることで知ることが出来る。mspopは、スタックが空でないならばスタックから1つ値を卸しそれを返し、さもなくばNULLを返す。先に述べたように、これらでは、スタックを巡る排他制御が行われるとする。snapshotは、Rのスナップショットを行う手続きであるとする。

4.2 mutatorの呼び出す手続き

セルを確保する手続きcreateを定義する。
```
24 procedure create;
25 var t:pointer_t; { 一時変数 }
26 begin while fexit=fenter do ;
27   t←heap[fexit].right;
28   heap[fexit].mark←fixed;
29   heap[fexit].right←NULL;
30   heap[fexit].left←NULL
31   alloc←fexit;
32   fexit←t;
33 end;
```
確保されたセルのleft部とright部はNULLで初期化される。この様子をFigure 3に示す。
```
34 procedure storel(d,v:pointer_t);
35 begin mspush(heap[d].left);
36   heap[d].left←v;
37 end;
38 procedure storer(d,v:pointer_t);
39 begin mspush(heap[d].right);
40   heap[d].right←v;
41 end;
```
storelとstorerはdで指されたセルのそれぞれleft部とright部をvに書換える手続きである。これらが呼び出される時、dとvに割り当てられる値は、Rから到達可能な箇所に存在するとする。

4.3 collector

collectorを定義する。collectorは、一度回収したセルのleftフィールドをFREEにし、それを再び回収することはない。

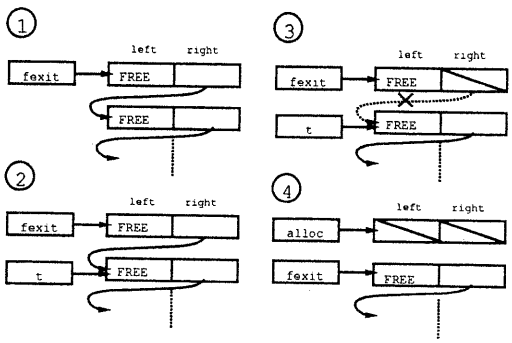

Figure 3: Allocation of a new cell

```
42 procedure collector;
43 begin
44   while TRUE do begin
45     if heap[fp].mark ≠ fixed and
46        heap[fp].mark ≠ unfixed and
47        heap[fp].left ≠ FREE then
48       begin heap[fp].right = NULL;
49         heap[fp].left = FREE;
50         heap[fenter].right = fp;
51         fenter = fp;
52       end;
53     fp←(fp+1) mod H
54   end
55 end;
```

4.4 marker

marker を以下に定義する。marker の制御が *l*60 ~ *l*64 にある間を「印付け期間」、*l*65 ~ *l*66 にある間を「休止期間」、*l*67 ~ *l*69 にある間を「印付け期間への切替え」と定義する。また、*l*59 ~ *l*70 を「GC 期間」と定義する。

```
56 procedure marker;
57 var p:pointer_t;
58 begin
59   while TRUE do begin
60     snapshot;
61     repeat p←msopop;
62       mspush(heap[p].left);
63       mspush(heap[p].right);
64     until p = NULL;
65     unfixed←fixed;
66     休止期間;
67     fixed←(fixed+1) mod M;
68     { collector との同期 }
69     p←fp; while p=fp do ;
70   end
71 end;
```

5 健全性の証明

ここで、印付け期間に関して、以下の主張が成立しているとする。(印付けの正当性)

- marker は、スナップショット時に到達可能であったセルの全てに、印付け期間終了の時点で fixed を付け終わっている。(A1)

本稿ではこの主張に関する証明は行わない。印付けの正当性に関しては、用いる印付け戦略(本稿の場合はスナップショット印付け)を提案している論文で議論されている。
本稿の GC に関して、次の定理が成立する。

定理 1 尺取り虫印付け法ではごみになったセルは必ず回収可能となる.

証明 (1) ある到達可能なセルが i 番目の GC 期間にごみ (プログラムの実行環境から到達不能であること) になるとする。ごみになった i 番目の印付け期間の終了時点で、その印は fixed または unfixed であることが考えられる。marker は i+1 番目の印付け期間以降ではそのセルに fixed 印を付けないので、i+1 番目の印付け期間の開始時点では、そのセルの印は unfixed でも fixed でもないか、または unfixed である。前者の場合、i+1 番目以後の印付け期間は回収可能である。後者の場合、i+2 番目以後の印付け期間は回収可能である。よって、ごみになったセルは、最大で 2 つの印付け期間を経た後、必ず回収可能となる。

定理 2 *collector* は到達可能なセルを回収しない.

証明 (2) collector は fixed および unfixed 印が付いているセルを回収しない。従って、この命題が真であることを示すには、到達可能性のあるセルの全てに unfixed か fixed が付けられていること (命題 P1 としよう) を示せば良い。印付け期間、休止期間、印付け期間への切替えのそれぞれで P1 が成立することを示す。

1. 「系の初期化」の直後の印付け期間の入口では、到達可能なセルは存在しない。また、印付け期間に到達可能であるセル (確保されたもの) には fixed 印がつけられる。よって P1 は成立する。
2. その次の休止期間では到達可能となったセル (確保されたもの) には fixed 印が付けられ、その前の印付け期間で到達可能であったセルには fixed 印がついているので、P1 は成立する。

3. 印付け期間への切替えについて考える。marker が新しい fixed の値で到達可能なセルに印付けを開始するが、collector が fixed の新しい値を知らないで古い値との比較を行い、marker が新しい値を付けた「到達可能」であるセルを回収してしまう可能性がある。この問題は、プログラム中で marker が fixed を更新した後で、fp が更新されるのを待って、次の印付け期間に入ることで解消される。
その理由は、collector は Figure 4のような制御構造を持つが、fp の値の更新は collector が C 点を通過することと同値であり、collector が fixed の最新の値を使うことを保証しているからである。そして、印付け期間への切替えの終了時点で到達可能なセルには unfixed 印が付いている。よって P1 は成立する。

4. その次の印付け期間では到達可能なセル(この場合は確保されたセルと marker が追跡したセル) には fixed 印が付けられ、終了時点では A1 よりこの期間中に確保されたセルとスナップショット時に到達可能であったセルの全てに fixed 印が付けられる。よって P1 は成立する。

第 1 項と、帰納法を第 2,3,4 項に適用することにより、常に P1 が成立することがわかる。
よって、仮定は証明された。

定理 3 *marker*はフリーリストのセルに印を付けない

証明 (3) create が唯一フリーリストのセルを到達可能にする手続きであり、フリーリストからセルを確保し、そのセルへのポインタを alloc に書き込んでいる。その様子は Figure 3の通りであるが、1 と 2、2 と 3、3 と 4 のどの段階の間でスナップショットが実施されるとしても、確保されるセルがフリーリストの一部である時に alloc がそれを指すことはない。
よって仮定は証明された。

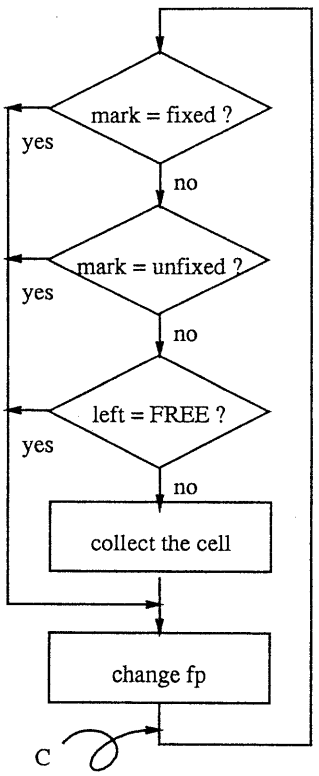

Figure 4: The controll flow of the collector

6 関連研究

Queinnec らも印付けと回収を並列に行なう GC[5] を提案した。彼らの GC の印付け法は [2] で述べられている走査法の変形である。よって、印付けにかかる時間はヒープの大きさ H とヒープの走査回数 S の積に比例する。最悪の場合、S はリストの最大長であることに注意されたい。一方、回収にかかる時間は H に比例する。一つのセルの印付けと回収にかかる単位時間を考えれば、印付けにかかる時間が回収にかかる時間の数倍になるであろう。そして、彼らの GC では collector と marker はそれぞれの作業の終了時点で片方の作業の終了を待つという同期を行なう。以上より、collector が作業の終了後に回収に要した時間の何倍かの間、marker が作業を終了するのを待ち続けるであろう。このことは、印付けと回収を並列に行なう利点を失わせている。

中沢らは [8] で本稿で述べた尺とり虫印付け法の特殊な場合 ($M = 3$) を示した。

7 まとめ

純計算と並行して、印付けと回収を並行に実施するごみ集めアルゴリズムを提案した。印付けと回収の間の同期のオーバヘッドはほとんどなく、共有変数を巡るきわどい領域がない。純計算と印付けのきわどい領域は、印付けスタックを巡るもののみである。
実用に際しては、文字列や配列等の可変長データ構造を扱う能力も必要かも知れない。2 ヒープのコピー圧縮方式との組み合わせは、この問題の解決法の一例である。具体的には、固定長セルを可変長データ用のヒープへのポインタであるとし、印付け時に未コピーの可変長セルをコピーする。mutator と marker の競合は、セマフォを可変長セル毎に設けることで解消する。同じセルを巡って両者が競合する可能性は充分低いであろう。あるいは、[10] で提案されている方法を用いることも考えられる。
また、Johnson は [3] で両者の競合による latency を小さくすための方策を述べている。

References

[1] BEN-ARI, M. *Principles of Concurrent Programming*, Prentice-Hall (1982).

[2] 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, *CACM*, **21**, 11 (November 1978), 966–975.

[3] JOHNSON, R. E. Reducing the Latency of a Real-Time Garbage Collector, *ACM Letters on Programming Language and Systems*, **1**, 1 (March 1992), 46–58.

[4] KUNG, H. T. and SONG, S. W. AN EFFICIENT GARBAGE COLLECTION SYSTEM AND ITS CORRECTNESS PROOF, *Proc. 18th Annual Symposium on Foundation of Computer Science* (1977), 120–131.

[5] QUEINNEC, C., BEAUDOING, B. and QUEILLE, J.-P. Mark DURING Sweep rather than Mark THEN Sweep, PARLE'89 (LNCS 365,366), Springer-Verlag (1989).

[6] WILLSON, P. R. Uniprocessor Garbage Collection Techniques, IWMM 92 (LNCS 637), Springer-Verlag (1992).

[7] YUASA, T. Real-Time Garbage Collection on General-Purpose Machines, *J. SYSTEMS SOFTWARE* (November 1990), 181–198.

[8] 中沢雅博, 田内康之, 斉藤宗昭 使用中セルのちらばりに注目した実時間塵集めの提案, 情報処理学会研究会報告 SYM63(7)AI80(7) (January 1992).

[9] 古荘進一, 松岡聡, 米澤明憲 Concurrent Conservative Garbage Collection, 情報処理学会研究会報告 PRG3(20) (July 1992).

[10] 鈴木貢, 寺島元章 可変長セル用並列・実時間ごみ集め, 情報処理学会研究会報告 PRG3(20) (July 1992).