圧縮型高速ガーベッジコレクション

寺島元章† 石田 満† 笹原豪士†

Morrisの方法とソート技法を結合した圧縮型高速ガーベッジコレクションであるSMC（Sort & refined Morris Compaction）の機能とその性能評価について述べる．SMCは使用中セルの塊である各クラスタの代表アドレスをソートした結果を利用することで，クラスタの総容量に比例する時間で圧縮処理を行うことができる．ソート処理に必要な作業領域量はセルの格納領域量の5%を上限としてその効率的な運用ができるよう動的に決められる．ソート処理が放棄される場合でも，従来の圧縮型ガーベッジコレクションと比較して負荷が増大することはない．Lisp処理系であるPHLに組み込まれたSMCの処理速度は，同じく実装された既存の圧縮型や複写型のガーベッジコレクションよりも高速であることがLispプログラム実行結果から示されている．

Fast Compactifying Garbage Collection

MOTOAKI TERASHIMA,† MITSURU ISHIDA† and TSUYOSHI SASAHARA†

This paper presents the design and analysis of fast compactifying garbage collection called SMC (Sort & refined Morris Compaction) which is a mixture of a sorting scheme and Morris's compaction scheme. The SMC performs sliding compaction with the time proportional to the total size of clusters which are successive cells in use by virtue of sorted address data of every clusters. The size of working space which is required to store such address data is decided at run time considering its space efficiency, and is limited to 5% of the total storage space size. The SMC also performs sliding compaction with similar time cost to conventional compactifying garbage collection, even if sorting is not carried out. The SMC which has been implemented successfully on PHL, a dialect of Lisp, shows its processing time is shorter than that of conventional garbage collection of both compactifying and copying types.

1. はじめに

本論文は停止回収型ガーベッジコレクション（GC）の一種である圧縮型GCの効率化とその評価に関するものである．圧縮型GCは複写型GCとともに可変容量セルに対する効果的なGCの手法である．可変容量セルとはその大きさが固定でないセルのことで，シンボルやブロックなどの構造体の構築に有用である．圧縮型GCの問題点は処理時間がセルの格納領域の総容量に比例することであったが，ソート技法の採用により処理時間がほぼ使用中セルの総容量に比例するような手法[1)~3)]が最近提案されている．こうした手法を用いれば，圧縮型GCはその時間計算量で複写型GCと同じになること，さらに，格納領域全体が使用できること，格納領域中のセルの配置順序[4)]が保存されることなどで複写型GCより優位となる．

本論文では，そうした手法を圧縮型GCとして定評のあるMorrisの方法[5)]に適用し，さらに洗練することで，処理の高速化や記憶使用の効率化が図られることを述べる．本GCは基本的にMorris法とソート技法の結合様式であることから，SMC（Sort & refined Morris Compaction）法と呼ぶ．SMC法では，ソート処理に必要な作業領域量（以下，U-領域と呼ぶ）は格納領域量の5%を上限にしてその効率的な運用ができるよう動的に決められる．また，U-領域が不足してソート処理が不可能な場合には，単にMorris法により作業が行われるが，そのために多大の負荷が生じることはない．

SMC法はワークステーション（SONY NEWS-5000）で稼働中のリスト処理系であるPHL（Portable Hash Lisp）[6)]のGCとして実装されている．その処理速度は同じく実装した従来の複写型GCや圧縮型GCよりも高速であることがLispプログラムの実行結果から示されている．

† 電気通信大学情報システム学研究科
Graduate School of Information Systems, University of Electro-Communications

2. 背　景

2.1 停止回収型 GC

本論文で述べる GC は停止回収型 GC に属する．それは GC が起動される間はそれ以外の処理（純計算）を停止させるが，GC の設計や具現さらに評価が容易などの利点をもつ．実時間処理を必要としない言語処理系や汎用計算機上の処理系では，その効率的側面から停止回収型 GC が現に大勢である．現在でもその開発や改良などが行われている．

2.2 圧縮型 GC

圧縮型 GC は使用中セルをそれらの配置順序（アドレスの大小関係）[4]も保持しながら格納領域の一端に移す．使用中セルは相互の位置関係を変えずに一方向に移動することから「圧縮」という名前が付いている．圧縮型 GC は印付け（marking）という処理で使用中セルを識別してから，それらの移動に伴うポインタ補正と移動とを格納領域全体を走査しながら行う．このため，圧縮型 GC は格納領域の総容量（S）に比例する時間（$O(S)$）を要するというのが定説であった．

例えば，Morris 法はポインタ補正を含む圧縮処理に格納領域の 1 往復の走査を必要とする．行きの走査でそれと同じ向きのポインタ補正を，帰りの走査でも同じ向きのポインタ補正と圧縮とを行うためである．

この時間量の改善は圧縮型 GC の長年の懸案であったが，使用中セルのアドレスをデータとして大小順に高速ソートする方式により，

$$O(A)+O(R\log(R)) \tag{2.1}$$

に短縮された[1]．ここで，A と R はそれぞれ使用中セルの総容量と参照される（使用中）セルの個数である．使用中セルのアドレスが大小順にわかれば，格納領域を一方向に使用中セルのみを $O(A)$ の時間量で走査することが可能になるからである．一般に，$A=kR$（k は 1 以上の定数）の関係が成り立つので，式 (2.1) は

$$O(A\log(A)) \tag{2.2}$$

と同等である．さらに，ソートの対象を参照される個々のセルではなく，使用中セルの塊であるクラスタの端のアドレスにできるならば時間量は

$$O(A)+O(N\log(N)) \tag{2.3}$$

にまで短縮できる．ここで，Nはクラスタの個数（$N \leq R$）である．

Sahlin[2]らはこの点に注目して，使用中セルを 2 回たどることでクラスタの連鎖を作り，ソート技法によりそれを一方向の連鎖にする方法を提案した．この方法は付加的な領域を必要としないが，格納領域上に散在する連鎖を整列するための時間的負荷が大きいし，仮想記憶上の具現には適さない．

鈴木らも LLGC と呼ぶ新方式の圧縮型 GC[3]で式 (2.3) の時間量を達成している．LLGC 法ではポインタ補正に必要な表（補正表）を格納領域中の使用済みセルが占める場所の一部を利用して作り，記憶使用の効率化を追求している半面で，ソート対象のアドレスを格納するための付加的な領域（U-領域）を必要とする．その必要量は R である．格納されたアドレスは各クラスタの先頭を指すもの以外は削除され，残りの N 個がソートされる．

こうしたソート処理に要する時間量は

$$O(R)+O(N\log(N)) \tag{2.4}$$

である．LLGC 法は常にソートを使用する訳ではない．U-領域の容量が $A/10$ を超えないように設定されるため，ソートを使用するのは占有率（$\alpha=A/S$）が小さい場合に限られる．なお，LLGC 法は独自の補正・圧縮アルゴリズムを使用するために格納領域の走査は 3 回と多い．

一般的なアプリケーションでは使用中セルの集りはクラスタを形成しやすいので，$N \ll R < A$ であることが予想される．この場合，式 (2.3) は後半が無視できて，

$$O(A) \tag{2.5}$$

となり，Sahlin や鈴木らが提案した GC は時間量で複写型 GC と同じになる．

圧縮型 GC が従来の複写型 GC と大きく違う点は使用可能な格納領域の容量である．前者は格納領域すべてが使用できるのに対して，後者は一度に使用できるのはその半分である[7]．

圧縮型 GC のもう 1 つの優位な点は使用中セルがそれらの配置順序（アドレスの大小関係）[4]を保持しながら格納領域の一端に再配置されることである．GC の処理を経ても使用中セルの配置順序が保持されていれば，Lisp の SETF などによるリスト構造の変更がなく，新しい構造は常に古い構造を参照して作られるような場合，すべてのポインタは常に同じ方向を指すことになる．

この特質は 4.1 節で述べる SMC 法に利用される．また，選択点（choice points）の作成順序を効果的に知る必要のある Prolog 処理系の具現でもこの特質が利用されている[8]．

2.3 複写型 GC

複写型 GC は格納領域を二分した半領域（semi-space）と呼ばれる領域間で使用中セルを移しながら GC を行う方式で，その利点は $O(A)$ の時間計算量にある．さらに，移動（複写）作業の削減と使用領域の

効率化を目的として，いわゆる世代別（generation）管理を行う世代型複写法も具現されている[9]．新旧の二世代かそれ以上の世代（年齢）別に使用中セルを効果的に管理する方式であり，これが効率的に機能すると，多くの古いセルが複写作業から除かれ，GC処理が高速になり，複写に使用できる領域も相対的に大きくなる．しかし，純計算時に課せられる付加や管理のための表（付加的な領域）も必要となる．世代別管理による利得とその仕掛けのための損失とは兼合の関係にあることも明らかである．

複写型GCでは一般に各データセルの配置順序と複写順序とは一致しない．複写方式によっては参照関係にあるセルが近接して配置されることから仮想記憶には有利という指摘もあるが，元々近接していたセルや共有セルでは特に効果がある訳ではない．

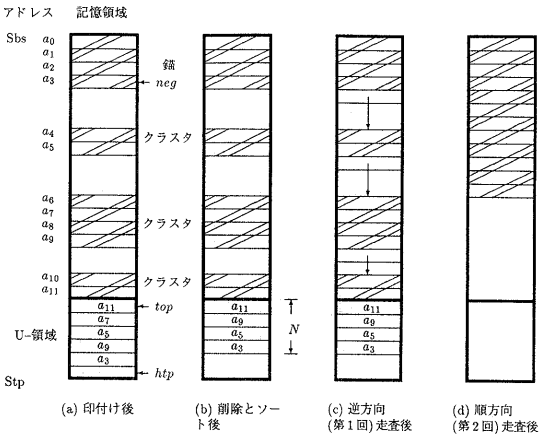
ソート技法を複写型GCに採用したものに生成順序保存複写型GCがある．従来の複写型GCに対して，複写対象のセルのアドレスをソートしてから，その順序に基づいて複写を行うものである．圧縮型GCと同様にセルの配置順序が保存されるが，従来の複写型GCに比べて処理時間ではソートによる負担増 $O(A\log(A))$ が生じる．このため，複写対象となる使用中セルが増加すれば，従来の複写型GC以上に処理時間が増加する．この対策として，データセルを新と旧（古）の二者に類別し，古いセルについては複写の作業を省き，1回当たりのGCの処理時間を確実に短縮させる方式が提案されている[10]．それでも一時に利用できる領域は古いセルがある領域を除いた残りの領域の半分である．

3. SMC法

SMC法はポインタ補正を含む圧縮処理にMorrisの方法を採用している．その理由は圧縮処理を格納領域の1往復の走査で実現するという実用性とそのアルゴリズム記述の明確さである．ただし，それは仮想的な処理系に基づいているため，SMC法での採用にあたり本章で述べるような変更が必要となった．

SMC法は，印付け，ソート処理，走査と処理が進む（図1参照）．ソート処理は省かれることもあるが，個々の処理部はMorrisの方法には見られない特徴をもつ．SMC法はソート対象のアドレスをU-領域と呼ぶ格納領域の一部に格納する点でLLGC法と似ているが，その領域の効率的運用のためにLLGC法に見られない機能を有している．こうした点については4章で述べる．

SMC法が実装されたPHLはCommon LISP[11]準拠の仕様を有し，翻訳系（compiler）指向と256MBの大容量格納領域の使用という特徴をもつ．PHLはそのデータ表現（図2参照）からも明白なように特異な処理系ではない．



（注）U-領域内のデータ配置は一例であり，そのサイズは図示のために誇張されている．

図1 SMCの動作
Fig. 1 Action of SMC.

データ型	フィールド	
	タグ	アドレス/データ
CONSデータ	0001	..... xy
シンボル	0011	..... xy
ブロック	0101	..... xy
長数値	0111	..... xy
短実数	1x00	.....
短整数	1x01	.....
文字符号	1x10	.....
システム（予備）	1x11	.....

(注) 'x' は印付けビット (marking bit) を示す.
'y' は Morris のシフトビット (shift bit) を示す.

図2 PHLのデータ表現
Fig. 2 PHL data representation.

3.1 2個のビット

PHLでは，個々のLispデータは対象計算機の1語（32ビットを想定）を最小の単位としてポインタタグを用いて表現される．セルの構成要素でもあるこの1語のことをフィールドと呼ぶ．アドレスとはこうしたフィールドを指す値のことである．各フィールドの上位4ビットがタグ部で，残りの28ビットで格納領域のアドレスや埋め込み型のデータを表す．圧縮で変化するのは前者である．

Morris法は，印付け用ビット（marking bit）に加えて，セルを指すアドレス部にシフトビット（shift bit）と呼ぶ1個のビットを必要とする．バイトアドレス方式では不要となるアドレス部の最下位2ビットをこれらに充てている．埋め込み型のデータはそのタグ部に印付け用のビットのみをもつ．これらはデータの値が変化しないためシフトビットは不要である．PHLの

データ表現ではこれらが無理なく具現できた．

3.2 走 査 法

PHL の格納領域はアドレスの小さい方から順（順方向）に使用される．このため，圧縮はアドレスの小さい向き（逆方向）に行う必要があり，Morris の方法（原型）とは対照的に，逆方向そして順方向と走査することになる．そこで，ソート処理の対象となるデータは各クラスタの終端フィールドを指すアドレスとなる．この効果的な登録方式については 4.1 節で述べる．

順方向の走査でも，印の付いていない使用済みフィールドは調べる必要はない．そこで，各クラスタの終端直後の使用済みフィールドを使用して，そこに次のクラスタの先頭アドレスを置くと，使用済みフィールド群を効果的に飛び越えることができる（図 1 (c) 参照）．こうした細工は最初の逆方向走査の処理過程でわずかな負荷で行うことができる半面，順方向の走査時間も $O(A)$ にすることができる．

4. 効 率 化

セルのアドレスを格納するための U-領域は，セルの圧縮方向とは反対側のアドレスの大きい格納領域の一端に確保される．これにより，錨（anchor）を成長させることと，U-領域量の変化に対処できる格納領域の効率的な運用とが可能になる．

錨とは，幾多の（圧縮型）GC を経て存在し続けるセルの集団が消滅や移動もなく格納領域の一端に詰まり固定化されたもので，ポインタ補正が不要という特徴をもつ．

U-領域は前もって確保しておく必要がある．その見積りとともにその量的な節約は GC を含む処理系全体の効率向上に大きく寄与することになる．

4.1 後掛け順の登録

U-領域へのアドレスの登録は印付け作業の一環として行うことができる．ただし，各クラスタの末尾を指すアドレスだけが最終的なソート処理に必要なデータであるので，それ以外のアドレスはできる限り登録から外すようにする．次は C 言語で記述された SMC 法の印付け手続き（関数）の一部である．各行先頭のコメントは説明の便宜上のものである．

```
/* 1 */ void store(Object *obj) {
/* 2 */ if ((Object)htp<Stp)
/* 3 */   *(htp++)=(Object)obj;
/* 4 */ else { /* U-領域の再使用 */ ;}
/* 5 */ }
/* 6 */ void markset_go(Object *obj) {
/* 7 */ mn++; /* 印付け個数の累計 */
/* 8 */ if (_sncip(*obj)) *obj|=UPPER_MARK;
/* 9 */ else {
/* 10 */  Object w=*obj; *obj|=LOWER_MARK;
/* 11 */  mark(w);}
/* 12 */ }
/* 13 */ void mark_cons(Object *obj) {
/* 14 */ if (obj<Sbs || markedp(*obj)) return;
/* 15 */ mn++; /* 印付け個数の累計 */
/* 16 */ if (_sncip(*obj)) {
/* 17 */   *obj|=UPPER_MARK;
/* 18 */   markset_go((Object *)obj+1);}
/* 19 */ else {
/* 20 */   Object w=*obj; *obj|=LOWER_MARK;
/* 21 */   markset_go((Object *)obj+1);
/* 22 */   mark1(w);}
/* 23 */ if (markedp(*(obj+2))) return;
/* 24 */ store(obj+1);
/* 25 */ }
```

関数 mark_cons はそのアドレスが obj で与えられる CONS セルの印付け処理を行う．ただし，実際の印付け作業は関数 markset_go との分担であり，CAR 部のみその先行処理をしている．セルの内容がポインタであれば，印付け関数 mark が再帰的に呼ばれる．行 18，21 と行 22 は CONS の CDR 部と CAR 部の処理である．このようにポインタは CDR 部，CAR 部の順でたどられるが，この順序は新規に CONS セルが作られる場合にその CAR 部の実体が CDR の実体よりも前に作られるという事実を考慮したものである．こうした手法を用いると，**図 3** に示すような CONS データについては登録される可能性のあるアドレスは 1 つ (a7) だけである．CONS を形成する他のセルのアドレス，a1，a3 と a5 が登録されることはない．行 24 ではその直後のセルに対して印の検査を行い，印がない場合にはクラスタの末尾であることから，関数 store を呼び，セルのアドレス (a7) を U-領域に登録

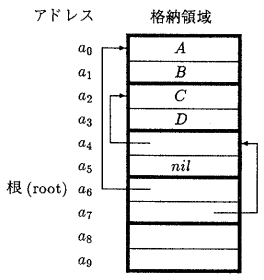


図 3 $((A\ .\ B)\ (C\ .\ D))$ の CONS セル
Fig. 3 CONS cells for $((A\ .\ B)\ (C\ .\ D))$.

する（行24）．これが後掛け順と言われる登録である．

行2はU-領域の使用の可否を調べている．U-領域がセルのアドレスで満たされる（オーバフロー）と，この条件が不成立になる．これ以降はアドレスが登録ができず，後のソート処理は放棄される．ただし，登録済みのアドレスの中にも印付け処理の進行で不要なものができている可能性があるので，それらを新規のものと置き換えるようにしてU-領域の再利用は可能であるが，現在はこうした処理は行っていない（行4を参照）．

行7と15は印を付けたフィールドの個数の累計を行っている．これは，印付きフィールドの総数が後続するポインタ補正の処理過程で必要となるからである．

4.2 U-領域量の設定

先に述べたようにU-領域の容量は事前に決められる．その初期値は格納領域量の1%であるが，各回のGC処理の結果により次のように更新され，次回のGCに適用される．

(1) U-領域がオーバフローしたときは，その領域量を1.5倍にする．

(2) U-領域の使用量が半分に満たないときは，その未使用域の半分を減らす． (4.1)

U-領域量が増加するのは(1)の場合であるが，そうした場合でもその上限は格納領域量の5%である．この値を越えるようなときはセルの生成が極めて特異な場合か占有率が大きい場合が想定される．そうした場合にはソート処理によるGCの高速化が期待できないので，その後はU-領域を確保しない．以降すべて，オーバフローした時と同様にMorris法でGC処理が進行する．

U-領域を効率的に使用する方策が(2)である．未使用部分が半分を超えた時に，次回から領域を小さくするのであるが，「半分の減量」にするのは実際の使用量の変動を考慮したためである．この評価については5.2節で述べる．

4.3 錨の処置

前述したように，錨を指すポインタは補正する必要はない．錨の存在は印付け後に，Sbs番地（格納領域の先頭番地）のフィールドに印が付くことによってわかる．また，その終端フィールドの番地はソート後に，その末尾の値として与えられる（図1(b)のa3）．このアドレスをnegとすると，neg以下のポインタ値については印はずし（unmark）以外の処理を省くことができて，これは明らかに処理の高速化となる．

5. 評　　価

5.1 従来の方法との比較

SMC法の評価のために，LLGC法，MorrisのGC，複写型GCの各処理に要する総時間とを比較した．これを四者の実装によるPHL処理系の実験から示した．

PHLでは，GC時にスタックの使用部分とハッシュ表（シンボルのエントリ表）の全部が調べられる．そこで，一般性を保つためにハッシュ表を便宜的に小さく（エントリ数199）してその処理時間に対する寄与を極力減らしている．この数値は四者の実装すべてに共通する．

LLGC法の文献[3]にはU-領域量の動的変更法の具体的な記述がないので，SMC法に準拠した変更法(4.1)をLLGCに実装した．複写型GCの実装では，スタックを用いる「深さ優先方式」でなく，2つのヘッダを用いる「幅優先方式」を複写過程に採用した．理由は，陽あるいは陰にスタックを用いないで済むことよる平易さとその高速性にある．

実験に用いたLispプログラムは，TPU[12]，BitA-9[13]と次に示す変形Tarai-4（TARAI[13]の変形版）である．

```
(defun tarai (x y z a) (progn
  (setq a (list
    (cons 'x x) (cons 'y y) (cons 'z z) x y z))
  (cond ((> x y) (tarai
    (tarai (1- x) y z ()) (tarai (1- y) z x ())
    (tarai (1- z) x y ()) ()))
  (t y))))
(tarai 8 4 0 ())
```

各プログラムについてセルの格納領域量を変えて，翻訳系（compiler）の実行環境で行った．TPUはPROG形式を解釈（PROG interpreter）する実行環境でも行った．その違いは実行過程で解釈系自身が生成するCONSセルの増加によるGCの回数増に表れる．

表1の結果は，SMC法がMorris法やLLGC法そして複写型GCよりも常に優位にあることを示している．**図4**はこのことを視覚的に示している．図4(a)は表1のTPU（PROG interpreter）を処理時間（縦軸）と格納領域量（横軸）でプロットしたものである．図では下側にある方がGCの処理時間が短く優位になる．SMC法（図中の黒丸）はいずれの方法よりも下側に位置しその優位性が示されている．なお，格納領域が大きい右側で優位となる複写型GC（図中の＋印）と格納領域が小さい左側で優位になるMorris法（図中の星印）の両者が途中で交差することもわかる．

表1 GC 処理時間
Table 1 GC processing time.

プログラム		TPU (compiler)							
格納領域量	占有率	SMC		LLGC		Morris		Copy	
KB	平均値	回数	時間 (sec.)	回数	時間 (sec.)	回数	時間 (sec.)	回数	時間 (sec.)
48	0.641	3 + 55	0.91	3 + 55	1.04	56	1.03	–	–
64	0.469	2 + 26	0.47	2 + 26	0.46	27	0.58	–	–
120	0.246	1 + 9	0.16	1 + 9	0.21	10	0.26	56	0.45
200	0.137	5	0.07	5	0.08	5	0.18	22	0.19
280	0.103	3	0.04	3	0.05	3	0.14	13	0.10
400	0.084	2	0.03	2	0.04	2	0.13	8	0.06
プログラム		TPU (PROG interpreter)							
格納領域量	占有率	SMC		LLGC		Morris		Copy	
KB	平均値	回数	時間 (sec.)	回数	時間 (sec.)	回数	時間 (sec.)	回数	時間 (sec.)
48	0.642	3 + 63	1.04	3 + 63	1.20	66	1.23	–	–
64	0.471	2 + 30	0.50	2 + 30	0.57	32	0.67	–	–
120	0.252	12	0.19	2 + 10	0.24	11	0.30	65	0.60
200	0.153	6	0.10	6	0.13	6	0.24	25	0.21
280	0.105	4	0.05	4	0.07	4	0.18	15	0.11
400	0.075	2	0.03	2	0.05	2	0.13	10	0.08
プログラム		BitA-9 (compiler)							
格納領域量	占有率	SMC		LLGC		Morris		Copy	
KB	平均値	回数	時間 (sec.)	回数	時間 (sec.)	回数	時間 (sec.)	回数	時間 (sec.)
240	0.791	3 + 2	0.57	5 + 0	0.76	5	0.59	–	–
320	0.614	2 + 0	0.22	2 + 0	0.34	2	0.25	–	–
400	0.495	1 + 0	0.11	1 + 0	0.20	1	0.13	–	–
480	(0.404)	0	0.0	0	0.0	0	0.0	5	0.25
600	(0.312)	0	0.0	0	0.0	0	0.0	2	0.09
800	(0.250)	0	0.0	0	0.0	0	0.0	1	0.05
プログラム		変形 Tarai-4 (compiler)							
格納領域量	占有率	SMC		LLGC		Morris		Copy	
KB	平均値	回数	時間 (sec.)	回数	時間 (sec.)	回数	時間 (sec.)	回数	時間 (sec.)
16	0.147	2 + 66	0.043	1 + 67	0.086	68	0.186	168	0.126
32	0.071	31	0.020	31	0.042	31	0.131	67	0.050
64	0.034	14	0.010	14	0.019	14	0.102	30	0.016

(注) BitA-9 の括弧内の占有率は Copy 法による計測結果.

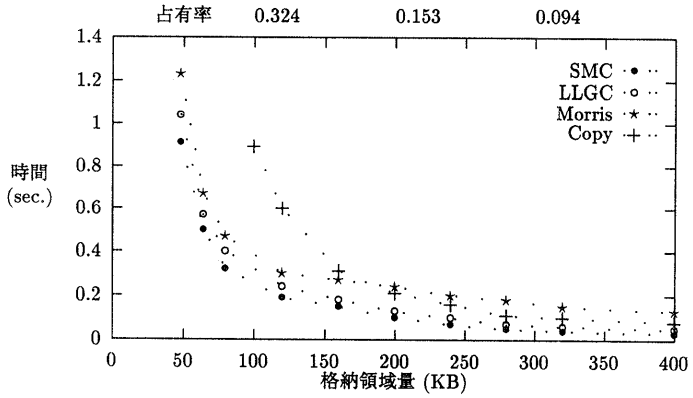
図4(b) は同じデータを用いて，GC 1回当たりの処理時間（縦軸）と占有率の逆数（横軸）でプロットしたものである．横軸の 1.0 以下は意味をもたない．表1からわかるように，格納領域量と占有率を掛け合わせた使用中セルの総容量（A）は測定区間を通じてほぼ一定（約 30 KB）である．したがって，処理時間が $O(A)$ である GC は水平の軌跡を描く．LLGC 法では 19 msec 付近に，SMC 法では 15 msec 付近に，複写型 GC では 7 msec 付近にほぼ水平にプロットされる．この差異は方式の違いによる単位容量（フィールド）当たりの作業量の差である．SMC 法が LLGC 法より優位にあることは明らかであるが，複写型 GC よりも1回当たり約2倍の処理時間を要する．複写型 GC の方が作業量が少ないためであるが，この差が GC 総時間での優位性につながらないのは GC 回数の比が3倍以上となるためである．

Morris 法の処理時間は A の他にセルの格納領域量（S）にも比例する．$S + A = A/\alpha + A$（A は定数，α は占有率）の恒等式から，図4(b) が示すように Morris 法は傾きのある直線となる．

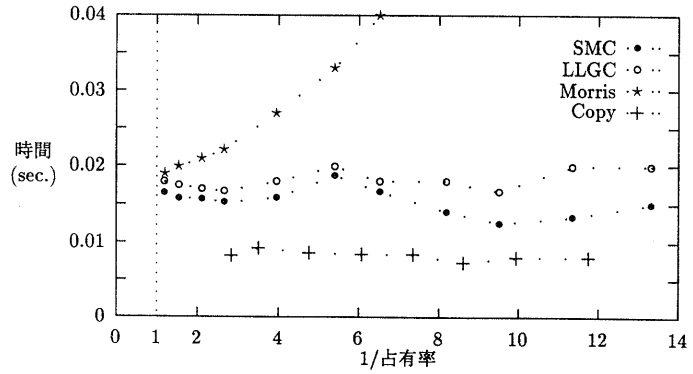
5.2 U-領域の効率化

図5 は2つの TPU を 48KB，変形 Tarai-4 を 16KB の格納領域量で実行したときの U-領域の使用状況を示したものである．図中では，確保された U-領域量（reserved）が実線で，その使用量（used）が黒丸で，最終的にソートの対象となった量（sorted）が白丸で表される．量はフィールドを単位にしているので，これらの量はアドレスの個数に相当する．グラフ外の数値はそれらの平均値と格納領域量に対する比率（括弧内）である．

三者ともソートの対象となるクラスタの数（N）は使用量（登録されたアドレスの個数）の 1/3～1/7 くらいである．これは後掛け順でも依然として登録の無駄があることを示している．しかし，U-領域量の設定



(a) GC 総時間



(b) 平均 GC 時間

図4 TPU の GC 処理時間
Fig. 4 GC time for TPU processing.

表2 GC 処理時間 (BitA-9 (240 KB))
Table 2 GC processing time.

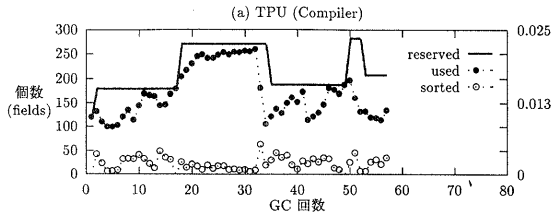
GC	SMC					LLGC		Morris	
	総時間	U-領域	使用量	ソート対象	使用中セル	総時間	使用中セル	総時間	使用中セル
回	sec.	K fields	K fields	K 個数	K fields	sec.	K fields	sec.	K fields
1	0.08	0.620	0.620	—	30.7 (0.496)	0.12	32.2 (0.528)	0.08	30.3 (0.501)
2	0.11	0.930	0.930	—	46.1 (0.745)	0.15	48.4 (0.782)	0.10	45.7 (0.752)
3	0.12	1.395	1.395	—	52.9 (0.856)	0.16	55.5 (0.880)	0.13	52.8 (0.873)
4	0.13	2.092	1.745	0.485	56.3 (0.910)	0.16	59.1 (0.941)	0.14	56.4 (0.937)
5	0.13	2.092	1.555	0.206	58.7 (0.948)	0.17	61.0 (0.989)	0.14	59.5 (0.984)
計	0.57	2.092	1.650	0.346	48.9 (0.791)	0.76	51.3 (0.830)	0.59	48.9 (0.801)

(注) 使用中セル量の括弧内は占有率．計の欄で処理時間以外は平均値（使用中セル量は 5 回の平均）．

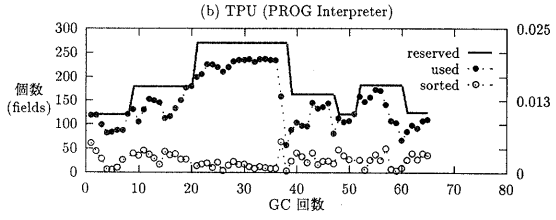
は効果的に行われて，領域の使用効率は 70～80%であり，オーバフローも数回以下という効率的な運用結果が示されている．さらに，A（使用中セルの総容量）に対する比率（N/A）は 1/300～1/120 であり，$N \ll A$ の成立が示されている．

5.3 ソート処理時間

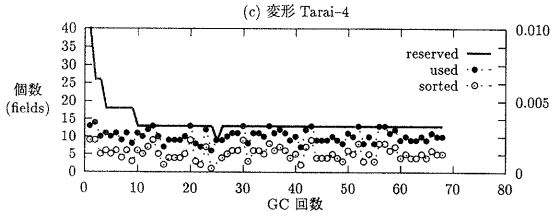
図 5 の三者について，Nの平均値は 25 以下である．また，U-領域内でそれらが部分的に整列しているのも実験データの収集から確かめられている．Nが小さい場合はソートの技法のいかんによらずその影響は少ない．そこで，挿入ソートなどの単純な方法で十分である．しかし，プログラムによってはNが大きくなることもある．**表 2** は前述の BitA-9 を 240KB の格納領域量で実行したときの SMC 法，LLGC 法と Morris 法の様子である．占有率（α）が非常に高い事例であるが，SMC 法の 5 回の作業のうち 2 回はソート処理



平均値 reserved: 218 (1.8%), used: 169 (1.4%), sorted: 24 (0.2%)
marked: 7680 (64%)



平均値 reserved: 190 (1.6%), used: 146 (1.2%), sorted: 25 (0.2%)
marked: 7689 (64%)



平均値 reserved: 14 (0.4%), used: 10 (0.3%), sorted: 5 (0.1%)
marked 600 (15%)

(注) reserved, used, sorted, markedはそれぞれ, U-領域の確保量, データ使用量, ソート対象量 (N), 使用中セル量 (A) を field 数で表す. 括弧内は格納領域量に対する比率.

図5 U-領域使用の変化
Fig. 5 Change of U-space size and its utilization.

が実行される. Nは485と206である. N/Aの比率はそれぞれおよそ, 1/100と1/300である. ソートの回数は多くなるが, Morris法と比較してもソート処理の負担増は陽に表れない. また, LLGC法はU-領域が不足するためソート処理が行われず, その処理はMorris法より遅くなる. なお, 複写型GCは, この格納領域量では領域不足のため実行不能である. このように, 占有率が高いときでもSMC法は効果的に動作することがわかる.

6. まとめ

本論文で提案したSMC法は圧縮型GCの懸案であった処理時間の大幅な削減を実現した. それはSMC法が従来の圧縮型GCや複写型GCよりもGCに要する総時間が短いことを実験結果から示した. また, 同様の効果が期待できるLLGC法よりも, U-領域の節約や処理時間の短縮化などの利点があることも示した.

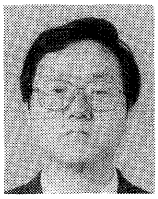
Morris法では, CONSデータやシンボルなどのデータが共有される場合, ポインタ補正は定数時間にならないことが知られているが, 今回の実験から言えることはそれが顕著に現れていないことである.

参考文献

1) Carlsson, S. et al.: A Fast Expected-Time Compacting Garbage-Collection Algorithm, Dept. of Computer Science, Lund University, Sweden (1990) (*ECOOP/OOPSLA '90 Workshop on Garbage Collection in Object-Oriented Systems* に掲載).
2) Sahlin, D.: Making Garbage Collection Independent of the Amount of Garbage, SICS Research Report R86008, Sweden (1987).
3) Suzuki, M. and Terashima, M.: Time- and Space-Efficient Garbage Collection Based on Sliding Compaction, Graduate School of Information Science Technical Report, University of Electro-Communications (1993) (情報処理学会論文誌, Vol.36, No.4, pp.925–931 (1995) に掲載).
4) Terashima, M. and Goto, E.: Genetic Order and Compactifying Garbage Collectors, *Information Processing Letters*, Vol.7, No.1, pp.27–32 (1978).
5) Morris, F.L.: Time- and Space-Efficient Garbage Collection Algorithm, *Comm. ACM*, Vol.21, No.8, pp.662–665 (1978).
6) Terashima, M. and Kanada, Y.: HLisp−Its Concept, Implementation and Applications, *J. of Inf. Process.*, Vol.13, No.3, pp.265–275 (1990).
7) Fenichel, R. and Yochelson, J.: A Lisp Garbage Collector for Virtual-Memory Computer Systems, *Comm. ACM*, Vol.12, No.11, pp.611–612 (1969).
8) Appleby, K. et al.: Garbage Collection for Prolog Based on WAM, *Comm. ACM*, Vol.31, No.6, pp.719–741 (1988).
9) Liberman, H. and Hewitt, C.: A Real-Time Garbage Collector Based on the Lifetimes of Objects, *Comm. ACM*, Vol.26, No.6, pp.419–429 (1983).
10) 小出 洋, 鈴木 貢, 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, 情報処理学会論文誌, Vol.35, No.11, pp.2529–2532 (1994).
11) Steele, G.L. Jr.: *Common LISP*, 2nd ed., Digital Press, Mass. (1990).
12) Chang, C.L.: The Unit Proof and The Input Proof in Theorem Proving, *Journal of ACM*, Vol.17, No.4, pp.698–707 (1970).
13) 竹内郁雄: LISP処理系コンテストの結果, 情報

処理学会記号処理研究会資料，SYM 5-3 (1978).

(平成6年12月19日受付)
(平成7年 9 月 6 日採録)

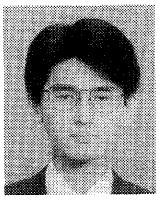


石田　満

1955年生．1982年東京学芸大学大学院教育学研究科修士課程修了．同年神奈川県相模原市立中央中学校勤務．1992年電気通信大学情報工学科卒業．同年東京都立小金井工業高等学校（定）勤務．現在電気通信大学大学院情報システム学研究科博士後期課程在学中．記号処理系の逐次型および並列型の記憶管理アルゴリズムに興味を持つ．

寺島　元章（正会員）

1948年生．1973年東京大学理学部物理学科卒業．1975年同大学院修士課程，1978年同博士課程修了．理学博士．1978年より電気通信大学計算機科学科勤務．現在，同大学院情報システム学研究科助教授．プログラミング言語とその処理系，記憶管理方式，記号数式処理系などに興味をもつ．AAAI，ACMの各会員．



笹原　豪士

1972年生．1994年東京工科大学工学部情報工学科卒業．現在，電気通信大学大学院情報システム学研究科博士前期課程在学中．記号処理系における並列アルゴリズムに興味を持つ．