ICOT Technical Report: TR-213

TR-213

PSIのガーベッジコレクタ

西川　宏（松下電器産業）
池田守宏（三菱電機）

October, 1986

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

PSI のガーベッジコレクタ

西川 宏
(松下技研(株))

池田守宏
(三菱電機)

第5 世代コンピュータ研究開発プロジェクトでは，その一環としてパーソナル逐次型推論マシンPSI の開発を行なってきた．PSI 上で実行される言語は論理型言語であり，Lisp同様プログラムの実行にともない動的に構造体データが生成され，ガーベッジコレクションの機能は必須となる．この論理型言語の実行には各種スタックとヒープ領域がPSI では用いられるが，ガーベッジコレクションの対象となるものは，Lisp 等とは異なり，スタック領域も対象となる．さらに，多重プロセス環境がPSI ではサポートされており，より複雑なガーベッジコレクションを行なわなくてはならない．本論文では，PSI のメモリ管理の観点から何故スライディングコンパクション方式を採用したかを述べ，セルのマーキング手順とコンパクションアルゴリズムを示す．最後にその評価結果を示す．

1. はじめに

知識情報処理の分野が最近注目を集めており，これに伴い，Prologに代表される論理型言語が広く用いられるようになってきた．このような背景の下，第 5世代コンピュータ研究開発プロジェクトでは，論理型言語KL0[1]を機械語として，これを直接解釈／実行する逐次型推論マシンPSI を開発した[2,3,4]．

論理型言語は，関数型言語Lispと同様，動的に構造体データを生成してゆくことから，ガーベッジコレクションの機能(ガーベッジコレクタ) を備えることは実用上必須となる．このガーベッジコレクタの実装方式は言語仕様，その言語のインプリメンテーション方式，及び，その言語が実行されるマシンの性格に大きく依存する．

例えば，PSI の場合は次のようである．

論理型言語KL0 の言語仕様は，DEC-10 Prolog に基づいており，ユニフィケーションにより構造体データがグローバルスタックの上に生成される．さらに，実用的問題に適用するためにサイドエフェクトを残すような構造体データも取り扱えるよう機能拡張されている．この構造体データはグローバルスタックとは別のヒープと呼ばれる領域に組込述語により生成される．構造体データが生成される領域はガーベッジコレクション(GC)の対象となるので，PSI ではスタックとヒープという２種の領域のGCが必要となる．さらに，PSI では多重プロセッシング機能を実現しているために多重プロセス空間でのGCが必要となり，これまでのProlog処理系で実現されたもの[5] より複雑となる．

PSI ではこのようなガーベッジコレクション機能をファームウェアレベルで高速にサポートしていることが特徴である．

本論文ではPSI に実装されたガーベッジコレクタの処理方式とその実現手法について論ずる．以下，2章においてGCの代表的な手法とその特徴を述べ，3章ではPSI で採用したGC処理方式について論じ，4章ではそのインプリメンテーションについて詳しく述べる．最後に 5章では評価結果を述べる．

2. GCの処理方式

GCの処理にはゴミとなったデータと使用中のデータとの区分け(マーキング) と，ゴミとなったデータが占めていた領域の回収という２つの処理が必要である[6,7]．GC におけるメモリ管理のための最小単位を以後セルと呼ぶが，通常”セル”＝”語”である．

GCの代表的な手法には次の４つがある．

(1) マーク＋スイープ

これはゴミでないメモリセルのマーキングをおこない，次に回収フェーズではメモリ空間を片方からスイープしつつ，ゴミセルをフリーリストに連結していく．

(2) マーク＋コンパクション

メモリセルのマーキング方法は(1) とおなじであるが，回収フェーズではゴミでないセルはメモリ空間の一端に詰めあわされる．この方式ではセルの移動をプログラムの実行中に行うことが困難であり，通常，プログラムの実行が一時中断される一括型GCとして実現される．

(3) コピー方式[8]

これはメモリ空間を 2分割し，使用中のセルについては新しい空間へコピーしなおすことでGCを行なう．この方法の特徴は区分けと回収の処理が一体化している点である．またこの方式は一括型として実現することもできるが，比較的容易にプログラムの実行と並行してGCを行うようなリアルタイムGCとすることも可能である[9]．

(4) 参照カウンタ方式[10,11]

すべての構造体データに参照カウンタを設け，参照数のメンテナンスをプログラムの実行中に行ない，その数が"0" になったことで，その構造体データをゴミと認識する．この方式は構造体データが参照される度に参照数の増減を行っているので，マーキング処理はプログラムの実行中に分散されていることになる．通常ゴミとなったセルはフリーリストにつながれるので，この方法ではリアルタイムGCが実現できる．

以上の4 つの処理方式を比較するために，まずマーキングのために必要な情報量について考えてみる．

(1),(2) の方式では目印をつけるためにセル毎に 1ビットのマーク情報が必要である．ネストした構造体データのマーキングを行なうためには，スタックを用いてネストをたどる方法[12]とポインタ反転を用いる方法[13]がある．スタックを用いるマーキングでは，スタックのための領域が必要であり，ポインタ反転を用いる方法ではセルに反転ポインタが格納されているかを識別するためにもう 1ビット，計 2ビットの付加情報が必要である．

(3) の方式ではメモリ空間を2 分割するので，メモリのセル毎に作業領域を 1セル用意したことになる．つまり，コピー方式では使用できるメモリ量が実質的に半分になってしまう欠点がある．

(4) の方式では構造体データごとに参照カウンタの領域が必要であるが，リアルタイムGCが実現できるという点で魅力がある．ただしループ構造を持ったセルが回収できないこと；さらにプログラムの実行速度が全体に遅くなるという欠点がある．

更にメモリセルの回収方式について考えると，(1),(4) の方式では，空き領域がメモリ空間中に分散してしまう．また(3) の方式では新しい空間へ移動されたセルの内容が元の空間の中の構造体を指している場合，その構造体が順次新しい空間へ移動されるので，コピー先の空間でのセル並びは元の空間の並びと一致しない．したがって，スタックのようなデータの並び順が問題になるような領域のGCには適用できない．

3. PSI のGCシステム

3.1 PSI のメモリ管理

KL0 の解釈／実行のためには他のPrologシステムと同様に，複数のスタックとプログラムを格納するためのヒープ領域が必要である．スタックの種別はPSI の場合グローバルスタック，ローカルスタック，コントロールスタック，トレールスタックの4 本である．これらのスタック，ヒープはプログラムの実行にともなって任意に伸長されるため固定領域に割りあてることができない．更にPSI では多重プロセスを実現しているために，これらスタック群がさらにプロセスの数だけ必要となる．

これらのスタック群とヒープ領域を効率良く実現するためにPSI ではエリアという概念を導入した．このエリアを実現するために32ビットあるPSI のアドレスの上位 8ビットはエリア番号として使用し， 256枚の独立した論理空間に分割した．各エリアは独立に16M語まで任意に伸長可能であり，スタック，ヒープはエリア毎に割りあてられる．

ヒープ領域をすべてのプロセスで共有し 1枚のエリアに割り与えると 256枚のエリアからは最大63個のプロセスが生成できる(図 1参照)．

論理空間への物理ページの割りあては，スタック，ヒープの伸長にともなってページ単位(1K 語) に行われる．この物理ページの最大容量，つまり，PSI の最大実装メモリ容量は16M セル(80Mバイト) であり，通常のプログラムの実行には十分である．この判断からPSI では 2次記憶を使用した仮想空間はサポートしていない．したがって全エリアのページ数の総和は実装物理メモリ容量で決定される．

3.2 PSIにおけるGC方式

PSI でGCの対象となるエリアはスタックとヒープである．特にスタックについては使用中のセルの並びを

保存する必要があり，スライディングコンパクションを行なわなければならない．また二次記憶を利用していないPSI では，伸長したエリアの使用領域をコンパクションすることで，未使用になった自由ページを作りださねばならない．

これら 2つの理由からPSI のGCには一括型のマーク＋スライディングコンパクション方式を採用した．マーキング処理とコンパクション処理の高速化を図るため，メモリセル毎に 2ビットのGC用のタグを設け，ハードウェアで高速に判定できるようにした．さらに，GC用のプロセスを特別に設けることで，きめ細かいGCが行えるようにした．

3.3 GCプロセス

GCプロセスとは自由物理ページがない状態で，物理ページ割り当て要求が発生した際に起動されるソフトウェアレベルのプロセスである．その中では以下のようにページ単位とセル単位の二段階GCの処理が行なわれる．

* ページ単位GC

スタックとして使用されているエリアの中にはスタックが縮んだことによって未使用になったページが存在する可能性がある．この未使用ページの回収をプロセス毎に行ない，対応する物理ページを実ページ管理スタックに返却する．この操作によって必要とされるページ数が回収できればセル単位のGCは行わない．

* セル単位GC

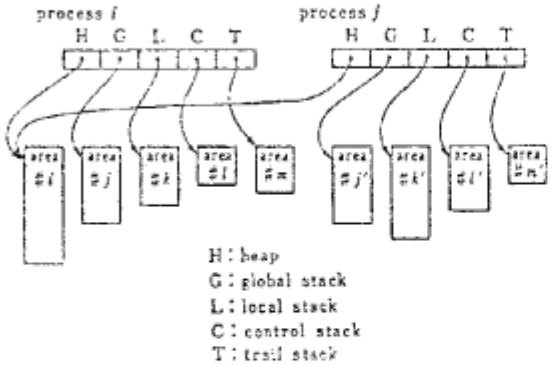


図 1 プロセスとエリアの関係

それでも充分な実ページが回収できなければセル単位のガーベッジコレクションをおこなうために，組込述語collect-garbage(ProcessTable) を実行する．この組込述語によってマイクロプログラムによって作られたGCルーチンが起動される．まず引数ProcessTableによってGCすることを指定されたプロセスについてセルのマーキングをプロセス毎に逐次行なう．次にマーキングを行なったエリアに関してコンパクションを行なう．

このようにマイクロインタプリタから直接GCルーチンを起動せずソフトウェアを介することにより柔軟性をうることができ，例えばProcessTableの与え方によりプロセス単位での部分GCも可能であり，あるプロセスのGCのみをおこなうことで，処理の中断時間を少なくすることが可能である．

4. collect-garbage のインプリメンテーション

組込述語collect-garbage で起動されるGCルーチンのマイクロコードの容量はマーキング処理に約 1.0K ステップ，コンパクション処理に約 0.5K ステップを要し，全体で 1.5K ステップである．以下に各フェーズでの処理を詳細に述べる．

4.1 マーキングフェーズ

マーキングはプロセス単位に実行される．マーキングを行なうためには

* ルートとなるセルの検出方法
* 構造性データのマーキング方法

が重要である．

(1) ルートの検出について

KL0 の実行に必要な各エリアには次のようなデータが格納されている．

* ローカル／グローバルスタック

クローズ内に出現したローカル変数，グローバル変数がそれぞれフレームの形にまとめられて格納される．

* トレールスタック

バックトラック時にリセットすべきセルアドレスが格納されている．

* コントロールスタック

クローズの実行順序を制御する情報がフレーム単位に格納されている．このフレームは述語呼び出しに対応して生成され，呼び出されたクローズのローカル，グローバルフレームへのベースアドレス，及びこのクロ

ーズの呼び出し元コントロールフレームのアドレス(P-link)，バックトラック時に起動されるコントロールフレームのアドレス(B-link)等が格納されている．最新のP-link,B-link はレジスタPCBR,BCBR に保持される(図-2参照)．

* ヒープエリア

プログラムのオブジェクトコードのほか，組込述語によって作られるベクタ等の構造体データが格納されている．

一方エリア間でのデータの参照方向はつぎのように規定されている：

コントロールスタック
↓
トレールスタック
↓
ローカルスタック
↓
グローバルスタック
↓
ヒープエリア

したがってマーキングのルートとしてコントロールスタックに格納されている情報を使えば各種スタックおよびヒープに格納されたゴミでないすべてのセルをマークすることができる．

ただしKL0 ではバックトラックポイントを変更するremote-cutの実行によってはゴミフレームがコントロールスタック自身にも生成される場合がある．図-2を例にとると，レジスタPCBRで指示されるフレーム内のP-linkを 1段たどったフレーム中のB-linkでさされるフレームをあらたなバックトラックフレームにするremote-cut命令が実行されると，レジスタBCBRは，(1)をしめすように変更される．斜線が施されたフレームはこの操作により以降の実行に不用となったフレームである．PCRRより下方のものはポップアップされるが，上方のものはコントロールスタックに残りゴミとなる．

ゴミでないコントロールフレームは，したがって，いま実行途中であるクローズの実行終了後に起動される(レジスタPCBRでさされる) フレーム，および実行に失敗してバックトラック時に起動されるフレーム(レジスタBCBRでさされる) の 2つを起点とし，そのフレームに格納されたP-linkあるいはB-linkを通して指されるコントロールフレームの集合として求めることができる．これを行なうために図-3に示すアルゴリズムでフレームのピックアップを行なっている．この方法を用いれば，ゴミでないコントロールフレームからリターンリンクおよびバックトラックリンクの 2つから指されないコントロールフレームはピックアップされない．

(2) 構造体データのマーキング

PSI で生成される構造体データには 2種のものがある：1つはマイクロインタプリタが動的につくりだすローカル，グローバルフレームであり，他方はユーザが直接つくるベクタ，ストリング等である．これらの構造体データは常に”かたまり”として取扱う必要がある．すなわちその構造の一要素でも参照されているとその全体をマークしなければならず，そのサイズを知る必要がある．以下に主なものについてサイズ情報の求め方を述べる．

* ローカルフレーム

ベースアドレスはコントロールフレームに格納されている．そのサイズは 1つ次のコントロールフレームが保持するローカルベースとの差として計算される．

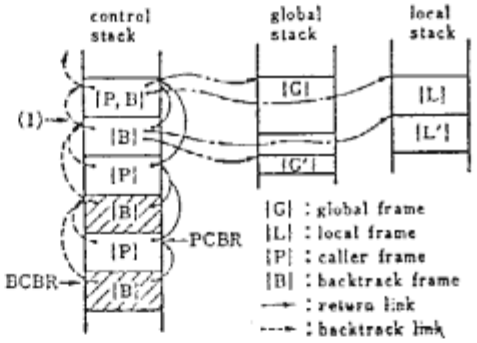


図 2 コントロールスタックとコントロールフレーム

```
struct control_frame *P_link, *B_link;

P_link = PCBR;
B_link = BCBR;

while (P_link != 0 && B_link != 0) {
    if (P_link >= B_link) {
        mark_cells_with(P_link);
        P_link = P_link -> parent;
    } else {
        mark_cells_with(B_link);
        P_link = P_link -> parent;
        B_link = B_link -> backtrack;
    }
}
```

図 3 コントロールスタックのトラバースアルゴリズム

* グローバルフレーム

グローバルスタック中には，グローバルフレーム以外の情報も生成されるので，ローカルフレームと同様な方法でサイズを決定できない．したがってフレームの先頭に，すなわち，第 0要素にサイズ情報を格納している．

* ベクタ ,ストリング，コンパイルドコード

これらのデータはKLO の内部表現として常にデータ記述子を介してアクセスされる．データ記述子にはサイズをしめすフィールドがあるので，ここからサイズを求めることができる．

ネストした構造体データのマーキングを行なう方法には，

(1) マークすべき残りの構造のアドレス(戻りアドレス) とその要素数をスタックに積む方法[12]

(2) 構造体要素として出現した(ネストした) 構造体データへのポインタを逆向きにして戻りアドレスを記憶するポインタ反転方法[13]

の 2つがある．

スタックを用いる方法は単純で高速であるが，ネストの深さ分のスタック領域が必要である．一方ポインタ反転法はガーベッジコレクションの際に新たな作業領域は不要であるが反転したポインタを元に戻すための操作が必要であり，スタックを用いる方法より低速である．

またポインタ反転法ではどこまでマークするかをしめすタグ(ENDタグ) を構造体データの先頭要素につける必要がある．すなわちこの方法では，構造体データの最終要素から逐次要素セルをマークしてゆき，ENDタグのついたセルのマークが終れば構造体データのマーキングが完了することになる(図-4参照) ．この方式が可能であるためには，構造体データの途中にENDタグがつけられた要素セルがあってはならない．

KLO ではベクタの一部分を共有することができるので，図-5にしめすように，一部分がオーバラップしたベクタA,B をつくることができる．ベクタA の要素としてベクタB があれば，ベクタB のマーキングをおこなっている途中にベクタA のEND タグが検出され，ベクタB のEND タグを正しく認識することができない．

したがってPSI ではマーキングアルゴリズムとしてスタックを用いる方法を採用する事を基本とした．しかしコンパイルドコードを格納した構造体データには，一部を共有するものがないことと，この構造体データのネストが深いことからポインタ反転方式を用い，2つの方法を併用したマーキング方式とした．

4.2 コンパクションフェーズ

コンパクションフェーズではゴミでないセルをアドレスに関する順序関係を保ちつつ各エリアのゼロアドレス側に詰め合わす作業を行う．

コンパクション後には各エリアの使用領域が縮むことで，以前に占有されていた論理ページに対応する実ページの解放が可能となる．この回収された実ページをGCを引き起したプロセスに割り当てることで通常の処理が再開される．

(1) スタックのコンパクションについて

スタックのコンパクションは各セルのアドレスの大小関係を保存したままでおこなう必要がある．これを効率良く(スタックの大きさに比例した手間で) おこなうためにPSI ではHorrisの方法[14]をベースとしたコンパクションアルゴリズムを用いている．

Morrisのアルゴリズムはコンパクションする領域を2回スイープしてコンパクションをおこなうものである．この方法ではマーキングが終了した時点で，各セルにはマークがついており，かつコンパクションを行

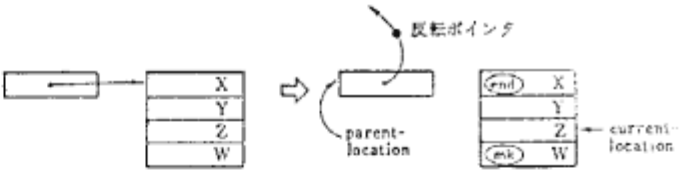


図 4 反転ポインタを用いたマーキング

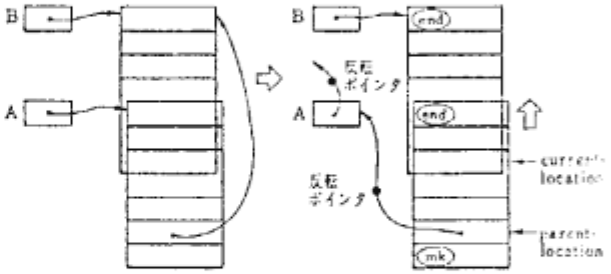
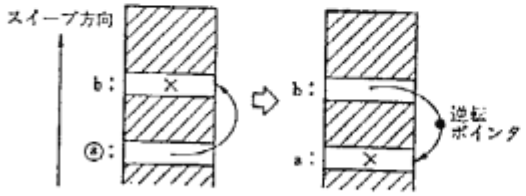


図 5 反転ポインタが使用できない例

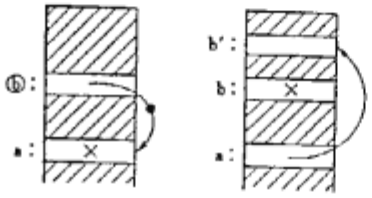
おうとする領域のゴミセル数が前もってわかっていることを前提としている

1回目のスイープはセルの詰め合せをおこなう向きと同一の方向におこなう．PSI では領域のゼロアドレス側にデータをコンパクションするため，まず上向きスイープがなされる(図-6参照) ．

スイープではマークがついたセルを捜していく．また，マークの付いていないセルを検出するたびに，ゴミセル数のカウントをおこなう．これは未スイープ領域にある残りのゴミセルの数(すなわちこれが移動量となる) を知るためであり，ポインタの値をコンパクション後の値につけなおす際に利用される．

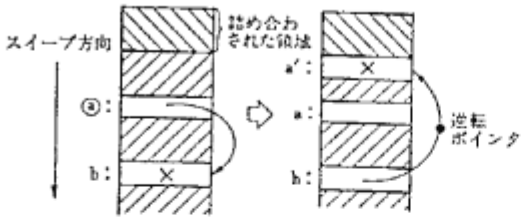


(a) 上向きポインタの処理

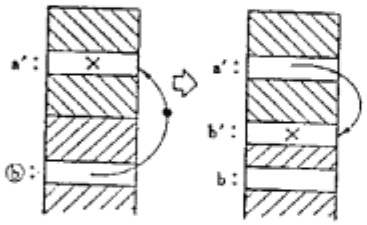


(b) 逆転ポインタの処理

図 6 上向きスイープの処理



(a) 下向きポインタの処理



(b) 逆転ポインタの処理と詰め合わせ

図 7 下向きスイープの処理

マークがついたセルの内容がスイープ方向と同じ向きのポインタ(上向きポインタ) であれば参照先セルとの間に逆転ポインタを生成する．つまりセルa には参照先セルb の内容X を移動し，セルb には逆転ポインタを格納することがおこなわれる(図-6(a)) ．

逆転ポインタが格納されたセルb を検出したときは，参照先セルa に退避されていた内容X をセルb にもどし，セルa にはセルb の移動先アドレスb'を計算して格納する(図-6(b)) ．これはセルb のアドレスから未スイープ領域にあるゴミセルの数を減じることで求められる．

このようにして領域の先頭にくると今度は逆向きのスイープ(下向きスイープ) をおこなう．このときに下向きポインタの処理とマークのついたセルの詰め合せがなされる(図-7参照) ．

まずマークのついていないセルをスキップしながらマークの付いたセルをゼロアドレス側へ移動する．

セルa の内容が下向きポインタであった場合，参照先セルの内容X はコンパクション後のアドレスa'に移動される．セルb には，セルa'への逆転ポインタが生成される(図-7(a)) ．セルa の内容が下向きポインタでなければ，それをアドレスa'に移動することのみが行われる．

逆転ポインタが格納されたセルb には次の操作がなされる．まずセルb の移動先アドレスb'にセルa'の内容X を戻し，次にセルa'にはセルb'へのポインタを格納する(図-7(b)) ．これでセルa,b はコンパクションされてセルa',b' に移動されたことになる．

領域の終りまでスイープをするとセルの詰め合せが終了する．

(2) 多重空間に対するコンパクションアルゴリズム

領域を逐次スイープしてコンパクションをおこなうHorrisの方法はコンパクションする領域は連続でなければならないという制約がある．

一方PSI のメモリ空間では32ビットのアドレスは 256枚のエリアに分割されている．さらにエリア間にまたがるポインタも存在するので，エリア単独ではコンパクションすることが不可能であり，Horrisの方法をそのまま適応できない．

PSI ではコンパクションすべき領域をエリア番号込みの32ビットアドレスから構成される単一不連続領域とみなし，セルの移動量は個々のエリア内ゴミ数に基

づいて算出することで，エリア単位のコンパクションをおこなうようにした．

具体的には，エリア番号の大きいエリアからエリア1までを，順次エリア単位にその使用領域についてスイープして上向きポインタの付けかえ処理を行なう．次にエリア 1から下向きポインタの処理と詰め合せをエリア単位におこなう．(図-8参照)

(3) コンパクション処理の高速化について

コンパクションを高速におこなうために以下のような点を考慮した手法を導入した．

トレールスタックにはゴミセルがないので詰合せをおこなう必要がない．しかしながらトレールスタックから他のスタックへの参照ポインタは，参照先セルの移動に応じて変更する必要があり，一般には 2回のスイープが必要となる．しかしトレールスタックは各プロセス内で最も大きなエリア番号を与えられるので，ポインタの向きはすべて上向きである．このことから1回目のスイープによって参照ポインタのアドレス変更が完了するので 2回目のスイープ時にはトレールスタックとして使用されているエリアを省くことができる．

5. 評価

図-9にGCに要する時間と実装メモリ容量との関係を示す．同一実装メモリ容量でもGCの実行時間がばらつくのは，そのときマークされたセルの数と複雑なデータ構造がどれだけ生成されたかに依存する．しかしながら実装メモリ容量にほぼ比例した時間がGCに要するのがわかる．

この例ではPSI のOSであるSIMPOS上のデバッガでインタプリティブコードを実行したためにモレキュールが多数生成され，このマーキングに時間がかかるため，メモリ容量に対する実行時間の増分がかなり急勾配になっている．

また図 -10にIPL 後の最初のGCで連続してマークされたブロックの大きさの分布と，2 回目のGCでのブロックの分布をしめす．これによると，1 回目では，小さなブロックが多数存在するが，2 回目では，小さなブロック同志が，1 回目のGCでより大きなブロックにまとめられたことがわかる．実際ヒープ領域について

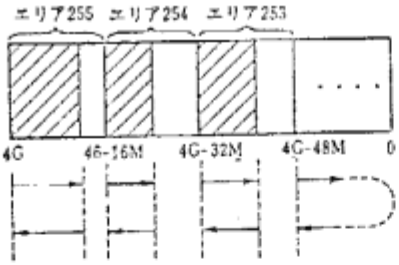


図 8 エリアのスイープ手順

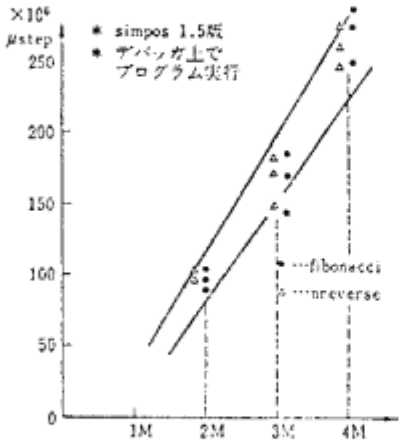
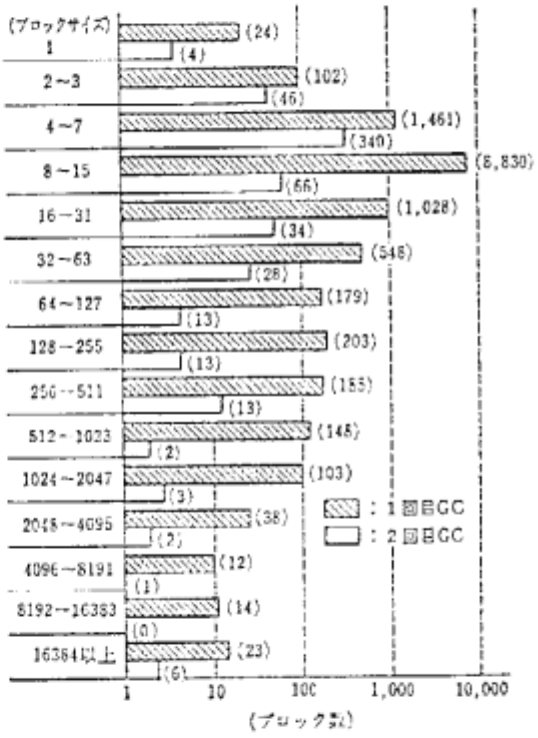
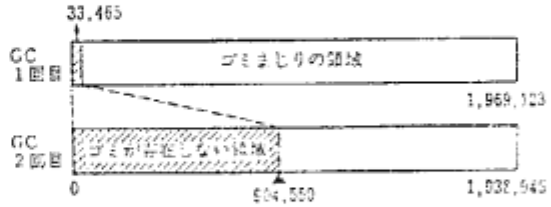


図 9 実装メモリ容量と GC に要するマイクロステップ数の関係



(a) ブロックサイズとブロック数の分布



(b) ゴミまじり領域の分布

図 10 ヒープエリアのマーク状態

は，2 回目以降では，約1Mセルの連続領域がゴミ無しになっている．これはプログラムコードがエリアのゼロアドレス側に押しやられたためである．

このように本方式のGCではゴミでないセルが各エリアのゼロアドレス側に沈澱する傾向がある．従って各エリア内でGCを行う領域と行う必要がない領域とを区別することができれば，より一層のGCの速度改善がはかれると思われる

6. まとめ

PSI に実装されたGCは，マルチプロセスを対象として，それらが使用しているエリアをスライディングコンパクションする一括型GCである．マーキングアルゴリズムには，スタックを用いる方法とポインタ反転法とを併用しており，コンパクションアルゴリズムではHorrisの方法をベースにしたものを用いている．

GCルーチンはマイクロプログラムで実現されており，16M 語(80Mバイト) の最大実装時でGCに要する時間はおよそ10分である．

PSI のGCシステムはソフトウェアで管理される方式を採用しておりこれにより柔軟なGCシステムが実現されている．例えばマーキングの際にスタックがオーバフローした場合には，GCルーチンは例外を発生してオーバフローを起こしたプロセス番号がソフトウェアに通知される．このようにGCができなかったからといって，PSI システム全体が倒れることをふせいでいる．またプロセス単位のGCがおこなえるので，GCがおこなわれているプロセスをバックグランドで実行し，フォアグランドでは通常の処理をおこなうことも可能ではある．

謝辞

GCの作成にあたり貴重な助言をいただいた近山隆氏またGCのコーディングとデバッグに協力いただいたシステムズデザイン(株) の久津間正勝氏，本研究をする機会を与えてくださったICOT第 4研究室長内田俊一氏，本論文に関する有意義なコメントをいただいた横田実氏に感謝します．

参考文献

[1]Chikayama,T.et al:Fifth Generation Kernel Language.Proc.of Logic Programming Conference'83 ,Japan ,1983.

[2]Nishikawa,H.et al:The Personal Sequential Inference Machine(PSI) - Its Design and Machine Architecture,Proc.of Logic Programming Workshop,Algrave/PORTUGAL,June,1983,pp.53-73.

[3]Yokota,M et al:A Miroprogrammed Interpreter for The Personal Sequential Inference Machine, Proc.of FGCS'84 ,Japan,1984,pp.410-418.

[4]Taki,K et al:Hardware Design and Implementation of The Personal Sequential Inference Machine (PSI), ibid,pp.398-409.

[5]Warren,D.H.D:Implementing Prolog-Compiling Predicate Logic Program Vol.1-2,D.A.I.Research Report,No.39-40,Dept.of A.I.,Univ.of Edinburgh , 1977.

[6]Cohen,J:Garbage collection of Linked Data Structures,ACM computing survey 13,3 sept.1981 , pp.341-367

[7] 日比野靖: ガーベジコレクションとそのハードウェア，情報処理,23,8,1982,pp.730-741.

[8]Minsky,M.L.: A Lisp garbage collectr or algorithm using serial secondary memory storage,Memo 58(rev.),Project MAC,MIT,Cambrige,Mass.,1963.

[9]Baker,H.G:List Processing in Real Time on a Serial Computer,Comm.of the ACM,21,4,1978, pp.280-294.

[10]Collins,G.E.:A Method for Overlapping and Erasure of Lists,Comm.of the ACM,3,12,1960,pp.655-667.

[11]Deutsch,L.P.and Bobrow,D.G.:An Efficient Incremental Automatic Garbage Collector,Comm.of the ACM,19,9,1976,pp.522-526.

[12]Knuth,D.:The Art of Computer Programming,Addison-Wesley,Reading,Mass.,1973

[13]Schorr,H.and Wait,W.:An efficient machine-independent procedure for Garbage collection in Various list structure,Comm.of the ACM,10,8,1967,pp.501-506.

[14]Moris,F.L.:A time- and Space- Efficient Garbage Compaction Algorithm,Comm.of the ACM,21, 8,1978,pp.662-665.