PSIのガーベッジコレクタ†

西川 宏†† 池田守宏†††

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 を開発してきた．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伴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，PSI では各種スタックとヒープを用いるが，ガーベッジコレクションの対象となる領域は，Lisp 等とは異なり，スタックとして使用された領域も対象になる．さらに，PSI では多重プロセス環境がサポートされており，ガーベッジコレクタは今までのものより複雑となる．本論文では，PSI のメモリ管理機構の観点から，なぜ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る．次に，PSI で行っているセルのマーキング手順とコンパクションのアルゴリズムを示す．最後にガーベッジコレクタの実行時間等の評価結果を示す．

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)~4)]．

論理型言語の処理システムは，関数型言語 Lisp と同様，動的に構造体データを生成するため，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，および，その言語が実行されるマシンの性格に大きく依存する．

以下，PSI の場合について述べる．

論理型言語 KL0 の言語仕様および処理方式は，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マーキング（区分け）と，ゴミとなったデータが占めていた領域の回収という二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，PSI においては“セル”＝“語”（5バイト）である．

GC の代表的な手法には次の四つがある．

（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行い，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)と同じであるが，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合わされる．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)]

† Garbage Collector on PSI by HIROSHI NISHIKAWA (Matsushita Research Institute Tokyo) and MORIHIRO IKEDA (Mitsubishi Electric Co., Ltd.).

†† 松下技研(株)
††† 三菱電機(株)

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 を行う．この方法の特徴はマーキング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たび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．ゴミとなったセルをフリーリストにつなぐ方式をとれば，この方法ではリアルタイム GC が実現できる．

以上の四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ごとに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，メモリのセルごとに作業領域を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空間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．

さら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伴って任意に伸長されるため固定領域に割り当てることができない．さらに 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 16 M 語まで任意に伸長可能であり，スタック，ヒープはエリアごとに割り当てられる．

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当ては，スタック，ヒープの伸長に伴ってページ単位（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 16 M セル（80 M 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

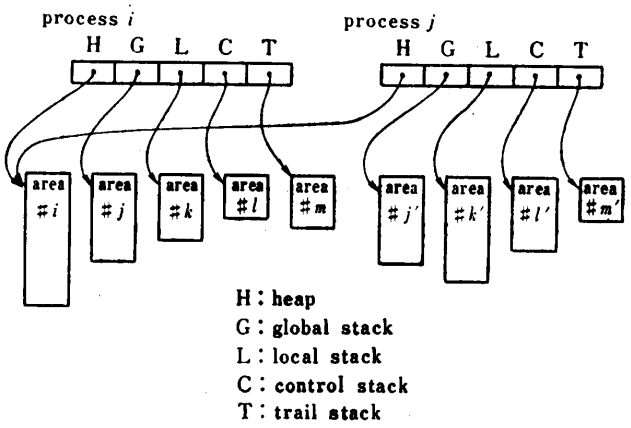


図1 プロセスとエリアの関係
Fig. 1 Process and area.

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圧縮することで，未使用になった自由ページを作りださねばならない．

これら二つの理由から 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，メモリセルごとに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の GC 処理が行われる．

＊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ごとに行い，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

それでも十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行うために，組込述語 collect-garbage(ProcessTable)を実行する．この組込述語の呼び出しにより，マイクロコードで記述された 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ごとに逐次行う．次にマーキングを行ったエリアに関してコンパクションを行う．

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得ることができ，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 のみを行うことで，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 1.0 K ステップ，コンパクション処理に約 0.5 K ステップを要し，全体で 1.5 K 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．マーキングを行うためには

＊ルートとなるセルの検出方法

＊構造体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（一種のベクタ）にまとめられて格納される．フレームベースに変数番号を加えることで，変数は参照される．

＊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，およ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：

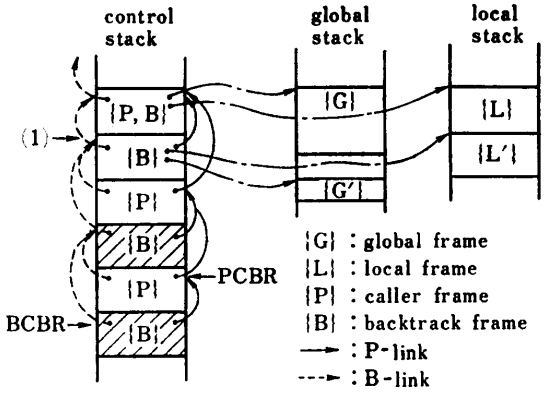
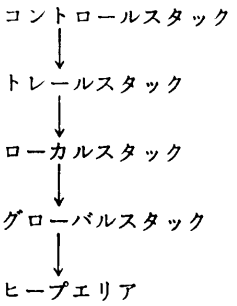


図 2 コントロールスタックとコントロールフレーム
Fig. 2 Control stack and control frame.



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 で指されるフレームを新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)を示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. PCBR 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 で指される) 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 で指される) の二つを起点とし, 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っている.

```
struct controlframe *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=B_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
Fig. 3 Traversing algorithm of control stack.

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二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. つまり, コントロールスタックのゴミフレームはピックアップされない. したがって, ゴミとなったコントロールフレームからのみ直接あるいは間接に参照されているセルの回収が可能となる.

(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: 一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" として取り扱う必要がある. 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. そのサイズは一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.

*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

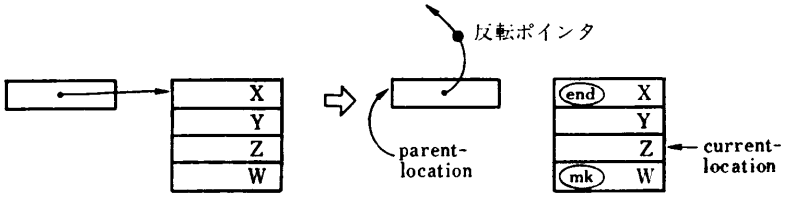


図 4 反転ポインタを用いたマーキング
Fig. 4 Marking with reversed pointer.

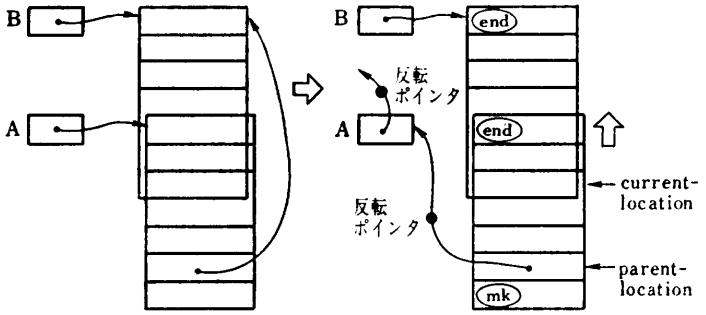


図 5 反転ポインタが使用できない例
Fig. 5 An example not marked correctly with reversed pointer.

KL0 ではベクタの一部分を共有することができるので，図 5 に示すように，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 のマーキングを行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ことを基本とした．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，二つの方法を併用したマーキング方式とした．

はサイズを示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]

の二つがある．

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示すタグ（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わ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．

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．

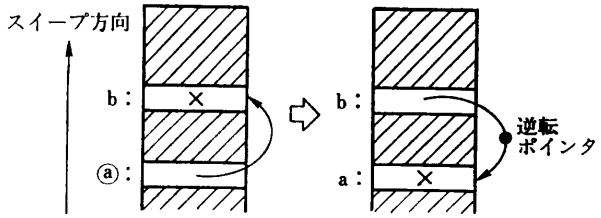
コンパクション後には各エリアの使用領域が縮むことで，以前に占有されていた論理ページに対応する実ページの解放が可能となる．この回収された実ページを GC を引き起したプロセスに割り当てることで通常の処理が再開される．

(1) スタックのコンパクションについて

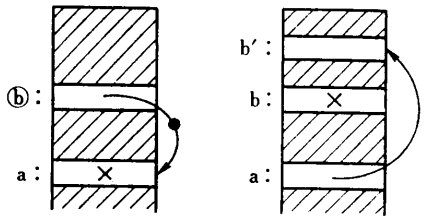
スタックのコンパクションは各セルのアドレスの大小関係を保存したままで行う必要がある．これを効率良く（スタックの大きさに比例した手間で）行うために PSI では Morris の方法[14]をベースとしたコンパクションアルゴリズムを用いている．

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行うものである．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わせを行う向きと同一の方向に行う．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）．

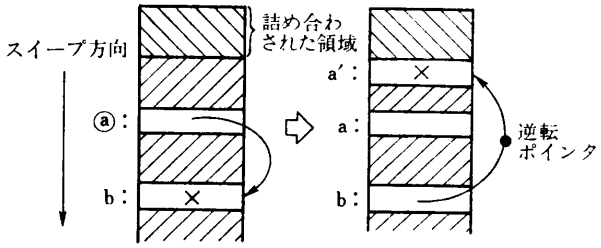


（a） 上向きポインタの処理

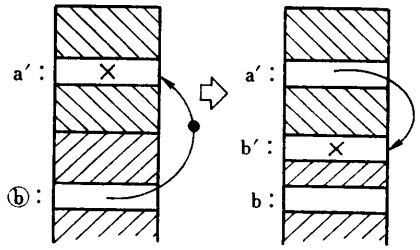


（b） 逆転ポインタの処理

図 6 上向きスイープの処理
Fig. 6 Upward sweep.



（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
Fig. 7 Downward sweep.

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行う．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．

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行われる（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）を行う．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わせがなされる（**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わせが終了する．

（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行うMorris 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，Morris 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行うようにした．

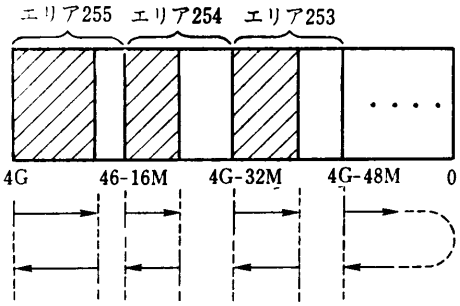


図 8 エリアのスイープ手順
Fig. 8 Sweep between areas.

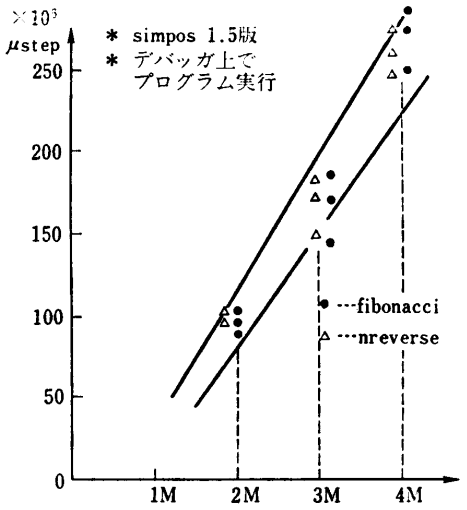


図 9 実装メモリ容量と GC に要するマイクロステップ数の関係
Fig. 9 Relation between real memory and micro steps for GC.

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う．次にエリア1から下向きポインタの処理と詰め合わせをエリア単位に行う(図8参照)．

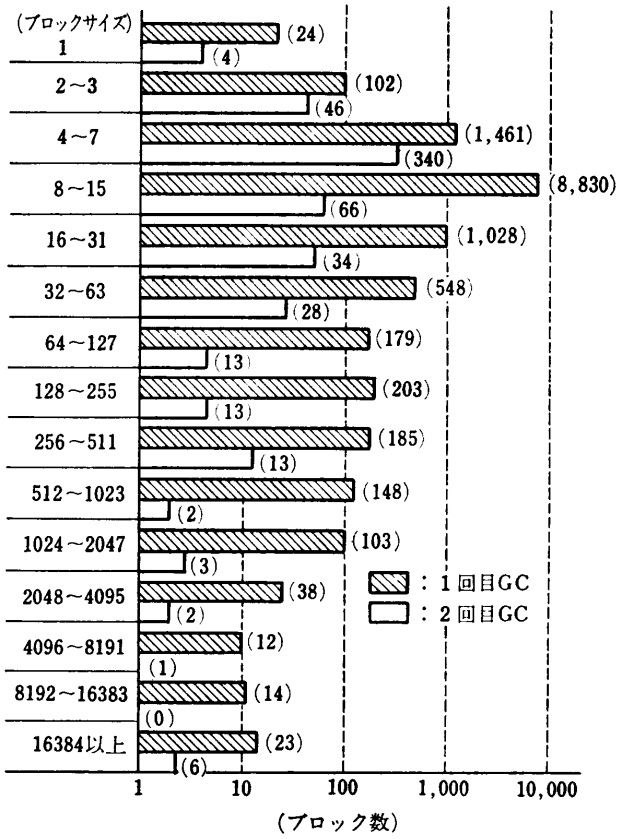
(3) コンパクション処理の高速化について

コンパクションを高速に行うために以下のような点を考慮した手法を導入した．

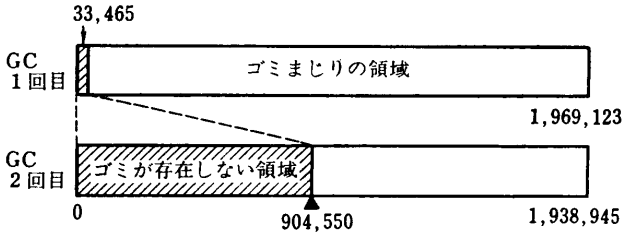
トレールスタックにはゴミセルがないので詰め合わせを行う必要がない．しかしながらトレールスタックから他のスタックへの参照ポインタは，参照先セルの移動に応じて変更する必要があり，一般には2回のスイープが必要となる．しかしトレールスタックは各プロセス内で最も大きなエリア番号を与えられるので，ポインタの向きはすべて上向きである．このことから1回目のスイープによって参照ポインタのアドレス変更が完了するので，2回目のスイープ時にはトレールスタックとして使用されているエリアを省くことができる．

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．



(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
Fig. 10 Marked state of heap area.

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示す．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，小さなブロック同士が，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殿する傾向がある．したがって各エリア内で 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 Morris 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．たと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，たとえ GC が不可能な場合でも，PSI システム全体が倒れることはふせがれている．またプロセス単位の GC が行えるので，GC が行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行うことも可能である.

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*, pp. 53-73 (June 1983).
3) Yokota, M. et al.: A Microprogrammed Interpreter for the Personal Sequential Inference Machine, *Proc. of FGCS '84 Japan*, pp. 410-418 (1984).
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 Comput. Surv.*, Vol. 13 No. 3, pp. 341-367 (1981).
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，Vol. 23, No. 8, pp. 730-741 (1982).
8) Minsky, M. L.: A Lisp Garbage Collector or Algorithm Using Serial Secondary Memory Storage, Memo 58 (rev.), Project MAC, MIT, Cambridge, Mass. (1963).
9) Baker, H. G.:List Processing in Real Time on a Serial Computer, *Comm. ACM*, Vol. 21, No. 4, pp. 280-294 (1978).
10) Collins, G. E.: A Method for Overlapping and Erasure of Lists, *Comm. ACM*, Vol. 3, No. 12, pp. 655-667 (1960).
11) Deutsch, L. P. and Bobrow, D. G.: An Efficient Incremental Automatic Garbage Collector, *Comm. ACM*, Vol. 19, No. 9, pp. 522-526 (1976).
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 Structures, *Comm.*

ACM, Vol. 10, No. 8, pp. 501-506 (1967).
14) Morris, F. L.: A Time- and Space-Efficient Garbage Compaction Algorithm, *Comm. ACM*, Vol. 21, No. 8, pp. 662-665 (1978).

(昭和 61 年 7 月 14 日受付)
(昭和 61 年 8 月 27 日採録)

西川　宏（正会員）

昭和 30 年生．昭和 53 年東京大学計数工学科卒業．同年，松下電器(株)に入社，松下技研(株)に勤務．マイクロプロセッサの研究開発に従事．昭和 57 年から昭和 60 年まで(財)新世代コンピュータ技術開発機構に出向．逐次型推論マシン PSI の研究/開発に従事．60 年 9 月より松下技研(株)勤務．計算機アーキテクチャ，言語処理系等に興味を持つ．

池田　守宏（正会員）

昭和 26 年生．昭和 47 年東京工業高等専門学校電気工学科卒業．同年，三菱電機(株)に入社，計算機製作所に勤務．ミニコンピュータの開発設計に従事．昭和 58 年より(財)新世代コンピュータ技術開発機構の委託による逐次型推論マシン PSI の開発に従事．