並列ガーベジコレクションアルゴリズムとLISPへの適用

A PARALLEL GARBAGE COLLECTION ALGORITHM AND ITS APPLICATION TO LISP

日比野 靖
Yasushi HIBINO

日本電信電話公社　武蔵野電気通信研究所
Musashino Electrical Communication Laboratory,NTT

1. はじめに

並列ガーベジコレクションは、リストプロセスとガーベジコレクションプロセスとが頻繁に干渉しあう、並列プログラミングの非常に困難な問題の一つである。

ガーベジコレクションとリストの処理とを別々のプロセスにやらせようというアイデアは相当古くからのものと見られるが[1]、この挑戦的な問題[2]に、はじめて具体的なアルゴリズムを示したのは、G.Steele [3]であった。このアルゴリズムは、臨界領域を多用しており、あまり効率のよいものとは言えない。

これより遅れて、E.Dijkstraら[2]は、臨界領域を必要としないアルゴリズムを示した。このアルゴリズムは、リストプロセス側のオーバーヘッドは小さいが、ガーベジコレクションの刻印法として走査法を繰返し用いているので、刻印時間が非常に長くなるという問題があった。

最近、KungとSong[4]は、dequeを用いて刻印にスタック法を用いることを可能とし、この欠点を改良しているが、リストプロセス側のオーバーヘッドが大きくなっている。

筆者はこれまで、1ビットの刻印ビットと、走査要求フラッグを用いるアルゴリズムを提案し[7]、小型の多重プロセッサシステムを構成して予備的な実験[8]を行ってきた。さらに最近、このアルゴリズムを発展させ、小規模なLISPシステムを完成させた[9]。

現在のところ、並列ガーベジコレクタを実際にシステムに組込んだ例はきわめて少く、このシステムの他には筆者の知る限りでは、薄田丸所らのLISPマシン[5]があるのみである。

走査要求フラッグを用いる刻印アルゴリズムは、当初の走査法に導入したものよりも発展した形態になっており、現在はスタック法を併用している。これによって[4]とは別の方法で刻印時間を大巾に短縮している。

並列ガーベジコレクタを実際にシステムに組込もうという立場からは、リストの束縛にリニアスタックを用いたいのは当然の要求である。

しかしながら、これまでのアルゴリズムの多くはルートが既知であるという前提のもとで議論されており、実用上は問題が多い。

本報告では、このような制限をなくし、リストの束縛にリニアスタックを用いるという条件のもとで、1ビットの刻印ビットと、走査要求フラッグを用いる並列ガーベジコレクションのアルゴリズムを示し、その正当性、有効性を議論する。また、このアルゴリズムを組込んだ小規模なLISPを紹介し、実測された性能を示して考察を加える。

2. ルートの扱いとスタック

2.1 実用的なアルゴリズムの条件

これまで提案されてきた並列ガーベジコレクションのアルゴリズムは原理的には次の3つのステップから成立っている。

G_1: ルート刻印フェーズ
　生きているリストの先頭ノード（ルート）に刻印する。

G_2: 刻印伝播フェーズ

ルートからの刻印を伝播させる。

G_3：回収フェーズ

刻印のないものを回収し、刻印は消す。

G_1へもどる。

並列ガーベジコレクションの基本的な困難はガーベジコレクションプロセス（以後、ガーベジコレクタ、GCと略す）が、刻印中にリストプロセス（LP）が新しいノードを生成したり（creat）、リストの書換え（change）を行ったりすることにある。しかし、このような状況が生じるのはLISPの場合は、consとreplaceに限られる。このことに注目すれば、このときLP側で何らかの処置をしてGCに知らせてやればよいことになる。例えばDijkstraのアルゴリズムでは、このために灰色の刻印が用いられている。

これまでのアルゴリズムはSteeleのものを除き、ルートが確定しているという前提のもとで議論されてきた。すなわち、G_1では、有限個のルートに順次刻印すればよいとしており、議論の中心はG_2と、creat, change時のLP側の手当てに集中している。この場合、ルート自身の書換えは、リストの書換えの場合と同様に取扱っている。

しかしながら、実際のリスト処理では、このような前提でアルゴリズムを考えるのは問題である。LISPインタプリタにも見られるように多くの処理系では計算の進行に従って変数の評価の環境が動的に変化し、ルートと考えるべきノードも時々刻々増減する。これは関数や手続きの入れ子呼出しや、再帰呼出しを可能とする処理系では必然の性質である。多くの場合、このような処理系では概念的には、プッシュダウンスタックを用いている。

スタックは計算の進行に従って常時、延び縮みしているが、ある瞬間をとらえれば、その瞬間に生存しているすべてのリストのルートを束縛している。このような性質をもつスタックをどのようなデータ構造で実現するかが、並列ガーベジコレクションを考える上での1つの分れ目になる。

例えば、スタックをリンクト・リストで実現したとすると、プッシュ時には、ノードの生成、書換えが必要となるし、スタックのトップの要素に対する操作もリストの書換えが伴うことになる。しかし、これでは、単にリストとたどるだけの操作を行うときも、LP側での手当てが必要になってしまい、実用的とは言い難い。

実用的な意味では、スタックにはリニアスタックを用いるのが望ましい。

しかし、リニアスタックを用いるとすると、単純に考えれば、GCがルートに刻印中は、LPを休止させておかなければならなくなる。これを避けるために、スタック操作と、GCのルート刻印操作とを臨界領域の中で行う方法[3]も考えられるがオーバーヘッドが大きく現実的とは言えない。

従って、実用的なアルゴリズムとしては、次の二つの性質を満すものでなければならない。

(1) リニアスタックを用いること

(2) スタック操作には、臨界領域や、特別な操作が必要ないこと。

2.2 データ構造

ここで、3章で述べるアルゴリズムの基礎となるデータ構造を定義する。

LPとGCが共通にアクセスするデータ構造は、ノード空間（space）、スタック（stack）、スタックポインタ（sp）、走査要求フラッグ（sreq）、走査要求フラッグセマフォア（sreqsema）である。記述はPASCALにならっている。

[定義 2.1]

```
type node=record link1:Ø..M;
                 link2:Ø..M;
                   .
                   .
                 linkn:Ø..M;
                 mark:Boolean end;
var space:array[1..M] of node;
var sp:integer;
var stack:array[1..S] of Ø..M;
var sreq:Boolean;
var sreqsema:semaphore
```

ここでMはノード空間の要素数、Sはスタックの大きさである。nはリンクフィールドの数でLISPの場合はn=2, $link_1 \equiv car$, $link_2 \equiv cdr$である。以後の議論ではn=2とする。またØは空ポインタであるとする。

[定義 2.2] リンク、リスト

2つのノード、$\alpha \equiv space[x], (1 \leq x \leq M)$,

$\beta \equiv space[y], (1 \leq y \leq M)$ において,

$\alpha.link_i = y$

であるとき, α から β へのリンクがあるという.

リンクで結ばれたノードの集合(単独のノードを含めて)をリストという.

[定義 2.3] フリーリスト

フリーリストの先頭は stack[1] に, 末尾は stack[2] に束縛されているものとする.

初期状態では, すべてのノードはフリーリストに属していて, space[i].mark = false $(1 \leq i \leq M)$ であるとする. (Fig. 2.1)

3. アルゴリズム

並列ガーベジコレクションのアルゴリズムはリストプロセスとガーベジコレクタの2つの部分から成立っている. 2つのプロセスは協調して並列ガーベジコレクションを行う.

リストプロセスはリスト基本操作の系列であるとみなせるから, 基本操作ごとに必要な手当てをほどこせばよい.

2章で述べたように, リストの束縛にリニアスタックを用いるモデルの上でのアルゴリズムであるから, リストの基本操作は, すべてスタックの操作と関連づけて定義しておく.

基本操作としては, 次の6種をとり上げる.

LP_1: スタックへのプッシュ操作 (push)
LP_2: スタックのポップ操作 (pop)
LP_3: リンクをたどる操作 (link)
LP_4: ノードの生成 (creat)
LP_5: リンクの書換え (change)
LP_6: スタックの任意の要素への代入 (assign)

リストの基本操作としては, この他に

LP_7: ノードが同一であることの判定
LP_8: リンクが空であることの判定

があるが, これらの操作は, リストに影響を与えないから, ここでの議論から除外する.

アルゴリズムの記述は PASCAL になろう. ただし, and は, 左から右へ評価し, false になったら右側の式は評価しないものとする.

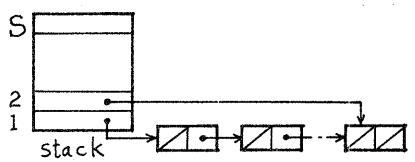

Fig 2.1 Free List

A_1 [リストプロセス側のアルゴリズム]

LP_1: {スタックの k 番目の要素 $(3 \leq k \leq sp)$ をプッシュする操作}

```
stack[sp+1]:=stack[k];
sp:=sp+1
```

LP_2: {スタックをポップする操作}

```
sp:=sp-1
```

LP_3: {$link_1$ または $link_2$ をたどる操作}

```
stack[sp]:=space[stack[sp]].link_i
                    where i=1 or 2
```

(注1)

LP_4: {フリーリストからノードをとり出しスタックにプッシュする操作}

```
while stack[1]=stack[2] do nothing;
space[stack[1]].mark:=true;
stack[sp+1]:=stack[1];
sp:=sp+1;
stack[1]:=space[stack[1]].link_2;
if not space[stack[1]].mark then
   begin
   P(sreqsema);
   space[stack[1]].mark:=true;
   sreq:=true;
   V(sreqsema)
   end
```

LP_5: {stack[sp-1] に束縛されているノードの $link_i$ を stack[sp] の内容に書換える操作}

```
z:=space[stack[sp-1]].link_i;
if z≠∅ and not space[z].mark then
   begin
   P(sreqsema);
   space[z].mark:=true;
   sreq:=true;
   V(sreqsema)
   end;
space[stack[sp-1]].link_i:=stack[sp];
sp:=sp-1
```

(注1)

LP_6: {stack[sp] に束縛されているリストを stack[k] $(3 \leq k \leq sp)$ に代入}

```
if stack[k]≠∅ and
   not space[stack[k]].mark then
   begin
   P(sreqsema);
   space[stack[k]].mark:=true;
   sreq:=true;
   V(sreqsema)
   end;
stack[k]:=stack[sp]
```

注1, 2) stack[sp]=∅ あるいは stack[sp-1]=∅ のチェックはしてあるものとする.

リストプロセス側で手当てを行っているのは，LP_4, LP_5, LP_6, である。

LP_4では、新しいノードに刻印している．また，フリーリストの新たなヘッドが刻印済でなかったら刻印し、sreqをtrueにする．

LP_5では、書換えを行う前に、書換えられるリンクを調べ，必要ならば先のノードに刻印しておく．刻印した時はsreqをtrueにする。

LP_6では、代入の結果、束縛が解かれるノードを調べ，必要ならば刻印しておく。刻印した時はsreqをtrueにする。

注意すべき点は，新しいノードへの刻印を除き，LP側で刻印を行った時は必ずsreqをtrueにすることである。このsreqは次に述べるガーベジコレクタが，刻印伝播フェーズの終了判定に用いる．

なお、LISPの関数consは，LP_4とLP_5の組合せで実現できるが，cons自身を基本操作とすれば，LP_4とLP_5の組合せによるより簡単になる．すなわち、LP_4に次のステップを加えればよい。

```
space[stack[sp]].link1:=stack[sp-2];
space[stack[sp]].link2:=stack[sp-1];
stack[sp-2]:=stack[sp];
sp:=sp-2
```

ガーベジコレクタ側のアルゴリズムは、ルート刻印フェーズ(GC_1)，刻印伝播フェーズ(GC_2)，回収フェーズ(GC_3)に分けられる。ガーベジコレクタはこの3つのフェーズを繰返し実行する．

GC_1ではスタックのトップからルートに刻印していく。

GC_2では、すべてのノード、すなわちspace[1]からspace[M]までをルートと見なして手続きmarklistによって、スタック法で刻印する．刻印の終了はsreqの真偽で判定する．

GC_3では、刻印のないノードをフリーリストの末尾につなぐ．刻印のあるノードは刻印を消す．

4. アルゴリズムの正当性と有効性

4.1 前提と諸定義

並列ガーベジコレクションのアルゴリズムが正当であるというためには、少くとも次の二つの条件が満される必要がある．

A_2[ガーベジコレクタ側のアルゴリズム]

```
begin
GC1:$root marking phase$
    for gcx:=sp downto 1 do
      if not stack[gcx]=ø then
         space[stack[gcx]].mark:=true;
    sreq:=false;
GC2:$mark propagating phase$
    for gcy:=1 to M do
      if space[gcy].mark then
         marklist(gcy);
    P(sreqsema);
    if sreq then
       begin
       sreq:=false;
       V(sreqsema);
       goto GC2
       end;
    V(sreqsema);
GC3:$reclaiming phase$
    for gcz:=1 to M do
      if space[gcz].mark
       then space[gcz].mark:=false
       else
         begin
         space[gcz].link1:=ø;
         space[gcz].link2:=ø;
         space[stack[2]].link2:=gcz;
         stack[2]:=gcz
         end;
    goto GC1
end
```

但し、marklist(x)は、次の手続きとする。

```
procedure marklist(x);
begin
L1:if space[x].link1≠ø and
      not space[space[x].link1].mark
    then
     begin
     space[space[x].link1].mark:=true;
     marklist(space[x].link1)
     end;
   if space[x].link2≠ø and
      not space[space[x].link2].mark
    then
     begin
     space[space[x].link2].mark:=true;
     x:=space[x].link2; goto L1
     end
end
```

C_1：ガーベジノードのみがフリーリストにもどされる．

C_2：ガーベジコレクタが生きているリストを書換えることはない．

ガーベジコレクタが走行中は，リストプロセスを休止させておく通常のガーベジコレクションのアルゴリズムも当然この2条件を満している．したがって，並列ガーベジコレクションのアルゴリズムとしては，この2条件を満しただけでは正当であっても，有効であるとは言えない．並列アルゴリズムとして有効であるためには次の条件を満さなければならない．

C_3：リストプロセスが，ガーベジの回収待ちに入らずに走行しつづける条件（いわゆる never run out of space）の条件が成立つこと．

この条件が成立つかどうかは，リストプロセスのノードの消費率，リストプロセスが使用中であるノードの割合，およびガーベジコレクタの刻印，回収の速度に依存する．これらのパラメータは，処理の対象となるプログラムの性質，2つのプロセスを実行するプロセッサの性能によって左右されるが，現実的な仮定のもとで，この条件が満され得ることが保証できなければならない．

つぎに，準備としてこのあとの議論で用いる言葉を定義しておこう．

[定義4.1] 活動ノード集合

stack[1]，…，stack[sp]に束縛されているリストを生きているリストという．

生きているリストに属するノードを生きているノードという．

すべての生きているノードの集合を活動ノード集合といい，文字Aで表わす．

[定義4.2] 刻印到達可能

ノード$\alpha_1, \alpha_2, \cdots \alpha_n$において，$\alpha_1.mark = true$で，$\alpha_i$から$\alpha_{i+1}$（$1 \le i \le n-1$）に向かうリンクがあるとき，ノード$\alpha_1$から$\alpha_n$に刻印到達可能であるという．

[定義4.3] 刻印到達可能集合

すべての space[i].mark = true（$1 \le i \le M$）なるノードから刻印到達可能なすべてのノードの集合を刻印到達可能集合といい，文字Rで表わす．

[記法4.1]

ノードαにおいて，$\alpha.mark = true$のときαから刻印到達可能なすべてのノードからなる集合を$\langle\alpha\rangle$と書く．

4.2 正当性の証明

3章で示したアルゴリズムがC_1とC_2を満すことを証明する．

Lemma 4.1

ルート刻印フェーズの任意の時刻tにおける sp の値をsp_t，gcx の値をgcx_t，活動ノード集合をA_t，刻印到達可能集合をR_tとする．ここでノードの集合M_tとN_tとを次のように定義する．

1) $gcx_t \le sp_t$のとき

$M_t \equiv$ stack[i]（$gcx_t \le i \le sp_t$）に束縛されているリストのすべてのノードの集合

$N_t \equiv$ stack[j]（$1 \le j < gcx_t$）に束縛されているリストのすべてのノードの集合

2) $gcx_t > sp_t$のとき

$M_t \equiv \phi$

$N_t \equiv$ stack[j]（$1 \le j \le sp_t$）に束縛されているリストのすべてのノードの集合

このとき，次の関係が成立つ．

$$A_t = M_t \cup N_t \subseteq R_t \cup N_t \qquad (4.1)$$

Proof.

1. $gcx_t > sp_t$のときは定義より明らか．
2. $gcx_t \le sp_t$のとき

ルート刻印フェーズに入る直前の時刻をt_0とすると，t_0において，定義より，

$$M_{t_0} = \phi$$

であるから，

$$A_{t_0} = N_{t_0} \subseteq R_{t_0} \cup N_{t_0} \qquad (4.2)$$

が成立つ．

3. t_0から数えてk回目の基本操作を行った直後の時刻をt_kとしたとき，

$$A_{t_k} = M_{t_k} \cup N_{t_k} \subseteq R_{t_k} \cup N_{t_k} \qquad (4.3)$$

が成立したと仮定し，$k+1$回目の基本操作を行った時刻t_{k+1}において

$$A_{t_{k+1}} = M_{t_{k+1}} \cup N_{t_{k+1}} \subseteq R_{t_{k+1}} \cup N_{t_{k+1}} \qquad (4.4)$$

を証明する．

3.1 基本操作が LP_1 のとき,

$$M_{t_{k+1}} = M_{t_k},\ N_{t_{k+1}} = N_{t_k}$$
$$R_{t_{k+1}} = R_{t_k}$$

であるから、(4.4) は明らか.

3.2 基本操作が LP_2 のとき,

$$M_{t_{k+1}} \subseteq M_{t_k},\ N_{t_{k+1}} = N_{t_k}$$
$$R_{t_{k+1}} = R_{t_k}$$

であるから、(4.4) は明らか

3.3 基本操作が LP_3 (link$_i$) のとき,

$$M_{t_{k+1}} \subseteq M_{t_k},\ N_{t_{k+1}} = N_{t_k}$$
$$R_{t_{k+1}} = R_{t_k}$$

であるから、(4.4) は明らか.

3.4 基本操作が LP_4 (creat) のとき,

生成された ノードを γ とすると,

$$M_{t_{k+1}} \cup N_{t_{k+1}} = M_{t_k} \cup N_{t_k} \quad (4.5)$$

また、LP_4 は γ に刻印しているから,

$$R_{t_{k+1}} = R_{t_k} \cup \langle\gamma\rangle$$

したがって

$$R_{t_{k+1}} \cup N_{t_{k+1}} = R_{t_k} \cup \langle\gamma\rangle \cup N_{t_{k+1}}$$

ところで

$$N_{t_{k+1}} \cup \langle\gamma\rangle = N_{t_k}$$

であったから、

$$R_{t_{k+1}} \cup N_{t_{k+1}} = R_{t_k} \cup N_{t_k} \quad (4.6)$$

よって (4.3), (4.5), (4.6) より (4.4) が成立.

3.5 基本操作が LP_5 (change$_i$) のとき,

$$\alpha \equiv \text{space}[\text{stack}[sp_{t_k}]]$$
$$\beta \equiv \text{space}[\text{stack}[sp_{t_k}-1]]$$
$$\gamma \equiv \text{space}[\beta.\text{link}i]$$

とし、γ を α に change したとする.

まず

$$N_{t_{k+1}} = N_t,\ M_{t_{k+1}} \subseteq M_{t_k} \quad (4.7)$$

ところで、$\gamma \in R_{t_k}$ であるから

$$R_{t_{k+1}} = R_{t_k} \cup \langle\gamma\rangle = R_{t_k}$$

したがって

$$R_{t_{k+1}} \cup N_{t_{k+1}} = R_{t_k} \cup N_{t_k} \quad (4.8)$$

よって (4.3), (4.7), (4.8) より (4.4) が成立.

3.6 基本操作が LP_6 (assign) のとき

$$\alpha \equiv \text{space}[\text{stack}[sp_{t_k}]]$$
$$\beta \equiv \text{space}[\text{stack}[j]]$$

で、stack[j] := stack[sp_{t_k}] なる代入を行ったとする.

1) $gcx_{t_k} \le j \le sp_{t_k}$ のとき,

$$N_{t_{k+1}} = N_{t_k},\ M_{t_{k+1}} \subseteq M_{t_k} \quad (4.9)$$

ところで、$\beta \in R_{t_k}$ であるから

$$R_{t_{k+1}} = R_{t_k} \cup \langle\beta\rangle = R_{t_k}$$

したがって,

$$R_{t_{k+1}} \cup N_{t_{k+1}} = R_{t_k} \cup N_{t_k} \quad (4.10)$$

よって (4.3), (4.9), (4.10) より (4.4) が成立.

2) $1 \le j < gcx_{t_k}$ のとき,

$$N_{t_{k+1}} \subseteq N_{t_k},\ M_{t_{k+1}} = M_{t_k} \quad (4.11)$$

ところで

$$N_{t_{k+1}} \cup \langle\beta\rangle = N_{t_k}$$

であるから、

$$R_{t_{k+1}} \cup N_{t_{k+1}} = R_{t_k} \cup \langle\beta\rangle \cup N_{t_{k+1}} = R_{t_k} \cup N_{t_k} \quad (4.12)$$

よって (4.3), (4.11), (4.12) より (4.4) が成立.

以上 3.1 ～ 3.6 をまとめると、(4.3) のもとで、(4.4) が成立する. 従って、帰納法により、n が任意の n のときに (4.3) は成立する.

4. 時刻 $t_n \le t < t_{n+1}$ なる t においても、(4.3) は成立するから、よって (4.1) は証明された. ■

Lemma 4.2

ガーベジコレクタがルート刻印フェーズを終了した時刻を t_e とすると,

$$A_{t_e} = M_{t_e} \subseteq R_{t_e} \quad (4.13)$$

が成立つ.

Proof.

時刻 t_e において、$N_{t_e} = \phi$ であるから、Lemma 4.1 より明らか.

Lemma 4.3

ガーベジコレクタの刻印伝播フェーズにおける任意の時刻を t とすると,

$$A_t \subseteq R_t \quad (4.14)$$

Proof.

1. Lemma 4.2 より、刻印伝播フェーズに入る直前の時刻 t_0 において

$$A_{t_0} \subseteq R_{t_0}$$

が成立する.

2. t_0 から数えて、k 回目の基本操作が行われた直後の時刻を t_k とし,

$$A_{t_k} \subseteq R_{t_k} \quad (4.15)$$

が成立すると仮定して、$k+1$ 回目の基本操作を行った直後の時刻 t_{k+1} において,

$$A_{t_{k+1}} \subseteq R_{t_{k+1}} \quad (4.16)$$

を証明する.

2.1 基本操作が LP_1, LP_2, LP_3, LP_4, LP_5, LP_6

のいずれであっても，明らかに

$$R_{t_{k+1}} = R_{t_k} \qquad (4.17)$$

が成立つ．

2.2 基本操作が LP_1(push)，LP_4(creat) のときは，常に

$$A_{t_{k+1}} = A_{t_k} \qquad (4.18)$$

が成立つ．

2.3 基本操作が LP_2(pop)，LP_3(link$_i$)，LP_6(assign) のときは，

$$A_{t_{k+1}} \subseteq A_{t_k} \qquad (4.19)$$

以上をまとめると，(4.15)，(4.17)，(4.18)，(4.19)によって，(4.16)が成立する．

従って，帰納法により，kが任意のnのとき，(4.15)が成立する．

3. $t_n \leq t < t_{n+1}$ なる時刻 t においても，(4.15)は成立するから，(4.14)は証明された． ■

Lemma 4.4

刻印伝播フェーズにおいて，刻印到達可能集合 R_t は一定，すなわち

$$R_t = R_{t_0} \qquad (4.20)$$

Proof.

(4.17)における t_k が一般の t について成立つことは明らか．よって(4.20)が成立． ■

Lemma 4.5

sreg の判定直前までに刻印済となるノードの集合を P とし，その時の刻印到達可能集合を R とすると，

$$\text{sreg} = \underline{\text{false}} \text{ ならば } P = R \qquad (4.21)$$

が成立つ．

Proof.

sreg を true にするのは LP側だけであるから，sreg = false が成立するということは 刻印伝播フェーズのはじめ t_{GC_2} で，sreg = false であり，かつ sreg の判定時 t_{sreg} まで，LP側が sreg を true にする操作を行わなかったということである．

t_{GC_2} から t_{sreg} までの期間において，LP側で，sreg を true にする操作と刻印を行う操作とは，セマフォア sregsema に対する臨界領域の中で行われているから，sreg = true ということと，LP側がこの期間に刻印を行ったということは，1対1に対応している．

従って，sreg = false ならば，この期間，LP側は刻印を行っていない．

刻印伝播フェーズのアルゴリズムは，すべての刻印のあるノードをルートと見たてて，手続き marklist によって刻印を伝播させる．

手続き marklist は，標準的なスタック法による刻印アルゴリズムであるから，LP側からの刻印がない限り，ルートと見たてたノードから，刻印到達可能なすべてのノードに刻印する．

すべての刻印のあるノードについて，marklist を適用するから，刻印到達可能集合 R のすべてのノードが刻印される．

したがって $P = R$ が成立する． ■

Lemma 4.6

回収フェーズのはじめの刻印済のノードの集合を P_0，回収フェーズの任意の時刻 t における活動ノード集合を A_t，回収フェーズ中に新たに刻印されるノードの集合を Q_t とすると

$$A_t \subseteq P_0 \cup Q_t \qquad (4.22)$$

Proof.

Lemma 4.3により、刻印伝播フェーズでは、$A \subseteq R$ が成立する．

また，Lemma 4.5により，刻印伝播フェーズをぬけ出した時，$P = R$ であるから，このとき，$A \subseteq P$ が成立する．

よって回収フェーズのはじめの時刻を t_0 とすれば $A_{t_0} \subseteq P_0$

回収フェーズ中に新たに活動ノードとなるノードの集合を B_t とすると

$$A_t \subseteq A_{t_0} \cup B_t$$

しかるに、B_t は、LP側の LP_4(creat) 操作によるものだけであるから、必ず刻印されており，

$$B_t \subseteq Q_t$$

よって

$$A_t \subseteq A_{t_0} \cup B_t \subseteq P_0 \cup Q_t$$

が成立する ■

Theorem 4.1

並列ガーベジコレクションのアルゴリズム．A_1，A_2 は．条件 C_1，C_2 を満す．

Proof.

Lemma 4.6より，回収フェーズにおいては，$A_t \subseteq P_0 \cup Q_t$ が成立つ．

回収フェーズでは，刻印のないノードだけも回収するから，$P_0 \cup Q_t$ に属するノードは回収されない．

したがって，A_t は回収されることはなく，条件 C_1 を満す．

また，ガーベジコレクタ側のアルゴリズムから明らかなように，回収されないノードのリンクは書換えられることはない．

よって条件 C_2 を満す． ■

Note

回収フェーズ終了時を t_e とすると，P_0 に属するノードの刻印は消されるが，Q_{te} に属するノードは，刻印が消されるものと，刻印されたまま残るものがある．　なぜならばGCの gc_2 が通過したあとで，LPが刻印することがあるからである．

この刻印の残ったノードに関する刻印到達可能集合を R_e とすると，この R_e が，Lemma 4.1における R_{t_0} となる．　従って一般には，Lemma 4.1において R_{t_0} は ϕ とはかぎらない．

4.3 有効性の考察．

有効性の条件 C_3 が満されるかどうかを考察する．　このため，次のようなパラメータを仮定する．

1) c：ノードの消費率　(nodes/unit time)
2) m：刻印伝播フェーズにおける，ノードの処理率　(nodes/unit time)
3) r：回収フェーズにおけるノードの処理率　(nodes/unit time)
4) ρ：ノードの回収率

C_3 が満される定常状態では，消費されるノードの数と，回収されるノードの数は，つり合がとれている．

簡単のために，ルート刻印フェーズの時間を無視し，刻印伝播フェーズにおける走査の繰返し回数を K とすると，次の式が成立つ．

$$c\left(\frac{KM}{m}+\frac{M}{r}\right)=\rho M \qquad (4.22)$$

(4.22)を整理すると

$$\frac{c}{r}\left(\frac{Kr}{m}+1\right)=\rho$$

ここで　$r \simeq m$ を仮定すると，

$$\frac{Kr}{m}+1 \simeq K+1$$

これを用いると

$$c=\frac{r}{K+1}\rho \qquad (4.24)$$

また，ρ の性質から

$$\rho < 1 \qquad (4.25)$$

でなければならない．

(4.24)，(4.25)の意味を考えると次のようなことが導ける．

1) ノードの消費率 c は，ノードの回収率 ρ に比例して小さくなければならない．
2) c は，回収フェーズにおけるノードの処理率 r の何分の1かでなければならない．

K が1に近ければ，この条件を満すことは可能である．　ρ はプログラムの性質によって定まる．

5. LISPのインプリメント

5.1 実験システムの概要

3章で述べた並列ガーベジコレクションの有効性を検証するために試作したシステムの概要を紹介する．

ハードウェアはTOSBAC-40Lのマルチプロセッサシステムである．　このプロセッサはメモリインタフェースが非同期式になっているので，主記憶共用型のマルチプロセッサが容易に構成できる．　結合方法は，マスタ・スレーブ式とし，スレーブプロセッサは簡単なバス制御アダプタを介してセレクタチャンネルの下位に接続されている．(Fig 5.1)

このマルチプロセッサは，並列プログラミングの実験用として構成したものであって，ハードウェアとしての性能向上をねらったものでは

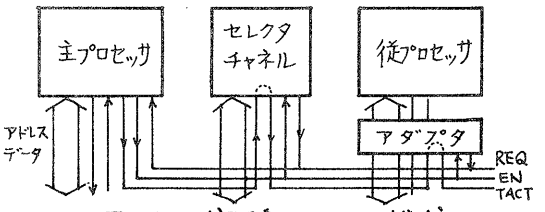

Fig 5.1 実験システムの構成

ない。　したがって，システムとしては完全にメモリアクセスネックとなっていて，1台当りのプロセッサの能力は，シングルプロセッサ時の70%程度に低下している。　これは，メモリのスループットが不足しているためで，やむを得ない。

相互排除には，T-40LのTest & Set命令を用いている。この命令はオペランドを読出して，そのMSBを条件コードに設定し，同時にオペランドの全ビットを'1'にする。　全ビットを'1'にする動作はメモリ側で行っている。

また40Lのメモリは，いわゆるリード・モディファイ・ライト動作モードがないので，ビットセット命令なども非可分操作と見なすことができない。

このように，ハードウェアとしては，並列プログラムの動作には不十分であるが，このシステムは，手軽に並列アルゴリズムの実験が行えるものとして十分意義があるものと考える。

次にLISPの概要を述べる。

このLISPは並列ガーベジコレクションのアルゴリズムの実験用として作成したもので，本格的な応用を目的としたものではない。したがって，実験に必要な最小限の関数しか組込まれていない。　しかし機能的にはLIPQ[6]と同様の引数機構，program feature，go，loopなど本格的なプログラムを書くのに必要なものは一応そろっている。

また，変数束縛にはスタックを用いている。

5.2 スタックポインタの扱い

スタックポインタは両方のプロセッサからアクセス可能な場所に置く必要がある。　相手のプロセッサのレジスタが直接に覗き込めるような特殊なハードウェアを持っていない限り，メモリ経由で知らせるしかない。

ここでは単純に，LPのスタック・ポインタ・レジスタ(spR)の値の写しをメモリ(spMEM)に書込む方法をとった。

GCから見てスタックポインタの値が意味を持つのは，ルート刻印フェーズの始めだけである。　このときspMEMがspRに追随できないと危険が生じる。　この危険が生じるのは，LPがspMEMに写しをとるより早く，GCが回収フェーズからルート刻印フェーズに移るときだけである。　したがって，この危険を避けるには，回収フェーズとルート刻印フェーズとの間にidle stepを挿入すれば十分である。

こうすれば，spR，spMEMの更新を非可分操作にする必要はなくなる。

5.3 刻印ビットの配置

3章のアルゴリズムでは，2章で定義したように，ノード毎に刻印ビットを設けている。

形式的なアルゴリズムでは，刻印操作はリンクフィールドに何ら影響を与えないものとしている。　しかし，インプリメント上，刻印ビットをリンクフィールドと同じ語に配置した場合は，主記憶のアクセス単位との関係で注意を要する。

この実験で用いたハードウェアは，前述のようにリード・モディファイ・ライト動作ができないので，ビット操作命令は非可分操作とならない。したがって，リンクの書換えと刻印操作とを，臨界領域の中で行わなければならなくなる。

これに対して，ビットテーブルを用いる場合は，リンクの書換えを臨界領域に入れる必要はなくなる。　しかし，アドレスからビット位置を求める計算量や，ビットテーブルへのアクセスはやはり非可分操作としなければならないことも考えると，必ずしも有利にならない。

結局，実際には，刻印ビットはリンクフィールドと同一の語にとり，臨界領域を用いることにした。

6 実測結果

実験システムは，現在，今年行われたLISPコンテスト(情報処理学会記号処理研究会主催)の問題[10]をすべて実行できるレベルにある。

このうちから，TPU(Unit ResolutionによるTheorem provingのプログラム)の実測結果を示す。(表6.1)

実測データに対する関心は次の2点にあろう。

1) 条件C_3が実際に満されるか

2) 通常のガーベジコレクションに比べてどの程度実行時間が短縮されるか

表6.1からわかるように，問題No.6を除いて，条件C_3は満されている。　問題No.6は，この中で最も問題のサイズが大きく，使用するノードの数も多い。

次に2)について述べる。　この実験で用

表6.1 並列GCと通常のGCの比較

No	parallel GC t_1	r_1	n_1	w	regular GC $t_2(t_g)$	r_2	n_2	G	I
1	193	352	44	-	196(5)	3531	4	2.6	1.5
2	580	363	131	-	601(27)	2639	17	4.5	3.5
3	243	329	59	-	244(6)	3324	5	2.5	0.4
4	335	368	74	-	342(11)	3143	8	3.2	2.0
5	38	287	11	-	38(2)	3226	1	5.3	-
6	1010	360	224	2	1055(56)	1926	40	5.3	4.3
7	218	368	53	-	222(7)	3068	6	3.2	1.8
8	161	298	42	-	161(3)	3446	3	1.9	-
9	118	289	32	-	118(2)	3622	2	1.7	-

t_i execution time (sec)
t_g execution time of GC (sec)
r_i number of reclaimed nodes/cycle
n_i frequency of GC (times)
w frequency of wait
G $t_g/t_2 \times 100$ (%)
I $(t_2-t_1)/t_1 \times 100$ (%)

いているハードウェアは，前述のようにメモリアクセスネックになっているので，プロセッサの性能は著しく低下してしまっており，純粋にアルゴリズムの比較を行うには適当でない。

ここで示した通常のガーベジコレクションを用いた場合のデータは，この点を考慮して，人為的にプロセッサの性能を低下させ，マルチプロセッサ時とほぼ等しくなるようにして測定したものである。

表からわかるように，ガーベジコレクションの時間(t_g)は，高々5%程度であり，少いものは2%未満である。(ノード数M=723)

従って，ガーベジコレクションを並列に行っても，この問題(TPU)に関する限り，高々数%の改善しか望めないわけである。

実測データの示すところによれば，改善率Iは，ガーベジコレクション時間比Gに近い値を示しており，本報告で示したアルゴリズムは，きわめてオーバーヘッドが小さいことが実証されたと言えよう。

データは，インタプリタで動作させた場合のもので，ノードの消費率が小さく，並列ガーベジコレクションの良さが発揮できていない。

プログラムがコンパイルされれば，リストプロセスの速度がガーベジコレクタに比べて相対的に上るので，改善率Iは更に向上することが期待できるが，一方，条件C_3を満すことも困難になってくる。

これらについては，今後の実験が必要である。

7. あとがき

本報告で示した並列ガーベジコレクションのアルゴリズムは，実測でも条件C_3を満し，並列アルゴリズムとして有効であることが示された。

また，そのオーバーヘッドも十分小さい。

しかしながら，ガーベジコレクションを並列に行うことによる処理速度の向上は，通常のガーベジコレクションが要していた時間がその上限であり，それほど多くを期待することはできない。

しかし，このことが並列ガーベジコレクションの価値を失わせるものではないと考える。

例えば，従来からも言われているように，Q&Aシステムの応答時間の改善には有効であろう。また理論的な興味としては，計算時間がガーベジコレクションによって乱されることがないという特徴がある。

さらに，リスト処理を並列に行おうとする場合には，並列ガーベジコレクションは不可欠となろう。その意味で，並列ガーベジコレクションが，実現可能であることが実証されたことは，今後のリストの並列処理に大きな足がかりを与えたものと言えよう。

最後に，本研究の機会を与えられ，終始ご指導いただいた電電公社武蔵野電気通信研究所，池野特別研究室，池野信一室長に感謝いたします。また，LISPについて教えていただき，アルゴリズムを熱心に御討論いただいた，竹内郁雄調査員に感謝いたします。

References
[1] Knuth, D.E. The Art of Computer Programming, Vol.1 Fundamental Algorithms. Addison-Wesley, 1969
[2] Dijkstra, E.W. et.al. "On-the-fly Garbage Collection: An Exercise in Cooperation", in Language Hierarchies and Interfaces, Springer-Verlag, 1976
[3] Steele, G.L.Jr. "Multiprocessing Compactifying Garbage Collection", C.ACM vol.18 No.9, 1976
[4] Kung, H.T. and Song, S.W. "An efficient Parallel Garbage Collection System and its Correctness Proof", Proc. the 18'th Annual Symposium on foundation of Computer Science, 1977
[5] 藤田丸所 "マルチプロセッサによるLISPマシンの試作" 情報処理学会第19回大会予稿 2B-7, 1978
[6] 奥乃, 竹内 "リスト処理言語LIPQとその処理系" 研究実用化報告 vol.26 No.10 1977
[7] 日比野 情報処理学会第18回大会予稿 No.215, 1977
[8] 〃 昭和53年信学会全国大会予稿 No.1378, 1978
[9] 〃 情報処理学会第19回大会予稿 2C-4, 1978
[10] 竹内 "LISPコンテストの結果" 記号処理 5-3, 1978