記　号　処　理 75－3
（１９９４．９．１６）

Timed-GCを備えたScheme処理系と評価

浅井 敏弘　伊藤 貴康

東北大学 大学院 情報科学研究科

Lispはその処理過程で記憶領域を動的に消費するためにガーベジコレクション(GC)と呼ばれるメモリの管理機構が必要である。Lispをリアルタイム処理に適用しようとする場合、GCよるユーザプログラムの処理の長時間の中断が実用化の大きな障害になる。この中断を回避する方法としてリアルタイムGCがある。リアルタイムGCにはシングルプロセッサシステム上ではインクリメンタルGC等がある。

本報告では汎用シングルプロセッサマシン上のリアルタイムGCとして提案されているTimed-GCの実現アルゴリズムを与え実験評価する。Timed-GCはGCの各段階を細分化し、外部からのタイマ割り込みによりそれを起動し、リスト処理と並行して行なうものである。また従来型のGC、インクリメンタルGC、Timed-GCを備えたScheme処理系を試作し、ベンチマークプログラムを実行し、各GCアルゴリズムのリアルタイム性と実行効率について評価した。

Scheme system with Timed-GC and its evaluation

Toshihiro ASAI　Takayasu ITO

Department of Computer and Mathematical Sciences
Graduate School of Information Sciences
Tohoku University

Lisp system employs a dynamic memory management system called Garbage Collection(GC),since list-processing requires a dynamic use of memory area. However,GC may incur a long suspended interruption of executions of user programs. The incremental GC is a real-time GC to avoid or reduce such long suspension on a single processor machine.

Timed-GC is another real-time GC on a single processor machine.With Timed-GC algorithm,each GC action is subdivided and invoked by external timer and it will be executed concurrntly with list-processing.This paper presents an implementation algorithm for Timed-GC,and we give its experimental evaluations,implementing a Scheme system with Timed-GC.Moreover,we compare Timed-GC with a conventional stop GC and Incremental-GC,implementing three Scheme systems equipped with these three GCs.

We show that Timed-GC is superior to Incremental-GC in its real-timedness using some benchmark programs.

1 はじめに

Lispはその処理過程で記憶領域を動的に消費するためにガーベジコレクション(GC)と呼ばれるメモリ管理機構が必要である。Lispをリアルタイム処理に適用しようとした場合、GCによるユーザープログラムの処理の長時間の中断が実用化の大きな障害になる。この中断を回避する方法としてはリアルタイムGCが提案されている。GC専用のプロセッサを設けてリスト処理と並列に実行する方法やシングルプロセッサーシステムではリストプロセス中にGCを分割して埋め込むインクリメンタルGC[2]等がある。

リアルタイム性を主に考えた場合にはGCを時間的な観点から考える必要がある。この観点から筆者の一人によって提案されたものとしてTimed-GC[3]がある。Timed-GCはマーク&スイープに基づくGCであり、GCの各段階を細分化し外部からのタイマ駆動により一定時間隔で細分化されたGC処理をリスト処理と並行に行うものである。

この論文ではTimed-GCの実現アルゴリズムを与え、その動的な振舞い、時間効率について解析し、試作したScheme処理系に従来の一括型GC、インクリメンタルGC、Timed-GCを実装し、ベンチマークプログラムを実行し、各GCの比較を行ないTimed-GCがリアルタイム性に優れたものであることを示す。

Lispにおけるリアルタイム性

逐次Lispシステムの場合にはGCがユーザープログラムの実行を中断して行われるので実行中に長時間の不意の中断が生じる。

GCによる長時間の中断を回避するという意味でのリアルタイム性をLispシステムに持たせるためにリアルタイムGCが提案されている。

リアルタイムGCを備えたLispシステムは次のような要求を満たすように設計される必要がある。

1. ユーザープログラム実行中にセルの飢餓状態に陥いって処理が中断することがない。(安全性)
2. GCによるユーザープログラムの中断がリアルタイム性の要求を満たすような有限時間内である。

2 Timed-GCのアルゴリズム

従来の一括型GCでは自由セルがなくなった時点でGCの処理を一括して行なっていた。このためにリスト処理に長時間の中断が生じリアルタイム処理には向かなかった。この問題を解決するために提案されているGCとしてはリアルタイムGCがある。リアルタイムGCの基本的な考えかたはGCの処理を細分化しリスト処理の実行の途中に埋め込むことによって、GCの処理による一度の中断時間を短くする方法である。従来、リアルタイムGCとしてインクリメンタルGC[2]が提案されている。インクリメンタルGCはメモリー空間の閾値に依存してGCを起動する。一方Timed-GCはリスト処理の実行時間に依存してGCを起動する。

以下2.1節ではリアルタイムGCの基礎となる一括型GCのアルゴリズムについて2.2節ではインクリメンタルGCのアルゴリズの簡単な説明とTimed-GCのアルゴリズムについて説明する。2.3節では各GCの概念的な比較を行なう。

2.1 一括型GCのアルゴリズム

Timed-GCの基礎となるGCのアルゴリズムとしてはGCスタックを用いた印付けと自由リスト方式により回収を行なう一括型GCを採用している。

GCスタックに基づく印付けではシステムでルートとなるセルから順次たどっていけるすべてのリスト要素の印付けをマークビットを用いて行なう。その際にマークしたセルの*car*部、*cdr*部それぞれをスタックに積み、次のマーク時にはGCスタックからポップしてさらに同じことをスタックが空になるまで繰り返す。

自由リスト法に基づく回収は自由セルを一本のリストにしてつなぎ自由リストとして管理する方法である。スイープ処理ではマークされていないゴミセルをこのリストに順次つなげることで回収を行う。

ヒープからのセルの切り出し(領域獲得)時に自由セルがなくなった時点でGCが起動される。その際にリスト処理の長時間の中断が生じる。

以下、本論文ではヒープからのセルの切出しを領域獲得、その関数を領域獲得関数という。

2.2 Timed-GCのアルゴリズム

2.1節で述べたアルゴリズムに基づいたリアルタイムGCとしてインクリメンタルGCとTimed-GCが提案されている。

これらのリアルタイムGCの基本的な考え方はGCの各段階を細分化し、マークとスイープの処理をユーザープログラムのリスト処理中に分散させることでGCによる長時間のリスト処理の中断を防ぐことを目的としている。しかし、分割GCの起動方法において両者は異なる。

(1) インクリメンタルGC[2]

インクリメンタルGCの場合、自由セル数のカウンタと分割GCのマーク処理、スイープ処理の一度の処理量を決めるパラメータ、自由セルの閾値、その時点のGCの段階を示すフラグが必要になる。

またマーク洩れを回避するためにリンク書き換え関数には特別な操作を加える。つまりリンクが書き換られる前に指していたセルがマーク洩れのおそれがあるのでGCスタックにつんでおく。

システム起動後、リスト処理において領域獲得時には自由セル数のカウンタと閾値とを比較し、自由セル数が閾値以上であれば自由セル数のカウンタを一つ減らしセルを得る。そうでない場合には以後領域獲得毎に分割GCを起動しフラグが示すGCの段階に応じてパラメータで定められた一定量のマーク、スイープ処理を行ない、その後自由セルのカウントを1つ減らす。この時にGCの段階が終了していればフラグを次のGCの段階に切り替え、その後にセルを獲得する。このことを全セルの走査が終了するまで領域獲得毎に行なう。

また分割GCを行なうためにヒープのどこまでを走査したかを示すポインタを設け、分割GCのスイープ処理はそのポインタが指すところから始める.

飢餓状態に陥いらないで実行を続けるためのパラメータと閾値の条件(安全条件)については文献[2]に詳しく述べられているが、一般には閾値を小さくとればパラメータを大きくとる必要があり、その場合には分割GCによる一度のリスト処理の中断時間は長くなるが領域獲得時の分割GCの回数は減る。逆に閾値を大きくとった場合には一度の分割GCによるリスト処理の中断時間は短くなるが領域獲得時に実行される分割GCの頻度は増す。

(2) Timed-GC

Timed-GCにはリアルタイム化にあたって一定間隔に割り込みをかけるためのタイマ、分割GCのマーク処理、スイープ処理の一度の処理量を決めるパラメータ、現在のGCの段階を示すフラグが必要になる。

リンク書き換え関数にはインクリメンタルGCと同様な操作を加えている。

システム起動時にタイマを初期化し割り込み間隔を設定する。リスト処理が一定時間行なわれるとタイマにより割り込みがかかりGC処理を行うルーチンが呼び出される。呼び出されたGC処理ルーチンではフラグが示すその時点のGCの段階によりマークまたはスイープ処理をパラメータで設定された回数行なう。その際にGCの各段階が終了すればフラグを次の段階に切り替えてリスト処理に戻る。タイマにより分割GCが起動された時には次のGCの段階が実行される。

分割GCのスイープ処理についてはインクリメンタルGCと同様にスイープポインターを用いる。

Timed-GCの場合にはリスト処理とGC処理が並行に行なわれるので両処理の共有資源であるGCスタックと自由リストを扱う場合に排他処理を行なう必要がある。問題になるのは領域獲得関数とリンク書き換え関数である。この部分が排他処理を必要とすることを示すために特別なフラグを立てておき、もしこの部分を実行中にタイマ割り込みがかかっても、この部分の実行が終わるまでGCが実行されないようにする。

飢餓状態にならないためにタイマの割り込み間隔とセルの消費速度、分割GCの一回の処理量の間に満たされなければいけない条件、すなわち安全条件がある。簡単にいうとリスト処理の間に消費されるセル数を上回るようにGCの処理量を設定しなければならない。

リスト処理の時間、つまりタイマの割り込み間隔を小さくとればその間に消費されるセルの数を少なく、一回のGCの処理量も小さくできる。逆に間隔を大きくとれば一回のGCの処理量を大きくする必要があるのでGCによる中断時間も大きくなる。

一回の分割GCの処理量、割り込み間隔、セルの消費速度、実行中の使用中セルの間には飢餓状態を生じないための条件がある。これについては3.2節で述べる。

以上の説明から知れるようにインクリメンタルGCがメモリー空間の閾値に依存したGCであるのに対して、Timed-GCは時間(すなわちタイミング)に依存したGCであると言えよう。

(3) GCの概念的な比較

図1は適切なパラメータを設定した場合の各GCによる実行の様子を概念的に表わしていて"－"はリスト処理を"●"はGCの起動を"＊"はGC処理を表す。

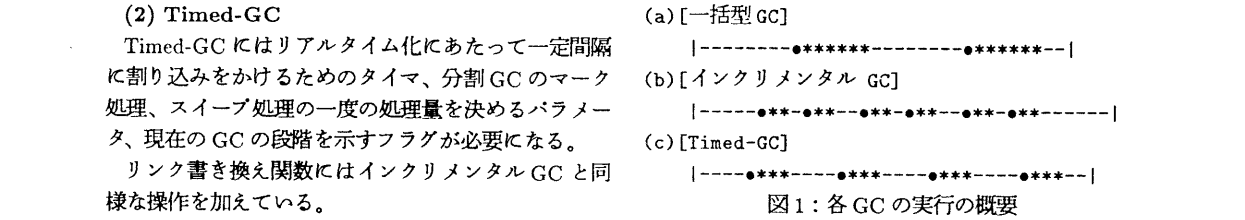

図1：各GCの実行の概要

(a)の一括型GCではGCの起動によりリスト処理に長時間の中断が生じるがGC起動回数は少ない。

一方(b)のインクリメンタルGCではGCによる中断はリスト処理中に分散されているので見掛け上は連続してリスト処理が行われていると考えられる。しかし実行全体を見るとGCが集中して呼び出されている部分もありリスト処理の応答性が悪化するので処理効率の観点からリアルタイム性を十分に満たさないこともある。(c)のTimed-GCの場合には一定間隔で一定時間のGCが実行全体に渡って均一に分散する。

リアルタイムGCは一括型と比較してGC起動回数が多くなるので実行時間が大きくなる。

リスト処理の効率だけの点から考えるとインクリメンタルGCではリスト処理中に領域獲得毎に自由セル数と閾値との比較とカウンタの減算を行なうのに対し、Timed-GCではリスト処理中にほとんどオーバーヘッドを持たない。インクリメンタルGCでは残りの自由セル数が閾値以下になると、領域獲得の際に分割GC

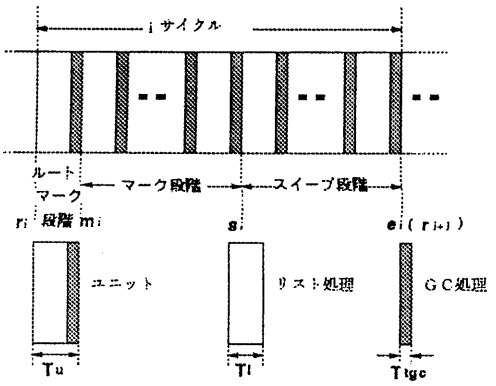

図 2: Timed-GC のユニットを考えた実行モデル

が行われるので、その間のリスト処理効率が低下する。一方 Timed-GC ではリスト処理と GC 処理が独立に扱われるのでリスト処理の効率は高い。このことはリアルタイム性の観点から好ましい性質である。

3 Timed-GC の解析

Timed-GC を備えた Scheme の実行モデルを考え、そのアルゴリズムを解析する。

3.1 Timed-GC の実行モデル

モデルの作成にあたりユニットというものを導入する。ユニットはリスト処理と分割 GC との組み合わせたものであり、実行はこのユニットを繰り返し行うことで進行する (図 2)。図 2 は i 回目のサイクルを表している。一つのサイクルはルートマーク段階、マーク段階、スイープ段階からなる。ユニットはこれらのどれかに所属する。

ルートマーク段階のユニットの分割 GC はルートのマーク付けを行ない、すぐにマーク段階に移る。ルートマーク段階には 1 つのユニットしかない。

マーク段階ではスタックを用い再帰的にマーク処理を繰り返す。1 ユニットの分割 GC では一定数 K_m セルのマークを行う。あるユニットの分割 GC のマークの途中でスタックが空になるとそのユニットは終了しスイープ段階に移る。マーク段階開始時に使用しているセル数を $A(m_i)$ とするとマーク段階のユニット数は $\lceil \frac{A(m_i)}{K_m} \rceil$ である。

スイープ段階ではヒープ上の全セルを走査する。1 ユニットの分割 GC では一定数 K_s 個のセルをスイープする。その際ゴミセルは自由リストにつないでいく。あるユニットの分割 GC のスイープ中に全セルの走査が終るとそのユニットは終了し再びルートマーク段階になる。スイープ段階のユニット数は $\lceil \frac{N}{K_s} \rceil$ である。

よって 1 サイクルの間には $1+\left\lceil \frac{A(m_i)}{K_m} \right\rceil + \left\lceil \frac{N}{K_s} \right\rceil$ 個のユニットが実行される。

ここでマークとスイープの 1 度の処理量 K_m と K_s の間には、本アルゴリズムがリアルタイム性を満足する条件がある。以下、それについて述べる。

3.2 アルゴリズムの安全性の条件

ユーザープログラム実行中に飢餓状態を生じないための条件を求める。すなわちアルゴリズムの安全性の条件を求める。まず、解析するにあたり条件として

- セルの消費率は実行を通じて一定である。

とし、この単位時間あたりのセル消費率を C とする。

今、i サイクルが $t = r_i$ でルートマーク段階になり、$t = m_i$ でマーク段階、$t = s_i$ でスイープ段階になり $t = e_i$ で終了し同時に $i+1$ サイクルのルートマーク段階になるとする。

ユニットの実行時間を T_u、その内リスト処理の時間を T_l、呼び出しのオーバーヘッドを含めた分割 GC にかかる時間を T_{lgc} とする。1 個のセルのマークにかかる時間 (T_m) とスイープにかかる時間 (T_s) とは異なるので、マークとスイープを同数のセル行なったのでは、T_{lgc} が異なる。そのため T_{lgc} を等しくなるようにマーク処理は K 回、スイープ処理は kK 回行う。

1 ユニットで消費されるセル数はリスト処理の時間が T_l なので CT_l である。よって 1 サイクルのマーク段階とスイープ段階で消費されるセルの数は

$$CT_l\left(\left\lceil \frac{A(m_i)}{K} \right\rceil + \left\lceil \frac{N}{kK} \right\rceil\right) \tag{1}$$

また 1 サイクルで回収できるセルは

$$N - F(m_i) - A(m_i) \tag{2}$$

となる。ここで N はヒープ領域にある全セルの数、$F(m_i)$ は i サイクルのマーク開始時の自由セルの数を表す。

i サイクル終了時の自由セル数 $F(e_i)$

$$F(e_i) = N - A(m_i) - CT_l\left(\left\lceil \frac{A(m_i)}{K} \right\rceil + \left\lceil \frac{N}{kK} \right\rceil\right) \tag{3}$$

さて上の式 (3) は 1 サイクルの間にセルが飢餓状態になって一括 GC が起動されない場合に成り立つ。

しかし、実際には 1 サイクルの中でルートマーク段階、マーク段階では自由セルは減少し、スイープ段階では減少と増加を繰り返す。**1 サイクルで飢餓が生じない条件はそのサイクルでの自由セルの最小値が必ず正であることである。**1 サイクルで自由セル数がとる最小値ついて考える。(図 3)

ルートマーク段階、マーク段階ではセルは単調減少する。スイープ段階開始時の自由セル数は

$$F(S_i) = F(r_i) - CT_l\left(1 + \left\lceil \frac{A(m_i)}{K} \right\rceil\right) \tag{4}$$

スイープ段階では各ユニットで自由セル数は減少と増加を繰り返しながらゴミセルの回収を行う。しかしそのユニットの分割 GC でゴミセルが回収できるかで

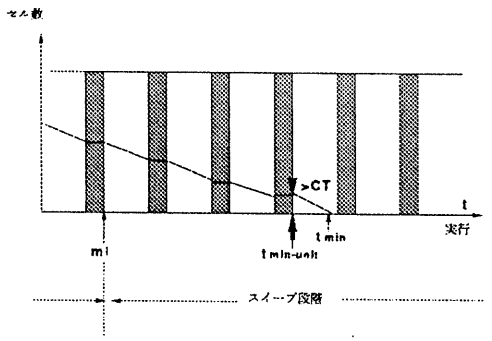

図 3: 自由セル数の最小値をとる場合

きないかはヒープ上のセルの配置による。最悪の場合には使用中のセルと自由セルがヒープの上方に偏っていて、スイープ段階に入ってからの最初の何回かのユニットではゴミセルの回収がまったく行われず、自由セルの消費のみが進行する。

回収不可能なセルは自由セル又はマーク済のセルである。つまり $A(m_i)+F(m_i)$ 個の回収不可能なセルがヒープの上方にある場合には, そのセルを走査してしまうまで、ゴミセルを全く回収できないことになる。この場合は最初にゴミセルを回収できるのは、スイープ段階に入って $\left\lceil \frac{A(m_i)+F(m_i)}{kK} \right\rceil$ 回目のユニットの分割GC実行中である。つまり、自由セル数が最小になるのは $\left\lceil \frac{A(m_i)+F(m_i)}{kK} \right\rceil$ 回目のユニットのリスト処理中である。このユニットのリスト処理中に飢餓を生じないためには, **そのユニットの開始時の自由セル数** $F(t_{min-unit})$ が CT_l **より大であればよい。**

$$F(t_{min-unit}) > CT_l \tag{5}$$

より、

$$F(r_i) - CT_l\left(\left\lceil \frac{A(m_i)+F(m_i)}{kK} \right\rceil + \left\lceil \frac{A(m_i)}{K} \right\rceil\right) > CT_l \tag{6}$$

である。ここで以後の計算を簡略化するために、

- $A(m_i)$ は K の倍数
- N と $A(m_i)+F(m_i)$ が kK の倍数

であるとする。また $F(m_i) = F(r_i) - CT_l$ である。すると式 (6) の条件は

$$F(r_i) > A(m_i)\frac{\frac{CT_l}{K}(1+\frac{1}{k})}{1-\frac{CT_l}{kK}} + CT_l \tag{7}$$

$$1 - \frac{CT_l}{kK} > 0 \tag{8}$$

となる。これで 1 サイクルに飢餓状態が生じないための条件が得られた。

次に連続するサイクルの中で飢餓を生じない条件について考える。その条件は『**全てのサイクル** i **に対してそのサイクル終了時の自由セル数が次のサイクル開始時の自由セル数についての条件を満たす**』である。

式 (7) より

$$F(r_{i+1}) > A(m_{i+1})\frac{\frac{CT_l}{K}(1+\frac{1}{k})}{1-\frac{CT_l}{kK}} + CT_l \tag{9}$$

また式 (3) より

$$F(r_{i+1}) = F(e_i) = N - A(m_i) - CT_l\left(\frac{A(m_i)}{K} + \frac{N}{kK}\right) \tag{10}$$

よって

$$N - A(m_i) - CT_l(\frac{A(m_i)}{K} + \frac{N}{kK}) > A(m_{i+1})\frac{\frac{CT_l}{K}(1+\frac{1}{k})}{1-\frac{CT_l}{kK}} + CT_l \tag{11}$$

以上から次の安全性の条件が得られる。

安全性の条件 A

すべての i に対して以下の式を満たすようにパラメータを決定することがアルゴリズムの安全性の条件、すなわち実行中に飢餓状態に陥らないことの条件になる。

$$N > A(m_i)\frac{1+\frac{CT_l}{K}}{1-\frac{CT_l}{kK}} + A(m_{i+1})\frac{\frac{CT_l}{K}(1+\frac{1}{k})}{(1-\frac{CT_l}{kK})^2} + \frac{CT_l}{1-\frac{CT_l}{kK}}$$

さらに安全条件 A を簡略化した条件として以下の条件が考えられる。

安全性の条件 B

A_{max} は $A_{max} = max\{A(m_1), \ldots, A(m_n)\}$ であるとする。この時安全条件 A は以下のように簡略化される。

$$N > A_{max}\frac{1+2\frac{CT_l}{K} - k(\frac{CT_l}{kK})^2}{(1-\frac{CT_l}{kK})^2} + \frac{CT_l}{1-\frac{CT_l}{kK}}$$

安全条件 A と B では明かに B の方が強い条件である。しかし $A(m_i)$ は特定が困難であるので、以下ではパラメータ設定に使用する条件としては安全条件 B を用いる。一般には全セル数 N、割り込み間隔 T_l は与えられ、C、k については試行で求めることができる。よって安全条件 B を満たすような分割 GC の処理量 K の値の下限を得られる。得られた下限値は安全に実行するために必要な GC の処理量を表わしている。この K を大きくとると無駄な GC が生じることになり、分割 GC による中断時間が増す。

ここで条件 B では実行中の最大マーク開始時使用セル数をパラメータ設定に用いているので、安全ではあるが $A(m_i)$ が実行中に大きく変動する場合には、A_{max} を用いて設定されたパラメータ K の値は冗長なものになり、実行の効率は低下することが考えられる。

4 Timed-GC を備えた Scheme 処理系の試作

リスト処理部

Timed-GC のアルゴリズムの評価に使用した Lisp 処理系のベース言語には Scheme を用いた。Scheme は静的なスコープを持ちその意味が表示的意味論で与えられている Lisp の一方言である。

試作したScheme処理系の言語仕様は[4] *Revised*[4]に基づく[1]。また処理系はインタプリタベースであり、SunOS上でC言語とアセンブリで記述されC言語のソースが約9000行、アセンブリのソース[2]が40行程度である。

GC部

Timed-GCには主に以下の3つの機能が必要になる。

1. タイマによる割り込み処理

Timed-GCはタイマにより一定間隔で分割GCを行なう。タイマ割り込みにはUNIXシステムコールsetitimerを用いた。タイマ割り込みによりGCの処理を行うルーチンt_gc_handler()が呼びだされる。

2. GCスタックを用いたマーク処理とスイープポインタの導入

分割GCを行うためにマーク処理はGCスタックを用いて再帰的に行う。GCスタックが空になった時にマークすべきセルがなくなったことになる。またスイープ処理において走査をヒープ領域のどこまで行なったかを示すポインタを大域変数で実現する。

3. 排他処理

GC処理はリスト処理と並行して行われるので、領域獲得関数とリンク書き換え関数を排他処理する必要がある。

この関数の排他処理したい部分の先頭に排他処理を必要とすることを示すフラグを立てておく。GC処理側ではGC処理を行うルーチンを実行する前にフラグが立っているかどうかチェックしその場合にはフラグが下がるまでGC処理は行なわないようにする。

この他にその時点のGCの段階を表わすフラグを大域変数で実現している。

5 Timed-GCの実験評価

試作した処理系でGCを評価するためにベンチマークプログラムを実行した結果、実行時間、リスト処理の効率の点からTimed-GCがリアルタイムGCとして優れていることが示された。しかし現在のような安全条件Bを用いたパラメータKの設定方法では実行のオーバーヘッドが大きくなってしまう例があることも分かった。以下詳細について述べていく。また今回は比較のために一括型GC、インクリメンタルGCを備えた処理系も同時に試作し評価している。

5.1 評価プログラム

各GCを評価するプログラム[5]として今回は以下の4種類を選んだ。それぞれ以下のような性質をもつプログラムである。

- マーク開始時使用セル数、セルの消費速度が一定。

```
P1 (list-tarai '(1..10)'(5..10)'(10))
                                D= 899709
P2 (srev '(1...9))              D=1944255
P3 (tarai 9 4 0)                D= 621333
P4 (fib 25)                     D= 120014
```

P1はリストのたらい回し関数である。P2はリストの要素を反転する関数である。P3はたらい回し関数である。P4はフィボナッチ数を求める関数である。

Dは各プログラムの実行中に使用された延べセル数を表す。

5.2 パラメータの設定

Timed-GCではマーク、スイープの処理量K_m,K_sを設定する必要がある。このパラメータは安全条件Bを満す最小のものを選んだ。ただし$T_l = \frac{1}{30}$秒、$N = 130000$個、kは2とした。A_{max}については理論的に求められるものはその値を、そうでないものは実測値を用いた。

インクリメンタルGCのパラメータについては[2]に基づいて同様に求めた。

5.3 実行時間の計測

求めたパラメータの値に基づいて各GCを備えた処理系で評価プログラムを実行した時間を表1、表2に示す。一括型GCの実行時間はリスト処理時間とGC時間の和である。インクリメンタルGC、Timed-GCともに一括型GCを分割して実行するものであり、一括型GCに比べて分割GCの呼び出し、回数制御等によるオーバーヘッドのために実行時間は一括型GCに比べて長くなる。そのために一括型GCの実行時間は最短の実行時間であると考えられる。

表1：評価プログラムの実行時間 (sec)

	一括型GC		TimedGC	Inc-GC
	実行時間	GC時間	実行時間	実行時間
P1	10.15	0.70	10.84	11.56
P2	20.24	1.64	21.61	23.44
P3	5.45	0.43	5.93	6.70
P4	25.74	2.43	28.34	30.59

表2：一括型GCに対する各GCのオーバーヘッド

	Timed-GC		Incremental-GC	
	差	差 / 一括型GCの実行時間	差	差 / 一括型GCの実行時間
P1	0.69	0.07	1.41	0.13
P2	1.57	0.08	3.40	0.17
P3	0.48	0.09	1.25	0.23
P4	2.60	0.10	4.85	0.19

表2は一括型GCと各GCとの実行時間の差と、その一括型GCの実行時間に対する比率を求めたもので、各GCの一括型GCに対するオーバーヘッドを表す。

表2よりTimed-GC、インクリメンタルGCとも一括型GCに対してオーバーヘッドがみられる。Timed-GCの場合は主に分割GCの呼び出しのコストが、イ

[1] 現在はベクトルはまだ扱えないが今回のGCの評価には支障はない。

[2] call/ccによるコンティニュエイション実現の為

ンクリメンタルGCでは分割GCの呼び出しコストと領域獲得時の自由セル数と閾値との比較が原因である。

Timed-GCの一括型GCの実行時間に対するオーバーヘッドはインクリメンタルGCに対してほぼ半分である。これはTimed-GCの方が分割GCの呼び出しコストが大きいにもかかわらず、インクリメンタルGCの分割GCの呼び出し回数がTimed-GCに比べて多いことと、インクリメンタルGCの場合には領域獲得毎にオーバーヘッドがあるためだと考えられる。

この結果からリスト処理にオーバーヘッドを持たないTimed-GCが実行時間についてはインクリメンタルGCよりも優れていることが分かる。

5.4 リアルタイム性

試作した各GCを備えた処理系のリアルタイム性について考察する。その指標としては

- 単位時間あたりの消費セル数の変化

を用いる。消費セル数の変化からリスト処理の効率とGCによる中断が分かる。

サンプルプログラムはP1(リストのたらい回し関数)である。P1はマーク開始時使用セル数が小さく、全セルの1.5パーセント程度であり、また実行を通してその数が一定のプログラムである。またセルの消費率は実行を通じてほぼ一定である。消費セル数の変化はプログラムの実行過程において$\frac{1}{100}$秒毎に消費されたセル数をカウントすることで調べる。

この方法によればGCが長時間続いた場合にはセルの消費が0であり一定時間リスト処理が行なわれないことが分かる。また消費されるセルの数が少ない場合にはリスト処理の効率が低下していると判断できる。

以下グラフは横軸には割り込みの回数(経過時間)、縦軸にはその間に消費されたセルの数を示している。

P1に対する消費セル数の変化

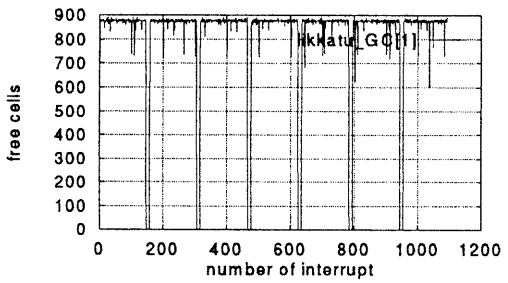

図3: P1に対する一括型GCの消費セル数

一括型GCの場合(図3)には、実行を通して一定間隔のGCが生じていることが顕著である。しかし単位時間当りのセルの消費が大きいことからリスト処理は高速に行なわれている。

インクリメンタルGCの場合(図4)にも一定の割合で谷間が生じている。この谷間にはリスト処理とGCが並行に行われていることが、消費セル数が減少してることから分かる。またGCが行われてない場合にも一括型に比べ消費セル数が小さいことから、リスト処理中に領域獲得のオーバーヘッドがあることが分かる。

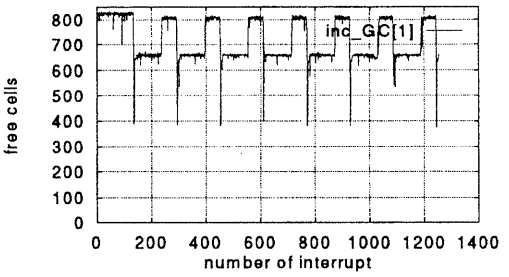

図4: P1に対するインクリメンタルGCの消費セル数

Timed-GCの場合(図5)ではリスト処理に長時間の中断や効率の変化は見られない。実行全体に渡ってほぼ一様な実行が行われている。更に細かくサンプリングをすればGCとリスト処理が交互に行われていることが分かるはずである。

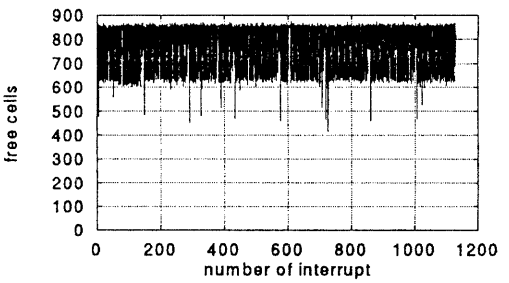

図5: P1に対するTimed-GCの消費セル数

このP1の評価プログラムの実行結果を見る限りTimed-GC実行全体で一定時間に一定の処理能力を確保していることが分かる。このことはリスト処理効率が実行中に変化するインクリメンタルGC等に比べてリアルタイム処理に向くことを表していると考えられる。

5.5 Timed-GCのパラメータ設定の問題

これまでの結果からTimed-GCがリアルタイムGCとして優れたものであることがわかったが現在の安全条件Bに基づいたパラメータ設定では実行のオーバーヘッドが大きくなってしまう例がある。以下の簡単なプログラムを実行し、表3、表4の結果が得られた。

```
(define (make-data x)
 (sum x '()))
(define (sum x data)
  (cond ((zero? x) data)
        (else (sum (- x 1) (cons x data)))))
```

上のプログラムは引数に指定された数の要素を持つconsリストを生成するプログラムである。評価の間には常にそのconsリストを保持しているので使用セル数は実行の進行とともに線型に増加し終了時には引数として指定した数の2倍のセルが使用中セルになる。またセルの消費速度は実行を通じてほぼ一定である。

```
P5 (make-data 20000)          D=240064
P6 (make-data 30000)          D=360014
P7 (make-data 40000)          D=480014
```

表3：評価プログラムの実行時間 (sec)

	一括型GC		TimedGC	Inc-GC
	実行時間	GC時間	実行時間	実行時間
P5	2.36	0.39	2.79	2.79
P6	3.58	0.65	5.28	4.26
P7	5.24	1.28	9.44	6.32

表4：一括型GCに対する各GCのオーバーヘッド

	Timed-GC		Incr-GC	
	差	差 / 一括型GCの実行時間	差	差 / 一括型GCの実行時間
P5	0.42	0.18	0.41	0.17
P6	1.70	0.47	0.68	0.19
P7	4.20	0.80	1.08	0.21

make-dataはP1からP4とは異り、マーク開始時使用セル数の最大値と最小値の差が実行中にかなり大きく変動するプログラムである。

表4から分るようにこのプログラムに対してはかなりオーバーヘッドが大きくなっている。

この原因はTimed-GCの、マークとスイープの一回の処理量を表すパラメータKの設定にあたって安全条件B、すなわち評価プログラム実行中の最大マーク開始時使用セル数A_{max}を用いていることにある。

このプログラムでは最大マーク開始時使用セル数は実行の最後にとり、それ以外の場合にはそれよりも小さい。安全性を考えて設定されたKは実行の最終段階に合わせてあり、それ以外の場合には無駄なスイープが多くなり実行効率は低下する。

このようにマーク開始時使用セル数が大きく変動するようなプログラムに対しては現在のTimed-GCのパラメータ設定方法では問題がある。しかし、この問題に対してはパラメータの設定を実行中に動的に行うことである程度解決できると考えられる。

6 まとめと今後の課題

本研究ではTimed-GCの実現アルゴリズムを与え、Scheme処理系にそれを実装し、他のGCとの比較をしその評価を行なった。

その結果、Timed-GCはインクリメンタルGCに比べて、オーバーヘッドが小さくリアルタイム性に優れていることも分った。

しかしマーク開始時使用セル数が実行中に大きく変動するようなプログラムに対しては現在のパラメータ設定方法ではオーバーヘッドが大きくなってしまう例のある。しかしそれに対してはパラメータの設定を実行中に動的に行うことで解決できると考えられる。

今後の課題としてはマーク&スイープ法以外のGC戦略へのTimed-GCの導入、マルチプロセッサ上のTimed-GCの検討等が考えられるが特に

- Timed-GCとインクリメンタルGCの融合による時空間的なGCの検討

が興味深い。特にインクリメンタルGCにおけるオーバーヘッドの大きな要因である領域獲得毎の閾値と自由セル数との比較とTimed-GCにおけるタイマ割り込み機能のハードウェアによるサポートができれば、空間的な観点から設計されたインクリメンタルGCと時間の観点から設計されたTimed-GCの双方の良さを持った理想的な時空間的リアルタイムGCを実現できる。またこのサポートするハードウェアの実現は困難ではないと考える。

具体的にはGCの最初の起動はインクリメンタルGCと同様にして自由セル数と閾値との比較で行ない、起動後はTimed-GCのように一定時間毎にタイマ割り込みで分割GCをリスト処理と並行して行なう。

その実行の様子は5.4節の消費セル数の変化のグラフで予想してみると図8のようになる。

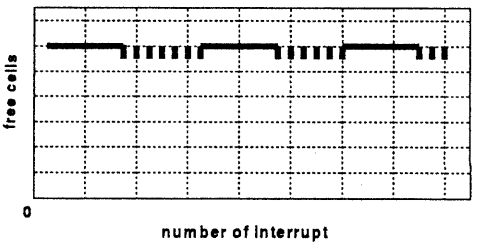

図8：時空間的なリアルタイムGCの実行

参考文献

1) Henry G.Baker,Jr.*List Processing in Real Time on Serial Computer*, Communicaton of the ACM.,Vol.21,No 4,April 1978

2) Taiichi YUASA.*REAL-TIME GARBAGE COLLECTION ON GENERAL-PURPOSE MACHINES*,The Journal of Systems and Software.,Vol.11,No 3,march 1990

3) Takayasu ITO.*Lisp and Parallelism*,Artificial Intelligence and Mathematical Theory of Computation,Papers in Honor of John mcCarthy,(ed. V.Lifshitz),187-206,Academic Press(1991)

4) William Clinger and Jonasan Rees, editors. *Revised[4] report on the algorithmic language Scheme.*, LISP Pointers,IV(3):1-55,July-September 1991.

5) 奥野 博. 第3回 *Lisp* コンテストおよび第1回 *Prolog* コンテストの課題案. 情報処理学会研究報告、記号処理 28-4、June 1984.