プログラミング 16-7
(1997. 11. 21)

GNU Emacs への世代別ごみ集めの実装

小林 広和　　寺田 実

東京大学大学院工学系研究科

GNU Emacs はテキストエディタとして世界中で、多くの人々に使われている。その GNU Emacs は Emacs Lisp という Lisp 言語で書かれているが、Emacs Lisp のごみ集めはマークスイープ法であり、ごみ集めの処理によって起こる通常処理の中断時間によりシステムの応答性の低下が起きている。ごみ集めによって起こる処理の中断時間を短縮し、システムの応答性を向上させるために世代別ごみ集めを Emacs に実装した。

本実装の特徴は世代間参照の検出方法に仮想メモリのダーティビット情報を利用することによって、通常処理の速度低下を招かず、既存のシステムに大きな変更を加えること無くごみ集めによる処理の中断時間を短縮することができたことである。本稿では、今回行なった世代別ごみ集めの実装法と、性能計測について報告する。

Implementation of Generational Garbage Collector in GNU Emacs

Hirokazu KOBAYASHI　　Minoru TERADA

Faculty of Engineering,University of Tokyo

GNU Emacs is one of the most popular text editor, and used by many people. GNU Emacs is written in a Lisp language of Emacs Lisp, but the Garbage Collector of Emacs Lisp is mark-sweep collector and the mutator pauses by collector make system response worse. To reduce the mutator pauses by collector and to promote response of the system, we implemented Generational GC in Emacs.

The feature of this implementation is that by using of dirty bit of Virtual Memory for tracking intergenerational references, without mutator overhead, and keeping the cost of implementation of Generational GC low, we reduce pause time. In this article , we report the method of implementation of this Generational GC and measurement of its performance.

1 はじめに

GNU Emacs (以下 Emacs と略す) は Emacs Lisp と呼ばれる Lisp 言語で作られている。Emacs は世界中で広く使われているテキストエディタで、多くの利用者が存在する。そのために Emacs で動作する Lisp アプリケーションは様々なものが作られていて、テキストの種類に応じたエディットモードの他にも、ニュースリーダやメイルリーダ、WWW ブラウザー、表計算などがある。

Emacs はメモリ管理のためにごみ集めを行なうが、現在の Emacs で行なわれているごみ集めは停止型のマークスイープ方式であり、この方式はすべての処理を中断してごみ集めを実行するために、ごみ集めの処理を始めるとユーザにとってはインタラクションが不意に停止することになる。このような不意に起こるインタラクションの停止によって、ユーザは不快な思いをしアプリケーションの使い勝手が低下する。したがって、ごみ集めによって起こる処理の中断時間をなるべく短くし、ユーザの操作に支障の無いようにするというのは研究の目標となる。

ところで、Emacs はインタラクティブに用いられるがロボットの制御のように常にリアルタイム性が強く要求されるわけではない。このことを考慮に入れて、ごみ集めにかかる時間の短縮を行なうために、世代別ごみ集めを Emacs に導入することにした。Emacs に世代別ごみ集めを実装することで、ごみ集めによる処理の中断が耐え難いほど長時間にわたる頻度を少なくすることができる。

世代別ごみ集めの実装 [3, 4] では、ヒープの構造、殿堂入りのポリシー、ごみ集めのスケージューリング、世代間参照の検出方法などの問題がある [5]。本稿では、特に世代間参照を検出するためのオブジェクトの書き換えの検出方法として、仮想メモリのダーティビット情報を用いた実装を行なった。そして、Emacs のような広く使われている実用システムにおいて、世代別ごみ集めの実装でどのように性能が改善されるかを計測した。

2 GNU Emacs の概要

GNU Emacs は Emacs Lisp という Lisp で書かれたテキストエディタであり、Emacs Lisp は C 言語で書かれている。Emacs Lisp 処理系はエディタ内で使用するために設計された言語なので、ファイルやバッファ、ディスプレイ、サブプロセスなどを扱う機能や、テキストをスキャンしたりパースしたりするための特別な機能をもつ。また、多くの種類のオブジェクトを持ち、その中にはベクターのような可変長のオブジェクトも持つ。また、動的スコープを持っていて、浅い束縛を用いて実装を行なっている。さらに、Emacs Lisp のごみ集めはマークスイープ方式であるが、一部のオブジェクトタイプについては領域圧縮を行なっている。

本稿で用いた Emacs は GNU Emacs version 19.28 ベースの mule 2.3 である。

3 世代別ごみ集め

世代別ごみ集めとは「ほとんどのオブジェクトは生成後間もなく死に、ある程度生き続けたオブジェクトはさらに長期にわたって生き残る」という観察結果に基づいている [3, 4]。生成後間もないオブジェクトを若いオブジェクトという。つまり、若いオブジェクトは短寿命である可能性が高いということである。この性質を利用することで、若いオブジェクトだけをごみ集めの対象とすることで、ごみを短時間で効率良く集める手法が世代別ごみ集めである。

4 Emacs への世代別ごみ集めの実装

Emacs に世代別ごみ集めを実装する場合に次のようなことを考慮しなければならない。

1. Emacs はマークスイープ法のごみ集めを採用しており、現在のごみ集めの実装方式を前提としてその他の部分も実装している。したがって、違うごみ集めを実装する場合には Emacs のごみ集めのコード以外のコードの変更が必要であり、その変更の量は実装するごみ集めによって変化する。特に、Emacs のソースコードは十万行以上におよぶので、ソースコードの書換えの場所を減らしたい。

2. Emacs はオブジェクトのアドレスが移動しないことを前提にして実装している部分があるのでその点の注意が必要である。

3. Emacs では、世代別ごみ集めで必要なライトバリアが実装されていないので、それを実装する必要がある。また、Emacs は動的スコープであり静的スコープの処理系に比べオブジェクトの書換えが多くなる。そのため、世代別ごみ集めの実装で必要なライトバリアの動作によりミューテータの効率が低下する。

4. 通常の Lisp 処理系はごみ集めはメモリをアロケートした場合に開始される場合が多いが、Emacs は、Eval を呼び出した時か、ユーザからの入力が一定時間以上なくなった時にごみ集めが開始される。

以上のようなことを考慮して、Emacs に世代別ごみ集めを次のように実装することにした。

1. 生成領域と長寿命領域の 2 つの世代に分け、それぞれの領域を 1 つずつ用意することにした。生成領域でのごみ集めを一回生き残ったオブジェクトはすべて長寿命領域へと移動することにした。

2. 長寿命領域のごみ集めをユーザの入力がない時に行なうことにより、長時間にわたる処理の停止時間をユーザから目立たないようにする。

3. 長寿命領域のごみ集めは、生成領域のごみ集めを行ない生成領域が空になった後に実行する。したがって若い世代から古い世代を指すポインタを探す必要はなくなる。また、長寿命領域は Emacs に元から実装されていたマークスイープ法のごみ集めをそのまま用いることができるので実装の負担が軽くなった。

4. Emacs ではメモリをアロケートする時にごみ集めを実行するのではなく、Eval からごみ集めを呼び出しているので、「ある一定量以上のメモリをアロケートしようとする時、ごみ集めを実行する」ということができない。したがって、生成領域の大きさは可変とすることにした。このことにより、マシンのスピードやアプリケーションの種類により生成領域の大きさをユーザが自由に変更できる機能をそのまま生かすことができた。しかし、あるポインタが生成領域を指すかのチェックに Sparc プロセッサのマシンコードで 8 命令必要であり、生成領域の大きさが固定の時の 2 命令 [6] に比べて 4 倍かかる。

5. ライトバリアは Page Marking を利用する。本研究では、Page Marking の実装法として特に Solaris 2 で提供されている仮想メモリのダーティビットを読み出す機能を利用することにする。これを利用する利点は、ポインタの書換えが多い場合にライトバリアによるミューテータのオーバヘッドが少ないという点と、ソースコード上でオブジェクトのポインタの書換え場所にライトバリアのためのコードを挿入する必要がないので、ソースコードの変更箇所が少なくなる。しかし、Page Marking は、ページ毎に書き込みの有無を検出する方法である。標準的なマシンでは 1 ページは 4096 バイトであり、書き込みのあったオブジェクトを検出するには大き過ぎ、実際に必要なオブジェクト、つまり若い世代を指しているオブジェクトを得るのに時間がかかってしまうという欠点がある [1, 6]。

Emacs Lisp 処理系本体への変更点は主に、ごみ集めをコピー方式にしたためにおこった。つまり、ごみ集め実行前と実行後でオブジェクトのアドレスが変化するために、ごみ集め実行時にスタック上に存在する変数をごみ集めのルートに追加する必要が出た。また、Emacs では、アドレスが移動しないオブジェクトへのポインタが入っている大域変数で、その変数が指しているオブジェクトが他のオブジェクトから指されているとわかっているものにもごみ集めのルートとなっていないものがいくつか見られたが、これらもルートに追加しなければならなかった。これらの変更はソースコードで 46 箇所あった。

4.1 Solaris 2のページダーティビット読み出し機能

ダーティビットはOSが仮想メモリのスワップやページングに使用するもので、一般的にはユーザからはアクセスする手段が用意されていないが、Solaris 2では、それをユーザからアクセスする手段が用意されている。

このダーティビット読み出し機能を利用することにより、Page Markingを実装する場合、従来利用されていたメモリをプロテクトしライトフォールトをシグナルハンドリングするという方法を用いずに済む。メモリをライトプロテクトする方法ではOSの提供するシグナル機構を使用するため、ライトバリアにかなりのオーバヘッドが存在するが、ダーティビットで代用するとシグナルハンドラの処理にかかっていた時間が節約できる。

しかし、Solaris 2での実装の場合、必要な情報、つまり古い世代の領域のダーティビット情報だけを読み出すことができず、そのプロセスが持つすべてのメモリのすべての情報を読み出してしまうので、ダーティビット読み出し時の速度の低下を招いている。

5 性能計測

Emacsのような対話型アプリケーションを用いる場合に問題となるのは、ごみ集めによって起こる処理の停止時間であるので、どの程度停止時間が短縮できたかによって評価を行なう。

本稿では、Emacsの停止時間を計測するのにユーザが対話的にEmacsを利用している状況に近付けるために次のような方法を用いた。

1. ユーザのキー操作を記録するプログラム(「打鍵記録プログラム」)を作成し、打鍵記録プログラムでEmacs利用時のユーザのキー操作を打鍵データとして記録する。
2. 記録した打鍵データを再生するプログラム(「打鍵再生プログラム」)を作成し、打鍵再生プログラムで打鍵データを再生することによってEmacsを操作しその時の性能を計測する。

この方法を用いることによって、キーボードからの入力によりアプリケーションが動作するという Emacs の特徴をほとんど損なうことなく、同じ状況での性能の計測を何度も行なうことができる。また、実際にユーザが利用している状況を記録して行なっているので現実の利用状況に即した性能の計測ができる。

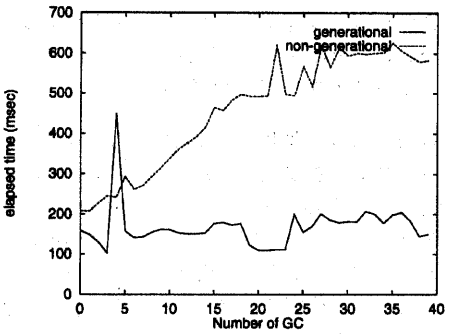

図 1: ごみ集めの停止時間の比較

以下で行なった計測はSparc Station ELC (OS Solaris 2.5)を用いて行なった。

実験のために次のようなアプリケーションの利用をしているユーザの利用状況を記録した。

- Cプログラムのエディット
- Infoの利用
- Makefileの作成
- Gnusによるニュースの読み書き

この時打鍵数は約2600で作業時間は30分であった。この時にごみ集めは40回起こった。

この打鍵記録をベンチマークプログラムとして用いた。この時のgc-cons-thresholdの値は100000バイトである。gc-cons-thresholdというのはevalを呼び出した時に前回にごみ集めを行なった時からアロケートしたメモリの量がこの値を越えていたらごみ集めを行なうという値である。

5.1 停止時間の比較

図1は世代別ごみ集めのEmacsとそうでないEmacsにおいてのごみ集めの停止時間の比較である。横軸はごみ集めの回数を表し、縦軸はその時のごみ集めにかかった時間を表す。このグラフで世代別ごみ集めのEmacsで4回目のごみ集めの時間が長くなっているのは、この時にキーボードからの入力が一定時間以上なかっ

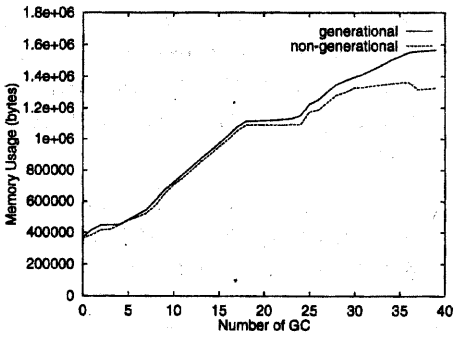

図 2: 使用メモリ量の比較

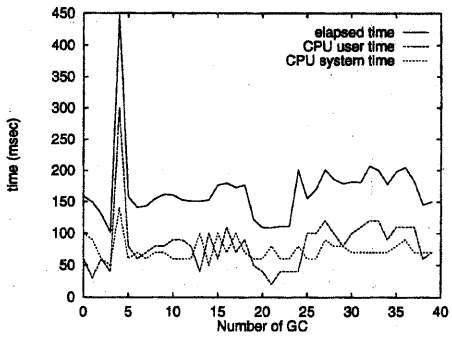

図 3: 世代別ごみ集めの時間

たので長寿命領域のごみ集めが行われたからである。長寿命領域のごみ集めは、生成領域のごみ集めを行なった後に、世代別ごみ集めでないEmacsのごみ集めを行なうのと同等のことを行なっているので、ごみ集めの処理時間も世代別ごみ集めでないEmacsのごみ集めの時間に生成領域のごみ集めにかかる時間を足したものになっている。

5.2 使用メモリ量の比較

図2は使用メモリ量を比較したものであり、縦軸はLispオブジェクトが使用するメモリ量を表す。図1と図2を比較すると、世代別ごみ集めでは、使用メモリ量が増加してもごみ集めにかかる時間の増加がほとんどないことが分かる。また、世代別ごみ集めのEmacsと世代別ごみ集めでないEmacsのメモリ使用量の差は世代別ごみ集めのEmacsで長寿命領域にたまったごみの量を表している。図2から長寿命領域のごみは、多くても生きているオブジェクトの約20%程度であり、ユーザからの入力がない時にだけ長寿命領域のごみ集めを実行するようにしても、ごみが長寿命領域に大量にたまることはないことが分かる。また、オブジェクトの寿命を計測した結果によると、25回目のごみ集め以降で長寿命領域にごみが増えているのは、比較的寿命の長いオブジェクトがアプリケーションの終了とともに死亡してごみになったからである[2]。

5.3 世代別ごみ集めの停止時間

図3は世代別ごみ集めを実装したEmacsでごみ集めを実行する時の、CPUのシステム時間とユーザ時間を調べたものである。世代別ごみ集めでないEmacsでは計測によるとCPUシステム時間はほとんどかからなかったので、世代別ごみ集めのEmacsで使われているCPUシステム時間は、ダーティビットをアクセスするのにかかる時間であると考えられる。したがって、ごみ集めにかかる時間の半分程度がダーティビットのアクセスに使われていることがわかる。

Solaris 2の実装では、ダーティビットの必要なページの範囲を指定できず、共有ライブラリなども含んだそのプロセスのもつ全ページについての情報が得られる。実際にダーティビットの情報が必要なのはヒープ領域だけでそれ以外の情報は必要無い。Emacsを立ち上げた直後に使用しているメモリサイズは約5700Kバイトでそのうちダーティビットの情報が必要なヒープ領域は156Kバイトである。この計測の場合ではヒープの容量が最も大きくなった場合でも約1700Kバイトなので必要な部分だけのダーティビットの情報が得られるように実装されていれば、ダーティビットの読みだしにかかる時間は現在の約25%程度にすることができると考えられる。

5.4 書き換えページ率

図4は長寿命領域にある全オブジェクトページに対する書き換えのあったページの割合を示す。縦軸は書き換えのあった割合である。to-

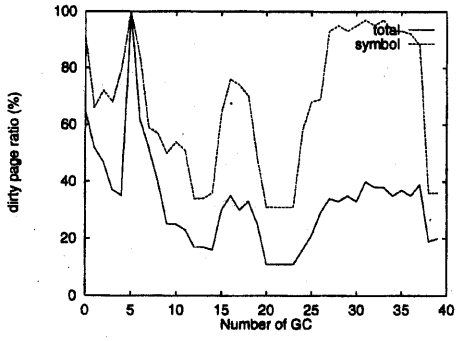

図 4: 書き換えページ率

tal のグラフはオブジェクトの中身としてポインタを持つことができるオブジェクト全種類での書き換えがあった割合で、symbol のグラフは Lisp のシンボルを持っているページで書き換えのあった割合である。図 4を見るとシンボルの書き換えが他のオブジェクトに比べてかなり多いことが分かる。これは Emacs が動的スコープを持つためシンボルの書き換えが多くなるためだと思われる。

6 結論

本稿では、GNU Emacs に世代別ごみ集めを実装する場合の方法および結果を示した。計測から、世代別ごみ集めにすることで使用メモリ量に関わらずほぼ一定の中断時間になることが分かった。また、マークスイープ法に比べ全体のごみ集めによる停止時間が減少しているのが観測され、その時のメモリ使用量もそんなに増加しないことが観測された。

今後の予定として、ライトバリアとしてメモリプロテクトの方法を用いた実装との性能の比較、gc-cons-threshold の値を変更した時の性能の比較などを行ないたいと思う。さらに、長寿命領域のごみ集めを開始するタイミングについても検討を行ないたいと思う。

参考文献

[1] Anthony L. Hosking, J. Eliot B. Moss, and Darko Stefanović. A comparative performance evaluation of write barrier implementations. In Andreas Paepcke, editor, *OOPSLA'92 ACM Conference on Object-Oriented Systems, Languages and Applications*, Vol. 27(10) of *ACM SIGPLAN Notices*, pp. 92–109, October 1992. ACM Press.

[2] 小林広和, 寺田実. Emacs のごみ集め改良の試み. 日本ソフトウェア科学会第 13 回大会論文集, pp. 165–168, 1996.

[3] Henry Lieberman and Carl E. Hewitt. A real-time garbage collector based on the lifetimes of objects. *Communications of the ACM*, Vol. 26(6), pp. 419–29, 1983.

[4] David M. Ungar. Generation scavenging: A non-disruptive high performance storage reclamation algorithm. *ACM SIGPLAN Notices*, Vol. 19, No. 5, pp. 157–167, April 1984.

[5] Paul R. Wilson. Uniprocessor garbage collection techniques. Technical report, University of Texas, January 1994.

[6] Benjamin Zorn. Barrier methods for garbage collection. Technical Report CU-CS-494-90, University of Colorado, Boulder, November 1990.