# スタックフレームの情報を使った世代型ガーベジコレクション

林 芳樹, 寺田 実

平成 14 年 9 月 17 日

## 概 要

本論文では世代型ガーベジコレクションの新しい殿堂入りポリシーを提案、評価する。実験の結果、単純な実装方法では、新しいコードのオーバヘッドが多く、10%-20% の速度低下が起こることがわかった。

## 1 はじめに

Java の浸透にともない、Java、Lisp、ML などガーベジコレクション (GC) [7] 付きのプログラミング言語で書かれたプログラムは急速に増えている。メモリの速度とプロセッサの処理速度の乖離により、効率の良いメモリ管理はますます重要になっている。ゆえに、自動的にメモリを管理する GC の性能はプログラム全体の実行速度に大きく影響する。

GC の中でも非常に重要な手法に世代型 GC [3] がある。世代型 GC はオブジェクトを年齢の近似に従ってわけることにより性能の改善を図る。オブジェクトは最初は新世代に割り当てられる。新世代のオブジェクトは頻繁に回収され、生き残ったオブジェクトはより回収頻度の少ないより古い旧世代に移動する。ほとんどのオブジェクトは短命であるので、オブジェクトは旧世代へはあまりコピーされない。ごみの割合が高い領域を集中的に回収することにより、世代型 GC は非世代型 GC よりも効率良くオブジェクトを回収することができる。

世代型 GC が効率良く動作するためには、多くのパラメータを正しく設定する必要がある。その中でも最も重要なパラメータが殿堂入りポリシーである。殿堂入りポリシーは、どのオブジェクトが旧世代に移動し、どのオブジェクトが新世代に留まるかを決定する。オブジェクトの殿堂入りが早過ぎると、短命オブジェクトが旧世代に移動され、旧世代にごみがたまる。このごみは全領域の回収が行なわれるまで旧領域のスペースを占め続ける。一方、オブジェクトの殿堂入りが遅過ぎると、新世代でオブジェクトが何度もコピーされることになり、コストがかかる。

本論文では、より良いオブジェクトの寿命予測を達成するために、スタックフレームからの到達性を用いた殿堂入りポリシーを提案する。スタックフレームの伸縮はプログラムの実行を反映するので、スタックフレームに基づいた寿命予測は既存の GC 回数に応じたものに比べより正確にできる可能性があると考える。

これ以後は、既存の殿堂入りポリシーの説明と、新しいスタックフレームに基づいた殿堂入りポリシーの説明をする。その後、提案手法の実装と実験結果を記述する。

## 2 既存の殿堂入りポリシー

最も簡単な殿堂入りポリシーは GC を生き残ったすべてのオブジェクトを殿堂入りさせるものである。これには二つの利点がある。一つはオブジェクトが次の世代に早く移動させられるので、何度もコピーするコストがかからないことである。もう一つは新世代のセミスペースは一つですむため、メモリをより効率良く利用できることである。

この方法の主な欠点はオブジェクトの殿堂入りが早過ぎることがあることである。オブジェクトは最初に経験する GC でコピーされるので、コレクション回数は 0 から 1 の間のどれかになる。GC の直前に割り当てられたオブジェクトは殿堂入りしてしまうため、すぐに死にそうなオブジェクトの早過ぎる殿堂入りが発生する。これにより、旧世代は早く一杯になってしまい、メジャーコレクションの回数が多くなってしまう可能性がある。

他の主要な殿堂入りポリシーは GC を $n$ 回 $(n > 1)$

経験したオブジェクトを殿堂入りさせるというのである。この方法の利点は前回の GC 以降に割り当てられたオブジェクトを殿堂入りさせないことで、早過ぎる殿堂入りを避けられるとである。主な欠点は、オブジェクトは殿堂入りする前に必ず $n$ 回コピーされなければならないことである。このコピーによるコストが、早過ぎる殿堂入りをしない利点と相殺する可能性がある。

この方法の一種として、Moher et al. は opportunistic ガーベジコレクタ [8] を提案した。これは、$n$ を適応的に 1 と 2 の間で変えることができる。年齢の精度とコピーのコストの釣合をとることができるので、より良い殿堂入りの決定をできる可能性がある。

他の早過ぎる殿堂入りを避ける手法として、Stefanović et al. による older-first ガーベジコレクタがある [5, 6]。新世代全部を回収する世代型 GC と違い、older-first GC は非常に新しいオブジェクトは回収しない。回収する領域を、少し古い領域に移動させることで、短命オブジェクトは回収前に全部ごみになることが期待されている。この方法を使えば、早過ぎる殿堂入りは避けることができる。しかし、古い領域だけを回収するためには、世代間ポインタの追跡のコストが増えることになる。大抵の場合、短命オブジェクトをコピーしない利点は世代間ポインタの管理のコストで相殺される。

# 3 スタックからの到達性を用いた殿堂入りポリシー

理想的な殿堂入りポリシーは短命オブジェクトは新世代にコピーし、それ以外のオブジェクトを旧世代にコピーするものである。以前の殿堂入りポリシーは GC 経験回数のみに制御されていたので、細い制御は不可能であり、理想的な殿堂入りを達成することはできていない。

ここで提案する新しい殿堂入りポリシーはスタックフレームからの到達性を用いる (静的領域は 0 番目のスタックフレームとみなす)。これを使用することで、殿堂入りを決定するためにより多くの情報を使うことができるようになる。Hirzel et al. はオブジェクトの寿命とスタック、ヒープ、静的領域からの接続の関係を調べた [2]。それによると、スタックからのみ参照されているオブジェクトはほぼすべてが短命で、静的領域から指されているオブジェクトはほぼ死なない。彼らの結果を元に、以下の仮説をたてる。

- 静的領域から到達可能なオブジェクトはほぼ死なない
- 比較的浅いスタックフレームからのみ到達可能なオブジェクトはほぼ短命である
- 比較的深いスタックフレームから到達可能なオブジェクトは短命ではない

つまり、あるスタックフレームよりも深い領域から指されているオブジェクトは殿堂入りさせ、それ以外のオブジェクトは新世代に留める、という方法である。この方法では、殿堂入りをわけるスタックフレームを決める必要がある。本論文では、return 系の操作により到達したことのあるスタックフレームの中で一番深いものを使用する方法と、スタックの半分のところにあるフレームでわける方法を実験した。

# 4 方法

## 4.1 Jikes RVM

提案手法を実装する処理系として Jikes RVM を使用した。Jikes RVM は Java で書かれた高性能な Java Research Virtual Machine である [1]。Jikes RVM は JIT コンパイラのみからなる VM であり、Jikes RVM 自身が Jikes RVM のアプリケーションである。Jikes RVM 用のコンパイラとアプリケーション用のコンパイラは共に base コンパイラを用いた。

## 4.2 ガーベジコレクタ

実験用のガーベジコレクタとして、Jikes RVM に付属の世代型コピー GC と、新たに実装した四つのコレクタを使用した。各コレクタの説明は表 1 を参照。

実験では、新世代用のセミスペースを 5MB、旧世代用のセミスペースを 5MB、大きなオブジェクト用のラージオブジェクトスペースを 10MB とった。

新世代領域が一杯になったときにマイナーコレクションが行なわれ、マイナーコレクションの実行後、旧世

| 名前 | 殿堂入りポリシー | 説明 |
|---|---|---|
| copyGen | マイナーコレクションを生き残る | Jikes RVM 付属の GC |
| stackGen | 静的領域、指定フレームより上のスタックから到達可能 | |
| stackGenHalf | 静的領域、スタックフレームの上半分から到達可能 | |
| stackGenNoTrace | マイナーコレクションを生き残る | フレーム追跡以外は stackGen と同じ |
| trace | マイナーコレクションを生き残る | copyGen + フレーム追跡 |

表 1: ガーベジコレクタ

代領域の残りが 2.5MB 以下になっていればさらにメジャーコレクションを行なう。

stackGen は、提案手法を実装したガーベジコレクタである。殿堂入りを判定するためのフレームを決めるために、return 系のバイトコードの所で、スタックフレームの深さを調べるコードが追加されている。さらに、スタックフレームを 2 回スキャンするようになっている。

stackGenHalf は、半分のフレームを調べるため、旧世代、新世代のコピー様にスタックフレームを 3 回スキャンしている。

stackGenNoTrace は stackGen から、フレーム追跡とそれを使用した殿堂入りコードを削除したものである。

trace は copyGen に stackGen のフレーム追跡用コードを加えたものである。

### 4.3 ベンチマーク

ベンチマークは表 2 の様に、SPECjvm98 [4] の四つのプログラムを使用した。

| ベンチマーク | 説明 |
|---|---|
| compress | 修正 Lampel-Ziv 法 (LZW) |
| jack | パーザ生成器 |
| jess | Java Expert Shell System |
| mtrt | マルチスレッドレイトレーサ |

表 2: ベンチマークプログラム

### 4.4 ハードウェア

実験結果は、IBM ThinkPad X22, Pentium 3 800MHz プロセッサ、メモリ 640 MB、Linux 2.4.18 で計測した。

## 5 実験結果

各 GC の実験結果は表 3 の通り。実行は 5 回行ない、その中最も短いものを使った。

表 3 からわかるように、stackGen コレクタは copyGen コレクタに比べて、10%-20% の速度低下がみられる。stackGenNoTrace コレクタ、trace コレクタと copyGen コレクタを比較すると、速度低下の主な原因はスタックフレーム追跡コードと、追加のスタックスキャンであることがわかる。

## 6 まとめ

本論文では、新しい殿堂入りポリシーの提案、評価を行なった。単純な実装ではオーバヘッドによりプログラムの実行速度が低下してしまうことがわかった。今後の課題として、新しい殿堂入りポリシーの寿命予測の精度を調べることと、実装方法を改善して、オーバヘッドを少なくするようにすること必要である。

## 参考文献

[1] Bowen Alpern, Dick Attanasio, John J. Barton, M. G. Burke, P. Cheng, J.-D. Choi, Anthony Cocchi, Stephen J. Fink, David Grove, Michael Hind, Susan Flynn Hummel, D. Lieber, V. Litvinov, Mark Mergen, Ton Ngo, J. R. Russell,

| GC | 総実行時間 (秒) | | | |
|---|---|---|---|---|
| | jess | jack | mtrt | compress |
| copyGen | 49.425 | 44.145 | 75.480 | 59.214 |
| stackGen | 58.898 | 49.983 | 91.355 | 61.177 |
| stackGenHalf | 50.919 | 44.091 | 75.654 | 59.928 |
| stackGenNoTrace | 50.963 | 44.702 | | |
| trace | 56.691 | 43.942 | | |

表 3: 実験結果

Vivek Sarkar, Manuel J. Serrano, Janice Shepherd, S. Smith, V. C. Sreedhar, H. Srinivasan, and J. Whaley. The Jalapeño virtual machine. *IBM System Journal*, 39(1), February 2000.

[2] Martin Hirzel, Johannes Henkel, Amer Diwan, and Michael Hind. Understanding the connectivity of heap objects. In David Detlefs, editor, *ISMM'02 Proceedings of the Third International Symposium on Memory Management*, ACM SIGPLAN Notices, pages 36–49, Berlin, June 2002. ACM Press.

[3] Henry Lieberman and Carl E. Hewitt. A real-time garbage collector based on the lifetimes of objects. *Communications of the ACM*, 26(6):419–429, 1983. Also report TM–184, Laboratory for Computer Science, MIT, Cambridge, MA, July 1980 and AI Lab Memo 569, 1981.

[4] Standard Performance Evaluation Corporation (SPEC). Specjvm98 benchmarks. http://www.spechbench.org/osg/jvm98.

[5] Darko Stefanović, Matthew Hertz, Steve M. Blackburn, K. S. McKinley, and J. Eliot B. Moss. Older-first garbage colleciton in practice: Evaluation in a java virtual machine. Memory System Performance, Berlin, Germany, June 2002.

[6] Darko Stefanović, Kathryn S. McKinley, and J. Eliot B. Moss. Age-based garbage collection. In *OOPSLA'99 ACM Conference on Object-Oriented Systems, Languages and Applications*, volume 34(10) of *ACM SIGPLAN Notices*, pages 370–381, Denver, CO, October 1999. ACM Press.

[7] Paul R. Wilson. Uniprocessor garbage collection techniques. Technical report, University of Texas, January 1994. Expanded version of the IWMM92 paper.

[8] Paul R. Wilson and Thomas G. Moher. Design of the opportunistic garbage collector. In *OOPSLA'89 ACM Conference on Object-Oriented Systems, Languages and Applications*, volume 24(10) of *ACM SIGPLAN Notices*, pages 23–35, New Orleans, LA, October 1989. ACM Press.