スタックスキャンを中断させるリターンバリアごみ集め

鵜川 始陽[†1] 湯淺 太一[†2]
京都大学大学院情報学研究科[†3]

1. はじめに

インクリメンタルごみ集めアルゴリズムのひとつであるスナップショットごみ集め[1]は，リターンバリア[2,3]を使って改良することで，実時間アプリケーションのごみ集めとしても利用できる[4].

スナップショットごみ集めでは，ごみ集め開始時に，大域変数やスタックなど，システムがいつでも参照できるデータ領域（ルート集合）を一括してスキャンし，それらから直接ポインタで指されているすべてのオブジェクトにマークを付け，それらへのポインタをごみ集め用のスタック（マークスタック）に積む．マークスタックに積まれたオブジェクトを起点にポインタをたどることで，ごみ集め開始時に生きているオブジェクトをすべて見つけることができる．ただし，ルート集合のスキャンは一括して行われるため，その間アプリケーション（ミューテータ）の実行が停止してしまい実時間性が損なわれる可能性がある．

改良したスナップショットごみ集めでは，ミューテータはごみ集め開始時に，大域変数などのスタック以外のルート集合および，現在実行中の関数のフレーム（カレントフレーム）とその下位（呼び出し元）の関数フレームのみ一括してスキャンする．残りのスタック領域は下位に向かってミューテータと並行してスキャンする．ミューテータは，カレントフレーム以外のスタックの領域に対して読み書きを行わないため，ミューテータが実行する前にカレントフレームがスキャンされている限り，マーク漏れは起こらない．

関数からリターンすると呼び出し元の関数のフレームがカレントフレームとなり，読み書きされるようになる．そのため改良した方式ではスキャン済の関数から未スキャンの関数へのリターンにバリア（リターンバリア）を設定する．バリアを越えてリターンする際は，呼び出し元関数のフレームをスキャンし，リターンバリアを再設定してからリターンする．

多くの場合，関数からのリターンよりごみ集めによるスタックのスキャンが先に進むため，リターンバリアが働くことは少ない．しかし，アプリケーションプログラムやごみ集めが開始されるタイミングによってはリターンがスタックのスキャンに追いつくことがある．さらに，リターンが連続する場合，ひとたびリターンがスタックのスキャンに追いつくと，以降のリターンでは連続してバリアが働く．実際 Java VM にリターンバリアを実装し，深い再帰呼び出しを行うプログラムを実行した際にこのようなことが起きた．短時間で大量の関数フレームがスキャンされることで，ミューテータの実行が極端に遅くなり，実時間性が損なわれる可能性がある．

そこで，本稿ではリターンバリアが働いた際にリターン先をスキャンするのではなく，スタックスキャンを中断し，次にごみ集めが起動したときに最初からスキャンし直す方式を提案する．

2. 提案手法

リターンがスタックスキャンに追いつきリターンバリアが働くと，スタックスキャンを中断し，スタック以外の領域も含めて最初からスキャンし直す．一度リターンバリアが働くとごみ集めが中断されるため，それ以降のリターンではバリアは働かず，連続したリターンで実行が極端に遅くなることはない．

図1にその様子を示す．図1(a)は最後のスキャン済み関数フレームまでリターンした様子を示している．さらにリターンすると，リターンバリアが起動してごみ集めを中断する．リターンした後の図1(b)では，ごみ集めが中断しているため，リターンバリアは設定されていない．したがって，さらにリターンが続いてもバリアは働かず，直ちに図1(c)の状態になる．この後もリターンが続いた後の様子を図1(d)に示す．ごみ集めが起動すると，ごみ集めを最初からやり直す．これにより，スタック以外のルート集

Aborting Stack Scanning in Garbage Collection with Return Barrier.
†1 Tomoharu Ugawa †2 Taiichi Yuasa
†3 Graduate School of Informatics, Kyoto University

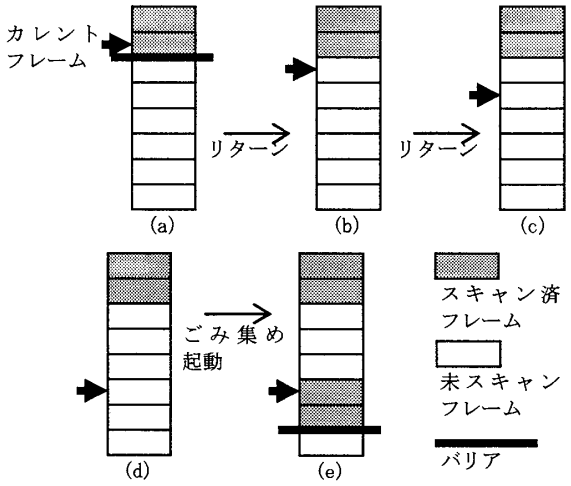

図1　ごみ集めの中断
Fig.1　Aborting Garbage Collection.

合およびカレントフレームとその下位のフレームがスキャンされバリアが設定される（図1(e)）.

3.　ごみ集めの中断処理

厳密にごみ集めを中断するためには，それまでにごみ集め処理が与えた副作用を巻き戻さなければならないが，実際には厳密に中断させる必要はない．ルート集合をスキャンしている最中であれば，ごみ集め処理により与えられた副作用は，スキャンの終わった領域から指されていたオブジェクトのマークと，それらがマークスタックに積まれているだけである．そのままにしておいても，次のごみ集めで誤って生き残るオブジェクトが増える可能性があるだけで，誤って生きているオブジェクトを回収することはない．さらに，誤って生き残ったオブジェクトもその次のごみ集めで回収される．

4.　ごみ集めが完了することの保証

提案手法では，ごみ集めを中断して最初からやり直す．そのため，何度もやり直してごみ集めが完了しない可能性がないとは言い切れない．ごみ集めを中断した後，次にごみ集めが起動される前にスタックが伸び，さらにごみ集め起動後に，伸びた部分のスタックをスキャンしている最中に再び中断される場合である．

図2(a)はスタックスキャンの途中の状態を示している．リターンが連続するとリターンバリアによりごみ集めが中断され図2(b)のようになる．その後，関数呼び出しにより図2(c)のように未スキャンの関数フレームが積まれる．そこでごみ集めが起動すると，図2(d)のように図2(a)と同じ状態に戻り，これが繰り返されるといつまでたってもごみ集めが完了しない．

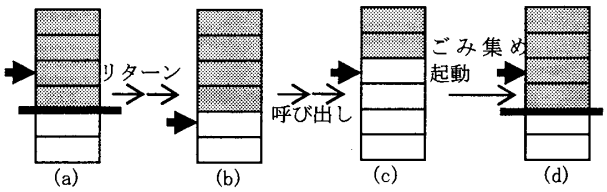

図2　中断後の関数呼び出し
Fig.2　Calls after Abort.

そこで，ごみ集めを中断した後，スタックが一定以上伸びようとしたときにごみ集めを起動し，それより多くの関数フレームをスキャンする．例えば，スタックが固定長のチャンクに分かれていれば[5]，ごみ集め中断後の関数呼び出しでチャンクがオーバフローすると，新しいチャンクを用意する前にごみ集めを起動し，オーバフローしたチャンクをスキャンする．こうすることで，ごみ集めが中断したとしても，そのときスキャン済みの関数フレームよりも下位のフレームまで次にごみ集めが起動したときにスキャンされ，いつかはスキャンが完了する．

5.　まとめ

本稿では，スタックスキャンよりもミューテータによるリターンが速く進み，未スキャンの関数にリターンしようとしたとき，スタックスキャンを中断して，次にごみ集めが起動したときに最初からやり直す手法を提案した．今後の課題として，実際の処理系に本手法を実装して効果を確認することを計画している．

参考文献

[1]T. Yuasa: Real-time garbage collection on general-purpose machines, The Journal of Systems and Software, Vol.11, No.3, pp.181-198, 1990.

[2]湯淺太一，中川雄一郎，小宮常康，八杉昌宏：リターン・バリア，情報処理学会論文誌,41巻SIG9 (PRO8) 号，pp.1-13, 2000.

[3]T. Yuasa, Y. Nakagawa, T. Komiya, M.Yasugi: Return Barrier, Proceedings International Lisp Conference, 2001.

[4]H. Saiki, Y. Konaka, T. Komiya, M. Yasugi, T. Yuasa: Real-time GC in JeRTy™ VM Using the Return-Barrier Method, Proceedings ISORC 2005, pp.140-148, 2005.

[5]N. Shaylor, D. M. Simon, W. R. Bush: A Java Virtual Machine Architecture for Very Small Devices, ACM SIGPLAN Notices, Vol.38, No.7, pp.34-41, 2003.