A_015

メソッドキャッシュへのメソッド呼び出し回数を用いたエージングアルゴリズムの導入
Introduction of call-count aging algorithm to method-cache

長田 忍† Shinobu Nagata
楢崎 修二‡ Shuji Narazaki

1. はじめに

オブジェクト指向プログラミング言語（OOPL: Object-Oriented Programming Language）は定義時に変数及びメソッドの型付けを行う静的型付け OOPL と実行時に型付けを行う動的型付け OOPL とに分けられ，適用できるメッセージ送信の高速化手法が異なる．静的型付け OOPL では型情報を用いた静的解析が可能となり，これに基づいたコンパイル時最適化や JIT（Just In Time）コンパイルなどを施すことが容易である．一方，動的型付け OOPL では実行時まで型情報が得られず静的解析が困難になるため，実行時における動的な最適化としてメソッドキャッシュ[1, 3] を利用するのが一般的である．

メソッドキャッシュは引数の型の並びとメッセージ名（セレクタ）とを組としたシグネチャを鍵としてメソッドを記憶するハッシュ表で構成され，その有効性は利用コストとヒット率とによって示すことができる．ここで利用コストにはハッシュ関数の計算コストに加えて，ミスした場合のメソッドの登録，置換の管理コストを含むものとする．管理コストを減らすためには，メソッドを記憶する際に必要なエントリと呼ばれる記憶領域の置換アルゴリズムが重要となり，効率的なエントリ置換を実現できれば動的型付け OOPL の処理系の性能向上が見込める．

本論文では効率的なエントリ置換アルゴリズムとして，メソッド呼び出し回数を用いたメソッドキャッシュ制御法を提案する．また動的型付け OOPL である Squeak の仮想マシン（VM: Virtual Machine）へ呼び出しカウンタ一体型メソッドキャッシュを実装した効果について示す．

2. メソッドキャッシュのエントリ置換

理想的な置換アルゴリズムとして，NFU（Not Frequently Used）アルゴリズムから派生したエージング（aging）アルゴリズム [4] が挙げられる．NFU アルゴリズムは最も使用頻度の低いものを置換対象とするアルゴリズムであるが，参照回数のみを基準としているため最近使用していないにも関らず置換対象にならないという問題が生じる．また更新時間を置換の基準として利用する LRU（Least Recently Used）アルゴリズムでは，更新時間のみを基準としているため頻繁に参照するものとほとんど参照しないものとを区別できない．NFU アルゴリズム及び LRU アルゴリズムの問題を解決するため，参照回数を時間が経過する毎に減らすのがエージングアルゴリズムである．これによりエージングアルゴリズムでは NFU アルゴリズムと LRU アルゴリズム両方の特徴を利用することができる．これらの置換アルゴリズムは，メモリ管理としてページング方式を使用するオペレーティングシステムにおいてページを置換する際に利用されている．

本論文では効率的なエントリ置換アルゴリズムを実現するため，メソッド呼び出し回数を基準とするエージングアルゴリズムを提案する．提案する手法はメソッドキャッシュにメソッドの呼び出し回数を記憶しておくための呼び出しカウンタを一体化し，呼び出しカウンタの値が最も小さいものを置換対象とすることでエントリ管理を行うものである．本手法は，複数のエントリ候補から置換対象を選択するタイプのメソッドキャッシュであれば言語を問わず適用可能であり，ページング方式を使用するオペレーティングシステムにおいて研究されてきた効率的な置換アルゴリズムをメソッドキャッシュのエントリ置換に適用したものである．

3. 実装

今回は Squeak VM へ呼び出しカウンタ一体型メソッドキャッシュを実装する．Squeak はメッセージを受信したレシーバのクラスとセレクタとを用いてメソッドの探索を行う言語であるため，メソッドキャッシュはセレクタとレシーバクラスのみを鍵として検索を行う．検索の際はセレクタとレシーバクラスとを使って 3 パターンのハッシュ値を求め，3 つのエントリを検索対象とする．この 3 つのエントリを，参照する順に第 1probe，第 2probe，第 3probe と呼ぶ．第 3probe まで参照してヒットしなかった場合は，通常のメソッド探索を行って第 1probe に上書き登録し，第 2probe 及び第 3probe の情報を消去する．この際，Squeak VM 中のメソッドキャッシュの構造が 1 次元配列であることから，情報を消去した第 2probe 及び第 3probe が他の鍵で計算したインデックスのエントリである可能性がある．従って提案手法によって無駄な情報の消去を減らし，かつ最適なエントリと置換することで管理コストを下げることができれば処理速度の向上が見込める．

実際にメソッド呼び出し回数を基準として用いるエージングアルゴリズムを導入したメソッドキャッシュにおけるメソッド検索アルゴリズムを以下に示す．

1. セレクタとレシーバクラスとを使って 3 つの probe のハッシュ値を求め，
2. 第 1probe から第 3probe までの検索でヒットしたらエントリの呼び出しカウンタの値を増やしてメソッドの処理へ，ミスしたら以降の処理へ進み，
3. 第 1probe から第 3probe までを参照した際に求めた最も呼び出しカウンタの値が小さいエントリを置換対象に決定，
4. 通常のメソッド探索を行って置換対象のエントリに登録する．

†長崎大学大学院 生産科学研究科
‡長崎大学 工学部

表 1: 実験環境

CPU	Intel Xeon 3.2[GHz]×2
メモリ	1024[MB]
OS	Linux 2.6.16-1-686-smp

表 2: エントリ置換アルゴリズムによるヒット率の変化

	ヒット率 [%]	
	キャッシュ全体	第 1probe
通常のエントリ置換	98.1	51.9
提案手法	94.3	87.2

呼び出しカウンタを直接エントリに埋め込むとエントリ上の情報を消去する際やエントリの置換が起きた場合に記憶していた呼び出し回数が消えてしまい，同じメソッドを登録する際に履歴として使用できなくなってしまう．これを防ぐためメソッドとその呼び出し回数とを組とした履歴情報を管理する表を新しく用意した．この表の大きさを変えることで履歴情報を残すメソッドの数を調整することが可能となり，呼び出しカウンタの値を減らす際のオーバーヘッドを削減することができる．詳細なアルゴリズムは発表において説明する．

4. 評価

提案手法の導入による効果を評価するために実験を行った．実験環境を表 1 に示す．実験を行った Squeak イメージのバージョンは 3.7-5989-full で，変更を加える前の Squeak VM のバージョンは 3.7-7 である．

提案手法によるエントリ置換について評価するため，通常のエントリ置換アルゴリズムを使用した VM と提案手法を使用した VM とでメソッドキャッシュのヒット率を計測した．結果を表 2 に示す．表 2 を見ると提案手法を使用した場合，第 1 から第 3probe まで全体の検索においてはヒット率が約 4%低下してしまった．一方，第 1probe のみ注目した場合のヒット率は約 35%上がったことが分かる．

実行速度の評価のために，複数のベンチマークプログラムを通常のエントリ置換アルゴリズムを使用した VM と提案手法を使用した VM とで実行した．通常のエントリ置換アルゴリズムを使用する VM と比較した，提案手法を使用する VM の性能比率を表 3 へ示す．ベンチマークプログラムの内容はそれぞれ，

LoadInstVari インスタンス変数の読み込み，
PopStoreTemp 一時変数への代入，
WhileLoop 単純な繰り返し，
Size 文字列の大きさ判定，
Class オブジェクトのクラス取得，
Perform `perform:`メソッドによる機械的なメソッド実行，
Compiler メソッドのコンパイル，

を繰り返し行うプログラムである．表 3 より提案手法を使用した VM の性能を向上できたことが分かる．これは表 2 が示す通り，第 1probe のヒット率の向上によるものであると考えられ，第 2probe 及び第 3probe の消去による他の鍵での検索ミスが引き起こしたメソッドの登録や置換による管理コストの増加を抑制できたことを表している．

表 3: エントリ置換アルゴリズム変更前と変更後との性能比率

ベンチマークプログラム	性能比率 [%]
LoadInstVari	115.0
PopStoreTemp	157.9
WhileLoop	110.0
Size	115.4
Class	133.3
Perform	122.2
Compiler	128.2

5. まとめ

本論文では，メソッドキャッシュにおける効率的なエントリ置換アルゴリズムとしてメソッドの呼び出し回数を基準としたエージングアルゴリズムを Squeak VM に実装した．また実装した効果についてヒット率とベンチマーク結果を用いて評価した．

通常のエントリ置換アルゴリズムを使用した VM と提案手法を使用した VM とでメソッドキャッシュのヒット率を計測したところ，提案手法では全体のヒット率は下がったものの第 1probe でのヒット率が約 35%増加した．またベンチマークプログラムを実行したところ，提案手法の実装によって性能が向上した．以上から提案手法の実装によって複数エントリの参照，登録，及び置換による管理コストを削減できたものと思われる．また今回行った実験から，複数のエントリ候補から置換対象を選択するタイプのメソッドキャッシュを持つ処理系では管理コストの評価を行うため，メソッドキャッシュ全体のヒット率だけでなくエントリレベルでのヒット率に関しても計測する必要があるということが分かった．エントリレベルでの評価を行って改善の余地があるならば，他の処理系においても管理コストを削減できると考えられる．

今回はメソッドの呼び出し回数をメソッドキャッシュの高速化を目的として使用したが，レシーバクラス予測 [2] のようにプロファイル情報としても利用できると考えられる．今後はメソッドキャッシュのみに限らず処理系全体の性能向上を目指す．

参考文献

[1] L. Peter Deutsch and Allan M. Schiffman. Efficient Implementation of the Smalltalk-80 System. In *POPL '84: Proceeding of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pp. 297–302, New York, NY, USA, 1984. ACM Press.

[2] David Grove, Jeffrey Dean, Charles Garrett, and Craig Chambers. Profile-Guided Receiver Class Prediction. *ACM SIGPLAN Notices*, Vol. 30, No. 10, pp. 108–123, Oct. 1995.

[3] Glen Krasner, editor. *Smalltalk-80: Bits of History, Words of Advice.* Addison-Wesley, Aug. 1983.

[4] Andrew S. Tanenbaum. *Modern Operating Systems.* Prentice Hall, 2nd edition, Feb. 2001.