1K-3

Ruby 用動的コンパイラにおける脱仮想化の実装

齋藤 学† 千葉 雄司‡ 土居 範久†

† 中央大学大学院 理工学研究科 情報工学専攻 ‡ 中央大学 研究開発機構

1 はじめに

Ruby は，プログラムのしやすさからユーザ数が増加している．そして，それに伴い，利用分野も広がりつつある．しかし，既存の Ruby 処理系には他のスクリプト言語処理系に比べ処理速度が遅いという欠点があり，高速な処理を必要とする分野への適用は難しい．

Ruby で記述したプログラムの実行を遅くする原因の 1 つにメソッド呼出しがある．Ruby は，すべてをオブジェクトとして表現するという言語モデルをとっており，その結果メソッド呼出しが多発する．そこで，我々はメソッド呼び出しの高速化を目的として，脱仮想化を実装し，その効果を評価した．ここで脱仮想化とは呼び出し先が唯一に決まる動的なメソッド呼び出しを静的なメソッド呼び出しに変換することで，冗長なメソッドの検索を省略する最適化技法の 1 つである [1].

2 Ruby 用動的コンパイラの概要

脱仮想化の実装に先立ち，我々は Ruby 向け動的コンパイラを実装した．実装は既存の Ruby 処理系 [2] を機能拡張する形で行った (図 1)．実装のベースとした Ruby 処理系はスクリプトの実行にあたり，まずスクリプトを木構造の中間表現に変換し，つぎにインタプリタを使ってこの中間表現を再帰的に評価するが，我々の動的コンパイラは，この中間表現をコンパイルして x86 向けの機械コードに変換する．そして，変換後の機械コードをリンカにより，Ruby の共有ライブラリとリンクし，その後実行する．動的コンパイル済みコードの実行に当たって，中間表現を再帰的にたどる処理を必要としないため，高速に動作する．コンパイルを開始するタイミングは，スコープの先頭に達するときもしくは，コンパイルしていないメソッドを呼ぶときに開始するようにした．なお，今のところ実装した動的コンパイラにおけるの最適化としては，脱仮想化のみを実装してある．

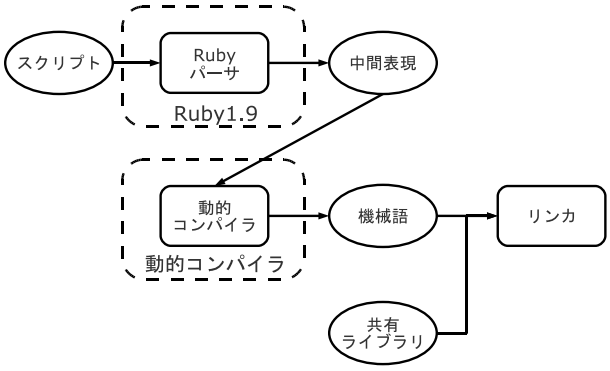

図 1: 動的コンパイラの全体像

3 脱仮想化

オブジェクト指向言語では，メソッド呼び出しの高速化が重要である [3]．なぜなら，オブジェクト指向のプログラミングでは小さなメソッドを頻繁に呼び出すことが多いので，メソッド呼出しのオーバヘッドが大きくなりやすいからである．また，メソッド呼出しの際に，呼出先のメソッドを検索する処理も大きなオーバヘッドの発生要因となっている．なぜなら，メソッド呼び出しは，呼出先がレシーバのクラスに依存することから，単純な分岐として実現できず，呼出しの呼出先のメソッドを検索する必要が生じる．

たとえば，図 2 の例ではクラス C1 にメソッド m を定義し，次にクラス C2 に C1 と同名のメソッド m を定義する．そして，if 文の本体内に変数 c にクラスをインスタンス化した値を代入する処理を記述すると最終行の c.m() において，クラス C1 のメソッド m を呼ぶのか，クラス C2 のメソッド m を呼ぶのかをコンパイル時に判断することはできない．

しかし，図 3 のようにメソッド m の定義が全クラスに対して，一つのときには呼び出すメソッド m の呼び出し先を，唯一に決めることができる．よって，これ以降のクラス C に対するメソッド m の検索処理を省略することができる．

そこで，我々は図 3 のように，呼び出し先が唯一に決まるメソッド呼び出しについて呼び出し先のメソッドの検索を省く最適化 (脱最適化) を実装した．

この方法の利点は呼び出し先のメソッドを，インライン展開可能になることである．さらに重要なのは，実

Implementation of devirtualization in a dynamic compiler for Ruby
†Manabu SAITO ‡Yuji CHIBA †Norihisa DOI
†Information and System Engineering Course, Graduate School of Science and Engineering, Chuo University
‡Research and Development Initiative, Chuo University

```
class C1
  def m end
end

class C2
  def m end
end

if cond
  c = C1.new
else
  c = C2.new
end

c.m # call C1.m or C2.m ?
```

図 2: 呼び出すメソッドが唯一に決まらない例

```
class C
  def m end
end

c.m # call C
```

図 3: 呼び出すメソッドが唯一に決まる例

際にインライン展開することにより，呼び出しメソッドが唯一であるかをチェックする必要があるが，共通部分式の除去や定数たたみ込みなどの様々な最適化が可能になることである．

4 評価と考察

実装した脱最適化が実行速度に及ぼす影響を評価した．評価環境を表 1 に示す．

表 1: 評価環境

CPU	Pentium4 1.8GHz
Memory	512MByte
OS	Ubunto Linux (kernel-2.6.15)

評価には 2 つのマイクロベンチマークを用いた．これらのマイクロベンチマークは共に，メソッド呼び出しを 300000 回実行するが，一方ではメソッドの呼出先を唯一に定めることができ，他方では唯一に定めることができない．それぞれの実行にかかった時間を表 2 に示す．また，表中の実行時間にはコンパイルする時間も含まれている．

表 2 の結果より，呼び出し先が唯一に決まる呼び出しの際には，メソッド呼び出しの際の検索が無くなり，約 2 倍程度速くなることがわかった．また，YARV[4] に付属している，Ruby で実装したフィボナッチ数など

表 2: マイクロベンチマークの結果 (単位は秒)

	脱仮想化なし	脱仮想化あり
polymorphism method	3.96	1.71
no polymorphism method	4.07	4.11

の簡単ベンチマークプログラムの評価も行った (表 3)．また，表中のあり・なしは脱仮想化の有効・無効を示している．評価結果から相乗平均で 1.7 倍程度，高速化できていることがわかる．

表 3: ベンチマークの結果 (単位は秒)

プログラム名	なし (A)	あり (B)	比率 (A/B)
fib	7.83	5.83	1.34
strconcat	4.76	4.11	1.2
times	148.45	138.25	1.08
whileloop	3.38	1.99	1.70
ackermann	34.74	12.03	2.89
matrix	2.02	3.01	1.50
object	29.59	21.62	1.37
const	60.64	49.72	1.22
swap	19.19	2.27	8.45
arry	10.11	6.32	1.66
regexp	8.48	4.56	1.86

5 まとめ

本稿では我々が実装した動的コンパイラに対して，脱仮想化を実装し，その結果を示した．呼び出し先メソッドが唯一に決まるメソッドを呼ぶ際には，約 2 倍程度速くなることがわかった．

参考文献

[1] Kazuaki Ishizaki, Motohiro Kawahito, Toshiaki Yasue, Hideaki Komatsu, and Toshio Nakatani. A Study of Devirtualization Techniques for a Java Just-In-Time Compiler. *IBM Research Report*, Vol. RT0352, , 2000.

[2] オブジェクト指向スクリプト言語 ruby. http://www.ruby-lang.org/ja/.

[3] 小野寺民也．オブジェクト指向言語におけるメッセージ送信の高速化技法．情報処理, Vol. 38, pp. 301–310, 4 1997.

[4] K Sasada. Yarv: Yet another ruby vm. http://www.atdot.net/yarv/.