4J-02

Ruby 用トレーシング実行時コンパイラの実装

田中 達也　　松原 俊一　　Martin J. Dürst

青山学院大学理工学部情報テクノロジー学科

1 はじめに

オブジェクト指向スクリプト言語 Ruby は非常に直感的な文法で記述でき, 様々な環境で動作することから幅広く利用されている. その結果, 処理速度も求められるようになってきている. しかし, Ruby は処理速度を重視して設計された言語ではないため, 適用範囲を狭めている原因になっている.

そこで処理速度の向上を目指して, 様々な処理系が開発されてきた. Java で実装された JRuby, RPython[1] で記述された Topaz, Ruby 自身で実装された Rubinius が代表的である. 例に挙げた高速化を実現した処理系の全てに, ネイティブコードを動的に生成する Just-in-Time (JIT) コンパイラが実装されている. しかし C 言語で実装された CRuby は JIT コンパイラを有していない. CRuby はまつもとゆひきろが開発した Ruby の参考実装であり, Mac OSX などで標準インストールされている. そこで我々は CRuby の高速化を目指して CRuby 用の JIT コンパイラを開発した. なおコード生成にはコンパイラ基盤 LLVM[2] を用いている.

2 プログラミング言語 Ruby

CRuby には 2007 年にリリースされたバージョン 1.9 から YARV [1] と呼ばれる仮想マシンが導入された. CRuby はスクリプトを読み込むと抽象構文木 (AST: Abstract Syntax Tree) に変換した後, 独自のバイトコードに変換している. 図 1 に Ruby の簡単なスクリプトをバイトコードに変換したものを示す. バイトコードはトップレベルやメソッドなどの単位で分割して生成される. そして生成したバイトコードを YARV に解釈させることでスクリプトを実行している [2].

3 JIT コンパイラの実装

開発した JIT コンパイラは, 複数回実行されている区間を検知し, 実行中にその部分をネイティブコードにコンパイルする. その後コンパイルされた区間を実行する時はバイトコードではなく, ネイティブコードの方を実行する.

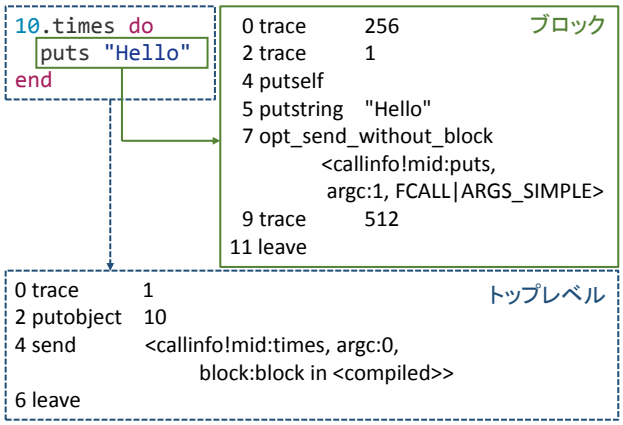

図 1: Ruby のバイトコード

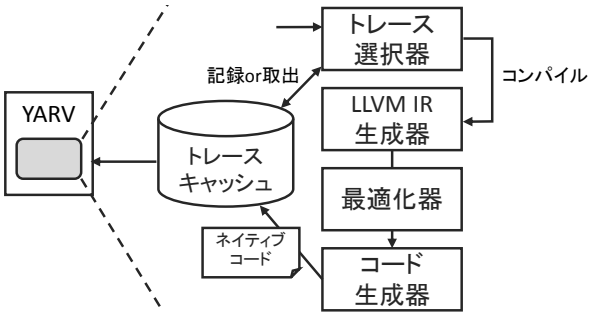

図 2: システム構成図

3.1 JIT コンパイラの構成

システムの構成図を図 2 に示す. 以後は記録した区間をトレースと表記する.

開発した JIT コンパイラは, CRuby がバイトコードの命令を実行する前に, トレースの記録またはトレースのコンパイルと生成されたネイティブコードの実行を行う. ネイティブコードを実行した場合はそのトレースの終了位置まで CRuby の仮想マシンのプログラムカウンタを進める. トレースのコンパイルは繰り返し実行されている区間のみに対して行う. 図 1 の場合はブロック部分がコンパイル対象となる.

トレースのコンパイルは LLVM を用いて行われる. まず Ruby のバイトコードを LLVM の API を利用して LLVM 独自の中間表現 (LLVM IR) に変換する. その後, LLVM の最適化モジュール (Pass) を利用して, ピープホール最適化, 共通部分式除去, 定数畳み込みなどの最適化を行う. 次に LLVM の `ExecutionEngine` クラスを介して LLVM IR をターゲットのアーキテクチャ用のネイティブコードに変換する.

Implementation of a Tracing Just-in-Time Compiler for Ruby

Tatsuya Tanaka, Shunichi Matsubara and Martin J. Dürst
Department of Integrated Information Technology, College of Science and Engineering, Aoyama Gakuin University
5–10–1 Fuchinobe, Chuo-ku, Sagamihara, Kanagawa 252-5258, Japan
duerst@sw.it.aoyama.ac.jp

[1] https://pypy.readthedocs.org/en/release-2.4.x/translation.html
[2] http://llvm.org/

3.2 トレースの選択

トレースの開始位置の決定手法には Next Executing Tail (NET) [3] を採用した. 具体的には (1) 後方分岐の命令, (2) バイトコード遷移, の後の命令でトレースを開始する. (1) は主に繰り返している区画の先頭の命令である. (2) は主にメソッド呼び出し後の命令やメソッドから戻ってきてすぐの命令である. トレース選択の疑似コードをソースコード 1 に示す. トレースのコンパイルには時間がかかるため, 指定した回数以上繰り返し実行している箇所のみに対して行っている.

ソースコード 1: トレース選択の疑似コード

```
1  void select_trace(pc) {
2      if (pc の位置の命令がトレース済み) {
3          if (counter > THRESHOLD) {
4              if (トレースが未コンパイル)
5                  トレースのコンパイル
6              コンパイル済みトレースの実行
7          }
8          else
9              counter++;
10     }
11     else
12         トレースの開始
13 }
```

3.3 トレースの記録

CRuby の仮想マシンは VM 生成系 [1] というツールを用いて, ある程度自動的に生成されている. そのツールを書き換えてバイトコードの各命令に命令の記録を行う処理を追加した. これにより各命令を通常通り実行するだけで命令の記録が自動的に行われる. またトレースの開始位置を高速に検出するために `jump` や `branchif`, `branchunless` などの後方分岐となりうる命令に検出処理を追加した. 一つのトレースは, ループの検知, C 言語で記述されたメソッドの実行が起こるまで続ける.

4 実行時間の評価

開発した JIT コンパイラの評価のために CRuby に付随しているマイクロベンチマークで処理速度を比較した. 比較対象は本研究の Ruby と CRuby である. 多く存在するマイクロベンチマークの中から現在, ネイティブコード生成に対応しているものを使用した. この評価では 2 回同じ区間を実行している場合にコンパイルする設定にした. 評価は 2.6 GHz Intel Core i5 CPU の Mac OSX で動作する VirtualBox 上の Ubuntu 15.10 で行った. 使用した Ruby のバージョンは Ruby 2.3.0 dev (trunk 50920), LLVM のバージョンは 3.7.0 である. 比較した結果は 図 3 に示す. 図中の橙色は CRuby よりも高速化されたことを示している.

結果としては一部のベンチマークでは 7 倍ほどの高速化に成功した. 高速化されたものは繰り返し実行される部分で多くの計算を行っている. 計算はそのままネイティブコードに変換することが可能なため, 高速化したと考えられる. 幾つかのベンチマークでは CRuby と変わらないか, または少し遅くなった. それらのベンチマークでは繰り返し実行される部分が空, または C 言語で実装されたメソッドの呼び出しが複数あった. 現時点ではコンパイル対象のトレースのサイズが小さい場合でもコンパイルされる. そのためネイティブコードとバイトコードの遷移にかかる時間が, ネイティブコード化によって短縮された時間よりも大きくなっていることが遅くなった原因の一つである.

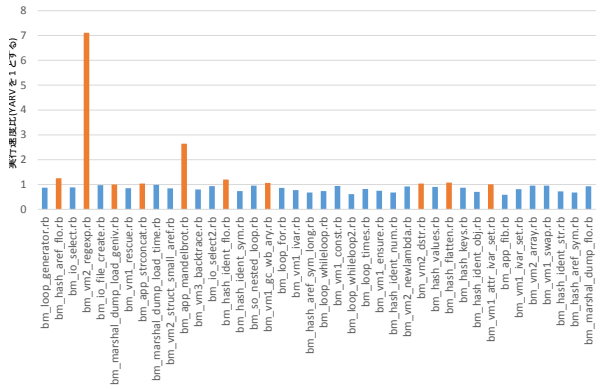

図 3: マイクロベンチマークの評価結果

5 まとめと今後の課題

本研究では既存の Ruby の処理系にトレーシング JIT コンパイラを追加した. Ruby のバイトコードをネイティブコードに変換することで大幅な高速化に成功した. JIT コンパイラのオーバーヘッドが大きいため, 単純な処理やトレース区間が小さくなる場合に速度が低下することもあった. LLVM を用いることで Ruby の様々なアーキテクチャで動作するというメリットを損なわずに JIT コンパイラを追加できた.

今後の課題としてはトレースの区間が細かく区切れているため, ネイティブコードとバイトコードの遷移が多く発生している. これはトレースの区間を広げることで減らすことが可能なので, トレース選択のアルゴリズムの見直しも必要である.

謝辞 本研究を進めるにあたり, 笹田耕一氏にご教示頂きました. ここに感謝の意を表します.

参考文献

[1] 笹田耕一, 松本行弘, 前田敦司, 並木美太郎. Ruby 用仮想マシン YARV の実装と評価. 情報処理学会論文誌プログラミング (PRO), Vol. 47, No. 2, pp. 57–73, 2006 年 2 月.

[2] Pat Shaughnessy 著, 島田 浩二訳, 角谷 信太郎訳. Ruby のしくみ – Ruby Under a Microscope. オーム社, 2014 年 11 月.

[3] Evelyn Duesterwald and Vasanth Bala. Software Profiling for Hot Path Prediction: Less is More. *SIGOPS Oper. Syst. Rev.*, Vol. 34, No. 5, pp. 202–211, 2000 年 11 月.