論 文 内 容 の 要 旨

論文題目　高速な Ruby 用仮想マシンの開発

氏　名　笹 田 耕 一

近年，多様化するソフトウェア需要に応えるため，開発効率の良いプログラミング言語が求められている．その中で，オブジェクト指向スクリプト言語 Ruby が実用的で開発効率の良い言語として注目されている．しかし，従来の Ruby 処理系は抽象構文木を辿る単純な実装のため実行が遅いという問題点があった．そのため高い性能が必要となるソフトウェアの開発には用いることができなかった．

本研究は Ruby を高速に実行する処理系の実現手法を研究し，プログラミング言語 Ruby の可能性を広げることを目的とする．そして，世界中で利用されるような実用的なソフトウェアを開発することを目標とする．

高速な Ruby 処理系を実現するための課題として，まず従来の Ruby 処理系の抽象構文木を辿る単純な実行方式の改善がある．従来の方式は，実現は容易であるが性能に問題があることが知られている．動的型システムおよび強力なリフレクション機能に代表される Ruby の動的特性は，Ruby プログラミングに柔軟性を与えるが，静的解析による最適化が不可能となるため，Ruby 処理系に適した実行時最適化技術の開拓が必要となる．さらなる高速化を目指すためには，近年とくに普及してきたマルチコア，マルチプロセッサを利用したメモリ共有型並列計算機による並列実行の実現が重要である．しかし，過去のプログラム資産を生かすためにはスレッドセーフでない機能の扱いが問題となる．

上記の課題に加え，実用的な Ruby 処理系を実現するため，互換性，保守性，移植性が開発全体での課題となった．実際的なプログラミングを行うためには過去の豊富なプログラム資産を活かさなければならない．そのため従来の処理系との互換性の維持が必須である．と

くに Ruby 処理系を拡張するための C 拡張ライブラリと，C 拡張ライブラリを記述するために必要となる Ruby C API の実現が課題となった．また，言語処理系は複雑なソフトウェアであるため，保守性の向上が重要である．そして，移植性の維持は多くの計算機環境で動作する実用的な処理系とするために重要である．

高速な Ruby 処理系を実現に関するこれらの課題を解決するために，本研究では YARV: Yet Another RubyVM という仮想マシン方式による Ruby 用言語処理系を新たに開発した．まず，Ruby プログラムを正しく表現できる YARV 命令セットを設計し，Ruby プログラムを YARV 命令セットで構成された YARV 命令列へ変換するコンパイラを作成した．YARV 命令列を実行するための仮想マシン YARV は仮想マシンアーキテクチャとして一般的なスタックマシンとした. 例外処理は実行時にオーバヘッドが不要となる表引き法を採用したが，Ruby C API の互換性確保のため C 言語での大域ジャンプを用いる方式と併用して実現した．

さらに，開発した仮想マシン YARV に実行時最適化技術を Ruby プログラムの意味を変更しないよう適用した．とくに，Ruby の動的特性を堅持しながら高速化を行うため，実行時最適化を中心とした Ruby 処理系に適用可能なさまざまな高速化手法を検討し, 特化命令やインラインキャッシュを利用した高速化を行った．また，既知の仮想マシン高速化手法である命令の融合操作，静的スタックキャッシングを YARV 命令列に適用した．

仮想マシンの逐次実行の性能評価を行うため，典型的な Ruby プログラムを実行して，それぞれの最適化がどの程度性能向上に寄与しているかを調べた．総合的には，仮想マシンを利用しない場合にくらべてマイクロベンチマークで最大 20 倍，マクロベンチマークで平均 1.5 倍程度の性能向上を達成したことを確認した．これによって本研究で検討した最適化手法の有効性を示すことができた．

YARV の開発では，VM（仮想マシン）記述言語および VM 生成系を作成することで仮想マシンの開発を容易にし，ソフトウェアの保守性を高めた．作成した VM 生成系は VM 記述言語で記述された VM 記述からコンパイラやアセンブラ，評価器のプログラム片を自動的に生成する．また，命令の融合操作，静的スタックキャッシングといった機械的に適用可能な最適化に関して, VM 生成系を用いて必要な命令を半自動生成することで仮想マシンの高速化を容易にした．この工夫により，本システムの方式検討を容易にし，実際に開発期間を大幅に短縮することができた．

メモリ共有型並列計算機上での並列実行による高速化を行うため，Ruby スレッド処理機構を計算機システムが提供するネイティブスレッドを利用して構成し，Ruby スレッドの並列実行をサポートした．対応したネイティブスレッド処理機構は POSIX Thread および Windows スレッドの 2 種類である．Ruby スレッドとネイティブスレッドの対応は，移植性を重視し，1 対 1 対応とする方式をとった．ネイティブスレッドを利用した Ruby スレッド処理機構を実現するためには，とくに Ruby スレッドに対する割り込みが問題となるが，ブロック解除関数を登録する方式を採用することで解決した．

Ruby スレッドの並列化にあたっては，スレッドセーフでない既存の拡張機能が問題となった．本研究では，このような膨大なプログラム資産をそのまま利用するためにスレッドセーフでない機能を呼び出すときにだけジャイアントロックを利用し，そのほかは細粒度ロックを利用して並列に動作するシステムを構築した．しかし，既存のスレッドセーフでない機能を順次スレッドセーフな処理に書き変えていくことで，段階的に並列度を向上することができるシステムとした．Ruby のようなオープンソースソフトウェアとして開発を進められている言語の性能や機能を継続的に発展させていくためには，この方法が適している．こうした上で，ジャイアントロックや細粒度ロック導入によるオーバヘッドを削減するためにデータ構造を最適化し，ロックを考慮した Ruby スレッドスケジューリングを実現した．これを実際にマルチコアプロセッサを搭載したメモリ共有型並列計算機上で実行し評価した結果，逐次実行に比べて利用するプロセッサコア数に応じた実行時間の短縮を実現出来たことを確認した．

本研究で開発した仮想マシン YARV は公式 Ruby 処理系に取り込まれることが決定しており，オープンソースソフトウェアとして公開される予定である．本研究によって世界中で多くの人に利用されるソフトウェアを開発するという実用的な面での貢献も行うことができた．今後はより詳細な解析や実行時フィードバックを利用した高速化，Ruby 向けの新しい並列化手法の検討が必要である．