審査の結果の要旨

氏名　笹田 耕一

計算機システムの需要の増大とともに，ソフトウェアに対する要求も大きくなり，迅速な開発が求められている．本研究は，近年急速に普及している記述力の高いプログラミング言語 Ruby を高速実行する仮想マシンとその並列化について述べたものである．

1．仮想マシン YARV の構築（第 2 章～第 5 章）

高速な Ruby 用仮想マシンの設計と評価について述べている．高速な Ruby 処理系開発に必要な課題を解決するために設計したのが，Ruby 用仮想マシン YARV（Yet Another RubyVM）である．YARV の特徴は以下の 4 点である．

(1) 実行時に Ruby プログラムをコンパイルして YARV 命令列へ変換・実行する．YARV の計算モデルはスタックマシンである．
(2) 例外処理は，例外が発生しない限り余計なオーバヘッドを生じない例外表方式と，従来型（setjmp/longjmp）のハイブリッド方式とした．
(3) 静的解析を必要としない実行時高速化技術を用いた．また，C 言語レベルで実現できる移植性の高い最適化を行い，環境によらず高速に実行できるようにした．
(4) スタックマシンモデルの仮想マシンのオーバヘッドへの対処法は以下の通り．
  - 命令ディスパッチ：ダイレクトスレッデッドコードの適用
  - 命令オペランドのフェッチ：上記と，頻出命令パターンから新規命令を生成するオペランド融合と命令融合の利用
  - スタック操作：スタックトップを特別なレジスタに格納する静的スタックキャッシングの実装
  - 命令本体の実行：メソッド検索結果を命令列中にキャッシュするインラインメソッドキャッシュの採用と，メソッド本体の小さい整数演算などを置き換える特化命令の採用

仮想マシンの構築は，文字の置き換えだけで行う簡単な VM 生成系を利用して行った．VM 生成系はユーザの簡潔な記述から仮想マシン本体のプログラム片やコンパイラ，アセンブラ，逆アセンブラのプログラム片を自動的に生成する．また，融合命令，静的スタックキャッシングで利用する最適化用の命令を自動的に生成する．この仕組みを利用することで保守性が向上し，高速化の試みが容易になった．

YARV は，旧 Ruby 処理系と比べ，マイクロベンチマークで顕著な性能向上を得ている．特に，整数演算を多用するプログラムでは特化命令により，最大 25 倍の性能向上を得た．また，他のスクリプト言語の処理系と比較しても，YARV の性能が優れていることを確認した．それぞれの最適化の効果はベンチマークプログラムによって異なるが，多くのベンチマークにおいて，仮想マシン化のみで 2 倍程度の性能向上を得ることができている．

2. Ruby の並列化（第 6 章～第 7 章）

Ruby の並列化について述べている. マルチコアプロセッサに代表される, コモディティ化した並列計算機上でプログラムを並列実行する高速化の需要は大きい. Ruby は言語レベルで並行実行のためのスレッド処理機構を有している. 既存の Ruby のマルチスレッドプログラムも並列実行によって性能向上が実現できる.

旧 Ruby 処理系では, Ruby スレッドを独自実装のユーザレベルスレッドを利用して実現していた. この方式は, 移植性は高いが, Ruby スレッドを並列実行することができない. この問題を解決するために, OS などが提供するネイティブスレッドを1対1で直接的に利用することとした. Ruby スレッドを並列実行するにあたり, 仮想マシンがメモリ保護違反などを起こさないようにするには適切かつ軽量な排他制御が必要になる. YARV において排他制御が必要になるのは以下の 4 点である.

(1) スレッド間で共有するデータ: データ管理表へのアクセスを排他制御した.
(2) オブジェクトの管理: 排他制御を激減するスレッドローカルなオブジェクト管理を行った.
(3) インラインキャッシュ: キャッシュを更新する際に毎回新しいキャッシュエントリを生成する. アクセス時に排他制御が必要ない点で有利.
(4) スレッドセーフでないネイティブメソッド: 膨大なプログラム資産と互換性を保ちつつ, 徐々に細粒度排他制御をもつスレッドセーフな実装に移行するための過渡的手段として, 既存のネイティブメソッドの実行時は VM に一つあるジャイアントロック（GL）を使う. ただし, GL 競合を監視し, 競合回数が閾値を越えると, 利用する CPU を制限して性能低下を抑える.

Ruby スレッドの生成, 合流に関しては, OS 依存のオーバヘッドによる性能低下があったが, スレッド切り替えは旧 Ruby 処理系に比べ 6～11 倍高速になった. 8 プロセッサコア上での並列実行により, 最大 7.4 倍の性能向上を得, 台数効果が確認できた. ただし, 台数効果が制限される例もある. また, GL の競合が頻発する場合も性能低下が発生したが, 動的な利用 CPU 制限により, 最悪値に比べて性能低下を 1/3.4 に抑えることができた.

本研究は, 近年注目されているプログラミング言語 Ruby の高速化に挑戦し, 次期の Ruby 公式リリース（Ruby 1.9.1）に採用される高い性能と, 信頼性・保守性を含めた高い品質を達成した. 本研究によって開発された Ruby 処理系がオープンソースソフトウェアとして公開された暁には, 世界中の Ruby プログラマに利用・参照されることとなる. すなわち, これまで実行が遅いために Ruby を利用することが出来なかったユーザに対し, 速度の問題を緩和し, 利用可能領域を広めるという研究の目的を達成することができた. 特に, Ruby の並列化は, 並列計算機がコモディティ化している現在, インパクトのある成果となった.

以上のように, 本研究は実用的なプログラミング言語処理系に関して世界的に認められる高いレベルの貢献を果たした. すなわち, 本研究は情報理工学に関する研究的意義とともに, 情報理工学における創造的実践に関して高い価値が認められる. よって, 本論文は博士(情報理工学)の学位請求論文として合格と認められる.