

Title	マルチコアプロセッサでのRubyプログラムの高速実行方式の開発
Author(s)	米澤, 直記
Citation	
Date	2009-06-19
Type	Research Paper
Rights	author

平成 21 年 6 月 19 日現在

研究種目：若手研究 (B)  
 研究期間：2007～2008  
 課題番号：19700046  
 研究課題名(和文) マルチコアプロセッサでの Ruby プログラムの高速実行方式の開発  
 研究課題名(英文) Ruby for Multicore Processor

研究代表者  
 米澤 直記 (YONEZAWA NAOKI)  
 神奈川大学・理学部・助手  
 研究者番号：70312832

研究成果の概要：本研究は、国産スクリプティング言語である Ruby を高速に実行する方式を提案し、評価するものである。具体的には、Ruby で書かれたプログラムを、マルチコアプロセッサ(1 チップに複数のコアを搭載するプロセッサ)で実行可能なコードに変換するトランスレータを実装した。評価のため、ヤコビの反復法で連立一次方程式を解くプログラムを実行した結果、4 個のコアを利用した場合に最大 3.95 倍、8 コアで同 6.67 倍の性能向上が得られた。

## 交付額

(金額単位：円)

	直接経費	間接経費	合計
2007 年度	600,000	0	600,000
2008 年度	500,000	150,000	650,000
年度			
年度			
年度			
総計	1,100,000	150,000	1,250,000

研究分野：並列プログラミング環境

科研費の分科・細目：情報学、計算機システム・ネットワーク

キーワード：超高速情報処理、コンパイラ、プログラミング言語、並列処理、マルチコアプロセッサ

## 1. 研究開始当初の背景

(1) Ruby の科学計算分野での普及  
 国産スクリプティング言語である Ruby は、数あるオブジェクト指向言語の中でも、少量のコーディングでアルゴリズムを簡潔に記述できるという特徴がある。また、Ruby を用いた Web アプリケーション開発のためのフレームワークである Ruby on Rails の普及により、Ruby は 2005 年ごろから産業界で多く用いられるようになった。

一方、学術分野においても、Ruby を利用した

プログラム開発例が多く見られるようになった。具体的には、<http://pub.cozmixng.org/~the-rwiki/?cmd=view;name=Ruby+for+Science> 等の Web サイトに多くの事例が挙げられている。また、情報処理推進機構 (IPA) で 2005 年度上期未踏ソフトウェア創造事業として開発および普及が進められた BioRuby、ChemRuby といったプロジェクトがある。

学術分野における Ruby 言語の普及の理由として、科学者本人が簡潔に素早く (Agile に)、

目的とするプログラムを開発することができる事が挙げられる。今後、Ruby で記述された科学技術計算コードは、更に蓄積されていくものと考えられる。

## (2) マルチコアプロセッサの普及

2006年9月の時点で、市場では、1チップにプロセッサコアが2基搭載された2コアプロセッサが投入されていた(事例: Intel Core 2 Duo)。2006年11月には、4コアプロセッサが投入された。マイクロプロセッサのこの方向での進化は当面続くことが学术界・産業界の共通認識であり、近い将来、数十を超えるコアを持つプロセッサが登場すると考えられる。

## (3) 並列実行を指示する OpenMP の利用の拡大

OpenMP は、並列プログラミングのための標準規格であり、プログラマは逐次コードに数行の OpenMP 指示文を追加することによって漸進的に並列化することができる。現在、C/C++ および Fortran のための規格が定められている。従来、並列プログラミング言語では、完全に並列化が完了しなければプログラムの実行が不可能であったのに対して、OpenMP では、漸進的な並列化により、高速実行可能なコードを得るまでの工程数が少なく済む。また、従来言語では、複数のスレッドの生成/管理、スレッド間通信、各スレッドが担当する計算範囲の指定といった並列実行のために必要なコードをプログラマが明示的に記述しなければならなかったのに対して、OpenMP では、コンパイラが並列実行のために必要なコードを自動生成する。

このように記述が簡潔であることから、Ruby と同様に OpenMP も Agile 開発のためのツールと位置づけることができる。

また、マルチコアプロセッサを推進する Intel の C++/Fortran コンパイラに OpenMP が取り込まれており、OpenMP を利用する事例が拡大している。

## 2. 研究の目的

本研究は、OpenMP に準じた指示文で並列化した Ruby プログラムをマルチコアプロセッサで実行可能とする、ソースコードレベルのトランスレータを開発する。指示文の行を消去すれば (あるいはコンパイラが無視すれば)、従来のシングルプロセッサ用のコードと同一であるため、既存の資源(コードおよび計算機)との共存性、連続性が高いと言える。評価対象として、Ruby で書かれた科学技術計算コードを設定し、これらのコードを OpenMP で並列化し、マルチコアプロセッサで実行す

る際の、記述の容易性、性能向上、資源利用の効率性等について検証する。

## 3. 研究の方法

### (1) トランスレータの実装

本研究で実装するトランスレータは、Ruby プログラムを読み込み、マルチコアプロセッサで実行可能なコードを生成する(図1)。実際に機能するトランスレータを早期に実現するため、サポートする OpenMP 指示文を絞った。具体的には、科学技術計算で多様される for 文を並列化する指示文を選定した。

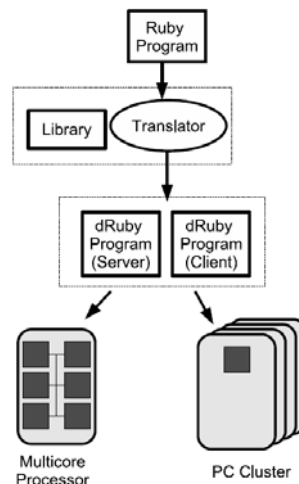


図1: 本研究で実装したトランスレータ

### (2) 評価

並列化した科学技術計算コード(ヤコビの反復法)を、マルチコアプロセッサで実行し、性能を評価する。評価では、本補助金で購入した、8個のCPUコアを持つパーソナルコンピュータを用いた。

## 4. 研究成果

### (1) 並列プログラムが利用するメソッドの実装

本研究では、逐次プログラムとして記述された Ruby プログラムを、並列実行するために、Ruby に付属する分散オブジェクトシステムである dRuby を用いた。dRuby が提供する機能を利用して、複数のプロセスが同期し、変数を共有する方式を確立し、メソッドとして実装した(図2,3,4)。

```
Server:
ga = []
DRb.start_service('druby://localhost:12345', ga)
$global_array = DRbObject.new_with_uri('druby://localhost:12345')

Clients:
DRb.start_service
$global_array = DRbObject.new_with_uri('druby://localhost:12345')
```

図2: dRuby の機構を用いた共有変数の実現

```

def quav_barrier_client pid, nprocs
  while $bar[pid] == 1
    end
    $bar[pid] = 1
    while $bar[pid] == 1
    end
  end
end

```

図 3 : dRuby の機構を用いたバリア動機の実現 (クライアント用)

```

def quav_barrier_server pid, nprocs
  sum = 0
  while (sum < nprocs - 1)
    sum = 0
    for i in 0..$bar.length - 1
      sum += $bar[i]
    end
  end
  for i in 0..$bar.length - 1
    $bar[i] = 0
  end
end

```

図 4 : dRuby の機構を用いたバリア動機の実現 (サーバ用)

### (2) トランスレータの実装

逐次プログラムを並列実行可能なプログラムに変換するトランスレータを実装した。このトランスレータはプロセス間通信が必要な箇所に上記のメソッドを生成する。

### (3) 手動変換による評価

上記の開発と並行して、ヤコビの反復法 (問題サイズ 2,048、反復回数 1,000 回) を手動変換し、8 個のコアから構成されるマルチコアプロセッサ (1 基あたり 4 コアを持つ 3GHz Intel Xeon が 2 基搭載された Apple 社 Mac Pro) で実行した。その結果、コア数 1 個の実行時間に対して、4 個で 3.95 倍、8 個で 6.67 倍の台数効果が得られた (図 5, 6)。

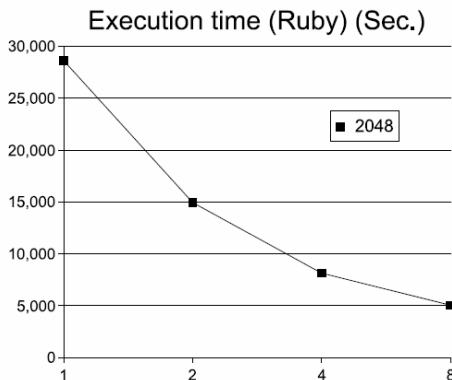


図 5 : ヤコビの反復法の実行時間 (実行環境 : Ruby 1.8)

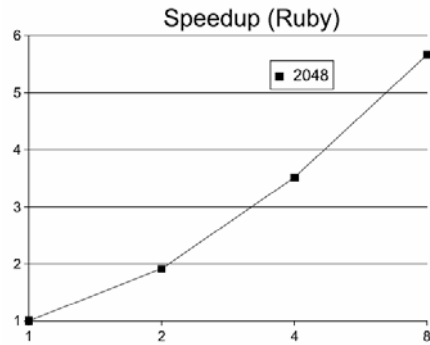


図 6 : ヤコビの反復法の台数効果 (実行環境 : Ruby 1.8)

ただし、C プログラムで記述したヤコビの反復法の実行時間と比較すると、Ruby の実行時間が 2000 倍以上大きい結果となった。一般に、コンパイル方式の C と比較してインタプリタ方式の Ruby が実行時間が大きいため、この結果は想定内である。Ruby は、プログラムの開発に要する時間が短いという特長があるため、実行時間だけではなく、開発時間を含めた総合的な評価が必要であると考えられる。

得られた結果と本研究の構想を、日本 Ruby 会議 2007 (2007 年 6 月 9, 10 日) で発表した。

### (4) 次世代 Ruby 処理系を用いた実装

本研究の研究期間中に、実行性能の改善を謳った JRuby や Ruby 1.9 といった次世代の Ruby 処理系が登場した。

平成 20 年度においては、JRuby を用いた実験を行なった。その結果、上記と同じヤコビの反復法の実行時間が Ruby 1.8 と比較して、2 プロセッサ時に 93.6%、8 プロセッサ時に 82.3% 短縮した (図 7)。ただし、1 プロセッサの実行時間が約 4 分の 1 になったことにより、台数効果自体は大幅に悪化することになった (図 8)。本研究ではプロセス間通信として dRuby を利用したが、JRuby ではネイティブスレッドを実現しているため、今後は並列処理の単位としてスレッドを利用することによってマルチコアプロセッサの性能が引き出せる可能性がある。

また、本研究で扱う共有メモリ型プログラムのオーバーヘッド要因であるバリア同期に注目し、その同期処理に関する研究を進め、成果について論文誌で公表した。

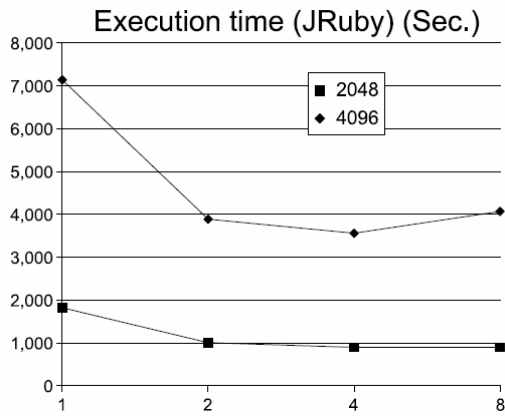


図7：ヤコビの反復法の実行時間  
(実行環境：JRuby 1.1.1)

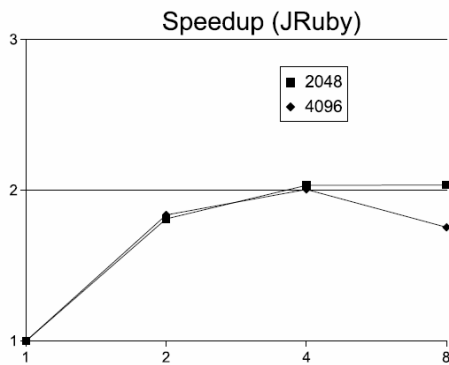


図8：ヤコビの反復法の台数効果  
(実行環境：JRuby 1.1.1)

## 6. 研究組織

### (1) 研究代表者

米澤 直記 (YONEZAWA NAOKI)  
神奈川大学・理学部・助手  
研究者番号：70312832

### (2) 研究分担者

### (3) 連携研究者

## 5. 主な発表論文等

(研究代表者、研究分担者及び連携研究者には下線)

〔雑誌論文〕(計1件)

米澤直記, 和田耕一, "バリヤ同期除去による行列演算プログラムのアイドル時間の削減", 電子情報通信学会論文誌 Vol. J91-D, No. 4, pp. 907-921 (2008) 査読あり

〔その他〕

博士学位論文: 米澤直記, 分散メモリ環境における共有メモリ型プログラムの高速実行方式の研究, 2008年9月, (神奈川大学)

口頭発表: 米澤直記, マルチコアプロセッサでのRubyプログラムの高速実行方式, 日本Ruby会議 2007, 2007年6月