Ractorを用いたRubyの並列処理性能評価とRubocopによる並列コード記述支援

成蹊大学理工学部情報科学科　ソフトウェア研究室　S202148　柳澤快

1. 研究背景

　Rubyは1995年に一般公開されたオブジェクト指向スクリプト言語である。Rubyは多くの支持を集めているが、こうした成長はWebアプリケーションフレームワークRuby on Railsの人気に起因している。Ruby on Railsは、開発工数を抑えられ、比較的短期間で目的のサービスやアプリ開発の実現を最大の特徴としていることから、Rubyは多くの企業で採用されている。  
　また、近年、コンピュータの性能はマルチコアプロセッサの普及により向上している。それに伴い、ソフトウェアの並列処理能力が重要となっている。並列計算機上で複数の処理を同時に実行するために、多くのプログラミング言語では複数スレッドを同時実行させることができる。しかし、RubyのスレッドはGIL(Global Interpreter Lock)によって同時に実行可能なスレッドは1つのみとなっている。これにより、スレッドを使う限り、通常の方法では並列プログラムをRubyで記述することはできない[1]。  
　Ractorの登場により異なるRactor間でスレッドを並列に実行することが可能になった。Ractorは開発途上であるものの積極的に改善が進行中であり、将来的にはRuby on Railsのアプリケーションサーバにおいて、リクエストを並列に処理することで更なるパフォーマンス向上などが期待されている。本研究では、このような将来性を見据えて、Ractorのより良い記述方法を明らかにし、静的解析ツールRubocopによるRactorの記述を支援することを目指す。

1. Ractorの概要

　本章では、並行・並列処理を可能にするActor Modelの抽象化であるRactorについて説明する。

* 1. Ractorとは

　Ractorとは、2020年にリリースされたRuby3.0で導入された並行・並列処理を可能にする機構であり、スレッドセーフな並列実行を提供するRubyのActor Modelの抽象化である。

　Actor Modelは1973年に米国マサチューセッツ工科大学のCarl Hewitt氏によって発表された並列計算の数学的モデルの一種である。Actor Modelの構成を図1に示す。

じょうごグラフ が含まれている画像

自動的に生成された説明

図 1 : Actor Modelの構成[2]

　Ruby処理系を起動すると、1つのRactorを作成する(これをMain Ractorと呼ぶ)。同一Ractorは最低1つのスレッドを持つ。同一Ractor内の複数スレッドはグローバルインタープリターロックにより同時には実行されない。

1. 実験

　バブルソートを用いてRactorの実験を行った。実験はApple M1チップ搭載のMacBook Air(8コアCPU、 macOS Sonoma 14.5)上のRuby3.3.4で実行させた。

* 1. 方法

バブルソートの並列化の方法については図2に示す。整数の乱数を指定の数だけ生成し、配列に格納する。

ダイアグラム

中程度の精度で自動的に生成された説明

図 2 : Ractorを用いたバブルソートの並列化

　配列要素の中央値を基準に次の手順に従い2つに分割する。配列の左端から右に向かって、中央値以上の値を探し、右端から左に向かって中央値未満の値を探索する。見つけた2つの値を交換し、これにより、中央値未満の値が左側に、中央値以上の値が右側に移動する。この操作を左右からの探索が衝突するまで繰り返す。2つのRactorを生成し、各Ractorに分割された部分配列を送信する。その後、各Ractorで昇順にバブルソートを実施する。各Ractorからソートされた配列を受け取り、それらを結合する。

　筆者は、C言語とMPIを用いてバブルソートの並列化を行った経験があるため、Ractorでもこれを用いることにした。

* 1. 結果

　実験の結果、C言語とMPIを用いて、2並列でバブルソートさせた時、逐次実行から約4倍の性能向上だったが、Ractorでは最も良い場合で約2.8倍の性能向上となった。メモリ再配置の頻発や共通の配列要素アクセス時の排他制御などの原因が考えられ、これらのRactorのオーバーヘッドは現状ユーザー側の記述の工夫だけで避けるのは難しいことがわかった。

　これらの課題があるものの、背景でも述べた通り、Ractorは今後も改善が進められ、将来的な発展が期待される。そこでRactor の記述を支援し、より扱いやすくするために Rubocop の活用を提案する。

1. 静的解析ツールRubocopとは

　静的解析ツールにはRubocopを用いる。Rubocopとは、Rubyの静的コード解析およびコードフォーマッターの機能を持つライブラリである。コード内で発見された問題を報告するだけでなく、Rubocopはそれらの多くを自動的に修正することも可能である[3]。

　複数人で開発を行っていても書き方を統一することが可能であり、コードの品質向上に貢献することができるという利点があることから、Rubyを採用している企業では、Rubocopの導入事例は多く存在する。

1. Rubocopによるコード支援

　ここでは、実際に実装した指摘の1例を記載する。Ractor間の通信では、Receive処理に対応するSend処理が存在しない場合、Receive処理で無限に待機状態となる。この問題のコードを図3に示す。

グラフィカル ユーザー インターフェイス, アプリケーション

AI によって生成されたコンテンツは間違っている可能性があります。

図 3 : 問題のあるコード例

　この問題のコードに対して、Receive処理がある場合、対応するSend処理が存在するかをチェックする指摘についてRubocopを用いて実装した。

　実装したカスタムルールについて図4のサンプルプログラムを用いて、動作を示す。

テキスト

AI によって生成されたコンテンツは間違っている可能性があります。

図 4 : Ractor Send/Receive サンプルコード

使用するサンプルコードについて説明する。最初に、新しいRactorを生成する。このRactorの中で、Receive処理を使ってメッセージを受け取り、受け取ったメッセージをそのまま返す。次に、Send処理を実行して、Ractorに文字列’ok’というメッセージを送信する。最後に、takeメソッドを使ってRactorから返された文字列’ok’を受け取り、標準出力に返す。サンプルプログラムの実行結果を図5に示す。

テキスト

AI によって生成されたコンテンツは間違っている可能性があります。

図 5 : source/send\_receive.rbの実行結果

Send処理でRactorに送信した文字列’ok’が標準出力されていることが確認できる。この状態でCLI上でrubocop ./source/send\_receive.rbを実行した結果を図6に示す。

グラフィカル ユーザー インターフェイス が含まれている画像

AI によって生成されたコンテンツは間違っている可能性があります。

図 6 : rubocop ./source/send\_receive.rbの出力

　Receive処理に対応するSend処理が存在するため、Rubocopでの警告は出力されない。  
　次に、サンプルプログラム内のSend処理(r.send ‘ok’)をコメントアウトして実行する。しかし、実行すると、強制終了しない限り、Ractor内のReceive処理が送信されるのを無限に待機している状態になる。Rubocopを実行すると、Receive処理に対応するSend処理が存在しないため、実装したカスタムルールの警告が出力される。この時のRubocop実行結果を図7に示す。

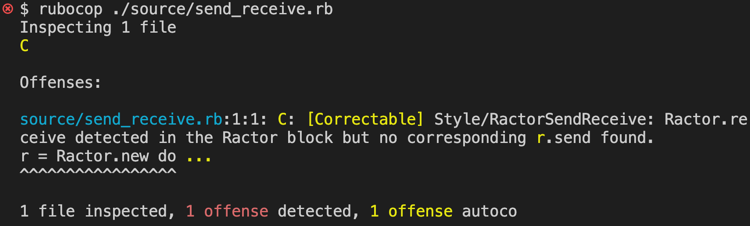


図 7 : Send処理がない場合のrubocop実行結果

また、VSCodeであれば、作成したカスタムルールを読み込み、コード上に警告が出力され、Rubocopを手動で実行する手間を省くことが可能である。この様子を図8に示す。

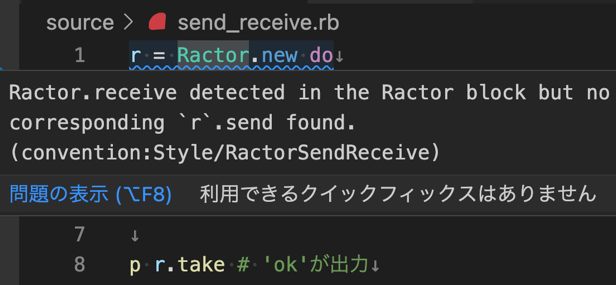


図 8 : コード上でのRubocop警告

さらに、実装したカスタムルールを自動修正機能に対応させた。実際に自動修正機能を実行してみるとRactorの処理の後にSend処理が追加される。自動修正機能を使用する場合はrubocopコマンドを-Aオプション付きで実行する。開発者は、自動で挿入されたSend処理に引数を記述するのみでよくなる。実際に、自動修正機能を実行した結果を図9に示す。8行目にSend処理が自動で追加され、図下部の出力では自動修正機能を実行し、修正されたことが出力されている。

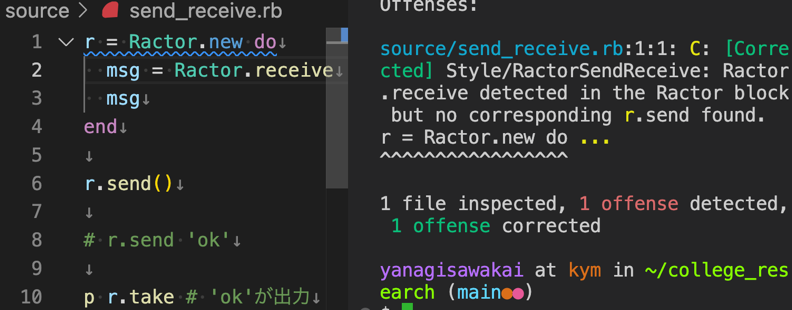


図 9 : 自動修正機能の実行

　他にもRubocopによるRactorの記述に対するカスタムルールの実装を行なった。これらのカスタムルールの実装により、開発者がデバッグに費やす時間を短縮することができる。また、Ractorの使用に不慣れな開発者でも、制約を自然に学びながらコードを書くことが可能となる。

1. まとめ

　本研究では、Rubyの並列処理機構Ractorを性能評価し、その利点と課題を明らかにした。実験では、メモリ再配置や排他制御の影響による性能低下も確認され、これらの課題はあるものの、Ractorは今後も改善が進められ、将来的な発展が期待される。そこでRactorの記述を支援し、より扱いやすくするためにRubocopの活用を提案した。Ractorは並列処理の安全性と効率性を両立し、GILの制約を超えた並列実行を実現するが、新たなプログラミングモデルの理解が求められるため、学習コストや記述ミスが課題となる。これに対応するため、RubocopにRactor専用のカスタムルールを実装し、潜在的なエラーの検出や自動修正を可能にすることで、デバッグ時間の削減や効率的なコード記述を支援した。本研究で実装したルールは、初心者にも有用であり、Send/Receiveの対応関係などの検証機能や自動修正機能を通じて、Ractor特有の仕様を学びながら利用できる環境を整備した。一方で、さらなるルールの追加や解析速度の最適化が課題として残り、特に大規模コードに対する効率的な解析が求められる。本研究は、Ractorをより扱いやすくするものであり、Rubocopの拡張を通じてRactor以外のRubyプログラムへの支援も期待される。

参考文献

1. 笹⽥耕⼀：「Ruby 向け並列化機構 Guild の試作」. 情報処理学会プログラミング研究会. 2018
2. “「アクターモデル」による並列処理プログラミング入門”. SIOS Tech Lab. <https://tech-lab.sios.jp/archives/8738>, (参照 2025-2-2)
3. “rubocop/robocop”. GitHub. <https://github.com/rubocop/rubocop>, (参照 2025-2-2)