Vol.2011-HPC-130 No.2
2011/7/27

多種言語処理系性能の評価に適したベンチマークプログラム

野瀬貴史[†1]　泊久信[†1]　平木敬[†1]

高級・高生産な言語によるハイパフォマンスコンピューティングの必要性が高まっているが，決定打となる高級言語は現状存在しない．このため，複数の言語の中から，プログラムを記述する際の生産性とプログラム実行速度とのトレードオフを考慮し選択することが必要である．検討のための指標の一つとして，同じベンチマークプログラムを各種言語で実装し，その実行速度を計測するというものがある．しかし既存のベンチマーク集では人力による最適化や，性能と同時にコードの生産性を測っていることによる不確実性と，人力で実装をするためにベンチマークの規模を拡大できないという問題があった．本論文では，基礎となるベンチマークプログラムからトランスレータを用いて各種言語へ移植し，人力で移植した場合に発生するチューニングのバラつきを排除した性能比較を行う．また基礎ベンチマークが備えるべき性質と，それを満たすベンチマークプログラムの構成を提案する．

Benchmark Programs Suitable for Evaluating Various Programming Launguage Implementation

Takafumi Nose ,[†1] Hisanobu Tomari [†1]
and Kei Hiraki[†1]

High-performance computing (HPC) by using a high-level/high-productive language is increasingly needed, but there is no decisive high-level language for HPC. Therefore, programmers must select a language to use by considering trade-off between programming productivity and program execution speed. One tool to consider the trade-off is to implement the same benchmark program with different languages and measure performance of them. However, existing benchmarks that adopt such method have two problems: One is a uncertainty caused because they are optimized by hand, and measured both performance and productivity at the same time, the other is a restriction of program size because they are implemented by human. In this paper, we propose that an automatic language-to-language translator that keep the same semantics between base language and destination language can be the solution to compare performance of multiple languages fairly, and show results of translated benchmarks.

1. はじめに

ハイパフォーマンスコンピューティングにおいては，プログラムの実装に Fortran と C が使われることが多い．しかし，その生産性は Java や Ruby などのより高級な言語と比較して低い．より高級・高生産な言語によるハイパフォーマンスコンピューティングが望まれていることは，X10，Chapel といった PGAS 言語の登場や，Ruby，Python に代表される軽量言語に科学技術計算向けライブラリが存在することから明らかである．しかし，現状では科学技術計算において Fortran/C と同等のプログラム実行速度を達成でき，かつ Fortran/C と比較して高い生産性を持つ言語に決定打は存在せず，アプリケーションの規模や性質に応じて生産性と性能のトレードオフを考慮し，プログラムを記述する言語を選択する必要がある．

生産性については，プログラムの簡潔さのみならず，ライブラリの充実度，統合開発環境の支援，型チェックによる堅牢性，さらには開発者個人の趣味などの要因に左右されるため，その定量的評価は困難である．しかし，実行速度については，同じベンチマークプログラムを各種言語に移植することで言語処理系間の比較をすることができる．

しかし，既存の複数言語による実装を備えたベンチマークでは，人力で実装しているために，実装者の各言語への習熟度の差や，気に入った言語に対して最適化の努力を集中してしまうという恣意性により，言語によってチューニングにばらつきが発生する可能性がある．また，移植の労力が大きいため，ベンチマークの規模を拡大しづらいという欠点がある．例えば 1) はベンチマーク対象言語の数は多いもののベンチマークプログラムそのものの規模は小さく，2) は実装言語の種類が少なくベンチマークで測る項目が限定されている．この二つは人力で実装されており最適化は実装によってまちまちである．

これらの欠点は，基礎となるある程度の規模を持ったベンチマークプログラムを決め，なるべくもとのベンチマークとの一対一対応が取れるように他言語へ自動変換するトランスレータを用いてベンチマークを構成することにより克服することができる．一対一対応のルールに基づいて移植することで特定の言語に対する過度なチューニングを排除することができ，言語間での対応関係がソースコードの行レベルで厳密につくために性能比較が容易に

†1 東京大学
The University of Tokyo

なる．またトランスレータにより移植作業が自動化されるため移植に伴うコストが削減され，プログラムの規模を拡大しやすくなる．

本論文では，基礎ベンチマークが備えるべき性質と，それを満たすベンチマークプログラムの構成を提案する．また，基礎となるベンチマークプログラムからトランスレータを用いて各種言語へ移植し，人力の場合に発生するチューニングの恣意性を極力排除した性能比較を行う．

2. 基礎ベンチマーク

2.1 実装言語

言語同士の比較をするための基礎ベンチマークの実装言語は，既存のベンチマークの実装言語の中では Java が最適である．

まず，C や C++と違ってプリプロセスやマクロが存在しないため，プログラマが認識するソースコードと実際にコンパイルされるソースコードの間の乖離が発生せず，変換後のソースコードの一対一対応を取りやすい．また，マクロはユーザーの多い言語では C，C++，Lisp にしか存在せず，十分に普遍的であるとは言いがたいので，比較の公平性の観点や変換のしやすさという観点からもマクロの存在しない言語が望ましい．次に，Java は静的型付けであり，かつ型推論機能がないためトランスレータ側で型に関する機能を充実させなくても良い．静的型付けが望ましいのは，動的型言語から静的型言語へ変換したとき，型を決定できない場合は Variant 型を使わなければならないので性能劣化が発生し，静的言語が著しく不利になる一方，静的型言語から動的型言語へ変換した場合は Ahead-of-Time コンパイルや Just-in-Time コンパイルで性能を向上させることができるので，動的型言語から静的型言語への変換に比べ不利ではなくなるからである．さらには，Java にはクラスベースオブジェクト指向の基本的な機能が整っており，高級な言語が備えるべき最低限の要件とすることで，Fortran/C より高級・高生産な言語の性能を測りたいという目的に合致する．

2.2 変換元の候補

同じベンチマークを多言語に移植し，多言語処理系間の性能比較を行う試みには The Computer Language Benchmarks Game[1] がある．2011 年 6 月時点で Java を含む 27 種の言語に関して測定が行われており，同様の試みの中では我々の知る限り最多なので，出来る限り多くの言語を比較したい場合にはもっとも有用なベンチマーク集である．ただ，1) では移植および最適化はボランティアの貢献に基づいているため，全ての言語に対してソースコードの提供が行われているわけではない．また同じ言語でも複数の実装が投稿されることがあり，言語に対する評価はその中で最良の実装に基づいて行われる．つまり，言語というよりは各実装の評価を行っているベンチマークと言うべきであり，これをそのまま使うことには問題がある．1) は 13 種類のベンチマークすべてに対して Java 版が存在するので，自動変換により手動最適化版との比較や欠けたベンチマークの補完を行うことはできるが，含まれるプログラムのうちマイクロベンチマークでないものはバイオインフォマティクスに関連するものに偏っている．また実装の行数が最大でも 180 行程度であり，小さなベンチマークとして有名な Dhrystone[3] の Java 移植版が 300 行程度になることをあわせて考えると，数値計算の評価用としては実装・評価の形態のみならずプログラムの内容も不適切なベンチマークである．

性能評価の客観性からは，実際のアプリケーションのコードを反映し，アーキテクチャの評価に広く用いられ，性能評価データが多く存在する SPEC を移植することが望ましい．しかし，SPEC の実装は C, C++, Fortran で構成されており言語が統一されていないこと，またオリジナルの SPEC は，Xeon E5530 2.4GHz を用いた場合実行するのに 3 時間以上かかり[4][5]，これを高級言語に移植した場合は実行時間が一部言語で 100 倍以上長くなるため，SPEC を使用することは現実的でない．

NAS Parallel Benchmarks (NPB)[6] にはバージョン 3.0 から Java 版が存在し，実装言語が統一されている．また，NPB は 7) で SPEC CFP 2006 との高い相関が示されているため，客観性を確保できる．ベンチマークプログラムの規模は行数にして最大でも数千行で，問題サイズ A の実行にかかる時間は Xeon E5530 2.4GHz の場合数分で済み，たとえ 100 倍遅い言語処理系に移植したとしても数時間で実行を終えることができるため，SPEC とオーダーで変わらない実用性を得ることができる．ベンチマークプログラムの内容は数値計算で用いられるカーネルと数値流体力学に基づいており，ハイパフォーマンスコンピューティングにおける性能を測るという目的に合致する．Dhrystone も 7) で SPEC CINT 2006 との相関が示されているので，NPB と同時に使うことで SPEC と同等の客観性をもった比較を行うことができる．また，Dhrystone はベンチマークとしては 300 行程度の規模なので，NPB での実験の前に浮動小数点数演算を除いた概略的な性能を知るのに都合が良い．以上の理由から，NPB と Dhrystone を用いた比較を行うことにした．

3. トランスレータの実装

一般的なプログラミング言語において，配列やリストなどの基本的なデータ構造，ループや分岐に関する基本的な制御構文，そしてオブジェクト指向に関する機能は，表現力には差

があるものの共通して備えられている．よって，基礎ベンチマークから各言語へ変換するとき，意味論の一対一対応を保持したままプログラムを変換するための変換ルールを構成することが容易に可能である．我々は Java の構文木から以下に述べる応急処置を除き変換前と一対一対応となる対象言語のソースコードを出力するトランスレータを実装した．

3.1 For 文の表現力

For 文の表現力は制御構造の中でも言語間の差異が大きい（**表 1**）．このため，他の言語に単純には変換できないケース（**図 1**）が存在するため，その場合は while 文で代用する変換ルールとなっている．

表 1 各言語の For 文（Fortran 2003 では Do）の差異

	条件式	刻み幅の指定	無限ループ
Java	有	増減式	条件式の省略
C99	有	増減式	条件式の省略
Fortran 2003	無	有	不可
Ruby	無	Range オブジェクト	Infinity を含む Range オブジェクト
Python	無	Range オブジェクト	不可
PHP	有	増減式	条件式の省略

3.2 I/O，メモリ確保

I/O，メモリ確保に関しては，言語間の仕様に差異があるが，それを行う部分は数値計算部分のコード量・実行時間と比較して少ないため，差異を吸収できない一部は手動で個別に実装した．

3.3 オーバーフローの扱い

Ruby，Python では整数演算時にオーバーフローが発生すると数値型が自動的に多倍長整数に変換される．また，PHP では浮動小数点数に変換される．これらに起因する問題はトランスレータで回避することは難しい．実際に NPB の PHP への移植でオーバーフロー時の動作に起因する不具合が発生したため，乱数生成に関する 1 箇所の実装をオーバーフローを回避する実装に変更した．

3.4 継承の C 実装における再現

NPB Java 版の各プログラムはクラスの継承を利用しているが，C の構造体には継承機能が存在しないため，何らかの形で再現する必要がある．今回は簡単のためにカプセル化などは行わず，基底クラスを表す構造体に派生クラスのメンバ変数を単純に付け足したものを派生クラスを表す構造体として扱う実装として出力するようにした．

```
//変換前の Java ソース
for(i=isize-1,j=0;i>=0;i--,j++){
  処理内容 ();
}

#変換後の Ruby ソース
i = isize - 1
j = 0
while i >= 0 do
  処理内容 ()
  i -= 1
  j += 1
end
```

図 1 Java から単純に変換を行えないケースの例

4. 性能評価

変換した NPB のシングルスレッド性能を**表 2** に示した言語および言語処理に関して測定した（**図 2**〜**図 5**）．また，Ruby 版のマルチスレッド性能を，Ruby 1.9.2-p136, Ruby 1.9.3-110206, JRuby 1.6.0.RC1, Rubinius 1.2.0 について測定した（**図 7**）．変換した Dhrystone の性能を，LuaJIT 2.0.0 beta6 を加えてイテレーション回数 50000000 で測定した（**図 6**）．評価環境には Dell PowerEdge R410 (Xeon E5530 2.4GHz，CentOS 5.5，メモリ 12GB) を使用した．処理系のコンパイルには GCC 4.5.1 を用い，最適化オプションは-O3 -msse4.2 を指定した．JRuby および Jython の実行には Sun JDK 1.6.0 Update23 を使用した．ベンチマークの問題サイズはすべて A であった．一部データが欠損しているのは，変換後のベンチマークのバグあるいは処理系のバグにより実行が完了しなかったものである．

5. 議　　論

JVM の結果では，おおむね IBM Java, JRockit, Harmony の順に性能が高くなる傾向が見て取れるのに比べ，Sun JDK の他 JVM に比べた性能がベンチマークによって大きく変動していることがわかる．NPB のような数値計算では，Sun の JVM ではなく他の JVM

表 2 NPB シングルスレッド性能の評価を行った言語処理系一覧

Java	Sun JDK 1.6.0_23
	Sun JDK 1.7.0-ea-b127
	IBM Java 6.0-9.0
	Oracle JRockit JDK 1.6.0_20-R28.1.0-4.0.1
	Apache Harmony-6.0-jdk-991881
Ruby	Ruby 1.8.7-p330
	Ruby 1.9.2-p136
	Ruby 1.9.3-110206
	JRuby 1.6.0.RC1
	Rubinius 1.2.0
	Rubinius 1.3.0dev hydra
Python	Python2.7.1
	pypy 1.4.1
	Jython 2.5.2.RC3
	unladen-swallow (–without-llvm)
PHP	PHP 5.3.6
C99	GCC 4.5.1 (-O3 -msse4.2)
	ICC 11.1.073 (-fast -xSSE4.2)
	LLVM 2.8(-O3 -msse4.2)
Fortran 2003	IFORT 12.0.2 (-fase xSSE4.2)
	GCC 4.5.1 (-O3 -msse4.2)

を使用したほうが安定した計算速度を得られると言える．

Ruby 1.8.7 と 1.9.x の間では，常に性能に 2 倍以上の差が安定に存在する．一方，JRuby の性能に着目すると，1.9.x にくらべ JRuby の性能が安定しないことがわかる．この傾向が JRuby 実装に起因するものか，JVM に起因するものかを判断するにはさらに Sun 以外の JVM 上で JRuby を実行する実験が必要である．

ネイティブコンパイルする C99, Fortran 2003, オリジナルの比較では，IS において性能にほとんど差が生じていない．これは IS の処理内容は整数ソートであり，性能が I/O に律速されるためである．また，IS ほどではないが CG でも全ての処理系が同じような性能を示している．Ruby, Python, PHP において CG の Mops が他のベンチマークに比べ大きいこと，特に pypy での加速が顕著であることを考慮すると，他のベンチマークに比べ何らかの特異な性質がある可能性がある．

JRuby を除く Ruby 処理系には Giant VM Lock (GVL) と呼ばれる Lock が存在し，スレッドが複数存在していても，同時に実行出来るスレッド数が 1 つに限られるという制限がある．このため図 7 では JRuby のみがスレッド数に応じたスケールをしており，他の処

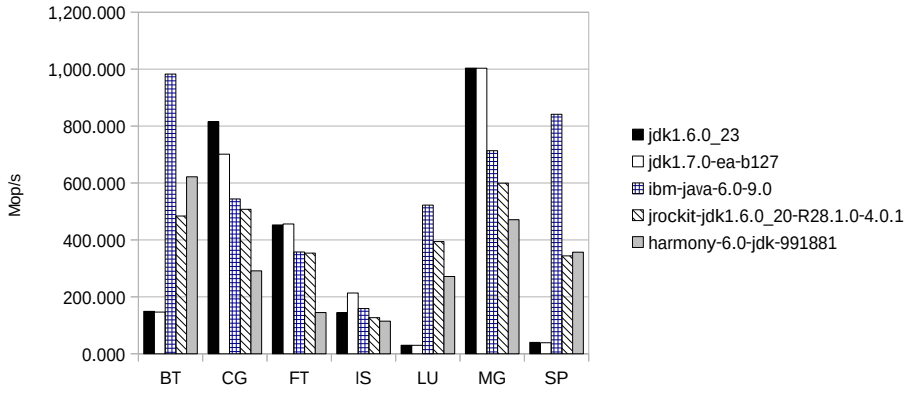

図 2 各種 JVM の性能比較

理系では GVL のコストを測るグラフになっている．IS の JRuby のグラフからは，他のベンチマークに比べスケールしにくいことが読み取れ，I/O に律速されていることを示している．他の Ruby 処理系では Rubinius, Ruby1.9.3, Ruby1.8.7 の順に悪いスケーリングとなっており，Ruby1.9.3 は 1.8.7 に比べ演算性能は向上しているものの GVL のコストは上がっていることが分かる．

Dhrystone の結果では，LuaJIT がスクリプト言語としては顕著な性能を見せているものの，なおも Java や C に比べ 2 桁の性能差をつけられている．

6. ま と め

本論文では多言語処理系の比較に適した基礎ベンチマークの実装言語は Java が適していることと，基礎ベンチマークの実装として，SPEC より小さくかつ十分な規模と客観性を持つ NPB と Dhrystone を用いることを提案した．

トランスレータによる 1 対 1 対応のついた自動変換によりベンチマークを生成することによって，手動での移植による欠点，すなわち実装者の技量や恣意性により最適化の度合いに差が発生し，言語処理系そのものの性能測定を困難にする可能性を排除し，多数の言語処理系の性能を公平性を確保しながら比較することができた．

その結果，各言語処理系の持つ性能特性と，ベンチマーク自体が持つ傾向に関する知見を

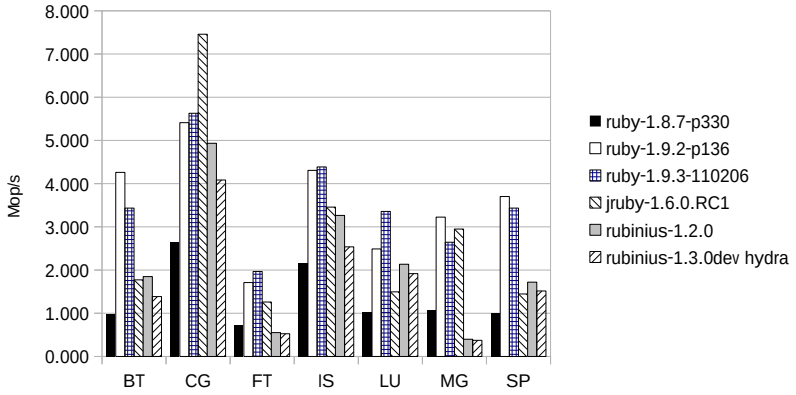

図 3 各種 Ruby 処理系の性能比較

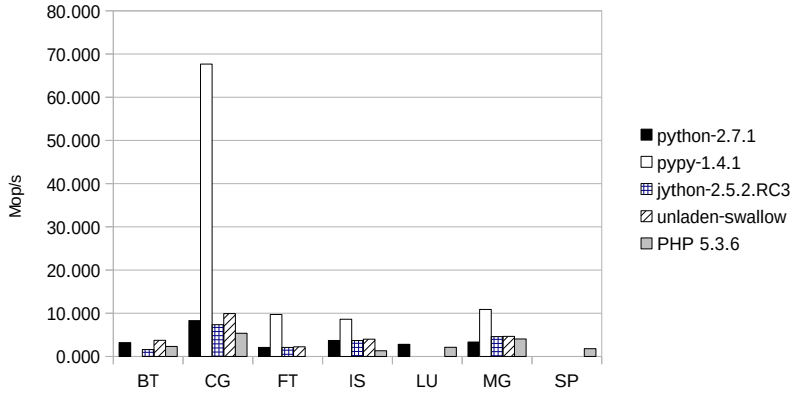

図 4 各種 Python 処理系と PHP の性能比較

得ることができた．得た知見は以下の通りである：NPB のような数値計算を行う際には Sun の JVM より IBM Java, JRockit，Apache Harmony のほうが性能が安定する．JRuby の性能も安定しておらず，JRuby または JVM，あるいはその両方に問題を抱えている可能性がある．Ruby1.9.x は 1.8.7 にくらべ安定して数値演算性能の改善が達成されているが，マルチスレッドを使用する際のコストは上がっている．NPB の IS，CG は他のベンチマークに比べ変換後も性能が落ちにくいという性質を持っている．LuaJIT はスクリプト言語の中では速いが，それでも大規模な数値計算に使うには 100 倍以上の性能改善が必要である．

7. 今後の課題

今回我々は主に NPB を用いて言語間比較を行ったが，改善すべき点も存在する．基礎ベンチマークが Java に最適化されているため，その時点で恣意性が発生している．また，NPB 並列化版のプログラミングモデルは分散メモリ環境に適していない．そして，数値流体力学の計算が主であるので，基礎ベンチマークを，例えば SPECjvm2008[8] の内容を取り入れることによりベンチマーク内容をより普遍的にする必要がある．これらを解消したベンチマークとなるよう NPB を改造する，あるいはより適切な新規ベンチマークを作成することが今後の課題である．

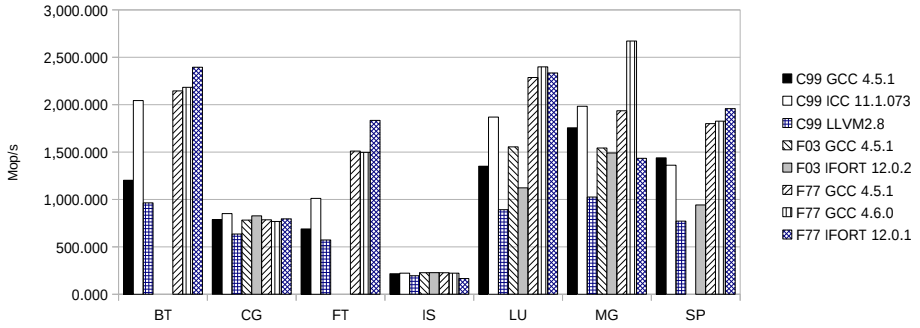

図 5 ネイティブコンパイルする処理系の性能比較

参 考 文 献

1) Gouy, I.: The Computer Language Benchmarks Game, http://shootout.alioth.debian.org/.

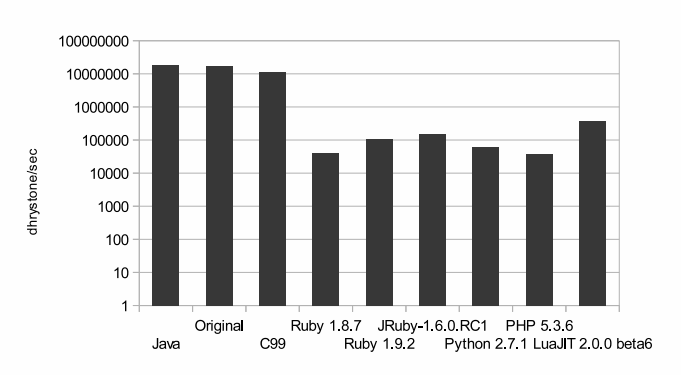

図 6 Dhrystone を用いた性能比較

2) Hundt, R.: Loop Recognition in C++/Java/Go/Scala.
3) Weicker, R.: Dhrystone: a synthetic systems programming benchmark, *Communications of the ACM*, Vol.27, No.10, pp.1013–1030 (1984).
4) Standard Performance Evaluation Corporation: CINT2006 Result: Dell Inc. PowerEdge R410 (Intel Xeon E5530, 2.4 GHz), http://www.spec.org/cpu2006/results/res2009q2/cpu2006-20090609-07852.html.
5) Standard Performance Evaluation Corporation: CFP2006 Result: Dell Inc. PowerEdge R410 (Intel Xeon E5530, 2.4 GHz), http://www.spec.org/cpu2006/results/res2009q2/cpu2006-20090609-07851.html.
6) Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Frederickson, P., Lasinski, T., Schreiber, R. et al.: The NAS parallel benchmarks, *International Journal of High Performance Computing Applications*, Vol.5, No.3, p.63 (1991).
7) Tomari, H. and Hiraki, K.: Retrospective Study of Performance and Power Consumption of Computer System, *SACSIS2011* (2011).
8) Standard Performance Evaluation Corporation: SPECjvm2008, http://www.spec.org/jvm2008/.

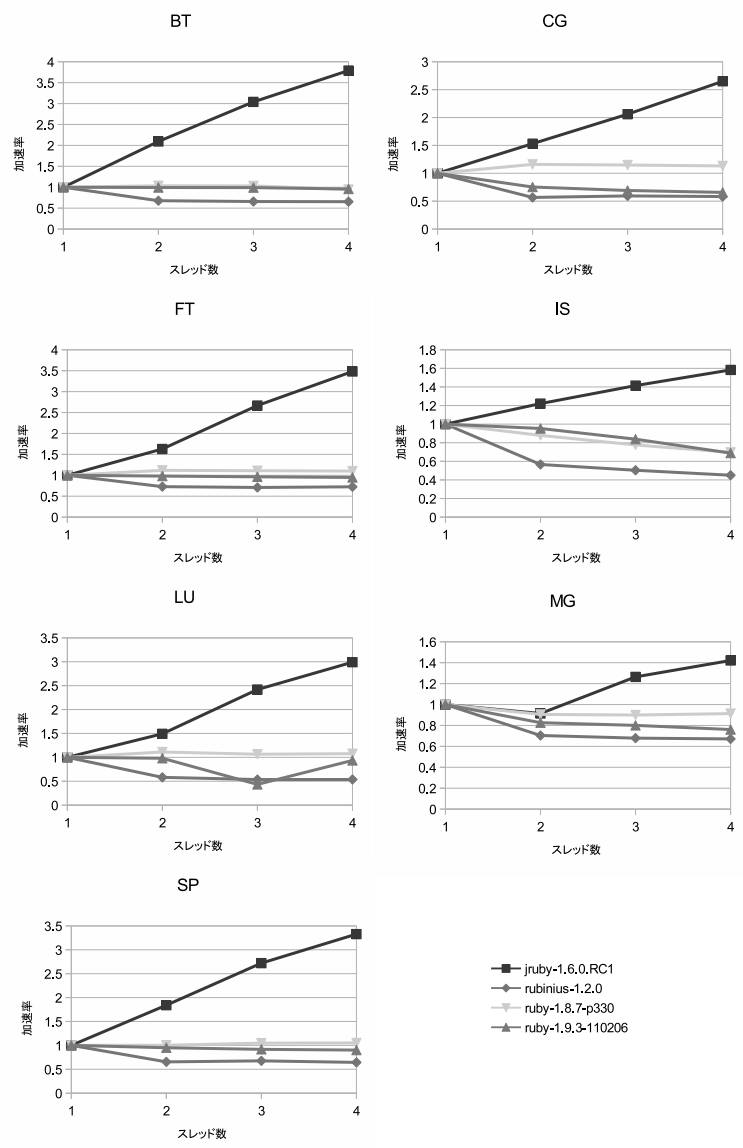

図 7 Ruby 版マルチスレッド性能比較