FX10 4800ノードを用いた
密行列向け固有値ソルバEigenExaの性能評価

深谷 猛，今村 俊幸

理化学研究所 計算科学研究機構，JST CREST

１．はじめに

ペタスケール・ポストペタスケールの大規模並列計算機を用いたシミュレーションを効率的に行うためには，高性能な数値計算ライブラリが必須となる。しかし，これまで幅広く利用されてきた行列計算用のライブラリScaLAPACK[1]は,90年代の並列計算機を想定して開発されたものであり，近年の並列計算機の特性に合っていない部分が多い。そのため，ScaLAPACKでは，もはや高い性能が期待できないということが共通認識となっており，新たな並列行列計算ライブラリの開発が望まれている。

密行列の固有値計算は，科学技術計算における重要な行列計算の一つであり，特に，実対称密行列の固有値計算はその代表例である。そのため，海外において、ELPA[2,3]やDPLASMA[4,5]といった，新たな固有値ソルバの開発が行われている。一方，国内でも，現在，我々のグループを中心に，EigenExa[6,7]と呼ばれる固有値ソルバの開発が進んでいる。

EigenExaの開発は現在も進行中ではあるが，同時に，最新版がオープンソースソフトウェアとして公開されており[6]，アプリケーション分野の研究者などに利用され始めている。また，京コンピュータを中心に行った性能評価により，その性能が確認されている[8]（併せて行ったポスター発表の一部については[6]を参照）。しかしながら，EigenExaのプログラムの計算時間の詳しい内訳や，実行条件・内部パラメータによる性能の変化，といった部分については，計算機システムの利用機会が限られているため,これまであまり評価ができていなかった。また，EigenExaでは，これまでと同様に行列を三重対角化して計算を行うルーチン（eigen_s）と行列を五重対角化して計算を行うルーチン（eigen_sx）の二つが提供されているが，両者の比較も十分に行われていなかった。

そこで，今回，「大規模HPCチャレンジ」において，FX10の4800ノードを利用して，上述の二種類のルーチンの比較を主眼においた性能評価を行った。本稿では，この二種類のルーチンの主な違いを紹介し，実際に，得られた性能評価結果の一部を示してこれを確認する。なお，今回のHPCチャレンジでは，本稿で紹介するデータ以外にも様々なデータを得ることができたが，（ライブラリの開発者には有益であるが）固有値計算にあまりなじみのない読者の方にとっては非常に細かいものであるため，本稿では，上述の二種類のルーチンの違いが分かりやすいデータの紹介にとどめる。

２．固有値ソルバEigenExaの概要

本節では，密行列向け固有値ソルバEigenExaについて簡単に説明する。なお，密行列向け固有値計算アルゴリズムの一般的な内容は，[9,10,11]などを参照されたい。また，EigenExaの前段階（Eigen-K）の開発などについては[12,13]などを参照されたい。

現状の EigenExa は,実対称密行列の全ての固有値と固有ベクトルを求める場合を主に想定して開発が進められている。また，京コンピュータやポストペタスケールの計算機システムを想定しており，数千から数万ＣＰＵといった並列数において，高い性能を実現することを目指している。

EigenExa には,現在,eigen_s と eigen_sx という二つの固有値計算ルーチンが含まれている。eigen_s は，ScaLAPACK などでも採用されている，

(1) ハウスホルダー変換による行列の三重対角化

(2) 分割統治法による三重対角行列の固有値と固有ベクトルの計算

(3) 固有ベクトルの逆変換

という一般的な計算手法に基づくルーチンである。一方，eigen_sx は，

(1) ブロック版のハウスホルダー変換による帯行列への変形

(2) 分割統治法による帯行列の固有値と固有ベクトルの計算

(3) 固有ベクトルの逆変換

という手法に基づいており，現在は，帯行列として五重対角行列を採用している。以下では，計算時間の大部分を占める，一番目と二番目のステップについて，eigen_s と eigen_sx の違いを簡単に説明する。

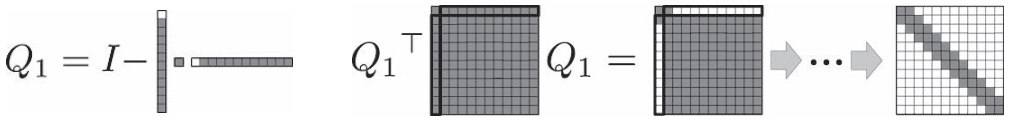
２．１．行列の三重（五重）対角化

eigen_s が採用している三重対角化の様子を図 1 に示す。ここで，ハウスホルダー変換と呼ばれる直交行列を両側から繰り返し作用させることで,密行列を三重対角行列に変形している。その際，図 1 に示したように，ハウスホルダー変換はベクトルで構成されるので，この作用は行列ベクトル積相当の演算（Level-2 BLAS）となる。従って，メモリバンド幅に律速され，近年の計算機上では，あまり高い性能が期待できない。なお，eigen_s では，Dongarra の方法[14]を採用することで，一部を行列積によって処理することが可能になっているが，依然として行列ベクトル積の部分が残っており，この部分が高性能化のボトルネックとなっている。

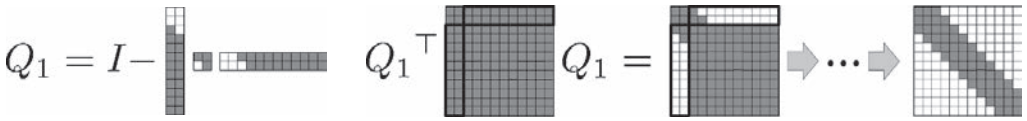
このことを踏まえて，eigen_sx では，図 2 に示すような五重対角化を採用している。五重対角化では，二本のベクトルをセットにして扱うことで，アルゴリズム中の byte/flops 値が減少し，メモリバンド幅の影響が軽減される。なお，更に帯幅を増やすことで，さらにメモリバンド幅の影響を減らすことができるが，後で述べるように，二番目のステップの計算コストが増加するというトレードオフが存在するため，両者のコストを考慮して，最適な帯幅を採用することが必要となる。

また，分散環境での並列計算では，計算対象の行列データなどは，各プロセッサに分散して保持されており，必要に応じてプロセッサ間の通信（主に MPI による）が必要となる。EigenExa でも同様で，行列データは二次元サイクリック･サイクリック分散を採用している。そのため，図1や2中の直交変換を構成するベクトルや計算対象の行列は,各プロセッサに分散しており，三重（五重）対角化の計算の中で通信をする必要が生じる。図 1 や 2 から分かるように，三重（五重）対角化の処理は，一列（二列）ずつ行われ，この中でブロードキャストやリダクションの通信が行われる。この点において，eigen_sx は eigen_s に比べて，処理の単位が二列になったことにより，通信回数が減少し，通信のセットアップコスト（レイテンシ）の削減が期待できる。ただし，一回の通信のデータ量は eigen_sx の方が eigen_s より多くなるため，通信さ

れるデータの総量は eigen_s も eigen_sx も同程度となり，こちらの部分でのコスト削減は期待できない。



第 1 図：三重対角化（eigen_s）の概略図。

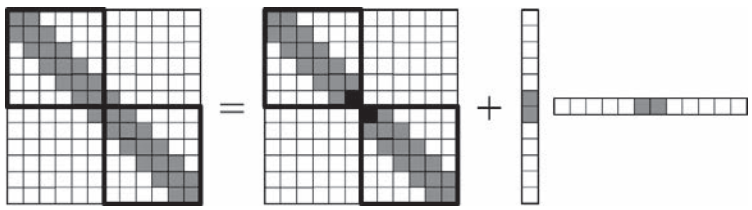


第 2 図：五重対角化（eigen_sx）の概略図。

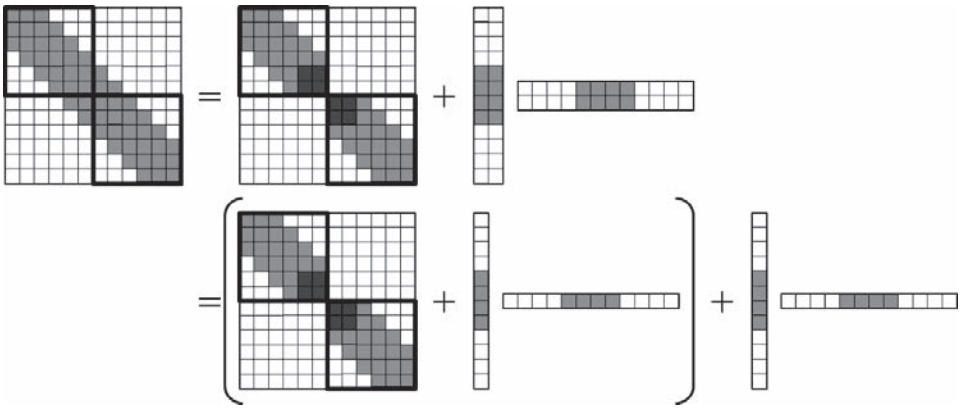
２．２．分割統治法による固有値・固有ベクトルの計算

eigen_s では，Cuppen によって提案された分割統治法[15]を用いて，三重対角行列の固有値と固有ベクトルを計算している。プログラムとしては，ScaLAPACK のソースコードを必要に応じて修正した形となっている。この手法では，図 3 に示すように，三重対角行列を，二つのサイズが半分の三重対角行列から構成されるブロック対角行列とランク 1 の摂動の和で表して，小さいサイズの固有値と固有ベクトルから，元の行列の固有値と固有ベクトルを計算する（統治フェーズ)。なお，この分割は再帰的に行うことが可能なので，実際のプログラムでは，一定のサイズになるまで分割を繰り返し，あるサイズの行列の固有値と固有ベクトルを QR 法などで計算し，その結果から，最終的に最初の三重対角行列の固有値と固有ベクトルを計算する。

一方，eigen_sx では，帯行列向けに拡張された分割統治法[16]を利用する。まず，図 4 に示すように，五重対角行列をランク 2 の摂動を用いて分割する。次に，ランク 2 の摂動を二つのランク 1 の摂動とみなし，得られた固有値と固有ベクトルを二回更新する。従って，統治フェーズの処理が，三重対角行列の場合の処理を二回行うことになるため，計算コストが増加する。なお，プログラムは，三重対角行列に対する分割統治法の中で利用しているルーチンが再利用できるため，eigen_sx では，ScaLAPACK の分割統治法のルーチンを修正・組み合わせる形で実装がなされている。



第 3 図：三重対角行列に対する分割統治法における，ランク 1 の摂動による行列の分割。



第 4 図: 五重対角行列に対する分割統治法における，ランク 2 の摂動による行列の分割。

３．性能評価

本節では，FX10 スーパーコンピュータシステム（Oakleaf-FX）の 4800 ノードを使用して行った EigenExa の性能評価の結果を報告する。

３．１．評価条件等

使用したプログラムは，開発中の EigenExa のバージョン 2.0 に，各部の時間計測用のタイマーを独自に挿入したものである。ただし，最初と最後以外にバリアを挿入することはしていない。コンパイラとそのオプションは公開されている EigenExa の makefile をそのまま使用した（mpifrtpx -Kfast -Kopenmp -Kocl -Ksimd -KXFILL）。また，EigenExa の内部で使用する BLAS や比較用の ScaLAPACK はともに富士通が提供しているもの(-SSL2BLAMP でリンク)を使用した。評価の際に用いた行列は，$A_{ij} = N + 1 - \max(i,j)$で構成される実対称密行列であり，これの全固有値と固有ベクトルを計算した。行列サイズ（N）は，10000，50000，100000，230000 の四種類を試した。ノードの形状は 60x80 の二次元形状で指定し，1 ノード当たり 1MPI プロセス，1 プロセス当たり 16 スレッドを指定した。一方，EigenExa 内のプロセスグリッドは，行方向 60x 行方向 80 として，ノードの形状と対応するようにした。

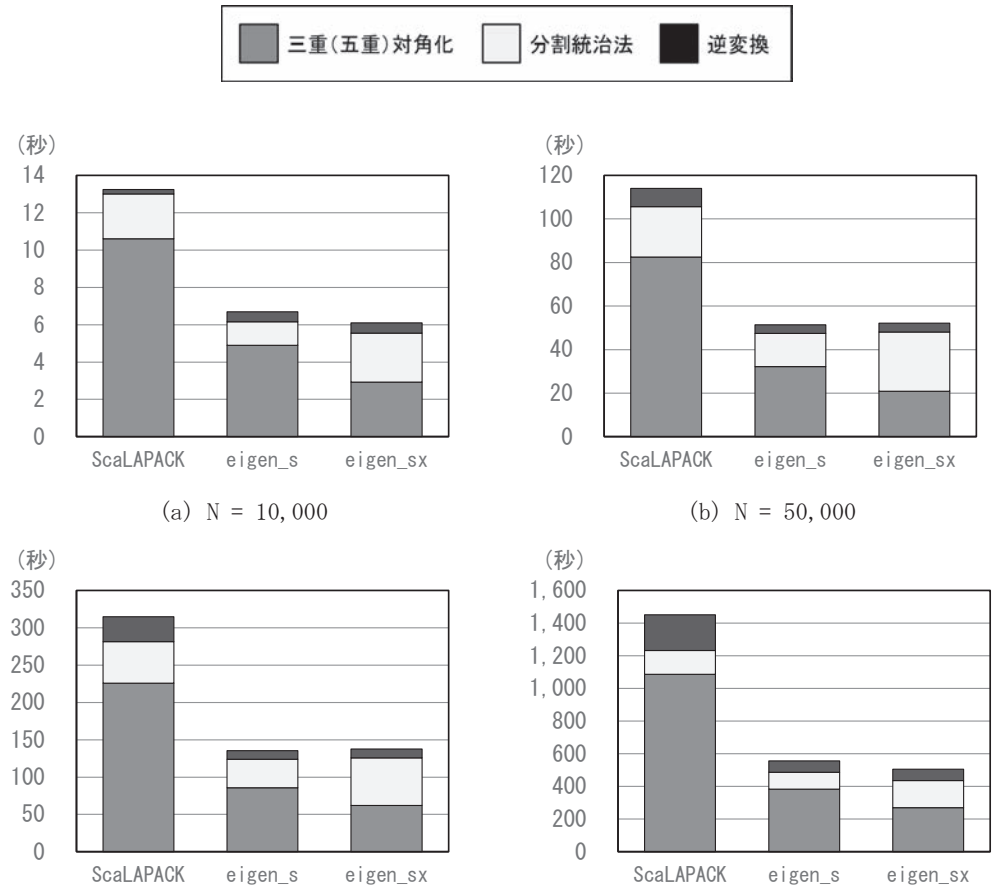
３．２．結果

最初に，四種類のサイズの行列に対する，EigenExa の中にある二種類のルーチン（eigen_s と eigen_sx）と ScaLAPACK のルーチン（PDSYEVD に相当）の計算時間を図 5 に示す。グラフより，EigenExa の二つのルーチンはどちらも，ScaLAPACK のルーチンよりも二倍から三倍程度高速であることが確認できる。

まず，ScaLAPACK と eigen_s を比較すると，三重対角化の部分で大きな差があることが分かる。両者の採用しているアルゴリズムは基本的に同じであるが，ScaLAPACK が最近の並列計算機の特徴を想定して開発されていないのに対して，eigen_s は十分にそれを考慮して開発や実装がなされているため，このように状況になっていると言える。逆に，分割統治法の部分は，現状の eigen_s も ScaLAPACK の内部ルーチンを数多く利用する実装になっているため，両者の

間にあまり差が生じていない。また，逆変換の部分は，計算の依存性が低く，計算自体も行列積を効率的に利用できるため，全体の計算時間に占める割合が非常に小さい。

次に，eigen_s と eigen_sx を比較すると，前節で述べたように，eigen_s の三重対角化の計算時間よりも eigen_sx の五重対角化の計算時間の方が短くなっており，反面，分割統治法の計算時間については，逆の状況となっていることが確認できる。今回の条件では，全体の計算時間としては，eigen_sx の方が若干短いか同程度，という結果になっている。

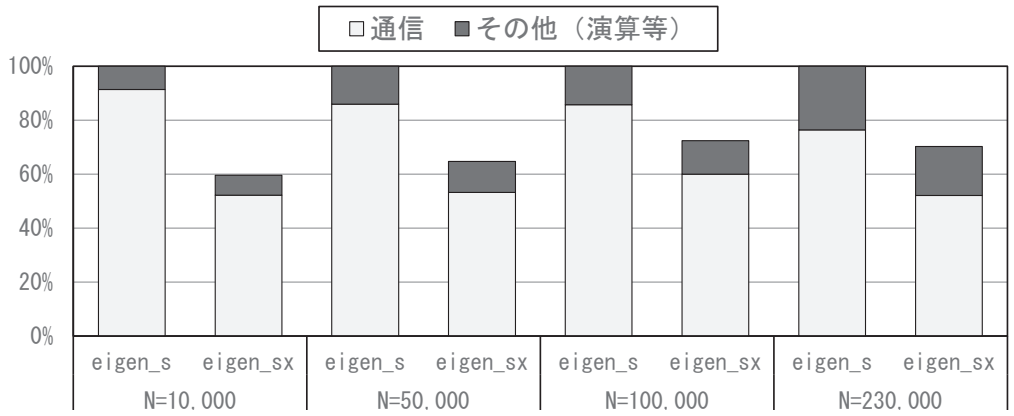


第 5 図：4800 ノードを用いた固有値計算の実行時間とその内訳。

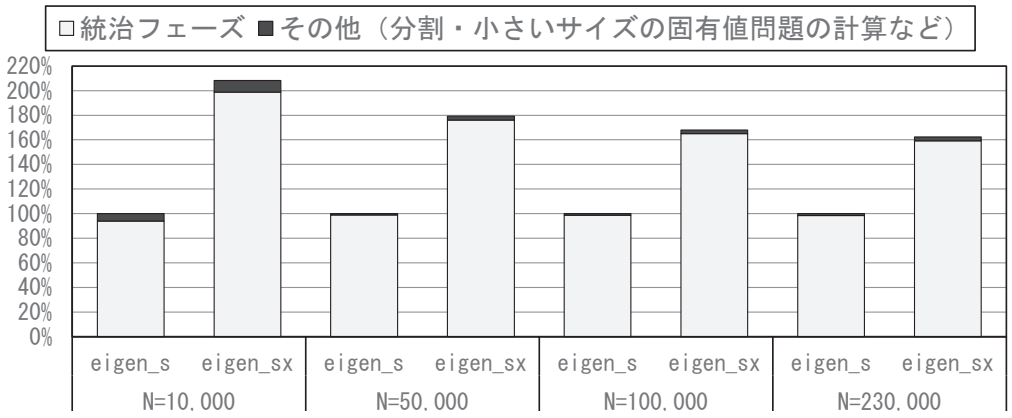
この部分をもう少し詳しく評価するために，図 6 に両者の三重（五重）対角化の計算時間に対して通信時間の占める割合を示す。このグラフが示すように，今回の並列数と問題サイズにおいては，三重（五重）対角化の計算時間の大半は通信に費やされていることになる。行列サイズが大きくなるにつれて通信時間の占める割合が小さくなっていることが確認できるが，これは，演算量がサイズの三乗で増加するのに対して，通信回数がサイズの一乗，通信データ量がサイズの二乗で増加からであると考えられる。また，前節で述べたように，三重対角化を五重対角化にすることで，メモリバンド幅の影響が軽減され，演算の実効性能の向上が期待できるが，eigen_s と eigen_sx のその他（演算等）の部分を比べると，後者の方が小さくなってい

ることが確認できる。一方，通信時間についても，前節で述べたように，通信回数が eigen_sx の方が少ないことから，eigen_s よりも eigen_sx における通信時間の方が少なくなることが予想されており，実際にグラフの結果も eigen_sx の通信時間の方が短くなっている。

次に，分割統治法の計算時間に対して，小さいサイズの行列の固有値・固有ベクトルから大きなサイズのそれを計算する，統治フェーズの計算時間が占める割合を図 7 に示す。このグラフから，分割された最も小さいサイズの問題を計算する時間の占める割合は非常に小さく，統治フェーズの計算時間が重要であることが分かる。前節で述べたように，五重対角行列に対する分割統治法では，三重対角行列に対する分割統治法の統治処理を二回繰り返すことが必要となる。そのため，グラフから分かるように，eigen_sx の統治フェーズの計算時間は，eigen_s のそれより長くなっている。興味深いのは，行列サイズが小さいときは，eigen_sx の統治フェーズの計算時間は eigen_s の約二倍程度となっているが，行列サイズが大きくなるにつれて，比率が小さくなっている点である。この理由については，現在のところ不明であり，更なる調査が必要である。



第 6 図：三重（五重）対角化の計算時間に対して通信時間が占める割合。



第 7 図：分割統治法の計算時間に対して統治フェーズの計算時間が占める割合。

4．おわりに

本稿では，FX10 の 4800 ノードを用いて行った，固有値計算ソルバ EigenExa の性能評価の結果の一部を報告した。EigenExa は現在も開発が進められているが，既に，ScaLAPACK のプログラムより高性能になっていることが確認できた。一方で，現在のプログラムの計算時間の大半が通信に費やされているという事実も明確に示されており，今後の開発でこの部分をどれだけ削減できるかが課題となっている。

また，今回の評価結果により，三重対角化を五重対角化に変更することで，演算の実効性能の向上による演算時間の削減と，通信回数の削減による通信時間の削減の両方の効果を確認することができた。今後の計算機システムでは，さらに，メモリバンド幅が演算性能に対して相対的に低下し，通信コスト（特に通信のセットアップコスト）の増加することが予想されるため，五重対角化（さらには七重対角化）とすることの効果がより大きくなることが期待できる。

この効果を生かすためには，分割統治法のルーチンの性能改善が必須となる。現状では，ScaLAPACK のコードをベースとしているために，そもそも，性能評価や性能のボトルネックの調査が十分に行えていない(提供されているライブラリの中の性能測定が困難なため)。今後は，できる限りの性能調査を行うとともに，必要に応じて，ScaLAPACK のコードを利用するのではなく，独自の実装も視野に入れて開発を進めることを検討している。

加えて，我々が別途行っている，性能モデルに関する研究[17,18]の結果と，今回の性能評価で得られたデータを比較して，モデルの検証と改善を行い，その結果を性能チューニング等につなげることも今後の課題となっている。

最後になるが，今回の「大規模 HPC チャレンジ」では，本稿で紹介した結果以外にも様々なデータを得ることができた。今後，得られたデータを踏まえて開発を進め，より高性能な固有値ソルバの開発を目指したいと思う。高性能な固有値ソルバが完成した際には，改めて，本誌等でご報告できればと思う。

謝辞

今回の「大規模 HPC チャレンジ」を通して大変お世話になりました，東京大学情報基盤センターの関係者の皆様に深く感謝申し上げます。また，EigenExa は JST CREST プロジェクト「ポストペタスケールに対応した階層モデルによる超並列固有値解析エンジンの開発」(領域名：ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出）の援助の下で開発中であり，プロジェクトメンバーの皆様にも感謝いたします。なお，EigenExa の開発・評価の一部は理化学研究所のスーパーコンピュータ「京」を用いて行われています(課題番号：hp120170，および ra000005)。

参 考 文 献

[1] ScaLAPACK, http://www.netlib.org/scalapack/

[2] ELPA, http://elpa.rzg.mpg.de/

[3] T. Auckenthaler, et al., Developing algorithms and software for the parallel solution of the symmetric eigenvalue problem, Journal of Computational Science, Vol.2, pp.272-278, 2011.

[4] DPLASMA, http://icl.cs.utk.edu/dplasma/

[5] G. Bosilca, et al., Distributed Dense Numerical Linear Algebra Algorithms on Massively Parallel Architectures: DPLASMA, University of Tennessee Computer Science Technical Report, UT-CS-10-660, 2010.
[6] EigenExa, http://www.aics.riken.jp/labs/lpnctrt/EigenExa.html
[7] 今村俊幸 他, Eigen-Exa: ポストペタ スケール環境での 密行列固有値ソルバー開発, 第17回計算工学会講演会, 京都, 2012.
[8] T. Imamura, et al., CREST: Dense Eigen-Engine Groups", Poster presentation, International Workshop on Eigenvalue Problems: Algorithms, Software and Applications, in Petascale Computing (EPASA2014), Tsukuba, 2014.
[9] 杉原正顯 他, 線形計算の数理, 2009.
[10]J. Demmel, Applied Numerical Linear Algebra, 1997.
[11]山本有作, 密行列固有値解法の最近の発展(I), 日本応用数理学会論文誌, Vol.15, No.2, pp.181-208, 2005.
[12]今村俊幸, T2Kスパコンにおける固有値ソルバの開発, 東京大学スーパーコンピューティングニュース, Vol.11, No.6, pp.12-32, 2009.
[13]T. Imamura, et al., Development of a High-Performance Eigensolver on a Peta-Scale Next-Generation Supercomputer System, Progress in Nuclear Science and Technology, Vol.2, pp.643-650, 2011.
[14]S. Hammarling, et al., Block Reduction of Matrices to Condensed Forms for Eigenvalue Computations, J. Comput. Appl. Math, Vol.27, pp.215-227, 1987.
[15]J. Cuppen, A divide and conquer method for the symmetric tridiagonal eigenproblem, Numer. Math, Vol.36, pp.177-195, 1981.
[16]P. Arbenz, Divide and conquer algorithms for the bandsymmetric eigenvalue problem, Parallel Computing, Vol.18, No.10, pp.1105-1128, 1992.
[17]深谷猛, 超並列環境向け固有値計算プログラムの性能予測モデルの開発, 東京大学スーパーコンピューティングニュース, Vol.15 No.6, pp.33-43, 2013.
[18]深谷猛, 超並列環境向け固有値計算プログラムの性能予測モデルの開発（続）, 東京大学スーパーコンピューティングニュース, Vol.16 No.1, pp.21-28, 2014.