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）

深谷　猛

神戸大学大学院システム情報学研究科（現　理化学研究所計算科学研究機構），JST CREST

１．はじめに

「京」コンピュータなどのペタスケールやその次のポストペタスケールの計算機を用いた科学技術計算において，行列計算をはじめとする数値計算プログラムは重要な基盤技術の一つであると認識されている。そのため，ScaLAPACK などの従来のライブラリに代わる，新しい高性能なライブラリの開発が様々な場所で進められている。

著者の所属するプロジェクトでも，ペタ・ポストペタスケールの大規模並列計算機を想定した，高性能な密行列向け固有値計算プログラムの開発が進められている[1]。特に，著者はこの研究の一環である，固有値計算プログラムの性能予測モデルの構築に関する研究に携わっている。本プロジェクトでは，まだ次世代のスパコンのアーキテクチャの検討段階，あるいはシステム全体が完成していない段階で，次世代向けの固有値計算プログラムを開発するため，プログラムの性能を予測するモデルの活用を考えている。また，今後のスパコン向けのプログラム開発やチューニングを考えてみても，スパコンの使用機会は限られているため，モデルを使って開発やチューニングを効率化することは必要だといえる。そのため，今回，モデルの開発に主眼をおいた研究を行っている。

前回の記事[2]では，固有値計算プログラム EigenK[3]の主要部の一つである，実対称密行列の三重対角化のプログラムの並列実行時間を予測するモデルの構築について，研究背景やプログラムの概要を説明した上で，我々の目的において求められるモデルの性質を示して，実際にモデルの構築を行った。そして，「京」の上での三重対角化の実行時間の予測例を紹介した。

本稿では，最初に前回の記事で報告したモデルの概要を振り返る。その後，モデルの検証に関する検討を行い，実際に東京大学の FX10 の上で行った，モデルの検証結果について報告する。

２．前回の記事の概要

本節では，前回の記事で述べた内容の中で，特に構築したモデルを中心に概要を述べる。前回の記事を読まれた読者の方は飛ばしてもらって構わない。また，詳細について興味を持たれた読者の方は，時間が許すのであれば前回の記事を読んでいただけると幸いである。

２．１．モデル構築の方針

本研究でモデルを構築するプログラムは，現在開発中の固有値計算ライブラリ EigenExa の前段階である EigenK[3]の中にある，実対称行列を三重対角化するプログラムである。行列の三重対角化は実対称行列の固有値計算の計算時間の大半を占めているため，この部分の計算時間を予測するモデルを開発することは，固有値計算プログラムの開発に有用であると考えている。なお，三重対角化の数理的な部分の説明は数値計算の教科書や前回の記事[2]に委ねる。また，

プログラムの実装面としては，二次元サイクリック分散で，MPI によって分散環境向けに並列化されており，プロセスは二次元グリッド上に配置されている。

この三重対角化のプログラムを題材として，今回，プログラムの開発者向けの性能モデルを開発する。また，今回は，

- プログラム内部はブラックボックスではなく，演算回数や通信回数などは既知である。
- ペタ・ポストペタスケールのマシンを想定するため，実システム上で長時間，あるいはシステムの大部分を使用したベンチマーク等の実測は困難である。
- システムのごく一部を使用した短時間のベンチマーク等は可能である。
- プログラムの開発やチューニングに活用するため，全体の実行時間だけでなく，内部の主要な処理ごとの時間も精度良く予測することを目指す。

ということを念頭においてモデルの開発を行う。

２．２．構築したモデルの概要

前述の点を考慮した上で構築したモデルの概要を紹介する。なお，以下では，問題サイズが N，並列数が P（ただし平方数）とする。三重対角化の実行時間を$T(N,P)$とすると，今回のモデルでは，

$$T(N,P)=\sum_{j\in J}T_{\mathrm{f\_total}}^{(j)}(N,P)+\sum_{j\in J}\sum_{i\in I_j}\left[\tilde{\alpha}^{(j,i)}\left(\sqrt{P}\right)\cdot C_{\mathrm{total}}^{(j,i)}(N)+\tilde{\beta}^{(j,i)}\left(\sqrt{P}\right)\cdot W_{\mathrm{total}}^{(j,i)}(N,P)\right]$$

という式で表される。モデル式中の各項の意味などは，

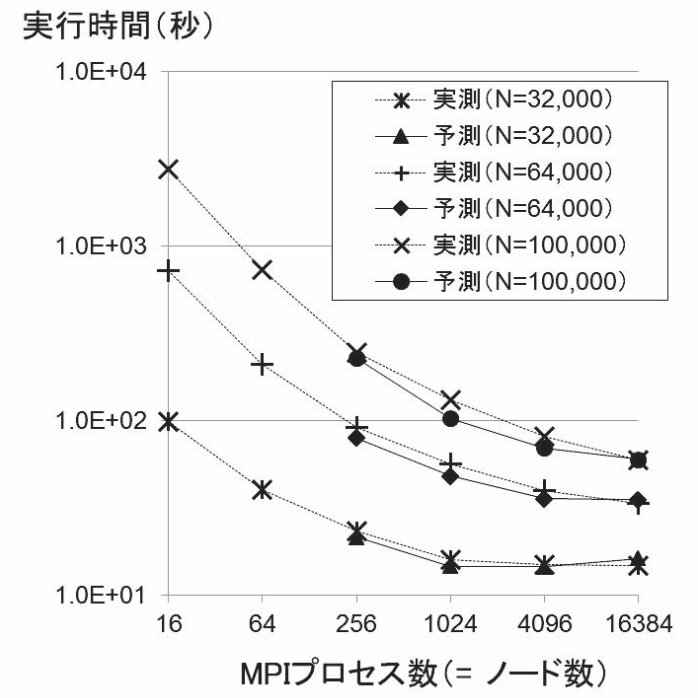
- J: 処理の集合（つまり，J={ハウスホルダー変換の生成，行列ベクトル積，...}）
- I_j：処理 J 中の MPI 通信の集合（つまり，I_j ={MPI_Allreduce, MPI_Bcast, …}）
- $T_{\mathrm{f\_total}}^{(j)}(N,P)$ ：処理 j の演算時間の合計で，もとのプログラムから通信部分を除いて作成するベンチマークプログラムを 1 ノード上で実行することで取得する。
- $\tilde{\alpha}^{(j,i)}\left(\sqrt{P}\right)$：処理 j 中のある MPI 通信 i における通信のセットアップコストで，$\sqrt{P}$程度のノード数で MPI 通信のみを測定するベンチマークプログラムを実行して取得する。
- $C_{\mathrm{total}}^{(j,i)}(N)$ ：処理 j 中のある MPI 通信 i を行う回数で既知。
- $\tilde{\beta}^{(j,i)}\left(\sqrt{P}\right)$:処理 j 中のある MPI 通信 i における実効バンド幅の逆数で，取得方法は$\tilde{\alpha}^{(j,i)}\left(\sqrt{P}\right)$に同じ。
- $W_{\mathrm{total}}^{(j,i)}(N,P)$：処理 j 中のある MPI 通信 i における通信データ量の総量で既知。

となっている。また，　このモデルを使って実際に実行時間を予測する際のコストは，

- $T_{\mathrm{f\_total}}^{(j)}(N,P)$ を取得するためのプログラムの実行時間は，実際に並列数が P で，問題サイズが N の計算を並列実行した時間から通信時間相当を差し引いたものとなり，N がそれなりに大きい場合は，それなりの時間が必要となる。ただし，必要なノード数は 1 であるため，現実的にはあまり問題にならないと思われる。
- $\tilde{\alpha}^{(j,i)}\left(\sqrt{P}\right)$ と$\tilde{\beta}^{(j,i)}\left(\sqrt{P}\right)$を取得するためには，$\sqrt{P}$程度のノード数が必要となるが，$N$とは関係していないため，各 Pに対して一度測定しておけば，その結果を再利用することができる。また，処理間で MPI 通信の種類とパターン（どちらの軸方向か）が共通している部分も多いため，実際には，MPI_Allreduce と MPI_Bcast の二種類についてのみの測定で済むことが多い。

となっており，システムのごく一部を使用した短時間のベンチマークのみで取得可能である。

実際にこのモデルを使って,「京」コンピュータにおける三重対角化の実行時間を予測した結果を実測した結果を併せて図１に示す（前回の記事の結果の再掲)。このグラフより，多少の誤差はあるものの，合計時間としては，それなりの精度で予測できていることが確認できる。



第１図:「京」おける EigenK の三重対角化の実行時間の予測結果（前回記事の結果の再掲)。
「京」コンピュータにおける EigenK の三重対角化部分の実行時間を，構築したモデルにより予測した結果と実際にプログラムを並列実行して測定した結果を比較したグラフであり，合計時間のみを見ると，モデルによる予測結果が良好であることが確認できる。なお，実行条件等の詳しい部分は前回の記事を参照されたい。

３．モデルの検証

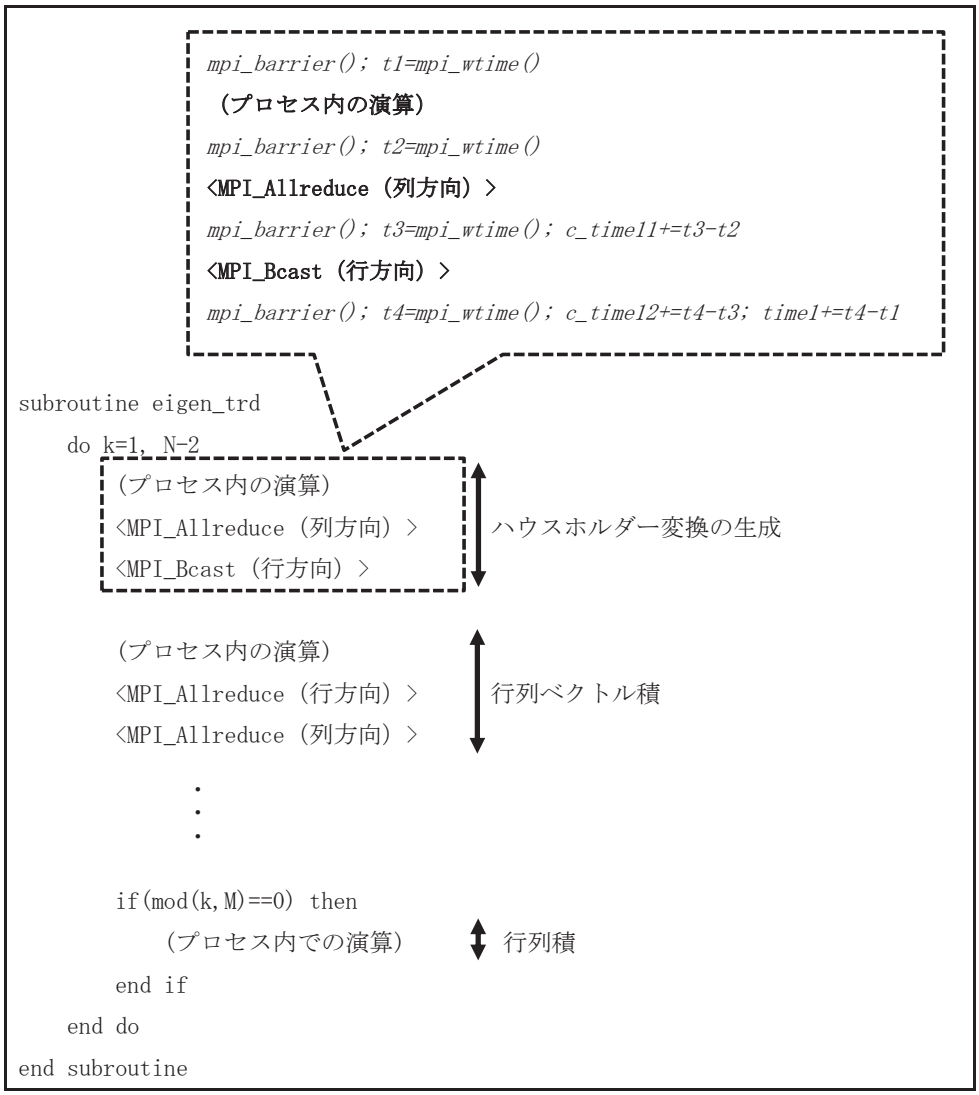
本節では，構築したモデルによる予測結果の検証方法を考える。そもそも，今回のモデルは固有値計算プログラムの開発者が使用することを想定したものである。そのため，プログラム全体の実行時間だけでなく，プログラム内部の処理ごと，さらには処理内の演算時間と通信時間それぞれを予測することを目指している。そして，実際に構築したモデルは，各部分のモデルを構築し,各部分の予測時間を積み上げることで全体の時間を予測する階層型になっている。そのため，合計時間だけでなく，各部分の時間の予測精度を検証することが不可欠である。

さて，三重対角化の計算中の処理ごとの演算時間や通信時間の予測精度を検証しようとすると，簡単に行えない。というのは，予測値を評価するために必要となる「真の値」をどのように測定するか,が問題となる。三重対角化のプログラムの構成は図２に示した形になっている。したがって，処理や内部の通信ごとの時間を測定するためには，図２に示したようにタイマーを挿入するのが自然だと思われる。また，MPI_Bcast などの通信はプロセスによって処理が戻るタイミングが異なることがあるので，プロセス間の足並みを同じにしようとバリアを挿入すると，図２のようになってしまう。このようにバリアを挿入すると，最内のループ内でバリア

を何回も呼ぶことになるため，バリアのコストが無視できなくなることが予想される。

一方，バリアを挿入しない場合，各プロセスで処理のタイミングがずれるため，全プロセスで測定した時間データを全て取得して，何らかの処理を加えることが必要となる。ただし，プロセス数が増えると膨大なデータが得られるため，この処理が問題となることが予想される。

このように，簡単に「真の値」を測定することは難しい状況になっている。今回は，比較的簡単に行える，最初に述べたバリアを細かく入れてプロセス間の処理のタイミングを合わせて時間を測定する方法を試みる。ただし，時間測定の際のバリアの影響を検証して，必要に応じて得られた実行時間を補正した上で，モデルによる予測結果の検証に用いることとする。



第２図：EigenK のプログラム中の処理の流れと処理ごとの時間測定方法。

図のように，行列サイズ相当のループ（ここではｋのループ）があり，その中で行列ベクトル積などの各処理を行う形になっている。また，各処理の内部で必要に応じて MPI 通信を行っている。そのため，処理ごと（さらにその内部の通信ごと）の時間を測定するためには，上図のようにタイマーを挿入する必要がある。また，プロセス間での処理時間のばらつきを抑えるためにバリアを入れることを検討する必要がある。

４．モデルの検証の具体例

本節では，実際にモデルの検証を行った例を報告する。具体的には，東京大学の FX10 の 64 ノード（形状は 8x8）を用いた計算に対するモデルの予測結果を検証する。まず，三重対角化の実測時間とモデルに基づく予測時間を表１に示す。予測値の実測値に対する相対誤差も併せて載せたが，これを見る限りでは，三重対角化全体の実行時間については，今回のケースでは比較的良い精度で予測できていると判断できる。以下では，この具体例について，三重対角化の処理ごと，さらに演算時間と通信時間を区別して，それぞれの予測精度について検証を行う。

表１： FX10 の 64 ノードでの三重対角化の実測時間と予測時間。

行列サイズ	実測（秒）	予測（秒）	相対誤差（%）
10000	4.29	4.63	+7.8
30000	31.4	31.2	−0.9
50000	90.7	88.8	−2.1

４．１．比較のための実測データの扱い

予測結果を検証するためには，比較用として，三重対角化の処理ごとに演算時間と通信時間を測定する必要がある。しかし，前節で述べたように，これらの時間を測定するために必要となるタイマーやバリアに伴うオーバーヘッドも予想される。そこで，まず，タイマーとバリアの設定方法の違いによる，三重対角化全体の実行時間の違いを表２に示す。この表から分かるように，特にバリアに伴うコストは無視できるものではなく，予測結果との比較をする場合には，バリアのコストを考慮した上で比較を行う必要があると言える。

そこで，タイマーとバリアのコストを測定して，そのコストの補正をする。今回の環境では，タイマー1 回のコストが約 1.5μ秒，バリア１回のコストが約 4.1μ秒であった。プログラム内部で測定のためにタイマーとバリアが呼ばれている回数は既知なので（ただし，行列サイズには依存するが），表２の計算時間からタイマーとバリアのコストに相当する時間を差し引いた結果を表３に示す。結果を見ると，まだ，タイマーやバリアを入れたことによる計算時間の増加が確認できるが（この原因はまだ調査中），少なくとも，表２の結果をそのまま予測値と比較するよりは，表３のようにタイマーやバリアのコストを補正した上で，モデルの検証用として予測値との比較に用いる方が妥当であると判断できる。よって，以下では，実測値にタイマーやバリアのコストを補正した値を用いて，モデルによる予測値の検証を行うことにする。

表２： 測定方法の違いによる全体の計算時間（秒）。

括弧内の数字は最初と最後のみにタイマーとバリアを設定した場合の計算時間に対する比。

タイマー バリア	最初と最後 最初と最後	処理・通信ごと 最初と最後	処理・通信ごと 処理ごと	処理・通信ごと 処理・通信ごと
10000	4.29	4.52（1.05）	5.06（1.18）	5.85（1.36）
30000	31.4	32.9（1.05）	35.0（1.12）	38.0（1.21）
50000	90.7	93.7（1.03）	97.3（1.07）	104（1.14）

表３：タイマーとバリアのコスト補正後の測定方法の違いによる全体の計算時間（秒）。
括弧内の数字は最初と最後のみにタイマーとバリアを設定した場合の計算時間に対する比。

タイマー	最初と最後	処理・通信ごと	処理・通信ごと	処理・通信ごと
バリア	最初と最後	最初と最後	処理ごと	処理・通信ごと
10000	4.29	4.13（0.96）	4.27（0.99）	4.40（1.03）
30000	31.4	31.8（1.01）	32.7（1.04）	33.7（1.07）
50000	90.7	91.8（1.01）	93.3（1.03）	96.5（1.06）

４．２．処理ごとの演算時間・計算時間の検証

次に，三重対角化の計算の処理ごとに，演算時間と通信時間を区別して，予測した値を検証する。なお，比較する実測値は，前節で述べた測定方法の中で，処理と通信ごとにバリア（とタイマー）を設定して測定した値からバリアとタイマーのコスト相当の時間を差し引いたものを用いる。また，演算時間の実測値は処理の実行時間から処理内の通信時間を引いた値としている。

今回は，以下に挙げる，三重対角化の計算の中の主要な４つの処理について検証する。

① ハウスホルダー変換の生成：ベクトルのノルムの計算が中心
② 行列ベクトル積
③ ベクトル v の計算：ベクトル（もしくはブロックベクトル）とベクトルの内積が中心
④ A の更新（行列積）

なお，これらの処理の三重対角化の中での位置づけなどは先の記事を参照されたい。

各行列サイズについて，上記の４種類の処理について，演算時間と通信時間，両者の合計時間について実測値と予測値を比較した結果を表４に示す。まず，演算時間に関しては，

- 行列サイズに関わらず，全ての処理で予測値が小さい。
- 行列サイズに関わらず，①と③の予測精度が悪く，一方で②と④の精度は良い。
- 演算時間の主要部は②であり，この部分の予測精度が良いため，演算時間全体の予測精度としては，比較的許容できるものになっている。（Nが大きくなるにつれて顕著である）

ということが読み取れる。一方，通信時間については，

- ①と②については，今回の結果のみでは顕著な特徴を見出すことができない。
- ③については，行列サイズが大きくなるほど，マイナス側に予測の精度が低下している。

ということが読み取れる。最後に，合計時間については，

- 計算時間の主要部である②の予測精度が比較的良いため，三重対角化全体の計算時間の予測も比較的良いものになっている。
- ①や③の予測精度はあまり良いとは言えない。
- ④の予測精度は良いが，三重対角化全体の計算時間に占める割合が少ないため，全体の計算時間の予測精度にはあまり寄与していない。

といったことが読み取れる。

表４： 処理ごとの演算・通信時間の実測値と予測値の比較

処理は①ハウスホルダー変換の生成，②行列ベクトル積，③ベクトル v の計算，④A の更新（行列積）。また誤差は相対誤差。

(a)N=10000 の場合

処理	演算			通信			合計		
	実測（秒）	予測（秒）	誤差（%）	実測（秒）	予測（秒）	誤差（%）	実測（秒）	予測（秒）	誤差（%）
①	0.05	0.02	-55.2	1.02	1.23	+21.1	1.07	1.25	+17.3
②	0.51	0.48	-5.9	0.94	1.00	+6.8	1.45	1.49	+2.3
③	0.47	0.42	-11.2	0.74	0.90	+22.0	1.21	1.32	+9.1
④	0.19	0.17	-8.5	-	-	-	0.19	0.17	-8.5
合計	1.22	1.09	-10.5	2.69	3.13	+16.3	3.91	4.23	+8.0

(b)N=30000 の場合

処理	演算			通信			合計		
	実測（秒）	予測（秒）	誤差（%）	実測（秒）	予測（秒）	誤差（%）	実測（秒）	予測（秒）	誤差（%）
①	0.22	0.08	-64.0	4.87	5.36	+10.2	5.09	5.44	+7.0
②	10.9	10.6	-2.5	6.19	5.57	-10.0	17.1	16.2	-5.2
③	2.18	1.69	-22.8	4.68	3.54	-24.3	6.86	5.23	-23.8
④	3.13	2.98	-4.6	-	-	-	3.13	2.98	-4.6
合計	16.4	15.4	-6.4	15.7	14.5	-8.0	32.1	29.8	-7.2

(c)N=50000 の場合

処理	演算			通信			合計		
	実測（秒）	予測（秒）	誤差（%）	実測（秒）	予測（秒）	誤差（%）	実測（秒）	予測（秒）	誤差（%）
①	0.41	0.14	-66.9	9.34	10.1	+8.3	9.75	10.3	+5.1
②	45.0	43.4	-3.4	13.2	11.8	-10.4	58.2	55.2	-5.0
③	4.19	3.04	-27.5	9.84	6.49	-34.0	14.0	9.53	-32.1
④	11.4	11.2	-2.4	-	-	-	11.4	11.2	-2.4
合計	61.0	57.8	-5.3	32.4	28.4	-12.2	93.4	86.2	-7.7

５．おわりに

本稿では，主に前回の記事で述べた，我々が構築したモデルに関して，目的の一つであった三重対角化の計算の各部分の演算時間や通信時間がどの程度の精度で予測できているかについて，FX10 の上で行った検証の結果を報告した。今回の検証例では，演算量の多い，行列ベクトル積や行列積の部分の演算時間については,比較的良い精度で予測できていることが確認でき，一方で，演算量が相対的に少ないベクトルのノルムや内積が中心の処理の演算時間の予測精度は低かった。また，通信時間については，相対誤差が±10%程度であることが多かった。

この結果を踏まえて，数千から数万という超並列時の実行時間の予測を考えてみると，1 プロセス当たりの演算量は今回の例よりも相対的に小さくなることが予想されるため，演算時間の誤差は拡大する可能性が高いと思われる。一方，通信時間については，並列数が増えた場合の挙動を今後，検証する必要があると言える。また，今回の例からも明らかなように，プラスの誤差とマイナスの誤差が打ち消しあった結果として，全体の合計時間の予測誤差が小さくなっているということが十分に起こりえるので，より超並列実行時におけるモデルの検証が今後の課題となる。

最後に，3 節で述べたように，今回，「真の」実行時間として採用した値が必ずしも適切だったとは言えない可能性がある。少なくとも，処理や通信単位でバリアをとらずに実行した場合に，各プロセスで測定された値を全て確認してみることが必要となる。また，単純に MPI のベンチマークをする場合に関しても，先行研究[4]等で注意事項等が指摘されているので，これらの結果を活用することも今後の課題といえる。

謝辞

本研究に関して日ごろから有益なご議論をさせていただいている，神戸大学（現　電気通信大学）の山本有作教授と理化学研究所計算科学研究機構の今村俊幸チームリーダーをはじめとする下記 CREST プロジェクトのメンバーの皆様に深く感謝申し上げます。また，本研究は東京大学情報基盤センターの「若手・女性利用者推薦」（H24 年度後期・H25 年度前期）に基づいており，制度の利用においてお世話になった東京大学の片桐孝洋准教授をはじめとする関係者の皆様にも感謝いたします。なお，本研究は 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）の援助を受けており，結果の一部は理化学研究所のスーパーコンピュータ「京」を利用して得られたものです（一般利用課題：hp120170）。

参　考　文　献

[1] EigenExa:http://www.aics.riken.jp/labs/lpnctrt/EigenExa.html.

[2] 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.

[3] EigenK:http://ccse.jaea.go.jp/ja/download/eigenk.html.

[4] T. Hoefler, T. Schneider and A. Lumsdaine, Accurately Measuring Overhead, Communication Time and Progression of Blocking and Nonblocking Collective Operations at Massive Scale, International Journal of Parallel, Emergent and Distributed Systems. Vol.25, pp.241-258, 2010.