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

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」コンピュータをはじめとするペタスケールの計算機やその先のポストペタスケールの計算機を使用した大規模なシミュレーションへの期待が高まっている。大規模シミュレーションを実現するためには様々な技術が必要になるが，行列計算はその一つであり，本稿で扱う固有値計算はその代表例である。

現在，ペタ・ポストペタスケールの計算機に向けた新しい固有値計算プログラムの開発が活発に行われているが，その際，開発したプログラムを計算機上で性能評価する機会が限られているという課題が生じている。そこで，固有値計算プログラムの開発の一環として，「京」コンピュータのような超並列環境における，密行列向け固有値計算プログラムの性能予測モデルの開発を目的とした研究を現在行っているので，その一部を本稿で紹介する。

２．研究背景と目的

本節では，まず，本研究の研究背景として，固有値計算プログラムの現状とペタ・ポストペタスケールの計算機を想定したプログラム開発における性能予測モデルの必要性について述べる。その後，本研究の目的と具体的な目標を述べる。

２．１．固有値計算プログラムの現状

実対称密行列の固有値計算は，量子化学計算をはじめとする様々なシミュレーションで必要とされる基本的な行列計算の一つである。そのため，「京」コンピュータのようなペタスケールの計算機やその先のポストペタスケールの計算機を用いた大規模なシミュレーションを高速に行うためには，ペタ・ポストペタスケールの計算機向けの高性能な固有値計算プログラムが必要とされている[1]。

実対称密行列の固有値計算は，

① 与えられた行列を三重対角行列に変形，

② 三重対角行列の固有値・固有ベクトルの計算，

③ 固有ベクトルの逆変換，

という 3 ステップの計算手順で行うことが一般的となっている[2]。この計算手順を分散並列環境向けに実装したプログラムとしては，ScaLAPACK（具体的には PDDSYEVD など）[3]が有名で，これまでに幅広い計算で使用された実績を持っている。しかしながら，近年の計算機において，ScaLAPACK を使用した固有値計算の性能の限界が指摘されるようになってきた[4]。この主な原因として，まず，三重対角化の部分の性能がメモリバンド幅に律速されるために，メモリアクセス性能が演算性能に対して相対的に低下している近年の計算機では性能が出にくい，という計算手順そのものの問題が挙げられる。また，別の原因としては，ScaLAPACK は元々1990 年代

後半の計算機を対象に開発されたので，マルチコアや数万以上のノード数といった最新の計算機の特徴への対応が間に合っていない，という点が挙げられる。

以上のように，現状の ScaLAPACK ではペタ・ポストペタスケールの計算機上で高い性能を得ることが難しくなっているので，ScaLAPACK に代わる新しい固有値計算プログラムの開発が進められている。例えば，ELPA[5]や DPLASMA[6]といったプロジェクトでは，上述の 3 ステップの計算手順ではなく，帯行列を経由して三重対角化を行う別の計算手順を採用して，これの高性能実装を行っている。日本でも，今村らを中心の Eigen シリーズ[7,8]と呼ばれる固有値計算プログラムの開発が進められている。開発が終了した EigenK は，ScaLAPACK と同じ計算手順ではあるが，最近の計算機の特徴に応じた最適化が施されたプログラムであり，その後継で現在開発中の EigenExa は，三重対角行列ではなく帯行列への変形を行った後に帯行列の固有値・固有ベクトルを直接計算するという新しい計算手順を採用したプログラムである。また，これらのプログラムとは別のアプローチとして，ブロックヤコビ法に基づく超並列環境向けの固有値計算プログラムに関する研究[9]なども行われている。

２．２．ペタ・ポストペタスケール向けのプログラム開発

著者は JST CREST のプロジェクト「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」のメンバーとして，主に上述の EigenExa を中心とした密行列向けの固有値計算プログラムに関する研究に携わっている。具体的には「京」コンピュータで開発中の EigenExa の性能評価を行ったり，ポストペタスケールの計算機を想定して，アルゴリズムの妥当性などを検証したりしている。

「京」コンピュータのような最先端の計算機や計画段階であるポストペタスケールの計算機におけるプログラムの性能を評価する場合，十分にプログラムを実行して性能を測定するということが困難となっている。例えば，「京」コンピュータは非常に多くのユーザのジョブが投入されているため，プログラムの実行機会が限られている。特に，詳しい評価が必要である，並列数を増やした場合などについては，より実行が困難となっている。また，ポストペタスケールの計算機における性能の検証については，システムが全て完成してから行うのでは遅いため，システムのスペックやシステムの一部のみしか使えない状況でも，プログラムの性能を何らかの方法で検証する必要性がある。

このような状況を踏まえて，本プロジェクトでは，プログラムの性能予測モデルを構築し，このモデルを通してプログラムの性能評価を行うことを考え，そのための研究をプログラム開発とともに進めている。

２．３．本研究の目的

本研究では，２．１および２で述べた状況を踏まえて，ペタ・ポストペタスケールの計算機上での密行列向け固有値計算プログラムの性能予測モデルの構築法に焦点を当てた基礎研究を行う。具体的には，最初にモデルに求められる要件を整理して，それを踏まえた上で，プログラムの特徴を考慮してモデルの構築方法を検討する。そして，実際の計算機上で実測した計算時間とモデルを用いて予測した計算時間とを比較して，その精度や誤差の原因の調査と改良に向けた検討を行うことを考える。

なお，本研究では，現在開発中の EigenExa の前のバージョンである EigenK のプログラムを予測対象とする。また，今回は第一段階として，固有値計算の計算時間の大半を占めている行列の三重対角化の部分の計算時間を予測することを目標として研究を進める。

３．固有値計算プログラム EigenK の概要

本研究では，EigenK を用いて，固有値計算の主要部分である，行列の三重対角化を行う際の性能を予測するモデルの開発を行う。そこで，本節では，EigenK のプログラムの三重対角化部分について，数理面と実装面を簡単に紹介する。

３．１．数理面

一般的に実対称密行列の三重対角化はハウスホルダー変換とよばれる直交変換を用いた相似変換により行われる。A を入力された実対称密行列とすると，ハウスホルダー変換

$$H_k = I - \frac{1}{\beta_k} u_k u_k^{\mathsf{T}}$$

を使って，

$$H_{N-2} \cdots H_k \cdots H_2 H_1 A H_1 H_2 \cdots H_k \cdots H_{N-2} = T,$$

と三重対角行列 Tに変換する。

また，ステップ 1 回分の処理，つまりハウスホルダー変換 1 個による相似変換の計算は，

$$\begin{aligned} HAH &= \left(I - \frac{1}{\beta} u u^{\mathsf{T}}\right) A \left(I - \frac{1}{\beta} u u^{\mathsf{T}}\right) \\ &= A - u \frac{1}{\beta}\left(Au - \frac{u^{\mathsf{T}} A u}{2\beta}\, u\right)^{\mathsf{T}} - \frac{1}{\beta}\left(Au - \frac{u^{\mathsf{T}} A u}{2\beta}\, u\right) u^{\mathsf{T}} \\ &= A - (u v^{\mathsf{T}} + v u^{\mathsf{T}}), \end{aligned}$$

と変形できる。ただし，

$$\mathrm{v} = \frac{1}{\beta}\left(Au - \frac{u^{\mathsf{T}} A u}{2\beta}\, u\right)$$

である。従って，ステップ内の処理の流れとしては，

① ハウスホルダー変換の生成：u, β の計算でベクトルのノルムの計算に相当，

② 行列ベクトル積：$\mathrm{w} = Au$，

③ vの計算： $\mathrm{v} = \frac{1}{\beta}\left(w - \frac{u^{\mathsf{T}} w}{2\beta}\, u\right)$，

④ A の更新：$A - (u v^{\mathsf{T}} + v u^{\mathsf{T}})$，

となる。

なお，EigenK のプログラム中では，Dongarra の手法により，④の A の更新を遅らせて，先に複数本の$u_k, v_k\ (k = 1, 2, \ldots, M)$を求めて，$M$ステップごとに，

$$A - ([u_1 u_2 \cdots u_M][v_1 v_2 \cdots v_M]^{\mathsf{T}} + [v_1 v_2 \cdots v_M][u_1 u_2 \cdots u_M]^{\mathsf{T}}),$$

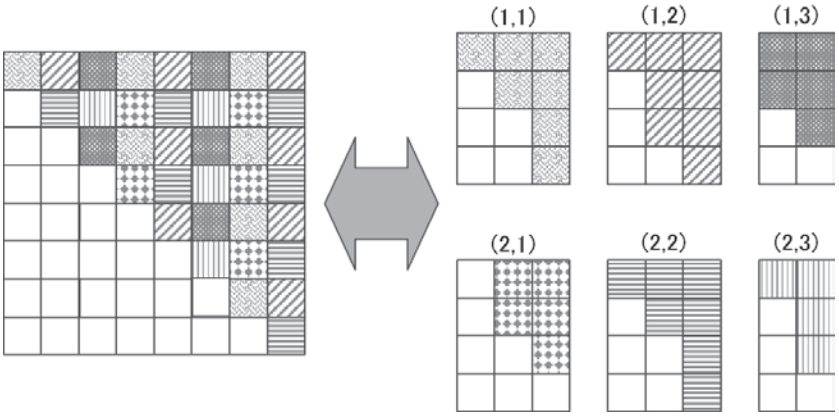
と行列積の形で更新する工夫を採用している。

３．２．実装面

EigenK のプログラムの実装面について簡単に述べる。まず，行列データの分散については，EigenK は二次元のプロセスグリッドを採用しており，これに行列の対称性から，行列の上三角部分のデータを二次元サイクリック・サイクリックの形で分散している（第 1 図）。

プログラムの構造としては，行列サイズ相当のループがあり，ループの 1 ステップにおいて 3.1 で説明した一連の処理を行う，という形になっている（第 2 図）。また，ステップが進むと行列の端から順番に三重対角化されていくため，ステップが進むごとに実質的に計算対象となる行列のサイズが 1 ずつ縮小していく。

次にループ内で行われる行列ベクトル積などといった処理の内部についてだが，これはプロセス内での演算とプロセス間の MPI 通信から構成されている（第 2 図)。なお，通信はスカラー量やベクトル量を行列の列もしくは行方向で行う，という形のみであり，二次元プロセスグリッド上の全てのプロセス間での通信することはなく，どちらか一つの軸方向の通信のみとなっている。また，プロセス内部の演算部分については，OpenMP でスレッド並列化がされている。



第 1 図: EigenK における二次元サイクリック・サイクリック分散の様子。

```
subroutine eigen_trd
    do k=1, N-2
        (プロセス内の演算)
        <MPI_Allreduce (列方向) >      ↕ ハウスホルダー変換の生成
        <MPI_Bcast (行方向) >

        (プロセス内の演算)
        <MPI_Allreduce (行方向) >      ↕ 行列ベクトル積
        <MPI_Allreduce (列方向) >
              .
              .
              .
        if(mod(k,M)==0) then
            (プロセス内での演算)       ↕ 行列積
        end if
    end do
end subroutine
```

第 2 図: EigenK のプログラム中の処理の流れ。

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。

４．性能モデリング

本節では，EigenK による三重対角化の計算の性能予測モデルの構築に向けた検討内容とそれに基づいて構築した実際のモデルについて述べる。

４．１．モデルが満たすべき要件

性能予測モデルの検討には，どのような情報を使って何が（どの程度）予測できれば良いのか，ということを明確にしておく必要がある。

今回の目標は，ペタ・ポストペタスケールの計算機向けの固有値計算プログラムの開発において，プログラム開発者が開発やチューニングに活用できる性能予測モデルを構築することである。まず，プログラム開発者側を考えているので，

- プログラムはブロックボックスではなく，演算回数や通信回数などの内部が分かっている。
- プログラムを修正して，モデリングに必要なベンチマーク等を行うことが可能である。

ということを前提とする。一方，ペタ・ポストペタスケールの計算機を想定しているので，

- 長時間の実行やシステムの大部分を使用する実行が難しく，実測可能な問題サイズや並列数に制限がある。

という制約を考慮する必要がある。そのうえで性能予測モデルには，開発やチューニングに活用する，という目的を果たすために，

- プログラム全体の実行時間に加えて，演算と通信，また，プログラム中の主要な処理ごとにそれぞれ精度良く予測する。

ということが求められる。

４．２．モデルの構築

４．１で述べたように，今回想定している状況では，並列数や問題サイズについて十分な範囲の実測データを得ることが難しいため，実測データを関数で最小二乗近似する，といった方法は適していないと判断できる。一方で，プログラム内部がブラックボックスではないため，各処理の演算量や通信パターンについては事前に明らかになっている。そこで，これらを使って各処理の実行時間を予測してそれを積み上げる，という階層的な性能モデリング[10,11]が可能であると考える。また，階層的なモデリングを行うと，処理単位でも予測時間が得られるため，今回のモデルに対する要件の一つを満たすことができると考える。以上のことから，今回は階層的なモデリングという方針の下でモデルの構築を行う。

三重対角化の計算は，行列サイズ相当のループの中で，1 ずつ行列のサイズを縮小しながら，ハウスホルダー変換の作成や行列ベクトル積といった数種類の処理を行う，という構造になっている。今回のモデリング対象のプログラムである EigenK では，サイクリック・サイクリック分散で行列データを分散しているため，各処理の時間はプロセス間で差がなく同じであると仮定する。このようにすると，

- N：入力行列のサイズ
- P：並列数（MPI プロセス数）

としたときの，

- $T(N,P)$：サイズ N の行列を P 並列で三重対角化する計算時間

は，各処理の時間の積み上げとして，

$$T(N,P) = \sum_{1 \le k \le N-2} \sum_{j \in J} T^{(j)}(N-k+1,P)$$

とモデル化することができる。なお,

- 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，...}）
- $T^{(j)}(N',P)$ ：並列数が Pのもとで，行列サイズが N'のときの処理 jを行う時間

としている。

次に，各処理 1 回分の時間$T^{(j)}(N',P)$に関して考える。各処理の中身は，プロセス内での浮動小数点演算とプロセス間の通信から成り立っている（演算のみの処理もある）。三重対角化の計算は逐次性が強く演算と通信のオーバーラップが困難であり，EigenK でもオーバーラップは施されていない。したがって，

- $T_{\mathrm{f}}^{(j)}(N',P)$ ：並列数が Pのもとで，行列サイズが N'のときの処理 j中の演算時間
- $T_{\mathrm{c}}^{(j)}(N',P)$ ：並列数が Pのもとで，行列サイズが N'のときの処理 j中の通信時間

とすれば,

$$T^{(j)}(N',P) = T_{\mathrm{f}}^{(j)}(N',P) + T_{\mathrm{c}}^{(j)}(N',P)$$

と演算時間と通信時間の和で表すことができる。

これを用いて，和をとる順番を入れ替えると，結局,

$$\begin{aligned} T(N,P) &= \sum_{j\in J}\left[\sum_{1\le k\le N-2}\left\{T_{\mathrm{f}}^{(j)}(N-k+1,P) + T_{\mathrm{c}}^{(j)}(N-k+1,P)\right\}\right] \\ &= \sum_{j\in J}\left\{\sum_{1\le k\le N-2} T_{\mathrm{f}}^{(j)}(N-k+1,P)\right\} + \sum_{j\in J}\left\{\sum_{1\le k\le N-2} T_{\mathrm{c}}^{(j)}(N-k+1,P)\right\} \end{aligned}$$

と,処理ごとに三重対角化全体における演算時間と通信時間を個別に求めればよいことになる。よって,以下では,ある処理に対して,その演算時間と通信時間をどのようにモデル化するか,について考える。

４．２．１．演算時間のモデル化

演算時間のモデル化について考える。今回の場合は演算量が事前に分かっているので,

- $F^{(j)}(N',P)$ ：処理 j（並列数が P，行列サイズが N'）における浮動小数点演算の回数
- $\gamma^{(j)}(N',P)$ ：処理 j（並列数が P，行列サイズが N'）における実効 FLOPS 値

として,

$$T_{\mathrm{f}}^{(j)}(N',P) = \frac{F^{(j)}(N',P)}{\gamma^{(j)}(N',P)}$$

とモデル化することできる。ここで，実効 FLOPS 値がデータのサイズ（全体の行列サイズと並列数で決まる）に依らず一定：$\gamma^{(j)}$だとすれば,

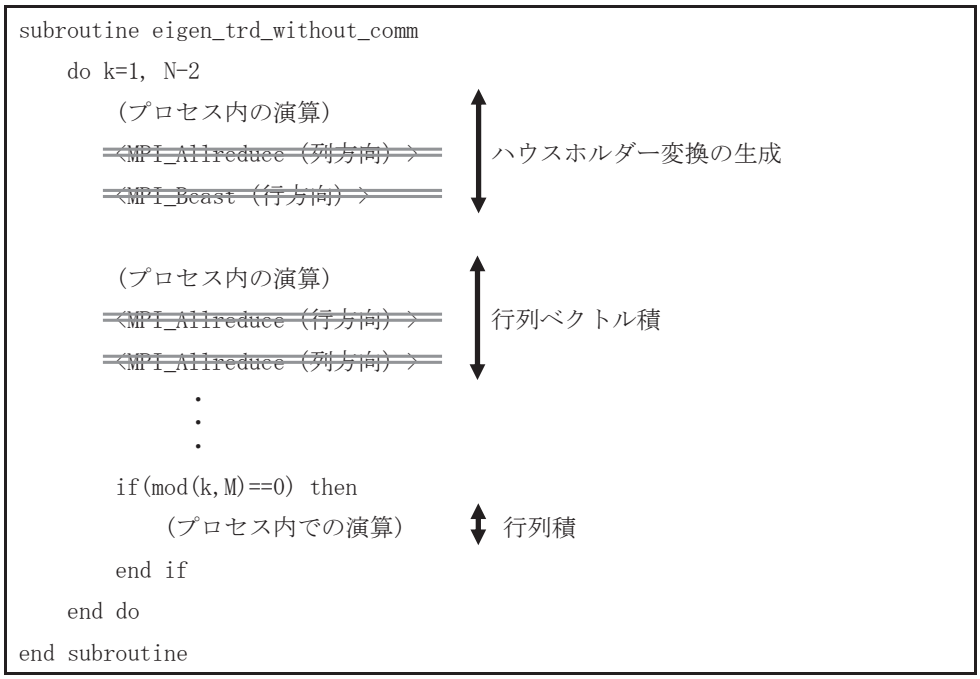
$$\begin{aligned} \sum_{1\le k\le N-2} T_{\mathrm{f}}^{(j)}(N-k+1,P) &= \sum_{1\le k\le N-2}\frac{F^{(j)}(N-k+1,P)}{\gamma^{(j)}} \\ &= \frac{\sum_{1\le k\le N-2} F^{(j)}(N-k+1,P)}{\gamma^{(j)}} \\ &= \frac{(\text{処理}j\text{の演算量の合計})}{\gamma^{(j)}} \end{aligned}$$

と，三重対角化全体を通しての処理の演算量の合計（これは既知）から演算時間を算出することができる。ただし，一般的に並列数が大きいような場合，1 プロセス当たりのデータサイズは小さくなくことが多く，そこからさらにサイズが縮小していくため，$\gamma^{(j)}$が一定という仮定

は無理があると言える。

一方，$\sum_{1\le k\le N-2} T_{\mathrm{f}}^{(j)}(N-k+1,P)$はプログラム中の演算部分のみの時間に相当するので，プログラムの通信部分を全て取り外して 1 プロセスだけで実行することで，その値を実測することが可能である。今回想定している三重対角化のような密行列計算はアルゴリズム中の各変数の値が数学的に意味のないものであったとしても，アルゴリズムが破綻してプログラムが途中で止まることはほとんどない。そこで，今回は対象としているシステム全体の使用は難しいが，1 ノードは使用が可能である，と仮定して，EigenK の三重対角化のプログラムの通信部分を全てコメントアウトした（第 3 図），演算時間測定用のプログラムを 1 ノード上で実行して，各処理（j）に対する$\sum_{1\le k\le N-2} T_{\mathrm{f}}^{(j)}(N-k+1,P)$の値を実測する。

なお，1 ノードですら使用が不可能なシステムを対象にする場合には，本節の前半で述べたように，実効 FLOPS 値を一定として合計演算量から算出する等の方法を採用することになる。



第 3 図：演算時間のみを測定する修正版のプログラムの様子。

図のように MPI 通信関数の部分をコメントアウトしても，プログラムとしては破綻することなく最後まで動くことができるので，これを 1 ノード上で実行することで，各処理の演算時間を測定することができる。

４．２．２．通信時間のモデル化

通信時間のモデル化について考える。以下では，処理の中で複数回の通信を行う場合があるので，それをインデックス i で区別する。すると，

$$\sum_{1\le k\le N-2} T_{\mathrm{c}}^{(j)}(N-k+1,P) = \sum_{1\le k\le N-2}\sum_{i} T_{\mathrm{c}}^{(j,i)}(N-k+1,P)$$

と書くことができる。このようにすると$T_{\mathrm{c}}^{(j,i)}(N',P)$，が MPI 通信一つに対応することになる。そして，ある MPI 通信 1 回の通信時間については，

- $\alpha^{(j,i)}(P)$：並列数が Pのときのある MPI 通信（j,i）のセットアップコスト

- $\beta^{(j,i)}(P)$ ：並列数が P のときのある MPI 通信（j, i）の実効バンド幅の逆数
- $W^{(j,i)}(N', P)$ ：並列数が P で行列サイズが N' のときのある MPI 通信（j, i）のデータ量

として，一般的に使用されているモデル

$$T_c^{(j,i)}(N', P) = \alpha^{(j,i)}(P) + \beta^{(j,i)}(P) \cdot W^{(j,i)}(N', P)$$

で書けるとする。

これより，和をとる順番を入れ替えて整理すると，

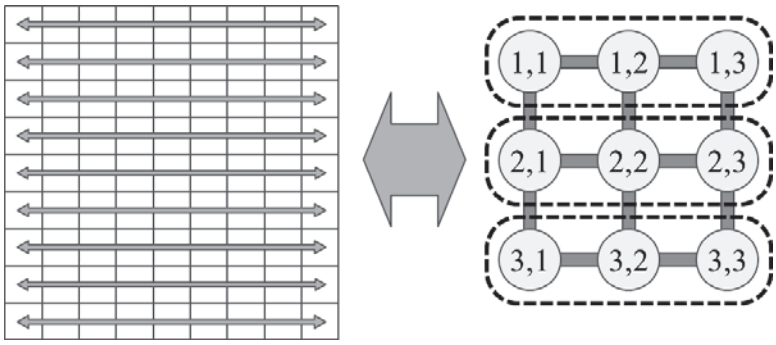
$$\begin{aligned}\sum_{1\le k\le N-2} T_c^{(j)}(N-k+1, P) &= \sum_i \sum_{1\le k\le N-2i} T_c^{(j,i)}(N-k+1, P) \\ &= \sum_i \sum_{1\le k\le N-2} \left\{\alpha^{(j,i)}(P) + \beta^{(j,i)}(P) \cdot W^{(j,i)}(N', P)\right\} \\ &= \sum_i \left[\alpha^{(j,i)}(P) \cdot \left\{\sum_{1\le k\le N-2} 1\right\} + \beta^{(j,i)} \cdot \left\{\sum_{1\le k\le N-2} W^{(j,i)}(N-k+1, P)\right\}\right] \\ &= \sum_i \left[\alpha^{(j,i)}(P) \cdot (\text{通信回数}) + \beta^{(j,i)}(P) \cdot (\text{通信データの総量})\right]\end{aligned}$$

となり，モデル式の係数：$\alpha^{(j,i)}(P)$，$\beta^{(j,i)}(P)$と既知の量である通信回数と通信データの総量から通信時間を算出することができる。

したがって，モデルを構築する際に問題となるのは，このモデル式の係数をどのようにして求めるか，ということになるので，その点について検討する。まず，EigenK では 2 次元のプロセスグリッドを採用しており，通信は全て 1 次元の軸内で閉じている，という特徴がある（第 4 図）。そのため，プロセス数を平方数に限定した上で，同時に行われる通信同士が互いに影響を与えないと仮定すると，

$$\alpha^{(j,i)}(P) = \tilde{\alpha}^{(j,i)}\left(\sqrt{P}\right), \qquad \beta^{(j,i)}(P) = \tilde{\beta}^{(j,i)}\left(\sqrt{P}\right)$$

と係数を置き換えることができる。これより，今回は$\sqrt{P}$程度のノード数で MPI 関数のベンチマークを行って係数（$\tilde{\alpha}^{(j,i)}\left(\sqrt{P}\right)$，$\tilde{\beta}^{(j,i)}\left(\sqrt{P}\right)$）を算出することは可能だと判断して，実測値から係数を求めることにする。なお，$\sqrt{P}$程度のノードですら使用が難しい場合は，一対一通信の場合の係数を基にして算出するなどの方法を採用することになる。



第 4 図：EigenK における通信パターンの様子。

EigenK では左図のように行列の行内もしくは列内での通信のみを行う。これらの通信は右図のように二次元プロセスグリッドの 1 つの軸内で閉じており，全体としては 1 軸方向の通信を同時に行うことになる。

４．３．構築したモデルの全体像

前節の内容をまとめると，問題サイズが N，並列数が P（ただし平方数）の場合の三重対角化の並列実行時間は，

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

というモデル式で表される。また，モデル式中の各項は意味と取得方法は，

- $T_{\mathrm{f\_total}}^{(j)}(N,P)$ ：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。

このモデルを使って実際に並列実行時間を予測する際のコストについては，

- $T_{\mathrm{f\_total}}^{(j)}(N,P)$ 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。

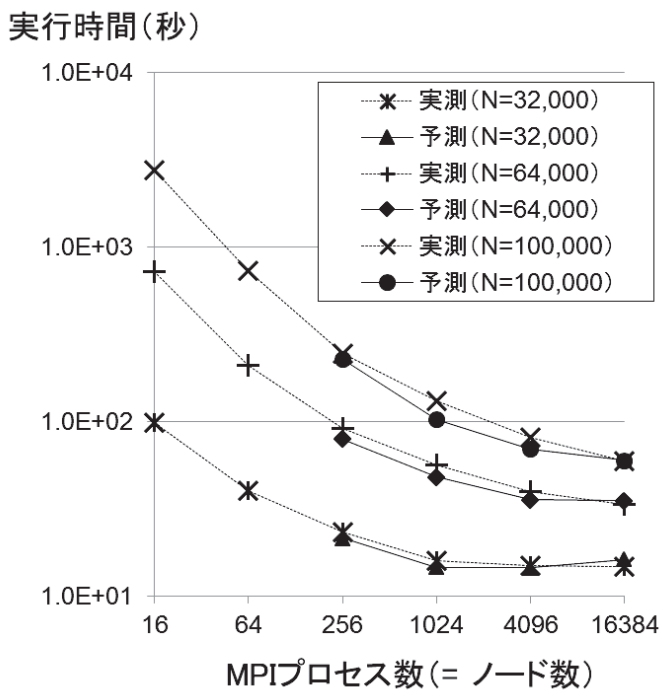
となっており，今回のモデル化における制約は満たしていると言える。

５．数値実験

前節で述べた性能予測モデルを使って，「京」コンピュータにおける EigenK の三重対角化の部分の並列実行時間を実際に予測して，プログラムを実行して測定した実測値と比較した結果を紹介する。

行列サイズは 32,000，64,000，100,000 の三種類として，MPI プロセス数が 256，1024，4096，16,384 の場合についての予測を行った。なお，プログラムを実際に実行した際にはノード数は二次元形状で指定しており，1 ノードに 1MPI プロセス，1 プロセスについて 8 スレッドを指定した。また，BLAS は「京」コンピュータで提供されている富士通製の BLAS をリンクしている。

第 5 図に示した結果のグラフより，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。このグラフにあるような予測精度であれば，例えば，モデルを用いて与えられた行列サイズに対して，どの程度の並列数までスケールするか，といった検証をすることができるのではないかと考えている。



第５図: 「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の予測結果。

「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について，4 節で述べ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，モデルによる予測結果が良好であることが確認できる。

５．おわりに

本稿では，ペタ・ポストペタスケールの計算機向けの密行列固有値計算プログラムの開発への活用を念頭において，固有値計算プログラムの性能予測モデルの構築を目標とした研究の現状を紹介した。限られた条件での実測結果のみが使用できるという制約の下と，通信パターンといったプログラムの特徴を踏まえたうえで，モデルの構築に向けた検討を行い，限られた条件の中で取得可能な実測データを利用した階層的な性能予測モデルを実際に構築した。そして，「京」コンピュータ上で検証を行った結果，行列の三重対角化部分の計算時間を比較的良い精度で予測できていることが確認できた。

今後は，今回の議論で導入した仮定の妥当性，合計時間だけでなくプログラム中の処理ごとの演算時間や通信時間の予測精度の検証，といったことを行う必要がある。これらを行うためには，実際に並列実行しているプログラム中の処理単位の演算時間や通信時間を比較対象として測定することが必要になるが，その際には，測定のためのオーバーヘッドなどを考慮することも必要になる。より詳細な検証を行い，実際にプログラム開発に活用できる性能予測モデルを構築するとともに，固有値計算以外の行列計算プログラムの性能予測への展開も考える。

最後になるが，4 節で述べたモデル構築の議論は，「京」と同じアーキテクチャを持つ東京大学の FX10 を用いた様々な予備実験に基づいた試行錯誤の結果である。今回は話の構成と紙面の都合から，この部分を述べることができなかったが，別の機会に紹介できればと思う。

謝辞

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」（平成 24 年度後期）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] T2K Open Super Computer: HPCI 技術ロードマップ白書,
http://open-supercomputer.org/wpcontent/uploads/2012/03/hpci-roadmap.pdf.

[2] L. Trefethen and D. III, Numerical Linear Algebra, SIAM, 1997.

[3] ScaLAPACK: http://www.netlib.org/scalapack/.

[4] 今村俊幸, T2K スパコンにおける固有値ソルバの開発, スーパーコンピューティングニュース, Vol.11, Mo.6, pp.12-32, 2009.

[5] T. Auckenthaler et al., Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations, Parallel Comput., Vol. 37, No. 12, pp.783-794, 2011.

[6] G. Bosilca et al., Distibuted Dense Numerical Linear Algebra Algorithms on massively parallel architectures: DPLASMA, in Proceedings of the 25th IEEE International Symposium on Parallel & Distributed Processing Workshops and Phd Forum (IPDPSW'11), pp.1432-1441, 2011.

[7] T. Imamura et al., Development of a high performance eigensolver on the peta-scale next generation supercomputer system, in NUCLEAR SCIENCE and TECHNOLOGY, pp.643-650 2011.

[8] 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, 2012.

[9] 高橋佑輔 他, ブロックヤコビ法による超並列固有値計算プログラムの開発, スーパーコンピューティングニュース, Vol.15, No.2, pp.21-30, 2013.

[10]K. Dackland et al., A Hierarchical Approach for Performance Analysis of ScaLAPACK-Based Routines Using the Distributed Linear Algebra Machine, in Proceedings of the Third International Workshop on Applied Parallel Computing, Industrial Computation and Optimization (PARA'96), pp.186-195, 1996.

[11]Y. Yamamoto, Performance modeling and optimal block size selection for the small-bulge multishift QR algorithm, in Proceedings of the 4th international conference on Parallel and Distributed Processing and Applications(ISPA'06), pp.451-463, 2006.