Vol.2013-HPC-140 No.41
2013/8/2

超並列環境における密行列計算プログラムの性能モデリングに向けた検討

深谷 猛[1,3,a)]　今村 俊幸[2,3]　山本 有作[1,3]

概要：現在のペタスケールの計算機やこれから先のポストペタスケールの計算機向けに密行列計算プログラムを開発する際，性能モデルを用いて性能を予測することで作業を効率化することが期待される．本稿では，そのような性能モデルを構築するための検討を行うことを目的とし，その検討材料として，我々が行った「京」コンピュータにおける行列の三重対角化プログラムの性能モデリングと，その過程で行った予備調査の結果を報告する．そして，その結果を踏まえて，密行列計算の性能モデリングに向けた検討を行う．

1. はじめに

「京」コンピュータをはじめとする超並列計算機を用いた科学技術計算が行われるようになり，これらの環境で効率的に行列計算を行うためのライブラリに対するニーズが高まっている [1]．分散環境用の密行列計算ライブラリとしては ScaLAPACK[2] が有名であるが，一方で，ScaLAPACK の超並列環境での性能の限界も指摘されているため，新しい行列計算ライブラリの開発が望まれている．例えば，密行列向けの固有値計算ライブラリに関しては，海外では DPLASMA[3] や ELPA[4] といったライブラリの開発が進められている．同様に国内でも，「京」コンピュータ向けに Eigen-K[5] が開発されたり，ポストペタスケール環境向けに Eigen-Exa[6] の開発が始まったりしている．

このように最近あるいは今後の超並列環境向けに新しい行列計算プログラムの開発や従来のプログラムの改良が進められているが，その過程では当然プログラムの性能評価を行う必要がある．しかしながら，「京」について考えてみても，多くのユーザで計算機資源を共有しており，必要な分だけプログラムをテスト実行して性能データを採取することができるとは限らない．さらに，計画段階のシステム向けの開発を行う場合，システムの一部のみしか利用できないといった状況も考えられる．このような状況において，プログラムの性能モデリングを行い，評価に活用することが有効ではないかと著者らは考えている．

実際，これまでも行列計算プログラムの評価やプログラム内のパラメータの最適化をモデルに基づいて行う，といったことはなされてきた [7], [8]．例えば，最近の超並列環境向けに提案されている CAQR (Communication Avoiding QR) アルゴリズムに関する論文では，簡単なモデルにより超並列環境上での性能を予測し，その結果を用いてアルゴリズムの評価を行っている [9]．一方で，これらの既存研究では性能モデリングそのものが主題となっていないため，実際に自分でモデリングをしようとした場合に必要となる情報が欠けていることがある．また，モデリングの精度などについてあまり詳しく検証されていないこともある．

現在，著者らはポストペタスケール向けの密行列向け固有値ライブラリの開発を進めており，その一環として，ポストペタ環境での性能を予測するためのモデルの開発を行っている．具体的には，「京」コンピュータを中心に，固有値計算プログラムの性能の挙動を調査したり，実際にモデルを構築して予測結果と実測結果の比較をしたりしている．そこで本稿では，これまでに「京」の上で行った固有値計算プログラムに関して，プログラムの並列実行時の挙動調査およびモデリング自体の結果を報告する．そして，その内容を踏まえて密行列計算プログラムの一般的なモデリングに向けた検討を行う．

以降，2 節で我々が「京」の上で行っているモデリングの概要を説明し，3 節でプログラムの挙動調査の結果の報告，4 節でモデリング自体の結果を報告する．これらの内容を踏まえて，5 節で一般の密行列計算プログラムのモデリングに向けた検討をし，最後にまとめを 6 節で述べる．

1 神戸大学
2 理化学研究所
3 JST CREST
a) fukaya@people.kobe-u.ac.jp

ⓒ 2013 Information Processing Society of Japan

2. 固有値計算プログラムの性能モデリングの概要

本節では，我々が「京」コンピュータを中心に行っている，固有値計算プログラムの性能モデリングに関する研究の概要を説明する．

2.1 目的・方針

我々のモデリングの目的は，「京」のようなペタスケール環境やその先のポストペタスケール環境における固有値計算プログラムの性能をモデルにより予測し，その結果をアルゴリズム選択やチューニングに活用することである．つまり，プログラムを開発する立場の人間が行うモデリングであり，

- プログラムはブラックボックスではなく，内部の処理について把握している．
- モデリング対象のプログラムを必要に応じて修正して，ベンチマーク等に利用できる．
- プログラム全体の実行時間だけでなく，内部の各処理における演算時間や通信時間も予測することを目的とする．

ということを以降では想定している．

また，対象とする計算機環境は，すでに述べたようにペタ・ポストペタスケールの環境であり，そのため，

- 十分な実測結果（特に問題サイズが大きい場合や並列数が多い場合）の取得が困難である場合でもモデリングを可能とする．
- できるだけ，シングルノードや少ないノードでの実測結果や MPI のベンチマークなどの結果を利用する．

ということをモデリングの際の方針としている．

2.2 モデリングの対象とするプログラムの概要

今回，性能モデリングの対象とするプログラムは，「京」コンピュータ向けに公開されている対称密行列の固有値計算ライブラリ Eigen-K[5] 中の，実対称行列を三重対角行列に変形するルーチン（以下，Eigen-Trd）である．

処理の内容としては，与えられた $N \times N$ の対称行列に対して，ハウスホルダー変換を繰り返し作用させることを行列を三重対角化する [10]．したがって，プログラムとしては，**図 1**（左）に示したように，行列サイズに相当するループ[*1]があり，その中でハウスホルダー変換の生成（gen_house，ベクトルの内積計算など）や行列ベクトル積（mat_vec）といった，いわゆる BLAS 演算に相当する処理を行う，という流れになっている．

実装方法についてだが，MPI と OpenMP のハイブリッド

```
subroutine eigen_trd(...)
    ⋮
  do k = 1, N / M
    do i = 1, M
        ⋮
      gen_house(...)
      mat_vec(...)
        ⋮
    end do
    mat_mat(...)
      ⋮
  end do
    ⋮
end subroutine
```

```
subroutine sample_func1(...)
  MPI_Barrier(...)
  t1 = MPI_Wtime()          } ···(*)
    ⋮
  MPI_Barrier(...)
  ct1 = MPI_Wtime()         } ···(**)
  MPI_Allreduce(...)
  MPI_Barrier(...)
  ct2 = MPI_Wtime()         } ···(**)
  g_ctime1 += ct2 - ct1
    ⋮
  MPI_Barrier(...)
  t2 = MPI_Wtime()          } ···(*)
  g_time1 += t2 - t1
end subroutine
```

図 1 三重対角化プログラムの流れ（左）と各処理の実行時間の測定方法（右）

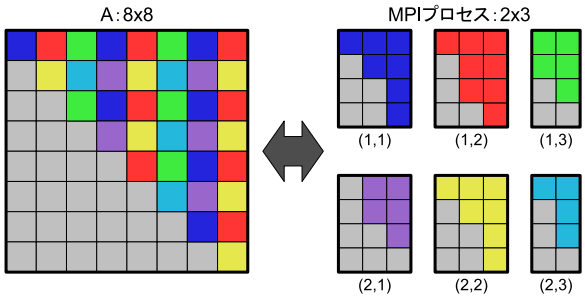

図 2 Eigen-Trd におけるデータ分散とプロセス配置

並列化となっている．行列データは**図 2** に示したような二次元サイクリック分散を採用しており，それに応じて MPI プロセスも二次元グリッド上に配置されている．並列化に関しては，内積や行列ベクトル積などの各処理をデータ並列の形で MPI により並列化し，さらに，各 MPI プロセス内の処理も OpenMP によりスレッド並列化している．通信に関しては，上記の処理の過程で必要に応じてベクトルもしくはスカラーの通信を同じ行（もしくは列）のプロセス間で行っている．[*2]なお，演算と通信のオーバーラップは行われていない．

2.3 計算機環境に関する条件

本稿で報告する調査結果は「京」コンピュータにおけるものである．「京」のハードウェアに関する詳細な仕様については [11] 等に委ねるが，実行時の条件などは**表 1** に挙げた通りである．

*1 厳密にはハウスホルダー変換の作用は $N-2$ 回であるが，説明を簡略化するために N としている．また，図 1 のループが二段になっているのは行列積（mat_mat）を使用するためにブロック化を行っているからである．

*2 行方向と列方向のプロセス数が異なる場合に，データの分散においてこれ以外の通信パターンが発生することがある．

表 1 評価環境

	スーパーコンピュータ「京」
CPU	SPARC64 VIIIfx 2.0GHz, 8cores/socket
メモリ	DDR3 SDRAM 64GB/s 16GB/socket
ネットワーク	Torus fusion 6D mesh/torus, 5GB/s
言語環境	K-1.2.0-09
コンパイラ	mpifrtpx, オプション：-Kfast,openmp
MPI	Fujitsu MPI
BLAS	Fujitsu BLAS (-SSL2)
プロセス配置	1process/1node
スレッド数	8threads/1process

3. 「京」における三重対角化プログラムの挙動の調査

本節では，三重対角化プログラム Eigen-Trd の性能モデリングに向けた予備調査として，「京」コンピュータ上で実際に Eigen-Trd を実行して，演算性能や通信性能の挙動を調査する．

3.1 タイマー・同期の影響

2.1 節で述べたように，今回はプログラムの全体の実行時間だけでなく，内部の各処理に関してもできるだけ詳しく実行時間を予測することを目指している．つまり，図 1（左）における mat_vec などの関数ごとに演算時間と通信時間を評価することを念頭においている．

今回は，時間を測定するために図 1（右）に示したような形で，タイマー及び同期処理をプログラムに加えた．ここで，g_が頭についている変数はグローバル変数としている．また，同期をとっているのは，MPI_Bcast などの関数で処理が戻るタイミングがプロセスごとに異なることがあるからである．

さて，前述のようにタイマーや同期処理を加えることで実行結果に影響があるかどうかが懸念される．そこで，タイマーや同期処理のコストについて調査を行った．[*3]その結果，MPI_Wtime1 回当たりの実行コストは 1.5μ 秒程度，MPI_Barrier1 回当たりの実行コストはプロセス数に依存する形で**表 2** の値となった．

次に，このコストが実行時間に与える影響を調査するため，Eigen-Trd 全体の実行時間を図 1（右）の $*, **$ が両方ない場合，$*$ のみがある場合，$*, **$ の両方がある場合の 3 パターンで測定した．この結果を，**表 3** に示す．ここで $*, **$ の有無は図 1（右）に対応しており，調整が有りの場合は得られた実行時間から前述したタイマーや同期のコストをコールした回数分だけ差し引いている．

表 3 の結果より，今回の状況では，図 1（右）のような形でタイマーや同期処理を入れると，それに伴うコストが全体の実行時間に対して無視できないものになることが分かった．したがって，モデリングやモデリング結果の評価を行う際には，これらのコストの影響を考慮することが必要となる．なお，表 3 より，タイマーや同期処理を除いたとしても，全体の実行時間が遅くなっていることが読み取れるが，この原因は調査中である．[*4]

[*3] MPI_Wtime や MPI_Barrier のみを繰り返し呼ぶようなループを作り，全体の実行時間を実行回数で割ることでコストを求めた．

表 2 MPI_Barrier1 回当たりの実行コスト

プロセス数	16	64	256	1024	4096
コスト (μ 秒)	2.9	3.9	5.2	7.6	9.5

表 3 タイマーと同期処理が Eigen-Trd 全体の実行時間に与える影響（4096 プロセス）

		Eigen-Trd 全体の実行時間 (秒)				
条件	$*$	無	有	有	有	有
	$**$	無	無	無	有	有
	調整	–	無	有	無	有
N	6400	2.5E+0	3.5E+0	2.8E+0	4.9E+0	3.1E+0
	32000	1.4E+1	1.9E+1	1.5E+1	2.5E+1	1.6E+1
	64000	3.5E+1	4.6E+1	3.9E+1	6.1E+1	4.2E+1
	100000	7.1E+1	9.0E+1	7.9E+1	1.2E+2	8.6E+1

3.2 演算性能の挙動

次に，各処理における演算時間の挙動を調べた結果について報告する．ここで，演算時間 t は処理の実行時間から処理内の MPI 通信の時間を差し引いた時間（図 1（右）の g_time1 - g_ctime1）としている．また，この際も前節で述べたタイマーや同期のコストが影響するので，これらのコスト分を差し引いている．

今回は，Eige-Trd の主要な処理の中から，

(1) gen_house : ベクトルの 2-ノルムの計算で演算量は $\frac{N^2}{\sqrt{P}}$.
(2) mat_vec：行列ベクトル積で演算量は $\frac{2N^3}{3P}$.
(3) calc_v：段々とサイズが増える細長い行列とベクトルの積で演算量は $\frac{2MN^2}{\sqrt{P}}$.
(4) mat_mat：行列積で演算量は $\frac{2N^3}{3P}$.

に関して演算性能を調査した結果を報告する．[*5]なお，P は MPI プロセス数，演算量は 1 プロセス当たりのもの，M は内部のパラメータで今回は 32 に固定している．

上記の 4 種類の処理に対して，行列サイズ N とプロセス数 P をいろいろと変化させて Eigen-Trd を実行して時間を測定し，それぞれの場合について 1 プロセス当たりの実効性能を算出した．実効性能はデータサイズに依存して変化することが予想されるので，横軸を 1 プロセス当たりの行列データのサイズ $\frac{N}{\sqrt{P}}$ をとして，算出した実効性能をプ

[*4] もともと，このプログラムでは行列を 2 次元サイクリック分散しており，各プロセスの負荷は均一に近いはずではあるが，それでも少なからず負荷が不均一になると思われる．加えて，MPI 通信から処理が戻るタイミングはプロセスによって異なるので，これらの影響で処理がうまく詰まっていたところ，同期のためにそれがなくなって実行時間が遅くなった，ということが考えられる．

[*5] これ以外にもうひとつ主要な処理があるが，演算としては gen_house に比較的近いので，紙面の関係上割愛した．

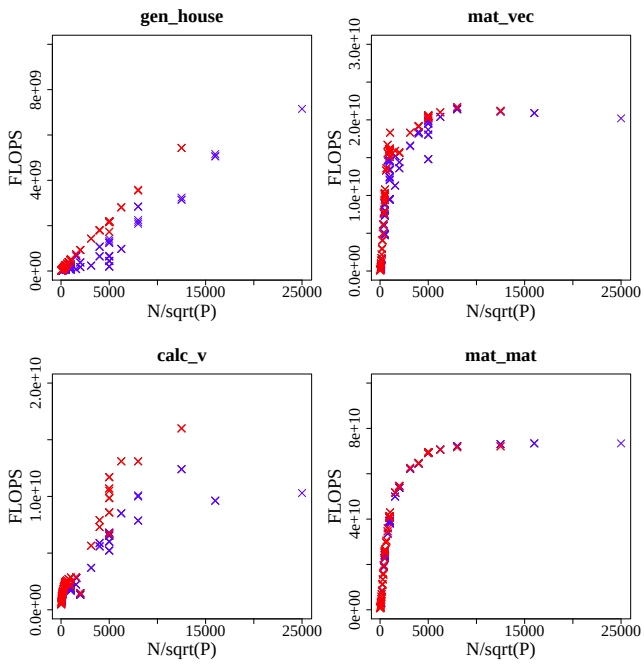

図 3 Eigen-Trd の各処理の演算性能の挙動（青：実際に並列実行して測定，赤：シングルノード上で仮想的に実行して測定）

ロットした結果を**図 3** に示す（青のマーカー）．

一方，今回の演算では計算途中の値自体は演算性能に影響を与えることがない．そのため，MPI 通信部分を取り除いて，シングルノード上で同じパターンの演算を行うプログラムを作成することが可能である．[*6]そこで，そのようなプログラムを作成して，シングルノード上で N や P を変えて演算性能を測定した結果を併せて図 3 に示す（赤のマーカー）．

図 3 の結果より，演算性能は演算パターンによって挙動が異なり，いずれの場合もデータサイズによって性能が変化していることが確認できる．そのため，単純に理論ピーク性能の何パーセントというような値を用いてモデリングを行うことは難しいと判断できる．一方，mat_vec と mat_mat については，実際に並列実行した際の演算性能の挙動とシングルノード上で仮想的に実行して測定した性能の挙動が似ていることが確認できる．この事実より，今回の三重対角のように，全体に対して mat_vec や mat_mat の計算が大部分を占める場合には，シングルノード上で実行して測定した値から並列実行時の演算時間を算出しても，許容できる結果になるのではないかと期待できる．なお，gen_house と calc_v に関して，並列実行したときの挙動とシングルノードで実行したときの挙動がずれている原因については，まだ明らかになっていない．

3.3 通信性能の挙動

最後に，MPI 通信の性能の挙動を調べた結果について報告する．

[*6] 当然，このプログラムの計算結果には何の意味もない．

図 2 に示したように，Eigen-Trd では二次元グリッド上にプロセスを配置しており，通信はグリッドの一次元方向のみとなっている．したがって，「京」においてノードの形状を二次元に指定した場合，ネットワークは仮想的に二次元トーラスとなり，並行して行われる一次元の通信同士が互いに干渉しないことが期待される．すなわち，並列数を P とした場合，$\sqrt{P}$ 程度の並列数での MPI の通信性能が分かれば，並列数 P で Eigen-Trd を実行した場合の通信性能が予測できる可能性がある．

そこで，ノードの形状を 64x64(=4096) として確保して Eigen-Trd を実行した場合の MPI 通信ごとに通信時間と，64 ノードでデータサイズを変えながら MPI 通信の時間を測定するベンチマークを実行した結果の比較を行った．その結果を**図 4** に示す．ここで，グラフ中の青いマーカー 1 つが Eigen-Trd 中の MPI 通信 1 つに対応している．また，赤い直線は 64 ノードで実行した MPI 通信のベンチマークで得られた結果を最小二乗近似した直線となっている．さらに，図 4 のそれぞれにグラフにおいて，各マーカーの直線からの相対的なずれの分布をヒストグラムにまとめた結果を**図 5** に示す．なお，ヒストグラムの区間は 0.05 きざみとしている．この二つの結果より，多少の影響はあるものの，二次元形状でノードを確保した場合の Eigen-Trd 中の MPI 通信の性能は，$\sqrt{P}$ 程度で実行した MPI のベンチマーク結果を利用して予測することが可能だと言える．なお，図 5 から，Allreduce より Bcast の方がずれが小さいことが確認できるが，これは Bcast の方が Allreduce と比べてネットワークを流れるデータ量が少ないためではないかと考えらる．

一方，ノード形状を二次元にしなかった場合についても調査した．ノードを単に 4096 と指定して Eigen-Trd を実行した場合について，図 4 と同様の比較をした結果を**図 6** に示す．また，相対的なずれの分布をまとめたヒストグラムを**図 7** に示す．図 4 と図 6 を比較すると，行方向の通信に関してはあまり違いがないが，列方向については直線からのずれが大きくなっていることが確認できる．これは図 5 と図 7 のヒストグラムを比較しても明らかである．

おそらくノードの形状を指定しない場合，一次元トーラスとしてノードが確保されたため，そのノードを二次元グリッドに配置すると，どちらかの次元（今回は行方向）についてはノードが隣接し，かつ，お互いの通信があまり干渉しないためベンチマークの結果と比較的近く，それに対して，もうひとつの次元方向（列方向）の通信ではノードがとびとびになり経路が重なったことで通信の衝突が頻発したため，ベンチマークの結果（通信同士の衝突が少ない場合）からのずれが大きくなったと予想できる．このようなケースでは，$\sqrt{P}$ のノードで行ったベンチマークの結果をそのまま利用するのではなく，通信同士の衝突を考慮する工夫を考える必要があると言える．

ⓒ 2013 Information Processing Society of Japan

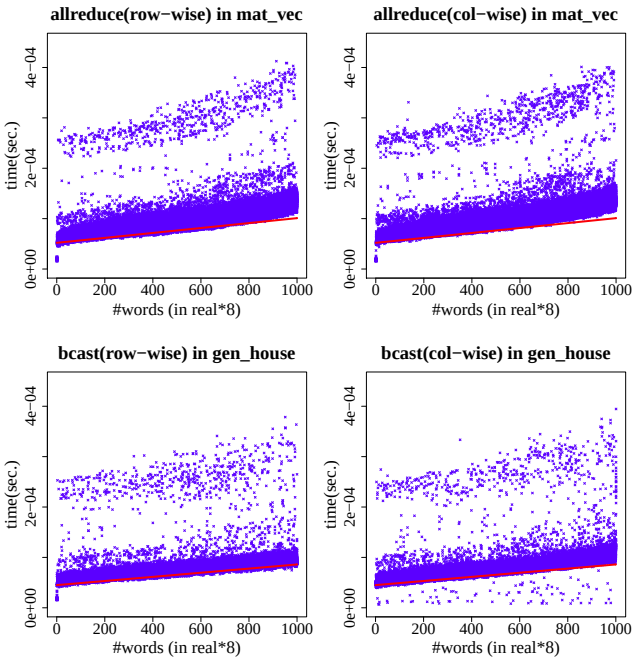

図 4 Eigen-Trd における MPI 通信の挙動とベンチマークにおける挙動の比較（青：4096 ノード（64x64 で形状指定）で Eigen-Trd を実行した際の各 MPI 通信の実行時間，赤の線：64 ノードで MPI のベンチマークを実行した結果を最小二乗近似した直線）

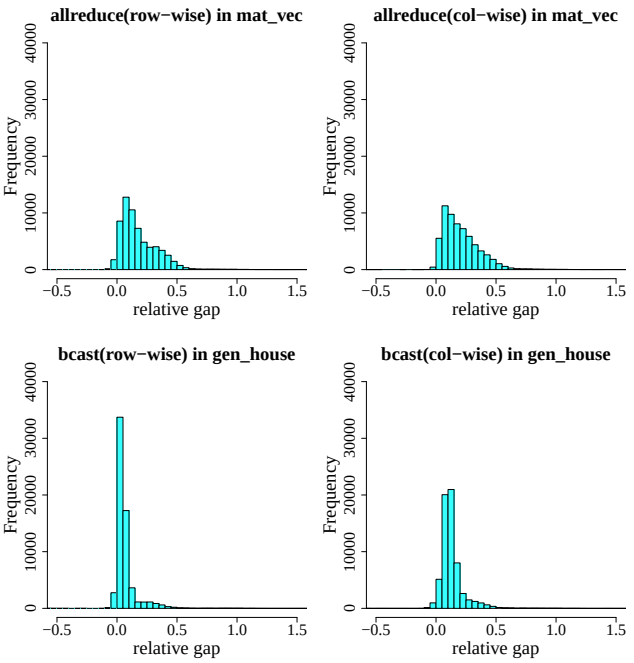

図 5 図 4 における各マーカーの直線からのずれの相対値（relative gap）のヒストグラム（区間は 0.05 きざみ）

4. 「京」における三重対角化プログラムの性能モデリングとその評価

本節では，実際に「京」コンピュータにおける Eigen-Trd の性能モデリングを行い，モデルによる予測結果と実測結果とを比較して評価する．なお，今回は並列数 P は平方数で，ノードは $\sqrt{P} \times \sqrt{P}$ の二次元形状で確保された場合のみを扱う．

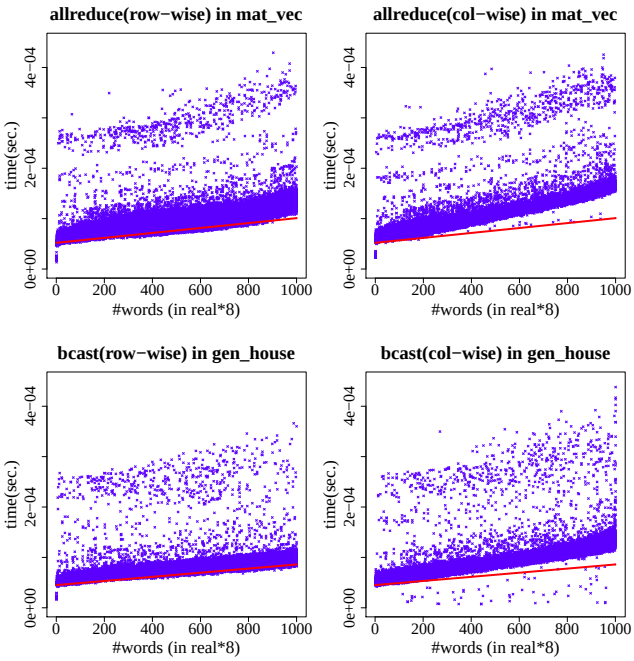

図 6 Eigen-Trd における MPI 通信の挙動とベンチマークにおける挙動の比較（青：4096 ノード（二次元形状ではない）で Eigen-Trd を実行した際の各 MPI 通信の実行時間，赤の線：64 ノードで MPI のベンチマークを実行した結果を最小二乗近似した直線）

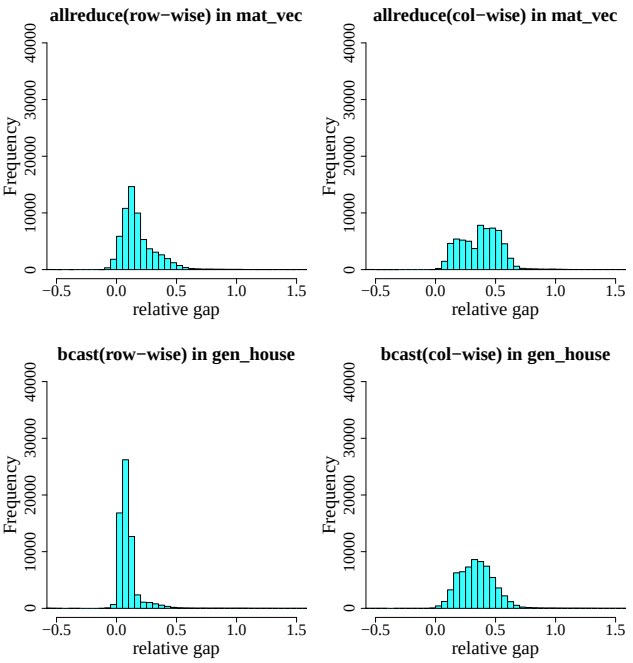

図 7 図 6 における各マーカーの直線からのずれの相対値（relative gap）のヒストグラム（区間は 0.05 きざみ）

4.1 性能モデリングの方法

Eigen-Trd のモデリングの方針だが，Eigen-Trd では演算と通信のオーバーラップがなく，データをサイクリック分散していることからプロセス間のロードバランスが著しく不均一になることがないと考え，

$$(\text{並列実行時間}) = (\text{演算時間}) + (\text{通信時間})$$

であると仮定してモデリングを行う．

まず，演算時間については，3.2 節の結果を考慮すると，実効性能を N や P に依存しない定数とするのは無理がありそうである．一方，シングルノードで仮想的に実行させて得られた演算性能はある程度，有用であることが確認できている．また，シングルノードでベンチマークプログラムを実行してデータを取得することは比較的現実的でもあると思われる．そこで，今回は，

$$(\text{演算時間}) = (\text{1 ノードで演算のみを仮想実行した時間})$$

とする．

次に，通信時間については，1 回の MPI 通信の通信時間がデータサイズを x，通信に参加するプロセス数を P として，

$$(\text{1 回の MPI 通信時間}) = a(P) \cdot x + b(P)$$

と書けるとする．ただし，$a(P)$ と $b(P)$ については Allreduce と Bcast で別々に扱う．すると，MPI 通信の合計時間は，

$$\begin{aligned}(\text{合計の MPI 通信時間}) &= \sum_{x_i} \{a(P) \cdot x_i + b(P)\} \\ &= a(P) \cdot (\text{通信データの総量}) \\ &\quad + b(P) \cdot (\text{通信回数})\end{aligned}$$

と書くことができる．ここで，通信の発行回数と通信データ量は既知である．一方，$a(P)$ と $b(P)$ の値は，Allreduce と Bcast のベンチマーク実行結果から決定することにする．

以上の内容をまとめると，行列サイズが N，並列数が P の場合の実行時間を予測する手順は，

(1) $\sqrt{P}$ ノードで MPI_Allreduce と MPI_Bcast のベンチマークを行い，その結果から，それぞれについて $a(\sqrt{P})$ と $b(\sqrt{P})$ の値を算出する．
(2) 1 ノード上で行列サイズ N，並列数 P の場合と同じ演算のみを仮想的に行うベンチマークを実行し，演算時間を測定する．
(3) 1 で算出した値と既知である通信回数と通信データ量から通信時間を算出する．
(4) 2 で測定した演算時間と 3 で算出した通信時間を足すことで，並列実行時間とする．

上記の手順のうち，1 については N に依存していないので，各 P について一度だけ実行すればよい．また，「京」に関して言えば，$\sqrt{P}$ は高々数百程度なので，比較的簡単に実行することができると思われる．一方，2 については，予測対象とする P がある程度大きいこと場合は，[*7] N が極端に大きくなければ，1 ノード当たりの演算量はそこまで大きくならない．したがって，2 を実行するコストが問題になることはないと考えても構わない．以上より，上記の方法により並列実行時間を予測するのは十分に現実的であると判断できる．

[*7] 今回は並列数が多い場合の予測が目的である．

4.2 予測結果の評価

今回対象とした行列サイズは $N = 32000, 64000, 100000$ の 3 つで，並列数は，予測については $P = 256, 1024, 4096, 16384, 65536$，実測については $P = 64, 256, 1024, 4096$ とした．[*8]

まず，前節で述べた手順の 1 により得られた MPI 通信に関するモデルのパラメータは**表 4** にようになった．なお，「京」の MPI ライブラリではデータが 1 個の場合は特別な通信方法を採用している [12] ため，別に扱っている．この値を用いて，実際に Eigen-Trd の実行時間を予測した結果と実測した結果を比較したグラフを**図 8** に示す．

演算時間については，mat_vec と mat_mat は十分良い予測になっていることが確認できる．これは，図 3 にあるように，並列実行時と仮想的にシングルノードで実行した時の演算性能の挙動が非常に近いからである．逆に，この挙動の差が大きい gen_house については，予測と実測のずれが大きくなってしまっている．

一方，通信時間については，どれも比較的良い精度の予測になっていると判断できる．これは，今回の実験ではノードを二次元形状で確保しているため，通信の衝突が小さく，$\sqrt{P}$ のノードで行ったベンチマークの挙動とのずれがあまり生じなかったからだといえる．なお，mat_mat 内では通信は行っていない．

最後に，Eigen-Trd 全体の実行時間については，グラフより良い精度で予測できていることが分かる．演算時間と通信時間に分けてみると予測がずれている部分もあるが，全体の実行時間の大半を占めるのが通信時間や演算量の多い mat_vec や mat_mat の演算時間であるため，合計時間の予測としては，gen_house の演算時間などにおける予測のずれがあまり大きな影響を与えなかったといえる．なお，今回の精度で全体の実行時間や主要な処理の実行時間を予測することができれば，並列数を増やした際のスケールの様子やボトルネック箇所の調査などに，モデリングの結果を活用することができるのではないかと思う．

5. 密行列計算プログラムの性能モデリングに向けた検討

本節では，これまでに述べた EigenTrd のモデリングの結果を踏まえて，密行列計算プログラムの性能モデリング全般に向けた検討を行う．

全体的には，今回性能モデリングを行った三重対角化プ

[*8] ただし，実際は 65538=256x256 の形状では「京」の上では確保できない．

表 4 MPI のベンチマークから求めた MPI 通信時間のモデルパラメータ（Allreduce の b のカッコ内の値はデータサイズが 1 の場合の値）

		Allreduce		Bcast	
P	$\sqrt{P}$	$a(P)$	$b(P)$	$a(P)$	$b(P)$
256	16	2.8E-8	3.2E-5 (1.2E-5)	2.1E-8	3.1E-5
1024	32	3.9E-8	3.8E-5 (1.2E-5)	2.5E-8	4.0E-5
4096	64	4.9E-8	5.3E-5 (1.4E-5)	3.5E-8	4.7E-5
16384	128	6.1E-8	6.4E-5 (1.5E-5)	3.5E-8	6.0E-5
65536	256	7.6E-8	7.5E-5 (1.7E-5)	4.7E-8	6.6E-5

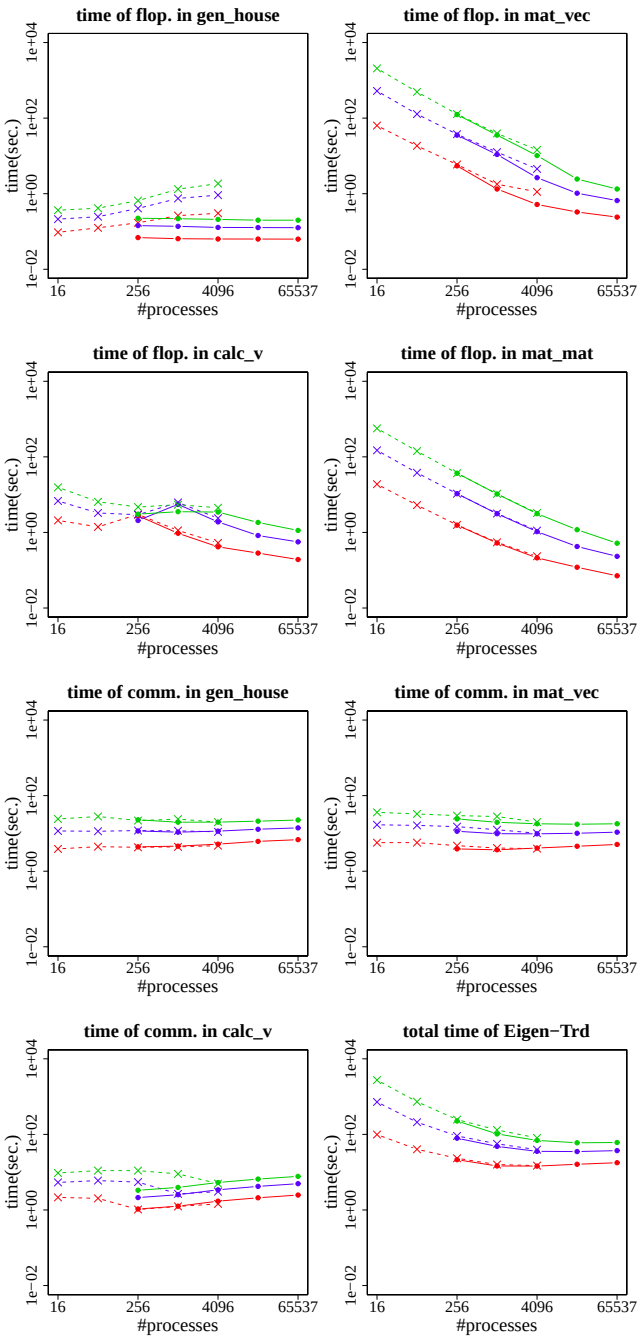

図 8 モデルによる予測結果と実測結果の比較：×が実測，●が予測の値．赤が $N = 32000$，青が $N = 64000$，緑が $N = 100000$ の結果．

ログラム Eigen-Trd における処理の流れやデータ分散方法，実装方法は LU 分解や QR 分解をはじめとする一般的な密行列計算プログラムに共通する部分が多々ある．そのため，計算機環境に関する条件が比較的近ければ，今回と同じような方針でモデリングを行うことができるのではないかと考えている．

以下，細かい部分に関して気がついた点を挙げる．

- 性能を測定する際，特に内部の処理ごとの時間を測定する場合，最内のループ内でバリアやタイマーをコールすることになるが，演算が内積などの場合には 1 回の演算時間が非常に短いので，バリアなどのコストの扱いに注意が必要になる．特に，並列数が大きくなると演算時間が減り，一方でバリアのコストは増えるため，余計に注意が必要となる．
- 密行列計算プログラム（特に直接法）では，計算過程の値に関係なく処理をすすめることができる（途中で破綻することがあまりない）ので，MPI 通信部分を取り除き，シングルノード上で演算のみを仮想的に実行して演算性能を取得する，という方法は幅広く使える可能性がある．
- 行列の形状が特殊（例えば，非常に縦長）でない場合，データを二次元分散することが多く，その場合はプロセスも二次元グリッドに配置することが一般的となる．その場合，今回と同じように，全プロセスが参加する通信はあまり発生せず，二次元グリッドのどちらかの軸方向に閉じた通信を並列に行うことが多くなる．したがって，本稿で示したように，ネットワークの形状次第では効率的に MPI の通信モデルを構築することが可能になり，逆に，形状によっては通信の衝突まで考慮したモデルが必要となる可能性が高い．

6. おわりに

本稿では，これまでに「京」コンピュータの上で行ってきた行列の三重対角化プログラムの挙動の調査や性能モデリングの結果の報告を行い，その結果を踏まえて，「京」やその先のポストペタスケールのシステムにおける密行列計算プログラムの性能モデリングについての検討を行った．

プログラムの挙動を調査した結果，

- 時間測定のための同期やタイマーのコストを適切に処理することが必要である．
- シングルノード上で仮想的にプログラムを実行させて演算性能を測定するのは有効な手法の一つになり得る．
- ノードの形状によっては，MPI のベンチマークに必要なプロセス数を大幅に削減できる可能性がある．

といったことが明らかになった．また，この結果を踏まえて行った三重対角化のモデリングでは，合計時間及び主要部分の実行時間が良い精度で予測できていることが確認できた．そして，これらの結果として得られた知見は，三重

対角化に限らずに，一般の密行列計算プログラムの性能モデリングに役立つことが期待される．

一方，今後の課題としては，

- 他の密行列計算プログラムの性能モデリング
- 「京」あるいは FX10 とは異なる特徴を持ったシステムでの調査
- 他のモデリング手法（例えば，実測データを一切使わずにスペック値だけから予測する）との比較

といったことが挙げられる．加えて，モデリングの結果を利用した

- ポストペタのマシン上での性能の見積もり
- プログラムのチューニング（可能なら手動ではなく自動）

なども今後の展開として考えられる．

謝辞　日頃から有益なご議論をいただいている，神戸大学大学院システム情報学研究科の山本研究室と横川研究室の皆様，下記 CREST プロジェクトのメンバーの皆様，自動チューニング研究会の皆様に深く感謝を申し上げます．本研究の一部は 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の技術の創出）および東京大学情報基盤センターの「若手・女性利用者推薦」の援助を受けています．また，本論文の結果の一部は，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/wp-content/uploads/2012/03/hpci-roadmap.pdf.

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

[3] Bosilca, G., Bouteiller, A., Danalis, A., Faverge, M., Haidar, A., Herault, T., Kurzak, J., Langou, J., Lemarinier, P., Ltaief, H., Luszczek, P., Yarkhan, A. and Dongarra, J. J.: Distibuted Dense Numerical Linear Algebra Algorithms on massively parallel architectures: DPLASMA, *Proceedings of the 25th IEEE International Symposium on Parallel & Distributed Processing Workshops and Phd Forum (IPDPSW'11), PDSEC 2011*, pp. 1432–1441 (2011).

[4] Auckenthaler, T. and Blum, V. and Bungartz, H. -J. and Huckle, T. and Johanni, R. and Krämer, L. and Lang, B. and Lederer, H. and Willems, P. R.: Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations, *Parallel Comput.*, Vol. 37, No. 12, pp. 783–794 (2011).

[5] Imamura, T., Yamada, S. and Machida, M.: Development of a high performance eigensolver on the peta-scale next generation supercomputer system, *in NUCLEAR SCIENCE and TECHNOLOGY*, pp. 643–650 (2011).

[6] 今村俊幸，山田　進，町田昌彦：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).

[7] Dackland, K. and Kågström, B.: A Hierarchical Approach for Performance Analysis of ScaLAPACK-Based Routines Using the Distributed Linear Algebra Machine, *Proceedings of the Third International Workshop on Applied Parallel Computing, Industrial Computation and Optimization*, PARA '96, Springer-Verlag, pp. 186–195 (1996).

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

[9] Demmel, J., Grigori, L., Hoemmen, M. and Langou, J.: Communication-optimal Parallel and Sequential QR and LU Factorizations, *SIAM J. Sci. Comput.*, Vol. 34, No. 1, pp. 206–239 (2012).

[10] Hammarling, S. J., Sorensen, D. C., Dongarra, J. J. and Dongarra, J. J.: Block Reduction of Matrices to Condensed Forms for Eigenvalue Computations, *J. Comput. Appl. Math*, Vol. 27, pp. 215–227 (1987).

[11] 高瀬　亮，横川三津夫（編）：特集：スーパーコンピュータ「京」，情報処理，Vol. 53, No. 8 (2012).

[12] 安島雄一郎，井上智宏，平本新哉，清水俊幸：スーパーコンピュータ「京」のインターコネクト Tofu，雑誌 FUJITSU 2012-5 月号，Vol. 63, No. 3, pp. 260–264 (2012).