Vol.2012-HPC-135 No.19
2012/8/2

マルチコアを考慮した通信隠蔽手法の自動チューニング機能付き高性能固有値ソルバの開発

今村俊幸[†1,†2,†4] 吉田剛啓[†1] 田村遼也[†1]
近藤大貴[†1] 山田進[†3,†4] 町田昌彦[†3,†4]

京コンピュータ向けに開発された固有値ソルバ Eigen-K について，マルチコアの並列特性を考慮した通信ならびに計算最適化について議論する．先行研究では，分配則を利用した集団通信関数削減，冗長な集団通信部分の自動チューニング手法を考慮してきたが，本研究はこれらに加えて通信制御スレッドを導入することによる，ノンブロッキング集団通信に等価な機能を実現する．ノンブロッキング集団通信の制御を自動チューニングにより，異なる並列計算の状況で高いスケーラビリティを達成することができる．本報告では，これらについて FX10 での試験結果を報告する．

High Performance eigenvalue solver accelerated with a auto-tunner mechanism concering with multicore communication hiding

Toshiyuki Imamura,[†1,†2,†4] Takehiro Yoshida,[†1]
Ryoya Tamura,[†1] Hiroki Kondo,[†1]
Susumu Yamada[†3,†4] and Masahiko Machida[†3,†4]

We have recently introduced a communication and computation hiding optimization technique for the eigenvalue solver Eigen-K developed for K computer. In prior works, we focused on the reduction of collective communication by the distributive law and auto-tuning of redundant computation. The present work introduces an auto-tuning technique in an equivalent operation to asynchronous collective communication, which is realized by an expense of a thread dedicating itself to the communication control. This yields very good scalability on various parallel computer environments. This paper reports the latest results on a FX10 supercomputer system.

1. はじめに

「京」を始め Sequoia などのトップクラスのスパコンはおよそ 100 万コアを有している[1)]．エキサスケールスパコンでは現在よりもさらにコア数も多くなり，更に階層性も深くなり，より並列計算の複雑化が進むと予想される．この複雑化によりソフトウェアの設計も複雑化しており，数値計算ライブラリも大規模計算機を考慮して開発する必要がある．

我々が開発を進めてきた超並列固有値ソルバ Eigen-K[7)] でも，10000 コアを超える環境での実用試験と性能評価が進んできている，この過程で，所謂アムダールの法則がより顕著に体感できる状況に来ている．アムダールの法則は，「プログラム中の非並列実行箇所の割合 f があれば，無限大の並列計算機資源を導入した際の速度向上の上限が $1/f$ となる」というものである．並列化可能な箇所は $1/p$ の部分で p が非常に大きくなることで，非並列化部分に対して相対的に 0 とすることができる．Eigen-K における，非並列化部分の内訳は，1) 冗長計算部分，2) 集団通信，3) データ再分散 などである．

先行研究[2)] では，1) と 2) 特に集団的に実施される冗長計算を更に小規模計算に分割し並列計算することでトータルの計算効率を高める試みを実施した．一般に，小規模問題では計算を分割し並列計算を行った後に，その結果を集めて何らかの処理をする際の集団通信のコストが計算時間よりも優位となってしまう．そのため，多くの場合は並列計算を選択せずに，個々のプロセスが同一の計算を実施する冗長計算を行うことで通信コストを抑えている．先行研究では，プロセスグループを可変にすることで，並列計算と冗長計算の切り替えを判断し自動で両者を切り替える自動チューニング手法を組み込んだ．報告では十分なプロセス数での実験ではなかったために，特定グループ数分割が最良の結果を示した．自動チューニングによる最適化もそれを再現したが，オーバヘッドもあるため僅かながら最速とは言えなかった．しかしながら，集団通信の事前コスト解析ではグループ分割の変化による効果は予想できる．自動チューニングによる通信最適化は高並列計算環境では，今後その効果が十分

†1 電気通信大学大学院 情報理工学研究科
Informatics and Engineering, the University of Electro-Communications
†2 理化学研究所 計算科学研究機構
Advanced Institute of Computational Science, RIKEN
†3 日本原子力研究開発機構
Japan Atomic Energy Agency
†4 戦略的創造研究推進事業 CREST(科学技術振興機構 JST)
Core Research for Evolutional Science and Technology(Japan Science and Technology Agency)

1: **for** i:=1 to N step M **do**
2: $U \leftarrow [u_i, u_{i+1}, \ldots, u_{i+M-1}]$.
3: $S \leftarrow \mathrm{diag}^{-1}(\beta_i, \beta_{i+1}, \ldots, \beta_{i+M-1}) + \mathrm{Lower}(U^T U)$.
4: $Z \leftarrow U^T X$.
4: $V \leftarrow US^{-1}$.
5: $X \leftarrow X - VZ$
6: **end for**

図 1 Householde 逆変換
Fig. 1 Householder back-transformation

期待できるものである.

一方, アルゴリズムによっては, データ依存関係などの理由から通信最適化や他の手法による最適化になじまないことも多い. 場合によってはデータの多重化 (ダブルバッファリング) などを駆使することで, 通信量は削減できないが, 他の計算と非同期的に実行できる場合もある. MPI での集団通信は MPI2 のセマンティックスでは同期的で, データ依存関係を結果として強くしてしまうことが指摘されている. 本研究は, 先行研究[4] で実施されたり, MPI-3.0 ドラフト[3] の中でも提案されている非同期的な集団通信を活用し高並列環境で増加する集団通信の通信オーバーヘッドを積極的に隠蔽する実装を行う. さらに, 自動チューニングにより適切なオーバーラップのレベルを選択しデータ量もしくは計算コストとプロセス数の変化に柔軟に対応する. これにより, 1 万コア並列程度までで非常にスケーラブルな固有値計算の逆変換部分の作成を実施する.

2. Householder 逆変換

本研究で対象となる Householder 逆変換について図 1 に示す. 逆変換アルゴリズムには compat WY representation 手法を用いて, 行列 C の計算に陽な再帰計算手法ではなく三角行列の連立一次方程式に基づく陰的手法を採用する.

本アルゴリズムを並列実装する際には, 2 行目に枢軸ベクトルを行方向にブロードキャスト, 3 行目の $U^T U$ と $U^T X$ の計算に列方向グループ内でオールレデュースを必要とする. 2 つのオールレデュースは 1 つにまとめられるので, for 分の反復の内側で都合ブロードキャストとオールレデュースを 1 回ずつを 1 回転あたりに発行する.

1: $U^s \leftarrow [u_1, u_2, \ldots, u_M]$.
2: **for** i:=1 to N step M **do**
3: **if** $i + M < N$ **then**
4: $U^{1-s} \leftarrow [u_{i+M}, u_{i+M+1}, \ldots, u_{i+2M-1}]$.
5: **end if**
6: $S \leftarrow \mathrm{diag}^{-1}(\beta_i, \beta_{i+1}, \ldots, \beta_{i+M-1}) + \mathrm{Lower}(U^{sT} U^s)$.
7: $Z \leftarrow U^{sT} X$.
8: $V \leftarrow U^s S^{-1}$.
9: $X \leftarrow X - VZ$
10: $s \leftarrow 1 - s$
11: **end for**

図 2 Householde 逆変換 (ダブルバッファ版)
Fig. 2 double-buffered Householder back-transformation

3. 非同期的な集団通信最適化

図 1 の Householder 逆変換では, 枢軸ベクトルが入力データとして予め保持されている. 従って, 十分な量のワークベクトルが確保できるのであれば, 2 行目のブロードキャスト操作は先行して実施することができる. バッファを 2 重化したダブルバッファ方式で図 1 を書き直すと図 2 とできる.

図中の U^s の表記で s は odd-even バッファの切替え添字を表している. データの依存関係から, 3〜5 行目の先行するブロードキャスト操作は 10 行目までの間であれば何れの行にも置くことができる. また, ブロードキャストに使用するバッファは計算に登場しない. これは, 適当な箇所でブロードキャスト操作を開始して, 計算の裏側で実施することを意味している. 10 行目のバッファ切り替えまでにブロードバンドが終了すればよい. しかしながら, 現在の MPI では集団通信は同時に 2 種類を発行することが認められていない. ブロードキャスト操作の開始位置は 7 行目の演算の直後以降に限定される. 9 行目の計算は計算量が多い部分であるので, ブロードキャスト操作と重ね合わせることは十分に可能である. 一方, 非同期な集団通信関数は MPI-3 では議論されてはいるものの, 実装系には存在しない. 非同期集団通信を利用するには独自に等価な機能を実装する必要がある. 我々は, マルチスレッド計算を前提とした並列プログラムを作成しており, 例えばマスタースレッドをブロードキャストに専念させ, それ以外のスレッドに計算を並行的に実行させることが可能である.

1: **do parallel** in a thread fashion
2: **if** ismaster() **then**
3: $U^s \leftarrow [u_1, u_2, \ldots, u_M]$.
4: **end if**
5: **barrier**
6: **for** i:=1 to N step M **do**
7: $S \leftarrow \mathrm{diag}^{-1}(\beta_i, \beta_{i+1}, \ldots, \beta_{i+M-1}) + \mathrm{Lower}({U^s}^T U^s)$.
8: $Z \leftarrow {U^s}^T X$.
9: $V \leftarrow U^s S^{-1}$.
10: **if** ismaster() **then**
11: **if** $i + M < N$ **then**
12: $U^{1-s} \leftarrow [u_{i+M}, u_{i+M+1}, \ldots, u_{i+2M-1}]$.
13: **end if**
14: **else**
15: $X \leftarrow X - VZ$
16: **endif**
17: **barrier**
18: $s \leftarrow 1 - s$
19: **end for**
20: **end do parallel**

図 3 Householde 逆変換 (非同期ブロードキャスト)
Fig. 3 Householder back-transformation with asynchronous broadcast

この様な実装方法により, 非同期的なブロードキャスト操作を実現する.

これに似た実装は先行研究 4), 5), 6) で行われてきており, システムや MPI 実装系に大きく依存しない非同期通信もしくはノンブロッキング通信による実装方法である.

4. アルゴリズム改良について

ダブルバッファ+非同期ブロードキャストによる通信の動作概要は図??の様になる. 上下の矢印が個々のスレッドに対応していると見る. 最左のスレッドがブロードキャスト操作を担当し, スレッドチーム全体で非同期通信を実現させている.

図 4 では, 通信や計算のコストを矢印の長さで表現している. 図 4 は, ブロードキャスト

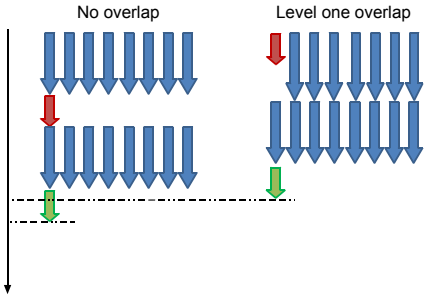

図 4 通信オーバーラップの概要 (レベル 1)
Fig. 4 Schematic diagram of communication overlapping (level 1)

のコストは計算のコストよりも小さい状況であり通信を計算の裏に隠蔽することが可能である. 通信の隠蔽により僅かであるが実行時間短縮がなされることがわかる.

一方, 図 5 は, ブロードキャストのコストの方が計算のコストよりも大きく, 通信を単純に計算の裏に隠蔽できない例を示している. 図の中央のものでは, 通信が終了するまで次のブロックに制御が移らないために全体の稼働率が低くなっている. 右のものは更に後続するブロックと通信をオーバーラップさせることで, 稼働率を上げ結果として中央のものよりも実行時間短縮につながっている. この様に, いくつの計算ブロックと通信をオーバーラップさせるかによってオーバーラップのレベルを 0(図 5 の左), 1(図 5 の中央) ないしは 2(図 5 の右) とする. なお, オーバーラップレベル 2 の実装を考えるとき, 各ブロックで利用するバッファが異なるためバッファは 3 本必要となる. つまり, トリプルバッファ方式での実装が必要である.

図 5 の議論から, ブロードキャストのコスト, 計算ブロックのコストに応じて適切なオーバーラップレベルが決定でき, それを選択することでシステムの実行効率を高め, 実行時間を短縮することができる.

5. 自動チューニングについて

5.1 オーバーラップレベルのモデル化

Householder 逆変換における非同期ブロードキャストと計算のオーバーラップはレベル 0, 1, 2 の段階があった. また, 前節では説明しなかったが通信に専念するスレッドを導入することで計算能力は若干下がるため, 計算部分の所要コストは増加する. これらのトレードオ

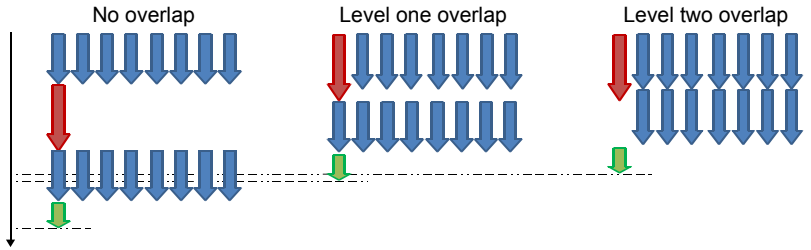

図 5 通信オーバーラップの概要 (レベル 2)
Fig. 5 Schematic diagram of communication overlapping (level 2)

フを考慮し, 最適なオーバーラップレベルを決定しなくてはならない.

ブロードキャストと計算部分の処理時間がそれぞれ既知であると仮定してレベル 0,1,2 のコストをモデル化してみる. まず, ブロードキャストのコストを T_B, レベル 1 で重ねる計算部分を全スレッドで処理した場合のコストを T_1, レベル 2 で重ねる計算部分のそれを T_2 と書く. また, 総スレッド数を d と置く.

レベル 0, つまり, オーバーラップをしない単純な計算でのコスト C_0 は

$$C_0 := T_B + T_1 + T_2 \tag{1}$$

となる. レベル 1 のオーバーラップでは

$$C_1 := \max\left\{T_B, \frac{d}{d-1}T_1\right\} + T_2, \tag{2}$$

レベル 2 のオーバーラップでは

$$C_2 := \max\left\{T_B, \frac{d}{d-1}(T_1 + T_2)\right\} \tag{3}$$

となる. したがって,

$$l := \mathrm{argmin}_{k=0,1,2} C_k \tag{4}$$

とする l を決定できれば全体の処理時間を最小化できることになる.

次に, 実行中に時間測定関数等を用いて部分ごとの処理時間を計測しながら l を決定することを考える. 処理中は, オーバーラップレベルが 0,1,2 のいずれかであり, レベルが 1 もしくは 2 で計算が重ねられた場合は, 上で用いた T_1 や T_2 ではない. $d-1$ スレッドによる処理が d スレッドで処理される時間を, もっとも単純な推定である $(d-1)/d$ 倍を乗じることで得るとしよう. そうすると, 式 (4) に必要な C_* はオーバーラップレベルに応じて次のように計算される. ここで, t_* は直前ステップでの測定時間, また, $f = d/(d-1)$ である.

$$C_0 := \begin{cases} t_B + t_1 + t_2 & \text{level 0} \\ t_B + t_1/f + t_2 & \text{level 1} \\ t_B + (t_1 + t_2)/f & \text{level 2} \end{cases} \tag{5}$$

$$C_1 := \begin{cases} \min\{t_B, t_1 f\} + t_2 & \text{level 0} \\ \min\{t_B, t_1\} + t_2 & \text{level 1} \\ \min\{t_B, t_1\} + t_2/f & \text{level 2} \end{cases} \tag{6}$$

$$C_2 := \begin{cases} \min\{t_B, (t_1 + t_2)f\} & \text{level 0} \\ \min\{t_B, t_1 + t_2 f\} & \text{level 1} \\ \min\{t_B, t_1 + t_2\} & \text{level 2} \end{cases} \tag{7}$$

これらをもとに, オーバーラップレベルを切り替えていく. 統計学的には直前のブロックの処理時間で代用するのは適当ではないが, 実用上はこれでも十分といえる. 本研究では統計上の技法は取り入れず, この手法で処理時間を推定する.

5.2 並列処理における注意事項

前節で必要な時間測定はローカルな操作のため最少処理時間の算出もローカルとなり選択すべきオーバーラップレベルがプロセスごとに異なるという不整合が生じる可能性がある. そのため, 式 (4) によって得られたローカルなオーバーラップレベル候補に対して,

(1) 最小値
(2) 最大値
(3) 最多候補

による値 (採用基準) を算出してグローバルなオーバーラップレベルとする. これらの解釈は, (1) の最小値はオーバーラップに消極的な判定, (2) の最大値はオーバーラップに積極的な判定, (3) は全体の揺れを考慮した平均的なレベルを採用するといえる.

さらに, オーバーラップレベルの切り替え判定には補助的な集団通信関数の呼び出しが必要となるため, ループの反復毎に実行することはコスト高である. したがって, 切り替え判定のサイクルを適当に設定し, 連続して同じオーバーラップレベルが選択されれば, 判定サイクルを 2 倍にしていく δ サイクル方式をとる. オーバーラップレベルが切り替われば, 判定サイクルを最少サイクルに設定し直せばよい.

6. 実験結果

6.1 実験環境

本実験では東京大学情報基盤センターの FX10 システムを使用した．FX10 のマシン構成は**表 1** の通りである．なお, 本実験は FX10 システムの試験運転期間中に実施されたものであり, 一部の運用仕様は実運用時と異なる可能性がある．FX10 は 1 ノードに 16 コアを搭載しており, 開発中の Householder 逆変換ルーチンは 1 スレッド 1 コアに対応させる．オーバーラップを行う際は, 1 コアが通信専用スレッドに割り当てられ, 残りの 15 コアが計算担当となる．一時的に 15/16=93.75%にまで演算能力が落ちるが, 通信のみを処理中は 0%であるから, その減少分は無視できる (コア数が少ないプロセッサの場合は様相が異なる).

表 1 東大 FX10 システムのマシン構成
Table 1 Hardware specification of FX10 housed at Tokyo University

ノード	プロセッサ数 (コア数): 1(16)
主記憶容量	32GB/node
プロセッサ	CPU: SPARC64 IXfx (1.848GHz)
キャッシュメモリ	L1:32KB/コア, 共有 L2:12MB/プロセッサ
ノード間結合	Tofu(6D-mesh/torus)

6.2 自動チューニングによる性能改善について

表 2 は, 非同期集団通信関数（ブロードキャスト）の自動チューニングの効果を測定したものである．先に示したように, オーバーラップレベルの採用基準には 3 種類有り, TYPE1, TYPE2, TYPE3 としている．また, オーバーラップをしないものも TYPE0 として, 比較材料として載せている．行列サイズ N が小さい場合は, 非同期集団通信関数による自動チューニングの効果ははっきりとしないが, N が十分大きいとその効果も明白である．図 6 から 8 までは, これら表の数値を両対数スケールによりプロットしたグラフである．グラフの系列 1, 系列 2, 系列 3, 系列 4 がそれぞれ採用基準の 0, 1, 2, 3 に対応する．これらを見ることで, 今回実装した非同期集団通信関数による自動チューニング機構による高い効果が認められる．

- 通信オーバーヘッドが隠蔽されることで, 高速化が達成されている．
- 高並列の状況下でも高い並列化効果が認められる．
- オーバーラップの選択基準では性能に有意な差を認められない．

行列サイズ N が大きい場合は, 計算量自身が支配的で通信オーバヘッドも想定的に小さいのであるが, 通信の隠蔽がなされることでより理想的な並列化に近づいている．

表 3 TYPE3 での選択過程 (上段: ステップ, 下段: オーバーラップレベル)
Table 3 Selected level on TYPE3 (Upper: #step, Bottom: level of overlapping)

ノード数											
64	209	465	593	849	1361	2385	4433	8529	16721	33105	65873
	2	1	1	1	1	1	1	1	1	1	1
256	209	465	977	2001	4049	8145	16337	32721	65489		
	2	2	2	2	2	2	2	2	2		
1024	209	465	977	2001	4049	8145	16337	32721	65489		
	2	2	2	2	2	2	2	2	2		

表 3 は, ステップ毎に選択されたオーバーラップレベルを示したものである．使用ノード数が少ないときは, 個々のプロセスの計算量が多く, 通信量が相対的に小さくなるためレベル 1 が選択されている．プロセス数が多くなると, 計算量も減少しかつ通信コストも増大するのでレベル 2 がより選択されるはずである．

本研究での選択基準は直前の実行時間から推定される数値と単純な予測実行時間に基づく．より高度な統計的手法を用いることで, 推定の精度の向上, 推定コストの削減なども期待できる．そのような方向での研究も継続する必要があろう．

6.3 更なる通信隠蔽の可能性

ここまでは, ブロードキャスト関数のみをオーバーラップの対象としてきたが, 非同期的なオールレデュース関数レベルが 2 の場合でも, と効率的なパイプライン実装ができれば, オールレデュース関数も隠蔽可能となる．本実験は試験利用期間ということもあり, 今回は実装まで至っていない．今後の課題としたい．

7. まとめ

京コンピュータ向けに開発された固有値ソルバ Eigen-K の Householder 逆変換について, マルチコアの 1 部のコアを通信に専念させる並列化を行い非同期的な集団通信関数の実装を行うとともに, 通信隠蔽による計算最適化を実施した．ノンブロッキング集団通信の制御を自動チューニングにより実施することで, 異なる並列計算の状況で高いスケーラビリティが期待される実装がなされた．FX10 での試験結果では, 非常に高いスケーラビリティを示し, 高並列実行の状況下でよい成績を示した．

今後の課題は，今回題案したアルゴリズムを京コンピュータやその他各種並列計算機上で実装して性能評価を行うことである．また, 高度な統計手法の採用, オールレデュース関数

表 2 非同期集団通信による実行時間 (TYPE0: オーバーラップなし, TYPE1: 候補 (1) 採用, TYPE2: 候補 (2) 採用, TYPE3: 候補 (3) 採用)
Table 2 Elapsed time with asynchronous collective communication (TYPE0; without overlapping, TYPE1: (1) is selected, TYPE2: (2) is selected, TYPE3: (3) is selected)

	N=20000				N=30000				N=40000			
ノード数	TYPE 0	TYPE 1	TYPE 2	TYPE 3	TYPE 0	TYPE 1	TYPE 2	TYPE 3	TYPE 0	TYPE 1	TYPE 2	TYPE 3
32	5.38	4.95	5.10	4.97	15.17	14.28	14.28	14.28	33.14	31.91	32.58	32.47
64	3.51	3.10	3.11	3.22	9.34	8.45	8.67	8.54	18.73	17.43	17.43	17.43
128	3.05	2.17	2.18	2.18	6.37	4.93	4.93	4.86	12.64	10.61	10.68	10.58
256	2.27	1.80	1.79	1.79	4.35	3.20	3.25	3.25	8.16	6.48	6.48	6.48
512	2.31	1.36	1.37	1.37	3.80	2.76	2.77	2.73	14.29	5.86	5.84	6.29
1024	4.29	1.45	1.43	1.41	4.69	2.79	2.83	2.81	5.64	4.58	4.64	4.53

	N=80000				N=130000			
ノード数	TYPE 0	TYPE 1	TYPE 2	TYPE 3	TYPE 0	TYPE 1	TYPE 2	TYPE 3
32	231.10	233.98	235.76	235.85	938.06	952.98	953.39	—
64	123.26	123.38	124.96	123.36	486.42	498.13	497.84	497.62
128	69.39	65.14	65.51	65.38	259.20	255.55	254.25	253.38
256	41.35	37.07	37.18	37.40	145.07	139.52	139.52	138.60
512	33.87	24.62	24.71	24.64	98.91	77.32	75.70	75.82
1024	19.69	16.26	15.83	15.42	62.28	48.16	45.45	45.36

のオーバーラップの検討, 更に, 先行研究とも組み合わせ, 通信最適化を最大限に実施した Householder 逆変換の最終形を開発したい.

参 考 文 献

1) Top500 ホームページ, `http://www.top500.org/`
2) 近藤 大貴, 吉田 剛啓, 田村 遼也, 今村 俊幸: 自動チューニングによる通信最適化を施した固有値ソルバの開発について, 情報処理学会研究報告, 2012-HPC-133(24), pp. 1–7, 2012-03-19.
3) MPI-forum のホームページより MPI3.0draft が参照できる. `http://meetings.mpi-forum.org/`
4) 今村 俊幸, 山田 進, 町田 昌彦: 大規模 SMP クラスタにおける固有値ライブラリの通信最適化について情報処理学会研究報告, 2006-ARC-167, 2006-HPC-105(20), pp. 37–42, 2006-02-27.
5) Susumu Yamada, Toshiyuki Imamura, Takuma Kano, and Masahiko Machida: High-performance computing for exact numerical approaches to quantum many-body problems on the earth simulator, SC '06 Proceedings of the 2006 ACM/IEEE conference on Supercomputing, Article No. 47, 2006.
6) 山田進, 今村 俊幸, 町田 昌彦 他: 共有分散メモリ型並列計算機における新規通信手法, 日本計算工学会論文集 Vol. 7, pp. 243–252, 2005.
7) 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, No 6 , pp.12–32, 2009.
8) 今村 俊幸: ペタスケール環境での高並列固有値ソルバの開発, 計算工学講演会論文集, Vol.15, No.1, pp.103–106(2010).
9) Toshiyuki Imamura, Susumu Yamada, and Masahiko Machida: 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.

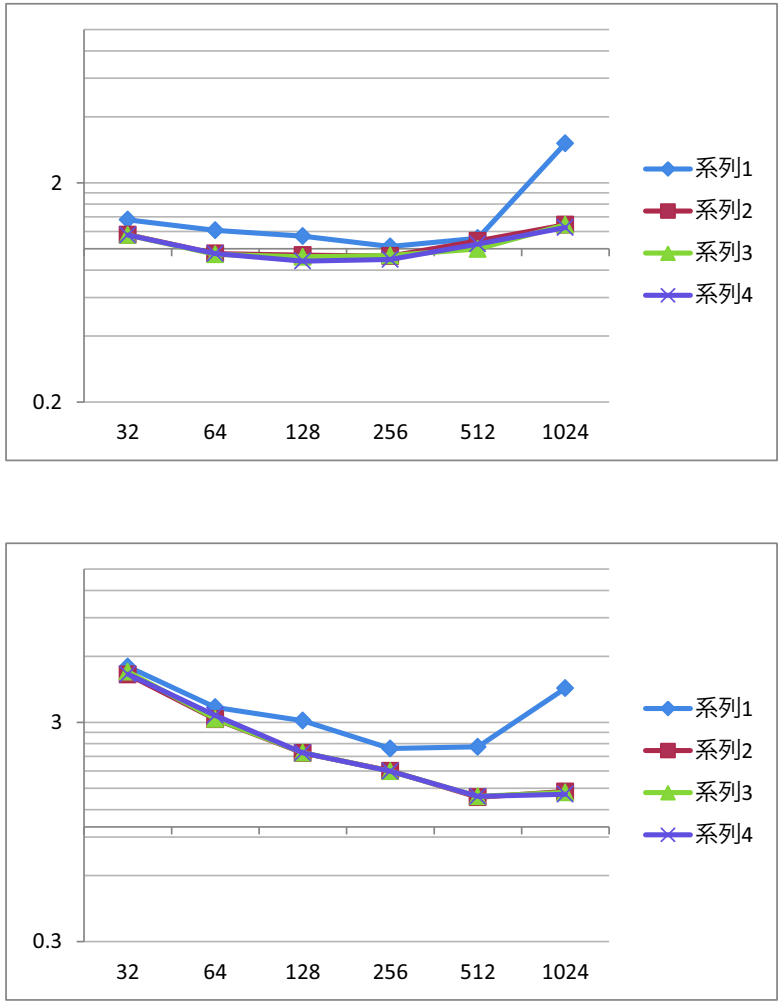

図 6 実験結果 (上: N=10000, 下: N=20000)
Fig. 6 results(Top: N=10000, Bottom: N=20000)

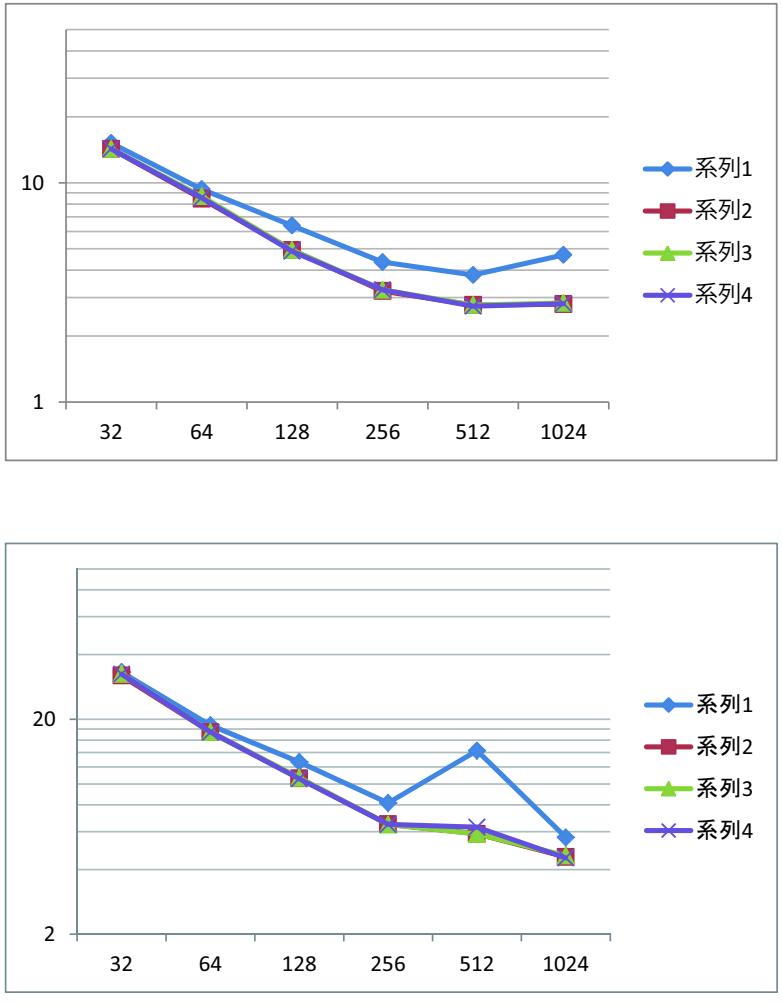

図 7 実験結果 (上: N=30000, 下: N=40000)
Fig. 7 results(Top: N=30000, Bottom: N=40000)

Vol.2012-HPC-135 No.19
2012/8/2

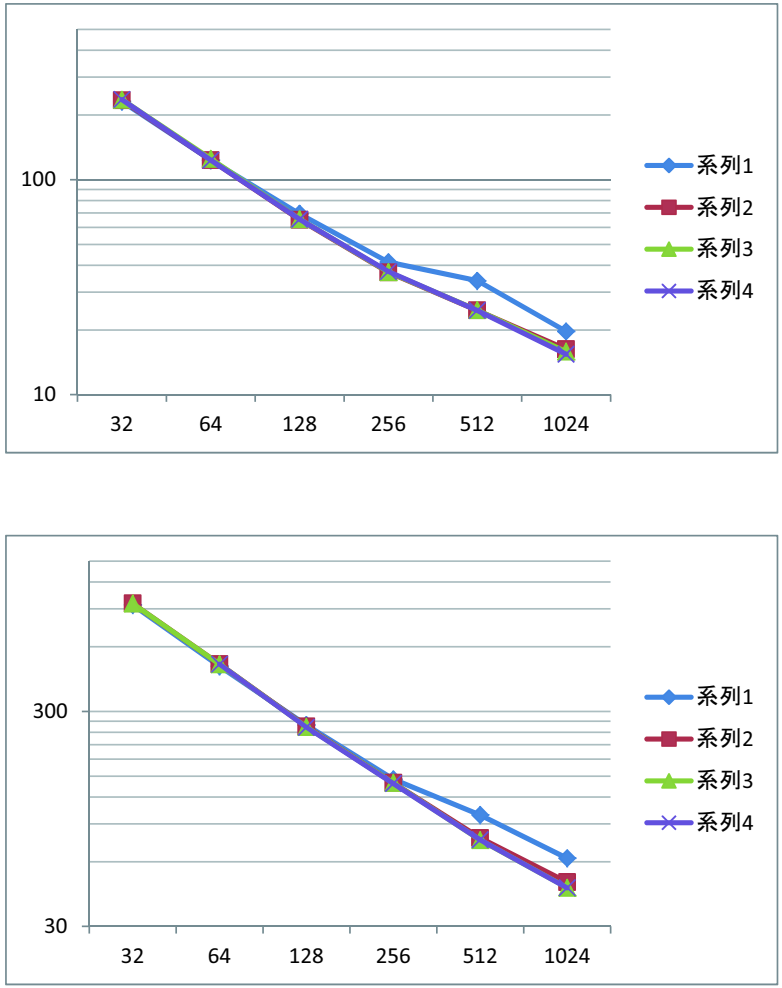

図 8 実験結果 (上: N=80000, 下: N=130000)
Fig. 8 results(Top: N=80000, Bottom: N=130000)