情報処理学会論文誌　コンピューティングシステム　Vol.9　No.3　1–13 (Aug. 2016)

共役勾配法への種々の通信削減手法の適用と評価

熊谷 洋佑[1]　藤井 昭宏[1,a)]　田中 輝雄[1,b)]　深谷 猛[2]　須田 礼仁[3]

受付日 2016年1月5日, 採録日 2016年4月28日

概要：スーパコンピュータの性能はコア数の増加とともに向上している．大規模な線形解法として共役勾配法（CG 法）が広く用いられる．高並列な環境において，内積計算で発生する集団通信が深刻なボトルネックになると指摘されている．近年，Communication-avoiding CG 法の一種として Chebyshev 基底共役勾配法（CBCG 法）が提案されている．本論文では，CBCG 法で現れる集団通信の回数を減らした CBCGR 法を示し，CBCGR 法に対して通信削減手法である Matrix Powers Kernel（MPK）の適用を行った．また，2 次元と 3 次元の Poisson 方程式に対して FX10（oakleaf-fx）スーパコンピュータシステムで最大 1,440 ノードを使用した OpenMP/MPI の Hybrid 並列での計測を行った．2 次元 Poisson 方程式では CBCGR 法および CBCGR-MPK 法が一定の並列数以上で CG 法および CBCG 法よりも高速になり，3 次元 Poisson 方程式では一定の並列数以上で CBCGR 法が高速となった．

キーワード：線形解法，通信削減アルゴリズム，共役勾配法，Matrix Powers Kernel

Application and Evaluation of Various Communication Avoiding Techniques for the Conjugate Gradient Method

Yosuke Kumagai[1]　Akihiro Fujii[1,a)]　Teruo Tanaka[1,b)]　Takeshi Fukaya[2]　Reiji Suda[3]

Received: January 5, 2016, Accepted: April 28, 2016

Abstract: The performance of supercomputers improves as the number of cores increases. The conjugate gradient (CG) method is useful for solving large and sparse linear systems. It has been pointed out that collective communication needed for calculating inner products becomes serious bottleneck when executing the CG method on massively parallel systems. Recently, the Chebyshev basis CG (CBCG) method, a variant of the Communication-avoiding CG method, has been proposed. In this paper, we reduced collective communication of CBCG method (CBCGR) and applied Matrix Powers Kernel (MPK) for CBCGR method. We then measured the execution time of these methods for 2D and 3D Poisson problems using OpenMP/MPI hybrid parallel model on the FX10 (oakleaf-fx) supercomputer system. For the 2D-Poisson problem, the CBCGR and CBCGR-MPK methods are faster than the CG and CBCG methods when the number of processes is sufficiently large. For the 3D-Poisson problem, the CBCGR method is faster than the CG and CBCG methods when the number of processes is sufficeint large.

Keywords: linear solver, communication avoiding algorithm, conjugate gradient method, Matrix Powers Kernel

1. はじめに

近年，スーパコンピュータの性能はコア数の増加とともに向上している．反面，並列に処理を行う際にネットワークで結ばれたノード間での通信が必要となることが一般的である．特にリダクションやブロードキャスト等の集団通信は，ノード数に応じて通信時間が増加するために，高並

[1] 工学院大学
Kogakuin University, Shinjuku, Tokyo 163–8677, Japan
[2] 北海道大学
Hokkaido University, Sapporo, Hokkaido 060–0811, Japan
[3] 東京大学
The University of Tokyo, Bunkyo, Tokyo 113–8656, Japan
[a)] fujii@cc.kogakuin.ac.jp
[b)] teru@cc.kogakuin.ac.jp

列時にボトルネックになることが指摘されている．ハードウェアレベルでの通信時間の削減は物理的に限界があるため，アルゴリズムの観点からの通信時間の削減が必要であり，特に通信回数を削減することによる通信のレイテンシの削減が重要であるといわれている．

一方，正定値対称な行列を係数に持つ連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を解くのに反復解法である共役勾配（CG）法が広く用いられる [1]．CG 法では，1 反復中に 1 回の疎行列ベクトル積（SpMV）と 3 回のベクトル加算（axpy），2 回の内積計算が行われる．係数行列をブロック行分割で分散環境向けに並列化をすると，ベクトルデータを各プロセスに分散して保持することになる．そのため，内積のたびに集団通信（`MPI_AllReduce`）が必要となり，この部分が並列数を増やしていくと深刻なボトルネックとなる．

そこで，内積計算を削減した CG 法が提案されてきた．Chronopoulos らにより CG 法の計算順序を変えることで反復中の 2 回の内積計算に必要な `MPI_AllReduce` をまとめて行う CG 法（C-CG 法）が提案されている [2]．Ghysels らにより C-CG 法のアルゴリズムの改良を行い非同期集団通信向けの Pipelined CG 法が提案されている [3]．また，Chronopoulos らは同論文において CG 法 k 反復分の計算を 1 反復にまとめて計算を行い `MPI_AllReduce` を CG 法の $1/k$ 回にする s-step CG 法，本谷らは k 段飛ばし CG 法 [4] を提案している．しかし，これらの手法は k の値を大きくすると，収束に要する反復回数の増大または発散するといった，数値的に不安定になることが報告されている．Hoemmen の博士論文より，Kyrlov 部分空間を多項式基底でまとめて生成する Communication-avoiding CG（CA-CG）法が提案されている [5]．須田らにより CA-CG 法の一種として Chebyshev 多項式を基底とし，Kyrlov 部分空間をまとめて生成する Chebyshev 基底共役勾配（CBCG）法が提案されている [6], [7]．CA-CG 法と CBCG 法ともに CG 法と同等の反復回数で収束するといった数値的安定性が報告されている．

我々の研究で，京コンピュータおよび FX10 上において FlatMPI 並列での CBCG 法が高並列時に CG 法よりも高速になることが明らかになっている [8], [9]．また，Carson も同様に Hopper において FlatMPI 並列での Newton 多項式を基底とする CA-CG 法が高並列時に CG 法よりも高速になることが報告している [10]．

一方で，通信削減 CG 法に現れる $(A\boldsymbol{r}, A^2\boldsymbol{r}, \cdots, A^k\boldsymbol{r})$ の計算に必要な通信回数を削減する Matrix Powers Kernel（MPK）が Demmel らにより提案されている [11], [12]．MPK では，各プロセスが $A^k\boldsymbol{r}$ の計算に必要な領域を拡張し，保持することで，1 回の一対一通信で計算を可能とするが，プロセス間での重複計算が発生する．また，MPK は共有メモリ環境において，各スレッドが CPU のキャッシュに収まる範囲の計算範囲をブロックに分割することで，通常の SpMV を k 回行うよりも高速になるが，スレッド間での重複計算が発生する．MPK は CBCG 法および CA-CG 法に対して適用でき，さらに通信削減の効果が期待でき，高速化が見込まれる．MPK の性能面に関しては，Dehnavi らは Graphic Processing Unit 上において，複数の一般行列に対して MPK の性能評価を行い，SpMV よりも高速になることを報告している [13]．黒田らは 2 次元と 3 次元の差分問題に対して，Multicolor orderling で計算領域を色分けすることでスレッド間の重複計算をなくし，共有メモリ環境で従来の MPK よりも高速になることを報告している [14]．しかし，MPK および MPK を適用した CBCG 法の高並列環境での性能面についての研究報告は我々の知る限りではない．

本論文では，CBCG 法に対して通信削減アルゴリズムである MPK を適用し，FX10 上における OpenMP/MPI の Hybrid 並列での CBCG 法および MPK の性能について述べる．具体的には，まず，CBCG 法の計算順序を変え，1 反復に発生する `MPI_AllReduce` を 2 回から 1 回にする CBCGR 法について述べ，MPK についてと CBCG 法への適用方法を述べ，演算量と通信回数を示す．実際に CG 法と CBCG 法，CBCGR 法，CBCGR-MPK 法のプログラムを実装して FX10 を最大 1,440 ノードを使用し，2 次元と 3 次元の Poisson 方程式を対象に実行時間を測定した．実行時間をストロングスケーリングで評価した結果，2 次元 Poisson 方程式では，一定の並列数以上において，CBCGR-MPK 法，CBCGR 法，CBCG 法，CG 法の順で高速になることが確認できた．3 次元 Poisson 方程式では，CBCGR 法が一定の並列数以上において，CBCGR 法が高速となったが，CBCGR-MPK 法はプロセス間の重複計算が影響しどの並列数においても遅い結果が得られた．

以下，2 章では CBCG 法のアルゴリズムについて簡単に説明する．次に，3 章で CBCG 法の反復中に現れる `MPI_AllReduce` の回数を削減する CBCGR 法と MPK についてを述べる．また，各手法の演算量と通信回数について比較を示す．4 章で Hybrid 並列での CBCG 法で現れる密行列積と各手法で現れる SpMV の実装方法について述べる．その後，5 章で FX10 上において OpenMP/MPI の Hybrid 並列での性能評価結果を示し，最後に 6 章でまとめと今後の課題について述べる．

2. Chebsyhev 基底共役勾配法

CG 法では，係数行列 A と初期残差 $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$ により作られる Krylov 部分空間を 1 反復に 1 次元ずつ拡大しながら，近似解 $\boldsymbol{x}_i$ を更新する．CG 法のアルゴリズムを Algorithm 1 に示す．詳細については，文献 [1] に委ねる．CG 法のアルゴリズムでは，$A\boldsymbol{p}_i$（4 行目）の計算で SpMV を 1 回，$\boldsymbol{p}_i^T A\boldsymbol{p}_i$（4 行目）と $\boldsymbol{r}_{i+1}^T\boldsymbol{r}_{i+1}$（8 行目）の計算で内積が 2 回行われる．そのため CG 法 1 反復中に 2 回の

Algorithm 1 The CG method

1: $\boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0$
2: $\boldsymbol{p}_0 := \boldsymbol{r}_0$
3: **for** $i = 0, 1, 2, \cdots$ until $\| \boldsymbol{r}_i \|_2 / \| \boldsymbol{r}_0 \|_2 < \epsilon$ **do**
4: Compute $\boldsymbol{p}_i^T A\boldsymbol{p}_i$
5: $\alpha_i := \boldsymbol{r}_i^T\boldsymbol{r}_i / \boldsymbol{p}_i^T A\boldsymbol{p}_i$
6: $\boldsymbol{x}_{i+1} := \boldsymbol{x}_i + \alpha_i\boldsymbol{p}_i$
7: $\boldsymbol{r}_{i+1} := \boldsymbol{r}_i - \alpha_i A\boldsymbol{p}_i$
8: Compute $\boldsymbol{r}_{i+1}^T\boldsymbol{r}_{i+1}$
9: $\beta_i := \boldsymbol{r}_{i+1}^T\boldsymbol{r}_{i+1} / \boldsymbol{r}_i^T\boldsymbol{r}_i$
10: $\boldsymbol{p}_{i+1} := \boldsymbol{r}_{i+1} + \beta_i\boldsymbol{p}_i$
11: **end for**

MPI_AllReduce が発生することとなる．

Chronopoulos らは CG 法の計算順序を変えることで，内積による MPI_AllReduce を 1 回にする CG 法（C-CG 法）を提案している．Algorithm 1 の 4 行目に現れる $\boldsymbol{p}_i^T A\boldsymbol{p}_i$ について式展開を行う．$\boldsymbol{p}_{i+1} = \boldsymbol{r}_{i+1} + \beta_i\boldsymbol{p}_i$ より，

$$\begin{aligned}\boldsymbol{p}_{i+1}^T A\boldsymbol{p}_{i+1} &= (\boldsymbol{r}_{i+1} - \beta_i\boldsymbol{p}_i)^T A\boldsymbol{p}_{i+1} \\ &= \boldsymbol{r}_{i+1}^T A\boldsymbol{p}_{i+1} - \beta_i\boldsymbol{p}_i^T A\boldsymbol{p}_{i+1} \\ &= \boldsymbol{r}_{i+1}^T A\boldsymbol{p}_{i+1} \\ &= \boldsymbol{r}_{i+1}^T(A\boldsymbol{r}_{i+1} + \beta_i A\boldsymbol{p}_i) \\ &= \boldsymbol{r}_{i+1}^T A\boldsymbol{r}_{i+1} + \beta_i\boldsymbol{r}_{i+1}^T A\boldsymbol{p}_i \end{aligned} \quad (1)$$

また，

$$\begin{aligned}&\boldsymbol{r}_{i+1} = \boldsymbol{r}_i - \alpha_i A\boldsymbol{p}_i \\ &\alpha_i A\boldsymbol{p}_i = \boldsymbol{r}_i - \boldsymbol{r}_{i+1} \\ &A\boldsymbol{p}_i = (\boldsymbol{r}_i - \boldsymbol{r}_{i+1})/\alpha_i \end{aligned} \quad (2)$$

であるため，

$$\begin{aligned}\boldsymbol{r}_{i+1}^T A\boldsymbol{p}_i &= \boldsymbol{r}_{i+1}^T(\boldsymbol{r}_i - \boldsymbol{r}_{i-1})/\alpha_i \\ &= (\boldsymbol{r}_{i+1}^T\boldsymbol{r}_i - \boldsymbol{r}_{i+1}^T\boldsymbol{r}_{i+1})/\alpha_i \\ &= -\boldsymbol{r}_{i+1}^T\boldsymbol{r}_{i+1}/\alpha_i \end{aligned} \quad (3)$$

となり，式 (1) に代入すると，

$$\boldsymbol{p}_{i+1}^T A\boldsymbol{p}_{i+1} = \boldsymbol{r}_{i+1}^T A\boldsymbol{r}_{i+1} - (\beta_i\boldsymbol{r}_{i+1}^T\boldsymbol{r}_{i+1})/\alpha_i \quad (4)$$

となり，$\boldsymbol{r}_{i+1}^T\boldsymbol{r}_{i+1}$ と $\boldsymbol{r}_{i+1}^T A\boldsymbol{r}_{i+1}$ の内積計算が必要となるが，この 2 つの計算は同時に行うことができる．ここで，$A\boldsymbol{r}_{i+1}$ の計算で SpMV が 1 回行われ，$A\boldsymbol{p}_i$ の計算も必要となるが，Algorithm 1 の 10 行目に対して両辺に A をかけることで $A\boldsymbol{r}_i$ を用いて $A\boldsymbol{p}_i$ を算出することが可能である．C-CG 法のアルゴリズムを Algorithm 2 に示す．C-CG 法では，CG 法と比較し，6 行目のベクトル加算が 1 回増えるが，9 行目で 2 個のスカラー値を MPI_Allreduce1 回で行うことができる．詳細に関しては文献 [2] に委ねる．

CBCG 法では，Kyrlov 部分空間を 1 反復に k 次元ずつ拡大しながら，近似解を更新することを目指す．CBCG 法のアルゴリズムを Algorithm 3 に示す．Algorithm 3 の 2 行目と 9 行目が Krylov 部分空間を生成する部分であり，ここで k 回の SpMV が行われる．Chebyshev 多項式を基底とした Krylov 部分空間の生成のアルゴリズムを Algorithm 4 に示す．Chebyshev の三項漸化式により $T(A)$ に対して係数行列 A の最小固有値と最大固有値を与えることで，$T(A)$ の最大固有値の絶対値を小さくなるため数値的に安定となる．また，Algorithm 4 に現れる λ_{min} と λ_{max} はそれぞれ係数行列 A の最小および最大固有値となる．Algorithm 3 に現れる B_i と $\boldsymbol{a_i}$ は k 次元の行列とベクトルであり，非常に小さなものであるため，QR 分解による最小二乗法で解いている．CBCG 法の詳細については，文献 [5], [7] に委ねる．

Algorithm 2 C-CG method

1: $\boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0$
2: $\alpha_0 := \boldsymbol{r}_0^T\boldsymbol{r}_0 / \boldsymbol{r}_0^T A\boldsymbol{r}_0$
3: $\beta_{-1} := 0$
4: **for** $i = 0, 1, 2, \cdots$ until $\| \boldsymbol{r}_i \|_2 / \| \boldsymbol{r}_0 \|_2 < \epsilon$ **do**
5: $\boldsymbol{p}_i := \boldsymbol{r}_i + \beta_i\boldsymbol{p}_{i-1}$
6: $A\boldsymbol{p}_i := A\boldsymbol{r}_i + \beta_i A\boldsymbol{p}_{i-1}$
7: $\boldsymbol{x}_{i+1} := \boldsymbol{x}_i + \alpha_i\boldsymbol{p}_i$
8: $\boldsymbol{r}_{i+1} := \boldsymbol{r}_i + \alpha_i A\boldsymbol{p}_i$
9: Compute $\boldsymbol{r}_{i+1}^T\boldsymbol{r}_{i+1}, \boldsymbol{r}_{i+1}^T A\boldsymbol{r}_{i+1}$
10: $\beta_i := \boldsymbol{r}_{i+1}^T\boldsymbol{r}_{i+1} / \boldsymbol{r}_i^T\boldsymbol{r}_i$
11: $\alpha_{i+1} := \boldsymbol{r}_{i+1}^T\boldsymbol{r}_{i+1} / (\boldsymbol{r}_{i+1}^T A\boldsymbol{r}_{i+1} - \beta_i\boldsymbol{r}_{i+1}^T\boldsymbol{r}_{i+1}/\alpha_i)$
12: **end for**

Algorithm 3 The CBCG method

1: $\boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0$
2: $S_0 := (T_0(A)\boldsymbol{r}_0, T_1(A)\boldsymbol{r}_0, \cdots, T_{k-1}(A)\boldsymbol{r}_0)$
3: $Q_0 := S_0$
4: **for** $i = 0, 1, 2, \cdots$ until $\| \boldsymbol{r}_i \|_2 / \| \boldsymbol{r}_0 \|_2 < \epsilon$ **do**
5: Compute $Q_i^T AQ_i, Q_i^T\boldsymbol{r}_{ik}$
6: $\boldsymbol{a}_i := (Q_i^T AQ_i)^{-1} Q_i^T\boldsymbol{r}_{ik}$
7: $\boldsymbol{x}_{(i+1)k} := \boldsymbol{x}_{ik} + Q_i\boldsymbol{a}_i$
8: $\boldsymbol{r}_{(i+1)k} := \boldsymbol{r}_{ik} - AQ_i\boldsymbol{a}_i$
9: $S_{i+1} := (T_0(A)\boldsymbol{r}_{(i+1)k}, T_1(A)\boldsymbol{r}_{(i+1)k}, \cdots, T_{k-1}(A)\boldsymbol{r}_{(i+1)k})$
10: Compute $Q_i^T AS_{i+1}$
11: $B_i := (Q_i^T AQ_i)^{-1} Q_i^T AS_{i+1}$
12: $Q_{i+1} := S_{i+1} - Q_iB_i$
13: $AQ_{i+1} := AS_{i+1} - AQ_iB_i$
14: **end for**

Algorithm 4 Chebyshev Basis

1: $\eta := 2/(\lambda_{max} - \lambda_{min})$
2: $\zeta := (\lambda_{max} + \lambda_{min})/(\lambda_{max} - \lambda_{min})$
3: $\boldsymbol{s}_0 := \boldsymbol{r}$
4: $\boldsymbol{s}_1 := \eta A\boldsymbol{s}_0 - \zeta\boldsymbol{s}_0$
5: **for** $j = 2$ to k **do**
6: $\boldsymbol{s}_j := 2\eta A\boldsymbol{s}_{j-1} - 2\zeta\boldsymbol{s}_{j-1} - \boldsymbol{s}_{j-2}$
7: **end for**
8: $S := (\boldsymbol{s}_0, \boldsymbol{s}_1, \cdots, \boldsymbol{s}_{k-1})$
9: $AS := (A\boldsymbol{s}_0, A\boldsymbol{s}_1, \cdots, A\boldsymbol{s}_{k-1})$

CBCG に現れる $Q_i^T \times AQ_i$（5 行目）と $Q_i^T \times AS_{i+1}$（10 行目）は k 本のベクトルどうしの内積計算に相当する計算となり，$Q_i^T \times \boldsymbol{r}_i$（5 行目）は k 本と 1 本のベクトルの内積計

算に相当する．また，$Q_i^T AQ_i$ と $Q_i^T \boldsymbol{r}$ は同時に計算が可能である．そのため，係数行列 A をブロック行分割で MPI 並列で実装した場合，CBCG 法 1 反復あたり MPI_AllReduce が 2 回発生する．

3. CBCG 法への通信削減アルゴリズムの適用

この章では，3.1 節に C-CG 法のアイディアをもとに CBCG 法 1 反復あたりの MPI_AllReduce を 2 回から 1 回に減らすアルゴリズムについて，3.2 節で CBCG 法への MPK の適用について，3.3 節で演算量と通信回数について述べる．

3.1 MPI_AllReduce のさらなる削減

本論文では，C-CG 法のアイディアを基に Algorithm 3 の 5 行目に現れる，$Q_{i+1}^T AQ_{i+1}$ と $Q_{i+1}^T \boldsymbol{r}_{(i+1)k}$ を式展開することで，CBCG 法 1 反復中に発生する MPI_AllReduce を 1 回にする方法を示す．CBCG 法では，Algorithm 3 の 11 行目で $Q_i^T AS_{i+1}$ の計算と 5 行目の $Q_{i+1}^T AQ_{i+1}$ と $Q_{i+1}^T \boldsymbol{r}_{(i+1)k}$ の計算で MPI_AllReduce が計 2 回行われる．ここで，5 行目の計算は 11 行目の計算の後に行われるものとして考える．つまり，Q_i と S_{i+1} を用いて $Q_{i+1}^T AQ_{i+1}$ と $Q_{i+1}^T \boldsymbol{r}_{(i+1)k}$ の計算を行う必要がある．

はじめに，$Q_{i+1}^T AQ_{i+1}$ について，Algorithm 3 の 12 行目より，

$$Q_{i+1} = S_{i+1} - Q_i B_i \tag{5}$$

であるため，$Q_{i+1}^T AQ_{i+1}$ に代入すると，

$$\begin{aligned} Q_{i+1}^T AQ_{i+1} &= (S_{i+1} - Q_i B_i)^T A(S_{i+1} - Q_i B_i) \\ &= S_{i+1}^T AS_{i+1} - S_{i+1}^T AQ_i B_i \\ &\quad - B_i^T Q_i^T AS_{i+1} + B_i^T Q_i^T AQ_i B_i \end{aligned} \tag{6}$$

となる．ここで，

$$B_i = (Q_i^T AQ_i)^{-1} Q_i^T AS_{i+1} \tag{7}$$

であり，式 (6) に現れる，$B_i^T Q_i^T AQ_i B_i$ に式 (7) を代入すると，

$$\begin{aligned} B_i^T Q_i^T AQ_i B_i &= B_i^T Q_i^T AQ_i (Q_i^T AQ_i)^{-1} Q_i^T AS_{i+1} \\ &= B_i^T Q_i^T AS_{i+1} \end{aligned} \tag{8}$$

となる．したがって，式 (6) に式 (8) を代入すると，

$$\begin{aligned} Q_{i+1}^T AQ_{i+1} &= S_{i+1}^T AS_{i+1} - S_{i+1}^T AQ_i B_i \\ &\quad - B_i^T Q_i^T AS_{i+1} + B_i^T Q_i^T AQ_i B_i \\ &= S_{i+1}^T AS_{i+1} - S_{i+1}^T AQ_i B_i \\ &\quad - B_i^T Q_i^T AS_{i+1} + B_i^T Q_i^T AS_{i+1} \\ &= S_{i+1}^T AS_{i+1} - S_{i+1}^T AQ_i B_i \end{aligned} \tag{9}$$

次に，$Q_{i+1}^T \boldsymbol{r}_{(i+1)k}$ も同様に，式 (5) を代入すると，

Algorithm 5 The CBCGR method

1: $\boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0$
2: $S_0 := (T_0(A)\boldsymbol{r}_0, T_1(A)\boldsymbol{r}_0, \cdots, T_{k-1}(A)\boldsymbol{r}_0)$
3: $\boldsymbol{a}_0 := (S_0^T AS_0)^{-1} S_0^T \boldsymbol{r}_0$
4: $B_0 := 0$
5: **for** $i = 0, 1, 2, \cdots$ until $\| \boldsymbol{r}_i \|_2 / \| \boldsymbol{r}_0 \|_2 < \epsilon$ **do**
6: $\quad Q_i := S_i - Q_{i-1} B_{i-1}$
7: $\quad AQ_i := AS_i + AQ_{i-1} B_{i-1}$
8: $\quad \boldsymbol{x}_{(i+1)k} := \boldsymbol{x}_{ik} + Q_i \boldsymbol{a}_i$
9: $\quad \boldsymbol{r}_{(i+1)k} := \boldsymbol{r}_{ik} - AQ_i \boldsymbol{a}_i$
10: $\quad S_{i+1} := (T_0(A)\boldsymbol{r}_{(i+1)k}, T_1(A)\boldsymbol{r}_{(i+1)k}, \cdots, T_{k-1}(A)\boldsymbol{r}_{(i+1)k})$
11: $\quad$ Compute $S_{i+1}^T AS_{i+1}, Q_i^T AS_{i+1}, S_{i+1}^T \boldsymbol{r}_{(i+1)k}, Q_i^T \boldsymbol{r}_{(i+1)k}$
12: $\quad B_{i+1} := (Q_i^T AQ_i)^{-1} Q_i^T AS_{i+1}$
13: $\quad Q_{i+1}^T AQ_{i+1} := S_{i+1}^T AS_{i+1} - S_{i+1}^T AQ_i B_{i+1}$
14: $\quad Q_{i+1}^T \boldsymbol{r}_{(i+1)k} := S_{i+1}^T r_{(i+1)k} - B_{i+1}^T Q_i^T r_{(i+1)k}$
15: $\quad \boldsymbol{a}_{i+1} := (Q_{i+1}^T AQ_{i+1})^{-1} Q_{i+1}^T r_{(i+1)k}$
16: **end for**

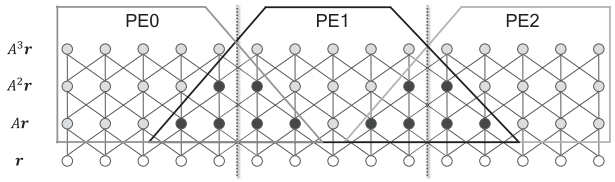

図 1　行列 A が 3 重対角行列のとき MPK で $(A\boldsymbol{r}, A^2\boldsymbol{r}, A^3\boldsymbol{r})$ の計算を 3 プロセスに分散したときの要素間の依存関係

Fig. 1　Dependency between elements when A is a tridiagonal matrix. MPK divides $(A\boldsymbol{r}, A^2\boldsymbol{r}, A^3\boldsymbol{r})$ into three processes.

$$\begin{aligned} Q_{i+1}^T \boldsymbol{r}_{(i+1)k} &= (S_{i+1} - Q_i B_i)^T \boldsymbol{r}_{(i+1)k} \\ &= S_{i+1}^T \boldsymbol{r}_{(i+1)k} - B_i^T Q_i^T \boldsymbol{r}_{(i+1)k} \end{aligned} \tag{10}$$

となる．よって，式 (9)，式 (10) より，$S_{i+1}^T AS_{i+1}$ と $S_{i+1}^T AQ_i$，$S_{i+1}^T \boldsymbol{r}_{(i+1)k}$，$Q_i^T \boldsymbol{r}_{(i+1)k}$ の 4 つの計算で導出可能となる．また，A は対称行列であることから，

$$S_{i+1}^T AQ_i = (Q_i^T AS_{i+1})^T \tag{11}$$

であるため，$S_{i+1}^T AS_{i+1}$ と $Q_i^T AS_{i+1}$，$S_{i+1}^T \boldsymbol{r}_{(i+1)k}$，$Q_i^T \boldsymbol{r}_{(i+1)k}$ の 4 つを計算することで，CBCG 法 1 反復に必要な MPI_AllReduce が 1 回となる．本論文ではこの手法を CBCGR 法と呼ぶことにする．CBCGR 法のアルゴリズムを Algorithm 5 に示す．

3.2 CBCG 法への Matrix Powers Kernel の適用

MPK は $(A\boldsymbol{r}, A^2\boldsymbol{r}, \cdots, A^k\boldsymbol{r})$ の計算に必要な領域を各プロセスが拡張して保持することで，1 回の一対一通信で計算を可能とする手法である．行列 A が 3 重対角行列のとき $(A\boldsymbol{r}, A^2\boldsymbol{r}, A^3\boldsymbol{r})$ の計算を 3 プロセスに分散したときの依存関係を**図 1** に示す．図 1 の PE1 の領域は 5 であるが，MPK を適用する場合は，プロセスの境界に隣接する部分を拡張するため，PE0 の方向に 2，PE2 の方向に 2 ずつ拡張する．そのため PE1 は領域サイズ 9 まで保持する必要があるため，メモリ使用量が増加する．行列 A が

表 1　CG 法 k 反復あたりの CG 法と C-CG 法，CBCG 法，CBCGR 法，CBCGR-MPK 法の演算と通信のコスト

Table 1　The computation and communication costs for the CG, C-CG, CBCG, CBCGR and CBCGR-MPK methods per the CG method k iterations.

	F_{other}	F_{SpMV}	$S_{\text{collective}}$	S_{P2P}
CG	$(10N/P)k$	$(10N/P)k$	$2k$	$4k$
C-CG	$(12N/P)k$	$(10N/P)k$	k	$4k$
CBCG	$(8k^2+12k-3)N/P$	$(10N/P)k$	2	$4k$
CBCGR	$(8k^2+14k-3)N/P+k^2+k$	$(10N/P)k$	1	$4k$
CBCGR-MPK	$(8k^2+10k+5)N/P+12(k-1)\sqrt{N/P}$ $+13k^2-23k+12+\sum_{i=2}^{k-1}\{\sqrt{N/P}+2(k-1-i)\}^2$	$10\sum_{i=0}^{k-1}\{\sqrt{N/P}+2(k-1-i)\}^2$	1	8

3 重対角行列で要素数が N，プロセス数が P とし，$A^k\boldsymbol{r}$ を計算する場合は，各プロセスは $\{N/P+2(k-1)\}$ 次元の行列に拡張し保持する必要がある．また，演算量は $6\sum_{i=0}^{k-1}\{N/P+2(k-1-i)\}$ となる．メモリ使用量や演算量は行列の構造や k によって大きく変化し，2 次元や 3 次元に拡張した場合に上記の 3 重対角行列の場合以上にメモリ使用量および演算量が増加する．CBCG 法では Krylov 部分空間をまとめて生成する部分である Algorithm 4 に対して MPK を適用することができる．本研究では，CBCG 法（k）に MPK を適用する場合は，k 乗までまとめて計算する MPK を適用した．また，CBCG 法は行列べき乗演算の間にベクトル演算（Algorithm 4 の 4 行目と 6 行目）も含まれているため，この部分もプロセス間での重複計算が発生する．本論文では，CBCGR 法に対して MPK の適用を行い，この手法を CBCGR-MPK 法と呼ぶことにする．

3.3　演算量と通信回数

CG 法と C-CG 法，CBCG 法，CBCGR 法，CBCGR-MPK 法の演算量と通信回数について議論する．CG 法は Algorithm 1 の 3 行目から 11 行，C-CG 法は Algorithm 2 の 4 行目から 12 行目，CBCG 法は Algorithm 3 の 4 行目から 14 行目，CBCGR 法は Algorithm 5 の 5 行目から 16 行目までの演算量と通信回数をそれぞれ算出する．ここでの演算量は浮動小数点の加算・乗算の回数 F とし，通信回数はメッセージ数 S とする．また，SpMV に関しては係数行列の構造に依存し，演算量の算出が困難であるため，2 次元 5 点差分の問題として算出を行う．SpMV の演算量を F_{SpMV} として，その他の演算量 F_{other} と分類し，通信も SpMV で発生する一対一通信（`MPI_Send/Recv`）を S_{P2P} と，集団通信（`MPI_AllReduce`）を $S_{\text{collective}}$ としてそれぞれ分類する．$S_{\text{collective}}$ は `MPI_AllReduce` の回数とする．

係数行列 A の次元数を N とし，立ち上げるプロセス数を P とする．各プロセスが均等に領域を保持すると仮定すると 1 プロセスあたりの領域は $\sqrt{N/P}\times\sqrt{N/P}$ となる．メッセージ数は通常の 2 次元 5 点差分での SpMV の場合，通信が必要となるデータは各プロセスが保持する正方形に接する辺の部分となる．そのため，通信相手となるプロセスは基本的に 4 となるため，1 回の SpMV でのメッセージ数は 4 となる．MPK の場合，正方形に接する辺の部分に加えて，頂点に接する部分も追加されるため，8 となる．また，1 プロセスが保持する領域サイズは 3 重対角行列のときと同様に隣接するプロセスの方向に拡張するため，$\{\sqrt{N/P}+2(k-1)\}^2$ となり，演算量は $10\sum_{i=0}^{k-1}\{\sqrt{N/P}+2(k-1-i)\}^2$ となる．

CG 法 k 反復あたりの CG 法と C-CG 法，CBCG 法，CBCGR 法，CBCGR-MPK 法の演算と通信コストの比較を**表 1** に示す．CBCG 法で行われる QR 分解はサイズが k（2〜30 程度）と非常に小さいものであるため，演算コストとして含めていない．この表が示すとおり，通信を削減することにより演算量が増えていることが分かる．

4.　実装方法

この章では CBCG 法に現れる密行列積と各手法に現れる疎行列ベクトル積の Hybrid 並列による実装方法について述べる．

4.1　密行列積

CBCG 法では Algorithm 3 の 5 行目（Q^TAQ）と 10 行目（Q^TAS），CBCGR 法では Algorithm 5 の 11 行目（S^TAS, Q^TAS）で N 行 k 列と k 行 N 列の密行列積を行い k 行 k 列の行列を求める．N は係数行列の次元数，k は CBCG 法の段数である．ここで，行列 A を N 行 k 列，行列 B を k 行 N 列，行列 C を k 行 k 列とし，N をプロセス数 P で各プロセスに分割した場合の $C=AB$ は

$$C=\sum_{i=0}^{P}C_i=\sum_{i=0}^{P}(A_iB_i)$$

という演算をすることになる．このとき，各プロセスが C_i を計算し，総和を計算するため，`MPI_AllReduce` が発生する．各プロセスでスレッド並列を行う場合，N/P をスレッド数 T で分割し行うため，$C_i=A_iB_i$ は

$$C_i=\sum_{j=0}^{T}(A_{i,j}B_{i,j})$$

となる．この場合，スレッド数分だけ各スレッドの演算結

```
1  for(i = 0; i < N; i++)
2  {
3    for(j = A.ptr[i]; j < A.ptr[i+1]; j++)
4    {
5      idx = A.col[j];
6      y[i] += A.val[j] * x[idx];
7      y[idx] += A.val[j] * x[i];
8    }
9    y[i] += A.d[i] * x[i];
10 }
```

図 2　対称行列とベクトル積の疑似コード
Fig. 2　Pseudo-code of symmetric sparse matrix vector multiply.

果を格納するワーク行列が必要となり，全スレッドの計算が終わったあとに演算結果を足し合わせる必要がある．したがって，Hybrid 並列では

$$C = \sum_{i=0}^{P}\{\sum_{j=0}^{T}(A_{i,j}B_{i,j})\}$$

となる．本研究では，上記の要領で並列化を行い行列積の演算部分は BLAS ライブラリの dgemm_() を用いて実装を行った．

4.2　疎行列ベクトル積

CG 法および CBCG 法は係数行列が正定値対称な場合にのみ解を得ることができる．そのため，問題として与えられる係数行列 A は，下三角部 L と対角部 D を用いて

$$A = L + D + L^T$$

と表すことができる．そのため，行列を下三角部と対角部のみを保持することで SpMV が可能である．行列を下三角部と対角部で Compressed row Storage（CRS）[15] 形式で格納した場合の $\boldsymbol{y} = A\boldsymbol{x}$ の C コードを**図 2** に示す．対称行列向けの SpMV をスレッド並列で行う場合，図 2 の 7 行目のベクトル $\boldsymbol{y}$ へのストアでスレッド間の競合が発生してしまう．我々は，対角ブロックによる並列化を行った．具体的には行列 A を対角にスレッド数分分解した対角ブロック行列 D_j と対角ブロックに入らなかったその他の部分を行列 C として，

$$\boldsymbol{y} = \sum_{j=0}^{T} D_j\boldsymbol{x} + C\boldsymbol{x} \tag{12}$$

という演算をする．ここで，下三角のみ保持した対角ブロックを各スレッドに割当てることで，スレッド間でのベクトル $\boldsymbol{y}$ へのストアの競合がなくなる．$C\boldsymbol{x}$ は C を下三角と上三角部を保持して並列に計算を行う．また，MPK では対角ブロックでの実装が困難なため，下三角部と上三角部と対角部を CRS 形式で保持し，並列化を行う実装とした．

5.　数値実験

5.1　実験環境と評価条件

FX10（oakleaf-fx）スーパコンピュータシステム [16] を使用して実験を行った．FX10 は 1 ノードに 1 個の SPARC64 fxIX プロセッサ（16 コア，1.848 GHz），32 GB のメモリ（DDR3 SDRAM, 85 GB/sec.）を搭載している．また，インターコネクトは 6 次元メッシュ/トーラス（5 GB/sec./link）で接続されている．プログラムは C 言語で実装を行い，密行列の演算に BLAS ルーチン，プロセス並列に MPI ライブラリを使用した，コンパイラは “`mpifccpx`”，オプションには “`-Kfast,openmp,ipo -lm -SSL2`” を使用した．

1 ノードあたり 1 プロセス立ち上げ，1 プロセスあたり 16 スレッド立ち上げる OpenMP/MPI の Hybrid 並列モデルで実験を行い，最大 1,440 ノードまで使用した．収束条件は相対残差が 10^{-12} とした．また，前処理として対角スケーリングを施した．CBCG 法で必要となる最大・最小固有値は，べき乗法で推定した値と 0 とした．また，CBCG 法の実行時間にはべき乗法による推定時間は含めていない．時間計測には FX10 の詳細プロファイラを用いて全プロセスの平均時間を算出した．

対象とする問題は以下の 2 種類とした．

(1) 2 次元 Poisson 方程式
　拡散係数が一定の領域が 1,024 × 1,024 の未知数が 1,048,576 とした．

(2) 3 次元 Poisson 方程式
　不均質な多孔質媒体中の地下水の流れを Poisson 方程式によって解く問題 [17]．透水係数は Sequential Gaussian アルゴリズム [18] により発生させた値とした．透水係数の最小値，最大値，平均値は 10^{-5}，10^{5}，1.0 となるように設定されている．領域が 240 × 240 × 240 の未知数が 13,824,000 とした．

問題サイズを一定とし並列数を増加させるストロングスケーリングでの実験を行った．行列の分散は，XYZ 軸の分割数をそれぞれ P_{X}，P_{Y}，P_{Z} とし，プロセス数 P が $P_{\mathrm{X}} \times P_{\mathrm{Y}}(\times P_{\mathrm{Z}})$ となるように設定した．2 次元 Poisson 方程式と 3 次元 Poisson 方程式でのプロセス数とグリッドの分割数および 1 プロセスあたりの領域サイズを**表 2** にそれぞれ示す．また，係数行列および反復中で使用するスカラー，ベクトル，密行列はすべて倍精度で保持している．

5.2　2 次元 Poisson 方程式での実験結果

はじめに，2 次元 Poisson 方程式での実験結果について述べる．k ごとの収束に要した反復回数を**表 3** に示す．表の CBCG 法の $k = 1$ は CG 法，CBCGR 法の $k = 1$ は C-CG 法を表している．また，() 内は CBCG 法の反復回数を CG 法換算にしたものとなっている．この表から k を増やすことで反復回数が増えていることが分かる．また，CG 法と

表 2　本実験での 2 次元 Poisson 方程式（上段）と 3 次元 Poisson 方程式（下段）のグリッドの分割数と各プロセスの領域サイズ

Table 2　The shape of the process grid and the local domain on each process of 2D-Poisson problem (top) and 3D-Poisson problem (bottom) in the experiments.

N	P	P_X	P_Y	P_Z	the shape of local domain per process
$1{,}048{,}576 = 1{,}024 \times 1{,}024$	32	8	4		$32{,}768 = 128 \times 256$
	64	8	8		$16{,}384 = 128 \times 128$
	128	16	8		$8{,}192 = 64 \times 128$
	256	16	16		$4{,}096 = 64 \times 64$
	512	32	16		$2{,}048 = 32 \times 64$
	1,024	32	32		$1{,}024 = 32 \times 32$
$13{,}824{,}000 = 240 \times 240 \times 240$	180	6	6	5	$76{,}800 = 40 \times 40 \times 48$
	360	12	6	5	$38{,}400 = 20 \times 40 \times 48$
	720	12	12	5	$19{,}200 = 20 \times 20 \times 48$
	1,440	12	12	10	$9{,}600 = 20 \times 20 \times 24$

表 3　CG 法，C-CG 法，CBCG 法，CBCGR 法の k ごとの収束に要した反復回数：2 次元 Poisson 方程式（$1{,}024 \times 1{,}024$）．() 内は CG 法換算の反復回数

Table 3　The number of iterations of CG, C-CG, CBCG, and CBCGR methods in each k: 2D-Poisson problem ($1{,}024 \times 1{,}024$). () is the number of iterations of the CG method conversion.

k	CG / CBCG	C-CG / CBCGR
1	2,861	3,123
2	1,431(2,862)	1,431(2,862)
3	954(2,862)	1,041(3,123)
4	716(2,864)	716(2,864)
5	573(2,865)	625(3,125)
6	521(3,126)	521(3,126)
7	447(3,129)	447(3,129)
8	391(3,128)	391(3,128)
9	347(3,123)	347(3,123)
10	313(3,130)	313(3,130)
11	284(3,124)	284(3,124)
12	261(3,132)	261(3,132)
13	241(3,133)	241(3,133)
14	224(3,136)	224(3,136)
15	209(3,135)	209(3,135)
16	196(3,136)	196(3,136)
17	184(3,128)	184(3,128)
18	174(3,132)	174(3,132)
19	165(3,135)	166(3,154)
20	157(3,140)	157(3,140)
21	149(3,129)	149(3,129)
22	142(3,124)	142(3,124)
23	136(3,128)	136(3,128)
24	131(3,144)	131(3,144)
25	125(3,125)	125(3,125)
26	121(3,146)	121(3,146)
27	117(3,159)	122(3,294)
28	112(3,136)	112(3,136)
29	108(3,132)	108(3,132)
30	105(3,150)	105(3,150)

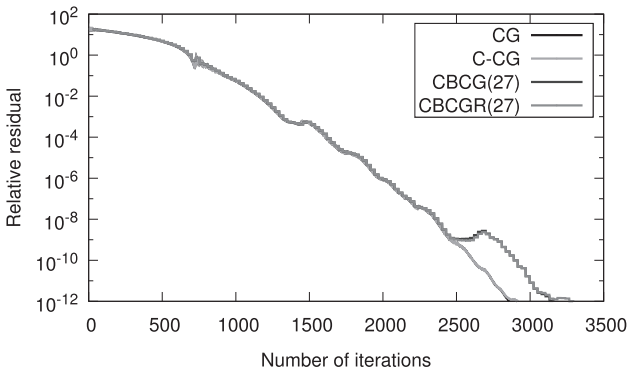

図 3　CG 法，CBCG 法，CBCGR 法の収束履歴：2 次元 Poisson 方程式（$1{,}024 \times 1{,}024$）

Fig. 3　Convergence history of the CG, CBCG, and CBCGR methods: 2D-Poisson problem ($1{,}024 \times 1{,}024$).

C-CG 法を比較すると，計算順序を変えたことによる誤差の影響で反復回数が増えていることが分かる．また，CBCG 法から CBCGR 法に変わったことにより $k = 3, 5, 19, 27$ のときに増加していることが確認できる．ここで，CG 法と C-CG 法，$k = 27$ のときの CBCG 法，CBCGR 法の収束履歴を**図 3** に示す．C-CG 法の履歴をみると，今回の終了条件である 10^{-12} 付近で収束が悪化していることが分かる．CBCG 法と CBCGR 法の履歴をみると，両者ともに 2,500 反復付近までは CG 法と同じ収束履歴となっているが，それ以降で収束が悪化している結果となっている．また，CBCGR 法は C-CG 法と同様に 10^{-12} 付近で収束が CBCG 法に比べ悪化している結果となった．この原因の調査や対策については，今後の課題である．

次に CBCG 法と CBCGR 法，CBCGR-MPK 法の通信削減の効果について述べる．1,024 プロセスでの CG 法と C-CG 法，CBCG 法，CBCGR 法，CBCGR-MPK 法の k ごとの収束に要した時間を**表 4** に示す．表より C-CG 法は `MPI_AllReduce` の回数が CG 法の半分に対して実行結果が増大している結果となった．CBCG 法と CG 法を比較

表 4　1,024 プロセスのときの CG 法, C-CG 法, CBCG 法, CBCGR 法，CBCGR-MPK 法の k ごとの収束に要した時間：2 次元 Poisson 方程式（1,024 × 1,024）

Table 4　The convergence time of the CG, C-CG, CBCG, CBCGR, and CBCGR-MPK methods in each k with 1,024 processes: 2D-Poisson problem (1,024 × 1,024).

k	CG/CBCG	C-CG/CBCGR	CBCGR-MPK
1	6.73×10^{-1}	7.74×10^{-1}	
2	8.95×10^{-1}	7.66×10^{-1}	7.67×10^{-1}
3	7.05×10^{-1}	6.40×10^{-1}	5.89×10^{-1}
4	6.23×10^{-1}	5.59×10^{-1}	5.02×10^{-1}
5	5.63×10^{-1}	5.67×10^{-1}	5.04×10^{-1}
6	5.81×10^{-1}	5.39×10^{-1}	4.15×10^{-1}
7	5.53×10^{-1}	5.15×10^{-1}	4.30×10^{-1}
8	5.26×10^{-1}	4.97×10^{-1}	4.14×10^{-1}
9	4.67×10^{-1}	4.84×10^{-1}	4.04×10^{-1}
10	5.05×10^{-1}	4.76×10^{-1}	3.95×10^{-1}
11	4.94×10^{-1}	4.69×10^{-1}	3.82×10^{-1}
12	4.82×10^{-1}	4.60×10^{-1}	3.56×10^{-1}
13	4.36×10^{-1}	4.58×10^{-1}	3.73×10^{-1}
14	4.81×10^{-1}	4.53×10^{-1}	3.65×10^{-1}
15	4.69×10^{-1}	4.51×10^{-1}	3.63×10^{-1}
16	4.73×10^{-1}	4.55×10^{-1}	3.66×10^{-1}
17	4.49×10^{-1}	4.37×10^{-1}	3.53×10^{-1}
18	4.51×10^{-1}	4.35×10^{-1}	3.57×10^{-1}
19	4.48×10^{-1}	4.36×10^{-1}	3.57×10^{-1}
20	4.46×10^{-1}	4.34×10^{-1}	3.53×10^{-1}
21	4.43×10^{-1}	4.34×10^{-1}	3.47×10^{-1}
22	4.43×10^{-1}	4.33×10^{-1}	3.53×10^{-1}
23	4.47×10^{-1}	4.34×10^{-1}	3.56×10^{-1}
24	4.41×10^{-1}	4.36×10^{-1}	3.58×10^{-1}
25	4.40×10^{-1}	4.32×10^{-1}	3.88×10^{-1}
26	4.42×10^{-1}	4.38×10^{-1}	3.60×10^{-1}
27	4.45×10^{-1}	4.56×10^{-1}	3.88×10^{-1}
28	4.35×10^{-1}	4.30×10^{-1}	3.88×10^{-1}
29	4.39×10^{-1}	4.37×10^{-1}	3.69×10^{-1}
30	4.43×10^{-1}	4.44×10^{-1}	4.03×10^{-1}

すると，k が 8 以下のときは CG 法よりも遅い結果となっている．それ以降では CG 法よりも高速となった．また，CBCGR 法も同様に k が 3 以下のときは CG 法よりも遅く，それ以降では高速であった．CBCG 法と CBCGR 法を比較すると k が 5, 9, 13, 19, 27, 30 のときだけ CBCG 法が速く，それ以外の場合では CBCGR 法の方が高速となっている．CBCGR-MPK 法は k が 2 のとき以外は CBCGR 法よりも高速となった．CBCG 法は $k = 28$，CBCGR 法は $k = 28$，CBCGR-MPK 法は $k = 21$ のときに最速であったため，CG 法と C-CG 法，CBCG 法（$k = 28$），CBCGR 法（$k = 28$），CBCGR-MPK 法（$k = 21$）の 32 プロセスから 1,024 プロセスまでのストロングスケーリングでの収束に要した時間を**図 4** に示す．CG 法と C-CG 法が停滞しているのに対して CBCG 法と CBCGR 法と CBCGR-MPK 法は 1,024 プロセスまで並列数の増加とともに実行時間が減少

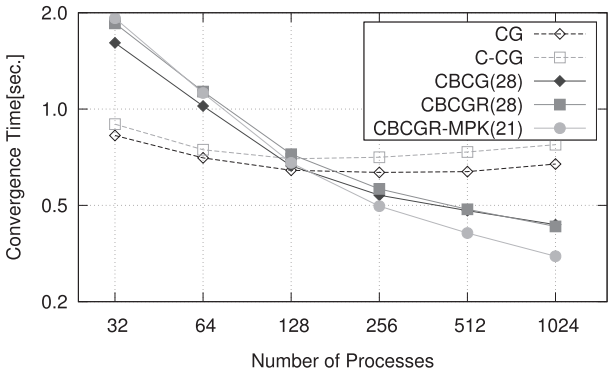

図 4　ストロングスケーリングによる CG 法と C-CG 法，CBCG 法（$k = 28$），CBCGR 法（$k = 28$），CBCGR-MPK 法（$k = 21$）の収束に要した時間：2 次元 Poisson 方程式（1,024 × 1,024）

Fig. 4　The convergence time of the CG, C-CG, CBCG ($k = 28$), CBCGR ($k = 28$), and CBCGR-MPK ($k = 21$) methods. The strong scaling is evaluated: 2D-Poisson problem (1,024 × 1,024).

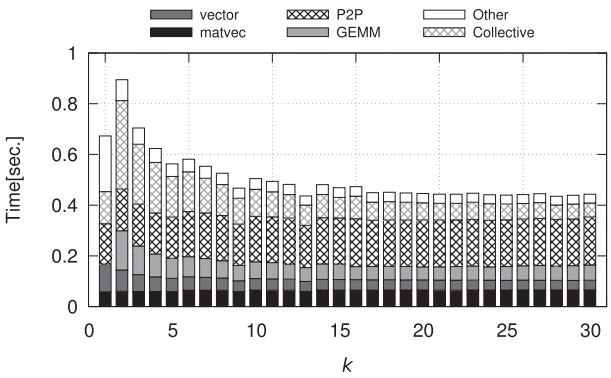

図 5　1,024 プロセスのときの k ごとの CG 法と CBCG 法の実行時間の内訳：2 次元 Poisson 方程式（1,024 × 1,024）

Fig. 5　The detailed breakdown of execution time of the CG and CBCG methods in each k with 1,024 processes: 2D-Poisson problem (1,024 × 1,024).

している．また，CBCG 法と CBCGR 法，CBCGR-MPK 法は 32 プロセスでは CG 法の 2 倍近くの時間を要していたのに対して，256 プロセス以降で CG 法よりも高速となる結果となった．CBCG 法と CBCGR 法を比較すると，512 プロセスまでは CBCG 法が速い結果であるのに対して，1,024 プロセスでは CBCGR 法が速い結果となった．また，CBCGR-MPK 法は 256 プロセス以降でどの手法よりも高速となった．

それぞれの演算時間と通信時間の比較をするために実行時間の内訳について述べる．各手法の k を変えたときの 1,024 プロセスにおける実行時間の内訳を**図 5**，**図 6**，**図 7** にそれぞれ示す．実行時間の内訳は FX10 の詳細プロファイラを用いてプロセスごとの各ルーチンに要した時間を計測し，平均を算出した．図の matvec は疎行列ベクトル積，vector はベクトル演算，GEMM は CBCG 法で新たに現れる行列積，P2P は疎行列ベクトル積のときに発生する

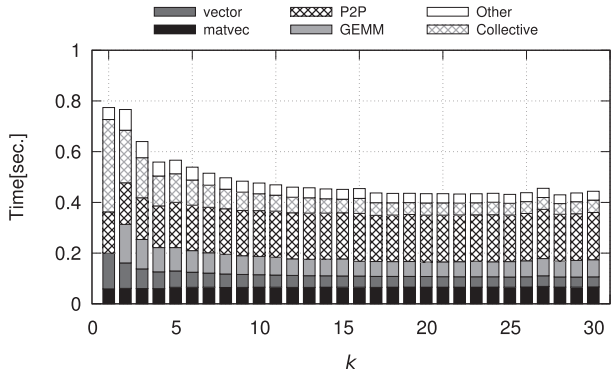

図 6　1,024 プロセスのときの k ごとの C-CG 法と CBCGR 法の実行時間の内訳：2 次元 Poisson 方程式（1,024 × 1,024）

Fig. 6　The detailed breakdown of execution time of the C-CG and CBCGR methods in each k with 1,024 processes: 2D-Poisson problem (1,024 × 1,024).

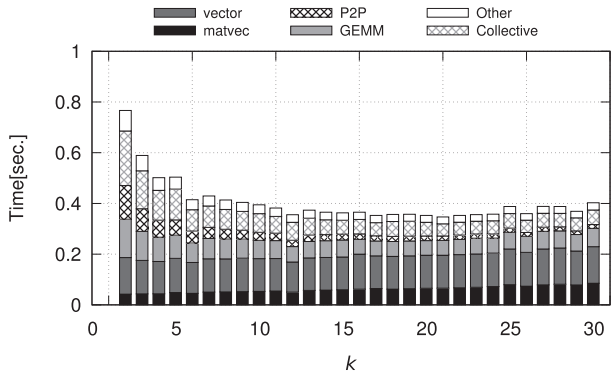

図 7　1,024 プロセスのときの k ごとの CBCGR-MPK 法の実行時間の内訳：2 次元 Poisson 方程式（1,024 × 1,024）

Fig. 7　The detailed breakdown of execution time of the CBCGR-MPK method in each k with 1,024 processes: 2D-Poisson problem (1,024 × 1,024).

MPI_Send/Recv，Collective は MPI_AllReduce，Other は実行時間から上記の時間を引いたものとなっている．CBCG 法の k が 8 以上のとき CG 法よりも集団通信の時間が少ない結果となった．CBCG 法と CBCGR 法を比較すると，全体的に集団通信の時間が削減されていることが分かる．CBCGR-MPK 法を見ると，一対一通信の時間が大幅に削減されている．その反面，ベクトル演算の時間がプロセス間の重複計算により増加している．

C-CG 法の MPI_AllReduce の時間を見てみると，CG 法の 2 倍以上かかっている結果となった．この結果について，詳細に分析するため FX10 上での Intel MPI Benchmarks [19] を用いて MPI_AllReduce の性能を測定した，IMB の MPI_AllReduce のベンチマークでは単精度データの総和計算を行うため，倍精度データのベンチマークの計測を行うようにコードを変更した．プロセス数を 256，512，1,024 にしたときのベンチマーク結果を図 **8** に示す．図が示すとおり，8 byte のときに比べ 16 byte 以降では 5〜6 倍の性能差となった．これは，スカラデータに対するリダクションは，Tofu のハードウェアにより高速に処理されるため，8 byte のときの MPI_AllReduce が高速に実行されるからである．

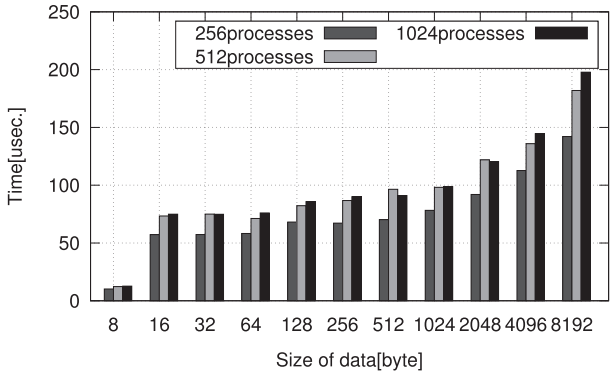

図 8　IMB による MPI_AllReduce の計測結果

Fig. 8　Result of MPI_AllReduce for IMB.

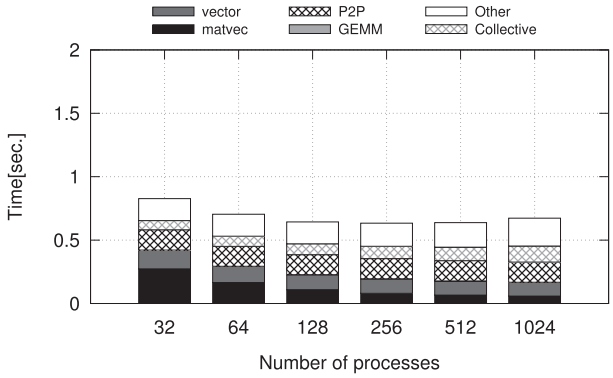

図 9　CG 法の実行時間の内訳：2 次元 Poisson 方程式（1,024 × 1,024）

Fig. 9　The detailed breakdown of execution time of the CG method: 2D-Poisson problem (1,024 × 1,024).

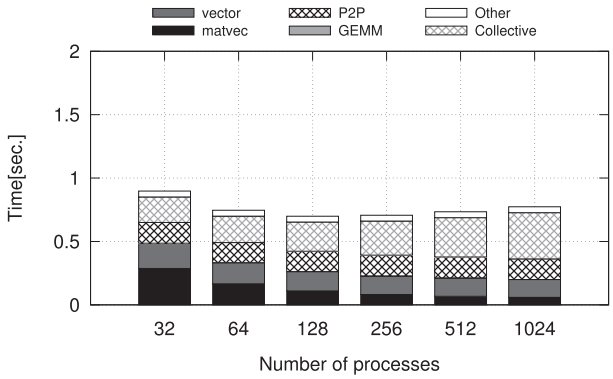

図 10　C-CG 法の実行時間の内訳：2 次元 Poisson 方程式（1,024 × 1,024）

Fig. 10　The detailed breakdown of execution time of the C-CG method: 2D-Poisson problem (1,024 × 1,024).

次に 32 プロセスから 1,024 プロセスのストロングスケーリングによる実行時間の内訳をそれぞれ，CG 法を図 **9**，C-CG 法を図 **10**，CBCG 法（$k = 28$）を図 **11**，CBCGR 法（$k = 28$）を図 **12** に，CBCGR-MPK 法（$k = 21$）を

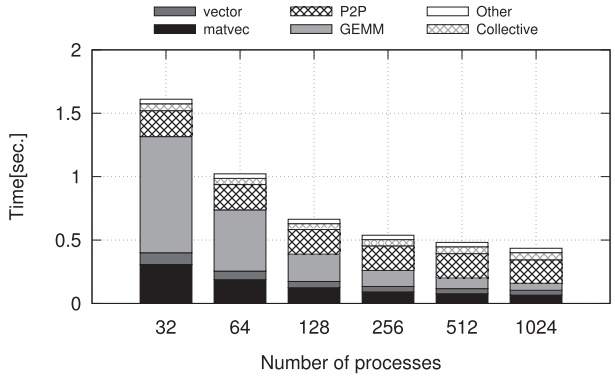

図 11　CBCG 法（$k = 28$）の実行時間の内訳：2 次元 Poisson 方程式（1,024 × 1,024）

Fig. 11　The detailed breakdown of execution time of the CBCG method ($k = 28$): 2D-Poisson problem (1,024× 1,024).

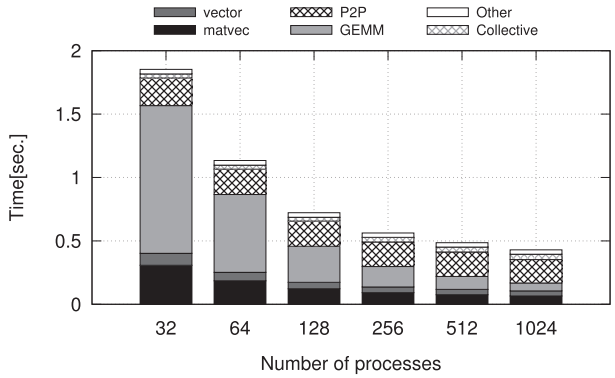

図 12　CBCGR 法（$k = 28$）の実行時間の内訳：2 次元 Poisson 方程式（1,024 × 1,024）

Fig. 12　The detailed breakdown of execution time of the CBCGR method ($k = 28$): 2D-Poisson problem (1,024 × 1,024).

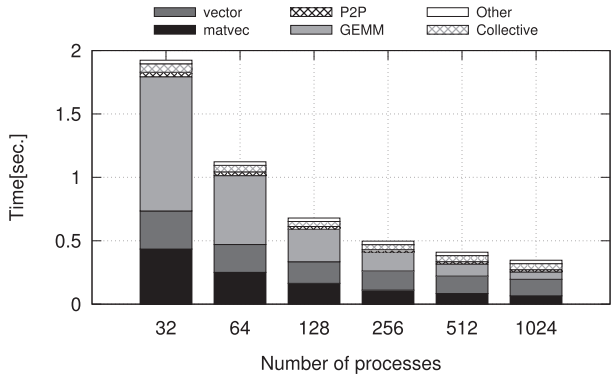

図 13　CBCGR-MPK 法（$k = 21$）の実行時間の内訳：2 次元 Poisson 方程式（1,024 × 1,024）

Fig. 13　The detailed breakdown of execution time of the CBCGR-MPK method ($k = 21$): 2D-Poisson problem (1,024 × 1,024).

図 13 に示す．CG 法を見てみると，並列数の増加とともに集団通信の時間が増えている．通信だけで見てみると集団通信の時間よりも一対一通信通信の時間の方が要して

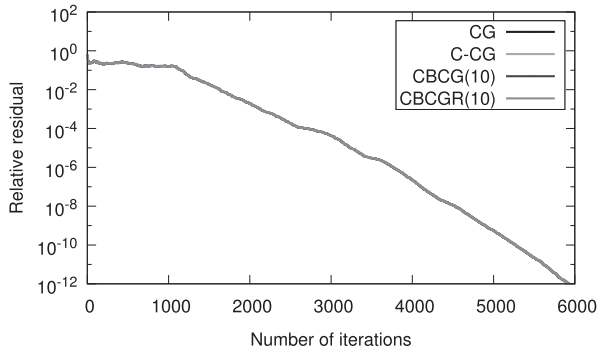

図 14　CG 法，C-CG 法，CBCG 法，CBCGR 法の収束履歴：3 次元 Poisson 方程式（240 × 240 × 240）

Fig. 14　Convergence history of the CG, C-CG, CBCG and CBCGR methods: 3D-Poisson problem (240 × 240 × 240).

いることが分かる．C-CG 法はどの並列数においても集団通信の時間が CG 法よりも多いことが分かる．CBCG 法は並列数が少ないときは行列積の時間が大半を占めているため，CG 法よりも遅い結果となったことが分かる．並列数の増大とともに演算時間の合計が短縮され，集団通信の時間も CG 法よりも少ない結果となった．CBCGR 法は並列数が少ないときは CBCG 法と同様の結果である．また，集団通信の時間は CBCG 法よりも少ない結果となった．CBCGR-MPK 法は一対一通信が大きく減少しているが，MPK によるプロセス間の重複計算によりベクトル演算の時間が増加している．

5.3　3 次元 Poisson 方程式での実験結果

3 次元 Poisson 方程式での結果について述べる．1,440 プロセスのときの 1 プロセスあたりの領域サイズが 20×20×24 とサイズが小さいため，CBCGR-MPK の k を 21 以上にすると 2 つ隣のプロセスを跨いで拡張してしまう．そのため，本実験では，k を 10 にして実験を行った．はじめに 3 次元 Poisson 方程式での収束に要した反復回数について述べる．**図 14** に CG 法と C-CG 法，CBCG 法，CBCGR 法の収束履歴を示す．CG 法の収束に要した反復回数は 5,922 反復，C-CG 法は 5,922 反復，CBCG 法は 593 反復（CG 法 5,930 反復相当），CBCGR 法は 593 反復（CG 法 5,930 反復相当）でそれぞれ収束した．グラフが示すとおり，どの手法も同じ収束履歴をたどり，ほぼ同じ反復回数で収束した．

次に，180 プロセスから 1,440 プロセスまでのストロングスケーリングによる CG 法と C-CG 法，CBCG 法，CBCGR 法，CBCGR-MPK 法の収束に要した時間を**図 15** に示す．CG 法は 720 プロセスから 1,440 プロセスにかけて実行時間が停滞している傾向となった．C-CG 法は 2 次元 Poisson 方程式と同様の理由によりどの並列数においても CG 法よりも遅い結果となった．CBCG 法は 720 プロ

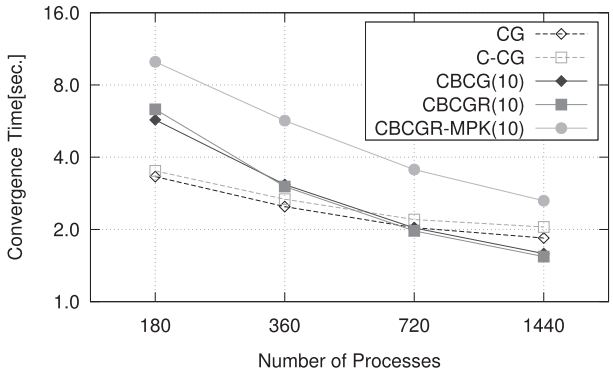

図 15　ストロングスケーリングによる CG 法と C-CG 法，CBCG 法（$k = 10$），CBCGR 法（$k = 10$），CBCGR-MPK 法（$k = 10$）の収束に要した時間：3 次元 Poisson 方程式（240×240×240）

Fig. 15　The convergence time of the CG, C-CG, CBCG ($k = 10$), CBCGR ($k = 10$), and CBCGR-MPK ($k = 10$) methods. The strong scaling is evaluated: 3D-Poisson problem ($240 \times 240 \times 240$).

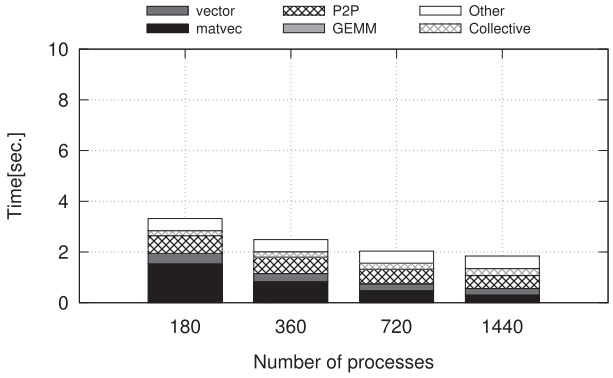

図 16　CG 法の実行時間の内訳：3 次元 Poisson 方程式（240 × 240 × 240）

Fig. 16　The detailed breakdown of execution time of the CG method: 3D-Poisson problem ($240 \times 240 \times 240$).

セス以降で CG 法よりも高速となった．CBCGR 法は 180 プロセスでは CBCG 法よりも遅い結果となったが，720 プロセス以降では CBCG 法よりも高速となる結果となった．また，CBCGR-MPK 法はどの並列数においても一番遅い結果となった．

180 プロセスから 1,440 プロセスのストロングスケーリングによる実行時間の内訳をそれぞれ，CG 法を**図 16**，C-CG 法を**図 17**，CBCG 法を**図 18**，CBCGR 法を**図 19**，CBCGR-MPK 法を**図 20** に示す．C-CG 法を見てみると，どの並列数においても集団通信の時間が CG 法よりも多く，2 次元 Poisson 方程式と同様の結果が得られた．CBCG 法は，並列数が少ないときは演算が大半を占めており，高並列時に短縮される結果となった．しかし，集団通信の時間は CG 法と比較すると，ほとんど同じであることが分かる．CBCGR 法は CBCG 法に比べ，行列演算の時間が増えている結果となった．CBCGR-MPK 法は一対一通信の時間は削減されているが，MPK の重複計算により SpMV とベ

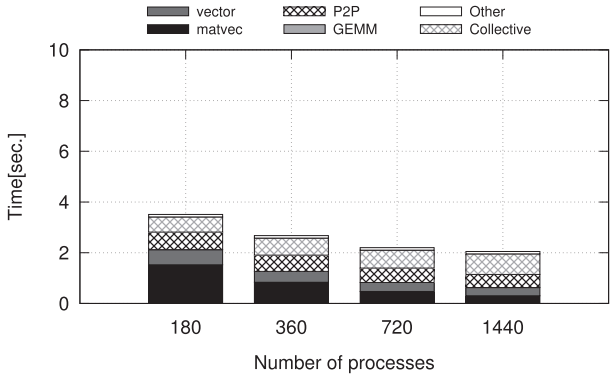

図 17　C-CG 法の実行時間の内訳：3 次元 Poisson 方程式（240 × 240 × 240）

Fig. 17　The detailed breakdown of execution time of the C-CG method: 3D-Poisson problem ($240 \times 240 \times 240$).

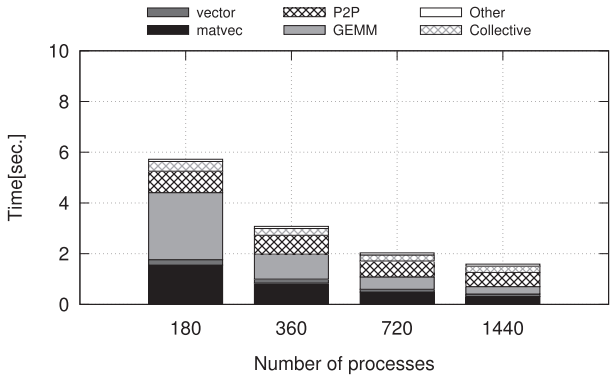

図 18　CBCG 法（$k = 10$）の実行時間の内訳：3 次元 Poisson 方程式（240 × 240 × 240）

Fig. 18　The detailed breakdown of execution time of the CBCG method ($k = 10$): 3D-Poisson problem ($240 \times 240 \times 240$).

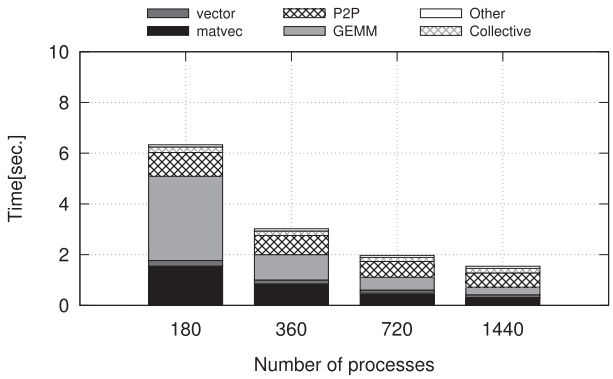

図 19　CBCGR 法（$k = 10$）の実行時間の内訳：3 次元 Poisson 方程式（240 × 240 × 240）

Fig. 19　The detailed breakdown of execution time of the CBCGR method ($k = 10$): 3D-Poisson problem ($240 \times 240 \times 240$).

クトル演算がどの並列数においても増えているため，全体の時間としては遅い結果となった．3 次元の問題の場合のメッセージ数は各プロセスが保持する立方体に隣接する面との通信となるため，k 回の SpMV で発生するメッセージ

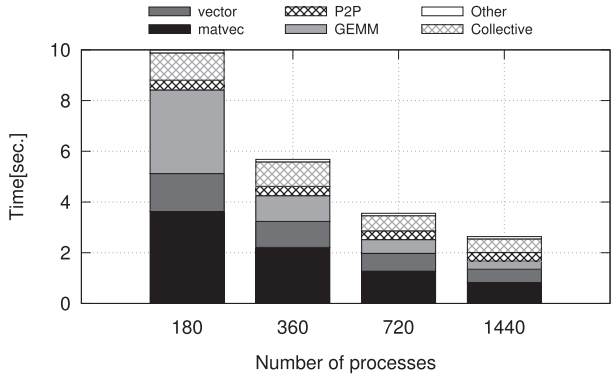

図 20　CBCGR-MPK 法（$k = 10$）の実行時間の内訳：3 次元 Poisson 方程式（$240 \times 240 \times 240$）

Fig. 20　The detailed breakdown of execution time of the CBCGR-MPK method ($k = 10$): 3D-Poisson problem ($240 \times 240 \times 240$).

数は $6k$ となる．MPK の場合は，立方体を拡張するとメッセージ数は 26 となるため k が 10 の場合約半分のメッセージ数となる．今回の実験では，全体的に通信が占める割合よりも演算が占める割合が大きかったため，通信削減の効果があまり見受けられない結果となった．

6. 結論

本論文では，通信削減 CG 法である CBCG 法に対して，CG 法の 1 反復あたり `MPI_AllReduce` を 1 回にする C-CG 法のアイディアを基に CBCGR 法を示し，さらに一対一通信を削減する MPK を適用した CBCGR-MPK 法についてそれぞれ，演算量と通信回数について示し，FX10 上での OpenMP/MPI の Hybrid 並列で 2 次元と 3 次元の Poisson 方程式を対象とした性能評価を行った．

2 次元 Poisson 方程式では，並列数が一定数以上になると，CG 法は集団通信がボトルネックとなり，実行時間が停滞するのに対し，CBCG 法は $1/k$ 倍となっているため，一定の並列数以上で CG 法よりも高速になる結果となった．

また，CBCGR 法の集団通信の回数は CBCG 法の 1/2 倍であるため，高並列時に CBCG 法よりも高速となる結果となった．CBCGR-MPK 法は 1 プロセスあたりが保持する領域サイズを拡張し，保持するため，プロセス間の重複計算により演算時間が増大してしまうが，一対一通信の時間が大幅に削減されたため，高並列時に CBCGR 法よりも高速になる結果となった．また，今回実験で使用した FX10 は 1 ノードに 1 プロセス立ち上げる Hybrid 並列で実行した場合にスカラデータに対するリダクションは，Tofu のハードウェアにより高速に処理されるため，C-CG 法は効果が出ず，CBCG 法においても一定の k 以上でなければ通信削減の恩恵が得られないことが分かった．

3 次元 Poisson 方程式では，一定の並列数以上で CBCGR 法が高速となる結果となった．また，CBCGR-MPK 法はプロセス間の重複計算により，良い結果が得られなかった．

今後の課題として，本論文で示した CBCGR 法の数値精度に関して，2 次元 Poisson 方程式で収束が悪化した原因と対策について検討する必要がある．今回実装をした 3 次元 Poisson 方程式に対する MPK は通信削減の効果よりもプロセス間の重複計算の増加が全体の実行時間への影響が大きく，プロセス間の重複計算を削減する必要がある．須田が提案している一般行列向けの MPK においてプロセス間の重複計算をなくす手法 [20] や黒田らの Multicolor ordering によるスレッド間の重複計算をなくす手法を分散環境向けに拡張することで，CBCG 法が CG 法よりもさらに優位になることが期待できる．この優位性を明確にするために，プロセス間の重複計算なしの MPK を適用した上で，より詳しく CBCG 法の有効性を実機上での計測を含め評価する必要がある．

その他の今後の課題として，問題規模を大きくした場合や，構造の複雑な問題，悪条件な問題で評価する必要がある．また，本論文では FX10 上での評価例のみであったため，他の機種上での評価も重要である．CBCG 法の実用化を想定すると，前処理の適用とその場合の性能評価も不可欠であり，前処理と MPK の組み合わせについても重要な課題である．

謝辞　査読者の皆さまから有益なコメントをいただきました．ここに感謝の意を表します．本研究の一部は ISPS 科学研究費 25330144, 15H02708 の助成を受けて行われた．

参考文献

[1] Hestenes, M. and Stiefel, E.: Method of Conjugate Gradient for Solving Linear Systems, *Journal of Research of the National Bureau of Standards*, Vol.49, No.6, pp.408–436 (1952).
[2] Chronopoulos, A. and Gear, C.: S-step Iterative Methods for Symmetirc Linear Systems, *Journal of Computational and Applied Mathematics*, Vol.25, pp.153–168 (1989).
[3] Ghysels, P. and Vanrose, W.: Hiding Synchronization Latency in the Preconditioned Conjugate Gradient Algorithm, *Parallel Computing*, Vol.40, pp.224–238 (2014).
[4] 本谷　徹，須田礼仁：k 段飛ばし共役勾配法：通信を削減することで大規模並列計算で有効な対称正定値行列連立 1 次方程式の反復解法，情報処理学会研究報告，Vol.2012–HPC–133, No.30 (2012).
[5] Hoemmen, M.: Communication-avoiding Krylov Subspace Methods, PhD Thesis, University of California Berkeley (2010).
[6] 須田礼仁，本谷　徹：チェビシェフ基底共役勾配法，情報処理学会ハイパフォーマンスコンピューティングと計算科学シンポジウム，Vol.2013, p.72 (2013).
[7] 須田礼仁，李　聡，島根浩平：数値的に安定性な通信削減クリロフ部分空間法，計算工学講演会論文集，Vol.19, No.F–6–4 (2014).
[8] Kumagai, Y., Fujii, A., Tanaka, T., Hirota, Y., Fukaya, T., Imamura, T. and Suda, R.: Performance Analysis of the Chebyshev Basis Conjugate Gradient Method on the K Computer, *11th International Conference on Parallel Processing and Applied Mathematics* (2015).

[9] 熊谷洋佑，藤井昭宏，田中輝雄，須田礼仁：超高並列環境での通信削減を目的とした Chebyshev 基底共役勾配法の特性評価，情報処理学会研究報告，Vol.2014–HPC–145, No.17 (2014).
[10] Carson, E.: Communication-Avoiding Krylov Subspace Methods in Theory and Practice, Technical report, Electrical Engineering and Computer Science University of California at Berkeley (2015).
[11] Demmel, J., Hoemmen, M., Mohiyuddin, M. and Yelick, K.: Avoiding Communication in Sparse Matrix Computations, *IEEE International Parallel and Distributed Processing Symposium* (2008).
[12] Demmel, J., Hoemmen, M., Mohiyuddin, M. and Yelick, K.: Minimizing Communication in Sparse Matrix Solvers, *Proc. ACM/IEEE Conference on Supercomputing* (2009).
[13] Dehnavi, M.M., Kurdi, Y.E., Demmel, J. and Giannacopoulos, D.: Communication-avoiding Krylov Techniques on Graphic Processing Units, *IEEE Trans. Magnetics*, Vol.49, pp.1749–1752 (2013).
[14] 黒田勝汰，藤井昭宏，田中輝雄：Matrix Powers Kernel の共有メモリ環境への適用における Multicolor ordering による重複計算の削減，情報処理学会研究報告，Vol.2015–HPC–148, No.6 (2015).
[15] Barrett, R., Berry, M., Chan, T.F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C. and der Vorst, H.V.: *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*, SIAM, Philadelphia, PA (1994).
[16] Information Technology Center: The University of Tokyo (online), available from ⟨http://www.cc.u-tokyo.ac.jp/⟩ (accessed 2015-12-01).
[17] Nakajima, K.: OpenMP/MPI Hybrid Parallel Multigrid Method on Fujitsu FX10 Supercomputer System, *IEEE International Conference on Cluster Computing Workshops* (2012).
[18] Deutsch, C. and Journel, A.: *GSLIB Geostatistical Software Library and User's Guide*, Oxford University Press, 2nd edition (1998).
[19] Intel: Intel MPI Benchmarks (online), available from ⟨https://software.intel.com/en-us/articles/intel-mpi-benchmarks⟩ (accessed 2016-03-02).
[20] 須田礼仁：一般の行列冪カーネルにむけて，日本応用数理学会 (2015).

熊谷 洋佑　（正会員）

1991 年生．2014 年工学院大学情報学部コンピュータ科学科卒業．2016 年同大学大学院工学研究科情報学専攻修士課程修了．2016 年（株）日立情報通信エンジニアリング入社．大規模数値計算アルゴリズムに興味を持つ．

藤井 昭宏　（正会員）

1975 年生．1999 年東京大学理学部情報科学科卒業．2004 年同大学大学院情報理工学系研究科コンピュータ科学専攻博士課程修了．博士（情報理工学）．2004 年工学院大学 CPD センタ講師となり，2015 年 4 月同大学情報学部コンピュータ科学科准教授，現在に至る．ハイパフォーマンスコンピューティング，階層的なアルゴリズム，不規則疎行列に関する並列処理等の研究に従事．IEEECS，電子情報通信学会各会員．

田中 輝雄　（正会員）

1958 年生．1983 年電気通信大学大学院情報数理工学研究科修士課程修了．2007 年同大学院情報システム学研究科博士課程修了．博士（工学）．1983 年（株）日立製作所中央研究所入所．2011 年工学院大学情報学部教授．専門は，大規模数値計算アルゴリズム．日本応用数理学会，IEEE-CS 各会員．

深谷 猛　（正会員）

1983 年生．2012 年名古屋大学大学院工学研究科計算理工学専攻博士課程（後期課程）修了．博士（工学）．神戸大学大学院システム情報学研究科特命助教，理化学研究所計算科学研究機構大規模並列数値計算技術研究チーム特別研究員を経て，2015 年より北海道大学情報基盤センター助教．線形計算アルゴリズムや高性能計算に関する研究に従事．日本応用数理学会会員．

須田 礼仁　（正会員）

1968 年生．1993 年東京大学理学系研究科情報科学専攻修士課程修了．1996 年東京大学博士（理学）．東京大学理学部助手，名古屋大学工学研究科講師・助教授，東京大学情報理工学系研究科准教授を経て 2010 年から東京大学情報理工学系研究科教授．高性能並列計算，数値アルゴリズムの研究に従事．日本応用数理学会会員．