Vol.2018-HPC-165 No.17
2018/7/31

Chebyshev基底通信削減CG法のマルチコア・メニーコア計算環境における性能評価

大島 聡史[1,a)]　藤井 昭宏[2]　田中 輝雄[2]　深谷 猛[3]　須田 礼仁[4]

概要：大規模並列計算環境における共役勾配法 (Conjugate Gradient Method、以下 CG 法) は集団通信 (MPI_Allreduce) の通信性能が性能ボトルネックとなることが指摘されている。この問題を解決するため、計算順序を変更して集団通信回数を削減する手法や、非同期通信を活用する手法など、いくつかの先進的な通信削減 CG 法が提案されている。我々は Chebyshev 基底通信削減 CG 法に着目し、「京」コンピュータなどいくつかの計算機環境にて性能評価を行ってきた。本稿では、最新のマルチコア CPU やメニーコアプロセッサを搭載したスーパーコンピュータシステムを対象計算機として複数の Chebyshev 基底通信削減 CG 法の性能を測定し、計算機環境の性能と得られた性能との関係について考察する。

1. はじめに

大規模な問題や複雑な問題を解くために、より高速な計算機環境が必要とされている。一方でムーアの法則による半導体集積度の向上が限界を迎えプロセッサ単体の性能向上が難しくなっている今日、高い性能を持つ計算機環境を構築するためには多数のプロセッサによる大規模分散並列処理が必要である。しかし多数のノードによる集団通信は長い時間を必要とすることがあり、大規模な計算を行う上での性能向上阻害要因となる可能性がある。ネットワークハードウェアの性能向上によりこれを解決することは、技術的な問題やコスト的な問題で容易ではなく、また物理的な限界も存在する。そのため、通信回数を削減するなど、計算アルゴリズム側での解決も重要となっている。

一方、多くのアプリケーションにおいて連立一次方程式 $Ax = b$ を解くことが必要とされており、正定値対称な行列を係数に持つ場合には反復解法である CG 法が広く用いられている。CG 法は 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) および 2 回の内積計算を必要とするアルゴリズムである。CG 法で用いる係数行列をブロック行分割で分散環境向けに並列化すると、ベクトルを各プロセスに分散して保持することとなる。そのため内積計算のたびにノード間でデータを集約・交換する必要があり、1 反復あたり 2 回の集団通信 (MPI_Allreduce) が必要となる。また、SpMV 計算のたびに、他のプロセスとの 1 対 1 通信によるベクトルデータの交換も必要である。使用するノード数 (MPI プロセス数) が増えると、1 対 1 通信の時間は大きく増加しない一方、集団通信の時間は大きく増加し性能ボトルネックとなりやすい。

そこで、CG 法の通信時間を削減する様々な手法が提案されてきた。我々は過去の研究においてそれらの幾つかを実装し、また新しい通信削減 CG 法アルゴリズムを提案し、「京」コンピュータや FX10 スーパーコンピュータシステムにおいて性能評価を実施してきた [1], [2]。本稿では、我々がこれまで利用してきた計算機環境よりも新しい世代のマルチコア CPU 環境およびメニーコアプロセッサ環境を実験環境として通信削減 CG 法の性能評価を行い、演算と通信の性能バランスの異なる環境における性能を評価する。

以下、2 章ではこれまでに提案されてきた通信削減 CG 法についてまとめる。3 章では対象問題と実験環境についてまとめ、4 章では性能評価結果を示し考察を行う。5 章はまとめの章とする。

2. 通信削減 CG 法

CG 法は対称正定値行列を係数とする連立一次方程式を解くための反復法アルゴリズムである。多くの計算科学アプリケーションの主要な計算部分として用いられており、大規模化・収束性改善・実行時間の短縮への要求は極めて大きい。

CG 法には 1 反復あたり 2 回の MPI_Allreduce を伴う内積計算が含まれる (Algorithm 1, 4 行目および 8 行目)。MPI_Allreduce は大規模な計算 (多数の計算ノードを用い

1 九州大学 情報基盤研究開発センター
2 工学院大学 情報学部
3 北海道大学 情報基盤センター
4 東京大学 情報理工学系研究科
a) ohshima@cc.kyushu-u.ac.jp

Algorithm 1 CG 法のアルゴリズム

$\boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0$
$\boldsymbol{p}_0 := \boldsymbol{r}_0$
for $i = 0, 1, 2, \cdots$ until convergence **do**
  Compute $\boldsymbol{p}_i^T A \boldsymbol{p}_i$
  $\alpha_i := \boldsymbol{r}_i^T \boldsymbol{r}_i / \boldsymbol{p}_i^T A \boldsymbol{p}_i$
  $\boldsymbol{x}_{i+1} := \boldsymbol{x}_i + \alpha_i \boldsymbol{p}_i$
  $\boldsymbol{r}_{i+1} := \boldsymbol{r}_i - \alpha_i A \boldsymbol{p}_i$
  Compute $\boldsymbol{r}_{i+1}^T \boldsymbol{r}_{i+1}$
  $\beta_i := \boldsymbol{r}_{i+1}^T \boldsymbol{r}_{i+1} / \boldsymbol{r}_i^T \boldsymbol{r}_i$
  $\boldsymbol{p}_{i+1} := \boldsymbol{r}_{i+1} + \beta_i \boldsymbol{p}_i$
end for

た計算) を行う際には長い時間を要する重たい処理となるため、回数の削減や、計算とのオーバーラップによる必要時間短縮のための取り組みが多数行われてきた。これらの手法は通信時間をどのように削減するかによって色々な呼称があるが、本稿ではまとめて「通信削減 CG 法」と呼ぶことにする。

Chronopoulos らは、CG 法の計算順序を変えることで 2 回の内積計算に必要な MPI_AllReduce を 1 回にまとめて行う CG 法 (以降、C-CG 法) を提案した [3](Algorithm 2)。さらに Ghysels らは C-CG 法をもとに非同期集団通信向けの Pipelined CG 法を提案した [4]。また Chronopoulos らは CG 法 s 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/s 回にする s-step CG 法も C-CG 法と同時に提案しており、本谷らも同様に MPI_Allreduce を 1/k 回にする k 段飛ばし CG 法 [5] を提案している。しかし、これらの手法は束ねる反復回数 (s または k の値) を大きくすると収束性の低下や発散が生じるといった不安定さが報告されている。

一方 Hoemmen は、Krylov 部分空間を多項式基底でまとめて生成する Communication-avoiding CG 法 (CA-CG 法) を提案している [6]。また須田らは、CA-CG 法の一種として、Chebyshev 多項式を基底として Krylov 部分空間をまとめて生成する Chebyshev 基底共役勾配法 (CBCG 法) を提案している [7] (Algorithm 3, 2 行目と 9 行目に対応する Krylov 部分空間は Algorithm 4)。CA-CG 法と CBCG 法については CG 法と同程度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。

さらに、通信削減 CG 法に現れる $(Ar, A^2r, ..., A^kr)$ 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 [8]。MPK の適用には通信回数の削減というメリットがある一方で、プロセス間での重複計算を追加で行わねばならないというデメリットもある。MPK は CBCG 法や CA-CG 法に対して適用が可能である。熊谷らは、「京」コンピュータおよび FX10 スーパーコンピュータシステムにおいて 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を明らかにした。さらに、CBCG 法に対して 1 反復あ

Algorithm 2 C-CG 法のアルゴリズム

$\boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0$
$\alpha_0 := \boldsymbol{r}_0^T \boldsymbol{r}_0 / \boldsymbol{r}_0^T A \boldsymbol{r}_0$
$\beta_{-1} := 0$
for $i = 0, 1, 2, \cdots$ until convergence **do**
  $\boldsymbol{p}_i := \boldsymbol{r}_i + \beta_{i-1} \boldsymbol{p}_{i-1}$
  $A\boldsymbol{p}_i ;= A\boldsymbol{r}_i + \beta_{i-1} A \boldsymbol{p}_{i-1}$
  $\boldsymbol{x}_{i+1} := x_i + \alpha_i \boldsymbol{p}_i$
  $\boldsymbol{r}_{i+1} := r_i + \alpha_i A \boldsymbol{p}_i$
  Compute $\boldsymbol{r}_{i+1}^T \boldsymbol{r}_{i+1}, \boldsymbol{r}_{i+1}^T A \boldsymbol{r}_{i+1}$
  $\beta_i := \boldsymbol{r}_{i+1}^T \boldsymbol{r}_{i+1} / \boldsymbol{r}_{i+1}^T \boldsymbol{r}_{i+1}$
  $\alpha_{i+1} := \boldsymbol{r}_{i+1}^T \boldsymbol{r}_{i+1} / (\boldsymbol{r}_{i+1}^T A \boldsymbol{r}_{i+1} - \beta_i \boldsymbol{r}_i^T \boldsymbol{r}_i / \alpha_i)$
end for

Algorithm 3 CBCG 法のアルゴリズム

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 convergence **do**
5:   Compute $Q_i^T A Q_i, Q_i^T \boldsymbol{r}_{ik}$
6:   $\boldsymbol{a}_i := (Q_i^T A Q_i)^{-1} Q_i^T \boldsymbol{r}_{ik}$
7:   $\boldsymbol{x}_{(i+1)k} := \boldsymbol{x}_{ik} + Q_i \boldsymbol{a}_i$
8:   $\boldsymbol{r}_{(i+1)k} := \boldsymbol{r}_{ik} - A Q_i \boldsymbol{a}_i$
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 $Q_i^T A S_{i+1}$
11:   $B_i := (Q_i^T A Q_i)^{-1} Q_i^T A S_{i+1}$
12:   $Q_{i+1} := S_{i+1} - Q_i B_i$
13: **end for**

Algorithm 4 Chebyshev 多項式を基底とした Krylov 部分空間の生成方法

$\eta := 2/(\lambda_{max} - \lambda_{min})$
$\zeta := (\lambda_{max} + \lambda_{min})/(\lambda_{max} - \lambda_{min})$
$\boldsymbol{s}_0 := \boldsymbol{r}_{ik}$
$\boldsymbol{s}_1 := \eta A \boldsymbol{s}_0 - \zeta \boldsymbol{s}_0$
for $j = 2$ to k **do**
  $\boldsymbol{s}_j := 2\eta A \boldsymbol{s}_{j-1} - 2\eta \boldsymbol{s}_{j-1} - \boldsymbol{s}_{j-2}$
end for
$S_i := (\boldsymbol{s}_0, \boldsymbol{s}_1, \cdots, \boldsymbol{s}_{k-1})$
$AS_i := (A\boldsymbol{s}_0, A\boldsymbol{s}_1, \cdots, A\boldsymbol{s}_{k-1})$

Algorithm 5 CBCGR 法のアルゴリズム

$\boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0$
$S_0 := (T_0(A)\boldsymbol{r}_0, T_1(A)\boldsymbol{r}_0, \cdots, T_{k-1}(A)\boldsymbol{r}_0)$
$\boldsymbol{a}_0 := (S_0^T A S_0)^{-1} S_0^T \boldsymbol{r}_0$
$B_0 := 0$
for $i = 0, 1, 2, \cdots$ until convergence **do**
  $Q_i := S_i - Q_{i-1} B_{i-1}$
  $AQ_i := AS_i + AQ_{i-1} B_{i-1}$
  $\boldsymbol{x}_{(i+1)k} := \boldsymbol{x}_{ik} + Q_i \boldsymbol{a}_i$
  $\boldsymbol{r}_{(i+1)k} := \boldsymbol{r}_{ik} - A Q_i \boldsymbol{a}_i$
  $S_{i+1} := (T_0(A)\boldsymbol{r}_{(i+1)k}, T_1(A)\boldsymbol{r}_{(i+1)k}, \cdots, T_{k-1}(A)\boldsymbol{r}_{(i+1)k})$
  Compute $S_{i+1}^T A S_{i+1}, Q_i^T A S_{i+1}, S_{i+1}^T \boldsymbol{r}_{(i+1)k}, Q_i^T \boldsymbol{r}_{(i+1)k}$
  $B_{i+1} := (Q_i^T A Q_i)^{-1} Q_i^T A S_{i+1}$
  $Q_{i+1}^T A Q_{i+1} := S_{i+1}^T A S_{i+1} - S_{i+1}^T A Q_i B_{i+1}$
  $Q_{i+1}^T \boldsymbol{r}_{(i+1)k} := S_{i+1}^T r_{(i+1)k} - B_{i+1}^T Q_i^T r_{(i+1)k}$
  $\boldsymbol{a}_{i+1} := (Q_{i+1}^T A Q_{i+1})^{-1} Q_{i+1}^T r_{(i+1)k}$
end for

たり 2 回の MPI_Allreduce を 1 回に削減する CBCGR 法 (Algorithm 5) を提案している。そのうえで、CBCGR 法に MPK を適用し FX10 スーパーコンピュータシステムにて性能評価を行い、2 次元 Poisson 方程式に由来する問題では性能向上が得られるケースがあった一方、3 次元 Poisson 方程式に由来する問題では重複計算の影響が大きく性能向上を得られなかったという結果を報告している [1], [2]。

一方 Carson も Hopper システムにおいて、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 法よりも高速となることを報告している [9]。Idomura らは、実際のアプリケーションで現れる問題に関して、CACG 法の性能を評価している [10]。また、非対称行列向けの通信削減型反復法である CA-GMRES の評価についても報告が行われている [11]。

以上のように、通信削減 CG 法には様々な種類が存在するが、基本的には、多ノード実行時に実行時間が大きくなる MPI_Allreduce の回数を減らし、その分を追加の計算や通信 1 回あたりの量の増加によって肩代わりするものである。そのため対象問題や実行する計算機システムの性能 (通信のレイテンシと計算性能のバランス) により効果の度合いが大きく変わる可能性がある。本稿では、最近の世代の 3 つのスーパーコンピュータシステムを用いて、いくつかの通信削減 CG 法の性能を測定し評価を行う。

3. 問題設定

3.1 実験環境

本稿では表 **3.1** に示すスーパーコンピュータシステムを用いて性能評価を行う。

マルチコア CPU 環境として、名古屋大学情報基盤センターに設置されている FX100 スーパーコンピュータシステム (以下 FX100)[12] および九州大学情報基盤研究開発センターに設置されている ITO スーパーコンピュータシステム (バックエンドサブシステム A)(以下 ITO)[14] を用いる。FX100 ではコンパイラとして富士通コンパイラ (Fujitsu C/C++ Compiler Driver Version 2.0.0 P-id: T01815-01 (Dec 1 2017 16:57:22))、MPI として富士通 MPI(FUJITSU MPI Library 2.0.0) を用いる。主なコンパイラオプションは`-Kfast,openmp` である。1 ノードに対して常に 1MPI プロセスを割り当て、1MPI プロセスあたり 32 スレッドによる OpenMP 並列処理を行う。ITO ではコンパイラとして Intel コンパイラ 2018 (icc 18.0.0)、MPI として MVAPICH2 2.2 を用いる。主なコンパイラオプションは`-O3 -qopenmp -no-prec-div -fp-model fast=2 -xCORE-AVX512` である。1 ノードに対して常に 1MPI プロセスを割り当て、1MPI プロセスあたり 1 ソケット 18 スレッド (NIC が接続されている CPU ソケットのみ使用) による OpenMP 並列処理を行う。

メニーコアプロセッサ環境として、JCAHPC に設置されている Oakforest-PACS(以下 OFP, 東大情報基盤センター側の制度で利用)[13] を用いる。コンパイラとして Intel コンパイラ 2018 Update 1 (icc 18.0.1)、MPI として Intel MPI Library 2018 Update 1 を用いる。主なコンパイラオプションは`-O3 -qopenmp -xMIC-AVX512` である。MPI_Allreduce の性能向上を狙って実行時に `I_MPI_FABRICS=tmi:tmi` を指定しているが、この指定の有無による性能差については未調査であり今後調査を行う予定である。1 ノードに対して常に 1MPI プロセスを割り当て、1MPI プロセスあたり 64 スレッドによる OpenMP 並列処理を行う。メモリモードは Flat モード、MCDRAM のみを使用している。クラスタリングモードは Quadrant モードである。(OFP は Quadrant モードのみ提供している。)

いずれの環境においても、今回は 16 ノードおよび 128 ノードの 2 つの並列度にて性能を測定し分析・評価する。これらの結果を用いて、将来的には千ノード規模以上での性能評価を目指す。

3.2 対象とする計算手法

本稿では以下の計算手法について性能を測定・比較する。各手法の具体的なアルゴリズム等の説明については、前章および前章にて提示しているそれぞれの文献を参照されたい。

(1) 通常の CG 法
(2) C-CG 法 (1 反復あたりの MPI_Allreduce を 2 回から 1 回に削減した CG 法)
(3) CBCG 法 (Chebyshev 基底 CG 法、Krylov 部分空間を 1 反復ごとに k 次元ずつ拡大しながら近似解を目指す手法)
(4) CBCGR 法 (CBCG 法における k 反復あたり 2 回の MPI_Allreduce を 1 回に削減した手法)
(5) CBCGR-MPK 法 (CBCGR 法に MPK を導入した手法)

いずれもプログラムは C 言語で記述されており、OpenMP と MPI によりハイブリッド並列化されている。

計算対象となる行列については、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 [16] から得られる行列を用いる。領域は 128x128x128、未知数の数は 2,097,152 である。16 ノード実行時には 4x2x2 に分割 (32x64x64 の領域ごとに分割) し、128 ノード実行時には 8x4x4 に分割 (16x32x32 の領域ごとに分割) する。収束条件は相対残差が 10^{-12} 未満となることとした。前処理には対角スケーリングを用いている。CBCG 法で必要となる最大・最小固有値はべき情報で推定した値と 0 を設定している。各種の時間計測には omp_get_wtime 関数を用いた。

3.3 MPI 性能の確認

性能評価の際の参照情報として、各実験環境における

表 1　実験環境

プロセッサ	Fujitsu SPARC64XIfx	Intel Xeon Gold 6154	Intel Xeon Phi 7150
搭載システム名	FX100	ITO サブシステム A	Oakforest-PACS (OFP)
ノードあたりソケット数	1	2	1
動作周波数	2.2 GHz	3.0 GHz - 3.7 GHz	1.4 GHz - 1.6 GHz
1 ソケットあたりコア数	32 + 2 †	18	68 ‡
1 ソケットあたり HPL 性能	約 1.0 TF	約 1.1 TF	約 1.6 TF
メモリ種別と 1 ソケットあたり容量	HMC 32 GB	DDR4 96 GB	MCDRAM 16 GB
1 ソケットあたり STREAM Triad 性能	105 GB/s	95 GB/s	495 GB/s
ノード間接続	Fujitsu Tofu2 100 Gbps 6 次元メッシュ/トーラス	Mellanox InfiniBand EDR 100 Gbps Fll Bisection BW Fat Tree	Intel Omni-Path 100 Gbps Full Bisection BW Fat Tree

† 「+ 2」コアは通信補助用コアであり計算には使わない
‡ 性能評価時には 64 コアのみを使用

MPI 性能を確認する。各ノード数における 1 対 1 通信および MPI_Allreduce 集団通信の性能の差を確認するのが目的であるため、測定項目は OSU Micro-Benchmarks 5.4.2 (以下 OSU) の osu_allreduce と、Intel MPI Benchmarks (以下 IMB)[15] の sendrecv と allreduce とする。sendrecv は MPI_Sendrecv 関数による 1 対 1 通信の性能を測定するベンチマークであり、CG 法における近接領域とのデータ交換を想定している。ただし、今回用いた実際のコードでは persistent 通信を用いた 1 対 1 通信が記述されており、sendrecv ベンチマークとは挙動が異なる可能性もある。より実際のコードに近いベンチマークとの比較については今後の課題とする。OSU の osu_allreduce および IMB の allreduce は MPI_Allreduce 関数による集団通信の性能を測定するベンチマークであり、CG 法における内積計算のためのデータ集約・交換に対応する。通信削減 CG 法における MPI_Allreduce で重要なのはレイテンシのため、osu_allreduce の値を主に確認し、バンド幅を測定する IMB の allreduce は参考値とする。

図 1 から**図 3** には osu_allreduce の結果、**図 4** から**図 6** には allreduce の結果、**図 7** から**図 9** には sendrecv の結果を示す。いずれも横軸はプロセス数 (=ノード数)、縦軸は実行時間である。*1

CG 法および C-CG 法の実装にて行われる allreduce は単純な内積計算のみであり、double 型変数 1 要素に対する通信のみが用いられる。一方 CBCG/CBCGR/CBCGR-MPK 法における allreduce は拡大幅 k に対して $k \times k$ 要素分程度 (k=20 の際に 20×20×8byte×2 組 +α=6KB 強) の通信を要する。ただし、通信削減 CG 法にて特に重要なのは MPI_Allreduce のレイテンシが削減されることとされている。そこで、allreduce の結果は、幾つかの結果をグラフに示した。

*1 ITO と FX100 ではスケジュールの都合上、一部のデータの収集ができなかったため、測定できた範囲のみ掲載する。

osu_allreduce の結果 (レイテンシ) は、測定された結果のうち 4Byte から 16KByte までを適宜間引いて折れ線グラフに示している。いずれの実験環境についても allreduce のレイテンシはプロセス数や転送サイズが増えるに従って増加しているが、その度合いには環境差がある。ITO と FX100 は 32 プロセスまでしか測定できなかったが、この範囲で比較する限り、最も高速なのは ITO、最も低速なのは OFP である。OFP は最も低速ではあるが、少なくとも 256 プロセスまでは急激に時間が増加することはないようである。

IMB の allreduce の結果は、折れ線グラフがベンチマーク結果の avg、誤差バーの上下端がそれぞれベンチマーク結果の max と min に対応している。この結果からも、やはり ITO, FX100, OFP の順に実行時間が短いことがわかる。0byte 以外の allreduce の実行時間はプロセス数に比例に近い形で増加し、概ね 32 ノード程度以上のノード数では実行時間のばらつきも目立つことが確認できる。

一方 sendrecv については、各 CG 法の実装にて行われる 1 対 1 通信のサイズに幅があることから、1Byte, 16KB, 64KB, 128KB, 256KB の 5 種類のサイズについて、allreduce 同様に avg スコアを中心に max と min を誤差バーとして示している。sendrecv については、同一の通信サイズであればプロセス数の影響は小さいことが確認できる。当然であるが通信サイズが大きくなればそれだけ通信時間も必要である。sendrecv と allreduce は同じ容量でも実行時間には大きな差があり、例えば OFP と FX100 の 128 プロセスに注目すると、64KB の allreduce と 8Byte の allreduce が同程度の時間を要する。

4. 性能評価

4.1 全体実行時間の比較

各実験環境における、MPI プロセス数 (=ノード数)16 と 128 の 2 つの実行条件における実行時間を比較する。

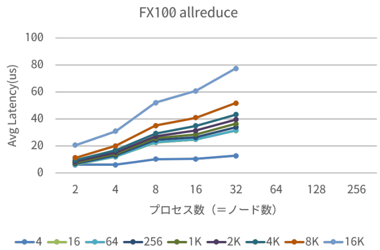


図 1　FX100 の osu_allreduce 実行時間

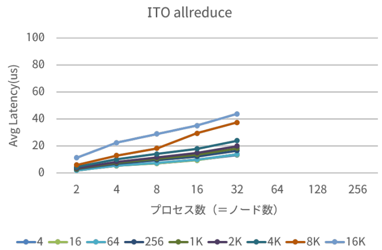


図 2　ITO の osu_allreduce 実行時間

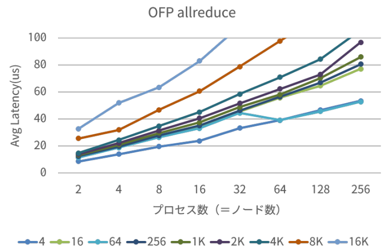
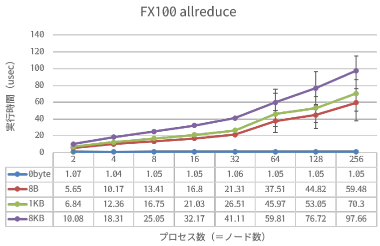
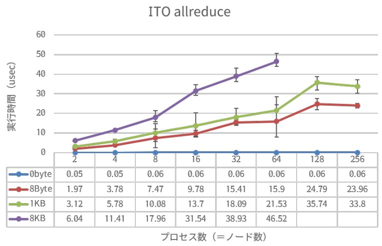


図 3　OFP の osu_allreduce 実行時間



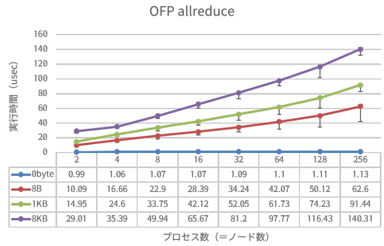
	2	4	8	16	32	64	128	256
0byte	1.07	1.04	1.05	1.05	1.06	1.05	1.05	1.05
8B	5.65	10.17	13.41	16.8	21.31	37.51	44.82	59.48
1KB	6.84	12.36	16.75	21.03	26.51	45.97	53.05	70.3
8KB	10.08	18.31	25.05	32.17	41.11	59.81	76.72	97.66

図 4　FX100 の IMB allreduce 実行時間



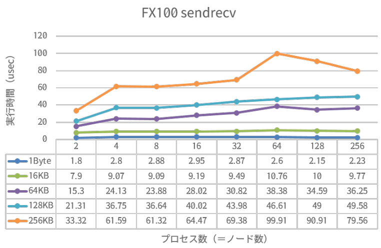
	2	4	8	16	32	64	128	256
0byte	0.05	0.05	0.06	0.06	0.06	0.06	0.06	0.06
8Byte	1.97	3.78	7.47	9.78	15.41	15.9	24.79	23.96
1KB	3.12	5.78	10.08	13.7	18.09	21.53	35.74	33.8
8KB	6.04	11.41	17.96	31.54	38.93	46.52		

図 5　ITO の IMB allreduce 実行時間



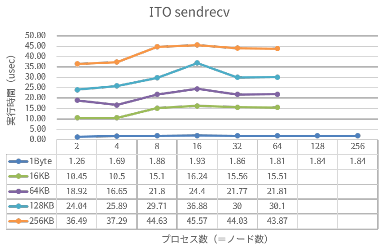
	2	4	8	16	32	64	128	256
0byte	0.99	1.06	1.07	1.07	1.09	1.1	1.11	1.13
8B	10.09	16.66	22.9	28.39	34.24	42.07	50.12	62.6
1KB	14.95	24.6	33.75	42.12	52.05	61.73	74.23	91.44
8KB	29.01	35.39	49.94	65.67	81.2	97.77	116.43	140.31

図 6　OFP の IMB allreduce 実行時間



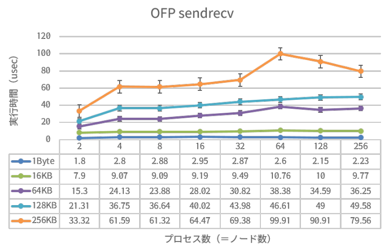
	2	4	8	16	32	64	128	256
1Byte	1.8	2.8	2.88	2.95	2.87	2.6	2.15	2.23
16KB	7.9	9.07	9.09	9.19	9.49	10.76	10	9.77
64KB	15.3	24.13	23.88	28.02	30.82	38.38	34.59	36.25
128KB	21.31	36.75	36.64	40.02	43.98	46.61	49	49.58
256KB	33.32	61.59	61.32	64.47	69.38	99.91	90.91	79.56

図 7　FX100 の sendrecv 実行時間



	2	4	8	16	32	64	128	256
1Byte	1.26	1.69	1.88	1.93	1.86	1.81	1.84	1.84
16KB	10.45	10.5	15.1	16.24	15.56	15.51		
64KB	18.92	16.65	21.8	24.4	21.77	21.81		
128KB	24.04	25.89	29.71	36.88	30	30.1		
256KB	36.49	37.29	44.63	45.57	44.03	43.87		

図 8　ITO の sendrecv 実行時間



	2	4	8	16	32	64	128	256
1Byte	1.8	2.8	2.88	2.95	2.87	2.6	2.15	2.23
16KB	7.9	9.07	9.09	9.19	9.49	10.76	10	9.77
64KB	15.3	24.13	23.88	28.02	30.82	38.38	34.59	36.25
128KB	21.31	36.75	36.64	40.02	43.98	46.61	49	49.58
256KB	33.32	61.59	61.32	64.47	69.38	99.91	90.91	79.56

図 9　OFP の sendrecv 実行時間

はじめに、疎行列反復法ソルバー全体の実行時間を比較する。**図 11** に FX100 の実行時間 (上段に 16 ノード、下段に 128 ノード、以下同様)、**図 12** に ITO の実行時間、**図 13** に OFP の実行時間をそれぞれ示す。いずれも縦軸は疎行列反復法ソルバー全体の実行時間、横軸には測定に用いた手法をとっている。CBCG/CBCGR/CBCGR-MPK 法に付記された数値は次元の拡大幅 k に対応する。実行時間のばらつきについても確認するため、いずれも何回か実行したうちの 3 回分の実行時間をグラフに示している。実行時間はいずれも全プロセスのなかでもっとも時間がかかったプロセスの時間を採用しているが、ソルバー全体で見た場合には MPI_Allreduce がバリアの役割を果たすことなどからプロセス間の時間差は軽微であった。また収束までの反復回数については、並列計算の順序による誤差が生じるため全計算機環境・全試行において完全に一致するわけではないが、使用する計算機やノード数の影響はほとんどなく、手法によって大きく変化する。OFP における全手法の収束回数を**図 10** に示す。ソルバーの反復回数自体は k が増加するに従って減少するが、ソルバーの反復回数に k を乗じた値 (CG 法と C-CG 法では k を 1 とみなす) を比べるとほぼ一定となっている。CBCG/CBCGR/CBCGR-MPK 法では元の CG 法における k 回の反復における k 回の MPI_Allreduce を 1 回に減らすとともにソルバーの反復

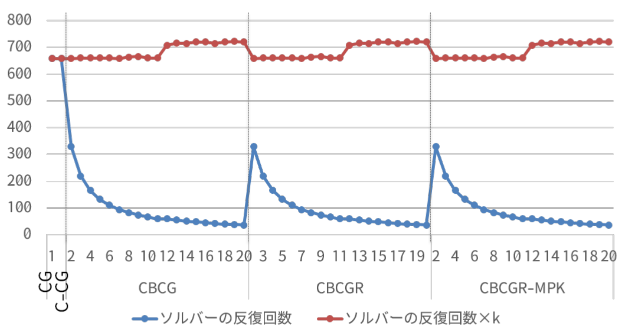


図 10　ソルバーの収束までの反復回数の比較

回数も k 分の 1 に減らしているが、ソルバーの反復回数 $\times k$ がほぼ一定ということは、計算順序や通信回数を変えても反復法ソルバーとしての収束にはほぼ影響がないことを示していると言える。

まず全体的な傾向としては、CBCGR-MPK の k の値が大きい際に実行時間が大きく増加しやすいことがわかる。この傾向は FX100 の 16 ノード及び 128 ノード、ITO の 16 ノード、OFP の 16 ノードで顕著である。OFP の 16 ノードについては CBCG および CBCGR についても同様に k の値が大きい際に実行時間が長い。また、実行回毎に異なるポイントで実行時間の低下が発生している。このような実行時間の低下は何らかのランダムな処理を含んでいたり

システム側のプロセスの割り込みなどで生じることがあるが一概には説明し難い問題であり、現時点では原因は明らかになっていない。各実験環境における最速ケースとその実行時間は**表 2** の通りである。全体的な傾向としては、16 ノードでは CG 法や C-CG 法が高速であるのに対して 128 ノードでは CBCGR 法が高速となっており、また計算機間の性能差については、ITO が最速で OFP が最遅という結果となっている。

表 2　最速ケース

	16 ノード	128 ノード
FX100	CG 法 2.50e-1 秒	CBCGR 法 (16) 8.59e-2 秒
ITO	C-CG 法 8.99e-2 秒	CBCGR 法 (4) 5.63e-2 秒
OFP	C-CG 法 3.06e-1 秒	CBCGR 法 (14) 1.24e-1 秒

4.2　実行時間内訳の調査

つづいて、実行時間の内訳についてさらに詳細に調査を行う。従来の研究から、各手法において長い実行時間を占める処理や多ノードでの実行時に影響が大きな処理はおおよそ明らかになっている。そこで、全体の実行時間およびそれらの主要な処理の時間を測定し、内訳を確認する。主要な処理としては、疎行列ベクトル積 **SpMV**(CBCGR-MPK 法では他の手法と傾向が異なることがわかっていたため、**SpMV_MPK** として区別した)、BLAS 関数呼び出しである **GEMM** および **GELS**、内積処理に必要な集団通信 **MPI_Allreduce**、SpMV 計算の際に近接ノードと 1 対 1 通信をするための通信時間 **SpMV_COMM**(SpMV 関連の通信) および **MPK_COMM**(SpMV_MPK 関連の通信) をとりあげ、それぞれ該当する計算や通信を omp_get_wtime 関数で囲んで差を確認するという方法で時間を測定した。疎行列反復法ソルバー全体の実行時間から上記の和を減じた残りは **etc.** とした。測定対象は 0 番プロセスとしており、試行ごとやプロセス間の実行時間にある程度のばらつきがあることから、上記の全体実行時間とは差が生じている部分がある点には注意されたい。

図 14 に FX100 の実行時間内訳 (上段に 16 ノード、下段に 128 ノード、以下同様)、**図 15** に ITO の実行時間内訳、**図 16** に OFP の実行時間内訳をそれぞれ示す。

全体的な傾向を確認すると、3 つの実行環境ともに共通して、CG 法, C-CG 法, CBCG 法, CBCGR 法の 4 つは 16 ノード実行では特に SpMV と GEMM といった計算に由来する時間の占める割合が大きいのに対して、128 ノード実行では MPI_Allreduce や SpMV_COMM といった通信に由来する時間の占める割合が長いことがわかる。これは 16 ノード実行では各ノードの担当領域の大きさがある程度大きいために計算時間の割合が多いのに対して、128 ノード実行では担当領域の大きさが十分小さくなったために計算時間は目立たなくなり、代わりに通信時間が目立ってきたと考えると納得できる結果である。CBCGR-MPK 法については、16 ノード実行と 128 ノード実行ともに SpMV_MPK が一定の大きな割合を占め、それに加えて 16 ノード実行では GEMM、128 ノード実行では MPI_Allreduce などの時間が目立つ結果となっている。さらに、ITO と OFP(特に ITO) の 128 ノード実行については MPI_Allreduce の時間が乱高下しているようにも見える。

ここで、計算手法によって反復法ソルバー全体の反復回数が異なる (CBCG/CBCGR/CBCGR-MPK 法では元の CG 法に対して $1/k$ 回程度となる) ことを考慮し、元の CG 法における 1 反復相当あたりの実行時間、すなわち、全体の実行時間を反復法の反復回数で除算したうえでさらに k によって除算した時間を比較することにする。図 14、図 15、図 16 をそれぞれ上記のとおり計算し、軸と軸ラベルを整理して横に並べて視覚的に比較しやすくしたものを**図 17** に示す。さらに、縦軸の範囲を統一して実行環境の違いによる性能差を分かりやすくしたものを**図 18**、図 18 から通信に関する部分だけを抽出したものを**図 19** に示す。

図 17 を用いて元の CG 法における 1 反復相当の時間を比較すると、16 ノード実行のグラフは SpMV や SpMV_MPK、GEMM といった計算に由来する割合が目立つのに対して、128 ノード実行ではそれらの占める割合が減少し、MPI_Allreduce や SpMV_COMM および MPK_COMM といった通信に由来する割合が大きくなることがあらためて確認できる。また、従来研究で既に示されているように、CBCGR-MPK 法における SpMV_MPK の割合は k の値が大きな場合を中心に常にかなりの割合を占めており、MPK による重複計算の影響が大きいことが再確認された。

一方、MPI_Allreduce の時間についてみてみると、CBCG 法および CBCGR 法と、CBCGR-MPK 法では状況が大きく異なっている。CBCG 法および CBCGR 法については、k の値を増やすに従って 1 反復あたりの所要時間が明確に減少している。16 ノードと 128 ノードを比べても、k の値がある程度大きい場合の MPI_Allreduce の所要時間には大きな差がなく、担当領域サイズの減少に伴って 1 対 1 通信の時間も減少していることもあり、通信時間全体で比較しても 128 ノードの方が 16 ノードより短いという期待通りの結果が得られている。

ところが、CBCGR-MPK 法については期待したような結果が得られておらず、k の値が増加するとむしろ MPI_Allreduce の要する時間が延びているように見えるという結果となった。さらに分析を行ったところ、これは負荷の不均衡と測定方法の組み合わせによる問題であることがわかった。すなわち、MPI_Allreduce 関数の前後の時刻を見て実行時間を確認した場合、MPI_Allreduce 関数への到達が早いプロセスは他のプロセスの到着を待つ時間も測定対象に入ってしまうため、測定結果として見える MPI_Allreduce の時間が不当に長くなってしまう。そ

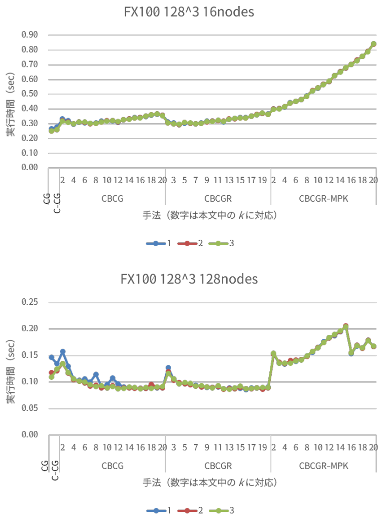


図 11　FX100 の実行時間

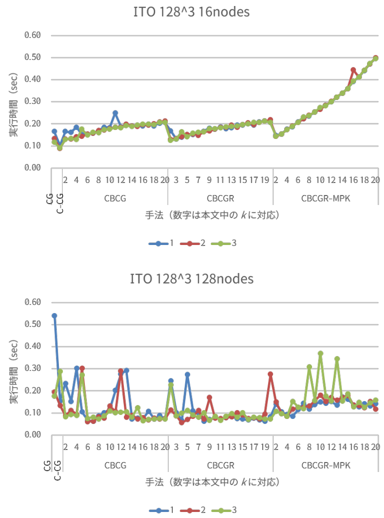


図 12　ITO の実行時間

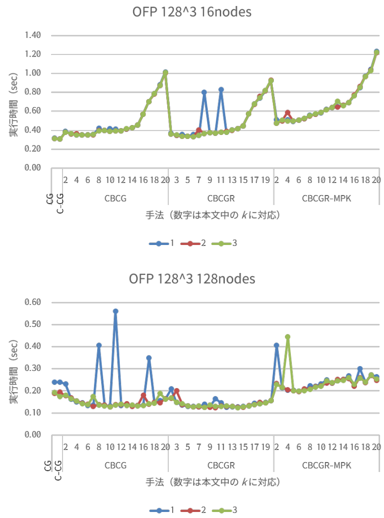


図 13　OFP の実行時間

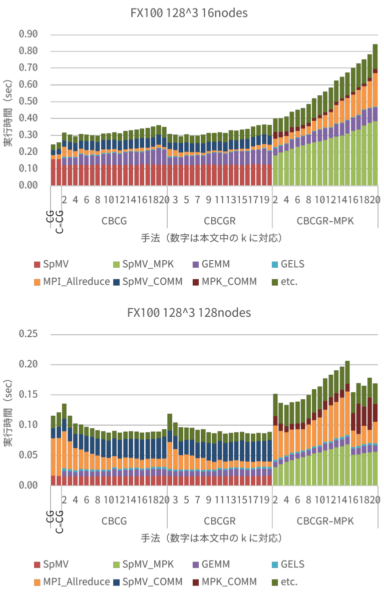


図 14　FX100 の実行時間内訳

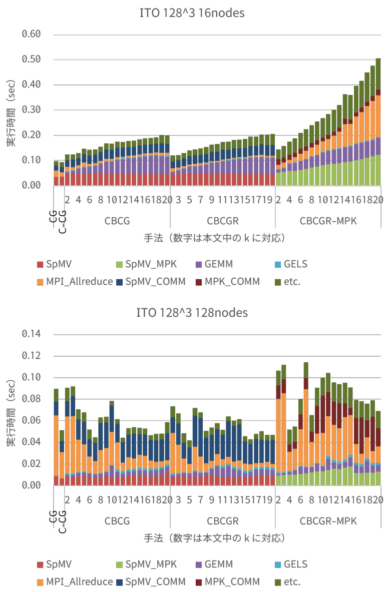


図 15　ITO の実行時間内訳

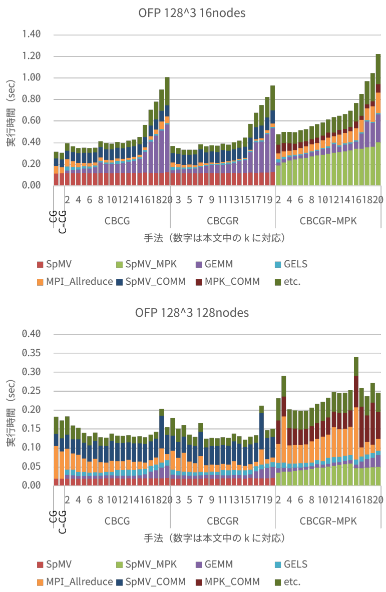


図 16　OFP の実行時間内訳

こで、MPI_Allreduce の前に MPI_Barrier を挿入し、バリア同期後から MPI_Allreduce 終了後までの時間を OFP にて測定し直したところ、図 **20** に示すように CBCG 法や CBCGR 法と同様の傾向が見られるようになった。

最後に、各計算環境の演算性能や通信性能と本性能評価にて得られた性能とを照らし合わせて考察を行う。今回用いた 3 つの計算機環境は、いずれも理論上のノード間通信性能が 100Gbps に揃っている。プロセッサあたりの演算性能とメモリ転送性能については、FX100 と ITO が HPL 性能・STREAM Triad 性能ともにほぼ同程度、OFP は他の 2 機種よりも大きく突出している。しかし実際の実行時間を見ると、全手法のうちで最も実行時間が短かったケースの実行時間が短い順に、ITO, FX100, OFP となった。CBCG/CBCGR/CBCGR-MPK 法の効果の程については、いずれも 16 ノード小規模実行では効果が発揮されず元の CG 法や C-CG 法の方が高速であったが、128 ノード実行では最適な k こそ環境により異なるものの、いずれも CBCGR 法が最も良い性能を得た。最も高速な手法とそれに対応する k の値が算出できると有用であると思われるが、MPI_Allreduce の実行時間に振れ幅があることもあり、現時点では達成できていない。

5.　おわりに

大規模並列計算環境における CG 法においては集団通信 (MPI_Allreduce) の実行時間 (レイテンシ) が並列高速化の阻害要因となることが指摘されており、それを解消するために様々な通信削減 CG 法が提案されている。我々は Chebyshev 基底通信削減 CG 法に着目して研究を進めており、本稿では従来研究よりも新しい世代のスーパーコンピュータシステム 3 種 (FX100, ITO, OFP) にて通信

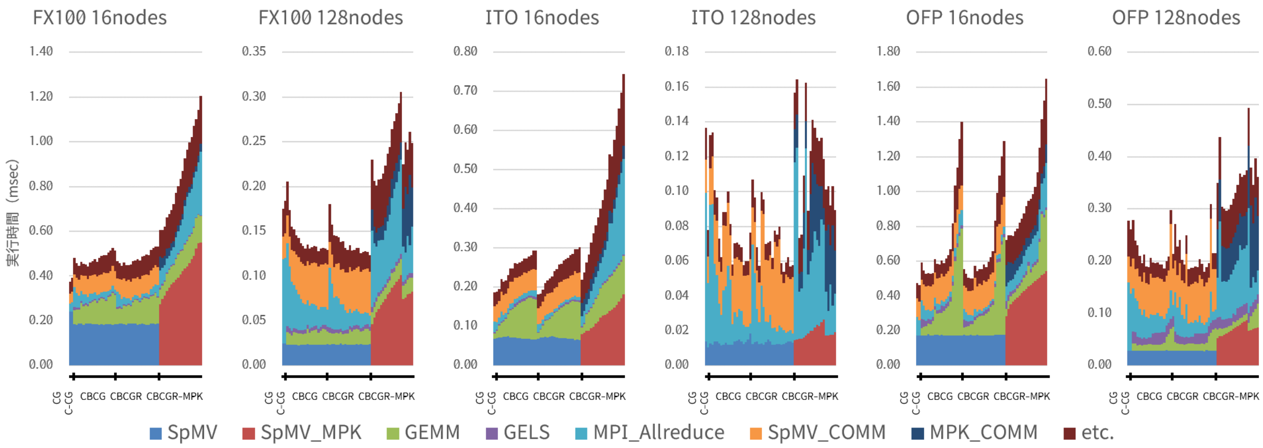


図 17　全実験条件の実行時間内訳の比較 (元の CG 法の 1 反復相当あたり)

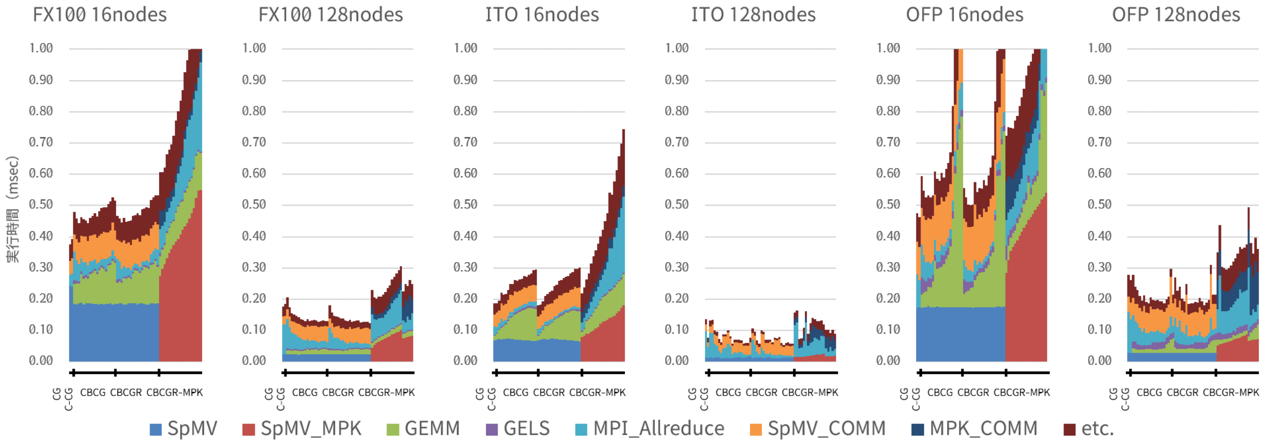


図 18　全実験条件の実行時間内訳の比較 (元の CG 法の 1 反復相当あたり)：縦軸統一版

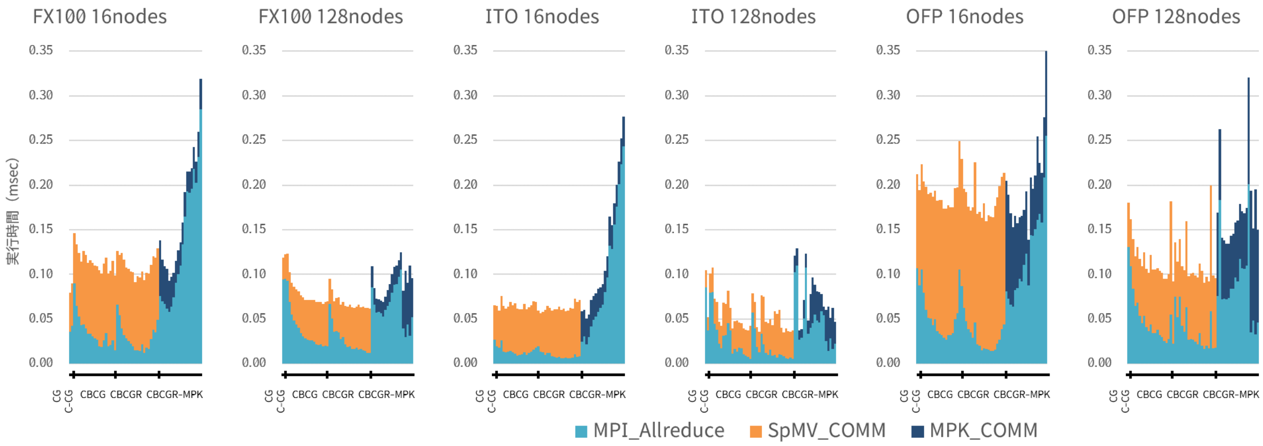


図 19　全実験条件の実行時間内訳の比較 (元の CG 法の 1 反復相当あたり)：通信時間のみ

性能の測定と評価を行った。現時点では 16 ノードと 128 ノードという小さめの規模での評価しか行えていないが、いずれの対象計算機においても、16 ノードでは CG 法や C-CG 法の性能が高いのに対して 128 ノードでは CBCG 法や CBCGR 法が高速になるという妥当な結果が得られた。

今回は条件を揃えて計算機環境同士の違いを確認したいという要求があったために 128 ノードまでの小規模な性能評価を行ったが、今後は 1000 ノード以上の環境で実行し、多ノード実行時の性能評価を行いたいと考えている。また、現在は対象としている問題 (行列) が非常に単純であり、大きく異なる問題を用いた場合の性能についても評価を行いたい。さらに、小規模実行時の性能評価結果を用い

Vol.2018-HPC-165 No.17
2018/7/31

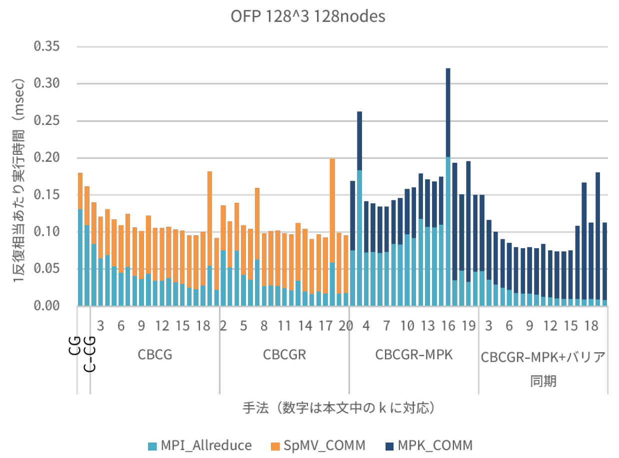


図 20 CBCGR-MPK 法に MPI_Allreduce 前にバリア同期を入れた場合を加えた通信時間の比較 (元の CG 法の 1 反復相当あたり)

て大規模実行時の性能予測ができれば実用上有用であることから、性能予測式の検討なども行いたい。

謝辞 本研究は JSPS 科研費 16H02823(基盤研究 (B) 通信回避・削減アルゴリズムのための自動チューニング技術の新展開) の助成を受けたものです．プログラムの提供や多くの有益な助言をいただいた熊谷洋佑氏に感謝します。

参考文献

[1] 熊谷洋佑, 藤井昭宏, 田中輝雄, 深谷猛, 須田礼仁: 共役勾配法への種々の通信削減手法の適用と評価, 情報処理学会論文誌コンピューティングシステム（ACS）, Vol.9, No.3, pp.1–13, 2016.

[2] Yosuke Kumagai, Akihiro Fujii, Teruo Tanaka, Yusuke Hirota, Takeshi Fukaya, Toshiyuki Imamura, Reiji Suda: Performance Analysis of the Chebyshev Basis Conjugate Gradient Method on the K Computer, Parallel Processing and Applied Mathematics (PPAM) 2015, Springer LNCS vol.9573, pp.74–85, 2016.

[3] A. T. Chronopoulos, C. W. Gear: S-step Iterative Methods for Symmetric Linear Systems, Journal of Computational and Applied Mathematics, Volume 25, Issue 2, pp.153–168, 1989.

[4] P. Ghysels, W. Vanrose: Hiding Global Synchronization Latency in the Preconditioned Conjugate Gradient Algorithm, Parallel Computing, Volume 40, Issue 7, pp.224–238, 2014.

[5] 本谷徹, 須田礼二: 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象正定値行列連立一次方程式の反復解法, 情報総理学会研究報告 Vol.2012-HPC-133, No.30, 2012.

[6] Mark Hoemmen: Communication-avoiding Krylov Subspace Methods, PhD Thesis, University of California at Berkeley, 2010.

[7] 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.

[8] James Demmel, Mark Hoemmen, Marghoob Mohiyuddin, Katherine Yelick: Avoiding communication in sparse matrix computations, 2008 IEEE International Symposium on Parallel and Distributed Processing, pp.1–12, 2008.

[9] Erin Carson; Communication-Avoiding Krylov Subspace Methods in Theory and Practice, Technical Report No.UCB/EECS-2015-179, University of California at Berkeley, 2015.

[10] Idomura Y., Ina T., Mayumi A., Yamada S., Imamura T.: Application of a Preconditioned Chebyshev Basis Communication-Avoiding Conjugate Gradient Method to a Multiphase Thermal-Hydraulic CFD Code, Supercomputing Frontiers (SCFA) 2018, Springer LNCS vol.10776, pp.257–273, 2018.

[11] Idomura, Y. and Ina, T. and Mayumi, A. and Yamada, S. and Matsumoto, K. and Asahi, Y. and Imamura, T.: Application of a Communication-avoiding Generalized Minimal Residual Method to a Gyrokinetic Five Dimensional Eulerian Code on Many Core Platforms, Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA'17), pp.1–8, 2017.

[12] 全体構成 — 名古屋大学 情報連携統括本部 `http://www.icts.nagoya-u.ac.jp/ja/sc/overview.html#FX10`

[13] Oakforest-PACS スーパーコンピュータシステム — 東京大学情報基盤センター スーパーコンピューティング部門 `https://www.cc.u-tokyo.ac.jp/supercomputer/ofp/service/`

[14] スーパーコンピュータシステム ITO — 九州大学情報基盤研究開発センター `https://www.cc.kyushu-u.ac.jp/scp/system/ITO/`

[15] Introducing Intel MPI Benchmarks — Intel Software `https://software.intel.com/en-us/articles/intel-mpi-benchmarks`

[16] Kengo Nakajima: OpenMP/MPI Hybrid Parallel Multigrid Method on Fujitsu FX10 Supercomputer System, 2012 IEEE International Conference on Cluster Computing Workshops, pp.199–206, 2012.