Vol.2016-HPC-153 No.3
2016/3/1

通信削減Jacobi法を前処理とした共役勾配法の性能評価

熊谷 洋佑[1]　野地 優希[1]　藤井 昭宏[1]　田中 輝雄[1]　須田 礼仁[2]

概要：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，その前処理として定常反復解法である Jacobi 法がある．Jacobi 法はすべての解要素の計算が終わるまで，解の更新を行わない特徴があり，並列計算において解要素間の依存性が生じず並列性が高い．しかし，係数行列をブロック行分割で並列化した場合，解の更新のたびに一対一通信が発生する．また，近年 $(A\boldsymbol{r}, A^2\boldsymbol{r}, \cdots, A^k\boldsymbol{r})$ の計算で発生する k 回の一対一通信を 1 回に削減する Matrix Powers Kernel が提案されている．本研究では，Jacobi 法の解の更新で発生する一対一通信を MPK の考えを基に 1 回に削減する通信削減 Jacobi 法（CA-Jacobi 法）を提案する．また，CA-Jacobi 法を前処理とした CG 法の 1 反復あたりの一対一通信を 1 回にする方法を示した．実際に 2 次元 Poisson 方程式を対象に CA-Jacobi 法を前処理とした CG 法を FX10（oakleaf-fx）において最大 1,024 ノード使用し実験を行い，通常の Jacobi 前処理付き CG 法よりも高速となる結果となった．

1. はじめに

近年，スーパー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，並列に処理を行な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．しかし，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レイテンシの削減が重要であると言われている．

一方，正定値対称な行列を係数にもつ連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を解くのに反復解法である共役勾配法（CG 法）が広く用いられる [1]．一般的に CG 法は前処理を施すことで収束が改善される．その前処理として定常反復解法である Jacobi 法がある．Jacobi 法はすべての解要素の計算が終わるまで，解の更新を行わない特徴があり，並列計算において解要素間の依存性が生じず並列性が高い，しかし，係数行列をブロック行分割で並列化した場合，解の更新のたびに一対一通信が発生する．

通信削減を施した前処理として，Grigori らにより通信削減不完全 LU 分解（CA-ILU(0)）が提案されている [2]．これは，Demmel らにより提案されている $(A\boldsymbol{x}, A^2\boldsymbol{x}, \cdots, A^k\boldsymbol{x})$ の計算で発生する k 回の一対一通信を 1 回に削減する Matrix Powers Kernel（MPK）[3][4] の考えに基づいて導出されているアルゴリズムである．Yamazaki らはプロセス間での依存関係のない領域内だけに対して前処理をすることにより一対一通信が発生しない CA-Domain Decomposition Preconditioners を提案し，それを前処理とした CA-GMRES を CPU/GPU クラスタで実測し，前処理なしの CA-GMRES より高速になることが報告している [5]．

本研究では，MPK の考えを基に Jacobi 法の解更新で発生する一対一通信を削減する通信削減 Jacobi 法（CA-Jacobi 法）を提案する．さらに CA-Jacobi 法を前処理とした CG 法の一対一通信を 1 回にする方法を示す．本研究では，FX10（oakleaf-fx）[6] 上でキャッシュブロッキングによる効果を評価したが，速度向上しなかったため，通信削減に焦点当てた CA-Jacobi 法を前処理とした CG 法の実装を行い評価した．

以下，2 章で提案手法である CA-Jacobi 法について述べ，演算量とメッセージ数，メモリ量の比較を示す．3 章で前処理付き CG 法への適用について述べる．その後，4 章で FX10 上において CA-Jacobi 前処理付き CG 法の性能評価結果を示し，最後に 5 章でまとめと今後の課題について述べる．

2. 通信削減 Jacobi 法

2.1 Jacobi 法

Jacobi 法とは，連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ に対して，

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + D^{-1}(\boldsymbol{b} - A\boldsymbol{x}^{(k)}) \quad (1)$$

で定義される反復解法である．ここで，D とは，係数行列 A の対角行列である．Jacobi 法はすべての解要素の計算が

[1] 工学院大学
Kogakuin University
[2] 東京大学
The University of Tokyo

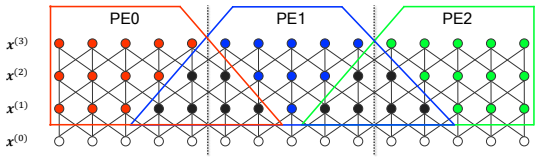

図 1 三重対角行列を 3 プロセスで分割したときの CA-Jacobi 法による要素間の依存関係.

終わるまで，解の更新を行わない特徴がある．そのため，並列計算において解要素間に依存性が生じず並列性が高いが，係数行列をブロック行分割で各プロセスに分散した場合に解の更新のたびに一対一通信が発生する．

2.2 通信削減 Jacobi 法

本節では，MPK の考えを基に Jacobi 法の解の更新で発生する一対一通信を削減する CA-Jacobi 法について述べる．

3 重対角行列での CA-Jacobi 法の 3 反復分の計算を 3 プロセスで分散したときの要素間の依存関係**図 1** に示す．図 1 の PE はそれぞれが独立したメモリ領域をもっている単位を示している．PE1 の領域は 5 であるが，CA-Jacobi 法では，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，一対一通信を 1 回で Jacobi 法 3 反復分の計算をすることが可能であるが，図の黒点が示すようにプロセス間での重複計算が発生する．

2.3 メモリ量と演算量，メッセージ数

ここで，Jacobi 法と CA-Jacobi 法のメモリ量と演算量，メッセージ数の比較を行なう．ここでのメモリ量は倍精度データの個数とし，演算量は浮動小数点の加算・乗算の回数 F，通信回数は一対一通信（`MPI_Send/Recv`）のメッセージ数 S とする．Jacobi 法では係数行列の構造に依存し，演算量と通信回数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なう．

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 2 次元 5 点差分での Jacobi 法の場合，通信が必要となるデータは各プロセス数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数は基本的に 4 となる．CA-Jacobi 法の場合，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，$\{\sqrt{N/P} + 2(k-1)\}^2$ となる．

また，メモリ量は Jacobi 法では右辺ベクトル $\boldsymbol{b}$ と解ベクトル $\boldsymbol{x}^{(k)}$，新しい解を格納するベクトル $\boldsymbol{x}^{(k+1)}$ の合計で 3 本のベクトルデータと係数行列 A が必要となる．

表 1 Jacobi 法と CA-Jacobi 法の演算（F）とメッセージ数（S）とメモリ量

	Jacobi	CA-Jacobi
Memory	$8N/P$	$8\{\sqrt{N/P} + 2(k-1)\}^2$
F	$(11N/P)k$	$11\sum_{i=0}^{k-1}\{\sqrt{N/P} + 2(k-1-i)\}^2$
S	$4k$	8

各プロセスが保持する係数行列の次元数を N/P とすると，合計で $8N/P$ 個のデータが必要となる．CA-Jacobi 法でも同様に各プロセスが保持する係数行列の次元数 $\{\sqrt{N/P} + 2(k-1)\}^2$ のベクトル 3 本と係数行列が必要となるため，$8\{\sqrt{N/P} + 2(k-1)\}^2$ となる．Jacobi 法と CA-Jacobi 法のメモリ量と演算量，メッセージ数を**表 1** に示す．メモリ量は 1 プロセスあたりのメモリ量とし，演算量とメッセージ数は Jacobi 法 k 反復分としている．

3. 前処理付き CG 法への適用

本研究では，集団通信の削減を施していない CG 法に対して CA-Jacobi 法を適用する方法を示す．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．

前処理付き CG 法では，$M \simeq A$ とし，$M^{-1}A\boldsymbol{x} = M^{-1}\boldsymbol{b}$ とすることで，係数行列の条件数が緩和され収束が改善する．前処理付き CG 法のアルゴリズムを Algorithm 1 に示す．詳細については参考文献 [7] に委ねる．Algorithm 1 の 2 行目と 9 行目で Jacobi 法を用いて，$\boldsymbol{z}$ を解くことになる．ここで，9 行目で $\boldsymbol{z}$ を解いたあとに 5 行目で $A\boldsymbol{p}$ の疎行列ベクトル積（SpMV）が発生する．CA-Jacobi 法では Jacobi 法の反復回数 k 回分の計算を行なうために係数行列を各プロセスが拡張することになる．ここで，SpMV で発生する一対一通信を削減するために，行列を $k+1$ 回分に拡張することで，Jacobi 法と SpMV で発生する一対一通信をまとめて削減することができる．CA-Jacobi 法を前処理とした CG 法の計算イメージを**図 2** に示す．図では Jacobi 法 2 反復と SpMV1 回の計算を示している．図の一段目で $\boldsymbol{r}_{i+1}$ の通信を行い，2，3 段目で Jacobi 法により $M\boldsymbol{z}_{i+1} = \boldsymbol{r}_{i+1}$ を解く．また，3 段目では次プロセスの $\boldsymbol{z}_{i+1}$ を保持しているため，Algorithm 1 の 10 行目で現れる $\boldsymbol{r}_{i+1}^T\boldsymbol{z}_{i+1}$ の内積計算をすることが可能である．ただし，内積計算で発生する集団通信は行われる．ここで，内積計算を行なうことにより，β_i の計算ができ，Algorithm 1 の 12 行目の $\boldsymbol{p}_{i+1}$ の更新が可能である．しかし，$\boldsymbol{p}_{i+1}$ は隣接領域分も更新を行うため，ここでもプロセス間の重複計算が発生する．そして，図の 4 段目で $\boldsymbol{p}_{i+1}$ の隣接領域分保持しているため，$A\boldsymbol{p}_{i+1}$ の SpMV を一対一通信なしで計算することができる．したがって，CA-Jacobi 前処理付き CG 法 1 反復あたりの隣接通信は 1 回で実行できる．

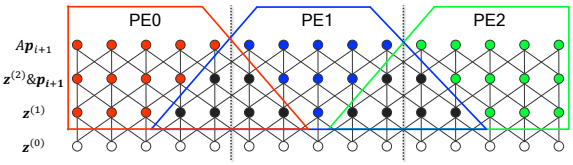

図 2 三重対角行列を 3 プロセスで分割したときの CA-Jacobi 法を前処理とした CG 法の要素間の依存関係.

4. 数値実験

4.1 実験環境と評価条件

Algorithm 1 The preconditioned CG method

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

FX10（oakleaf-fx）スーパーコンピュータシステム [6] 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，1,848GHz），32GB のメモリ（DDR3 SDRAM, 85GB/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（5GB/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，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,opnemp,ipo -lm`” 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 1,024 ノード使用した．Jacobi 法の加速係数は 1.0 とし，収束条件は相対残差が 10^{-12} とした．

対象とする問題は 2 次元 Poisson 方程式の領域が $2,048 \times 2,048$ とした．拡散係数は X が 1,024 未満の領域を 1.0，1,024 以上の領域を 100.0 とした．境界条件は $X = 0$ のときディリクレ条件とした．それ以外の境界は微分が 0 となるようなノイマン条件とした．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，X・Y 軸の分割数をそれぞれ P_X，P_Y とし，プロセス数 P が $P_X \times P_Y$ となるように設定した．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** に示す．係数行列のデータは Compressed row storage（CRS）形式 [8] で格納した．また，係数行列および反復中で使用するスカラー，ベクトルはすべて倍精度で保持している．

4.2 キャッシュブロッキングによる効果

MPK ではブロックの計算領域をキャッシュに収まる範囲にすることで GPU および共有メモリ環境で高速になることが知られている [9][10]．キャッシュブロッキングの性能評価のため 1 プロセスでの評価を行なう．今回は CA-Jacobi 法ではなくべき乗数を 2 から 21 まで変えたときの行列べき乗演算 $A^k\boldsymbol{r}$ の性能について議論する．本実験では，スレッド間での重複計算が発生しない MPK を実装した．MPK ではブロックごとの計算の順序を固定にすることによりスレッド間での重複計算を削減できることがわかっている [4]．三重対角行列における MPK でスレッド間の重複計算が発生しない計算イメージを**図 3** に示す．図 3 の Block0，1，2，の順で計算を実行することでそれぞれのブロック内での要素はメモリ上に近い位置にあるので，キャッシュヒット率向上が見込める．しかし，この方法では，ブロックサイズが大きくできないときは，並列性が低くなる可能性がある．2 次元 Poisson 方程式の場合は三重対角行列のときと同様に，Y 方向にも拡張することで実現できる．2 次元 Poisson 方程式における MPK でスレッド間の重複計算が発生しない計算イメージを**図 4** に示す．図 4 の Block0，1，2，3，の順で計算を実行することで三重対角行列と同様に重複なしで計算でき，k 回の SpMV と比較してキャッシュヒット率向上が見込める．

本実験では，行列のサイズはプロセス並列による分割を考慮し，$N = 64 \times 64, 128 \times 128, 256 \times 256, 512 \times 512$ の 4 つの領域サイズの問題を使用し，MPK の X 方向のブロックサイズを Xbs，Y 方向を Ybs とし，ブロックサイズを 16 から領域サイズの一辺の長さに変えて実験を行った．行列べき乗演算 $A^k\boldsymbol{x}$ における最適なブロックサイズでの MPK の k 回の SpMV に対する速度向上率を**図 5** に示す．図が示すとおり，どの問題サイズにおいても MPK は k 回の SpMV よりも性能が良くなることはなかった．$N = 512 \times 512$ での $A^{20}\boldsymbol{x}$ の計算におけるブロックサイズごとの MPK の性能を**図 6** に示す．Xbs が小さいときは性能があまり出ず，Xbs=512，Ybs=512 のときが一番高い性能が出ている結果となった．FX10 では 16 スレッドで並列化を行なうため，ブロックサイズが小さいときは並列化効率があまり良くないために性能が低い結果となった．

この結果から今回実験に使用した FX10 ではキャッシュブロッキングの性能向上がなかったため，CA-Jacobi-CG 法にはキャッシュブロッキングを適用していないもので評価を行なうことにする．

4.3 通信削減による効果

Jacobi-CG 法と CA-Jacobi-CG 法の Jacobi 法の適用回数ごとの収束に要した反復回数を**表 3** に示す．Jacobi 法

表 2 本実験におけるグリッドの分割数と各プロセスの領域サイズ.

N	P	P_X	P_Y	the shape of local domain per process
$4,194,304 = 2,048 \times 2,048$	16	4	4	$262,144 = 512 \times 512$
	32	8	4	$131,072 = 256 \times 512$
	64	8	8	$65,536 = 256 \times 256$
	128	16	8	$32,768 = 128 \times 256$
	256	16	16	$16,384 = 128 \times 128$
	512	32	16	$8,192 = 64 \times 128$
	1,024	32	32	$4,096 = 64 \times 64$

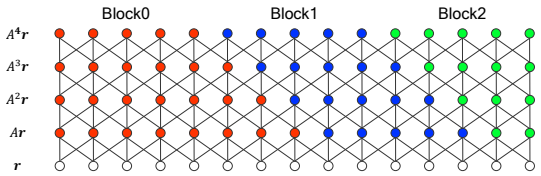

図 3 三重対角行列におけるスレッド間の重複計算が発生しない MPK の計算イメージ.

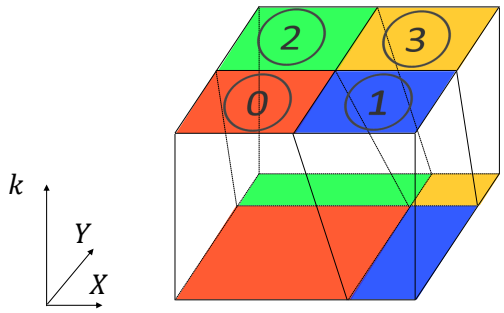

図 4 2 次元 Poisson 方程式でのスレッド間の重複計算が発生しない MPK の計算イメージ.

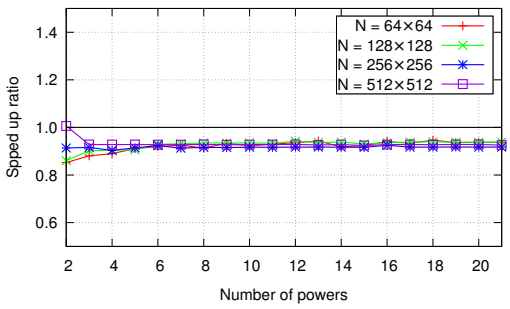

図 5 行列べき乗演算 $A^k\boldsymbol{x}$ における最適なブロックサイズでの MPK の k 回の SpMV に対する速度向上率.

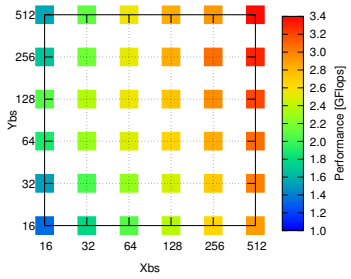

図 6 $A^{20}\boldsymbol{x}$ の計算におけるブロックサイズごとの MPK の性能. $N = 512 \times 512$.

の適用回数を増加させることで収束に要する反復回数が減少する結果となった．また，Jacobi-CG 法から CA-Jacobi-CG 法に変わったことによる誤差の影響についてはあまり受けていない結果となった．しかし，適用回数が奇数の場合に反復回数が増加している原因と調査については今後の課題である．

次に収束に要した時間について議論する．**表 4** に並列数ごとの最適な適用回数のときの Jacobi-CG 法，CA-Jacobi-CG 法の収束に要した時間を示す．この結果からどの並列数においても CA-Jacobi-CG 法が優位であることがわかる．**図 7** に 16 プロセス，**図 8** に 1,024 プロセスのときの Jacobi-CG 法と CA-Jacobi-CG 法の収束に要した時間をそれぞれ示す．16 プロセス，1,024 プロセスともに CA-Jacobi-CG 法が大半の Jacobi 法の適用回数において高速となる結果となった．また，Jacobi 法の適用回数を多くすることで，Jacobi-CG 法と CA-Jacobi-CG 法の収束時間の差が大きくなることがわかる．

それぞれの演算時間と通信時間の比較をするために収束に要した時間の内訳について述べる．16 プロセスと 1,024 プロセスのときの Jacobi 法と CA-Jacobi 法の収束に要した時間の内訳を**図 9** と**図 10** にそれぞれ示す．時間の内訳は時間取得関数（`omp_get_wtime`）の直前にバリア同期（`MPI_Barrier`）を挟み計測を行った．そのため，表 4 や図 7，図 8 の時間とは異なる．図の SpMV は CG 法で現れる SPMV，Preconditioned は Jacobi 法の計算，Vector は CG 法で現れる axpy と内積計算，P2P は SpMV と Jacobi 法で発生する一対一通信，Collective は内積計算で発生する集団通信，Other は実行時間から上記の時間を引いたものと

表 3 Jacobi-CG 法と CA-Jacobi-CG 法の収束に要した反復回数.

	Jacobi-CG	CA-Jacobi-CG
1	7696	7696
2	3832	3832
3	4448	4449
4	2707	2707
5	3448	3448
6	2210	2210
7	2916	2916
8	1914	1914
9	2574	2573
10	1709	1709
11	2329	2329
12	1559	1559
13	2143	2143
14	1443	1443
15	1996	1996
16	1348	1348
17	1875	1876
18	1270	1270
19	1775	1775
20	1204	1204

なっている．また，図の () で囲まれている Jacobi 法の適用回数は CA-Jacobi-CG 法を表している．16 プロセスでは，Jacobi-CG 法と CA-Jacobi-CG 法ともに Jacobi 法の計算が大半を占めていることがわかる．また，CA-Jacobi-CG 法はプロセス間の重複計算が発生するが，Jacobi-CG 法と比較すると Jacobi 法の計算時間が減少していることがわかる．これは，Jacobi 法の反復ごとに発生する一対一通信が発生しないことにより，計算速度が向上したと考えられる．1,024 プロセスでは，Jacobi-CG 法は一対一通信が占める割合が大きくなっているのに対して，CA-Jacobi-CG 法は Jacobi 法の適用回数を多くするほど一対一通信が削減されていることがわかる．また，Jacobi 法の計算時間は 16 プロセスのときよりも重複計算のオーバーヘッドが大きくなるため，CA-Jacobi-CG 法の方が少し増加している結果となった．

以上のことから，CA-Jacobi-CG 法は低並列では Jacobi 法の計算時間が減少，高並列時では一対一通信が削減され高速になる結果となった．そのため，CA-Jacobi 法は並列数に関わらず，通常の Jacobi 法よりも有効であることがわかる．

表 4 並列数ごとの最適な適用回数のときの Jacobi-CG 法，CA-Jacobi-CG 法の収束に要した時間．() 内は Jacobi 法の適用回数.

Number of processes	Jacobi-CG	CA-Jacobi-CG
16	16.96 (2)	14.75 (2)
32	8.91 (2)	7.78 (2)
64	4.90 (2)	4.35 (2)
128	2.64 (2)	2.48 (2)
256	1.86 (2)	1.72 (2)
512	1.38 (2)	1.37 (2)
1024	1.25 (2)	1.06 (6)

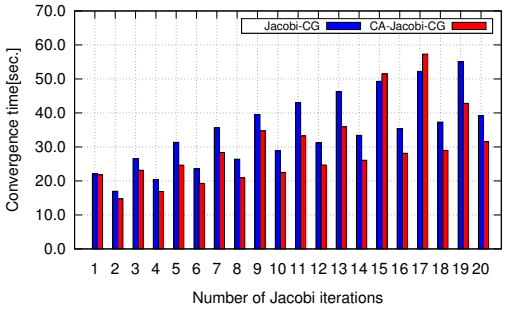

図 7 16 プロセスのときの Jacobi-CG と CA-Jacobi-CG の収束に要した時間.

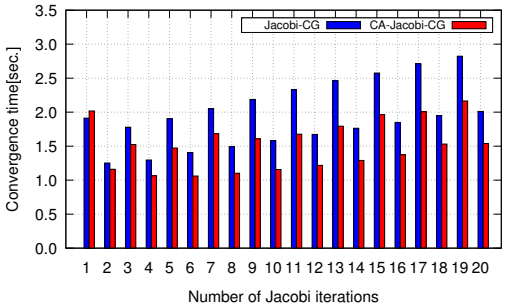

図 8 1,024 プロセスのときの Jacobi-CG と CA-Jacobi-CG の収束に要した時間.

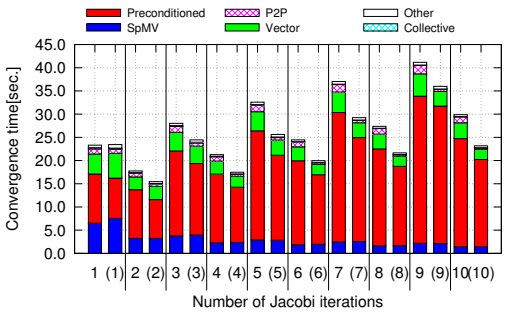

図 9 16 プロセスのときの Jacobi-CG 法と CA-Jacobi-CG 法の収束に要した時間の内訳.

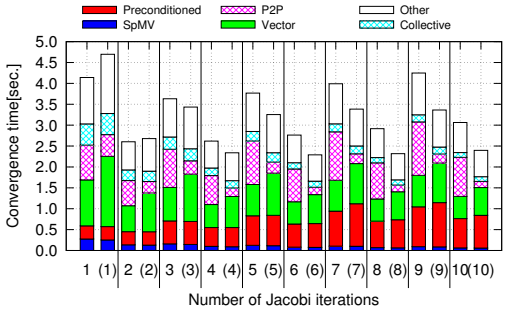

図 10 1,024 プロセスのときの Jacobi-CG 法と CA-Jacobi-CG 法の収束に要した時間の内訳.

5. 結論

本研究では，Jacobi 法に対して MPK の考えを基に一対一通信を削減した CA-Jacobi 法を提案し，Jacobi 法と CA-Jacobi 法のメモリ量，演算量，通信回数について示した．また，CA-Jacobi 法を CG 法の前処理に適用することで，CG 法で現れる SpMV で発生する一対一通信も同時に削減できることを示し，FX10 最大 1,024 ノード使用し OpenMP/MPI の Hybrid 並列で 2 次元 Poisson 方程式を対象とした性能評価を行った．

今回の実験環境である FX10 では MPK のキャッシュブロッキングが k 回の SpMV に対して高速になることはなかった．そのため，高並列での実験では一対一通信を削減したもののみで評価を行った．また，ストロングスケーリングでの実験では，低並列時ではプロセス間での重複計算が発生しているのにも関わらず Jacobi 法の計算時間が減少したことに収束に要した時間では CA-Jacobi-CG 法が高速になる結果となり，高並列時は，Jacobi 法の計算時間は増加したが，一対一通信が削減されたことにより収束に要した時間では CA-Jacobi-CG 法が高速になる結果となった．

今後の課題として，今回は 2 次元 Poisson 方程式の一例を主に評価したのみであったため，構造の異なる問題や悪条件の問題で評価する必要がある．また，2 次元の問題においてもプロセス間の重複計算が増加しているため，3 次元に拡張した場合にさらにに増加することが考えられる．そのため，須田が提案している MPK においてプロセス間の重複計算を削減する手法 [11] を適用し評価する必要がある．我々のいままでの研究で通信削減 CG 法である Chebyshev 基底 CG 法 [12] が高並列時に CG 法よりも高速になることがわかっている [13][14]．通信削減 CG 法に対して本研究での提案手法である CA-Jacobi 法を適用し評価する必要がある．その他の今後の課題として CA-ILU や CA-Domain Decomposition Preconditiners などの種々の通信削減前処理と今回提案した CA-Jacobi 法との比較も不可欠である．

謝辞 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] Grigori, L. and Moufawad, S.: Communication Avoiding ILU0 Preconditioner, *SIAM Journal on Scientific Computing*, Vol. 37, No. 2, pp. C217–C246.
[3] Demmel, J., Hoemmen, M., Mohiyuddin, M. and Yelick, K.: Avoiding Communication in Sparse Matrix Computations, *in IEEE International Parallel and Distributed Processing Symposium* (2008).
[4] Demmel, J., Hoemmen, M., Mohiyuddin, M. and Yelick, K.: Minimizing Communication in Sparse Matrix Solvers, *in Proceedings of the ACM/IEEE Conference on Supercomputing* (2009).
[5] Yamazaki, I., Rajamanickam, S., Boman, E. G. and Hoemmen, M.: Domain Decomposition Preconditioners for Communication-Avoiding Krylov Methods on a Hybrid CPU/GPU Cluster, *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*, pp. 933–944 (2014).
[6] Information Technology Center: The University of Tokyo (online), available from ⟨http://www.cc.u-tokyo.ac.jp/⟩ (accessed 2016-1-1).
[7] Saad, Y.: *Iterative Methods for Sparse Linear Systems*, Society for Industrial Applied Mathematics Philadelphia, PA, USA, 2nd edition edition (2003).
[8] 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).
[9] Dehnavi, M. M., Kurdi, Y. E., Demmel, J. and Giannacopoulos, D.: Communication-avoiding Krylov Techniques on Graphic Processing Units, *Magnetics, IEEE Transactions on*, Vol. 49, pp. 1749–1752 (2013).
[10] 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).
[11] 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).
[12] 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).
[13] 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).
[14] 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).