大規模多相流体解析向け省通信型マルチグリッド前処理付き共役勾配法

井戸村泰宏、小野寺直幸、山田進
日本原子力研究開発機構システム計算科学センター
山下晋
日本原子力研究開発機構原子力基礎工学研究センター
伊奈拓哉、今村俊幸
理化学研究所計算科学研究センター

1　はじめに

反復行列解法は大規模疎行列の連立一次方程式の解法として広く用いられており、そのスケーラビリティはエクサスケール計算における重要課題の一つとなっている。原子力分野では、原子炉における多相流体問題の数値流体力学（CFD: Computational Fluid Dynamics）解析にエクサスケール計算が必要となっている。このような大規模 CFD 解析では、反復行列解法に基づく陰解法ソルバが主要なコストを占め、コード全体のスケーラビリティと性能を左右する。クリロフ部分空間法 [1] は様々な分野で使用されている最も有力な反復行列解法の一つである。しかしながら、大規模クリロフソルバには大域的同期処理と収束特性劣化という 2 つの大きな課題が存在する。

クリロフ部分空間法では、各反復において基底ベクトルの生成と直交化が行われる。後者は内積処理を必要とし、大域的縮約通信とそれに伴う同期処理が発生する。最先端メニーコア環境では計算性能と通信性能のギャップが拡大しており、大域的同期処理の遅延が重要なボトルネックとなっている。この通信ボトルネックを緩和するために、省通信型（CA: Communication Avoiding）クリロフ部分空間法 [2–4] やパイプライン型クリロフ部分空間法 [5,6] といった通信削減手法が提案されてきた。前者は複数の基底ベクトルをまとめて計算することによって大域的同期処理を削減するのに対し、後者は非同期大域的通信によって同期処理のコストを隠蔽する。本研究では、CA クリロフ部分空間法を用いて大規模原子力 CFD 解析の強スケーリングの向上に取り組んできた [7–12]。

一方、近年の大規模 CFD 解析では問題のマルチスケール性が顕在化しており、行列の条件数が増大している。そのような悪条件問題では、クリロフソルバの反復回数が問題規模とともに増大する傾向にあり、前処理の高度化が必須となっている。マルチグリッド（MG: Multi-Grid）法はこのようなマルチスケール問題に対して最も有効な手法の一つであり、約 1 兆自由度に達する最近の大規模問題 [13,14] でも使われている。MG 前処理を用いた収束特性の向上により、同期処理を伴う反復計算の回数が大幅に削減される。しかしながら、この収束特性を維持しつつ MG 前処理における通信処理を削減することが残された課題となっている。

本稿では、2018 年度「大規模 HPC チャレンジ」において上記 2 つの課題解決に取り組んだ成果 [15] について概説する。この研究では 3 次元多相多成分熱流動 CFD コード JUPITER [16] 向けに省通信型マルチグリッド（CAMG）前処理を新たに開発し、従来の前処理付き共役勾配（P-CG: Preconditioned Conjugate Gradient）法やより進んだ CA クリロフ部分空間法との性能比較を Oakforest-PACS 上で行った。この結果、以下の成果が得られた。

- 前処理付きチェビシェフ反復（P-CI: Preconditioned Chebyshev Iteration）スムーザを用いて幾何的マルチグリッド（GMG: Geometric Multi-Grid）法に基づく新しい CAMG 前処理を開発した。P-CI スムーザにおいて、袖通信に混合精度処理を適用し、固有値計算に CA ランチョス法を用いることで CAMG 処理の省通信化を実現した。
- P-CG 法、前処理付き省通信型共役勾配（P-CACG）法、前処理付きチェビシェフ基底省通信型共役勾配（P-CBCG: Preconditioned Chebyshev Basis CACG）法、CAMG 前処理付き共役勾配（CAMGCG）法の計算性能と収束特性の系統的な比較結果が得られた。
- CAMGCG ソルバを約 900 億自由度の大規模多相流体解析に適用した。この結果、P-CG ソルバに比べて反復回数を約 1/800 に削減し、8,000 台の KNL に至る良好な強スケーリングを維持しつつ約 11.6 倍の高速化を達成した。

2 JUPITER コードにおけるクリロフソルバ

2.1 コード概要

JUPITER コード [16] は、非圧縮粘性流体を仮定した連続の式、ナビエ・ストークス方程式、エネルギー方程式によって原子炉内の溶融物の熱流動現象を記述する。燃料ペレット、燃料被覆管、チャンネルボックス、制御棒、炉内構造材から構成される多成分の気相、液相、固相の挙動を Volume of Fluid（VOF）関数の移流によって表現する。密度によって与えられる圧力ポアソン方程式の行列係数は気相と固相の密度比によって $\sim 10^7$ という極端なコントラストを示すために悪条件問題となる。このため、主要な計算コスト（約 90%以上）をポアソンソルバが占める。ポアソン方程式は直交座標系 (x, y, z) における 2 次精度中心差分（7 ステンシル）によって離散化される。この対称ブロック対角行列が与える連立一次方程式をクリロフソルバによって計算する。本研究のクリロフソルバは圧縮対角格納（CDS: Compressed Diagonal Storage）形式を採用している。メモリの間接参照を必要とする圧縮行格納（CSR: Compressed Sparse Row）形式と比較して、CDS 形式はブロック対角行列に対する効率的な直接メモリ参照を可能とする。並列化は (x, y, z) 方向の粗い 3 次元領域分割に MPI を適用し、各分割領域内における z 方向の細かい 1 次元領域分割に OpenMP を用いるハイブリッド並列モデルを採用する。この細かい分割領域の単位でスレッド並列処理を行えるように不完全 LU 分解に基づくブロックヤコビ（BJ: Block Jacobi）前処理 [1] を適用する。

2.2 P-CG 法

Algorithm 1 前処理付き共役勾配（P-CG）法

Input: $Ax = b$, Initial guess x_1

Output: Approximate solution x_i

1: $r_1 := b - Ax_1, z_1 = M^{-1}r_1, p_1 := z_1$
2: **for** $j = 1, 2, ...$ until convergence **do**
3: Compute $w := Ap_j$
4: $\alpha_j := \langle r_j, z_j \rangle / \langle w, p_j \rangle$
5: $x_{j+1} := x_j + \alpha_j p_j$
6: $r_{j+1} := r_j - \alpha_j w$
7: $z_{j+1} := M^{-1} r_{j+1}$
8: $\beta_j := \langle r_{j+1}, z_{j+1} \rangle / \langle r_j, z_j \rangle$
9: $p_{j+1} := z_{j+1} + \beta_j p_j$
10: **end for**

オリジナルの JUPITER コードでは圧力ポアソン方程式を BJ 前処理に基づく P-CG 法で計算していた。Algorithm 1 に示す P-CG 法では 1 回の反復処理が疎行列ベクトル積（SpMV）、BJ 前処理、2 回の内積処理、3 回のベクトル処理（AXPY）によって構成される。ここで、反復毎に SpMV は袖通信を、内積処理は 2 回の大域的縮約通信（All_reduce）をそれぞれ必要とする。line 4 の All_reduce は残差ベクトルのノルムを含めて 2 要素を転送し、line 8 の All_reduce は 1 要素を転送する。

2.3 P-CACG 法

Algorithm 2 は P-CACG 法 [2, 7] を示す。P-CACG 法では主要な計算は外部ループにおける省通信ステップ数 s 回の SpMV（line 5）と BJ 前処理（line 6）、グラム行列を構築する GEMM 処理（line 7）、および、内部ループの 3 項間漸化式の計算に必要な 3 つのベクトル処理となる。ここで、GEMM 処理のサイズは s に依存するため、演算強度も s とともに増大する。キャッシュブロッキングの最適化を適用すると 3 項間漸化式の係数も s 回再利用できるため、3 つのベクトル処理の演算強度も s の拡大によって向上する。SpMV は P-CG 法と同様に内部反復毎に袖通信が必要となるが、グラム行列計算は外部反復に

Algorithm 2 前処理付き省通信型共役勾配（P-CACG）法

Input: $Ax = b$, Initial guess x_1

Output: Approximate solution x_i

1: $z_0 := 0, z_1 := b - Ax_1$
2: $q_0 := 0, q_1 := M^{-1}z_1$
3: **for** $k = 0, 1, 2, ...$ until convergence **do**
4: $v_{sk+1} := z_{sk+1}$
5: Compute $\underline{V_k}$ $(v_{sk+1}, M^{-1}Av_{sk+1}, ..., (M^{-1}A)^s v_{sk+1})$
6: Compute $\underline{W_k}$ $(M^{-1}\underline{W_k} = \underline{V_k})$
7: $G_{k,k-1} := \underline{V_k^*} Z_{k-1}, G_{kk} := \underline{V_k^*}\underline{W_k}$
8: $G_k = \begin{pmatrix} D_{k-1} & G_{k,k-1}^* \\ G_{k,k-1} & G_{kk} \end{pmatrix}$
9: **for** $j = 1$ to s **do**
10: Compute d_{sk+j} that satisfies $Aq_{sk+j} = [Z_{k-1}, \underline{W_k}]d_{sk+j}$, $M^{-1}Aq_{sk+j} = [Q_{k-1}, \underline{V_k}]d_{sk+j}$
11: Compute g_{sk+j} that satisfies $z_{sk+j} = [Z_{k-1}, \underline{W_k}]g_{sk+j}$, $q_{sk+j} = [Q_{k-1}, \underline{V_k}]g_{sk+j}$
12: $\mu_{sk+j} := g_{sk+j}^* G_k g_{sk+j}$
13: $\nu_{sk+j} := g_{sk+j}^* G_k d_{sk+j}$
14: $\gamma_{sk+j} := \mu_{sk+j}/\nu_{sk+j}$
15: **if** $sk + j = 1$ **then**
16: $\rho_{sk+j} := 1$
17: **else**
18: $\rho_{sk+j} := (1 - \frac{\gamma_{sk+j}}{\gamma_{sk+j-1}} \cdot \frac{\mu_{sk+j}}{\mu_{sk+j-1}} \cdot \frac{1}{\rho_{sk+j-1}})^{-1}$
19: **end if**
20: $u_{sk+j} := [Q_{k-1}, \underline{V_k}]d_{sk+j}$
21: $y_{sk+j} := [Z_{k-1}, \underline{W_k}]d_{sk+j}$
22: $x_{sk+j+1} := \rho_{sk+j}(x_{sk+j} + \gamma_{sk+j}q_{sk+j}) + (1 - \rho_{sk+j})x_{sk+j-1}$
23: $q_{sk+j+1} := \rho_{sk+j}(q_{sk+j} + \gamma_{sk+j}u_{sk+j}) + (1 - \rho_{sk+j})q_{sk+j-1}$
24: $z_{sk+j+1} := \rho_{sk+j}(z_{sk+j} + \gamma_{sk+j}y_{sk+j}) + (1 - \rho_{sk+j})z_{sk+j-1}$
25: **end for**
26: **end for**

つき 1 回だけの All_reduce を行う。この All_reduce は $s(s+1)$ 要素の $G_{k,k-1}$ と $(s+2)(s+1)/2$ 要素の $G_{k,k}$ の上三角行列を転送する。これに加えて、収束確認のための残差ベクトル $r_{sk} = b - Ax_{sk+1}$ のノルム計算に外部反復あたり 1 回の All_reduce を必要とする。このため、P-CACG 法は外部反復あたり 2 回の All_reduce を必要とする。

2.4 P-CBCG 法

Algorithm 3 は P-CBCG 法 [4, 17] を示す。P-CBCG 法の主な計算コストは SpMV と BJ 前処理を含むチェビシェフ基底ベクトル生成（line 10）と残りの行列計算により与えられる。line 10 の SpMV は s 回の袖通信を必要とするのに対し、line 5、6、11 の行列計算では大域的縮約通信が行われる。このため、P-CBCG 法では s ステップにつき 2 回の All_reduce を必要とする。line 5、6 の All_reduce は $s(s+1)/2$ 要素の $Q_k^* A Q_k$ の上三角行列、s 要素の $Q_k^* r_{sk}$、および、1 要素の残差ベクトルノルムを転送し、line 11 の All_reduce は s^2 要素の $Q_k^* A S_{k+1}$ を転送する。

2.5 CAMGCG 法

Algorithm 1 の P-CG 法において BJ 前処理を CAMG 前処理で置き換える。ここで、JUPITER コードは直交格子系に基づくため、標準的な V サイクルの GMG 法 [1] を用いる（図 1）。スムーザとして BJ 前

Algorithm 3 前処理付きチェビシェフ基底省通信型共役勾配（P-CBCG）法

Input: $Ax = b$, Initial guess x_0

Output: Approximate solution x_i

$r_0 := b - Ax_0$
Compute S_0 $(T_0(AM^{-1})r_0, ..., T_{s-1}(AM^{-1})r_0\ )$
$Q_0 = S_0$
for $k = 0, 1, 2, ...$ until convergence **do**
  Compute $Q_k^* A Q_k$
  Compute $Q_k^* r_{sk}$
  $a_k := (Q_k^* A Q_k)^{-1} Q_k^* r_{sk}$
  $x_{s(k+1)} := x_{sk} + Q_k a_k$
  $r_{s(k+1)} := r_{sk} - AQ_k a_k$
  S_{k+1} $(T_0(AM^{-1})r_{s(k+1)}, ..., T_{s-1}(AM^{-1})r_{s(k+1)})$
  Compute $Q_k^* A S_{k+1}$
  $B_k := (Q_k^* A Q_k)^{-1} Q_k^* A S_{k+1}$
  $Q_{k+1} := S_{k+1} - Q_k B_k$
  $AQ_{k+1} := AS_{k+1} + AQ_k B_k$
end for

Algorithm 4 前処理付きチェビシェフ反復（P-CI）法

Input: $Ax = b$, Initial guess x_0, Approximate minimum/maximum eigenvalues of $A\tilde{M}^{-1}$, $\lambda_{\min}, \lambda_{\max}$

Output: Approximate solution x_i

$d := (\lambda_{\max} + \lambda_{\min})/2, c := (\lambda_{\max} - \lambda_{\min})/2$
$r_0 := b - Ax_0, z_0 := \tilde{M}^{-1} r_0, p_0 := z_0, \alpha_0 := 1/d$
for $i = 1, 2, ...$ until convergence **do**
  $x_i := x_{i-1} + \alpha_{i-1} p_{i-1}$
  $r_i := b - Ax_i$
  $z_i := \tilde{M}^{-1} r_i$
  $\beta_i := (\alpha_{i-1} c/2)^2$
  $\alpha_i := 1/(d - \beta_i/\alpha_{i-1})$
  $p_i := z_i + \beta_i p_{i-1}$
end for

処理を用いた P-CI 法 [18] を実装するが、この手法は内積処理を全く含まない。P-CI 法ではチェビシェフ多項式を用いて解ベクトルを向上することにより、反復法の収束特性を加速するが、チェビシェフ多項式の計算に前処理行列 $A\tilde{M}^{-1}$ の最小・最大固有値の計算が必要となる（Algorithm 4）。本研究ではこの固有値計算にニュートン基底を用いた前処理付き CA ランチョス法 [2] を用いた。また、収束確認のための大域的縮約通信を回避するためにスムーザの反復回数は初期の収束特性スキャン結果に基づき固定値を設定した。

CAMGCG 法において、計算強度の向上と袖通信の削減を図るために混合精度処理を採用した。ここで、精度の選択は各アルゴリズムの数値的特性に応じて最適化した。CG 法では収束極限近傍での残差ベクトル計算に倍精度が必要となる。CA ランチョス法では CA クリロフ部分空間法と同様に複数基底ベクトルの生成と直交化をまとめて処理するが、これらの基底ベクトルの線形独立性は丸め誤差に敏感であるため倍精度を必要とする。一方、P-CI スムーザは典型的に数 10 回の反復で得られる近似解を計算するため、単精度を利用可能である。このため GMG 前処理を単精度で処理し、CA ランチョス法と CG 法を倍精度で計算した。

CAMGCG 法の主要な計算コストは SpMV、BJ 前処理、および、3 回の AXPY で構成される P-CI 法が与える。これらは All_reduce を必要とする内積処理以外は P-CG 法の計算カーネルとほぼ同様である。このため、P-CI スムーザは P-CG ソルバをわずかに修正するだけで容易に実装できる。

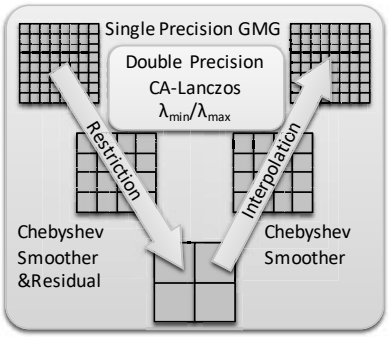

図 1: V サイクルの前処理付きチェビシェフ反復スムーザに基づく混合精度幾何的マルチグリッド前処理。固有値計算は省通信型ランチョス法によって行う。

3 カーネル性能評価

本節では P-CG、P-CACG、P-CBCG、および、CAMGCG ソルバの計算カーネル性能を分析する。ここでは小規模テスト問題サイズ $N = 104 \times 104 \times 265$ を KNL1 ノードで処理する。この問題規模は大規模問題におけるプロセッサあたりの典型的な問題規模に相当する。コンパイラやスレッドの設定は 4 節に示す数値実験と同じ条件を用いている。KNL における各カーネルの実行性能を修正ルーフラインモデル [19] と比較する。このモデルでは各カーネルの理論的な処理時間を浮動小数点演算とメモリアクセスの処理時間の和 $t_R = f/F + b/B$ によって評価する。ここで、f と b はカーネル中の浮動小数点演算数とメモリアクセス数を示し、F と B はプロセッサのピーク演算性能と STREAM メモリバンド幅を示す。

3.1 P-CG ソルバ

P-CG 法の主要計算カーネルは SpMV（line 3、4）、BJ（line 7、8）、および、Vector（AXPY、line 5、6、9）によって与えられる。ここで、SpMV と BJ に内積処理が含まれる。各カーネルのソースコードからカウントした浮動小数点演算数 f、メモリアクセス数 b、および、演算強度 f/b を表 1 にまとめる。圧力ポアソン方程式は 2 次精度中心差分を用いて計算するため、SpMV と BJ は比較的低い演算強度 $f/b < 0.2$ となる。さらに、残りの AXPY は $f/b = 0.1$ というメモリアクセスが支配的なカーネルとなる。このため、KNL の高いメモリバンド幅は P-CG ソルバの性能向上に大きく貢献する。Vector における AXPY はルーフラインモデルとの性能比が $t_R/t \sim 0.84$ というほぼ理想的な実行性能を達成しているが、SpMV と BJ におけるステンシル計算は $t_R/t \sim 0.6$ という性能劣化を示す。

3.2 P-CACG ソルバ

P-CACG 法の主要計算カーネルは SpMV（line 5、6）、BJ（line 5、6）、Gram（line 7、8）、および、3-term（line 20-24）となる。Gram と 3-term の演算強度は s に比例するため、P-CACG 法の演算強度は s に依存して変化する。ここで、Gram は $f = 2(s+1)(2s+1)/s$ および $b = 8(3s+2)/s$ とスケールし、3-term はキャッシュブロッキング最適化によって $f = (8s^2 + 12s + 2)/s$ および $b = 48(s+2)/s$ と変化する。表 1 では $s = 3$ のカーネル性能をまとめている。この省通信ステップ数は 4.2 節のベンチマーク問題が収束する上限値となっている。P-CACG 法における SpMV と BJ は内積処理を含まないため、P-CG 法よりも低い f と b を与える。Gram と 3-term では f が大きく増大するが、高い演算強度 $f/b > 0.5$ のために、P-CG 法からの処理時間の増大（約 1.53 倍）は f の増大（約 2.24 倍）に比べて小さい。

3.3 P-CBCG ソルバ

P-CBCG 法の主要計算カーネルはチェビシェフ基底計算 CB（line 10）と残りの行列計算 Matrix となる。P-CACG 法と同様に P-CBCG 法の演算強度も s に依存する。CB は $f = 2(9s+4)/s$ および $b = 8(4s+35)/s$

表 1: KNL1 ノードを用いたカーネル性能評価。ここで、各変数の定義は浮動小数点演算数 f [Flop/grid]、メモリアクセス数 b [Byte/grid]、演算強度 f/b、ピーク演算性能 F [Flops]、STREAM メモリバンド幅 B [Byte/s]、ルーフライン時間 $t_R = f/F + b/B$ [ns/grid]、処理時間 t [ns/grid]、実効性能 P [GFlops] となる。単精度（SP: Single Precision）の場合には F は倍精度（DP: Double Precision）の 2 倍となる。

Solver	Kernel	f	b	f/b	t	P	t_R/t
P-CG	SpMV	15.0	80.0	0.19	0.28	52.8	0.60
	BJ	20.0	128.0	0.16	0.46	43.3	0.59
	Vector	4.0	40.0	0.10	0.10	39.7	0.84
	Total	39.0	248.0	0.16	0.85	46.0	0.62
P-CACG ($s = 3$)	SpMV	13.0	80.0	0.16	0.24	54.6	0.72
	BJ	14.0	120.0	0.12	0.44	31.9	0.58
	Gram	18.7	29.3	0.64	0.13	142.9	0.51
	3term	41.7	80.0	0.52	0.49	84.2	0.36
	Total	87.3	309.3	0.28	1.30	67.0	0.52
P-CBCG ($s = 12$)	CB	30.6	228.7	0.13	0.89	34.5	0.55
	Matrix	93.2	83.3	1.12	0.63	147.1	0.32
	Total	123.8	312.0	0.40	1.52	81.4	0.45
P-CI (DP)	SpMV	14.0	88.0	0.16	0.33	42.0	0.56
	BJ	14.0	104.0	0.13	0.44	31.7	0.50
	Vector	4.0	40.0	0.10	0.10	38.4	0.81
	Total	32.0	232.0	0.14	0.88	36.4	0.56
P-CI (SP)	SpMV	14.0	44.0	0.32	0.24	57.5	0.39
	BJ	14.0	52.0	0.27	0.36	38.8	0.31
	Vector	4.0	20.0	0.20	0.05	75.4	0.80
	Total	32.0	116.0	0.28	0.66	48.7	0.38

とスケールし、Matrix は $f = (7s + 2)(s + 1)/s$ および $b = 40(2s + 1)/s$ と変化する。表 1 では $s = 12$ のカーネル性能をまとめている。この省通信ステップ数は 4.2 節のベンチマーク問題で用いられている。P-CBCG 法の浮動小数点演算数 f はさらに P-CG 法の約 3.17 倍まで増大するが、演算強度の向上によって処理時間の増大は約 1.79 倍に抑制されている。

3.4 CAMGCG ソルバ

CAMGCG 法の主要計算カーネルは SpMV（line 5）、BJ（line 6）、および、Vector（AXPY、line 4、9）を含む P-CI スムーザとなる。CAMGCG 法の数値特性は P-CG 法とほぼ同じとなり、処理時間も同程度となる。しかしながら、P-CI スムーザに単精度処理を適用するとメモリアクセス数 b が半分となり、ピーク演算性能 F が 2 倍となる。このため、ルーフラインモデルによると約 2 倍の性能向上が期待される。表 1 では Vector は約 2 倍の性能向上を示すが、SpMV と BJ の性能向上はそれを下回る。この結果、倍精度演算と比較した処理速度の向上は約 1.33 倍にとどまる。

4 数値実験

Oakforest-PACS において数値実験を実施した。Oakforest-PACS は 8,208 台の Intel Xeon Phi 7250 プロセッサ（KNL、68 コア、F=3046GFlops、B=480GB/s）および OmniPath（fat tree トポロジ、12.5GB/s）から構成される。コンパイラは Intel compiler 17.0.4（-O3 -qopenmp -xMIC-AVX512）および Intel MPI library 2017 を使用した。並列処理は 1 ノードあたり 1MPI プロセス、64 スレッドで行い、ハイパース

表 2: P-CG、P-CACG（$s = 3$）、P-CBCG（$s = 12$）、CAMGCG（SSOR/P-CI スムーザ）ソルバの収束特性。問題サイズ $N = 800 \times 500 \times 3,540 \sim 1.4 \times 10^9$ に対する収束特性と 500KNL を用いた時間ステップあたりの処理時間を示す。ここで、s は省通信ステップ数、ω は SSOR スムーザの加速パラメータを示す。CAMGCG ソルバは 5 レベルの V サイクルを使用し、SSOR スムーザと P-CI スムーザはそれぞれ各レベルで 30 回の内部反復を行った。

Solver	iterations/step	sec/step
P-CG	7063	34.1
P-CACG($s = 3$)	7065	32.8
P-CBCG($s = 12$)	7080	39.6
CAMGCG(SSOR, $\omega = 1.2$)	143	110.3
CAMGCG(SSOR, $\omega = 1.0$)	133	104.3
CAMGCG(SSOR, $\omega = 0.8$)	146	116.8
CAMGCG(P-CI, double precision)	33	14.5
CAMGCG(P-CI, mixed precision)	33	9.8

レッドは使用しなかった。また、MCDRAM （16GByte）と DDR4（96GByte）から構成される階層的なメモリをキャッシュモードで使用した。

4.1 収束特性

この数値実験では原子炉の燃料集合体 1 体における溶融デブリの非線形発展を計算した。問題サイズは $N = 800 \times 500 \times 3,540 \sim 1.4 \times 10^9$ とした。この問題サイズは先行研究 [7, 9, 16] でも使われた。P-CG、P-CACG、P-CBCG、および、CAMGCG ソルバの収束特性を表 2 にまとめる。ここで、収束条件は相対残差 $|b - Ax|/|b| < 10^{-8}$ によって与えた。文献 [7, 9] に従い、P-CACG および P-CBCG ソルバの省通信ステップ数はそれぞれ $s = 3$ および $s = 12$ とした。P-CACG は丸め誤差の影響により $s > 3$ で破綻するが、P-CBCG ではチェビシェフ基底を用いることでこのような数値不安定性が解決されており、$s > 12$ でも安定して収束する。ここでは、基底ベクトルを格納するメモリ使用量の制限によって $s = 12$ と選んだ。CAMGCG ソルバでは P-CI 法および赤黒オーダリングに基づく対称逐次加速緩和（SSOR: Symmetric Successive Over-Relaxation）法 [1] という 2 つの異なるスムーザアルゴリズムを用いて GMG 前処理を計算した。レベル数は最も粗いレベルの格子数が 10^6 以下となるように選んだ。各レベルのスムーザの並列化には最も細かい元のグリッドにおける (x, y, z) 方向の 3 次元領域分割モデルと同じ領域分割モデルおよび袖通信を用いた。この場合、レベルを粗くする毎に計算と袖通信はそれぞれ $\sim 1/2^3$ および $\sim 1/2^2$ に減少する。上記問題サイズに対し、 CAMGCG ソルバは 5 レベルの V サイクルを使用し、 各レベルの P-CI スムーザおよび SSOR スムーザの内部反復数は 30 回に固定した。

表 2 において、P-CACG ソルバと P-CBCG ソルバは P-CG ソルバとほぼ同じ反復回数となっている。これは、これらのアルゴリズムは数学的に等価であり、丸め誤差の影響による収束特性劣化が生じないように省通信ステップ数を選べば、同様の収束特性となることが期待されるためである。一方、CAMGCG ソルバは収束特性を大幅に向上する。 SSOR スムーザおよび P-CI スムーザを用いた CAMGCG ソルバは反復回数をそれぞれ $\sim 1/50$ および $\sim 1/200$ に削減する。これにより All_reduce の回数も大幅に削減される。しかしながら、SSOR スムーザを用いた CAMGCG ソルバは非効率なストライドメモリアクセスと赤黒で 2 回の袖通信が必要となるために全体処理時間が P-CG ソルバよりも長くなる。P-CI スムーザを用いた CAMGCG ソルバでは SSOR スムーザと比較して反復回数が $\sim 1/4$ となり、計算と通信の両方が削減される。これに加え、混合精度処理を適用することでさらに処理性能が加速される。ここで、単精度処理の P-CI スムーザによる収束特性劣化は全く見られない。

JUPITER コードにおいては、ポアソンソルバの行列係数が密度分布に依存して時間変化する。このため、P-CI スムーザは時間ステップ毎に CA ランチョス法による固有値計算を必要とする。固有値ソルバにおいて、最大・最小固有値は典型的には 10 回程度の反復で収束するが、今回の実装ではノルムの計算

による All_reduce を必要とする収束条件を検証せずに十分な精度を保証するために反復回数を 100 回（省通信ステップ数 $s = 5$）に固定した。この結果、固有値ソルバの処理時間は約 0.7 秒となったが、これは全体コストの 10%以下である。ここで、省通信型でないランチョス法を用いた場合には、固有値ソルバの処理時間は約 2 倍になる。以降のスケーリングテストでは P-CI スムーザに基づく混合精度 CAMGCG ソルバを使用する。

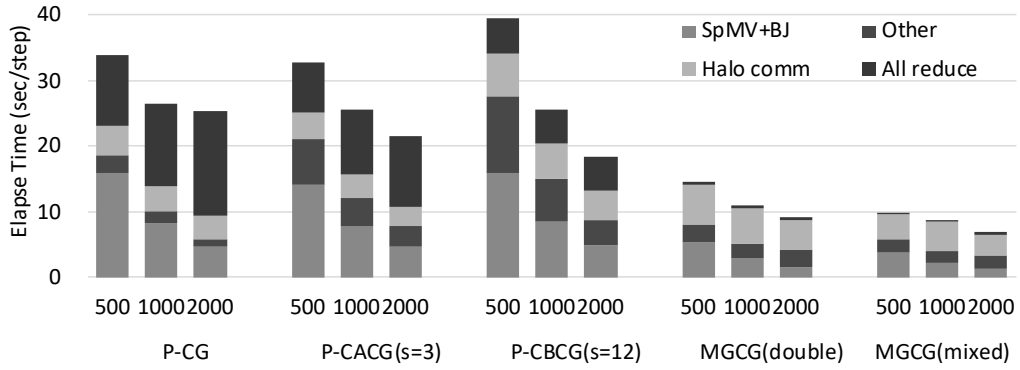

図 2: 500、1,000、2,000KNL を用いた P-CG、P-CACG（$s = 3$）、P-CBCG(s=12)、および、CAMGCG（倍精度/混合精度）ソルバ の強スケーリング。問題サイズ $N = 800 \times 500 \times 3,540 \sim 1.4 \times 10^9$ に対する 1 時間ステップあたりの処理時間のコスト分布を示す。

表 3: 図 2 におけるコスト分布（sec/step）。

Solver	Node	SpMV +BJ	Other	Halo Comm.	All Reduce	Total
P-CG	500	16.0	2.7	4.6	10.8	34.0
	1000	8.3	1.7	4.0	12.6	26.5
	2000	4.6	1.2	3.5	16.1	25.4
P-CACG ($s = 3$)	500	14.1	6.9	4.2	7.6	32.8
	1000	7.7	4.2	3.6	9.9	25.6
	2000	4.7	3.0	3.1	10.6	21.4
P-CBCG ($s = 12$)	500	15.8	11.7	6.7	5.3	39.6
	1000	8.4	6.6	5.4	5.2	25.7
	2000	4.8	4.0	4.4	5.2	18.4
MGCG (double)	500	5.3	2.8	6.1	0.3	14.5
	1000	2.9	2.2	5.4	0.4	11.0
	2000	1.6	2.7	4.4	0.5	9.2
MGCG (mixed)	500	3.8	2.0	3.9	0.1	9.8
	1000	2.2	1.8	4.3	0.3	8.8
	2000	1.2	2.2	3.2	0.3	6.9

4.2 14億自由度の強スケーリングテスト

P-CG、P-CACG、P-CBCG、および、CAMGCG ソルバの強スケーリングテスト結果を図 2 と表 3 にまとめる。この強スケーリングテストでは 500、1,000、2,000KNL を用いた。P-CG ソルバは All_reduce の影響による強スケーリングの大きな劣化を示す。P-CACG ソルバ（$s = 3$）は All_reduce のコストを削減し、強スケーリングを向上させる。しかしながら、 All_Reduce のコスト削減率は理論的評価 $1/s$ を大幅に下回り、2,000KNL における P-CG ソルバからの性能向上は約 1.18 倍にとどまる。 P-CBCG ソルバ（$s = 12$）は P-CACG ソルバより良い強スケーリングを示す。しかしながら、All_Reduce のコスト削減率はやはり $1/s$ を大幅に下回り、計算カーネルのコストも P-CACG ソルバより大きい。この結果、2,000KNL における P-CG ソルバからの性能向上は約 1.38 倍となる。CAMGCG ソルバの計算コストは最も細かいレベルの P-CI スムーザによって占められる。これは $30 \times 2 \times 33 = 1,980$ 回呼ばれるため、SpMV のコストは反復回数が約 7,000 回の P-CG 法に比べて $\sim 1/3$ となる。All_reduce の回数が約 7,000 回から 33 回に減少するため、主要な通信コストは P-CI スムーザの袖通信によって与えられる。今回の強スケーリングテストでは分割領域サイズを z 方向に 1/2 および 1/4 と細分化したため、MPI プロセスあたりの袖通信データサイズは 500KNL からそれぞれ 2/3 および 1/2 に減少する。しかしながら、袖通信のスケーリングはこの評価を下回る。この特徴は全てのソルバで見られるが、CAMGCG ソルバでは袖通信の影響が相対的に大きいため、強スケーリングが他のソルバより悪化する。ただし、全体処理時間自体は他のソルバに比べて大幅に短縮されている。混合精度処理を適用することで計算と通信の両方が加速され、倍精度処理と比べて約 1.34 倍の性能向上が得られる。この結果、2,000KNL における P-CG ソルバと混合精度 CAMGCG ソルバの全体処理性能比は約 3.7 倍となる。

4.3 900億自由度の強スケーリングテスト

本節では前節で使用したものと同じ多相流体問題において全方向に解像度を 4 倍した大規模問題 $N = 3,200 \times 2,000 \times 14,160 \sim 9.06 \times 10^{10}$ の数値実験を示す。この問題では適合型時間ステップ幅も自動的に 1/4 となる。密度分布はより一層シャープなコントラストを示すため、行列の条件数が増大する。これにより P-CG ソルバの収束特性は大幅に劣化し、反復回数は 26,475 回に達する。一方、CAMGCG ソルバは混合精度処理、倍精度処理の両方の場合において 32 回の反復で収束し、All_reduce の回数が P-CG ソルバの $\sim 1/800$ に減少する。ここで、CAMGCG ソルバは 7 レベル（各方向 4 倍のグリッド数のため 2 レベル増加）の V サイクルを使用し、各レベルで 50 内部反復を処理する。

強スケーリングテストにおいて、Oakforest-PACS 全系規模まで P-CG ソルバと混合精度 CAMGCG ソルバのテストを実施した。図 3 および表 4 に 2,000、4,000、8,000KNL における 2 つのソルバのコスト分布を示す。 2,000KNL における P-CG ソルバのコスト分布は、14 億自由度の問題と比較するとプロセスあたりの問題サイズが 64 倍になるため、より計算コストの比率が高くなり、強スケーリングが向上する。しかしながら、8,000KNL においては、コスト分布の約半分が All_reduce で占められる。CAMGCG ソルバでは最も細かいグリッドレベルにおける P-CI スムーザの SpMV は $50 \times 2 \times 32 = 3,200$ 回呼ばれる。この数字は P-CG ソルバの反復回数より一桁小さく、SpMV のコストは $\sim 1/10$ に減少する。しかしながら、2,000KNL では全てのレベルを含めた袖通信コストは P-CG ソルバの約 2 倍となる。このコスト増大は 2,000KNL のみで発生し、4,000KNL および 8,000KNL における袖通信コストは P-CG ソルバより小さくなる。このような通信性能劣化はメモリ使用量が MCDRAM のサイズを超える場合にしばしば見られ、その原因としてはキャッシュモードにおけるダイレクトマップキャッシュでのライン競合の可能性が考えられる。CAMGCG ソルバも計算量の増大によって 14 億自由度の問題よりも良い強スケーリングを示し、4,000KNL および 8,000KNL において、それぞれ 2,000KNL の約 2.6 倍および約 4.2 倍の性能向上を示す。P-CG ソルバと CAMGCG ソルバの性能差は 14 億自由度の問題よりも大幅に拡大し、8,000KNL で約 11.6 倍の性能向上を示す。

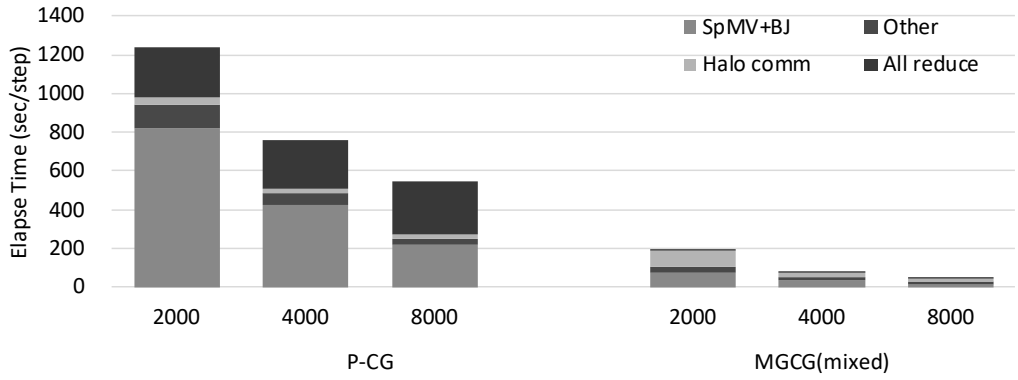

図 3: 2,000、4,000、8,000KNL を用いた P-CG ソルバと混合精度 CAMGCG ソルバの強スケーリング。問題サイズ $N = 3,200 \times 2,000 \times 14,160 \sim 9.06 \times 10^{10}$ に対して 1 時間ステップあたりの処理時間のコスト分布を示す。

表 4: 図 3 におけるコスト分布（sec/step）。

Solver	Node	SpMV +BJ	Other	Halo Comm.	All Reduce	Total
P-CG	2000	821.1	117.1	37.9	265.4	1241.6
	4000	424.8	63.0	22.9	247.3	758.1
	8000	218.3	32.6	21.4	270.8	543.1
MGCG (mixed)	2000	77.3	25.2	89.9	4.6	197.0
	4000	39.7	13.7	20.5	2.1	76.0
	8000	17.9	9.6	17.2	1.9	46.7

5 まとめ

本稿では JUPITER コードにおける圧力ポアソン方程式用に開発された新しい CAMGCG ソルバを紹介した。CAMGCG ソルバは P-CI スムーザと CA ランチョス法を用いて大域的縮約通信を回避するように設計されている。これに加えて、混合精度処理を最適化することにより、収束特性を劣化させることなく計算と通信の両方をさらに削減することに成功した。

CAMGCG ソルバの処理性能と収束特性を CA クリロフソルバと比較した。省通信ステップ数 $s > 10$ で動作する P-CBCG 法によって CA クリロフソルバのロバースト性が向上したが、14 億自由度の問題における P-CG ソルバと比べた性能向上は約 1.34 倍にとどまる。これは CA クリロフ部分空間法は元のクリロフ部分空間法と数学的に等価であり、収束特性の向上が期待できないためである。一方、CAMGCG ソルバは通信処理の削減と収束特性の向上を両立する。ここで、後者が前者に大きく影響する。今回の多相流体問題では二桁以上という劇的な収束特性の向上を示し、大規模問題になるほどより大きな性能向上が得られた。このような収束特性向上、混合精度処理、および、省通信設計の効果により、CAMGCG ソルバは 14 億自由度および 900 億自由度の問題において、P-CG 法に比べてそれぞれ約 3.7 倍および約 11.6 倍という大幅な性能向上を達成し、Oakforest-PACS 全系規模の 8,000KNL まで良好な強スケーリングを示した。

JUPITER コードにおいて、CAMGCG ソルバは問題サイズによらずほぼ同様の収束特性を示し、900 億自由度の大規模多相流体問題をリーズナブルな計算コストで処理することができた。この結果から、提案手法は将来のエクサスケール CFD シミュレーションにおいて有望な手法であることが示された。

謝辞

本研究は文部科学省（ポスト京重点課題 6：革新的クリーンエネルギーの実用化）および最先端共同 HPC 基盤施設（JCAHPC）「大規模 HPC チャレンジ」の支援を受けた。本研究の計算の一部は Oakforest-PACS（JCAHPC）および ICEX（JAEA）において実施された。

参考文献

[1] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[2] M. Hoemmen. *Communication-avoiding Krylov subspace methods*. PhD thesis, University of California, Berkeley, 2010.

[3] E. C. Carson. *Communication-Avoiding Krylov Subspace Methods in Theory and Practice*. PhD thesis, University of California, Berkeley, 2015.

[4] Reiji Suda, Li Cong, Daichi Watanabe, et al. Communication-avoiding CG method: New direction of krylov subspace methods towards exa-scale computing. *RIMS Kôkyûroku*, Vol. 1995, pp. 102–111, 2016.

[5] P. Ghysels, T. Ashby, K. Meerbergen, and W. Vanroose. Hiding global communication latency in the GMRES algorithm on massively parallel machines. *SIAM Journal on Scientific Computing*, Vol. 35, No. 1, pp. C48–C71, 2013.

[6] P. Ghysels and W. Vanroose. Hiding global synchronization latency in the preconditioned conjugate gradient algorithm. *Parallel Computing*, Vol. 40, No. 7, pp. 224–238, 2014.

[7] A. Mayumi, Y. Idomura, T.Ina, et al. Left-preconditioned communication-avoiding conjugate gradient methods for multiphase CFD simulations on the K computer. In *Proceedings of the 7th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ScalA '16, pp. 17–24, Piscataway, NJ, USA, 2016. IEEE Press.

[8] Y. Idomura, T. Ina, A. Mayumi, et al. Application of a communication-avoiding generalized minimal residual method to a gyrokinetic five dimensional eulerian code on many core platforms. In *Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ScalA '17, pp. 7:1–7:8, New York, NY, USA, 2017. ACM.

[9] Y. Idomura, T .Ina, A. Mayumi, et al. Application of a preconditioned chebyshev basis communication-avoiding conjugate gradient method to a multiphase thermal-hydraulic CFD code. *Lecture Notes in Computer Science*, Vol. 10776, pp. 257–273, 2018.

[10] Kazuya Matsumoto, Yasuhiro Idomura, Takuya Ina, Akie Mayumi, and Susumu Yamada. Implementation and performance evaluation of a communication-avoiding gmres method for stencil-based code on gpu cluster. *The Journal of Supercomputing*, Vol. 75, No. 12, pp. 8115–8146, 2019.

[11] Y. Ali, N. Onodera, Y. Idomura, T. Ina, and T. Imamura. Gpu acceleration of communication avoiding chebyshev basis conjugate gradient solver for multiphase cfd simulations. In *2019 IEEE/ACM 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, pp. 1–8, Nov 2019.

[12] Y. Idomura, T. Ina, Y. Ali, and T. Imamura. Acceleration of fusion plasma turbulence simulations using the mixed-precision communication-avoiding krylov method. *accepted for publication in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'20)*, 2020.

[13] Tsuyoshi Ichimura, Kohei Fujita, Pher Errol Balde Quinay, et al. Implicit nonlinear wave simulation with 1.08T DOF and 0.270T unstructured finite elements to enhance comprehensive earthquake simulation. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pp. 4:1–4:12, New York, NY, USA, 2015. ACM.

[14] Chao Yang, Wei Xue, Haohuan Fu, et al. 10M-core scalable fully-implicit solver for nonhydrostatic atmospheric dynamics. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, pp. 6:1–6:12, Piscataway, NJ, USA, 2016. IEEE Press.

[15] Y. Idomura, T. Ina, S. Yamashita, N. Onodera, S. Yamada, and T. Imamura. Communication avoiding multigrid preconditioned conjugate gradient method for extreme scale multiphase cfd simulations. In *2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (scalA)*, pp. 17–24, 2018.

[16] Susumu Yamashita, Takuya Ina, Yasuhiro Idomura, and Hiroyuki Yoshida. A numerical simulation method for molten material behavior in nuclear reactors. *Nuclear Engineering and Design*, Vol. 322, No. Supplement C, pp. 301 – 312, 2017.

[17] Yosuke Kumagai, Akihiro Fujii, Teruo Tanaka, et al. Performance analysis of the chebyshev basis conjugate gradient method on the K computer. *Lecture Notes in Computer Science*, Vol. 9537, pp. 74–85, 2016.

[18] Martin H. Gutknecht and Stefan Röllin. The chebyshev iteration revisited. *Parallel Computing*, Vol. 28, No. 2, pp. 263 – 283, 2002.

[19] T. Shimokawabe, T. Aoki, C. Muroi, et al. An 80-fold speedup, 15.0 TFlops full GPU acceleration of non-hydrostatic weather model ASUCA production code. In *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–11, Nov 2010.