

Communication avoiding multigrid preconditioned conjugate gradient method for extreme scale multiphase CFD simulations

Y. Idomura^{*§}, T. Ina^{*}, S. Yamashita[†], N. Onodera^{*}, S. Yamada^{*} and T. Imamura[‡]

^{*}Japan Atomic Energy Agency, Kashiwa, Chiba 277-0871, Japan

[†]Japan Atomic Energy Agency, Tokai, Ibaraki 319-1195, Japan

[‡]R-CCS, Riken, Kobe, Hyogo 650-0047, Japan

[§] Email: idomura.yasuhiro@jaea.go.jp

Abstract—A communication avoiding (CA) multigrid preconditioned conjugate gradient method (CAMGCG) is applied to the pressure Poisson equation in a multiphase CFD code JUPITER, and its computational performance and convergence property are compared against CA Krylov methods. A new geometric multigrid preconditioner is developed using a preconditioned Chebyshev iteration smoother, in which no global reduction communication is needed, halo data communication is reduced by a mixed precision approach, and eigenvalues are computed using the CA Lanczos method. In the JUPITER code, the CAMGCG solver has robust convergence properties regardless of the problem size, and shows both communication reduction and convergence improvement, leading to higher performance gain than CA Krylov solvers, which achieve only the former. The CAMGCG solver is applied to extreme scale multiphase CFD simulations with ~ 90 billion DOFs, and it is shown that compared with a preconditioned CG solver, the number of iterations, and thus, All_Reduce is reduced to $\sim 1/800$, and $\sim 11.6\times$ speedup is achieved with keeping excellent strong scaling up to 8,000 KNLs on the Oakforest-PACS.

Index Terms—Multigrid method, Krylov method, Communication Avoiding, Chebyshev iteration, CFD simulation

I. Introduction

Iterative methods are widely used for solving linear systems given by extreme scale sparse matrices, and thus, their scalability is one of critical issues in exascale computing. In nuclear engineering, exascale computing is needed for Computational Fluid Dynamics (CFD) simulations of multiphase turbulent flows in nuclear reactors. In such extreme scale CFD simulations, the dominant cost comes from an implicit solver based on iterative methods, which dictate the scalability and performance of the total code. The Krylov method [1] is one of the most powerful iterative methods in many fields. However, extreme scale Krylov solvers have the following two critical issues. One is global synchronization and the other is convergence degradation.

In the Krylov method, a basis vector is generated and orthogonalized at each iteration. The latter procedure requires inner product operations, leading to global reduction communication or synchronization. On the latest many core platforms, the performance gap between computation and communication is expanding, and the latency

of global synchronization is becoming a severe performance bottleneck. In order to relax this communication bottleneck, several communication reduction approaches such as communication avoiding (CA) Krylov methods [2]–[4] and pipelined Krylov methods [5], [6] have been proposed. In the former, global synchronization is reduced by computing multiple basis vectors at once, while the latter hides the cost of global synchronization via asynchronous global communication. In recent works [7]–[9], it was shown that CA Krylov methods improve strong scaling of extreme scale nuclear CFD simulations.

Recent extreme scale CFD simulations have a multi-scale feature, and the conditions of matrices are becoming worse. In such ill-conditioned problems, the number of iterations in Krylov solvers tends to increase with the problem size, and therefore, the improvement of preconditioning is of critical importance. The multigrid (MG) method is one of the most promising approaches for this kind of multi-scale problem, and was used in recent extreme scale simulations reaching ~ 1 trillion DOFs [10], [11]. By improving the convergence property, the number of iterations, and thus, global synchronization can be significantly reduced. However, the remaining issue is to reduce communication in MG preconditioners with keeping the convergence property.

In this work, we apply a CA MG preconditioned conjugate gradient (CAMGCG) method to the pressure Poisson equation in a multiphase thermal-hydraulic CFD code JUPITER [12], and compare its computational performance and convergence property against the conventional preconditioned CG (P-CG) method and the advanced CA Krylov methods on the latest many core platform. The followings are main contributions in this work.

- A new CA MG preconditioner based on the geometric multigrid (GMG) method is developed using a preconditioned Chebyshev iteration (P-CI) smoother, in which no global reduction communication is needed, halo data communication is reduced by a mixed precision approach, and eigenvalues are computed using the CA Lanczos method.
- Systematic comparisons of the computational perfor-

mance and the convergence property are made for the P-CG method, the preconditioned CA CG (P-CACG) method, the preconditioned Chebyshev basis CA CG (P-CBCG) method, and the CAMGCG method.

- The CAMGCG solver is applied to extreme scale multiphase CFD simulations with ~ 90 billion DOFs, and it is shown that compared with the P-CG solver, the number of iterations is reduced to $\sim 1/800$ and $\sim 11.6\times$ speedup is achieved with keeping excellent strong scaling up to 8,000 KNLs.

II. Related works

CA Krylov methods are based on the so-called s -step Krylov method, in which the data dependency between SpMV and inner product operations in the standard Krylov method is removed. Van Rosendale [13] first developed a s -step version of the CG method. Chronopoulos and Gear [14] called their own variant of the CG method as the s -step CG method. Toledo optimized the computation of the s -step basis in the s -step CG method [15], in which data transfer between levels of the memory hierarchy is reduced. The CACG method by Hoemmen [2] reduced communications between levels of the memory hierarchy and between processors by a matrix power kernel (MPK) [16].

Preconditioning for CA Krylov methods is essential in most of real applications. However, when parallel preconditioners have data dependency over the whole local subdomain as in block Jacobi (BJ) preconditioning, it is difficult to construct a MPK without additional communications. To avoid the additional communications, Yamazaki et al. [17] proposed an underlap approach, in which each subdomain is divided into an inner part and the remaining surface part, and preconditioning for the latter is approximated by point Jacobi preconditioning. However, in Ref. [7], we showed that for ill-conditioned problems in the JUPITER code, the underlap approach leads to significant convergence degradation, and a hybrid CA approach, in which SpMVs and BJ preconditioning are unchanged and CA is applied only to inner product operations, was proposed.

In most of performance studies [3], [8], [17], CA Krylov methods were applied to well-conditioned problems, where CA-steps are extended for $s > 10$. However, in Ref. [7], it was also shown that for ill-conditioned problems in the JUPITER code, the P-CACG method is numerically stable only within a few CA-steps even with the original BJ preconditioning. This issue is attributed to the monomial basis vectors, which are aligned to the eigenvector with the maximum eigenvalue as s increases, and their linear independency is ruined by the round off errors. To resolve this issue, Suda et al. [4] proposed the P-CBCG method, which was tested with point Jacobi preconditioning on the K-computer [18]. In Ref. [9], the P-CBCG method with BJ preconditioning was applied to the JUPITER code, and the above numerical stability issue was resolved for

$s > 10$. However, the convergence property of the P-CBCG method is still limited by that of the P-CG method, as they are mathematically equivalent in exact arithmetic.

As a promising solution for this convergence issue, the MG method is widely used in recent extreme scale simulations [10], [11]. When the number of iterations is reduced by MG preconditioning, the convergence improvement itself leads to communication reduction in Krylov methods, and the remaining issue is communication reduction in MG preconditioners. One critical component of the MG method is the smoother, which can be implemented using stationary iterative methods such as the Gauss-Seidel method, and non-stationary iterative methods such as the CI method and Krylov methods [1]. Here, stationary iterative methods and the CI method do not require inner product operations, and are suitable for CA MG preconditioning. In Ref. [19], several parallel smoothers were compared, and it was shown that the CI method has a robust convergence property. In this work, we develop a GMG preconditioner using the P-CI method with BJ preconditioning. In Ref. [20], a CI smoother was optimized to improve the arithmetic intensity using a MPK like approach. However, a MPK cannot be used with BJ preconditioning. In this work, we propose a mixed precision approach to improve the arithmetic intensity and reduce halo data communication. We also propose to use the CA Lanczos method [2] for the eigenvalue computation in the P-CI method.

III. Krylov solvers in JUPITER code

A. Code overview

In the JUPITER code [12], thermal-hydraulics of the molten material in nuclear reactors is described by the equations of continuity, Navier-Stokes, and energy, assuming Newtonian and incompressible viscous fluids. The dynamics of gas, liquid, and solid phases of multiple components consisting of fuel pellets, fuel cladding, the channel box, the absorber, reactor internal components, and the atmosphere are described by an advection equation of the volume of fluid (VOF) function. The main computational cost ($\sim 90\%$) comes from computation of the pressure Poisson equation, because the Poisson operator given by the density has extreme contrast $\sim 10^7$ between gas and solid phases, and is ill-conditioned. The Poisson equation is discretized by the second order accurate centered finite difference scheme (7 stencils) in the Cartesian grid system (x, y, z) . The linear system of the pressure Poisson equation, which is a symmetric block diagonal sparse matrix, is solved using Krylov methods. These Krylov solvers use the compressed diagonal storage (CDS) format, which enables highly efficient direct memory access for the block diagonal sparse matrix than the compressed sparse row (CSR) format, which requires indirect memory access, and are parallelized using a MPI+OpenMP hybrid parallelization model, in which MPI is used for coarse 3D domain decomposition in (x, y, z) and fine 1D domain

Algorithm 1 Preconditioned Conjugate Gradient (P-CG) method

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 $j = 1, 2, \dots$ 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

decomposition in z is applied to each local domain via OpenMP. BJ preconditioning is applied to each fine subdomain so that it is computed in thread parallel.

B. P-CG method

In the original JUPITER code, the pressure Poisson equation was computed using the P-CG method with BJ preconditioning [1]. In the P-CG method in Algorithm 1, a single iteration consists of SpMV, BJ preconditioning, two inner product operations, and three vector operations (AXPYs). Here, the SpMV requires halo data communication per iteration, and the inner product operations need two global reduction communications (All_reduce) per iteration. One All_reduce at line 4 transfers two elements including the norm of residual vector, while the other All_reduce at line 8 sends one element.

C. P-CACG method

Algorithm 2 shows the P-CACG method [2], [7], in which dominant computation comes from the s -step SpMVs (line 5) and the following BJ preconditioning (line 6) in the outer loop, GEMM operations for constructing the Gram matrix (line 7) in the outer loop, and three vector operations for the three term recurrence formulae (lines 22-24) in the inner loop. Here, the size of GEMM operations depends on s , and thus, their arithmetic intensity is increased with s . If one applies cache blocking optimization, coefficients of the three term recurrence formulae can be reused for s -steps, and the arithmetic intensity of three vector operations is also improved by extending s . The SpMV requires halo data communication per inner iteration as in the P-CG method, while the Gram matrix computation needs only one All_reduce for $s(s+1)$ elements of $G_{k,k-1}$ and $(s+2)(s+1)/2$ upper-triangular elements of $G_{k,k}$ per outer iteration. In addition, we compute the norm of residual vector $r_{sk} = b - Ax_{sk+1}$ for the convergence check, which require one All_reduce per outer iteration. Therefore, the P-CACG method requires two All_reduces per outer iteration.

Algorithm 2 Preconditioned Communication Avoiding CG (P-CACG) method

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 $k = 0, 1, 2, \dots$ until convergence do
- 4: $v_{sk+1} := z_{sk+1}$
- 5: Compute $\underline{V}_k(v_{sk+1}, M^{-1}Av_{sk+1}, \dots, (M^{-1}A)^s v_{sk+1})$
- 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

D. P-CBCG method

Algorithm 3 shows the P-CBCG method [4], [9], in which dominant computational costs come from the preconditioned Chebyshev basis vector generation involving the SpMVs and the BJ preconditioning (line 10), and the remaining matrix computations. The SpMVs at line 10 require s local halo data communications, while the matrix computations at lines 5, 6 and 11 need global data reduction communications. Therefore, the P-CBCG method requires two All_reduces per s -steps. One All_reduce at lines 5, 6 transfers $s(s+1)/2$ upper-triangular elements of $Q_k^* A Q_k$, s elements of $Q_k^* r_{sk}$, and one element for the norm of residual vector, while the other All_reduce at line 11 sends s^2 elements of $Q_k^* A S_{k+1}$.

E. CAMGCG method

In the P-CG method in Algorithm 1, we replace the BJ preconditioner with a multigrid preconditioner. Since

Algorithm 3 Preconditioned Chebyshev Basis communication avoiding CG (P-CBCG) method

Input: $Ax = b$, Initial guess x_0

Output: Approximate solution x_i

- 1: $r_0 := b - Ax_0$
 - 2: Compute S_0 ($T_0(AM^{-1})r_0, \dots, T_{s-1}(AM^{-1})r_0$)
 - 3: $Q_0 = S_0$
 - 4: for $k = 0, 1, 2, \dots$ until convergence do
 - 5: Compute $Q_k^* A Q_k$
 - 6: Compute $Q_k^* r_{sk}$
 - 7: $a_k := (Q_k^* A Q_k)^{-1} Q_k^* r_{sk}$
 - 8: $x_{s(k+1)} := x_{sk} + Q_k a_k$
 - 9: $r_{s(k+1)} := r_{sk} - A Q_k a_k$
 - 10: S_{k+1} ($T_0(AM^{-1})r_{s(k+1)}, \dots, T_{s-1}(AM^{-1})r_{s(k+1)}$)
 - 11: Compute $Q_k^* A S_{k+1}$
 - 12: $B_k := (Q_k^* A Q_k)^{-1} Q_k^* A S_{k+1}$
 - 13: $Q_{k+1} := S_{k+1} - Q_k B_k$
 - 14: $A Q_{k+1} := A S_{k+1} + A Q_k B_k$
 - 15: end for
-

Algorithm 4 Preconditioned Chebyshev iteration (P-CI) method

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

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

the JUPITER code is based on the Cartesian grid system, we use the GMG method with a V-cycle [1] (Fig.1). The smoother is implemented using the P-CI method [21] with the BJ preconditioning, which requires no inner product operation. The P-CI method accelerates the convergence of iterative methods by improving solution vectors using Chebyshev polynomials, which require eigenvalue computation for the preconditioned matrix $A\tilde{M}^{-1}$. In the P-CI method in Algorithm 4, one needs to estimate the minimum and maximum eigenvalues of $A\tilde{M}^{-1}$. We use the preconditioned CA Lanczos method based on Newton basis [2] for the eigenvalue computation. We also use a fixed iteration number, which is chosen based on the initial convergence scan, to avoid the convergence check that requires global reduction communication.

In the CAMGCG method, we use a mixed precision approach to improve the arithmetic intensity and reduce halo data communication. Here, the choice of precision is optimized based on the arithmetic property of each

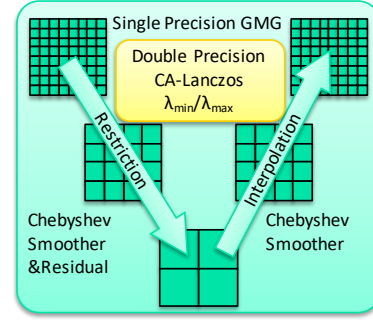


Fig. 1. The mixed precision geometric multigrid preconditioner based on a V-cycle with the preconditioned Chebyshev iteration smoother. The communication avoiding Lanczos method is used for the eigenvalue computation.

algorithm. In the main CG method, the computation of the residual vector requires double precision near the convergence limit. The CA lanczos method generates and orthogonalizes multiple basis vectors at once as in CA Krylov methods, and the linear independency of those vectors is sensitive to the round off errors. On the other hand, the P-CI smoother requires an approximate solution obtained typically with several tens iterations, which allows us to use single precision. Therefore, the GMG preconditioner is computed in single precision, while the preconditioned CA-Lanczos method and the main CG method are computed in double precision.

In the CAMGCG method, dominant computational costs come from the P-CI method, which consists of SpMV, BJ preconditioning, and three AXPYs. They are quite similar to the kernels in the P-CG method except for the inner product operations, which require All_Reduce. Therefore, the P-CI smoother can be easily implemented with minor corrections to the existing P-CG solver.

IV. Kernel performance analysis

In this section, we analyze the kernel performance of the P-CG, P-CACG, P-CBCG, and CAMGCG solvers using a small problem size of $N = 104 \times 104 \times 265$, which corresponds to a typical problem size per processor. The sustained performance of each kernel on KNL is compared against the modified roofline model [22], in which a theoretical processing time of each kernel is estimated by the sum of costs for floating point operations and memory access, $t_R = f/F + b/B$. Here, f and b are the numbers of floating point operations and memory access of the kernel. F and B are the peak performance and the STREAM memory bandwidth of the processor.

A. P-CG solver

Computational kernels of the P-CG method consist mainly of SpMV (lines 3, 4), BJ (lines 7, 8), and Vector (AXPYs, lines 5,6,9). Here, SpMV and BJ involve inner product operations in the same loop. The numbers of floating point operations f and memory access b and the resulting arithmetic intensity f/b of each kernel

TABLE I

Kernel performance analysis on a single KNL (Floating point operation f [Flop/grid], Memory access b [Byte/grid], Arithmetic intensity f/b , Peak performance F [Flops], STREAM bandwidth B [Byte/s], Roofline time $t_R = f/F + b/B$ [ns/grid], Elapse time t [ns/grid], and Sustained performance P [GFlops]). F in single precision (SP) is doubled from that of double precision (DP).

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

are summarized in Table I. Since the pressure Poisson equation is solved using the second order accurate centered finite difference scheme, SpMV and BJ have relatively low arithmetic intensity $f/b < 0.2$. In addition, the remaining AXPYs are memory intensive kernels with $f/b = 0.1$. Therefore, the high memory bandwidth on KNL has a great impact on the acceleration of the P-CG solver. Although AXPYs in Vector achieve ideal sustained performances with the performance ratio against the roofline model $t_R/t \sim 0.84$, stencil computations in SpMV and BJ show performance degradation $t_R/t \sim 0.6$.

B. P-CACG solver

Computational kernels of the P-CACG method consist mainly of SpMV (lines 5,6), BJ (lines 5,6), Gram (lines 7,8), and 3-term (lines 20-24). The arithmetic intensity of the P-CACG method changes depending on s , because the arithmetic intensity of Gram and 3-term are proportional to s . Gram scales as $f = 2(s+1)(2s+1)/s$ and $b = 8(3s+2)/s$. 3-term scales as $f = (8s^2 + 12s + 2)/s$ and $b = 48(s+2)/s$, where the s -dependency comes from cache blocking optimization [7]. In Table I, the kernel performance is summarized at $s = 3$, which is the upper limit of CA-steps in the benchmark problem in Sec. V. SpMV and BJ in the P-CACG method have lower f and b than the P-CG method, because they do not involve inner product operations. Although Gram and 3-term gives significant increase of computation f , they have high arithmetic intensity $f/b > 0.5$, and thus, an impact of additional computation on the total computational cost ($\sim 1.53\times$) is much lower than the increase of f ($\sim 2.24\times$) from the P-CG method.

C. P-CBCG solver

Computational kernels of the P-CBCG method consist mainly of the Chebyshev basis computation CB (line 10), and the remaining matrix computations Matrix. The arithmetic intensity of the P-CBCG method depends on s as in the P-CACG method. CB scales as $f = 2(9s+4)/s$ and $b = 8(4s+35)/s$, and Matrix scales as $f = (7s+2)(s+1)/s$ and $b = 40(2s+1)/s$. In Table I, the kernel performance is summarized for $s = 12$, which is used in the benchmark problem in Sec. V. Although f of the P-CBCG method becomes $3.17\times$ larger than the P-CG method, the increase of computational cost is suppressed to $1.79\times$, because of the improved arithmetic intensity.

D. CAMGCG solver

The main computational kernel of the CAMGCG method is the P-CI smoother, which consists of SpMV (line 5), BJ (line 6), and Vector (AXPYs, lines 4,9). The arithmetic intensity of the CAMGCG method is similar to that of the P-CG method, and their computational costs are comparable. However, if one applies single precision to the P-CI smoother, the memory access becomes a half, and the width of SIMD operation, thus the peak performance is doubled. According to the roofline model, $\sim 2\times$ speedup is expected in single precision. In Table I, Vector shows $2\times$ speedup, while SpMV and BJ show less performance improvement. As a result, the total performance gain from double precision is limited to $1.33\times$.

V. Numerical experiment

Numerical experiments are performed on the Oakforest-PACS, which consists of 8,208 Xeon Phi 7250 processors (KNL, 68cores, $F=3046$ GFlops, $B=480$ GB/s) and OmniPath (fat tree topology, 12.5GB/s). We use the Intel compiler 17.0.4 with the Intel MPI library 2017 (-O3 -qopenmp -xMIC-AVX512). We assign a single MPI process per node, and use 64 threads. The hierarchical memory architecture consisting of MCDRAM (16GByte) and DDR4 (96GByte) is used in a cache mode.

A. Convergence property

In the present numerical experiment, we compute nonlinear evolutions of molten debris in a single fuel assembly component of nuclear reactor. The problem size is chosen as $N = 800 \times 500 \times 3,540 \sim 1.4 \times 10^9$, which was used also in the former works [7], [9], [12]. The problem treats multiphase flows consisting of gas and multi-component liquid and solid of fuel pellets, fuel cladding, the channel box, the absorber, and the other reactor internal components. The convergence property and the computational performance are investigated for fully developed multiphase flows, which give the largest iteration number. Because of the large problem size and the extreme density contrast of multiphase flows, the pressure Poisson equation is ill-conditioned.

TABLE II

Convergence properties of the P-CG, P-CACG($s=3$), P-CBCG($s=12$), CAMGCG solvers with the SSOR and P-CI smoothers. The number of iterations and the computational cost per time step on 500 KNs are shown for the problem size $N = 800 \times 500 \times 3,540 \sim 1.4 \times 10^9$. Here, s is the CA steps and ω is the acceleration parameter of the SSOR smoother. The CAMGCG solver uses a V-cycle with 5 levels, and the SSOR and P-CI smoothers are computed for 30 internal iterations at each level.

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

The convergence properties of the P-CG, P-CACG, P-CBCG, and CAMGCG solvers are summarized in Table II. Here, the convergence condition is given by the relative residual error of $|b - Ax|/|b| < 10^{-8}$. The CA steps for the P-CACG and P-CBCG solvers are respectively chosen as $s = 3$ and $s = 12$ following Refs. [7], [9]. In the CAMGCG solver, we compare GMG preconditioners using two different smoother algorithms. One is the P-CI method and the other is the Symmetric Successive Over Relaxation (SSOR) method with red-black ordering [1]. The number of levels is chosen so that the coarsest grids become below million grids. The coarser grids are processed using the same domain decomposition model with halo data communication as in the original finest grids, while computation and halo data communication are respectively reduced by $\sim 1/2^3$ and $\sim 1/2^2$ with increment of the level. In the current problem size, the CAMGCG solver uses a V-cycle with 5 levels, and the number of internal iterations used in the P-CI and SSOR smoothers at each level is fixed to 30.

In Table II, the P-CACG and P-CBCG solvers show the similar number of iterations as the P-CG solver, because they are mathematically equivalent in exact arithmetic, and the similar convergence properties are expected when the CA steps are chosen so that convergence degradation due to the round off errors does not occur. On the other hand, the CAMGCG solver dramatically improves the convergence property. The CAMGCG solvers with the SSOR and P-CI smoothers reduce the number of iterations to $\sim 1/50$ and $\sim 1/200$, respectively. These convergence improvements lead to the reduction of All_Reduce. However, the CAMGCG solver with the SSOR smoother requires a large computational cost due to inefficient strided memory access and two halo data communications for red and black phases, and the total cost is larger than the P-CG solver. The CAMGCG solver with the P-CI smoother is converged with $\sim 1/4$ iterations compared with the SSOR smoother, and reduces both computation and communication. In addition, by applying a mixed

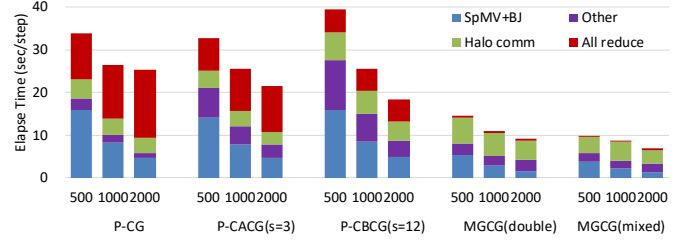


Fig. 2. Strong scaling of the P-CG, P-CACG($s=3$), P-CBCG($s=12$), and CAMGCG (double/mixed precision) solvers using 500, 1,000, and 2,000 KNs. The cost distribution in a single time step is shown for the problem size $N = 800 \times 500 \times 3,540 \sim 1.4 \times 10^9$.

TABLE III
The cost distribution (sec/step) shown in Fig.2.

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

precision approach, it is further accelerated. As expected, we do not see any convergence degradation by using the P-CI smoother in single precision. It is noted that in the JUPITER code, the Poisson operator changes in time depending on the density distribution, and thus, the P-CI smoother requires the eigenvalue calculation via the CA Lanczos method at each time step. In the CA Lanczos solver, 100 iterations with the CA steps of $s = 5$ are computed to guarantee enough accuracy, while the maximum and minimum eigenvalues are converged typically with ~ 10 iterations. The cost of the CA Lanczos solver is ~ 0.7 sec, which is less than 10% of the total cost. It is noted that the cost of the eigenvalue calculation is doubled when the (non-CA) Lanczos method is used. In the following scaling study, we use the mixed-precision CAMGCG solver with the P-CI smoother.

B. Strong scaling test with 1.4 billion DOFs

The strong scaling of the P-CG, P-CACG, P-CBCG, and CAMGCG solvers are summarized in Fig. 2 and Table III. In the strong scaling test, we use 500, 1,000, and 2,000 KNs. The P-CG solver shows significant degradation of strong scaling due to All_Reduce. The P-CACG solver with $s = 3$ reduces the cost of All_Reduce, and shows better strong scaling. However, the cost reduction

of All_Reduce is much less than theoretical estimation $1/s$, and the performance improvement from the P-CG solver is limited to $\sim 1.18\times$ at 2,000 KNLs. The P-CBCG solver with $s = 12$ shows much better strong scaling than the P-CACG solver. However, the cost reduction of All_Reduce is again much less than $1/s$, and the costs of computing kernels are larger than those of the P-CACG method. As a result, the performance improvement from the P-CG solver is $\sim 1.38\times$ at 2,000 KNLs. The computational cost of the CAMGCG solver comes mainly from the P-CI smoother at the finest level, which is called $30 \times 2 \times 33 = 1980$ times, and thus, the cost of SpMV becomes $\sim 1/3$ of the P-CG solver, which is converged with $\sim 7,000$ iterations. As the number of All_Reduce is reduced from $\sim 7,000$ to 33, the dominant communication cost is given by halo data communication for the P-CI smoother. In the current strong scaling test, the decomposed domain size is reduced to $1/2$ and $1/4$ in the z -direction, and the halo data size per MPI process scales as $2/3$ and $1/2$ from 500 KNLs, respectively. However, in all solvers, the scalability of halo data communication is worse than this estimate. Accordingly, the strong scaling of the CAMGCG solver is worse than the other solvers, while the total performance is dramatically improved. By applying the mixed precision approach, both computation and communication are accelerated, and the performance gain between mixed and double precision versions becomes $\sim 1.34\times$. The total performance ratio between the P-CG solver and the mixed precision CAMGCG solver becomes $\sim 3.7\times$ at 2,000 KNLs.

C. Strong scaling test with 90 billion DOFs

In this section, we show numerical experiments for an extreme scale problem $N = 3,200 \times 2,000 \times 14,160 \sim 9.06 \times 10^{10}$, in which the grid resolution is increased $4\times$ in all directions in the same multiphase flow problem used in the previous section. In this case, an adaptive time step width is automatically adjusted to $\sim 1/4$, and the interface width of multiphase flows becomes $\sim 1/4$. As the density shows more sharp contrast, the condition of the Poisson operator becomes even worse. Accordingly, the P-CG solver shows significant convergence degradation reaching 26,475 iterations. On the other hand, the CAMGCG solver is converged with 32 iterations in both mixed and double precision, and the number of iterations, and thus, All_Reduce is reduced to $\sim 1/800$ from the P-CG solver. Here, the CAMGCG solver uses a V-cycle with 7 levels (two more levels with $4\times$ grids in each direction), and 50 internal iterations are computed at each level.

In the strong scaling test, the P-CG and mixed precision CAMGCG solvers are tested up to the full system size of the Oakforest-PACS. In Fig. 3 and Table IV, the cost distributions of two solvers at 2,000, 4,000, and 8,000 KNLs are summarized. At 2,000 KNLs, compared with the 1.4 billion DOFs case, the cost distribution of the P-CG solver is dominated by computation because the

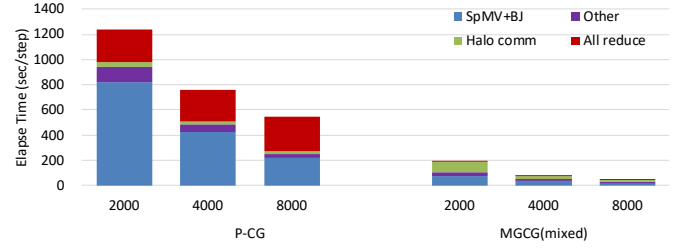


Fig. 3. Strong scaling of the P-CG and CAMGCG (mixed precision) solvers using 2,000, 4,000, and 8,000 KNLs. The cost distribution in a single time step is shown for the problem size $N = 3,200 \times 2,000 \times 14,160 \sim 9.06 \times 10^{10}$

TABLE IV
The cost distribution (sec/step) shown in Fig.3.

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

problem size per process is increased by $64\times$, leading to better strong scaling. However, at 8,000 KNLs, a half of the cost distribution is occupied by All_Reduce. In the CAMGCG solver, SpMV in the P-CI smoother is called $50 \times 2 \times 32 = 3200$ times, which is an order of magnitude smaller than the P-CG solver, and the cost of SpMV is reduced to $\sim 1/10$. However, at 2,000 KNLs, the cost of halo data communication, which involves communication at all levels, is doubled from the P-CG solver. This cost increase happens only at 2,000 KNLs, and at 4,000 and 8,000 KNLs, the cost of halo data communication is lower than the P-CG solver. This kind of communication degradation often occurs when the memory usage exceeds the size of MCDRAM, and thus, may be related to some additional data transfer between MCDRAM and DDR4. The CAMGCG solver also shows better strong scaling than the 1.4 billion DOFs case due to increased computation, and the performance gains of $\sim 2.6\times$ and $\sim 4.2\times$ are achieved at 4,000 and 8,000 KNLs, respectively. Compared with the 1.4 billion DOFs case, the performance ratio between the P-CG solver and the CAMGCG solver is expanded to $\sim 11.6\times$ at 8,000 KNLs.

VI. Summary

In this work, we developed a new CAMGCG solver for the pressure Poisson equation in the JUPITER code. The CAMGCG solver is designed to avoid global reduction communication using the P-CI smoother and the CA Lanczos method. In addition, by optimizing mixed precision, both computation and communication are further reduced without convergence degradation.

The CAMGCG solver was compared against the CA Krylov solvers. Although the robustness of the CA Krylov

method is improved by the P-CBCG method, which operates at larger CA steps $s > 10$, the total performance improvement from the P-CG solver was limited to $\sim 1.34\times$ for the 1.4 billion DOFs case, because CA Krylov methods are mathematically equivalent to non-CA Krylov methods, and thus, they do not improve the convergence property. On the other hand, the CAMGCG solver achieved both communication reduction and convergence improvement, where the latter also has a great impact on the former. In the current multiphase flow problem, the convergence property was dramatically improved by more than two orders of magnitudes, and higher performance gain was achieved for larger problem sizes. Thanks to the convergence improvement, the CA MG preconditioner, and the mixed precision approach, the CAMGCG solver achieved $3.7\times$ and $11.6\times$ speedups for the 1.4 and 90 billion DOFs cases, respectively, and showed excellent strong scaling up to 8,000 KNLs, which are almost the full system size of the Oakforest-PACS.

In the JUPITER code, the CAMGCG solver showed the similar convergence property regardless of the problem size, and an extreme scale multiphase flow problem with 90 billion DOFs can be computed with a reasonable computational cost. These results are promising for future exascale CFD simulations.

Acknowledgement

This work is supported by the MEXT (Grant for Post-K priority issue No.6: Development of Innovative Clean Energy) and "Large-scale HPC Challenge" Project, the Joint Center for Advanced High Performance Computing (JCAHPC). Computations were performed on the Oakforest-PACS (JCAHPC), the K-computer (Riken, hp180100), and the ITEX (JAEA).

References

- [1] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003.
- [2] M. Hoemmen, "Communication-avoiding krylov subspace methods," Ph.D. dissertation, University of California, Berkeley, 2010.
- [3] E. C. Carson, "Communication-avoiding krylov subspace methods in theory and practice," Ph.D. dissertation, University of California, Berkeley, 2015.
- [4] R. Suda, L. Cong, D. 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*, ser. *Scala '16*. Piscataway, NJ, USA: IEEE Press, 2016, pp. 17–24.
- [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*, ser. *Scala '17*. New York, NY, USA: ACM, 2017, pp. 7:1–7:8.
- [9] Y. Idomura, T. Ina, A. Mayumi et al., "Application of a pre-conditioned 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] T. Ichimura, K. Fujita, P. E. B. 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*, ser. *SC '15*. New York, NY, USA: ACM, 2015, pp. 4:1–4:12.
- [11] C. Yang, W. Xue, H. 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*, ser. *SC '16*. Piscataway, NJ, USA: IEEE Press, 2016, pp. 6:1–6:12.
- [12] S. Yamashita, T. Ina, Y. Idomura, and H. 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.
- [13] J. Van Rosendale, "Minimizing inner product data dependencies in conjugate gradient iteration," NASA contractor report, 1983.
- [14] A. Chronopoulos and C. Gear, "s-step iterative methods for symmetric linear systems," *Journal of Computational and Applied Mathematics*, vol. 25, no. 2, pp. 153 – 168, 1989.
- [15] S. A. Toledo, "Quantitative performance modeling of scientific computations and creating locality in numerical algorithms," Ph.D. dissertation, Massachusetts Institute of Technology, 1995.
- [16] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick, "Avoiding communication in sparse matrix computations," in *2008 IEEE International Symposium on Parallel and Distributed Processing*, April 2008, pp. 1–12.
- [17] I. Yamazaki, H. Anzt, S. Tomov et al., "Improving the performance of CA-GMRES on multicore with multiple GPUs," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, May 2014, pp. 382–391.
- [18] Y. Kumagai, A. Fujii, T. 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.
- [19] A. Baker, R. Falgout, T. Kolev, and U. Yang, "Multigrid smoothers for ultraparallel computing," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2864–2887, 2011.
- [20] P. Ghysels, P. Kłosiewicz, and W. Vanroose, "Improving the arithmetic intensity of multigrid with the help of polynomial smoothers," *Numerical Linear Algebra with Applications*, vol. 19, no. 2, pp. 253–267, 2012.
- [21] M. H. Gutknecht and S. Röllin, "The chebyshev iteration revisited," *Parallel Computing*, vol. 28, no. 2, pp. 263 – 283, 2002.
- [22] 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*, Nov 2010, pp. 1–11.