Application of a Preconditioned Chebyshev Basis Communication-Avoiding Conjugate Gradient Method to a Multiphase Thermal-Hydraulic CFD Code

Yasuhiro Idomura[1(✉)], Takuya Ina[1], Akie Mayumi[1], Susumu Yamada[1], and Toshiyuki Imamura[2]

[1] Japan Atomic Energy Agency, Kashiwa, Chiba 227-0871, Japan
idomura.yasuhiro@jaea.go.jp
[2] RIKEN, Kobe, Hyogo 650-0047, Japan

Abstract. A preconditioned Chebyshev basis communication-avoiding conjugate gradient method (P-CBCG) is applied to the pressure Poisson equation in a multiphase thermal-hydraulic CFD code JUPITER, and its computational performance and convergence properties are compared against a preconditioned conjugate gradient (P-CG) method and a preconditioned communication-avoiding conjugate gradient (P-CACG) method on the Oakforest-PACS, which consists of 8,208 KNLs. The P-CBCG method reduces the number of collective communications with keeping the robustness of convergence properties. Compared with the P-CACG method, an order of magnitude larger communication-avoiding steps are enabled by the improved robustness. It is shown that the P-CBCG method is 1.38× and 1.17× faster than the P-CG and P-CACG methods at 2,000 processors, respectively.

1 Introduction

Krylov subspace methods are widely used for solving linear systems given by extreme scale sparse matrices, and thus, their scalability is one of critical issues towards exascale computing. In nuclear engineering, exascale computing is needed for Computational Fluid Dynamics (CFD) simulations of turbulent flows such as multiphase thermal-hydraulic simulations of nuclear reactors and fusion plasma simulations. In these CFD simulations, implicit solvers based on Krylov subspace methods occupy dominant computational costs, and the scalability of such CFD simulations largely depends on the performance of Krylov solvers.

The current Peta-scale machines are characterized by extreme concurrency reaching at ∼100 k computing nodes. In addition to this feature, on future exascale machines, which may be based on many-core processors or accelerators, significant acceleration of computation is expected. In Ref. [1], we optimized stencil computation kernels from CFD simulations on the latest many-core

R. Yokota and W. Wu (Eds.): SCFA 2018, LNCS 10776, pp. 257–273, 2018.
https://doi.org/10.1007/978-3-319-69953-0_15

processors and GPUs, and significant performance gains were achieved. However, the accelerated computation revealed severe bottlenecks of communication.

Krylov solvers involve local halo data communications for stencil computations or sparse matrix vector operations SpMVs, and global data reduction communications for inner product operations in orthogonalization procedures for basis vectors. Although communication overlap techniques [2] may reduce the former latency, it can not be applied to the latter. In order to resolve this issue at mathematics or algorithm levels, in Refs. [3,4], we have introduced communication-avoiding (CA) Krylov methods to a fusion plasma turbulence code GT5D [5] and a multiphase thermal-hydraulic CFD code JUPITER [6].

The implicit solver in the GT5D is well-conditioned, and the communication-avoiding general minimum residual (CA-GMRES) method [7] was stable for large CA-steps $s > 10$. On the other hand, the Poisson solver in the JUPITER is ill-conditioned, and the convergence of the left-preconditioned communication-avoiding conjugate gradient (P-CACG) method [7] was limited for $s \leq 3$. Even with $s = 3$, the strong scaling of the JUPITER on the K-computer [8] was dramatically improved by reducing the number of global data reduction communications to $1/s$. However, for practical use, it is difficult to operate CA Krylov solvers at the upper limit of CA-steps, because the Poisson operator is time dependent and its condition number may increase in time. Therefore, we need to use more robust CA Krylov methods at CA-steps well below the upper limit, beyond which they become numerically unstable. In order to resolve this issue, in this work, we introduce the preconditioned Chebyshev basis communication-avoiding conjugate gradient (P-CBCG) method to the JUPITER, and examine its robustness and computational performance on the Oakforest-PACS, which consists of 8,208 KNLs.

The reminder of this paper is organized as follows. Related works are reviewed in Sect. 2. In Sect. 3, we explain CA Krylov subspace methods used in this work. In Sect. 4, we discuss numerical properties and kernel performances of CA Krylov solvers. In Sect. 5, we present the convergence property of CA Krylov methods and the computational performances of CA Krylov solvers on the JAEA ICEX and the Oakforest-PACS. Finally, a summary is given in Sect. 6.

2 Related Works

The CACG method is based on the so-called s-step CG method, in which the data dependency between SpMV and inner product operations in the standard CG method is removed. Van Rosendale [9] first developed a s-step version of the CG method. Chronopoulos and Gear [10] called their own variant of the CG method as the s-step CG method. However, the above works did not change SpMV operations for generating the s-step basis. Toledo optimized the computation of the s-step basis in the s-step CG method [11], in which the number of words transferred between levels of the memory hierarchy is reduced. The CACG method by Hoemmen [7] reduced communications between levels of the memory hierarchy and between processors by a matrix power kernel (MPK) [12].

Carson [13] showed the performance of the CACG method on the Hopper supercomputer using a simple Poisson model problem.

CA-preconditioning is based on sparse approximate inverses with the same sparsity pattern as the matrix A, or block Jacobi (BJ) and polynomial preconditioners [9,11,14]. For instance, in BJ preconditioning, each processor independently solves its local problem. However, when the local preconditioner has data dependency over the whole local problem as in ILU factorization, it is difficult to construct a MPK without additional communications, because each local SpMV requires preconditioned input vector elements from neighboring processors. To avoid the additional communications, Yamazaki et al. [15] 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 our previous work [3], it was shown that for ill-conditioned problems given by the JUPITER, 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 [4,13,15], CA Krylov methods were applied to well-conditioned problems, where CA-steps are extended for $s > 10$. However, in Ref. [3], it was shown that for ill-conditioned problems given by the JUPITER, 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 the other eigen-components become relatively smaller and are hidden by the round-off errors. This violates the linear independency of the monomial basis vectors, and makes them ill-conditioned, when each basis vectors are not orthogonalized after creating it. To resolve this issue, Hoemmen [7] proposed to use the Newton basis vectors and the Chebyshev basis vectors. Suda et al. [16] proposed the P-CBCG method, which was tested with point Jacobi preconditioning on the K-computer [17]. In this work, we apply the P-CBCG method with BJ preconditioning to the JUPITER, compare its convergence property and numerical stability against the P-CACG method, and demonstrate its computational performance on the Oakforest-PACS.

3 Krylov Solvers in JUPITER Code

3.1 Code Overview

In the JUPITER code [6], 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 (∼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 subspace methods explained in the following subsections. 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 is commonly used in many matrix libraries, 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 decomposition in z is applied to each domain via OpenMP. BJ preconditioning is applied to each fine subdomain so that it is computed in thread parallel.

3.2 Preconditioned Conjugate Gradient (P-CG) Method

In the original version of the JUPITER, the pressure Poisson equation was computed using the P-CG method [18] with BJ preconditioning, in which ILU factorization [18] is applied to each block. 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 a local halo data communication per iteration, and the inner product operations need two global data 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.

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** $j = 1, 2, ...$ 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**

3.3 Preconditioned Communication-Avoiding Conjugate Gradient (P-CACG) Method

The P-CACG method in Algorithm 2 [7] is based on a three term recurrence variant of CG (CG3) method [18]. The CG3 method is decomposed into the outer loop and the inner s-step loop, and the algorithm is modified so that the latter is processed without any communication. At the k-th outer loop, firstly, the

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** $k = 0, 1, 2, ...$ 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 $Aq_{sk+j} = [Z_{k-1}, \underline{W}_k]d_{sk+j}$ and $M^{-1}Aq_{sk+j} = [Q_{k-1}, \underline{V}_k]d_{sk+j}$
11: Compute g_{sk+j} that satisfies $z_{sk+j} = [Z_{k-1}, \underline{W}_k]g_{sk+j}$ and $q_{sk+j} = [Q_{k-1}, \underline{V}_k]g_{sk+j}$
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**

s-step monomial basis vectors $\underline{V}_k$ (line 5) and the corresponding preconditioned basis vectors $\underline{W}_k$ (line 6) are generated at once. Secondly, the Gram matrix G_k (line 8) is computed for the inner product operations, which are replaced as $\mu = \langle z, q \rangle = g_{sk+j}^* G_k g_{sk+j}$ (line 12) and $\nu = \langle Aq, q \rangle = g_{sk+j}^* G_k d_{sk+j}$ (line 13). Here, d_{sk+j} (line 10) and g_{sk+j} (line 11) are defined so that they satisfy $Aq_{sk+j} = [Z_{k-1}, \underline{W}_k]d_{sk+j}$ and $z_{sk+j} = [Z_{k-1}, \underline{W}_k]g_{sk+j}$, respectively. Here, x^* denotes its transpose. At the j-th inner loop, these coefficients are computed to obtain the inner products, and then, the solution vector x_{sk+j} (line 22) and two sets of the residual vectors, the unpreconditioned residual vector z_{sk+j} (line 24) and the preconditioned residual vector q_{sk+j} (line 23), are updated by the three term recurrence formulae. In exact arithmetic, s iterations of the P-CG3 method and one outer loop iteration of the P-CACG method are equivalent.

In Ref. [3], we compared convergence properties of the pressure Poisson solver between the original BJ preconditioning and CA preconditioning based on the

underlap approach [15], and significant convergence degradation was observed with the latter preconditioning. In addition, if one uses a MPK with CA preconditioning, s-step halo data is transferred at once. The number of halo data communication directions are significantly increased from 6 (bidirectional in x, y, z) to 26 (including three dimensional diagonal directions), and redundant computations are needed for the halo data. In order to avoid these issues, in this work, we use the BJ preconditioning with a hybrid CA approach [3]. In the P-CACG method, dominant computational costs come 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 one local 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.

3.4 Preconditioned Chebyshev Basis Communication-Avoiding Conjugate Gradient (P-CBCG) Method

The P-CBCG method [16] is shown in Algorithm 3. Unlike the P-CACG method which is based on the CG3 method, the P-CBCG method computes two term recurrences as in the P-CG method. The inner product operations are performed using the so-called look-unrolling technique [9] instead of a Gram matrix approach in the P-CACG method. A multi-step CG method constructed using the above approach is computed using s-step Chebyshev basis vectors. Here, the preconditioned Chebyshev basis vectors (line 10) are computed using Algorithm 4. In this algorithm, the basis vectors are generated using $T_j(AM^{-1})$, which is the j-th Chebyshev polynomials scaled and shifted within $[\lambda_{\min}, \lambda_{\max}]$ and thus, satisfies $|T_j(AM^{-1})| < 1$, where $\lambda_{\min}$ and $\lambda_{\max}$ are the minimum and maximum eigenvalues of AM^{-1}. In the monomial basis vectors, the generated vectors are aligned to the eigenvector with $\lambda_{\max}$ as s increases, and the other eigencomponents become relatively smaller and are hidden by the round-off errors. This violates the linear independency of the monomial basis vectors, and makes them ill-conditioned, when each basis vector is not orthogonalized after creating it. On the other hand, the minimax property of Chebyshev polynomials helps to keep the basis vectors well-conditioned without orthogonalizing each basis vector. By using this method, one can construct a Krylov subspace, which is mathematically equivalent to that given by the monomial basis vectors, with much less impact of the round-off errors, and s can be extended compared with

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 $(T_0(AM^{-1})r_0, T_1(AM^{-1})r_0, ..., T_{s-1}(AM^{-1})r_0\ )$
3: $Q_0 = S_0$
4: **for** $k = 0, 1, 2, ...$ 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: Compute S_{k+1} $(T_0(AM^{-1})r_{s(k+1)}, T_1(AM^{-1})r_{s(k+1)}, ..., T_{s-1}(AM^{-1})r_{s(k+1)}\ )$
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: $AQ_{k+1} := AS_{k+1} + AQ_k B_k$
15: **end for**

CA Krylov methods based on the monomial basis vectors. In this work, $\lambda_{\max}$ is computed by a power method, while $\lambda_{\min}$ is approximated as zero.

In the P-CBCG method, 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, 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 sends s^2 elements of $Q_k^* A S_{k+1}$.

Algorithm 4. Preconditioned Chebyshev basis

Input: r_{sk}, Approximate minimum/maximum eigenvalues of AM^{-1}, $\lambda_{\min}, \lambda_{\max}$
Output: S_k $(\tilde{z}_0, \tilde{z}_1, ..., \tilde{z}_{s-1})$, AS_k $(A\tilde{z}_0, A\tilde{z}_1, ..., A\tilde{z}_{s-1})$
1: $\eta := 2/(\lambda_{\max} - \lambda_{\min})$
2: $\zeta := (\lambda_{\max} + \lambda_{\min})/(\lambda_{\max} - \lambda_{\min})$
3: $z_0 := r_{sk}$
4: $\tilde{z}_0 := M^{-1} z_0$
5: $z_1 := \eta A \tilde{z}_0 - \zeta z_0$
6: $\tilde{z}_1 := M^{-1} z_1$
7: **for** $j = 2, 3, ..., s$ **do**
8: $z_j := 2\eta A \tilde{z}_{j-1} - 2\zeta z_{j-1} - z_{j-2}$
9: $\tilde{z}_j := M^{-1} z_j$
10: **end for**

4 Kernel Performance Analysis

4.1 Computing Platforms

In this work, we estimate computing performances of the P-CG, P-CACG, and P-CBCG solvers on computing platforms in Table 1. The JAEA ICEX is based on the Xeon E5-2680v3 processor (Haswell) and the dual plane $4 \times$ FDR Infiniband with a hyper cube topology, and the Oakforest-PACS (KNL) consists of the Xeon Phi 7250 processor (Knights Landing) and the Omni Path with a fat tree topology. The compilers used are the Intel Fortran compiler 16.0.1 with the Intel MPI library 5.0 (-O3 -mcmodel= large -qopenmp -fpp -align array64byte -no-prec-div -fma -xHost) and the Intel Fortran compiler 17.0.4 with the Intel MPI library 2017 (-O3 -mcmodel= large -qopenmp -fpp -align array64byte -no-prec-div -fma -axMIC-AVX512) on ICEX and KNL, respectively. In this work, cross-platform comparisons are performed using the same number of processors. Since ICEX is based on a NUMA architecture with two processors per node, we assign two MPI processes per node. As for OpenMP parallelization, we use 12 and 68 threads on ICEX and KNL, respectively. On KNL, we choose 68 threads without hyper threading to avoid performance degradation in MPI communications [4]. Although KNL has hierarchical memory architecture consisting of MCDRAM (16 GByte, B$\sim$480 GByte/s) and DDR4 (96 GByte), we suppress the problem size below 16 GB per node and use only MCDRAM in a flat mode.

In this section, we analyze a single processor performance for the three solvers using a small problem size of $N = 104 \times 104 \times 265$, which corresponds to a typical problem size on a single processor. The achieved performance is compared against the modified roofline model [19], in which a theoretical processing time of each kernel is estimated by the sum of costs for floating point operations and memory access, $t_{RL} = f/F + b/B$. Here, f and b are the numbers of floating point operations and memory access of the kernel. F and B are the

Table 1. Specifications of the JAEA ICEX and the Oakforest-PACS (KNL).

	ICEX	KNL
Number of nodes	2,510	8,208
Total performance [PFlops]	2.41	25.00
Number of cores per node	12×2	68
Peak performance F [GFlops/processor]	480	3046
STREAM bandwidth B [GByte/s/processor]	58	480(MCDRAM)
B/F	0.12	0.16
Cache [MB/cores]	30/12	1/2
Memory per node [GByte]	64	16
Interconnect bandwidth [GByte/s]	13.6	12.5

peak performance and the STREAM memory bandwidth of the processor. The performance ratios of F and B between ICEX and KNL are 6.3× and 8.6×, respectively.

Table 2. Kernel performance analysis (Floating point operation f [Flop/grid], Memory access b [Byte/grid], Arithmetic intensity f/b, Roofline time $t_{RL} = f/F + b/B$ [ns/grid], Peak performance F [Flops], STREAM bandwidth B [Byte/s], Elapse time t [ns/grid], Sustained performance P [GFlops], Roofline ratio $R_{RL} = t_{RL}/t$, and ICEX/KNL ratio R_{ICEX}) in the kernel benchmarks using a single processor of ICEX and KNL.

Algorithm	Kernel	f	b	f/b	ICEX				KNL				
					t_{RL}	t	P	R_{RL}	t_{RL}	t	P	R_{RL}	R_{ICEX}
P-CG	SpMV	15.00	80.00	0.19	1.40	1.94	7.75	0.72	0.17	0.28	52.78	0.60	6.81
	BJ	20.00	128.00	0.16	2.24	2.53	7.90	0.88	0.27	0.46	43.26	0.59	5.48
	Vector	4.00	40.00	0.10	0.69	0.72	5.55	0.96	0.08	0.10	39.74	0.84	7.16
	Total	39.00	248.00	0.16	4.33	5.19	7.52	0.84	0.53	0.85	46.03	0.62	6.12
P-CACG ($s = 3$)	SpMV	13.00	80.00	0.16	1.40	1.83	7.11	0.76	0.17	0.24	54.59	0.72	7.68
	BJ	14.00	120.00	0.12	2.09	2.02	6.94	1.03	0.25	0.44	31.87	0.58	4.59
	Gram	18.67	29.33	0.64	0.54	0.54	34.70	1.01	0.07	0.13	142.86	0.51	4.12
	3-term	41.67	80.00	0.52	1.46	1.42	29.26	1.02	0.18	0.49	84.20	0.36	2.88
	Total	87.33	309.33	0.28	5.49	5.81	15.04	0.94	0.67	1.30	67.03	0.52	4.46
P-CBCG ($s = 12$)	CB	30.58	228.67	0.13	3.98	4.91	6.22	0.81	0.49	0.89	34.46	0.55	5.54
	Matrix	93.17	83.33	1.12	1.62	1.79	51.98	0.91	0.20	0.63	147.14	0.32	2.83
	Total	123.75	312.00	0.40	5.61	6.71	18.45	0.84	0.69	1.52	81.38	0.45	4.41

4.2 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 the following 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 are summarized in Table 2. 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, and the performance ratio between ICEX and KNL exceeds $R_{ICEX} > 6$. Although AXPYs in Vector achieve ideal sustained performances with the performance ratio against the roofline model $R_{RL} \sim 0.9$ both on ICEX and KNL, stencil computations in SpMV and BJ show performance degradation from $R_{RL} \sim 0.8$ on ICEX to $R_{RL} \sim 0.6$ on KNL.

4.3 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 [3]. In Table 2, the kernel performance is summarized at $s = 3$, which is the upper limit of CA-steps in the benchmark problem in Sect. 5. 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. Compared with SpMV and BJ, Gram and 3-term have higher arithmetic intensity $f/b > 0.5$, and thus, an impact of additional computation in the P-CACG method on the total computational cost ($\sim$1.12$\times$ on ICEX) is much lower than the increase of f ($\sim$2.24$\times$) from the P-CG method. These compute-intensive kernels achieve ideal performances with $R_{RL} \sim 1$ on ICEX. However, they show significant performance degradation with $R_{RL} \sim 0.4$ on KNL, and thus, the performance ratio is limited to $R_{ICEX} \sim 4.46$.

4.4 P-CBCG Solver

Computational kernels of the P-CACG 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 2, the kernel performance is summarized for $s = 12$, which is used in the benchmark problem in Sect. 5. 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 1.3$\times$ on ICEX, because of the improved arithmetic intensity. However, on KNL, the performance ratio between the P-CG and P-CBCG methods is expanded to 1.79$\times$, because the compute-intensive Matrix kernel shows performance degradation from $R_{RL} \sim 0.9$ on ICEX to $R_{RL} \sim 0.3$ on KNL. As a result, the performance ratio between ICEX and KNL is limited to $R_{ICEX} \sim 4.41$.

5 Numerical Experiment

5.1 Convergence Property

In the present numerical experiment, we compute nonlinear evolutions of molten debris in a single fuel assembly component of nuclear reactor (see Fig. 1). 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 [3,6]. The problem treats multi-phase 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 multi-phase flows, which give the largest iteration number. Because of the large problem size and the extreme density contrast of multi-phase flows, the pressure Poisson equation is ill-conditioned, and the P-CG solver is converged

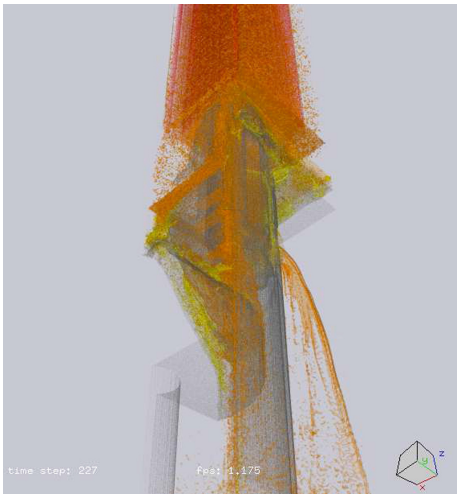

Fig. 1. Visualization of nonlinearly evolved multiphase flows of molten debris in reactor internal components computed by the JUPITER with $N = 800 \times 500 \times 3,540$.

with $\sim 6,000$ iterations (see Fig. 2). Here, the convergence condition is given by the relative residual error of $|b - Ax|/|b| < 10^{-8}$.

The convergence properties of the P-CG, P-CACG, P-CBCG, and P-MBCG solvers are summarized in Fig. 2. Here, the P-MBCG method is a variant of the P-CBCG method, in which the Chebyshev basis vectors at lines 2, 10 are replaced by the monomial basis vectors $S_k(r_{sk}, (AM^{-1})r_{sk}, (AM^{-1})^2 r_{sk}, ..., (AM^{-1})^{s-1} r_{sk})$. Although the P-MBCG method is mathematically similar to the P-CACG method, the former uses the two term recurrence formulae, while the latter is based on the CG3 method or the three term recurrence formulae. In Ref. [20], it was shown that Krylov subspace methods based on three term recurrences give significantly less accurate residuals than those with two term recurrences. In this work, we examine this point for CA Krylov subspace methods. As shown in Ref. [3], the convergence of the P-CACG solver is limited to $s = 3$, while in the P-MBCG solver, the convergence is somewhat extended to $s = 5$. On the other hand, in the P-CBCG method, the convergence property is dramatically extended to $s = 40$. These observations show that the main cause of the convergence degradation is not the three term recurrence formulae, but the ill-conditioned monomial basis vectors. Another important property is in the P-CBCG solver, the convergence property becomes worse gradually above the upper limit of CA-steps, while the P-CACG and P-MBCG solvers breaks down immediately above the upper limit. This property is important for practical use in extreme-scale CFD simulations.

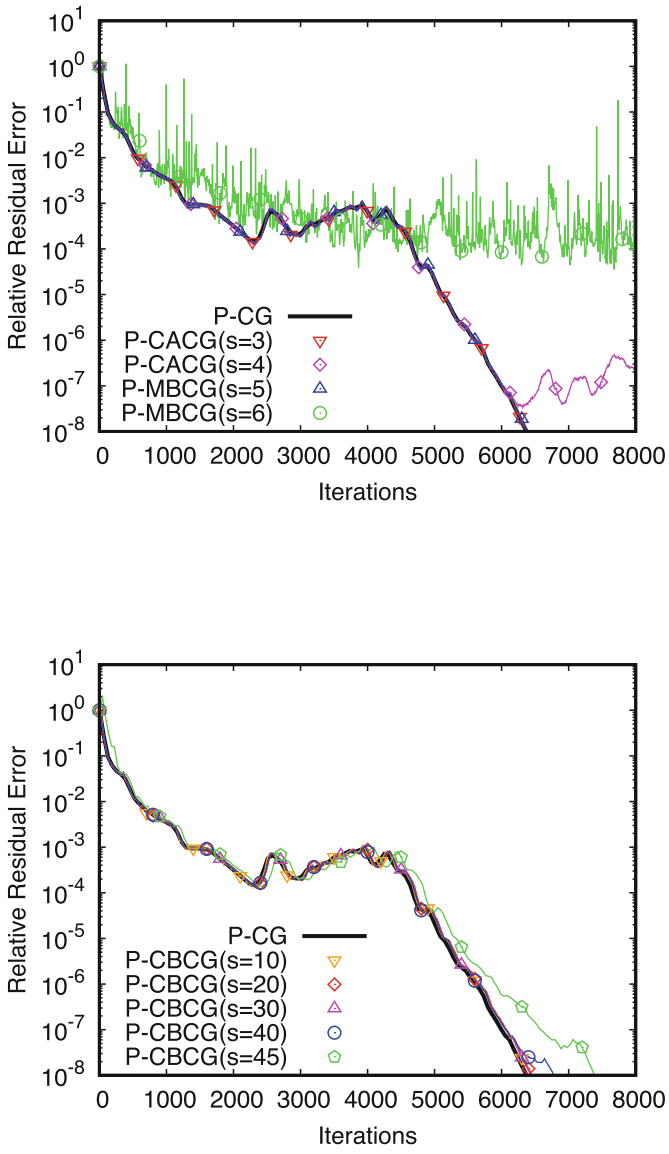

Fig. 2. Comparisons of convergence properties among the P-CG, P-CACG, P-CBCG, and P-MBCG (a variant of the P-CBCG method using the monomial basis vectors) solvers. The relative residual error $|b - Ax|/|b|$ is plotted for the JUPITER with $N = 800 \times 500 \times 3,540$. The computation is performed using 800 processors on ICEX.

5.2 Strong Scaling Test

In the P-CACG solver, we use $s = 3$, which is the upper limit of CA-steps from the viewpoint of numerical stability. On the other hand, the choice of s in the

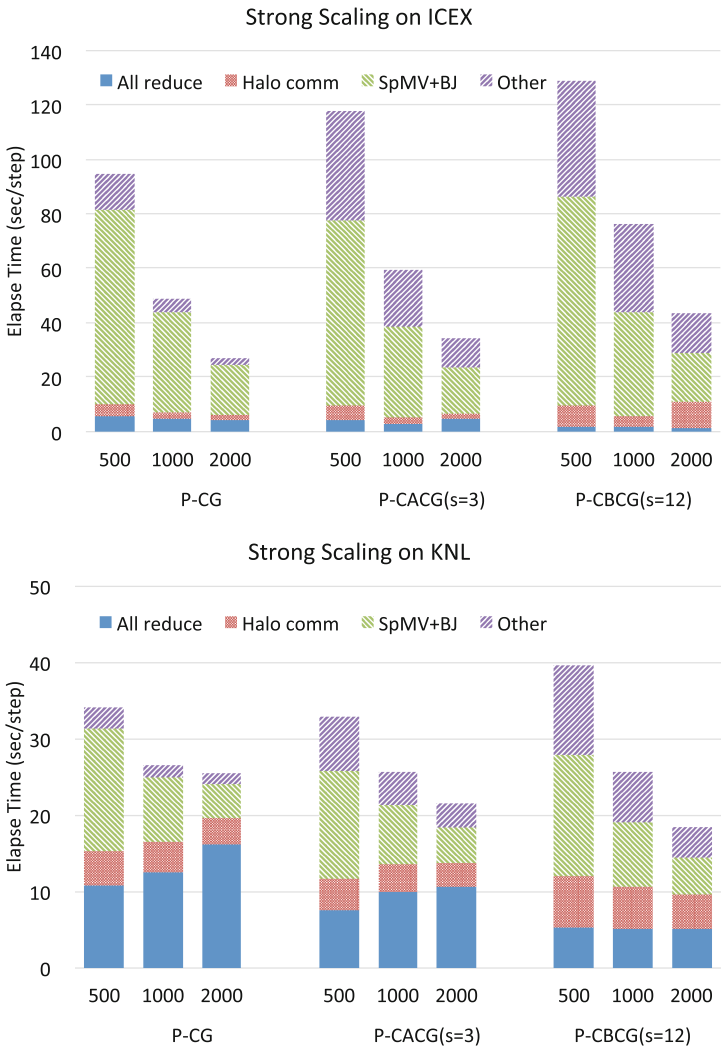

Fig. 3. Strong scaling of the P-CG, P-CACG($s = 3$), and P-CBCG($s = 12$) solvers using 500, 1,000, and 2,000 processors (MPI processes) on ICEX and KNL. The cost distribution in a single time step is shown.

P-CBCG solver is rather flexible, and the optimum s depends on the following factors. Firstly, the number of All_reduce is reduced to $1/s$ compared to the P-CG method. Here, the communication data size scales as $\sim s^2$. Secondly, the numbers of floating point operations and memory access per iteration in Matrix kernel scale as $f \sim s$ and $b \sim const.$, respectively, and the arithmetic intensity of Matrix scales as $f/b \sim s$. Thirdly, cache efficiency of CB is affected by the number of basis vectors. Therefore, the computational performance of each kernel varies depending

on s. Finally, the communication performance is also affected when the data size is changed from a latency bound regime to a bandwidth bound regime. Although a simple performance model was presented in Refs. [13,17], we need more detailed performance models to predict the above complex behaviors. In this work, we chose $s = 12$ from s-scan numerical experiments.

The strong scaling of the P-CG, P-CACG, and P-CBCG solvers are summarized in Fig. 3. In the strong scaling test, we use 500, 1,000, and 2,000 processors on ICEX and KNL, respectively. On ICEX, all Krylov solvers show good strong scaling, because the computation part is dominant in all cases and the communication part is suppressed below $\sim$10 s. Therefore, the P-CACG and P-CBCG solvers are slower than the P-CG solver, because of additional computation in CA Krylov methods. On the other hand, on KNL, the computation part is significantly accelerated ($3.5\times \sim 5.1\times$) and the communication part is comparable or slower ($0.3\times \sim 1.1\times$) compared to ICEX. Here, the cause of slower communication performance on KNL is still under investigation. As a result, the remaining communication part, in particular, All_reduce becomes a severe bottleneck. On KNL, the cost of All_reduce in the P-CG solver increases with the number of processors. This tendency is observed even in the P-CACG solver. However, in the P-CBCG solver, the cost increase of All_reduce is suppressed, and at 2,000 processors, it is reduced to $\sim 1/3$ and $\sim 1/2$ compared to the P-CG and P-CACG solvers, respectively. Because of this CA feature, the best performance on KNL is obtained by the P-CBCG solver, and the P-CBCG solver is $1.38\times$ and $1.17\times$ faster than the P-CG and P-CACG solvers at 2,000 processors, respectively.

It is noted that in Ref. [3], the P-CACG solver on the K-computer showed ideal cost reduction of All_reduce by $1/s$. However, in the present numerical experiment, the cost reduction of All_reduce from the P-CG solver is limited to $\sim 2/3$ and $\sim 1/3$ in the P-CACG and P-CBCG solvers, respectively. These performance ratios are far above the ideal one $1/s$. In order to understand this issue, more detailed performance analysis for All_reduce is needed. Another issue is that the cost of halo data communications increases in the P-CBCG solver, while the number of SpMVs is almost the same as the other solvers. It is confirmed that this cost becomes comparable to that in the P-CACG solver, when the number of CA-steps is reduced to $s = 3$. Therefore, the performance degradation of halo data communications seems to depend on the memory usage, which increases with s. These issues will be addressed in the future work.

6 Summary

In this work, we applied the P-CBCG method to the pressure Poisson equation in the JUPITER. We analyzed numerical properties of the P-CACG and P-CBCG methods in detail, and compared it against the P-CG method, which was used in the original code. The P-CACG and P-CBCG methods reduce data reduction communications to $1/s$, but additional computation is needed for CA procedures. The P-CACG ($s = 3$) and P-CBCG ($s = 12$) methods have $\sim 2\times$ and $\sim 3\times$ larger f, while the increase in b is only $\sim 1.25\times$. Because of the improved arithmetic intensity f/b, the resulting computational costs of the P-CACG and P-CBCG solvers

on a single processor were respectively suppressed to ∼1.1× and ∼1.3× on ICEX, while they were expanded to ∼1.5× and ∼1.8× on KNL.

We tested convergence properties of CA Krylov subspace methods based on the monomial basis vectors (P-CACG, P-MBCG) and the Chebyshev basis vectors (P-CBCG). In the comparison between the P-CACG and P-MBCG methods, which are based on three term recurrences and two term recurrences, the latter showed slightly improved convergence. However, the convergence of both solvers were limited for $s \ll 10$. On the other hand, the convergence of the P-CBCG method was extended to $s \sim 40$, and the robustness of CA Krylov solvers was dramatically improved.

Strong scaling tests of the P-CG, P-CACG ($s = 3$), and P-CBCG ($s = 12$) solvers were performed using 500, 1,000, and 2,000 processors on ICEX and KNL. On ICEX, the computational costs were dominated by the computation part, and all three solvers showed good strong scaling. As the communication part is minor, the P-CG solver was fastest on ICEX. On the other hand, on KNL, the computation part was significantly accelerated, and the remaining communication part, in particular, All_reduce became a severe bottleneck. By reducing the cost of All_reduce, the best performance was achieved by the P-CBCG solver, and the P-CBCG solver is 1.38× and 1.17× faster than the P-CG and P-CACG solvers at 2,000 processors, respectively. As the P-CBCG method satisfies both high computational performance and excellent robustness, it is promising algorithm for extreme scale simulations on future exascale machines with limited network and memory bandwidths.

Acknowledgement. The authors would like to thank Dr. S. Yamashita for providing the JUPITER for the present benchmark, and Dr. T. Kawamura for the visualization image. This work is supported by the MEXT (Grant for Post-K priority issue No.6: Development of Innovative Clean Energy). Computations were performed on the Oakforest-PACS (Univ. Tokyo/Univ. Tsukuba) and the ICEX (JAEA).

References

1. Asahi, Y., et al.: Optimization of fusion Kernels on accelerators with indirect or strided memory access patterns. IEEE Trans. Parallel Distrib. Syst. **28**(7), 1974–1988 (2017)
2. Idomura, Y., et al.: Communication-overlap techniques for improved strong scaling of Gyrokinetic Eulerian code beyond 100k cores on the K-computer. Int. J. High Perform. Comput. Appl. **28**(1), 73–86 (2014)
3. Mayumi, A., 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 2016, Piscataway, NJ, USA, pp. 17–24. IEEE Press (2016)
4. Idomura, Y., Ina, T., Mayumi, A., Yamada, S., Matsumoto, K., Asahi, Y., Imamura, T.: 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 2017, New York, NY, USA, pp. 7:1–7:8. ACM (2017)

5. Idomura, Y., et al.: Study of ion turbulent transport and profile formations using global gyrokinetic full-f Vlasov simulation. Nucl. Fusion **49**, 065029 (2009)
6. Yamashita, S., Ina, T., Idomura, Y., Yoshida, H.: A numerical simulation method for molten material behavior in nuclear reactors. Nucl. Eng. Des. **322**(Suppl. C), 301–312 (2017)
7. Hoemmen, M.: Communication-avoiding Krylov subspace methods. Ph.D. thesis, University of California, Berkeley (2010)
8. Fujitsu Global: K computer. http://www.fujitsu.com/global/about/businesspolicy/tech/k/
9. Van Rosendale, J.: Minimizing inner product data dependencies in conjugate gradient iteration. NASA contractor report (1983)
10. Chronopoulos, A., Gear, C.: s-step iterative methods for symmetric linear systems. J. Comput. Appl. Math. **25**(2), 153–168 (1989)
11. Toledo, S.A.: Quantitative performance modeling of scientific computations and creating locality in numerical algorithms. Ph.D. thesis, Massachusetts Institute of Technology (1995)
12. Demmel, J., Hoemmen, M., Mohiyuddin, M., Yelick, K.: Avoiding communication in sparse matrix computations. In: 2008 IEEE International Symposium on Parallel and Distributed Processing, pp. 1–12, April 2008
13. Carson, E.C.: Communication-avoiding Krylov subspace methods in theory and practice. Ph.D. thesis, University of California, Berkeley (2015)
14. Chronopoulos, A., Gear, C.W.: Implementation of preconditioned s-step conjugate gradient methods on a multiprocessor system with memory hierarchy. Technical report, Department of Computer Science, Illinois University, Urbana, USA (1987)
15. Yamazaki, I., Anzt, H., Tomov, S., Hoemmen, M., Dongarra, J.: Improving the performance of CA-GMRES on multicores with multiple GPUs. In: 2014 IEEE 28th International Parallel and Distributed Processing Symposium, pp. 382–391, May 2014
16. Suda, R., Cong, L., Watanabe, D., Kumagai, Y., Fujii, A., Tanaka, T.: Communication-avoiding CG method: new direction of Krylov subspace methods towards exa-scale computing. RIMS Kôkyûroku **1995**, 102–111 (2016)
17. Kumagai, Y., Fujii, A., Tanaka, T., Hirota, Y., Fukaya, T., Imamura, T., Suda, R.: Performance analysis of the Chebyshev basis conjugate gradient method on the K computer. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K., Kitowski, J., Wiatr, K. (eds.) PPAM 2015. LNCS, vol. 9573, pp. 74–85. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-32149-3_8
18. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. Society for Industrial and Applied Mathematics, Philadelphia (2003)
19. Shimokawabe, T., 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, November 2010
20. Gutknecht, M.H., Strakos, Z.: Accuracy of two three-term and three two-term recurrences for Krylov space solvers. SIAM J. Matrix Anal. Appl. **22**(1), 213–229 (2000)