Communication-Computation Overlapping for Preconditioned Parallel Iterative Solvers with Dynamic Loop Scheduling

Kengo Nakajima[*]
The University of Tokyo
Tokyo, Japan
nakajima@cc.u-tokyo.ac.jp

Balazs Gerofi[†]
RIKEN Center for Computational Science (R-CCS)
Kobe, Japan
bgerofi@riken.jp

Masashi Horikoshi[‡]
Intel Corporation
Tokyo, Japan
masashi.horikoshi@intel.com

Yutaka Ishikawa[§]
National Institute of Informatics
Tokyo, Japan
yutaka.ishikawa@nii.ac.jp

ABSTRACT

Preconditioned parallel solvers based on the Krylov iterative method are widely used in scientific and engineering applications. Communication overhead is a critical issue when executing these solvers on large-scale massively parallel supercomputers. In the previous work, we introduced communication-computation overlapping with dynamic loop scheduling of OpenMP to the sparse matrix-vector multiplication (SpMV) process of a parallel iterative solver by Conjugate Gradient (CG) method in a parallel finite element application (GeoFEM/Cube) on multicore and manycore clusters. In the present work, first, we re-evaluated the method on our new system, Wisteria/BDEC-01 (Odyssey) (Fujitsu PRIMEHPC FX1000 with A64FX), and a significant performance improvement of 25-30% for parallel iterative solver at 2,048 nodes (98,304 cores) was obtained. Moreover, we proposed a new reordering method for communication-computation overlapping in ICCG solvers for a parallel finite volume application (Poisson3D/Dist), and attained 5-12% improvement at 1,024 nodes of Odyssey.

CCS CONCEPTS

• **Computing methodologies** → Parallel computing methodologies.

KEYWORDS

Parallel Iterative Solvers, Communication-Computation Overlapping, Dynamic Loop Scheduling, Multicoloring, Reordering

ACM Reference Format:
Kengo Nakajima, Balazs Gerofi, Masashi Horikoshi, and Yutaka Ishikawa. 2022. Communication-Computation Overlapping for Preconditioned Parallel Iterative Solvers with Dynamic Loop Scheduling. In *International Conference on High Performance Computing in Asia-Pacific Region Workshops (HPCAsia 2022 Workshop), January 11–14, 2022, Virtual Event, Japan.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3503470.3503477

1 INTRODUCTION

Preconditioned parallel solvers based on the Krylov iterative method are widely used in scientific and engineering applications. The overhead in global communications is a critical issue when executing these solvers on large-scale massively parallel supercomputers. The method for *communication-computation overlapping* (CC-Overlapping) in halo exchanges [1] is a well-known remedy for this in stencil computation.

In the previous work [2], we introduced CC-Overlapping with dynamic loop scheduling of OpenMP for improvement of the performance of the *sparse matrix-vector multiplication* (SpMV) process of the parallel iterative solver by Conjugate Gradient (CG) method in a parallel finite element application (GeoFEM/Cube) [2][3][4] based on the method in [5], and performance improvement of 10-15 % was obtained at 60 nodes (3,840 cores) of the Oakforest-PACS at JCAHPC [6][7] with Intel Xeon/Phi (Knights Landing, KNL) manycore processors [2]. In the present work, first, we evaluated the code in [2] to our new system, Wisteria/BDEC-01 (Odyssey), Information Technology Center, the University of Tokyo, which is based on Fujitsu PRIMEHPC FX1000 with A64FX [8][9]. Then, we proposed a new reordering method for CC-Overlapping in ICCG (Conjugate Gradient with Incomplete Cholesky factorization without fill-in) solvers for a parallel finite volume application (Poisson3D/Dist). Performance of the proposed method was evaluated on both of Oakforest-PACS (OFP) and Wisteria/BDEC-01 (Odyssey) (ODY). Furthermore, effects of IHK/McKernel [10] were also evaluated on the OFP.

The rest of this paper is organized as follows. In Section 2, an overview of the first application (GeoFEM/Cube) is provided. In Section 3, we summarize CC-Overlapping and related work in reducing communication overhead in parallel computing on massively parallel supercomputers, and introduce CC-Overlapping by dynamic loop scheduling of OpenMP. Section 4 describes the hardware environments and IHK/McKernel, and Section 5 gives preliminary results for GeoFEM/Cube. Section 6 and Section 7 give overview of the second application (Poisson3D/Dist) using ICCG solver with description of special reordering method, and results of numerical experiments by Poisson3D/Dist. Finally, Section 8 offers summary and remark

2 GEOFEM/CUBE

GeoFEM/Cube [2][3][4] solves 3D linear elasticity problems in simple cube geometries using a parallel finite element method (FEM). Tri-linear hexahedral elements are used for the discretization. Material properties are defined as homogeneous, where Poisson's ratio is set to 0.30 for all elements and Young's modulus is 1.00. The boundary conditions are described in Fig. 1. Large-scale linear equations with sparse matrices derived from the application are solved by preconditioned Krylov iterative methods, such as Conjugate Gradient (CG) method, where the derived coefficient matrices are symmetric-positive-definite (SPD) [11]. The original GeoFEM/Cube adopts the ICCG (Conjugate Gradient with Incomplete Cholesky factorization without fill-in) method [3][4][11]. In the previous work [2], we evaluated the effects of CC-Overlapping by the code using conjugate gradient (CG) method with block diagonal LU factorization.

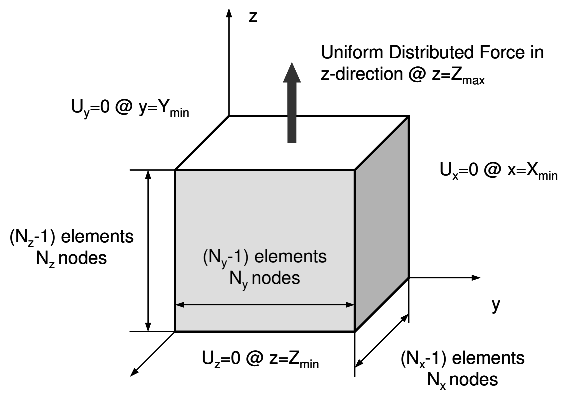


Figure 1: Simple Cube Geometry for 3D Linear Elasticity Problems in GeoFEM/Cube [2][3][4]

The preconditioner by block diagonal LU factorization utilizes the mathematical and physical properties of GeoFEM/Cube. Because GeoFEM/Cube solves 3D solid mechanics problems, there are three degrees of freedom (DOF) (u,v,w) (displacement component in x-, y-, and z-direction) at each finite-element node. Block diagonal LU factorization preconditioning applies full LU factorization to each 3×3 diagonal block of the coefficient matrix and applies forward/backward substitution to each block at the preconditioning stage. This preconditioning is easy to parallelize and more robust than Point-Jacobi preconditioning [11]. The number of iterations until convergence is not affected by the number of MPI processes when this preconditioning method is applied [2][3][4].

The GeoFEM/Cube application is parallelized by domain decomposition using the Message-Passing Interface (MPI) [2][3][4]. In the OpenMP/MPI hybrid parallel programming model, multithreading by OpenMP is applied to each partitioned domain. GeoFEM/Cube is written in Fortran with MPI and OpenMP. In GeoFEM/Cube, the entire model is divided into domains, and each domain is assigned to an MPI process, as shown in Fig. 2. The local data structure in GeoFEM/Cube is node-based with overlapping elements, and this type of procedure is appropriate for the preconditioned iterative solvers

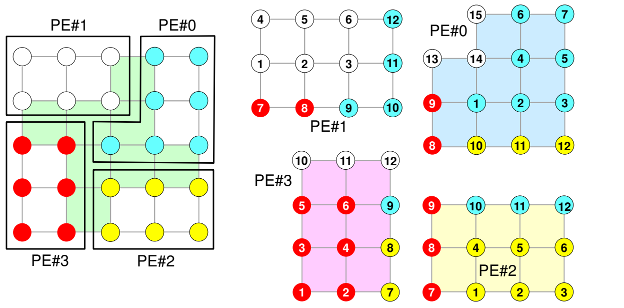
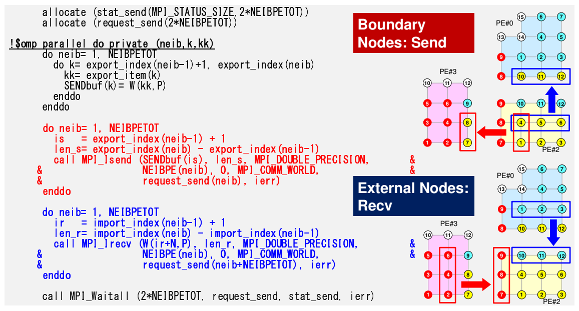


Figure 2: Node-based Partitioning in GeoFEM/Cube [2][3][4]

used in GeoFEM/Cube. Finite-element nodes are classified into the following three categories from the viewpoint of message passing: (1) internal nodes, (2) external nodes imported from other domains (halo exchanges), and (3) boundary nodes, which are external nodes of other domains. Tables of communication between neighboring domains are included in the local data. Values on boundary nodes in the domains are sent to the neighboring domains and are received as external nodes at the destination domain. This data structure shown in Fig. 2 and the communication method shown in Fig. 3 provide excellent parallel efficiency [2][3][4]. This type of communication occurs in the SpMV of Krylov iterative solvers. In GeoFEM/Cube, coefficient matrices for linear solvers are assembled in each domain according to FEM operations. This process can be performed without communication among processors using the information of overlapping elements.GeoFEM/Cube generates distributed local meshes and coefficient matrices in a parallel manner using MPI. In GeoFEM/Cube, the total number of vertices in each direction (Nx, Ny, Nz) and the number of partitions in each direction (Px, Py, Pz) are specified before computation. The total number of MPI processes is equal to PxxPyxPz, and each MPI process has (Nx/Px) × (Ny/Py) × (Nz/Pz) vertices.



```
      allocate (stat_send(MPI_STATUS_SIZE,2*NEIBPETOT))
      allocate (request_send(2*NEIBPETOT))

!$omp parallel do private (neib,k,kk)
      do neib= 1, NEIBPETOT
        do k= export_index(neib-1)+1, export_index(neib)
          kk= export_item(k)
          SENDbuf(k)= W(kk,P)
        enddo
      enddo

      do neib= 1, NEIBPETOT
        is   = export_index(neib-1) + 1
        len_s= export_index(neib) - export_index(neib-1)
        call MPI_Isend (SENDbuf(is), len_s, MPI_DOUBLE_PRECISION,      &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,               &
     &                  request_send(neib), ierr)
      enddo

      do neib= 1, NEIBPETOT
        ir   = import_index(neib-1) + 1
        len_r= import_index(neib) - import_index(neib-1)
        call MPI_Irecv (W(ir+N,P), len_r, MPI_DOUBLE_PRECISION,        &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,               &
     &                  request_send(neib+NEIBPETOT), ierr)
      enddo

      call MPI_Waitall (2*NEIBPETOT, request_send, stat_send, ierr)
```

Figure 3: Communications among MPI Processes using Communication Table [2][3][4]

3 COMMUNICATION-COMPUTATION OVERLAPPING AND RELATED WORK

3.1 Matrix Power Kernel and Pipelined CG

Figure 4 shows the Preconditioned CG (PCG) algorithm [11], which includes SpMV, dot products, the AXPY ($ax + y$) computations, and preconditioning.

```
Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1,2, …
    solve $[M]z^{(i-1)}= r^{(i-1)}$
    $\rho_{i-1}= r^{(i-1)} z^{(i-1)}$
    if i=1
      $p^{(1)}= z^{(0)}$
     else
      $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
      $p^{(i)}= z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
    endif
    $q^{(i)}= [A]p^{(i)}$
    $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
    $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
    $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
    check convergence |r|
end
```

Figure 4: Preconditioned CG Algorithm for Solving the Ax=b, M: Preconditioning Matrix[11]

In computations on parallel computers with distributed memory, SpMV, dot products, and preconditioning may require communication. Although there are various types of communication patterns in preconditioning, SpMV relies upon point-to-point communication with neighbors and dot products rely upon global collective communication [2][3][4]. If the code is implemented by MPI, `MPI_Isend` and `MPI_Irecv` with `MPI_Wait/Waitall` are used in SpMV, and `MPI_Allreduce` is used in dot products. The overhead for such communications is a critical issue on massively parallel supercomputers with more than 10^4 MPI processes. In recent years, many algorithms and methods to avoid and reduce communication overhead have been proposed.

Methods based on the *matrix powers kernel* [12] reduce the number of global communications at each iteration by extending the halo region in Fig. 2 and Fig. 3. They generally require complicated data structures and are not suitable for general preconditioning methods, such as incomplete LU (ILU).

The *pipelined method* [13] reduces communications overhead by overlapping collective communications and computations. In the pipelined method, the sequence of Krylov iterations is changed using recurrence relations without changing the original algorithm. In pipelined CG [13], collective communication for dot products can be overlapped with heavier computations, such as SpMV and preconditioning. Asynchronous collective communication (e.g., `MPI_Iallreduce`supported in MPI-3 or later is effective for such procedures. In [14], the authors implemented pipelined CG to the original GeoFEM/Cube with ICCG using up to 12,288 cores (384 nodes) of Intel Xeon/Broadwell system [8]. Figure 5 compares the original and pipelined CG method for strong scaling. Pipelined CG provides much better scalability than the original CG. Pipelined CG is effective in the very limited case of strong scaling, where the problem size per MPI process is very small.

3.2 Communication-Computation Overlapping with Dynamic Loop Scheduling

Pipelined CG overlaps collective communications for dot products and computations efficiently, as described in the previous

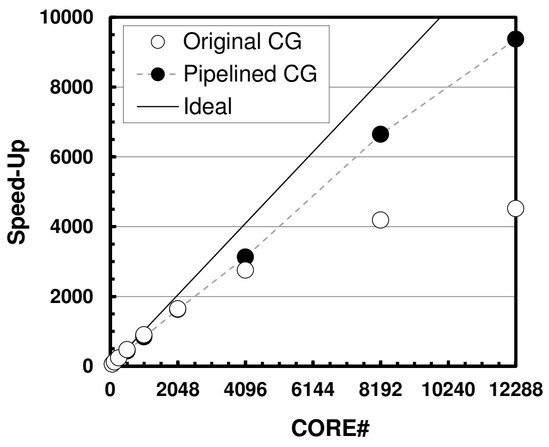


Figure 5: Effects of pipelined CG [13] for GeoFEM/Cube with ICCG using up to 12,288 cores (384 nodes) of Intel Xeon/Broadwell Cluster [8] and Strong Scaling; Entire Problem Size: 28,311,552 DOF

subsection. Another approach for CC-Overlapping is overlapping communications in halo exchanges with computations in SpMV. The overhead for global synchronization by `MPI_Waitall` in halo exchanges may be critical on massively parallel supercomputers. The idea for CC-overlapping was originally applied to stencil-type computations with explicit time-marching [1]. In this subsection, procedures of CC-Overlapping for GeoFEM/Cube [2] are described. In CC-Overlapping for parallel FEM, the internal nodes in Fig. 2 and Fig. 3 are divided into the following two categories:

- Boundary Nodes: Directly adjacent to external nodes Fig. 6(b))
- Pure-Internal Nodes: Not adjacent to external nodes Fig. 6(b))

Because pure-internal nodes are not adjacent to external ones, computations on these can be done without communication with external processes. Therefore, halo exchanges can be overlapped with computations for pure-internal nodes.

First of all, reordering of internal node/meshes is needed, where the numbering starts from the pure internal nodes, and the boundary nodes/meshes are numbered later. Continuous numbering for each of the pure internal and boundary nodes provides efficient memory access (Fig. 6(c)). Finally, CC-Overlapping for halo exchanges can be described as follows (Fig. 7):

(1) Communications: Importing information on external nodes/meshes (halo) from external processes
(2) Computation of pure internal nodes: Overlapped with communications in (1)
(3) Synchronization of communications: `MPI_Waitall`
(4) Computation of internal nodes/meshes on boundaries (boundary nodes/meshes).

Figure 7 describes communication pattern of *original* implementation and that with CC-Overlapping. This type of

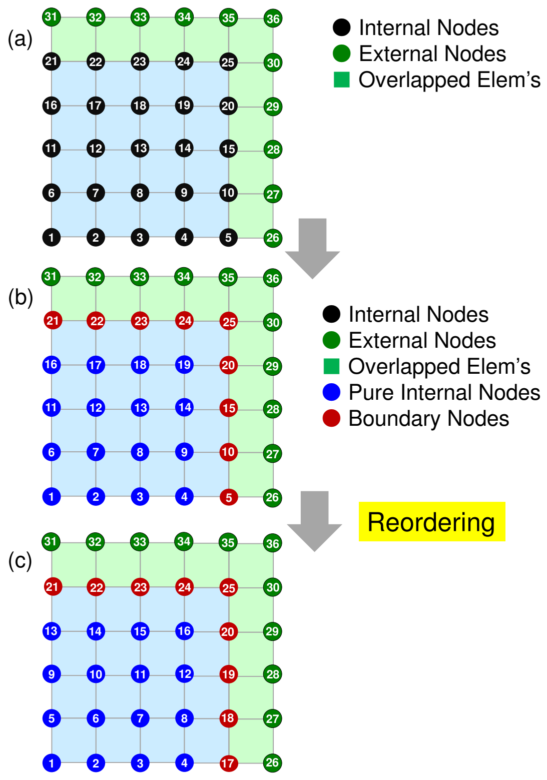
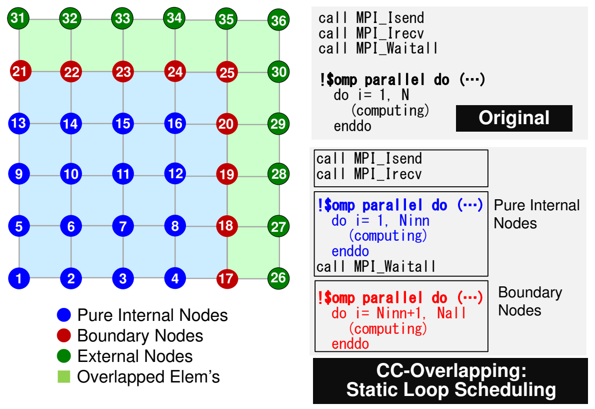


Figure 6: Overlapping Communications in Halo Exchanges with SpMV Computation with Reordering

CC-Overlapping is called *CC-Overlapping with Static Loop Scheduling*. Furthermore, dynamic loop scheduling of OpenMP was introduced for CC-Overlapping in halo exchanges based on the method in [5]. In this approach, halo exchanges, including transfers of buffer copies, are done by the master thread, and dynamic loop scheduling is applied to the computations for pure internal nodes, as shown in Fig. 8. The computations for pure internal nodes (blue letters in Fig. 8) start without the master thread, which performs communication (red letters in Fig. 8). The master thread can join the computations for pure internal nodes after completing the communication. Directives for parallel loops in OpenMP are written in the following way:

- C: `#pragma omp parallel for schedule (kind, [, chunk])`
- Fortran: `!$omp parallel do schedule (kind, [, chunk])`

There are four *kinds* of loop schedules (static, dynamic, guided, and auto) [2], and the optional parameter (chunk) must be a positive integer when specified. The default kind is *static*, and loops are divided into equal chunks, or as equal as possible when the number of loop iterations is not evenly divisible by the number of threads multiplied by the chunk size. By default, the chunk size is loop-count/number-of-threads. If the kind *dynamic* is applied, the internal work queue is used for giving a chunk-sized block of loop iterations to each thread. When operations of a thread have finished, it retrieves the next block of loop iterations from the top of the work queue. The chunk size is 1 by default. Dynamic scheduling requires extra overhead.

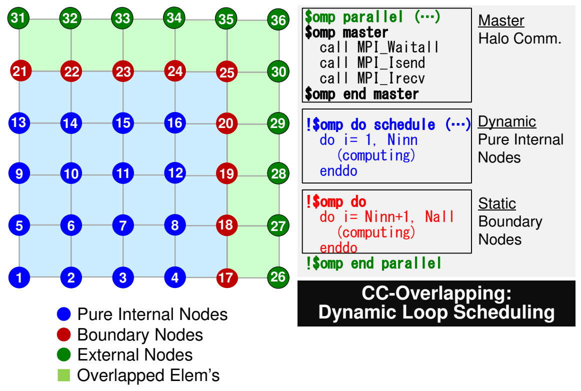


```
call MPI_Isend
call MPI_Irecv
call MPI_Waitall

!$omp parallel do (…)
  do i= 1, N
    (computing)
  enddo
```

```
call MPI_Isend
call MPI_Irecv
!$omp parallel do (…)
  do i= 1, Ninn
    (computing)
  enddo
call MPI_Waitall
!$omp parallel do (…)
  do i= Ninn+1, Nall
    (computing)
  enddo
```

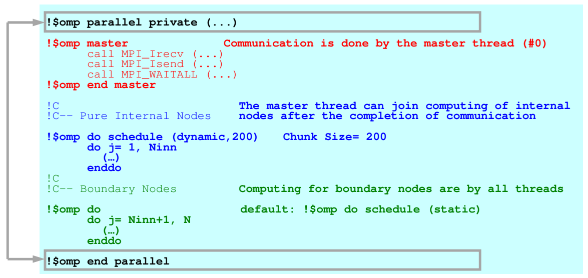
Figure 7: Overlapping communications in halo exchanges with SpMV computation (N_{inn} = number of pure internal nodes/meshes; N_{all} = total number of internal nodes/meshes)



```
$omp parallel (…)
$omp master
  call MPI_Waitall
  call MPI_Isend
  call MPI_Irecv
$omp end master
!$omp do schedule (…)
  do i= 1, Ninn
    (computing)
  enddo
!$omp do
  do i= Ninn+1, Nall
    (computing)
  enddo
!$omp end parallel
```

Figure 8: Overlapping communications in halo exchanges with SpMV computations using dynamic loop scheduling by OpenMP [2]

Figure 9 shows the pseudo code using dynamic loop scheduling of OpenMP with a chunk size of 200. Global communication is done by the master thread between `!$omp master` and `!$omp end master` in Fig. 8. The loop for computations of pure inner nodes/meshes with dynamic scheduling starts without the master thread, which joins the loop operations after completing communication. Smaller chunk sizes may prevent load imbalance among threads, but extra operations related to the internal work are more frequent, which may lead to very significant overhead. CC overlapping with dynamic loop scheduling is expected to hide the communication overhead more efficiently than CC overlapping with static loop scheduling in Fig. 7.



```
!$omp parallel private (...)

!$omp master                  Communication is done by the master thread (#0)
      call MPI_Irecv (...)
      call MPI_Isend (...)
      call MPI_WAITALL (...)
!$omp end master

!C                             The master thread can join computing of internal
!C-- Pure Internal Nodes       nodes after the completion of communication

!$omp do schedule (dynamic,200)   Chunk Size= 200
      do j= 1, Ninn
        (…)
      enddo
!C
!C-- Boundary Nodes            Computing for boundary nodes are by all threads

!$omp do                       default: !$omp do schedule (static)
      do j= Ninn+1, N
        (…)
      enddo

!$omp end parallel
```

Figure 9: Algorithm for overlapping communications in halo exchanges with SpMV computations using dynamic loop scheduling by OpenMP (chunk size= 200) [2]

Table 1: Summary of the computational environment and performance of a single node [6][7][8][9]

System	Oakforest-PACS	Wisteria/BDEC-01 (Odyssey)
Name in this Paper	OFP	ODY
Frequency(GHz)	1.40	2.20
Core # /CPU (socket)	68	48
Peak Performance (GFLOPS)	3,046	3,379
Memory Size (GB)	MCDRAM:16 DDR:96	32
Memory Bandwidth (GB/sec)[16]	MCDRAM:490 DDR4:84	840+

4 ENVIRONMENT FOR COMPUTING

4.1 Hardware

In the present work, Oakforest-PACS (OFP) [7] at JCAHPC [6], and Wisteria/BDEC-01 (Odyssey) (ODY) [9] at Information Technology Center, the University of Tokyo[8] are considered. Table 1 summarizes the performance of a single node of the two systems.

The Oakforest-PACS system (OFP) [7] is the premiere supercomputer system at the Joint Center for Advanced High-Performance Computing [6]. The system consists of 8,208 nodes of Intel Xeon Phi 7250 (Knights Landing, or KNL), and Omni-Path Architecture provides a 100 Gbps interconnection in a fat-tree topology with full bisection bandwidth. Each Xeon Phi 7250 node is built using 68 Silvermont (Atom) cores with 1.4 GHz, and the memory unit consists of 96 GB of DDR4 RAM and 16 GB of stacked 3D MCDRAM, which can be utilized as an L3 cache or high-bandwidth memory. Each core has two 512-bit vector units and supports AVX-512 SIMD instructions. Each core can host four threads and is equipped with 2x512-bit floating-point vector ALU. Therefore, the peak performance of a single node of Xeon Phi 7250 for double precision is 3,046.4 GFLOPS. The total theoretical computational performance is 25 PFLOPS, and the system achieved 13.55 PFLOPS on the HPL benchmark, which ranks it at #39 in the 58th TOP500 list released in November 2021 [15]. The measured performance of the STREAM Triad benchmark [16] for DDR4 RAM of the Oakforest-PACS is 84.5 GB/sec, while that of MCDRAM is 490 GB/sec. In this work, only MCDRAM was used for memory in the flat/quadrant mode on OFP. Moreover, 64 of 68 cores are used on each node with a single thread for each core.

Wisteria/BDEC-01 [9] is a platform for integration of (Simulation+Data+Learning), and started its operation in May 2021. Wisteria/BDEC-01 is a Hierarchical, Hybrid, Heterogeneous (h3) system, and it consists of two types of node groups for computing, Simulation Nodes (Odyssey) and Data/Learning Nodes (Aquarius), Shared File System (25.8 PB) and Fast File System (1.0PB). Total peak performance is 33.1 PFLOPS, and aggregated memory bandwidth is 8.38 PB/sec. The system is constructed by Fujitsu. Simulation nodes for HPC (Odyssey) with more than 25 Peta FLOPS is based on Fujitsu's PRIMEHPC FX 1000 with A64FX with High Bandwidth Memory. Data/Learning nodes (Aquarius) are GPU cluster consisting of Intel Xeon Ice Lake and NVIDIA A100 Tensor Core GPUs, with 7.2 Peta FLOPS for Data Analytics, AI and Machine Learning Workloads. Some of Data/Learning nodes are connected to external resources directly through SINET, Japan. In the present work, up to 2,048 nodes of Odyssey with A64FX have been used. Each node of Odyssey is connected through the Tofu Interconnect D, which is a highly scalable interconnect. Odyssey is ranked #17 in the 58th TOP500. While peak performance of a single node of OFP and Odyssey is similar, memory bandwidth of Odyssey is 1.7 times that of OFP with MCDRAM.

4.2 IHK/McKernel

IHK/McKernel is a lightweight multi-kernel operating system designed for high-end supercomputing [10], and is developed by RIKEN R-CCS. There are two main components of the software stack, a low-level software infrastructure, called Interface for Heterogeneous Kernels (IHK) and a lightweight co-kernel called McKernel. McKernel is a lightweight co-kernel developed on top of IHK. It is designed explicitly for high-performance computing workloads, but it retains a Linux compatible application binary interface (ABI) so that it can execute unmodified Linux binaries. McKernel implements only a small set of performance sensitive system calls and the rest of the OS services are delegated to Linux. Applications are executed on the lightweight kernel, and only performance insensitive operations are offloaded to Linux, therefore OS noise is significantly reduced. In particular, the performance of collective communication by IHK/McKernel is better than that of Linux on OFP. Finally, many applications attained efficient and robust performance on OFP with many nodes [17]. In the previous works [18][19], we examined the impact of the IHK/McKernel to parallel multigrid solvers, and improvement of the performance was 10-20 % at 2,048 nodes of OFP. In the present work, effects of IHK/McKernal were also evaluated for CC-Overlapping on OFP.

5 PRELIMINARY RESULTS OF GEOFEM/CUBE

5.1 Overview

In this section, we evaluated the performance of GeoFEM/Cube with CC-Overlapping [2] on OFP and Odyssey (ODY). Moreover, effects of IHK/McKernel were examined on OFP. We evaluated the performance of GeoFEM/Cube using up to 2,048 nodes of OFP (131,072 cores) and ODY (98,304 cores). Hybrid M × N (HB M × N) parallel programming models were applied, where "M" denotes the number of OpenMP threads for each MPI process, and "N" is the number of MPI processes on each node. In the present work, HB 16×4 for OFP, and HB 12×4 for ODY were evaluated. For each case, the following three implementation were evaluated:

- **Original**: Original code without any CC-Overlapping. Local computation of SpMV starts after completion of a halo exchange.
- **Static**: CC-Overlapping with static loop scheduling (Fig. 7)
- **Dynamic**: CC-Overlapping with dynamic loop scheduling (Fig. 8), The chunk size was vaied from 100 to 1,000

Measurements were repeated five times, and the fastest one of the five measurements are presented as the results for each case. Performance was evaluated at computation time for each iteration of CG solver. Following three types of problem size were considered. The largest problem size at 2,048 nodes is 2.4576×10^{10} DOF:

- Small (S): 96 × 96 × 48 FEM nodes/node (1,327,104 DOF/node)
- Medium (M): 128 × 128 × 64 FEM nodes/node (3,145,728 DOF/node)
- Large (L): 200 × 200 × 100 FEM nodes/node (12,000,000 DOF/node)

5.2 Results

Figures 10(a) and 10(b) show improvement of performance over "original" implementation in Fig. 7 at 2,048 nodes for each of OFP (with IHK/McKernel) and Odyssey (ODY). Generally, performance improvement is 10-15 % at OFP, and 25-30 % at Odyssey. In both of OFP and ODY, this improvement is significant for *Medium* and *Small* cases. Effects of dynamic loop scheduling is rather smaller for *Large* cases, because effects of communication overhead by halo exchange are smaller. This feature is more significant in ODY. Optimum chunk size for dynamic loop scheduling is 700 for Medium and Small, and is 1,000 for Large cases on both systems. Improvement of performance in *static* implementation is rather smaller on OFP, but it is very large on ODY, and this is very significant in Large cases.

Figures 11(a) and 11(b) show the best case of performance improvement over *Original* implementation for each number of nodes of OFP (with IHK/McKernel) and Odyssey (ODY). Generally, results show similar features in Fig. 10(a) and Fig. 10(b), but it is clear that performance improvement is decreasing in Medium and Small cases for OFP, as number of node increases. Similar features are also observed in Fig. 11(b) for ODY, although decreasing is very slight. This is mainly because of overhead by `MPI_Allreduce` for dot products in CG solver.

Figure 12 shows performance improvement by Odyssey (ODY) over OFP with IHK/McKernel. The performance of the best case for each number of nodes was compared. Because the CG solver for sparse coefficient matrices derived from FEM is memory-bound, performance of the memory affects the speed of the application. Ratio of the memory bandwidth between ODY and OFP is 1.70, as shown in Table 1. Performance improvement of ODY over OFP at 128 nodes is 75-80 % for all cases. Generally, improvement for ODY is larger, as number of nodes gets larger, and it is more significant

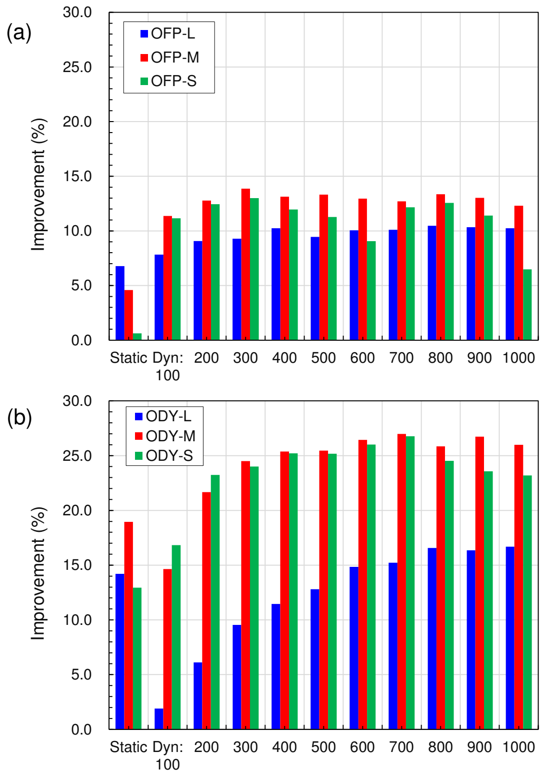


Figure 10: Improvement by Static and Dynamic (Dyn) CC-Overlapping over "Original" implementations in Fig. 7 at 2,048 nodes (a) OFP with IHK/McKernel, (b) Odyssey

for Medium and Small cases. This is because of relationship between number of nodes and improvement of performance, shown in Fig. 11(a) and Fig. 11(b).

Finally, improvement by IHK/McKernel on OFP is 2-5 % for Large and Medium cases, and 7-15 % for Small cases by reduction of communication overhead and noises. These results for improvement are rather smaller than those in the previous works [18][19]. More MPI processes for each node have been applied (e.g. Flat MPI, HB 4x16 etc.) in [18][19], therefore noise and communication overhead were more significant in the previous works [18][19].

Moreover, Figure 13 compares results of Original, Static, and Best Case for Dynamic at 2,048 nodes of ODY. Three bars are relative performance compared to the Original implementation of OFP, and lines/symbols show memory throughput peak ratio (%) measured by Fujitsu's detailed profiler [9]. Both of performance and memory throughput are gradually improving from *Original* through *Static* to *Dynamic*.

6 POISSON3D/DIST

6.1 Overview

Poisson3D/Dist solves 3D Poisson's equation by finite-volume method (FVM). Derived linear equations are solved by preconditioned conjugate gradient (PCG) iterative solvers. The code is written in Fortran 90 using MPI and OpenMP. The target geometry

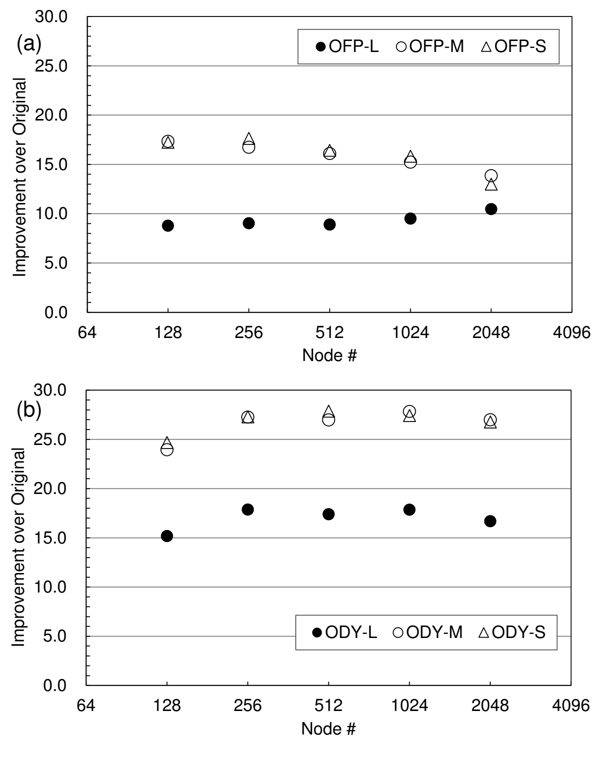


Figure 11: Improvement by Static and Dynamic CC-Overlapping over "Original", best case for each number of nodes (a) OFP with IHK/McKernel, (b) Odyssey

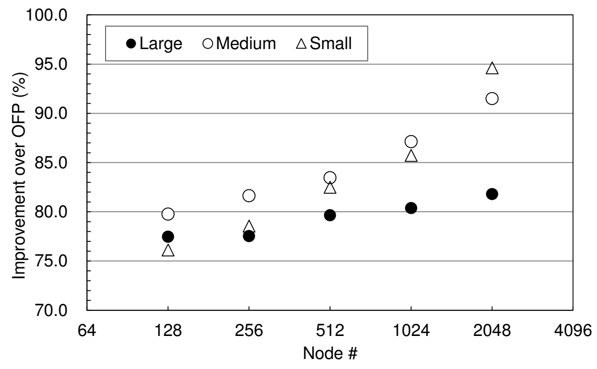


Figure 12: Performance Improvement by Odyssey over OFP (with IHK/McKernel), Best cases for each number of node

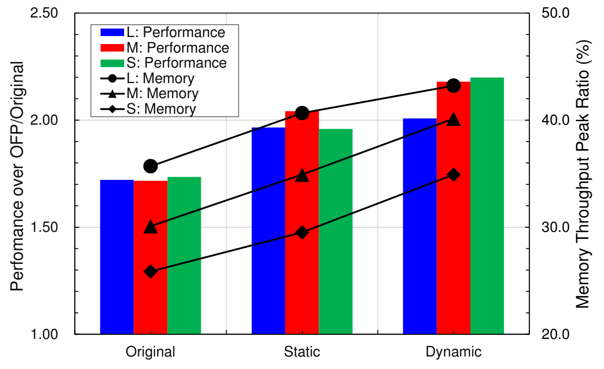


Figure 13: Performance Analyses on Odyssey using Detailed Profiler, Bars: Relative Performance compared to Original implementation of OFP, Lines & Symbols: Memory Throughput Peak Ratio

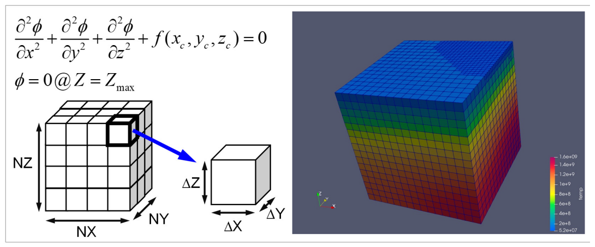

$\frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}+\frac{\partial^2\phi}{\partial z^2}+f(x_c,y_c,z_c)=0$
$\phi=0@Z=Z_{max}$
NZ
NX
NY
ΔZ
ΔX
ΔY

Figure 14: Overview of Poisson3D/Dist

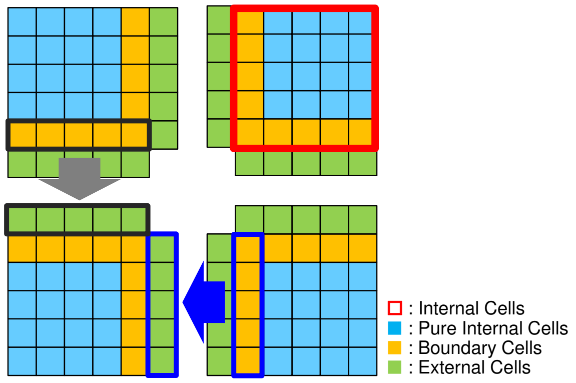


Figure 15: Parallel Data Structure of Poisson3D/Dist [18][19] [20]

is structured uniform meshes (cells) with 7-point stencil, as shown in Fig. 14. Each component of the *right-hand-side* vector of the linear equation is calculated by the location of the cells, as shown in Fig. 14.

Unknowns are defined at the center of each cell, and same data structure for parallel FVM described in [18][19] [20] is applied. Figure 15 shows parallel data structure of Poisson3D/Dist with halo exchange.

Linear equations are solved by the following three types of preconditioned conjugate gradient method (PCG), because the derived matrices are symmetric-positive-definite (SPD):

(1) PJ: PCG with Point-Jacobi (Diagonal Scaling) Preconditioning with CRS format (Compressed Row Storage)
(2) CRS: ICCG (CG with Incomplete Cholesky factorization without fill-in) [11] with CRS format
(3) ELL: ICCG with Sliced ELL (Ellpack-Itpack) format

Point-Jacobi (Diagonal Scaling) Preconditioning adopts diagonal matrix for the preconditioning matrix [M] in Fig. 4, where diagonal components of the original coefficient matrix $[A]$ are extracted, and the k-th components of $r^{(i-1)}$, and $z^{(i-1)}$ are defined as $z_k{}^{(i-1)} = r_k{}^{(i-1)}/D_{kk}$, where D_{kk} is the k-th diagonal component of $[A]$. While this preconditioning is simple and easy to be parallelized, it is not suitable for ill-conditioned problems. In the present work, the target geometry is homogeneous and cell size is uniform, therefore PJ preconditioning provides good convergence. ICCG (CG with Incomplete Cholesky factorization without fill-in) is a widely-used PCG for various types of applications in science and engineering. Because it includes forward/backward substation and factorization with data dependency, reordering is needed for extracting parallelism in multi-threading using OpenMP. In the present work, multicolor reordering with 2-colors [11][18][19] [20] has been applied. Generally, number of iterations increases, as number of MPI processes increase, because IC preconditioning is applied to each MPI process locally. In the present work, Additive Schwartz Domain Decomposition (ASDD, Fig. 16) [21] is applied for stabilizing this localized preconditioning.

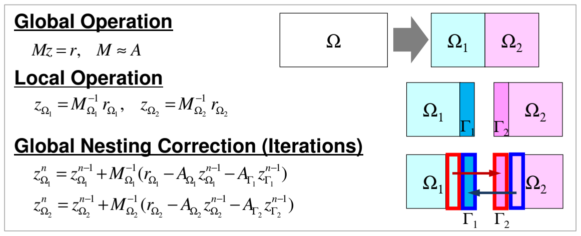

Global Operation
$Mz = r, \quad M \approx A$
Local Operation
$z_{\Omega_1} = M_{\Omega_1}^{-1} r_{\Omega_1}, \quad z_{\Omega_2} = M_{\Omega_2}^{-1} r_{\Omega_2}$
Global Nesting Correction (Iterations)
$z_{\Omega_1}^{n} = z_{\Omega_1}^{n-1} + M_{\Omega_1}^{-1}(r_{\Omega_1} - A_{\Omega_1} z_{\Omega_1}^{n-1} - A_{\Gamma_1} z_{\Gamma_1}^{n-1})$
$z_{\Omega_2}^{n} = z_{\Omega_2}^{n-1} + M_{\Omega_2}^{-1}(r_{\Omega_2} - A_{\Omega_2} z_{\Omega_2}^{n-1} - A_{\Gamma_2} z_{\Gamma_2}^{n-1})$
Ω
Ω_1
Ω_2
Γ_1
Γ_2

Figure 16: Overview of Additive Schwartz Domain Decomposition [21]

In ASDD, effects of external nodes are introduced by *global nesting correction* in Fig. 16. In the present work, a single iteration of the global nesting correction is applied. In general, ICCG has twice as many operations per iteration as PCG with PJ preprocessing. Moreover, cost of ICCG is almost doubled by adding one iteration of ASDD, because one SpMV and one forward/backward substation are added. Therefore, cost of ICCG with a single iteration of ASDD (ASDD-Single) is three to four times that of PCG with PJ. Generally, number of iterations until convergence for ICCG with ASDD-Single is 1/3 of that of PCG with PJ for well-conditioned problems. Therefore, the total computation time for the linear solver is about the same for PCG with PJ and ICCG with one iteration of ASDD.

Mostly, CC-Overlapping has been applied to applications and explicit time marching, and SpMV procedure in iterative solver with a simple preconditioning method, such as PJ. In the present work, we applied CC-Overlapping to iterative solver with more complicated preconditioning, although CC-Overlapping is applied to SpMV.

Generally, GeoFEM/Cube and Poisson3D/Dist are similar applications, more practical and complicated preconditioning methods, such as IC factorization, are applied to Poisson3D/Dist.

6.2 Method for Storage of Sparse Coefficient Matrices

Generally, computations with sparse matrices are memory-bound processes, because of the indirect memory accesses. Various types of storage formats have been proposed. The compressed row storage (CRS) format [11][18][19] [20] is the most popular and widely used because of its flexibility. It stores only non-zero components of sparse matrices, as shown in Fig. 17(a).

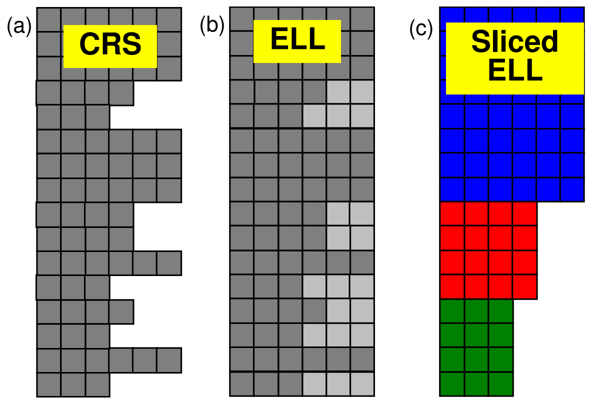


Figure 17: Formats of sparse matrix storage. (a) Compressed row storage (CRS); (b) Ellpack-Itpack (ELL), (c) Sliced ELL [11][18][19] [20]

In the Ellpack-Itpack (ELL) format [18][19] [20], the number of non-zero components of each row are set to that of the longest non-zero entry row of the matrix, as shown in Fig. 17(b). This format allows one to achieve better performance for memory access than CRS, but introduces extra computations and memory requirements, since some rows are zero-padded. In [18][19] [20][22], improved version of ELL (Sliced ELL) was proposed for unstructured/irregular geometries in Fig. 17(c), where a sparse matrix is first divided into sub-matrices (*slices*) consisting of *S* non-zero off-diagonal components (*S=1,…, N*), and each slice is then stored in the ELL format. Thus, the redundancy inherent in the ELL format is eliminated, and cache is well-utilized in loops for pure internal cells. In the present work, CRS is applied to (1) PJ (Point-Jacobi Preconditioning) and (2) CRS (ICCG with CRS), and Sliced ELL is applied to (3) ELL (ICCG with Sliced ELL).

6.3 Reordering Method for CC-Overlapping

Reordering of internal nodes into *pure-internal* and *boundary* ones are essential for efficiency of CC-Overlapping, as shown in Fig. 6. Same procedures in Fig. 6 can be applied to PJ preconditioning base on cell-centered manner, as shown Fig. 18. In CRS and ELL for ICCG, we need to apply different strategy for reordering. Because ICCG includes procedures with global dependency, such as IC factorization and forward/backward substitution, we need to apply reordering for extracting parallelism. In the present work, reordering by multicolor with 2-colors (Red-Black-Coloring (RB-Coloring)) [11][18][19] [20] has been applied.

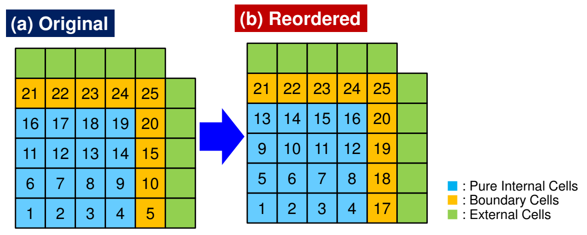


Figure 18: Formats of sparse matrix storage. (a) Compressed row storage (CRS); (b) Ellpack-Itpack (ELL), (c) Sliced ELL [11][18][19] [20][22]

Figure 19(a) shows original ordering by RB-Coloring. If we apply efficient CC-Overlapping, we need to divide *internal cells* into *pure-internal* and *boundary* cells. If we reorder the internal cells from pure-internal ones to boundary ones, as shown in Fig. 19(b), convergence may be affected, because the sequence of computation has been changed. In the coloring in Fig 19(b), we have four colors, 2 colors for pure-internal cells, and 2 color for boundary ones. Table 2 shows number of iterations until convergence, if we apply Poisson3D/Dist to a single MPI process for the geometry with ordering according to Fig. 19(a) and 19(b).

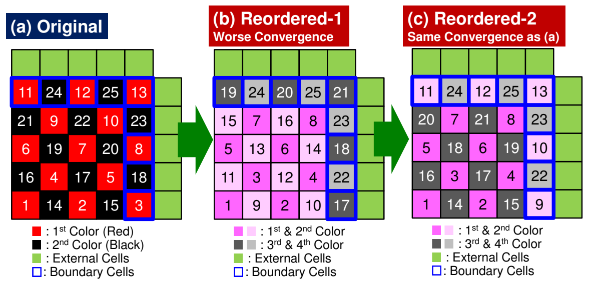
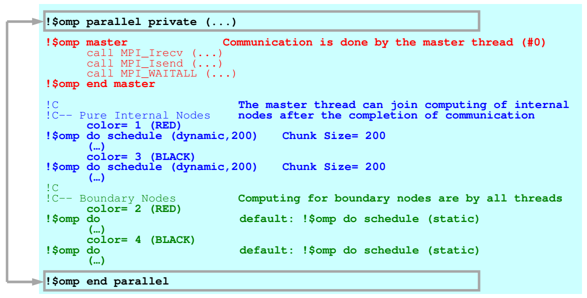


Figure 19: Reordering for CC-Overlapping (CRS, ICCG) by Red-Black-Coloring

Table 2: Effects of reordering method on convergence of ICCG solver

NX,NY,NZ	Cell #	Original: Fig. 19(a)	Reordered-1: Fig. 19(b)
64^3	262,144	225	236
80^3	512,000	279	294
100^3	1,000,000	333	366
128^3	2,097,152	424	467

Number of iterations until convergence increases by 10 % for larger problems, if the reordering in Fig. 19(b) is applied. Therefore, careful reordering is essential for better convergence. The method in Fig. 19(c) keeps same ordering as that of Fig. 19(a) based on original RB-Coloring. In this case, cells in the Red color of Fig. 19(a) are divided into 1st color for pure-internal cells and into 2nd color for boundary cells, while cells in the Black color are into 3rd color (pure-internal cells) and 4th color (boundary cells). Thus, the sequence of computation is same as that of Fig. 19(a), and convergence is not affected. In the present work, reordering in Fig. 19(c) is adopted. Finally, Figure 20 describes the overview of SpMV procedures based on the reordering strategy described in Fig. 19(c). Basically, operations are similar to those of the CC-Overlapping with dynamic loop scheduling in Fig. 9. Operations of the cells the 1st and 3rd colors are done by dynamic scheduling, while those in the 2nd and 4th colors are in static manner.



```
!$omp parallel private (...)
!$omp master                    Communication is done by the master thread (#0)
      call MPI_Irecv (...)
      call MPI_Isend (...)
      call MPI_WAITALL (...)
!$omp end master

!C                                The master thread can join computing of internal
!C-- Pure Internal Nodes          nodes after the completion of communication
      color= 1 (RED)
!$omp do schedule (dynamic,200)    Chunk Size= 200
       (…)
      color= 3 (BLACK)
!$omp do schedule (dynamic,200)    Chunk Size= 200
       (…)
!C
!C-- Boundary Nodes               Computing for boundary nodes are by all threads
      color= 2 (RED)
!$omp do                          default: !$omp do schedule (static)
       (…)
      color= 4 (BLACK)
!$omp do                          default: !$omp do schedule (static)
       (…)
!$omp end parallel
```

Figure 20: Algorithm for Overlapping Communications in Halo Exchanges with SpMV Computations using Dynamic Loop Scheduling by OpenMP for Krylov Iterative Solvers with ILU/IC-type Preconditioning by Red-Black Coloring

7 RESULTS OF POISSON3D/DIST

7.1 Overview

In this section, we evaluated the performance of Poisson3D/Dist with CC-Overlapping on OFP and Odyssey (ODY). Effects of IHK/McKernel were also examined on OFP. We evaluated the performance of Poisson3D/Dist for PJ, CRS and ELL using up to 1,024 nodes of OFP (65,536 cores) with HB 16×4 and ODY (49,152 cores) with HB 12×4. For each case, (1) Original (without CC-Overlapping), (2) Static, and (3) Dynamic (Chunk size between 500 and 3,000) are evaluated. Measurements were repeated five times, and the fastest one of the five measurements is presented as the result for each case. Performance was evaluated for elapsed computation time of the PCG solver. Computation time is normalized by that of PJ solver without CC-Overlapping (Original) using OFP with IHK/McKernel for each number of nodes. Following two types of problem size were considered. The largest problem size at 1,024 nodes is 2.4576×10^{10} DOF:

- Medium (M): $128 \times 128 \times 64$ cells/node (1,048,576 DOF/node)
- Large (L): $160 \times 160 \times 80$ cells/node (2,048,000 DOF/node)

7.2 Results

Figures 21(a) and 21(b) show improvement of performance for each implementation over *Original* one at 1,024 nodes on each of OFP (with IHK/McKernel) and Odyssey (ODY). Generally, performance improvement is 5-10 % at OFP, and 5-25 % at Odyssey. In OFP, results of performance improvement for PJ, CRS and ELL are similar, and improvement is larger for Large (L) cases. On the contrast, results by Odyssey show that performance improvement of PJ is 15-25 %, that of CRS is 10 % and improvement of ELL is 5-12 %. Performance improvement of PJ for Medium cases is significant.

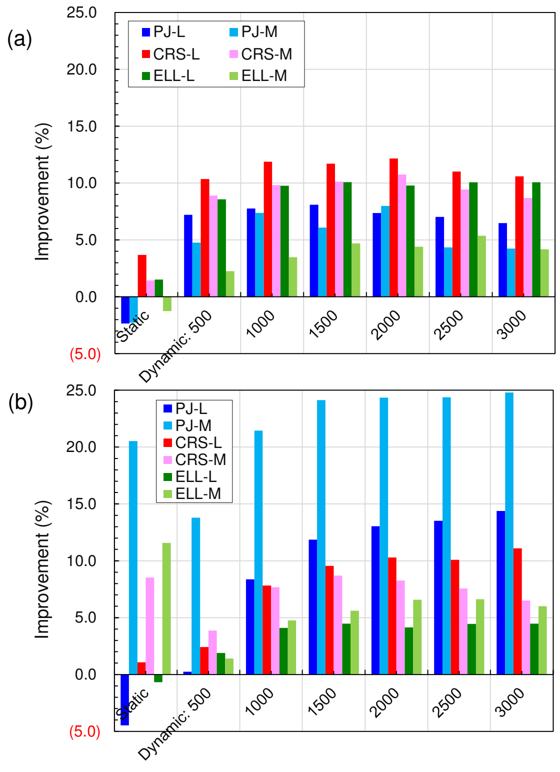


Figure 21: Improvement by Static and Dynamic CC-Overlapping over "Original" implementation at 1,024 nodes (a) OFP with IHK/McKernel, (b) Odyssey

While optimum chunk size for dynamic loop scheduling is 1,500-2,000 for OFP, and it is 3,000 for ODY, *static* loop scheduling provides the best performance for CRS and ELL on ODY with Medium cases, as shown in Fig. 21(b). Figures 22(a) and 22(b) show best case of performance improvement over *Original* implementation for each number of nodes of OFP (with IHK/McKernel) and ODY. Generally, results show similar features in Fig. 21(a) and Fig. 21(b), improvement in ODY is more significant if the number of nodes is more than 384 except ELL-Large. On ODY, difference of the three implementation (PJ, CRS, ELL) is clear. Because algorithm of PJ is very simple, improvement ratio (15-25 %) is very similar to that by GeoFEM/Cube in Fig. 10(b) and Fig. 11(b). Improvement is smaller in CRS, and ELL, because the algorithm is more complicated, and getting higher performance is rather difficult. Cache misses may happen in switching from 1st color to 3rd color, and from 2nd color to 4th color in Fig. 20.

Figures 23(a) and 23(b) show relative performance based on elapsed computation time of PCG solvers. The values at each number of nodes are normalized by the time of PCG solver with PJ by Original implementation on OFP with IHK/McKernel. Number of iterations at 1,024 nodes is 8,086 for PJ-Large, and 2,634 for CRS/ELL-Large, while it is 6,470 for PJ-Medium, and 2,114 for

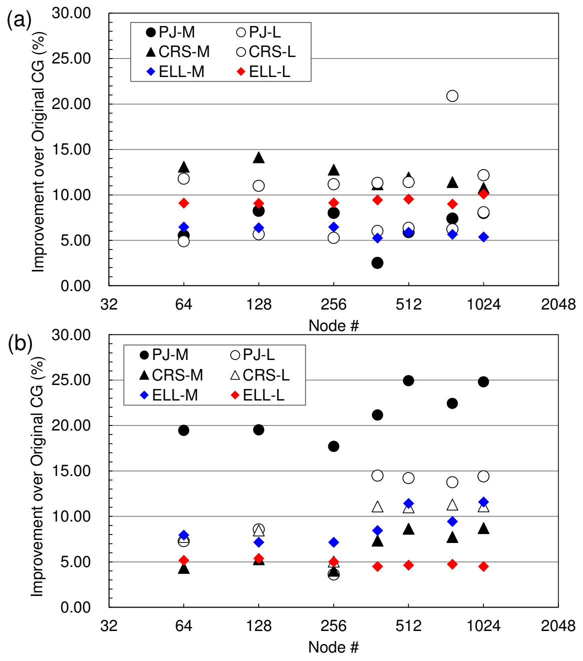


Figure 22: Improvement by Static and Dynamic CC-Overlapping over "Original", best case for each number of nodes (a) OFP with IHK/McKernel, (b) Odyssey

CRS/ELL-Medium. Number of iterations for PJ is as three times as that of CRS/ELL, but PJ and CRS are competitive from the viewpoint of the elapsed computation time for linear solver, as shown in Fig. 23(a) and Fig. 23(b). Performance of ELL is 1.50 times that of CRS on OFP, and more on ODY. Difference between performance for Medium cases and that for Large ones is more significant on ODY. The characteristics of the behavior of Poisson3D/Dist are slightly different from those of GeoFEM/Cube, and further investigation is required.

Figure 24 shows performance improvement by Odyssey (ODY) over OFP with IHK/McKernel. The performance of the best case for each number of nodes was compared. Ratio between ODY and OFP is similar to the results for GeoFEM/Cube in Fig. 12, and ratio of improvement is between 60 % and 125 %. Improvement for ODY over OFP is more significant with more than 384 nodes, because the improvement by CC-Overlapping is more significant on ODY with more than 384 nodes, as shown in Fig. 23(b).

Each of Fig. 25(a) and Fig. 25(b) compares results of Original, Static, and the best case for Dynamic Loop Scheduling (Dynamic) at 1,024 nodes of ODY. Three bars are relative performance compared to the Original implementation with PJ on OFP, and lines/symbols show memory throughput peak ratio (%) measured by Fujitsu's detailed profiler [9]. Features of memory throughput are consistent with those of computational performance. Performance improvement and memory throughput by Sliced ELL is significant, especially for Medium cases, as shown in Fig. 25(a). If CRS and ELL are compared, ratio of memory throughput is proportional to that of performance.

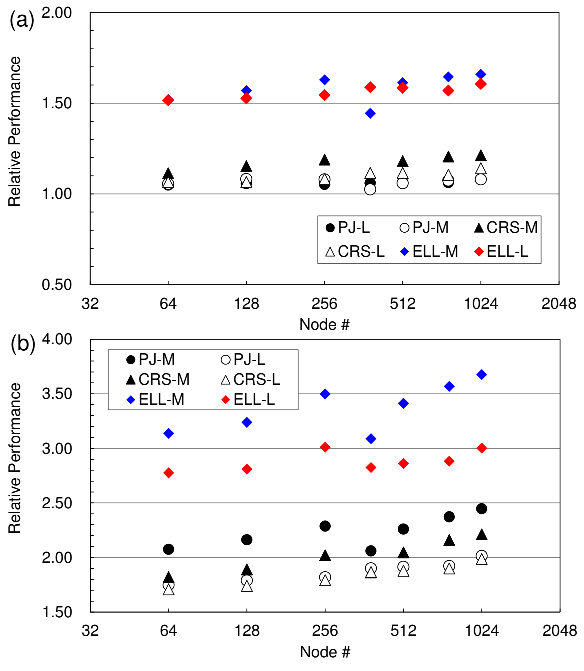


Figure 23: Relative Performance over PJ/Original on OFP with IHK/McKernel, best case for each number of nodes (a) OFP with IHK/McKernel, (b) Odyssey

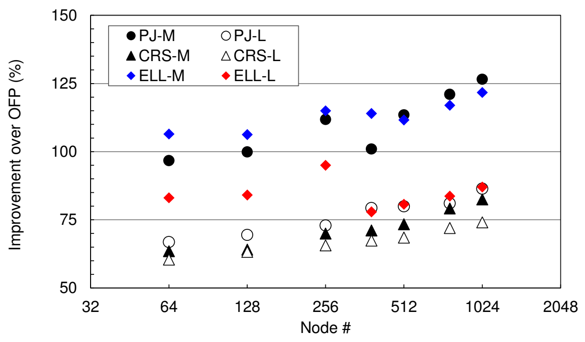


Figure 24: Performance Improvement by Odyssey over OFP (with IHK/McKernel), Best cases for each number of nodes

8 CONCLUDING REMARKS

Preconditioned parallel solvers based on the Krylov iterative method are widely used in scientific and engineering applications. Communication overhead is a critical issue when executing these solvers on large-scale massively parallel supercomputers. In the previous work, we introduced CC-Overlapping with dynamic loop scheduling of OpenMP to the SpMV process of a parallel iterative solver in a parallel finite element application (GeoFEM/Cube) on multicore and manycore clusters. In the present work, first, we re-evaluated the method on our new system, Wisteria/BDEC-01 (Odyssey) with A64FX, as well as on the Oakforest-PACS (OFP) system with Intel Xeon/Phi and, and a significant performance improvement of

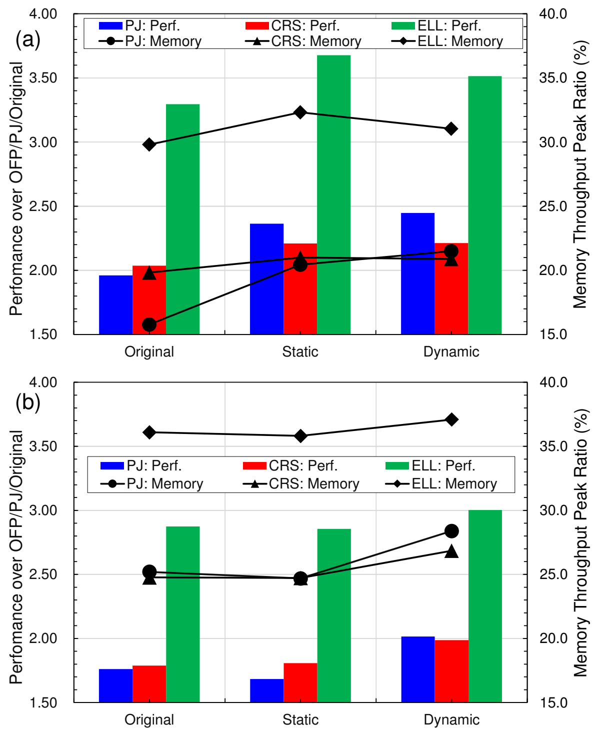


Figure 25: Performance Analyses on Odyssey using Detailed Profiler, Bars: Relative Performance compared to Original implementation of OFP, Lines & Symbols: Memory Throughput Peak Ratio (a)Medium, (b)Large

25-30% for parallel iterative solver at 2,048 nodes was obtained for Odyssey and 10-15% improvement on OFP. Moreover, we proposed a new reordering method for CC-Overlapping in ICCG solvers for a parallel finite volume application (Poisson3D/Dist), evaluated the performance using up to 1,024 nodes of OFP and Odyssey, and attained 5-10% improvement on OFP and 5-12% improvement on Odyssey at 1,024 nodes. Generally, CC-Overlapping with dynamic loop scheduling improved performance of applications with parallel preconditioned iterative solvers, especially on Odyssey with A64FX. Furthermore, effects of IHK/McKernel have been examined for both applications on OFP, and 5-20% improvement of performance has been attained by reduction of communication overhead and noises. Because IHK/McKernel has been already evaluated on Fugaku with A64FX, that will be examined also on Odyssey.

Mostly, CC-Overlapping has been applied to applications and explicit time marching, and SpMV procedure in iterative solver with a simple preconditioning method, such as Point-Jacobi. In the present work, we proposed a new reordering method for CC-Overlapping with dynamic loop scheduling for parallel iterative solvers with ILU/IC-type preconditioning. Although effects of CC-Overlapping were not so significant as those in PCG with Point-Jacobi preconditioning or GeoFEM/Cube, the results are promising and attained more than 10% improvement for rather smaller problems with larger number of MPI processes, where communication overhead is more significant. This type of approach will be useful for strong-scaling cases on highly parallel computations.

We are applying the idea of CC-Overlapping with dynamic loop scheduling to smoothers of parallel multigrid method, and also planning to combine this with pipelined iterative method [13][14] by reduction of communication overhead by `MPI_Allreduce`. In the present work, we just applied reordering with 2-colors (Red-Black-Coloring) for simple geometries. We are also extending this idea to more complicated geometries, where more colors are required for extracting parallelism. If we have more colors, overhead will be increase. Therefore, we need to investigate more efficient ways for reduction of such overhead.

Effects of CC-Overlapping are different on two platforms (OFP and Odyssey), which are evaluated in the present work.
CC-Overlapping with static loop scheduling sometimes better than that with dynamic one on Odyssey. Further investigations on performance analyses are needed.

Currently, performance on Odyssey with A64FX is not necessarily good for both applications, because memory throughput is less than 50% of the peak, as shown in Fig. 13 and Fig. 25. While sliced ELL is very effective, more efficient method for storage of sparse matrices, such as SELL-C-σ [19][23], should be considered for higher performance of applications by FEM and FVM. Furthermore, utilization of lower precision computing, such as FP32 and FP16, on A64FX is another issue for future work.

ACKNOWLEDGMENTS

This work is supported by JSPS Grant-in-Aid for Scientific Research (S) (19H05662), and by Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures (jh210021-NAHI, jh210026-NAH).

REFERENCES

[1] Shimokawabe, T., Aoki, T., Takaki, T., Yamanaka, A., Nukada, A., Endo, T., Maruyama, N., Matsuoka, S., Peta-scale Phase-Field Simulation for Dendritic Solidification on the TSUBAME 2.0 Supercomputer, ACM/IEEE Proceedings of SC'11, 2011

[2] Nakajima, K., Hanawa, T., Communication-Computation Overlapping with Dynamic Loop Scheduling for Preconditioned Parallel Iterative Solvers on Multicore and Manycore Clusters, IEEE Proceedings of ICPPW 2017 (P2S2 2017), 210-219, 2017

[3] Nakajima, K, Okuda, H., Parallel Iterative Solvers for Unstructured Grids using an OpenMP/MPI Hybrid Programming Model for the GeoFEM Platform on SMP Cluster Architectures, Lecture Notes in Computer Science 2327, 437-448, 2002

[4] Nakajima, K., Parallel Iterative Solvers of GeoFEM with Selective Blocking Preconditioning for Nonlinear Contact Problems on the Earth Simulator, ACM/IEEE Proceedings of SC 2003, 2003

[5] Ina, T., Asahi, Y., Idomura, Y., Development of optimization of stencil calculation on Tera-flops many-core architecture, IPSJ SIG Technical Report, Vol.2015-HPC-152, No.10, 2015 (in Japanese)

[6] Joint Center for Advanced High Performance Computing (JCAHPC): `http://jcahpc.jp/`

[7] Oakforest-PACS (OFP): `https://www.cc.u-tokyo.ac.jp/supercomputer/ofp/service/`

[8] Supercomputing Research Division, Information Technology Center, The University of Tokyo (ITC/U.Tokyo): `https://www.cc.u-tokyo.ac.jp/`

[9] Wisteria/BDEC-01: `https://www.cc.u-tokyo.ac.jp/supercomputer/wisteria/service/`

[10] Gerofi, B., Takagi, M., Hori, A., Nakamura, G., Shirasawa, T., Ishikawa, Y., On the Scalability, Performance Isolation and Device Driver Transparency of the IHK/McKernel Hybrid Lightweight Kernel, IEEE Proceedings of IPDPS 2016, 1041-1050, 2016

[11] Saad, Y., Iterative Methods for Sparse Linear Systems (2nd Edition), SIAM, 2003

[12] Demmel, J., Hoemmen, M., Mohiyuddin, M., Yelick, K., Avoiding communication in sparse matrix computations, IEEE Proceedings of IPDPS 2008, 2008

[13] Ghysels, P.„ Vanroose, W., Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm, Parallel Computing 40-7, 224-238, 2014

[14] Hanawa, T., Nakajima, K., Ohshima, S., Hoshino, T., Ida, A., Performance Evaluation of Pipelined CG Method, IPSJ SIG Technical Report, Vol.2016-HPC-157, No.6, 2016 (in Japanese)

[15] Top 500: `https://www.top500.org/`

[16] STREAM Benchmark: `https://www.cs.virginia.edu/stream/`

[17] Gerofi, B, Riesen, R., Takagi, M., Boku, T., Nakajima, K., Ishikawa, Y., Wisniewski, R.W., Performance and Scalability of Lightweight Multi-kernel Based Operating Systems, IEEE Proceedings of IPDPS 2018, 116-125, 2018

[18] Nakajima, K., Gerofi, B., Ishikawa, Y., Horikoshi, M., Parallel Multigrid Methods on Manycore Clusters with IHK/McKernel, IEEE Proceedings of 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA 2019) in conjunction with SC19, Denver, CO, 2019

[19] Nakajima, K., Gerofi, B., Ishikawa, Y., Horikoshi, M., Efficient Parallel Multigrid Method on Intel Xeon Phi Clusters, ACM Proceedings of IXPUG HPC Asia 2021, 2021

[20] Nakajima, K., Optimization of Serial and Parallel Communications for Parallel Geometric Multigrid Method, Proceedings of the 20th IEEE International Conference for Parallel and Distributed Systems (ICPADS 2014) 25-32, Hsin-Chu, Taiwan, 2014

[21] Smith, B., P. Bjorstad, and W. Gropp, Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations, Cambridge Press, 1996

[22] Monakov, A., Lokhmotov, A., Avetisyan, A., Automatically tuning sparse matrix-vector multiplication for GPU architectures, Lecture Notes in Computer Science 5952, 112-125, 2010

[23] Kreutzer, M., Hager, G., Wellein, G., Fehske, H., Bishop, A.R.: A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units, SIAM Journal on Scientific Computing 36-5, C401-C423, 2014