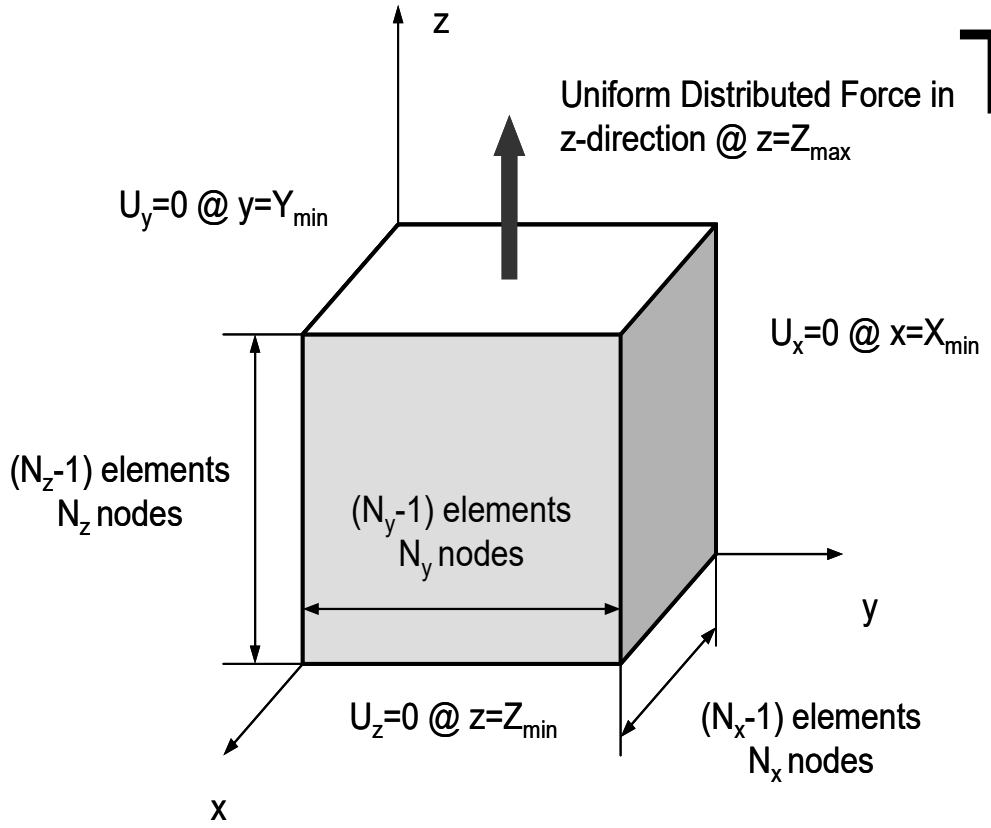
3D Parallel FEM (V) Communication-Computation Overlapping

Kengo Nakajima

Programming for Parallel Computing (616-2057)
Seminar on Advanced Computing (616-4009)

Target Application



- Boundary Conditions
  - Symmetric B.C.
    - $U_X=0@X=0$
    - $U_Y=0@Y=0$
    - $U_Z=0@Z=0$
  - Uniform Distributed Force
    - $F_Z=1@Z=Z_{max}$

- Elastic Material
  - Young's Modulus: $E\ (=1.00)$, Poisson's Ratio: $\nu\ \ (=0.30)$
- Rectangular Priem
  - 1x1x1 cubes (hexahedra)
  - NX, NY, NZ nodes in each direction

Overview of the Program

- 3D Static-Linear-Elastic Problem (Solid Mechanics)
  - 3x3 Block Operation
- CG without Preconditioning
- Hybrid
- Parallel distributed meshes are generated in the program automatically.
  - NO need for separate process of mesh generation, and domain partitioning
- Communication-computation overlapping is introduced to SpMV (matrix vector products) of CG
  - Send_Recv: Amount of message is small, most of the time is spent for “latency”

Files ...

Fortran Only

```
>$ cd <$O-TOP>
>$ cp /home/z30088/class_eps/cc_overlap.tar .

>$ cd cc_overlap
>$ cd src0
>$ make
>$ cd ../src0m
>$ make
>$ cd ../src0m2
>$ make

>$ cd ../run
>$ ls -l sol*
   sol0        No overlapping
   sol0m       Comm.-Comp. Overlapping
   sol0m_xxx   +dynamic scheduling
```

Execution

Fortran Only

```
>$ cd <$O-TOP>/cc_overlap/run

Please modify the following files:

mesh.inp    Problem Configuration
go00.sh     Batch script file for “sol0”
go0m.sh     Batch script file for “sol0m”
go0m2.sh    Batch script file for “sol0m_XXX”
```

“mesh.inp” for Hybrid 16x1

```
(values)   (variables)  (descriptions)

400 400 600 npx,npy,npz   Total number of nodes in X-, Y-,  and Z-
                          direction (Nx, Ny, Nz in the prev. page
  2   2   3 ndx,ndy,ndz   Partition # in each direction (X,Y,Z)
 16   1     PEsmpTOT,(unused) Thread # on each MPI proc., unused
200          ITERmax            Number of iterations for CG method
```

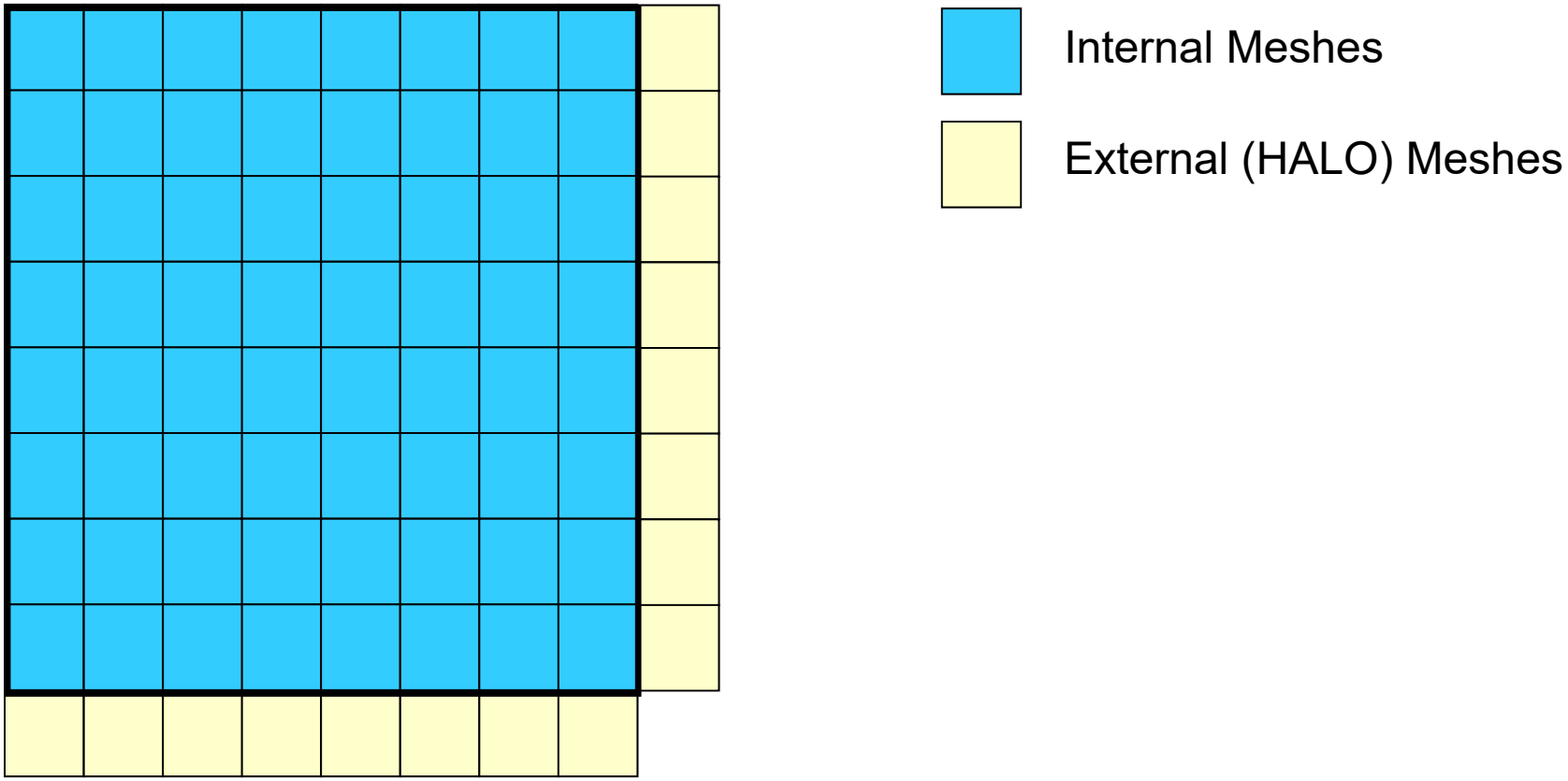
- Each of “npx,npy,npz” must be “divisible（割り切れる）” by each of “ndx,ndy,ndz”
- MPI process # = ndx × ndy × ndz
  - Example: 12 nodes, 192 cores, 12 processes, 16 threads
- “ITERmax” can be small, if you want evaluate the perfomance (it takes a long time until convergence).

Example of “go00.sh”

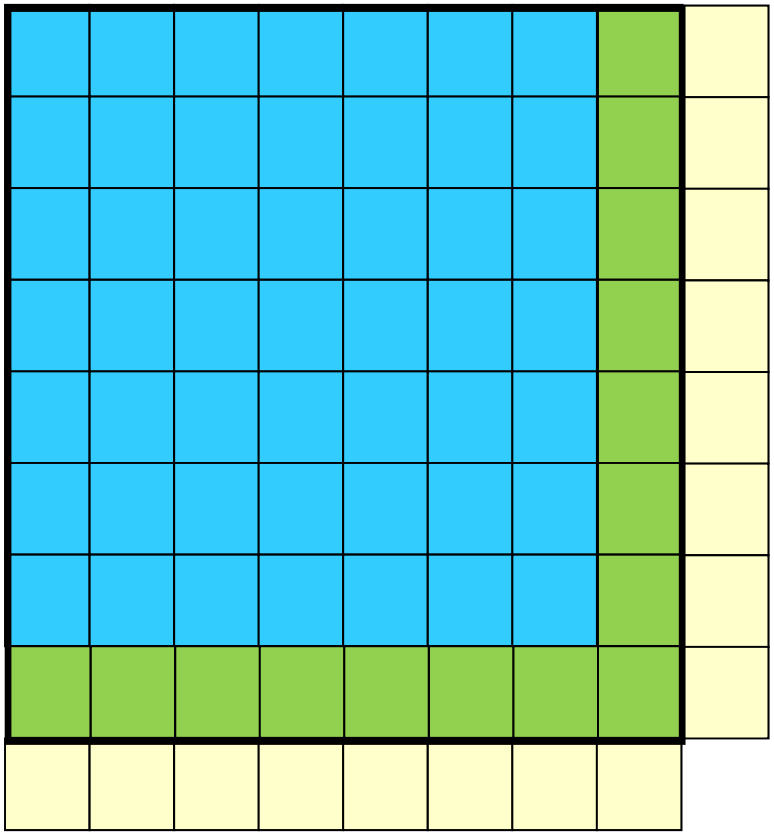
export OMP_NUM_THREADS=PEsmpTOT

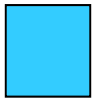
```
#!/bin/sh
#PJM -L "node=12"
#PJM -L "elapse=00:05:00"
#PJM -j
#PJM -L “rscgrp=lecture3“
#PJM -g “gt13“
#PJM -o “test.lst"
#PJM --mpi "proc=12"

export OMP_NUM_THREADS=16
mpiexec ./sol0
rm wk.*
```

Comm.-Comp. Overlapping

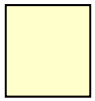


Comm.-Comp. Overlapping





Internal Meshes



External (HALO) Meshes

Internal Meshes on Boundary's

Mat-Vec operations (SpMV)

- Renumbering: ■ ⇒ ■
- Communications of info. on external meshes
- Computation of ■ BEFORE completion of comm. (comm.-comp. overlapping)
- Synchronization of communications
- Computation of ■

q=Ap in src0

```
      call SOLVER_SEND_RECV_3                                          &
     &   ( N, NP, NEIBPETOT, NEIBPE, STACK_IMPORT, NOD_IMPORT,        &
     &     STACK_EXPORT, NOD_EXPORT, WS, WR, WW(1,P) , SOLVER_COMM,   &
     &     my_rank)

!$omp parallel do private (j,k,i,X1,X2,X3,WVAL1,WVAL2,WVAL3)
      do j= 1, N
           X1= WW(3*j-2,P)
           X2= WW(3*j-1,P)
           X3= WW(3*j  ,P)
        WVAL1= D(9*j-8)*X1 + D(9*j-7)*X2 + D(9*j-6)*X3
        WVAL2= D(9*j-5)*X1 + D(9*j-4)*X2 + D(9*j-3)*X3
        WVAL3= D(9*j-2)*X1 + D(9*j-1)*X2 + D(9*j  )*X3
        do k= INL(j-1)+1, INL(j)
           i= IAL(k)
          X1= WW(3*i-2,P)
          X2= WW(3*i-1,P)
          X3= WW(3*i  ,P)
          WVAL1= WVAL1 + AL(9*k-8)*X1 + AL(9*k-7)*X2 + AL(9*k-6)*X3
          WVAL2= WVAL2 + AL(9*k-5)*X1 + AL(9*k-4)*X2 + AL(9*k-3)*X3
          WVAL3= WVAL3 + AL(9*k-2)*X1 + AL(9*k-1)*X2 + AL(9*k  )*X3
        enddo
        do k= INU(j-1)+1, INU(j)
           i= IAU(k)
          X1= WW(3*i-2,P)
          X2= WW(3*i-1,P)
          X3= WW(3*i  ,P)
          WVAL1= WVAL1 + AU(9*k-8)*X1 + AU(9*k-7)*X2 + AU(9*k-6)*X3
          WVAL2= WVAL2 + AU(9*k-5)*X1 + AU(9*k-4)*X2 + AU(9*k-3)*X3
          WVAL3= WVAL3 + AU(9*k-2)*X1 + AU(9*k-1)*X2 + AU(9*k  )*X3
        enddo

        WW(3*j-2,Q)= WVAL1
        WW(3*j-1,Q)= WVAL2
        WW(3*j  ,Q)= WVAL3
      enddo
```

q=Ap in src0m (1/3)

```
      do neib= 1, NEIBPETOT
        istart= STACK_EXPORT(neib-1)
        inum  = STACK_EXPORT(neib  ) - istart
!$omp parallel do private (ii)
        do k= istart+1, istart+inum
              ii   = 3*NOD_EXPORT(k)
           WS(3*k-2)= WW(ii-2,P)
           WS(3*k-1)= WW(ii-1,P)
           WS(3*k  )= WW(ii  ,P)
        enddo
        call MPI_ISEND (WS(3*istart+1), 3*inum,MPI_DOUBLE_PRECISION,    &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD, req1(neib),    &
     &                  ierr)
      enddo

      do neib= 1, NEIBPETOT
        istart= STACK_IMPORT(neib-1)
        inum  = STACK_IMPORT(neib  ) - istart
        call MPI_IRECV (WW(3*(istart+N)+1,P), 3*inum,                   &
     &                  MPI_DOUBLE_PRECISION,                           &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                &
     &                  req1(neib+NEIBPETOT), ierr)
      enddo

!C
!C-- Pure Inner Nodes

!$omp parallel do private (j,k,i,X1,X2,X3,WVAL1,WVAL2,WVAL3)
      do j= 1, Ninn
           X1= WW(3*j-2,P)
           X2= WW(3*j-1,P)
           X3= WW(3*j  ,P)
        WVAL1= D(9*j-8)*X1 + D(9*j-7)*X2 + D(9*j-6)*X3
        WVAL2= D(9*j-5)*X1 + D(9*j-4)*X2 + D(9*j-3)*X3
        WVAL3= D(9*j-2)*X1 + D(9*j-1)*X2 + D(9*j  )*X3
```

q=Ap in src0m (2/3)

```
!C
!C-- Pure Inner Nodes

!$omp parallel do private (j,k,i,X1,X2,X3,WVAL1,WVAL2,WVAL3)
      do j= 1, Ninn
           X1= WW(3*j-2,P)
           X2= WW(3*j-1,P)
           X3= WW(3*j  ,P)
        WVAL1= D(9*j-8)*X1 + D(9*j-7)*X2 + D(9*j-6)*X3
        WVAL2= D(9*j-5)*X1 + D(9*j-4)*X2 + D(9*j-3)*X3
        WVAL3= D(9*j-2)*X1 + D(9*j-1)*X2 + D(9*j  )*X3
        do k= INL(j-1)+1, INL(j)
           i= IAL(k)
          X1= WW(3*i-2,P)
          X2= WW(3*i-1,P)
          X3= WW(3*i  ,P)
          WVAL1= WVAL1 + AL(9*k-8)*X1 + AL(9*k-7)*X2 + AL(9*k-6)*X3
          WVAL2= WVAL2 + AL(9*k-5)*X1 + AL(9*k-4)*X2 + AL(9*k-3)*X3
          WVAL3= WVAL3 + AL(9*k-2)*X1 + AL(9*k-1)*X2 + AL(9*k  )*X3
        enddo
        do k= INU(j-1)+1, INU(j)
           i= IAU(k)
          X1= WW(3*i-2,P)
          X2= WW(3*i-1,P)
          X3= WW(3*i  ,P)
          WVAL1= WVAL1 + AU(9*k-8)*X1 + AU(9*k-7)*X2 + AU(9*k-6)*X3
          WVAL2= WVAL2 + AU(9*k-5)*X1 + AU(9*k-4)*X2 + AU(9*k-3)*X3
          WVAL3= WVAL3 + AU(9*k-2)*X1 + AU(9*k-1)*X2 + AU(9*k  )*X3
        enddo

        WW(3*j-2,Q)= WVAL1
        WW(3*j-1,Q)= WVAL2
        WW(3*j  ,Q)= WVAL3
      enddo

      call MPI_WAITALL (2*NEIBPETOT, req1, sta1, ierr)      ここで同期をとる
```

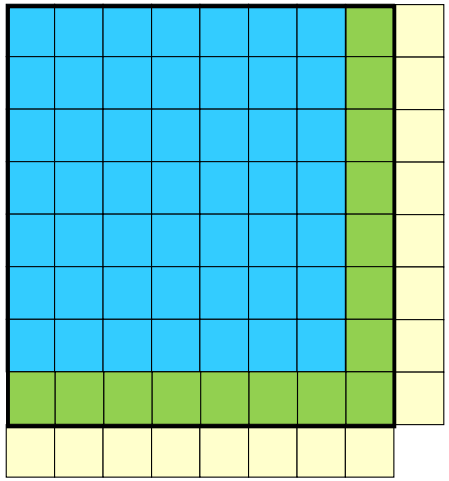
q=Ap in src0m (3/3)

```
!C
!C-- Boundary Nodes

!$omp parallel do private (j,k,i,X1,X2,X3,WVAL1,WVAL2,WVAL3)
      do j= Ninn+1, N
           X1= WW(3*j-2,P)
           X2= WW(3*j-1,P)
           X3= WW(3*j  ,P)
        WVAL1= D(9*j-8)*X1 + D(9*j-7)*X2 + D(9*j-6)*X3
        WVAL2= D(9*j-5)*X1 + D(9*j-4)*X2 + D(9*j-3)*X3
        WVAL3= D(9*j-2)*X1 + D(9*j-1)*X2 + D(9*j  )*X3
        do k= INL(j-1)+1, INL(j)
           i= IAL(k)
          X1= WW(3*i-2,P)
          X2= WW(3*i-1,P)
          X3= WW(3*i  ,P)
          WVAL1= WVAL1 + AL(9*k-8)*X1 + AL(9*k-7)*X2 + AL(9*k-6)*X3
          WVAL2= WVAL2 + AL(9*k-5)*X1 + AL(9*k-4)*X2 + AL(9*k-3)*X3
          WVAL3= WVAL3 + AL(9*k-2)*X1 + AL(9*k-1)*X2 + AL(9*k  )*X3
        enddo
        do k= INU(j-1)+1, INU(j)
           i= IAU(k)
          X1= WW(3*i-2,P)
          X2= WW(3*i-1,P)
          X3= WW(3*i  ,P)
          WVAL1= WVAL1 + AU(9*k-8)*X1 + AU(9*k-7)*X2 + AU(9*k-6)*X3
          WVAL2= WVAL2 + AU(9*k-5)*X1 + AU(9*k-4)*X2 + AU(9*k-3)*X3
          WVAL3= WVAL3 + AU(9*k-2)*X1 + AU(9*k-1)*X2 + AU(9*k  )*X3
        enddo

        WW(3*j-2,Q)= WVAL1
        WW(3*j-1,Q)= WVAL2
        WW(3*j  ,Q)= WVAL3
      enddo
```

Comm.-Comp. Overlapping

With Reordering (current)

```
      call MPI_Isend
      call MPI_Irecv

      do i= 1, Ninn
        (calculations)
      enddo

      call MPI_Waitall

      do i= Ninn+1, Nall
        (calculationas)
      enddo
```



Without Reordering

```
      call MPI_Isend
      call MPI_Irecv

      do i= 1, Nall
        if (INNflag(i).eq.1) then
          (calculations)
        endif
      enddo

      call MPI_Waitall

      do i= 1, Nall
        if (INNflag(i).eq.0) then
          (calculations)
        endif
      enddo
```

OpenMP: Loop Scheduling

```
!$omp parallel do schedule (kind, [chunk])
!$omp do schedule (kind, [chunk])
```

```
#pragma parallel for schedule (kind, [chunk])
#pragma for schedule (kind, [chunk])
```

Kind	Description
static	Divide the loop into equal-sized chunks or as equal as possible in the case where the number of loop iterations is not evenly divisible by the number of threads multiplied by the chunk size. By default, chunk size is loop_count/number_of_threads.Set chunk to 1 to interleave the iterations.
dynamic	Use the internal work queue to give a chunk-sized block of loop iterations to each thread. When a thread is finished, it retrieves the next block of loop iterations from the top of the work queue. By default, the chunk size is 1. Be careful when using this scheduling type because of the extra overhead involved.
guided	Similar to dynamic scheduling, but the chunk size starts off large and decreases to better handle load imbalance between iterations. The optional chunk parameter specifies them minimum size chunk to use. By default the chunk size is approximately loop_count/number_of_threads.
auto	When schedule (auto) is specified, the decision regarding scheduling is delegated to the compiler. The programmer gives the compiler the freedom to choose any possible mapping of iterations to threads in the team.
runtime	Uses the OMP_schedule environment variable to specify which one of the three loop-scheduling types should be used. OMP_SCHEDULE is a string formatted exactly the same as would appear on the parallel construct.

Strategy [Idomura et al. 2014]

- “dynamic”
- “!$omp master～!$omp end master”

```
!$omp parallel private (neib,j,k,i,X1,X2,X3,WVAL1,WVAL2,WVAL3)
!$omp&         private (istart,inum,ii,ierr)

!$omp master              Communication is done by the master thread (#0)
!C
!C– Send & Recv.
(…)
      call MPI_WAITALL (2*NEIBPETOT, req1, sta1, ierr)
!$omp end master

!C                        The master thread can join computing of internal
!C-- Pure Inner 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
```

Idomura, Y. et al., Communication-overlap techniques for improved strong scaling of gyrokinetic Eulerian code beyond 100k cores on the K-computer, Int. J. HPC Appl. 28, 73-86, 2014

Results: Elapsed time for a single iteration of CG (sec.)

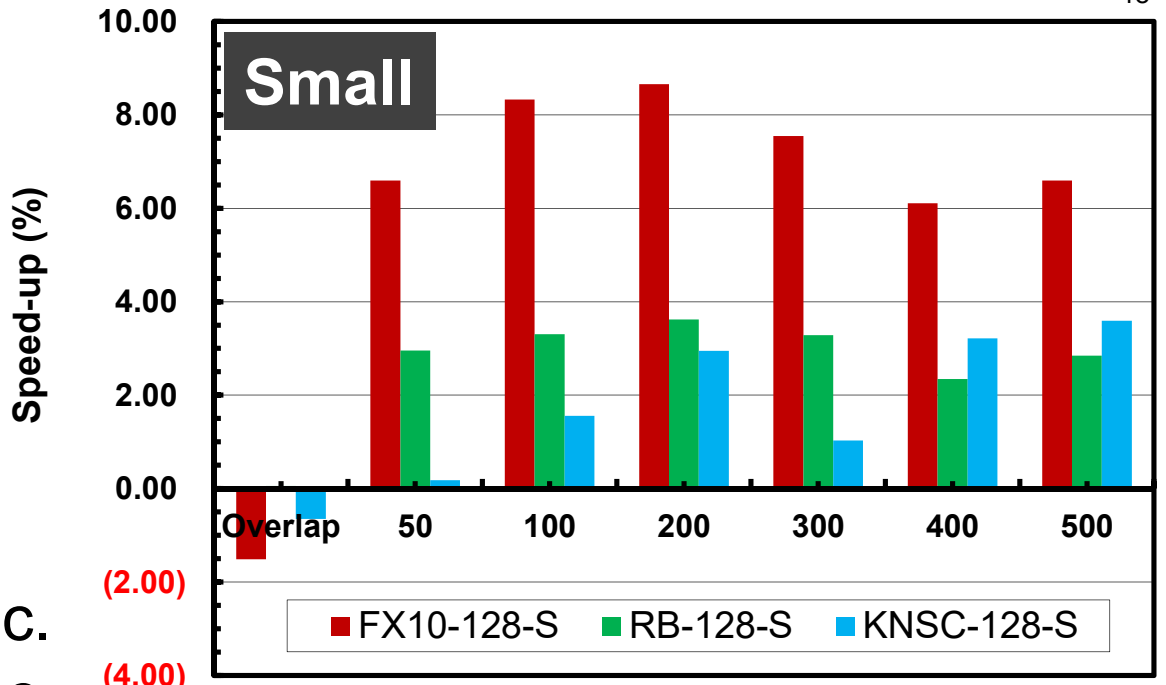
for 12 nodes (processes), 200^3 nodes on each process

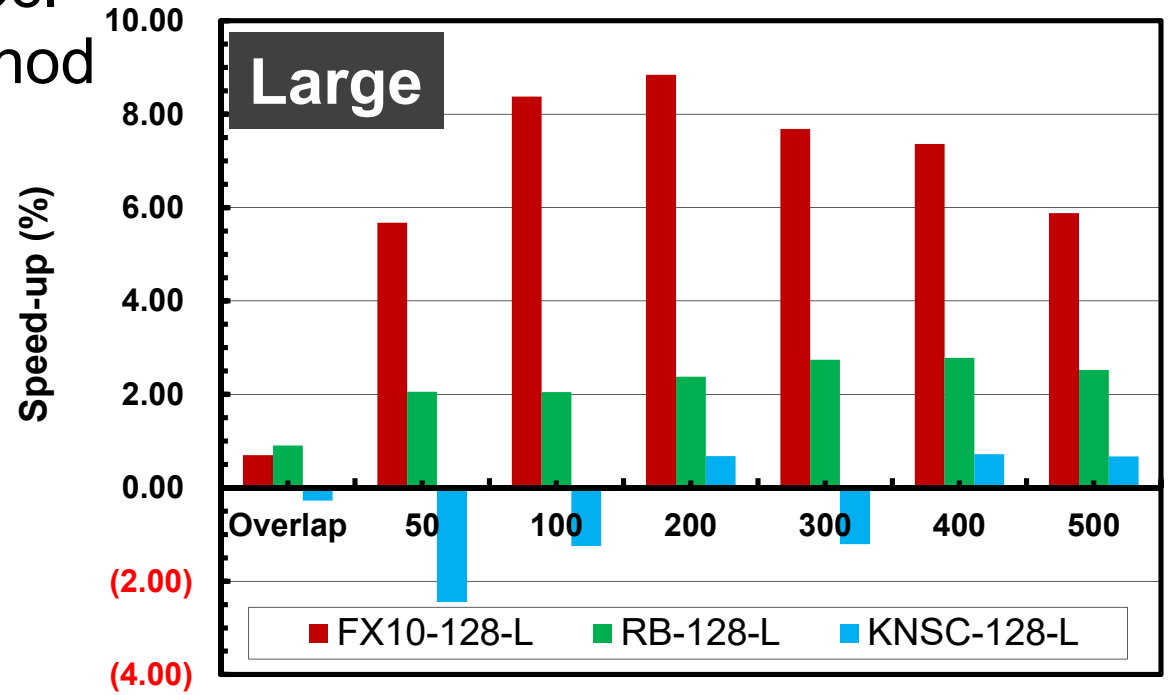
		Chunk Size	sec.
Original (**sol0**)	static	-	0.311
Comm.-Comp. Overlapping（**sol0m**）	static	-	0.313
Comm.-Comp. Overlapping + Loop Scheduling（**sol0m_XXXXX**）	dynamic	50	0.309
		100	0.300
		200	0.297
		300	0.302
		500	0.309
		750	0.310
		1000	0.314
	guided	50	0.323

Block Diagonal CG sec./iteration (1/2)

- Hybrid
- Small : 100^3 nodes/proc.
- Large : 200^3 nodes/proc.
- Overlap: Classical Method
- Number: Chunk Size
- Difference from the Original Method

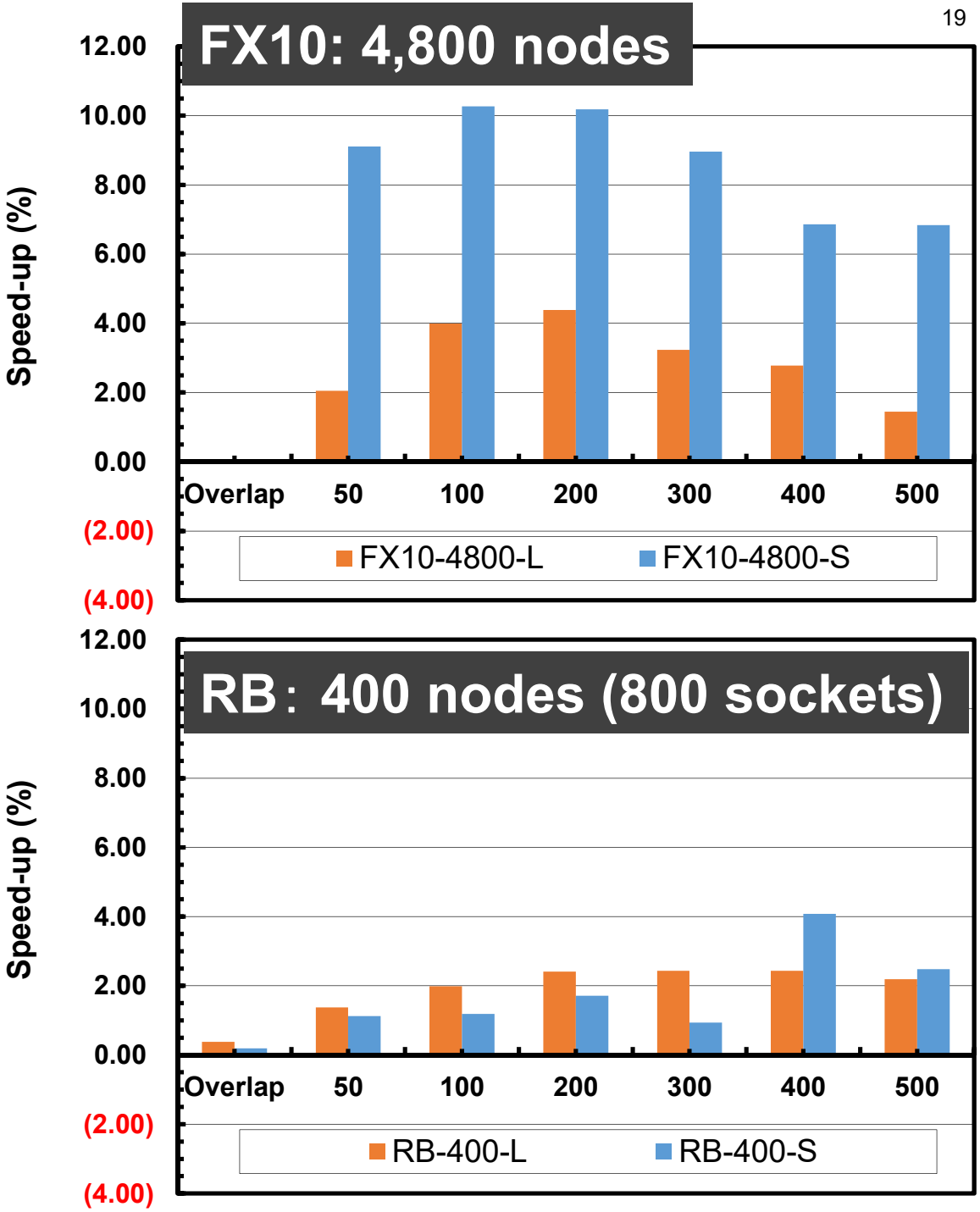
- Oakleaf-FX : FX10
- Reedbush-U : RB
- IVB Cluster : KNSC
- 128 MPI Processes





Block Diagonal CG sec./iteration (2/2)

- No effects by classical overlapping
- Very effective on FX10
  - There is a report describing significant effects of “assist cores for communications” on Fujitsu’s FX100



Summary

- Try other cases
  - Problem Size
    - Avoid 2^n on each MPI process (Bank Conflict)
  - Thread #/MPI Process
  - Chunk Size

- Communication-Computation Overlapping
  - No effects on SpMV without loop scheduling
    - We need certain amount of communications
    - Larger communications mean larger computations
      - Ratio of communication overhead is small …
      - Communication time itself is not so large