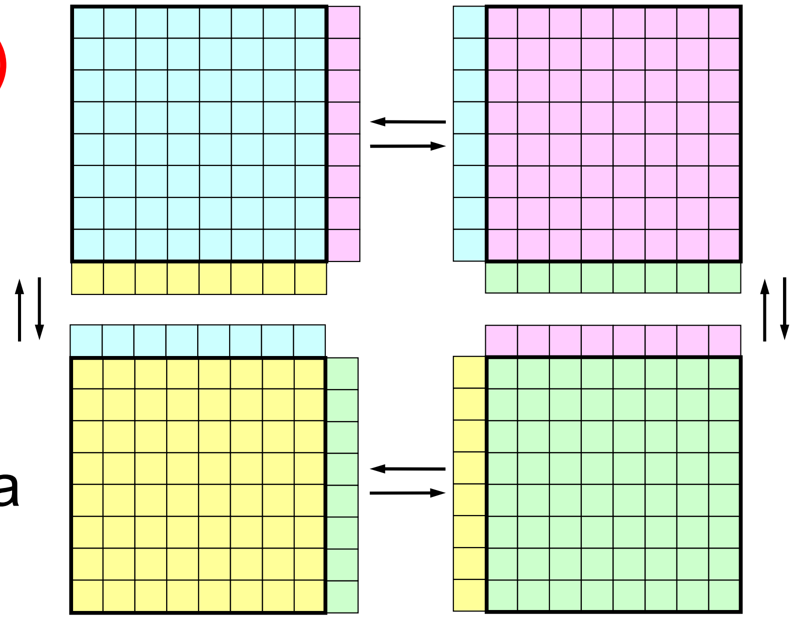
Parallel Iterative Solvers with Preconditioning in the Post-Moore Era

Kengo Nakajima

Information Technology Center, The University of Tokyo

First International Workshop on Deepening Performance Models for Automatic Tuning (DPMAT)
September 7, 2016, Nagoya University

Parallel (Krylov) Iterative Solvers



- Both of convergence (robustness) and efficiency (single/parallel) are important
- Communications needed
  - SpMV (P2P communications, MPI_Isend/Irecv/Waitall): Local Data Structure with HALO
  - Dot-Products (MPI_Allreduce)
    - ✓ effect of latency
  - Preconditioning (up to algorithm)
- Remedy for Robust Parallel ILU Preconditioner
  - Additive Schwartz Domain Decomposition
  - HID (Hierarchical Interface Decomposition, based on global nested dissection) [Henon & Saad 2007], ext. HID [KN 2010]

Assumptions & Expectations towards Post-K/Post-Moore Era

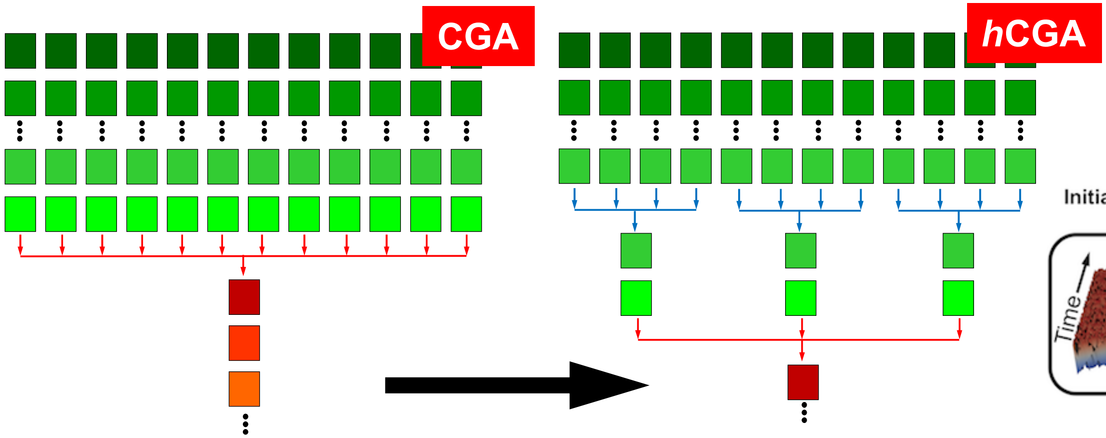
- Post-K (-2020, 2021?)
  - Memory Wall
  - Hierarchical Memory (e.g. KNL: MCDRAM-DDR)
- Post-Moore (-2025? -2029?)
  - High bandwidth in memory and network, Large capacity of memory and cache
  - Large and heterogeneous latency due to deep hierarchy in memory and network
  - Utilization of FPGA
- Common Issues
  - Hierarchy, Latency (Memory, Network etc.)
  - Large Number of Nodes, High concurrency with $O(10^3)$ threads on each node
    - under certain constraints (e.g. power, space …)

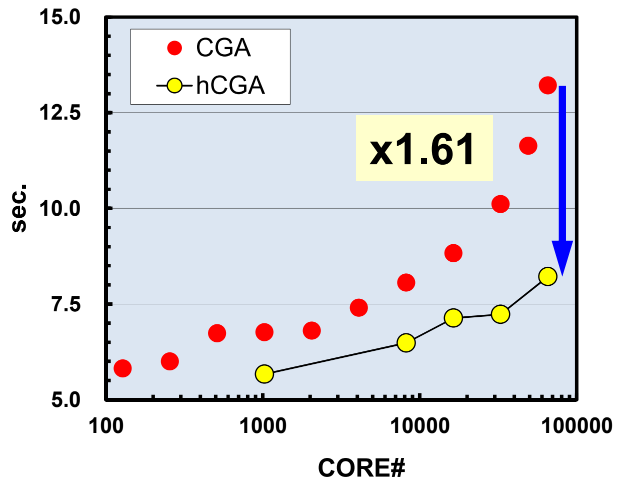
App's & Alg's in Post-Moore Era

- **Compute Intensity -> Data Movement Intensity**
  - **It is very important and helpful for the convergence of BDA and HPC to think about algorithms and applications in the Post Moore Era.**
- Implicit scheme strikes back !: but not straightforward
- **Hierarchical Methods for Hiding Latency**
  - **Hierarchical Coarse Grid Aggregation (hCGA) in MG**
  - **Parallel in Space/Time (PiST)**
- **Comm./Synch. Avoiding/Reducing Algorithms**
  - **Network latency is already a big bottleneck for parallel sparse linear solvers (SpMV, Dot Products)**
  - **Matrix Powers Kernel, Pipelined/Asynchronous CG**
- Power-aware Methods
  - Approximate Computing, Power Management, FPGA

Hierarchical Methods for Hiding Latency

*h*CGA in Parallel Multigrid

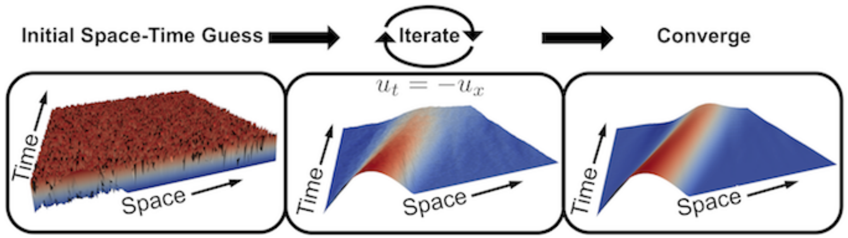




Groundwater Flow Simulation with up to 4,096 nodes on Fujitsu FX10 (GMG-CG) up to 17,179,869,184 meshes (64^3 meshes/core) [KN ICPADS 2014]

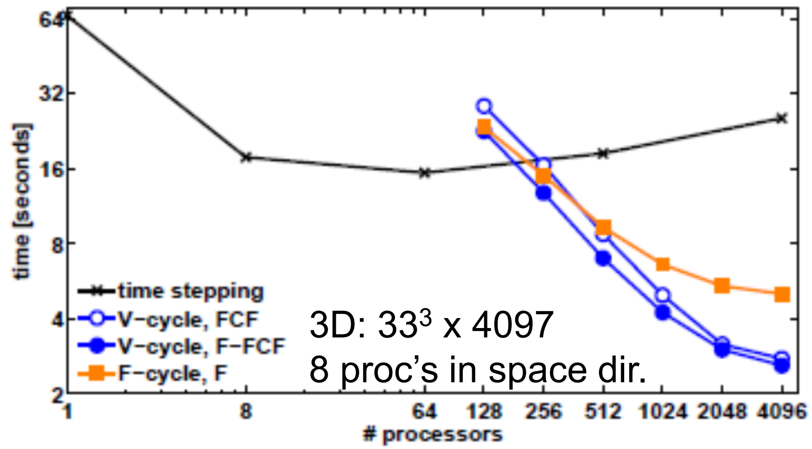
Parallel in Space/Time (PiST)

PiST approach is suitable for the Post-Moore Systems with a complex and deeply hierarchical network that causes large latency.



Comparison between PiST and "Time Stepping" for Transient Poisson Equations

Effective if processor# is VERY large



[R.D.Falgout et al. SIAM/SISC 2014]

App's & Alg's in Post-Moore Era

- Compute Intensity -> Data Movement Intensity
  - It is very important and helpful for the convergence of BDA and HPC to think about algorithms and applications in the Post Moore Era.
- Implicit scheme strikes back !: but not straightforward
- Hierarchical Methods for Hiding Latency
  - Hierarchical Coarse Grid Aggregation (hCGA) in MG
  - Parallel in Space/Time (PiST)
- **Comm./Synch. Avoiding/Reducing Algorithms**
  - **Network latency is already a big bottleneck for parallel sparse linear solvers (SpMV, Dot Products)**
  - **Matrix Powers Kernel, Pipelined/Asynchronous CG**
- Power-aware Methods
  - Approximate Computing, Power Management, FPGA

- Communication/Synchronization Avoiding/Reducing in Krylov Iterative Solvers
- Pipelined CG: Background
- Pipelined CG: Results
- Communication-Computation Overlapping
- Summary

Communication/Synchronization Avoiding/Reducing/Hiding

for Parallel Preconditioned Krylov Iterative Methods

- SpMV
  - Overlapping of Computations & Communications
  - Matrix Powers Kernel
- Dot Products
  - Pipelined Methods
  - Gropp's Algorith,

Algorithm 1 Preconditioned CG

1: $r_0 := b - Ax_0;\quad u_0 := M^{-1}r_0;\quad p_0 := u_0$
2: **for** $i = 0, \ldots$ **do**
3: $\quad s := Ap_i$
4: $\quad \alpha := (r_i, u_i) \,/\, (s, p_i)$
5: $\quad x_{i+1} := x_i + \alpha p_i$
6: $\quad r_{i+1} := r_i - \alpha s$
7: $\quad u_{i+1} := M^{-1} r_{i+1}$
8: $\quad \beta := (r_{i+1}, u_{i+1}) \,/\, (r_i, u_i)$
9: $\quad p_{i+1} := u_{i+1} + \beta p_i$
10: **end for**

Communication Avoiding/Reducing Algorithms for Sparse Linear Solvers utilizing Matrix Powers Kernel

- Matrix Powers Kernel: Ax, A^2x, A^3x ...
- Krylov Iterative Method without Preconditioning
  - Demmel, Hoemmen, Mohiyuddin etc. (UC Berkeley)
- *s-step* method
  - Just one P2P communication for each Mat-Vec during *s* iterations. Convergence may become unstable for large *s*.
- Communication Avoiding ILU0 (CA-ILU0) [Moufawad & Grigori, 2013]
  - First attempt to CA preconditioning
  - Nested dissection reordering for limited geometries (2D FDM)
- Generally, it is difficult to apply Matrix Powers Kernel to preconditioned iterative solvers

- Communication/Synchronization Avoiding/Reducing in Krylov Iterative Solvers
- **Pipelined CG: Background**
- Pipelined CG: Results
- Communication-Computation Overlapping
- Summary

Hiding Overhead by Collective Comm. in Krylov Iterative Solvers

- Dot Products in Krylov Iterative Solvers
  - MPI_Allreduce: Collective Communications
  - Large overhead with many nodes
- Pipelined CG [Ghysels et al. 2014]
  - Utilization of asynchronous collective communications (e.g. MPI_Iallreduce) supported in MPI-3 for hiding such overhead.
  - Algorithm is kept, but order of computations is changed
  - [Reference] P. Ghysels et al., Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm, Parallel Computing 40, 2014
  - When I visited LBNL in September 2013, Dr. Ghysels asked me to evaluate his idea in my parallel multigrid solvers

4 Algorithms [Ghysels et al. 2014]

- Alg.1 Original Preconditioned CG
- Alg.2 Chronopoulos/Gear
  - 2 dot products are combined in a single reduction
- Alg.3 Pipelined CG (MPI_Iallreduce)
- Alg.4 Gropp's asynchronous CG (MPI_Iallreduce)

- Algorithm itself is not different from the original one
  - Recurrence Relations: 漸化式
  - Order of computation changed -> Rounding errors are propagated differently
    - Convergence may be affected (not happened in my case)
    - update of r= b-Ax needed at every 50 iterations (original paper)

Original Preconditioned CG (Alg.1)

Original Preconditioned CG (Alg.1)

$r_0 := b - Ax_0; \quad u_0 := M^{-1} r_0; \quad p_0 := u_0$
for $i = 0, \ldots$ **do**
$\quad s := Ap_i$
$\quad \alpha := (r_i, u_i) \,/\, (s, p_i)$
$\quad x_{i+1} := x_i + \alpha p_i$
$\quad r_{i+1} := r_i - \alpha s$
$\quad u_{i+1} := M^{-1} r_{i+1}$
$\quad \beta := (r_{i+1}, u_{i+1}) \,/\, (r_i, u_i)$
$\quad p_{i+1} := u_{i+1} + \beta p_i$
end for

$$s_i = Ap_i$$
$$x_{i+1} = x_i + \alpha_i p_i$$
$$r_{i+1} = b - Ax_{i+1} = b - Ax_i - \alpha_i Ap_i$$
$$= r_i - \alpha_i Ap_i = r_i - \alpha_i s_i$$

Chronopoulos/Gear CG (Alg.2)

2 dot products are combined into a single reduction

Chronopoulos/Gear CG (Alg.2)

$r_0 := b - Ax_0; \quad u_0 := M^{-1}r_0; \quad w_0 := Au_0$

$\alpha_0 := (r_0, u_0) / (w_0, u_0); \quad \beta_0 := 0; \quad \gamma_0 := (r_0, u_0)$

for $i = 0, \ldots$ **do**

$\quad p_i := u_i + \beta_i p_{i-1}$

$\quad s_i := w_i + \beta_i s_{i-1}$

$\quad x_{i+1} := x_i + \alpha_i p_i$

$\quad r_{i+1} := r_i - \alpha_i s_i$

$\quad u_{i+1} := M^{-1} r_{i+1}$

$\quad w_{i+1} := Au_{i+1}$

$\quad \gamma_{i+1} := (r_{i+1}, u_{i+1})$

$\quad \delta := (w_{i+1}, u_{i+1})$

$\quad \beta_{i+1} := \gamma_{i+1} / \gamma_i$

$\quad \alpha_{i+1} := \gamma_{i+1} / (\delta - \beta_{i+1}\gamma_{i+1}/\alpha_i)$

end for

$$s_i = Au_i + \beta_i s_{i-1},\ p_i = u_i + \beta_i p_i \Leftrightarrow s_i = Ap_i$$

$$x_{i+1} = x_i + \alpha_i p_i$$

$$r_{i+1} = b - Ax_{i+1} = b - Ax_i - \alpha_i Ap_i$$

$$= r_i - \alpha_i Ap_i = r_i - \alpha_i s_i$$

- 2 dot products combined

- $s_i = Ap_i$ is not computed explicitly: by recurrence

Pipelined Chronopoulos/Gear (No Preconditioning)

Pipelined Chronopoulos/Gear (No Precond.)

```
r_0 := b − A x_0;   w_0 := A r_0
for i = 0, ... do
  γ_i := (r_i, r_i)
  δ := (w_i, r_i)
  q_i := A w_i
  if i > 0 then
    β_i := γ_i/γ_{i−1};   α_i := γ_i/(δ − β_i γ_i/α_{i−1})
  else
    β_i := 0;   α_i := γ_i/δ
  end if
  z_i := q_i + β_i z_{i−1}
  s_i := w_i + β_i s_{i−1}
  p_i := r_i + β_i p_{i−1}
  x_{i+1} := x_i + α_i p_i
  r_{i+1} := r_i − α_i s_i
  w_{i+1} := w_i − α_i z_i
end for
```

- Global synchronization of dot products are overlapped with SpMV

$$u_i = r_i,\ w_i = Au_i = Ar_i$$

$$Ar_{i+1} = Ar_i - \alpha_i As_i \Rightarrow w_{i+1} = w_i - \alpha_i As_i$$

$$As_i = Aw_i + \beta_i As_{i-1} \Rightarrow z_i \left(= As_i = A^2 p_i\right) = Aw_i + \beta_i z_{i-1}$$

$$q_i = Aw_i = A^2 r_i$$

$$= r_i - \alpha_i Ap_i = r_i - \alpha_i s_i$$

Preconditioned Pipelined CG (Alg.3)

Preconditioned Pipelined CG (Alg.3)

$r_0 := b - Ax_0; \quad u_0 := M^{-1}r_0; \quad w_0 := Au_0$
for $i = 0, \dots$ **do**
  $\gamma_i := (r_i, u_i)$
  $\delta := (w_i, u_i)$
  $m_i := M^{-1}w_i$
  $n_i := Am_i$
  if $i > 0$ **then**
    $\beta_i := \gamma_i / \gamma_{i-1}; \quad \alpha_i := \gamma_i / (\delta - \beta_i \gamma_i / \alpha_{i-1})$
  else
    $\beta_i := 0; \quad \alpha_i := \gamma_i / \delta$
  end if
  $z_i := n_i + \beta_i z_{i-1}$
  $q_i := m_i + \beta_i q_{i-1}$
  $s_i := w_i + \beta_i s_{i-1}$
  $p_i := u_i + \beta_i p_{i-1}$
  $x_{i+1} := x_i + \alpha_i p_i$
  $r_{i+1} := r_i - \alpha_i s_i$
  $u_{i+1} := u_i - \alpha_i q_i$
  $w_{i+1} := w_i - \alpha_i z_i$
end for

- Global synchronization of dot products are overlapped with SpMV and Preconditioning

$$\gamma_i = (u_i, u_i)_M = (Mu_i, u_i) = (r_i, u_i)$$

$$\delta_i = (M^{-1}Au_i, u_i)_M = (Au_i, u_i) = (w_i, u_i)$$

$$M^{-1}r_{i+1} = M^{-1}r_i - \alpha_i M^{-1}s_i \Rightarrow u_{i+1} = u_i - \alpha_i q_i \quad (q_i = M^{-1}s_i)$$

$$M^{-1}s_{i+1} = M^{-1}w_i + \beta_i M^{-1}s_i \Rightarrow q_{i+1} = M^{-1}w_i + \beta_i q_i$$

$$Au_{i+1} = Au_i - \alpha_i Aq_i \Rightarrow w_{i+1} = w_i - \alpha_i Aq_i$$

$$Aq_i = AM^{-1}w_i + \beta_i Aq_{i-1} \Rightarrow z_i = Am_i + \beta_i z_{i-1}$$

$$(m_i = M^{-1}w_i = M^{-1}Au_i = M^{-1}AM^{-1}r_i,\ z_i = Aq_i)$$

Gropp's Asynchronous CG (Alg.4) Smaller Computations than Alg.3

<u>Gropp's Asynchronous CG (Alg.4)</u>

$r_0 := b - Ax_0;\quad u_0 := M^{-1}r_0;\quad p_0 := u_0;\quad s_0 := Ap_0;\quad \gamma_0 := (r_0, u_0)$

for $i = 0, \ldots$ **do**

$\quad \delta := (p_i, s_i)$

$\quad q_i := M^{-1}s_i$

$\quad \alpha_i := \gamma_i / \delta$

$\quad x_{i+1} := x_i + \alpha_i p_i$

$\quad r_{i+1} := r_i - \alpha_i s_i$

$\quad u_{i+1} := u_i - \alpha_i q_i$

$\quad \gamma_{i+1} := (r_{i+1}, u_{i+1})$

$\quad w_{i+1} := Au_{i+1}$

$\quad \beta_{i+1} := \gamma_{i+1} / \gamma_i$

$\quad p_{i+1} := u_{i+1} + \beta_{i+1} p_i$

$\quad s_{i+1} := w_{i+1} + \beta_{i+1} s_i$

end for

- Definition of δ is different from that of Alg.3
- Global synchronization of dot products are overlapped with SpMV and Preconditioning

W. Gropp, Update on Libraries for Blue Waters.
http://jointlab-pc.ncsa.illinois.edu/events/workshop3/pdf/presentations/Gropp-Update-on-Libraries.pdf
Presentation Material (not a paper, article)

Implementation (Alg.4)

```
!C
!C +---------------+
!C | Delta= (p,s)  |
!C +---------------+
!C===
      DL0= 0.d0
!$omp parallel do private(i) reduction(+:DL0)
      do i= 1, 3*N
        DL0= DL0 + P(i)*S(i)
      enddo
      call MPI_Iallreduce (DL0, Delta, 1, MPI_DOUBLE_PRECISION,            &
     &                     MPI_SUM, MPI_COMM_WORLD, req1, ierr)
!C===

!C
!C +-----------------+
!C | {q}= [Minv]{s}  |
!C +-----------------+
!C===

（前処理：省略）

!C===

      call MPI_Wait (req1, sta1, ierr)
```

Gropp's Asynchronous CG (Alg.4)

$r_0 := b - Ax_0;\quad u_0 := M^{-1}r_0;\quad p_0 := u_0;\quad s_0 := Ap_0;\quad \gamma_0 := (r_0, u_0)$

for $i = 0, \ldots$ **do**

$\delta := (p_i, s_i)$

$q_i := M^{-1}s_i$

$\alpha_i := \gamma_i / \delta$

$x_{i+1} := x_i + \alpha_i p_i$

$r_{i+1} := r_i - \alpha_i s_i$

$u_{i+1} := u_i - \alpha_i q_i$

$\gamma_{i+1} := (r_{i+1}, u_{i+1})$

$w_{i+1} := Au_{i+1}$

$\beta_{i+1} := \gamma_{i+1} / \gamma_i$

$p_{i+1} := u_{i+1} + \beta_{i+1} p_i$

$s_{i+1} := w_{i+1} + \beta_{i+1} s_i$

end for

Pipelined CR (Conjugate Residuals)

<u>Preconditioned Pipelined CR</u>

$r_0 := b - Ax_0; \quad u_0 := M^{-1}r_0; \quad w_0 := Au_0$
for $i = 0, \ldots$ **do**
  $m_i := M^{-1}w_i$
  $\gamma_i := (w_i, u_i)$
  $\delta := (m_i, w_i)$
  $n_i := Am_i$
  if $i > 0$ **then**
    $\beta_i := \gamma_i / \gamma_{i-1}; \quad \alpha_i := \gamma_i / (\delta - \beta_i \gamma_i / \alpha_{i-1})$
  else
    $\beta_i := 0; \quad \alpha_i := \gamma_i / \delta$
  end if
  $z_i := n_i + \beta_i z_{i-1}$
  $q_i := m_i + \beta_i q_{i-1}$
  $p_i := u_i + \beta_i p_{i-1}$
  $x_{i+1} := x_i + \alpha_i p_i$
  $u_{i+1} := u_i - \alpha_i q_i$
  $w_{i+1} := w_i - \alpha_i z_i$
end for

- Global synchronization of dot products are overlapped with SpMV

Amount of Computations

DAXPY could very smaller than (sophisticated) preconditioning

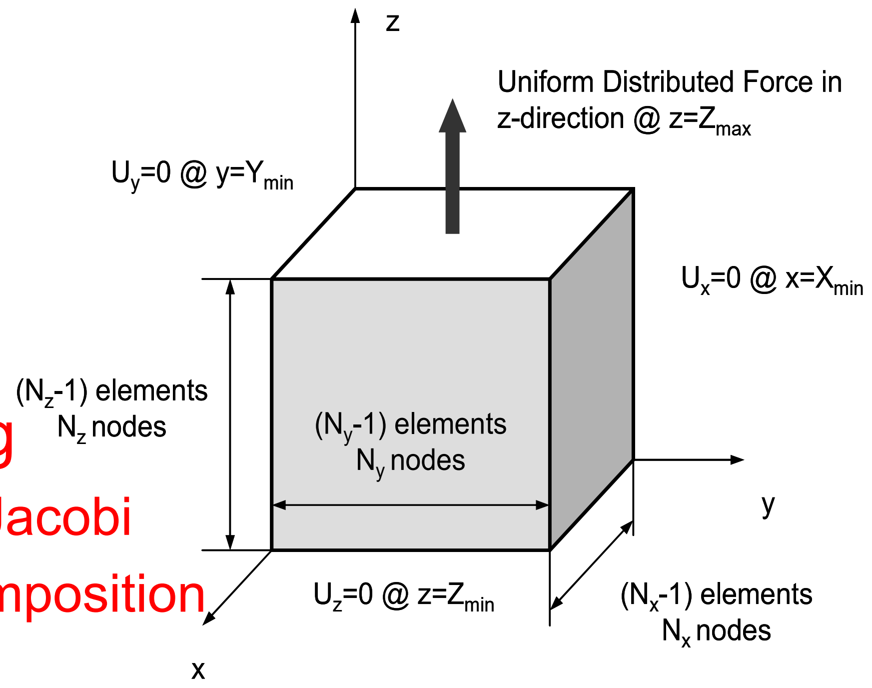
		SpMV	Precond.	Dot Prod.	DAXPY
1	Original CG	1	1	2+1	3
2	Chronopoulos/ Gear	1	1	2+1	4
3	Pipelined CG	1	1	2+1	8
4	Grppp’s Algorithm	1	1	2+1	5
	Pipelined CR	1	1	2+1	6

+1 for residual norm

- Communication/Synchronization Avoiding/Reducing in Krylov Iterative Solvers
- Pipelined CG: Background
- **Pipelined CG: Results**
- Communication-Computation Overlapping
- Summary

GeoFEM/Cube

- Parallel FEM Code (& Benchmarks)
- 3D-Static-Elastic-Linear (Solid Mechanics)
- Performance of Parallel Precond. Iterative Solvers
  - 3D Tri-linear Elements
  - SPD matrices: CG solver
  - Fortran90+MPI+OpenMP
  - Distributed Data Structure
  - Localized SGS Preconditioning
    - Symmetric Gauss-Seidel, Block Jacobi
    - Additive Schwartz Domain Decomposition
  - Reordering by CM-RCM: RCM
  - MPI, OpenMP, OpenMP/MPI Hybrid

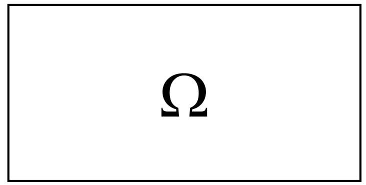


Overlapped Additive Schwartz Domain Decomposition Method

Stabilization of Localized Preconditioning: ASDD

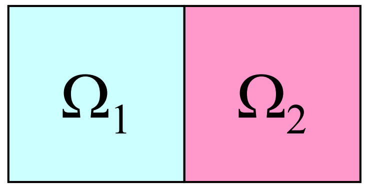
<u>Global Operation</u>

$$Mz = r$$



<u>Local Operation</u>

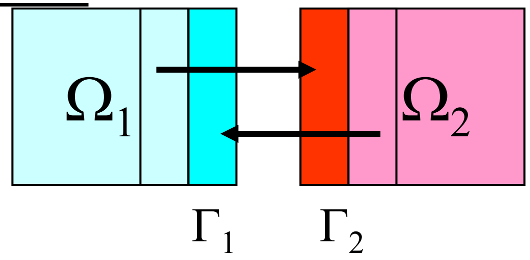
$$z_{\Omega_1} = M_{\Omega_1}^{-1} r_{\Omega_1}, \quad z_{\Omega_2} = M_{\Omega_2}^{-1} r_{\Omega_2}$$



<u>Global Nesting Correction: Repeating -> Stable</u>

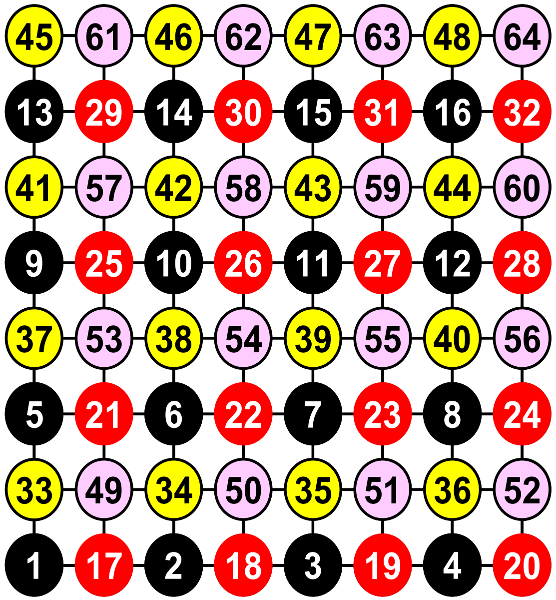
$$z_{\Omega_1}^{n} = z_{\Omega_1}^{n-1} + M_{\Omega_1}^{-1}(r_{\Omega_1} - M_{\Omega_1} z_{\Omega_1}^{n-1} - M_{\Gamma_1} z_{\Gamma_1}^{n-1})$$

$$z_{\Omega_2}^{n} = z_{\Omega_2}^{n-1} + M_{\Omega_2}^{-1}(r_{\Omega_2} - M_{\Omega_2} z_{\Omega_2}^{n-1} - M_{\Gamma_2} z_{\Gamma_2}^{n-1})$$

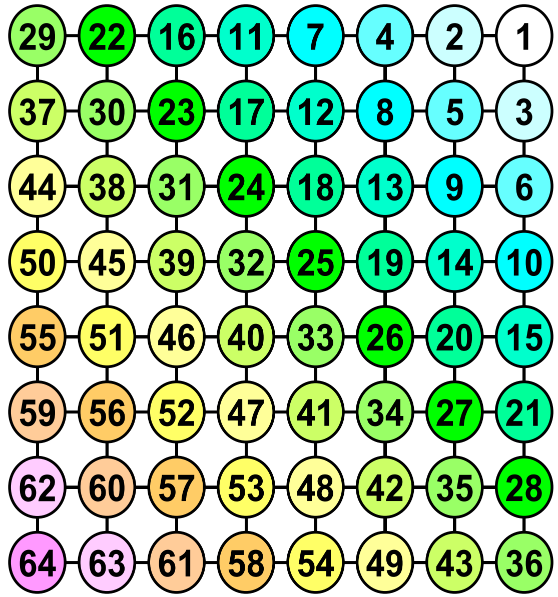


Reordering for avoiding data dependency in IC/ILU computations on each MPI process

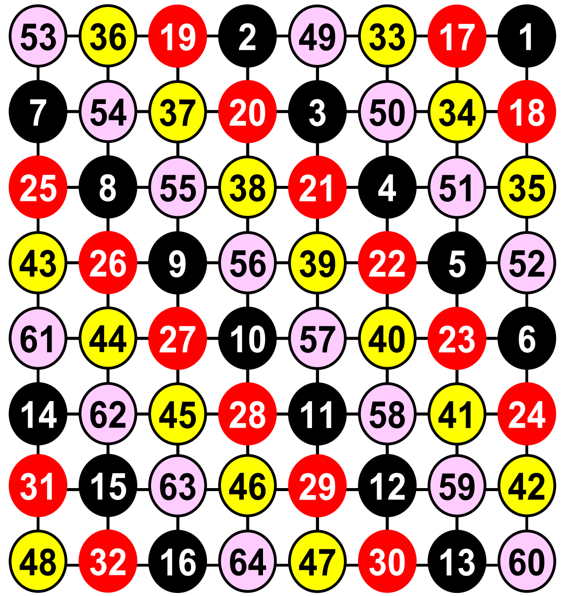
Elements in “same color” are independent: to be parallelized by OpenMP on each MPI process.



MC (Color#=4)
Multicoloring



RCM
Reverse Cuthill-Mckee



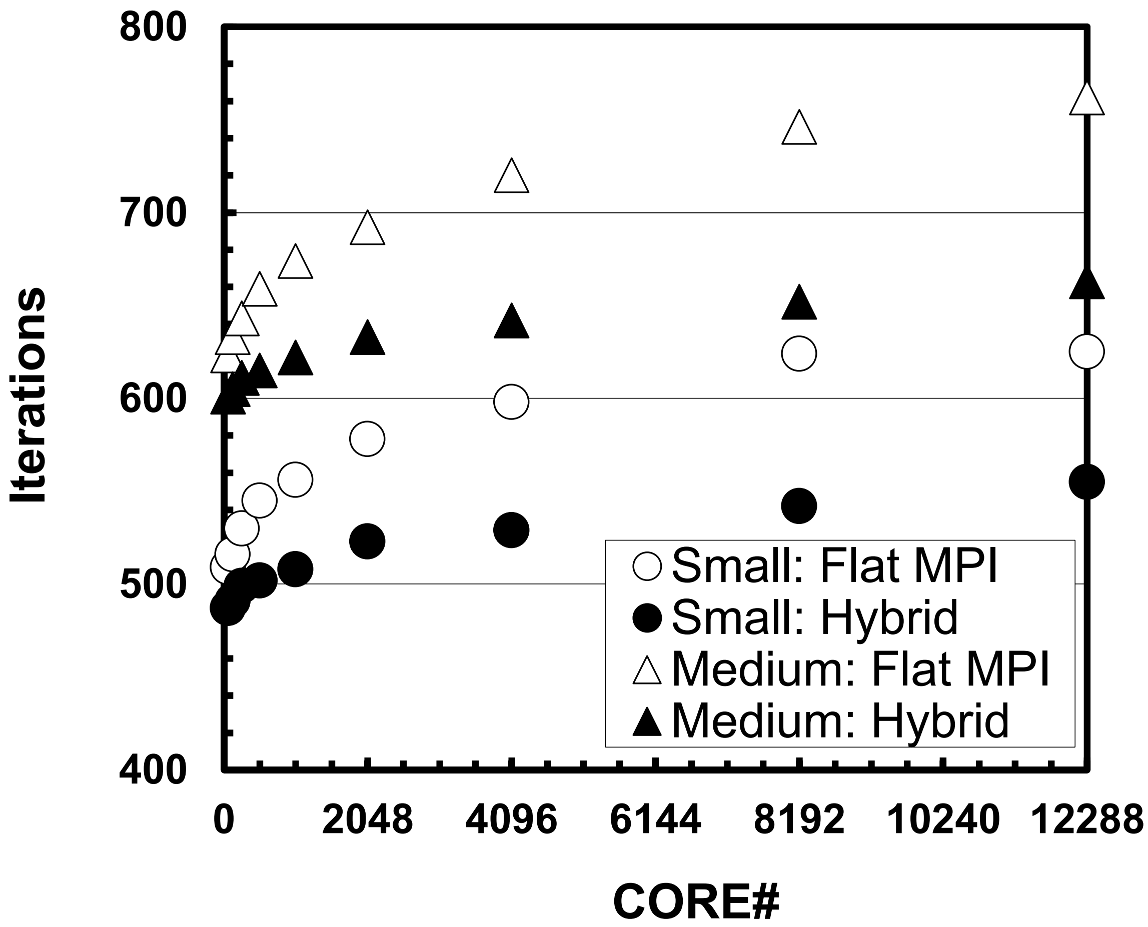
CM-RCM (Color#=4)
Cyclic MC + RCM

Results on Reedbush-U

- 4 Types of Algorithms
  - Alg.1 Original Preconditioned CG
  - Alg.2 Chronopoulos/Gear
  - Alg.3 Pipelined CG (MPI_Allreduce, MPI_Iallreduce,)
  - Alg.4 Gropp's asynchronous CG (MPI_Allreduce, MPI_Iallreduce)
- Flat MPI, OpenMP/MPI Hybrid with Reordering
- Platform
  - Integrated Supercomputer System for Data Analyses & Scientific Simulations (Reedbush-U)
  - Intel Broadwell-EP 18 cores x 2 sockets x 420 nodes
  - Intel Fortran + Intel MPI
  - 16 of 18 cores/socket, 384 nodes (= 768 sockets, 12,288 cores)

Results: Number of Iterations

- Strong Scaling
- Small
  - 256×128×144 nodes (=4,718,592)
  - 14,155,776 DOF
  - at 384 nodes (12,288 cores)
    - 8×8×6＝384 nodes/core
    - 1,152 DOF/core
- Medium
  - 256×128×288 nodes (=9,437,184)
  - 28,311,552自由度

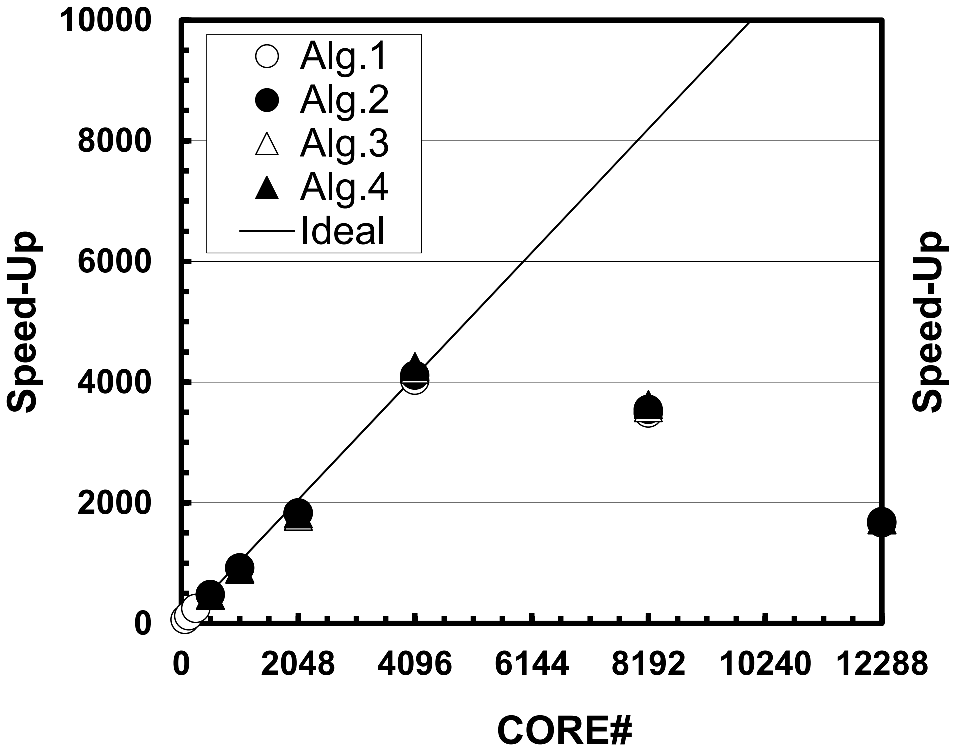


Results: Speed-Up: Small

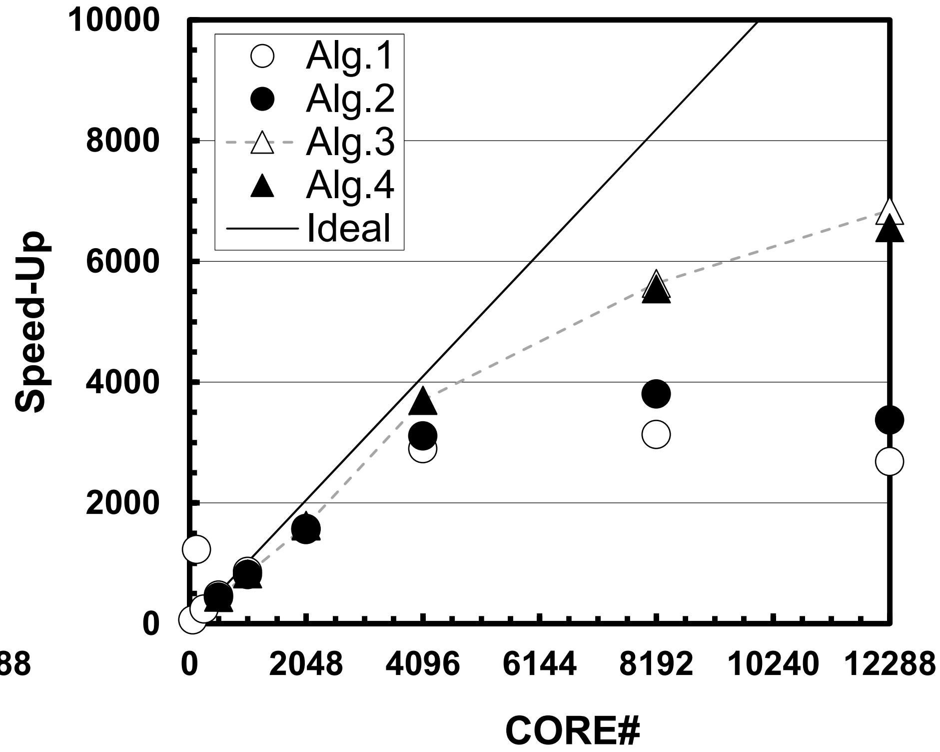
Performance of 2 nodes of Flat MPI = 64.0
(4 sockets, 64 cores)

Alg.1	Original PCG
Alg.2	Chronopoulos/Gear
Alg.3	Pipelined CG
Alg.4	Gropp's CG

Flat MPI



Hybrid

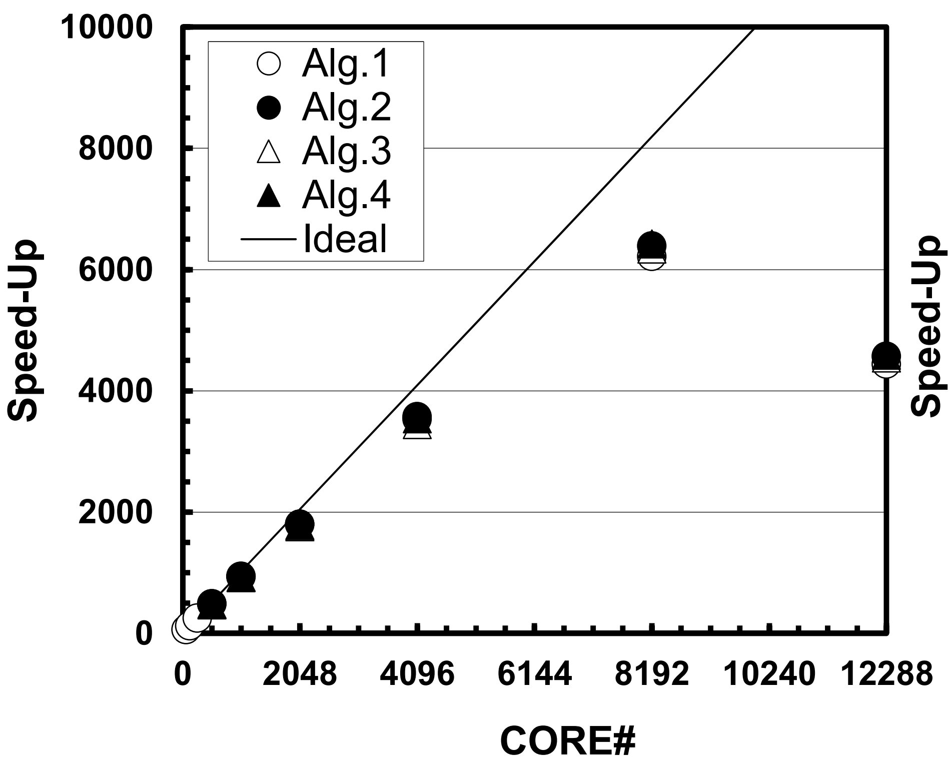


Results: Speed-Up: Medium

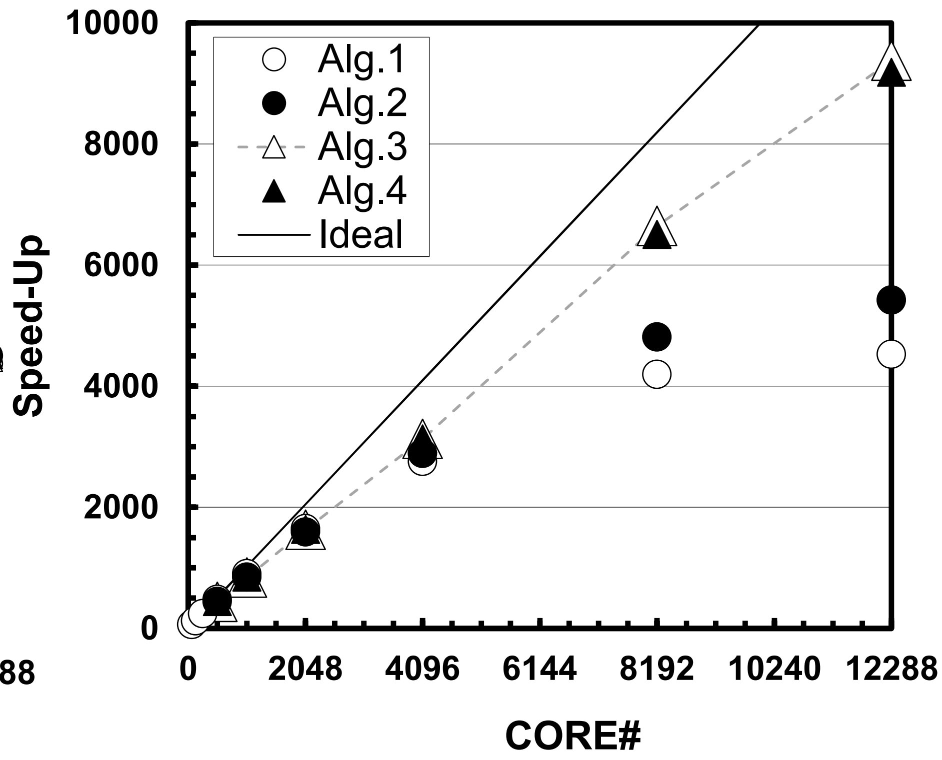
Performance of 2 nodes of Flat MPI = 64.0 (4 sockets, 64 cores)

Alg.1	Original PCG
Alg.2	Chronopoulos/Gear
Alg.3	Pipelined CG
Alg.4	Gropp's CG

Flat MPI



Hybrid



Preliminary Results on IVB Cluster

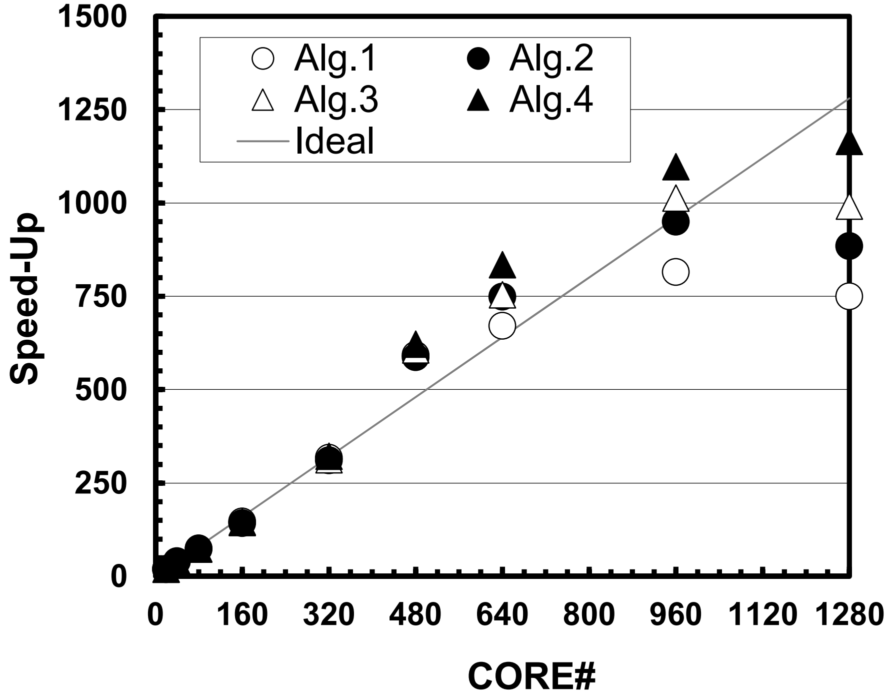
96x80x64 (491,520) nodes, 1,474,560 DOF

Flat MPI using up to 64 nodes (1,280 cores)

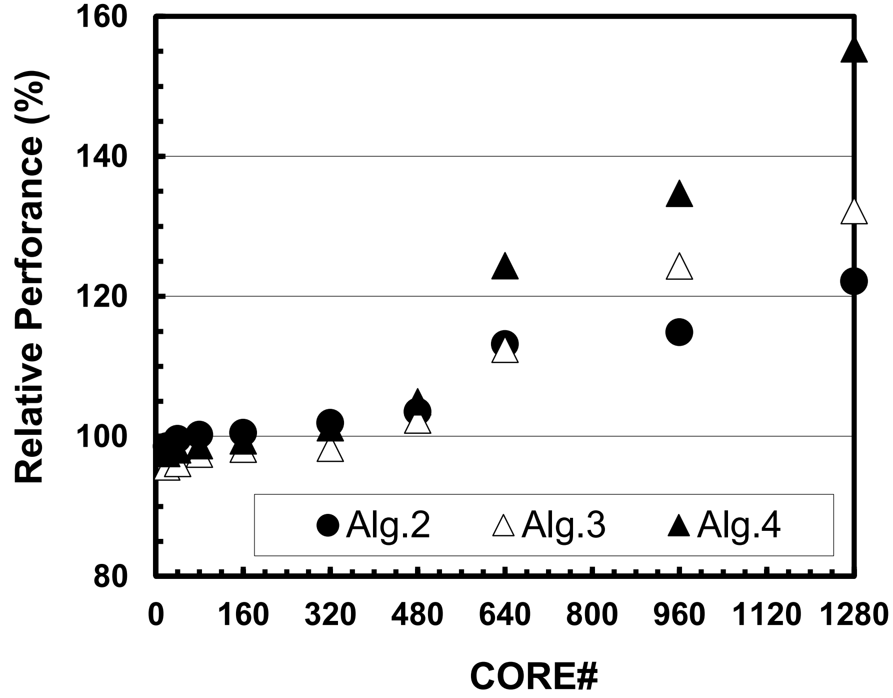
Flat MPI worked well in this case

At 64 nodes, problem size per core is equal to that of “small” case

Speed-Up (20-1,280 cores)



Relative Performance to Alg.1 (Original)

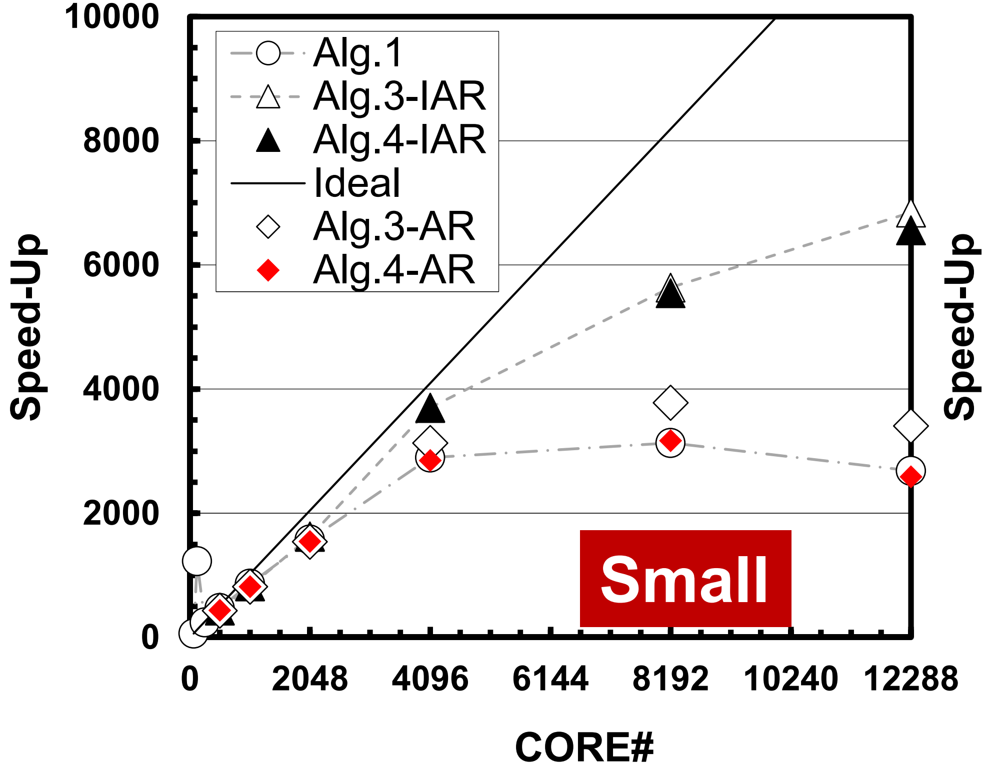


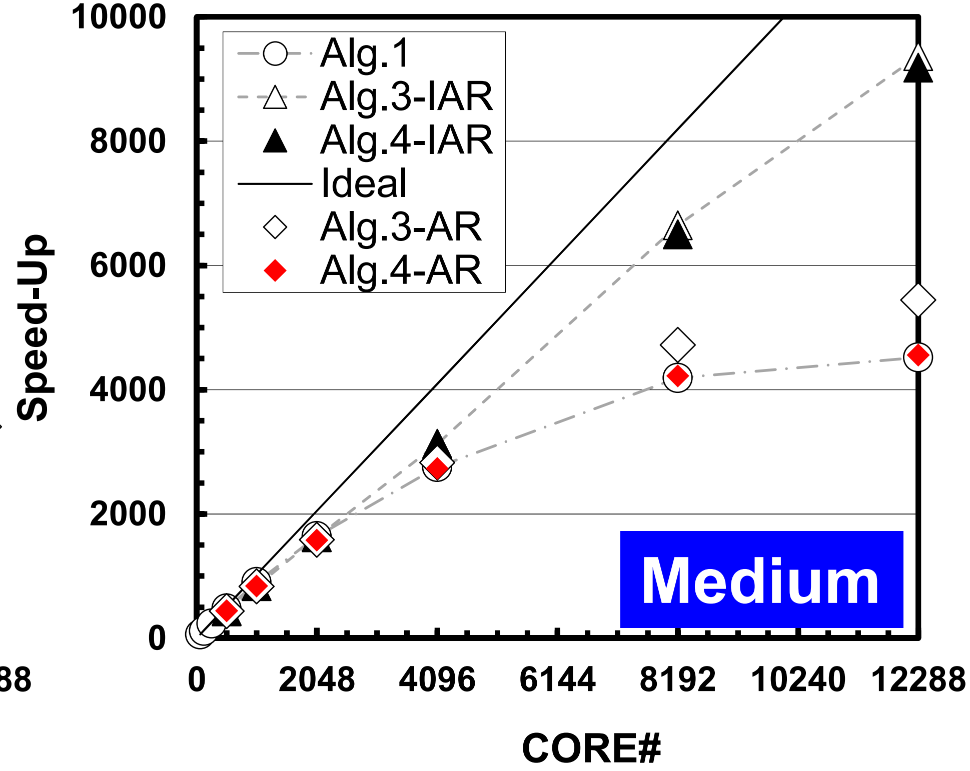
Allreduce vs. Iallreduce for Hybrid

Performance of 2 nodes of Flat MPI = 64.0 (4 sockets, 64 cores)

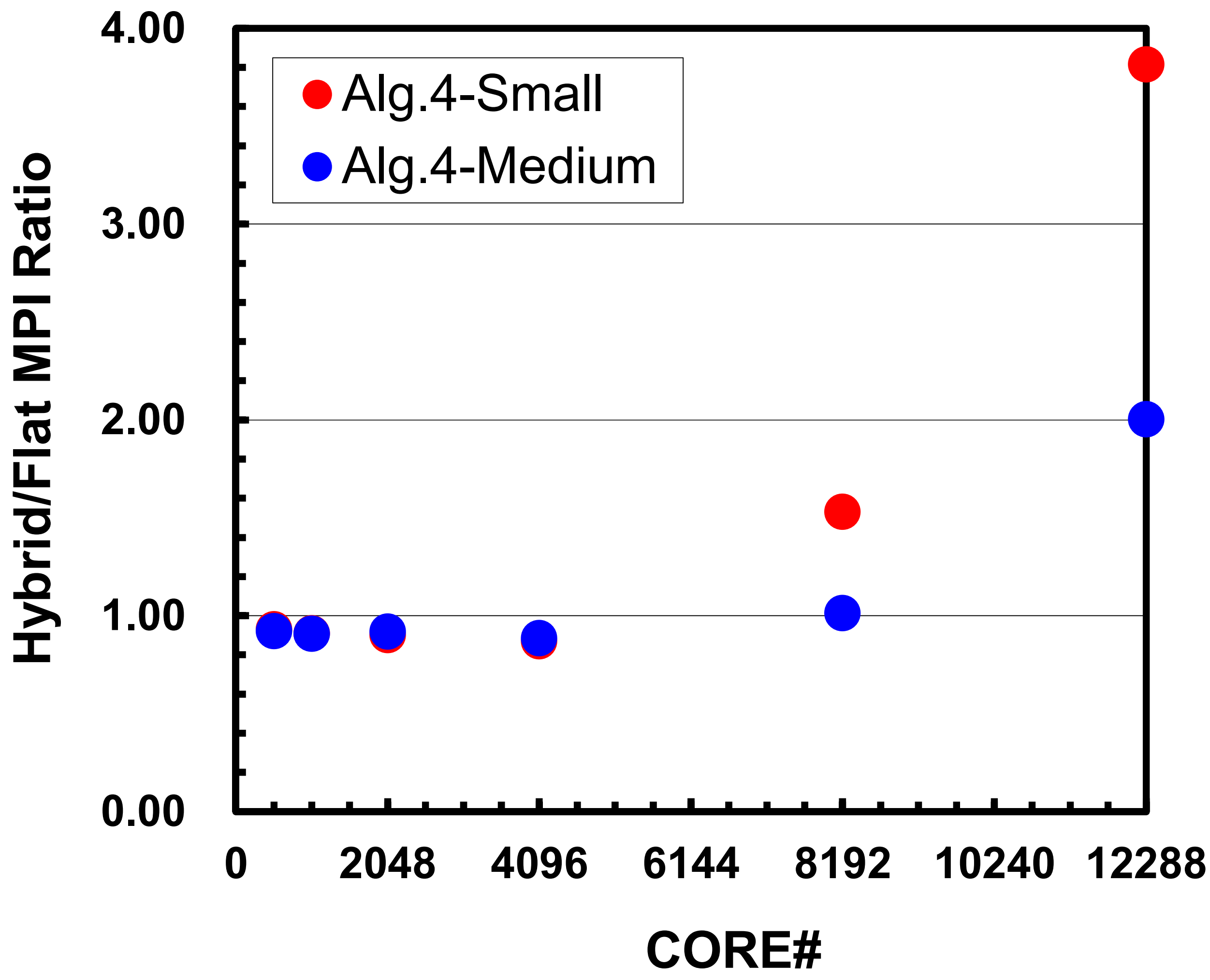
IAR: MPI_Iallreduce
AR : MPI_Allreduce

Alg.1	Original PCG
Alg.3	Pipelined CG
Alg.4	Gropp's CG



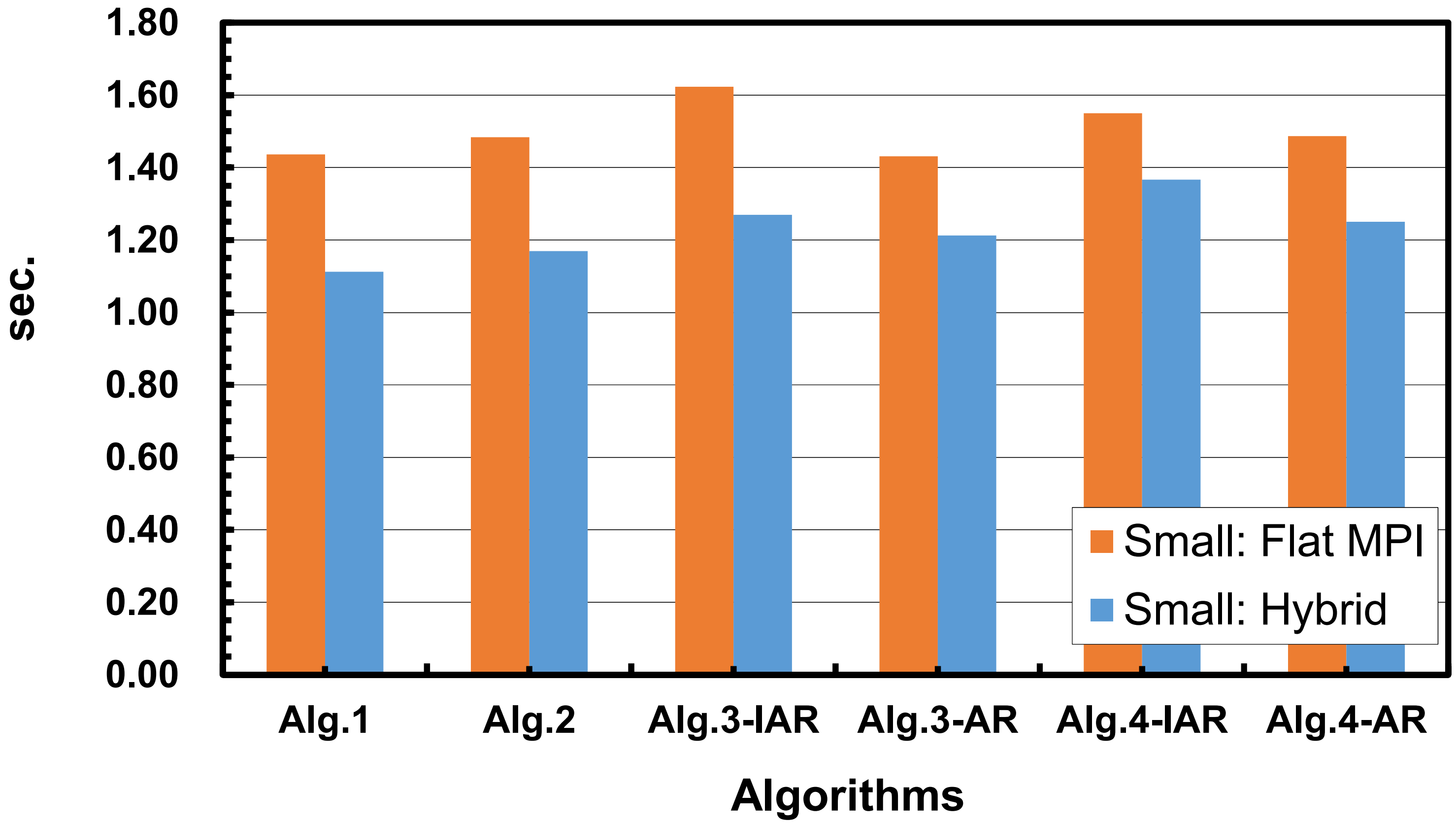


Hybrid vs. Flat MPI for Alg.4



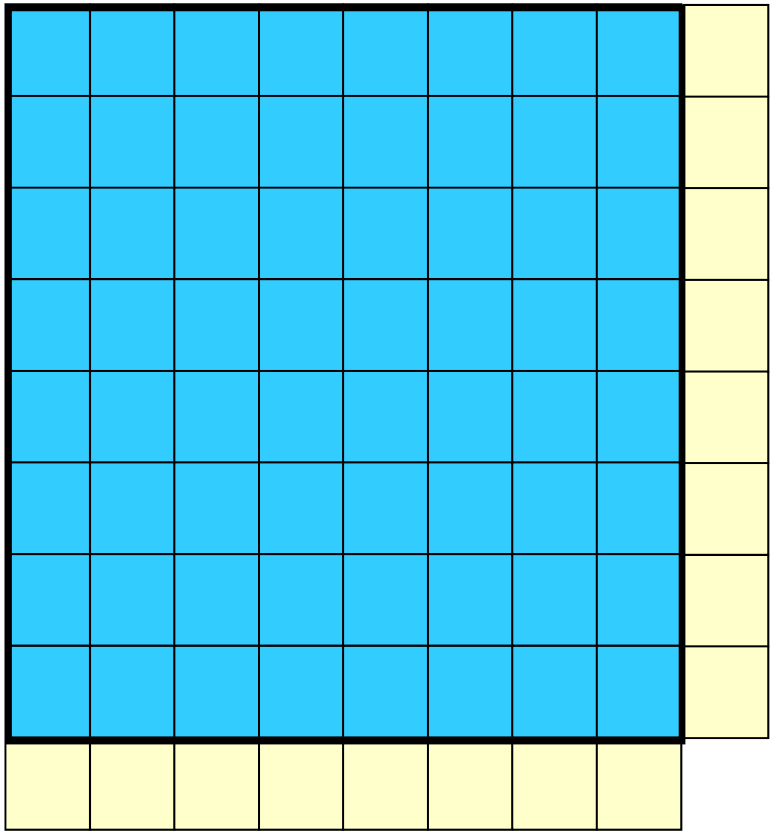
Results on 768 nodes (12,288 cores) of Fujitsu FX10 (Oakleaf-FX)

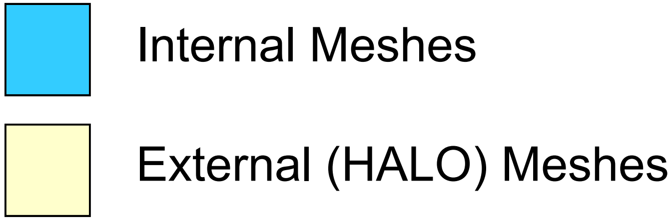
MPI-3 is not optimized
FX100 has special HW for communication



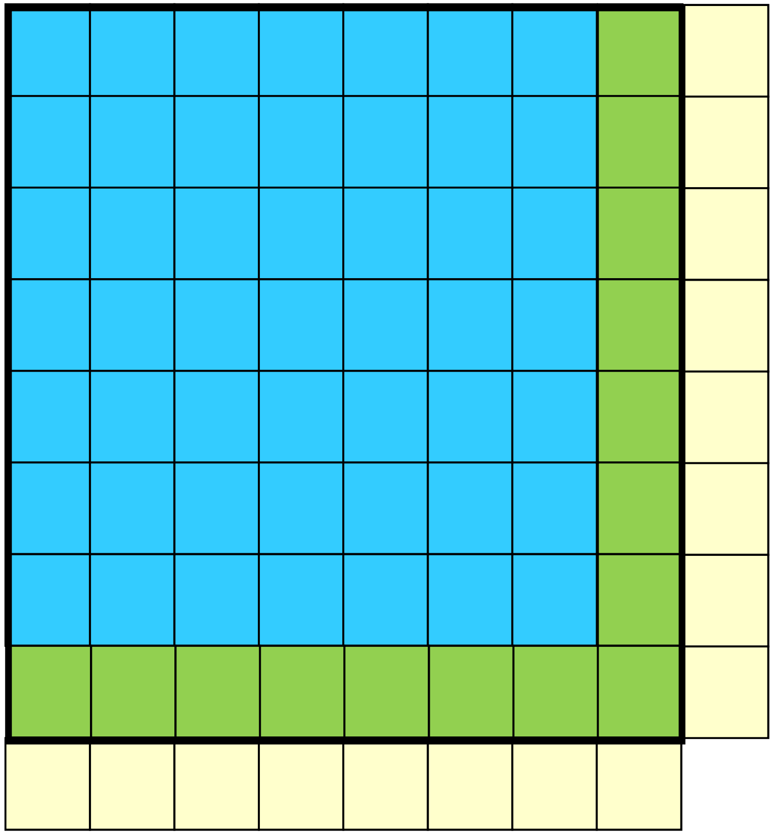
- Communication/Synchronization Avoiding/Reducing in Krylov Iterative Solvers
- Pipelined CG: Background
- Pipelined CG: Results
- **Communication-Computation Overlapping**
- Summary

Comm.-Comp. Overlapping





Comm.-Comp. Overlapping



Internal Meshes

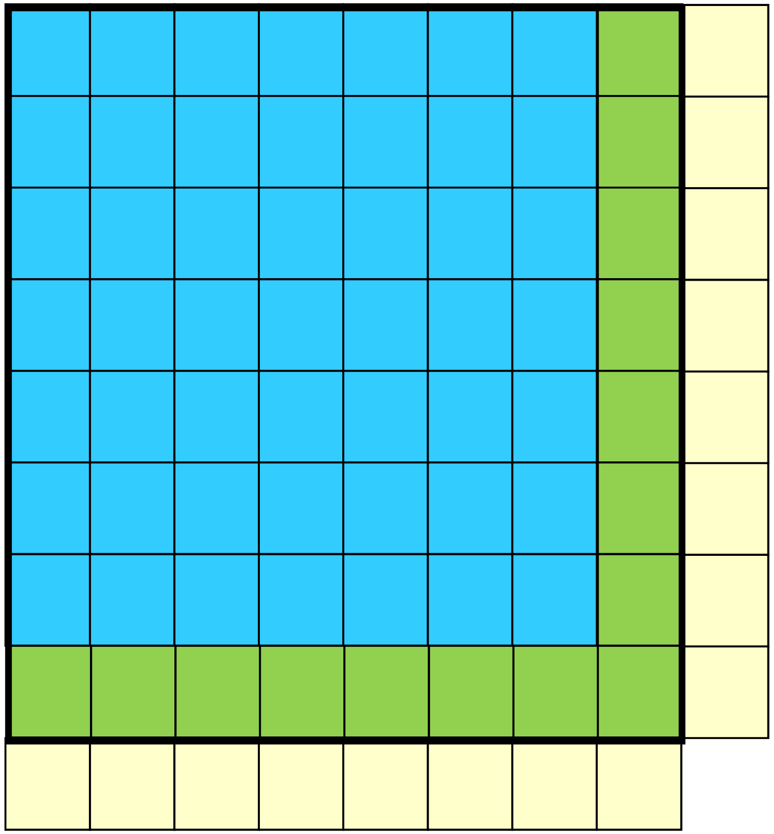
External (HALO) Meshes

Internal Meshes on Boundary’s

<u>Mat-Vec operations (SpMV)</u>

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

Comm.-Comp. Overlapping



Internal Meshes

External (HALO) Meshes

Internal Meshes on Boundary's

With Renumbering

```
call MPI_Isend
call MPI_Irecv

do i= 1, Ninn
  (calculations)
enddo

call MPI_Waitall

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

Comm.-Comp. Overlapping for SpMV

- No effects on SpMV (will be shown later)
- We need certain amount of communications
- Larger communications mean larger computations
  - Ratio of communication overhead is small …
  - Communication time itself is not so large

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”

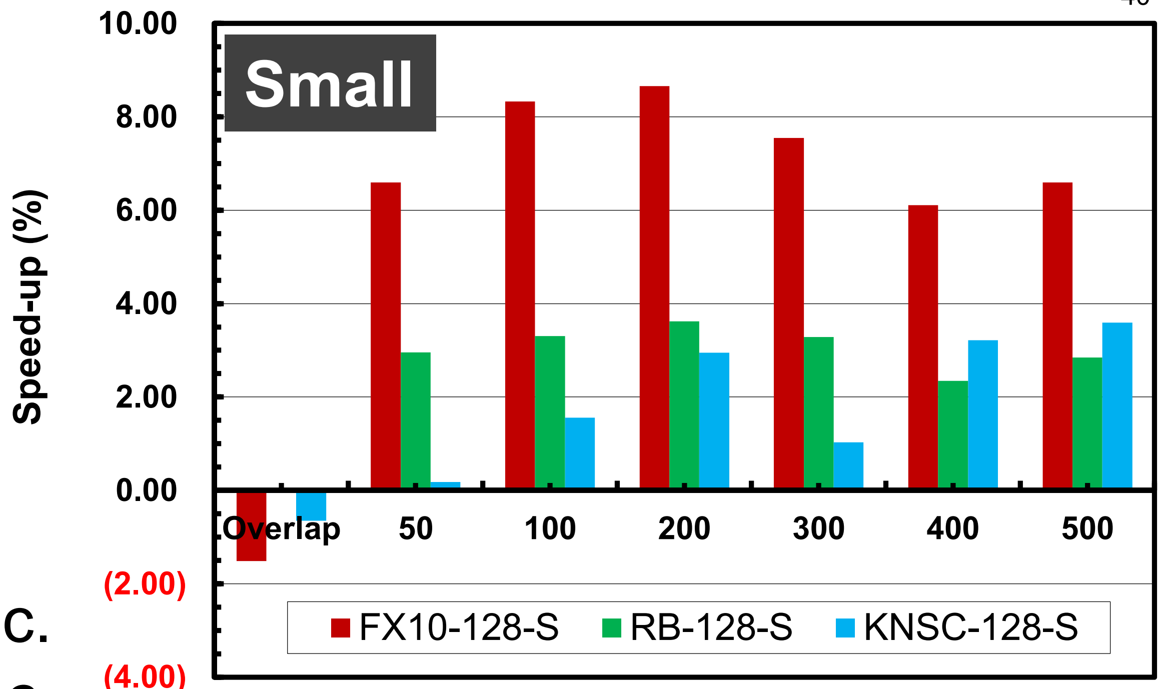
```
!$omp parallel private (neib,j,k,i,X1,X2,X3,WVAL1,WVAL2,WVAL3)
!$omp&         private (istart,inum,ii,ierr)

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

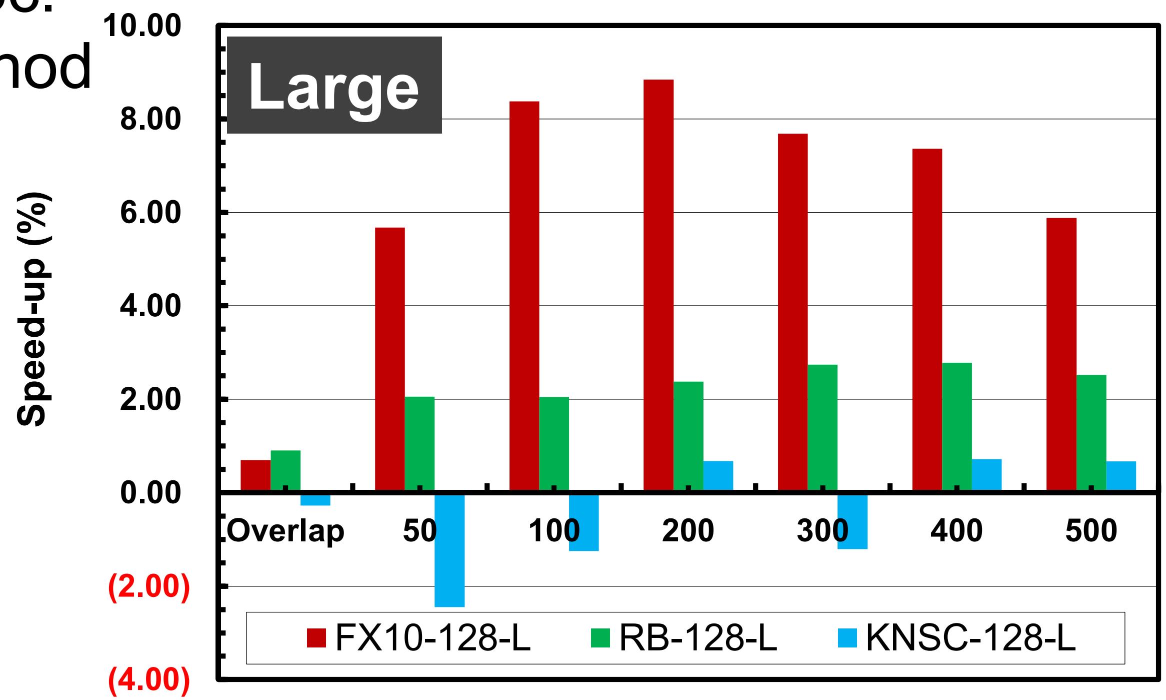
Idomura, Y. et al., Communication-overlap techniques for improved strong scaling of gyrokinetic Eulerian code beyond 100k cores on the K-computer, Int. J. HPC Appl. 28, 73-86, 2014

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

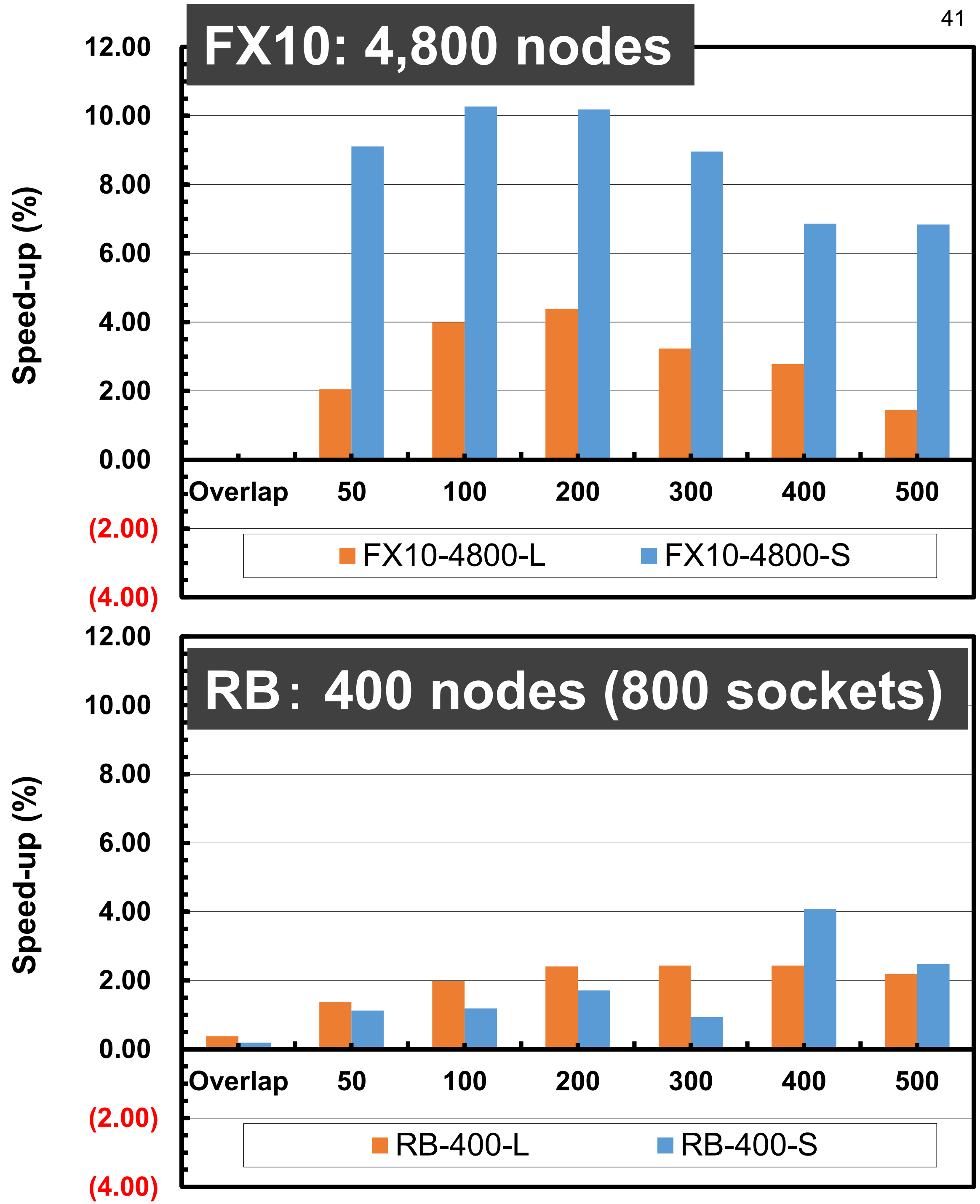
- Oakleaf-FX：FX10
- Reedbush-U：RB
- IVB Cluster：KNSC
- 128 MPI Processes





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



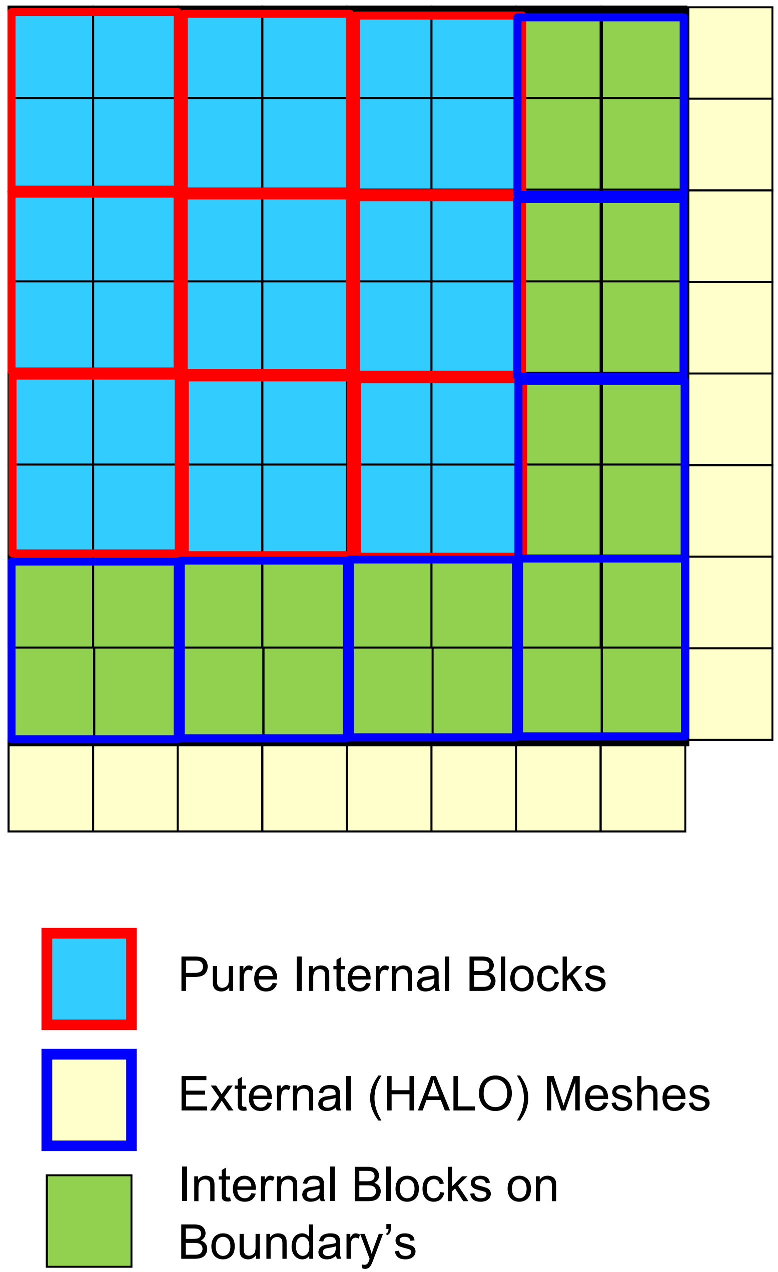
- Communication/Synchronization Avoiding/Reducing in Krylov Iterative Solvers
- Pipelined CG: Background
- Pipelined CG: Results
- Communication-Computation Overlapping
- **Summary**

Summary

- Pipelined CG, Gropp's CG
  - Effect of hiding collective communication by MPI_Iallreduce is significant, especially for strong scaling
  - Alg.3 ~ Alg.4
  - Future works
    - Pipelined CR should be also evaluated
      - Dr. Ghysels's recommendation
    - Application to Multigrid, HID (Hierarchical Interface Decompotision)
    - Evaluation on FX100
- Loop Scheduling for OpenMP
  - Effect is significant on FX10
  - Detailed profiling needed
  - Good target for AT (already done)

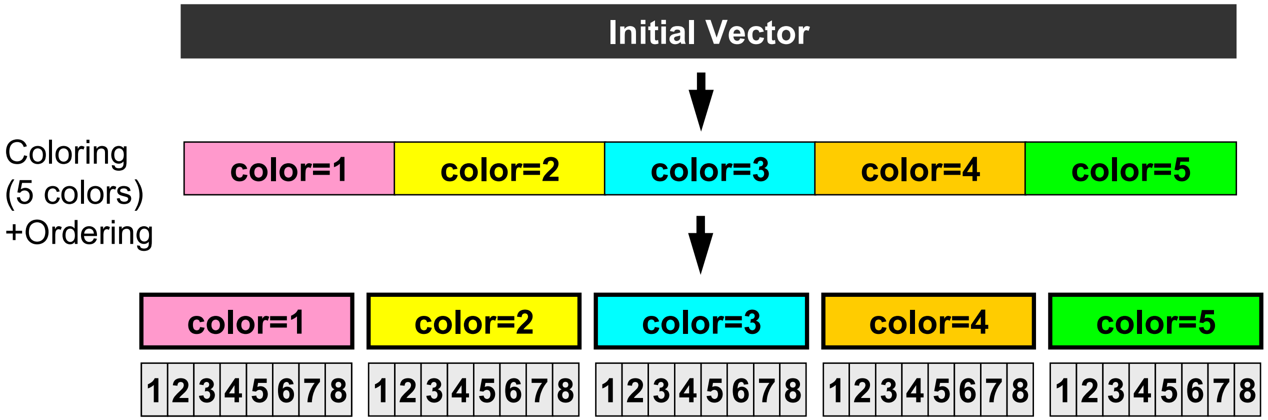
Next Stage …

- Combined Methods
  - Pipelined CG
  - ILU/IC
  - Comm.-Comp. Overlapping
  - Loop Scheduling
- We need separate numbering by reordering for internal and boundary nodes
  - More iterations needed
  - Blocking could be a remedy
- Coalesced numbering is more suitable than sequential numbering.



Coalesced & Sequential Numbering

Coalesced
Good for GPU
Continuous numbering in each color



Sequential
Good for CPU
Cache is well-utilized
Continuous numbering for each thread