全国共同利用施設
東京大学情報基盤センター
Information Technology Center, The University of Tokyo

東京大学
THE UNIVERSITY OF TOKYO

Iterative Linear Solvers for Sparse Matrices

Kengo Nakajima

Information Technology Center, The University of Tokyo, Japan

2013 International Summer School on HPC Challenges
in Computational Sciences
New York University, New York, NY
June 24-28, 2013

TOC

- Sparse Matrices
- Iterative Linear Solvers
  - Preconditioning
  - Parallel Iterative Linear Solvers
  - Multigrid Method
  - Recent Technical Issues
- Example of Parallel MGCG
- ppOpen-HPC

TOC

- Sparse Matrices
- Iterative Linear Solvers
  - Preconditioning
  - Parallel Iterative Linear Solvers
  - **Multigrid Method**
  - **Recent Technical Issues**
- **Example of Parallel MGCG**
- ppOpen-HPC

Parallel Iterative Solvers

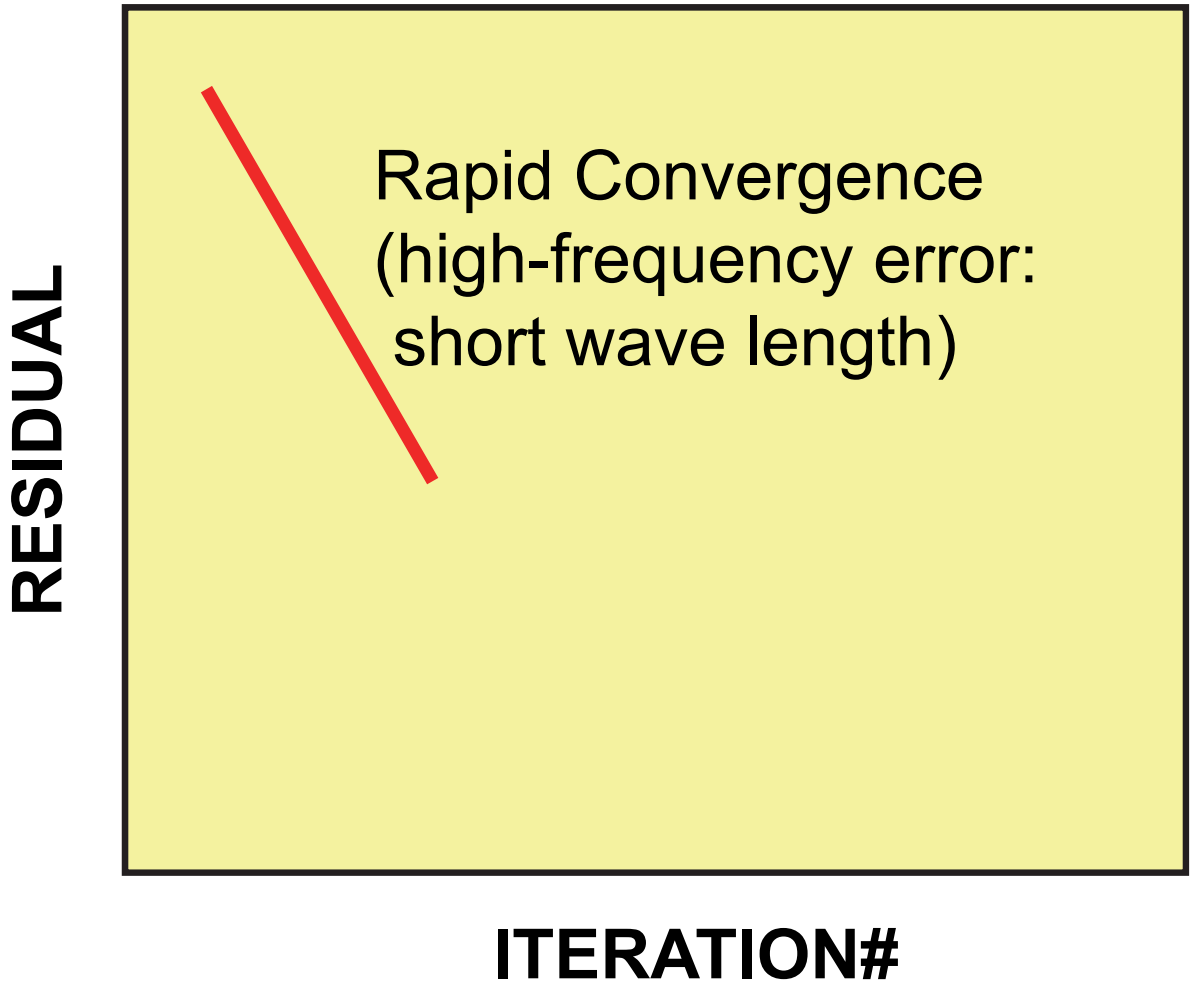
- Both of convergence (robustness) and efficiency (single/parallel) are important
- Global communications needed
  - Mat-Vec (P2P communications, MPI_Isend/Irecv/Waitall): Local Data Structure with HALO
    - effect of latency
  - Dot-Products (MPI_Allreduce)
  - Preconditioning (up to algorithm)
- Remedy 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]
- Parallel “Direct” Solvers (e.g. SuperLU, MUMPS etc.)

- Sparse Matrices
- Iterative Linear Solvers
  - Preconditioning
  - Parallel Iterative Linear Solvers
  - Multigrid Method
  - Recent Technical Issues
- Example of Parallel MGCG
- ppOpen-HPC

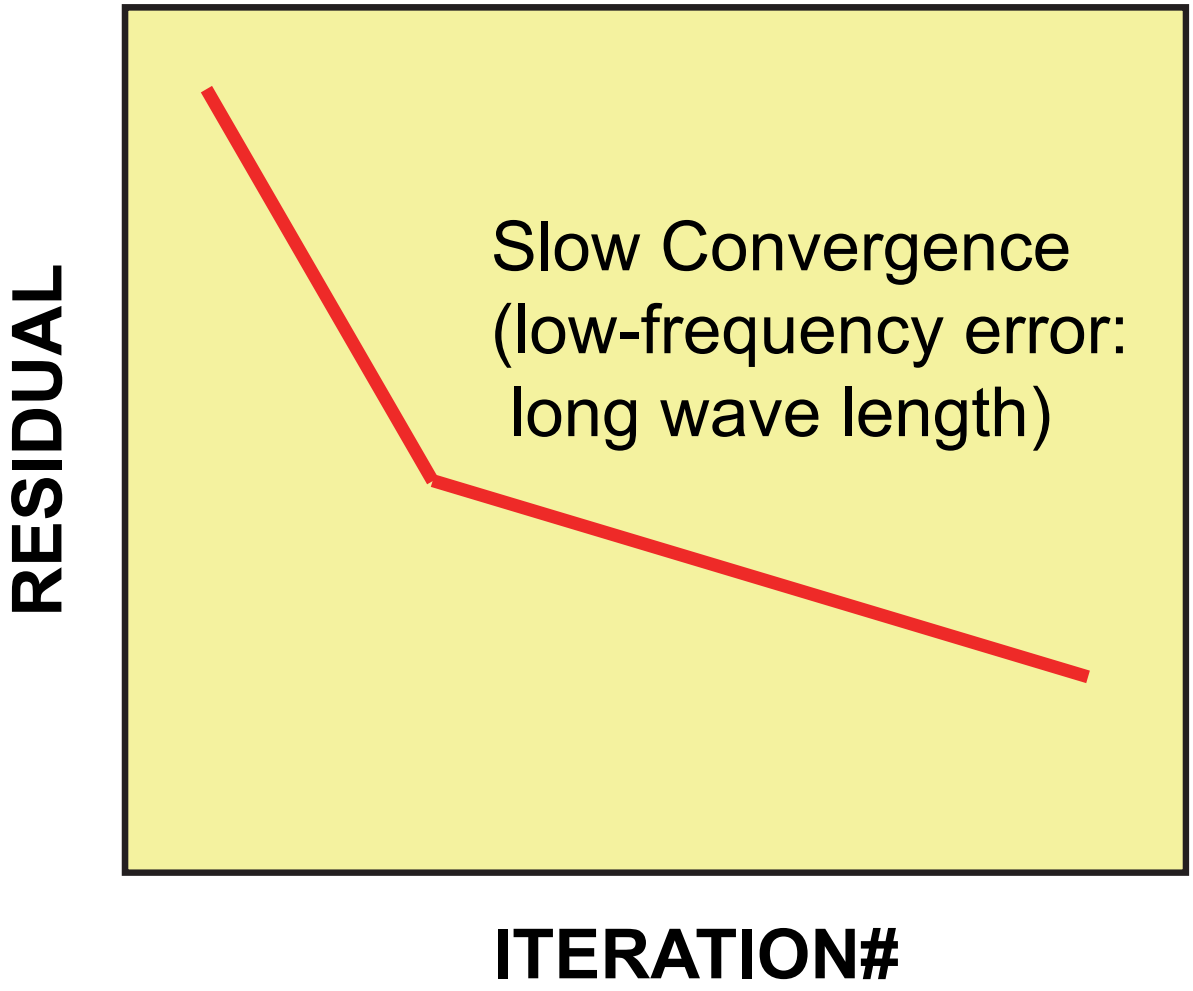
Around the multigrid in a single slide

- Multigrid is a scalable method for solving linear equations.
- Relaxation methods (smoother/smoothing operator in MG world) such as Gauss-Seidel efficiently damp high-frequency error but do not eliminate low-frequency error.
- The multigrid approach was developed in recognition that this low-frequency error can be accurately and efficiently solved on a coarser grid.
- Multigrid method uniformly damps all frequencies of error components with a computational cost that depends only linearly on the problem size (=scalable).
  - Good for large-scale computations
- Multigrid is also a good preconditioning algorithm for Krylov iterative solvers.

Convergence of Gauss-Seidel & SOR



Convergence of Gauss-Seidel & SOR

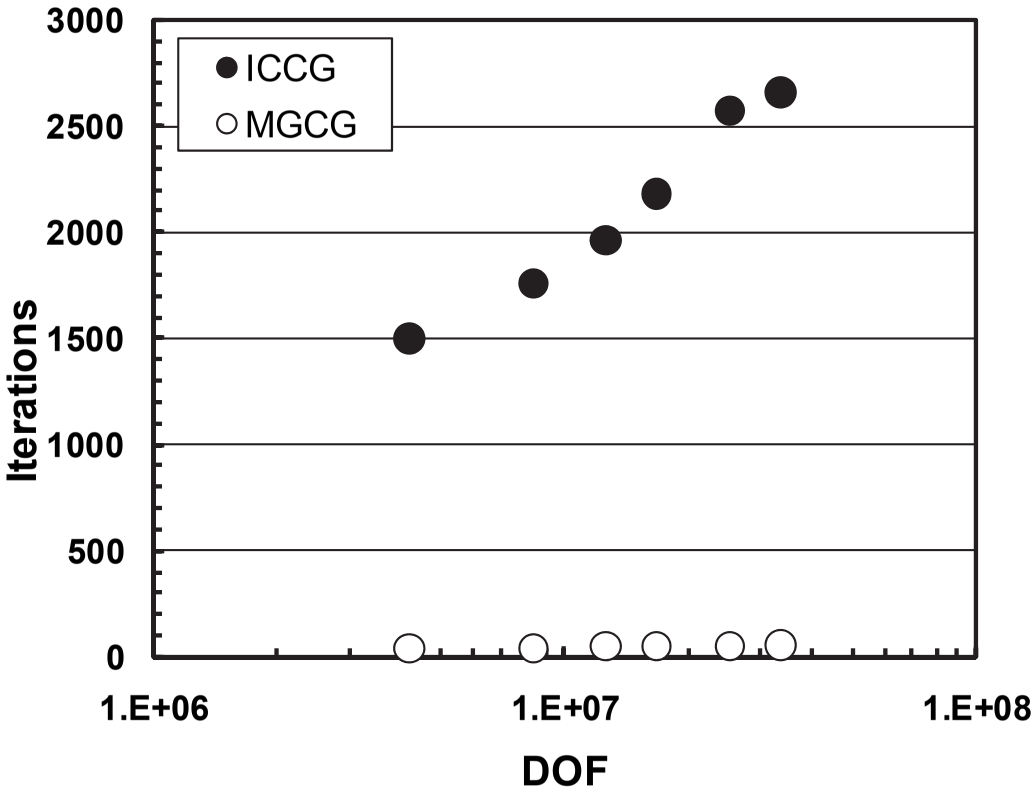


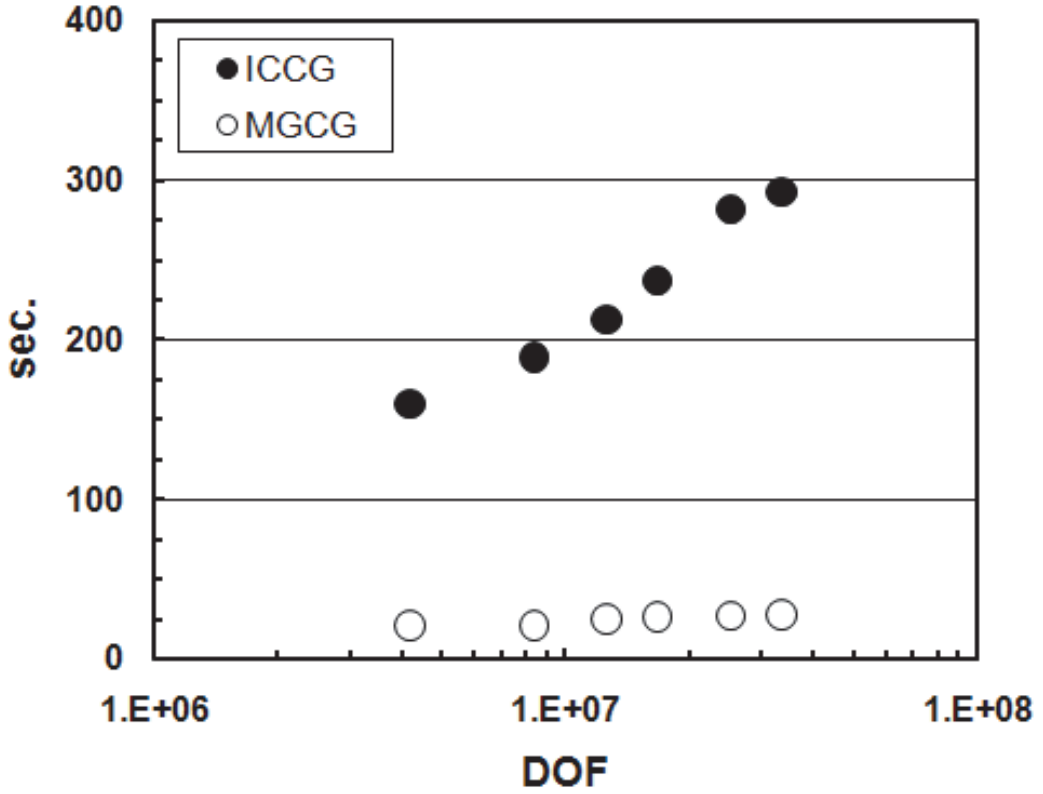
Around the multigrid in a single slide

- Multigrid is a scalable method for solving linear equations.
- Relaxation methods (smoother/smoothing operator in MG world) such as Gauss-Seidel efficiently damp high-frequency error but do not eliminate low-frequency error.
- The multigrid approach was developed in recognition that this low-frequency error can be accurately and efficiently solved on a coarser grid.
- Multigrid method uniformly damps all frequencies of error components with a computational cost that depends only linearly on the problem size (=scalable).
  - Good for large-scale computations
- Multigrid is also a good preconditioning algorithm for Krylov iterative solvers.

Multigrid is scalable
Weak Scaling: Problem Size/Core Fixed for 3D Poisson Eqn's ($\Delta\phi=q$)

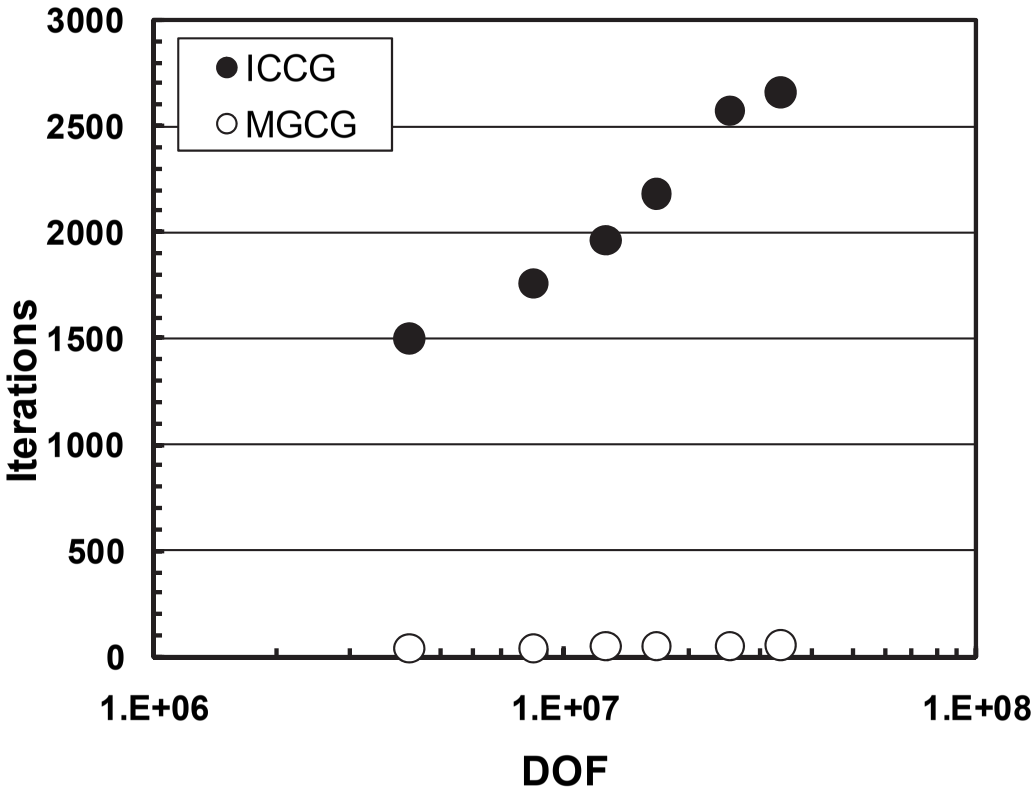
MGCG= Conjugate Gradient with Multigrid Preconditioning

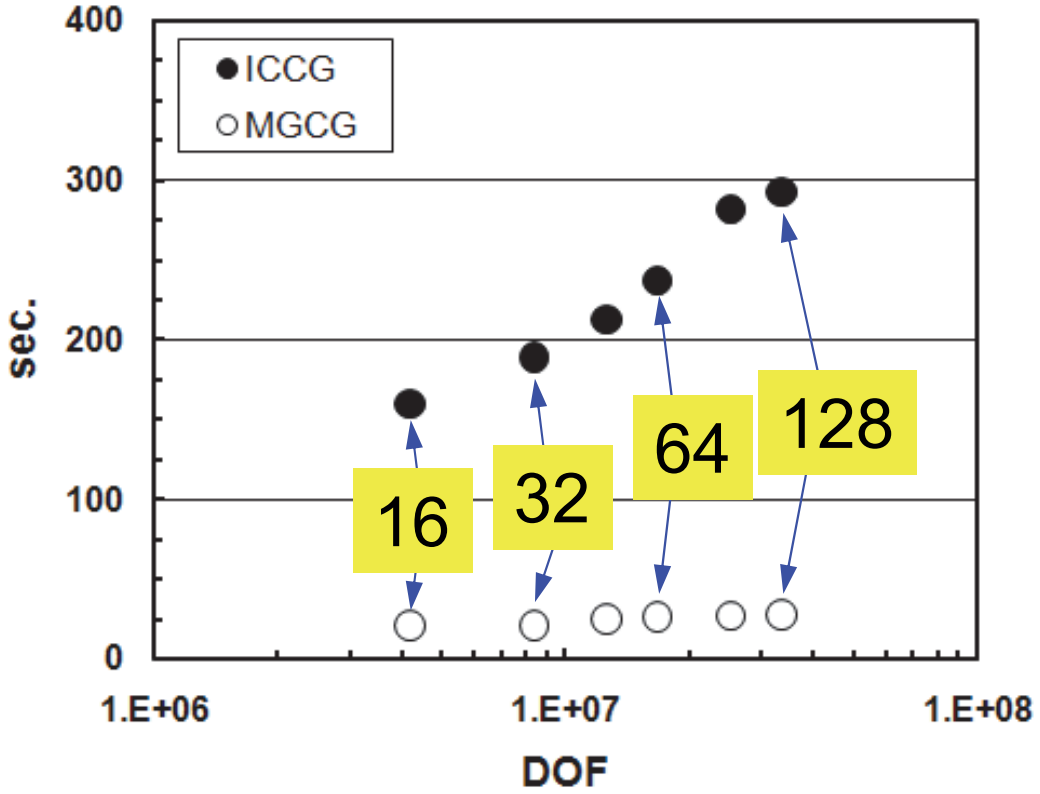




Multigrid is scalable
Weak Scaling: Problem Size/Core Fixed

Comp. time of MGCG for weak scaling is constant:
=> scalable





Procedure of Multigrid (1/3)

Multigrid is a scalable method for solving linear equations. Relaxation methods such as Gauss-Seidel efficiently damp high-frequency error but do not eliminate low-frequency error. The multigrid approach was developed in recognition that this low-frequency error can be accurately and efficiently solved on a coarser grid. This concept is explained here in the following simple 2-level method. If we have obtained the following linear system on a fine grid :

$$A_F \, u_F = f$$

and A_C as the discrete form of the operator on the coarse grid, a simple coarse grid correction can be given by :

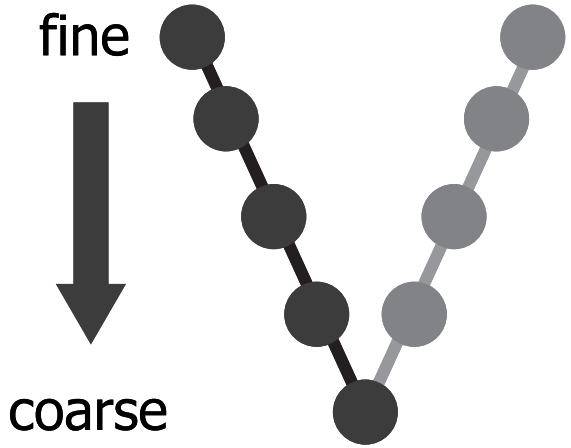
$$u_F^{(i+1)} = u_F^{(i)} + R^T \, A_C^{-1} \, R \, (\, f - A_F \, u_F^{(i)} \,)$$

where R^T is the matrix representation of linear interpolation from the coarse grid to the fine grid (*prolongation* operator) and R is called the restriction operator. Thus, it is possible to calculate the residual on the fine grid, solve the coarse grid problem, and interpolate the coarse grid solution on the fine grid.

Procedure of Multigrid (2/3)

This process can be described as follows :

1. Relax the equations on the fine grid and obtain the result $u_F^{(i)} = S_F (A_F, f)$. This operator S_F (e.g., Gauss-Seidel) is called the *smoothing operator (or)*.
2. Calculate the residual term on the fine grid by $r_F = f - A_F u_F^{(i)}$.
3. *Restric*t the residual term on to the coarse grid by $r_C = R r_F$.
4. Solve the equation $A_C u_C = r_C$ on the coarse grid ; the accuracy of the solution on the coarse grid affects the convergence of the entire multigrid system.
5. Interpolate (or *prolong*) the coarse grid correction on the fine grid by $\Delta u_F^{(i)} = R^T u_C$.
6. Update the solution on the fine grid by $u_F^{(i+1)} = u_F^{(i)} + \Delta u_F^{(i)}$



$L^k W^k = F^k$ **(Linear Equation: Fine Level)**

$R^k = F^k - L^k w_1^k$

$v^k = W^k - w_1^k$, $L^k v^k = R^k$

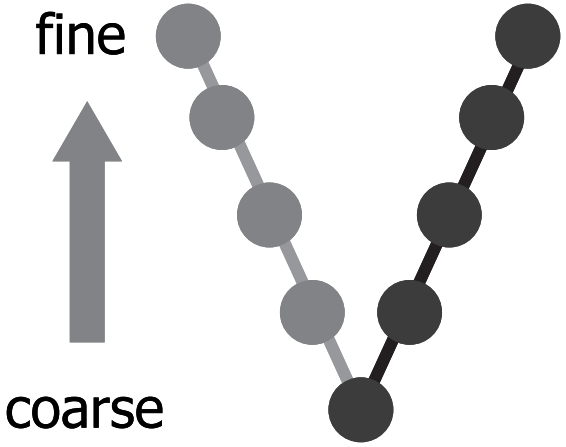
$R^{k-1} = I_k^{k-1} R^k$

$L^{k-1} v^{k-1} = R^{k-1}$ **(Linear Equation: Coarse Level)**

$v^k = I_{k-1}^k v^{k-1}$

$w_2^k = w_1^k + v^k$

w_1^k : Approx. Solution
v^k : Correction
I_k^{k-1} : Restriction Operator



$L^k W^k = F^k$ **(Linear Equation: Fine Level)**

$R^k = F^k - L^k w_1^k$

$v^k = W^k - w_1^k$, $L^k v^k = R^k$

$R^{k-1} = I_k^{k-1} R^k$

$L^{k-1} v^{k-1} = R^{k-1}$ **(Linear Equation: Coarse Level)**

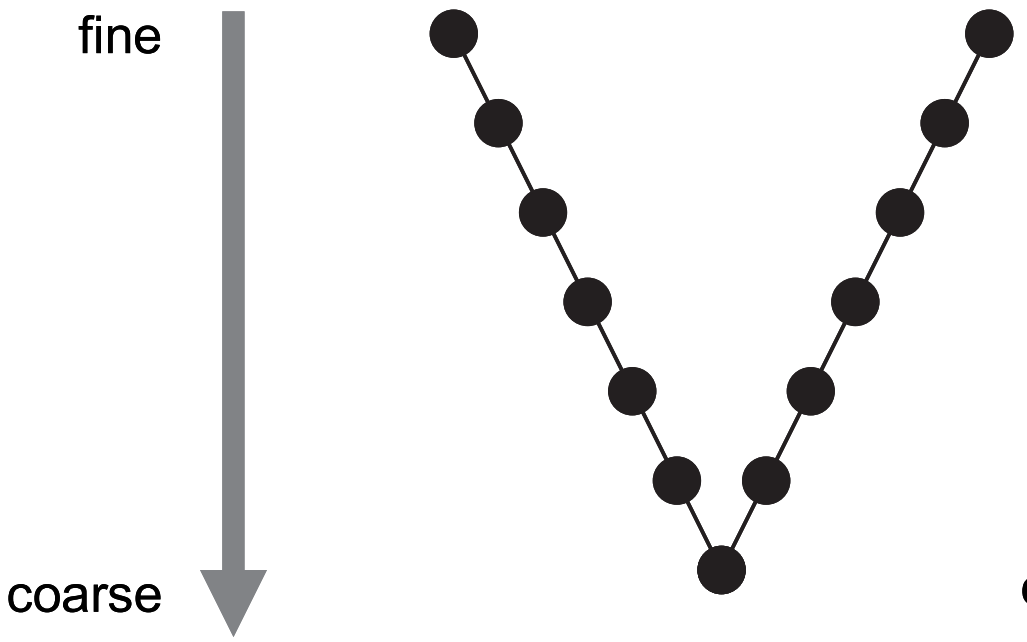
$v^k = I_{k-1}^k v^{k-1}$

$w_2^k = w_1^k + v^k$

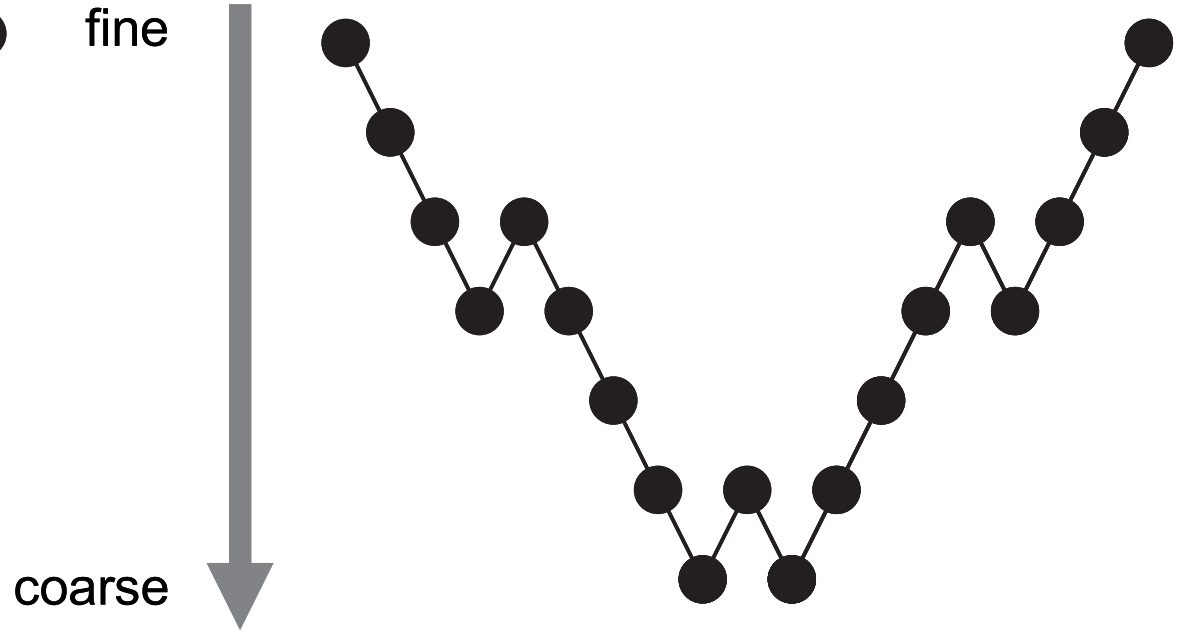
I_{k-1}^k : Prolongation Operator
w_2^k : Approx. Solution by Multigrid

Procedure of Multigrid (3/3)

- Recursive application of this algorithm for 2-level procedure to consecutive systems of coarse-grid equations gives a multigrid V-cycle. If the components of the V-cycle are defined appropriately, the result is a method that uniformly damps all frequencies of error with a computational cost that depends only linearly on the problem size.
  - In other words, multigrid algorithms are *scalable*.
- In the V-cycle, starting with the finest grid, all subsequent coarser grids are visited only once.
  - In the down-cycle, smoothers damp oscillatory error components at different grid scales.
  - In the up-cycle, the smooth error components remaining on each grid level are corrected using the error approximations on the coarser grids.
- Alternatively, in a W-cycle, the coarser grids are solved more rigorously in order to reduce residuals as much as possible before going back to the more expensive finer grids.



(a) V-Cycle



(b) W-Cycle

Multigrid as a Preconditioner

- Multigrid algorithms tend to be problem-specific solutions and less robust than preconditioned Krylov iterative methods such as the IC/ILU methods.
- Fortunately, it is easy to combine the best features of multigrid and Krylov iterative methods into one algorithm
  - multigrid-preconditioned Krylov iterative methods.
- The resulting algorithm is robust, efficient and scalable.

- Mutigrid solvers and Krylov iterative solvers preconditioned by multigrid are intrinsically suitable for parallel computing.

Geometric and Algebraic Multigrid

- One of the most important issues in multigrid is the construction of the coarse grids.
- There are 2 basic multigrid approaches
  - geometric and algebraic
- In geometric multigrid, the geometry of the problem is used to define the various multigrid components.
- In contrast, algebraic multigrid methods use only the information available in the linear system of equations, such as matrix connectivity.
- Algebraic multigrid method (AMG) is suitable for applications with unstructured grids.
- Many tools for both geometric and algebraic methods on unstructured grids have been developed.

“Dark Side” of Multigrid Method

- Its performance is excellent for well-conditioned simple problems, such as homogeneous Poisson equations.
- But convergence could be worse for ill-conditioned problems.
- Extension of applicability of multigrid method is an active research area.

References

- Briggs, W.L., Henson, V.E. and McCormick, S.F. (2000) A Multigrid Tutorial Second Edition, SIAM

- Trottemberg, U., Oosterlee, C. and Schüller, A. (2001) Multigrid, Academic Press

- https://computation.llnl.gov/casc/
- Hypre (AMG Library)
  - https://computation.llnl.gov/casc/linear_solvers/sls_hypre.html

- Sparse Matrices
- Iterative Linear Solvers
  - Preconditioning
  - Parallel Iterative Linear Solvers
  - Multigrid Method
  - Recent Technical Issues
- Example of Parallel MGCG
- ppOpen-HPC

Key-Issues for Appl's/Algorithms towards Post-Peta & Exa Computing

Jack Dongarra (ORNL/U. Tennessee) at ISC 2013

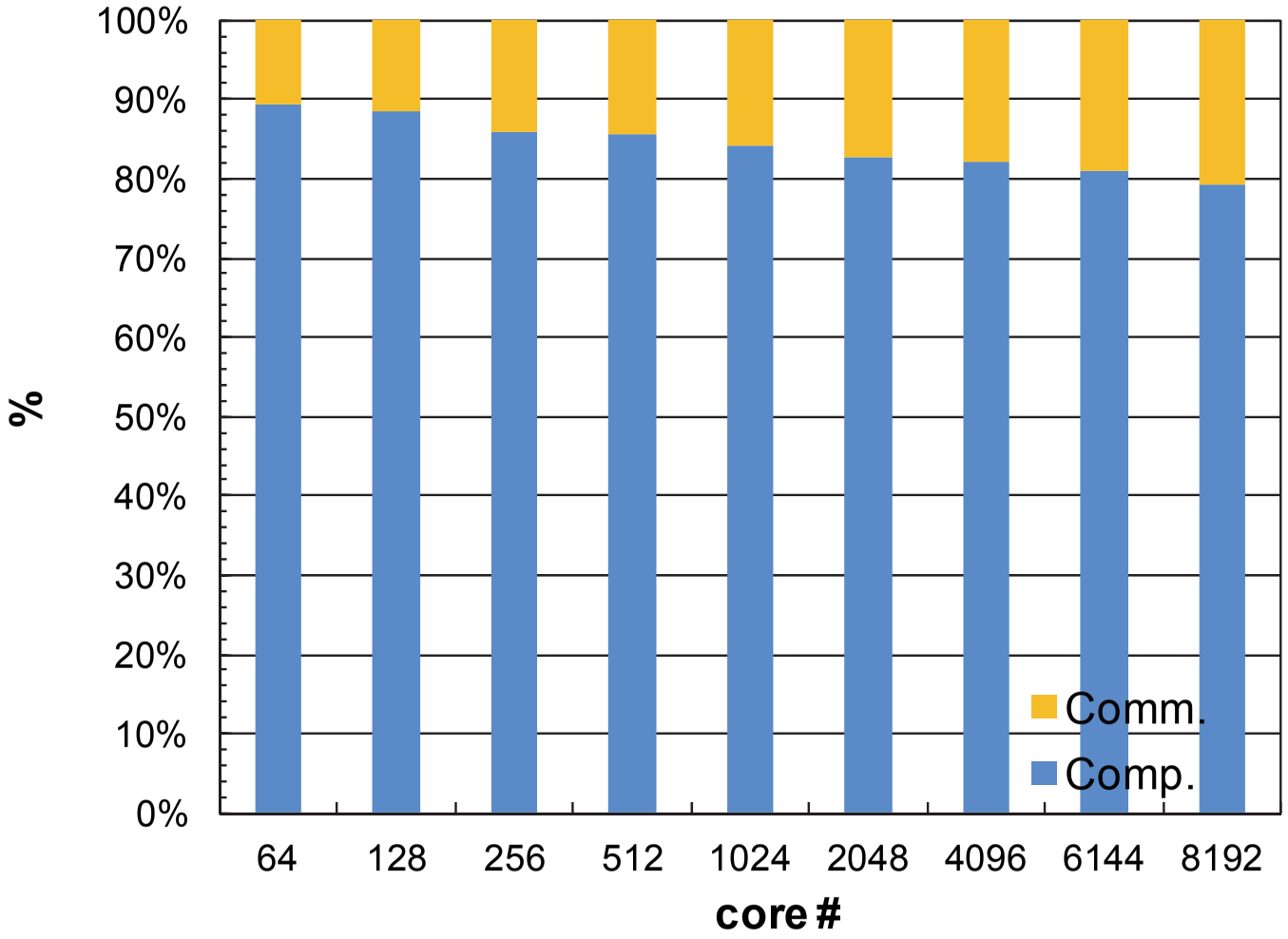
- Hybrid/Heterogeneous Architecture
  - Multicore + GPU/Manycores (Intel MIC/Xeon Phi)
    - Data Movement, Hierarchy of Memory
- Communication/Synchronization Reducing Algorithms
- Mixed Precision Computation
- Auto-Tuning/Self-Adapting
- Fault Resilient Algorithms
- Reproducibility of Results

Recent Technical Issues in Parallel Iterative Solvers

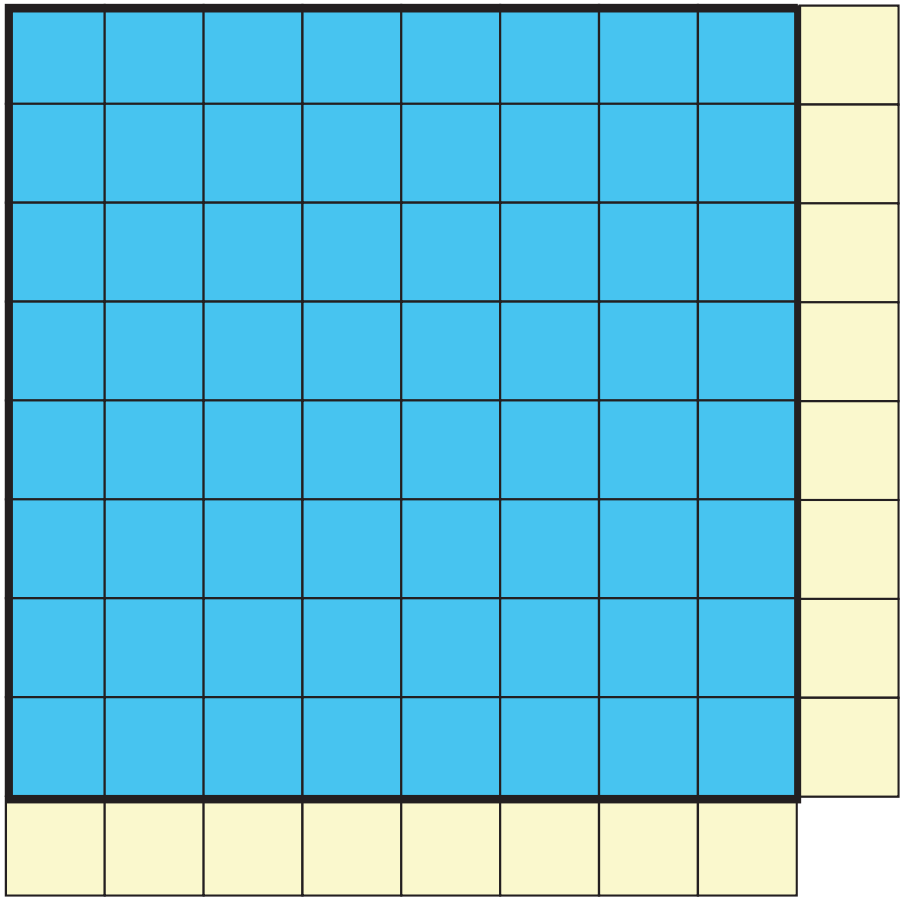
- Communication overhead becomes significant
- Communication-Computation Overlap
  - Not so effective for Mat-Vec operations
- Communication Avoiding/Reducing Algorithms

- OpenMP/MPI Hybrid Parallel Programming Model
  - (Next section)

Communication overhead becomes larger as node/core number increases Weak Scaling: MGCG on T2K Tokyo



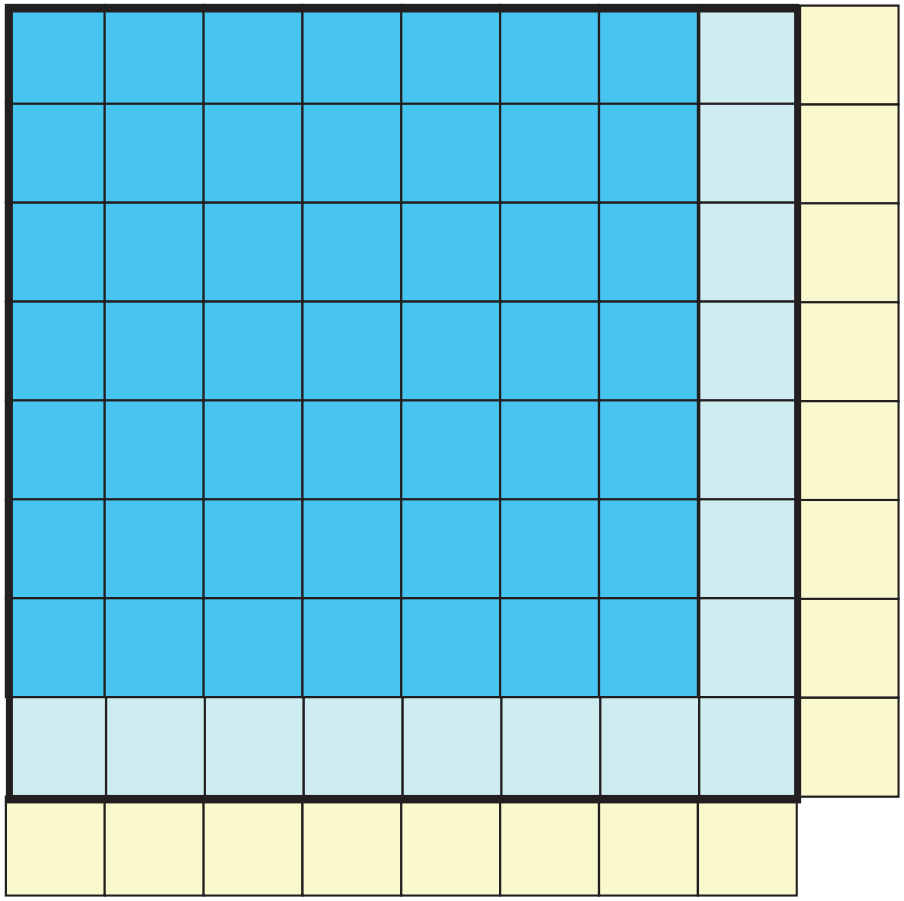
Comm.-Comp. Overlapping



Internal Meshes

External (HALO) Meshes

Comm.-Comp. Overlapping



Internal Meshes

External (HALO) Meshes

Internal Meshes on Boundary's

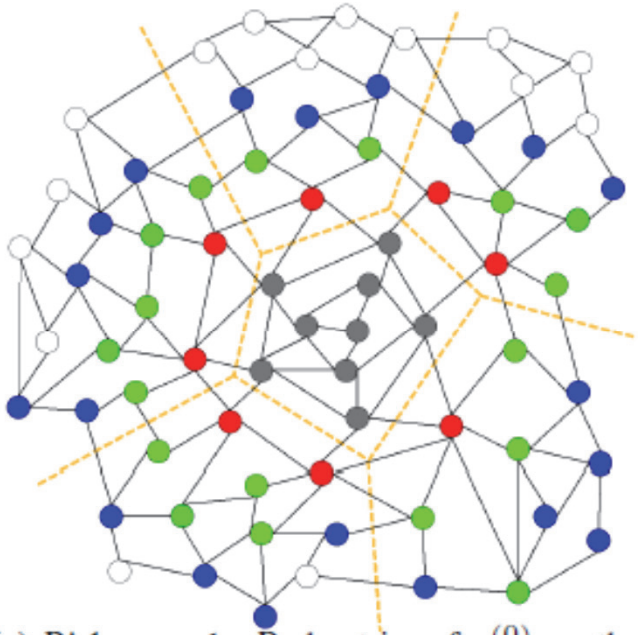
Mat-Vec operations

- Overlapping of computations of internal meshes, and importing external meshes.
- Then computation of international meshes on boundary's
- Difficult for IC/ILU on Hybrid

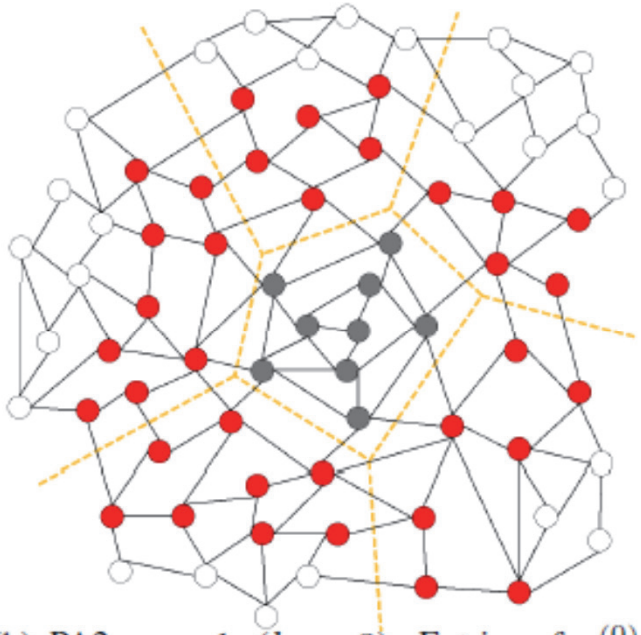
Communication Avoiding/Reducing Algorithms for Sparse Linear Solvers

- 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 becomes unstable for large *s*.
  - matrix powers kernel: Ax, A^2x, A^3x ...
    - additional computations needed
- Communication Avoiding ILU0 (CA-ILU0) [Moufawad & Grigori, 2013]
  - First attempt to CA preconditioning
  - Nested dissection reordering for limited geometries (2D FDM)

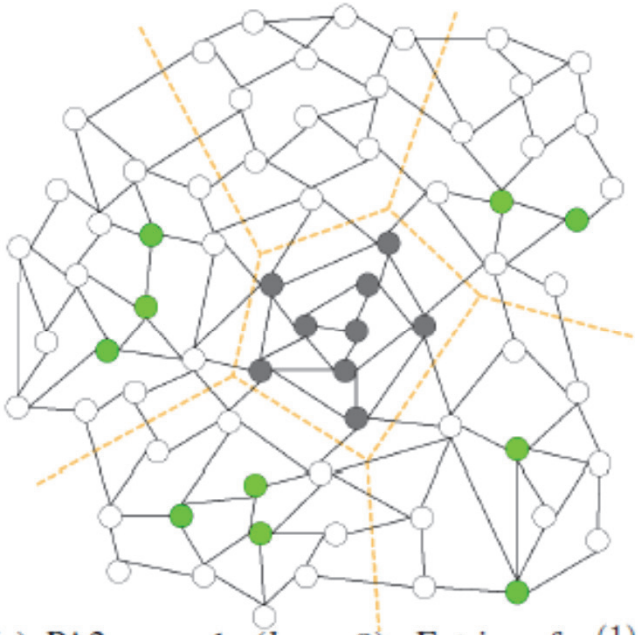
Comm. Avoiding Krylov Iterative Methods using "Matrix Powers Kernel"



(a) PA1 example: Red entries of $x^{(0)}$ are the ones needed when $k = 1$, green are the additional ones needed when $k = 2$ and blue are the additional ones needed when $k = 3$.



(b) PA2 example ($k = 3$): Entries of $x^{(0)}$ which need to be fetched are colored red.

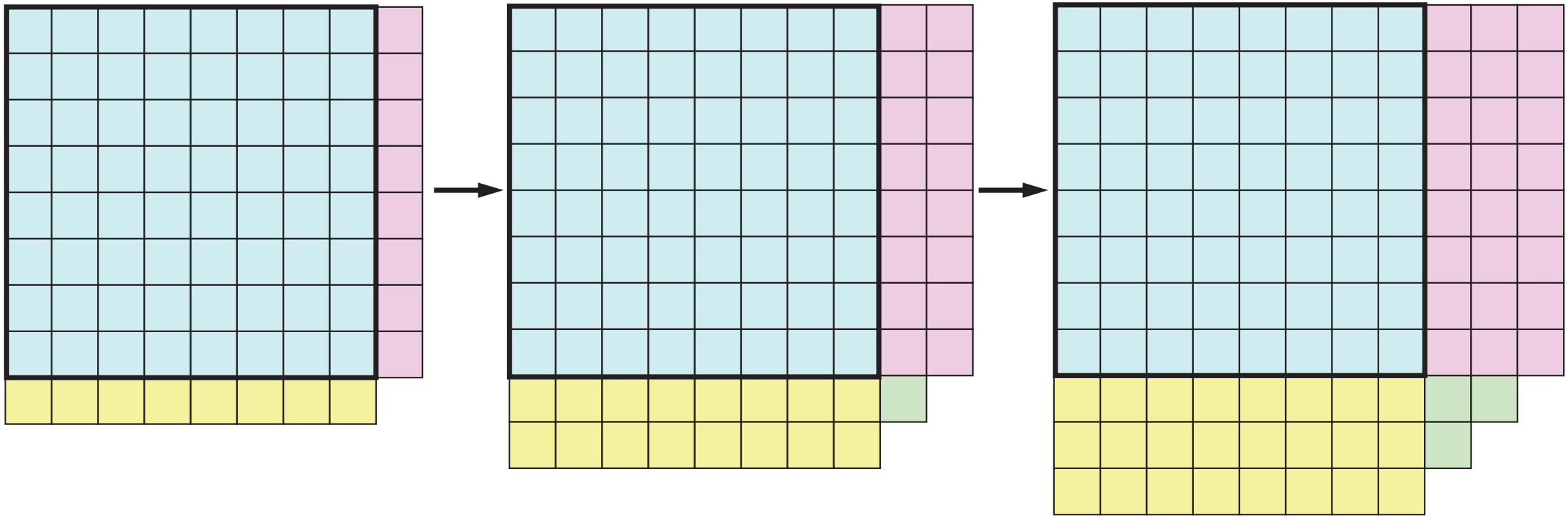


(c) PA2 example ($k = 3$): Entries of $x^{(1)}$ which need to be fetched are colored green.

Figure 2. Example for PA1 and PA2. The dotted lines define the different blocks. Each block resides on a different processor. The example shows from the perspective of the processor holding the central block.

Avoiding Communication in Sparse Matrix Computations. James Demmel, Mark Hoemmen, Marghoob Mohiyuddin, and Katherine Yelick. , 2008 IPDPS

Required Information of Local Meshes for s-step CA computations (2D 5pt.)



s=1
(original)

s=2

s=3

- Sparse Matrices
- Iterative Linear Solvers
  - Preconditioning
  - Parallel Iterative Linear Solvers
  - Multigrid Method
  - Recent Technical Issues
- Example of Parallel MGCG
- ppOpen-HPC

Key-Issues for Appl's/Algorithms towards Post-Peta & Exa Computing

Jack Dongarra (ORNL/U. Tennessee) at ISC 2013

- Hybrid/Heterogeneous Architecture
  - Multicore + GPU/Manycores (Intel MIC/Xeon Phi)
    - Data Movement, Hierarchy of Memory
- Communication/Synchronization Reducing Algorithms
- Mixed Precision Computation
- Auto-Tuning/Self-Adapting
- Fault Resilient Algorithms
- Reproducibility of Results

Motivation of This Study

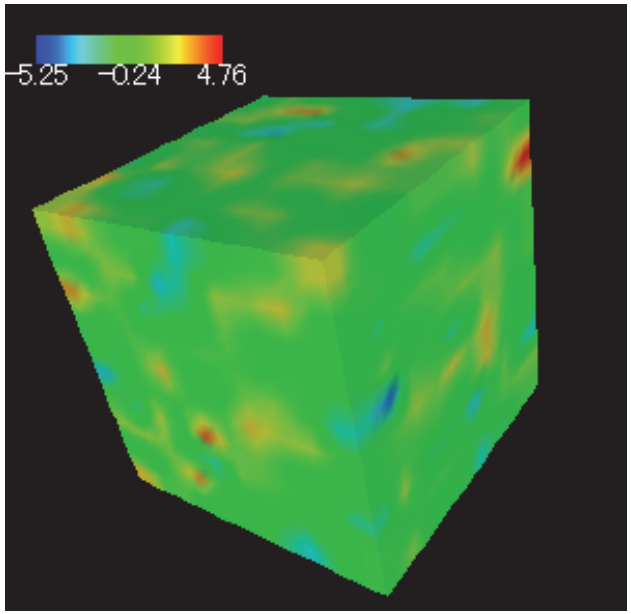
- Large-scale 3D Groundwater Flow
  - Poisson equations
  - Heterogeneous porous media
- Parallel (Geometric) Multigrid Solvers for FVM-type appl. on Fujitsu PRIMEHPC FX10 at University of Tokyo (Oakleaf-FX)
- Flat MPI vs. Hybrid (OpenMP+MPI)
- Expectations for Hybrid Parallel Programming Model
  - Number of MPI processes (and sub-domains) to be reduced
  - O(10^8-10^9)-way MPI might not scale in Exascale Systems
  - Easily extended to Heterogeneous Architectures
    - CPU+GPU, CPU+Manycores (e.g. Intel MIC/Xeon Phi)
    - MPI+X: OpenMP, OpenACC, CUDA, OpenCL

Target Application: *pGW3D-FVM*

- 3D Groundwater Flow via. Heterogeneous Porous Media
  - Poisson's equation
  - Randomly distributed water conductivity

$$\nabla \cdot \left(\lambda(x, y, z) \nabla \phi \right) = q, \phi = 0 \; at \; z = z_{\max}$$

  - Distribution of water conductivity is defined through methods in geostatistics〔Deutsch & Journel, 1998〕
- Finite-Volume Method on Cubic Voxel Mesh
- Distribution of Water Conductivity
  - 10^{-5}-10^{+5}, Condition Number ~ 10^{+10}
  - Average: 1.0
- Cyclic Distribution: 128^3

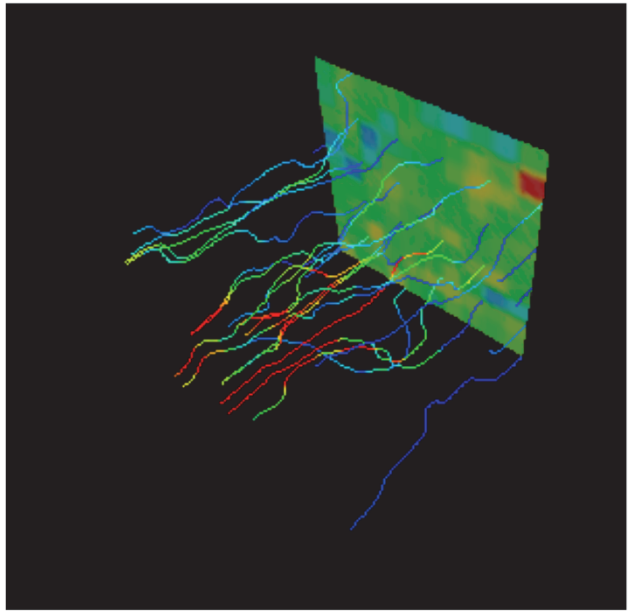


Target Application: *pGW3D-FVM*

- 3D Groundwater Flow via. Heterogeneous Porous Media
  - Poisson's equation
  - Randomly distributed water conductivity

$$\nabla \cdot \left(\lambda(x, y, z) \nabla \phi \right) = q, \phi = 0 \; at \; z = z_{\max}$$

  - Distribution of water conductivity is defined through methods in geostatistics〔Deutsch & Journel, 1998〕
- Finite-Volume Method on Cubic Voxel Mesh
- Distribution of Water Conductivity
  - 10^{-5}-10^{+5}, Condition Number ~ 10^{+10}
  - Average: 1.0
- Cyclic Distribution: 128^3



Motivation of This Study

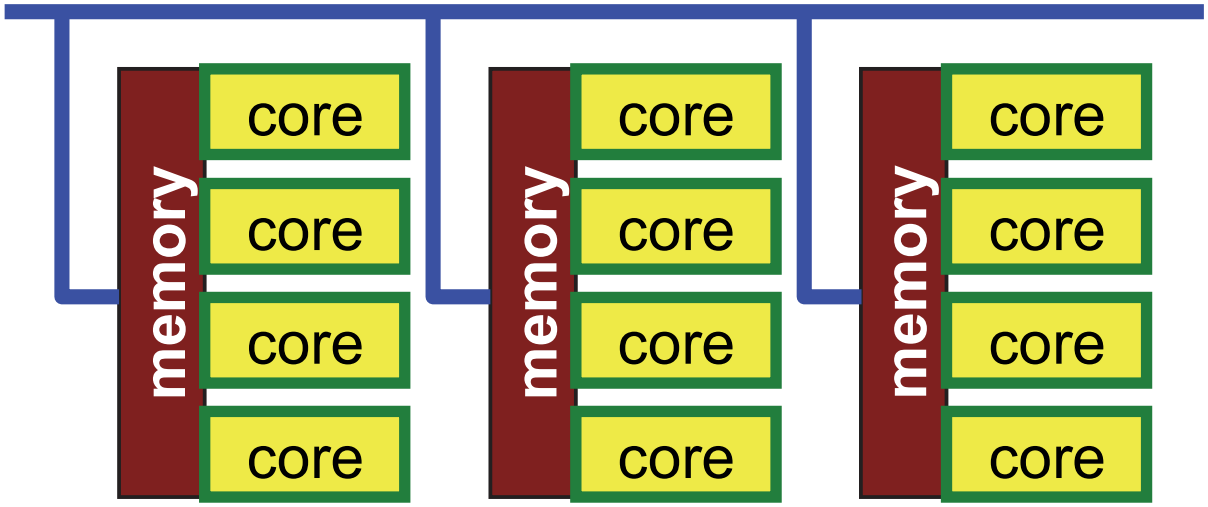
- Large-scale 3D Groundwater Flow
  - Poisson equations
  - Heterogeneous porous media
- Parallel (Geometric) Multigrid Solvers for FVM-type appl. on Fujitsu PRIMEHPC FX10 at University of Tokyo (Oakleaf-FX)
- Flat MPI vs. Hybrid (OpenMP+MPI)
- Expectations for Hybrid Parallel Programming Model
  - Number of MPI processes (and sub-domains) to be reduced
  - $O(10^8\text{-}10^9)$-way MPI might not scale in Exascale Systems
  - Easily extended to Heterogeneous Architectures
    - CPU+GPU, CPU+Manycores (e.g. Intel MIC/Xeon Phi)
    - MPI+X: OpenMP, OpenACC, CUDA, OpenCL

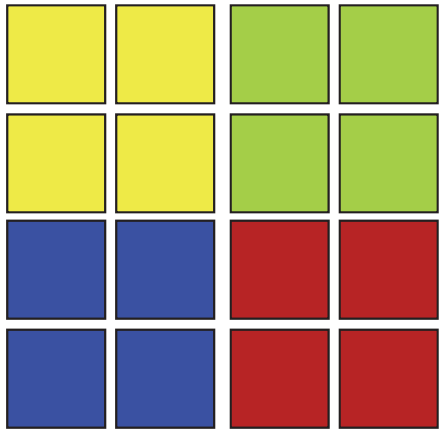
Keywords

- Parallel Geometric Multigrid
- OpenMP/MPI Hybrid Parallel Programming Model
- Localized Block Jacobi Preconditioning
  - Overlapped Additive Schwartz Domain Decomposition (ASDD)
- OpenMP Parallelization with Coloring
- Coarse Grid Aggregation (CGA), Hierarchical CGA

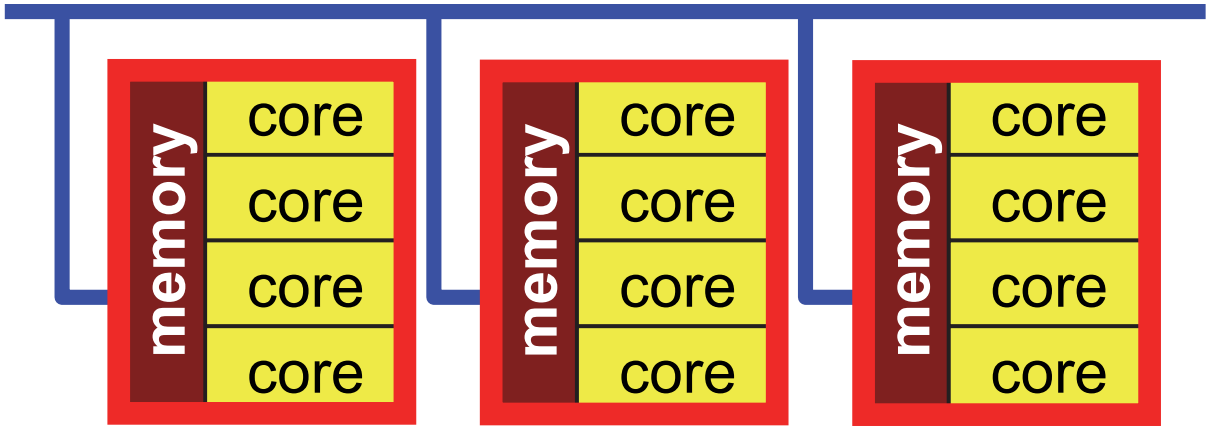
Flat MPI vs. Hybrid

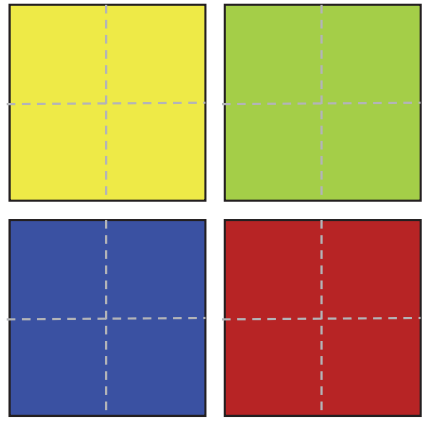
Flat-MPI : Each Core -> Independent





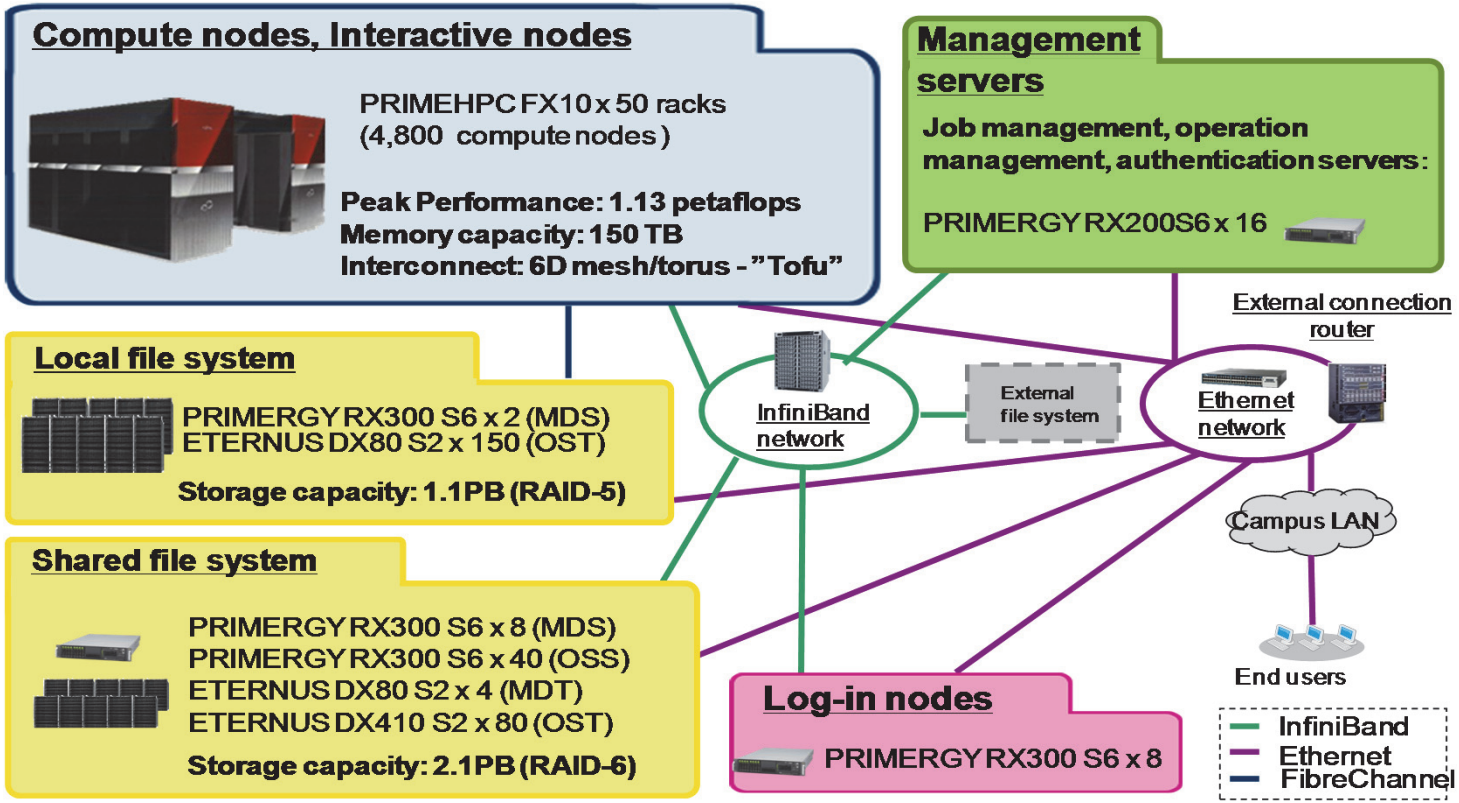
Hybrid : Hierarchal Structure





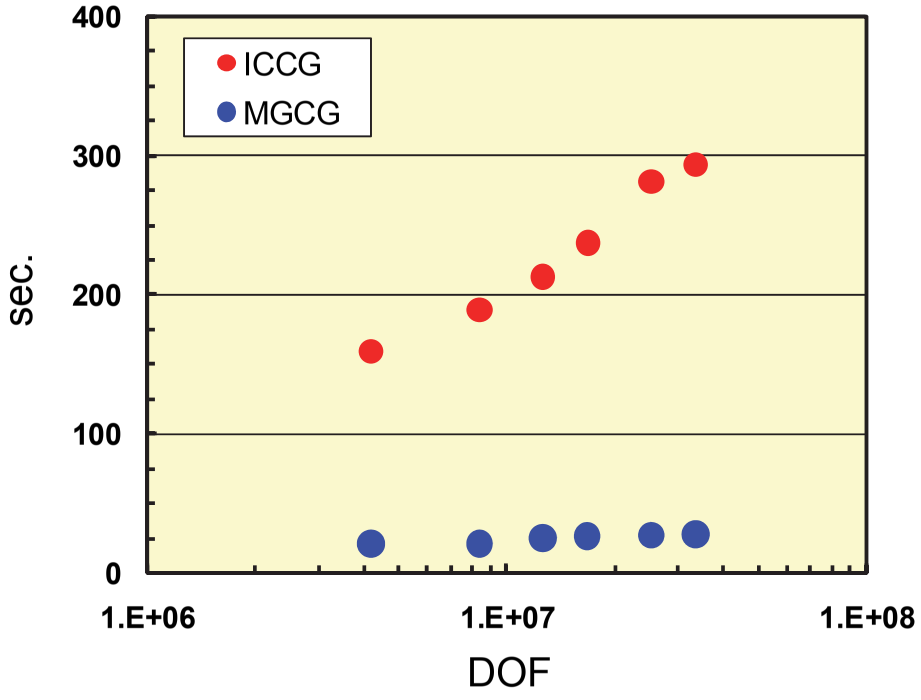
Fujitsu PRIMEHPC FX10 (Oakleaf-FX) at the U. Tokyo

- SPARC64 Ixfx (4,800 nodes, 76,800 cores)
- Commercial version of K computerx
- Peak: 1.13 PFLOPS (1.043 PF, 26th, 41th TOP 500 in 2013 June.)
- Memory BWTH 398 TB/sec.



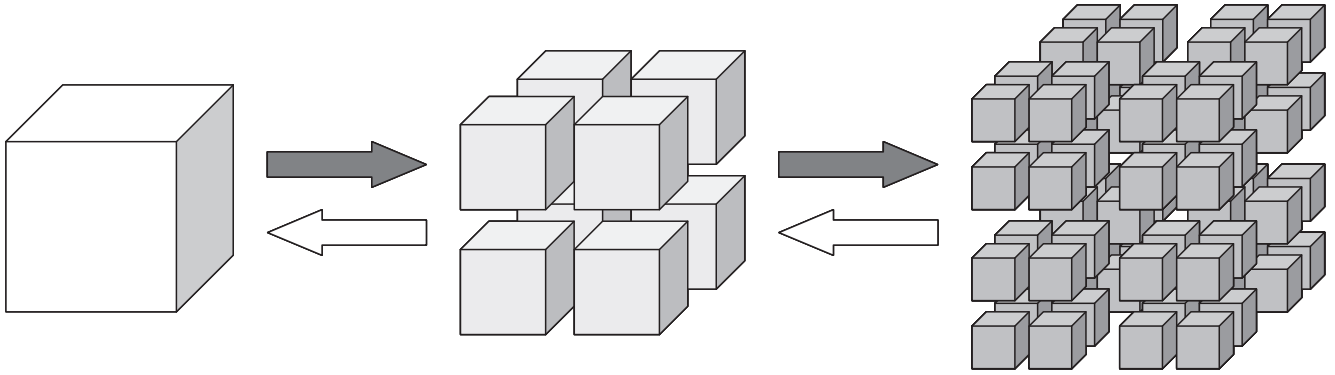
Multigrid

- Scalable Multi-Level Method using Multilevel Grid for Solving Linear Eqn's
  - Computation Time ~ O(N) (N: # unknowns)
  - Good for large-scale problems
- Preconditioner for Krylov Iterative Linear Solvers
  - MGCG



Linear Solvers

- Preconditioned CG Method
  - Multigrid Preconditioning (MGCG)
  - IC(0) for Smoothing Operator (Smoother): good for ill-conditioned problems
- Parallel Geometric Multigrid Method
  - 8 fine meshes (children) form 1 coarse mesh (parent) in isotropic manner (octree)
  - V-cycle
  - Domain-Decomposition-based: Localized Block-Jacobi, Overlapped Additive Schwartz Domain Decomposition (ASDD)
  - Operations using a single core at the coarsest level (redundant)

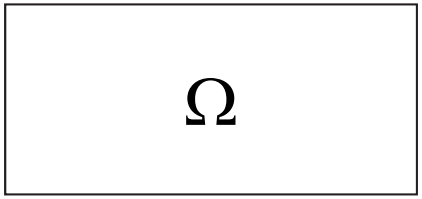


Overlapped Additive Schwartz Domain Decomposition Method

ASDD: Localized Block-Jacobi Precond. is stabilized

Global Operation

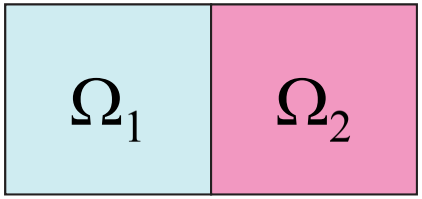
$$Mz = r$$



Local Operation

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

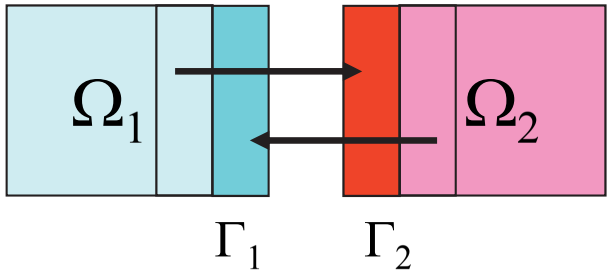
Ω_i : Internal (i≦N)
Γ_i : External (i>N)



Global Nesting Correction

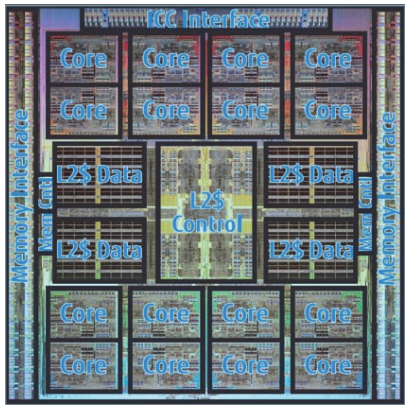
$$z_{\Omega_1}^{n} = z_{\Omega_1}^{n-1} + M_{\Omega_1}^{-1}(r_{\Omega_1} - M_{\Omega_1} z_{\Omega_1}^{n-1} - M_{\Gamma_1} z_{\Gamma_1}^{n-1})$$

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

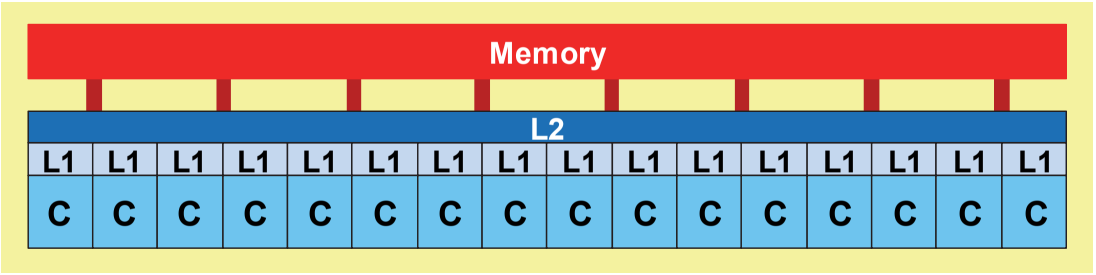


Computations on Fujitsu FX10

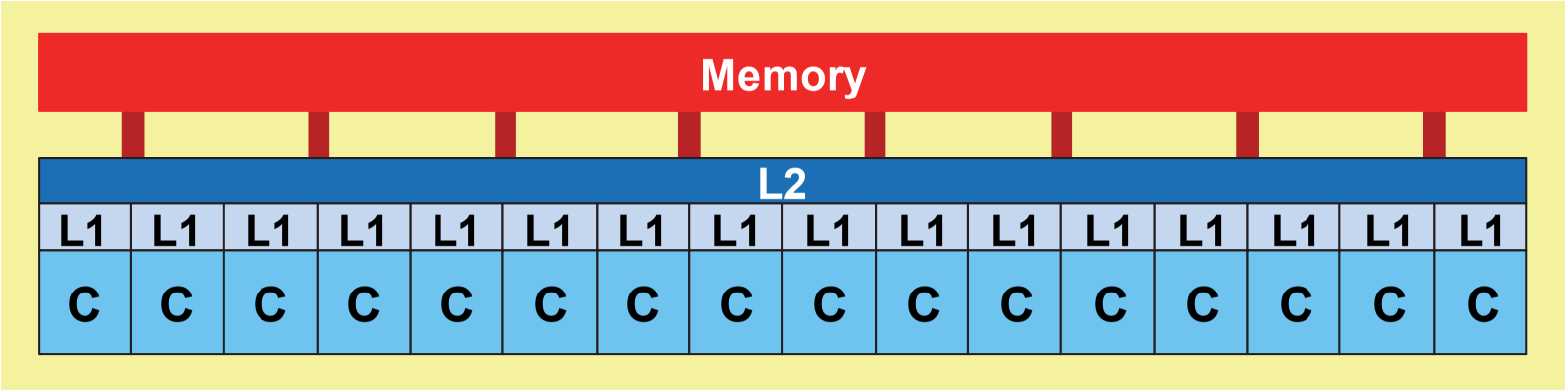
- Fujitsu PRIMEHPC FX10 at U.Tokyo (Oakleaf-FX)
  - 16 cores/node, flat/uniform access to memory
- Up to 4,096 nodes (65,536 cores) (Large-Scale HPC Challenge)
  - Max 17,179,869,184 unknowns
  - Flat MPI, HB 4x4, HB 8x2, HB 16x1
    - HB MxN: M-threads x N-MPI-processes on each node



- Weak Scaling
  - 64^3 cells/core
- Strong Scaling
  - $128^3 \times 8 = 16{,}777{,}216$ unknowns, from 8 to 4,096 nodes



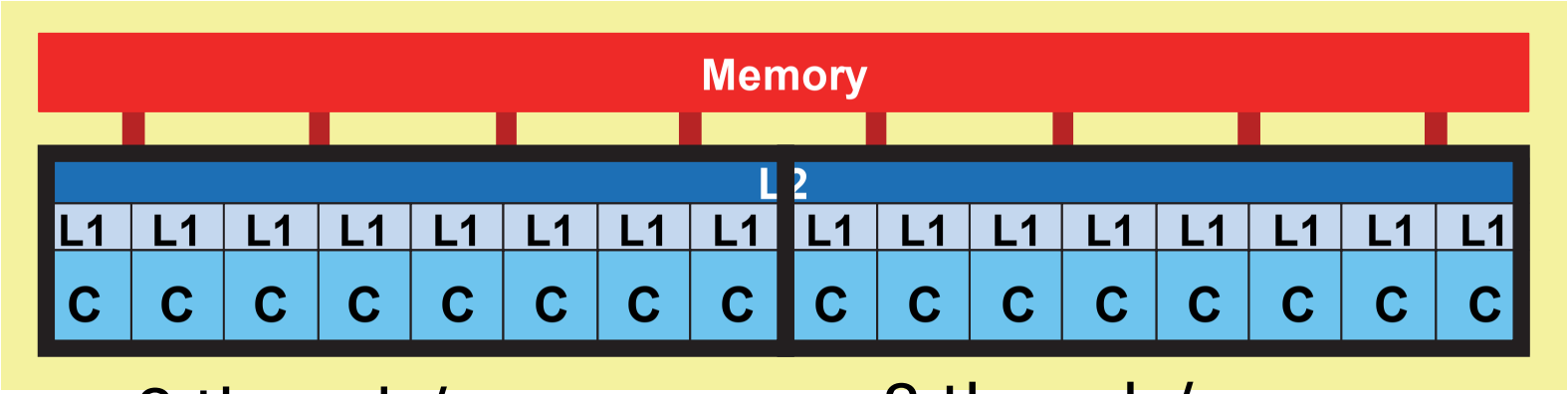
- Network Topology is not specified
  - 1D



HB M x N

Number of OpenMP threads per a single MPI process

Number of MPI process per a single node



8 threads/process 8 threads/process

HB 8 x 2

Number of OpenMP threads per a single MPI process

Number of MPI process per a single node

Reordering for extracting parallelism in each domain (= MPI Process)

- Krylov Iterative Solvers
  - Dot Products
  - SMVP
  - DAXPY
  - Preconditioning
- IC/ILU Factorization, Forward/Backward Substitution
  - Global Data Dependency
  - Reordering needed for parallelism ([KN 2003] on the Earth Simulator, KN@CMCIM-2002)
  - Multicoloring, RCM, CM-RCM

Parallerization of ICCG

IC Factorization

```
do i= 1, N
  VAL= D(i)
  do k= indexL(i-1)+1, indexL(i)
    VAL= VAL - (AL(k)**2) * W(itemL(k),DD)
  enddo
  W(i,DD)= 1.d0/VAL
enddo
```

Forward Substitution

```
do i= 1, N
  WVAL= W(i,Z)
  do k= indexL(i-1)+1, indexL(i)
    WVAL= WVAL -  AL(k) * W(itemL(k),Z)
  enddo
  W(i,Z)= WVAL * W(i,DD)
enddo
```

(Global) Data Dependency:

Writing/reading may occur simultaneously, hard to parallelize

IC Factorization

```
do i= 1, N
  VAL= D(i)
  do k= indexL(i-1)+1, indexL(i)
    VAL= VAL - (AL(k)**2) * W(itemL(k),DD)
  enddo
  W(i,DD)= 1.d0/VAL
enddo
```

Forward Substitution

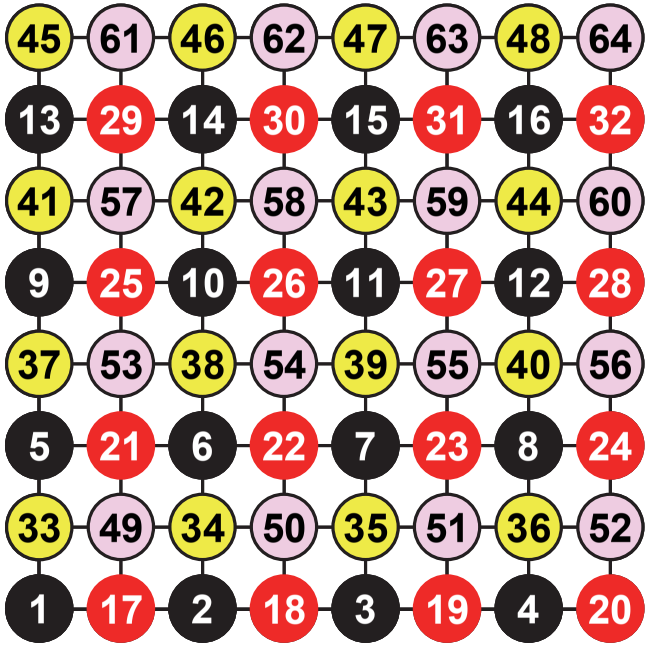
```
do i= 1, N
  WVAL= W(i,Z)
  do k= indexL(i-1)+1, indexL(i)
    WVAL= WVAL -  AL(k) * W(itemL(k),Z)
  enddo
  W(i,Z)= WVAL * W(i,DD)
enddo
```

OpenMP for SpMV: Straightforward NO data dependency

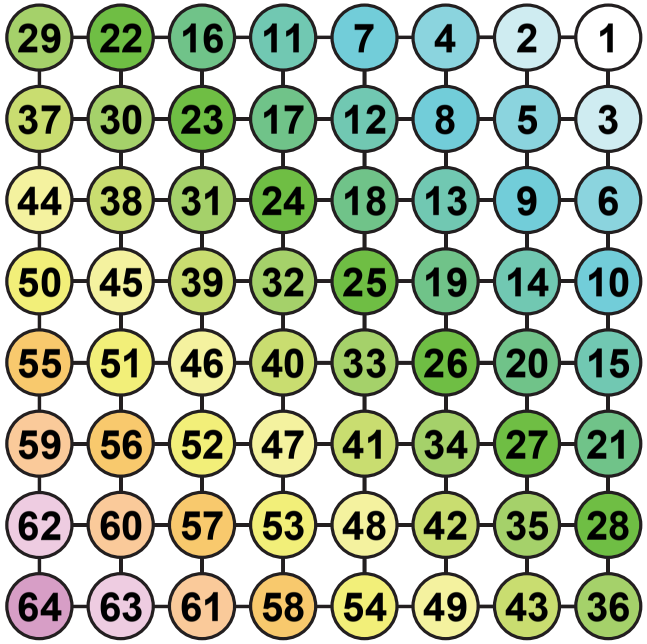
```
!$omp parallel do private(ip,i,VAL,k)
      do ip= 1, PEsmpTOT
        do i = INDEX(ip-1)+1, INDEX(ip)
          VAL= D(i)*W(i,P)
          do k= indexL(i-1)+1, indexL(i)
            VAL= VAL + AL(k)*W(itemL(k),P)
          enddo
          do k= indexU(i-1)+1, indexU(i)
            VAL= VAL + AU(k)*W(itemU(k),P)
          enddo
          W(i,Q)= VAL
        enddo
      enddo
```

Ordering Methods

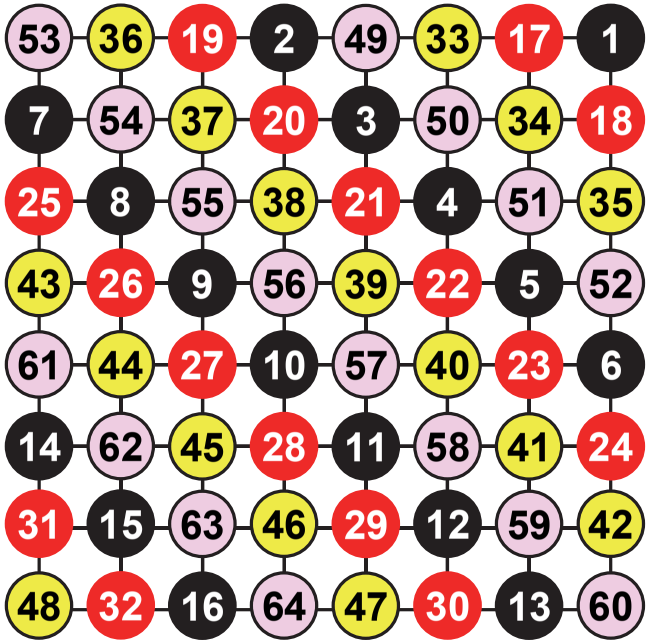
Elements in “same color” are independent: to be parallelized



MC (Color#=4)
Multicoloring



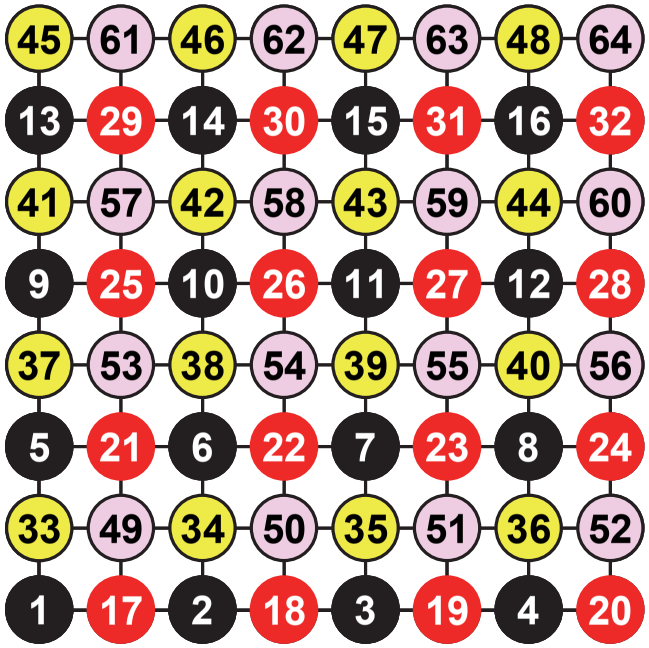
RCM
Reverse Cuthill-Mckee



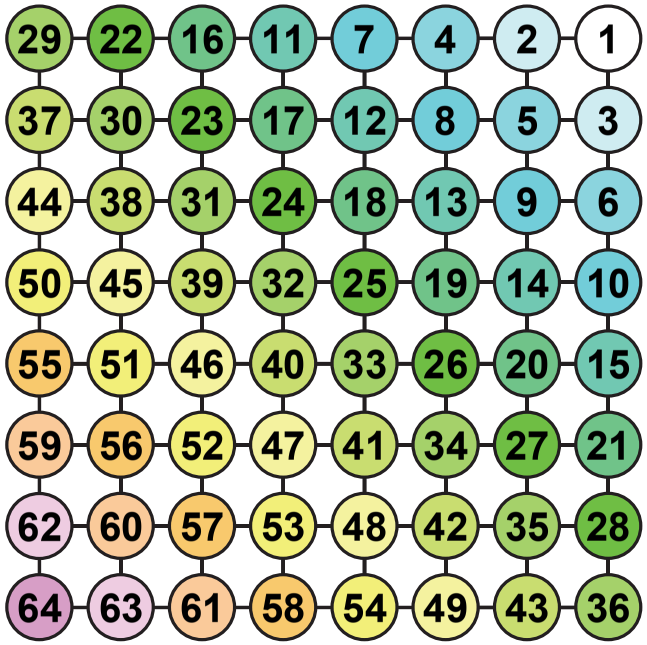
CM-RCM (Color#=4)
Cyclic MC + RCM

Ordering Methods

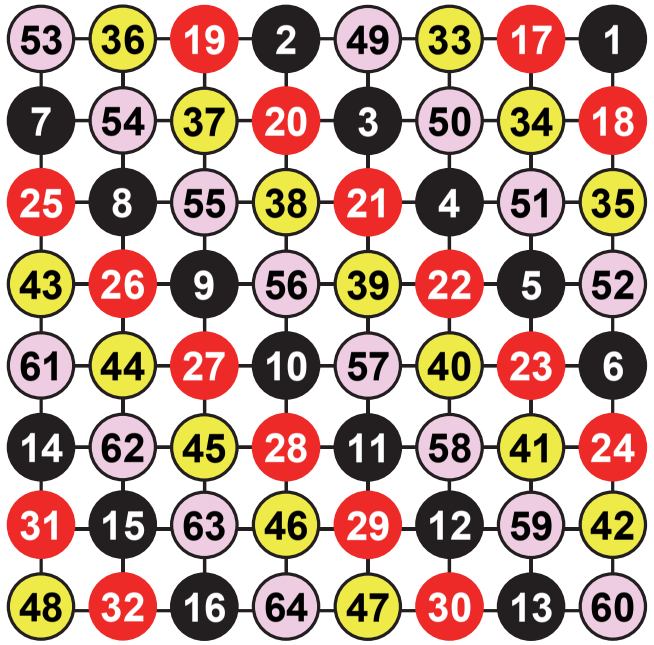
Elements in “same color” are independent: to be parallelized



MC (Color#=4)
Multicoloring



RCM
Reverse Cuthill-Mckee



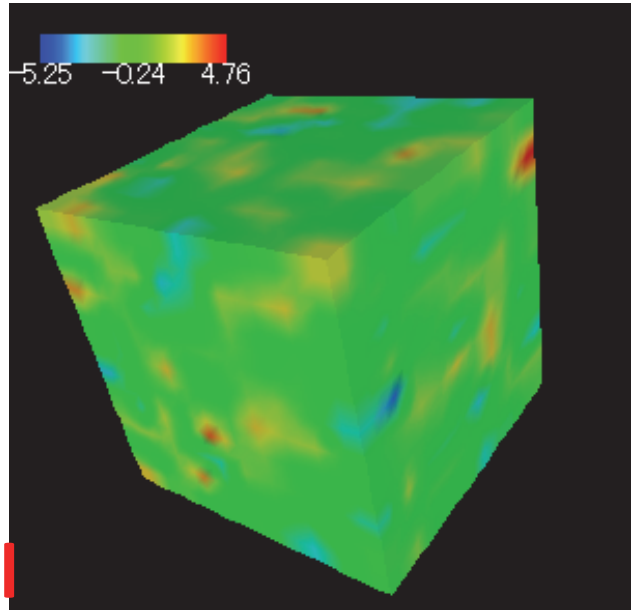
CM-RCM (Color#=4)
Cyclic MC + RCM

Recent Progress (2013-2014)

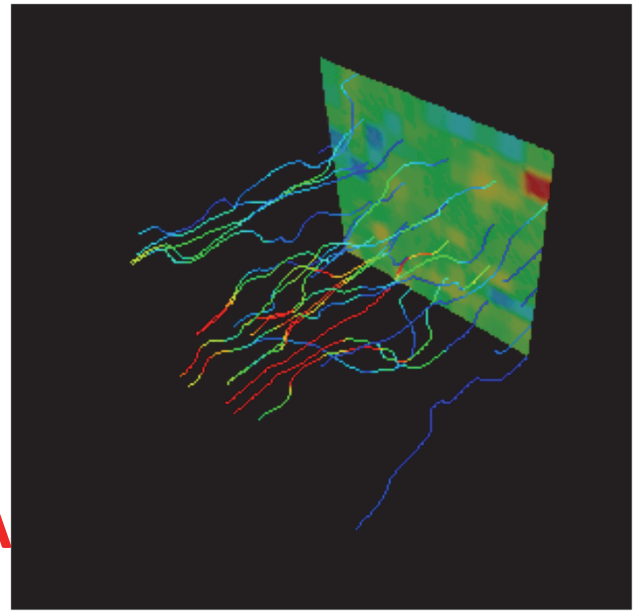
- Optimization of Parallel MGCG
  - Conjugate Gradient Solver with Multigrid Preconditioning
  - OpenMP/MPI Hybrid Parallel Programming Model
  - Efficiency & Convergence
- Communications are expensive
  - Serial Communications
    - Data Transfer through Hierarchical Memory
  - Parallel Communications
    - Message Passing through Network
- Parallel Multigrid
  - "Coarse Grid Solver" is important
    - Efficiency & Convergence
  - HPCG (High-Performance Conjugate Gradients)
    - MGCG by Geometric Multigrid

Parallel MG Solvers: pGW3D-FVM

- 3D Groundwater Flow via Heterogeneous Porous Media
  - Poisson's equation $\nabla \cdot (\lambda(x,y,z)\nabla\phi) = q$
  - Randomly distributed water conductivity
  - Finite-Volume Method on Cubic Voxel Mesh
  - $\lambda=10^{-5}\sim10^{+5}$, Average: 1.00
  - MGCG Solver



- **Storage format of coefficient matrices (Serial Comm.)**
  - **CRS (Compressed Row Storage)**
  - **ELL (Ellpack-Itpack)**
- **Comm. /Sych. Reducing MG (Parallel Comm.)**
  - **Coarse Grid Aggregation (CGA)**
  - **Hierarchical CGA: Communication Reducing CGA**



ELL: Fixed Loop-length, Nice for Pre-fetching

$$\begin{bmatrix} 1 & 3 & 0 & 0 & 0 \\ 1 & 2 & 5 & 0 & 0 \\ 4 & 1 & 3 & 0 & 0 \\ 0 & 3 & 7 & 4 & 0 \\ 1 & 0 & 0 & 0 & 5 \end{bmatrix}$$

1	3	
1	2	5
4	1	3
3	7	4
1	5	

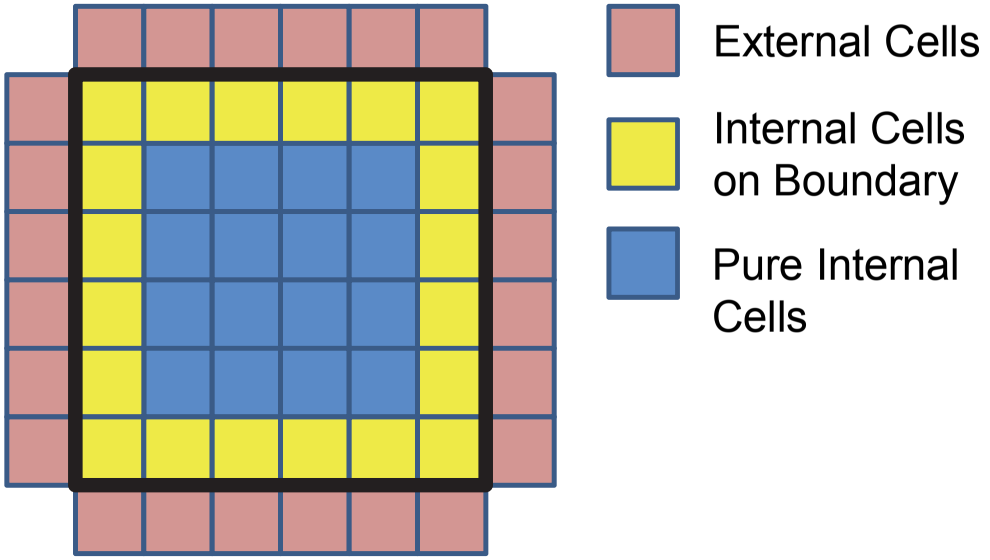
(a) CRS

1	3	0
1	2	5
4	1	3
3	7	4
1	5	0

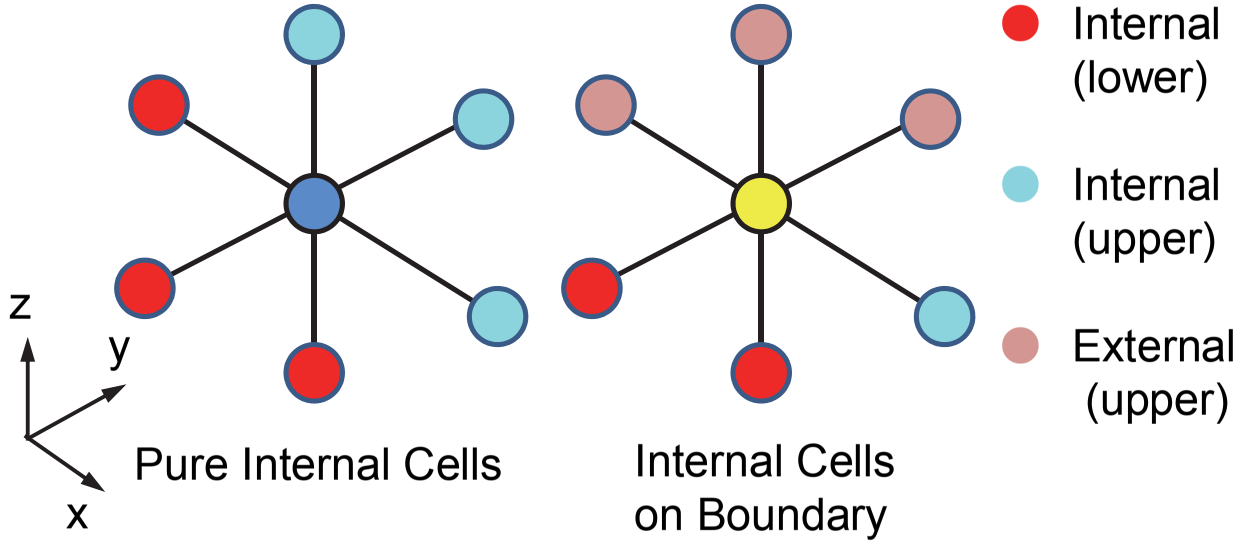
(b) ELL

Special Treatment for "Boundary" Cells connected to "Halo"

- Distribution of Lower/Upper Non-Zero Off-Diagonal Components



- If we adopt RCM (or CM) reordering ...
- Pure Internal Cells
  - L: ~3, U: ~3
- Boundary Cells
  - L: ~3, U: ~6



Original ELL: Backward Subst.

Cache is not well-utilized: IAUnew(6,N), Aunew(6,N)

```
      do icol= NHYP(lev), 1, -1
        if (mod(icol,2).eq.1) then
!$omp parallel do private (ip,icel,j,SW)
          do ip= 1, PEsmpTOT
          do icel= SMPindex(icol-1,ip,lev)+1, SMPindex(icol,ip,lev)
            SW= 0.0d0
            do j= 1, 6
              SW= SW + AUnew(j,icel)*Rmg(IAUnew(j,icel))
            enddo
            Rmg(icel)= Rmg(icel) - SW*DDmg(icel)
          enddo
          enddo
        else
!$omp parallel do private (ip,icel,j,SW)
          do ip= 1, PEsmpTOT
          do icel= SMPindex(icol-1,ip,lev)+1, SMPindex(icol,ip,lev)
            SW= 0.0d0
            do j= 1, 3
              SW= SW + AUnew(j,icel)*Rmg(IAUnew(j,icel))
            enddo
            Rmg(icel)= Rmg(icel) - SW*DDmg(icel)
          enddo
          enddo
        endif
      enddo
```

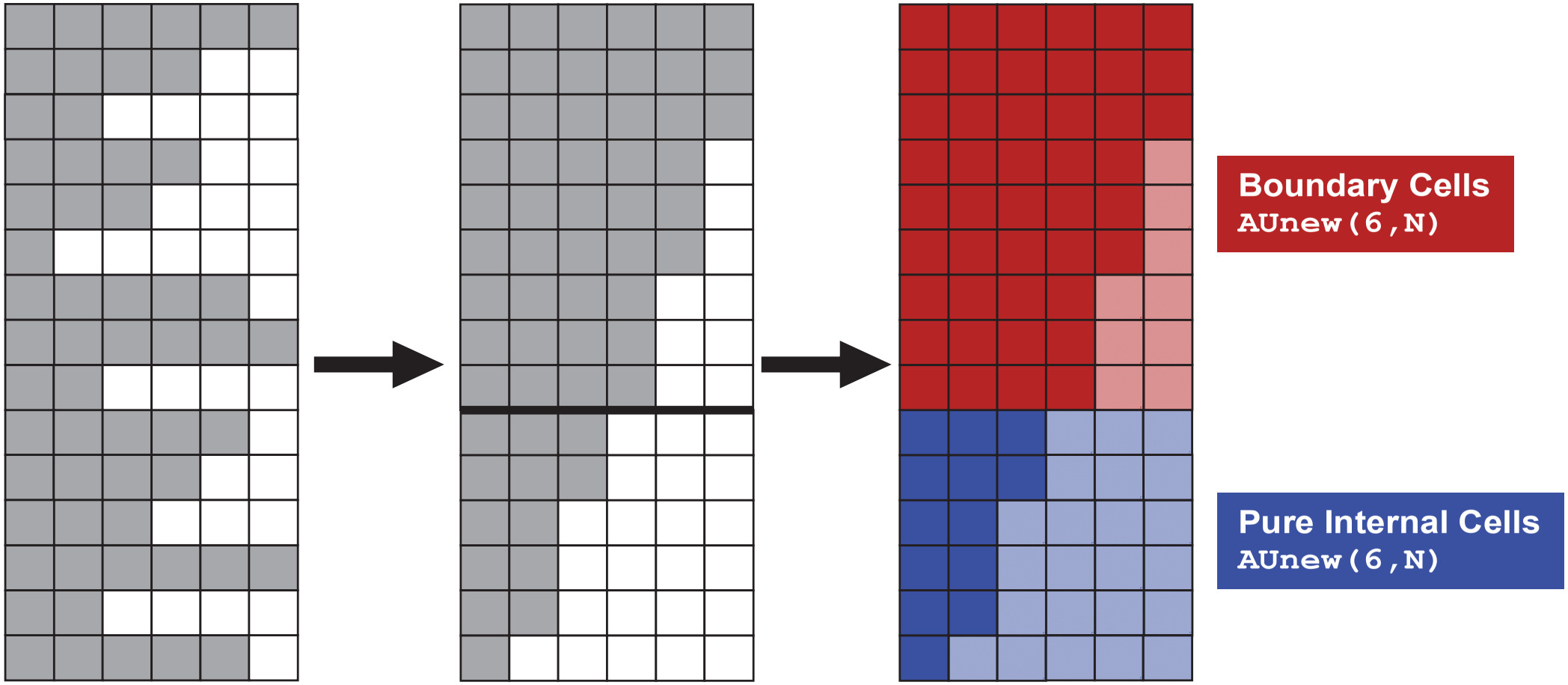
for Boundary Cells

for Pure Internal Cells

`IAUnew (6,N), AUnew (6,N)`

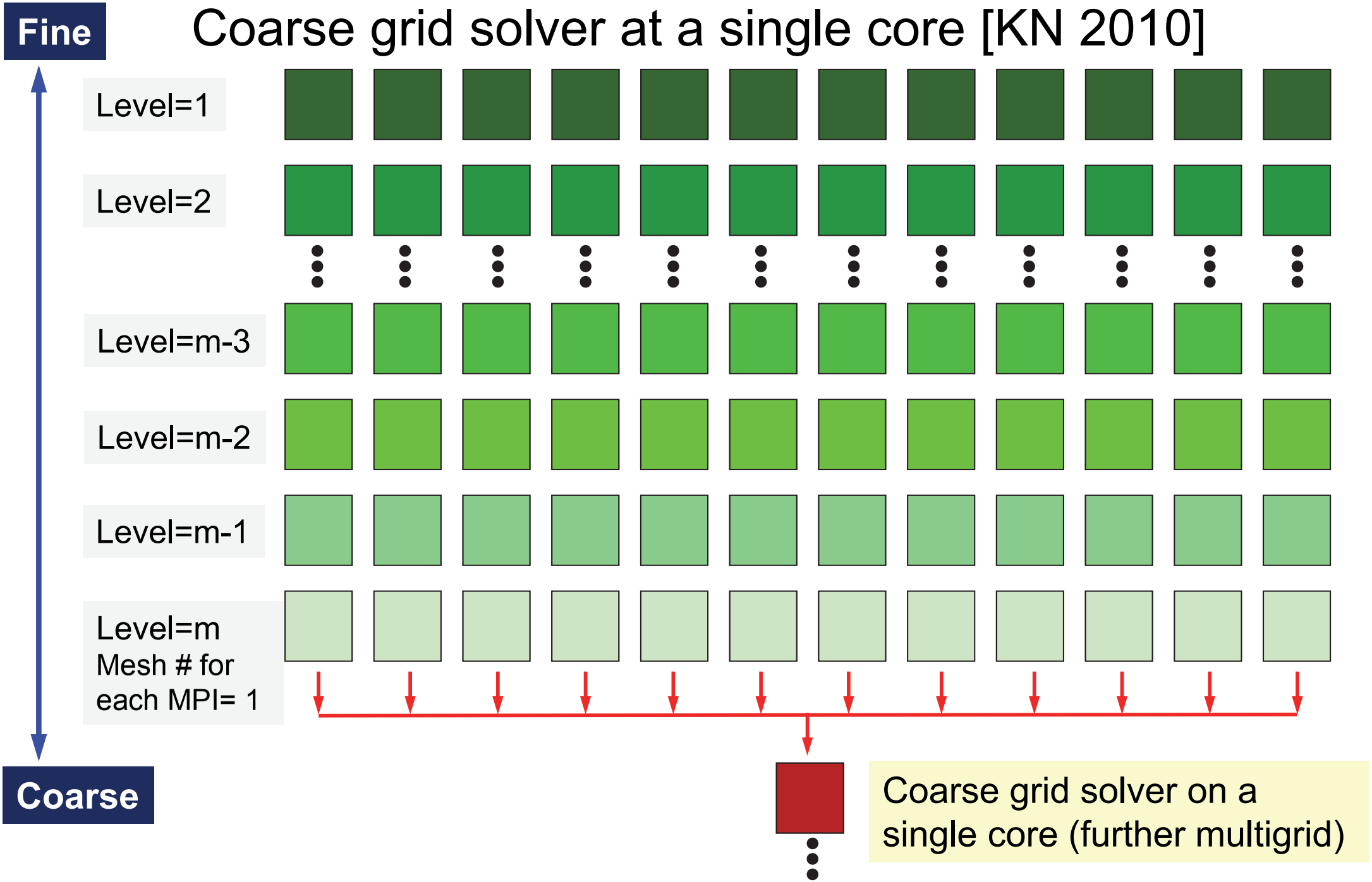
Original ELL: Backward Subst.

Cache is not well-utilized: IAUnew(6,N), Aunew(6,N)



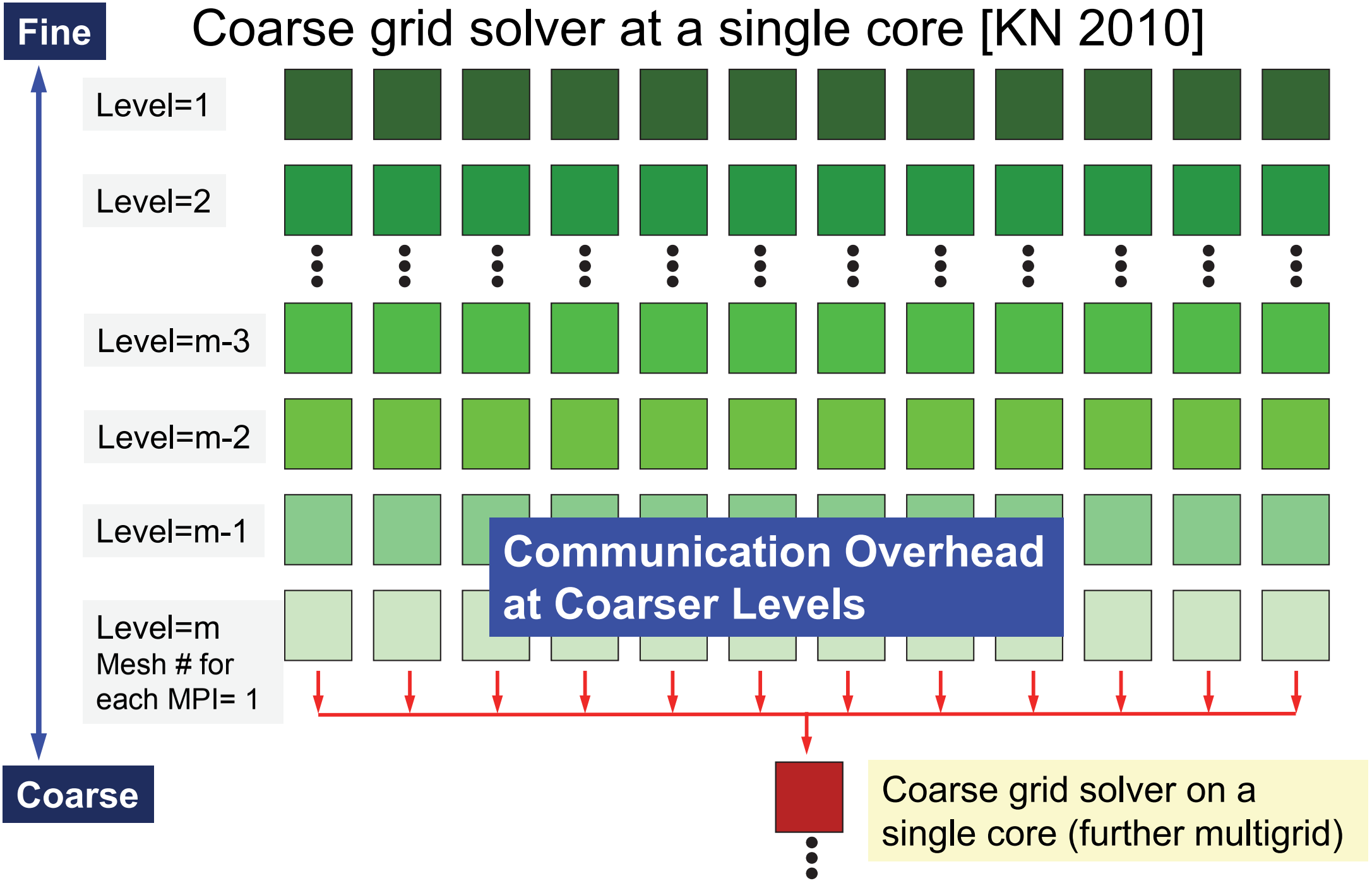
Original Approach (restriction)

Coarse grid solver at a single core [KN 2010]



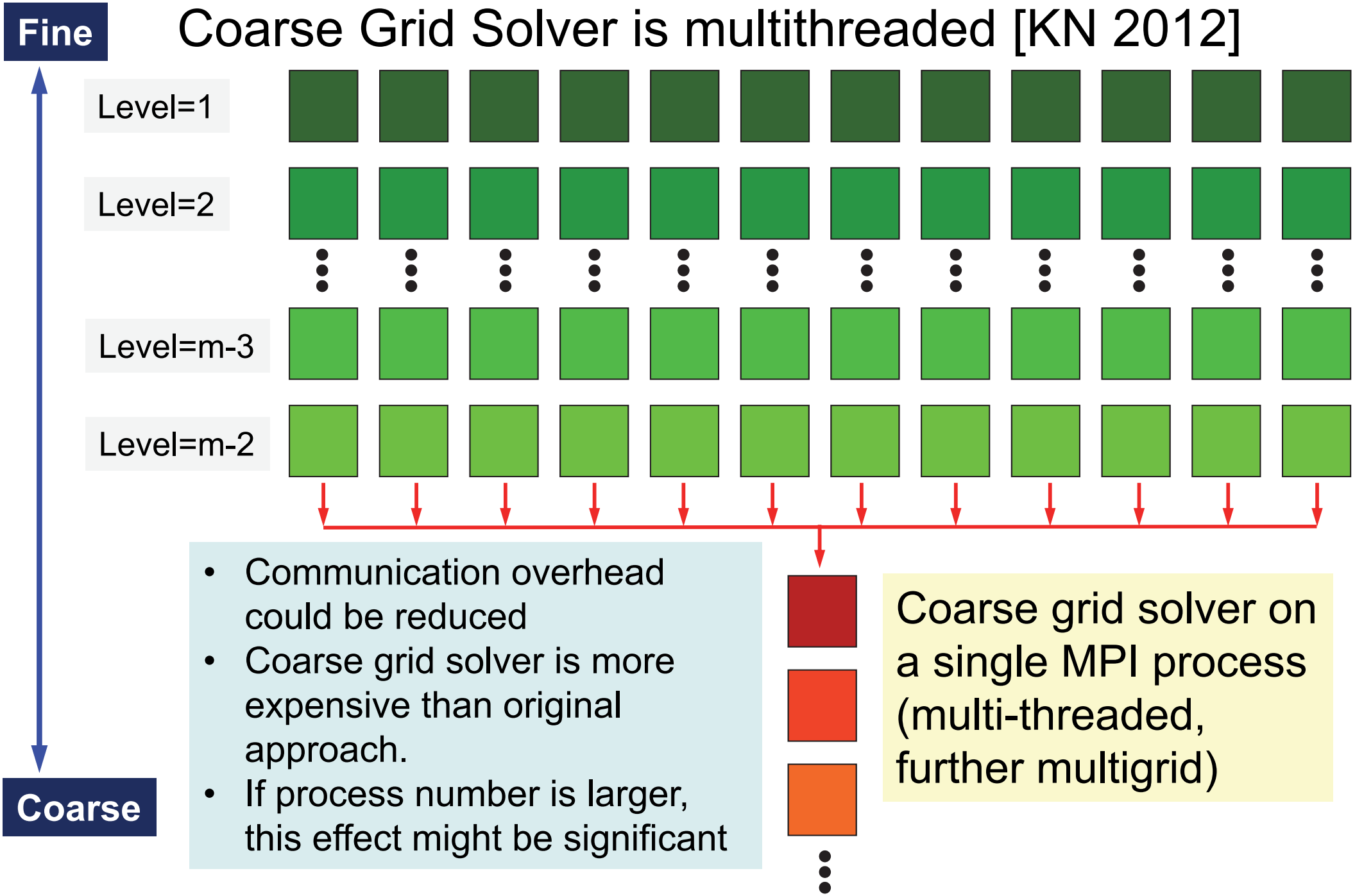
Original Approach (restriction)

Coarse grid solver at a single core [KN 2010]



Coarse Grid Aggregation (CGA)

Coarse Grid Solver is multithreaded [KN 2012]



- Communication overhead could be reduced
- Coarse grid solver is more expensive than original approach.
- If process number is larger, this effect might be significant

Coarse grid solver on a single MPI process (multi-threaded, further multigrid)

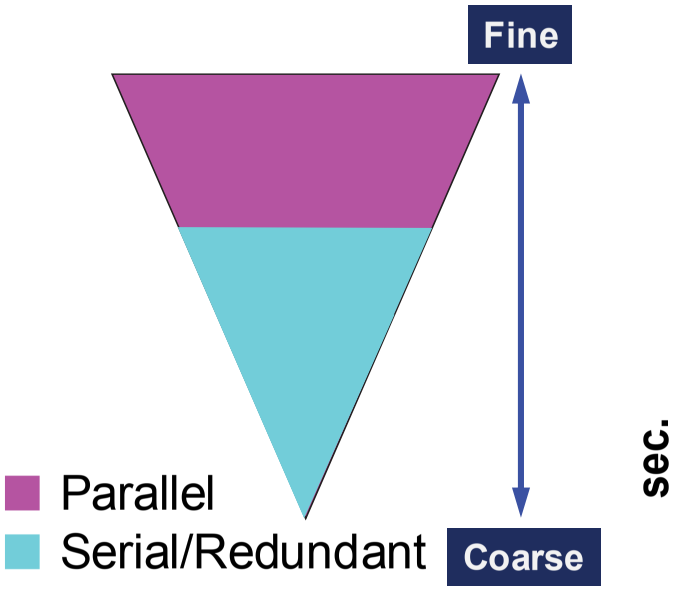
Results

CASE	Matrix	Coarse Grid
C0	CRS	Single Core
C1	ELL (original)	Single Core
C2	ELL (original)	CGA
C3	ELL (new)	CGA
C4	ELL (new)	hCGA

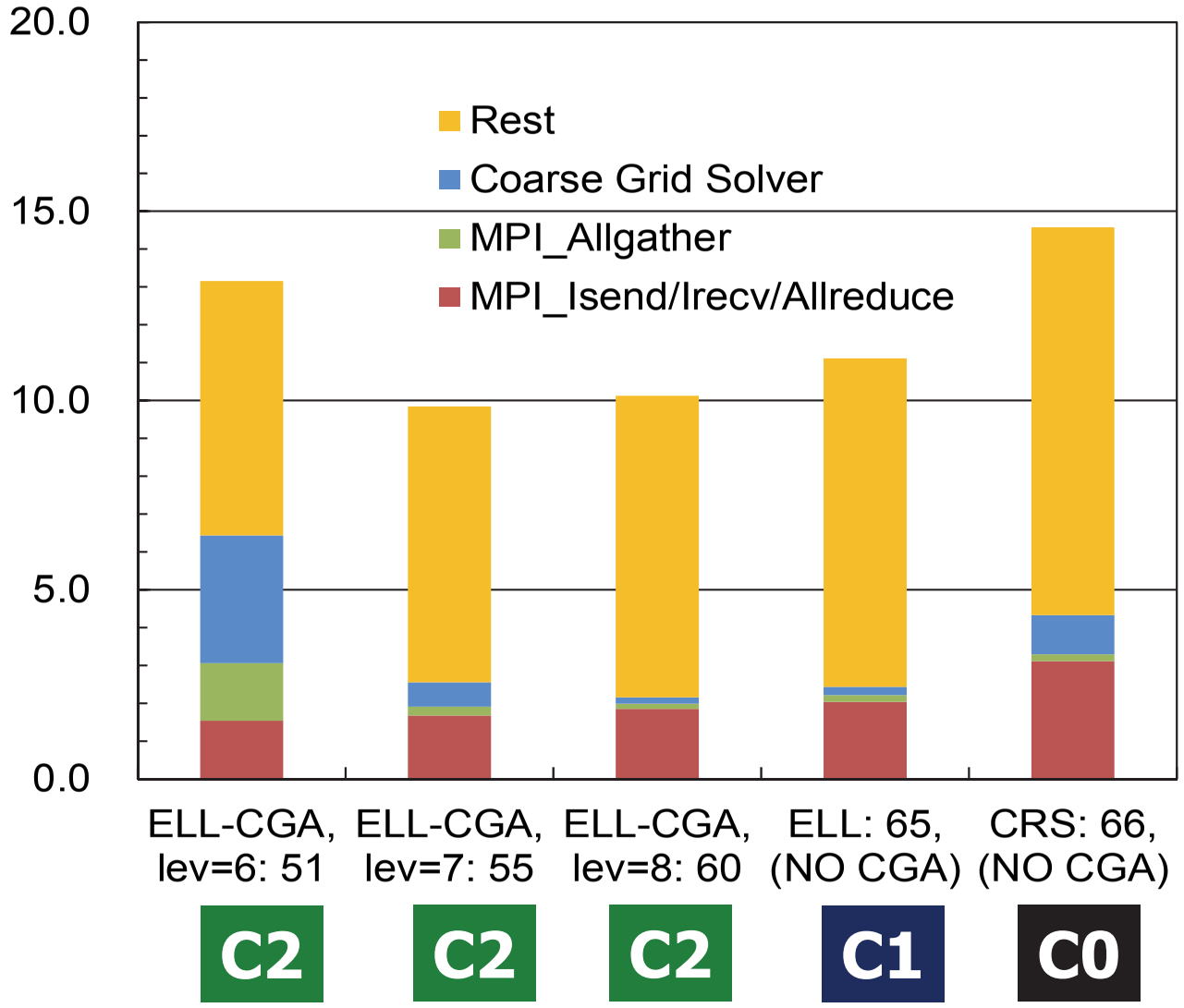
Class	Size	
Weak Scaling	64^3 cells/core	262,144
Strong Scaling	256^3 cells	16,777,216

Results at 4,096 nodes (1.72×10^{10} DOF) (Fujitsu FX10: Oakleaf-FX): HB 8x2

lev: switching level to “coarse grid solver”, Opt. Level= 7

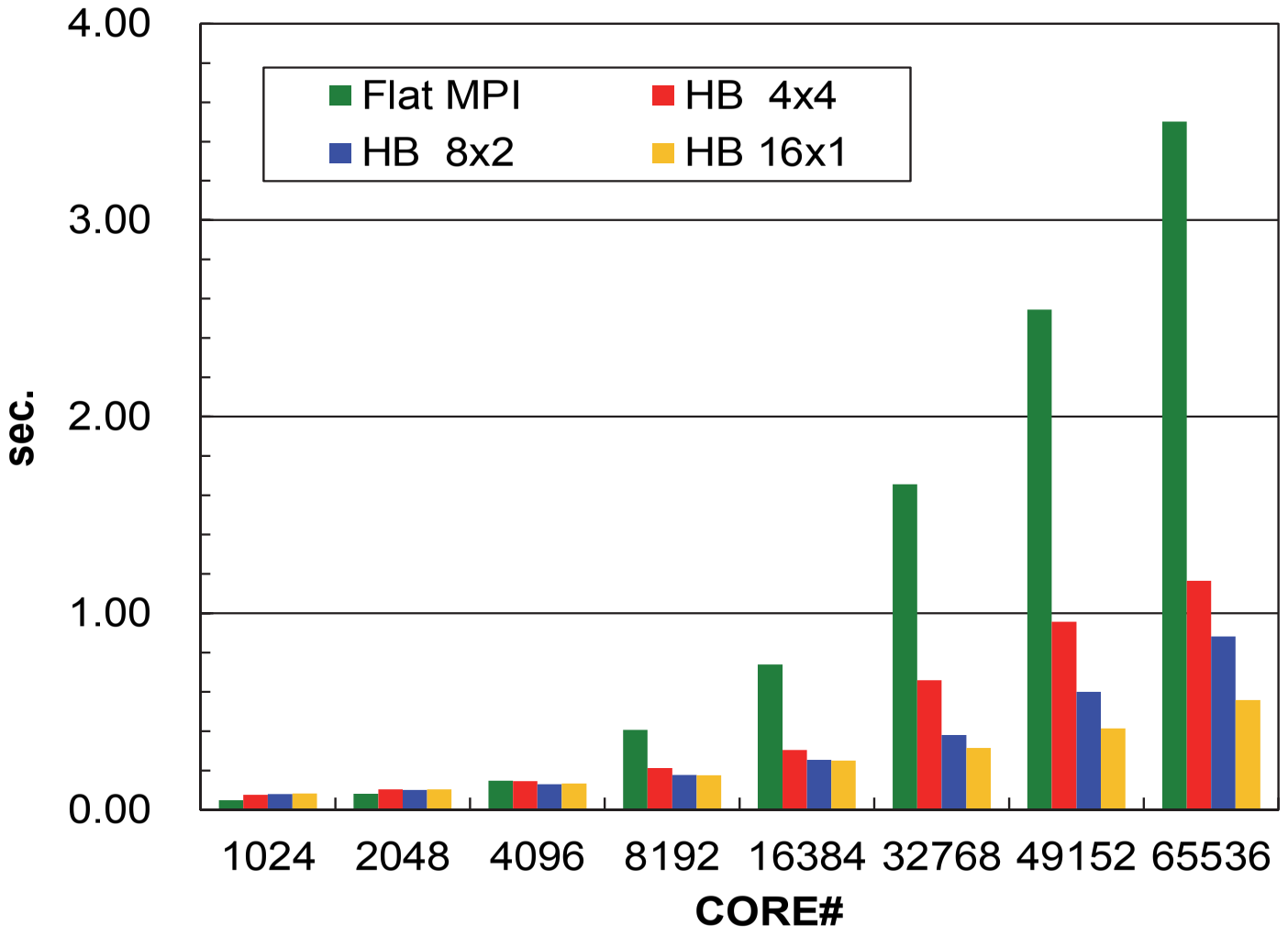


	Matrix	Coarse Grid
C0	CRS	Single Core
C1	ELL (org)	Single Core
C2	ELL (org)	CGA
C3	ELL (sliced)	CGA



Weak Scaling: C2 (with CGA)
Time for Coarse Grid Solver

Efficiency of coarse grid solver for HB 16x1 is x256 of that of flat MPI
(1/16 problem size, x16 resource for coarse grid solver)



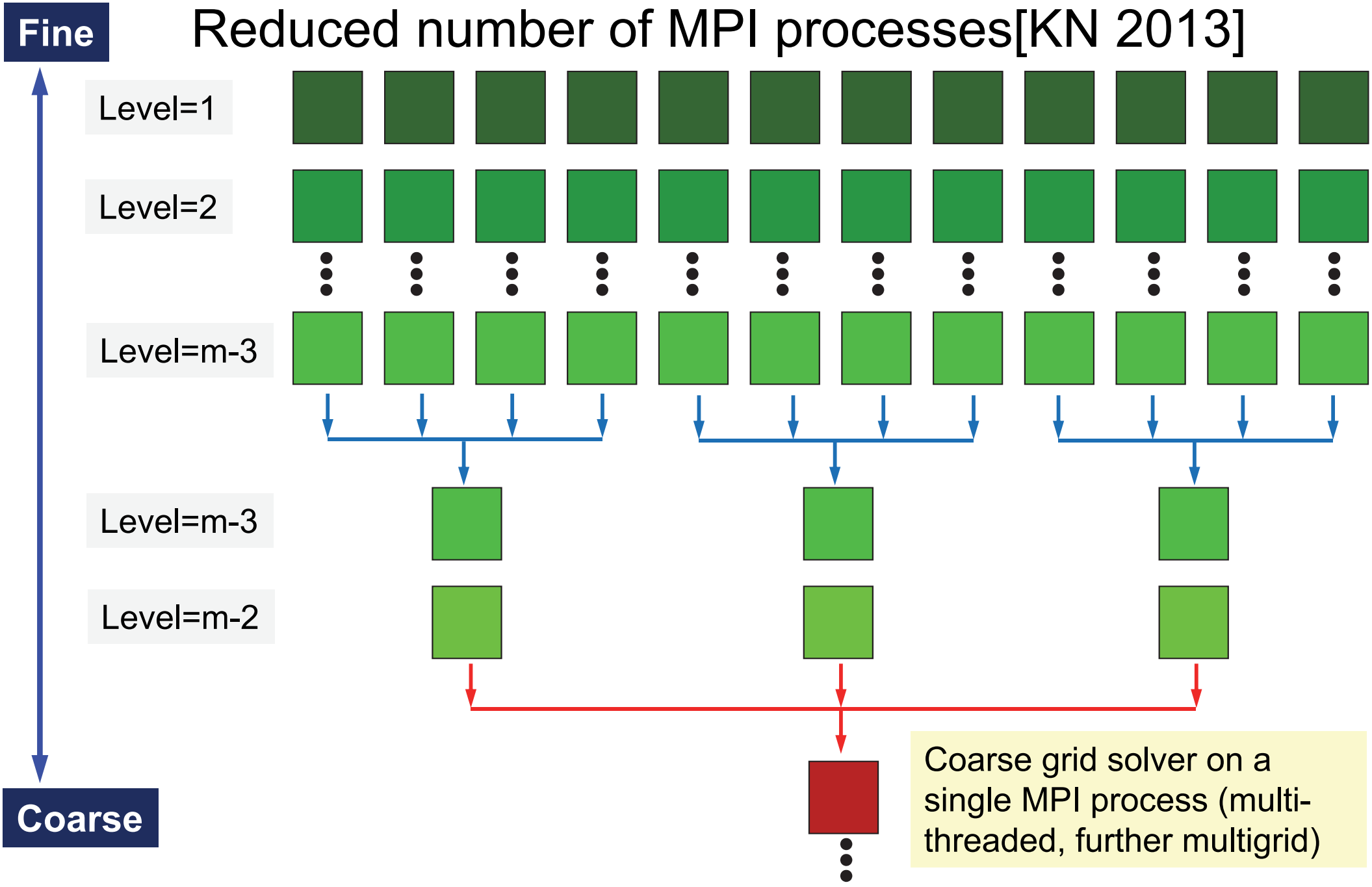
Summary so far ...

- "Coarse Grid Aggregation (CGA)" is effective for stabilization of convergence at O(10^4) cores for MGCG
  - Smaller number of parallel domains
  - HB 8x2 is the best at 4,096 nodes
  - Flat MPI, HB 4x4
    - Coarse grid solvers are more expensive, because their number of MPI processes are more than those of HB 8x2 and HB 16x1.
- ELL format is effective !
  - C0 (CRS) -> C1 (ELL-org.): +20-30%
  - C2 (ELL-org)-> C3(ELL-new): +20-30%
  - C0 -> C3: +80-90%
- Coarse Grid Solver
  - (May be) very expensive for cases with more than O(10^5) cores
  - Memory of a single node is not enough
  - Multiple nodes should be utilized for coarse grid solver

	Matrix	Coarse Grid
C0	CRS	Single Core
C1	ELL (org)	Single Core
C2	ELL (org)	CGA
C3	ELL (sliced)	CGA

Hierarchical CGA: Comm. Reducing MG

Reduced number of MPI processes[KN 2013]

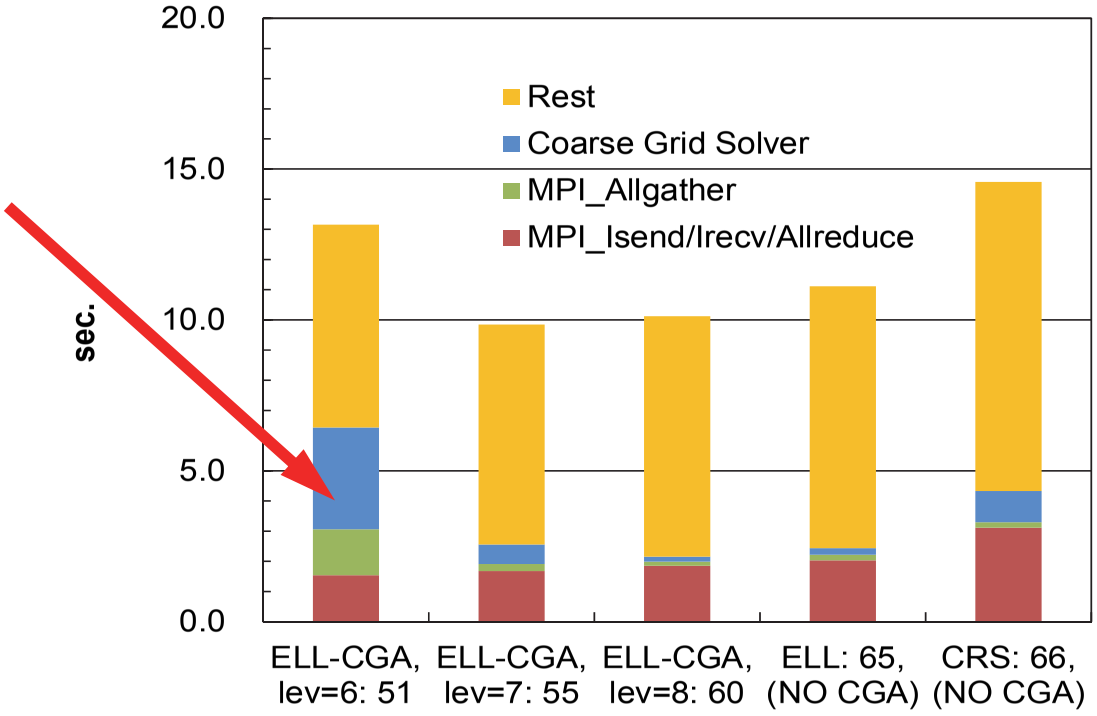


*h*CGA: Related Work

- Not a new idea, but very few implementations.
  - Not effective for peta-scale systems (Dr. U.M.Yang (LLNL), developer of Hypre)
- Existing Works: Repartitioning at Coarse Levels
  - Lin, P.T., Improving multigrid performance for unstructured mesh drift-diffusion simulations on 147,000 cores, International Journal for Numerical Methods in Engineering 91 (2012) 971-989 (Sandia)
  - Sundar, H. et al, Parallel Geometric-Algebraic Multigrid on Unstructured Forests of Octrees, ACM/IEEE Proceedings of the 2012 International Conference for High Performance Computing, Networking, Storage and Analysis (SC12) (2012) (UT Austin)
  - Flat MPI, Repartitioning if DOF < $O(10^3)$ on each process

*h*CGA in the present work

- Accelerate the coarser grid solver
  - using multiple processes instead of a single process in CGA
  - Only 64 cells on each process of *lev=6* in the figure
- Straightforward Approach
  - MPI_Comm_split, MPI_Gather, MPI_Bcast etc.

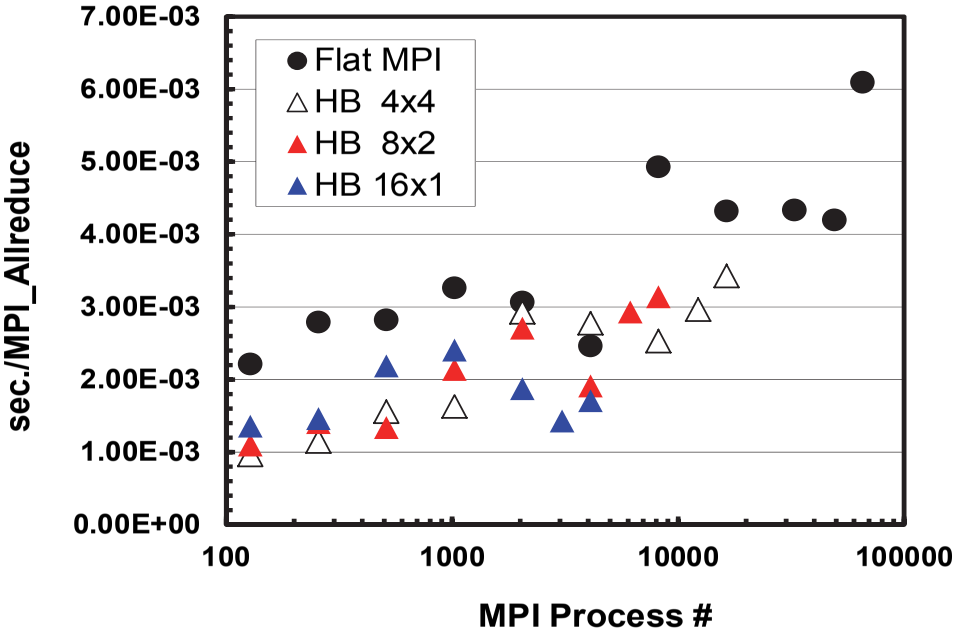


Summary

- *h*CGA is effective, but not so significant (except flat MPI)
  - flat MPI: x1.61 for weak scaling, x6.27 for strong scaling at 4,096 nodes of Fujitsu FX10
  - *h*CGA will be effective for HB 16x1 with more than 2.50×10^5 nodes (= 4.00×10^6 cores) of FX10 (=60 PFLOPS)
    - effect of coarse grid solver is significant for Flat MPI with $>10^3$ nodes
  - Communication overhead has been reduced by *h*CGA
- Future/On-Going Works and Open Problems
  - Improvement of *h*CGA
    - Overhead by MPI_Allreduce etc. -> P2P comm.
  - Algorithms
    - CA-Multigrid (for coarser levels), CA-SPAI, Pipelined Method
  - Strategy for Automatic Selection
    - switching level, number of processes for *h*CGA, optimum color #
    - effects on convergence
  - More Flexible ELL for Unstructured Grids
  - Xeon Phi Clusters
    - Hybrid 240(T)x1(P) is not the only choice

Overhead by Collective Communication

Overhead by MPI_Allreduce for MGCG case



Algorithm 7. Gropp's asynchronous CG

1: $r_0 := b - Ax_0$; $u_0 := M^{-1}r_0$; $p_0 := u_0$; $s_0 := Ap_0$; $\gamma_0 := (r_0, u_0)$
2: **for** $i = 0, \ldots$
3: $\quad \delta := (p_i, s_i)$
4: $\quad q_i := M^{-1}s_i$
5: $\quad \alpha_i := \gamma_i / \delta$
6: $\quad x_{i+1} := x_i + \alpha_i p_i$
7: $\quad r_{i+1} := r_i - \alpha_i s_i$
8: $\quad u_{i+1} := u_i - \alpha_i q_i$
9: $\quad \gamma_{i+1} := (r_{i+1}, u_{i+1})$
10: $\quad w_{i+1} := Au_{i+1}$
11: $\quad \beta_{i+1} := \gamma_{i+1} / \gamma_i$
12: $\quad p_{i+1} := u_{i+1} + \beta_{i+1} p_i$
13: $\quad s_{i+1} := w_{i+1} + \beta_{i+1} s_i$
14: **end for**

- Overhead by global collective comm. (e.g. MPI_Allreduce)
- Change original Krylov solver so that comm. overhead by global coll. comm. are hidden by overlapping with other computations (Gropp's asynch. CG, s-step, pipelined ...)
- "MPI_Iallreduce" in MPI-3 specification