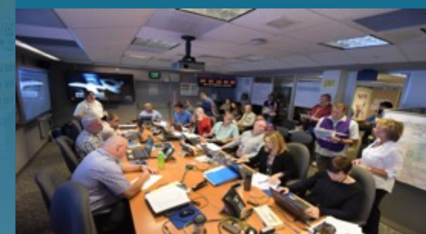
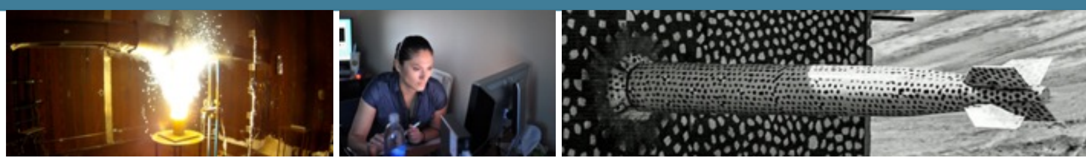
A Distributed-Memory Schur-complement PCA Preconditioner for Gemma Ill-conditioned Problems





Vinh Dang and Joseph Kotulski (1352)

vqdang@sandia.gov, jdkotul@sandia.gov

Trilinos User Group Meeting 2023

October 30th - November 2nd

Sandia National Laboratories

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

SAND 1727220

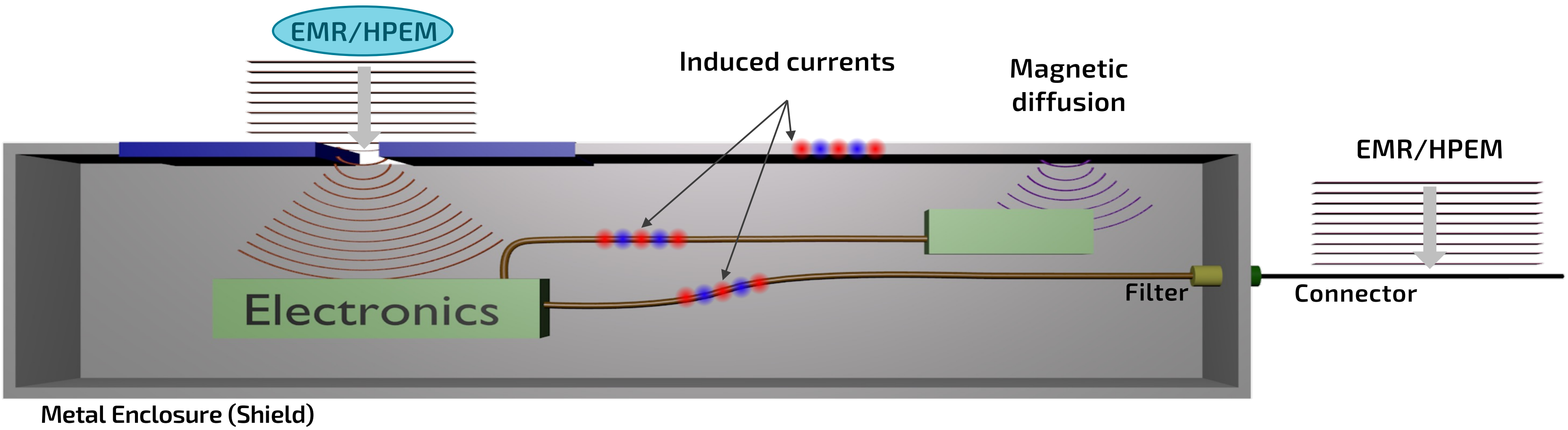
Outline

- Overview of Electromagnetic Radiation (EMR) Problem
- Motivation and Objective
- Performance Portability
- Overview of the Schur-complement Principal Component Analysis (PCA) Preconditioner
- Parallel Implementation
  - Distributed Binary Tree
  - Schur-complement PCA Factorization
  - Generalized Conjugate Residual (GCR) Solver
- Numerical Results
- Conclusions and Future Work

Electromagnetic Radiation (EMR) Problem Overview

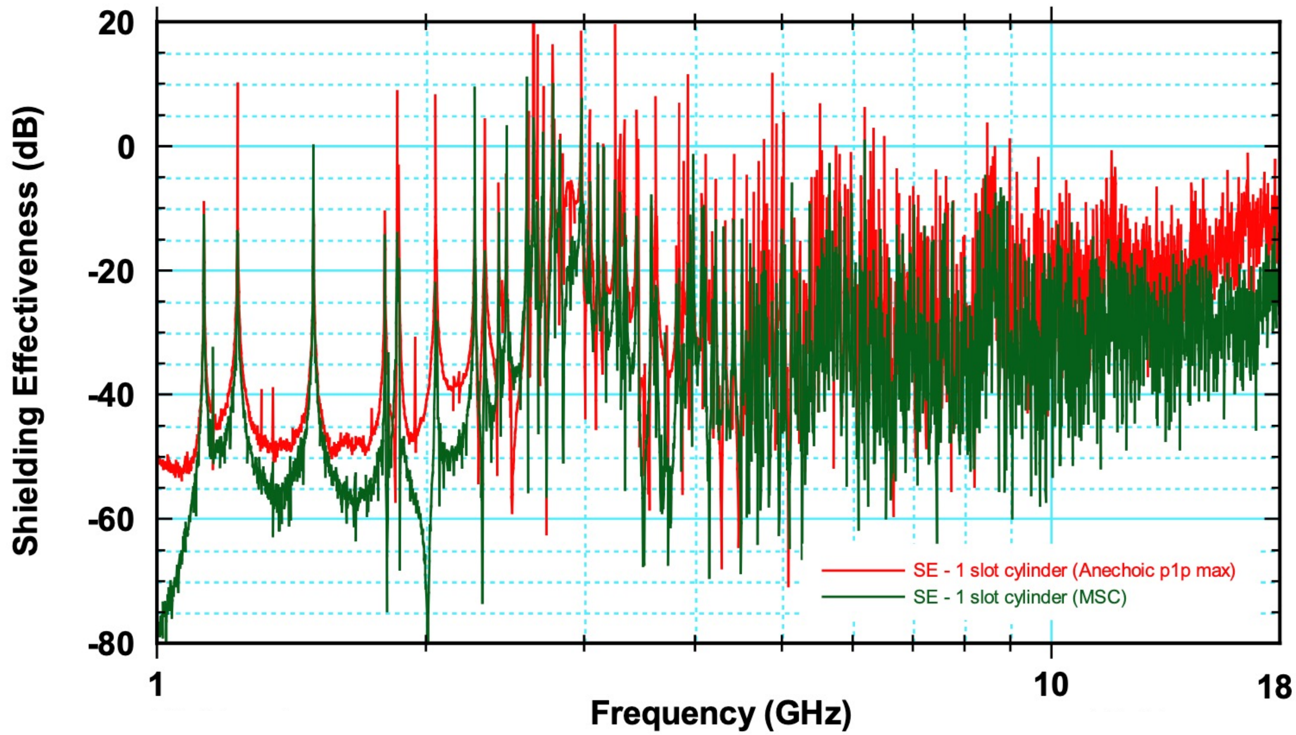
EMR Problem Overview

Electromagnetic (EM) energy couples into systems in many different ways. Our focus is energy coupling through mechanical seams or joints in the system housing, which acts as a good but imperfect EM shield. The joints form slots that allow EM energy into the system. Therefore, the problem has two parts that must be well-characterized: the slot and the cavity.



Shielding Effectiveness against Frequency for the Higgins Cylinder

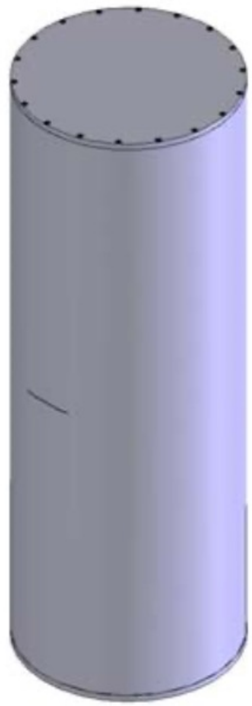
The 2” slot is halfway up the cylinder. A monopole probe is located inside the cylinder on one of the ends



Cylinder shielding effectiveness $SE = 20\ log_{10}\left(\frac{E_{interior}}{E_{exterior}}\right)$ in the mode-stirred and anechoic chambers

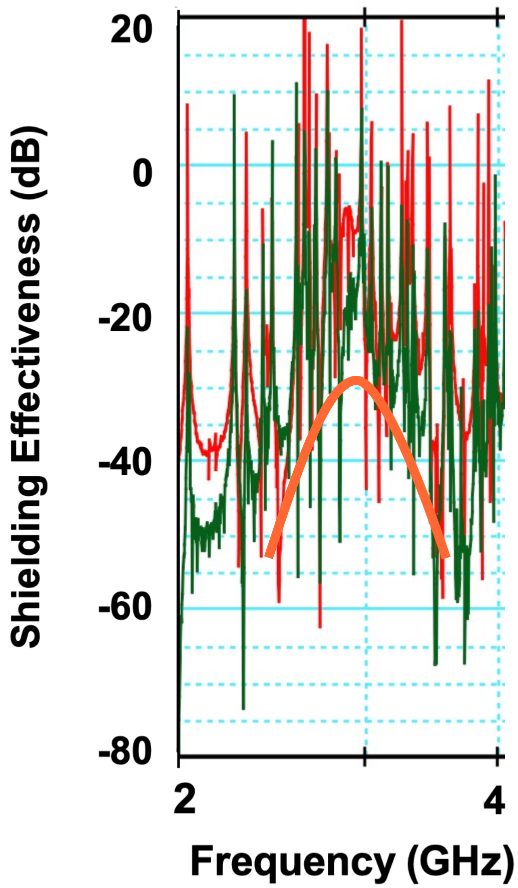
[1] Matthew B. Higgins and Dawna R. Charley, *Electromagnetic Radiation (EMR) Coupling to Complex Systems: Aperture Coupliing into Canonical Cavities in Reverberant and Anechoic Environments and Model Validation,* SAND2007-7931

Shielding Effectiveness against Frequency for the Higgins Cylinder



The 2" slot is halfway up the cylinder. A monopole probe is located inside the cylinder on one of the ends

- When the slot is at a resonance, it provides a larger drive to the cavity → the cavities modes have larger field magnitudes
- Therefore, the slot and the cavity have to be well-characterized *at resonances*



[1] Matthew B. Higgins and Dawna R. Charley, *Electromagnetic Radiation (EMR) Coupling to Complex Systems: Aperture Coupliing into Canonical Cavities in Reverberant and Anechoic Environments and Model Validation,* SAND2007-7931

Motivation and Objective

SNL's Electromagnetic Code Gemma

- Frequency-domain EM
- Hybrid approach, using:
  - A narrow slot sub-cell model to represent slots and the corresponding coupling
  - Surface integral equation method for outer region and inner region of cavity

Electric field integral equation (EFIE), for simplification:

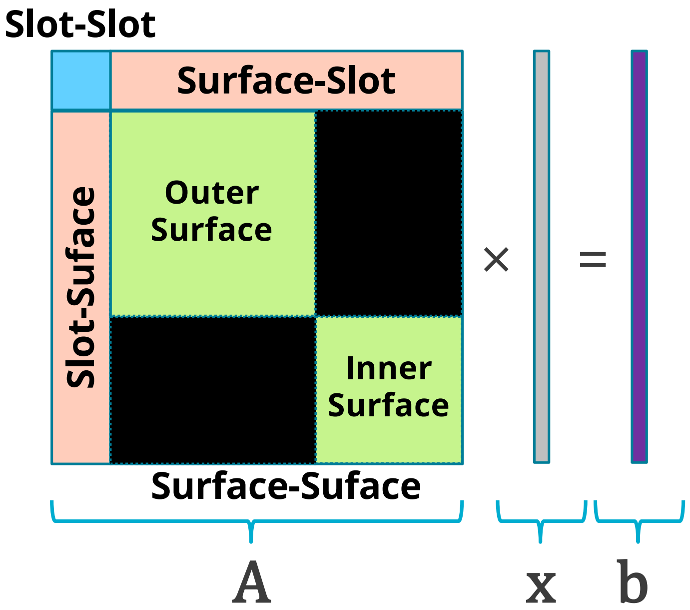
$$L\{\mathbf{J_S}\} = \frac{1}{j\omega\mu}\hat{\mathbf{n}} \times \mathbf{E_{inc}}$$

Expand unknown in a set of RWG basis functions:

$$\mathbf{J_S}(\mathbf{r}) \approx \sum_n I_n \mathbf{f_n}(\mathbf{r})$$

Test integral equation with RWG basis functions:

$$\int_S \mathbf{f_m} \cdot L\{\mathbf{J_S}\}\, ds = \frac{1}{j\omega\mu}\int_S \mathbf{f_m} \cdot (\hat{\mathbf{n}} \times \mathbf{E_{inc}})\, ds$$



- When an iterative method is used to solve the large, dense, complex matrix equation system:
  - **A** is extremely ill-conditioned for the problem of interest
  - Effective and efficient preconditioning is required

Objective

- A preconditioner using Schur-complement and Principal Component Analysis (PCA) on distributed-memory accelerator-based computing platforms for dense matrices (an extension of our collaboration with Ohio State University EM group)

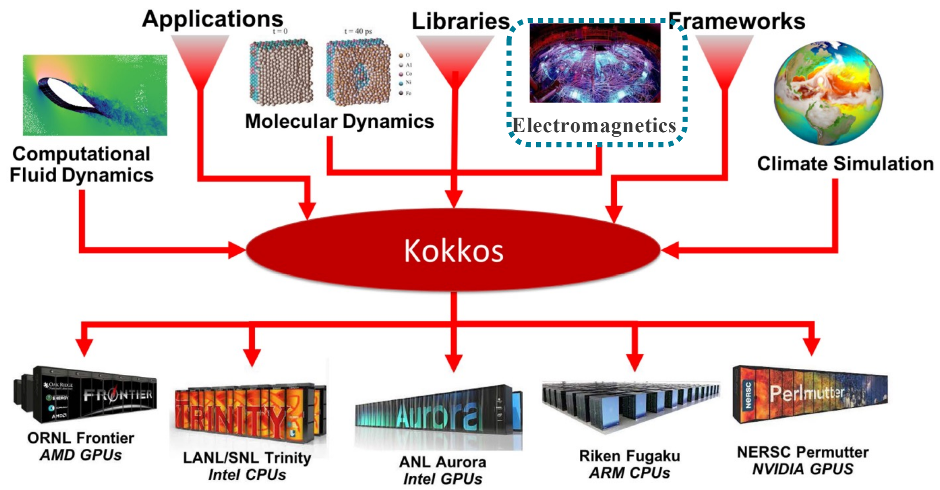
  - Propose a hierarchical binary distribution scheme for the binary tree

  - Target performance portability

  - For use with the Multilevel Fast Multipole Method (MLFMM) or the Adaptive Cross Approximation (ACA) for future work

Performance Portability

Performance Portability with Kokkos and Kokkos Kernels

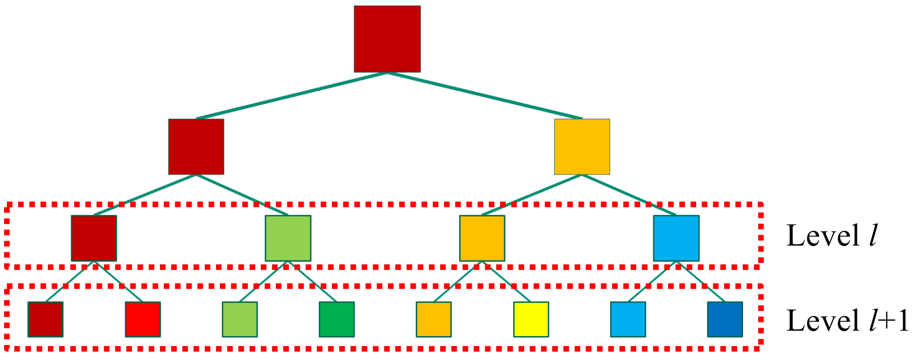


- Gemma is written on top of Kokkos (an open-source **productive**, **portable**, **performant**, shared-memory programming model) and Kokkos Kernels (a library for node-level, performance-portable, computational kernels for sparse/dense linear algebra and graph operations)

[2] C. Trott *et al.*, "The Kokkos EcoSystem: Comprehensive Performance Portability for High Performance Computing," in *Computing in Science & Engineering*, vol. 23, no. 5, pp. 10-18, 1 Sept.-Oct. 2021

Overview of the Schur-complement PCA Preconditioner

Schur-complement PCA Preconditioner



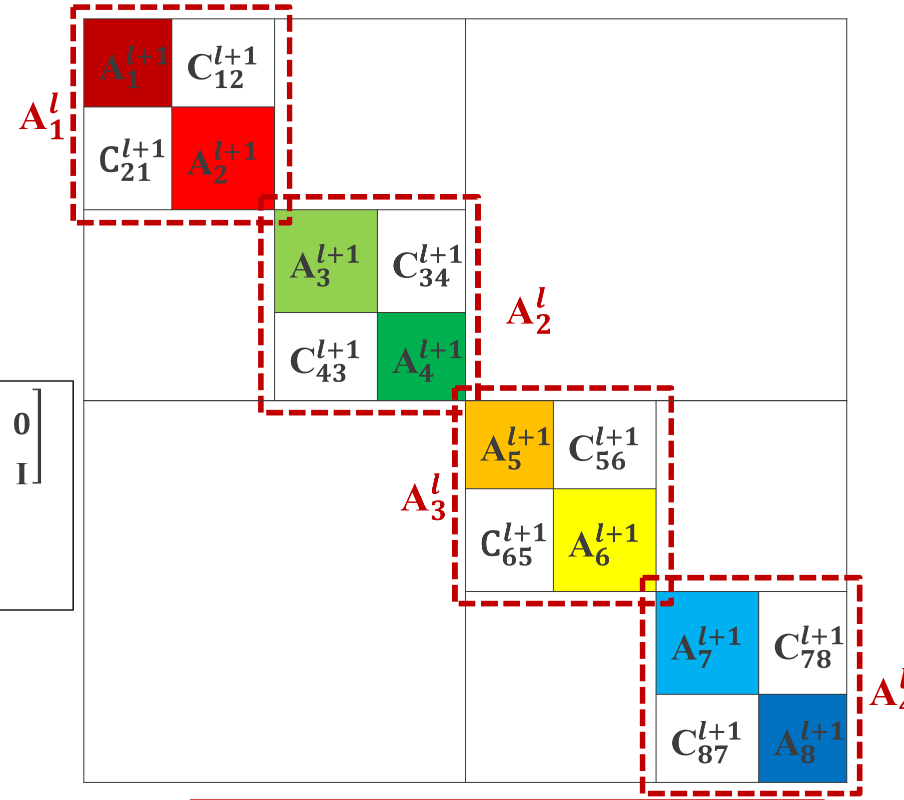
$$(\mathrm{A}_1^l)^{-1} = \begin{bmatrix} \mathrm{I} & 0 \\ -(\mathrm{A}_2^{l+1})^{-1}\mathrm{C}_{21}^{l+1} & \mathrm{I} \end{bmatrix} \begin{bmatrix} \left(\mathrm{I} - (\mathrm{A}_1^{l+1})^{-1}\mathrm{C}_{12}^{l+1}(\mathrm{A}_2^{l+1})^{-1}\mathrm{C}_{21}^{l+1}\right)^{-1} & \mathbf{0} \\ \mathbf{0} & \mathrm{I} \end{bmatrix} \begin{bmatrix} \mathrm{I} & -(\mathrm{A}_1^{l+1})^{-1}\mathrm{C}_{12}^{l+1} \\ \mathbf{0} & \mathrm{I} \end{bmatrix} \begin{bmatrix} (\mathrm{A}_1^{l+1})^{-1} & \mathbf{0} \\ \mathbf{0} & (\mathrm{A}_2^{l+1})^{-1} \end{bmatrix}$$

Factorization using PCA:

$$(\mathrm{A}_1^{l+1})^{-1}\mathrm{C}_{12}^{l+1} \xrightarrow{PCA} \mathrm{L}_1^{l+1}\mathrm{D}_1^{l+1}\mathrm{R}_1^{l+1} = \mathrm{L}_1^{l+1}\mathrm{R}_1^{l+1}$$
$$(\mathrm{A}_2^{l+1})^{-1}\mathrm{C}_{21}^{l+1} \xrightarrow{PCA} \mathrm{L}_2^{l+1}\mathrm{D}_2^{l+1}\mathrm{R}_2^{l+1} = \mathrm{L}_2^{l+1}\mathrm{R}_2^{l+1}$$

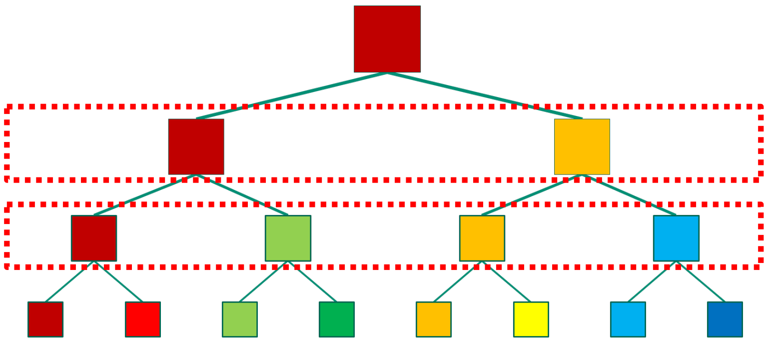
Inverse of Schur-complement:

$$\left(\mathrm{I} - (\mathrm{A}_1^{l+1})^{-1}\mathrm{C}_{12}^{l+1}(\mathrm{A}_2^{l+1})^{-1}\mathrm{C}_{21}^{l+1}\right)^{-1} \approx \left(\mathrm{I} - \mathrm{L}_1^{l+1}\mathrm{R}_1^{l+1}\mathrm{L}_2^{l+1}\mathrm{R}_2^{l+1}\right)^{-1}$$
$$= \mathrm{I} + \mathrm{L}_1^{l+1}\left(\mathrm{I} - \mathrm{R}_1^{l+1}\mathrm{L}_2^{l+1}\mathrm{R}_2^{l+1}\mathrm{L}_1^{l+1}\right)^{-1}\mathrm{R}_1^{l+1}\mathrm{L}_2^{l+1}\mathrm{R}_2^{l+1} = \mathrm{I} + \mathrm{L}_1^{l+1}S^l\mathrm{R}_2^{l+1}$$



Store: $\mathrm{L}_1^{l+1}(m\times k), \mathrm{R}_1^{l+1}(k\times n)$
$\mathrm{L}_2^{l+1}(n\times q), \mathrm{R}_2^{l+1}(q\times m)$
$S^l(k\times q)$

Schur-complement PCA Preconditioner



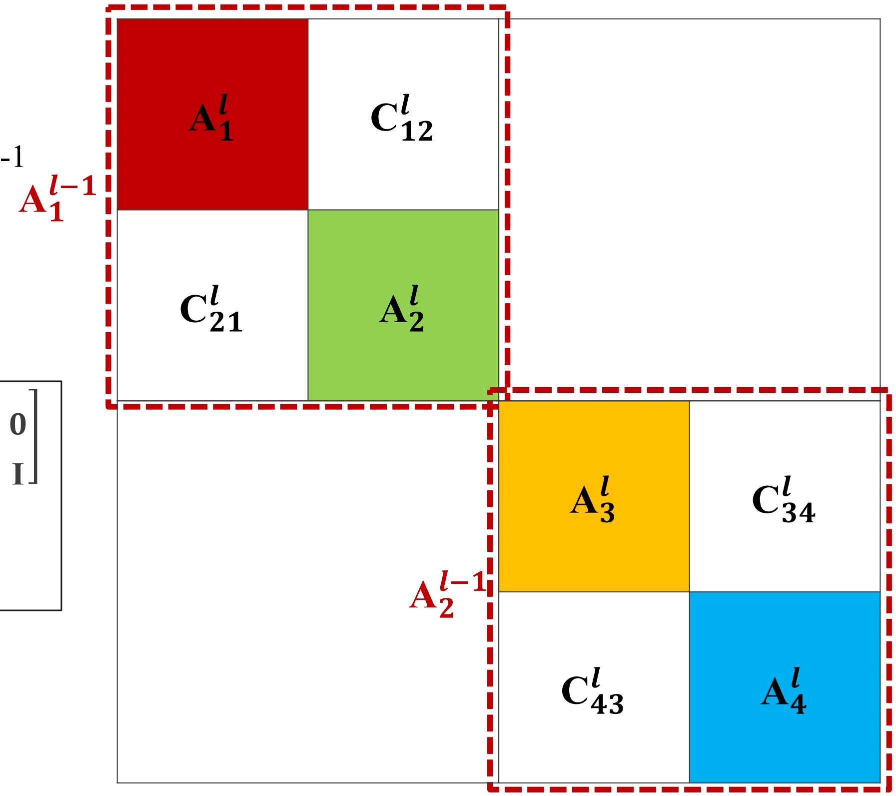
$$(A_1^{l-1})^{-1} = \begin{bmatrix} I & 0 \\ -(A_2^l)^{-1}C_{21}^l & I \end{bmatrix} \begin{bmatrix} \left(I-(A_1^l)^{-1}C_{12}^l(A_2^l)^{-1}C_{21}^l\right)^{-1} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & -(A_1^l)^{-1}C_{12}^l \\ 0 & I \end{bmatrix} \begin{bmatrix} (A_1^l)^{-1} & 0 \\ 0 & (A_2^l)^{-1} \end{bmatrix}$$

Factorization using PCA:

$$(A_1^l)^{-1}C_{12}^l \xrightarrow{PCA} L_1^l D_1^l R_1^l = L_1^l R_1^l$$
$$(A_2^l)^{-1}C_{21}^l \xrightarrow{PCA} L_2^{l+1} D_2^{l+1} R_2^{l+1} = L_2^{l+1} R_2^{l+1}$$

Inverse of Schur-complement:

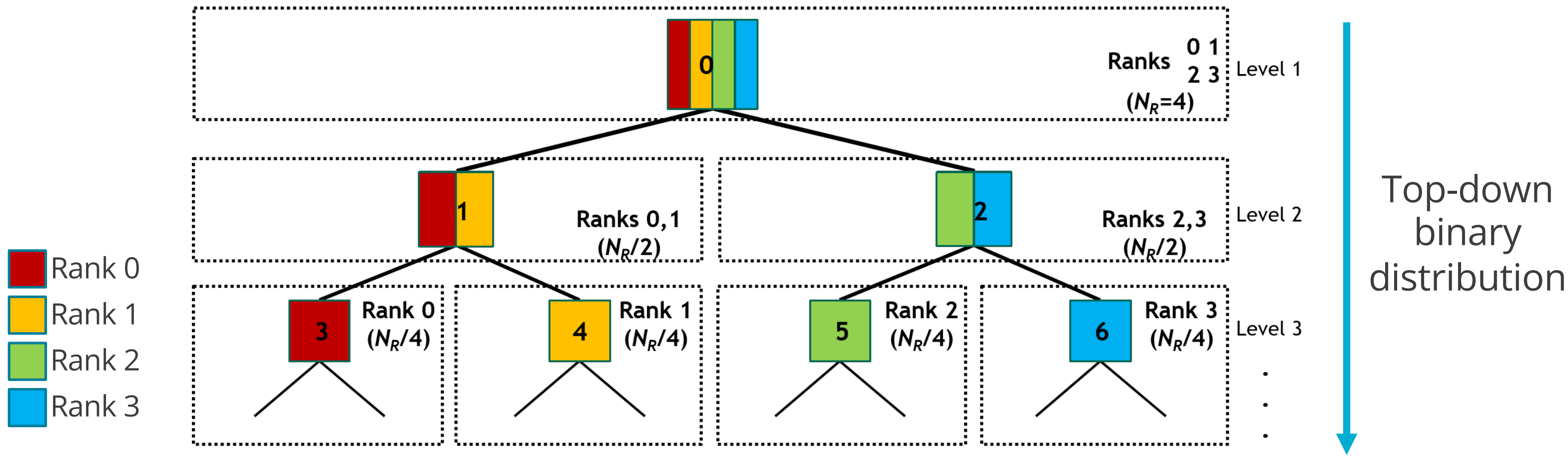
$$\left(I-(A_1^l)^{-1}C_{12}^l(A_2^l)^{-1}C_{21}^l\right)^{-1} \approx \left(I - L_1^l R_1^l L_2^l R_2^l\right)^{-1}$$
$$= I + L_1^l\left(I - R_1^l L_2^l R_2^l L_1^l\right)^{-1} R_1^l L_2^l R_2^l = I + L_1^l S^{l-1} R_2^l$$



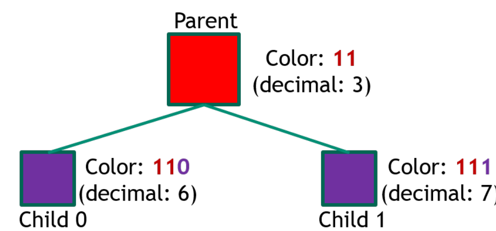
Store: $L_1^l(m'\times k'), R_1^l(k'\times n')$
$L_2^l(n'\times q'), R_2^l(q'\times m')$
$S^{l-1}(k'\times q')$

Parallel Implementation

Distributed Binary Tree

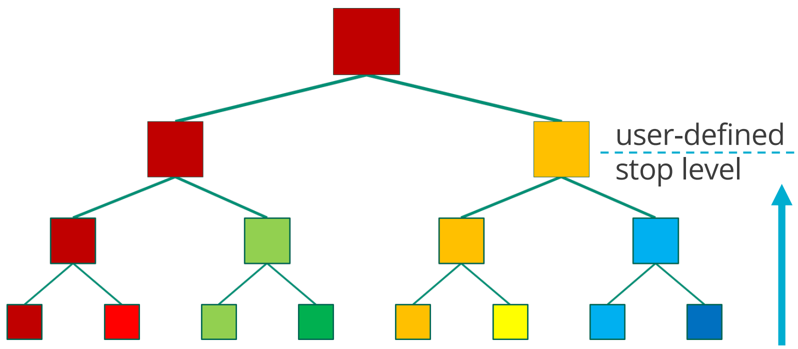


- Distribution scheme at each level:
  - Calculate colors for sub-communicators
  - Calculate colors for each binary-tree nodes
  - Add nodes to MPI processes if node colors match sub-communicator colors
- Binary color calculation:
  - Colors (binary code) are unique for nodes/communicators in the same binary-tree level
  - Colors of child nodes/communicators are calculated from colors of their parents

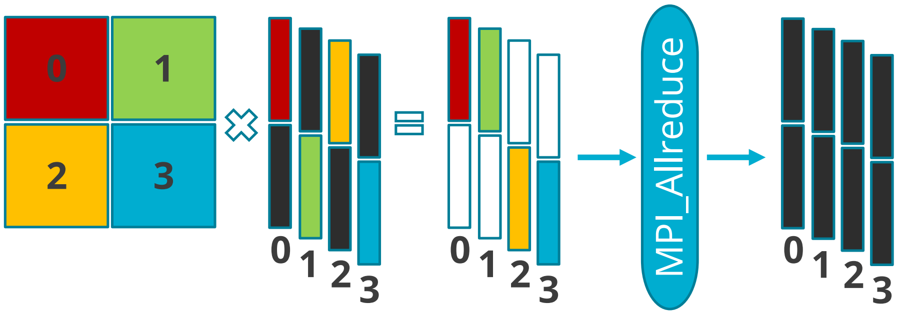


Schur-complement PCA Factorization

- What need to be stored in binary tree nodes?
  - All coupling matrices **C**
  - Diagonal block matrices of **A** at the lowest level
  - Block matrices of **A** at higher levels: store the **L**, **R**, and **S** matrices
- Block-based matrix distribution if a tree node is handled by more than one MPI rank
- Factorization: traversing the binary tree from the finest level up to a user-defined stop level while computing the **L**, **R**, and **S** matrices at each level using the PCA algorithm [3] (based on singular value decomposition - SVD)
- In-house distributed matrix-matrix (vector) multiplication (GEMM/GEMV)
- TPLs: singular value decomposition (SVD) and distributed linear equation solver GESV



Bottom-up Traversal for Factorization

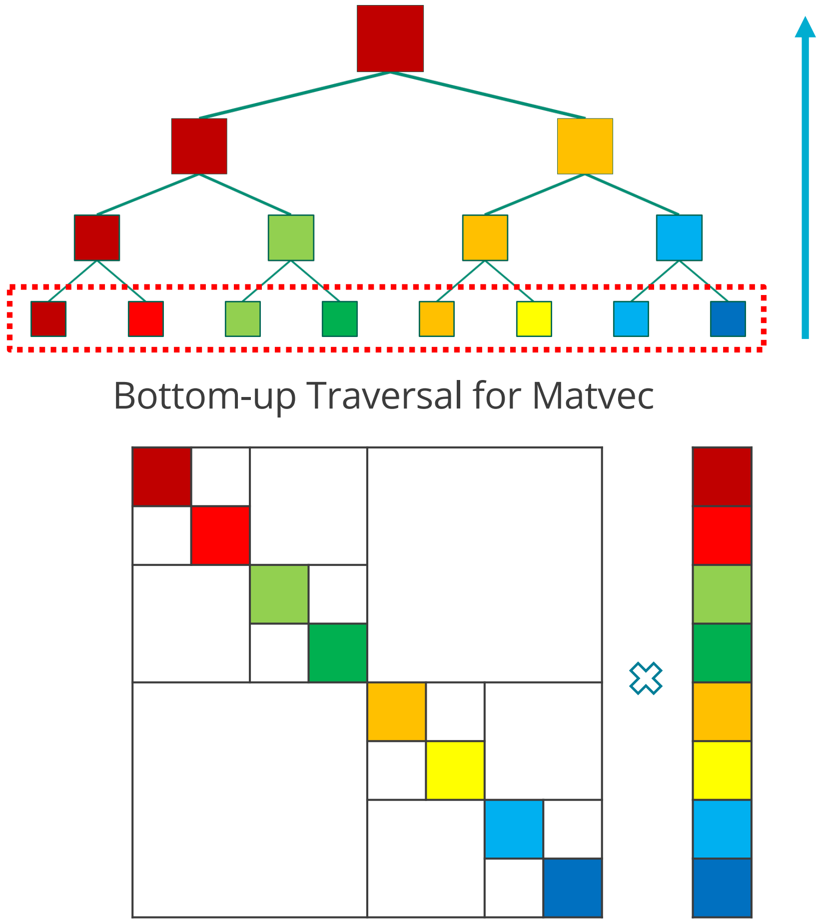


Distributed matrix-matrix/vector multiplication GEMM/V

[3] http://helper.ipam.ucla.edu/publications/setut/setut_7373.pdf

GCR Solver

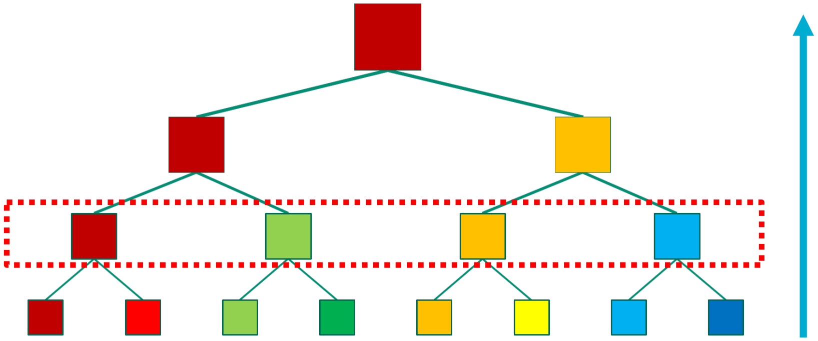
- Generalized Conjugate Residual (GCR) method [4] was chosen for its simplicity and effectiveness
- Two most computational operations: applying preconditioner and matvec
- Applying preconditioner: using the bottom-up traversal as in the factorization and the in-house distributed GEMV with on-the-fly inverse matrices calculated from the stored **L**, **R**, and **S** matrices at each level
- Distributed binary-tree based matvec:
  - using the diagonal block matrices of **A** and the coupling matrices **C**
  - using the **entire** binary tree
  - output of a level becomes input for its upper level
  - using **MPI_Allreduce** to gather final results



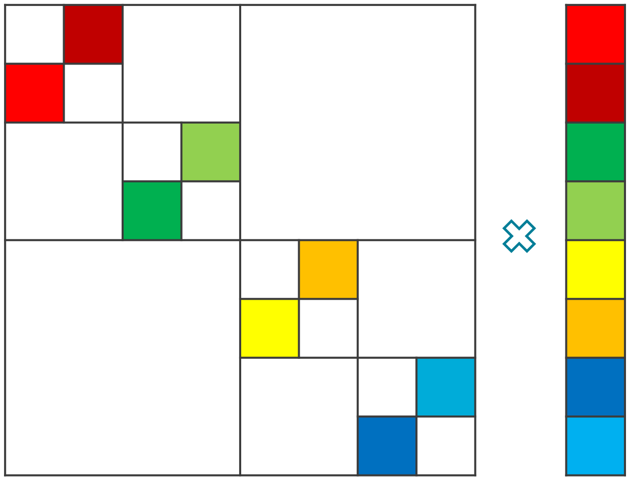
[4] Yousef Saad, Iterative Methods for Sparse Linear Systems; SIAM, 2003

GCR Solver

- Generalized Conjugate Residual (GCR) method [4] was chosen for its simplicity and effectiveness
- Two most computational operations: applying preconditioner and matvec
- Applying preconditioner: using the bottom-up traversal as in the factorization and the in-house distributed GEMV with on-the-fly inverse matrices calculated from the stored **L**, **R**, and **S** matrices at each level
- Distributed binary-tree based matvec:
  - using the diagonal block matrices of **A** and the coupling matrices **C**
  - using the **entire** binary tree
  - output of a level becomes input for its upper level
  - using **MPI_Allreduce** to gather final results



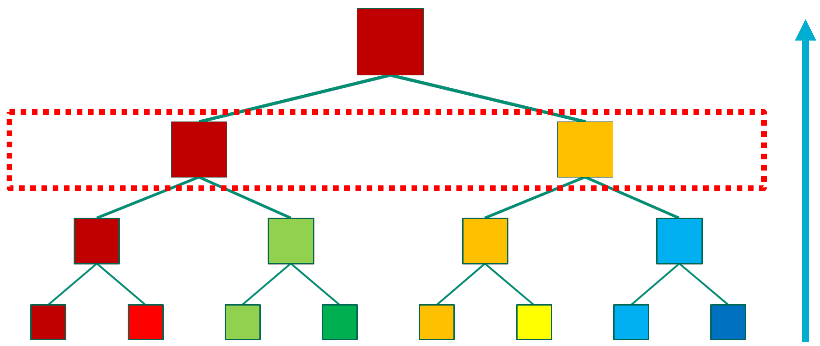
Bottom-up Traversal for Matvec



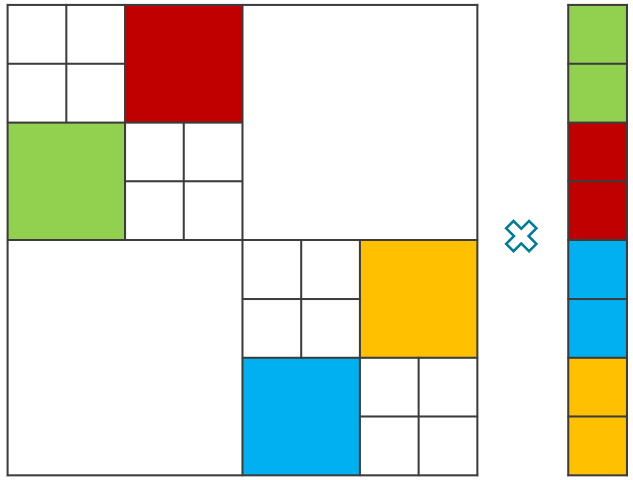
[4] Yousef Saad, Iterative Methods for Sparse Linear Systems; SIAM, 2003

GCR Solver

- Generalized Conjugate Residual (GCR) method [4] was chosen for its simplicity and effectiveness
- Two most computational operations: applying preconditioner and matvec
- Applying preconditioner: using the bottom-up traversal as in the factorization and the in-house distributed GEMV with on-the-fly inverse matrices calculated from the stored **L**, **R**, and **S** matrices at each level
- Distributed binary-tree based matvec:
  - using the diagonal block matrices of **A** and the coupling matrices **C**
  - using the **entire** binary tree
  - output of a level becomes input for its upper level
  - using **MPI_Allreduce** to gather final results



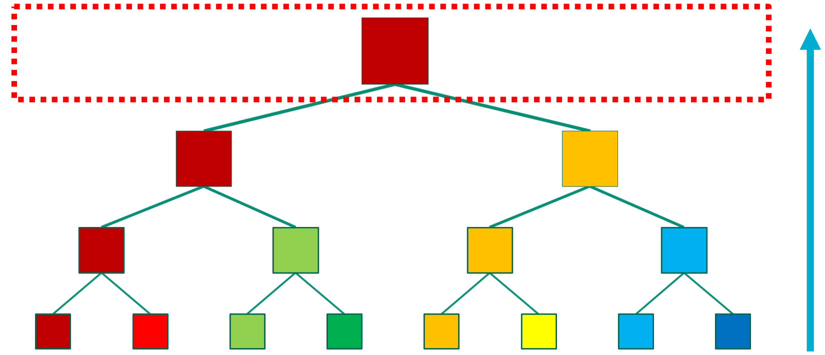
Bottom-up Traversal for Matvec



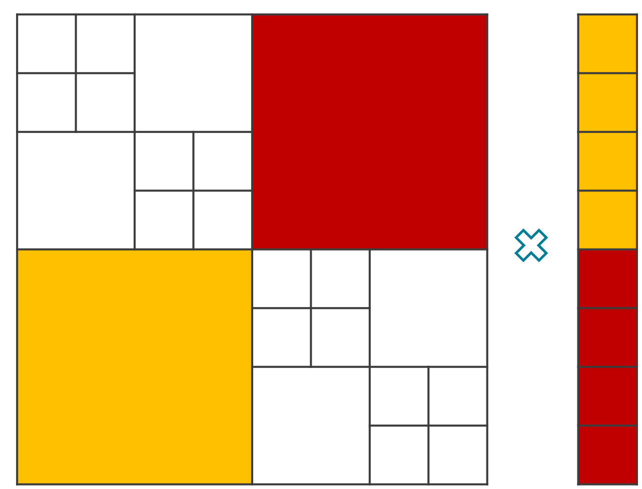
[4] Yousef Saad, Iterative Methods for Sparse Linear Systems; SIAM, 2003

GCR Solver

- Generalized Conjugate Residual (GCR) method [4] was chosen for its simplicity and effectiveness
- Two most computational operations: applying preconditioner and matvec
- Applying preconditioner: using the bottom-up traversal as in the factorization and the in-house distributed GEMV with on-the-fly inverse matrices calculated from the stored **L**, **R**, and **S** matrices at each level
- Distributed binary-tree based matvec:
  - using the diagonal block matrices of **A** and the coupling matrices **C**
  - using the **entire** binary tree
  - output of a level becomes input for its upper level
  - using **MPI_Allreduce** to gather final results



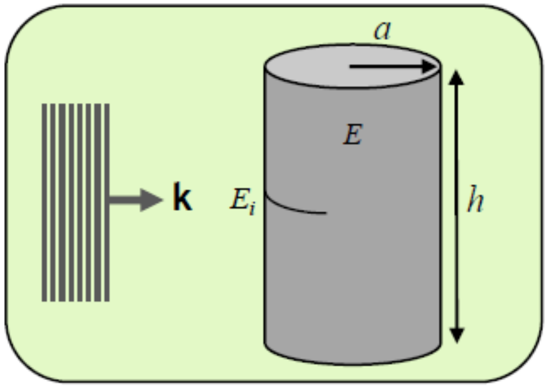
Bottom-up Traversal for Matvec



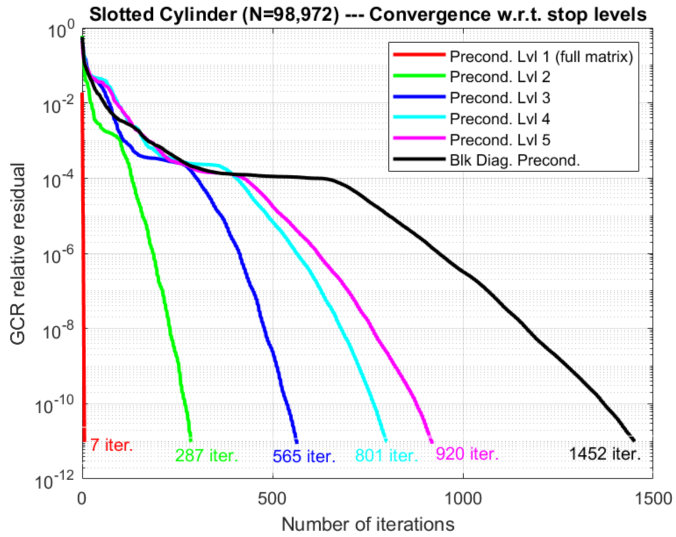
[4] Yousef Saad, Iterative Methods for Sparse Linear Systems; SIAM, 2003

Numerical Results

Simulation Setup and Convergence Behavior

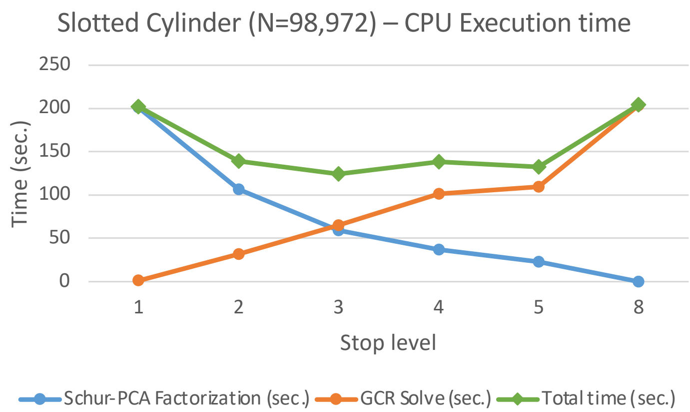


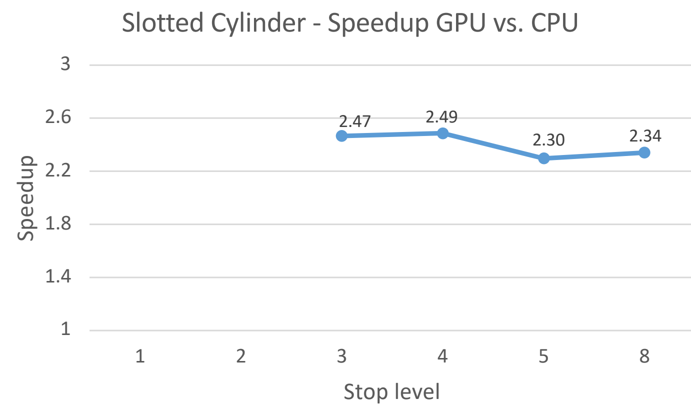
- High-Q factor slotted cylindrical cavity at f = 1.1295GHz (near resonance) with 98,972 unknowns
- The matrix and RHS vector are generated by the method of moments code EIGER
- Schur-PCA tolerance: 10e-5 and GCR solver tolerance: 10e-11
- Test platform: LLNL's Lassen with POWER9 CPUs, V100 GPUs, gcc/8.3.1, cuda/11.8.0, essl/6.3.0.1, lapack/3.10.0-xl-2022.03.10, spectrum-mpi/rolling-release



- Lvl 1 → using the whole binary tree
- Lvl 8 (finest level) → block diagonal preconditioner
- The convergence rate is fastest with lvl 1 and reduces (i.e. more iterations) when we increase the stop level
  - The iteration change is significant from 7 (lvl 1) to 287 (lvl 2) and 1452 (lvl 8)

CPU and GPU Executions at Different Stop Levels

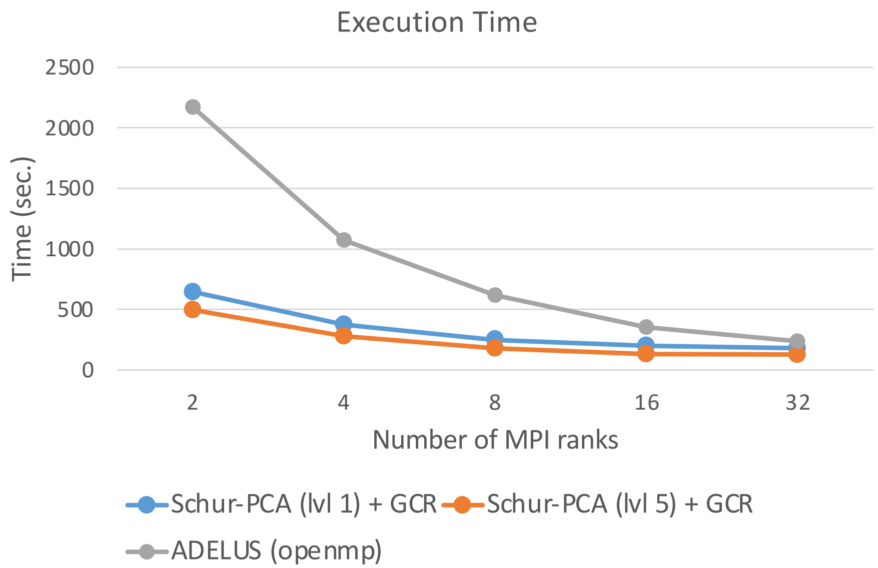


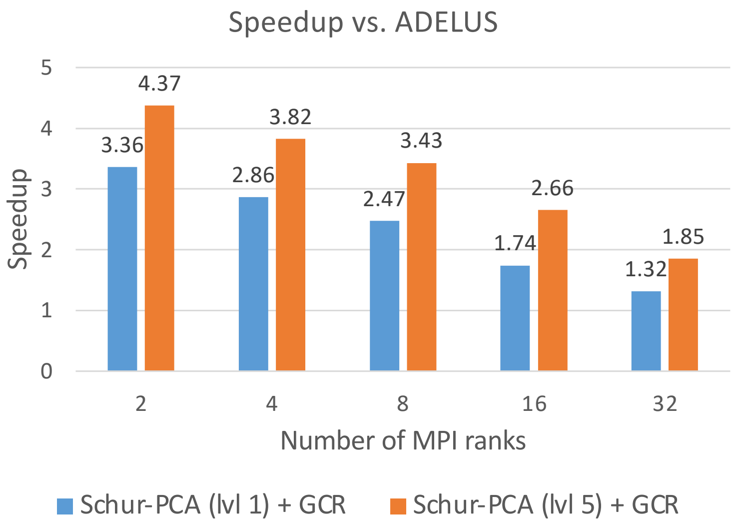


- 16 computing nodes with 16 MPI ranks (1 rank per node): 44-core CPUs per rank vs. 1 GPU per rank
- The factorization time (blue) decreases as the stop level increases while the solve time (orange) increases with more iterations and becomes dominant in the total time (green)
- As the stop level increases, the memory requirement of the Schur-PCA factorization are less while the memory requirement of Krylov subspace is higher

- *Should use a stop level corresponding to the middle of the binary tree to balance between run time and memory footprint*
- GPU runs at stop levels of 3, 4, 5, 8 due to GPU memory limitation
- GPU execution is ~2.3x faster CPU execution

Schur-PCA Preconditioner + GCR vs. ADELUS: Scalability





- The total CPU run times with stop levels 1 and 5 are compared with those of the direct solver ADELUS on 2 ranks, 4 ranks, 8 ranks, 16 ranks, 32 ranks
- Schur-PCA Preconditioner + GCR outperforms ADELUS
- Scalability needs optimization as the number of MPI ranks increases (current limitation: use of full-size vectors)

Conclusions and Future Work

- Developed a parallel implementation of the Schur-complement PCA preconditioner on distributed-memory systems with Kokkos for performance portability
- Effective in solving ill-conditioned problems
- Schur-complement PCA preconditioner + GCR on CPUs is faster than ADELUS
- Future work:
  - Has just been integrated into Gemma: full performance with on-the-fly matrix construction will be done next
  - Performance optimization to achieve better scalability on larger computing systems for larger sized problems

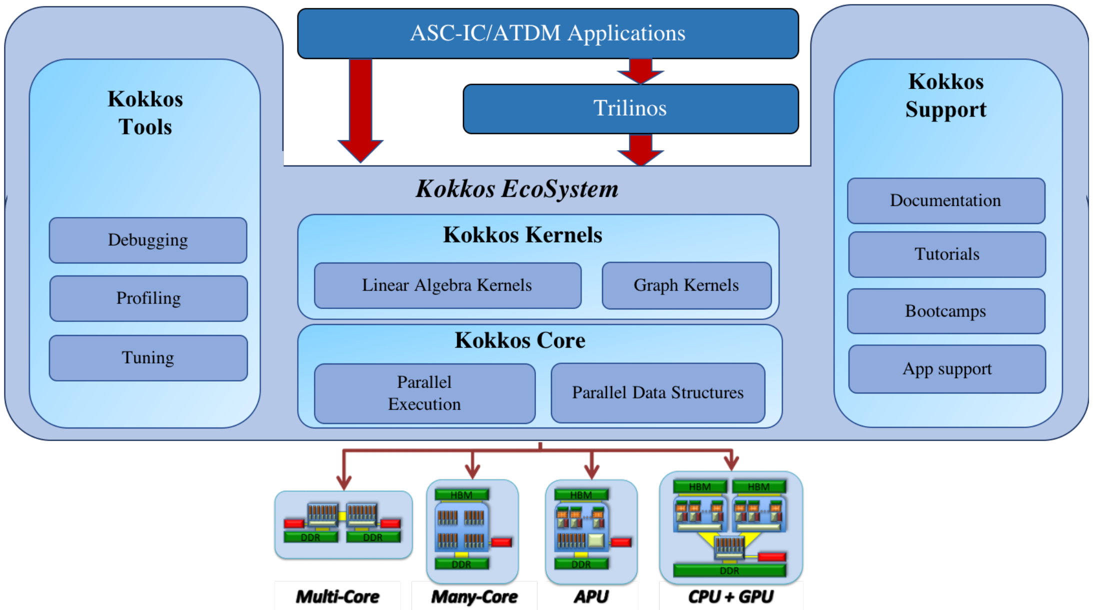
Thank You!

Backup

Kokkos Kernels Overview

Kokkos Kernels is a library for *node-level*, performance-portable, computational kernels for sparse/dense linear algebra and graph operations, using the Kokkos

- KK is available publicly both as part of Trilinos and as part of the **Kokkos ecosystem** (https://github.com/kokkos/kokkos-kernels)
- **Building block** of a solver, linear algebra library that uses MPI and threads for parallelism, or it can be used stand-alone in an application
- Interfaces to **vendor-provided kernels** available in order to leverage their high-performance libraries



Binary Tree-Based Inverse Matrix-Vector Multiplication

.....in compact form,

$$\begin{bmatrix} \mathbf{Y}_1' \\ \mathbf{Y}_2' \end{bmatrix} = \mathbf{B}_1^{-1} \begin{bmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \end{bmatrix} = \begin{bmatrix} (\mathbf{I} + \mathbf{L}_1 \mathbf{S} \mathbf{R}_1)(\mathbf{Y}_1 - \mathbf{L}_1 \mathbf{R}_1 \mathbf{Y}_2) \\ \mathbf{Y}_2 - \mathbf{L}_2 \mathbf{R}_2 (\mathbf{I} + \mathbf{L}_1 \mathbf{S} \mathbf{R}_1)(\mathbf{Y}_1 - \mathbf{L}_1 \mathbf{R}_1 \mathbf{Y}_2) \end{bmatrix}$$

where $\mathbf{Y}_1 = \mathbf{A}_1^{-1}\mathbf{X}_1$ is an update on $\mathbf{X}_1$ from child1

$\mathbf{Y}_2 = \mathbf{A}_2^{-1}\mathbf{X}_2$ is an update on $\mathbf{X}_2$ from child2

$\mathbf{S}^{k\times q} = (\mathbf{I} - \mathbf{R}_1 \mathbf{L}_2 \mathbf{R}_2 \mathbf{L}_1)^{-1} \mathbf{R}_1 \mathbf{L}_2$

$\mathbf{M} \times \mathbf{V}$ procedure:

1) Declare 2 working vectors $\mathbf{W}_1$ and $\mathbf{W}_2$, size max(k,q)
2) Compute $\mathbf{W}_1^k = \mathbf{R}_1^{k\times n} \cdot \mathbf{Y}_1^n$
3) Update $\mathbf{Y}_1^m = \mathbf{Y}_1^m - \mathbf{L}_1^{m\times k} \cdot \mathbf{W}_1^k$
4) Compute $\mathbf{W}_1^q = \mathbf{R}_2^{q\times m} \cdot \mathbf{Y}_1^m$
5) Compute $\mathbf{W}_2^k = \mathbf{S}^{k\times q} \cdot \mathbf{W}_1^q$
6) Update $\mathbf{Y}_1^m = \mathbf{Y}_1^m + \mathbf{L}_1^{m\times k} \cdot \mathbf{W}_1^k$
7) Compute $\mathbf{W}_1^q = \mathbf{R}_2^{q\times m} \cdot \mathbf{Y}_1^m$
8) Update $\mathbf{Y}_2^n = \mathbf{Y}_2^n - \mathbf{L}_2^{n\times q} \cdot \mathbf{W}_1^q$