Argonne Training Program on Extreme-Scale Computing

ATPESC 2023

Krylov Solvers and Algebraic Multigrid with *hypre*

Sarah Osborn, Ulrike Meier Yang
Lawrence Livermore National Laboratory

July 30 – August 11, 2023

Sarah Osborn, Ulrike Meier Yang
Lawrence Livermore National Laboratory

Outline

- What are Krylov Solvers?

- Why are they used?

- Why multigrid methods?

- Algebraic multigrid software

- Hypre software library – interfaces
  - Why different interfaces?

- How does multigrid work?

- Unstructured vs structured multigrid solvers

Iterative Solvers

- Solve linear system $Ax = b$,
  where A is a large sparse matrix of size n

- Direct solvers (e.g., Gaussian elimination) too expensive

- Iterative solvers

- Richardson iteration:
  $$x^{n+1} = x^n + (b - Ax^n)$$
  $$e^{n+1} = (I - A)e^n$$

- Introduce a preconditioner B:
  $$x^{n+1} = x^n + B(b - Ax^n)$$
  $$e^{n+1} = (I - BA)e^n$$

- Jacobi: $B = D^{-1}$;   Richardson: $B = \lambda I$

Generalized Minimal Residual (GMRES)

- $x^{n+1} = x^n + B(b - Ax^n)$

- $\Rightarrow x^{n+1} = \sum_{i=0}^{n} \alpha_i (BA)^i Bb$

- $x^{n+1} \in K^n = span\{Bb, (BA)Bb, (BA)^2 Bb, \dots, (BA)^n Bb\}$

Krylov space

- Now optimize by defining x^{n+1} through
$$\min_{x^{n+1} \in K^n} \|B(Ax^{n+1} - b)\|$$

- Construct a new basis for K^n through orthonormalization
$$\{q_0 = \frac{Bb}{\|Bb\|}, q_1, \dots, q_n\}$$

- Solve the minimization in the new basis

- q_i also called search directions

4

Some comments on GMRES

- GMRES consists of fairly simple operations:
  - Inner products and norms (global reductions)
  - Vector updates (embarrassingly parallel)
  - Matvecs (nearest neighbor updates)
  - Application of preconditioner (can be very complicated)

- Often used restarted as GMRES(k), i.e., after k iterations throw out q_i and start again using latest approximation

- Many variants to reduce and/or overlap communication (pipelined GMRES, etc)

Other Krylov solvers

- Conjugate Gradient (CG)
  - For symmetric positive definite matrices
  - Possesses like GMRES an orthogonality property
  - Uses a three-term concurrence
  - Requires only two inner products and a norm per iteration

- BiCGSTAB (Biconjugate Gradient Stabilized)
  - Like CG uses a three-term recurrence relation
  - No orthogonality property, can break down
  - Requires several inner products and a norm at each iteration (and two matvecs)
  - More erratic convergence than GMRES, but needs generally less memory

Hands-on Exercises: Krylov methods

- Go to https://xsdk-project.github.io/MathPackagesTraining2023/lessons/krylov_amg_hypre/

- Poisson equation: $-\Delta\varphi$ = RHS

  with Dirichlet boundary conditions $\varphi = 0$

- Grid: cube
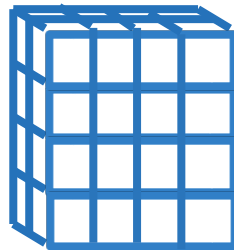
- Finite difference discretization:
  - Central differences for diffusion term
  - 7-point stencil
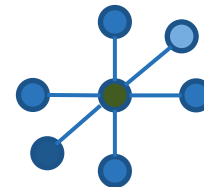
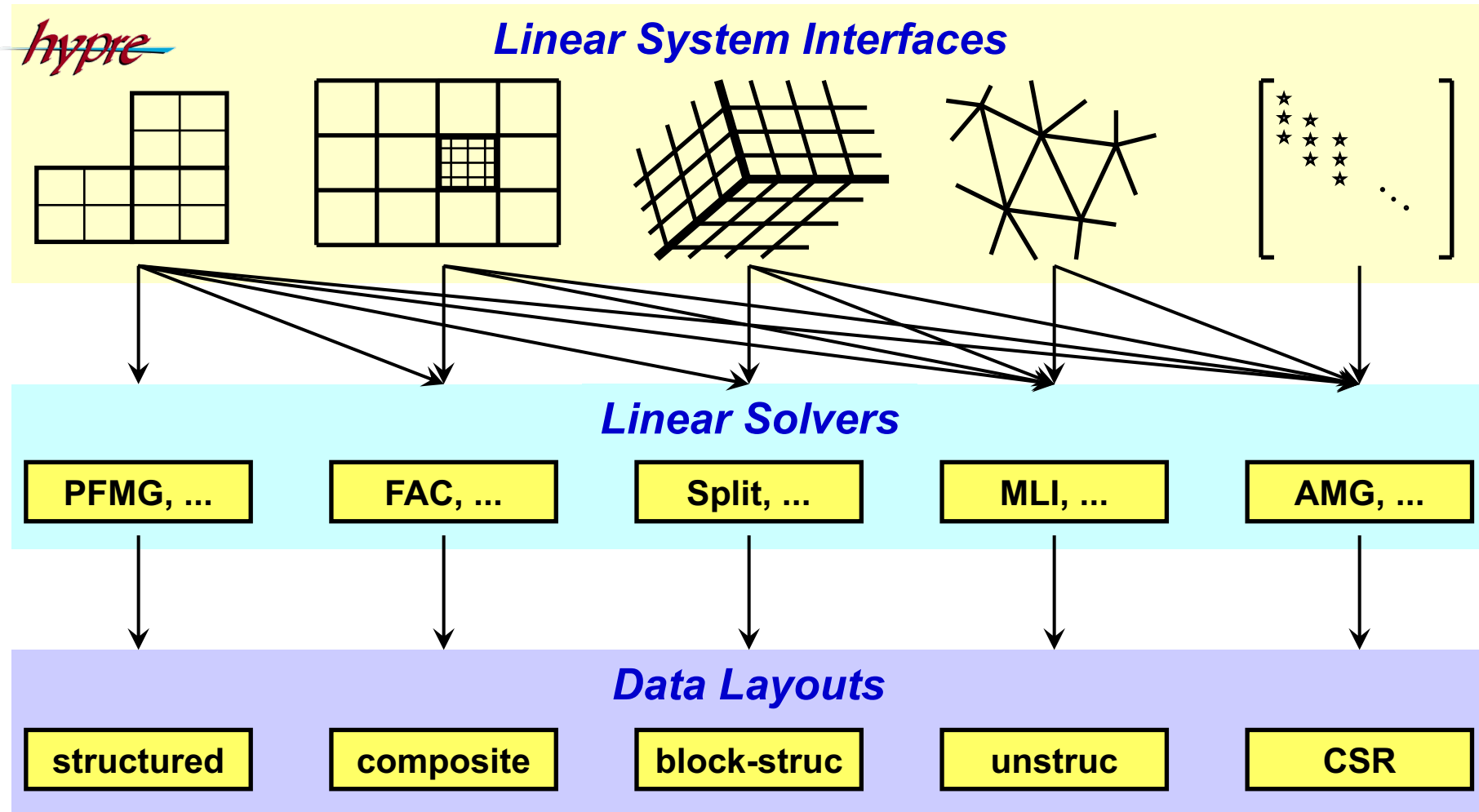# Multigrid linear solvers are optimal ($O(N)$ operations), and hence have good scaling potential
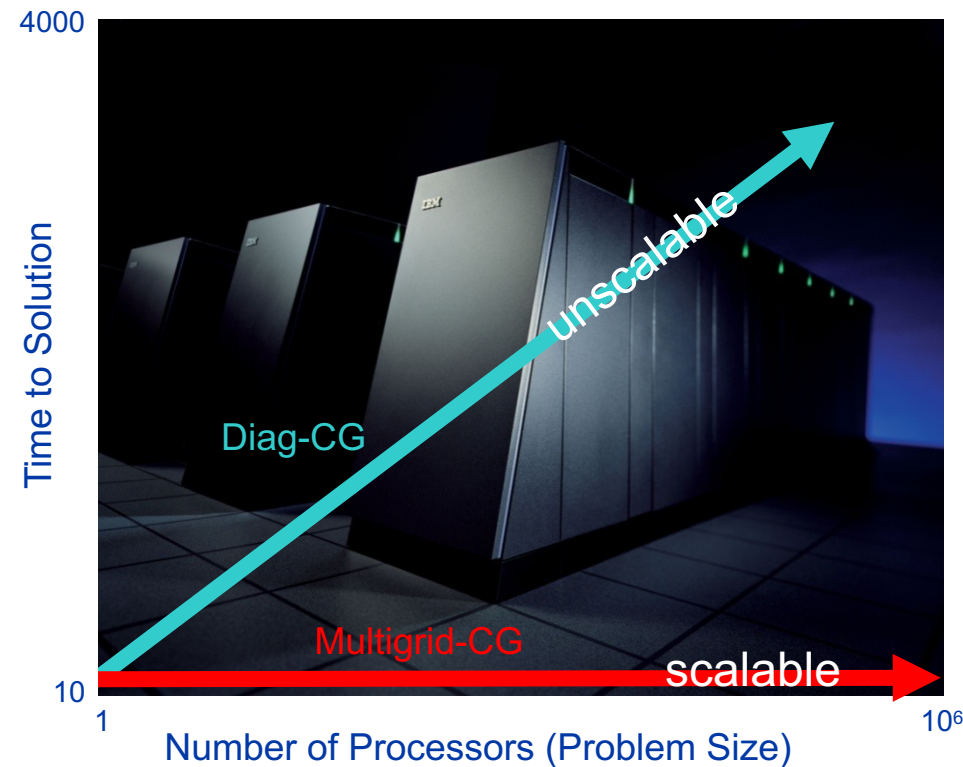
- Weak scaling – want constant solution time as problem size grows in proportion to the number of processors

Multigrid software

- ML, MueLu included in **TRILINOS**

- GAMG in **PETSc**

- The *hypre* library provides various algebraic multigrid solvers, including multigrid solvers for special problems e.g., Maxwell equations, …

- …

- All of these provide different flavors of multigrid and provide excellent performance for suitable problems

- Focus here on *hypre*

# (Conceptual) linear system interfaces are necessary to provide "best" solvers and data layouts

Linear System Interfaces

Linear Solvers

| PFMG, ... | FAC, ... | Split, ... | MLI, ... | AMG, ... |

Data Layouts

| structured | composite | block-struc | unstruc | CSR |

Why multiple interfaces? The key points

- Provides natural "views" of the linear system

- Eases some of the coding burden for users by eliminating the need to map to rows/columns

- Provides for more efficient (scalable) linear solvers

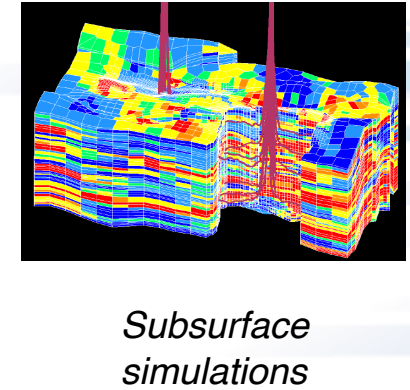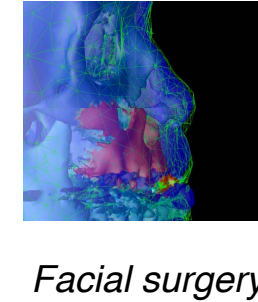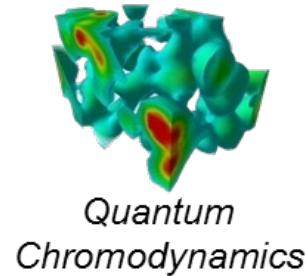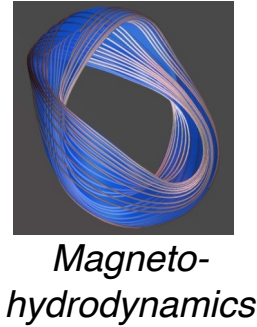- Provides for more effective data storage schemes and more efficient computational kernels
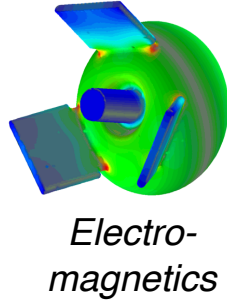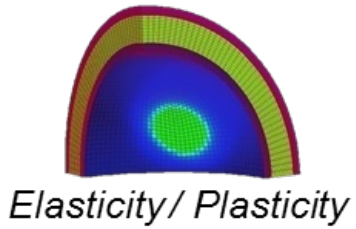
hypre supports these system interfaces

- ## Structured-Grid (`Struct`)
  - *logically rectangular grids*

- ## Semi-Structured-Grid (`SStruct`)
  - *grids that are mostly structured*
  - *Examples: block-structured grids, structured adaptive mesh refinement grids, overset grids*
  - *Finite elements*

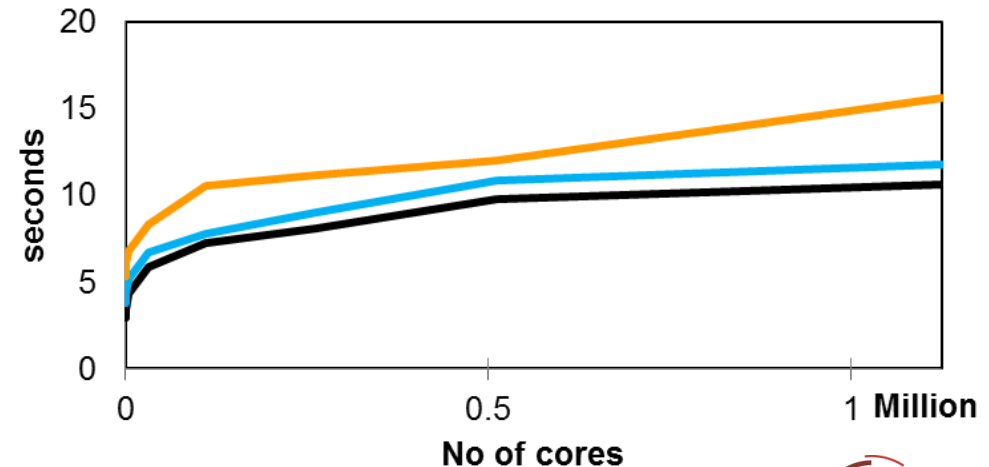- ## Linear-Algebraic (`IJ`)
  - *general sparse linear systems*
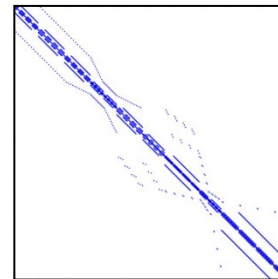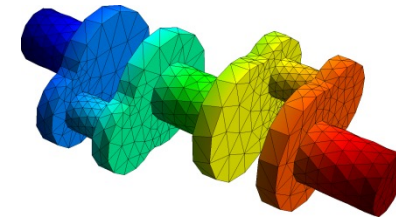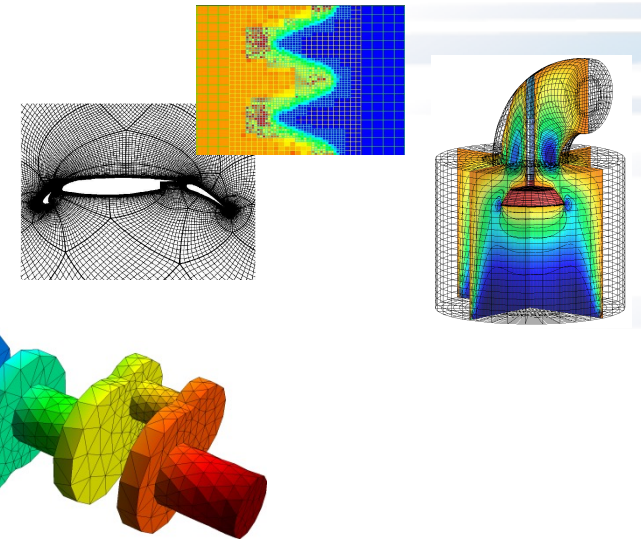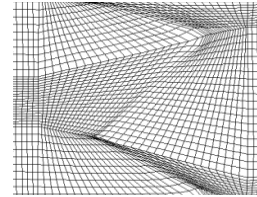
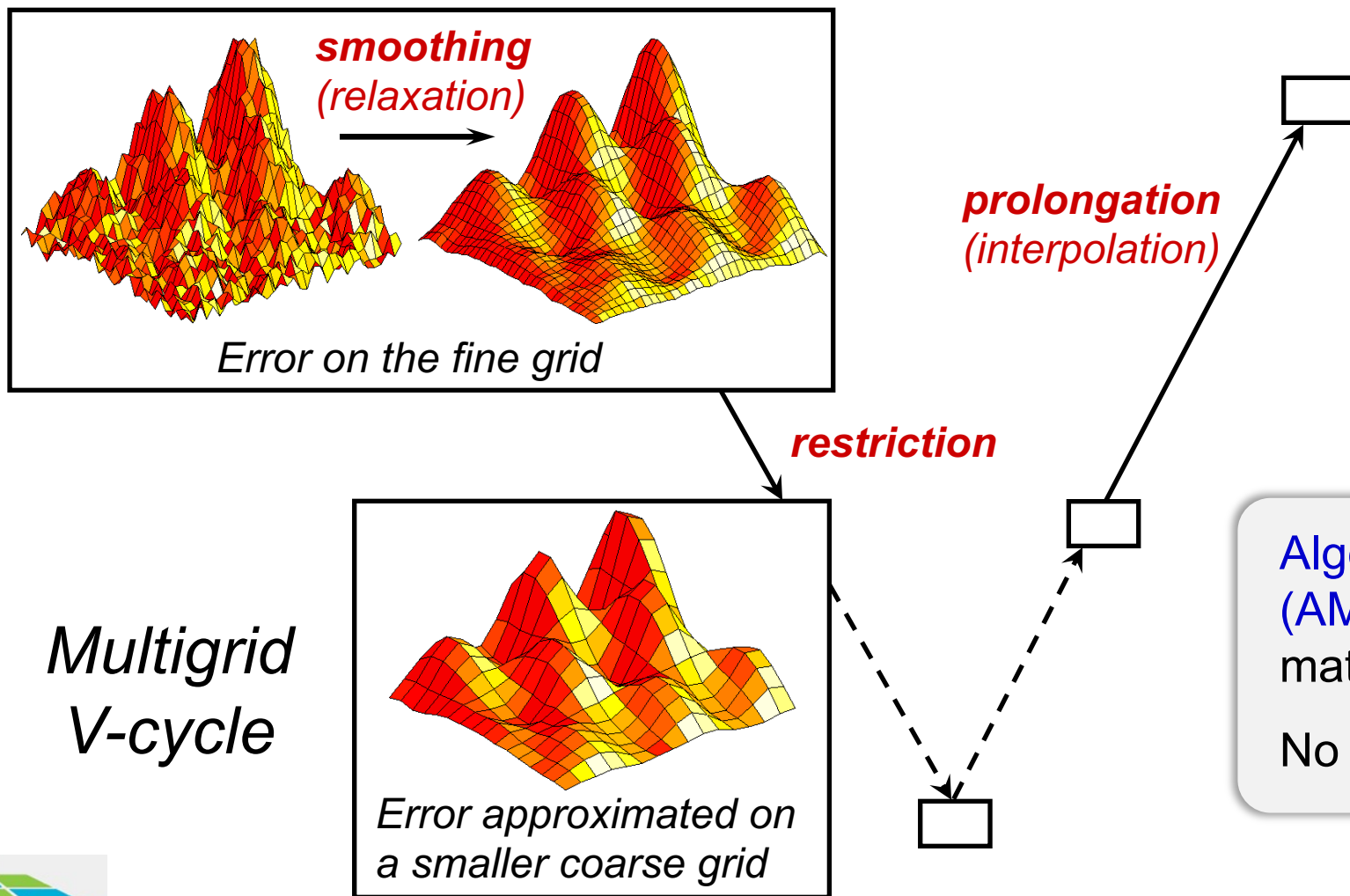# The *hypre* software library provides structured and unstructured multigrid solvers

- Used in many applications

Elasticity / Plasticity

Electro-magnetics

Magneto-hydrodynamics

Quantum Chromodynamics

Facial surgery

Subsurface simulations

- Displays excellent weak scaling and parallelization properties on BG/Q type architectures

FASTMATH

ECP EXASCALE COMPUTING PROJECT

Multigrid (MG) uses a sequence of coarse grids to accelerate the fine grid solution

smoothing
(relaxation)

Error on the fine grid

prolongation
(interpolation)

restriction

*Multigrid
V-cycle*

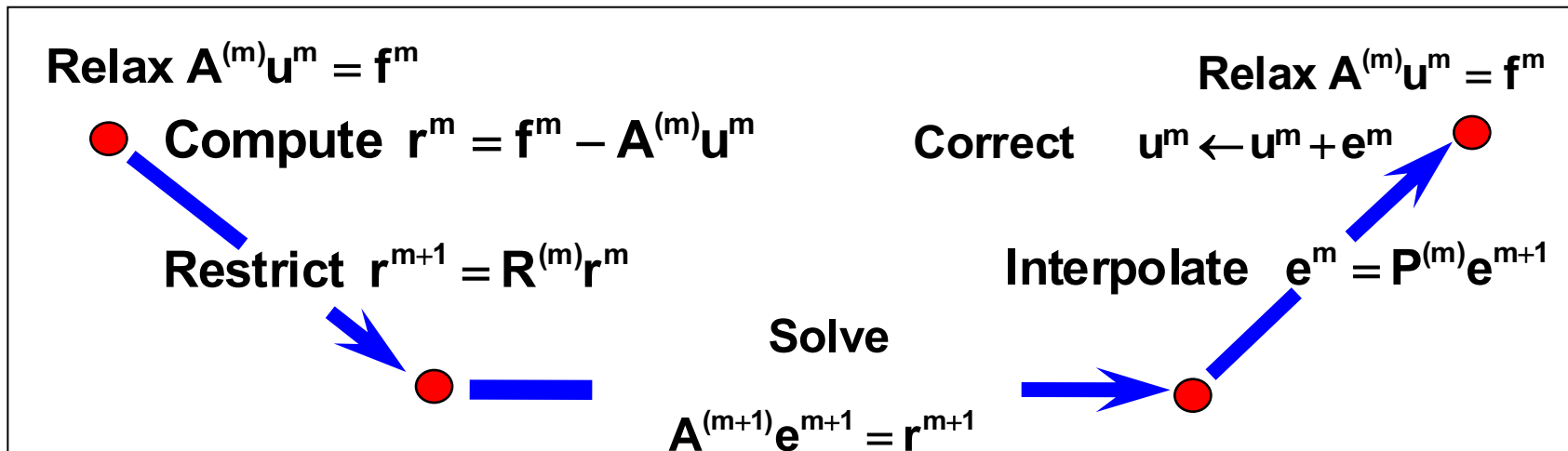*Error approximated on
a smaller coarse grid*

Algebraic multigrid (AMG) only uses matrix coefficients

No actual grids!

AMG Building Blocks

Setup Phase:

- Select coarse "grids"
- Define interpolation: $P^{(m)}, \quad m = 1, 2, ...$
- Define restriction: $R^{(m)}, \quad m = 1, 2, ...,$ often $R^{(m)} = (P^{(m)})^T$
- Define coarse-grid operators: $A^{(m+1)} = R^{(m)} A^{(m)} P^{(m)}$

Galerkin product

Solve Phase:

Relax $A^{(m)} u^m = f^m$

Compute $r^m = f^m - A^{(m)} u^m$

Restrict $r^{m+1} = R^{(m)} r^m$

Solve $A^{(m+1)} e^{m+1} = r^{m+1}$

Relax $A^{(m)} u^m = f^m$

Correct $u^m \leftarrow u^m + e^m$

Interpolate $e^m = P^{(m)} e^{m+1}$

15

FASTMATH

ECP EXASCALE COMPUTING PROJECT

# BoomerAMG is an algebraic multigrid method for unstructured grids

- Interface: `SStruct, IJ`

- Matrix Class: `ParCSR`

- Originally developed as a general matrix method (i.e., assumes given only A, x, and b)

- Various coarsening, interpolation and relaxation schemes

- Automatically coarsens "grids"

- Can solve systems of PDEs if additional information is provided

- Can also be used through PETSc and Trilinos

- Can now also be used on GPUs (CUDA, HIP)
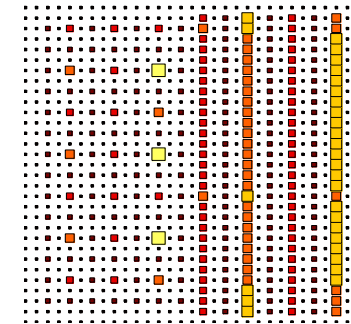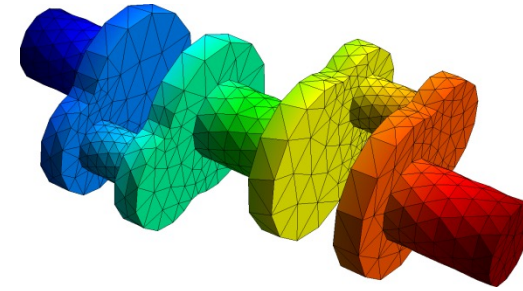
Complexity issues

- Coarse-grid selection in AMG can produce unwanted side effects

- Operator (RAP) "stencil growth" reduces efficiency

- For BoomerAMG, we will also consider complexities:
  - Operator complexity:
    $C_{op} = (\sum_{i=0}^{L} nnz\,(A_i))/nnz(A_0)$
  - Affects flops and memory
  - Generally, would like $C_{op} < 2$, close to 1

- Can control complexities in various ways
  - varying strength threshold
  - more aggressive coarsening
  - Operator sparsification (interpolation truncation, non-Galerkin approach)

- Needs to be done carefully to avoid excessive convergence deterioration


AMG Communication patterns, 128 cores
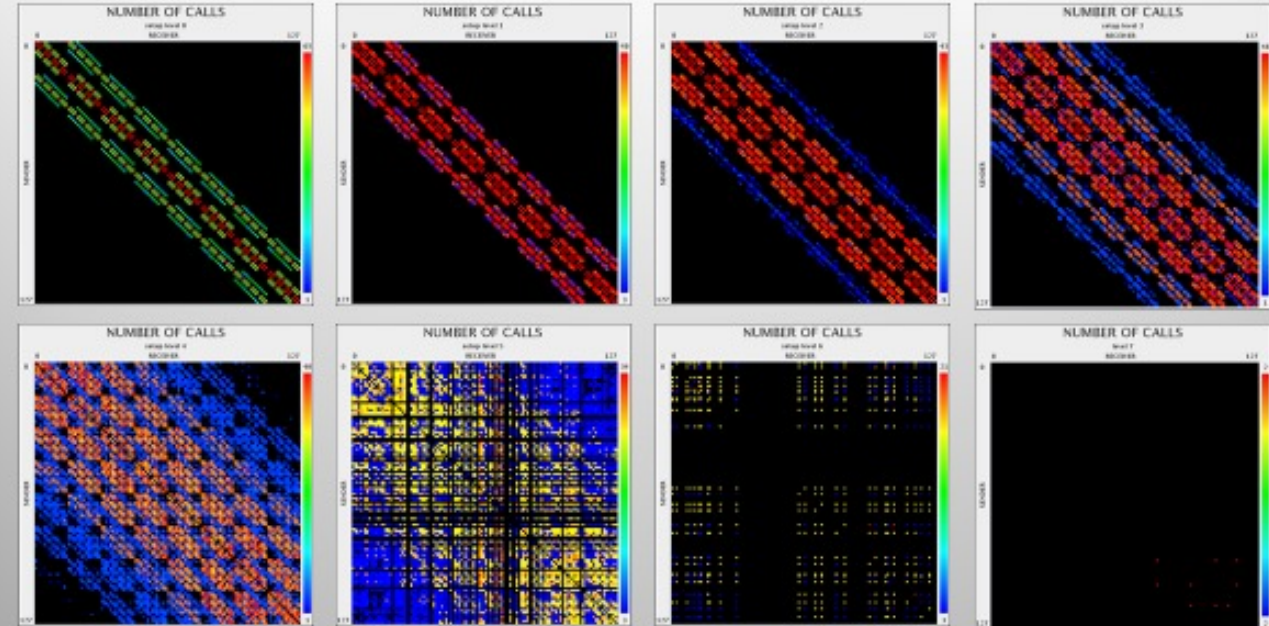
Performance degradation caused by increased communication complexity on coarser grids !

Lawrence Livermore National Laboratory

ParCSRMatrix data structure

- Based on compressed sparse row (CSR) data structure

- Consists of two CSR matrices:
  - One containing local coefficients connecting to local column indices
  - The other (Offd) containing coefficients with column indices pointing to off processor rows

- Also contains a mapping between local and global column indices for Offd

- Requires much indirect addressing, integer computations, and computations of relationships between processes etc,

$A =$

Proc 0
Proc 1

Proc p
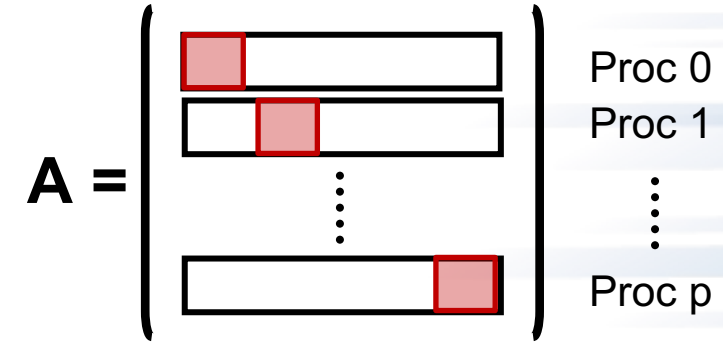
SMG and PFMG are semicoarsening multigrid methods for structured grids

- Interface: `Struct`

- Matrix Class: `Struct`

- SMG uses plane smoothing in 3D, where each plane "solve" is affected by one 2D V-cycle

- SMG is very robust

- PFMG uses simple pointwise smoothing, and is less robust

- Note that stencil growth is limited for SMG and PFMG (to at most 27 points per stencil in 3D)

- Constant-coefficient versions

- Can be used on GPUs (CUDA, HIP, RAJA, Kokkos)
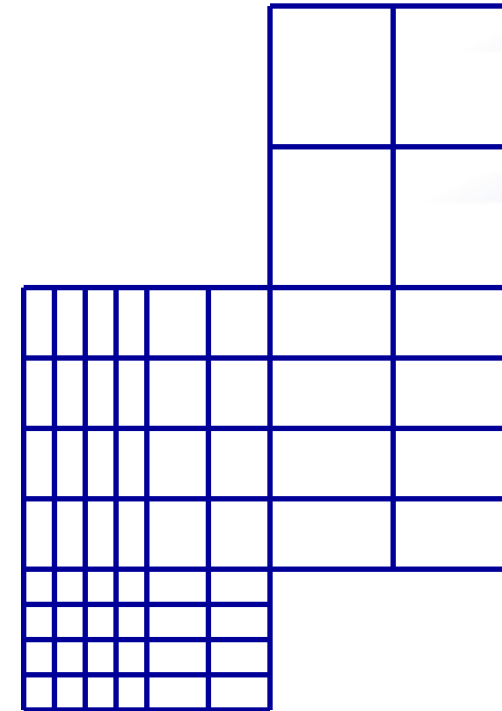
PFMG is an algebraic multigrid method for structured grids

- Matrix defined in terms of grids and stencils

- Uses semicoarsening

- Simple 2-point interpolation
  → limits stencil growth to at most 9pt (2D), 27pt (3D)

- Optional non-Galerkin approach (Ashby, Falgout), uses geometric knowledge, preserves stencil size

- Pointwise smoothing

- Highly efficient for suitable problems
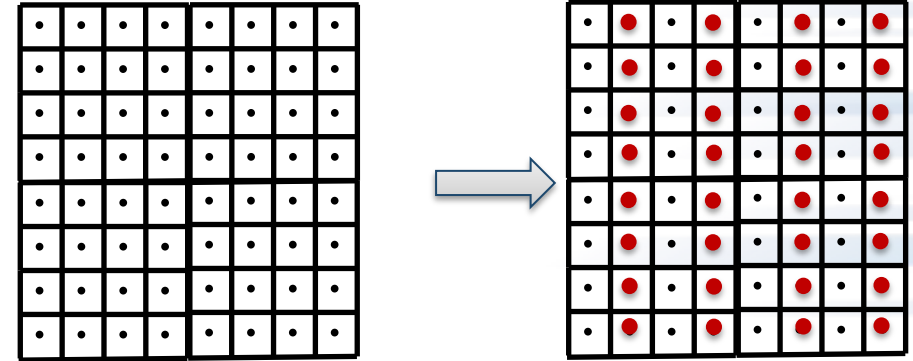
Structured-Grid System Interface (`Struct`)

- Appropriate for scalar applications on structured grids with a fixed stencil pattern

- Grids are described via a global d-dimensional *index space* (singles in 1D, tuples in 2D, and triples in 3D)

- A *box* is a collection of cell-centered indices, described by its "lower" and "upper" corners

- The grid is a collection of boxes

- Matrix coefficients are defined via stencils

$$\begin{bmatrix} & S4 & \\ S1 & S0 & S2 \\ & S3 & \end{bmatrix} = \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}$$

Index Space

(6,11)

(-3,2)

(7,3)

(15,8)
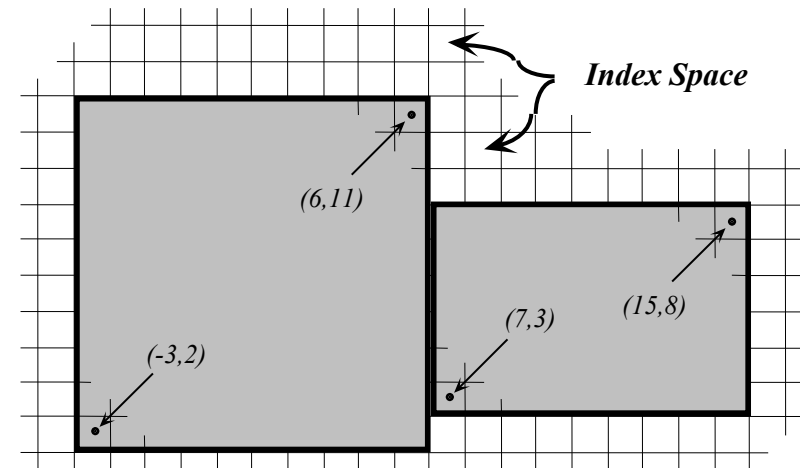
StructMatrix data structure

- Stencil
$$\begin{bmatrix} & S4 & \\ S1 & S0 & S2 \\ & S3 & \end{bmatrix} = \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}$$

- Grid boxes: [(-3,1), (-1,2)] [(0,1), (2,4)]

- Data Space: grid boxes + ghost layers:
[(-4,0), (0,3)] , [(-1,0), (3,5)]

- Data stored

| S0 | S1 | S2 | S3 | S4 | S0 | S1 | S2 | S3 | S4 |
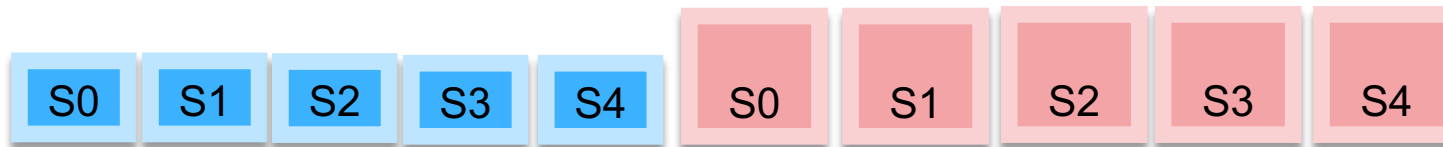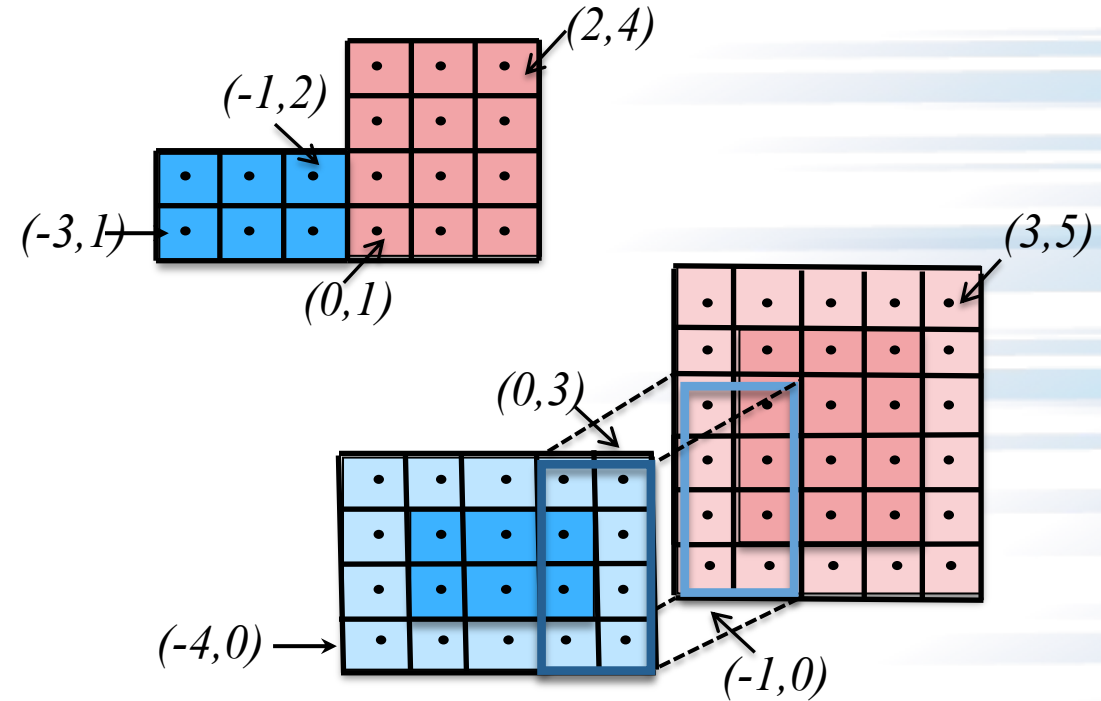
- **Operations applied to stencil entries per box (corresponds to matrix (off) diagonals from a matrix point of view)**
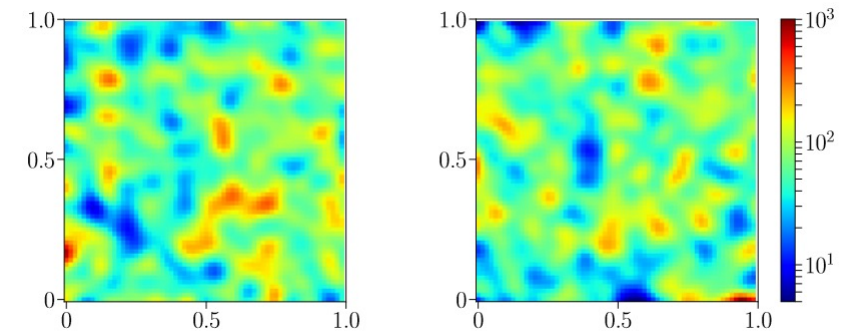
Algebraic multigrid as preconditioner

- Generally algebraic multigrid methods are used as preconditioners to Krylov methods, such as conjugate gradient (CG) or GMRES

- This often leads to additional performance improvements
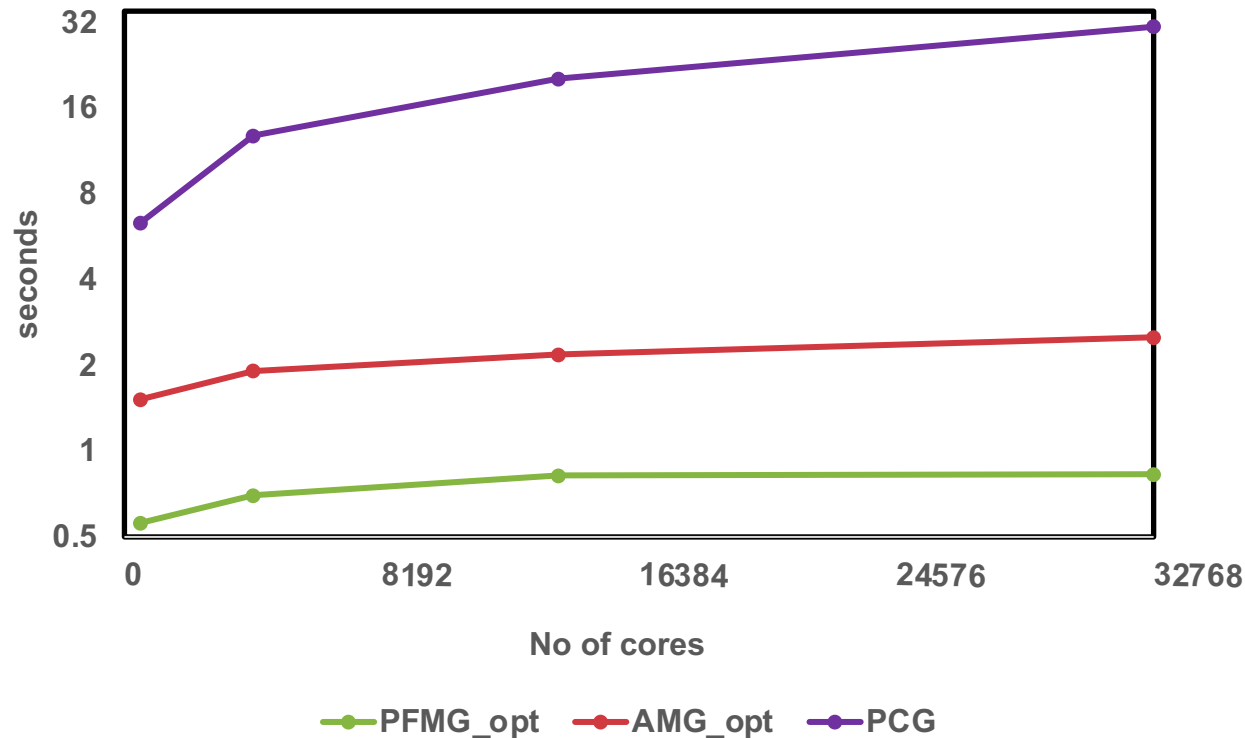
Classic porous media diffusion problem:
$$-\nabla \cdot \kappa \nabla u = f$$
with κ having jumps of 2-3 orders of magnitude

Weak scaling: 32x32x32 grid points per core, BG/Q

Porting to GPUs required inclusion of new programming models and different strategies for structured/unstructured interfaces

- Strategy for structured interface and solvers
  - Include new programming models (CUDA, HIP, RAJA, Kokkos, OMP, and SYCL) in hypre_BoxLoops (macros that operate on data in loops).

- Strategy for unstructured interface and solversCSR-based matrix data structures
  - Modularize into smaller chunks/kernels to be ported to CUDA for Nvidia GPUS initially
  - Convert CUDA kernels to HIP for AMD GPUs and SYCL for Intel GPUs
  - Develop new algorithms for portions not suitable for GPUs (interpolation operators, smoothers) → different defaults for CPU and GPU use
  - Various special solvers (e.g., Maxwell solver AMS, ADS, AME, pAIR, MGR) built on BoomerAMG benefit from this strategy

Structured multigrid methods perform significantly better than unstructured ones on CPUs and - even more - on GPUs

Total times .MG/PCG for a 7pt 3D Laplace problem on a n x n x n grid

Legend: AMG-CPU (blue dashed), PFMG-CPU (red dashed), PFMG-GPU (red solid), AMG-GPU (blue solid)
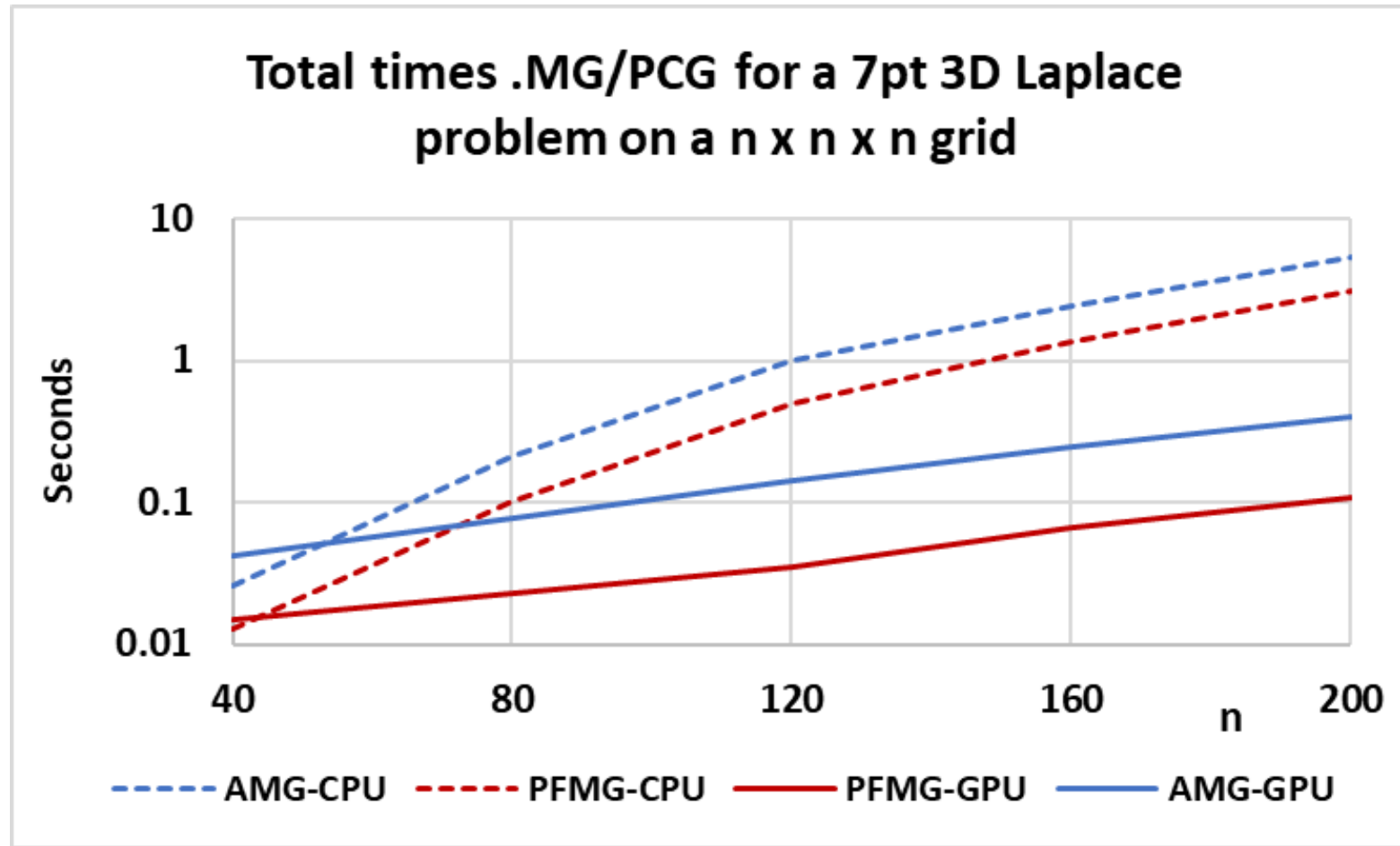
ThetaGPU
GPU: 1 Nvidia A100
CPU: 16 MPI tasks

Used optimal settings for AMG, which are different for CPU and GPU!

Speedups at n=200

Speedup GPU/CPU	CPU Speedup PFMG/AMG
13.2	1.7
Speedup GPU/CPU	GPU Speedup PFMG/AMG
28.5	3.8

Most gains of PFMG over AMG in setup phase
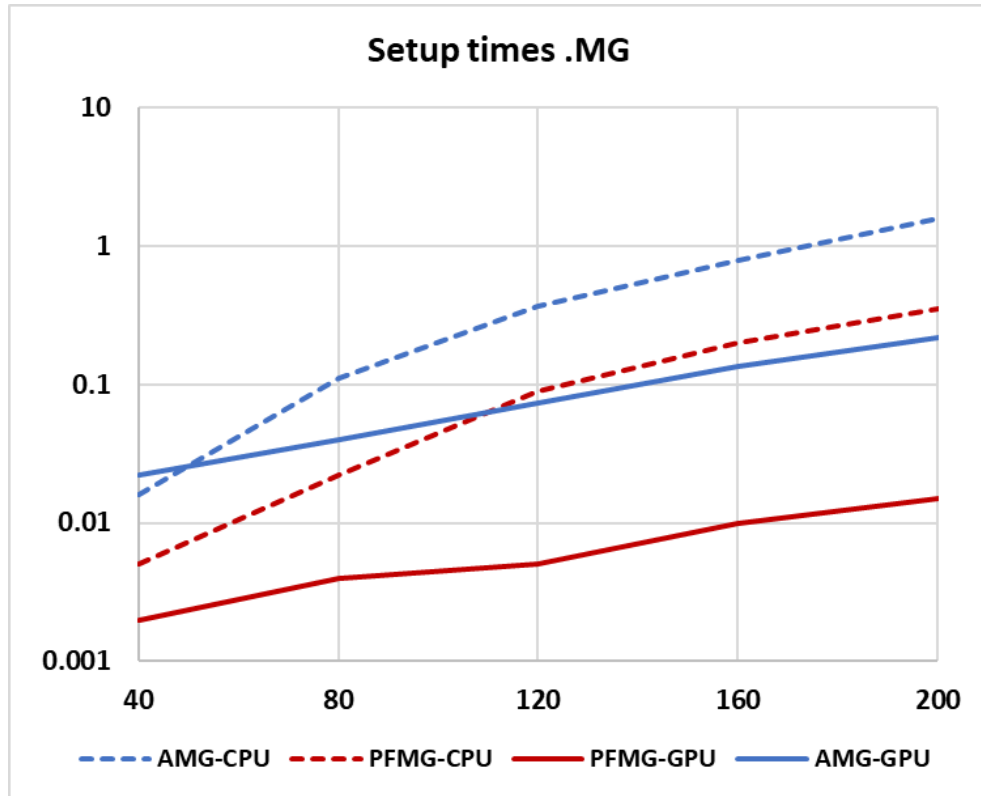
Setup times .MG

Solve times .MG/PCG

Legend: AMG-CPU — PFMG-CPU — PFMG-GPU — AMG-GPU

Speedup GPU/CPU	Speedup GPU/CPU	CPU Speedup PFMG/AMG	GPU Speedup PFMG/AMG
7.2	23.3	4.5	14.5
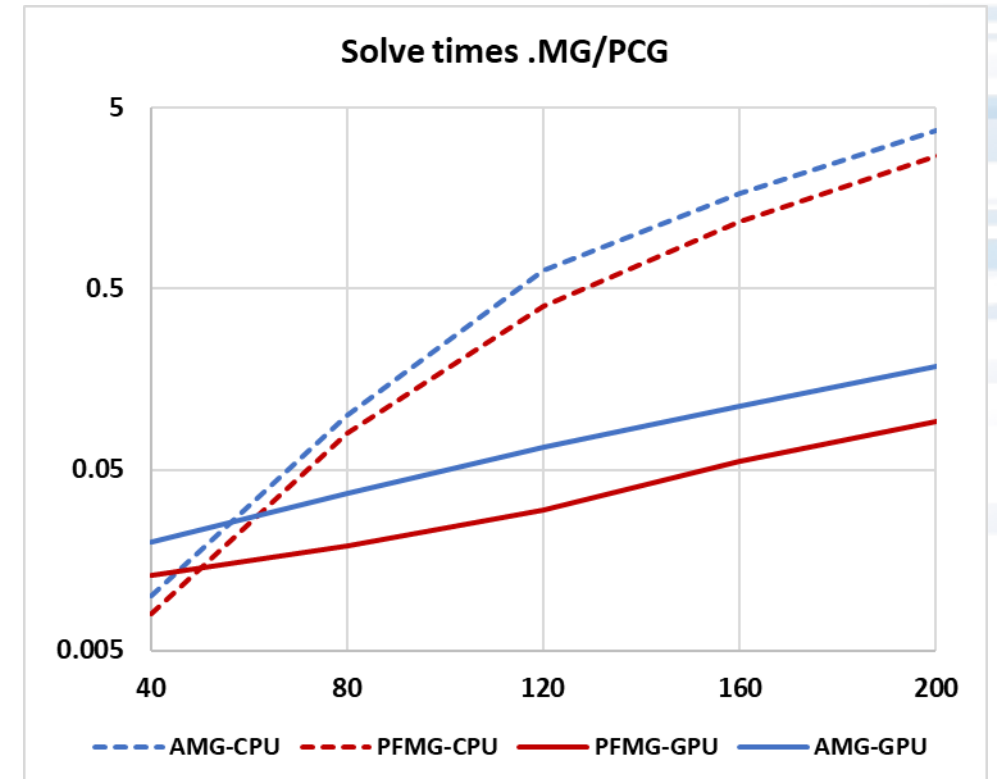
Speedup GPU/CPU	Speedup GPU/CPU	CPU Speedup PFMG/AMG	GPU Speedup PFMG/AMG
20.1	29.3	1.4	2.0

Successfully used hypre on Frontier (AMD GPUs) for solving complex multiphysics simulations

Single-phase flow (Poisson-like problem)
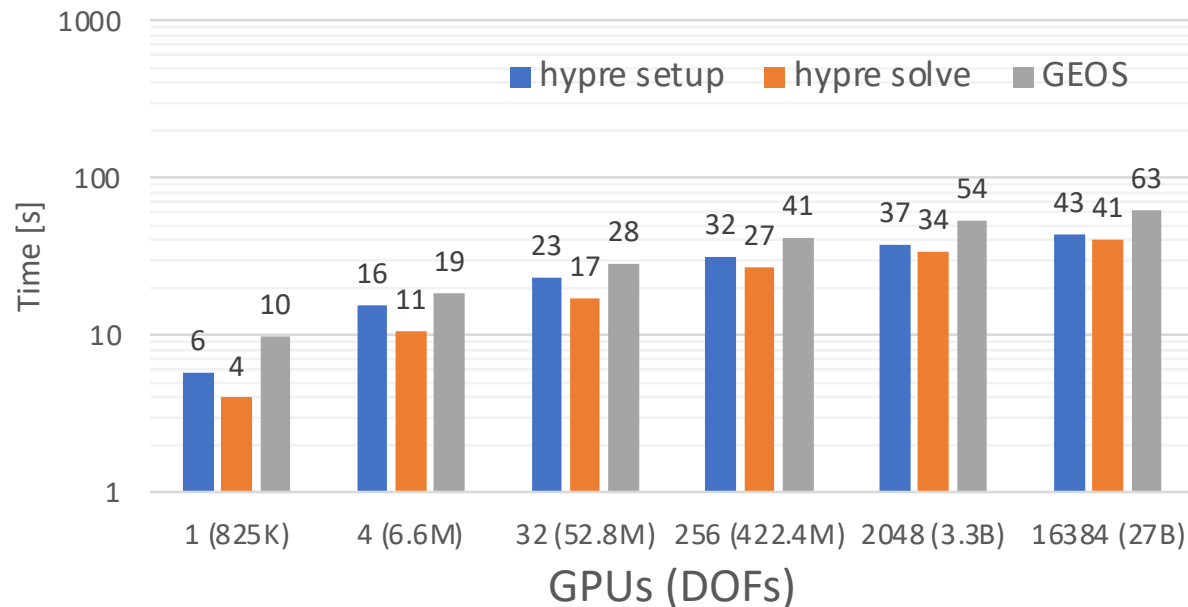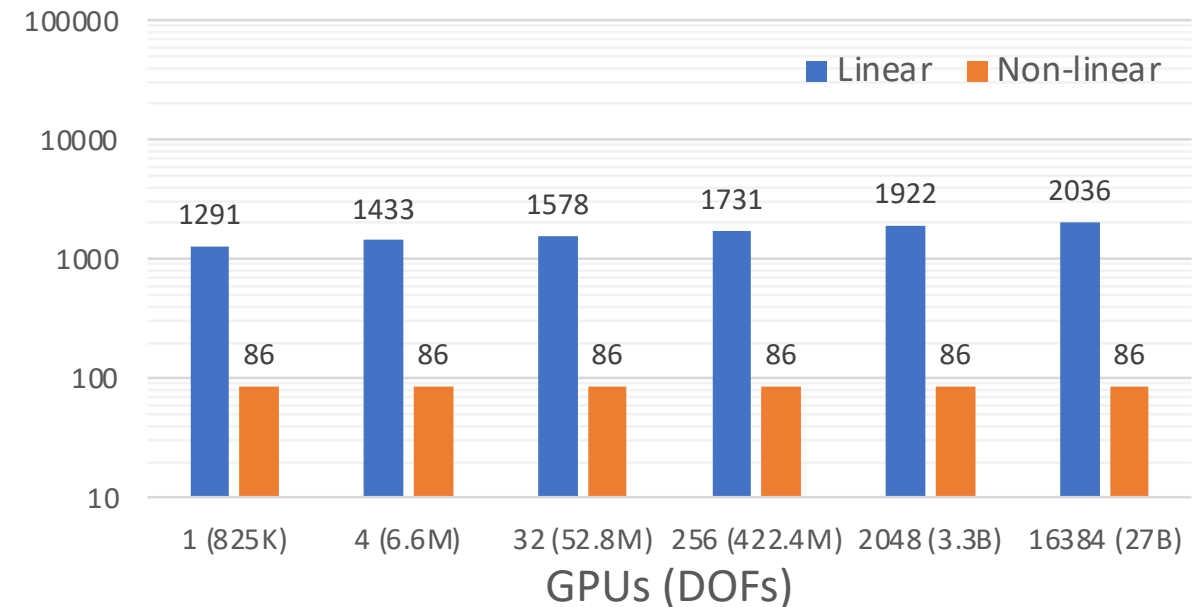
- Weak scaling with BoomerAMG/GMRES(50)
- Time complexity ~ O(log(N)); Iteration counts ~ O(1).

Frontier (AMD GPUs) results - Solved system with 80B DOFs using less than 25% of the machine

Mechanics (linear elasticity problem)
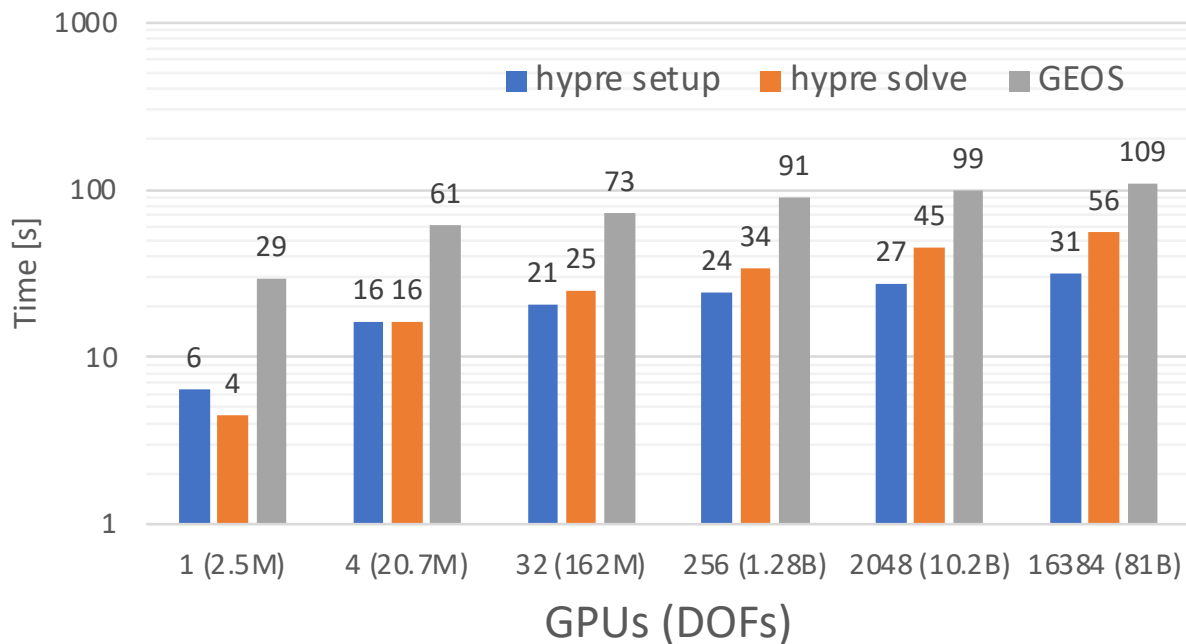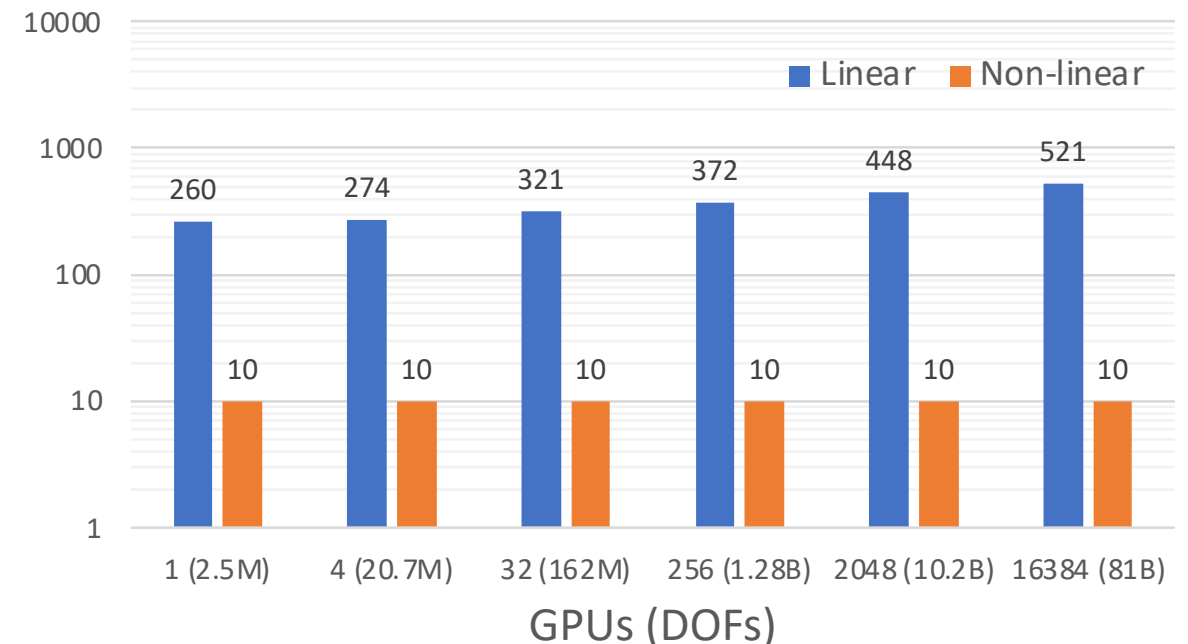
- Weak scaling with BoomerAMG/GMRES(40)

Hands-on Exercises: Algebraic multigrid methods, including comparing GPU to CPU performance

- Go to https://xsdk-project.github.io/MathPackagesTraining2023/lessons/krylov_amg_hypre/

- Poisson equation: $-\Delta\varphi$ = RHS

    with Dirichlet boundary conditions $\varphi = 0$

- Grid: cube
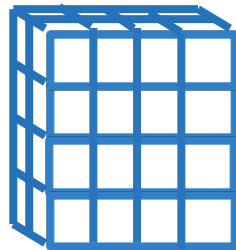
- Finite difference discretization:
    - Central differences for diffusion term
    - 7-point stencil
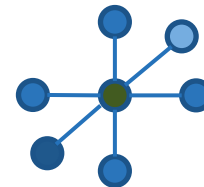
Thank you!

CASC
Center for Applied
Scientific Computing

**Lawrence Livermore
National Laboratory**