Argonne Training Program on Extreme-Scale Computing

ATPESC 2025

Krylov Solvers and Algebraic Multigrid with *hypre*

Daniel Osei-Kuffuor, Ulrike Meier Yang
Lawrence Livermore National Laboratory

July 27 – August 8, 2025

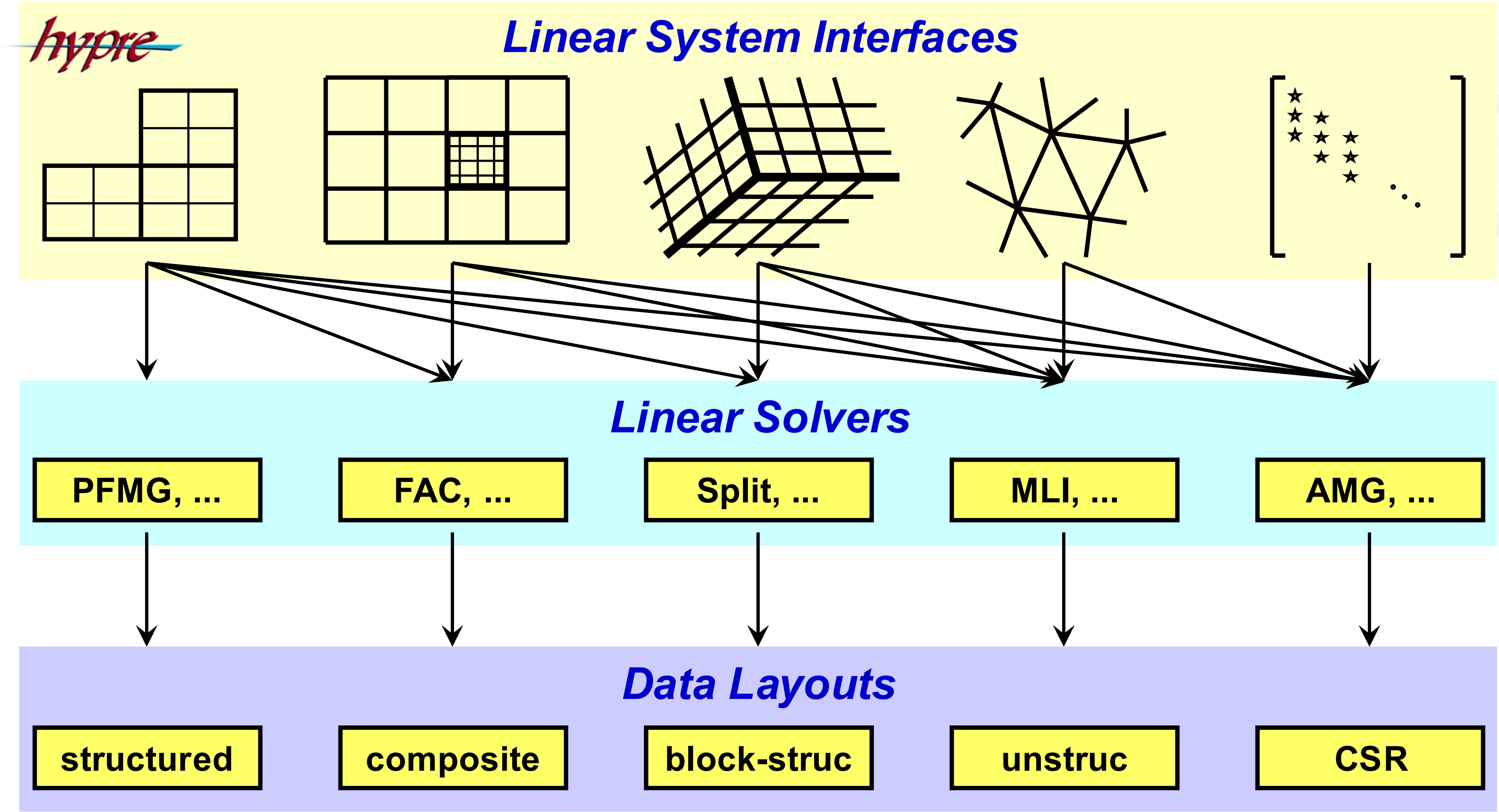
Outline

- Interfaces and Data Structures
  - IJ interface / ParCSR data structure
  - Structured interface / Struct data structure
- Iterative Solvers
  - Krylov Solvers
  - Multigrid solvers
- Some hands-on examples

https://www.github.com/LLNL/hypre

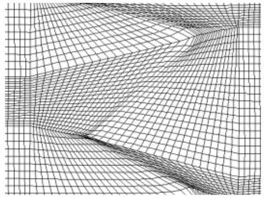
	System Interfaces		
Solvers	**Struct**	**SStruct**	**IJ**
Jacobi	X	X	
SMG	X	X	
PFMG	X	X	
Split		X	
SysPFMG		X	
FAC		X	
Maxwell		X	
BoomerAMG		X	X
AMS		X	X
ADS		X	X
MLI		X	X
MGR			X
FSAI			X
ParaSails		X	X
ILU			X
Euclid		X	X
PILUT		X	X
PCG	X	X	X
GMRES	X	X	X
FlexGMRES	X	X	X
LGMRES	X	X	X
BiCGSTAB	X	X	X
Hybrid	X	X	X
LOBPCG	X	X	X

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

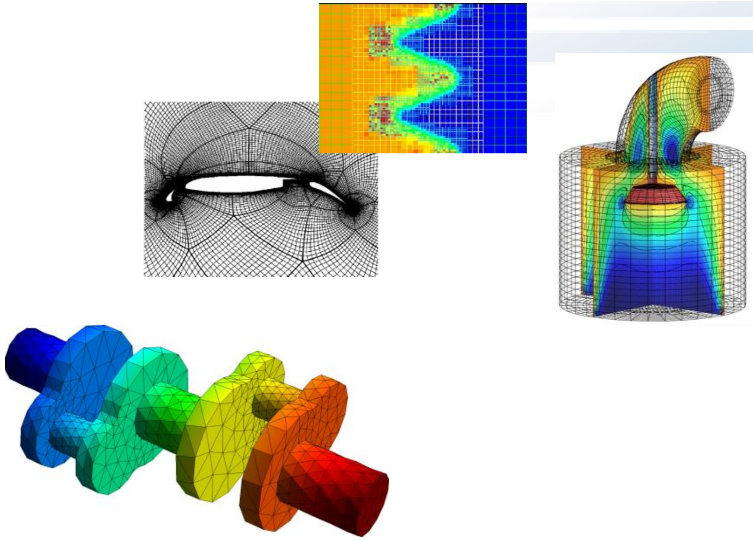


hypre supports these system interfaces

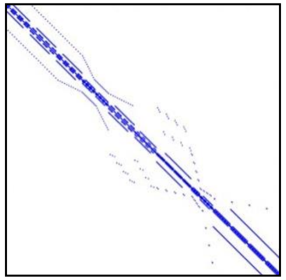
- Structured-Grid (`Struct`)
  - *logically rectangular grids*



- 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*

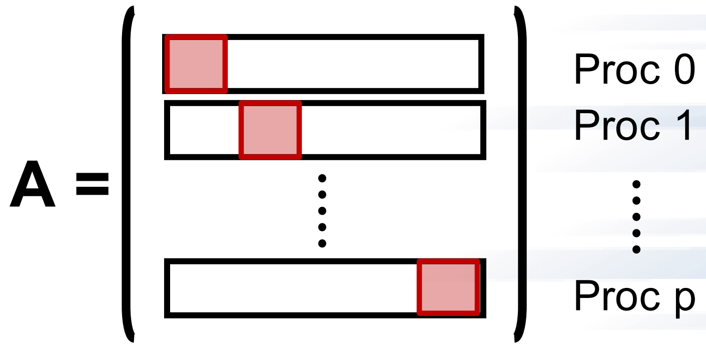


Why multiple interfaces? The key points

- 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

ParCSRMatrix data structure

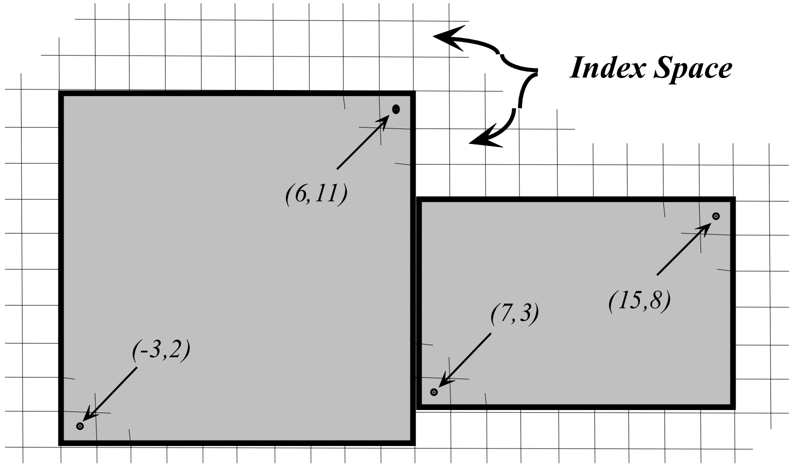
- Based on compressed sparse row (CSR) data structure
- Consists of two CSR matrices:
  - One containing local coefficients connecting to local column indices
  - The other (Offd) containing coefficients with column indices pointing to off processor rows
- Also contains a mapping between local and global column indices for Offd
- Requires much indirect addressing, integer computations, and computations of relationships between processes etc,



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

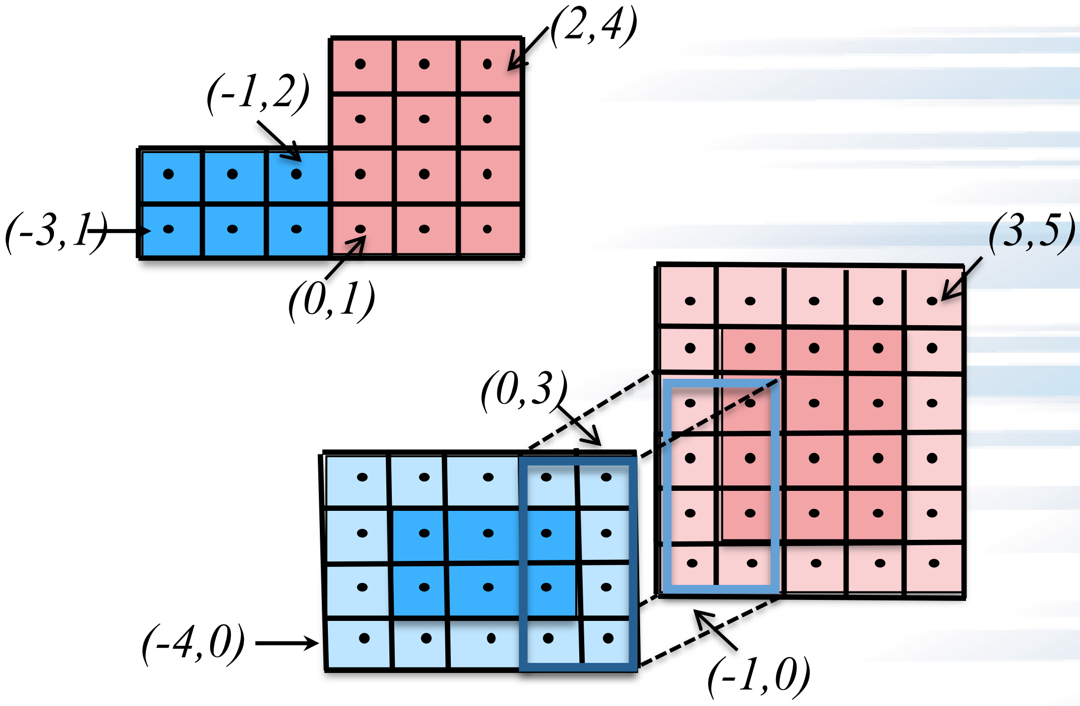
$$\begin{bmatrix} & S4 & \\ S1 & S0 & S2 \\ & S3 & \end{bmatrix} = \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}$$



StructMatrix data structure

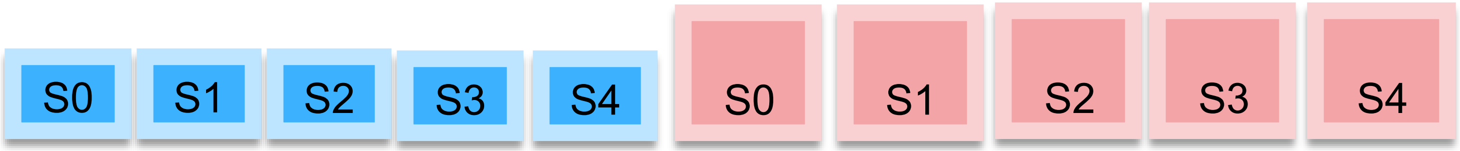
- Stencil $\begin{bmatrix} & S4 & \\ S1 & S0 & S2 \\ & S3 & \end{bmatrix} = \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}$

- 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



- **Operations applied to stencil entries per box (corresponds to matrix (off) diagonals from a matrix point of view)**

Iterative Solvers

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

- Dense direct solvers (e.g., Gaussian elimination) too expensive (Sparse direct solvers offer better complexity)

- 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)

- $x^{n+1} = x^n + B(b - Ax^n)$
- $\Rightarrow x^{n+1} = \sum_{i=0}^{n} \alpha_i (BA)^i Bb$
- $x^{n+1} \in K^n = span\{Bb, (BA)Bb, (BA)^2 Bb, \ldots, (BA)^n Bb\}$

  Krylov space

- Construct a new basis for K^n through orthonormalization

$$\{q_0 = \frac{Bb}{||Bb||}, q_1, \ldots, q_n\}$$

- q_i also called search directions
- Now optimize by defining x^{n+1} through

$$\min_{x^{n+1} \in K^n} ||B(Ax^{n+1} - b)||$$

Some comments on GMRES

- GMRES consists of fairly simple operations:
  - Inner products and norms (global reductions)
  - Vector updates (embarrassingly parallel)
  - Matvecs (nearest neighbor updates)
  - Residual decreases monotonically at each step
- 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

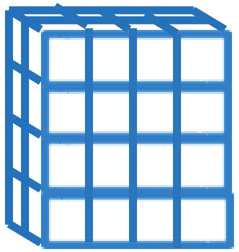
- Conjugate Gradient (CG)
  - For symmetric positive definite matrices
  - Possesses like GMRES an orthogonality property
  - Uses a three-term recurrence
  - Requires only two inner products and a norm per iteration
- BiCGSTAB (BiConjugate Gradient Stabilized)
  - Like CG uses a three-term recurrence 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 (First Set of Runs)

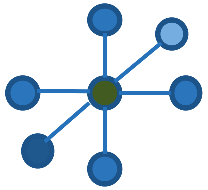
- Go to https://xsdk-project.github.io/MathPackagesTraining2025/lessons/krylov_amg_hypre/
- Important: export MPICH_GPU_SUPPORT_ENABLED=0
- Poisson equation: $-\Delta\varphi = \text{RHS}$

  with Dirichlet boundary conditions $\varphi = 0$

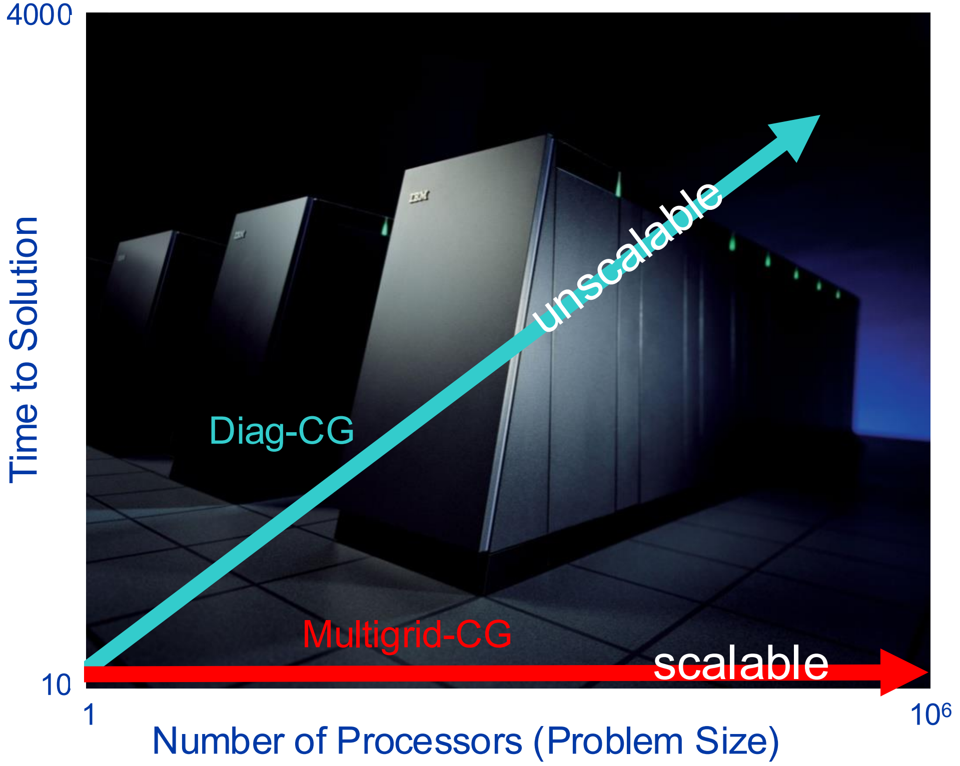
- Grid: cube



- Finite difference discretization:
  - Central differences for diffusion term
  - 7-point stencil

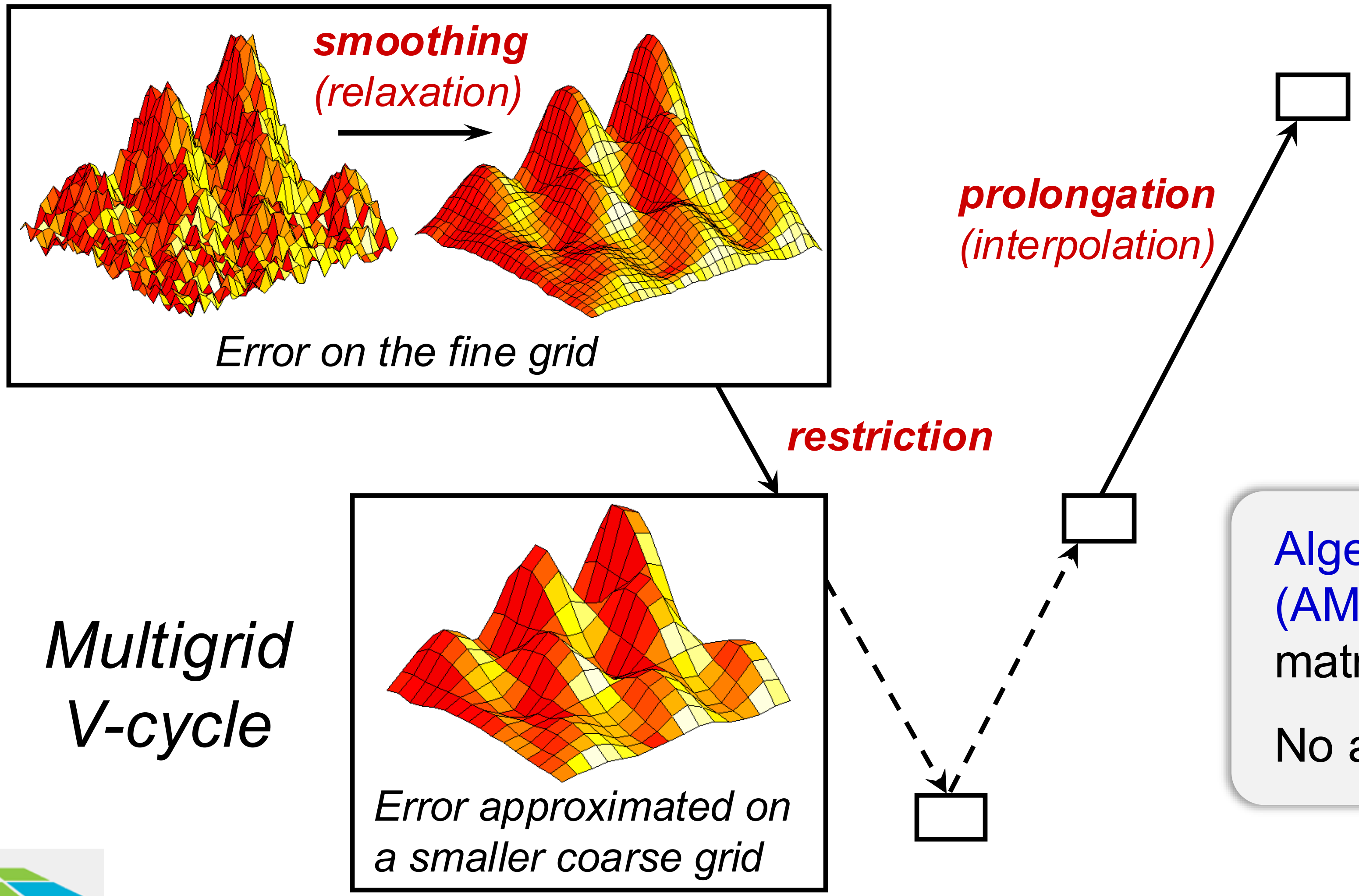


Multigrid linear solvers are optimal ($O(N)$ operations), and hence have good scaling potential



- Weak scaling – want constant solution time as problem size grows in proportion to the number of processors

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



Algebraic multigrid (AMG) only uses matrix coefficients

No actual grids!

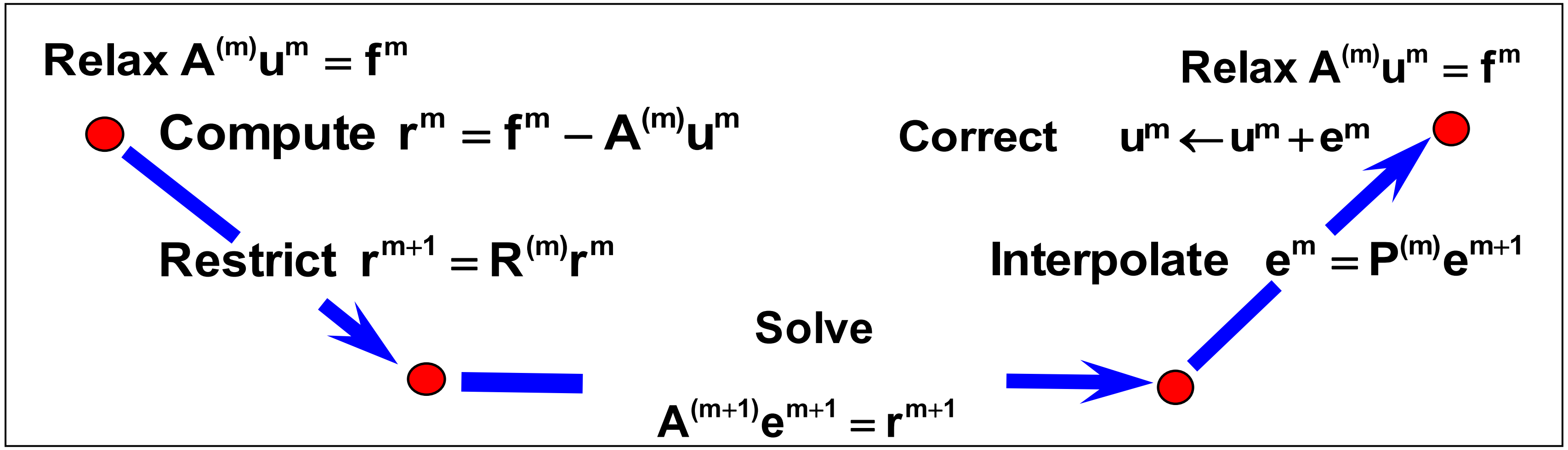
AMG Building Blocks

Setup Phase:

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

  Galerkin product

Solve Phase:

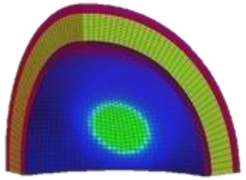


Multigrid software

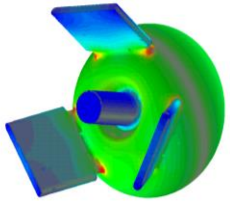
- ML, MueLu included in Trilinos
- GAMG in PETSc
- The hypre library provides various algebraic multigrid solvers, including multigrid solvers for special problems e.g., Maxwell equations, …
- …
- All of these provide different flavors of multigrid and provide excellent performance for suitable problems
- Focus here on hypre

The *hypre* software library provides structured and unstructured multigrid solvers

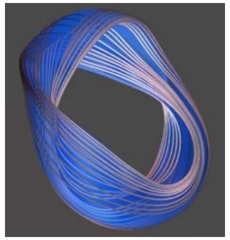
- Used in many applications



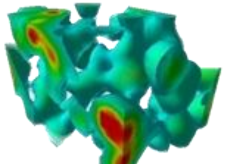
Elasticity/ Plasticity



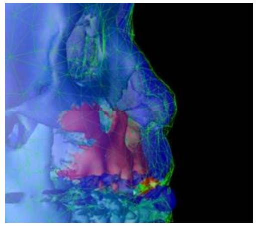
Electro-magnetics



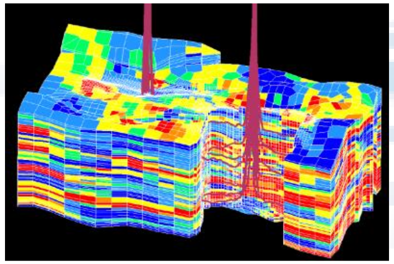
Magneto-hydrodynamics



Quantum Chromodynamics

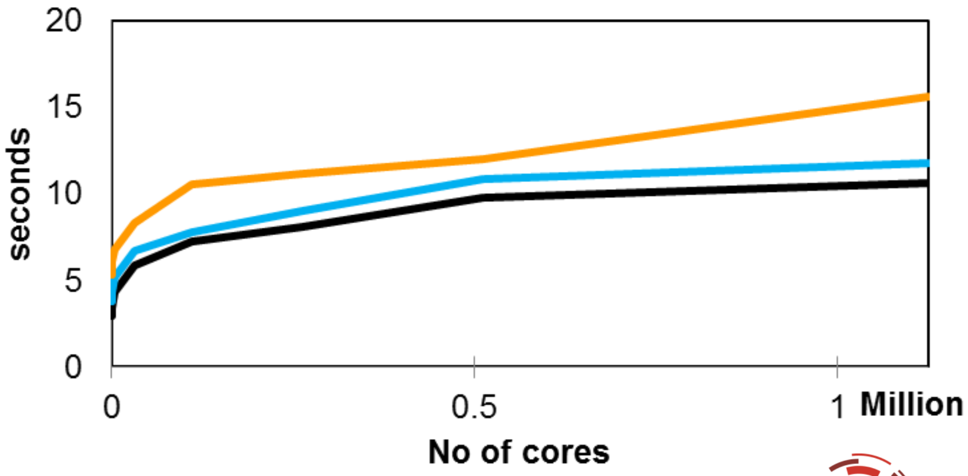


Facial surgery



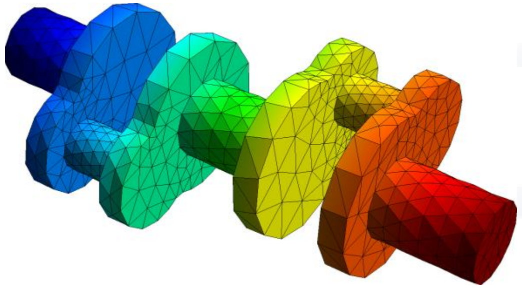
Subsurface simulations

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

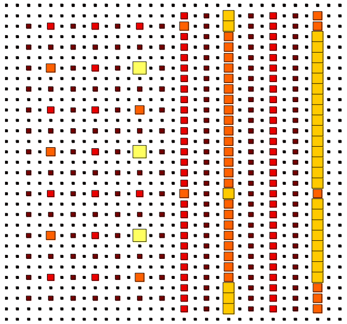


BoomerAMG is an algebraic multigrid method for unstructured grids

- Interface: `SStruct, IJ`
- Matrix Class: `ParCSR`

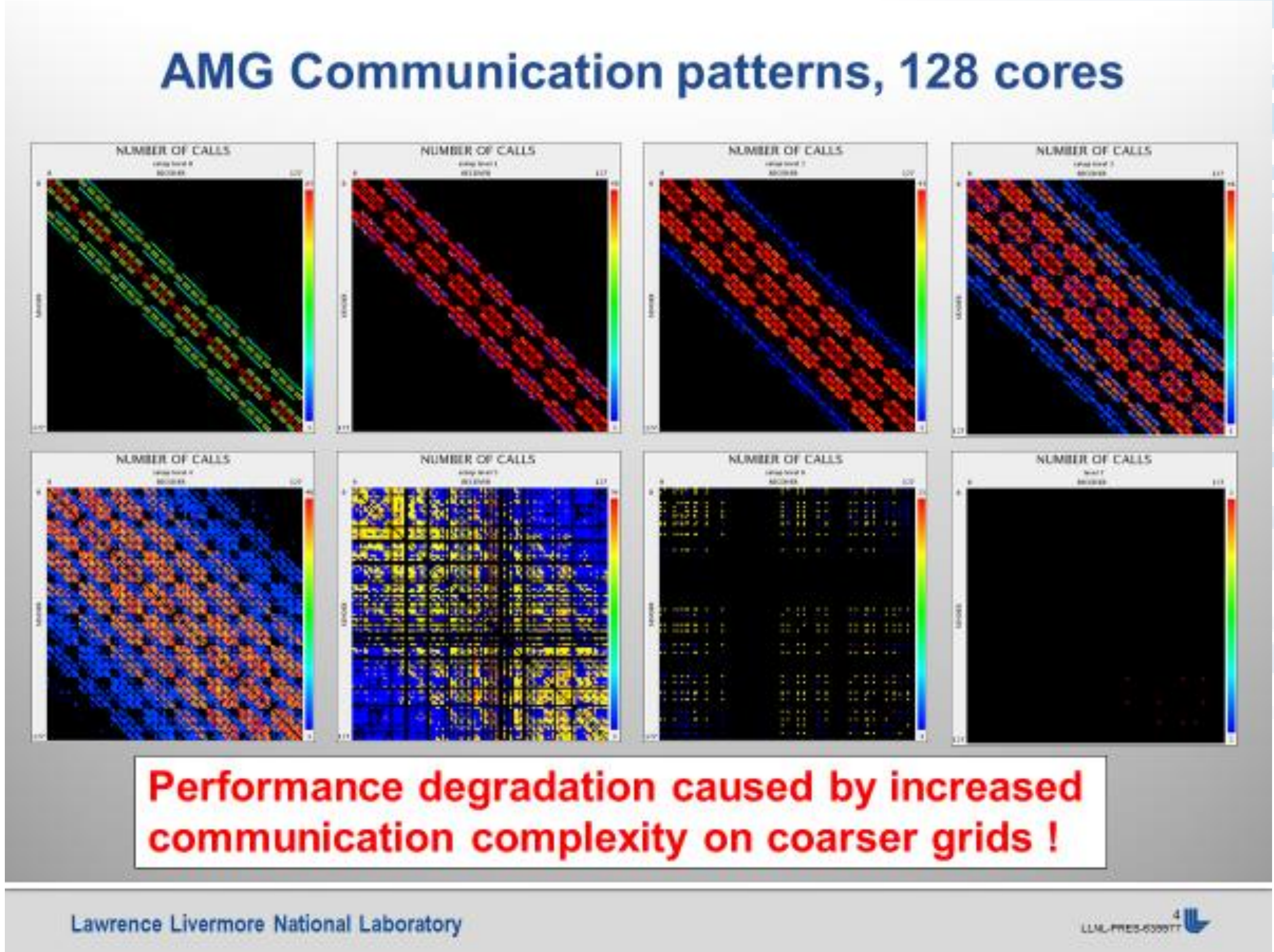


- Originally developed as a general matrix method (i.e., assumes given only A, x, and b)
- Various coarsening, interpolation and relaxation schemes
- Automatically coarsens “grids”
- Can solve systems of PDEs if additional information is provided
- Can also be used through PETSc and Trilinos
- Can be used on GPUs (CUDA, HIP, SYCL)



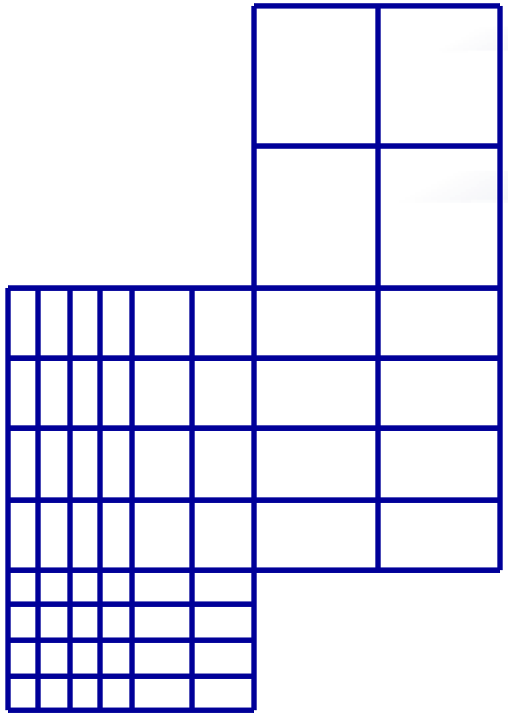
Complexity issues

- 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



SMG and PFMG are semicoarsening multigrid methods for structured grids

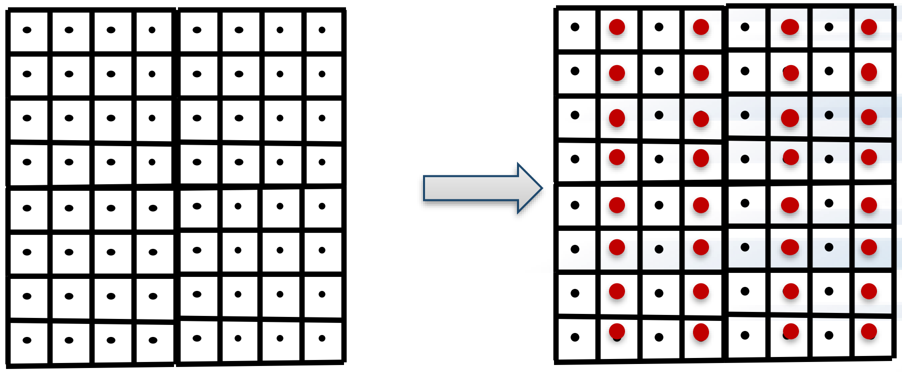
- Interface: `Struct`
- Matrix Class: `Struct`
- SMG uses plane smoothing in 3D, where each plane "solve" is affected by one 2D V-cycle
- SMG is very robust
- PFMG uses simple pointwise smoothing, and is less robust
- Note that stencil growth is limited for SMG and PFMG (to at most 27 points per stencil in 3D)
- Constant-coefficient versions
- Can be used on GPUs (CUDA, HIP, SYCL, RAJA, Kokkos)



FASTMATH

EXASCALE COMPUTING PROJECT

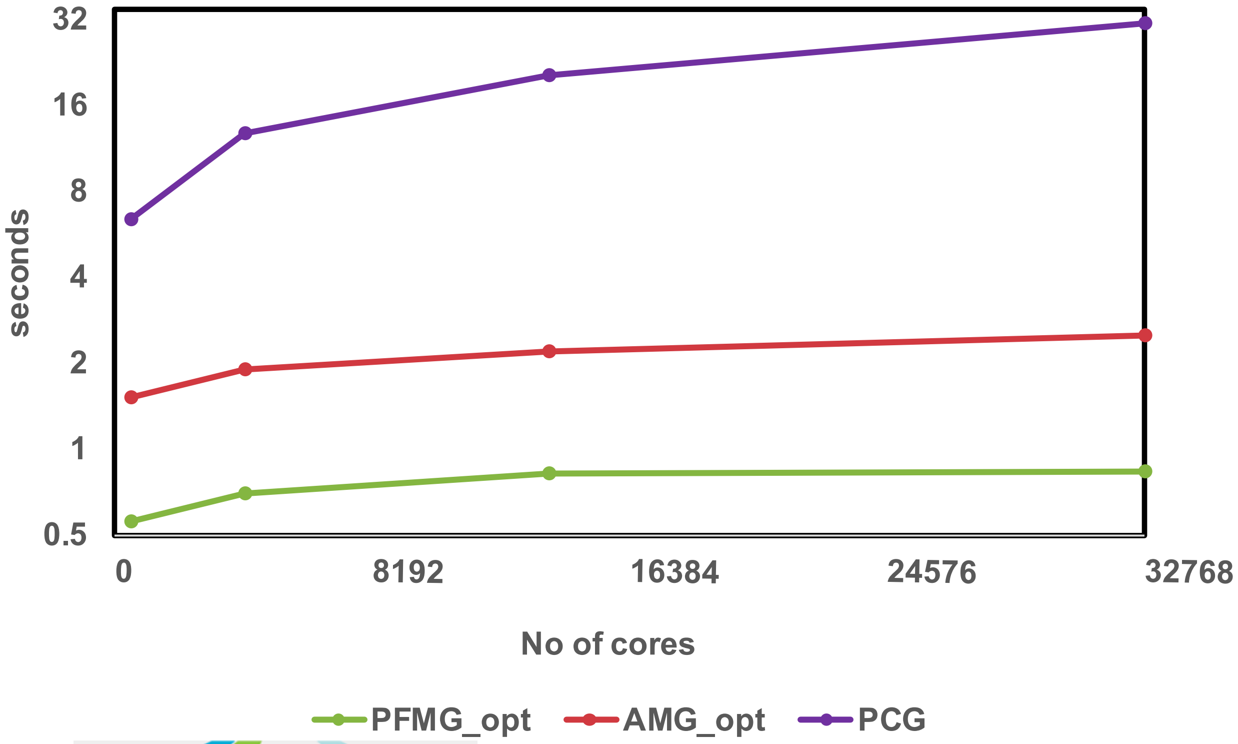
PFMG is an algebraic multigrid method for structured grids



- 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

Algebraic multigrid as preconditioner

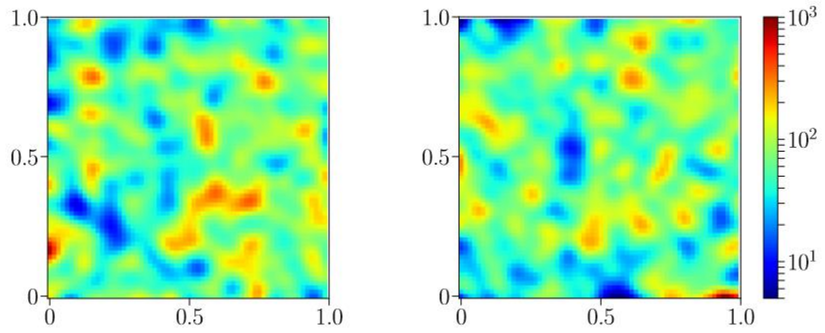
- Generally algebraic multigrid methods are used as preconditioners to Krylov methods, such as conjugate gradient (CG) or GMRES
- This often leads to additional performance improvements



Classic porous media diffusion problem:

$$-\nabla \cdot \kappa \nabla \mathrm{u} = \mathrm{f}$$

with κ having jumps of 2-3 orders of magnitude



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

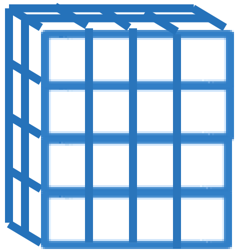
FASTMATH

ECP EXASCALE COMPUTING PROJECT

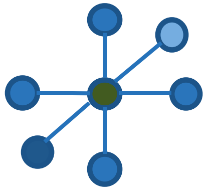
Hands-on Exercises: Algebraic multigrid (Second Set of Runs)

- Go to https://xsdk-project.github.io/MathPackagesTraining2025/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

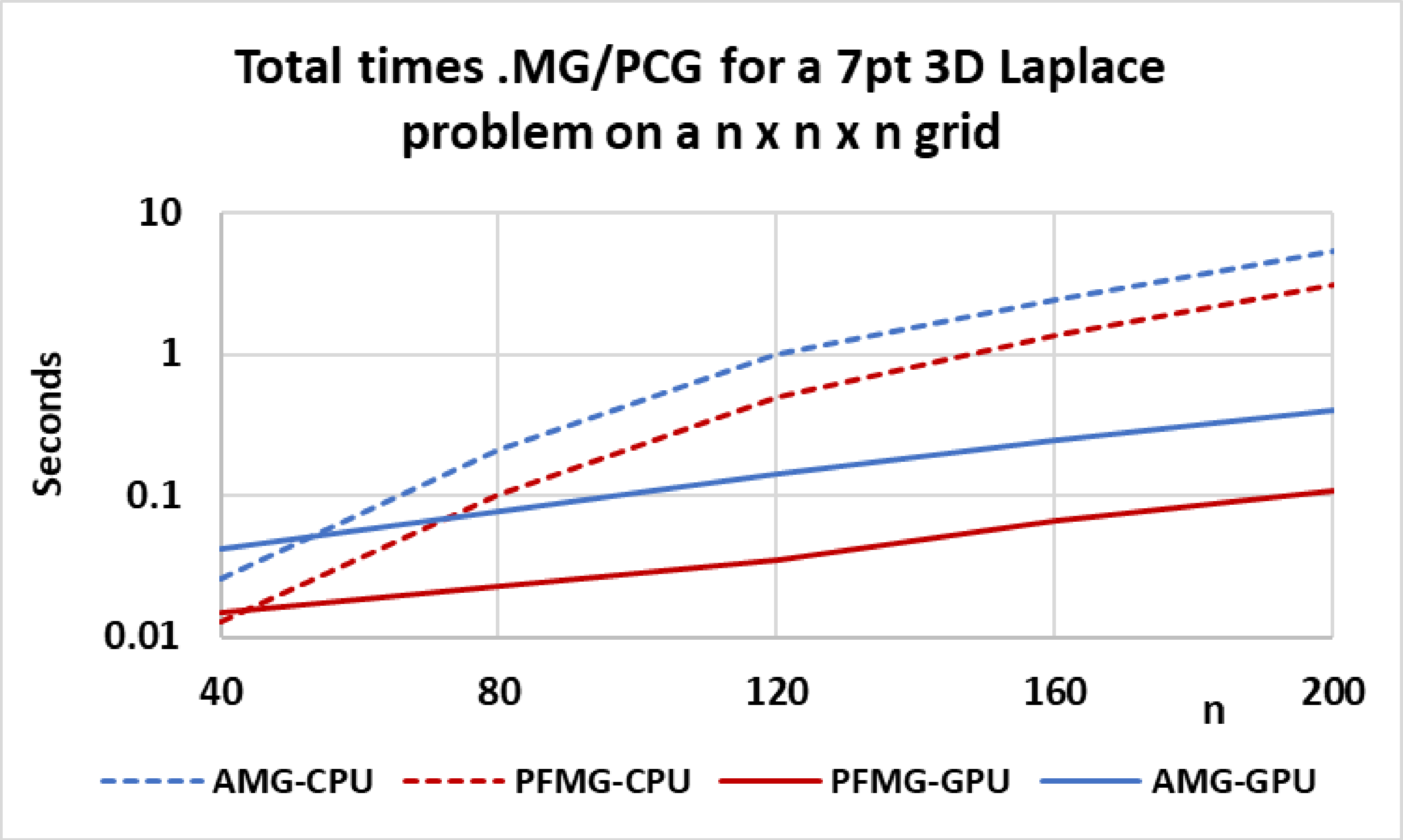


Porting to GPUs required inclusion of new programming models and different strategies for structured/unstructured interfaces

- 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 solvers (CSR-based 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



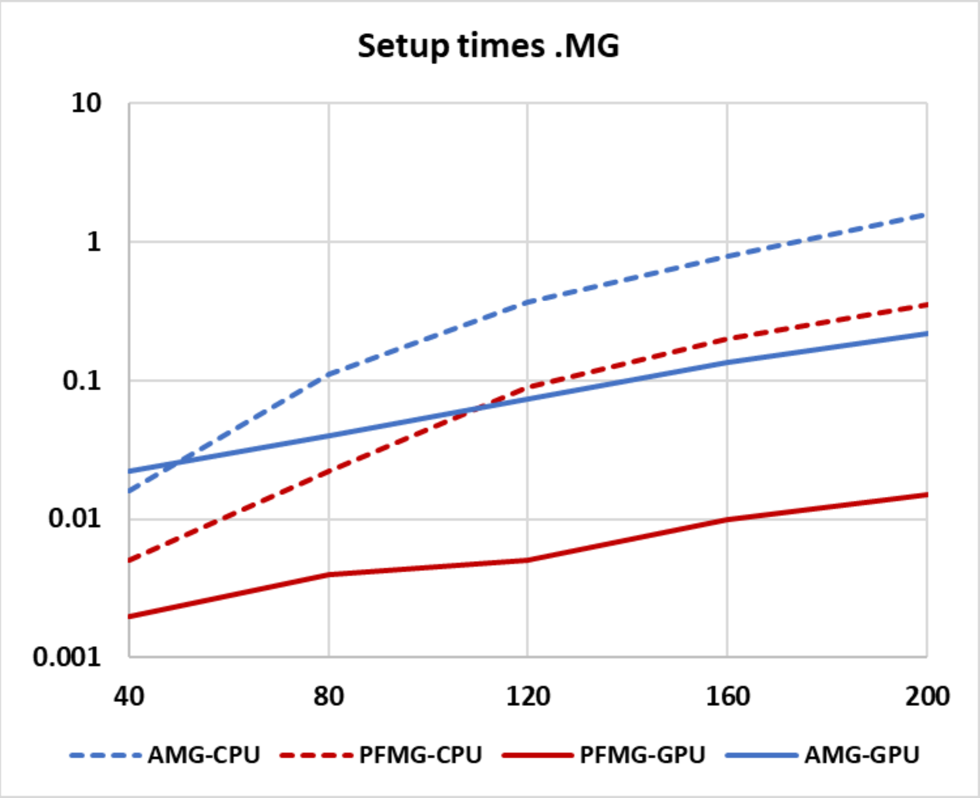
ThetaGPU
GPU: 1 Nvidia A100
CPU: 16 MPI tasks

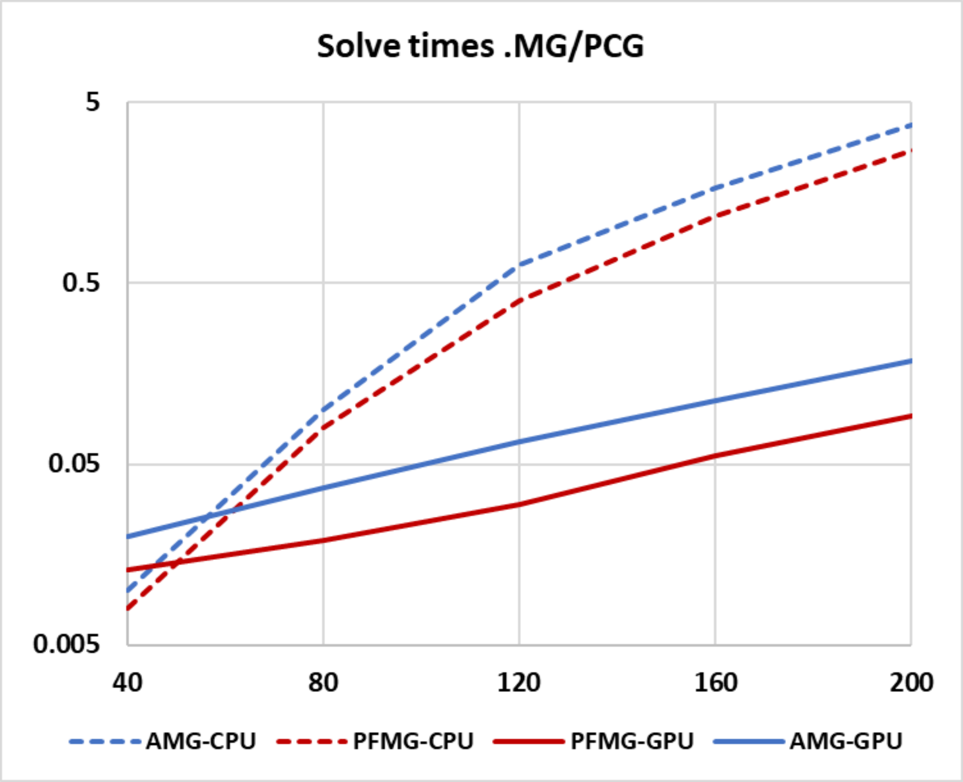
Used optimal settings for AMG, which are different for CPU and GPU!

Speedups at n=200

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

Most gains of PFMG over AMG in setup phase



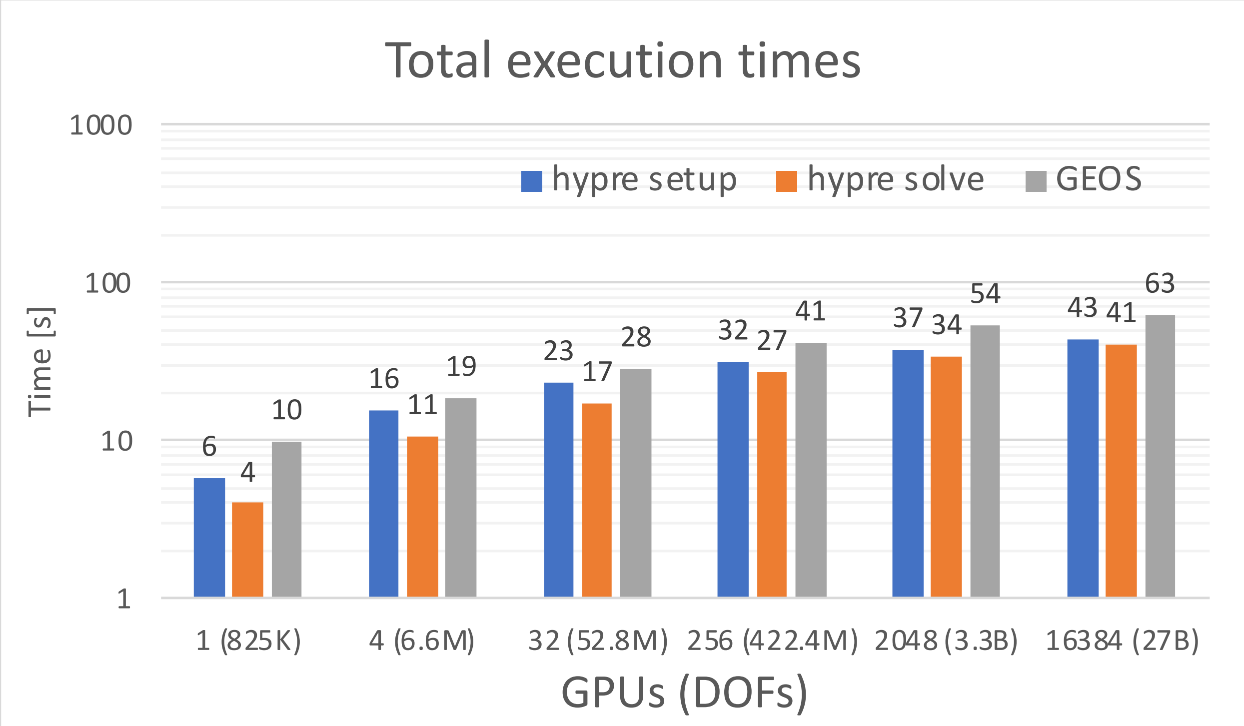


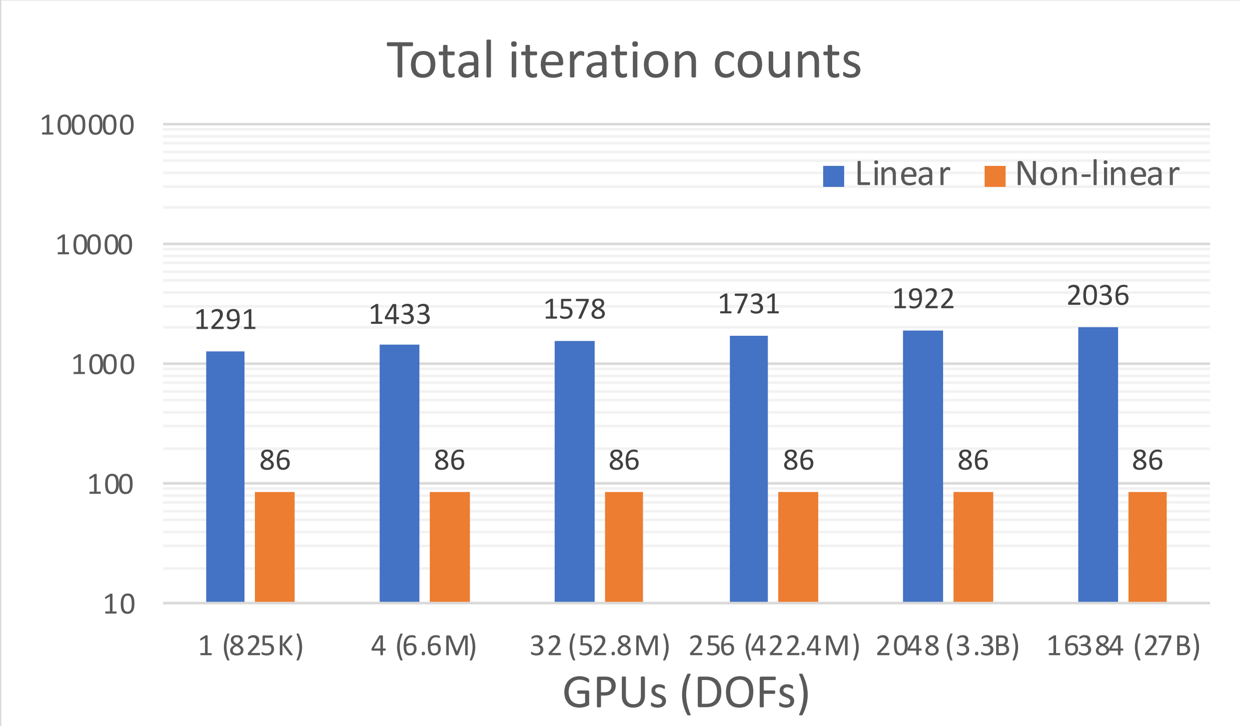
Speedup GPU/CPU	Speedup GPU/CPU	CPU Speedup PFMG/AMG	GPU Speedup PFMG/AMG
7.2	23.3	4.5	14.5

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 – (GEOS Simulator)

Single-phase flow (Poisson-like problem)



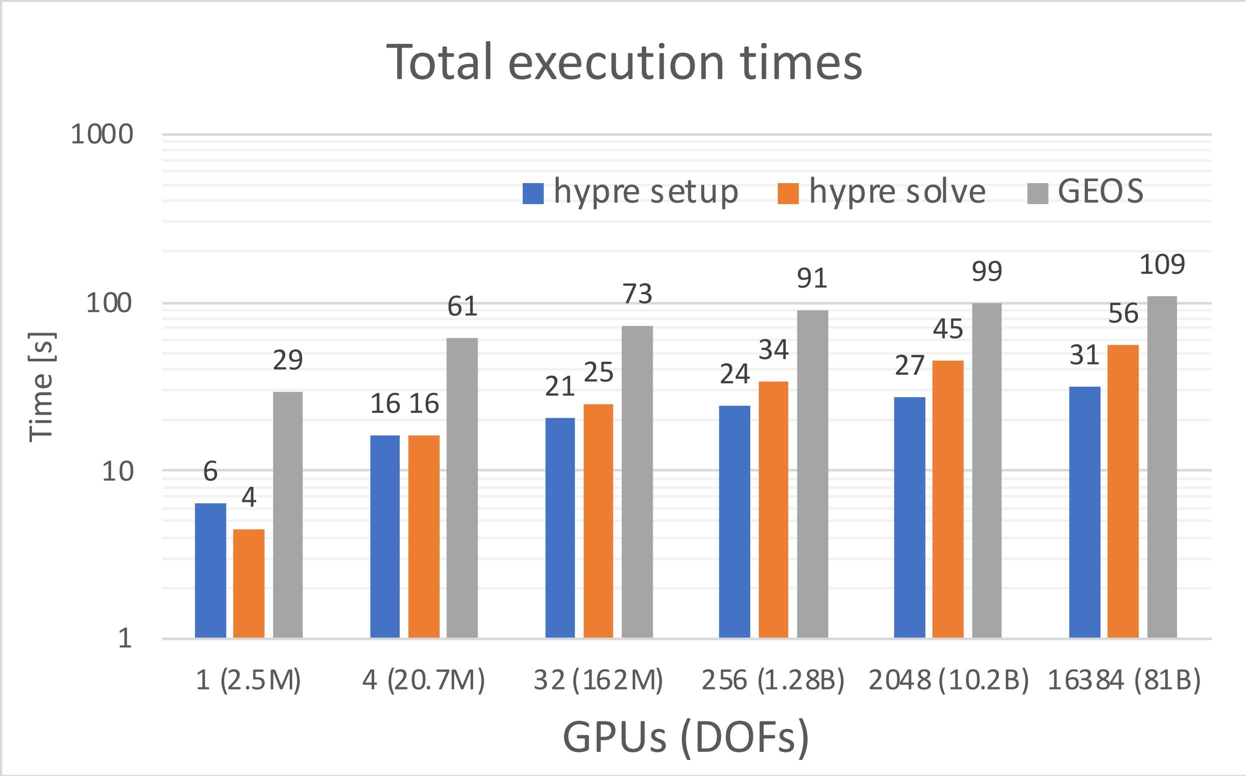


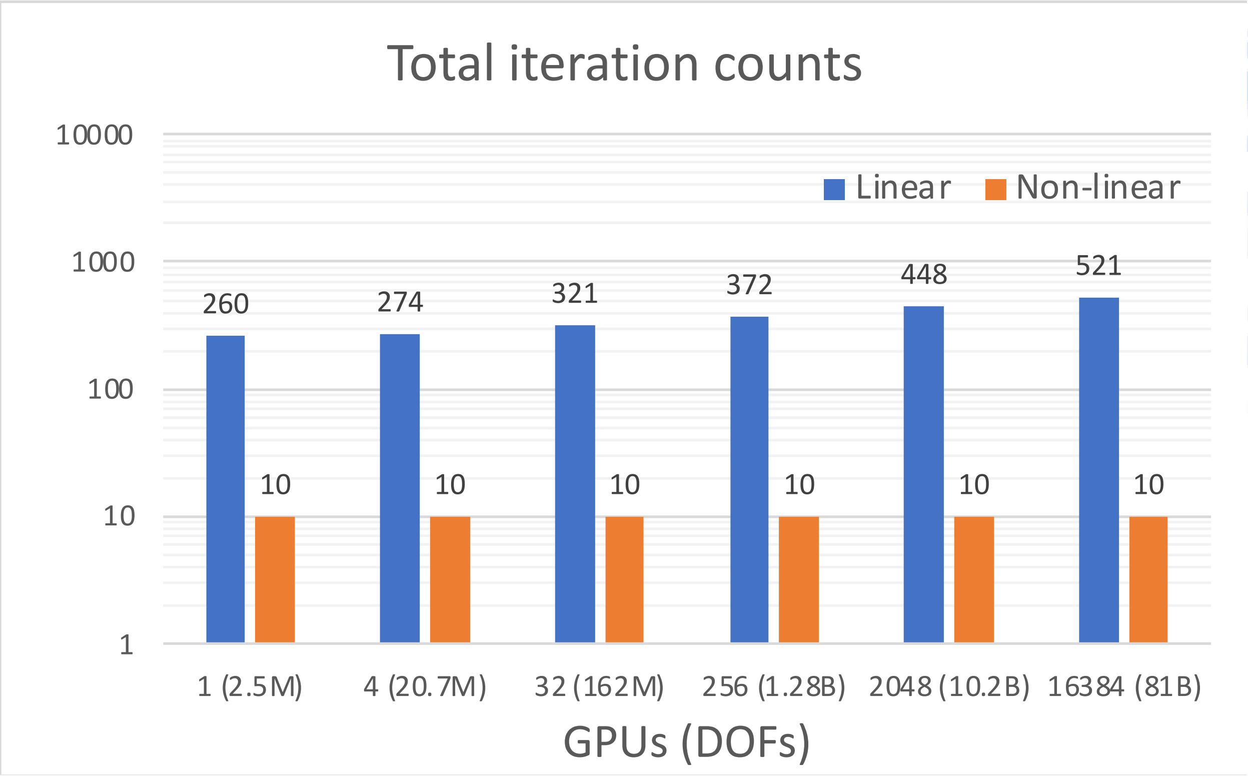
- Weak scaling with BoomerAMG/GMRES(50)
- Time complexity ~ O(log(N)); Iteration counts ~ O(1).

FASTMATH
EXASCALE COMPUTING PROJECT

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

Mechanics (linear elasticity problem)



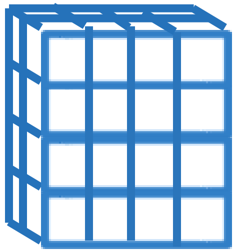


- Weak scaling with BoomerAMG/GMRES(40)

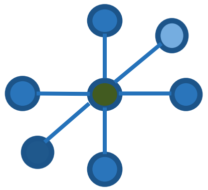
Hands-on Exercises: Comparing GPU to CPU Performance Algebraic Multigrid methods (Third Set of Runs)

- Go to https://xsdk-project.github.io/MathPackagesTraining2025/lessons/krylov_amg_hypre/
- Poisson equation: $-\Delta\varphi$ = RHS

  with Dirichlet boundary conditions $\varphi = 0$
- Grid: cube

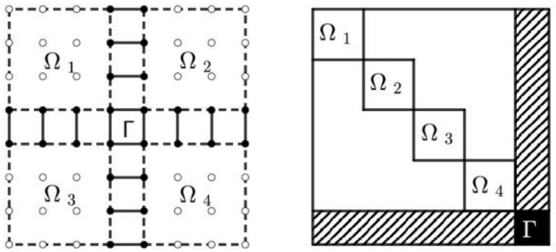


- Finite difference discretization:
  - Central differences for diffusion term
  - 7-point stencil



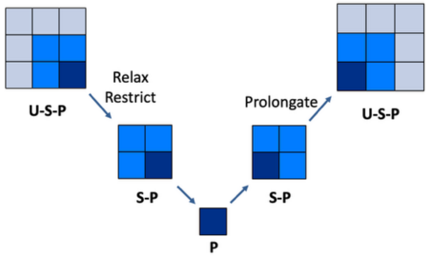
Some special general-purpose solvers and utilities in *hypre*

- Incomplete LU factorization
  - Based on a domain decomposition framework
    - Local ILU solve with global Schur complement solve
    - Various combinations of local ILU and global Schur solvers
  - GPU support available (for certain options)



- Multigrid reduction for PDE systems and Multiphysics applications
  - Reduction-based solver in a multigrid framework
  - Utilizes BoomerAMG as coarse solver
  - Effective Multiphysics preconditioner
  - GPU support available

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}$$



- Hypredrive: a lightweight interface hypre
  - YAML input files (no code recompilation)
  - Quickly test different solver strategies on different hardware

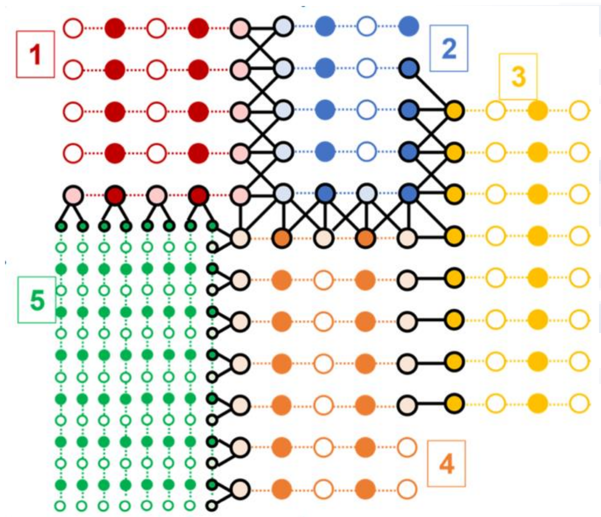
[1] Magri, V. A. P., (2024). https://doi.org/10.21105/joss.06654

GitHub

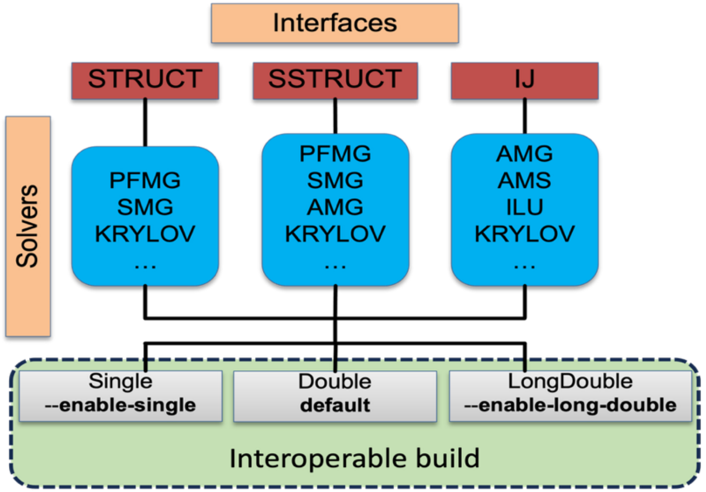
Some incoming features: *hypre-3.0*

- Semi-structured AMG (SSAMG)
  - Combines structured PFMG solver with BoomerAMG
    - Structured behavior within parts
    - Unstructured behavior at part boundaries
  - Support for rectangular matrix multiplication
    - Enables efficient construction of coarse grid operator

$$P^T A P = (P_s + P_u)^T (A_s + A_u)(P_s + P_u)$$



- Multiprecision support and Mixed-Precision solvers
  - Support to build hypre in multiple precisions in a single library
    - Enables interoperable use of hypre in different precisions
    - Currently supports single, double, longdouble
  - New mixed-precision solvers
    - Mixed-precision Krylov solvers
      - Double precision Krylov with single precision preconditioner
    - Defect correction/ iterative refinement-based solvers



Thank you!

CASC

Center for Applied Scientific Computing

Lawrence Livermore National Laboratory

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC.

This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), Scientific Discovery through Advanced Computing (SciDAC) program, and by the Exascale Computing Project, a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.