New Advances in *hypre* 3.0 for Mixed Precision and Semi-Structured Problems

FEM@LLNL Seminar

CASC
Center for Applied Scientific Computing

Robert D. Falgout

Rui Peng Li, Victor Magri, Wayne Mitchell, Daniel Osei-Kuffuor, Ulrike Yang

November 18, 2025

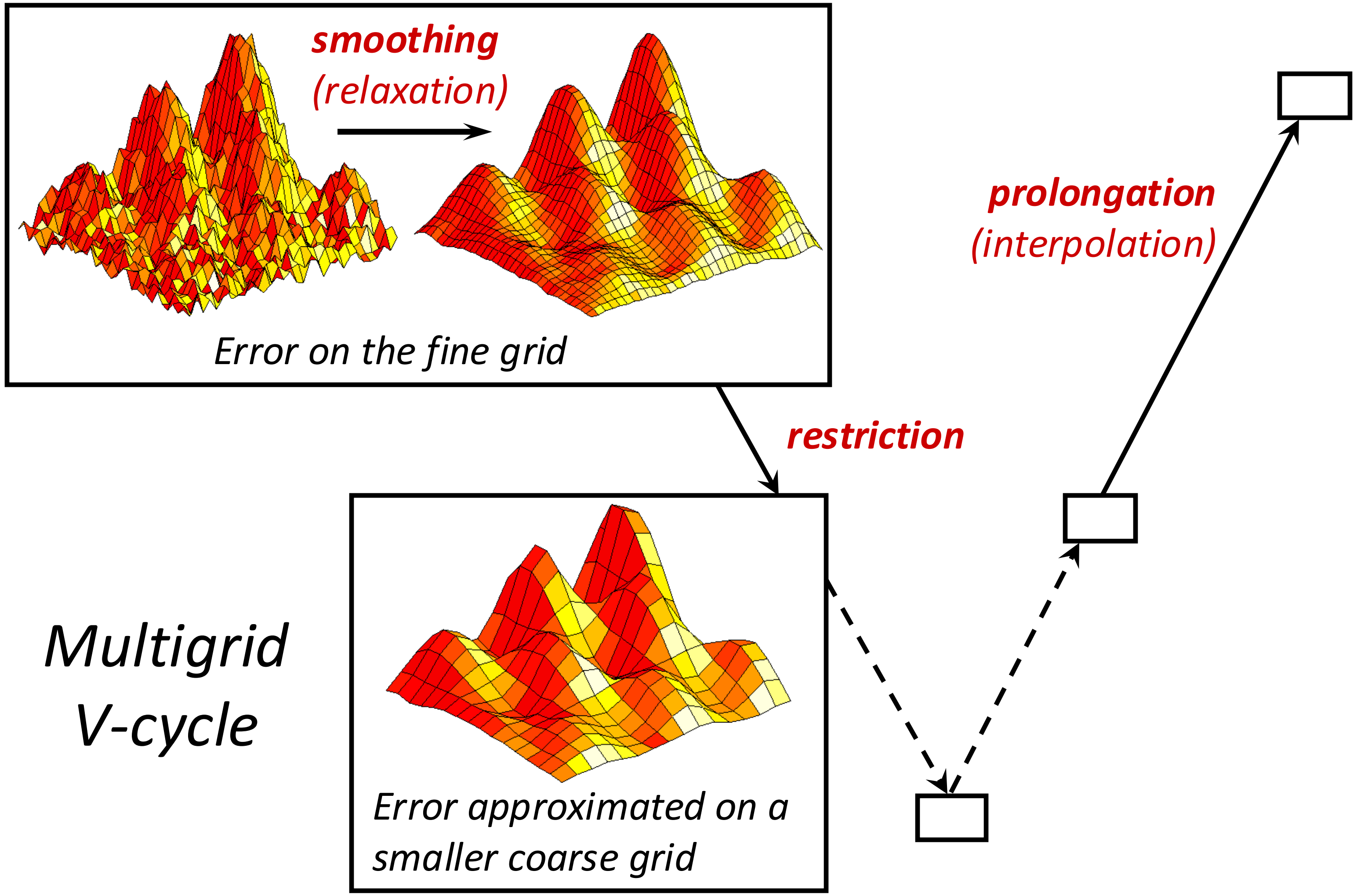
LLNL-PRES- 2013716

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

Lawrence Livermore National Laboratory

Multigrid (MG) is among the fastest and most scalable linear solvers
– uses a sequence of coarse grids to accelerate the fine grid solution

Multigrid V-cycle

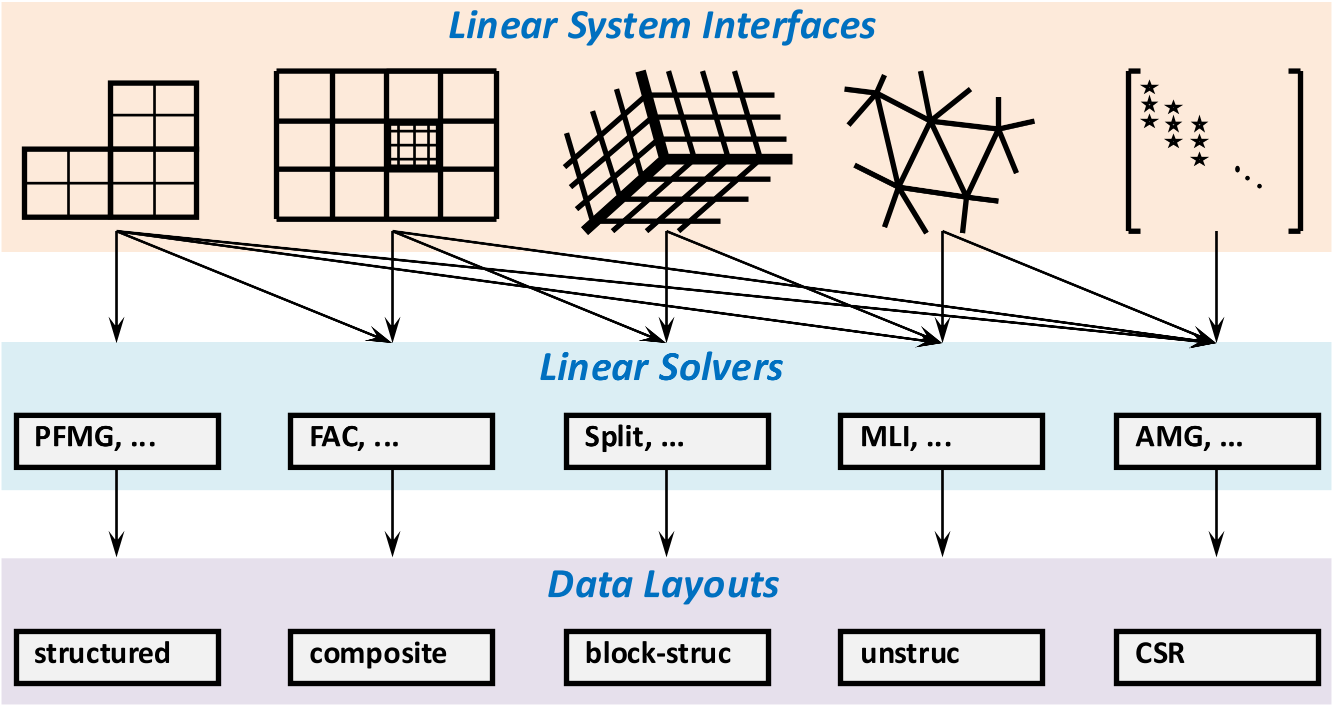


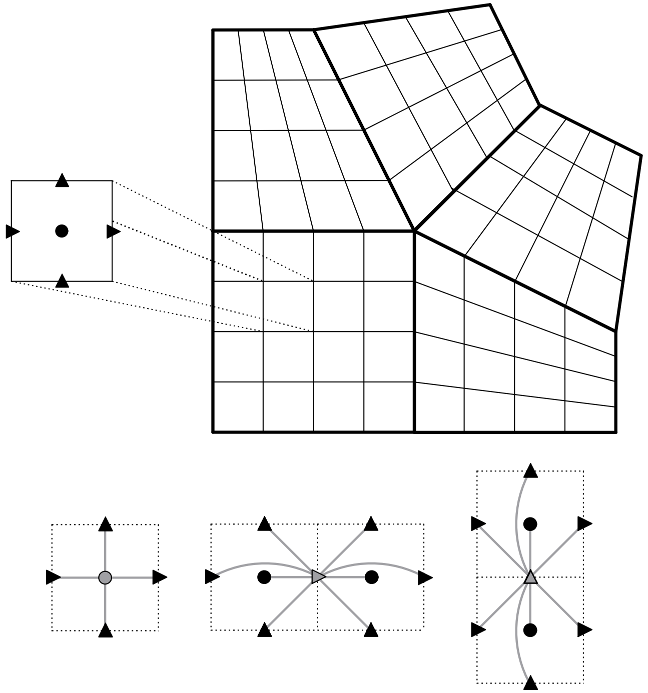
hypre

Widely used at LLNL, the DOE, and around the world

- Scalable, $O(N)$
- Many algorithms
- Geometric / algebraic
- Linear / nonlinear
- Math & CS research

Unique software interfaces in *hypre* enable more efficient solvers, kernels, and storage



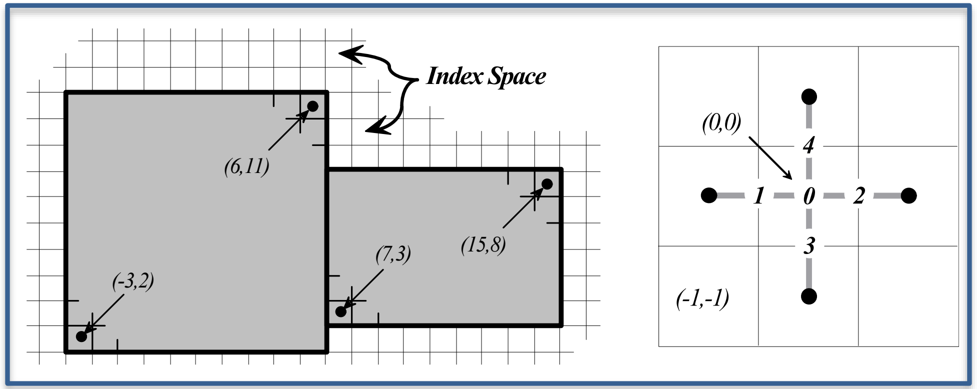


Block-structured grid with 3 variable types and 3 discretization stencils

- Example: *hypre*'s interface for semi-structured grids
  - Based on "grids" and either "stencils" or "finite elements"
  - Allows for specialized solvers for semi-structured problems
  - Also provides for more general solvers like AMG

There are three matrix object types (data structures) in *hypre*

- Struct matrix
  - Grid (list of boxes)
  - Stencil
  - Data array
  - Optimizations for constant coefficients and symmetric



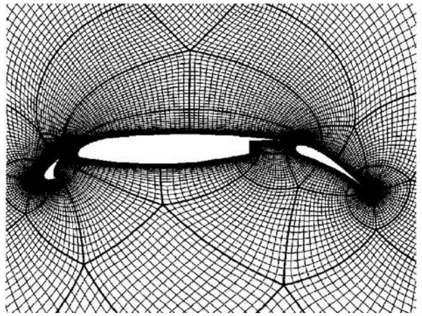
- ParCSR matrix
  - Two CSR matrices (diag, offd)
  - Mapping for off-proc connections

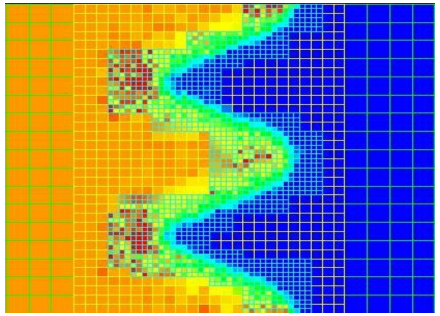
Row Pointer: $[i_0, i_1, i_2, \ldots, i_{Nrows}]$

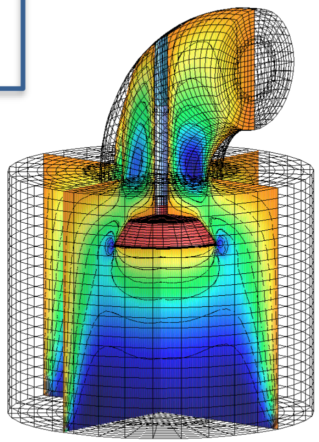
Column Indices: $[j_0, j_1, j_2, \ldots, j_{NNZ}]$

Data: $[a_0, a_1, a_2, \ldots, a_{NNZ}]$

- SStruct (semi-struct) matrix
  - Struct matrices for each part
  - ParCSR matrix for part connections

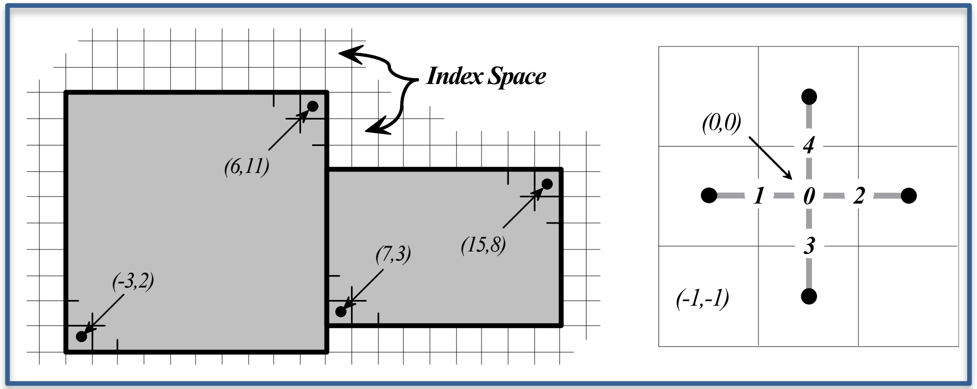




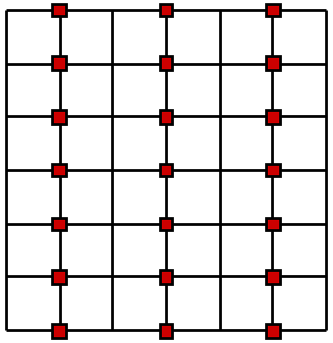


(S)Struct multigrid solver development in *hypre* has been hampered by limited matrix-vector tools

- (S)Struct matrices must be square
  - Struct matrix uses a stencil applied to a single grid
  - SStruct matrix depends on Struct: $A = A_s + A_u$
  - Matrix-vector multiply (Matvec) is only for square matrices
  - Matrix-matrix multiply (Matmat) is not available



- Struct solvers
  - SMG: semi-coarsening with plane/line smoothing
  - PFMG: semi-coarsening with point smoothing
  - Interpolation and coarse-grid operators are hard-coded for specific stencil cases



- SStruct (semi-structured) solvers
  - Split SMG/PFMG: approximate block Jacobi with SMG/PFMG on each part
  - True multigrid solvers are difficult to implement!

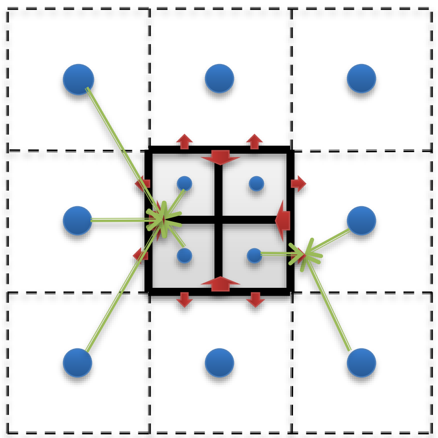
Overhaul of (S)Struct code in hypre-3.0 enhances matrix-vector tools and provides new SStruct solver SSAMG

- Matmat and Matvec are key kernels, but they often involve rectangular matrices
  - E.g., Interpolation/restriction, RAP in multigrid, RAP in AMR interface

- Matrix-vector class has been extended beyond square matrices (also added Matmat)
  - Implementation supports both variable and constant-coefficients (and combinations)
  - Struct matrix still based on stencils, but with domain/range grids
  - Simplifies implementation of multigrid algorithms and AMR interface
  - Centralizes code optimization to a few core routines (but optimization is trickier)
  - Allows for faster solvers – implemented semi-structured AMG (SSAMG) algorithm

- (S)Struct re-design began in about 2014
  - hypre-3.0 release was in September 2025
  - It's been a long process

- Longer term work
  - Additional solver updates (SMG, PFMG extensions, etc.)
  - Finish implementing an AMR interface



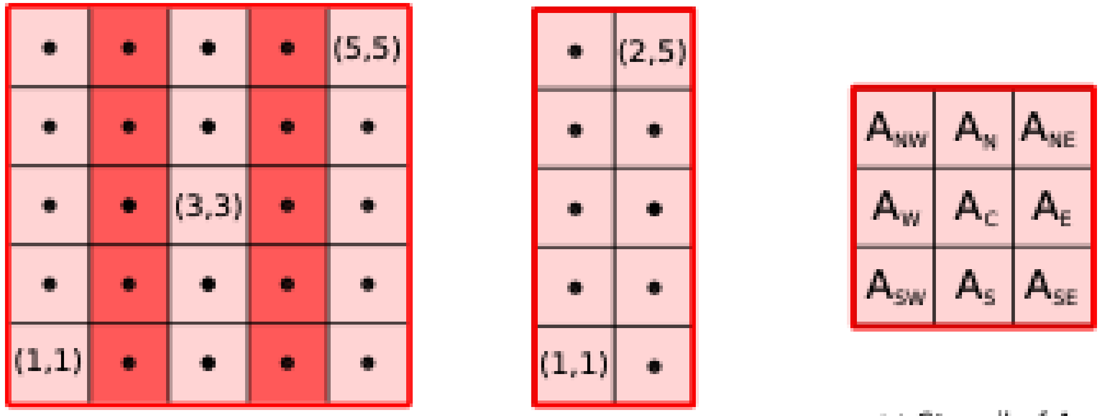
- hypre-3.0 was the first major release since hypre-2.0 in December of 2006!

Struct rectangular matrices are also based on stencils and grids but extend the current implementation

- Matrices have a domain and range grid where each coarsens a common base grid
- Matrix stencils act on an index space that is the same or finer than either grid
  - The stencil grid need not be the same as the base grid (currently they are the same)
  - Either the matrix or its transpose must be representable by a single “range stencil”
- Example: Interpolation in PFMG (semi-coarsened coarse grid in x)

$$
\begin{aligned}
P &\sim [P_W \quad 1 \quad P_E]_c \\
&= [P_W \quad * \quad P_E]_c^{r_\mathrm{f}} \oplus [* \quad 1 \quad *]_c^{r_\mathrm{c}}
\end{aligned}
$$

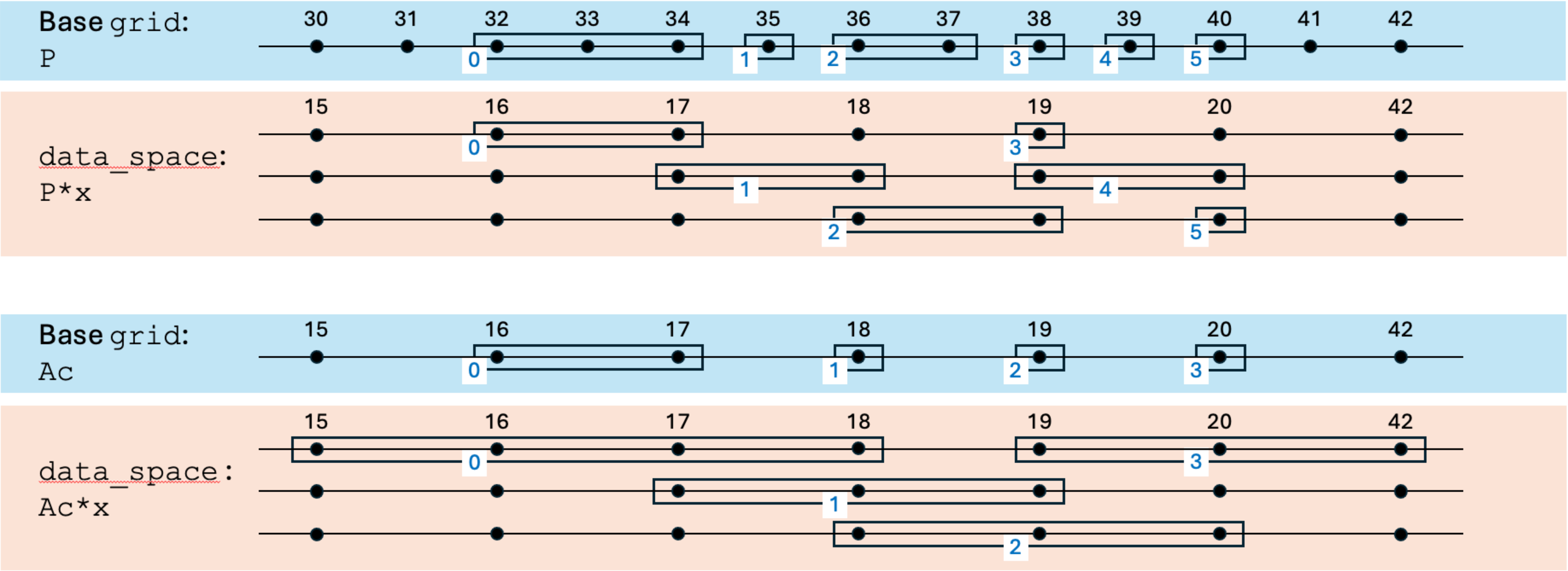
$$
\begin{aligned}
(P\boldsymbol{u})_{i,j} &= P_W u_{i-1,j} + P_E u_{i+1,j}\,, & (i,j) \in r_\mathrm{f} \\
(P\boldsymbol{u})_{i,j} &= u_{i,j}\,, & (i,j) \in r_\mathrm{c}
\end{aligned}
$$



(a) Fine grid (b) Coarse grid (c) Stencil of A centered at (3,3)

Struct Matvec uses ghost zones for communication – these are now determined automatically

- Ghost zone size for a vector $\boldsymbol{x}$ depends on how it's used – hard for users to specify
  - Example: Matvec with two different matrices P and A_c in multigrid



Struct Matmat also produces a stencil-based matrix with constant/variable coefficients

- Stencil algebra determines the product stencil and equations for computing its entries
  - Use the fact that $stencil(AB) = B^T stencil(A)$

$$ABC = \begin{bmatrix} X_{i,j}^{-1,1} & X_{i,j}^{0,1} & X_{i,j}^{1,1} \\ X_{i,j}^{-1,0} & X_{i,j}^{0,0} & X_{i,j}^{1,0} \\ X_{i,j}^{-1,-1} & X_{i,j}^{0,-1} & X_{i,j}^{1,-1} \end{bmatrix}, \qquad X_*^* = \sum a_*^* \, b_*^* \, c_*^*$$

- Assume 2 “data spaces”, compute necessary ghost zones, aggregate communication, use mask if needed for constant coefficient stencil entries, unroll kernel loops
- Kernel – write $abc = \alpha \, v_1 v_2 v_3$ where
  - the constant α is 1x the product of the constant coefficient entries in $\{a, b, c\}$ (if any)
  - the v_i associated with each constant coefficient entry is set to a $mask = \{0, 1\}$
  - each v_i is associated with one of possibly two ”data spaces” → 3 cases to implement in 3D
- Symmetric matrices – store and compute only “half” of the entries

Struct Matmat currently has two kernel implementations

- Loop unrolling works well on CPUs – matches current hardcoded $P^T A\ P$ code

$$X_i \mathrel{+}= \sum \alpha v_1 v_2 v_3$$

- Loop fusion is better for GPUs – fewer kernel launches

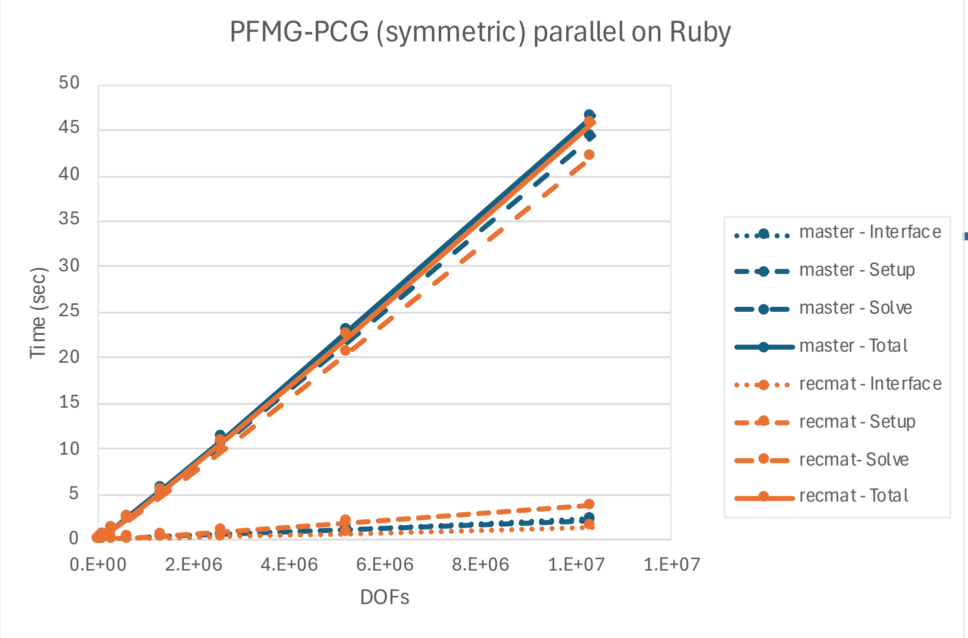
$$X_1 \mathrel{+}= \alpha v_1 v_2 v_3$$
$$X_2 \mathrel{+}= \alpha v_1 v_2 v_3$$
$$\ldots$$

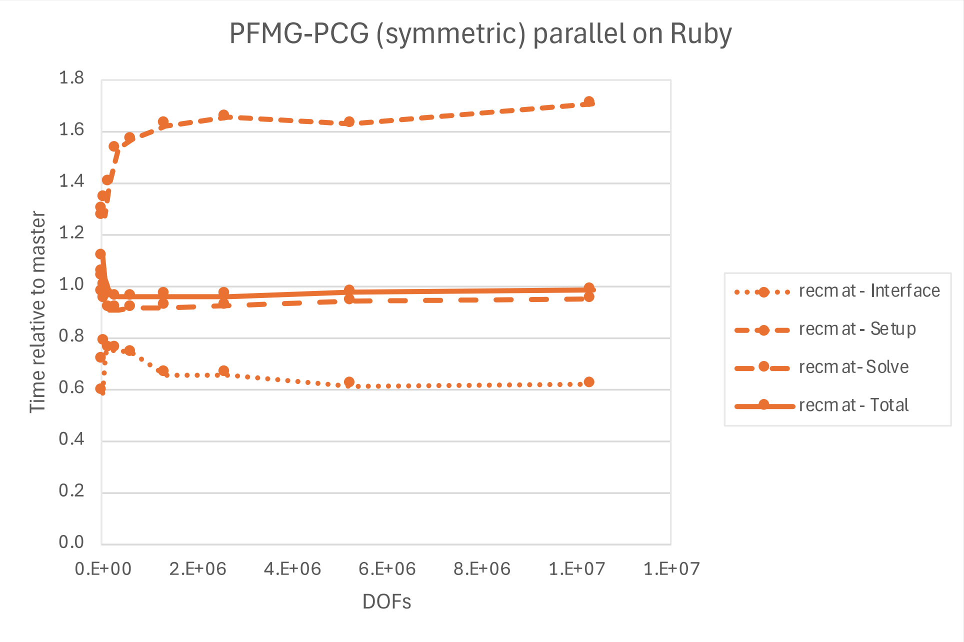
Performance optimizations in Struct

- Optimizations to bring performance in line with main branch
  - Matvec and Matmat loop unrolling
  - Smarter handling of ghost zone resizing
  - Support for symmetric matrices in mat-mat

- PFMG optimizations Jan-Apr 2025
  - Improved setup by 3.8x
  - Improved solve by 1.8x
  - Total times are now slightly faster than master

- Ruby at LLNL
  - Intel Supermicro Zeon
  - 85,568 cores = 1512 nodes * 56 cores/node
  - #310 on June Top 500, 3.7 PFlops/s LINPACK



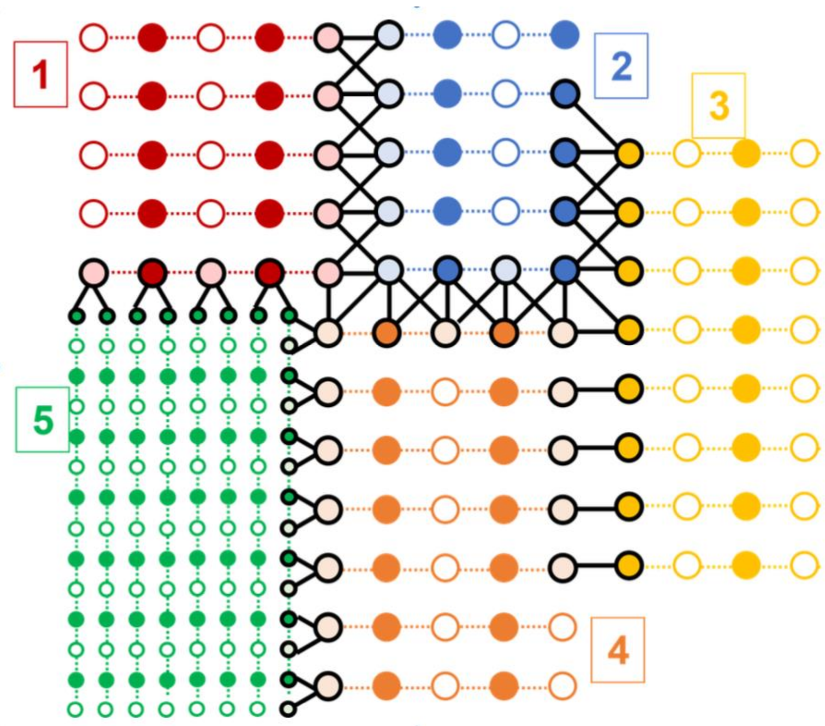


Semi-structured algebraic multigrid (SSAMG) combines the structured PFMG solver with BoomerAMG

- Structured behavior within parts (PFMG)
  - Semi-coarsening
  - Operator-based, two-point structured interpolation
- Unstructured behavior at part boundaries
  - One-sided structured interpolation: $P = P_s$
  - Full interpolation (WIP) can greatly improve convergence (e.g., 7 iterations instead of 20): $P = P_s + P_u$
- Galerkin coarse-grid operator

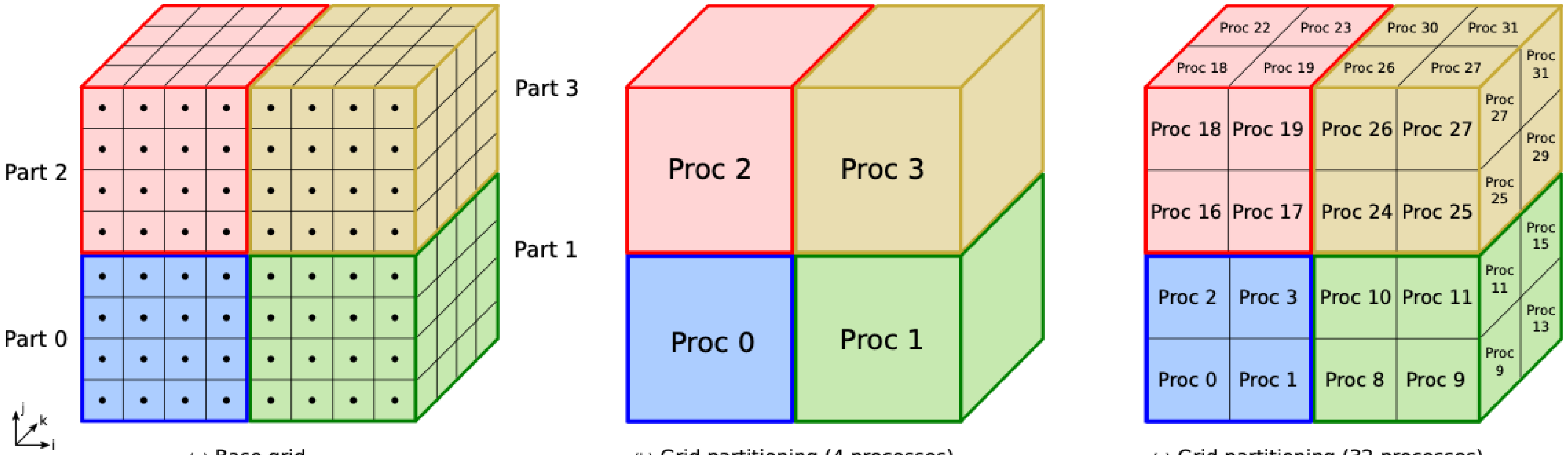
$$P^T A P = (P_s + P_u)^T (A_s + A_u)(P_s + P_u)$$

  - Rectangular structured mat-mats
  - Need to multiply Struct and ParCSR matrices
- BoomerAMG as coarsest-grid solver
  - Workhorse multigrid solver (uses ParCSR)
  - Variation of classical AMG



Numerical results show qualitatively similar results to earlier work but with improved performance

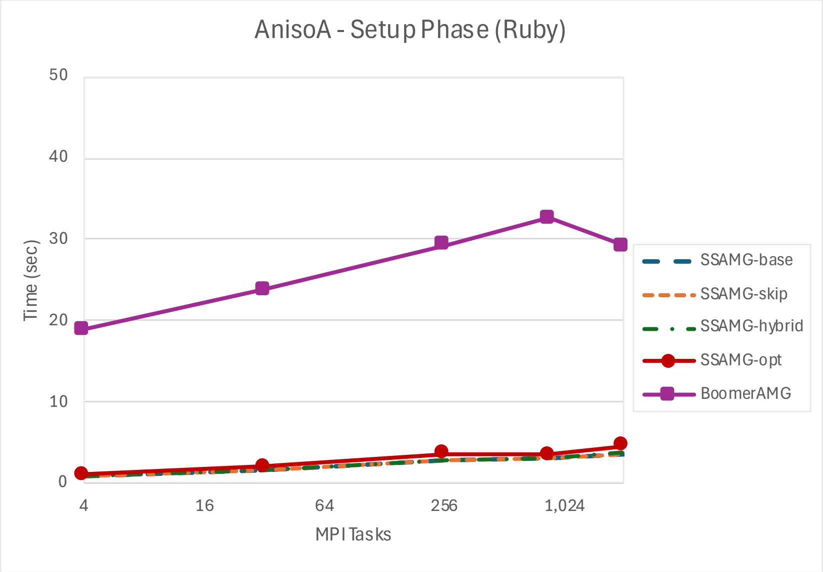
- 3D cubic grid with four parts, 128 x 128 x 128 grid per MPI task
- Three scenarios (A, B, C) of anisotropies in each part
- Discretization stencil: $A \sim [-\gamma] \begin{bmatrix} & -\beta & \\ -\alpha & 2(\alpha+\beta+\gamma) & -\alpha \\ & -\beta & \end{bmatrix} [-\gamma]$

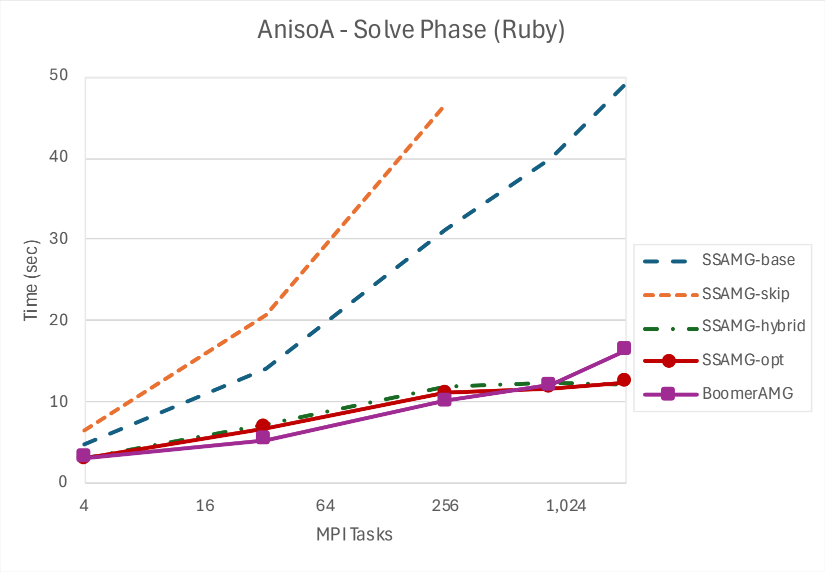


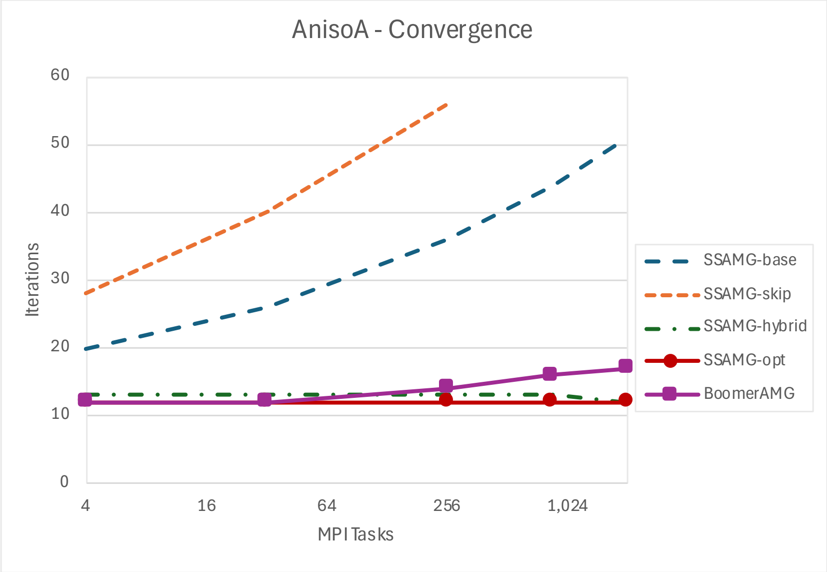
(a) Base grid (b) Grid partitioning (4 processes) (c) Grid partitioning (32 processes)

1. Magri, Falgout, Yang, "A new semi-structured algebraic multigrid method", *SIAM J. Sci. Comput.* (2023)

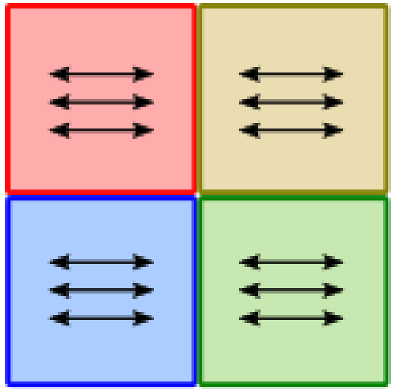
Results (A) – SSAMG Setup is more than 7x faster than AMG





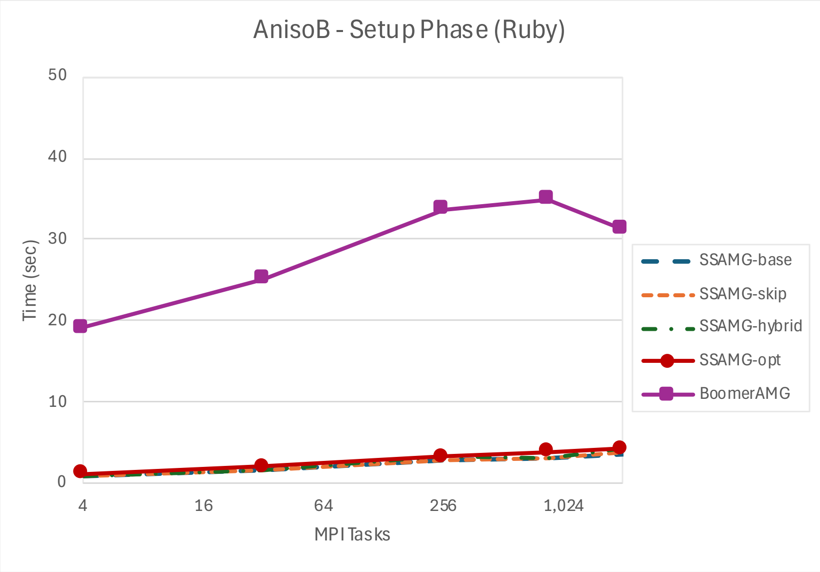


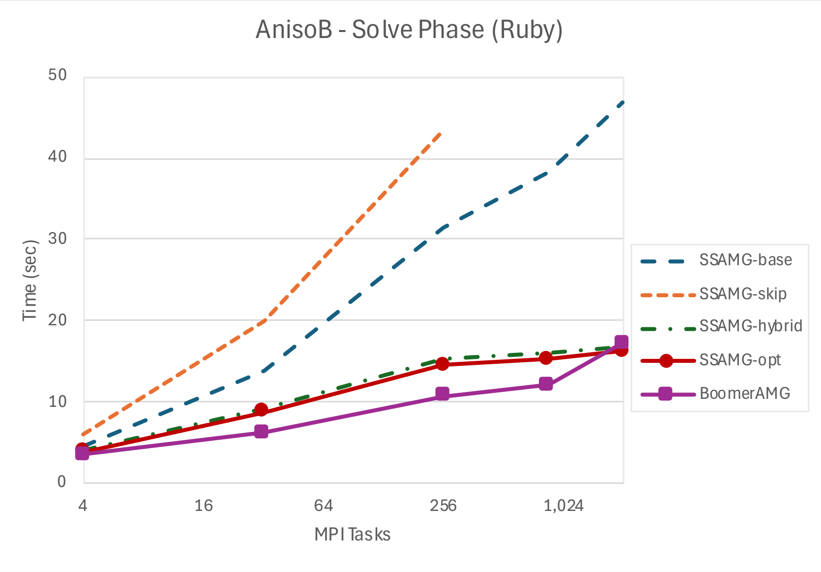
- Solvers:
  - SSAMG-base – L1 Jacobi smoother, $\omega = 3/2$
  - SSAMG-skip – SSAMG-base with skip-relaxation option
  - SSAMG-hybrid – switch to BoomerAMG on level 10
  - SSAMG-opt – switch to BoomerAMG on level 7
  - BoomerAMG – one level aggressive coarsening, L1 GS, $\theta = 0.25$, ext+i interpolation

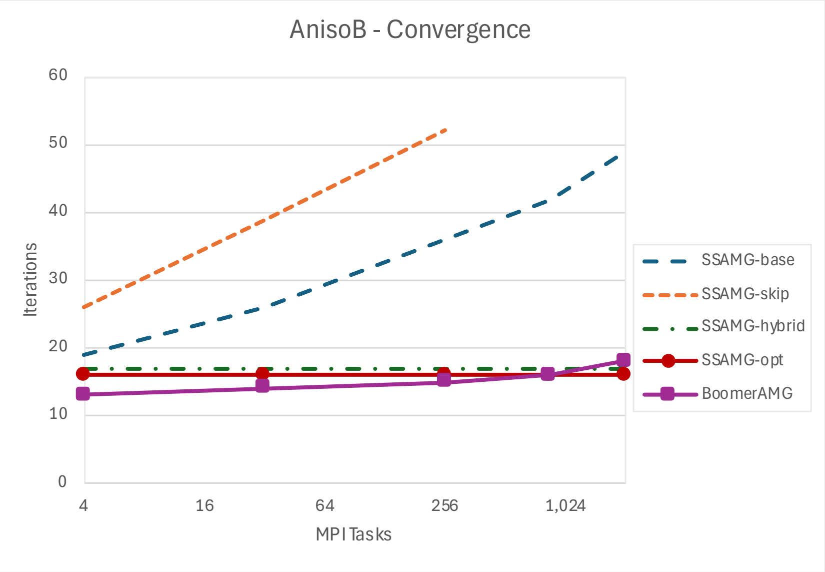


1. Magri, Falgout, Yang, "A new semi-structured algebraic multigrid method", *SIAM J. Sci. Comput.* (2023)

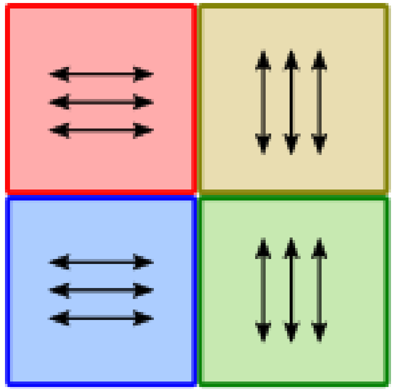
Results (B) – SSAMG Setup is more than 7x faster than AMG





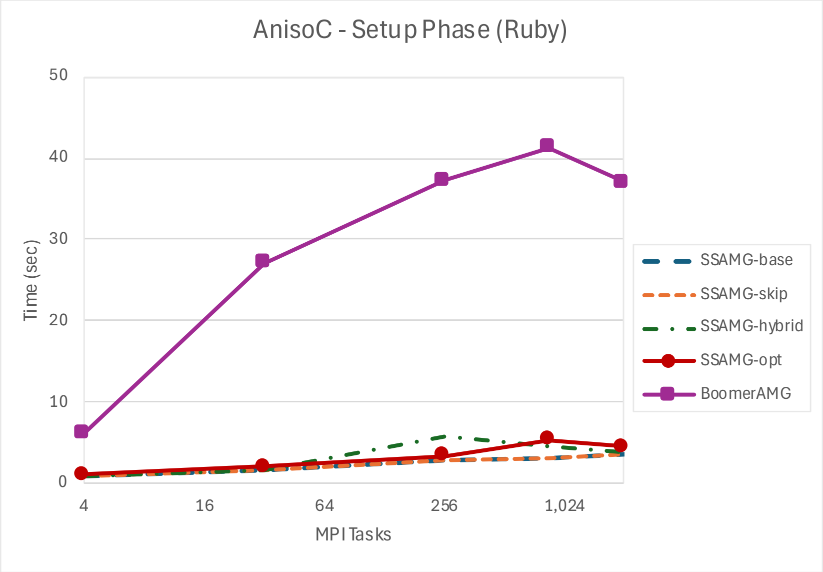


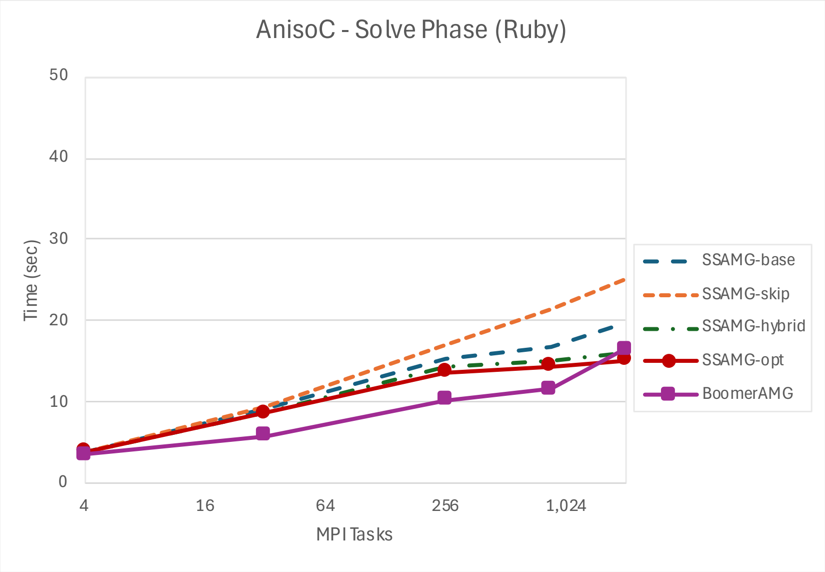
- Solvers:
  - SSAMG-base – L1 Jacobi smoother, $\omega = 3/2$
  - SSAMG-skip – SSAMG-base with skip-relaxation option
  - SSAMG-hybrid – switch to BoomerAMG on level 10
  - SSAMG-opt – switch to BoomerAMG on level 7
  - BoomerAMG – one level aggressive coarsening, L1 GS, $\theta = 0.25$, ext+i interpolation

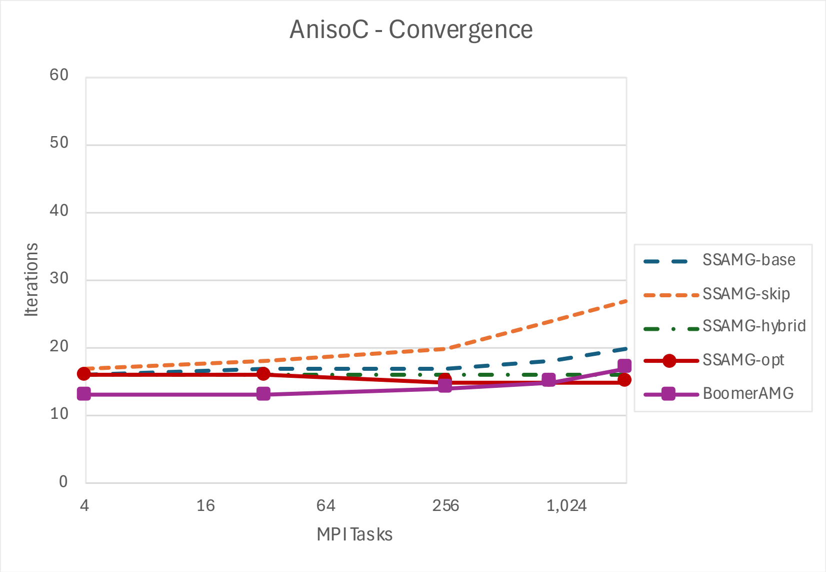


1. Magri, Falgout, Yang, "A new semi-structured algebraic multigrid method", *SIAM J. Sci. Comput.* (2023)

Results (C) – SSAMG Setup is more than 6x faster than AMG

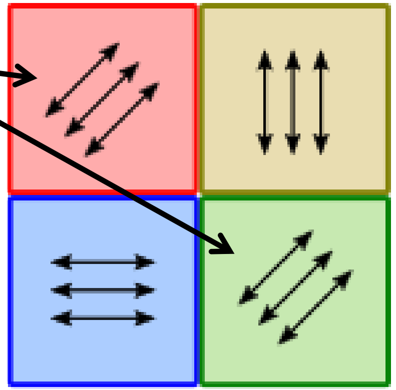






- Solvers:
  - SSAMG-base – L1 Jacobi smoother, $\omega = 3/2$
  - SSAMG-skip – SSAMG-base with skip-relaxation option
  - SSAMG-hybrid – switch to BoomerAMG on level 10
  - SSAMG-opt – switch to BoomerAMG on level 7
  - BoomerAMG – one level aggressive coarsening, L1 GS, $\theta = 0.25$, ext+i interpolation

Anisotropic in z



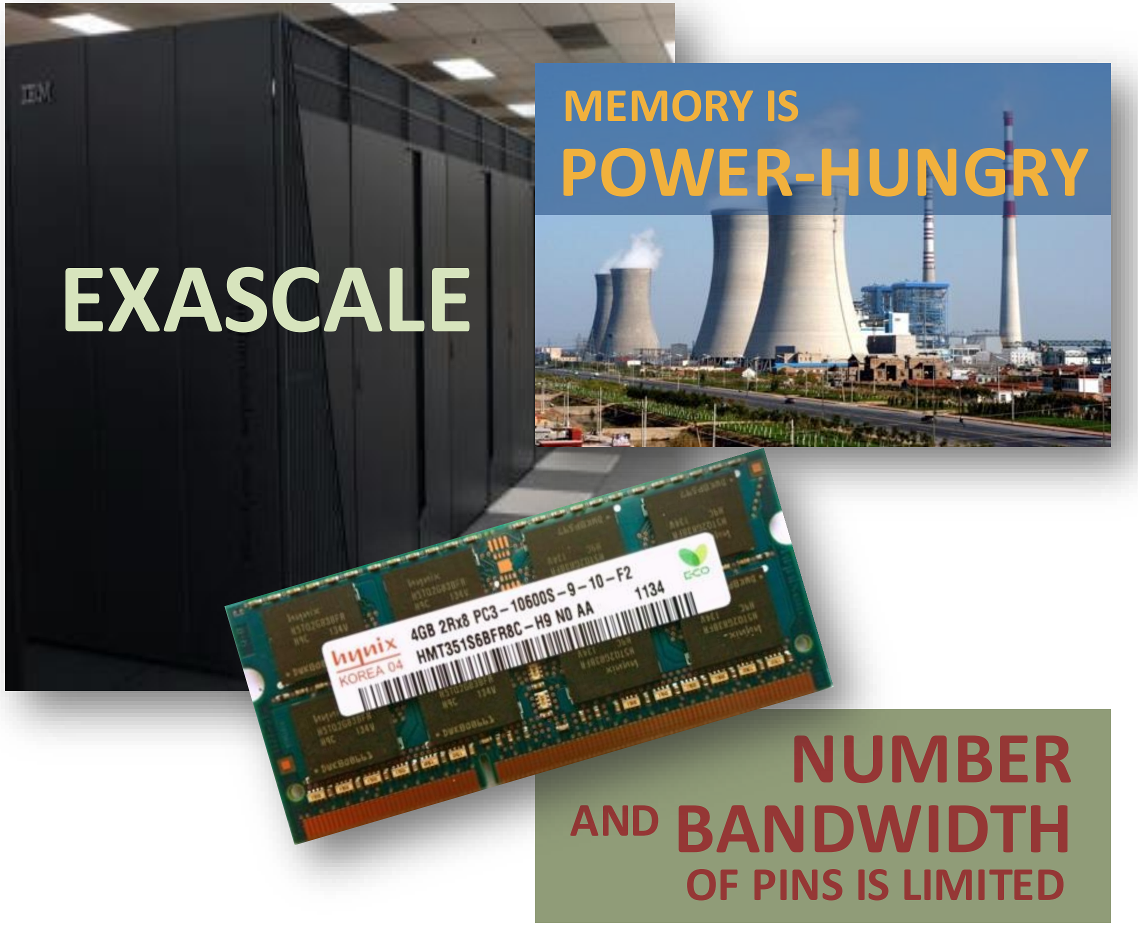
1. Magri, Falgout, Yang, "A new semi-structured algebraic multigrid method", *SIAM J. Sci. Comput.* (2023)

Mixed Precision

Mixed precision solver development is driven by several factors

- HPC support for low precision arithmetic
  - Single (or lower) precision can be more than 2x faster
  - Tensor core hardware on NVIDIA GPUs

- Data motion and memory capacity are becoming limiting factors in HPC
  - Better bandwidth utilization



Mixed precision in hypre-3.0 has several goals and features

- Terminology
  - multiprecision – a solver, function, or feature that uses one precision at a time
  - mixed precision – a solver, function, or feature that uses different precisions simultaneously
- Mixed precision is primarily intended to
  - provide multiple precisions at runtime
  - provide faster and more memory efficient mixed-precision solvers
- Features include
  - User code runs as before without requiring modification
  - All compile-time precisions are available at runtime
  - Precision can be set globally at runtime and requires minimal changes to user code
  - [Future] Precision of objects (e.g., matrices) determines the precision of related methods (e.g., `Matvec`)
  - Mixed precision solvers are available

HYPRE provides runtime multiprecision and mixed precision capabilities with minimal impact on users

- Multiprecision has been available for many years

```
--enable-single        (autotools)
-DHYPRE_SINGLE=ON      (CMake)
```

- With hypre-3.0: multiprecision and mixed precision are available at runtime

```
--enable-mixed-precision.          (autotools)
-DHYPRE_ENABLE_MIXED_PRECISION=ON  (CMake)
```

- Users can turn on mixed precision without changing code
  - Code will compile and run as before
- Several levels of support for using runtime precision
  - Code changes required

Runtime precision is available in several ways

- Every function `Foo` in hypre (e.g., `HYPRE_PCGSolve`) is available in fixed precision

`Foo_flt`	(precision in C = `float`)
`Foo_dbl`	(precision in C = `double`)
`Foo_long_dbl`	(precision in C = `long double`)

- Prototypes are the same as before except for real types
  – `HYPRE_Real` maps to specific C types (e.g., `float` for function `Foo_flt`)

- Every user-API function `Foo` has precision determined (globally) by
  – `HYPRE_Int HYPRE_SetGlobalPrecision(HYPRE_Precision precision)`
  – where `precision` has value `HYPRE_REAL_DOUBLE` for example (default value is set at configuration time by the user)

- Prototypes for `Foo` are different because they need to support all precisions
  – Scalar `HYPRE_Real` becomes `long double`, while array `HYPRE_Real*` becomes `void*`
  – No casting is necessary, but care is needed to manage arrays

- Function `Foo_pre(precision, …)` is useful for writing mixed precision code

Test driver `sstruct` illustrates use of runtime multiprecision (example drivers are to come)

- Additional `-precision` argument switches the global runtime precision

```
> sstruct -in TEST_sstruct/sstruct.in.cube -precision 0
...
Iterations = 4
Final Relative Residual Norm = 2.187087e-07
```

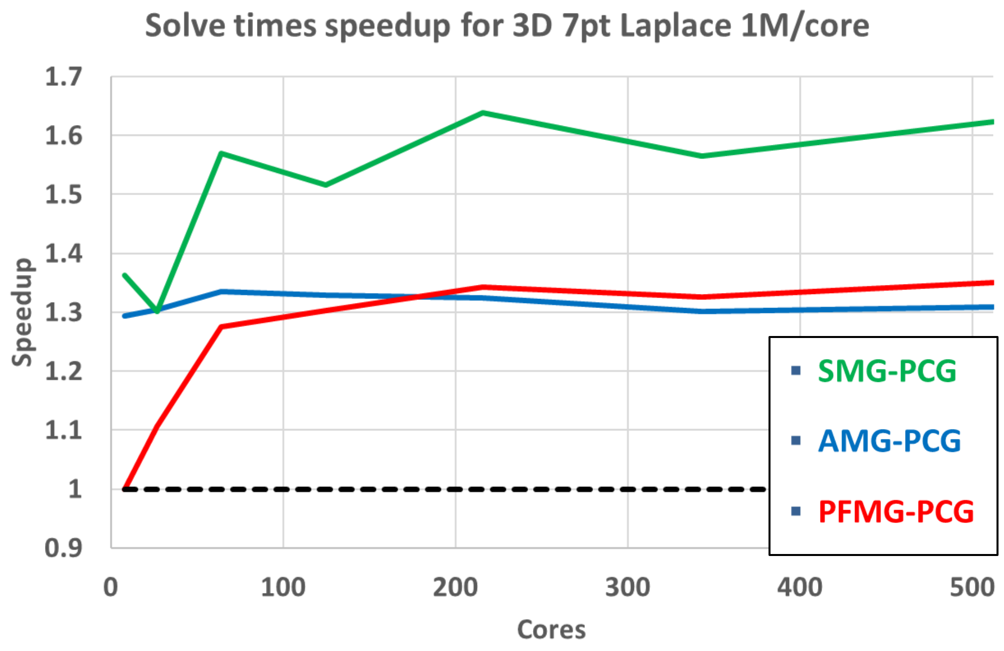
```
> sstruct -in TEST_sstruct/sstruct.in.cube -precision 1
...
Iterations = 4
Final Relative Residual Norm = 5.459331e-16
```

- Code changes needed
  - Add call to `HYPRE_SetGlobalPrecision()`
  - Manage real arrays with various utility routines (to keep things clean) such as
    `hypre_MuPDataCopyToMP(h_values, values, values_size);`

Mixed precision solver availability is limited to lower-precision preconditioned Krylov methods – more capability coming soon

- Using lower-precision preconditioners requires minimal code changes:
  - Add `SetPrecondMatrix()` call
  - Utilities available for creating lower-precision matrix
- Results for single-precision preconditioners
  - Need flexible CG for convergence at lower tolerances
  - Solve phase speedups for AMG and PFMG are comparable, with larger speedups for SMG
  - Structured solvers have higher per-iteration speedups
- Within a month or two:
  - GPU support for mixed precision (draft PR in progress)
  - Mixed precision BoomerAMG (results on next slide)

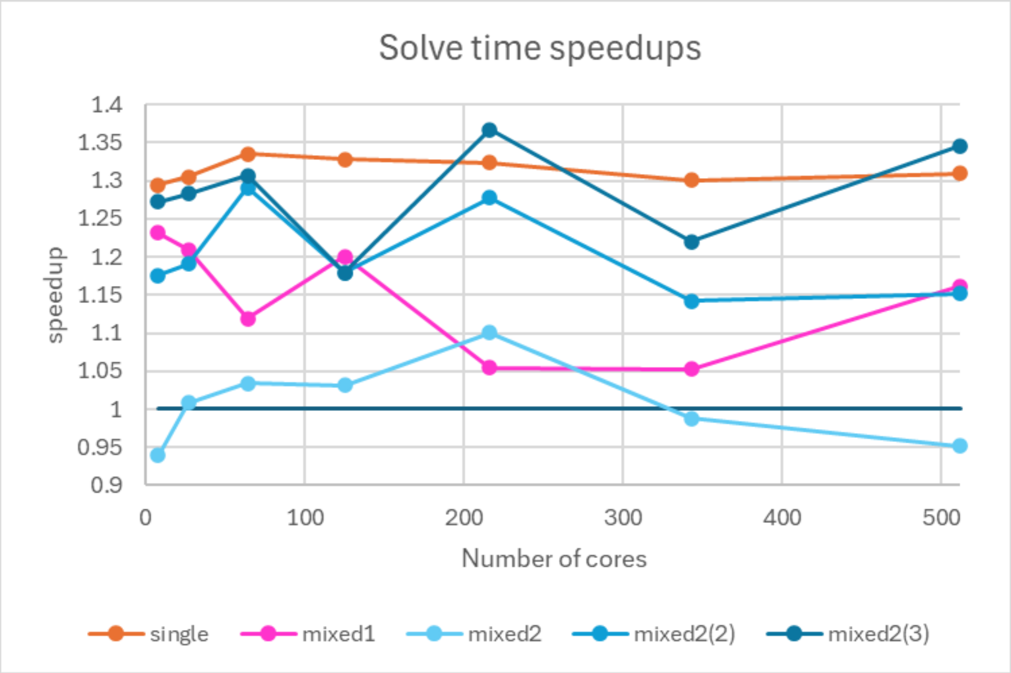
Speedups for single-precision preconditioners (double-precision PCG solve) on LLNL RZHound (Intel Sapphire Rapids) with 100x100x100 = 1M/core for 7pt Laplace problem. Results for AMG, PFMG, and SMG.



Mixed precision AMG uses different precisions on each grid level

- Setup phase
  - Additional creation of a lower precision matrix
- Solve phase
  - Additional precision conversions of the residual and error during restriction and prolongation
- Different API
  - `HYPRE_BoomerAMGSolve_mp()` (for example)
- Major challenge – size of AMG data structure
- Results:
  - Defaults: HMIS coarsening, ext+i(4), L1-GS
  - Setup: no real benefit
  - Solve: using single precision on lower levels is best

Results for mixed-precision AMG-PCG with different precision variations on each grid level. 7pt 3D Laplace problem with 100x100x100 = 1M/core on RZHound (Intel Sapphire Rapids).



Automation is used as much as possible to implement and maintain the mixed precision support in hypre-3.0

- Developers use scripts (awk, sed) to auto-generate multiprecision wrapper code
  - Keep lists of library symbols in special files (e.g., `mup.functions`)
  - Run `mup_check` to check library symbols – update lists as needed
  - Run `mup_code` to generate wrapper code
  - Commit everything to the repo

- HYPRE build is more involved:
  - Compiles 3 times to generate fixed-precision symbols
  - Compiles extra auto-generated code

Build Type / Function List	mup.fixed	mup.functions	mup.methods
MP_BUILD_SINGLE	Foo_flt	Foo_flt	Foo_flt
MP_BUILD_DOUBLE	Foo_dbl	Foo_dbl	Foo_dbl
MP_BUILD_LONGDOUBLE	Foo_long_dbl	Foo_long_dbl	Foo_long_dbl
No MP (standard compile)	Foo	Foo, Foo_pre	Foo, Foo_pre

```
HYPRE_Int
HYPRE_BiCGSTABSetTol_pre( HYPRE_Precision precision, HYPRE_Solver solver, hypre_long_double tol )
{
   switch (precision)
   {
      case HYPRE_REAL_SINGLE:
         return HYPRE_BiCGSTABSetTol_flt( solver, (hypre_float)tol );
      case HYPRE_REAL_DOUBLE:
         return HYPRE_BiCGSTABSetTol_dbl( solver, (hypre_double)tol );
      case HYPRE_REAL_LONGDOUBLE:
         return HYPRE_BiCGSTABSetTol_long_dbl( solver, (hypre_long_double)tol );
      default:
```

Summary / Conclusions

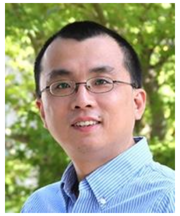
- (S)Struct rectangular matrix-vector tools
  - Implemented and optimized for CPUs and GPUs
  - SSAMG can outperform BoomerAMG on semi-structured problems – more optimizations to come
  - New tools will simplify solver development and enable solver improvements

- Mixed precision algorithms
  - Multi-precision code is available at runtime
  - Mixed-precision solvers have potential for GPUs (coming soon)

- We look forward to getting feedback on these new features!

Our Multigrid / Parallel-in-Time Research Team and Alumni


Rob Falgout


Ruipeng Li


Victor Magri


Wayne Mitchell


Daniel Osei-Kuffuor


Ulrike Yang

Alumni:

Allison Baker
Chuck Baldwin
Andrew Barker
Guillermo Castilla
Edmond Chow
Andy Cleary
Veselin Dobrev
Noah Elliott
Stefanie Günther
David Gardner
Hormozd Gahvari
Van Henson
Ellen Hill
David Hysom
Jim Jones
Tzanio Kolev
Mike Lambert
Matthieu Lecouvez
Barry Lee
Sarah Osborn
Jeff Painter
Anders Petersson
Jacob Schroder
Bjorn Sjogreen
Charles Tong
Tom Treadway
Deborah Walker
Lu Wang
Carol Woodward
Panayot Vassilevski

- University collaborators and former interns
  - CU Boulder (Manteuffel, McCormick, Ruge, O'Neill, Southworth), Penn State (Brannick, Shen), Michigan Tech (Ong), U Wuppertal (Friedhoff, Kahl), Memorial University (MacLachlan), U Virginia (Gao), U Illinois (Gropp, Olson, Bienz), U Bordeaux (Ramet, Richefort)
- Software, publications, and other information

hypre http://llnl.gov/casc/hypre

XBRAID http://llnl.gov/casc/xbraid

Hyperbolic PinT


Jacob Schroder


David Vargas


Hans De Sterck


Oliver Krzysik

Thank You!

CASC

Lawrence Livermore National Laboratory