Research and Development Team

Former

- Allison Baker
- Chuck Baldwin
- Guillermo Castilla
- Edmond Chow
- Andy Cleary
- Noah Elliott
- Van Henson
- Ellen Hill
- David Hysom
- Jim Jones
- Mike Lambert
- Barry Lee
- Jeff Painter
- Charles Tong
- Tom Treadway
- Deborah Walker

Rob Falgout
Hormozd Gahvari
Tzanio Kolev
Ruipeng Li
Daniel Osei-Kuffuor
Jacob Schroder
Panayot Vassilevski
Lu Wang
Ulrike Yang

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

- Introduction / Motivation
- Getting Started / Linear System Interfaces
  - Structured-Grid Interface (Struct)
  - Semi-Structured-Grid Interface (SStruct)
- Solvers and Preconditioners
- Additional Information
- Introduction to hands-on exercises
Multigrid solvers have $O(N)$ complexity, 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.

- **Error on the fine grid**
- **Error approximated on a smaller coarse grid**
- **Restriction**
  - **Smoothing (relaxation)**
  - **Prolongation (interpolation)**

Algebraic multigrid (AMG) only uses matrix coefficients. No actual grids!
Parallel AMG in *hype* now scales to 1.1M cores on Sequoia (IBM BG/Q)

- $m \times n$ denotes $m$ MPI tasks and $n$ OpenMP threads per node
- Largest problem above: 72B unknowns on 1.1M cores
Getting Started

- **Before writing your code:**
  - choose a linear system interface
  - choose a solver / preconditioner
  - choose a matrix type that is compatible with your solver / preconditioner and system interface

- **Now write your code:**
  - build auxiliary structures (e.g., grids, stencils)
  - build matrix/vector through system interface
  - build solver/preconditioner
  - solve the system
  - get desired information from the solver
(Conceptual) linear system interfaces are necessary to provide “best” solvers and data layouts.
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
Currently, hypre supports four system interfaces

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

- Semi-Structured-Grid (**SSstruct**)  
  - *grids that are mostly structured*

- Finite Element (**FEI**)  
  - *unstructured grids with finite elements*

- Linear-Algebraic (**IJ**)  
  - *general sparse linear systems*

- More about the first two next…
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 scalar grid data is always associated with cell centers (unlike the more general SStruct interface).
Structured-Grid System Interface (Struct)

- There are four basic steps involved:
  - set up the Grid
  - set up the Stencil
  - set up the Matrix
  - set up the right-hand-side Vector

- Consider the following 2D Laplacian problem

\[
\begin{cases}
-\nabla^2 u = f & \text{in the domain} \\
u = g & \text{on the boundary}
\end{cases}
\]
Structured-grid finite volume example:

Standard 5-point finite volume discretization

Partition and distribute

Process 0

Process 1

(-3, 1)
Structured-grid finite volume example:
Setting up the grid on process 0

Create the grid object

```c
HYPRE_StructGrid grid;
int ndim = 2;
HYPRE_StructGridCreate(MPI_COMM_WORLD, ndim, &grid);
```
Structured-grid finite volume example:
Setting up the grid on process 0

Set grid extents for first box

```
int ilo0[2] = {-3,1};
int iup0[2] = {-1,2};
HYPRE_StructGridSetExtents(grid, ilo0, iup0);
```
Structured-grid finite volume example:
Setting up the grid on process 0

```
int ilol[2] = {0,1};
int iup1[2] = {2,4};

HYPRE_StructGridSetExtents(grid, ilol, iup1);
```

Set grid extents for second box
Structured-grid finite volume example:
Setting up the grid on process 0

Assemble the grid

HYPRE_StructGridAssemble(grid);
Structured-grid finite volume example: Setting up the stencil (all processes)

```c
HYPRE_StructStencil stencil;
int ndim = 2;
int size = 5;

HYPRE_StructStencilCreate(ndim, size, &stencil);
```
Structured-grid finite volume example:
Setting up the stencil (all processes)

```
int entry = 0;
int offset[2] = {0,0};

HYPRE_StructStencilSetElement(stencil, entry, offset);
```
Structured-grid finite volume example: Setting up the stencil (all processes)

```
int entry = 1;
int offset[2] = {-1,0};

HYPRE_StructStencilSetElement(stencil, entry, offset);
```
Structured-grid finite volume example:
Setting up the stencil (all processes)

```
int entry = 2;
int offset[2] = {1,0};

HYPRE_StructStencilSetElement(stencil, entry, offset);
```
Structured-grid finite volume example:
Setting up the stencil (all processes)

```
int entry = 3;
int offset[2] = {0,-1};

HYPRE_StructStencilSetElement(stencil, entry, offset);
```
Structured-grid finite volume example:
Setting up the stencil (all processes)

Set stencil entries

```c
int entry = 4;
int offset[2] = {0,1};
HYPRE_StructStencilSetElement(stencil, entry, offset);
```
Structured-grid finite volume example:
Setting up the stencil (all processes)

That’s it!

There is no assemble routine
Structured-grid finite volume example:
Setting up the matrix on process 0

```
HYPRE_StructMatrix A;
double vals[24] = {4, -1, 4, -1, ...};
int nentries = 2;
int entries[2] = {0,3};

HYPRE_StructMatrixCreate(MPI_COMM_WORLD, grid, stencil, &A);
HYPRE_StructMatrixInitialize(A);
HYPRE_StructMatrixSetBoxValues(A, ilo0, iup0, nentries, entries, vals);
HYPRE_StructMatrixSetBoxValues(A, ilo1, iup1, nentries, entries, vals);

/* set boundary conditions */
...
HYPRE_StructMatrixAssemble(A);
```
Structured-grid finite volume example:
Setting up the matrix bc’s on process 0

```c
int ilo[2] = {-3, 1};
int iup[2] = {2, 1};
double vals[6] = {0, 0, ...};
int nentries = 1;

/* set interior coefficients */
...

/* implement boundary conditions */
...

i = 3;
HYPRE_StructMatrixSetBoxValues(A,
    ilo, iup, nentries, &i, vals);

/* complete implementation of bc’s */
...
```
A structured-grid finite volume example:
Setting up the right-hand-side vector on process 0

```c
HYPRE_StructVector b;
double vals[12] = {0, 0, ...};

HYPRE_StructVectorCreate(MPI_COMM_WORLD, grid, &b);
HYPRE_StructVectorInitialize(b);
HYPRE_StructVectorSetBoxValues(b, ilo0, iup0, vals);
HYPRE_StructVectorSetBoxValues(b, ilo1, iup1, vals);
HYPRE_StructVectorAssemble(b);
```

Diagram showing a structured grid with values set at indices `(-3,1)` and `(2,4)`.
Symmetric Matrices

- Some solvers support symmetric storage

- **Between** `Create()` and `Initialize()`, call:
  
  ```
  HYPRE_StructMatrixSetSymmetric(A, 1);
  ```

- For best efficiency, only set half of the coefficients

\[
\begin{pmatrix}
(0,1) \\
(0,0) (1,0)
\end{pmatrix} \leftrightarrow \begin{pmatrix}
S2 \\
S0 & S1
\end{pmatrix}
\]

- This is enough info to recover the full 5-pt stencil
Semi-Structured-Grid System Interface (SStruct)

- Allows more general grids:
  - Grids that are mostly (but not entirely) structured
  - Examples: block-structured grids, structured adaptive mesh refinement grids, overset grids

Block-Structured

Adaptive Mesh Refinement

Overset
Semi-Structured-Grid System Interface (SStruct)

- Allows more general PDE’s
  - Multiple variables (system PDE’s)
  - Multiple variable types (cell centered, face centered, vertex centered, …)

Variables are referenced by the abstract cell-centered index to the left and down
The interface calls are very similar to SStruct.

The SStruct grid is composed out of structured grid parts.

A graph enables nearly arbitrary relationships between parts.

The graph is constructed from stencils or finite elements plus additional data-coupling information set either:
- directly with `GraphAddEntries()`, or
- by relating parts with `GridSetNeighborPart()` and `GridSetSharedPart()`

We will briefly consider two examples:
- block-structured grid using stencils
- star-shaped grid with finite elements
Consider the following block-structured grid discretization of the diffusion equation

\[-\nabla \cdot K \nabla u + \sigma u = f\]

A block-structured grid with 3 variable types

The 3 discretization stencils
Block-structured grid example (SStruct)

- The **Grid** is described via 5 logically-rectangular parts.

- We assume 5 processes such that process $p$ owns part $p$ (user defines the distribution).

- Consider the interface calls made by process 3
  - Part 3 is set up similarly to Struct, then `GridSetNeighborPart()` is used to connect it to parts 2 and 4.
Block-structured grid example:
some comments on `SetNeighborPart()`

All parts related via this routine must have consistent lists of variables and types.

Some variables on different parts become “the same.”

Variables may have different types on different parts (e.g., y-face on part 3 and x-face on part 2).
Block-structured grid example:
Setting up the three stencils (all processes)

- Setting up a stencil is similar to the Struct interface, requiring only one additional variable argument
  - Example: Above y-face stencil is coupled to variables of types x-face, y-face, and cell-centered
Finite element (FEM) style interface for SStruct as an alternative to stencils

- Beginning with *hypre* version 2.6.0b

- `GridSetSharedPart()` is similar to `SetNeighborPart`, but allows one to specify shared cells, faces, edges, or vertices

- `GridSetFEMOrdering()` sets the ordering of the unknowns in an element (always a cell)

- `GraphSetFEM()` indicates that an FEM approach will be used to set values instead of a stencil approach

- `GraphSetFEMSparsity()` sets the nonzero pattern for the stiffness matrix

- `MatrixAddFEMValues()` and `VectorAddFEMValues()`

- See examples: `ex13.c`, `ex14.c`, and `ex15.c`
Finite Element (FEM) example (SStruct)

- FEM nodal discretization of the Laplace equation on a star-shaped domain

\[
\begin{align*}
-\nabla^2 u &= 1 \quad \text{in } \Omega \\
 u &= 0 \quad \text{on } \Gamma
\end{align*}
\]

- FEM stiffness matrix

\[
\begin{pmatrix}
0 & 1 & 2 & 3 \\
0 & 4 - k & -1 & -2 + k & -1 \\
1 & -1 & 4 + k & -1 & -2 - k \\
2 & -2 + k & -1 & 4 - k & -1 \\
3 & -1 & -2 - k & -1 & 4 + k
\end{pmatrix} \alpha
\]

\[\alpha = (6 \sin(\gamma))^{-1}, \quad k = 3 \cos(\gamma), \quad \gamma = \pi/3\]

See example code ex14.c
FEM example (SStruct)

- The Grid is described via 6 logically-rectangular parts
- We assume 6 processes, where process $p$ owns part $p$
- The Matrix is assembled from stiffness matrices (no stencils)

```c
int part = 0, index[2] = {i,j};
double values[16] = {...};
HYPRE_SStructMatrixAddFEMValues(A, part, index, values);
```
Building different matrix/vector storage formats with the SStruct interface

- Efficient preconditioners often require specific matrix/vector storage schemes

- Between \texttt{Create()} and \texttt{Initialize()}, call:
  
  \begin{verbatim}
  HYPRE_SStructMatrixSetObjectType(A, HYPRE_PARCSR);
  \end{verbatim}

- After \texttt{Assemble()}, call:
  
  \begin{verbatim}
  HYPRE_SStructMatrixGetObject(A, &parcsr_A);
  \end{verbatim}

- Now, use the \texttt{ParCSR} matrix with compatible solvers such as BoomerAMG (algebraic multigrid)
Current solver / preconditioner availability via *hypre*‘s linear system interfaces

<table>
<thead>
<tr>
<th>Data Layouts</th>
<th>Solvers</th>
<th>System Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured</td>
<td>Jacobi, SMG, PFMG</td>
<td>Struct, SStruct, FEI, IJ</td>
</tr>
<tr>
<td></td>
<td>Split, SysPFMG</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Maxwell</td>
<td></td>
</tr>
<tr>
<td>Semi-structured</td>
<td>AMS, ADS, BoomerAMG</td>
<td>FEI, IJ</td>
</tr>
<tr>
<td></td>
<td>MLI, ParaSails</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Euclid</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PILUT</td>
<td></td>
</tr>
<tr>
<td>Sparse matrix</td>
<td>PCG, GMRES, BiCGSTAB</td>
<td>FEI, IJ</td>
</tr>
<tr>
<td></td>
<td>Hybrid</td>
<td></td>
</tr>
<tr>
<td>Matrix free</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lawrence Livermore National Laboratory
Setup and use of solvers is largely the same (*see Reference Manual for details*).

- **Create the solver**
  ```c
  HYPRE_SolverCreate(MPI_COMM_WORLD, &solver);
  ```

- **Set parameters**
  ```c
  HYPRE_SolverSetTol(solver, 1.0e-06);
  ```

- **Prepare to solve the system**
  ```c
  HYPRE_SolverSetup(solver, A, b, x);
  ```

- **Solve the system**
  ```c
  HYPRE_SolverSolve(solver, A, b, x);
  ```

- **Get solution info out via system interface**
  ```c
  HYPRE_StructVectorGetValues(struct_x, index, values);
  ```

- **Destroy the solver**
  ```c
  HYPRE_SolverDestroy(solver);
  ```
Solver example: SMG-PCG

/* define preconditioner (one symmetric V(1,1)-cycle) */
HYPRE_StructSMGCreate(MPI_COMM_WORLD, &precond);
HYPRE_StructSMGSetMaxIter(precond, 1);
HYPRE_StructSMGSetTol(precond, 0.0);
HYPRE_StructSMGSetZeroGuess(precond);
HYPRE_StructSMGSetNumPreRelax(precond, 1);
HYPRE_StructSMGSetNumPostRelax(precond, 1);

HYPRE_StructPCGCreate(MPI_COMM_WORLD, &solver);
HYPRE_StructPCGSetTol(solver, 1.0e-06);

/* set preconditioner */
HYPRE_StructPCGSetPrecond(solver,
    HYPRE_StructSMGSolve, HYPRE_StructSMGSetup, precond);

HYPRE_StructPCGSetup(solver, A, b, x);
HYPRE_StructPCGSolve(solver, A, b, x);
SMG and PFMG are semicoarsening multigrid methods for structured grids

- **Interface**: Struct, SStruct
- **Matrix Class**: Struct

- SMG uses plane smoothing in 3D, where each plane “solve” is effected by one 2D V-cycle
- SMG is very robust
- PFMG uses simple pointwise smoothing, and is less robust

- **Constant-coefficient versions!**
BoomerAMG is an algebraic multigrid method for unstructured grids

- **Interface**: SStruct, FEI, 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
AMS is an auxiliary space Maxwell solver for unstructured grids

- **Interface:** SStruct, FEI, IJ
- **Matrix Class:** ParCSR

Solves definite problems:

\[ \nabla \times \alpha \nabla \times E + \beta E = f, \quad \alpha > 0, \beta \geq 0 \]

- Requires additional gradient matrix and mesh coordinates
- Variational form of Hiptmair-Xu
- Employs BoomerAMG
- Only for FE discretizations
- ADS is a related solver for FE grad-div problems.

Copper wire in air, conductivity jump of $10^6$

25x faster on 80M unknowns
ParaSAILS is an approximate inverse method for sparse linear systems

- **Interface:** SStruct, FEI, IJ
- **Matrix Class:** ParCSR

- Approximates the inverse of $A$ by a sparse matrix $M$ by minimizing the Frobenius norm of $I - AM$
- Uses graph theory to predict good sparsity patterns for $M$
Euclid is a family of Incomplete LU methods for sparse linear systems

- **Interface:** SStruct, FEI, IJ
- **Matrix Class:** ParCSR

- Obtains scalable parallelism via local and global reorderings
- Good for unstructured problems

http://www.cs.odu.edu/~pothen/Software/Euclid
Solver parameters can greatly impact performance!

- Defaults are usually chosen to perform well across some (possibly small) set of problems
  - They may be poor choices for other problems

- All parameters have pros and cons

- BoomerAMG example:
  - HMIS coarsening, long-range interpolation, and truncation reduce communication but degrade convergence
  - For Diffusion, this is an overall win in time-to-solution

- Don’t hesitate to contact hypre-support for guidance
Getting the code

- To get the code, go to

  http://www.llnl.gov/CASC/hypre/

- User’s / Reference Manuals can be downloaded directly

- Release includes example programs!
Building the library

- Usually, *hypre* can be built by typing `configure` then `make`.
- Configure supports several options (for usage information, type `configure --help`):
  - `configure --enable-debug` - turn on debugging
  - `configure --with-openmp` - use openmp
  - `configure --disable-fortran` - disable Fortran tests
  - `configure --with-CFLAGS=...` - set compiler flags

- CMake is also supported
  - Ideal for Windows environment

- See `INSTALL` file for more details
Calling *hypre* from Fortran

- **C code:**

  ```c
  HYPRE_IJVector vec;
  int nvalues, *indices;
  double *values;

  HYPRE_IJVectorSetValues(vec, nvalues, indices, values);
  ```

- **Corresponding Fortran code:**

  ```fortran
  integer*8 vec
  integer nvalues, indices(NVALUES)
  double precision values(NVALUES)

  call HYPRE_IJVectorSetValues(vec, nvalues, indices, values, ierr)
  ```
Reporting bugs, requesting features, usage questions

- Send email to the following address:
  
  hypre-support@llnl.gov

- We use an email-centered tool called Roundup to automatically tag and track issues
Introduction to hypre hands-on exercises

- Open hypre/README.html
Thank You!

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.