

Lawrence Livermore National Laboratory

HYPRE: High Performance Preconditioners

August 19, 2010



Robert D. Falgout

Center for Applied Scientific Computing

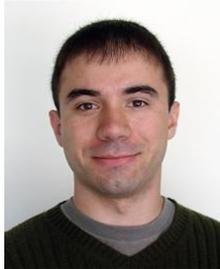
The *hypre* Team



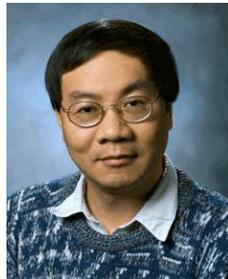
Allison Baker



Rob Falgout



Tzanio Kolev



Charles Tong



Panayot Vassilevski



Ulrike Yang

Former

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

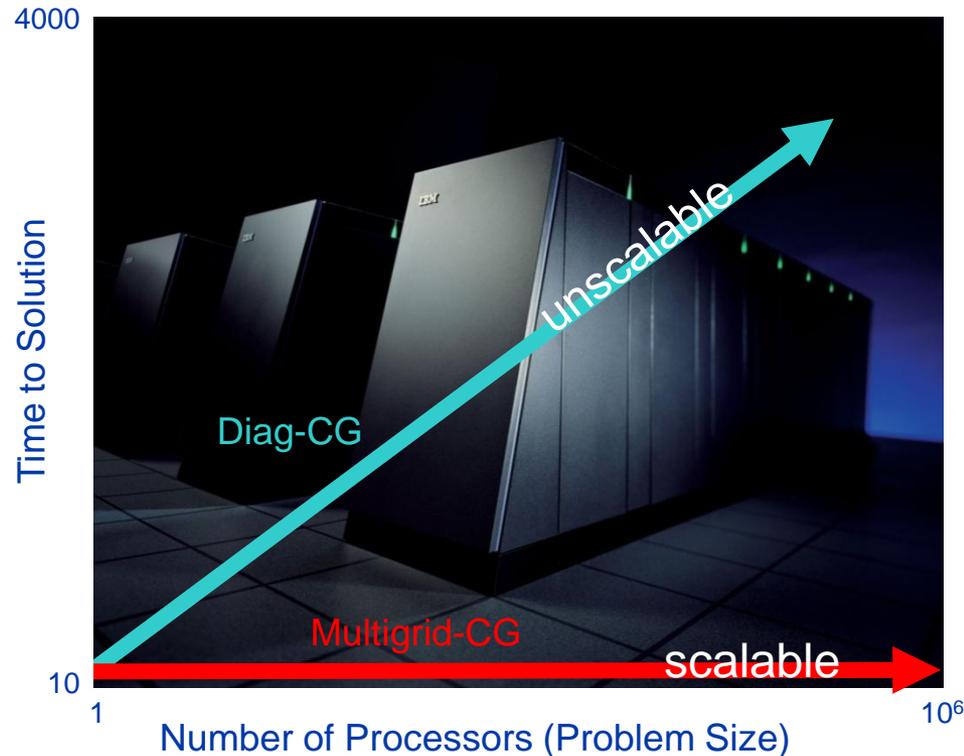
<http://www.llnl.gov/CASC/hypre/>

Outline

- Introduction / Motivation
- Getting Started / Linear System Interfaces
 - Structured-Grid Interface (`Struct`)
 - Semi-Structured-Grid Interface (`SStruct`)
 - Finite Element Interface (`FEI`)
 - Linear-Algebraic Interface (`IJ`)
- Solvers and Preconditioners
- Additional Information

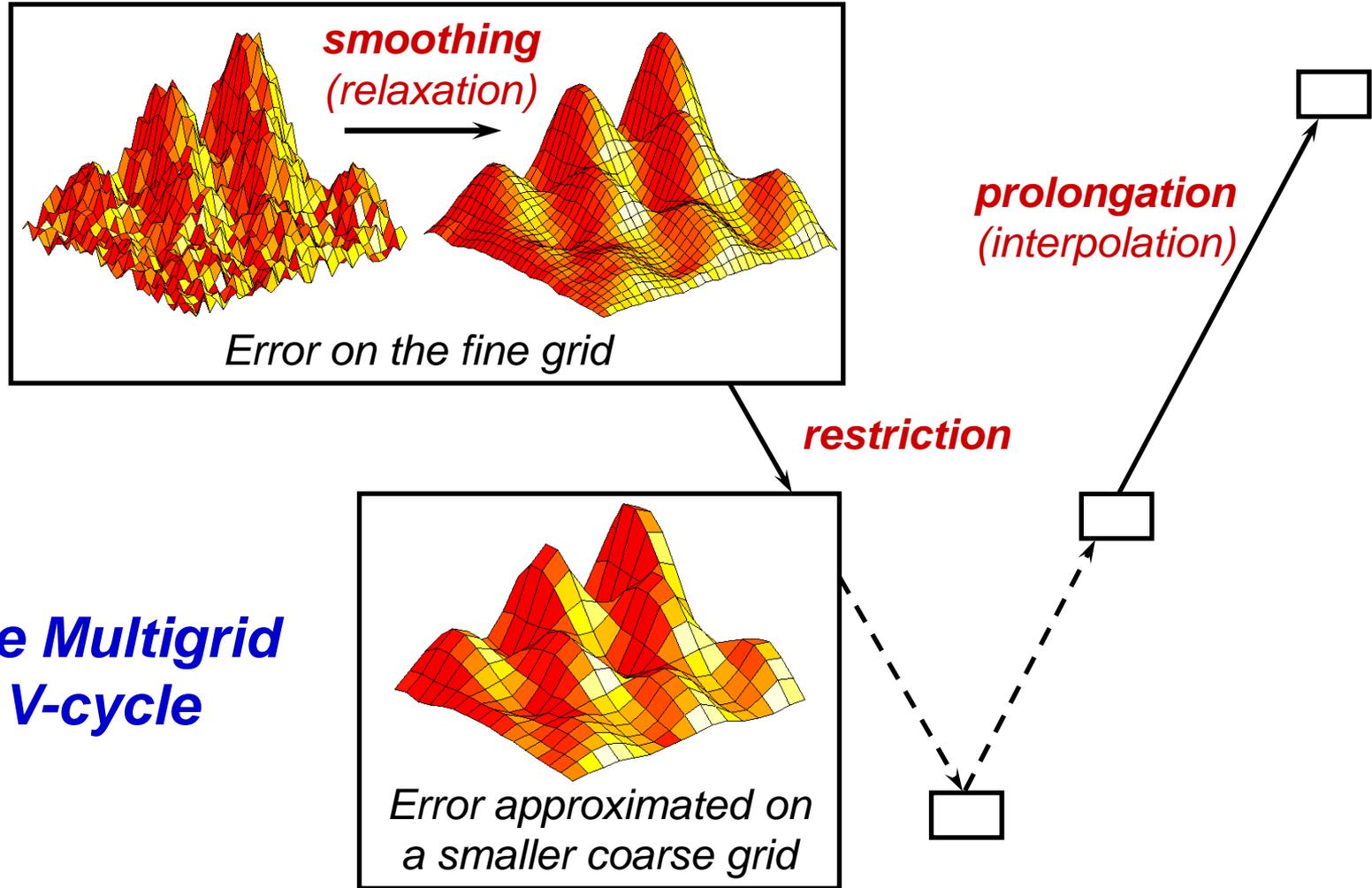


Multigrid linear solvers are optimal ($O(N)$ operations), and hence have good scaling potential



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

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



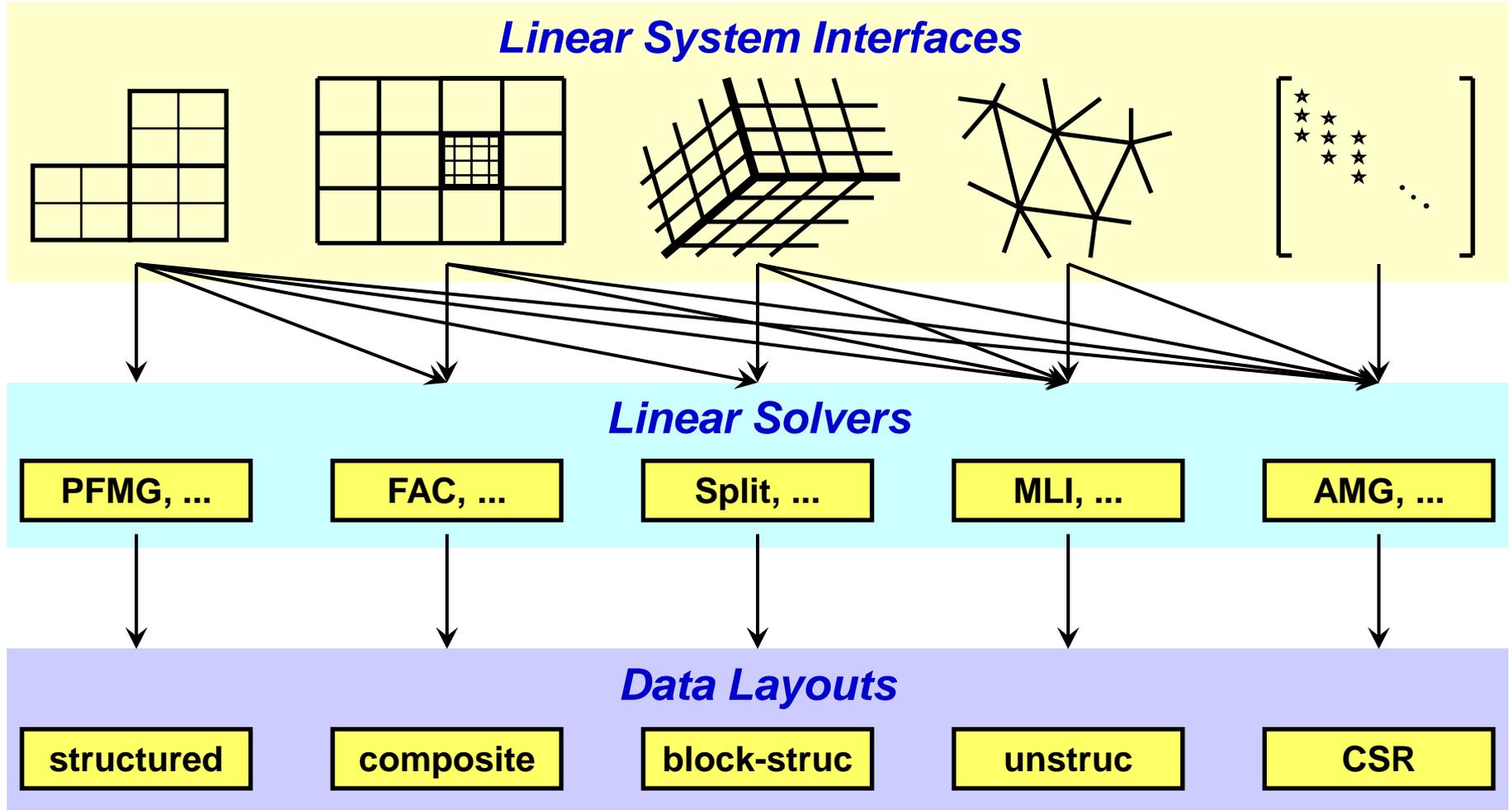
The Multigrid V-cycle

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



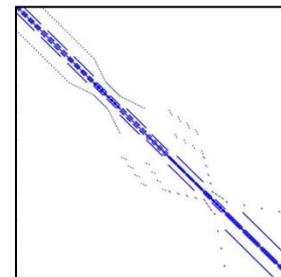
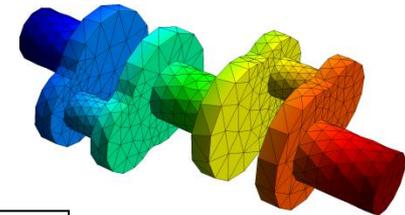
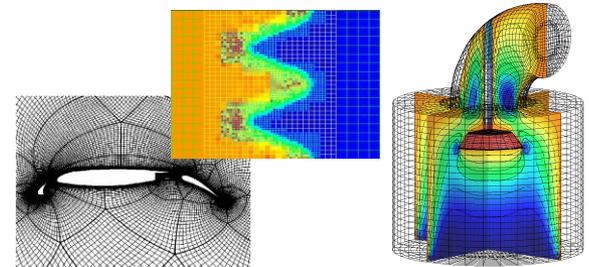
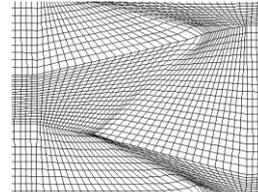
Why multiple interfaces? The key points

- 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



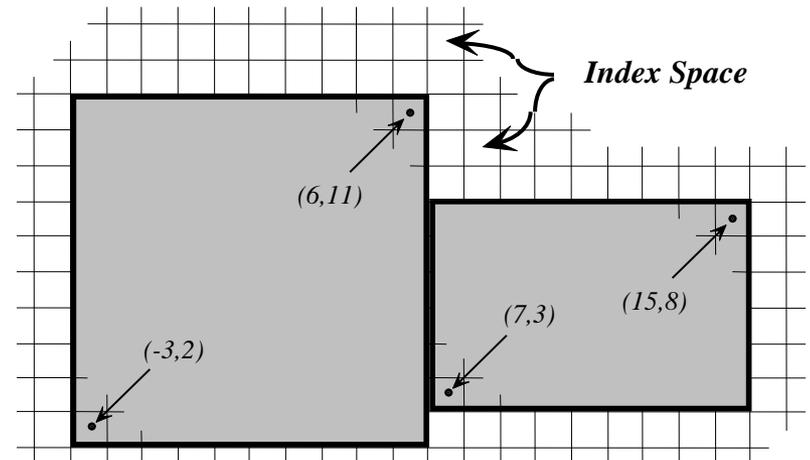
Currently, *hypra* supports four system interfaces

- Structured-Grid (`Struct`)
 - *logically rectangular grids*
- Semi-Structured-Grid (`SStruct`)
 - *grids that are mostly structured*
- Finite Element (`FEI`)
 - *unstructured grids with finite elements*
- Linear-Algebraic (`IJ`)
 - *general sparse linear systems*
- **More about each 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)

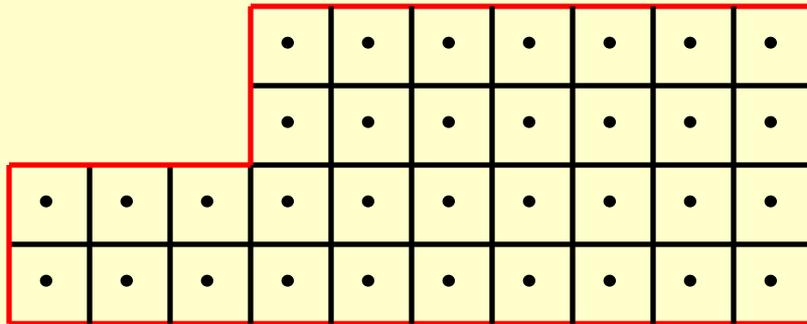


Structured-Grid System Interface (Struct)

- 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

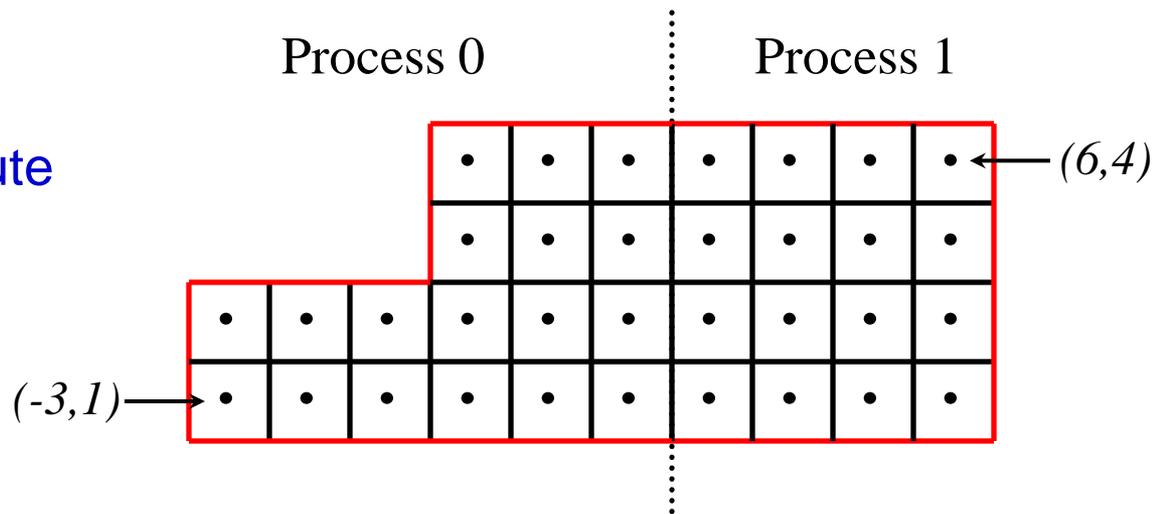
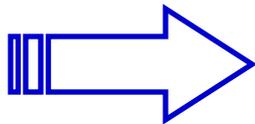
$$\begin{cases} -\nabla^2 u = f & \text{in the domain} \\ u = g & \text{on the boundary} \end{cases}$$

Structured-grid finite volume example:

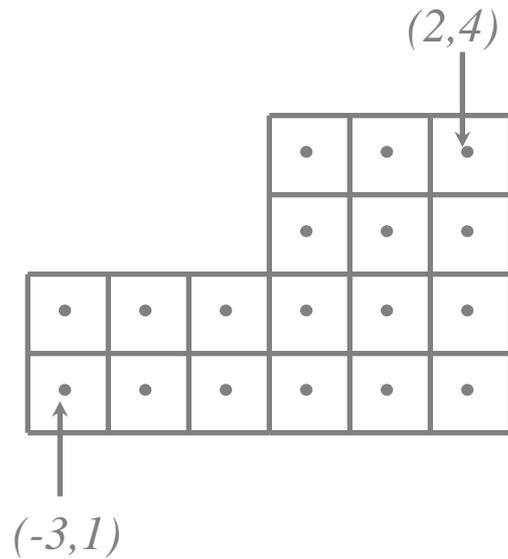


Standard 5-point finite volume discretization

Partition and distribute



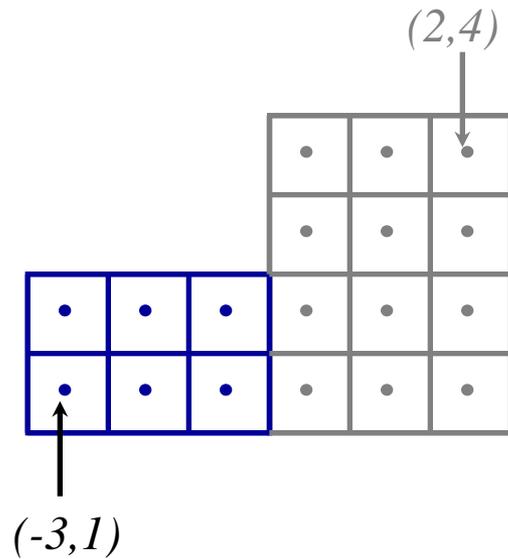
Structured-grid finite volume example: Setting up the grid on process 0



Create the grid object

```
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

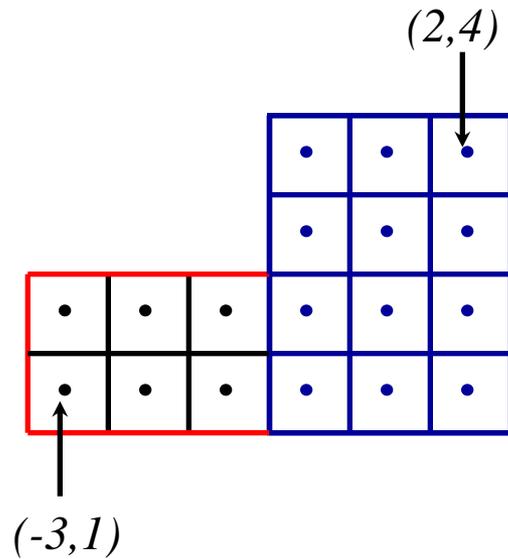


**Set grid extents for
first box**

```
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

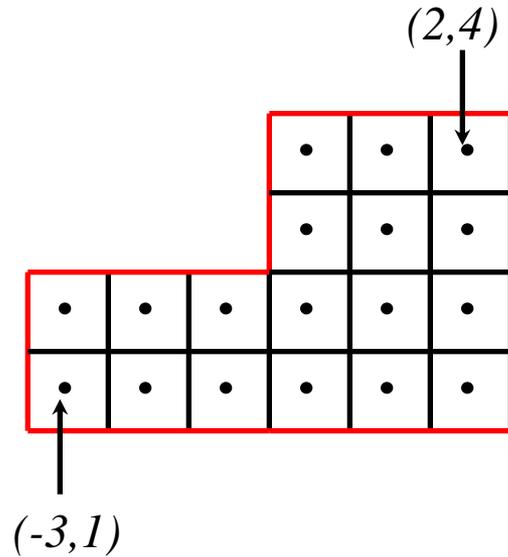


**Set grid extents for
second box**

```
int ilo1[2] = {0,1};  
int iup1[2] = {2,4};
```

```
HYPRE_StructGridSetExtents(grid, ilo1, iup1);
```

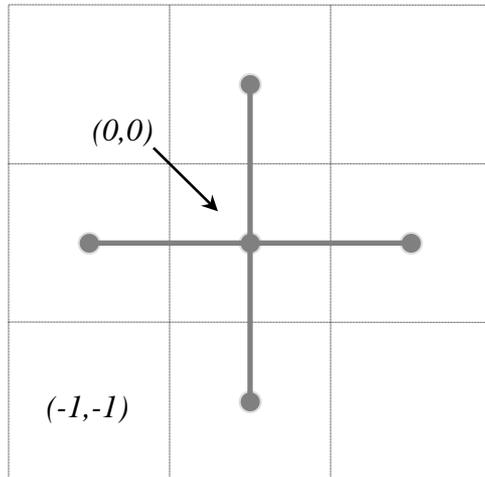
Structured-grid finite volume example: Setting up the grid on process 0



Assemble the grid

```
HYPRE_StructGridAssemble (grid) ;
```

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



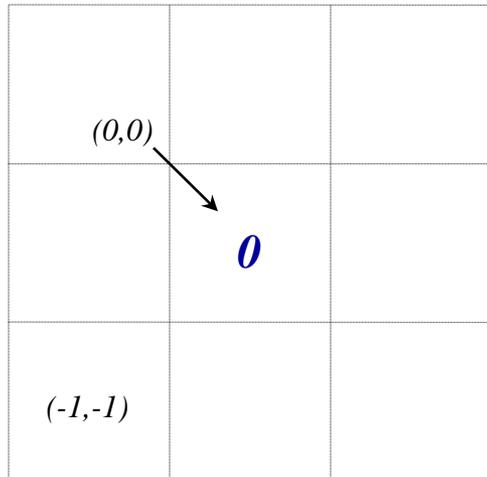
<i>stencil entries</i>	0	↔	(0, 0)	<i>geometries</i>
	1	↔	(-1, 0)	
	2	↔	(1, 0)	
	3	↔	(0,-1)	
	4	↔	(0, 1)	

**Create the stencil
object**

```
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)

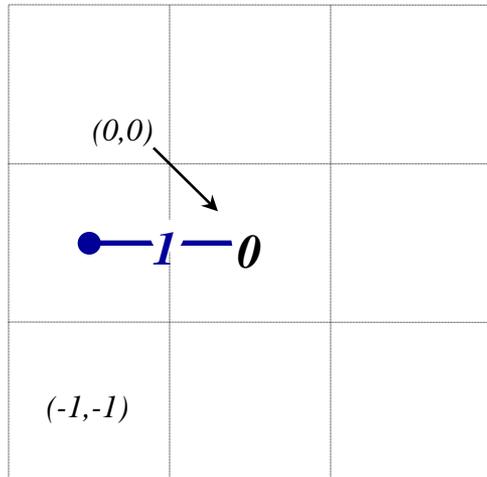


stencil entries	0 ↔ (0, 0)	geometries
	1 ↔ (-1, 0)	
	2 ↔ (1, 0)	
	3 ↔ (0, -1)	
	4 ↔ (0, 1)	

Set stencil entries

```
int entry = 0;  
int offset[2] = {0,0};  
  
HYPRE_StructStencilSetElement(stencil, entry, offset);
```

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



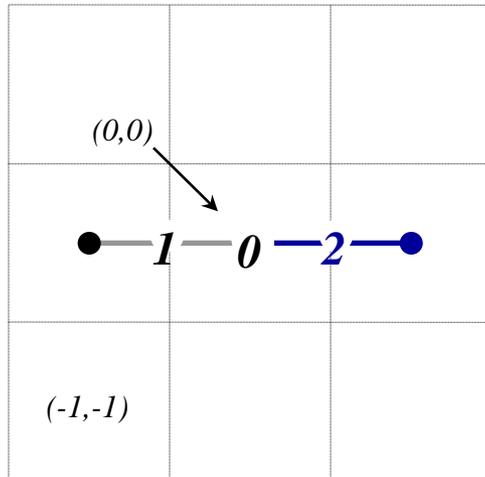
stencil entries	0	↔	(0, 0)	geometries
	1	↔	(-1, 0)	
	2	↔	(1, 0)	
	3	↔	(0, -1)	
	4	↔	(0, 1)	

Set stencil entries

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

```
HYPRE_StructStencilSetElement(stencil, entry, offset);
```

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



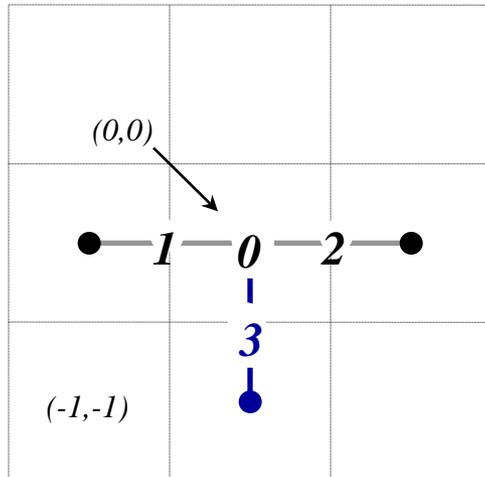
stencil entries	0	↔	(0, 0)	geometries
	1	↔	(-1, 0)	
	2	↔	(1, 0)	
	3	↔	(0,-1)	
	4	↔	(0, 1)	

Set stencil entries

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

```
HYPRE_StructStencilSetElement(stencil, entry, offset);
```

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



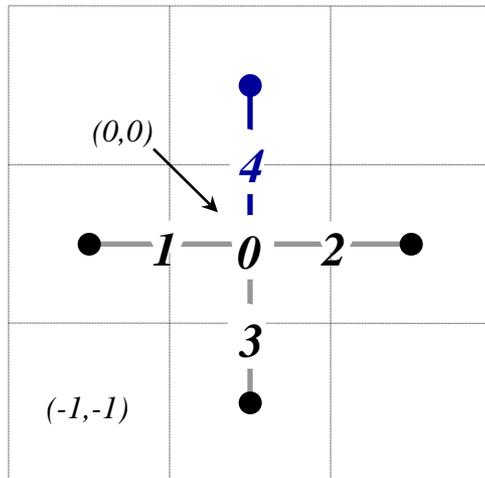
stencil entries	0	↔	(0, 0)	geometries
	1	↔	(-1, 0)	
	2	↔	(1, 0)	
	3	↔	(0, -1)	
	4	↔	(0, 1)	

Set stencil entries

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

```
HYPRE_StructStencilSetElement(stencil, entry, offset);
```

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

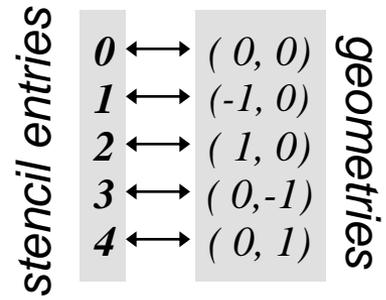
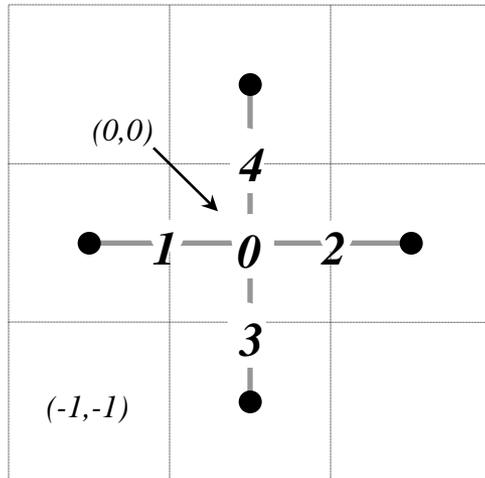


stencil entries	0	↔	(0, 0)	geometries
	1	↔	(-1, 0)	
	2	↔	(1, 0)	
	3	↔	(0, -1)	
	4	↔	(0, 1)	

Set stencil entries

```
int entry = 4;  
int offset[2] = {0,1};  
HYPRE_StructStencilSetElement(stencil, entry, offset);
```

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

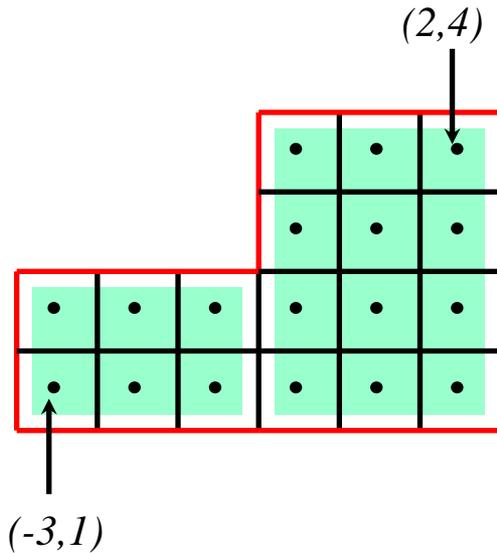


That's it!
**There is no assemble
routine**



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 */
```

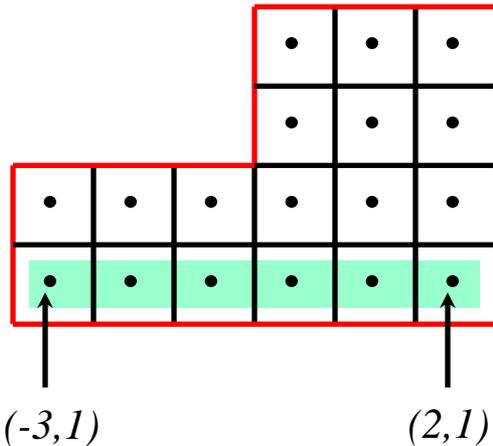
```
...
```

```
HYPRE_StructMatrixAssemble (A);
```

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

Structured-grid finite volume example :

Setting up the matrix bc's on process 0



$$\begin{pmatrix} & \mathbf{S4} & & & & \\ \mathbf{S1} & \mathbf{S0} & \mathbf{S2} & & & \\ & \mathbf{S3} & & & & \end{pmatrix} = \begin{pmatrix} & -1 & & & & \\ -1 & 4 & -1 & & & \\ & \mathbf{0} & & & & \end{pmatrix}$$

```

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

/* set interior coefficients */
...

/* implement boundary conditions */
...

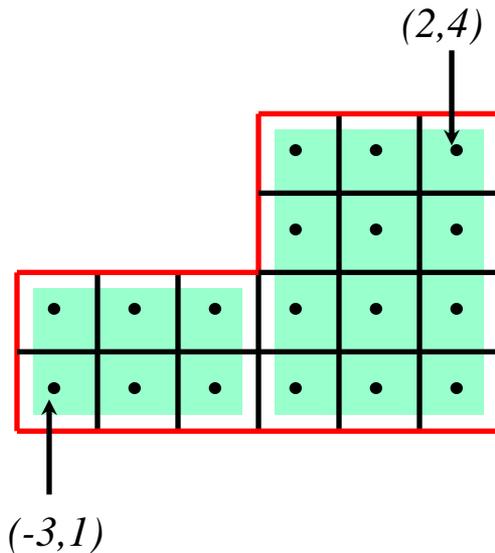
i = 3;
HYPRE_StructMatrixSetBoxValues(A,
    ilo, iup, nentries, &i, vals);

/* complete implementation of bc's */
...

```

A structured-grid finite volume example :

Setting up the right-hand-side vector on process 0



```
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);
```

Symmetric Matrices

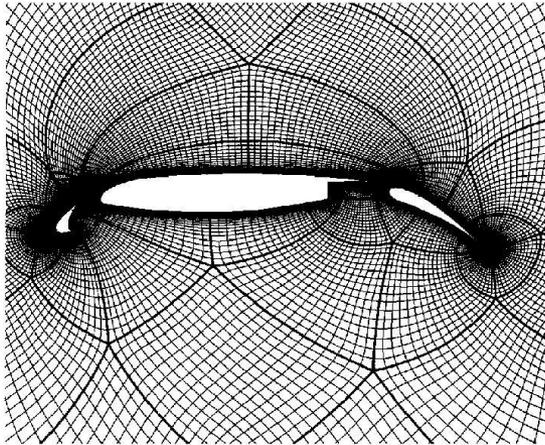
- Some solvers support symmetric storage
- Between `Create()` and `Initialize()`, call:
`HYPRE_StructMatrixSetSymmetric(A, 1);`
- For best efficiency, only set half of the coefficients

$$\begin{pmatrix} (0,1) \\ (0,0) (1,0) \end{pmatrix} \iff \begin{pmatrix} s2 \\ s0 \quad s1 \end{pmatrix}$$

- This is enough info to recover the full 5-pt stencil

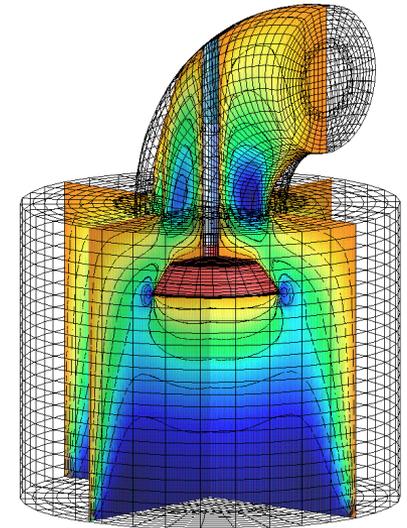
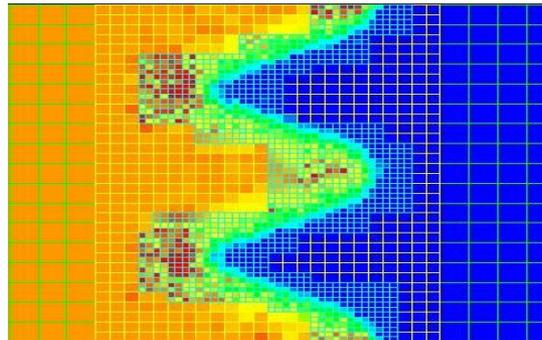
Semi-Structured-Grid System Interface (SStruct)

- 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

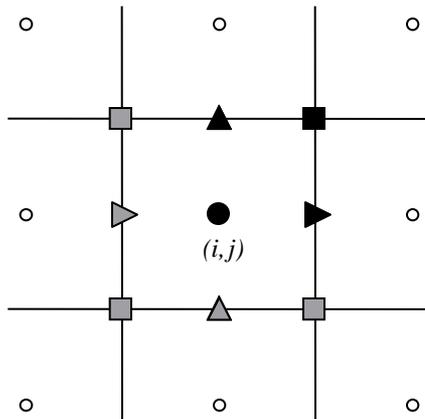
Adaptive Mesh Refinement



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

Semi-Structured-Grid System Interface (SStruct)

- The `SStruct` grid is composed out of a number of structured grid *parts*
- The interface uses a *graph* to allow nearly arbitrary relationships between part data
- The graph is constructed from stencils plus some additional data-coupling information set either
 - directly with `GraphAddEntries()`, or
 - by relating parts with `GridSetNeighborPart()`
- We will consider two examples:
 - block-structured grid
 - structured adaptive mesh refinement



Semi-Structured-Grid System Interface (SStruct)

- There are five basic steps involved:
 - set up the `Grid`
 - set up the `Stencils`
 - **set up the `Graph`**
 - set up the `Matrix`
 - set up the right-hand-side `Vector`

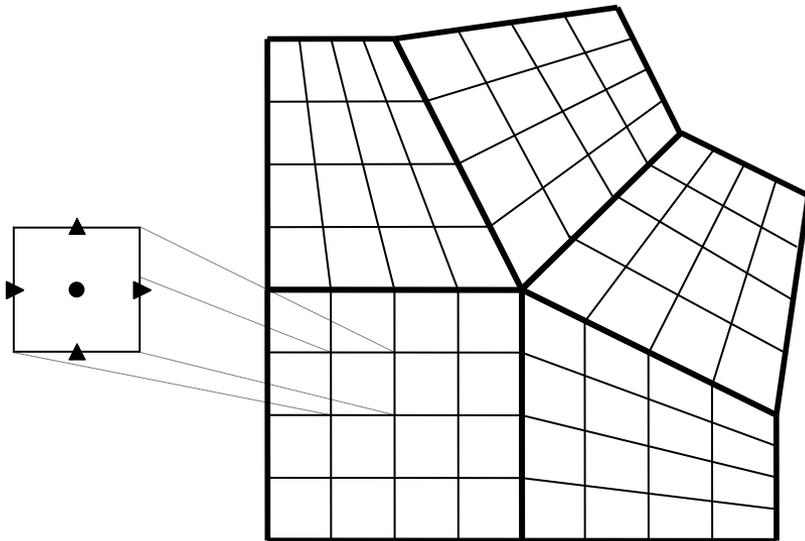


Block-structured grid example (SStruct)

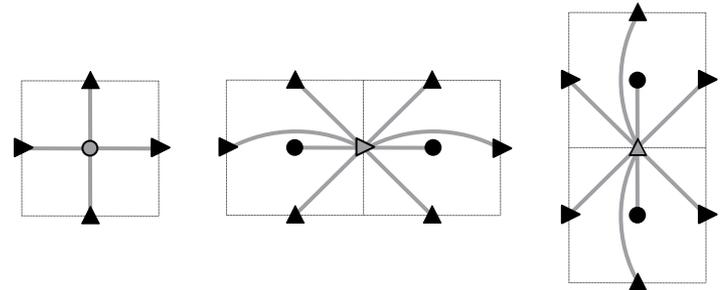
- Consider the following block-structured grid discretization of the diffusion equation

$$-\nabla \cdot \mathbf{K} \nabla u + \sigma u = f$$

A block-structured grid with
3 variable types

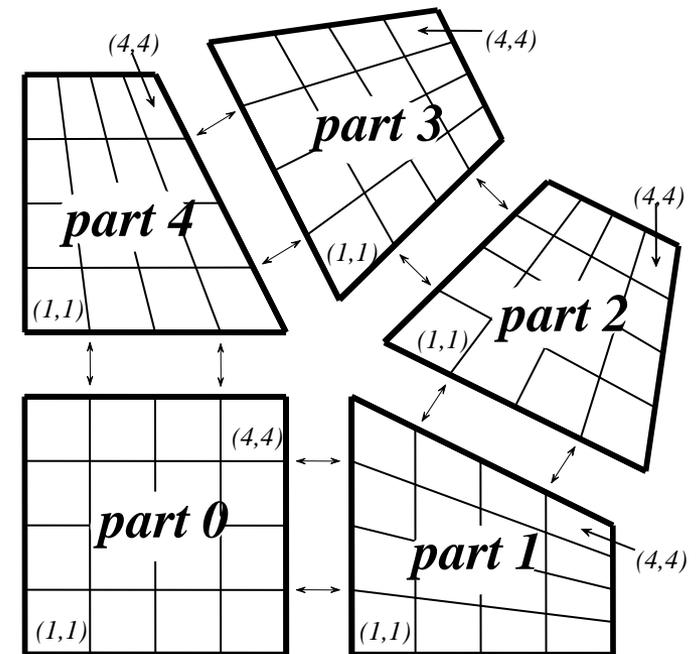


The 3 discretization stencils

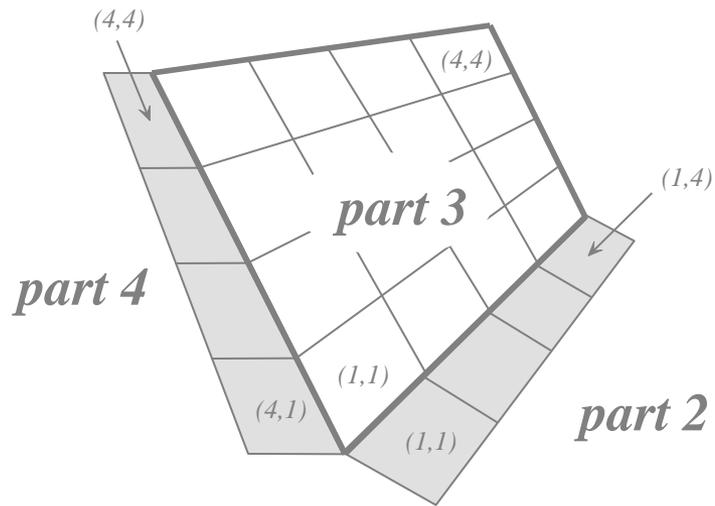


Block-structured grid example (SStruct)

- The `Grid` is described via 5 logically-rectangular parts
- We assume 5 processes such that process p owns part p (user defines the distribution)
- We consider the interface calls made by process 3



Block-structured grid example: Setting up the grid on process 3

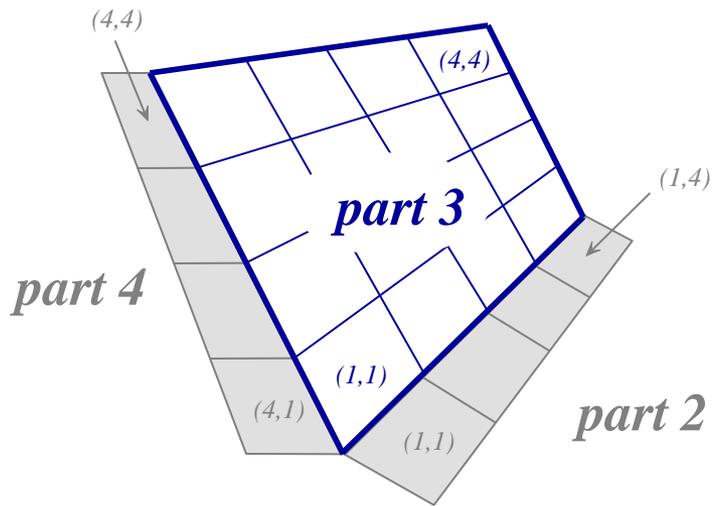


Create the grid object

```
HYPRE_SStructGrid grid;  
int ndim    = 2;  
int nparts  = 5;
```

```
HYPRE_SStructGridCreate(MPI_COMM_WORLD, ndim, nparts, &grid);
```

Block-structured grid example: Setting up the grid on process 3

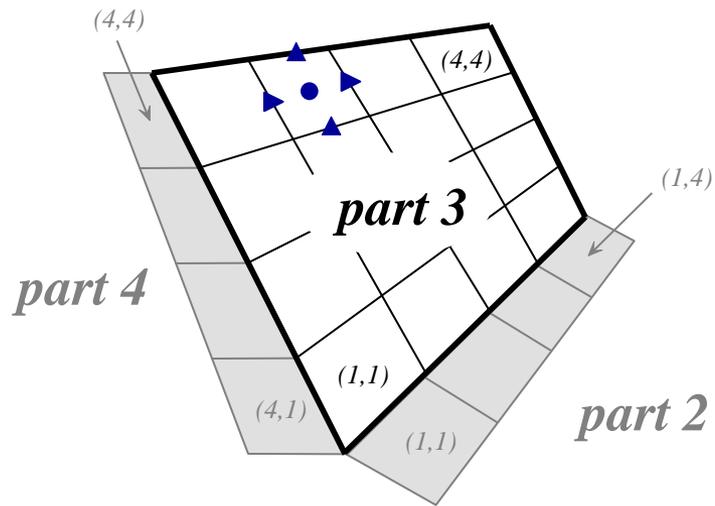


**Set grid extents for
part 3**

```
int part = 3;  
int ilower[2] = {1,1};  
int iupper[2] = {4,4};
```

```
HYPRE_SStructGridSetExtents(grid, part, ilower, iupper);
```

Block-structured grid example: Setting up the grid on process 3

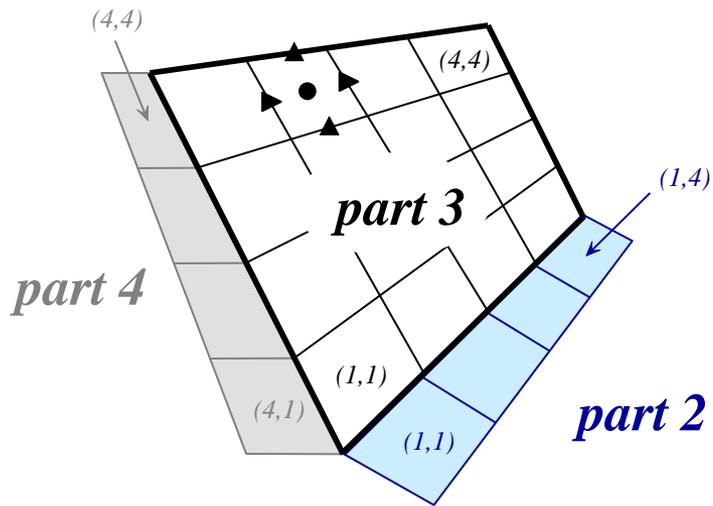


**Set grid variables for
part 3**

```
int part = 3;  
int nvars = 3;  
int vartypes[3] = {HYPRE_SSTRUCT_VARIABLE_CELL,  
                   HYPRE_SSTRUCT_VARIABLE_XFACE,  
                   HYPRE_SSTRUCT_VARIABLE_YFACE};
```

```
HYPRE_SStructGridSetVariables(grid, part, nvars, vartypes);
```

Block-structured grid example: Setting up the grid on process 3

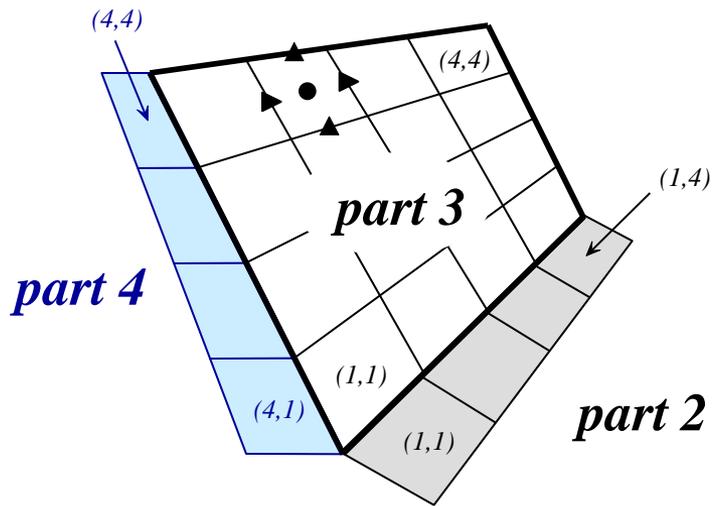


**Set spatial relationship
between parts 3 and 2**

```
int part = 3, nbor_part = 2;  
int ilower[2] = {1,0}, iupper[2] = {4,0};  
int nbor_ilower[2] = {1,1}, nbor_iupper[2] = {1,4};  
int index_map[2] = {1,0}, dir_map[2] = {1,-1};
```

```
HYPRE_SStructGridSetNeighborPart(grid, part, ilower, iupper,  
nbor_part, nbor_ilower, nbor_iupper, index_map, dir_map);
```

Block-structured grid example: Setting up the grid on process 3

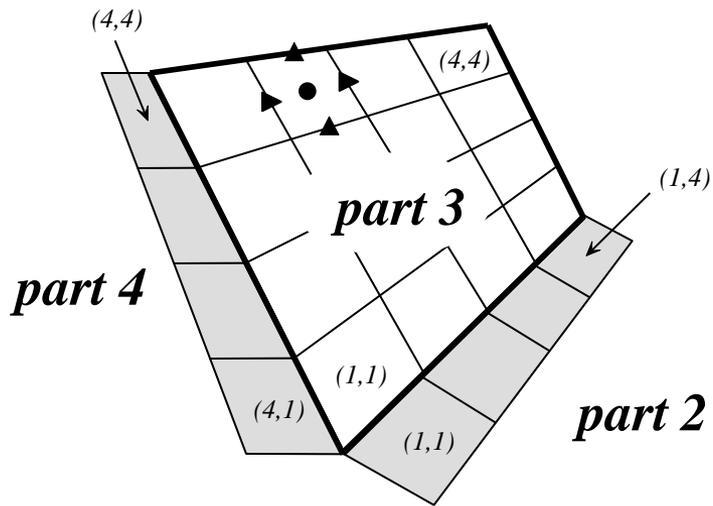


**Set spatial relationship
between parts 3 and 4**

```
int part = 3, nbor_part = 4;  
int ilower[2] = {0,1}, iupper[2] = {0,4};  
int nbor_ilower[2] = {4,1}, nbor_iupper[2] = {4,4};  
int index_map[2] = {0,1}, dir_map[2] = {1,1};
```

```
HYPRE_SStructGridSetNeighborPart(grid, part, ilower, iupper,  
nbor_part, nbor_ilower, nbor_iupper, index_map, dir_map);
```

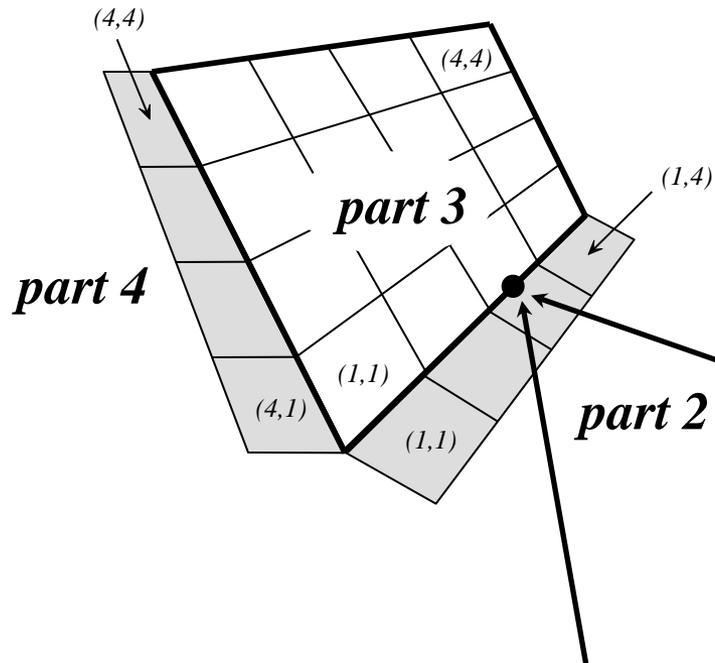
Block-structured grid example: Setting up the grid on process 3



Assemble the grid

```
HYPRE_SStructGridAssemble(grid);
```

Block-structured grid example: some comments on `SetNeighborPart()`

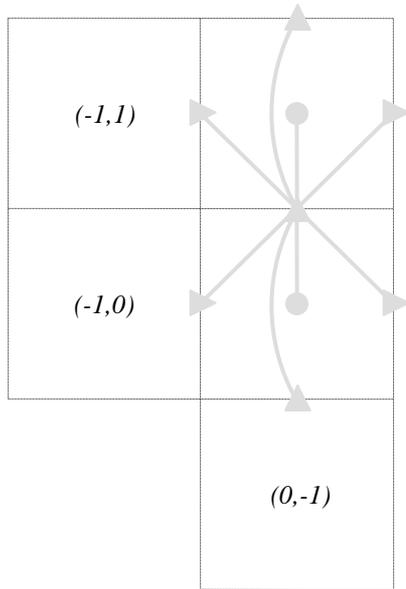


All parts related via this routine must have consistent lists of variables and types

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 y -face stencil (all processes)



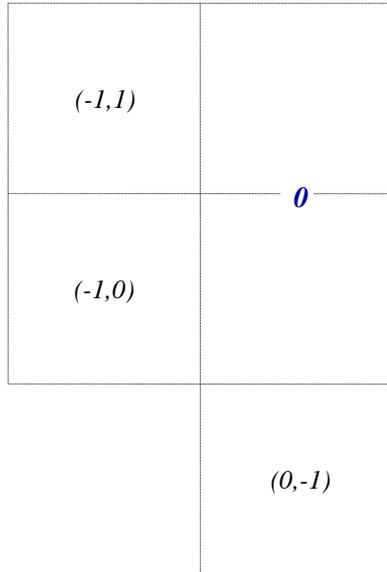
stencil entries	0	↔	(0,0);▲	geometries
	1	↔	(0,-1);▲	
	2	↔	(0,1);▲	
	3	↔	(0,0);●	
	4	↔	(0,1);●	
	5	↔	(-1,0);▶	
	6	↔	(0,0);▶	
	7	↔	(-1,1);▶	
	8	↔	(0,1);▶	

**Create the stencil
object**

```
HYPRE_SStructStencil stencil;  
int ndim = 2;  
int size = 9;
```

```
HYPRE_SStructStencilCreate(ndim, size, &stencil);
```

Block-structured grid example: Setting up the y -face stencil (all processes)



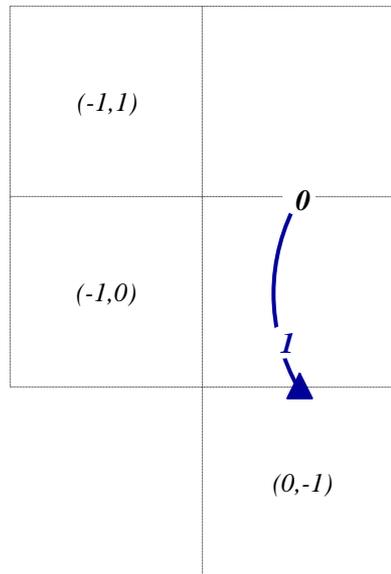
stencil entries	0	↔	(0,0);▲	geometries
	1	↔	(0,-1);▲	
	2	↔	(0,1);▲	
	3	↔	(0,0);●	
	4	↔	(0,1);●	
	5	↔	(-1,0);▶	
	6	↔	(0,0);▶	
	7	↔	(-1,1);▶	
	8	↔	(0,1);▶	

Set stencil entries

```
int entry = 0;  
int offset[2] = {0,0};  
int var = 2; /* the y-face variable number */
```

```
HYPRE_SStructSetStencilEntry(stencil, entry, offset, var);
```

Block-structured grid example: Setting up the y -face stencil (all processes)



	0	↔	(0,0);▲	
	1	↔	(0,-1);▲	
	2	↔	(0,1);▲	
	3	↔	(0,0);●	
	4	↔	(0,1);●	
	5	↔	(-1,0);▶	
	6	↔	(0,0);▶	
	7	↔	(-1,1);▶	
	8	↔	(0,1);▶	

stencil entries

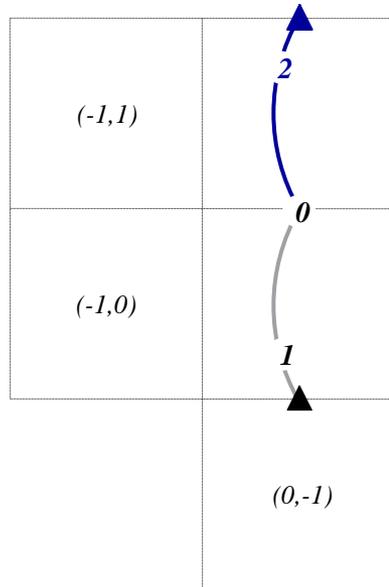
geometries

Set stencil entries

```
int entry = 1;  
int offset[2] = {0,-1};  
int var = 2; /* the y-face variable number */
```

```
HYPRE_SStructSetStencilEntry(stencil, entry, offset, var);
```

Block-structured grid example: Setting up the y -face stencil (all processes)



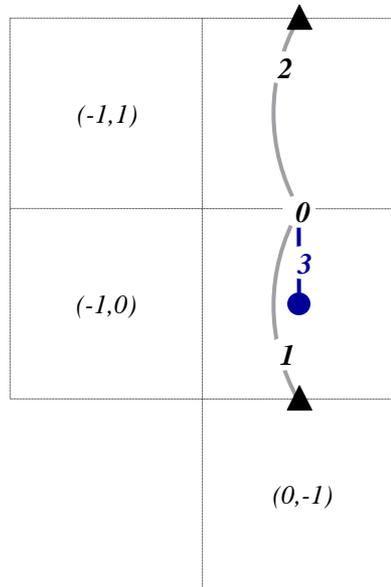
stencil entries	0	↔	(0,0);▲	geometries
	1	↔	(0,-1);▲	
	2	↔	(0,1);▲	
	3	↔	(0,0);●	
	4	↔	(0,1);●	
	5	↔	(-1,0);▶	
	6	↔	(0,0);▶	
	7	↔	(-1,1);▶	
	8	↔	(0,1);▶	

Set stencil entries

```
int entry = 2;  
int offset[2] = {0,1};  
int var = 2; /* the y-face variable number */
```

```
HYPRE_SStructSetStencilEntry(stencil, entry, offset, var);
```

Block-structured grid example: Setting up the y -face stencil (all processes)



0	↔	(0,0);▲
1	↔	(0,-1);▲
2	↔	(0,1);▲
3	↔	(0,0);●
4	↔	(0,1);●
5	↔	(-1,0);▶
6	↔	(0,0);▶
7	↔	(-1,1);▶
8	↔	(0,1);▶

stencil entries

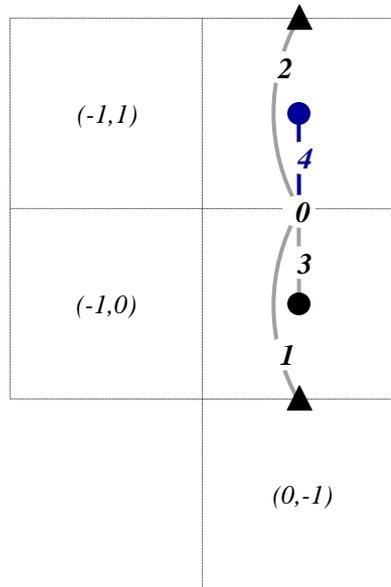
geometries

Set stencil entries

```
int entry = 3;  
int offset[2] = {0,0};  
int var = 0; /* the cell-centered variable number */
```

```
HYPRE_SStructSetStencilEntry(stencil, entry, offset, var);
```

Block-structured grid example: Setting up the y -face stencil (all processes)

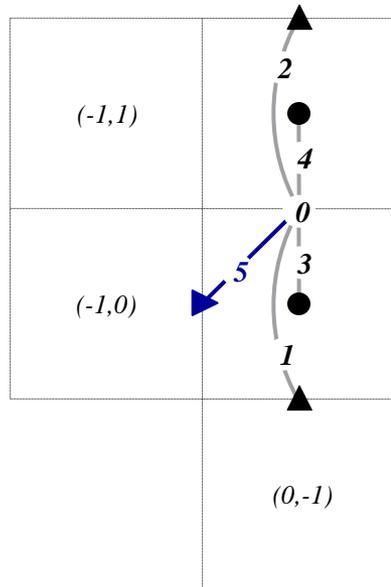


0	↔	$(0,0); \blacktriangle$	geometries
1	↔	$(0,-1); \blacktriangle$	
2	↔	$(0,1); \blacktriangle$	
3	↔	$(0,0); \bullet$	
4	↔	$(0,1); \bullet$	
5	↔	$(-1,0); \blacktriangleright$	
6	↔	$(0,0); \blacktriangleright$	
7	↔	$(-1,1); \blacktriangleright$	
8	↔	$(0,1); \blacktriangleright$	

Set stencil entries

```
int entry = 4;
int offset[2] = {0,1};
int var = 0; /* the cell-centered variable number */
HYPRE_SStructSetStencilEntry(stencil, entry, offset, var);
```

Block-structured grid example: Setting up the y -face stencil (all processes)



0	↔	(0,0);▲
1	↔	(0,-1);▲
2	↔	(0,1);▲
3	↔	(0,0);●
4	↔	(0,1);●
5	↔	(-1,0);▶
6	↔	(0,0);▶
7	↔	(-1,1);▶
8	↔	(0,1);▶

stencil entries

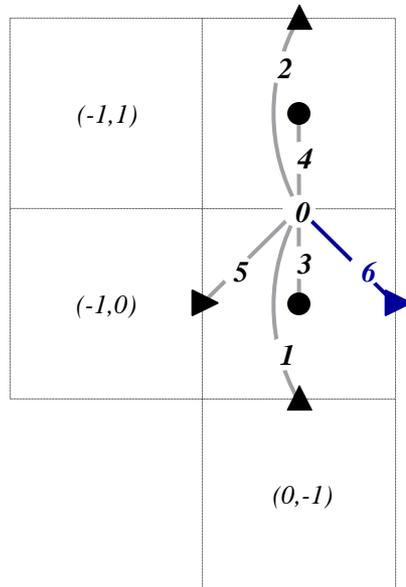
geometries

Set stencil entries

```
int entry = 5;  
int offset[2] = {-1,0};  
int var = 1; /* the x-face variable number */
```

```
HYPRE_SStructSetStencilEntry(stencil, entry, offset, var);
```

Block-structured grid example: Setting up the y -face stencil (all processes)



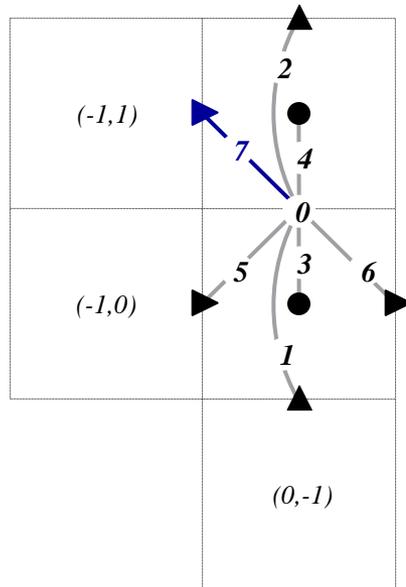
0	↔	(0,0);▲	geometries
1	↔	(0,-1);▲	
2	↔	(0,1);▲	
3	↔	(0,0);●	
4	↔	(0,1);●	
5	↔	(-1,0);▲	
6	↔	(0,0);▶	
7	↔	(-1,1);▶	
8	↔	(0,1);▶	

Set stencil entries

```
int entry = 6;  
int offset[2] = {0,0};  
int var = 1; /* the x-face variable number */
```

```
HYPRE_SStructSetStencilEntry(stencil, entry, offset, var);
```

Block-structured grid example: Setting up the y -face stencil (all processes)



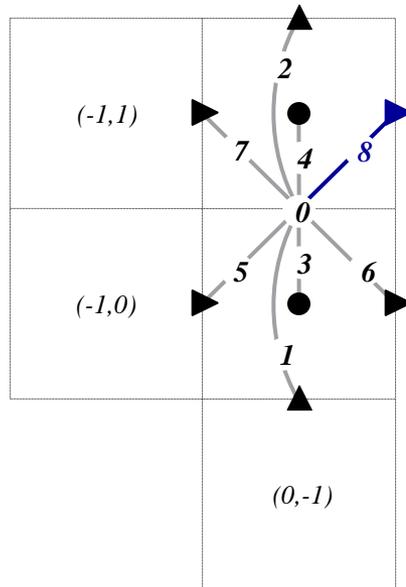
0	↔	(0,0);▲	geometries
1	↔	(0,-1);▲	
2	↔	(0,1);▲	
3	↔	(0,0);●	
4	↔	(0,1);●	
5	↔	(-1,0);▶	
6	↔	(0,0);▶	
7	↔	(-1,1);▶	
8	↔	(0,1);▶	

Set stencil entries

```
int entry = 7;  
int offset[2] = {-1,1};  
int var = 1; /* the x-face variable number */
```

```
HYPRE_SStructSetStencilEntry(stencil, entry, offset, var);
```

Block-structured grid example: Setting up the y -face stencil (all processes)



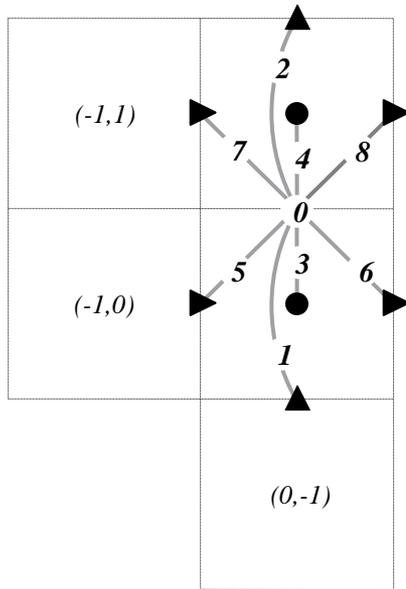
0	↔	(0,0);▲	geometries
1	↔	(0,-1);▲	
2	↔	(0,1);▲	
3	↔	(0,0);●	
4	↔	(0,1);●	
5	↔	(-1,0);▶	
6	↔	(0,0);▶	
7	↔	(-1,1);▶	
8	↔	(0,1);▶	

Set stencil entries

```
int entry = 8;
int offset[2] = {0,1};
int var = 1; /* the x-face variable number */
```

```
HYPRE_SStructSetStencilEntry(stencil, entry, offset, var);
```

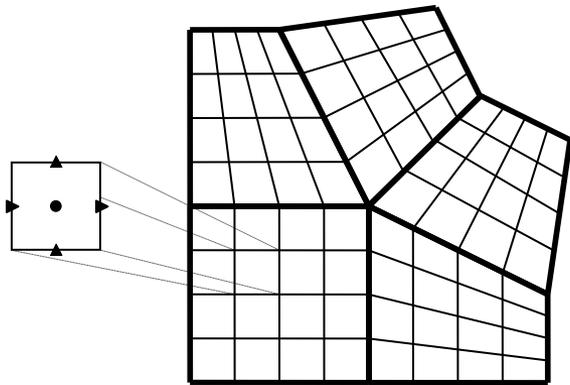
Block-structured grid example: Setting up the y -face stencil (all processes)



<i>stencil entries</i>	0	↔	(0,0);▲	<i>geometries</i>
	1	↔	(0,-1);▲	
	2	↔	(0,1);▲	
	3	↔	(0,0);●	
	4	↔	(0,1);●	
	5	↔	(-1,0);▶	
	6	↔	(0,0);▶	
	7	↔	(-1,1);▶	
	8	↔	(0,1);▶	

That's it!
**There is no assemble
routine**

Block-structured grid example: Setting up the graph on process 3

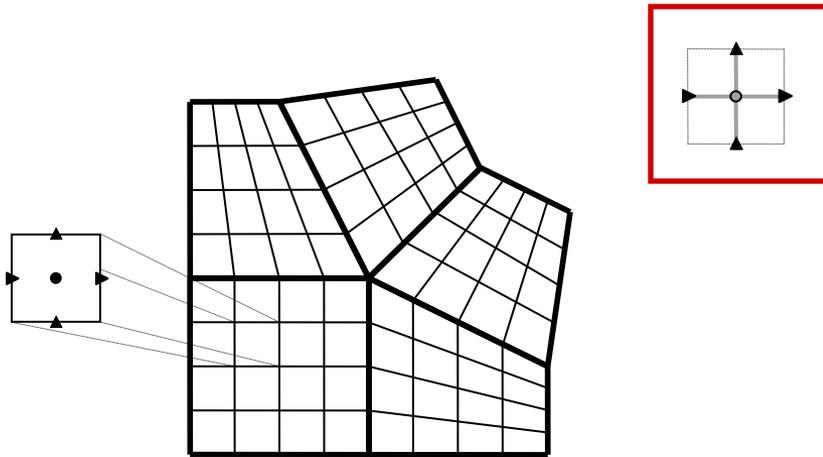


**Create the graph
object**

```
HYPRE_SStructGraph graph;
```

```
HYPRE_SStructGraphCreate(MPI_COMM_WORLD, grid, &graph);
```

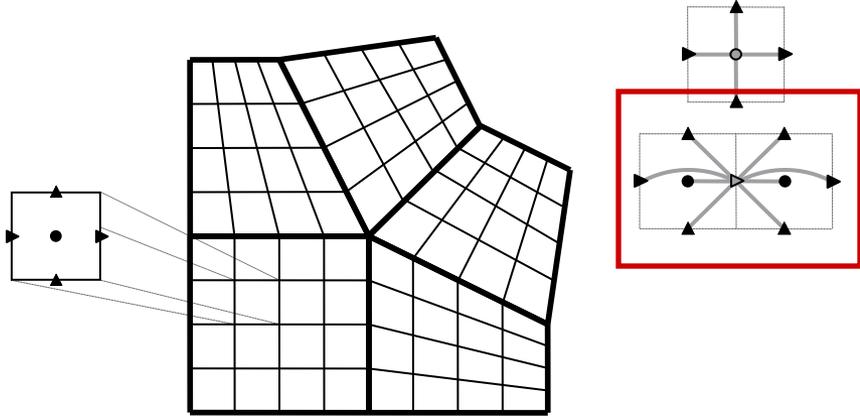
Block-structured grid example: Setting up the graph on process 3



**Set the cell-centered
stencil for each part**

```
int part;  
int var = 0;  
HYPRE_SStructStencil cell_stencil;  
  
HYPRE_SStructGraphSetStencil(graph, part, var, cell_stencil);
```

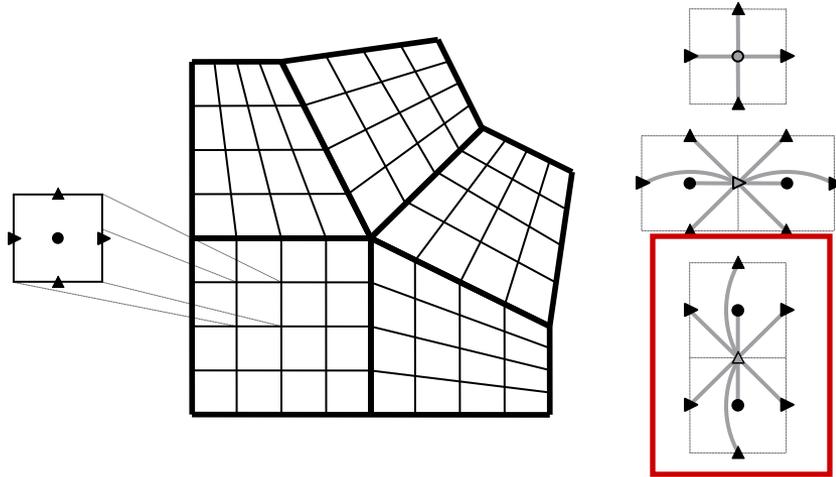
Block-structured grid example: Setting up the graph on process 3



**Set the x-face stencil
for each part**

```
int part;  
int var = 1;  
HYPRE_SStructStencil x_stencil;  
  
HYPRE_SStructGraphSetStencil(graph, part, var, x_stencil);
```

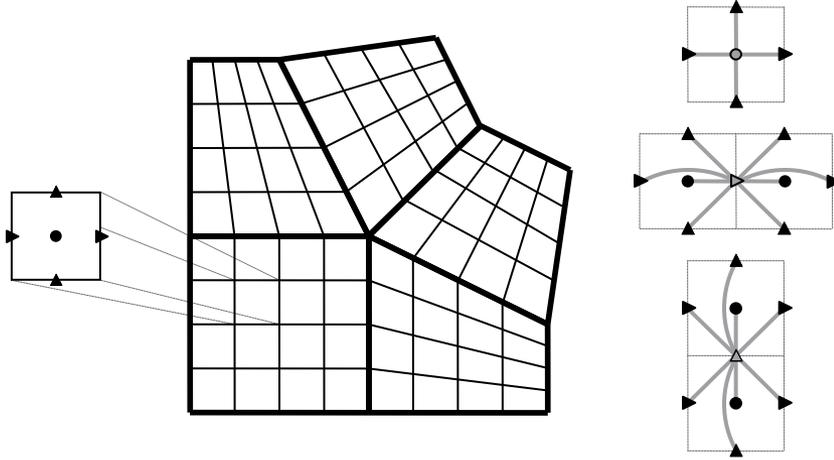
Block-structured grid example: Setting up the graph on process 3



**Set the y-face stencil
for each part**

```
int part;  
int var = 2;  
HYPRE_SStructStencil y_stencil;  
  
HYPRE_SStructGraphSetStencil(graph, part, var, y_stencil);
```

Block-structured grid example: Setting up the graph on process 3



Assemble the graph

```
/* No need to add non-stencil entries  
* with HYPRE_SStructGraphAddEntries() */  
HYPRE_SStructGraphAssemble(graph);
```

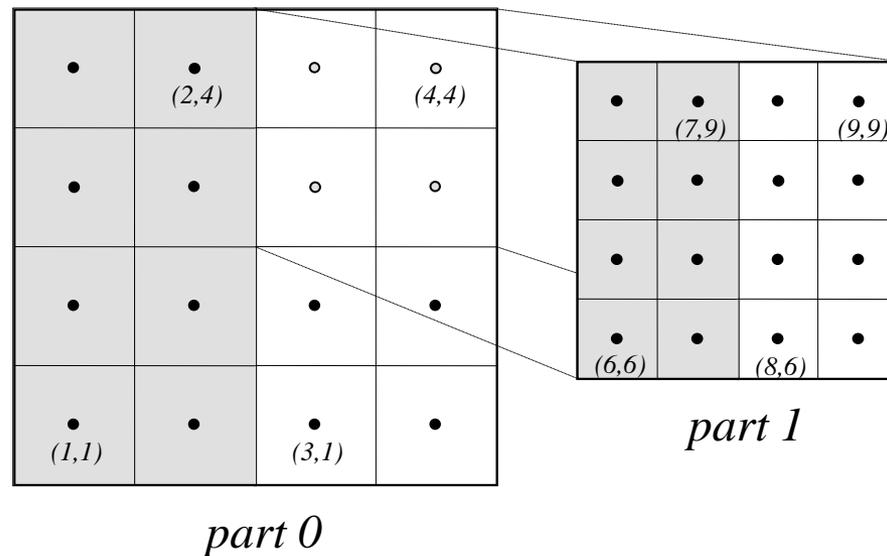
Block-structured grid example:

Setting up the matrix and vector

- The matrix and vector objects are constructed in a manner similar to the `Struct` interface
- Matrix coefficients are set with the routines
 - `HYPRE_SStructMatrixSetValues()`
 - `HYPRE_SStructMatrixAddToValues()`
- Vector values are set with similar routines
 - `HYPRE_SStructVectorSetValues()`
 - `HYPRE_SStructVectorAddToValues()`

Structured AMR example (SStruct)

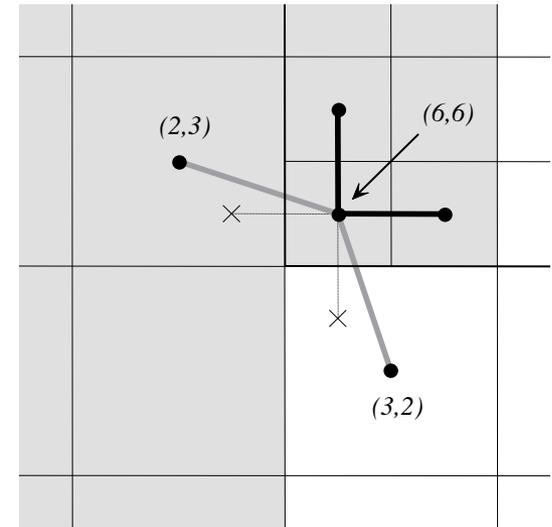
- Consider a simple cell-centered discretization of the Laplacian on the following structured AMR grid



- Each AMR grid level is defined as a separate part
- Assume 2 processes with shaded regions on process 0 and unshaded regions on process 1

Structured AMR example (SStruct)

- The **grid** is constructed using straightforward calls to the routines `HYPRE_SStructGridSetExtents()` and `HYPRE_SStructGridSetVariables()` as in the previous block-structured grid example
- The **graph** is constructed from a cell-centered stencil plus additional *non-stencil entries* at coarse-fine interfaces
- These non-stencil entries are set one variable at a time using `HYPRE_SStructGraphAddEntries()`



New finite element (FEM) style interface for SStruct as an alternative to stencils

- Beginning with *hypr* 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`**

Building different matrix/vector storage formats with the SStruct interface

- Efficient preconditioners often require specific matrix/vector storage schemes
- Between `Create()` and `Initialize()`, call:
`HYPRE_SStructMatrixSetObjectType(A, HYPRE_PARCSR);`
- After `Assemble()`, call:
`HYPRE_SStructMatrixGetObject(A, &parcsr_A);`
- Now, use the `ParCSR` matrix with compatible solvers such as `BoomerAMG` (algebraic multigrid)

Comments on SStruct interface

- The routine `HYPRE_SStructGridAddVariables()` allows additional variables to be added to individual cells, but is not yet implemented
- The routine `HYPRE_SStructGridSetNeighborPart()` now supports cell-centered, edge-centered, face-centered and nodal variable types
- FAC solver in *hypre* for AMR

Finite Element Interface (FEI)

- The FEI interface is designed for finite element discretizations on unstructured grids
- The interface supports C++ only
- See the following for detailed information on using it

R. L. Clay et al. An annotated reference guide to the Finite Element Interface (FEI) Specification, Version 1.0. Sandia National Laboratories, Livermore, CA, Technical Report SAND99-8229, 1999.

- There is a brief description in *hypr* user's manual and an example code

Linear-Algebraic System Interface (IJ)

- The IJ interface provides access to general sparse-matrix solvers, but not specialized solvers
- There are two basic steps involved:
 - set up the `Matrix`
 - set up the right-hand-side `Vector`



An example for the IJ interface: setting up the Matrix (on Proc 1)

$$\begin{array}{l} \text{rows} \end{array} \begin{array}{c} 4 \\ 5 \\ 6 \end{array} \begin{pmatrix} & \text{column indices} \\ & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \begin{array}{c} 4 \\ 5 \\ 6 \end{array} & & -1 & & -1 & 4 & -1 & & -1 & \\ & & & -1 & & -1 & 4 & -1 & & -1 \\ & & & & -1 & & -1 & 4 & -1 & \end{pmatrix}$$

**Create and initialize
matrix**

```
HYPRE_IJMatrix A;  
int ilower = 4, iupper = 6;  
int jlower = 4, jupper = 6;  
  
HYPRE_IJMatrixCreate(MPI_COMM_WORLD, ilower, iupper, jlower,  
jupper, &A);  
HYPRE_IJMatrixSetObjectType(A, HYPRE_PARCSR);  
  
HYPRE_IJMatrixInitialize(A);
```

An example for the IJ interface: setting up the Matrix (on Proc 1)

$$\begin{array}{l} \text{rows} \end{array} \begin{array}{c} 4 \\ 5 \\ 6 \end{array} \left(\begin{array}{cccccccc} & \text{column indices} & & & & & & & & \\ & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ & & -1 & & -1 & 4 & -1 & & -1 & \\ & & & -1 & & -1 & 4 & -1 & & -1 \\ & & & & -1 & & -1 & 4 & -1 & \end{array} \right)$$

**Set matrix
coefficients**

```
HYPRE_IJMatrix A;
int nrows = 3, ncols[3] = {5, 4, 3}, rows[3] = {4, 5, 6};
int cols[12] = {1,3,4,5,7, 2,4,5,8, 3,6,7};
double values[12] = {-1,-1,4,-1,-1, -1,-1,4,-1, -1,4,-1};

/* set matrix coefficients several rows at a time */
HYPRE_IJMatrixSetValues(A, nrows, ncols, rows, cols, values);
```

An example for the IJ interface: setting up the Matrix (on Proc 1)

$$\begin{array}{l} \text{rows} \end{array} \begin{array}{c} 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \text{column indices} \\ \left(\begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & -1 & & -1 & 4 & -1 & & -1 \\ & & & -1 & & -1 & 4 & -1 & \end{array} \right) \end{array}$$

**Assemble matrix
and retrieve object**

```
HYPRE_IJMatrix A;
```

```
HYPRE_IJMatrixAssemble(A);
```

```
HYPRE_ParCSRMatrix parcsr_A;
```

```
HYPRE_IJMatrixGetObject(A, &parcsr_A);
```

Current solver / preconditioner availability via *hypre*'s linear system interfaces

Data Layouts		Solvers	System Interfaces			
			Struct	SStruct	FEI	IJ
Structured	{	Jacobi	✓	✓		
		SMG	✓	✓		
		PFMG	✓	✓		
Semi-structured	{	Split		✓		
		SysPFMG		✓		
		FAC		✓		
		Maxwell		✓		
Sparse matrix	{	AMS		✓	✓	✓
		BoomerAMG		✓	✓	✓
		MLI		✓	✓	✓
		ParaSails		✓	✓	✓
		Euclid		✓	✓	✓
		PILUT		✓	✓	✓
		PCG	✓	✓	✓	✓
Matrix free	{	GMRES	✓	✓	✓	✓
		BiCGSTAB	✓	✓	✓	✓
		Hybrid	✓	✓	✓	✓

Setup and use of solvers is largely the same (see *Reference Manual for details*)

- Create the solver

```
HYPRE_SolverCreate(MPI_COMM_WORLD, &solver);
```

- Set parameters

```
HYPRE_SolverSetTol(solver, 1.0e-06);
```

- Prepare to solve the system

```
HYPRE_SolverSetup(solver, A, b, x);
```

- Solve the system

```
HYPRE_SolverSolve(solver, A, b, x);
```

- Get solution info out via system interface

```
HYPRE_StructVectorGetValues(struct_x, index,  
values);
```

- Destroy the solver

```
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);

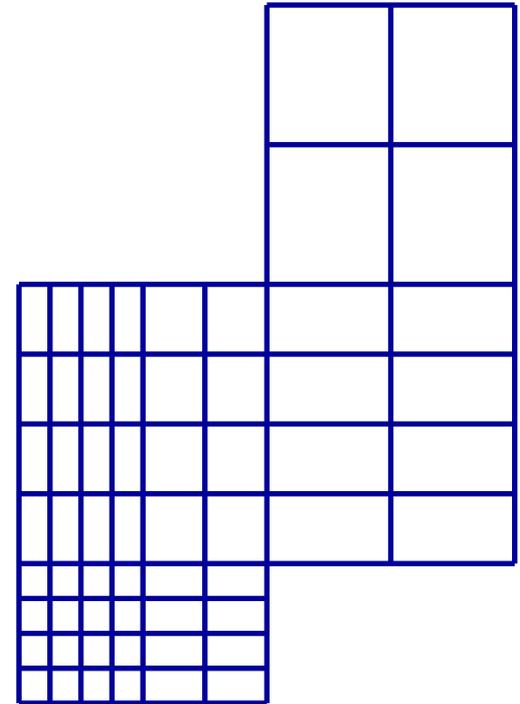
/* set preconditioner */
HYPRE_StructPCGSetPrecond(solver,
    HYPRE_StructSMGSolve, HYPRE_StructSMGSetup, precond);

HYPRE_StructPCGSetup(solver, A, b, x);
HYPRE_StructPCGSolve(solver, A, b, x);
```



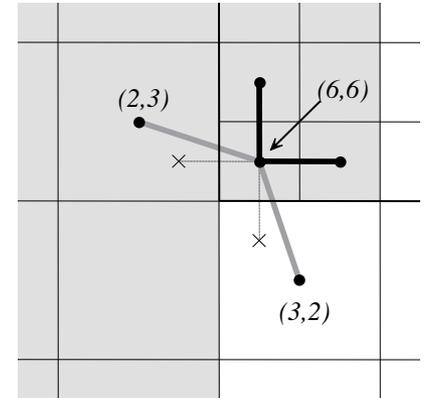
SMG and PFMG are semicoarsening multigrid methods for structured grids

- 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!**



FAC is an algebraic cell-centered fast adaptive composite grid solver in *hypre*

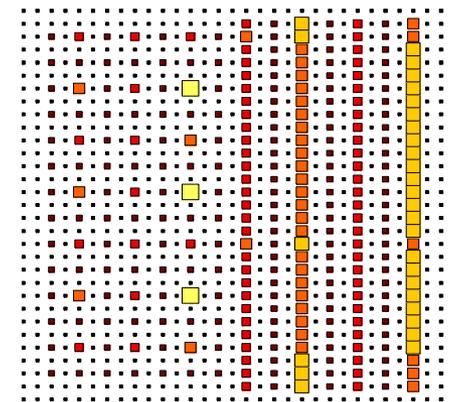
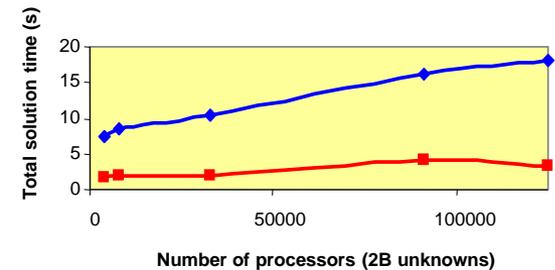
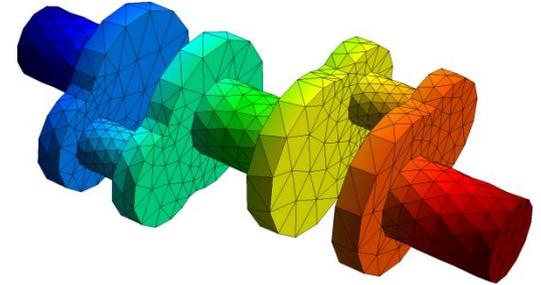
- Interface: `SStruct`
- Matrix Class: `SStruct`
- Requires only the composite matrix
 - no coarse underlying matrix needed
- Does not require nested AMR levels in the processor distribution, e.g., 3 levels on 2 procs
 - uses intra- and inter-level communication



- Designed for smooth-coefficient diffusion problems

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

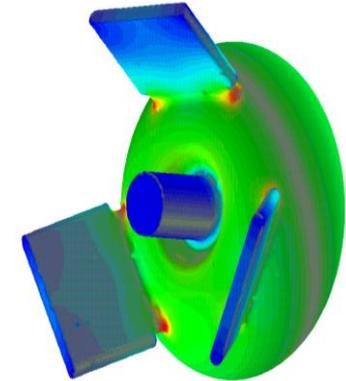


Maxwell is a definite Maxwell solver for (semi)-structured grids

- Interface: `SStruct`
- Matrix Class: `SStruct`
- Solves definite problems

$$\nabla \times \alpha \nabla \times E + \beta E = f, \quad \alpha > 0, \beta > 0$$

- Uses multiple coarsening, special relaxation, and a coupled hierarchy to resolve different error components
- Requires the linear system and a gradient matrix
- Only for edge finite element discretization



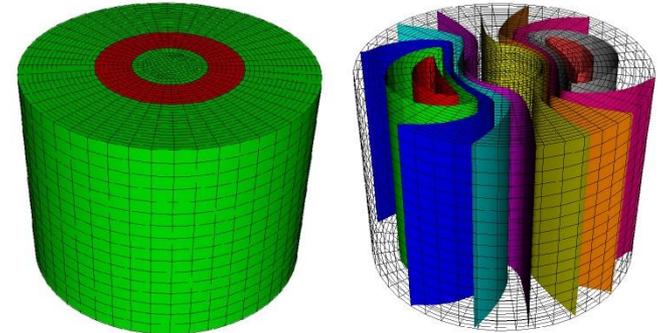
AMS is an auxiliary space Maxwell solver for unstructured grids

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

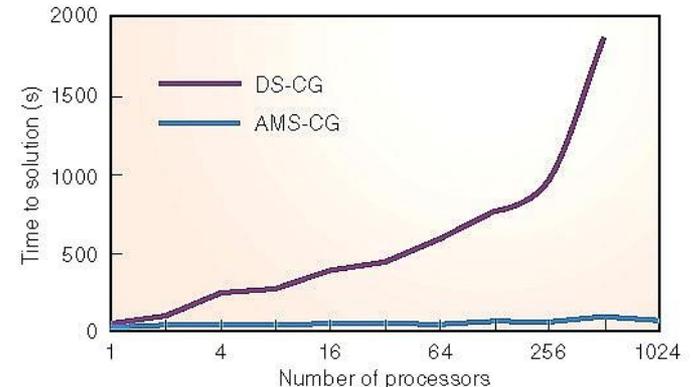
- Solves definite problems:

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

- Requires additional gradient matrix and mesh coordinates
- Variational form of Hiptmair-Xu
- Employs BoomerAMG
- Only for FE discretizations



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



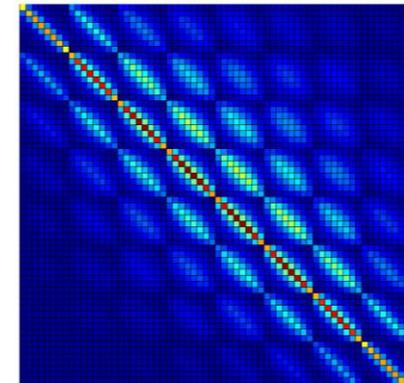
25x faster on 80M unknowns

ParaSAILS is an approximate inverse method for sparse linear systems

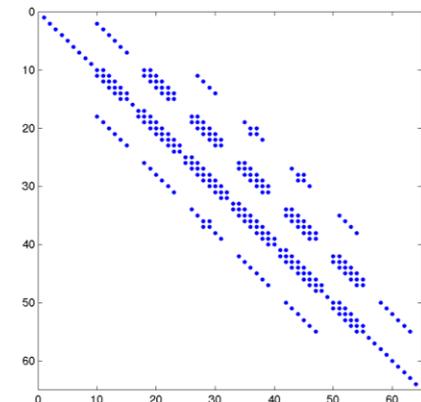
- Interface: `SStruct`, `FEI`, `IJ`
- Matrix Class: `ParCSR`

- 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

Exact inverse



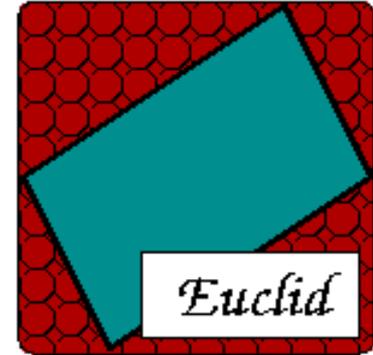
Approx inverse



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>

Getting the code

- To get the code, go to

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

- User's / Reference Manuals can be downloaded directly
- A short form must be filled out (just for our own records)

Building the library

- Usually, *hypr* can be built by typing `configure` followed by `make`
- Configure supports several options (for usage information, type '`configure --help`'):
 - '`configure --enable-debug`' - turn on debugging
 - '`configure --with-openmp`' - use openmp
 - '`configure --with-CFLAGS=...`' - set compiler flags
- **Release now includes example programs!**

Calling *hypre* from Fortran

- C code:

```
HYPRE_IJMatrix A;  
int           nvalues, row, *cols;  
double       *values;  
  
HYPRE_IJMatrixSetValues(A, nvalues, row, cols, values);
```

- Corresponding Fortran code:

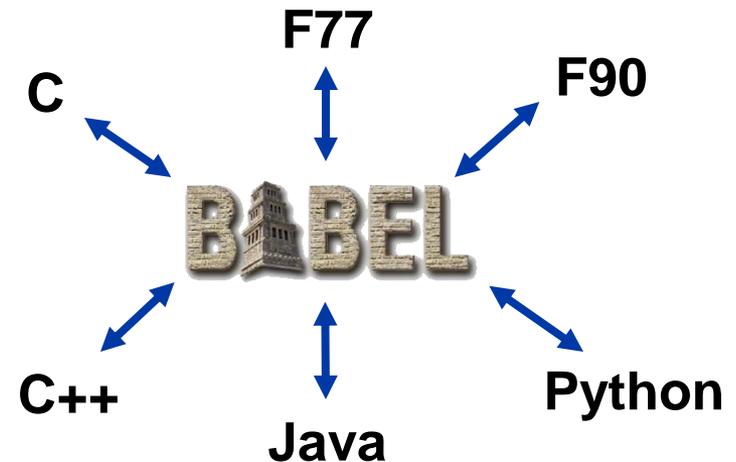
```
integer*8      A  
integer        nvalues, row, cols(MAX_NVALUES)  
double precision values(MAX_NVALUES)  
  
call HYPRE_IJMatrixSetValues(A, nvalues, row, cols, values)
```

The *hypre* library also features the Babel language interoperability tool

- Babel provides:
 - language interoperability
 - OO support

- Releases 1.11.0b and later use Babel for all major interface classes (except FEI)

- Currently supporting C, C++, Fortran, Python, and Java



More about the Babel interfaces

- Users download, build, and use *hypre* in virtually the same manner as before

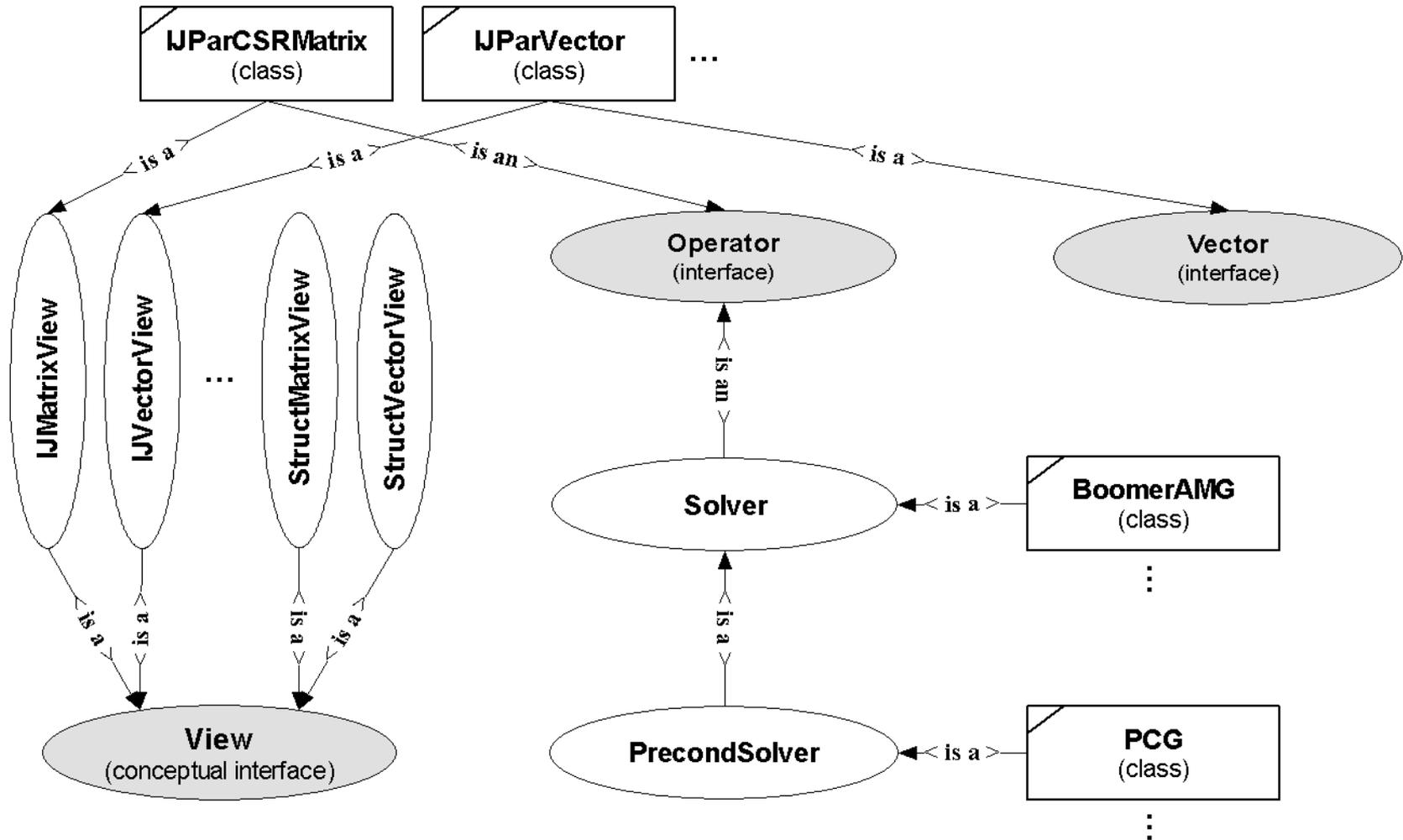
```
/* current interface */
```

```
MPI_COMM mpi_comm;  
HYPRE_IJMatrix ij_A;  
  
HYPRE_IJMatrixCreate(mpi_comm, i_lower, i_upper, j_lower,  
j_upper, &ij_A);  
HYPRE_IJMatrixSetObjectType(ij_A, HYPRE_PARCSR);
```

```
/* new babel interface */
```

```
bHYPRE_MPICommunicator mpi_comm;  
bHYPRE_IJParCSRMatrix parcsr_A;  
  
parcsr_A = bHYPRE_IJParCSRMatrix_Create(mpi_comm, i_lower,  
i_upper, j_lower, j_upper);
```

Central to hypre's object design is the use of multiple inheritance of interfaces



Reporting bugs, requesting features, general usage questions

- Send email to:

hypre-support@llnl.gov

- We use a tool called Roundup to automatically tag and track issues



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.

