The (Marginally) Complete Guide to

Hypre

What Hypre is:

- A library of numerical solvers and preconditioners
- Written in C, uses MPI
- Interfaces with our code as a series of black-box functions (data goes in, data comes out).
- Our hypre-calling code is contained within lib/hypreBEAR.f90
- The C-Fortran interfaces are found in the C libraries lib/extended_hypre_wrapper.c, extended_hypre_wrapper.h

How Hypre interfaces with AstroBEAR:

- ***ReadHypreData()***: Reads in hypre-specific data parameters from domain.data.

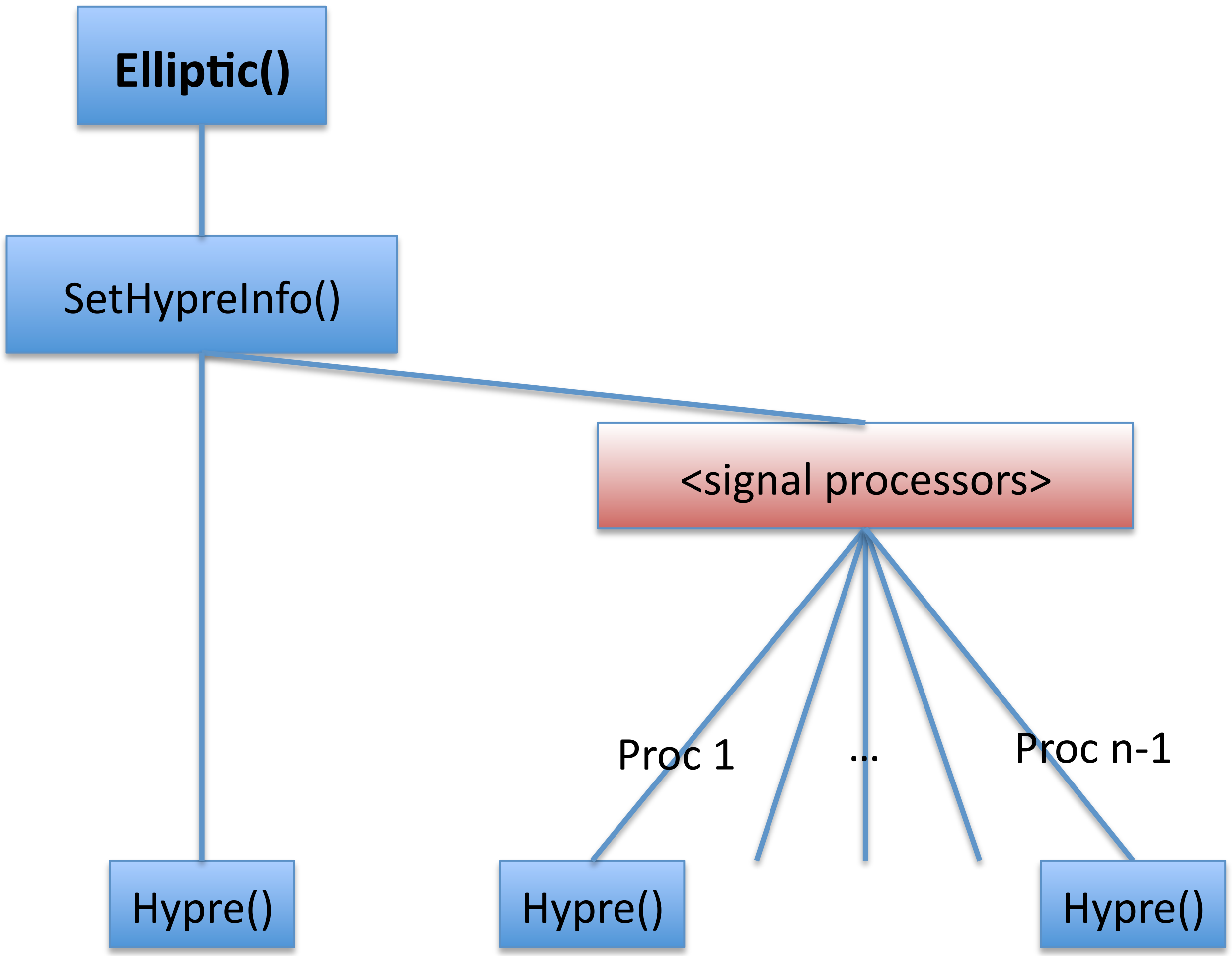
- ***Hypre_InitializeMatrixCoeffs()***: Initializes an array containing the weighted coefficients of a 1D equation's matrix entry.

- ***SetHypreInfo()***: Loads the settings of a hypre run into a HypreInfo data structure.

- ***HypreSize()***: Returns the size of a HypreInfo structure.

- ***Hypre()***: A single hypre "solve" is executed on each processor.

A higher-level view…



- An elliptic step is started in bearez.f90 by calling Elliptic().
- Elliptic() sets up the hypre step's parameters and signals each processor to call Hypre().
- Depending on the nature of the problem, multiple hypre steps can be called within Elliptic().
- Hypre handles its own MPI calls, so the elliptic solving is synchronized without AstroBEAR's involvement.

Using Hypre

- Hypre solves equation systems cast in the form of an Ax=b matrix problem.

- A is the coefficient matrix.
- x is the solution vector that is returned.
- b is the variable or input vector.

- Constructing these three structures constitutes the bulk of AstroBEAR's work in this process.

Important Hypre terms:

- ***Conceptual Interface:*** Hypre's term to describe the structure, nature and geometry of the problem domain. These interfaces can resemble a mesh, a matrix, or something considerably more complicated.

- ***Structured Interface:*** A conceptual interface where the problem domain resembles a regular mesh. The matrix is assembled using stencils, because the entire problem domain has a regular shape. Use this interface for fixed-grid problems.

- ***Semi-Structured Interface:*** A conceptual interface where problem domains are composed of several structured subdomains called *partitions*. These partitions are joined at the edges by user-specified mappings called *graph entries*. AMR uses the semi-structured interface, and each level is considered a partition.

- ***Solver:*** The numerical engine used to solve the system of equations. Hypre has many solver options; right now self-gravity only uses PCG and GMRES.

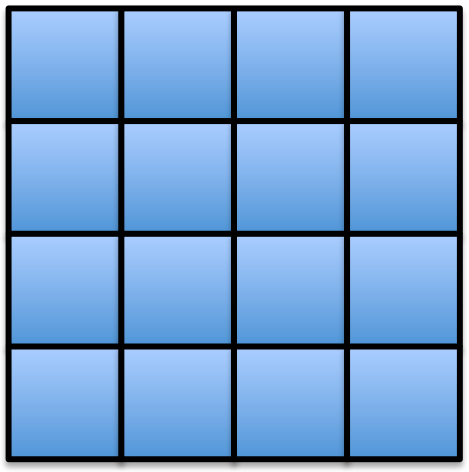
- ***Preconditioner:*** An operator applied to the equation matrix to improve the system's speed or its rate of convergence. We currently do not use a preconditioner in self-gravity.

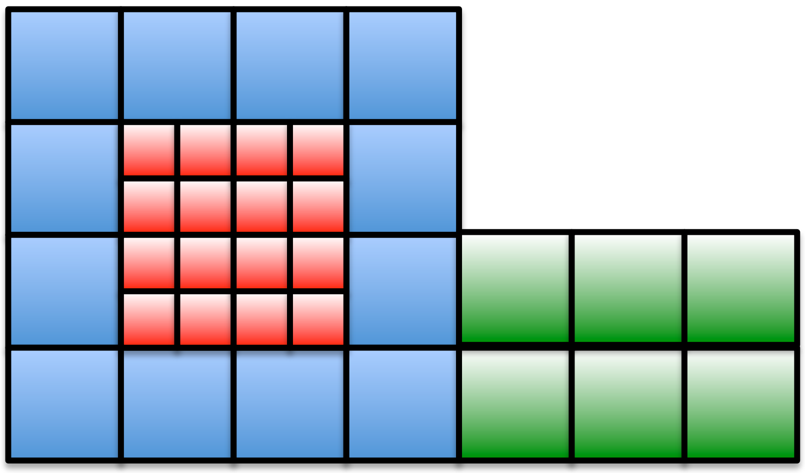
Rough Hypre algorithm

1. Create Grid (set grid extents).
2. Create Stencil (set up stencil entries).
3. Create Graphs (set up graph entries).
4. Create Matrix (set matrix coefficients).
5. Create Vectors (populate variable, solution vectors).
6. Create Solver (solve system).
7. Retrieve Solution.

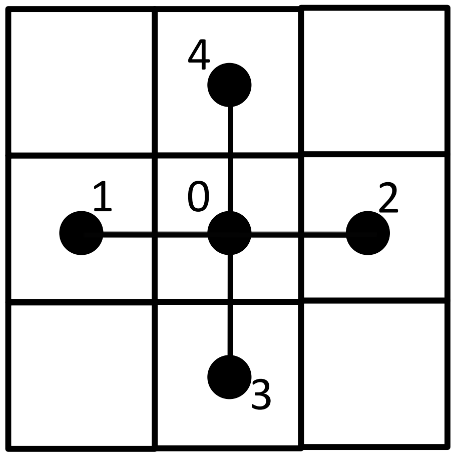
Hypre data structures

- ***Grid:*** Represents the problem domain. The dimensions of AstroBEAR grids are fed into this structure.



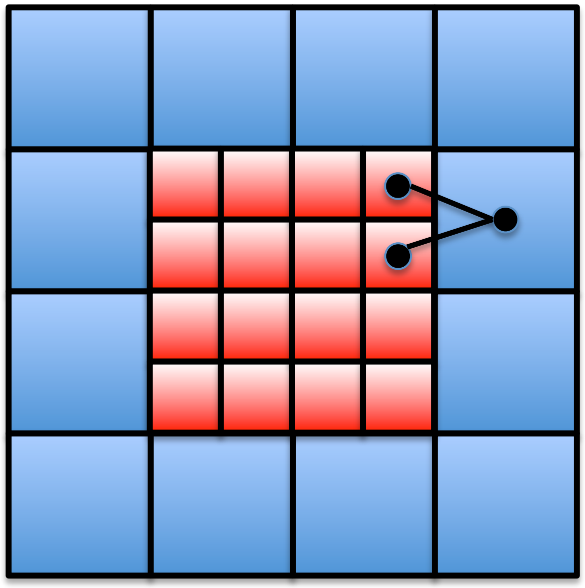


- ***Stencil:*** The arrangement of values in the matrix that hypre uses to solve the system. Stencil values are usually expressed in terms of steps from a cell.
- Stencils generally have a size 2*nDim + 1.



Index:	x-offset	y-offset
0	0	0
1	-1	0
2	1	0
3	0	-1
4	0	1

- ***Graph:*** Only used in a semi-structured interface. This is an explicit mapping between two different partitions.
  - Graph entries are unidirectional—a graph entry can map a child cell to its parent, but a separate entry is required to map a parent to its child.
  - The current algorithm only maps cells to their parents.



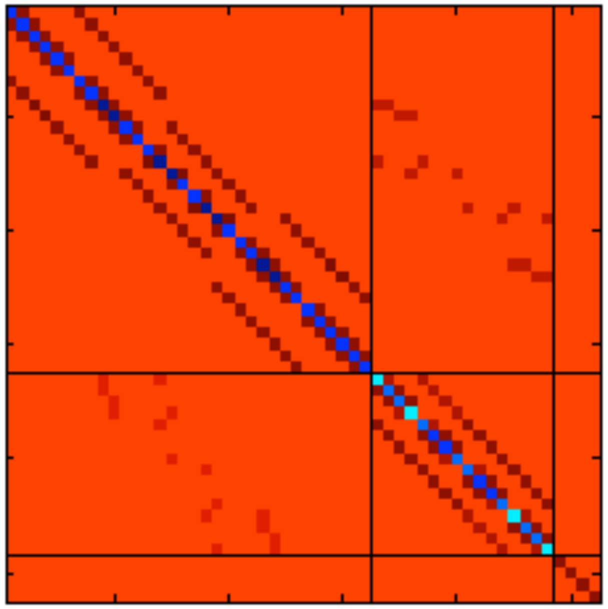
- ***Matrix:*** The coefficient matrix used by hypre to represent spatial derivatives.
  - In a structured interface, these matrix elements are determined only by stencils.
  - In a semi-structured interface, graph entries set the matrix values as well.
  - The matrix information is constructed and passed into hypre as a 1D array.
  - Each cell *n* has a number of matrix entries equal to *stencil_entries + graph_entries(n)*.
  - A three-dimensional *i * j * k* problem domain will have a *i * j * k* by *i * j * k* matrix.

Matrix: A self-gravity example

$$\nabla^2\Phi \Rightarrow \frac{\partial^2\Phi}{\partial x^2} + \frac{\partial^2\Phi}{\partial y^2} \Rightarrow \Phi_{i-1,j} + \Phi_{i+1,j} - 4\Phi_{i,j} + \Phi_{i,j-1} + \Phi_{i,j+1}$$

$$matrix(i,j) = \begin{bmatrix} -4 & 1 & 1 & 1 & 1 \end{bmatrix}$$

A more complicated example:



- ***Vector***: A 1D array that either feeds data into hypre or reads it out.
  - variableVector (b) is constructed by flattening data from the problem domain into a 1D array.
  - solutionVector (x) returns the results of hypre's solve attempts.
  - solutionVector is initially populated with a best-guess solution to speed up convergence.
  - In self-gravity, variable vector is initialized to $4\pi G\rho$ and solution vector returns Φ.

- ***Solver***: This is the component that will actually take all these inputs and return a solution.
  - Hypre has a lot of different solvers, but we currently only have a few implemented in AstroBEAR.
  - AstroBEAR has PCG solvers implemented on the structured interface, and PCG and GMRES solvers for the unstructured interface.

- Solvers tend to have the same core sequence of steps:
  1. Create solver.
  2. Set tolerance.
  3. Load A, b, x data into solver (the previously-discussed data structures revolved around constructing this information).
  4. Solve system.
  5. Retrieve solved data.

Tolerance and Convergence

- Hypre’s solvers iterate over the system until a certain number of iterations complete or until the system convergence.
- Convergence criteria are set as part of the solver setup. The most common criterion is setting the *tolerance* of the relative residual.
- Once the relative residual is at or below the tolerance, then the system is considered converged.
- The PCG solver is not guaranteed to converge, but is faster than the GMRES solver.
- The GMRES solver is guaranteed to converge, but doing so may require more time and memory resources than is practical.