hypre Reference Manual

— Version 1.10.0b —

Contents

1

extern **Struct System Interface**

Names

This interface represents a structured-grid conceptual view of a linear system.

Author: Robert D. Falgout

1.1

Struct Grids

Names

typedef struct hypre_StructGrid_struct* **HYPRE_StructGrid**

A grid object is constructed out of several "boxes", defined on a global abstract index space

int
HYPRE_StructGridCreate (MPI_Comm comm, int ndim, HYPRE_StructGrid *grid)

Create an `ndim`*-dimensional grid object*

1.1.1 int
HYPRE_StructGridDestroy (HYPRE_StructGrid grid)

Destroy a grid object 5

int
HYPRE_StructGridSetExtents (HYPRE_StructGrid grid, int *ilower, int *iupper)

Set the extents for a box on the grid

int

HYPRE_StructGridAssemble (HYPRE_StructGrid grid)
Finalize the construction of the grid before using

int
HYPRE_StructGridSetPeriodic (HYPRE_StructGrid grid, int *periodic)
(Optional) Set periodic

int
HYPRE_StructGridSetNumGhost (HYPRE_StructGrid grid, int *num_ghost)
(Optional) Set the ghost layer in the grid object

1.1.1

int **HYPRE_StructGridDestroy** (HYPRE_StructGrid grid)

Destroy a grid object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

1.2

Struct Stencils

Names

typedef struct hypre_StructStencil_struct* **HYPRE_StructStencil**
The stencil object

int
HYPRE_StructStencilCreate (int ndim, int size, HYPRE_StructStencil *stencil)
Create a stencil object for the specified number of spatial dimensions and stencil entries

int
HYPRE_StructStencilDestroy (HYPRE_StructStencil stencil)
Destroy a stencil object

1.2.1

int
HYPRE_StructStencilSetElement (HYPRE_StructStencil stencil, int entry, int *offset)

Set a stencil entry.

NOTE: The name of this routine will eventually be changed to `HYPRE_StructStencilSetEntry`.

1.3

Struct Matrices

Names

typedef struct hypre_StructMatrix_struct* **HYPRE_StructMatrix**
The matrix object

int
HYPRE_StructMatrixCreate (MPI_Comm comm, HYPRE_StructGrid grid, HYPRE_StructStencil stencil, HYPRE_StructMatrix *matrix)
Create a matrix object

int
HYPRE_StructMatrixDestroy (HYPRE_StructMatrix matrix)
Destroy a matrix object

int
HYPRE_StructMatrixInitialize (HYPRE_StructMatrix matrix)
Prepare a matrix object for setting coefficient values

int
HYPRE_StructMatrixSetValues (HYPRE_StructMatrix matrix, int *index, int nentries, int *entries, double *values)
Set matrix coefficients index by index

int
HYPRE_StructMatrixSetBoxValues (HYPRE_StructMatrix matrix, int *ilower, int *iupper, int nentries, int *entries, double *values)
Set matrix coefficients a box at a time

int

1.3.1

int
HYPRE_StructMatrixSetSymmetric (HYPRE_StructMatrix matrix, int symmetric)

(Optional) Define symmetry properties of the matrix. By default, matrices are assumed to be nonsymmetric. Significant storage savings can be made if the matrix is symmetric.

1.3.2

int
HYPRE_StructMatrixSetConstantEntries (HYPRE_StructMatrix matrix, int nentries, int *entries)

Specifiy which stencil entries are constant over the grid. Presently supported: - no entries constant (this function need not be called) - all entries constant - all but the diagonal entry constant

1.3.3

int
HYPRE_StructMatrixPrint (const char *filename, HYPRE_StructMatrix matrix, int all)

Print the matrix to file. This is mainly for debugging purposes.

1.4 Struct Vectors

Names

typedef struct hypre_StructVector_struct* **HYPRE_StructVector**
The vector object

int
HYPRE_StructVectorCreate (MPI_Comm comm, HYPRE_StructGrid grid, HYPRE_StructVector *vector)
Create a vector object

int

HYPRE_StructVectorDestroy (HYPRE_StructVector vector)
Destroy a vector object

int
HYPRE_StructVectorInitialize (HYPRE_StructVector vector)
Prepare a vector object for setting coefficient values

int
HYPRE_StructVectorSetValues (HYPRE_StructVector vector, int *index, double value)
Set vector coefficients index by index

int
HYPRE_StructVectorSetBoxValues (HYPRE_StructVector vector, int *ilower, int *iupper, double *values)
Set vector coefficients a box at a time

int
HYPRE_StructVectorAddToValues (HYPRE_StructVector vector, int *index, double value)
Set vector coefficients index by index

int
HYPRE_StructVectorAddToBoxValues (HYPRE_StructVector vector, int *ilower, int *iupper, double *values)
Set vector coefficients a box at a time

int
HYPRE_StructVectorAssemble (HYPRE_StructVector vector)
Finalize the construction of the vector before using

int
HYPRE_StructVectorGetValues (HYPRE_StructVector vector, int *index, double *value)
Get vector coefficients index by index

int
HYPRE_StructVectorGetBoxValues (HYPRE_StructVector vector, int *ilower, int *iupper, double *values)
Get vector coefficients a box at a time

1.4.1

int
HYPRE_StructVectorPrint (const char *filename, HYPRE_StructVector vector, int all)

Print the vector to file. This is mainly for debugging purposes.

2

extern **SStruct System Interface**

Names

This interface represents a semi-structured-grid conceptual view of a linear system.

Author: Robert D. Falgout

2.1

SStruct Grids

Names

2.1.1

#define **HYPRE_SStructGrid**

A grid object is constructed out of several structured "parts" and an optional unstructured "part". Each structured part has its own abstract index space.

2.1.2

#define **HYPRE_SStructVariable**

An enumerated type that supports cell centered, node centered, face centered, and edge centered variables. Face centered variables are split into x-face, y-face, and z-face variables, and edge centered variables are split into x-edge, y-edge, and z-edge variables. The edge centered variable types are only used in 3D. In 2D, edge centered variables are handled by the face centered types.

Variables are referenced relative to an abstract (cell centered) index in the following way:

- cell centered variables are aligned with the index;
- node centered variables are aligned with the cell corner at relative index (1/2, 1/2, 1/2);
- x-face, y-face, and z-face centered variables are aligned with the faces at relative indexes (1/2, 0, 0), (0, 1/2, 0), and (0, 0, 1/2), respectively;
- x-edge, y-edge, and z-edge centered variables are aligned with the edges at relative indexes (0, 1/2, 1/2), (1/2, 0, 1/2), and (1/2, 1/2, 0), respectively.

The supported identifiers are:

- `HYPRE_SSTRUCT_VARIABLE_CELL`
- `HYPRE_SSTRUCT_VARIABLE_NODE`
- `HYPRE_SSTRUCT_VARIABLE_XFACE`
- `HYPRE_SSTRUCT_VARIABLE_YFACE`
- `HYPRE_SSTRUCT_VARIABLE_ZFACE`
- `HYPRE_SSTRUCT_VARIABLE_XEDGE`
- `HYPRE_SSTRUCT_VARIABLE_YEDGE`
- `HYPRE_SSTRUCT_VARIABLE_ZEDGE`

NOTE: Although variables are referenced relative to a unique abstract cell-centered index, some variables are associated with multiple grid cells. For example, node centered variables in 3D are associated with 8 cells (away from boundaries). Although grid cells are distributed uniquely to different processes, variables may be owned by multiple processes because they may be associated with multiple cells.

2.1.3

int **HYPRE_SStructGridDestroy** (HYPRE_SStructGrid grid)

Destroy a grid object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

2.1.4

int
HYPRE_SStructGridAddVariables (HYPRE_SStructGrid grid, int part, int *index, int nvars, HYPRE_SStructVariable *vartypes)

Describe additional variables that live at a particular index. These variables are appended to the array of variables set in HYPRE_SStructGridSetVariables (→ *page 12*), and are referenced as such.

2.1.5

int
HYPRE_SStructGridSetNeighborBox (HYPRE_SStructGrid grid, int part, int *ilower, int *iupper, int nbor_part, int *nbor_ilower, int *nbor_iupper, int *index_map)

Describe how regions just outside of a part relate to other parts. This is done a box at a time.

The indexes `ilower` and `iupper` map directly to the indexes `nbor_ilower` and `nbor_iupper`. Although, it is required that indexes increase from `ilower` to `iupper`, indexes may increase and/or decrease from `nbor_ilower` to `nbor_iupper`.

The `index_map` describes the mapping of indexes 0, 1, and 2 on part `part` to the corresponding indexes on part `nbor_part`. For example, triple (1, 2, 0) means that indexes 0, 1, and 2 on part `part` map to indexes 1, 2, and 0 on part `nbor_part`, respectively.

NOTE: All parts related to each other via this routine must have an identical list of variables and variable types. For example, if part 0 has only two variables on it, a cell centered variable and a node centered variable, and we declare part 1 to be a neighbor of part 0, then part 1 must also have only two variables on it, and they must be of type cell and node.

2.1.6

int
HYPRE_SStructGridAddUnstructuredPart (HYPRE_SStructGrid grid, int ilower, int iupper)

Add an unstructured part to the grid. The variables in the unstructured part of the grid are referenced by a global rank between 0 and the total number of unstructured variables minus one. Each process owns some unique consecutive range of variables, defined by `ilower` and `iupper`.

NOTE: This is just a placeholder. This part of the interface is not finished.

2.2 SStruct Stencils

Names

typedef struct hypre_SStructStencil_struct* **HYPRE_SStructStencil**
The stencil object

int
HYPRE_SStructStencilCreate (int ndim, int size, HYPRE_SStructStencil *stencil)
Create a stencil object for the specified number of spatial dimensions and stencil entries

int
HYPRE_SStructStencilDestroy (HYPRE_SStructStencil stencil)
Destroy a stencil object

int
HYPRE_SStructStencilSetEntry (HYPRE_SStructStencil stencil, int entry, int *offset, int var)
Set a stencil entry

2.3

SStruct Graphs

Names

typedef struct hypre_SStructGraph_struct* **HYPRE_SStructGraph**
The graph object is used to describe the nonzero structure of a matrix

int
HYPRE_SStructGraphCreate (MPI_Comm comm,
HYPRE_SStructGrid grid,
HYPRE_SStructGraph *graph)
Create a graph object

int
HYPRE_SStructGraphDestroy (HYPRE_SStructGraph graph)
Destroy a graph object

int
HYPRE_SStructGraphSetStencil (HYPRE_SStructGraph graph, int part,
int var, HYPRE_SStructStencil stencil)
Set the stencil for a variable on a structured part of the grid

int
HYPRE_SStructGraphAssemble (HYPRE_SStructGraph graph)
Finalize the construction of the graph before using

2.3.1

int **HYPRE_SStructGraphAddEntries** (HYPRE_SStructGraph graph, int part, int *index, int var, int to_part, int *to_index, int to_var)

Add a non-stencil graph entry at a particular index. This graph entry is appended to the existing graph entries, and is referenced as such.

NOTE: Users are required to set graph entries on all processes that own the associated variables. This means that some data will be multiply defined.

2.3.2

int
HYPRE_SStructGraphSetObjectType (HYPRE_SStructGraph graph, int type)

It is used before AddEntries and Assemble to compute the right ranks in the graph. Currently, `type` can be either `HYPRE_SSTRUCT` (the default) or `HYPRE_PARCSR`.

2.4

SStruct Matrices

Names

typedef struct hypre_SStructMatrix_struct* **HYPRE_SStructMatrix**
The matrix object

int
HYPRE_SStructMatrixCreate (MPI_Comm comm, HYPRE_SStructGraph graph, HYPRE_SStructMatrix *matrix)
Create a matrix object

int
HYPRE_SStructMatrixDestroy (HYPRE_SStructMatrix matrix)
Destroy a matrix object

int
HYPRE_SStructMatrixInitialize (HYPRE_SStructMatrix matrix)
Prepare a matrix object for setting coefficient values

2.4.1

int
HYPRE_SStructMatrixSetValues (HYPRE_SStructMatrix matrix, int part, int *index, int var, int nentries, int *entries, double *values)

Set matrix coefficients index by index.

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

NOTE: The entries in this routine must all be of the same type: either stencil or non-stencil, but not both. Also, if they are stencil entries, they must all represent couplings to the same variable type (there are no such restrictions for non-stencil entries).

If the matrix is complex, then `values` consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: HYPRE_SStructMatrixSetComplex (→ *page 18*)

2.4.2

int
HYPRE_SStructMatrixSetBoxValues (HYPRE_SStructMatrix matrix, int part, int *ilower, int *iupper, int var, int nentries, int *entries, double *values)

Set matrix coefficients a box at a time.

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

NOTE: The entries in this routine must all be of the same type: either stencil or non-stencil, but not both. Also, if they are stencil entries, they must all represent couplings to the same variable type (there are no such restrictions for non-stencil entries).

If the matrix is complex, then `values` consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: HYPRE_SStructMatrixSetComplex (→ *page 18*)

2.4.3

int
HYPRE_SStructMatrixAddToValues (HYPRE_SStructMatrix matrix, int part, int *index, int var, int nentries, int *entries, double *values)

Add to matrix coefficients index by index.

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

NOTE: The entries in this routine must all be of the same type: either stencil or non-stencil, but not both. Also, if they are stencil entries, they must all represent couplings to the same variable type.

If the matrix is complex, then `values` consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: HYPRE_SStructMatrixSetComplex (→ *page 18*)

2.4.4

int
HYPRE_SStructMatrixAddToBoxValues (HYPRE_SStructMatrix matrix, int part, int *ilower, int *iupper, int var, int nentries, int *entries, double *values)

Add to matrix coefficients a box at a time.

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

NOTE: The entries in this routine must all be of stencil type. Also, they must all represent couplings to the same variable type.

If the matrix is complex, then `values` consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: HYPRE_SStructMatrixSetComplex (→ *page 18*)

2.4.5

int
HYPRE_SStructMatrixSetSymmetric (HYPRE_SStructMatrix matrix, int part, int var, int to_var, int symmetric)

Define symmetry properties for the stencil entries in the matrix. The boolean argument `symmetric` is applied to stencil entries on part `part` that couple variable `var` to variable `to_var`. A value of -1 may be used for `part`, `var`, or `to_var` to specify "all". For example, if `part` and `to_var` are set to -1, then the boolean is applied to stencil entries on all parts that couple variable `var` to all other variables.

By default, matrices are assumed to be nonsymmetric. Significant storage savings can be made if the matrix is symmetric.

2.4.6

int
HYPRE_SStructMatrixSetObjectType (HYPRE_SStructMatrix matrix, int type)

Set the storage type of the matrix object to be constructed. Currently, `type` can be either `HYPRE_SSTRUCT` (the default), `HYPRE_STRUCT`, or `HYPRE_PARCSR`.

See Also: HYPRE_SStructMatrixGetObject (→2.4.7, *page 21*)

2.4.7

int
HYPRE_SStructMatrixGetObject (HYPRE_SStructMatrix matrix, void **object)

Get a reference to the constructed matrix object.

See Also: HYPRE_SStructMatrixSetObjectType (→2.4.6, *page 21*)

2.4.8

int
HYPRE_SStructMatrixPrint (const char *filename, HYPRE_SStructMatrix matrix, int all)

Print the matrix to file. This is mainly for debugging purposes.

2.5 SStruct Vectors

Names

typedef struct hypre_SStructVector_struct* **HYPRE_SStructVector**
The vector object

int
HYPRE_SStructVectorCreate (MPI_Comm comm,
HYPRE_SStructGrid grid,
HYPRE_SStructVector *vector)
Create a vector object

int
HYPRE_SStructVectorDestroy (HYPRE_SStructVector vector)
Destroy a vector object

int
HYPRE_SStructVectorInitialize (HYPRE_SStructVector vector)
Prepare a vector object for setting coefficient values

2.5.1

int
HYPRE_SStructVectorSetValues (HYPRE_SStructVector vector, int part, int *index, int var, double *value)

Set vector coefficients index by index.

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

If the vector is complex, then `value` consists of a pair of doubles representing the real and imaginary parts of the complex value.

See Also: HYPRE_SStructVectorSetComplex (→ *page 23*)

2.5.2

int
HYPRE_SStructVectorSetBoxValues (HYPRE_SStructVector vector, int part, int *ilower, int *iupper, int var, double *values)

Set vector coefficients a box at a time.

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

If the vector is complex, then `values` consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: HYPRE_SStructVectorSetComplex (→ *page 23*)

2.5.3

int
HYPRE_SStructVectorAddToValues (HYPRE_SStructVector vector, int part, int *index, int var, double *value)

Set vector coefficients index by index.

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

If the vector is complex, then `value` consists of a pair of doubles representing the real and imaginary parts of the complex value.

See Also: HYPRE_SStructVectorSetComplex (→ *page 23*)

2.5.4

int
HYPRE_SStructVectorAddToBoxValues (HYPRE_SStructVector vector, int part, int *ilower, int *iupper, int var, double *values)

Set vector coefficients a box at a time.

NOTE: Users are required to set values on all processes that own the associated variables. This means that some data will be multiply defined.

If the vector is complex, then `values` consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: HYPRE_SStructVectorSetComplex (→ *page 23*)

2.5.5

int
HYPRE_SStructVectorGetValues (HYPRE_SStructVector vector, int part, int *index, int var, double *value)

Get vector coefficients index by index.

NOTE: Users may only get values on processes that own the associated variables.

If the vector is complex, then `value` consists of a pair of doubles representing the real and imaginary parts of the complex value.

See Also: HYPRE_SStructVectorSetComplex (→ *page 23*)

2.5.6

int
HYPRE_SStructVectorGetBoxValues (HYPRE_SStructVector vector, int part, int *ilower, int *iupper, int var, double *values)

Get vector coefficients a box at a time.

NOTE: Users may only get values on processes that own the associated variables.

If the vector is complex, then `values` consists of pairs of doubles representing the real and imaginary parts of each complex value.

See Also: HYPRE_SStructVectorSetComplex (→ *page 23*)

2.5.7

int
HYPRE_SStructVectorSetObjectType (HYPRE_SStructVector vector, int type)

Set the storage type of the vector object to be constructed. Currently, `type` can be either `HYPRE_SSTRUCT` (the default) or `HYPRE_PARCSR`.

See Also: HYPRE_SStructVectorGetObject (→2.5.8, *page 26*)

2.5.8

int
HYPRE_SStructVectorGetObject (HYPRE_SStructVector vector, void **object)

Get a reference to the constructed vector object.

See Also: HYPRE_SStructVectorSetObjectType (→2.5.7, *page 26*)

2.5.9

int
HYPRE_SStructVectorPrint (const char *filename, HYPRE_SStructVector vector, int all)

Print the vector to file. This is mainly for debugging purposes.

3 extern IJ System Interface

Names

This interface represents a linear-algebraic conceptual view of a linear system. The 'I' and 'J' in the name are meant to be mnemonic for the traditional matrix notation A(I,J).

3.1 IJ Matrices

Names

3.1.1

int
HYPRE_IJMatrixCreate (MPI_Comm comm, int ilower, int iupper, int jlower, int jupper, HYPRE_IJMatrix *matrix)

Create a matrix object. Each process owns some unique consecutive range of rows, indicated by the global row indices `ilower` and `iupper`. The row data is required to be such that the value of `ilower` on any process p be exactly one more than the value of `iupper` on process $p-1$. Note that the first row of the global matrix may start with any integer value. In particular, one may use zero- or one-based indexing.

For square matrices, `jlower` and `jupper` typically should match `ilower` and `iupper`, respectively. For rectangular matrices, `jlower` and `jupper` should define a partitioning of the columns. This partitioning must be used for any vector v that will be used in matrix-vector products with the rectangular matrix. The matrix data structure may use `jlower` and `jupper` to store the diagonal blocks (rectangular in general) of the matrix separately from the rest of the matrix.

Collective.

3.1.2

int **HYPRE_IJMatrixDestroy** (HYPRE_IJMatrix matrix)

Destroy a matrix object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

3.1.3

int **HYPRE_IJMatrixInitialize** (HYPRE_IJMatrix matrix)

Prepare a matrix object for setting coefficient values. This routine will also re-initialize an already assembled matrix, allowing users to modify coefficient values.

3.1.4

int
HYPRE_IJMatrixSetValues (HYPRE_IJMatrix matrix, int nrows, int *ncols, const int *rows, const int *cols, const double *values)

Sets values for `nrows` rows or partial rows of the matrix. The arrays `ncols` and `rows` are of dimension `nrows` and contain the number of columns in each row and the row indices, respectively. The array `cols` contains the column indices for each of the `rows`, and is ordered by rows. The data in the `values` array corresponds directly to the column entries in `cols`. Erases any previous values at the specified locations and replaces them with new ones, or, if there was no value there before, inserts a new one.

Not collective.

3.1.5

int
HYPRE_IJMatrixAddToValues (HYPRE_IJMatrix matrix, int nrows, int *ncols, const int *rows, const int *cols, const double *values)

Adds to values for `nrows` rows or partial rows of the matrix. Usage details are analogous to HYPRE_IJMatrixSetValues (→3.1.4, *page 31*). Adds to any previous values at the specified locations, or, if there was no value there before, inserts a new one.

Not collective.

3.1.6

int
HYPRE_IJMatrixGetValues (HYPRE_IJMatrix matrix, int nrows, int *ncols, int *rows, int *cols, double *values)

Gets values for `nrows` rows or partial rows of the matrix. Usage details are analogous to HYPRE_IJMatrixSetValues (→3.1.4, *page 31*).

3.1.7

int **HYPRE_IJMatrixSetObjectType** (HYPRE_IJMatrix matrix, int type)

Set the storage type of the matrix object to be constructed. Currently, `type` can only be `HYPRE_PARCSR`.

Not collective, but must be the same on all processes.

See Also: HYPRE_IJMatrixGetObject (→3.1.8, *page 32*)

3.1.8

int **HYPRE_IJMatrixGetObject** (HYPRE_IJMatrix matrix, void **object)

Get a reference to the constructed matrix object.

See Also: HYPRE_IJMatrixSetObjectType (→3.1.7, *page 32*)

3.1.9

int **HYPRE_IJMatrixSetRowSizes** (HYPRE_IJMatrix matrix, const int *sizes)

(Optional) Set the max number of nonzeros to expect in each row. The array `sizes` contains estimated sizes for each row on this process. This call can significantly improve the efficiency of matrix construction, and should always be utilized if possible.

Not collective.

3.1.10

int
HYPRE_IJMatrixSetDiagOffdSizes (HYPRE_IJMatrix matrix, const int *diag_sizes, const int *offdiag_sizes)

(Optional) Set the max number of nonzeros to expect in each row of the diagonal and off-diagonal blocks. The diagonal block is the submatrix whose column numbers correspond to rows owned by this process, and the off-diagonal block is everything else. The arrays `diag_sizes` and `offdiag_sizes` contain estimated sizes for each row of the diagonal and off-diagonal blocks, respectively. This routine can significantly improve the efficiency of matrix construction, and should always be utilized if possible.

Not collective.

3.1.11

int
HYPRE_IJMatrixSetMaxOffProcElmts (HYPRE_IJMatrix matrix, int max_off_proc_elmts)

(Optional) Sets the maximum number of elements that are expected to be set (or added) on other processors from this processor This routine can significantly improve the efficiency of matrix construction, and should always be utilized if possible.

Not collective.

3.1.12

int
HYPRE_IJMatrixRead (const char *filename, MPI_Comm comm, int type, HYPRE_IJMatrix *matrix)

Read the matrix from file. This is mainly for debugging purposes.

3.1.13

int **HYPRE_IJMatrixPrint** (HYPRE_IJMatrix matrix, const char *filename)

Print the matrix to file. This is mainly for debugging purposes.

3.2 IJ Vectors

Names

3.2.1

int
HYPRE_IJVectorCreate (MPI_Comm comm, int jlower, int jupper,
HYPRE_IJVector *vector)

Create a vector object. Each process owns some unique consecutive range of vector unknowns, indicated by the global indices `jlower` and `jupper`. The data is required to be such that the value of `jlower` on any process p be exactly one more than the value of `jupper` on process $p-1$. Note that the first index of the global vector may start with any integer value. In particular, one may use zero- or one-based indexing.

Collective.

3.2.2

int **HYPRE_IJVectorDestroy** (HYPRE_IJVector vector)

Destroy a vector object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

3.2.3

int **HYPRE_IJVectorInitialize** (HYPRE_IJVector vector)

Prepare a vector object for setting coefficient values. This routine will also re-initialize an already assembled vector, allowing users to modify coefficient values.

3.2.4

int
HYPRE_IJVectorSetMaxOffProcElmts (HYPRE_IJVector vector, int
max_off_proc_elmts)

(Optional) Sets the maximum number of elements that are expected to be set (or added) on other processors from this processor This routine can significantly improve the efficiency of matrix construction, and should always be utilized if possible.

Not collective.

3.2.5

int
HYPRE_IJVectorSetValues (HYPRE_IJVector vector, int nvalues, const int *indices, const double *values)

Sets values in vector. The arrays `values` and `indices` are of dimension `nvalues` and contain the vector values to be set and the corresponding global vector indices, respectively. Erases any previous values at the specified locations and replaces them with new ones.

Not collective.

3.2.6

int
HYPRE_IJVectorAddToValues (HYPRE_IJVector vector, int nvalues, const int *indices, const double *values)

Adds to values in vector. Usage details are analogous to HYPRE_IJVectorSetValues (→3.2.5, *page 37*).

Not collective.

3.2.7

int
HYPRE_IJVectorGetValues (HYPRE_IJVector vector, int nvalues, const int *indices, double *values)

Gets values in vector. Usage details are analogous to HYPRE_IJVectorSetValues (→3.2.5, *page 37*).

Not collective.

3.2.8

int **HYPRE_IJVectorSetObjectType** (HYPRE_IJVector vector, int type)

Set the storage type of the vector object to be constructed. Currently, `type` can only be `HYPRE_PARCSR`.

Not collective, but must be the same on all processes.

See Also: HYPRE_IJVectorGetObject (→3.2.9, *page 38*)

3.2.9

int **HYPRE_IJVectorGetObject** (HYPRE_IJVector vector, void **object)

Get a reference to the constructed vector object.

See Also: HYPRE_IJVectorSetObjectType (→3.2.8, *page 38*)

3.2.10

int
HYPRE_IJVectorRead (const char *filename, MPI_Comm comm, int type, HYPRE_IJVector *vector)

Read the vector from file. This is mainly for debugging purposes.

3.2.11

int **HYPRE_IJVectorPrint** (HYPRE_IJVector vector, const char *filename)

Print the vector to file. This is mainly for debugging purposes.

4

extern **Struct Solvers**

Names

These solvers use matrix/vector storage schemes that are tailored to structured grid problems.

4.1

Struct Solvers

Names

typedef struct hypre_StructSolver_struct* **HYPRE_StructSolver**
The solver object

4.2

Struct Jacobi Solver

Names

int
HYPRE_StructJacobiCreate (MPI_Comm comm,
HYPRE_StructSolver *solver)
Create a solver object

4.2.1 int
HYPRE_StructJacobiDestroy (HYPRE_StructSolver solver)

int
HYPRE_StructJacobiSetup (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b,
HYPRE_StructVector x)

int
HYPRE_StructJacobiSolve (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b,
HYPRE_StructVector x)
Solve the system

int
HYPRE_StructJacobiSetTol (HYPRE_StructSolver solver, double tol)
(Optional) Set the convergence tolerance

int
HYPRE_StructJacobiSetMaxIter (HYPRE_StructSolver solver, int max_iter)
(Optional) Set maximum number of iterations

int
HYPRE_StructJacobiSetZeroGuess (HYPRE_StructSolver solver)
(Optional) Use a zero initial guess

int
HYPRE_StructJacobiSetNonZeroGuess (HYPRE_StructSolver solver)
(Optional) Use a nonzero initial guess

int
HYPRE_StructJacobiGetNumIterations (HYPRE_StructSolver solver,
int *num_iterations)
Return the number of iterations taken

int
HYPRE_StructJacobiGetFinalRelativeResidualNorm
(HYPRE_StructSolver
solver,
double *norm)
Return the norm of the final relative residual

4.2.1

int **HYPRE_StructJacobiDestroy** (HYPRE_StructSolver solver)

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

4.3

Struct PFMG Solver

Names

int
HYPRE_StructPFMGCreate (MPI_Comm comm,
HYPRE_StructSolver *solver)
Create a solver object

int
HYPRE_StructPFMGDestroy (HYPRE_StructSolver solver)
Destroy a solver object

int
HYPRE_StructPFMGSetup (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b,
HYPRE_StructVector x)

int
HYPRE_StructPFMGSolve (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b,
HYPRE_StructVector x)
Solve the system

int
HYPRE_StructPFMGSetTol (HYPRE_StructSolver solver, double tol)
(Optional) Set the convergence tolerance

int
HYPRE_StructPFMGSetMaxIter (HYPRE_StructSolver solver,
int max_iter)
(Optional) Set maximum number of iterations

int
HYPRE_StructPFMGSetRelChange (HYPRE_StructSolver solver,
int rel_change)
(Optional) Additionally require that the relative difference in successive iterates be small

int

HYPRE_StructPFMGSetZeroGuess (HYPRE_StructSolver solver)
(Optional) Use a zero initial guess

int
HYPRE_StructPFMGSetNonZeroGuess (HYPRE_StructSolver solver)
(Optional) Use a nonzero initial guess

int
HYPRE_StructPFMGSetRelaxType (HYPRE_StructSolver solver, int relax_type)
(Optional) Set relaxation type

int
HYPRE_StructPFMGSetRAPType (HYPRE_StructSolver solver, int rap_type)
(Optional) Set type of code used for coarse operator

int
HYPRE_StructPFMGSetNumPreRelax (HYPRE_StructSolver solver, int num_pre_relax)
(Optional) Set number of pre-relaxation sweeps

int
HYPRE_StructPFMGSetNumPostRelax (HYPRE_StructSolver solver, int num_post_relax)
(Optional) Set number of post-relaxation sweeps

int
HYPRE_StructPFMGSetSkipRelax (HYPRE_StructSolver solver, int skip_relax)
(Optional) Skip relaxation on certain grids for isotropic problems

int
HYPRE_StructPFMGSetLogging (HYPRE_StructSolver solver, int logging)
(Optional) Set the amount of logging to do

int
HYPRE_StructPFMGSetPrintLevel (HYPRE_StructSolver solver, int print_level)
(Optional) To allow printing to the screen

int
HYPRE_StructPFMGGetNumIterations (HYPRE_StructSolver solver, int *num_iterations)
Return the number of iterations taken

int
HYPRE_StructPFMGGetFinalRelativeResidualNorm (HYPRE_StructSolver solver, double *norm)
Return the norm of the final relative residual

4.4 Struct SMG Solver

Names

int
HYPRE_StructSMGCreate (MPI_Comm comm,
HYPRE_StructSolver *solver)
Create a solver object

int
HYPRE_StructSMGDestroy (HYPRE_StructSolver solver)
Destroy a solver object

int
HYPRE_StructSMGSetup (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b, HYPRE_StructVector x)

int
HYPRE_StructSMGSolve (HYPRE_StructSolver solver,
HYPRE_StructMatrix A, HYPRE_StructVector b,
HYPRE_StructVector x)
Solve the system

int
HYPRE_StructSMGSetTol (HYPRE_StructSolver solver, double tol)
(Optional) Set the convergence tolerance

int
HYPRE_StructSMGSetMaxIter (HYPRE_StructSolver solver, int max_iter)
(Optional) Set maximum number of iterations

int
HYPRE_StructSMGSetRelChange (HYPRE_StructSolver solver,
int rel_change)
(Optional) Additionally require that the relative difference in successive iterates be small

int
HYPRE_StructSMGSetZeroGuess (HYPRE_StructSolver solver)
(Optional) Use a zero initial guess

int
HYPRE_StructSMGSetNonZeroGuess (HYPRE_StructSolver solver)
(Optional) Use a nonzero initial guess

int
HYPRE_StructSMGSetNumPreRelax (HYPRE_StructSolver solver,
int num_pre_relax)
(Optional) Set number of pre-relaxation sweeps

int

HYPRE_StructSMGSetNumPostRelax (HYPRE_StructSolver solver, int num_post_relax)
(Optional) Set number of post-relaxation sweeps

int
HYPRE_StructSMGSetLogging (HYPRE_StructSolver solver, int logging)
(Optional) Set the amount of logging to do

int
HYPRE_StructSMGSetPrintLevel (HYPRE_StructSolver solver, int print_level)
(Optional) To allow printing to the screen

int
HYPRE_StructSMGGetNumIterations (HYPRE_StructSolver solver, int *num_iterations)
Return the number of iterations taken

int
HYPRE_StructSMGGetFinalRelativeResidualNorm (HYPRE_StructSolver solver, double *norm)
Return the norm of the final relative residual

4.5 Struct PCG Solver

Names

int
HYPRE_StructPCGCreate (MPI_Comm comm, HYPRE_StructSolver *solver)
Create a solver object

int
HYPRE_StructPCGDestroy (HYPRE_StructSolver solver)
Destroy a solver object

int
HYPRE_StructPCGSetup (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)

int
HYPRE_StructPCGSolve (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)
Solve the system

int

HYPRE_StructPCGSetTol (HYPRE_StructSolver solver, double tol)
(Optional) Set the convergence tolerance

int
HYPRE_StructPCGSetMaxIter (HYPRE_StructSolver solver, int max_iter)
(Optional) Set maximum number of iterations

int
HYPRE_StructPCGSetTwoNorm (HYPRE_StructSolver solver,
int two_norm)
(Optional) Use the two-norm in stopping criteria

int
HYPRE_StructPCGSetRelChange (HYPRE_StructSolver solver,
int rel_change)
(Optional) Additionally require that the relative difference in successive iterates be small

int
HYPRE_StructPCGSetPrecond (HYPRE_StructSolver solver,
HYPRE_PtrToStructSolverFcn precond,
HYPRE_PtrToStructSolverFcn
precond_setup,
HYPRE_StructSolver precond_solver)
(Optional) Set the preconditioner to use

int
HYPRE_StructPCGSetLogging (HYPRE_StructSolver solver, int logging)
(Optional) Set the amount of logging to do

int
HYPRE_StructPCGSetPrintLevel (HYPRE_StructSolver solver, int level)
(Optional) Set the print level

int
HYPRE_StructPCGGetNumIterations (HYPRE_StructSolver solver,
int *num_iterations)
Return the number of iterations taken

int
HYPRE_StructPCGGetFinalRelativeResidualNorm (HYPRE_StructSolver
solver,
double *norm)
Return the norm of the final relative residual

int
HYPRE_StructPCGGetResidual (HYPRE_StructSolver solver,
void **residual)
Return the residual

int
HYPRE_StructDiagScaleSetup (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector y,
HYPRE_StructVector x)
Setup routine for diagonal preconditioning

int

HYPRE_StructDiagScale (HYPRE_StructSolver solver,
HYPRE_StructMatrix HA,
HYPRE_StructVector Hy,
HYPRE_StructVector Hx)

Solve routine for diagonal preconditioning

4.6

Struct GMRES Solver

Names

int
HYPRE_StructGMRESCreate (MPI_Comm comm,
HYPRE_StructSolver *solver)

Create a solver object

int
HYPRE_StructGMRESDestroy (HYPRE_StructSolver solver)

Destroy a solver object

int
HYPRE_StructGMRESSetup (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b,
HYPRE_StructVector x)

set up

int
HYPRE_StructGMRESSolve (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b,
HYPRE_StructVector x)

Solve the system

int
HYPRE_StructGMRESSetTol (HYPRE_StructSolver solver, double tol)

(Optional) Set the convergence tolerance

int
HYPRE_StructGMRESSetMaxIter (HYPRE_StructSolver solver,
int max_iter)

(Optional) Set maximum number of iterations

int
HYPRE_StructGMRESSetPrecond (HYPRE_StructSolver solver,
HYPRE_PtrToStructSolverFcn precond,
HYPRE_PtrToStructSolverFcn
precond_setup,
HYPRE_StructSolver precond_solver)

(Optional) Set the preconditioner to use

int

HYPRE_StructGMRESSetLogging (HYPRE_StructSolver solver,
int logging)

(Optional) Set the amount of logging to do

int
HYPRE_StructGMRESSetPrintLevel (HYPRE_StructSolver solver,
int level)

(Optional) Set the print level

int
HYPRE_StructGMRESGetNumIterations (HYPRE_StructSolver solver,
int *num_iterations)

Return the number of iterations taken

int
HYPRE_StructGMRESGetFinalRelativeResidualNorm (
HYPRE_StructSolver
solver,
double *norm)

Return the norm of the final relative residual

int
HYPRE_StructGMRESGetResidual (HYPRE_StructSolver solver,
void **residual)

Return the residual

4.7

Struct BiCGSTAB Solver

Names

int
HYPRE_StructBiCGSTABCreate (MPI_Comm comm,
HYPRE_StructSolver *solver)

Create a solver object

int
HYPRE_StructBiCGSTABDestroy (HYPRE_StructSolver solver)

Destroy a solver object

int
HYPRE_StructBiCGSTABSetup (HYPRE_StructSolver solver,
HYPRE_StructMatrix A,
HYPRE_StructVector b,
HYPRE_StructVector x)

set up

int

HYPRE_StructBiCGSTABSolve (HYPRE_StructSolver solver, HYPRE_StructMatrix A, HYPRE_StructVector b, HYPRE_StructVector x)

Solve the system

int
HYPRE_StructBiCGSTABSetTol (HYPRE_StructSolver solver, double tol)

(Optional) Set the convergence tolerance

int
HYPRE_StructBiCGSTABSetMaxIter (HYPRE_StructSolver solver, int max_iter)

(Optional) Set maximum number of iterations

int
HYPRE_StructBiCGSTABSetPrecond (HYPRE_StructSolver solver, HYPRE_PtrToStructSolverFcn precond, HYPRE_PtrToStructSolverFcn precond_setup, HYPRE_StructSolver precond_solver)

(Optional) Set the preconditioner to use

int
HYPRE_StructBiCGSTABSetLogging (HYPRE_StructSolver solver, int logging)

(Optional) Set the amount of logging to do

int
HYPRE_StructBiCGSTABSetPrintLevel (HYPRE_StructSolver solver, int level)

(Optional) Set the print level

int
HYPRE_StructBiCGSTABGetNumIterations (HYPRE_StructSolver solver, int *num_iterations)

Return the number of iterations taken

int
HYPRE_StructBiCGSTABGetFinalRelativeResidualNorm (HYPRE_StructSolver solver, double *norm)

Return the norm of the final relative residual

int
HYPRE_StructBiCGSTABGetResidual (HYPRE_StructSolver solver, void **residual)

Return the residual

5

extern **SStruct Solvers**

Names

These solvers use matrix/vector storage schemes that are taylored to semi-structured grid problems.

5.1

SStruct Solvers

Names

typedef struct hypre_SStructSolver_struct* **HYPRE_SStructSolver**
The solver object

5.2

SStruct PCG Solver

Names

int
HYPRE_SStructPCGCreate (MPI_Comm comm,
HYPRE_SStructSolver *solver)
Create a solver object

int
HYPRE_SStructPCGSetup (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector b,
HYPRE_SStructVector x)

int
HYPRE_SStructPCGSolve (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector b,
HYPRE_SStructVector x)
Solve the system

int
HYPRE_SStructPCGSetTol (HYPRE_SStructSolver solver, double tol)
(Optional) Set the convergence tolerance

int
HYPRE_SStructPCGSetMaxIter (HYPRE_SStructSolver solver,
int max_iter)
(Optional) Set maximum number of iterations

int
HYPRE_SStructPCGSetTwoNorm (HYPRE_SStructSolver solver,
int two_norm)
(Optional) Set type of norm to use in stopping criteria

int
HYPRE_SStructPCGSetRelChange (HYPRE_SStructSolver solver,
int rel_change)
(Optional) Set to use additional relative-change convergence test

int
HYPRE_SStructPCGSetPrecond (HYPRE_SStructSolver solver,
HYPRE_PtrToSStructSolverFcn precond,
HYPRE_PtrToSStructSolverFcn
precond_setup, void *precond_solver)
(Optional) Set the preconditioner to use

int
HYPRE_SStructPCGSetLogging (HYPRE_SStructSolver solver, int logging)
(Optional) Set the amount of logging to do

int
HYPRE_SStructPCGSetPrintLevel (HYPRE_SStructSolver solver, int level)
(Optional) Set the print level

int

HYPRE_SStructPCGGetNumIterations (HYPRE_SStructSolver solver, int *num_iterations)
Return the number of iterations taken

int
HYPRE_SStructPCGGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, double *norm)
Return the norm of the final relative residual

int
HYPRE_SStructPCGGetResidual (HYPRE_SStructSolver solver, void **residual)
Return the residual

5.2.1

int **HYPRE_SStructPCGDestroy** (HYPRE_SStructSolver solver)

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

5.3

SStruct BiCGSTAB Solver

Names

int
HYPRE_SStructBiCGSTABCreate (MPI_Comm comm, HYPRE_SStructSolver *solver)
Create a solver object

int

HYPRE_SStructBiCGSTABSetup (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector b,
HYPRE_SStructVector x)

int
HYPRE_SStructBiCGSTABSolve (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector b,
HYPRE_SStructVector x)

Solve the system

int
HYPRE_SStructBiCGSTABSetTol (HYPRE_SStructSolver solver,
double tol)

(Optional) Set the convergence tolerance

int
HYPRE_SStructBiCGSTABSetMaxIter (HYPRE_SStructSolver solver,
int max_iter)

(Optional) Set maximum number of iterations

int
HYPRE_SStructBiCGSTABSetPrecond (HYPRE_SStructSolver solver,
HYPRE_PtrToSStructSolverFcn
precond,
HYPRE_PtrToSStructSolverFcn
precond_setup,
void *precond_solver)

(Optional) Set the preconditioner to use

int
HYPRE_SStructBiCGSTABSetLogging (HYPRE_SStructSolver solver,
int logging)

(Optional) Set the amount of logging to do

int
HYPRE_SStructBiCGSTABSetPrintLevel (HYPRE_SStructSolver solver,
int level)

(Optional) Set the print level

int
HYPRE_SStructBiCGSTABGetNumIterations (HYPRE_SStructSolver
solver,
int *num_iterations)

Return the number of iterations taken

int
HYPRE_SStructBiCGSTABGetFinalRelativeResidualNorm
(HYPRE_SStructSolver
solver,
double
*norm)

Return the norm of the final relative residual

int

HYPRE_SStructBiCGSTABGetResidual (HYPRE_SStructSolver solver,
void **residual)

Return the residual

5.3.1

int **HYPRE_SStructBiCGSTABDestroy** (HYPRE_SStructSolver solver)

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

5.4

SStruct GMRES Solver

Names

int
HYPRE_SStructGMRESCreate (MPI_Comm comm,
HYPRE_SStructSolver *solver)

Create a solver object

int
HYPRE_SStructGMRESSetup (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector b,
HYPRE_SStructVector x)

int
HYPRE_SStructGMRESSolve (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector b,
HYPRE_SStructVector x)

Solve the system

int

HYPRE_SStructGMRESSetKDim (HYPRE_SStructSolver solver, int k_dim)
(Optional) Set the maximum size of the Krylov space

int
HYPRE_SStructGMRESSetTol (HYPRE_SStructSolver solver, double tol)
(Optional) Set the convergence tolerance

int
HYPRE_SStructGMRESSetMaxIter (HYPRE_SStructSolver solver, int max_iter)
(Optional) Set maximum number of iterations

int
HYPRE_SStructGMRESSetPrecond (HYPRE_SStructSolver solver, HYPRE_PtrToSStructSolverFcn precond, HYPRE_PtrToSStructSolverFcn precond_setup, void *precond_solver)
(Optional) Set the preconditioner to use

int
HYPRE_SStructGMRESSetLogging (HYPRE_SStructSolver solver, int logging)
(Optional) Set the amount of logging to do

int
HYPRE_SStructGMRESSetPrintLevel (HYPRE_SStructSolver solver, int print_level)
(Optional) Set the print level

int
HYPRE_SStructGMRESGetNumIterations (HYPRE_SStructSolver solver, int *num_iterations)
Return the number of iterations taken

int
HYPRE_SStructGMRESGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, double *norm)
Return the norm of the final relative residual

int
HYPRE_SStructGMRESGetResidual (HYPRE_SStructSolver solver, void **residual)
Return the residual

5.4.1

int **HYPRE_SStructGMRESDestroy** (HYPRE_SStructSolver solver)

Destroy a solver object. An object should be explicitly destroyed using this destructor when the user's code no longer needs direct access to it. Once destroyed, the object must not be referenced again. Note that the object may not be deallocated at the completion of this call, since there may be internal package references to the object. The object will then be destroyed when all internal reference counts go to zero.

5.5

SStruct SysPFMG Solver

Names

int
HYPRE_SStructSysPFMGCreate (MPI_Comm comm, HYPRE_SStructSolver *solver)
Create a solver object

int
HYPRE_SStructSysPFMGDestroy (HYPRE_SStructSolver solver)
Destroy a solver object

int
HYPRE_SStructSysPFMGSetup (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)

int
HYPRE_SStructSysPFMGSolve (HYPRE_SStructSolver solver, HYPRE_SStructMatrix A, HYPRE_SStructVector b, HYPRE_SStructVector x)
Solve the system

int
HYPRE_SStructSysPFMGSetTol (HYPRE_SStructSolver solver, double tol)
(Optional) Set the convergence tolerance

int
HYPRE_SStructSysPFMGSetMaxIter (HYPRE_SStructSolver solver, int max_iter)
(Optional) Set maximum number of iterations

int
HYPRE_SStructSysPFMGSetRelChange (HYPRE_SStructSolver solver, int rel_change)
(Optional) Additionally require that the relative difference in successive iterates be small

int
HYPRE_SStructSysPFMGSetZeroGuess (HYPRE_SStructSolver solver)
(Optional) Use a zero initial guess

int

HYPRE_SStructSysPFMGSetNonZeroGuess (HYPRE_SStructSolver solver)

(Optional) Use a nonzero initial guess

int
HYPRE_SStructSysPFMGSetRelaxType (HYPRE_SStructSolver solver, int relax_type)

(Optional) Set relaxation type

int
HYPRE_SStructSysPFMGSetNumPreRelax (HYPRE_SStructSolver solver, int num_pre_relax)

(Optional) Set number of pre-relaxation sweeps

int
HYPRE_SStructSysPFMGSetNumPostRelax (HYPRE_SStructSolver solver, int num_post_relax)

(Optional) Set number of post-relaxation sweeps

int
HYPRE_SStructSysPFMGSetSkipRelax (HYPRE_SStructSolver solver, int skip_relax)

(Optional) Skip relaxation on certain grids for isotropic problems

int
HYPRE_SStructSysPFMGSetLogging (HYPRE_SStructSolver solver, int logging)

(Optional) Set the amount of logging to do

int
HYPRE_SStructSysPFMGSetPrintLevel (HYPRE_SStructSolver solver, int print_level)

(Optional) Set the print level

int
HYPRE_SStructSysPFMGGetNumIterations (HYPRE_SStructSolver solver, int *num_iterations)

Return the number of iterations taken

int
HYPRE_SStructSysPFMGGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, double *norm)

Return the norm of the final relative residual

5.6

SStruct FAC Solver

Names

int

HYPRE_SStructFACCreate (MPI_Comm comm,
HYPRE_SStructSolver *solver)
Create a FAC solver object

int
HYPRE_SStructFACDestroy2 (HYPRE_SStructSolver solver)
Destroy a FAC solver object

int
HYPRE_SStructFACSetup2 (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector b,
HYPRE_SStructVector x)
Set up the FAC solver structure

int
HYPRE_SStructFACSolve3 (HYPRE_SStructSolver solver,
HYPRE_SStructMatrix A,
HYPRE_SStructVector b,
HYPRE_SStructVector x)
Solve the system

int
HYPRE_SStructFACSetPLevels (HYPRE_SStructSolver solver, int nparts,
int *plevels)
Set up amr structure

int
HYPRE_SStructFACSetPRefinements (HYPRE_SStructSolver solver,
int nparts, int (*rfactors)[3])
Set up amr refinement factors

int
HYPRE_SStructFACSetMaxLevels (HYPRE_SStructSolver solver,
int max_levels)
(Optional) Set max FAC levels

int
HYPRE_SStructFACSetTol (HYPRE_SStructSolver solver, double tol)
(Optional) Set the convergence tolerance

int
HYPRE_SStructFACSetMaxIter (HYPRE_SStructSolver solver,
int max_iter)
(Optional) Set maximum number of iterations

int
HYPRE_SStructFACSetRelChange (HYPRE_SStructSolver solver,
int rel_change)
(Optional) Additionally require that the relative difference in successive iterates be small

int
HYPRE_SStructFACSetZeroGuess (HYPRE_SStructSolver solver)
(Optional) Use a zero initial guess

int
HYPRE_SStructFACSetNonZeroGuess (HYPRE_SStructSolver solver)
(Optional) Use a nonzero initial guess

int

HYPRE_SStructFACSetRelaxType (HYPRE_SStructSolver solver, int relax_type)

(Optional) Set relaxation type

int
HYPRE_SStructFACSetNumPreRelax (HYPRE_SStructSolver solver, int num_pre_relax)

(Optional) Set number of pre-relaxation sweeps

int
HYPRE_SStructFACSetNumPostRelax (HYPRE_SStructSolver solver, int num_post_relax)

(Optional) Set number of post-relaxation sweeps

int
HYPRE_SStructFACSetCoarseSolverType (HYPRE_SStructSolver solver, int csolver_type)

(Optional) Set coarsest solver type

int
HYPRE_SStructFACSetLogging (HYPRE_SStructSolver solver, int logging)

(Optional) Set the amount of logging to do

int
HYPRE_SStructFACGetNumIterations (HYPRE_SStructSolver solver, int *num_iterations)

Return the number of iterations taken

int
HYPRE_SStructFACGetFinalRelativeResidualNorm (HYPRE_SStructSolver solver, double *norm)

Return the norm of the final relative residual

6

extern **ParCSR Solvers**

Names

These solvers use matrix/vector storage schemes that are taylored for general sparse matrix systems.

6.1

ParCSR Solvers

Names

#define **HYPRE_SOLVER_STRUCT**
The solver object

6.2

ParCSR BoomerAMG Solver and Preconditioner

Names

Parallel unstructured algebraic multigrid solver and preconditioner

6.2.1

int
HYPRE_BoomerAMGSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Set up the BoomerAMG solver or preconditioner. If used as a preconditioner, this function should be passed to the iterative solver `SetPrecond` function.

Parameters: `solver` — [IN] object to be set up.
`A` — [IN] ParCSR matrix used to construct the solver/preconditioner.
`b` — Ignored by this function.
`x` — Ignored by this function.

6.2.2

int
HYPRE_BoomerAMGSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Solve the system or apply AMG as a preconditioner. If used as a preconditioner, this function should be passed to the iterative solver `SetPrecond` function.

Parameters: `solver` — [IN] solver or preconditioner object to be applied.
`A` — [IN] ParCSR matrix, matrix of the linear system to be solved
`b` — [IN] right hand side of the linear system to be solved
`x` — [OUT] approximated solution of the linear system to be solved

6.2.3

int
HYPRE_BoomerAMGSolveT (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Solve the transpose system $A^T x = b$ or apply AMG as a preconditioner to the transpose system . If used as a preconditioner, this function should be passed to the iterative solver `SetPrecond` function.

Parameters: `solver` — [IN] solver or preconditioner object to be applied.
`A` — [IN] ParCSR matrix
`b` — [IN] right hand side of the linear system to be solved
`x` — [OUT] approximated solution of the linear system to be solved

6.2.4

int **HYPRE_BoomerAMGSetTol** (HYPRE_Solver solver, double tol)

(Optional) Set the convergence tolerance, if BoomerAMG is used as a solver. If it is used as a preconditioner, this function has no effect. The default is 1.e-7.

6.2.5

int **HYPRE_BoomerAMGSetMaxIter** (HYPRE_Solver solver, int max_iter)

(Optional) Sets maximum number of iterations, if BoomerAMG is used as a solver. If it is used as a preconditioner, this function has no effect. The default is 20.

6.2.6

int
HYPRE_BoomerAMGSetMaxLevels (HYPRE_Solver solver, int max_levels)

(Optional) Sets maximum number of multigrid levels. The default is 25.

6.2.7

int
HYPRE_BoomerAMGSetStrongThreshold (HYPRE_Solver solver, double strong_threshold)

(Optional) Sets AMG strength threshold. The default is 0.25. For 2d Laplace operators, 0.25 is a good value, for 3d Laplace operators, 0.5 or 0.6 is a better value. For elasticity problems, a large strength threshold, such as 0.9, is often better.

6.2.8

int
HYPRE_BoomerAMGSetMaxRowSum (HYPRE_Solver solver, double max_row_sum)

(Optional) Sets a parameter to modify the definition of strength for diagonal dominant portions of the matrix. The default is 0.9. If max_row_sum is 1, no checking for diagonally dominant rows is performed.

6.2.9

int
HYPRE_BoomerAMGSetCoarsenType (HYPRE_Solver solver, int coarsen_type)

(Optional) Defines which parallel coarsening algorithm is used. There are the following options for coarsen_type:

0	CLJP-coarsening (a parallel coarsening algorithm using independent sets.
1	classical Ruge-Stueben coarsening on each processor, no boundary treatment (not recommended!)
3	classical Ruge-Stueben coarsening on each processor, followed by a third pass, which adds coarse points on the boundaries
6	Falgout coarsening (uses 1 first, followed by CLJP using the interior coarse points generated by 1 as its first independent set)
7	CLJP-coarsening (using a fixed random vector, for debugging purposes only)
8	PMIS-coarsening (a parallel coarsening algorithm using independent sets, generating lower complexities than CLJP, might also lead to slower convergence)
9	PMIS-coarsening (using a fixed random vector, for debugging purposes only)
10	HMIS-coarsening (uses one pass Ruge-Stueben on each processor independently, followed by PMIS using the interior C-points generated as its first independent set)
11	one-pass Ruge-Stueben coarsening on each processor, no boundary treatment (not recommended!)

The default is 6.

6.2.10

int
HYPRE_BoomerAMGSetCycleType (HYPRE_Solver solver, int cycle_type)

(Optional) Defines the type of cycle. For a V-cycle, set cycle_type to 1, for a W-cycle set cycle_type to 2. The default is 1.

6.2.11

int
HYPRE_BoomerAMGSetNumGridSweeps (HYPRE_Solver solver, int *num_grid_sweeps)

(Optional) Defines the number of sweeps for the fine and coarse grid, the up and down cycle.

Note: This routine will be phased out!!!! Use HYPRE_BoomerAMGSetNumSweeps or HYPRE_BoomerAMGSetCycleNumSweeps instead.

6.2.12

int
HYPRE_BoomerAMGSetNumSweeps (HYPRE_Solver solver, int num_sweeps)

(Optional) Sets the number of sweeps. On the finest level, the up and the down cycle the number of sweeps are set to num_sweeps and on the coarsest level to 1. The default is 1.

6.2.13

int
HYPRE_BoomerAMGSetCycleNumSweeps (HYPRE_Solver solver, int num_sweeps, int k)

(Optional) Sets the number of sweeps at a specified cycle. There are the following options for k:

the finest level	if k=0
the down cycle	if k=1
the up cycle	if k=2
the coarsest level	if k=3.

6.2.14

int
HYPRE_BoomerAMGSetGridRelaxType (HYPRE_Solver solver, int *grid_relax_type)

(Optional) Defines which smoother is used on the fine and coarse grid, the up and down cycle.

Note: This routine will be phased out!!!! Use HYPRE_BoomerAMGSetRelaxType or HYPRE_BoomerAMGSetCycleRelaxType instead.

6.2.15

int
HYPRE_BoomerAMGSetRelaxType (HYPRE_Solver solver, int relax_type)

(Optional) Defines the smoother to be used. It uses the given smoother on the fine grid, the up and the down cycle and sets the solver on the coarsest level to Gaussian elimination (9). The default is Gauss-Seidel (3).

There are the following options for relax_type:

0	Jacobi
1	Gauss-Seidel, sequential (very slow!)
2	Gauss-Seidel, interior points in parallel, boundary sequential (slow!)
3	hybrid Gauss-Seidel or SOR, forward solve
4	hybrid Gauss-Seidel or SOR, backward solve
5	hybrid chaotic Gauss-Seidel (works only with OpenMP)
6	hybrid symmetric Gauss-Seidel or SSOR
9	Gaussian elimination (only on coarsest level)

6.2.16

int
HYPRE_BoomerAMGSetCycleRelaxType (HYPRE_Solver solver, int relax_type, int k)

(Optional) Defines the smoother at a given cycle. For options of relax_type see description of HYPRE_BoomerAMGSetRelaxType). Options for k are

the finest level	if k=0
the down cycle	if k=1
the up cycle	if k=2
the coarsest level	if k=3.

6.2.17

int
HYPRE_BoomerAMGSetRelaxOrder (HYPRE_Solver solver, int relax_order)

(Optional) Defines in which order the points are relaxed. There are the following options for relax_order:

0	the points are relaxed in natural or lexicographic order on each processor
1	CF-relaxation is used, i.e on the fine grid and the down cycle the coarse points are relaxed first, followed by the fine points; on the up cycle the F-points are relaxed first, followed by the C-points. On the coarsest level, if an iterative scheme is used, the points are relaxed in lexicographic order.

The default is 1 (CF-relaxation).

6.2.18

int
HYPRE_BoomerAMGSetGridRelaxPoints (HYPRE_Solver solver, int **grid_relax_points)

(Optional) Defines in which order the points are relaxed.

Note: This routine will be phased out!!!! Use HYPRE_BoomerAMGSetRelaxOrder instead.

6.2.19

int
HYPRE_BoomerAMGSetRelaxWeight (HYPRE_Solver solver, double *relax_weight)

(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid SOR.

Note: This routine will be phased out!!!! Use HYPRE_BoomerAMGSetRelaxWt or HYPRE_BoomerAMGSetLevelRelaxWt instead.

6.2.20

int
HYPRE_BoomerAMGSetRelaxWt (HYPRE_Solver solver, double relax_weight)

(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid SOR on all levels.

relax_weight > 0	this assigns the given relaxation weight on all levels
relax_weight = 0	the weight is determined on each level with the estimate $\frac{3}{4\|D^{-1/2}AD^{-1/2}\|}$, where D is the diagonal matrix of A (this should only be used with Jacobi)
relax_weight = -k	the relaxation weight is determined with at most k CG steps on each level this should only be used for symmetric positive definite problems)

The default is 1.

6.2.21

int
HYPRE_BoomerAMGSetLevelRelaxWt (HYPRE_Solver solver, double relax_weight, int level)

(Optional) Defines the relaxation weight for smoothed Jacobi and hybrid SOR on the user defined level. Note that the finest level is denoted 0, the next coarser level 1, etc. For nonpositive relax_weight, the parameter is determined on the given level as described for HYPRE_BoomerAMGSetRelaxWt. The default is 1.

6.2.22

int **HYPRE_BoomerAMGSetOmega** (HYPRE_Solver solver, double *omega)

(Optional) Defines the outer relaxation weight for hybrid SOR. Note: This routine will be phased out!!!! Use HYPRE_BoomerAMGSetOuterWt or HYPRE_BoomerAMGSetLevelOuterWt instead.

6.2.23

int **HYPRE_BoomerAMGSetOuterWt** (HYPRE_Solver solver, double omega)

(Optional) Defines the outer relaxation weight for hybrid SOR and SSOR on all levels.

omega > 0	this assigns the same outer relaxation weight omega on each level
omega = -k	an outer relaxation weight is determined with at most k CG steps on each level (this only makes sense for symmetric positive definite problems and smoothers, e.g. SSOR)

The default is 1.

6.2.24

int
HYPRE_BoomerAMGSetLevelOuterWt (HYPRE_Solver solver, double omega, int level)

(Optional) Defines the outer relaxation weight for hybrid SOR or SSOR on the user defined level. Note that the finest level is denoted 0, the next coarser level 1, etc. For nonpositive omega, the parameter is determined on the given level as described for HYPRE_BoomerAMGSetOuterWt. The default is 1.

6.2.25

int
HYPRE_BoomerAMGSetTruncFactor (HYPRE_Solver solver, double trunc_factor)

(Optional) Defines a truncation factor for the interpolation. The default is 0.

6.2.26

int
HYPRE_BoomerAMGSetSCommPkgSwitch (HYPRE_Solver solver, double S_commpkg_switch)

(Optional) Defines the largest strength threshold for which the strength matrix S uses the communication package of the operator A. If the strength threshold is larger than this values, a communication package is generated for S. This can save memory and decrease the amount of data that needs to be communicated, if S is substantially sparser than A. The default is 1.0.

6.2.27

int
HYPRE_BoomerAMGSetSmoothType (HYPRE_Solver solver, int smooth_type)

(Optional) Enables the use of more complex smoothers. The following options exist for smooth_type:

value	smoother	routines needed to set smoother parameters
6	Schwarz smoothers	HYPRE_BoomerAMGSetDomainType, HYPRE_BoomerAMGSetOverlap,
		HYPRE_BoomerAMGSetVariant, HYPRE_BoomerAMGSetSchwarzRlxW
7	Pilut	HYPRE_BoomerAMGSetDropTol, HYPRE_BoomerAMGSetMaxNzPerRo
8	ParaSails	HYPRE_BoomerAMGSetSym, HYPRE_BoomerAMGSetLevel,
		HYPRE_BoomerAMGSetFilter, HYPRE_BoomerAMGSetThreshold
9	Euclid	HYPRE_BoomerAMGSetEuclidFile
16	CG preconditioned with Schwarz	see routines under 6
17	CG preconditioned with Pilut	see routines under 7
18	CG preconditioned with ParaSails	see routines under 8
19	CG preconditioned with Euclid	see routines under 9

The default is 6. Also, if no smoother parameters are set via the routines mentioned in the table above, default values are used.

6.2.28

int
HYPRE_BoomerAMGSetSmoothNumLevels (HYPRE_Solver solver, int smooth_num_levels)

(Optional) Sets the number of levels for more complex smoothers. The smoothers, as defined by HYPRE_BoomerAMGSetSmoothType, will be used on level 0 (the finest level) through level smooth_num_levels-1. The default is 0, i.e. no complex smoothers are used.

6.2.29

int
HYPRE_BoomerAMGSetSmoothNumSweeps (HYPRE_Solver solver, int smooth_num_sweeps)

(Optional) Sets the number of sweeps for more complex smoothers. The default is 1.

6.2.30

int **HYPRE_BoomerAMGSetLogging** (HYPRE_Solver solver, int logging)

(Optional) Requests additional computations for diagnostic and similar data to be logged by the user. Default to 0 for do nothing. The latest residual will be available if logging > 1.

6.2.31

int
HYPRE_BoomerAMGSetNumFunctions (HYPRE_Solver solver, int num_functions)

(Optional) Sets the size of the system of PDEs, if using the systems version. The default is 1.

6.2.32

int **HYPRE_BoomerAMGSetNodal** (HYPRE_Solver solver, int nodal)

(Optional) Sets whether to use the nodal systems version. The default is 0.

6.2.33

int **HYPRE_BoomerAMGSetDofFunc** (HYPRE_Solver solver, int *dof_func)

(Optional) Sets the mapping that assigns the function to each variable, if using the systems version. If no assignment is made and the number of functions is k > 1, the mapping generated is (0,1,...,k-1,0,1,...,k-1,...).

6.2.34

int
HYPRE_BoomerAMGSetAggNumLevels (HYPRE_Solver solver, int agg_num_levels)

(Optional) Defines the number of levels of aggressive coarsening. The default is 0, i.e. no aggressive coarsening.

6.2.35

int
HYPRE_BoomerAMGSetNumPaths (HYPRE_Solver solver, int num_paths)

(Optional) Defines the degree of aggressive coarsening. The default is 1.

6.2.36

int **HYPRE_BoomerAMGSetVariant** (HYPRE_Solver solver, int variant)

(Optional) Defines which variant of the Schwarz method is used. The following options exist for variant:

0	hybrid multiplicative Schwarz method (no overlap across processor boundaries)
1	hybrid additive Schwarz method (no overlap across processor boundaries)
2	additive Schwarz method
3	hybrid multiplicative Schwarz method (with overlap across processor boundaries)

The default is 0.

6.2.37

int **HYPRE_BoomerAMGSetOverlap** (HYPRE_Solver solver, int overlap)

(Optional) Defines the overlap for the Schwarz method. The following options exist for overlap:

0	no overlap
1	minimal overlap (default)
2	overlap generated by including all neighbors of domain boundaries

6.2.38

int
HYPRE_BoomerAMGSetDomainType (HYPRE_Solver solver, int domain_type)

(Optional) Defines the type of domain used for the Schwarz method. The following options exist for domain_type:

0	each point is a domain
1	each node is a domain (only of interest in "systems" AMG)
2	each domain is generated by agglomeration (default)

6.2.39

int **HYPRE_BoomerAMGSetSym** (HYPRE_Solver solver, int sym)

(Optional) Defines symmetry for ParaSAILS. For further explanation see description of ParaSAILS.

6.2.40

int **HYPRE_BoomerAMGSetLevel** (HYPRE_Solver solver, int level)

(Optional) Defines number of levels for ParaSAILS. For further explanation see description of ParaSAILS.

6.2.41

int
HYPRE_BoomerAMGSetThreshold (HYPRE_Solver solver, double threshold)

(Optional) Defines threshold for ParaSAILS. For further explanation see description of ParaSAILS.

6.2.42

int **HYPRE_BoomerAMGSetFilter** (HYPRE_Solver solver, double filter)

(Optional) Defines filter for ParaSAILS. For further explanation see description of ParaSAILS.

6.2.43

int **HYPRE_BoomerAMGSetDropTol** (HYPRE_Solver solver, double drop_tol)

(Optional) Defines drop tolerance for PILUT. For further explanation see description of PILUT.

6.2.44

int
HYPRE_BoomerAMGSetMaxNzPerRow (HYPRE_Solver solver, int max_nz_per_row)

(Optional) Defines maximal number of nonzeros for PILUT. For further explanation see description of PILUT.

6.2.45

int
HYPRE_BoomerAMGSetEuclidFile (HYPRE_Solver solver, char *euclidfile)

(Optional) Defines name of an input file for Euclid parameters. For further explanation see description of Euclid.

6.2.46

int **HYPRE_BoomerAMGSetGSMG** (HYPRE_Solver solver, int gsmg)

(Optional) Specifies the use of GSMG - geometrically smooth coarsening and interpolation. Currently any nonzero value for gsmg will lead to the use of GSMG. The default is 0, i.e. (GSMG is not used)

6.3

ParCSR ParaSails Preconditioner

Names

Parallel sparse approximate inverse preconditioner for the ParCSR matrix format.

6.3.1

int
HYPRE_ParaSailsSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Set up the ParaSails preconditioner. This function should be passed to the iterative solver `SetPrecond` function.

Parameters:
`solver` — [IN] Preconditioner object to set up.
`A` — [IN] ParCSR matrix used to construct the preconditioner.
`b` — Ignored by this function.
`x` — Ignored by this function.

6.3.2

int
HYPRE_ParaSailsSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Apply the ParaSails preconditioner. This function should be passed to the iterative solver `SetPrecond` function.

Parameters:

`solver` — [IN] Preconditioner object to apply.
`A` — Ignored by this function.
`b` — [IN] Vector to precondition.
`x` — [OUT] Preconditioned vector.

6.3.3

int
HYPRE_ParaSailsSetParams (HYPRE_Solver solver, double thresh, int nlevels)

Set the threshold and levels parameter for the ParaSails preconditioner. The accuracy and cost of ParaSails are parameterized by these two parameters. Lower values of the threshold parameter and higher values of levels parameter lead to more accurate, but more expensive preconditioners.

Parameters:

`solver` — [IN] Preconditioner object for which to set parameters.
`thresh` — [IN] Value of threshold parameter, $0 \leq$ thresh ≤ 1. The default value is 0.1.
`nlevels` — [IN] Value of levels parameter, $0 \leq$ nlevels. The default value is 1.

6.3.4

int **HYPRE_ParaSailsSetFilter** (HYPRE_Solver solver, double filter)

Set the filter parameter for the ParaSails preconditioner.

Parameters:

`solver` — [IN] Preconditioner object for which to set filter parameter.
`filter` — [IN] Value of filter parameter. The filter parameter is used to drop small nonzeros in the preconditioner, to reduce the cost of applying the preconditioner. Values from 0.05 to 0.1 are recommended. The default value is 0.1.

6.3.5

int **HYPRE_ParaSailsSetSym** (HYPRE_Solver solver, int sym)

Set the symmetry parameter for the ParaSails preconditioner.

Parameters:

`solver` — [IN] Preconditioner object for which to set symmetry parameter.
`sym` — [IN] Value of the symmetry parameter:

value	meaning
0	nonsymmetric and/or indefinite problem, and nonsymmetric preconditioner
1	SPD problem, and SPD (factored) preconditioner
2	nonsymmetric, definite problem, and SPD (factored) preconditioner

6.3.6

int **HYPRE_ParaSailsSetLoadbal** (HYPRE_Solver solver, double loadbal)

Set the load balance parameter for the ParaSails preconditioner.

Parameters:

`solver` — [IN] Preconditioner object for which to set the load balance parameter.
`loadbal` — [IN] Value of the load balance parameter, $0 \leq$ loadbal ≤ 1. A zero value indicates that no load balance is attempted; a value of unity indicates that perfect load balance will be attempted. The recommended value is 0.9 to balance the overhead of data exchanges for load balancing. No load balancing is needed if the preconditioner is very sparse and fast to construct. The default value when this parameter is not set is 0.

6.3.7

int **HYPRE_ParaSailsSetReuse** (HYPRE_Solver solver, int reuse)

Set the pattern reuse parameter for the ParaSails preconditioner.

Parameters:

`solver` — [IN] Preconditioner object for which to set the pattern reuse parameter.
`reuse` — [IN] Value of the pattern reuse parameter. A nonzero value indicates that the pattern of the preconditioner should be reused for subsequent constructions of the preconditioner. A zero value indicates that the preconditioner should be constructed from scratch. The default value when this parameter is not set is 0.

6.3.8

int **HYPRE_ParaSailsSetLogging** (HYPRE_Solver solver, int logging)

Set the logging parameter for the ParaSails preconditioner.

Parameters:

`solver` — [IN] Preconditioner object for which to set the logging parameter.
`logging` — [IN] Value of the logging parameter. A nonzero value sends statistics of the setup procedure to stdout. The default value when this parameter is not set is 0.

6.3.9

int
HYPRE_ParaSailsBuildIJMatrix (HYPRE_Solver solver, HYPRE_IJMatrix *pij_A)

Build IJ Matrix of the sparse approximate inverse (factor). This function explicitly creates the IJ Matrix corresponding to the sparse approximate inverse or the inverse factor. Example: HYPRE_IJMatrix ij_A; HYPRE_ParaSailsBuildIJMatrix(solver, &ij_A);

Parameters:
`solver` — [IN] Preconditioner object.
`pij_A` — [OUT] Pointer to the IJ Matrix.

6.4

ParCSR Euclid Preconditioner

Names

MPI Parallel ILU preconditioner

Options summary:

Option	Default	Synopsis
-level	1	ILU(k) factorization level
-bj	0 (false)	Use Block Jacobi ILU instead of PILU
-eu_stats	0 (false)	Print internal timing and statistics
-eu_mem	0 (false)	Print internal memory usage

6.4.1

int
HYPRE_EuclidSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Set up the Euclid preconditioner. This function should be passed to the iterative solver `SetPrecond` function.

Parameters:
`solver` — [IN] Preconditioner object to set up.
`A` — [IN] ParCSR matrix used to construct the preconditioner.
`b` — Ignored by this function.
`x` — Ignored by this function.

6.4.2

int
HYPRE_EuclidSolve (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector b, HYPRE_ParVector x)

Apply the Euclid preconditioner. This function should be passed to the iterative solver `SetPrecond` function.

Parameters:
`solver` — [IN] Preconditioner object to apply.
`A` — Ignored by this function.
`b` — [IN] Vector to precondition.
`x` — [OUT] Preconditioned vector.

6.4.3

int **HYPRE_EuclidSetParams** (HYPRE_Solver solver, int argc, char *argv[])

Insert (name, value) pairs in Euclid's options database by passing Euclid the command line (or an array of strings). All Euclid options (e.g, level, drop-tolerance) are stored in this database. If a (name, value) pair already exists, this call updates the value. See also: HYPRE_EuclidSetParamsFromFile.

Parameters: `argc` — [IN] Length of argv array
`argv` — [IN] Array of strings

6.4.4

int
HYPRE_EuclidSetParamsFromFile (HYPRE_Solver solver, char *filename)

Insert (name, value) pairs in Euclid's options database. Each line of the file should either begin with a "#," indicating a comment line, or contain a (name value) pair, e.g:

```
>cat optionsFile
#sample runtime parameter file
-blockJacobi 3
-matFile /home/hysom/myfile.euclid
-doSomething true
-xx_coeff -1.0
```

See also: HYPRE_EuclidSetParams.

Parameters: `filename[IN]` — Pathname/filename to read

6.5 ParCSR Pilut Preconditioner

Names

int
HYPRE_ParCSRPilutCreate (MPI_Comm comm, HYPRE_Solver *solver)
Create a preconditioner object

int
HYPRE_ParCSRPilutDestroy (HYPRE_Solver solver)
Destroy a preconditioner object

int
HYPRE_ParCSRPilutSetup (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)

int

HYPRE_ParCSRPilutSolve (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)

Precondition the system

int
HYPRE_ParCSRPilutSetMaxIter (HYPRE_Solver solver, int max_iter)

(Optional) Set maximum number of iterations

int
HYPRE_ParCSRPilutSetDropTolerance (HYPRE_Solver solver, double tol)

(Optional)

int
HYPRE_ParCSRPilutSetFactorRowSize (HYPRE_Solver solver, int size)

(Optional)

6.6

ParCSR PCG Solver

Names

int
HYPRE_ParCSRPCGCreate (MPI_Comm comm, HYPRE_Solver *solver)

Create a solver object

int
HYPRE_ParCSRPCGDestroy (HYPRE_Solver solver)

Destroy a solver object

int
HYPRE_ParCSRPCGSetup (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)

int
HYPRE_ParCSRPCGSolve (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)

Solve the system

int
HYPRE_ParCSRPCGSetTol (HYPRE_Solver solver, double tol)

(Optional) Set the convergence tolerance

int
HYPRE_ParCSRPCGSetMaxIter (HYPRE_Solver solver, int max_iter)

(Optional) Set maximum number of iterations

int
HYPRE_ParCSRPCGSetTwoNorm (HYPRE_Solver solver, int two_norm)

(Optional) Use the two-norm in stopping criteria

int

HYPRE_ParCSRPCGSetRelChange (HYPRE_Solver solver, int rel_change)
(Optional) Additionally require that the relative difference in successive iterates be small

int
HYPRE_ParCSRPCGSetPrecond (HYPRE_Solver solver, HYPRE_PtrToParSolverFcn precond, HYPRE_PtrToParSolverFcn precond_setup, HYPRE_Solver precond_solver)
(Optional) Set the preconditioner to use

int
HYPRE_ParCSRPCGGetPrecond (HYPRE_Solver solver, HYPRE_Solver *precond_data)

int
HYPRE_ParCSRPCGSetLogging (HYPRE_Solver solver, int logging)
(Optional) Set the amount of logging to do

int
HYPRE_ParCSRPCGSetPrintLevel (HYPRE_Solver solver, int print_level)
(Optional) Set the print level

int
HYPRE_ParCSRPCGGetNumIterations (HYPRE_Solver solver, int *num_iterations)
Return the number of iterations taken

int
HYPRE_ParCSRPCGGetFinalRelativeResidualNorm (HYPRE_Solver solver, double *norm)
Return the norm of the final relative residual

int
HYPRE_ParCSRDiagScaleSetup (HYPRE_Solver solver, HYPRE_ParCSRMatrix A, HYPRE_ParVector y, HYPRE_ParVector x)
Setup routine for diagonal preconditioning

int
HYPRE_ParCSRDiagScale (HYPRE_Solver solver, HYPRE_ParCSRMatrix HA, HYPRE_ParVector Hy, HYPRE_ParVector Hx)
Solve routine for diagonal preconditioning

6.7

ParCSR GMRES Solver

Names

int
HYPRE_ParCSRGMRESCreate (MPI_Comm comm,
HYPRE_Solver *solver)
Create a solver object

int
HYPRE_ParCSRGMRESDestroy (HYPRE_Solver solver)
Destroy a solver object

int
HYPRE_ParCSRGMRESSetup (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A,
HYPRE_ParVector b,
HYPRE_ParVector x)

int
HYPRE_ParCSRGMRESSolve (HYPRE_Solver solver,
HYPRE_ParCSRMatrix A,
HYPRE_ParVector b, HYPRE_ParVector x)
Solve the system

int
HYPRE_ParCSRGMRESSetKDim (HYPRE_Solver solver, int k_dim)
(Optional) Set the maximum size of the Krylov space

int
HYPRE_ParCSRGMRESSetTol (HYPRE_Solver solver, double tol)
(Optional) Set the convergence tolerance

int
HYPRE_ParCSRGMRESSetMaxIter (HYPRE_Solver solver, int max_iter)
(Optional) Set maximum number of iterations

int
HYPRE_ParCSRGMRESSetPrecond (HYPRE_Solver solver,
HYPRE_PtrToParSolverFcn precond,
HYPRE_PtrToParSolverFcn
precond_setup,
HYPRE_Solver precond_solver)
(Optional) Set the preconditioner to use

int
HYPRE_ParCSRGMRESGetPrecond (HYPRE_Solver solver,
HYPRE_Solver *precond_data)

int
HYPRE_ParCSRGMRESSetLogging (HYPRE_Solver solver, int logging)
(Optional) Set the amount of logging to do

int
HYPRE_ParCSRGMRESSetPrintLevel (HYPRE_Solver solver,
int print_level)
(Optional) Set print level

int
HYPRE_ParCSRGMRESGetNumIterations (HYPRE_Solver solver,
int *num_iterations)
Return the number of iterations taken

int

HYPRE_ParCSRGMRESGetFinalRelativeResidualNorm (HYPRE_Solver solver, double *norm)

Return the norm of the final relative residual

7 Finite Element Interface

Names

7.1 FEI functions

Names

7.1.1

LLNL_FEI_Fei (MPI_Comm comm)

Parameters: `comm` — - an MPI communicator

7.1.2

~LLNL_FEI_Fei ()

Parameters: - — no parameter needed

7.1.3

int **parameters** (int numParams, char **paramStrings)

Parameters:

`numParams` — - number of command strings
`paramStrings` — - the command strings

7.1.4

int **initFields** (int numFields, int *fieldSizes, int *fieldIDs)

Each node or element variable has one or more fields. The field information can be set up using this function.

Parameters:

`numFields` — - total number of fields for all variable types
`fieldSizes` — - degree of freedom for each field type
`fieldIDs` — - a list of field identifiers

7.1.5

int
initElemBlock (int elemBlockID, int numElements, int numNodesPerElement, int *numFieldsPerNode, int **nodalFieldIDs, int numElemDOFFieldsPerElement, int *elemDOFFieldIDs, int interleaveStrategy)

The whole finite element mesh can be broken down into a number of element blocks. The attributes for each element block are: an identifier, number of elements, number of nodes per elements, the number of fields in each element node, etc.

Parameters:

`elemblockID` — - element block identifier
`numElements` — - number of element in this block
`numNodesPerElement` — - number of nodes per element in this block
`numFieldsPerNode` — - number of fields for each node
`nodalFieldIDs` — - field identifiers for the nodal unknowns
`numElemDOFFieldsPerElement` — - number of fields for the element
`elemDOFFieldIDs` — - field identifier for the element unknowns
`interleaveStratety` — - indicates how unknowns are ordered

7.1.6

int **initElem** (int elemBlockID, int elemID, int *elemConn)

This function initializes element connectivity (that is, the node identifiers associated with the current element) given an element block identifier and the element identifier with the element block.

Parameters:
`elemblockID` — - element block identifier
`elemID` — - element identifier
`elemConn` — - a list of node identifiers for this element

7.1.7

int
initSharedNodes (int nShared, int *sharedIDs, int *sharedLengs, int **sharedProcs)

This function initializes the nodes that are shared between the current processor and its neighbors. The FEI will decide a unique processor each shared node will be assigned to.

Parameters:
`nShared` — - number of shared nodes
`sharedIDs` — - shared node identifiers
`sharedLengs` — - the number of processors each node shares with
`sharedProcs` — - the processor identifiers each node shares with

7.1.8

int **initCRMult** (int CRListLen, int *CRNodeList, int *CRFieldList, int *CRID)

Parameters:
`CRListLen` — - the number of constraints
`CRNodeList` — - node identifiers where constraints are applied
`CRFieldList` — - field identifiers within nodes where constraints are applied
`CRID` — - the constraint identifier

7.1.9

int **initComplete** ()

This function signals to the FEI that the initialization step has been completed. The loading step will follow.

Parameters: - — no parameter needed

7.1.10

int **resetSystem** (double s)

This function resets the global matrix to be of the same sparsity pattern as before but with every entry set to s. The right hand side is set to 0.

Parameters: s — - the value each matrix entry is set to.

7.1.11

int **resetMatrix** (double s)

Parameters: s — - the value each matrix entry is set to.

7.1.12

int **resetRHSVector** (double s)

Parameters: `s` — - the value each right hand side vector entry is set to.

7.1.13

int **resetInitialGuess** (double s)

Parameters: `s` — - the value each solution vector entry is set to.

7.1.14

int
loadNodeBCs (int nNodes, int *nodeIDs, int fieldID, double **alpha, double **beta, double **gamma)

This function loads the nodal boundary conditions. The boundary conditions allowed are of the robin type.

Parameters:
`nNodes` — - number of nodes boundary conditions are imposed
`nodeIDs` — - nodal identifiers
`fieldID` — - field identifier with nodes where BC are imposed
`alpha` — - the multipliers for the field
`beta` — - the multipliers for the normal derivative of the field
`gamma` — - the boundary values on the right hand side of the equations

7.1.15

int
sumInElem (int elemBlockID, int elemID, int *elemConn, double **elemStiff, double *elemLoad, int elemFormat)

Parameters:

`elemBlockID` — - element block identifier
`elemID` — - element identifier
`elemConn` — - a list of node identifiers for this element
`elemStiff` — - element stiffness matrix
`elemLoad` — - right hand side (load) for this element
`elemFormat` — - the format the unknowns are passed in

7.1.16

int
sumInElemMatrix (int elemBlock, int elemID, int* elemConn, double **elemStiffness, int elemFormat)

Parameters:

`elemBlockID` — - element block identifier
`elemID` — - element identifier
`elemConn` — - a list of node identifiers for this element
`elemStiff` — - element stiffness matrix
`elemFormat` — - the format the unknowns are passed in

7.1.17

int
sumInElemRHS (int elemBlock, int elemID, int *elemConn, double *elemLoad)

Parameters:

`elemBlockID` — - element block identifier
`elemID` — - element identifier
`elemConn` — - a list of node identifiers for this element
`elemLoad` — - right hand side (load) for this element

7.1.18

int **loadComplete** ()

This function signals to the FEI that the loading phase has been completed.

Parameters: `-` — no parameter needed

7.1.19

int **getNumBlockActNodes** (int elemBlockID, int *nNodes)

Parameters:
`elemBlockID` — - element block identifier
`nNodes` — - the number of nodes to be returned

7.1.20

int **getNumBlockActEqns** (int elemBlockID, int *nEqns)

Parameters:
`elemBlockID` — - element block identifier
`nEqns` — - the number of unknowns to be returned

7.1.21

int **getBlockNodeIDList** (int elemBlockID, int numNodes, int *nodeIDList)

Parameters:
`elemBlockID` — - element block identifier
`numNodes` — - the number of nodes
`nodeIDList` — - the node identifiers

7.1.22

int
getBlockNodeSolution (int elemBlockID, int numNodes, int *nodeIDList, int *solnOffsets, double *solnValues)

Parameters:

`elemBlockID` — - element block identifier
`numNodes` — - the number of nodes
`nodeIDList` — - the node identifiers
`solnOffsets` — - the equation number for each nodal solution
`solnValues` — - the nodal solution values

7.1.23

int
loadCRMult (int CRID, int CRListLen, int *CRNodeList, int *CRFieldList, double *CRWeightList, double CRValue)

Parameters:

`CRID` — - the constraint identifier
`CRListLen` — - the number of constraints
`CRNodeList` — - node identifiers where constraints are applied
`CRFieldList` — - field identifiers within nodes where constraints are applied
`CRWeightList` — - a list of weights applied to each specified field
`CRValue` — - the constraint value (right hand side of the constraint)

Class Graph