Reference Manual

Generated by Doxygen 1.4.7

Thu Nov 30 22:56:56 2006

Contents

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 2

File Documentation

2.1 parms.h File Reference

This is the main pARMS include file. It should be included in all application program.

```
#include "parms_mem.h"
#include "parms_map.h"
#include "parms_vec.h"
#include "parms_mat.h"
#include "parms_pc.h"
#include "parms_solver.h"
#include "parms_viewer.h"
#include "parms_timer.h"
```

2.1.1 Detailed Description

This is the main pARMS include file. It should be included in all application program.

Author:

Zhongze Li

Date:

Sun Oct 22 16:39:37 2006

2.2 parms_comm.h File Reference

The communication handler used for the matrix-vector product.

```
#include "parms_sys.h"
#include "parms_viewer.h"
```

Typedefs

- typedef typedefPARMS_CXX_BEGIN struct parms_Comm_ ∗ **parms_Comm**

Functions

- int **parms_CommCreate (parms_Comm ∗self, MPI_Comm comm)**
- **int parms_CommFree (parms_Comm ∗self)**
- **int parms_CommDataBegin (parms_Comm self, void ∗data, int offset)**
- **int parms_CommDataEnd (parms_Comm self)**
- **int parms_CommView (parms_Comm self, parms_Viewer v)**
- **int parms_CommGetNumRecv (parms_Comm self)**
- **int parms_CommGetRecvBuf (parms_Comm self, FLOAT ∗∗rbuf)**
- **int parms_CommGetNumSend (parms_Comm self)**

2.2.1 Detailed Description

The communication handler used for the matrix-vector product.

Author:

Zhongze Li

Date:

Tue Oct 17 10:01:17 2006

2.2.2 Function Documentation

2.2.2.1 int parms_CommCreate (parms_Comm ∗ *self*, MPI_Comm *comm*)

Create a parms_Comm object.

Parameters:

self A pointer to the parms_Comm object.

comm MPI communicator

Returns:

0 on success.

2.2.2.2 int parms_CommDataBegin (parms_Comm *self*, void * *data*, int *offset*)

Exchange data for the matrix-vector product.

This functions exchanges the interface variables. The offset indicates the distance between start of the data to be exchanged and the start of the local vector. This is useful for Schur complement based preconditioner since the data parameter is only the interface part of the local vector rather than the entire local vector.

Parameters:

self A parms_Comm object.

data The data to be exchanged.

offset The distance between the start of the data and the beginning of the local vector.

Returns:

0 on success.

2.2.2.3 int parms_CommDataEnd (parms_Comm *self*)

Wait for all messages.

After calling this function, you may use data in receive buffer safely.

Parameters:

self A parms_Comm object.

Returns:

0 on success.

2.2.2.4 int parms_CommFree (parms_Comm * *self*)

Free the memory for the parms_Comm object.

Parameters:

self A pointer to the parms_Comm object.

Returns:

0 on success.

2.2.2.5 int parms_CommGetNumRecv (parms_Comm *self*)

Get the total numbjer of variables received.

Parameters:

self A communication handler.

Returns:

The number of variables received.

2.2.2.6 int parms_CommGetNumSend (parms_Comm *self*)

Get the number of variables sent.

Parameters:

self A communication handler.

Returns:

The number of vars sent.

2.2.2.7 int parms_CommGetRecvBuf (parms_Comm *self*, FLOAT $**$ *rbuf*)

Get receive buffer.

Parameters:

self A communication handler.

rbuf Receive buffer returned.

Returns:

0 on success.

2.2.2.8 int parms_CommView (parms_Comm *self*, parms_Viewer *v*)

Dump the communication handler comm.

Parameters:

self A communication handler.

v A parms_Viewer object.

Returns:

0 on success.

2.3 parms_map.h File Reference

Functions related to the parms_Map object.

`#include "parms_sys.h"`

`#include "parms_viewer.h"`

Typedefs

- typedef typedefPARMS_CXX_BEGIN struct parms_Map_ ∗ **parms_Map**

Functions

- int **parms_MapCreateFromLocal (parms_Map ∗self, int gsize, int offset)**
- **int parms_MapCreateFromGlobal (parms_Map ∗self, int gsize, int ∗npar, MPI_Comm comm, int offset, int dof, VARSTYPE vtype)**
- **int parms_MapCreateFromDist (parms_Map ∗self, int ∗vtxdist, int ∗part, MPI_Comm comm, int offset, int dof, VARSTYPE vtype)**
- **int parms_MapCreateFromPetsc (parms_Map ∗self, int m, int M, MPI_Comm comm)**
- **int parms_MapCreateFromPtr (parms_Map ∗self, int gsize, int ∗nodes, int ∗p2nodes, MPI_Comm comm, int dof, VARSTYPE vtype)**
- **int parms_MapFree (parms_Map ∗self)**
- **int parms_MapGetLocalSize (parms_Map self)**
- **int parms_MapGetGlobalSize (parms_Map self)**
- **int parms_MapGetPid (parms_Map self)**
- **int parms_MapGetNumProcs (parms_Map self)**
- **int parms_MapView (parms_Map self, parms_Viewer v)**
- **int ∗ parms_MapGlobalToLocal (parms_Map self, int gindex)**

2.3.1 Detailed Description

Functions related to the parms_Map object.

Author:

Zhongze Li

Date:

Tue Oct 17 10:02:53 2006

parms_Map describes how variables are distributed across processors. It is used for creating parms_Vec and parms_Mat objects.

2.3.2 Function Documentation

2.3.2.1 int parms_MapCreateFromDist (parms_Map ∗ *self*, int ∗ *vtxdist*, int ∗ *part*, MPI_Comm *comm*, int *offset*, int *dof*, VARSTYPE *vtype*)

Create a parms_Map object based on the output of ParMetis.

Parameters:

self A parms_Map object created.

vtxdist An integer array of size np+1, where np is the number of PEs. This array indicates the range of vertices that are local to each processor. PE i stores vertices in the range of [vtxdist[i], vtxdist[i+1]).

part An array of size equal to the number of locally-stored vertices. part[j] indicates the ID of the PE to which the vertex with local index j and global index vtxdist[pid]+j belongs (pid is ID of local PE).

comm MPI communicator.

offset The start index.

- 1 FORTRAN
- 0 C

dof The number of variables associated with each vertex.

vtype Assuming the variables u_i, v_i are associated with vertex i, two styles of numbering variables are as follows:

- INTERLACED. Variables are numbered in the order of $u_1, v_1, u_2, v_2, \cdots$;
- NONINTERLACED. Variables are numbered in the order of $u_1, u_2, u_3, ..., v_1, v_2, ...$

Returns:

0 on success.

2.3.2.2 int parms_MapCreateFromGlobal (parms_Map * *self*, int *gsize*, int * *npar*, MPI_Comm *comm*, int *offset*, int *dof*, VARSTYPE *vtype*)

Create a parms_Map object based on the Metis partitioning.

Parameters:

self A pointer to the parms_Map object created.

gsize The total number of vertices.

npar An integer array of size gsize. node i resides on PE npar[i].

comm MPI communicator.

offset The start index.

- 1 FORTRAN
- 0 C

dof The number of variables associated with each vertex.

VARSTYPE Assuming the variables u_i, v_i are associated with vertex i, two styles of numbering variables are as follows:

- INTERLACED. Variables are numbered in the order of $u_1, v_1, u_2, v_2, \cdots$;
- NONINTERLACED. Variables are numbered in the order of $u_1, u_2, u_3, ..., v_1, v_2, ...$

Returns:

0 on success.

2.3.2.3 int parms_MapCreateFromLocal (parms_Map * *self*, int *gsize*, int *offset*)

Create a parms_Map object on the local processor.

Parameters:

self A pointer to the parms_Map object created.

gsize The size of unknowns on the local processor.

offset The start index.

- 1 FORTRAN
- 0 C

Returns:

0 on success.

2.3.2.4 int parms_MapCreateFromPetsc (parms_Map * *self*, int *m*, int *M*, MPI_Comm *comm*)

Create a parms_Map object with the default partioning strategy in PETSc.

Parameters:

self A parms_Map object created.

m The local size of variables.

M The global size of variables.

comm MPI communicatior.

Returns:

0 on success.

2.3.2.5 int parms_MapCreateFromPtr (parms_Map * *self*, int *gsize*, int * *nodes*, int * *p2nodes*, MPI_Comm *comm*, int *dof*, VARSTYPE *vtype*)

Create a parms_Map object.

Parameters:

self A parms_Map object created.

gsize The total number of vertices.

nodes A list of all vertices stored PE by PE.

p2nodes An integer array of size np+1, np is the number of PEs. If k1 = p2nodes[i], k2 = p2nodes[i+1] , then PE i contains the vertices in the range of [[nodes[k1], [nodes[k2-1]].

comm MPI communication.

dof The number of variables associated with each node.

vtype Assuming the variables u_i, v_i are associated with vertex i, two style of numbering variables are as follows:

- INTERLACED. Variables are numbered in the order of $u_1, v_1, u_2, v_2, \cdots$;
- NONINTERLACED. Variables are numbered in the order of $u_1, u_2, u_3, ..., v_1, v_2, ...$

Returns:

0 on success.

2.3.2.6 int parms_MapFree (parms_Map ∗ *self*)

Free the memory for the parms_Map object pointed to by self.

Parameters:

self A pointer to the parms_Map object.

Returns:

0 on success.

2.3.2.7 int parms_MapGetGlobalSize (parms_Map *self*)

Get global size of variables.

Parameters:

self A parms_Map object.

Returns:

The global size of variables rather than vertices.

2.3.2.8 int parms_MapGetLocalSize (parms_Map *self*)

Get the size of variables on the local PE.

Parameters:

self A parms_Map object.

Returns:

The size of variables on the local PE.

2.3.2.9 int parms_MapGetNumProcs (parms_Map *self*)

Get the number of PEs.

Parameters:

self A parms_Map object.

Returns:

The number of PEs.

2.3.2.10 int parms_MapGetPid (parms_Map *self*)

Get PE's ID.

Parameters:

self A parms_Map object.

Returns:

The PE's ID.

2.3.2.11 int∗ parms_MapGlobalToLocal (parms_Map *self*, int *gindex*)

Get local index for a given global index.

Return a pointer to an integer. If it is NULL, then the variable with global index gindex doesn't reside on the local PE. Otherwise, it points to an address of a variable whose value is local index.

Parameters:

self A parms_Map object.

gindex A global index.

Returns:

A pointer to an integer whose value is the corresponding local index.

2.3.2.12 int parms_MapView (parms_Map *self*, parms_Viewer *v*)

Dump the parms_Map object.

Parameters:

self A parms_Map object.

v A parms_Viewer object.

Returns:

0 on success.

2.4 parms_mat.h File Reference

Functions related to the matrix computations.

```
#include "parms_sys.h"
#include "parms_vec.h"
#include "parms_viewer.h"
#include "parms_operator.h"
```

Typedefs

- typedef typedefPARMS_CXX_BEGIN struct parms_Mat_ * **parms_Mat**

Functions

- int **parms_MatCreate (parms_Mat *self, parms_Map map)**
- **int parms_MatView (parms_Mat self, parms_Viewer v)**
- **int parms_MatFree (parms_Mat *self)**
- **int parms_MatApply (parms_Mat self, parms_Vec x, parms_Vec y)**
- **int parms_MatVec (parms_Mat self, parms_Vec x, parms_Vec y)**
- **int parms_MatSetup (parms_Mat self)**
- **int parms_MatSetValues (parms_Mat self, int m, int *im, int *ia, int *ja, FLOAT *values, INSERTMODE mode)**
- **int parms_MatSetCommType (parms_Mat self, COMMTYPE ctype)**
- **int parms_MatSetILUType (parms_Mat self, PCILUTYPE type)**
- **int parms_MatGetDiag (parms_Mat self, void **mat)**
- **int parms_MatILU (parms_Mat self, parms_FactParam param, void *mat, parms_Operator *op)**
- **int parms_MatMVPY (parms_Mat self, REAL alpha, parms_Vec x, REAL beta, parms_Vec y, parms_Vec z)**
- **int parms_MatVecOffDiag (parms_Mat self, FLOAT *x, FLOAT *y, int pos)**
- **int parms_MatGetCommHandler (parms_Mat self, parms_Comm *handler)**
- **int parms_MatFreeSubMat (parms_Mat self, void *mat)**
- **int parms_MatGetSubMat (parms_Mat self, void **mat)**
- **int parms_MatExtend (parms_Mat self, parms_Comm handler, int start, void *mat, int *n, void **ext_mat)**
- **int parms_MatSetType (parms_Mat self, MATTYPE type)**

2.4.1 Detailed Description

Functions related to the matrix computations.

Author:

Zhongze Li

Date:

Tue Oct 17 10:05:30 2006

2.4.2 Function Documentation

2.4.2.1 int parms_MatApply (parms_Mat *self*, parms_Vec *x*, parms_Vec *y*)

Perform $y = self \times x$.

Parameters:

self A parms_Mat object.

x A parms_Vec object.

y Another parms_Vec object.

Returns:

0 on success.

2.4.2.2 int parms_MatCreate (parms_Mat * *self*, parms_Map *map*)

Create a parms_Mat object.

Create a parms_Mat object based on data distribution layout map.

Parameters:

self A pointer to the parms_Mat object created.

map A parms_Map object, which describes the data distribution among processors.

Returns:

0 on success.

2.4.2.3 int parms_MatExtend (parms_Mat *self*, parms_Comm *handler*, int *start*, void * *mat*, int * *n*, void ** *ext_mat*)

Extend submatrix by including equations correspond to the immediate neighbouring variables.

Parameters:

self A matrix object.

handler A communication handler.

start The beginning location of mat in the local matrix.

mat The submatrix to be extended.

n The size of extended matrix returned.

ext_mat The extended matrix created.

Returns:

0 on success.

2.4.2.4 int parms_MatFree (parms_Mat ∗ *self*)

Free the parms_Mat object pointed to by self.

Parameters:

self A pointer to a parms_Mat object.

Returns:

0 on success.

2.4.2.5 int parms_MatFreeSubMat (parms_Mat *self*, void ∗ *mat*)

Free the memory for the submatrix.

Parameters:

self A parms_Mat object.

mat The submatrix to be freed.

Returns:

0 on success.

2.4.2.6 int parms_MatGetCommHandler (parms_Mat *self*, parms_Comm ∗ *handler*)

Get the communication handler.

Parameters:

self A matrix object.

handler The communication handler returned.

Returns:

0 on success.

2.4.2.7 int parms_MatGetDiag (parms_Mat *self*, void ∗∗ *mat*)

Get the diagonal part of the local matrix.

Parameters:

self A parms_Mat object.

mat The diagonal part of the local matrix.

Returns:

0 on success.

2.4.2.8 int parms_MatGetSubMat (parms_Mat *self*, void ** *mat*)

Get the local matrix.

Parameters:

self A matrix object.

mat The submatrix returned in a specific format.

Returns:

0 on success.

2.4.2.9 int parms_MatILU (parms_Mat *self*, parms_FactParam *param*, void * *mat*, parms_Operator * *op*)

Perform ILU factorization.

Parameters:

self A matrix object.

param The parameters used for ILU factorization.

mat Matrix to be factored.

op The operator created.

Returns:

0 on success.

2.4.2.10 int parms_MatMVPY (parms_Mat *self*, REAL *alpha*, parms_Vec *x*, REAL *beta*, parms_Vec *y*, parms_Vec *z*)

Perform $z = alpha * self * x + beta * y$.

Parameters:

self A matrix object.

alpha A scalar.

x A vector object.

beta A scalar.

y A vector object.

z A vector stores the result.

Returns:

0 on success.

2.4.2.11 int parms_MatSetCommType (parms_Mat *self*, COMMTYPE *ctype*)

Set the communication type.

Set the communication style across processors. communication style:

- P2P point-to-point (data copied to/from auxilliary buffers).
- DERIVED derived datatype.

Parameters:

self A matrix object.

ctype Communication style:

- P2P point-to-point (data copied to/from auxilliary buffers).
- DERIVED derived datatype.

Returns:

0 on success.

2.4.2.12 int parms_MatSetILUType (parms_Mat *self*, PCILUTYPE *type*)

Set local ILU type.

Parameters:

self A matrix object.

type ILU type:

- PCILU0
- PCILUK
- PCILUT
- PCARMS

Returns:

0 on success.

2.4.2.13 int parms_MatSetType (parms_Mat *self*, MATTYPE *type*)

Set the matrix type.

Parameters:

self A matrix object.

type The matrix type:

- MAT_VCSR for VCSR format.
- MAT_CSR for CSR format.

Returns:

0 on success.

2.4.2.14 int parms_MatSetup (parms_Mat *self*)

Set up parms_Mat object self.

This is the most important functoin for the parms_Mat object. This function combines the function bdry and setup in the old version of pARMS. The function sets up the data structure needed by the distributed matrix-vector multiplication, divides the variables on the local processors into two categories: interior and interface variables.

Parameters:

self A parms_Mat object.

Returns:

0 on success.

2.4.2.15 int parms_MatSetValues (parms_Mat *self*, int *m*, int ∗ *im*, int ∗ *ia*, int ∗ *ja*, FLOAT ∗ *values*, INSERTMODE *mode*)

Insert/add values to the parms_Mat object self.

Parameters:

self A parms_Mat object.

m The number of rows inserted.

im An array of global row indices.

ia An array of pointer to the beginning of each row in array ja.

ja An array of column global indices.

values An array of values.

mode Insert value mode:

- INSERT insert values to parm_Mat self.
- ADD add values to parm_Mat self.

Returns:

0 on success.

2.4.2.16 int parms_MatVec (parms_Mat *self*, parms_Vec *x*, parms_Vec *y*)

Perform $y = self \times x$.

Parameters:

self A parms_Mat object.

x A parms_Vec object.

y Another parms_Vec object.

Returns:

0 on success.

2.4.2.17 int parms_MatVecOffDiag (parms_Mat *self*, FLOAT ∗ *x*, FLOAT ∗ *y*, int *pos*)

Perform the multiplication of the off-diagonal matrix and the external vars.

The local matrix can be written as follows:

$$\begin{pmatrix} B & E & 0 \\ F & C & M_{ext} \end{pmatrix}$$

, where $\begin{pmatrix} B & E \\ F & C \end{pmatrix}$ corresponds to the variables on the local PE. This function performs

$$y[pos..n] = M_{ext} \times x_{ext}$$

Parameters:

self A matrix object.

x A vector object.

y A vector object.

pos The offset of x from the beginning of he local vector.

Returns:

0 on success.

2.4.2.18 int parms_MatView (parms_Mat *self*, parms_Viewer *v*)

Dump parms_Mat object.

Parameters:

self A pointer to a parms_Mat object.

v A parms_Viewer object.

Returns:

0 on success.

2.5 parms_mem.h File Reference

Macros and functions for memory allocation in pARMS.

`#include <stdlib.h>`

`#include <string.h>`

`#include "parms_sys.h"`

Defines

- #define **PARMS_ALLOC**(n) parms_malloc((long)(n), __LINE__, __FILE__)
- #define **PARMS_CALLOC**(n, size) parms_calloc((long)(n), (long)(size), __LINE__-,__FILE__)
- #define **PARMS_NEW**(p) ((p) = PARMS_ALLOC(sizeof *(p)))
- #define **PARMS_NEW0**(p) ((p) = PARMS_CALLOC(1, sizeof *(p)))
- #define **PARMS_NEWARRAY**(p, n) ((p) = PARMS_ALLOC((n)*sizeof *(p)))
- #define **PARMS_NEWARRAY0**(p, n) ((p) = PARMS_CALLOC((n), sizeof *(p)))
- #define **PARMS_RESIZE**(p, size) ((p) = parms_resize((p), ((size) * (long)sizeof (*(p))), __LINE__, __FILE__))
- #define **PARMS_FREE**(p) (parms_free((p), __LINE__, __FILE__), (p) = NULL)
- #define **PARMS_MEMCPY**(dest, src, size) memcpy((dest), (src), (size)*(long)sizeof *(src))

Functions

- PARMS_CXX_BEGIN void * **parms_malloc** (long size, const int line, const char *fname)
- void * **parms_calloc** (long count, long size, int line, const char *fname)
- void * **parms_resize** (void *ptr, long size, const int line, const char *fname)
- void **parms_free** (void *ptr, const int line, const char *fname)

2.5.1 Detailed Description

Macros and functions for memory allocation in pARMS.

Author:

Zhongze Li

Date:

Tue Oct 17 11:48:51 2006

- PARMS_NEW and PARMS_NEW0 allocate memory for an object of struct type.
- PARMS_NEWARRAY and PARMS_NEWARRYA0 allocate memory for an array.
- PARMS_ALLOC(n) is used for allocating n bytes, which is useful for reducing the number of invoking malloc function. by combining multiple malloc calls together.
- PARMS_RESIZE changes the size of the allocated memory.
- PARMS_FREE frees the memory allocated.

2.6 parms_operator.h File Reference

Functions related to the preconditioner operator objects.

```
#include "parms_sys.h"
#include "parms_vec.h"
#include "parms_viewer.h"
```

Typedefs

- typedef typedefPARMS_CXX_BEGIN struct parms_Operator_ ∗ **parms_Operator**
- typedef int(∗) **opt_apply** (parms_Operator self, parms_Vec y, parms_Vec x)

Functions

- int **parms_OperatorCreate (parms_Operator ∗self)**
- **int parms_OperatorFree (parms_Operator ∗self)**
- **int parms_OperatorView (parms_Operator self, parms_Viewer v)**
- **int parms_OperatorApply (parms_Operator self, parms_Vec y, parms_Vec x)**
- **int parms_OperatorLsol (parms_Operator self, FLOAT ∗y, FLOAT ∗x)**
- **int parms_OperatorInvS (parms_Operator self, FLOAT ∗y, FLOAT ∗x)**
- **int parms_OperatorAscend (parms_Operator self, FLOAT ∗y, FLOAT ∗x)**
- **int parms_OperatorGetSchurPos (parms_Operator self)**
- **void parms_OperatorGetNnz (parms_Operator self, int ∗nnz_mat, int ∗nnz_pc)**

2.6.1 Detailed Description

Functions related to the preconditioner operator objects.

Author:

Zhongze Li

Date:

Tue Oct 17 11:51:20 2006

A operator created implicitly by invoking a ILU-type function. The corresponding destructor is set automatically when the ILU-type function is called.

2.6.2 Function Documentation

2.6.2.1 int parms_OperatorApply (parms_Operator *self*, parms_Vec *y*, parms_Vec *x*)

Perform $x = self^{-1}y$.

Assume $A \approx \begin{pmatrix} L & 0 \\ FU^{-1} & I \end{pmatrix}\begin{pmatrix} U & L^{-1}E \\ 0 & S \end{pmatrix}$, this function performs $x = \begin{pmatrix} U & L^{-1}E \\ 0 & S \end{pmatrix}^{-1}\begin{pmatrix} L & 0 \\ FU^{-1} & I \end{pmatrix}^{-1} y$.

Parameters:

self An operator.

y A right-hand-side vector.

x The solution vector.

Returns:

0 on success.

2.6.2.2 int parms_OperatorAscend (parms_Operator *self*, FLOAT * *y*, FLOAT * *x*)

Perform block backward substitution.

Assume $A \approx \begin{pmatrix} L & 0 \\ FU^{-1} & I \end{pmatrix} \begin{pmatrix} U & L^{-1}E \\ 0 & S \end{pmatrix}$, this function performs $x = \begin{pmatrix} U & L^{-1}E \\ 0 & I \end{pmatrix}^{-1} y$.

Parameters:

self An operator object.

y A right-hand-side vector.

x The solution vector.

Returns:

0 on success.

2.6.2.3 int parms_OperatorCreate (parms_Operator * *self*)

Create an operator object.

Parameters:

self A pointer to the operator object created.

Returns:

0 on success.

2.6.2.4 int parms_OperatorFree (parms_Operator * *self*)

Free the memory for the operator object pointed to by self.

Parameters:

self A pointer to the operator object.

Returns:

0 on success.

2.6.2.5 void parms_OperatorGetNnz (parms_Operator *self*, int ∗ *nnz_mat*, int ∗ *nnz_pc*)

Get the number of nonzero entries of the original matrix and the preconditioning matrix.

Parameters:

self An operator object.
nnz_mat A pointer to the number of nonzeros of the original matrix.
nnz_pc A pointer to the number of nonzeros of the preconditioning matrix.

2.6.2.6 int parms_OperatorGetSchurPos (parms_Operator *self*)

Return the start position of the Schur complement in the local matrix.

Parameters:

self An operator object.

Returns:

The beginning location of the Schur complement in the local system.

2.6.2.7 int parms_OperatorInvS (parms_Operator *self*, FLOAT ∗ *y*, FLOAT ∗ *x*)

Perform $x = S^{-1}y$.
Assume $A \approx \begin{pmatrix} L & 0 \\ FU^{-1} & I \end{pmatrix} \begin{pmatrix} U & L^{-1}E \\ 0 & S \end{pmatrix}$, this function performs $x = S^{-1}y$.

Parameters:

self An operator object.
y A right-hand-side vector.
x The solution vector.

Returns:

0 on success.

2.6.2.8 int parms_OperatorLsol (parms_Operator *self*, FLOAT ∗ *y*, FLOAT ∗ *x*)

Perform forward sweep.
Assume $A \approx \begin{pmatrix} L & 0 \\ FU^{-1} & I \end{pmatrix} \begin{pmatrix} U & L^{-1}E \\ 0 & S \end{pmatrix}$, this function performs $x = \begin{pmatrix} L & 0 \\ FU^{-1} & I \end{pmatrix}^{-1} y$.

Parameters:

self An operator object.
y A right-hand-side vector.
x The solution vector.

Returns:

0 on success.

2.6.2.9 int parms_OperatorView (parms_Operator *self*, parms_Viewer *v*)

Dump the operator object.

Output the L and U part of the operator object.

Parameters:

self An operator object.

v A parms_Viewer object.

Returns:

0 on success.

2.7 parms_pc.h File Reference

Preconditioner-related Functions.

```
#include "parms_sys.h"
#include "parms_operator.h"
#include "parms_mat.h"
#include "parms_vec.h"
#include "parms_viewer.h"
```

Typedefs

- typedef typedefPARMS_CXX_BEGIN struct parms_PC_ * **parms_PC**

Functions

- int **parms_PCSolve (parms_PC self, parms_Vec y, parms_Vec z)**
- **int parms_PCSetOP (parms_PC self, parms_Mat A)**
- **int parms_PCSetup (parms_PC self)**
- **int parms_PCCreate (parms_PC *self, parms_Mat A)**
- **int parms_PCCreateAbstract (parms_PC *self)**
- **int parms_PCFree (parms_PC *self)**
- **void parms_PCView (parms_PC self, parms_Viewer v)**
- **int parms_PCSetType (parms_PC self, PCTYPE pctype)**
- **int parms_PCSetILUType (parms_PC self, PCILUTYPE pcstype)**
- **int parms_PCSetParams (parms_PC self, int nflags, char **params)**
- **int parms_PCSetPermScalOptions (parms_PC self, int *meth, int flag)**
- **int parms_PCSetFill (parms_PC self, int *fill)**
- **int parms_PCSetTol (parms_PC self, double *tol)**
- **int parms_PCSetNlevels (parms_PC self, int nlevel)**
- **int parms_PCSetPermType (parms_PC self, int type)**
- **int parms_PCSetBsize (parms_PC self, int bsize)**
- **int parms_PCSetInnerKSize (parms_PC self, int im)**
- **int parms_PCSetInnerMaxits (parms_PC self, int imax)**
- **int parms_PCSetInnerEps (parms_PC self, REAL eps)**
- **int parms_PCSetTolInd (parms_PC self, REAL tolind)**
- **int parms_PCGetRatio (parms_PC self, double *ratio)**
- **int parms_PCGetName (parms_PC self, char **name)**
- **int parms_PCILUGetName (parms_PC self, char **iluname)**

2.7.1 Detailed Description

Preconditioner-related Functions.

Author:

Zhongze Li

Date:

Tue Oct 17 11:52:38 2006

2.7.2 Function Documentation

2.7.2.1 int parms_PCCreate (parms_PC * *self*, parms_Mat *A*)

Create a preconditioner object based on the matrix A.

Parameters:

self A preconditioner object.

A A matrix object.

Returns:

0 on success.

2.7.2.2 int parms_PCCreateAbstract (parms_PC * *self*)

Create an abstract preconditioner object.

Parameters:

self A pointer to the preconditioner object.

Returns:

0 on success.

2.7.2.3 int parms_PCFree (parms_PC * *self*)

Free the memory for the preconditioner object pointed to by self.

Parameters:

self A pointer to the memory for the preconditioner object.

Returns:

0 on success.

2.7.2.4 int parms_PCGetName (parms_PC *self*, char ** *name*)

Return the name of a preconditioner.

Parameters:

self A preconditioner.

name The name of preconditioner.

Returns:

0 on success.

2.7.2.5 int parms_PCGetRatio (parms_PC *self*, double ∗ *ratio*)

Get the ratio of the number of nonzero entries of the preconditioning matrix to that of the original matrix.

Parameters:

self A preconditioner.

ratio A pointer to the ratio.

Returns:

0 on success.

2.7.2.6 int parms_PCILUGetName (parms_PC *self*, char ∗∗ *iluname*)

Return the name of a local preconditioner.

Parameters:

self A preconditioner.

iluname The name of local ILU preconditioner.

Returns:

0 on success.

2.7.2.7 int parms_PCSetBsize (parms_PC *self*, int *bsize*)

Set the block size for ARMS.

Parameters:

self A preconditioner object.

bsize The block size for ARMS.

Returns:

0 on success.

2.7.2.8 int parms_PCSetFill (parms_PC *self*, int ∗ *fill*)

Set fill-in parameter for ILUT and ARMS.

Parameters:

self A preconditioner object.

fill A int array of size 7.

- fill[0] amount of fill-in kept in L_B.
- fill[1] amount of fill-in kept in U_B.
- fill[2] amount of fill-in kept in EL^{-1}.

- fill[3] amount of fill-in kept in $U_B^{-1}F$.
- fill[4] amount of fill-in kept in S.
- fill[5] amount of fill-in kept in L_S.
- fill[6] amount of fill-in kept in U_S.

Returns:

0 on success.

2.7.2.9 int parms_PCSetILUType (parms_PC *self*, PCILUTYPE *pcstype*)

Set local preconditioner type.

Supported ILU preconditioners:

PCILU0	ILU0
PCILUK	ILUK
PCILUT	ILUT in SPARSKIT
PCARMS	ARMS implemented by Yousef Saad

Parameters:

self A preconditioner object.

pcstype The type of local preconditioner:

- PCILU0
- PCILUK
- PCILUT
- PCARMS

Returns:

0 on success.

2.7.2.10 int parms_PCSetInnerEps (parms_PC *self*, REAL *eps*)

Set the convergence tolerance for the inner GMRES.

Parameters:

self A preconditioner object.

eps The convergence tolerance.

Returns:

0 on success.

2.7.2.11 int parms_PCSetInnerKSize (parms_PC *self*, int *im*)

Set the restart size for the inner GMRES.

Parameters:

self A preconditioner object.

im The restart size of the inner GMRES.

Returns:

0 on success.

2.7.2.12 int parms_PCSetInnerMaxits (parms_PC *self*, int *imax*)

Set the maximum iteration counts for the inner GMRES.

Parameters:

self A preconditioner object.

imax The maximum iteration counts.

Returns:

0 on success.

2.7.2.13 int parms_PCSetNlevels (parms_PC *self*, int *nlevel*)

Set the number of levels for ILUK and ARMS.

Parameters:

self A preconditioner object.

nlevel The number of levels.

Returns:

0 on success.

2.7.2.14 int parms_PCSetOP (parms_PC *self*, parms_Mat *A*)

Set the matrix to create the preconditioning matrix.

Parameters:

self A preconditioner object.

A The matrix to be used for creating PC.

Returns:

0 on success.

2.7.2.15 int parms_PCSetParams (parms_PC *self*, int *nflags*, char ** *params*)

Set parameters for the preconditioner object.

Supported parameters:

- tol drop tolerance
- fil fill-in
- nlev number of levels
- bsize block size for finding independent sets in ARMS.
- tolind drop tolerance for finding independent sets.
- iksize the restart size for the inner GMRES.
- imax the number of iterations for the inner GMRES.

Parameters:

self A preconditioner object.

nflags The number of parameters.

params A pointer to parameters.

Returns:

0 on success.

2.7.2.16 int parms_PCSetPermScalOptions (parms_PC *self*, int * *meth*, int *flag*)

Set permutation and scaling options for interlevel blocks.

Parameters:

self A preconditioner object.

meth Options:

- meth[0] nonsummetric permutations of 1: yes. affects rperm USED FOR LAST SCHUR COMPLEMENT
- meth[1] permutations of columns 0:no 1: yes. So far this is USED ONLY FOR LAST BLOCK [ILUTP instead of ILUT]. (so ipar[11] does not matter ,enter zero). If ipar[15] is one then ILUTP will be used instead of ILUT. Permutation data stored in: perm2.
  - meth[2] diag. row scaling. 0:no 1:yes. Data: D1
  - meth[3] diag. column scaling. 0:no 1:yes. Data: D2

flag Options:

- 1 interlevel block.
- 0 last block.

Returns:

0 on success.

2.7.2.17 int parms_PCSetPermType (parms_PC *self*, int *type*)

Set the type of permutation in ARMS

Parameters:

self A preconditioner object.

type Permutation type.

- 1 non-symmetric permutaion.
- 0 symmetric permutation.

Returns:

0 on success.

2.7.2.18 int parms_PCSetTol (parms_PC *self*, double $*$ *tol*)

Set the drop tolerance for ILUT preconditioner.

Parameters:

self A preconditioner object.

tol A double array of size 7.

- tol[0] threshold for dropping in L_{B}.
- tol[1] threshold for dropping in U_{B}.
- tol[2] threshold for dropping in L^{-1} F
- tol[3] threshold for dropping in E U^{-1}
- tol[4] threshold for dropping in Schur complement
- tol[5] threshold for dropping in L in last block
- tol[6] threshold for dropping in U in last block

Returns:

0 on success.

2.7.2.19 int parms_PCSetTolInd (parms_PC *self*, REAL *tolind*)

Set the tolerance for finding independent sets.

Parameters:

self A preconditioner object.

tolind The drop tolerance for finding independent sets.

Returns:

0 on success.

2.7.2.20 int parms_PCSetType (parms_PC *self*, PCTYPE *pctype*)

Set preconditioner type.

	PCBJ	block Jacobi
Currently supported preconditioners:	PCRAS	restricted additive Schwarz
	PCSCHUR	Schur complement

Parameters:

self A preconditioner object.

pctype The type of preconditioner.

- PCBJ block Jacobi
- PCRAS restricted additive Schwarz
- PCSCHUR Schur complement

Returns:

0 on success.

2.7.2.21 int parms_PCSetup (parms_PC *self*)

Set up the preconditioner (create the preconditioning matrix).

Parameters:

self A preconditioner object.

Returns:

0 on success.

2.7.2.22 int parms_PCSolve (parms_PC *self*, parms_Vec *y*, parms_Vec *z*)

Solve $selfz = y$

Parameters:

self A preconditioner object.

y A right-hand-side vector.

z The solution vector.

Returns:

0 on success.

2.7.2.23 void parms_PCView (parms_PC *self*, parms_Viewer *v*)

Dump preconditioner object self.

Parameters:

self A preconditioner object.

v A viewer object.

2.8 parms_solver.h File Reference

Functions related to the Krylov subspace methods. Only FGMRES is supported.

```
#include "parms_vec.h"
#include "parms_mat.h"
#include "parms_viewer.h"
#include "parms_pc.h"
#include "parms_operator.h"
```

Typedefs

- typedef typedefPARMS_CXX_BEGIN struct parms_Solver_ * **parms_Solver**

Functions

- int **parms_SolverSetup (parms_Solver self)**
- **int parms_SolverGetMat (parms_Solver self, parms_Mat *A)**
- **int parms_SolverGetPC (parms_Solver self, parms_PC *PC)**
- **int parms_SolverApply (parms_Solver self, parms_Vec x, parms_Vec y)**
- **int parms_SolverSetType (parms_Solver self, SOLVERTYPE stype)**
- **int parms_SolverCreate (parms_Solver *self, parms_Mat A, parms_PC pc)**
- **int parms_SolverFree (parms_Solver *self)**
- **void parms_SolverView (parms_Solver self, parms_Viewer v)**
- **void parms_SolverSetParam (parms_Solver self, PARAMTYPE ptype, char *param)**
- **int parms_SolverGetIts (parms_Solver self)**

2.8.1 Detailed Description

Functions related to the Krylov subspace methods. Only FGMRES is supported.

Author:

Zhongze Li

Date:

Tue Oct 17 11:58:22 2006

2.8.2 Function Documentation

2.8.2.1 int parms_SolverApply (parms_Solver *self*, parms_Vec *x*, parms_Vec *y*)

Solve the equation $Ax = y$.

Parameters:

self A parms_Solver object.

x The solution vector.

y The right-hand-side vector.

Returns:

0 on success.

2.8.2.2 int parms_SolverCreate (parms_Solver * *self*, parms_Mat *A*, parms_PC *pc*)

Create a parms_Solver object.

Parameters:

self A pointer to the parms_Solver object created.

A The matrix of the linear system.

pc The preconditioner.

Returns:

0 on success.

2.8.2.3 int parms_SolverFree (parms_Solver * *self*)

Free the memory for the parms_Solver object.

Parameters:

self A pointer to the parms_Solver object to be freed.

Returns:

0 on success.

2.8.2.4 int parms_SolverGetIts (parms_Solver *self*)

Get the iteration counts.

Parameters:

self A parms_Solver object.

Returns:

The iteration counts.

2.8.2.5 int parms_SolverGetMat (parms_Solver *self*, parms_Mat ∗ *A*)

Get the matrix of the linear system.

Parameters:

self A parms_Solver object.

A A pointer to the matrix returned.

Returns:

0 on success.

2.8.2.6 int parms_SolverGetPC (parms_Solver *self*, parms_PC ∗ *PC*)

Get the preconditioning matrix.

Parameters:

self A parms_Solver object.

PC A pointer to the preconditioning matrix.

Returns:

0 on success.

2.8.2.7 void parms_SolverSetParam (parms_Solver *self*, PARAMTYPE *ptype*, char ∗ *param*)

Set parameter for the solver.

Set the maximum iteration counts, the restart size of GMRES, and the convergence tolerance.

Parameters:

self A parms_Solver object.

ptype The type of parameter.

- MAXITS maximum iteration counts.
- KSIZE restart size of GMRES.
- DTOL converence tolerance.
- NEIG number of eigenvectors.

param Parameters for the solver.

2.8.2.8 int parms_SolverSetType (parms_Solver *self*, SOLVERTYPE *stype*)

Set the type of the solver.

Only FGMRES solver is available in the package.

Parameters:

self A parms_Solver object.

stype The type of Krylov subspace.

- SOLFGMRES
- SOLDGMRES

Returns:

0 on success.

2.8.2.9 int parms_SolverSetup (parms_Solver *self*)

Setup parms_Solver solver self.

Parameters:

self A parms_Solver object.

Returns:

0 on success.

2.8.2.10 void parms_SolverView (parms_Solver *self*, parms_Viewer *v*)

Dump te solver object.

Parameters:

self A parms_Solver object.

v A parms_Viewer object.

2.9 parms_sys.h File Reference

Macros and typedef needed by all other header files.

`#include <stdio.h>`

`#include "mpi.h"`

Classes

- struct **parms_FactParam**

Defines

- #define **PARMS_CXX_BEGIN**
- #define **PARMS_CXX_END**
- #define **true** 1
- #define **false** 0
- #define **PARMS_COUT** "parms_cout"
- #define **PARMS_CERR** "parms_cerr"
- #define **FINDEX** 1
- #define **ZERO** 0.0
- #define **EPSILON** 1.0e-20
- #define **EPSMAC** 1.0e-16

Typedefs

- typedef int **BOOL**

Enumerations

- enum **VARSTYPE** { **INTERLACED**, **NONINTERLACED** }
- enum **INSERTMODE** { **INSERT**, **ADD** }
- enum **COMMTYPE** { **P2P**, **DERIVED** }
- enum **SOLVERTYPE** { **SOLFGMRES**, **SOLDGMRES** }
- enum **PARAMTYPE** { **MAXITS**, **KSIZE**, **DTOL**, **NEIG** }
- enum **MATTYPE** { **MAT_NULL** = -1, **MAT_VCSR** = 0, **MAT_CSR** = 1 }
- enum **PCTYPE** { **PCBJ**, **PCSCHUR**, **PCRAS** }
- enum **PCILUTYPE** { **PCILU0**, **PCILUK**, **PCILUT**, **PCARMS** }

2.9.1 Detailed Description

Macros and typedef needed by all other header files.

Author:

Zhongze Li

Date:

Tue Oct 17 11:59:09 2006

2.10 parms_table.h File Reference

The hash table functions.

`#include "parms_sys.h"`

Typedefs

- typedef parms_Table_ ∗ **parms_Table**

Functions

- int **parms_TableCreate (parms_Table ∗newT, HashFcn hf, int size)**
- **int parms_TableFree (parms_Table ∗self)**
- **void ∗ parms_TableGet (parms_Table self, int key)**
- **int parms_TablePut (parms_Table self, int key, int val)**
- **int parms_TableGetSize (parms_Table self)**

Variables

- PARMS_CXX_BEGIN typedef unsigned int(∗) **HashFcn** (int key)

2.10.1 Detailed Description

The hash table functions.

Author:

Zhongze Li

Date:

Tue Oct 17 11:59:56 2006

2.10.2 Function Documentation

2.10.2.1 int parms_TableCreate (parms_Table ∗ *newT*, HashFcn *hf*, int *size*)

Create a hash table.

Create a hash table with hf as the hash function.

Parameters:

newT A pointer to the hash table created.

hf The hash function. If hf is null, the default hash function is used.

size The number of entries stored in the table.

Returns:

0 on success.

2.10.2.2 int parms_TableFree (parms_Table ∗ *self*)

Free the memory for the table object pointed by self.

Parameters:

self A pointer to the parms_Table object.

Returns:

0 on success.

2.10.2.3 void∗ parms_TableGet (parms_Table *self*, int *key*)

Get the corresponding value for a given key.

If return NULL, then the entry with key is not in the table, otherwise return a pointer to the value.

Parameters:

self A parms_Table object.

key The key value.

Returns:

A pointer to the value.

2.10.2.4 int parms_TableGetSize (parms_Table *self*)

Get the number of entries stored in the table.

Parameters:

self A hash table object.

Returns:

The number of entries stored in the table.

2.10.2.5 int parms_TablePut (parms_Table *self*, int *key*, int *val*)

Put the pair (key, value) into the table.

Parameters:

self A parms_Table object.

key The key of the pair.

val The value of the pair.

Returns:

0 on success.

2.11 parms_timer.h File Reference

Functions related to the parms_Timer object.

`#include "parms_sys.h"`

`#include "parms_viewer.h"`

Typedefs

- typedef typedefPARMS_CXX_BEGIN struct parms_Timer_ ∗ **parms_Timer**

Functions

- void **parms_TimerCreate (parms_Timer ∗self)**
- **int parms_TimerReset (parms_Timer self)**
- **int parms_TimerResetDelay (parms_Timer self, double delay)**
- **int parms_TimerPause (parms_Timer self)**
- **int parms_TimerRestart (parms_Timer self)**
- **double parms_TimerGet (parms_Timer self)**
- **int parms_TimerFree (parms_Timer ∗self)**
- **int parms_TimerView (parms_Timer self, parms_Viewer v)**

2.11.1 Detailed Description

Functions related to the parms_Timer object.

Author:

Zhongze Li

Date:

Tue Oct 17 12:00:25 2006

2.11.2 Function Documentation

2.11.2.1 void parms_TimerCreate (parms_Timer ∗ *self*)

Create a parms_Timer object.

Parameters:

self A pointer to the parms_Timer object created.

2.11.2.2 int parms_TimerFree (parms_Timer ∗ *self*)

Free the memory for the parms_Timer object

Parameters:

self A pointer to the parms_Timer object

Returns:

0 on success.

2.11.2.3 double parms_TimerGet (parms_Timer *self*)

Return The wall-clock time since the last call to parms_TimerReset, parms_TimerResetDelay, parms_TimerRestart.

Parameters:

self A parms_Timer object.

Returns:

The elapsed wall-clock time in seconds.

2.11.2.4 int parms_TimerPause (parms_Timer *self*)

Pause the parms_Timer object self.

Parameters:

self A parms_Timer object.

Returns:

0 on success.

2.11.2.5 int parms_TimerReset (parms_Timer *self*)

Reset the parms_Timer object self.

Parameters:

self A parms_Timer object.

Returns:

0 on success.

2.11.2.6 int parms_TimerResetDelay (parms_Timer *self*, double *delay*)

Reset the elapsed time of self to delay.

Parameters:

self A parms_Timer object.

delay Reset the elapsed time to delay seconds.

Returns:

0 on success.

2.11.2.7 int parms_TimerRestart (parms_Timer *self*)

Restart the timer.

Parameters:

self A parms_Timer object.

Returns:

0 on success.

2.11.2.8 int parms_TimerView (parms_Timer *self*, parms_Viewer *v*)

Dump parms_Timer self via parms_Viewer object v.

Parameters:

self A parms_Timer object.

v A parms_Viewer object.

Returns:

0 on success.

2.12 parms_vec.h File Reference

Functions related to the parms_Vec object.

`#include "parms_sys.h"`

`#include "parms_map.h"`

`#include "parms_viewer.h"`

`#include "parms_comm.h"`

Typedefs

- typedef typedefPARMS_CXX_BEGIN struct parms_Vec_ * **parms_Vec**

Functions

- int **parms_VecCreate (parms_Vec *self, parms_Map map)**
- **int parms_VecFree (parms_Vec *self)**
- **int parms_VecView (parms_Vec self, parms_Viewer v)**
- **int parms_VecGetArray (parms_Vec self, FLOAT **array)**
- **int parms_VecRestoreArray (parms_Vec self, FLOAT **array)**
- **int parms_VecGetNorm2 (parms_Vec self, REAL *value)**
- **int parms_VecScale (parms_Vec self, FLOAT scalar)**
- **int parms_VecGetLocalSize (parms_Vec self)**
- **int parms_VecGetGlobalSize (parms_Vec self)**
- **int parms_VecSetup (parms_Vec self)**
- **int parms_VecPut (parms_Vec self, FLOAT value)**
- **int parms_VecSetValues (parms_Vec self, int m, int *im, FLOAT *values, INSERTMODE mode)**
- **int parms_VecAXPY (parms_Vec self, parms_Vec x, FLOAT scalar)**
- **int parms_VecAYPX (parms_Vec self, parms_Vec x, FLOAT scalar)**
- **int parms_VecDOT (parms_Vec self, parms_Vec x, REAL *value)**
- **int parms_VecDotArray (parms_Vec self, int n, parms_Vec *vecarray, FLOAT *aux, FLOAT *result)**
- **int parms_VecPerm (parms_Vec self)**
- **int parms_VecAttachVec (parms_Vec self, int start, parms_Vec x)**
- **int parms_VecAttachArray (parms_Vec self, FLOAT *array)**
- **int parms_VecAlloc (parms_Vec self)**
- **int parms_VecPermAux (parms_Vec self, FLOAT *aux)**
- **int parms_VecInvPermAux (parms_Vec self, FLOAT *aux)**

2.12.1 Detailed Description

Functions related to the parms_Vec object.

Author:

Zhongze Li

Date:

Tue Oct 17 12:01:25 2006

2.12.2 Function Documentation

2.12.2.1 int parms_VecAlloc (parms_Vec *self*)

Allocate the memory space for the parms_Vec object self.

Parameters:

self A vector object.

Returns:

0 on success.

2.12.2.2 int parms_VecAttachArray (parms_Vec *self*, FLOAT $*$ *array*)

Attach vector self to the memory pointed to by array

Parameters:

self A vector

array The memory to which self is attached.

Returns:

0 on success.

2.12.2.3 int parms_VecAttachVec (parms_Vec *self*, int *start*, parms_Vec *x*)

Attach a vector self to another vector x.

Attach vector self to a chunk of vector x with beginning index start.

self[0] → x[start]

self[1] → x[start+1]

. . .

self[lsize] → x[start+lsize-1]

Parameters:

self A vector object.

start The start index of x.

x The attached vector object.

Returns:

0 on success.

2.12.2.4 int parms_VecAXPY (parms_Vec *self*, parms_Vec *x*, FLOAT *scalar*)

Perform $self := scalar \times x + self$.

Parameters:

self A vector object.

x Another vector object.

scalar A scalar.

Returns:

0 on success.

2.12.2.5 int parms_VecAYPX (parms_Vec *self*, parms_Vec *x*, FLOAT *scalar*)

Perform $self = scalar \times self + x$.

Parameters:

self A vector object.

x Another vector object.

scalar A scalar.

Returns:

0 on success.

2.12.2.6 int parms_VecCreate (parms_Vec * *self*, parms_Map *map*)

Create a parms_Vec object.

Create a parms_Vec object based on the data distribution indicated by map.

Parameters:

self A pointer to the vector created.

map A map object which describes how variables are distributed across PEs.

Returns:

0 on success.

2.12.2.7 int parms_VecDOT (parms_Vec *self*, parms_Vec *x*, REAL * *value*)

Perform the inner product of two vectors.

If self and x are real vectors, value = self x^{T}. If self and x are complex vectors, value = self {x}^{T}.

Parameters:

self A vector object.

x Another vector object.

value The inner product returned.

Returns:

0 on success.

2.12.2.8 int parms_VecDotArray (parms_Vec *self*, int *n*, parms_Vec ∗ *vecarray*, FLOAT ∗ *aux*, FLOAT ∗ *result*)

Perform the inner product between self and an array of parms_Vec objects.

The pseudo code:

```
for (i = 0; i < n; i++) { result[i] = self * vecarray[i]; }
```

Parameters:

self A vector object.

n The size of vecarray.

vecarray An array of vector objects.

aux An auxiliary array.

result An array of size n to store inner products.

Returns:

0 on success.

2.12.2.9 int parms_VecFree (parms_Vec ∗ *self*)

Free the memory for the vector object.

Parameters:

self A pointer to the vector object.

Returns:

0 on success.

2.12.2.10 int parms_VecGetArray (parms_Vec *self*, FLOAT ∗∗ *array*)

Get a pointer to a contiguous region containing the local part of the distributed vector.

Parameters:

self A vector object.

array A pointer to the local array in vector self.

Returns:

0 on success.

2.12.2.11 int parms_VecGetGlobalSize (parms_Vec *self*)

Get the global size of parms_Vec object self.

Parameters:

self A vector object.

Returns:

The global size of parms_Vec object self.

2.12.2.12 int parms_VecGetLocalSize (parms_Vec *self*)

Get the local size of parms_Vec object self.

Parameters:

self A vector object.

Returns:

The local size of parms_Vec object self.

2.12.2.13 int parms_VecGetNorm2 (parms_Vec *self*, REAL ∗ *value*)

Return the 2-norm of the vector.

Parameters:

self A vector object.

value The 2-norm returned.

Returns:

0 on success.

2.12.2.14 int parms_VecInvPermAux (parms_Vec *self*, FLOAT ∗ *aux*)

Perform inverse permutation.

self[i] = aux[perm[i]].

Parameters:

self A vector object.

aux An working array which stores permuted values.

Returns:

0 on success.

2.12.2.15 int parms_VecPerm (parms_Vec *self*)

Permute the vector object self.

If the parms_Vec object and the parms_Mat object are created based on the same parms_Map object. Once matrix object is setup, variables on each processor are divided into two cateries: internal unknowns and interface unknowns. The parms_Vec should be permuted accordingly. The user needn't call self function directly.

Parameters:

self A vector object.

Returns:

0 on success.

2.12.2.16 int parms_VecPermAux (parms_Vec *self*, FLOAT ∗ *aux*)

Perform permutation.

aux[perm[i]] = self[i].

Parameters:

self A vector object.

aux An array which stores permuted values.

Returns:

0 on success.

2.12.2.17 int parms_VecPut (parms_Vec *self*, FLOAT *value*)

Assign value to each component of the parms_Vec object self.

Parameters:

self A vector object.

value A scalar value to be set.

Returns:

0 on success.

2.12.2.18 int parms_VecRestoreArray (parms_Vec *self*, FLOAT ∗∗ *array*)

Restore values to the parms_Vec object self.

After calling this function, array cannot be used.

Parameters:

self A vector object.

array A pointer to the array of values to be restored in the parms_Vec object self.

Returns:

0 on success.

2.12.2.19 int parms_VecScale (parms_Vec *self*, FLOAT *scalar*)

Scale a vector.

All components of vector self on the local processor times scalar. $self = scalar \times self$.

Parameters:

self A vector object.

scalar A scalar.

Returns:

0 on success.

2.12.2.20 int parms_VecSetup (parms_Vec *self*)

Set up the vector object self.

This function must be called before the following parms_Vec computation functions: parms_VecGetNorm2, parms_VecGetArray, parms_VecRestoreArray, parms_VecScale, parms_VecAXPY, parms_VecAYPX, parms_VecDOT, parms_VecDotArray.

Parameters:

self A parms_Vec object.

Returns:

0 on success.

2.12.2.21 int parms_VecSetValues (parms_Vec *self*, int *m*, int * *im*, FLOAT * *values*, INSERTMODE *mode*)

Insert values to parms_Vec object self.

A pseudo code from the global point of view:

```
for (i = 0; i < m; i++) { self[im[i]] = values[i]; }
```

Parameters:

self A vector object.

m The number of variables to be inserted.

im An array of global variable indices.

value An array of values to be inserted to self.s

mode The style of set values:

- ADD add values to parms_Vec object self.
- INSERT assign values to parms_Vec object self.

Returns:

0 on success.

2.12.2.22 int parms_VecView (parms_Vec *self*, parms_Viewer *v*)

Dump the vector object.

Output the vector to the viewer v.

Parameters:

self A vector object.

v A viewer object.

Returns:

0 on success.

2.13 parms_viewer.h File Reference

Functions related to parms_Viewer objects.

`#include <stdio.h>`

`#include <string.h>`

`#include "parms_sys.h"`

Typedefs

- typedef typedefPARMS_CXX_BEGIN struct parms_Viewer_ * **parms_Viewer**

Functions

- int **parms_ViewerCreate (parms_Viewer *self, char *fname)**
- **int parms_ViewerFree (parms_Viewer *self)**
- **int parms_ViewerGetFP (parms_Viewer self, FILE **fp)**
- **int parms_ViewerStoreFP (parms_Viewer self, FILE *fp)**
- **int parms_ViewerGetFname (parms_Viewer self, char **fname)**

2.13.1 Detailed Description

Functions related to parms_Viewer objects.

Author:

Zhongze Li

Date:

Tue Oct 17 12:01:47 2006

2.13.2 Function Documentation

2.13.2.1 int parms_ViewerCreate (parms_Viewer * *self*, char * *fname*)

Create a parms_Viewer object.

If PARMS_COUT and PARMS_CERR are input as fname, they stand for standard output and standard error, respectively. Otherwise, each PE create a file "fnameID.dat". ID stands for ID of PE.

Parameters:

self A pointer to the parms_Viewer object.

fname A file name to store data.

Returns:

0 on success.

2.13.2.2 int parms_ViewerFree (parms_Viewer * *self*)

Free the memory of the parms_Viewer object.

Parameters:

self A pointer to the parms_Viewer object.

Returns:

0 on success.

2.13.2.3 int parms_ViewerGetFname (parms_Viewer *self*, char ** *fname*)

Retrieve the file name.

Parameters:

self A parms_Viewer object.

fname The file name retrieved.

Returns:

0 on success.

2.13.2.4 int parms_ViewerGetFP (parms_Viewer *self*, FILE ** *fp*)

Get a pointer to file pointer.

Parameters:

self A parms_Viewer object.

fp A pointer to the file pointer.

Returns:

0 on success.

2.13.2.5 int parms_ViewerStoreFP (parms_Viewer *self*, FILE * *fp*)

Store fp to the parms_Viewer object.

Parameters:

self A parms_Viewer object.

fp A file pointer.

Returns:

0 on success.

Index