PARMS Reference Manual

Generated by Doxygen 1.3

Sat Nov 29 14:56:48 2003

Contents

1 PARMS Data Structure Index **1**

1.1 PARMS Data Structures 1

2 PARMS File Index **3**

2.1 PARMS File List 3

3 PARMS Data Structure Documentation **5**

3.1 _p_Comm Struct Reference 5

3.2 _p_Csr Struct Reference 7

3.3 _p_CsrSpar Struct Reference 8

3.4 _p_DistMatrix Struct Reference 9

3.5 _p_DistMatrix::_p_Hash Struct Reference 10

3.6 _p_DistMatrix::_p_Hash::_p_HashOps Struct Reference 11

3.7 _p_DistMatrix::_p_Sparse_Matrix_Storage_Format Struct Reference 12

3.8 _p_DistMatrix::_p_Sparse_Matrix_Storage_Format::_p_MatOps Struct Reference . . . 13

3.9 _p_DistMatrixCsr Struct Reference 15

3.10 _p_Ext_Max Struct Reference 16

3.11 _p_ILUfac Struct Reference 17

3.12 _p_IterPar Struct Reference 18

3.13 _p_Mix_Schur Struct Reference 20

3.14 _p_MultiSch Struct Reference 21

3.15 _p_MultiSchIlu Struct Reference 22

3.16 _p_Pilu_Comm Struct Reference 23

3.18 _p_Precon Struct Reference 27

3.18 _p_Precon Struct Reference 27

3.19 _p_PrePar Struct Reference 28

3.20 _p_SchPilu Struct Reference 30

3.21 _p_Vec Struct Reference 31

3.22 _p_Vec::_p_Vec_Comm Struct Reference 33

3.23 _p_Vec::_p_Vec_Comm::_p_Vec_ComOps Struct Reference 34
3.24 Communication Struct Reference 35
3.25 ILUTfac Struct Reference 36
3.26 PerMat4 Struct Reference 38
3.27 SparRow Struct Reference 40

4 PARMS File Documentation 41

4.1 amxdis.c File Reference 41
4.2 aps.c File Reference 43
4.3 arms2.c File Reference 44
4.4 armsheads.h File Reference 46
4.5 armsol2.c File Reference 67
4.6 base.h File Reference 72
4.7 comm.c File Reference 74
4.8 data.h File Reference 76
4.9 dd-grid-edge.c File Reference 82
4.10 dd-grid-simple.c File Reference 85
4.11 dd-grid-solver.c File Reference 87
4.12 dd-grid.c File Reference 90
4.13 dd-HB-simple.c File Reference 92
4.14 dd-HB-dse.c File Reference 93
4.15 dd-HB-metis.c File Reference 94
4.16 dd-HB-parmetis.c File Reference 95
4.17 defs.h File Reference 97
4.18 dgmr.f File Reference 98
4.19 fdmat.f File Reference 102
4.20 gprecsol.c File Reference 106
4.21 hash.c File Reference 112
4.22 heads.h File Reference 114
4.23 ilu.c File Reference 115
4.24 iluNEW.c File Reference 117
4.25 ilutpC.c File Reference 118
4.26 indsetC.c File Reference 120
4.27 iters.c File Reference 122
4.28 itersf.f File Reference 124
4.29 matrix.c File Reference 125
4.30 matrops.c File Reference 128

4.31 memus.c File Reference 131
4.32 misc.f File Reference 132
4.33 piluNEW.c File Reference 134
4.34 precon.c File Reference 137
4.35 psparslib.h File Reference 144
4.36 qzhes.f File Reference 164
4.37 qzit.f File Reference 165
4.38 qzval.f File Reference 166
4.39 qzvec.f File Reference 167
4.40 schgilu0.c File Reference 168
4.41 setpar.c File Reference 169
4.42 sets.c File Reference 170
4.43 setup.c File Reference 174
4.44 skitc.c File Reference 176
4.45 skitf.f File Reference 179
4.46 solver.c File Reference 202
4.47 time.c File Reference 203
4.48 tools.f File Reference 204
4.49 uread.f File Reference 207
4.50 vec.c File Reference 208
4.51 wreadmtpar.f File Reference 211

Chapter 1

PARMS Data Structure Index

1.1 PARMS Data Structures

Here are the data structures with brief descriptions:

_p_Comm 5
_p_Csr 7
_p_CsrSpar 8
_p_DistMatrix 9
_p_DistMatrix::_p_Hash 10
_p_DistMatrix::_p_Hash::_p_HashOps 11
_p_DistMatrix::_p_Sparse_Matrix_Storage_Format 12
_p_DistMatrix::_p_Sparse_Matrix_Storage_Format::_p_MatOps 13
_p_DistMatrixCsr 15
_p_Ext_Max 16
_p_ILUfac 17
_p_IterPar 18
_p_Mix_Schur 20
_p_MultiSch 21
_p_MultiSchIlu 22
_p_Pilu_Comm 23
_p_PreCon . . . . . **??**
_p_Precon 27
_p_PrePar 28
_p_SchPilu 30
_p_Vec 31
_p_Vec::_p_Vec_Comm 33
_p_Vec::_p_Vec_Comm::_p_Vec_ComOps 34
Communication 35
ILUTfac 36
PerMat4 38
SparRow 40

Chapter 2

PARMS File Index

2.1 PARMS File List

Here is a list of all files with brief descriptions:

amxdis.c 41
aps.c 43
arms2.c 44
armsheads.h 46
armsol2.c 67
base.h 72
comm.c 74
data.h 76
dd-grid-edge.c 82
dd-grid-simple.c 85
dd-grid-solver.c 87
dd-grid.c 90
dd-HB-simple.c 92
dd-HB-dse.c 93
dd-HB-metis.c 94
dd-HB-parmetis.c 95
defs.h 97
dgmr.f 98
fdmat.f 102
gprecsol.c 106
hash.c 112
heads.h 114
ilu.c 115
iluNEW.c 117
ilutpC.c 118
indsetC.c 120
iters.c 122
itersf.f 124
matrix.c 125
matrops.c 128
memus.c 131
misc.f 132
piluNEW.c 134

precon.c 137
psparslib.h 144
qzhes.f 164
qzit.f 165
qzval.f 166
qzvec.f 167
schgilu0.c 168
setpar.c 169
sets.c 170
setup.c 174
skitc.c 176
skitf.f 179
solver.c 202
time.c 203
tools.f 204
uread.f 207
vec.c 208
wreadmtpar.f 211

Chapter 3

PARMS Data Structure Documentation

3.1 _p_Comm Struct Reference

`#include <data.h>`

Data Fields

- MPI_Comm **mpi_comm**
- int **myproc**
- int **nloc**
- int **nbnd**
- int **nl**
- int **npro**
- int **dtype_flag**
- int **nproc**
- int * **proc**
- int * **color**
- int * **ix**
- int * **ipr**
- int * **ovp**
- MPI_Datatype * **dtype**

3.1.1 Field Documentation

3.1.1.1 int* _p_Comm::color

3.1.1.2 MPI_Datatype* _p_Comm::dtype

Contains derived data type processor by processor

3.1.1.3 int _p_Comm::dtype_flag

Derived data type has been created or not 0 not created 1 created

3.1.1.4 int* _p_Comm::ipr

Pointer array. ipr[i] points to the beginning of the list of interface points that are coupled with processor proc[i].

3.1.1.5 int* _p_Comm::ix

Local interface points found which are stored processor by processor

3.1.1.6 MPI_Comm _p_Comm::mpi_comm

Communicator used in PSPARSLIB.

3.1.1.7 int _p_Comm::myproc

processor id

3.1.1.8 int _p_Comm::nbnd

The number of interior variables

3.1.1.9 int _p_Comm::nl

nloc + EXTERNAL boundary points(variables)

3.1.1.10 int _p_Comm::nloc

The number of local variables

3.1.1.11 int _p_Comm::npro

The number of processors which are involved in computation

3.1.1.12 int _p_Comm::nproc

The number of adjacent processors found

3.1.1.13 int* _p_Comm::ovp

An array. ovp[i] stores the number of processors contain local interface variable i

3.1.1.14 int* _p_Comm::proc

The list of adjacent processors found

The documentation for this struct was generated from the following file:

- **data.h**

3.2 _p_Csr Struct Reference

```
#include <data.h>
```

Data Fields

- int **n**
- int **alusize**
- double * **ma**
- int * **ja**
- int * **ia**

3.2.1 Detailed Description

CSR sparse storage format

3.2.2 Field Documentation

3.2.2.1 int _p_Csr::alusize

3.2.2.2 int * _p_Csr::ia

3.2.2.3 int* _p_Csr::ja

3.2.2.4 double* _p_Csr::ma

3.2.2.5 int _p_Csr::n

The documentation for this struct was generated from the following file:

- **data.h**

3.3 _p_CsrSpar Struct Reference

```
#include <data.h>
```

Data Fields

- **Csr smsf**
- **MatOps ops**

3.3.1 Field Documentation

3.3.1.1 MatOps _p_CsrSpar::ops

3.3.1.2 Csr _p_CsrSpar::smsf

The documentation for this struct was generated from the following file:

- **data.h**

3.4 _p_DistMatrix Struct Reference

```
#include <data.h>
```

Data Fields

- char **type** [TYPESIZE]
- **Comm comm**
- int ** **map**
- int * **perm**
- int * **node**
- **_p_DistMatrix::_p_Hash * hash**
- **_p_DistMatrix::_p_Sparse_Matrix_Storage_Format A**
- **_p_DistMatrix::_p_Sparse_Matrix_Storage_Format X**

3.4.1 Detailed Description

Distributed matrix

3.4.2 Field Documentation

3.4.2.1 struct _p_DistMatrix::_p_Sparse_Matrix_Storage_Format _p_DistMatrix::A

3.4.2.2 Comm _p_DistMatrix::comm

Define the communication struct of distributed matrix

3.4.2.3 struct _p_DistMatrix::_p_Hash * _p_DistMatrix::hash

3.4.2.4 int** _p_DistMatrix::map

map = is a pointer to a pointer map[i][0] = the number of processors contain global label i map[i][j] = the processor ids contains the global label i j>=1

3.4.2.5 int * _p_DistMatrix::node

3.4.2.6 int* _p_DistMatrix::perm

3.4.2.7 char _p_DistMatrix::type[TYPESIZE]

Indicates object type to be used to retrieve correct function pointers type = csr at current implementation

3.4.2.8 struct _p_DistMatrix::_p_Sparse_Matrix_Storage_Format _p_DistMatrix::X

The documentation for this struct was generated from the following file:

- **data.h**

3.5 _p_DistMatrix::_p_Hash Struct Reference

`#include <data.h>`

Data Fields

- int **hash_size**
- int ** **hash_table**
- **_p_DistMatrix::_p_Hash::_p_HashOps * hash_ops**

3.5.1 Detailed Description

Hash struct

3.5.2 Field Documentation

3.5.2.1 struct _p_DistMatrix::_p_Hash::_p_HashOps * _p_DistMatrix::_p_Hash::hash_ops

3.5.2.2 int _p_DistMatrix::_p_Hash::hash_size

Size of hash table

3.5.2.3 int** _p_DistMatrix::_p_Hash::hash_table

Hash Table

The documentation for this struct was generated from the following file:

- **data.h**

3.6 _p_DistMatrix::_p_Hash::_p_HashOps Struct Reference

`#include <data.h>`

Data Fields

- int(* **createhash**)(struct **_p_DistMatrix** *)
- int(* **storeinhash**)(struct **_p_DistMatrix** *, int, int)
- int(* **gethashvalue**)(struct **_p_DistMatrix** *, int)
- int(* **freehash**)(struct **_p_DistMatrix** *)
- int(* **printhash**)(struct **_p_DistMatrix** *)

3.6.1 Field Documentation

3.6.1.1 int(* _p_DistMatrix::_p_Hash::_p_HashOps::createhash)(struct _p_DistMatrix *)

Create hash table

3.6.1.2 int(* _p_DistMatrix::_p_Hash::_p_HashOps::freehash)(struct _p_DistMatrix *)

Free memory allocated for hash table

3.6.1.3 int(* _p_DistMatrix::_p_Hash::_p_HashOps::gethashvalue)(struct _p_DistMatrix *, int)

Retrieve entry from hash table

3.6.1.4 int(* _p_DistMatrix::_p_Hash::_p_HashOps::printhash)(struct _p_DistMatrix *)

Print the entry stored in hash table

3.6.1.5 int(* _p_DistMatrix::_p_Hash::_p_HashOps::storeinhash)(struct _p_DistMatrix *, int, int)

Insert entry into hash table

The documentation for this struct was generated from the following file:

- **data.h**

3.7 _p_DistMatrix::_p_Sparse_Matrix_Storage_Format Struct Reference

```
#include <data.h>
```

Data Fields

- void * **smsf**
- **_p_DistMatrix::_p_Sparse_Matrix_Storage_Format::_p_MatOps * ops**

3.7.1 Detailed Description

sparse matrix storage format and matrix operations

3.7.2 Field Documentation

3.7.2.1 struct _p_DistMatrix::_p_Sparse_Matrix_Storage_Format::_p_MatOps * _p_DistMatrix::_p_Sparse_Matrix_Storage_Format::ops

3.7.2.2 void* _p_DistMatrix::_p_Sparse_Matrix_Storage_Format::smsf

The documentation for this struct was generated from the following file:

- **data.h**

3.8 _p_DistMatrix::_p_Sparse_Matrix_Storage_Format::_p_-MatOps Struct Reference

```
#include <data.h>
```

Data Fields

- void(∗ **createmat**)(struct **_p_DistMatrix** ∗∗dm, char ∗**type**)
- void(∗ **deletemat**)(struct **_p_DistMatrix** ∗∗dm)
- void(∗ **printmat**)(struct **_p_DistMatrix** ∗dm, char ∗base)
- void(∗ **getmap**)(struct **_p_DistMatrix** ∗dm, int ∗part, int ∗p, int ∗n)
- void(∗ **copycsrtodm**)(struct **_p_DistMatrix** ∗dm, double ∗a, int ∗ja, int ∗ia)
- int(∗ **getvalofdim**)(struct **_p_DistMatrix** ∗dm)
- int(∗ **getvalofnnz**)(struct **_p_DistMatrix** ∗dm)
- void(∗ **bdry**)(struct **_p_DistMatrix** ∗)
- void(∗ **setup**)(struct **_p_DistMatrix** ∗)
- void(∗ **lamux**)(struct **_p_DistMatrix** ∗, double ∗x, double ∗y)
- void(∗ **amxdis**)(struct **_p_DistMatrix** ∗, struct **_p_Vec** ∗, struct **_p_Vec** ∗)

3.8.1 Field Documentation

3.8.1.1 void(∗ _p_DistMatrix::_p_Sparse_Matrix_Storage_Format::_p_MatOps::amxdis)(struct _p_DistMatrix ∗,struct _p_Vec ∗,struct _p_Vec ∗)

amxdis

3.8.1.2 void(∗ _p_DistMatrix::_p_Sparse_Matrix_Storage_Format::_p_MatOps::bdry)(struct _p_DistMatrix ∗)

bdry and setup seperate interior and interface variables

3.8.1.3 void(∗ _p_DistMatrix::_p_Sparse_Matrix_Storage_Format::_p_MatOps::copycsrtodm)(struct _p_DistMatrix ∗dm, double ∗a, int ∗ja, int ∗ia)

copy CSR to distributed matrix structure

3.8.1.4 void(∗ _p_DistMatrix::_p_Sparse_Matrix_Storage_Format::_p_MatOps::createmat)(struct _p_DistMatrix ∗∗dm, char ∗type)

Create and Delete distributed local matrix

3.8.1.5 void(∗ _p_DistMatrix::_p_Sparse_Matrix_Storage_Format::_p_MatOps::deletemat)(struct _p_DistMatrix ∗∗dm)

Create and Delete distributed local matrix

3.8.1.6 void(* _p_DistMatrix::_p_Sparse_Matrix_Storage_Format::_p_MatOps::getmap)(struct _p_DistMatrix *dm, int *part, int *p, int *n)

set up map from global label to processors

3.8.1.7 int(* _p_DistMatrix::_p_Sparse_Matrix_Storage_Format::_p_MatOps::getvalofdim)(struct _p_DistMatrix *dm)

get the dimension of local matrix

3.8.1.8 int(* _p_DistMatrix::_p_Sparse_Matrix_Storage_Format::_p_MatOps::getvalofnnz)(struct _p_DistMatrix *dm)

get the size of none zero elements in the matrix

3.8.1.9 void(* _p_DistMatrix::_p_Sparse_Matrix_Storage_Format::_p_MatOps::lamux)(struct _p_DistMatrix *, double *x, double *y)

amxdis

3.8.1.10 void(* _p_DistMatrix::_p_Sparse_Matrix_Storage_Format::_p_MatOps::printmat)(struct _p_DistMatrix *dm, char *base)

output distributed local matrix

3.8.1.11 void(* _p_DistMatrix::_p_Sparse_Matrix_Storage_Format::_p_MatOps::setup)(struct _p_DistMatrix *)

bdry and setup seperate interior and interface variables

The documentation for this struct was generated from the following file:

- **data.h**

3.9 _p_DistMatrixCsr Struct Reference

```
#include <data.h>
```

Data Fields

- char **type** [TYPESIZE]
- **Comm comm**
- int ** **map**
- int * **perm**
- int * **node**
- **Hash hash**
- **CsrSpar A**
- **CsrSpar X**

3.9.1 Detailed Description

Distribted Matrix structure – local matrix are stored in CSR format

3.9.2 Field Documentation

3.9.2.1 CsrSpar _p_DistMatrixCsr::A

3.9.2.2 Comm _p_DistMatrixCsr::comm

3.9.2.3 Hash _p_DistMatrixCsr::hash

3.9.2.4 int** _p_DistMatrixCsr::map

3.9.2.5 int * _p_DistMatrixCsr::node

3.9.2.6 int * _p_DistMatrixCsr::perm

3.9.2.7 char _p_DistMatrixCsr::type[TYPESIZE]

3.9.2.8 CsrSpar _p_DistMatrixCsr::X

The documentation for this struct was generated from the following file:

- **data.h**

3.10 _p_Ext_Max Struct Reference

```
#include <data.h>
```

Data Fields

- **csptr L**
- **csptr U**

3.10.1 Field Documentation

3.10.1.1 csptr _p_Ext_Max::L

3.10.1.2 csptr _p_Ext_Max::U

The documentation for this struct was generated from the following file:

- **data.h**

3.11 _p_ILUfac Struct Reference

```
#include <data.h>
```

Data Fields

- **csptr L**
- **csptr U**

3.11.1 Detailed Description

Storage format

3.11.2 Field Documentation

3.11.2.1 csptr _p_ILUfac::L

3.11.2.2 csptr _p_ILUfac::U

The documentation for this struct was generated from the following file:

- **data.h**

3.12 _p_IterPar Struct Reference

```
#include <data.h>
```

Data Fields

- int **iters**
- int **in_iters**
- int **ipar** [18]
- double **pgfpar** [2]

3.12.1 Detailed Description

Parameters for iteration

3.12.2 Field Documentation

3.12.2.1 int _p_IterPar::in_iters

the number of inner iteration

3.12.2.2 int _p_IterPar::ipar[18]

ipar[0:17] = integer array to store parameters for both arms construction (arms2) and iteration (armsol2).

ipar[0]:=nlev. number of levels (reduction processes). see also "on return" below.

ipar[1]:=bsize. Dimension of the blocks. In this version, bsize is only a target block size. The size of each block can vary and is >= bsize.

ipar[2]:=iout if (iout > 0) statistics on the run are printed to FILE *ft

The following are not used by arms2 – but should set before calling the preconditionning operation armsol2: ipar[3]:= Krylov subspace dimension for last level ipar[3] == 0 means only backward/forward solve is performed on last level. ipar[4]:= maximum # iterations on last level

ipar[5-9] NOT used [reserved for later use] - must be set to zero. [see however a special use of ipar[5] in fgmresC.]

The following set method options for arms2. Their default values can all be set to zero if desired.

ipar[10-13] == meth[0:3] = method flags for interlevel blocks ipar[14-17] == meth[0:3] = method flags for last level block - with the following meaning meth[0] permutations of rows 0:no 1: yes. affects rperm NOT USED IN THIS VERSION ** enter 0.. Data: rperm 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 no 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 similarly for meth[14], ..., meth[17] all transformations related to parametres in meth[*] (permutation, scaling,..) are applied to the matrix before processing it

3.12.2.3 int _p_IterPar::iters

the number of iterations

3.12.2.4 double _p_IterPar::pgfpar[2]

tolerance for stopping criterion. process is stopped as soon as (||.|| is the euclidean norm): || current residual|| / ||initial residual| <= eps pgfpar[0] – eps on the last level pgfpar[1] – eps on the intermediate level

The documentation for this struct was generated from the following file:

- **data.h**

3.13 _p_Mix_Schur Struct Reference

```
#include <data.h>
```

Data Fields

- **ilutptr int_max**
- **Ext_Max ext_max**

3.13.1 Field Documentation

3.13.1.1 Ext_Max _p_Mix_Schur::ext_max

3.13.1.2 ilutptr _p_Mix_Schur::int_max

The documentation for this struct was generated from the following file:

- **data.h**

3.14 _p_MultiSch Struct Reference

```
#include <data.h>
```

Data Fields

- **p4ptr levmat**
- **ilutptr ilsch**

3.14.1 Detailed Description

Data structures returned by arms. see **heads.h** for the definitions p4ptr and ilutptr

3.14.2 Field Documentation

3.14.2.1 ilutptr _p_MultiSch::ilsch

3.14.2.2 p4ptr _p_MultiSch::levmat

The documentation for this struct was generated from the following file:

- **data.h**

3.15 _p_MultiSchIlu Struct Reference

```
#include <data.h>
```

Data Fields

- **p4ptr levmat**
- **SchPilu schpilu**

3.15.1 Field Documentation

3.15.1.1 p4ptr _p_MultiSchIlu::levmat

3.15.1.2 SchPilu _p_MultiSchIlu::schpilu

The documentation for this struct was generated from the following file:

- **data.h**

3.16 _p_Pilu_Comm Struct Reference

```
#include <data.h>
```

Data Fields

- MPI_Comm **mpi_comm**
- int **myproc**
- int **nloc**
- int **nbnd**
- int **npro**
- int **nproc**
- int **ncolor**
- int **np_hcolor**
- int **np_lcolor**
- int **nrecv**
- int **tag**
- MPI_Datatype * **snddtype**
- MPI_Datatype * **shdtype**
- MPI_Datatype * **rldtype**
- MPI_Datatype * **sldtype**
- MPI_Datatype * **rhdtype**
- MPI_Status * **status**
- MPI_Request * **request**
- int * **color**
- int * **proc_hcolor**
- int * **proc_lcolor**
- int * **nodes_recv**
- int * **ix**
- int * **ipr**
- int * **proc**

3.16.1 Detailed Description

Data structure returned by gilu0 and gilut.

3.16.2 Field Documentation

3.16.2.1 int* _p_Pilu_Comm::color

An integer array of size npro contains colors assigned to processors

3.16.2.2 int* _p_Pilu_Comm::ipr

Pointer array. ipr[i] points to the beginning of the list of interface points that are coupled with processor proc[i]

3.16.2.3 int* _p_Pilu_Comm::ix

Local interface nodes which are stored processor by processor. For parallel ilu0, ix is the same as the member ix in the structure Comm, for parallel ilut, ix maybe contains more nodes.

3.16.2.4 MPI_Comm _p_Pilu_Comm::mpi_comm

3.16.2.5 int _p_Pilu_Comm::myproc

processor id

3.16.2.6 int _p_Pilu_Comm::nbnd

3.16.2.7 int _p_Pilu_Comm::ncolor

The nubmer of colors assigned to processors

3.16.2.8 int _p_Pilu_Comm::nloc

The number of processors which are involved in computation

3.16.2.9 int* _p_Pilu_Comm::nodes_recv

An integer array of size nrecv which contains external nodes received from adjacent processors. those external nodes are stored with local label in their own processor

3.16.2.10 int _p_Pilu_Comm::np_hcolor

The number of adjacent processors whose colors are higher that of this processor

3.16.2.11 int _p_Pilu_Comm::np_lcolor

The number of adjacent processors whose colors are less that of this processor

3.16.2.12 int _p_Pilu_Comm::npro

3.16.2.13 int _p_Pilu_Comm::nproc

The number of adjacent proc

3.16.2.14 int _p_Pilu_Comm::nrecv

The number of external nodes received from adjacent processors

3.16.2.15 int* _p_Pilu_Comm::proc

The list of adjacent processors which are sorted in ascending order. proc[i] is the processor ID, proc[i+nproc] contains the number of nodes received from processor proc[i]

3.16.2.16 int* _p_Pilu_Comm::proc_hcolor

An integer array contains the index of processors in proc whose colors are greater than that of this processor

3.16.2.17 int* _p_Pilu_Comm::proc_lcolor

An integer array contains the index of processors in proc whose colors are less than that of this processor

3.16.2.18 MPI_Request* _p_Pilu_Comm::request

3.16.2.19 MPI_Datatype* _p_Pilu_Comm::rhdtype

A derivated datatype array for receiving data from processors with higher colors

3.16.2.20 MPI_Datatype* _p_Pilu_Comm::rldtype

A derivated datatype array for receiving data from processors with lower colors

3.16.2.21 MPI_Datatype* _p_Pilu_Comm::shdtype

A derivated datatype array for sending data to processors with higher colors

3.16.2.22 MPI_Datatype* _p_Pilu_Comm::sldtype

A derivated datatype array for sending data to processors with lower colors

3.16.2.23 MPI_Datatype* _p_Pilu_Comm::snddtype

3.16.2.24 MPI_Status* _p_Pilu_Comm::status

3.16.2.25 int _p_Pilu_Comm::tag

The documentation for this struct was generated from the following file:

- **data.h**

3.17 _p_Precon Struct Reference

```
#include <heads.h>
```

3.17.1 Detailed Description

Preconditioner struct

The documentation for this struct was generated from the following file:

- **data.h**

3.18 _p_Precon Struct Reference

```
#include <heads.h>
```

3.18.1 Detailed Description

Preconditioner struct

The documentation for this struct was generated from the following file:

- **data.h**

3.19 _p_PrePar Struct Reference

```
#include <data.h>
```

Data Fields

- int **mc**
- int **lfil** [7]
- int **ipar** [18]
- double **droptol** [7]
- double **tolind**

3.19.1 Detailed Description

Parameters for preconditioner construction

3.19.2 Field Documentation

3.19.2.1 double _p_PrePar::droptol[7]

Threshold parameters for dropping elements in ILU factorization. droptol[0:4] = related to the multilevel block factorization droptol[5:5] = related to ILU factorization of last block. This flexibility is more than is really needed. one can use a single parameter for all. it is preferable to use one value for droptol[0:4] and another (smaller) for droptol[5:6] droptol[0] = threshold for dropping in L [B]. See **piluNEW.c**: droptol[1] = threshold for dropping in U [B]. droptol[2] = threshold for dropping in L^{-1} F droptol[3] = threshold for dropping in E U^{-1} droptol[4] = threshold for dropping in Schur complement droptol[5] = threshold for dropping in L in last block [see **ilutpC.c**] droptol[6] = threshold for dropping in U in last block [see **ilutpC.c**]

3.19.2.2 int _p_PrePar::ipar[18]

ipar[0:17] = integer array to store parameters for both arms construction (arms2) and iteration (armsol2).

ipar[0]:=nlev. number of levels (reduction processes). see also "on return" below.

ipar[1]:=bsize. Dimension of the blocks. In this version, bsize is only a target block size. The size of each block can vary and is >= bsize.

ipar[2]:=iout if (iout > 0) statistics on the run are printed to FILE *ft

The following are not used by arms2 – but should set before calling the preconditionning operation armsol2: ipar[3]:= Krylov subspace dimension for last level ipar[3] == 0 means only backward/forward solve is performed on last level. ipar[4]:= maximum # iterations on last level

ipar[5-9] NOT used [reserved for later use] - must be set to zero. [see however a special use of ipar[5] in fgmresC.]

The following set method options for arms2. Their default values can all be set to zero if desired.

ipar[10-13] == meth[0:3] = method flags for interlevel blocks ipar[14-17] == meth[0:3] = method flags for last level block - with the following meaning meth[0] permutations of rows 0:no 1: yes. affects rperm NOT USED IN THIS VERSION ** enter 0.. Data: rperm 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 no 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 similarly for meth[14], ..., meth[17] all transformations related to parametres in meth[*] (permutation, scaling,..) are applied to the matrix before processing it

3.19.2.3 int _p_PrePar::lfil[7]

lfil[0:6] is an array containing the fill-in parameters. similar explanations as above, namely: lfil[0] = amount of fill-in kept in L [B]. lfil[1] = amount of fill-in kept in U [B]. etc..

3.19.2.4 int _p_PrePar::mc

multi color the domains or not 1 multi-coloring; 0 non-multi-coloring

3.19.2.5 double _p_PrePar::tolind

Tolerance parameter used by the indset function. a row is not accepted into the independent set if the *relative* diagonal tolerance is below tolind. see indset function for details. Good values are between 0.05 and 0.5 – larger values tend to be better for harder problems.

The documentation for this struct was generated from the following file:

- **data.h**

3.20 _p_SchPilu Struct Reference

```
#include <data.h>
```

Data Fields

- **Pilu_Comm pcomm**
- **Mix_Schur mschur**

3.20.1 Field Documentation

3.20.1.1 Mix_Schur _p_SchPilu::mschur

3.20.1.2 Pilu_Comm _p_SchPilu::pcomm

The documentation for this struct was generated from the following file:

- **data.h**

3.21 _p_Vec Struct Reference

`#include <data.h>`

Data Fields

- **_p_Vec::_p_Vec_Comm** ∗ **vec_comm**
- double ∗ **vec**
- double ∗ **alias**
- double ∗ **ext_node**
- MPI_Request ∗ **request**
- int ∗ **perm**
- int ∗ **node**

3.21.1 Detailed Description

Define operations on vector

3.21.2 Field Documentation

3.21.2.1 double∗ _p_Vec::alias

The back-up pointer to the memory allocated when creating Vec object

3.21.2.2 double∗ _p_Vec::ext_node

Nodes received from adjacent processors

3.21.2.3 int∗ _p_Vec::node

Local label i (after permutation) corresponds global label node[i]

3.21.2.4 int∗ _p_Vec::perm

Permutation array local label perm[i] is mapped to new local label i after permutation (that is interior followed by interface nodes)

3.21.2.5 MPI_Request∗ _p_Vec::request

Communication request (handle)

3.21.2.6 double∗ _p_Vec::vec

The vector

3.21.2.7 struct _p_Vec::_p_Vec_Comm * _p_Vec::vec_comm

The documentation for this struct was generated from the following file:

- **data.h**

3.22 _p_Vec::_p_Vec_Comm Struct Reference

```
#include <data.h>
```

Data Fields

- **Comm comm**
- **_p_Vec::_p_Vec_Comm::_p_Vec_ComOps * vec_comops**

3.22.1 Detailed Description

Communication structure

3.22.2 Field Documentation

3.22.2.1 Comm _p_Vec::_p_Vec_Comm::comm

3.22.2.2 struct _p_Vec::_p_Vec_Comm::_p_Vec_ComOps * _p_Vec::_p_Vec_Comm::vec_comops

The documentation for this struct was generated from the following file:

- **data.h**

3.23 _p_Vec::_p_Vec_Comm::_p_Vec_ComOps Struct Reference

`#include <data.h>`

Data Fields

- void(* **msg_bdx_bsend**)(struct **_p_Vec** *)
- void(* **mtype_create**)(struct **_p_Vec** *)
- void(* **mtype_free**)(struct **_p_Vec** *)

3.23.1 Detailed Description

Communication functions related to vectors msg_bdx_bsend send and receive data mtype_create create derived data type mtype_free free derived data type

3.23.2 Field Documentation

3.23.2.1 void(* _p_Vec::_p_Vec_Comm::_p_Vec_ComOps::msg_bdx_bsend)(struct _p_Vec *)

3.23.2.2 void(* _p_Vec::_p_Vec_Comm::_p_Vec_ComOps::mtype_create)(struct _p_Vec *)

3.23.2.3 void(* _p_Vec::_p_Vec_Comm::_p_Vec_ComOps::mtype_free)(struct _p_Vec *)

The documentation for this struct was generated from the following file:

- **data.h**

3.24 Communication Struct Reference

The documentation for this struct was generated from the following file:

- **data.h**

3.25 ILUTfac Struct Reference

```
#include <heads.h>
```

Data Fields

- int **n**
- **SparRow** ∗ **C**
- **SparRow** ∗ **L**
- **SparRow** ∗ **U**
- int **meth** [4]
- int ∗ **rperm**
- int ∗ **perm2**
- double ∗ **D1**
- double ∗ **D2**
- double ∗ **wk**

3.25.1 Detailed Description

struct for storing data related to the last schur complement we need to store the C matrix associated with the last block and the ILUT factorization of the related Schur complement.

3.25.2 Field Documentation

3.25.2.1 struct SparRow∗ ILUTfac::C

C matrix of last block

3.25.2.2 double∗ ILUTfac::D1

diagonal matrices used for scaling if scaling option is turned on

3.25.2.3 double∗ ILUTfac::D2

diagonal matrices used for scaling if scaling option is turned on

3.25.2.4 struct SparRow∗ ILUTfac::L

LU factorization

3.25.2.5 int ILUTfac::meth[4]

related to variants and methods

3.25.2.6 int ILUTfac::n

size of C block

3.25.2.7 int* ILUTfac::perm2

symmetric permutation used in factorization comes from independent set ordering

3.25.2.8 int* ILUTfac::rperm

unsymmetric permutation not used in this version...but left for compatibility

3.25.2.9 struct SparRow* ILUTfac::U

LU factorization

3.25.2.10 double* ILUTfac::wk

work vector of length n needed for various tasks

The documentation for this struct was generated from the following file:

- **heads.h**

3.26 PerMat4 Struct Reference

```
#include <heads.h>
```

Data Fields

- int **n**
- int **nB**
- **SparRow** ∗ **L**
- **SparRow** ∗ **U**
- **SparRow** ∗ **E**
- **SparRow** ∗ **F**
- int **meth** [4]
- int ∗ **rperm**
- int ∗ **perm**
- double ∗ **D1**
- double ∗ **D2**
- double ∗ **wk**
- **p4ptr prev**
- **p4ptr next**

3.26.1 Detailed Description

struct for storing the block LU factorization contains all the block factors except the data related to the last block.

3.26.2 Field Documentation

3.26.2.1 double∗ PerMat4::D1

diagonal matrices used for scaling if scaling option is turned on

3.26.2.2 double∗ PerMat4::D2

diagonal matrices used for scaling if scaling option is turned on

3.26.2.3 struct SparRow∗ PerMat4::E

sparse matrices in (1,2) and (2,1) parts of matrix

3.26.2.4 struct SparRow∗ PerMat4::F

sparse matrices in (1,2) and (2,1) parts of matrix

3.26.2.5 struct SparRow∗ PerMat4::L

ILU factors of B-block

3.26.2.6 int PerMat4::meth[4]

3.26.2.7 int PerMat4::n

Size of current block

3.26.2.8 int PerMat4::nB

Size of B-block

3.26.2.9 p4ptr PerMat4::next

pointer to next struct

3.26.2.10 int* PerMat4::perm

symmetric permutation used in factorization comes from independent set ordering

3.26.2.11 p4ptr PerMat4::prev

pointer to previous struct

3.26.2.12 int* PerMat4::rperm

unsymmetric permutation not used in this version...but left for compatibility

3.26.2.13 struct SparRow* PerMat4::U

ILU factors of B-block

3.26.2.14 double* PerMat4::wk

work vector of length n needed for various tasks

The documentation for this struct was generated from the following file:

- **heads.h**

3.27 SparRow Struct Reference

`#include <heads.h>`

Data Fields

- int **n**
- int * **nnzrow**
- double ** **ma**
- int ** **ja**

3.27.1 Detailed Description

C-style CSR format - used internally for all matrices in CSR format

3.27.2 Field Documentation

3.27.2.1 int** SparRow::ja

pointer-to-pointer to store column indices

3.27.2.2 double** SparRow::ma

pointer-to-pointer to store nonzero entries

3.27.2.3 int SparRow::n

3.27.2.4 int* SparRow::nnzrow

Length of each row

The documentation for this struct was generated from the following file:

- **heads.h**

Chapter 4

PARMS File Documentation

4.1 amxdis.c File Reference

```
#include "../../INCLUDE/psparslib.h"
```

Functions

- void **amux** (**DistMatrix** dm, double ∗x, double ∗y)
- void **amux1** (**DistMatrix** dm, double ∗x, double ∗y)
- void **amuxe** (**DistMatrix** dm, double ∗x, double ∗y)
- void **amxdis** (**DistMatrix** dm, **Vec** x, **Vec** y)

4.1.1 Function Documentation

4.1.1.1 void amux (DistMatrix *dm*, double ∗ *x*, double ∗ *y*)

Dot-product version of local matrix-vector product

Distributed matrix and vector product for CSR format

y := dm∗x

ON ENTRY :

dm = distributed local matrix handler

x = local vector

ON RETURN :

y = output local vector

4.1.1.2 void amux1 (DistMatrix *dm*, double ∗ *x*, double ∗ *y*)

axpy version of local matarix-vector product (for transposed)

Distributed matrix and vector product for CSR format

y = y + dm∗x

ON ENTRY :

dm = distributed local matrix handler

x = external nodes

ON RETURN :

y = output local vector

4.1.1.3 void amuxe (DistMatrix *dm*, double * *x*, double * *y*)

Dot-product version of local RECTANGULAR matrix-vector product

Distributed external matrix and vector product for CSR format

y = dm*x

ON ENTRY :

dm = distributed local matrix handler

x = external nodes

ON RETURN :

y = output local vector

4.1.1.4 void amxdis (DistMatrix *dm*, Vec *x*, Vec *y*)

Dot-product version of distributed matrix-vector product.

Distributed matrix and vector product for CSR format

y := dm*x

ON ENTRY :

dm = distributed local matrix handler

x = input local vector

ON RETURN :

y = output local vector

4.2 aps.c File Reference

```
#include "../../INCLUDE/psparslib.h"
```

Functions

- int **lsch** (**DistMatrix** dm, **PreCon** precon, **IterPar** iterpar, **Vec** rhs, **Vec** sol)
- int **rsch** (**DistMatrix** dm, **PreCon** precon, **IterPar** iterpar, **Vec** rhs, **Vec** sol)

4.2.1 Function Documentation

4.2.1.1 int lsch (DistMatrix *dm*, PreCon *precon*, IterPar *iterpar*, Vec *rhs*, Vec *sol*)

Approximate LU-SCHUR left preconditioner

This is a preconditioner for the global linear system which is based on solving approximately the Schur complement system.

More precisely, an approximation to the local Schur complement is obtained (implicitly) in the form of an LU factorization from the LU factorization L_i U_i of the local matrix A_i. Solving with this approximation amounts to simply doing the forward and backward solves with the bottom part of L_i and U_i only (corresponding to the interace variables). This is done using a special version of the subroutine lusol0 called lusol0_p.

Then it is possible to solve for an approximate Schur complement system using GMRES on the global approximate Schur complement system (which is preconditionied by the diagonal blocks represented by these restricted LU matrices).

4.2.1.2 int rsch (DistMatrix *dm*, PreCon *precon*, IterPar *iterpar*, Vec *rhs*, Vec *sol*)

Approximate LU-SCHUR left preconditioner

This is a preconditioner for the global linear system which is based on solving approximately the Schur complement system.

More precisely, an approximation to the local Schur complement is obtained (implicitly) in the form of an LU factorization from the LU factorization L_i U_i of the local matrix A_i. Solving with this approximation amounts to simply doing the forward and backward solves with the bottom part of L_i and U_i only (corresponding to the interace variables). This is done using a special version of the subroutine lusol0 called lusol0_p.

Then it is possible to solve for an approximate Schur complement system using GMRES on the global approximate Schur complement system (which is preconditionied by the diagonal blocks represented by these restricted LU matrices).

4.3 arms2.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../../INCLUDE/psparslib.h"
```

Defines

- #define **PERMTOL** 0.2
- #define **MBLOC** 5
- #define **OPTION** 0

Functions

- int **arms2** (**DistMatrix** dm, **PreCon** precon, **PrePar** prepar)

4.3.1 Define Documentation

4.3.1.1 #define MBLOC 5

4.3.1.2 #define OPTION 0

4.3.1.3 #define PERMTOL 0.2

4.3.2 Function Documentation

4.3.2.1 int arms2 (DistMatrix *dm*, PreCon *precon*, PrePar *prepar*)

MULTI-LEVEL BLOCK ILUT PRECONDITIONER.

ON ENTRY :

dm = distributed matrix handler

precon = a pointer to struct **_p_PreCon** containg preconditioner

prepar = a pointer to struct **_p_PrePar** containing parameters for preconditioner.

Main parameters in prepar are ipar, lfil, ipar, droptol and tolind.

ipar[0:17] = integer array to store parameters for both arms construction (arms2) and iteration (armsol2).

ipar[0]:=nlev. number of levels (reduction processes). see also "on return" below.

ipar[1]:=bsize. Dimension of the blocks. In this version, bsize is only a target block size. The size of each block can vary and is >= bsize.

ipar[2]:=iout if (iout > 0) statistics on the run are printed to FILE *ft

The following are not used by arms2 – but should set before calling the preconditionning operation armsol2: ipar[3]:= Krylov subspace dimension for last level ipar[3] == 0 means only backward/forward solve is performed on last level.

ipar[4]:= maximum # iterations on last level

ipar[5-9] NOT used [reserved for later use] - must be set to zero. [see however a special use of ipar[5] in fgmresC.]

The following set method options for arms2. Their default values can all be set to zero if desired.

ipar[10-13] == meth[0:3] = method flags for interlevel blocks

ipar[14-17] == meth[0:3] = method flags for last level block

- with the following meaning

meth[0] permutations of rows 0:no 1: yes. affects rperm NOT USED IN THIS VERSION ** enter 0.. Data: rperm

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 no 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 similarly for meth[14], ..., meth[17] all transformations related to parametres in meth[*] (permutation, scaling,..) are applied to the matrix before processing it

droptol = Threshold parameters for dropping elements in ILU factorization.

droptol[0:4] = related to the multilevel block factorization

droptol[5:5] = related to ILU factorization of last block. This flexibility is more than is really needed. one can use a single parameter for all. it is preferable to use one value for droptol[0:4] and another (smaller) for droptol[5:6] droptol[0] = threshold for dropping in L [B]. See **piluNEW.c**:

droptol[1] = threshold for dropping in U [B].

droptol[2] = threshold for dropping in L^{-1} F

droptol[3] = threshold for dropping in E U^{-1}

droptol[4] = threshold for dropping in Schur complement

droptol[5] = threshold for dropping in L in last block [see **ilutpC.c**] droptol[6] = threshold for dropping in U in last block [see **ilutpC.c**]

lfil = lfil[0:6] is an array containing the fill-in parameters. similar explanations as above, namely:

lfil[0] = amount of fill-in kept in L [B].

lfil[1] = amount of fill-in kept in U [B]. etc..

tolind = tolerance parameter used by the indset function. a row is not accepted into the independent set if the *relative* diagonal tolerance is below tolind. see indset function for details. Good values are between 0.05 and 0.5 – larger values tend to be better for harder problems.

4.4 armsheads.h File Reference

```
#include "data.h"
#include "base.h"
```

Functions

- int **setupCS** (**csptr**, int)
- int **cleanCS** (**csptr**)
- int **ilutpC** (**csptr**, double ∗, int ∗, double, int, **ilutptr**)
- int **setupILUT** (**ilutptr**, int)
- int **setupBLU** (**p4ptr**, int, int, **csptr**, **csptr**)
- int **csSplit4** (**csptr** amat, int bsize, int csize, **csptr** B, **csptr** F, **csptr** E, **csptr** C)
- int **CSRcs** (int n, double ∗ma, int ∗ja, int ∗ia, **csptr** bmat)
- int **cscpy** (**csptr** amat, **csptr** bmat)
- int **SparTran** (**csptr** amat, **csptr** bmat, int job, int flag)
- int **csPer** (**csptr**, int, int, **p4ptr**)
- int **copymat** (**csptr**, **csptr**)
- int **copymat2** (**csptr**, **csptr**)
- int **indsetC** (**csptr** mat, int bsize, int ∗iord, int ∗nnod, double tol, int nbnd)
- int **levset** (**csptr**, **csptr**, int, int ∗, double ∗, double)
- int **dpermC** (**csptr** mat, int ∗perm)
- void **all4mat** (**p4ptr**, int)
- void **plSpar** (**csptr**, int)
- void **sparmat** (**csptr**, char ∗)
- void **pILUTmat** (**ilutptr**)
- void **plperm** (**p4ptr**, int)
- int **pilu** (**p4ptr**, **csptr**, **csptr**, double ∗, int ∗, **csptr**)
- int **ilutD** (**csptr**, double ∗, int ∗, **ilutptr**)
- int **roscalC** (**csptr**, double ∗, int)
- int **coscalC** (**csptr**, double ∗, int)
- int **armsol2** (int type, double ∗b, double ∗x, **p4ptr** levmat, **ilutptr** ilusch, double ∗pgfpar, int ∗ipar, int flag, FILE ∗fp)
- int **armschsol** (double ∗b, double ∗x, **p4ptr** levmat, **Csr** bmat, **SchPilu** schpilu, **IterPar** ipar, FILE ∗fp)
- int **add2is** (int ∗last, int nod, int ∗iord, int ∗riord)
- int **add2com** (int ∗nback, int nod, int ∗iord, int ∗riord)
- int **weightsC** (**csptr** mat, double ∗w)
- int **cs_nnz** (**csptr** A)
- int **lev4_nnz** (**p4ptr** levmat, int ∗lev, FILE ∗ft)
- int **nnz_arms** (**p4ptr** levmat, **ilutptr** ilschu, int nlev, FILE ∗ft)
- int **nnz_arms1** (**p4ptr** levmat, **ilutptr** ilschu, FILE ∗ft)
- **qsplitC** (double ∗a, int ∗ind, int n, int ncut)
- int **cleanP4** (**p4ptr** amat)
- int **cleanILUT** (**ilutptr** amat, int indic)
- int **cleanARMS** (**p4ptr** amat, **ilutptr** cmat)
- int **rpermC** (**csptr** mat, int ∗perm)
- int **cpermC** (**csptr** mat, int ∗perm)
- void **swapj** (int v[], int i, int j)

- void **swapm** (double v[], int i, int j)
- void **qsortC** (int *ja, double *ma, int left, int right, int abval)
- void **printmat** (FILE *ft, **csptr** A, int i0, int i1)
- void **Lsol** (**csptr** mata, double *b, double *x)
- void **Usol** (**csptr** mata, double *b, double *x)
- void **matvec** (**csptr** mata, double *x, double *y)
- void **matvecz** (**csptr** mata, double *x, double *y, double *z)
- void **dscale** (int, double *, double *, double *)
- int **descend** (**p4ptr** levmat, double *x, double *wk)
- int **ascend** (**p4ptr** levmat, double *wk, double *x)
- void **lusolD** (**ilutptr** ilusch, double *y, double *x)
- int **schurprod** (**p4ptr** levmat, **ilutptr** Smat, double *x, double *y)
- int **dlusol** (**SchPilu** schpilu, double *y, double *x)
- void **giluLsol** (**SchPilu** schpilu, double *b, double *y)
- void **giluUsol** (**SchPilu** schpilu, double *y, double *x)
- void **Lsolp** (int start, int n, **csptr** mata, double *b, double *y)
- void **Usolp** (int start, int n, **csptr** mata, double *y, double *x)
- void **lusolDp** (int start, int rflag, **ilutptr** ilusch, double *y, double *x)
- int **pgmr** (double *rhs, double *sol, **p4ptr** levmat, **ilutptr** ilusch, double *pgfpar, int *ipar, FILE *fp)
- int **dpgmr** (double *rhs, double *sol, **p4ptr** levmat, **Csr** bmat, **SchPilu** schpilu, **IterPar** ipar, FILE *fp)
- int **schuramux** (**p4ptr** levmat, **ilutptr** Smat, **Csr** bmat, int nbnd, double *x, double *ww, double *y)
- void **exbound** (**SchPilu** schpilu, double *x, double *w)
- int **schsgssol** (double *b, double *x, **p4ptr** levmat, **Csr** bmat, **SchPilu** schpilu, **IterPar** ipar, FILE *fp)
- void **sgslusol** (**SchPilu** schpilu, double *wk1, double *wk2)
- int **sgsschur** (double *rhs, double *sol, **p4ptr** levmat, **Csr** bmat, **SchPilu** schpilu, **IterPar** ipar, FILE *fp)
- int **schpart** (**csptr** schur, **Csr** xmat, **SchPilu** schpilu)
- int **gilupgmr** (double *rhs, double *sol, **p4ptr** levmat, **Csr** bmat, **SchPilu** schpilu, **IterPar** ipar, FILE *fp)
- int **sgspgmr** (double *rhs, double *sol, **p4ptr** levmat, **Csr** bmat, **SchPilu** schpilu, **IterPar** ipar, FILE *fp)

4.4.1 Function Documentation

4.4.1.1 int add2com (int * *nback*, int *nod*, int * *iord*, int * *riord*)

Adds element nod to independent set

4.4.1.2 int add2is (int * *last*, int *nod*, int * *iord*, int * *riord*)

Adds element nod to independent set

4.4.1.3 void all4mat (p4ptr, int)

4.4.1.4 int armschsol (double * *b*, double * *x*, p4ptr *levmat*, Csr *bmat*, SchPilu *schpilu*, IterPar *ipar*, FILE * *fp*)

4.4.1.5 int armsol2 (int *type*, double * *b*, double * *x*, p4ptr *levmat*, ilutptr *ilusch*, double * *pgfpar*, int * *ipar*, int *flag*, FILE * *fp*)

Solve subroutine used in preconditioning operation associated with the ARMS multi-level ilu preconditioner – It solves M x = b where

| L 0 | | U L^{-1}F |

M = | | | |

| EU^{-1} I | | 0 S |

M is the multi-level block ILUT preconditioner constructed by arms2 revised by Zhongze Li, Apr. 16th, 2001

ON ENTRY :

b = double array of length n, the right-hand side of M x = b. unchanged on return.

(levmat) = permuted and sorted matrices for each level stored in **PerMat4** struct.

(ilschu) = matrix stored in IluSpar struct containing the ILU decomposition of the last level.

pgfpar[0] = tolerance for convergence criterion for last level iteration.. [when applicable, i.e., when ipar[3]>0 see below] –

ipar[0:17] = integer array to store parameters for both arms construction (arms2) and iteration (armsol2).:

ipar[0]:=nlev. number of levels.

ipar[1]:=bsize. Not used here [used in arms2]

ipar[2]:=iout if (iout > 0) statistics on the run are printed to FILE *ft

ipar[3]:= Krylov subspace dimension for last level ipar[3] == 0 means only backward/forward solve is performed on last level. |

ipar[4]:= maximum # iterations on last level

ipar[5-9] NOT used [reserved for later use] - must be set to zero. [see however a special use of ipar[5] in fgmresC.]

ipar[10-13] == meth[0:3] = method flags for interlevel blocks

ipar[14-17] == meth[0:3] = method flags for last level block - with the following meaning:

meth[0] permutations of rows 0:no 1: yes. affects rperm NOT USED IN THIS VERSION ** enter 0.. Data: rperm

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 no matter - enter zero; but 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 similarly for meth[14], ..., meth[17] all transformations related to parametres in meth[*] (permutation, scaling,..) are applied to the matrix before processing it flag indicates which part is solved 0 – the whole linear system is solved, that is U^{-1}L^{-1}rhs 1 – the interface nodes is solved, that is U_S^{-1}L^{-1}rhs, U_S stands for the U part of the interface linear system

ON RETURN :

x = double vector of length n, store the preconditioned residual, i.e., the solution of arms x = b.

4.4.1.6 int ascend (p4ptr *levmat*, double * *wk*, double * *x*)

This function does the (block) backward substitution:

| | | | | |

| U L^{-1}F | | x1 | | wk1 |

| | | | = | |

| 0 S | | x2 | | wk2 |

| | | | | |

with x2 = S^{-1} wk2 [assumed to have been computed]

4.4.1.7 int cleanARMS (p4ptr *amat*, ilutptr *cmat*)

Free up memory allocated for entire ARMS preconditioner.

ON ENTRY :

(amat) = Pointer to a Per4Mat struct.

(cmat) = Pointer to a IluSpar struct.

4.4.1.8 int cleanCS (csptr *amat*)

Free up memory allocated for **SparRow** structs.

ON ENTRY :

(amat) = Pointer to a **SparRow** struct.

len = size of matrix

4.4.1.9 int cleanILUT (ilutptr *amat*, int *indic*)

Free up memory allocated for IluSpar structs.

ON ENTRY :

(amat) = Pointer to a IluSpar struct.

indic = indicator for number of levels. indic=0 -> zero level

4.4.1.10 int cleanP4 (p4ptr *amat*)

Free up memory allocated for Per4Mat structs.

ON ENTRY :

(amat) = Pointer to a Per4Mat struct

4.4.1.11 int copymat (csptr, csptr)

4.4.1.12 int copymat2 (csptr, csptr)

4.4.1.13 int coscalC (csptr *mata*, double * *diag*, int *nrm*)

This routine scales each column of mata so that the norm is 1.

ON ENTRY :

mata = the matrix (in **SparRow** form)

nrm = type of norm

0 (infty), 1 or 2

ON RETURN :

diag = diag[j] = 1/norm(row[j])

0 –> normal return

j –> column j is a zero column

4.4.1.14 int cpermC (csptr *mat*, int * *perm*)

This subroutine permutes the columns of a matrix in **SparRow** format. cperm computes B = A P, where P is a permutation matrix. that maps column j into column perm(j), i.e., on return The permutation P is defined through the array perm: for each j, perm[j] represents the destination column number of column number j.

ON ENTRY :

(mat) = a matrix stored in **SparRow** format.

ON RETURN :

(mat) = A P stored in **SparRow** format.

INTEGER VALUE RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.4.1.15 int cs_nnz (csptr *A*)

Counts the number of nonzero elements in CSR matrix A

4.4.1.16 int cscpy (csptr *amat*, csptr *bmat*)

Copy amat in **SparRow** struct to bmat in **SparRow** struct

ON ENTRY :

(amat) = Matrix stored in **SparRow** format

ON RETURN :

(bmat) = Matrix stored as **SparRow** struct containing a copy of amat

INTEGER VALUE RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.4.1.17 int csPer (csptr, int, int, p4ptr)

4.4.1.18 int CSRcs (int *n*, double * *a*, int * *ja*, int * *ia*, csptr *bmat*)

Convert CSR matrix to **SparRow** struct

ON ENTRY :

a, ja, ia = Matrix stored in CSR format (with FORTRAN indexing).

ON RETURN :

(bmat) = Matrix stored as **SparRow** struct.

INTEGER VALUE RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.4.1.19 int csSplit4 (csptr *amat*, int *bsize*, int *csize*, csptr *B*, csptr *F*, csptr *E*, csptr *C*)

Convert permuted csrmat struct to **PerMat4** struct

- matrix already permuted

ON ENTRY :

(amat) = Matrix stored in **SparRow** format. Internal pointers (and associated memory) destroyed before return.

ON RETURN :

B, E, F, C = 4 blocks in

| B F |

Amat = | |

| E C |

INTEGER VALUE RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.4.1.20 int descend (p4ptr *levmat*, double * *x*, double * *wk*)

This function does the (block) forward elimination in ARMS

| | | | | |

| L 0 | | wk1 | | x1 |

| | | | = | |

| EU^{-1} I | | wk2 | | x2 |

| | | | | |

wk is permuted

4.4.1.21 int dlusol (SchPilu *schpilu*, double * *y*, double * *x*)

This routine solves the system (LU) x = y, given an LU decomposition of a matrix stored in (ilusch) in modified sparse row format

ON ENTRY :

y = the right-hand-side vector

schpilu = the LU matrix as provided from the ILU routines.

ON RETURN :

x = solution of LU x = y.

meth[0] rperm = 0:none 1:row (CTC)

meth[1] perm2 = 0:none 1:column (ILUTP)

meth[2] D1 = 0:none 1:scaling

meth[3] D2 = 0:none 1:scaling

NOTE :

Routine is in place: call lusolD(ilusch, x, x) will solve the system with rhs x and overwrite the result on x.

4.4.1.22 int dpermC (csptr *mat*, int * *perm*)

This subroutine permutes the rows and columns of a matrix in **SparRow** format. dperm computes B = P^T A P, where P is a permutation matrix.

ON ENTRY :

(amat) = a matrix stored in **SparRow** format.

ON RETURN :

(amat) = P^T A P stored in **SparRow** format.

INTEGER VALUE RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.4.1.23 int dpgmr (double * *rhs*, double * *sol*, p4ptr *levmat*, Csr *bmat*, SchPilu *schpilu*, IterPar *ipar*, FILE * *fp*)

4.4.1.24 void dscale (int *n*, double * *dd*, double * *x*, double * *y*)

Computes y == DD * x scales the vector x by the diagonal dd - output in y

4.4.1.25 void exbound (SchPilu *schpilu*, double * *x*, double * *w*)

exchange boundary data in nonblocking mode */

4.4.1.26 void giluLsol (SchPilu *schpilu*, double * *b*, double * *y*)

This routine does the forward solve L y = b. Can be done in place.

ON ENTRY :

b = a vector

mata = the matrix (in **SparRow** form)

ON RETURN :

y = the product L^{-1} * b

4.4.1.27 int gilupgmr (double * *rhs*, double * *sol*, p4ptr *levmat*, Csr *bmat*, SchPilu *schpilu*, IterPar *ipar*, FILE * *fp*)

Distributed preconditioned GMRES algorithm that uses distributed ILU0 as a right preconditioner. gilupgmr is used on the expanded Schur Complement system, which is constructed by ARMS preconditioner

ON ENTRY :

rhs = real vector of length n containing the right hand side.

sol = real vector of length n containing an initial guess to the solution on input.

(levmat) = precon matrix in CSR format containing permuted, sorted Schur complement matrices at each level.

(ilusch) = LU factorization of input matrix from ILUT, in MSR format

(schpilu) = factors of distributed ILU0.

ON RETURN :

sol == contains an approximate solution (upon successful return).

int = 0 –> successful return.

int = 1 –> convergence not achieved in itmax iterations.

int = -1 –> the initial guess seems to be the exact solution (initial residual computed was zero)

int = -2 –> error in schurprod operation

WORK ARRAYS :

vv == work array of length [im+1][n] (used to store the Arnoldi basis)

hh == work array of length [im][im+1] (Householder matrix)

z == work array of length [n]

zz == work array of length [im][n] used for Flexible GMRES

SUBROUTINES CALLED :

lusolD : combined forward and backward solves BLAS1 routines.

PARAMETERS :

pgfpar[0] = epsilon on last level

pgipar[1] = Krylov space dimension on last level

pgipar[2] = maxits = max # iterations on last level

IMPORTANT NOTE :

maxits = 0 will still do at least one step. call lusolD instead if you need just a precon operation - see sol2.c

4.4.1.28 void giluUsol (SchPilu *schpilu*, double * *y*, double * *x*)

This routine does the backward solve U x = y. Can be done in place.

ON ENTRY :

y = a vector

schpilu = the matrix

ON RETURN :

x = the product U^{-1} * y

4.4.1.29 int ilutD (csptr *amat*, double * *droptol*, int * *lfil*, ilutptr *ilusch*)

ILUT factorization with dual truncation.

ON ENTRY :

(amat) = Matrix stored in **SparRow** struct.

(ilusch) = Pointer to **ILUTfac** struct

lfil[5] = number nonzeros in L-part

lfil[6] = number nonzeros in U-part (lfil >= 0)

droptol[5] = threshold for dropping small terms in L during factorization.

droptol[6] = threshold for dropping small terms in U.

ON RETURN :

(ilusch) = Contains L and U factors in an LUfact struct.

Individual matrices stored in **SparRow** structs. On return matrices have C (0) indexing.

INTEGER VALUE RETURNED :

0 –> successful return.

1 –> Error. Input matrix may be wrong. (The elimination process has generated a row in L or U whose length is > n.)

2 –> Memory allocation error.

5 –> Illegal value for lfil or last.

6 –> zero row encountered.

WORK ARRAYS :

jw, jwrev = integer work arrays of length n

w = real work array of length n

4.4.1.30 int ilutpC (csptr *amat*, double * *droptol*, int * *lfil*, double *permtol*, int *mband*, ilutptr *ilusch*)

ILUT factorization with dual truncation.

ON ENTRY :

(amat) = Matrix stored in **SparRow** struct.

lfil[5] = number nonzeros in L-part

lfil[6] = number nonzeros in U-part (lfil >= 0)

droptol[5] = threshold for dropping small terms in L during factorization.

droptol[6] = threshold for dropping small terms in U

permtol = tolerance ratio used to determine whether or not to permute two columns At step i columns i and j are permuted when

abs(a(i,j))*permtol > abs(a(i,i))

[0 –> never permute; good values 0.1 to 0.01]

mband = permuting is done within a band extending to mband diagonals only.

mband = 0 –> no pivoting

mband = n –> pivot is searched in whole column

ON RETURN :

(ilusch) = Contains L and U factors in an LUfact struct

Individual matrices stored in **SparRow** structs On return matrices have C (0) indexing

iperm = reverse permutation array

INTEGER VALUES RETURNED :

0 –> successful return

1 –> Error. Input matrix may be wrong (The elimination process has generated a row in L or U whose length is > n.)

2 –> Memory allocation error

5 –> Illegal value for lfil

6 –> zero row encountered

WORK ARRAYS :

jw, jwrev = integer work arrays of length n

w = real work array of length n

4.4.1.31 int indsetC (csptr *mat*, int *bsize*, int * *iord*, int * *nnod*, double *tol*, int *nbnd*)

Greedy algorithm for independent set ordering –

ON ENTRY :

(mat) = matrix in **SparRow** format

bsize = integer (input) the target size of each block. each block is of size >= bsize.

w = weight factors for the selection of the elements in the independent set. If w(i) is small i will be left for the vertex cover set.

tol = a tolerance for excluding a row from independent set.

ON RETURN :

iord = permutation array corresponding to the independent set ordering. Row number i will become row number iord[i] in permuted matrix.

nnod = (output) number of elements in the independent set.

The algorithm searches nodes in lexicographic order and groups the (BSIZE-1) nearest nodes of the current to form a block of size BSIZE. The current algorithm does not use values of the matrix.

4.4.1.32 int lev4_nnz (p4ptr *levmat*, int * *lev*, FILE * *ft*)

Counts all nonzero elements in levmat struct – recursive

4.4.1.33 int levset (csptr, csptr, int, int *, double *, double)

4.4.1.34 void Lsol (csptr *mata*, double * *b*, double * *x*)

This function does the forward solve L x = b. Can be done in place.

ON ENTRY :

mata = the matrix (in **SparRow** form)

b = a vector

ON RETURN :

x = the solution of L x = b

4.4.1.35 void Lsolp (int *start*, int *n*, csptr *mata*, double * *b*, double * *y*)

Can be done in place.

This routine does the forward solve L y = b. where L is upper or bottom part of local low triangular matrix

ON ENTRY :

start = the index of the first component

n = one ore than the index owned by the processor

b = a vector

mata = the matrix (in **SparRow** form)

ON RETURN :

y = the product L^{-1} * b

4.4.1.36 void lusolD (ilutptr *ilusch*, double * *y*, double * *x*)

Combination forward-backward substitution - This function solves the system (LU) x = y, given an LU decomposition of a matrix stored in (ilusch) in modified sparse row format

ON ENTRY :

ilusch = the LU matrix as provided from the ILU functions.

y = the right-hand-side vector

ON RETURN :

x = solution of LU x = y.

meth[0] rperm = 0:none 1:row (CTC)

meth[1] perm2 = 0:none 1:column (ILUTP)

meth[2] D1 = 0:none 1:scaling

meth[3] D2 = 0:none 1:scaling

NOTE :

Function is in place: call lusolD(ilusch, x, x) will solve the system with rhs x and overwrite the result on x.

4.4.1.37 void lusolDp (int *start*, int *rflag*, ilutptr *ilusch*, double * *y*, double * *x*)

This routine solves the system (LU) x = y, given an LU decomposition of a matrix stored in (ilusch) in modified sparse row format

ON ENTRY :

start = the index of fist component

rflag = indicates which part of matrix is solved

0 – left upper part of matrix

1 – right bottom part of matrix

y = the right-hand-side vector

ilusch = the LU matrix as provided from the ILU routines.

ON RETURN :

x = solution of LU x = y.

met[0] rperm = 0:none 1:row (CTC)

met[1] perm2 = 0:none 1:column (ILUTP)

met[2] D1 = 0:none 1:scaling

met[3] D2 = 0:none 1:scaling

NOTE :

routine is in place: call lusolD(ilusch, x, x) will solve the system with rhs x and overwrite the result on x.

4.4.1.38 void matvec (csptr *mata*, double * *x*, double * *y*)

This function does the matrix vector product y = A x.

ON ENTRY :

mata = the matrix (in **SparRow** form)

x = a vector

ON RETURN :

y = the product A ∗ x

4.4.1.39 void matvecz (csptr *mata*, double ∗ *x*, double ∗ *y*, double ∗ *z*)

This function does the matrix vector z = y - A x.

ON ENTRY :

mata = the matrix (in **SparRow** form)

x, y = two input vector

ON RETURN :

z = the result: y - A ∗ x

4.4.1.40 int nnz_arms (p4ptr *levmat*, ilutptr *ilschu*, int *nlev*, FILE ∗ *ft*)

Computes and prints out total number of nonzero elements used in ARMS factorization

4.4.1.41 int nnz_arms1 (p4ptr *levmat*, ilutptr *ilschu*, FILE ∗ *ft*)

4.4.1.42 int pgmr (double ∗ *rhs*, double ∗ *sol*, p4ptr *levmat*, ilutptr *ilusch*, double ∗ *pgfpar*, int ∗ *ipar*, FILE ∗ *fp*)

This is a simple version of the ARMS preconditioned GMRES algorithm. Set up as a right preconditioner.

ON ENTRY :

rhs = real vector of length n containing the right hand side.

sol = real vector of length n containing an initial guess to the solution on input.

(levmat) = precon matrix in CSR format containing permuted, sorted Schur complement matrices at each level.

(ilusch) = LU factorization of input matrix from ILUT, in MSR format

ON RETURN :

pgmr int = 0 –> successful return.

int = 1 –> convergence not achieved in itmax iterations.

int = -1 –> the initial guess seems to be the exact

solution (initial residual computed was zero)

int = -2 –> error in schurprod operation

sol == contains an approximate solution (upon successful return).

WORK ARRAYS :

vv == work array of length [im+1][n] (used to store the Arnoldi basis) hh == work array of length [im][im+1] (Householder matrix)+ z == work array of length [n] to store preconditioned vectors.

SUBROUTINES CALLED :

lusolD : combined forward and backward solves BLAS1 routines.

PARAMETERS :

pgfpar[0] = epsilon on last level

ipar[3] = Krylov space dimension on last level

ipar[4] = maxits = max # iterations on last level

IMPORTANT NOTE:

maxits = 0 will still do at least one step. call lusolD instead if you need just a precon operation - see sol2.c

4.4.1.43 int pilu (p4ptr *amat*, csptr *B*, csptr *C*, double * *droptol*, int * *lfil*, csptr *schur*)

Converted to C so that dynamic memory allocation may be implememted in order to have no dropping in block LU factors.

Partial block ILU factorization with dual truncation.

| B F | | L 0 | | U L^{-1} F |

| | = | | * | |

| E C | | E U^{-1} I | | 0 S |

where B is a sub-matrix of dimension B->n.

ON ENTRY :

(amat) = Permuted matrix stored in a **PerMat4** struct on entry – Individual matrices stored in **SparRow** structs. On entry matrices have C (0) indexing. on return contains also L and U factors. Individual matrices stored in **SparRow** structs. On return matrices have C (0) indexing.

lfil[0] = number nonzeros in L-part

lfil[1] = number nonzeros in U-part

lfil[2] = number nonzeros in L^{-1} F

lfil[3] = not used

lfil[4] = number nonzeros in Schur complement

droptol[0] = threshold for dropping small terms in L during factorization.

droptol[1] = threshold for dropping small terms in U.

droptol[2] = threshold for dropping small terms in L^{-1} F during factorization.

droptol[3] = threshold for dropping small terms in E U^{-1} during factorization.

droptol[4] = threshold for dropping small terms in Schur complement after factorization is completed.

ON RETURN :

(schur) = contains the Schur complement matrix (S in above diagram) stored in **SparRow** struct with C (0) indexing.

INTEGER VALUES RETURNED:

0 –> successful return.

1 –> Error. Input matrix may be wrong. (The elimination process has generated a row in L or U

whose length is > n.)

2 –> Memory allocation error.

5 –> Illegal value for lfil or last.

6 –> zero row in B block encountered.

7 –> zero row in [E C] encountered.

8 –> zero row in new Schur complement

WORK ARRAYS :

jw, jwrev = integer work arrays of length B->n.

w = real work array of length B->n.

jw2, jwrev2 = integer work arrays of length C->n.

w2 = real work array of length C->n.

4.4.1.44 void pILUTmat (ilutptr)

4.4.1.45 void plperm (p4ptr, int)

4.4.1.46 void plSpar (csptr, int)

4.4.1.47 void printmat (FILE * *ft*, csptr *A*, int *i0*, int *i1*)

To dump rows i0 to i1 of matrix for debugging purposes

4.4.1.48 void qsortC (int * *ja*, double * *ma*, int *left*, int *right*, int *abval*)

4.4.1.49 qsplitC (double * *a*, int * *ind*, int *n*, int *ncut*)

Does a quick-sort split of a real array.

ON ENTRY :

a[0 : (n-1)] is a real array

ON RETURN :

It is permuted such that its elements satisfy:

abs(a[i]) >= abs(a[ncut-1]) for i < ncut-1 and

abs(a[i]) <= abs(a[ncut-1]) for i > ncut-1

ind[0 : (n-1)] is an integer array permuted in the same way as a.

4.4.1.50 int roscalC (csptr *mata*, double * *diag*, int *nrm*)

This routine scales each row of mata so that the norm is 1.

ON ENTRY :

mata = the matrix (in **SparRow** form)

nrm = type of norm

0 (infty), 1 or 2

ON RETURN :

diag = diag[j] = 1/norm(row[j])

0 –> normal return

j –> row j is a zero row

4.4.1.51 int rpermC (csptr *mat*, int * *perm*)

This subroutine permutes the rows of a matrix in **SparRow** format. rperm computes B = P A where P is a permutation matrix. The permutation P is defined through the array perm: for each j, perm[j] represents the destination row number of row number j.

ON ENTRY :

(amat) = a matrix stored in **SparRow** format.

ON RETURN :

(amat) = P A stored in **SparRow** format.

INTEGER VALUE RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.4.1.52 int schpart (csptr *schur*, Csr *xmat*, SchPilu *schpilu*)

4.4.1.53 int schsgssol (double * *b*, double * *x*, p4ptr *levmat*, Csr *bmat*, SchPilu *schpilu*, IterPar *ipar*, FILE * *fp*)

| L 0 | | U L^{-1}F | M x = | | | | x = b | EU^{-1} I | | 0 S |

Solve subroutine used in preconditioning operation associated with the multi-level ilu preconditioner – It solves

where M is the multi-level block ILUT preconditioner constructed by arms.c. The last level is solved with symmetric Gauss-Seidel.

ON ENTRY :

b = double array of length n, store the right-hand side of arms x = b. Return unchanged.

(levmat) = permuted and sorted matrices for each level stored in **PerMat4** struct.

(ilschu) = matrix stored in IluSpar struct containing the ILU decomposition of the last level.

(schpilu) = factors for Gauss-Seidel preconditioner.

pgfpar[0] = epsilon on last level

pgfpar[1] = epsilon on intermediate levels

pgipar[0] = method - see below

pgipar[1] = Krylov dimension on last level

pgipar[2] = maximum # iterations on last level

pgipar[3] = Krylov dimension on intermediate levels

pgipar[4] = maximum # iterations on intermediate levels

pgipar[5] = print options

pgipar[6] = current level

ON RETURN :

x = double vector of length n, store the preconditioned residual, i.e., the solution of arms x = b.

METHODS :

FGMRES for outer iteration:

first digit: method / 10 =

0 no iteration on intermediate levels

1 iteration using RIGHT precon. GMRES between levels

2 iteration using LEFT precon. GMRES between levels

3 iteration using FGMRES between levels

second digit: method 10 =

0 no iteration on last level (one sweep of ILUTsolve)

1 iteration using RIGHT precon. GMRES on last level

2 iteration using LEFT precon. GMRES on last level

3 iteration using FGMRES on last level

100: DGMR for outer iteration, no iteration between levels

4.4.1.54 int schuramux (p4ptr *levmat*, ilutptr *Smat*, Csr *bmat*, int *nbnd*, double * *x*, double * *ww*, double * *y*)

| B F |

The Schur complement of | | is S = C - E B^{-1} F.

| E C |

This routine computes y = S * x using the different blocks of the original matrix. B^{-1} is approximated with (U^{-1})*(L^{-1}).

ON ENTRY :

x = vector of length nB

(levmat) = Current level full matrix in **PerMat4** struct.

ON RETURN :

y = contains the product y = Ax, where A is the reduced system.

4.4.1.55 int schurprod (p4ptr *levmat*, ilutptr *Smat*, double * *x*, double * *y*)

Schurprod

| B F |

The Schur complement of | | is S = C - E B^{-1} F.

| E C |

This function computes y = S * x using the different blocks of the original matrix. B^{-1} is

approximated with (U^{-1})∗(L^{-1}).

ON ENTRY :

levmat = struct for the multilevel decomp.

Smat = struct for last level Schur complement matrix

x = vector of length nB

ON RETURN :

y = the product y = S ∗ x

4.4.1.56 int setupBLU (p4ptr, int, int, csptr, csptr)

4.4.1.57 int setupCS (csptr *amat*, int *len*)

Initialize **SparRow** structs.

ON ENTRY :

(amat) = Pointer to a **SparRow** struct.

len = size of matrix

ON RETURN :

amat->n

->∗nnzrow

->∗∗ja

->∗∗ma

INTEGER VALUES RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.4.1.58 int setupILUT (ilutptr *amat*, int *len*)

Allocate pointers for **ILUTfac** structs.

ON ENTRY :

(amat) = Pointer to a **ILUTfac** struct.

len = size of L U blocks

ON RETURN :

amat->L for each block: amat->M->n

->U->nnzrow

->ja

->ma

->rperm (if meth[0] > 0)

->perm2 (if meth[1] > 0)

->D1 (if meth[2] > 0)

->D2 (if meth[3] > 0)

Permutation arrays are initialized to the identity. Scaling arrays are initialized to 1.0.

INTEGER VALUE RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.4.1.59 void sgslusol (SchPilu *schpilu*, double ∗ *wk1*, double ∗ *wk2*)

Solve with LU factors of Gauss-Seidel preconditioner

4.4.1.60 int sgspgmr (double ∗ *rhs*, double ∗ *sol*, p4ptr *levmat*, Csr *bmat*, SchPilu *schpilu*, IterPar *ipar*, FILE ∗ *fp*)

Distributed preconditioned GMRES algorithm that uses distributed ILU0 as a right preconditioner. gilupgmr is used on the expanded Schur Complement system, which is constructed by ARMS preconditioner

ON ENTRY :

rhs = real vector of length n containing the right hand side.

sol = real vector of length n containing an initial guess to the solution on input.

(levmat) = precon matrix in CSR format containing permuted, sorted Schur complement matrices at each level.

(ilusch) = LU factorization of input matrix from ILUT

(schpilu) = factors for the Gauss-Seidel preconditioner

ON RETURN :

sol == contains an approximate solution (upon successful return).

int = 0 –> successful return.

int = 1 –> convergence not achieved in itmax iterations.

int = -1 –> the initial guess seems to be the exact solution (initial residual computed was zero)

int = -2 –> error in schurprod operation

WORK ARRAYS :

vv == work array of length [im+1][n] (used to store the Arnoldi basis)

hh == work array of length [im][im+1] (Householder matrix)

z == work array of length [n]

zz == work array of length [im][n] used for Flexible GMRES

SUBROUTINES CALLED :

lusolD : combined forward and backward solves BLAS1 routines.

PARAMETERS :

pgfpar[0] = epsilon on last level

pgipar[1] = Krylov space dimension on last level

pgipar[2] = maxits = max # iterations on last level

IMPORTANT NOTE:

maxits = 0 will still do at least one step. call lusolD instead if you need just a precon operation - see sol2.c

4.4.1.61 int sgsschur (double * *rhs*, double * *sol*, p4ptr *levmat*, Csr *bmat*, SchPilu *schpilu*, IterPar *ipar*, FILE * *fp*)

4.4.1.62 void sparmat (csptr, char *)

4.4.1.63 int SparTran (csptr *amat*, csptr *bmat*, int *job*, int *flag*)

Finds the transpose of a matrix stored in **SparRow** format.

ON ENTRY :

(amat) = a matrix stored in **SparRow** format.

job = integer to indicate whether to fill the values (job.eq.1) of the matrix (bmat) or only the pattern.

flag = integer to indicate whether the matrix has been filled

0 - no filled

1 - filled

ON RETURN :

(bmat) = the transpose of (mata) stored in **SparRow** format.

INTEGER VALUE RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.4.1.64 void swapj (int *v*[], int *i*, int *j*)

4.4.1.65 void swapm (double *v*[], int *i*, int *j*)

4.4.1.66 void Usol (csptr *mata*, double * *b*, double * *x*)

This function does the backward solve U x = b. Can be done in place.

ON ENTRY :

mata = the matrix (in **SparRow** form)

b = a vector

ON RETURN :

x = the solution of U * x = b

4.4.1.67 void Usolp (int *start*, int *n*, csptr *mata*, double * *y*, double * *x*)

Can be done in place.

This routine does the backward solve U x = y, where U is upper or bottom part of local upper triangular matrix

ON ENTRY :

start = the index of the first component

n = one ore than the index owned by the processor

y = a vector

mata = the matrix (in **SparRow** form)

ON RETURN :

x = the product U^{-1} ∗ y

4.4.1.68 int weightsC (csptr *mat*, double ∗ *w*)

Defines weights based on diagonal dominance ratios

4.5 armsol2.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "../../INCLUDE/armsheads.h"
```

Defines

- #define **ZERO** 0.0
- #define **EPSILON** 1.0e-20
- #define **EPSMAC** 1.0e-16
- #define **epsmac** 1.0e-16

Functions

- int **armsol2** (int type, double ∗b, double ∗x, **p4ptr** levmat, **ilutptr** ilusch, double ∗pgfpar, int ∗ipar, int flag, FILE ∗fp)
- int **pgmr** (double ∗rhs, double ∗sol, **p4ptr** levmat, **ilutptr** ilusch, double ∗pgfpar, int ∗ipar, FILE ∗fp)
- int **schuramux** (**p4ptr** levmat, **ilutptr** Smat, **Csr** bmat, int nbnd, double ∗x, double ∗ww, double ∗y)
- void **Lsolp** (int start, int n, **csptr** mata, double ∗b, double ∗y)
- void **Usolp** (int start, int n, **csptr** mata, double ∗y, double ∗x)
- void **lusolDp** (int start, int rflag, **ilutptr** ilusch, double ∗y, double ∗x)

4.5.1 Define Documentation

4.5.1.1 #define EPSILON 1.0e-20

4.5.1.2 #define epsmac 1.0e-16

4.5.1.3 #define EPSMAC 1.0e-16

4.5.1.4 #define ZERO 0.0

4.5.2 Function Documentation

4.5.2.1 int armsol2 (int *type*, double ∗ *b*, double ∗ *x*, p4ptr *levmat*, ilutptr *ilusch*, double ∗ *pgfpar*, int ∗ *ipar*, int *flag*, FILE ∗ *fp*)

Solve subroutine used in preconditioning operation associated with the ARMS multi-level ilu preconditioner – It solves M x = b where

| L 0 | | U L^{-1}F |

M = | | | |

| EU^{-1} I | | 0 S |

M is the multi-level block ILUT preconditioner constructed by arms2 revised by Zhongze Li, Apr. 16th, 2001

ON ENTRY :

b = double array of length n, the right-hand side of M x = b. unchanged on return.

(levmat) = permuted and sorted matrices for each level stored in **PerMat4** struct.

(ilschu) = matrix stored in IluSpar struct containing the ILU decomposition of the last level.

pgfpar[0] = tolerance for convergence criterion for last level iteration.. [when applicable, i.e., when ipar[3]>0 see below] –

ipar[0:17] = integer array to store parameters for both arms construction (arms2) and iteration (armsol2).:

ipar[0]:=nlev. number of levels.

ipar[1]:=bsize. Not used here [used in arms2]

ipar[2]:=iout if (iout > 0) statistics on the run are printed to FILE ∗ft

ipar[3]:= Krylov subspace dimension for last level ipar[3] == 0 means only backward/forward solve is performed on last level. |

ipar[4]:= maximum # iterations on last level

ipar[5-9] NOT used [reserved for later use] - must be set to zero. [see however a special use of ipar[5] in fgmresC.]

ipar[10-13] == meth[0:3] = method flags for interlevel blocks

ipar[14-17] == meth[0:3] = method flags for last level block - with the following meaning:

meth[0] permutations of rows 0:no 1: yes. affects rperm NOT USED IN THIS VERSION ∗∗ enter 0.. Data: rperm

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 no matter - enter zero; but 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 similarly for meth[14], ..., meth[17] all transformations related to parametres in meth[∗] (permutation, scaling,..) are applied to the matrix before processing it flag indicates which part is solved 0 – the whole linear system is solved, that is U^{-1}L^{-1}rhs 1 – the interface nodes is solved, that is U_S^{-1}L^{-1}rhs, U_S stands for the U part of the interface linear system

ON RETURN :

x = double vector of length n, store the preconditioned residual, i.e., the solution of arms x = b.

4.5.2.2 void Lsolp (int *start*, int *n*, csptr *mata*, double ∗ *b*, double ∗ *y*)

Can be done in place.

This routine does the forward solve L y = b. where L is upper or bottom part of local low triangular matrix

ON ENTRY :

start = the index of the first component

n = one ore than the index owned by the processor

b = a vector

mata = the matrix (in **SparRow** form)

ON RETURN :

y = the product L^{-1} ∗ b

4.5.2.3 void lusolDp (int *start*, int *rflag*, ilutptr *ilusch*, double ∗ *y*, double ∗ *x*)

This routine solves the system (LU) x = y, given an LU decomposition of a matrix stored in (ilusch) in modified sparse row format

ON ENTRY :

start = the index of fist component

rflag = indicates which part of matrix is solved

0 – left upper part of matrix

1 – right bottom part of matrix

y = the right-hand-side vector

ilusch = the LU matrix as provided from the ILU routines.

ON RETURN :

x = solution of LU x = y.

met[0] rperm = 0:none 1:row (CTC)

met[1] perm2 = 0:none 1:column (ILUTP)

met[2] D1 = 0:none 1:scaling

met[3] D2 = 0:none 1:scaling

NOTE :

routine is in place: call lusolD(ilusch, x, x) will solve the system with rhs x and overwrite the result on x.

4.5.2.4 int pgmr (double ∗ *rhs*, double ∗ *sol*, p4ptr *levmat*, ilutptr *ilusch*, double ∗ *pgfpar*, int ∗ *ipar*, FILE ∗ *fp*)

This is a simple version of the ARMS preconditioned GMRES algorithm. Set up as a right preconditioner.

ON ENTRY :

rhs = real vector of length n containing the right hand side.

sol = real vector of length n containing an initial guess to the solution on input.

(levmat) = precon matrix in CSR format containing permuted, sorted Schur complement matrices at each level.

(ilusch) = LU factorization of input matrix from ILUT, in MSR format

ON RETURN :

pgmr int = 0 –> successful return.

int = 1 –> convergence not achieved in itmax iterations.

int = -1 –> the initial guess seems to be the exact

solution (initial residual computed was zero)

int = -2 –> error in schurprod operation

sol == contains an approximate solution (upon successful return).

WORK ARRAYS :

vv == work array of length [im+1][n] (used to store the Arnoldi basis) hh == work array of length [im][im+1] (Householder matrix)+ z == work array of length [n] to store preconditioned vectors.

SUBROUTINES CALLED :

lusolD : combined forward and backward solves BLAS1 routines.

PARAMETERS :

pgfpar[0] = epsilon on last level

ipar[3] = Krylov space dimension on last level

ipar[4] = maxits = max # iterations on last level

IMPORTANT NOTE:

maxits = 0 will still do at least one step. call lusolD instead if you need just a precon operation - see sol2.c

4.5.2.5 int schuramux (p4ptr *levmat*, ilutptr *Smat*, Csr *bmat*, int *nbnd*, double * *x*, double * *ww*, double * *y*)

| B F |

The Schur complement of | | is S = C - E B^{-1} F.

| E C |

This routine computes y = S * x using the different blocks of the original matrix. B^{-1} is approximated with (U^{-1})*(L^{-1}).

ON ENTRY :

x = vector of length nB

(levmat) = Current level full matrix in **PerMat4** struct.

ON RETURN :

y = contains the product y = Ax, where A is the reduced system.

4.5.2.6 void Usolp (int *start*, int *n*, csptr *mata*, double * *y*, double * *x*)

Can be done in place.

This routine does the backward solve U x = y, where U is upper or bottom part of local upper triangular matrix

ON ENTRY :

start = the index of the first component

n = one ore than the index owned by the processor

y = a vector

mata = the matrix (in **SparRow** form)

ON RETURN :

x = the product U^{-1} ∗ y

4.6 base.h File Reference

Defines

- #define **Ddot** ddot_
- #define **Dcopy** dcopy_
- #define **Dscal** dscal_
- #define **Daxpy** daxpy_
- #define **Dnrm2** dnrm2_
- #define **Idmin** idmin_
- #define **DDOT**(n, x, incx, y, incy) Ddot(&(n),(x),&(incx),(y),&(incy))
- #define **DCOPY**(n, x, incx, y, incy) Dcopy(&(n),(x),&(incx),(y),&(incy))
- #define **DSCAL**(n, alpha, x, incx) Dscal(&(n),&(alpha),(x), &(incx))
- #define **DAXPY**(n, alpha, x, incx, y, incy)
- #define **DNRM2**(n, x, incx) Dnrm2(&(n), (x), &(incx))
- #define **IDMIN**(n, sx, incx) Idmin((&(n), (sx), &(incx))

Functions

- double **Ddot** (int *n, double *x, int *incx, double *y, int *incy)
- void **Dcopy** (int *n, double *x, int *incx, double *y, int *incy)
- void **Dscal** (int *n, double *alpha, double *x, int *incx)
- void **Daxpy** (int *n, double *alpha, double *x, int *incx, double *y, int *incy)
- double **Dnrm2** (int *n, double *x, int *incx)
- void **Idmin** (int *n, double *sx, int *incx)

4.6.1 Define Documentation

4.6.1.1 #define DAXPY(n, alpha, x, incx, y, incy)

Value:

```
Daxpy(&(n), &(alpha), (x), \
                                &(incx), y, &(incy))
```

4.6.1.2 #define Daxpy daxpy_

4.6.1.3 #define DCOPY(n, x, incx, y, incy) Dcopy(&(n),(x),&(incx),(y),&(incy))

4.6.1.4 #define Dcopy dcopy_

4.6.1.5 #define DDOT(n, x, incx, y, incy) Ddot(&(n),(x),&(incx),(y),&(incy))

4.6.1.6 #define Ddot ddot_

4.6.1.7 #define DNRM2(n, x, incx) Dnrm2(&(n), (x), &(incx))

4.6.1.8 #define Dnrm2 dnrm2_

4.6.1.9 #define DSCAL(n, alpha, x, incx) Dscal(&(n),&(alpha),(x), &(incx))

4.6.1.10 #define Dscal dscal_

4.6.1.11 #define IDMIN(n, sx, incx) Idmin((&(n), (sx), &(incx))

4.6.1.12 #define Idmin idmin_

4.6.2 Function Documentation

4.6.2.1 void Daxpy (int * *n*, double * *alpha*, double * *x*, int * *incx*, double * *y*, int * *incy*)

4.6.2.2 void Dcopy (int * *n*, double * *x*, int * *incx*, double * *y*, int * *incy*)

4.6.2.3 double Ddot (int * *n*, double * *x*, int * *incx*, double * *y*, int * *incy*)

4.6.2.4 double Dnrm2 (int * *n*, double * *x*, int * *incx*)

4.6.2.5 void Dscal (int * *n*, double * *alpha*, double * *x*, int * *incx*)

4.6.2.6 void Idmin (int * *n*, double * *sx*, int * *incx*)

4.7 comm.c File Reference

```
#include "../../INCLUDE/psparslib.h"
```

Functions

- void **PARMS_Init** (int *argc, char ***argv)
- void **PARMS_Final** ()
- void **MSG_bdx_bsend** (**Vec** x)
- void **Mtype_Create** (**Vec** x)
- void **Mtype_Free** (**Vec** x)
- int **ErrHand** (int ierr, **DistMatrix** dm, char *msg)

4.7.1 Function Documentation

4.7.1.1 int ErrHand (int *ierr*, DistMatrix *dm*, char * *msg*)

Sums the error messages in all processors and writes *msg if at least is > 0 (in abs value)

ON ENTRY :

ierr = error message number in each processor

dm = distributed matrix object (communicator only used)

msg = message to be printed if sum | err | != 0

ON RETURN :

Returns the sum of the absolute values of the terms ierr on all processors

4.7.1.2 void MSG_bdx_bsend (Vec *x*)

Interface information exchange general routine with non-blocking

ON ENTRY :

x = distribued local vector handler

ON RETURN :

x = distributed local vector contains external variables received from adjacent processors

4.7.1.3 void Mtype_Create (Vec *x*)

Create derived data types used for MPI

ON ENTRY :

x = distributed local vector handler

ON RETURN :

x = local vector handler which contains derived data types on return

4.7.1.4 void Mtype_Free (Vec *x*)

Free derived data type contained in distributed local vector x

ON ENTRY :

x = local vector handler

ON RETURN :

x = derived data types contained in x is freed on return

4.7.1.5 void PARMS_Final (void)

Exit PARMS and MPI environment

4.7.1.6 void PARMS_Init (int * *argc*, char *** *argv*)

Initialize the PARMS environment

ON ENTRY :

argc, argv = have the same meaning as that in the main function The function also sets up various linked lists for mat objects

4.8 data.h File Reference

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>
#include <memory.h>
#include "heads.h"
```

Data Structures

- struct **_p_Comm**
- struct **_p_Csr**
- struct **_p_CsrSpar**
- struct **_p_DistMatrix**
- struct **_p_DistMatrix::_p_Hash**
- struct **_p_DistMatrix::_p_Hash::_p_HashOps**
- struct **_p_DistMatrix::_p_Sparse_Matrix_Storage_Format**
- struct **_p_DistMatrix::_p_Sparse_Matrix_Storage_Format::_p_MatOps**
- struct **_p_DistMatrixCsr**
- struct **_p_Ext_Max**
- struct **_p_ILUfac**
- struct **_p_IterPar**
- struct **_p_Mix_Schur**
- struct **_p_MultiSch**
- struct **_p_MultiSchIlu**
- struct **_p_Pilu_Comm**
- struct **_p_PreCon**
- struct **_p_PrePar**
- struct **_p_SchPilu**
- struct **_p_Vec**
- struct **_p_Vec::_p_Vec_Comm**
- struct **_p_Vec::_p_Vec_Comm::_p_Vec_ComOps**

Defines

- #define **_DATA_HeADER_**
- #define **CHECKERR**(ierr) assert(!(ierr))
- #define **PARMS_malloc**(base, nmem, type)
- #define **PARMS_realloc**(base, nmem, type)
- #define **TYPESIZE** 20
- #define **COMM_TAG** 100
- #define **TRUE** 1
- #define **FALSE** 0
- #define **add_ilu0** 0
- #define **add_ilut** 1

- #define **add_iluk** 2
- #define **add_arms** 3
- #define **lsch_ilu0** 4
- #define **lsch_ilut** 5
- #define **lsch_iluk** 6
- #define **lsch_arms** 7
- #define **rsch_ilu0** 8
- #define **rsch_ilut** 9
- #define **rsch_iluk** 10
- #define **rsch_arms** 11
- #define **sch_gilu0** 12
- #define **sch_sgs** 13

Typedefs

- typedef **_p_Comm _Comm**
- typedef **_Comm** ∗ **Comm**
- typedef **_p_Vec _Vec**
- typedef **_Vec** ∗ **Vec**
- typedef _p_Vec_ComOps **_Vec_ComOps**
- typedef **_Vec_ComOps** ∗ **Vec_ComOps**
- typedef _p_Vec_Comm **_Vec_Comm**
- typedef **_Vec_Comm** ∗ **Vec_Comm**
- typedef **_p_PrePar _PrePar**
- typedef **_PrePar** ∗ **PrePar**
- typedef **_p_IterPar _IterPar**
- typedef **_IterPar** ∗ **IterPar**
- typedef **_p_ILUfac _ILUfac**
- typedef **_ILUfac** ∗ **ILUfac**
- typedef **_p_MultiSch _MultiSch**
- typedef **_MultiSch** ∗ **MultiSch**
- typedef **_p_Pilu_Comm _Pilu_Comm**
- typedef **_Pilu_Comm** ∗ **Pilu_Comm**
- typedef **_p_Ext_Max _Ext_Max**
- typedef **_Ext_Max** ∗ **Ext_Max**
- typedef **_p_Mix_Schur _Mix_Schur**
- typedef **_Mix_Schur** ∗ **Mix_Schur**
- typedef **_p_SchPilu _SchPilu**
- typedef **_SchPilu** ∗ **SchPilu**
- typedef **_p_MultiSchIlu _MultiSchIlu**
- typedef **_MultiSchIlu** ∗ **MultiSchIlu**
- typedef **_p_DistMatrix _DistMatrix**
- typedef _p_MatOps **_MatOps**
- typedef **_MatOps** ∗ **MatOps**
- typedef _p_Sparse_Matrix_Storage_Format **_Sparse_Matrix_Storage_Format**
- typedef **_Sparse_Matrix_Storage_Format** ∗ **Sparse_Matrix_Storage_Format**
- typedef _p_HashOps **_HashOps**
- typedef **_HashOps** ∗ **HashOps**
- typedef _p_Hash **_Hash**
- typedef **_Hash** ∗ **Hash**

- typedef **_DistMatrix** ∗ **DistMatrix**
- typedef **_p_Csr _Csr**
- typedef **_Csr** ∗ **Csr**
- typedef **_p_CsrSpar _CsrSpar**
- typedef **_CsrSpar** ∗ **CsrSpar**
- typedef **_p_DistMatrixCsr _DistMatrixCsr**
- typedef **_DistMatrixCsr** ∗ **DistMatrixCsr**
- typedef **_p_PreCon _PreCon**
- typedef **_PreCon** ∗ **PreCon**
- typedef int(∗ **PREC**)(struct **_p_DistMatrix** ∗dm, **PreCon** precon, **PrePar** prepar)
- typedef int(∗ **PRECOND_ROUTINE**)(**DistMatrix** dm, struct **_p_PreCon** ∗precon, **IterPar** iterpar, **Vec** rhs, **Vec** sol)

4.8.1 Define Documentation

4.8.1.1 #define _DATA_HeADER_

4.8.1.2 #define add_arms 3

4.8.1.3 #define add_ilu0 0

4.8.1.4 #define add_iluk 2

4.8.1.5 #define add_ilut 1

4.8.1.6 #define CHECKERR(ierr) assert(!(ierr))

4.8.1.7 #define COMM_TAG 100

4.8.1.8 #define FALSE 0

4.8.1.9 #define lsch_arms 7

4.8.1.10 #define lsch_ilu0 4

4.8.1.11 #define lsch_iluk 6

4.8.1.12 #define lsch_ilut 5

4.8.1.13 #define PARMS_malloc(base, nmem, type)

Value:

```
{\
  (base) = (type *)malloc((nmem)*sizeof(type)); \
  CHECKERR((base) == NULL); \
}
```

4.8.1.14 #define PARMS_realloc(base, nmem, type)

Value:

```
{\
  (base) = (type *)realloc((base), (nmem)*sizeof(type)); \
  CHECKERR((base) == NULL); \
}
```

4.8.1.15 #define rsch_arms 11

4.8.1.16 #define rsch_ilu0 8

4.8.1.17 #define rsch_iluk 10

4.8.1.18 #define rsch_ilut 9

4.8.1.19 #define sch_gilu0 12

4.8.1.20 #define sch_sgs 13

4.8.1.21 #define TRUE 1

4.8.1.22 #define TYPESIZE 20

4.8.2 Typedef Documentation

4.8.2.1 typedef struct _p_Comm _Comm

4.8.2.2 typedef struct _p_Csr _Csr

4.8.2.3 typedef struct _p_CsrSpar _CsrSpar

4.8.2.4 typedef struct _p_DistMatrix _DistMatrix

4.8.2.5 typedef struct _p_DistMatrixCsr _DistMatrixCsr

4.8.2.6 typedef struct _p_Ext_Max _Ext_Max

4.8.2.7 typedef struct _p_Hash _Hash

4.8.2.8 typedef struct _p_HashOps _HashOps

4.8.2.9 typedef struct _p_ILUfac _ILUfac

4.8.2.10 typedef struct _p_IterPar _IterPar

4.8.2.11 typedef struct _p_MatOps _MatOps

4.8.2.12 typedef struct _p_Mix_Schur _Mix_Schur

4.8.2.13 typedef struct _p_MultiSch _MultiSch

4.8.2.14 typedef struct _p_MultiSchIlu _MultiSchIlu

4.8.2.15 typedef struct _p_Pilu_Comm _Pilu_Comm

4.8.2.16 typedef struct _p_PreCon _PreCon

4.8.2.17 typedef struct _p_PrePar _PrePar

4.8.2.18 typedef struct _p_SchPilu _SchPilu

4.8.2.19 typedef struct _p_Sparse_Matrix_Storage_Format _Sparse_Matrix_Storage_Format

4.8.2.20 typedef struct _p_Vec _Vec

4.8.2.21 typedef struct _p_Vec_Comm _Vec_Comm

4.8.2.40 typedef int(* PRECOND_ROUTINE)(DistMatrix dm, struct _p_PreCon *precon, IterPar iterpar, Vec rhs, Vec sol)

4.8.2.41 typedef _PrePar* PrePar

4.8.2.42 typedef _SchPilu* SchPilu

4.8.2.43 typedef _Sparse_Matrix_Storage_Format* Sparse_Matrix_Storage_Format

4.8.2.44 typedef _Vec* Vec

4.8.2.45 typedef _Vec_Comm* Vec_Comm

4.8.2.46 typedef _Vec_ComOps* Vec_ComOps

4.9 dd-grid-edge.c File Reference

```
#include "../../INCLUDE/psparslib.h"
#include <stdlib.h>
#include <string.h>
```

Defines

- #define **SOLVER** "fgmres"
- #define **part2** part2_
- #define **partedge** partedge_
- #define **gen5loc** gen5loc_
- #define **setinit** setinit_
- #define **BUFLEN** 80

Functions

- void **setpar** (char *filename, int *method, int *mprocx, int *mprocy, int *nmesh, **PrePar** prepar, **IterPar** ipar, int *iov, **DistMatrix** dm)
- void **part2** (int *nx, int *ny, int *nz, int *mpx, int *mpy, int *mpz, int *ovp, int *lst, int *lstptr, int *iout)
- void **partedge** (int *nx, int *ny, int *nz, int *mpx, int *mpy, int *mpz, int *ovp, int *lst, int *lstptr, int *iout)
- void **gen5loc** (int *nx, int *ny, int *nz, int *nloc, int *riord, double *a, int *ja, int *ia, double *stencil)
- void **printm** (int *, int *, double *, int *, int *)
- void **setinit** (double *, int *)
- double **dwalltime** (void)
- int **main** (int argc, char *argv[])

  TEST PROGRAM.

4.9.1 Define Documentation

4.9.1.1 #define BUFLEN 80

4.9.1.2 #define gen5loc gen5loc_

4.9.1.3 #define part2 part2_

4.9.1.4 #define partedge partedge_

4.9.1.5 #define setinit setinit_

4.9.1.6 #define SOLVER "fgmres"

4.9.2 Function Documentation

4.9.2.1 double dwalltime ()

Wallclock timer function

dwaltime : Time in seconds since 00:00:00 UTC, Jan. 1st, 1970

4.9.2.2 void gen5loc (int * *nx*, int * *ny*, int * *nz*, int * *nloc*, int * *riord*, double * *a*, int * *ja*, int * *ia*, double * *stencil*)

4.9.2.3 int main (int *argc*, char * *argv*[])

TEST PROGRAM.

This driver generates a distributed sparse linear system from a 5-point finite difference problem on a rectangular grid (2-D or 3-D) and solves it with any of the methods available in pARMS. Each node generates its own part of the linear system using a simple regular partitioning. dd-grid-edge uses an edge-based partitioniner while dd-grid uses a vertex-based partitioner and dd-grid-solver is set up to use both (see comments within each driver). Arbitrary overlap is also allowed. Also dd-grid-solver allows the use of three accelerators (Flexible GMRES, Deflated GMRES, and BCGSTAB). Most of the options are set in the input file (see "inputs" for a sample) but a couple are set by define commands.

The mesh is defined as follows. The grid is virtually laid out on a 2-D or 3-D array of processors: mprocx in x direction, mprocy in y direction and mprocz in the z direction. mprocx and mprocy are both read from the input file ("inputs") and mprocz is determined as mprocz = numproc / (mprocx∗mprocy) where numproc is the total number of processors available for this run. Then the mesh sizes in each direction are set by the commands:

nx = nmesh∗mprocx

ny = nmesh∗mprocy

nz = nmesh∗mprocz

If numproc is not a multiple of mprocx∗mprocy the code will abort.

USAGE :

Parameters for methods, problem setting, etc, are read from a file. The default input file is "inputs" located in the same directory.

grid-solver.ex : Uses "inputs" as input file and stdout for output.

grid-solver.ex inp : Uses "inp" as default input file and stdout for output.

grid-solver.ex inp out : Uses "inp" as default input file and "out" for output.

ACCELERATORS :

This driver uses fgmres as an accelerator.

EDGE and VERTEX partitioning :

This driver performs an edge-based partitioninig of the grid.

4.9.2.4 void part2 (int * *nx*, int * *ny*, int * *nz*, int * *mpx*, int * *mpy*, int * *mpz*, int * *ovp*, int * *lst*, int * *lstptr*, int * *iout*)

4.9.2.5 void partedge (int * *nx*, int * *ny*, int * *nz*, int * *mpx*, int * *mpy*, int * *mpz*, int * *ovp*, int * *lst*, int * *lstptr*, int * *iout*)

4.9.2.6 void printm (int *, int *, double *, int *, int *)

4.9.2.7 void setinit (double *, int *)

4.9.2.8 void setpar (char * *filename*, int * *method*, int * *mprocx*, int * *mprocy*, int * *nmesh*, PrePar *prepar*, IterPar *ipar*, int * *iov*, DistMatrix *dm*)

4.10 dd-grid-simple.c File Reference

```
#include "../../INCLUDE/psparslib.h"
#include <stdlib.h>
#include <string.h>
```

Defines

- #define **part1** part1_
- #define **part2** part2_
- #define **gen5loc** gen5loc_
- #define **setinit** setinit_
- #define **BUFLEN** 80

Functions

- void **part1** (int *nx, int *ny, int *mpx, int *mpy, int *ovp, int *lst, int *lstptr, int *iout)
- void **part2** (int *nx, int *ny, int *nz, int *mpx, int *mpy, int *mpz, int *ovp, int *lst, int *lstptr, int *iout)
- void **gen5loc** (int *nx, int *ny, int *nz, int *nloc, int *riord, double *a, int *ja, int *ia, double *stencil)
- void **printm** (int *, int *, double *, int *, int *)
- void **setinit** (double *, int *)
- double **dwalltime** (void)
- int **main** (int argc, char *argv[])

  TEST PROGRAM.

4.10.1 Define Documentation

4.10.1.1 #define BUFLEN 80

4.10.1.2 #define gen5loc gen5loc_

4.10.1.3 #define part1 part1_

4.10.1.4 #define part2 part2_

4.10.1.5 #define setinit setinit_

4.10.2 Function Documentation

4.10.2.1 double dwalltime ()

Wallclock timer function

dwaltime : time in seconds since 00:00:00 UTC, Jan. 1st, 1970

4.10.2.2 void gen5loc (int * *nx*, int * *ny*, int * *nz*, int * *nloc*, int * *riord*, double * *a*, int * *ja*, int * *ia*, double * *stencil*)

4.10.2.3 int main (int *argc*, char * *argv*[])

TEST PROGRAM.

This driver generates a distributed sparse linear system from a 5-point finite difference problem on a rectangular grid (2-D or 3-D) and solves it with any of the methods available in pARMS. Each node generates its own part of the linear system using a simple regular partitioning. dd-grid-edge uses an edge-based partitioniner while dd-grid-simple uses a vertex-based partitioner and dd-grid-solver is set up to use both (see comments within each driver). Arbitrary overlap is also allowed. Also dd-grid-solver allows the use of three accelerators (Flexible GMRES, Deflated GMRES, and BCGSTAB)

The mesh is defined as follows. The grid is virtually laid out on a 2-D or 3-D array of processors: mprocx in x direction, mprocy in y direction and mprocz in the z direction. mprocx and mprocy are both set to values of 2 and mprocz is determined as follows mprocz = numproc / (mprocx∗mprocy) where numproc is the total number of processors available for this run. Then the mesh sizes in each direction are set by the commands nmesh value is given to be 30 in this test driver nx = nmesh∗mprocx ny = nmesh∗mprocy nz = nmesh∗mprocz

If numproc is not a multiple of mprocx∗mprocy the code will abort.

All parameters are input directly instead of being read from a file. Also standard output is used for showing the results

ACCELERATORS

This driver uses fgmres as an accelerator

EDGE and VERTEX partitioning

This driver uses only a vertex-based partitioner

4.10.2.4 void part1 (int * *nx*, int * *ny*, int * *mpx*, int * *mpy*, int * *ovp*, int * *lst*, int * *lstptr*, int * *iout*)

4.10.2.5 void part2 (int * *nx*, int * *ny*, int * *nz*, int * *mpx*, int * *mpy*, int * *mpz*, int * *ovp*, int * *lst*, int * *lstptr*, int * *iout*)

4.10.2.6 void printm (int *, int *, double *, int *, int *)

4.10.2.7 void setinit (double *, int *)

4.11 dd-grid-solver.c File Reference

```
#include "../../INCLUDE/psparslib.h"
#include <stdlib.h>
#include <string.h>
```

Defines

- #define **SOLVER** "fgmres"
- #define **EDGE** 1
- #define **part2** part2_
- #define **part_edge** part_edge_
- #define **gen5loc** gen5loc_
- #define **setinit** setinit_
- #define **BUFLEN** 80

Functions

- void **setpar** (char *filename, int *method, int *mprocx, int *mprocy, int *nmesh, **PrePar** prepar, **IterPar** ipar, int *iov, **DistMatrix** dm)
- void **part2** (int *nx, int *ny, int *nz, int *mpx, int *mpy, int *mpz, int *ovp, int *lst, int *lstptr, int *iout)
- void **part_edge** (int *nx, int *ny, int *nz, int *mpx, int *mpy, int *mpz, int *ovp, int *lst, int *lstptr, int *iout)
- void **gen5loc** (int *nx, int *ny, int *nz, int *nloc, int *riord, double *a, int *ja, int *ia, double *stencil)
- void **printm** (int *, int *, double *, int *, int *)
- void **setinit** (double *, int *)
- double **dwalltime** (void)
- int **main** (int argc, char *argv[])

  TEST PROGRAM.

4.11.1 Define Documentation

4.11.1.1 #define BUFLEN 80

4.11.1.2 #define EDGE 1

4.11.1.3 #define gen5loc gen5loc_

4.11.1.4 #define part2 part2_

4.11.1.5 #define part_edge part_edge_

4.11.1.6 #define setinit setinit_

4.11.1.7 #define SOLVER "fgmres"

4.11.2 Function Documentation

4.11.2.1 double dwalltime ()

Wallclock timer function

dwaltime : time in seconds since 00:00:00 UTC, Jan. 1st, 1970

4.11.2.2 void gen5loc (int * *nx*, int * *ny*, int * *nz*, int * *nloc*, int * *riord*, double * *a*, int * *ja*, int * *ia*, double * *stencil*)

4.11.2.3 int main (int *argc*, char * *argv*[])

TEST PROGRAM.

This driver generates a distributed sparse linear system from a 5-point finite difference problem on a rectangular grid (2-D or 3-D) and solves it with any of the methods available in pARMS. Each node generates its own part of the linear system using a simple regular partitioning. dd-grid-edge uses an edge-based partitioniner while dd-grid uses a vertex-based partitioner and dd-grid-solver is set up to use both (see comments within each driver). Arbitrary overlap is also allowed. Also dd-grid-solver allows the use of three accelerators (Flexible GMRES, Deflated GMRES, and BCGSTAB). Most of the options are set in the input file (see "inputs" for a sample) but a couple are set by define commands.

The mesh is defined as follows. The grid is virtually laid out on a 2-D or 3-D array of processors: mprocx in x direction, mprocy in y direction and mprocz in the z direction. mprocx and mprocy are both read from the input file ("inputs") and mprocz is determined as mprocz = numproc / (mprocx∗mprocy) where numproc is the total number of processors available for this run. Then the mesh sizes in each direction are set by the commands:

nx = nmesh∗mprocx ny = nmesh∗mprocy nz = nmesh∗mprocz

If numproc is not a multiple of mprocx∗mprocy the code will abort.

USAGE :

parameters for methods, problem setting, etc, are read from a file. the default input file is "inputs" located in the same directory

grid-solver.ex : uses "inputs" as input file and stdout for output grid-solver.ex inp : uses "inp" as default input file and stdout for output grid-solver.ex inp out : uses "inp" as default input file

and "out" for output

ACCELERATORS

Three accelerators can be used with this code :

FGMRESD: flexible GMRES used with all preconditioner options(with or without inner iteration)

DGMRESD: distributed deflated GMRES – this uses eigenvalude deflation. NOTE: NO INNER ITERATIONS SHOULD BE ALLOWED WHEN YOU USE DGMRES (otherwise the method will not work). There is no checking done for this.

BCGSTABD: distributed BCGSTAB. same note as for the Deflated GMRES: this cannot be used with inner iterations

These are set with a define command (define SOLVER).

EDGE and VERTEX partitioning

This driver allows both edge-based partitioning and vertex based partitioning. This is set by a define command (see define EDGE below). If the EDGE parameter is set to one then the code will select edge partitioning, otherwise it will perform a vertex partitioning.

4.11.2.4 void part2 (int * *nx*, int * *ny*, int * *nz*, int * *mpx*, int * *mpy*, int * *mpz*, int * *ovp*, int * *lst*, int * *lstptr*, int * *iout*)

4.11.2.5 void part_edge (int * *nx*, int * *ny*, int * *nz*, int * *mpx*, int * *mpy*, int * *mpz*, int * *ovp*, int * *lst*, int * *lstptr*, int * *iout*)

4.11.2.6 void printm (int *, int *, double *, int *, int *)

4.11.2.7 void setinit (double *, int *)

4.11.2.8 void setpar (char * *filename*, int * *method*, int * *mprocx*, int * *mprocy*, int * *nmesh*, PrePar *prepar*, IterPar *ipar*, int * *iov*, DistMatrix *dm*)

4.12 dd-grid.c File Reference

```
#include "../../INCLUDE/psparslib.h"
#include <stdlib.h>
#include <string.h>
```

Defines

- #define **part1** part1_
- #define **part2** part2_
- #define **gen5loc** gen5loc_
- #define **setinit** setinit_
- #define **BUFLEN** 80

Functions

- void **setpar** (char *filename, int *method, int *mprocx, int *mprocy, int *nmesh, **PrePar** prepar, **IterPar** ipar, int *iov, **DistMatrix** dm)
- void **part1** (int *nx, int *ny, int *mpx, int *mpy, int *ovp, int *lst, int *lstptr, int *iout)
- void **part2** (int *nx, int *ny, int *nz, int *mpx, int *mpy, int *mpz, int *ovp, int *lst, int *lstptr, int *iout)
- void **gen5loc** (int *nx, int *ny, int *nz, int *nloc, int *riord, double *a, int *ja, int *ia, double *stencil)
- void **printm** (int *, int *, double *, int *, int *)
- void **setinit** (double *, int *)
- double **dwalltime** (void)
- int **main** (int argc, char *argv[])

  TEST PROGRAM *.

4.12.1 Define Documentation

4.12.1.1 #define BUFLEN 80

4.12.1.2 #define gen5loc gen5loc_

4.12.1.3 #define part1 part1_

4.12.1.4 #define part2 part2_

4.12.1.5 #define setinit setinit_

4.12.2 Function Documentation

4.12.2.1 double dwalltime ()

Wallclock timer function

dwaltime : time in seconds since 00:00:00 UTC, Jan. 1st, 1970

4.12.2.2 void gen5loc (int * *nx*, int * *ny*, int * *nz*, int * *nloc*, int * *riord*, double * *a*, int * *ja*, int * *ia*, double * *stencil*)

4.12.2.3 int main (int *argc*, char * *argv*[])

TEST PROGRAM *.

This driver generates a distributed sparse linear system from a 5-point finite difference problem on a rectangular grid (2-D or 3-D) and solves it with any of the methods available in pARMS. Each node generates its own part of the linear system using a simple regular partitioning. dd-grid-edge uses an edge-based partitioniner while dd-grid uses a vertex-based partitioner and dd-grid-solver is set up to use both (see comments within each driver). Arbitrary overlap is also allowed. Also dd-grid-solver allows the use of three accelerators (Flexible GMRES, Deflated GMRES, and BCGSTAB). Most of the options are set in the input file (see "inputs" for a sample) but a couple are set by define commands.

The mesh is defined as follows. The grid is virtually laid out on a 2-D or 3-D array of processors: mprocx in x direction, mprocy in y direction and mprocz in the z direction. mprocx and mprocy are both read from the input file ("inputs") and mprocz is determined as mprocz = numproc / (mprocx*mprocy) where numproc is the total number of processors available for this run. Then the mesh sizes in each direction are set by the commands:

nx = nmesh*mprocx ny = nmesh*mprocy nz = nmesh*mprocz

If numproc is not a multiple of mprocx*mprocy the code will abort.

USAGE : parameters for methods, problem setting, etc, are read from a file. the default input file is "inputs" located in the same directory

grid-solver.ex : uses "inputs" as input file and stdout for output grid-solver.ex inp : uses "inp" as default input file and stdout for output grid-solver.ex inp out : uses "inp" as default input file and "out" for output

ACCELERATORS : This driver uses fgmres as an accelerator

EDGE and VERTEX partitioning This driver uses only a vertex-based partitioner

4.12.2.4 void part1 (int * *nx*, int * *ny*, int * *mpx*, int * *mpy*, int * *ovp*, int * *lst*, int * *lstptr*, int * *iout*)

4.12.2.5 void part2 (int * *nx*, int * *ny*, int * *nz*, int * *mpx*, int * *mpy*, int * *mpz*, int * *ovp*, int * *lst*, int * *lstptr*, int * *iout*)

4.12.2.6 void printm (int *, int *, double *, int *, int *)

4.12.2.7 void setinit (double *, int *)

4.12.2.8 void setpar (char * *filename*, int * *method*, int * *mprocx*, int * *mprocy*, int * *nmesh*, PrePar *prepar*, IterPar *ipar*, int * *iov*, DistMatrix *dm*)

4.13 dd-HB-simple.c File Reference

```
#include "../../INCLUDE/psparslib.h"
#include <stdlib.h>
#include <string.h>
#include "generaldefs.h"
```

Defines

- #define **BUFLEN** 100

Functions

- int **main** (int argc, char *argv[])

4.13.1 Define Documentation

4.13.1.1 #define BUFLEN 100

4.13.2 Function Documentation

4.13.2.1 int main (int *argc*, char * *argv*[])

In this test program all processors read the whole matrix from a file. The matrix is assumed to be in Harwell-Boeing format. It then partitions its graph using DSE, a simple partitioning routine. After that, the submatrices assigned to a given processor are extracted. (The global matrix is then discarded). Each processor solves the problem using FGMRES preconditioned with the available preconditioners. In this driver the right-hand side is set-up artificially to be A*ones(1:n). All parameters are input directly instead of being read from a file. Also standard output is used for showing the results.

4.14 dd-HB-dse.c File Reference

```
#include "../../INCLUDE/psparslib.h"
#include <stdlib.h>
#include <string.h>
#include "generaldefs.h"
```

Defines

- #define **BUFLEN** 100

Functions

- int **main** (int argc, char *argv[])

4.14.1 Define Documentation

4.14.1.1 #define BUFLEN 100

4.14.2 Function Documentation

4.14.2.1 int main (int *argc*, char * *argv*[])

In this test program processor number one reads the whole matrix from a file. The matrix is assumed to be in Harwell-Boeing format. It then partitions its graph using DSE, a simple partitioning routine, and scatters the local matrices to each processor. Once these submatrices are received each processor solves the problem using preconditioned FGMRES. See README or comments for the list of available preconditioners. This driver also allows a user to enter a matrix in arbitrary format by using a user-defined routine for reading the data – see comments related to "useread".

4.15 dd-HB-metis.c File Reference

```
#include "../../INCLUDE/psparslib.h"
#include <stdlib.h>
#include <string.h>
#include "metis.h"
#include "generaldefs.h"
```

Defines

- #define **BUFLEN** 100

Functions

- int **main** (int argc, char *argv[])

4.15.1 Define Documentation

4.15.1.1 #define BUFLEN 100

4.15.2 Function Documentation

4.15.2.1 int main (int *argc*, char * *argv*[])

In this test program, each processor reads the whole matrix from a file. The matrix is assumed to be in Harwell-Boeing format. Matrix graph is then partitioned using Metis (see `http://www.cs.umn.edu/~karypis`) after which the submatrices assigned to a given processor are extracted. (The global matrix is then discarded.) Each processor solves the problem using preconditioned FGMRES. See README or comments for the list of available preconditioners. This driver also allows a user to enter a matrix in arbitrary format by using a user-defined routine for reading the data – see comments related to "useread".

4.16 dd-HB-parmetis.c File Reference

```
#include "../../INCLUDE/psparslib.h"
#include <stdlib.h>
#include <string.h>
#include "../../../ParMetis-3.0/METISLib/metis.h"
```

Defines

- #define **BUFLEN** 100
- #define **wreadmtpar** wreadmtpar_
- #define **wreadmtpar2** wreadmtpar2_

Functions

- void **makeptr** (int *mapptr, int *maptmp, int *riord, int riordn, int npro, int n)
- void **rordcsr** (double *a, int *ia, int *ja, double *b, int *ib, int *jb, int *riord, int nnz, int riordn)
- void **setpar** (char *filename, char *matrix, int *iov, int *scale, int *unsym, int *method, PrePar prepar, IterPar ipar, DistMatrix dm)
- void **wreadmtpar** (int *job, char *fname, int *len, double *a, int *ja, int *ia, int *lia, int *lian, double *rhs, int *nrhs, char *guesol, int *nrow, int *ncol, int *nnz, int *lnnz, char *title, char *key, char *type, int *nyrange, int *npro, int *mypro, int *ierr)
- void **wreadmtpar2** (int *job, char *fname, int *len, double *a, int *ja, int *ia, int *lia, int *lian, double *rhs, int *nrhs, char *guesol, int *nrow, int *ncol, int *nnz, char *title, char *key, char *type, int *riord, int *riordn, int *ierr)
- int **main** (int argc, char *argv[])

4.16.1 Define Documentation

4.16.1.1 #define BUFLEN 100

4.16.1.2 #define wreadmtpar wreadmtpar_

4.16.1.3 #define wreadmtpar2 wreadmtpar2_

4.16.2 Function Documentation

4.16.2.1 int main (int *argc*, char * *argv*[])

In this test program all processors read its own part of the matrix in HB format file. The matrix is assumed to be in Harwell-Boeing format. It then partitions its graph [using a parmetis partitioning routine]. This program is just to show the utility of PARMS in parallel. It is a very simple case as there is no scaling done, as well as, the matrix is a test matrix which is symmetric of size 21x21. Further work needs to be done in order to solve a general class of matrix which is in progress at the moment. This program illustrates that there is no need for the whole matrix to be residing on a single processor in order to use PARMS. Currently the subroutine wreadmtpar and wreadmtpar2 reads the matrix from all the processors. It is possible to modify these routines in order to be read by a single processor and then distribute the chuncks of rows to different processors. This will be useful in cases where the I/O is slow in comparision to communication between processors.

4.16.2.2 **void makeptr (int * *mapptr*, int * *maptmp*, int * *riord*, int *riordn*, int *npro*, int *n*)**

4.16.2.3 **void rordcsr (double * *a*, int * *ia*, int * *ja*, double * *b*, int * *ib*, int * *jb*, int * *riord*, int *nnz*, int *riordn*)**

4.16.2.4 **void setpar (char * *filename*, char * *matrix*, int * *iov*, int * *scale*, int * *unsym*, int * *method*, PrePar *prepar*, IterPar *ipar*, DistMatrix *dm*)**

4.16.2.5 **void wreadmtpar (int * *job*, char * *fname*, int * *len*, double * *a*, int * *ja*, int * *ia*, int * *lia*, int * *lian*, double * *rhs*, int * *nrhs*, char * *guesol*, int * *nrow*, int * *ncol*, int * *nnz*, int * *lnnz*, char * *title*, char * *key*, char * *type*, int * *nyrange*, int * *npro*, int * *mypro*, int * *ierr*)**

4.16.2.6 **void wreadmtpar2 (int * *job*, char * *fname*, int * *len*, double * *a*, int * *ja*, int * *ia*, int * *lia*, int * *lian*, double * *rhs*, int * *nrhs*, char * *guesol*, int * *nrow*, int * *ncol*, int * *nnz*, char * *title*, char * *key*, char * *type*, int * *riord*, int * *riordn*, int * *ierr*)**

4.17 defs.h File Reference

Variables

- **_HashOps hops**
- **PREC prec_list** []
- **_Vec_ComOps vec_comops**
- **_PreCon _precon** []
- **PRECOND_ROUTINE sol0_dispatch** []

4.17.1 Variable Documentation

4.17.1.1 _PreCon _precon[]

4.17.1.2 _HashOps hops

4.17.1.3 PREC prec_list[]

4.17.1.4 PRECOND_ROUTINE sol0_dispatch[]

4.17.1.5 _Vec_ComOps vec_comops

4.18 dgmr.f File Reference

Functions

- **dgmr** (n, im, neig, ip, maxits, iout, ioutEig, eps, rhs, sol, i, its, vv, nvv, icode, ipo, ipi, ro)
- **mgsr** (n, i0, i1, ss, r)
- **deigen** (ndim, iii, im, ip, neig, work1, work2, hh, vv, ifirst, iwork, s, c, KryDim, alfr, alfi, beta, eigMag, eigScale, ieigOrder)
- **eigResid** (ndim, neig, vv, ifirst, iwork, ioutEig, icodeE, ipo, ipi, KryDim, alfr, alfi, beta, eigMag, eigScale, ieigOrder)
- **norder** (iii, eigMag, ieigOrder, alfr, alfi, beta, work, ldw)

4.18.1 Function Documentation

4.18.1.1 deigen (ndim, iii, im, ip, neig, work1, work2, hh, vv, ifirst, iwork, s, c, KryDim, alfr, alfi, beta, eigMag, eigScale, ieigOrder)

ON ENTRY :

ndim - size of matrix A

iii - number of Arnoldi vectors, including eigenvectors

im - maximum number of Arnoldi vectors, including eigenvectors

ip - number of eigenvectors in subspace W when subroutine called

neig - number of eigenvectors to be calculated

work1,2 - work arrays, must have same dimension as hh

hh - R matrix of QR decomposition of GMRES matrix { H}_m

vv - array used to store Krylov subspace V_{n+1} at vv(1,1), eigenvectors of W{n} at vv(1,ifirst), and workspace at vv(1,iwork). Here W and V are the subspaces from dgmr satisfying A M^{-1} W_{m} = V_{m+1} { H}_{m} Size of vv must be >= ndim * (im+1 + max(ip,neig) + 2)

ifirst - points to first column of vv containing eigenvectors

iwork - points to first column of work space of vv

s,c - contain rotations for QR decomposition of { H}_m

KryDim - number of vectors in Krylov subspace, and leading dimension of arrays

ON RETURN :

vv - contains neig eigenvectors in vv(1,ifirst),vv(1,ifirst+1),... vv(1,ifirst+neig-1)

WORK ARRAYS :

alfr - denominator of real part of eigenvalue

alfi - denominator of imaginary part of eigenvalue

beta - numerator of eigenvalue

eigMag - magnitude of eigenvalue

eigScale - scaling for eigenvectors

ieigOrder- order of eigenvalues

4.18.1.2 dgmr (n, im, neig, ip, maxits, iout, ioutEig, eps, rhs, sol, i, its, vv, nvv, icode, ipo, ipi, ro)

DGMR

Deflated GMRES with right preconditioning. The subroutine deigen is called to estimate the eigenvectors corresponding to the eigenvalues of smallest magnitude. dgmr uses reverse communication for matrix vector products and preconditioning.

REFERENCES : R B Morgan 'A restarted GMRES method augmented with eigenvectors' SIAM J. Matrix Analysis and Applications , 16(4):1154-1171,1995. Andrew Chapman and Yousef Saad 'Deflated and augmented Krylov subspace techniques' MSI report UMSI 95/181 September 1995

In the references above, eigenvectors are added as the last vectors of the augmented Krylov subspace. Here they are added as the first vectors. This usually results in better convergence between restarts.

ON ENTRY :

n = dimension of matrix

im = max number vectors in Krylov subspace (including eigenvectors)

neig = number of eigenvectors to be calculated and used in deflation. Set neig=0 for no deflation, and 1<=neig<=(im-1) for deflation. Usually neig<<(im-1), and a typical value is neig=4. See notes on ip and neig below.

ip = number of eigenvectors available on calling bgmr. Set ip=0 unless dgmr is used to solve for for multiple rhs. Here set ip=0 for first rhs, and dgmr will automatically set ip=neig for all other rhs. This causes eigenvectors to be recycled, and used as starting eigenvectors when solving for subsequent rhs. See notes on ip and neig below

maxits = max number of total gmres steps allowed

iout = number of file for writing output, iout=0 —> no output. This file needs to be opened in the calling program.

ioutEig= number of file for writing eigenvalue/vector residual. if ioutEig = 0, no eigenvalue/vector residuals are calculated or written. Note that calculating the residuals is time consuming, and they are they are calculated only to be output in ioutEig, they are never used. Set ioutEig=0, unless you are interested in seeing the residual in the calculated eigenvectors. This file needs to be opened in the calling program

eps = tolerance parameter, dgmr stops if ro_{its}.le.ro_{0}∗eps where ro is the residual norm |rhs-A sol|, and its is the step number

rhs = right-hand-side vector, dimension n

sol = initial guess of solution on input, dimension n

vv = array used to store Krylov subspace V_{m+1} at vv(1,1), eigenvectors at vv(1,ifirst), and workspace at vv(1,iwork). In calling program, vv must be an array of size >= n ∗ (im+1 + max(neig,ip) + 2) . For more information, see notes on V_{m+1} and W_{m} below

nvv = number entries in vv in calling program, used to check size

ON RETURN :

sol = solution on return

i = step number in current Arnoldi .. Should not be changed

its = total number of steps, including restarts

icode = used for reverse communication protocol. See notes below

ipo,ipi= pointers for reverse communication. See notes below

ip = If solving for multiple rhs, ip is the number of eigenvectors available as starting eigenvectors when solving for the next rhs

vv = contains ip eigenvectors at vv(1,ifirst), vv(1,ifirst+1), ... vv(1,ifirst-1+ip)

4.18.1.3 eigResid (ndim, neig, vv, ifirst, iwork, ioutEig, icodeE, ipo, ipi, KryDim, alfr, alfi, beta, eigMag, eigScale, ieigOrder)

ON ENTRY :

ndim - size of matrix A

neig - number of eigenvectors

vv - contains neig eigenvectors in vv(1,ifirst),vv(1,ifirst+1),...

ifirst - points to first column of vv containing eigenvectors

iwork - points to first column of work space of vv

ioutEig - iounit number for printing eigenvalue/vector residual. if ioutEig = 0, no eigenvalue/vector residuals are calculated or printed. This file needs to be opened in the calling program

icodeE - flag for reverse communication

ipo,ipi - pointers for reverse communication

KryDim - dimension of Krylov subspace, and leading dimension of arrays containing eigenvector information

alfr - denominator of real part of eigenvalue

alfi - denominator of imaginary part of eigenvalue

beta - numerator of eigenvalue

eigMag - magnitude of eigenvalue

eigScale - scaling for eigenvectors

ieigOrder- order of eigenvalues

ON RETURN :

none

4.18.1.4 mgsr (n, i0, i1, ss, r)

Modified Gram - Schmidt with partial reortho. The vector ss(∗,i1) is orthogonalized against vectors ss(∗,i0)...ss(∗,i1-1) (which are already orthogonal). The coefficients of the orthogonalization are returned in the array r

4.18.1.5 norder (iii, eigMag, ieigOrder, alfr, alfi, beta, work, ldw)

Orders eigenvalues and eigenvectors from eispack subroutine rgg so that eigenvalues are from largest to smallest.

ON ENTRY :

iii - number of eigenvalues calculated by rgg

alfr,alfi - real and imag parts of eigenvalues

beta - denominator of eigenvalues

work - array containing eigenvectors in columns

ldw - leading dimension of work

ON RETURN :

alfr,alfi,beta,work - eigenvalues and vectors, ordered largest to smallest

WORK ARRAYS :

eigMag - work array, contains magnitudes of eigenvalues

ieigOrder - work array, keeps order of eigenvalues

4.19 fdmat.f File Reference

Functions

- **gen5pt** (nx, ny, nz, a, ja, ia, iau)
- **part0** (nx, ny, mpx, mpy, ovlp, lst, lstptr, iout)
- **part1** (nx, ny, mpx, mpy, ovlp, lst, lstptr, iout)
- **part2** (nx, ny, nz, mpx, mpy, mpz, ovlp, lst, lstptr, iout)
- **part_edge** (nx, ny, nz, mpx, mpy, mpz, ovlp, lst, lstptr, iout)

4.19.1 Function Documentation

4.19.1.1 gen5pt (nx, ny, nz, a, ja, ia, iau)

This subroutine computes the sparse matrix in compressed format for the elliptic operator

L u = delx(a delx u) + dely (b dely u) + delz (c delz u) + d delx (u) + e dely (u) + f delz(u) + g u

with Dirichlet Boundary conditions, on a rectangular 1-D, 2-D or 3-D grid using centered difference schemes.

The functions a, b, ..., g are known through the subroutines afun, bfun, ..., gfun. note that to obtain the correct matrix, any function that is not needed should be set to zero. For example for two-dimensional problems, nz should be set to 1 and the functions cfun and ffun should be zero functions.

This uses natural ordering, first x direction, then y, then z mesh size h is uniform and determined by grid points in the x-direction.

PARAMETERS :

nx = number of points in x direction

ny = number of points in y direction

nz = number of points in z direction

a, ja, ia = resulting matrix in row-sparse format

iau = integer∗n containing the poisition of the diagonal element in the a, ja, ia structure

stencil = work array of size 7, used to store local stencils.

stencil [1:7] has the following meaning:

center point = stencil(1)

west point = stencil(2)

east point = stencil(3)

south point = stencil(4)

north point = stencil(5)

front point = stencil(6)

back point = stencil(7) / }

gen5loc (nx,ny,nz,nloc,riord,a,ja,ia,stencil) {

/ Local version of gen5pt. This subroutine generates only the nloc / rows of the matrix that are specified by the integer array riord / rows riord(1), ..., riord(nloc) of the matrix will be generated

/ and put in the a, ja, ia data structure. The column indices in / ja will still be given in the original labeling.

/** ON ENTRY :

same arguments as for gen5pt.

nloc = number of rows to be generated

riord = integer array of length nloc. the code will generate rows riord(1), ..., riord(nloc) of the matrix.

ON RETURN :

a, ja, ia = resulting matrix in CSR format.

Can be viewed as a rectangular matrix of size nloc x n, containing the rows riord(1),riord(2), ... riord(nloc) of A (in this order) in CSR format. / }

getsten (nx,ny,nz,kx,ky,kz,stencil,h) { / This subroutine calcultes the correct stencil values for / centered difference discretization of the elliptic operator

/**

L u = delx(a delx u) + dely (b dely u) + delz (c delz u) + delx (d u) + dely (e u) + delz(f u) + g u

For 2-D problems the discretization formula that is used is:

h**2 * Lu == a(i+1/2,j)*{u(i+1,j) - u(i,j)} + a(i-1/2,j)*{u(i-1,j) - u(i,j)} + b(i,j+1/2)*{u(i,j+1) - u(i,j)} + b(i,j-1/2)*{u(i,j-1) - u(i,j)} + (h/2)*d(i,j)*{u(i+1,j) - u(i-1,j)} + (h/2)*e(i,j)*{u(i,j+1) - u(i,j-1)} + (h/2)*e(i,j)*{u(i,j+1) - u(i,j-1)} + (h**2)*g(i,j)*u(i,j)

4.19.1.2 part0 (nx, ny, mpx, mpy, ovlp, lst, lstptr, iout)

This does a trivial mapping of a square grid into a virtual mpx x mpy processor grid. One way overlapping allowed: when assigning a block as a subdomain then 'ovlp' extra lines to the right and 'ovlp' extra lines to the top of the subrectangle being consired are added.

ON ENTRY :

nx, ny = number of mesh points in x and y directions respectively

mpx, mpy = number of processors in x and y directions respectively with (mpx .le. nx) and (mpy .le. ny)

ovlp = a nonnegative integer determing the amount of overlap (number of lines) in each direction. One=way overlap see above explanation.

ON RETURN :

lst = node per processor list. The nodes are listed contiguously from proc 1 to nproc = mpx*mpy.

lstptr = pointer array for array lst. list for proc. i starts at lstptr(i) and ends at lstptr(i+1)-1 in array lst.

iout = not used.

4.19.1.3 part1 (nx, ny, mpx, mpy, ovlp, lst, lstptr, iout)

does a trivial map of a square grid into a virtual mpx x mpy processor grid. Overlapping allowed: when assigning a block to a subdomain then ovlp extra lines are added to each of the four sides of the subrectangle being considered, outward.

ON ENTRY :

nx, ny = number of mesh points in x and y directions respectively

mpx, mpy = number of processors in x and y directions respectively with (mpx .le. nx) and (mpy .le. ny)

ovlp = a nonnegative integer determing the amount of overlap (number of lines) in each direction.

ON RETURN :

lst = node per processor list. The nodes are listed contiguously from proc 1 to nproc = mpx∗mpy.

lstptr = pointer array for array lst. list for proc. i starts at lstptr(i) and ends at lstptr(i+1)-1 in array lst.

iout = not used.

4.19.1.4 part2 (nx, ny, nz, mpx, mpy, mpz, ovlp, lst, lstptr, iout)

This does a trivial map of a square grid into a virtual mpx x mpy processor grid. Overlapping allowed: when assigning a block to a subdomain then ovlp extra lines are added to each of the four sides of the subrectangle being considered, outward.

ON ENTRY :

nx, ny,nz = number of mesh points in x, y and z directions respectively

mpx,mpy,mpz = number of processors in x, y and z directions respectively with (mpx .le. nx), (mpy .le. ny) and (mpz .le. nz)

ovlp = a nonnegative integer determing the amount of overlap (number of lines) in each direction.

ON RETURN :

lst = node per processor list. The nodes are listed contiguously from proc 1 to nproc = mpx∗mpy.

lstptr = pointer array for array lst. list for proc. i starts at lstptr(i) and ends at lstptr(i+1)-1 in array lst.

iout = not used.c

4.19.1.5 part_edge (nx, ny, nz, mpx, mpy, mpz, ovlp, lst, lstptr, iout)

This does a trivial edge based map of a square grid into a virtual mpx x mpy processor grid. Overlapping allowed: when assigning a block to a subdomain then ovlp extra lines are added to each of the four sides of the subrectangle being considered, outward.

ON ENTRY :

nx, ny,nz = number of mesh points in x, y and z directions respectively

mpx,mpy,mpz = number of processors in x, y and z directions respectively with (mpx .le. nx), (mpy .le. ny) and (mpz .le. nz)

ovlp = a nonnegative integer determing the amount of overlap (number of lines) in each direction.

ON RETURN :

lst = node per processor list. The nodes are listed contiguously from proc 1 to nproc = mpx∗mpy.

lstptr = pointer array for array lst. list for proc. i starts at lstptr(i) and ends at lstptr(i+1)-1 in array lst.

iout = not used.

4.20 gprecsol.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "../../INCLUDE/armsheads.h"
```

Defines

- #define **ZERO** 0.0
- #define **EPSILON** 1.0e-20
- #define **EPSMAC** 1.0e-16

Functions

- int **schgilusol** (double ∗b, double ∗x, **p4ptr** levmat, **Csr** bmat, **SchPilu** schpilu, **IterPar** ipar, FILE ∗fp)
- int **gilupgmr** (double ∗rhs, double ∗sol, **p4ptr** levmat, **Csr** bmat, **SchPilu** schpilu, **IterPar** ipar, FILE ∗fp)
- void **exbound** (**SchPilu** schpilu, double ∗x, double ∗w)
- int **schsgssol** (double ∗b, double ∗x, **p4ptr** levmat, **Csr** bmat, **SchPilu** schpilu, **IterPar** ipar, FILE ∗fp)
- void **sgslusol** (**SchPilu** schpilu, double ∗wk1, double ∗wk2)
- int **sgspgmr** (double ∗rhs, double ∗sol, **p4ptr** levmat, **Csr** bmat, **SchPilu** schpilu, **IterPar** ipar, FILE ∗fp)
- void **giluLsol** (**SchPilu** schpilu, double ∗b, double ∗y)
- void **giluUsol** (**SchPilu** schpilu, double ∗y, double ∗x)
- int **dlusol** (**SchPilu** schpilu, double ∗y, double ∗x)

4.20.1 Define Documentation

4.20.1.1 #define EPSILON 1.0e-20

4.20.1.2 #define EPSMAC 1.0e-16

4.20.1.3 #define ZERO 0.0

4.20.2 Function Documentation

4.20.2.1 int dlusol (SchPilu *schpilu*, double ∗ *y*, double ∗ *x*)

This routine solves the system (LU) x = y, given an LU decomposition of a matrix stored in (ilusch) in modified sparse row format

ON ENTRY :

y = the right-hand-side vector

schpilu = the LU matrix as provided from the ILU routines.

ON RETURN :

x = solution of LU x = y.

meth[0] rperm = 0:none 1:row (CTC)

meth[1] perm2 = 0:none 1:column (ILUTP)

meth[2] D1 = 0:none 1:scaling

meth[3] D2 = 0:none 1:scaling

NOTE :

Routine is in place: call lusolD(ilusch, x, x) will solve the system with rhs x and overwrite the result on x.

4.20.2.2 void exbound (SchPilu *schpilu*, double * *x*, double * *w*)

exchange boundary data in nonblocking mode */

4.20.2.3 void giluLsol (SchPilu *schpilu*, double * *b*, double * *y*)

This routine does the forward solve L y = b. Can be done in place.

ON ENTRY :

b = a vector

mata = the matrix (in **SparRow** form)

ON RETURN :

y = the product L^{-1} * b

4.20.2.4 int gilupgmr (double * *rhs*, double * *sol*, p4ptr *levmat*, Csr *bmat*, SchPilu *schpilu*, IterPar *ipar*, FILE * *fp*)

Distributed preconditioned GMRES algorithm that uses distributed ILU0 as a right preconditioner. gilupgmr is used on the expanded Schur Complement system, which is constructed by ARMS preconditioner

ON ENTRY :

rhs = real vector of length n containing the right hand side.

sol = real vector of length n containing an initial guess to the solution on input.

(levmat) = precon matrix in CSR format containing permuted, sorted Schur complement matrices at each level.

(ilusch) = LU factorization of input matrix from ILUT, in MSR format

(schpilu) = factors of distributed ILU0.

ON RETURN :

sol == contains an approximate solution (upon successful return).

int = 0 –> successful return.

int = 1 –> convergence not achieved in itmax iterations.

int = -1 –> the initial guess seems to be the exact solution (initial residual computed was zero)

int = -2 –> error in schurprod operation

WORK ARRAYS :

vv == work array of length [im+1][n] (used to store the Arnoldi basis)

hh == work array of length [im][im+1] (Householder matrix)

z == work array of length [n]

zz == work array of length [im][n] used for Flexible GMRES

SUBROUTINES CALLED :

lusolD : combined forward and backward solves BLAS1 routines.

PARAMETERS :

pgfpar[0] = epsilon on last level

pgipar[1] = Krylov space dimension on last level

pgipar[2] = maxits = max # iterations on last level

IMPORTANT NOTE :

maxits = 0 will still do at least one step. call lusolD instead if you need just a precon operation - see sol2.c

4.20.2.5 void giluUsol (SchPilu *schpilu*, double ∗ *y*, double ∗ *x*)

This routine does the backward solve U x = y. Can be done in place.

ON ENTRY :

y = a vector

schpilu = the matrix

ON RETURN :

x = the product U^{-1} ∗ y

4.20.2.6 int schgilusol (double ∗ *b*, double ∗ *x*, p4ptr *levmat*, Csr *bmat*, SchPilu *schpilu*, IterPar *ipar*, FILE ∗ *fp*)

| L 0 | | U L^{-1}F | M x = | | | | x = b | EU^{-1} I | | 0 S |

Solve subroutine used in preconditioning operation associated with the multi-level ilu preconditioner – It solves

where M is the multi-level block ILUT preconditioner constructed by arms.c. The last level is solved with distributed ILU0.

ON ENTRY :

b = double array of length n, store the right-hand side of arms x = b. Return unchanged.

(levmat) = permuted and sorted matrices for each level stored in **PerMat4** struct.

(ilschu) = matrix stored in IluSpar struct containing the ILU decomposition of the last level.

pgfpar[0] = epsilon on last level

pgfpar[1] = epsilon on intermediate levels

pgipar[0] = method - see below

pgipar[1] = Krylov dimension on last level

pgipar[2] = maximum # iterations on last level

pgipar[3] = Krylov dimension on intermediate levels

pgipar[4] = maximum # iterations on intermediate levels

pgipar[5] = print options

pgipar[6] = current level

ON RETURN :

x = double vector of length n, store the preconditioned residual, i.e., the solution of arms x = b.

METHODS :

FGMRES for outer iteration:

first digit: method / 10 =

0 no iteration on intermediate levels

1 iteration using RIGHT precon. GMRES between levels

2 iteration using LEFT precon. GMRES between levels

3 iteration using FGMRES between levels

second digit: method 10 =

0 no iteration on last level (one sweep of ILUTsolve)

1 iteration using RIGHT precon. GMRES on last level

2 iteration using LEFT precon. GMRES on last level

3 iteration using FGMRES on last level

100: DGMR for outer iteration, no iteration between levels

4.20.2.7 int schsgssol (double * *b*, double * *x*, p4ptr *levmat*, Csr *bmat*, SchPilu *schpilu*, IterPar *ipar*, FILE * *fp*)

| L 0 | | U L^{-1}F | M x = | | | | x = b | EU^{-1} I | | 0 S |

Solve subroutine used in preconditioning operation associated with the multi-level ilu preconditioner – It solves

where M is the multi-level block ILUT preconditioner constructed by arms.c. The last level is solved with symmetric Gauss-Seidel.

ON ENTRY :

b = double array of length n, store the right-hand side of arms x = b. Return unchanged.

(levmat) = permuted and sorted matrices for each level stored in **PerMat4** struct.

(ilschu) = matrix stored in IluSpar struct containing the ILU decomposition of the last level.

(schpilu) = factors for Gauss-Seidel preconditioner.

pgfpar[0] = epsilon on last level

pgfpar[1] = epsilon on intermediate levels

pgipar[0] = method - see below

pgipar[1] = Krylov dimension on last level

pgipar[2] = maximum # iterations on last level

pgipar[3] = Krylov dimension on intermediate levels

pgipar[4] = maximum # iterations on intermediate levels

pgipar[5] = print options

pgipar[6] = current level

ON RETURN :

x = double vector of length n, store the preconditioned residual, i.e., the solution of arms x = b.

METHODS :

FGMRES for outer iteration:

first digit: method / 10 =

0 no iteration on intermediate levels

1 iteration using RIGHT precon. GMRES between levels

2 iteration using LEFT precon. GMRES between levels

3 iteration using FGMRES between levels

second digit: method 10 =

0 no iteration on last level (one sweep of ILUTsolve)

1 iteration using RIGHT precon. GMRES on last level

2 iteration using LEFT precon. GMRES on last level

3 iteration using FGMRES on last level

100: DGMR for outer iteration, no iteration between levels

4.20.2.8 void sgslusol (SchPilu *schpilu*, double * *wk1*, double * *wk2*)

Solve with LU factors of Gauss-Seidel preconditioner

4.20.2.9 int sgspgmr (double * *rhs*, double * *sol*, p4ptr *levmat*, Csr *bmat*, SchPilu *schpilu*, IterPar *ipar*, FILE * *fp*)

Distributed preconditioned GMRES algorithm that uses distributed ILU0 as a right preconditioner. gilupgmr is used on the expanded Schur Complement system, which is constructed by ARMS preconditioner

ON ENTRY :

rhs = real vector of length n containing the right hand side.

sol = real vector of length n containing an initial guess to the solution on input.

(levmat) = precon matrix in CSR format containing permuted, sorted Schur complement matrices at each level.

(ilusch) = LU factorization of input matrix from ILUT

(schpilu) = factors for the Gauss-Seidel preconditioner

ON RETURN :

sol == contains an approximate solution (upon successful return).

int = 0 –> successful return.

int = 1 –> convergence not achieved in itmax iterations.

int = -1 –> the initial guess seems to be the exact solution (initial residual computed was zero)

int = -2 –> error in schurprod operation

WORK ARRAYS :

vv == work array of length [im+1][n] (used to store the Arnoldi basis)

hh == work array of length [im][im+1] (Householder matrix)

z == work array of length [n]

zz == work array of length [im][n] used for Flexible GMRES

SUBROUTINES CALLED :

lusolD : combined forward and backward solves BLAS1 routines.

PARAMETERS :

pgfpar[0] = epsilon on last level

pgipar[1] = Krylov space dimension on last level

pgipar[2] = maxits = max # iterations on last level

IMPORTANT NOTE:

maxits = 0 will still do at least one step. call lusolD instead if you need just a precon operation - see sol2.c

4.21 hash.c File Reference

```
#include "../../INCLUDE/psparslib.h"
```

Functions

- int **CreateHash** (**DistMatrix** dm)
- int **FreeHash** (**DistMatrix** dm)
- int **StoreInHash** (**DistMatrix** dm, int index, int value)
- int **GetHashValue** (**DistMatrix** dm, int index)
- int **PrintHash** (**DistMatrix** dm)

4.21.1 Function Documentation

4.21.1.1 int CreateHash (DistMatrix *dm*)

Create hash table according to data in structure dm

ON ENTRY & RETURN :

dm = local matrix contains a pointer to hash table

4.21.1.2 int FreeHash (DistMatrix *dm*)

Free the memory allocated for hash

ON ENTRY & RETURN :

dm = distributed local matrix handler, free memory taken up by the hash table member of the structure pointed by dm

4.21.1.3 int GetHashValue (DistMatrix *dm*, int *index*)

Retrive entry from hash

ON ENTRY :

dm = local matrix handler

index = index which is stored in the hash table

ON RETURN :

value = a pair (index,value) is stored in the hash table, value can be retrieved according to index

4.21.1.4 int PrintHash (DistMatrix *dm*)

Print entry stored in hash

ON ENTRY :

dm = distributed local matrix handler

ON RETURN :

print the pair (index,value)

4.21.1.5 int StoreInHash (DistMatrix *dm*, int *index*, int *value*)

Insert a pair (index,entry) into hash table

ON ENTRY :

dm = distributed local matrix handler

index = the index of a pair

value = the entry of a pair

ON RETURN :

4.22 heads.h File Reference

Data Structures

- struct **ILUTfac**
- struct **PerMat4**
- struct **SparRow**

Typedefs

- typedef **SparRow** ∗ **csptr**
- typedef **SparRow SparMat**
- typedef **PerMat4** ∗ **p4ptr**
- typedef **PerMat4 Per4Mat**
- typedef **ILUTfac** ∗ **ilutptr**
- typedef **ILUTfac IluSpar**

4.22.1 Typedef Documentation

4.22.1.1 typedef struct SparRow∗ csptr

4.22.1.2 typedef struct ILUTfac IluSpar

4.22.1.3 typedef struct ILUTfac∗ ilutptr

4.22.1.4 typedef struct PerMat4∗ p4ptr

4.22.1.5 typedef struct PerMat4 Per4Mat

4.22.1.6 typedef struct SparRow SparMat

4.23 ilu.c File Reference

```
#include "../../INCLUDE/psparslib.h"
```

Defines

- #define **min**(a, b) (a) > (b) ? (b) : (a)

Functions

- int **ilu0** (**DistMatrix** dm, **PreCon** precon, **PrePar** prepar)
- int **iluk** (**DistMatrix** dm, **PreCon** precon, **PrePar** prepar)
- int **ilut** (**DistMatrix** dm, **PreCon** precon, **PrePar** prepar)

4.23.1 Define Documentation

4.23.1.1 #define min(a, b) (a) > (b) ? (b) : (a)

4.23.2 Function Documentation

4.23.2.1 int ilu0 (DistMatrix *dm*, PreCon *precon*, PrePar *prepar*)

Construct local ilu0 preconditioner

ON ENTRY :

dm = distributed local matrix handler

prepar = parameters for constructing preconditioner

ON RETURN :

precon = preconditioner which is stored in MSR format

IMPORTANT :

It is assumed that the the elements in the input matrix are stored in such a way that in each row the lower part comes first and then the upper part. To get the correct ILU factorization, it is also necessary to have the elements of L sorted by increasing column number. It may therefore be necessary to sort the elements of a, ja, ia prior to calling ilu0. This can be achieved by transposing the matrix twice using csrcsc.

4.23.2.2 int iluk (DistMatrix *dm*, PreCon *precon*, PrePar *prepar*)

Construct local iluk preconditioner

ON ENTRY :

dm = distributed local matrix handler

prepar = parameters for constructing preconditioner

ON RETURN :

precon = preconditioner which is stored in MSR format

ierr = integer. Error message with the following meaning.

ierr = 0 –> successful return.

ierr > 0 –> zero pivot encountered at step number ierr.

ierr = -1 –> Error. input matrix may be wrong. (The elimination process has generated a row in L or U whose length is greater than n)

ierr = -2 –> Illegal value for lfil.

ierr = -3 –> zero row encountered in A or U

4.23.2.3 int ilut (DistMatrix *dm*, PreCon *precon*, PrePar *prepar*)

Construct local ilut preconditioner

ON ENTRY :

dm = distributed local matrix handler

prepar = parameters for constructing preconditioner

ON RETURN :

precon = preconditioner which is stored in MSR format

ierr = integer. Error message with the following meaning.

ierr = 0 –> successful return

ierr > 0 –> zero pivot encountered at step number ierr.

ierr = -1 –> Error input matrix may be wrong. (The elimination process has generated a row in L or U whose length is greater n)

ierr = -2 –> Illegal value for lfil.

ierr = -3 –> zero row encountered

4.24 iluNEW.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "../../INCLUDE/armsheads.h"
```

Functions

- int **ilutD** (**csptr** amat, double *droptol, int *lfil, **ilutptr** ilusch)

4.24.1 Function Documentation

4.24.1.1 int ilutD (csptr *amat*, double * *droptol*, int * *lfil*, ilutptr *ilusch*)

ILUT factorization with dual truncation.

ON ENTRY :

(amat) = Matrix stored in **SparRow** struct.

(ilusch) = Pointer to **ILUTfac** struct

lfil[5] = number nonzeros in L-part

lfil[6] = number nonzeros in U-part (lfil >= 0)

droptol[5] = threshold for dropping small terms in L during factorization.

droptol[6] = threshold for dropping small terms in U.

ON RETURN :

(ilusch) = Contains L and U factors in an LUfact struct.

Individual matrices stored in **SparRow** structs. On return matrices have C (0) indexing.

INTEGER VALUE RETURNED :

0 –> successful return.

1 –> Error. Input matrix may be wrong. (The elimination process has generated a row in L or U whose length is > n.)

2 –> Memory allocation error.

5 –> Illegal value for lfil or last.

6 –> zero row encountered.

WORK ARRAYS :

jw, jwrev = integer work arrays of length n

w = real work array of length n

4.25 ilutpC.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "../../INCLUDE/armsheads.h"
```

Functions

- int **ilutpC** (**csptr** amat, double *droptol, int *lfil, double permtol, int mband, **ilutptr** ilusch)

4.25.1 Function Documentation

4.25.1.1 int ilutpC (csptr *amat*, double * *droptol*, int * *lfil*, double *permtol*, int *mband*, ilutptr *ilusch*)

ILUT factorization with dual truncation.

ON ENTRY :

(amat) = Matrix stored in **SparRow** struct.

lfil[5] = number nonzeros in L-part

lfil[6] = number nonzeros in U-part (lfil >= 0)

droptol[5] = threshold for dropping small terms in L during factorization.

droptol[6] = threshold for dropping small terms in U

permtol = tolerance ratio used to determine whether or not to permute two columns At step i columns i and j are permuted when

abs(a(i,j))*permtol > abs(a(i,i))

[0 –> never permute; good values 0.1 to 0.01]

mband = permuting is done within a band extending to mband diagonals only.

mband = 0 –> no pivoting

mband = n –> pivot is searched in whole column

ON RETURN :

(ilusch) = Contains L and U factors in an LUfact struct

Individual matrices stored in **SparRow** structs On return matrices have C (0) indexing

iperm = reverse permutation array

INTEGER VALUES RETURNED :

0 –> successful return

1 –> Error. Input matrix may be wrong (The elimination process has generated a row in L or U whose length is > n.)

2 –> Memory allocation error

5 –> Illegal value for lfil

6 –> zero row encountered

WORK ARRAYS :

jw, jwrev = integer work arrays of length n

w = real work array of length n

4.26 indsetC.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "../../INCLUDE/armsheads.h"
```

Functions

- int **add2is** (int *last, int nod, int *iord, int *riord)
- int **add2com** (int *nback, int nod, int *iord, int *riord)
- int **indsetC** (**csptr** mat, int bsize, int *iord, int *nnod, double tol, int nbnd)
- int **weightsC** (**csptr** mat, double *w)

4.26.1 Function Documentation

4.26.1.1 int add2com (int * *nback*, int *nod*, int * *iord*, int * *riord*)

Adds element nod to independent set

4.26.1.2 int add2is (int * *last*, int *nod*, int * *iord*, int * *riord*)

Adds element nod to independent set

4.26.1.3 int indsetC (csptr *mat*, int *bsize*, int * *iord*, int * *nnod*, double *tol*, int *nbnd*)

Greedy algorithm for independent set ordering –

ON ENTRY :

(mat) = matrix in **SparRow** format

bsize = integer (input) the target size of each block. each block is of size >= bsize.

w = weight factors for the selection of the elements in the independent set. If w(i) is small i will be left for the vertex cover set.

tol = a tolerance for excluding a row from independent set.

ON RETURN :

iord = permutation array corresponding to the independent set ordering. Row number i will become row number iord[i] in permuted matrix.

nnod = (output) number of elements in the independent set.

The algorithm searches nodes in lexicographic order and groups the (BSIZE-1) nearest nodes of the current to form a block of size BSIZE. The current algorithm does not use values of the matrix.

4.26.1.4 int weightsC (csptr *mat*, double * *w*)

Defines weights based on diagonal dominance ratios

4.27 iters.c File Reference

```
#include "../../INCLUDE/psparslib.h"
#include <sys/time.h>
```

Defines

- #define **PRINT** 0

Functions

- void **fgmresd** (**DistMatrix** dm, **PreCon** precon, **IterPar** ipar, **Vec** rhs, **Vec** x)
- void **dgmresd** (**DistMatrix** dm, **PreCon** precon, **IterPar** ipar, **Vec** rhs, **Vec** x)
- void **bcgstabd** (**DistMatrix** dm, **PreCon** precon, **IterPar** ipar, **Vec** rhs, **Vec** x)

4.27.1 Define Documentation

4.27.1.1 #define PRINT 0

4.27.2 Function Documentation

4.27.2.1 void bcgstabd (DistMatrix *dm*, PreCon *precon*, IterPar *ipar*, Vec *rhs*, Vec *x*)

Bi Conjugate Gradient stabilized (BCGSTAB) with right-hand preconditioner M^{-1}. This is an improved BCG routine: (1) no matrix transpose is involved; (2) the convergence is smoother.

Preconditioned BCGSTAB – Distributed version

ON ENTRY :

dm = distributed matrix output from setup

precon = a pointer to precon strucutre

ipar = a pointer to the structure contains parameters(maxits,eps,etc.) related to iteration

rhs = right hand side vector

ON RETURN :

x = local solution vector handler

ipar = contains the number of iteration

4.27.2.2 void dgmresd (DistMatrix *dm*, PreCon *precon*, IterPar *ipar*, Vec *rhs*, Vec *x*)

Preconditioned GMRES based on **DGMR()** using reverse communication technique

ON ENTRY :

dm = distributed matrix output from setup

precon = a pointer to precon strucutre

rhs = right hand side vector

ipar = a pointer to the structure contains parameters(maxits,eps,etc.) related to iteration

ON RETURN :

x = local solution vector handler

ipar = contains the number of iteration

4.27.2.3 void fgmresd (DistMatrix *dm*, PreCon *precon*, IterPar *ipar*, Vec *rhs*, Vec *x*)

Distributed version of flexible GMRES.

ON ENTRY :

dm = distributed matrix output from setup

precon = a pointer to precon strucutre

ipar = a pointer to the structure contains parameters(maxits,eps,etc.) related to iteration

rhs = right hand side vector

ON RETURN :

x = local solution vector handler

ipar = contains the number of iteration

4.28 itersf.f File Reference

Functions

- **pddot** (n, x, ix, y, iy)

4.28.1 Function Documentation

4.28.1.1 pddot (n, x, ix, y, iy)

This interface of pddot is exactly the same as that of DDOT (or SDOT if real == real∗8) from BLAS-1. It should have same functionality as DDOT on a single processor machine. On a parallel/distributed environment, each processor can perform DDOT on the data it has, then perform a summation on all the partial results.

4.29 matrix.c File Reference

```
#include "../../INCLUDE/psparslib.h"
```

Functions

- int **GetValOfDim** (**DistMatrix** dm)
- int **CopyComm** (**DistMatrix** dm, **Vec** x)
- void **CreateMat** (**DistMatrix** ∗dm, char ∗type)
- void **DeleteMat** (**DistMatrix** ∗dm)
- void **CopyCsrToDm** (**DistMatrix** dm, double ∗a, int ∗ja, int ∗ia)
- int **GetValOfNnz** (**DistMatrix** dm)
- void **PrintMat** (**DistMatrix** dm, char ∗base)

Variables

- **_HashOps hops**

4.29.1 Function Documentation

4.29.1.1 int CopyComm (DistMatrix *dm*, Vec *x*)

Copy communication structure in DistMatrix to Vec structure

ON ENTRY :

dm = distributed local matrix handler

x = distributed local vector handler

ON RETURN :

x = distributed local vector handler. x contains communication structure copied from dm on return. the communication structure of dm is obtained from function setup

4.29.1.2 void CopyCsrToDm (DistMatrix *dm*, double ∗ *a*, int ∗ *ja*, int ∗ *ia*)

Copy CSR structure to distributed local matrix handler

ON ENTRY :

a,ja,ia = CSR format

dm = distributed local matrix handler

ON RETURN :

dm = dm handler contains CSR format copied from the tuple (a,ja,ia)

4.29.1.3 void CreateMat (DistMatrix ∗ *dm*, char ∗ *type*)

Create local matrix handler stored in CSR format

ON ENTRY :

dm = a pointer to distributed local matrix handler

type = the type of the distributed ojbect (matrix). Only "csr" is available in this version

ON RETURN :

dm = contains the pointer to local matrix handler

4.29.1.4 void DeleteMat (DistMatrix * *dm*)

Delete distributed local matrix handler

ON ENTRY :

dm = a pointer to local matrix handler

ON RETURN :

dm = free memory allocate for local matrix handler pointed by dm

4.29.1.5 int GetValOfDim (DistMatrix *dm*)

Return the dimension of the local matrix

ON ENTRY :

dm = dimension local matrix handler

ON RETURN :

return the dimension of local matrix

4.29.1.6 int GetValOfNnz (DistMatrix *dm*)

Return the number of none zero elements in the local matrix

ON ENTRY :

dm = distributed local matrix handler

ON RETURN :

nnz = the number of none zero entries in the local matrix stored in A member of the structure

4.29.1.7 void PrintMat (DistMatrix *dm*, char * *base*)

Output distributed matrix to a file with name 'base' Note processor i output data to the file 'base.i'

ON ENTRY :

dm = a distributed matrix

ON RETURN :

base = the base name of a file

4.29.2 Variable Documentation

4.29.2.1 _HashOps hops

Initial value:

```
 {
  CreateHash,
  StoreInHash,
  GetHashValue,
  FreeHash,
  PrintHash
}
```

4.30 matrops.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "../../INCLUDE/armsheads.h"
```

Functions

- void **Lsol** (**csptr** mata, double ∗b, double ∗x)
- void **Usol** (**csptr** mata, double ∗b, double ∗x)
- void **matvec** (**csptr** mata, double ∗x, double ∗y)
- void **matvecz** (**csptr** mata, double ∗x, double ∗y, double ∗z)
- int **descend** (**p4ptr** levmat, double ∗x, double ∗wk)
- int **ascend** (**p4ptr** levmat, double ∗wk, double ∗x)
- void **lusolD** (**ilutptr** ilusch, double ∗y, double ∗x)
- int **schurprod** (**p4ptr** levmat, **ilutptr** Smat, double ∗x, double ∗y)

4.30.1 Function Documentation

4.30.1.1 int ascend (p4ptr *levmat*, double ∗ *wk*, double ∗ *x*)

This function does the (block) backward substitution:

| | | | | |

| U L^{-1}F | | x1 | | wk1 |

| | | | = | |

| 0 S | | x2 | | wk2 |

| | | | | |

with x2 = S^{-1} wk2 [assumed to have been computed]

4.30.1.2 int descend (p4ptr *levmat*, double ∗ *x*, double ∗ *wk*)

This function does the (block) forward elimination in ARMS

| | | | | |

| L 0 | | wk1 | | x1 |

| | | | = | |

| EU^{-1} I | | wk2 | | x2 |

| | | | | |

wk is permuted

4.30.1.3 void Lsol (csptr *mata*, double * *b*, double * *x*)

This function does the forward solve L x = b. Can be done in place.

ON ENTRY :

mata = the matrix (in **SparRow** form)

b = a vector

ON RETURN :

x = the solution of L x = b

4.30.1.4 void lusolD (ilutptr *ilusch*, double * *y*, double * *x*)

Combination forward-backward substitution - This function solves the system (LU) x = y, given an LU decomposition of a matrix stored in (ilusch) in modified sparse row format

ON ENTRY :

ilusch = the LU matrix as provided from the ILU functions.

y = the right-hand-side vector

ON RETURN :

x = solution of LU x = y.

meth[0] rperm = 0:none 1:row (CTC)

meth[1] perm2 = 0:none 1:column (ILUTP)

meth[2] D1 = 0:none 1:scaling

meth[3] D2 = 0:none 1:scaling

NOTE :

Function is in place: call lusolD(ilusch, x, x) will solve the system with rhs x and overwrite the result on x.

4.30.1.5 void matvec (csptr *mata*, double * *x*, double * *y*)

This function does the matrix vector product y = A x.

ON ENTRY :

mata = the matrix (in **SparRow** form)

x = a vector

ON RETURN :

y = the product A * x

4.30.1.6 void matvecz (csptr *mata*, double * *x*, double * *y*, double * *z*)

This function does the matrix vector z = y - A x.

ON ENTRY :

mata = the matrix (in **SparRow** form)

x, y = two input vector

ON RETURN :

z = the result: y - A ∗ x

4.30.1.7 int schurprod (p4ptr *levmat*, ilutptr *Smat*, double ∗ *x*, double ∗ *y*)

Schurprod

| B F |

The Schur complement of | | is S = C - E B^{-1} F.

| E C |

This function computes y = S ∗ x using the different blocks of the original matrix. B^{-1} is approximated with (U^{-1})∗(L^{-1}).

ON ENTRY :

levmat = struct for the multilevel decomp.

Smat = struct for last level Schur complement matrix

x = vector of length nB

ON RETURN :

y = the product y = S ∗ x

4.30.1.8 void Usol (csptr *mata*, double ∗ *b*, double ∗ *x*)

This function does the backward solve U x = b. Can be done in place.

ON ENTRY :

mata = the matrix (in **SparRow** form)

b = a vector

ON RETURN :

x = the solution of U ∗ x = b

4.31 memus.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "../../INCLUDE/armsheads.h"
```

Functions

- int **cs_nnz** (**csptr** A)
- int **lev4_nnz** (**p4ptr** levmat, int *lev, FILE *ft)
- int **nnz_arms** (**p4ptr** levmat, **ilutptr** ilschu, int nlev, FILE *ft)

4.31.1 Function Documentation

4.31.1.1 int cs_nnz (csptr *A*)

Counts the number of nonzero elements in CSR matrix A

4.31.1.2 int lev4_nnz (p4ptr *levmat*, int * *lev*, FILE * *ft*)

Counts all nonzero elements in levmat struct – recursive

4.31.1.3 int nnz_arms (p4ptr *levmat*, ilutptr *ilschu*, int *nlev*, FILE * *ft*)

Computes and prints out total number of nonzero elements used in ARMS factorization

4.32 misc.f File Reference

Functions

- **rgg** (nm, n, a, b, alfr, alfi, beta, matz, z, ierr)
- **mgsro** (full, lda, n, m, ind, ops, vec, hh, ierr)

4.32.1 Function Documentation

4.32.1.1 mgsro (full, lda, n, m, ind, ops, vec, hh, ierr)

MGSRO – Modified Gram-Schmidt procedure with Selective Re- Orthogonalization

The indth vector of VEC is orthogonalized against the rest of the vectors.

The test for performing re-orthogonalization is performed for each indivadual vectors. If the cosine between the two vectors is greater than 0.99 (REORTH = 0.99**2), re-orthogonalization is performed. The norm of the 'new' vector is kept in variable NRM0, and updated after operating with each vector.

full – .ture. if it is necessary to orthogonalize the indth against all the vectors vec(:,1:ind-1), vec(:,ind+2:m) .false. only orthogonalize againt vec(:,1:ind-1)

lda – the leading dimension of VEC

n – length of the vector in VEC

m – number of vectors can be stored in VEC

ind – index to the vector to be changed

ops – operation counts

vec – vector of LDA X M storing the vectors

hh – coefficient of the orthogonalization

ierr – error code

0 : successful return

-1: zero input vector

-2: input vector contains abnormal numbers

-3: input vector is a linear combination of others

External routines used: real*8 pddot

4.32.1.2 rgg (nm, n, a, b, alfr, alfi, beta, matz, z, ierr)

Calls the recommended sequence of subroutines from the eigensystem subroutine package (eispack) to find the eigenvalues and eigenvectors (if desired) for the real general generalized eigenproblem ax = (lambda)bx.

ON ENTRY :

nm must be set to the row dimension of the two-dimensional array parameters as declared in the calling program dimension statement.

n is the order of the matrices a and b.

a contains a real general matrix.

b contains a real general matrix.

matz is an integer variable set equal to zero if only eigenvalues are desired. otherwise it is set to any non-zero integer for both eigenvalues and eigenvectors.

ON RETURN :

alfr and alfi contain the real and imaginary parts, respectively, of the numerators of the eigenvalues.

beta contains the denominators of the eigenvalues, which are thus given by the ratios (alfr+i∗alfi)/beta. complex conjugate pairs of eigenvalues appear consecutively with the eigenvalue having the positive imaginary part first.

z contains the real and imaginary parts of the eigenvectors if matz is not zero. if the j-th eigenvalue is real, the j-th column of z contains its eigenvector. if the j-th eigenvalue is complex with positive imaginary part, the j-th and (j+1)-th columns of z contain the real and imaginary parts of its eigenvector. the conjugate of this vector is the eigenvector for the conjugate eigenvalue.

ierr is an integer output variable set equal to an error completion code described in the documentation for qzit. the normal completion code is zero.

questions and comments should be directed to burton s. garbow, mathematics and computer science div, argonne national laboratory

4.33 piluNEW.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "../../INCLUDE/armsheads.h"
```

Functions

- int **qsplitC** (double ∗, int ∗, int, int)
- int **setupCS** (**csptr**, int)
- int **pilu** (**p4ptr** amat, **csptr** B, **csptr** C, double ∗droptol, int ∗lfil, **csptr** schur)

4.33.1 Function Documentation

4.33.1.1 int pilu (p4ptr *amat*, csptr *B*, csptr *C*, double ∗ *droptol*, int ∗ *lfil*, csptr *schur*)

Converted to C so that dynamic memory allocation may be implememted in order to have no dropping in block LU factors.

Partial block ILU factorization with dual truncation.

| B F | | L 0 | | U L^{-1} F |

| | = | | ∗ | |

| E C | | E U^{-1} I | | 0 S |

where B is a sub-matrix of dimension B->n.

ON ENTRY :

(amat) = Permuted matrix stored in a **PerMat4** struct on entry – Individual matrices stored in **SparRow** structs. On entry matrices have C (0) indexing. on return contains also L and U factors. Individual matrices stored in **SparRow** structs. On return matrices have C (0) indexing.

lfil[0] = number nonzeros in L-part

lfil[1] = number nonzeros in U-part

lfil[2] = number nonzeros in L^{-1} F

lfil[3] = not used

lfil[4] = number nonzeros in Schur complement

droptol[0] = threshold for dropping small terms in L during factorization.

droptol[1] = threshold for dropping small terms in U.

droptol[2] = threshold for dropping small terms in L^{-1} F during factorization.

droptol[3] = threshold for dropping small terms in E U^{-1} during factorization.

droptol[4] = threshold for dropping small terms in Schur complement after factorization is completed.

ON RETURN :

(schur) = contains the Schur complement matrix (S in above diagram) stored in **SparRow** struct with C (0) indexing.

INTEGER VALUES RETURNED:

0 –> successful return.

1 –> Error. Input matrix may be wrong. (The elimination process has generated a row in L or U whose length is > n.)

2 –> Memory allocation error.

5 –> Illegal value for lfil or last.

6 –> zero row in B block encountered.

7 –> zero row in [E C] encountered.

8 –> zero row in new Schur complement

WORK ARRAYS :

jw, jwrev = integer work arrays of length B->n.

w = real work array of length B->n.

jw2, jwrev2 = integer work arrays of length C->n.

w2 = real work array of length C->n.

4.33.1.2 int qsplitC (double * *a*, int * *ind*, int *n*, int *ncut*)

Does a quick-sort split of a real array.

ON ENTRY :

a[0 : (n-1)] is a real array

ON RETURN :

It is permuted such that its elements satisfy:

abs(a[i]) >= abs(a[ncut-1]) for i < ncut-1 and

abs(a[i]) <= abs(a[ncut-1]) for i > ncut-1

ind[0 : (n-1)] is an integer array permuted in the same way as a.

4.33.1.3 int setupCS (csptr *amat*, int *len*)

Initialize **SparRow** structs.

ON ENTRY :

(amat) = Pointer to a **SparRow** struct.

len = size of matrix

ON RETURN :

amat->n

->*nnzrow

->**ja

->**ma

INTEGER VALUES RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.34 precon.c File Reference

```
#include "../../INCLUDE/psparslib.h"
```

Functions

- int **CreatePrec** (**DistMatrix** dm, **PreCon** *precon, int type, **PrePar** prepar, **IterPar** ipar)
- int **DeletePrecIlu** (**PreCon** precon)
- int **DeletePrecArms** (**PreCon** precon)
- int **DeletePrecGilu** (**PreCon** precon)
- int **DeletePrec** (**PreCon** precon)
- int **GetIndSize** (**PreCon** precon)
- int **prec_ilu** (**DistMatrix** dm, **PreCon** precon, **PrePar** prepar, **IterPar** ipar)
- int **prec_arms** (**DistMatrix** dm, **PreCon** precon, **PrePar** prepar, **IterPar** ipar)
- int **prec_gilu** (**DistMatrix** dm, **PreCon** precon, **PrePar** prepar, **IterPar** ipar)
- int **sol0** (**DistMatrix** dm, **PreCon** precon, **IterPar** iterpar, **Vec** rhs, **Vec** sol)
- int **sol0p** (**PreCon** precon, **Vec** rhs, **Vec** sol, int rflag)
- int **sol0_ilu** (**DistMatrix** dm, **PreCon** precon, **IterPar** iterpar, **Vec** rhs, **Vec** sol)
- int **sol0_arms** (**DistMatrix** dm, **PreCon** precon, **IterPar** iterpar, **Vec** rhs, **Vec** sol)
- int **sol0_gilu** (**DistMatrix** dm, **PreCon** precon, **IterPar** iterpar, **Vec** rhs, **Vec** sol)
- int **sol0p_ilu** (**PreCon** precon, **Vec** rhs, **Vec** sol, int rflag)
- int **sol0p_arms** (**PreCon** precon, **Vec** rhs, **Vec** sol, int rflag)
- int **sol0_sgs** (**DistMatrix** dm, **PreCon** precon, **IterPar** iterpar, **Vec** rhs, **Vec** sol)

Variables

- **PREC prec_list** []
- **_PreCon _precon** []
- **PRECOND_ROUTINE sol0_dispatch** []

4.34.1 Function Documentation

4.34.1.1 int CreatePrec (DistMatrix *dm*, PreCon * *precon*, int *type*, PrePar *prepar*, IterPar *ipar*)

Create preconditioner handler

ON ENTRY :

type = the type of preconditioner. there are 8 types available in the release:

add_ilu0 additive schwarz preconditioner, ilu0 as local preconditioner

add_ilut additive schwarz preconditioner, ilut as local preconditioner

add_iluk additive schwarz preconditioner, iluk as local preconditioner

add_arms additive schwarz preconditioner, arms as local preconditioner

lsch_ilu0 left schur complement preconditioner, ilu0 as local preconditioner

lsch_ilut left schur complement preconditioner, ilut as local preconditioner

lsch_iluk left schur complement preconditioner, iluk as preconditioner

lsch_arms left schur complement preconditioner, arms as preconditioner

sch_gilu0 arms for interior nodes, ilu0 for interface nodes

sch_sgs arms for interior nodes, sgs for interface nodes

ON RETURN :

ierr = integer error flag: if ierr != 0 then error ocurred in preconditioner construction (see corresponding construction routine for the ierr value meaniong)

if ierr != 0 then

precon = preconditioning handler

else

precon = NULL

4.34.1.2 int DeletePrec (PreCon *precon*)

Delete preconditioner handler

ON ENTRY :

precon = preconditioning handler

ON RETURN :

free allocated memory for preconditioner pointed by precon

4.34.1.3 int DeletePrecArms (PreCon *precon*)

Delete preconditioner handler

ON ENTRY :

precon = preconditioning handler

ON RETURN :

free allocated memory for preconditioner ARMS pointed by precon

4.34.1.4 int DeletePrecGilu (PreCon *precon*)

Delete preconditioner handler

ON ENTRY :

precon = preconditioning handler

ON RETURN :

free allocated memory for preconditioner gilu0 or gilut pointed by precon

4.34.1.5 int DeletePrecIlu (PreCon *precon*)

Delete preconditioner handler

ON ENTRY :

precon = preconditioning handler

ON RETURN :

free allocated memory for preconditioner(ilu0,ilut or iluk) pointed by precon

4.34.1.6 int GetIndSize (PreCon *precon*)

Return the independent size (note, that is nbnd, for MSR)

ON ENTRY :

precon = a pointer to the preconditioner structure

ON RETURN :

the independent size

4.34.1.7 int prec_arms (DistMatrix *dm*, PreCon *precon*, PrePar *prepar*, IterPar *ipar*)

Additive schwarz preconditiner, arms as local preconditiner

ON ENTRY :

dm = distributed local matrix handler

prepar = parameters for preconditioner

ipar = parameters for iteration

ON RETURN :

precon = preconditioning handler contains arms preconditioner

4.34.1.8 int prec_gilu (DistMatrix *dm*, PreCon *precon*, PrePar *prepar*, IterPar *ipar*)

Global distributed iluX (gilu0 or gilut)

ON ENTRY :

dm = distributed local matrix handler

prepar = parameters for preconditioner

ipar = parameters for iteration

ON RETURN :

precon = preconditioning handler contains arms preconditioner

4.34.1.9 int prec_ilu (DistMatrix *dm*, PreCon *precon*, PrePar *prepar*, IterPar *ipar*)

Additive schwarz preconditiner, iluX(X stands for 0,t,k) as local preconditiner

ON ENTRY :

dm = distributed local matrix handler

prepar = parameters for preconditioner

ipar = parameters for iteration

ON RETURN :

precon = preconditioning handler contains ilu preconditioner

4.34.1.10 int sol0 (DistMatrix *dm*, PreCon *precon*, IterPar *iterpar*, Vec *rhs*, Vec *sol*)

Performs a forward followed by a backward solve for LU matrix pre.x = rhs

ON ENTRY :

dm = distributed local matrix handler

precon = preconditioner

rhs = right hand side vector handler

iterpar = parameters for iteration(not used in sol0)

ON RETURN :

sol = solution vector handler

4.34.1.11 int sol0_arms (DistMatrix *dm*, PreCon *precon*, IterPar *iterpar*, Vec *rhs*, Vec *sol*)

Forward followed by backward triangular solve for ARMS preconditioner. Preconditioner is stored in MultiSch format

ON ENTRY :

dm = distributed local matrix handler

precon = preconditioner stroed in MSR format

rhs = right hand side vector handler

iterpar = parameters for iteration(not used in sol0)

ON RETURN :

sol = solution vector handler

4.34.1.12 int sol0_gilu (DistMatrix *dm*, PreCon *precon*, IterPar *iterpar*, Vec *rhs*, Vec *sol*)

Forward followed by backward triangular solve for ARMS preconditioner with distributed ILU0 for the expanded Schur Complement. Preconditioner is stored in MultiSch format

ON ENTRY :

dm = distributed local matrix handler

precon = preconditioner stroed in MSR format

rhs = right hand side vector handler

iterpar = parameters for iteration(not used in sol0)

ON RETURN :

sol = solution vector handler

4.34.1.13 int sol0_ilu (DistMatrix *dm*, PreCon *precon*, IterPar *iterpar*, Vec *rhs*, Vec *sol*)

Performs a forward followed by a backward solve for LU matrix pre.x = rhs

ON ENTRY :

dm = distributed local matrix handler precon = preconditioner stroed in MSR format rhs = right hand side vector handler iterpar = parameters for iteration(not used in sol0)

ON RETURN :

sol = solution vector handler

4.34.1.14 int sol0_sgs (DistMatrix *dm*, PreCon *precon*, IterPar *iterpar*, Vec *rhs*, Vec *sol*)

Forward followed by backward triangular solve for ARMS preconditioner with Gauss-Seidel for the expanded Schur Complement. Preconditioner is stored in the MultiSch format.

ON ENTRY :

dm = distributed local matrix handler

precon = preconditioner stroed in MSR format

rhs = right hand side vector handler

iterpar = parameters for iteration(not used in sol0)

ON RETURN :

sol = solution vector handler

4.34.1.15 int sol0p (PreCon *precon*, Vec *rhs*, Vec *sol*, int *rflag*)

Performs a forward followed by a backward sweep for part of matrix

ON ENTRY :

precon = preconditioner

rhs = right hand side vector handler

rflag = indicates which part is to be solved

0 – interior part

1 – bottom part(interface part)

ON RETURN :

sol = solution vector handler

4.34.1.16 int sol0p_arms (PreCon *precon*, Vec *rhs*, Vec *sol*, int *rflag*)

Performs a forward followed by a backward solve for LU matrix as produced by arms $*$ does it for left upper or bottom parts of L $*$ and U.. Used in Schur complement techniques..

ON ENTRY :

precon = preconditioner stored in multisch format

rhs = right hand side vector handler

rflag = indicates which part is to be solved

0 – interior part

1 – bottom part(interface part)

ON RETURN :

sol = solution vector handler

4.34.1.17 int sol0p_ilu (PreCon *precon*, Vec *rhs*, Vec *sol*, int *rflag*)

Performs a forward followed by a backward solve for LU matrix as produced by ILU0, ILUK, ILUT.. does it for left upper or bottom parts of L and U.. Used in Schur complement techniques..

ON ENTRY :

precon = preconditioner stored in MSR format

rhs = right hand side vector handler

rflag = indicates which part is to be solved

0 – interior part

1 – bottom part(interface part)

ON RETURN :

sol = solution vector handler

4.34.2 Variable Documentation

4.34.2.1 _PreCon _precon[]

Initial value:

```
 {
  {add_ilu0,-1,-1,0,prec_ilu, NULL, sol0p_ilu, DeletePrecIlu, NULL},
  {add_ilut,-1,-1,0,prec_ilu, NULL, sol0p_ilu, DeletePrecIlu, NULL},
  {add_iluk,-1,-1,0,prec_ilu, NULL, sol0p_ilu, DeletePrecIlu, NULL},
  {add_arms,-1,-1,0,prec_arms, NULL, sol0p_arms, DeletePrecArms, NULL},
  {lsch_ilu0,-1,-1,1,prec_ilu, NULL, sol0p_ilu, DeletePrecIlu, NULL},
  {lsch_ilut,-1,-1,1,prec_ilu, NULL, sol0p_ilu, DeletePrecIlu, NULL},
  {lsch_iluk,-1,-1,1,prec_ilu, NULL, sol0p_ilu, DeletePrecIlu, NULL},
  {lsch_arms,-1,-1,1,prec_arms, NULL, sol0p_arms, DeletePrecArms, NULL},
  {rsch_ilu0,-1,-1,1,prec_ilu, NULL, sol0p_ilu, DeletePrecIlu, NULL},
  {rsch_ilut,-1,-1,1,prec_ilu, NULL, sol0p_ilu, DeletePrecIlu, NULL},
  {rsch_iluk,-1,-1,1,prec_ilu, NULL, sol0p_ilu, DeletePrecIlu, NULL},
  {rsch_arms,-1,-1,1,prec_arms, NULL, sol0p_arms, DeletePrecArms, NULL},
  {sch_gilu0,-1,-1,0,prec_gilu, NULL, NULL, DeletePrecGilu, NULL},
  {sch_sgs,-1,-1,0,prec_gilu, NULL, NULL, DeletePrecGilu, NULL},
}
```

4.34.2.2 PREC prec_list[]

Initial value:

```
 {
  ilu0,
  ilut,
  iluk,
  arms2
}
```

4.34.2.3 PRECOND_ROUTINE sol0_dispatch[]

Initial value:

```
 {

   sol0_ilu, sol0_ilu, sol0_ilu, sol0_arms, sol0_ilu, sol0_ilu, sol0_ilu,
   sol0_arms, sol0_ilu, sol0_ilu, sol0_ilu,  sol0_arms, sol0_gilu, sol0_sgs

}
```

4.35 psparslib.h File Reference

```
#include "data.h"
#include "base.h"
#include <math.h>
#include <stdlib.h>
```

Defines

- #define **ZERO** 0.0
- #define **EPSILON** 1.0e-20
- #define **EPSMAC** 1.0e-16
- #define **DSE** dse_
- #define **DPERM1** dperm1_
- #define **CSORT** csort_
- #define **DGMR** dgmr_

Functions

- int **CreateHash** (**DistMatrix** dm)
- int **StoreInHash** (**DistMatrix** dm, int index, int j)
- int **GetHashValue** (**DistMatrix** dm, int index)
- int **FreeHash** (**DistMatrix** dm)
- int **PrintHash** (**DistMatrix** dm)
- void **PARMS_Init** (int *, char ***)
- void **PARMS_Final** (void)
- void **CreateVec** (**Vec** *x)
- void **DeleteVec** (**Vec** *x)
- void **VecSetVal** (**Vec** x, double b)
- void **VecSetFunc** (**Vec** x, double lv, double rv, double(*func)(double y))
- void **VecAssign** (double *x, **Vec** vec)
- void **VecRand** (**Vec** x)
- double **VecNorm2** (**Vec** x)
- double **ResiNorm2** (**DistMatrix** dm, **Vec** sol, **Vec** rhs)
- void **CreateMat** (**DistMatrix** *dm, char *type)
- void **DeleteMat** (**DistMatrix** *dm)
- void **getmap** (**DistMatrix** dm, int *part, int *p, int *n)
- void **CopyCsrToDm** (**DistMatrix** dm, double *a, int *ja, int *ia)
- int **GetValOfDim** (**DistMatrix** dm)
- int **GetValOfNnz** (**DistMatrix** dm)
- void **bdry** (**DistMatrix** dm)
- void **setup** (**DistMatrix** dm)
- void **setuprhs** (**Vec** vec)
- void **multicD** (**DistMatrix** dm, int *mycol)
- void **amux** (**DistMatrix** dm, double *x, double *y)
- void **amuxe** (**DistMatrix** dm, double *x, double *y)
- void **amux1** (**DistMatrix** dm, double *x, double *y)
- void **consis** (**DistMatrix** dm, **Vec** s)

- void **amxdis** (**DistMatrix** dm, **Vec** x, **Vec** y)
- void **PrintMat** (**DistMatrix** dm, char *base)
- int **CreatePrec** (**DistMatrix** dm, **PreCon** *precon, int type, **PrePar** prepar, **IterPar** ipar)
- int **DeletePrecIlu** (**PreCon** precon)
- int **DeletePrecArms** (**PreCon** precon)
- int **DeletePrecGilu** (**PreCon** precon)
- int **DeletePrec** (**PreCon** precon)
- int **prec** (**DistMatrix** dm, **PreCon** precon, **PrePar** prepar)
- int **prec_ilu** (**DistMatrix** dm, **PreCon** precon, **PrePar** prepar, **IterPar** ipar)
- int **ilu0** (**DistMatrix** dm, **PreCon** precon, **PrePar** prepar)
- int **ilut** (**DistMatrix** dm, **PreCon** precon, **PrePar** prepar)
- int **iluk** (**DistMatrix** dm, **PreCon** precon, **PrePar** prepar)
- int **arms2** (**DistMatrix** dm, **PreCon** precon, **PrePar** prepar)
- int **schgilu0** (**csptr** schur, **Csr** xmat, **SchPilu** schpilu)
- int **prec_arms** (**DistMatrix** dm, **PreCon** precon, **PrePar** prepar, **IterPar** ipar)
- int **prec_gilu** (**DistMatrix** dm, **PreCon** precon, **PrePar** prepar, **IterPar** ipar)
- int **GetIndSize** (**PreCon** precon)
- int **lsch** (**DistMatrix** dm, **PreCon** precon, **IterPar** iterpar, **Vec** rhs, **Vec** sol)
- int **rsch** (**DistMatrix** dm, **PreCon** precon, **IterPar** iterpar, **Vec** rhs, **Vec** sol)
- int **sol0** (**DistMatrix** dm, **PreCon** precon, **IterPar** iterpar, **Vec** rhs, **Vec** x)
- int **sol0p** (**PreCon** precon, **Vec** rhs, **Vec** x, int rflag)
- int **sol0_ilu** (**DistMatrix** dm, **PreCon** precon, **IterPar** iterpar, **Vec** rhs, **Vec** x)
- int **sol0_arms** (**DistMatrix** dm, **PreCon** precon, **IterPar** iterpar, **Vec** rhs, **Vec** x)
- int **sol0_gilu** (**DistMatrix** dm, **PreCon** precon, **IterPar** iterpar, **Vec** rhs, **Vec** x)
- int **sol0_sgs** (**DistMatrix** dm, **PreCon** precon, **IterPar** iterpar, **Vec** rhs, **Vec** x)
- int **sol0p_ilu** (**PreCon** precon, **Vec** rhs, **Vec** x, int rflag)
- int **sol0p_arms** (**PreCon** precon, **Vec** rhs, **Vec** x, int rflag)
- int **armschsol** (double *rhs, double *vec, **p4ptr** levmat, **Csr** bmat, **SchPilu** schpilu, **IterPar** ipar, FILE *fp)
- int **pgmres** (struct **_p_DistMatrix** *dm, struct **_p_PreCon** *precon, **IterPar** ipar, **Vec** rhs, **Vec** sol)
- void **lusolD** (**ilutptr** ilusch, double *y, double *x)
- void **fgmresd** (**DistMatrix** dm, **PreCon** preops, **IterPar** ipar, **Vec** rhs, **Vec** x)
- void **dgmresd** (**DistMatrix** dm, **PreCon** precon, **IterPar** ipar, **Vec** rhs, **Vec** x)
- void **dbcgstab** (**DistMatrix** dm, **PreCon** precon, **IterPar** ipar, **Vec** rhs, **Vec** x)
- void **MSG_bdx_bsend** (**Vec** x)
- void **Mtype_Create** (**Vec** x)
- void **Mtype_Free** (**Vec** x)
- void **PrintVec** (**Vec** x, char *base)
- void **DSE** (int *n, int *ja, int *ia, int *ndom, int *riord, int *dom, int *idom, int *mask, int *jwk, int *link)
- void **DPERM1** (int *i1, int *i2, double *a, int *ja, int *ia, double *b, int *jb, int *ib, int *perm, int *ipos, int *job)
- void **CSORT** (int *, double *, int *, int *, int *, int *)
- void **DGMR** (int *, int *, int *, int *, int *, int *, int *, double *, double *, double *, int *, int *, double *, int *, int *, int *, int *, double *)
- double **dwalltime** ()

Variables

- **_HashOps hops**
- **PreCon prec_dispatch** []

4.35.1 Define Documentation

4.35.1.1 #define CSORT csort_

4.35.1.2 #define DGMR dgmr_

4.35.1.3 #define DPERM1 dperm1_

4.35.1.4 #define DSE dse_

4.35.1.5 #define EPSILON 1.0e-20

4.35.1.6 #define EPSMAC 1.0e-16

4.35.1.7 #define ZERO 0.0

4.35.2 Function Documentation

4.35.2.1 void amux (DistMatrix *dm*, double * *x*, double * *y*)

Dot-product version of local matrix-vector product

Distributed matrix and vector product for CSR format

y := dm∗x

ON ENTRY :

dm = distributed local matrix handler

x = local vector

ON RETURN :

y = output local vector

4.35.2.2 void amux1 (DistMatrix *dm*, double * *x*, double * *y*)

axpy version of local matarix-vector product (for transposed)

Distributed matrix and vector product for CSR format

y = y + dm∗x

ON ENTRY :

dm = distributed local matrix handler

x = external nodes

ON RETURN :

y = output local vector

4.35.2.3 void amuxe (DistMatrix *dm*, double ∗ *x*, double ∗ *y*)

Dot-product version of local RECTANGULAR matrix-vector product

Distributed external matrix and vector product for CSR format

y = dm∗x

ON ENTRY :

dm = distributed local matrix handler

x = external nodes

ON RETURN :

y = output local vector

4.35.2.4 void amxdis (DistMatrix *dm*, Vec *x*, Vec *y*)

Dot-product version of distributed matrix-vector product.

Distributed matrix and vector product for CSR format

y := dm∗x

ON ENTRY :

dm = distributed local matrix handler

x = input local vector

ON RETURN :

y = output local vector

4.35.2.5 int arms2 (DistMatrix *dm*, PreCon *precon*, PrePar *prepar*)

MULTI-LEVEL BLOCK ILUT PRECONDITIONER.

ON ENTRY :

dm = distributed matrix handler

precon = a pointer to struct **_p_PreCon** containg preconditioner

prepar = a pointer to struct **_p_PrePar** containing parameters for preconditioner.

Main parameters in prepar are ipar, lfil, ipar, droptol and tolind.

ipar[0:17] = integer array to store parameters for both arms construction (arms2) and iteration (armsol2).

ipar[0]:=nlev. number of levels (reduction processes). see also "on return" below.

ipar[1]:=bsize. Dimension of the blocks. In this version, bsize is only a target block size. The size of each block can vary and is >= bsize.

ipar[2]:=iout if (iout > 0) statistics on the run are printed to FILE ∗ft

The following are not used by arms2 – but should set before calling the preconditionning operation armsol2: ipar[3]:= Krylov subspace dimension for last level ipar[3] == 0 means only backward/forward solve is performed on last level.

ipar[4]:= maximum # iterations on last level

ipar[5-9] NOT used [reserved for later use] - must be set to zero. [see however a special use of ipar[5] in fgmresC.]

The following set method options for arms2. Their default values can all be set to zero if desired.

ipar[10-13] == meth[0:3] = method flags for interlevel blocks

ipar[14-17] == meth[0:3] = method flags for last level block

- with the following meaning

meth[0] permutations of rows 0:no 1: yes. affects rperm NOT USED IN THIS VERSION ∗∗ enter 0.. Data: rperm

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 no 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 similarly for meth[14], ..., meth[17] all transformations related to parametres in meth[∗] (permutation, scaling,..) are applied to the matrix before processing it

droptol = Threshold parameters for dropping elements in ILU factorization.

droptol[0:4] = related to the multilevel block factorization

droptol[5:5] = related to ILU factorization of last block. This flexibility is more than is really needed. one can use a single parameter for all. it is preferable to use one value for droptol[0:4] and another (smaller) for droptol[5:6] droptol[0] = threshold for dropping in L [B]. See **piluNEW.c**:

droptol[1] = threshold for dropping in U [B].

droptol[2] = threshold for dropping in L^{-1} F

droptol[3] = threshold for dropping in E U^{-1}

droptol[4] = threshold for dropping in Schur complement

droptol[5] = threshold for dropping in L in last block [see **ilutpC.c**] droptol[6] = threshold for dropping in U in last block [see **ilutpC.c**]

lfil = lfil[0:6] is an array containing the fill-in parameters. similar explanations as above, namely:

lfil[0] = amount of fill-in kept in L [B].

lfil[1] = amount of fill-in kept in U [B]. etc..

tolind = tolerance parameter used by the indset function. a row is not accepted into the independent set if the ∗relative∗ diagonal tolerance is below tolind. see indset function for details. Good values are between 0.05 and 0.5 – larger values tend to be better for harder problems.

4.35.2.6 int armschsol (double ∗ *rhs*, double ∗ *vec*, p4ptr *levmat*, Csr *bmat*, SchPilu *schpilu*, IterPar *ipar*, FILE ∗ *fp*)

4.35.2.7 void bdry (DistMatrix *dm*)

C version of bdry which determines the boundary / interface information for symmetric patterns. this version supports both FORTRAN and C format

ON ENTRY :

dm = a pointer to structure **_p_DistMatrix** (distributed matrix handler)

ON RETURN :

dm = distributed local matrix handler with the following structure members being set up: dm->perm – mapping "local-to-new_local" label ordering of local nodes, where "new_local" means ordering interface after external.

dm->comm->proc – array of neighbor ranks and number of nodes from each neighbor.

dm->comm->ix – list of nodes, in global numbering, to send to each neighbor

dm->comm->ipr – pointer into list ix, which is stored neighbor by neighbor.

dm->comm->nbnd – pointer to inter-domain interface variables

dm->comm->nproc – number of neighbors

4.35.2.8 void consis (DistMatrix *dm*, Vec *s*)

4.35.2.9 void CopyCsrToDm (DistMatrix *dm*, double * *a*, int * *ja*, int * *ia*)

Copy CSR structure to distributed local matrix handler

ON ENTRY :

a,ja,ia = CSR format

dm = distributed local matrix handler

ON RETURN :

dm = dm handler contains CSR format copied from the tuple (a,ja,ia)

4.35.2.10 int CreateHash (DistMatrix *dm*)

Create hash table according to data in structure dm

ON ENTRY & RETURN :

dm = local matrix contains a pointer to hash table

4.35.2.11 void CreateMat (DistMatrix * *dm*, char * *type*)

Create local matrix handler stored in CSR format

ON ENTRY :

dm = a pointer to distributed local matrix handler

type = the type of the distributed ojbect (matrix). Only "csr" is available in this version

ON RETURN :

dm = contains the pointer to local matrix handler

4.35.2.12 int CreatePrec (DistMatrix *dm*, PreCon * *precon*, int *type*, PrePar *prepar*, IterPar *ipar*)

Create preconditioner handler

ON ENTRY :

type = the type of preconditioner. there are 8 types available in the release:

add_ilu0 additive schwarz preconditioner, ilu0 as local preconditioner

add_ilut additive schwarz preconditioner, ilut as local preconditioner

add_iluk additive schwarz preconditioner, iluk as local preconditioner

add_arms additive schwarz preconditioner, arms as local preconditioner

lsch_ilu0 left schur complement preconditioner, ilu0 as local preconditioner

lsch_ilut left schur complement preconditioner, ilut as local preconditioner

lsch_iluk left schur complement preconditioner, iluk as preconditioner

lsch_arms left schur complement preconditioner, arms as preconditioner

sch_gilu0 arms for interior nodes, ilu0 for interface nodes

sch_sgs arms for interior nodes, sgs for interface nodes

ON RETURN :

ierr = integer error flag: if ierr != 0 then error ocurred in preconditioner construction (see corresponding construction routine for the ierr value meaniong)

if ierr != 0 then

precon = preconditioning handler

else

precon = NULL

4.35.2.13 void CreateVec (Vec * *x*)

Create a Vec handler

ON ENTRY :

x = a pointer to Vec handler

ON RETURN :

x = allocate memory for x and initialize the members of the structure

4.35.2.14 void CSORT (int *, double *, int *, int *, int *, int *)

4.35.2.15 void dbcgstab (DistMatrix *dm*, PreCon *precon*, IterPar *ipar*, Vec *rhs*, Vec *x*)

4.35.2.16 void DeleteMat (DistMatrix * *dm*)

Delete distributed local matrix handler

ON ENTRY :

dm = a pointer to local matrix handler

ON RETURN :

dm = free memory allocate for local matrix handler pointed by dm

4.35.2.17 int DeletePrec (PreCon *precon*)

Delete preconditioner handler

ON ENTRY :

precon = preconditioning handler

ON RETURN :

free allocated memory for preconditioner pointed by precon

4.35.2.18 int DeletePrecArms (PreCon *precon*)

Delete preconditioner handler

ON ENTRY :

precon = preconditioning handler

ON RETURN :

free allocated memory for preconditioner ARMS pointed by precon

4.35.2.19 int DeletePrecGilu (PreCon *precon*)

Delete preconditioner handler

ON ENTRY :

precon = preconditioning handler

ON RETURN :

free allocated memory for preconditioner gilu0 or gilut pointed by precon

4.35.2.20 int DeletePrecIlu (PreCon *precon*)

Delete preconditioner handler

ON ENTRY :

precon = preconditioning handler

ON RETURN :

free allocated memory for preconditioner(ilu0,ilut or iluk) pointed by precon

4.35.2.21 void DeleteVec (Vec $*$ *x*)

Free memory allocated for local vector handler

ON ENTRY :

x = a pointer to the Vec handler

ON RETURN :

free the memory allocated for the handler pointed by x

4.35.2.22 void DGMR (int *, int *, int *, int *, int *, int *, int *, double *, double *, double *, int *, int *, double *, int *, int *, int *, int *, double *)

4.35.2.23 void dgmresd (DistMatrix *dm*, PreCon *precon*, IterPar *ipar*, Vec *rhs*, Vec *x*)

Preconditioned GMRES based on **DGMR()** using reverse communication technique

ON ENTRY :

dm = distributed matrix output from setup

precon = a pointer to precon strucutre

rhs = right hand side vector

ipar = a pointer to the structure contains parameters(maxits,eps,etc.) related to iteration

ON RETURN :

x = local solution vector handler

ipar = contains the number of iteration

4.35.2.24 void DPERM1 (int * *i1*, int * *i2*, double * *a*, int * *ja*, int * *ia*, double * *b*, int * *jb*, int * *ib*, int * *perm*, int * *ipos*, int * *job*)

4.35.2.25 void DSE (int * *n*, int * *ja*, int * *ia*, int * *ndom*, int * *riord*, int * *dom*, int * *idom*, int * *mask*, int * *jwk*, int * *link*)

4.35.2.26 double dwalltime (void)

Wallclock timer function

dwaltime : time in seconds since 00:00:00 UTC, Jan. 1st, 1970

4.35.2.27 void fgmresd (DistMatrix *dm*, PreCon *precon*, IterPar *ipar*, Vec *rhs*, Vec *x*)

Distributed version of flexible GMRES.

ON ENTRY :

dm = distributed matrix output from setup

precon = a pointer to precon strucutre

ipar = a pointer to the structure contains parameters(maxits,eps,etc.) related to iteration

rhs = right hand side vector

ON RETURN :

x = local solution vector handler

ipar = contains the number of iteration

4.35.2.28 int FreeHash (DistMatrix *dm*)

Free the memory allocated for hash

ON ENTRY & RETURN :

dm = distributed local matrix handler, free memory taken up by the hash table member of the structure pointed by dm

4.35.2.29 int GetHashValue (DistMatrix *dm*, int *index*)

Retrive entry from hash

ON ENTRY :

dm = local matrix handler

index = index which is stored in the hash table

ON RETURN :

value = a pair (index,value) is stored in the hash table, value can be retrieved according to index

4.35.2.30 int GetIndSize (PreCon *precon*)

Return the independent size (note, that is nbnd, for MSR)

ON ENTRY :

precon = a pointer to the preconditioner structure

ON RETURN :

the independent size

4.35.2.31 void getmap (DistMatrix *dm*, int * *part*, int * *p*, int * *n*)

Set up map from global label to processors

ON ENTRY :

part = sequence of nodes (variables) (nodes) is stored processor by processor

p = p[i] points to the beginning of processor (domain) i in array part

n = the total number of nodes (variables)

ON RETURN : dm = distributed local matrix handler with the following structure members being set up: assigned dm->map – mapping "node to processor list that will need this node" assigned dm->node – mapping "local-to-global" label of a node

4.35.2.32 int GetValOfDim (DistMatrix *dm*)

Return the dimension of the local matrix

ON ENTRY :

dm = dimension local matrix handler

ON RETURN :

return the dimension of local matrix

4.35.2.33 int GetValOfNnz (DistMatrix *dm*)

Return the number of none zero elements in the local matrix

ON ENTRY :

dm = distributed local matrix handler

ON RETURN :

nnz = the number of none zero entries in the local matrix stored in A member of the structure

4.35.2.34 int ilu0 (DistMatrix *dm*, PreCon *precon*, PrePar *prepar*)

Construct local ilu0 preconditioner

ON ENTRY :

dm = distributed local matrix handler

prepar = parameters for constructing preconditioner

ON RETURN :

precon = preconditioner which is stored in MSR format

IMPORTANT :

It is assumed that the the elements in the input matrix are stored in such a way that in each row the lower part comes first and then the upper part. To get the correct ILU factorization, it is also necessary to have the elements of L sorted by increasing column number. It may therefore be necessary to sort the elements of a, ja, ia prior to calling ilu0. This can be achieved by transposing the matrix twice using csrcsc.

4.35.2.35 int iluk (DistMatrix *dm*, PreCon *precon*, PrePar *prepar*)

Construct local iluk preconditioner

ON ENTRY :

dm = distributed local matrix handler

prepar = parameters for constructing preconditioner

ON RETURN :

precon = preconditioner which is stored in MSR format

ierr = integer. Error message with the following meaning.

ierr = 0 –> successful return.

ierr > 0 –> zero pivot encountered at step number ierr.

ierr = -1 –> Error. input matrix may be wrong. (The elimination process has generated a row in L or U whose length is greater than n)

ierr = -2 –> Illegal value for lfil.

ierr = -3 –> zero row encountered in A or U

4.35.2.36 int ilut (DistMatrix *dm*, PreCon *precon*, PrePar *prepar*)

Construct local ilut preconditioner

ON ENTRY :

dm = distributed local matrix handler

prepar = parameters for constructing preconditioner

ON RETURN :

precon = preconditioner which is stored in MSR format

ierr = integer. Error message with the following meaning.

ierr = 0 –> successful return

ierr > 0 –> zero pivot encountered at step number ierr.

ierr = -1 –> Error input matrix may be wrong. (The elimination process has generated a row in L or U whose length is greater n)

ierr = -2 –> Illegal value for lfil.

ierr = -3 –> zero row encountered

4.35.2.37 int lsch (DistMatrix *dm*, PreCon *precon*, IterPar *iterpar*, Vec *rhs*, Vec *sol*)

Approximate LU-SCHUR left preconditioner

This is a preconditioner for the global linear system which is based on solving approximately the Schur complement system.

More precisely, an approximation to the local Schur complement is obtained (implicitly) in the form of an LU factorization from the LU factorization L_i U_i of the local matrix A_i. Solving with this approximation amounts to simply doing the forward and backward solves with the bottom part of L_i and U_i only (corresponding to the interace variables). This is done using a special version of the subroutine lusol0 called lusol0_p.

Then it is possible to solve for an approximate Schur complement system using GMRES on the global approximate Schur complement system (which is preconditionied by the diagonal blocks represented by these restricted LU matrices).

4.35.2.38 void lusolD (ilutptr *ilusch*, double * *y*, double * *x*)

Combination forward-backward substitution - This function solves the system (LU) x = y, given an LU decomposition of a matrix stored in (ilusch) in modified sparse row format

ON ENTRY :

ilusch = the LU matrix as provided from the ILU functions.

y = the right-hand-side vector

ON RETURN :

x = solution of LU x = y.

meth[0] rperm = 0:none 1:row (CTC)

meth[1] perm2 = 0:none 1:column (ILUTP)

meth[2] D1 = 0:none 1:scaling

meth[3] D2 = 0:none 1:scaling

NOTE :

Function is in place: call lusolD(ilusch, x, x) will solve the system with rhs x and overwrite the result on x.

4.35.2.39 void MSG_bdx_bsend (Vec *x*)

Interface information exchange general routine with non-blocking

ON ENTRY :

x = distribued local vector handler

ON RETURN :

x = distributed local vector contains external variables received from adjacent processors

4.35.2.40 void Mtype_Create (Vec *x*)

Create derived data types used for MPI

ON ENTRY :

x = distributed local vector handler

ON RETURN :

x = local vector handler which contains derived data types on return

4.35.2.41 void Mtype_Free (Vec *x*)

Free derived data type contained in distributed local vector x

ON ENTRY :

x = local vector handler

ON RETURN :

x = derived data types contained in x is freed on return

4.35.2.42 void multicD (DistMatrix *dm*, int * *mycol*)

multicoloring subdomains

ON ENTRY :

dm = distributed local matrix handler

ON RETURN :

mycol = color assigned to my processor

4.35.2.43 void PARMS_Final (void)

Exit PARMS and MPI environment

4.35.2.44 void PARMS_Init (int * *argc*, char *** *argv*)

Initialize the PARMS environment

ON ENTRY :

argc, argv = have the same meaning as that in the main function The function also sets up various linked lists for mat objects

4.35.2.45 int pgmres (struct _p_DistMatrix * *dm*, struct _p_PreCon * *precon*, IterPar *ipar*, Vec *rhs*, Vec *sol*)

4.35.2.46 int prec (DistMatrix *dm*, PreCon *precon*, PrePar *prepar*)

4.35.2.47 int prec_arms (DistMatrix *dm*, PreCon *precon*, PrePar *prepar*, IterPar *ipar*)

Additive schwarz preconditiner, arms as local preconditiner

ON ENTRY :

dm = distributed local matrix handler

prepar = parameters for preconditioner

ipar = parameters for iteration

ON RETURN :

precon = preconditioning handler contains arms preconditioner

4.35.2.48 int prec_gilu (DistMatrix *dm*, PreCon *precon*, PrePar *prepar*, IterPar *ipar*)

Global distributed iluX (gilu0 or gilut)

ON ENTRY :

dm = distributed local matrix handler

prepar = parameters for preconditioner

ipar = parameters for iteration

ON RETURN :

precon = preconditioning handler contains arms preconditioner

4.35.2.49 int prec_ilu (DistMatrix *dm*, PreCon *precon*, PrePar *prepar*, IterPar *ipar*)

Additive schwarz preconditiner, iluX(X stands for 0,t,k) as local preconditiner

ON ENTRY :

dm = distributed local matrix handler

prepar = parameters for preconditioner

ipar = parameters for iteration

ON RETURN :

precon = preconditioning handler contains ilu preconditioner

4.35.2.50 int PrintHash (DistMatrix *dm*)

Print entry stored in hash

ON ENTRY :

dm = distributed local matrix handler

ON RETURN :

print the pair (index,value)

4.35.2.51 void PrintMat (DistMatrix *dm*, char * *base*)

Output distributed matrix to a file with name 'base' Note processor i output data to the file 'base.i'

ON ENTRY :

dm = a distributed matrix

ON RETURN :

base = the base name of a file

4.35.2.52 void PrintVec (Vec *x*, char * *base*)

Output vector to a file with name 'base' Note processor i output data to the file 'base.i'

ON ENTRY :

x = distributed vector

base = base name of a file

ON RETURN :

output local vector to the file 'base.i'. the format is: local_label global_label value

4.35.2.53 double ResiNorm2 (DistMatrix *dm*, Vec *sol*, Vec *rhs*)

Return residual norm ||rhs-dm*sol||

ON ENTRY :

dm = local matrix

sol = solution vector

rhs = right hand side

ON RETURN :

return residual norm

4.35.2.54 int rsch (DistMatrix *dm*, PreCon *precon*, IterPar *iterpar*, Vec *rhs*, Vec *sol*)

Approximate LU-SCHUR left preconditioner

This is a preconditioner for the global linear system which is based on solving approximately the Schur complement system.

More precisely, an approximation to the local Schur complement is obtained (implicitly) in the form of an LU factorization from the LU factorization L_i U_i of the local matrix A_i. Solving with this approximation amounts to simply doing the forward and backward solves with the bottom part of L_i and U_i only (corresponding to the interace variables). This is done using a special version of the subroutine lusol0 called lusol0_p.

Then it is possible to solve for an approximate Schur complement system using GMRES on the global approximate Schur complement system (which is preconditionied by the diagonal blocks represented by these restricted LU matrices).

4.35.2.55 int schgilu0 (csptr *schur*, Csr *xmat*, SchPilu *schpilu*)

Do distributed ILU(0) construction on the expanded Schur Complement System (containing local and interdomain interface nodes)

ON ENTRY :

schur = last reduced matrix

xmat = external matrix

ON RETURN :

schpilu = distributed ILU(0) matrix for last reduced system

4.35.2.56 void setup (DistMatrix *dm*)

C version of setup which sets up the local data structure for the sparse matrix.

ON ENTRY :

dm = distributed local matrix handler which contains a pointer to the local matrix (A member of the structure) which can be stored in FORTRAN or C style

ON RETURN :

dm = member A of the structure has been changed: interior nodes first followed by interface nodes. X member of the structure is created which is the matrix for external parts. permutation array and communication structure are also set up

NOTE :

Both A and X members of the structure are stored in C style on return. both A and X are sorted in increasing column numbers.

The following members of DistMatrix have been changed or assigned:

dm->comm->ix – changed to local numbering

dm->node – permuted for interface to follow internal nodes

dm->comm->nl – number of local nodes plus external nodes to be received.

dm->A – rows of local matrix A are sorted (see above)

dm->X – matrix connecting external nodes to local ones.

4.35.2.57 void setuprhs (Vec *vec*)

set up righ-hand-side

ON ENTRY :

vec = local vector

ON RETURN :

vec = components of vector has been permuted

4.35.2.58 int sol0 (DistMatrix *dm*, PreCon *precon*, IterPar *iterpar*, Vec *rhs*, Vec *sol*)

Performs a forward followed by a backward solve for LU matrix pre.x = rhs

ON ENTRY :

dm = distributed local matrix handler

precon = preconditioner

rhs = right hand side vector handler

iterpar = parameters for iteration(not used in sol0)

ON RETURN :

sol = solution vector handler

4.35.2.59 int sol0_arms (DistMatrix *dm*, PreCon *precon*, IterPar *iterpar*, Vec *rhs*, Vec *sol*)

Forward followed by backward triangular solve for ARMS preconditioner. Preconditioner is stored in MultiSch format

ON ENTRY :

dm = distributed local matrix handler

precon = preconditioner stroed in MSR format

rhs = right hand side vector handler

iterpar = parameters for iteration(not used in sol0)

ON RETURN :

sol = solution vector handler

4.35.2.60 int sol0_gilu (DistMatrix *dm*, PreCon *precon*, IterPar *iterpar*, Vec *rhs*, Vec *sol*)

Forward followed by backward triangular solve for ARMS preconditioner with distributed ILU0 for the expanded Schur Complement. Preconditioner is stored in MultiSch format

ON ENTRY :

dm = distributed local matrix handler

precon = preconditioner stroed in MSR format

rhs = right hand side vector handler

iterpar = parameters for iteration(not used in sol0)

ON RETURN :

sol = solution vector handler

4.35.2.61 int sol0_ilu (DistMatrix *dm*, PreCon *precon*, IterPar *iterpar*, Vec *rhs*, Vec *sol*)

Performs a forward followed by a backward solve for LU matrix pre.x = rhs

ON ENTRY :

dm = distributed local matrix handler precon = preconditioner stroed in MSR format rhs = right hand side vector handler iterpar = parameters for iteration(not used in sol0)

ON RETURN :

sol = solution vector handler

4.35.2.62 int sol0_sgs (DistMatrix *dm*, PreCon *precon*, IterPar *iterpar*, Vec *rhs*, Vec *sol*)

Forward followed by backward triangular solve for ARMS preconditioner with Gauss-Seidel for the expanded Schur Complement. Preconditioner is stored in the MultiSch format.

ON ENTRY :

dm = distributed local matrix handler

precon = preconditioner stroed in MSR format

rhs = right hand side vector handler

iterpar = parameters for iteration(not used in sol0)

ON RETURN :

sol = solution vector handler

4.35.2.63 int sol0p (PreCon *precon*, Vec *rhs*, Vec *sol*, int *rflag*)

Performs a forward followed by a backward sweep for part of matrix

ON ENTRY :

precon = preconditioner

rhs = right hand side vector handler

rflag = indicates which part is to be solved

0 – interior part

1 – bottom part(interface part)

ON RETURN :

sol = solution vector handler

4.35.2.64 int sol0p_arms (PreCon *precon*, Vec *rhs*, Vec *sol*, int *rflag*)

Performs a forward followed by a backward solve for LU matrix as produced by arms ∗ does it for left upper or bottom parts of L ∗ and U.. Used in Schur complement techniques..

ON ENTRY :

precon = preconditioner stored in multisch format

rhs = right hand side vector handler

rflag = indicates which part is to be solved

0 – interior part

1 – bottom part(interface part)

ON RETURN :

sol = solution vector handler

4.35.2.65 int sol0p_ilu (PreCon *precon*, Vec *rhs*, Vec *sol*, int *rflag*)

Performs a forward followed by a backward solve for LU matrix as produced by ILU0, ILUK, ILUT.. does it for left upper or bottom parts of L and U.. Used in Schur complement techniques..

ON ENTRY :

precon = preconditioner stored in MSR format

rhs = right hand side vector handler

rflag = indicates which part is to be solved

0 – interior part

1 – bottom part(interface part)

ON RETURN :

sol = solution vector handler

4.35.2.66 int StoreInHash (DistMatrix *dm*, int *index*, int *value*)

Insert a pair (index,entry) into hash table

ON ENTRY :

dm = distributed local matrix handler

index = the index of a pair

value = the entry of a pair

ON RETURN :

4.35.2.67 void VecAssign (double ∗ *x*, Vec *vec*)

Assign double array pointed by x to vec

ON ENTRY : x = double array vec = distributed vec

ON RETURN : vec = contains the values stored in x

4.35.2.68 double VecNorm2 (Vec *x*)

Return the norm of the global vector

ON ENTRY :

x = distributed local vector handler

ON RETURN :

return the norm of global vector

NOTE : T

This a collective function, all processors which contain parts of global vector should call this function

4.35.2.69 void VecRand (Vec *x*)

4.35.2.70 void VecSetFunc (Vec *x*, double *lv*, double *rv*, double(∗ *func*)(double y))

Set every components of vector x to the value returned by function func whose domain is [lv, rv]

ON ENTRY :

x = distributed local vector

lv,rv = the domain of function y, [lv,rv]

y = a function pointer to a function, such as sin, cos, etc.

ON RETURN :

x[i] = i∗(rv-lv)/n, n is the dimension of the vector

4.35.2.71 void VecSetVal (Vec *x*, double *b*)

Set all components of vector x to value b

ON ENTRY :

b – scalar, a value should be set to every component of vector

ON RETURN :

x – distributed local vector

4.35.3 Variable Documentation

4.35.3.1 _HashOps hops

4.35.3.2 PreCon prec_dispatch[]

4.36 qzhes.f File Reference

Functions

- **qzhes** (nm, n, a, b, matz, z)

4.36.1 Function Documentation

4.36.1.1 qzhes (nm, n, a, b, matz, z)

This is the first step of the qz algorithm for solving generalized matrix eigenvalue problems, siam j. numer. anal. 10, 241-256(1973) by moler and stewart.

This subroutine accepts a pair of real general matrices and reduces one of them to upper hessenberg form and the other to upper triangular form using orthogonal transformations. it is usually followed by qzit, qzval and, possibly, qzvec.

ON ENTRY :

nm must be set to the row dimension of two-dimensional array parameters as declared in the calling program dimension statement.

n is the order of the matrices.

a contains a real general matrix.

b contains a real general matrix.

matz should be set to .true. if the right hand transformations are to be accumulated for later use in computing eigenvectors, and to .false. otherwise.

ON RETURN :

a has been reduced to upper hessenberg form. the elements below the first subdiagonal have been set to zero.

b has been reduced to upper triangular form. the elements below the main diagonal have been set to zero.

z contains the product of the right hand transformations if matz has been set to .true. otherwise, z is not referenced.

questions and comments should be directed to burton s. garbow, mathematics and computer science div, argonne national laboratory

4.37 qzit.f File Reference

Functions

- **qzit** (nm, n, a, b, eps1, matz, z, ierr)

4.37.1 Function Documentation

4.37.1.1 qzit (nm, n, a, b, eps1, matz, z, ierr)

4.38 qzval.f File Reference

Functions

- **qzval** (nm, n, a, b, alfr, alfi, beta, matz, z)

4.38.1 Function Documentation

4.38.1.1 qzval (nm, n, a, b, alfr, alfi, beta, matz, z)

This is the third step of the qz algorithm for solving generalized matrix eigenvalue problems, siam j. numer. anal. 10, 241-256(1973) by moler and stewart.

This subroutine accepts a pair of real matrices, one of them in quasi-triangular form and the other in upper triangular form. it reduces the quasi-triangular matrix further, so that any remaining 2-by-2 blocks correspond to pairs of complex eigenvalues, and returns quantities whose ratios give the generalized eigenvalues. it is usually preceded by qzhes and qzit and may be followed by qzvec.

ON ENTRY :

nm must be set to the row dimension of two-dimensional array parameters as declared in the calling program dimension statement.

n is the order of the matrices.

a contains a real upper quasi-triangular matrix.

b contains a real upper triangular matrix. in addition, location b(n,1) contains the tolerance quantity (epsb) computed and saved in qzit.

matz should be set to .true. if the right hand transformations are to be accumulated for later use in computing eigenvectors, and to .false. otherwise.

z contains, if matz has been set to .true., the transformation matrix produced in the reductions by qzhes and qzit, if performed, or else the identity matrix. if matz has been set to .false., z is not referenced.

ON RETURN :

a has been reduced further to a quasi-triangular matrix in which all nonzero subdiagonal elements correspond to pairs of complex eigenvalues.

b is still in upper triangular form, although its elements have been altered. b(n,1) is unaltered.

alfr and alfi contain the real and imaginary parts of the diagonal elements of the triangular matrix that would be obtained if a were reduced completely to triangular form by unitary transformations. non-zero values of alfi occur in pairs, the first member positive and the second negative.

beta contains the diagonal elements of the corresponding b, normalized to be real and non-negative. the generalized eigenvalues are then the ratios ((alfr+i∗alfi)/beta).

z contains the product of the right hand transformations (for all three steps) if matz has been set to .true.

questions and comments should be directed to burton s. garbow, mathematics and computer science div, argonne national laboratory

4.39 qzvec.f File Reference

Functions

- **qzvec** (nm, n, a, b, alfr, alfi, beta, z)

4.39.1 Function Documentation

4.39.1.1 qzvec (nm, n, a, b, alfr, alfi, beta, z)

This is the optional fourth step of the qz algorithm for solving generalized matrix eigenvalue problems, siam j. numer. anal. 10, 241-256(1973) by moler and stewart.

this subroutine accepts a pair of real matrices, one of them in quasi-triangular form (in which each 2-by-2 block corresponds to a pair of complex eigenvalues) and the other in upper triangular form. it computes the eigenvectors of the triangular problem and transforms the results back to the original coordinate system. it is usually preceded by qzhes, qzit, and qzval.

on input

nm must be set to the row dimension of two-dimensional array parameters as declared in the calling program dimension statement.

n is the order of the matrices.

a contains a real upper quasi-triangular matrix.

b contains a real upper triangular matrix. in addition, location b(n,1) contains the tolerance quantity (epsb) computed and saved in qzit.

alfr, alfi, and beta are vectors with components whose ratios ((alfr+i∗alfi)/beta) are the generalized eigenvalues. they are usually obtained from qzval.

z contains the transformation matrix produced in the reductions by qzhes, qzit, and qzval, if performed. if the eigenvectors of the triangular problem are desired, z must contain the identity matrix.

on output

a is unaltered. its subdiagonal elements provide information about the storage of the complex eigenvectors.

b has been destroyed.

alfr, alfi, and beta are unaltered.

z contains the real and imaginary parts of the eigenvectors. if alfi(i) .eq. 0.0, the i-th eigenvalue is real and the i-th column of z contains its eigenvector. if alfi(i) .ne. 0.0, the i-th eigenvalue is complex. if alfi(i) .gt. 0.0, the eigenvalue is the first of a complex pair and the i-th and (i+1)-th columns of z contain its eigenvector. if alfi(i) .lt. 0.0, the eigenvalue is the second of a complex pair and the (i-1)-th and i-th columns of z contain the conjugate of its eigenvector. each eigenvector is normalized so that the modulus of its largest component is 1.0 .

questions and comments should be directed to burton s. garbow, mathematics and computer science div, argonne national laboratory

4.40 schgilu0.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../../INCLUDE/psparslib.h"
```

Defines

- #define **PERMTOL** 0.1
- #define **MBLOC** 5

Functions

- int **schgilu0** (**csptr** schur, **Csr** xmat, **SchPilu** schpilu)

4.40.1 Define Documentation

4.40.1.1 #define MBLOC 5

4.40.1.2 #define PERMTOL 0.1

4.40.2 Function Documentation

4.40.2.1 int schgilu0 (csptr *schur*, Csr *xmat*, SchPilu *schpilu*)

Do distributed ILU(0) construction on the expanded Schur Complement System (containing local and interdomain interface nodes)

ON ENTRY :

schur = last reduced matrix

xmat = external matrix

ON RETURN :

schpilu = distributed ILU(0) matrix for last reduced system

4.41 setpar.c File Reference

```
#include "../../INCLUDE/psparslib.h"
```

Defines

- #define **BUFLEN** 100

Functions

- void **set_def_params** (PrePar prepar, IterPar ipar)
- void **setpar** (char *filename, char *matrix, int *iov, int *scale, int *unsym, int *method, PrePar prepar, IterPar ipar, DistMatrix dm)
- int **assignprecon** (char *precon_str, DistMatrix dm)

4.41.1 Define Documentation

4.41.1.1 #define BUFLEN 100

4.41.2 Function Documentation

4.41.2.1 int assignprecon (char * *precon_str*, DistMatrix *dm*)

4.41.2.2 void set_def_params (PrePar *prepar*, IterPar *ipar*)

4.41.2.3 void setpar (char * *filename*, char * *matrix*, int * *iov*, int * *scale*, int * *unsym*, int * *method*, PrePar *prepar*, IterPar *ipar*, DistMatrix *dm*)

4.42 sets.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "../../INCLUDE/armsheads.h"
```

Functions

- int **setupCS** (**csptr** amat, int len)
- int **cleanCS** (**csptr** amat)
- int **setupP4** (**p4ptr** amat, int Bn, int Cn, **csptr** F, **csptr** E)
- int **cleanP4** (**p4ptr** amat)
- int **setupILUT** (**ilutptr** amat, int len)
- int **cleanILUT** (**ilutptr** amat, int indic)
- int **cleanARMS** (**p4ptr** amat, **ilutptr** cmat)
- int **CSRcs** (int n, double ∗a, int ∗ja, int ∗ia, **csptr** bmat)
- int **cscpy** (**csptr** amat, **csptr** bmat)
- int **csSplit4** (**csptr** amat, int bsize, int csize, **csptr** B, **csptr** F, **csptr** E, **csptr** C)

4.42.1 Function Documentation

4.42.1.1 int cleanARMS (p4ptr *amat*, ilutptr *cmat*)

Free up memory allocated for entire ARMS preconditioner.

ON ENTRY :

(amat) = Pointer to a Per4Mat struct.

(cmat) = Pointer to a IluSpar struct.

4.42.1.2 int cleanCS (csptr *amat*)

Free up memory allocated for **SparRow** structs.

ON ENTRY :

(amat) = Pointer to a **SparRow** struct.

len = size of matrix

4.42.1.3 int cleanILUT (ilutptr *amat*, int *indic*)

Free up memory allocated for IluSpar structs.

ON ENTRY :

(amat) = Pointer to a IluSpar struct.

indic = indicator for number of levels. indic=0 -> zero level

4.42.1.4 int cleanP4 (p4ptr *amat*)

Free up memory allocated for Per4Mat structs.

ON ENTRY :

(amat) = Pointer to a Per4Mat struct

4.42.1.5 int cscpy (csptr *amat*, csptr *bmat*)

Copy amat in **SparRow** struct to bmat in **SparRow** struct

ON ENTRY :

(amat) = Matrix stored in **SparRow** format

ON RETURN :

(bmat) = Matrix stored as **SparRow** struct containing a copy of amat

INTEGER VALUE RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.42.1.6 int CSRcs (int *n*, double ∗ *a*, int ∗ *ja*, int ∗ *ia*, csptr *bmat*)

Convert CSR matrix to **SparRow** struct

ON ENTRY :

a, ja, ia = Matrix stored in CSR format (with FORTRAN indexing).

ON RETURN :

(bmat) = Matrix stored as **SparRow** struct.

INTEGER VALUE RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.42.1.7 int csSplit4 (csptr *amat*, int *bsize*, int *csize*, csptr *B*, csptr *F*, csptr *E*, csptr *C*)

Convert permuted csrmat struct to **PerMat4** struct

- matrix already permuted

ON ENTRY :

(amat) = Matrix stored in **SparRow** format. Internal pointers (and associated memory) destroyed before return.

ON RETURN :

B, E, F, C = 4 blocks in

| B F |

Amat = | |

| E C |

INTEGER VALUE RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.42.1.8 int setupCS (csptr *amat*, int *len*)

Initialize **SparRow** structs.

ON ENTRY :

(amat) = Pointer to a **SparRow** struct.

len = size of matrix

ON RETURN :

amat->n

->*nnzrow

->**ja

->**ma

INTEGER VALUES RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.42.1.9 int setupILUT (ilutptr *amat*, int *len*)

Allocate pointers for **ILUTfac** structs.

ON ENTRY :

(amat) = Pointer to a **ILUTfac** struct.

len = size of L U blocks

ON RETURN :

amat->L for each block: amat->M->n

->U->nnzrow

->ja

->ma

->rperm (if meth[0] > 0)

->perm2 (if meth[1] > 0)

->D1 (if meth[2] > 0)

->D2 (if meth[3] > 0)

Permutation arrays are initialized to the identity. Scaling arrays are initialized to 1.0.

INTEGER VALUE RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.42.1.10 int setupP4 (p4ptr *amat*, int *Bn*, int *Cn*, csptr *F*, csptr *E*)

Initialize **PerMat4** struct given the F, E, blocks.

ON ENTRY :

(amat) = Pointer to a **PerMat4** struct.

Bn = size of B block

Cn = size of C block

F, E = the two blocks to be assigned to srtruct - without the

ON RETURN :

amat->L for each block: amat->M->n

->U ->nnzrow

->E ->ja

->F ->ma

->perm

->rperm (if meth[1] > 0)

->D1 (if meth[2] > 0)

->D2 (if meth[3] > 0)

Scaling arrays are initialized to 1.0.

INTEGER VALUE RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.43 setup.c File Reference

```
#include "../../INCLUDE/psparslib.h"
```

Functions

- void **getmap** (**DistMatrix** dm, int *part, int *p, int *n)
- void **bdry** (**DistMatrix** dm)
- void **setup** (**DistMatrix** dm)
- void **setuprhs** (**Vec** vec)
- void **multicD** (**DistMatrix** dm, int *mycol)

4.43.1 Function Documentation

4.43.1.1 void bdry (DistMatrix *dm*)

C version of bdry which determines the boundary / interface information for symmetric patterns. this version supports both FORTRAN and C format

ON ENTRY :

dm = a pointer to structure **_p_DistMatrix** (distributed matrix handler)

ON RETURN :

dm = distributed local matrix handler with the following structure members being set up: dm->perm – mapping "local-to-new_local" label ordering of local nodes, where "new_local" means ordering interface after external.

dm->comm->proc – array of neighbor ranks and number of nodes from each neighbor.

dm->comm->ix – list of nodes, in global numbering, to send to each neighbor

dm->comm->ipr – pointer into list ix, which is stored neighbor by neighbor.

dm->comm->nbnd – pointer to inter-domain interface variables

dm->comm->nproc – number of neighbors

4.43.1.2 void getmap (DistMatrix *dm*, int * *part*, int * *p*, int * *n*)

Set up map from global label to processors

ON ENTRY :

part = sequence of nodes (variables) (nodes) is stored processor by processor

p = p[i] points to the beginning of processor (domain) i in array part

n = the total number of nodes (variables)

ON RETURN : dm = distributed local matrix handler with the following structure members being set up: assigned dm->map – mapping "node to processor list that will need this node" assigned dm->node – mapping "local-to-global" label of a node

4.43.1.3 void multicD (DistMatrix *dm*, int * *mycol*)

multicoloring subdomains

ON ENTRY :

dm = distributed local matrix handler

ON RETURN :

mycol = color assigned to my processor

4.43.1.4 void setup (DistMatrix *dm*)

C version of setup which sets up the local data structure for the sparse matrix.

ON ENTRY :

dm = distributed local matrix handler which contains a pointer to the local matrix (A member of the structure) which can be stored in FORTRAN or C style

ON RETURN :

dm = member A of the structure has been changed: interior nodes first followed by interface nodes. X member of the structure is created which is the matrix for external parts. permutation array and communication structure are also set up

NOTE :

Both A and X members of the structure are stored in C style on return. both A and X are sorted in increasing column numbers.

The following members of DistMatrix have been changed or assigned:

dm->comm->ix – changed to local numbering

dm->node – permuted for interface to follow internal nodes

dm->comm->nl – number of local nodes plus external nodes to be received.

dm->A – rows of local matrix A are sorted (see above)

dm->X – matrix connecting external nodes to local ones.

4.43.1.5 void setuprhs (Vec *vec*)

set up righ-hand-side

ON ENTRY :

vec = local vector

ON RETURN :

vec = components of vector has been permuted

4.44 skitc.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "../../INCLUDE/armsheads.h"
```

Functions

- int **qsplitC** (double *a, int *ind, int n, int ncut)
- int **SparTran** (**csptr** amat, **csptr** bmat, int job, int flag)
- int **rpermC** (**csptr** mat, int *perm)
- int **cpermC** (**csptr** mat, int *perm)
- int **dpermC** (**csptr** mat, int *perm)
- int **roscalC** (**csptr** mata, double *diag, int nrm)
- int **coscalC** (**csptr** mata, double *diag, int nrm)
- void **dscale** (int n, double *dd, double *x, double *y)
- void **printmat** (FILE *ft, **csptr** A, int i0, int i1)

4.44.1 Function Documentation

4.44.1.1 int coscalC (csptr *mata*, double * *diag*, int *nrm*)

This routine scales each column of mata so that the norm is 1.

ON ENTRY :

mata = the matrix (in **SparRow** form)

nrm = type of norm

0 (infty), 1 or 2

ON RETURN :

diag = diag[j] = 1/norm(row[j])

0 –> normal return

j –> column j is a zero column

4.44.1.2 int cpermC (csptr *mat*, int * *perm*)

This subroutine permutes the columns of a matrix in **SparRow** format. cperm computes B = A P, where P is a permutation matrix. that maps column j into column perm(j), i.e., on return The permutation P is defined through the array perm: for each j, perm[j] represents the destination column number of column number j.

ON ENTRY :

(mat) = a matrix stored in **SparRow** format.

ON RETURN :

(mat) = A P stored in **SparRow** format.

INTEGER VALUE RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.44.1.3 int dpermC (csptr *mat*, int * *perm*)

This subroutine permutes the rows and columns of a matrix in **SparRow** format. dperm computes B = P^T A P, where P is a permutation matrix.

ON ENTRY :

(amat) = a matrix stored in **SparRow** format.

ON RETURN :

(amat) = P^T A P stored in **SparRow** format.

INTEGER VALUE RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.44.1.4 void dscale (int *n*, double * *dd*, double * *x*, double * *y*)

Computes y == DD * x scales the vector x by the diagonal dd - output in y

4.44.1.5 void printmat (FILE * *ft*, csptr *A*, int *i0*, int *i1*)

To dump rows i0 to i1 of matrix for debugging purposes

4.44.1.6 int qsplitC (double * *a*, int * *ind*, int *n*, int *ncut*)

Does a quick-sort split of a real array.

ON ENTRY :

a[0 : (n-1)] is a real array

ON RETURN :

It is permuted such that its elements satisfy:

abs(a[i]) >= abs(a[ncut-1]) for i < ncut-1 and

abs(a[i]) <= abs(a[ncut-1]) for i > ncut-1

ind[0 : (n-1)] is an integer array permuted in the same way as a.

4.44.1.7 int roscalC (csptr *mata*, double * *diag*, int *nrm*)

This routine scales each row of mata so that the norm is 1.

ON ENTRY :

mata = the matrix (in **SparRow** form)

nrm = type of norm

0 (infty), 1 or 2

ON RETURN :

diag = diag[j] = 1/norm(row[j])

0 –> normal return

j –> row j is a zero row

4.44.1.8 int rpermC (csptr *mat*, int * *perm*)

This subroutine permutes the rows of a matrix in **SparRow** format. rperm computes B = P A where P is a permutation matrix. The permutation P is defined through the array perm: for each j, perm[j] represents the destination row number of row number j.

ON ENTRY :

(amat) = a matrix stored in **SparRow** format.

ON RETURN :

(amat) = P A stored in **SparRow** format.

INTEGER VALUE RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.44.1.9 int SparTran (csptr *amat*, csptr *bmat*, int *job*, int *flag*)

Finds the transpose of a matrix stored in **SparRow** format.

ON ENTRY :

(amat) = a matrix stored in **SparRow** format.

job = integer to indicate whether to fill the values (job.eq.1) of the matrix (bmat) or only the pattern.

flag = integer to indicate whether the matrix has been filled

0 - no filled

1 - filled

ON RETURN :

(bmat) = the transpose of (mata) stored in **SparRow** format.

INTEGER VALUE RETURNED :

0 –> successful return.

1 –> memory allocation error.

4.45 skitf.f File Reference

Functions

- **dperm** (nrow, a, ja, ia, ao, jao, iao, perm, qperm, job)
- **dperm1** (i1, i2, a, ja, ia, b, jb, ib, perm, ipos, job)
- **dperm2** (i1, i2, a, ja, ia, b, jb, ib, perm, rperm, istart, ipos, job)
- **multic** (n, ja, ia, ncol, kolrs, il, iord, maxcol, ierr)
- **xtrows** (i1, i2, a, ja, ia, ao, jao, iao, iperm, job)
- **amub** (nrow, ncol, job, a, ja, ia, b, jb, ib, c, jc, ic, nzmax, iw, ierr)
- **readmt** (nmax, nzmax, job, iounit, a, ja, ia, rhs, nrhs, guesol, nrow, ncol, nnz, title, key, type, ierr)
- **roscal** (nrow, job, nrm, a, ja, ia, diag, b, jb, ib, ierr)
- **coscal** (nrow, job, nrm, a, ja, ia, diag, b, jb, ib, ierr)
- **rnrms** (nrow, nrm, a, ja, ia, diag)
- **cnrms** (nrow, nrm, a, ja, ia, diag)
- **diamua** (nrow, job, a, ja, ia, diag, b, jb, ib)
- **amudia** (nrow, job, a, ja, ia, diag, b, jb, ib)
- **aplb** (nrow, ncol, job, a, ja, ia, b, jb, ib, c, jc, ic, nzmax, iw, ierr)
- **csrcsc** (n, job, ipos, a, ja, ia, ao, jao, iao)
- **csrcsc2** (n, n2, job, ipos, a, ja, ia, ao, jao, iao)
- **atmux** (n, x, y, a, ja, ia)
- **atmuxr** (m, n, x, y, a, ja, ia)
- **readmt_c** (nmax, nzmax, job, fname, a, ja, ia, rhs, nrhs, guesol, nrow, ncol, nnz, title, key, type, ierr)
- **csort** (n, a, ja, ia, iwork, values)
- **dvperm** (n, x, perm)
- **ivperm** (n, ix, perm)
- **rperm** (nrow, a, ja, ia, ao, jao, iao, perm, job)
- **cperm** (nrow, a, ja, ia, ao, jao, iao, perm, job)
- **wreadmtc** (nmax, nzmax, job, fname, length, a, ja, ia, rhs, nrhs, guesol, nrow, ncol, nnz, title, key, type, ierr)
- **readmtc** (nmax, nzmax, job, fname, a, ja, ia, rhs, nrhs, guesol, nrow, ncol, nnz, title, key, type, ierr)
- **dblstr** (n, ja, ia, ip1, ip2, nfirst, riord, ndom, map, mapptr, mask, levels, iwk)
- **rdis** (n, ja, ia, ndom, map, mapptr, mask, levels, size, iptr)
- **dse2way** (n, ja, ia, ip1, ip2, nfirst, riord, ndom, dom, idom, mask, jwk, link)
- **dse** (n, ja, ia, ndom, riord, dom, idom, mask, jwk, link)
- **BFS** (n, ja, ia, nfirst, iperm, mask, maskval, riord, levels, nlev)
- **add_lvst** (istart, iend, nlev, riord, ja, ia, mask, maskval)
- **stripes** (nlev, riord, levels, ip, map, mapptr, ndom)
- **stripes0** (ip, nlev, il, ndom, iptr)
- **perphn** (n, ja, ia, init, mask, maskval, nlev, riord, levels)
- **mapper4** (n, ja, ia, ndom, nodes, levst, marker, link)
- **get_domns2** (ndom, nodes, link, levst, riord, iptr)
- **mindom** (n, ndom, levst, link)
- **add_lk** (new, nod, idom, ndom, lkend, levst, link, nodes, marker)
- **find_ctr** (n, nsiz, ja, ia, init, mask, maskval, riord, levels, center, iwk)
- **rversp** (n, riord)

4.45.1 Function Documentation

4.45.1.1 add_lk (new, nod, idom, ndom, lkend, levst, link, nodes, marker)

Adds from head –

adds one entry (new) to linked list and ipdates everything.

new = node to be added

nod = current number of marked nodes

idom = domain to which new is to be added

ndom = total number of domains

lkend= location of end of structure (link and nodes)

levst= pointer array for link, nodes

link = link array

nodes= nodes array –

marker = marker array == if marker(k) =0 then node k is not assigned yet.

4.45.1.2 add_lvst (istart, iend, nlev, riord, ja, ia, mask, maskval)

Adds one level set to the previous sets.. span all nodes of previous mask

4.45.1.3 amub (nrow, ncol, job, a, ja, ia, b, jb, ib, c, jc, ic, nzmax, iw, ierr)

Performs the matrix by matrix product C = A B

ON ENTRY :

nrow = integer. The row dimension of A

ncol = integer. The column dimension of A

job = integer. Job indicator. When job = 0, only the structure (i.e. the arrays jc, ic) is computed and the real values are ignored.

a,ja,ia = Matrix A in compressed sparse row format.

b,jb,ib = Matrix B in compressed sparse row format.

nzmax = integer. The length of the arrays c and jc. amub will stop if the result matrix C has a number of elements that exceeds exceeds nzmax. See ierr.

ON RETURN :

c,jc,ic = resulting matrix C in compressed sparse row sparse format.

ierr = integer. serving as error message.

ierr = 0 means normal return, ierr .gt. 0 means that amub stopped while computing the i-th row of C with i=ierr, because the number of elements in C exceeds nzmax.

WORK ARRAYS :

iw = integer work array of length equal to the number of columns in A.

NOTES :

The column dimension of B is not needed. / }

dump (i1,i2,values,a,ja,ia,iout) { /** Outputs rows i1 through i2 of a sparse matrix stored in CSR format (or columns i1 through i2 of a matrix stored in CSC format) in a file, one (column) row at a time in a nice readable format. This is a simple routine which is useful for debugging.

ON ENTRY :

i1 = first row (column) to print out

i2 = last row (column) to print out

values= logical. indicates whether or not to print real values. if value = .false. only the pattern will be output.

a,ja,ia = matrix in CSR format (or CSC format)

iout = logical unit number for output.

ON RETURN :

The output file iout will have written in it the rows or columns of the matrix in one of two possible formats (depending on the max number of elements per row. The values are output with only two digits of accuracy (D9.2).)

4.45.1.4 amudia (nrow, job, a, ja, ia, diag, b, jb, ib)

Performs the matrix by matrix product B = A * Diag (in place)

ON ENTRY :

nrow = integer. The row dimension of A

job = integer. job indicator. Job=0 means get array b only job = 1 means get b, and the integer arrays ib, jb.

a,ja,ia = Matrix A in compressed sparse row format.

diag = diagonal matrix stored as a vector dig(1:n)

ON RETURN :

b,jb,ib = resulting matrix B in compressed sparse row sparse format.

NOTES :

1) The column dimension of A is not needed.

2) algorithm in place (B can take the place of A).

4.45.1.5 aplb (nrow, ncol, job, a, ja, ia, b, jb, ib, c, jc, ic, nzmax, iw, ierr)

Performs the matrix sum C = A+B.

ON ENTRY :

nrow = integer. The row dimension of A and B

ncol = integer. The column dimension of A and B.

job = integer. Job indicator. When job = 0, only the structure (i.e. the arrays jc, ic) is computed and the real values are ignored.

a,ja,ia = Matrix A in compressed sparse row format.

b,jb,ib = Matrix B in compressed sparse row format.

nzmax = integer. The length of the arrays c and jc. amub will stop if the result matrix C has a number of elements that exceeds exceeds nzmax. See ierr.

ON RETURN :

c,jc,ic = resulting matrix C in compressed sparse row sparse format.

ierr = integer. serving as error message. ierr = 0 means normal return, ierr .gt. 0 means that amub stopped while computing the i-th row of C with i=ierr, because the number of elements in C exceeds nzmax.

WORK ARRAYS :

iw = integer work array of length equal to the number of columns in A.

4.45.1.6 atmux (n, x, y, a, ja, ia)

transp(A) times a vector

Multiplies the transpose of a matrix by a vector when the original matrix is stored in compressed sparse row storage. Can also be viewed as the product of a matrix by a vector when the original matrix is stored in the compressed sparse column format.

ON ENTRY :

n = row dimension of A

x = real array of length equal to the column dimension of the A matrix. a, ja,ia = input matrix in compressed sparse row format.

ON RETURN :

y = real array of length n, containing the product y=transp(A)∗x

4.45.1.7 atmuxr (m, n, x, y, a, ja, ia)

transp(A) times a vector, A can be rectangular

See also atmux. The essential difference is how the solution vector is initially zeroed. If using this to multiply rectangular CSC matrices by a vector, m number of rows, n is number of columns.

ON ENTRY :

m = column dimension of A

n = row dimension of A

x = real array of length equal to the column dimension of the A matrix. a, ja,ia = input matrix in compressed sparse row format.

ON RETURN :

y = real array of length n, containing the product y=transp(A)∗x

4.45.1.8 BFS (n, ja, ia, nfirst, iperm, mask, maskval, riord, levels, nlev)

Finds the level-structure (breadth-first-search or CMK) ordering for a given sparse matrix. Uses add_lvst. Allows an set of nodes to be the initial level (instead of just one node).

ON ENTRY :

n = number of nodes in the graph

ja, ia = pattern of matrix in CSR format (the ja,ia arrays of csr data structure)

nfirst = number of nodes in the first level that is input in riord

iperm = integer array indicating in which order to traverse the graph in order to generate all connected components. if iperm(1) .eq. 0 on entry then BFS will traverse the nodes in the order 1,2,...,n.

riord = (also an ouput argument). On entry riord contains the labels of the nfirst nodes that constitute the first level.

mask = array used to indicate whether or not a node should be condidered in the graph. see maskval. mask is also used as a marker of visited nodes.

maskval= consider node i only when: mask(i) .eq. maskval maskval must be .gt. 0. thus, to consider all nodes, take mask(1:n) = 1. maskval=1 (for example)

ON RETURN :

mask = on return mask is restored to its initial state.

riord = reverse permutation array. Contains the labels of the nodes constituting all the levels found, from the first level to the last.

levels = pointer array for the level structure. If lev is a level number, and k1=levels(lev),k2=levels(lev+1)-1, then all the nodes of level number lev are: riord(k1),riord(k1+1),...,riord(k2)

nlev = number of levels found

4.45.1.9 cnrms (nrow, nrm, a, ja, ia, diag)

Gets the norms of each column of A. (choice of three norms)

ON ENTRY :

nrow = integer. The row dimension of A

nrm = integer. norm indicator. nrm = 1, means 1-norm, nrm =2 means the 2-nrm, nrm = 0 means max norm

a,ja,ia = Matrix A in compressed sparse row format.

ON RETURN :

diag = real vector of length nrow containing the norms

4.45.1.10 coscal (nrow, job, nrm, a, ja, ia, diag, b, jb, ib, ierr)

Scales the columns of A such that their norms are one on return result matrix written on b, or overwritten on A. 3 choices of norms: 1-norm, 2-norm, max-norm. in place.

ON ENTRY :

nrow = integer. The row dimension of A

job = integer. job indicator. Job=0 means get array b only job = 1 means get b, and the integer arrays ib, jb.

nrm = integer. norm indicator. nrm = 1, means 1-norm, nrm =2 means the 2-nrm, nrm = 0 means max norm

a,ja,ia = Matrix A in compressed sparse row format.

ON RETURN :

diag = diagonal matrix stored as a vector containing the matrix by which the columns have been scaled, i.e., on return we have B = A ∗ Diag

b,jb, ib = resulting matrix B in compressed sparse row sparse format.

ierr = error message. ierr=0 : Normal return

ierr=i > 0 : Column number i is a zero row.

NOTES :

1) The column dimension of A is not needed.

2) algorithm in place (B can take the place of A).

4.45.1.11 cperm (nrow, a, ja, ia, ao, jao, iao, perm, job)

This permutes the columns of a matrix a, ja, ia. the result is written in the output matrix ao, jao, iao. cperm computes B = A P, where P is a permutation matrix that maps column j into column perm(j), i.e., on return a(i,j) becomes a(i,perm(j)) in new matrix

ON ENTRY :

nrow = row dimension of the matrix

a, ja, ia = input matrix in csr format.

perm = integer array of length ncol (number of columns of A containing the permutation array the columns: a(i,j) in the original matrix becomes a(i,perm(j)) in the output matrix.

job = integer indicating the work to be done:

job = 1 permute a, ja, ia into ao, jao, iao (including the copying of real values ao and the array iao).

job .ne. 1 : ignore real values ao and ignore iao.

ON RETURN :

ao, jao, iao = input matrix in a, ja, ia format (array ao not needed)

NOTES :

1. if job=1 then ao, iao are not used.

2. This routine is in place: ja, jao can be the same.

3. If the matrix is initially sorted (by increasing column number) then ao,jao,iao may not be on return.

4.45.1.12 csort (n, a, ja, ia, iwork, values)

This routine sorts the elements of a matrix (stored in Compressed Sparse Row Format) in increasing order of their column indices within each row. It uses a form of bucket sort with a cost of O(nnz) where nnz = number of nonzero elements. requires an integer work array of length 2∗nnz.

on entry:

n = the row dimension of the matrix a = the matrix A in compressed sparse row format. ja = the array of column indices of the elements in array a. ia = the array of pointers to the rows. iwork =

integer work array of length max (n+1, 2∗nnz) where nnz = (ia(n+1)-ia(1))) . values= logical indicating whether or not the real values a(∗) must also be permuted. if (.not. values) then the array a is not touched by csort and can be a dummy array.

on return:

the matrix stored in the structure a, ja, ia is permuted in such a way that the column indices are in increasing order within each row. iwork(1:nnz) contains the permutation used to rearrange the elements.

4.45.1.13 csrcsc (n, job, ipos, a, ja, ia, ao, jao, iao)

Compressed Sparse Row to Compressed Sparse Column

(transposition operation) Not in place.

– not in place – this subroutine transposes a matrix stored in a, ja, ia format.

ON ENTRY :

n = dimension of A.

job = integer to indicate whether to fill the values (job.eq.1) of the matrix ao or only the pattern., i.e.,ia, and ja (job .ne.1)

ipos = starting position in ao, jao of the transposed matrix. the iao array takes this into account (thus iao(1) is set to ipos.) Note: this may be useful if one needs to append the data structure of the transpose to that of A. In this case use for example call csrcsc (n,1,n+2,a,ja,ia,a,ja,ia(n+2)) for any other normal usage, enter ipos=1.

a = real array of length nnz (nnz=number of nonzero elements in input matrix) containing the nonzero elements.

ja = integer array of length nnz containing the column positions of the corresponding elements in a.

ia = integer of size n+1. ia(k) contains the position in a, ja of the beginning of the k-th row.

ON RETURN :

output arguments:

ao = real array of size nzz containing the "a" part of the transpose

jao = integer array of size nnz containing the column indices.

iao = integer array of size n+1 containing the "ia" index array of the transpose.

4.45.1.14 csrcsc2 (n, n2, job, ipos, a, ja, ia, ao, jao, iao)

Compressed Sparse Row to Compressed Sparse Column

(transposition operation) Not in place.

Rectangular version. n is number of rows of CSR matrix, n2 (input) is number of columns of CSC matrix.

– not in place – this subroutine transposes a matrix stored in a, ja, ia format.

ON ENTRY :

n = number of rows of CSR matrix.

n2 = number of columns of CSC matrix.

job = integer to indicate whether to fill the values (job.eq.1) of the matrix ao or only the pattern., i.e.,ia, and ja (job .ne.1)

ipos = starting position in ao, jao of the transposed matrix. the iao array takes this into account (thus iao(1) is set to ipos.) Note: this may be useful if one needs to append the data structure of the transpose to that of A. In this case use for example call csrcsc2 (n,n,1,n+2,a,ja,ia,a,ja,ia(n+2)) for any other normal usage, enter ipos=1.

a = real array of length nnz (nnz=number of nonzero elements in input matrix) containing the nonzero elements.

ja = integer array of length nnz containing the column positions of the corresponding elements in a.

ia = integer of size n+1. ia(k) contains the position in a, ja of the beginning of the k-th row.

ON RETURN :

output arguments:

ao = real array of size nzz containing the "a" part of the transpose

jao = integer array of size nnz containing the column indices.

iao = integer array of size n+1 containing the "ia" index array of the transpose.

4.45.1.15 dblstr (n, ja, ia, ip1, ip2, nfirst, riord, ndom, map, mapptr, mask, levels, iwk)

This routine does a two-way partitioning of a graph using level sets recursively. First a coarse set is found by a simple cuthill-mc Kee type algorithm. Them each of the large domains is further partitioned into subsets using the same technique. The ip1 and ip2 parameters indicate the desired number number of partitions 'in each direction'. So the total number of partitions on return ought to be equal (or close) to ip1∗ip2

ON ENTRY :

n = row dimension of matrix == number of vertices in graph

ja, ia = pattern of matrix in CSR format (the ja,ia arrays of csr data structure)

ip1 = integer indicating the number of large partitions ('number of paritions in first direction')

ip2 = integer indicating the number of smaller partitions, per large partition, ('number of partitions in second direction')

nfirst = number of nodes in the first level that is input in riord

riord = (also an ouput argument). on entry riord contains the labels of the nfirst nodes that constitute the first level.

ON RETURN :

ndom = total number of partitions found

map = list of nodes listed partition by partition from partition 1 to paritition ndom.

mapptr = pointer array for map. All nodes from position k1=mapptr(idom),to position k2=mapptr(idom+1)-1 in map belong to partition idom.

WORK ARRAYS :

mask = array of length n, used to hold the partition number of each node for the first (large) partitioning. mask is also used as a marker of visited nodes.

levels = integer array of length .le. n used to hold the pointer arrays for the various level structures obtained from BFS.

4.45.1.16 diamua (nrow, job, a, ja, ia, diag, b, jb, ib)

Performs the matrix by matrix product B = Diag ∗ A (in place)

ON ENTRY :

nrow = integer. The row dimension of A

job = integer. job indicator. Job=0 means get array b only job = 1 means get b, and the integer arrays ib, jb.

a,ja,ia = Matrix A in compressed sparse row format.

diag = diagonal matrix stored as a vector dig(1:n)

ON RETURN :

b,jb,ib = resulting matrix B in compressed sparse row sparse format.

NOTES :

1) The column dimension of A is not needed.

2) algorithm in place (B can take the place of A). in this case use job=0.

4.45.1.17 dperm (nrow, a, ja, ia, ao, jao, iao, perm, qperm, job)

This routine permutes the rows and columns of a matrix stored in CSR format. i.e., it computes P A Q, where P, Q are permutation matrices. P maps row i into row perm(i) and Q maps column j into column qperm(j): a(i,j) becomes a(perm(i),qperm(j)) in new matrix In the particular case where Q is the transpose of P (symmetric permutation of A) then qperm is not needed. note that qperm should be of length ncol (number of columns) but this is not checked.

ON ENTRY :

n = dimension of the matrix

a, ja,ia = input matrix in a, ja, ia format

perm = integer array of length n containing the permutation arrays for the rows: perm(i) is the destination of row i in the permuted matrix – also the destination of column i in case permutation is symmetric (job .le. 2)

qperm = same thing for the columns. This should be provided only if job=3 or job=4, i.e., only in the case of a nonsymmetric permutation of rows and columns. Otherwise qperm is a dummy

job = integer indicating the work to be done:

job = 1,2 permutation is symmetric Ao :== P ∗ A ∗ transp(P) job = 1 permute a, ja, ia into ao, jao, iao job = 2 permute matrix ignoring real values.

job = 3,4 permutation is non-symmetric Ao :== P ∗ A ∗ Q job = 3 permute a, ja, ia into ao, jao, iao job = 4 permute matrix ignoring real values.

ON RETURN :

ao, jao, iao = input matrix in a, ja, ia format

in case job .eq. 2 or job .eq. 4, a and ao are never referred to and can be dummy arguments.

NOTES :

1) algorithm is in place

2) column indices may not be sorted on return even though they may be on entry.

4.45.1.18 dperm1 (i1, i2, a, ja, ia, b, jb, ib, perm, ipos, job)

General submatrix permutation/ extraction routine.

Extracts rows perm(i1), perm(i1+1), ..., perm(i2) (in this order) from a matrix (doing nothing in the column indices.) The resulting submatrix is constructed in b, jb, ib. A pointer ipos to the beginning of arrays b,jb,is also allowed (i.e., nonzero elements are accumulated starting in position ipos of b, jb).

ON ENTRY :

n = dimension of the matrix

a,ja,ia = input matrix in CSR format

perm = integer array of length n containing the indices of the rows to be extracted.

job = job indicator. if (job .ne.1) values are not copied (i.e., only pattern is copied).

ON RETURN :

b,jb,ib = matrix in csr format. b(ipos:ipos+nnz-1),jb(ipos:ipos+nnz-1) contain the value and column indices respectively of the nnz nonzero elements of the permuted matrix. thus ib(1)=ipos.

NOTES :

Algorithm is NOT in place

4.45.1.19 dperm2 (i1, i2, a, ja, ia, b, jb, ib, perm, rperm, istart, ipos, job)

General submatrix permutation/ extraction routine.

Extracts rows rperm(i1), rperm(i1+1), ..., rperm(i2) and does an associated column permutation (using array perm). The resulting submatrix is constructed in b, jb, ib. For added flexibility, the extracted are put in sequence starting from row 'istart' of B. In addition a pointer ipos to the beginning of arrays b,jb,is also allowed (i.e., nonzero elements are accumulated starting in position ipos of b, jb). extremely flexible. exple to permute msr to msr (excluding diagonals) call dperm2 (1,n,a,ja,ja,b,jb,jb,perm,rperm,1,n+2)

ON ENTRY :

n = dimension of the matrix

a,ja,ia = input matrix in CSR format

perm = integer array of length n containing the permutation arrays for the rows: perm(i) is the destination of row i in the permuted matrix – also the destination of column i.

rperm = reverse permutation. defined by rperm(iperm(i))=i for all i

job = job indicator. if (job .ne.1) values are not copied (i.e., only pattern is copied).

ON RETURN :

b,ja,ib = matrix in csr format. positions 1,2,...,istart-1 of ib are not touched. b(ipos:ipos+nnz-1),jb(ipos:ipos+nnz-1) contain the value and column indices respectively of the nnz nonzero elements of the permuted matrix. thus ib(istart)=ipos.

NOTES :

1) algorithm is NOT in place

2) column indices may not be sorted on return even though they may be on entry.

4.45.1.20 dse (n, ja, ia, ndom, riord, dom, idom, mask, jwk, link)

Uses centers produced from rdis to get a new partitioning – see calling sequence in rdis..

ON ENTRY :

n, ja, ia = graph

ndom = number of desired subdomains

ON RETURN :

dom, idom = pointer array structure for the ndom domains.

size of array jwk = 2∗ndom

size array link

dom = array of size at least n

mask = array of size n.

link = same size as riord

riord = n list of nodes listed in sequence for all subdomains

4.45.1.21 dse2way (n, ja, ia, ip1, ip2, nfirst, riord, ndom, dom, idom, mask, jwk, link)

Uses centers obtained from dblstr partition to get new partition

ON ENTRY :

n, ja, ia = matrix

nfirst = number of first points

riord = riord(1:nfirst) initial points

ON RETURN :

ndom = number of domains

dom, idom = pointer array structure for domains.

mask , jwk, link = work arrays,

4.45.1.22 dvperm (n, x, perm)

This performs an in-place permutation of a real vector x according to the permutation array perm(∗), i.e., on return, the vector x satisfies,

x(perm(j)) :== x(j), j=1,2,.., n

ON ENTRY :

n = length of vector x.

perm = integer array of length n containing the permutation array.

x = input vector

ON RETURN :

x = vector x permuted according to x(perm(∗)) := x(∗)

4.45.1.23 find_ctr (n, nsiz, ja, ia, init, mask, maskval, riord, levels, center, iwk)

Finds a center point of a subgraph –

n, ja, ia = graph

nsiz = size of current domain.

init = initial node in search

mask

maskval

4.45.1.24 get_domns2 (ndom, nodes, link, levst, riord, iptr)

Constructs the subdomains from its linked list data structure

ON ENTRY :

ndom = number of subdomains

nodes = sequence of nodes are produced by mapper4.

link = link list array as produced by mapper4.

ON RETURN :

riord = contains the nodes in each subdomain in succession.

iptr = pointer in riord for beginnning of each subdomain.

Thus subdomain number i consists of nodes riord(k1),riord(k1)+1,...,riord(k2) where k1 = iptr(i), k2= iptr(i+1)-1

4.45.1.25 ivperm (n, ix, perm)

This performs an in-place permutation of an integer vector ix according to the permutation array perm(∗), i.e., on return, the vector x satisfies,

ix(perm(j)) :== ix(j), j=1,2,.., n

ON ENTRY :

n = length of vector x.

perm = integer array of length n containing the permutation array.

ix = input vector

ON RETURN :

ix = vector x permuted according to ix(perm(∗)) := ix(∗)

4.45.1.26 mapper4 (n, ja, ia, ndom, nodes, levst, marker, link)

Finds domains given ndom centers – by doing a level set expansion

ON ENTRY :

n = dimension of matrix

ja, ia = adajacency list of matrix (CSR format without values) –

ndom = number of subdomains (nr output by coarsen)

nodes = array of size at least n. On input the first ndom entries of nodes should contain the labels of the centers of the ndom domains from which to do the expansions.

ON RETURN :

link = linked list array for the ndom domains.

nodes = contains the list of nodes of the domain corresponding to link. (nodes(i) and link(i) are related to the same node).

levst = levst(j) points to beginning of subdomain j in link.

WORK ARRAYS :

levst : work array of length 2*ndom – contains the beginning and end of current level in link. beginning of last level in link for each processor. also ends in levst(ndom+i)

marker : work array of length n.

NOTES ON IMPLEMENTATION :

for j .le. ndom link(j) is <0 and indicates the end of the linked list. The most recent element added to the linked list is added at the end of the list (traversal=backward) For j .le. ndom, the value of -link(j) is the size of subdomain j.

4.45.1.27 mindom (n, ndom, levst, link)

Returns the domain with smallest size

4.45.1.28 multic (n, ja, ia, ncol, kolrs, il, iord, maxcol, ierr)

Multicoloring ordering – greedy algorithm – determines the coloring permutation and sets up corresponding data structures for it.

ON ENTRY :

n = row and column dimention of matrix

ja = column indices of nonzero elements of matrix, stored rowwise.

ia = pointer to beginning of each row in ja.

maxcol= maximum number of colors allowed – the size of il is maxcol+1 at least. Note: the number of colors does not exceed the maximum degree of each node +1.

ON RETURN :

ncol = number of colours found

kolrs = integer array containing the color number assigned to each node

il = integer array containing the pointers to the beginning of each color set. In the permuted matrix the rows /columns il(kol) to il(kol+1)-1 have the same color.

iord = permutation array corresponding to the multicolor ordering. row number i will become row nbumber iord(i) in permuted matrix. (iord = destination permutation array).

ierr = integer. Error message. normal return ierr = 0

4.45.1.29 perphn (n, ja, ia, init, mask, maskval, nlev, riord, levels)

Finds a peripheral node and does a BFS search from it.

see routine dblstr for description of parameters

ON ENTRY :

ja, ia = list pointer array for the adjacency graph

mask = array used for masking nodes – see maskval

maskval = value to be checked against for determing whether or not a node is masked. If mask(k) .ne. maskval then node k is not considered.

init = init node in the pseudo-peripheral node algorithm.

ON RETURN :

init = actual pseudo-peripherial node found.

nlev = number of levels in the final BFS traversal.

riord =

levels =

4.45.1.30 rdis (n, ja, ia, ndom, map, mapptr, mask, levels, size, iptr)

Recursive dissection algorithm for partitioning. initial graph is cut in two - then each time, the largest set is cut in two until we reach desired number of domains.

ON ENTRY :

n, ja, ia = graph

ndom = desired number of subgraphs

ON RETURN :

map, mapptr = pointer array data structure for domains. if k1 = mapptr(i), k2=mapptr(i+1)-1 then map(k1:k2) = points in domain number i work arrays:

mask(1:n) integer

levels(1:n) integer

size(1:ndom) integer

iptr(1:ndom) integer

4.45.1.31 readmt (nmax, nzmax, job, iounit, a, ja, ia, rhs, nrhs, guesol, nrow, ncol, nnz, title, key, type, ierr)

This reads a boeing/harwell matrix. handles right hand sides in full format only (no sparse right hand sides). Also the matrix must be in assembled forms.

ON ENTRY :

nmax = max column dimension allowed for matrix. The array ia should be of length at least ncol+1 (see below) if job.gt.0 nzmax = max number of nonzeros elements allowed. the arrays a, and ja should be of length equal to nnz (see below) if these arrays are to be read (see job).

job = integer to indicate what is to be read. (note: job is an input and output parameter, it can

be modified on return)

job = 0 read the values of ncol, nrow, nnz, title, key, type and return. matrix is not read and arrays a, ja, ia, rhs are not touched.

job = 1 read srtucture only, i.e., the arrays ja and ia.

job = 2 read matrix including values, i.e., a, ja, ia

job = 3 read matrix and right hand sides: a,ja,ia,rhs. rhs may contain initial guesses and exact solutions appended to the actual right hand sides. this will be indicated by the output parameter guesol [see below].

nrhs = integer. nrhs is an input as well as ouput parameter. at input nrhs contains the total length of the array rhs. See also ierr and nrhs in output parameters.

iounit = logical unit number where to read the matrix from.

ON RETURN :

job = on return job may be modified to the highest job it could do: if job=2 on entry but no matrix values are available it is reset to job=1 on return. Similarly of job=3 but no rhs is provided then it is rest to job=2 or job=1 depending on whether or not matrix values are provided. Note that no error message is triggered (i.e. ierr = 0 on return in these cases. It is therefore important to compare the values of job on entry and return).

a = the a matrix in the a, ia, ja (column) storage format

ja = row number of element a(i,j) in array a.

ia = pointer array. ia(i) points to the beginning of column i.

rhs = real array of size nrow + 1 if available (see job)

nrhs = integer containing the number of right-hand sides found each right hand side may be accompanied with an intial guess and also the exact solution.

guesol = a 2-character string indicating whether an initial guess (1-st character) and / or the exact solution (2-nd character) is provided with the right hand side. if the first character of guesol is 'G' it means that an an intial guess is provided for each right-hand side. These are appended to the right hand-sides in the array rhs. if the second character of guesol is 'X' it means that an exact solution is provided for each right-hand side. These are appended to the right hand-sides and the initial guesses (if any) in the array rhs.

nrow = number of rows in matrix

ncol = number of columns in matrix

nnz = number of nonzero elements in A. This info is returned even if there is not enough space in a, ja, ia, in order to determine the minimum storage needed.

title = character∗100 = title of matrix test (character a∗72).

key = character∗8 = key of matrix

type = charatcer∗3 = type of matrix. for meaning of title, key and type refer to documentation Harwell/Boeing matrices.

ierr = integer used for error messages

ierr = 0 means that the matrix has been read normally.

ierr = 1 means that the array matrix could not be read because ncol+1 .gt. nmax

ierr = 2 means that the array matrix could not be read because nnz .gt. nzmax

ierr = 3 means that the array matrix could not be read because both (ncol+1 .gt. nmax) and

(nnz .gt. nzmax)

ierr = 4 means that the right hand side (s) initial guesse (s) and exact solution (s) could not be read because they are stored in sparse format (not handled by this routine ...)

ierr = 5 means that the right-hand-sides, initial guesses and exact solutions could not be read because the length of rhs as specified by the input value of nrhs is not sufficient to store them. The rest of the matrix may have been read normally.

NOTES :

1) The file inout must be open (and possibly rewound if necessary) prior to calling readmt.

2) Refer to the documentation on the Harwell-Boeing formats for details on the format assumed by readmt. We summarize the format here for convenience.

a) all lines in inout are assumed to be 80 character long.

b) the file consists of a header followed by the block of the column start pointers followed by the block of the row indices, followed by the block of the real values and finally the numerical values of the right-hand-side if a right hand side is supplied.

c) the file starts by a header which contains four lines if no right hand side is supplied and five lines otherwise.

first line contains the title (72 characters long) followed by the 8-character identifier (name of the matrix, called key) [A72,A8]

second line contains the number of lines for each of the following data blocks (4 of them) and the total number of lines excluding the header. [5i4]

the third line contains a three character string identifying the type of matrices as they are referenced in the Harwell Boeing documentation [e.g., rua, rsa,..] and the number of rows, columns, nonzero entries. [A3,11X,4I14]

The fourth line contains the variable fortran format for the following data blocks. [2A16,2A20]

The fifth line is only present if right-hand-sides are supplied. It consists of three one character-strings containing the storage format for the right-hand-sides (F = full, M =sparse=same as matrix), an initial guess indicator ('G' for yes), an exact solution indicator ('X' for yes), followed by the number of right-hand-sides and then the number of row indices. [A3,11X,2I14]

d) The three following blocks follow the header as described above.

e) In case the right hand-side are in sparse formats then the fourth block uses the same storage format as for the matrix to describe the NRHS right hand sides provided, with a column being replaced by a right hand side.

4.45.1.32 readmt_c (nmax, nzmax, job, fname, a, ja, ia, rhs, nrhs, guesol, nrow, ncol, nnz, title, key, type, ierr)

This reads a boeing/harwell matrix, given the corresponding file. handles right hand sides in full format only (no sparse right hand sides). Also the matrix must be in assembled forms. It differs from readmt, in that the name of the file needs to be passed, and then the file is opened and closed within this routine.

ON ENTRY :

nmax = max column dimension allowed for matrix. The array ia should be of length at least ncol+1 (see below) if job.gt.0

nzmax = max number of nonzeros elements allowed. the arrays a, and ja should be of length equal

to nnz (see below) if these arrays are to be read (see job).

job = integer to indicate what is to be read. (note: job is an input and output parameter, it can be modified on return)

job = 0 read the values of ncol, nrow, nnz, title, key, type and return. matrix is not read and arrays a, ja, ia, rhs are not touched.

job = 1 read srtucture only, i.e., the arrays ja and ia.

job = 2 read matrix including values, i.e., a, ja, ia

job = 3 read matrix and right hand sides: a,ja,ia,rhs. rhs may contain initial guesses and exact solutions appended to the actual right hand sides. this will be indicated by the output parameter guesol [see below].

fname = name of the file where to read the matrix from.

nrhs = integer. nrhs is an input as well as ouput parameter. at input nrhs contains the total length of the array rhs. See also ierr and nrhs in output parameters.

ON RETURN :

job = on return job may be modified to the highest job it could do: if job=2 on entry but no matrix values are available it is reset to job=1 on return. Similarly of job=3 but no rhs is provided then it is rest to job=2 or job=1 depending on whether or not matrix values are provided. Note that no error message is triggered (i.e. ierr = 0 on return in these cases. It is therefore important to compare the values of job on entry and return).

a = the a matrix in the a, ia, ja (column) storage format

ja = column number of element a(i,j) in array a.

ia = pointer array. ia(i) points to the beginning of column i.

rhs = real array of size nrow + 1 if available (see job)

nrhs = integer containing the number of right-hand sides found each right hand side may be accompanied with an intial guess and also the exact solution.

guesol = a 2-character string indicating whether an initial guess (1-st character) and / or the exact solution (2-nd character) is provided with the right hand side. if the first character of guesol is 'G' it means that an an intial guess is provided for each right-hand side. These are appended to the right hand-sides in the array rhs. if the second character of guesol is 'X' it means that an exact solution is provided for each right-hand side. These are appended to the right hand-sides and the initial guesses (if any) in the array rhs.

nrow = number of rows in matrix

ncol = number of columns in matrix

nnz = number of nonzero elements in A. This info is returned even if there is not enough space in a, ja, ia, in order to determine the minimum storage needed.

title = character$*$72 = title of matrix test (character a$*$72).

key = character$*$8 = key of matrix

type = charatcer$*$3 = type of matrix. for meaning of title, key and type refer to documentation Harwell/Boeing matrices.

ierr = integer used for error messages

ierr = 0 means that the matrix has been read normally.

ierr = 1 means that the array matrix could not be read because ncol+1 .gt. nmax

ierr = 2 means that the array matrix could not be read because nnz .gt. nzmax

ierr = 3 means that the array matrix could not be read because both (ncol+1 .gt. nmax) and (nnz .gt. nzmax)

ierr = 4 means that the right hand side (s) initial guesse (s) and exact solution (s) could not be read because they are stored in sparse format (not handled by this routine ...)

ierr = 5 means that the right-hand-sides, initial guesses and exact solutions could not be read because the length of rhs as specified by the input value of nrhs is not sufficient to store them. The rest of the matrix may have been read normally.

NOTES :

1) This routine can be interfaced with the C language, since only the name of the file needs to be passed and no iounti number.

2) Refer to the documentation on the Harwell-Boeing formats for details on the format assumed by readmt. We summarize the format here for convenience.

a) all lines in inout are assumed to be 80 character long.

b) the file consists of a header followed by the block of the column start pointers followed by the block of the row indices, followed by the block of the real values and finally the numerical values of the right-hand-side if a right hand side is supplied.

c) the file starts by a header which contains four lines if no right hand side is supplied and five lines otherwise.

first line contains the title (72 characters long) followed by the 8-character identifier (name of the matrix, called key) [A72,A8]

second line contains the number of lines for each of the following data blocks (4 of them) and the total number of lines excluding the header. [5i4]

the third line contains a three character string identifying the type of matrices as they are referenced in the Harwell Boeing documentation [e.g., rua, rsa,..] and the number of rows, columns, nonzero entries. [A3,11X,4I14]

The fourth line contains the variable fortran format for the following data blocks. [2A16,2A20]

The fifth line is only present if right-hand-sides are supplied. It consists of three one character-strings containing the storage format for the right-hand-sides ('F'= full,'M'=sparse=same as matrix), an initial guess indicator ('G' for yes), an exact solution indicator ('X' for yes), followed by the number of right-hand-sides and then the number of row indices. [A3,11X,2I14]

d) The three following blocks follow the header as described above.

e) In case the right hand-side are in sparse formats then the fourth block uses the same storage format as for the matrix to describe the NRHS right hand sides provided, with a column being replaced by a right hand side.

4.45.1.33 readmtc (nmax, nzmax, job, fname, a, ja, ia, rhs, nrhs, guesol, nrow, ncol, nnz, title, key, type, ierr)

This reads a boeing/harwell matrix, given the corresponding file. handles right hand sides in full format only (no sparse right hand sides). Also the matrix must be in assembled forms. It differs from readmt, in that the name of the file needs to be passed, and then the file is opened and closed within this routine.

ON ENTRY :

nmax = max column dimension allowed for matrix. The array ia should be of length at least ncol+1 (see below) if job.gt.0

nzmax = max number of nonzeros elements allowed. the arrays a, and ja should be of length equal to nnz (see below) if these arrays are to be read (see job).

job = integer to indicate what is to be read. (note: job is an input and output parameter, it can be modified on return)

job = 0 read the values of ncol, nrow, nnz, title, key, type and return. matrix is not read and arrays a, ja, ia, rhs are not touched.

job = 1 read srtucture only, i.e., the arrays ja and ia.

job = 2 read matrix including values, i.e., a, ja, ia

job = 3 read matrix and right hand sides: a,ja,ia,rhs. rhs may contain initial guesses and exact solutions appended to the actual right hand sides. this will be indicated by the output parameter guesol [see below].

fname = name of the file where to read the matrix from.

nrhs = integer. nrhs is an input as well as ouput parameter. at input nrhs contains the total length of the array rhs. See also ierr and nrhs in output parameters.

ON RETURN :

job = on return job may be modified to the highest job it could do: if job=2 on entry but no matrix values are available it is reset to job=1 on return. Similarly of job=3 but no rhs is provided then it is rest to job=2 or job=1 depending on whether or not matrix values are provided. Note that no error message is triggered (i.e. ierr = 0 on return in these cases. It is therefore important to compare the values of job on entry and return).

a = the a matrix in the a, ia, ja (column) storage format

ja = column number of element a(i,j) in array a.

ia = pointer array. ia(i) points to the beginning of column i.

rhs = real array of size nrow + 1 if available (see job)

nrhs = integer containing the number of right-hand sides found each right hand side may be accompanied with an intial guess and also the exact solution.

guesol = a 2-character string indicating whether an initial guess (1-st character) and / or the exact solution (2-nd character) is provided with the right hand side. if the first character of guesol is 'G' it means that an an intial guess is provided for each right-hand side. These are appended to the right hand-sides in the array rhs. if the second character of guesol is 'X' it means that an exact solution is provided for each right-hand side. These are appended to the right hand-sides and the initial guesses (if any) in the array rhs.

nrow = number of rows in matrix

ncol = number of columns in matrix

nnz = number of nonzero elements in A. This info is returned even if there is not enough space in a, ja, ia, in order to determine the minimum storage needed.

title = character$*$72 = title of matrix test (character a$*$72).

key = character$*$8 = key of matrix

type = charatcer$*$3 = type of matrix. for meaning of title, key and type refer to documentation Harwell/Boeing matrices.

ierr = integer used for error messages

ierr = 0 means that the matrix has been read normally.

ierr = 1 means that the array matrix could not be read because ncol+1 .gt. nmax

ierr = 2 means that the array matrix could not be read because nnz .gt. nzmax

ierr = 3 means that the array matrix could not be read because both (ncol+1 .gt. nmax) and (nnz .gt. nzmax)

ierr = 4 means that the right hand side (s) initial guesse (s) and exact solution (s) could not be read because they are stored in sparse format (not handled by this routine ...)

ierr = 5 means that the right-hand-sides, initial guesses and exact solutions could not be read because the length of rhs as specified by the input value of nrhs is not sufficient to store them. The rest of the matrix may have been read normally.

NOTES :

1) This routine can be interfaced with the C language, since only the name of the file needs to be passed and no iounti number.

2) Refer to the documentation on the Harwell-Boeing formats for details on the format assumed by readmt. We summarize the format here for convenience.

a) all lines in inout are assumed to be 80 character long.

b) the file consists of a header followed by the block of the column start pointers followed by the block of the row indices, followed by the block of the real values and finally the numerical values of the right-hand-side if a right hand side is supplied.

c) the file starts by a header which contains four lines if no right hand side is supplied and five lines otherwise.

first line contains the title (72 characters long) followed by the 8-character identifier (name of the matrix, called key) [A72,A8]

second line contains the number of lines for each of the following data blocks (4 of them) and the total number of lines excluding the header. [5i4]

the third line contains a three character string identifying the type of matrices as they are referenced in the Harwell Boeing documentation [e.g., rua, rsa,..] and the number of rows, columns, nonzero entries. [A3,11X,4I14]

The fourth line contains the variable fortran format for the following data blocks. [2A16,2A20]

The fifth line is only present if right-hand-sides are supplied. It consists of three one character-strings containing the storage format for the right-hand-sides ('F'= full,'M'=sparse=same as matrix), an initial guess indicator ('G' for yes), an exact solution indicator ('X' for yes), followed by the number of right-hand-sides and then the number of row indices. [A3,11X,2I14]

d) The three following blocks follow the header as described above.

e) In case the right hand-side are in sparse formats then the fourth block uses the same storage format as for the matrix to describe the NRHS right hand sides provided, with a column being replaced by a right hand side.

4.45.1.34 rnrms (nrow, nrm, a, ja, ia, diag)

Gets the norms of each row of A. (choice of three norms)

ON ENTRY :

nrow = integer. The row dimension of A

nrm = integer. norm indicator. nrm = 1, means 1-norm, nrm =2 means the 2-nrm, nrm = 0 means max norm

a,ja,ia = Matrix A in compressed sparse row format.

ON RETURN :

diag = real vector of length nrow containing the norms

4.45.1.35 roscal (nrow, job, nrm, a, ja, ia, diag, b, jb, ib, ierr)

Scales the rows of A such that their norms are one on return 3 choices of norms: 1-norm, 2-norm, max-norm.

ON ENTRY :

nrow = integer. The row dimension of A

job = integer. job indicator. Job=0 means get array b only job = 1 means get b, and the integer arrays ib, jb.

nrm = integer. norm indicator. nrm = 1, means 1-norm, nrm =2 means the 2-nrm, nrm = 0 means max norm

a,ja,ia = Matrix A in compressed sparse row format.

ON RETURN :

diag = diagonal matrix stored as a vector containing the matrix by which the rows have been scaled, i.e., on return we have B = Diag∗A.

b,jb,ib = resulting matrix B in compressed sparse row sparse format.

ierr = error message. ierr=0 : Normal return

ierr=i > 0 : Row number i is a zero row.

NOTES :

1) The column dimension of A is not needed.

2) algorithm in place (B can take the place of A).

4.45.1.36 rperm (nrow, a, ja, ia, ao, jao, iao, perm, job)

This permutes the rows of a matrix in CSR format. rperm computes B = P A where P is a permutation matrix. the permutation P is defined through the array perm: for each j, perm(j) represents the destination row number of row number j.

ON ENTRY :

n = dimension of the matrix

a, ja, ia = input matrix in csr format

perm = integer array of length nrow containing the permutation arrays for the rows: perm(i) is the destination of row i in the permuted matrix. —> a(i,j) in the original matrix becomes a(perm(i),j) in the output matrix.

job = integer indicating the work to be done:

job = 1 permute a, ja, ia into ao, jao, iao (including the copying of real values ao and the array iao).

job .ne. 1 : ignore real values. (in which case arrays a and ao are not needed nor used).

ON RETURN :

ao, jao, iao = input matrix in a, ja, ia format

NOTE :

if (job.ne.1) then the arrays a and ao are not used.

4.45.1.37 rversp (n, riord)

This routine does an in-place reversing of the permutation array riord

4.45.1.38 stripes (nlev, riord, levels, ip, map, mapptr, ndom)

This is a post processor to BFS. stripes uses the output of BFS to find a decomposition of the adjacency graph by stripes. It fills the stripes level by level until a number of nodes .gt. ip is is reached.

ON ENTRY :

nlev = number of levels as found by BFS

riord = reverse permutation array produced by BFS –

levels = pointer array for the level structure as computed by BFS. If lev is a level number, and k1=levels(lev),k2=levels(lev+1)-1, then all the nodes of level number lev are: riord(k1),riord(k1+1),...,riord(k2)

ip = number of desired partitions (subdomains) of about equal size.

ON RETURN :

ndom = number of subgraphs (subdomains) found

map = node per processor list. The nodes are listed contiguously from proc 1 to nproc = mpx∗mpy.

mapptr = pointer array for array map. list for proc. i starts at mapptr(i) and ends at mapptr(i+1)-1 in array map.

4.45.1.39 stripes0 (ip, nlev, il, ndom, iptr)

This routine is a simple level-set partitioner. It scans the level-sets as produced by BFS from one to nlev. each time the number of nodes in the accumulated set of levels traversed exceeds the parameter ip, this set defines a new subgraph.

ON ENTRY :

ip = desired number of nodes per subgraph.

nlev = number of levels found as output by BFS

il = integer array containing the pointer array for the level data structure as output by BFS. thus il(lev+1) - il(lev) = the number of nodes that constitute the level numbe lev.

ON RETURN :

ndom = number of sungraphs found

iptr = pointer array for the sugraph data structure. thus, iptr(idom) points to the first level that consistutes the subgraph number idom, in the level data structure.

4.45.1.40 wreadmtc (nmax, nzmax, job, fname, length, a, ja, ia, rhs, nrhs, guesol, nrow, ncol, nnz, title, key, type, ierr)

This reads a boeing/harwell matrix, given the corresponding file. handles right hand sides in full format only (no sparse right hand sides). Also the matrix must be in assembled forms. It differs from readmt, in that the name of the file needs to be passed, and then the file is opened and closed within this routine.

4.45.1.41 xtrows (i1, i2, a, ja, ia, ao, jao, iao, iperm, job)

This subroutine extracts given rows from a matrix in CSR format. Specifically, rows number iperm(i1), iperm(i1+1), ..., iperm(i2) are extracted and put in the output matrix ao, jao, iao, in CSR format. NOT in place.

ON ENTRY :

i1,i2 = two integers indicating the rows to be extracted. xtrows will extract rows iperm(i1), iperm(i1+1),..,iperm(i2), from original matrix and stack them in output matrix ao, jao, iao in csr format

a, ja, ia = input matrix in csr format

iperm = integer array of length nrow containing the reverse permutation array for the rows. row number iperm(j) in permuted matrix PA used to be row number j in unpermuted matrix. —> a(i,j) in the permuted matrix was a(iperm(i),j) in the inout matrix.

job = integer indicating the work to be done: job .ne. 1 : get structure only of output matrix,, i.e., ignore real values. (in which case arrays a and ao are not used nor accessed). job = 1 get complete data structure of output matrix. (i.e., including arrays ao and iao).

ON RETURN :

ao, jao, iao = input matrix in a, ja, ia format

NOTE :

if (job.ne.1) then the arrays a and ao are not used.

4.46 solver.c File Reference

```
#include "../../INCLUDE/psparslib.h"
```

Functions

- int **pgmres** (**DistMatrix** dm, **PreCon** precon, **IterPar** ipar, **Vec** rhs, **Vec** x)

4.46.1 Function Documentation

4.46.1.1 int pgmres (DistMatrix *dm*, PreCon *precon*, IterPar *ipar*, Vec *rhs*, Vec *x*)

Preconditioned GMRES as preconditioning operation for FGMRES

ON ENTRY :

dm = distributed matrix output from setup

precon = a pointer to precon strucutre

ipar = a pointer to the structure contains parameters(maxits,eps,etc.) related to iteration

rhs = right hand side vector

ON RETURN :

x = local solution vector handler

4.47 time.c File Reference

```
#include <sys/time.h>
#include "../../INCLUDE/psparslib.h"
```

Functions

- double **dwalltime** ()

4.47.1 Function Documentation

4.47.1.1 double dwalltime (void)

Wallclock timer function

dwaltime : time in seconds since 00:00:00 UTC, Jan. 1st, 1970

4.48 tools.f File Reference

Functions

- **nod2dom** (n, ndom, nodes, pointer, map, mapsize, ier)
- **getjamap** (nnz, ja, map, jamap, mapsize, ierr)
- **getjamap1** (n, ndom, nodes, pointer, nnzloc, ja, map, lenmap, jamap, lenjamap, ierr)
- **expnddom** (n, ndom, ja, ia, riord, iptr, riordo, maxout, iptro, iwk)

4.48.1 Function Documentation

4.48.1.1 expnddom (n, ndom, ja, ia, riord, iptr, riordo, maxout, iptro, iwk)

Expands each subdomain by adding external points (one level)

ON ENTRY :

n = total number of input nodes –

ndom = number of subdomains to be expanded.

ja,ia = matrix pattern information. riord,

iptr = subdomain data structure as obtained from get_domns

maxout = maximum number of points allowed on return. This is the length of the array riordo – maxout is also a return argument, if maxout < 0 it means that the algorithm failed because the number of points after expansion exceeds maxout..

ON RETURN :

riordo = contains the nodes in each domain in succession.

iptro = pointer in riord for beginnning of each domain. Thus domain number i consists of nodes riordo(k1),riordo(k1+1),...,riordo(k2) where k1 = iptro(i), k2 = iptro(i+1)-1

WORK ARRAYS :

iwk(n)

4.48.1.2 getjamap (nnz, ja, map, jamap, mapsize, ierr)

Given the map array [node to subdomain information], and the list of nnz nodes in ja, this routines constructs the node-to subdomain information for each of the items in ja. In other it constructs a similar array as map for the items in ja.

ON ENTRY :

nnz = number of elements in ja($*$)

ja = array of nodes to be mapped

map = map array as produced by nod2dom – see nod2proc for data structure used.

ON RETURN :

jamap = subdomain information for each item in ja. the length of jamap should be the sum over all items of the number of subdomains to which these items belong.. In the non-overlap case, this is the same as nnz.

if jamap(i) .gt. 0 then map(i) = (unique) domain to which node ja(i) belongs.

if map(i) .lt. 0 then -map(i) contains the start address of a linked list for all domains to which node ja(i) belongs ierr > 0 for normal return, actual length of the list = -2 length of jamap was insufficient.

4.48.1.3 getjamap1 (n, ndom, nodes, pointer, nnzloc, ja, map, lenmap, jamap, lenjamap, ierr)

NOTE :

This combines the routines nod2dom and getjamap together..

Given the mapping information and the list of nnzloc cloumn indices in the ja array assigned to a processor, this routines will constructs the node-to subdomain information for each of the items in ja. It will give a list

In other it constructs a similar array as map for the items in ja.

ON ENTRY :

n = number of nodes for global problem.

ndom = number of subdomains

nodes = mapping information. If i1=pointer(k), i2=pointer(k+1)-1, then the nodes in nodes(i1:i2) are assigned to subdomain k

pointer= pointer(k) points to beginning of domain k (see above)

nnzloc = number of elements in ja(∗)

ja = array of nodes to be mapped

map = map array as produced by nod2dom – see nod2proc for data structure used.

lenmap = length of array map as declared from calling program

lenjamap= length of array jamap as declared from calling program

ON RETURN :

jamap = subdomain information for each item in ja. the length of jamap should be the sum over all items of the number of subdomains to which these items belong.. In the non-overlap case, this is the same as nnzloc.

if jamap(i) .gt. 0 then map(i) = (unique) domain to which node ja(i) belongs.

if map(i) .lt. 0 then -map(i) contains the start address of a linked list for all domains to which node ja(i) belongs

map = contains the node to processor mapping array.

if map(i) .gt. 0 then map(i) = (unique) processor to which node i belongs.

if map(i) .lt. 0 then - map(i) contains the beginning of a a linked list for all processors to which node i belongs

ierr = 0 for normal return

-1 length of map was insufficient (returned from nod2dom)

-2 length of jamap was insufficient. / }

getpro(item, map, number, list) { /∗∗ Given the map array this routine obtains the number of neighboring subdomains and their list for entry in position item in map.

ON ENTRY :

item = subdomain list for which map(item) is searched

map = map array as produded by nod2map, or jamap

ON RETURN :

number = number of subdomains found

list = list(1:number) is the list of subdomains found

4.48.1.4 nod2dom (n, ndom, nodes, pointer, map, mapsize, ier)

Constructs the node-to-domain data structure (possibly a linked list) given the list-pointer data structure of the partitioning

input = pointer array data structure for each subdomain. nodes(j) for j= pointer(ii), pointer(ii+1)-1 are all the nodes in domain ii.

output = The information we want is given a node j, to which processor(s) it belongs. This is made complicated by overlapping (j may belong to several processors) for any node j, map(j) = processor number to which j is assigned. If there are more than one processor to which node j is assigned then the list of processors is given as a linked list. see details below.

ON ENTRY :

n = number of nodes

ndom = number of subdomains

nodes = sequence of nodes are produced by mapper4.

pointer= pointer(i) points to beginning of domain i in array nodes.

ON RETURN :

map = contains the node to processor mapping array.

if map(i) .gt. 0 then map(i) = (unique) processor to which node i belongs.

if map(i) .lt. 0 then -map(i) contains the beginning of a a linked list for all processors to which node i belongs

ier = serves as an error indicator. It also returns space used in array map if return is successful.

ier = -1 indicates an error – not enough storage in map

ier .gt. 0 is a normal return – ier is the actual length of the array map.

4.49 uread.f File Reference

Functions

- **userread** (matrix, len, job, n, nnz, a, ja, ia, nrhs, rhs, ierr)

4.49.1 Function Documentation

4.49.1.1 userread (matrix, len, job, n, nnz, a, ja, ia, nrhs, rhs, ierr)

User-defined read function:

ON ENTRY :

matrix : file containing names of the files with matrix input

len : size of variable matrix

job : flag: if 0 return the sizes only; if 1 read the matrix information.

nrhs : if nrhs is zero, then no RHS is read from file

ON RETURN :

n : number of unknowns

nnz : number of nonzeros

(a,ja,ia) matrix in CSR format, returned only if job=1

nrhs (integer) number of rhs

rhs (double) right-hand side

ierr (integer) not equal zero if read error occurs

4.50 vec.c File Reference

```
#include "../../INCLUDE/psparslib.h"
```

```
#include "../../INCLUDE/defs.h"
```

Functions

- void **CreateVec** (**Vec** ∗x)
- void **VecAssign** (double ∗x, **Vec** vec)
- void **DeleteVec** (**Vec** ∗x)
- void **PrintVec** (**Vec** x, char ∗base)
- void **VecSetVal** (**Vec** x, double b)
- void **VecSetFunc** (**Vec** x, double lv, double rv, double(∗func)(double y))
- double **VecNorm2** (**Vec** x)
- double **ResiNorm2** (**DistMatrix** dm, **Vec** sol, **Vec** rhs)

Variables

- **_Vec_ComOps vec_comops**

4.50.1 Function Documentation

4.50.1.1 void CreateVec (Vec ∗ *x*)

Create a Vec handler

ON ENTRY :

x = a pointer to Vec handler

ON RETURN :

x = allocate memory for x and initialize the members of the structure

4.50.1.2 void DeleteVec (Vec ∗ *x*)

Free memory allocated for local vector handler

ON ENTRY :

x = a pointer to the Vec handler

ON RETURN :

free the memory allocated for the handler pointed by x

4.50.1.3 void PrintVec (Vec *x*, char ∗ *base*)

Output vector to a file with name 'base' Note processor i output data to the file 'base.i'

ON ENTRY :

x = distributed vector

base = base name of a file

ON RETURN :

output local vector to the file 'base.i'. the format is: local_label global_label value

4.50.1.4 double ResiNorm2 (DistMatrix *dm*, Vec *sol*, Vec *rhs*)

Return residual norm ||rhs-dm∗sol||

ON ENTRY :

dm = local matrix

sol = solution vector

rhs = right hand side

ON RETURN :

return residual norm

4.50.1.5 void VecAssign (double ∗ *x*, Vec *vec*)

Assign double array pointed by x to vec

ON ENTRY : x = double array vec = distributed vec

ON RETURN : vec = contains the values stored in x

4.50.1.6 double VecNorm2 (Vec *x*)

Return the norm of the global vector

ON ENTRY :

x = distributed local vector handler

ON RETURN :

return the norm of global vector

NOTE : T

This a collective function, all processors which contain parts of global vector should call this function

4.50.1.7 void VecSetFunc (Vec *x*, double *lv*, double *rv*, double(∗ *func*)(double y))

Set every components of vector x to the value returned by function func whose domain is [lv, rv]

ON ENTRY :

x = distributed local vector

lv,rv = the domain of function y, [lv,rv]

y = a function pointer to a function, such as sin, cos, etc.

ON RETURN :

x[i] = i∗(rv-lv)/n, n is the dimension of the vector

4.50.1.8 void VecSetVal (Vec *x*, double *b*)

Set all components of vector x to value b

ON ENTRY :

b – scalar, a value should be set to every component of vector

ON RETURN :

x – distributed local vector

4.50.2 Variable Documentation

4.50.2.1 _Vec_ComOps vec_comops

Initial value:

```
 {
  MSG_bdx_bsend,
  Mtype_Create,
  Mtype_Free
}
```

4.51 wreadmtpar.f File Reference

Functions

- **wreadmtpar** (job, fname, length, a, ja, ia, lia, lian, rhs, nrhs, guesol, nrow, ncol, nnz, lnnz, title, key, type, nyrange, npro, mypro, ierr)

4.51.1 Function Documentation

4.51.1.1 wreadmtpar (job, fname, length, a, ja, ia, lia, lian, rhs, nrhs, guesol, nrow, ncol, nnz, lnnz, title, key, type, nyrange, npro, mypro, ierr)

This reads a boeing/harwell matrix in parallel, given the corresponding file. handles right hand sides in full format only (no sparse right hand sides). Also the matrix must be in assembled forms.

The arguments are

job - integer to indicate what is to be read. (note: job is an input and output parameter, it can be modified on return)

job = 0 read the values of ncol, nrow, nnz,lnnz, title, key, type and return. matrix is not read and arrays a, ja, ia, rhs are not touched.

job = 1&2 read matrix i.e., the arrays a ja and ia.

job = 3 read matrix and right hand sides: a,ja,ia,rhs. rhs may contain initial guesses and exact solutions appended to the actual right hand sides. this will be indicated by the output parameter guesol [see below].

fname (input)- file name which has to be read

length(input)- filename lenght

a (output)- elements in part of the matrix to be read from file

ja (output)- row indices to be read by one processor

ia (output)- all column pointers of the matrix of size n+1

lia (output)- local column pointers of the matrix

lian (output)- number of elements in lia

rhs (output)- right hand side of the matrix

nrhs (output & input)- integer. nrhs is an input as well as ouput parameter at input nrhs contains the total length of the array rhs. See also ierr and nrhs in output parameters.

guesol(output) = a 2-character string indicating whether an initial guess (1-st character) and / or the exact solution (2-nd character) is provided with the right hand side. if the first character of guesol is 'G' it means that an an intial guess is provided for each right-hand side. These are appended to the right hand-sides in the array rhs. if the second character of guesol is 'X' it means that an exact solution is provided for each right-hand side. These are appended to the right hand-sides and the initial guesses (if any) in the array rhs.

nrow (output)- number of rows of matrix

ncol (output)- number of column of matrix

nnz (output)- total number of non zero elements of matrix

lnnz (output)- array of size number of processors containg the local numbers of non zero elements

title (output) - character title of the matrix character $*$ 72

key (output)- character of length 8 byte (key to matrix)

type (output)- type of the matrix character$*$3

nyrange (output)- int array of size npro (number of processors reading the matrix and points the range of rows to be read by each processors (e.g. second processor will be reading the rows starting from nyrange(2) till nyrange(3)-1)

npro (integer input) - total number of processors

mypro (integer input) - processor id of local process

ierr (integer output)- return in case of error.

NOTE :

This routine reads the part of Matrix a,lia and ja from row starting nyrange(mypro) till nyrange(mypro)-1. It reads and stores ia and rhs in full. The output of the matrix has CSC format just like HB uses to store the matrix. Also the maxar is the parameter which could be modified here. The arrays of this size are work arrays to be used in order to read the matrix without assigning the big memory to the matrix arrays a, and ja. / }

wreadmtpar2(job,fname,length,a,ja,ia,lia,lian,rhs,nrhs,guesol,nrow,ncol,nnz,title,key,type,riord,riordn,ier
{ /** This reads a boeing/harwell matrix, given the corresponding file. handles right hand sides in full format only (no sparse right hand sides). Also the matrix must be in assembled forms. this subroutine reads a boeing/harwell matrix in parallel, given the corresponding file. handles right hand sides in full format only (no sparse right hand sides). Also the matrix must be in assembled forms.

The arguments are

job - integer to indicate what is to be read. (note: job is an input and output parameter, it can be modified on return)

job = 0 read the values of ncol, nrow, nnz,lnnz, title, key, type and return. matrix is not read and arrays a, ja, ia, rhs are not touched.

job = 1&2 read matrix i.e., the arrays a ja and ia.

job = 3 read matrix and right hand sides: a,ja,ia,rhs. rhs may contain initial guesses and exact solutions appended to the actual right hand sides. this will be indicated by the output parameter guesol [see below].

fname (input)- file name which has to be read

length(input)- filename lenght

a (output)- elements in part of the matrix to be read from file

ja (output)- row indices to be read by one processor

ia (output)- all column pointers of the matrix of size n+1

lia (output)- local column pointers of the matrix

lian (output)- number of elements in lia

rhs (output)- right hand side of the matrix

nrhs (output & input)- integer. nrhs is an input as well as ouput parameter at input nrhs contains the total length of the array rhs. See also ierr and nrhs in output parameters.

guesol(output) = a 2-character string indicating whether an initial guess (1-st character) and / or the exact solution (2-nd character) is provided with the right hand side. if the first character

of guesol is 'G' it means that an an intial guess is provided for each right-hand side. These are appended to the right hand-sides in the array rhs. if the second character of guesol is 'X' it means that an exact solution is provided for each right-hand side. These are appended to the right hand-sides and the initial guesses (if any) in the array rhs.

nrow (output)- number of rows of matrix

ncol (output)- number of column of matrix

nnz (output)- total number of non zero elements of matrix

lnnz (output)- array of size number of processors containg the local numbers of non zero elements

title (output) - character title of the matrix character $*$ 72

key (output)- character of length 8 byte (key to matrix)

type (output)- type of the matrix character$*$3

riord (input) - the permutation integer array indicating the rows which are to be picked up by the local processor.

riordn (input) - the length of array riord.

ierr (integer output)- return in case of error.

NOTE :

This routine reads the part of Matrix a,lia and ja i.e. row numbers contained in riord array. It reads and stores ia and rhs in full. The output of the matrix has CSC format just like HB uses to store the matrix. also the maxar is the parameter which could be modified here. The arrays of this size are work arrays to be used in order to read the matrix without assigning the big memory to the matrix arrays a, and ja.

Index

_Comm
data.h, 80
_Csr
data.h, 80
_CsrSpar
data.h, 80
_DATA_HeADER_
data.h, 78
_DistMatrix
data.h, 80
_DistMatrixCsr
data.h, 80
_Ext_Max
data.h, 80
_Hash
data.h, 80
_HashOps
data.h, 80
_ILUfac
data.h, 80
_IterPar
data.h, 80
_MatOps
data.h, 80
_Mix_Schur
data.h, 80
_MultiSch
data.h, 80
_MultiSchIlu
data.h, 80
_Pilu_Comm
data.h, 80
_PreCon
data.h, 80
_PrePar
data.h, 80
_SchPilu
data.h, 80
_Sparse_Matrix_Storage_Format
data.h, 80
_Vec
data.h, 80
_Vec_ComOps
data.h, 80
_Vec_Comm
data.h, 80
_p_Comm, 5
color, 5
dtype, 5
dtype_flag, 5
ipr, 5
ix, 6
mpi_comm, 6
myproc, 6
nbnd, 6
nl, 6
nloc, 6
npro, 6
nproc, 6
ovp, 6
proc, 6
_p_Csr, 7
alusize, 7
ia, 7
ja, 7
ma, 7
n, 7
_p_CsrSpar, 8
ops, 8
smsf, 8
_p_CsrSpar
ops, 8
smsf, 8
_p_DistMatrix, 9
A, 9
comm, 9
hash, 9
map, 9
node, 9
perm, 9
type, 9
X, 9
_p_DistMatrix
A, 9
comm, 9
hash, 9
map, 9
node, 9
perm, 9
type, 9

X, 9
_p_DistMatrix::_p_Hash, 10
hash_ops, 10
hash_size, 10
hash_table, 10
_p_DistMatrix::_p_Hash
hash_ops, 10
hash_size, 10
hash_table, 10
_p_DistMatrix::_p_Hash::_p_HashOps, 11
createhash, 11
freehash, 11
gethashvalue, 11
printhash, 11
storeinhash, 11
_p_DistMatrix::_p_Hash::_p_HashOps
createhash, 11
freehash, 11
gethashvalue, 11
printhash, 11
storeinhash, 11
_p_DistMatrix::_p_Sparse_Matrix_Storage_-Format, 12
ops, 12
smsf, 12
_p_DistMatrix::_p_Sparse_Matrix_Storage_-Format
ops, 12
smsf, 12
_p_DistMatrix::_p_Sparse_Matrix_Storage_-Format::_p_MatOps, 13
amxdis, 13
bdry, 13
copycsrtodm, 13
createmat, 13
deletemat, 13
getmap, 13
getvalofdim, 14
getvalofnnz, 14
lamux, 14
printmat, 14
setup, 14
_p_DistMatrix::_p_Sparse_Matrix_Storage_-Format::_p_MatOps
amxdis, 13
bdry, 13
copycsrtodm, 13
createmat, 13
deletemat, 13
getmap, 13
getvalofdim, 14
getvalofnnz, 14
lamux, 14
printmat, 14
setup, 14
_p_DistMatrixCsr, 15
A, 15
comm, 15
hash, 15
map, 15
node, 15
perm, 15
type, 15
X, 15
_p_DistMatrixCsr
A, 15
comm, 15
hash, 15
map, 15
node, 15
perm, 15
type, 15
X, 15
_p_Ext_Max, 16
L, 16
U, 16
_p_ILUfac, 17
L, 17
U, 17
_p_IterPar, 18
in_iters, 18
ipar, 18
iters, 18
pgfpar, 19
_p_IterPar
in_iters, 18
ipar, 18
iters, 18
pgfpar, 19
_p_Mix_Schur, 20
ext_max, 20
int_max, 20
_p_MultiSch, 21
ilsch, 21
levmat, 21
_p_MultiSch
ilsch, 21
levmat, 21
_p_MultiSchIlu, 22
levmat, 22
schpilu, 22
_p_MultiSchIlu
levmat, 22
schpilu, 22
_p_Pilu_Comm, 23
color, 23
ipr, 23
ix, 23

mpi_comm, 24
myproc, 24
nbnd, 24
ncolor, 24
nloc, 24
nodes_recv, 24
np_hcolor, 24
np_lcolor, 24
npro, 24
nproc, 24
nrecv, 24
proc, 24
proc_hcolor, 24
proc_lcolor, 25
request, 25
rhdtype, 25
rldtype, 25
shdtype, 25
sldtype, 25
snddtype, 25
status, 25
tag, 25
_p_PrePar, 28
droptol, 28
ipar, 28
lfil, 29
mc, 29
tolind, 29
_p_PrePar
droptol, 28
ipar, 28
lfil, 29
mc, 29
tolind, 29
_p_Precon, 26, 27
_p_SchPilu, 30
mschur, 30
pcomm, 30
_p_SchPilu
mschur, 30
pcomm, 30
_p_Vec, 31
alias, 31
ext_node, 31
node, 31
perm, 31
request, 31
vec, 31
vec_comm, 31
_p_Vec::_p_Vec_Comm, 33
comm, 33
vec_comops, 33
_p_Vec::_p_Vec_Comm::_p_Vec_ComOps, 34
msg_bdx_bsend, 34
mtype_create, 34
mtype_free, 34
_p_Vec::_p_Vec_Comm::_p_Vec_ComOps
msg_bdx_bsend, 34
mtype_create, 34
mtype_free, 34
_precon
defs.h, 97
precon.c, 142

A
_p_DistMatrix, 9
_p_DistMatrix, 9
_p_DistMatrixCsr, 15
_p_DistMatrixCsr, 15
add2com
armsheads.h, 47
indsetC.c, 120
indsetC.c, 120
add2is
armsheads.h, 47
indsetC.c, 120
indsetC.c, 120
add_arms
data.h, 78
add_ilu0
data.h, 78
add_iluk
data.h, 78
add_ilut
data.h, 78
add_lk
skitf.f, 180
add_lvst
skitf.f, 180
alias
_p_Vec, 31
all4mat
armsheads.h, 47
alusize
_p_Csr, 7
amub
skitf.f, 180
amudia
skitf.f, 181
amux
amxdis.c, 41
psparslib.h, 146
amux1
amxdis.c, 41
psparslib.h, 146
amuxe
amxdis.c, 42
psparslib.h, 146

amxdis
_p_DistMatrix::_p_Sparse_Matrix_-Storage_Format::_p_MatOps, 13
_p_DistMatrix::_p_Sparse_Matrix_-Storage_Format::_p_MatOps, 13
amxdis.c, 42
psparslib.h, 147
amxdis.c, 41
amux, 41
amux1, 41
amuxe, 42
amxdis, 42
aplb
skitf.f, 181
aps.c, 43
lsch, 43
rsch, 43
arms2
arms2.c, 44
psparslib.h, 147
arms2.c, 44
arms2, 44
MBLOC, 44
OPTION, 44
PERMTOL, 44
armschsol
armsheads.h, 48
psparslib.h, 148
armsheads.h, 46
add2com, 47
add2is, 47
all4mat, 47
armschsol, 48
armsol2, 48
ascend, 49
cleanARMS, 49
cleanCS, 49
cleanILUT, 49
cleanP4, 49
copymat, 49
copymat2, 50
coscalC, 50
cpermC, 50
cs_nnz, 50
cscpy, 50
csPer, 51
CSRcs, 51
csSplit4, 51
descend, 51
dlusol, 52
dpermC, 52
dpgmr, 52
dscale, 52
exbound, 52
giluLsol, 52
gilupgmr, 53
giluUsol, 54
ilutD, 54
ilutpC, 54
indsetC, 55
lev4_nnz, 56
levset, 56
Lsol, 56
Lsolp, 56
lusolD, 56
lusolDp, 57
matvec, 57
matvecz, 58
nnz_arms, 58
nnz_arms1, 58
pgmr, 58
pilu, 59
pILUTmat, 60
plperm, 60
plSpar, 60
printmat, 60
qsortC, 60
qsplitC, 60
roscalC, 60
rpermC, 61
schpart, 61
schsgssol, 61
schuramux, 62
schurprod, 62
setupBLU, 63
setupCS, 63
setupILUT, 63
sgslusol, 64
sgspgmr, 64
sgsschur, 65
sparmat, 65
SparTran, 65
swapj, 65
swapm, 65
Usol, 65
Usolp, 65
weightsC, 66
armsol2
armsheads.h, 48
armsol2.c, 67
armsol2.c, 67
armsol2, 67
EPSILON, 67
EPSMAC, 67
epsmac, 67
Lsolp, 68
lusolDp, 69
pgmr, 69

schuramux, 70
Usolp, 70
ZERO, 67
ascend
armsheads.h, 49
matrops.c, 128
assignprecon
setpar.c, 169
atmux
skitf.f, 182
atmuxr
skitf.f, 182

base.h, 72
DAXPY, 72
Daxpy, 72, 73
DCOPY, 73
Dcopy, 73
DDOT, 73
Ddot, 73
DNRM2, 73
Dnrm2, 73
DSCAL, 73
Dscal, 73
IDMIN, 73
Idmin, 73
bcgstabd
iters.c, 122
bdry
_p_DistMatrix::_p_Sparse_Matrix_-Storage_Format::_p_MatOps, 13
_p_DistMatrix::_p_Sparse_Matrix_-Storage_Format::_p_MatOps, 13
psparslib.h, 148
setup.c, 174
BFS
skitf.f, 182
BUFLEN
dd-grid-edge.c, 83
dd-grid-simple.c, 85
dd-grid-solver.c, 88
dd-grid.c, 90
dd-HB-dse.c, 93
dd-HB-metis.c, 94
dd-HB-parmetis.c, 95
dd-HB-simple.c, 92
setpar.c, 169

C
ILUTfac, 36
CHECKERR
data.h, 78
cleanARMS
armsheads.h, 49
sets.c, 170
cleanCS
armsheads.h, 49
sets.c, 170
cleanILUT
armsheads.h, 49
sets.c, 170
cleanP4
armsheads.h, 49
sets.c, 170
cnrms
skitf.f, 183
color
_p_Comm, 5
_p_Pilu_Comm, 23
Comm
data.h, 80
comm
_p_DistMatrix, 9
_p_DistMatrix, 9
_p_DistMatrixCsr, 15
_p_DistMatrixCsr, 15
_p_Vec::_p_Vec_Comm, 33
comm.c, 74
ErrHand, 74
MSG_bdx_bsend, 74
Mtype_Create, 74
Mtype_Free, 74
PARMS_Final, 75
PARMS_Init, 75
COMM_TAG
data.h, 78
Communication, 35
consis
psparslib.h, 149
CopyComm
matrix.c, 125
CopyCsrToDm
matrix.c, 125
psparslib.h, 149
copycsrtodm
_p_DistMatrix::_p_Sparse_Matrix_-Storage_Format::_p_MatOps, 13
_p_DistMatrix::_p_Sparse_Matrix_-Storage_Format::_p_MatOps, 13
copymat
armsheads.h, 49
copymat2
armsheads.h, 50
coscal
skitf.f, 183
coscalC
armsheads.h, 50
skitc.c, 176

cperm
skitf.f, 184
cpermC
armsheads.h, 50
skitc.c, 176
CreateHash
hash.c, 112
psparslib.h, 149
createhash
_p_DistMatrix::_p_Hash::_p_HashOps, 11
_p_DistMatrix::_p_Hash::_p_HashOps, 11
CreateMat
matrix.c, 125
psparslib.h, 149
createmat
_p_DistMatrix::_p_Sparse_Matrix_-Storage_Format::_p_MatOps, 13
_p_DistMatrix::_p_Sparse_Matrix_-Storage_Format::_p_MatOps, 13
CreatePrec
precon.c, 137
psparslib.h, 149
CreateVec
psparslib.h, 150
vec.c, 208
cs_nnz
armsheads.h, 50
memus.c, 131
cscpy
armsheads.h, 50
sets.c, 171
CSORT
psparslib.h, 146, 150
csort
skitf.f, 184
csPer
armsheads.h, 51
csptr
heads.h, 114
Csr
data.h, 80
CSRcs
armsheads.h, 51
sets.c, 171
csrcsc
skitf.f, 185
csrcsc2
skitf.f, 185
CsrSpar
data.h, 80
csSplit4
armsheads.h, 51
sets.c, 171
D1
ILUTfac, 36
PerMat4, 38
PerMat4, 38
D2
ILUTfac, 36
PerMat4, 38
PerMat4, 38
data.h, 76
_Comm, 80
_Csr, 80
_CsrSpar, 80
_DATA_HeADER_, 78
_DistMatrix, 80
_DistMatrixCsr, 80
_Ext_Max, 80
_Hash, 80
_HashOps, 80
_ILUfac, 80
_IterPar, 80
_MatOps, 80
_Mix_Schur, 80
_MultiSch, 80
_MultiSchIlu, 80
_Pilu_Comm, 80
_PreCon, 80
_PrePar, 80
_SchPilu, 80
_Sparse_Matrix_Storage_Format, 80
_Vec, 80
_Vec_ComOps, 80
_Vec_Comm, 80
add_arms, 78
add_ilu0, 78
add_iluk, 78
add_ilut, 78
CHECKERR, 78
Comm, 80
COMM_TAG, 78
Csr, 80
CsrSpar, 80
DistMatrix, 80
DistMatrixCsr, 80
Ext_Max, 80
FALSE, 78
Hash, 80
HashOps, 80
ILUfac, 80
IterPar, 80
lsch_arms, 78
lsch_ilu0, 78
lsch_iluk, 78
lsch_ilut, 78
MatOps, 80

Mix_Schur, 80
MultiSch, 80
MultiSchIlu, 80
PARMS_malloc, 78
PARMS_realloc, 78
Pilu_Comm, 80
PREC, 80
PreCon, 80
PRECOND_ROUTINE, 80
PrePar, 81
rsch_arms, 79
rsch_ilu0, 80
rsch_iluk, 80
rsch_ilut, 80
sch_gilu0, 80
sch_sgs, 80
SchPilu, 81
Sparse_Matrix_Storage_Format, 81
TRUE, 80
TYPESIZE, 80
Vec, 81
Vec_Comm, 81
Vec_ComOps, 81
DAXPY
base.h, 72
Daxpy
base.h, 72, 73
dbcgstab
psparslib.h, 150
dblstr
skitf.f, 186
DCOPY
base.h, 73
Dcopy
base.h, 73
dd-grid-edge.c, 82
BUFLEN, 83
dwalltime, 83
gen5loc, 83
main, 83
part2, 83, 84
partedge, 83, 84
printm, 84
setinit, 83, 84
setpar, 84
SOLVER, 83
dd-grid-simple.c, 85
BUFLEN, 85
dwalltime, 85
gen5loc, 85
main, 86
part1, 85, 86
part2, 85, 86
printm, 86
setinit, 85, 86
dd-grid-solver.c, 87
BUFLEN, 88
dwalltime, 88
EDGE, 88
gen5loc, 88
main, 88
part2, 88, 89
part_edge, 88, 89
printm, 89
setinit, 88, 89
setpar, 89
SOLVER, 88
dd-grid.c, 90
BUFLEN, 90
dwalltime, 90
gen5loc, 90
main, 91
part1, 90, 91
part2, 90, 91
printm, 91
setinit, 90, 91
setpar, 91
dd-HB-dse.c, 93
BUFLEN, 93
main, 93
dd-HB-metis.c, 94
BUFLEN, 94
main, 94
dd-HB-parmetis.c, 95
BUFLEN, 95
main, 95
makeptr, 95
rordcsr, 96
setpar, 96
wreadmtpar, 95, 96
wreadmtpar2, 95, 96
dd-HB-simple.c, 92
BUFLEN, 92
main, 92
DDOT
base.h, 73
Ddot
base.h, 73
defs.h, 97
_precon, 97
hops, 97
prec_list, 97
sol0_dispatch, 97
vec_comops, 97
deigen
dgmr.f, 98
DeleteMat
matrix.c, 126

psparslib.h, 150
deletemat
_p_DistMatrix::_p_Sparse_Matrix_-Storage_Format::_p_MatOps, 13
_p_DistMatrix::_p_Sparse_Matrix_-Storage_Format::_p_MatOps, 13
DeletePrec
precon.c, 138
psparslib.h, 150
DeletePrecArms
precon.c, 138
psparslib.h, 151
DeletePrecGilu
precon.c, 138
psparslib.h, 151
DeletePrecIlu
precon.c, 138
psparslib.h, 151
DeleteVec
psparslib.h, 151
vec.c, 208
descend
armsheads.h, 51
matrops.c, 128
DGMR
psparslib.h, 146, 151
dgmr
dgmr.f, 98
dgmr.f, 98
deigen, 98
dgmr, 98
eigResid, 100
mgsr, 100
norder, 100
dgmresd
iters.c, 122
psparslib.h, 152
diamua
skitf.f, 187
DistMatrix
data.h, 80
DistMatrixCsr
data.h, 80
dlusol
armsheads.h, 52
gprecsol.c, 106
DNRM2
base.h, 73
Dnrm2
base.h, 73
dperm
skitf.f, 187
DPERM1
psparslib.h, 146, 152
dperm1
skitf.f, 188
dperm2
skitf.f, 188
dpermC
armsheads.h, 52
skitc.c, 177
dpgmr
armsheads.h, 52
droptol
_p_PrePar, 28
_p_PrePar, 28
DSCAL
base.h, 73
Dscal
base.h, 73
dscale
armsheads.h, 52
skitc.c, 177
DSE
psparslib.h, 146, 152
dse
skitf.f, 189
dse2way
skitf.f, 189
dtype
_p_Comm, 5
dtype_flag
_p_Comm, 5
dvperm
skitf.f, 189
dwalltime
dd-grid-edge.c, 83
dd-grid-simple.c, 85
dd-grid-solver.c, 88
dd-grid.c, 90
psparslib.h, 152
time.c, 203

E
PerMat4, 38
PerMat4, 38
EDGE
dd-grid-solver.c, 88
eigResid
dgmr.f, 100
EPSILON
armsol2.c, 67
gprecsol.c, 106
psparslib.h, 146
EPSMAC
armsol2.c, 67
gprecsol.c, 106
psparslib.h, 146

epsmac
armsol2.c, 67
ErrHand
comm.c, 74
exbound
armsheads.h, 52
gprecsol.c, 107
expnddom
tools.f, 204
Ext_Max
data.h, 80
ext_max
_p_Mix_Schur, 20
ext_node
_p_Vec, 31

F
PerMat4, 38
PerMat4, 38
FALSE
data.h, 78
fdmat.f, 102
gen5pt, 102
part0, 103
part1, 103
part2, 104
part_edge, 104
fgmresd
iters.c, 123
psparslib.h, 152
find_ctr
skitf.f, 190
FreeHash
hash.c, 112
psparslib.h, 152
freehash
_p_DistMatrix::_p_Hash::_p_HashOps, 11
_p_DistMatrix::_p_Hash::_p_HashOps, 11

gen5loc
dd-grid-edge.c, 83
dd-grid-simple.c, 85
dd-grid-solver.c, 88
dd-grid.c, 90
gen5pt
fdmat.f, 102
get_domns2
skitf.f, 190
GetHashValue
hash.c, 112
psparslib.h, 153
gethashvalue
_p_DistMatrix::_p_Hash::_p_HashOps, 11
_p_DistMatrix::_p_Hash::_p_HashOps, 11
GetIndSize
precon.c, 139
psparslib.h, 153
getjamap
tools.f, 204
getjamap1
tools.f, 205
getmap
_p_DistMatrix::_p_Sparse_Matrix_-
Storage_Format::_p_MatOps, 13
_p_DistMatrix::_p_Sparse_Matrix_-
Storage_Format::_p_MatOps, 13
psparslib.h, 153
setup.c, 174
GetValOfDim
matrix.c, 126
psparslib.h, 153
getvalofdim
_p_DistMatrix::_p_Sparse_Matrix_-
Storage_Format::_p_MatOps, 14
_p_DistMatrix::_p_Sparse_Matrix_-
Storage_Format::_p_MatOps, 14
GetValOfNnz
matrix.c, 126
psparslib.h, 153
getvalofnnz
_p_DistMatrix::_p_Sparse_Matrix_-
Storage_Format::_p_MatOps, 14
_p_DistMatrix::_p_Sparse_Matrix_-
Storage_Format::_p_MatOps, 14
giluLsol
armsheads.h, 52
gprecsol.c, 107
gilupgmr
armsheads.h, 53
gprecsol.c, 107
giluUsol
armsheads.h, 54
gprecsol.c, 108
gprecsol.c, 106
dlusol, 106
EPSILON, 106
EPSMAC, 106
exbound, 107
giluLsol, 107
gilupgmr, 107
giluUsol, 108
schgilusol, 108
schsgssol, 109
sgslusol, 110
sgspgmr, 110
ZERO, 106

Hash

data.h, 80
hash
_p_DistMatrix, 9
_p_DistMatrix, 9
_p_DistMatrixCsr, 15
_p_DistMatrixCsr, 15
hash.c, 112
CreateHash, 112
FreeHash, 112
GetHashValue, 112
PrintHash, 112
StoreInHash, 112
hash_ops
_p_DistMatrix::_p_Hash, 10
_p_DistMatrix::_p_Hash, 10
hash_size
_p_DistMatrix::_p_Hash, 10
_p_DistMatrix::_p_Hash, 10
hash_table
_p_DistMatrix::_p_Hash, 10
_p_DistMatrix::_p_Hash, 10
HashOps
data.h, 80
heads.h, 114
csptr, 114
IluSpar, 114
ilutptr, 114
p4ptr, 114
Per4Mat, 114
SparMat, 114
hops
defs.h, 97
matrix.c, 126
psparslib.h, 163

ia
_p_Csr, 7
IDMIN
base.h, 73
Idmin
base.h, 73
ilsch
_p_MultiSch, 21
_p_MultiSch, 21
ilu.c, 115
ilu0, 115
iluk, 115
ilut, 116
min, 115
ilu0
ilu.c, 115
psparslib.h, 154
ILUfac
data.h, 80
iluk
ilu.c, 115
psparslib.h, 154
iluNEW.c, 117
ilutD, 117
iluNEW.c
ilutD, 117
IluSpar
heads.h, 114
ilut
ilu.c, 116
psparslib.h, 154
ilutD
armsheads.h, 54
iluNEW.c, 117
iluNEW.c, 117
ILUTfac, 36
C, 36
D1, 36
D2, 36
L, 36
meth, 36
n, 36
perm2, 36
rperm, 37
U, 37
wk, 37
ilutpC
armsheads.h, 54
ilutpC.c, 118
ilutpC.c, 118
ilutpC.c, 118
ilutpC, 118
ilutpC.c
ilutpC, 118
ilutptr
heads.h, 114
in_iters
_p_IterPar, 18
_p_IterPar, 18
indsetC
armsheads.h, 55
indsetC.c, 120
indsetC.c, 120
indsetC.c, 120
add2com, 120
add2is, 120
indsetC, 120
weightsC, 120
indsetC.c
add2com, 120
add2is, 120
indsetC, 120
weightsC, 120

int_max
_p_Mix_Schur, 20
ipar
_p_IterPar, 18
_p_IterPar, 18
_p_PrePar, 28
_p_PrePar, 28
ipr
_p_Comm, 5
_p_Pilu_Comm, 23
IterPar
data.h, 80
iters
_p_IterPar, 18
_p_IterPar, 18
iters.c, 122
bcgstabd, 122
dgmresd, 122
fgmresd, 123
PRINT, 122
itersf.f, 124
pddot, 124
ivperm
skitf.f, 190
ix
_p_Comm, 6
_p_Pilu_Comm, 23

ja
_p_Csr, 7
SparRow, 40
SparRow, 40

L
_p_Ext_Max, 16
_p_ILUfac, 17
ILUTfac, 36
PerMat4, 38
PerMat4, 38
lamux
_p_DistMatrix::_p_Sparse_Matrix_-
Storage_Format::_p_MatOps, 14
_p_DistMatrix::_p_Sparse_Matrix_-
Storage_Format::_p_MatOps, 14
lev4_nnz
armsheads.h, 56
memus.c, 131
levmat
_p_MultiSch, 21
_p_MultiSch, 21
_p_MultiSchIlu, 22
_p_MultiSchIlu, 22
levset
armsheads.h, 56
lfil
_p_PrePar, 29
_p_PrePar, 29
lsch
aps.c, 43
psparslib.h, 155
lsch_arms
data.h, 78
lsch_ilu0
data.h, 78
lsch_iluk
data.h, 78
lsch_ilut
data.h, 78
Lsol
armsheads.h, 56
matrops.c, 128
Lsolp
armsheads.h, 56
armsol2.c, 68
lusolD
armsheads.h, 56
matrops.c, 129
psparslib.h, 155
lusolDp
armsheads.h, 57
armsol2.c, 69

ma
_p_Csr, 7
SparRow, 40
SparRow, 40
main
dd-grid-edge.c, 83
dd-grid-simple.c, 86
dd-grid-solver.c, 88
dd-grid.c, 91
dd-HB-dse.c, 93
dd-HB-metis.c, 94
dd-HB-parmetis.c, 95
dd-HB-simple.c, 92
makeptr
dd-HB-parmetis.c, 95
map
_p_DistMatrix, 9
_p_DistMatrix, 9
_p_DistMatrixCsr, 15
_p_DistMatrixCsr, 15
mapper4
skitf.f, 190
MatOps
data.h, 80
matrix.c, 125
CopyComm, 125

CopyCsrToDm, 125
CreateMat, 125
DeleteMat, 126
GetValOfDim, 126
GetValOfNnz, 126
hops, 126
PrintMat, 126
matrops.c, 128
ascend, 128
descend, 128
Lsol, 128
lusolD, 129
matvec, 129
matvecz, 129
schurprod, 130
Usol, 130
matvec
armsheads.h, 57
matrops.c, 129
matvecz
armsheads.h, 58
matrops.c, 129
MBLOC
arms2.c, 44
schgilu0.c, 168
mc
_p_PrePar, 29
_p_PrePar, 29
memus.c, 131
cs_nnz, 131
lev4_nnz, 131
nnz_arms, 131
meth
ILUTfac, 36
PerMat4, 38
PerMat4, 38
mgsr
dgmr.f, 100
mgsro
misc.f, 132
min
ilu.c, 115
mindom
skitf.f, 191
misc.f, 132
mgsro, 132
rgg, 132
Mix_Schur
data.h, 80
mpi_comm
_p_Comm, 6
_p_Pilu_Comm, 24
mschur
_p_SchPilu, 30
_p_SchPilu, 30
MSG_bdx_bsend
comm.c, 74
psparslib.h, 156
msg_bdx_bsend
_p_Vec::_p_Vec_Comm::_p_Vec_ComOps, 34
_p_Vec::_p_Vec_Comm::_p_Vec_ComOps, 34
Mtype_Create
comm.c, 74
psparslib.h, 156
mtype_create
_p_Vec::_p_Vec_Comm::_p_Vec_ComOps, 34
_p_Vec::_p_Vec_Comm::_p_Vec_ComOps, 34
Mtype_Free
comm.c, 74
psparslib.h, 156
mtype_free
_p_Vec::_p_Vec_Comm::_p_Vec_ComOps, 34
_p_Vec::_p_Vec_Comm::_p_Vec_ComOps, 34
multic
skitf.f, 191
multicD
psparslib.h, 156
setup.c, 174
MultiSch
data.h, 80
MultiSchIlu
data.h, 80
myproc
_p_Comm, 6
_p_Pilu_Comm, 24

n
_p_Csr, 7
ILUTfac, 36
PerMat4, 39
PerMat4, 39
SparRow, 40
SparRow, 40
nB
PerMat4, 39
PerMat4, 39
nbnd
_p_Comm, 6
_p_Pilu_Comm, 24
ncolor
_p_Pilu_Comm, 24
next

PerMat4, 39
PerMat4, 39
nl
_p_Comm, 6
nloc
_p_Comm, 6
_p_Pilu_Comm, 24
nnz_arms
armsheads.h, 58
memus.c, 131
nnz_arms1
armsheads.h, 58
nnzrow
SparRow, 40
SparRow, 40
nod2dom
tools.f, 206
node
_p_DistMatrix, 9
_p_DistMatrix, 9
_p_DistMatrixCsr, 15
_p_DistMatrixCsr, 15
_p_Vec, 31
nodes_recv
_p_Pilu_Comm, 24
norder
dgmr.f, 100
np_hcolor
_p_Pilu_Comm, 24
np_lcolor
_p_Pilu_Comm, 24
npro
_p_Comm, 6
_p_Pilu_Comm, 24
nproc
_p_Comm, 6
_p_Pilu_Comm, 24
nrecv
_p_Pilu_Comm, 24

ops
_p_CsrSpar, 8
_p_CsrSpar, 8
_p_DistMatrix::_p_Sparse_Matrix_-
Storage_Format, 12
_p_DistMatrix::_p_Sparse_Matrix_-
Storage_Format, 12
OPTION
arms2.c, 44
ovp
_p_Comm, 6

p4ptr
heads.h, 114
PARMS_Final
comm.c, 75
psparslib.h, 156
PARMS_Init
comm.c, 75
psparslib.h, 156
PARMS_malloc
data.h, 78
PARMS_realloc
data.h, 78
part0
fdmat.f, 103
part1
dd-grid-simple.c, 85, 86
dd-grid.c, 90, 91
fdmat.f, 103
part2
dd-grid-edge.c, 83, 84
dd-grid-simple.c, 85, 86
dd-grid-solver.c, 88, 89
dd-grid.c, 90, 91
fdmat.f, 104
part_edge
dd-grid-solver.c, 88, 89
fdmat.f, 104
partedge
dd-grid-edge.c, 83, 84
pcomm
_p_SchPilu, 30
_p_SchPilu, 30
pddot
itersf.f, 124
Per4Mat
heads.h, 114
perm
_p_DistMatrix, 9
_p_DistMatrix, 9
_p_DistMatrixCsr, 15
_p_DistMatrixCsr, 15
_p_Vec, 31
PerMat4, 39
PerMat4, 39
perm2
ILUTfac, 36
PerMat4, 38
D1, 38
D2, 38
E, 38
F, 38
L, 38
meth, 38
n, 39
nB, 39
next, 39

perm, 39
prev, 39
rperm, 39
U, 39
wk, 39
PerMat4
D1, 38
D2, 38
E, 38
F, 38
L, 38
meth, 38
n, 39
nB, 39
next, 39
perm, 39
prev, 39
rperm, 39
U, 39
wk, 39
PERMTOL
arms2.c, 44
schgilu0.c, 168
perphn
skitf.f, 191
pgfpar
_p_IterPar, 19
_p_IterPar, 19
pgmr
armsheads.h, 58
armsol2.c, 69
pgmres
psparslib.h, 157
solver.c, 202
pilu
armsheads.h, 59
piluNEW.c, 134
piluNEW.c, 134
Pilu_Comm
data.h, 80
piluNEW.c, 134
pilu, 134
qsplitC, 135
setupCS, 135
piluNEW.c
pilu, 134
qsplitC, 135
setupCS, 135
pILUTmat
armsheads.h, 60
plperm
armsheads.h, 60
plSpar
armsheads.h, 60
PREC
data.h, 80
prec
psparslib.h, 157
prec_arms
precon.c, 139
psparslib.h, 157
prec_dispatch
psparslib.h, 163
prec_gilu
precon.c, 139
psparslib.h, 157
prec_ilu
precon.c, 139
psparslib.h, 157
prec_list
defs.h, 97
precon.c, 142
PreCon
data.h, 80
precon.c, 137
_precon, 142
CreatePrec, 137
DeletePrec, 138
DeletePrecArms, 138
DeletePrecGilu, 138
DeletePrecIlu, 138
GetIndSize, 139
prec_arms, 139
prec_gilu, 139
prec_ilu, 139
prec_list, 142
sol0, 140
sol0_arms, 140
sol0_dispatch, 143
sol0_gilu, 140
sol0_ilu, 140
sol0_sgs, 141
sol0p, 141
sol0p_arms, 141
sol0p_ilu, 142
PRECOND_ROUTINE
data.h, 80
PrePar
data.h, 81
prev
PerMat4, 39
PerMat4, 39
PRINT
iters.c, 122
PrintHash
hash.c, 112
psparslib.h, 158
printhash

_p_DistMatrix::_p_Hash::_p_HashOps, 11
_p_DistMatrix::_p_Hash::_p_HashOps, 11
printm
dd-grid-edge.c, 84
dd-grid-simple.c, 86
dd-grid-solver.c, 89
dd-grid.c, 91
PrintMat
matrix.c, 126
psparslib.h, 158
printmat
_p_DistMatrix::_p_Sparse_Matrix_-Storage_Format::_p_MatOps, 14
_p_DistMatrix::_p_Sparse_Matrix_-Storage_Format::_p_MatOps, 14
armsheads.h, 60
skitc.c, 177
PrintVec
psparslib.h, 158
vec.c, 208
proc
_p_Comm, 6
_p_Pilu_Comm, 24
proc_hcolor
_p_Pilu_Comm, 24
proc_lcolor
_p_Pilu_Comm, 25
psparslib.h, 144
amux, 146
amux1, 146
amuxe, 146
amxdis, 147
arms2, 147
armschsol, 148
bdry, 148
consis, 149
CopyCsrToDm, 149
CreateHash, 149
CreateMat, 149
CreatePrec, 149
CreateVec, 150
CSORT, 146, 150
dbcgstab, 150
DeleteMat, 150
DeletePrec, 150
DeletePrecArms, 151
DeletePrecGilu, 151
DeletePrecIlu, 151
DeleteVec, 151
DGMR, 146, 151
dgmresd, 152
DPERM1, 146, 152
DSE, 146, 152
dwalltime, 152
EPSILON, 146
EPSMAC, 146
fgmresd, 152
FreeHash, 152
GetHashValue, 153
GetIndSize, 153
getmap, 153
GetValOfDim, 153
GetValOfNnz, 153
hops, 163
ilu0, 154
iluk, 154
ilut, 154
lsch, 155
lusolD, 155
MSG_bdx_bsend, 156
Mtype_Create, 156
Mtype_Free, 156
multicD, 156
PARMS_Final, 156
PARMS_Init, 156
pgmres, 157
prec, 157
prec_arms, 157
prec_dispatch, 163
prec_gilu, 157
prec_ilu, 157
PrintHash, 158
PrintMat, 158
PrintVec, 158
ResiNorm2, 158
rsch, 158
schgilu0, 159
setup, 159
setuprhs, 159
sol0, 160
sol0_arms, 160
sol0_gilu, 160
sol0_ilu, 161
sol0_sgs, 161
sol0p, 161
sol0p_arms, 161
sol0p_ilu, 162
StoreInHash, 162
VecAssign, 162
VecNorm2, 163
VecRand, 163
VecSetFunc, 163
VecSetVal, 163
ZERO, 146

qsortC
armsheads.h, 60
qsplitC

armsheads.h, 60
piluNEW.c, 135
piluNEW.c, 135
skitc.c, 177
qzhes
qzhes.f, 164
qzhes.f, 164
qzhes, 164
qzit
qzit.f, 165
qzit.f, 165
qzit, 165
qzval
qzval.f, 166
qzval.f, 166
qzval, 166
qzvec
qzvec.f, 167
qzvec.f, 167
qzvec, 167

rdis
skitf.f, 192
readmt
skitf.f, 192
readmt_c
skitf.f, 194
readmtc
skitf.f, 196
request
_p_Pilu_Comm, 25
_p_Vec, 31
ResiNorm2
psparslib.h, 158
vec.c, 209
rgg
misc.f, 132
rhdtype
_p_Pilu_Comm, 25
rldtype
_p_Pilu_Comm, 25
rnrms
skitf.f, 198
rordcsr
dd-HB-parmetis.c, 96
roscal
skitf.f, 199
roscalC
armsheads.h, 60
skitc.c, 177
rperm
ILUTfac, 37
PerMat4, 39
PerMat4, 39
skitf.f, 199
rpermC
armsheads.h, 61
skitc.c, 178
rsch
aps.c, 43
psparslib.h, 158
rsch_arms
data.h, 79
rsch_ilu0
data.h, 80
rsch_iluk
data.h, 80
rsch_ilut
data.h, 80
rversp
skitf.f, 200

sch_gilu0
data.h, 80
sch_sgs
data.h, 80
schgilu0
psparslib.h, 159
schgilu0.c, 168
schgilu0.c, 168
MBLOC, 168
PERMTOL, 168
schgilu0, 168
schgilusol
gprecsol.c, 108
schpart
armsheads.h, 61
SchPilu
data.h, 81
schpilu
_p_MultiSchIlu, 22
_p_MultiSchIlu, 22
schsgssol
armsheads.h, 61
gprecsol.c, 109
schuramux
armsheads.h, 62
armsol2.c, 70
schurprod
armsheads.h, 62
matrops.c, 130
set_def_params
setpar.c, 169
setinit
dd-grid-edge.c, 83, 84
dd-grid-simple.c, 85, 86
dd-grid-solver.c, 88, 89
dd-grid.c, 90, 91

setpar
dd-grid-edge.c, 84
dd-grid-solver.c, 89
dd-grid.c, 91
dd-HB-parmetis.c, 96
setpar.c, 169
setpar.c, 169
assignprecon, 169
BUFLEN, 169
set_def_params, 169
setpar, 169
sets.c, 170
cleanARMS, 170
cleanCS, 170
cleanILUT, 170
cleanP4, 170
cscpy, 171
CSRcs, 171
csSplit4, 171
setupCS, 172
setupILUT, 172
setupP4, 173
setup
_p_DistMatrix::_p_Sparse_Matrix_-Storage_Format::_p_MatOps, 14
_p_DistMatrix::_p_Sparse_Matrix_-Storage_Format::_p_MatOps, 14
psparslib.h, 159
setup.c, 175
setup.c, 174
bdry, 174
getmap, 174
multicD, 174
setup, 175
setuprhs, 175
setupBLU
armsheads.h, 63
setupCS
armsheads.h, 63
piluNEW.c, 135
piluNEW.c, 135
sets.c, 172
setupILUT
armsheads.h, 63
sets.c, 172
setupP4
sets.c, 173
setuprhs
psparslib.h, 159
setup.c, 175
sgslusol
armsheads.h, 64
gprecsol.c, 110
sgspgmr
armsheads.h, 64
gprecsol.c, 110
sgsschur
armsheads.h, 65
shdtype
_p_Pilu_Comm, 25
skitc.c, 176
coscalC, 176
cpermC, 176
dpermC, 177
dscale, 177
printmat, 177
qsplitC, 177
roscalC, 177
rpermC, 178
SparTran, 178
skitf.f, 179
add_lk, 180
add_lvst, 180
amub, 180
amudia, 181
aplb, 181
atmux, 182
atmuxr, 182
BFS, 182
cnrms, 183
coscal, 183
cperm, 184
csort, 184
csrcsc, 185
csrcsc2, 185
dblstr, 186
diamua, 187
dperm, 187
dperm1, 188
dperm2, 188
dse, 189
dse2way, 189
dvperm, 189
find_ctr, 190
get_domns2, 190
ivperm, 190
mapper4, 190
mindom, 191
multic, 191
perphn, 191
rdis, 192
readmt, 192
readmt_c, 194
readmtc, 196
rnrms, 198
roscal, 199
rperm, 199
rversp, 200

stripes, 200
stripes0, 200
wreadmtc, 200
xtrows, 201
sldtype
_p_Pilu_Comm, 25
smsf
_p_CsrSpar, 8
_p_CsrSpar, 8
_p_DistMatrix::_p_Sparse_Matrix_-Storage_Format, 12
_p_DistMatrix::_p_Sparse_Matrix_-Storage_Format, 12
snddtype
_p_Pilu_Comm, 25
sol0
precon.c, 140
psparslib.h, 160
sol0_arms
precon.c, 140
psparslib.h, 160
sol0_dispatch
defs.h, 97
precon.c, 143
sol0_gilu
precon.c, 140
psparslib.h, 160
sol0_ilu
precon.c, 140
psparslib.h, 161
sol0_sgs
precon.c, 141
psparslib.h, 161
sol0p
precon.c, 141
psparslib.h, 161
sol0p_arms
precon.c, 141
psparslib.h, 161
sol0p_ilu
precon.c, 142
psparslib.h, 162
SOLVER
dd-grid-edge.c, 83
dd-grid-solver.c, 88
solver.c, 202
pgmres, 202
SparMat
heads.h, 114
sparmat
armsheads.h, 65
SparRow, 40
ja, 40
ma, 40
n, 40
nnzrow, 40
SparRow
ja, 40
ma, 40
n, 40
nnzrow, 40
Sparse_Matrix_Storage_Format
data.h, 81
SparTran
armsheads.h, 65
skitc.c, 178
status
_p_Pilu_Comm, 25
StoreInHash
hash.c, 112
psparslib.h, 162
storeinhash
_p_DistMatrix::_p_Hash::_p_HashOps, 11
_p_DistMatrix::_p_Hash::_p_HashOps, 11
stripes
skitf.f, 200
stripes0
skitf.f, 200
swapj
armsheads.h, 65
swapm
armsheads.h, 65

tag
_p_Pilu_Comm, 25
time.c, 203
dwalltime, 203
tolind
_p_PrePar, 29
_p_PrePar, 29
tools.f, 204
expnddom, 204
getjamap, 204
getjamap1, 205
nod2dom, 206
TRUE
data.h, 80
type
_p_DistMatrix, 9
_p_DistMatrix, 9
_p_DistMatrixCsr, 15
_p_DistMatrixCsr, 15
TYPESIZE
data.h, 80

U
_p_Ext_Max, 16
_p_ILUfac, 17

ILUTfac, 37
PerMat4, 39
PerMat4, 39
uread.f, 207
userread, 207
userread
uread.f, 207
Usol
armsheads.h, 65
matrops.c, 130
Usolp
armsheads.h, 65
armsol2.c, 70

Vec
data.h, 81
vec
_p_Vec, 31
vec.c, 208
CreateVec, 208
DeleteVec, 208
PrintVec, 208
ResiNorm2, 209
vec_comops, 210
VecAssign, 209
VecNorm2, 209
VecSetFunc, 209
VecSetVal, 209
Vec_Comm
data.h, 81
vec_comm
_p_Vec, 31
Vec_ComOps
data.h, 81
vec_comops
_p_Vec::_p_Vec_Comm, 33
defs.h, 97
vec.c, 210
VecAssign
psparslib.h, 162
vec.c, 209
VecNorm2
psparslib.h, 163
vec.c, 209
VecRand
psparslib.h, 163
VecSetFunc
psparslib.h, 163
vec.c, 209
VecSetVal
psparslib.h, 163
vec.c, 209

weightsC
armsheads.h, 66
indsetC.c, 120
indsetC.c, 120
wk
ILUTfac, 37
PerMat4, 39
PerMat4, 39
wreadmtc
skitf.f, 200
wreadmtpar
dd-HB-parmetis.c, 95, 96
wreadmtpar.f, 211
wreadmtpar.f, 211
wreadmtpar, 211
wreadmtpar2
dd-HB-parmetis.c, 95, 96

X
_p_DistMatrix, 9
_p_DistMatrix, 9
_p_DistMatrixCsr, 15
_p_DistMatrixCsr, 15
xtrows
skitf.f, 201

ZERO
armsol2.c, 67
gprecsol.c, 106
psparslib.h, 146