PSPASES : Scalable Parallel Direct Solver Library for Sparse Symmetric Positive Definite Linear Systems*

User's Manual (version 1.0.3)

Mahesh Joshi, George Karypis, and Vipin Kumar
Department of Computer Science, University of Minnesota
Minneapolis, MN 55455.
{mjoshi,karypis,kumar}@cs.umn.edu

Anshul Gupta and Fred Gustavson
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598.
anshul@watson.ibm.com

Last updated on May 9, 1999 at 2:01pm

1 Introduction

PSPASES (Parallel SPArse Symmetric dirEct Solver) is a MPI-based parallel stand-alone library intended to solve a system of linear equations, $AX = B$, where A is a sparse symmetric positive definite matrix. PSPASES uses direct method of solution, which consists of four consecutive stages of processing: ordering, symbolic factorization, Cholesky factorization, and triangular systems solution. Each of these phases is implemented using scalable and high performance algorithms developed by the authors [1, 4, 2, 3].

PSPASES can be used on any parallel computer or network of workstations equipped with MPI and BLAS libraries, and Fortran-90 and C language compilers. It has been tested on IBM SP, SGI Power Challenge, SGI Origin 2000, Cray T3E, network of IBM RS6000 workstations, and network of Sun Solaris workstations.

A faster version with enhanced functionality for IBM RS6000 workstations and IBM SP parallel computers is available as WSSMP [5].

1.1 Organization of this Manual

This manual describes different functions provided by PSPASES, and explains how to use them. Section 2 describes the input and output formats of A, B, and X accepted by PSPASES functions.

*This work was supported by NSF CCR-9423082, by Army Research Office contract DA/DAAH04-95-1-0538, by Army High Performance Computing Research Center cooperative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008, by the IBM Partnership Award, and by the IBM SUR equipment grant. Access to computing facilities was provided by AHPCRC, Minnesota Supercomputer Institute.

We describe the functionality of PSPASES in Section 3, and Section 4 gives the calling sequences for different subroutines available in the current release. The contact information for obtaining the library and reporting the bugs can be found in Section 5.

2 Data distribution and Input/Output Formats

This section first describes the distribution of input matrices A and B, and the output solution matrix X. Then, the formats accepted by PSPASES for the local parts of the distributed data are elaborated.

It is assumed that a system of dimension N is solved using p processors. The algorithms used in PSPASES restrict p to be a power of two satisfying $p \geq 2$.

Figure 1 illustrates the formats for an example system being solved on 4 processors.

2.1 Data Distribution

The input matrix A is assumed to be distributed among p processors using a horizontal striped partitioning. Figure 1(a) shows an example. Let A_i denote the local part of A residing on p_i.

The input matrix B is also assumed to be distributed among processors using a horizontal striped partitioning. Partitioning of B need not be the same as that of A. At output, the solution matrix, X, will be distributed in a manner identical to B. Figure 1(a) shows an example. Let B_i and X_i denote the local parts of B and X, respectively, residing on processor p_i.

2.2 Format for A

The distributed input matrix A is described using following parameters.

- $rowdista$: This is a vector of size $(p+1)$. A processor, p_i, has $ma = rowdista(p_i+1) - rowdista(p_i)$ rows starting from the $rowdista(p_i)^{th}$ row. The rows are numbered from 0, hence, $rowdista(p+1) = N$. This vector must be identical on all the processors.
- A_i : PSPASES requires the matrix A_i to be stored using three arrays: $aptrs$, $ainds$, and $avals$.
  - $ainds$: This is a single dimensional array, storing column indices for the non-zeros in the rows in A_i. Although the column indices for a given row must be stored in a contiguous manner, PSPASES format does not require the indices for two consecutive rows to be stored contiguously. Also, for correct results, PSPASES internally needs the column indices of a given row to have a sorted order. If the user is not sure of the sorted order, PSPASES must be instructed to do the sorting by specifying a particular option in the subroutines that take ainds as their input (see section 4).
  - $avals$: This is a single dimensional array, storing the values for the non-zeros in A_i appearing at the column indices stored in $ainds$. Value stored at $avals(k)$ must correspond to the column index stored at $ainds(k)$.
  - $aptrs$: This is a two dimensional array of dimensions $(2, ma)$. For a local row j of A_i, $aptrs(2,j)$ gives the number of non-zeros in row j, whereas $aptrs(1,j)$ gives the pointer into the arrays $ainds$ and $avals$, from where the column indices and their corresponding

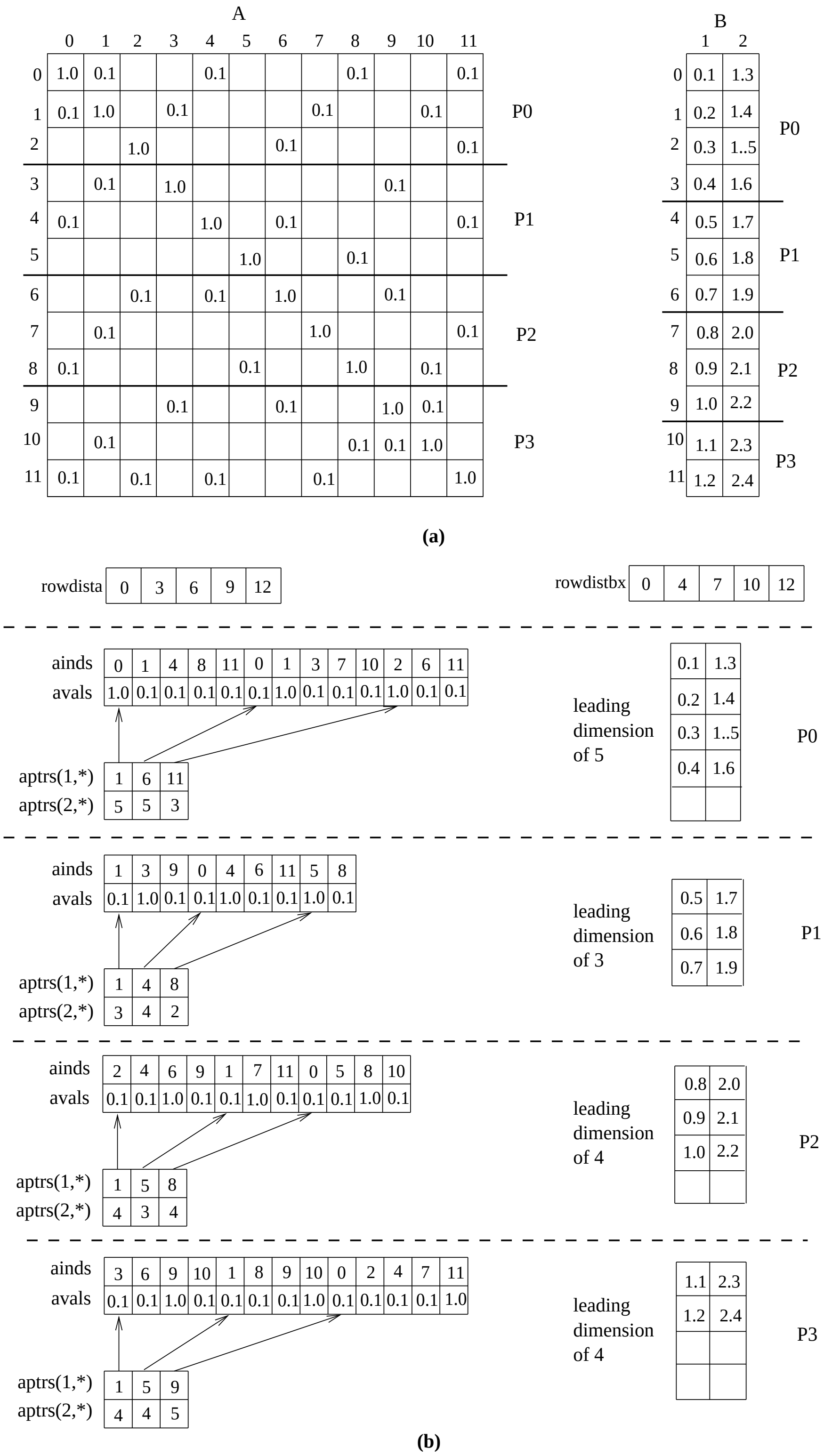

Figure 1: Input/Output Formats accepted by PSPASES. (a) Example Matrices, A and B. (b) Input Formats for Distributed A and Distributed B.

values are stored for these non-zeros. The pointers, $aptrs(1,i)$, must be stored assuming that the array indices for $ainds$ and $avals$ begin from 1 (Fortran style).

Figure 1(b) shows $rowdista$, and A_i on each of the processors for the example matrix A shown in part (a) of the figure.

2.3 Format for B and X

The linear system can be solved for multiple right hand sides at the same time. Let there be $nrhs$ number of right hand sides. B and X are described using following parameters.

- $rowdistbx$: This is a vector of size $(p+1)$. A processor p_i has, $mbx = rowdistbx(p_i + 1) - rowdistbx(p_i)$, rows of B and X starting from the $rowdistbx(p_i)^{th}$ row. The rows are numbered from 0, hence, $rowdistbx(p+1) = N$. This vector must be identical on all the processors.

- B_i : At the input, the local part B_i, residing on processor p_i, is stored as an array of dimensions ldb x $nrhs$, where ldb is the leading dimension satisfying the condition $ldb \geq mbx$.

- X_i : At the output, the local part of the solution, X_i, residing on processor p_i, will be stored in an array of dimensions ldx x $nrhs$, where ldx is the leading dimension satisfying the condition $ldx \geq mbx$. Sufficient storage space ($\geq ldx$ x $nrhs$) is assumed to be provided by the user.

Figure 1(b) shows B_i on each of the processors for the example right-hand-side matrix B shown in part (a) of the figure.

2.4 Format of Supplying Ordering Information

PSPASES allows advanced users to input a permutation order on rows of an unordered matrix. PSPASES would then reorder this matrix to realize the specified permutation. However, the user must be familiar with the algorithms of PSPASES in order to provide such ordering information. Refer to [4] for the concept of elimination trees and their abstraction to supernodal trees used in PSPASES algorithms. The ordering provided by the user must yield a supernodal tree that is binary in the top $log\ p$ levels, when the problem is being solved on p processors. The reordered nodes must be numbered by a preorder traversal of this supernodal tree. Figure 2 shows an example supernodal tree and the numbering of its nodes.

The ordering information which satisfies above-stated PSPASES constraints, is fed to PSPASES in the following format.

The global permutation vector is denoted by $order$. A row j in the unordered matrix goes to a new row number given by $order(j)$ in the reordered matrix. The $order$ vector must be distributed among p processors using the same striped mapping that is used for distributing rows of A. Let $order_i$ denote the local part of $order$ residing on processor p_i. PSPASES needs the ordering information in the form of following two parameters.

- $order_i$: This is a vector of size $ma = rowdista(p_i + 1) - rowdista(p_i)$. It has ma elements of global permutation vector, $order$, starting from the $rowdista(p_i)^{th}$ element.

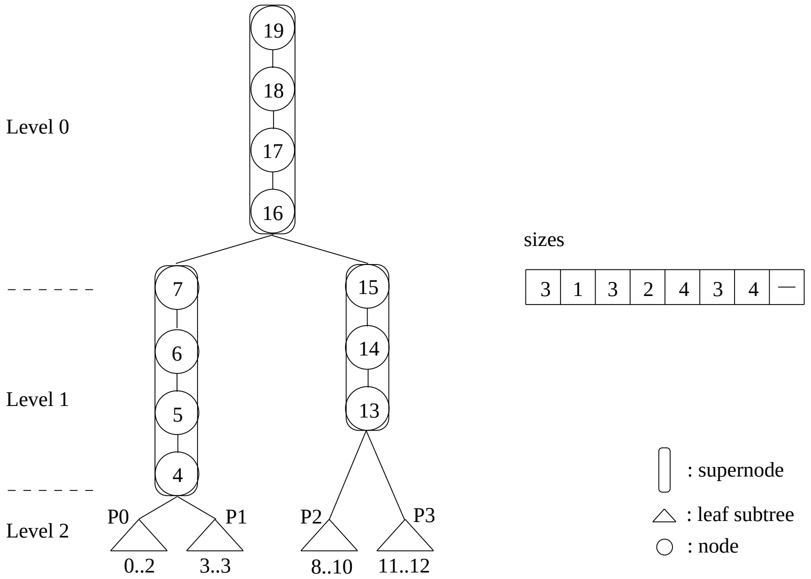

Figure 2: A supernodal (elimination) tree of the reordered matrix as required by PSPASES. The nodes in the reordered matrix are numbered in a preorder fashion (even at the leaf subtrees). The $sizes$ array is shown.

- $sizes$: This is a vector of size $2p$. First p elements, $sizes(1)$ through $sizes(p)$ give the sizes of the leaf subtrees at $log\ p^{th}$ level in the supernodal tree of the ordered matrix. The next $p-1$ elements in $sizes$ are the sizes of the supernodes in the top $log\ p$ levels of the tree, first $p/2$ for the supernodes at level $log\ p-1$, next $p/4$ for supernodes at level $log\ p-2$, and so on. The size for the root supernode, at level 0, is given in $sizes(2p-1)$. The sizes of supernodes and leaf subtrees at any given level must appear in the $sizes$ array in a left to right order. Figure 2 shows the sizes vector for the example supernodal tree.

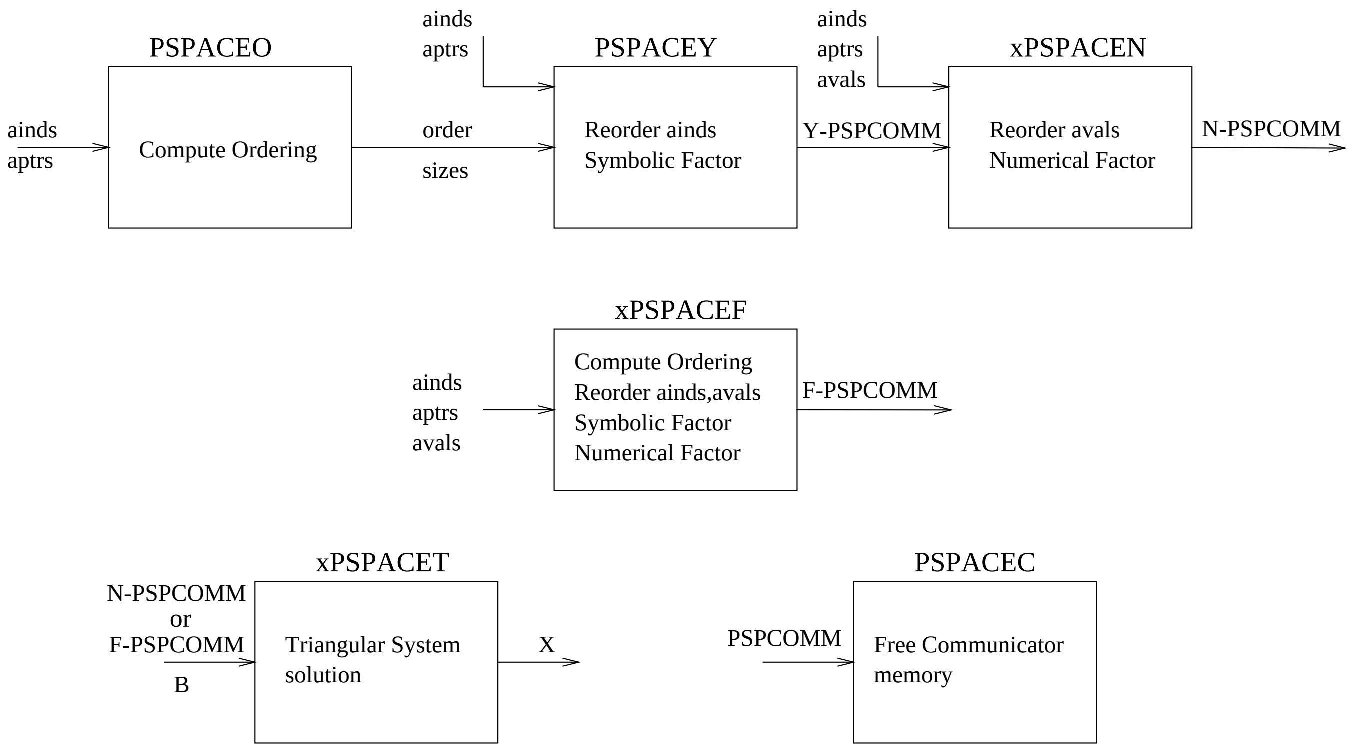

Figure 3: PSPASES Functionality.

3 PSPASES Functionality

We first describe the subroutine naming convention used in PSPASES. Then, we describe functionality and corresponding subroutines provided by PSPASES. Finally, we explain the concept of a communicator used by PSPASES to communicate data across different subroutines.

- **Subroutine Naming Convention**

  A subroutine is named xPSPAyEz. x indicates, wherever applicable, the precision and data type of elements in A, B, and X : **S** for single precision real, **D** for double precision real, **C** for single precision complex, and **Z** for double precision complex. y indicates the type of factorization used: **C** for Cholesky (LL^T) and **D** for LDL^T. *Current release of PSPASES supports only the double precision real type (x=***D***), and Cholesky factorization (y=***C***). Hence, currently available subroutines are of the form* ***D****PSPA****C****Ez or PSPA****C****Ez.* z indicates the functionality as described next.

- **Subroutines and their Functionality**

  Refer to Figure 3 for a flowchart of PSPASES functionality. Following is the description of each of the subroutines.

  - PSPACEO (**O**rdering):

    Computes a fill reducing ordering using ParMETIS [1], given the nonzero structure of A. This is the first phase of the direct solution method.

– PSPACEY (S**Y**mbolic factorization):

  Computes the nonzero structure of L, given the ordering. This is the second stage of the direct solution method.

– xPSPACEN (**N**umerical factorization):

  Performs the numerical Cholesky factorization; i.e., given the values of A and the nonzero structure of L as computed by PSPACEY, it fills in the values in L. This is the third stage of the direct solution method.

– xPSPACEF (**F**actor):

  Computes fill reducing ordering using ParMETIS, reorders A, and factors A both symbolically and numerically. This encapsulates the first three phases of a direct solution method.

– xPSPACET (**T**riangular Systems Solution):

  Permutes B to B', solves the triangular systems, first $LW = B'$, followed by $L^T X' = W$, and reorders X' back to original order to get the solution, X. This is the final stage of a direct solution method.

– PSPACEC (**C**leanup routine):

  Cleans up the internal memory required to be stored across different PSPASES subroutine calls.

- **Memory Management using PSPASES Communicator**

  PSPASES uses a communicator to communicate the required data across different subroutines. The communicator allows a user to choose different paths and entry points as shown in Figure 3. In particular, a user can do the following.

  – A sparse nonzero structure can be symbolically factored before it can be used to do numerical factorization for multiple matrices having different values with the same nonzero structure.

  – Many matrices can be factored before their respective factors can be used to solve different right-hand-side matrices, B.

  – A given matrix can be solved repeatedly for different right-hand-side matrices. (This can be done with a single call to xPSPACET, too.)

  – The memory space provided for some of the matrices can be retrieved from the system by making a call to PSPACEC to free the memory associated with its communicator.

  There are three types of communicators in PSPASES.

  – **Y-communicator**:

    This communicator is generated by a PSPACEY call. It can be used in multiple calls to xPSPACEN to generate multiple numerical factors. Each call to xPSPACEN takes as input a Y-communicator, and produces an N-communicator at output. Each such N-communicator is a child of the Y-communicator supplied to the xPSPACEN call.

    Y-communicator stores the data needed for performing numerical factorization as well as data that can be passed on to a xPSPACET call via N-communicator. The memory needed specifically for generating new numerical factors can be recovered using option

1 of PSPACEC, when one decides that there will not be any more numerical factorizations performed with this Y-communicator. The entire memory associated with a Y-communicator can be freed (using option 0 of PSPACEC) only when all its child N-communicators are freed up. This condition is imposed to allow users to keep track of how many numerical factors are present in the memory for a given symbolic factorization.

- **N-communicator**:

  This communicator is generated by a xPSPACEN call. It can be used in multiple calls to xPSPACET to solve for multiple B matrices (right hand sides). Each N-communicator has a unique parent Y-communicator, who stores the memory required for xPSPACET calls. The numerical factor memory, which is unique to each N-communicator, can be freed using a call to PSPACEC with cleanup option 0.

- **F-communicator**:

  This communicator is generated by a xPSPACEF call. It can be used in multiple calls to xPSPACET to solve for multiple B matrices (right hand sides). Unlike N-communicator, F-communicator does not have any parent communicator. If same matrix A is numerically factored using calls to xPSPACEF as well as PSPACEY and xPSPACEN, then the memory space of F-communicator is disjoint from that of Y- and N-communicators. So, it is advised to choose only one of the two paths for numerical factorization. Entire memory allocated for a F-communicator can be recovered by calling PSPACEC with cleanup option 0.

An *open communicator* is a Y-, N-, or F-communicator that has not yet been freed with PSPACEC by using cleanup option 0. *Currently, the PSPASES limits the maximum number of open communicators at any given time to 256. This limit can be changed by changing the MAX_OPEN_PSPCOMMS constant in pspaces.h in the PSPASES directory of the distribution, and recompiling the library.*

4 Calling Sequences

This section gives the calling sequences for the subroutines described in section 3.

We assign two attributes to each of the parameters in a calling sequence. This attribute pair is denoted by (a,b), where a indicates the type of the parameter: I for IN, and O for OUT, and M for INOUT; and b indicates the significance of the parameter in the parallel context: L means local significance and G means global significance. The parameters with local significance can have different values on different processors, whereas the parameters with global significance must have identical values on all the processors.

For each subroutine, both C and FORTRAN bindings are given. The parameters are described in the C binding and the array sizes, wherever applicable, are shown in the FORTRAN binding.

PSPASES needs all the required two-dimensional arrays to be stored in a column-major order in the memory. The user is responsible for ensuring this order when the arrays are declared in C, which by default, uses a row-major storage order.

- **PSPACEO**: Computes Fill-reducing Ordering.

  C Binding :

  PSPACEO (int *rowdista, int *aptrs, int *ainds, int *order, int *sizes,
  int *ioptions, MPI_Comm *mpicomm);

rowdista	(I,G)		As described in section 2.2.
aptrs	(I,L)		As described in section 2.2. Note that, aptrs must be stored in a column-major order.
ainds	(I,L)		As described in section 2.2.
order	(O,L)		As described in section 2.4. Sufficient storage space ($\geq$ ma) should be provided by the user.
sizes	(O,G)		As described in section 2.4. Sufficient storage space ($\geq$ 2*p) should be provided by the user.
ioptions	(I,G)		An array of size 16.
	ioptions[3]	(I,G)	if 1, does sequential ordering using METIS. Default is to do parallel ordering using ParMETIS.
mpicomm	(I,G)		Pointer to MPI Communicator (The size of the communicator must be $2^i, i \geq 1$).

  FORTRAN Binding :

  CALL PSPACEO (rowdista,aptrs,ainds,order,sizes,ioptions,mpicomm)

  INTEGER rowdista(p+1), aptrs(2,ma), ainds(*)
  INTEGER order(ma), sizes(2*p), ioptions(16), mpicomm

- **PSPACEY**: Computes Symbolic Cholesky Factorization.

  C Binding :

```
PSPACEY (int *rowdista, int *aptrs, int *ainds, int *order, int *sizes,
         int *ioptions, double *doptions, long *pspcommy,
         MPI_Comm *mpicomm);
```

rowdista	(I,G)	As described in section 2.2.
aptrs	(I,L)	As described in section 2.2. Note that, aptrs must be stored in a column-major order.
ainds	(I,L)	As described in section 2.2.
order	(I,L)	As described in section 2.4.
sizes	(I,G)	As described in section 2.4.
ioptions	(I,G)	An array of size 16.
	ioptions[0] (M,G)	block size for distributing A internally. Refer to [2] for details. (default = 64)
	ioptions[1] (I,G)	if 1, checks the symmetry of the non-zeros of A. (default = 0)
	ioptions[2] (I,G)	if 1, sorts the column indices, stored in ainds, for every row of A. Must be set to 1, if not sure of the sorted order (default = 0)
	ioptions[3-15]	Unused.
doptions	(O,L)	An array of size 16.
	doptions[0] (O,L)	Memory used in PSPASES Communicator (in bytes) for the calling process. This is already allocated.
	doptions[1] (O,L)	Estimated extra memory requirement (in bytes), for the calling process, to complete numerical factorization. Add this to doptions[0] to get the total Memory Requirement.
	doptions[2] (O,L)	Number of non-zeros in the triangular part of global A. (Relevant only on processor 0).
	doptions[3] (O,L)	Number of non-zeros in global L. (Relevant only on processor 0).
	doptions[4] (O,L)	Opcount for sequential factorization. (Relevant only on processor 0).
	doptions[5] (O,L)	Computational load imbalance caused by imbalance in the supernodal tree. A value of 1.0 indicates perfect balance. (Relevant only on processor 0).

	doptions[6-15]	Unused.
pspcommy (O,G)		Pointer to PSPASES Communicator. This will be a Y-communicator (refer to section 3).
mpicomm (I,G)		Pointer to MPI Communicator (The size of the communicator must be $2^i, i \geq 1$).

FORTRAN Binding :

```
CALL PSPACEY      (rowdista,aptrs,ainds,order,sizes,ioptions,doptions,
                   pspcommy,mpicomm)

INTEGER             rowdista(p+1), aptrs(2,ma), ainds(*), order(ma)
INTEGER             sizes(2*p), ioptions(16), mpicomm
INTEGER*8           pspcommy
DOUBLE PRECISION    doptions(16)
```

- **DPSPACEN**: Computes Numerical Cholesky Factorization.

  C Binding :

```
DPSPACEN (int *rowdista, int *aptrs, int *ainds, double *avals,
          long *pspcommy, long *pspcommn, MPI_Comm *mpicomm);
```

rowdista	(I,G)	As described in section 2.2.
aptrs	(I,L)	Must be identical to aptrs supplied to a corresponding PSPACEY call.
ainds	(I,L)	Must be identical to aptrs supplied to a corresponding PSPACEY call.
avals	(I,L)	As described in section 2.2. Note that, the value stored at avals[k] must correspond to the nonzero index stored at ainds[k].
pspcommy	(I,G)	Pointer to PSPASES Communicator. This has to be a Y-communicator which has not been freed with a call to PSPACEC with option 1 (refer to section 3).
pspcommn	(O,G)	Pointer to PSPASES Communicator. This will be a N-communicator (refer to section 3).
mpicomm	(I,G)	Pointer to MPI Communicator (The size of the communicator must be $2^i, i \geq 1$).

  FORTRAN Binding :

```
CALL DPSPACEN (rowdista,aptrs,ainds,avals,pspcommy,pspcommn,mpicomm)

INTEGER            rowdista(p+1), aptrs(2,ma), ainds(*), mpicomm
INTEGER*8          pspcommy, pspcommn
DOUBLE PRECISION   avals(*)
```

- **DPSPACEF**: Encapsulates PSPACEO, PSPACEY, and DPSPACEN.

  C Binding :

```
DPSPACEF (int *rowdista, int *aptrs, int *ainds, double *avals,
          int *ioptions, double *doptions, long *pspcommf,
          MPI_Comm *mpicomm);
```

rowdista	(I,G)		As described in section 2.2.
aptrs	(I,L)		As described in section 2.2. Note that, aptrs must be stored in a column-major order.
ainds	(I,L)		As described in section 2.2.
avals	(I,L)		As described in section 2.2. Note that, the value stored at avals[k] must correspond to the nonzero index stored at ainds[k].
ioptions	(I,G)		Same as described in PSPACEY, plus
	ioptions[3]	(I,G)	if 1, does sequential ordering using METIS. Default is to do parallel ordering using ParMETIS.
doptions	(O,L)		Same as described in PSPACEY.
pspcommf	(O,G)		Pointer to PSPASES Communicator. This will be a F-communicator (refer to section 3).
mpicomm	(I,G)		Pointer to MPI Communicator (The size of the communicator must be $2^i, i \geq 1$).

FORTRAN Binding :

```
CALL DPSPACEF    (rowdista,aptrs,ainds,avals,ioptions,doptions,
                 pspcommf,mpicomm)

INTEGER              rowdista(p+1), aptrs(2,ma), ainds(*), ioptions(16), mpicomm
INTEGER*8            pspcommf
DOUBLE PRECISION     doptions(16), avals(*)
```

- **DPSPACET**: Computes Triangular Systems Solution.

  C Binding :

  DPSPACET (int *rowdistbx, int *pnrhs, double *b, int *pldb, double *x,
  int *pldx, int *ioptions, long *pspcomm, MPI_Comm *mpicomm);

rowdistbx	(I,G)	As described in section 2.3.
pnrhs	(I,G)	Pointer to a location storing the number of right hand sides (nrhs).
b	(I,L)	As described in section 2.3. Note that, b must be stored in a column-major order. So, an example C declaration would declare b as b[nrhs][ldb].
pldb	(I,L)	Pointer to a location storing the leading dimension of b (ldb). Note that, ldb $\geq$ mbx (section 2.3)
x	(O,L)	As described in section 2.3. Note that, at output, x will be stored in a column-major order. So, for correct access to the solution, an example C declaration would declare x as x[nrhs][ldx]. User must provide sufficient storage space ($\geq$ ldx x nrhs).
pldx	(I,L)	Pointer to a location storing the leading dimension of x (ldx). Note that, ldx $\geq$ mbx (section 2.3).
ioptions	(I,G)	An array of size 8.
	ioptions[0] (I,G)	blocking factor for nrhs, br. The system will be solved in blocks of br right-hand-sides at a time. If $br <$ nrhs, then less memory is required internally. (default = nrhs)
	ioptions[1-7]	Unused.
pspcomm	(I,G)	Pointer to PSPASES Communicator. This has to be either N- or F-communicator (refer to section 3).
mpicomm	(I,G)	Pointer to MPI Communicator (The size of the communicator must be $2^i, i \geq 1$).

  FORTRAN Binding :

  CALL DPSPACET (rowdistbx,nrhs,b,ldb,x,ldx,ioptions,pspcomm,mpicomm)

INTEGER	rowdistbx(p+1), nrhs, ldb, ldx, ioptions(8), mpicomm
INTEGER*8	pspcomm
DOUBLE PRECISION	b(ldb,nrhs), x(ldx,nrhs)

- **PSPACEC**: Cleans up PSPASES Communicator memory.

  C Binding :

  PSPACEC (long *pspcomm, int *poption);

poption	(I,G)	Pointer to a location storing the option value. If option value is 1, then frees the extra memory stored in PSPASES Y-communicator, that is necessary for making multiple calls to DPSPACEN for a given symbolic factorization. After using this option, the Y-communicator will not be able to produce any more numerical factors If option value is 0, then frees all the memory that is specific to the PSPASES communicator supplied in pspcomm argument, and deletes (or closes) the communicator. Refer to section 3 to learn about different communicator types.
pspcomm	(M,G)	Pointer to PSPASES Communicator. This has to be a Y-communicator when cleanup option is 1. (section 3)

  FORTRAN Binding :

```
CALL PSPACEC    (pspcomm,option)

INTEGER             option
INTEGER*8           pspcomm
```

5 Obtaining and Installing PSPASES and Related Libraries

PSPASES can be used on any parallel computer or network of workstations equipped with MPI and BLAS libraries, and Fortran-90 and C language compilers. Native implementations of BLAS and MPI, tuned to the underlying architecture, help in improving the performance obtained out of PSPASES. There is one LAPACK-compatible call that PSPASES uses. This call, xPOTRF, is required for serial Cholesky factorization. Public-domain version of xPOTRF is supplied with the distribution.

PSPASES has been tested on IBM SP, SGI Power Challenge, SGI Origin 2000, Cray T3E, network of IBM RS6000 workstations, and network of Sun Solaris workstations.

The software library is available at the URL:

http://www.cs.umn.edu/˜mjoshi/pspases

This user's manual is provided with the distribution. Installation instructions are given in README.INSTALL file provided with the distribution. Some notes answering commonly asked questions are in README.USAGE. Any comments, questions, or bugs may be directed to Mahesh Joshi (mjoshi@cs.umn.edu).

PSPASES uses ParMETIS and METIS as the default ordering libraries. The current available version of these libraries is supplied with PSPASES distribution. The latest versions can be obtained from the METIS web-site : http://www.cs.umn.edu/˜metis.

A faster version of PSPASES, with enhanced functionalities, is available for IBM RS6000 workstations and IBM SP parallel computers as the WSSMP [5] library. This library can solve positive definite as well as indefinite systems. For further information, refer to URL:

http://www.research.ibm.com/mathsci/ams/ams_WSSMP.htm

The public-domain implementations of BLAS and MPI libraries, LAPACK and MPICH, respectively, are available from various sources on the World Wide Web.

6 Notice: Terms and Conditions for use of PSPASES

This code is meant to be used solely for educational, research, and benchmarking purposes by non-profit institutions and US government agencies only. Use by any other organization requires prior written permission from both IBM Corporation and the University of Minnesota. The software may not be sold or redistributed. One may make copies of the software or modify it for their use provided that the copies, modified or otherwise, are not sold or distributed, are used under the same terms and conditions, and this notice and any part of the source code that follows this notice are not separated.

As unestablished research software, this code is provided on an "as is" basis without warranty of any kind, either expressed or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose. IBM does not warrant that the functions contained in this software will meet the user's requirements or that the operation of its routines will be uninterrupted or error-free. Acceptance and use of this program constitutes the user's understanding that he/she will have no recourse to IBM for any actual or consequential damages, including, but not limited to, lost profits or savings, arising out of the use or inability to use these libraries. Even if the user informs IBM of the possibility of such damages, IBM expects the user to accept the risk of any such harm, or the user shall not attempt to use these libraries for any purpose.

The downloading, compiling, or executing any part of this software constitutes an implicit agreement to these terms. These terms and conditions are subject to change at any time without prior notice.

ParMETIS and METIS, which are supplied with the PSPASES package as the default ordering libraries, are solely owned by the University of Minnesota and the above notice does not apply to them.

References

[1] George Karypis and Vipin Kumar, *ParMETIS: Parallel Graph Partitioning Library, Version 1.0*, Available via URL : http://www.cs.umn.edu/~metis, July 1997.

[2] Anshul Gupta, George Karypis, and Vipin Kumar, *Highly Scalable Parallel Algorithms for Sparse Matrix Factorizations*, IEEE Transactions on Parallel and Distributed Systems, vol.8, no.5, 1995.

[3] Mahesh Joshi, Anshul Gupta, George Karypis, and Vipin Kumar, *A High Performance Two Dimensional Scalable Parallel Algorithm for Solving Sparse Triangular Systems*, Proc. 1997 International Conference on High Performance Computing (HiPC'97), Bangalore, India, 1997.

[4] Anshul Gupta, Fred G. Gustavson, Mahesh Joshi, George Karypis, and Vipin Kumar,*Design and Implementation of a Scalable Parallel Direct Solver for Sparse Symmetric Positive Definite Systems: Preliminary Results*, Proc. Eighth SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis, March 1997.

[5] Anshul Gupta, Mahesh Joshi, and Vipin Kumar, WSSMP: Watson Symmetric Sparse Matrix Package, IBM Research Report RC 20923 (92669), 1997. (Also available at ftp://ftp.cs.umn.edu//users/kumar/anshul//WSSMP-manual.ps)