A User's Guide to PSSPD

Chunguang Sun *

September 26, 1996

1 Introduction

PSSPD(Parallel Solution of Sparse Symmetric Positive Definite Systems) is a sparse matrix package for solving large, sparse, symmetric positive systems on distributed-memory machines. This package is implemented in ANSI C using double-precision arithmetic. The Message-Passing Interface(MPI) is used for communication among processors.

PSSPD is a parallel direct sparse solver. It solves a sparse symmetric positive system $Ax = b$ by computing the sparse Cholesky factorization $PAP^T = LL^T$, where P is a permutation matrix. PSSPD is designed to be run on all processors in a communicator `comm` in parallel. The sparse matrix A and the right hand side vector b are distributed among the processors in `comm`. However, PSSPD doesn't assume any particular data-to-processor distribution of matrix A or vector b among the processors. The overall architecture of PSSPD is illustrated in Fig. 1.

1. **Matrix structure input.** Provide the sparsity pattern of A to PSSPD.
2. **Symbolic computation.** Symbolic computation consists of symmetric ordering, symbolic factorization and task allocation. A permutation matrix P is determined so that PAP^T has a sparse Cholesky factor L. The symbolic structure of L is then obtained. The Cholesky factorization of PAP^T is partitioned into a collection of tasks according to the symbolic structure of L; the tasks are allocated to the processors in `comm` by a task-to-processor mapping scheme proposed in [3].
3. **Matrix value input.** Provide the numerical values of the nonzero entries of A to PSSPD.
4. **Numerical factorization.** Compute the numerical values of the nonzero entries of L.
5. **Right hand side input.** Provide the right hand side vector b to PSSPD.
6. **Triangular solution.** Solve the sparse triangular systems $Lu = Pb$ and $L^T v = u$. The vector $x = P^T v$ is the desired solution to $Ax = b$.

*Advanced Computing Research Institute, Cornell Theory Center, Cornell University, Ithaca, NY 14853-3801 (csun@cs.cornell.edu).

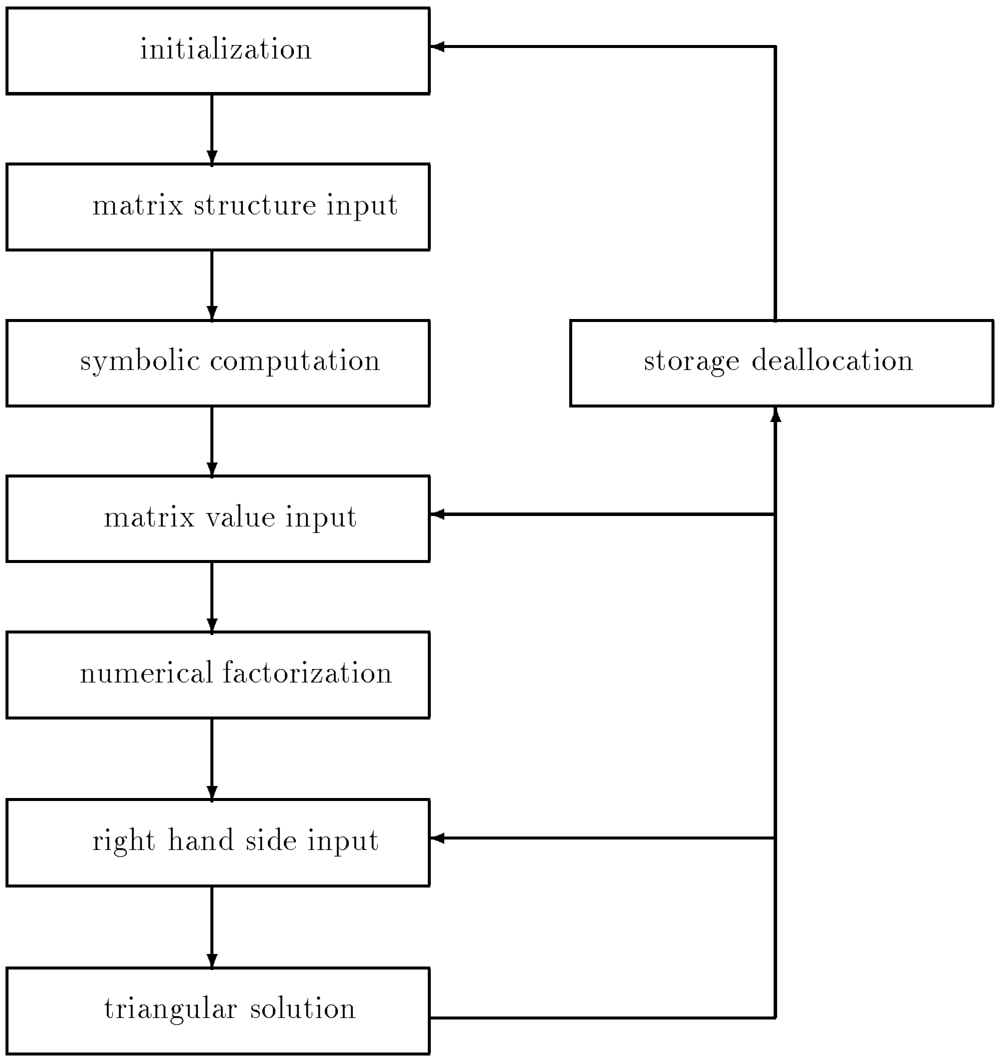

Figure 1: The overall architecture of PSSPD

All components of PSSPD except the symbolic computation are performed in parallel. The symbolic computation is not parallelized for the present version of PSSPD. The numerical factorization and the triangular solution are carried out by a parallel multifrontal scheme described in [4]. Parallel multifrontal sparse triangular solution is also discussed in [5].

A system with multiple right hand sides is solved by executing the symbolic computation and the numerical factorization once. For each right hand side, the corresponding sparse triangular systems are solved. This process is repeated until all right hand sides are processed.

PSSPD can be conveniently used for solving systems with the same sparsity structure but different sets of numerical values. The symbolic computation is executed only once. For each set of numerical values, the numerical factorization is performed, and it is followed by a number of triangular solutions.

2 User Interface

The user interface of PSSPD consists of a set of simple and flexible user interface functions described in this section. Every user interface function returns an integer error code. A zero error code indicates that the function has been successfully executed. A nonzero error code indicates that the execution of the function has failed. Interpretation of a nonzero error code is given in Table 1 in section 4.

The last formal parameter of every user interface function except the function `PsspdInitialize` is of type `void *`. This formal parameter is denoted by `data`. A variable of type `void *` (e.g., `void *user_data`) must be declared in a user program which uses PSSPD for solving sparse linear systems in parallel. To call a user interface function, the actual argument corresponding to the formal parameter `data` must be this user-declared variable `user_data`.

Every processor in `comm` is allocated a portion of the matrix A and a portion of the vector b. Processors provide their local portions of A and b to the package PSSPD in parallel. Each and every processor in `comm` must call the user interface functions in the order shown in Fig. 1. **All array indices start from zero**.

2.1 Initialization

The function `PsspdInitialize` must be called first for solving a sparse linear system. The first parameter `comm` is the underlying communicator on which the package PSSPD will be running. All processors in `comm` participate in the parallel solution of the sparse linear system. The second parameter `n` is the number of equations in the sparse linear system. The third parameter `nnz` must be greater than or equal to the number of nonzero entries in the lower triangle of A including diagonal elements which are allocated to the calling processor. The fourth parameter `psspd_data` corresponds to the address of the actual argument `user_data`(i.e. `&user_data`) described above.

```
int PsspdInitialize(MPI_Comm comm, int n, int nnz, void **psspd_data)
```

2.2 Matrix Structure Input

For an entry $A(i, j)$, the pair of integers (i, j) is referred to as an *index pair*. A processor needs only to supply to the package the index pairs of the nonzero entries in the lower triangle of A which are allocated to the processor.

The function `PsspdStructInputBegin` must be called to signal the start of the matrix structure input before any index pair is supplied to the package. It must be called by every processor even if no index pair is provided to the package on a particular processor. Similarly, the function `PsspdStructInputEnd` must be called by every processor to signal the end of the matrix structure input. The index pairs may be provided to the package in any order.

2.2.1 `int PsspdStructInputBegin(void *data)`

2.2.2 `int PsspdInputIJ(int i, int j, void *data)`

This function tells the package that the `(i,j)` location of A is nonzero.

2.2.3 `int PsspdStructInputEnd(void *data)`

2.3 Symbolic Computation

If `perm == NULL`, default symmetric ordering is applied to A. The well-known minimum degree ordering [2] is used as the default symmetric ordering. If `perm != NULL`, `perm` is an `int` vector containing a permutation of $0, 1, \cdots, n-1$ provided by the user, and the symmetric ordering corresponding to the user-supplied permutation in `perm` is applied to A.

```
int PsspdSymbolicFactor(int *perm, void *data)
```

2.4 Matrix Value Input

A processor needs only to supply to the package the numerical values of the nonzero entries in the lower triangle of A including diagonal elements which are allocated to the processor.

The function `PsspdValueInputBegin` must be called to signal the start of the matrix value input before any numerical value is supplied to the package. It must be called by every processor even if no numerical value is provided to the package on a particular processor. Similarly, the function `PsspdValueInputEnd` must be called by every processor to signal the end of the matrix value input. The numerical values may be provided to the package in any order.

2.4.1 `int PsspdValueInputBegin(void *data)`

2.4.2 `int PsspdInputAIJ(int i, int j, double aij, void *data)`

This function provides the numerical value `aij` at the `(i,j)` location of A to PSSPD.

2.4.3 `int PsspdValueInputEnd(void *data)`

2.5 Numerical Factorization

```
int PsspdNumericFactor(void *data)
```

2.6 Right Hand Side Input

A processor needs only to supply to the package the numerical values of a right hand side vector which are allocated to the processor.

The function `PsspdRhsInputBegin` must be called to signal the start of a right hand side input before any numerical value of the right hand side is supplied to the package. It must be called by every processor even if no numerical value of the right hand side is provided to the package on a particular processor. Similarly, the function `PsspdRhsInputEnd` must be called by every processor to signal the end of the right hand side input. The numerical values of the right hand side vector may be provided to the package in any order. If the numerical value of the i^{th} component of the right hand side vector is not provided to the package, it is assumed to be zero.

2.6.1 `int PsspdRhsInputBegin(void *data)`

2.6.2 `int PsspdInputBI(int i, double bi, void *data)`

This function inputs the numerical value `bi` at location `i` of the right hand side vector.

2.6.3 `int PsspdRhsInputEnd(void *data)`

2.7 Triangular Solution

This function computes the solution corresponding to the right hand side vector provided in the preceding right hand side input. The parameter `solution` must be a vector of size at least `n`, and it must be provided by the calling function. When this function returns, `solution` contains the solution vector on each and every processor in the communicator `comm`.

```
int PsspdTriSolve(double *solution, void *data)
```

2.8 Storage Deallocation

This function deallocates memory space used for solving a sparse linear system.

```
int PsspdFinalize(void *data)
```

3 Pivot Modification

PSSPD is designed for solving sparse symmetric positive definite systems. However, it can be used to handle sparse symmetric positive semidefinite systems to some extent. To solve a sparse symmetric positive semidefinite system, simply change the file `libpsspd/prob.h` to the following:

```
#define SPD_PROB
#undef  SPD_PROB

#define LP_PROB
```

Then recompile the code. The pivot modification strategy proposed in [1] is implemented in PSSPD. A pivot is considered small if

$$A_{ii}^{(i-1)} \leq 10^{-30} \max_{j=1,2,\cdots,n} A_{jj},$$

where $A^{(i-1)}$ is the active submatrix after $i-1$ steps of the Cholesky factorization on the matrix A. A small pivot is replaced by a very large number 10^{128}.

4 Interpretation of Error Code

20	`n < 1`
21	`nnz < 0`
22	invalid index pair during matrix structure input
23	`nnz` is too small for matrix structure input
24	invalid permutation vector is given
25	invalid elimination tree encountered in symbolic factorization
26	error in symbolic factorization
27	invalid entry in matrix value input
28	`nnz` is too small for matrix value input
29	inconsistent entry in matrix value input
30	invalid entry in right hand side input
31	very small pivot encountered in numerical factorization

Table 1: Interpretation of the error codes.

5 Computational Results

In the following tables, N is the number of equation, $|A|$ is the number of nonzero entries in the lower triangle of A including diagonal elements, $|L|$ is the number of nonzero entries in the sparse Cholesky factor L, and p is the number of processors used. 'Symbolic' denotes the running time for symbolic computation. 'Numeric' denotes the running time for numerical factorization. All running times are in second. 'Fact_Flops' is the total number of mega flops performed in the numerical factorization. 'MFLOPS' is the number of mega flops performed per second in the numerical factorization. 'Speedup' denotes the speedup achieved for the numerical factorization.

A test problem $k \times k$ denotes a k-by-k two-dimensional 9-point grid. A test problem $k \times k \times k$ denotes a three-dimensional 27-point grid. BCSSTK25 and BCSSTK33 are taken from the Harwell-Boeing collection of sparse matrices. The grid problems are ordered by nested-dissection ordering which is supplied through `perm` in `PsspdSymbolicFactor` and is not a part of the symbolic computation. The Harwell-Boeing problems are ordered by the default ordering–i.e., the minimum degree ordering. All performance results are obtained on an IBM Scalable POWERparallel System SP2 at Cornell Theory Center. The compilation optimization level `-O2` is used.

Table 2: Test Problems

Problem	N	$\|A\|$	$\|L\|$	Fact_Flops
300×300	90,000	448,202	3,734,104	517
400×400	160,000	797,602	7,121,429	1233
$40 \times 40 \times 40$	64,000	853,516	24,626,969	26106
$50 \times 50 \times 50$	125,000	1,683,396	62,819,416	102106
BCSSTK25	15,439	133,840	1,416,568	292
BCSSTK33	8,738	300,321	2,546,802	1214

References

[1] J. Czyzyk, S. Mehrotra, and S. J. Wright, *PCx user guide*, Tech. Report OTC 96/01, Optimization Technology Center, Argonne National Laboratory and Northwestern University, May 1996.

[2] J. A. George and J. W. H. Liu, *The evolution of the minimum degree algorithm*, SIAM Review, 31 (1989), pp. 1–19.

[3] A. Pothen and C. Sun, *A mapping algorithm for parallel sparse Cholesky factorization*, SIAM J. Sci. Comput., 14 (1993), pp. 1253–1257.

[4] C. Sun, *Efficient parallel solutions of large sparse SPD systems on distributed-memory multi-processors*, Tech. Report CTC92TR102, Advanced Computing Research Institute, Center for Theory and Simulation in Science and Engineering, Cornell University, Ithaca, NY, Aug. 1992.

[5] ——, *Parallel solution of sparse linear least squares problems on distributed-memory multi-processors*, Tech. Report CTC95TR212, Advanced Computing Research Institute, Center for Theory and Simulation in Science and Engineering, Cornell University, Ithaca, NY, May 1995.

Table 3: Performance Results on an IBM SP2

Problem	p	Symbolic	Numeric	MFLOPS	Speedup
300×300	1	2.611	22.120	23.385	1.0
	2	2.420	11.189	46.229	2.0
	4	2.405	5.714	90.521	3.9
	8	2.354	3.626	142.640	6.1
	16	2.402	1.971	262.447	11.2
	32	2.512	1.350	383.226	16.4
400×400	1	4.717	52.604	23.437	1.0
	2	4.373	26.308	46.864	2.0
	4	4.251	13.310	92.630	4.0
	8	4.181	8.080	152.594	6.5
	16	4.237	4.834	255.062	10.9
	32	4.338	2.966	415.724	17.7
$40 \times 40 \times 40$	4	3.827	344.260	75.832	
	8	3.739	172.307	151.508	
	16	3.693	100.387	260.053	
	32	3.894	53.234	490.402	
$50 \times 50 \times 50$	8	9.978	875.529	116.622	
	16	7.359	438.919	232.631	
	32	7.792	223.896	456.042	
BCSSTK25	1	2.471	12.805	22.783	1.0
	2	2.411	6.736	43.314	1.9
	4	2.386	5.009	58.245	2.6
	8	2.463	4.043	72.162	3.2
	16	2.498	3.188	91.512	4.0
BCSSTK33	1	3.015	48.254	25.158	1.0
	2	2.846	27.093	44.806	1.8
	4	2.780	19.540	62.128	2.5
	8	2.892	11.906	101.957	4.1
	16	2.774	6.744	180.015	7.2
	32	3.034	4.819	251.891	10.0