A User's Guide to PSSLS

Chunguang Sun *

October 15, 1996

1 Introduction

PSSLS(Parallel Solution of Sparse Least Squares Problems) is a sparse matrix package for solving large, sparse, linear least squares problems on distributed-memory machines. This package is implemented in C using double-precision arithmetic. The IBM Message-Passing Library(MPL) is used for communication among processors. PSSLS is a parallel direct sparse solver. To solve a sparse linear least squares problem

$$\min_x \|Ax - b\|_2, \tag{1}$$

it first computes the QR decomposition

$$A = Q \begin{bmatrix} R \\ 0 \end{bmatrix}, \tag{2}$$

where A is an $m \times n$ sparse matrix with full column rank, Q is an $m \times m$ orthogonal matrix, and R is a sparse upper triangular matrix. It then obtains the solution to (1) by solving the sparse triangular system $Rx = v$, where v is the first n components of $Q^t b$.

For this version of PSSLS, the sparse matrix A and the right hand side vector b are assumed to be available on each and every processor. The major components of PSSLS are briefly described below.

1. **Matrix structure input.** Provide the sparsity pattern of A to PSSLS.

2. **Symbolic computation.** Symbolic computation consists of column ordering, symbolic factorization and task allocation. A permutation matrix P is determined so that AP has a sparse triangular factor R. The symbolic structure of R is then obtained. The QR decomposition of AP is partitioned into a collection of tasks according to the symbolic structure of R; the tasks are allocated to the processors 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 PSSLS.

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

4. **Numerical factorization.** Compute the numerical values of the nonzero entries of R.
5. **Right hand side input.** Provide the right hand side vector b to PSSLS.
6. **Triangular solution.** Solve the sparse triangular system.

The numerical factorization and the triangular solution are carried out by a blocked-oriented parallel multifrontal scheme described in [4, 5]. PSSLS can be used to solve sparse linear least squares problems with the same sparsity structure but different sets of numerical values. The symbolic computation is executed once. For each set of numerical values, the numerical factorization and the triangular solution are performed. The architecture of PSSLS is illustrated in Fig. 1.

2 User Interface

The user interface of PSSLS 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.

The last formal parameter of every user interface function except the function `PsslsInitialize` 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 PSSLS for solving a sparse linear least squares problem 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`.

Each processor has a copy of the matrix A and the vector b. All processors must call the user interface functions in the order shown in Fig. 1. **All array indices start from zero**.

2.1 Initialization

The function `PsslsInitialize` must be called first for solving a sparse linear least squares problem. For the current version of PSSLS, set the first parameter `method` to 1. The second and the third parameters are the number of rows and the number of columns of the matrix A, respectively. The fourth parameter `h` is the number of right hand side vectors in the sparse least squares problem. For the current version of PSSLS, set the fourth parameter `h` to 1. The fifth parameter `nnz` is the number of nonzeros in A. The last parameter `psspd_data` corresponds to the address of the actual argument `user_data`(i.e. `&user_data`) described above.

```
int PsslsInitialize(int method, int m, int n, int h, 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 must supply to the package the index pairs of all nonzero entries in A.

The function `PsslsStructInputBegin` must be called by every processor to signal the start of the matrix structure input before any index pair is supplied to the package. Similarly, the function

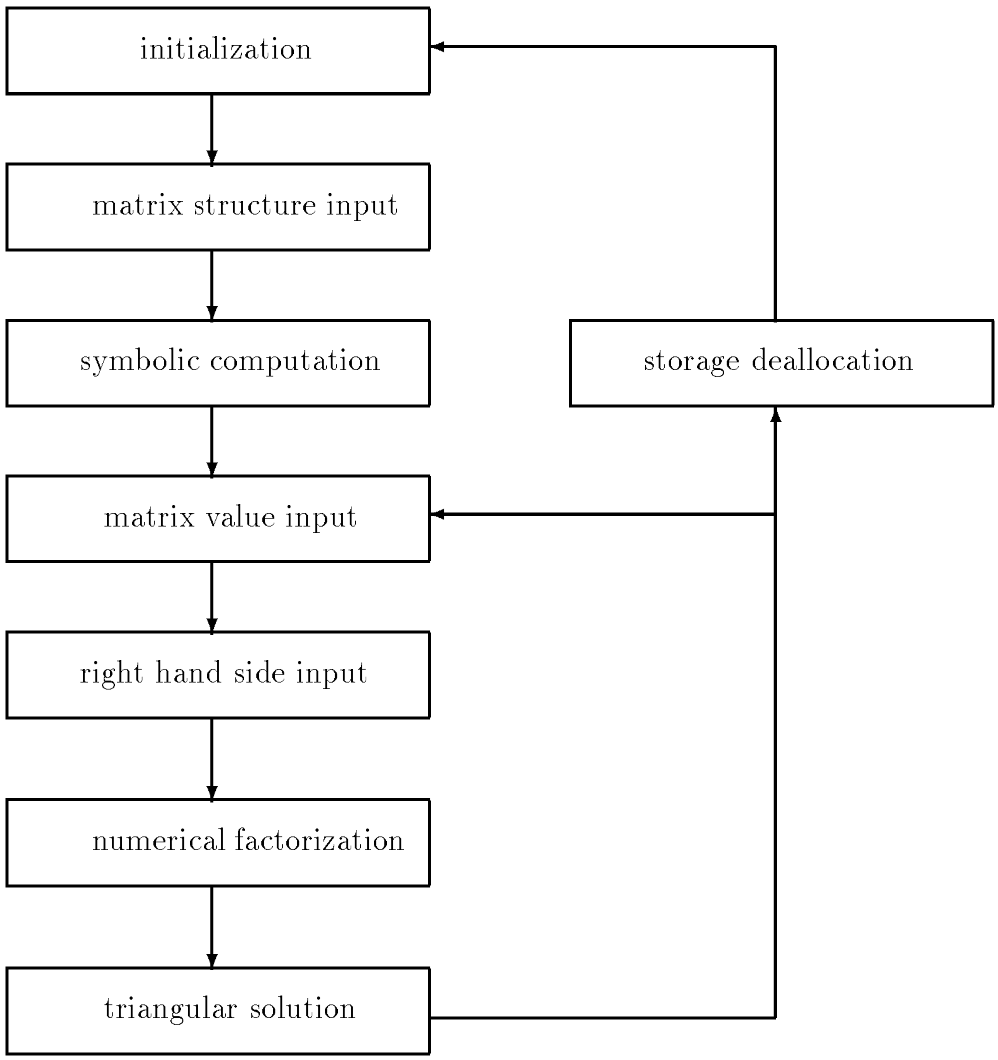

Figure 1: The overall architecture of PSSLS

PsslsStructInputEnd 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 PsslsStructInputBegin(void *data)`

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

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

2.2.3 `int PsslsStructInputEnd(void *data)`

2.3 Symbolic Computation

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

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

2.4 Matrix Value Input

A processor must supply to the package the numerical values of all nonzero entries in A. The function `PsslsValueInputBegin` must be called by every processor to signal the start of the matrix value input before any numerical value is supplied to the package. Similarly, the function `PsslsValueInputEnd` 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 PsslsValueInputBegin(void *data)`

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

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

2.4.3 `int PsslsValueInputEnd(void *data)`

2.5 Right Hand Side Input

The right hand side vector `rhs` is an array of size at least `m`, and it must be available on each and every processor. A processor must supply to the package the numerical values of the entire right hand side vector. Note that the right hand side input should precede the numerical factorization.

```
int PsslsInputRHS(double *rhs, void *data)
```

2.6 Numerical Factorization

```
int PsslsNumericFactor(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 an array 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.

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

2.8 Storage Deallocation

This function deallocates memory space used for solving a sparse linear least squares problem.

```
int PsslsFinalize(void *data)
```

3 Interpretation of Error Code

20	`m < 1`
21	`n < 1`
22	`m < n`
23	`nnz < 0`
24	invalid index pair during matrix structure input
25	`nnz` is too small for matrix structure input.
26	invalid permutation vector is given.
27	invalid elimination tree encountered in symbolic factorization
28	error in symbolic factorization
29	invalid entry in matrix value input
30	`nnz` is too small for matrix value input.
31	inconsistent entry in matrix value input.
32	the matrix is rank-deficient.
33	right hand side vector is not given.

Table 1: Interpretation of the error codes.

4 Remarks and Future Work

All user interface functions must be called by all processors in the order shown in Fig. 1. The sparse matrix A and the right hand side vector b must be available on every processor. Performance results of PSSLS on IBM Scalable POWERparallel System SP2 and Intel iPSC/860 hypercube are described in [4] and [5], respectively.

We have already extended the capability of PSSLS to solve sparse linear least squares problems with dense rows and linear equality constraints [2]. Future release of PSSLS will include capabilities for handling dense rows and linear equality constraints. We are currently developing parallel algorithms and programs for solving rank-deficient sparse linear least squares problems.

References

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

[2] M. T. Heath, *Some extensions of an algorithm for sparse linear least squares problems*, SIAM J. Sci. Stat. Comput., 3 (1982), pp. 223–237.

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

[5] ——, *Parallel sparse orthogonal factorization on distributed-memory multiprocessors*, SIAM J. Sci. Comput., 17 (1996), pp. 666–685.