S^+ Users' Guide (Release 1.1)

Kai Shen, Tao Yang, Xiangmin Jiao, and Steven Richman
University of Rochester & University of California at Santa Barbara
Contact: kshen@cs.rochester.edu; tyang@cs.ucsb.edu

1 Introduction

This document presents the users' guide for S^+. The S^+ package is a set of subroutines for solving general sparse linear systems of the form $A * X = B$ on parallel architectures. Here A is a non-singular $n \times n$ sparse real-coefficient matrix, the right-hand side B is an $n \times m$ dense real-coefficient matrix, and the solution X is an $n \times m$ dense real-coefficient matrix.

S^+ solves sparse linear systems using LU decomposition with partial pivoting and 2D data mapping [2, 4, 5, 6] and it is based on our previous work on parallel sparse LU factorization [1]. S^+ uses the properties of elimination forests to guide supernode partitioning/amalgamation and execution scheduling. This design with 2D mapping effectively identifies dense structures without introducing too many zeros in the BLAS computation and exploits asynchronous parallelism with low buffer space cost. Two space optimization techniques are also incorporated into S^+ to improve the worst-case performance of static symbolic factorization [2].

The package is implemented in ANSI C. It uses MPI (Message-Passing Interface) to handle communications and also uses the BLAS (Basic Linear Algebra Subprograms) library for numerical operations. This version of S^+ can run on both shared-memory and distributed-memory architectures as long as an MPI implementation is available.

S^+ can be obtained on the World Wide Web at `http://www.cs.rochester.edu/u/kshen/research/s+`. Some related papers and other information are also available at the web site. S^+ has been tested and packaged by Sun Microsystems and is part of Sun HPC ClusterTools 4 Software [3]. The binaries and sources for Sun HPC ClusterTools Software can be obtained at `http://www.sun.com/software/hpc/tryandbuy.html`. The source code is made available under Sun Community Source Licensing program.

2 Installation

S^+ package requires an ANSI C compiler for compilation. An MPI library compliant with the Message-Passing Interface Standard Version 1.1 is required. The installation of S^+ includes the following two steps:

- **Step 1.** Depending on the underlying architecture, set up the make.inc file in the top level directory. We have tested S^+ on Cray T3E, SGI Origin 2000, IBM p690 "Regatta", and Linux PC cluster. We provided two sample make.inc files (make.inc.Regatta and make.inc.Linux). Most likely you have to change this file depending on your system configuration.

- **Step 2.** Go to subdirectory src/ and make the library. Library libSplus.a will be generated at the top level directory.

3 How to Use S^+

3.1 Data Structures of Input/Output Matrix

S^+ solves sparse linear system $A * X = B$. In order to use S^+, the user should provide sparse matrix A and right-hand side B. For matrix A, the user should provide the order of the matrix, number of nonzeros, the coefficient and row index of each nonzero in a column-major order, and also the starting index of each column in the column-major order. The following data structure describes the format of A:

```
typedef struct {
  int order;            /* order of the sparse matrix */
  int numNZ;            /* number of nonzeros in the sparse matrix */
  ELEMTYPE *coe;        /* the nonzero coefficients in a column-major order, size=numNZ */
  int *rowIndex;        /* the row indices of nonzeros in coe, size=numNZ */
  int *colStart;        /* starting index of each column in array coe, size=order+1 */
} SP_SparseMatrix;
```

The right-hand side B is a dense matrix. For B, the user should provide the row and column numbers of the matrix, and also the element array in column-major order. The following data structure describes the format of B:

```
typedef struct {
  int rowNum;           /* row number of the dense matrix */
  int colNum;           /* column number of the dense matrix */
  ELEMTYPE *element;    /* element array, size=rowNum*colNum */
} SP_DenseMatrix;
```

The solution X has the same data structure as B. And S^+ actually overwrites right-hand side B with solution X on completion.

3.2 Sample Program for Using S^+

A sample program (sample.c) for using S^+ is provided in subdirectory examples/ of the package. The sample program solves the following sparse linear system:

$$\begin{bmatrix} 19.0 & & 21.0 & 21.0 & \\ 12.0 & 21.0 & & & \\ & 12.0 & 16.0 & & \\ & & & 5.0 & 21.0 \\ 12.0 & 12.0 & & & 18.0 \end{bmatrix} \times X = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \end{bmatrix} \text{ and the solution } X = \begin{bmatrix} -0.031 \\ 0.065 \\ 0.013 \\ 0.062 \\ 0.033 \end{bmatrix}$$

A brief description of the sample program is as follows. After initializing the MPI system, the MPI node 0 initializes the sparse matrix A and right-hand side B, and MPI node 0 also performs the column MMD ordering on input matrix A. After MPI node 0 finishes initialization, all MPI nodes participate in solving the sparse linear system. And finally MPI node 0 releases the space for matrix A and prints out some information.

Two sample programs (sample_inputfromijfile.c and sample_inputfromhbfile.c) are also provided to read the input matrix from files in the ij-value format and the Harwell Boeing format respectively. Currently we only support RUA-type input matrix in the Harwell Boeing format. Two small sample matrix files (sample_inputmatrix.ij and sample_inputmatrix.rua) are also available.

3.3 Matrix Ordering

Matrix ordering is crucial to reduce the space expansion for LU factorization. S^+ provided the column minimum degree ordering based on Joseph Liu's algorithm. The user can also choose to order the matrix beforehand and bypass the ordering routine provided by S^+. (See routine `SP_Ordering` for details.)

4 User-callable S^+ Subroutines

All S^+ routines use the `MPI_COMM_WORLD` communicator for their communication. MPI should be initialized with `MPI_Init` before calling any S^+ function.

4.1 SP_Ordering

```
void SP_Ordering (int mode, SP_SparseMatrix *A, int **perm_c, int **perm_r);

=======
SP_Ordering obtains permutation vectors Pc and Pr. The LU factorization of A*Pc tends
to have less fill than the LU factorization of A. And Pr is applied after Pc to
minimize the number of zero diagonal elements. This is a sequential routine and should
only be called by one MPI node before calling SP_Solve.

    int mode,             /* INPUT - specifies the way ordering is conducted:
                             mode==0: natural ordering (i.e., Pc=I, Pr=I)
                             mode==1: MMD ordering of the structure of A'*A and
                                      a row permutation to minimize zero diagonals */
    SP_SparseMatrix *A,   /* INPUT - sparse matrix to be ordered */
    int **perm_c,         /* OUTPUT - column permutation vector of size A->order,
                             which defines the permutation matrix Pc; perm_c[i] = j
                             means column i of A is in position j in A*Pc */
    int **perm_r          /* OUTPUT - row permutation vector of size A->order */
```

4.2 SP_Solve

```
void SP_Solve (SP_SparseMatrix *A, SP_DenseMatrix *B, int *perm_c, int *perm_r,
               int me_no, int nproc);
```

=======
SP_Solve solves linear system A*X=B, the storage of B is overwritten by result X on completion. This routine should be called by all participating MPI nodes.

```
    SP_SparseMatrix *A,   /* INPUT - sparse matrix */
    SP_DenseMatrix *B,    /* INPUT/OUTPUT - right-hand size dense matrix,
                             will be overwritten by result matrix on completion */
    int *perm_c,          /* INPUT - column permutation vector */
    int *perm_r,          /* INPUT - row permutation vector */
    int me_no,            /* INPUT - local MPI node ID */
    int nproc             /* INPUT - total number of MPI nodes */
```

4.3 SP_SetParam_BlkSize

```
int SP_SetParam_BlkSize (int size);
```

=======
SP_SetParam_BlkSize sets the maximum size a block can be. This value affects caching performance and it should be set properly to allow 3 blocks coexist in the first level cache at the same time, the default value is 28. Returns 1 if successful, 0 otherwise. This routine should be called by all participating MPI nodes with the same parameter.

```
    int size              /* the value set for the block size limit */
```

4.4 SP_GetParam_BlkSize

```
int SP_GetParam_BlkSize ();
```

=======
SP_GetParam_BlkSize returns the block size limit. Please refer to SP_SetParam_BlkSize for the description of the block size limit.

4.5 SP_SetParam_Relax

```
int SP_SetParam_Relax (double relax);
```

=======
SP_SetParam_Relax sets the relax parameter for amalgamation. This value specifies the maximum amount of extra fill-ins to be generated by amalgamation. The value should be big enough to increase the average block size while it should also be small enough to

reduce the number of extra fill-ins. The default value is 0.3 which means at most 30% extra fill-ins are allowed in amalgamation. Returns 1 if successful, 0 otherwise. This routine should be called by all participating MPI nodes with the same parameter.

```
    double relax          /* the value set for the relax parameter */
```

4.6 SP_GetParam_Relax

```
double SP_GetParam_Relax ();
```

```
=======
```

SP_GetParam_Relax returns the relax parameter for amalgamation. Please refer to SP_SetParam_Relax for the description of the relax parameter.

4.7 SP_SetParam_ThrholdPivot

```
int SP_SetParam_ThrholdPivot (double thrhold);
```

```
=======
```

SP_SetParam_ThrholdPivot sets the parameter for threshold pivoting. Threshold pivoting allows the pivot choice to be other than the largest element in the pivot column, as long as it is within a certain fraction (u <= 1.0) of the largest element. A smaller u allows more freedom in pivot selection, however, it might also weaken the numerical stability of LU factorization. Returns 1 if successful, 0 otherwise. This routine should be called by all participating MPI nodes with the same parameter.

```
    double thrhold          /* the value set for the threshold pivoting parameter */
```

4.8 SP_GetParam_ThrholdPivot

```
double SP_GetParam_ThrholdPivot ();
```

```
=======
```

SP_GetParam_ThrholdPivot returns the parameter for threshold pivoting. Please refer to SP_SetParam_ThrholdPivot for the description of the parameter.

4.9 SP_SetParam_BatchPivot

```
int SP_SetParam_BatchPivot (int batchpivot);
```

```
=======
```

SP_SetParam_BatchPivot sets the parameter for batch pivoting. Batch pivoting can reduce the synchronization frequency during pivot selection. There are three options for this parameter: SP_PIVOTING_COLBYCOL means column-by-column pivoting (or no batch

```
pivoting), SP_PIVOTING_LD means large diagonal batch pivoting; SP_PIVOTING_SBP means
speculative batch pivoting. Returns 1 if successful, 0 otherwise. This routine should
be called by all participating MPI nodes with the same parameter.

    int batchpivot          /* the value set for batch pivoting parameter */
```

4.10 SP_GetParam_BatchPivot

```
int SP_GetParam_BatchPivot ();

=======
SP_GetParam_BatchPivot returns the parameter for batch pivoting. Please refer to
SP_SetParam_BatchPivot for the description of the parameter.
```

Acknowledgment. This work was supported in part by NSF CCR-9702640, NSF ITR-0082666, NSF CCF-0448413, and by DARPA through UMD (ONR Contract Number N6600197C8534). We would like to thank Cong Fu, Horst Simon, Stefan Boeriu, Andrew Sherman, Vinod Gupta, Esmond Ng, Apostolos Gerasoulis, Joseph Liu, Tim Davis, Jim Demmel, and Sherry Li for their helpful comments and valuable support. Bin Jiang contributed to the implementation of S^+. Chuanpeng Li contributed to the support of the Harwell Boeing input matrix format.

References

[1] C. Fu, X. Jiao, and T. Yang. Efficient Sparse LU Factorization with Partial Pivoting on Distributed Memory Architectures. *IEEE Trans. on Parallel and Distributed Systems*, 9(2):109–125, February 1998.

[2] B. Jiang, S. Richman, K. Shen, and T. Yang. Efficient Sparse LU Factorization with Lazy Space Allocation. In *Proc. of the 9th SIAM Conf. on Parallel Processing for Scientific Computing*, San Antonio, Texas, March 1999.

[3] G. Kechriotis. Personal Communication, 1999, 2000, 2001.

[4] K. Shen. Parallel Sparse LU Factorization on Second-class Message Passing Platforms. In *Proc. of the 19th ACM Conf. on Supercomputing*, pages 351–360, Cambridge, MA, June 2005.

[5] K. Shen, X. Jiao, and T. Yang. Elimination Forest Guided 2D Sparse LU Factorization. In *Proc. of the 10th ACM Symp. on Parallel Algorithms and Architectures*, pages 5–15, Puerto Vallarta, Mexico, June 1998.

[6] K. Shen, T. Yang, and X. Jiao. S+: Efficient 2D Sparse LU Factorization on Parallel Machines. *SIAM J. Matrix Anal. Appl.*, 22(1):282–305, 2000.