SpLLT Documentation

Release 1.0

Florent Lopez

Aug 23, 2019

Contents:

1 Purpose **3**

2 Installation **5**
- 2.1 Quick Start . . . 5
- 2.2 Third-party libraries . . . 5
- 2.3 Compilers and compiler options . . . 6
- 2.4 Support . . . 6

3 Subroutines **7**
- 3.1 Basic subroutines . . . 7

4 Indices and tables **9**

Index **11**

SpLLT

CHAPTER 1

Purpose

SpLLT is a direct solver for solving **large sparse symmetric positive-definite linear systems** of equations:

$$AX = B$$

This is done by computing the Cholesky decomposition of the input matrix:

$$PAP^T = LL^T$$

where the factor L is a lower triangular matrix and the matrix P is a permutation matrix used to reduce the fill-in generated during the factorization. Following the matrix factorization the solution can be retrieved by successively solving the system $LY = PB$ (forward substitution) and $L^T PX = Y$ (backward substitutions).

CHAPTER 2

Installation

2.1 Quick Start

Under Linux, or Mac OS X:

```
# Get latest development version from github
git clone https://github.com/NLAFET/SpLLT
# Move to source directory
cd SpLLT
# Create build directory
mkdir build
cd build
# Create Makefile with cmake command. The -DRUNTIME option can be
# used to select a runtime system.
cmake <path-to-source> -DRUNTIME=<StarPU|OMP|Parsec>
# Build SpLLT software
make
```

2.2 Third-party libraries

2.2.1 BLAS and LAPACK

The solver depends on high performance BLAS and LAPACK libraries to perform the dense linear algebra operations that are abundant in our kernels. For best performance, please use the library recommended by your computer manufacturer (normally the Intel MKL). If this is not available, use an optimized alternative, such as OpenBLAS. The reference BLAS and reference LAPACK libraries from netlib are at least an order of magnitude slower than modern optimized BLAS, and should be avoided. If bit-compatible results are desired, a bit-compatible BLAS library must be used.

The BLAS library can be passed to *cmake* using the `LBLAS` variable and the LAPACK library can be passed using the `LLAPACK` variable as following:

```
cmake <path-to-source> -DLBLAS=/path/to/blas -DLLAPACK=/path/to/lapack
```

2.2.2 SPRAL

SPRAL is an open source (BSD) library for sparse linear algebra and associated algorithms. It can be downloaded directly from the SPRAL GitHub repository: https://github.com/ralna/spral.

```

```

2.2.3 Runtime system

In this package, we use a runtime system for handling the parallel execution of the code. SpLLT currently supports three runtime systems among OpenMP, StarPU and Parsec that can be set using the `RUNTIME` variable when running the *cmake* command. For example, to compile the OpenMP version:

```
cmake <path-to-source> -DRUNTIME=OMP
```

For StarPU and Parsec runtime systems, it is possible to specify which library to use for the compilation. For example, when building SpLLT with StarPU, you can pass the StarPU directory using the `STARPU_DIR` variable:

```
cmake <path-to-source> -DRUNTIME=StarPU -DSTARPU_DIR=/path/to/StarPU
```

2.3 Compilers and compiler options

If no compiler is specified, *cmake* will pick a default compiler to use. If *cmake* cannot find an appropriate compiler, or you wish to specify a different compiler you can do so by setting the following variables:

CC specifies the C compiler to use.

FC specifies the Fortran 90/95/2003/2008 compiler to use.

NVCC specifies the CUDA compiler to use.

Additionally, compiler flags can be specified using the following variables:

CFLAGS specifies options passed to the C compiler.

FCFLAGS specifies options passed to the Fortran compiler

NVCCFLAGS specifies options passed to the CUDA compiler.

For example, to compile with `ifort -g -O3 -ip` we could use:

```
FC=ifort FCFLAGS="-g -O3 -ip" cmake <path-to-source>
```

2.4 Support

Feeback may be sent to florent.lopez@stfc.ac.uk or by filing an issue on our github: https://github.com/NLAFET/SpLLT/issues.

CHAPTER 3

Subroutines

3.1 Basic subroutines

subroutine spllt_analyse(*akeep*, *fkeep*, *options*, *n*, *ptr*, *row*, *info*, *order*)

Perform the analyse phase of the factorization (referred to as symbolic factorization) for a matrix supplied in Compressed Sparse Column (CSC) format. The resulting symbolic factors stored in *akeep* should be passed unaltered in the subsequent calls to `spllt_factor()`. This routines also initializes the numeric factorization data stored in *fkeep* that should also be passed in the subsequent calls to `spllt_factor()`.

Parameters

- **akeep** *[spllt_akeep,inout]* :: symbolic factorization data.
- **fkeep** *[spllt_fkeep,inout]* :: numeric factorization data.
- **options** *[spllt_options,in]* :: user-supplied options to be used.
- **n** *[integer,in]* :: order of the system.
- **ptr** (n+1) *[integer,in]* :: column pointers for lower triangular part.
- **row** (ptr(n+1)-1) *[integer,in]* :: row indices of lower triangular part.
- **info** *[spllt_inform,out]* :: exit status.
- **order** (n) *[integer,out]* :: permutation array used for matrix ordering.

subroutine spllt_factor(*akeep*, *fkeep*, *options*, *val*, *info*)

Perform the numerical factorization of the matrix whose structure is kept in *akeep* that is determined with the `spllt_analyse()` routine. The numerical factors are stored in *fkeep* and should be passed unaltered in the subsequent calls to `spllt_solve()` for computing the solution to the system.

Parameters

- **akeep** *[spllt_akeep,inout]* :: symbolic factorization data.
- **fkeep** *[spllt_fkeep,inout]* :: numeric factorization data.
- **options** *[spllt_options,in]* :: user-supplied options to be used.

- **val** (*) *[real,in]* :: non-zero values for A in same format as for the call to `ssids_analyse()`
- **info** *[spllt_inform,out]* :: exit status.

Note: This routine call is asynchronous, the routine *`spllt_wait()`* should be call afterwards to make sure that the factorization has been completed.

`subroutine spllt_solve`(*fkeep*, *options*, *order*, *x*, *info*[, *job*])
Solves for multiple right-hand sides on the following problems:

job	Equation solved
0 (or absent)	$AX = B$
1	$PLX = B$
2	$(PL)^T X = B$

Parameters

- **fkeep** *[spllt_fkeep,in]* :: numeric factorization data.
- **options** *[spllt_options,in]* :: user-supplied options to be used.
- **order** (n) *[integer,out]* :: permutation array used for matrix ordering.
- **x** (n,nrhs) *[real,inout]* :: right-hand sides B on entry, solutions X on exit. *n* represents the order of matrix A.
- **info** *[spllt_inform,out]* :: exit status.

Options **job** *[integer,in]* :: specifies equation to solve, as per above table.

`subroutine spllt_wait`()
Wait for all the tasks submitted in a previous function call to be completed.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index

S

spllt_analyse() (fortran subroutine), **7**
spllt_factor() (fortran subroutine), **7**
spllt_solve() (fortran subroutine), **8**
spllt_wait() (fortran subroutine), **8**