SCALAPACKFX

Release 1.2.0

B. Aradi

Jun 30, 2023

Contents

Chapter 1

About SCALAPACKFX

SCALAPACKFX is a library containing modern Fortran (Fortran 2003) wrappers around SCALAPACK, PBLAS and BLACS routines. The goal is to make the use of those libraries as simple as possible in Fortran.

Consider for example a simple broadcast in BLACS. In order to broadcast an integer array (aa) with 5x5 elements using the appropriate BLACS routine, you have to issue:

```
call igebs2d(ictxt, "All", " ", 5, 5, aa, 5)
```

Additional to the object to be broadcasted and the communicator, you also *must* specify following arguments:

- type of the array as first letter of the subroutine name (although it is *known* at compile-time)
- number of row, number of columns and leading dimension of the array (although those are *known* at run-time)
- scope of the broadcast (could be optional as very often it is “All” any way)
- communcation pattern for the broadcast (could be optional as very often ” ” is used)

Using SCALAPACKFX the call above is as simple as:

```
call blacsfx_gebs(mygrid, aa)
```

No redundant arguments, sensible defaults. Nevertheless the full functionality still available via optional parameters if needed. E.g. if you wanted to the scope, you could write:

```
call blacsfx_gebs(mygrid, aa, scope="Row")
```

Also, the array aa does not have to be a rank two array, it can be also rank one (vector) or rank zero (scalar).

A few routines are already covered (see *List of routines* (page 7)). If your desired routine is not among them yet, you are cordially invited to extend SCALAPACKFX and to share it in order to let others profit from your work (SCALAPACKFX is licensed under the simplified BSD license). For more details see the project home page.

Chapter 2

Compiling and installing SCALAPACKFX

In order to compile SCALAPACKFX, you need following prerequisites:

- Fortran 2003 compiler*[0],
- GNU M4 macro interpreter,
- GNU Make.

There are basically two different ways of using the library in your project:

- *Precompiling the library* (page 3) and linking it later to your project.
- *Compiling the library during your build process* (page 4).

Both are described below.

2.1 Precompiling the library

In order to create a precompiled library

1. Copy the file *make.arch.template* to *make.arch* in the root directory of the source and customize the settings for the compilers and the linker according to your system.
2. Change to the *src/* folder.
3. Issue *make* to build the library.
4. Copy *all* module files (usually ending on *.mod* and the library *libscalapackfx.a* to a place, where your Fortran compiler and your linker can recognize them.

During the build process of your project, you may link the library with the *-lscalapackfx* option. Eventually, you may need to specify options for your compiler and your linker to specify the location of those directories. Assuming you've put the module files in the directory *<MODFILEDIR>* and the library file in *<LIBRARYDIR>*, you would typically invoke your compiler for the source files using the *libscalapackfx_module* as:

```
F2003_COMPILER -I<MODFILEDIR> -c somesource.f90
```

and link your object files at the end with:

```
LINKER -I<LIBRARYDIR> somesource.o ... -L<LIBRARYDIR> -lscalapackfx
```

[0] GNU Fortran 4.9 (earlier versions may not work!), NAG Fortran 5.3.1 and Intel Fortran 12.1 seem to work.

2.2 Compiling the library during your build process

In order to build the library during the build process of your project:

1. Copy the content of the *src/* folder into a *separate* folder within your project.
2. During the make process of your project, invoke the library makefile (*Makefile.lib*) to build the module files and the library in the folder where you've put the library sources.

   You must pass the compiler and linker options via variable defintions at the make command line. Assuming that the variables *$(FXX)*, *$(FXXOPT)*, *$(LN)* and *$(LNOPT)*, *$(M4)* and *$(M4OPT)* contain the Fortran compiler, the Fortran compiler options, the linker, the linker options, the M4 preprocessor and its options, respectively, you would have something like:

```
$(SCALAPACKFX_SRCDIR)/libscalapackfx.a:
        $(MAKE) -C $(SCALAPACKFX_SRCDIR) \
            FXX="$(FXX)" FXXOPT="$(FXXOPT)" \
            LN="$(LN)" LNOPT="$(LNOPT)" \
            M4="$(M4)" M4OPT="$(M4OPT)" \
            -f Makefile.lib
```

   in the makefile of your project with *$(SCALAPACKFX_SRCDIR)* being the directory where you've put the source of SCALAPACKFX.

You should also have a look at the *GNUmakefile* in the *test/* folder of SCALAPACKFX, which uses exactly the same technique to compile the library during the build process for the tests.

Chapter 3

Using SCALAPACKFX

Before you can use the SCALAPACKFX routines, you need basically the following two steps.

1. Import the module *libscalapackfx_module* in your routines.
2. Initialize a grid for the BLACS-communcation using *type(blacsgrid)*.

Below you find an example draft for reading in a matrix from a file, distributing it across the processes and finally diagonalizing it:

```
program test_diag
  use libscalapackfx_module

  integer, parameter :: nprow = 4, npcol = 4   ! process rows/cols
  integer, parameter :: bsize = 64             ! block size
  integer, parameter :: nn = 1000              ! matrix size

  type(blacsgrid) :: mygrid  ! BLACS grid descriptor
  real(dp), allocatable :: aa(:,:), bb(:,:), eigvecs(:,:), eigvals(:)
  integer :: desc(DLEN_)    ! matrix descriptor
  integer :: mloc, nloc     ! nr. of local rows/columns of the matrices
  type(linecomm) :: distr               ! distributes matrix when read
  real(dp), allocatable :: iobuffer(:)  ! buffer during read

  ! Initialize your BLACS grid
  call mygrid%gridinit(nprow, npcol)

  ! Allocate the local part of the distributed matrices
  call scalafx_getdescriptor(mygrid, nn, nn, bsize, bsize, desc)
  call scalafx_getlocalshape(mygrid, desc, mloc, nloc)
  allocate(aa(mloc, nloc))
  allocate(bb(mloc, nloc))
  allocate(eigvecs(mloc, nloc))
  allocate(eigvals(nn))
  ...
  ! Here comes the code which distributes your matrix
  ! You can use the linecomm type to distribute it if you read it from file
  if (mygrid%lead) then
      allocate(iobuffer(nn))
  end if
  call distr%init(desc, "c")
  do icol = 1, nn
    if (mygrid%lead) then
      read(12, *) iobuffer(:)
      call distr%setline_lead(mygrid, icol, iobuffer, aa)
```

(continues on next page)

(continued from previous page)

```
    else
      call distr%setline_follow(mygrid, icol, mtxloc)
    end if
  end do
  ...
  ! Get eigenvalues (on all nodes) and eigenvectors (distributed)
  call psygvd(aa, desc, bb, desc, eigvals, eigvecs, desc, jobz="V", uplo="L")
  ...
end program test_diag
```

Have a look at the test folder in the source tree for further examples.

Chapter 4

List of routines

You can generate the list and the description of the SCALAPACKFX routines via doxygen (see folder *doc/doxygen/* in the source tree) or examples as sphix documentation.

Chapter 5

License

SCALAPACKFX is licensed under the simplified BSD license:

```
Copyright (c) 2013, Bálint Aradi

All rights reserved.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice,
  this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```