MPI-SCATCI

Ahmed F. Al-Refaie

September 5, 2017

1 Introduction

> People are very flexible and learn to adjust to strange surroundings — they can become accustomed to read Lisp and Fortran programs, for example.
>
> Leon Sterling and Ehud Shapiro

MPI-SCATCI is a complete rewrite of SCATCI that utilizes MPI in order to compute and diagonalize target and scattering Hamiltonians. The code heavily uses Object Oriented Programming (OOP) paradigms in the FORTRAN 2003 standard to provide a robust and flexible codebase that can cope with extremely large matrices and scale well with large problem sizes of order 1,000,000+

2 Compiling

Supplied with the code is a Makefile and an included makefile.inc to provide the necessary environment variables to compile the code. It is recommended that the user supplies these variables thorugh a make.inc file that is configured for their system. As of writing, only 64-bit integer compilation has been tested rigourously and is therefore a requirement for the code. Optimally the latest compilers of your respective vendor are recommended as they are likely to fully implement the FORTRAN 2003 standard as well.

2.1 Requirements

Currently, MPI-SCATCI requires the following compilers and libraries in order to be compiled:

- C++99 and FORTRAN 2003 (64-bit integer) compilers
- LAPACK (via MKL or otherwise)
- ARPACK
- URKMOL+ integral library precompiled

Optional features can be added depending on compiler flags:

- MPI and associated MPI compilers
- SCALAPACK and BLACS
- PETSC (3.7+)
- SLEPC (3.7+)

In make.inc files the current set of flags that can be optionally set are:

- `HAVE_MPI`: Adds MPI and allows for distributed paralleism (requires SCALAPACK) (Must be also set in UKRMOL+)
- `HAVE_64BIT_MPI`: Determines whether MPI uses 64-bit integers or not
- `MPI_TYPE`: Determines what type of MPI library you have (`-Dintelmpi` only supported)
- `HAVE_SLEPC`: Wheter you have the SLEPC sparse diagonalizer library included.
- `HAVE_MPI3`: If you want to use bigger integrals in MPI-SCATCI using shared memory feature of MPI-3.0 (Must be also set in UKRMOL+)
- `HAVE_64_BIT_BLAS`: Allows MPI-SCATCI to know if 32-bit or 64-bit BLAS is used. (`-lp64` vs `-ilp64`)

If all flags have been determined then it is simply a case of running make. Removing all of these flags will generate a serial MPI-SCATCI code with standard SCATCI diagonalizers.

MPI libraries generally supply a compiler wrapper of some kind (e.g. mpif90, mpicc etc) that automatically provides MPI includes and libraries at the correct stages of compilation. When using MPI, the `HAVE_MPI` variable must be set to `-Dusempi`. Additionally, Intel MPI libraries that are 64-bit integer compatible require the additional `HAVE_64BIT_MPI= -Dmpi64bitinteger` and `MPI_TYPE= -Dintelmpi` flags to be set as well.

The LAPACK and SCALAPACK can either be downloaded, compiled and linked from various sources but it is recommended to link the MKL libraries, in particular the parallel versions, for best performance. The Intel MKL link advisor can be used to help find the right link flags to use. When using the MKL libraries, it is important to know which interface is being used, i.e. 32-bit lp64 integers or 64-bit ilp64 integers. MPI-SCATCI can handle both regardless but requires knowledge through the setting/unsetting of `HAVE_64_BIT_BLAS= -Dblas64bit` flag.

ARPACK can either be downloaded and compiled as given in the project guides folder in UKRMol-in or supplied by the high performance computing (HPC) center through the loading of modules. If this is a dynamic library then it must be of the same integer kind as the one used by MKL.

2.2 PETSC and SLEPC

PETSC and SLEPC are optional if the `HAVE_SLEPC= -Dslepc` flag is set. They are MPI sparse diagonalizers and can be freely downloaded from `https://www.mcs.anl.gov/petsc/` and `http://slepc.upv.es/`. They require the exporting of environment variables `PETSC_DIR`, `PETSC_ARCH` and `SLEPC_DIR` in order to find includes and libraries.

The requirements are fairly small (only BLAS and LAPACK) for this code and don't require a lot of the extensions available for them. It is simply a case of running the commands:

```
>wget  http://ftp.mcs.anl.gov/pub/petsc/release-snapshots/petsc-lite-3.7.5.tar.gz
>tar -xzvf petsc-lite-3.7.5.tar.gz
>cd petsc-3.7.5
>export PETSC_DIR=‘pwd‘
>./configure --with-cc=mpiicc --with-cxx=mpiicpc --with-fc=mpiifort \
  --with-blas-lapack-dir=$MKLROOT/lib/intel64/
>make
```

It will default to a `PETSC_ARCH=arch-linux2-c-debug` which should be exported:

```
>export PETSC_ARCH=arch-linux2-c-debug
```

Compiling SLEPC is even simpler:

```
>wget http://slepc.upv.es/download/download.php?filename=slepc-3.7.3.tar.gz
>tar -xzvf slepc-3.7.3.tar.gz
>cd slepc-3.7.3
>export SLEPC_DIR=‘pwd‘
>./configure
>make
```

After this has been done, MPI-SCATCI will automatically link these libraries as long as `PETSC_DIR`,`PETSC_ARCH` and `SLEPC_DIR` are defined.

3 Usage

MPI-SCATCI has been designed to be backwards compatible with inputs from serial SCATCI. IT has also been designed to work with all currently available scripts as well. There are two ways to run MPI-SCATCI, first through standard input:

```
>./MPI-SCATCI.x < input > output
```

Or though reading the file:

```
>./MPI-SCATCI.x input > output
```

When running without MPI or with only one MPI process, all output will be passed into standard output. When running with 4 process for example:

```
>mpirun -np 4 ./MPI-SCATCI.x input > output
```

Rank 0 outputs to stdout and can be piped whilst all other ranks the output is printed to files with name `log_file` with the processor number appended at the end. It is not recommened to pass standard input when using more than one process as it cannot be guaranteed that all MPI processes will get it.

3.1 Input

All input terms used by SCATCI are compatible with MPI-SCATCI. There are a few new keywords:

- `VECSTORE`: (String) Determines which eigenvector coefficients are stored. Can be either:
  - `CONT`: Continuum coefficients only
  - `L2`: L^2 coefficients only
  - `ALL`: All coefficents are stored (default)
- `FORSE`: (Integer)
  - 0: (default) Use diagonlizers as normal

  - 1: Force use of serial diagonlizers when using MPI.

- `EXRC`: (Integer) Removes row and column given from the Hamiltonian matrix before diagonlization. Reduces Hamiltonian size by one.

- `MEMP`: (Double) Size of memory per processor in gigabytes (Default is 2.5)

The most important is `MEMP` which describes the memory in gigabytes for each MPI process, *not the total memory available*. Setting this appropriately can imporve performance. An example is a 4 core system with 16GB, when running with 4 MPI-processes this value is 4, with 2 processes this is 8 etc. The minimum memory required for each process is the amount needed to hold all of the integrals and their portion of the matrix. The default value if this is not set is 2.5 GB (the default per core for ARCHER).

3.2 Output

Both target only and scattering calculations output fort files that are compatible with further stages of the UKRmol pipeline (such as DENPROP and outer region codes). The only caveat is that scattering calculations with certain diagonalizers will not always output matrix fort files as they are done in-core.

3.3 Running on ARCHER

Included in the scripts folder are submission scripts that will automatically submit the job and input file to the ARCHER queue. Both `b_mpici.sh` and `run_mpici.csh` are required. To submit a job simply requires the command:

```
>./b_mpici.sh [inputfile] [number of cores] [wall time]
```

For example, to run a 240 core job with the input file `ci.inp` for 2 hours is:

```
>./b_mpici.sh ci.inp 240 2
```

For testing purposes, this method may take a while to run. An alternative script `b_mpici_short.sh` will submit the job into the short queue which will begin running quickly but is limited to 192 cores and 20 minutes.

3.4 Diagonalization

MPI-SCATCI utilizes either Legacy SCATCI diagonalizers for single process runs or MPI diagonalizers for multi process runs. All diagonalizers can be mixed into the same run depending on the number of eigenvalues requested by `NSTAT` or forced to use a single type. Table 1 describes all available diagonalizers in the current code.

Table 1: Table describing the diagonalizers available. IGH is the forced diagonalizer keyword in input. Rule describes which one is used when IGH is not defined. N is the matrix size and N_λ is the number of eigenvalues which can be set by the NSTAT keyword. Serial and MPI describe which library is used depending on whether number of processes is 1 or > 1 respectively

		Diagonalizer			
Type	IGH	Rule	Serial	MPI	Notes
Davidson	0	$N_\lambda \leq 3$	SCATCI-Davidson	SLEPC-Krylovschur	
Iterative	-1	$3 < N_\lambda < 0.2N$	ARPACK	SLEPC-Krylovschur	
Dense	1	$N_\lambda > 0.2N$	LAPACK	SCALAPACK	

4 Extending the code

One of the key features of MPI-SCATCI is the ability to add new features without touching the main Hamiltonian building process. Currently it is possible to easilt add new integrals and MPI diagonalizers.

4.1 Integrals

All integrals are extensions of the `BaseIntegral` class. Setup is performed by defining an `initialize_self` routine, loading into core by `load_integrals` and cleanup by `destroy.` In particular, the `get_integralf` subroutine is the main inteface between the actual integrals and the hamiltonain build process. Additionally a `write_geometries` subroutine must also be provided in order to generate correct CI vector files.

4.2 Diagonalizers

Diagonalizers come in two parts. First there is the `BaseMatrix` class which defines a format that the matrix elements are stored and act as the interface between the hamiltonain and diagonalizers via the `insert_matrix_element` subroutine. Using optimal matrix formats on certain diagonalizers can improve performance substantially and reduce memory footprints significantly. Whilst simple to implement for serial runs, this may prove complicated for MPI matrices such as SCALAPACK and SLEPC. Therefore another class `Distributed Matrix` which inherits from `BaseMatrix` provides a fast and easy way to define these formats. It requires only `setup_diag_matrix` which is used to setup any important arrays (e.g. for the SLEPC matrix it sets up the PETSC matrix backbone) and most importantly `insert_into_diag_matrix` which defines the rules on whether to store the matrix element or not. It is imperative that memory usage is well defined in `setup_diag_matrix` so that the class can work as efficiently as possible.

Whilst the main technical details of this is described in the technical manual, the main point of `Distributed Matrix` is that every process will touch every single matrix element at some point. The question is whether it should personally keep it or not which is where `insert_into_diag_matrix` comes in. An example for SCALAPACK is simply:

```
  !>@brief
!>This inserts an element into the hard storage which is considered the final
!>location before diagonalization
!>It also checks wherther the element exists within the aloowed range and tells us
!> if it was successfully inserted
logical function insert_into_local_matrix(this,row,column,coefficient)
class(SCALAPACKMatrix)  :: this
integer,intent(in) :: row,column
real(wp),intent(in) :: coefficient
BLAS_INT :: proc_row,proc_col,i_loc,j_loc
BLAS_INT :: blas_row,blas_col

blas_row = row
blas_col = column

if(row==column) call this%store_diagonal(row,coefficient)

if(this%am_i_involved() == .false.) return

!Figure out which proc it belongs to and the local matrix index
call infog2l(blas_row, blas_col, this%descr_a_mat, this%nprow, this%npcol, &
  & this%myrow, this%mycol, i_loc, j_loc, proc_row, proc_col)
!If it does belong to me then store
if(this%is_this_me(proc_row,proc_col)) then

this%a_local_matrix(i_loc, j_loc) = coefficient

 insert_into_local_matrix = .true.

else
 !Otherwise it is ignored
 insert_into_local_matrix = .false.

endif

end function
```

It checks whether it is within the block cyclic distribution. If it is then it is stored for diagonalization, if not then it is discarded. The routine is a function that must return true if it was successful and false if it was not.

Once these functions have been defined, MPI-SCATCI will automatically distribute the matrix correctly during the Hamiltonain build. This is a powerful feature as it allows a range of diagonalizers to be used without any change to the Hamiltonain build code.

The second part is the diagonalizer itself which is defined by the `Diagonalizer` abstract class. Here the matrix is passed into the `diagonalize` subroutine with parameters such as the number of eigenvalues etc and diagonalization is performed. A check on whether the marix format passed is supported. Whilst there is a `get_matrix_element` subroutine in the Matrix classes, it is often difficult to define it (especially for distributed matrices) so it is recommeneded to check and then typecast the BaseMatrix into the appropriate format to be used directly by the diagonalizer. The advantage of this is that more often then not the matrix elements are immediately available for diagonalization and require no conversion.

Inserting both is easily accomplished by modifing the matrix diagonalizer Dispatcher and inserting a new rule. Once this is done then MPI-SCATCI will automatically utilize the Diagonalizers.

5 Issues

5.1 SLEPC Eigenvector limit

the only big issue is SLEPC has a storage requirement per core of ncv^2 (number of column vectors) for each process which is generally ncv > nev (number of eigenvalues). this is not an issue for 10% of a 100,000 matrix which requires only a minimum of 10,001 column vectors (800 MB per core) but becomes an issue when we deal with 1,000,000 size matrix. This would require storing a matrix of 100,000 x 100,000 for each processor requiring 74 GB!!! There are a lot of other methods like spectrum slicing that could alleviate this but it requires adding in multiple libraries such as ParMetis, MUMPS and is also extremely slow. Right now it is best to use SCALAPACK until I can find a tangible solution.