Build-system internals

Target audience

This document is for all Abinit developers who want to understand how the build system works, for Abinit maintainers who want to hack the build system, and for collaborators who want to enhance the interoperability of their projects with Abinit.

Linear algebra

The following simplified diagram summarizes how the build system detects linear algebra features. The starting point is colored in blue, the ending point in green, and the decision points in pink. Enhancements are framed with dashed lines.

The driving parameter of the detection process is the with_linalg_libs option. When specified, the locations of the libraries are known but their features are unknown. When not specified, the other options of configure provide hints on what should be looked for, but the possible locations and names of the libraries are only partly known. Enters the with_linalg_flavor option into play, orientating the search for libraries and/or features. As can be seen here, the most influencing parameters are the kinds of parallelisms selected at configure-time. They determine the number of tests performed and the priorities attributed to each linear algebra component. For sake of simplicity, we will not go any further into the details, the essential being the overall picture and the interactions between the involved options of configure.

FFT specifications

The following specifications were initially proposed by Matteo Giantomassi in a Gitlab issue on February 7, 2020.

I think that the release of Abinit9 gives us the opportunity to simplify a bit the treatment of the FFT libraries. We should indeed clarify the procedure that must be followed by users to link an external FFT library.

So far the approach has been “I have an external FFT library -> I use fft_flavor=“fftw3” and I expect that everything will work out of the box”.

Unfortunately, this approach is error prone because MKL and FFTW3 exports the same (fftw3_*) symbols although the two APIs are not equivalent especially if OpenMP threads are used. More specifically, the fftw3 wrappers provided by MKL are not thread-safe hence the results are unpredictable due to race conditions if the user specifies fftw3-threads to link against MKL.
In a nutshell, the (highly) recommended procedure for Abinit9 should be:

- If you are linking against the true FFTW3 library, use “fftw3” or “fftw3-threads”
- If you are linking against MKL, use “dfti” or “dfti-threads”
- Don’t mix FFTW3 with MKL but use the MKL library for BLAS/LAPACK/FFTs (and optionally BLACS/SCALAPACK) On intel architecture, indeed, DFTI-MKL is significantly faster than FFTW3 so I don’t see the point in mixing the two libraries.

From the programmer's perspective, we need the -threads switch and the associated CPP variable so that we know that the FFT library supports threads. In the case of FFTW3, indeed, we need to initialize the threaded version and the API call must be enclosed between CPP options because this function is not available in FFTW3 sequential. Knowing that threads are supported is also useful to implement optimizations at runtime in fourwf when ndat > 1 Besides I think that the build system should issue a Warning if either fftw3-threads or dfti-threads are used without enable_openmp because there are sections in the Abinit FFT routines that are parallelized with OpenMP.

I hope I’ve clarified the reason why we need the -thread switch. Now let's have a look at the different FFT options available in the build system

```
# Supported libraries:
#
# * auto          : select library depending on build environment
#          (default)
# * custom        : bypass build-system checks
# * dfti          : native MKL FFT library
# * dfti- threads : threaded MKL FFT library
# * fftw3         : serial FFTW3 library
# * fftw3-mkl     : FFTW3 over MKL library
# * fftw3- threads : threaded FFTW3 library
# * fftw3-mpi     : MPI-parallel FFTW3 library
# * pfft          : MPI-parallel PFFT library (for maintainers only)
# * goedecker     : Abinit internal FFT
```

According to the previous discussion about threads, the four options (fftw3, fftw3-threads, dfti and dfti-threads) are needed.

fftw3-mkl, on the other hand, is redundant with dfti and can be removed.

As concerns fftw3-mpi: in principle there are routines in Abinit in which the MPI version of FFTW3 is used to transform densities or potentials. Note, however, that as far as I know these routines are not used in the production version. My version of fftalg 312 uses a routine inspired to Goedecker's algorithm for dense MPI-FFTs in which only the sequential fftw3 version and MPI_ALLTOALLV is needed.

Note that the most CPU-critical part in Abinit is represented by the FFT of the wavefunctions computed by fourwf when option == 2. Fourwf implements a rather sophisticated algorithm that performs two composite zero-padded FFTs to compute the FFT of \( \begin{pmatrix} v \\ u \end{pmatrix} \) starting from \( u(G) \). This algorithm is specific to ab-initio codes based on plane-waves and, obviously, neither FFTW3 nor DFTI-MKL provide an optimized implementation for this rather specific case.

Having said that, I think that also fftw3-mpi is redundant. If we want to maintain the possibility of calling the MPI version provided by FFTW3, we can just check at configure time whether the fftw3
libraries passed by the user provided the fftw_mpi_init symbol and then define HAVE_FFTW3_MPI

By the same token, there's little hope that a dense MPI-FFT provided by pfft will be able to beat our version of zero-padded MPI-FFT version implemented in fourwf. Perhaps, we can speedup fourdp but, again, this part is not the main bottleneck. Moreover, from what Jordan wrote, most of the new optimizations are now focusing on OpenMP-threads instead of MPI-FFT.

To summarize:

The following options are needed:

- dfti
- fftw3
- fftw3-threads

while fftw3-mkl can be removed. dfti-threads can also be removed because MKL manages threading through environment variables.

fftw3-mpi can be replaced by a simple check on the presence of fftw_mpi_init that defines HAVE_FFTW3_MPI

The pfft option may be used to activate a specialized routine for fourdp but pfft should also activate HAVE_FFTW3 so that we can use the zero-padded version in fourwf.

From:
https://wiki.abinit.org/ - Tips for ABINIT users and developers

Permanent link:

Last update: 2020/03/06 17:04