Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Enabling FFTE Library and FFTW3 Threading in Quantum ESPRESSO

Dušan Stanković*[a], Aleksandar Jović[a], Petar Jovanović[a], Dušan Vudragović[a], Vladimir Slavnić[a]

[a]*Institute of Physics Belgrade, Serbia*

Abstract

In this whitepaper we report work that was done on enabling support for FFTE library for Fast Fourier Transform in Quantum ESPRESSO, enabling threading for FFTW3 library already supported in Quantum ESPRESSO (only a serial version), benchmarking and comparing their performance with existing implementations of FFT in Quantum ESPRESSO.

1. Introduction

Quantum ESPRESSO (QE) [1] is an integrated suite of open-source codes for electronic structure calculations and materials modeling at the nanoscale. It is based on density-functional theory, plane waves and pseudopotentials. A large part of calculations performed in QE is related to the Fourier Transform. Most major hardware platforms are supported in Quantum ESPRESSO, along with their corresponding numerical libraries (such as IBM ESSL, SGI SCSL, NEC MathKeisan, Sun SunPerf), which can compute FFT among other things. Also, the FFTW and FFTW3 Open-source libraries for FFT are supported (FFTW is supplied with QE, so no external linking needs to be done). Parallelization in Quantum ESPRESSO is achieved by using MPI and OpenMP technologies, though only the internally supplied FFTW library supports the threaded and hybrid parallelization modes (both MPI and OpenMP at the same time).

In this project, Quantum ESPRESSO computing codes are extended to use the FFTE [2] numerical library, as well as threaded version of the FFTW3 [3] numerical library. The work on enabling these additional libraries has been motivated by excellent performance results of the FFTE described in Ref. [4], and by the expectation that the hybrid approach with FFTW3 in Quantum ESPRESSO will achieve better performance compared to the existing MPI implementation. Code development was done according to the Quantum ESPRESSO development manual [5], which defines guidelines regarding programming style (variable naming and capitalization, indentation style, use of automatic variables, use of pointers etc.) The latest version of Quantum ESPRESSO modified code is available at Ref. [6].

2. Code structure and modifications

The application is written in Fortran 90 and is parallelized using MPI and OpenMP. Quantum ESPRESSO code is divided in modules (for example PW and CP), with the code parts in which some more general-purpose computations and operations are performed (such as FFT computation, or time logging) separated in their own module.

The development of this project used QE version 5.0 (currently the last version) as a baseline, and was focused on the parts of the code responsible for FFT. Analysis of the QE code showed that all routines for performing FFT are located in `Modules/fft_scalar.f90` source file of the distribution. Routines for 1D, 2D and 3D FFT, defined in this file, serve as wrappers and invoke corresponding FFT routines of the above mentioned supported numerical libraries. Specification of the particular FFT library to be used is performed by conditional compilation. Using the pre-processor directives (`#ifdef, #elif, #endif` etc.), individual sections of the `Modules/fft_scalar.f90` file are compiled, depending on which parameter macros were defined during configuration. In the case when configuration process was successfully performed and at least one of the supported FFT libraries was available in the environment, matching parameter macros will be listed in the parameters section of the Makefile. For example, the `__FFTW3` macro parameter will be defined in the Makefile parameters section if the FFTW3

* Corresponding author. E-mail address: dusan.stankovic@ipb.ac.rs

numerical library is available on the system, and only the code that is used for invoking FFTW3 routine calls will be compiled.

2.1 Enabling FFTE library in Quantum ESPRESSO

As a first part of this project, we have extended Quantum ESPRESSO to enable the use of FFTE numerical library following the described conventions. We have used FFTE version 5.0, which was the latest version at the time. Because it does not provide a Makefile to automatically build a library from its source code, it is necessary to manually compile FFTE source files and create a library from generated object files. A new macro parameter **`__FFTE`** was defined, and all lines of source code that implement FFTE routine calls and supporting operations were put after the **`#elif defined __FFTE`** preprocessor directive. Also, the configure script was modified to automatically recognize and use FFTE library if it is found on the system path, or a path to the library is manually provided to configure, by using LIBDIRS variable. That means that the code which implements FFTE in Quantum ESPRESSO will be compiled and executed only if an FFTE library was found and no other FFT library is used in QE, which should be satisfied after a successful configure process. Variables needed to successfully initialize FFTE and execute its routines were introduced as to be easily distinguishable by their prefix (**`fftе_`**).

FFTE provides the routines for performing 1D, 2D and 3D Fourier transforms and supports MPI and OpenMP parallel technologies. But, because Quantum ESPRESSO uses its own decomposition of the data to perform 3D FFT as a combination of serial 1D and 2D transforms, performing MPI communication in-between FFT routine calls, only a serial versions of the FFTE routines were used. The reason for such behavior is mostly to avoid unnecessary transformation of sub-arrays of a large 3D grid, which already have zero values. Decomposition of the data and detailed explanation of 3D FFT computation in Quantum ESPRESSO can be found in [7]. A direct call to 3D FFT routine is performed in Quantum ESPRESSO only if it was not configured with parallel execution in mind, which was not of interest to us when working on this project, and was implemented only for completeness.

2.1.1 FFTE implementation details

FFTE routines for FFT execution serve both as routines which perform FFT on an array given as a parameter, and which perform necessary initialization before calling any computational routines. The operation that should be performed is selected by passing a parameter for FFT direction to the routine, which distinguishes between forward FFT, backward FFT and initialization call. Some FFTE routines store temporary work vectors internally, while some take the vector as a parameter and fill it with appropriate values if initialization was selected, or use it during computation otherwise. Because of that inconsistency, a temporary variable had to be introduced to Quantum ESPRESSO code when performing 1D transforms. This variable is named **`fftе_work`**, and it stores the values of work vectors between FFTE 1D calls.

In the FFTE documentation it is mentioned that the initialization has to be performed before calling corresponding routines for FFT computation, but it is not specified whether initialization can be reused, as long as dimensions of the array on which the transformation is computed stay the same. This is possible with other numerical libraries supported in Quantum ESPRESSO, so we tried to use the same pattern with FFTE. Calls to initialization routines were performed only if they have not been performed before, or if the dimensions of the FFT have changed with regard to the previous FFT call. This seemed to have worked for some tests, but on some other benchmarks we have obtained very strange and incorrect results. Because of such behavior and because of a lack of detailed documentation for FFTE that would explain initialization, we have resorted to calling initialization routines before every single call of FFTE routines for computation of the Fourier transform. This might be slightly suboptimal with regard to the performance (because with other numerical libraries initialization is performed only when absolutely needed), but it was necessary for correct execution.

Also, it should be mentioned that FFTE does not support the computation of many Fourier transforms in a single routine call. This can be important because in Quantum ESPRESSO, FFT routines are called for many 1D and 2D arrays that are consecutive in memory, and usually dimensions of these arrays are not very large individually. Because of that there is a slight overhead when using FFTE, because we need to call an FFT routine N times in order to transform N consecutive arrays, and every invocation of some routine has some overhead time which could instead be used for calculation. As an example, it can be seen in Ref. [8] how this is solved in FFTW3 library and how it helps gain performance in certain cases.

2.2 Enabling FFTW3 threading in Quantum ESPRESSO

As a second part of this project, we have implemented threading support for the FFTW3 library inside of Quantum ESPRESSO. The library is already supported in Quantum ESPRESSO, but only the serial version is used. Since QE supports execution in hybrid parallel mode (but allows only its internal FFTW library to be used when executing in hybrid), and since FFTW3 has both the reentrant thread-safe routines for FFT execution, and a support for natively threaded FFT library, we wanted to implement and test OpenMP threading with FFTW3 and measure its performance compared both to the pure MPI implementation and an existing hybrid FFTW implementation in QE. We have used FFTW3 version 3.3.2.

2.2.1 Threading modes in FFTW3 library

FFTW3 library supports threading in two variants:

- implicit, where an additional library named ***libfftw3_omp*** needs to be built, and is used to call FFTW3 routines that support execution in multi-threaded mode (threading implemented using OpenMP), and
- explicit, where the ordinary ***libfftw3*** is used, and its routines which perform FFT are called inside of the manually created OpenMP parallel region by many threads simultaneously (this is possible because FFT execution routines are thread-safe)

To implement both implicit and explicit threading of the FFTW3 library, we introduced two more macro parameters to enable conditional compilation. Implicit mode is enabled by defining ***__FFTW3_OMP_IMPL*** macro parameter inside ***Makefile***, and explicit mode is enabled by defining ***__FFTW3_OMP_EXPL*** macro parameter.

In order to use implicit threading, we had to call thread initialization routines before calling any FFTW3 routines for FFT planning and execution. After the thread initialization has been successfully performed, calls to planning and execution routines are the same as in a serial version, and the library routines execute in multi-threaded mode, so there is some overlapping between the code for implicit threading and the existing serial FFTW3 code. We also introduced some variables (names start with an ***fftw3_*** prefix), which are used during thread initialization.

A new section of the code had to be inserted in order to use explicit threading of the FFTW3 library. Aside from introducing an OpenMP parallel region, we had to modify the calls to the planning and execution routines of the FFTW3 library. In the serial version of the code already present in Quantum ESPRESSO, multiple calls for performing Fourier transform on adjacent arrays of same dimensions are aggregated to a single call, using the FFTW3 advanced interface as mentioned in the previous chapter. Performing multiple transforms like this is not suitable for threaded parallelization, because there is actually just a single call to an FFTW3 routine, which later expands to multiple transforms. Because of this, we had to manually split a single call with many transforms to many calls with a single transform in each.

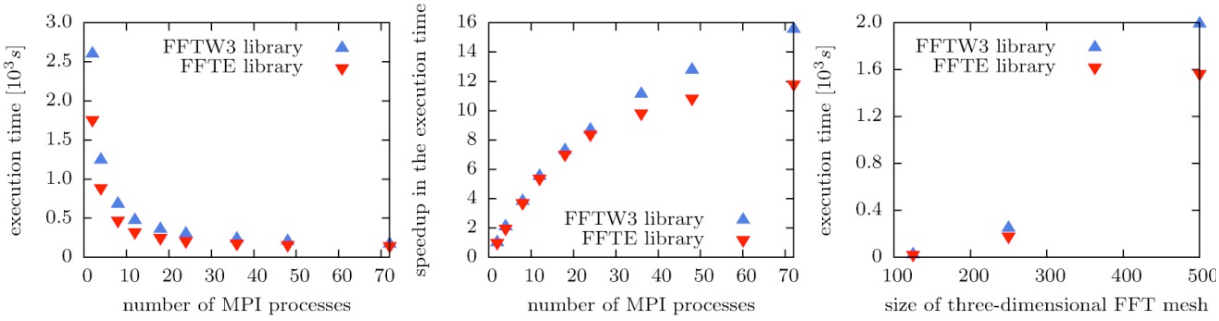
3. Performance and scaling tests

3.1 Performance of FFTE in Quantum ESPRESSO

We have tested the FFTE implementation in Quantum ESPRESSO on an AMD cluster with nodes containing 2 AMD Magny-Cours Opteron 6174 processors containing 12 cores each, with Infiniband interconnecting network, using the GNU compiler suite, and also on an Intel cluster with 2 Intel Xeon E5345 processors per node, with 4 cores each and Gigabit Ethernet interconnecting network, using the Intel compiler suite. As a basis for comparison we took a version of Quantum ESPRESSO with FFTW3 library used for FFT. Apart from the FFT computation, other parts of the QE code are the same. To configure Quantum ESPRESSO to use either FFTE or FFTW3 numerical library, we just invoked the configure script, passing the path to the desired library in LIBDIRS variable:

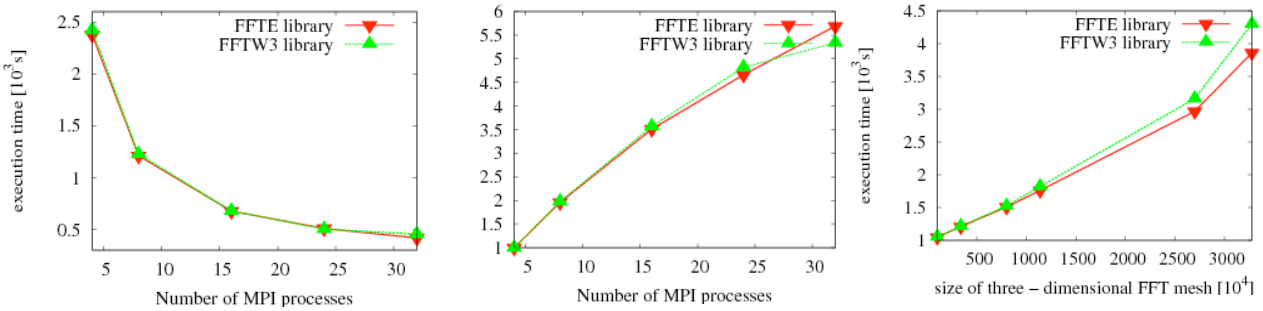
```
./configure LIBDIRS=/path/to/FFTE
./configure LIBDIRS=/path/to/FFTW3
```

The program was tested on benchmarks for PW and CP modules of Quantum ESPRESSO, first when the number of MPI processes was changed and then when the size of the grid in the input example was increased. For testing of the PW module, the FFTE library was built using gfortran version 4.1.2, with flag ***-O3***, and the FFTW3 library was built using gcc version 4.1.2, with flags ***-O3 -fomit-frame-pointer -fstrict-aliasing -fno-schedule-insns -ffast-math*** on the AMD cluster. The performance and scaling of the PW module and the performance for the case with variable input are shown in Figure 1. For testing of the CP module, the FFTE library was built using ifort version 11.1, with flag ***-fast***, and the FFTW3 library was built using icc version 11.1, with flag ***-O3*** on the Intel cluster. The performance and scaling of the CP module and the performance for the case with variable input are shown in Figure 2.



a) (b) (c)

Figure 1: Performance of the PW module of QE FFTE extension compared with the QE FFTW3 implementation: (a) Execution times of QE FFTW3/FFTE codes for different number of MPI processes; (b) Speedup in the execution time of QE FFTW3/FFTE codes as functions of a number of MPI processes; (c) QE FFTW3/FFTE execution times as functions of 3D FFT mesh size.



(a) (b) (c)

Figure 2: Performance of the CP module of QE FFTE extension compared with the QE FFTW3 implementation: (a) Execution times of QE FFTW3/FFTE codes for different number of MPI processes; (b) Speedup in the execution time of QE FFTW3/FFTE codes as functions of a number of MPI processes; (c) QE FFTW3/FFTE execution times as functions of 3D FFT mesh size.

From these figures it can be seen that the FFTE library slightly outperforms FFTW3 for different number of MPI processes, and that the difference diminishes as the number of MPI processes grows. Since the results are obtained using fixed three-dimensional FFT mesh (dimensions are 125x125x125), when the mesh is distributed over an increasing number of processors, the time each process spends in FFT routines is reduced, so the differences in performance of a particular library are less visible. This is also visible on figures in the middle that show speedup in the execution time over the same number of MPI processes. Although the speedup of the FFTW3 library is better (more evident for the PW case), we know that the execution time of the FFTE library is smaller overall, and even if in some point it becomes even to, or worse than the execution time of the FFTW3 library, due to the reduction of time spent in FFT routines as the size of the MPI world grows, difference in performance will be almost irrelevant. On the other hand, when a number of MPI processes is fixed and the size of the FFT mesh grows, as it can be seen on the third figure, each process gets more data to compute, and the difference in performance between the two libraries increases.

3.2 Performance of hybrid implementations of FFTW3 in Quantum ESPRESSO

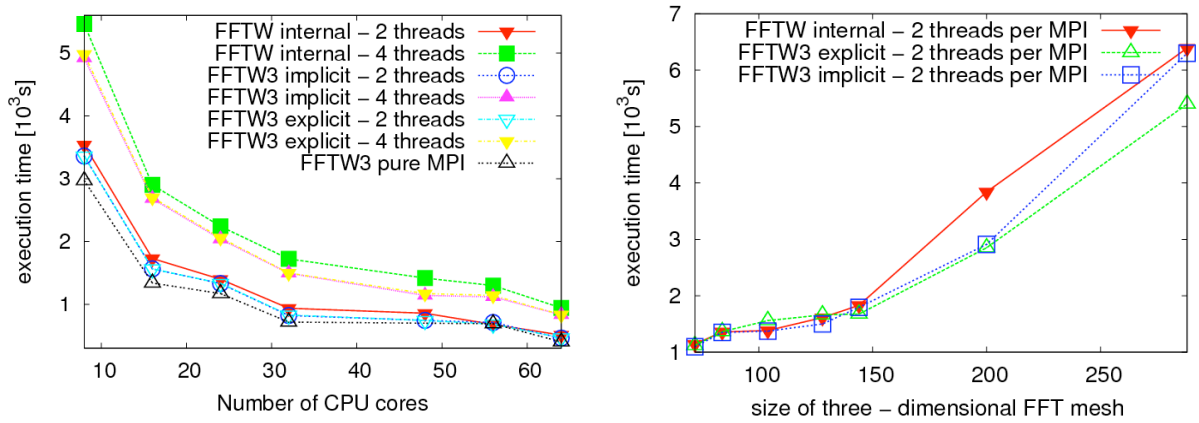
Hybrid implementations of the FFTW3 library in Quantum ESPRESSO were also tested on the Intel Xeon cluster mentioned in the previous chapter, using the Intel compiler suite. As a basis for comparison we took the hybrid version of Quantum ESPRESSO where the internally supplied FFTW library is used, and also a pure MPI version of the QE where FFTW3 library is used. Our goal was to show that hybrid version of the Quantum ESPRESSO when FFTW3 is used has better performance than when internal FFTW is used, and to test if hybridization of the FFT computation helps to gain performance benefits compared to the pure MPI version.

In order to enable support for threaded versions of the FFTW3 library it is necessary to manually edit the Makefile and add some parameter macros and libraries. When configured to use threads, Quantum ESPRESSO defaults to the internal FFTW library, even if some other library can be found on path. Because of that, it is necessary to first invoke the configuration of Quantum ESPRESSO with threading support, and then to change FFTW to FFTW3. This can be done by editing parts of the Makefile which control conditional compilation and linking of FFT libraries.

For implicit threading mode, a prerequisite is to have an FFTW3 threaded library installed (***`libfftw3_omp`***). If it is not present, then FFTW3 has to be reconfigured and rebuilt with threading support, before Quantum ESPRESSO is configured. Implicit threading mode can be enabled by defining ***`-D__FFTW3`*** (instead of ***`-D__FFTW`***) and ***`-D__FFTW3_OMP_IMPL`*** macro parameters in ***`DFLAGS`*** variable inside of the ***`make.sys`*** file. In addition, it is necessary to specify the path to the FFTW3 library

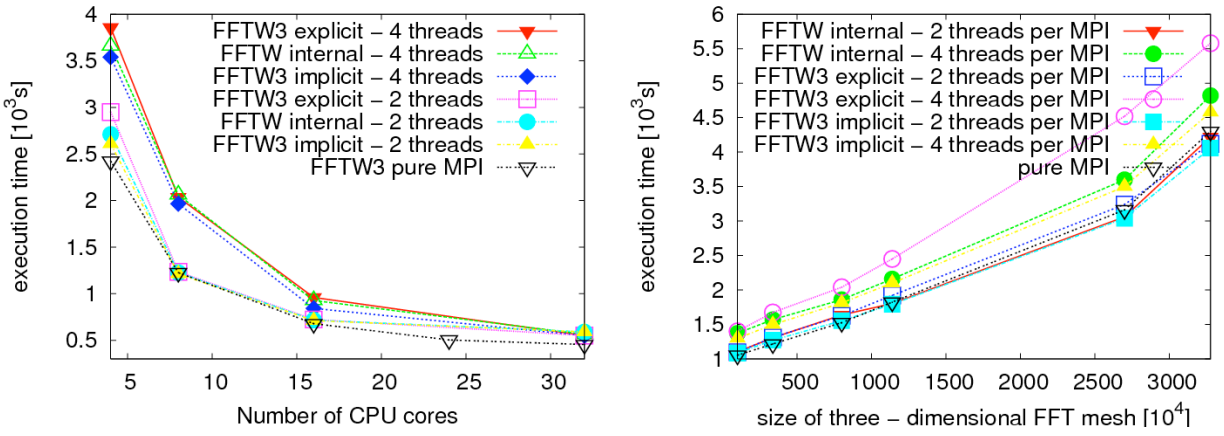
within FFT_FLAGS variable (***-L/path/to/fftw3***), as well as FFTW3 libraries to be linked with, in the same line (***-lfftw3_omp*** and ***-lfftw3***). Explicit mode can be enabled in a similar way, only adding ***-D__FFTW3_OMP_EXPL*** macro parameter instead of the one for implicit mode, and linking with just the serial FFTW3 library, instead of also linking with the threaded one.

Hybrid extension was also tested with PW and CP modules of Quantum ESPRESSO, with variable number of MPI processes and variable size of FFT mesh. Both FFTW3 library and Quantum ESPRESSO were compiled with the Intel compiler suite, version 11.1, and results are shown as a total execution time of benchmark inputs as a function of the number of MPI processes. We have tested the cases with 2 and 4 threads per single MPI process. The performance of the FFTW3 hybrid extension is shown along with the performances of the internally supplied FFTW library (labeled as internal) and pure MPI FFTW3 implementation on Figure 3 for PW module and Figure 4 for CP module. The total number of computing cores shown on figures is fixed at a certain point and is equal to the number of MPI processes times the number of threads per MPI process.



(a) (b)

Figure 3: Performance of the PW module of QE FFTW3 hybrid extension compared to internal FFTW and pure MPI implementation: (a) Execution times of QE FFTW3 implicit and explicit/internal FFTW/pure MPI codes for different number of MPI processes; (b) QE FFTW3 implicit and explicit / internal FFTW execution times as functions of 3D FFT mesh size.



(a) (b)

Figure 4: Performance of the CP module of QE FFTW3 hybrid extension compared to internal FFTW and pure MPI implementation: (a) Execution times of QE FFTW3 implicit and explicit/internal FFTW/pure MPI codes for different number of MPI processes; (b) QE FFTW3 implicit and explicit/internal FFTW execution times as functions of 3D FFT mesh size.

The figures show that both hybrid versions of FFTW3 implementation in Quantum ESPRESSO show better performance than the internal FFTW library used in hybrid mode in most cases. Also, both hybrid implementations have slightly worse performance than the pure MPI version. The overhead related to thread management is probably greater than are benefits of reduced MPI communication. The reduction in communication happens because for some fixed number of total cores, number of MPI processes decreases when programs are run in hybrid mode. This can also be seen if we compare cases for 2 and 4 threads per MPI process, where it is visible that the case with 4 threads per MPI process has significantly worse performance.

These observations agree with ones presented in Ref. [7] where threading of FFT computations in Quantum ESPRESSO was also investigated. It was shown there that threading does not provide benefits for performance in all cases, but instead in a few cases where the number of MPI processes was significantly large. Also Quantum ESPRESSO has a way to control parallelization on a level other than MPI (for example task groups, division of processes into pools etc.) which is related to a particular context (data structures in Quantum ESPRESSO inputs). These options were not implemented with hybrid parallelization in mind, so it is a bit harder to fine-tune Quantum ESPRESSO in order to achieve best performance when threading is used.

We would also like to mention that we haven't noticed any significant difference in performance of explicit and implicit

variants of threaded FFTW3 library that would help to select one that is performing better. In the results presented here, their performance is very similar. In some other tests we performed with different inputs and on different hardware, which are not presented here, performance of those 2 variants was varying. In some tests the explicit variant was faster, while in some others it was the other way around. Generally speaking, an advantage of the explicit mode of threading FFT is that the parallel region is created only once, and inside of it calls to many routines are made, contrary to an implicit mode, where parallel region is created for every call of an FFT routine. This helps to reduce overhead related to thread creation and synchronization. But, because we are using the FFTW3 advanced interface which enables us to wrap many Fourier transforms on different arrays into one routine call, it is probable that the implementation of the threaded FFTW3 library was aware of that, and that it successfully avoids creating a separate parallel region for the each FFT that is performed, when called in that way. So the difference in performance between explicit and implicit threading, if there is any, is due to some other factors and cannot be easily predicted for some test case.

Conclusions

We have implemented the extension of Quantum ESPRESSO to support the FFTE library for Fourier transforms in serial mode, and for FFTW3 library in threaded mode. The Quantum ESPRESSO FFT extension produced in this project shows better performance compared to the default QE FFT. In the case of the FFTE extension, this performance benefit could be significant when a large charge density mesh is required by the physical system. QE FFTW3 hybrid explicit and implicit extensions illustrate better performance compared to the internal Quantum ESPRESSO FFTW hybrid approach, but still worse than the pure MPI version. As evidenced in Ref. [7], this is probably because the overhead related to thread management outweighs the benefits of reduced MPI communication, up to a certain number of MPI processes. It is possible that in some other configurations, probably as the number of MPI processes grows even further, the hybrid version of Quantum ESPRESSO where FFTW3 library is used will have better performance than the pure MPI version.

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. FP7-261557. The work is achieved using the PRACE Research Infrastructure resource PARADOX at IPB in Serbia, and NIIFI SC in Hungary. We would also like to thank Filippo Spiga for helping us in understanding the configuration options for parallel running of Quantum ESPRESSO, so we could obtain sufficiently good performance with benchmarks we have used.

References

[1] Quantum ESPRESSO official web site
http://www.quantum-espresso.org/
[2] FFTE: A Fast Fourier Transform Package official web site
http://www.ffte.jp/
[3] M. Frigo, S. G. Johnson, "The Design and Implementation of FFTW3", Proceedings of the IEEE 93 (2005) 216
[4] A. Sunderland, S. Pickles, M. Nikolic, et al., “An Analysis of FFT Performance in PRACE Application Codes”, PRACE-1IP T7.5 Whitepaper (2012)
[5] Developer’s Manual for Quantum ESPRESSO (v. 5.0)
http://www.quantum-espresso.org/wp-content/uploads/Doc/developer_man.pdf
[6] QE-SCL, Quantum ESPRESSO extended with FFTE library and FFTW3 threading support
http://www.scl.rs/QE-SCL/
[7] F. Spiga, Implementing and Testing Mixed Parallel Programming Model into Quantum ESPRESSO, Science and Supercomputing in Europe – research highlights 2009, ISBN 978-88-86037-23-5, pp. 6
[8] FFTW3 advanced interface
http://www.fftw.org/doc/Advanced-Interface.html