Implementation and Benchmarking of New FFT Libraries in Quantum ESPRESSO

Dušan Stanković, Petar Jovanović, Aleksandar Jović, Vladimir Slavnić, Dušan Vudragović, and Antun Balaž

Scientific Computer Laboratory, Institute of Physics Belgrade, University of Belgrade, Pregrevica 118, 11080 Belgrade, Serbia
{dusan.stankovic,petar.jovanovic,aleksandar.jovic,vladimir.slavnic, dusan.vudragovic,antun.balaz}@ipb.ac.rs

Abstract. Quantum ESPRESSO (QE) software package allows electronic-structure calculations and materials modeling at the nanoscale, based on density-functional theory, plane waves, and pseudopotentials. It extensively uses Fast Fourier Transform (FFT) during all computations. In addition to the built-in FFT libraries, QE enables integration of newly developed FFT algorithms. Since Fastest Fourier Transform of the East (FFTE) library has shown performance comparable with the widely used and vendor-supplied libraries, the same behavior is foreseen in QE. In this paper we present FFTE-enabled and thread-enabled FFTW3 extensions of QE, together with benchmarking and performance results.

Keywords: FFT, Quantum ESPRESSO, multithreading, hybrid parallelism, OpenMP, MPI.

1 Introduction

Quantum Espresso 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 [1].

Fourier transformation is used in a large part in calculations performed in QE, so any gains in FFT performance would be positively reflected in the performance of the entire QE suite. Most major hardware platforms, along with their corresponding numerical libraries, are already supported in QE (such as IBM ESSL, Intel MKL, SGI SCSL and so on), which include routines for FFT calculations. Also, the open-source FFTW (version 2) and FFTW3 libraries [2] are supported.

Parallelization in Quantum ESPRESSO is achieved using MPI and OpenMP, and hybrid parallelism using both MPI and OpenMP together is currently supported only with the internally supplied FFTW library. The work on implementing the support for the open-source FFTE library was motivated by its performance results [3], so it was expected to show better performance than the open-source libraries already supported in QE. The work on implementing the support for hybrid FFTW3 library was considered because hybrid parallelism

M. Dulea et al. (eds.), *High-Perf. Comp. Infrastr. for South East Europe's Research Communities*, Modeling and Optimization in Science and Technologies 2,
DOI: 10.1007/978-3-319-01520-0_19,

is becoming more important, as computing nodes on modern HPC systems often comprise many CPU cores. Since the open-source FFTW3 library is widely used, and has both multi-threaded routines, and serial thread-safe routines, it was selected for implementation.

2 Quantum ESPRESSO Code Structure and Applied Modifications

Quantum ESPRESSO is written mostly in FORTRAN 90. It has a modular structure, with different modules for higher level domain specific calculations (for example, CP or PW modules), and also some general purpose parts which are then used in many other modules (for example FFT calculations or time logging).

The development of this project used QE 5.0 as a baseline, and was localized to the parts of the code responsible for FFT calculations. Analysis of the QE source code revealed that all the routines for performing FFT are located in a file named *fft_scalar.f90*. Routines for 1D, 2D and 3D FFT are defined in this file. They serve as wrappers and invoke corresponding routines of the aforementioned numerical libraries, where the actual computation is performed. Selecting which particular numerical library will be used is performed by conditional compilation, using preprocessor directives (such as `#ifdef, #elif, #endif` and so on). Whenever a numerical library supporting FFT is found during the configuration phase of the QE software package, a corresponding macro parameter is defined in the Makefile, and is used to select an appropriate compilation path. For example, when the FFTW3 library is used, a macro parameter named `__FFTW3` will be defined, and only the code where FFTW3 routines are called will be compiled.

2.1 Enabling FFTE Library in Quantum ESPRESSO

We have extended QE to utilize the FFTE numerical library for performing FFT in 1D, 2D or 3D. The version of FFTE used is 5.0, accessible on the website [3]. FFTE is written in Fortran, supports parallelism with MPI, OpenMP, or both when hybrid variant is used. Also, FFT transformations for up to 3 dimensions are supported. Code development was done according to Quantum ESPRESSO development manual [4], which defines guidelines regarding the programming style.

A new macro parameter named `__FFTE` was created, and used in parts of the source code whenever a FFTE routine is called, or some initialization is performed. The configure script was also modified so that the configuration process can recognize if the FFTE library is present on the system, whether on the system path, or in the path specified during configuration. If the library is found, the `__FFTE` macro parameter is added to the Makefile. Variables needed to initialize FFTE, or store data between execution of FFTE routines were introduced as to be easily distinguishable by their prefix (`fftе_`).

In Quantum ESPRESSO, an internal decomposition of the data is used to perform 3D FFT transforms as a combination of multiple calls to serial 1D and 2D FFT routines, which are divided among processes. MPI is used for communication and data exchange in-between these phases. The reason for this approach is to avoid performing unnecessary transforms of subsections of the large 3D grid which already have zero values, as this pattern is common in data sets used by QE. A more detailed explanation of this decomposition can be found in Ref. [5].

It should be mentioned that the FFTE library does not support computation on many Fourier Transforms (on different arrays), in a single routine call. This can have some impact on the performance, because in QE there are many calls to 1D and 2D routines needed to complete transform on the entire data set. Also, when using FFTE, an initialization routine needs to be called before each transform, which includes even more overhead during execution. Significant drops in performance were not observed during our testing, but these factors should be considered when using the FFTE library in other projects.

2.2 Enabling FFTW3 Threading in Quantum ESPRESSO

Second extension of QE is related to support of threading of the FFTW3 library, which would enable hybrid parallelism (when used combined with the MPI), since it is already supported in Quantum ESPRESSO. The FFTW3 library supports threading in two modes:

- implicit, where an additional library `libfftw3_omp` has to be installed; in this case, FFTW3 routines support multi-threaded execution internally, so they are called like the serial ones, and
- explicit, where serial routines are used, but are called from within multiple threads running in parallel; this is possible because routines for FFT execution are thread-safe.

The following pseudocode representation roughly shows how the two threading modes were implemented in Quantum ESPRESSO (for the implicit mode, a single internally threaded routine call performs `nsl` transforms on arrays with length of `dim_z`, and for the explicit mode each routine call is serial):

- implicit

```
fftw_execute_many_dft(fw_plan, c, cout, nsl, dim_z)
```

- explicit

```
#pragma omp for
for i=1 to nsl
  offset=(i-1)*dim_z
  fftw_execute_dft(fw_plan, c[offset], cout[offset])
end for
```

The FFTW3 library supports reusing of plans, and also supports calculation of many transforms within a single routine call. This allows greater flexibility when using multiple transforms, and is optimal in terms of performance. More details on this can be found in Ref. [6].

In order to use implicit threading, FFTW3 thread initialization routines had to be called before calling any FFTW3 routines for FFT planning and execution. After the thread initialization has been successfully performed, the code for serial version can be reused, and threading is done automatically in the library routines.

With explicit threading, some modifications had to be made with the code. Because in the serial version many 1D or 2D transforms are aggregated in a single call for efficiency, execution had to be split into separate routine calls for each transform. This way, we actually had many routine calls, which can then be called from parallel threads. An OpenMP `parallel for` region was inserted, where in each iteration of the loop, FFT is performed on a separate sub-array. Since these routines are executed in parallel, and there are no data dependencies between loop iterations, this approach could be applied successfully.

3 Performance Tests

Here we will present performance tests done to compare newly supported FFTE library, and also performance of threaded FFTW3 library. Benchmarks were performed so that the performance was compared to most similar numerical libraries already supported in Quantum ESPRESSO.

3.1 FFTE Performance

We have tested Quantum ESPRESSO with enabled FFTE library, and compared it with the FFTW3 library that is already supported. These tests show only performances of serial libraries, since threaded FFTE was not implemented (because it wasn't always reliable when built with some compilers).

The cluster used for testing is made of nodes containing two AMD Magny-Cours Opteron 6174 processors, with 12 cores each. Nodes are connected via Infiniband network. The GCC compiler suite [7] was used in testing on this cluster. Our implementation was tested on benchmarks for PW module of Quantum ESPRESSO. FFTE Code was compiled with gfortran, version 4.1.2 with flags `-O3`, and the FFTW3 library was compiled with gcc, version 4.1.2 with flags

```
-O3 -fomit-frame-pointer -fstrict-aliasing -fno-schedule-insns
-ffast-math.
```

For the first test, up to 6 computing nodes were used (up to 72 processes). Execution times and scaling of the PW module are shown in Figure 1 for the case when the number of MPI processes is increased, and in Figure 2 when the problem size is increased, and the number of MPI processes stays constant (24 MPI processes were used in this test).

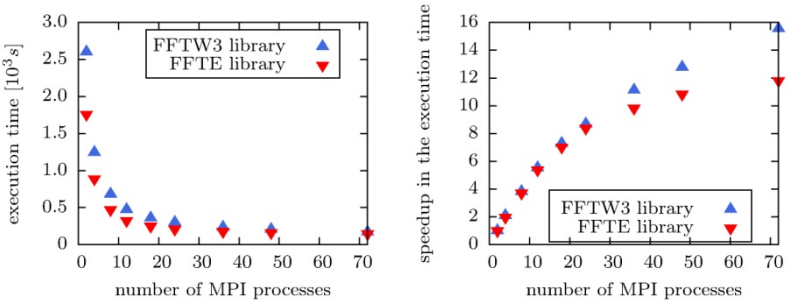

Fig. 1. Performance of the PW module of QE FFTE extension compared with the QE FFTW3 implementation: (left) Execution times of QE FFTW3/FFTE codes for different number of MPI processes; (right) Speedup in the execution time of QE FFTW3/FFTE codes as a function of a number of MPI processes (execution time on 1 MPI process used as a baseline)

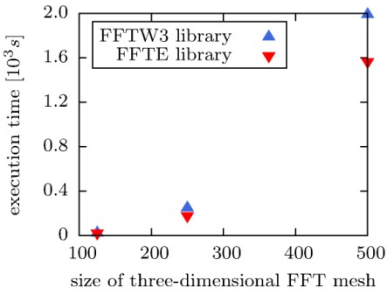

Fig. 2. Performance of the PW module of QE FFTE extension compared with the QE FFTW3 implementation: execution times as a function of 3D FFT mesh size

It can be seen that the FFTE library slightly outperforms FFTW3 in both cases (execution times are lower for the FFTE). The gap in performance grows as the size of the problem grows, so the FFTE seems suitable for large test cases. The difference in performance that is related to the problem size is also exhibited in the test case with the increasing number of MPI processes. As the number of MPI processes grows, each process gets less and less data to compute, and the difference in execution time diminishes. Because of this, the FFTE library shows worse speedup than the FFTW3.

3.2 FFTW3 Threaded Performance

For the performance testing of the threaded FFTW3 library, an FFTW (version 2) library internally supplied with Quantum ESPRESSO was selected for comparison. This was done because it was the only library supporting threading in

the hybrid mode (when used together with the MPI), and is also open-source and widely available.

Implementation of threaded FFTW3 was tested on a cluster with Intel Xeon processors, with two quad-core CPUs per node, and with Gigabit Ethernet interconnecting network. Library code was compiled with the Intel's icc compiler version 11.1 with the `-O3` flag, and Intel's ifortran was used for compilation of Quantum ESPRESSO.

Hybrid extension of the FFTW3 library was also tested with benchmarks for the PW module of QE. Tests were conducted again in the similar way, increasing the number of CPU cores in one case, and increasing input grid size in another. Configurations of 2 and 4 threads per MPI process were used, and also compared to the pure MPI case. Both threading variants (implicit and explicit) were tested with the FFTW3 library, and its performance is shown along with the internally supplied FFTW library (labeled as FFTW internal) in Figures 3 and 4. Total number of computing cores at some point is fixed, and is equal to a number of MPI processes times the number of threads per MPI process.

From this we see that both threading variants implemented for FFTW3 outperform the internal FFTW when executed with hybrid parallelism for most cases. Although, both threaded libraries are still slower than the pure MPI version. This is probably due to the fact that for the type of input data used with Quantum ESPRESSO, the overhead related to the thread management is probably greater than benefits of reduced MPI communication. Evidence for this are runs with four threads per MPI process, where performance gets significantly worse.

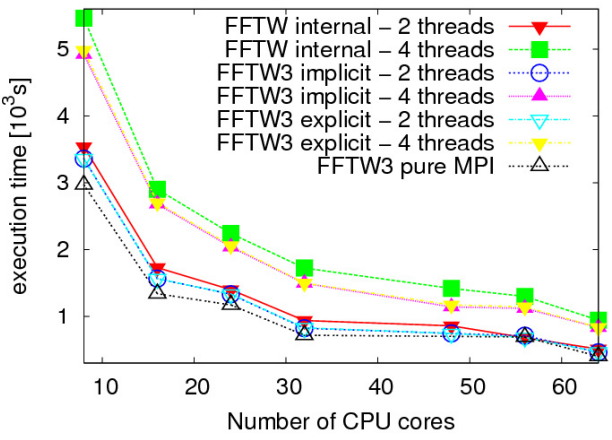

Fig. 3. Performance of the PW module of QE FFTW3 threaded extensions compared with the internal QE FFTW hybrid implementation and pure MPI FFTW3 implementation: Execution times of QE FFTW3 implicit and explicit/internal FFTW/pure MPI codes for different number of MPI processes

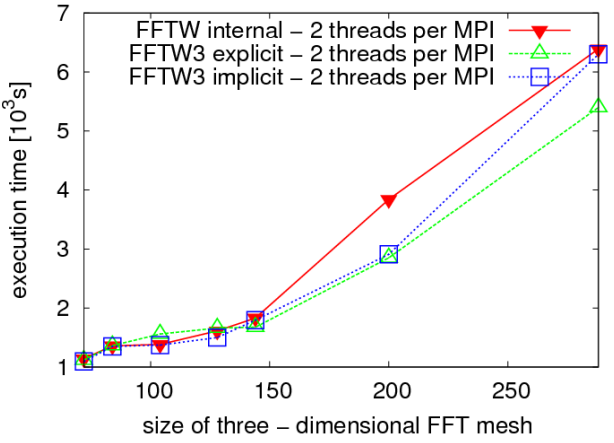

Fig. 4. Performance of the PW module of QE FFTW3 threaded extensions compared with the internal QE FFTW hybrid implementation and pure MPI FFTW3 implementation: QE FFTW3 implicit and explicit / internal FFTW execution times as functions of 3D FFT mesh size.

These results agree with what was presented in Ref. [5], where similar thing was investigated, and was shown that threading does not increase performance in all cases. Better performance was observed only in some cases where the number of MPI processes was significantly large. Also, Quantum ESPRESSO has other ways to control parallelism in software (for example, task groups, pools of processes, etc.) which is related to a particular input data set. Because these options were not primarily designed with hybrid parallelism in mind, it is not easy to fine tune Quantum ESPRESSO to achieve optimal performance when threading is used.

It is also worth mentioning that no significant difference in performance between implicit and explicit variants of FFTW3 threading was noticed. Looking at how threading is implemented in those two cases, an advantage of the explicit mode is that the OpenMP parallel region is created only once, and inside of it there are calls to many routines where FFT is computed. This should be optimal with regards to the overhead related to thread creation and synchronization. On the other hand, when using implicit threading, a new OpenMP parallel region has to be created with every routine call. However, because an advanced FFTW3 interface is used with implicit threading mode, it allows many transforms on different arrays to be aggregated in a single routine call from FORTRAN. It is possible that the native implementation of FFTW3 threaded library is aware of that, and that it successfully avoids unnecessary creation of parallel regions for each separate Fourier Transform.

4 Conclusions

In this project two extensions to Quantum ESPRESSO were implemented: the support for FFTE library for computing Fourier Transform in the serial mode, as well as the FFTW3 library in threaded mode. These extensions showed better performance compared to default QE libraries (open-source FFTW version 2 and 3 were selected for comparison). In the case of the FFTE library, performance increase could be significant when the large charge density mesh is requested for the simulation of a physical system. Both the explicit and implicit variants of FFTW3 threading showed better performance compared to internally supplied FFTW (version 2) when tested in hybrid configuration (two and four threads per MPI process), and while still not faster than the pure MPI version, should be considered when there is a need for hybrid parallelism. It is expected that a much larger problem size and more CPU cores are needed in order to get satisfactory performance of the hybrid FFTW3, which can match, or even surpass the performance of the pure MPI version.

Acknowledgements. Numerical results were obtained on the PARADOX cluster at the Scientific Computing Laboratory of the Institute of Physics Belgrade and on the NIIFI SC in Hungary. We acknowledge the support by the Serbian Ministry of Education, Science and Technological Development under projects No. ON171017 and III43007, and by the European Commission under FP7 projects HP-SEE, PRACE-2IP, PRACE-3IP and EGI-InSPIRE.

References

1. Quantum ESPRESSO official web site, `http://www.quantum-espresso.org`
2. Frigo, M., Johnson, S.G.: The Design and Implementation of FFTW3. Proceedings of the IEEE 93, 216 (2005)
3. FFTE: A Fast Fourier Transform package, `http://www.ffte.jp/`
4. Developer's Manual for Quantum ESPRESSO, `http://www.quantum-espresso.org/?page_id=47`
5. Spiga, F.: Implementing and Testing Mixed Parallel Programming Model into Quantum ESPRESSO. In: Science and Supercomputing in Europe - research highlights 2009, CINECA Consorzio Interuniversitario, Bologna (2010)
6. FFTW3 advanced interface, `http://www.fftw.org/doc/Advanced-Interface.html`
7. GCC compiler suite, `http://gcc.gnu.org/`