OpenFFT Version 1.2 User Guide

Truong Vinh Truong Duy and Taisuke Ozaki

April 5, 2015

Contents

1 Introduction

OpenFFT is an open source parallel package for computing multi-dimensional Fast Fourier Transforms (3-D and 4-D FFTs) of both real and complex numbers of arbitrary input size. It originates from OpenMX (Open source package for Material eXplorer, http://www.openmx-square.org/). OpenFFT adopts a communication-optimal domain decomposition method that is adaptive and capable of localizing data when transposing from one dimension to another for reducing the total volume of communication [1, 2]. It is written in C and MPI, with support for Fortran through the Fortran interface, and employs FFTW3 for computing 1-D FFTs.

OpenFFT is developed by Truong Vinh Truong Duy and Taisuke Ozaki at the University of Tokyo.

2 Features

- Domain decomposition method: OpenFFT adopts a decomposition method that is capable of localizing data when transposing from one dimension to another for reducing the total volume of communication. Also, the decomposition is adaptive, and can automatically switch between 1-D, 2-D, and 3-D (for 4-D FFTs) depending on the number of processes and data size. Please refer to the publications for detail.
- Support for fast parallel complex-to-complex and real-to-complex transforms of 3-D FFTs and 4-D FFTs with arbitrary input size.
- Tuning of communication with an auto-tuning feature.
- Support for hybrid MPI/OpenMP execution.
- Portable, tested on various general-purpose Linux clusters and popular supercomputers, including Cray XC30, SGI Altix UV1000, SGI InfiniBand cluster, FX10, and the K computer.
- Written in C and MPI, with support for Fortran through the Fortran interface.
- Open source package, released under the GNU General Public License (GPL).

3 Download

OpenFFT version 1.2 is the latest vesion of OpenFFT, and is downloadable from http://www.openmx-square.org/openfft/.

RELEASE NOTES

- OpenFFT version 1.2 (March 31, 2015)
  - Download: http://www.openmx-square.org/openfft/openfft1.2.tar.gz.
    Manual: http://www.openmx-square.org/openfft/manual1.2.html.
  - Addition of parallel 4-D FFTs with openfft_init_c2c_4d() and openfft_exec_c2c_4d() for initialization and execution, respectively.
  - Addition of hybrid MPI/OpenMP execution.
- OpenFFT version 1.1 (November 11, 2014)

  - Download: http://www.openmx-square.org/openfft/openfft1.1.tar.gz. Manual: http://www.openmx-square.org/openfft/manual1.1.html.
  - The c2c interface is changed from openfft_initialize() and openfft_execute() to openfft_init_c2c_3d() and openfft_exec_c2c_3d() for initialization and execution, respectively.
  - Addition of the r2c interface comprised of openfft_init_r2c_3d() and openfft_exec_r2c_3d().
  - Addition of the tuning of communication to the c2c and r2c interfaces.

- OpenFFT version 1.0 (August 23, 2013)
  - Download: http://www.openmx-square.org/openfft/openfft1.0.tar.gz. Manual: http://www.openmx-square.org/openfft/manual1.0.html.

4 Installation

Requirements: OpenFFT requires FFTW3 (or FFTW3 wrappers such as those provided by the Intel MKL library), a C compiler, and an MPI library. Fortran users will also need a Fortran compiler to compile the Fortran sample programs.

1. Step 1: Download and install FFTW3. Assume that FFTW3 is installed in /opt/fftw3. Those who already have the Intel MKL library or FFTW3 wrappers installed can skip this step.

2. Step 2: Download and extract the OpenFFT tarball. Assume that OpenFFT is extracted to /opt/openfft1.2.

3. Step 3: Modify CC (the C compiler) and LIB (the library path to FFTW3) in makefile in the root folder of OpenFFT to reflect your environment. Fortran users also need to specify FC (the Fortran compiler) to compile the sample programs. Samples of CC (and FC) and LIB in several environments are given in makefile.

   ```
   CC = mpicc -O3 -openmp -I/opt/fftw3/include -I./include
   LIB = -L/opt/fftw3/lib -lfftw3
   FC = mpif90 -O3 -openmp -I/opt/fftw3/include -I./include
   ```

4. Step 4: Issue the make command to compile and install the OpenFFT library. The library will be made available at /opt/openfft1.2/lib/libopenfft.a if successful.

5. Step 5: Link the OpenFFT library to compile a user program.

```
mpicc -O3 -openmp -o userprogram userprogram.c -I/opt/fftw3/include
-I/opt/openfft1.1/include -L/opt/fftw3/lib -lfftw3
-L/opt/openfft1.1/lib -lopenfft
mpif90 -O3 -openmp -o userprogram userprogram.f90 -I/opt/fftw3/include
-I/opt/openfft1.1/include -L/opt/fftw3/lib -lfftw3
-L/opt/openfft1.1/lib -lopenfft
```

5 Directory Structure

Directory structure of OpenFFT1.2 is as follows:

- toplevel: this is where makefile is located, as well as README.
- source: core source files of the package.
  - openfft_init_c2c_3d.c: initialization of complex-to-complex 3-D transforms.
  - openfft_init_r2c_3d.c: initialization of real-to-complex 3-D transforms.
  - openfft_init_c2c_4d.c: initialization of complex-to-complex 4-D transforms.
  - openfft_exec_c2c_3d.c: execution of complex-to-complex 3-D transforms.
  - openfft_exec_r2c_3d.c: execution of real-to-complex 3-D transforms.
  - openfft_exec_c2c_4d.c: execution of complex-to-complex 4-D transforms.
  - openfft_finalize.c: finalization of transforms.
  - openfft_dtime.c: built-in time measurement.
- include: C and Fortran header files.
- lib: the library file is installed here if successful.
- samples: sample programs for illustrating how to use OpenFFT.
  - C: C sample programs.
  - FORTRAN: Fortran sample programs.
- doc: documents on the website of OpenFFT.

6 Sample Programs

- C sample programs:
  - check_c2c_3d.c: This program illustrates how to use the c2c 3-D interface. It can be executed with an arbitrary number of processes. Its input and output should match the corresponding values in check_c2c_3d.dat. This program does not require any input parameter.
  - check_r2c_3d.c: This program illustrates how to use the r2c 3-D interface. It can be executed with an arbitrary number of processes. Its input and output should match the corresponding values in check_r2c_3d.dat. This program does not require any input parameter.
  - check_c2c_4d.c: This program illustrates how to use the c2c 4-D interface. It can be executed with an arbitrary number of processes. Its input and output should match the corresponding values in check_c2c_4d.din and check_c2c_4d.dout. This program does not require any input parameter.
  - timing_c2c_3d.c: This program is used for benchmarking performance of the c2c 3-D interface with timing and GFLOPS results. It can be executed with an arbitrary number of processes. Time is measured by MPI_Wtime(). A numeric input parameter can be provided for specifying the size of the 3 dimensions. If no input parameter is provided, it will be executed with a default size of 128^3 data points.
  - timing_r2c_3d.c: This program is used for benchmarking performance of the r2c 3-D interface with timing and GFLOPS results. It can be executed with an arbitrary number of processes. Time is measured by MPI_Wtime(). A numeric input parameter can be provided for specifying the size of the 3 dimensions. If no input parameter is provided, it will be executed with a default size of 128^3 data points.
  - timing_c2c_4d.c: This program is used for benchmarking performance of the c2c 4-D interface with timing and GFLOPS results. It can be executed with an arbitrary number of processes. Time is measured by MPI_Wtime(). A numeric input parameter can be provided for specifying the size of the 4 dimensions. If no input parameter is provided, it will be executed with a default size of 32^4 data points.

- breaktime_c2c_3d.c: This program is used for benchmarking performance of the c2c 3-D interface with timing result broken down into several parts and GFLOPS. It can be executed with an arbitrary number of processes. Time is measured by the built-in time measurement function. A numeric input parameter can be provided for specifying the size of the 3 dimensions. If no input parameter is provided, it will be executed with a default size of 128^3 data points. Please note that the timing breakdown can only be correctly done with the communication pattern number 6, as other patterns may feature communication and computation overlap.
- breaktime_r2c_3d.c: This program is used for benchmarking performance of the r2c 3-D interface with timing result broken down into several parts and GFLOPS. It can be executed with an arbitrary number of processes. Time is measured by the built-in time measurement function. A numeric input parameter can be provided for specifying the size of the 3 dimensions. If no input parameter is provided, it will be executed with a default size of 128^3 data points. Please note that the timing breakdown can only be correctly done with the communication pattern number 6, as other patterns may feature communication and computation overlap.
- breaktime_c2c_4d.c: This program is used for benchmarking performance of the c2c 4-D interface with timing result broken down into several parts and GFLOPS. It can be executed with an arbitrary number of processes. Time is measured by the built-in time measurement function. A numeric input parameter can be provided for specifying the size of the 4 dimensions. If no input parameter is provided, it will be executed with a default size of 32^4 data points. Please note that the timing breakdown can only be correctly done with the communication pattern number 6, as other patterns may feature communication and computation overlap.

- Fortran sample programs:
  - check_c2c_3d.f90: This program illustrates how to use the c2c 3-D interface. It can be executed with an arbitrary number of processes. Its input and output should match the corresponding values in check_c2c_3d.dat. This program does not require any input parameter.

- check_r2c_3d.f90: This program illustrates how to use the r2c 3-D interface. It can be executed with an arbitrary number of processes. Its input and output should match the corresponding values in check_r2c_3d.dat. This program does not require any input parameter.
- timing_c2c_3d.f90: This program is used for benchmarking performance of the c2c 3-D interface with timing and GFLOPS results. It can be executed with an arbitrary number of processes. Time is measured by MPI_Wtime(). A numeric input parameter can be provided for specifying the size of the 3 dimensions. If no input parameter is provided, it will be executed with a default size of 128^3 data points.
- timing_r2c_3d.f90: This program is used for benchmarking performance of the r2c 3-D interface with timing and GFLOPS results. It can be executed with an arbitrary number of processes. Time is measured by MPI_Wtime(). A numeric input parameter can be provided for specifying the size of the 3 dimensions. If no input parameter is provided, it will be executed with a default size of 128^3 data points.

7 Domain Decomposition

OpenFFT adopts a decomposition method that is capable of localizing data when transposing from one dimension to another to reduce the total volume of communication. Also, the decomposition is adaptive, and automatically switches between 1-D, 2-D, and 3-D (for 4-D FFTs) depending on the number of processes and data size. For example, with 3-D FFTs, OpenFFT decomposes in the order of abc, cab, and cba for performing the 1-D FFTs along the c-, b-, and a-axes, respectively (Fig. 1). Please refer to [1] for detail. Other publications [2, 3, 4, 5, 6, 7] may also be useful.

8 Tuning of Communication

OpenFFT implements a number of communication patterns that can be selected manually by users or automatically by the auto-tuning feature when initializing with openfft_init_c2c_3d(), openfft_init_r2c_3d(), or openfft_init_c2c_4d() . The communication patterns available are:

- 0: auto-tuning of communication, where OpenFFT automatically per-

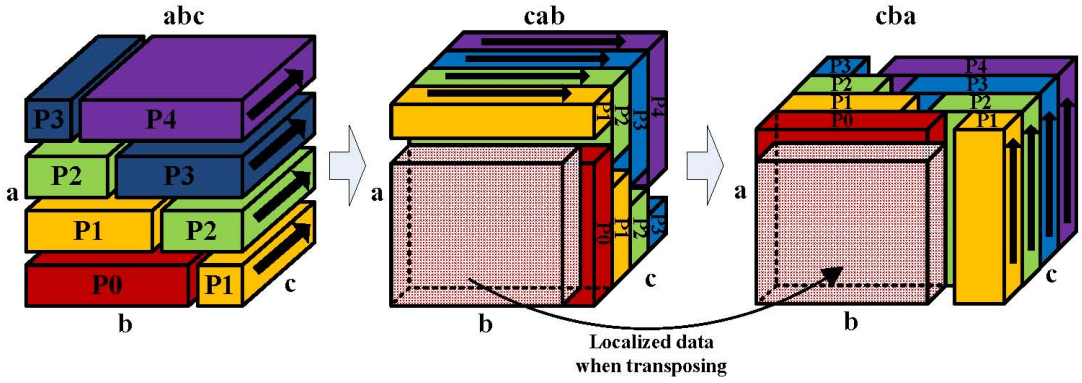

Figure 1: Domain decomposition.

forms tests with all of the following patterns and picks the best performer in run time (recommended for high performance).

- 1: MPI_Alltoallv.
- 2: MPI_Isend and MPI_Irecv within sub-groups of 256 processes.
- 3: MPI_Isend and MPI_Irecv with communication-computation overlap.
- 4: MPI_Isend and MPI_Irecv within sub-groups of 256 processes with communication-computation overlap.
- 5: MPI_Sendrecv.
- 6: MPI_Isend and MPI_Irecv.
- Others: default communication, which is 3.

9 Calling OpenFFT from a C User Program

Please refer to the C sample programs which illustrate how to call OpenFFT from a C user program. Basically, it involves several steps as follows.

1. Step 1: Include the OpenFFT header file, openfft.h, in the program.

```
#include <openfft.h>
```

2. Step 2: Initialize OpenFFT by calling openfft_init_c2c_3d() for the c2c 3-D interface, openfft_init_r2c_3d() for the r2c 3-D interface, or openfft_init_c2c_4d() for the c2c 4-D interface.

```
openfft_init_c2c_3d(N1,N2,N3,
                    &My_Max_NumGrid,&My_NumGrid_In,My_Index_In,
                    &My_NumGrid_Out,My_Index_Out,
                    offt_measure,measure_time,print_memory);
```

OR

```
openfft_init_r2c_3d(N1,N2,N3,
                    &My_Max_NumGrid,&My_NumGrid_In,My_Index_In,
                    &My_NumGrid_Out,My_Index_Out,
                    offt_measure,measure_time,print_memory);
```

OR

```
openfft_init_c2c_4d(N1,N2,N3,N4,
                    &My_Max_NumGrid,&My_NumGrid_In,My_Index_In,
                    &My_NumGrid_Out,My_Index_Out,
                    offt_measure,measure_time,print_memory);
```

   - Input:
     - 3 dimensions of data: N1, N2, N3 for 3-D FFTs, or 4 dimensions of data: N1, N2, N3, N4 for 4-D FFTs.
     - offt_measure for the tuning of communication (see Tuning of Communication, default 0).
     - measure_time for the built-in time measurement function and print_memory for printing memory usage (0: disabled (default), 1: enabled).
   - Output: arrays allocated and variables initialized.
     - My_Max_NumGrid: the maximum number of grid points allocated to a process, used for allocating local arrays.
     - My_NumGrid_In: the number of grid points allocated to a process upon starting.

- My_Index_In: the 6 or 8 indexes of grid points allocated to a process upon starting for 3-D and 4-D FFTs, respectively.
- My_NumGrid_Out: the number of grid points allocated to a process upon finishing.
- My_Index_Out: the 6 or 8 indexes of grid points allocated to a process upon finishing for 3-D and 4-D FFTs, respectively.

3. Step 3: After openfft_init_c2c_3d(), openfft_init_r2c_3d(), or openfft_init_c2c_4d() is called, important variables are initialized, and can be used for allocating and initializing local input and output data arrays.

   Allocate the local input and output data arrays based on My_Max_NumGrid, which is the maximum number of grid points allocated to a process during the transformation.

```
input  = (dcomplex*)malloc(sizeof(dcomplex)*My_Max_NumGrid);
output = (dcomplex*)malloc(sizeof(dcomplex)*My_Max_NumGrid);
```

   OR

```
input  = (double*)malloc(sizeof(double)*My_Max_NumGrid);
output = (dcomplex*)malloc(sizeof(dcomplex)*My_Max_NumGrid);
```

   Initialize the local input array from the global input array. A process is allocated (My_NumGrid_In) grid points continuously from AasBbsCcs to AaeBbeCce of the 3-D global array for 3-D FFTs, where:

```
as = My_Index_In[0];
bs = My_Index_In[1];
cs = My_Index_In[2];
ae = My_Index_In[3];
be = My_Index_In[4];
ce = My_Index_In[5];
```

   For 4-D FFTs, a process is allocated (My_NumGrid_In) grid points continuously from AasBbsCcsDds to AaeBbeCceDde of the 4-D global array, where:

```
as = My_Index_In[0];
bs = My_Index_In[1];
cs = My_Index_In[2];
ds = My_Index_In[3];
ae = My_Index_In[4];
be = My_Index_In[5];
ce = My_Index_In[6];
de = My_Index_In[7];
```

4. Step 4: Call openfft_exec_c2c_3d(), openfft_exec_r2c_3d(), or openfft_exec_c2c_4d() to transform input to output.

   ```
   openfft_exec_c2c_3d(input, output);
   ```

   OR

   ```
   openfft_exec_r2c_3d(input, output);
   ```

   OR

   ```
   openfft_exec_c2c_4d(input, output);
   ```

5. Step 5: Obtain the result stored in the local output array. Upon exiting, a process is allocated (My_NumGrid_Out) grid points continuously from CcsBbsAas to CceBbeAae of the 3-D global array for 3-D FFTs, where:

   ```
   cs = My_Index_Out[0];
   bs = My_Index_Out[1];
   as = My_Index_Out[2];
   ce = My_Index_Out[3];
   be = My_Index_Out[4];
   ae = My_Index_Out[5];
   ```

   For 4-D FFTs, a process is allocated (My_NumGrid_Out) grid points continuously from DdsCcsBbsAas to DdeCceBbeAae of the 4-D global array, where:

```
ds = My_Index_Out[0];
cs = My_Index_Out[1];
bs = My_Index_Out[2];
as = My_Index_Out[3];
de = My_Index_Out[4];
ce = My_Index_Out[5];
be = My_Index_Out[6];
ae = My_Index_Out[7];
```

6. Step 6: Finalize the calculation by calling openfft_finalize().

```
openfft_finalize();
```

10 Calling OpenFFT from a Fortran User Program

Please refer to the Fortran sample programs which illustrate how to call OpenFFT from a Fortran user program. Basically, it is similar to calling from C, except for the indexes that must be incremented by 1.

1. Step 1: Include the Fortran interface and the standard iso_c_binding module for defining the equivalents of C types (integer(C_INT) for int, real(C_DOUBLE) for double, complex(C_DOUBLE_COMPLEX) for dcomplex, etc.).

```
use, intrinsic :: iso_c_binding
include 'openfft.fi'
```

2. Step 2: Initialize OpenFFT by calling openfft_init_c2c_3d() for the c2c 3-D interface, openfft_init_r2c_3d() for the r2c 3-D interface, or openfft_init_c2c_4d() for the c2c 4-D interface .

```
openfft_init_c2c_3d(%VAL(N1),%VAL(N2),%VAL(N3),&
       My_Max_NumGrid,My_NumGrid_In,My_Index_In,&
       My_NumGrid_Out,My_Index_Out,&
       %VAL(offt_measure),%VAL(measure_time),%VAL(print_memory))
```

OR

```
openfft_init_r2c_3d(%VAL(N1),%VAL(N2),%VAL(N3),&
       My_Max_NumGrid,My_NumGrid_In,My_Index_In,&
       My_NumGrid_Out,My_Index_Out,&
       %VAL(offt_measure),%VAL(measure_time),%VAL(print_memory))
```

OR

```
openfft_init_c2c_4d(%VAL(N1),%VAL(N2),%VAL(N3),%VAL(N4),&
       My_Max_NumGrid,My_NumGrid_In,My_Index_In,&
       My_NumGrid_Out,My_Index_Out,&
       %VAL(offt_measure),%VAL(measure_time),%VAL(print_memory))
```

- Input:
  - 3 dimensions of data: N1, N2, N3 for 3-D FFTs, or 4 dimensions of data: N1, N2, N3, N4 for 4-D FFTs.
  - offt_measure for the tuning of communication (see Tuning of Communication, default 0).
  - measure_time for the built-in time measurement function and print_memory for printing memory usage (0: disabled (default), 1: enabled).
- Output: arrays allocated and variables initialized.
  - My_Max_NumGrid: the maximum number of grid points allocated to a process, used for allocating local arrays.
  - My_NumGrid_In: the number of grid points allocated to a process upon starting.
  - My_Index_In: the 6 or 8 indexes of grid points allocated to a process upon starting for 3-D and 4-D FFTs, respectively.
  - My_NumGrid_Out: the number of grid points allocated to a process upon finishing.
  - My_Index_Out: the 6 or 8 indexes of grid points allocated to a process upon finishing for 3-D and 4-D FFTs, respectively.

3. Step 3: After openfft_init_c2c_3d(), openfft_init_r2c_3d(), or openfft_init_c2c_4d() is called, important variables are initialized, and can be used for allocating and initializing local input and output data arrays.

   Allocate the local input and output data arrays based on My_Max_NumGrid, which is the maximum number of grid points allocated to a process during the transformation.

```
allocate(input(My_Max_NumGrid))
allocate(output(My_Max_NumGrid))
```

Initialize the local input array from the global input array. A process is allocated (My_NumGrid_In) grid points continuously from AasBbsCcs to AaeBbeCce of the 3-D global array for 3-D FFTs, where:

```
as = My_Index_In(1) + 1
bs = My_Index_In(2) + 1
cs = My_Index_In(3) + 1
ae = My_Index_In(4) + 1
be = My_Index_In(5) + 1
ce = My_Index_In(6) + 1
```

For 4-D FFts, a process is allocated (My_NumGrid_In) grid points continuously from AasBbsCcsDds to AaeBbeCceDde of the 4-D global array, where:

```
as = My_Index_In(1) + 1
bs = My_Index_In(2) + 1
cs = My_Index_In(3) + 1
ds = My_Index_In(4) + 1
ae = My_Index_In(5) + 1
be = My_Index_In(6) + 1
ce = My_Index_In(7) + 1
de = My_Index_In(8) + 1
```

4. Step 4: Call openfft_exec_c2c_3d(), openfft_exec_r2c_3d(), or openfft_exec_c2c_4d() to transform input to output.

   ```
   openfft_exec_c2c_3d(input, output);
   ```

   OR

   ```
   openfft_exec_r2c_3d(input, output);
   ```

   OR

   ```
   openfft_exec_c2c_4d(input, output);
   ```

5. Step 5: Obtain the result stored in the local output array. Upon exiting, a process is allocated (My_NumGrid_Out) grid points continuously from CcsBbsAas to CceBbeAae of the 3-D global array for 3-D FFTs, where:

```
cs = My_Index_Out(1) + 1
bs = My_Index_Out(2) + 1
as = My_Index_Out(3) + 1
ce = My_Index_Out(4) + 1
be = My_Index_Out(5) + 1
ae = My_Index_Out(6) + 1
```

   For 4-D FFTs, a process is allocated (My_NumGrid_Out) grid points continuously from DdsCcsBbsAas to DdeCceBbeAae of the 4-D global array, where:

```
ds = My_Index_Out(1) + 1
cs = My_Index_Out(2) + 1
bs = My_Index_Out(3) + 1
as = My_Index_Out(4) + 1
de = My_Index_Out(5) + 1
ce = My_Index_Out(6) + 1
be = My_Index_Out(7) + 1
ae = My_Index_Out(8) + 1
```

6. Step 6: Finalize the calculation by calling openfft_finalize().

```
openfft_finalize();
```

11 Hybrid MPI/OpenMP Execution

OpenFFT has a native support for hybrid MPI/OpenMP execution. This means users only need to set the maximum number of threads in the parallel region as follows.

```
set OMP_NUM_THREADS=16
```

12 Complex-to-complex and Real-to-complex Transforms of 3-D FFTs

While the sizes of the input and output arrays of the c2c transform stay unchanged, i.e., a complex input array of size N1xN2xN3 will have a corresponding complex output array of the same size N1xN2xN3, the size of the complex output array is only about half of that of the real input array of the r2c transform, i.e., a real input array of size N1xN2xN3 will have a corresponding complex output array of the size N1xN2x(N3/2+1). This means both computational cost and memory usage of an r2c transform are only about half those of a c2c one. Please refer to the C and Fortran examples for illustration of input and output data manipulation.

Acknowledgements

This package has its origins in OpenMX (Open source package for Material eXplorer), and has been funded by CMSI (Computational Materials Science Initiative) of the HPCI Strategic Program (SPIRE) of the Ministry of Education, Culture, Sports, Science and Technology of Japan. We are thankful to Japan Advanced Institute of Science and Technology (JAIST) for the computational resources. We also thank Prof. Katsumi Hagita of National Defense Academy of Japan for helpful discussions and contribution to the r2c 3-D interface.

Feedback

Please feel free to drop us a line at duytvt@issp.u-tokyo.ac.jp (Truong Vinh Truong Duy) or t-ozaki@issp.u-tokyo.ac.jp (Taisuke Ozaki) for questions, comments, suggestions, and bug reports.

Bibliography

[1] T.V.T. Duy and T. Ozaki, "A decomposition method with minimum communication amount for parallelization of multi-dimensional FFTs", Computer Physics Communications, Vol. 185, Issue 1, pp. 153-164, 2014.

[2] T.V.T. Duy and T. Ozaki, "A three-dimensional domain decomposition method for large-scale DFT electronic structure calculations", Computer Physics Communications, Vol. 185, Issue 3, pp. 777-789, 2014.

[3] T.V.T. Duy and T. Ozaki, "OpenFFT: An Open-Source Package for 3-D FFTs with Minimal Volume of Communication", 29th International Supercomputing Conference (ISC'14), pp. 517-518, 2014 (Best Research Poster Award).

[4] T.V.T. Duy and T. Ozaki, "A decomposition method with minimal communication volume for parallelization of multi-dimensional FFTs", 27th International ACM Conference on Supercomputing (ICS2013), pp. 467-468, 2013.

[5] T.V.T. Duy and T. Ozaki, "A massively parallel domain decomposition method for large-scale DFT electronic structure calculations", 27th International ACM Conference on Supercomputing (ICS2013), pp. 469-470, 2013.

[6] T.V.T. Duy and T. Ozaki, "Performance Tuning of an Open-Source Parallel 3-D FFT Package OpenFFT", arXiv:1501.07350, 2015.

[7] T.V.T. Duy and T. Ozaki, "Hybrid and 4-D FFT Implementations of an Open-Source Parallel FFT Package OpenFFT", 2015.