AICS TECHNICAL REPORT

N0. 2016-002

User Manual KMATH FFT3D Version 1.0

By

Toshiyuki Imamura, Yusuke Hirota, Daichi Mukunoki,
Yoshiharu Ohi, and Yiyu Tan

RIKEN Advanced Institute for Computational Science,

7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo 650–0047, Japan

Submitted on 22/02/2017

Accepted on 14/03/2017

Published and copyrighted by
RIKEN Advanced Institute for Computational Science (AICS)
7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe, 650-0047, Japan

User Manual KMATH_FFT3D Version 1.0

Toshiyuki Imamura, Yusuke Hirota, Daichi Mukunoki,
Yoshiharu Ohi, and Yiyu Tan

RIKEN Advanced Institute for Computational Science,
7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo 650–0047, Japan

Abstract

KMATH_FFT3D is a library for realizing the cubic-decomposition method, 3D-FFT on a massively parallel supercomputer such as the K computer. When the number of processes participating in 3D-FFT is quite large, we usually suffer from large comunication overhead. Supposed that the number of processes in all-to-all can be broken down into factors, operations corresponding to all-to-all collectives are extended in order to facilitate division into several rounds, thus enabling reduction of the communications overhead. In addition, KMATH_FFT3D adopts Jung's approach to reduce the tremendus cost which comes from data-redistribution.

This document is a user manual of KMATH_FFT3D, and it describes the technical issues such as module interface and algorithms used for computation of 3D-FFTs related to KMATH_FFT3D, and a benchmark program used to verify the operations.

Keywords: 3D data transformation, volumetric FFT, distributed-memory parallel computing, MPI, FFTE, KMATH_FFT3D, the K computer

User Manual

KMATH_FFT3D

Version 1.0

Toshiyuki Imamura*, Yusuke Hirota, Daichi Mukunoki,
Yoshiharu Ohi, and Yiyu Tan
Large-Scale Parallel Numerical Computation Technology Research Team
RIKEN Advanced Institute for Computational Science

February 22, 2017

Contents

Chapter 1

Summary of parallel 3D FFT

1.1 Introduction

In numerical science and technology, there exist many simulation codes that make use of higher-order differential or convolution operations. In these kinds of computations, substitutions are often performed in frequency spaces based on Fourier transforms to reduce computational complexity and ensure computation accuracy.

$$X_k = \sum_{j=0}^{N-1} x_j \mathrm{e}^{-\frac{2\pi jk}{N}\sqrt{-1}}, \quad \text{for } k = 0, \ldots, N-1. \tag{1.1}$$

The Discrete Fourier Transform (DFT) defined in the above formula uses the Fast Fourier Transform (FFT), derived by Cooley-Tukey et al., to significantly reduce the computation volume from $O(N^2)$ to $O(N \log_2 N)$ (see Press et al. [1] and Loan [2] for details on the numerical computation method and explanation of the solution). FFT, a lightweight and extremely effective computation tool, is used in many computational science simulations and data analysis programs. There are many proposals for even higher performance FFT algorithms, and the existence of Open Source Software (OSS) such as FFTW [4], FFTE [3], and SPIRAL [6] has offered a situation in which ordinary users can easily utilize an FFT library.

Meanwhile, in large-scale supercomputers, as represented by the K computer, the selection from multiple FFT libraries such as single-node content and distributed parallel processing is significant. Whereas the presence of either of state-of-the-art FFT libraries would be adequate, with user problem establishment, and the characteristics of computer networks, etc., being limited to a single library is difficult. In particular, the three-dimensional data FFT (3D volumetric FFT) often used in scientific computation has its performance affected by the data decomposition method for one-dimensional FFT, and also by the combination of collective communications among multiple nodes. In many FFT libraries, implementation utilizes the ease of single-dimensional domain decomposition (stub decomposition), for example, in the above-mentioned FFTW [4]. Because this decomposition method does not have parallelism to a degree of resolution higher than the dimension of the axial direction selected in the easy data decomposition implementation, its unsuitability for highly parallel processing is well-known. An implementation based on two-dimensional domain decomposition (pencil decomposition) can be used to rectify this defect. FFTE [3] and 2decomp [5] are representative FFT libraries that utilize the pencil-decomposition. With this decomposition method, because the degree of resolution spanned by two-axial directions selected corresponds to the maximum degree of parallelism, a higher degree of parallelism is preserved than with single-dimensional decomposition. Moreover, three-dimensional domain decomposition (hereafter, cubic decomposition), in which three axes are selected to do the domain-decomposition at the same time to guarantee a high degree of parallelism also exists

When performing highly parallel processing in a distribute parallel environment, the tradeoff between reduction in operation time as a result of the high degree of parallelism and increase in node-to-node communication costs has to be considered. In the 3D FFT distributed parallel implementation, in particular, all-to-all collective communication is required. In the Message Passing Interface (MPI), this is the standard for distributed parallel processing, corresponding to `MPI_Alltoall`. The cost of the all-to-all process between P number of processes arrayed in a one-dimensional topology (set as a ring linked to a terminus) is

$$(\alpha_0 + \alpha P) + \beta N \tag{1.2}$$

Here, N is the data length, and α and β are, respectively, the coefficients related to communication start-up and the transfer time per unit data length. The above formula shows that the communication overhead increases in proportion to P. In general, in the above formula, α is a scale that plays a dominant role, and concealing this requires a sufficiently large N to ensure large-scale computational complexity.

In the cubic decomposition method, when the number of nodes used (the number of processes) is fixed at a constant value, increasing the numbers of split axes to two and three naturally reduces the number of processes participating in the group when viewed from each axial direction, compared to the decomposition on one axis. As a result, reduction of the all-to-all overhead cost can naturally be expected. In other words, increasing the number of split axes to two and three axes can be expected to be more effective not only in high parallelism, but also in reduction of the communications overhead. On the other hand, when a lot of axises are decomposed, extra data re-distribution should be issued in order to reflesh the data reordering. In the two-axial decomposition of FFTE[3] by Takahashi and 2decomp[5], they already supported the communication reduction technique for data re-distribution function MPI_Alltoall by omitting the ordering of intermidiate data. For the three-axial decomposition for three dimension data described in this manual, Jung's approach [8] is adopted for the implementation of the 3D-FFT to reduce unnecessary data redistribution operations (details are presented in Section 5.4).

Furthermore, when the all-to-all collective communication in the P processes is applied to the two-dimensional process grid of $P = P_1 \times P_2$ units (here, as well, it is set as a torus linked to a terminus), the parallel all-to-all operation for the P_1 and P_2 process groups existing within the vertical and horizontal one-dimensional ring continue to be executed, and the data can be rearranged to realize the same process. At this point,

$$(2\alpha_0 + \alpha(P_1 + P_2)) + 2\beta N \tag{1.3}$$

is the cost of the communication portion corresponding to all-to-all implemented by a two-round operation. Assuming that $P_1 \geq 2$ and $P_2 \geq 2$, because $P = P_1P_2 \geq P_1 + P_2$, if a single round of all-to-all is divided into two rounds, it results in reduction of the communication overhead portion (for example, in $P = 100 = 10 \times 10$, because $P_1 + P_2 = 10 + 10 = 20$, the result is $1/5$). What is more, if P can be broken down into even more factors, the items applying to α decrease, but the items applying to β increase. While this induces a certain type of tradeoff problem, in general, when N is small, the effect of α is large and quite dominant. In this way, responding as much as possible to the processes with a number of direct-product topologies, implementation after dividing all-to-all communication into multiple rounds can be expected to bring about general cost reductions demonstrated as in [9].

KMATH_FFT3D is a library for realizing the above-mentioned cubic-decomposition method, 3D-FFT. When the number of processes participating in all-to-all can be broken down into factors, operations corresponding to all-to-all collectives are extended in order to facilitate division into several rounds, thus enabling reduction of the communications overhead. In addition, KMATH_FFT3D adopts Jung's approach [8] to reduce the number of data-redistribution. This document describes the technical issues such as module, interface and algorithmes, which are used for computation of 3D-FFTs related to KMATH_FFT3D. A benchmark program is also explained in order to verify the operations.

1.2 Usage Consent / Copyright

Consent to use KMATH_FFT3D is based on the BSD 2-Clause License (noted in the LICENCE.txt file in the tarball of the library). Use of lower-level routines (one-dimensional FFT kernels) called from the KMATH_FFT3D library shall be in accordance with each individual FFT kernel. Note that the KMATH_FFT3D tarball incorporates FFTE-6.0 source code, and that FFTE was developed by Prof. Daisuke Takahashi of Tsukuba University. Further, the consent conditions as regards its use and redistribution are in accordance with the license shown below, and listed at the distribution source, `http://www.ffte.jp/`.

LICENCE.txt for KMATH_FFT3D

```
Copyright (C) 2014-2016 RIKEN.

------------------------------------------------------------------------
 Copyright notice is from here
------------------------------------------------------------------------
  Redistribution  and  use  in  source and binary forms, with or without
  modification,  are  permitted  provided  that the following conditions
  are met:

  * Redistributions  of  source  code  must  retain  the above copyright
    notice,  this  list  of  conditions  and  the  following  disclaimer.
  * Redistributions  in  binary  form must reproduce the above copyright
    notice,  this list of conditions and the following disclaimer in the
    documentation  and/or other materials provided with the distribution.

  THIS  SOFTWARE  IS  PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
  ``AS IS''  AND  ANY  EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
  LIMITED  TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
  A  PARTICULAR  PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
  HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
  SPECIAL,  EXEMPLARY,  OR  CONSEQUENTIAL  DAMAGES  (INCLUDING,  BUT NOT
  LIMITED  TO,  PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
  DATA,  OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
  THEORY  OF  LIABILITY,  WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
  (INCLUDING  NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
  OF  THIS  SOFTWARE,  EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
------------------------------------------------------------------------
```

License for FFTE

```
License

Copyright(C) 2000-2004, 2008-2014 Daisuke Takahashi
 (e-mail: daisuke[at]cs.tsukuba.ac.jp or ffte[at]ffte.jp)

You may use, copy, modify this code for any purpose (include
commercial use) and without fee. You may distribute this ORIGINAL
package.
```

Chapter 2

Getting Started

2.1 Software Required for Installation

A number of dependent software packages are required to compile KMATH_FFT3D. The software packages confirmed to be currently in operation are shown in Table 2.1.

Table 2.1: Operating software packages already confirmed, required to install KMATH_FFT3D

FFTE	FFTE version 6.0 (bundled with KMATH_FFT3D tarball) With a patch, it can also run on version 5.0
MPI	MPICH2 version 1.5 or later, MPICH version 3.0.2 or later OpenMPI version 1.6.4 or later
GNU Compiler	gcc, gfortran, g++ version 4.1.2 or later
Intel Compiler	icc, ifort version 13.0 or later
Fujitsu Compiler	mpifrtpx, mpifccpx, mpiFCCpx (cross-compilers for K and FX10)

2.2 Obtaining KMATH_FFT3D

Information related to KMATH_FFT3D can be obtained at the following URL:

`http://www.aics.riken.jp/labs/lpnctrt/KMATH_FFT3D.html`

In addition to tarball, bug and version data provision are also planned.

2.3 KMATH_FFT3D Directory Configuration

The directory configuration for this program is shown in Table 2.2. The configuration includes the `src/` directory for storing the FFT library body and the directory `doc/` for documents. Underneath those directories are working places for the library creation. In addition, a test program (`bench/`) for development users, and the directory ffte-6.0 for FFTE 6.0 and its tarball are included. Please note that deployment of the FFTE 6.0 directory is necessary in order to compile standard KMATH_FFT3D.

2.4 Compilation and Installation Procedure

A number of steps are required in order to compile KMATH_FFT3D. Please proceed in accordance with the procedure below.

Table 2.2: KMATH_FFT3D version 1.0 directory configuration

Directory	Storage and files
`KMATH_FFT3D/`	Directory storing the FFT3D library main body
`src/`	FFT3D source code
`bench/`	Source code for operations verification
`ffte-6.0/`	FFTE 6.0 source code
`doc/`	Document storage directory

2.4.1 Configure: Preparation of Makefile for Standard Compilation

For KMATH_FFT3D (the current version 1.0), use a configure script to detect the differences in each environment and automatically create a Makefile. By default, a Makefile is organized in and for the cross-compiler environment, which is standard on the K computer, and the optimization option for the test objective receives the effects of `-g`, corresponding to `-O0`. For this reason, as is also written in the `README` file, a suitable optimized option is specified by setting the appropriate environment variables.

1. `FC` (Fortran 90 compiler for MPI)
2. `FCFLAGS` (option passed to `FC` compiler)
3. `F77` (Fortran 77 compiler for MPI, usually designates the same item as `FC`)
4. `FFFLAGS` (option passed to `FF` compiler)
5. `LDFLAG` (option passed during test program linking)

Configure execution (example in K)

```
% export FC=mpifrtpx
% export FCFLAGS='-Kopenmp -Cpp -Kfast'
% export F77=mpifrtpx
% export FFLAGS='-Kopenmp -Cpp -Kfast'
% export LDFLAGS='-Kopenmp'
% ./configure --host=K --prefix=$HOME
```

The details for the configure script option can be viewed by attaching `--help` option. An effective options for ordinary users are probably the cross-compile instruction based on the option `--host=`*(host name or architecture name)* or `--{enable|disable}-openmp` on whether to activate thread parallelism. Moreover, install root directory can be disgnated by the option `--prefix=`*(install root directory)*.

Enable/disable OpenMP

```
% ./configure --enable-openmp
% ./configure --disable-openmp
```

If there is a problem that might happen due to version differences, etc., and the configure execution fails, first execute `auto.sh` to regenerate the `configure` script in the environment, and then retry execution.

Execution of auto.sh

```
% ./auto.sh
```

2.4.2 Using FFTE 5.0

While use of the latest version, FFTE 6.0, is recommended, if FFTE 5.0 is being used for mutual compatibility of applications, first deploy ffte-5.0 and then execute `patch-ffte5.sh`. This script changes the references for the symbolically linked directory, ffte, to ffte-5.0, and performs a process that absorbs the differential in the differing file configurations (in fact, it performs `factor.f` processing).

Following script execution, carry out this procedure starting from `configure`, in the same way as the normal compile procedure.

FFTE5 validation

```
% pwd
/home/foo/KMATH_FFT3D-1.0
% tar zxf ffte-5.0.tar.gz
% ls -l ffte
lrwxrwxrwx 1 foo  foo  8 Feb  3 17:40 ffte -> ffte-6.0
% ./patch-ffte5.sh
lrwxrwxrwx 1 foo  foo  8 Feb  3 23:02 ffte -> ffte-5.0
```

2.4.3 Make

Use the `Makefile` created by the `configure` script to perform make.

Make

```
% make
```

The make command generates `libkmath_fft_3d.a` in the `src/` directory. In addition, a benchmark program is generated in the `bench/` directory.

2.4.4 Instllation

By the issue of ‘make install’ command, `libkmath_fft_3d.a` and related module files are installed. Under the install root directory, which is designated on the options of the configure script, subdirectories `lib/` and `bin/` are generated, then the related library modules and the benchmark program are installed on the subdirectories, respectively.

Install

```
% make install
```

2.5 Application Builds

When building applications using this routine on the K computer, the respective parameters below must be specified during compilation and linking. Note that the library option specification order is important, and the linking process may fail if the example below is not followed. The module file (`*.mod`) referenced during the fortran90 compile process must have been installed in the directory specified during the compile process (an option specification example for the Fujitsu cross compiler in the K computer is shown below; note that this may vary in other environments or compilers).

Compile:

```
    -M<kmath fft3d root directory>/lib
```

Link:

```
-L<kmath fft3d root directory>/lib -lkmath_fft_3d
```

Chapter 3

Simple Tutorial

3.1 Warm Up

Figure 3.1 shows a program that displays a minimum portion of the software framework required for starting KMATH_FFT3D.

3.1.1 Initialization

MPI Initialization

First, use `MPI_Init` or `MPI_Init_thread` to perform MPI initialization.

KMATH_FFT3D Initial Settings

Immediately following MPI initialization, set the following three arrays and store the X, Y, Z axial direction values in the array elements in the order 1, 2, 3, respectively;

- Targeted 3D data size `nsize(1:3)`,
- Process grid shape `nproc(1:3)`, and
- Specification of multistage transposition frequency `nstage(1:3).`

Targeted 3D data size specifies the size of each X, Y, Z direction of the three-dimensional data that become the FFT transform target. Process grid specifies the number of processes in each X, Y, Z direction when the transform target three-dimensional data is decomposed into three axes. Multistage transposition will be discussed later, but if this function is not particularly necessary, setting the frequency in each axial direction as '1' is advised.

KMATH_FFT3D Initialization and Handle Acquisition

Call the initialization routine `KMATH_FFT3D_Init`, and acquire the handle. This routine is a collective operation, and all processes participating in the communicator must be called at the same time. The operation of each communicator specified by KMATH_FFT3D at initialization can be performed in parallel. The handle links the FFT routine to the data shape, process grid, and multistage transposition action information, and if these three conditions are the same, it does not matter if data differing from the same handle are called to the FFT call function `KMATH_FFT3D_Transform`, shown as follows:

```
 use MPI
 use kmath_fft3d_mod

 ! local variables
 complex(kind(0d0)), allocatable :: X(:), F(:)
 ! handle and arguments
 integer :: handle
 integer, dimension(1:3) :: nsize, nproc, nstage
........

 ! MPI initialization and setup
 call MPI_Init_thread(MPI_THREAD_MULTIPLE, provided, ierr)
 call MPI_Comm_size  (MPI_COMM_WORLD, comm_size, ierr)
 call MPI_Comm_rank  (MPI_COMM_WORLD, comm_rank, ierr)

 ! read the information about process grid and data dimension
 read*, (nsize(I), I=1, 3)
 read*, (nproc(I), I=1, 3)
 read*, (nstage(I), I=1, 3)

 ! Init KMATH_FFT3D and get a handle
 !
 call KMATH_FFT3D_Init(handle, MPI_COMM_WORLD, nsize, nproc, nstage)

... // X(:), F(:) must be allocated by appropriate data length

 ! Run FFT
 !
 call KMATH_FFT3D_Transform(handle, X, F, .true.)

 ! Finalize FFT
 call KMATH_FFT3D_Finalize(handle)

 call MPI_Finalize(ierr)
 end
```

Figure 3.1: KMATH_FFT3D simple exercise program

3.1.2 Calling the Computation Routine

Specify the handle variable, the array storing the input data, the array storing the output data, and the order and the inverse transform, into `KMATH_FFT3D_Transform`, and call to execute the cubic-decomposition FFT. Because the internal processes of this routine are collective, the processes found in the communicator specified at handle acquisition must also call this routine at the same time. In addition, because this routine has a blocking action, after the internal processes of this routine have ended, call the controls from the routine, and return to the origin.

3.1.3 End Processing

Handle Release

When the handle is released or FFT ends, call `KMATH_FFT3D_Finalize`. The handle released by the end process is invalid even if used after that time in `KMATH_FFT3D_Transform`. If the FFT process is being used again after the end process, a handle needs to be acquired from `KMATH_FFT3D_Init`.

MPI End

Finally, at the end of the MPI program, call `MPI_Finalize`.

3.2 Description of Benchmark Program

3.2.1 Program Function

In this section, we describe the benchmark program `bench_km_fft3d` function in the tarball package. Benchmark execution is performed via the following command. In the program, generate random number data and use the FFT routine provided with KMATH_FFT3D to perform the computation. In addition, perform a comparison of the results sequentially computing DFT, and confirm the computation accuracy. Note that, for DFT sequential computation, the same computation is performed at all nodes, and for large amounts of data, caution is necessary because of the computation burden.

The benchmark program has 11 argument items, but the final five items are possible to omit.

Benchmark execution

```
mpirun -n NP ./bench_km_fft3d NSIZE_X _Y _Z NPROC_X _Y _Z
                          [NSTAGE_X NSTAGE_Y NSTAGE_Z INVERSE CHECK]
```

3.2.2 Description of Each Argument

1. `NSIZE_X`: Size of data in X direction (represented by $2^P \times 3^Q \times 5^R$, P, Q, R are integers greater than zero.)

2. `NSIZE_Y`: Size of data in Y direction (represented by $2^P \times 3^Q \times 5^R$)

3. `NSIZE_Z`: Size of data in Z direction (represented by $2^P \times 3^Q \times 5^R$)

4. `NPROC_X`: Process size in X direction

5. `NPROC_Y`: Process size in Y direction

6. `NPROC_Z`: Process size in Z direction

   Note Must be `NPROC_X*NPROC_Y*N_PROC_Z = NP`.

   (The following can be omitted)

7. `NSTAGE_X`: Multistage transposition frequency in X direction (Default 1)

8. `NSTAGE_X`: Multistage transposition frequency in Y direction (Default 1)

9. `NSTAGE_X`: Multistage transposition frequency in Z direction (Default 1)

   (Note) The multistage transposition frequency must be a number that is less than the total individual number of factors.
   [Example] If `NPROC_X = 8`, because $8 = 2^3$, `NSTAGE_X = 3` is acceptable, but `NSTAGE_X = 4` is not.

10. `INVERSE`:

    0: FORWARD (Default)
    1: INVERSE
    2: FORWARD → INVERSE
    (Note) Calculate Mode 2 in FORWARD, and compute the result in INVERSE, to verify that it returns to the origin.

11. `CHECK`:

    0: NO CHECK
    1: CHECK (Default)
    This is a flag that indicates whether an accuracy check was performed. When it is set to 1, a comparison is conducted with a simple computation (if the INVERSE flag is 2, check whether it returns to the origin). When it is set to 0, only computation is performed; no comparison is carried out. If a check is performed (CHECK == 1), the result is output to a file. Note that outputting the file is a process only for rank 0. The output file consists of the following two types:

    - File 1 `dif_fft3d_dft3d`: The difference from the simple computation value is output. On the final line, the maximum value of the error (by real and imaginary) is output. If INVERSE ==2, only the final line is targeted. Note that a small number means that the computed result is correct.
    - File 2 _`out_fft3d`: The computed result is output as is (arranged in the X direction → Y direction → Z direction).

Chapter 4

API of KMATH_FFT3D

This chapter explains user interfaces of the cubic-decomposition FFT

Note: Function described in this document is not thread-safe in the current version (ver 1.0). While it is also possible to call the functions inside the OpenMP parallel domain, ensure that the internal loop that originally was subjected to thread parallel processing is processed sequentially. In addition, the thread parallel processing depends on the KMATH_FFT3D internal loop and on the FFT library called at the lower software layer.

4.1 Main Module `kmath_fft3d_mod`

When compiling source code using KMATH_FFT3D, use the module `kmath_fft3d_mod`. When this module is used, an option suitable for the command line argument during program compilation needs to be specified. An option specification example for the Fujitsu cross compiler on the K computer is shown below. Please note that this may vary in other environments and compilers.

```
Include path: -M<Install directory>/lib
Library file: -L<Install directory>/lib  -lkmath_fft_3d
```

In the program, import the cubic-decomposition FFT module, as shown below:

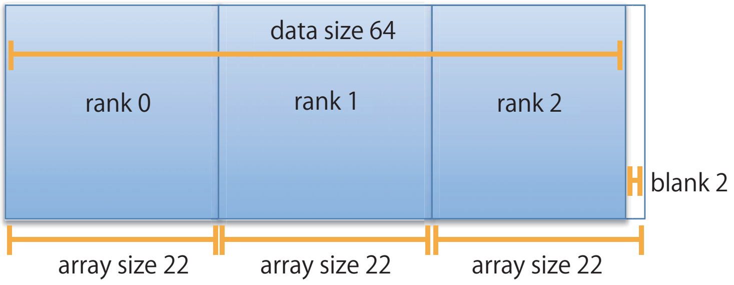
```
use kmath_fft3d_mod
```

4.2 `KMATH_FFT3D_Init`

```
use kmath_fft3d_mod
subroutine KMATH_FFT3D_Init(handle, comm,
                            box_size, num_proc, num_stage)
```

Argument	Type	IO	Descriptions
`handle`	`integer`	Out	Handle
`comm`	`integer`	In	MPI communicator
`box_size`	`integer(1:3)`	In	Size of 3D data in each XYZ axis
`num_proc`	`integer(1:3)`	In	Number of processes in each XYZ axis
`num_stage`	`integer(1:3)`	In	Number of stages in multistage transposition of each XYZ axis

- Specify the communicator comm, initialize the cubic-decomposition FFT, and return a valid handle. This subroutine is a collective operation, and all processes found in the group must be called at the same time.
- `box_size`, `num_proc`, `num_stage` are arrays of length three, and the X, Y, Z axis information is stored, respectively.
- Each `box_size` in XYZ axis value must be a product of the FFT base or, in other words, must satisfy $2^P \times 3^Q \times 5^R$, $(P, Q, R \geq 0)$.
- Each `num_proc` in XYZ axis value product must be the same as the number of ranks within the group. Note that, in the current version, even if comm is Cartesian, the process topology information is dealt with independently.
- The 3D data granted to the subroutine `KMATH_FFT3D_Transform()`, described below, must have an assured array, with `ceil(box_size/num_proc)` set to the length of each axial direction, if `box_size/num_proc` cannot be divided. This rule corresponds to the block decomposition, and as regards, for example, the X axis direction (in the figure below, the horizontal direction), if the data size is 64 and process number is 3, give the input data as shown below.



- For `num_stage`, allocate the results of factor number analysis of the number of processes (the value of `num_proc`) for each axis, for each stage, to determine the number of transposition processes in the stage. In this instance, if a stage exists in which the number of transposition processes is one, an error occurs. However, when the number of processes is one or, in other words, when limited to execution of a single process, even if the transposition process size is one, exceptionally, no error will result.

  Example: If `num_proc(1)`=40,
  Performing an analysis of factor number yields $(2, 2, 2, 5)$. If the number of stages, `num_stage`, is set to three, the values will be assigned as shown below:

Stage	Factor	Transposition process size
1st stage	2, 5	10
2nd stage	2	2
3rd stage	2	2

4.3 KMATH_FFT3D_Finalize

```
use kmath_fft3d_mod
subroutine KMATH_FFT3D_Finalize(handle)
```

Argument	Type	IO	Description
`handle`	`integer`	In	Handle

- Specify the handle obtained by the initialization, and perform the cubic-decomposition FFT. This subroutine is a collective operation, and all processes found in the communicator group specified at time of initialization must be called at the same time.

4.4 KMATH_FFT3D_Transform

```
use kmath_fft3d_mod
subroutine KMATH_FFT3D_Transform(handle, X, F, mode)
```

Argument	Type	IO	Description
`handle`	`integer`	In	Handle
`X`	`complex(kind(0d0))(:)`	In	Input 3D data array
`F`	`complex(kind(0d0))(:)`	Out	Output 3D data array
`mode`	`logical`	In	Forward or Inverse transform

- Specify the handle obtained by the initialization, and execute the cubic-decomposition FFT.
- This subroutine is a collective operation, and all processes found in the communicator group specified at time of the initialization must be called at the same time.
- Both input data X and output data F must be arrays of the same size or larger than the overall cubic-decomposed data in the total process number. For example, if the size of the overall three-dimensional data is $60 \times 60 \times 60$, and the number of processes is $4 \times 4 \times 4$, then the input array size should be set to $15 \times 15 \times 15$. In addition, if the process number is $7 \times 7 \times 7$, the input array size should be set to $9 \times 9 \times 9$. The specific method of computation is described in Section 5.4.2.
- In addition, the data use the same rules as the Fortran three-dimensional array with size expressed as a group of three indices in the order X axis direction, Y axis direction, Z axis direction, and requires storage linked to memory.
- Use mode specification to select the forward transform (`[mode=].TRUE.`) and the inverse transform (`[mode=].FALSE.`). This is optional and default is forward.
- This subroutine uses OpenMP to conduct parallel execution of multistage transposition processing in the forward direction, and FFT processing. As a result, the effects are received in the environment variable `OMP_NUM_THREADS` setting. In addition, if this subroutine has already been called within the parallel execution domain, these processes are sequentially executed.
- In actual implementation, the error process is not internally performed. For example,
  - if data size is inadequate,
  - the process number product does not match the total number of processes in the communicator group,
  - and the data size is not factored by the radices; 2, 3, and 5,

  and it is the user's responsibility to confirm this.

Chapter 5

Parallel Algorithm and Implementation Method

5.1 Three-Dimensional Fourier Transform

As shown in the following formula, the three-dimensional Fourier transform carries out a Fourier transform along the respective directions of the three-dimensional data, and generally can be realized by performing FFT in the order XYZ:

$$F(i,j,k) = \sum_{I=0}^{N_x-1} \sum_{J=0}^{N_y-1} \sum_{K=0}^{N_z-1} X(I,J,K)\omega_{N_x}^{iI}\omega_{Ny}^{jJ}\omega_{N_z}^{kK}, \quad \omega_N = \mathrm{e}^{-\frac{2\pi}{N}\sqrt{-1}} \tag{5.1}$$

The naive implementation method of the cubic-decomposition Fourier transform is realized through the following procedure. Note that, in the description below, when the process is started the input data are expressed in a group of three indices in the order X axis direction, Y axis direction, and Z axis direction, and assumed to be stored following the same rules as the Fortran three-dimensional arrays, linked to memory.

1. Multistage transposition processing in one axis with respect to the X axis direction
2. One-dimensional FFT with respect to the X axis direction
3. Multistage transposition processing in one axis with respect to the X axis direction
4. Data linkage in the Y axis direction
5. Multistage transposition processing in one axis with respect to the Y axis direction
6. One-dimensional FFT with respect to the Y axis direction
7. Multistage transposition processing in one axis with respect to the Y axis direction
8. Data linkage in the Z axis direction
9. Multistage transposition processing in one axis with respect to the Z axis direction
10. One-dimensional FFT with respect to the Z axis direction
11. Multistage transposition processing in one axis with respect to the Z axis direction
12. Data linkage in the X axis direction

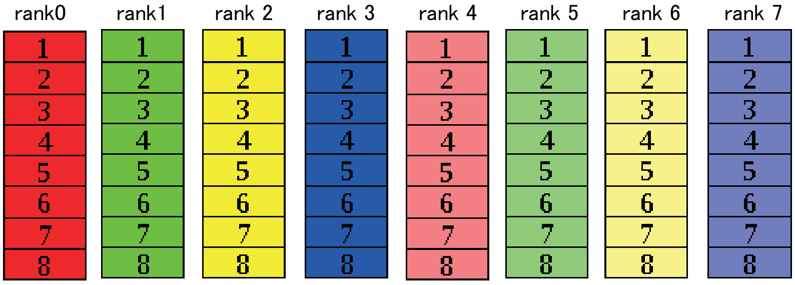
5.2 Multistage Transposition Processing in One Axis

In this section, we provide an example to illustrate the flow of the multistage transposition processing in one axis, which was proposed in [9].

In essence, the transposition processing in one axis, as seen in `MPI_Alltoall`, is an operation items are rearranged with matching the factor number and rank, with respect to the factor number of the vector data held by each process. In general, the factor data suffix (rank, factor number) becomes the same transposition operation after replacement.

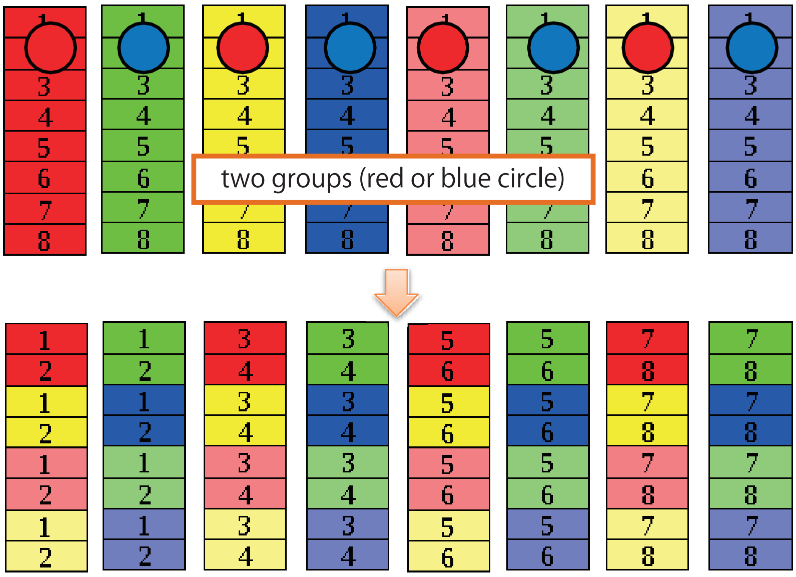
The course of the processing for a two-stage transposition in the $8 = 4 \times 2$ process is shown below. In the first stage, a 4×4 transposition operation is performed, and a 2×2 transposition operation is utilized in the second stage.

1. Initial state



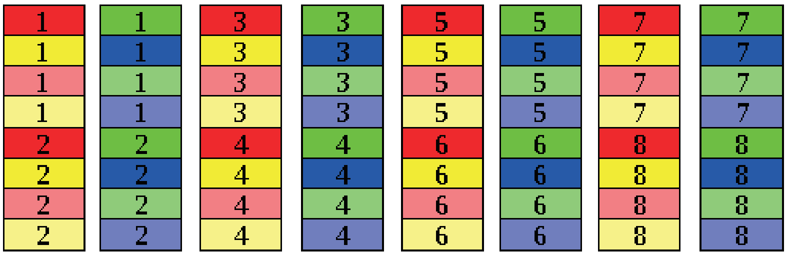
rank0	rank1	rank 2	rank 3	rank 4	rank 5	rank 6	rank 7
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8

2. Global transposition by 4 processes (distance 2) × 4 sets (2 words per set) × 2 groups



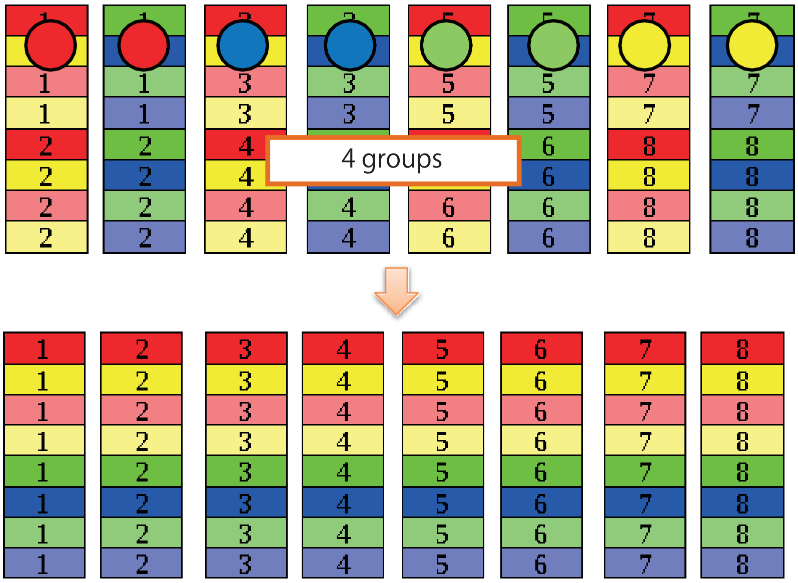
1	1	3	3	5	5	7	7
2	2	4	4	6	6	8	8
1	1	3	3	5	5	7	7
2	2	4	4	6	6	8	8
1	1	3	3	5	5	7	7
2	2	4	4	6	6	8	8
1	1	3	3	5	5	7	7
2	2	4	4	6	6	8	8

3. Local transposition in node ((2 words, 4 processes) → (4 processes, 2 words))

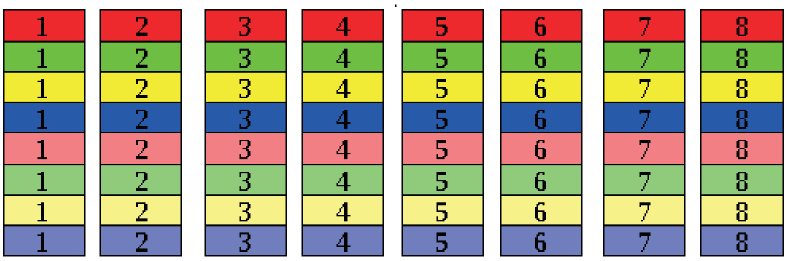


1	1	3	3	5	5	7	7
1	1	3	3	5	5	7	7
1	1	3	3	5	5	7	7
1	1	3	3	5	5	7	7
2	2	4	4	6	6	8	8
2	2	4	4	6	6	8	8
2	2	4	4	6	6	8	8
2	2	4	4	6	6	8	8

4. Global transposition by 2 processes (distance 1) × 2 sets (4 words per set) × 4 groups



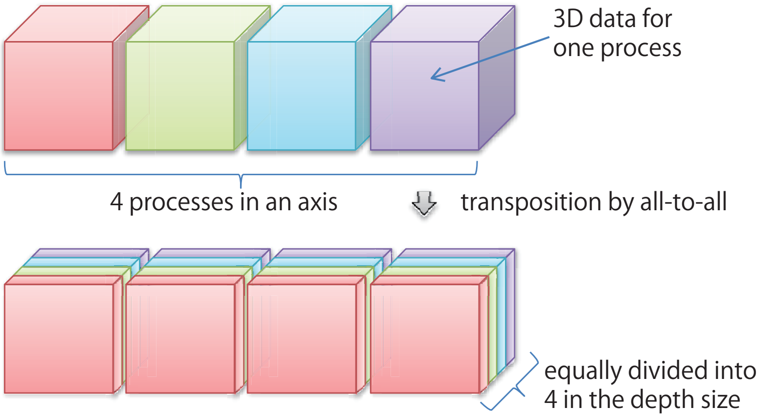
5. Local transposition in node ((4 words, 2 processes) → (2 processes, 4 words))



In the case that data more than the number of processes are stored in one node, in fact, a block (couple of elements) is handled as one factor. Performing a local transposition once in a node positions the block in order to sort the data by index, when the initial state. Next step is a suitable rearrangement of transposition operations in the node after global transposition needs to be performed.

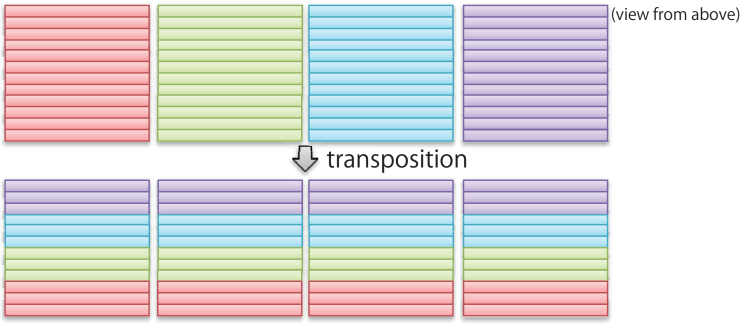
5.2.1 (Note 1) Transposition Operation in One Axis of Three-Dimensional Data

In the cubic-decomposition Fourier transform, the above-mentioned transposition operation in one axis collects a long string of data that has been distributed and positioned in each process with respect to one axis into one process (or performs the inverse operation). This corresponds to the preparation for performing a one-dimensional FFT. As shown in the figure below, we consider the data transposition operation of the distributed and held data.

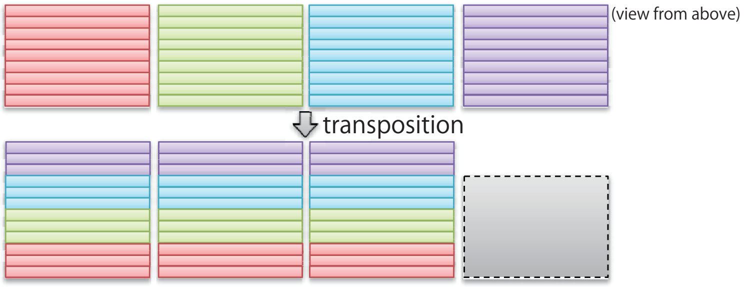


In the above figure, the X, Y, and Z axes are, respectively, the flat direction, the perpendicular direction, and the depth direction. Using the Fortran notation method, when writing the three-dimensional array holding each process as `X (1:LX, 1:LY, 1:LZ)`, we consider the transposition operation making the block `(1:LX, 1:LY, 1:LZ/4)` size as a unit.

If the depth size is 12, the size three per process (same color) is a transposition operation stacking the blocks of each color in the depth direction, with the remainder value of three obtained as the quotient of 12 and four as the basic unit. The figure below shows the appearance of the data distribution when viewed from directly above.



If the depth size is nine, then the size per process is three.



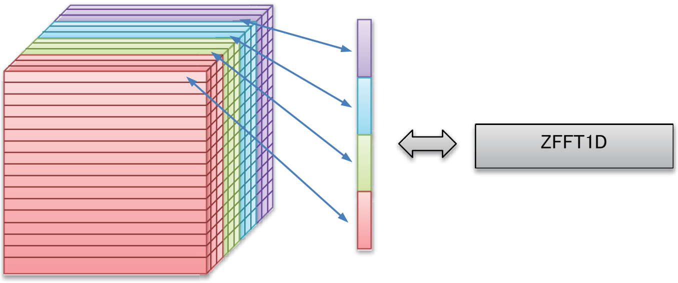
In this case, because the depth size is not divisible by the number of processes, the remainder value of three becomes the block size for one process. However, because $3 \times 4 = 12 > 9 = 3 \times 3$, the data corresponding to the left-hand process are eliminated. This generates a load imbalance in the parallel processing, leading performance degradation. Therefore, dividing the data size and process number by combinations with respect to an axis is desirable.

Note that, with respect to the standard axis and the remaining two axes, as shown by the data in the above figure, a degree of freedom to select between the 'depth direction' and 'height direction' exists. In the current implementation, the response to the multistage transposition axis, and to the depth direction and height direction, becomes as shown below:

Multistage transposition axis	Depth direction	Height direction
X axis	Z axis	Y axis
Y axis	X axis	Z axis
Z axis	X axis	Y axis

5.3 One-Dimensional FFT

Following the transposition operation outlined above, the data are rearranged to appear as shown in the figure below, the one-dimensional array cut along the decomposing axis is linked and stored in a bundle (depth size × height/4) of one-dimensional arrays not cut in the long direction (corresponding to the center panel in the figure). The FFT operations for each one-dimensional array are respectively independent, enabling parallel processing. Therefore, in the current implementation, OpenMP is used to execute the one-dimensional FFT in thread level parallel processing.



5.4 Implementation of 'Communication Reduction'-oriented High Performance Transposition

In this section, we describe the high-performance transposition implementation that reduces the transposition operations count. In naive implementation, perform the FFT after the transposition operation in one axis, and carry out a transposition operation in one axis with data of the same shape as the data structure in the original cubic-decomposition. In other words, in the FFT execution in the direction of one axis, two rounds of transposition operations (all-to-all) are required. If three axes are utilized, perform a total of six rounds of transposition operations (all-to-all). If there is no need to match the data array transformed to the frequency domain of the original space data, the pair of forward and inverse transposition operations in one axis is not necessary. The figure below shows an implementation method in which the transposition operations in one axis have been reduced to four rounds. We refer to a method to reduce the number of all-to-all operations by Jung et al.[8]. They devised a method for implementing the last two rounds of all-to-all in one round; however, in the current implementation, we use a method that configures a single-axis direction in all-to-all.

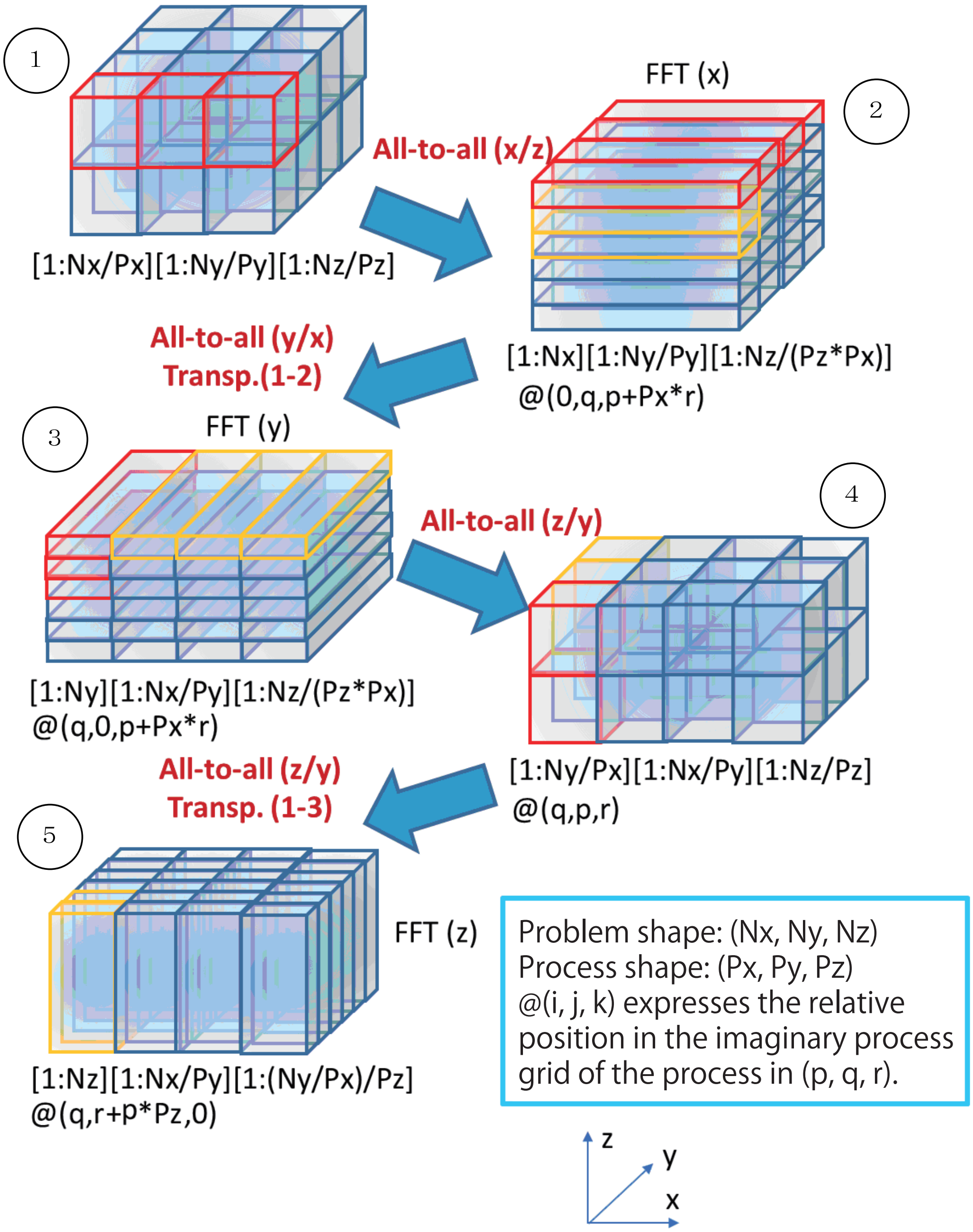

1
[1:Nx/Px][1:Ny/Py][1:Nz/Pz]
All-to-all (x/z)
FFT (x)
2
[1:Nx][1:Ny/Py][1:Nz/(Pz*Px)]
@(0,q,p+Px*r)
All-to-all (y/x)
Transp.(1-2)
3
FFT (y)
[1:Ny][1:Nx/Py][1:Nz/(Pz*Px)]
@(q,0,p+Px*r)
All-to-all (z/y)
4
[1:Ny/Px][1:Nx/Py][1:Nz/Pz]
@(q,p,r)
All-to-all (z/y)
Transp. (1-3)
5
FFT (z)
[1:Nz][1:Nx/Py][1:(Ny/Px)/Pz]
@(q,r+p*Pz,0)
Problem shape: (Nx, Ny, Nz)
Process shape: (Px, Py, Pz)
@(i, j, k) expresses the relative position in the imaginary process grid of the process in (p, q, r).
z
y
x

Figure 5.1: Summary of the transposition processing of FFT3D in which communication is reduced

5.4.1 (Note 2) Handling Internal Data

Input data, intermediate data, and output data are all allocated to each of the processes. In the communication reduction-type implementation, the data storage position cannot easily be determined. To access the data setting or specific frequency component, the process grid and data size relationship must be accurately understood.

Consider the rank (p, q, r) process in the $P_x \times P_y \times P_z$ process grid. When the general size of the data is set to $N_x \times N_y \times N_z$, the size of the input data held by each process is $L_x \times L_y \times L_z$, (where $L_x = N_x/P_x, L_y = N_y/P_y, L_z = N_z/P_z$). A description of this process is given below.

Process Grid

The process grid in the initial state, which is presented in the figure, is a (P_x, P_y, P_z) rectangle. The first transposition operation and process rearrangement operation results in $(1, P_y, P_z \cdot P_x)$. This is a logical rearrangement in one dimension of the process that positions (P_z, P_x) in two dimensional plane, and reattaches the indices. Focusing on the (p, q, r) process within the initial state, reattachment and restacking of the index in an X-Z plane arrangement, is performed, and mapped to $(0, q, p + P_z r)$. In the second round transposition operation, the process grid corresponds to $(P_y, 1, P_z \cdot P_x)$, and the (p, q, r) process to $(q, 0, p + P_z r)$. In the third and later rounds, the respective (P_y, P_x, P_z) and (q, p, r) are grouped with $(P_y, P_z \cdot P_x, 1)$ and $(q, r + P_z p, 0)$. These series of operations are summarized in Table 5.1 below.

Table 5.1:

	Logical Process Grid	Rank	Note
Initial state	(P_x, P_y, P_z)	(p, q, r)	
1st round	$(1, P_y, P_z \cdot P_x)$	$(0, q, p + P_x r)$	(Stack&Rearrange)(X/Z)
2nd round	$(P_y, 1, P_z \cdot P_x)$	$(q, 0, p + P_x r)$	Swap(X,Y)
3rd round	(P_y, P_x, P_z)	(q, p, r)	Inv(Stack&Rearrange)(Y/Z)
4th round	$(P_y, P_x \cdot P_z, 1)$	$(q, r + P_z p, 0)$	(Stack&Rearrange)(Z/Y)

Array Shape

Next, we discuss the array shapes held by the individual processes, and the corresponding relationships of the space index and the array index in the frequency domain[1]. The array shape at each stage is shown in Table 5.2. First, we assume that the (p, q, r) process holds the global index (I, J, K) of the frequency domain in the fourth round.

Table 5.2:

	Array state	Notes
Initial state	$(N_x/P_x, N_y/P_y, N_z/P_z)$	
1st round	$(N_x, N_y/P_y, N_z/(P_x P_z))$	X-axis continuation
2nd round	$(N_y, N_x/P_y, N_z/(P_x P_z))$	Y-axis continuation+XY axis switch
3rd round	$(N_y/P_x, N_x/P_y, N_z/P_z)$	
4th round	$(N_z, N_x/P_y, N_y/(P_x P_z))$	Z-axis continuation+YZ axis switch

In the spatial grid, because the process rank corresponds to $(I/(N_x/P_y), J/(N_y/(P_x P_z)), 0) = (q, r + P_z p, 0)$, the owner process (p, q, r) of the global index (I, J, K) is set as

$$p = (J/(N_y/(P_x P_z)))/P_z, \tag{5.2}$$

$$q = I/(N_x/P_y), \tag{5.3}$$

$$r = (J/(N_y/P_x P_z))\%P_z. \tag{5.4}$$

[1] However, there may be cases in which the FFT implementation differs in the array index. Here, we proceed with the discussion assuming that the frequency index based on FFT is a rising sequence.

The local index (I_1, I_2, I_3) corresponding to the global index (I, J, K) is as

$$\begin{aligned} I_1 &= K, && (5.5)\\ I_2 &= I\%(N_x/P_y), && (5.6)\\ I_3 &= J\%(N_y/(P_xP_y)), && (5.7) \end{aligned}$$

if be the owner process. Inversely, the local index (I_1, I_2, I_3) in the process (p, q, r) corresponds to the global index (I, J, K) such as:

$$\begin{aligned} I &= I_2 + q * (N_x/P_y), && (5.8)\\ J &= I_3 + (r + p * P_z) * (N_y/(P_xP_y)), && (5.9)\\ K &= I_1. && (5.10) \end{aligned}$$

5.4.2 (Note 3) Array Size

In the above discussion, array size is argued on the presumption that it is allocated by the process number or process grid size. In fact, inability to allocate according to multiple of $P*$ is a common occurrence. In such cases, the array size must be determined by the remainder (ceil) in the integer division. As a result, computation of the array size has a partially complex surface. Further, it is suggested that it should exceed the remainder values computed in all integer divisions appearing in the above-mentioned array shapes. Therefore, the data in the global data in the (N_x, N_y, N_z) data are each processed in the (P_x, P_y, P_z) process grid.

In an individual process, an array must be prepared enabling to hold the

$$\begin{aligned} \max\big\{&\max\{D_{x/x}D_{y/y}, D_{y/x}D_{x/y}\}D_{z/z},\\ &\max\{N_xD_{y/y}, N_yD_{z/y}\}\lceil D_{z/z}/P_x\rceil,\\ &N_zD_{x/y}\lceil D_{y/x}/P_z\rceil\big\} \qquad (5.11) \end{aligned}$$

elements, and passed to KMATH_FFT3D_Transform, where $D_{a/b}$ denotes $\lceil N_a/P_b\rceil$.

Chapter 6

Conclusion

6.1 Current State and Future of KMATH_FFT3D

In the current KMATH_FFT3D implementation, the functions discussed below are studied as constraints or additions, and it is possible that these will be rapidly improved and realized.

First, the current implementation does not perform concealment owing to communication and computation overlap. As a result, the FFT process has to be split into suitably sized granurality in order for communication and computation to be performed at the same time and implementation that conceals communication to proceed.

Second, as is also described in this manual, load imbalances occur too easily in the intermediate processing stage. There is much room for improving the performance through introduction of a suitable dynamic load balancing technology.

Third, in the current manual, FFTE is used as a one-dimensional FFT kernel implementation; however, if a complex FFT were to be implemented for support, this would become optional. In fact, a careful reading of the current implementation source code shows that it is usable even with the FFTW [4] or Fujitsu SSL II [7] implementations as the single dimensional FFT kernel. This should be set such that a configure script can be used for suitable selection of these options. We plan to introduce a plug-in structure or other method to respond to multi-implementation FFT kernel environments.

In addition, the utility functions also needs to be maintained. The local array size or data size to be obtained, etc., is currently the responsibility of the user. Further, because error checks, etc., are almost never carried out, we wish to proceed with function improvements that take convenience into account, while also considering versatility and migration ability.

6.2 Acknowledgments

KMATH_FFT3D development was achieved by utilizing the RIKEN K supercomputer (with the support of ‘the K computer system, improvement and user support framework: issue No. ra000005 from FY2013’). The present implementation adopted the method to reduce the number of data re-distribution reported by Jung Jaewoon in AICS, which was one of the biggest themes in the joint work. We are grateful for the support received from the members of the RIKEN Advanced Institute for Computational Science. In addition, we received feedback from many users regarding bug data and quality improvement. Without the efforts of many related members, we would surely not have arrived at this software publication, and so we would like to express our sincere appreciation here.

Bibliography

[1] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, Numerical Recipes, the Art of Scientific Computing, third edition, Cambridge University Press, 2007.

[2] C. V. Loan, Computational Frameworks for the Fast Fourier Transform, Frontiers in Applied Mathematics, SIAM, 1992.

[3] D. Takahashi, FFTE: A Fast Fourier Transform Package. `http://www.ffte.jp/`.

[4] F. Matteo, and J. G. Steven, The Design and Implementation of FFTW3, Proceedings of the IEEE, Special issue on Program Generation, Optimization, and Platform Adaptation, Vol. 93, No. 2, 2005, Pages 216–231.

[5] Library for 2D pencil decomposition and distributed Fast Fourier Transform, `http://www.2decomp.org/`.

[6] J. M. F. Moura, J. Johnson, R. W. Johnson, D. Padua, V. K. Prasanna, M. Püschel, and M. Veloso, SPIRAL: Automatic Implementation of Signal Processing Algorithms, Proceedings of High Performance Embedded Computing (HPEC), 2000, See also the SPIRAL project page, `http://www.spiral.net/`.

[7] Fujitsu SSL II User's Handbook, 1999, Fujitsu SSL II/MPI User's Handbook, 2008, (both in Japanese).

[8] J. Jung, C. Kobayashi, T. Imamura, and Y. Sugita, Parallel implementation of 3D FFT with volumetric decomposition schemes for efficient molecular dynamics simulations, Computer Physics Communications, Vol. 200, March 2016, Pages 57–65.

[9] S. Yamada, T. Imamura. and M. Machida, Parallelization Design on Multi-core Platforms in Density Matrix Renormalization Group toward 2-D Quantum Strongly-correlated Systems, Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis Date of Conference (SC11), 12-18 Nov. 2011