

# User manual

Thomas Koopman

August 18, 2021

## Introduction

This manual documents a distributed memory version of the higher-dimensional discrete Fourier transform routines the FFTW library [1], using MPI. Familiarity with FFTW and MPI is assumed.

## 1 Installation/usage

As the code is only a few hundred lines, I did not see the point in compiling it into a library. If the user wants to use the parallel DFT routine into a program `application.c`, (s)he simply copies `mpi_fft.h` and `mpi_fft.c` into the same directory `application.c` is located, and adds the line `#include "mpi_fft.h"` at the top of `application.c`. See for example `mpi_test.c`. Then compile `application.c` together with `mpi_fft.c`. On a Unix-like system, the compilation command could look like

```
mpicc -o application application.c mpi_fft.c -lfftw3 -lm
```

On my machine, the flag `-O3` speeds up `mpi_fft.c`, the flags `-mavx2` or `-march=native` do not.

## 2 Warning

Due to the use of a function of the MPI library, the number of local elements is restricted to  $2^{32}$ . A complex number is 10 bytes, and we need at least twice as much space as we have elements because we buffer the communication, so  $2^{32}$  elements take at least 84GB of memory per core. So this is probably not a problem for most people, but be aware just in case you have very high memory nodes.

## 3 Data distribution and initialization

The parallel function

```
void mpiFft(fftw_complex* localData, int dimension, int n[], int p[], fftw_plan bigPlan, fftw_plan manyPlans
```

takes its data as variable `localData` of size  $n[0] \times \dots \times n[\text{dimension} - 1]$ . The distribution is  $n$ -dimensional cyclic over a grid of  $p[0] \times \dots \times p[\text{dimension} - 1]$  processors. That means that local position  $(k_0, \dots, k_{d-1})$  on processor  $(s_0, \dots, s_{d-1})$  corresponds to global position  $(s_0 + k_0 p_0, \dots, s_{d-1} + k_{d-1} p_{d-1})$  and global position  $(j_0, \dots, j_{d-1})$  corresponds to local position  $(j_0 \text{ div } p_0, \dots, j_{d-1} \text{ div } p_{d-1})$  on processor  $(j_0 \text{ mod } p_0, \dots, j_{d-1} \text{ mod } p_{d-1})$ . The plans can be made with the functions `createBigPlan` and `createManyPlans`.

We must have that  $p[i]^2 \mid n[i]$  for all  $i$ . With `printAdmissibleProcessors` you can see what total numbers of processors you can use. The function `fillProcessors` you can decompose an admissible number of processors into a grid.

## References

- [1] Matteo Frigo and Steven G. Johnson. The fastest fourier transform in the west. In *the Proceedings of the 1998 International Conference on Acoustics, Speech, and Signal Processing, ICASSP '98*, 1997.