XPHASE3D

User's Guide

(Version 2024.04)

Computational Structural Biology Research Team

RIKEN Center for Computational Science

Table of Contents

1. Introduction

Xphase3d comprises a suite of programs written in the C language. Its primary program can perform multiprocessing phase retrieval on large-scale 3D volumes, utilizing the distributed memory in clusters or supercomputers. It also features other memory-distributed programs to prepare for the phase retrieval and process the reconstruction results.

A supplementary Python package xphase3dpy is also provided. It can perform single-process and single-threaded phase retrieval on small 3D volumes. Its algorithms and mathematical procedures are highly consistent with those in the xphase3d C programs. The Python scripts in xphase3dpy are easy to read and edit, allowing users new to phase retrieval to practice and test on small 3D volumes. Users can also run xphase3dpy on supercomputers to perform massive independent phase retrieval tasks simultaneously.

Reference:

W. Zhao, O. Miyashita, M. Nakano, and F. Tama, “Xphase3d: Memory-distributed phase retrieval for reconstructing large-scale 3D density maps of biological macromolecules,” 2024.

2. Installation

2.1. Installing xphase3d on your own workstation

2.1.1. System requirement

In principle, xphase3d can be run on any system with a decent C compiler and supporting MPI. Authors have confirmed its successful operation on Ubuntu 20.04 and CentOS 7.

2.1.2. Dependencies

Xphase3d requires dependencies including Open MPI, FFTW, and HDF5. Authors have not fully tested the compatibility of their different versions on various systems. For reference, authors are using the following versions:

Open MPI 4.0.4	https://www.open-mpi.org/software/ompi/v4.0/
FFTW 3.3.10	https://www.fftw.org/download.html
HDF5 1.14.2	https://portal.hdfgroup.org/downloads/

Install these dependencies following their respective installation instructions. For example, without root authority, FFTW can be installed to `/home/xxx/FFTW`. Such directories are needed later to compile xphase3d.

When installing FFTW, make sure to add the flag `--enable-mpi` during configuration to install the FFTW MPI library.

2.1.3. Compilation

Ensure that the pre-installed MPI compiler, mpicc, is available by the system. Go to the root directory of xphase3d and then:

```
cd src/
./configure
./make
```

During configuration, provide the installation directories of FFTW and HDF5, such as /home/xxx/fftw and /home/xxx/hdf5. The compiler flags indicated by h5cc -show can be printed via

```
/home/xxx/hdf5/bin/h5cc -show
```

2.2. Installing xphase3d on Fugaku

On Fugaku login node, Go to the root directory of xphase3d and then:

```
cd src/
./fugaku_make
```

2.3 Python environment for xphase3dpy

Xphase3dpy requires dependencies including numpy, scipy, mpi4py, h5py, and mrcfile. There are no strict requirements for their versions. For reference, authors are using the following versions:

python = 3.8

numpy >= 1.23

scipy >= 1.9

mpi4py = 3.1.4

h5py >= 3.0

To pip install mpi4py, ensure that the pre-installed MPI compiler, mpicc, is available.

If Anaconda is being used to manage Python environments, an environment named xphase can be built using xphase.yml provided in the root directory of xphase3d:

```
conda env create -f xphase.yml
```

To import xphase3dpy into your own Python program, copy or link the directory xphase3dpy/ to your working directory.

3. Quick start with examples

A few examples are provided in `examples/` to demonstrate how to use xphase3d commands and xphase3dpy.

3.1. On your own workstation

Go to the root directory of xphase3d and activate the Python environment for xphase3dpy. Ensure that the pre-installed MPI launcher mpiexec is available.

```
cd examples/

ln -s ../bin bin

ln -s ../xphasd3dpy xphase3dpy

# To demonstrate the usage of xphase3dpy. Find the created files in
0_data/

python 0_demo.py

# To demonstrate the usage of xphase3d commands. In halfway, 1_demo.py
will be executed for data reformatting. Find the created files in
1_data/

bash 1_demo.sh
```

3.2. On Fugaku

Go to the root directory of xphase3d and then:

```
cd examples

ln -s ../bin bin

ln -s ../xphasd3dpy xphase3dpy
```

```
# To perform multiple independent single-process phase retrieval
trials using xphase3dpy. Find the created files in fugaku_data_py/

pjsub fugaku_demo_py.job

# To perform multiple independent single-process phase retrieval
trials using xphase3dpy. This is a bulk job consisting of two subjobs.
Find the created files in fugaku_data_py/

pjsub --bulk --sparam “0-1” fugaku_demo_py_bulk.job

# or

./fugaku_demo_py_bulk.job.pjsub

# To perform a memory-distributed phase retrieval using xphase3d. Find
the created files in fugaku_data_bin/

pjsub fugaku_demo_bin.job
```

Note 1: Ensure to replace `ra000015` with your own Fugaku group ID in `fugaku_demo_py.job`, `fugaku_demo_py_bulk.job`, and `fugaku_demo_bin.job`.

Note 2: Ensure to replace the scripts below `# Activate the pre-set Python environment` with your own scripts for activating the Python environment for `xphase3dpy` in `fugaku_demo_py.job` and `fugaku_demo_py_bulk.job`.

4. Data formats

4.1. Data format for xphase3d

The data format for xphase3d C programs is HDF5.

For a 3D volume with dimensions $N_x \times N_y \times N_z$, it is segmented into N_x slices along the x-axis. Each 2D slice of $N_y \times N_z$ is stored at `/dataset` within an HDF5 named as `${keyword}_${x}.h5`, where `$x` indicates the x-th slice.

For Fourier modulus (usually denoted as M) and density map (usually denoted as R), the datatype of the 2D slice at `/dataset` is double (H5T_IEEE_F64LE).

For support (usually denoted as S) and the mask of Fourier modulus (usually denoted as H), the datatype of the 2D slice at `/dataset` is uint8 (H5T_NATIVE_CHAR).

4.2. Data format for xphase3dpy

Xphase3dpy uses the NPY format which is designed for storing numpy arrays.

4.3. Data format conversion

To segment a 3D volume of double in NPY format to HDF5 slices, use the function `segment_f()` in xphase3dpy.tools. For example:

```
segment_f("M.npy", 64, 64, 64)
```

Here, `M.npy` stores a 3D array of double with dimensions $64 \times 64 \times 64$. It is segmented into `M_0.h5`, `M_1.h5`, …, `M_63.h5`. These created 64 HDF5 slices can be read by xphase3d.

Similarly, to segment a 3D volume of uint8 in NPY format to HDF5 slices, use the function `segment_b()` in xphase3dpy.tools. The example below demonstrates segmenting `S.npy` into `S_0.h5`, `S_1.h5`, …, `S_63.h5`:

```
segment_b("S.npy", 64, 64, 64)
```

Reversely, to connect the HDF5 slices into a 3D volume in NPY format, whether the datatype is double or uint8, use the function `connect()` in xphase3dpy.tools. The example below demonstrates connecting `M_0.h5`, `M_1.h5`, …, `M_63.h5` into `M.npy`:

```
connect("M.npy", 64, 64, 64)
```

To view the shape of `M.npy`, `import xphase3dpy.tools` and then do as follows. The created `M.mrc` can be viewed in Chimera:

```
M = numpy.load("M.npy")
save_mrc("M.mrc", M)
```

It is recommended to use `segment_f()` and `segment_b()` to prepare data files for xphase3d.

5. List of commands in xphase3d

Xphase3d has 10 commands: run, make_m, make_s, make_r0, align, prtf, fsc, merge, bin_f, and bin_b. You can type in these commands without arguments to print their usages. The usages and remarks of these commands are summarized here.

5.1. run

To perform multiprocessing, memory-distributed phase retrieval.

For a 3D volume consisting of V voxels, it requires at least nV bytes of memory. Here, n equals 146 for the error reduction (ER) algorithm, 154 for the hybrid input-output (HIO) algorithm, or 162 for averaged successive reflections (ASR), hybrid projection reflection (HPR), relaxed averaged alternating reflectors (RAAR), or difference map (DM).

```
Usage: mpiexec -n <num_processes> run sample.config


Arguments
    - mpiexec -n <num_processes>      : Number of MPI processes
    - sample.config                   : Configuration file
```

Examples of the configuration file can be found in `examples/`, such as `1_demo_0.config`:

```
# Configurations

# Size of X, Y, Z dimensions
XN: 64
YN: 64
ZN: 64

# M - Fourier modulus
KEYWORD M: 1_data/M
# R0 - Initial 3D model in real space
KEYWORD R0: 1_data/R0
# S - Initial support in real space
KEYWORD S: 1_data/S
# H - Mask in Fourier space
KEYWORD H: 1_data/H
# Rp - Reconstructed model in real space, to be generated
KEYWORD Rp: 1_data/Rp_0

# 1 for needed, 0 for not needed
NEED R0: 1
NEED S: 1
NEED H: 1

# Available methods: HIO, ER, ASR, HPR, RAAR, DM
METHOD: HIO
BETA: 0.9
LOWER BOUND: 0.0
UPPER BOUND: 1000.0

# The number of times of shrink wrapping
NUM LOOP: 50
# The number of optimizations between two shrink wrapping processes
NUM ITERATION: 20

# Parameters in shrink wrapping
SIGMA0: 3
SIGMAR: 0.01
TH: 0.05
```

For the Fourier modulus, M should put high-frequency components at the center and low-frequency components at the edges. The zero-frequency component is at the leftmost position. This follows the convention of scipy.fft (https://docs.scipy.org/doc/scipy/tutorial/fft.html):

> *"*For N even, the elements $y[1] \ldots y[N/2-1]$ contain the positive-frequency terms, and the elements $y[N/2] \ldots y[N-1]$ contain the negative-frequency terms, in order of decreasingly negative frequency. For N odd, the elements $y[1] \ldots y[(N-1)/2]$ contain the positive-frequency terms, and the elements $y[(N+1)/2] \ldots y[N-1]$ contain the negative-frequency terms, in order of decreasingly negative frequency.*"*

This is also the convention of FFTW (https://www.fftw.org/fftw3_doc/The-1d-Discrete-Fourier-Transform-_0028DFT_0029.html):

> "For those who like to think in terms of positive and negative frequencies, this means that the positive frequencies are stored in the first half of the output and the negative frequencies are stored in backwards order in the second half of the output."

When NEED R0 is 0, the initial model R0 consists of random numbers in the range of $[0,1)$ across the entire 3D volume.

When NEED S is 0, the support S extends across the entire 3D volume.

When NEED H is 0, there is no mask for the Fourier modulus.

SIGMA0 σ_0 and SIGMAR σ_r are responsible to the standard deviation σ_i of Gaussian filtering in the i-th process of shrink wrap:

$$\sigma_i = \sigma_0(1 - \sigma_r)^{i-1}.$$

5.2. make_m

To calculate the Fourier modulus (M) of a density model (R), usually for simulation or testing purposes.

For a 3D volume consisting of V voxels, it requires at least $48V$ bytes of memory.

```
Usage: mpiexec -n <num_processes> make_m IN OUT XN YN ZN


Arguments
    - mpiexec -n <num_processes> : Number of MPI processes
    - IN                   : Keyword of input file: Density model
    - OUT                  : Keyword of output file: Fourier modulus
    - XN                   : Size of the X dimension
```

```
    - YN                    : Size of the Y dimension
    - ZN                    : Size of the Z dimension
```

5.3. make_s

To estimate a support based on the Fourier modulus (M) when there is no priori knowledge on the density model (R) to be reconstructed. The created support (S) is approximately the autocorrelation of the density model (R).

For a 3D volume consisting of V voxels, it requires at least $105V$ bytes of memory.

```
Usage: mpiexec -n <num_processes> make_s IN OUT XN YN ZN SIGMA TH


Arguments
    - mpiexec -n <num_processes> : Number of MPI processes
    - IN              : Keyword of input file: Fourier modulus M
    - OUT             : Keyword of output file: Support S
    - XN              : Size of the X dimension
    - YN              : Size of the Y dimension
    - ZN              : Size of the Z dimension
    - SIGMA           : Standard deviation of Gaussian convolution
    - TH              : Threshold factor
```

SIGMA represents the standard deviation of Gaussian convolution utilized for smoothing the calculated autocorrelation to facilitate the identification of a smooth contour boundary. This contour boundary is delineated where the maximum value of the smoothed autocorrelation scales with TH.

5.4. make_r0

To create a random seed of the density model (R0) to be iteratively optimized during phase retrieval. This seed fills random values within [0,1) in the given support (S).

make_r0 reads, processes, and outputs density model files slice by slice. Running it doesn't require a large amount of memory.

```
Usage: mpiexec -n <num_processes> make_r0 IN OUT XN YN ZN

Arguments
    - mpiexec -n <num_processes> : Number of MPI processes
    - IN                         : Keyword of input file: Support S
    - OUT                        : Keyword of output file: Initial model R0
    - XN                         : Size of the X dimension
    - YN                         : Size of the Y dimension
    - ZN                         : Size of the Z dimension
```

5.5. align

To align two density models reconstructed from the same Fourier modulus to eliminate the ambiguity of translational shift and conjugate flip.

For a 3D volume consisting of V voxels, it requires at least $168V$ bytes of memory.

```
Usage: mpiexec -n <num_processes> align IN1 IN2 OUT XN YN ZN

Arguments
    - mpiexec -n <num_processes> : Number of MPI processes
    - IN1  : Keyword of input file 1: Density model of the reference
    - IN2  : Keyword of input file 2: Density model to be aligned
```

```
    - OUT  : Keyword of output file:  Density model IN2 after alignment
    - XN   : Size of the X dimension
    - YN   : Size of the Y dimension
    - ZN   : Size of the Z dimension
```

5.6. PRTF

To compute the phase-retrieval transfer function (PRTF) for a series of density models reconstructed from the same Fourier modulus but starting from different random seeds. These density models must be pre-aligned.

For a 3D volume consisting of V voxels, it requires at least $72V$ bytes of memory.

```
Usage: mpiexec -n <num_processes> prtf IN START END OUT XN YN ZN


Arguments
    - mpiexec -n <num_processes> : Number of MPI processes
    - IN                    : Keyword of input 3D density models
                              Use '#' as wildcard for sequential number
    - START                 : Starting sequential number
    - END                   : Ending sequential number
    - OUT                   : Filename of the output PRTF data
    - XN                    : Size of the X dimension
    - YN                    : Size of the Y dimension
    - ZN                    : Size of the Z dimension
```

5.7. merge

To average a series of density models reconstructed from the same Fourier modulus but starting from different random seeds. These density models must be pre-aligned.

merge reads, calculates, and outputs density model files slice by slice. Running it doesn't require a large amount of memory. However, special attention should be given to avoiding running too many processes to reduce the I/O burden on the operating system. If "merge" becomes stuck, consider reducing the number of processes.

```
Usage: mpiexec -n <num_processes> merge IN START END OUT XN YN ZN

Arguments
    - mpiexec -n <num_processes> : Number of MPI processes
    - IN              : Keyword of input 3D density models
                        Use '#' as wildcard for sequential number
    - START           : Starting sequential number
    - END             : Ending sequential number
    - OUT             : Keyword of the output merged 3D density model
    - XN              : Size of the X dimension
    - YN              : Size of the Y dimension
    - ZN              : Size of the Z dimension
```

5.8. fsc

To compute the Fourier shell correlation (FSC) between two density models. The two models must be pre-aligned.

For a 3D volume consisting of V voxels, it requires at least $72V$ bytes of memory.

```
Usage: mpiexec -n <num_processes> fsc IN1 IN2 OUT XN YN ZN
```

```
Arguments
    - mpiexec -n <num_processes> : Number of MPI processes
    - IN1                  : Keyword of input file 1: 3D density model
    - IN2                  : Keyword of input file 2: 3D density model
    - OUT                  : Filename of the output FSC data
    - XN                   : Size of the X dimension
    - YN                   : Size of the Y dimension
    - ZN                   : Size of the Z dimension
```

5.9. bin_f

To reduce the dimensions of a density model (R) or Fourier modulus (M) for easier data preview by binning voxels.

For a 3D volume consisting of $N_x \times N_y \times N_z$ voxels, it requires at least $N_x \times \left(N_y/BF\right) \times (N_z/BF) \times 8$ bytes of memory, where BF is the binning factor.

```
Usage: bin_f BF IN OUT XN YN ZN


Arguments
    - BF              : Binning factor
    - IN              : Keyword of input file: Original density model
    - OUT             : Name of output file: Binned density model
    - XN              : Size of the X dimension
    - YN              : Size of the Y dimension
    - ZN              : Size of the Z dimension
```

5.10. bin_b

To reduce the dimensions of a support (S) or a mask (H) for easier data preview by binning voxels.

For a 3D volume consisting of $N_x \times N_y \times N_z$ voxels, it requires at least $N_x \times (N_y/BF) \times (N_z/BF) \times 8$ bytes of memory, where BF is the binning factor.

```
Usage: bin_b BF IN OUT XN YN ZN


Arguments
    - BF              : Binning factor
    - IN              : Keyword of input file: Original density model
    - OUT             : Name of output file: Binned density model
    - XN              : Size of the X dimension
    - YN              : Size of the Y dimension
    - ZN              : Size of the Z dimension
```