

THUNDER v1.3 User Guide

January 18, 2018

1 Installation

1.1 Requirement of Installation

- C/C++ compiler supporting C++98 standard along with MPI wrapper
- `cmake`

We recommend `gcc` and Intel C/C++ compiler as C/C++ compiler. Moreover, `gcc42` has been tested as the oldest supporting version of `gcc`. OpenMPI and MPICH both can be used as MPI standard. In Tsinghua, we use `openmpi-gcc43` as the C/C++ compiler for compiling THUNDER.

`cmake` is a tool for configuring source code for installation.

`openmpi-gcc43` is open-source software, which can easily installed using `yum` on CentOS and `apt-get` on Ubuntu. `cmake` has been already installed in most Linux operating systems. If not, it can also be conveniently installed by `yum` on CentOS and `apt-get` on Ubuntu.

1.2 Installing from Source Code

1.2.1 Preparation Before Configuring Source Code

Make sure `cmake` have been installed and correctly placed in environment. Thus, `cmake` can correctly set up the environment for compiling THUNDER.

1.2.2 Configure Using `cmake`

In THUNDER source code directory, please type in the following commands for configuring source code. `install_dir` stands for where you want THUNDER to be installed.

```
1 mkdir build
2 cd build
3 cmake -DCMAKE_INSTALL_PREFIX="install_dir" ..
```

1.2.3 Compile and Stage Binaries into Environment

Please type in the following command for compiling source code using 20 threads. You may change the number after `-j` to be number of threads you desire for compiling.

```
1 make -j20
2 make install
```

After compiling and installation, several folders will appear under the directory `install_dir`: `include` containing header files, `bin` containing executable binaries, `lib` containing several libraries, `script` containing scripts needed and `manual` containing this user guide. The compiled binaries are listed as

- `thunder`
- `thunder_average`
- `thunder_genmask`
- `thunder_lowpass`
- `thunder_mask`
- `thunder_postprocess`
- `thunder_resize`

For the purpose of convenience, you may stage binaries into environment. For example, you may add the following command into shell configuration file

```
1 setenv PATH=install_dir/bin:$PATH
```

when `csh` or `tcsh` is used as shell. Meanwhile, you may add the following command into shell configuration file when `bash`, `zsh` or `ksh` is used as shell.

```
1 export PATH=install_dir/bin:$PATH
```

After staging binaries into environment, you may directly access these binaries by typing their filenames in shell.

2 Submit Your Job

`thunder` is the core program of THUNDER. It executes 3D classification and refinement. It reads in a JSON parameter file. After parsing the JSON parameter, it reads in initial model, a `.thu` file and particle images. It also reads in mask if necessary.

2.1 Set Up .thu File

THUNDER uses .thu file for storing information of each particle image. .thu file is a space-separate tabular file as each column stands for a specific variable, as listed below.

1. Voltage (Volt)
2. DefocusU (Angstrom)
3. DefocusV (Angstrom)
4. DefocusTheta (Radian)
5. Cs (Angstrom)
6. Amplitude Contrast
7. Phase Shift (Radian)
8. Path of Particle
9. Path of Micrograph
10. Coordinate X in Micrograph (Pixel)
11. Coordinate Y in Micrograph (Pixel)
12. Group ID
13. Class ID
14. 1st Element of the Unit Quaternion
15. 2nd Element of the Unit Quaternion
16. 3rd Element of the Unit Quaternion
17. 4th Element of the Unit Quaternion
18. 1st Standard Deviation of Rotation
19. 2nd Standard Deviation of Rotation
20. 3rd Standard Deviation of Rotation
21. Translation X (Pixel)
22. Translation Y (Pixel)
23. Standard Deviation of Translation X (Pixel)
24. Standard Deviation of Translation Y (Pixel)
25. Defocus Factor

26. Standard Deviation of Defocus Factor

27. Score

.thu file is generated by **thunder** at the end of each iteration to save the information of each particle image.

2.1.1 Generate .thu from Relion

.thu file can be converted from STAR file of Relion by script **STAR_2_THU.py** and **STAR_2_THU_NO_GROUP.py** by the following commands.

```
1 python STAR_2_THU.py filename.star > filename.thu
```

```
1 python STAR_2_THU_NO_GROUP.py filename.star >
   filename.thu
```

STAR_2_THU.py is used for converting STAR files containing group information and **STAR_2_THU_NO_GROUP.py** is used for converting those do not. **You can find these two scripts under directory `install_dir/script`.**

It is worth noticed that both of two scripting only convert CTF information but not rotation and translation information. Thus, .thu files converted from STAR files can be only used for global search stage of **thunder**. Meanwhile, .thu files generated by **thunder** can be used for global search, local search and CTF search. The precise meaning of global search, local search and CTF search will be further discussed in detail in section 2.2.

2.1.2 Generate .thu from Frealign

The converting script will be provided soon.

2.1.3 Generate .thu from SPIDER

The converting script will be provided soon.

2.2 Configure with JSON Parameter File

thunder reads in a JSON file which is parsed into parameters of **thunder**. You may change the values of the keys to fit your purpose. The definition of keys in this JSON parameter file is listed in Table 1.

thunder divides 3D refinement into three stages: global search, local search and CTF search. During global search, the rotation and translation result of the previous iteration will **not** inherited into the next iteration. Meanwhile, during local search, the rotation and translation of each particle image will be adjust based on the result of the previous iteration. During CTF search, the CTF parameters will be adjusted for achieving better resolution.

Meanwhile, 3D classification of **thunder** typically only involves global search.

You may find a demo version of this JSON parameter file named **demo.json** under directory **install_dir/script**.

2.3 Processes and Threads

thunder needs at least 3 processes. It has perfect linear speed-up when number of nodes increases. Thus, please use as many nodes as possible. We highly recommend assigning a node with only one process and using multiple cores in each node by threads. For example, if you have 100 nodes and each node has 20 cores, you may use 100 processes for running **thunder**, and each process should generate 20 threads to achieve maximum usage of computing resource. By changing the value of the key **Number of Threads Per Process** in the JSON parameter file, you may set the number of threads of each process to which you desire. In this example, this value should be set to 20.

2.4 Submit

Please examine whether you have generated the correct **.thu** file and configured the JSON parameter file properly, and make sure that the initial model and mask (if necessary) are placed in the right directory. Moreover, please check whether **the directory of the destination described in the JSON parameter exists** or not. Now, you can submit your job. You may leave it to the cluster job managing software, or you may assign nodes manually by **mpirun**.

3 Get Your Result

A log file named **thunder.log** will appear in your submitting directory, recording the state of your job.

In the destination directory, the density maps are outputted as **Reference_xxx.A.Round_xxx.mrc** and **Reference_xxx.B.Round_xxx.mrc**, during 3D refinement or classification. For example, the density map of the 5th reference of round 15 from hemisphere A has the filename **Reference.005.A.Round.015.mrc**. On contrast, the 5th reference of round 15 from hemisphere B has the filename **Reference.005.B.Round.015.mrc**.

Meanwhile, during 2D classification, the density maps of each round are stored in a MRC stack. For example, the density maps of round 15 has the filename **Reference.Round.015.mrcs** which contains N slices of images. N stands for the number of classes.

FSC/FRCs are outputted as **FSC.Round_xxx.txt**. The first column of this file is signal frequency in pixel. The second column is signal frequency in Angstrom. From the third column to the rest of columns, the FSC of each reference is listed in order.

During classification, the resolution and ratio of images of each class is listed in a file named **Class_Info.Round_xxx.txt**. Each row of this file stands for a class in order. The first column is the index of each class, the second column is the

Key	Description
Number of Threads Per Process	the number of threads used in each process
2D or 3D Mode	2D/3D classification or refinement
Global Search	whether to perform global search or not
Local Search	whether to perform local search or not
CTF Search	whether to perform CTF search or not
Number of Classes	the number of density maps, aka. more than 1 when undergoing classification
Size of Image	the size of the images ¹
Pixel Size (Angstrom)	the pixel size of the images
Radius of Mask on Images (Angstrom)	the radius of mask you want to be masked on the images
Estimated Translation (Pixel)	the standard deviation of translation in pixel which may occurred on the input images
Initial Resolution (Angstrom)	the resolution the program starts its iterations
Perform Global Search Under (Angstrom)	the resolution limit for performing global search
Symmetry	the symmetry of the macromolecular to be processed
Initial Model	the initial model for classification/refinement
.thu File Storing Paths and CTFs of Images	the .thu file which stores the information of where to read the images and the CTF paramters of them
Prefix of Particles	the prefix to be added in the path of the particle image
Prefix of Destination	the prefix (path) to save the outcomes
Calculate FSC Using Core Region	whether to calculate FSC using core region of the reference or not
Calculate FSC Using Masked Region	whether to calculate FSC using masked region of the reference or not
Particle Grading	whether to turn on the particle grading optimization or not
Perform Reference Mask	whether to mask on the density map or not
Perform Reference Mask during Global Search	whether to mask on the density map during global search or not
Provided Mask	the path of the mask if needed

Table 1: Key Words of JSON Parameter File

resolution in Angstrom of each class and the third column is the ratio of image of each class.

The rotation and translation information of each particle at each iteration is outputted as `Meta_Round_xxx.thu`, which follows the `.thu` file format. For example, rotation and translation of round 15 has the filename `Meta_Round_015.thu`.

4 Typically Workflow

The typically workflow of cryo-EM single particle analysis includes 3 steps: 2D classification, 3D classification and 3D refinement.

4.1 2D Classification

The first step of cryo-EM single particle analysis is 2D classification for removing ice and "noisy" particles.

You can find a demo version of this JSON parameter file for 2D classification named `demo_2D.json` in this package. There are some options worth noticed in this JSON parameter file. They are listed below.

Local Search Performing local search or not will **NOT** affect the result of 2D classification. However, it gives you a higher resolution density map for examining the detail of the 2D density map. You may turn it off when the computing resource is limited.

Number of Classes It stands for the number of classes you want the images to be classified into.

Initial Resolution (Angstrom) It is recommended to start from lower resolution for achieving ideal result of classification.

Symmetry Symmetry has **NO** effect on 2D classification.

Initial Model It is recommended to use a blank initial model in 2D classification. **Please leave it empty.**

Calculate FSC Using Core Region It is not supported in 2D classification. Please turn it off, otherwise a warning will be raised and **thunder** will turn it off forcefully.

Calculate FSC Using Masked Region It is not supported in 2D classification. Please turn it off, otherwise a warning will be raised and **thunder** will turn it off forcefully.

Particle Grading It is not recommended to use particle grading in 2D classification, because the importance of "noisy" particles may be overlooked when particle grading is turned on.

Performing Reference Mask It is **NOT** supported to use provided mask in 2D classification. If so, a fatal error will occur. Please turn it off.

4.2 3D Classification

The next step of cryo-EM single particle analysis is 3D classification for removing particles belong to "wrong" conformation.

You can find a demo version of this JSON parameter file for 3D classification named `demo_3D.json` in this package. There are some options worth noticed in this JSON parameter file. They are listed below.

Local Search Performing local search or not will **NOT** affect the result of 3D classification. However, it gives you a higher resolution density map for examining the detail of the 2D density map. You may turn it off when the computing resource is limited.

Number of Classes It stands for the number of classes you want the images to be classified into.

Initial Resolution (Angstrom) It is recommended to start from lower resolution for achieving ideal result of classification.

Particle Grading It is **NOT** recommended to use particle grading in 3D classification, because the importance of "noisy" particles may be overlooked when particle grading is turned on.

4.3 3D Refinement

The final step of cryo-EM single particle analysis is 3D refinement for achieving high resolution density map. You may turn on **particle grading** and **CTF search** for obtaining more information in density map.

You can find a demo version of this JSON parameter file for 3D classification named `demo.json` in this package. There are some options worth noticed in this JSON parameter file. They are listed below.

CTF Search You can refine CTF parameters using CTF search. It may cost some computing resource.

Particle Grading It is recommend to turn on particle grading in refinement.