STEMsalabim Documentation and Manual

Dr. Jan Oliver Oelerich

November 14, 2016

This is the reference manual of the STEMsalabim (S)TEM image simulator. It includes instructions how to use the software, as well as some technical implementation details. Users interested only in running the software may jump to section 4.

Contents

1 Introduction

The STEMsalabim software aims to provide accurate scanning transmission electron microscopy (STEM) image simulation of a specimen whose atomic structure is known. It implements the frozen lattice multislice algorithm as described in great detail in the book *Advanced computing in electron microscopy* by Earl J. Kirkland [1].

While there are multiple existing implementations of the same technique, none of them is suitable for running on high-performance computing (HPC) clusters, which is required in order to simulate large supercells or large sets of simulations, e.g. for parameter sweeps.

The purpose of STEMsalabim is to fill this gap by providing a multislice implementation that is well parallelizable both within and across computing nodes, using a mixture of threaded parallelization and message passing interface (MPI).

Section 2 includes some descriptions of implementation of the algorithm. Instructions how to compile and link the code are given in section 3 and usage instructions can be found in section 4.

2 Implementation details

3 Installation

STEMsalabim uses the CMake build system for compilation, discovering compilers and libraries, and linking. Please make sure that the prerequisites for compiling are all matched.

3.1 Requirements

STEMsalabim depends on the following freely available libraries. Note, that the NetCDF C++ library is just an interface to the NetCDF C library and, therefore, depends on it, which in turn depends on HDF5.

library	**website**
boost C++ library	`http://www.boost.org/`
NetCDF C++	`http://www.unidata.ucar.edu/software/netcdf/`
libConfig	`http://www.hyperrealm.com/libconfig/`
FFTW	`http://www.fftw.org/`

Furthermore, an implementation of MPI is required, for example OpenMPI.

For compilation, you need a C++ compiler supporting the C++11 standard (e.g., g++ > 5.0) and CMake > 3.3.

On most Linux distributions, you can install the requirements using a package manager. For example, on Debian the commands would look like this:

```
apt-get install \
    gcc \
    cmake \
    libnetcdfc++4 \
    libconfig-dev \
    libfftw3-dev \
    libboost-dev
```

We have tested the code only with rather new versions of the respective libraries.

3.2 Compilation

Extract the code archive to some folder on your hard drive, e.g.

```
cd /tmp
tar xzf stemsalabim-1.0.tar.gz
```

The, create a build directory and run CMake to generate the build files:

```
mkdir /tmp/stemsalabim-build
cd /tmp/stemsalabim-build
cmake ../stemsalabim-1.0
```

Please refer to the CMake documentation for instructions how to specify library paths and other environment variables, in case the above command failed. For instance, we are using the following command to compile the program:

```
cmake ../stemsalabim-1.0
    -DBOOST_ROOT=/my/custom/boost/     \# custom location of boost libs
    -DFFTW_ROOT=/my/custom/fftw/       \# custom location of FFTW
    -DCMAKE_INSTALL_PREFIX=/usr/local  \# custom installation location
    -DCMAKE_EXE_LINKER_FLAGS='-Wl,-rpath,\
        /my/custom/boost/:\
        /my/custom/lib64:\
        /my/custom/lib' \
    -DCMAKE_CXX_COMPILER=/usr/bin/g++
```

In the above example, some custom library paths are given, and the program's run-time search path is specified. If cmake doesn't detect the correct compiler automatically, you can specify it with $-$DCMAKE_CXX_COMPILER=.

Having generated the necessary build files with CMake, simply compile the program using make and move it to the install location with make install:

```
make -j8   # use 8 cores for compilation
make install
```

4 Usage

STEMsalabim is executed on the command line and configured via input configuration files given in libConfig syntax. The structure of the configuration files is explained below in section 4.1. A few parameters can be changed via command line parameters, which are listed in section 4.2.

A typical execution of STEMsalabim would be as follows:

```
stemsalabim --params=./my_config_file.cfg --num-threads=32
```

This would run a simulation detailed in ./ my_config_file .cfg in parallel on 32 threads on a single machine.

For MPI parallelization, STEMsalabim must be called via mpirun or mpiexec, depending on the MPI implementation:

```
mpirun -n 32 stemsalabim --params=./my_config_file.cfg --num-threads=1
```

This command will run the simulation in parallel on 32 MPI processors without spawning additional threads.

For hybrid parallelization, make sure that on each machine only a single MPI process is spawned and that there is no CPU pinning active, i.e., STEMsalabim needs to be able to spawn threads on different cores.

For example, if we wanted to run a simulation in parallel on 32 machines and on each machine on 16 cores, we would issue (using OpenMPI)

```
mpirun -n 32 --bind-to none --map-by ppr:1:node:pe=16 \
    stemsalabim
        --params=./my_config_file.cfg \
        --num-threads=16 \
        --package-size=160
```

The options −−bind−to none −−map−by ppr:1:node:pe=16 tell MPI not to bind the process to anything (for example, a single core) and to reserve 16 threads for each instance. Please refer to the manual of your MPI implementation to figure out how exactly to run the software.

The −−package−size parameter tells STEMsalabim, how many work packages are calculated by each MPI process until the results are communicated with the master process. The parameter is explained in more detail in section 2.

4.1 Parameter files

The configuration file that STEMsalabim expects for the command line parameter −−params is formatted using the simple JSON-like syntax of libConfig syntax. An example file is given below.

```
application: {
    random_seed = 0;
    verbose = false;
    precision = "single";
}

simulation: {
    title = "benchmarksmall";
    bandwidth_limiting = true;
}

probe: {
    c5 = 5e6;
    cs = 2e3;
    defocus = [6.0, -2.0, 7.0];
    astigmatism_ca = 0;
    astigmatism_angle = 0;
    min_apert = 0.0;
    max_apert = 24.0;
    beam_energy = 200.0;
}

detector: {
    angles = (1.0, 300.0, 300);
    interval_exponent = 1.0;
}

specimen: {
    max_potential_radius = 0.3;
    crystal_file = "input.xyz";
}

grating: {
    density = 360;
    slice_thickness = 0.5430;
}

scan: {
    density = 40;
    x = (0.0, 1.0);
    y = (0.0, 1.0);
}

output: {
    output_file = "benchmarksmall_80x80.nc";
    average_configurations = true;
    average_defoci = true;
    save_slices_every = 0;
}

frozen_phonon: {
    number_configurations = 15;
}
```

We will go through the parameters below in the form of *category.parameter*.

application.random_seed (unsigned int, default=0) In the frozen lattice approxima-

tion, the atoms are randomly dislocated from their equilibrium position. The random seed for that can be given here. If 0, a random seed is generated by the program. This parameter should be 0 unless the user wants to reproduce previous results.

application.verbose (boolean, default=false) Set this to true to get some more output on stdout.

application.precision (string, default="single") Set to d or double or high to use double precision for the calculation. Note, that this increases computer time, memory consumption and output file size.

simulation.title (string, default="") The title of the simulation, as saved in the output nc file.

simulation.bandwidth_limiting (boolean, default="true") Enforce cylindrical symmetry on all wave functions/operations.

probe.c5 (number, unit=nm, default=5e7) Fifth order spherical aberrations coefficient.

probe.cs (number, unit=nm, default=2e4) Third order spherical aberrations coefficient.

probe.defocus (array, default=2.0) Probe defocus. STEMsalabim can calculate a defocus series to model chromatic aberrations or just simulate a single defocus. In the latter case, only a single number with unit nm is specified. In case of a defocus series, a triplet of parameters is given, where the first one is the full-width-half-maximum of the normal distribution of defocus spread in nm, the second one is the center of the gaussian in nm and the third parameter is the number of defoci calculated. For example, defocus=[6.0, −2.0, 1.] would lead to the following defocus values being calculated: -14nm, -10nm, -6nm -2nm, 2nm, 6nm, 10nm.

Note: Due to a restriction of the libConfig library, all three parameters must be given as floating point values, i.e., with a decimal point.

probe.astigmatism_ca (number, unit=nm, default=0) Two-fold astigmatism.

probe.astigmatism_angle (number, unit=mrad, default=0) two-fold astigmatism angle.

probe.min_apert (number, unit=mrad, default=0) Minimum numerical aperture of the objective.

probe.max_apert (number, unit=mrad, default=24.0) Maximum numerical aperture of the objective.

probe.beam_energy (number, unit=kV, default=200) Electron beam energy.

detector.angles (array, default=[1.0, 300.0, 300.]) The detector angle grid. Intensity is collected for angles between the first and the second value ($\theta_{\min}$ and $\theta_{\max}$,

both are given in mrad), and the number of bins N is given in the third parameter.

detector.interval_exponent (number, default=1.0) The detector grid can be chosen as not linear, to make the grid of collected intensities at smaller angles finer than at larger angles. The ith grid point between $\theta_{\min}$ and $\theta_{\max}$ is calculated by $\theta_i = \theta_{\min} + (i/N)^p(\theta_{\max} - \theta_{\min})$, where p is the interval exponent and N is the number of grid points as given by **detector.angles**.

specimen.max_potential_radius (number, unit=nm, default=0.3) Distance, after which an atomic potential is cut off during generation of the slice's transmission functions.

specimen.crystal_file (string, default="") Path to the file containing the atomic coordinates. Please read section 4.3 to learn about its format.

grating.density (number, unit=1/nm, default=360.0) The density for the real space and fourier space grids. This number multiplied by the supercell size in x and y direction gives the number of sampling grid points for the calculation. This also determines the maximum angle $\alpha = k\lambda$ that is described by the k-space grids.

grating.slice_thickness (number, unit=nm, default=0.2715) The thickness of the slices in the multislice algorithm.

scan.density (number, unit=1/nm, default=40.0) The sampling density for the electron probe scanning. This number multiplied by the supercell size in x and y direction gives the number of position, i.e., scan points, that are calculated. This equals the pixel dimensions of the resulting STEM images. It is affected by the **scan.x** and **scan.y** parameters.

scan.x (array, default=[0.0,1.0]) The relative coordinates, between which the supercell is scanned by the electron probe. When 0.0 and 1.0, the whole x width of the supercell is scanned. This parameter can be used to exclude edge effects.

scan.y (array, default=[0.0,1.0]) The relative coordinates, between which the supercell is scanned by the electron probe. When 0.0 and 1.0, the whole y width of the supercell is scanned. This parameter can be used to exclude edge effects.

output.output_file (string, default="") Path to the results file. Please read section 4.4 to learn about its format.

output.average_configurations (boolean, default=true) Whether or not to average over frozen lattice configurations during writing of the output file.

output.average_defoci (boolean, default=true) Whether or not to average over defoci of a defocus series (see **probe.defocus**) during writing of the output file.

output.save_slices_every (number, default=1) If other than 1, intensity after every Nth slice is saved to the output file. The intensity at the bottom of the sample is always included. If set to 0, only the intensity at the very bottom of the sample is written to the output file.

frozen_phonon.number_configurations (number, default=1) Number of frozen lattice configurations to calculate.

4.2 Command line parameters

STEMsalabim accepts a few parameters specified on the command line, of which only −−params is required.

–help (flag) Display a help message with a brief description of available command line parameters.

–params (string, required) Path to the configuration file as explained in section 4.1.

–num-threads (int, default=1) Number of threads per MPI core. Note, that STEMsalabim will do nothing if only parallelized via threads and −−params=1, as thread 0 of the master MPI process does only book keeping.

–package-size (int, default=10) The amount of tasks that are sent to an MPI process, calculated and then sent back. This should be scaled with the number of threads each MPI process spawns. A good value seems to be 10 × num_threads.

–skip-simulation (flag) When specified, the STEM simulation is skipped, but otherwise everything else is executed. This can be used to quickly generated output files with, say, a number of frozen lattice configurations or so.

–num-configurations (int) Override the number of frozen lattice configurations.

–defocus (single or three floats) Override the value of the defocus setting (see section 4.1). If this is a single float number, a single defocus is calculated. If three floats are given, the syntax is similar to the probe.defocus parameter (see above).

–output-file (string) Override the path to the output file, parameter output. output_file .

–crystal-file (string) Override the path to the crystal file, parameter specimen. crystal_file .

–title (string) Override the simulation title, parameter simulation. title .

4.3 Crystal file format

The crystal file must be in the form of an xyz file format, which is a text file with the content as follows.

The **first line** contains the number of atoms as a single number.

The **second line** contains the cell dimension in x,y,z direction as floating point numbers in units of nm, separated by a space.

The single atoms are given from the **third line onwards**, with each line corresponding to one atom. Each line must have 5 columns: The atomic species as its elemental abbreviation (e.g., Ga for gallium), the x,y,z coordinates as floating point numbers in units of nm, and the mean square displacement for the frozen lattice dislocations in units of nm^2.

The example file below is a perfectly valid crystal file:

```
5
1.0  2.0  10.0
Ga   0.0   0.0   0.0   1e-5
P    0.2   0.1   0.0   2e-5
Ga   0.0   0.0   1.0   1e-5
P    1.2   0.1   0.0   2e-5
O    1.0   2.0  10.0   0.0
```

4.4 Output file format

The output of a STEMsalabim simulation is in NetCDF format. The file is divided into the following groups:

runtime This group contains some information about the STEMsalabim version the calculation was carried out with, start and stop time, its title and the content of the parameter file.

AMBER This group contains the atomic coordinates, species, displacements, radii, etc. for the complete crystal for each single calculated frozen lattice configuration, as well as for each calculated defocus value. The AMBER group content is compatible with the AMBER specifications. A STEMsalabim NetCDF file can be opened seamlessly with the ovito crystal viewer.

intensities This group contains the simulated beam intensities, the coordinates of the electron probe beam during scanning, the detector angle grid that is used, coordinates of the slices as used in the multislice algorithm and the calculated defoci.

params All simulation parameters are collected in the params group as attributes.

For a detailed view of the structure, we suggest using the ncdump utility: ncdump −h some_results_file.n
As the underlying file format of NetCDF is HDF5, you may use any other HDF5 viewer to have a look at the results.

There are NetCDF bindings for most popular programming languages. In MATLAB, we recommend using h5read() and the other HDF5 functions.

5 Tips and Tricks

coming soon.

References

[1] Earl J. Kirkland. *Advanced computing in electron microscopy*. 2010, pp. 47–50. ISBN: 978-1-4419-6532-5. DOI: 10.1007/978-1-4419-6533-2. URL: http://link.springer.com/10.1007/978-1-4419-6533-2.