Chemical crystallography using `cctbx.small_cell_process`

This page reproduces the structure determination of mithrene from XFEL diffraction at SACLA. About 350GB of storage are required. The whole structure determination took about 3 hours on a 64-core (Opteron 6380) Linux server.

References

The *small_cell* indexing algorithm was described in Brewster, 2015: https://doi.org/10.1107/S1399004714026145

The experimental structure determination in this README is described in Schriber and Paley, 2021, Nature (in press).

Installation of software

▼ Developer installation

For the work described in Schriber et al., DIALS/CCTBX and dependencies were installed as described in the "general build" section here: https://github.com/cctbx/cctbx_project/tree/master/xfel/conda_envs

GSASII must also be installed in the conda environment. Instructions will be printed when calling `cctbx.xfel.candidate_cells` for the first time.

▼ Docker installation

To promote future-safety, a working Docker image will be available from Dockerhub as `dwpaley/cctbx:20211012`. Installation instructions are given for a Macbook running MacOS 10.15.7, although processing this full dataset is a space- and compute-intensive task for a personal laptop.
Nils De Moor is acknowledged for a helpful post on configuring GUI applications via XQuartz.

- Install Docker Desktop as described here: https://docs.docker.com/desktop/mac/install/
- Install XQuartz from here: https://www.xquartz.org. A restart might be needed.
- Start XQuartz, visit Preferences and enable "Security->Allow connections from network clients".
- Install and start `socat` to expose a port for the X server:

```
$ socat TCP-LISTEN:6000,reuseaddr,fork UNIX-CLIENT:\"$DISPLAY\" &
$ ifconfig en0 |grep '\<inet\>' | awk '{print $2}'
192.168.1.222
$ docker run -e DISPLAY=192.168.1.222:0 gns3/xeyes
```

A pair of eyeballs should be displayed. This confirms Docker can contact the X server. When Docker applications will need to access the display, follow the example of the last two steps above to set the `DISPLAY` environment variable.

- Clone the image (approx. 7 GB): `$ docker clone dwpaley/cctbx:20211012`
- Make sure enough resources are available for the Docker machine. I provided 3/4 of my system or 12 cores and 24GB memory.

Availability of data

The raw data collected at SACLA will be made available via CXIDB. For now contact the author: dwpaley at lbl dot gov. Runs 3133 through 3214 are sufficient to determine the structure. A carefully refined detector metrology file, `step2_refined2.expt` , is also required.

Extract the data and export an environment variable pointing to the path:

```
$ tar -xvzf *.tar.gz
$ export SACLA_DATA=$PWD/data
```

Run spotfinding

We will use the DIALS spotfinder on 9 selected runs with a lot of hits; this is enough spots to synthesize a high-quality powder pattern for indexing.

With `run.py` containing:

```
import sys, os
i = int(sys.argv[1])
j = int(sys.argv[2])
phil = sys.argv[3]
cores = sys.argv[4]
root_path = os.getenv('SACLA_DATA')
data_path = os.path.join(root_path, 'data')
geom_path = os.path.join(root_path, 'geom', 'step2_refined2.expt')
h5_path = os.path.join(data_path, f'78{i}-{j}', f'run78{i}-{j}.h5')
out_path = os.path.join(f'{i}-{j}', 'out')
log_path = os.path.join(f'{i}-{j}', 'log')
cmd = \
  f"cctbx.small_cell_process {phil} {h5_path} output.output_dir={out_path}
\
  output.logging_dir={log_path} input.reference_geometry={geom_path}"
cmd_mpi = f"mpirun -n {cores} {cmd} mp.method=mpi"
print (cmd_mpi)
os.makedirs(out_path)
os.makedirs(log_path)
os.system(cmd_mpi)
```

and `spotfind.phil` containing:

```
output {
  experiments_filename = %s_imported.expt
  strong_filename = %s_strong.refl
}
dispatch.hit_finder {
  minimum_number_of_reflections = 3
  maximum_number_of_reflections = 40
}
dispatch.index=False
output.composite_output=True
spotfinder.filter.min_spot_size=3
```

do:

```
$ for i in {3192..3200}; do for j in {0..2}; do python run.py $i $j
spotfind.phil 64; done; done
```

(but adapt for the number of cores available).

▼ Docker instructions

Prepare `run.py` and `spotfind.phil` as above, plus a file `run.sh`:

```
#!/bin/sh

cd /cwd
source /img/build/conda_setpaths.sh
export SACLA_DATA=/data
python run_integrate.py $1 $2 spotfind.phil 12
```

We run the jobs in containers that have the data and output folders (on the local drive) exposed via bind mounts:

```
$ chmod +x run.sh
$ for i in {3192..3200}; do for j in {0..2}; do
>   docker run -v $PWD:/cwd -v $SACLA_DATA:/data dwpaley/cctbx:20211012
/cwd/run.sh $i $j
> done; done
```

Compute powder pattern from spots

Spotfinding will run on the 27 selected h5 files within a few minutes. Then prepare a single set of combined files:

```
$ mkdir combined
$ for d in 3*; do dials.combine_experiments $d/out/*imported.expt
$d/out/*strong.refl output.experiments=combined/$d.expt
output.reflections=combined/$d.refl reference_from_experiment.detector=0 &
done
$ cd combined; dials.combine_experiments *.expt *.refl
reference_from_experiment.detector=0
```

Plot the powder pattern:

```
$ cctbx.xfel.powder_from_spots combined.* output.xy_file=powder.xy
```

This should display an interactive matplotlib window. Manually identify about 20 d-spacings. Record them with 3 decimal places accuracy in a new file `peaks.txt` . Then run GSASII powder indexing through the `candidate_cells` wrapper:

```
$ cctbx.xfel.candidate_cells nproc=64 input.peak_list=peaks.txt
input.powder_pattern=powder.xy search.timeout=300
```

▼ Docker instructions

Start an interactive container with the display configured and the working directory bind-mounted:

```
$ ifconfig en0 |grep '\<inet\>' | awk '{print $2}'
192.168.1.225
$ docker run -it -e DISPLAY=192.168.1.225:0 -v $PWD:/cwd
dwpaley/cctbx:20211012
# source /img/build/conda_setpaths.sh
# cd /cwd
```

Then proceed with combining the spotfinding results, plotting the powder pattern, and searching for the unit cell as described above.

The correct cell should be in the top 5 candidates. We will index a single run with the candidate unit cells and crystal systems.

Trial indexing jobs

Make 5 copies of the following indexing phil file, named `1.phil` through `5.phil` :

```
dispatch {
  hit_finder {
    minimum_number_of_reflections = 3
    maximum_number_of_reflections = 40
   }
  }
dispatch {
  refine = True
}
output {
  composite_output=True
  experiments_filename = None
  strong_filename = None
}

spotfinder {
  filter {
    min_spot_size = 3
  }
}
refinement {
  parameterisation {
    crystal {
      fix = all *cell orientation
    }
  }
  reflections {
    outlier {
      algorithm = null auto mcd tukey sauter_poon
    }
  }
}
small_cell {
  powdercell = "1,2,3,90,90,90"
  spacegroup = "P2"
  high_res_limit = 1.2
  min_spots_to_integrate = 3
  faked_mosaicity = 0.1
  spot_connection_epsilon = 0.01
  d_ring_overlap_limit = None
}
```

Manually enter the unit cells and space groups of the top 5 candidates from candidate_cells.

With `run_index.py` containing:

```
import sys, os
i, j = 3192, 0
phil_root = sys.argv[1]
cores = sys.argv[2]
root_path = os.getenv('SACLA_DATA')
data_path = os.path.join(root_path, 'data')
geom_path = os.path.join(root_path, 'geom', 'step2_refined2.expt')
h5_path = os.path.join(data_path, f'78{i}-{j}', f'run78{i}-{j}.h5')
out_path = os.path.join(phil_root, 'out')
log_path = os.path.join(phil_root, 'log')
```

```
cmd = \
  f"cctbx.small_cell_process {phil_root}.phil {h5_path} output.output_dir=
{out_path} \
  output.logging_dir={log_path} input.reference_geometry={geom_path}"
cmd_mpi = f"mpirun -n {cores} {cmd} mp.method=mpi"
print (cmd_mpi)
os.makedirs(out_path)
os.makedirs(log_path)
os.system(cmd_mpi)
```

do

```
$ for n in 1 2 3 4 5; do python run_index.py $n 64; done
```

then count the resulting indexing hits:

```
$ for n in {1..5}; do
    echo
    echo $n
    egrep 'powdercell|spacegroup' $n.phil
    grep real_space_a $n/out/*refined.expt 2>/dev/null|wc -l
  done
```

▼ Docker instructions

Set up the phil files as above. Make a script `run.sh` containing the following:

```
#!/bin/sh

cd /cwd
source /img/build/conda_setpaths.sh
export SACLA_DATA=/data
python run_index.py $1 $2
```

And run the jobs in containers with the appropriate bind mounts:

```
$ chmod +x run.sh
$ for n in 1 2 3 4 5; do
>   docker run -v $PWD:/cwd -v $SACLA_DATA:/data dwpaley/cctbx:20211012
/cwd/run.sh $n 12
> done
```

Then count the indexing hits as described above.

Integration

After choosing the correct cell, make a new directory for integration scripts and results. Prepare this file `integrate.phil`:

```
dispatch {
  hit_finder {
    minimum_number_of_reflections = 3
    maximum_number_of_reflections = 40
   }
  }
dispatch {
  refine = True
}
output {
  composite_output=True
  experiments_filename = None
  strong_filename = None
}

spotfinder {
  filter {
    min_spot_size = 3
  }
}
refinement {
  parameterisation {
    crystal {
      fix = all *cell orientation
    }
  }
  reflections {
    outlier {
      algorithm = null auto mcd tukey sauter_poon
    }
  }
}
small_cell {
  powdercell = "5.93762, 7.32461, 29.202, 90, 95.4404, 90"
  spacegroup = "C2"
  high_res_limit = 0.8
  min_spots_to_integrate = 3
  faked_mosaicity = 0.1
  spot_connection_epsilon = 0.004
  d_ring_overlap_limit = None
}
```

And this script `run.py` :

```
import sys, os
i = int(sys.argv[1])
j = int(sys.argv[2])
phil = sys.argv[3]
cores = sys.argv[4]
root_path = os.getenv('SACLA_DATA')
data_path = os.path.join(root_path, 'data')
geom_path = os.path.join(root_path, 'geom', 'step2_refined2.expt')
h5_path = os.path.join(data_path, f'78{i}-{j}', f'run78{i}-{j}.h5')
out_path = os.path.join(f'{i}-{j}', 'out')
log_path = os.path.join(f'{i}-{j}', 'log')
cmd = \
  f"cctbx.small_cell_process integrate.phil {h5_path} output.output_dir=
{out_path} \
  output.logging_dir={log_path} input.reference_geometry={geom_path}"
cmd_mpi = f"mpirun -n 64 {cmd} mp.method=mpi"
print (cmd_mpi)
os.makedirs(out_path)
os.makedirs(log_path)
os.system(cmd_mpi)
```

Prepare a todo script and run it: [note 1]

```
$ for m in {3133..3214}; do for n in {0..2}; do
>   echo "python run.py $m $n integrate.phil 64" >> todo.sh
> done; done
$ source todo.sh
```

▼ Docker instructions

Prepare `run.py` and `integrate.phil` as above, plus a file `run.sh` :

```
#!/bin/sh

cd /cwd
source /img/build/conda_setpaths.sh
export SACLA_DATA=/data
python run.py $1 $2 integrate.phil 12
```

Run the jobs as described above for spotfinding:

```
$ chmod +x run.sh
$ for i in {3133..3214}; do for j in {0..2}; do
>   docker run -v $PWD:/cwd -v $SACLA_DATA:/data dwpaley/cctbx:20211012
/cwd/run.sh $i $j
> done; done
```

Optionally, you can instead echo the `docker` commands to a `todo.sh` script as above, e.g. for running multiple jobs under control of `parallel` or similar.

Merging

After integration, merging is performed in two steps:

```
$ mkdir merging; cd merging
$ cat > mark1.phil
input.path=../3*/out
dispatch.step_list = input balance model_scaling modify errors_premerge
scale statistics_unitcell statistics_beam model_statistics
statistics_resolution group errors_merge statistics_intensity merge
statistics_intensity_cxi
scaling.algorithm=mark1
scaling.unit_cell=5.93762 7.32461 29.202 90 95.4404 90
scaling.space_group=C2
merging.d_min=1.15
merging.merge_anomalous=True
merging.error.model=errors_from_sample_residuals
statistics.n_bins=20
output.prefix=mark1
output.do_timing=True
output.output_dir=.

$ cat > mark0.phil
input.path=../3*/out
dispatch.step_list = input balance model_scaling modify errors_premerge
scale statistics_unitcell statistics_beam model_statistics
statistics_resolution group errors_merge statistics_intensity merge
statistics_intensity_cxi
scaling.model=mark1_all.mtz
merging.d_min=1.15
merging.merge_anomalous=True
output.do_timing=True
statistics.n_bins=20
merging.error.model=errors_from_sample_residuals
output.prefix=mark0
output.do_timing=True
output.output_dir=.
```

```
$ mpirun -n 32 cctbx.xfel.merge mark1.phil
$ mpirun -n 32 cctbx.xfel.merge mark0.phil
$ iotbx.reflection_file_converter mark0_all.mtz --label="Iobs,SIGIobs" --
shelx=mark0.hkl
```

At this point, the file mark0.hkl can be used for structure solution. An ins file must be prepared by hand. The following would be suitable for mithrene in C2/c:

```
TITL mithrene
CELL 1.032066 5.93762 7.32461 29.202 90 95.4404 90
ZERR 8 0 0 0 0 0 0
LATT 7
SYMM -X,+Y,0.5-Z
SFAC C H Se Ag
DISP Ag -0.2289 2.1223 13899.9697
DISP C 0.0087 0.0039 30.8059
DISP H -0 0 0.6385
DISP Se -2.5865 0.5775 2790.4387
UNIT 80 72 8 8
HKLF 4
```

To perform a final round of scaling with a partially completed reference structure (here combined with reindexing to resolve an indexing ambiguity), save the structure in standard cif format and do this:

```
$ cat > rtr.phil
input.path=../3*/out
dispatch.step_list = input balance model_scaling modify
modify_reindex_to_reference errors_premerge scale statistics_unitcell
statistics_beam model_statistics statistics_resolution group errors_merge
statistics_intensity merge statistics_intensity_cxi
scaling.model=full.cif
merging.d_min=1.15
merging.merge_anomalous=True
output.do_timing=True
statistics.n_bins=20
merging.error.model=errors_from_sample_residuals
output.prefix=rtr
output.do_timing=True
output.output_dir=.

$ mpirun -n 32 cctbx.xfel.merge rtr.phil
$ iotbx.reflection_file_converter rtr_all.mtz --label="Iobs,SIGIobs" --
shelx=rtr.hkl
```

The resulting file `rtr.hkl` contains the final intensities for structure refinement.

▼ Docker instructions

Merging is most convenient in an interactive container:

```
$ docker run -it -v $PWD:/cwd dwpaley/cctbx:20211012
# source /img/build/conda_setpaths.sh
# cd /cwd
```

Then proceed with the steps above (but be sure to adjust the # of cores for mpirun jobs).

Notes

[Note 1]: To optimize disk access, running a few smaller parallel jobs may be faster. The utility GNU Parallel is available from conda_forge:

```
$ conda install parallel -c conda-forge
```

and the todo list can be piped into `parallel`:

```
$ cat todo.sh | parallel -j 4
```

to run 4 concurrent jobs (in which case you should modify the `mpirun` command to use 16 tasks apiece).