KinCat

Release 1.0

Kyungjoo Kim, Craig Daniels, Habib Najm, Nathan Roberts

Feb 27, 2023

GETTING STARTED:

CHAPTER

ONE

CONTENTS

1.1 Introduction

KinCat is an open source software library for parallel kinetic Monte Carlo (KMC) simulations focusing on heterogeneous catalysis applications. In particular, we implement lattice KMC and uses Kokkos for its performance portability layer.

—

1.2 Building KinCat

For convenience, we explain how to build the KinCat using the following environment variables that a user can modify according to their working environments. Note that cmake requires to use a separate build directory protecting the source code.

```
# repositories
export KINCAT_REPOSITORY_PATH=/where/you/clone/kincat/git/repo
export KOKKOS_REPOSITORY_PATH=/where/you/clone/kokkos/git/repo
export GTEST_REPOSITORY_PATH=/where/you/clone/gtest/git/repo

# build directories
export KINCAT_BUILD_PATH=/where/you/build/kincat
export KOKKOS_BUILD_PATH=/where/you/build/kokkos
export GTEST_BUILD_PATH=/where/you/build/gtest

# install directories
export KINCAT_INSTALL_PATH=/where/you/install/kincat
export KOKKOS_INSTALL_PATH=/where/you/install/kokkos
export GTEST_INSTALL_PATH=/where/you/install/gtest
```

1.2.1 System Requirements

- CMake version 3.16 and higher.
- Compiler supporting C++14 and higher standards.
- Boost library 1.75 and higher.
- Jupyter notebook (or lab) and Python3 for visualization and post-processing.
- sphinx for documentation.
- OpenMPI for running KinCat with multiple data.

Mac OSX

We use macports for the standard software distributions of required tools and libraries. Install or check the following tools are installed using macports.

```
sudo port install cmake clang-13 boost python310 py38-sphinx py38-sphinx_rtd_theme␣
↪openmpi-clang13

which mpirun-openmpi-clang13
mpirun-openmpi-clang13 is /opt/local/bin/mpirun-openmpi-clang13

which clang++-mp-13
clang++-mp-13 is /opt/local/bin/clang++-mp-13

which cmake
cmake is /opt/local/bin/cmake

export MYCXX=clang++mp-13

pip install jupyterlab
```

Weaver

Weaver is equipped with IBM Power9 processor accelerated by NVIDIA V100 GPU. I use the following modules. These modules can be changed as the system is upgraded.

```
module purge
module load devpack/20210226/openmpi/4.0.5/gcc/7.2.0/cuda/10.2.2
module swap cmake cmake/3.19.3
module swap boost boost/1.75.0

which gcc
/home/projects/ppc64le/gcc/7.2.0/bin/gcc

echo ${BOOST_ROOT}
/home/projects/ppc64le-pwr9/spack/opt/spack/linux-rhel7-power9le/gcc-7.2.0/boost-1.75.0-
↪qf3d47g2bt3dhlbruldwpqfu3rqkrdtk

which cmake
/home/projects/ppc64le/cmake/3.19.3/bin/cmake
```

1.2.2 How to Build

First, clone Kokkos, GTEST and KinCat code repositories.

```
git clone https://github.com/kokkos/kokkos.git ${KOKKOS_REPOSITORY_PATH};
git clone https://github.com/google/googletest.git ${GTEST_REPOSITORY_PATH}
git clone ssh://kyukim@getz.ca.sandia.gov:1867/home/gitroot/kmc ${KINCAT_REPOSITORY_PATH}
↪;
```

Kokkos

This build Kokkos on Intel Haswell architectures and install Kokkos to ${KOKKOS_INSTALL_PATH}. For more details, see [Kokkos github pages](https://github.com/kokkos/kokkos). We can use this script for OSX.

```
cd ${KOKKOS_BUILD_PATH}
cmake \
  -D CMAKE_INSTALL_PREFIX="${KOKKOS_INSTALL_PATH}" \
  -D CMAKE_CXX_COMPILER="${MYCXX}"  \
  -D Kokkos_ENABLE_SERIAL=ON \
  -D Kokkos_ENABLE_OPENMP=ON \
  -D Kokkos_ENABLE_DEPRECATED_CODE=OFF \
  -D Kokkos_ARCH_HSW=ON \
  ${KOKKOS_REPOSITORY_PATH}
make -j install
```

On Weaver, we compile Kokkos for NVIDIA GPUs. Note that we use Kokkos nvcc_wrapper as its compiler instead of directly using the nvcc compiler. The architecture flag indicates that the host architecture is IBM Power9 and the GPU architecture is Volta70 generation.

```
cd ${KOKKOS_BUILD_PATH}
cmake \
  -D CMAKE_INSTALL_PREFIX="${KOKKOS_INSTALL_PATH}" \
  -D CMAKE_CXX_COMPILER="${KOKKOS_REPOSITORY_PATH}/bin/nvcc_wrapper"  \
  -D Kokkos_ENABLE_SERIAL=ON \
  -D Kokkos_ENABLE_OPENMP=ON \
  -D Kokkos_ENABLE_CUDA:BOOL=ON \
  -D Kokkos_ENABLE_CUDA_UVM:BOOL=OFF \
  -D Kokkos_ENABLE_CUDA_LAMBDA:BOOL=ON \
  -D Kokkos_ENABLE_DEPRECATED_CODE=OFF \
  -D Kokkos_ARCH_VOLTA70=ON \
  -D Kokkos_ARCH_POWER9=ON \
  ${KOKKOS_REPOSITORY_PATH}
make -j install
```

GTEST

We use GTEST as our testing infrastructure. With the following cmake script, the GTEST can be compiled and installed.

```
cd ${GTEST_BUILD_PATH}
cmake \
  -D CMAKE_INSTALL_PREFIX="${GTEST_INSTALL_PATH}" \
  -D CMAKE_CXX_COMPILER="${MYCXX}"  \
  ${GTEST_REPOSITORY_PATH}
make -j install
```

KinCat

KinCat and link with Kokkos and Gtest. The following script shows how to compile KinCat on OSX linking with TPLs explained in the above.

```
cd ${KINCAT_BUILD_PATH}
cmake \
  -D CMAKE_INSTALL_PREFIX=${KINCAT_INSTALL_PATH} \
  -D CMAKE_CXX_COMPILER="${MYCXX}" \
  -D CMAKE_CXX_FLAGS="-g" \
  -D CMAKE_EXE_LINKER_FLAGS="" \
  -D CMAKE_BUILD_TYPE=RELEASE \
  -D KINCAT_SITE_TYPE="char" \
  -D KINCAT_ENABLE_DEBUG=OFF \
  -D KINCAT_ENABLE_VERBOSE=ON \
  -D KINCAT_ENABLE_TEST=ON \
  -D KINCAT_ENABLE_EXAMPLE=ON \
  -D KOKKOS_INSTALL_PATH="${KOKKOS_INSTALL_PATH}" \
  -D GTEST_INSTALL_PATH="${GTEST_INSTALL_PATH}" \
  ${KINCAT_REPOSITORY_PATH}/code/src
make -j install
export KINCAT_INSTALL_PATH=${KINCAT_INSTALL_PATH}
```

To install KinCat on Weaver (GPU platform), replace the c++ compiler with nvcc wrapper providing `-D CMAKE_COMPILER="${KOKKOS_INSTALL_PATH}/bin/nvcc_wrapper` instead. A successful installation creates following directory structure in `${KINCAT_INSTALL_PATH}`. Note that the `site_type` is determined at compile time. In the above cmake configuration, the type is set `char` and the max number of species in KinCat is 256. For a bigger simulation, users can set this `short` or `int`.

```
- bin
  - kincat.x: an executable for solving single problem
  - kincat-batch.x: an executable for solving multiple problem with batch parallelism
  - plot-dump.ipynb: visualization for jupyter notebook
- example
  - RuO2
    - input-RuO2.json: main user input specifying solver interface
    - input-RuO2-{0,1}.json: main user inputs for solving multiple samples with mpirun
    - input-batch-RuO2.json: main user input for solving multiple samples with batch␣
↪parallelism
    - input-dictionary-RuO2-full.json: dictionary of full configurations (auto-generated␣
↪from the preprocessor)
    - input-dictionary-RuO2.json: dictionary of reduced configurations (auto-generated␣
↪from the preprocessor)
```

(continues on next page)

(continued from previous page)

```
    - input-rates-RuO2.json: user input rates (could be auto-generated from a kinetic␣
↪model file or RMG)
    - input-rates-override-RuO2.json: user input rates to override rates for certain␣
↪processes of different samples
  - ToyKitten
    - input-toycat.json: Craig, include the main input file
    - input-dictionary-toycat-full.json: dictionary of full configurations
    - input-dictionary-toycat.json: Craig, include the reduced configuration input
    - input-rates-toycat.json: user input rates (could be auto-generated from a kinetic␣
↪model file or RMG)
- include
  - kincat
    - header files
- lib (or lib64)
  - cmake: cmake environment when other software interface KinCat via cmake
  - libkincat.a
- unit-test
  - kincat-test.x: unit test executable (Craig, update unit tests)
  - test-files: sample files that will be used in test (Craig, put sample files for␣
↪testing)
```

1.3 Running KinCat

We can run the `kincat.x` executable in the `${KINCAT_INSTALL_PATH}/bin` directory. The following command line options are available for the version of today. There are two verbose options showing input parsing and main object constructed based on the parse information. Configurations and events specification can be different as KinCat required sorted list of configurations. The `verbose-parse` shows raw input from the input json file. The `verbose` option shows the object details used in KinCat. The verbosity has four levels: 0) show nothing, 1) basic information, 2) reserved for users verbose comments, 3) lengthy details, 4) show everything.

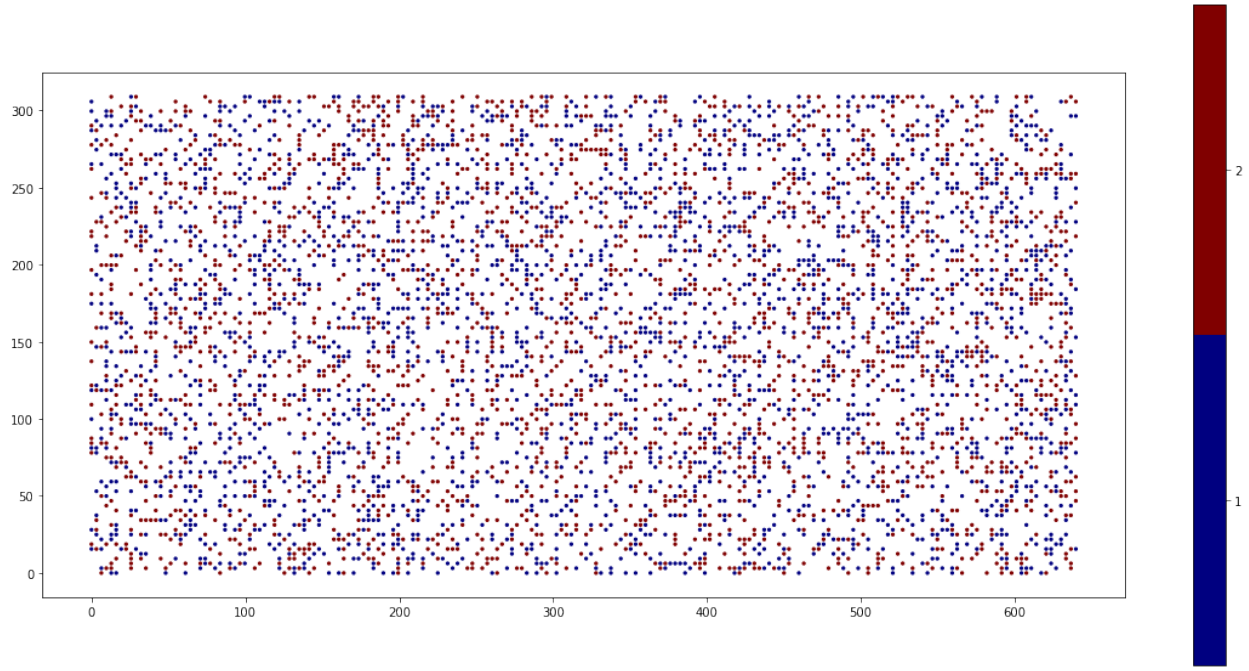
```
cd ${KINCAT_INSTALL_PATH}/bin
./kincat.x --help
 Usage: ./kincat.x [options]
   options:
   --echo-command-line           bool      Echo the command-line but continue as normal
   --help                        bool      Print this help message
   --input                       string    Input dictionary file name
                                           (default: --input=input.json)
   --verbose                     int       Verbosity level
                                           (default: --verbose=0)
   --verbose-iterate             int       Verbosity level in KMC iteration
                                           (default: --verbose-iterate=0)
   --verbose-parse               int       Verbosity level in parsing step (not yet␣
↪sorted)
                                           (default: --verbose-parse=0)
 Description:
    KinCat Main
```

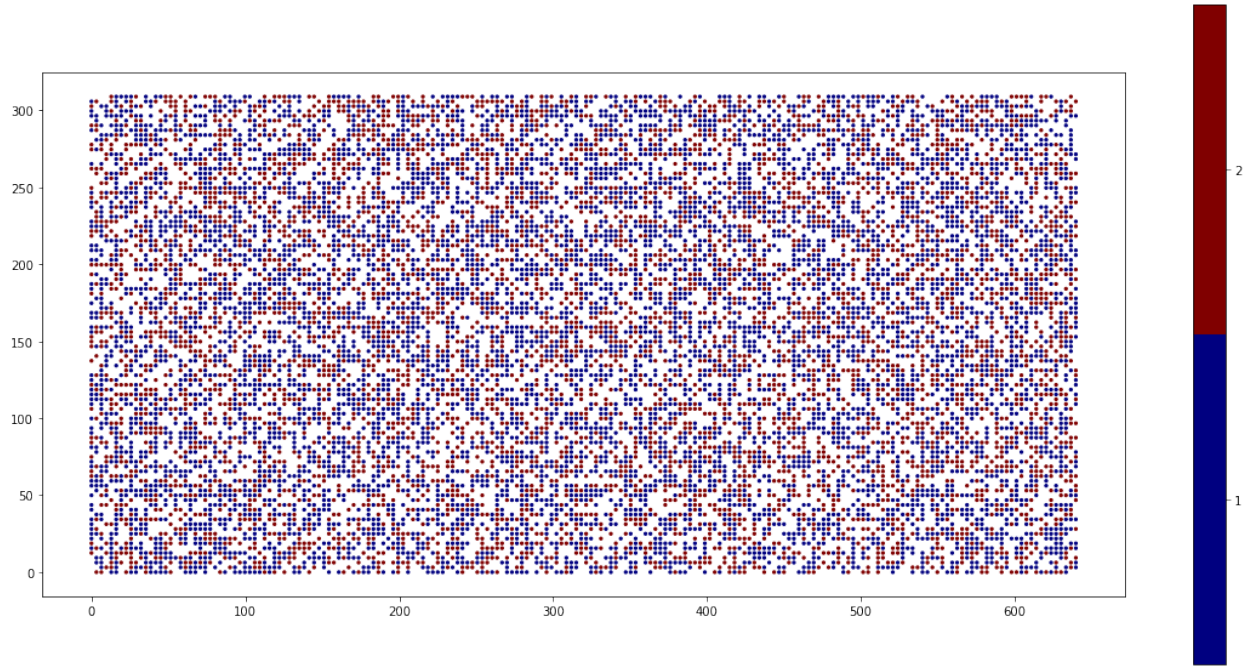
The following command runs the RuO2 example. This example uses `${KINCAT_INSTALL_PATH}` environment vari-

able. Users make sure that you export the variable in the current terminal. By default, we use Kokkos OpenMP as a default execution space for CPU processors while this example uses the serial residential time algorithm. Thus, users may want to set `export OMP_NUM_THREADS=1` not to carry some Kokkos overhead of using multiple threads.

```
cd ${KINCAT_INSTALL_PATH}/bin
./kincat.x --input="input.json" --verbose-iterate=1
-- Kokkos::OpenMP is used
solver_type = serial-rta
-- Lattice : Lattice
  -- number of species : 3
  -- domain : [40, 40, 2]
  -- edge vectors :
  [6.43,0]
  [0,3.12]
  -- basis :
  [0,0]
  [0.5,0]
-- ProcessDictionary : Dictionary
  -- variant ordering : (2, 4)
  -- configuration list : (45, 4)
  -- process instance list : (258, 3), constraints: (258, 2), rates : (258)
-- ProcessDictionary : Instance details
  -- process instance list : (258), process list : (22)
-- Solver : serial-rta
-- Dump : Dump
  -- filename dump : "dump.json"
  -- filename site: "dump-site.json"
-- Stats : Stats
  -- filename stats : "stats.json"
epoch = 0, t = 3.47263e-08
  -- # of events occured : 50
 ...
```

This KMC runs and produces the output files `dump.json` and `stats.json`. We can post-process these by using `jupyter-lab plot-dump.ipynb`. The figures show the initial random configuration and the final configuration of the example. More details of the input file specifications will come later.


300
250
200
150
100
50
0
0
100
200
300
400
500
600
2
1


300
250
200
150
100
50
0
0
100
200
300
400
500
600
2
1

1.3.1 Solving for Multiple Samples

For small and medium problem sizes, we recommend to use a batch parallel version of kincat i.e., `kincat-batch.x`. The specific use case of the batch parallelism will be explained later with the batch input file.

```
cd ${KINCAT_INSTALL_PATH}/bin
./kincat-batch.x --input="../example/RuO2/input-batch-RuO2.json"
```

For a larger problem size, we can solve multiple samples in parallel in a Sinlge Program Multiple Data (SPMD) style. The following example illustrates that the mpirun command launches two processes with two input files `input-RuO2-0.json` and `input-RuO2-1.json`. These two inputs use different random seed for a demonstration purpose.

```
cd ${KINCAT_INSTALL_PATH}/bin
mpirun-openmpi-clang13 -np 2 bash -c './kincat.x --input="../example/RuO2/input-RuO2-$
↪{OMPI_COMM_WORLD_RANK}.json"'
```

1.3.2 Running on Weaver (GPU)

To run the code with a GPU, we first allocate an interactive compute node with following command. A single node is dedicated for the user. The Power9 CPU has 40 cores and can utilize 160 threads with symmetric multi-processing (SMP4) accelerated with 4 GPUs. Since Kokkos does not support the multi GPU use case, a user explicitly maps the MPI processes to different GPUs by adding `--kokkos-num-devices=4`.

```
[weaver11] bsub -Is -q rhel8 bash
***Forced exclusive execution
Job <42355> is submitted to queue <rhel7W>.
<<Waiting for dispatch ...>>
<<Starting on weaver1>>
[weaver1 ~]$ ./kincat.x --input='input.json'
```

1.3.3 Restart Simulation

When a simulation completes before it reaches its steady state, the simulation can be restarted using `dump-sites.json` and `dump-batch-sites.json` output. The dump files includes the last snapshot of site configurations and they are created at the end of simulation or when the code catches an exception. See the lattice section of the input file format explained next.

1.4 File Format

KinCat requires three input files specifying 1) solver, 2) dictionary, and 3) rates. The solver input can be specified by a user to control the KMC simulations while the dictionary input is auto-generated from a pre-processor, KinCatPy. The rates input is separated as we may want to use/test different set of rates that can be computed by multiple sources e.g., RMG.

1.4.1 Main Input

The following json input is the main input file extracted from the RuO2 example.

```
{
    "kincat": {
        "dictionary" : "${KINCAT_INSTALL_PATH}/example/RuO2/input-dictionary-RuO2.json",
        "rates" : "${KINCAT_INSTALL_PATH}/example/RuO2/input-rates-RuO2.json",
        "sites" : {
            "lattice" : [ 10, 10, 2 ],
            "type" : "random",
            "random seed" : 12345,
            "random fill ratio" : 0.0,
            "filename" : "dump-sites.json"
        },
        "dump" : {
             "dump filename" : "dump.json",
             "dump interval" : 5E-7
          },
        "statistics" : {
             "stats filename" : "stats.json",
             "types" : ["species_coverage","process_counts"],
             "stats interval" : 4E-7
         },
        "solver" : {
            "type" : "sublattice",
            "domain" : [ 5, 5 ],
            "random seed" : 13542,
            "random pool size" : 10000,
            "max number of kmc kernel launches" : 100,
            "max number of kmc steps per kernel launch" : 50,
            "time range" : [ 0, 1000, 1E-7 ]
        }
    }
}
```

- **`kincat` object includes the following items:**
  - `dictionary` specifies the file path to the dictionary json file.
  - `rates` specifies the file path to the rate json file.
  - `sites` object includes lattice information and its initialization method.
    - `lattice`: 2D lattice object. The array input implies `[ x-length, y-length, n-basis ]`. The lengths are specified in unit cells, and must be integers.
    - **`type`: lattice initialization method.**
      - `random` value fills the lattice randomly with percentage specified by `random fill ratio` keyword; otherwise, `random fill ratio` is ignored. A `random seed` is given just for the initialization routine.
      - `file` value loads site specification from a file given by the `filename`; otherwise, `filename` is ignored.
  - **`dump` object relates to an output file with the full simulation state.**
    - `dump filename` object specifies the name of the dump output file.

    * `dump interval` object indicates the minimum time between outputs of the full simulation state.

- **`statistics` object relates to an output file of various desired statistics of the KMC simulation.**

    * `filename` object specifies the name of the statistics output file.
    * `types` object is a list of the various statistics desired by the user. Currently implemented types include `"species_coverage"`, which outputs the fraction of the lattice sites filled by each species, and `"process_counts"`, which outputs the number of times each process has occurred up to that point in the KMC simulation. Further types may be added later, or added by the user.
    * `stats interval` object indicates the minimum time between outputs of the statsitics listed in `types`.

- **`solver` object include the following items to specify the KMC simulation.**

    * `type` specifies the algorithm selected i.e., `serial-rta`, `sublattice`, `batch-rta`.
    * `domain` specifies the lattice size `[x-length, y-length]` processed by a single process (or a group of threads) in a parallel algorithm (`sublattice`).
    * `serial-rta` solver type requires that the domain size matches to the lattice size; thus, this domain information is ignored. This solver is based on the BKL or 1st Order KMC algorithm. The `batch-rta` solver implements this algorithm for multiple samples.
    * **`sublattice` solver is based on the synchronous sublattice algorithm of Shim and Amar. It requires satisfi**

        · The lattice size should be an even integer multiple of sublattice domain size in both x and y dimensions.
        · The sublattice domain size should be bigger than the interaction range of the lattice model (included in the dictionary file) to avoid potential corruptions.

- `random seed` specifies the random seed used in the KMC simulations.
- KinCat uses a random number generator and pre-generates an array of random numbers and `random pool size` indicates the array size. The minimum random number array is `2*(# of subdomains)*(max number of KMC iterations per kernel launch)`.
- The simulation will complete either after it meets `max number of kmc kernel launches` or the simulation reaches `time end`.
- Each KMC kernel launch will complete either after the `max number of kmc steps per kernel launch` or the `dt` is reached. Note that if a kernel completes before reaching `dt`, the next kernel will only proceed until the prior `dt` is reached.
- `time range` specifies the `[ time begin, time end, dt (time-increment)]`.

Note that a single `solver.advance` function runs until it reaches `dt` time step or the maximum number of KMC iterations per kernel launch. When `dt` is set zero, the code runs for the specified number of KMC steps. If a user wants to ensure the `dt` time step is reached for each kernel launch, then the number of KMC steps needs to be sufficiently large. If the number of steps is not large enough to reach the desired `dt` timestep, then another kernel will launch with the same limiting endtime as the original `dt` timestep. For the `serial-rta`, running the code setting without the `dt` constraint does not cause any simulation errors. The `sublattice` algorithm requires synchronization among subdomains. A subset of subdomains are evolved simultaneously while the others are frozen. The algorithm rotates through subsets until all subdomains are synchronized at `dt`, rejecting the final steps that would extend past `dt`. The timestep should be sufficiently small so that the kinetics in the subdomain will be significantly different than at the beginning of the timestep or otherwise lead to significant errors. On the other hand, shorter timesteps lead to more

numerous rejected events and reduced efficiency. The user is encouraged to carefully consider what parameters will balance errors and efficiency for their system.

1.4.2 Ensemble Input

To exploit `kincat-batch.x`, it is required to append the following ensemble section to the above main input.

```
{
     "ensemble" : {
     "number of samples" : 4,
     "solver random number variations" :{
         "apply" : "enabled",
         },
     "sites random variations" : {
         "apply" : "disabled",
         "random fill ratio" : [ 0.3 ]
         },
     "rates variations" : {
         "apply" : "enabled",
         "type" : "file",
         "processes" :  ["CO_ads_cus", "CO_ads_br"],
         "process rates" : {
             "0" : [1.85e+06 , 2.15e+06 ],
             "1" : [1.90e+06 , 2.10e+06 ],
             "2" : [1.95e+06 , 2.05e+06 ],
             "3" : [2.00e+06 , 2.00e+06 ]
             },
         "process instances" : [0 , 1],
         "instance rates" : {
             "0" : [1.85e+06 , 2.15e+06 ],
             "1" : [1.90e+06 , 2.10e+06 ],
             "2" : [1.95e+06 , 2.05e+06 ],
             "3" : [2.00e+06 , 2.00e+06 ]
             },
         "override filename" : "../example/RuO2/input-rates-override-RuO2.json"
         }
     }
}
```

- `ensemble` object includes the following items: * `number of samples` specifies the number of samples.
  - When `solver random number variations` is enabled, each sample uses a different sequence of random numbers selecting a KMC event.
  - **`site random variations` varies the site configurations of samples.**
    - `random fill ratio` indicates the amount of random species fill on sites. Samples are mapped to the array in a round-robin fashion. For example, `[ 0.3 ]` means that all samples are randomly filled up to `0.3*n_sites`. If the array is `[ 0.3, 0.2 ]`, sample(0) is filled `0.3*n_sites`; sample(1) is filled `0.2*n_sites`; sample(2) is filled again `0.3*n_sites` and the rest of samples are filled in the same way.
    - When this option is `disabled`, samples use the same initial configuration, which is randomly configured.
  - **`process rates variations` allows for samples to use different process rates when it is enabled.**

* `type` can be either `inlined` or `file`.
  - **`inlined` looks for `processes` and `process instances`. Either or both may be included.**

    `processes` keyword indicates an array of the processes which rates are to be modified.

    If the `processes` keyword is present, then the `process rates` must also be present.

    The `process rates` object is a dictonary with the sample index and an array of rates. The rates correspond to the processes listed in the `processes` array.

    `process instances` keyword indicates an array of the process instances which rates are to be modified.

    If the `process instances` keyword is present, then the `instance rates` must also be present.

    The `instance rates` object is a dicitonary with the sample index and an array of rates. The rates correspond to the process instances listed in the `process instances` array.
  - `file` type value take `override filename` keyword to load the user specified rates for samples. The format of the file should be a json file with the `processes` and/or `process instances` arrays and their corresponding rates dictionaries.

Note that the `batch-rta` solver type is the only solver that supports the ensemble section for now. The batch input should be used with `kincat-batch.x` executable.

1.4.3 Dictionary Input

This file contains the information needed to set up the lattice shape and define possible KMC events. It is generated as an output of KinCatPy, and should not need to be modified by the user. This is an explanation of the information contained in the dictionary file. KinCatPy can be used to generate a dictionary file for two use-cases. KinCat uses so-called 1) reduced symmetry and 2) full symmetry configurations specifying the event mechanism. The full symmetry case specifies each possible process, while the reduced symmetry case uses the symmetry of the lattice to reduce amount of information needed to be stored in the dictionary. The following example dictionary file specifies the reduced symmetry case.

```
{
    "crystal": {
        "edge vectors": [[6.43, 0.0], [0.0, 3.12]],
        "basis vectors": [[0.0, 0.0], [0.5, 0.0]],
        "symmetry operations": {
            "shape": [4, 6],
            "data": [-1, 0, 0, -1, 0.0, 0.0, -1, 0, 0, 1, 0.0, 0.0, ...]
        }
    },
    "configurations": {
        "site coordinates": [[0.0, 0.0], [0.5, 0.0], [0.0, 1.0], [0.5, 1.0]],
        "variant orderings": [[2, 3, 0, 1], [0, 1, 2, 3]],
        "interaction range": [3, 3],
        "shape": [45, 4],
        "data": [1, 1, 1, 1, 1, 2, 1, ...]
    },
```

(continues on next page)

(continued from previous page)

```
    "process dictionary": {
        "processes": ["CO_ads_cus", "CO_ads_br", "O_ads_cus_cus", "O_ads_br_br", ...],
        "process constraints": [[[1, 0, 2]], [[0, 0, 2]], [[1, 0, 1], [3, 0, 1]], ...],
        "process symmetries": [[0], [0], [0], [0, 1], [0, 2], [0], [0], [0, 1], ...],
        "shape": [258, 3],
        "data": [0, 17, 7, 0, 39, 8, 0, 8, 9, 1, 40, 8, 1, 16, 9, 1, 2, 5, ...]
    }
}
```

- `crystal` object describes a unit cell structure and its symmetry operations.
  - `edge vectors` includes two vectors forms a unit cell (parallelogram) that is repeated to tile the lattice domain.
  - `basis vectors` represents the position of sites in the unit cell coordinates.
  - `symmetry operations` provides rotation matrices and translation vectors that produces equivalent symmetry configuration.
    - `shape` indicates `[ # of symmetries, array size (4 entries for 2x2 matrix, 2 entries for 2x1 vector) ]` and is used to interpret the `data` array.
- `configurations` object includes a list of possible configurations. A configuration is defined as a unique arrangement of simulation species within the interaction range of a system process.
  - `site coordinates` includes the position of sites in the reference configuration. The position is given in units of the crystal edge vectors.
  - `variant orderings` represents the possible enumerations (or re-ordering of sites) forming symmetry equivalent configurations when the configuration is mapped to the lattice sites.
  - `interaction range` defines the range around the central site that needs to be recalculated after each event due to possible changes to processes and rates in that region. It is given in unit cells. It also represents the minimum domain size for parallel solvers.
  - The list of configurations is stored as a 2D array `[ # of configurations, configuration size ]` where the entries of `data` are the species index.
- `process dictionary` object describes process mapping from one configuration to the other configuration.
  - `processes` is a list of process labels. For the reduced symmetry configurations, the processes are unique.
  - `process constraints` is a rank-3 array `[ # of processes, # of constraints, constraint size(3) ]`. For a corresponding process, a constraint `[ site index, initial species, final species]` represents the initial and final species of the specified site. The specified site in the initial and final configurations should match to those given in the constraint. Otherwise, we do not consider it as a valid process instance. The initial and final species may be the same, indicating that the site is not changed by the process, but that it is important for the process definition.
  - `process symmetries` indicates a pattern index that should be accounted when computing valid events. For example, an absorption event can be counted multiple times in the reduced symmetry configurations. To prevent this multiple counting, the process symmetry information include `[0]` pattern index so that the first symmetry pattern is only used when searching for possible events. Without the process symmetry information, the KMC process will find multiple events that are equivalent (for the adsorption example, it would find four events: one for each symmetry operation). To avoid the duplicated event search, we can also use the full symmetry dictionary as explained in the next section.
  - A process instance is specified as `[initial configuration, final configuration, process index]` and stored as rank-2 array `[# of events, event size(3)]`.

A full symmetry input is shown in the below. Note that the event dictionary grows exponentially with the number of sites and the number of species. Using a full symmetry dictionary might be prohibitive for a large reaction model.

```
{
     "crystal": {
     "edge vectors": [[6.43, 0.0], [0.0, 3.12]],
     "basis vectors": [[0.0, 0.0], [0.5, 0.0]],
     "symmetry operations": {
         "shape": [1, 6],
         "data": [1, 0, 0, 1, 0.0, 0.0]
     }
 },
 "configurations": {
     "site coordinates": [[-0.5, 0.0], [0.0, -1.0], [0.5, -1.0], [0.0, 0.0], [0.5, 0.0],␣
↪...],
     "variant orderings": [[0, 1, 2, 3, 4, 5, 6, 7]],
     "interaction range": [3, 3],
     "shape": [6561, 8],
     "data": [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, ...]
        },
"process dictionary": {
   "processes": ["CO_ads_cus", "CO_ads_br", "O_ads_cus_cus", "O_ads_cus_cus", ...],
   "process constraints": [[[4, 0, 2]], [[3, 0, 2]], [[4, 0, 1], [6, 0, 1]], ...],
   "process symmetries": [[0], [0], [0], [0], [0], [0], [0], [0], [0], [0], ...],
   "shape": [34992, 3],
   "data": [0, 540, 9, 0, 60, 10, 0, 1620, 11, 0, 180, 12, 0, 4536, 13, ...]
   }
}
```

Here, we only explain the major difference from the reduced symmetry case.

- The full symmetry configuration has an identity matrix for `symmetry operations`.
- The `variant` ordering is trivial i.e., identity map.
- As expected, the number of possible configurations and process intances is much bigger than the reduced configuration case e.g., 6561 vs 45 and 34992 vs 258.
- `events` can have duplicated processes names e.g., same absorption process with different initial configurations.
- `process constraints` and `process symmetries` requires inputs for the same number of `processes` array size.
- `process symmetries` is trivial and the full symmetry case does not have the duplicated event search issue.

1.4.4 Rate Input

The rate input is explained with the sample script in the below.

```
{
    "default rate": 0,
    "process specific rates" : {
    "CO_ads_cus" : 2.04E+06 ,
    "CO_ads_br" : 2.04E+06,
    "O_ads_cus_cus" : 3.81E+03,
    "O_ads_br_br" : 3.81E+03,
```

(continues on next page)

(continued from previous page)

```
    "O_ads_br_cus" : 3.81E+03,
    "CO_des_cus" : 1.82E+07,
    "CO_des_br" : 5.50E+04,
    ...
    },
    "event specific rates": {
    "5" : 1.85E+07,
    "11" : 2.00E+06,
    ...
    }
}
```

- `default rate` is used when the rate is not otherwise specified. A user can use a negative default value for error checking if the input file must specify all processes (or process instances).
- **`process specific rates` includes rates for processes.**
  - All process instances with the same process will be set to the same rate.
  - Events with processes not specified will be set to the default rate.
- **`process instance specific rates` includes rates for specific instances.**
  - Process instances not specified will be set to the process rate if one was specified, or the default rate otherwise.

1.4.5 Dump Output

When `dump` is enabled from the main input, the code dumps the output of the sites in the following format.

```
{
    "number of species": 3,
    "coordinates": {
         "shape": [ 200, 2 ],
         "data": [ 0, 0, 3.215, 0, 0, 3.12, 3.215, 3.12, 0, 6.24 ... ]
    },
    "sites": [
        {
            "sample": 0,
            "time": 0,
            "data": [ 0, 0, 0, 2, 2, 0, 2, 0, 0, 1, 0, 0 ... ]
        }
        {
            "sample": 1,
            "time": 0,
            "data": [ 0, 1, 0, 1, 2, 0, 2, 0, 0, 1, 0, 0 ... ]
        }
        {
            "sample": 0,
            "time": 0.1,
            "data": [ 1, 0, 0, 2, 2, 0, 2, 0, 0, 1, 0, 0 ... ]
        }
        {
            "sample": 1,
```

(continues on next page)

(continued from previous page)

```
            "time": 0.15,
            "data": [ 0, 2, 0, 2, 2, 0, 2, 2, 0, 1, 0, 0 ... ]
        }
    ]
}
```

A dump file can be used for post-processing and it includes 1) number of species, 2) coordinates, and 3) time series of sites information. An example of post-processing is illustrated in `${KINCAT_INSTALL_PATH}/bin/plot-dump.ipynb`. Additionally, `dump-sites.json` and `dump-batch-sites.json` files are created to record the last site configurations when the code completes, which can be used for restarting the simulation.

1.4.6 Stats Output

When `statistics` is enabled from the main input, the code dumps the selected statistics in the following format.

```
{
    "number of species": 3,
    "number of processes": 22,
    "processes": [ "CO_ads_cus", "CO_ads_br", "O_ads_cus_cus", "O_ads_br_br", ...],
    "readings" : [
        {
            "sample": 0,
            "time": 0,
            "species coverage": [ 1, 0, 0 ],
            "process counts": [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,␣
↪...]
        },
        {
            "sample": 0,
            "time": 4.02101e-07,
            "species coverage": [ 0.705, 0, 0.295 ],
            "process counts": [ 75, 54, 0, 0, 0, 67, 3, 0, 0, 0, 0, 1751, 0, 0, 0, 0, 0,␣
↪...]
        },
        {
            "sample": 0,
            "time": 8.00534e-07,
            "species coverage": [ 0.59, 0, 0.41 ],
            "process counts": [ 159, 74, 0, 0, 0, 146, 5, 0, 0, 0, 0, 4116, 0, 0, 0, 0, .
↪..]
        }
        ...
    ]
}
```

In the `readings` object, the `sample` and `time` will always be present. However, the rest of the objects will depend on the selected options.

—

1.5 KinCatPy

KinCatPy is an open source python script to write dictionary input files for use with KinCat. It recieves inputs describing the lattice, possible species and process mechanisms, and the range of possible interactions. It outputs a KinCat dictionary file, which contains information about all possible symmetrically unique process instances and all configurations needed to define those (given the input parameters). A given process mechanism, such as desorption from the surface, may have different rates due to lateral interactions with species on nearby sites. KinCatPy defines a unique process instance for each process that may occur between two unique initial and final configurations. Thus, rates for each instance can be adjusted to account for the lateral interactions, giving the user as much specificity in setting rates as desired. KinCatPy is adapted from KineCluE, another open source code intended to calculate transport coefficients of clusters. Note that while KineCluE was written to handle bulk crystals, KinCatPy is only intended for 2D lattice descriptions.

1.5.1 Main Input

Some of the input language and structure remains the same as KinCluE. While we outline the inputs here, the user may find KineCluE documentation to be helpful background. It is also helpful to understand how the Bravais lattice construction allows any lattice to be defined by a repeating unit cell and basis vectors within that cell. The following input is extracted from the RuO2 example.

```
& CRYSTAL test # creation of the crystal with 2 vectors of 2 components
6.43 0.0
0.0 3.12
& BASIS s 2
0 0 0
1 0.5 0
& UNIQUEPOS 2 # describes the sites in the crystal that the atoms and defect will occupy.
s 0 0
s 0.5 0
& RANGE 3.0  # the float is the interaction radius
& SPECIES 2
2 1 1 O 0.1
2 1 1 CO 0.1
& PROCMECH
#adsorption processes
%% 1 CO_ads_cus
s 0.5 0 0 > 2
%% 1 CO_ads_br
s 0 0 0 > 2
...
#desorption processes
%% 1 CO_des_cus
s 0.5 0 2 > 0
%% 1 CO_des_br
s 0 0 2 > 0
...
##diffusion processes
%% 2 CO_cus_cus
s 0.5 0 2 > 0
```

(continues on next page)

(continued from previous page)

```
s 0.5 1 0 > 2
%% 2 CO_br_br
s 0 0 2 > 0
s 0 1 0 > 2
...
##Recombination/desorption of CO2
%% 2 CO_cus_O_cus
s 0.5 1 1 > 0
s 0.5 0 2 > 0
%% 2 CO_br_O_br
s 0 1 1 > 0
s 0 0 2 > 0
...
```

It is useful to recognize two possible coordinate systems that KinCatPy will accept. The first is the standard x-y coordinates. The second is in terms of lattice unit cells that tile the plane. These are referred to as orthogonal and supercell coordinate systems respectively, and 'o' and 's' are used to specify which is used in the input file. Also recognize that & is used to indicate a new input command, and that # may be used to comment the remainder of the line.

- `& CRYSTAL` is used to define the unit cell of the lattice. The string after the command can be used to name the lattice if the user wishes. The next lines comprise two edge vectors that define the lattice unit cell. These values are always given in the orthogonal coordinate system, and these values are used to map between the orthogonal and supercell coordinate systems moving forward.
- `& BASIS` command is used to define the basis vectors of the lattice. If it is not present, then a single basis vector of (0,0,0) is assumed. The coordinate system of the basis vectors ('o' or 's') must be specified, and the number of basis vectors given. For each basis vector, a line of the form `site_type a b` is required. The `site_type` is an integer, and can be used to indicate if the site is fundamentally the same or different than another which is important for symmetry considerations. The `a b` terms are the coordinates of the basis vector in the specified coordinate system.
- `& UNIQUEPOS` command is similar to the `& BASIS` command, and must always be included. Theoretically, the BASIS vectors are used to define the symmetry of the system, while UNIQUEPOS define sites the species can occupy. Thus far, KinCatPy has only been tested where `BASIS` vectors and `UNIQUEPOS` vectors are the same. In `UNIQUEPOS`, the number of vectors is again required, but the coordinate system is defined for each vector, in the form `s(o) a b`.
- `& RANGE` command is used to define the interaction range, and the distance is in the units of the orthogonal coordinate system. Each configuration will include all sites within this range of a process, so increasing it may dramatically increase the complexity of the KMC model generated.
- `& SPECIES` command is used to give information about the possible species in the system. Currently, each species is considered as a single point, so it is impossible to include information about orientation or multi-site binding in the model. The number of species needs to be given, and then each species is specified by a line of the form `n pos_bools... name size`. The number of each species is intended to be provided by `n` (as used in KineCluE). However, the user just needs to ensure that this integer is larger than the number of that species specified in any single `JUMPMECH`. Next, `pos_bools` refers to a set of 0/1 flags indicating if that species can occupy the `UNIQUEPOS` given. There should be as many flags are there are `UNIQUEPOS` specified. The `name` is a unique text string. The last number is `size` indicating the implied radius of the species (orthogonal units). Species with sufficient size may 'block' neighboring sites and prevent other species from occupying them. The size may be set arbitrarily small if this functionality is not desired.

`& PROCMECH` command is used to define possible process mechanisms through sets of constraints. After the `& PROCMECH` command is given, a new process mechanism definition is denoted by a line of form `%% n_constraints name`. Each process name should be unique. `n_constraints` is the number of constraints that define the process. Each

contstraint on the process is given by a line of the form `s(o) a b ini_species_type > fin_species_type`. The letter `s(o)` is used to denote which coordinate system will be used for `a b`, which are the site coordinates. `ini_species_type` and `fin_species_type` are the integer indecies of the initial and final species respecively. The species inputed using the `& SPECIES` command are assigned indicies in their list order, beginning with one. The species index 0 is reserved to indicate a vacant site. Note that each constraint definition consists of a site and the change of species on that site. Thus, defining a diffusion jump involves two constraints, one for the species dissapearing from the initial site and another for the species appearing in the final site. So called 'bystander' or 'spectator' species that do not change may also be included in the constraints. For example, a reaction between two 'A' species may be catalyzed by the presence of a 'B' species. The B species would need to be included in the constraints with an approriate coordinate relative to the A species, but the initial and final `species_type` of that constraint would be the same.Note that a only single symmetry of a process needs to be included. Symmetric processes are found by accounting for the symmetries of the lattice.

`& DIRECTORY` command can be used to specify an output directory where the output files will be generated. If this command is not included, an output directory named 'CALC/' is auto-generated in the working directory.

`& FULLSYM` command can be used to create a dicitonary where the configurations and process instance dictionaries are not reduced by symmetry operations.

Note again that the complexity of the model given will greatly depend on 1) the number of species included, 2) the range of interactions accounted for (set by `RANGE`), and 3) the choice of process definitions (for the reduced symmetry case). The computational demands of KinCatPy are largely determined by the number of unique configurations that need to be specified. Since it will identify all possible permutations of the species arranged on a set of lattice sites, the number of species and number of sites in the configuration definition will both have an exponential impact on the number of configurations overall. All lattice sites within the interaction range of a site that is changed by a process (the species on that site changes, not just that it is included in a constraint). Increasing the interaction range will increase the number of sites needed for the configuration definition. However, for the reduced symmetry case, careless process definition may also increase the number of sites defined. The process definitions should use the symmetries that overlap as much as possible. In this example defining CO adsorption and desorption processes using 'cus' sites `0 0.5` and `0 -0.5` is valid, since they are symmetrically equivalent to each other. However this would lead to an expansion of the configuration definition and increase the complexity of the model defined by kincatpy. Since all symmetries of each process are included in the full symmetry case, it does not matter. We reccomend using the reduced symmetry case for most simulations, but the full-symmetry case may be easier for the user to interpret.

We also note that for large model dictionaries, a multi-thread script (kincat_event_calc.py) is called. However, this only significantly improves efficiency with large numbers of configurations, and so is not called if the model has fewer than 100,000 configurations. Even models with over a 100,000 configurations and over 1,000,000 process instances only takes a few minutes to generate.

KinCatPy is run in the command line window as `python ./kincat_release1.py input.txt`.

1.6 KinCat Overview

A basic KMC algorithm in the below. The algorithm is often called residential time algorithm (RTA) and inherently sequential. In KinCat, we consider three solver implementations: 1) serial reference implementation, 2) sublattice parallel implementation, and 3) batch parallel implementation.

```
SolverSerialRTA::advance(in: t, in: t_step, in: n_max_iterations,
                         in: lattice, in: dictionary, in: rates, in: rates_scan) {
  /// update event rates on lattice
  updateEventRatesLattice(in: lattice, in: dictionary, out: rates);

  for (iter=0; iter<n_max_iteration && t<t_end; ++iter) {
```

(continues on next page)

(continued from previous page)

```
    /// perform prefix sum
    scanRatesLattice(in: rates, out: rates_scan);

    getRandomNumber(out: pi, out: zeta);
    sum_rates = rates_scan(in: rates.extent(0));
    dt = log(zeta)/sum_rates;
    rate_to_search = sum_rates * pi;

    /// cid is the cell index corresponding to a specific rate (sum_rates*pi) in the␣
↪scanned array
    searchCellIndex(in: rate_to_search, in: rates_scan, out: cid);

    /// select an event in the selected cell
    findEvent(in: cid, in: dictionary, out: event);

    /// update the cell configuration
    updateLattice(in: cid, in: event, out: lattice);

    /// local updates of rates on the neighborhood of the selected cell
    updateEventRates(in: lattice, in: dictionary, out: rates);

    /// advance time
    t += dt;
  }
}
```

The sublattice algorithm decomposes the lattice into multiple subdomains. In each subdomain, a sequential RTA algorithm runs and time is updated asynchronously. To avoid the potential conflicts on the boundary regions, the algorithm updates even and odd numbered domains separately.

```
SolverSublattice::advance(in: t, in: t_step, in: n_max_iterations,
                          in: lattice, in: dictionary, in: rates, in: rates_scan) {

/// update event rates on lattice
updateEventRatesLattice(in: lattice, in: dictionary, out: rates);

/// even odd selector
quad = { {0,0}, {1,0), {0,1}, {1,1} };

for (iter=0; iter<n_max_iteration && t<t_end; ++iter) {
  for (q=0; q<4; ++q) {
    parallel_for(in: team_policy(n_domains), in: LAMBDA(member) {
      /// compute domain indices corresponding to the member's league rank
      getDomainIndex(in: member, in: lattice, out: d0, out: d1);

      /// select even or odd domain in 2D
      if (d0%2 == quad[q][0] && d1%2 == quad[q][1] && t(d0,d1) < t_step) {
        /// get domain specific containers
        auto rates = rantes_lattice(d0, d1);
        auto rates_scan = rantes_scan_lattice(d0, d1);

        /// nested parallel scan over rates
```

(continues on next page)

(continued from previous page)

```
        parallelScanRates(in: member, in: rates, out: rates_scan);

        /// update event
        single([=]() {
          getRandomNumber(out: pi, out: zeta);
          sum_rates = rates_scan(in: rates.extent(0));
          dt = log(zeta)/sum_rates;
          rate_to_search = sum_rates * pi;

          /// cid is the cell index corresponding to a specific rate (sum_rates*pi) in␣
↪the scanned array
          searchCellIndex(in: rate_to_search, in: rates_scan, out: cid);

          /// select an event in the selected cell
          findEvent(in: cid, out: event);

          /// update the cell configuration
          updateLattice(in: cid, in: event, out: lattice);

          /// update the domain clock
          t(d0,d1) += dt;
        });

        /// local updates of rates on the neighborhood of the selected cell
        parallelUpdateEventRates(in: member, in: lattice, in: dictionary, in:␣
↪interaction_range, out: rates);
      }
    });
  }
}
```

The batch RTA solver runs the serial RTA in parallel solving multiple KMC problems considering the use case that requires to solve multiple samples which arises from UQ analysis. The user interface and container structure needs to be changed to handle the use case correctly. For the batch mode, we implement a searate executable.

1.7 API

Here, we manually and briefly document important things only.

1.7.1 Kokkos Things

Note that some functions are decorated with `KOKKOS_INLINE_FUNCTION`. This means that the function can be called on device. All captured object in the device function is `const` object. We cannot alter the members nor call non-const functions.

`KOKKOS_LAMBDA` cannot capture the implict or explicit "this" pointer. If a member variable needs to be used in the lambda function, assign the member variable to a local variable. Most of Kokkos view does soft copy and no actual cost for the assignment.

1.7.2 class Lattice

The lattice object provides means for accessing specific location of sites and providing symmetry patterns. These patterns are used to construct an configuration array which is used for comparing against dictionary to identify the configuration and possible events.

- `value_type_2d_view<site_type,device_type> _sites`. - The site_type is determined at compile time i.e., `char,short int, int` to save bytes

  for frequent memory transfer between host and device.

  - `_sites(sid,cid)` stores the current status of species occupied on the site location (cid) for the sample (sid).
  - `copySites`. When 2d site view is used, it overwrites for all samples. For 1d site view input with sample id, it overwrites sites for the specific sample.
  - `getCellIndex(k0, k1, out: cid)`. For a given lattice index k0 and k1, it computes the corresponding cell index. The cell index is different from site index and site index accounts for multiple basis points. Note that we use k0 and k1 indices indicating the location of the lattice and we use l0 and l1 indices for local domain (sublattice algorithm).
  - `getDomainCellIndex(k0, k1, out: d0, out: d1, out: lid)`. For a given k0 and k1, it computes domain index d0 and d1 and site location (lid).
  - `getLatticeCellIndex(d0, d1, l0, l1, out: k0, out: k1)`. The function computes lattice index from domain indicies.
  - `getSiteIndex(k0, k1, ib, out: site_index)`. The function returns the site_index in 1d array. This function needs basis index (ib).
  - `adjustPeriodicBoundary(inout: k0, input: k1)`. The lattice assumes periodic conditions and input k0 and k1 are adjusted accordingly.
  - `getCoordinates`. Used for post processing.

1.7.3 class ProcessDictionary

- `value_type_2d_view<ordinal_type, device_type> _variant_orderings`. This includes correct enumeration orders for configuration.
- `value_type_2d_view<site_type, device_type> _configurations`. `_configurations(conf_id, conf_index) = species on this configuration and index`. The configuration array is stored after sorted for binary search.
- `value_type_2d_view<ordinal_type, device_type> _processints`. `_processints(initial_conf_id, final_conf_id, process_id)`. The process instance is stored after it is sorted for binary search. The instances change the current configuration to the final configuration and its mechanism is explained by the process.

- `value_type_2d_view<ordinal_type, device_type> _constraints`. Certain variant ordering of the configuration is legit and this constraint view filter out non-feasible configuration.
- `value_type_2d_view<real_type, device_type> _rates`. `_rates(sample_id, event_id)` indicates a specific fate for the event of the sample. Each sample can use the same or different rates.
- `searchConfiguration(input_key, out: index_configuration, out: index_variant)`. The function performs binary search and report corresponding configuration index and its variant index.
- `searchFirstEvent(index_configuration, out: index_first_event)`. The function reports the index of the first process instance for the given configuration. When system rate is scanned, each cell has a symmetry unique configuration and this function find a list of possible process instances.

1.7.4 class SolverBase

A base class for different solver variants i.e., serial-rta, batch-rta, sublattice.

- `initialize` setup the solver internal objects. Since the lattice and dictionary object uses 2d views that can be used for multiple samples, single problem (non-batch mode) can use the same data structure by setting number of samples is 1.
- `advance(2d_view/1dview, ....` The function provides two interface. 2d the time view is used for sublattice algorithm and the serial version just use `t_in(0,0)` only. The 1d time array is for the batch mode `t_in(sample_id)`.
- `createSolver(solver_type)`. The function creates solver object based on the input string (solver_type).

1.7.5 class Dump

- `initialize` function dump the header (lattice coordinates).
- `finalize` function close the dump file and create the last snapshot of sites which can be used for restarting of the simulations.
- `snapshot` function dump the current site information with sample and time stamp.

1.7.6 class Stats

-`initialize` function outputs the header of a statistics json file, relevant to the statistics desired. -`finalize` function closes the statistics file and creates the last snapshot of the statistics. -`snapshot` function outputs the chosen statistics to the statistics file with the sample and time stamp.

1.7.7 class ProcessCounter

- `initialize` function create counter views for both device and host. The internal objects will be properly deleted as they are reference counted objects.
- `reset` function zeros all counters.
- `syncToHost` perform deep copy to host so that a host function can access updated counter information.
- `update(sample index, event index)` function will increase the counter for the sample.