College on Multiscale Computational Modeling of Materials for Energy Applications

Trieste, 4-15 July 2016

Training Materials

Table of contents

1 Preparation

Install and setup software packages

The packages ‘mongodb’ and ‘nwchem’ are already installed and configured on the machines.

Install Python 2.7 (anaconda):

```
mkdir -p /scratch/$USER

cd /scratch/$USER

tar xzf /scratch/smr2874/kondov-tutorial-2.tgz Anaconda2-4.0.0-Linux-x86_64.sh

bash Anaconda2-4.0.0-Linux-x86_64.sh

(install directory: /scratch/<username>/anaconda2)

export PATH="/scratch/$USER/anaconda2/bin:$PATH"

ln -s /scratch/$USER ~/fw-tutorial

cd ~/fw-tutorial

tar xzf /home/nfs3/smr2874/ictp-tutorial-fireworks.tgz
```

Install python packages: ase, fireworks, python-igraph, pjson:

```
pip install --upgrade pip

pip install ase

pip install fireworks

pip install python-igraph

pip install pjson

pip install pyaml
```

Start the database

You should skip this step if mongod is running. Otherwise there might be an address conflict. Open a new terminal and enter the following commands:

```
cd ~/mongodb

export PATH=$PWD/bin:$PATH

export MONGO_DBPATH=$PWD/data/db

mongod --dbpath=$MONGO_DBPATH
```

Tutorial environment setup

Open a new terminal and enter the following commands:

```
cd ~/fw-tutorial/ictp-tutorial-fireworks

export PATH="/scratch/$USER/anaconda2/bin:$PATH"

export PYTHONPATH=$HOME/fw-tutorial/ictp-tutorial-fireworks/lib:$PYTHONPATH
```

2 General steps

LaunchPad cleanup

When used productively the LaunchPad contains many workflows in different states. To distinguish between different workflows, the query commands have to specify the firework ID or workflow ID of the relevant workflow on the LaunchPad. At the beginning of each exercise in this tutorial we will clean up the launch pad from previous fireworks so that we do not have to use IDs to execute and query. To clean up all workflows on the LaunchPad this command can be used:

```
lpad reset
```

Create a workflow

```
python singletask.py

pjson < workflow.json

more workflow.yaml
```

Add a workflow to LaunchPad

To add a workflow to the LaunchPad:

```
lpad add workflow.yaml
```

Alternatively in JSON format:

```
lpad add workflow.json
```

Query FireWorks on the LaunchPad

To query fireworks available on the LaunchPad:

```
lpad get_fws

lpad get_fws -d all
```

To query workflows available on the LaunchPad:

```
lpad get_wflows -d all
```

Launch FireWorks

In the FireWorks framework workflows are executed by "launching rockets". Either one FireWork is processed at a time using the rlaunch command, or multiple FireWorks can be processed with the mlaunch command. The rlaunch command has two modes of operation: singleshot and rapidfire. The command

```
rlaunch singleshot
```

will execute one FireWork on the LaunchPad which has "READY" state and exit. The command

```
rlaunch rapidfire
```

will run all FireWorks in "READY" state in a sequence. Please note that if a FireWork changes its state to "READY" after all dependencies are completed. This means that a linear workflow with downstream dependencies will be executed completely in one call of this command. To suppress verbose information on the screen the "-s" flag can be added:

```
rlaunch -s rapidfire
```

3 Exercise 1: SingleTask

The purpose of this custom-made FireTask is to call a Python function 'function' passing the data objects listed in 'args' as positional argument and to store the returned objects under the name or names listed in 'output' in the specs of all children FireWorks using FWAction. All names in 'args' must be available in the spec of the current FireWork. The names of the Python module and function must be found in $PYTHONPATH. Changes of objects passed will be stored in the current but not in next FireWork. The SingleTask argument 'function' is mandatory, the 'args' and 'output' arguments are optional.

The example demonstrates the use of SingleTask and ScriptTask and how data can be passed from one FireWork to another within a very simple workflow. To create the workflow change to the example directory:

```
cd exercises/exercise_1
```

```
cp ../demos/singletask_demo.py .
```

and run the Python script

```
python singletask_demo.py
```

Then inspect the created workflow in the file `singletask_demo.yaml`. The Python script automatically resets the LaunchPad and adds the workflow to it. You can query the workflow with

```
lpad get_fws -d all
```

Then execute the first FireWork with

```
rlaunch -s singleshot
```

You will see the data element printed on the screen. If you query the FireWork with ID=1, i.e. the second FireWork in the workflow

```
lpad get_fws -i 1 -d all
```

you will find that its status has changed to "READY" and that 'outputs' contains the data passed from the previous FireWork. Now execute the second FireWork:

```
rlaunch -s singleshot
```

Please note how the two FireTasks of the second FireWork were executed one after another "in one shot".

4 Exercise 2: ForeachTask

The purpose of ForeachTask is to dynamically branch the workflow between this FireTask and its children inserting dynamically a parallel section of children FireWorks using FWAction detour. The number of the parallel FireWorks is determined by the length of the list specified by the 'split' ForeachTask argument. Each child FireWork contains a SingleTask which processes one element from this list. The return value is passed to the spec of the FireWork after the detour using a push method, i.e. the values of all parallel SingleTasks are collected in a list named by the 'output' argument. However, the ordering of items in the resulting 'output' list can be different from that in the original 'split' list.

Change to the directory `exercise_2` and create the workflow using the Python script:

```
cp ../demos/foreachtask_demo.py .
```

```
python foreachtask_demo.py
```

The python script dumps the workflow definition to JSON and YAML files and adds the workflow to the launch pad. Now open the file `foreachtask_demo.yaml` with a text editor to see the workflow definition and make a query on the LaunchPad:

```
lpad get_fws
```

The first FireWork is in "READY" state and the second one is in "WAITING" state. Now execute the first FireWork with the command:

```
rlaunch -s singleshot
```

```
lpad get_fws
```

Now we see that the first FireWork is in "COMPLETED" state and that the workflow has been extended with three additional FireWorks in "READY" state: one for each element in 'array'. Now start all these Fireworks:

```
mlaunch -s --nlaunches 1 3
```

```
lpad get_fws -i 1 -d all
```

The query shows that the last FireWork has the results from the three parent FireWorks which were inserted by the first FireWork. These results are in the data element 'new array'. Now execute this last FireWork which will print the reassembled list on the screen:

```
rlaunch -s singleshot
```

Note the possibly different ordering of the elements in 'new array' compared to 'array'.

5 Exercise 3: Charge transfer in dimers

The goal is to create a *sequential workflow* to calculate the charge transfer rates in dimers with given pre-optimized structures. For this purpose use a template in the directory `problems`:

```
cd exercise_3

cp ../problems/ct_workflow_seq.py dimer_workflow_seq.py

gedit dimer_workflow_seq.py
```

If you do not feel comfortable with Python you can chose to edit a JSON or YAML description of the workflow. For this copy and edit the solution templates:

```
cp ../problems/ct_workflow_seq.json dimer_workflow_seq.json

gedit dimer_workflow_seq.json
```

Remove the unnecessary FireWorks and fill in the names of the relevant python functions. The relevant python functions are collected in the file `../../lib/fw_task_functions_seq.py`. The simulation parameters and the dimer structures are available in the directory `../inputs`. A solution workflow in Python, JSON and YAML formats is available in directory `../solutions`. Before adding the workflow to the LaunchPad make sure to change the key 'monomer file' to 'morphology file' add the key 'morphology file' to parameters.json. Also make sure to set a correct "src" path for the upload of the dimer structure. You should set it to the path to your exercise_3 directory. You can find this path by running the "pwd" command. After this reset the LaunchPad and add the workflow. Run simulations for different dimers: formic acid, benzene, uracil, alq3, for holes and electrons changing the input parameters in the parameter file.

Make the workflow run faster

Adding the key "reorganization energy" will make the workflow skip the computing of the reorganization energy. This will reduce the total computing time, especially for benzene, uracil and alq3. You can use the following values for the reorganization energy:

Monomer	Reorganization energy, eV
Benzene	0.510
Uracil	1.29
Alq3	0.157

In addition, you can run the DFT code in parallel and speedup every DFT calculation. To run NWChem on 4 processor cores, you should add the "command" keyword:

`"command": "mpirun -np 4 nwchem PREFIX.nw > PREFIX.out"` (in JSON)

`command: mpirun -np 4 nwchem PREFIX.nw > PREFIX.out` (in YAML)

6 Exercise 4: Charge transport in disordered structures

The goal of this exercise is to create a workflow to simulate charge mobility in a disordered material consisting of small organic monomers.

Task 1: Sequential workflow

As first task create a sequential workflow by editing a template:

```
cd exercise_4

cp ../problems/ct_workflow_seq.py .

gedit ct_workflow_seq.py
```

If you do not feel comfortable with Python you can chose to edit a JSON or YAML description of the workflow. For this copy and edit the solution templates:

```
cp ../problems/ct_workflow_seq.json .

gedit ct_workflow_seq.json
```

Again fill in the names of the relevant functions (see previous exercise).

An alternative, more involved method: An alternative way to solution is to reuse the solution of Exercise 3 (for JSON and YAML simply substitute .py with .json and .yaml respectively):

```
cp ../solutions/dimer_workflow_seq.py ct_workflow_seq.py

gedit ct_workflow_seq.py
```

Then add the two missing FireWorks (compare the workflow graphs to find which).

After the sequential workflow is ready, add it to the LaunchPad and perform a simulation for formic acid.

Task 2: Parallel workflow

The dataflow structure of the sequential workflow is very simple and the workflow seems to be sufficient for a dimer simulation. However, for disordered systems with many molecules one might take advantage of the fact that the numerous DFT calculations of the site energies and the electronic couplings are independent and can be performed concurrently. Also the evaluation of the reorganization energy can be done at any time of the simulation after the preparation and before the analysis step.

The goal of this task is to create a *parallel workflow* that exploits these two types of concurrency.

```
cp ../problems/ct_workflow_par.py .

gedit ct_workflow_par.py
```

For JSON/YAML you have to only substitute .py with .json or .yaml respectively.

The task can be solved by completing the ‘args’, ‘split’ and ‘output’ tags of the SingleTasks and ForeachTasks. For this you have to study the dataflow graph of the parallel charge transport workflow. An alternative solution is to look at the interfaces of the relevant python functions under

`../../lib/fw_task_functions_par.py`. The solution of this task can be found in the directory `../solutions`. After you have composed the workflow perform simulations for benzene and alq3.

Make the workflow run faster

The command "rlaunch" will run the FireWorks one by one. If you have more resources on the machine you can process multiple FireWorks at a time. The "mlaunch" command:

```
mlaunch --sleep 5 --nlaunches infinite 4
```

will start four workers in parallel which will process all FireWorks in "READY" state on the LaunchPad. After a FireWork is finished the worker waits 5 seconds until checking out the next FireWork.

7 Exercise 5: Extending an existing workflow

A workflow in any state (for example: ready, running or completed) can be extended with a single FireWork or with another workflow. This is especially useful if some data (intermediate, final) must be further processed (for example another simulation with the same model but other parameters, or in another model), analyzed, visualized and /or downloaded. Let us repeat the last analysis step of the charge transport workflow and download the generated files (visualization and topological analysis of the morphology, the morphology itself in xyz format, the radial distribution function etc.).

Identify the name of the FireWork after which the new (short) workflow has to be appended and put it in the FireWork ID query in the template solution:

```
cd exercise_5

cp ../problems/ct_workflow_++.py .

gedit ct_workflow_++.py

python ct_workflow_++.py
```

Running the python script will append the workflow to the target workflow on the LaunchPad. After this, check the state and execute the workflow. Find the archived files with the results in the current working directory. Unpack then using the program `unzip` and view the results using a web browser and a graphics viewer (such as `okular` or `eog`).

8 Figures

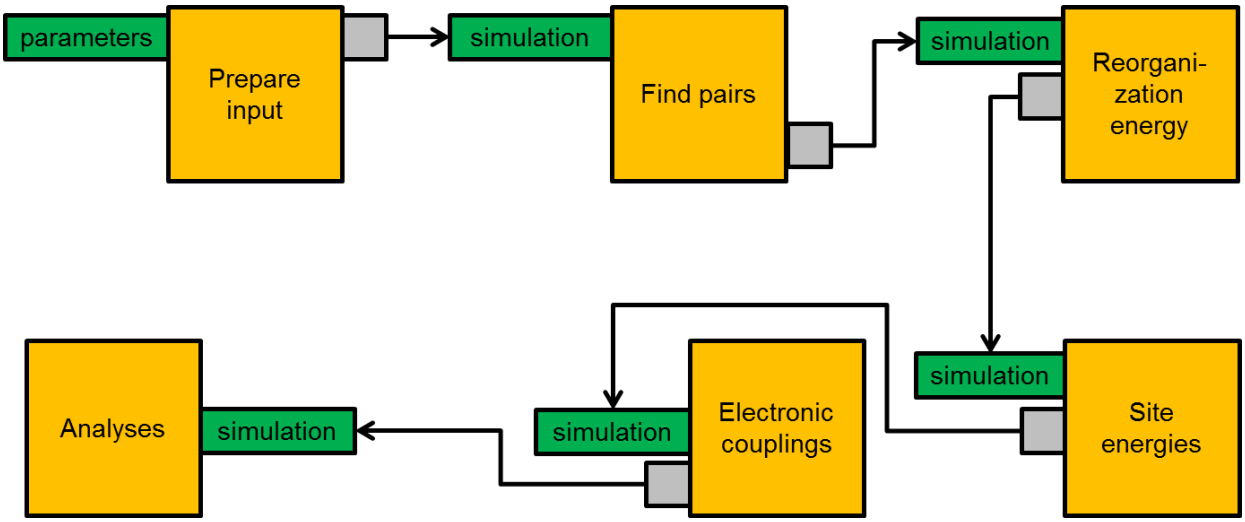

Figure 1: Sequential workflow for dimer simulation

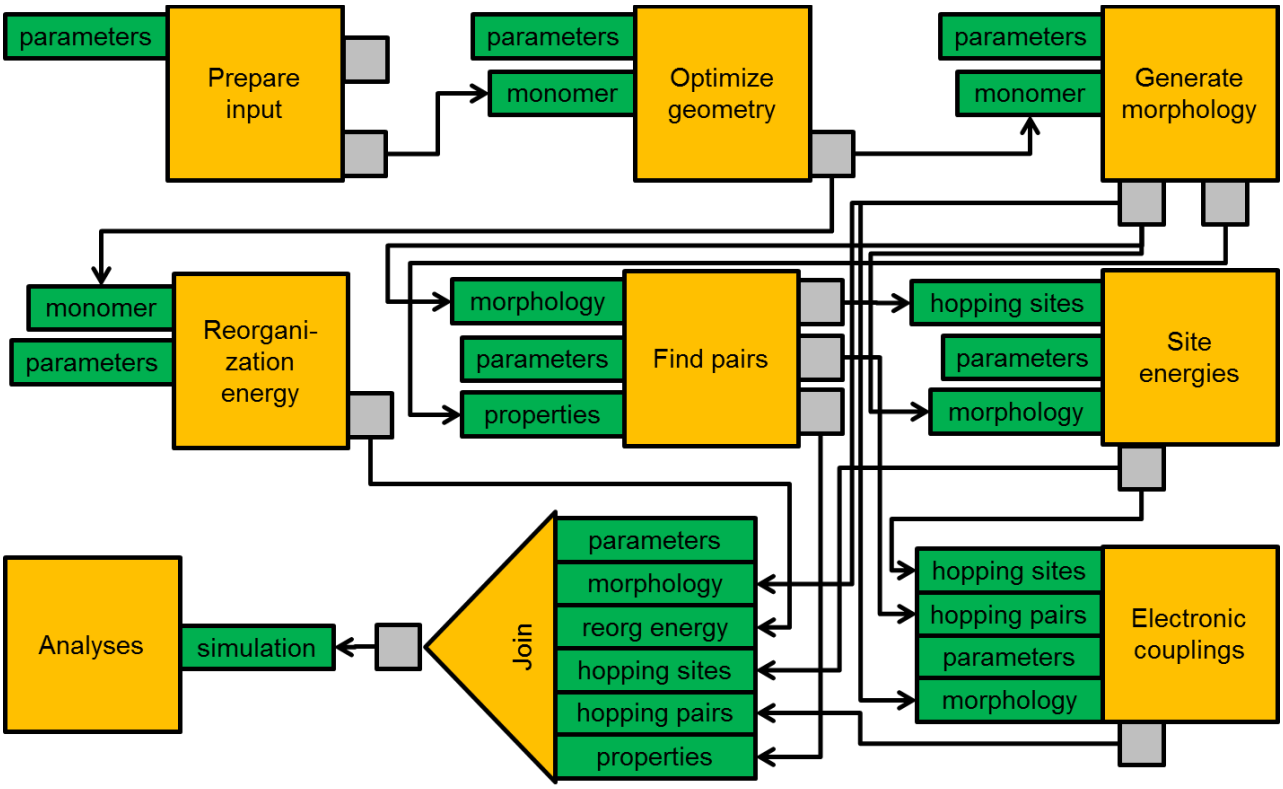

Figure 2: Parallel workflow for organic electronics simulation. The links for the 'parameters' data entity are not shown for the sake of clarity.