PyMDSetup

February 28, 2019

1 PyMDSetup Demo

2 PyMDSetup Description

PyMDSetup is a python package to setup systems to run molecular dynamics simulations. The current version uses the following applications:

- GROMACS: Open source and widely used molecular dynamics simulation package.
- SCWRL4: Application to determine the protein side chain conformations.
- GNUPLOT: Gnuplot is a portable command-line driven graphing utility for Linux.
- PyCOMPSs: Python library for parallel computing.

3 PyCOMPSs Description

PyCOMPSs is a task-based programming model that aims to ease the development of parallel applications, targeting distributed computing platforms. It relies on its runtime to exploit the inherent parallelism of the application at execution time by detecting the task calls and the data dependencies between them. The Runtime natively supports Java applications but also provides bindings for Python and C/C++.

4 Code

4.1 Start PyCOMPSs

In [ ]:
```
# Import Interactive PyCOMPSs module
import pycompss.interactive as ipycompss
```

In [ ]:
```
# Start PyCOMPSs
ipycompss.start(app_name='pymdsetup',                                  # Application config
                project_xml='../scripts/xmls/project.xml',
                resources_xml='../scripts/xmls/resources.xml',
                summary=True,                                          # Tools configuratio
                graph=True,
                monitor=2000,
                trace=True,
                debug=True                                             # Enable debug mode
)
```

4.2 Tasks Definition: Regular tasks

```
In [ ]: from pycompss.api.task import task
        from pycompss.api.constraint import constraint

        from pycompss.api.parameter import FILE_IN, FILE_OUT
```

```
In [ ]: @task(input_solute_gro_path=FILE_IN, output_gro_path=FILE_OUT, input_top_zip_path=FILE_
        def solvate_pc(input_solute_gro_path, output_gro_path, input_top_zip_path, output_top_z
            # Internally zips the outputs
            from wrappers.gromacs_wrapper import solvate
            solvate.Solvate(input_solute_gro_path=input_solute_gro_path,
                            output_gro_path=output_gro_path,
                            input_top_zip_path=input_top_zip_path,
                            output_top_zip_path=output_top_zip_path,
                            properties=properties,
                            **kwargs
                            ).launch()
```

```
In [ ]: @constraint(MemorySize="2.0")
        @task(output_plotscript_path=FILE_OUT, varargsType=FILE_IN)
        def gnuplot_pc_plotscript(output_plotscript_path, output_png_path, properties, mutatio
            from wrappers.gnuplot_wrapper import gnuplot
            try:
                gnuplot.Gnuplot.generate_plot_script(output_plotscript_path,
                                                     output_png_path,
                                                     properties,
                                                     mutations_list,
                                                     args)
            except Exception:
                import traceback
                from utils.tasks_jupyter import write_failed_output
                traceback.print_exc()
                write_failed_output(output_plotscript_path)
```

4.3 Tasks Definition: Binary tasks

```
In [ ]: from pycompss.api.binary import binary
```

```
In [ ]: @binary(binary="${GMX_BIN}/gmx")
        @task(input_gro_path=FILE_IN, output_gro_path=FILE_OUT)
        def editconf_pc(editconf="editconf",
                        f="-f", input_gro_path="",
                        o="-o", output_gro_path="",
                        d="-d", distance_to_molecule="",
                        bt="-bt", box_type="cubic",
                        c="-c"):
            pass
```

```
        # The task call is:
        # editconf_pc(input_gro_path="", output_gro_path="", distance_to_molecule="")

        # The task execution will automatically call:
        # gmx editconf -f igp -o ogp -d dtm -bt cubic -c
```

In [ ]:
```
from pycompss.api.parameter import Type, Prefix

@binary(binary="gnuplot")
@task(plotscript_path=FILE_IN, output_png_path={Prefix: "#"})
def gnuplot_pc_image(plotscript_path="", output_png_path=""):
    pass

# The task call is:
# gnuplot_pc_image(plotscript_path="", output_png_path="")

# The task execution will automatically call:
# gnuplot plotscript_path

# The prefix can also be used for parameters of the form: --x=value
```

4.4 Tasks Definition: MPI tasks

In [ ]:
```
from pycompss.api.mpi import mpi
```

In [ ]:
```
computing_units = "2"
computing_nodes = 1
```

In [ ]:
```
@constraint(ComputingUnits=computing_units)
@mpi(runner="mpirun", binary="${GMX_BIN}/gmx_mpi", computingNodes=computing_nodes)
@task(input_tpr_path=FILE_IN, output_gro_path=FILE_OUT, output_trr_path=FILE_OUT, outpu
      output_log_path=FILE_OUT)
def mdrun_pc(mdrun="mdrun",
             s="-s", input_tpr_path="",
             c="-c", output_gro_path="",
             o="-o", output_trr_path="",
             x="-x", output_xtc_path="",
             e="-e", output_edr_path="",
             g="-g", output_log_path="",
             nt="-nt", nt_value=0):
    pass

# The task call is:
# mdrun_pc(input_tpr_path="", output_gro_path="", ...)

# The task execution will automatically call:
# mpirun -np 2 -hostfile X gmx_mpi mdrun -s itp -c ogp ...
```

In [ ]:
```
@constraint(ComputingUnits=computing_units)
@mpi(runner="mpirun", binary="${GMX_BIN}/gmx_mpi", computingNodes=computing_nodes)
```

```
        @task(input_tpr_path=FILE_IN, output_gro_path=FILE_OUT, output_trr_path=FILE_OUT, outpu
              output_cpt_path=FILE_OUT, output_log_path=FILE_OUT)
        def mdrun_pc_cpt(mdrun="mdrun",
                         s="-s", input_tpr_path="",
                         c="-c", output_gro_path="",
                         o="-o", output_trr_path="",
                         x="-x", output_xtc_path="",
                         e="-e", output_edr_path="",
                         cpo="-cpo", output_cpt_path="",
                         g="-g", output_log_path="",
                         nt="-nt", nt_value=0):
            pass
```

4.5 Import the rest of the tasks

```
In [ ]: from utils.tasks_jupyter import scwrl_pc
        from utils.tasks_jupyter import pdb2gmx_pc
        from utils.tasks_jupyter import grompp_pc
        from utils.tasks_jupyter import grompp_pc_cpt
        from utils.tasks_jupyter import genion_pc
        from utils.tasks_jupyter import rms_pc
```

4.6 Some useful methods

```
In [ ]: from utils.workflow import process_arguments
        from utils.workflow import setup
        from utils.workflow import log_mutation_execution
        from utils.workflow import setup_mutation_dirs
        from utils.workflow import plot_results
        from utils.workflow import show_execution_results
```

4.7 Main code

```
In [ ]: # Main flow imports
        import sys
        import time

        # PyCOMPSs imports
        from pycompss.api.api import compss_barrier


        ################################################################
        # MAIN WORKFLOW FUNCTION
        ################################################################

        def pymdsetup_main(exec_cfg, entry_cfg):
            # Start timer
            start_time = time.time()
```

```python
# Retrieve and process arguments
app_conf, app_logger = process_arguments([exec_cfg, entry_cfg])

# Log
app_logger.info("")
app_logger.info("_______GROMACS FULL WORKFLOW_______")
app_logger.info("")

# Setup application
structure, mutations, mutations_limit = setup(app_conf, app_logger)

# Main workflow
rms_list = []
mutations_counter = 0
for mut in mutations:
    # Check mutation counter
    if mutations_counter == mutations_limit:
        break
    mutations_counter += 1

    # Log mutation
    log_mutation_execution(app_logger, mutations_counter, mutations_limit, mut)

    # Get mutation specific paths and properties
    paths = app_conf.get_paths_dic(mut)
    prop = app_conf.get_prop_dic(mut)
    paths["step3_scw"]["input_pdb_path"] = structure
    paths["step17_rmsd"]["output_xvg_path"] = paths["step17_rmsd"]["output_xvg_pat

    # Setup mutation directories
    setup_mutation_dirs(app_logger, app_conf, prop)

    # ------------------------------------------------------------
    # Spawn tasks

    # Step 3
    app_logger.info("- Step 3")
    scwrl_pc(prepared_file_path=paths["step3_scw"]["input_pdb_path"],
             output_pdb_path=paths["step3_scw"]["output_pdb_path"])

    # Step 4
    app_logger.info("- Step 4")
    pdb2gmx_pc(properties=prop["step4_p2g"], **paths["step4_p2g"])

    # Step 5
    app_logger.info("- Step 5")
    editconf_pc(input_gro_path=paths["step5_ec"]["input_gro_path"],
```

```
            output_gro_path=paths["step5_ec"]["output_gro_path"],
            distance_to_molecule=str(prop["step5_ec"].get("distance_to_molecule
            box_type=prop["step5_ec"].get("box_type", "cubic"))

# Step 6
app_logger.info("- Step 6")
solvate_pc(properties=prop["step6_sol"], **paths["step6_sol"])

# Step 7
app_logger.info("- Step 7")
grompp_pc(properties=prop["step7_gppions"], **paths["step7_gppions"])

# Step 8
app_logger.info("- Step 8")
genion_pc(properties=prop["step8_gio"], **paths["step8_gio"])

# Step 9
app_logger.info("- Step 9")
grompp_pc(properties=prop["step9_gppmin"], **paths["step9_gppmin"])

# Step 10
app_logger.info("- Step 10")
mdrun_pc(input_tpr_path=paths["step10_mdmin"]["input_tpr_path"],
         output_gro_path=paths["step10_mdmin"]["output_gro_path"],
         output_trr_path=paths["step10_mdmin"]["output_trr_path"],
         output_xtc_path=paths["step10_mdmin"]["output_xtc_path"],
         output_edr_path=paths["step10_mdmin"]["output_edr_path"],
         output_log_path=paths["step10_mdmin"]["output_log_path"])

# Step 11
app_logger.info("- Step 11")
grompp_pc(properties=prop["step11_gppnvt"], **paths["step11_gppnvt"])

# Step 12
app_logger.info("- Step 12")
mdrun_pc_cpt(input_tpr_path=paths["step12_mdnvt"]["input_tpr_path"],
             output_gro_path=paths["step12_mdnvt"]["output_gro_path"],
             output_trr_path=paths["step12_mdnvt"]["output_trr_path"],
             output_xtc_path=paths["step12_mdnvt"]["output_xtc_path"],
             output_edr_path=paths["step12_mdnvt"]["output_edr_path"],
             output_cpt_path=paths["step12_mdnvt"]["output_cpt_path"],
             output_log_path=paths["step12_mdnvt"]["output_log_path"])

# Step 13
app_logger.info("- Step 13")
grompp_pc_cpt(properties=prop["step13_gppnpt"], **paths["step13_gppnpt"])

# Step 14
```

```
    app_logger.info("- Step 14")
    mdrun_pc_cpt(input_tpr_path=paths["step14_mdnpt"]["input_tpr_path"],
                 output_gro_path=paths["step14_mdnpt"]["output_gro_path"],
                 output_trr_path=paths["step14_mdnpt"]["output_trr_path"],
                 output_xtc_path=paths["step14_mdnpt"]["output_xtc_path"],
                 output_edr_path=paths["step14_mdnpt"]["output_edr_path"],
                 output_cpt_path=paths["step14_mdnpt"]["output_cpt_path"],
                 output_log_path=paths["step14_mdnpt"]["output_log_path"])

    # Step 15
    app_logger.info("- Step 15")
    grompp_pc_cpt(properties=prop["step15_gppeq"], **paths["step15_gppeq"])

    # Step 16
    app_logger.info("- Step 16")
    mdrun_pc(input_tpr_path=paths["step16_mdeq"]["input_tpr_path"],
             output_gro_path=paths["step16_mdeq"]["output_gro_path"],
             output_trr_path=paths["step16_mdeq"]["output_trr_path"],
             output_xtc_path=paths["step16_mdeq"]["output_xtc_path"],
             output_edr_path=paths["step16_mdeq"]["output_edr_path"],
             output_log_path=paths["step16_mdeq"]["output_log_path"])

    # Step 17
    app_logger.info("- Step 17")
    rms_pc(properties=prop['step17_rmsd'], **paths['step17_rmsd'])
    rms_list.append(paths["step17_rmsd"]["output_xvg_path"])

# Step 18: Plot task
app_logger.info("- Step 18")
prop = app_conf.get_prop_dic()["step18_gnuplot"]
paths = app_conf.get_paths_dic()["step18_gnuplot"]
output_png_path = paths["output_png_path"]
output_plotscript_path = paths["output_plotscript_path"]
gnuplot_pc_plotscript(output_plotscript_path,
                      output_png_path,
                      prop,
                      mutations,
                      *rms_list)
gnuplot_pc_image(plotscript_path=output_plotscript_path,
                 output_png_path=output_png_path)

# Wait for application completion
app_logger.info("")
app_logger.info("-------------------------")
app_logger.info("- Waiting for tasks completion")
compss_barrier()

# End timer
```

```
    elapsed_time = time.time() - start_time

    # Plot results
    plot_results(output_png_path)

    # Write execution results
    show_execution_results(app_logger, elapsed_time)


############################################################
# MAIN
############################################################
%matplotlib inline
import ipywidgets as widgets

w_exec_cfg = widgets.RadioButtons(
    options=['../confs/conf_local.yaml'],
    value='../confs/conf_local.yaml',
    description='Configuration File:',
    disabled=False
)
w_entry_cfg = widgets.RadioButtons(
    options=['local', 'marenostrum'],
    value='local',
    description='Configuration Entry:',
    disabled=False
)

widgets.interact_manual(pymdsetup_main, exec_cfg=w_exec_cfg, entry_cfg=w_entry_cfg)
```

4.8 Stop Interactive PyCOMPSs

```
In [ ]: ipycompss.stop()
```

5 Post Mortem Analysis

5.1 Open PyCOMPSs Monitor

Open PyCOMPSs Monitor in a separate tab

```
In [ ]: # Embeded PyCOMPSs Monitor

    from IPython.display import HTML
    HTML('<style>#frame { width: 1300px; height: 700px; border: 1px solid black; } #frame
```

5.2 PyMDSetup Mutations plot

```
In [ ]: from IPython.display import Image
    Image("../test_local/step18_gnuplot/gplot.png")
```

5.3 Task Graph

```
In [ ]: %env LD_PRELOAD=
```

```
In [ ]: %%bash

        dot -Tpng -Gnewrank=true $HOME/.COMPSs/pymdsetup_01/monitor/complete_graph.dot > pymdse
```

```
In [ ]: from IPython.display import Image
        Image("pymdsetup_task_graph.png")
```

5.4 Paraver Trace file

```
In [ ]: %%bash

        wxparaver $HOME/.COMPSs/pymdsetup_01/trace/*.prv
```