Building Modular Parsl Workflows in Parallel Works

Contact: alvaro@parallelworks.com
1. Workflows in Parallel Works

2. Modular workflows
   a. Motivation
   b. Sweep_CSV
   c. Pipeline

3. Wrapping Parsl Apps: SimpleBashRunner
Workflows in Parallel Works

Parsl script(s) are encapsulated into a Parallel Works “workflow” or “app”
Use Parallel Works to:

- Develop
- Execute
- Share
1. Workflows in Parallel Works

2. Modular workflows
   a. Motivation
   b. Sweep_CSV
   c. Pipeline

3. Wrapping Parsl Apps: SimpleBashRunner
Modular Workflows: Motivation

Typical simple workflow

Input file(s) with input parameters defining single case

Run simulation

Case simulation result file(s)

Post-process results with another workflow?

Case simulation result file(s)

Post-process results

Case metrics, images and animations

x1=1
x2=2
...

y1=3
y2=1
...

Parallel Works Inc. Copyright 2020. All Rights Reserved.
Modular Workflows: Motivation

3 Workflows:
1. CONVERGE_RUNNER
2. PVPOST
3. CONVERGE_PV_RUNNER

Copy and edit blocks of code to make a third workflow:
- Pros:
  - Workflow is self contained
- Cons:
  - Hard to maintain
  - Need to update many workflows
  - Too many workflows
  - Hard to test
  - Slow development
  - More code
  - ...

Run and post-process in the same workflow?
Modular Workflows: Motivation

5 Workflows:
1. CONVERGE_RUNNER
2. PVPOST
3. CONVERGE_PV_RUNNER
4. OPENFOAM_RUNNER
5. OPENFOAM_PV_RUNNER

Use a different CFD tool?
Modular Workflows: Motivation

- x1 = 1
- x2 = 2
- y1 = 3
- y2 = 1
- Run different cases in parallel and merge results?

AND MORE...
1. CONVERGE_RUNNER
2. PVPOST
3. CONVERGE_PV_RUNNER
4. OPENFOAM_RUNNER
5. OPENFOAM_PV_RUNNER
6. CONVERGE_PV_SWEEP
7. OPENFOAM_PV_SWEEP
8. ...

one workflow for every compatible software tool and workflow topology combination
Modular Workflows: Motivation

Build workflows as Python modules that can be imported by other workflows
- Pipelining
- CSV Sweep
- Optimization
- Active Learning
- ...

Advantages:
- Fast development
- Easy to maintain
- Less workflows
- Less code
- ...

Less workflows!
1. CONVERGE_RUNNER
2. PVPOST
3. OPENFOAM_RUNNER
4. PIPELINE
5. SWEEP_CSV

Better than one workflow for every compatible software tool and workflow topology combination
Create a workflow script (main.py) that can be:

1. Executed directly
   - runs:
     
     ```python
     python main.py
     in /pw/jobs/job_num/
     ```

2. Imported by other workflows
   - ```python
     imported_workflow = wfbuilder.import_workflow(workflow_name)
     ```

Main parts:

1. Run workflow function(s): Imported and executed by other workflows
   - Do not wait for futures inside these functions
     - If imported cannot be executed multiple times in parallel
   - To be compatible with wfbuilder module
     - Inputs:
       - i. (Required) `wf_pwargs`: Python Namespace with functions IO
       - ii. (Optional) `wf_dir`: Workflow directory for intermediate IO, logs, etc.
     - Outputs:
       - i. Dictionary of objects with a `.result()` method were keys are output parameter names

2. Only when executed directly:
   - Load Parsl configuration
   - Load and preprocess IO
   - Run workflow function(s)
   - Wait for results

3. Only when imported → Build workflow as module

4. Execute always

```python
import parsl
from parslpw import pwconfig, pwargs

# RUN WORKFLOW FUNCTION
def run_workflow(wf_pwargs, wf_dir="./workflow"):
    # Workflow code HERE
    # ...
    # ...
    # Return dictionary where keys are output parameter names and values
    objects with `.result()` method or dictionaries in the same format
    return out_futs

if __name__ == "__main__":
    # Workflow executed directly
    # Write code HERE
    # ...
    # ...
    # Load Parsl configuration
    parsl.load(pwconfig)
    # Run workflow
    out_futs = run_workflow(pwargs)
    # Wait for results
    wfbuilder.wfconn.wait_for_futs(out_futs)

else:
    # Workflow imported by other workflow
    # Write code HERE
    # ...
    if not os.path.isdir("module_sample"):
        shutil.copytree("/pw/workflows/workflow/module_sample", "module_sample")
    import module_sample
    # ...
```
Modular Workflows: SWEEP_CSV

Example 1: SWEEP_CSV(MMS_RUNNER) ("placeholder workflow")
**Modular Workflows: SWEEP_CSV**

**MMS:**

```python
import os, shutil
import parsl
from parsl import import_pwconfig,pwargs

if not os.path.isdir("wfbuilder"):  
    shutil.copytree("/pw/modules/wfbuilder", "wfbuilder")  
import wfbuilder

def run(wf_pwargs, wf_dir = "mms_runner"):  
    os.makedirs(wf_dir, exist_ok=True)  
    print("MMS_RUNNER INPUTS:")  
    print(wf_pwargs)  
    # Define runner  
    runner = wfbuilder.pwrunners.SimpleBashRunner(  
        cmd="/bin/bash mms/mms_eval.sh",  
        cmd_arg_names=["in_mms", "out_mms"],  
        inputs={  
            "in_mms": wfbuilder.Path(wf_pwargs.in_mms),  
            "scripts": wfbuilder.Path("/pw/workflows/mms_runner/./mms" ),  
        },  
        outputs={"out_mms": wfbuilder.Path(wf_pwargs.out_mms)},  
        logs={  
            "stdout": wf_dir + "mms.out",  
            "stderr": wf_dir + "mms.err"  
        }  
    )  
    return runner.run()

if __name__ == "__main__":  
    # Runs only when executed (not when imported  
    parsl.load(pwconfig)  
    case_fut = run(pwargs)  
    case_fut["out_txt"].result()
```

**NAMESPACE:**

- `pwargs.casename = "TEST"`
- `pwargs.in_mms = "/pw/projects/mms/xy.txt"`
- `pwargs.out_mms = "/pw/project/mms/mms-TEST-date-time.txt"`

**Values:**

- `in_mms: x=0.0404, y=0.5454`
- `out_mms: x=0.0404, y=0.5454, z=0.8123`
Workflow Building: SWEEP_CSV

SWEEP_CSV:

1. Splits a CSV (in_csv) file into several case inputs files (in_txt)

2. Submits “runner” workflows in parallel such that each workflow gets a case file. Compatibility:
   - Input and output files in the right format
   - Other inputs remain constant
   - Other outputs (images, logs, etc) need to appear in the workflow output directory (wf_dir)
Workflow Building: SWEEP_CSV

Workflow inputs:

```json
{
  "import": ["mms_runner"],      (workflows to import)
  "runner":                        (workflow info)
    {
      "wfname": "mms_runner",   (name)
      "run_func": "run",        (run function)
      "in_sep": ",",           (input parameter name/value separation)
      "out_sep": "=",          (output parameter name/value separation)
      "in_excld": [],           (input parameter names to exclude from in_txt)
      "out_excld": [],          (output parameter names to exclude from out_csv)
      "wfparams": {
        "in_mms": "in_txt",    (tagged input parameter to be replaced by the Sweep_CSV)
        "out_mms": "out_txt"   (tagged output parameter to be replaced by the Sweep_CSV)
      }
    }
  }
}```
You need to wait and merge the results but you cannot do it inside the `run_csv` function:

- Return a `SweepFut` object with a `.result()` method that waits for the futures and merges all the case output files (out_txt) into a single CSV output file (out_csv)

```python
class SweepFut:
    def __init__(self, rwf_fut_list, rwf_conn, wf_pwargs):
        self.rwf_fut_list = rwf_fut_list
        self.rwf_conn = rwf_conn
        self.wf_pwargs = wf_pwargs

    def result(self):
        # Wait for results
        out_txt_paths = []
        for rwf_fut in self.rwf_fut_list:
            out_txt_paths.append(rwf_fut[ self.rwf_conn["out_txt"]].result().path)

        # Merge results in CSV
        wfbuilder.data_reformat.txts2csv(out_txt_paths, self.wf_pwargs.out_csv, exclude=self.wf_pwargs.runner[ "out_excld" ], sep=self.wf_pwargs.runner[ "out_sep" ],)
        return self.wf_pwargs.out_csv
```

### Parts of the main.py script of SWEEP_CSV

```python
import os, sys, shutil, json
import parsl
from parslw import pconfig,pwargs
from copy import deepcopy
import inspect
if not os.path.isdir("wfbuilder"):  
    shutil.copytree("/pw/modules/wfbuilder", "wfbuilder")
import wfbuilder

# Run CSV
def run_csv( wf_pwargs, wf_dir = "./sweep_csv"):  
    os.makedirs(wf_dir, exist_ok = True)
    print("Sweep CSV wf_pwargs:", flush = True)
    print(wf_pwargs, flush = True)
    # RUNS SWEEP OF MMS RUNNERS
    # DELETED CODE FOR SPACE
    return { "out_csv": SweepFut(rwf_fut_list, rwf_conn, wf_pwargs)}

if __name__ == "__main__":
    # This pwarg is only seen when executed from the form!
    with open(pwargs.sweepconf_json, 'r') as json_file:
        sweepconf = json.load(json_file)

        # Imported workflows
        if "import" in sweepconf:
            for wf_name in sweepconf["import"]:  
                rwf = wfbuilder.pwimport.import_workflow(wf_name)

        # Add runner info to workflow arguments
        pwargs.runner = sweepconf["runner"]
        parsl.load(pwconfig)
        sweep_csv_fut = run_csv(pwargs)
        sweep_csv_fut["out_csv"].result()
```
Workflow Building: PIPELINE

Example 2: PIPELINE(DOE_GEN, SWEEP_CSV(MMS_RUNNER))

Two base workflows

Two workflow topologies
Design of experiments:

# INPUTS:
pwargs.casename = "CaseName"
pwargs.dspace = "/pw/projects/pipeline/xy_dspace.json"
pwargs.method = "lhs-spacefill"
pwargs.num_samples = "50"

# OUTPUTS:
pwargs.out_csv = "/storage/mms/doe-CaseName-date-time.csv"
pwargs.out_png = "/storage/mms/doe-CaseName-date-time.png"
pwargs.out_html = "/storage/mms/doe-CaseName-date-time.html"

```javascript
{
  "x": [0, 1],
  "y": [0, 1]
}
```

<table>
<thead>
<tr>
<th>dspace</th>
<th>out_csv</th>
<th>out_png</th>
</tr>
</thead>
<tbody>
<tr>
<td>x,y</td>
<td>0.0404, 0.5454</td>
<td>![Plot 1]</td>
</tr>
<tr>
<td></td>
<td>0.0202, 0.8787</td>
<td>![Plot 2]</td>
</tr>
<tr>
<td></td>
<td>0.9898, 0.8080</td>
<td>![Plot 3]</td>
</tr>
</tbody>
</table>
Workflow Building: PIPELINE

```json
{
  "x": [0, 1],
  "y": [0, 1]
}
```
### Workflow Building: PIPELINE

- **Pipeline:**
  - Runs a list of workflows in order piping the output of the previous workflows to the input of the next workflow(s)
  - Only waits for the required DataFutures
  - Returns a dictionary with remaining DataFutures and completed results

```python
if __name__ == "__main__":
    # This pwarg is only seen when executed from the form!
    with open(pwargs.pipeconf_json, 'r') as json_file:
        pwargs.pipeconf = json.load(json_file)

    # Imported workflows!
    if "import" in pwargs.pipeconf:
        for wf_name in pwargs.pipeconf["import"]:
            rwf = wfbuilder.pwimport.import_workflow(wf_name)

    parsl.load(pwconfig)
    wfbuilder.wfconn.wait_for_futs(run_pipeline(pwargs))
```

```python
# Run pipeline
def run_pipeline(wf_pwargs, wf_dir = "./pipeline"):
    os.makedirs(wf_dir, exist_ok = True)
    print("Pipeline wf_pwargs:", flush = True)
    print(wf_pwargs, flush = True)
    pipeconf = wf_pwargs.pipeconf
    wf_futs = {}
    for wi, wf_info in enumerate(pipeconf["pipeline"]):
        # Import workflow:
        wf = wfbuilder.pwimport.import_workflow(wf_info["wfname"])  
        wf_run_func = getattr(wf, wf_info["run_func"])       
        next_wf_pwargs = Namespace(**wf_info["wfparams"])
        if wi > 0:
            # Depends on the previous workflows
            # Get current workflow input from previous workflow outputs
            # Get workflow connections (dependencies)
            for pwi in reversed(range(wi)):
                # For previous workflow index (wfi)
                next_wf_pwargs, wf_conn = wfbuilder.wfconn.get_wf_pwargs(vars(next_wf_pwargs), pipeconf["pipeline"][pwi]["wfparams"])
            # Make sure all dependencies are ready from previous workflows:
            for fut_key in wf_conn.keys():
                wf_futs[pwargs.pipeconf["pipeline"][pwi]["wfname"]][fut_key].result()
        # Run workflow:
        wf_futs[wf_info["wfname"]].run_func(next_wf_pwargs, wf_dir = wf_dir + ""/
                                            + wf_info["wfname"])
        prev_wf_info = wf_info
    return wf_futs
```

main.py script of PIPELINE
Workflow Building: PIPELINE

Any workflow parameter value that corresponds to a workflow parameter key from a previous workflow will be replaced by the corresponding parameter value.
1. Workflows in Parallel Works

2. Modular workflows
   a. Motivation
   b. Sweep_CSV
   c. Pipeline

3. Wrapping Parsl Apps: SimpleBashRunner
Wrapping Parsl Apps: SimpleBashRunner

Build wrappers around Parsl Apps to execute tasks every time a Parsl App is executed

SimpleBashRunner object:
- Runs a `bash_app`
- Builds and runs a bash command
  - `cmd cmd_args`
- IO are defined as dictionaries
- Streams standard output and error files from remote VM to local (PW)
- Run command as a given user
- Implements extra logging for debugging
- Writes resource information in the remote VM
- Returns a dictionary with the DataFutures

```python
crunner = wfbuilder.pwrunners.SimpleBashRunner("bash scripts/run.sh
cmd_arg_names = ["in_zip", "lic_server", "np", "out_zip"],
inputs = {
  "in_zip": wfbuilder.Path(wf_pwargs.in_zip),
  "lic_server": wf_pwargs.lic_server,
  "np": wf_pwargs.np,
  "scripts": wfbuilder.Path("/pw/workflows/converge_runner/./scripts")
},
outputs = {
  "out_zip": wfbuilder.Path(wf_pwargs.out_zip)
},
logs = {
  "stdout": wf_dir + "/std.out",
  "stderr": wf_dir + "/std.err"
},
stream_host = "localhost",
stream_port = os.environ["PARSL_CLIENT_SSH_PORT"],
user = "cluster",
write_pool_info = True
}
crunner_fut = crunner.run()

{"out_zip": <DataFuture at 0x7f2f638c11d0 state=pending>}
```
SUMMARY

Parsl App Wrappers:
- Run tasks every time a Parsl App is executed

Modular workflows:
- Built as a Python modules that can be executed directly or imported
- Workflow functions return futures and do not wait for results
- Only wait for results when executed directly (`if __name__ == "__main__"`)
Thanks for your attention!

Questions?

Contact: alvaro@parallelworks.com