Automatic Benchmarking with JUBE

Sebastian Lührs
s.luehrs@fz-juelich.de
Jülich Supercomputing Centre
Forschungszentrum Jülich GmbH

Reykjavik, August 25th, 2017
Automatic benchmarking?

Alternatives:

- **Manual benchmarking:**
  - Easy to use
  - Time-consuming
  - Very error-prone

- **Benchmark specific script solution:**
  - Optimized
  - Changes can be time-consuming
  - Portability problems

Why should I spend time writing additional configuration files?
Can you run your benchmark every day, using ten different parameterizations?
Was the last run for optimization level three? …

JUBE provides a generic workflow and parameter handling environment, but also supports more flexible and specialised approaches.
What is JUBE?

- Generic, configurable environment to run, monitor and analyse benchmarks in a systematic way
- Also usable for testing or production scenarios
DEMO
JUBE availability

Jötunn:

- tar -xf JUBE.tar.gz
- cd JUBE-2.1.4
- python setup.py install --user

Other systems:

- www.fz-juelich.de/jsc/jube
- Dependency: Python 2.6 / Python 2.7 / Python 3.2, ...
- Examples are included
Command line access

Start a new benchmark run

- `jube run benchmark.xml`

Continue an existing benchmark run

- `jube continue benchmark_dir [--id <id>]`

Analyse the benchmark data

- `jube analyse benchmark_dir [--id <id>]`

Create and show result representation

- `jube result benchmark_dir [--id <id>]`
Help?!

Online documentation and tutorial
- www.fz-juelich.de/jsc/jube

Info mode
- jube info benchmark_dir [--id <id>] [--step <stepname>]

Command line accessible glossary
- jube help <keyword>

Logs
- jube log benchmark_dir [--id <id>] [--command <cmd>]

Debug mode
- jube --debug run|continue|analyse|result ...

Verbose mode
- jube -v[vv] run ...
HowTo: General file layout

<?xml version="1.0" encoding="UTF-8" ?>
<jube>
  <benchmark name="..." outpath="...">
    <parameterset/>
    <fileset/>
    <substituteset/>
    <patternset/>
    <step/>
  </benchmark>
</jube>

>> jube help general_structure
HowTo: Sets

- Main JUBE information storage technique
- Four different types of sets are available
  - `<parameterset>` Parameter storage
  - `<fileset>` Define all available files
  - `<substituteset>` Define file substitution
  - `<patternset>` Define the analyse pattern
- Set names must be unique
- Can be initialised by using an additional configuration file
- Available `<parameterset>`, `<fileset>` and `<substituteset>` are used and combined within a `<step>`
- Available `<patternset>` are used within `<analyse>`

```bash
>> jube help <setname>_tag
```
HowTo: Command execution

- `<do>...</do>` holds the executable commands
- All commands must use SHELL syntax (they will be executed by using `/bin/sh`)
- JUBE parameter can be used by using `$parametername`
- Parameter will be expanded in a pre-processing step
- Environment parameter can also be used
- JUBE stops execution if the command’s return code fails
- Commands will only be executed once
- All `<do>` within the same `<step>` shares the same environment
DEMO

(/home/s.luehrs/JUBE_example)
Key Concept: Workflow creation

- Dependency driven step structure
- Parameter based expansion of steps

```xml
<parameterset name="preset">
  <parameter name="const">0</parameter>
</parameterset>

<parameterset name="compset">
  <parameter name="p1">a,b</parameter>
</parameterset>

<parameterset name="execset">
  <parameter name="p2">1,2</parameter>
</parameterset>

<step name="preprocess">
  <use>preset</use>
</step>

<step name="compile">
  <use>compset</use>
</step>

<step name="execute" depend="preprocess,compile">
  <use>execset</use>
</step>
```
DEMO

(/home/s.luehrs/JUBE_example)
Key Concept: Directory and data handling

- Each parameter/step combination runs in a separate sandbox directory
- Dependent steps can be accessed using sym. links
DEMO

(/home/s.luehrs/JUBE_example)
Key Concept: Data re-usage

- Separation of platform dependent and independent configuration options

```xml
<parameterset name="set"
    init_with="platform.xml">
  ...
</parameterset>

<step ...
  <use from="platform.xml">
    ...
  </use>
  <do>$submit $submit_script</do>
</step>

<include from="platform.xml" />
```
DEMO

(/home/s.luehrs/JUBE_example)
HowTo: Analyse

- Files will be analysed by using regular expressions which are defined by the given patterns.
- Multiple occurrences of the same pattern create statistical values (average, minimum, maximum etc.)

```xml
<analyser name="...">  
  <use>...</use>  
  <analyse step="...">  
    <file>...</file>  
  </analyse>  
</analyser>
```

- `analyser area`
- `used patternset`
- `step which should be analysed`
- `list of files`

`>> jube help analyser_tag`
## HowTo: Result creation

```xml
<result>
  <use>...</use>
  <table name="...">
    <column>...</column>
  </table>
</result>
```

- **result area**
- **used analyser**
- **table result type definition**
- **column definition**

```
>> jube help result_tag
>> jube help table_tag
```
DEMO

(/home/s.luehrs/JUBE_example)
HowTo: Job submission

- A job template and substitution can be used to generalize the job submission process
- `<do>…</do>` is used to submit the job
- `<do>` returns immediately after the job was submitted. To wait for its execution use: `<do done_file="...">`
- The marker file, given by `done_file`, must be generated by the job script after the parallel part was executed
- `continue` triggers JUBE to check all available marker files
DEMO

(/home/s.luehrs/JUBE_example)