JUBE

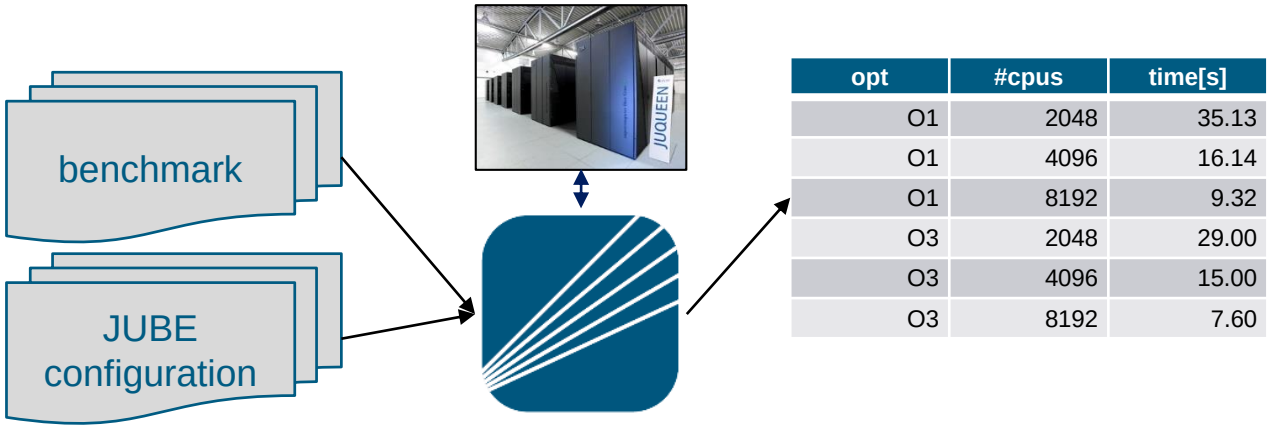
A Flexible, Application- and Platform-Independent Environment for Benchmarking

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

Cy-Tera/LinkSCEEM HPC Administrator Workshop, 19 - 21 January 2015

What is JUBE?

- Environment to run, monitor and analyse benchmarks (command line based)
- Motivation: objective comparison of computer architectures
- Use cases: procurements, monitoring the effects of system and configuration changes



opt	#cpus	time[s]
O1	2048	35.13
O1	4096	16.14
O1	8192	9.32
O3	2048	29.00
O3	4096	15.00
O3	8192	7.60

JUBE history

- Development started in 2004
- JUBE version 1

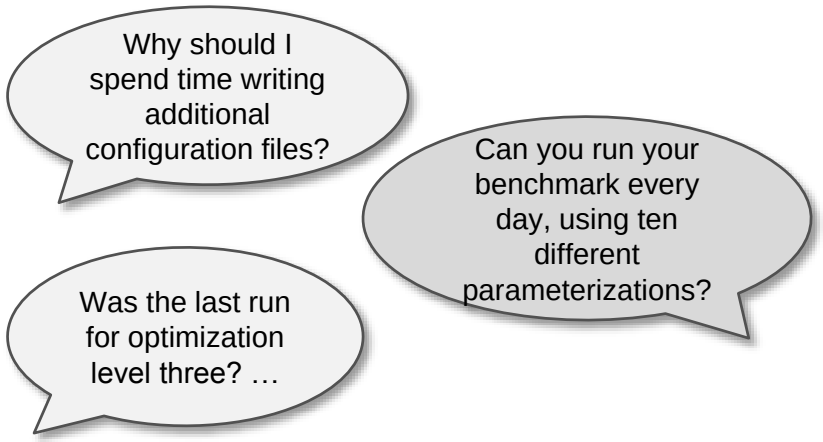
  - Perl based
  - Used in many European projects like DEISA and PRACE
- 2014 complete new release: JUBE version 2
  - Python based
  - New, more flexible input file layout
  - New command line options
  - Current version: 2.0.2

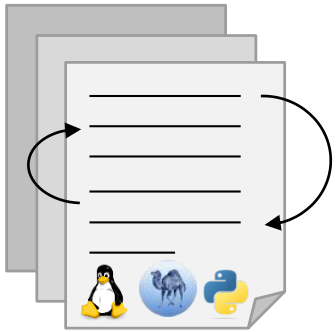
! JUBE 1 file format JUBE 2 file format !

Why JUBE?

Alternatives:

- Manual benchmarking:
  - Easy to use
  - Time-consuming
  - Very error-prone
- Benchmark specific script solution:
  - Optimized
  - Changes can be time-consuming
  - Portability problems

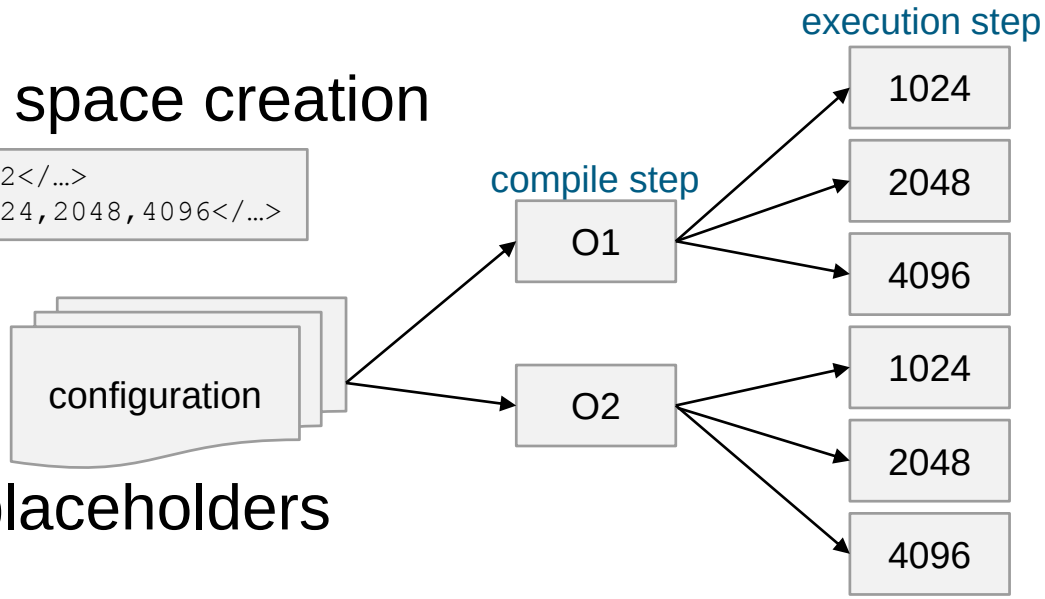




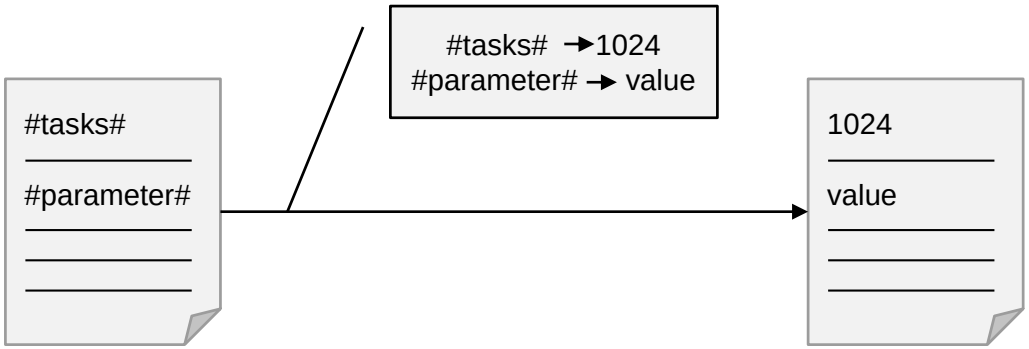
Main concepts

- Easy parameter space creation

```
<parameter name="opt">O1,O2</…>
<parameter name="tasks">1024,2048,4096</…>
```



- Substitution of placeholders

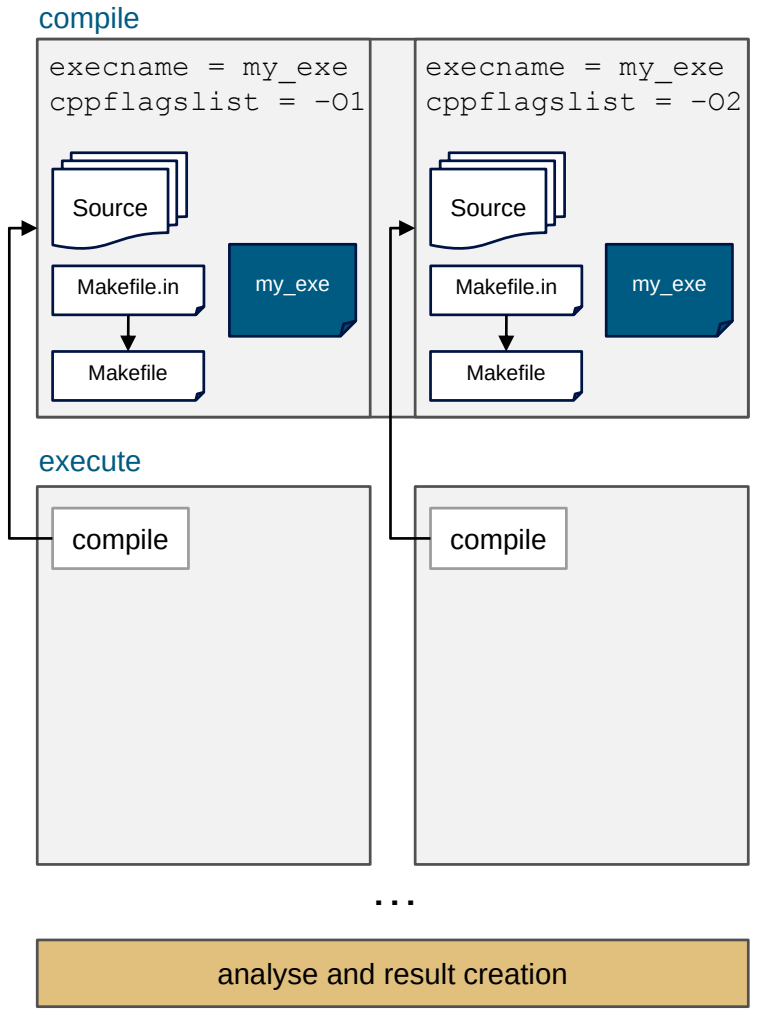


XML input file format

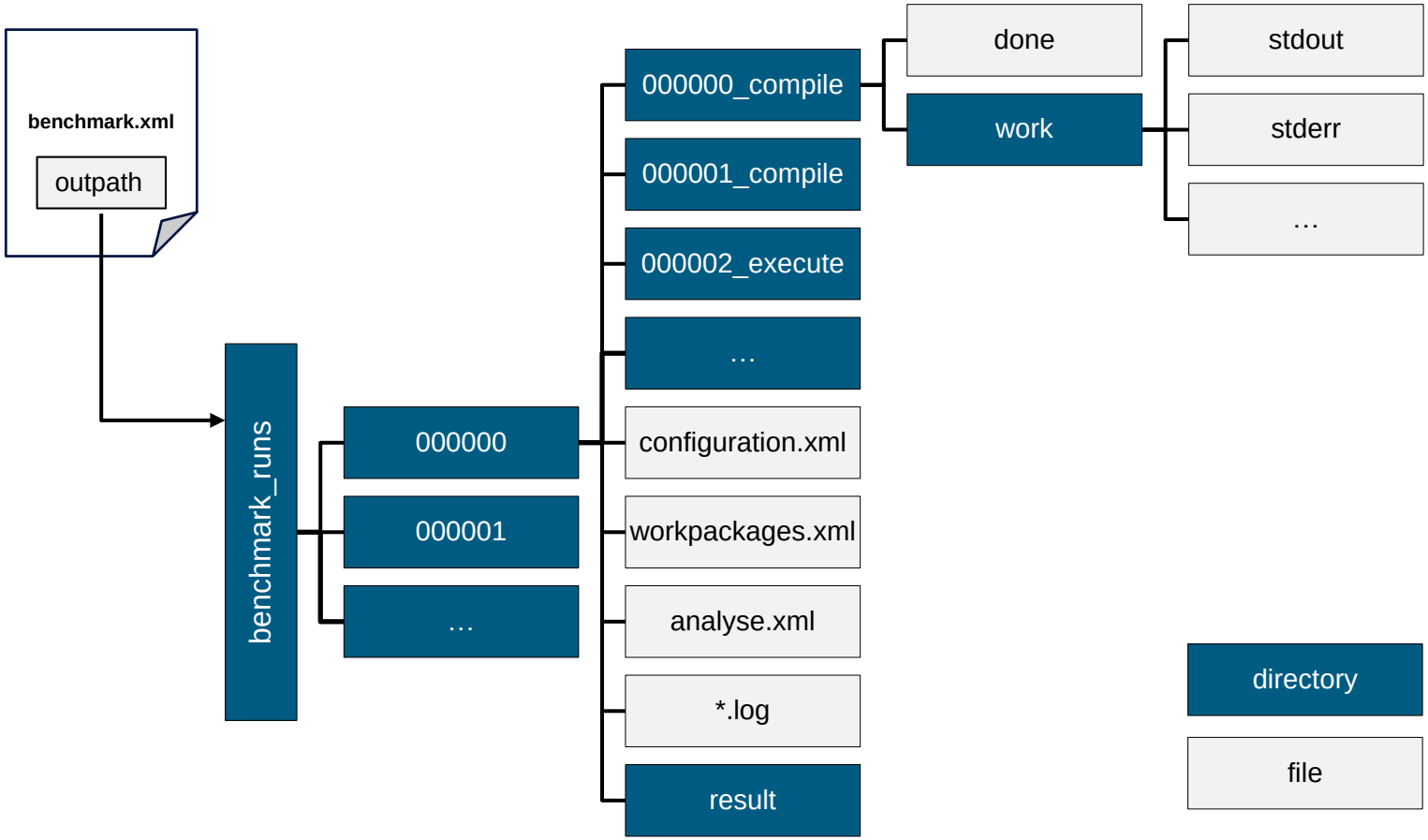
- XML must be well formed:
  - Only one root element: `<jube>`
  - `<a><b>…</a></b>` not allowed
  - Every tag must be closed (`<a>…</a>` or `<a/>`)
  - `<a attr1="…" attr1="…"/>` not allowed
  - `<a attr=… />` not allowed (missing `""`)
- Normal XML comment syntax can be used:
  - `<!-- … -->`
- JUBE tags can be validated using available DTD, schema, or RELAX NG file

Workflow

```
<jube>
  <benchmark name="bench" outpath="./benchmark_runs">
    <parameterset name="compileset">
      <parameter name="execname">my_exe</parameter>
      <parameter name="cppflagslist">
        -O1,-O2
      </parameter>
    </parameterset>

    <fileset name="sources">
      <copy>src/*</copy>
    </fileset>

    <substituteset name="compilesub">
      <iofile in="Makefile.in" out="Makefile" />
      <sub source="#PROGNAME#" dest="$execname" />
    </substituteset>

    <step name="compile">
      <use>compileset</use>
      <use>sources</use>
      <use>compilesub</use>
      <do>make OPT=$cppflagslist</do>
    </step>

    <step name="execute" depend="compile">
      …
```



Directory structure



Command line access

1. Start a new benchmark run
   - `jube run benchmark.xml`
2. Continue an existing benchmark run
   - `jube continue benchmark_dir [--id <id>]`
3. Analyse the benchmark data
   - `jube analyse benchmark_dir [--id <id>]`
4. Create and show result representation
   - `jube result benchmark_dir [--id <id>]`

Help?!

1. Online documentation and tutorial
   - `www.fz-juelich.de/jsc/jube`
2. Info mode
   - `jube info benchmark_dir [--id <id>] [--step <stepname>]`
3. Command line accessible glossary
   - `jube help <keyword>`
4. Logs
   - `jube log benchmark_dir [--id <id>]`
5. Debug mode
   - `jube --debug run|continue|analyse|result …`

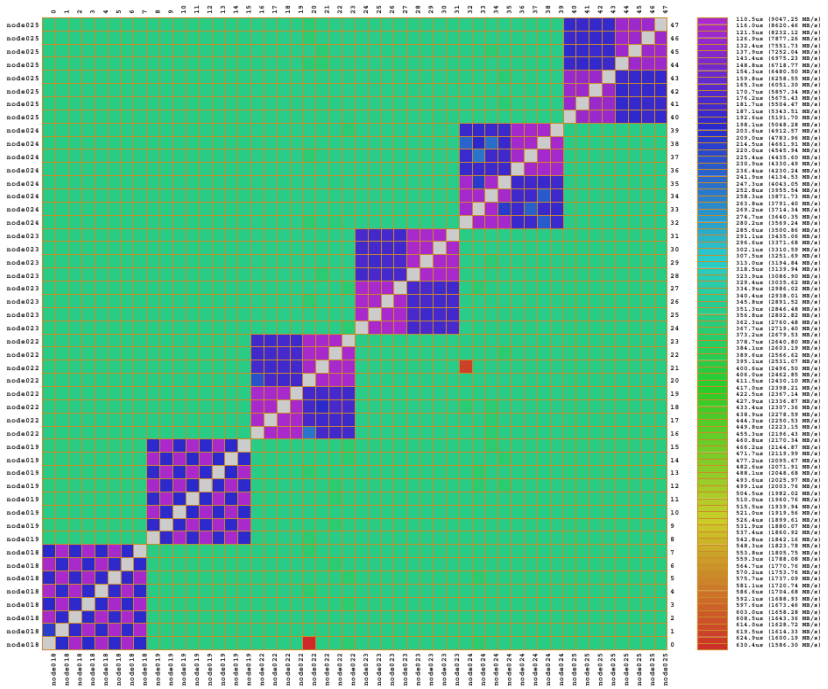
Platform independent benchmarking

Example JUBE Linktest implementation

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>
  <benchmark name="linktest" outpath="benchmark_runs">
    <comment>A linktest JUBE benchmark</comment>
    …
  </benchmark>
</jube>
```

Linktest:

- Parallel ping-pong test between all possible MPI connections of a machine
- www.fz-juelich.de/jsc/linktest



Example JUBE Linktest implementation

compile

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>
  <benchmark name="linktest" outpath="benchmark_runs">
    <comment>A linktest JUBE benchmark</comment>

    <!-- ============================ compile ============================  -->
    <fileset name="source">
      <link>fzjlinktest-1.1p5.tar.gz</link>
      <prepare>tar -xf fzjlinktest-1.1p5.tar.gz</prepare>
    </fileset>

    <step name="compile" export="true">
      <use>source</use>
      <do>module load intel-para</do>
      <do>module load SIONlib</do>
      <do work_dir="fzjlinktest/src">make -f Makefile_LINUX SIONLIB_INST=$$EBROOTSIONLIB</do>
    </step>

  …
```

link files to work directory

load modules and run make

Example JUBE Linktest implementation

execute

```
…
  <benchmark name="linktest" outpath="benchmark_runs">
    …
    <!-- ============================ execute ============================  -->
    <parameterset name="systemParameter" init_with="platform.xml">
      <parameter name="nodes" type="int">1,2,4</parameter>
      <parameter name="executable">mpilinktest</parameter>
      <parameter name="mail">s.luehrs@fz-juelich.de</parameter>
      <parameter name="taskspernode" type="int">28</parameter>
      <parameter name="timelimit">00:15:00</parameter>
      <parameter name="args_exec">-k $msg_size</parameter>
    </parameterset>
    <parameterset name="linktestParameter">
      <parameter name="msg_size" type="int">128,1024,10240</parameter>
    </parameterset>
    <fileset name="binaries">
      <link rel_path_ref="internal"
            directory="compile/fzjlinktest/src">mpilinktest,pingponganalysis</link>
    </fileset>
    <step name="execute" depend="compile">
      <use from="platform.xml">executeset</use>
      <use from="platform.xml">executesub</use>
      <use from="platform.xml">jobfiles</use>
      <use>systemParameter</use>
      <use>binaries</use>
      <use>linktestParameter</use>
      <do done_file="$done_file">$submit $submit_script</do>
    </step>
```

platform specific parameters

linktest specific parameters

create internal file link

submit job

Example JUBE Linktest implementation

postprocess

```
…
  <benchmark name="linktest" outpath="benchmark_runs">
    …

    <!-- ========================= postprocess ========================== -->
    <step name="postprocess" depend="compile,execute">
      <use>binaries</use>
      <do>./pingponganalysis -p execute/pingpong_results_bin.sion</do>
    </step>
  …
```

create report

Example JUBE Linktest implementation

analyse

```
…
  <benchmark name="linktest" outpath="benchmark_runs">
    …
    <!-- ============================ analyse ============================  -->
    <patternset name="pattern">
      <pattern name="min_time" unit="us">RESULT: Min Time Stest:\s*$jube_pat_fp</pattern>
      <pattern name="max_time" unit="us">RESULT: Max Time Stest:\s*$jube_pat_fp</pattern>
      <pattern name="avg_time" unit="us">RESULT: Avg Time Stest:\s*$jube_pat_fp</pattern>
    </patternset>

    <analyzer name="analyse">
      <use>pattern</use>
      <analyse step="execute"><file>stdout</file></analyse>
    </analyzer>

    <result>
      <use>analyse</use>
      <table name="result" style="pretty" sort="nodes,msg_size">
        <column>nodes</column>
        <column title="msg_size[kB]">msg_size</column>
        <column format=".3f">min_time</column>
        <column format=".3f">max_time</column>
        <column format=".3f">avg_time</column>
      </table>
    </result>
  …
```

regular expressions

files to analyse

create a result table

Example JUBE Linktest implementation

Result:

nodes	msg_size[kB]	min_time[us]	max_time[us]	avg_time[us]
1	128	30.319	35.842	32.858
1	1024	227.849	261.505	244.477
1	10240	230.352	315.825	243.127
2	128	27.339	33.339	30.144
2	1024	174.165	179.529	176.697
2	10240	174.522	180.999	177.264
4	128	26.504	33.816	29.742
4	1024	173.330	181.675	176.864
4	10240	172.814	181.834	176.962

Available JUBE (v. 1) benchmarks

DEISA benchmark suite: (eDeisa, Deisa2)

- GADGET, Ramses, Fenfloss, IFS, NEMO, NAMD, IQCS, CPMD, Quantum_ESPRESSO, GENE, PEPC, BQCD, SU3_Ahiggs

PABS: PRACE application benchmark suite (PRACE-PP/1IP)

- Code_Saturne, CP2K, CPMD, EUTERPE, GADGET, GROMACS, NAMD, NEMO, NS3D, QCD, Quantum_ESPRESSO, WRF, ALYA, AVBP, ELMER, GPAW, HELIUM, OCTOPUS, PEPC, SPECFEM3D, TRIPOLI_4
- Synthetic codes: bonnie, euroben, hpcc, ior, psnap, selfish, skampi, smb, stream2

PUABS: PRACE Unified Application benchmark suite (PRACE-2IP)

- ALYA, CP2K, Code_Saturne, GADGET, GENE, GPAW, GROMACS, NAMD, NEMO, QCD, Quantum_ESPRESSO, SPECFEM3D

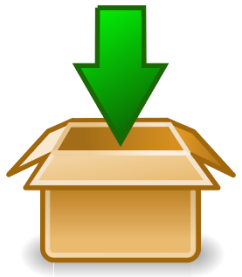
Available JUBE (v. 2) benchmarks

- mdtest: Metadata test benchmark
- IOR: (InterleavedOrRandom) I/O benchmark
- HPL: High-Performance Linpack Benchmark
- LinkTest: Parallel MPI PingPong Test

Where to start?

Website (download and tutorials):

- `www.fz-juelich.de/jsc/jube`



Versions:

- **1.x** older JUBE release, Perl based
- **2.x** new release, Python based
- **1.x** to **2.x** converter included in JUBE v. 2

Prerequisites:

- OS: Linux
- Python 2.6*, Python 2.7, Python 3.2 (or a more recent version)

* `argparse` module needed

Installation:

- `python setup.py --user`

install into `$HOME/.local`

Development and upcoming features

- Reusing steps
  - E.g. reuse existing binaries
- More flexible parameter creation
  - Create new parameter while running the benchmark
- High-level job submission API
  - Instead of generic <do>

Contact

- jube.jsc@fz-juelich.de
- s.luehrs@fz-juelich.de

Thank you for your attention!