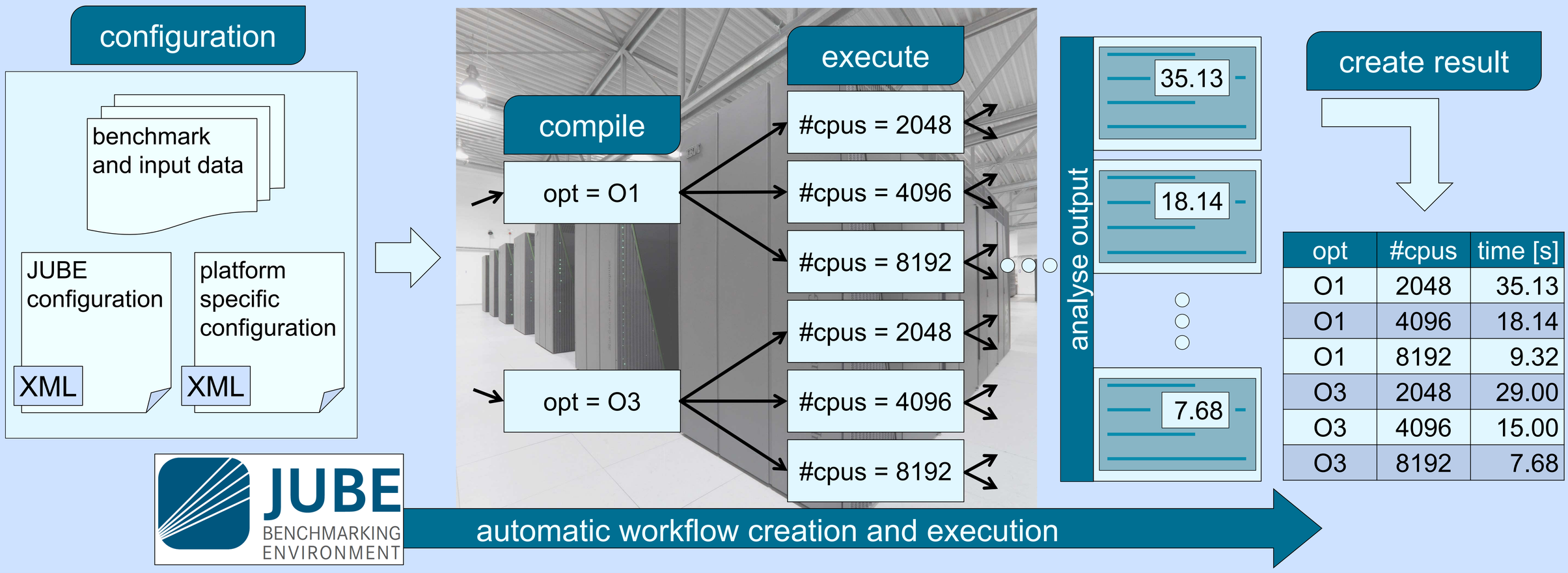
JUBE

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

Contact: Sebastian Lührs, Kay Thust, Alexander Schnurpfeil, Stephan Graf, Wolfgang Frings
{s.luehrs, k.thust, a.schnurpfeil, st.graf, w.frings}@fz-juelich.de



Introduction

Automating benchmarks is important for **reproducibility** and hence **comparability**, which is the major intent when performing benchmarks. Furthermore managing different combinations of parameters is error-prone and often results in a significant amount of work especially if the parameter space gets large.

In order to alleviate these problems, JUBE supports performing and analysing benchmarks in a **systematic way**. It allows adapting custom workflows to new architectures easily.

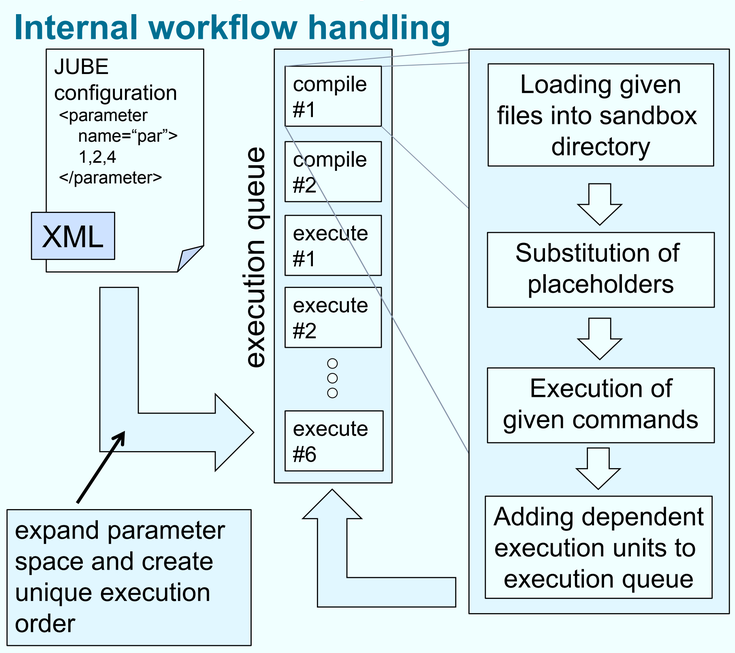
Because of the generic layout JUBE can also be used for **testing** or **production** scenarios whenever a structured workflow generation and evaluation is needed.

The new **Python based** version of JUBE enhances and restructures the older Perl based version[1]. It was used in many projects like DEISA and PRACE.

Key concept and strategy of JUBE

- **Separation of data and commands** in a way that the same commands are executed with different input data in a similar manner
- Parameters span a **multi-dimensional parameter space**, which can be used for substitution of placeholders in given templates
- Definition of parameters creates an **independence of applications and platforms**, and generates reproducible and comparable results
- XML based input file layout

Internal workflow handling

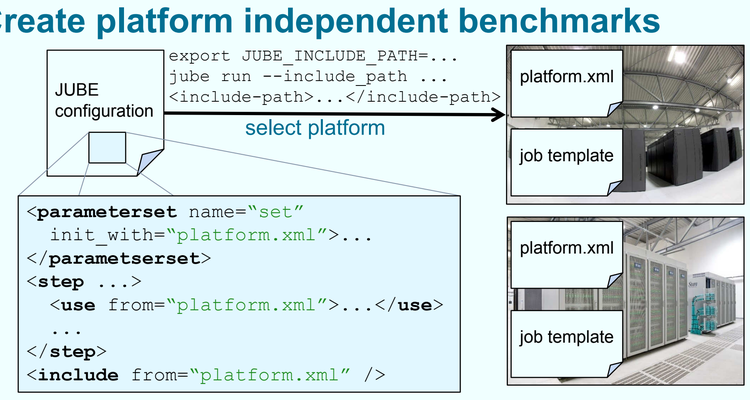


Input file layout

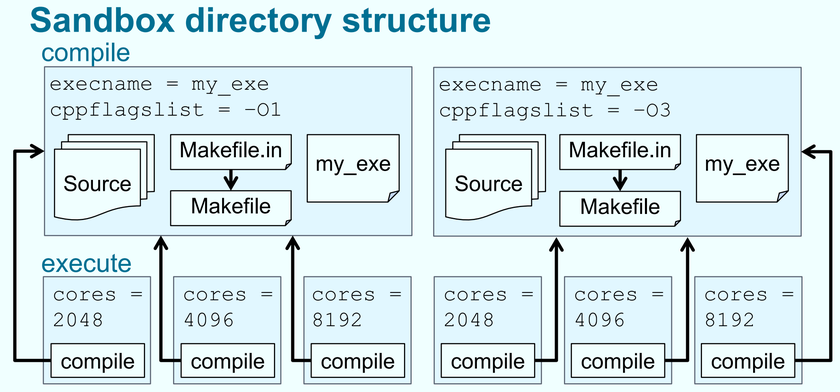
```
<jube>
  <benchmark name="bench" outpath="dir">
    <parameterset name="compileparameter"> <!-- compile parameter -->
      <parameter name="flags">-O1,-O3</parameter>
      <parameter name="execname">my_exe</parameter>
    </parameterset>
    <fileset name="sources"> <!-- ext. files used by benchmark -->
      <copy>sources/*</copy>
    </fileset>
    <substituteset name="compilesub"> <!-- file substitution -->
      <iofile in="Makefile.in" out="Makefile" />
      <sub source="#EXECNAME#" dest="$execname" />
    </substituteset>
    ...

    <step name="compile"> <!-- compile step definition -->
      <use>compileparameter</use>
      <use>sources</use>
      <use>compilesub</use>
      <do>make OPT=$flags</do>
    </step>
    <step name="execute" depend="compile">...</step> <!-- depend -->

    <patternset name="time_pattern"> <!-- regular expression -->
      <pattern name="time" unit="s">execution_time: (.+?) </pattern>
    </patternset>
    <analyser name="analyse"> <!-- scan definition -->
      <use>time_pattern</use>
      <analyse step="execute"><file>stdout</file></analyse>
    </analyser>
    <result> <!-- result layout -->
      <table name="result" sort="flags,cores">
        <column>flags</column>
        ...
      </table>
    </result>
  </benchmark>
</jube>
```

- Set definitions are used to create collections of parameters, files, substitutions or patterns
- Sets will be combined and executed within the given step environment which also provides the execution commands
- Parameter lists are automatically expanded
- Dependencies between different steps possible
- Support for asynchronous execution to **allow job submission**
- **Platform specific configuration files** can be used to provide platform wide parameter- or filesets

Create platform independent benchmarks



Sandbox directory structure



- Each parameter/step combination **runs in a separate directory**
- External additional files can be linked or copied
- Dependent step directories can be accessed using automatic generated symbolic links

Use case: System monitoring

JUBE is used in the European project DEEP-ER[2] for different purposes: On the one hand, modifications of software or hardware potentially also change the system's baseline performance, which impairs the comparability of benchmarks over time. Hence JUBE is used for **continuous monitoring** to distinguish system changes from application changes. On the other hand, classical application benchmarking is conducted to **compare different I/O strategies and libraries**.

In the project, JUBE configurations for *HPL*[3], *IOR*[4], *mdtest*[5] and the *MPI Linktest*[6] benchmark were created.

Implementation and availability

- Implemented in Python (compatible with Python 2.6, 2.7, 3.2 or any newer version)
- Command line accessible options
- Open Source (GPLv3)
- Documentation and tutorial available online
- Download: **www.fz-juelich.de/jsc/jube**
- Benchmark configuration examples: **github.com/FZJ-JSC/jube-configs**
- Contact: **jube.jsc@fz-juelich.de**

Outlook and future work

- High-level HPC queue management system configuration and communication options
- Benchmark run file reusage

[1] W.Frings et al., A Flexible, Application- and Platform-Independent Environment for Benchmarking, *Parallel Computing: From Multicores and GPU's to Petascale, IOS Press, 2010, Advances in Parallel Computing Volume 19*
[2] www.deep-er.eu
[3] www.netlib.org/benchmark/hpl
[4] sourceforge.net/projects/ior-sio
[4] sourceforge.net/projects/mdtest
[5] www.fz-juelich.de/jsc/linktest