Wikiprint Book

Title: Integrate applications in JUBE

Subject: DEEP - Public/User_Guide/JUBE

Version: 22

Date: 25.04.2024 17:03:07

Table of Contents

Integrate applications in JUBE **3**
Introduction 3
Example application 3
JUBE compile input file 3
Compile application 4
JUBE run job input file 5

Integrate applications in JUBE

Introduction

- JUBE Benchmarking Environment: provides a script based framework to easily create benchmark sets, run those sets on different computer systems and evaluate the results.
- Documentation: see ?JUBE Online Documentation
- Current version available in the DEEP, DEEP-ER and DEEP-EST systems: please check ?here
- Example commands:

```
ssh user@deep

$ jube info benchmark_directory
$ jube status benchmark_directory [-i run_id]
$ jube run benchmark_xml_file [--tag tag1 tag2 ...]
$ jube analyse benchmark_directory [-i run_id]
$ jube result benchmark_directory [-i run_id]
```

Example application

```
ssh manzano@deep

cd /usr/local/deep-er/sdv-benchmarks/applications/MAXW-DGTD
```

JUBE compile input file

See for instance the JUBE xml file of the example application:

```
vim MAXW-DGTD-jube-master-SDV.compile.xml
```

Important parts in the file are:

1) Platform

```
  <include-path>
   <path>/usr/local/jube2/platform/deep</path>
 </include-path>
```

Under /usr/local/jube2/platform/deep there are a series of files with default values for the DEEP, DEEP-ER and DEEP-EST systems:

```
deep-chainJobs.sh
submit.job.in
platform.xml
```

These files won't be modified but the default values can be overwritten in the JUBE xml file.

2) Benchmark name and outpath

```
<benchmark name="MAXW-DGTD-compile" outpath="./compile">
```

The outpath describes the benchmark run directory (relative to the position of the input file). This directory will be managed by JUBE and will be automatically created if it doesn?t exist. The directory name and position are very important, because they are the main interface to communicate with your benchmark, after it was submitted.

3) Source files

```
    <fileset name="sources">
     <copy>MAXW-DGTD.tar.gz</copy>
```

```
    <prepare>tar -xzf MAXW-DGTD.tar.gz</prepare>
  </fileset>
```

The source files will be copied to the user sandbox directory and untar before the compilation.

4) Parameterset

```
   <parameterset init_with="MAXW-DGTD_specs.xml" name="systemParameter">
    <parameter name="modules">intel/15.2.164 parastation/intel-5.1.4-1_1_g064e3f7</parameter>
    <parameter name="targetdir">$jube_benchmark_home/executable/k$k</parameter>
    <parameter name="FC" type="string" >mpif90</parameter>
    <parameter name="FFLAGS" type="string" >-align dcommons -openmp -no-opt-prefetch -O3 -r8 -axCORE-AVX2 -fpp</parameter
  </parameterset>
```

The values in the set of parameters with name "systemParamter" will be initialized with the default values found in the file "MAXW-DGTD_specs.xml". This last file takes in turn some of the default values of platform.xml and overwrites them. See the file for more information.

We can reference other parameters with $ like in the example $k or $jube_benchmark_home. $jube_benchmark_home is a JUBE variable with the value of the original input file location and $k is 2:

```
   <parameterset init_with="MAXW-DGTD_specs.xml" name="MAXW-DGTDParameter-head">
    <parameter name="k" type="int" >2</parameter>
  </parameterset>
```

So the value of the targetdir parameter will be /usr/local/deep-er/sdv-benchmarks/applications/MAXW-DGTD/executable/k2.

5) Step compile

```
   <step name="compile">
    <use>sources</use>
    <use from="MAXW-DGTD_specs.xml">MAXW-DGTDMakefileFile</use>
    <use from="platform.xml">compileset</use>
    <use>systemParameter</use>
    <use>MAXW-DGTDParameter-head</use>
    <use from="MAXW-DGTD_specs.xml">MAXW-DGTDMakefileSub</use>
    <do>module purge; module load $modules; export LD_LIBRARY_PATH=/opt/parastation/mpi2/lib:/usr/local/deep-er/sdv-bench
  </step>
```

In the step compile we tell which parameters and source files to use with the tag <use></use>. The <do></do> contains a single shell command. This command will run inside of a sandbox directory environment (inside the outpath directory tree).

6) Step copy_exec

```
   <step depend="compile" name="copy_exec">
    <use>systemParameter</use>
    <use>MAXW-DGTDParameter-head</use>
    <do>mkdir -p $targetdir</do>
    <do>cp -p compile/MAXW-DGTD/exe/gdL$k $targetdir/gdL$k-jubeID_$jube_benchmark_id</do>
    <do>rm -f $targetdir/gdL$k; ln -s $targetdir/gdL$k-jubeID_$jube_benchmark_id $targetdir/gdL$k</do>
  </step>
```

The copy_exec step depends on the compile step and will wait until it is finished for copying the executables resulting from the compilation to the $targetdir directory.

Compile application

```
jube run MAXW-DGTD-jube-master-SDV.compile.xml
```

A series of output files will be generated and stored under /usr/local/deep-er/sdv-benchmark/applications/MAXW-DGTD/compile/benchmark_id/. The structure is as follows:

```
compile                 # the given outpath
|
+- 000000               # the benchmark id
  |
  +- configuration.xml # the stored benchmark configuration
  +- workpackages.xml  # workpackage information
  +- run.log           # log information
  +- 000000_compile    # the workpackage
  |  |
  |  +- done           # workpackage finished marker
  |  +- work           # user sanbox folder
  |     |
  |     +- stderr      # standard error messages of used shell commands
  |     +- stdout      # standard output of used shell commands
  +- 000000_copy_exec  # the workpackage
     |
     +- done           # workpackage finished marker
     +- work           # user sanbox folder
        |
        +- stderr      # standard error messages of used shell commands
        +- stdout      # standard output of used shell commands
```

The executable resulting of the compilation will be stored under /usr/local/deep-er/sdv-benchmark/applications/MAXW-DGTD/executable/.

JUBE run job input file

See for instance the JUBE xml file of the example application:

```
vim MAXW-DGTD-jube-master-SDV.modules.test.xml
```

Important parts in the file are:

1) Use of tags

Notice the 2 different parameter sets:

```
    <parameterset name="MAXW-DGTDParameter-head" init_with="MAXW-DGTD_specs.xml" tag="!woman">
   [...]
   </parameterset>


   <parameterset name="MAXW-DGTDParameter-woman" init_with="MAXW-DGTD_specs.xml" tag="woman">
   [...]
   </parameterset>
```

You can tell JUBE to use one parameter set or another depending on the tag. For doing so you can specify it with the jube2 run command:

```
jube run MAXW-DGTD-jube-master-SDV.modules.test.xml --tag woman
```

will use MAXW-DGTDParameter-woman,

```
jube run MAXW-DGTD-jube-master-SDV.modules.test.xml
```

will use MAXW-DGTDParameter-head.

2) Several values for the parameters

```
<parameter name="k" type="int" >1,3</parameter>
[...]
<parameter name="iopath" type="string" >"/nvme/tmp/manzano/MAXW-DGTD-test/","/sdv-work/manzano/MAXW-DGTD-test/"</parameter
```

You can specify comma separated values for the parameters. In this case JUBE will execute the benchmark 4 times with the following values:

```
k=1, iopath=/nvme/tmp/manzano/MAXW-DGTD-test/
k=1, iopath=/sdv-work/manzano/MAXW-DGTD-test/
k=3, iopath=/nvme/tmp/manzano/MAXW-DGTD-test/
k=3, iopath=/sdv-work/manzano/MAXW-DGTD-test/
```

3) Substitute set

See for instance the following section in platform.xml

```
    <substituteset name="executesub">
       <iofile in="${submit_script}.in" out="$submit_script" />
       <sub source="#ENV#" dest="$env" />
       <sub source="#NOTIFY_EMAIL#" dest="" />
       <sub source="#BENCHNAME#" dest="$jube_benchmark_name" />
       <sub source="#NODELIST#" dest="nodes=$nodes:$nodetype:ppn=$taskspernode" />
       <sub source="#QUEUE#" dest="$queue" />
       <sub source="#JOBPOLICY#" dest="NACCESSPOLICY:SINGLEJOB" />
       <sub source="#TIME_LIMIT#" dest="$timelimit" />
       <sub source="#PREPROCESS#" dest="" />
       <sub source="#POSTPROCESS#" dest="" />
       <sub source="#STARTER#" dest="$starter" />
       <sub source="#ARGS_STARTER#" dest="$args_starter" />
       <sub source="#MEASUREMENT#" dest="" />
       <sub source="#STDOUTLOGFILE#" dest="$outlogfile" />
       <sub source="#STDERRLOGFILE#" dest="$errlogfile" />
       <sub source="#EXECUTABLE#" dest="" />
       <sub source="#ARGS_EXECUTABLE#" dest="" />
       <sub source="#FLAG#" dest="touch $done_file" />
    </substituteset>
```

And in MAXW-DGTD-jube-master-SDV.modules.test.xml:

```
    <substituteset name="executesub" init_with="platform.xml">
     <sub source="#NOTIFY_EMAIL#" dest="c.manzano@fz-juelich.de" />
     <sub source="#PREPROCESS#" dest="date +'(start) %F %T (%s)'; module purge; module load $modules; export LD_LIBRARY_PA
     <sub source="#POSTPROCESS#" dest="echo $LD_LIBRARY_PATH; echo $SCR_USER_NAME; echo $SCR_JOB_NAME; echo $SCR_JOB_ID; e
     <sub source="#EXECUTABLE#" dest="gdL$k" />
     <sub source="#ARGS_EXECUTABLE#" dest="" />
    </substituteset>
```

This will tell JUBE to take the file ${submit_script}.in and create an output file named ${submit_script} with the occurrences #NOTIFY_EMAIL#, #QUEUE#, etc. substituted by the corresponding values.

4) Step execute - chainjob_script

NOTE: For integrating an application the step prepare is not necessary and won't be covered in this guide. A slightly modificated version of the xml file is explained here.

```
    <step name="execute" shared="shared">
     <use>MAXW-DGTD-exec</use>
     <use from="MAXW-DGTD_specs.xml">MAXW-DGTDInputFile</use>
     <use>executeset</use>
     <use>executesub</use>
     <use from="platform.xml">chainsub</use>
     <use>systemParameter</use>
     <use tag="!woman">MAXW-DGTDParameter-head</use>
     <use tag="woman">MAXW-DGTDParameter-woman</use>
     <use>MAXW-DGTDInputSub</use>
     <use>MAXW-DGTDSCRSub</use>
```

```
     <use from="platform.xml">jobfiles</use>
     <use from="platform.xml">chainfiles</use>
     <do>mkdir exe; chgrp deeper exe; chmod g+s exe; cp -pr exe_$testcase/* exe/</do>
     <do>cp -p gdL$k exe/</do>
     <do>rm -f exe/Data_Num*; cp -p Data_Num.out exe/Data_Num</do>
     <do>rm -f exe/Data_IO*; cp -p Data_IO.out exe/Data_IO</do>
     <do>rm -f exe/scr.sh*; cp -p scr.sh.out exe/scr.sh</do>
     <do>rm -f exe/test.part*; cp -p test.part.gz exe/; gzip -d exe/test.part.gz</do>
     <do>$chainjob_script $shared_job_info $submit_script</do>
     <do done_file="$done_file"></do>
     <do>chgrp deeper job.log job.err</do>
   </step>
```

The interesting parts of this section are at the end. With

```
<do>$chainjob_script $shared_job_info $submit_script</do>
```

you can submit a series of jobs (in this example 4, see section 2). The first job will run and the next one will wait until the first one is finished. The ${chainjob_script} looks as follows:

```
cat /usr/local/jube2/platform/deep/deep-chainJobs.sh

#!/usr/bin/env bash

if [ $# -lt 2 ]
then
   echo "$0: ERROR (MISSING ARGUMENTS)"
   exit 1
fi

LOCKFILE=$1
shift
SUBMITSCRIPT=$*


if  [ -f $LOCKFILE ]
then
   DEPEND_JOBID=`head -1 $LOCKFILE`
   JOBID=`qsub -W depend=afterany:${DEPEND_JOBID} $SUBMITSCRIPT`
else
   JOBID=`qsub $SUBMITSCRIPT`
fi

echo ${JOBID} > $LOCKFILE

exit 0
```

In ${shared_jobinfo} the jobid of the preceding job is stored. Notice that

```
${shared_jobinfo} = ${shared_folder}/jobid = ./shared/jobid
```

and all jobs have in their sandbox a soft link called "shared" pointing to ./benchmarks_modules_test/benchmark_id/execute_shared.

5) Analyse

```
    <analyzer name="analyse" >
     <use from="MAXW-DGTD_specs.xml">pattern</use>
     <analyse step="execute">
       <file>exe/maxdg3DP$k.out</file>
       <file>job.log</file>
```

```
    </analyse>
  </analyzer>
```

After the job has run, you can analyse the results with:

```
jube analyse benchmarks_modules_test --id jube_id
```

The analyse step will look for the corresponding patterns in the output files, in this case exe/maxdg3DP$k.out and job.log, and the data will be stored inside the benchmark directory. The patterns to look for are specified in the MAXW-DGTD_specs.xml file:

```
  <patternset name="pattern">
   <pattern name="fluxpvector" unit="W" type="float">\s*Flux of the Poynting vector :\s+$jube_pat_wrd\s*</pattern>

   <pattern name="attau" unit="m" type="float">\s*AT TAU =\s+$jube_pat_wrd\s*</pattern>
   <pattern name="t" unit="nsec" type="float">\s*T =\s+$jube_pat_wrd\s*</pattern>
   <pattern name="iterations" unit="" type="int">\s*USING\s+$jube_pat_wrd\s*</pattern>
   <pattern name="maxcputime" unit="sec" type="float">\s*MAX CPU TIME    :\s+$jube_pat_wrd\s*</pattern>
   <pattern name="realtime" unit="sec" type="float">\s*REAL TIME       :\s+$jube_pat_wrd\s*</pattern>
   <pattern name="timetosolution" unit="sec" type="float">\s*TIME TO SOLUTION:\s+$jube_pat_wrd\s*</pattern>
   <pattern name="iotime" unit="sec" type="float">\s*I/O TIME        :\s+$jube_pat_wrd\s*</pattern>
   <pattern name="electrtime" unit="sec" type="float">\s*ELECTR TIME     :\s+$jube_pat_wrd\s*</pattern>
   <pattern name="omptime" unit="sec" type="float">\s*OMP TIME        :\s+$jube_pat_wrd\s*</pattern>
   <pattern name="tfluxelectr" unit="sec" type="float">\s*T FLUX ELECTR   :\s+$jube_pat_wrd\s*</pattern>
   <pattern name="tfluxmagnet" unit="sec" type="float">\s*T FLUX MAGNET   :\s+$jube_pat_wrd\s*</pattern>
   <pattern name="tupelectr" unit="sec" type="float">\s*T UP ELECTR     :\s+$jube_pat_wrd\s*</pattern>
   <pattern name="tupmagnet" unit="sec" type="float">\s*T UP MAGNET     :\s+$jube_pat_wrd\s*</pattern>
   <pattern name="tfourier" unit="sec" type="float">\s*T FOURIER       :\s+$jube_pat_wrd\s*</pattern>

   <pattern name="starttime" type="string" >\(start\)$jube_pat_bl($jube_pat_wrd$jube_pat_bl$jube_pat_wrd)</pattern>
   <pattern name="stoptime"  type="string" >^\s*\(finished\)\s*(.*)\s*\(</pattern>
 </patternset>
```

If there are multiple matches inside a single file you can add a reduce option. Normally only the first match will be extracted. See ?https://apps.fz-juelich.de/jsc/jube/jube2/docu/glossar.html#term-pattern-tag for more information.

6) Result

```
    <result>
     <use>analyse</use>
     <table name="result" style="pretty" sort="jube_wp_id">
       <column>modules</column>
       <column>iopath</column>
       <column>iotime</column>
       <column>starttime</column>
       <column>stoptime</column>
     </table>
   </result>
```

The last part is the result table creation. Here you have to use an existing analyser. The <column> contains a pattern or a parameter name. sort is the optional sorting order (separated by ,). The style attribute can be csv or pretty to get different ASCII representations. To create the result table you have to write:

```
jube result benchmarks_modules_test --id jube_id
```

The result table will be written to STDOUT and into a result.dat file inside ./benchmarks_modules_test/benchmark_id/result.