H2020-FETHPC-01-2016

DEEP-EST

DEEP Extreme Scale Technologies

Grant Agreement Number: 754304

D2.1

DEEP-EST benchmark suites

Final

Version: 1.0
Author(s): K. Kulkarni (JSC)
Contributor(s): C. Manzano (JSC), J. Corbalan (BSC), A. Jokanovic (BSC)
Date: 29.03.2018

Project and Deliverable Information Sheet

DEEP-EST Project	**Project ref. No.:**	754304
	Project Title:	DEEP Extreme Scale Technologies
	Project Web Site:	www.deep-projects.eu
	Deliverable ID:	D2.1
	Deliverable Nature:	Report
	Deliverable Level: PU*	**Contractual Date of Delivery:** 31/03/2018
		Actual Date of Delivery: 29/03/2018
	EC Project Officer:	Juan Pelegrin

*– The dissemination level are indicated as follows: **PU** - Public, **PP** - Restricted to other participants (including the Comissions Services), **RE** - Restricted to a group specified by the consortium (including the Comission Services), **CO** - Confidential, only for members of the consortium (including the Comission Services).

Document Control Sheet

Document	**Title:** DEEP-EST benchmark suites	
	ID: D2.1	
	Version: 1.0	**Status:** Final
	Available at: www.deep-projects.eu	
	Software Tool: LaTeX	
	File(s): DEEP-EST_D2.1_DEEP-EST_benchmark_suites_v1.0.pdf	
Authorship	**Written by:**	K. Kulkarni (JSC)
	Contributors:	C. Manzano (JSC), J. Corbalan (BSC), A. Jokanovic (BSC)
	Reviewed by:	M. Cintra (Intel) J. Kreutz (JSC)
	Approved by:	BoP/PMT

Document Status Sheet

1.0	28/03/2018	Final	

Document Keywords

Keywords:	DEEP-EST, HPC, Exascale

Contents

List of Figures

List of Tables

Executive Summary

This deliverable report describes set of synthetic and application benchmarks to be used for continuous benchmarking activities described for the task 2.1. These benchmarks will run periodically to evaluate system and application performance on evaluators and final prototype DEEP-EST system. The selected benchmarks are integrated in Juelich Benchmarking Environment (JUBE) for automated execution and analysis. Along with the description of synthetic and application benchmarks, example analysis of results from synthetic benchmarks is presented. For detailed description of application benchmark cases please also refer to the deliverable report D1.2 from WP1.

As well as evaluating the benefits in terms of performance for individual applications, the DEEP-EST modular system is intended to provide a global view of the system where the different modules will be managed as a whole. The job scheduler functionalities and policies developed in the context of WP5 will be evaluated in WP2 through simulations. In this deliverable, we have proposed a first version for what we have called Modular Workload Format (MWF). The MWF has been proposed based on the already existing Standard Workload Format, widely used in job scheduling evaluation, with additional extensions to fit into the DEEP-EST project requirements. We have also taken into account special characteristics existing in the SLURM Workload Manager when submitting jobs since it is the job scheduler to be used in the project. For a better understanding of the format, we have included a synthetic example using the DEEP-EST applications as reference.

1. Introduction

Within WP2 the DEEP-EST prototype will be evaluated and the benefits to application developers and infrastructure operators will be measured. Examples of measurable benefits include time-to-solution, utilization of resources, energy, energy efficiency, system throughput and job waiting time. In order to do so, Task 2.1 will provide two benchmark suites comprised of application and synthetic benchmarks and Task 2.2 will define a set of workloads representative of how a system based on a Modular Supercomputing Architecture (MSA) will be used.

This deliverable describes the initial set of benchmarks and workloads to be used by Tasks 2.1 and 2.2 and presents some first results measured in currently available evaluation platform DEEP-ER SDV and also describes the future work.

The document is structured in 7 main sections: Section 1 comprises the present introduction. Section 2 focuses on the evaluation platforms. Section 3 introduces the JUBE Benchmarking Environment and describes the benchmarking strategy which will be followed in order to fulfill WP2's objectives. Section 4 explains the selected synthetic benchmarks and defines a series of initial test cases. Section 5 presents the set of applications chosen for benchmarking, gives a general outlook of each and describes possible test cases. Section 6 focuses on workloads, its definition and application for the evaluation of the DEEP-EST prototype. Finally, Section 7 provides a review of the work done and gives a description of the future steps.

The work presented in this document will enable WP2 to perform the benchmarking measurements and evaluate the DEEP-EST prototype. Additionally, the workloads defined will help assessing the benefits of co-allocation and parametrized scheduling policies. Application developers and infrastructure operators will benefit from the description done here, since having a better understanding of the performance indicators will guide them to make use of and operate the MSA-system more efficiently.

2. Evaluation platforms

WP3 will define the system-level architecture and the high level specifications of the DEEP-EST prototype. To administer the overall progress, it will introduce intermediate evaluation points in the form of evaluators and software development vehicles, which will in turn be used by other WPs.

WP2's Task 2.1 will define a set of benchmarks and will use them to obtain baseline measurements with existing large systems, and will obtain also additional benchmark results in the aforementioned evaluators and SDVs procured by WP3. As soon as the DEEP-EST system becomes available the testing will move to the prototype.

At the moment of writing this deliverable, the initial benchmarking set has already run in the DEEP-ER SDV. This section gives an overview of this evaluation platform.

2.1. DEEP-ER SDV

The DEEP-ER project investigated innovative concepts to provide fast and highly scalable parallel I/O and increased resilience functionality to a Cluster-Booster system based on new storage technologies. The Booster part was designed to achieve highest energy efficiency and density. To enable system and application software developers to port and implement their codes before the DEEP-ER Booster would became available a software development vehicle (SDV) using off-the-shelf technology was configured, installed and made available to the project. [1]

The DEEP-ER SDV can be seen as a MSA-precursor providing a cluster and a booster module. Though in a different form factor, the DEEP-EST prototype will also use an EXTOLL network compatible with the one in the SDV. It will be available for the duration of project as long as there is need of it and used for pre-evaluating the applications and developing the benchmarks which will ported to the DEEP-EST prototype.

2.1.1. System architecture of the DEEP-ER SDV

The DEEP-ER SDV consists of 16 dual-processor Intel® Xeon® nodes complemented by 8 nodes equipped with the second generation of Intel® Xeon Phi™ processors, a local storage system, a network attached memory and an EXTOLL based interconnect. See Table 1 for a detailed description of the hardware.

Currently, the system is available via three login nodes (two physical ones and one virtual machine). Jobs can be submitted via the batch system SLURM. The home filesystem on the SDV is provided via GPFS/NFS. The local storage system of SDV is running BeeGFS and is available at /sdv-work. Each node also provides a non-volatile memory device which can be used directly or through an additional BeeGFS file system which is running on demand

[1] See DEEP-ER Deliverable 3.5 "Prototype Installation" [1]

8 KNL nodes	1 x Intel® Xeon Phi 7210 (96 GB DDR4, 16 GB MCDRAM) on Intel® Server Board S7200AP (Adams Pass) pre-release boards
16 Cluster nodes	2 x Intel® Xeon CPU E5-2680 v3 (128 GB DDR4) Haswell generation
Non-volatile memory	Per node: 1 x Intel® DC P3700 SSD 400 GB
High-speed interconnect	EXTOLL TOURMALET NICs
1 Metadata Server	2 x Intel® Xeon CPU ES-2609 v3 (32 GB DDR4) + 2 x SSD 200 GB
2 Storage Servers	2 x Intel® Xeon CPU ES-2609 v3 (32 GB DDR4)
RAID System	4 x 8 Gbit FC connector and 24 x 6 TB SAS Nearline
Network attached memory	Per rack: 2 x EXTOLL FPGA + Hybrid Memory Cube (HMC) 2 GB

Table 1.: Hardware details for the DEEP-ER SDV

independently in each of these devices. [2] A complete overview of the file systems is provided in Table 2.

Mount point	User access	Type
/home[a-b]	/home[a-b]/$USER	GPFS exported via NFS
/gpfs-work	/gpfs-work/$USER	GPFS exported via NFS
/sdv-work	/sdv-work/$USER	BeeGFS
/nvme	/nvme/tmp	NVMe device
/mnt/beeond	/mnt/beeond	BeeGFS On Demand running on the NVMe

Table 2.: File systems available in the DEEP-ER SDV

For the tests, both cluster and booster modules will be used, as well as the local BeeGFS file system and potentially the NVMe devices.

[2] See DEEP-ER Deliverables 4.4 "I/O software packages" and 4.5 "Results of I/O performance measurements" [1]

3. JUBE

Benchmarking activities should be automated to guarantee fair and reproducible comparisons between different platforms or environments. Automated benchmarking also simplifies managing of different combinations of parameters in a large parameter space and reduces risk of introducing errors in process. The JUBE package enables a systematic benchmarking and allows adapting custom workflows to new architectures. In DEEP-EST, JUBE will be used to automatically test and analyze the system to ensure that it covers all High Performance Computing (HPC) and High Performance Data Analytics (HPDA) areas important to Europe and to enable detailed measurement of all important performance aspects of the prototype.

3.1. Introduction

The JUBE package provides a script based framework to create benchmark sets, run these sets on different computer systems and evaluate the results. It is actively developed by the Juelich Supercomputing Centre of Forschungszentrum Juelich, Germany.

3.2. Description

The input file-format for JUBE uses the well-spread markup language XML. JUBE offers schema-validation files, which help preventing syntactic errors already in editors that support schema validation before an input file is actually parsed by JUBE. The structure is designed to reduce the amount of text duplication while retaining enough verbosity to debug possible problems. See the following examples for better understanding of structure and its different parts. The complete code listings and additional JUBE files can be found in the Appendix JUBE XML files.

3.2.1. Hello World Example 1

JUBE uses the `jube` tag as its root. Inside it contains a `benchmark` element which includes the actual benchmark description:

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>

  <benchmark name="hello_world" outpath="bench_run">
    <comment>A simple hello world</comment>
  ...
```

One of the key functionalities of JUBE is separation of data and commands in a way that allows commands with different input data to be executed in a similar manner. This concept is most obviously reflected in the `parameterset` elements inside the `benchmark` tag. By adding multiple choices to a `parameter` inside a `parameterset`, JUBE will automatically use each possible combination. The actual execution is described in the `step` element which is also an integral part of a `benchmark`. It defines a single block of execution. A possible example is shown in the next code listing. JUBE will execute the command in the `<do>` line with different values of the `parameter` ”text” (e.g. ”Hello” and ”World”) and ”number” (e.g. ”1”, ”2” and ”4”). JUBE would then execute this benchmark for all combinations of ”text” and ”number” which are, for this example, the six combinations (”Hello”, ”1”), (”World”, ”1”), (”Hello”, ”2”), (”World”, ”2”), (”Hello”, ”4”) and (”World”, ”4”):

```
...
<!-- Benchmark configuration -->
<parameterset name="hello_parameter">
  <!-- Create a parameterspace out of two template parameters -->
  <parameter name="text">Hello,World</parameter>
  <parameter name="number">1,2,4</parameter>
</parameterset>

<!-- Operation -->
<step name="say_hello">
  <use>hello_parameter</use> <!-- use parameterset "hello_parameter" -->
  <do>echo This is ‘hostname‘. $text $number</do> <!-- shell command -->
</step>
...
```

As a final task, the executed benchmarks should be analyzed. This process is split into an `analyzer` part and a `result` part. An example is shown in the following code listing:

```
...
<!-- Regex pattern -->
<patternset name="pattern">
  <pattern name="node" type="string">This is (.*)\.\s+</pattern>
</patternset>

<!-- Analyse -->
<analyzer name="analyse" >
  <use>pattern</use> <!-- use existing patternset -->
  <analyse step="say_hello">
    <file>stdout</file> <!-- file which should be scanned -->
  </analyse>
```

```
    </analyzer>

    <!-- Result -->
    <result>
      <use>analyse</use> > <!-- use existing analyser -->
      <table name="result" style="pretty" sort="node">
        <column>node</column>
        <column>text</column>
        <column>number</column>
      </table>
    </result>

  </benchmark>

</jube>
```

The `analyzer` element describes data to be extracted and how to extract it, which in this case uses the `patternset` `pattern` on standard output. Using this data `result` element defines how the extracted data should be shown to the user. For the above code listing this means printing a human-readable ASCII formatted table with columns containing the node which executed task as first column and the ”text” and ”number” as second and third ones, respectively.

3.2.2. Hello World Example 2

In this example, a job is created which will run on 2 nodes in the DEEP-ER SDV.

In order to send a job to batch system there are a series of additional files which can be used for creating the corresponding submit scripts. In following code listing we instruct JUBE to make use of the files in /usr/local/jube2/platform/deep which include platform specific parameters:

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>

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

  <benchmark name="hello_world" outpath="bench_run">
    <comment>A simple hello world</comment>
...
```

We include job configuration parameters, which will be initialized with the default values found in platform.xml. Platform directory also contains a submit job script template which will be used to create the corresponding submit script. The `say_hello` step will send it to batch system:

```
...
    <!-- Job configuration -->
    <parameterset name="systemParameter" init_with="platform.xml">
      <parameter name="nodes">2</parameter>
      <parameter name="taskspernode">1</parameter>
      <parameter name="queue">sdv</parameter>
    </parameterset>

    <!-- Substitute jobfile -->
    <substituteset name="executesub" init_with="platform.xml">
      <sub source="#NOTIFY_EMAIL#" dest="c.manzano@fz-juelich.de" />
      <sub source="#EXECUTABLE#" dest="echo" />
      <sub source="#ARGS_EXECUTABLE#" dest="This is ‘hostname‘. $text $number" />
    </substituteset>

    <!-- Operation -->
    <step name="say_hello">
      <use>hello_parameter</use>
      <use>systemParameter</use>
      <use>executesub</use>
      <use from="platform.xml">executeset</use>
      <use from="platform.xml">jobfiles</use>
      <do>$submit $submit_script</do>
    </step>
...
```

3.2.3. Hello World Example 3

JUBE is designed to be non-blocking for operations such as sending jobs to a queuing system. This is often preferable since waiting for batch jobs to finish can be quite time consuming and should not interrupt the usual workflow. However, this behavior can be undesirable for avoiding the usage of shared resources e.g. file system or network links.

In previous example 6, jobs are sent in parallel to the batch system. In cases where benchmarks should be prevented from parallel execution, chain jobs with dependencies can be used. See the following code listing:

```
...
    <!-- Operation -->
    <step name="say_hello" shared="shared">
      <use>hello_parameter</use>
      <use>systemParameter</use>
      <use>executesub</use>
      <use from="platform.xml">executeset</use>
      <use from="platform.xml">jobfiles</use>
      <use from="platform.xml">chainfiles</use>
      <do>$chainjob_script $shared_job_info $submit_script</do>
    </step>
...
```

Here we again make use of the platform specific files under /usr/local/jube2/platform/deep.

3.3. Job output

The command ”jube run” is used for running benchmarks. For case of examples in previous section,

```
$ jube run hello_world_1.xml
```

will produce the following output:

```
$ jube run hello_world.xml
################################################################################
# benchmark: hello_world
#
# A simple hello world
################################################################################

Running workpackages (#=done, 0=wait, E=error):
############################################################ (  6/  6)

   stepname | all | open | wait | error | done
  ----------+-----+------+------+-------+-----
```

```
  say_hello |   6 |    0 |    0 |     0 |    6

>>>> Benchmark information and further useful commands:
>>>>       id: 0
>>>>   handle: bench_run
>>>>      dir: bench_run/000003
>>>>  analyse: jube analyse bench_run --id 0
>>>>   result: jube result bench_run --id 0
>>>>     info: jube info bench_run --id 0
>>>>      log: jube log bench_run --id 0
################################################################################
```

The id (in addition to the benchmark directory handle) is an important number. Every benchmark run will get a new unique id inside the benchmark directory. The benchmark directory has following structure:

```
bench_run               # the given outpath
|
+- 000000               # the benchmark id
   |
   +- configuration.xml # the stored benchmark configuration
   +- workpackages.xml  # workpackage information
   +- run.log           # log information
   +- 000000_say_hello  # 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
```

This overview explains some fundamental features of JUBE which are important for this deliverable. These should give the reader a basic idea of the design principles and general usage of JUBE. A complete and detailed documentation including tutorials for getting started, a description of the more advanced features and a reference describing all available functionality can be found in the online documentation [3]. Please refer to Lührs et. al. for an in-depth description of JUBE's workflow [2].

3.4. Benchmarking strategy

JUBE will be used to perform continuous benchmarks in special reservations. The frequencies will most likely be daily or weekly, depending on the duration of the benchmark and need for changes in codes. Several kinds of benchmarking will be performed. Mostly consistent benchmarks will be used to measure changes in the system performance, for example when driver changes are applied or a hardware is substituted. Application benchmarks will be used to measure the change in application performance over time, for example due to integration of new strategies or modifications in the code. Unless there is a good reason not to (for instance when the compilation time is considerable), the code of the benchmarks will be freshly compiled with default modules[1] before sending the jobs to the batch system.

As shown in Figure 1, a cronjob triggers series of benchmark suites which will send several dependent jobs to the system. A trigger process signals the completion of current suite and launches the next. In some cases, the jobs will run sequentially[2] as shown in the figure, for avoiding contamination of the measured results. In other special cases, several jobs can be sent in parallel to run in the different partitions of the MSA-system. This split-up strategy can help in evaluating the scheduler of modular systems showing the performance gain when several applications requesting various parts of modular environment. The results will be stored,

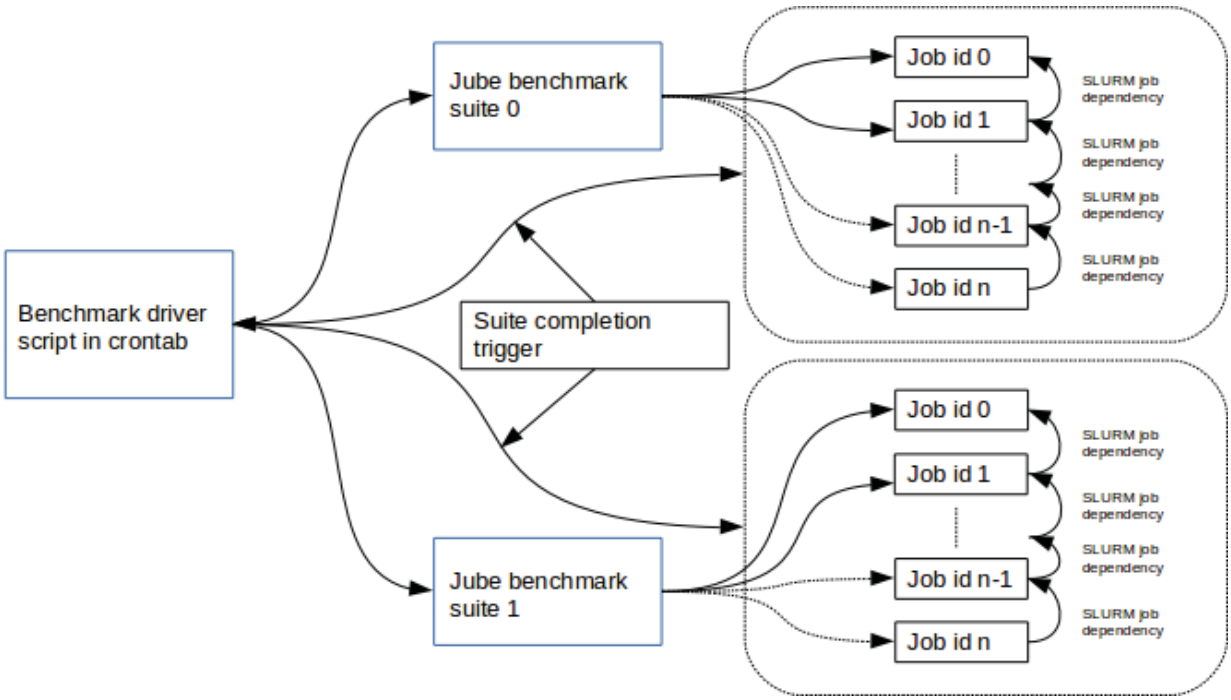


Figure 1.: Benchmarking strategy

archived and made accessible to developers and system operators through one of the file systems of the prototype (currently available under /usr/local/deep/deep-est_benchmarks/). A

[1] Default modules typically point to latest versions of software stack
[2] Benchmark suit 0 is launched first with its dependent job and benchmark suit 1 executes after completion of suit 0

GitLab repository is in place for collecting the different contributions to the benchmark suites. [4]

In cases where the applications in WP1 do not cover a typical workflow or use case in HPC or HPDA, 3rd-party applications will be selected and used for the benchmarking. For instance, after consultation with data analytics experts, it was clear that the TensorFlow application, which was not initially included in the Consortium Amendment, has to be added to the application portfolio. See Section 5.5.3. for more details about the application.

Additionally, WP3 selected and run a series of microbenchmarks for design of the DEEP-EST prototype. The main goal was to evaluate a set of technologies, in particular for the Extreme Scale Booster (ESB). To reach a data-driven decision, it was needed to collect performance data from benchmarks close to the requirements of the workloads of WP1's applications. For many of the above mentioned technologies, access to multi-node systems with a software stack which would allow the applications to run was not possible and therefore the need of low level or micro benchmarks. WP3 will document the work that has been done in the upcoming Deliverable 3.2 High level system design.

4. Synthetic Benchmarks

Synthetic benchmarks offer wide variety of options for stress testing the system for important parameters such as network latency, bandwidth and I/O performance. For this purpose, MPI LinkTest (for network bandwidth), Interleaved Or Randomized(IOR) and h5perf (for I/O performance) are chosen to cover the three above mentioned system parameters. Following sections briefly describe various parameters and results of these benchmark suites. The chosen benchmark cases are integrated in JUBE and execute serially to avoid cross contamination of measurements. As the newer components on DEEP-EST system are available, relevant benchmarks will be identified and integrated in the central suites.

4.1. MPI LinkTest

The MPI LinkTest program is a parallel ping pong test between all possible MPI connections of a machine. The output of this program is a full communication matrix which shows the bandwidth and message latency between each processor-pair, together with a report including the minimum bandwidth. The program can also be used for communication stress tests by running it repeatedly for a specific duration. The MPI LinkTest software has been developed by the Juelich Supercomputing Centre and is freely available[5].

The LinkTest runs for n processors in n steps, where in each step n/2 pairs of processors perform the MPI ping pong test. The Ping pong tests run parallely in each step by default which can be serialized. Assignment of MPI tasks is performed once at the beginning of the program, according to the underlying hardware and operating system (e.g. on Linux the hostname and rank/core are used for identification). The selection of the pairs is random but it is guaranteed that, after running all steps, all possible pairs have been covered. A top N analysis of the results is done and the poorest N connections are identified, where N can be specified by the user. SIONlib is used for writing an output file containing the results of the whole communication matrix[6][7]. An analysis tool included with the software can be used to generate pattern files, list of bad links and graphical output illustrating the communication matrix and providing a histogram about timings and bandwidths ranges. Following are the available options.

```
Parameters description:
[-a 0|1]           do alltoall mode                        (0 or 1)  (0)
[-i  <iterations>] number of pingpong iterations           (3)
[-s  <size>]       message size in Bytes                   (131072)
[-k  <size>]       message size in KBytes                  (128k)
[-M 0|1]           randomized processor numbers            (0)
[-S 0|1]           run in a serialized mode                (0)
[-W 0|1]           write result file (sion)                (1)
[-T  <min>]        run  <min> minutes, repeating the test  (-1, off)
[-C  <num>]        number of collected tasks for output    (32)
```

The LinkTest program can be launched as follows.

```
srun -n  <#tasks> ./mpilinktest <options>
```

Upon successful completion of execution, the results can be processed with an analysis program provided with the package as follows.

```
pingponganalysis  <options> insionfn
```

Following options are available for the analysis program.

```
Parameters description:
[-a]    generate accesspattern file     (PPM)
[-A]    generate accesspattern file     (ASCII)
[-B]    generate badlink list           (ASCII)
[-d]    generate distancepattern file   (PPM)
[-b]    generate bandwidthpattern file  (PPM)
[-l]    <minbw> min. bandwidth, conection below will be reported
        (def. 1 MB/s)
[-L]    <maxbw> max. bandwidth, conection above will be reported
        (def. 10000 MB/s)
[-g]    generate gnuplot 2d input file
[-p]    generate postscript report
[-v]    verbose mode
```

The analysis program generates statistical distribution of bandwidth and duration data as well as graphical communication checkerboard pattern and frequency distribution in postscript format. A sample graphical output is shown in Figure 2. Intra-node and inter-node communication patterns and bandwidths as denoted in Figure 2 often reveal underlying problems in the network e.g degradation of performance of a certain communication link or switch.

With LinkTest, network latency, peer to peer(P2P) and congestion bandwidth tests are implemented. P2P and congestion bandwidth tests are grouped in small and large cases with message sizes ranging from 8 KBytes to 4 MBytes. Selected 2 MPI processes per node are placed on different sockets to test intra-node latency and bandwidth. Table 3 lists the details of parameters used for each case. Appendix A.7,A.8 and A.9 list the JUBE xml files used for this benchmark. Figures 3 shows the results of typical measurements taken on the cluster compo-

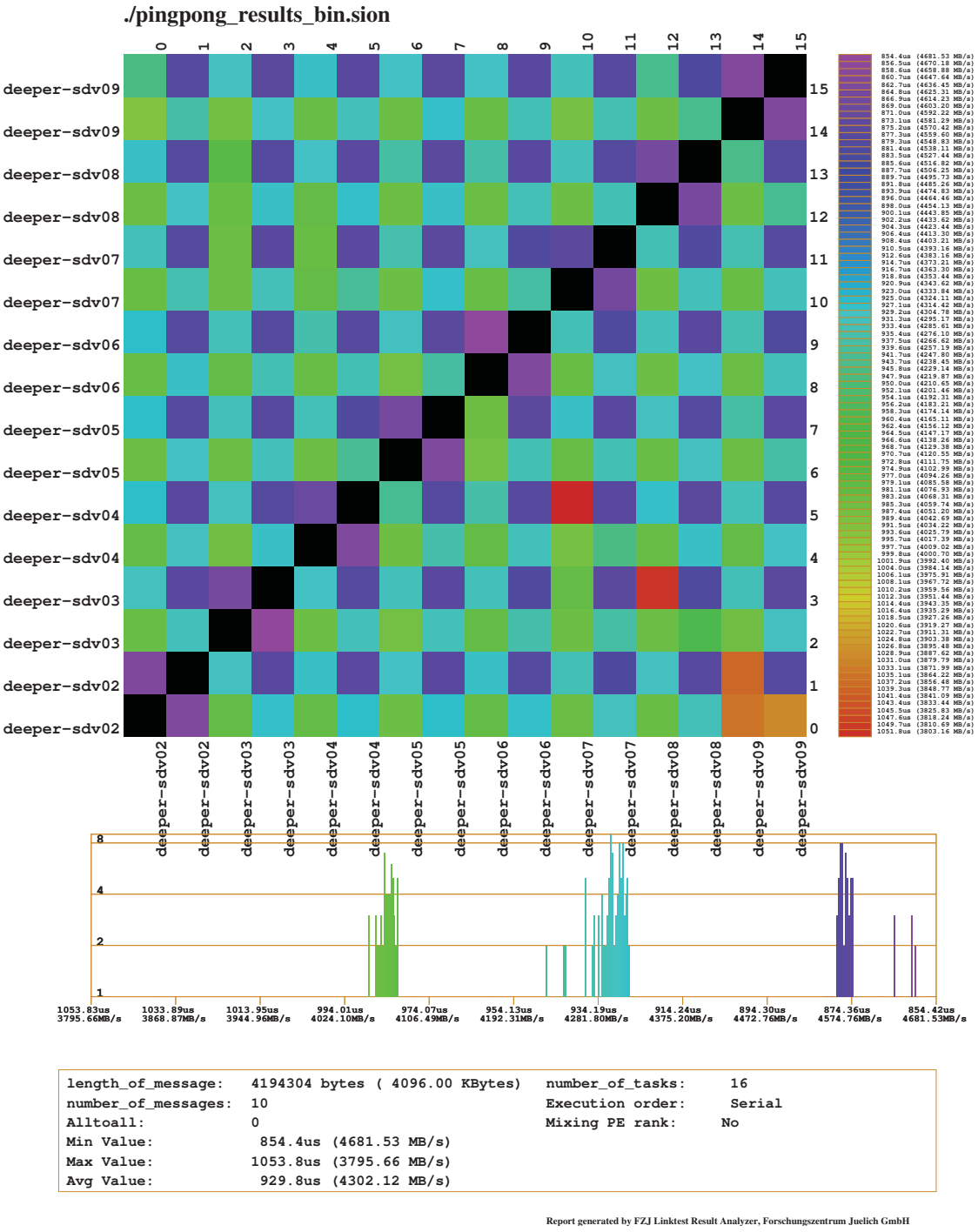


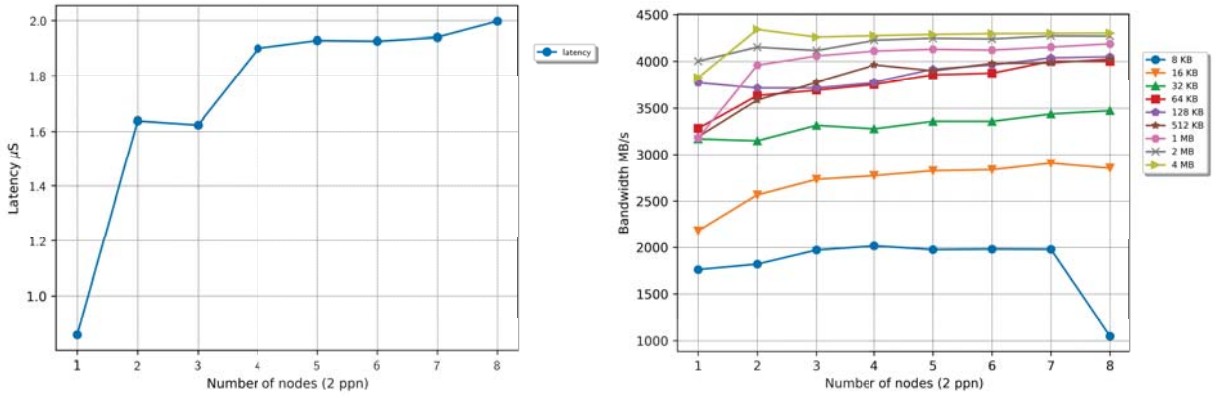
Figure 2.: Pingpong analysis communication pattern and distribution

nent of DEEP-EST SDV. Figure 3a shows average latency of intra-node and inter-node communication obtained by sending 1 Byte of data. Correspondingly by sending various lengths of messages in serialized mode, bandwidth measurements (Table 3 Test case: bandwidth) are taken as shown in Figure 3b. These values are obtained by parsing the standard log output of

Test Case	MPI ppn	Message size	Iterations	Serial Mode	All to all
Latency	2	1 B	10	True	False
Bandwidth	2	8 KB to 4 MB	10	True	False
Congestion	2	8 KB to 4 MB	10	False	False

Table 3.: Parameter details of MPI LinkTest cases

MPI LinkTest program through JUBE's inbuilt analysis and result steps.



(a) Average latency (μs)

(b) Average bandwidth (MB/s)

Figure 3.: MPI LinkTest number of nodes vs latency(a) and bandwidth(b)

4.2. Interleaved Or Randomized (IOR)

The Interleaved Or Randomized (IOR) benchmark developed at Lawrence Livermore National Labs, USA, is implemented to assess the performance of a parallel file system [8]. IOR provides options to test through many middleware libraries including MPI, HDF5 and NetCDF along with standard POSIX interface. For benchmarks in the scope of DEEP-EST, POSIX and MPIIO APIs are used. Following are few of the relevant options for this benchmark.

```
-a S  api           --  API for I/O [POSIX|MPIIO|HDF5|NCMPI]
-b N  blockSize     -- contiguous bytes to write per task (e.g.: 8, 4k, 2m, 1g)
-B    useO_DIRECT   -- uses O_DIRECT for POSIX, bypassing I/O buffers
-c    collective    -- collective I/O
-e    fsync         -- perform fsync upon POSIX write close
-F    filePerProc   -- file-per-process
```

```
-H    showHints     -- show hints
-i N  repetitions   -- number of repetitions of test
-o S  testFile      -- full name for test
-t N  transferSize  -- size of transfer in bytes (e.g.: 8, 4k, 2m, 1g)
-U S  hintsFileName -- full name for hints file
-v    verbose       -- output information (repeating flag increases level)
-V    useFileView   -- use MPI_File_set_view
```

Following options are valid only for POSIX interface.

```
useO_DIRECT       - use O_DIRECT for POSIX, bypassing I/O buffers [0]
singleXferAttempt - will not continue to retry transfer entire buffer
                    until it is transferred [0]
fsync             - perform fsync after POSIX write close [0]
```

Operating systems cache the data to be written to the file. With `fsync` call, it is ensured that the data is written to the physical storage medium or reaches file system controller devices.

Following options are valid only for MPIIO interface.

```
preallocate            - preallocate the entire file before writing [0]
useFileView            - use an MPI datatype for setting the file view option
                         to use individual file pointer [0]
                         NOTE: default IOR uses explicit file pointers
useSharedFilePointer   - use a shared file pointer [0] (not working)
                         NOTE: default IOR uses explicit file pointers
useStridedDatatype     - create a datatype (max=2GB) for strided access;
                         akin to MULTIBLOCK_REGION_SIZE [0] (not working)
collective             - uses collective operations for access [0]
showHints              - show hint/value pairs attached to open file [0]
                         NOTE: not available in NCMPI
```

For MPIIO usage in independent mode, each process operates independently from the rest of the processes. The resulting I/O performance will depend mostly on the access pattern. Independent mode would cause each process to perform small I/O operations every time the memory buffer is transferred to disk. On the other hand, collective I/O enables different processes to coordinate their transfer requests to improve I/O performance. A common strategy is to aggregate many small requests of different processes that may be non contiguous into fewer

larger requests to minimize latency; resulting in fewer I/O operations. Both the combinations are implemented in varying block sizes (Table 4).
IOR application can be launched as follows:

```
srun ./ior -f ior_input.cfg
```

where `ior_input.cfg` is a file containing various options list.

Table 4 lists the options used for this benchmark.[1] Appendix A.10 and A.11 list the JUBE xml files used for this benchmark. Figures 4a and 4b show the typical result from IOR measure-

API / options	POSIX	MPIIO
filePerProc	0	0
blockSize	8m, 16MB, 32MB, 64MB, 128MB, 256MB, 512MB, 1GB	8m, 16MB, 32MB, 64MB, 64m, 128m, 256m, 512m, 1GB
transferSize	2MB, 4MB, 8MB	2MB, 4MB, 8MB
fsync[2]	True or False	–
collective or independent[3]	–	True or False
iterations	5	5

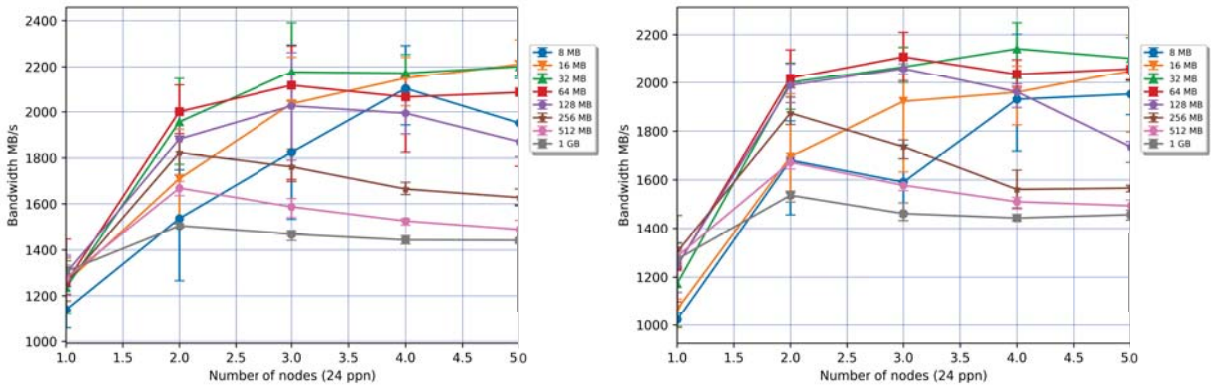
Table 4.: Parameter details of IOR cases

ments. These experiments are performed on the cluster component of DEEP-EST SDV, where BeeGFS is the target file system mounted at `/sdv-work`. Figure 4a shows that higher bandwidths can be achieved without `fsync` as opposed to synchronization calls after every write operation(not shown here). For MPIIO 4b, advantages of collective I/O are evident on larger scales while on smaller scales, performance gains are negligible to nonexistent.

4.3. h5perf

HDF5(Hierarchichal Data Format) offers self-contained platform independent file format with parallel I/O support, designed to store and organize large amounts of data. Few applications in DEEP-EST already use or intend to use this file fromat. h5perf is a utility application provided with HDF5 library which can be used to test I/O performance similar to IOR [9]. It offers wide variety of parameters to test for POSIX, MPIIO and parallel HDF5 api. The application can perform 1D or 2D tests on buffers and datasets. Following are some of the important options. For this benchmark case, parallel HDF5 api is chosen.

[1] For IOR, block size must be multiple of transfer size.
[2] Valid only for POSIX
[3] Valid only for MPIIO



(a) POSIX write no fsync

(b) MPIIO write collective

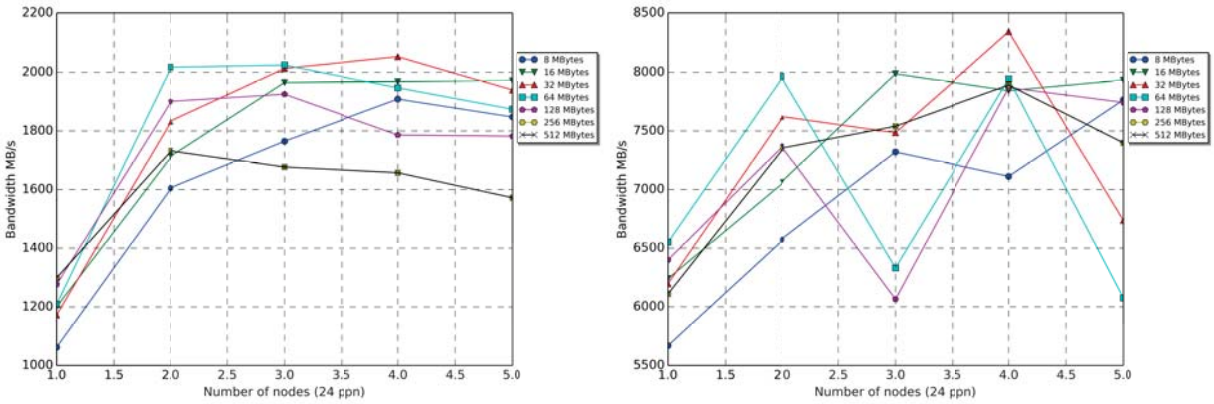
Figure 4.: IOR number of nodes vs write bandwidth(MB/s) (a) POSIX and (b) MPIIO

```
OPTIONS
-h,    --help                Print a usage message and exit
-A AL, --api=AL              Which APIs to test [default: all of them]
-B S,  --block-size=S        Block size within transfer buffer
-c,    --chunk               Create HDF5 datasets using chunked storage
                             [default: contiguous storage]
-C,    --collective          Use collective I/O for MPI and HDF5 APIs
                             [default: independent I/O)
-e S,  --num-bytes=S         Number of bytes per process per dataset
-g,    --geometry            Use 2D geometry [default: 1D geometry]
-i N,  --num-iterations=N    Number of iterations to perform [default: 1]
```

h5perf provides two types of storage layouts: contiguous (default) and chunked. Contiguous layout defines a single contiguous region of storage for the dataset. In contrast, chunked layout allocates several smaller regions of similar dimensions called chunks. Although HDF5 allows the user to define arbitrary dimensions for the chunks, h5perf uses the same dimensions of the blocks for the chunks, i.e. each block is stored in a separate chunk. Since accessing data on a chunk requires a single I/O operation, the use of chunked layout can improve performance for certain patterns.
Table 5 lists the details of parameters used for this case. Remaining parameters are kept at their default value. Owing to large number of bytes per processor, default 1D geometry is used. Appendix A.12 lists the JUBE xml files for h5perf. Figures 5a and 5b show the typical result from h5perf measurements.

Parameter	Value
- -api	phdf5
- -num-bytes	8m, 16MB, 32MB, 64MB, 128MB, 256MB, 512MB, 1GB
- -num-iterations	10
- -collective	True or False
- -chunk	True or False

Table 5.: Parameter details of h5perf benchmark



(a) HDF5 write (b) HDF5 read

Figure 5.: HDF5 number of nodes vs (a) write bandwidth (MB/s) and (b) read bandwidth(MB/s) case: collective, contiguous(non chunked)

5. Application Benchmarks

Within WP2.1, application and benchmark cases delivered along with D1.2 are being implemented in continuous benchmarking fashion. This will facilitate application and system developers to monitor their respective activities with reference to developments from source code and system side. This chapter briefly describes the involved applications. The chosen applications form a wide array for scientific fields, which would demonstrate the general purpose usage of DEEP-EST system. A subset of applications and their benchmark cases have already been integrated in JUBE framework and remainder of applications will be subsequently included. For detailed description of the applications, please refer to deliverable report D1.2.

5.1. Neuroscience (Task leader: NMBU)

NEST, Arbor and Elephant represent the applications from neuroscience which are coupled together with MUSIC library. NEST models brain tissue as an abstract collection of neurons and connections. The state of each neuron changes according to a set of ordinary differential equations and delayed data pulses represent interaction between nodes. The interaction strength of neurons can be statically or dynamically modeled and depends on activity of two connections joined by neurons. Build and simulation phases represent the two distinct phases of NEST. Build phase which constructs data structures representing nodal connections can dominate the overall simulation time. Simulation phase updates the individual neuron state and spikes are emitted based on a threshold criterion.
Local Field Potentials(LFP) are a measure of activity of neuronal population. Within DEEP-EST framework, NEST will stream point-neuron simulations into compartmental neuron simulations with Arbor simulation package. The point-neuronal spikes will be communicated to Arbor package via MPI-based MUSIC library.
Functioning and dynamics of large neuronal networks will be analyzed with the Elephant package where NEST will transmit the spike trains to this package coupled via MUSIC library.
From a user perspective, the most essential metric is the application turnaround time, since work in neuronal network simulation entails a high proportion of exploratory simulations. It is assumed that overall runtimes for all use cases will be determined by the speed of the NEST simulation. The runtime for a NEST simulation has two distinct components: the network construction time and the network simulation time. For short exploratory simulations, the network construction time can be significant, for long running simulations such as long multi-area model simulations (100 s simulated time), network construction is less relevant. Tables 6 and 7 list the details of implemented cases for NEST and Elephant. Arbor benchmarks are implemented

Case	**Number of synapses**	**Firing rates** (Hz)	**Minimal delay** (ms)
Standard HPC benchmark	11250	5 to 10	1.5
Simplified multiarea model	5625	14.8	0.1

Table 6.: NEST benchmark case detail

using inbuilt `miniapp.exe` with various neuron configurations such as standard three-segment-dendrite benchmark and non-spiking pyramidal cell benchmark.

Case	Details
Pure-Python cross-correlation	serial
NumPy-supported cross-correlation histograms	thread parallel
ASSET algorithm on random data	MPI via MPI4Py
ASSET on data with patterns	MPI via MPI4Py

Table 7.: Elephant benchmark case detail

5.2. Molecular dynamics (Task leader: NCSA)

Molecular dynamics (MD) is a theoretical approach widely used in the field of material sciences, life sciences, chemistry, etc. for studies of processes and phenomena on spatial and temporal scales unreachable by experimental techniques. For this purpose GROMACS which is one of the leading molecular dynamics package is chosen. Data preparation(system construction, force field assignment etc.) simulation and post-processing(analysis and visualization) represent general workflow for an end to end simulation. GROMACS is written in C/C++ in a hybrid MPI/OpenMP manner. Minimal software requirements are: C compiler, C++ compiler that supports C++11 standards, MPI, OpenMP, either FFTW or MKL library. Three use cases will be used for benchmarking. These are chosen to represent small, average and big atomic systems and consist of approximately 34k(Magainin), 325k(Bombinin) and 2.2M(Ribosome) atoms respectively. Given the atomic system and algorithms, the performance of a molecular dynamics program is measured by the simulated time per unit wall-clock time. The most convenient units are ns/day (nanoseconds per day). Another important metric for a machine benchmarking is the application turnaround. Table 8 describe the details of each benchmark case. Appendix

Name	Description	Number of atoms Simulation box dimensions	DEEP Partitions (number of nodes)
Magainin	Antimicrobial peptide Magainin solvated in water	34k 8X8X5.3nm^3	KNL(1,2,4) Xeon(1,2,4,8)
Bombinin	27 antimicrobial peptide Bombinin molecules forming 2 aggregates in water solutions	325k 15X15X15nm^3	KNL(1,2,4) Xeon(1,2,4,8)
Ribosome	Ribosome unit solvated in water	2.2M 31.6X31.6X22.3.3nm^3	KNL(1,2,4) Xeon(1,2,4,8)

Table 8.: GROMACS use case description

A.22, A.23 and A.24 list the JUBE xml files for this benchmark.

5.3. Radio astronomy (Task leader: ASTRON)

ASTRON has two applications: the correlator and the imager for which performance baseline will be set. The correlator combines the signals from tens or hundreds of receivers, while the imager creates sky images from these correlations (after removing bad data affected by interference and after correlations are calibrated to compensate for instrumental and environmental effects). Both the applications contain internal benchmark suites, which can measure runtimes(in seconds) and the performance(in TFLOPS) of relevant compute kernels as well as energy efficiency (in GFLOP/J) using external libraries. Additionally, both applications can be run in such a way that they generate synthetic input, on the fly. This simplifies benchmarking and allows isolating compute performance from I/O performance.
Within DEEP-EST project FPGA and CUDA based implementations of these applications are planned for development. Due to experimental and incomplete nature of implementations, baseline measurement setups can not be used. Neither applications use MPI. 32-bit minimum size of operand on modern CPU is superfluous and thus throughput is limited where 8 or 16 bit operands are sufficient. For the imager application performance of sine/cosine operations is important. GPUs can perform these operations on dedicated hardware where on CPUs the imager application spends roughly 85% of time in calculating sine/cosine. The performance/energy gap between GPUs and CPUs is typically about a factor of 15. Thus no meaningful baseline measurement is set on current DEEP-EST SDV. As mentioned in 4, as the appropriate hardware components are avaialble, relevant benchmark cases will be implementd. Table 9 lists proposed parameter space for imager application.

Parameter	Value
NR_STATIONS	120
NR_CHANNELS	16
NR_TIME	8192
NR_TIMESLOTS	do not set
IMAGESIZE	do not set
GRIDSIZE	8192
SUBGRIDSIZE	32
KERNELSIZE	do not set
NR_CYCLES	4

Table 9.: Proposed ASTRON imager benchmark parameters

5.4. Space Weather (Task leader: KU Leuven)

The space weather application(SWA) is composed of two complementary sections:

- **DLMOS**: A Deep Learning Model of the Solar Wind to forecast plasma conditions at the orbit of the Earth from images of the Sun
- **xPic**: A Particle-in-Cell code for the detailed simulation of the plasma environment of the planets.

The SWA will demonstrate that a coupling of Machine Learning and HPC can improve the understanding of the Sun-Earth interactions, with the long-term goal of performing accurate forecasting of the effects of solar activity on the Earth plasma environment. For each application, 4 benchmarks are proposed.

5.4.1. DLMOS

The second application DLMOS based on TensorFlow will be used for modelling solar winds using deep learning algorithms. Current benchmarks suite includes performance of Python packages, TensorFlow and I/O efficiency.

1. Multi-Layer Perceptron based on NumPy and Pandas framework
2. TensorFlow and Keras based benchmark of case 1.
3. Extended case 1 with higher number of nodes and layers
4. Operational testing of convolutional neural networks based on SDO satellite data input.

Memory	Inputs	Nodes in input layer	Layers	Nodes per layer	Epochs	I/O
280 MB	52000	40	3	20X10X1	200	data sets: 16 MB NN weights: 13 KB
280 MB	52000	40	3	20X10X1	200	data sets: 16 MB NN weights: 13 KB
600 MB	52000	192	5	100X40X20X10X1	2005	data sets: 80 MB NN weights: 194 KB
approx.3 GB	260000	7.8 million	3 convolutional 2 fully connected	8, 12, 24 (channels) + 128, 8 (nodes)	1	data sets: 600 GB NN wights: 600 KB

Table 10.: DLMOS benchmark parameters

5.4.2. xPic

1. A micro-benchmark for comparisons with iPic3D.
2. A large benchmark to check the code under realistic conditions.
3. A mid-size benchmark to test the code under realistic conditions and test I/O.
4. A micro-benchmark to test the effects of cache misses.

Type of run	Memory	Number of cells per species	Particles per cell per process	Blocks per MPI process	Iterations	Numerical method	I/O frequency
iPic3D performance comparison	Blocks of size 200 KB Total memory use 1.3 GB	16384	100	64	10	IMM	no I/O
xPic performance test	Blocks of size 200 KB Total memory use 16 GB	24576	1000	1024	10	IMM	no I/O
Small size production run of xPic with realistic workloads	Blocks of size 200 KB Total memory use 0.5 GB	768	1000	32	51	IMM	Field file per 10 iter. particle file per 25 iter.
Small size production run of xPic with realistic workloads	Blocks of size 6.1 KB and 200 KB Total memory use 1 GB	1536	1000	2 and 64	10	IMM	no I/O

Table 11.: xPIC benchmark parameters

Appendix A.18, A.19, A.20 and A.21 list the JUBE xml files for this benchmark.

5.5. Data analytics in Earth Science (Task leader: UoI)

5.5.1. HPDBSCAN

HPDBSCAN is a highly parallel implementation of the established DBSCAN clustering algorithm that takes points of an arbitrary dimension as one of its input argument and correctly labels each point to a cluster ID or identifies a point as noise. DBSCAN involves two required input parameters: the minimal number of points required to form a cluster and the maximum neighborhood search radius.
Its implementation is written in C++ and is highly optimized with regards to performance, using both shared and distributed memory parallelisms via OpenMP and MPI respectively. Furthermore, parallelism is also employed with the HDF5 API when the input data is read and the output data written.
HPDBSCAN uses a dataset of the Inner city of Bremen, a digital cartography of the city using point-clouds generated with 3D laser scans that results in over one GB of raw point-cloud data.The application is benchmarked with this dataset in two configurations, both of which are executed on the cluster(Xeon) partition of the DEEP-EST system. Its most important benchmarking metric is the total execution time, but another important metric is how well the application scales when it is applied using a different number of hardware resources.

1. For the first benchmark, a sub-sample of the Bremen dataset is used with a low number of cores which the application developers use as a comparative baseline to the second benchmark specified below. This benchmark uses two nodes, an equal amount of MPI processes, and two threads for each process.
2. For the second benchmark, the full Bremen dataset is used ($\approx$1.3GB) which is 32 times larger than its sub-sample used in the benchmark above, using identical input parameters. Therefore, it uses 32 times the amount of cores, i.e. 8 nodes, an equal amount of MPI processes, and 16 threads per process.

Appendix A.15, A.16 and A.17list the JUBE xml files for this benchmark.

5.5.2. PiSVM

PiSvM is a parallel support vector machine (SVM) implementation which is used to classify hyper-spectral data of natural and man-made land covers via supervised learning. First, the input data is pre-processed with feature engineering before the trained models are produced using different kernels or datasets, using cross-validation when necessary. Finally, the models classification accuracies and error-rates are determined by performing predictions on a separate dataset, which is usually sampled from the original dataset used for training.
The PiSvM implementation is written in C++ and uses MPI for distributed memory parallelism, but not OpenMP. The application uses the ”Indian Pines” dataset which was gathered by the AVIRIS sensor on board of an aircraft over a test site in North-western Indiana in the US. It is made up of 145x145 pixels and 224 spectral reflectance bands of labelled data, consisting mostly of agriculture but also includes forests, roads and some large structures. The most important metrics to be measured on the DEEP-EST system is the execution time of the training and prediction phases respectively, and the measured accuracy of a trained model that is

reported after the prediction has finished its execution. It was reported that the application's performance is sensitive towards hyper-threading which was therefore omitted in the benchmarks.
For training, the optimized Indian Pines training dataset, which is ≈12 MB is used to ensure convergence. Training will take a few minutes using the configuration below, using 48 cores with an equal amount of MPI processes, on two nodes. The model file generated by this phase (100 MB) can then be used to classify unseen data, this is performed in the subsequent prediction phase below.
For the prediction, the same number of cores as in the training phase above is used to allow the application developer to compare the execution times of these two phases. For inference the application uses an "Indian Pines" test dataset, which is significantly larger than the dataset from the previous training phase, ≈106 MB vs ≈12 MB of the training phase. Appendix A.13 and A.14 describe the JUBE xml files used for this benchmark.

5.5.3. Deep Learning with Tensorflow and the Keras extension

TensorFlow is an open-source machine learning framework that is used to create neural networks for all kinds of applications. The core of TensorFlow is implemented in C++, but it is also possible to run TensorFlow on accelerator hardware such as GPUs.
Keras is a user-friendly library on top of TensorFlow that allows to use high-level Python code to create machine learning applications based on TensorFlow.
This application is in fact the same machine learning application as was described in the PiSVM section above, i.e. the classification of hyper-spectral data of natural and man-made land covers via supervised learning. However, instead of a support vector machine (SVM) a 3D convolutional neural network (3DCNN) is used instead. Furthermore, it uses the same "Indian Pines" dataset. However, unfortunately 3DCNNs, as currently used in the application, are not well supported by Intel MKL, therefore it is not possible to properly benchmark the application at this time.

5.6. High Energy Physics (Task leader: CERN)

The CMS Experiment is one of the four large experiments at the Large Hadron Collider. The overall collection of software used by the CMS Experiment for data analysis, referred to as CMSSW, is built around a Framework, an Event Data Model (EDM) and Services needed by the simulation, calibration, alignment and reconstruction modules that process event data so that physicists can perform analysis. The primary goal of the Framework and EDM is to facilitate the development and deployment of reconstruction and analysis software.The CMSSW event processing model consists of one executable, called cmsRun and many plug-in modules which are managed by the Framework. All the code needed in the event processing (calibration, reconstruction algorithms, etc.) is contained in the modules. The same executable is used for both detector and simulation data. Application throughput is the important benchmark metric CMSSW. Throughput is defined as the number of simulated and/or processed collision events per unit of time (e.g. second). Monte Carlo generation, simulation, reconstruction and collision data detection with various workflows are implemented as benchmarks.

6. Workload Format for Modular architectures

This section proposes a first version for what we have called Modular Workload Format (MWF). The MWF proposal is based on the already existing Standard Workload Format with additional extensions to fit into the DEEP-EST project requirements. We have also taken into account special characteristics existing in the SLURM Workload Manager when submitting jobs since it is the job scheduler to be used in the project. For a better understanding of the format, we have included a synthetic example using the DEEP-EST applications as reference.

Standard Workload Format (SWF) is a widely used format in job scheduling research as a standard way for the evaluation of job scheduling policies. A repository of SWF workload traces provided by many HPC centres can be found at [11][10]. There are also several proposals of workload models that generate job scheduling logs in SWF. However, when it comes to evaluation of modular architectures, the SWF has several limitations:

- It was specified in 1999 when the job's and architecture's models were simpler than the actual models, i.e., they were not designed for modular architecture (MA)
- It only includes CPU and Memory as requested resources
- It does not include power/energy requirements/usage information
- Per-job information does not include dynamic resource requirements such as, for example, an application asking dynamically for additional module resources
- It is designed for HPC workload. It is still not clear if other applications, beside traditional HPC applications, will require additional fields in the workload format.

Here, we use SWF as a reference to propose a Modular Workload Format (MWF). Next section presents and analyzes fields already included in the SWF. For each field, we will evaluate the context to which it applies (job, module, etc). Our goal is to propose a new format that can include, as many as possible, SWF's fields which may allow us to potentially reuse the existing SWF repository.

6.1. Standard Workload Format

The SWF was defined in order to ease the use of workload logs and models. SWF allows for simplification of the programs that analyze workloads or simulate system's job scheduling since they only need to be able to parse a single format, and it can be applied to multiple workloads. The SWF files are portable and easy to parse:

- Each workload is stored in a single ASCII file.
- Each job is represented by a single line in the file.
- Lines contain a predefined number of fields, which are mostly integers, separated by whitespace(s).
- Fields that are irrelevant for a specific log or model appear with a value of -1.
- Comments are allowed, and identified by lines that start with a ';'. In particular, files are

expected to start with a set of header comments that define the environment or model.

- The same format is used for logs and model outputs.
- The format is completely defined, with no scope for user extensibility.

6.1.1. Current fields in SWF

In this section we give the description of SWF fields. As we mentioned earlier, SWF is not designed for MAs, but it can be envisioned as a workload format for a MA with one module. We have analysed fields to identify which ones are job-specific (J) or potentially module-specific (M). Fields marked as W are used to specify job dependences, i.e., workflows. [1]

1. (J) Job Number – a counter field, starting from one.
2. (J) Submit Time – in seconds. The earliest submit time in the log is zero, and usually, it is the submit time of the first job. The lines in the log are sorted by ascending submit times. It makes sense for jobs to also be numbered in this order.
3. (J) Wait Time – in seconds. The difference between the job's submit time and the time at which it actually started its execution. It is only relevant to real logs, not to models.
4. (J) Run Time – in seconds. The total execution time of the job, i.e., end time minus start time. We decided to use "wait time" and "run time" instead of the equivalent "start time" and "end time" because they are directly attributable to the scheduler and application, and are more suitable for models where only the run time is relevant. Note that when values are rounded to an integral number of seconds (as often happens in logs) a run time of 0 is possible and means the job ran for less than 0.5 seconds. On the other hand it is permissible to use floating point values for time fields.
5. (M) Number of Allocated Processors – an integer. In most cases this is also the number of processors the job uses; if the job does not use all of them, we typically don't know about it.
6. (M) Average CPU Time Used – both user and system, in seconds. This is the average over all processors of the CPU time used, and may therefore be smaller than the wall clock runtime. If a log contains the total CPU time used by all the processors, it is divided by the number of allocated processors to derive the average.
7. (M) Used Memory – in kilobytes. This is again the average per processor.
8. (M) Requested Number of Processors.
9. (M) Requested Time. This can be either runtime (measured in wallclock seconds), or average CPU time per processor (also in seconds) – the exact meaning is determined by a header comment. In many logs this field is used for the user runtime estimate (or upper bound) used in backfilling. If a log contains a request for total CPU time, it is divided by the number of requested processors.
10. (M) Requested Memory (again kilobytes per processor).
11. (J) Status 1 if the job was completed, 0 if it failed, and 5 if cancelled. If information about

[1] All these fields have been copied from the SWF web page

checkpointing or swapping is included, other values are also possible. See usage note below. This field is meaningless for models, so would be -1.

12. (J) User ID – a natural number, between one and the number of different users.
13. (J) Group ID – a natural number, between one and the number of different groups. Some systems control resource usage by groups rather than by individual users.
14. (J/M) Executable (Application) Number – a natural number, between one and the number of different applications appearing in the workload. in some logs, this might represent a script file used to run jobs rather than the executable directly; this should be noted in a header comment.
15. (J/M) Queue Number – a natural number, between one and the number of different queues in the system. The nature of the system's queues should be explained in a header comment. This field is where batch and interactive jobs should be differentiated: we suggest the convention of denoting interactive jobs by 0.
16. (J/M) Partition Number – a natural number, between one and the number of different partitions in the systems. The nature of the system's partitions should be explained in a header comment. For example, it is possible to use partition numbers to identify which machine in a cluster was used.
17. (W) Preceding Job Number – this is the number of a previous job in the workload, such that the current job can only start after the termination of this preceding job. Together with the next field, this allows the workload to include feedback as described below.
18. (W) Think Time from Preceding Job – this is the number of seconds that should elapse between the termination of the preceding job and the submittal of this one.

6.1.2. Modular systems requirements

According to the collected application's descriptions from application developers in [17], the following job requirements have been identified :

- Jobs asking for resources in a specific module
- Jobs asking for resources in more than one module at the same time
- Jobs asking for resources in more than one module but not at the same time, i.e., asking for additional resources dynamically
- Jobs asking for not only computing resources
- Jobs with dependencies among them, i.e., workflows

6.2. SLURM Heterogeneous Jobs

The last version of SLURM [12][13] includes support for heterogeneous jobs, also called job packs[15]. Heterogeneous jobs are composed by different components with same or different binary and executed in different sets of resources. Since DEEP-EST job scheduling is going to

be deployed based on SLURM, we have used the job pack concept, together with the Standard Workload Format previously existing, to define a Modular Workload Format to be usable with the newest architectures and where previously existing traces can be easily adapted to the new format.

The scheduling of heterogeneous jobs in the last SLURM version has the following characteristics:

- SLURM defines heterogeneous jobs as multiple independent job specifications identified in a single command.
- The entire request is validated or rejected (as a whole). All the limits are validated before starting any component
- Only backfill scheduler will allocate resources for heterogeneous jobs
- Components are allocated on different nodes
- All the components are allocated in the same cluster
- The Job pack ID and Job ID are defined as follows:
  - Pack Job ID: Unique for all the jobs of a heterogeneous job
  - Job ID: Unique for the job (packID+offset)
  - Job Offset: Local to packed job
  - Packed job set: job IDs range in a Packed job
- Job IDs are unique even though in federated clusters

6.3. Modular Workload Format (MWF) proposal

We define a modular job as a scheduling unit belonging to a single user containing a single or multiple binaries. A modular job will be the equivalent to a job pack defined in SLURM. At submission time, the job will include N, potentially one, list of requirements for allocation and components submission. All these allocations will be validated before any of these components start. These allocations could refer to different modules. At runtime, one component can start new processes in some previously created allocation, or can create new components. Before starting any new component, job limits will be validated.

One should differentiate between resource requirements, i.e., job scheduler's input data, and resource allocation, i.e., job scheduler's output data. Our proposal is to include in the trace file both pieces of information, as it was done in SWF, in order to describe jobs requirements and, in the same format, resource allocation when describing a real workload trace file. Only resource requirements will be used for job scheduling evaluation. Resource allocation will be used for comparison.

Each line will describe one component from one modular job. A modular job must contain at least one component. Dependencies can be specified between components of different modular jobs. Each workload trace file needs to provide a system description, i.e., available modules, resources at each module, etc.

Next subsections include the list of fields and a brief description of the semantic and valid values following a similar approach as in the SWF.

6.3.1. Modular Workload Format fields

Modular fields

Modular_Job_Id – An ID common to all the components of the modular job.

Total_Components – Number of components in the modular job (minimum one)

Modular_Job_Name – Text

Submit_Modular_Job_Time – in seconds. Submission time for the first set of components

Wait_Modular_Job_Time – in seconds. The difference between the job's submit time and the time at which it actually began to run (some of its components). It is not needed for evaluation, only for comparison between results.

Modular_Requested_Time – in seconds. Limit for the modular job. -1 if this value is not provided. In that case, the partition limit will be used.

Num_Components_At_Submit_Time – Integer. This field is the number of components submitted together at modular submit time.

Component fields

Component_Job_Id : Modular_Job_Id+ Offset – This JOB ID is unique. It goes from Modular_Job_Id to Modular_Job_Id+(NumComponents-1).

Component_Job_Name – text

Wait_Component_Job_Time – in seconds. The difference between the job's submit time and the time at which it actually began to run. It is not needed for evaluation, only for comparison between results.

Component_Run_Time – in seconds. Integral number of seconds.

Status – 0 means COMPLETED with success. Values different from 0 will represent errors.

Component resource requirements description

The job scheduler receives job component requirements, applies the job scheduler and resource selection policy, and reports a set of resources allocated. Resource allocation is reported for comparison but it is not part of the input. One component will run in a single module. If one job needs more than one module, one component per module will be specified.

HW resources

Executable_Number – a natural number, between one and the number of different applications appearing in the workload. in some logs, this might represent a script file used to

run jobs rather than the executable directly; this should be noted in a header comment.

Partition_Name – Text with the partition name; NA, if no specific partition is requested.

Nodes – an integer. Number of nodes requested

Processes_Per_Node – an integer

Threads_Per_Process – an integer

Memory_Per_Node – In KB

Freq – frequency in kilohertz, min and max.

NAM – in KB. NAM is a global resource and users will be able to ask for it

Local_Storage – in MB

Network – list o network requirements. More than one can be added with AND $\&$ or OR $|$

Constraint – A set of keywords, potentially with $\&$ or $|$ special characters. These constraints must be specified in the different modules to simplify resource selection. For instance, based on sbatch manual [14] $intel\&gpu, intel|amd$

Hint – at least cpu_intensive and memory_intensive. SLURM supports hints for resource allocation. These hints can be also used, and extended, for energy–performance models

SW resources

Licenses – Text. More than one can be added with AND $\&$ or OR $|$

Component resource allocation description

One component will run in a single Module. If one job needs more than one module, one component per module will be specified.

HW resources

Component_Module_Id – 0 - Number of Modules (One component will run in a single module). Module ID where this component is executed.

Partition_Name – Text with the partition name selected.

Nodes – Number of allocated nodes.

Processes_Per_Node – an integer

Threads_Per_Process – an integer

Memory_Per_Node – In KB (0 if not requested)

Freq – frequency in kilohertz.

Nam – in KB (0 if not requested)

Local_Storage – in MB (0 if not requested)

Network – keywords describing resources allocated or NA if not requested

SW resources

Licenses – licenses allocated or NA if not requested

Dependencies

After_Component_Job_Id – This component must start after job ID. -1 if there is no dependency

Dependency_Type – NA/DYNAMIC/AFTER/AFTERANY/AFTEROK/AFTERNOTOK//SINGLE. This is the list of types of dependencies supported by SLURM. DYNAMIC is an additional type defined here.

- NA means there is not dependency
- DYNAMIC means the component must be started N seconds after AFTER_COMPONENT_JOB_ID. The number of seconds is defined in the next field, and in that case it is relative to the dependent job start time.
- AFTEROK/AFTERNOTOK – This job can begin execution after the specified jobs have successfully/not successfully executed
- SINGLE is a special case defined by SLURM. This job can begin execution after any previously launched jobs sharing the same job name and user have terminated

Think_Component_Time – in seconds. When DYNAMIC is selected, this think time is relative to the job stat time, otherwise it is related to the job finalization.

6.3.2. Headers

The SWF includes some headers comments describing workload characteristics. These headers describe the architecture where the trace were collected and are interpreted as comments when reading the trace file (lines starting by ; are comments). These headers are **not mandatory** and are included to characterize the system at which the trace file was recorded. Since most of the headers are text values

Since SWF is designed for a single Cluster Module, we should replicate those fields related to the different modules. We propose to extend it with additional fields such as the number of modules. New headers are marked as “new”. All these fields are used to describe the context at which the traces were collected. Most of them are text fields and are not designed to be automatically processed.

- Version: Version number of the standard format the file uses. The format proposed (mwf) could be tagged as version 1 for mwf.
- Conversion: Name and email of whoever converted the log to the standard format (for traces corresponding to real HPC centres)
- MaxJobs: Integer, total number of jobs in this workload file.
- MaxRecords: Integer, total number of records in this workload file. If no checkpointing/swapping information is included, there is one record per job, and this is equal to MaxJobs. But with checkpointing/swapping there may be multiple records per job.

- UnixStartTime: When the log starts, in Unix time (seconds since the epoch) TimeZoneString is a standard UNIX string indicating the time zone in which the log was generated; this is actually the name of a zoneinfo file, e.g. “Europe/Paris”. All times within the SWF file are in this time zone. For more details see the usage note below.
- StartTime: When the log starts, in human readable form, in this standard format: Tue Feb 21 18:44:15 IST 2006 (as printed by the UNIX ’date’ utility).
- EndTime: When the log ends (the last termination), formatted like StartTime.
- **(new)NumberModules**: Number of modules in the system, for each Module
- **(new)ModuleNumber**: from 0 to max modules
- Computer: Brand and model of computer
- Installation: Location of installation and machine name
- Acknowledge: Name of person(s) to acknowledge for creating/collecting the workload.
- Information: Web site or email that contain more information about the workload or installation.
- MaxNodes: Integer, number of nodes in the module. List the number of nodes in different partitions in parentheses if applicable.
- MaxProcs: Integer, number of processors in the computer. This is different from MaxNodes if each node is an SMP. List the number of processors in different partitions in parentheses if applicable.
- MaxRuntime: Integer, in seconds. This is the maximum that the system allowed, and may be larger than any specific job’s runtime in the workload.
- MaxMemory: Integer, in kilobytes. Again, this is the maximum the system allowed.
- MaxQueues: Integer, number of queues used.
- Queues: A verbal description of the system’s queues. It should explain the queue number field (if it has known values). As a minimum it should be explained how to tell between a batch and interactive job.
- Queue: A description of a single queue in the following format: queue-number queue-name (optional-details). This should be repeated for all the queues.
- MaxPartitions: Integer, number of partitions used.
- Partitions: A verbal description of the system’s partitions, to explain the partition number field. For example, partitions can be distinct parallel machines in a cluster, or sets of nodes with different attributes (memory configuration, number of CPUs, special attached devices), especially if this is known to the scheduler.
- Partition: Description of a single partition.

6.3.3. Example

We have created a simple synthetic workload trace file to see how these fields must be used. The workload covers the basic DEEP-EST use cases.

We assume a system with three modules: CM, ESB and DA

1. job 1: traditional HPC job with 1 component
2. job 2: job submitted with 2 components started at same time in CM and ESB
3. job 3: job with 1 component in CM and 10 min later a new one uses ESB
4. job 4 : job with 3 components: 2 at the same time in CM and ESB and 1 later in DA (5 min later)
5. job 5 : HPC job using 1 component
6. job 6 : HPC job started after job 5 only if it is ok
7. job 7 : HPC job started after job 6 only if it is ok

Each job is submitted 10 seconds after the previous one, except Jobs 5, 6, 7 which are submitted at the same time. Since it is complicated to include a trace file in this document, we have made it public in the following link [18]. Moreover, we have used the same workload two show how the same format can be used to specify the input for simulation and to include the input and output for evaluation. The first file, D2.1_example_input.mwf, is a trace file to be used as input. The second one, D2.1_example.mwf could represent a potential output after a job scheduling simulation.

The following example is a simplified version where applications coming from WP1 have been used as reference. We have removed fields corresponding to the output to reduce the amount of columns (fields names has also been reduced). In the example, id 1 is a gromacs application. It has been represented as one single job with 3 components submitted together on all different modules. Second and third jobs(represented by job id 4 and 5 respectively) creates a workflow where job 5 is started after job 4. Job 6 is a job with 3 components. First one is started alone and after 100 seconds starts job component 7 and 8. Job 9 is a single component to be executed in DAM specifically. And jobs 10 and 11 create a simple workflow where job 11 will start after job 10 ends. We can see how some jobs (or components) requests for specific partitions (such as Elephant or DLMOS-training) whereas others can be executed in more than one module. In that case, the number of nodes is not fixed and we propose to specify the number of processors and threads (for instance CMSSW)

MOD_JOB_ID, TOTAL_COMP, MOD_JOB_NAME , SUB_MOD_JOB_TIME, MOD_REQ_TIME, NUM_COMP_AT_SUBMIT_TIME, COMP_JOB_ID, COMP_JOB_NAME, COMP_RUN_TIME, EXEC_NUMBER, PART_NAME, NODES, PROCS_PER_NODE, THREADS_PER_PROCESS, MEM_PER_NODE, FREQ, NAM, LOCAL

1, 3, GROMACS, 0, 600, 3, 1, GROMACS-LRI, 100,1, Default, 10, 64, 1, 0, 0, 1000000, 0, no, IntelXeonPhi, no, no, -1, NA, 0

1, 3, GROMACS, 0, 600, 3, 2, GROMACS-FFT, 100, 2, DAM, 20, 48, 1, 0, 0, 0, 100000, no, highmem, no, no, -1, NA, 0

1, 3, GROMACS ,0, 600, 3 3, GROMACS-SRI, 100, 3, Default, 50, 48, 1, 4.000.000, 2100, 0, 0, OmniPath, skylake & highmem, no, no, -1, NA, 0

4, 1, CMSSW, 0, 200, 1 4, CMSSW, 100, 4, Default, 0, 100, 0, 0, 0, 0, no, no, DA | CM | ESB, no, -1, NA, 0

5, 1, CMSSW-Apache, 0, 200, 1, 5 CMSSWApache, 100, 5, DAM, 10, 64, 1, 0, 0, 1000000, 0, no, IntelXeonPhi, no, no, 4, AFTEROK, 0

6, 3, NEST, 20, 1200, 1, 6, CM, 700, 5, CM, 50, 48, 1, 4.000.000, 2100, 0, 0, OmniPath, skylake & highmem, no, no, -1, NA, 0

6, 3, ARBOR, 20, 1200, 1, 7, ESB, 400, ESB, 40, 64, 1, 0, 0, 0, 0, no, no, no, no, 6, DYNAMIC, 100

6, 3, ELEPHANT, 20, 1200, 1, 8, DAM, 400, 2, DAM, 10, 64, 1, 0, 0, 1000000, 0, no, IntelXeonPhi ,no, no, 6, DYNAMIC, 100

9, 1, DLMOS-training, 30, 1200, 1, 9, DLMOS, 300, 3, DAM, 20, 48, 1, 0, 0, 0, 100000, no, highmem, no, XXX, -1 ,NA, 0

10, 2, CORREL, 30, 2000, 2, 10, CORREL, 2000 , DAM, 20, 48, 1, 0, 0, 0, 100000, no ,highmem, no, no -1, NA, 0

11, 2, IMAGER, 30, 2000, 2, 11, IMAGER, 2000, DAM, 20, 48, 1, 0, 0, 0, 100000, no ,highmem, no, no 10, AFTEROK, 0

7. Summary and Next Steps

This document describes the set of synthetic and application benchmarks to monitor the progress of application and system development in continuous benchmarking fashion.(Chapters 4 and 5) MPI LinkTest for latency and bandwidth with various parameters is implemented to monitor the network performance. IOR and h5perf are implemented to gauge the I/O throughput. These benchmarks along with few application benchmarks are already implemented in JUBE for automated and independent execution. A continuous benchmarking strategy is proposed and partially implemented. Standard Workflow Format(SWF) for modular supercomputing architectures is presented in chapter 6

As a next step, remainder of the application will be integrated in central benchmarking suite with automated analysis and result extraction. JUBE has a provision for result extraction from all the involved steps from a given benchmark. Extracted results and automated graphical output will be stored in a central location and made available to the partners for evaluation. Additional baseline measurements on large scale available systems (e.g. JURECA at JSC) will be performed. Due to platform independent nature of the benchmarks, the suites can be ported to newer systems with minimal modifications. As the DEEP-EST system develops with newer modules, third party benchmarks will be implemented if the applications benchmarks do not sufficiently cover the system components.

Bibliography

[1] DEEP-ER Deliverables, http://www.deep-projects.eu/project/deliverables.html

[2] Lührs S., Rohe D., Schnurpfeil A., Thust K. and Frings W.: *Flexible and Generic Workflow Management*, in Proceedings of the International Conference on Parallel Computing 2015, ParCo 2015, Edinburgh, United Kingdom, 1 Sep 2015 - 4 Sep 2015, Parallel Computing: On the Road to Exascale, in Periodical Amsterdam : IOS Press, Advances in parallel computing, 27, (2016), http://dx.doi.org/10.3233/978-1-61499-621-7-431

[3] JUBE Online Documentation, https://apps.fz-juelich.de/jsc/jube/jube2/docu/index.html

[4] GitLab Benchmarks Repository, https://gitlab.version.fz-juelich.de/DEEP-EST/Benchmarks

[5] MPI LinkTest, http://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/LinkTest/_node.html

[6] Frings W., Wolf F., Petkov V.: *Scalable Massively Parallel I/O to Task-Local Files*, in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, Portland, Oregon, November 14 - 20, 2009, SC'09, SESSION: Technical papers, Article No. 17, New York, ACM, 2009. ISBN 978-1-60558-744-8. -S. 1-11 http://dx.doi.org/10.1145/1654059.1654077

[7] SIONlib, http://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/SIONlib/_node.html

[8] IOR, https://github.com/hpc/ior

[9] HDF5 Group, https://support.hdfgroup.org/HDF5/

[10] The Standard Workload Format,

http://www.cs.huji.ac.il/labs/parallel/workload/swf.html

[11] Steve J. Chapin, Walfredo Cirne, Dror G. Feitelson, James Patton Jones, Scott T. Leutenegger, Uwe Schwiegelshohn, Warren Smith, and David Talby, *Benchmarks and Standards for the Evaluation of Parallel Job Schedulers*. In Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph (Eds.), Springer-Verlag, 1999, Lect. Notes Comput. Sci. vol. 1659, pp. 66-89.

[12] Morris A. Jette, Andy B. Yoo and Mark Grondona: *SLURM: Simple Linux Utility for Resource Management*, in Proceedings of the 9th International Workshop Job Scheduling Strategies for Parallel Processing (JSSPP), Springer, Lecture Notes in Computer Science (LNCS), volume 2862, pages 44–60, http://dx.doi.org/10.1007/10968987_3

[13] SchedMD: *Slurm Workload Manager* [Online], https://slurm.schedmd.com/

[14] sbatch: sbatch - Submit a batch script to Slurm, https://slurm.schedmd.com/sbatch.html

[15] *Heterogeneous Resources and MPMD*, https://slurm.schedmd.com/SLUG15/Heterogeneous_Resources_and_MPMD.pdf

[16] SLURM: Heterogeneous job Support, https://slurm.schedmd.com/SLUG17/HeterogeneousJobs.pdf

[17] DEEP-EST Deliverable 1.1 Application co-design input

[18] Modular Workload Format: *Example files*, https://gitlab.version.fz-juelich.de/kulkarni1/modular_workload_format

Appendices

A. JUBE XML files

Listing A.1: hello_world_1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>

  <benchmark name="hello_world" outpath="bench_run">
    <comment>A simple hello world</comment>

    <!-- Benchmark configuration -->
    <parameterset name="hello_parameter">
      <!-- Create a parameterspace out of two template parameters -->
      <parameter name="text">Hello ,World</parameter>
      <parameter name="number">1,2,4</parameter>
    </parameterset>

    <!-- Operation -->
    <step name="say_hello">
      <use>hello_parameter</use> <!-- use parameterset "hello_parameter" -->
      <do>echo This is `hostname`. $text $number</do> <!-- shell command -->
    </step>

    <!-- Regex pattern -->
    <patternset name="pattern">
      <pattern name="node" type="string">This is (.*)\.\s+</pattern>
    </patternset>

    <!-- Analyse -->
    <analyzer name="analyse" >
      <use>pattern</use> <!-- use existing patternset -->
      <analyse step="say_hello">
        <file>stdout</file> <!-- file which should be scanned -->
      </analyse>
    </analyzer>

    <!-- Result -->
    <result>
      <use>analyse</use> <!-- use existing analyser -->
      <table name="result" style="pretty" sort="node">
        <column>node</column>
        <column>text</column>
        <column>number</column>
      </table>
    </result>

  </benchmark>

</jube>
```

Listing A.2: hello_world_2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>

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

  <benchmark name="hello_world" outpath="bench_run">
    <comment>A simple hello world</comment>

    <!-- Benchmark configuration -->
    <parameterset name="hello_parameter">
      <!-- Create a parameterspace out of two template parameters -->
      <parameter name="text">Hello,World</parameter>
      <parameter name="number">1,2,4</parameter>
    </parameterset>

    <!-- Job configuration -->
    <parameterset name="systemParameter" init_with="platform.xml">
      <parameter name="nodes">2</parameter>
      <parameter name="taskspernode">1</parameter>
      <parameter name="queue">sdv</parameter>
    </parameterset>

    <!-- Substitute jobfile -->
    <substituteset name="executesub" init_with="platform.xml">
      <sub source="#NOTIFY_EMAIL#" dest="c.manzano@fz-juelich.de" />
      <sub source="#EXECUTABLE#" dest="echo" />
      <sub source="#ARGS_EXECUTABLE#" dest="This is `hostname`. $text $number" /
          >
    </substituteset>

    <!-- Operation -->
    <step name="say_hello">
      <use>hello_parameter</use>
      <use>systemParameter</use>
      <use>executesub</use>
      <use from="platform.xml">executeset</use>
      <use from="platform.xml">jobfiles</use>
      <do>$submit $submit_script</do>
    </step>

    <!-- Regex pattern -->
    <patternset name="pattern">
      <pattern name="node" type="string">This is (.*)\.\s+</pattern>
    </patternset>

    <!-- Analyse -->
    <analyzer name="analyse" >
      <use>pattern</use> <!-- use existing patternset -->
      <analyse step="say_hello">
        <file>stdout</file> <!-- file which should be scanned -->
      </analyse>
    </analyzer>

    <!-- Result -->
```

```
56      <result>
57        <use>analyse</use> <!-- use existing analyser -->
58        <table name="result" style="pretty" sort="node">
59          <column>node</column>
60          <column>text</column>
61          <column>number</column>
62        </table>
63      </result>
64
65    </benchmark>
66
67  </jube>
```

Listing A.3: hello_world_3.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>

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

  <benchmark name="hello_world" outpath="bench_run">
    <comment>A simple hello world</comment>

    <!-- Benchmark configuration -->
    <parameterset name="hello_parameter">
      <!-- Create a parameterspace out of two template parameters -->
      <parameter name="text">Hello,World</parameter>
      <parameter name="number">1,2,4</parameter>
    </parameterset>

    <!-- Job configuration -->
    <parameterset name="systemParameter" init_with="platform.xml">
      <parameter name="nodes">2</parameter>
      <parameter name="taskspernode">1</parameter>
      <parameter name="queue">sdv</parameter>
    </parameterset>

    <!-- Substitute jobfile -->
    <substituteset name="executesub" init_with="platform.xml">
      <sub source="#NOTIFY_EMAIL#" dest="c.manzano@fz-juelich.de" />
      <sub source="#EXECUTABLE#" dest="echo" />
      <sub source="#ARGS_EXECUTABLE#" dest="This is `hostname`. $text $number" /
          >
    </substituteset>

    <!-- Operation -->
    <step name="say_hello" shared="shared">
      <use>hello_parameter</use>
      <use>systemParameter</use>
      <use>executesub</use>
      <use from="platform.xml">executeset</use>
      <use from="platform.xml">jobfiles</use>
      <use from="platform.xml">chainfiles</use>
      <do>$chainjob_script $shared_job_info $submit_script</do>
    </step>

    <!-- Regex pattern -->
    <patternset name="pattern">
      <pattern name="node" type="string">This is (.*)\.\s+</pattern>
    </patternset>

    <!-- Analyse -->
    <analyzer name="analyse" >
      <use>pattern</use> <!-- use existing patternset -->
      <analyse step="say_hello">
        <file>stdout</file> <!-- file which should be scanned -->
      </analyse>
    </analyzer>

```

```
56      <!-- Result -->
57      <result>
58        <use>analyse</use> <!-- use existing analyser -->
59        <table name="result" style="pretty" sort="node">
60          <column>node</column>
61          <column>text</column>
62          <column>number</column>
63        </table>
64      </result>
65
66    </benchmark>
67
68  </jube>
```

Listing A.4: submit.job.in

```
#!/bin/bash -x
#SBATCH --job-name=#BENCHNAME#
#SBATCH --mail-user=#NOTIFY_EMAIL#
#SBATCH --mail-type=#NOTIFICATION_TYPE#
#SBATCH --nodes=#NODES#
#SBATCH --ntasks=#TASKS#
#SBATCH --ntasks-per-node=#NCPUS#
#SBATCH --cpus-per-task=#NTHREADS#
#SBATCH --time=#TIME_LIMIT#
#SBATCH --output=#STDOUTLOGFILE#
#SBATCH --error=#STDERRLOGFILE#
#SBATCH --partition=#QUEUE#

#ENV#

#PREPROCESS#

#MEASUREMENT# #STARTER# #ARGS_STARTER# #EXECUTABLE# #ARGS_EXECUTABLE#

#POSTPROCESS#

JUBE_ERR_CODE=$?
if [ $JUBE_ERR_CODE -ne 0 ]; then
    exit $JUBE_ERR_CODE
fi

#FLAG#
```

Listing A.5: 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 $LOCKFILE`
    echo "sbatch --dependency=afterany:${DEPEND_JOBID} $SUBMITSCRIPT"
    JOBID=`sbatch --dependency=afterany:${DEPEND_JOBID} $SUBMITSCRIPT`
else
    echo "sbatch $SUBMITSCRIPT"
    JOBID=`sbatch $SUBMITSCRIPT`
fi

JUBE_ERR_CODE=$?
if [ $JUBE_ERR_CODE -ne 0 ]; then
    exit $JUBE_ERR_CODE
fi

echo "RETURN: $JOBID"
# the JOBID is the last field of the output line
echo ${JOBID##* } > $LOCKFILE

exit 0
```

Listing A.6: platform.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>
    <!-- Default DEEP sets -->
    <parameterset name="executeset">
        <!-- Jobscript handling -->
        <parameter name="submit">sbatch</parameter>
        <parameter name="submit_script">submit.job</parameter>
        <parameter name="done_file">ready</parameter>
        <parameter name="starter">srun</parameter>
        <parameter name="args_starter"></parameter>
        <!-- Chainjob handling -->
        <parameter name="shared_folder">shared</parameter>
        <parameter name="shared_job_info">${shared_folder}/jobid</parameter>
        <parameter name="chainjob_script">./deep-chainJobs.sh</parameter>
        <parameter name="chainjob_needs_submit">false</parameter>
    </parameterset>

    <parameterset name="systemParameter">
        <!-- Default jobscript parameter -->
        <parameter name="nodes" type="int">1</parameter>
        <parameter name="taskspernode" type="int">1</parameter>
        <parameter name="threadspertask" type="int">1</parameter>
        <parameter name="tasks" mode="python" type="int">
            $nodes * $taskspernode
        </parameter>
        <parameter name="OMP_NUM_THREADS" type="int" export="true">
            $threadspertask
        </parameter>
        <parameter name="queue">batch</parameter>
        <parameter name="executable"></parameter>
        <parameter name="args_exec"></parameter>
        <parameter name="mail"></parameter>
        <parameter name="env" separator=";">$jube_wp_envstr</parameter>
        <parameter name="notification">ALL</parameter>
        <parameter name="outlogfile">job.out</parameter>
        <parameter name="errlogfile">job.err</parameter>
        <parameter name="timelimit">00:30:00</parameter>
        <parameter name="preprocess"></parameter>
        <parameter name="postprocess"></parameter>
        <parameter name="measurement"></parameter>
    </parameterset>

    <substituteset name="executesub">
        <!-- Default jobscript substitution -->
        <iofile in="${submit_script}.in" out="$submit_script" />
        <sub source="#ENV#" dest="$env" />
        <sub source="#NOTIFY_EMAIL#" dest="$mail" />
        <sub source="#NOTIFICATION_TYPE#" dest="$notification" />
        <sub source="#BENCHNAME#"
             dest="${jube_benchmark_name}_${jube_step_name}_${jube_wp_id}" />
        <sub source="#NODES#" dest="$nodes" />
        <sub source="#TASKS#" dest="$tasks" />
        <sub source="#NCPUS#" dest="$taskspernode" />
        <sub source="#NTHREADS#" dest="$threadspertask" />
        <sub source="#TIME_LIMIT#" dest="$timelimit" />
        <sub source="#PREPROCESS#" dest="$preprocess" />
```

```
57          <sub source="#POSTPROCESS#" dest="$postprocess" />
58          <sub source="#QUEUE#" dest="$queue" />
59          <sub source="#STARTER#" dest="$starter" />
60          <sub source="#ARGS_STARTER#" dest="$args_starter" />
61          <sub source="#MEASUREMENT#" dest="$measurement" />
62          <sub source="#STDOUTLOGFILE#" dest="$outlogfile" />
63          <sub source="#STDERRLOGFILE#" dest="$errlogfile" />
64          <sub source="#EXECUTABLE#" dest="$executable" />
65          <sub source="#ARGS_EXECUTABLE#" dest="$args_exec" />
66          <sub source="#FLAG#" dest="touch $done_file" />
67      </substituteset>
68
69      <substituteset name="chainsub">
70          <!-- Default chainjob substitution -->
71      </substituteset>
72
73      <fileset name="jobfiles">
74          <!-- Default jobscript access -->
75          <copy>${submit_script}.in</copy>
76      </fileset>
77
78      <fileset name="chainfiles">
79          <!-- Chainjob script access -->
80          <copy>$chainjob_script</copy>
81      </fileset>
82  </jube>
```

Listing A.7: mpiLinkTest.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>
  <include-path>
    <path>/sdv-work/benchmarks/deep-est_benchmarks/synthetic/mpiLinkTest</path>
    <path>/usr/local/deep/platform/jureca</path>
  </include-path>
  <benchmark name="mpiLinkTest" outpath="mpiLinkTest_bench_run">
    <comment>mpiLinkTest benchmark</comment>

    <parameterset name="ExecArgsSmall" init_with="mpiLinkTest.params.xml">
    </parameterset>

    <parameterset name="ExecArgsLarge" init_with="mpiLinkTest.params.xml">
    </parameterset>

    <parameterset name="ExecArgsCongestionSmall" init_with="mpiLinkTest.params.
        xml">
    </parameterset>

    <parameterset name="ExecArgsCongestionLarge" init_with="mpiLinkTest.params.
        xml">
    </parameterset>

    <parameterset name="ExecArgsLatency" init_with="mpiLinkTest.params.xml">
    </parameterset>


    <!--parameterset name="systemParameter" tag="bandwidth" init_with="platform.
        xml"-->
    <parameterset name="systemParameter" init_with="platform.xml">
      <parameter name="nodes" type="init">1,2,3,4,5,6,7,8</parameter>
      <parameter name="taskspernode" type="int">2</parameter>
    </parameterset>

    <parameterset name="executeset" init_with="platform.xml">
      <parameter name="args_starter">-n $tasks</parameter>
    </parameterset>

    <substituteset name="executesub" init_with="platform.xml">
      <iofile in="submit.job.in" out="submit.job" />
      <sub source='#PREPROCESS#' dest="module purge; module load intel
          parastation extoll; date +'(start) %F %T (%s)'"/>
      <sub source='#EXECUTABLE#' dest="--cpu_bind=map_cpu:0,12 ./
          compile_mpiLinkTest/fzjlinktest/src/mpilinktest"/>
      <sub source='#QUEUE#' dest="sdv"/>
      <sub source='#ARGS_EXECUTABLE#' dest="-s ${Size} -I ${Warmup} -i ${
          Iterations} -S ${Serialized} -M ${Randomized} -a ${AllToAll}"/>
      <sub source='#POSTPROCESS#' dest="./compile_mpiLinkTest/fzjlinktest/src/
          pingponganalysis -p ./pingpong_results_bin.sion; date +'(end) %F %T (%
          s)'"/>
      <sub source='#MEASUREMENT#' dest="time"/>
    <sub source="#NOTIFICATION_TYPE#" dest="all"/>
    <sub source="#NODES#" dest="$nodes"/>
    <sub source="#NCPUS#" dest="$taskspernode"/>
    <sub source="#TASKS#" dest="$tasks"/>
    <sub source="#NTHREADS#" dest="1"/>
```

```
49    <sub source="#TIME_LIMIT#" dest="00:15:00"/>
50     </substituteset>
51
52     <step name="compile_mpiLinkTest">
53       <include from="mpiLinkTest.compile.xml" path="dos/do"/>
54     </step>
55
56     <step name="run_mpiLinkTest_small" depend="compile_mpiLinkTest" shared="
           shared">
57       <use from="platform.xml">jobfiles</use>
58       <use from="platform.xml">chainsub</use>
59       <use from="platform.xml">chainfiles</use>
60       <use>ExecArgsSmall</use> <!-- use existing parameterset -->
61       <use>executeset</use> <!-- use existing parameterset -->
62       <use>executesub</use> <!-- use existing parameterset -->
63       <use>systemParameter</use> <!-- use existing parameterset -->
64       <do done_file="ready">$chainjob_script ./shared/jobid $submit_script</do>
65     </step>
66
67     <step name="run_mpiLinkTest_large" depend="run_mpiLinkTest_small,
           compile_mpiLinkTest" shared="shared">
68       <use from="platform.xml">jobfiles</use>
69       <use from="platform.xml">chainsub</use>
70       <use from="platform.xml">chainfiles</use>
71       <use>ExecArgsLarge</use> <!-- use existing parameterset -->
72       <use>executeset</use> <!-- use existing parameterset -->
73       <use>executesub</use> <!-- use existing parameterset -->
74       <use>systemParameter</use> <!-- use existing parameterset -->
75       <do done_file="ready">$chainjob_script ./shared/jobid $submit_script</do>
76     </step>
77
78
79     <step name="run_mpiLinkTest_congestion_small" depend="run_mpiLinkTest_large,
           compile_mpiLinkTest" shared="shared">
80       <use from="platform.xml">jobfiles</use>
81       <use from="platform.xml">chainsub</use>
82       <use from="platform.xml">chainfiles</use>
83       <use>ExecArgsCongestionSmall</use> <!-- use existing parameterset -->
84       <use>executeset</use> <!-- use existing parameterset -->
85       <use>executesub</use> <!-- use existing parameterset -->
86       <use>systemParameter</use> <!-- use existing parameterset -->
87       <do done_file="ready">$chainjob_script ./shared/jobid $submit_script</do>
88     </step>
89
90
91     <step name="run_mpiLinkTest_congestion_large" depend="
           run_mpiLinkTest_congestion_small,compile_mpiLinkTest" shared="shared">
92       <use from="platform.xml">jobfiles</use>
93       <use from="platform.xml">chainsub</use>
94       <use from="platform.xml">chainfiles</use>
95       <use>ExecArgsCongestionLarge</use> <!-- use existing parameterset -->
96       <use>executeset</use> <!-- use existing parameterset -->
97       <use>executesub</use> <!-- use existing parameterset -->
98       <use>systemParameter</use> <!-- use existing parameterset -->
99       <do done_file="ready">$chainjob_script ./shared/jobid $submit_script</do>
100    </step>
101
```

```
102     <step name="run_mpiLinkTest_latency" depend="
            run_mpiLinkTest_congestion_large,compile_mpiLinkTest" shared="shared"
            active="$Size is not None">
103       <use from="platform.xml">jobfiles</use>
104       <use from="platform.xml">chainsub</use>
105       <use from="platform.xml">chainfiles</use>
106       <use>ExecArgsLatency</use> <!-- use existing parameterset -->
107       <use>executeset</use> <!-- use existing parameterset -->
108       <use>executesub</use> <!-- use existing parameterset -->
109       <use>systemParameter</use> <!-- use existing parameterset -->
110       <do done_file="ready">$chainjob_script ./shared/jobid $submit_script</do>
111     </step>
112
113     <analyzer name="analyse_small" reduce="false" >
114       <use from="mpiLinkTest.params.xml">pattern</use>
115       <analyse step="run_mpiLinkTest_small">
116         <file>job.out</file>
117       </analyse>
118     </analyzer>
119
120     <analyzer name="analyse_large" reduce="false" >
121       <use from="mpiLinkTest.params.xml">pattern</use>
122       <analyse step="run_mpiLinkTest_large">
123         <file>job.out</file>
124       </analyse>
125     </analyzer>
126
127     <analyzer name="analyse_congestion_small" reduce="false" >
128       <use from="mpiLinkTest.params.xml">pattern</use>
129       <analyse step="run_mpiLinkTest_congestion_small">
130         <file>job.out</file>
131       </analyse>
132     </analyzer>
133
134     <analyzer name="analyse_congestion_large" reduce="false" >
135       <use from="mpiLinkTest.params.xml">pattern</use>
136       <analyse step="run_mpiLinkTest_congestion_large">
137         <file>job.out</file>
138       </analyse>
139     </analyzer>
140
141     <analyzer name="analyse_latency" reduce="false" >
142       <use from="mpiLinkTest.params.xml">pattern</use>
143       <analyse step="run_mpiLinkTest_latency">
144         <file>job.out</file>
145       </analyse>
146     </analyzer>
147
148     <result>
149       <use>analyse_small</use>
150       <use>analyse_large</use>
151       <use>analyse_congestion_small</use>
152       <use>analyse_congestion_large</use>
153       <use>analyse_latency</use>
154       <table name="result" style="pretty" sort="nodes,taskspernode">
155     <column>jube_wp_id</column>
156         <column>nodes</column>
```

```
157           <column>taskspernode</column>
158           <column>serialized</column>
159           <column>iterations</column>
160           <column>messagesizeKB</column>
161           <column>mintimeus</column>
162           <column>maxbw</column>
163           <column>maxtimeus</column>
164           <column>minbw</column>
165           <column>avgtimeus</column>
166           <column>avgbw</column>
167           <column>starttime</column>
168           <column>stoptime</column>
169         </table>
170
171         <!--table name="result-csv" style="csv" sort="nodes,taskspernode"-->
172         <table name="result-csv" style="csv" sort="jube_wp_id">
173       <column>jube_wp_id</column>
174           <column>nodes</column>
175           <column>taskspernode</column>
176           <column>serialized</column>
177           <column>iterations</column>
178           <column>messagesizeKB</column>
179           <column>mintimeus</column>
180           <column>maxbw</column>
181           <column>maxtimeus</column>
182           <column>minbw</column>
183           <column>avgtimeus</column>
184           <column>avgbw</column>
185           <column>starttime</column>
186           <column>stoptime</column>
187         </table>
188       </result>
189
190     </benchmark>
191 </jube>
```

Listing A.8: mpiLinkTest.compile.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>
    <dos>
      <do>cp -r /usr/local/deep/sources/synthetic/fzjlinktest .</do> <!-- copy source code -->
      <do>module purge</do> <!-- load default modules -->
      <do>module load intel parastation extoll</do> <!-- load default modules -->
      <do>echo $$PWD</do>
      <do>cd fzjlinktest/src/; gmake clean; gmake -f Makefile_LINUX</do> <!-- copy source code -->
    </dos>
</jube>
```

Listing A.9: mpiLinkTest.params.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>
    <parameterset name="ExecArgsSmall">
      <parameter name="i">0,1,2,3,4</parameter>
      <parameter name="j">0</parameter>
      <parameter name="k">0</parameter>
      <parameter name="AllToAll" type="int">0</parameter>
      <parameter name="Size" update_mode="use" mode="python">[8192,16384,32768,65536,131072][$i]</parameter>
      <parameter name="Iterations" update_mode="use" mode="python">[10][$j]</parameter>
      <parameter name="Warmup" type="int">3</parameter>
      <parameter name="Randomized" type="int">0</parameter>
      <parameter name="Serialized" update_mode="use" mode="python">[1][$k]</parameter>
    </parameterset>

    <parameterset name="ExecArgsLarge">
      <parameter name="Size" update_mode="use" mode="python">[262144,524288,1048576,2097152,4194304][$i]</parameter>
      <parameter name="Iterations" update_mode="use" mode="python">[10][$j]</parameter>
      <parameter name="Serialized" update_mode="use" mode="python">[1][$k]</parameter>
    </parameterset>

    <parameterset name="ExecArgsCongestionSmall">
      <parameter name="Size" update_mode="use" mode="python">[8192,16384,32768,65536,131072][$i]</parameter>
      <parameter name="Iterations" update_mode="use" mode="python">[100][$j]</parameter>
      <parameter name="Serialized" update_mode="use" mode="python">[0][$k]</parameter>
    </parameterset>

    <parameterset name="ExecArgsCongestionLarge">
      <parameter name="Size" update_mode="use" mode="python">[262144,524288,1048576,2097152,4194304][$i]</parameter>
      <parameter name="Iterations" update_mode="use" mode="python">[100][$j]</parameter>
```

```
30        <parameter name="Serialized" update_mode="use" mode="python">[0][$k]</
              parameter>
31      </parameterset>
32
33      <parameterset name="ExecArgsLatency">
34        <parameter name="Size" update_mode="use" mode="python">[1,None,None,None,
              None][$i]</parameter>
35        <parameter name="Iterations" update_mode="use" mode="python">[10][$j]</
              parameter>
36        <parameter name="Serialized" update_mode="use" mode="python">[1][$k]</
              parameter>
37      </parameterset>
38
39      <patternset name="pattern">
40        <pattern mode="pattern" name="pat_walltime" type="float" unit="s">timings
              \[000\] \[work\]\s*t=\s*$jube_pat_fp[s]</pattern>
41        <pattern mode="pattern" name="mintimeus" type="float" unit="us">RESULT:
              Min Time:\s*$jube_pat_fp[u]s \(\s*$jube_pat_nfp MB\/s\)</pattern>
42        <pattern mode="pattern" name="maxbw" type="float" unit="MB/s">RESULT: Min
               Time:\s*$jube_pat_nfp[u]s \(\s*$jube_pat_fp MB\/s\)</pattern>
43        <pattern mode="pattern" name="maxtimeus" type="float" unit="us">RESULT:
              Max Time:\s*$jube_pat_fp[u]s \(\s*$jube_pat_nfp MB\/s\)</pattern>
44        <pattern mode="pattern" name="minbw" type="float" unit="MB/s">RESULT: Max
               Time:\s*$jube_pat_nfp[u]s \(\s*$jube_pat_fp MB\/s\)</pattern>
45        <pattern mode="pattern" name="avgtimeus" type="float" unit="us">RESULT:
              Avg Time:\s*$jube_pat_fp[u]s \(\s*$jube_pat_nfp MB\/s\)</pattern>
46        <pattern mode="pattern" name="avgbw" type="float" unit="MB/s">RESULT: Avg
               Time:\s*$jube_pat_nfp[u]s \(\s*$jube_pat_fp MB\/s\)</pattern>
47        <pattern mode="pattern" name="diffsockettimeus" type="float" unit="us">\s
              +1->    12:\s*$jube_pat_fp[u]s</pattern>
48        <pattern mode="pattern" name="diffsocketbw" type="float" unit="MB/s">\s
              +1->    12:\s*$jube_pat_nfp[u]s \(\s*$jube_pat_fp MB\/s\)</pattern>
49        <pattern mode="pattern" name="samesockettimeus" type="float" unit="us">\s
              +1->     2:\s*$jube_pat_fp[u]s</pattern>
50        <pattern mode="pattern" name="samesocketbw" type="float" unit="MB/s">\s
              +1->     2:\s*$jube_pat_nfp[u]s \(\s*$jube_pat_fp MB\/s\)</pattern>
51        <pattern mode="pattern" name="messagesizeKB" type="float" unit="KB">
              linktest: Message length:\s*$jube_pat_fp KBytes</pattern>
52        <pattern mode="perl" name="pat_mintimems" type="float" unit="s">$
              pat_mintimeus/1000</pattern>
53        <pattern mode="perl" name="pat_maxtimems" type="float" unit="s">$
              pat_maxtimeus/1000</pattern>
54        <pattern mode="perl" name="pat_avgtimems" type="float" unit="s">$
              pat_avgtimeus/1000</pattern>
55        <pattern mode="perl" name="pat_mintime" type="float" unit="s">$
              pat_mintimeus/1000/1000</pattern>
56        <pattern mode="perl" name="pat_maxtime" type="float" unit="s">$
              pat_maxtimeus/1000/1000</pattern>
57        <pattern mode="perl" name="pat_avgtime" type="float" unit="s">$
              pat_avgtimeus/1000/1000</pattern>
58        <pattern mode="pattern" name="pat_sionopen" type="float" unit="s">timings
              \[000\] \[sionopen\]\s*t=\s*$jube_pat_fp[s]</pattern>
59        <pattern mode="pattern" name="pat_sionwrite" type="float" unit="s">
              timings\[000\] \[sionwrite\]\s*t=\s*$jube_pat_fp[s]</pattern>
60        <pattern mode="pattern" name="pat_sionflush" type="float" unit="s">
              timings\[000\] \[sionflush\]\s*t=\s*$jube_pat_fp[s]</pattern>
61        <pattern mode="pattern" name="pat_sionclose" type="float" unit="s">
```

```
            timings\[000\] \[sionclose\]\s*t=\s*$jube_pat_fp[s]</pattern>
62          <pattern mode="pattern" name="pat_timeall" type="float" unit="s">timings
                \[000\] \[all\]\s*t=\s*$jube_pat_fp[s]</pattern>
63          <pattern mode="pattern" name="pat_Rtasks" type="int" unit="#">
                mpilinktest: Number of MPI-Task:\s*$jube_pat_int</pattern>
64          <pattern mode="pattern" name="pat_Rmsglen" type="int" unit="Bytes">
                mpilinktest: Message length:\s*$jube_pat_int Bytes</pattern>
65          <pattern mode="pattern" name="pat_RmsglenKB" type="int" unit="KB">
                mpilinktest: Message length:\s*$jube_pat_int KBytes</pattern>
66          <pattern mode="pattern" name="pat_Riter" type="int" unit="#">mpilinktest:
                 Number of Iteration:\s*$jube_pat_int</pattern>
67          <pattern mode="pattern" name="pat_Rlevel" type="float" unit="BW">
                mpilinktest: Level for suspected Links: .\s*$jube_pat_fp MB\/s</
                pattern>
68          <pattern mode="pattern" name="pat_Ratoa" type="int" unit="#">mpilinktest:
                 alltoall:\s*$jube_pat_int</pattern>
69          <pattern mode="pattern" name="pat_Rsleept" type="int" unit="#">
                mpilinktest: sleeptime:\s*$jube_pat_int us</pattern>
70          <pattern mode="pattern" name="pat_Rexecorder" type="int" unit="#">
                mpilinktest: execution order:\s*$jube_pat_int</pattern>
71          <pattern mode="pattern" name="serialized" type="int" >serial test:$
                jube_pat_bl$jube_pat_int</pattern>
72          <pattern mode="pattern" name="iterations" type="int" >Number of
                Iteration:$jube_pat_bl$jube_pat_int</pattern>
73
74          <pattern name="starttime" type="string" >^\s*\(start\)\s*(.*)\s+\(</
                pattern>
75          <pattern name="startdate" type="string" >^\s*\(start\)\s*(.*)\s+$
                jube_pat_int</pattern>
76          <pattern name="stoptime"  type="string" >^\s*\(end\)\s*(.*)\s+\(</pattern
                >
77          <pattern name="stopdate"  type="string" >^\s*\(end\)\s*(.*)\s+$
                jube_pat_int</pattern>
78      </patternset>
79  </jube>
```

Listing A.10: ior.run.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <jube>
3    <include-path>
4      <path>/usr/local/deep/deep-est_benchmarks/synthetic/ior</path>
5      <path>/usr/local/deep/platform/jureca</path>
6    </include-path>
7
8    <benchmark name="ior" outpath="ior_bench_run">
9      <comment>Input Output Randomizer benchmark</comment>
10
11     <fileset name="iorExec">
12       <copy>/usr/local/deep/sources/synthetic/ior/src/ior</copy>
13     </fileset>
14
15     <parameterset name="iorParameter" init_with="ior_specs.xml">
16       <parameter name="verbose" type="int" >5</parameter>
17       <parameter name="api" type="string" >POSIX,MPIIO</parameter>
18       <parameter name="filePerProc" type="int" >0</parameter>
19       <parameter name="blockSize" type="string" >8m,16m,32m,64m,128m,256m,512m,1
```

```
            GB</parameter>
20        <parameter name="transferSize" type="string" >2m,4m,8m</parameter>
21        <parameter name="collective" type="int" >0,1</parameter>
22        <parameter name="fsync" type="int" >0,1</parameter>
23      </parameterset>
24
25      <parameterset name="systemParameter" init_with="platform.xml">
26        <parameter name="nodes" type="init">1,2,3,4,5</parameter>
27        <parameter name="taskspernode" type="init">24</parameter>
28          <parameter name="testdir" type="string" >/sdv-work/slts13</parameter>
29      </parameterset>
30
31      <parameterset name="executeset" init_with="platform.xml">
32        <parameter name="args_starter">-n $tasks</parameter>
33      </parameterset>
34
35      <substituteset name="executesub" init_with="platform.xml">
36        <!--iofile in="/homea/slts/slts13/deep/deep-est_benchmarks/jureca/submit.
              job.in" out="submit.job" /-->
37        <!--sub source='#NOTIFY_EMAIL#' dest="k.kulkarni@fz-juelich.de"/-->
38        <sub source='#PREPROCESS#' dest="module purge; module load intel
              parastation; date +'(start) %F %T (%s)'"/>
39        <sub source='#POSTPROCESS#' dest="date +'(end) %F %T (%s)'"/>
40        <sub source='#EXECUTABLE#' dest="./ior"/>
41        <sub source='#QUEUE#' dest="sdv"/>
42        <sub source='#ARGS_EXECUTABLE#' dest="-f ior_input.cfg"/>
43        <sub source='#MEASUREMENT#' dest="time"/>
44      <sub source="#NOTIFICATION_TYPE#" dest="all"/>
45      <sub source="#NODES#" dest="$nodes"/>
46      <sub source="#NCPUS#" dest="$taskspernode"/>
47      <sub source="#TASKS#" dest="$tasks"/>
48      <sub source="#NTHREADS#" dest="1"/>
49       </substituteset>
50
51       <substituteset name="iorInputSub" init_with="ior_specs.xml">
52       </substituteset>
53
54       <step name="run_ior" shared="shared">
55         <!--use>SubmitScript</use-->
56         <use from="platform.xml">jobfiles</use>
57         <use from="platform.xml">chainsub</use>
58         <use from="platform.xml">chainfiles</use>
59         <use from="ior_specs.xml">iorInputFile</use>
60         <use>IorExec</use> <!-- use existing parameterset -->
61         <use>iorParameter</use>
62         <use>iorInputSub</use>
63         <use>executeset</use> <!-- use existing parameterset -->
64         <use>executesub</use> <!-- use existing parameterset -->
65         <use>systemParameter</use> <!-- use existing parameterset -->
66         <do>grep -v '^#' ior_input.out> ior_input.cfg</do>
67         <do done_file="ready">$chainjob_script ./shared/jobid $submit_script</do>
68         <!--do done_file="ready">sbatch submit.job</do-->
69       </step>
70
71       <analyzer name="analyse" >
72         <use from="ior_specs.xml">pattern</use>
73         <analyse step="run_ior">
```

```
74          <file>job.out</file>
75        </analyse>
76      </analyzer>
77
78      <result>
79
80        <use>analyse</use>
81
82        <table name="result" style="pretty" sort="nodes,taskspernode">
83          <column>jube_wp_id</column>
84      <column>API</column>
85      <column>Access</column>
86      <column>fsync</column>
87      <column>testdir</column>
88      <column>Aggregatesize</column>
89      <column>WRbwmax_last</column>
90      <column>RDbwmax_last</column>
91          <column>starttime</column>
92          <column>stoptime</column>
93        </table>
94
95        <table name="result-csv" style="csv" sort="nodes,taskspernode">
96          <column>jube_wp_id</column>
97      <column>API</column>
98      <column>Access</column>
99      <column>fsync</column>
100     <column>testdir</column>
101     <column>Aggregatesize</column>
102     <column>WRbwmax_last</column>
103     <column>RDbwmax_last</column>
104         <column>starttime</column>
105         <column>stoptime</column>
106       </table>
107
108       <table name="plot" style="csv" sort="nodes,taskspernode">
109         <column>testdir</column>
110         <column>WRbwmax_last</column>
111         <column>RDbwmax_last</column>
112       </table>
113
114     </result>
115
116   </benchmark>
117 </jube>
```

Listing A.11: ior.specs.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <jube>
3   <parameterset name="iorParameter">
4     <parameter name="api" type="string"                    >POSIX</parameter>
5     <parameter name="refNum" type="int"                    >9999</parameter>
6     <parameter name="testFile" type="string"               >testFile</parameter>
7     <parameter name="repetitions" type="int"               >5</parameter>
8     <parameter name="multiFile" type="int"                 >0</parameter>
9     <parameter name="reorderTasksConstant" type="int"      >1</parameter>
10    <parameter name="taskPerNodeOffset" type="int"         >1</parameter>
```

```
11      <parameter name="reorderTasksRandom" type="int"       >0</parameter>
12      <parameter name="reorderTasksRandomSeed" type="int"  >0</parameter>
13      <parameter name="quitOnError" type="int"              >0</parameter>
14      <parameter name="numTasks" type="int"                 >0</parameter>
15      <parameter name="interTestDelay" type="int"           >0</parameter>
16      <parameter name="outlierThreshold" type="int"         >0</parameter>
17      <parameter name="intraTestBarriers" type="int"        >1</parameter>
18      <parameter name="uniqueDir" type="int"                >0</parameter>
19      <parameter name="writeFile" type="int"                >1</parameter>
20      <parameter name="readFile" type="int"                 >1</parameter>
21      <parameter name="filePerProc" type="int"              >1</parameter>
22      <parameter name="checkWrite" type="int"               >0</parameter>
23      <parameter name="checkRead" type="int"                >0</parameter>
24      <parameter name="keepFile" type="int"                 >0</parameter>
25      <parameter name="keepFileWithError" type="int"        >0</parameter>
26      <parameter name="useExistingTestFile" type="int"      >0</parameter>
27      <parameter name="segmentCount" type="int"             >1</parameter>
28      <parameter name="blockSize" type="string"             >4MB</parameter>
29      <parameter name="transferSize" type="string"          >1MB</parameter>
30      <parameter name="verbose" type="int"                  >0</parameter>
31      <parameter name="storeFileOffset" type="int"          >0</parameter>
32      <parameter name="memoryPerNode" type="string"         >1%</parameter>
33      <parameter name="maxTimeDuration" type="int"          >0</parameter>
34      <parameter name="deadlineForStonewalling" type="int" >0</parameter>
35      <parameter name="randomOffset" type="int"             >0</parameter>
36      <parameter name="summaryAlways" type="int"            >1</parameter>
37      <parameter name="useO_DIRECT" type="int"              >0</parameter>
38      <parameter name="singleXferAttempt" type="int"        >0</parameter>
39      <parameter name="fsyncPerWrite" type="int"            >0</parameter>
40      <parameter name="fsync" type="int"                    >0</parameter>
41      <parameter name="preallocate" type="int"              >0</parameter>
42      <parameter name="useFileView" type="int"              >0</parameter>
43      <parameter name="useSharedFilePointer" type="int"     >0</parameter>
44      <parameter name="useStridedDatatype" type="int"       >0</parameter>
45      <parameter name="collective" type="int"               >0</parameter>
46      <parameter name="showHints" type="int"                >0</parameter>
47      <parameter name="setlustreopt" type="int"             >0</parameter>
48      <parameter name="lustretag" mode="python" type="string">' ' if $setlustreopt
            else '#'</parameter>
49      <parameter name="lustreStripeCount" type="int"        >0</parameter>
50      <parameter name="lustreStripeSize" type="int"         >0</parameter>
51      <parameter name="lustreStartOST" type="int"           >0</parameter>
52      <parameter name="lustreIgnoreLocks" type="int"        >0</parameter>
53      <parameter name="setgpfsopt" type="int"               >0</parameter>
54      <parameter name="gpfstag" mode="python" type="string">' ' if $setgpfsopt
            else '#'</parameter>
55      <parameter name="gpfsHintAccess" type="int"           >0</parameter>
56      <parameter name="gpfsReleaseToken" type="int"         >0</parameter>
57    </parameterset>
58
59    <fileset name="iorInputFile">
60      <copy>ior_input.in</copy>
61    </fileset>
62
63    <substituteset name="iorInputSub">
64      <iofile in=" ior_input.in" out="ior_input.out" />
65      <sub source="#api#" dest="$api" />
```

```
66      <sub source="#refNum#" dest="$refNum" />
67      <sub source="#testFile#" dest="$testdir/ior.dat" />
68      <sub source="#repetitions#" dest="$repetitions" />
69      <sub source="#multiFile#" dest="$multiFile" />
70      <sub source="#reorderTasksConstant#" dest="$reorderTasksConstant" />
71      <sub source="#taskPerNodeOffset#" dest="$taskPerNodeOffset" />
72      <sub source="#reorderTasksRandom#" dest="$reorderTasksRandom" />
73      <sub source="#reorderTasksRandomSeed#" dest="$reorderTasksRandomSeed" />
74      <sub source="#quitOnError#" dest="$quitOnError" />
75      <sub source="#numTasks#" dest="$numTasks" />
76      <sub source="#interTestDelay#" dest="$interTestDelay" />
77      <sub source="#outlierThreshold#" dest="$outlierThreshold" />
78      <sub source="#intraTestBarriers#" dest="$intraTestBarriers" />
79      <sub source="#uniqueDir#" dest="$uniqueDir" />
80      <sub source="#writeFile#" dest="$writeFile" />
81      <sub source="#readFile#" dest="$readFile" />
82      <sub source="#filePerProc#" dest="$filePerProc" />
83      <sub source="#checkWrite#" dest="$checkWrite" />
84      <sub source="#checkRead#" dest="$checkRead" />
85      <sub source="#keepFile#" dest="$keepFile" />
86      <sub source="#keepFileWithError#" dest="$keepFileWithError" />
87      <sub source="#useExistingTestFile#" dest="$useExistingTestFile" />
88      <sub source="#segmentCount#" dest="$segmentCount" />
89      <sub source="#blockSize#" dest="$blockSize" />
90      <sub source="#transferSize#" dest="$transferSize" />
91      <sub source="#verbose#" dest="$verbose" />
92      <sub source="#storeFileOffset#" dest="$storeFileOffset" />
93      <sub source="#memoryPerNode#" dest="$memoryPerNode" />
94      <sub source="#maxTimeDuration#" dest="$maxTimeDuration" />
95      <sub source="#deadlineForStonewalling#" dest="$deadlineForStonewalling" />
96      <sub source="#randomOffset#" dest="$randomOffset" />
97      <sub source="#summaryAlways#" dest="$summaryAlways" />
98      <sub source="#useO_DIRECT#" dest="$useO_DIRECT" />
99      <sub source="#singleXferAttempt#" dest="$singleXferAttempt" />
100     <sub source="#fsyncPerWrite#" dest="$fsyncPerWrite" />
101     <sub source="#fsync#" dest="$fsync" />
102     <sub source="#preallocate#" dest="$preallocate" />
103     <sub source="#useFileView#" dest="$useFileView" />
104     <sub source="#useSharedFilePointer#" dest="$useSharedFilePointer" />
105     <sub source="#useStridedDatatype#" dest="$useStridedDatatype" />
106     <sub source="#collective#" dest="$collective" />
107     <sub source="#showHints#" dest="$showHints" />
108     <sub source="#lustre#" mode="python" dest="$lustretag" />
109     <sub source="#lustreStripeCount#" dest="$lustreStripeCount" />
110     <sub source="#lustreStripeSize#" dest="$lustreStripeSize" />
111     <sub source="#lustreStartOST#" dest="$lustreStartOST" />
112     <sub source="#lustreIgnoreLocks#" dest="$lustreIgnoreLocks" />
113     <sub source="#gpfs#" mode="python" dest="$gpfstag" />
114     <sub source="#gpfsHintAccess#" dest="$gpfsHintAccess" />
115     <sub source="#gpfsReleaseToken#" dest="$gpfsReleaseToken" />
116   </substituteset>
117
118   <patternset name="pattern">
119     <pattern name="API"            unit=""    type="string">\s*api\s+=\s+$
          jube_pat_wrd\s*</pattern>
120     <pattern name="Testfn"         unit=""    type="string">\s*test filename\s
          +=\s+$jube_pat_wrd\s*</pattern>
```

```
121     <pattern name="Access"          unit=""    type="string">\s*access\s+=\s+([\w
            -]+)\s*</pattern>
122     <pattern name="Fileorder"       unit=""    type="string">\s*ordering in a
            file\s*=\s*$jube_pat_wrd offsets\s*</pattern>
123
124     <pattern name="Collective"      unit=""    type="string">\s*collective\s+=\s
            +$jube_pat_wrd\s*</pattern>
125
126     <pattern name="Taskoffsettype" unit=""     type="string">\s*ordering inter
            file\s*=\s*$jube_pat_wrd\s+task offsets\s+=\s*$jube_pat_nint\s*</pattern
            >
127     <pattern name="Taskoffset"      unit=""    type="int"   >\s*ordering inter
            file\s*=\s*$jube_pat_nwrd\s+task offsets\s+=\s*$jube_pat_int\s*</pattern
            >
128
129     <pattern name="Totalclients"    unit=""    type="int"   >\s+clients\s+=\s+$
            jube_pat_int \($jube_pat_nint per node\)\s*</pattern>
130     <pattern name="Clientspernode" unit=""     type="int"   >\s+clients\s+=\s+$
            jube_pat_nint \($jube_pat_int per node\)\s*</pattern>
131     <pattern name="Memorypernode"   unit="MiB" type="float" >\s*memoryPerNode\s
            +=\s+$jube_pat_fp\s*</pattern>
132     <pattern name="Repetitions"     unit=""    type="int"   >\s*repetitions\s+=\s
            +$jube_pat_int\s*</pattern>
133     <pattern name="Xfersize"        unit="MiB" type="string">\s*xfersize\s+=\s+($
            jube_pat_nfp $jube_pat_nwrd)\s*</pattern>
134 <!--<pattern name="Xfersize"        unit="MiB" type="string">\s*xfersize\s+=\s+($
        jube_pat_nfp)\s*</pattern> -->
135     <pattern name="Blocksize"       unit="MiB" type="string">\s*blocksize\s+=\s
            +($jube_pat_fp $jube_pat_nwrd)\s*</pattern>
136     <pattern name="Aggregatesize"   unit="GiB" type="string">\s*aggregate
            filesize\s+=\s+($jube_pat_nfp $jube_pat_nwrd)\s*</pattern>
137
138     <pattern name="WRbw"     unit="MiB" type="float">write\s+(?:$jube_pat_nfp\s+)
            {0}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){6}0</pattern>
139     <pattern name="WRblock" unit="KiB" type="float">write\s+(?:$jube_pat_nfp\s+)
            {1}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){5}0</pattern>
140     <pattern name="WRxfer"  unit="KiB" type="float">write\s+(?:$jube_pat_nfp\s+)
            {2}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){4}0</pattern>
141     <pattern name="WRopen"  unit="s"   type="float">write\s+(?:$jube_pat_nfp\s+)
            {3}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){3}0</pattern>
142     <pattern name="WRwr"    unit="s"   type="float">write\s+(?:$jube_pat_nfp\s+)
            {4}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){2}0</pattern>
143     <pattern name="WRclose" unit="s"   type="float">write\s+(?:$jube_pat_nfp\s+)
            {5}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){1}0</pattern>
144     <pattern name="WRtotal" unit="s"   type="float">write\s+(?:$jube_pat_nfp\s+)
            {6}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){0}0</pattern>
145
146     <pattern name="RDbw"     unit="MiB" type="float">read\s+(?:$jube_pat_nfp\s+)
            {0}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){6}0</pattern>
147     <pattern name="RDblock" unit="KiB" type="float">read\s+(?:$jube_pat_nfp\s+)
            {1}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){5}0</pattern>
148     <pattern name="RDxfer"  unit="KiB" type="float">read\s+(?:$jube_pat_nfp\s+)
            {2}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){4}0</pattern>
149     <pattern name="RDopen"  unit="s"   type="float">read\s+(?:$jube_pat_nfp\s+)
            {3}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){3}0</pattern>
150     <pattern name="RDrd"    unit="s"   type="float">read\s+(?:$jube_pat_nfp\s+)
            {4}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){2}0</pattern>
```

```
151     <pattern name="RDclose" unit="s" type="float">read\s+(?:$jube_pat_nfp\s+)
            {5}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){1}0</pattern>
152     <pattern name="RDtotal" unit="s" type="float">read\s+(?:$jube_pat_nfp\s+)
            {6}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){0}0</pattern>
153
154     <pattern name="REMremove" unit="s" type="float">remove\s+-\s+-\s+-\s+-\s
            +-\s+-\s+$jube_pat_fp\s+0</pattern>
155
156     <pattern name="WRbwmax" unit="MiB" type="float" reduce="last">write\s+(?:
            $jube_pat_nfp\s+){0}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){4}(?:$
            jube_pat_nint\s+){13}$jube_pat_nwrd\s+$jube_pat_nint</pattern>
157     <pattern name="WRbwmin" unit="MiB" type="float" reduce="last">write\s+(?:
            $jube_pat_nfp\s+){1}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){3}(?:$
            jube_pat_nint\s+){13}$jube_pat_nwrd\s+$jube_pat_nint</pattern>
158     <pattern name="WRbwmean" unit="MiB" type="float" reduce="last">write\s+(?:
            $jube_pat_nfp\s+){2}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){2}(?:$
            jube_pat_nint\s+){13}$jube_pat_nwrd\s+$jube_pat_nint</pattern>
159     <pattern name="WRbstddev" unit="" type="float" reduce="last">write\s+(?:
            $jube_pat_nfp\s+){3}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){1}(?:$
            jube_pat_nint\s+){13}$jube_pat_nwrd\s+$jube_pat_nint</pattern>
160     <pattern name="WRbmeans" unit="s" type="float" reduce="last">write\s+(?:
            $jube_pat_nfp\s+){4}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){0}(?:$
            jube_pat_nint\s+){13}$jube_pat_nwrd\s+$jube_pat_nint</pattern>
161     <pattern name="WRtpn" unit="" type="int" reduce="last">write\s+(?:
            $jube_pat_nfp\s+){5}(?:$jube_pat_nint\s+){2}$jube_pat_int\s+(?:$
            jube_pat_nint\s+){10}$jube_pat_nwrd\s+$jube_pat_nint</pattern>
162     <pattern name="WRapi" unit="" type="string" reduce="last">write\s+(?:
            $jube_pat_nfp\s+){5}(?:$jube_pat_nint\s+){13}$jube_pat_wrd\s+$
            jube_pat_nint</pattern>
163
164     <pattern name="RDbwmax" unit="MiB" type="float" reduce="last">read\s+(?:$
            jube_pat_nfp\s+){0}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){4}(?:$
            jube_pat_nint\s+){13}$jube_pat_nwrd\s+$jube_pat_nint</pattern>
165     <pattern name="RDbwmin" unit="MiB" type="float" reduce="last">read\s+(?:$
            jube_pat_nfp\s+){1}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){3}(?:$
            jube_pat_nint\s+){13}$jube_pat_nwrd\s+$jube_pat_nint</pattern>
166     <pattern name="RDbwmean" unit="MiB" type="float" reduce="last">read\s+(?:$
            jube_pat_nfp\s+){2}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){2}(?:$
            jube_pat_nint\s+){13}$jube_pat_nwrd\s+$jube_pat_nint</pattern>
167     <pattern name="RDbstddev" unit="" type="float" reduce="last">read\s+(?:$
            jube_pat_nfp\s+){3}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){1}(?:$
            jube_pat_nint\s+){13}$jube_pat_nwrd\s+$jube_pat_nint</pattern>
168     <pattern name="RDbmeans" unit="s" type="float" reduce="last">read\s+(?:$
            jube_pat_nfp\s+){4}$jube_pat_fp\s+(?:$jube_pat_nfp\s+){0}(?:$
            jube_pat_nint\s+){13}$jube_pat_nwrd\s+$jube_pat_nint</pattern>
169     <pattern name="RDtpn" unit="" type="int" reduce="last">read\s+(?:$
            jube_pat_nfp\s+){5}(?:$jube_pat_nint\s+){2}$jube_pat_int\s+(?:$
            jube_pat_nint\s+){10}$jube_pat_nwrd\s+$jube_pat_nint</pattern>
170     <pattern name="RDapi" unit="" type="string" reduce="last">read\s+(?:$
            jube_pat_nfp\s+){5}(?:$jube_pat_nint\s+){13}$jube_pat_wrd\s+$
            jube_pat_nint</pattern>
171
172     <pattern name="starttime" type="string" >^\s*\(start\)\s*(.*)\s+\(</pattern>
173     <pattern name="startdate" type="string" >^\s*\(start\)\s*(.*)\s+$
            jube_pat_int</pattern>
174     <pattern name="stoptime" type="string" >^\s*\(finished\)\s*(.*)\s+\(</
            pattern>
```

```
175     <pattern name="stopdate"  type="string" >^\s*\(finished\)\s*(.*)\s+$
            jube_pat_int</pattern>
176   </patternset>
177
178 </jube>
```

Listing A.12: h5perf.run.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <jube>
3    <include-path>
4     <path>/usr/local/deep/deep-est_benchmarks/synthetic/h5perf</path>
5     <path>/usr/local/deep/platform/jureca</path>
6    </include-path>
7
8    <benchmark name="h5perf" outpath="h5perf_bench_run">
9      <comment>Parallel HDF5 performance benchmark</comment>
10
11     <parameterset name="h5perfParameter">
12       <parameter name="api" type="string">phdf5</parameter>
13       <parameter name="nBytes" type="string" >8M,16M,32M,64M,128M,256M,512M,1G</
             parameter>
14       <parameter name="iterations" type="string">10</parameter>
15       <parameter name="collective" type="string">-C,</parameter>
16       <parameter name="chunked" type="string">-c,</parameter>
17     </parameterset>
18
19     <parameterset name="systemParameter" init_with="platform.xml">
20       <parameter name="nodes" type="init">1,2,3,4,5</parameter>
21       <parameter name="taskspernode" type="init">24</parameter>
22     </parameterset>
23
24     <parameterset name="executeset" init_with="platform.xml">
25       <parameter name="args_starter">-n $tasks</parameter>
26     </parameterset>
27
28     <substituteset name="executesub" init_with="platform.xml">
29       <sub source='#PREPROCESS#' dest="module purge; module load intel
             parastation hdf5/1.8.20-parallel; date +'(start) %F %T (%s)'"/>
30       <sub source='#POSTPROCESS#' dest="date +'(end) %F %T (%s)'"/>
31       <sub source='#EXECUTABLE#' dest="h5perf"/>
32       <sub source='#QUEUE#' dest="sdv"/>
33       <sub source='#ARGS_EXECUTABLE#' dest="--api=$api -e $nBytes -p $$
             SLURM_NTASKS -i $iterations $collective $chunked"/>
34       <sub source='#MEASUREMENT#' dest="time"/>
35     <sub source="#NOTIFICATION_TYPE#" dest="all"/>
36     <sub source="#NODES#" dest="$nodes"/>
37     <sub source="#NCPUS#" dest="$taskspernode"/>
38     <sub source="#TASKS#" dest="$tasks"/>
39     <sub source="#NTHREADS#" dest="1"/>
40     <sub source="#TIME_LIMIT#" dest="01:00:00"/>
41      </substituteset>
42
43     <step name="run_h5perf" shared="shared">
44       <!--use>SubmitScript</use-->
45       <use from="platform.xml">jobfiles</use>
46       <use from="platform.xml">chainsub</use>
```

```
47        <use from="platform.xml">chainfiles</use>
48        <use>h5perfParameter</use>
49        <use>executeset</use> <!-- use existing parameterset -->
50        <use>executesub</use> <!-- use existing parameterset -->
51        <use>systemParameter</use> <!-- use existing parameterset -->
52        <do done_file="ready">$chainjob_script ./shared/jobid $submit_script</do>
53        <!--do done_file="ready">sbatch submit.job</do-->
54      </step>
55
56      <patternset name="pattern">
57        <pattern name="nProcs" type="int">Number of processes=$jube_pat_int</
              pattern>
58    <pattern name="numBytes" type="string">Number of bytes per process per
          dataset=$jube_pat_wrd</pattern>
59    <pattern name="IO_Method" type="string">I/O Method for MPI and HDF5=$
          jube_pat_wrd</pattern>
60    <pattern name="Data_Storage_Method" type="string">Data storage method in HDF5
          =$jube_pat_wrd</pattern>
61    <!--pattern name="Write_average" tyep="string" mode="shell">sed -n 49p ${
          jube_benchmark_rundir}/${jube_benchmark_padid}_run_h5perf/work/job.out |
          cut -c37- | sed "s/ MB\/s//"</pattern-->
62    <pattern name="Write_average" tyep="string" mode="shell" unit="MB/s">sed -n
          49p ${jube_wp_abspath}/job.out | cut -c37- | sed "s/ MB\/s//"</pattern>
63    <pattern name="Read_average" tyep="string" mode="shell" unit="MB/s">sed -n 57
          p ${jube_wp_abspath}/job.out | cut -c37- | sed "s/ MB\/s//"</pattern>
64      </patternset>
65
66      <analyzer name="analyse" reduce="false">
67        <use>pattern</use>
68    <analyse step="run_h5perf">
69      <file>job.out</file>
70    </analyse>
71      </analyzer>
72
73      <result>
74        <use>analyse</use>
75    <table name="result" style="pretty">
76        <column>jube_wp_id</column>
77        <column>nProcs</column>
78        <column>nBytes</column>
79        <column>IO_Method</column>
80        <column>Data_Storage_Method</column>
81        <column>Write_average</column>
82        <column>Read_average</column>
83    </table>
84    <table name="result-csv" style="csv">
85        <column>jube_wp_id</column>
86        <column>nProcs</column>
87        <column>nBytes</column>
88        <column>IO_Method</column>
89        <column>Data_Storage_Method</column>
90        <column>Write_average</column>
91        <column>Read_average</column>
92    </table>
93      </result>
94    </benchmark>
95 </jube>
```

Listing A.13: UoI-piSVM.compile.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>
    <dos>
      <do>echo $$PWD</do>
      <do>tar -xzf /usr/local/deep/sources/applications/UoI-piSVM/UoI-piSVM.
          latest.tar.gz</do> <!-- copy source code -->
      <do>module purge</do> <!-- load default modules -->
      <do>module load intel parastation</do> <!-- load default modules -->
      <do>cd UoI-piSVM/source/; make clean; make</do> <!-- copy source code -->
    </dos>
</jube>
```

Listing A.14: UoI-piSVM.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>
  <include-path>
   <path>/usr/local/deep/deep-est_benchmarks/applications/UoI-piSVM</path>
   <path>/usr/local/deep/platform/jureca</path>
  </include-path>
  <benchmark name="UoI-piSVM" outpath="UoI-piSVM_bench_run">
    <comment>UoI-piSVM benchmark</comment>

    <step name="compile_UoI-piSVM">
      <include from="UoI-piSVM.compile.xml" path="dos/do"/>
    </step>

    <parameterset name="systemParameter" init_with="platform.xml">
      <parameter name="nodes" type="init">2</parameter>
      <parameter name="taskspernode" type="init">24</parameter>
    </parameterset>

    <parameterset name="executeset" init_with="platform.xml">
      <parameter name="args_starter">-n $tasks</parameter>
    </parameterset>

    <substituteset name="executesub" init_with="platform.xml">
      <iofile in="submit.job.in" out="submit.job" />
      <sub source='#PREPROCESS#' dest="module purge; module load intel
          parastation extoll; date +'(start) %F %T (%s)'; TRAINDATA=/usr/local/
          deep/sources/applications/UoI-piSVM/input/indian_processed_training.el
          "/>
      <sub source='#POSTPROCESS#' dest="date +'(end) %F %T (%s)'"/>
      <sub source='#EXECUTABLE#' dest="./compile_UoI-piSVM/UoI-piSVM/source/
          pisvm-train"/>
      <sub source='#QUEUE#' dest="sdv"/>
      <sub source='#ARGS_EXECUTABLE#' dest="-D -o 1024 -q 512 -c 100 -g 8 -t 2 -
          m 1024 -s 0 $$TRAINDATA "/>
      <sub source='#MEASUREMENT#' dest="time"/>
   <sub source="#NOTIFICATION_TYPE#" dest="all"/>
   <sub source="#NODES#" dest="$nodes"/>
   <sub source="#NCPUS#" dest="$taskspernode"/>
   <sub source="#TASKS#" dest="$tasks"/>
   <sub source="#NTHREADS#" dest="1"/>
    </substituteset>

```

```
38    <step name="train_UoI-piSVM" depend="compile_UoI-piSVM" shared="shared">
39      <use from="platform.xml">jobfiles</use>
40      <use from="platform.xml">chainsub</use>
41      <use from="platform.xml">chainfiles</use>
42      <use>executeset</use> <!-- use existing parameterset -->
43      <use>executesub</use> <!-- use existing parameterset -->
44      <use>systemParameter</use> <!-- use existing parameterset -->
45      <do done_file="ready">$chainjob_script ./shared/jobid $submit_script</do>
46    </step>
47
48    <parameterset name="predict_piSVM_params">
49      <parameter name="test_data">/usr/local/deep/sources/applications/UoI-piSVM
            /input/indian_processed_test.el</parameter>
50      <parameter name="model_data">./train_UoI-piSVM/indian_processed_training.
            el.model</parameter>
51    <parameter name="result_file">prediction_results.txt</parameter>
52    </parameterset>
53
54    <substituteset name="executesubPredict">
55      <iofile in="/usr/local/deep/platform/jureca/submit.job.in" out="submit.job
            " />
56      <sub source='#PREPROCESS#' dest="module purge; module load intel
            parastation; date +'(start) %F %T (%s)'"/>
57      <sub source='#POSTPROCESS#' dest="date +'(end) %F %T (%s)'"/>
58      <sub source='#EXECUTABLE#' dest="./compile_UoI-piSVM/UoI-piSVM/source/
            pisvm-predict"/>
59      <sub source='#QUEUE#' dest="sdv"/>
60      <sub source='#ARGS_EXECUTABLE#' dest="$test_data $model_data $result_file"
            />
61      <sub source='#MEASUREMENT#' dest="time"/>
62    <sub source="#NOTIFICATION_TYPE#" dest="all"/>
63    <sub source="#NODES#" dest="$nodes"/>
64    <sub source="#NCPUS#" dest="$taskspernode"/>
65    <sub source="#TASKS#" dest="$tasks"/>
66    <sub source="#NTHREADS#" dest="1"/>
67    <sub source="#STARTER#" dest="srun"/>
68    <sub source="#ARGS_STARTER#" dest="$args_starter"/>
69    <sub source="#BENCHNAME#" dest="UoI-piSVM_predict"/>
70    <sub source="#NOTIFY_EMAIL#" dest=""/>
71    <sub source="#TIME_LIMIT#" dest="00:30:00"/>
72    <sub source="#STDOUTLOGFILE#" dest="job.out"/>
73    <sub source="#STDERRLOGFILE#" dest="job.err"/>
74    <sub source="#FLAG#" dest="touch ready"/>
75    </substituteset>
76
77    <step name="predict_UoI-piSVM" depend="compile_UoI-piSVM,train_UoI-piSVM"
          shared="shared">
78      <use from="platform.xml">jobfiles</use>
79      <use from="platform.xml">chainsub</use>
80      <use from="platform.xml">chainfiles</use>
81      <use>predict_piSVM_params</use> <!-- use existing parameterset -->
82      <use>executeset</use> <!-- use existing parameterset -->
83      <use>systemParameter</use> <!-- use existing parameterset -->
84      <use>executesubPredict</use> <!-- use existing parameterset -->
85      <do done_file="ready">$chainjob_script ./shared/jobid $submit_script</do>
86    </step>
87
```

```
88      <patternset name="pattern">
89        <pattern mode="pattern" name="Accuracy" type="float" unit="%">Accuracy = $
              jube_pat_fp</pattern>
90        <pattern mode="pattern" name="Mean_squared_error" type="float">Mean
              squared error = $jube_pat_fp</pattern>
91        <pattern mode="pattern" name="Squared_correlation_coefficient" type="float
              ">Squared correlation coefficient = $jube_pat_fp</pattern>
92      </patternset>
93
94      <analyzer name="analyse" reduce="false">
95        <use>pattern</use>
96     <analyse step="predict_UoI-piSVM">
97       <file>job.out</file>
98     </analyse>
99      </analyzer>
100
101     <result>
102       <use>analyse</use>
103     <table name="result" style="pretty">
104       <column>Accuracy</column>
105       <column>Mean_squared_error</column>
106       <column>Squared_correlation_coefficient</column>
107     </table>
108     <table name="result-csv" style="csv">
109       <column>Accuracy</column>
110       <column>Mean_squared_error</column>
111       <column>Squared_correlation_coefficient</column>
112     </table>
113      </result>
114
115   </benchmark>
116 </jube>
```

Listing A.15: UoI-HPDBSCAN.hybrid.large.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <jube xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xsi:noNamespaceSchemaLocation="jube2.xsd">
4
5    <!-- UoI-HPDBSCAN -->
6
7    <include-path>
8      <path>/usr/local/jube2/platform/deep</path>
9    </include-path>
10
11   <benchmark name="UoI-HPDBSCAN" outpath="./benchmarks">
12
13     <fileset name="sources">
14       <copy>/usr/local/deep/sources/applications/UoI-HPDBSCAN/UoI-HPDBSCAN.
             latest.tar.gz</copy>
15       <prepare>tar -xzf UoI-HPDBSCAN.latest.tar.gz</prepare>
16     </fileset>
17
18     <fileset name="UoI-HPDBSCAN-exec">
19       <!-- <copy>executable/$flavor/dbscan</copy> -->
20       <copy>input/$dataset</copy>
21     </fileset>
```

```
22
23      <parameterset name="systemParameter" init_with="UoI-HPDBSCAN.params.xml">
24        <!-- i parameter selection: 0 for selecting flavor=hybrid; 1 for selecting
               flavor=openmp; 0,1 for selecting both -->
25        <parameter name="i">0</parameter>
26        <parameter name="nodes">8</parameter>
27        <parameter name="taskspernode">1</parameter>
28        <parameter name="threadspertask">16</parameter>
29        <parameter name="mail">c.manzano@fz-juelich.de</parameter>
30        <parameter name="timelimit">00:20:00</parameter>
31        <parameter name="jobname">HPDBSCAN-hybrid-large</parameter>
32      </parameterset>
33
34      <parameterset name="UoI-HPDBSCANParameter" init_with="UoI-HPDBSCAN.params.
            xml">
35        <parameter name="m">20</parameter>
36        <parameter name="e">30</parameter>
37        <parameter name="dataset">bremen.h5.h5</parameter>
38      </parameterset>
39
40      <parameterset name="executeset" init_with="UoI-HPDBSCAN.params.xml">
41      </parameterset>
42
43      <substituteset name="executesub" init_with="UoI-HPDBSCAN.params.xml">
44        <sub source="#BENCHNAME#" dest="$jobname"/>
45        <sub source='#PREPROCESS#' dest="date +'(start) %F %T (%s)'"/>
46        <sub source='#POSTPROCESS#' dest="date +'(finished) %F %T (%s)'"/>
47        <sub source='#EXECUTABLE#' dest="dbscan"/>
48        <sub source='#ARGS_EXECUTABLE#' dest="-m $m -e $e $dataset"/>
49      </substituteset>
50
51      <step name="compile">
52        <use>sources</use>
53        <use>systemParameter</use>
54        <do>. /etc/profile.d/modules.sh; module purge; module load $modules;
              module load extoll; cd UoI-HPDBSCAN/source/$flavor_path; make $
              flavor_option</do>
55      </step>
56
57      <step depend="compile" name="execute" shared="shared">
58        <use>systemParameter</use>
59        <use>UoI-HPDBSCANParameter</use>
60        <use>UoI-HPDBSCAN-exec</use>
61        <use>executeset</use>
62        <use>executesub</use>
63        <use from="platform.xml">jobfiles</use>
64        <use from="platform.xml">chainsub</use>
65        <use from="platform.xml">chainfiles</use>
66        <do>cp -p compile/UoI-HPDBSCAN/bin/$flavor/dbscan .</do>
67        <do done_file="ready">$chainjob_script ./shared/jobid $submit_script</do>
68        <!-- <do done_file="ready">$submit $submit_script</do> -->
69      </step>
70
71      <analyzer name="analyse" >
72        <use from="UoI-HPDBSCAN.params.xml">pattern</use>
73        <analyse step="execute">
74          <file>job.out</file>
```

```
75        </analyse>
76      </analyzer>
77
78      <result>
79        <use>analyse</use>
80        <table name="result" style="pretty" sort="jube_wp_id">
81          <column>jube_benchmark_id</column>
82          <column>nodes</column>
83          <column>tasks</column>
84          <column>taskspernode</column>
85          <column>threadspertask</column>
86          <column>flavor</column>
87          <column>m</column>
88          <column>e</column>
89          <column>dataset</column>
90          <column>totaltime</column>
91          <column>starttime</column>
92          <column>stoptime</column>
93        </table>
94      </result>
95
96    </benchmark>
97
98  </jube>
```

Listing A.16: UoI-HPDBSCAN.hybrid.small.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <jube xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xsi:noNamespaceSchemaLocation="jube2.xsd">
4
5    <!-- UoI-HPDBSCAN -->
6
7    <include-path>
8      <path>/usr/local/jube2/platform/deep</path>
9    </include-path>
10
11   <benchmark name="UoI-HPDBSCAN" outpath="./benchmarks">
12
13     <fileset name="sources">
14       <copy>/usr/local/deep/sources/applications/UoI-HPDBSCAN/UoI-HPDBSCAN.
             latest.tar.gz</copy>
15       <prepare>tar -xzf UoI-HPDBSCAN.latest.tar.gz</prepare>
16     </fileset>
17
18     <fileset name="UoI-HPDBSCAN-exec">
19       <!-- <copy>executable/$flavor/dbscan</copy> -->
20       <copy>input/$dataset</copy>
21     </fileset>
22
23     <parameterset name="systemParameter" init_with="UoI-HPDBSCAN.params.xml">
24       <!-- i parameter selection: 0 for selecting flavor=hybrid; 1 for selecting
             flavor=openmp; 0,1 for selecting both -->
25       <parameter name="i">0</parameter>
26       <parameter name="nodes">2</parameter>
27       <parameter name="taskspernode">1</parameter>
28       <parameter name="threadspertask">2</parameter>
```

```
29        <parameter name="mail">c.manzano@fz-juelich.de</parameter>
30        <parameter name="timelimit">00:20:00</parameter>
31        <parameter name="jobname">HPDBSCAN-hybrid-small</parameter>
32      </parameterset>
33
34      <parameterset name="UoI-HPDBSCANParameter" init_with="UoI-HPDBSCAN.params.
            xml">
35        <parameter name="m">20</parameter>
36        <parameter name="e">30</parameter>
37        <parameter name="dataset">bremenSmall.h5.h5</parameter>
38      </parameterset>
39
40      <parameterset name="executeset" init_with="UoI-HPDBSCAN.params.xml">
41      </parameterset>
42
43      <substituteset name="executesub" init_with="UoI-HPDBSCAN.params.xml">
44        <sub source="#BENCHNAME#" dest="$jobname"/>
45        <sub source='#PREPROCESS#' dest="date +'(start) %F %T (%s)'"/>
46        <sub source='#POSTPROCESS#' dest="date +'(finished) %F %T (%s)'"/>
47        <sub source='#EXECUTABLE#' dest="dbscan"/>
48        <sub source='#ARGS_EXECUTABLE#' dest="-m $m -e $e $dataset"/>
49      </substituteset>
50
51      <step name="compile">
52        <use>sources</use>
53        <use>systemParameter</use>
54        <do>. /etc/profile.d/modules.sh; module purge; module load $modules;
              module load extoll; cd UoI-HPDBSCAN/source/$flavor_path; make $
              flavor_option</do>
55      </step>
56
57      <step depend="compile" name="execute" shared="shared">
58        <use>systemParameter</use>
59        <use>UoI-HPDBSCANParameter</use>
60        <use>UoI-HPDBSCAN-exec</use>
61        <use>executeset</use>
62        <use>executesub</use>
63        <use from="platform.xml">jobfiles</use>
64        <use from="platform.xml">chainsub</use>
65        <use from="platform.xml">chainfiles</use>
66        <do>cp -p compile/UoI-HPDBSCAN/bin/$flavor/dbscan .</do>
67        <do done_file="ready">$chainjob_script ./shared/jobid $submit_script</do>
68        <!-- <do done_file="ready">$submit $submit_script</do> -->
69      </step>
70
71      <analyzer name="analyse" >
72        <use from="UoI-HPDBSCAN.params.xml">pattern</use>
73        <analyse step="execute">
74          <file>job.out</file>
75        </analyse>
76      </analyzer>
77
78      <result>
79        <use>analyse</use>
80        <table name="result" style="pretty" sort="jube_wp_id">
81          <column>jube_benchmark_id</column>
82          <column>nodes</column>
```

```
83          <column>tasks</column>
84          <column>taskspernode</column>
85          <column>threadspertask</column>
86          <column>flavor</column>
87          <column>m</column>
88          <column>e</column>
89          <column>dataset</column>
90          <column>totaltime</column>
91          <column>starttime</column>
92          <column>stoptime</column>
93        </table>
94      </result>
95
96    </benchmark>
97
98  </jube>
```

Listing A.17: UoI-HPDBSCAN.params.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <jube xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xsi:noNamespaceSchemaLocation="jube2.xsd">
4
5    <include-path>
6      <path>/usr/local/jube2/platform/deep</path>
7    </include-path>
8
9    <parameterset name="systemParameter" init_with="platform.xml">
10     <parameter name="modules">gcc/5.3.0-64bit hdf5/1.8.20-gcc parastation
           /5.2.0-1</parameter>
11     <parameter name="env">. /etc/profile.d/modules.sh; module purge; module load
            $modules; $jube_wp_envstr</parameter>
12     <!-- i parameter selection: 0 for selecting flavor=hybrid; 1 for selecting
           flavor=openmp; 0,1 for selecting both -->
13     <parameter name="i">0</parameter>
14     <parameter name="flavor" mode="python">["hybrid","openmp"][$i]</parameter>
15     <parameter name="flavor_path" mode="python">["jsc_mpi","jsc_openmp"][$i]</
           parameter>
16     <parameter name="flavor_option" mode="python">["deep",""][$i]</parameter>
17     <parameter name="targetdir">/homeb/zam/manzano/DEEP-EST/benchmarks/
           applications/UoI-HPDBSCAN/executable/$flavor</parameter>
18     <parameter name="queue">sdv</parameter>
19     <parameter name="jobname">HPDBSCAN</parameter>
20     <parameter name="nodes">1</parameter>
21     <parameter name="taskspernode">8</parameter>
22     <parameter name="timelimit">00:20:00</parameter>
23     <parameter name="threadspertask">2</parameter>
24   </parameterset>
25
26   <parameterset name="executeset" init_with="platform.xml">
27   </parameterset>
28
29   <substituteset name="executesub" init_with="platform.xml">
30   </substituteset>
31
32   <parameterset name="UoI-HPDBSCANParameter">
33     <parameter name="m">40</parameter>
```

```
    <parameter name="e">0.0003</parameter>
    <parameter name="dataset">twitterSmall.h5.h5</parameter>
  </parameterset>

  <patternset name="pattern">
    <pattern name="totaltime" unit="sec" type="float">\s*Took:\s+(.*)s\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>

</jube>
```

Listing A.18: xPic-bench-cpu.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>
  <benchmark name="CPU_speedup" outpath="bench_deepcpu">
    <comment>Benchmark of the code xPic on Intel Xeon procesor</comment>

    <!-- Configuration -->
    <parameterset name="param_set">
      <parameter name="submit_cmd"                >sbatch</parameter>
      <parameter name="ready_file"                >ready</parameter>
      <parameter name="job_script"                >job.slurm</parameter>
      <parameter name="job_nodetype"              >cluster</parameter>
      <parameter name="job_name"                  >xPic.cpu</parameter>
      <parameter name="job_err"                   >stderr</parameter>
      <parameter name="job_out"                   >stdout</parameter>
      <parameter name="job_outdir"                >output</parameter>
      <parameter name="job_srcname"               >xPic</parameter>
      <parameter name="job_input"                 >input.inp</parameter>
      <parameter name="job_afinity"               ></parameter>
      <parameter name="job_mpiproc" type="int"    >1</parameter>
      <parameter name="job_nthread" type="int"    >16,8,4,2,1</parameter>
      <parameter name="i"                         >0,1</parameter>
      <parameter name="run_ntcx"    mode="python">[1152,768][$i]</parameter>
      <parameter name="run_nblockx" mode="python">[48,32][$i]</parameter>
      <parameter name="run_niter"   mode="python">[1000,1000][$i]</parameter>
      <parameter name="run_nppc"    type="int"    >1024</parameter>
      <parameter name="extension"                 >intel64</parameter>
      <parameter name="mpiexec"                   >srun</parameter>
      <parameter name="mpiopts"                   ></parameter>
    </parameterset>

    <!-- Files -->
    <fileset name="files">
      <copy>${job_input}</copy>
      <copy>${job_srcname}.${extension}</copy>
      <copy>${job_script}</copy>
    </fileset>

    <!-- Substitute -->
    <substituteset name="subsjob">
      <iofile in="${job_script}" out="${job_script}.run" />
```

```
41        <sub source="#NAME#"         dest="${job_name}.${job_nthread}" />
42        <sub source="#OUTFILE#"      dest="$job_out" />
43        <sub source="#ERRFILE#"      dest="$job_err" />
44        <sub source="#NODETYPE#"     dest="$job_nodetype" />
45        <sub source="#NUMTHREADS#"   dest="$job_nthread" />
46        <sub source="#MPIPROC#"      dest="$job_mpiproc" />
47        <sub source="#MPIEXEC#"      dest="$mpiexec" />
48        <sub source="#MPIOPT#"       dest="$mpiopts" />
49        <sub source="#EXEC#"         dest="./${job_srcname}.${extension}" />
50        <sub source="#AFINITY#"      dest="${job_afinity}" />
51        <sub source="#INPUT#"        dest="${job_input}.run" />
52        <sub source="#READY#"        dest="$ready_file" />
53        <sub source="#OUTDIR#"       dest="$job_outdir" />
54      </substituteset>
55      <substituteset name="subsinput">
56        <iofile in="${job_input}" out="${job_input}.run" />
57        <sub source="#NTCX#"         dest="$run_ntcx" />
58        <sub source="#NBLOCKX#"      dest="$run_nblockx" />
59        <sub source="#NPPC#"         dest="$run_nppc" />
60        <sub source="#NITER#"        dest="$run_niter" />
61        <sub source="#OUTDIR#"       dest="$job_outdir" />
62      </substituteset>
63
64      <!-- Regex pattern -->
65      <patternset name="pattern">
66        <pattern name="number_pat" type="float">Main   loop: $jube_pat_fp s</
              pattern>
67      </patternset>
68
69      <!-- Operation -->
70      <step name="submit">
71        <use>param_set</use>     <!-- use existing parameterset -->
72        <use>files</use>         <!-- use existing fileset -->
73        <use>subsjob</use>       <!-- use existing substituteset -->
74        <use>subsinput</use>     <!-- use existing substituteset -->
75        <do>mkdir ${job_outdir}</do>
76        <do done_file="${ready_file}">${submit_cmd} ${job_script}.run</do>
77      </step>
78
79      <!-- Analyse -->
80      <analyser name="analyse">
81        <use>pattern</use> <!-- use existing patternset -->
82        <analyse step="submit">
83          <file>stdout</file> <!-- file which should be scanned -->
84        </analyse>
85      </analyser>
86
87      <!-- Create result table -->
88      <result>
89        <use>analyse</use> <!-- use existing analyser -->
90        <table name="result" style="pretty" sort="run_ntcx,job_nthread">
91          <column>run_ntcx</column>
92          <column>run_nblockx</column>
93          <column>job_nthread</column>
94          <column>number_pat_max</column>
95          <column>number_pat_min</column>
96          <column>number_pat_avg</column>
```

```
97        </table>
98      </result>
99
100   </benchmark>
101 </jube>
```

Listing A.19: xPic-bench-knl.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <jube>
3    <benchmark name="CPU_speedup" outpath="bench_deepknl">
4      <comment>Benchmark of the code xPic on Intel Xeon Phi - KNL processor</
           comment>
5
6      <!-- Configuration -->
7      <parameterset name="param_set">
8        <parameter name="submit_cmd"                >sbatch</parameter>
9        <parameter name="ready_file"                >ready</parameter>
10       <parameter name="job_script"                >job.slurm</parameter>
11       <parameter name="job_nodetype"              >knl</parameter>
12       <parameter name="job_name"                  >xPic.knl</parameter>
13       <parameter name="job_err"                   >stderr</parameter>
14       <parameter name="job_out"                   >stdout</parameter>
15       <parameter name="job_outdir"                >output</parameter>
16       <parameter name="job_srcname"               >xPic</parameter>
17       <parameter name="job_input"                 >input.inp</parameter>
18       <parameter name="job_afinity"               ></parameter>
19       <parameter name="job_mpiproc" type="int"    >1</parameter>
20       <parameter name="job_nthread" type="int"    >64,32,16,8,4,2,1</parameter>
21       <parameter name="i"                         >0,1</parameter>
22       <parameter name="run_ntcx"    mode="python">[1152,768][$i]</parameter>
23       <parameter name="run_nblockx" mode="python">[128,64][$i]</parameter>
24       <parameter name="run_niter"   mode="python">[1000,1000][$i]</parameter>
25       <parameter name="run_nppc"    type="int"    >1024</parameter>
26       <parameter name="extension"                 >knl</parameter>
27       <parameter name="mpiexec"                   >srun</parameter>
28       <parameter name="mpiopts"                   ></parameter>
29     </parameterset>
30
31     <!-- Files -->
32     <fileset name="files">
33       <copy>${job_input}</copy>
34       <copy>${job_srcname}.${extension}</copy>
35       <copy>${job_script}</copy>
36     </fileset>
37
38     <!-- Substitute -->
39     <substituteset name="subsjob">
40       <iofile in="${job_script}" out="${job_script}.run" />
41       <sub source="#NAME#"       dest="${job_name}.${job_nthread}" />
42       <sub source="#OUTFILE#"    dest="$job_out" />
43       <sub source="#ERRFILE#"    dest="$job_err" />
44       <sub source="#NODETYPE#"   dest="$job_nodetype" />
45       <sub source="#NUMTHREADS#" dest="$job_nthread" />
46       <sub source="#MPIPROC#"    dest="$job_mpiproc" />
47       <sub source="#MPIEXEC#"    dest="numactl -m 1 $mpiexec" />
48       <sub source="#MPIOPT#"     dest="$mpiopts" />
```

```
49        <sub source="#EXEC#"        dest="./${job_srcname}.${extension}" />
50        <sub source="#AFINITY#"     dest="${job_afinity}" />
51        <sub source="#INPUT#"       dest="${job_input}.run" />
52        <sub source="#READY#"       dest="$ready_file" />
53        <sub source="#OUTDIR#"      dest="$job_outdir" />
54      </substituteset>
55      <substituteset name="subsinput">
56        <iofile in="${job_input}" out="${job_input}.run" />
57        <sub source="#NTCX#"        dest="$run_ntcx" />
58        <sub source="#NBLOCKX#"     dest="$run_nblockx" />
59        <sub source="#NPPC#"        dest="$run_nppc" />
60        <sub source="#NITER#"       dest="$run_niter" />
61        <sub source="#OUTDIR#"      dest="$job_outdir" />
62      </substituteset>
63
64      <!-- Regex pattern -->
65      <patternset name="pattern">
66        <pattern name="number_pat" type="float">Main    loop: $jube_pat_fp s</
              pattern>
67      </patternset>
68
69      <!-- Operation -->
70      <step name="submit">
71        <use>param_set</use>     <!-- use existing parameterset -->
72        <use>files</use>         <!-- use existing fileset -->
73        <use>subsjob</use>       <!-- use existing substituteset -->
74        <use>subsinput</use>     <!-- use existing substituteset -->
75        <do>mkdir ${job_outdir}</do>
76        <do done_file="${ready_file}">${submit_cmd} ${job_script}.run</do>
77      </step>
78
79      <!-- Analyse -->
80      <analyser name="analyse">
81        <use>pattern</use> <!-- use existing patternset -->
82        <analyse step="submit">
83          <file>stdout</file> <!-- file which should be scanned -->
84        </analyse>
85      </analyser>
86
87      <!-- Create result table -->
88      <result>
89        <use>analyse</use> <!-- use existing analyser -->
90        <table name="result" style="pretty" sort="run_ntcx,job_nthread">
91          <column>run_ntcx</column>
92          <column>run_nblockx</column>
93          <column>job_nthread</column>
94          <column>number_pat_max</column>
95          <column>number_pat_min</column>
96          <column>number_pat_avg</column>
97        </table>
98      </result>
99
100   </benchmark>
101 </jube>
```

Listing A.20: xPic-bench-sdv.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>
  <benchmark name="CPU_speedup" outpath="bench_deepsdv">
    <comment>Benchmark of the code xPic on Intel Xeon procesor</comment>

    <!-- Configuration -->
    <parameterset name="param_set">
      <parameter name="submit_cmd"                >sbatch</parameter>
      <parameter name="ready_file"                >ready</parameter>
      <parameter name="job_script"                >job.slurm</parameter>
      <parameter name="job_nodetype"              >sdv</parameter>
      <parameter name="job_name"                  >xPic.sdv</parameter>
      <parameter name="job_err"                   >stderr</parameter>
      <parameter name="job_out"                   >stdout</parameter>
      <parameter name="job_outdir"                >output</parameter>
      <parameter name="job_srcname"               >xPic</parameter>
      <parameter name="job_input"                 >input.inp</parameter>
      <parameter name="job_afinity"               ></parameter>
      <parameter name="job_mpiproc" type="int"    >1</parameter>
      <parameter name="job_nthread" type="int"    >24,16,8,4,2,1</parameter>
      <parameter name="i"                         >0,1</parameter>
      <parameter name="run_ntcx"    mode="python">[11520,7680][$i]</parameter>
      <parameter name="run_nblockx" mode="python">[48,32][$i]</parameter>
      <parameter name="run_niter"   mode="python">[100,100][$i]</parameter>
      <parameter name="run_nppc"    type="int"    >1024</parameter>
      <parameter name="extension"                 >intel64</parameter>
      <parameter name="mpiexec"                   >srun</parameter>
      <parameter name="mpiopts"                   ></parameter>
    </parameterset>

    <!-- Files -->
    <fileset name="files">
      <copy>${job_input}</copy>
      <copy>${job_srcname}.${extension}</copy>
      <copy>${job_script}</copy>
    </fileset>

    <!-- Substitute -->
    <substituteset name="subsjob">
      <iofile in="${job_script}" out="${job_script}.run" />
      <sub source="#NAME#"       dest="${job_name}.${job_nthread}" />
      <sub source="#OUTFILE#"    dest="$job_out" />
      <sub source="#ERRFILE#"    dest="$job_err" />
      <sub source="#NODETYPE#"   dest="$job_nodetype" />
      <sub source="#NUMTHREADS#" dest="$job_nthread" />
      <sub source="#MPIPROC#"    dest="$job_mpiproc" />
      <sub source="#MPIEXEC#"    dest="$mpiexec" />
      <sub source="#MPIOPT#"     dest="$mpiopts" />
      <sub source="#EXEC#"       dest="./${job_srcname}.${extension}" />
      <sub source="#AFINITY#"    dest="${job_afinity}" />
      <sub source="#INPUT#"      dest="${job_input}.run" />
      <sub source="#READY#"      dest="$ready_file" />
      <sub source="#OUTDIR#"     dest="$job_outdir" />
    </substituteset>
    <substituteset name="subsinput">
      <iofile in="${job_input}" out="${job_input}.run" />
      <sub source="#NTCX#"      dest="$run_ntcx" />
```

```
58        <sub source="#NBLOCKX#"    dest="$run_nblockx" />
59        <sub source="#NPPC#"       dest="$run_nppc" />
60        <sub source="#NITER#"      dest="$run_niter" />
61        <sub source="#OUTDIR#"     dest="$job_outdir" />
62      </substituteset>
63
64      <!-- Regex pattern -->
65      <patternset name="pattern">
66        <pattern name="number_pat" type="float">Main    loop: $jube_pat_fp s</
              pattern>
67      </patternset>
68
69      <!-- Operation -->
70      <step name="submit">
71        <use>param_set</use>     <!-- use existing parameterset -->
72        <use>files</use>         <!-- use existing fileset -->
73        <use>subsjob</use>       <!-- use existing substituteset -->
74        <use>subsinput</use>     <!-- use existing substituteset -->
75        <do>mkdir ${job_outdir}</do>
76        <do done_file="${ready_file}">${submit_cmd} ${job_script}.run</do>
77      </step>
78
79      <!-- Analyse -->
80      <analyser name="analyse">
81        <use>pattern</use> <!-- use existing patternset -->
82        <analyse step="submit">
83          <file>stdout</file> <!-- file which should be scanned -->
84        </analyse>
85      </analyser>
86
87      <!-- Create result table -->
88      <result>
89        <use>analyse</use> <!-- use existing analyser -->
90        <table name="result" style="pretty" sort="run_ntcx,job_nthread">
91          <column>run_ntcx</column>
92          <column>run_nblockx</column>
93          <column>job_nthread</column>
94          <column>number_pat_max</column>
95          <column>number_pat_min</column>
96          <column>number_pat_avg</column>
97        </table>
98      </result>
99
100   </benchmark>
101 </jube>
```

Listing A.21: xPic-bench-weak.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <jube>
3   <benchmark name="CPU_speedup" outpath="bench_deepcpu">
4     <comment>Benchmark of the code xPic on Intel Xeon procesor</comment>
5
6     <!-- Configuration -->
7     <parameterset name="param_set">
8       <parameter name="submit_cmd"              >qsub</parameter>
9       <parameter name="ready_file"              >ready</parameter>
```

```
10      <parameter name="job_script"                    >job.pbs</parameter>
11      <parameter name="job_nodetype"                  >cluster</parameter>
12      <parameter name="job_name"                      >xPic.cpu</parameter>
13      <parameter name="job_ppn"       type="int"      >16</parameter>
14      <parameter name="job_mpiproc"   type="int"      >1</parameter>
15      <parameter name="job_err"                       >stderr</parameter>
16      <parameter name="job_out"                       >stdout</parameter>
17      <parameter name="job_srcname"                   >xPic</parameter>
18      <parameter name="job_input"                     >weak.inp</parameter>
19      <parameter name="job_afinity"                   >OMP_PROC_BIND=CLOSE</parameter
            >
20      <parameter name="job_nthread"   type="int"      >16</parameter>
21      <parameter name="i"                             >0,1,2,3,4,5,6,7</parameter>
22      <parameter name="job_nnodes"    mode="python">[1,2,4,8,16,32,64,128][$i]</
            parameter>
23      <parameter name="run_ntcx"      mode="python">
            [64,64,64,128,128,128,256,256][$i]</parameter>
24      <parameter name="run_ntcy"      mode="python">
            [32,64,64,64,128,128,128,256][$i]</parameter>
25      <parameter name="run_ntcz"      mode="python">[32,32,64,64,64,128,128,128][$
            i]</parameter>
26      <parameter name="run_nblockx"   mode="python">[8,8,8,8,8,8,8,8][$i]</
            parameter>
27      <parameter name="run_nblocky"   mode="python">[4,8,8,4,8,8,4,8][$i]</
            parameter>
28      <parameter name="run_nblockz"   mode="python">[4,4,8,4,4,8,4,4][$i]</
            parameter>
29      <parameter name="extension"                     >intel64</parameter>
30      <parameter name="mpiexec"                       >mpiexec</parameter>
31    </parameterset>
32
33    <!-- Files -->
34    <fileset name="files">
35      <copy>${job_input}</copy>
36      <copy>${job_srcname}.${extension}</copy>
37      <copy>${job_script}</copy>
38    </fileset>
39
40    <!-- Substitute -->
41    <substituteset name="subsjob">
42      <iofile in="${job_script}" out="${job_script}.run" />
43      <sub source="#NAME#"       dest="${job_name}.${job_nnodes}" />
44      <sub source="#OUTFILE#"    dest="$job_out" />
45      <sub source="#ERRFILE#"    dest="$job_err" />
46      <sub source="#PPN#"        dest="$job_ppn" />
47      <sub source="#NNODES#"     dest="$job_nnodes" />
48      <sub source="#NODETYPE#"   dest="$job_nodetype" />
49      <sub source="#NUMTHREADS#" dest="$job_nthread" />
50      <sub source="#MPIPROC#"    dest="$job_nnodes" />
51      <sub source="#MPIEXEC#"    dest="$mpiexec" />
52      <sub source="#EXEC#"       dest="./${job_srcname}.${extension}" />
53      <sub source="#AFINITY#"    dest="${job_afinity}" />
54      <sub source="#INPUT#"      dest="${job_input}.run" />
55      <sub source="#READY#"      dest="$ready_file" />
56    </substituteset>
57    <substituteset name="subsinput">
58      <iofile in="${job_input}" out="${job_input}.run" />
```

```
59        <sub source="#NTCX#"       dest="$run_ntcx" />
60        <sub source="#NTCY#"       dest="$run_ntcy" />
61        <sub source="#NTCZ#"       dest="$run_ntcz" />
62        <sub source="#NBLOCKX#"    dest="$run_nblockx" />
63        <sub source="#NBLOCKY#"    dest="$run_nblocky" />
64        <sub source="#NBLOCKZ#"    dest="$run_nblockz" />
65      </substituteset>
66
67      <!-- Regex pattern -->
68      <patternset name="pattern">
69        <pattern name="number_pat" type="float">loop: $jube_pat_fp s</pattern>
70        <pattern name="number_fld" type="float">field: $jube_pat_fp s</pattern>
71        <pattern name="number_mvr" type="float">mover: $jube_pat_fp s</pattern>
72        <pattern name="number_mom" type="float">mmnts: $jube_pat_fp s</pattern>
73      </patternset>
74
75      <!-- Operation -->
76      <step name="submit">
77        <use>param_set</use>     <!-- use existing parameterset -->
78        <use>files</use>         <!-- use existing fileset -->
79        <use>subsjob</use>       <!-- use existing substituteset -->
80        <use>subsinput</use>     <!-- use existing substituteset -->
81        <do done_file="${ready_file}">${submit_cmd} ${job_script}.run</do>
82      </step>
83
84      <!-- Analyse -->
85      <analyser name="analyse">
86        <use>pattern</use> <!-- use existing patternset -->
87        <analyse step="submit">
88          <file>stdout</file> <!-- file which should be scanned -->
89        </analyse>
90      </analyser>
91
92      <!-- Create result table -->
93      <result>
94        <use>analyse</use> <!-- use existing analyser -->
95        <table name="result" style="pretty" sort="job_nnodes">
96          <column>job_nnodes</column>
97          <column>number_pat_avg</column>
98          <column>number_fld_avg</column>
99          <column>number_mvr_avg</column>
100         <column>number_mom_avg</column>
101       </table>
102     </result>
103
104   </benchmark>
105 </jube>
```

Listing A.22: benchmarks-systems.xml

```
1 <jube>
2
3   <parameterset name="magainin">
4     <parameter name="benchmark_system_name" type="string">
5       magainin
6     </parameter>
7     <parameter name="number_of_atoms" type="int">
```

```
      34573
    </parameter>
    <parameter name="tprfile" type="string">
      $repo_base/tests/${benchmark_system_name}.tpr
    </parameter>
  </parameterset>

  <parameterset name="bombinin">
    <parameter name="benchmark_system_name" type="string">
      bombinin
    </parameter>
    <parameter name="number_of_atoms" type="int">
      325315
    </parameter>
    <parameter name="tprfile" type="string">
      $repo_base/tests/${benchmark_system_name}.tpr
    </parameter>
  </parameterset>

  <parameterset name="ribosome">
    <parameter name="benchmark_system_name" type="string">
      ribosome
    </parameter>
    <parameter name="number_of_atoms" type="int">
      2217003
    </parameter>
    <parameter name="tprfile" type="string">
      $repo_base/tests/${benchmark_system_name}.tpr
    </parameter>
  </parameterset>

</jube>
```

Listing A.23: gromacs-version.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>

  <parameterset name="gromacs_2016.4">
    <parameter name="version" type="string">2016.4</parameter>
  </parameterset>

  <parameterset name="gromacs_2018">
    <parameter name="version" type="string">2018</parameter>
  </parameterset>

</jube>
```

Listing A.24: systems-descriptions.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jube>

  <parameterset name="system_sdv">
    <parameter name="system_name" type="string">sdv</parameter>
    <parameter name="modules">intel/18.1  parastation/5.2.0-1-intel  extoll</
        parameter>
```

```
7     <parameter name="cmake_module">cmake/3.6.2</parameter>
8     <parameter name="CFLAGS" type="string">-g -static-intel</parameter>
9     <parameter name="CXXFLAGS" type="string">$CFLAGS -std=c++0x</parameter>
10    <parameter name="CC" type="string">mpicc</parameter>
11    <parameter name="CXX" type="string">mpicxx</parameter>
12    <parameter name="THREADS_PER_CORE" type="int">2</parameter>
13    <parameter name="NUMBER_OF_CORES" type="int">24</parameter>
14    <parameter name="MAX_NUM_THREADS" type="int">64</parameter>
15    <parameter name="GMXINSTDIR" type="string">${benchmark_home}/gromacs_x/${
          version}-${system_name}</parameter>
16    <parameter name="GMXSRCDIR" type="string">${repo_base}/src/gromacs-$version<
          /parameter>
17    <parameter name="cmake_defs" type="string">$GMXSRCDIR -DCMAKE_INSTALL_PREFIX
          =$GMXINSTDIR -DBUILD_SHARED_LIBS=OFF -DGMX_FFT_LIBRARY=mkl -DGMX_MPI=ON
          -DGMX_OPENMP=ON -DGMX_CYCLE_SUBCOUNTERS=ON -DGMX_GPU=OFF -
          DGMX_BUILD_HELP=OFF -DGMX_HWLOC=OFF -DGMX_OPENMP_MAX_THREADS=$
          MAX_NUM_THREADS -DGMX_DEFAULT_SUFFIX=OFF</parameter>
18    <parameter name="cmake_command_line" type="string">CFLAGS="$CFLAGS" CXXFLAGS
          ="$CXXFLAGS" CC=$CC CXX=$CXX cmake $cmake_defs</parameter>
19    <parameter name="make_command_line" type="string">make install</parameter>
20  </parameterset>
21
22  <parameterset name="system_knl">
23    <parameter name="system_name" type="string">knl</parameter>
24    <parameter name="modules">intel/18.1  parastation/5.2.0-1-intel extoll</
          parameter>
25    <parameter name="cmake_module">cmake/3.6.2</parameter>
26    <parameter name="CFLAGS" type="string">-xMIC-AVX512 -g -static-intel</
          parameter>
27    <parameter name="CXXFLAGS" type="string">$CFLAGS -std=c++0x</parameter>
28    <parameter name="CC" type="string">mpicc</parameter>
29    <parameter name="CXX" type="string">mpicxx</parameter>
30    <parameter name="THREADS_PER_CORE" type="int">4</parameter>
31    <parameter name="NUMBER_OF_CORES" type="int">64</parameter>
32    <parameter name="MAX_NUM_THREADS" type="int">256</parameter>
33    <parameter name="GMXINSTDIR" type="string">${benchmark_home}/gromacs_x/${
          version}-${system_name}</parameter>
34    <parameter name="GMXSRCDIR" type="string">${repo_base}/src/gromacs-$version<
          /parameter>
35    <parameter name="cmake_defs" type="string">$GMXSRCDIR -DGMX_SIMD=AVX_512_KNL
           -DCMAKE_INSTALL_PREFIX=$GMXINSTDIR -DBUILD_SHARED_LIBS=OFF -
          DGMX_FFT_LIBRARY=mkl -DGMX_MPI=ON -DGMX_OPENMP=ON -
          DGMX_CYCLE_SUBCOUNTERS=ON -DGMX_GPU=OFF -DGMX_BUILD_HELP=OFF -DGMX_HWLOC
          =OFF -DGMX_OPENMP_MAX_THREADS=$MAX_NUM_THREADS -DGMX_DEFAULT_SUFFIX=OFF<
          /parameter>
36    <parameter name="cmake_command_line" type="string">CFLAGS="$CFLAGS" CXXFLAGS
          ="$CXXFLAGS" CC=$CC CXX=$CXX cmake $cmake_defs</parameter>
37    <parameter name="make_command_line" type="string">make install</parameter>
38  </parameterset>
39
40  <parameterset name="system_cluster">
41    <parameter name="system_name" type="string">cluster</parameter>
42    <parameter name="modules">intel/18.1 parastation/5.2.0-1-intel</parameter>
43    <parameter name="cmake_module">cmake/3.6.2</parameter>
44    <parameter name="CFLAGS" type="string">-g -static-intel</parameter>
45    <parameter name="CXXFLAGS" type="string">$CFLAGS -std=c++0x</parameter>
46    <parameter name="CC" type="string">mpicc</parameter>
```

```
47      <parameter name="CXX" type="string">mpicxx</parameter>
48      <parameter name="THREADS_PER_CORE" type="int">2</parameter>
49      <parameter name="NUMBER_OF_CORES" type="int">8</parameter>
50      <parameter name="MAX_NUM_THREADS" type="int">64</parameter>
51      <parameter name="GMXINSTDIR" type="string">${benchmark_home}/gromacs_x/${
            version}-${system_name}</parameter>
52      <parameter name="GMXSRCDIR" type="string">${repo_base}/src/gromacs-$version<
            /parameter>
53      <parameter name="cmake_defs" type="string">$GMXSRCDIR -DCMAKE_INSTALL_PREFIX
            =$GMXINSTDIR -DBUILD_SHARED_LIBS=OFF -DGMX_FFT_LIBRARY=mkl -DGMX_MPI=ON
            -DGMX_OPENMP=ON -DGMX_CYCLE_SUBCOUNTERS=ON -DGMX_GPU=OFF -
            DGMX_BUILD_HELP=OFF -DGMX_HWLOC=OFF -DGMX_OPENMP_MAX_THREADS=$
            MAX_NUM_THREADS -DGMX_DEFAULT_SUFFIX=OFF</parameter>
54      <parameter name="cmake_command_line" type="string">CFLAGS="$CFLAGS" CXXFLAGS
            ="$CXXFLAGS" CC=$CC CXX=$CXX cmake $cmake_defs</parameter>
55      <parameter name="make_command_line" type="string">make install</parameter>
56    </parameterset>
57
58
59
60  </jube>
```

List of Acronyms and Abbreviations

A

API: Application Programming Interface
ASTRON: Netherlands Institute for Radio Astronomy, Netherlands

B

BeeGFS: The Fraunhofer Parallel Cluster File System (previously acronym FhGFS). A high-performance parallel file system.
BeeOND: BeeGFS-on-demand, parallel storage based on BeeGFS
BSC: Barcelona Supercomputing Centre, Spain

C

CERN: European Organisation for Nuclear Research/Organisation Europenne pour la Recherche Nuclaire, International organisation
CM: Cluster Module: with its Cluster Nodes (CN) containing high-end general-purpose processors and a relatively large amount of memory per core
CMS: Compact Muon Solenoid experiment at CERNs LHC
CN: Cluster Node (functional entity)
CNN: Convolutional Neural Networks
CPU: Central Processing Unit

D

DAM: Data Analytics Module: with nodes (DN) based on general-purpose processors a huge amount of (non-volatile) memory per core, and support for the specific requirements of data-intensive applications
DDG: Design and Developer Group of the DEEP-EST project
DEEP: Dynamical Exascale Entry Platform (project FP7-ICT-287530)
DEEP-ER: DEEP - Extended Reach (project FP7-ICT-610476)
DEEP-EST: DEEP - Extreme Scale Technologies
DRAM: Dynamic Random Access Memory. Typically describes any form of high capacity volatile memory attached to a CPU

E

EC:	European Commission
ESB:	Extreme Scale Booster: with highly energy-efficient many-core processors as Booster Nodes (BN), but a reduced amount of memory per core at high bandwidth
EU:	European Union
Exascale:	Computer systems or Applications, which are able to run with a performance above 1018 Floating point operations per second
EXTOLL:	High speed interconnect technology for HPC developed by UHEI

F

FFT:	Fast Fourier Transform
FHG-ITWM:	Fraunhofer Gesellschaft zur Foerderung der Angewandten Forschungs e.V., Germany
Flop/s:	Floating point Operation per second
FPGA:	Field-Programmable Gate Array, Integrated circuit to be configured by the customer or designer after manufacturing

G

GFlop/s:	Gigaflop, 109 Floating point operations per second
GPU:	Graphics Processing Unit
GROMACS:	A toolbox for molecular dynamics calculations providing a rich set of calculation types, preparation and analysis tools

H

H2020:	Horizon 2020
h5perf:	Performance utility of HDF5 library
HPC:	High Performance Computing
HPDA:	High Performance Data Analytics
HPDBSCAN:	A clustering code used by UoI in the field of Earth Science
HW:	Hardware

I

Intel:	Intel Germany GmbH, Feldkirchen, Germany
I/O:	Input/Output. May describe the respective logical function of a computer system or a certain physical instantiation
IOR:	Interleved Or Randomized
iPic3D:	Programming code developed by the KULeuven to simulate space weather

J

JUBE: Juelich Benchmarking Environment
JUELICH: Forschungszentrum Juelich GmbH, Juelich, Germany
JURECA: Juelich Research on Exascale Cluster Architectures

K

KB: Kilo Bytes
KNL: Knights Landing, second generation of Intel Xeon PhiTM
KULeuven: Katholieke Universiteit Leuven, Belgium

L

LHC: Large Hadron Collider (LHC), the worlds most powerful accelerator providing research facilities for High Energy Physics researchers across the globe
LLNL: Lawrence Livermore National Laboratory

M

MB: Mega Bytes
MPI: Message Passing Interface, API specification typically used in parallel programs that allows processes to communicate with one another by sending and receiving messages
MSA: Modular Supercomputer Architecture
MWF: Modular Workflow Format

N

NAM: Network Attached Memory
NCSA: National Centre for Supercomputing Applications, Bulgaria
NEST: Widely-used, publically available simulation software for spiking neural network models developed by NMBU.
NMBU: Norwegian University of Life Sciences, Norway
NUMA: Non-Uniform Memory Access
NVM: Non-Volatile Memory. Used to describe a physical technology or the use of such technology in a non-block-oriented way in a computer system

O

OpenMP: Open Multi-Processing, Application programming interface that support multiplatform shared memory multiprocessing

P

PFlop/s: Petaflop, 1015 Floating point operations per second
piSVM: Parallel classification algorithm
PMT: Project Management Team of the DEEP-EST project

Q

R

RAM: Random-Access Memory
RM: Resource Manager

S

SDV: Software Development Vehicle: HW systems to develop software in the time frame where the DEEP-EEST prototype is not yet available.
SIMD: Single Instruction Multiple Data
SIONlib: Parallel I/O library developed by Forschungszentrum Juelich
SLURM: Job scheduler that will be used and extended in the DEEP-EST prototype
SW: Software
SWF Standard Workload Format

T

TFlops: Teraflop, 10^{12} Floating point operations per second
Tk: Task, Followed by a number, term to designate a Task inside a Work Package of the DEEP-EST project

U

UoI: Haskoli Islands University of Iceland, Iceland

V

W

WP: Work package

X

Y

Z