Readme of the HPC-ODA Dataset Collection

Alessio Netti
Leibniz Supercomputing Centre, Garching bei München, Germany
alessio.netti@lrz.de

April 8, 2021

Contents

1 Overview of HPC-ODA **2**
1.1 Usage of the Datasets . . . 2
1.2 Structure of the Datasets . . . 2
1.3 Format of the Datasets . . . 4
1.4 Code Examples . . . 5

2 Description of the Datasets **5**
2.1 Fault Detection . . . 5
2.1.1 Sensor Data . . . 7
2.1.2 Responses . . . 7
2.2 Power Consumption Prediction . . . 7
2.2.1 Sensor Data . . . 8
2.2.2 Responses . . . 9
2.3 Application Classification . . . 9
2.3.1 Sensor Data . . . 10
2.3.2 Responses . . . 11
2.4 Infrastructure Management . . . 11
2.4.1 Sensor Data . . . 11
2.4.2 Responses . . . 12
2.5 Cross-architecture . . . 12
2.5.1 Sensor Data . . . 12
2.5.2 Responses . . . 14
2.6 DEEP-EST Dataset . . . 14
2.6.1 Cluster Module - Sensor Data . . . 15
2.6.2 Cluster Module - Responses . . . 16
2.6.3 Extreme-Scale Booster - Sensor Data . . . 16
2.6.4 Extreme-Scale Booster - Responses . . . 17

1 Overview of HPC-ODA

Here we provide a high-level overview of the HPC-ODA dataset collection, explaining its purpose, composition and structure. If you are using HPC-ODA for your research work, please cite it as "*Correlation-wise Smoothing: Lightweight Knowledge Extraction for HPC Monitoring Data*" by Alessio Netti et al. [6], where we originally introduced it.

1.1 Usage of the Datasets

HPC-ODA is a collection of datasets (also called *segments*) acquired on production HPC systems, which are representative of several real-world use cases in the field of *Operational Data Analytics* (ODA) for the improvement of reliability, energy efficiency and system operations in general. The datasets are composed of monitoring sensor data, acquired from different components in HPC systems, depending on the specific use case. Two tools, whose overhead is proven to be very light, were used to acquire data in the HPC-ODA datasets:

- *Data Center Data Base* (DCDB) [5]: a lightweight and holistic monitoring framework developed at the Leibniz Supercomputing Centre (LRZ). This solution was employed for the Power Consumption Prediction, Application Classification, Infrastructure Management and Cross-architecture segments.
- *Lightweight Distributed Metric Service* (LDMS) [1]: a monitoring solution developed at the Sandia Laboratories. This solution was used for the Fault Detection segment.

The aim of HPC-ODA is to provide several vertical slices of the monitoring data available in a large-scale HPC installation in the form of segments. The segments all have different granularities, in terms of data sources and time scale, and provide several use cases on which models and approaches to data processing can be evaluated. While having a production dataset from a whole HPC system - from the infrastructure down to the CPU core level - at a fine time granularity would be ideal, this is often not feasible due to the confidentiality of the data, as well as the sheer amount of storage space required.

1.2 Structure of the Datasets

HPC-ODA is composed of 6 different segments with different characteristics, which can be used independently. Each segment has an associated ODA problem that can be analyzed. The overall structure of HPC-ODA is summarized in Figure 1 and is divided in the following segments:

- *Power Consumption Prediction*: this is a fine-granularity dataset that was collected from a single compute node in the CooLMUC-3 HPC system at LRZ. It contains both node-level data as well as per-CPU core metrics.

Dataset	HPC System	Data Points	Length	Nodes	Granularity
Power Consumption	CooLMUC-3	∼300000 47 sensors	8 hours	1	100ms per-CPU
Fault Detection	ETH testbed	∼1000000 128 sensors	16 days	1	1s per-node
Application Classification	SuperMUC-NG	∼1000000 52 sensors	1 day	16	1s per-node
Infrastructure Management	CooLMUC-3	∼450000 31 sensors	16 days	All	10s cluster
Cross Architecture	1. CooLMUC-3, 2. LRZ testbed, 3. SuperMUC-NG	∼300000 1. 46, 2. 39, 3. 52 sensors	1 day	3	1s per-node
DEEP-EST Dataset	1. CM, 2. ESB (CPU), 3. ESB (GPU)	∼450000 1. 35, 2. 32, 3. 27 sensors	2 days	32	10s per-node

Table 1: Overview of the segments composing the HPC-ODA collection.

This dataset can be used to perform regression tasks such as power consumption prediction [7].

- *Fault Detection*: this is a medium-granularity dataset that was collected from a single compute node in a testbed system at ETH, while it was subjected to fault injection. It contains only node-level data, as well as the labels for both the applications and faults being executed on the HPC node in time. This dataset can be used to perform fault classification [4].
- *Application Classification*: this is a medium-granularity dataset that was collected from 16 compute nodes in the SuperMUC-NG HPC system at LRZ while running different parallel MPI applications. Data is at the compute node level, separated for each of them, and is paired with the labels of the applications being executed. This dataset can be used for tasks such as application classification [2].
- *Infrastructure Management*: this is a coarse-granularity dataset that was collected from the infrastructure of the CooLMUC-3 HPC system, and mostly contains data about its warm-water cooling system and power consumption. The data is at the rack level, and can be used for regression tasks such as outlet water temperature or removed heat prediction [3].
- *Cross-architecture*: this medium-granularity dataset is a variant of the Application Classification one, and shares the same ODA use case. Here, however, single-node configurations of the applications were executed on

three compute node types with different CPU architectures. This dataset can be used for cross-architecture application classification or comparison.

- *DEEP-EST Dataset*: this medium-granularity dataset was collected on the modular DEEP-EST HPC system at the Juelich Supercomputing Centre (JSC) and consists of three parts - one associated with the CPU-only CM module, and two with the GPU-enabled ESB module. The three parts were collected in two sessions, one on the CM and one on the ESB, on 16 compute nodes each while running several MPI applications under different warm-water cooling configurations. This dataset can be used for CPU and GPU temperature prediction, or for thermal characterization.

1.3 Format of the Datasets

HPC-ODA has the following directory structure, starting from the root:

1. One directory for each segment. Unlike the others, the DEEP-EST segment is divided in three separate directories.
2. A *sensors* and *responses* directory, containing respectively the sensor data of the segment, as well as the responses for its reference ODA problem.
3. A series of CSV files each containing data belonging to a different sensor. Response files are always named as *responses.csv*.
4. One or more sub-directories containing sensor data or responses related to sub-components in the segment (optional).

Each CSV file in HPC-ODA stores the time series of a certain sensor and has a very simple format: it is a two-column file, where each line contains a $timestamp, value$ pair. The timestamps are expressed in 64 bits (i.e., nanoseconds), whereas sensor values are always integers. In the case of label files (e.g., for the Fault Detection or Application Classification segments), these values are expressed as strings. Monotonic sensors (e.g., CPU performance counters) were converted to their first-order derivatives using the delta operation, except for metrics that are intrinsically monotonic (e.g., energy). The sensor files are named to reflect the name of the corresponding data source, and can be easily identified. Moreover, all sensor and response files are time-aligned within a certain segment and contain roughly the same number of readings. Sometimes, the response files correspond to a processed version of a sensor (e.g., after smoothing): if this is the case, a raw version of the response file is always supplied as *responses_raw.csv*. In case multiple alternative versions of the response files are available (e.g., fault or application labels in the Fault Detection segment), these are also supplied.

1.4 Code Examples

A small Python framework is supplied with the HPC-ODA collection, providing scripts and classes to automatically parse each segment discussed here and evaluate the performance of the respective ODA use cases. The segments can be processed with a series of *signature* methods, to aggregate the sensor data and make it suitable for machine learning usage, and several information loss metrics are supplied to evaluate the quality of the compression: scripts are included to reproduce all of the experiments in the paper "*Correlation-wise Smoothing: Lightweight Knowledge Extraction for HPC Monitoring Data*" by Alessio Netti et al. [6]. The framework is modular and users can easily integrate new analysis algorithms by implementing the appropriate interfaces.

The only dependencies consist in the *numpy*, *scipy*, *scikit-learn* and *seaborn* (optional for visualization) packages, and the suggested execution environment is Python 3.6. Before running the example script *launcher.py*, the directories of the HPC-ODA segments must be moved inside the *hpc-signatures-o-matic/datasets* directory.

2 Description of the Datasets

Here we describe in detail each of the segments that can be found in HPC-ODA and how they can be used to perform ODA tasks.

2.1 Fault Detection

This segment contains sensor data from a compute node in a testbed HPC system at ETH. The data in this segment was originally acquired in the context of a previous work [4] and is also publicly available[1]. Here, we employ only a subset of it: specifically, we use the CPU-MEM and HDD parts (corresponding to blocks I and III in the original dataset) that employ single-core applications, spanning a total of 16 days of monitoring data. The authors executed benchmark applications and at the same time injected faults in the system at specific times via dedicated programs, so as to trigger anomalies in the behaviour of the applications. The series of benchmarks that were used to load the compute node while acquiring the dataset is the following:

1. *DGEMM* measures matrix-to-matrix multiplication performance.
2. *HPCC* is a collection of benchmarks that stress both CPU and memory.
3. *HPL* measures performance in solving a system of linear equations.
4. *STREAM* measures a system's memory bandwidth.
5. *Bonnie++* measures HDD read-write performance.

[1] `https://zenodo.org/record/2553224`

6. *IOZone* measures HDD read-write performance.

Fault injection is achieved by means of an ad-hoc framework. Faults are injected at specific times and for specific durations following a certain statistical workload, in order to simulate a realistic scenario and avoid bias in the data, and the fault programs that were used to reproduce anomalous conditions are available at the Github repository associated with the original work. Each fault program can operate in a high or low-intensity mode, thus doubling the number of possible fault conditions. The programs are the following:

1. *leak* periodically allocates 16MB arrays which are never released. In low-intensity mode, 4MB arrays are allocated. This program produces a *memory leak* fault, which leads to memory fragmentation and severe system slowdown.

2. *memeater* allocates a 36MB array which is filled with integers. The size of the array is then periodically increased and new elements are filled in. The application restarts after 10 iterations. In low-intensity mode, an 18MB array is used. This program produces a *memory interference* fault by saturating bandwidth.

3. *ddot* repeatedly calculates the dot product between two equal-size matrices. The sizes of the matrices change periodically between 0.9, 5 and 10 times the cache's size. In low-intensity mode, the size of the matrices is halved. This program produces a *CPU and cache interference* fault.

4. *dial* repeatedly generates random floating-point numbers and performs numerical operations over them. In low-intensity mode, the program sleeps for 0.5 seconds for each second of operation. This program produces an *ALU interference* fault.

5. *cpufreq* decreases the maximum allowed CPU frequency by 50% of its original value through the Linux Intel P-State driver. In low-intensity mode, the maximum frequency is reduced by 30%. This program simulates a *system misconfiguration* or *failing CPU* fault.

6. *pagefail* makes any page allocation request fail with 50% probability, by using the Linux kernel's fault injection framework. In low-intensity mode, page allocations fail with 25% probability. This program simulates a *system misconfiguration* or *hardware malfunction* fault.

7. *ioerr* triggers errors upon hard-drive I/O operations, again using the Linux kernel's fault injection framework. One out of 500 I/O operations fails with 20% probability in high-intensity mode, and with 10% probability in low-intensity mode. This program simulates a *failing hard drive* fault.

8. *copy* repeatedly writes and then reads back a 400MB file from a hard drive. After such a cycle, the program sleeps for 2 seconds. In low-intensity mode, a 200MB file is used. This program simulates an *I/O interference* or *failing hard drive* fault by saturating I/O bandwidth.

2.1.1 Sensor Data

The compute node used for data acquisition consists of two Intel Xeon E5-2630 v3 CPUs, 128GB of RAM, a Seagate ST1000NM0055-1V4 1TB hard drive and runs the CentOS 7.3 operating system. Monitoring is performed via the LDMS framework, which was configured to sample at each second a variety of metrics coming from the following seven different plug-ins:

- *meminfo* collects general information on RAM usage.
- *perfevent* collects CPU performance counters.
- *procinterrupts* collects information on hardware and software interrupts.
- *procdiskstats* collects statistics about hard drive usage.
- *procsensors* collects metrics about CPU temperature and frequency.
- *procstat* collects general metrics about CPU usage.
- *vmstat* collects information about virtual memory usage.

A total of 128 sensors is included in this segment. All of the sensors are system-wide, describing the status of the compute node as a whole. The only exception lies in the CPU performance counters coming from the *perfevent* plugin: we include in the HPC-ODA segment only those for CPU core 0, on which both the applications and fault programs were running for the specific portion of the original dataset that we consider.

2.1.2 Responses

The main response file of this segment contains the labels of the fault programs (at most one at a time) being injected in the compute node. An alternative response file is available, named *responses_applications.csv*, and containing instead the labels of the benchmark applications being executed.

2.2 Power Consumption Prediction

This segment contains fine-granularity sensor data from a compute node in the CooLMUC-3 HPC system at LRZ. It is roughly 8 hours long, and all sensors are sampled every 100ms. As the name of the segment implies, its use case is tied to prediction of power consumption, which can be used to steer runtime management decisions such as CPU frequency assignment. While acquiring the data, we executed several HPC proxy applications from the Coral-2 suite,[2] plus other applications, in order to load the compute node and obtain realistic data. The applications are the following:

[2] `https://asc.llnl.gov/coral-2-benchmarks/`

Table 2: The main command-line and configuration parameters used for each of the applications in the HPC-ODA Power segment. Parameters that remain unaltered in smaller configurations are omitted.

Application	Configuration
Kripke	–groups 64 –niter 7 –gset 1 –quad 128 –dset 128 –legendre 4 –zones 64,64,64 –procs 1,1,1 **Small**: –niter 10 –zones 64,64,32
AMG	-problem 1 -n 192 384 384 -P 1 1 1 **Small**: -n 192 192 384
Nekbone	*iel0,ielN,ielD*=256 512 1 **Small**: *iel0,ielN,ielD*=256 448 4
LAMMPS	-v x 8 -v y 16 -v z 16 -in in.reaxc.hns **Small**: -v x 8 -v y 8 -v z 16
IHPL	*problem sizes*=20000 22000 25000 26000, *times to run a test*=2 2 2 1 **Small**: *problem sizes*=1000 2000 5000 10000 15000 18000 20000, *times to run a test*=4 2 2 2 2 2 2

1. *Kripke*: a structured deterministic transport using wavefront algorithms, which stress both memory and network latency and bandwidth.
2. *AMG*: a parallel algebraic multigrid solver for linear systems. It is highly memory and network-bound, relying on many small messages.
3. *Nekbone*: a mini-app derived from the Nek5000 Navier-Stokes CFD solver. It is a compute-intensive application.
4. *LAMMPS*: a classical molecular dynamics code, which evenly stresses most components in a HPC system.
5. *HPL*: measures performance in solving a system of linear equations.

All applications were executed with multiple input and size configurations in order to remove any bias related to the amount of allocated memory, or associated to the specific configurations being used. For these single-node runs, we use the applications with a number of OpenMP threads equal to 64, as the available CPU cores, and only one MPI rank. The specific parameters for each application are listed in Table 2.

2.2.1 Sensor Data

We collect data from a compute node in the CooLMUC-3 HPC system at LRZ:[3] it is composed of 148 compute nodes, each equipped with a 64-core Intel Xeon Phi 7210 CPU, 96GB of RAM and a dual-rail Intel OmniPath network interface. The operating system is SUSE Linux Enterprise Server 12. In addition, CooLMUC-3 uses warm-water cooling for all components in the system. We leverage the DCDB framework for monitoring, and in particular we use the following plugins:

[3] `https://doku.lrz.de/display/PUBLIC/CoolMUC-3`

- *procFS*: captures metrics regarding memory activity from the *meminfo* and *vmstat* files, as well as about CPU activity from the *stat* file.
- *sysFS*: collects information about compute node power and temperature from sensors exposed in the file system.
- *perfevent*: collects CPU performance counters.
- *OPA*: collects metrics regarding network activity.

This segment contains a total of 47 metrics at the compute node level. In addition, we supply a series of performance counters and activity indicators for each of the 64 CPU cores in the compute node. These are contained in separate sub-directories, one for each core.

2.2.2 Responses

The response file of the segment contains the power measurements of the compute node, each averaged with the subsequent 2 samples. The resulting regression task predicts the average power consumption in the next 300ms. The raw power measurements are supplied as a *responses_raw.csv* file. In all cases, the measurements are captured by a sensor at the power outlet, and describe the whole node's activity. A third *responses_applications.csv* file contains the labels of the applications being executed at each point in time.

2.3 Application Classification

This segment contains sensor data collected from 16 distinct compute nodes in the SuperMUC-NG HPC system at LRZ, while running certain parallel applications. The segment is 1 day long, and sensors are sampled every second. The associated ODA use case consists in application classification, which allows to recognize the codes running on compute nodes, or the underlying workloads, in order to improve scheduling and management decisions. Similarly to the Power Consumption Prediction segment, we executed several proxy applications from the Coral-2 suite, together with other applications, in order to acquire the data. The applications are the following:

1. *Kripke*: a structured deterministic transport using wavefront algorithms, which stress both memory and network latency and bandwidth.
2. *AMG*: a parallel algebraic multigrid solver for linear systems. It is highly memory and network-bound, relying on many small messages.
3. *PENNANT*: a mini-app for hydrodynamics on unstructured meshes, making use of highly irregular memory access patterns.
4. *LAMMPS*: a classical molecular dynamics code, which evenly stresses most components in a HPC system.

Table 3: The main command-line and configuration parameters used for each of the applications in the HPC-ODA Application segment. Parameters that remain unaltered in smaller configurations are omitted.

Application	Configuration
Kripke	–groups 64 –niter 8 –gset 1 –quad 128 –dset 128 –legendre 4 –zones 256,128,128 –procs 4,2,2 **Small**: –zones 128,128,128 **Smaller**: –niter 10 –zones 128,128,64
AMG	-problem 2 -n 160 160 160 -P 4 2 2; *time_steps*=30 **Small**: -n 120 120 120 **Smaller**: -n 100 100 100
PENNANT	"test/leblancx4/leblancx4.pnt"; *tstop=10.0*, *meshparams*=640 5760 1.0 9.0 **Small**: *meshparams*=480 4320 1.0 9.0 **Smaller**: *tstop=12.0*, *meshparams*=320 2880 1.0 9.0
LAMMPS	-v x 32 -v y 32 -v z 16 -in in.reaxc.hns **Small**: -v x 32 -v y 16 -v z 16 **Smaller**: -v x 16 -v y 16 -v z 16
Quicksilver	-i "Coral2_P1.inp" -X 256 -Y 128 -Z 128 -x 256 -y 128 -z 128 -I 4 -J 2 -K 2 –nParticles 2621440 **Small**: -X 128 -Y 128 -Z 128 -x 128 -y 128 -z 128 –nParticles 1310720 **Smaller**: -X 128 -Y 128 -Z 64 -x 128 -y 128 -z 64
HPL	*Ns*=258816, *NBs*=384, *Ps*=4, *Qs*=4 **Small**: *Ns*=192000 **Smaller**: *Ns*=173568

5. *HPL*: measures performance in solving a system of linear equations. In this case, we employ the MPI version of the benchmark.

6. *Quicksilver*: a Monte Carlo transport benchmark, with significant branching and stressing memory latency.

Like for the Power Consumption Prediction segment, all applications were executed with multiple configurations in order to minimize bias when performing classification. The runs were done in parallel using one MPI rank for each of the 16 compute nodes, and 48 OpenMP threads per compute node, as many as the available CPU cores. The parameters associated with each application are listed in Table 3.

2.3.1 Sensor Data

Data is collected from the SuperMUC-NG HPC system at LRZ:[4] this large-scale system is composed of 6440 compute nodes, each equipped with a 48-core Intel Skylake Xeon Platinum 8174 CPU, 96GB of RAM, an Intel OmniPath network interface and running SUSE Linux Enterprise Server 12. Like CooLMUC-3, SuperMUC-NG also employs warm-water cooling for the compute nodes. DCDB is used once again to collect sensor data, with the following plugins:

- *procFS*: captures metrics regarding memory activity from the *meminfo* and *vmstat* files, as well as about CPU activity from the *stat* file.

[4] `https://doku.lrz.de/display/PUBLIC/SuperMUC-NG`

- *sysFS*: collects information about compute node power and temperature from sensors exposed in the file system.
- *perfevent*: collects CPU performance counters.
- *OPA*: collects metrics regarding network activity.

The sensor data for each compute node is stored in a separate sub-directory in the segment, and the response files follow the same scheme. In this case, no CPU core-level data is stored, and all information is at the compute node level. A total of 52 sensors for each compute node is supplied with this segment.

2.3.2 Responses

The segment has one response file for each of the 16 compute nodes, containing the labels of the applications running at each point in time. The segment can also be modified to use one single response file (as all nodes execute the same application at a certain point in time) and to combine the sensors coming from the single compute nodes.

2.4 Infrastructure Management

This segment contains coarse-granularity sensor data from the entire CooLMUC-3 HPC system at LRZ. The segment is approximately 16 days long, and data is sampled every 10s. Available sensors capture the state of the warm-water cooling system, as well as power measurements at different levels. The use case associated to this segment is infrastructure management, in particular the prediction of the amount of heat removed from the HPC system: this information can be used, for example, to tune the inlet water temperature of the system or steer other decisions.

2.4.1 Sensor Data

The CooLMUC-3 system is as described for the Power Consumption Prediction segment, and we use DCDB to capture the infrastructure sensor data. This time we employ the following plugins:

- *PDU*: captures power consumption measurements from the power distribution units in the system.
- *SNMP*: collects metrics from the warm-water cooling system.

In this case, sensor data is collected at the rack level, and for each rack a separate sub-directory in the segment can be found. A total of 31 sensors for each rack is supplied with this segment.

2.4.2 Responses

The segment has one response file for each rack, stored in a separate subdirectory, containing the removed heat measurements, each averaged with the next 29 samples. The resulting regression problem predicts the average amount of heat removed in each rack in the next 5 minutes. For each rack, we also supply the raw removed heat measurements, with files named *responses_raw.csv*.

2.5 Cross-architecture

This segment contains sensor data collected from 3 distinct compute nodes belonging to three different HPC systems, with different CPU architectures, while running certain single-node parallel applications at separate times. The systems we use are SuperMUC-NG, CooLMUC-3, and an AMD testbed system at LRZ. For each node type the segment is 1 day long, and sensors are sampled every second. The associated ODA use case is the same as in the Application Classification segment, and we use the same Coral-2 applications as in the following:

1. *Kripke*: a structured deterministic transport using wavefront algorithms, which stress both memory and network latency and bandwidth.
2. *AMG*: a parallel algebraic multigrid solver for linear systems. It is highly memory and network-bound, relying on many small messages.
3. *PENNANT*: a mini-app for hydrodynamics on unstructured meshes, making use of highly irregular memory access patterns.
4. *LAMMPS*: a classical molecular dynamics code, which evenly stresses most components in a HPC system.
5. *HPL*: measures performance in solving a system of linear equations. In this case, we employ the MPI version of the benchmark.
6. *Quicksilver*: a Monte Carlo transport benchmark, with significant branching and stressing memory latency.

Like for the Application Classification segment, all applications were executed with multiple configurations in order to minimize bias when performing classification. The runs were done using shared-memory parallelism, with only one MPI rank and as many threads as the available physical CPU cores. The parameters for each application configuration are listed in Table 4.

2.5.1 Sensor Data

The SuperMUC-NG and CooLMUC-3 HPC systems were already introduced in the Application Classification and Power Consumption Prediction segments, and the corresponding sensor data has the same format. In addition, we use a single compute node from an AMD testbed cluster at LRZ, equipped with two 64-core AMD Epyc Rome 7742 CPUs and 512GB of RAM. DCDB is used once

Table 4: The main command-line and configuration parameters used for each of the applications in the HPC-ODA Cross-architecture segment. Parameters that remain unaltered in smaller configurations are omitted.

Application	Configuration
Kripke	–groups 64 –niter 14 –gset 1 –quad 128 –dset 128 –legendre 4 –zones 64,64,64 –procs 1,1,1 **Small**: –niter 20 –zones 64,64,32 **Smaller**: –zones 64,32,32
AMG	-problem 2 -n 160 160 160 -P 1 1 1; *time_steps*=60 **Small**: -n 120 120 120 **Smaller**: -n 100 100 100
PENNANT	"test/leblancx4/leblancx4.pnt"; *tstop=10.0*, *meshparams*=320 2880 1.0 9.0 **Small**: *meshparams*=240 2160 1.0 9.0 **Smaller**: *tstop=12.0*, *meshparams*=160 1440 1.0 9.0
LAMMPS	-v x 16 -v y 16 -v z 16 -in in.reaxc.hns **Small**: -v x 8 -v y 16 -v z 16 **Smaller**: -v x 8 -v y 8 -v z 16
Quicksilver	-i "Coral2_P1.inp" -X 128 -Y 128 -Z 64 -x 128 -y 128 -z 64 -I 1 -J 1 -K 1 –nParticles 163840 **Small**: -X 128 -Y 64 -Z 64 -x 128 -y 64 -z 64 –nParticles 81920 **Smaller**: -X 64 -Y 64 -Z 64 -x 64 -y 64 -z 64
HPL	N=4, Ns=57600 57600 57600 57600, NBs=384, Ps=1, Qs=1 **Small**: Ns=53760 53760 53760 53760 **Smaller**: Ns=49920 49920 49920 49920

again to collect sensor data, with different configurations for each node type. The following is the data that is available on all 3 architectures:

- *procFS*: captures metrics regarding memory activity from the *meminfo* and *vmstat* files, as well as about CPU activity from the *stat* file.
- *perfevent*: collects CPU performance counters.

This is instead the data specifically available on the SuperMUC-NG node:

- *sysFS*: power, energy and temperature sensors captured at the CPU level.
- *OPA*: collects metrics regarding network activity.

Finally, these are the metrics that are specific to the CooLMUC-3 system:

- *sysFS*: power, energy and temperature sensors captured both at the CPU level and at the compute node level.

The AMD testbed provides a limited amount of sensors due to its experimental nature. The sensor data for each compute node type is stored in a separate sub-directory in the segment, and the response files follow the same scheme. In this case, no CPU core-level data is stored, and all information is at the compute node level. For the SuperMUC-NG, CooLMUC-3 and AMD testbed nodes a total of 52, 46 and 39 sensors respectively are supplied with this segment.

2.5.2 Responses

The segment has one response file for each of the 3 compute node types, containing the application labels at each point in time. The data of the three node types can be combined and used in a single experiment only if some of the sensors are dropped - hence, all compute node types will have the same monitoring structure - or if the data is pre-processed to a uniform representation.

2.6 DEEP-EST Dataset

This segment was acquired on the modular DEEP-EST HPC system[5] hosted at JSC, leveraging its active production DCDB installation. The segment's main purpose is to supply training data for CPU and GPU temperature prediction models. It is split in three parts: one is associated with the CPU-only Cluster Module (CM), whereas the other two are respectively associated with CPUs and GPUs of the Extreme-Scale Booster (ESB), and were all acquired in two different sessions, one for each module. Each of the three parts of the segment is roughly 1-day long, and contains data sampled at 10s intervals. Similarly to what was done for the Application segment, we executed several MPI applications, under three possible configurations, on 16 compute nodes of the chosen module - the order of the applications is randomized at runtime. On the CM we use the following Coral-2 applications:

1. *Kripke*: a structured deterministic transport using wavefront algorithms, which stress both memory and network latency and bandwidth.
2. *AMG*: a parallel algebraic multigrid solver for linear systems. It is highly memory and network-bound, relying on many small messages.
3. *LAMMPS*: a classical molecular dynamics code, which evenly stresses most components in a HPC system.
4. *Quicksilver*: a Monte Carlo transport benchmark, with significant branching and stressing memory latency.
5. *HPL*: measures performance in solving a system of linear equations. In this case, we employ the MPI version of the benchmark.
6. *PENNANT*: a mini-app for hydrodynamics on unstructured meshes, making use of highly irregular memory access patterns.
7. *Nekbone*: a mini-app derived from the Nek5000 Navier-Stokes CFD solver. It is a compute-intensive application.

On the ESB we use a slightly different application set: on top of the CPU-only versions of Kripke, AMG and HPL, we use the GPU-accelerated versions of LAMMPS, Quicksilver and HPL itself. For most applications we use one

[5] `https://deeptrac.zam.kfa-juelich.de:8443/trac/wiki/Public/User_Guide`

Table 5: The main command-line and configuration parameters used for the applications in the DEEP-EST segment associated with the CM. Parameters that remain unaltered in smaller configurations are omitted.

Application	Configuration
Kripke	–groups 64 –niter 8 –gset 1 –quad 128 –dset 128 –legendre 4 –zones 256,256,128 –procs 4,2,2 **Small**: –zones 256,128,128 **Smaller**: –zones 128,128,128 **ID**: 1
AMG	-problem 2 -n 160 160 160 -P 4 2 2; *time_steps*=30 **Small**: -n 140 140 140 **Smaller**: -n 120 120 120 **ID**: 2
LAMMPS	-v x 64 -v y 32 -v z 32 -in in.reaxc.hns **Small**: -v x 32 -v y 32 -v z 32 **Smaller**: -v x 32 -v y 32 -v z 16 **ID**: 3
Quicksilver	-i "Coral2_P1.inp" -X 512 -Y 256 -Z 256 -x 512 -y 256 -z 256 -I 4 -J 2 -K 2 –nParticles 2621440 **Small**: -X 256 -Y 256 -Z 256 -x 256 -y 256 -z 256 **Smaller**: -X 256 -Y 128 -Z 128 -x 256 -y 128 -z 128 **ID**: 4
HPL	*N*=1, *Ns*=283392, *NBs*=384, *Ps*=4, *Qs*=4 **Small**: *N*=2, *Ns*=192000 192000 **Smaller**: *Ns*=173568 173568 **ID**: 5
PENNANT	"test/leblancx4/leblancx4.pnt"; *tstop=11.5*, *meshparams*=640 5760 1.0 9.0 **Small**: *meshparams*=480 4320 1.0 9.0 **Smaller**: *tstop=14.0*, *meshparams*=320 2880 1.0 9.0 **ID**: 6
Nekbone	*iel0,ielN,ielD*=64 512 1 **Small**: *iel0,ielN,ielD*=64 512 2 **Smaller**: *iel0,ielN,ielD*=64 448 1 **ID**: 7

MPI rank per node with as many OpenMP threads as physical CPU cores, with the exception of the GPU-accelerated LAMMPS and Quicksilver benchmarks, where we use one OpenMP thread and as many MPI ranks as physical cores per node. The specific configurations of the applications being used are summarized in Table 5 for the CM, and in Table 6 for the ESB. On top of running the chosen set of HPC applications, we also applied multiple settings for the secondary inlet water temperature of each rack's cooling unit throughout our experiments: in particular, we used 35C, 37.5C, 40C, 42.5C and 45C as settings, each applied for roughly 4 hours, thus providing an extensive overview of the thermal performance of CPUs and GPUs in the DEEP-EST system. It should be noted that the DEEP-EST system provides a third module, the Data Analytics Module (DAM), which is however air-cooled and hence was not deemed as interesting as the CM and ESB for data collection.

2.6.1 Cluster Module - Sensor Data

This part of the segment is stored within the *deepest_temperature_cn* directory. The CM module comprises 50 compute nodes within a single rack, each equipped with two 12-core Intel Skylake Gold 6146 CPUs, 192GB of RAM and a Mellanox Infiniband interconnect. Each compute node is fitted with a 400GB NVMe SSD and employs the CentOS 7 operating system, while BeeGFS and GPFS instances support the main file system. Both the CM and the ESB module are warm-

water-cooled, while a set of gateway nodes manages communication between the two. According to the heterogeneity of CM and ESB nodes, the DCDB Pusher daemons running in these have different configurations. In the case of the CM, we use the following monitoring plugins with a sampling interval of 10s:

- *Perfevent*: samples a variety of performance counters on available CPUs.
- *SysFS*: samples node, CPU and RAM energy and temperature, as well as performance counters from the Infiniband interconnect.
- *ProcFS*: samples metrics from the *meminfo*, *vmstat* and *stat* files in the ProcFS virtual file system.

Sensor data is arranged on a per-socket basis, for a total of 32 sub-directories, one for each of the 2 sockets in each of the 16 compute nodes. Response files follow the same scheme. In this case, there is no CPU core-level data, but all sensors are at the socket level. A subset of sensors is at the compute node level, in order to provide a more complete picture - a total of 35 sensors for each CPU socket is thus present. In addition to this data we provide 10 sensors from the rack cooling unit associated with the nodes, collected via the *SNMP* plugin through a dedicated DCDB Pusher. Finally, the *control_experiments* directory contains additional sensors, both at the CPU and at the rack cooling unit level, describing a series of secondary inlet water temperature control experiments.

2.6.2 Cluster Module - Responses

We supply 32 response files, one for each socket of the 16 CM compute nodes, each stored in a separate sub-directory. These contain the CPU sockets' temperature time series, which can be used for the purpose of temperature prediction. A series of alternative responses files, named *responses_applications.csv*, contain the labels of the applications being executed: these include a numerical identifier of the application, as described in Table 5, plus a second numerical identifier indicating the specific configuration being used. This can be either 1, 2 or 3, corresponding respectively to the *normal*, *small* and *smaller* configurations. The two identifiers are separated by an underscore, and 0 is the idle state.

2.6.3 Extreme-Scale Booster - Sensor Data

The CPU and GPU-related ESB sections of the segment are respectively stored in the *deepest_temperature_esb* and the *deepest_temperature_gpu* directories. The ESB module is composed of 75 nodes hosted in 3 separate racks, which employ an 8-core Intel Cascade Lake Silver 4215 CPU supported by a Nvidia V100 GPU, coupled with 48GB of RAM and an Extoll or Infiniband interconnect. Like in the CM, each compute node relies on a 400GB NVMe SSD and uses the CentOS 7 operating system. The 16 compute nodes we use belong to the same ESB rack and are all equipped with an Infiniband interface. The two sections were acquired through a single experiment: here we use a very similar DCDB

Table 6: The main command-line and configuration parameters used for the CPU and GPU applications in the DEEP-EST segment associated with the ESB. Parameters that remain unaltered in smaller configurations are omitted.

Application	Configuration
Kripke	–groups 64 –niter 10 –gset 1 –quad 128 –dset 128 –legendre 4 –zones 256,128,128 –procs 4,2,2 **Small**: –zones 128,128,128 **Smaller**: –zones 128,128,64 **ID**: 1
AMG	-problem 2 -n 140 140 140 -P 4 2 2; *time_steps*=30 **Small**: -n 120 120 120 **Smaller**: -n 100 100 100 **ID**: 2
GPU LAMMPS	-v x 32 -v y 32 -v z 32 -in in.reaxc.hns **Small**: -v x 32 -v y 32 -v z 16 **Smaller**: -v x 32 -v y 16 -v z 16 **ID**: 3
GPU Quicksilver	-i "Coral2_P1.inp" -X 512 -Y 256 -Z 256 -x 512 -y 256 -z 256 -I 8 -J 4 -K 4 –nParticles 2621440 **Small**: -X 256 -Y 256 -Z 256 -x 256 -y 256 -z 256 **Smaller**: -X 256 -Y 256 -Z 128 -x 256 -y 256 -z 128 **ID**: 4
GPU HPL	N=3, Ns=192000 192000 192000, NBs=768, Ps=4, Qs=4 **Small**: Ns=172800 172800 172800 **Smaller**: Ns=141696 141696 141696 **ID**: 5
HPL	N=1, Ns=141696, NBs=384, Ps=4, Qs=4 **Small**: N=2, Ns=96000 96000 **Smaller**: Ns=86784 86784 **ID**: 6

configuration as for the CM section, with the addition of the DCDB *NVML* plugin which samples a series of performance metrics such as utilization, clock and energy from available GPUs. Like in the case of the CM, for both the CPU and GPU ESB sections we provide 10 additional sensors from the rack cooling unit associated with the compute nodes, collected via SNMP.

CPU Section. For the section of the dataset associated with ESB CPUs, all sensors are either at the compute node level (e.g., amount of used memory), or are associated with the only CPU in ESB nodes (e.g., instructions executed). Therefore, we find a total of 16 sub-directories, each associated with a specific compute node and CPU, and each containing 32 distinct sensors.

GPU Section. Like for the ESB CPU-related section, also for the GPU section of the ESB we find 16 different sub-directories, one for each compute node. Some of the available sensors are at the compute node level and are the same as in the previous sections - however, we replaced CPU-specific sensors (i.e., most performance counters) with GPU sensors describing utilization, performance and energy consumption. This leads to a total of 27 available sensors.

2.6.4 Extreme-Scale Booster - Responses

Like for sensor data, the response files of the CPU and GPU-related sections of the ESB dataset are stored in separate locations and have different formats.

CPU Section. Responses come in the form of 16 different files, one for each ESB compute node and stored in separate sub-directories, containing the temperature time series for the corresponding CPU, which can be once again used for prediction purposes. Like for the CM, alternative response files (named *responses_applications.csv*) with the labels of applications are available, using the naming scheme previously described and according to the identifiers in Table 6.

GPU Section. Responses follow the same format as for the ESB CPU section with 16 files, one for each compute node being employed, describing the temperature of the associated GPU. Unlike in the CM and ESB CPU sections, this is described in Celsius degrees rather than MilliCelsius. We find alternative response files for the labels of applications like in the ESB CPU section.

Acknowledgements. This research activity has received funding from the DEEP-EST project under the EU H2020-FETHPC-01-2016 Programme grant agreement no. 754304. We'd like to thank MEGWARE GmbH for granting us access to infrastructure data on the DEEP-EST HPC system.

References

[1] Agelastos, A., Allan, B., Brandt, J., Cassella, P., et al.: The lightweight distributed metric service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications. In: Proc. of SC 2014. pp. 154–165. IEEE (2014)

[2] Ates, E., Tuncer, O., Turk, A., Leung, V.J., et al.: Taxonomist: Application detection through rich monitoring data. In: Proc. of Euro-Par 2018. Springer (2018)

[3] Conficoni, C., Bartolini, A., Tilli, A., Tecchiolli, G., et al.: Energy-aware cooling for hot-water cooled supercomputers. In: Proc. of DATE 2015. pp. 1353–1358. IEEE (2015)

[4] Netti, A., Kiziltan, Z., Babaoglu, O., Sirbu, A., et al.: Online fault classification in hpc systems through machine learning. In: Proc. of Euro-Par 2019. pp. 3–16. Springer (2019)

[5] Netti, A., Mueller, M., Auweter, A., Guillen, C., et al.: From facility to application sensor data: Modular, continuous and holistic monitoring with DCDB. In: Proc. of SC 2019. ACM (2019)

[6] Netti, A., Tafani, D., Ott, M., Schulz, M.: Correlation-wise smoothing: Lightweight knowledge extraction for hpc monitoring data. Proc. of IPDPS 2021 (2021)

[7] Ozer, G., Garg, S., Davoudi, N., Poerwawinata, G., et al.: Towards a predictive energy model for hpc runtime systems using supervised learning. In: Proc. of PMACS Workshop 2019. Springer (2019)