**Data driven Computational Mechanics at EXascale**

**Data driven Computational Mechanics at EXascale**

**Work program topic: EuroHPC-01-2019**

**Type of action: Research and Innovation Action (RIA)**

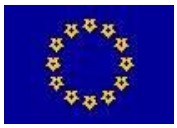
# System requirements, design and architecture (report)

**DELIVERABLE D.6.1**

**Version No 1**

http:// dcomex.eu/

**This project has received funding from the European High-Performance Computing Joint Undertaking Joint Undertaking ('the JU'), under Grant Agreement No 956201**



# DOCUMENT SUMMARY INFORMATION

| | |
|---|---|
| **Project Title** | **Data driven Computational Mechanics at EXascale** |
| **Project Acronym** | DCoMEX |
| **Project No:** | 956201 |
| **Call Identifier:** | EuroHPC-01-2019 |
| **Project Start Date** | 01/04/2021 |
| **Related work package** | WP 6 |
| **Related task(s)** | Task 6.1, 6.2, and 6.3 |
| **Lead Organisation** | ETHZ/CSELab |
| **Submission date** | 27/9/2021 |
| **Re-submission date** | |
| **Dissemination Level** | PU |

**Quality Control:**

| | Who | Affiliation | Date |
|---|---|---|---|
| **Checked by internal reviewer** | Vissarion Papadopoulos | NTUA | 31/09/2021 |
| **Checked by WP Leader** | Sergio Martin | ETHZ/CSELab | 31/09/2021 |
| **Checked by Project Coordinator** | Vissarion Papadopoulos | NTUA | 31/09/2021 |

**Document Change History:**

| Version | Date | Author (s) | Affiliation | Comment |
|---|---|---|---|---|
| 1.0 | 27.09.2021 | S. Martin, B. Cumming | ETHZ/CSELAB, ETHZ/CSCS | |
| | | | | |

# Contents

# 1. Introduction

This report provides a preliminary definition of the prerequisite software and hardware environments required by the DCoMEX project. Here, we define the testing and benchmarking technologies, frameworks, and methodologies that will be utilised throughout the project. In particular, we define:

The design and architecture of the systems (supercomputers) to be used:

- Node architecture, GPU capability, and memory hierarchy.
- Network architecture to be used

The main software components, including:

- Their own software and hardware requirements.
- An analysis of their compute components, following the principles of the Berkeley Dwarfs.
- Software libraries, modules, and frameworks we will require
- Energy considerations

# 2. Hardware Architecture

The DComEX framework will be provided on HPC systems that are available to scientific and industrial users in Europe, which will dictate the node and network architecture that DComEX will target. Here we will present a brief overview of the architecture of systems that will be used for developing the framework, and for the prototype and final releases of the framework in M15 and M24 respectively.

## *1. EuroHPC*

In order to understand the HPC architectures that the target audience of DComEX is likely to run on, we will look at the HPC systems being procured by the European High Performance Computing Joint Undertaking (EuroHPC JU[1]).

[1] https://eurohpc-ju.europa.eu/discover-eurohpc-ju

EuroHPC is a legal and funding entity, designed to allow the European Union and participating countries to coordinate their efforts and pool their resources to make Europe a world leader in supercomputing. The first of EuroHPCs stated aims is to:

> *"Develop, deploy, extend, and maintain a world-leading supercomputing and data infrastructure in Europe*."

The infrastructure provided by EuroHPC serves the audience of academic and industrial users that DComEX targets, given that they have the stated aim:

> *"Widen the use of HPC infrastructures to a large number of public and private users wherever they are located in Europe and support the development of key HPC skills, education and training for European science and industry. One of the objective is to create a network of national HPC Competence Centres to ease access to European HPC opportunities in different industrial sectors and deliver tailored solutions."*

Given the above aims, the systems that EuroHPC has and will fund and are a useful proxy for they architecture that DComEX will support to be accessible to its target audience.

To date, EuroHPC has procured 7 systems, 5 smaller petascale systems that (4 already available to users) and two larger pre-exascale systems which are to be delivered in the first half of 2022. See Table 1, which summarizes key statistics of the 7 EuroHPC systems procured to date, and the Alps system in Switzerland. The PFlops provided by the GPU partition on Alps is not published, but will be of the order of the EuroHPC pre-exascale systems. Performance marked with ^ indicate projections, for systems yet to be installed and benchmarked. Information about the systems was collected from the EuroHPC website, and from the top500 list[2] for June 2021.

| **Name** | **Host** | **Date** | **Vendor** | **GPU-arch** | **CPU-arc** | **GPU/CPU PFlops** | **Network** |
|---|---|---|---|---|---|---|---|
| Alps | CSCS Switzerland | H1 2023 | HPE | NVIDIA Ampere Next | AMD Epyc | >>100^/3 | Cray Slingshot |

[2] https://www.top500.org/lists/top500/2021/06/

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| LUMI | CSC<br>Finland | H1 2022 | HPE | AMD Instinct | Intel Xeon | 375^/5^ | Cray Slingshot |
| Leonardo | CINECA<br>Italy | H1 2022 | Atos | NVIDIA A100 | AMD Epyc | 249^/9^ | HDR Infiniband |
| MeluXina | Luxembourg | Q2 2021 | Atos | NVIDIA A100 | AMD Epyc | 10/2 | HDR Infiniband |
| Karolina | Czech Republic | Q2 2021 | Atos | NVIDIA A100 | AMD Epyc | 6/3 | HDR Infiniband |
| Vega | Slovenia | Q2 2021 | Atos | NVIDIA A100 | AMD Epyc | 3/4 | HDR Infiniband |
| Discoverer | Bulgaria | Q2 2021 | Atos | - | AMD Epyc | -/4.4 | HDR Infiniband |
| Deucalion | MACC<br>Portugal | Unknown | Fujitsu | NVIDIA A100 | Fujitsu A64FX | 1.7/3.8 | TofuD |

*Table 1*

## 2. Node Architecture

From the system descriptions in Table 1, we make the following observations:

- All 8 systems have a CPU-only partition;
- 7 of the 8 systems have hybrid partitions with nodes that nave both CPU and GPU:
- The 2 smallest systems have more computational power, as measured in PFlops, in the CPU partition;
- The other five have between 5-75 x more PFlops from GPU than CPU, and this ration increases for larger systems.
- A total 95% of FLops available on the systems will be provided by the GPU partitions.

We note that the Flops from the HPL benchmark, traditionally used to compare HPC systems, is not the only metric for system performance. For examples, a benchmark based on memory bandwidth (c.f. HPCG) is more suitable for some applications. However, while the exact ration between GPU and CPU performance will vary according to the metric used, the overall trend of the vast majority of computational throughput being provided by GPUs will be unchanged.

GPUs are an essential part of the computational service offered by HPC systems, and furthermore there is a clear trend towards GPUs providing an increasing proportion, with over 95% on the larger pre-exascale systems in 2022 and 2023.

Support for running workloads on CPU is also required for users who don't have access to GPU-enabled systems, or who whose have models that are not amenable to GPU implementation. However, to ensure that the DComEX framework is available to users of large systems, can utilize more than 5% of the available compute in Europe, and can run the most computationally demanding workloads, support for GPUs is a requirement.

Not all GPU node configurations are identical:

- there are two GPU vendors (AMD and NVIDIA)
- the ratio between GPUs and CPU sockets varies between 4:1 and 8:1.
- most an x86 CPU, provided by either AMD Epyc or Intel Xeon. The Alps GPU nodes will have an ARM CPU manufactured by NVIDIA.

The interconnects on each system (Mellanox HDR Inifiband, Cray Slingshot and TofuD) are state of the art, with tight integration between the on node CPU, GPU and memory. Each system integrator collaborates with the CPU, GPU and interconnect vendors to provide optimized MPI implementations, which makes MPI the obvious choice for communication. We may investigate using more low level interfaces and libraries, such as NCCL[3], RCCL[4] or UCX[5], however these are less portable and more low level, and should only be targeted if needed.

# 3. Software Components

In this section, we describe the main software components, their specifications, and their requirements:

## *1. Korali*

Korali is a high-performance framework for Bayesian UQ, optimization, and reinforcement learning. Korali's multi-language interface allows the execution of any type of computational model, either sequential or distributed (MPI), C++ or Python, and even pre-compiled/legacy

[3] https://developer.nvidia.com/nccl
[4] https://github.com/ROCmSoftwarePlatform/rccl
[5] https://openucx.org/

applications. Korali's execution engine enables scalable sampling on large-scale HPC systems. Next, we detail the relevant aspects of this software:

## 1. Relevant Links

- Web: https://www.cse-lab.ethz.ch/korali/
- Github: https://github.com/cselab/korali
- Readthedocs: https://korali.readthedocs.io/en/master/
- CircleCI: https://app.circleci.com/pipelines/github/cselab/korali

## 2. Development Status

- TRL 8 - Fully operational, used in scientific studies.
- Code has >95% test coverage on multiple OS (docker) images and compilers.
- Code is fully documented (doxygen) and user manual is available on read the docs.

## 3. Hardware Support

- NVIDIA - Optionally, when using the CUDNN neural network backend, it requires access to NVIDIA GPU devices.
- CPU Intensive computation (Dense Linear Algebra mostly) using GSL, Eigen3, and OneDNN
- Local Parallelism - It employs local parallel sampling through fork/join (multiprocessing). Some ML kernels in Korali use OpenMP for acceleration.
- Distributed Parallelism - It employs MPI for distributed sampling.

## 4. Dependencies

- Python3 - For the python interface. Additionally: pip, pybind11, python3-config
- GSL - For random distribution sampling and some Dense LA operations.
- Eigen3 - For Dense LA operations and NN backend.
- (Optional) MPI - For distributed sampling
- (Optional) OneDNN - For faster NN backend.
- (Optional) CUDA RT, CuDNN - For GPU-based NN backend.
- Compiler: GCC or clang, with support for C++17 or newer.
- Building: Meson, Cmake

5. Platform Support

- Linux: Compiling and tested on Fedora and Ubuntu
- MacOS: Compiling
- Windows: No support yet

6. Testing

- All commits to Pull Requests are automatically tested on CircleCI on a variety of Linux systems and compilers (the list of setups is under development)

7. Planned Development

- Adding graph-like Bayesian networks to define complex prior dependencies.
- Adding new RL methods
- Improve testing/documentation overall
- Integration with comprehensive neural network engines like pyTorch/TensorFlow

8. Development Resources

- Sergio Martin (Postdoc, 100%) leads the overall project (design, testing, development). He also develops the Reinforcement Learning side.
- George Arampatzis (Postdoc, 100%) develops the Bayesian UQ and surrogate modelling side.

CSELab members that also collaborate on the development of new ML and BUQ methods:

- Daniel Waelchli
- Pascal Weber

## *2. MSolve*

MSolve, is a general-purpose computational mechanics solution platform with multiphysics and multiscale capabilities, developed at NTUA, that exploits combined multi-core central processing units (CPU) and graphics processing units (GPU).

1. Relevant Links

- Github: https://github.com/mgroupntua

- Documentation: https://mgroupntua.github.io/

## 2. Development Status

- TRL 6 - Under refactoring, most unit tests work.

## 3. Hardware Support

- CPU Intensive computation (Dense Linear Algebra mostly) using MKL
- Local Parallelism - It employs local parallel sampling through fork/join (multiprocessing).

## 4. Dependencies

- (Optional) SuiteSparse - For linear algebra module
- (Optional) Intel MKL - For linear algebra module
- Compiler: Roslyn, support for C# 7.0 and higher.

## 5. Platform Support

- Linux: Compiling
- MacOS: Compiling
- Windows: Compiling

## 6. Testing

- No CI yet, unit tests are embedded in each module

## 7. Planned Development

- Incorporation of GPU modules
- Testing and improving MPI
- Improve testing/documentation overall

## 8. Development Resources

- George Stavroulakis (Postdoc), software architect, HPC, domain decomposition
- Serafeim Bakalakos (PhD), HPC, linear algebra, domain decomposition, X-FEM, optimization
- Thofilos Christodoulou (PhD), machine learning

MGroup members that also collaborate on the development:

- Yannis Kalogeris - machine learning, stochastic processes
- Gerasimos Sotiropoulos - multiscale analysis
- Stefanos Nikolopoulos - machine learning
- Ambrosios Savvidis - constitutive modeling
- Kostas Margaronis - contact mechanics
- Stefanos Pyrialakos - AI-enhanced constitutive modeling

## *3. Dwarf Analysis*

Korali's sampling engine applies monte-carlo sampling which, in principle, is *embarrassingly parallel*. This, however, assumes that all samples take similar time. Most probably, the execution of DCoMEX pipelines will involve highly irregular sampling times (combination of surrogate + actual simulation environments). Fortunately, Korali contains load balancing mechanisms that help in maximising performance in such cases, especially for hierarchical Bayesian problems.

For Neural Network-intensive operations, such as deep learning (required for the AI-Solve module) or surrogate modelling, we expect a high usage of dense linear algebra (matrix/matrix multiplication) operations. Both our main software component employ high-performance linear algebra (Intel MKL, Eigen3) and neural networking libraries (oneDNN, cuDNN) that should extract the most performance from the supercomputing systems we will employ.

The non-surrogated simulation of physical systems through MSolve will employ mostly dense linear algebra operations, and most probably also structured grid motifs. The first will be handled mainly by the use of the Intel MKL library. The second will depend on manual implementation of solver kernels. We understand MSolve makes use of threading, maximizing CPU usage.

## *4. Modules and Library Depenencies*

Through our investigation of the software libraries, we have found that mostly the default libraries and frameworks for scalable parallelism will be used by all the components and their interaction.

a. **MPI for distributed parallelism**. Korali's sampling engine runs entirely on MPI to efficiently scale to many nodes in a supercomputer[6]. MSolve nor the neural network-based components use distributed parallelism by themselves.
b. **PGAS parallelism** libraries such as UPC or Co-Array FORTRAN will not be employed, as MPI will suffice for all distributed parallelism.
c. **OpenMP, TBB, Cilk, OMPSS for Local Parallelism**. As MSolve uses Intel's MKL, it is likely that they will benefit from Intel's Thread Building Blocks (TBB) to drive their in-node performance. On the other hand, oneDNN relies on OpenMP for threaded Neural Network operations.
d. **Hybrid MPI+OpenMP approach**. It is likely from the above, that DCoMEX pipelines will benefit from the combined Distributed+Local parallelism by means of MPI+OpenMP or MPI+TBB, as Korali will distribute samples across nodes via MPI while MSolve and OneDNN will execute fully threaded.
e. **CUDA, OpenCL, OpenACC for GPU support**. At this point, We have not found yet any applications or software that requires the use of GPU devices. However, as indicated by the authors of MSolve, this will change in the near future, when they implement support for GPU-based kernels to accelerate their solvers.

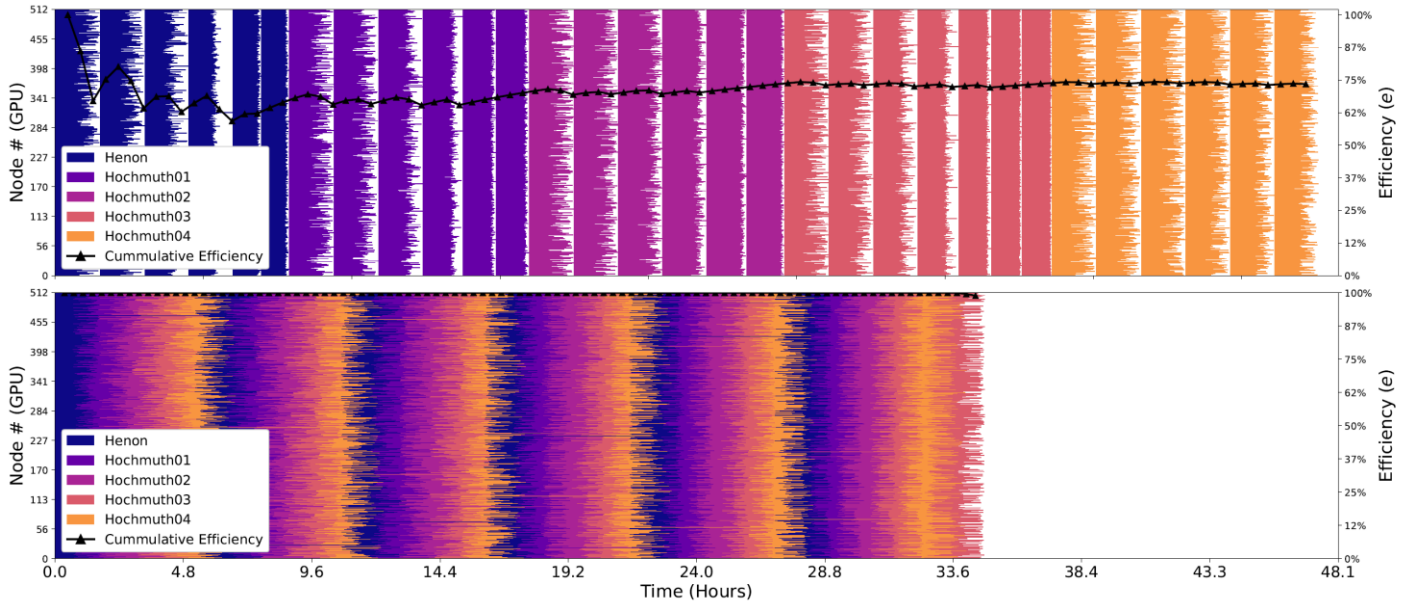
# 4. Energy Considerations

From a software perspective, the best approach to reduce the energy consumption of our frameworks is to maximize the node usage efficiency. This can be done twofold:

a. Reduce the amount of time an allocated node remains unused.
b. Improve the efficiency and time-to-solution of the solver.

For (a), our early results with the Korali framework[6] have shown that executing concurrent experiments in a single job allocation maximizes node usage efficiency and reduces time-to-solution. The next figure compares the per-node (512 nodes in total) usage of (top) a sequential approach (one experiment after the other), vs. (bottom) Korali's approach:

[6] Korali: Efficient and Scalable Software Framework for Bayesian Uncertainty Quantification and Stochastic Optimization" https://arxiv.org/abs/2005.13457




The sequential execution of sampling experiments took 48.1 hours to complete on 512 CSCS Piz Daint nodes, requiring a total of 24.6k node hours. This approach yielded sampling efficiency($e$) of 72.7%. It follows that nodes remained idle 27.3% of their running time. Therefore, this approach resulted in a loss of 6.6k (idle) node hours. In total, the energy usage, as reported from the supercomputer's job scheduler, was of 10.45 GJ. To alleviate the effect of load imbalance, we configured

On the other hand, when Korali has to schedule all five experiments simultaneously, the 512 nodes remained busy during most of the run with the concurrent experiment variant. The results, summarized in Table 2, indicate that this approach yields a superior efficiency (98.9%) compared to the former approach, wasting much fewer node hours (186), as well as requiring less energy (7.80 GJ). Furthermore, it also reduced the run-to-completion time from 47.32 to 34.78 hours.

*Table 2 - Comparison of Energy Consumption between Sampling Scheduling Methods*

| | | Node Hours | | | |
|---|---|---|---|---|---|
| **Scheduling** | **Time** | **Total** | **Idle** | $e$ | **Energy** |
| **Single** | 47.32 h | 24227 | 6604 | **72.7%** | 10.45 GJ |
| **Multiple** | 34.78 h | 17809 | 186 | **98.9%** | 7.80 GJ |

For (b), it is necessary to optimize the MSolve kernel solver to reduce the per-sample execution time. The project members at GRNet and NTUA are planning to implement new kernels in MSolve to make the most of the GPU environments.

Finally, by the time AISolve is operational, we expect sampling time to be greatly reduced as this surrogate will involve the forward propagation of a given state on a Neural Network, a few dozen $O(N^3)$ dense linear algebra operations. If successful, this approach will prove that surrogate modelling will provide tremendous potential in reducing not only time-to-solution, but also the energy requirements of large-scale computational applications.

# 5. Scalability Considerations

The Korali component has proven[6] to be able to scale to large number of nodes in the CSCS Piz Daint supercomputer, maintaining a higher efficiency than the main competitors. The next figure shows a scaling study based comparing the efficiency at different node scales.

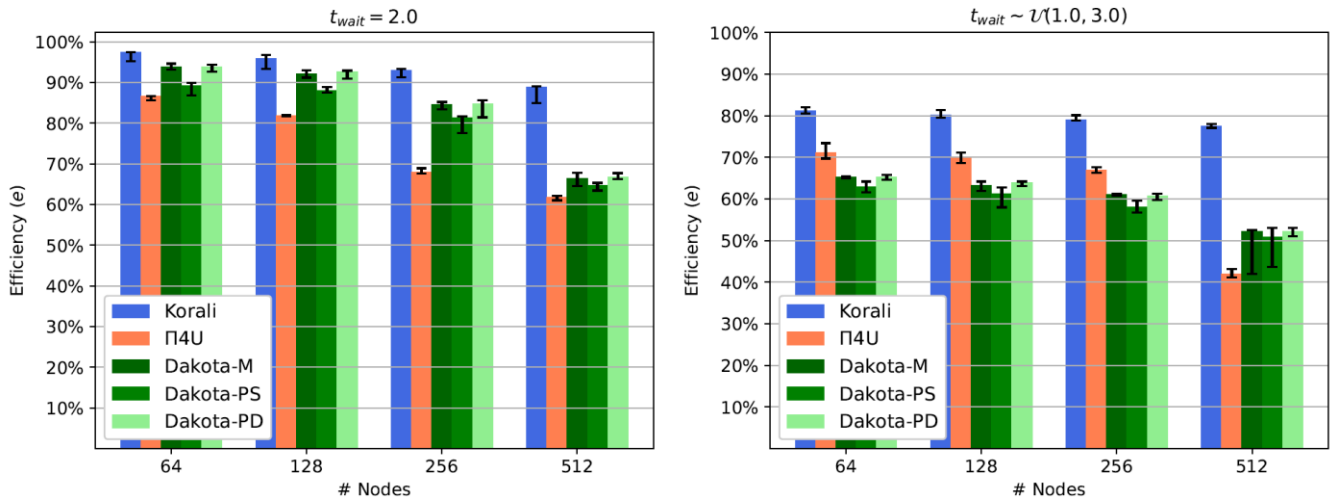



**Figure 10:** Weak scaling studies comparing the sampling efficiency ($e$) of Korali, Π4U, and Dakota (three variants) for a synthetic benchmark (left) without ($T_{wait} = 2.0$), and (right) with load imbalance ($T_{wait} \sim \mathcal{U}(1.0, 3.0)$) on 64, 128, 256, and 512 nodes. The efficiency is calculated as the ratio of busy and total running time, i.e. busy and idle time combined. The colored bars highlight the median efficiencies, and the black intervals indicate the maximum and minimum.

The synthetic benchmark runs a single-variable optimization experiment on a computational model that passively waits for a given number of seconds ($T_{wait}$) before returning a random result. By employing a passive wait, we can fix the running time of each sample, ruling out time variances common to compute-intensive computational models. To drive sampling, we employ evolutionary

optimization algorithms (CMAES[7], for Korali and Π4U[8], and; COLIN-EA[9] for Dakota[10]) configured to run 5 generations, $4N$ samples per generation3, where $N$ is the number of workers, and one node per worker. This configuration allows to conduct weak scaling studies, evaluating the impact of node scaling on efficiency, while keeping the workload per worker constant.

The sample times were configured to add up to 40 seconds in total per experiment, which represents their ideal running time. Efficiency ($e$) is measured for each framework by dividing the ideal time by its actual running time. The following 5 variants were tested: Korali, Π4U, Dakota- M (master/worker scheduler), Dakota-PS (static scheduler), and Dakota-PD (dynamic scheduler). The synthetic benchmark drove two weak scaling studies, with and without load imbalance. To account for the effect of stochastic waiting times, 10 repetitions of each experiment were performed.

The first study represents a scenario where there is no load imbalance ($T_{wait}$ = 2.0 seconds, for all samples). This measures the inherent efficiency of the frameworks in distributing samples to workers without the detrimental effect of imbalance. The gap between the attained and an ideal efficiency therefore illustrates the time spent on communication, I/O operations, and scheduling overhead only. Fig. 10 (left) shows the results of weak scaling by running the 5 variants from 64 to 512 nodes of the Piz Daint supercomputer. All variants provide high efficiencies (86% < $e$ < 97%) at smaller scales (64 and 128 nodes), with Korali as the most efficient by a small margin. At the largest scale (512 nodes), the differences are evident, since both Π4U ($e$ = 62%) and Dakota ($e$ = 67%) appear to be especially susceptible to the increasing scheduling costs, compared to Korali ($e$ = 86%).

The second study simulates experiments with a high load imbalance. Here, the waiting time for each sample is drawn from a random variable $T_{wait} \sim$ U(1.0, 3.0) seconds. We consider the same ideal case (in average, 40 seconds per experiment) as basis for the calculation of efficiency. Fig. 10 (right) show the results for this study. We observe that load imbalance plays a detrimental effect

---

7 N. Hansen, The CMA Evolution Strategy: A Tutorial (2016).arXiv:1604.00772.

8 P. Hadjidoukas, P. Angelikopoulos, C. Papadimitriou, P. Koumoutsakos, Π4U: A high performance computing framework for Bayesian uncertainty quantification of complex models,Journal of Computational Physics 284 (2015) 1–21

9 W. E. Hart, An introduction to the COLIN optimization interface.,Tech. rep., Sandia National Laboratories (2003).

10 K. Dalbey, M. S. Eldred, G. Geraci, J. D. Jakeman, K. A.Maupin, J. A. Monschke, D. T. Seidl, L. P. Swiler, A. Tran,F. Menhorn, et al., Dakota A Multilevel Parallel Object- Oriented Framework for Design Optimization Parameter Estimation Uncertainty Quantification and Sensitivity Analysis:Version 6.12 Theory Manual., Tech. rep., Sandia National Lab(2020).

on the efficiency of all variants. However, Korali is the least affected of them throughout all scales. We observe the larger difference between the variants when running on 512 nodes and the larger imbalance, where Π4U and Dakota show a low efficiency ($e$ = 41% and $e$ = 52%, respectively), while Korali sustains a higher performance ($e$ = 78%).