E-CAM Software Porting and Benchmarking Data III

E-CAM Deliverable 7.6
Deliverable Type: Report
Delivered in April, 2019

E-CAM
The European Centre of Excellence for
Software, Training and Consultancy
in Simulation and Modelling

Funded by the European Union under grant agreement 676531

Project and Deliverable Information

Project Title	E-CAM: An e-infrastructure for software, training and discussion in simulation and modelling
Project Ref.	Grant Agreement 676531
Project Website	https://www.e-cam2020.eu
EC Project Officer	Juan Pelegrín
Deliverable ID	D7.6
Deliverable Nature	Report
Dissemination Level	Public
Contractual Date of Delivery	Project Month 36(1st October, 2018)
Actual Date of Delivery	30th April, 2019
Description of Deliverable	Joint technical report on results of (a) porting and optimisation of at least 8 new modules related to those developed in the ESDWs to massively parallel machine (STFC); and (b) benchmarking and scaling of at least 8 new modules related to those developed in the ESDWs on a variety of architectures (Juelich).

Document Control Information

Document	Title:	E-CAM Software Porting and Benchmarking Data III
	ID:	D7.6
	Version:	As of April, 2019
	Status:	Accepted by WP Leader
	Available at:	https://www.e-cam2020.eu/deliverables
		with citable version on the E-CAM Zenodo Community page
	Document history:	Internal Project Management Link
Review	Review Status:	Reviewed
	Action Requested:	Submit
Authorship	Written by:	Alan O'Cais(Juelich Supercomputing Centre)
	Contributors:	Jony Castagna (STFC)
	Reviewed by:	Godehard Sutmann (Juelich Supercomputing Centre)
	Approved by:	Godehard Sutmann (Juelich Supercomputing Centre)

Document Keywords

Keywords:	E-CAM, HPC, CECAM, Materials

30th April, 2019

***Disclaimer**:This deliverable has been prepared by the responsible Work Package of the Project in accordance with the Consortium Agreement and the Grant Agreement. It solely reflects the opinion of the parties to such agreements on a collective basis in the context of the Project and to the extent foreseen in such agreements.*

[1] a.ocais@fz-juelich.de

Contents

Executive Summary **1**

1 Introduction **2**

2 Workflow **3**
2.1 Tools . . . 3
2.1.1 Software Builds - EasyBuild . . . 3
2.1.2 Benchmarking - JUBE . . . 3
2.1.3 Optimisation - Scalasca . . . 4
2.2 Interplay with ESDWs . . . 4

3 Porting and Optimisation **5**
3.1 Available Resources . . . 5
3.1.1 Primary Resources . . . 5
3.1.2 PRACE Resources . . . 5
3.2 Porting Effort . . . 5
3.2.1 Improving build reproducability . . . 6
3.2.2 Improving ability to easily switch toolchains . . . 6
3.2.3 Updating the software stack . . . 6
3.2.4 Improving architecture awareness for Autotools packages . . . 6
3.2.5 Porting LAMMPS+Kokkos to EasyBuild . . . 6
3.2.6 Building CP2K as a library . . . 7
3.3 HTC Optimisation Collaboration with PRACE . . . 7

4 Modules and Application Codes **8**
4.1 WP1: Classical Molecular Dynamics . . . 8
4.1.1 Relevance for E-CAM . . . 8
4.1.2 High Throughput Computing (HTC) with `Dask-jobqueue` . . . 9
4.1.3 LAMMPS as a task . . . 12
4.2 WP2: Electronic Structure . . . 13
4.2.1 Relevance for E-CAM . . . 13
4.2.2 The Electronic Structure Library (ESL) bundle and the ESL demonstrator . . . 13
4.3 WP3: Quantum Dynamics . . . 14
4.3.1 Relevance for E-CAM . . . 14
4.3.2 PaPIM . . . 14
4.3.3 Surface Hopping Propagator . . . 15
4.4 WP4: Meso- and Multi-scale Modelling . . . 16
4.4.1 Relevance for E-CAM . . . 16
4.4.2 Scaling of DL_MESO_DPD on GPU on Piz Daint . . . 17
4.4.3 Bond forces to DL_MESO_DPD GPU version . . . 18
4.4.4 Benchmarking GC-AdResS on Jureca . . . 19

5 Outlook **20**

References **21**

List of Figures

1 A Performance Optimisation Loop . . . 4
2 Total overhead of the framework in seconds for various numbers of tasks and on various architectures. . . 11
3 Overhead of the framework per task in seconds for various numbers of tasks and on various architectures. 11
4 Initial performance analysis of sample usage of the ESL demonstrator. . . . 13
5 PaPIM performance on JUQUEEN up to 131,072 CPUs (and 262,144 MPI tasks(without CP2K integration. The parallel efficiency on the X-axis is the time per sample relative to the most time-efficient result, the Y-axis is the node count (with 16 physical cores per node). . . . 14
6 Graphical representation of the MPI split communicator scheme used in parallelization of `PaPIM-CP2K_interface` module. . . . 15
7 QC single path MPI benchmark . . . 16
8 QC single path OpenMP benchmark . . . 16

9 Distribution of execution time of Surface Hopping Propagator for 96 MPI tasks. 17
10 DL_MESO_GPU weak scaling up to 512 GPUs. 17
11 DL_MESO_GPU strong scaling up to 2048 GPUs 18
12 Ternary system with bond force across phases. 18
13 Gc-AdresS system made of coarse and fine particles. 19

List of Tables

1 System configuraton time per job on each of the hardware types available on JURECA 11
2 Time savings (in seconds) of running tasks through the library rather than through the resource manager 12
3 Strong scaling for GcAdresS on JURECA 19

Executive Summary

The purpose of the current document is to deliver a joint technical report on results of the initial porting and optimisation of 8 new E-CAM modules to massively parallel machines and their benchmarking and scaling on a variety of architectures. The development of the modules was done in the context of the E-CAM program of Extended Software Development Workshop (ESDW) events.

The particular list of all relevant applications that were investigated were:

- for Classical Molecular Dynamics:
  `jobqueue_features`, a High Throughput Computing library developed by E-CAM. The associated modules were developed in the context of the ESDW "Intelligent High Throughput Computing for Scientific Applications".
- for Electronic Structure:
  The ESL demonstrator which is built from the components of the ESL bundle. The associated modules were developed in the context of an ESDW on scaling electronic structure applications.
- for Quantum Dynamics:
  CP2K integration into PaPIM code, and the new Surface Hopping code. The associated modules were developed in the context of an ESDW in Quantum Dynamics.
- for Meso- and Multi-scale Modelling:
  DL_MESO_DPD multi-GPU support, and GROMACS implementation of GC-AdResS. The associated modules were developed in relation to an ESDW in Meso and multiscale modeling.

For the `jobqueue_features` HTC library, PaPIM, and GC-AdResS; the modules presented in this deliverable represent the incorporation or use of external, scalable community code (in particular LAMMPS, CP2K and GROMACS) as libraries or test-beds. We have looked at the scalability of these community codes in previous iterations of this deliverable [2] and do not repeat this effort here. Since these applications are the computational workhorses, we rather investigate the overhead incurred by their incorporation. The HTC library developed by E-CAM is shown to have very low overhead with the potential for significant time (and CPU) savings for appropriate applications. The CP2K integration in PaPIM has been verified and a scientific use case that takes this combination to extreme scale is under preparation. For the GC-AdResS implementation in GROMACS, we find that the automated load-balancing of GROMACS does not deal well with the adaptive resolution scheme and scalability is quite poor as a result. The incorporation of the scheme into ESPResSo++ is being considered and is likely to benefit from the load balancing library also being developed by E-CAM.

For the ESL bundle and demonstrator, we see there is still some improvement to be made to the scalability of the demonstrator, which we hope to be further addressed in the upcoming second part of the relevant ESDW[3]. We only show here some initial assessments of the ESL demonstrator (which is built on top of the ESL bundle).

We find that the Surface Hopping code is quite scalable but suffers from a similar problem to the previous iteration of PaPIM: there is insufficient computational work to keep cores busy and MPI overheads can dominate as a result.

A significant success story has been the multi-GPU developments undertaken for DL_MESO_DPD. This has been shown to be scalable out to 2048 Tesla P100 GPUs, which is equivalent to almost 10 Petaflops of raw double precision compute performance.

[2] see [1]

[3] Initial event was held in January 2019, second part is planned for September 2019

1 Introduction

The purpose of the current deliverable is to present a joint technical report on results of porting and optimisation of at least 8 modules which were developed in relation to the ESDW events concerned with massively parallel machines, and the benchmarking and scaling of at least 8 modules out of those related to the ESDW events on a variety of architectures.

The associated applications have been ported to EasyBuild (the tool that delivers compiler/hardware portability for E-CAM applications) where the installation and dependency tree of the applications were optimised (described in Section 3). Increasing importance is being given to the handling of application "dependency hell", i.e., trying to provide a self-consistent software stack for multiple applications.

The modules and applications were then benchmarked on the High Performance Computing (HPC) resources available to the project and scaling plots were generated for a variety of relevant systems and architectures (detailed in Section 4).

While being part of an overall series, this deliverable is intended to stand alone (Section 2, in particular, includes only minor updates to the workflow that was originally described in Deliverable 7.2[1]).

In this deliverable, we have chosen to include software applications from each of the research Work Package (WP) (of which there are 4) where a minimum of 1 module developed in relation to an ESDW targets each application.

2 Workflow

In this section we describe the workflow of the programming team which is led by the Software Manager (at partner Jülich Supercomputing Centre (JSC)) and includes the programmers hired within the project. We also discuss the interplay between the services that E-CAM can offer, the tools that are used and the applications of the E-CAM community.

The essential elements in the workflow are:

- reproducible and efficient software builds,
- benchmarking,
- optimisation.

The implementation and tuning of this workflow is an ongoing process and requires significant collaboration with the organisers of ESDW events.

2.1 Tools

Each element of the workflow involves a different tool. At each stage there are multiple choices of tools but we choose within E-CAM to use only a single option (while maintaining awareness of other possibilities). When describing each tool here we also describe the context of its use.

2.1.1 Software Builds - EasyBuild

In order for the information that we gather to be useful to our end user community, that community needs to be able to easily reproduce a similarly optimised build of the software. EasyBuild is a software build and installation framework that allows the management of (scientific) software on HPC systems in an efficient way. The main motivations for using the tool within E-CAM are that:

- it provides a *flexible framework* for building/installing (scientific) software,
- it fully automates software builds,
- it allows for easily reproducing previous builds,
- it keeps the software build recipes/specifications simple and human-readable,
- it enables *collaboration* with the application developers and the wider HPC community,
- it provides an automated *dependency resolution* process.

EasyBuild currently supports cluster and Cray supercomputing systems, with limited support for BG/Q systems (this limitation is no longer significant since the architecture is no longer developed).

In our use case, we will produce a build of the software under study with an open source toolset (GCC compiler, OpenMPI MPI implementation, open source math libraries) for use by the community and the build procedure will be described in sufficient detail in the modules associated to the software package.

2.1.2 Benchmarking - JUBE

Automating benchmarks is important for reproducibility and hence comparability between builds of software, which is the major goal. Furthermore, managing different combinations of parameters is error-prone and often results in significant amounts of work especially if the parameter space becomes large.

In order to alleviate these problems JUBE helps to perform and analyse benchmarks in a systematic way. It allows the creation of custom work flows that can be adapted to new architectures.

For each benchmark application the benchmark data is written out in a particular format that enables JUBE to deduce the desired information. This data can be parsed by automatic pre- and post-processing scripts that draw information, and store it more densely for manual interpretation.

The JUBE benchmarking environment provides a script based framework to easily create benchmark sets, run those sets on different computer systems and evaluate the results.

Where relevant, we will use JUBE to provide a means for the community to evaluate the performance of their build of the software under study.

Collaboration with EoCoE

The E-CAM programmers and software manager attended the 3rd EoCoE/POP Workshop on Performance Analysis in Barcelona. We have decided to adopt the JUBE performance evaluation workflow that EoCoE has created with appropriate adaptations to the needs of E-CAM.

2.1.3 Optimisation - Scalasca

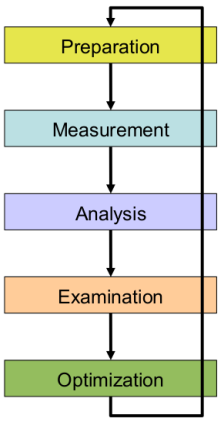

Figure 1: A Performance Optimisation Loop

Scalasca is a software tool that supports the performance optimisation of parallel programs by measuring and analyzing their runtime behavior. The analysis identifies potential performance bottlenecks – in particular those concerning communication and synchronization – and offers guidance in exploring their causes.

The Scalasca Trace Tools developed at the Jülich Supercomputing Centre are a collection of trace-based performance analysis tools that have been specifically designed for use on large-scale systems such as the IBM Blue Gene series or Cray XT and successors, but also suitable for smaller HPC platforms. While the current focus is on applications using MPI, OpenMP, POSIX threads, or hybrid parallelization schemes, support for other parallel programming paradigms may be added in the future. A distinctive feature of the Scalasca Trace Tools is its scalable automatic trace-analysis component which provides the ability to identify wait states that occur, for example, as a result of unevenly distributed workloads.

Scalasca is used as part of the JUBE performance evaluation workflow mentioned in the previous section.

Collaboration with POP

In addition to our own optimisation efforts we have been engaged with the POP Centre of Excellence for their support on two of the applications that appeared in previous iterations of this deliverable: PaPIM and ESPResSo++. Further details are included where relevant.

2.2 Interplay with ESDWs

As described in the ESDW guidelines (updated in Deliverable D5.4[2]), it is expected that the applications to be used in ESDW events are known 2 months in advance of the workshop. The programmers role in the months prior to the ESDW is to gain some familiarity with these applications. The programmers will put in place a performance analysis workflow for the applications using the tools described in Section 2.1.

During the ESDW, the programmers are there to provide instruction and support in the tools and assist the participants where necessary. They can also leverage the performance analysis workflow that they have prepared to help analyse the performance impact of the work undertaken during the ESDW (using the HPC resources to which E-CAM has access).

3 Porting and Optimisation

This section covers the hardware resources available for WP7 "Hardware considerations and the PRACE relationship" and some specifics of the porting effort required on these architectures.

The HPC resources available to E-CAM to date have come from either one of the HPC partners of the project or from Partnership for Advanced Computing in Europe (PRACE).

3.1 Available Resources

3.1.1 Primary Resources

A number of HPC sites are project partners and have generously made development resources available to the project, particularly in the case where a particular HPC architecture component was not already available to the project. In the current deliverable, the primary resource has been

- JURECA (cluster with GPU accelerators and KNL booster, through partner FZJ-JSC)

for general development work.

3.1.2 PRACE Resources

In the case of PRACE resources, there are two main avenues for access to resources. Each Centre of Excellence (CoE), such as E-CAM, has been allocated 0.5% of the production resource budget of PRACE. The second avenue is the normal PRACE Preparatory Access Call process. E-CAM has previously been successful twice in acquiring additional resources through this second avenue, making an additional 1.1M core hours available to the project. Given that most current architectures are covered by the resources provided by our partners (see Section 3.1.1), we did not pursue this second avenue further in 2018/2019 but relied solely on the CoE access avenue.

We provide the complete list of supercomputers available through PRACE here (the configuration details of the hardware are hyperlinked to the list):

- MareNostrum4 (Primarily a cluster system but also has Arm and OpenPower partitions, Spain): 300.000 core hours
- Hazel Hen (Cray XC40, Germany): 80.000 core hours
- Marconi (Cluster with Xeon Phi accelerators, Italy): 50.000 core hours (BWL partition) + 700.000 core hours (KNL partition)
- SuperMUC (Cluster, Germany): 120.000 core hours
- Piz Daint (hybrid Cray XC40/XC50 system with accelerators, Switzerland): 7000 node hours
- Joliot-Curie (Skylake cluster with Xeon Phi accelerators, France): 75.000 core hours (KNL partition) + 150.000 core hours (SKL partition)
- JUWELS (Skylake cluster with accelerators, Germany): 80.000 core hours

For 2019, we have requested access to Joliot-Curie, JUWELS, Piz-Daint and Marenostrum since this set covers all our architecture and scalability needs (in addition to our existing access to resources).

3.2 Porting Effort

Given the discussion with respect to hardware in D7.5: Hardware Developments III[3], we focus our efforts on cluster-type systems (with latest architectures) and accelerators. Our primary development hardware has been JURECA, a cluster system with both GPUs and a KNL *booster*[4] (see Section 3.1.1).

The initial porting effort mainly involves porting the E-CAM workflow and configuring the application for the software stack of the target system. In particular, the applications are incorporated into EasyBuild with the dependencies provided by it. This ensures that knowledge gained during this process can be easily communicated to the wider

[4]The booster module is intended to accelerate calculations on a cluster module. Complex parts of the code, which are difficult to calculate simultaneously on a large number of processors, are executed on the so-called cluster module with simpler parts of the program that can be processed in parallel with greater efficiency transferred to the booster module.

community. The performance analysis workflow can then provide information to assist tuning the application for the target platform.

3.2.1 Improving build reproducability

The structure of EasyBuild, its release cycle and the number of configuration options available to the end user make the reproducibility of individual software builds that leverage it potentially complex. E-CAM has improved this situation by implementing significant changes in order to better catalogue and store all influential factors that may affect the reproducability of a particular build:

- EasyBuild Merge Request 2574 - Make sure that the configuration file for the software is fully parsed when archived in the reproducability directory
- EasyBuild Merge Request 2661 - Ensure the checksums for sources/patches are stored alongside the names of the sources/patches
- EasyBuild Merge Request 2619 - Protect sensitive components of the configuration file from being templated during the archive process
- EasyBuild Merge Request 2653 - Archive the complete build mechanism as well as the configuration file
- EasyBuild Merge Request 2664 - Ensure that the internal representation of the configuration file used in the archive process is not influenced by the build mechanism itself
- EasyBuild Merge Request 2705 - Ensure the name of the temporary reproducability directory is unique

3.2.2 Improving ability to easily switch toolchains

Application developers typically require to try a number different compilers (or a new version of their existing compiler) but porting their dependency tree to new compilers can be tedious and may actually result in issues unrelated to their own application. In the EasyBuild context (where a toolchain consists of a compiler, MPI implementation and math libraries), we have automated the process to do this consistently and recursively:

- EasyBuild Merge Request 2539 - Increase the scope and capabilities of the `--try-toolchain` option of EasyBuild

3.2.3 Updating the software stack

When moving between toolchains there are frequently updates to be made to the dependency tree of an application. We introduced a mechanism in EasyBuild to automate these updates (again consistently and recursively).

- EasyBuild Merge Request 2599 - Introduce an option to update dependencies when the `--try-toolchain` option of EasyBuild is used

3.2.4 Improving architecture awareness for Autotools packages

Many packages built on Autotools frequently fail out-of-the-box on the latest architectures due to an out-of-date `config.guess` file. In EasyBuild Merge Request 1506, we implement a method in EasyBuild that automatically uses the latest release of this file for all Autotools packages.

3.2.5 Porting LAMMPS+Kokkos to EasyBuild

LAMMPS support in EasyBuild has been difficult to adopt due to the number of additional packages that it can support and the complications of their original build system. A CMake build system has been introduced in LAMMPS that greatly simplify this adoption. With this approach, also building the Kokkos support within LAMMPS is greatly simplified:

- EasyBuild Merge Request 6917 - Add CMake build system support for LAMMPS to EasyBuild and include most optional extensions

3.2.6 Building CP2K as a library

CP2K can be utilised as a library rather than an application. We add this functionality to the EasyBuild CP2K instances:

- EasyBuild Merge Request 1547 - Add an additional build step for CP2K that also builds it as a library
- EasyBuild Merge Request 1554 - Populate the include directory of the CP2K installation and include the Fortran module files

3.3 HTC Optimisation Collaboration with PRACE

While many High Throughput Computing (HTC) platforms/libraries exist (such as Dask.distributed, Celery or COMP Superscalar), we failed to find one that was easy to use, targeted directly to cluster systems and flexible enough to handle MPI workloads. For this reason we embarked on a development project in collaboration with PRACE that builds on top of Dask-Jobqueue (which in turn leverages Dask.distributed). This approach has allowed us to pursue what is effectively interactive supercomputing, where micro-scheduled workloads are created within a Python environment (potentially in real time through an environment such as a Jupyter notebook).

The initial motivation for this library was driven by the ensemble-type calculations that are required in many scientific fields, and in particular in the materials science domain in which the E-CAM Centre of Excellence operates. The scope for parallelisation potenial is best contextualised by the Dask documentation:

> A common approach to parallel execution in user-space is task scheduling. In task scheduling we break our program into many medium-sized tasks or units of computation, often a function call on a non-trivial amount of data. We represent these tasks as nodes in a graph with edges between nodes if one task depends on data produced by another. We call upon a task scheduler to execute this graph in a way that respects these data dependencies and leverages parallelism where possible, multiple independent tasks can be run simultaneously.
>
> Many solutions exist. This is a common approach in parallel execution frameworks. Often task scheduling logic hides within other larger frameworks (Luigi, Storm, Spark, IPython Parallel, and so on) and so is often reinvented.
>
> Dask is a specification that encodes task schedules with minimal incidental complexity using terms common to all Python projects, namely dicts, tuples, and callables. Ideally this minimum solution is easy to adopt and understand by a broad community.

While we were attracted by this approach, Dask did not support *task-level* parallelisation (in particular multi-node tasks). We researched other options (including Celery, PyCOMPSs, IPyParallel and others) and organised a workshop that explored some of these (see the event website for further details).

The initial idea for the implementation of this project was to use `MPI_Comm_spawn`, which is a collective call and spawns a child MPI job with n processes from within an MPI task. The problem with this is that, while part of the MPI standard, the implementation of this is highly specific to the MPI implementation itself, with a considerable amount of configuration information hidden (and often undocumented) in the `MPI_Info` argument to this call. That this information is not part of the standard means that the approach is highly non-portable, potentially requiring source code edits on all platforms where it is used. Not only this, but it was also our experience that, given it's relative obscurity, it is not actually implemented at all in some cases (since it requires coupling to the resource manager, of which there are many possibilities).

Ultimately, once we came to discover the existence of the Dask extension `dask_jobqueue`, we realised that it would be possible for us to build upon it's functionality to include support for parallel task workloads without having to resort to such *exotic* MPI functionality.

The approach described in the rest of this document allows for multi-level parallelisation (at the task level and at the framework level) while leveraging all the pre-existing effort within the Dask framework (such as scheduling, resilience, data management and resource scaling).

The resulting library is flexible, scalable, efficient and adaptive. It is capable of simultaneously utilising CPUs, KNL and GPUs and dynamically adjusting its use of these resources based on the resource requirements of the scheduled task workload. The ultimate scalability and hardware capabilities of the solution is dictated by the scalability characteristics of the tasks themselves (if there are 100 nodes per task with GPUs, then the library can carry out N of these at once depending on resource availability).

4 Modules and Application Codes

For the modules and application codes that have been available and selected in the project, we provide the following information on a per-WP basis:

- Relevance to E-CAM (including relevant modules and ESDWs);
- Benchmarks used;
- Results of our scaling analysis.

Hereafter we will indicate with *cores* the number of physical cores, to keep it distinguished from the *logical cores* (number of physical cores times the factor resulting from hyperthreading). As in most supercomputers, the hyperthreading is switched off for the host processors, while it is active on the Intel Xeon Phi coprocessor (4 in this case).

Where possible timing measurements are taken using internal timers available within the applications themselves. If no such feature is available, or it is more appropriate, then the CPU time reported by the resource management system of the HPC resource is used.

4.1 WP1: Classical Molecular Dynamics

Across scientific fields, HTC is becoming a necessary approach in order to fully utilize next-generation computer hardware. As an example, consider molecular dynamics: Excellent work over the years has developed software that can simulate a single trajectory very efficiently using massive parallelization. Unfortunately, for a fixed number of atoms, the extent of possible parallelization is limited. However, many methods, including semiclassical approaches to quantum dynamics and some approaches to rare events, require running thousands of independent molecular dynamics trajectories. Intelligent HTC, which can treat each trajectory as a task and manage data dependencies between tasks, provides a way to run these simulations on hardware up to the exascale, thus opening the possibility of studying previously intractable systems.

4.1.1 Relevance for E-CAM

The range of use for intelligent HTC in scientific programs is broad. For example, intelligent HTC can be used to select and run many single-point electronic structure calculations in order to develop approximate potential energy surfaces. Even more examples can be found in the wide range of methods that require many trajectories, where each trajectory can be treated as a task, such as:

- rare events methods, like transition interface sampling, weighted ensemble, committor analysis, and variants of the Bennett-Chandler reactive flux method
- semi-classical methods, including the phase integration method and the semi-classical initial value representation
- adaptive sampling methods for Markov state model generation
- approaches such as nested sampling, which use many short trajectories to estimate partition functions.

The challenge is that most developers of scientific software are not familiar with the way such packages can simplify their development process, and the packages that exist may not scale to exascale. The library E-CAM has created is intended to provide an opportunity for scientific developers to add support for HTC to their codes.

As an example use case take committor analysis, which is a powerful, but computationally expensive, tool to study reaction mechanisms in complex systems. For a committor simulation, configuration space is divided into a reactant region, a product region, and the transition region. The reactant and product regions are stable states defined as "core sets," where a trajectory launched near one state is extremely likely to return to that state before visiting the other state.

The committor for a given configuration is defined as the probability that a trajectory launched from that point reaches the product state before the reactant state. This can be determined by running many trajectories from the initial configuration with different initial velocities, and stopping them as soon as they enter one of the two states. This problem is highly parallelizable, however, the duration of each trajectory cannot be known in advance. Therefore load balancing can be difficult, and approaches such as the one presented here are well-suited to this problem.

4.1.2 HTC with `Dask-jobqueue`

The development of the `jobqueue_features` library (whose source code can be found on the `jobqueue_features` GitHub repository) had two major aspects, one that targeted a simplified user experience (minimising configuration and implementation overhead for the end user) and the other that targeted leveraging the hardware heterogeneity of a resource such as JURECA. In the first case, it was possible to integrate a test suite directly into the library. The second case requires the specific configuration of the computing resource and are included as examples that are specific to JURECA (and could be considered as regression tests for the particular resource).

The first module we use as the reference point for this deliverable relates to the configuration of the library for specific HPC resources (see WP1 Merge Request 84). The goal is to allow numerous `cluster` instances (which is a place where tasks are executed) to be defined more broadly and cover all possibilities that the queueing system might offer (such as the availability of GPU, KNL, high memory nodes, etc.) as well as in configurations that are required to execute MPI/OpenMP tasks. The implementation in the library is generic but the specific example provided is for SLURM on the JURECA system.

The configuration file is written in a generic way (in YAML syntax), following the style already present in `dask_jobqueue`. It has been significantly expanded to account for:

- system architecture;
- how MPI programs are launched;
- configuring for MPI tasks and MPI/OpenMP hybrid tasks;
- configuring for accessing resources with specific hardware characteristics (such as GPU, KNL, large-memory nodes, etc.).

Default settings for resources on JURECA are distributed in the default configuration file. This file stores all necessary settings to execute code on all resources types without the need for adding any (additional) resource manager parameters. An example of a configuration is presented in Listing 1.

```
...

jobqueue-features:
  scheduler: slurm

  slurm:
    default-queue-type: batch       # default queue_type to use
    cores-per-node: 24              # Physical cores per node
    hyperthreading-factor: 2        # hyperthreading factor available
    minimum-cores: 24               # Minimum number of cores per dask worker is 1 full node
    gpu-job-extra: []               # Only relevant for particular queue_type
    warning: null

    # MPI/OpenMP related settings ----
    mpi-mode: False                 # MPI mode is off by default
    mpi-launcher: srun              # Default launcher for MPI app (unused unless in MPI mode)
    nodes: null                     # Default node allocation (unused unless in MPI mode)
    ntasks-per-node: 24             # Default tasks per node (unused unless in MPI mode)
    # cpus-per-task: 1                # Default cpus per task (unused unless in MPI mode)
    openmp-env-extra: ['export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}',
                       'export OMP_PROC_BIND=spread',
                       'export OMP_PLACES=threads']

...
```

Listing 1: Example part of the default YAML configuration file

The second module we use as reference is Merge Request 85, which deals with enabling tasks to be run over a set of nodes (specifically MPI/OpenMP tasks). The initial goal was to allow the library to control tasks that are executed via the MPI launcher command. The task tracked by Dask is actually the process created by the launcher. The launcher is a forked process from within the library.

For a particular task, you may require a particular hardware environment and a particular software stack. for Dask, the caveat with respect to the hardware environment is that you need to be able to have a network that supports TCP connections between the scheduler and the workers. On JURECA, for CPU and GPU tasks the scheduler can be run from a login node and the connections made via IPoIB to the workers. For KNL tasks there is a complication in that there is no direct IPoIB connection between the KNL nodes and the login nodes (on JURECA). This requires that KNL tasks are only be started from within the batch environment, where such a connection does exist. It is possible for us

to support this via a cluster-in-cluster approach, but we would have much preferred to handle all workers from the login nodes.

With respect to the software stack, this is a more global problem but again is highlighted by the KNL booster on JURECA. On the booster, you have a different micro-architecture and you need to completely change your software stack to support this. The design of the software stack implementation on JURECA (built on EasyBuild) simplifies this but ensuring your tasks are run in the correct software environment is one of the more difficult things to get right in the library. The porting of library to the context of EasyBuild, the third related module Merge Request 85, streamlines this task on JURECA. To illustrate this complexity, some of the configuration from one of our examples is reproduced in Listing 2.

```
GROMACS_gpu_cluster = CustomSLURMCluster(
    name='GROMACS_gpu_cluster', walltime='00:15:00', nodes=2, mpi_mode=True,
    queue_type='gpus', maximum_scale=5,
    env_extra=[
        'module --force purge',
        'module use /usr/local/software/jureca/OtherStages',
        'module load Stages/Devel-2018b',
        'module load Intel/2019.0.117-GCC-7.3.0',
        'module load ParaStationMPI/5.2.1-1',
        'module load GROMACS/2018.3',
        'module load Dask/Nov2018Bundle-Python-2.7.15',
    ]
)

GROMACS_knl_cluster = CustomSLURMCluster(
    name='GROMACS_knl_cluster', walltime='00:15:00', nodes=4, mpi_mode=True,
    maximum_scale=10, queue_type='knl', python='python',
    env_extra=[
        'module --force purge',
        'unset SOFTWAREROOT',
        'module use /usr/local/software/jurecabooster/OtherStages',
        'module load Stages/Devel-2018b',
        'module load Intel/2019.0.117-GCC-7.3.0',
        'module load IntelMPI/2019.0.117',  # MUST use IntelMPI (don't know why yet)
        'module load GROMACS/2018.3',
        'module load Dask/Nov2018Bundle-Python-2.7.15',
    ]
)

GROMACS_cluster = CustomSLURMCluster(
    name='GROMACS_cluster', walltime='00:15:00', nodes=2, mpi_mode=True, maximum_scale=10,
    env_extra=[
        'module --force purge',
        'module use /usr/local/software/jureca/OtherStages',
        'module load Stages/Devel-2018b',
        'module load Intel/2019.0.117-GCC-7.3.0',
        'module load ParaStationMPI/5.2.1-1',
        'module load GROMACS/2018.3',
        'module load Dask/Nov2018Bundle-Python-2.7.15',
    ]
)
```

Listing 2: Example class instances for various clusters on JURECA

These configurations also lead us to our second point on ease of integration. What is interesting to note is that while the configuration of the clusters as shown is quite non-trivial, it can be located within a single file which will need to be tuned for the particular resource. With respect to the tasks themselves, no tuning is necessarily required. Take for example a task derived from one of these definitions, which can be seen in Listing 3. There is nothing in this task that depends on the particular system where it is run, the system dependencies are entirely handled by the (site-wide) configuration of the library and the (user-specific) configuration of the the clusters required. One can integrate a decision mechanism in the application to allow for the availability (or not) of each of the cluster types.

```
@on_cluster(cluster=GROMACS_cluster, cluster_id='GROMACS_cluster')
@mpi_task(cluster_id='GROMACS_cluster')
def run_mpi(**kwargs):
    script_path = os.path.join(os.getcwd(), 'resources', 'helloworld2.py')
    t = mpi_wrap(pre_launcher_opts='time -f "%e"', executable='python', exec_args=script_path, **kwargs)
    return t
```

Listing 3: Example task based on cluster from Listing 2

The final issue to address is that of the overhead of the library. It is not so easy to get this in a pure form due to the fact that the primary source of overhead comes from starting workers. A significant proportion of this overhead

is in turn due to the time to configure the nodes for a job within the cluster (which is a system overhead, completely independent of the library). An estimate of the system configuration overhead for the various architectures on JURECA is provided in Table 1.

	Haswell	KNL	GPU
Configuration time (per job, in seconds)	10	24	8

Table 1: System configuraton time per job on each of the hardware types available on JURECA

If we use a simple `Hello world!` program, we can directly measure the overhead of the library and use data from the resource manager to account for the system overhead. There are a number of ways one can attempt to visualise this, here we chose two: the total overhead of the framework in seconds, in Fig. 2, and the overhead per task, in Fig. 3. For each case, there are 2 nodes per worker (68 cores on a KNL node, 4 GPUs on a GPU node and 24 physical cores on a CPU node), with 10 workers each (so, at peak utilisation, a total of 480 CPUs, 1360 KNL cores and 80 GPUs). The efficiency of the utilisation of the resources is separate matter that depends on the task, which in the use case connected to this effort, reduces to the performance of GROMACS or LAMMPS (since these are the applications being called out to in the task).

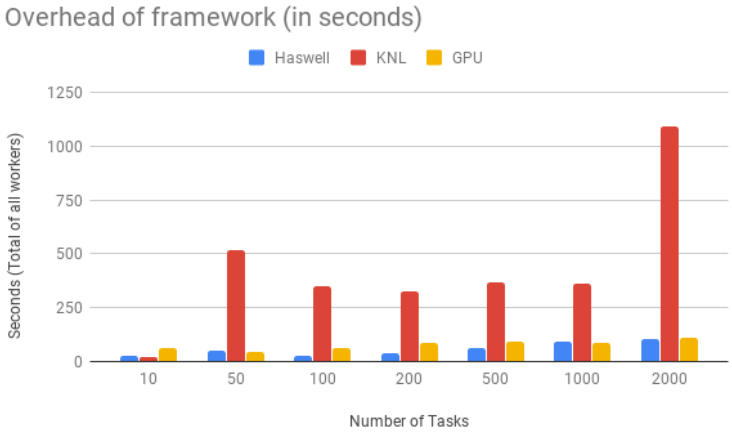

Figure 2: Total overhead of the framework in seconds for various numbers of tasks and on various architectures.

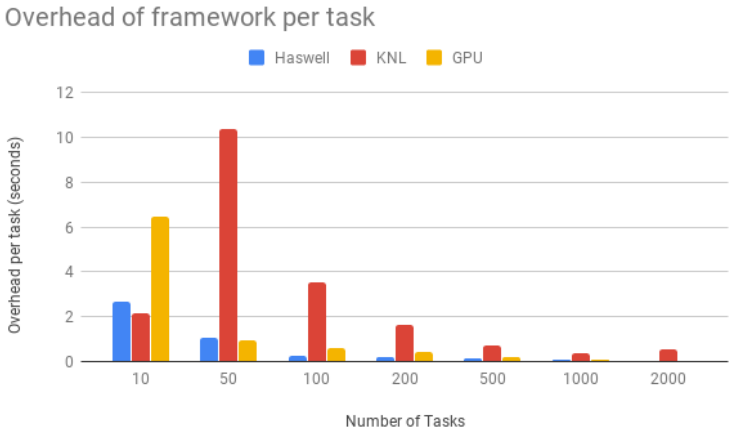

Figure 3: Overhead of the framework per task in seconds for various numbers of tasks and on various architectures.

The interesting point to note for Fig. 2 is that the total overhead is relatively static over time until we get to larger task counts. The sudden increase at 2000 tasks for the KNL data is mostly due to the fact that the simulation took long enough that our workers began to exceed their allowed walltimes, resulting in the resilience mechanism being utilised and additional workers being started.

The overall message is that overhead is very small, particularly if we look at it per task as in Fig. 3. At higher task counts we are seeing almost 90% throughput efficiency for trivial tasks, if the tasks executed for any reasonable length of time this throughout efficiency would be much higher.

While simplistic, it is informative to compare (given Table 1) what the savings are when compared to running the tasks directly through the resource manager. This can be seen in Table 2. A final remark is to note that these savings scale according to the resources used by the task (i.e, the savings scale linearly with respect to the resources used by the task).

Tasks	Haswell	KNL	GPU
10	33.19	-93.61	-48.8
50	357.99	441.55	297.58
100	871.84	1759.37	657.45
200	1860.38	4160.43	1423.28
500	4836.35	11321.85	3827.79
1000	9796.77	23327.39	7824.52
2000	19784.67	46548.92	15803.22

Table 2: Time savings (in seconds) of running tasks through the library rather than through the resource manager

4.1.3 LAMMPS as a task

LAMMPS is one of the molecular dynamics engines that can be used by Open Path Sampling (OPS) (a key application within WP1, see [4] for previous related work) but is also an application that is used heavily by others in the E-CAM community. We have configured the developed library for LAMMPS tasks as a show case for how it might be integrated with another E-CAM software project (see Merge Request 65 for more detailed context).

The Lmod module tool used on JURECA has the ability to store a set of modules to provide a particular environment, using `'module save ...'`, which can be restored with `'module restore ...'`, simplifying the long list of required modules. An example of this use is shown in Listing 4. Note the deliberate selection of `ntasks_per_node=2` and `cpus_per_task=12`, which are optimal choices for LAMMPS tasks in this particular use case.

```
lammps_cluster = CustomSLURMCluster(
    name='lammps_cluster', walltime='00:40:00', nodes=2,  ntasks_per_node=2, cpus_per_task=12,
    mpi_mode=True, maximum_scale=10,
    env_extra=[
        'module --force purge',
        'module use /usr/local/software/jureca/OtherStages',
        'module restore picore',
    ]
)
```

Listing 4: Cluster configuration for LAMMPS using Lmod module sets

The performance charactertistics of LAMMPS are set by command line arguments. This makes executing tasks efficiently on any of the architectures very straightforward: you ensure you have the correct software environment for the task and that you execute with the correct performance flags. This is illustrated in Listing 5.

```
# The generic lammps execution
def run_lammps(performance_args='', **kwargs):
    standard_exec_args = ' -in  templatev5_nokspace-scaling.in -var Replicate_nx 6 ' \
                         '-var Replicate_ny 8 -var Replicate_nz 4  -var SAMPLE_frequency 2000'
    exec_args = performance_args + standard_exec_args
    return mpi_wrap(executable='lmp', exec_args=exec_args, **kwargs)


@on_cluster(cluster=lammps_gpu_cluster, cluster_id='lammps_gpu_cluster', scale=10)
@mpi_task(cluster_id='lammps_gpu_cluster')
def run_mpi_gpu(**kwargs):
    t = run_lammps(performance_args='-k on g 2 -sf kk -pk kokkos gpu/direct off',
                   **kwargs)
    return t


@on_cluster(cluster=lammps_knl_cluster, cluster_id='lammps_knl_cluster', scale=10)
@mpi_task(cluster_id='lammps_knl_cluster')
def run_mpi_knl(**kwargs):
    t = run_lammps(performance_args="-sf omp -pk omp {}".format(os.getenv("OMP_NUM_THREADS")), **kwargs)
    return t


@on_cluster(cluster=lammps_cluster, cluster_id='lammps_cluster', scale=10)
@mpi_task(cluster_id='lammps_cluster')
```

```
def run_mpi(**kwargs):
    t = run_lammps(performance_args="-sf omp -pk omp {}".format(os.getenv("OMP_NUM_THREADS")), **kwargs)
    return t
```

Listing 5: Configuring LAMMPS tasks to run on 3 different architectures

4.2 WP2: Electronic Structure

In the Electronic Structure work package (WP2) the field is particularly well-developed with a number of heavily utilised community codes (some of which, such as Quantum ESPRESSO and SIESTA, are already the subject matter of another CoE). Within E-CAM, the main focus is more on extracting useful utilities from these applications so that they can be leveraged by a wider spectrum of applications as libraries, which is an effort coordinated and sustained by the Electronic Structure Library (ESL).

The ESL is consistently working towards creating a bundle of relevant libraries and constructing a demonstrator application that incorporates these libraries and highlights how they can be combined to create an efficient, scalable, fully-featured electronic structure application.

4.2.1 Relevance for E-CAM

E-CAM has consistently supported the efforts of the ESL and has funded a number of ESDW events to help give the development effort momentum. The ESL model is one that, if successful, has the potential to be adopted within other areas of the project.

4.2.2 The ESL bundle and the ESL demonstrator

There are two merge requests that combine to form the software for this discussion: one module for the ESL bundle and the other module for the ESL demonstrator.

The relevant ESDW for the current discussion is the ESDW for Scaling Electronic Structure Applications. Unfortunately at the time of writing the second part of this ESDW had not taken place and so we can only present an initial code analysis for a small test case.[5]

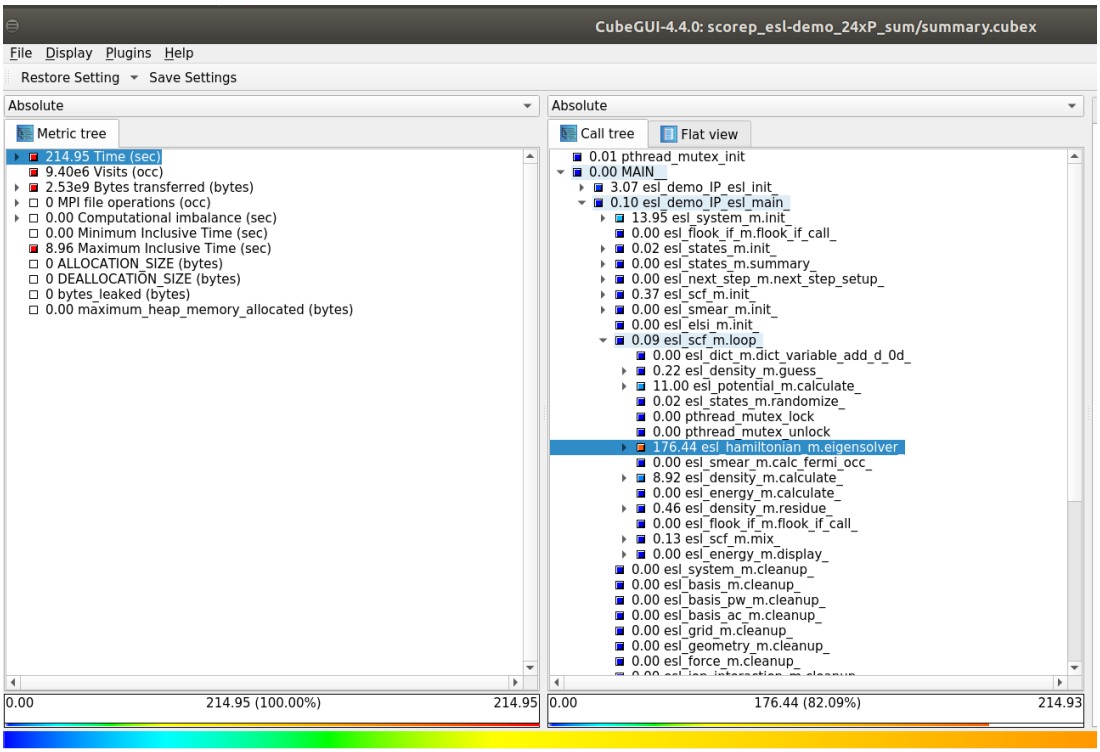

Figure 4: Initial performance analysis of sample usage of the ESL demonstrator.

[5] We have compensated for this shortcoming by including additional relevant modules in our treatment of the other WPs

In Figure 4, we show that the execution time (for this case) is heavily concentrated (82%) in the eigensolver. This area is exactly the target of the ELSI (a US-funded project) which is one of the core components of the ESL.

4.3 WP3: Quantum Dynamics

For the Quantum Dynamics (WP3) package two modules are presented: the parallelization of the coupling scheme between PaPIM code and CP2K to enable virtually any calculation of time-dependent correlation functions for any system, and the introduction of a Surface Hopping Propagator code for computing quantum rate processes in condensed phase systems by combining quantum and classical descriptions of the dynamics including non-adiabatic coupling.

4.3.1 Relevance for E-CAM

Both of the applications addressed were subjects of the third WP3 ESDW events. The related E-CAM modules for PaPIM code is PIM-CP2K Interface. PaPIM is part of E-CAM pilot project developed by the E-CAM PDRA Momir Malis. The other module, developed by Donal McKernan during the ESDW event, is part of a cross-WP collaboration between WP3 and WP1.

4.3.2 PaPIM

The PaPIM code is a package to study the properties of quantum materials (in particular time correlation functions from which experimental observations can be rationalised) via the so-called mixed quantum classical methods. In these schemes, quantum evolution is approximated by appropriately combining a set of classical trajectories for the system.

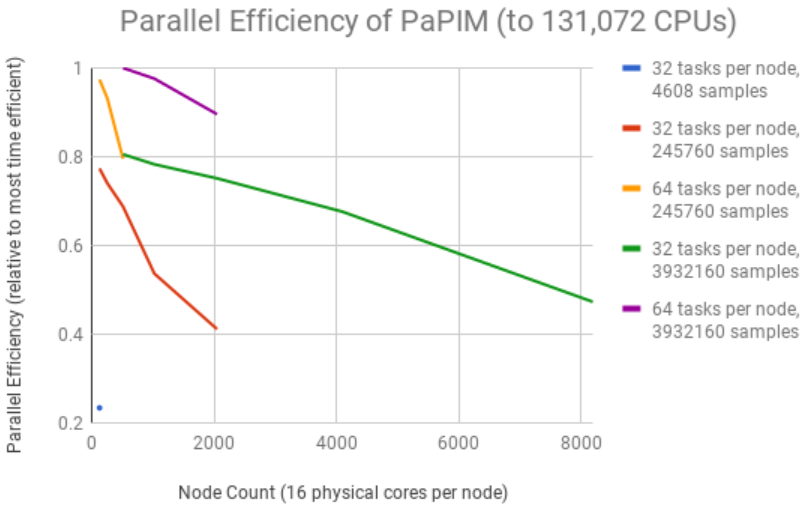

Figure 5: PaPIM performance on JUQUEEN up to 131,072 CPUs (and 262,144 MPI tasks(without CP2K integration. The parallel efficiency on the X-axis is the time per sample relative to the most time-efficient result, the Y-axis is the node count (with 16 physical cores per node).

In the previous deliverable in this series [4], we showed that PaPIM is highly scalable (reproduced in Figure 5) and showed that this scalability is only limited by the low workload required of the cores.

In order to explore more interesting and complex systems, PaPIM has been coupled to CP2K, which introduces a nested parallelism model while simultaneously increasing the work per core. The inclusion of CP2K for computation of system's electronic structure properties enables calculation of time-dependent correlation functions for a vast range of systems, while CP2K can perform atomistic simulations of solid state, liquid, molecular, periodic, material, crystal, and biological systems. The PaPIM code has also been upgraded with periodic boundary conditions to enable simulations of solid and liquid state systems. For any system whose properties can be determined with the CP2K code, a corresponding time-dependent correlation function can be computed now with the PaPIM code.

The nested parallelism of PaPIM linked with CP2K is achieved with a MPI split communicator approach, with a separate communicator given for the PaPIM code and for CP2K. The latter is split into groups, each of a number of processor cores given by the `group_size` value. Therefore, the number of trajectories which can be sampled simultaneously is given by the quotient of the total number of used processor cores with the value of the `group_size`. For the same reason the total number of cores must be divisible by the `group_size` value. Figure 6 explains in a simplified graphical manner the parallelization used in the PaPIM code linked to CP2K.

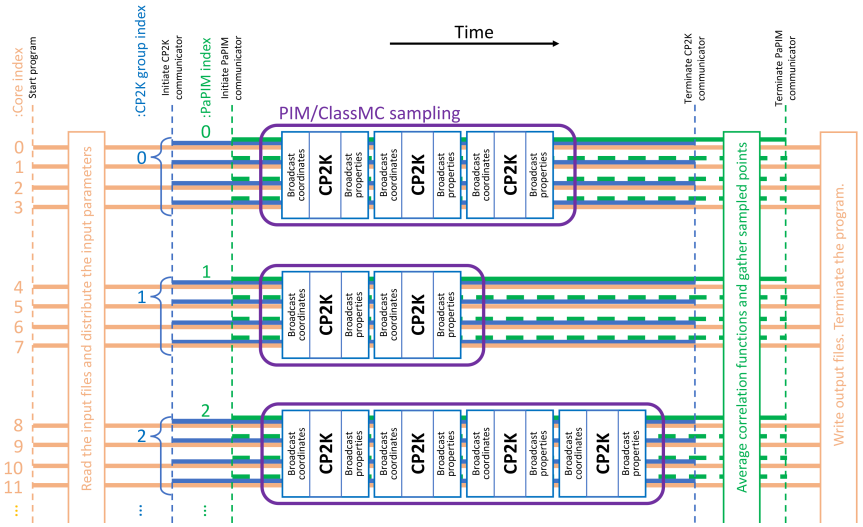

Figure 6: Graphical representation of the MPI split communicator scheme used in parallelization of `PaPIM-CP2K_interface` module.

Unfortunately, we have (as yet) been unable to explore the parallel efficency of this implementation beyond a trivial test case due to the availability of key scientific collaborators.

4.3.3 Surface Hopping Propagator

Quantum rate processes in condensed phase systems are often computed by combining quantum and classical descriptions of the dynamics including non-adiabatic coupling, using propagators which amount to quantum path integrals in a partial Wigner phase space representation, such as the mixed quantum-classical Dyson equation and variants thereof, or the Trotter decomposition of the quantum-classical propagator.

The module software has been entirely refactored in modern C++ (GNU 2011 or higher) so as to: (a) run with high-efficiency on massively parallel platforms under OpenMP or MPI; and (b) be at the core of additional software modules aimed at addressing important issues such as improving the speed of convergence of estimates using correlated sampling, and much more realistic treatment of the classical bath, and connecting to other problems such as constant pH simulation through an effective Hamiltonian.

Testing was performed on the Kay supercomputer from ICHEC. Kay is separated into nodes, each of which has 2 x (20 core) sockets. To test the parallel efficiency of both the OpenMP and MPI versions of the code they were benchmarked on 20 - 200 cores (1 - 5 nodes). Both versions were run for 10,000,000 samples (Nsample = 10,000,000) and for a bath size of 200 (N_bath = 200).

For the MPI benchmark in Figure 7, we observe near linear scaling to the full CPU count.

As can be seen in Figure 8 OpenMP scales perfectly for the physical cores on a single node (i.e. less than 40 cores), with performance continuing to improve slightly until we reach the maximum hardware thread count (80 hardware threads running on 40 physical cores) (Figure 8).

Similar to what we have observed for PaPIM, we found that it is the low workload of the cores that is the limiting factor with respect to the scalability of the Surface Hopping Propagator code. To investigate this, we studied the execution time with Scalasca, and look at where this time is spent. This is shown in Figure 9, where we can see that even with just 96 MPI tasks nearly 20% of the execution time is spent in the initialisation and finalisation of MPI. There is simply not enough computation required to warrant investigating the scalability further, even if the method itself is potentially highly scalable. Currently, there is no hybrid version of the application, but from Figures 7 and 8 one might expect a hybrid version should work well, and would also mitigate somewhat the bottleneck that comes from the use of MPI (since one MPI task per node would require much less MPI management overhead).

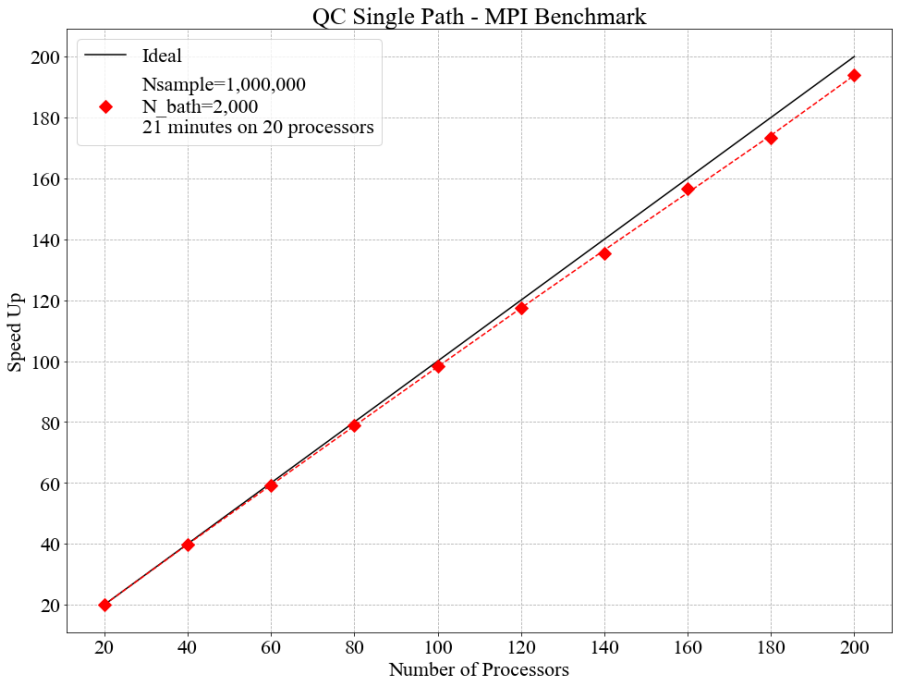

Figure 7: QC single path MPI benchmark

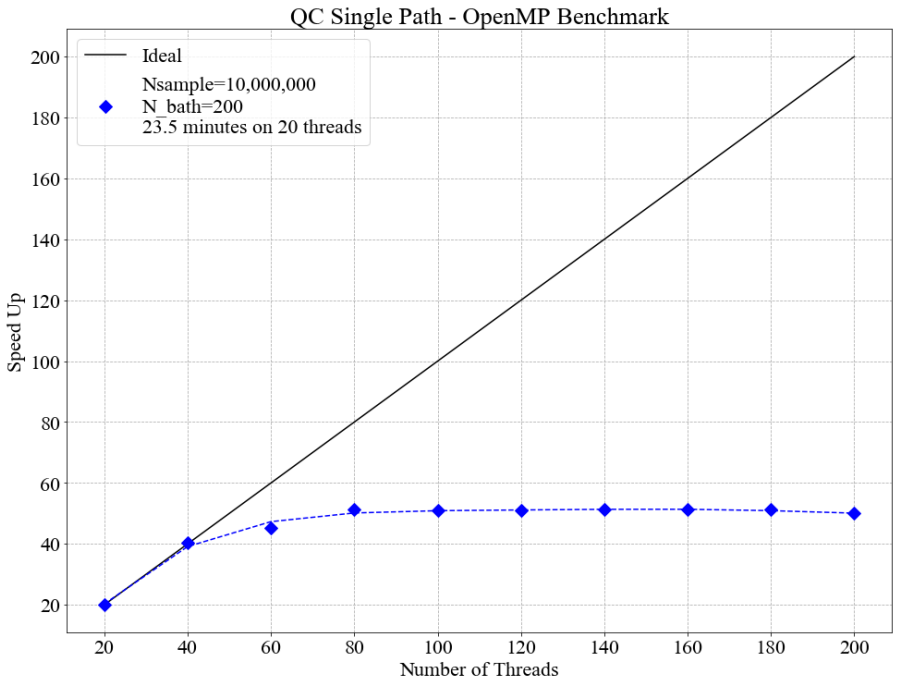

Figure 8: QC single path OpenMP benchmark

4.4 WP4: Meso- and Multi-scale Modelling

As part of the WP4 package the following two modules are presented: an improvement of the multi-GPU version of DL_MESO_DPD using CUDA_Aware_MPI GPUDirect technologies which allows good scaling of up to 2048 GPUs on Piz Daint Supercomputer, and the benchmark of GROMACS-GcAdresS on Jureca Supercomputer.

4.4.1 Relevance for E-CAM

The DL_MESO_DPD porting to a multi-GPU environment can be seen as an extension of the Pilot Project developed by the E-CAM PDRA Dr. Silvia Chiacchiera in WP4 on Polarizable Soft Water Model. The main purpose is to accelerate the DL_MESO_DPD code when electric particles (like polarised water) are added to the system. Charged particles

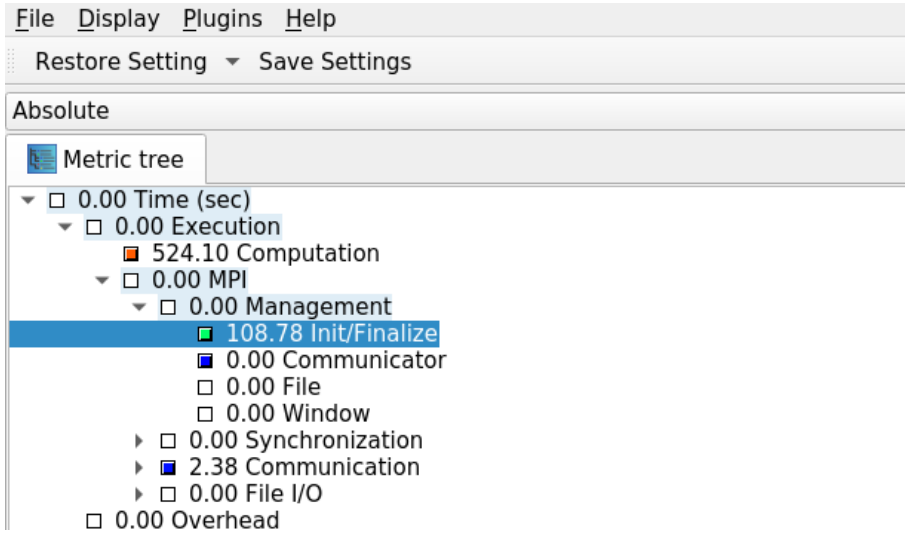

Figure 9: Distribution of execution time of Surface Hopping Propagator for 96 MPI tasks.

require the evaluation of long range interaction forces, notoriously expensive due to the complexity of the algorithms used (like the Smoothed Particle Mesh Ewald method). A GPU-enabled version of the code would allow the user to run large system of charged particles even on a simple workstation. The project involves a collaboration between computational scientists (STFC Daresbury), academia (University of Manchester), and industry (Unilever).

GROMACS-GcAdResS is used in the Pilot Project on Development of the GC-AdResS scheme developed by the E-CAM PDRA Dr. Christian Krekeler. The main aspect is to couple two simulation boxes together and combine the advantages of classical atomistic simulations with those from coarse gained simulations. The following module gives and impact on performance of the GC-AdResS scheme on Jureca Supercomputer using different system sizes.

4.4.2 Scaling of DL_MESO_DPD on GPU on Piz Daint

The multi-GPU version of DL_MESO_DPD is based on a classical domain decomposition with exchange of data between GPUs to take in account of the movement of particles from and to different domains. The overlap of computation and communication, based on the CUDA_aware_MPI technology, allows good scaling and performance provided that the memory occupied by each GPU for computation is no lower than 20% ($0.5M$ particles per GPU). Remote Direct Memory Access (RDMA) is enabled to enhance data transfer performance. A test case a two phase mixture separation with 1.8 billion particles has been used and run for 100 time steps without IO operations.

A weak scaling efficiency (η) related plot up to 512 GPUs (1.2 billion particles) on Piz Daint is presented in Fig. 10. This plot is obtained by taking the ratio between the wall time for the GPU count and a reference walltime of two GPUs (the single GPU version uses a non-scalable, faster, alternative implementation which would skew the results). As can be seen, the result ($\eta * GPUs$) oscillates near perfect scalability.

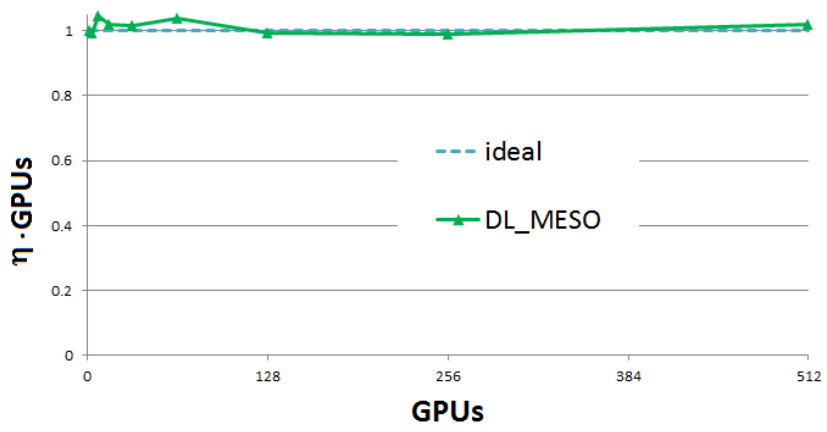

Figure 10: DL_MESO_GPU weak scaling up to 512 GPUs.

Strong scaling results (Fig. 11) are obtained using 1.8 billion particles for 256 to 2048 GPUs. Results show very good scaling, with efficiency always above 89% for 2048 GPUs (note that 2048 P100 GPUs on Piz Daint is equivalent to almost 10 Petaflops of raw double precision compute performance).

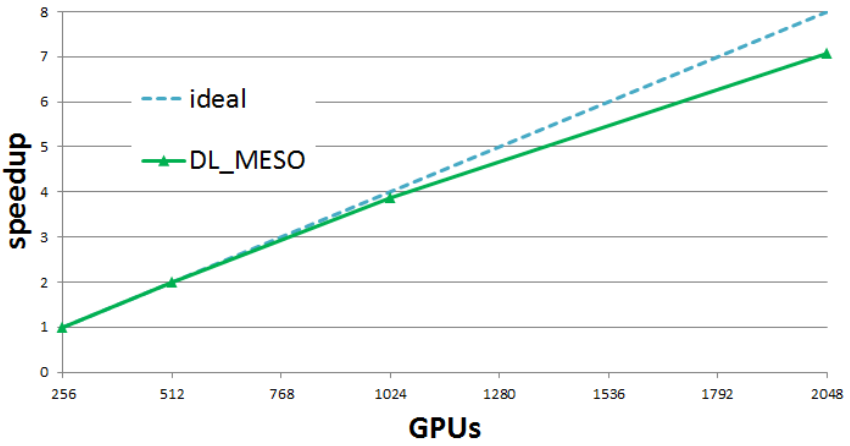

Figure 11: DL_MESO_GPU strong scaling up to 2048 GPUs

Further details on the implementation of these improvements and the source code can be found in the E-CAM GitLab service under the linked Merge Request 78.

4.4.3 Bond forces to DL_MESO_DPD GPU version

This module adds the bond forces to the multi-GPU version of DL_MESO_DPD. These take into account the interactions between different chemical species which allow to create complex molecules more representative of real systems. An example of application is the ternary solution where a main component contains bonds interacting with the other two phases (see Figure 12). This module is a prerequisite requirement for an implementation of the load imbalance library being developed as part of the WP4/WP7 collaboration (see the relevant ESDW hosted in Julich in September 2018 and to be continued in June 2019).

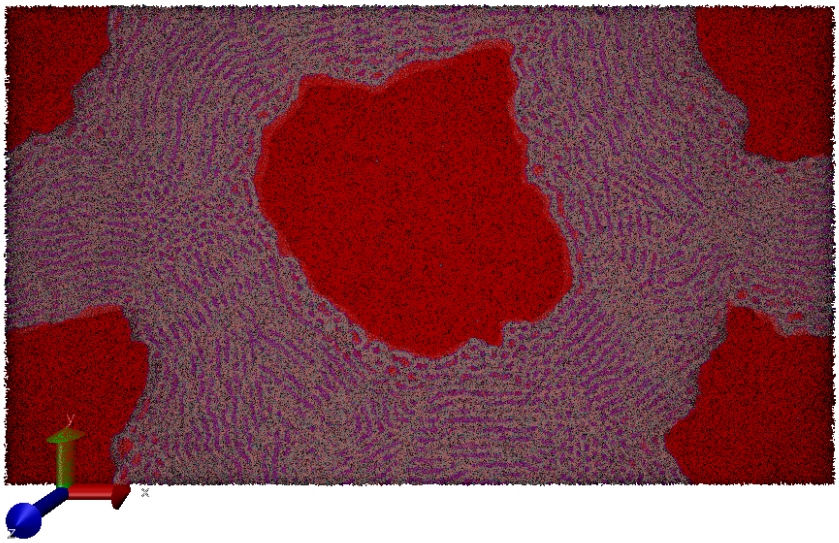

Figure 12: Ternary system with bond force across phases.

The algorithm used is the same of the DL_MESO serial version, but of course adapted for SIMT (Single Instruction Multiple Threads) architecture. The module includes also the angle and dihedral forces, all divided according a classical orthogonal domain decomposition. Considering that in a real case the number of bounds is usually much lower than the total number of particles, different CUDA streams for the three kernels (`k_findBondForce`, `k_findAngleForce`

and `k_findDihedralForce`) are used. This allow to launch them in parallel to improve the performance of the overall simulation.

Further details on the implementation of these improvements and the source code can be found in the E-CAM GitLab service under the linked Merge Request 103.

4.4.4 Benchmarking GC-AdResS on Jureca

This module presents the scaling of a GROMACS implementation of GC-AdResS on JURECA supercomputer. The purpose is to investigate the scalability of the GC-AdResS scheme when implemented in a specific molecular dynamics code. In this case GROMACS v5.1, where the original AdResS scheme has been modified, see the Abrupt AdResS Merge Request for full details.

A system of 50k atoms/particles has been chosen as benchmark and the simulation is run for 100 time steps without IO. The benchmark has been run with 6, 12 and 24 MPI processes binding 1 MPI task per physical core. Strong scaling results are presented in table 3. The poor scaling is due to the intrinsic heterogeneity of the method, i.e. the mixing of coarse and fine grain particles (see Figure 13). The automatic load-balancing in GROMACS does not appear to handle this well, where we see load imbalance values as high as 54.4% for just 24 cores.

cores	Simulation Time [ns/day]	Load imbalance [%]
6	130.8	6.0
12	180.2	18.5
24	211.4	54.4

Table 3: Strong scaling for GcAdresS on JURECA

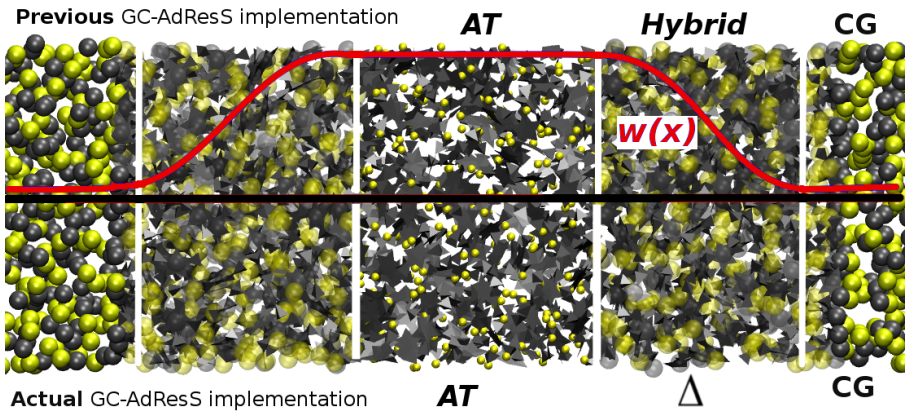

Figure 13: Gc-AdresS system made of coarse and fine particles.

The AdresS scheme has been removed entirely from more recent versions of GROMACS and further investigation of this poor scalability cannot, therefore, be addressed even if it's origin could be identified. For this reason, it is considered that the implementation of the method in ESPREsSo++ (which uses AdresS; is the implementation software for the Pilot Project on Rheological Properties of New Composite Materials; and has been investigated in the previous iteration of this deliverable [4]) coupled with the load-balancing library being developed as part of the ESDW for Atomistic, Meso- and Multiscale Methods on HPC Systems is the best route forward.

5 Outlook

Going forward the work of WP7 will continue to focus more on the cross-cutting development efforts and applications with more potential in terms of extreme scalability. This redirection of effort is part of the revised strategy of E-CAM that focuses more on extreme-scale challenges, as recommended by the project reviewers.

In particular, we expect further developments relating to the `jobqueue_features` HTC library and to the load-balancing library being developed at JSC (which will form a core part of the next deliverable in this series).

References

Acronyms Used

CECAM Centre Européen de Calcul Atomique et Moléculaire

HPC High Performance Computing

PRACE Partnership for Advanced Computing in Europe

ESDW Extended Software Development Workshop

WP Work Package

CoE Centre of Excellence

MPI Message Passing Interface

GPU Graphical Processing Unit

PDRA Post-doctoral Research Associate

OPS Open Path Sampling

HTC High Throughput Computing

ESL Electronic Structure Library

URLs referenced

Page ii

https://www.e-cam2020.eu ... https://www.e-cam2020.eu
https://www.e-cam2020.eu/deliverables ... https://www.e-cam2020.eu/deliverables
E-CAM Zenodo Community page ... https://zenodo.org/communities/e-cam/search?page=1&size=20&q=deliverable&type=publication&subtype=deliverable
Internal Project Management Link ... https://redmine.e-cam2020.eu/issues/49
a.ocais@fz-juelich.de ... mailto:a.ocais@fz-juelich.de
http://creativecommons.org/licenses/by/4.0 ... http://creativecommons.org/licenses/by/4.0

Page iii

Piz Daint ... https://www.cscs.ch/computers/piz-daint/

Page 1

jobqueue_features ... https://github.com/E-CAM/jobqueue_features
Intelligent High Throughput Computing for Scientific Applications ... https://www.e-cam2020.eu/legacy_event/extended-software-development-workshop-intelligent-high-throughput-computing-for-scientific-
ESL demonstrator ... https://gitlab.e-cam2020.eu/esl/esl-demo
ESL bundle ... https://gitlab.e-cam2020.eu/esl/esl-bundle
ESDW on scaling electronic structure applications ... https://www.e-cam2020.eu/legacy_event/extended-software-
PaPIM ... http://e-cam.readthedocs.io/en/latest/Quantum-Dynamics-Modules/modules/PaPIM/readme.html
Surface Hopping ... https://gitlab.e-cam2020.eu/Quantum-Dynamics/Surface-Hopping
ESDW in Quantum Dynamics ... https://www.e-cam2020.eu/legacy_event/extended-software-development-work
DL_MESO_DPD ... http://www.scd.stfc.ac.uk/support/40694.aspx
GC-AdResS ... https://www.e-cam2020.eu/pilot-project-gc-adress/
ESDW in Meso and multiscale modeling ... https://www.e-cam2020.eu/legacy_event/extended-software-developm
jobqueue_features ... https://github.com/E-CAM/jobqueue_features
GC-AdResS ... https://www.e-cam2020.eu/pilot-project-gc-adress/
GC-AdResS ... https://www.e-cam2020.eu/pilot-project-gc-adress/
Surface Hopping ... https://gitlab.e-cam2020.eu/Quantum-Dynamics/Surface-Hopping
DL_MESO_DPD ... http://www.scd.stfc.ac.uk/support/40694.aspx
Tesla P100 ... https://www.nvidia.com/en-us/data-center/tesla-p100/

Page 2

EasyBuild ... http://easybuild.readthedocs.org/en/latest/

Page 3

EasyBuild ... http://easybuild.readthedocs.org/en/latest/
EasyBuild ... http://easybuild.readthedocs.org/en/latest/
JUBE ... https://apps.fz-juelich.de/jsc/jube/jube2/docu/index.html

Page 4

EoCoE ... http://www.eocoe.eu/
3rd EoCoE/POP Workshop on Performance Analysis ... https://pop-coe.eu/blog/3rd-eocoe-pop-workshop-on-bench
Scalasca ... http://www.scalasca.org/
POP ... https://pop-coe.eu/
POP Centre of Excellence ... https://pop-coe.eu/
ESDW guidelines ... https://www.e-cam2020.eu/deliverables/

Page 5

JURECA ... http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JURECA/Configuration/Configuration_node.html;jsessionid=DFEDF689186F7E463728DBE6BF1BE02C
PRACE Preparatory Access Call ... http://www.prace-ri.eu/prace-preparatory-access/
MareNostrum4 ... https://www.bsc.es/marenostrum/marenostrum
Hazel Hen ... http://www.hlrs.de/en/systems/cray-xc40-hazel-hen/
Marconi ... http://www.cineca.it/en/content/marconi
SuperMUC ... https://www.lrz.de/services/compute/supermuc/systemdescription/
Piz Daint ... https://www.cscs.ch/computers/piz-daint/
Joliot-Curie ... http://www-hpc.cea.fr/en/complexe/tgcc-Irene.htm
JUWELS ... http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUWELS/Configuration/Configuration_node.html

Page 6

EasyBuild Merge Request 2574 ... https://github.com/easybuilders/easybuild-framework/pull/2574
EasyBuild Merge Request 2661 ... https://github.com/easybuilders/easybuild-framework/pull/2661
EasyBuild Merge Request 2619 ... https://github.com/easybuilders/easybuild-framework/pull/2619
EasyBuild Merge Request 2653 ... https://github.com/easybuilders/easybuild-framework/pull/2653
EasyBuild Merge Request 2664 ... https://github.com/easybuilders/easybuild-framework/pull/2664
EasyBuild Merge Request 2705 ... https://github.com/easybuilders/easybuild-framework/pull/2705
EasyBuild Merge Request 2539 ... https://github.com/easybuilders/easybuild-framework/pull/2539
EasyBuild Merge Request 2599 ... https://github.com/easybuilders/easybuild-framework/pull/2599
EasyBuild Merge Request 1506 ... https://github.com/easybuilders/easybuild-easyblocks/pull/1506
EasyBuild Merge Request 6917 ... https://github.com/easybuilders/easybuild-easyconfigs/pull/6917

Page 7

EasyBuild Merge Request 1547 ... https://github.com/easybuilders/easybuild-easyblocks/pull/1547
EasyBuild Merge Request 1554 ... https://github.com/easybuilders/easybuild-easyblocks/pull/1554
Dask.distributed ... http://distributed.dask.org/en/latest/
Celery ... http://www.celeryproject.org/
COMP Superscalar ... https://www.bsc.es/research-and-development/software-and-apps/software-list/comp-superscalar
Dask-Jobqueue ... https://jobqueue.dask.org/en/latest/
Dask documentation ... https://docs.dask.org/en/latest/
the event website ... https://www.e-cam2020.eu/legacy_event/extended-software-development-workshop-intel

Page 9

jobqueue_features ... https://github.com/E-CAM/jobqueue_features
jobqueue_features GitHub repository ... https://github.com/E-CAM/jobqueue_features
JURECA ... http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JURECA/JURECA_node.html
WP1 Merge Request 84 ... https://gitlab.e-cam2020.eu/e-cam/E-CAM-Library/merge_requests/84
JURECA ... http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JURECA/JURECA_node.html
YAML ... https://yaml.org/
Merge Request 85 ... https://gitlab.e-cam2020.eu/e-cam/E-CAM-Library/merge_requests/85

Page 10

Merge Request 85 ... https://gitlab.e-cam2020.eu/e-cam/E-CAM-Library/merge_requests/85

Page 12

OPS ... http://openpathsampling.org/latest/
Merge Request 65 ... https://gitlab.e-cam2020.eu/e-cam/E-CAM-Library/merge_requests/65

Page 13

ESL ... https://esl.cecam.org/Main_Page
module for the ESL bundle ... https://gitlab.e-cam2020.eu/e-cam/E-CAM-Library/merge_requests/41
module for the ESL demonstrator ... https://gitlab.e-cam2020.eu/e-cam/E-CAM-Library/merge_requests/47
ESDW for Scaling Electronic Structure Applications ... https://www.e-cam2020.eu/legacy_event/extended-software-

Page 14

ELSI ... https://wordpress.elsi-interchange.org/
coupling scheme ... https://e-cam.readthedocs.io/en/latest/Quantum-Dynamics-Modules/modules/PaPIM-CP2K_Interface/readme.html
PaPIM ... https://e-cam.readthedocs.io/en/latest/Quantum-Dynamics-Modules/
CP2K ... https://www.cp2k.org/
Surface Hopping Propagator ... https://gitlab.e-cam2020.eu/Quantum-Dynamics/Surface-Hopping
third WP3 ESDW events ... https://www.e-cam2020.eu/legacy_event/extended-software-development-workshop-
PIM-CP2K Interface ... https://e-cam.readthedocs.io/en/latest/Quantum-Dynamics-Modules/modules/PaPIM-CP2K_Interface/readme.html
Momir Malis ... https://www.e-cam2020.eu/pilot-project-ibm/
PaPIM code ... https://gitlab.e-cam2020.eu/Quantum-Dynamics/PIM/tree/deliverables

Page 16

GPUDirect technologies ... https://developer.nvidia.com/gpudirect
Piz Daint Supercomputer ... https://www.cscs.ch/computers/piz-daint/
Jureca Supercomputer ... http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JURECA/JURECA_node.html
Polarizable Soft Water Model ... https://www.e-cam2020.eu/pilot-project-unilever/

Page 17

Development of the GC-AdResS scheme ... https://www.e-cam2020.eu/pilot-project-gc-adress/
Jureca Supercomputer ... http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JURECA/JURECA_node.html
Piz Daint ... https://www.cscs.ch/computers/piz-daint/
Piz Daint ... https://www.cscs.ch/computers/piz-daint/

Page 18

Merge Request 78 ... https://gitlab.e-cam2020.eu/e-cam/E-CAM-Library/merge_requests/78
ESDW hosted in Julich in September 2018 ... https://www.e-cam2020.eu/legacy_event/extended-software-develop

Page 19

Merge Request 103 ... https://gitlab.e-cam2020.eu/e-cam/E-CAM-Library/merge_requests/103
GC-AdResS ... https://www.e-cam2020.eu/pilot-project-gc-adress/
JURECA ... http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JURECA/JURECA_node.html
Abrupt AdResS Merge Request ... https://gitlab.e-cam2020.eu/e-cam/Meso-Multi-Scale-Modelling-Modules/merge_requests/38
Rheological Properties of New Composite Materials ... https://www.e-cam2020.eu/pilot-project-michelin/
ESDW for Atomistic, Meso- and Multiscale Methods on HPC Systems ... https://www.cecam.org/workshop-1591.html

Page 20

`jobqueue_features` ... https://github.com/E-CAM/jobqueue_features

Citations

[1] A. O'Cais, L. Liang, and J. Castagna, "E-CAM Software Porting and Benchmarking Data I," Sep. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.1191428

[2] A. Mendonça, A. O. Cais, and D. Mackernan, "D5.4: Esdw guidelines and programme iv," Mar. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.2586966

[3] A. O'Cais, L. Liang, and J. Castagna, "Hardware developments iii," Jul. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1304088

[4] ——, "E-cam software porting and benchmarking data ii," Mar. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1210094