E-CAM Software Porting and Benchmarking Data II

E-CAM Deliverable 7.4
Deliverable Type: Report
Delivered in March, 2018

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	D7.4
Deliverable Nature	Report
Dissemination Level	Public
Contractual Date of Delivery	Project Month 24(1st October, 2018)
Actual Date of Delivery	29th March, 2018
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 II
	ID:	D7.4
	Version:	As of March, 2018
	Status:	Accepted by WP Leader
	Available at:	https://www.e-cam2020.eu/deliverables
	Document history:	Internal Project Management Link
Review	Review Status:	Reviewed
Authorship	Written by:	Alan O'Cais(Juelich Supercomputing Centre)
	Contributors:	Liang Liang (CNRS), 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

29th March, 2018

1 a.ocais@fz-juelich.de

Contents

List of Figures

List of Tables

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: benchmark of OPS using LAMMPS and GROMACS. The associated modules were developed in the context of ESDW7.
- for Electronic Structure: LibOMM and Wannier90. The associated modules were developed in the context of ESDW1 and ESDW3.
- for Quantum Dynamics: PaPIM code and Quantics. The associated modules were developed in the context of ESDW6.
- for Meso- and Multi-scale Modelling: performance improvements of DL_MESO_DPD on GPU and benchmark of ESPResSo++ with Easybuild and JUBE. The associated modules were developed in the context of ESDW5.

The scaling behaviour of these applications on a variety of architectures available to E-CAM was investigated. These architectures included Cray XC, Bluegene/Q and cluster-type systems (with Xeon Phi or GPU accelerators on various systems). These architectures cover the full spectrum of current production hardware available within PRACE.

There have been a number of significant success stories:

- the overhead of OPS when using various engines is very low and does not impact the scalabilty of the underlying engines,
- PaPIM has been shown to be able to scale up to ~250K MPI tasks,
- up to 12× improvements in the GPU performance of DL_MESO_DPD have been obtained,
- LibOMM, Wannier90, and ESPResSo++ have all been shown to scale well up to 1000s of cores.

A number of performance related issues arising from these codes have also been identified. In many cases we have observed performance bottlenecks in the overall application performance that are not the results of the E-CAM development efforts:

- To be properly evaluated, LibOMM needs to be run in a calling application. The recommended application has a very inefficient element in its initialisation which limits the analysis that can be carried out.
- The Wannier90 applications have only one tool with an MPI implementation in its workflow. The initial element in the workflow is the current bottleneck and is only recently being developed to run in parallel.
- The parallelisation approach adopted in Quantics is heavily imbalanced and a new parallelism strategy may be required for balanced and scalable parallelism.
- ESPResSo++ has initialisation and finalisation elements that appear to be serial and are the current bottleneck with respect to scalability (the core calculation itself scales very well).

Collaboration has also played a key role in the activities of E-CAM within the period with collaborative workshops being held with the POP and EoCoE centres of excellence and another with PRACE. In addition, there have been two collaborative projects with POP and there is an ongoing PRACE joint project undertaken by the project related to High Throughput Computing.

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). Much of the porting effort is in fact porting the tool EasyBuild itself to the target platform. In particular this year, we have ported EasyBuild to a POWER9 OpenPOWER system.

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 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 2 software applications from each 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 consists of the Software Manager (at partner Jülich Supercomputing Centre (JSC)) and the two programmers hired at partners Science and Technology Facilities Council (STFC) and Centre national de la recherche scientifique (CNRS). 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.

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.

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

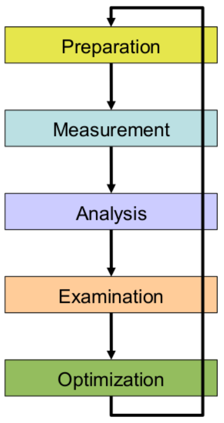


Figure 1: A Performance Optimisation Loop

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 appear in 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 D5.3[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.

- JURECA (cluster with GPU accelerators and KNL booster, through partner FZJ-JSC)
- Poincare (cluster, through partner CNRS)
- Ouessant (OpenPOWER prototype, through partner CNRS)

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 almost all 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 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, Spain)
- Hazel Hen (Cray XC40, Germany)
- JUQUEEN (BG/Q, Germany)
- Marconi (Cluster with Xeon Phi accelerators, Italy)
- SuperMUC (Cluster, Germany)
- Curie (Cluster with GPU accelerators, France)
- Piz Daint (hybrid Cray XC40/XC50 system with accelerators, Switzerland)

3.2 Porting Effort

Given that there will be no successor to the BG/Q and taking into account the discussion with respect to hardware in D7.3: Hardware Developments II[3], we focus our efforts on cluster-type systems, accelerators and the POWER9 chip in the OpenPOWER system. Our primary development hardware has therefore been JURECA, a cluster system with both GPUs and a KNL *booster*[2], and Ouessant, an OpenPOWER prototype (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 community. The performance analysis workflow can then provide information to assist tuning the application for the target platform.

[2]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.

3.2.1 Porting EasyBuild to an OpenPOWER System

Building upon work carried out as part of D7.2: E-CAM software porting and benchmarking data I[1], where we used EasyBuild to wrap existing compiler and MPI installations into the EasyBuild environment, we developed this approach further for use on the OpenPOWER system.

There are a few restrictions with respect to the environment on an OpenPOWER system. In particular for POWER9 systems, there are no relocatable RPMs available for either CUDA or Spectrum MPI (an IBM OpenMPI fork). For CUDA we were able to design a workaround. For Spectrum MPI there was no similar solution which makes it impossible to implement versioning of this MPI layer in the user environment.

For this reason wrapping the Spectrum MPI installation available on the system was required. This in turn required a number of modifications to EasyBuild to be implemented cleanly:

- Improved logic for generating installed modules,
- Add support for Spectrum MPI to the systemmpi easyblock,
- Improve testing behaviour of EasyBuild for system compilers/MPI.

In addition, when including the math libraries there were some changes required to correctly optimise FFTW installations for the POWER9 architecture:

- Add exceptions for FFTW/3.3.6 on POWER with GCC 5/6/7.

With these enhancements we were able to prepare a feature-complete toolchain (compiler, MPI and optimised math libraries) for use on the OpenPOWER system:

- Add framework support for foss-like toolchain with Spectrum MPI: *gsolf*.

To prove in principle that our toolchain was capable of successfully installing complex software, we built and tested the latest release of GROMACS (`2016.04`) with GPU support.

Porting Performance Analysis Tools to OpenPower

In order to execute our workflow on the system we would require a Scalasca installation. The dependency tree of Scalasca is deep and complex, with a total of 69 direct or indirect dependencies. This does not include a further 73 packages required to build a recent version of X11, of which most use Autotools for installation. In 63 of these cases, the version of Autotools used did not take into account the POWER9 architecture, therefore requiring patching.

The code instrumentation tool used by Scalasca is Score-P, however the current release does not build on POWER9 architectures. After contact with the developers, it was clear the development version could build correctly and we were able to create a small patch that allowed the release version to also build successfully (but only with GCC compilers).

Ultimately, we were able to correctly build and install a version of Scalasca configured to also instrument CUDA applications, which is important given that the main computing power of the OpenPOWER system lies in the GPUs.

3.2.2 Porting ESPResSo++ to EasyBuild

ESPResSo++ (see Section 4.4) is a python package that interfaces with a C++ library which does the heavy computation. Since it uses `CMake` as its build system we were able to leverage pre-existing support within EasyBuild for this type of package. We built ESPResSo++ for 3 different toolchains on JURECA in order to compare performance and these will now be available by default on JURECA in future:

- Add ESPResSo++ support to JURECA

The one caveat is that currently there is no way to configure ESPResSo++ to correctly pick up it's FFTW support from MKL.

3.2.3 Porting OPS to JURECA

OPS (see Section 4.1) is itself a pure python package which are already well supported by EasyBuild. The complexities arise due the reliance of OPS on a separate MD engine. In the current deliverable, 2 new MD engines were integrated, LAMMPS and GROMACS:

- OPS requires the Python interfaces to LAMMPS. While LAMMPS was already available on JURECA, the Python interfaces were not built. Including support for this on JURECA required changes to the LAMMPS easyblock and LAMMPS easyconfigs[3].
- In order to be able to draw a comparison between the default engine of OPS (OpenMM) and GROMACS, we required a GPU build of GROMACS on JURECA (since the GPU is the preferred workhorse for both). While GROMACS was installed, we needed to create a new GROMACS installation on JURECA that included GPU support. This build leveraged an MPI implementation that is specifically configured to optimise communication for GPUs.

3.2.4 Porting DL_MESO to multiple GPUs

Despite the single GPU version of DL_MESO showing good performance (see Section 4.4.2), only a relatively small number of particles (<5M) can be simulated due to the limited memory of the Tesla card (16GB). A multi GPU version is currently under development to extend this limit to very large system (>500M particles) which is often representative of practical industrial applications.

From the hardware perspective, this version will work for both intra and inter node GPUs using the latest NVlink high bandwidth connection (where available) and will be tested on the Swiss National Supercomputer Piz Daint (where each node contains 1 Tesla P100 Card).

From the software perspective, three different types of communication pattern will be available (with the decision made via a pre-processor directive): 1) Peer-to-Peer (P2P) transfers; 2) CUDA_MPI aware and 3) CUDA_MPI aware with Remote Direct Memory Access (RDMA). All these are available as GPUDirect technologies, which is an umbrella term used to refer to high bandwidth and low latency NVidia technologies.

The first multiple GPU version does not contain long range force interactions. This is due to the complexity of solving the Fast Fourier Transformation on several GPUs.

3.3 Optimisation Collaboration with POP Centre of Excellence

One of the services offered by POP is a "Parallel Application Performance Plan". While the performance workflow we use is derived from the collaborative effort between EoCoE and POP, it is a generic approach and therefore has inherent limitations. Given that POP is home to a large set of performance experts, we have collaborated with them on (to date) two applications that are of particular interest to E-CAM with respect to extreme scalability: `ESPResSo++` and `PaPIM`.

3.3.1 Enabling Score-P to target applications with Python as the primary interface

In the case of `ESPResSo++` (see Section 4.4.3) we immediately ran into the issue that the design of the application (a Python interface to a C++ library) had never been encountered by Score-P (which does the code instrumentation on behalf of Scalasca) and it was not immediately capable of analysing it. The creation of Python bindings for Score-P only began in May 2017 and they are still a work-in-progress with no official release to date.

Being able to use `ESPResSo++` with Score-P via the bindings required a long interaction with the Score-P developers in order for us to be able to gather a measurement. This discussion had guided some of the development of the bindings with the result that tracing C and C++ libs from Python should now be supported by default. These developments are important to E-CAM as it is explicitly this model (a Python interface to a C++ library) that E-CAM recommended to it's user community in D7.3 Hardware Developments II[3].

ESPResSo++

The development discussion did not conclude until January 2017 and we are still in contact with POP in order to analyse the code itself. To date we have seen that the initialisation and I/O steps of `ESPResSo++` are effectively serialised (making them very expensive at scale) but a deeper analysis of the more computationally expensive part of the application is still underway (at the time of writing).

[3]An easyblock is a generic installation handler for a specific software package, an easyconfig is a recipe for specific versions of the software package.

3.3.2 PaPIM

POP also carried out a study of `PaPIM` (also see Section 4.3.2). This resulted in a 10 page report on the performance of `PaPIM`. The report focussed mainly on computationally expensive part of the application.

The report highlights load imbalance as an issue in the initialisation phase. The imbalance is derived from the fact that master process handles all I/O and distributes the data to the tasks resulting in a lot of wait states. The suggestion is to replace this communication to the the master process with parallel I/O to the filesystem through a library such as SIONlib.

In the computationally expensive part of the application, load imbalance is also an issue. This is mainly related to an uneven spread of the sample groups among the MPI tasks. Of more interest was the communication pattern, where the POP analysis showed that replacing a number of successive collective communications with a single collective of a derived data type could lead to a 4.7× improvement in communication performance.

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 Classical Molecular Dynamics

Trajectory sampling (also called “path sampling”) is a family of methods for studying the thermodynamics and kinetics of rare events. In E-CAM Deliverable D1.1: Identification/selection of E-CAM MD codes for development[4], we provided an overview of existing software for rare events, and found that the software package Open Path Sampling (OPS) was the optimal choice for E-CAM development. OPS is an open-source Python package for path sampling *that wraps around other classical MD codes*. In Section 4.2 of D1.1, we highlighted several areas where E-CAM could make useful contributions to OPS.

4.1.1 Relevance for E-CAM

OPS was the subject of the first ESDW in Classical Molecular Dynamics (held from the 14th to the 25th of November 2016 in Traunkirchen, Austria). The ESDW focussed on making OPS feature complete in order to make the first official release of the application. As such, there were no performance related modules developed. In light of this, in the deliverable D7.2: E-CAM Software Porting and Benchmarking Data I[1] we instead benchmarked two applications that were targets for inclusion as MD engines of OPS: LAMMPS[4] and GROMACS[5]. Since the MD engines perform the vast majority of the computational work of OPS, the efficiency of these engines in the context of E-CAM inputs would have a clear impact on the scalability of OPS itself.

During the second ESDW in Classical Molecular Dynamics (held from 14th to the 25th of August in Leiden, Netherlands) the OPS interface for GROMACS and OPS interface for LAMMPS were completed and added to the E-CAM Library. In this deliverable we analyse the overhead introduced by OPS due to the interfacing, the scalability of the applications themselves has already been addressed as described above.

4.1.2 OPS with LAMMPS

To test the overhead added by OPS when using (the Python interface of) LAMMPS as the engine, the Lennard-Jones test (32K atoms) case presented in deliverable D7.2[1] has been used.

OPS gathers a frame[6] after a defined number of time steps. In Table 1, OPS has been set to 100 time steps (more frequent queries) and 1000 time steps between frames (less frequent queries) using the JURECA Supercomputer. For a total of 100K time steps, the Molecular Dynamics (MD) engine time (i.e., LAMMPS only time) and the total time (OPS + LAMMPS) using different number of nodes (with 24 cores per node) is presented.

Using 100 or 1000 time steps per frame, the overhead due to the OPS is within a maximum of 3%. However, one should note that when increasing the number of time steps per frame to 1000, the time spent in the MD engine decreases due

[4]LAMMPS is a classical molecular dynamics code that models an ensemble of particles in a liquid, solid, or gaseous state. It can model atomic, polymeric, biological, metallic, granular, and coarse-grained systems using a variety of force fields and boundary conditions. It is open source and available on the LAMMPS website. The version used in the current benchmarks is the 11 August 2017 (LAMMPS release versions corresponds to the actual day of release).

[5]GROMACS is a molecular dynamics package primarily designed for biomolecular systems such as proteins and lipids. It is a free software and it is one of the most widely used codes in the field. The source code can be downloaded from the GROMACS website.

[6]A *frame* describes the state of the physical system at a point in time, and in molecular dynamics, typically consist of coordinates, velocities, and periodic cell vectors.

Steps between frames	# Nodes	MD Engine time [s]	Total time [s]
100	1	106.25	108.48
	2	61.40	62.31
	4	36.46	37.28
1000	1	100.93	102.48
	2	53.56	54.45
	4	29.71	30.67

Table 1: Overhead of OPS when using with LAMMPS with 100/1000 time steps between frames (and a total of 100K time steps) and a simple Lennard-Jones test (32K atoms) case.

h

# GPUs	MD Engine time [s]	Total time [s]
1	352.63	353.84
2	271.56	272.73
4	230.08	231.27

Table 2: Overhead of OPS when using GROMACS as the MD engine with 100 time steps between frames with the Lysozyme test case (protein surrounded by water for a total of 11K atoms).

to less overhead from stopping and starting the engine. A suggested improvement to OPS has been to allow the engine to continue the trajectory while the frame is being evaluated by OPS, which should help eliminate this overhead. The OPS overhead remains relatively static and there is little discernible difference between the overheads for the two measurements. Given that OPS is effectively a serialisation point for the calculation, more intensive trajectories should also, therefore, lead to improved scalability results since they will reduce this ratio of serial to parallel workload. This is reflected in Section 4.1.3, where a much more demanding system is simulated.

Details about how to use OPS with LAMMPS are provided in the OPS iPython notebook for LAMMPS.

4.1.3 OPS with GROMACS

Unlike LAMMPS, GROMACS does not have a Python interface, this forces OPS to fork a process and interact with the running instance via the filesystem. This approach is likely to be slower since it forces writing to disk for the MD engine so that OPS can read the frame. However, it is also efficient in the sense that the engine continues to run the trajectory while the written file is evaluated by OPS. GROMACS does support signals and can be stopped by sending the kill signal to the master process.

For testing the impact of OPS with the GROMACS engine, the Lysozyme test case (see deliverable D7.2[1]) has been used here. This consists of a protein surrounded by water for a total of 11K atoms. The simulation runs for 0.6ns generating 41 frames (OPS sends a kill signal to GROMACS after the 40th frame). The engine is accelerated using the NVidia Tesla K80 GPUs available on the JURECA Supercomputer. Table 2 shows the impact of OPS using 1,2 and 4 GPUs when using GROMACS as the engine.

The overhead is within a maximum of 0.5%, and reflects the expected findings of 4.1.2 where a more demanding test case results in lower percentage overhead. The scaling from 2 to 4 GPUs is, however, very poor. This is not related to OPS but due to the small size of the system considered which is not enough to exploit the parallelism available from so many GPUs. How to use GROMACS efficiently with multiple GPUs is well documented on the NVIDIA website.

Details on how to run an example such as this is provided in the merge request for the OPS interface for GROMACS.

4.2 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. In this second porting and benchmarking deliverable we focus on 2 modules of this kind. The first is LibOMM, a library created during the first WP2 ESDW to implement the orbital minimization method for solving the Kohn-Sham equation as a generalized eigenvalue problem. LibOMM finds the set of Wannier Functions (WFs), from which the density matrix can then be calculated. It is integrated to ELSI, a software bundle for some electronic structure libraries, which we leverage in gathering our benchmarks.

We have also benchmarked the parallel component of Wannier90 code (the code which calculates the Maximally-Localised Wannier Functions (MLWFs)): postw90 which performs Wannier interpolation using MPI parallelisation, in particular the Boltzmann transport functionality added by Giovanni Pizzi at EPFL.

4.2.1 Relevance for E-CAM

Both of the applications addressed were subjects of the first and third ESDW events respectively. The documentation page of related modules produced during the ESDW events can be found at relevant section of the E-CAM library.

4.2.2 LibOMM

LibOMM, is a library created to use the exact cubic-scaling solver for the Kohn-Sham problem in order to improve the Kohn-Sham code scalability compared to an explicit diagonalization method in realistic systems. The link to the associated E-CAM module can be found here. This module was reported in D2.1: Electronic structure E-CAM modules I[5]. This library solves the Kohn-Sham equation as a generalized eigenvalue problem for a fixed Hamiltonian defined in the Orbital Minimization Method. In order to test the scalability of the LibOMM library, it is at first integrated with ELSI (ELectronic Structure Infrastructure). The input is taken from the `TOMATO_Seed` folder of that code.

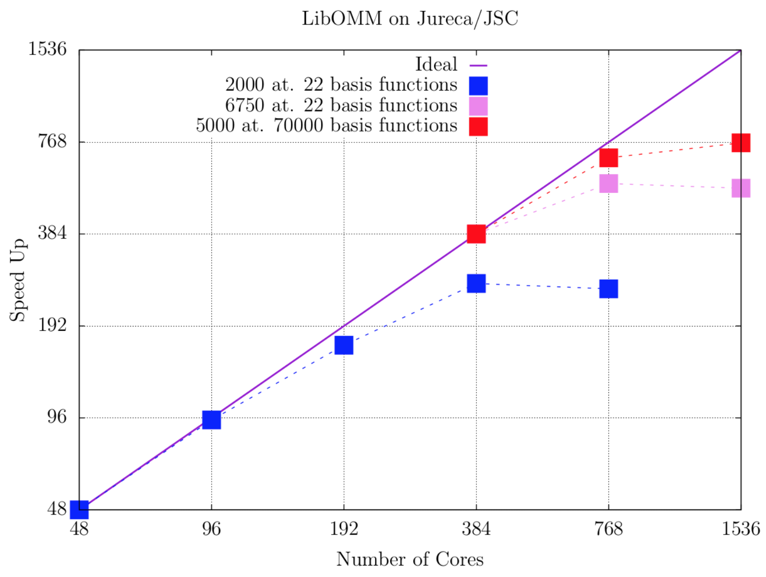
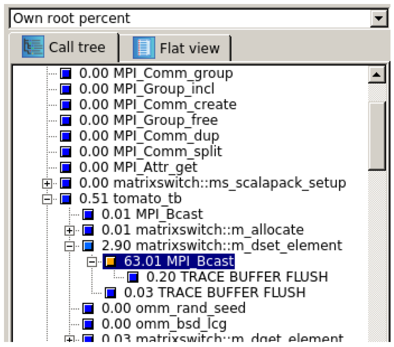


Figure 2: LibOMM code. Strong scaling test on JURECA of JSC. The silicon supercell size parameter is 10 × 10 × 10 (blue line, with 2000 atoms and 22 basis functions) and 15 × 15 × 15 (pink line, with 6750 atoms and 22 basis functions) and the largest case a 50 × 50 × 1 supercell (red line, with 5000 atoms and 70000 basis functions).

In Fig. 2 we show the scaling behaviour of the code for the example inputs (a 10 × 10 × 10 supercell with 2000 atoms and 22 basis functions, and a 15 × 15 × 15 supercell with 6750 atoms and 22 basis functions) plus a further example with a 50 × 50 × 1 supercell. The example with a 10 × 10 × 10 size supercell fails to scale well with 768 processes due to the relatively small supercell dimension used. When the supercell dimension is small, running on large scale computing resources makes the internode MPI communication time dominant compared to parallel calculation section, which heavily impacts the scalability. With the more intensive test, the 15 × 15 × 15 supercell, the scaling curve improves. With the added 50 × 50 × 1 supercell test, we confirm a return towards adequate scalability when the workload is high enough. The inherent scaling limitation is quickly reached again however.

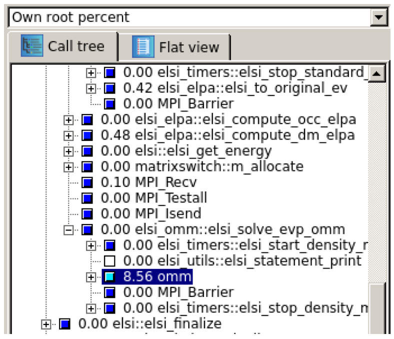
As a matter of fact, ELSI introduces an artificial barrier to our scaling analysis. The matrix generation step required to transform the `Tomato_SEED` input such that it can be used by LibOMM is part of the ELSI code (i.e., it is not part of LibOMM). For the case of 2000 atoms and 22 basis functions with 192 cores, this matrix generation step takes 209 seconds compared to only 80 seconds used by the LibOMM code itself. This makes the LibOMM scaling test procedure much slower and cumbersome than necessary, with 63% of the runtime taken up by highly inefficient MPI usage from the matrix generation and only 9% used by LibOMM (shown by the Scalasca analysis in Figures 3 and figure 4).



```
Own root percent
Call tree    Flat view
0.00 MPI_Comm_group
0.00 MPI_Group_incl
0.00 MPI_Comm_create
0.00 MPI_Group_free
0.00 MPI_Comm_dup
0.00 MPI_Comm_split
0.00 MPI_Attr_get
0.00 matrixswitch::ms_scalapack_setup
0.51 tomato_tb
  0.01 MPI_Bcast
  0.01 matrixswitch::m_allocate
  2.90 matrixswitch::m_dset_element
    63.01 MPI_Bcast
      0.20 TRACE BUFFER FLUSH
    0.03 TRACE BUFFER FLUSH
  0.00 omm_rand_seed
  0.00 omm_bsd_lcg
  0.03 matrixswitch::m_dget_element
```

.

Figure 3: MPI_Bcast subroutine on Tomato-Matrix-Generation represents 63% of the runtime.



```
Own root percent
Call tree    Flat view
    0.00 elsi_timers::elsi_stop_standard
    0.42 elsi_elpa::elsi_to_original_ev
    0.00 MPI_Barrier
  0.00 elsi_elpa::elsi_compute_occ_elpa
  0.48 elsi_elpa::elsi_compute_dm_elpa
  0.00 elsi::elsi_get_energy
  0.00 matrixswitch::m_allocate
  0.10 MPI_Recv
  0.00 MPI_Testall
  0.00 MPI_Isend
  0.00 elsi_omm::elsi_solve_evp_omm
    0.00 elsi_timers::elsi_start_density_r
    0.00 elsi_utils::elsi_statement_print
    8.56 omm
    0.00 MPI_Barrier
    0.00 elsi_timers::elsi_stop_density_n
0.00 elsi::elsi_finalize
```

.

Figure 4: LibOMM related subroutines represent only 8.6% of the runtime.

Further analysing this behaviour, we see millions of calls to MPI_Bcast for the matrix generation step, representing 36% of the total number of function calls. This has a the consequence that the total memory required for tracing analysis increases to 600 GB and the tracing procedure is unable to complete. Without modifying ELSI we cannot, therefore, then correctly create the LibOMM's parallel performance evaluation table. To give LibOMM fully considered recommendations would require an alternative implementation of the matrix generation in ELSI (which could as simple as generating the matrix separately instead of generating it on-the-fly). The ELSI developers have attended previous ESDW events and will attend the next WP2 ESDW where we can address this problem with them.

4.2.3 Wannier90

postw90, is the only parallel component in the `v2.1` release of Wannier90 code. This release was finalised during the E-CAM ESDW held in San Sebastian during September 2016 where 20 people attended the Wannier90 coding week to develop new features and work towards that release. Nine E-CAM modules were produced during this ESDW and reported in D2.3: Electronic structure E-CAM modules II[6]. `postw90` performs Wannier interpolation using MPI parallelisation, in particular with Boltzmann transport functionality. To create inputs for this, QuantumEspresso is firstly run to get the orbitals and wavefunction basis for the test case.

Following the procedures of Wannier90 tutorial, the results on different HPC platforms are shown in Figures 5 and 6. `postw90` scales well up to about 700 cores for the test case with 64 silicon atoms.

For `postw90`, running on 1536 cores takes just 26 seconds. This step is only part of an overall workflow, however, with the main Wannier function calculation before this step. This uses the *wannier90* executable which (for the current release version) is sequential and takes more than 3 hours to prepare the input. The main barrier for the scalability of the Wannier90 workflow is therefore clearly, at present, the sequential component in that workflow. This is already a development priority within the Wannier90 development community, with a relevant pull-request related to the parallelization efforts of this already in place. For completeness, we provide in Table 3 the performance table for `postw90` but this has little relevance given the work imbalance in the overall workflow.

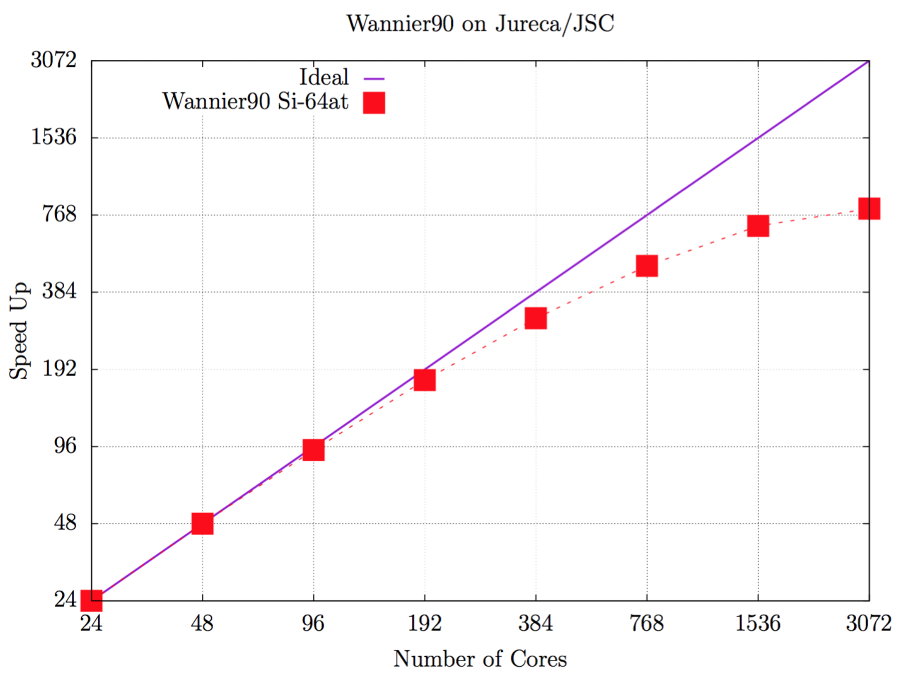


Figure 5: Wannier90 code (post-processing). Performance evaluation on Jureca/JSC

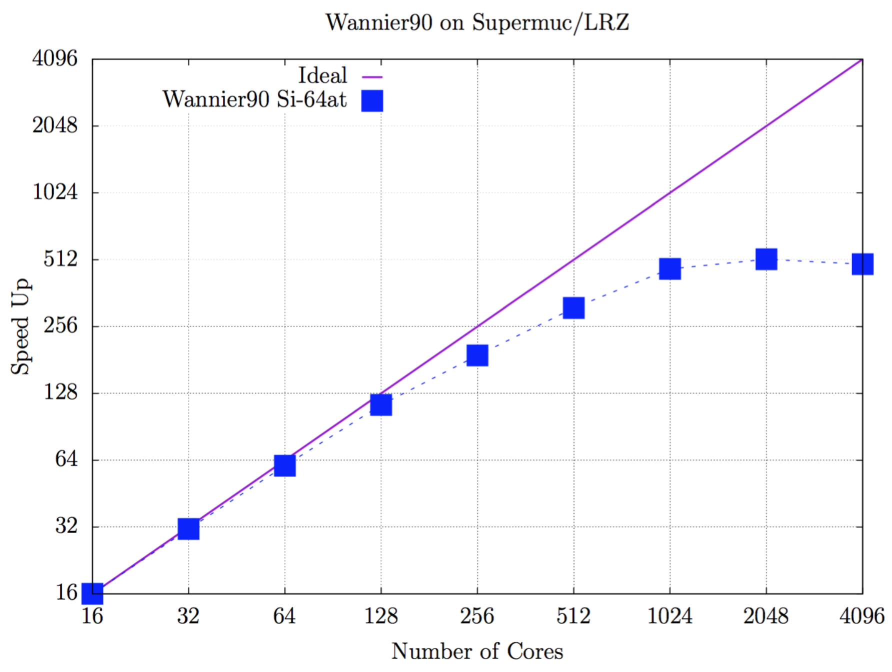


Figure 6: Wannier90 code (post-processing). Performance evaluation on Supermuc/LRZ.

4.3 Quantum Dynamics

Uniquely in the Quantum Dynamics (WP3) package, there are no well-established community codes. In the ESDW events organized in 2016 and 2017, either new application codes or parallel implementations of existing codes were developed. We have tested new module developments in the following codes that were targeted there: PaPIM and QUANTICS. In particular we evaluate the extreme scaling behaviour of the latest version of PaPIM and the performance of an OpenMP parallelisation of QUANTICS.

	Metric name	24 cores	48 cores	96 cores	192 cores	384 cores	768 cores	1536 cores	3072 cores
Global	Total Time (s)	739	372	192	104	80	64	54	48
	Time IO (s)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Time MPI (s)	15.4	2.4	4.3	10.3	51.7	12.4	29.4	18.4
	Load Imbalance (%)	1.4	2.4	4.3	6.5	11.7	10.3	17.2	14.7
IO	IO Volume (MB)	548	548	548	548	548	548	548	548
	Calls (nb)	70543	70543	70543	70543	70543	70543	70543	70543
	Throughput (GB/s)	2.5	5.1	11.0	29.5	32.5	47.5	22.0	40.1
	Individual IO Access (kB)	8	8	8	8	8	8	8	8
MPI	P2P Calls (nb)	0	0	0	0	0	0	0	0
	Collective Calls (nb)	123	123	123	123	123	123	123	123
	Collective Calls (s)	14.9	10.5	9.9	8.7	11.9	8.6	13.5	10.2
	Coll. Calls Message Size (MB)	N.A.	N.A.	N.A.	19.2	N.A.	12.2	3.3	19.2
	Synchro / Wait MPI (s)	9.1	7.6	7.3	7.1	8.2	7.2	8.5	7.5
	Ratio Synchro / Wait MPI (%)	59	66	66	69	16	57	29	41
Mem	Memory Footprint (MB)	800	878	907	964	2692	1080	1771	1309
	Cache Usage Intensity	0.91	0.91	0.92	0.92	0.90	0.91	0.91	0.91
OMP	Time OpenMP (s)	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
Core	IPC	2.74	2.74	2.73	2.74	2.66	2.74	2.71	2.73

Table 3: Performance metrics for `postw90` on JURECA with inputs from Silicon Supercell of 64 atoms, related to Figure 5).

4.3.1 Relevance for E-CAM

Both of the applications addressed were subjects of the first and second WP3 ESDW events. The content included here are evaluations of the parallelisation-focussed developments since the initial WP3 ESDW. The related E-CAM modules for PaPIM code are PIM_wd and PIM_qcf. The related module for Quantics code is the Quantics OpenMP Improvements Module.

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.

The CH5+ molecule with a classical sampling method is used as input. In the `INPUT` file, `NTraj`, the number of molecular dynamics trajectories, is set from 4096 to 245,760 on JURECA and up to 3,932,160 on JUQUEEN.

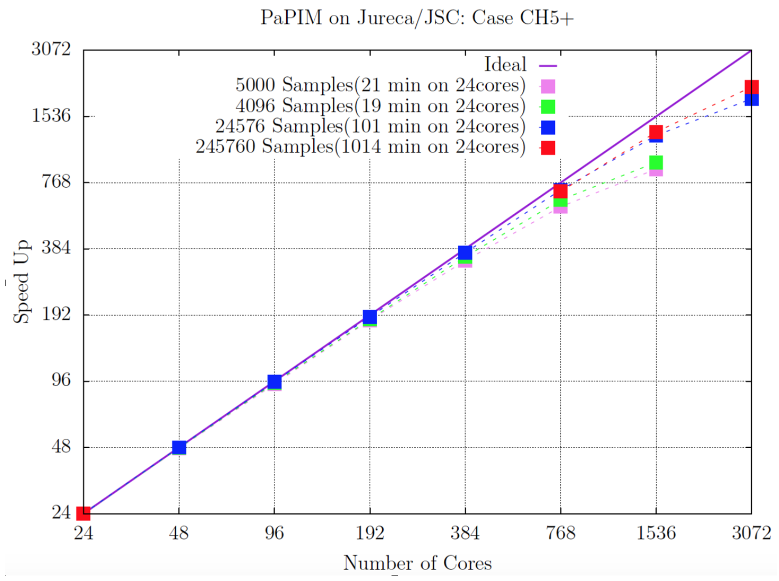


Figure 7: PaPIM code. Strong scaling test with high task load imbalance (as evidenced in Table 4).

Figure 7 shows the strong scaling results on JURECA from 1 to 3072 cores. The results show that the PaPIM code scales well up to 3072 cores with 245760 samples. The larger the number of samples (i.e., trajectories) the better the

	Metric name	24 cores	48 cores	96 cores	192 cores	384 cores	768 cores	1536 cores
Global	Total Time (s)	1244	629	321	166	91	58	53
	Time IO (s)	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Time MPI (s)	20.4	17.2	16.2	9.8	13.5	15.5	25.4
	Load Imbalance (%)	2.3	3.4	5.9	4.9	11.5	19.5	23.0
IO	IO Volume (MB)	0.0	0.0	0.0	0.0	0	0	0
	Calls (nb)	17	17	17	17	17	17	17
	Throughput (MB/s)	0.6	5.5	0.5	18.0	50.8	10.5	30.9
	Individual IO Access (kB)	0.8	0.8	0.8	0.8	0.8	0.8	0.8
MPI	P2P Calls (nb)	0	0	0	0	0	0	0
	Collective Calls (nb)	128	128	128	128	128	128	128
	Collective Calls (s)	20.4	16.6	15.1	7.9	9.9	8.4	8.9
	Coll. Calls Message Size (kB)	0.6	0.9	1.5	2.6	4.8	9.2	18.2
	Synchro / Wait MPI (s)	20.4	16.6	15.1	8.0	9.9	8.4	8.9
	Ratio Synchro / Wait MPI (%)	95.4	94.5	91.8	76.7	71.7	51.7	34.2
Mem	Memory Footprint (MB)	8.4	26.9	55.8	113.5	229.2	459.9	921.3
	Cache Usage Intensity	0.97	0.98	0.98	0.98	0.96	0.96	0.98
OMP	Time OpenMP (s)	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.
Core	IPC	0.73	0.66	0.65	0.64	0.69	0.68	0.63

Table 4: Performance metrics for PaPIM on JURECA with input of 5000 Samples (the pink line in Figure 7).

scalability.

A deeper analysis with Scalasca was used to create Table 4, which shows increasing load imbalance as the number of cores is increased (the result shown is for 5000 trajectories). The reason for this is clear when one considers that the number of samples remains static even as the core count increases, resulting in an increasingly uneven distribution of work. The total time spent in the MPI calls is dominated by wait states in collective communications and this has been analysed in detail by the POP CoE (see Section 3.3.2 for a summary of their feedback). We also see that the Instructions Per Cycle (IPC) are relatively low. The developers advised that this is because currently an analytic potential is being used, ab initio calculations to obtain the potential energies will be far more computationally intensive.

We were interested in evaluating the extreme-scale behaviour of PaPIM and leveraged resources on JUQUEEN in order to investigate the code behaviour at very high core counts. The results of this investigation are presented in Figure 8. Due to the hardware characteristics of JUQUEEN, it is recommended to use more threads/tasks than there are physical cores (if the application is able to support this). For this reason we choose a more generic metric for code performance: the time to solution. Since we do the investigation for a number of different ensemble sizes we use the time spent per sample as our metric.

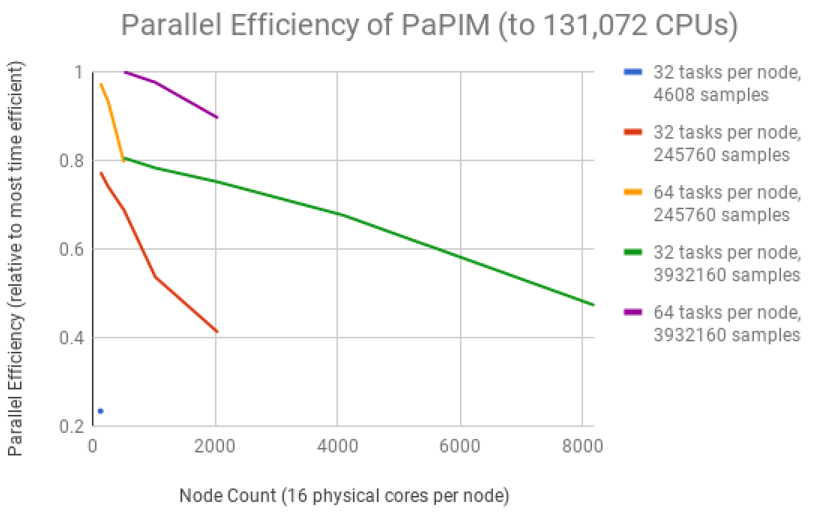


Figure 8: PaPIM performance on JUQUEEN up to 131,072 CPUs (and 262,144 MPI tasks). The parallel efficiency on the X-axis is the time per sample relative to the most time-efficient result, the Y-axis is the node count (with 16 physical cores per node).

What we see is that the number of samples has a dramatic effect on the efficiency: there is almost a factor of 4-5 improvement in efficiency when we move to large numbers of samples. What we also see is that overloading the cores also improves the efficiency: when we compare 64 to 32 tasks per node we consistently see improved performance. Both of these observations verify our conclusion drawn in relation to Table 4): the sample sizes affect efficency and the cores are typically underutilised. We were able to scale PaPIM up to 8192 nodes (which is 131,072 CPUs and 262,144 MPI tasks). With 64 MPI tasks per node, we ran out of memory at 4096 nodes. The efficiency of PaPIM is quite impressive, with 50% efficiency at the largest size. However, this is when comparing to our most efficient result which was with 64 tasks-per-node, if we compare like with like and only use the 32 tasks-per-node results the efficiency is closer to 60%.

4.3.3 QUANTICS

The QUANTICS package solves the time-dependent Schrödinger equation to simulate nuclear motion by propagating wavepackets. The focus of the package is the Multi-Configurational Time-Dependent Hartree (MCTDH) algorithm. The code is Fortran90-based with parallelisation using MPI, and also OpenMP in parts of the code.

The QUANTICS code version used here is 1.1, shared with E-CAM by Professor Gramham Worth. "Pyrazine, 24 modes" is used as input.

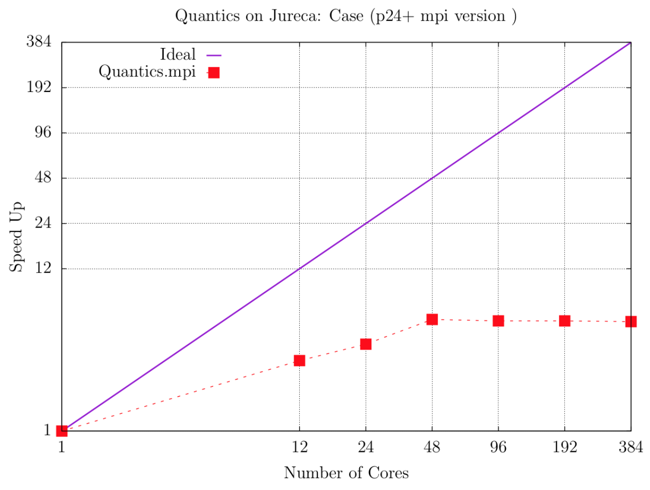


Figure 9: QUANTICS MPI version: strong scaling test.

Figure 9 shows strong scaling results on JURECA from 1 to 384 cores. The results show that the QUANTICS code scales relatively poorly even at low core counts. In the previous deliverable, the application was not capable of running on more than 1 node (16 cores in MareNostrum III). While there is an improvement this year in that the code can run at larger core counts, the efficiency is still very problematic.

In Table 5, our workflow provides an initial analysis of this MPI-based performance. Firstly, we see extremely large load imbalance even at small core counts. The MPI calls are taking up more than 74% of the runtime at the largest core count. In general, they are also large numbers of collective calls, which together with the load imbalance are the root of the poor scalability problem. Another point to note is that the cache usage intensity is also very low.

Upon further investigation, we found that Quantics adopts a simple master/slave strategy where data for tasks is distributed and then subsections of the data are updated by the individual MPI tasks. The origin of the imbalance is an `if/else` call within the main `do` loop:

```fortran
! Divide up the work.
      call mpe_decomp1d(i1,i2,k1,k2)
      do k=k1,k2
         s=kf(k)
         s1=ki(k)
        if (s1 .ne. s) then
            call matrizen2(k,s,s1,psi,dtpsi,hteil,hloch,psi1,&
                 modus)
         else
            call matrizen1(k,s,psi,dtpsi,hteil,hloch, psi1,&
                 modus,lsym)
         endif
      enddo
```

	Metric name	1 core	12 cores	24 cores	48 cores (2nodes)
Global	Total Time (s)	65537	22303	17325	11870
	Time IO (s)	141.6	5.3	3.1	1.6
	Time MPI (s)	128	12045	10559	8782
	Load Imbalance (%)	0	53.8	63.4	66.3
IO	IO Volume (MB)	28898	29083	28990	28990
	Calls (nb)	3712992	3736724	3724858	3724858
	Throughput (MB/s)	204.1	5472.0	9367.7	17878.5
	Individual IO Access (kB)	8	8	8	8
MPI	P2P Calls (nb)	0	0	0	0
	Collective Calls (nb)	1227050	1231876	1290561	1234606
	Collective Calls (s)	127.6	12044.8	10549.5	8781.5
	Coll. Calls Message Size (kB)	2078.1	2076.6	2029.2	-
	Synchro / Wait MPI (s)	0	5880	6258	5754
	Ratio Synchro / Wait MPI (%)	0	53.8	63.4	66.3
Mem	Memory Footprint (MB)	908	884	884	1324
	Cache Usage Intensity	0.83	0.79	0.79	0.79
OMP	Time OpenMP (s)	N.A.	N.A.	N.A.	N.A.
Core	IPC	1.9	2.02	1.99	2.20

Table 5: Performance metrics for Quantics (MPI version) on JURECA with the pyrazine input (the red line in Figure 9).

We observed in the Scalasca analysis that the load of `matrizen1` and `matrizen2` is very different (the developers agreed a factor of 2 is to be expected). In addition the distribution of the branching is not random, but falls into two regions with fluctuations that seem to only occur in the border between the two regions. The end result of this is that the group of processes in the region where `matrizen1` is used have a very different load to those in the `matrizen2` region. Given that it would appear that the same initial data set appears across all processes, one solution is to implement a load-balancing mechanism that increases the number of effective tasks and distributes them dynamically.

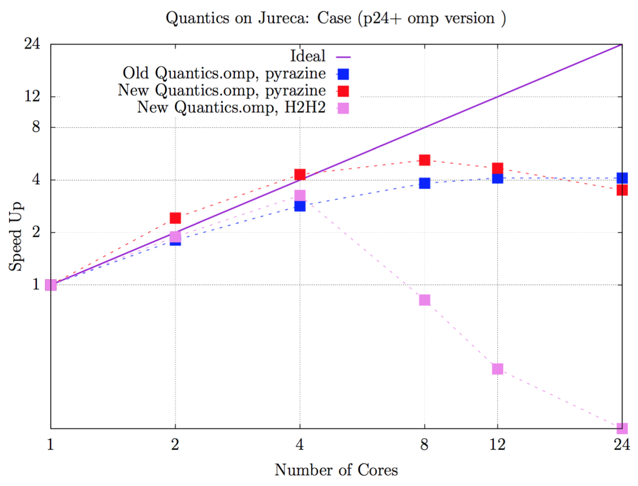


Figure 10: QUANTICS OpenMP version: strong scaling test.

During the second ESDW, the existing OpenMP implementation in QUANTICS was improved, adopting the same task-based approach but distributing the workload over the `do` loop via OpenMP pragmas in the relevant region (as well as elsewhere). The scaling behaviour in this case is shown in Fig. 10.At low core counts we see good scalability, which is, for the most part, significantly improved over that the previous implementation. However, the scalability plateaus early leading to behaviour similiar to the MPI case. We also investigated another case (H2H2) but did not observe markedly different results.

Taking into consideration the performance of both the MPI and OpenMP implementations, we felt that the application requires a fundamental change in the approach to the parallelisation that has the potential to deliver a more balanced workload. Specifically, we recommended to the developers to use OpenMP paralleisation for their custom linear algebra routines (which require roughly 90% of the computaional load) and only use the `do` loop parallelisation for distribution across nodes with balancing for the fact that the ration workload of the two paths is a known quantity. Given the significant performance issues, we have written a letter of support for the researchers to seek eCSE funding for dedicated professional support to improve the efficiency and scalability of the code.

4.4 Meso- and Multi-scale Modelling

As part of the (WP4) package the following two modules are presented:

- improvements of the DL_MESO_DPD porting on GPU
- integration of ESPREsSo++ with Easybuild and JUBE

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 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).

ESPREsSo++ is used in the Pilot Project on Rheological Properties of New Composite Materials developed by the E-CAM PDRA Dr. Hideki Kobayashi. The porting and benchmarking work carried out with ESPREsSo++ on different architectures was done in order to evaluate the impact of the new modules being developed during the pilot project.

4.4.2 Performance Improvements of DL_MESO_DPD on GPUs

With the introduction of CUDA 6.0 compute capability (for Pascal architecture and above) the double precision atomic operation has been introduced and greatly improved. This allows one to rewrite the main kernel of DL_MESO_DPD (the force field calculation) using the action-reaction Newton principle, halving the number of interactions between particles and exploring only 13 of the neighbour cells determined by the cutoff radius. This improved the speed-up on the P100 NVidia GPU of by a factor ~2× compared to the previous version.

The scheduling of the block threads has also been restructured in order to match one thread per particle during the pair force calculations. This allow a better utilisation of the available resources and allows a ~6× speed-up on the P100 (1 node of Piz Daint) when compared to a 12 cores Intel Haswell processor (E5-2690 v3). See Figure 11 Piz Daint curve.

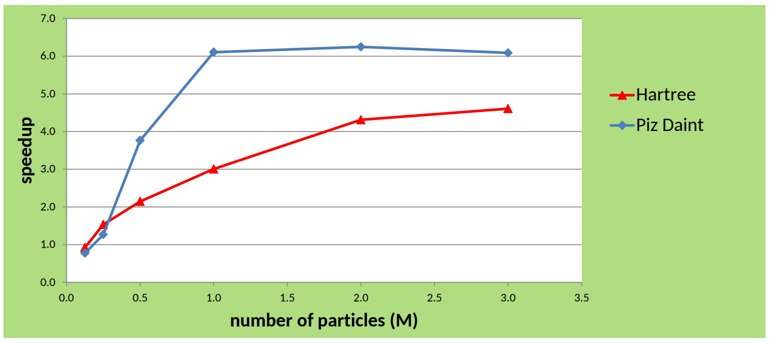


Figure 11: Speed-up of the applicattion on a Tesla P100 GPU as compared to a 12-core Intel Ivy Bridge (Hartree) and 12-core Intel Haswell (Piz Daint) CPUs.

These latest improvements have been presented to the GPU Programming with CUDA course from PRACE, hosted at Daresbury Laboratory (UK) in November 2017, as a GPU Case Study for scientific application. The test case used is a plasma of charged particles with a number ranging from 0.5M to 5M. A comparison between the GPU version on a Tesla P100 and a 12 cores Ivy Bridge Intel processor showed a maximum speedup of a factor 4.5 when the number of particles is > 3M (see the Hartree curve in Figure 11).

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.

# Nodes	Time [s] on JURECA	time [s] on Marconi
1	23.191	37.67
2	11.334	18.89
4	5.862	9.33
8	2.972	4.81
16	1.427	2.37
32	0.639	-
64	0.368	-
128	0.216	-

Table 6: Strong Scaling Performance table for ESPResSo++ obtained using JUBE.

4.4.3 ESPResSo++ with Easybuild and JUBE

ESPResSo++ can now be installed through EasyBuild with both Intel and GCC compilers. The integration with JUBE is, however, currently limited to simple performance metrics due to the Python interpreter used by ESPResSo++ conflicting with the instrumentation of the code (as described in Section 3.3.1). The resulting limited JUBE file has been used to benchmark ESPResSo++ using a polymer-melt system.

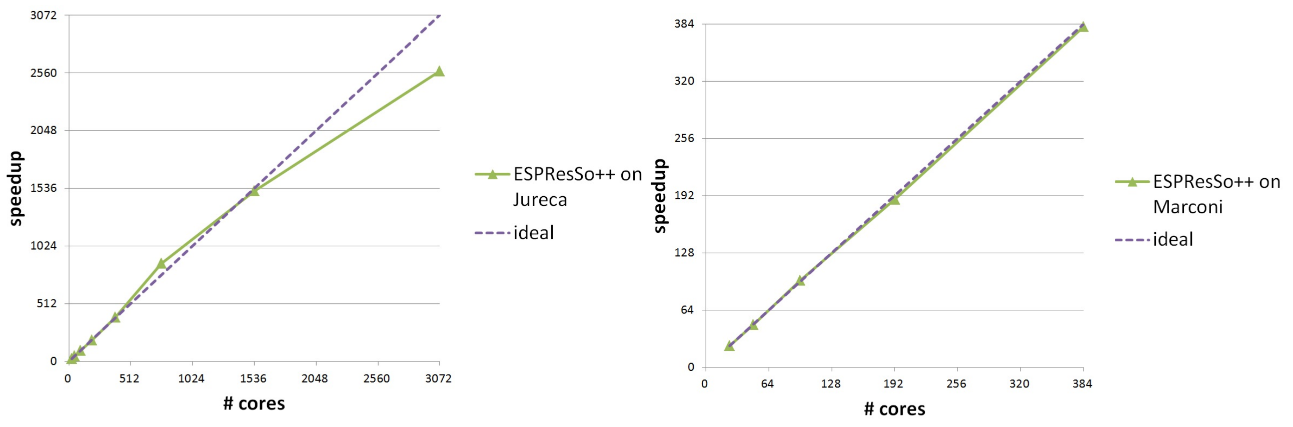


Figure 12: Scalability plot for ESPResSo++ on Jureca and Marconi supercomputers.

The test case considered contains 5M particles and it has been run for 1000 time steps on the JURECA supercomputer, where each node contains 24 Intel Haswell cores, and on Marconi where each node is made of 32 Intel Broadwell cores. Table 6 reports the strong scaling analysis based on the time per time step metrics obtained with JUBE. We clearly observe indications of good scaling behaviour but the runtime of particular benchmark is very short (at only $0.2s$ for 128 nodes of JURECA, which is a total of 3072 cores) limiting the level of achievable scalability.

We have included limited results on Marconi as we had a number of problems on the system. Larger scale runs failed and we continue to diagnose the problem with the user support team there (it may be related to a recent transition they had to the SLURM scheduler). More worryingly, the timings were slower on Marconi despite the CPU being newer and having a higher core count on the node. At the time of writing, these issues were still being investigated.

In addition, we note that the figures reported here relate to the core kernel calculation and the timings do not include either the initialisation or finalisation steps. We have observed that these steps appear not to be scalable and are most likely serialised, taking an increasing proportion of the runtime as the core count increases. This was most obvious when attempting to run the calculations on the Xeon Phi partition of Marconi, where we had only limited success running the application (only up to 2 KNL nodes was successfully executed). In order to further investigate the scalability of ESPResSo++ at larger core counts, these aspects would first need to be addressed and we are in touch with the developers to do this.

5 Outlook

Going forward the work of WP7 will 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 focusses more on extreme-scale challenges, as recommended by the project reviewers.

Under this revised strategy, WP7 takes a number of additional responsibilities:

1. Create a focus team on WP7 to manage new responsibilities and play a central role in the communication among WPs.
2. Incorporation of two new applications that can verifiably scale to extreme resources. There are two proposals: IMD and MP2C that have been checked to show good scalability at Juelich Supercomputing Center. Resources will have to be allocated to these new tasks. An additional 2-year position was added to WP7 to address this. This position is due to be filled in 2018.
3. Identify scalable applications in WP1-4. Explore the potential relationship between the newly selected applications that can verifiably scale to extreme resources and the ones identified in WP1-4.

In addition, WP7 will engage in a High Throughput Computing (HTC) related project that has likely impact for a number of WPs in E-CAM. This is outlined in further detail in the section below.

5.1 OPS Collaboration with PRACE

OpenPathSampling (OPS) (one of the target applications of the E-CAM Centre of Excellence) makes it easy to perform many variants of transition path sampling (TPS) and transition interface sampling (TIS), as well as other useful calculations for rare events, such as committor analysis and flux calculations. In addition, it is a powerful library to build new path sampling methods. OPS is independent of the underlying molecular dynamics engine, and currently has support for OpenMM, LAMMPS and GROMACS.

Work in the collaboration project with PRACE will be focused on OpenMM and/or GROMACS used as the molecular dynamics engines in the task scheduling library that will be leveraged by OPS. The library will be implemented generically and independently of OPS as the workflow is common to a number of applications relevant to E-CAM.

The implementation is intended to be agnostic to the programming paradigm of the workers (such as MPI/OpenMP/CUDA as in the initial set of applications chosen). The goal is to remove the requirement of interfacing with the resource manager to manage the workload (and in so doing allowing the manager to work "in real-time" with the available resources). This implementation will allow for dynamic decision making where necessary and opens the door to machine learning techniques that are already popular in the field. For communication between master and slaves we will use MPI which is the most common parallel programming model. However, we expect to use spawning techniques that are not in common usage and whose technical implementation varies dramatically among MPI implementations. We will implement the library in Python taking advantage of its flexibility in optimising task scheduling and defining generic tasks that are easy for the user to create.

We target some of the key workload requirements of E-CAM on the path to Exascale. It is the time scales, not the system sizes, that are a common limiting factor in the E-CAM community. This makes many of the scientific use cases of E-CAM HTC applications (with non-independent instances) rather than single instances of extreme-scale applications. This project will allow the E-CAM user community to leverage extreme-scale resources in a transparent and scalable way using the latest algorithms that are optimised for addressing large times for rare event sampling.

This collaboration is an outcome of the Exascale Workshop organised by PRACE WP7 Task 7.2 in June 2017 at Juelich Supercomputing Centre. From the PRACE side, it was a requirement of T7.2 to collaborate with a CoE to help them improve their applications on the road to Exascale by providing them support from a technical level by PRACE experts and run their codes on PRACE Tier-0 systems for better evaluation of potential for future European Exascale systems. From E-CAM side, external collaborations were fully supported and this is a very good example of such collaboration with a strong partner such as PRACE.

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
MD	Molecular Dynamics
OPS	Open Path Sampling
POP	Performance Optimisation and Productivity
EoCoE	Energy-oriented Centre of Excellence
WFs	Wannier Functions
MLWFs	Maximally-Localised Wannier Functions
IPC	Instructions Per Cycle
HTC	High Throughput Computing

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
Internal Project Management Link . . . https://redmine.e-cam2020.eu/issues/45
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 1

OPS . . . http://openpathsampling.org/latest/
LAMMPS . . . http://lammps.sandia.gov/
GROMACS . . . http://www.gromacs.org/
ESDW7 . . . https://www.e-cam2020.eu/event/extended-software-development-workshop-classical-molecular
LibOMM . . . http://e-cam.readthedocs.io/en/latest/Electronic-Structure-Modules/modules/libOMM/readme.html
Wannier90 . . . http://wannier.org
ESDW1 . . . https://www.e-cam2020.eu/event/electronic-structure-library-coding-workshop/
ESDW3 . . . https://www.e-cam2020.eu/event/extended-software-development-workshop-wannier90/
PaPIM . . . http://e-cam.readthedocs.io/en/latest/Quantum-Dynamics-Modules/modules/PaPIM/readme.html
Quantics . . . QUANTICS:About
ESDW6 . . . https://www.e-cam2020.eu/event/extended-software-development-workshop-quantum-md/
DL_MESO_DPD . . . http://www.scd.stfc.ac.uk/support/40694.aspx
ESPResSo++ . . . https://www.openhub.net/p/espressopp
ESDW5 . . . https://www.e-cam2020.eu/event/extended-software-development-workshop-meso-and-multiscale

Page 2

EasyBuild . . . http://easybuild.readthedocs.org/en/latest/
OpenPOWER . . . https://openpowerfoundation.org/

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
Poincare . . . https://groupes.renater.fr/wiki/poincare/public/description_de_poincare
Ouessant . . . http://www.idris.fr/eng/ouessant/
OpenPOWER . . . https://openpowerfoundation.org/
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/
JUQUEEN . . . http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/Configuration/Configuration_node.html
Marconi . . . http://www.cineca.it/en/content/marconi
SuperMUC . . . https://www.lrz.de/services/compute/supermuc/systemdescription/
Curie . . . http://www-hpc.cea.fr/en/complexe/tgcc-curie.htm
Piz Daint . . . https://www.cscs.ch/computers/piz-daint/

Page 6
Spectrum MPI . . . https://www.ibm.com/us-en/marketplace/spectrum-mpi
design a workaround . . . https://github.com/easybuilders/easybuild-easyblocks/issues/1264
Improved logic for generating installed modules . . . https://github.com/easybuilders/easybuild-easyblocks/pull/1261
Add support for Spectrum MPI to the systemmpi easyblock . . . https://github.com/easybuilders/easybuild-easybloc
pull/1275
Improve testing behaviour of EasyBuild for system compilers/MPI . . . https://github.com/easybuilders/easybuild-easyblocks/pull/1282
Add exceptions for FFTW/3.3.6 on POWER with GCC 5/6/7 . . . https://github.com/easybuilders/easybuild-easybloc
pull/1274
Add framework support for foss-like toolchain with Spectrum MPI: *gsolf* . . . https://github.com/easybuilders/easybuild-easyblocks/pull/1274
Autotools . . . https://www.gnu.org/software/automake/manual/html_node/Autotools-Introduction.html
Score-P . . . http://www.vi-hps.org/projects/score-p/
Add ESPResSo++ support to JURECA . . . https://github.com/easybuilders/JSC/pull/3

Page 7
changes to the LAMMPS easyblock and LAMMPS easyconfigs . . . https://github.com/easybuilders/JSC/pull/1
new GROMACS installation on JURECA that included GPU support . . . https://github.com/easybuilders/JSC/pull/2
Piz Daint . . . https://www.cscs.ch/computers/piz-daint/
services offered by POP . . . https://pop-coe.eu/services
Python bindings for Score-P . . . https://github.com/score-p/scorep_binding_python
long interaction with the Score-P developers . . . https://github.com/score-p/scorep_binding_python/issues/1

Page 8
SIONlib . . . http://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/SIONlib/_node.html

Page 9
OPS . . . http://openpathsampling.org/
LAMMPS . . . http://lammps.sandia.gov/

LAMMPS website . . . http://lammps.sandia.gov/
GROMACS . . . http://www.gromacs.org/
GROMACS website . . . http://www.gromacs.org/
OPS interface for GROMACS . . . https://gitlab.e-cam2020.eu/e-cam/E-CAM-Library/merge_requests/11
OPS interface for LAMMPS . . . https://gitlab.e-cam2020.eu:10443/e-cam/E-CAM-Library/merge_requests/25
E-CAM Library . . . http://e-cam.readthedocs.io/en/latest/
JURECA . . . http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JURECA/JURECA_node.html

Page 10
OPS iPython notebook for LAMMPS . . . https://github.com/jhprinz/openpathsampling/blob/1235c472217d32b2601
examples/misc/introduction_lammps.ipynb
JURECA . . . http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JURECA/JURECA_node.html
How to use GROMACS efficiently with multiple GPUs . . . https://www.nvidia.com/en-us/data-center/gpu-accelerated-applications/gromacs/
OPS interface for GROMACS . . . https://gitlab.e-cam2020.eu/e-cam/E-CAM-Library/merge_requests/11
LibOMM . . . https://gitlab.e-cam2020.eu/esl/omm

Page 11
ELSI . . . https://esl.cecam.org/ELSI
Wannier90 . . . http://www.wannier.org
postw90 . . . http://www.wannier.org/ford/program/postw90.html#src
relevant section of the E-CAM library . . . http://e-cam.readthedocs.io/en/latest/Electronic-Structure-Modules/
index.html#extended-software-development-workshops
LibOMM . . . https://gitlab.e-cam2020.eu/ESL/omm
here . . . http://e-cam.readthedocs.io/en/latest/Electronic-Structure-Modules/modules/libOMM/readme.html
Orbital Minimization Method . . . http://esl.cecam.org/LibOMM
ELSI . . . https://wordpress.elsi-interchange.org/

Page 12
postw90 . . . http://www.wannier.org/ford/program/postw90.html#src
Wannier90 . . . http://www.wannier.org
Nine E-CAM modules . . . http://e-cam.readthedocs.io/en/latest/Electronic-Structure-Modules/index.html#extended-software-development-workshops
Wannier90 tutorial . . . http://www.wannier.org/doc/tutorial.pdf
wannier90 . . . http://www.wannier.org/ford/program/wannier.html#src
pull-request . . . https://github.com/wannier-developers/wannier90/pull/125

Page 13
PaPIM . . . https://gitlab.e-cam2020.eu/Quantum-Dynamics/PIM/tree/deliverables
QUANTICS . . . http://stchem.bham.ac.uk/~quantics/doc/index.html

Page 14
PIM_wd . . . http://e-cam.readthedocs.io/en/latest/Quantum-Dynamics-Modules/modules/PIM_wd/readme.html
PIM_qcf . . . http://e-cam.readthedocs.io/en/latest/Quantum-Dynamics-Modules/modules/PIM_qcf/readme.html
Quantics OpenMP Improvements Module . . . https://gitlab.e-cam2020.eu/e-cam/E-CAM-Library/merge_requests/23
PaPIM code . . . https://gitlab.e-cam2020.eu/Quantum-Dynamics/PIM/tree/deliverables
CH5+ molecule with a classical sampling method . . . https://gitlab.e-cam2020.eu/Quantum-Dynamics/PIM/tree/deliverables/tests/ch5/CLASSICAL

Page 15
JUQUEEN . . . http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/Configuration/Configuration_node.html

Page 16
"Pyrazine, 24 modes" . . . http://homepage.univie.ac.at/mario.barbatti/papers/pyrazine_pyrimidine/

raab_jcp_110_936_1999.pdf

Page 17

eCSE funding . . . http://www.archer.ac.uk/community/eCSE/

Page 18

Polarizable Soft Water Model . . . https://www.e-cam2020.eu/pilot-project-unilever/
Rheological Properties of New Composite Materials . . . https://www.e-cam2020.eu/pilot-project-michelin/
Piz Daint . . . https://www.cscs.ch/computers/piz-daint/
GPU Programming with CUDA . . . https://events.prace-ri.eu/event/676/
Merge Request . . . https://gitlab.e-cam2020.eu/e-cam/Meso-Multi-Scale-Modelling-Modules/merge_requests/20

Page 19

ESPResSo++ can now be installed through EasyBuild . . . https://github.com/easybuilders/JSC/pull/3
JUBE file . . . https://gitlab.e-cam2020.eu/castagna/E-CAM-Library/blob/module_JUBE_Espressopp/Meso-Multi-Scale-Modelling-Modules/modules/JUBE/JUBE_espressopp.xml
JURECA . . . http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JURECA/JURECA_node.html
Marconi . . . http://www.cineca.it/en/content/marconi

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. Mendoncca and A. O. Cais, "ESDW guidelines and programme III," Feb. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1207531

[3] L. Liang, A. O'Cais, J. Castagna, S. Wong, and G. Sanchez, "Hardware developments II," Oct. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.1207613

[4] C. Dellago and D. Swenson, "Identification/selection of E-CAM MD codes for development," Nov. 2016. [Online]. Available: https://doi.org/10.5281/zenodo.841694

[5] F. C. Liang Liang, Micael Oliveira and Y. Pouillon, "Electronic structure E-CAM modules I," Sep. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.1185170

[6] V. Vitale and L. Liang, "Electronic structure E-CAM modules II," Nov. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.1207633