E-CAM Software Porting and Benchmarking Data I

E-CAM Deliverable 7.2
Deliverable Type: Report
Delivered in Month 15– December 2016

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 PELEGRIN
Deliverable ID	D7.2
Deliverable Nature	Report
Dissemination Level	Public
Contractual Date of Delivery	Project Month 15(December 2016)
Resubmission Date	04.09.2017
Description of Deliverable	Joint technical report on results of porting and optimisation of at least 8 new modules out of those developed in the ESDWs to massively parallel machine (STFC); and benchmarking and scaling of at least 8 new modules out of those developed in the ESDWs on a variety of architectures (Juelich).

Document Control Information

Document	Title:	E-CAM Software Porting and Benchmarking Data I
	ID:	D7.2
	Version:	As of September 4, 2017
	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(DE-JUELICH)
	Contributors:	Liang Liang (CNRS), Jony Castagna (STFC)
	Reviewed by:	Godehard Sutmann (DE-JUELICH)
	Approved by:	Godehard Sutmann (DE-JUELICH)

Document Keywords

Keywords:	E-CAM, HPC, CECAM, Materials, ...

September 4, 2017

***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

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.

We have done this for 1 module related to WP2, 5 modules from WP3 and 2 modules from WP4:

- for Electronic Structure: Libescdf,
- for Quantum Dynamics: PotMod, AuxMod, ClassMC, SodLib, ChebLib,
- for Meso- and Multi-scale Modelling: first GPU version of DL_MESO_DPD and the addition of bond forces to the GPU version of DL_MESO_DPD.

In addition, we have looked at the scalability behaviour of a further 5 applications. These applications were considered because of their potential for future cross-WP collaboration activities or their relevance to particular future E-CAM activities. Where possible, inputs relevant to E-CAM were used for these applications (however, it should be noted that specific use cases restrict us to strong-scaling analysis and therefore we also use cases that allow us to investigate weak scalability).

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

- for Classical Molecular Dynamics: GROMACS and LAMMPS,
- for Electronic Structure: Libescdf, Quantum ESPRESSO and CP2K,
- for Quantum Dynamics: PIM and Quantics,
- for Meso- and Multi-scale Modelling: Ludwig and DL_MESO_DPD.

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 clusters systems (with Xeon Phi or GPU accelerators on various systems). These architectures cover the full spectrum of current production hardware available within PRACE.

A number of performance related issues arising from the E-CAM modules have been found. These issues include scalability problems and, in the IO case of Libescdf, implementation problems. These have been raised with the developers and have been targetted for resolution in time for the second deliverable in this series.

This deliverable also prescribes a workflow to ensure that, going forward, the porting effort is efficiently integrated into the ESDW events and effectively communicated to the end-user community. This workflow includes

- creating reproducible and efficient software builds using EasyBuild,
- a benchmarking workflow using JUBE,
- application optimisation with Scalasca.

Elements of this workflow are already integrated into the current document. In particular, a large portion of the *Porting* section is dedicated to correctly incorporating the applications into the EasyBuild framework (and making appropriate changes to the EasyBuild application where necessary).

1 Introduction

The original purpose of the current deliverable was to deliver a joint technical report on results of porting and optimisation of at least 8 new modules out of those developed in the Extended Software Development Workshop (ESDW) events to massively parallel machines and the benchmarking and scaling of at least 8 new modules out of those developed in the ESDW events on a variety of architectures.

A number of relevant applications (with relevant use cases) are addressed in addition to the required 8 E-CAM modules.

All of the applications are 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). 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).

1.1 Distribution of modules and workflow

The initial set of ESDW events did not occur until month 9, with the first wrap-up event for an ESDW not taking place until month 15. For this reason, the E-CAM modules referenced here from ESDW events are skewed in favour of the initial events of WP2 and WP3 (which were the ones available in adequate time for the porting effort). WP4 is also represented due to the GPU porting efforts of one of the programmers for the DL_MESO_DPD application.

Only WP1 is unrepresented in terms of the associated E-CAM modules, this is mainly because the target application of that WP is OpenPathSampling (OPS), which is a Python application that wraps MD drivers. The incorporation of new drivers into OPS was not scheduled until the second ESDW and so no performance relevant modules were available.

We also target here applications that are expected to form a nexus of collaboration between various WPs. We attribute then to the WP we consider most relevant.

Both the Software Manager and E-CAM programmers were hired only in the second half of the first year. The workflow that has been created by them, which makes the ESDW module development part of the performance optimisation cycle (described in Section 2), has only been conceived since that time. It has, therefore, not been applied to the first set of ESDW events. As a result this workflow is only reflected in part in the results shown in the current deliverable.

2 Workflow

In this section we describe the workflow of the programming team which consists of the Software Manager (at partner JSC) and the two programmers hired at partners STFC and 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.

As mentioned in Section 1, this workflow was conceived only after the initial ESDWs had taken place and have therefore only been applied in part to the modules related to these ESDWs.

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 it's 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 and publish 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. This can be easily modified for use on other supported architectures.

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.

2.1.3 Optimisation - Scalasca

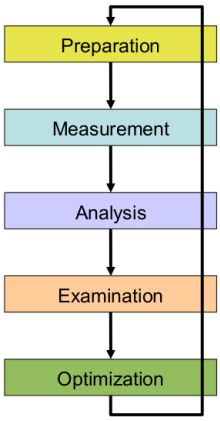

Figure 1: A Performance Optimisation Loop

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.

In addition to the main build using EasyBuild, we will provide an additional build recipe that provides an instrumented version of the software application using SCORE-P. This instrumented executable can then be utilised together with Scalasca to close a performance optimisation loop (see Fig. 1). End users can recycle this build recipe to create an instrumented version of the application with any modifications they may have implemented. EasyBuild does not currently have the technical capability to do this. This has been discussed with the EasyBuild developers and E-CAM will begin the process of integrating this functionality during the next EasyBuild User Meeting.

No example cases of Scalasca usage are presented in the current report, but will form a core component of future deliverables.

2.2 Interplay with ESDWs

As described in the ESDW guidelines (updated in Deliverable D5.2)[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).

[2] DOI: 10.5281/zenodo.841731

3 Porting

This section covers the hardware resources available for Work Package (WP) 7 and some specifics of the porting effort required on these architectures.

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

3.1 Available Resources

3.1.1 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 relevant time period for this deliverable coincided with a transitional phase in PRACE and resources were only continuously available on one cluster type system, i.e., MareNostrum3 (at Barcelona Supercomputing Centre), which also has Xeon Phi accelerators.

The second avenue is the normal PRACE Preparatory Access Call process. E-CAM has been successful twice in acquiring additional resources through this process, making an additonal 1.1M core hours available to the project.

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

- MareNostrum3 (Cluster with Xeon Phi accelerators, Spain)
- Hazel Hen (Cray XC40, Germany)
- JUQUEEN (BG/Q, Germany)
- Marconi (Cluster with Xeon Phi accelerators, Italy)
- SuperMUC (Cluster, Germany)

The only current-generation production HPC hardware missing from this collection are GPU accelerators, which are provided by project partners in the resources of Subsection 3.1.2.

Since MareNostrum3 was repeatedly used in the applications included in this document (due to the fact that they were the only PRACE resources available during the entire period of preparation of this deliverable), we will explicitly mention that this supercomputer has 3,056 nodes of 2× Intel SandyBridge-EP E5-2670/1600 with 42 of them having 2× Intel Xeon Phi 5110P coprocessors.

3.1.2 Other Resources

A number of HPC sites are project partners and have generously made additional resources available, particularly in the case where a particular HPC architecture component was not already available to the project.

- JURECA (cluster with GPU accelerators, through partner FZJ-JSC)
- Hartree Centre[3] (through partner STFC)
- Poincare (cluster, through partner CNRS)

3.2 Porting Effort

As previously mentioned, only MareNostrum3 was available over the entire benchmarking period and it is on this PRACE architecture that the majority of our results were obtained. Results for all other PRACE architectures are included, though only for a subset of cases. The additional resources outlined in section 3.1.2 are considered testing grounds and are typically not included unless they bring an architecture that is not included in PRACE systems, such as the GPUs.

[3] The Hartree Center consists of several supercomputers, but mainly based on IBM Blue Wonder (Iden) and IBM Power8 (Panther) architectures. Iden consists of 84 nodes of 2x Intel Ivy Bridge processors each + 42 Intel Xeon Phi 5110P coprocessors. The Panther machine is made of 32 Power 8335 nodes, each containing 2 x Power8 IBM processors + 4 Tesla K80 Nvidia cards.

The porting effort was mainly restricted to 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.

While Xeon Phi accelerators and GPUs were available, not all applications could take advantage of them. We include in the discussion the subset of those that could benefit from them.

3.2.1 Porting EasyBuild to PRACE Systems

A number of systems restrict network access from the platform, making installation procedures that automatically download sources non-functional. It was therefore necessary for E-CAM to further develop the installation procedure of EasyBuild so that it could be done completely offline. This was carried out in a *Pull Request*[4] to the main EasyBuild GitHub repository: Offline EasyBuild bootstrap.

In addition, system administrators on HPC platforms are rightfully conservative in their approach to OS updates, unfortunately this can mean that some system software can be extremely outdated as the platform approaches mid to end-of-life (particularly for things like the Operating System (OS) version of Python supplied). To overcome the restrictions this puts in place, E-CAM developed a script to bootstrap a modern Python2 release together with Lmod (a more modern and actively maintained implementation of the `module` environment management tool) and EasyBuild. This was done in the Pull Request "Bootstrap Python, EasyBuild and Lmod".

3.2.2 MPI Optimisation

The approach within EasyBuild is to control the entire software environment, taking as little as possible from the host system. As part of our initial efforts to leverage EasyBuild, we therefore built the entire software stack ourselves (starting with the compiler and the Message Passing Interface (MPI) implementation). Our use of our own MPI stack led to initial scaling problems with our build of some of the applications. We were able to correct for this by configuring MPI with the assistance of the hosting site (MareNostrum in this case).

Going forward, it is clear that any effort to build an MPI stack from source must be properly benchmarked and coordinated with the hosting site so as to ensure the stack is appropriately tuned. It is also clear that having the hosting site publish their configuration of the MPI stack is very valuable to those who would like to have this level of control.

An alternative that E-CAM created was to use EasyBuild to wrap the existing compiler and MPI installations into the EasyBuild environment. This is not considered ideal as the installation is no longer considered strictly reproducible with this approach. The implementation was carried out in two Pull Requests:

- Adding the intelligence to EasyBuild to recognise existing compilers and MPI implementations correctly (and verify them): "Add more complete system compiler and system MPI support to EasyBuild"
- Add a flexible software stack that can wrap existing GCC and OpenMPI installations (a specific instance of the capability introduced in the previous Pull Request): "OpenMPI vsystem, GCC vsystem"

3.2.3 Porting QuantumESPRESSO to EasyBuild

While initial support for QuantumESPRESSO already existed in EasyBuild, the implementation was not complete and a number of Pull Requests were created to improve this:

- Generic support for a multi-threaded Fast Fourier Transform (FFT) library in EasyBuild itself: "Multi-threaded FFT"
- Making the existing QuantumESPRESSO implementation aware of multi-threaded FFT: "Make QE easyblock aware of multithreaded FFT"
- Correct handling of some of the `gipaw` module: "Handle gipaw correctly in QE"
- Support for the QuantumESPRESSO v6.0 within EasyBuild: "QuantumESPRESSO v6.0"

[4] Pull requests let you tell others about changes you've pushed to a GitHub repository. Once a pull request is sent, interested parties can review the set of changes, discuss potential modifications, and even push follow-up commits if necessary.

3.2.4 Porting CP2K to EasyBuild

Again, initial support for CP2K already existed in EasyBuild, E-CAM implemented some tweaks to the installations already available there through 2 Pull Requests.

- Generic support for a hybrid build of CP2K with an open source toolchain: "CP2K add psmp build support for foss"
- Support for CP2K 4.1 with an open source toolchain: "CP2K v4.1"

3.2.5 Porting DL_MESO to the GPU

In order to accelerate the DL_MESO_DPD code on the latest and future exascale hardware, a first version for NVidia GPU has been developed. This is only a starting point, it does *not* cover all the possible cases and it does *not* yet support multiple GPUs. However, it represents an HPC milestone for the application, complementing the already present parallel versions developed for shared and distributed memory (MPI/OpenMP).

In this version, the full computational workload is offloaded to the GPUs with the `H_MAINLOOP` call present in the `dlmesodpd.f90` file. In this way, the initialisation IO operation are left unaltered and fully compatible with the serial version. A major change compared to the serial version is in the algorithm used to find the particle-particle interaction forces: in order to guarantee better coalescent access for the CUDA-threads, the algorithm has been re-adapted to the GPU architecture reordering the cell-linked list arrays.

Further details can be found in the Merge Request to the WP4 E-CAM repository: DL_MESO GPU implementation.

4 Modules and Application Codes

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

- Relevance to E-CAM.
- 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 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 rare events. In E-CAM Deliverable D1.1[5], we provided an overview of existing software for rare events, and found that the software package OpenPathSampling (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

Open Path Sampling (OPS) was the subject of the first ESDW in Classical Molecular Dynamics.The ESDW was 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 we instead benchmark two applications that are targets for inclusion as MD engines of OPS (scheduled for the second WP1 ESDW in August 2017): LAMMPS and GROMACS. 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 has a clear impact on the scalability of OPS itself.

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 these benchmarks is the 17 Nov 2016 (LAMMPS release versions corresponds to the actual day of release).

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.

4.1.2 Benchmarks and Scaling

For the LAMMPS code, the following two tests cases have been used for benchmarking:

- A mixture of water and surfactants for sustainable development in oil, gas, petrochemical and chemical process industries has been chosen for benchmarking the code LAMMPS. The system contains around 83000 coarse beads of 4 different constituents. Some of the beads are connected by harmonic bond and the degree of coarse graining and the density of the system are determined using the model proposed by Warren and Groot [6].
- A Lennard-Jones fluid using ~2M particles.

[5] DOI: 10.5281/zenodo.841694

[6] The Journal of Chemical Physics 107, 4423, 1997

For the code GROMACS, the benchmark case used here is the Lysozyme case provided in the tutorial (see Gromacs Lysozyme). It consists of ~12000 particles: a protein surrounded by water, which is representative of most real applications.

LAMMPS

Figure 2 shows the strong scaling results on the water and surfactants mixture run on the JURECA supercomputer. The results indicates that good scaling is achieved up to 192 cores, with an efficiency within 70%. Since we are dealing with a specific use case, the scalability is limited to strong-scaling which in turn is limited by the number of particles in the system.

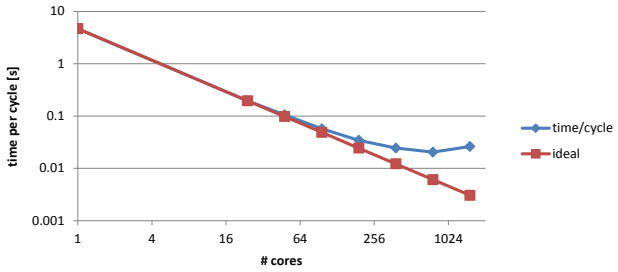

Figure 2: LAMMPS code. Strong scaling for the water and surfactants mixture .

Figure 3 shows the strong scaling results obtained on MareNostrum3 for the Lennard-Jones fluid. The code scales well up to 512 cores (efficiency within 94%) and then it quickly drops to 60% efficiency (larger test cases may have better strong scaling results), i.e. scalability saturates from ~4000 particles per core.. Good scalability is also observed from the weak scaling results (Figure 4) where a maximum of 147M particles have been used (36000 particles per core), the dip at 2048 cores is unexplained and we did not have the run statistics to do further analysis.

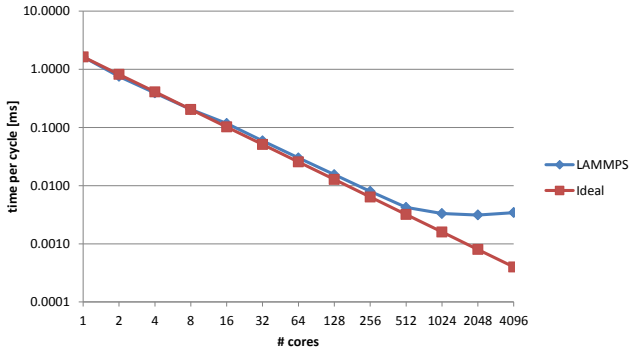

Figure 3: LAMMPS code. Strong scaling without I/O.

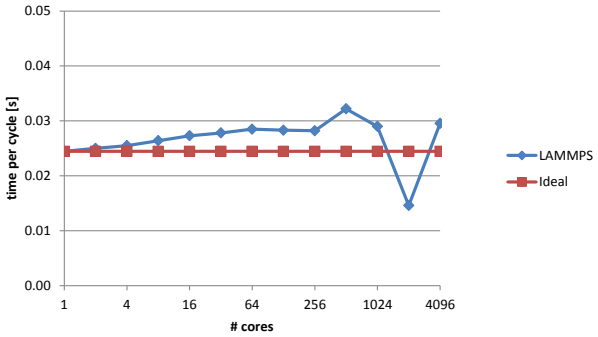

Figure 4: LAMMPS code. Weak scaling without I/O.

Figure 5 shows the strong scaling with the same inputs on JUQUEEN (BG/Q, Germany). The code is shown to scale extremely well up to 32768 cores and is already a member of the High-Q Club (codes that scale to the entire machine).

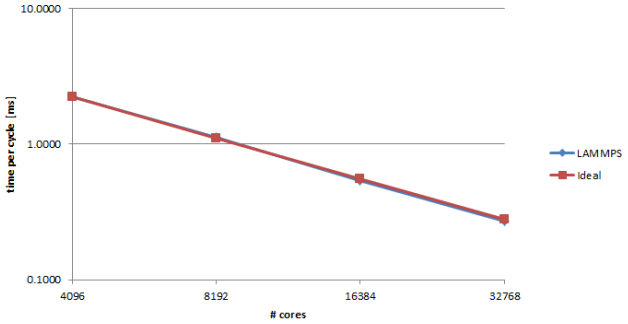

Figure 5: LAMMPS code on JUQUEEN. Strong scaling without I/O.

GROMACS

GROMACS takes advantage of combined distributed and shared memory parallelism using different combinations of libraries like MPI, OpenMP, thread-MPI, etc. In these tests, hybrid MPI/OpenMP solutions have been tested, using the GROMACS recommendations provided in their manual. GROMACS has the additional capability of making suggestions on what the best combinations of MPI/OpenMP are based on the tests run (these appear in the output logs), these suggestions were utilised in the scaling tests. A presentation on the runtime optimisaton of GROMACS is available from the BioExcel CoE.

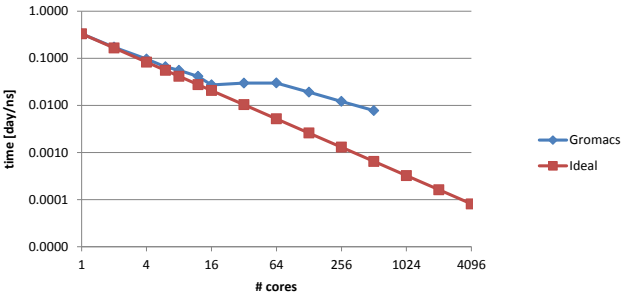

Figure 6: GROMACS code strong scaling - MareNostrum3.

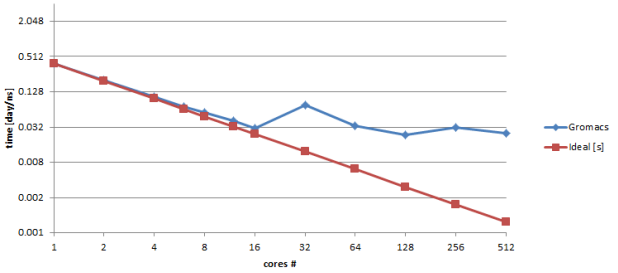

Figure 7: GROMACS code strong scaling - Marconi.

Figure 6 shows the strong scaling results (without I/O) on MareNostrum3 and Figure 7 shows the same results for Marconi. The effect on performance moving from intra-node (up to 16 cores, where OpenMP thread parallelism is used) to across nodes (where MPI processes are used) is very clear.

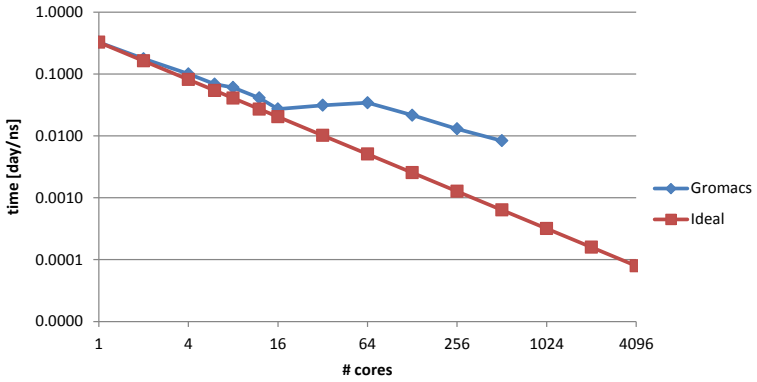

Figure 8: GROMACS code. Strong scaling with I/O

In Figure 8 we can see that the inclusion of I/O does not affect the performance of GROMACS. The I/O frequency for this case has been set to every 100 time steps.

Finally, Figure 9 shows the results for the same test case running on GPUs (Tesla K80 from the Hartree Centre). The code has been tested with up to 16 GPUs and shows very good relative efficiency given the hardware (within 60% of ideal).

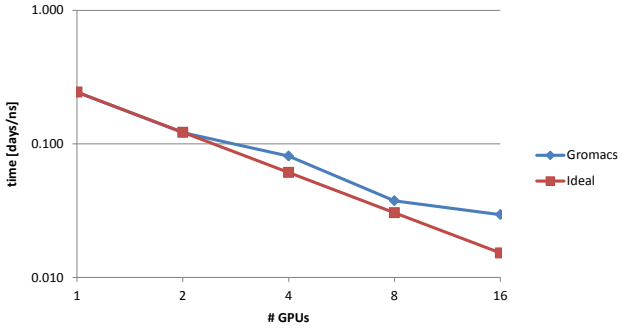

Figure 9: GROMACS code. Strong scaling without I/O on Tesla K80 GPUs

4.2 Electronic Structure

For WP2 we focus on three applications. The first is Libescdf, a library created to exchange electronic-structure-related data in a platform-independent and efficient manner. It is based on the Electronic Structure Common Data Format Specifications, as well as HDF5, and was finalised at the first E-CAM ESDW, We have also benchmarked two community application codes: Quantum ESPRESSO and CP2K.

4.2.1 Relevance for E-CAM

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 focus is more on extracting useful utilities from these applications so that they can be leveraged by a wider range of applications.

In this first porting and benchmarking deliverable we focus on the IO performance of the Libescdf library that code was finalised at the first WP2 ESDW in Zaragoza in June 2016. This library is an implementation referenced in a Psi-K white paper entitled "Towards a Common Format for Computational Materials Science Data". The work relates to a single E-CAM module:

- E-CAM module - Libescdf module.

We also note that another module from the same workshop (LibOMM) is the subject of a paper ([1]) where it's scaling behaviour to 9600 MPI tasks is shown.

We choose Quantum ESPRESSO as another application because it was targeted during the same ESDW for the extraction of some of the utilities used there (and scheduled for implementation in 2017). In particular there is interest in extracting the two basic Kohn-Sham solvers used in the Quantum ESPRESSO codes, which require parallel linear algebra routines, if parallel execution is desired, as well as parallel FFT routines. Initial discussion on how to implement this extraction was carried out with Quantum ESPRESSO developers during the first ESDW.

In addition to the two required applications above, we also include CP2K which is a collaboration point between WP2 and WP3. CP2K is already implemented as a library and is targeted as the workhorse when more complex forces are included in PIM (see Section 4.3).

4.2.2 Benchmarks and scaling

Libescdf

Libescdf is a library containing tools for reading and writing massive data structures related to electronic structure calculations, following the standards defined in the Electronic Structure Common Data Format. It is a library created to exchange electronic-structure-related data in a platform-independent and efficient manner. It is based on the Electronic Structure Common Data Format Specifications, and is built upon HDF5.

We used Darshan 3.1.3 to evaluate the I/O performance of Libescdf on the JURECA system at JSC. The particular example used is the one included in the source code.

NCPU	Bandwidth Achieved	Read/Write Ratio
96	7343 MiB/s	20
192	11989 MiB/s	39.2
384	22553 MiB/s	77.6

Table 1: I/O performance of the Libescdf library

From Table 1, we can see that the achieved bandwidth is scaling roughly linearly with the number of nodes (and is expected to tail off as we approach the maximum bandwidth of the underlying filesystem). We halted the experiment at 384 cores however as we noticed that the ratio of reads to writes from the library was scaling with the number of nodes (which was made obvious from the poor wall-time scaling of the runs). One would expect the complete amount of writing/reading of the example to remain constant regardless of the core count. This is true of the writing but the amount of file reading is scaling with the core count. It would appear that there is a problem in the implementation of the interface with HDF5 which is causing this issue.

This issue is also apparent when we look at the time spent by MPI I/O function for READ and WRITE, shown in Fig.10 compared to total time spent on I/O (the blue+violet lines). The real average time spent by MPI I/O functions rep-

resent only a very small portion of the overall file access times. The imbalance of read/write is very clear from this picture.

An issue for these performance problems has been raised with the Libescdf developers and performance analysis with Scalasca is ongoing.

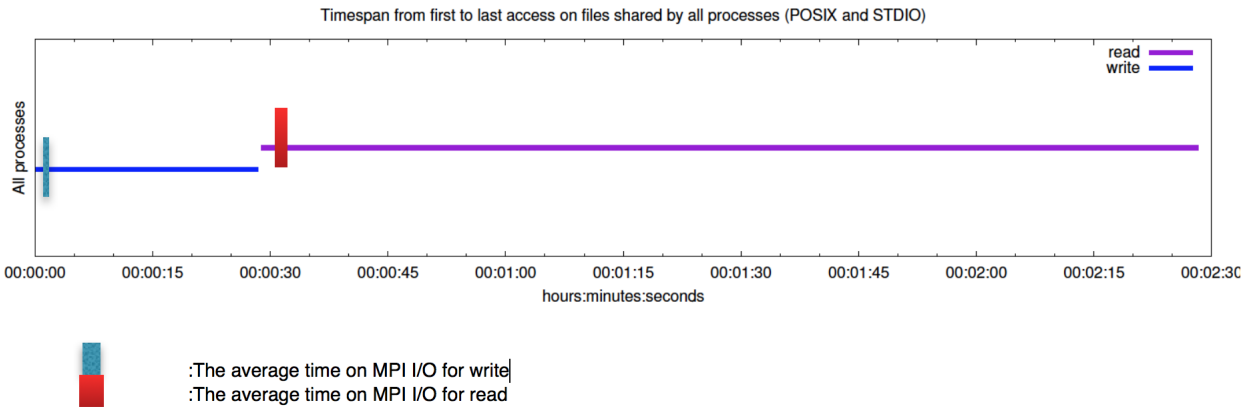

Figure 10: Timespan of code running with 384 cores, comparison with real average MPI I/O time

Quantum ESPRESSO code

Quantum ESPRESSO is an integrated suite of open-source computer codes for electronic-structure calculations and materials modeling at the nanoscale. It is based on density-functional theory, plane waves, and pseudopotentials. We used the 6.0 version released October 04, 2016 which can be downloaded from the Quantum ESPRESSO website.

EasyBuild is used to install a hybrid version (MPI+OpenMP) of Quantum ESPRESSO by using its foss toolchain (GCC 5.4.0, OpenMPI 1.10.3, OpenBLAS 0.2.18, LAPACK 3.6.1, ScaLAPACK 2.0.2, FFTW 3.3.4,). In Quantum ESPRESSO, several MPI parallelization levels are implemented, in which both calculations and data structures are distributed across processors. Processors are organized in a hierarchy of groups. To control the number of processors in each group, command line switches (`-nimage, -npools, -nband, -ntg, -ndiag`) are used. The Kohn-Sham orbitals (also called bands) parallelization `-nbands` is set to 1 since its implementation is still experimental and sometimes gives even slightly worse scalability in our small scale tests. The `-ntg` is set for good parallelization of the 3D FFT across processors. Within a self consistent iteration, the calculation of K points is separated into several groups `-nk`. Different `-nk` values have a large influence on code performance, the same inputs are thus tested with different `-nk` values. The number of OpenMP threads is set to 1 or 2 for this hybrid version.

Two benchmark cases are used. Inputs can be downloaded from small-scale test 2 with k-points setting 333 instead of 222 and large-scale DEISA benchmark from the Quantum ESPRESSO website.

The small scale input is a super-cell composed of 64 atoms of Fe with 384 electrons and 8 irreducible k-points. Convergence is set to `conv_thr=1.0D-8`.

The large scale input is a Thiol-covered gold surface and water, 4 k-points, 587 atoms, 2552 electrons. Convergence is set to `conv_thr=1.0D-8`.

Figure 11 shows then strong scaling results on MareNostrum3 with `NUM_THREAD_OMP=1`. It can be clearly seen that the effect of the `-nk` is very important. When set to 8, it doubles the performance and scaled to 512 cores compared to the results obtained by setting `nk=1`. We can also see that, by increasing the number of cores by a factor of 32, the speed up is only a factor of 5. The scalability for small scale inputs is thus relatively poor.

Figure 12 shows then strong scaling results on MareNostrum3 with `NUM_THREAD_OMP=1`. `-nk` is set to 4. The code scales well compared to the benchmark line given by CRAY-XT4. `NUM_THREAD_OMP=2` slightly improves the performance when using 128 and 256 cores.

In Figure 13, we can see that the use of more than 1024 cores with the Quantum ESPRESSO code (for this case) either didn't work (in the case of the EasyBuild OpenMPI installation at 4096 cores) or scaled poorly on SuperMUC (Cluster, Germany). A possible further optimisation of the OpenMPI installation may be needed when using the EasyBuild install procedure on the MareNostrum architecture (or one may resort to the solution described in Section 3.2.2).

A comparison between the scaling tests done with Quantum ESPRESSO installed by EasyBuild and the one installed with standard procedure is also shown in Fig. 14. The standard installation version uses intel 13.0.1, MKL 11.0.1, openmpi 1.8.1 and ScaLAPACK 2.0.2. The relative performance levels are close up to 512 cores, while the EasyBuild

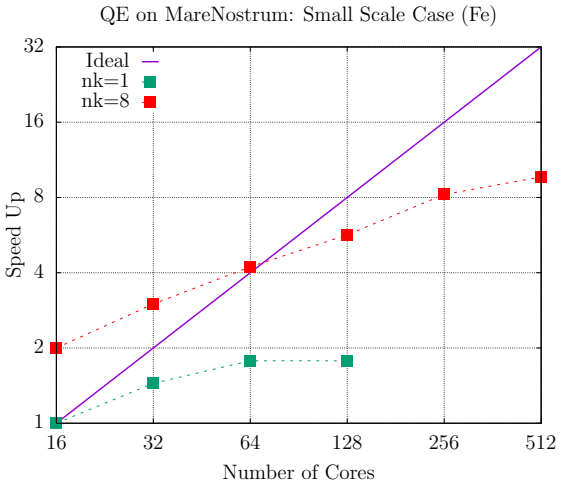

Figure 11: Quantum ESPRESSO code. Strong scaling test with small problem size. Speed-up of `nk=8` is relative to `nk=1`

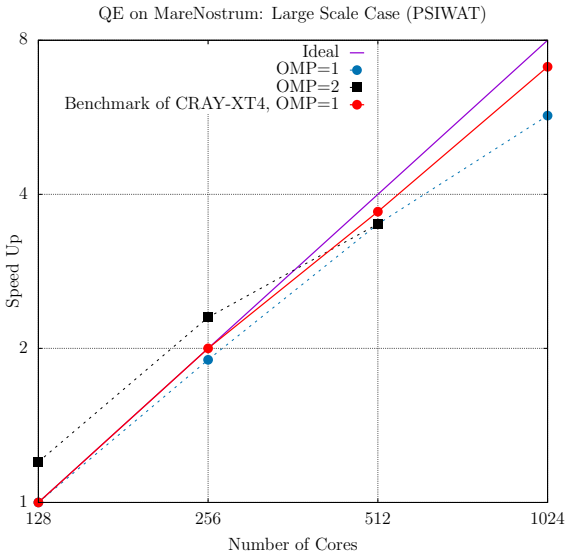

Figure 12: Quantum ESPRESSO code. Strong scaling test with large problem size.

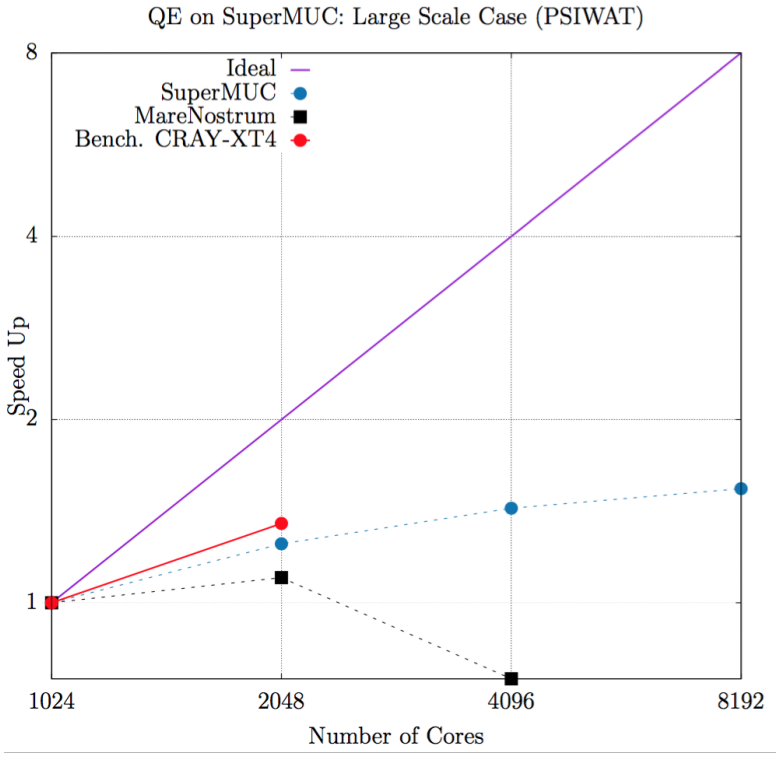

Figure 13: Quantum ESPRESSO code. Strong scaling test with large problem size and large node count (on MareNostrum3 and SuperMUC).

version had much better scalability for the test when using `Ncore=1024`. In the EasyBuild case, we are using a different compiler, optimisations, MPI version (1.10.4) and math libraries; and the Quantum Espresso dependencies are tuned externally to the package itself.

CP2K code

CP2K is a quantum chemistry and solid state physics software package that can perform atomistic simulations of solid state, liquid, molecular, material, crystal, and biological systems under periodic boundary conditions. CP2K provides a general framework for different modeling methods such as Density Functional Theory (DFT) using the mixed Gaussian and plane waves approaches GPW and GAPW. CP2K can do simulations of molecular dynamics, metadynamics, Monte Carlo, Ehrenfest dynamics, vibrational analysis, core level spectroscopy, energy minimization, and transition state optimization using Nudged Elastic Band (NEB) or dimer method. We used its 4.1 version released

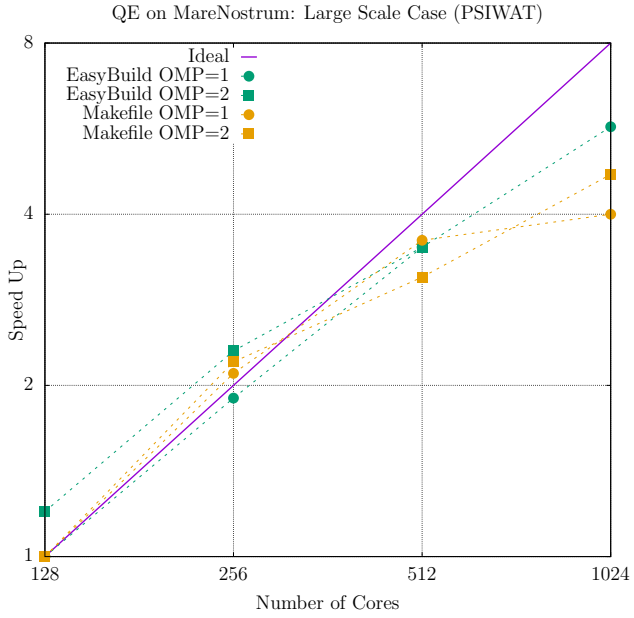

Figure 14: Quantum ESPRESSO code. Comparison between Easybuild installed version and Standard version.

October 05, 2016 which can be downloaded from the SourceForge website.

EasyBuild is used to install a hybrid version (MPI/OpenMP) of CP2K, again using its foss toolchain. The entire CP2K code is MPI parallelized. Some additional loops are also OpenMP parallelized. As advised on the CP2K website, we first take advantage of the MPI parallelization. However, given that running one MPI-rank per CPU-core will probably lead to memory shortages, the usage of OpenMP is sometimes needed, particularly for large structures and larger numbers of cores.

Two benchmark cases are used. Inputs can be downloaded from CP2K Benchmark Suite Section. The small scale and standard DFT case named "H2O-64" is an ab-initio molecular dynamics of liquid water using Quickstep DFT. Local Density Approximation (LDA) is used. The system contains 64 water molecules (192 atoms, 512 electrons) in a 12.4 $Å^3$ cell. The large scale and linear-scaling DFT case named "H2O-DFT-LS" is a single-point energy calculation. It consists of 6144 atoms in a 39 $Å^3$ box (2048 water molecules in total). The linear scaling cost results from the fact that the algorithm is based on an iteration on the density matrix. The cubically-scaling orthogonalisation step of standard Quickstep DFT is avoided when using the orbital transformation method.

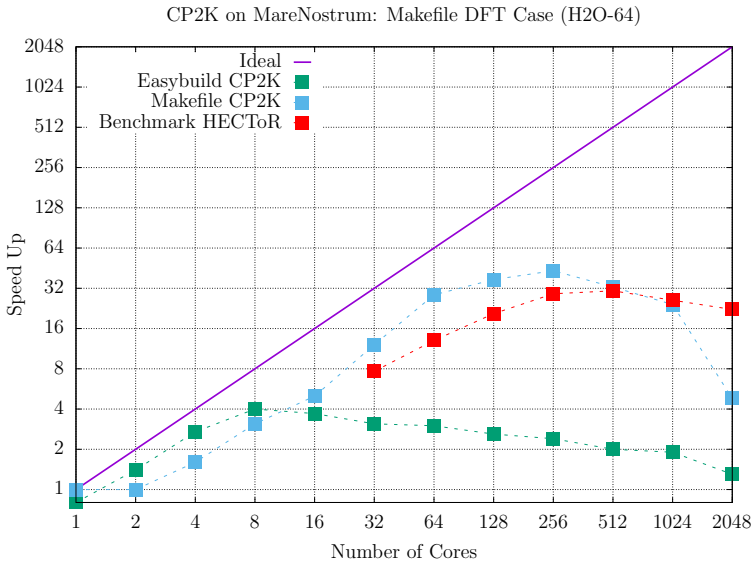

Figure 15: CP2K code. Strong scaling test.

Figure 15 shows then strong scaling results on MareNostrum3 with `NUM_THREAD_OMP=1`. It can be clearly seen that the version installed with the packaged Makefile gives a similar performance to the CP2K benchmark on HECToR

machine. Beyond 16 cores (1 node), the CP2K version installed by EasyBuild scales poorly. This may be related to a possible optimisation problem of OpenMPI installed by EasyBuild that we discovered during the Quantum ESPRESSO tests.

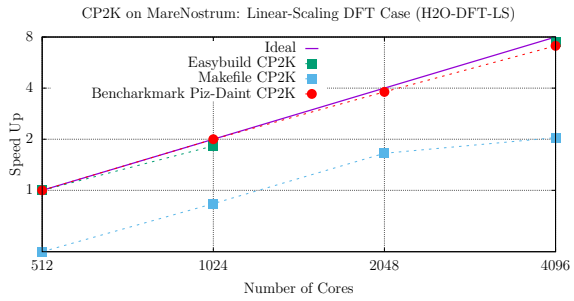

Figure 16: CP2K code. Strong scaling test (MareNostrum).

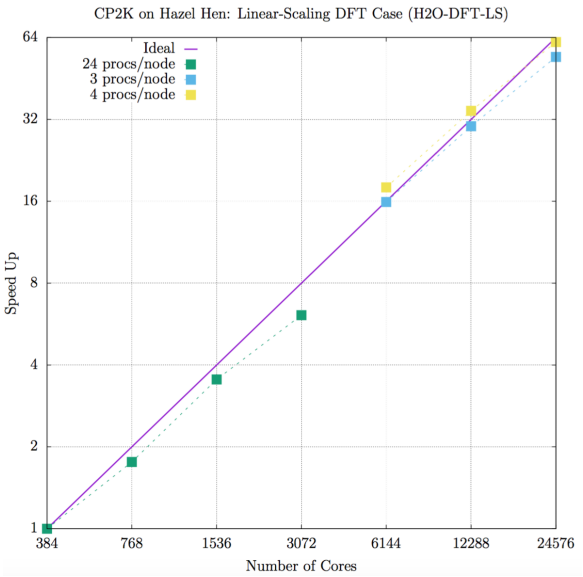

Figure 17: CP2K code. Strong scaling test (HazelHen).

Figure 16 shows then strong scaling results on MareNostrum3 with `NUM_THREAD_OMP=1` (except for the EasyBuild CP2K version running on 4096 cores where 8 threads were used in this case). The results show that the EasyBuild CP2K version exhibits excellent scalability for linear-scaling DFT calculation when an appropriate thread number is chosen. The EasyBuild CP2K version is two times faster than the standard version ("Makefile CP2K" line in blue) in term of linear-scaling DFT calculation. Higher core counts are shown for Hazel Hen (Cray XC40, Germany), a platform similar to the reference platform of the benchmark. Here we can see excellent scaling properties for CP2K, likely attributable to the predictability of the software environment of the platform, and the increased possibility, therefore, of targeted optimisation for the platform.

Since the tests on 4096 cores worked well with the EasyBuild installed OpenMPI version, the OpenMPI optimisation problem may be related to the cubically-scaling matrix orthogonalisation component of the benchmark.

4.3 Quantum Dynamics

Uniquely in the Quantum Dynamics (WP3) package, there are no well-established community codes. In the ESDW organized in June 2016, either new application codes or parallel implementations of existing codes were developed. We have tested the following codes that were targetted there: PIM and Quantics.

4.3.1 Relevance for E-CAM

Both of the applications addressed were subjects of the first ESDW and are likely to feature again in the second WP3 ESDW. The content included here are evaluations of the parallelisation efforts at the initial WP3 ESDW.

There are 5 E-CAM modules related to the work in this section, all of which have been described in detail in Deliverable 3.1.

For PIM, the relevant E-CAM modules are:

- E-CAM module - PotMod,
- E-CAM module - AuxMod,
- E-CAM module - ClassMC (particularly relevant).

For Quantics, the relevant modules are:

- E-CAM module - SodLib,
- E-CAM module - ChebLib.

4.3.2 Benchmarks and scaling

PIM code

The PIM code is an application which performs exact sampling of the Wigner density using the PIM method. It provides quantum initial conditions for the approximate calculation of the time correlation functions. The code is still under development.

The PIM code (Version 5 Dec, 2016) is installed with the EasyBuild free and open source software (foss) toolchain. The CH5+ molecule with classical sampling method is used as input. In the `MCINPUT` file, `NDump` is set to 2000, `NBlock` is set to the number of cores. `NTraj`, the number of molecular dynamics trajectories, is set to 5000.

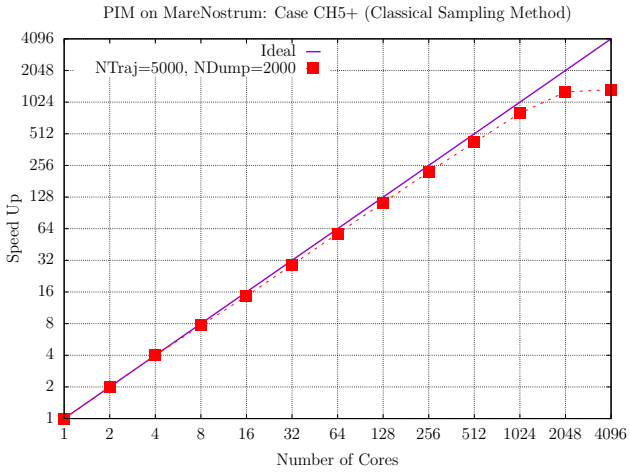

Figure 18: PIM code. Strong scaling test.

Figure 18 shows the strong scaling results on MareNostrum3 from 1 to 4096 cores. The results show that the PIM code scales linearly up to 1024 cores and then begins to tail off rapidly.

NCPU	Total CPU Time	Wall time
1024	1	1
2048	1.001949422	0.683671099
4096	0.9989863155	0.5302435568

Table 2: Scalability Results on JUQUEEN. CPU times (total time spent computing) and wall times have been normalised with respect to the value for 1024 CPUs

We see similar results on JUQUEEN in Table 2 with increasingly poor scalability as we move to 4096 cores. The output printed by PIM is done on a per CPU basis and in the 4096 core case runs to some 80K lines. This is excessive and reducing this is likely to have a large performance impact.

PIM is a target application in the second WP3 ESDW and further optimisation work will be carried out there (analysis with Scalasca, initial implementation of OpenMP parallelism).

Quantics code

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 package grew out of the Heidelberg MCTDH Package. Compared to the older MCTDH packages, the main changes in QUANTICS are the addition of the G-MCTDH algorithm and the direct dynamics DD-vMCG method. The code is now Fortran 90 based with full dynamical allocation of memory. Parallelisation using OpenMP and MPI is made in many parts of the code.

The Quantics code version is 1.1, shared with E-CAM by Professor Gramham Worth. The EasyBuild foss toolchain is used for the MPI version installation of Quantics. "Pyrazine, 24 modes" is used as input.

Figure 19 shows strong scaling results on MareNostrum3 from 1 to 16 cores. The results show that the QUANTICS code scales relatively poorly. The application of more than 16 cores turned out to be unsuccessful, as the code stopped after the first dynamic step. Indeed, the code has not yet been enabled by the developers to run on more than 16 cores and has never been tested on this number of cores before. Code development is still ongoing, rendering Quantics a good candidate for further optimisation.

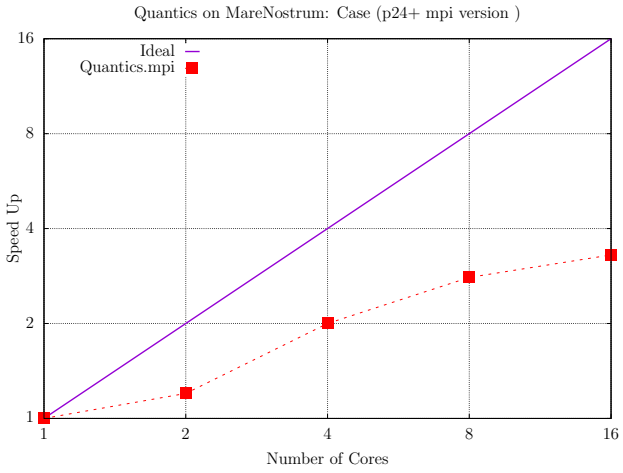

Figure 19: QUANTICS code. Strong scaling test.

4.4 Meso- and Multi-scale Modelling

In the Meso- and Multi-scale modelling (WP4) package the codes tested are: Ludwig and DL_MESO_DPD.

DL_MESO_DPD is a meso scale code based on the Dissipative Particle Dynamics method, an off-lattice, discrete particle method for modelling mesoscopic systems. It has little in common with Lattice Boltzmann methods, except in its application to systems of similar length and time scales. The code has been developed at Daresbury Laboratory and it can be downloaded at the DL_MESO project website. The version used is 2.6 (part of the DL_MESO v2.6 package).

Ludwig is a Lattice Boltzmann code used for the simulation of complex fluids. It is open source and available from the Ludwig svn repository. The version tested is 0.4.6.

4.4.1 Relevance for E-CAM

The first WP4 ESDW is scheduled for 2017, combining this with late hires of the relevant PDRAs leaves restricted scope for inclusion of modules generated by the WP.

DL_MESO_DPD is developed at one of the E-CAM partners, STFC, which also hosts the E-CAM programmer with expertise in GPU programming. It has been selected for porting to GPU accelerators. Accurate comparison of CPU and GPU targets requires prior knowledge of the scalability of DL_MESO_DPD. We include results here for both the current (at the time of writing) CPU implementation of DL_MESO_DPD (in `Fortran`) as well as some initial results for the results of the GPU porting effort (in `C++`). There are 2 E-CAM modules that are relevant to this work:

- E-CAM module - First GPU version of DL_MESO_DPD
- E-CAM module - Add bond forces to the GPU version of DL_MESO_DPD

The Barcelona team has been involved in the development of new modules within Ludwig. Specifically, it has generalized colloidal features and have included metabolic processes in a hybrid, continuum model which merges lattice Boltzmann with a continuum description of the microorganism populations. Ludwig is also considered a potential collaboration point between WP1 and WP4, with it's possible inclusion as an engine within OpenPathSampling.

4.4.2 Benchmarks and scaling

DL_MESO_DPD code

For the code DL_MESO the following two cases have been used:

- This system is a solution of 10% SDS (Sodium Dodecyl Sulfate) in water. SDS is an anionic surfactant and is used as an ingredient in many detergents. The system size is of the order of industrially relevant cases (~200k particles).
- Mixture of ~0.5M particles, not charged. It consists of 0.5M particles of two different species in a cubic box with periodic boundary conditions in all directions.

The source code is compiled using the Intel Compiler 13.0.1 and Intel impi 4.1.3.049 library. Moreover, for the chosen benchmark case the best performance has been obtained using MPI processes only, despite a hybrid MPI/OpenMP version being provided. The only flags used are: `-O3` and `-mmic` (for the Xeon Phi version).

Figure 20 shows the strong scaling on the watersurfactant mixture of 200k beads. The code scales efficiently (within %70) up to 1536 cores and then drops rapidly.

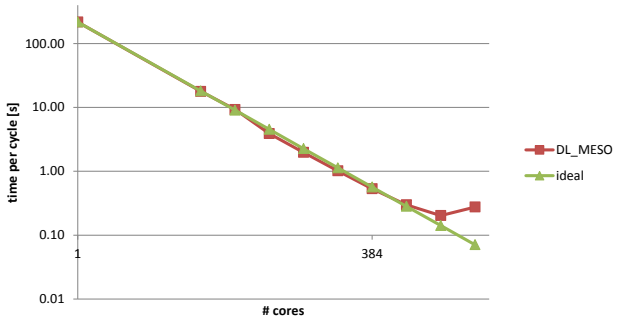

Figure 20: DL_MESO_DPD code. Strong scaling for the WaterSurfactant mixture

Figure 21 shows the strong scaling results obtained on MareNostrum3. The code scales well up to 256 cores (efficiency within 75%) and then it drastically loses performance, probably due to the increasing communication time. This is also confirmed from the weak scaling analysis where the time per cycle increases rapidly after 64 cores (see Figure 22).

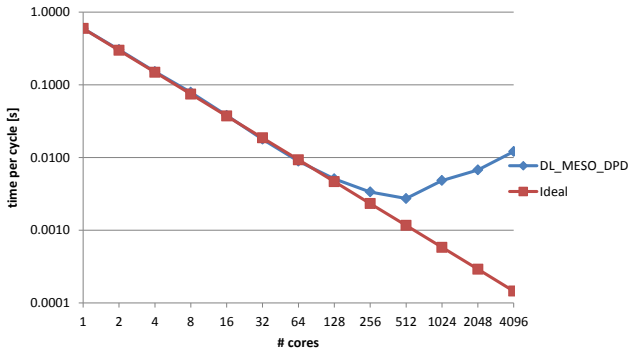

Figure 21: DL_MESO_DPD code. Strong scaling without I/O

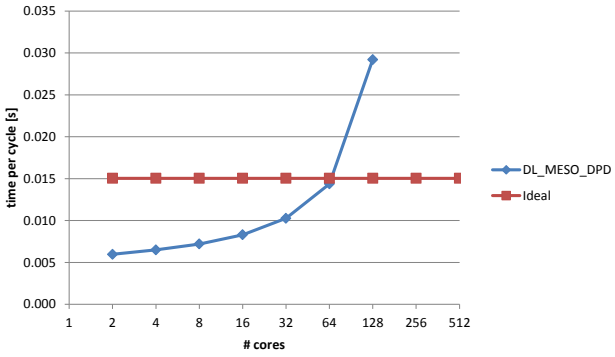

Figure 22: DL_MESO_DPD code. Weak scaling without I/O

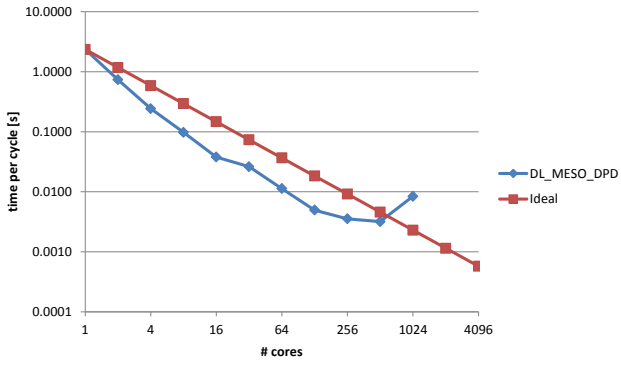

Figure 23: DL_MESO_DPD code. Strong scaling with I/O

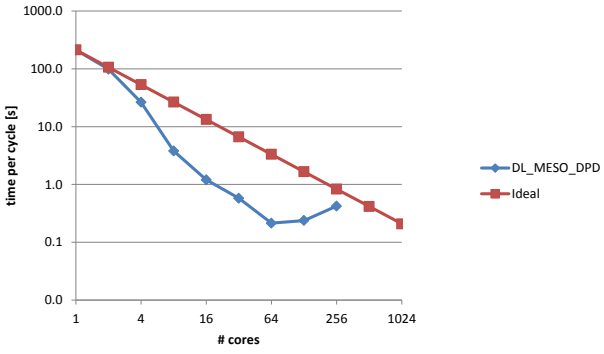

Figure 24: DL_MESO_DPD code. Strong scaling on the Xeon Phi (KNC) without I/O.

The effect of the I/O operation can be observed in Figure 23. The output frequency is every 10 time steps. The code scales well up to 512 cores in this case but the overall impact of the I/O operations is very significant, again showing the possible scope for I/O related optimisation (once possible explanations relating to network saturation are excluded).

In Figure 24 are shown the results on the Xeon Phi card in native mode. The performance is very slow compared to the Xeon processor, however strongly super linear, which indicates a very poor use of the Xeon Phi hardware.

Finally, the GPU version of the DL_MESO has been tested using the second input presented above. Below is a table about the performance on different GPU cards compared to the serial version on a single core:

CPU or GPU card	compute capability	time per cycle [s]	speedup
Intel Ivy Bridge E5-2697v2	none	0.4740	1.0
NVidia Tesla C1060	1.3	0.2280	2.1
NVidia Tesla C2075	2.0	0.1830	2.6
NVidia Tesla K40	3.5	0.1011	4.7
NVidia Tesla K80	3.7	0.0898	5.3
NVidia Tesla M60	5.2	0.0978	4.8
NVidia Tesla P100	6.0	0.0390	12.2

Preliminary results are encouraging, but still far from the expected performance (especially using the latest P100 card).

Ludwig code

The benchmark case chosen for the Ludwig code is a Poiseuille flow. For strong scaling tests we use a lattice consisting of 128^3 points, equally distributed along each direction. The results are shown in terms of time spent per cycle, which allows comparisons between them to be made. The source code is compiled using the Intel Compiler 13.0.1 and Intel impi 4.1.3.049 library. The only flags used are: `-O3`, `-DNDEBUG`, and `-mmic` (for the Xeon Phi version).

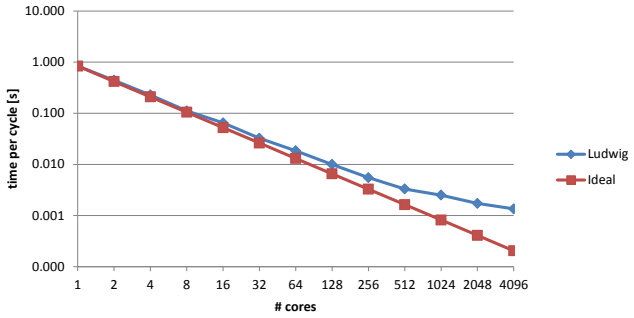

Figure 25: Ludwig code. Strong scaling without I/O.

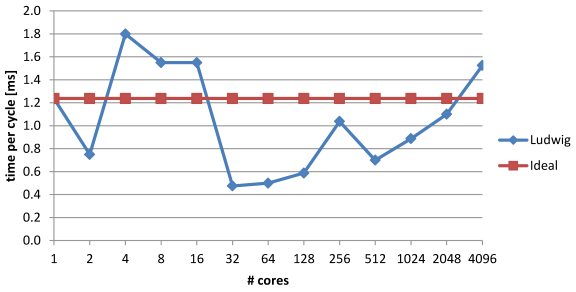

Figure 26: Ludwig code. Weak scaling without I/O.

Strong scaling results are presented in Figure 25. The code scales well (within 75%) up to 512 cores and within 60% up to 4096 cores. A slight variation in the scaling after 4 cores can be observed.

The Figure 26 shows the weak scaling results using a ratio $N_p/N_c = 8$, where N_p is the total size of the system and N_c is the number of cores. The plot has no clear trend, but overall it keeps below the ideal value obtained with 1 core only.

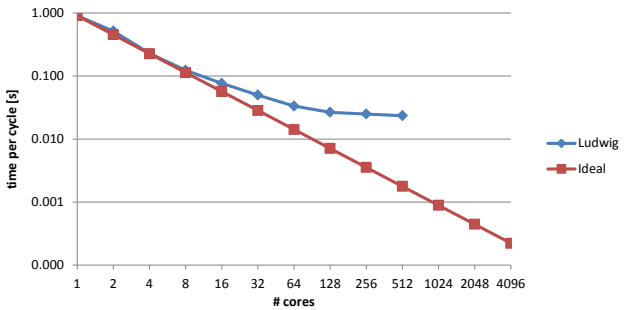

Figure 27: Ludwig code. Strong scaling with I/O.

The effect of the I/O operations have been tested using the same configuration of the strong scaling test, but dumping the output files at a frequency of every 100 time steps. Figure 27 shows that above 64 cores the efficiency drops to a value lower than 75% and quickly degenerates afterwords. This indicates that the effect of the I/O operation is very strong and becomes predominant very rapidly. This makes Ludwig a good candidate for an I/O optimisation effort.

Xeon Phi 0	Xeon Phi 1	Total Cores
1	0	1
2	0	2
4	0	4
8	0	8
16	0	16
32	0	32
32	32	64
64	64	128
128	128	256

Table 3: Ludwig code. Split of the logical cores between the Xeon Phi cards on the same node.

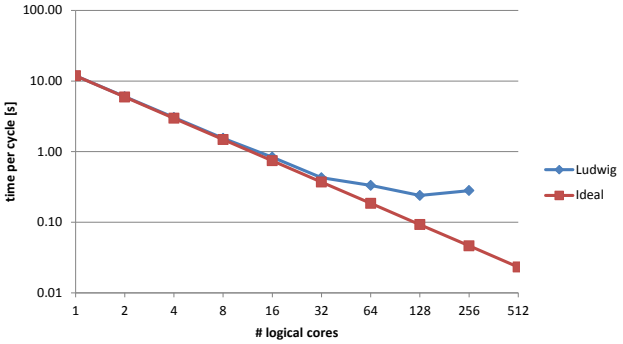

Figure 28: Ludwig code. Strong scaling on Xeon Phi (KNC) without I/O

Finally, Figure 28 presents the results running the code on the Xeon Phi coprocessor (KNC) in native mode. Each node of Mare Nostrum contains 2 Xeon Phi card (mic-0 and mic-1). The split of the logical cores used across the two cards is done according to the Table 3. The results show that the scaling is very efficient up to 32 logical cores and then quickly drops to 65%.

5 Outlook

Going forward WP 7 will focus more on the development efforts directly taking place with ESDW events using the workflow described in this document.

For the particular applications discussed in this deliverable, there are a number of performance related iddues to be addressed in the second set of ESDWs.

We would ultimately like to document the results included here on the E-CAM website. Prior to doing this we would like to engage the relevant developers to make them aware of our results and seek any further guidance to improve them (in a number of cases this has already happened).

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

GPW Gaussian Plane Waves

GAPW Gaussian augmented-Plane Waves

foss free and open source software

DFT Density Functional Theory

PIM Phase Integration Method

NEB Nudged Elastic Band

GPU Graphical Processing Unit

PDRA Post-doctoral Research Associate

MD Molecular Dynamics

FFT Fast Fourier Transform

OS Operating System

OPS Open Path Sampling

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/43
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

Libescdf ... http://e-cam-electronic-structure-modules.readthedocs.io/en/latest/modules/escdf/readme.html
PotMod ... http://e-cam-quantum-dynamics-modules.readthedocs.io/en/latest/modules/PotMod/readme.html
AuxMod ... http://e-cam-quantum-dynamics-modules.readthedocs.io/en/latest/modules/AuxMod/readme.html
ClassMC ... http://e-cam-quantum-dynamics-modules.readthedocs.io/en/latest/modules/ClassMC/readme.html
SodLib ... http://e-cam-quantum-dynamics-modules.readthedocs.io/en/latest/modules/SODLIB/sod_readme.html
ChebLib ... http://e-cam-quantum-dynamics-modules.readthedocs.io/en/latest/modules/cheb_doc/cheb_readme.html
first GPU version of DL_MESO_DPD ... https://gitlab.e-cam2020.eu/e-cam/Meso-Multi-Scale-Modelling-Module merge_requests/18/diffs
addition of bond forces to the GPU version of DL_MESO_DPD ... https://gitlab.e-cam2020.eu/e-cam/Meso-Multi-Scale-Modelling-Modules/merge_requests/19/diffs
GROMACS ... http://www.gromacs.org/

LAMMPS ... http://lammps.sandia.gov/
Libescdf ... https://gitlab.e-cam2020.eu/esl/escdf
Quantum ESPRESSO ... http://www.quantum-espresso.org
CP2K ... https://www.cp2k.org
PIM ... https://gitlab.e-cam2020.eu/Quantum-Dynamics/PIM/tree/deliverables
Quantics ... http://stchem.bham.ac.uk/~quantics/doc/index.html
Ludwig ... https://ccpforge.cse.rl.ac.uk/gf/project/ludwig/
DL_MESO_DPD ... http://www.scd.stfc.ac.uk/support/40694.aspx
Libescdf ... https://gitlab.e-cam2020.eu/esl/escdf
EasyBuild ... http://easybuild.readthedocs.org/en/latest/
JUBE ... https://apps.fz-juelich.de/jsc/jube/jube2/docu/index.html
Scalasca ... http://www.scalasca.org/

Page 2

EasyBuild ... http://easybuild.readthedocs.org/en/latest/
OpenPathSampling ... http://openpathsampling.org/latest/index.html

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

Scalasca ... http://www.scalasca.org/
SCORE-P ... http://www.vi-hps.org/projects/score-p/
next EasyBuild User Meeting ... https://github.com/hpcugent/easybuild/wiki/2nd-EasyBuild-User-Meeting
ESDW guidelines ... https://www.e-cam2020.eu/deliverables/

Page 5

PRACE Preparatory Access Call ... http://www.prace-ri.eu/prace-preparatory-access/
MareNostrum3 ... https://www.bsc.es/innovation-and-services/supercomputers-and-facilities/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/
JURECA ... http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JURECA/Configuration/Configuration_node.html;jsessionid=DFEDF689186F7E463728DBE6BF1BE02C
Hartree Centre ... http://www.hartree.stfc.ac.uk/hartree/
Poincare ... https://groupes.renater.fr/wiki/poincare/public/description_de_poincare

Page 6

Pull Request ... https://yangsu.github.io/pull-request-tutorial/
Offline EasyBuild bootstrap ... https://github.com/easybuilders/easybuild-framework/pull/1880
Lmod ... https://www.tacc.utexas.edu/research-development/tacc-projects/lmod
Bootstrap Python, EasyBuild and Lmod ... https://github.com/easybuilders/easybuild-framework/pull/1882
Add more complete system compiler and system MPI support to EasyBuild ... https://github.com/easybuilders/easybuild-easyblocks/pull/1106
OpenMPI vsystem, GCC vsystem ... https://github.com/easybuilders/easybuild-easyconfigs/pull/4136
Multi-threaded FFT ... https://github.com/easybuilders/easybuild-framework/pull/1802
Make QE easyblock aware of multithreaded FFT ... https://github.com/easybuilders/easybuild-easyblocks/pull/954
Handle gipaw correctly in QE ... https://github.com/easybuilders/easybuild-easyblocks/pull/1041
QuantumESPRESSO v6.0 ... https://github.com/easybuilders/easybuild-easyconfigs/pull/3809

Page 7

CP2K add psmp build support for foss ... https://github.com/easybuilders/easybuild-easyblocks/pull/1043
CP2K v4.1 ... https://github.com/easybuilders/easybuild-easyconfigs/pull/3810
DL_MESO GPU implementation ... https://gitlab.e-cam2020.eu/e-cam/Meso-Multi-Scale-Modelling-Modules/merge_requests/18/diffs

Page 8

E-CAM Deliverable D1.1 ... https://www.e-cam2020.eu/wp-content/uploads/2017/01/D1.1_30112016.pdf
OpenPathSampling ... http://openpathsampling.org/
LAMMPS ... http://lammps.sandia.gov/
GROMACS ... http://www.gromacs.org/
LAMMPS website ... http://lammps.sandia.gov/
GROMACS website ... http://www.gromacs.org/

Page 9

Gromacs Lysozyme ... http://www.bevanlab.biochem.vt.edu/Pages/Personal/justin/gmx-tutorials/lysozyme/
JUQUEEN ... http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/Configuration/Configuration_node.html
High-Q Club ... http://www.fz-juelich.de/ias/jsc/EN/Expertise/High-Q-Club/_node.html

Page 10

presentation on the runtime optimisaton of GROMACS ... http://bioexcel.eu/wp-content/uploads/2016/05/2016-05-11-Performance-Tuning-and-Optimization-of-GROMACS.pdf
BioExcel CoE ... http://bioexcel.eu/
MareNostrum3 ... https://www.bsc.es/innovation-and-services/supercomputers-and-facilities/marenostrum
Marconi ... http://www.cineca.it/en/content/marconi

Page 11

Libescdf ... https://gitlab.e-cam2020.eu/esl/escdf
Electronic Structure Common Data Format Specifications ... http://esl.cecam.org/ESCDF_-_Electronic_Structure_Common_Data_Format
Quantum ESPRESSO ... http://www.quantum-espresso.org
CP2K ... https://www.cp2k.org
Libescdf ... https://gitlab.e-cam2020.eu/esl/escdf
Towards a Common Format for Computational Materials Science Data ... http://psi-k.net/download/highlights/Highlight_131.pdf
Libescdf module ... http://e-cam-electronic-structure-modules.readthedocs.io/en/latest/modules/escdf/readme.html
LibOMM ... https://gitlab.e-cam2020.eu/ESL/omm/tree/master
CP2K ... https://www.cp2k.org
PIM ... https://gitlab.e-cam2020.eu/Quantum-Dynamics/PIM/tree/deliverables
Libescdf ... https://gitlab.e-cam2020.eu/esl/escdf/tree/master
Electronic Structure Common Data Format ... http://esl.cecam.org/ESCDF_-_Electronic_Structure_Common_Data_Format
HDF5 ... https://support.hdfgroup.org/HDF5/
Darshan 3.1.3 ... http://www.mcs.anl.gov/research/projects/darshan/publications/

Page 12

Quantum ESPRESSO website ... http://www.qe-forge.org/gf/project/q-e/frs/?action=FrsReleaseBrowse&frs_package_id=18
small-scale test 2 ... http://qe-forge.org/gf/project/q-e/frs/?action=FrsReleaseView&release_id=43
large-scale DEISA benchmark ... http://qe-forge.org/gf/project/q-e/frs/?action=FrsReleaseView&release_id=46
benchmark line given by CRAY-XT4 ... http://www.quantum-espresso.org/benchmarks/
SuperMUC ... https://www.lrz.de/services/compute/supermuc/systemdescription/

Page 14

SourceForge website ... https://sourceforge.net/projects/cp2k/files/
CP2K Benchmark Suite Section ... https://www.cp2k.org/performance
CP2K benchmark on HECToR machine ... https://www.cp2k.org/performance:hector-h2o-64

Page 15

Hazel Hen ... http://www.hlrs.de/en/systems/cray-xc40-hazel-hen/
PIM ... https://gitlab.e-cam2020.eu/Quantum-Dynamics/PIM/tree/deliverables
Quantics ... http://stchem.bham.ac.uk/~quantics/doc/index.html
Deliverable 3.1 ... https://www.e-cam2020.eu/wp-content/uploads/2017/01/D3.1_29122016.pdf

PotMod ... http://e-cam-quantum-dynamics-modules.readthedocs.io/en/latest/modules/PotMod/readme.html
AuxMod ... http://e-cam-quantum-dynamics-modules.readthedocs.io/en/latest/modules/AuxMod/readme.html
ClassMC ... http://e-cam-quantum-dynamics-modules.readthedocs.io/en/latest/modules/ClassMC/readme.html
SodLib ... http://e-cam-quantum-dynamics-modules.readthedocs.io/en/latest/modules/SODLIB/sod_readme.html
ChebLib ... http://e-cam-quantum-dynamics-modules.readthedocs.io/en/latest/modules/cheb_doc/cheb_readme.html

Page 16

PIM code ... https://gitlab.e-cam2020.eu/Quantum-Dynamics/PIM/tree/deliverables
CH5+ molecule with classical sampling method ... https://gitlab.e-cam2020.eu/Quantum-Dynamics/PIM/tree/deliverables/tests/ch5/CLASSICAL
"Pyrazine, 24 modes" ... http://s3.amazonaws.com/academia.edu.documents/42176548/Molecular_dynamics_of_pyrazine_after_exc20160205-14144-1dvyozm.pdf?AWSAccessKeyId=AKIAJ56TQJRTWSMTNPEA&Expires=1481299492&Signature=jcqyAKnKnchf9RBk8jYerPMJ%2B2s%3D&response-content-disposition=inline%3B%20filename%3DMolecular_dynamics_of_pyrazine_after_exc.pdf

Page 17

Ludwig ... https://ccpforge.cse.rl.ac.uk/gf/project/ludwig/
DL_MESO_DPD ... http://www.scd.stfc.ac.uk/support/40694.aspx
DL_MESO project website ... http://www.scd.stfc.ac.uk/support/40694.aspx
Ludwig svn repository ... https://ccpforge.cse.rl.ac.uk/gf/project/ludwig/scmsvn/?action=AccessInfo
First GPU version of DL_MESO_DPD ... https://gitlab.e-cam2020.eu/e-cam/Meso-Multi-Scale-Modelling-Module/merge_requests/18/diffs
Add bond forces to the GPU version of DL_MESO_DPD ... https://gitlab.e-cam2020.eu/e-cam/Meso-Multi-Scale-Module/merge_requests/19/diffs

Citations

[1] Alberto García William P. Huhn Mathias Jacquelin Weile Jia Björn Lange Lin Lin Jianfeng Lu Wenhui Mi Ali Seifitokaldani Álvaro Vázquez-Mayagoitia Chao Yang Haizhao Yang Victor Wen-zhe Yu, Fabiano Corsetti and Volker Blum. Elsi: A unified software interface for kohn-sham electronic structure solvers. *submitted*, 2017. https://arxiv.org/abs/1705.11191v1.