E-CAM Software Porting and Benchmarking Data V

E-CAM Deliverable 7.10
Deliverable Type: Report
Delivered in April, 2021

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	D7.10
Deliverable Nature	Report
Dissemination Level	Public
Contractual Date of Delivery	Project Month 64(31st January, 2021)
Actual Date of Delivery	21st April, 2021
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 machines (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 V
	ID:	D7.10
	Version:	As of April, 2021
	Status:	Accepted by WP leader
	Available at:	https://www.e-cam2020.eu/deliverables
		with citable version on the E-CAM Zenodo Community page
	Document history:	Internal Project Management Link
Review	Review Status:	Reviewed
Authorship	Written by:	Alan O'Cais(Juelich Supercomputing Centre)
	Contributors:	Jony Castagna (STFC), Fredrik Robertsén (CSC), Antti Puisto (Aalto University)
	Reviewed by:	Godehard Sutmann (Juelich Supercomputing Centre)
	Approved by:	Godehard Sutmann (Juelich Supercomputing Centre)

Document Keywords

Keywords:	E-CAM, HPC, CECAM, Materials

21st April, 2021

***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. The development of the modules was done in the context of the E-CAM program of Extended Software Development Workshop (ESDW) events.

As this is the final deliverable in the series we present work related to a total of 13 modules to give more complete context to what is presented. We focus the discussion on three areas:

- European Environment for Scientific Software Installations (EESSI):
  - We see that this effort creates the opportunity to provide complete software stack portability for the E-CAM community, optimised for all major CPU architectures.
  - We show that a centrally maintained, web-based software stack can compete with HPC installations in terms of performance *including* when considering interconnect hardware (which impacts the performance of MPI).
- Accelerator portability
  - We present showcase implementations of Kokkos in DL_Meso and n2p2 (as part of LAMMMPS).
  - We also present a HIP implementation of the Cubble foam simulation application.
  - We show that in both cases, using implementations that are portable among accelerators does not necessarily lead to any drop in performance.
- Performance analysis of specific applications
  - We study the performance of the DL_Meso application to 2048 GPUs on 3 different generations of NVIDIA GPUs and compare the performance for both a simple and complex use case. DL_Meso exhibits excellent weak scalability up to 18 billions particles on all GPU generations.
  - We look at the performance improvement of a Material Point Method code when leveraging the ALL load-balancing library.
  - We assess the performance impact of introducing additional scientific capabilities (the ability to simulate crystalline capillaries) to the Ludwig lattice Boltzmann application.

We also include a short assessment of the impact of the work package on the community. There, we show how we have prepared the extended E-CAM community for the demands of upcoming exascale systems through addressing core cross-cutting topics such as

- Load balancing - through the development of the ALL load-balancing library,
- Intelligent High Throughput Computing - through the development of the `jobqueue_features` library
- Accelerator portability - through the promotion of libraries such as Kokkos, and implementations in E-CAM codes,
- Portability of entire software stacks - through contributions to EasyBuild and, now, EESSI E-CAM has paved the way for the portability of entire application stacks and workflows to all the architectures that will be available through EuroHPC.

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 new modules related to those developed in the ESDW events to 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.

Previously, this series of deliverables has been presented based on the Work Package (WP) contributions. In this final deliverable we take the opportunity to present our results in more thematic manner, and also present work related to 13 E-CAM modules. In this deliverable we address

- portability of entire software stacks through European Environment for Scientific Software Installations
- accelerator portability through Kokkos and HIP

as well as presenting two individual application performance analysis cases.

The modules and applications have been benchmarked on the High Performance Computing (HPC) resources available to the project. Scaling and performance plots were generated for a variety of relevant systems and architectures (detailed in Section 3).

2 Porting and Optimisation

This section covers the hardware resources available for WP7 "Hardware considerations and the PRACE relationship" and a new approach to the porting effort required for these architectures.

In previous iterations of this deliverable, we have maintained a section which describes the E-CAM workflow when it comes to our porting, optimisation, benchmarking and scaling efforts. Since the current report is the final in the series, we forego this section to focus instead on a future-oriented workflow that we recommend to community software developers. This workflow centres around the EESSI initiative that E-CAM has been involved with, that will be described in detail below.

2.1 Available Resources

2.1.1 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 specific HPC architecture component was not already available to the project. Given the discussion with respect to hardware in D7.7: Hardware Developments IV[1], we focus our efforts on cluster-type systems (with latest architectures) and accelerators. Our primary development hardware has been

- JUWELS: a modular supercomputing architecture with a GPU *booster*[2], #7 in Top500 as of November 2020 (available through partner FZJ-JSC);

for general development work.

2.1.2 PRACE Resources

In the case of Partnership for Advanced Computing in Europe (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 been successful three times in acquiring additional resources through this second avenue, making an additional 1.55M core hours available to the project.

We provide the complete list of supercomputers available through PRACE here (the configuration details of the hardware are available in the PRACE Project Access Terms of Reference):

- Marconi Broadwell, 180.000 core hours
- Marconi KNL, 3.050.000 core hours
- Joliot Curie AMD, 1.715.000 core hours
- Joliot Curie KNL, 470.000 core hours
- Joliot Curie SKL, 660.000 core hours
- JUWELS, 350.000 core hours
- Hawk, 2.300.000 core hours
- MareNostrum4, 1.200.000 core hours
- Piz Daint, 2.720.000 core hours, 40.000 node hours

For 2019/2020, we have requested access to Joliot-Curie, JUWELS, Piz-Daint and Marenostrum since this set covers all our architecture and scalability needs (in addition to our existing access to resources).

2.2 European Environment for Scientific Software Installations

The European Environment for Scientific Software Installations (EESSI) (pronounced "easy") is a collaboration between a number of academic and industrial partners in the HPC community to which E-CAM is contributing. Through the EESSI project, they want to set up a shared stack of scientific software installations to avoid not only duplicate work across HPC sites but also the execution of sub-optimal applications on HPC resources.

[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.

They want to focus not only on the performance of the software, but also on automating the workflow for maintaining the software stack, thoroughly testing the installations (including correctness, performance and scalability), and collaborating efficiently.

2.2.1 Scope of EESSI

For end users, EESSI wants to provide a uniform user experience with respect to available scientific software, regardless of which system they use. The software stack is intended to work on laptops, personal workstations, in the cloud and on the largest HPC infrastructures. This means we will need to support different CPUs, networks, GPUs, and so on. We intend to make this work for any Linux distribution, and a wide variety of CPU architectures (Intel, AMD, ARM, POWER, RISC-V) and accelerators.

2.2.2 Architecture of EESSI

The project replicates how CERN implements software distribution using the CernVM File System (CernVM-FS) with modifications for the HPC space. The basic architecture can be seen in Fig. 1.

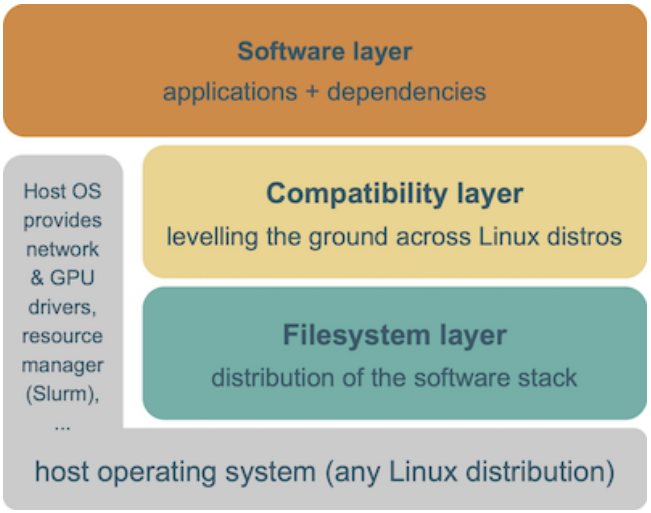

Figure 1: Architecture of the EESSI project.

The bottom layer is the filesystem layer CernVM-FS, which is responsible for distributing the software stack across clients. The middle layer is a compatibility layer, which is based on Gentoo Prefix and ensures that the software stack is compatible with multiple different client operating systems. The top layer is the software layer which contains the actual scientific software applications and their dependencies. This layer is managed via EasyBuild, a software build and installation framework that facilitates the management of (scientific) software on HPC systems in an efficient way. EasyBuild is used by dozens (and possibly hundreds) of HPC sites including JSC and CSCS who are PRACE Hosting Members.

The design is intended to allow the support of multiple architectures (x86, ARM, POWER,...) as well as accelerators such as GPUs. The compatibility and software layers are replicated (and individually optimised) for different architectures within CernVM-FS, with each client automatically addressing the appropriate architecture-specific subdirectory.

The host OS still provides a couple of things, like drivers for network and GPU, support for shared filesystems like GPFS and Lustre, a resource manager like Slurm, and so on.

We will use ReFrame (developed at CSCS) to do correctness, performance and scalability testing of the stack. The research methods, algorithms, and code parallelization approach used within the applications themselves is outside our control, but we will provide each software package such that it's hardware capabilities are fully exposed.

The details of how the implementation functions is described in the EESSI documentation and is based on the model currently implemented by ComputeCanada [2].

2.2.3 Current Status

EESSI is currently in the pilot phase, where the design is being realised and a variety of related workflows are being automated. For the pilot phase, a particular set of applications are being used which are notoriously difficult to install

and/or have a very large user base. These include GROMACS, OpenFoam, TensorFlow, Bioconductor and OSU-Micro-Benchmarks (for testing MPI performance). As a result of the extensive dependencies of these applications, the pilot software stack includes more than 100 software packages (and hundreds of Python and R extensions).

EESSI currently supports all major CPU vendors (Intel, AMD, Power and ARM) and a wide set of micro-architectures:

- `x86_64`
  - generic (currently implies `-march=x86-64` and `-mtune=generic`)
  - AMD
    - Zen2 (Rome)
  - Intel
    - Haswell
    - Skylake (`avx512`)
- `aarch64/arm64`
  - generic (currently implies `-march=armv8-a` and `-mtune=generic`)
  - AWS Graviton2
  - Fujitsu A64FX
  - Marvell Thunder X2
- `ppc64le`
  - IBM POWER9

EESSI is also eager to begin porting the infrastructure to RISC-V and is currently waiting for access to hardware. Ultimately, including an application in EESSI means the porting of that application to all these architectures, and covers all of the architectures available to E-CAM via PRACE.

The full set of repositories related to EESSI can be found on GitHub, and a documentation website for EESSI is already available. A number of tutorials related to EESSI were held (and recorded) during the 6th EasyBuild User meeting, these are linked on the 6th EasyBuild User meeting website and covered CernVM-FS, Singularity and ReFrame.

2.2.4 Accessing the EESSI infrastructure

The EESSI software stack can be made available by installing and configuring a CernVM-FS client to use the EESSI repositories. This method requires root privileges on the host, however.

One can also explore the EESSI software stack in entirely in user space on any system that supports Singularity. This access method is covered in the EESSI documentation.

2.2.5 Optimisation using EESSI

One of the goals within EESSI is to improve the lives of HPC application developers. An *identical* software stack will be available on their personal laptops and workstations as that of the HPC systems that they use. EESSI will provide a stable, optimised and continuously updated software stack upon which they can build and test their applications.

This introduces extensive continuous integration capabilities, for example through the use of light-weight containers (the latest EESSI container which includes the software stacks of all architectures is under `200MB`) and will support a number of open source compilers. It is also extensible, and can be used to also test other commercial compilers, as well as licensed software.

E-CAM has also developed a GitHub Action that can leverage EESSI. The EESSI GitHub Action is available to use in any GitHub workflow. This development is documented in the E-CAM module EESSI-based GitHub Action for Continuous Integration.

The relationship between EESSI and application developers can be a symbiotic one: application developers will ensure that the builds that appear within EESSI are correctly built and optimised (and help develop ReFrame tests to confirm this). EESSI will provide a lightweight distribution platform that actively tests the applications (including their performance and scalability) on a wide variety of CPUs, GPUs and interconnects.

EESSI will also provide performance optimisation tools, such as Scalasca which has been used in the past by E-CAM, to allow developers to easily explore the optimisation potential within their applications.

3 Modules and Application Codes

In the past, we have grouped the modules that we present in this family of deliverables by the Work Package from which they originate, providing information such as:

- Relevance to E-CAM (including relevant modules and ESDWs);
- Description of work

In this final deliverable, we take a slightly broader scope, retaining some modules with very specific WP connections but primarily focusing on modules that have scope beyond a single WP and address a wider HPC perspective:

- the performance of the EESSI software stack
- accelerator portability within application codes
- load-balancing as a means for improving portability, scalability and resilience

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.

3.1 Performance of the EESSI software stack

While the concept behind EESSI, as outlined in Sec. 2.2, is very attractive, it's success will depend on it's ability to *perform* as advertised. In particular, of critical interest is the ability of the EESSI software stack to leverage the hardware available on HPC resources. Given the number of nodes that appear in HPC infrastructures, it is the performance of EESSI on the interconnect that is of initial primary concern.

As reference modules for the work described here we use the following 2 modules:

- MPI support for EESSI-based containers
- EESSI and vGPU support in Magic Castle

Our initial investigation was to check basic point-to-point latency and bandwidth. For this we used the JUWELS system (a SkyLake architecture with EDR infiniband), we found a minimum latency of $< 1\mu$s and a point to point bandwidth of about 12GB/s. This is entirely consistent with the performance of the system-recommended Message Passing Interface (MPI) stack.

While such a confirmation is encouraging, it is in the performance of the applications themselves that is of most interest. To investigate this we select one of the pilot applications, GROMACS, which is a common Molecular Dynamics (MD) engine within E-CAM. We take a test case for GROMACS and compare the performance of the EESSI installation using the EESSI MPI, and a system-provided GROMACS installation which uses the system-recommended MPI installation. This is done for the JUSUF system (AMD EPYC with HDR100 interconnect), and the results can be seen in Fig. 2.

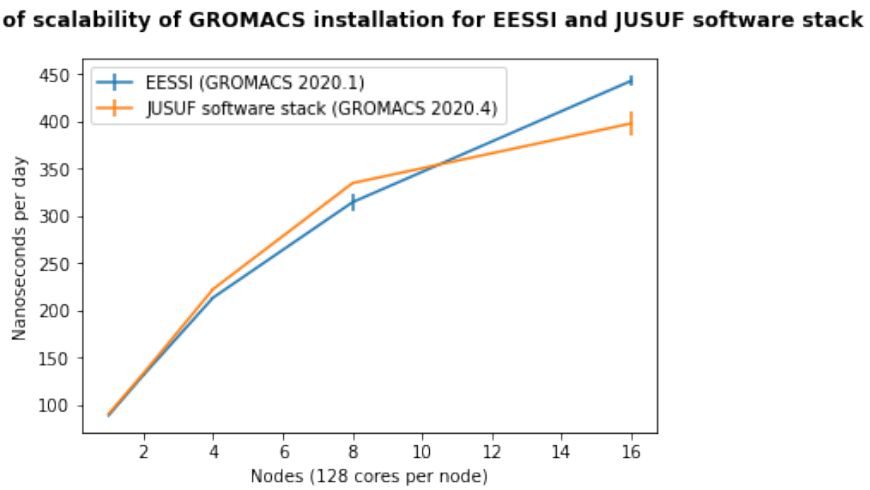

Figure 2: Comparison of the scalability of GROMACS using EESSI and the JUSUF software stack. The number of nodes is on the X-axis, giving a maximum of 2048 cores. On the y-axis we have the number of nanoseconds GROMACS can simulate per day (which ideally should scale linearly with the core count).

There are many qualifications on the results shown in Fig. 2, such as:

- the particular test case scales poorly (with only roughly 30% of perfect performance at 2048 cores)[3],
- the versions of the used GROMACS packages differ slightly,
- we have used a particular hardware configuration with 4 OpenMP threads per physical node.

Nevertheless, it is highly encouraging to note that the performance of both installations is entirely consistent. This is the case even though the EESSI installation was run from inside a singularity container.

There are benchmarking efforts underway in EESSI to automatically gather and share this type of information for a variety of architectures. Ultimately, it is hoped that the application developers would work with EESSI to ensure that the software distributed works as well as possible on the architectures EESSI will support. The currently supported list of architectures is given in Section 2.2.3 and already covers all of the CPU architectures found in current PRACE production systems.

3.1.1 Customising the MPI installation

EESSI is very aware that collective operations at scale is a crucial point for any communication library and that local customisation of MPI installations could have a significant impact there. The grouping of interest around the middle-layer library `UCX` and vendor support for the `libfabric` library make being able to create a universally performant MPI installation a possibility, but EESSI should still allow the scope for leveraging a site-customised MPI implementation.

To facilitate this, EESSI will leverage the Application Binary Interface (ABI) compatibility efforts of the MPI implementations themselves. In particular, the MPICH ABI Compatibility Initiative means that 6 differing MPI implementations are ABI compatible. EESSI is working on creating the ability for sites which prefer another ABI compatible implementation to be able to easily inject them into the stack (without requiring any recompiling or relinking). This should make large computing sites a lot more comfortable with using the EESSI stack in production environments.

3.2 Accelerator Portability

In previous WP7 deliverables, we have made the case that applications should be looking at leveraging frameworks that can relieve the developer of a lot of the effort of optimising their application for the latest available accelerators. In E-CAM we have repeatedly mentioned Kokkos [3] in this regard.

In this section we also introduce an E-CAM evaluation of HIP, which is a C++ runtime API and kernel language that allows developers to create portable applications for AMD and NVIDIA GPUs.

3.2.1 Kokkos

A number of modules have been developed recently in E-CAM that connect back to Kokkos. As reference modules for the work described here we use the following 3 modules:

- `n2p2` - Improved link to HPC MD software - includes changes to `n2p2` (which provides neural network potentials) to use it with Kokkos (and hence on multi-core CPUs and GPUs) which were implemented in a pull request that added CabanaMD support to `n2p2`
- DL_MESO (DPD) on Kokkos: Verlet Velocity step 1 and DL_MESO (DPD) on Kokkos: Verlet Velocity step 2 - which implemented some initial support for Kokkos in the DL_Meso package.

With respect to the `n2p2` integration in LAMMPS, we also note that E-CAM continues to maintain the LAMMPS support in EasyBuild, updating the LAMMPS installation process in EasyBuild for the latest stable release. This support includes architecture support for 3 AMD instruction set variants, 4 ARM instruction set variants, 7 Intel instruction set variants, 3 Power instruction set variants, 13 NVIDIA GPU generations, 2 AMD GPU generations and 1 Intel GPU generation. In addition, E-CAM has created a extensive set of training materials for running LAMMPS at scale which include a large section on how to leverage Kokkos in LAMMPS. These training materials include a selection of performance results related to the various performance-oriented packages available for LAMMPS. In Fig. 3, we share the comparison of the native CUDA GPU package in LAMMPS to that of the Kokkos package on up to 8 nodes (32 physical GPUs). We can see that the performance is consistent with both implementations.

The results of the DL_Meso Kokkos compared to the CUDA implementation are similar in this respect. For 5.12 million particles of the *Large Mixture* test case, we get 0.00114s and 0.00117s per kernel execution with both versions, which

[3]The test case is taken from the PRACE Unified European Applications Benchmark Suite for Gromacs and uses the `ion_channel` dataset that uses PME for electrostatics. This dataset is intended for Tier-1 systems, using the Tier-0 dataset would likely provide better scalability.

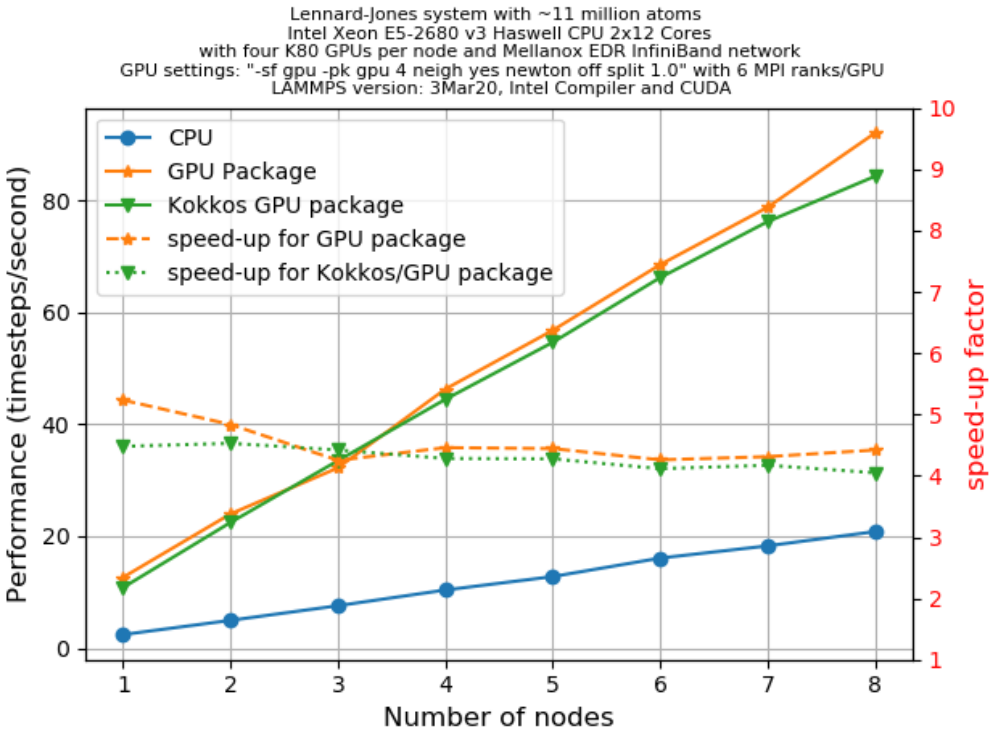

Figure 3: Comparison of the native CUDA GPU package in LAMMPS to that of the Kokkos package on up to 8 nodes (with 4 GPUs per node).

indicate no loss of performance in using Kokkos compared to native CUDA code. However, the data transfer between host and device currently occurs at every time step in the Kokkos version, taking 0.5721s and 0.4789s, therefore having a negative impact on the overall performance. For a fair comparison, this data should be transferred upstream to the time marching loop as is done in the CUDA version.

3.2.2 HIP

The canonical model for Wet and semi-dry foams is the Durian bubble model, which implements the foam as assemblies of spherical bubbles. Due to its practical and simple implementation, and surprisingly accurate description of soft particle systems, it has been the reference model for jammed soft materials for a long time. Several in-house implementations for the model exist. The largest ones run in parallel on multiple CPUs using variants of the MPI implementations. These are capable of simulating the dynamics of systems with sizes beyond 10^5 bubbles within a few days wall clock time. The model relies on local short-range interactions between the bubbles, making it a prototype case for almost perfect scalability. However, recent extensions of the model for the purposes of simulating coarsening break this perfect picture: Large number of bubbles are required for the simulations to reach experimentally relevant time-scales, and the implementation of gas exchange requires effective long range interactions between the bubbles to be computed.

Relevance for E-CAM

The Cubble case attempts another approach. Here an in-house CPU code is implemented to be run as much as possible on a completely different architecture, the GPUs, which are considered as a promising route to extreme-scale computing. While a comparison between run-times of different implementations running on either single CPU or single GPU hardly makes sense, the Cubble implementation is able to simulate experimentally relevant foam sizes of (10^6) bubbles, practically unreachable by a single thread CPU code, within just a few hours.

Three modules have been developed related to this effort and act as reference modules for the work described here:

- Modules Cubble Static and Cubble Flow which concern the initial CUDA implementation. These modules report CUDA based implementations of a CPU based bubble dynamics code implementing foam coarsening under different boundary conditions. These modules optimize the data-structures and algorithms used in a CPU based implementation for efficient execution on single GPU. The development optimized the code for the cur-

rent state-of-the-art GPUs, NVIDIA Volta v100 on the Puhti cluster. Example simulations showing coarsening of foam starting from one million bubbles is displayed in Fig. 4.

- Module CUBBLE HIP which concerns the effort for the implementation support of HIP via the HIP porting tools. More details are below.

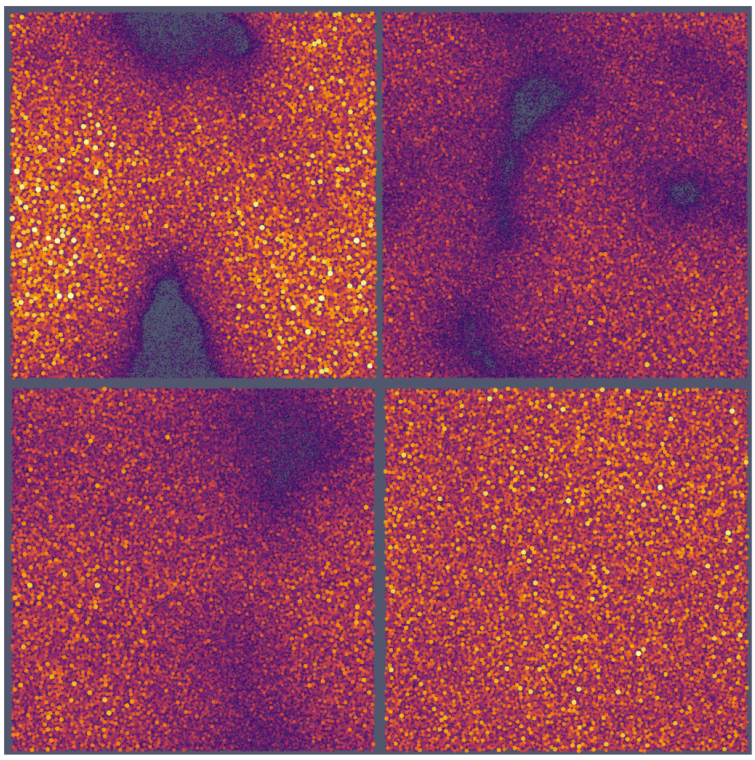

Figure 4: Cubble simulation in 2D showing a dynamical phase separation during the coarsening of a foam.

The future CSC resources include the LUMI HPC cluster, which includes a large array of AMD Instinct GPUs. As is well known, CUDA is an NVIDIA specific platform or API for Nvidia GPUs. Therefore, CUDA based programs are limited to NVIDIA hardware. HIP, on the other hand, provides porting tools which do most of the work to convert CUDA code into portable C++ code that uses the HIP APIs allowing the code to execute on AMD platforms as well, while maintaining the same performance as native CUDA code. The CUBBLE HIP module provides a HIP based implementation of the bubble dynamics Cubble code allowing it to be run also on non-NVIDIA manufactured GPUs. Our tests indicate, that the HIP implementation does not add overhead to the execution times of the code on NVIDIA GPUs. This is a natural, since HIP is implementing a subset of the CUDA API and when compiled for NVIDIA hardware the HIP code just calls the equivalent CUDA functions through header wrappers. To verify this, a set of test simulations were run with the average timings gathered in Table 1 below.

Implementation	t_{avg}
CUDA	609.25 s
HIP	606.19 s

Table 1: Comparison of the average wallclock execution times of the Cubble program's CUDA and HIP implementations.

3.3 Application Specific Porting and Scaling

We end this section with development efforts related to software packages that have been previously investigated in the WP7 series of deliverables: DL_Meso, the ALL load balancing library and Ludwig, a lattice Boltzmann code for complex fluids.

3.3.1 Porting and Scaling of DL_Meso on PRACE resources

In the previous iteration of this deliverable, we explored the scalability of DL_Meso up to 4096 GPUs on the Piz-Daint system. Here, we again use the implementation of ALL library in DL_MESO as the reference module, and explore the

scalability of DL_Meso on the different generations of NVIDIA GPUs that are available through our access to PRACE resources:

- Piz Daint - P100 NVIDIA GPU
- Marconi - V100 NVIDIA GPU
- JUWELS - A100 NVIDIA GPU

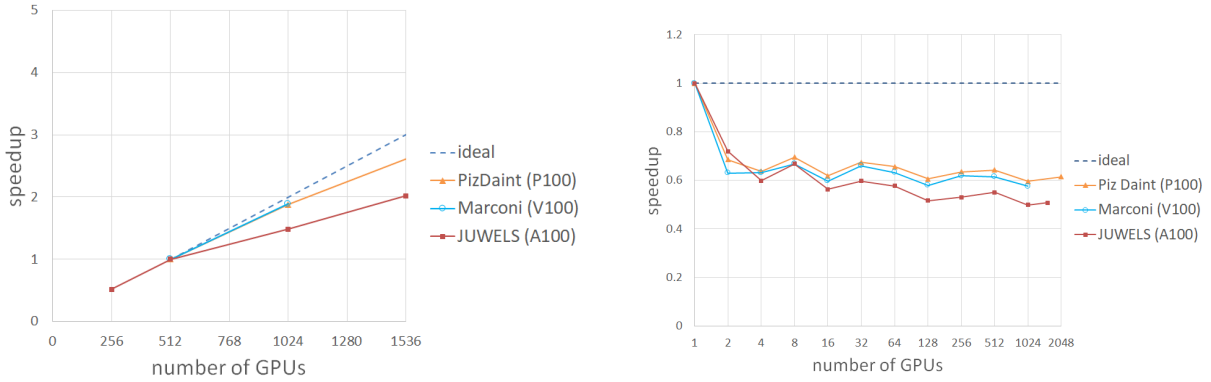

Figure 5: Strong and weak scalability plots for DL_Meso for a *simple* case using different GPU generations.

In Fig. 5, we show the strong and weak scalability for a simple test case. For the strong scalabilty (on the left), we note a dip in performance for A100 GPU, this is due to the fact that the A100 is significantly faster, and the halo communication is leaving the GPUs idle. We note that the A100 have 40GB of memory available (as opposed to 16GB for the P100), we would have been able to fit a larger simulation onto these GPUs which would have allowed better overlap between communication and computation. For weak scaling (on the right), we show scalability for up to 18 billion particles. We see generally consistent results, noting that the single GPU uses a different algorithm to the other cases which is why we see a drop in performance when moving to more than 1 GPU. From 2 GPUs, we see near perfect weak scalability.

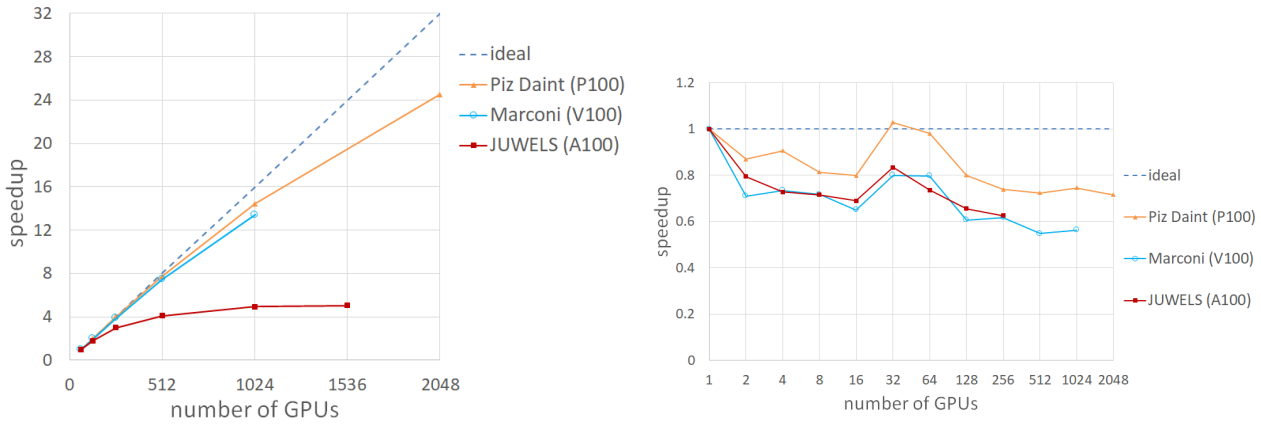

Figure 6: Strong and weak scalability plots for DL_Meso for a *complex* case using different GPU generations.

In Fig. 6, we show the strong and weak scalability for a more complex test case. For the strong scalabilty (on the left), we note a far pronounced performance drop-off for the NVIDIA A100. Again, the origin of this is the speed of the A100 GPUs, with enormous impact from the halo communications. A larger test case for 40GB of memory of the A100 would mitigate this to some extent. For our weak scalability results (on the right), we note that we get better performance than for the simple case. Of particular note is the consistent jump in performance at 32 GPUs, this is due to there being an ideal distribution of particles among the GPUs in this case. The number of steps used in the benchmark case is just 100 time steps. This number of steps is insufficient for the impact of the load-balancing from the ALL library to be notable. A realistic simulation would have thousands of time steps, and the ALL library would improve the weak scalability results even further.

3.3.2 Load-balancing for the Material Point Method

The Material Point Method (MPM) is used to simulate continuous matter and is especially suited for the simulation of large deformations. Once large deformations are present, a dynamic load balancing solution is sensible to efficiently simulate large systems. Even if the initial work distribution is good, it is very often changing during the simulation due to the dynamics of the system or changes in geometry. An example of such dynamic imbalance is shown in Fig. 7 with a sphere moving through a ring.

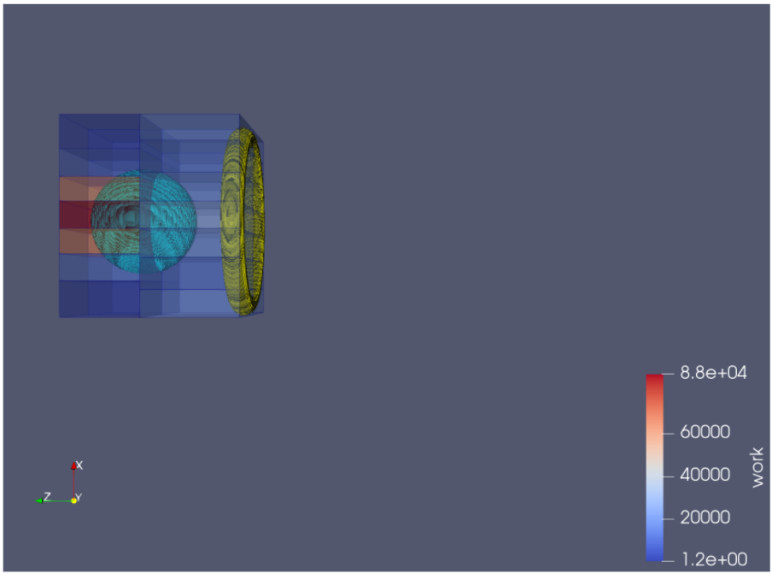

Figure 7: Load balancing of highly dynamic movements, a sphere flying through a ring.

The load balancing library ALL provides an easy plug and play solution to this problem. Thanks to the good load balancing provided by the library larger systems can be simulated with less computational cost.

The reference modules for this work are:

- the Integration of ALL in the MPM code GMPM-PoC, and the
- ALL Fortran Interface module which it required.

Taking the example seen in Fig. 7, we can study the quality of the load balancing, q, using the formula

$$q = \frac{w_{max} - w_{min}}{w_{max} + w_{min}} \tag{1}$$

where w_{max}, w_{min} is the maximum and minimum work, respectively, on a single domain. With ideal balance, the value of q would be zero.

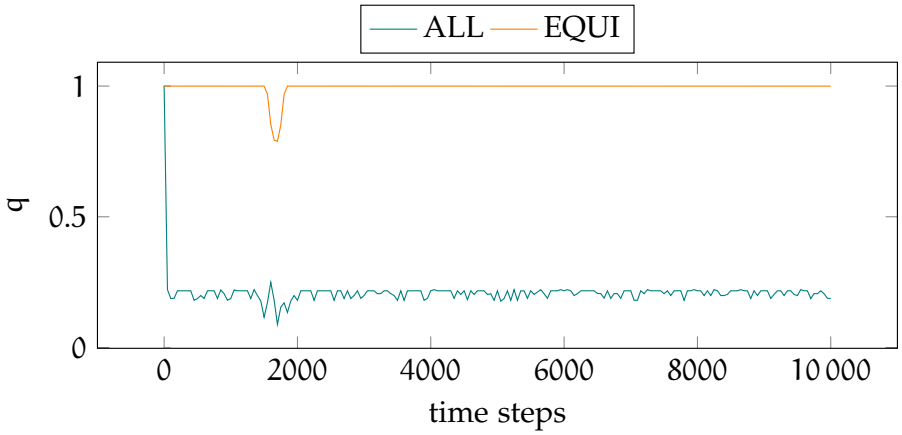

Figure 8: Evolution of q (from Eq. 1) over time as the sphere of Fig. 7 passes through the ring. We show the result with an equi-distribution of work across processes and when using ALL. The ideal value of q is zero.

In Fig. 8, we compare the evolution of q as the sphere moves through the ring, with and without the ALL library. We can clearly see that ALL provides far greater load balancing. It never reaches an ideal q value of zero as we are eventually restricted by a grid alignment requirement of the code, and a minimum domain size per processor.

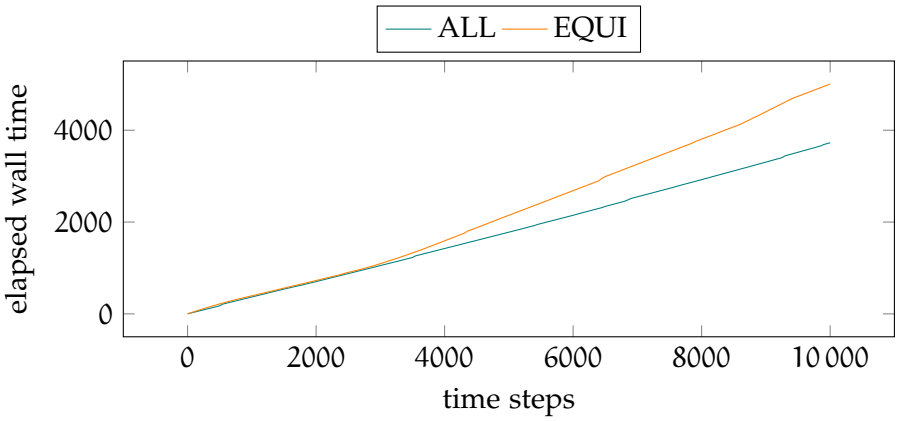

Figure 9: Evolution of required simulation walltime of the example from Fig. 7 with an equi-distribution of work across processes and when using ALL.

In Fig. 9, we show that this improved load-balance leads to a ~30% reduction in the time to solution for this problem over 10,000 time steps.

3.3.3 Expanding the Capabilities of the Ludwig Lattice Boltzmann Code

Finally, we provide some scalability results related to two modules that expand the capabilities of the Ludwig lattice Boltzmann code for complex fluids:

- Externally imposed chemical potential gradient for binary fluid mixture
- Implementation of simple cubic, body-centered cubic, and face-centered cubic crystalline capillaries

Ludwig was one of the original codes that we initially benchmarked in E-CAM [4], we again benchmark the most recent version of Ludwig which includes the crystalline capillaries module mentioned above. The benchmarks are carried out on MareNostrum 4 which has 48 physical cores per node.

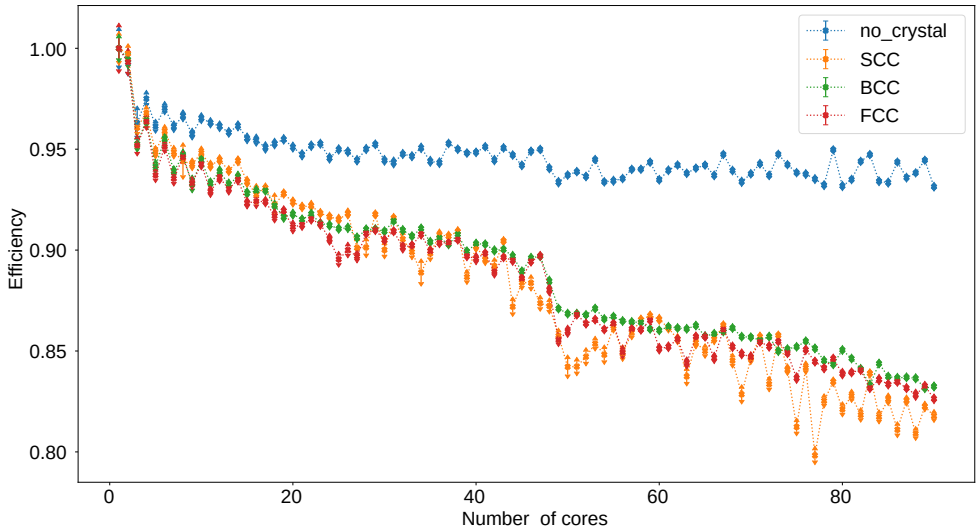

Figure 10: Weak scaling efficiency of Ludwig with and without simple cubic, body-centered cubic, and face-centered cubic crystalline capillaries.

In Fig. 10 we see the weak scaling efficiency results for Ludwig, where each CPU core simulates a $40 \times 40 \times 40$ region. Without capillaries we note that Ludwig, stabilises at about 95% scaling efficiency. When capillaries are included,

regardless of type, we note that efficiency falls as the node count increases. The capillaries also seem to be particularly susceptible to network conditions as there is a notable dip in performance when as we move beyond 48 cores (which is 1 node on MareNostrum). The origin of, and mitigation for, this performance impact is still under investigation.

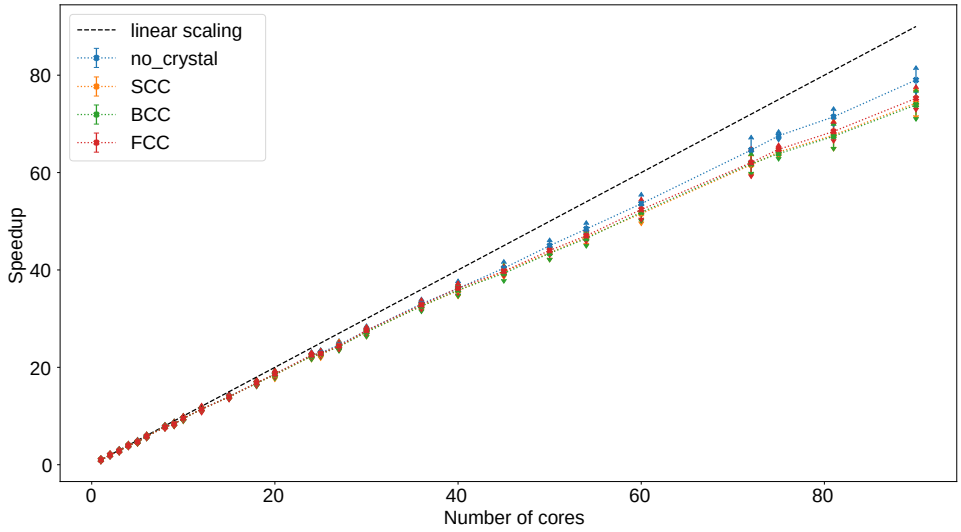

Figure 11: Strong scaling speedup of Ludwig with and without simple cubic, body-centered cubic, and face-centered cubic crystalline capillaries.

In Fig. 11, we take a single system of size $450 \times 450 \times 450$ and increase the core count up to 2 full nodes. In this case, the performance with and without capillaries are roughly consistent within errors but there are again indications that if we can increase the core counts that we will see a deviation from the results with no crystal.

Together, these results indicate that for very large systems we are likely to get reasonable efficiency, but that the implementation of capillaries does have an overall impact on Ludwig performance that will be exposed at scale.

4 Impact

WP7 has sought to prepare the extended E-CAM community for the demands of upcoming exascale systems. We have addressed core cross-cutting topics such as

- Load balancing - through the development of the ALL load-balancing library,
- Intelligent High Throughput Computing - through the development of the `jobqueue_features` library
- Accelerator portability - through the promotion of libraries such as Kokkos, and implementations in E-CAM codes,
- Portability of entire software stacks - through contributions to EasyBuild and, now, EESSI E-CAM has paved the way for the portability of entire application stacks and workflows to all the architectures that will be available through EuroHPC.

These alone are major contributions to preparing the E-CAM community, providing them with some of the tools that they will need to address major issues that will occur at scale: hardware diversity, portability of workflows, load imbalance, resiliency, coordinating ensemble petascale calculations... WP7 has gone far beyond cross-cutting work, however, and has repeatedly worked with individual applications to evaluate and improve their performance, collaborating with other projects in the EuroHPC eco-system along the way. A primary example of this is given by the DL_Meso application which, as a result of E-CAM, has been shown to be scalable to 4096 GPUs (the largest GPU partition available in Europe).

In addition, E-CAM has raised awareness of best practices in scientific computing, attempting to encourage people to consistently use

- version control,
- documentation,
- continuous integration,
- modular software development,
- configure/build/install installation processes and the tools that can support these,
- open source software licences.

E-CAM has trained hundreds of scientists and exposed them to HPC resources and tools that they might never otherwise have known. It has raised awareness of the EuroHPC eco-system among the thousands of scientists that are associated with Centre Européen de Calcul Atomique et Moléculaire (CECAM), and trained the next-generation of those scientists so that HPC is a go-to tool in their computational workflows.

The legacy of the efforts of WP7 will be felt in the community as the increasing amount of EuroHPC resources begin to materialise. E-CAM has put the community in a position where their entire software stack will be portable to these systems, and created the tools for them to use these resources to create great science without an overwhelming technical burden.

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
ABI Application Binary Interface
MD Molecular Dynamics
EESSI European Environment for Scientific Software Installations
MPM Material Point Method

URLs referenced

Page ii

https://www.e-cam2020.eu ... https://www.e-cam2020.eu
https://www.e-cam2020.eu/deliverables ... https://www.e-cam2020.eu/deliverables
E-CAM Zenodo Community page ... https://zenodo.org/communities/e-cam/search?page=1&size=20&q=deliverable&type=publication&subtype=deliverable
Internal Project Management Link ... https://redmine.e-cam2020.eu/issues/51
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

ALL load-balancing library ... http://slms.pages.jsc.fz-juelich.de/websites/all-website/
`jobqueue_features` ... https://github.com/E-CAM/jobqueue_features
EasyBuild ... https://docs.easybuild.io/en/latest/
EESSI ... https://eessi.github.io/docs/

Page 3

EESSI initiative ... https://eessi.github.io/docs/
JUWELS ... https://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUWELS/Configuration/Configuration_node.html
PRACE Preparatory Access Call ... http://www.prace-ri.eu/prace-preparatory-access/
PRACE Project Access Terms of Reference ... https://prace-ri.eu/wp-content/uploads/Terms_of_Reference_Call20.pdf
European Environment for Scientific Software Installations (EESSI) ... https://www.eessi-hpc.org/
academic and industrial partners in the HPC community ... https://eessi.github.io/docs/partners/

Page 4

CernVM-FS ... https://cernvm.cern.ch/portal/filesystem
Gentoo Prefix ... https://wiki.gentoo.org/wiki/Project:Prefix
EasyBuild ... https://easybuild.readthedocs.io/
ReFrame ... https://reframe-hpc.readthedocs.io/
EESSI documentation ... https://eessi.github.io/docs/

Page 5

full set of repositories related to EESSI ... https://github.com/EESSI/
documentation website for EESSI ... https://eessi.github.io/docs/
6th EasyBuild User meeting website ... https://easybuild.io/eum/
configuring a CernVM-FS client to use the EESSI repositories ... https://github.com/EESSI/filesystem-layer#clients
covered in the EESSI documentation ... https://eessi.github.io/docs/pilot/#accessing-the-eessi-pilot-repos
GitHub Action ... https://github.com/features/actions
EESSI GitHub Action ... https://github.com/marketplace/actions/eessi
EESSI-based GitHub Action for Continuous Integration ... https://e-cam.readthedocs.io/en/latest/Classical-MD-Modules/modules/EESSI/eessi_github_action.html
Scalasca ... https://www.scalasca.org/

Page 6

MPI support for EESSI-based containers . . . https://e-cam.readthedocs.io/en/latest/Classical-MD-Modules/modules/EESSI/singularity.html
EESSI and vGPU support in Magic Castle . . . https://e-cam.readthedocs.io/en/latest/Classical-MD-Modules/modules/EESSI/learnhpc_gpu.html

Page 7

PRACE Unified European Applications Benchmark Suite for Gromacs . . . https://repository.prace-ri.eu/git/UEABS/ueabs#gromacs
MPICH ABI Compatibility Initiative . . . https://www.mpich.org/abi/
n2p2 - Improved link to HPC MD software . . . https://e-cam.readthedocs.io/en/latest/Classical-MD-Modules/modules/n2p2/n2p2_improved_link_hpc/readme.html
CabanaMD support to n2p2 . . . https://github.com/CompPhysVienna/n2p2/pull/49
DL_MESO (DPD) on Kokkos: Verlet Velocity step 1 . . . https://e-cam.readthedocs.io/en/latest/Meso-Multi-Scale-Modules/modules/DL_MESO_DPD_onGPU/Kokkos_VV1/readme.html
DL_MESO (DPD) on Kokkos: Verlet Velocity step 2 . . . https://e-cam.readthedocs.io/en/latest/Meso-Multi-Scale-Modules/modules/DL_MESO_DPD_onGPU/Kokkos_VV2/readme.html
updating the LAMMPS installation process in EasyBuild for the latest stable release . . . https://github.com/easybuilders/easybuild-easyblocks/pull/2213
training materials for running LAMMPS at scale . . . https://fzj-jsc.github.io/tuning_lammps/

Page 8

Cubble Static . . . https://e-cam.readthedocs.io/en/latest/Meso-Multi-Scale-Modelling-Modules/modules/cubble_coarsening/cubble_coarsening_static/readme.html
Cubble Flow . . . https://e-cam.readthedocs.io/en/latest/Meso-Multi-Scale-Modelling-Modules/modules/cubble_coarsening/cubble_coarsening_flow/readme.html

Page 9

CUBBLE HIP . . . https://e-cam.readthedocs.io/en/latest/Meso-Multi-Scale-Modelling-Modules/modules/cubble_coarsening/cubble_hip/readme.html
CUBBLE HIP . . . https://e-cam.readthedocs.io/en/latest/Meso-Multi-Scale-Modelling-Modules/modules/cubble_coarsening/cubble_hip/readme.html
ALL load balancing library . . . http://slms.pages.jsc.fz-juelich.de/websites/all-website/
Ludwig . . . https://ludwig.epcc.ed.ac.uk/
implementation of ALL library in DL_MESO . . . https://e-cam.readthedocs.io/en/latest/Meso-Multi-Scale-Modelmodules/DL_MESO_DPD_onGPU/loadBalance/readme.html

Page 11

ALL . . . http://slms.pages.jsc.fz-juelich.de/websites/all-website/
Integration of ALL in the MPM code GMPM-PoC . . . https://e-cam.readthedocs.io/en/latest/Meso-Multi-Scale-Momodules/ALL_library/MPM_integration/MPMIntegration.html
ALL Fortran Interface . . . https://e-cam.readthedocs.io/en/latest/Meso-Multi-Scale-Modelling-Modules/modules/ALL_library/fortran_interface/readme.html

Page 12

Ludwig . . . https://ludwig.epcc.ed.ac.uk/
Externally imposed chemical potential gradient for binary fluid mixture . . . https://e-cam.readthedocs.io/en/latest/Meso-Multi-Scale-Modelling-Modules/modules/Lattice_Boltzmann/external_chemical_potential_gradient/readme.html
Implementation of simple cubic, body-centered cubic, and face-centered cubic crystalline capillaries . . . https://e-cam.readthedocs.io/en/latest/Meso-Multi-Scale-Modelling-Modules/modules/Lattice_Boltzmann/crystalline_capillaries/readme.html
MareNostrum 4 . . . https://www.bsc.es/marenostrum/marenostrum

Page 14

ALL load-balancing library . . . http://slms.pages.jsc.fz-juelich.de/websites/all-website/
`jobqueue_features` . . . https://github.com/E-CAM/jobqueue_features
EasyBuild . . . https://docs.easybuild.io/en/latest/
EESSI . . . https://eessi.github.io/docs/

Citations

[1] A. O'Cais, J. Castagna, and G. Sutmann, "D7.7: Hardware developments IV," Jun. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.3256137

[2] M. Boissonneault, B. E. Oldeman, and R. P. Taylor, "Providing a unified software environment for canada's national advanced computing centers," in *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)*, ser. PEARC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3332186.3332210

[3] H. C. Edwards, C. R. Trott, and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns," *Journal of Parallel and Distributed Computing*, vol. 74, no. 12, pp. 3202 – 3216, 2014, domain-Specific Languages and High-Level Frameworks for High-Performance Computing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731514001257

[4] 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