E-Infrastructures

H2020-EINFRA-2015-1

EINFRA-5-2015: Centres of Excellence for computing applications

EoCoE

Energy oriented Center of Excellence

for computing applications

Grant Agreement Number: EINFRA-676629

D1.16 - M12
Application Performance Evaluation

Project and Deliverable Information Sheet

EoCoE	Project Ref:	EINFRA-676629
	Project Title:	Energy oriented Centre of Excellence
	Project Web Site:	http://www.eocoe.eu
	Deliverable ID:	D1.16 - M12
	Lead Beneficiary:	CEA
	Contact:	Matthieu Haefele
	Contact's e-mail:	matthieu.haefele@maisondelasimulation.fr
	Deliverable Nature:	Report
	Dissemination Level:	PU*
	Contractual Date of Delivery:	M12 30/09/2016
	Actual Date of Delivery:	M12 29/09/2016
	EC Project Officer:	Carlos Morais-Pires

* - The dissemination level are indicated as follows: PU – Public, CO – Confidential, only for members of the consortium (including the Commission Services) CL – Classified, as referred to in Commission Decision 2991/844/EC.

Document Control Sheet

Document	Title :	Application Performance Evaluation
	ID :	D1.16 - M12
	Available at:	http://www.eocoe.eu
	Software tool:	LaTeX
Authorship	Written by:	Haefele (MdlS), Gibbon (JSC), Lührs (JSC), Rohe (JSC)
	Contributors:	Aeberhard (FZJ), Bernd (FZJ), Houzeaux (BSC), Kollet (FZJ), Latu (CEA), Napoli (JSC), Ould-Rouis (MdlS), Qu (RWTH), Salanne (MdlS), Sharples (FZJ), I. Herlin (INRIA), M. Gusso (ENEA), M. Levesque (MdlS), A. Joly (EDF), P. Börner, T. Breuer (JSC), F. Xing (BRGM)
	Reviewed by:	Haefele (MdlS), Gibbon (JSC)

Contents

List of Figures

List of Tables

1. Document release note

This document replaces D1.15 Application Performance Evaluation that has been delivered on M6. For the reader who read already the previous document, the major contribution material within this document with respect to the previous one can be found in the following sections:

- Section 3 reports on a second performance evaluation workshop that took place in Maison de la Simulation in May 2016
- Section 5 provides now the updated table for all 13 codes evaluated to date.
- Section 4 has been revised. The list of metrics and their definition have been adjusted and consolidated with Brian Wylie, scalasca developer and member of PoP CoE, in order to improve their reliability and their meaning.

2. Motivation

Within in its transversal basis (WP1), the EoCoE project has gathered a comprehensive range of HPC expertise that aims to enhance the performance of applications from the four domain pillars, thereby enabling them to effectively exploit the existing European computing infrastructure. Close interaction between WP1 and the application domains WP2-WP5 is a key feature of EoCoE, with the ultimate goal of expediting advances in simulations of low-carbon energy systems and technology.

In this context, application performance evaluation is an instrument of key importance, since it permits us to:

1. define the status of an application code at the moment when EoCoE HPC experts start to examine it,
2. monitor the impact of each code modification during the optimization process,
3. quantitatively assess the impact of such support activity when it comes to an end.

This deliverable report describes the status of performance evaluation activity over the first 6 months of the project, beginning with a dedicated workshop for this purpose, and various follow-up actions such as Section 4, which presents the definition of the EoCoE performance evaluation report and the performance metrics it uses; Subsection 4.3, on the establishment of an automated and reproduceable process that delivers all the required metrics; Section 5, which describes the system for monitoring progress in application optimisation.

3. Joint EoCoE-PoP benchmarking workshops

3.1 December 2015 in Juelich @ JSC

The first EoCoE-POP workshop on benchmarking and performance analysis brought together code developers of community codes associated with WP 2-5 with HPC experts associated with WP 1 and HPC experts from the CoE "POP". The goal of this 4-day event held at Jülich Supercomputing Centre from 8th-11th December, 2015 was to familiarise the developers from WP2-5 with state-of-the-art HPC performance analysis tools, enabling the teams to make a preliminary identification of bottlenecks, and to initiate the standardisation of benchmark procedures for these codes within the EoCoE project. The workshop comprised 4.5 hours of presentations on the benchmarking and performance tools followed by 12 hours of hands-on work supervised by the WP1 and PoP HPC experts.

Figure 1: Workshop participants and support activity during the first benchmarking workshop

As an initial step, all code developers were instructed on how to perform benchmarking within the JUBE[1] workflow environment, which will permit measurements to be documented, shared and rigorously reproduced over the project lifetime and beyond. Developers were then able to begin analysing their applications using specific HPC tools under the guidance of HPC experts (Score-P, Scalasca, Vampir, Paraver, Extrae, Darshan, VTune and others). Based on this face-to-face collaboration and common training, small teams of code developers and HPC experts from WP 1 were established, who have begun to follow up on the promising initial work to provide comprehensive benchmarks and performance data by the time the next workshop is held in June.

Each of the participating developer teams was allocated a WP1 mentor, tasked with assisting any follow-up benchmarking and tuning work, and acting as an initial contact point for enquiries going beyond the initial assessment (I/O issues, data management, visualisation etc). A summary of the participating codes is given in table 1. Four of these (ALYA, Metallwalls, PARFLOW and Gysela) belong to the set of codes already prioritised (triggered) for WP1 optimisation activity.

A further valuable outcome was the exchange of respective ideas and needs between code developers and HPC experts, as this helped clarifying the issues from either perspective and enabled both sides to interact more smoothly with a well defined focus on the next actions to be taken. For example, the requirements for a full code 'audit' from the EoCoE and POP perspectives were clarified: here it was decided that the initial benchmarking

[1] www.fz-juelich.de/jsc/jube

WP	Context	Code	Developer	WP1 contact
2	Wind farms	ALYA	Houzeaux (BSC)	Ould-Rouis (MdlS)
2	Ensemble forecasting	ESIAS	Bernd (FZJ)	Lührs (JSC)
3	Photovoltaics	PVnegf	Aeberhard (FZJ)	Napoli (JSC)
3	Materials	Metallwalls	Salanne	Haefele (MdlS)
4	Hydrology	PARFLOW	Kollet (FZJ)	Sharples
4	Geothermics	SHEMAT	Qu (RWTH)	Sharples
5	Plasma transport	Gysela	Latu (MdlS)	Guillame Latu

Table 1: Codes participating in first EoCoE benchmarking workshop

would take place within and immediately after the workshop by EoCoE WP1 members, whereas more in-depth follow-up analyses could be channelled via a formal request to POP at a later stage.

3.2 May 2016 in Saclay @ MdlS

The second joint EoCoE-POP workshop on benchmarking and performance analysis took place at Maison de la Simulation from 30th May - 2nd June 2016. The objectives and the organization of this workshop were similar to the previous one that took place in Jülich. A first version of the automated performance evaluation was available at that time and it sped up the process of getting started for all participants. This showed us that our methodology is improving and we plan to improve it further for the next workshop that will likely take place in Jan 2017.

This events welcomed the first two codes that are not part of the EoCoE consortium: ComPASS, developed at BRGM, the french national geological survey and Telemac, developed at EDF. The developers showed interest in joining this workshop and their feedback was good, they could learn about the performance tools as well as their codes. The framework in which they were welcome was not clear at the moment of the workshop. This experience will be used as a testbed for setting up an appropriate one for future codes that are not part of the consortium.

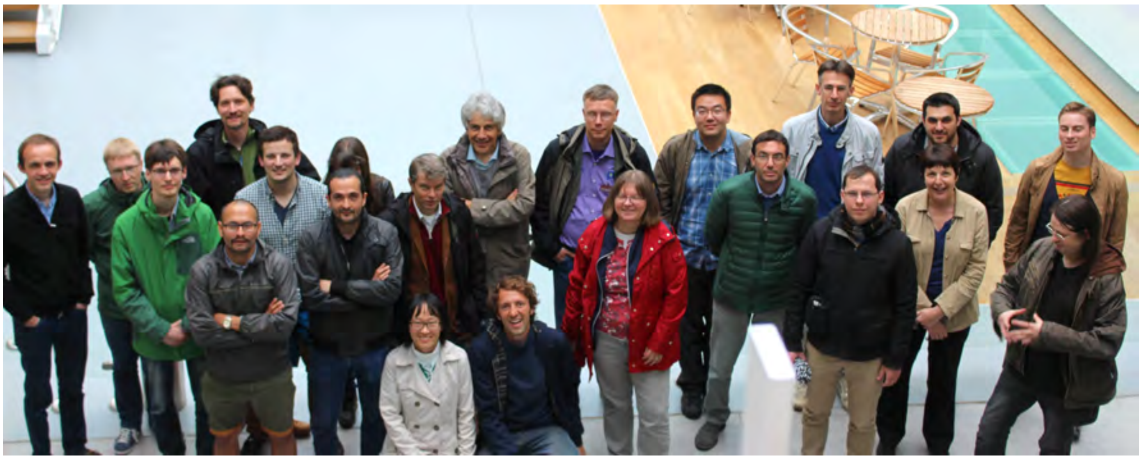


Figure 2: Workshop participants during the second benchmarking workshop

WP	Context	Code	Contact	WP1 Contact
2	meteorology	nowcast system	I. Herlin (INRIA)	Y. Ould Rouis (MdlS)
3	Quantum simulation	CP2K	M. Gusso (ENEA)	S. Lührs (JSC)
3	Molecular DFT	MDFT	M. Levesque (MdlS)	M. Haefele (MdlS)
4	River flows	TELEMAC	A. Joly (EDF)	Y. Ould Rouis (MdlS)
5	Particle transport	EIRENE	P. Börner (FZJ)	T. Breuer (JSC)
ext.	Geothermy	ComPASS	F. Xing (BRGM)	M. Haefele (MdlS)

Table 2: Codes participating in the second EoCoE benchmarking workshop

4. EoCoE performance evaluation report and metrics definition

Performance evaluation has the obvious purpose to uncover bottlenecks and possibly other technical areas of improvement for the codes under consideration. In order to verify the impact and success of code changes it is mandatory to apply it *iteratively and continuously* in a regular manner. In particular, it is *not* sufficient to analyse a code once and from the results create an optimised version of a code in a single step.

4.1 Organizational structure and reporting

The EoCoE management has carefully engineered a lean yet efficient organisational structure which ensures that such an ongoing and continuous process involving code developers and HPC-experts can be achieved and monitored, with a minimum of bureaucratic overhead. The elements and ingredients for this collaborative micro-community are

1. Permanent code teams, consisting of at least one developer and one HPC-experts, to corroborate the collaboration between in a sustainable manner.
2. Code identity card filled by the application developer to initiate the analysis.
3. A well-defined set of global performance metrics to have a common perspective on progress and development. Ideally, most of the initial measures are obtained during an EoCoE performance workshop.
4. The possibility to add further application-specific performance metrics if necessary.
5. A technical infrastructure based on Git which allows all code teams to share their reports and to provide a basis from which best practice methods can be deduced.

Appendix A shows the full information for the two most advanced codes in this performance evaluation process: Metalwalls and ESIAS.

4.2 Metrics definition and performance tools

The definition of all global performance metrics is given in the table 3. Several tools are used to extract these metrics:

- The UNIX *time* command is used to measure total application wall time
- Darshan[2] provides all metrics concerning IO

[2] http://www.mcs.anl.gov/research/projects/darshan/

- Scalasca[3] provides all metrics concerning MPI, OpenMP and load balancing
- PAPI[4], used through Scalasca, provides all performance counters
- SLURM[5] scheduling system is able to retrieve the memory footprint of the first MPI rank of the application.
- IdrMem[6] library is used to retrieve the memory footprint on systems where Slurm is not available.

Metrics Global.1, Global.2 and Global.3 might exhibit some inconsistencies as these three measures are extracted from three different runs performed with different binaries. This should not change the global picture as long as similar run times are observed for these three runs.

The MPI time (Global.3) is measured by Scalasca. But Scalasca will also measure MPIIO calls as part of the MPI time measurement, so this MPIIO time is substracted from MPI time during the metric extraction process.

The IO time (Global.2) is measured by Darshan. The IO time itself within in Darshan is separated into POSIX and MPIIO time. The POSIX IO handling is a subset of the MPIIO handling, so typically it would be enough just to use the MPIIO timings (if available) to represent the total IO time. Of course there are also applications which use MPIIO and POSIX file IO at the same time. In such a case the maximum of both will be selected to represent the IO time metric.

Memory vs Compute Bound metric (Global.4) is computed with the runtime coming out of two dedicated runs. The two runs use the same amount of MPI ranks and threads but on twice the number of nodes. This leads to depleted resources, and, by using specific deployments, one has the chance to observe memory bandwidth effects. Typically on current dual socket systems, a compact and a scatter run are performed. The compact run packs all the MPI processes and threads on a single socket, whereas the scatter run distributes them evenly on the two sockets. Going from the compact run to the scatter one, the available computing power is kept constant while doubling the available memory bandwidth. As a consequence, if both runs exhibit the same wall time, this means that the memory bandwidth available has no impact on the application. So the code is strongly compute bound and the ratio run time compact / run time scatter is 1.0. On the other hand, if the scatter run is twice as fast, the ratio is than 2.0 and this means that the code is strongly memory bound.

The load imbalance metric (Global.5) gives the potential for code improvement if the load imbalance would be perfectly fixed. Thanks to the trace analysis, Scalasca is able to compute the critical path of the application and the overhead due to load imbalances between ranks/threads. The metric used here is simply the ratio overhead / critical path. For instance, if a 20% load imbalance is measured, fixing perfectly this load imbalance would improve the performance of the code by 20%.

Synchro / Wait MPI (MPI.7) is calculated by gathering the communication overhead except the pure communication time. This metric sums up the average waiting time

[3] `http://www.scalasca.org/`

[4] `http://icl.cs.utk.edu/papi/`

[5] `http://slurm.schedmd.com/`

[6] `https://gitlab.maisondelasimulation.fr/dlecas/IdrMem`

		Metric name	Definition	Tool
Global	1	Total Time (s)	Total application wall time	*time*
	2	Time IO (s)	Average time spent in doing IO for each process	Darshan
	3	Time MPI (s)	Average time spent in MPI for each process	Scalasca
	4	Memory vs Compute Bound	1.0 means strongly compute bound, 2.0 means strongly memory bound	cf text
	5	Load Imbalance	Ratio of the load imbalance overhead towards the critical path duration	Scalasca
IO	1	IO Volume (MB)	Total amount of data read and written	Darshan
	2	Calls (nb)	Total number of IO calls	Darshan
	3	Throughput (MB/s)	IO.1 / Global.2	Computed
	4	Individual IO Access (kB)	IO.1 / IO.2	Computed
MPI	1	P2P Calls (nb)	Average number of peer to peer communications per MPI rank	Scalasca
	2	P2P Calls (s)	Average time spent in peer to peer communications per MPI rank	Scalasca
	3	P2P Message Size (kB)	Average message size in peer to peer communications per MPI rank	Scalasca
	4	Collective Calls (nb)	Average number of collective communications per MPI rank	Scalasca
	5	Collective Calls (s)	Average time spent in collective communications per MPI rank	Scalasca
	6	Collective Message Size (kB)	Average message size in collective communications per MPI rank	Scalasca
	7	Synchro / Wait MPI (s)	Average time spent in synchronization per MPI rank	Scalasca
	8	Ratio Synchro / Wait MPI	MPI.7 / Global.3	Computed
Node	1	Time OpenMP (s)	Time spent in OpenMP parallel region	Scalasca
	2	Ratio OpenMP	Ratio of the time spent in OpenMP parallel region towards the total calculation time	Scalasca
	3	Time Synchro / Wait OpenMP	Average time spent in synchronization/OpenMP overhead per thread	Scalasca
	4	Ratio Synchro / Wait OpenMP	Node.4 / Node.1	Computed
Mem	1	Memory Footprint	Average memory footprint of an MPI process	IdrMem/ Slurm
	2	Cache Usage Intensity	Cache Hit / (Cache Hit + miss) in Last Level Cache	PAPI
Core	1	IPC	Total number of instructions executed / Total number of cycles	PAPI
	2	Runtime without vectorization	Total application wall time compiled with vectorization disabled	*time*
	3	Vectorisation efficiency	Global.1 / Core.2	Computed
	4	Runtime without FMA	Total application wall time when compiled with FMA disabled	*time*
	5	FMA efficiency	Global.1 / Core.4	Computed

Table 3: Global performance metrics definition

per process (e.g. because of a MPI barrier operation) and the synchronisation time to start collective operations.

Metrics Mem.2 and Core.1 use the PAPI counter interface. The implementation of this interface and the available metrics are highly platform specific. Because of that not all applications might allow the extraction of these two metrics.

4.3 Towards an automated metrics extraction process

To make it easier for all code teams to carry out performance evaluation of their application themselves, without the need for detailed familiarisation of the tools, it was decided to strive for an automatic generation of as many metrics in table 3 as possible. Two codes, out of the first workshop - Metalwalls and Esias - use already a very extended automatisation process, which will be described in the following paragraphs. Also the other codes already included several profiling tools within a automated JUBE script. Section 5 gives an overview about the status of all codes involved so far.

The code team of Metalwalls has dedicated themselves to set up a comprehensive and well documented example of how this can be done. Thanks to very intensive collaborative efforts, such a process has been successfully implemented and has proven its value. The code team created scripts to extract the relevant metrics out of the different profiling tools result files and allow the integration of these metrics into the JUBE environment. Thus, it has the potential to serve as a best practice anchor for other code teams and can thereby strongly leverage the overall work within EoCoE, even more so since this achievement was reached very early, not even six months into the project.

Specifically, for the purpose of automation four separate code binaries are initially needed:

- Normal (bin)
- scalasca instrumented (scalasca)
- Normal plus ”no-vectorization” (bin-no-vec)
- Normal plus ”no-fma” (bin-no-fma)

Next, 8 runs are performed:

1. bin ⇒ reference run, only time and mem footprint is taken
2. bin + Darshan ⇒ IO metrics
3. scalasca profile run ⇒ CPU counters
4. scalasca trace analyse ⇒ Global, MPI, OMP
5. (bin-no-vec) ⇒ Core, vectorization efficiency
6. (bin-no-fma) ⇒ Core, FMA efficiency
7. bin compact run ⇒ mem vs comp. bound
8. bin scatter run ⇒ mem vs comp. bound

The generation of the binaries as well as the execution of all necessary runs has been automised by using the JUBE environment. Specific metrics as well as a full metric overview can be created with a single JUBE execution.

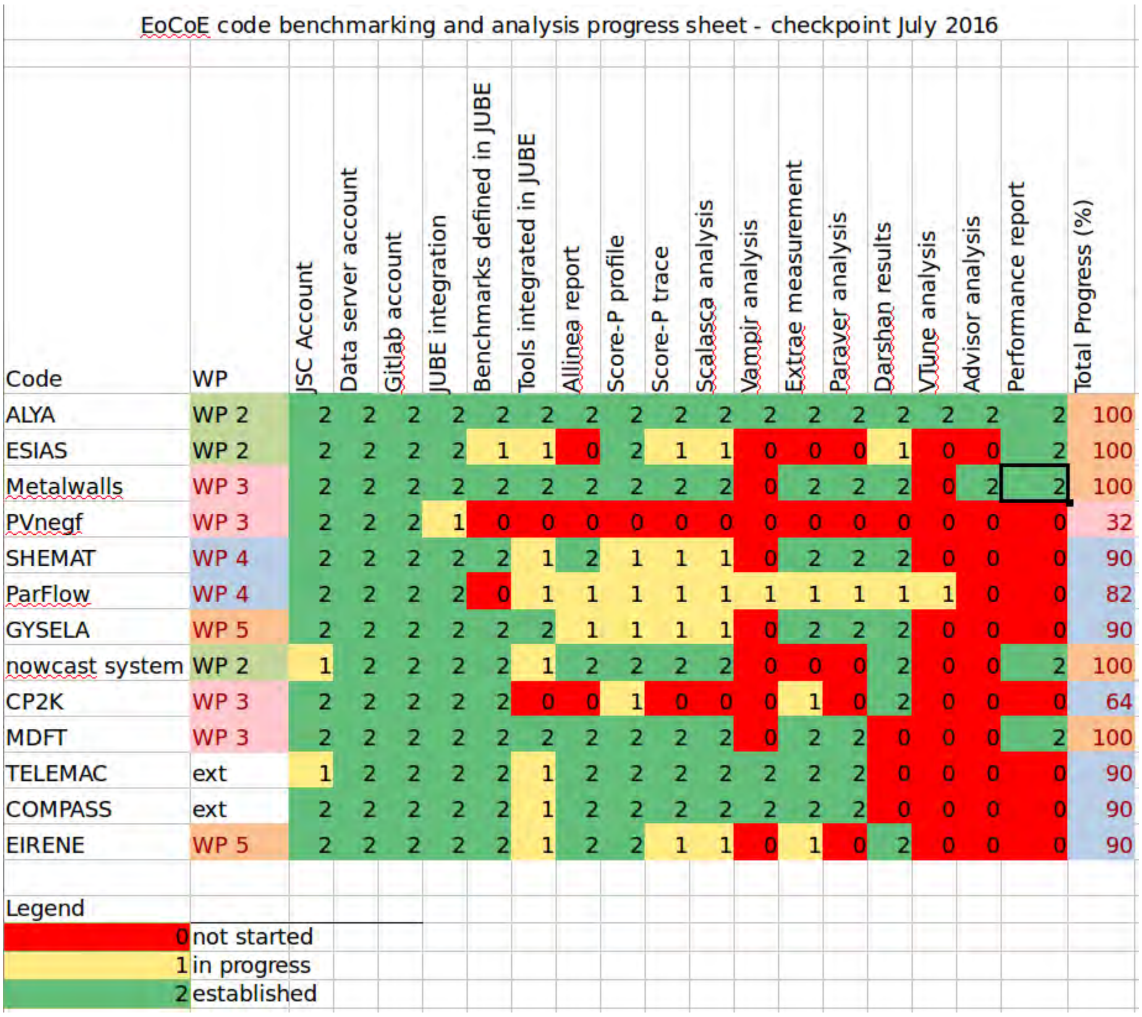
To proof the automation process, designed by the Metallwall code team, the Esias code team used the provided scripts and configuration techniques to automate their code in a similar manner on a different HPC system (JUQUEEN). The metrics provided by Scalasca, Darshan and the reference values could be easily included into the automated process and analyzed in a very short amount of time with the help of the Metallwalls configuration examples.

This procedure can now serve as a blueprint for other code teams and eventually of course also by the general public, via dissemination through WP 6. Within the project the relevant code examples were distributed via the Gitlab infrastructure. Table 4 and 5 in appendix shows the results of fully automated runs for Metalwalls and ESIAS.

5. Codes evaluated on the period Oct 2015 - July 2016

All codes mentioned in table 1 and 2 have established a close cooperation between HPC-experts and code developers following the above mentioned underlying lean management structure. They regularly update and report on their progress by means of the Code Diaries which are maintained on the Git structure along with code changes, automation processes and metrics.

Figure 3 shows the status of all codes regarding the implementation and analysation of the different profiling tools and of the benchmark automatisation process.



EoCoE code benchmarking and analysis progress sheet - checkpoint July 2016

Code	WP	JSC Account	Data server account	Gitlab account	JUBE integration	Benchmarks defined in JUBE	Tools integrated in JUBE	Allinea report	Score-P profile	Score-P trace	Scalasca analysis	Vampir analysis	Extrae measurement	Paraver analysis	Darshan results	VTune analysis	Advisor analysis	Performance report	Total Progress (%)
ALYA	WP 2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	100
ESIAS	WP 2	2	2	2	2	1	1	0	2	1	1	0	0	0	1	0	0	2	100
Metalwalls	WP 3	2	2	2	2	2	2	2	2	2	2	0	2	2	2	0	2	2	100
PVnegf	WP 3	2	2	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	32
SHEMAT	WP 4	2	2	2	2	2	1	2	1	1	1	0	2	2	2	0	0	0	90
ParFlow	WP 4	2	2	2	2	0	1	1	1	1	1	1	1	1	1	1	0	0	82
GYSELA	WP 5	2	2	2	2	2	2	1	1	1	1	0	2	2	2	0	0	0	90
nowcast system	WP 2	1	2	2	2	2	1	2	2	2	2	0	0	0	2	0	0	2	100
CP2K	WP 3	2	2	2	2	2	0	0	1	0	0	0	1	0	2	0	0	0	64
MDFT	WP 3	2	2	2	2	2	2	2	2	2	2	0	2	2	0	0	0	2	100
TELEMAC	ext	1	2	2	2	2	1	2	2	2	2	2	2	2	0	0	0	0	90
COMPASS	ext	2	2	2	2	2	1	2	2	2	2	2	2	2	0	0	0	0	90
EIRENE	WP 5	2	2	2	2	2	1	2	2	1	1	0	1	0	2	0	0	0	90

Legend

🟥 0	not started
🟨 1	in progress
🟩 2	established

Figure 3: Code benchmarking and analysis progress sheet

A. Performance evaluation reports

A.1 Metalwalls

Code ID card

Code name: Metalwalls
Scientific domain: WP3 Molecular dynamic
Description:

Metalwalls is a classical molecular dynamics code that simulates energy storage devices: supercapacitors. These devices could replace in the future the batteries used in nowadays hybrid vehicles.
Languages: Fortran90 (20k lines)
Library dependencies: MPI, OpenMP is in project.
Programing models: MPI, OpenMP is in project.
Platforms:

- PRACE Tier0 Mare Nostrum (20 MCPUh in 2016)
- French Tier1 Occigen (5 MCPUh in 2015)

Scalability results: It has been ported on X86 architectures, scaling results are good up to 1000 cores.
Typical production run: 24h on 64 - 512 cores
Input / Output requirement:

- Size: 10 GB / 24h run
- Single post-processing output: 50MB
- Single restart output: 50MB

Application references:

Merlet, C.; Rotenberg, B.; Madden, P. A.; Taberna, P.-L.; Simon, P.; Gogotsi, Y.; Salanne, M. Nature Materials. 2012, 11, 306–310
Contact:

- Mathieu Salanne (mathieu.salanne@upmc.fr)
- Matthieu Haefele (matthieu.haefele@maisondelasimulation.fr)

Metrics and performance report

Code team:

- Matthieu Haefele (MdlS) for WP1
- Mathieu Salanne (MdlS) for WP3

Case1 characteristics:

- Domain size: 3776 ions (walls + melt)

- Resources: 1 node on Jureca (24 cores)
- IO details: Checkpoint written every 10 steps instead of 1000 ⇒ much larger than production
- Type of run: both a development and small production run

	Metric name	03/01/2016
	Test-case	case1
Golbal	Total Time (s)	43.2
	Time IO (s)	0.3
	Time MPI (s)	12.4
	Memory vs Compute Bound	1.1
IO	IO Volume (MB)	35.8
	Calls (nb)	384000
	Throughput (MB/s)	105.0
	Individual IO Access (kB)	0.1
MPI	P2P Calls (nb)	0
	P2P Calls (s)	0.0
	Collective Calls (nb)	2721
	Collective Calls (s)	0.1
	Synchro / Wait MPI (s)	11.7
	Ratio Synchro / Wait MPI	94.8
	Message Size (kB)	908.4
	Load Imbalance MPI	24.8
Node	Ratio OpenMP	0.0
	Load Imbalance OpenMP	0.0
	Ratio Synchro / Wait OpenMP	0.0
Mem	Memory Footprint (B)	66 mB
	Cache Usage Intensity	N.A.
	RAM Avg Throughput (GB/s)	N.A.
Core	IPC	N.A.
	Runtime without vectorisation (s)	46.5
	Vectorisation efficiency	1.1
	Runtime without FMA (s)	44.6
	FMA efficiency	1.0

Table 4: Performance metrics for Metalwalls on the JURECA HPC system

According to Table 4, Metalwalls does not seem to need support on IO as less than 1% of execution time is spent in IO on a case that produces much more data than a production run. However, the IO metrics show a very large number of calls compared to the amount data written on disk and this is typical for such ASCII based outputs. The implementation of binary based outputs would help here but it is not a priority.

The 30% time spent in MPI is mostly due to load imbalance. The root of this imbalance could be spot thanks to the analysis of the scalasca trace. It resides in the *cgwallrealE* subroutine. The uniform distribution of atom pairs leads here to a load imbalance because some pairs require more computations than others. The implementation of an ad hoc load balancing scheme that would distribute the load between the MPI processes rather than the pairs could solve the issue and let the code scale much better.

Table 4 shows a poor vectorization efficiency. The trace obtained with scalasca allowed us to identify the most intensive parts of the code. A careful examination of these code regions on top of a very good compute bound indicator of 1.1 gives the feeling that the vectorization efficiency could be improved.

During this code investigation, we also noticed a discrepancy between the size of the data structures manipulated in the intensive regions and the global memory footprint measured on Table 4. This memory footprint is much larger than expected, some progress can certainly be made in this area.

Finally, the fact that Metalwalls is a pure MPI code can be a limitation on nowadays multi-core architectures and will definitely be one with the upcoming many-core architectures. An OpenMP implementation that could extract a fine grain parallelism could alleviate this limitation.

As a conclusion, in order to improve Metalwalls, we would recommend the following roadmap:

1. Single core optimizations would cure the memory footprint issue as well as the vectorization one.
2. An ad hoc load balancing scheme would allow the code to scale better in its pure MPI form.
3. An OpenMP implementation would prepare the code for the upcoming architectures.

A.2 Esias

Code ID card

Code name: ESIAS (Ensemble for Stochastic Integration of Atmopheric Simulations)
Scientific domain: WP2: Meteo4Energy
Description:

Coupled Ensemble implementation of Weather Research and Forecasting Model (WRF) and European Air Pollution and Dispersion Inverse Model (EURAD-IM) for short to medium range probabilistic forecasts and emission parameter estimation using Monte Carlo and Variational Data assimilation techniques. WRF is a state-of-the-art mesoscale numerical weather prediction system which is used extensively for research and operational real-time forecasting at numerous public research organizations and the private sector throughout the world and is open to the public. It offers various sophisticated physics and dynamics options. EURAD-IM is a fully adjoint chemistry transport model on the regional scale for chemical species and aerosols which is used for both, operational air quality forecasts and research applications. A main feature is the joint intital value and emission factor optimization using four dimensional variational data assimilation.
Languages: Fortran90 and C (500k lines)
Library dependencies: MPI, OpenMP, NetCDF, zlib, libpng, JasPer
Programing models: MPI, OpenMP
Platforms:

- IBM Blue Gene/Q JUQUEEN

Scalability results: It has been ported on X86 architectures, scaling results are good up to 524288 cores (512 each ensemble member).
Typical production run: 2h on 16384 - 32768 cores

Input / Output requirement:

- Size: 1 TB / 24h run (1000 ensemble members, 1 GB each)
- Single post-processing output: 10 GB (1000 ensemble members, 1 GB each)
- Single restart output: 100 TB (1000 ensemble members, 1 GB each)

Relevant kernel algorithms: Particle Filtering, 4DVAR, Quasi-Newton Minimization (LBFGS), FFT
Software licence: None
Application references:

W. C. Skamarock, J. B. Klemp, J. Dudhia et al., “A Description of the Advanced Research WRF Version 3”. NCAR Technical Note, NCAR, Boulder, Colo, USA, 2008.
Contact:

- Hendrik Elbern (h.elbern@fz-juelich.de)
- Jonas Berndt (j.berndt@fz-juelich.de)

Metrics and performance report

Code team:

- Sebastian Lührs (FZJ) for WP1
- Jonas Berndt (FZJ) for WP2

Case characteristics:

The benchmark setup contains a random simulation period of 6 hours with 240x240x24 gridpoints as a typical size. For benchmarking, solely 2 ensemble members run in parallel (instead of the order 1000 for production runs, would be too computational intensive for benchmarking). No particle filtering is performed due to the small ensemble size. 1024 Processors are used. Parallel NetCDF is used. The metrics results by using the Darshan and the Scalasca instrumentation are given in Table 5.

I/O and metadata handling can be a bottleneck when using larger numbers of ensemble members. This will be tested in additional benchmarks by using a higher number of ensemble members. Also the usage of the NetCDF4 instead of the pNetCDF library will be tested.

The single core performance can still be improved by using a higher compiler optimization level but options create stability problems, or will change the final result and has to be checked. Especially vectorization wasn’t successfully tested so far.

OpenMP can be used in WRF underneath the Esias ensemble creation, but currently the feature isn’t used. The performance benefit towards a full MPI parallelization will be tested.

	Metric name	out.json
Golbal	Total Time (s)	259.7
	Time IO (s)	27.2
	Time MPI (s)	178.5
	Memory vs Compute Bound	N.A.
IO	IO Volume (MB)	3570.9
	Calls (nb)	63594
	Throughput (MB/s)	131.3
	Individual IO Access (kB)	118.4
MPI	P2P Calls (nb)	135267
	P2P Calls (s)	8.1
	Collective Calls (nb)	6170
	Collective Calls (s)	1.1
	Synchro / Wait MPI (s)	98.4
	Ratio Synchro / Wait MPI	55.1
	Message Size (kB)	16.0
	Load Imbalance MPI	38.3
Node	Ratio OpenMP	N.A.
	Load Imbalance OpenMP	N.A.
	Ratio Synchro / Wait OpenMP	N.A.
Mem	Memory Footprint (B)	N.A.
	Cache Usage Intensity	N.A.
	RAM Avg Throughput (GB/s)	N.A.
Core	IPC	N.A.
	Runtime without vectorisation (s)	N.A.
	Vectorisation efficiency	N.A.
	Runtime without FMA (s)	N.A.
	FMA efficiency	N.A.

Table 5: Performance metrics for Esias on the JUQUEEN HPC system