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

D1.17 - M18
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:	D1.17 - M18
	Lead Beneficiary:	CEA
	Contact:	Matthieu Haefele
	Contact's e-mail:	matthieu.haefele@maisondelasimulation.fr
	Deliverable Nature:	Report
	Dissemination Level:	PU*
	Contractual Date of Delivery:	M18 31/03/2017
	Actual Date of Delivery:	M18 31/03/2017
	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 :	D1.17 - M18
	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

1 Document release note **5**

2 Motivation **5**

3 Joint EoCoE-PoP benchmarking workshops **6**

3.1 December 2015 in Juelich @ JSC 6

3.2 May 2016 in Saclay @ MdlS 7

4 EoCoE performance evaluation report and metrics definition **8**

4.1 Organizational structure and reporting 8

4.2 Metrics definition and performance tools 8

4.3 Automated metrics extraction process 11

5 Codes evaluated on the period Oct 2015 - March 2017 **13**

A Performance evaluation reports **14**

A.1 Metalwalls 14

A.2 Esias 18

A.3 Parflow 22

A.4 Gysela 25

A.5 Alya 28

List of Figures

1 Photo from the first workshop 6

2 Photo from the second workshop 7

3 Automated metric extraction process with JUBE 11

4 Steps of the metric extraction workflow. 12

5 Code benchmarking and analysis progress sheet 13

6 Screenshot of the VTune profiling of the original code 31

7 ALYA strong scaling 32

List of Tables

1 Codes participating in first EoCoE benchmarking workshop 7

2 Codes participating in the second EoCoE benchmarking workshop 8

3 Global performance metrics definition 10
4 Performance metrics for Metalwalls on the JURECA HPC system 15
5 Performance metrics for Esias on the JUQUEEN HPC system 19
6 Performance metrics for ParFlow on the JURECA HPC system 23
7 Performance metrics for Gysela on the JURECA HPC system (Small case) 26
8 Performance metrics for Alya on the JURECA HPC system 29
9 Detailed time performance on JURECA - January 30
10 Detailed time performance on JURECA - April 30

1. Document release note

This document replaces D1.16 Application Performance Evaluation that has been delivered on M12. 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 4 has been revised. The automated performance evaluation process has been improved, modularized and stabilized.
- Section 5 provides the updated table for all 13 codes evaluated to date.
- Section A now contains performance reports for the ESIAS and Metalwalls codes that have been obtained with the improved performance evaluation process.

In a nutshell, the two joint EoCoE/POP workshops triggered further cooperation between EoCoE code teams and POP. This corroborated the collaboration and mutual exchange between these two COEs

2. Motivation

As documented in the original proposal, the Energy oriented Center of Excellence (EoCoE) is a user driven consortium dedicated to tackling modelling challenges in the field of renewable energy. Consequently, the implementation and organization of the project places High Performance Computing (HPC) applications of the four chosen user communities at the heart of the project. Within in its transversal basis (WP1), the EoCoE project has gathered a comprehensive range of HPC expertise that aims to enhance the performance of these applications, 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 18 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.

Acknowledgement

In case of PSNC the scientific/academic work is co-financed from financial resources for science in the years 2016-2018 granted for the realization of the international project financed by Polish Ministry of Science and Higher Education (agreement number 3543/H2020/2016/2)

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 (CEA)	Latu (CEA)

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 during the first semester of 2017.

This event 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.

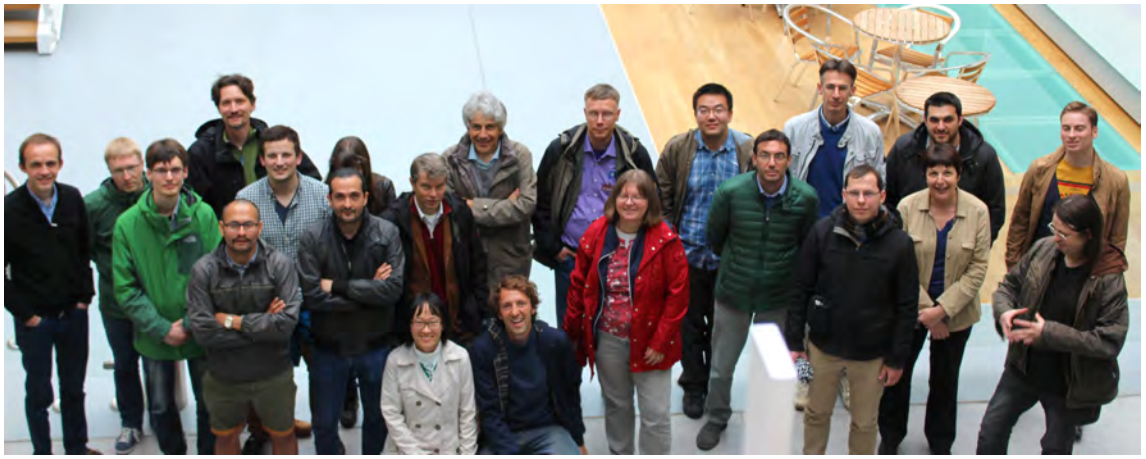


Figure 2: Workshop participants during the second benchmarking workshop

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 performance report for five codes: Metalwalls, ESIAS, Parflow, Gysela and Alya.

4.2 Metrics definition and performance tools

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

- The UNIX *time* command is used to measure total application wall time and the memory footprint of the first MPI rank of the application.
- 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
- IdrMem[5] 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 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 per process (e.g. because of a MPI barrier operation) and the synchronisation time to start collective operations.

[3] `http://www.scalasca.org/`
[4] `http://icl.cs.utk.edu/papi/`
[5] `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

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 Automated metrics extraction process

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

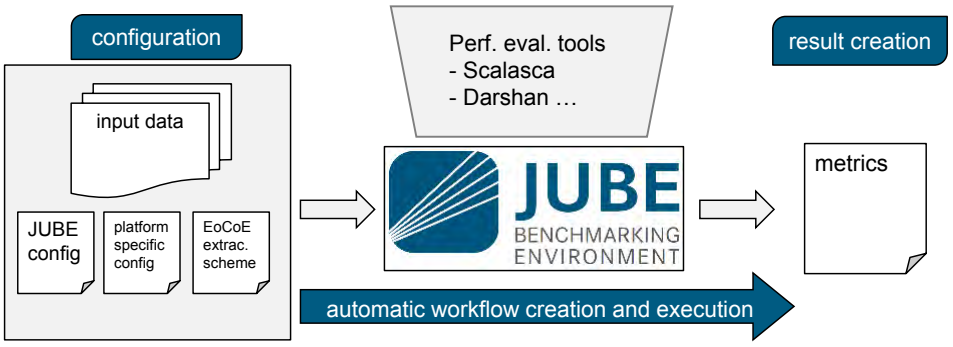


Figure 3: General JUBE workflow for the EoCoE metric extraction process.

Figure 3 shows the main workflow by using the JUBE environment. The application build and run procedure is included into a JUBE configuration file. This part is application specific. Platform specfic configuration datasets and the EoCoE specific execution scheme is added together with the relevant input data for the different benchmarking cases of the application. Within the JUBE environment, different runs are performed as written below. Different metric extraction tools like Scalasca and Darshan are called from within the JUBE environment. The final outcome of the execution is the set of metrics as shown in table 3.

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

- Normal (ref)
- scalasca instrumented (scalasca)
- Normal plus "no-vectorization" (no-vec)
- Normal plus "no-fma" (no-fma)

If needed a separate executable could be created for the Darshan or the memory instrumentation.

Next, 9 runs are performed:

1. ref ⇒ reference run
2. ref ⇒ memory footprint run

3. ref + Darshan ⇒ IO metrics
4. scalasca profile run ⇒ CPU counters
5. scalasca trace analyse ⇒ Global, MPI, OMP
6. (no-vec) ⇒ Core, vectorization efficiency
7. (no-fma) ⇒ Core, FMA efficiency
8. ref compact run ⇒ mem vs comp. bound
9. ref scatter run ⇒ mem vs comp. bound

The dependencies between the different runs are also shown in Figure 4.

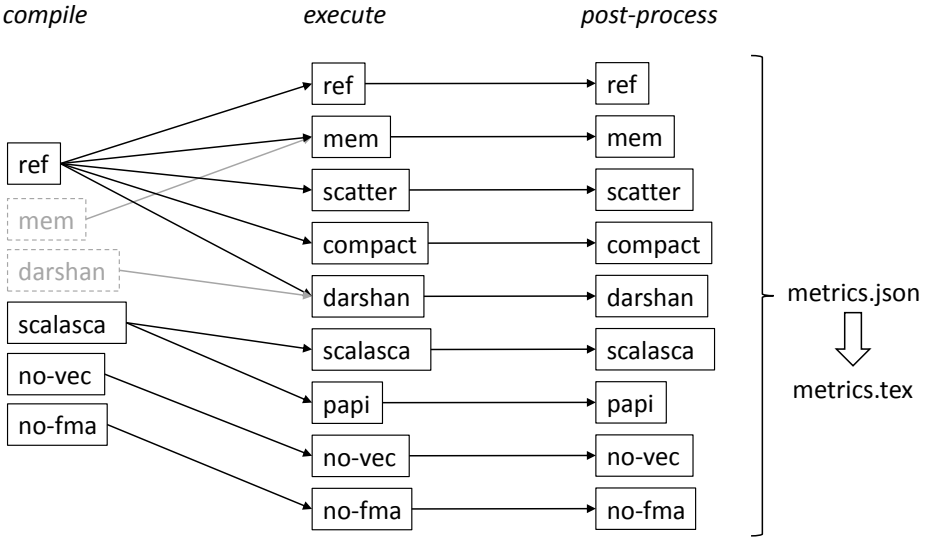


Figure 4: Steps in the automated JUBE workflow for the EoCoE metric extraction process.

All metrics paths could also be executed separately if needed.

A general EoCoE JUBE include file was created to cover these different runs to automatically build the underlying structure. This include file can be used in the application specific part of the metric extraction process, which avoids rewriting the structure multiple times. The file also covers the post-processing of the different tool output formats to create a parse-able final JSON file, which can be transformed into TeX table. To parse the different output formats, two Python scripts were created (mainly to parse the Scalasca and the Darshan output) which takes over the work to convert the binary formats into a ASCII based representation. These scripts are triggered automatically in the post-processing part of the JUBE run and can be used within all applications in the same way.

To allow to use this procedure as a blueprint for other code teams and eventually of course also by the general public, via dissemination through WP 6, a JUBE template for new codes was created which allows an easier adoption. Within the project the relevant code examples, templates and include files were distributed via the Gitlab infrastructure. The metrics tables in the appendix shows the results of fully automated runs using this architecture.

The automation allows a reproducible way to rerun the full metrics extraction scheme to track code changes and improvements during the application support phase. It can also be used by the code developers themselves within a testing setup to validate future development projects.

5. Codes evaluated on the period Oct 2015 - March 2017

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

Code	WP	JSC Account	Data server account	Gitlab account	JUBE integration	Benchmarks defined in	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	0	2	1	2	100
ESIAS	WP 2	2	2	2	2	2	2	0	2	2	2	0	0	0	2	0	0	2	100
Metalwalls	WP 3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	0	2	2	100
PVnegf	WP 3	2	2	2	2	2	2	2	2	2	2	1	0	0	2	0	0	0	90
SHEMAT	WP 4	2	2	2	2	2	1	2	1	1	1	0	2	2	2	0	0	1	90
ParFlow	WP 4	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	100
GYSELA	WP 5	2	2	2	2	2	2	1	1	1	1	0	2	2	2	0	0	2	100
nowcast system	WP 2	2	2	2	2	2	2	2	0	0	0	0	0	0	2	2	2	2	100
CP2K	WP 3	2	2	2	2	2	2	0	2	2	1	2	1	0	2	0	0	2	100
MDFT	WP 3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	0	2	100
TELEMAC	WP 4	2	2	2	2	2	1	2	1	1	1	1	1	1	0	0	0	0	86
COMPASS	ext	2	2	2	2	2	1	2	1	1	1	1	1	1	0	0	0	0	86
EIRENE	WP 5	2	2	2	2	2	2	2	2	1	1	0	1	0	2	0	0	0	90
Legend																			
0	not started																		
1	in progress																		
2	established																		

Figure 5: 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)

Performance metrics

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

Performance report

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)

Performance metrics

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. This benchmark was selected to allow Scalasca Trace analysis, which were not posssible (due to the size) with the 24 hour benchmark. The metrics results by using the Darshan and the Scalasca instrumentation are given in Table 5.

	Metric name	metrics_O2.json	metrics_O3.json
Global	Total Time (s)	259.46	199.71
	Time IO (s)	28.53	27.42
	Time MPI (s)	150.01	132.33
	Memory vs Compute Bound	N.A.	N.A.
	Load Imbalance (%)	31.03	31.36
IO	IO Volume (MB)	3570.93	3570.93
	Calls (nb)	63594	63594
	Throughput (MB/s)	125.16	130.24
	Individual IO Access (kB)	118.42	118.45
MPI	P2P Calls (nb)	135267	135267
	P2P Calls (s)	70.25	57.07
	P2P Calls Message Size (kB)	15	15
	Collective Calls (nb)	6170	6170
	Collective Calls (s)	21.93	18.35
	Coll. Calls Message Size (kB)	14	14
	Synchro / Wait MPI (s)	85.89	68.73
	Ratio Synchro / Wait MPI (%)	48.05	42.20
Node	Time OpenMP (s)	N.A.	N.A.
	Ratio OpenMP (%)	N.A.	N.A.
	Synchro / Wait OpenMP (s)	N.A.	N.A.
	Ratio Synchro / Wait OpenMP (%)	N.A.	N.A.
Mem	Memory Footprint	N.A.	N.A.
	Cache Usage Intensity	N.A.	N.A.
Core	IPC	N.A.	N.A.
	Runtime without vectorisation (s)	N.A.	N.A.
	Vectorisation efficiency	N.A.	N.A.
	Runtime without FMA (s)	N.A.	N.A.
	FMA efficiency	N.A.	N.A.

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

Performance report

I/O and metadata handling can be a bottleneck when using larger numbers of ensemble members. The Scalasca analyses highlighted these parts and the involved overhead. This will be tested in additional benchmarks by using a higher number of ensemble members.

The usage of the NetCDF4 instead of the pNetCDF library was tested but showed up much slower results, because the current implementation within the WRF backend uses only a serial filesystem access if NetCDF4 is activated.

Table 5 also highlights long waiting times within the MPI parts of the code.

The single core performance can still be improved by using a higher compiler optimization level but a direct change to `O3` create stability problems, or will change the final result and has to be checked. Especially vectorization wasn't successfully tested so far. Nevertheless the compiler settings on the BlueGene system could be optimzed by switching the default `O2` setting. This reduces the total execution time up to 25% as shown in Table 5 (current established compile setting on JUQUEEN: `-O3 -qnohot=noarraypad:level=2:novector:fastmath -qstrict=nolibrary -qdebug=recipf:forcesqrt -qsimd=noauto`).

OpenMP is available 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.

A.3 Parflow

Code ID card

Code name	ParFlow
Scientific domain	WP4: Environmental modelling (hydrology)
Description	ParFlow is a 3D variably saturated groundwater flow code with integrated overland flow and a land surface model and is used extensively as part of research on the water cycle in idealized and real data setups as part of process studies, forecasts, data assimilation frameworks, hind-cast as well as climate change projections from the plot-scale to the continent, ranging from days to years.
Languages	C (117k lines), Fortran90 (20k lines, the CLM land surface model)
Library dependencies	Silo (I/O), Hypre (preconditioner), KINSol (SUNDIALS, non-linear solver)
Programing models	MPI2
Platforms	• Tier0 JUQUEEN IBM BG/Q, JUGENE IBM BG/P, etc. • Tier1 JURECA, etc. • Tier0/1/2 Linux clusters in Europe and the US
Scalability results	It has been ported on x86_64 and BG/Q architectures, scaling results are good up to 32k tasks on BG/Q. See references given below.
Typical production run	Depends on experiment, from minutes up to months; continental model domains (e.g., CONUS on BG/Q on 16384 cores)
Input/Output requirement	Highly variable, depending on spatial resolution, simulation time span and output interval, 40 GB / output interval (Kollet et al., 2010)
Main bottleneck:	CPU
Relevant algorithms:	ParFlow simulates saturated and variably saturated subsurface flow in heterogeneous porous media in three spatial dimensions using a Newton-Krylov nonlinear solver and multigrid-preconditioners.
Software licence:	GNU LGPLi v3
Application references:	• S. F. Ashby, F. R. D., A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations, Nuclear Science and Engineering 124 (1996) 145–159. • J. E. Jones, C. S. Woodward, Newton–Krylov-multigrid solvers for large-scale, highly heterogeneous, variably saturated flow problems, Advances in Water Resources 24 (7) (2001) 763–774. doi:http://dx.doi.org/10.1016/S0309-1708(00)00075-0. • S. J. Kollet, R. M. Maxwell, Integrated surface-groundwater flow modeling: A free-surface overland flow boundary condition in a parallel groundwater flow model, Advances in Water Resources 29 (7) (2006) 945–958. doi:http://dx.doi.org/10.1016/j.advwatres.2005.08.006. • S. J. Kollet, R. M. Maxwell, Capturing the influence of groundwater dynamics on land surface processes using an integrated, distributed watershed model, Water Resources Research 44 (2) (2008) W02402. doi:10.1029/2007WR006004. • S. J. Kollet, R. M. Maxwell, C. S. Woodward, S. Smith, J. Vanderborght, H. Vereecken, C. Simmer, Proof of concept of regional scale hydrologic simulations at hydrologic resolution utilizing massively parallel computer resources, Water Resources Research 46 (4) (2010) W04201. doi:10.1029/2009WR008730. • R. M. Maxwell, L. E. Condon, S. J. Kollet, A high-resolution simulation of groundwater and surface water over most of the continental US with the integrated hydrologic model ParFlow v3, Geoscientific Model Development 8 (3) (2015) 923–937. doi:10.5194/gmd-8-923-2015.
Contact	Stefan KOLLET (stefan.kollet@fz-juelich.de)

Performance metrics

Code team:

- Wendy Sharples (FZJ) for WP1
- Stefan Kollet (FZJ) for WP4 (Carsten Burstedde, Jose Fonseca, Klaus Goergen, Ilya Zhukov, Ketan Kulkarni, Thomas Breuer, Bibi Naz, Jens-Henrik Goebbert, Lukas Poorthuis)

Case1 characteristics:

Domain size	50 x 50 x 40 regular grid
Resources	1 node on Jureca (24 cores)
IO details	Checkpoint written every 1 steps, ⇒ much larger than production
Type of run	development run

	Metric name	06/30/2016
	Test-case	case1
Golbal	Total Time (s)	4.05
	Time IO (s)	0.09
	Time MPI (s)	0.57
	Memory vs Compute Bound	NA
IO	IO Volume (MB)	183.11
	Calls (nb)	24002518
	Throughput (MB/s)	40
	Individual IO Access (kB)	NA
MPI	P2P Calls (nb)	10850
	P2P Calls (s)	0.14
	Collective Calls (nb)	2721
	Collective Calls (s)	0.01
	Synchro / Wait MPI (s)	0.796
	Ratio Synchro / Wait MPI	0.35
	Message Size (kB)	7.09
	Load Imbalance MPI	0.915
Node	Ratio OpenMP	0.0
	Load Imbalance OpenMP	0.0
	Ratio Synchro / Wait OpenMP	0.0
Mem	Memory Footprint (B)	23.1 mB
	Cache Usage Intensity	N.A.
	RAM Avg Throughput (GB/s)	0.008
Core	IPC	N.A.
	Runtime without vectorisation (s)	3.89
	Vectorisation efficiency	1
	Runtime without FMA (s)	3.83
	FMA efficiency	1.0

Table 6: Performance metrics for ParFlow on the JURECA HPC system

Performance report

ParFlow has undergone extensive performance analysis in addition to the performance metrics gathered (see PoP "ParFlow_POP_audit.pdf" report committed to the EoCoE ParFlow/docs repository).

ParFlow performance on a KNL cluster has also been assessed with a separate PoP report due at the end of this month.

According to Table 6, ParFlow does not seem to need support on IO as less than 1% of execution time is spent in IO on a case that produces as much data as production run. However binary files are not very portable compared to the standard climate science simulation file format, netCDF and thus much time is wasted in postprocessing-converting between binary to netCDF. In addition, there is a lot of postprocessing of data needed to turn output into scientifically valuable data, with the use of insitu visualization, these outputs could be generated interactively on the fly, further reducing postprocessing overheads.

It was determined that load imbalance was not an issue in this symmetric case upon analysis with scalasca (see PoP report) however in "real life" cases where much of the domain is inactive due to a land sea mask, adaptive mesh refinement would be desireable.

Memory footprint is an issue when scaling up to above 64,000 processors (on Juqueen-see PoP report), due to all cells having the COMPLETE grid information. Employment of an adaptive mesh refinement library would mean that each cell only stores neighbouring grid information, thus lowering the memory footprint.

Table 6 shows a fairly decent vectorization efficiency. Using Vector Advisor it was determined that nearly all loops that "could" be vectorised have many dependencies so it would take a huge amount of refactoring to get any better than this.

At the moment ParFlow is unable to take advantage of booster architecture. This is due to a heavy reliance on the solver library KINSOL. As KINSOL is tightly meshed with ParFlow at the moment this will take a considerable amount of refactoring.

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

1. Memory improvement and load imbalance would be improved by addition of adaptive mesh library
2. Booster architecture could be utilized once reliance on KINSOL is removed (E.g. PETSc)- first evaluate and quantify the benefits using a MiniApp
3. NetCDF IO would improve portability and reduce postprocessing overheads
4. Postprocessing overheads would be further reduced with insitu visualization

A.4 Gysela

Code ID card

Code name	Gysela
Scientific domain	WP5 Fusion
Description	The GYSELA code is a non-linear 5D global gyrokinetic full-f code which performs flux-driven simulations of ion temperature gradient driven turbulence (ITG) in the electrostatic limit with adiabatic electrons. No assumption on scale separation between equilibrium and perturbations is done.
Languages	Fortran 90 + some routines in C ($\approx$ 50 000 lines)
Library dependencies	MPI, OpenMP, HDF5
Programing models	MPI, OpenMP
Platforms	• Fusion dedicated international machines (Helios, Marconi) • French Tier1 (Occigen, Occigen2, Curie, Cobalt) Total core-hours consumed in 2016: 113.6 Mh
Scalability results	• Strong scaling: 60% relative efficiency at 65 kcores on Curie (x86) and Turing (BG/Q) • Weak scaling: 91% relative efficiency at 459 kcores on Juqueen (BG/Q)
Typical production run	200h on 4096 cores
Input / Output requirement	• Size: 400 GB / 24h run (restart files) • Single post-processing output: 100 GB • Single restart output: 200 GB
Main bottleneck	complex memory patterns, communication costs at very large scale
Relevant kernel algorithms	Semi-Lagragian scheme, cubic spline interpolation, FFT, 2D poisson solver
Software licence	CEA proprietary software
Application references	V. Grandgirard & al., A 5D gyrokinetic full- global semi-Lagrangian code for flux-driven ion turbulence simulations, Computer Physics Communications, Volume 207, October 2016, Pages 35-68, ISSN 0010-4655, http://dx.doi.org/10.1016/j.cpc.2016.05.007
Contact	• virginie.grandgirard@cea.fr • guillaume.latu@cea.fr

Performance metrics

Code team:

- Matthieu Haefele (MdlS) for WP1
- Guillaume Latu (CEA) for WP5

Small case characteristics:

Domain size	64 x 128 x 64 x 31 x 1
Resources	part of 1 node on JURECA (16 cores)
IO details	Checkpoint written every 4 steps instead of 100 ⇒ larger than production
Type of run	development run

Large case characteristics:

Domain size	512 x 256 x 128 x 60 x 32
Resources	43 nodes on JURECA (1024 cores)
IO details	Checkpoint written every 8 steps instead of 100 ⇒ larger than production
Type of run	production run

Table 7: Performance metrics for Gysela on the JURECA HPC system (Small case)

Performance report

GYSELA is a 5D gyrokinetic global code for simulating flux-driven plasma turbulence in a tokamak. The benchmark test case is based on a semi-Lagrangian scheme solving 5D gyrokinetic ion turbulence in tokamak plasmas. The GYSELA code is mainly written in Fortran90 and parallelised using both MPI and OpenMP. The code was built and run on the JURECA cluster with Scalasca/Score-P (profile and trace) measurements provided for examination. The code was built using Intel MPI 5.1 and Intel 15.0.3 compilers, and instrumented with Score-P 1.4.2 as part of Scalasca 2.2.2. Part of the information contained in this paragraph have been extracted from a report written by the PoP center of excellence (https://pop-coe.eu/).

Two execution traces were collected on JURECA each running 128 MPI processes with 8 OpenMP threads per process considering the `Large case`. One execution on 43 compute nodes had 3 MPI processes per node and therefore a dedicated core for each thread, whereas the other for comparison used hyperthreading with 6 MPI processes per node on 22 compute nodes. Program spent most of its time in two routines 80% in `blz_predcorr`, 15% in `diagnostics_compute`. Main equations (Vlasov and Poisson) are solved in `blz_predcorr` and post-processing of physical vaules and export on disk are done in `diagnostics_compute`. Most of the computations are tackled within OpenMP regions. MPI communications represents less than 2% of execution time inside `blz_predcorr`. For conventional production runs (number of cores is below 16 000 cores) the MPI overheads and MPI parallel imbalance are not an issue. We will not investigate here large configurations with high number of cores (32k and more) and will assume that MPI communication costs and parallel domain decomposition are not a major bottleneck.

80% of GYSELA total time in `blz_predcorr` is computation, 71% of which is in three OpenMP parallel regions with significant load imbalance. Work should be done to improve this, especially whenever hyperthreading is activated because it reinforces the imbalance. Furthermore, within `blz_predcorr`, 2D advection operator located in `advec2d_bsl.F90` shows specific problems: it is notable that the OpenMP synchronisation cost is particularly high

for half of the OpenMP threads for the MPI rank straddling the two processors on each compute node. This is due to the number of threads per MPI process chosen (8) that does not fit very well on a node that has 2 sockets of 12 cores. Something has to be done to avoid MPI processes straddling the 2 sockets.

Efficiency of vectorisation should be investigated. One can expect better speedup than a factor 2 with (31.2s) or without vectorisation (68s).

On large production runs, IO becomes an issue because checkpoint file size represents 100 gB up to 1 tB to be written down several times per run. HDF5 format is used up to now, but other strategy can be looked at in order to improve performance.

We have investigated the most intensive computation parts of the code with Paraver set of tools (www.bsc.es/paraver). These tools are based on traces capturing the detailed behavior of the different MPI processes and threads along time. Calls to the MPI and OpenMP runtime can be enriched with hardware counters, so we were able to measure the instructions and cycles for each computation region. In the next section we will show how the use of the Paraver tool helped to efficiently put into place simultaneous multi-threading in Gysela.

A.5 Alya

Code ID card

Code name	Alya
Scientific domain	Computational mechanics. In this project used for CFD for Wind energy
Description	The Alya System is the BSC simulation code for multi-physics problems, specifically designed to run efficiently in supercomputers. See web page: https://www.bsc.es/computer-applications/alya-system
Languages	Fortran90 (750k lines)
Library dependencies	Metis.
Programing models	MPI, OpenMP is in project.
Platforms	PRACE Tier0: Marenostrum, SuperMuc, Fermi, Juqueen, etc. Int: Blue Waters
Scalability results	It scalability has been tested in several Tier 0 European and international Supercomputers up to 130000 cores (SuperMuc).
Typical production run	12h on 128 - 512 cores
Input / Output requirement	• Size: 10 GB / 24h run • Single post-processing output: 500MB • Single restart output: 500MB
Main bottleneck	Memory access.
Relevant kernel algorithms	• Finite Element matrix calculation. • Iterative Solvers (GMRES, Deflated CG)
Software licence	It depends.
Application references	Alya: Towards Exascale for Engineering Simulation Codes', M. Vázquez, G. Houzeaux, S. Koric, A. Artigues, J. Aguado-Sierra, R. Arís, D. Mira, H. Calmet, F. Cucchietti, H. Owen, A. Taha, and J.M. Cela. The International Conference for HPC, Networking, Storage, and Analysis. http://arxiv.org/pdf/1404.4881v1.pdf
Contact	• Guillaume Houzeaux (guillaume.houzeaux@bsc.es) • Mariano Vazquez (mariano.vazquez@ bsc.es)

Performance metrics

Code team:

- Herbert Owen (BSC), WP2
- Guillaume Houzeaux (BSC), WP2

- Yacine Ould Rouis (MdlS), WP1

Benchmark characteristics:

Domain size	1 Million elements
Number of timesteps	30
Compile options	-O2 -xHost -DNDIMEPAR
Resources	1 node on Jureca (24 cores)
IO details	default sequential IOs, parallel hdf5 output is tested in a second step
Type of run	the size of benchmark aims to be faithful to the regular use of the program, in terms of number of elements per node

	Metric name	jan2016.json	apr2016.json
Golbal	Total Time (s)	385.4	346.3
	Time IO (s)	0.5	0.4
	Time MPI (s)	99.7	90.1
	Memory vs Compute Bound	1.3	1.3
IO	IO Volume (MB)	2449.9	2449.9
	Calls (nb)	97655	97573
	Throughput (MB/s)	5069.0	6423.6
	Individual IO Access (kB)	4.9	4.9
MPI	P2P Calls (nb)	154493	151985
	P2P Calls (s)	4.1	4.3
	Collective Calls (nb)	100071	98609
	Collective Calls (s)	0.7	0.8
	Synchro / Wait MPI (s)	94.2	84.9
	Ratio Synchro / Wait MPI	94.5	94.2
	Message Size (kB)	15.4	15.4
	Load Imbalance MPI	20.6	19.9
Node	Ratio OpenMP	N.A.	N.A.
	Load Imbalance OpenMP	N.A.	N.A.
	Ratio Synchro / Wait OpenMP	N.A.	N.A.
Mem	Memory Footprint (B)	584 mB	584 mB
	Cache Usage Intensity	N.A.	N.A.
	RAM Avg Throughput (GB/s)	N.A.	N.A.
Core	IPC	N.A.	N.A.
	Runtime without vectorisation (s)	383.2	362.9
	Vectorisation efficiency	1.0	1.0
	Runtime without FMA (s)	392.7	353.5
	FMA efficiency	1.0	1.0

Table 8: Performance metrics for Alya on the JURECA HPC system

Performance report

The Alya application submitted to EOCOE contains 2 major modules : NASTIN module, solving incompressible Navier Stokes equations and TURBUL module, solving turbulence equations.

It is pure MPI. Each module contains a matrix assembly part, that is perfectly distributed, and a solver part that requires communications at each iteration. The code has a master-slaves organization, with the rank 0 as master, and the rest as calculation

mode	CPU_time	Start_ops	NSI_total	NSI_mat	NSI_sol	TUR_total	TUR_mat	TUR_sol
ref	**384.66**	37.9	**203.57**	**67.24**	**132.27**	**125.46**	**87.87**	**32.21**
darshan	385.34	37.48	204.16	67.19	132.68	125.79	87.6	32.25
scatter	311.28	36.15	148.91	65.71	79.04	111.22	84.81	20.33
compact	396.5	35.89	207.43	68.42	134.05	130.53	88.75	36.35
memory	384.95	38.17	202.98	67.12	131.99	125.96	88.35	32.22
scalasca	**477.1**	49.96	213.28	76.08	133.24	187.04	149.02	32.97
no-fma	392.03	38.44	207.12	70.93	132.15	127.32	89.76	32.35
no-vec	**381.93**	38.94	**199.97**	60.92	134.94	125.83	85.25	34.95

Table 9: Detailed time performance on JURECA - January

mode	CPU_time	Start_ops	NSI_total	NSI_mat	NSI_sol	TUR_total	TUR_mat	TUR_sol
ref	**345.65**	37.79	**180.85**	43.85	130.4	108.29	**68.04**	31.61
darshan	346.04	37.16	181.72	44.04	130.37	109.66	67.95	31.6
scatter	279.39	35.97	131.09	43.37	79.23	96.62	66.27	20.32
compact	351.33	36.14	184.36	44.46	131.54	113.0	68.8	34.72
memory	348.77	38.4	182.63	44.04	130.92	108.88	67.65	31.9
scalasca	424.13	49.35	190.1	51.91	131.02	155.94	114.58	32.15
no-fma	352.59	37.86	185.82	47.89	130.43	110.44	69.1	31.66
no-vec	361.56	40.01	193.52	56.74	128.78	110.75	68.84	31.99

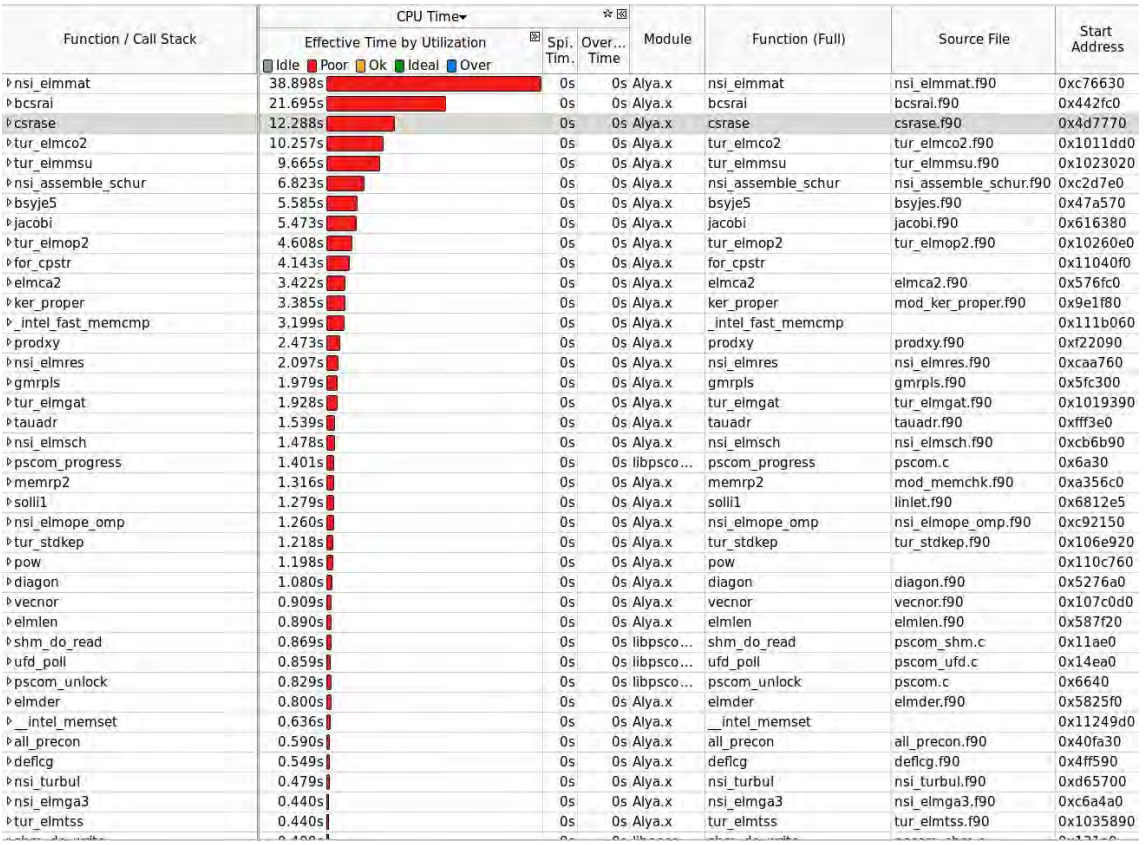
Table 10: Detailed time performance on JURECA - April

processes.

The first performance audit results in january allow us to make the following observations :

- Low memory consumption for this size of benchmark, compared to the memory of 1 node (<10%).
- The scatter vs compact results show a strong memory bound behavior, especially in the solver parts that run 40% faster in the scatter mode.
- The time measurements through direct instrumentation, show the following distribution in the different parts of the code (wall time, expressed in seconds and percentage of the total) :
  Total time : 385 s , 100 %
  - NASTIN module : 204 s , 52 %
    - matrix assembly : 67 s , 17 %
    - solver : 133 s , 34 %
  - TURBUL module : 125 s , 32 %
    - matrix assembly : 88 s , 23 %
    - solver : 32 s , 8 %
- The Scorep trace collection introduces a rather big overhead (20%), despite filtering a long list of subroutines. We can read in the perf eval table an MPI time representing 20% of the execution time, most of it due to synchronization. But a close look to both paraver and Scalasca traces show that MPI occupies only 5.8% of the calculation loop on the calculation processes, which concludes in a good balance and MPI performance. The rest of the 20% are spent in rank 0 (master

Figure 6: Screenshot of the VTune profiling of the original code



Function / Call Stack	CPU Time: Effective Time by Utilization (Idle Poor Ok Ideal Over)	Spi. Tim.	Over... Time	Module	Function (Full)	Source File	Start Address
nsi_elmmat	38.898s	0s	0s	Alya.x	nsi_elmmat	nsi_elmmat.f90	0xc76630
bcsrai	21.695s	0s	0s	Alya.x	bcsrai	bcsrai.f90	0x442fc0
csrase	12.288s	0s	0s	Alya.x	csrase	csrase.f90	0x4d7770
tur_elmco2	10.257s	0s	0s	Alya.x	tur_elmco2	tur_elmco2.f90	0x1011dd0
tur_elmmsu	9.665s	0s	0s	Alya.x	tur_elmmsu	tur_elmmsu.f90	0x1023020
nsi_assemble_schur	6.823s	0s	0s	Alya.x	nsi_assemble_schur	nsi_assemble_schur.f90	0xc2d7e0
bsyje5	5.585s	0s	0s	Alya.x	bsyje5	bsyjes.f90	0x47a570
jacobi	5.473s	0s	0s	Alya.x	jacobi	jacobi.f90	0x616380
tur_elmop2	4.608s	0s	0s	Alya.x	tur_elmop2	tur_elmop2.f90	0x10260e0
for_cpstr	4.143s	0s	0s	Alya.x	for_cpstr		0x11040f0
elmca2	3.422s	0s	0s	Alya.x	elmca2	elmca2.f90	0x576fc0
ker_proper	3.385s	0s	0s	Alya.x	ker_proper	mod_ker_proper.f90	0x9e1f80
_intel_fast_memcmp	3.199s	0s	0s	Alya.x	_intel_fast_memcmp		0x111b060
prodxy	2.473s	0s	0s	Alya.x	prodxy	prodxy.f90	0xf22090
nsi_elmres	2.097s	0s	0s	Alya.x	nsi_elmres	nsi_elmres.f90	0xcaa760
gmrpls	1.979s	0s	0s	Alya.x	gmrpls	gmrpls.f90	0x5fc300
tur_elmgat	1.928s	0s	0s	Alya.x	tur_elmgat	tur_elmgat.f90	0x1019390
tauadr	1.539s	0s	0s	Alya.x	tauadr	tauadr.f90	0xfff3e0
nsi_elmsch	1.478s	0s	0s	Alya.x	nsi_elmsch	nsi_elmsch.f90	0xcb6b90
pscom_progress	1.401s	0s	0s	libpsco...	pscom_progress	pscom.c	0x6a30
memrp2	1.316s	0s	0s	Alya.x	memrp2	mod_memchk.f90	0xa356c0
solli1	1.279s	0s	0s	Alya.x	solli1	linlet.f90	0x6812e5
nsi_elmope_omp	1.260s	0s	0s	Alya.x	nsi_elmope_omp	nsi_elmope_omp.f90	0xc92150
tur_stdkep	1.218s	0s	0s	Alya.x	tur_stdkep	tur_stdkep.f90	0x106e920
pow	1.198s	0s	0s	Alya.x	pow		0x110c760
diagon	1.080s	0s	0s	Alya.x	diagon	diagon.f90	0x5276a0
vecnor	0.909s	0s	0s	Alya.x	vecnor	vecnor.f90	0x107c0d0
elmlen	0.890s	0s	0s	Alya.x	elmlen	elmlen.f90	0x587f20
shm_do_read	0.869s	0s	0s	libpsco...	shm_do_read	pscom_shm.c	0x11ae0
ufd_poll	0.859s	0s	0s	libpsco...	ufd_poll	pscom_ufd.c	0x14ea0
pscom_unlock	0.829s	0s	0s	libpsco...	pscom_unlock	pscom.c	0x6640
elmder	0.800s	0s	0s	Alya.x	elmder	elmder.f90	0x5825f0
__intel_memset	0.636s	0s	0s	Alya.x	__intel_memset		0x11249d0
all_precon	0.590s	0s	0s	Alya.x	all_precon	all_precon.f90	0x40fa30
deflcg	0.549s	0s	0s	Alya.x	deflcg	deflcg.f90	0x4ff590
nsi_turbul	0.479s	0s	0s	Alya.x	nsi_turbul	nsi_turbul.f90	0xd65700
nsi_elmga3	0.440s	0s	0s	Alya.x	nsi_elmga3	nsi_elmga3.f90	0xc6a4a0
tur_elmtss	0.440s	0s	0s	Alya.x	tur_elmtss	tur_elmtss.f90	0x1035890

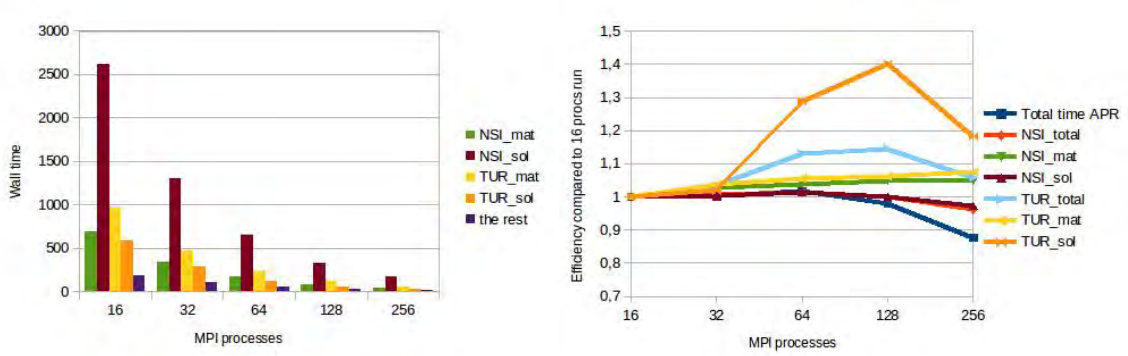
process) waiting for the calculation to be done (3.6%) and the basic serial IOs used in this benchmark (9.5% in the read and 1.8% in the write). However, these time losses should be put into perspective : The read time becomes less important for a production simulation length. In addition, a serial program allows to prepare very large data prior to the execution, so Alya can use a parallel read. Alya also has an HDF5 output option.

- Paraver analysis has shown a mean of 2 instructions per cycle in the matrix assembly parts. That denotes of a good performance. However the comparison between the ref and no-vec runs shows a very poor vectorization performance in these regions : especially in NASTIN matrix assembly that runs 10% faster when vectorization is disabled. This strongly suggests the necessity to improve the vectorization of this part.

- VTune hotspot profiling shows that 75% of the time is spent in the 12 hottest subroutines. These top 12 hottest are :
  - In Nastin matrix assembly : *nsi_elmmat, nsi_assemble_schur, jacobi, elmca2, ker_proper*
  - In Turbul mat assembly : *csrase, tur_elmco2, tur_elmmsu, tur_elmop2, ker_proper*
  - In solvers : *bcsrai, bsyje5*

- IO performance evaluation has been conducted later in a separate step, using HDF5 parallel output. The outcome, using darshan and wall time measurement, shows negligible output time in the regular use, even for large models (64Melem

Figure 7: ALYA strong scaling on 16, 128 and 1024 processes, on MareNostrum - Model size : 8Melem



on 64 nodes, generating 4.6Gb files). A regular use, according to Alya team, generates an output every 100 timesteps. In order to give an idea of IO time, Increasing the outputs frequency to every time step gives the following overheads on a weak scaling :

- 1Melem on 1 node (16 processes) : still under 1% overhead.
- 8Melem on 8 nodes (128 processes) : 8% overhead.
- 64Melem on 64 nodes (1024 processes) : 15% time overhead.

- Strong scaling results show a good scaling of the main parts of the program, as shown in figure 7.

In conclusion

1. No identified need of IO level optimization.
2. MPI performance judged good in the actual context and code version.
3. Matrix assemblies (40% of exec time) : pathologies identified and potential optimization possibilities on the sequential level : memory and cache accesses, vectorization, padding... The code holders expressed a big need on this point.
4. Solvers (42%) : pathologies identified on the sequential level, mainly memory access indirections and unpredictible loop boundaries. The problem may be solvable with a large data restructuring. An other choice, prefered by code holders, is to put efforts into the solver's method itself, within WP1 task 2 dedicated to linear algebra.