D5.1 First Report on Analysis
Version 1.0

Document Information

Contract Number	824080
Project Website	www.pop-coe.eu
Contractual Deadline	M18, June 2020
Dissemination Level	Public
Nature	Report
Author	Judit Gimenez (BSC)
Contributor(s)	
Reviewer	Brian Wylie (JSC)
Keywords	Performance assessments, Evolution and analysis, first-year report

Acknowledgements:
The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No "824080".

Change Log

Version	Author	Description of Change
V0.1	Judit Gimenez	Initial Draft
		(Final Change Log entries reserved for releases to the EC)
V0.2	Brian Wylie	Reviewed version
V0.3	Judit Gimenez	Revised and completed version
V0.4	Brian Wylie	Revised version
V1.0	Judit Gimenez	Final version

Table of Contents

Executive Summary

This deliverable reports on the services provided by the Performance Assessments Work Package (WP5) of the POP2 CoE project. The Performance Assessments Work Package is the framework for one of the main services provided by the POP Centre of Excellence with the goal to promote best practices for evaluating and diagnosing the performance of POP2 customers' parallel codes.

This deliverable describes the work done during the first half of the project and characterizes the cases analysed summarizing findings and recommendations provided to the customers. Some of the metrics are compared with the experience collected on the predecessor POP CoE project.

The annexes of the deliverable are a list of the services and the reports produced in the first half of the project. Due to confidentiality issues of some users, those annexes are not included in the public version of the deliverable.

1. Introduction

This deliverable summarizes the Performance Assessment services carried out during the first half of the project. These services are provided free of charge to developers and users of parallel codes with the goal to help them to identify the current bottlenecks and to promote the use of performance analysis as a best practice when running parallel codes.

After the end of the first POP project, some of the partners were able to continue providing services on a best-effort basis. The main goal was to complete the assigned studies that were either in progress or not yet started, but also to demonstrate our commitment with the Centre of Excellence. Despite during that period the volume of work was significantly reduced, we maintained the possibility to request new services, and when POP2 started on December 1st 2018, we already had 24 studies either in progress or ready to start.

As of May the 1st 2020 (at the time writing this deliverable), we have 113 assessment services, 89 studies originated since the start of the POP2 project. Most of the assessment services correspond to initial audits (92 studies) while 11 of them are follow-on assessments to extend the initial study working on the same code (or a revised version) with the same user (or colleagues from the same group). During this period 10 assessment studies have been cancelled, which were mainly studies from the original POP project that were delayed until the start of POP2 but there was also a training request through the web form.

In POP2, KPIs as well as milestones combine the studies from both WP5 and WP6 (Proof-of-Concept). And we use as reference the studies either completed or in progress because in the original POP CoE project it was identified as the best indicator to measure the CoE progress. In order to have a reference value to measure the progress of each work package independently, WP5 and WP6 agreed that initially around 75% of the studies would be Performance Assessments and the other 25% would correspond to Proof-of-Concept studies. This distribution was decided considering both the effort assigned to each work package and the average effort required by each type of study and is going to be re-evaluated at the end of PM18.

The milestone M2 scheduled for M12 (November 2019) had the goal to reach 50 studies completed or in progress with the contribution of both WP5 and WP6. This milestone was successfully reached 3 months in advance in August 2019 (M9), with the contribution of 46 assessment studies from work package 5. Next milestone is scheduled for M24 (November 2020) targeting 120 studies. While writing this deliverable, we have 83 assessment studies completed or in progress, which is 92% with the current distribution that corresponds to 90 Performance Assessments.

In the rest of this document, we use the term “executed” to refer to the studies either completed or in progress.

2. Performance Assessments Evolvement

This section describes the growth of the service requests and their status as well as their distribution within the partners that participate in this work package (all except TERATEC).

Figure 1 plots the evolution of the POP2 Performance Assessments during this first half of the project. To evaluate the number of studies with respect to the work plan, we include as reference a flat distribution (blue line).

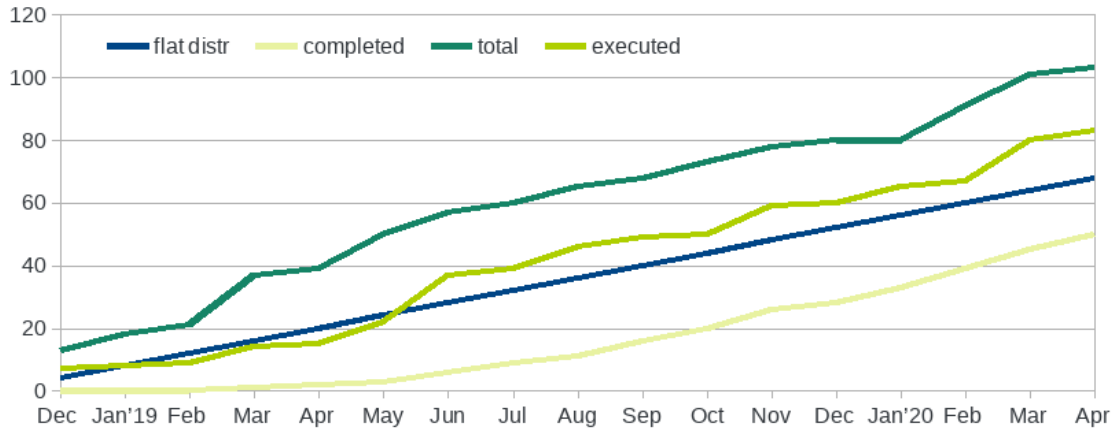

Figure 1: POP2 Performance assessments evolution

We can see that during the first 6 months of the project the number of studies executed was very close to the plan with very small deviations as for some months was lower than planned. Since June 2019, the number of studies executed is higher than the initial plan.

The figure also includes the total number of studies (not including the cancelled ones) as well as the studies completed. We can see that the number of studies completed was very low in the first months, when new analyst staff were being hired and trained, and that the last months are getting closer to the target of a flat distribution. On the other side, the total number of studies has been always significantly higher than the plan indicating that the current volume of new requests guarantees that there is enough work even when some studies are delayed, stopped by the users or even cancelled.

Figure 2 plots the distribution of the POP2 Performance Assessments with a more detailed classification of their state.

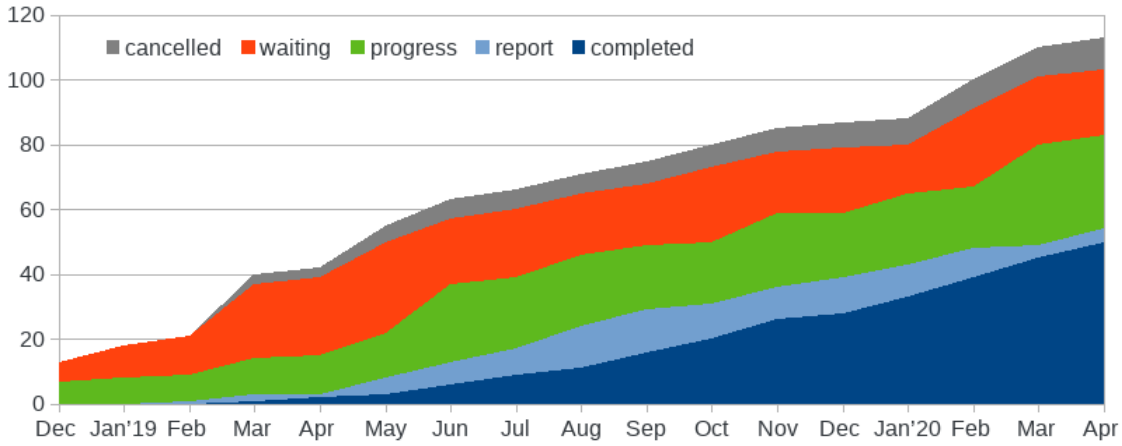

Figure 2: Distribution of the POP2 Performance assessments

In this plot we can also identify the executed studies that are in a reporting state, either writing the report or reporting the results to the customer, so very

close to completion. We can also see the evolution of the number of studies cancelled, verifying they represent a very small percentage of the requests. Finally, with respect to the studies that are waiting, they correspond in most cases to studies where the POP analyst is waiting for some input from the user (for instance providing input cases or access to the binary, or waiting that the customer collects the performance data).

Figure 3 plots the assessments distribution per partner using the previous states. As not all the partners have the same effort and budget we agreed on a weighted distribution to compute the target number of studies per partner, but to facilitate the comparison the plot is expressed as a percentage. We include in the plot the values for the whole consortium (labelled as *Total*). As this report is written very close to the half-way point of the project, with a uniform flat distribution the target would be to reach 50% with the studies completed, reporting and in progress. This means the orange part of the bar should start above 50%.

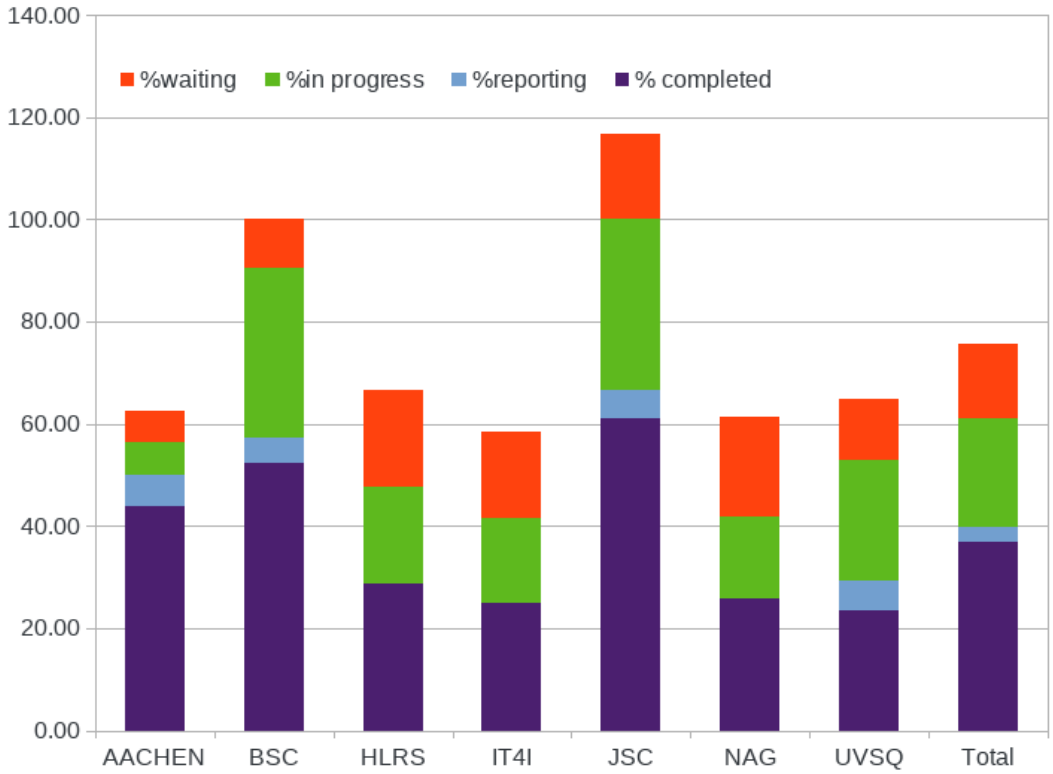

Figure 3: POP2 Performance assessments per partner

We can see that despite initially expecting a very similar bar for each of the partners, the differences are significant. Two of the partners reach the 50% goal with the completed studies. JSC has the higher bar and the completed studies are 61%, reaching 100% of the assignment considering only the assessments executed. Around 25% of the POP service requests for the JSC user community are charged to other projects and not to the POP2 project. BSC has completed 52% of the studies and the services executed are 90% of

the assignment. There are two partners that are a little bit higher than the flat distribution with respect to the target of executed services: AACHEN with 56% and UVSQ with 53%. HRLS, IT4I and NAG have values above the threshold but very close (48%, 42% and 42% respectively). Finally, the consortium as a whole has completed close to 40% of the studies and has executed 61% of the planned work in 50% of the time.

The typical situation is that the partner that establishes the contact with the user is also the one that does the assessment study, nevertheless close to 25% of the studies were offered to the consortium and the assignment was done considering the statistics of the partner and their availability.

As a last measurement of the assessment studies, Figure 4 plots the distribution of POP2 WP5 and WP6 studies to measure the ratio of studies that are extended after the initial audit. We also include as reference the total number of cancelled studies considering both work packages.

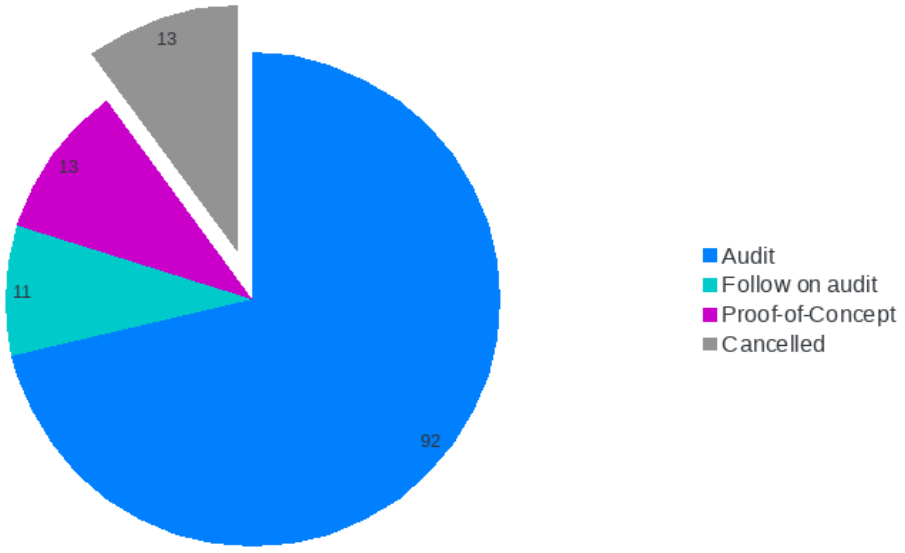

Figure 4: POP2 Studies (WP5+WP6)

We can see the continuation studies are very similarly distributed between Follow on Audits and Proof-of-Concept studies and globally indicate that 26% of the Audits are extended with a second service. Considering that the total number of studies completed when writing this deliverable is 50 and that for some of the codes the assessment reports a good performance where there is no need for improvement, the ratio of second services is around 50% and seems a very high percentage. Still, we have to take into account that this metric is not necessarily a very good indicator to measure the percentage of maintained collaborations as it is limited to cases where the second service is done for the same code-team and the same code.

3. Performance Assessments Analysis

In this section we characterise the assessments carried out in the reported period. The three axes considered for the analysis are: the user request, the code being assessed and the execution efficiencies measured in the study. For the two first characterizations we focus on the initial audits (to avoid that multiple studies for the same user and code generates a small deviation) and we include the cancelled studies when the data is available, except for the scaling where we include also the core-counts of the Follow on studies. For the characterization of the results we focus on the assessments to which the progress allows us to collect the data being analysed.

1 User request

The first aspect we analyse is the profile of the user with respect to the code. Figure 5 plots the distribution on the 3 roles we identify in the request form. Close to 80% of the users are core developers of the code, a profile that maximizes the potential impact of the assessment as they are in a good position to implement the assessment recommendations to improve their code. A sizable 14% of the requests come from users of a code that cannot implement modifications themselves, where they are nonetheless interested on identifying the bottlenecks and in many cases they want to share the report with the code owners. These percentages are very similar to the ones reported in the previous POP CoE project.

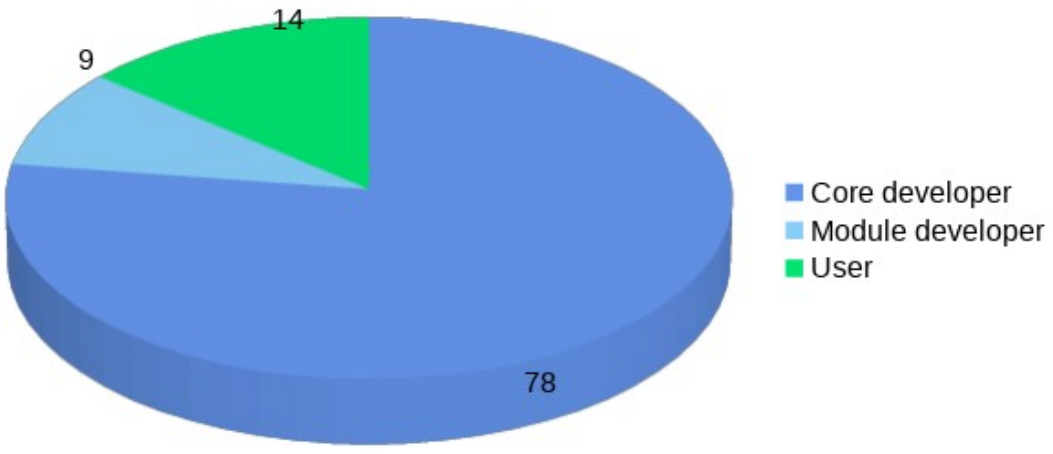

Figure 5: User role

A second aspect that we consider relevant to remark is the profile of the users with respect to their familiarity with performance tools. 71% of the users that replied to that question (62 answers) have no previous experience with performance tools, reflecting the need of support from an expert to assess the code performance.

The request form includes a question about the aspect the user considers is most relevant to focus on in the assessment. Figure 6 plots the classification of the service requests. Close to 75% of the users request a performance check, indicating they are interested in a global analysis of the code. Around 11% selects either to identify areas of improvements or analysing the efficiency of the parallel execution.

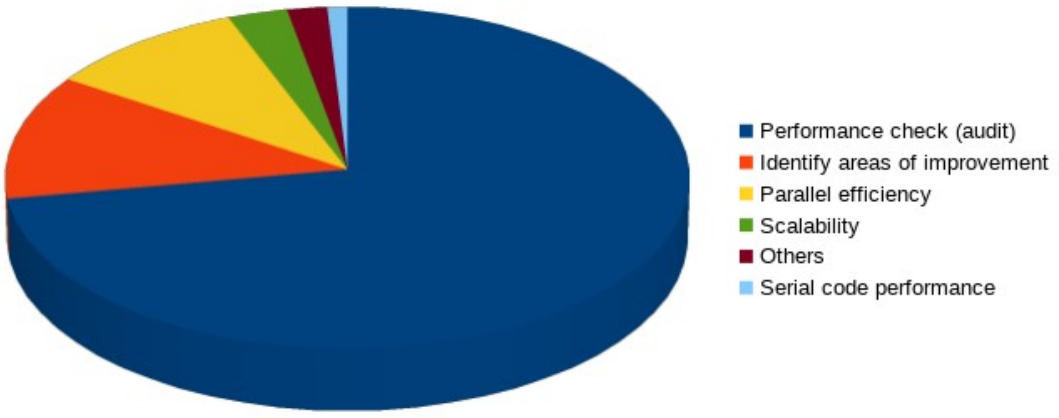

Figure 6: Service request

The last aspect we analyse for the user request is the answer to the question on how they found out about the POP project and services. Figure 7 plots the distribution of sources. 66% of the requests come after a direct contact with one of the POP partners. Word of mouth and project partners sum up close to 24%. Grouping the sources of social media, website, email and news the percentage is much lower (only 6%) indicating the need for a direct contact either from a POP partner or from some person that is already aware of POP CoE and its services.

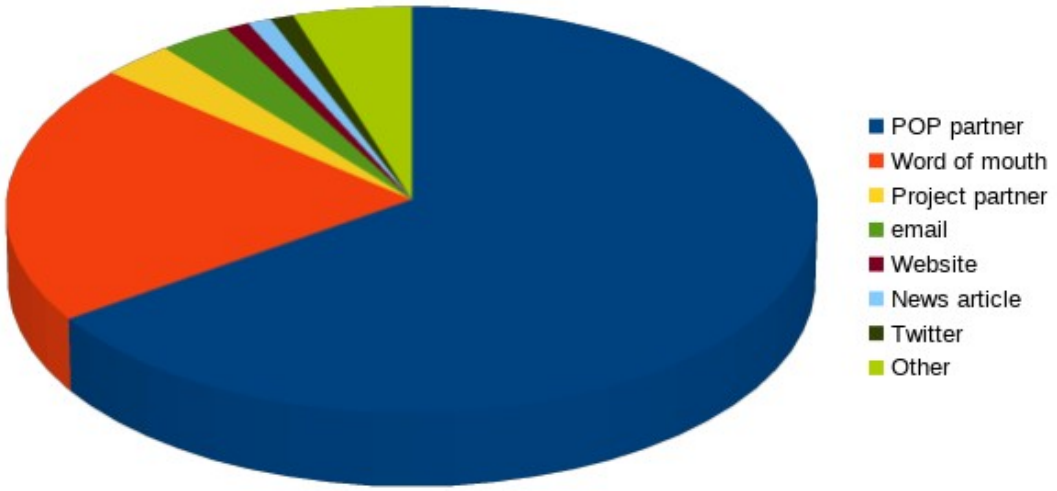

Figure 7: Source of contact

2 Code

The first characterization of the codes is based on the scientific/technical area as specified in the request form. Figure 8 plots the distribution over the different areas listed in the form. Close to 73% of the codes are almost equally distributed between the areas of *Earth*, *Engineering* and *Physics*. All other areas (except *Others*) represent less than 5%. We are going to refine the list of areas with the goal to better characterize the codes that are currently grouped as *Others*.

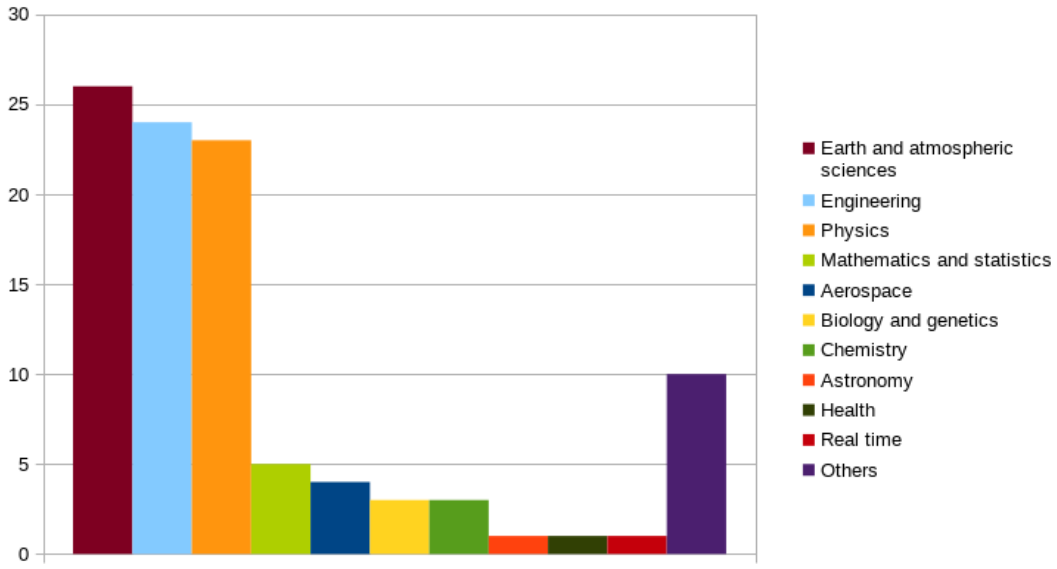

Figure 8: Code scientific/technical area

Comparing the results with the initial POP CoE project, there is a very significant increase on the number of *Earth Science* codes and a reduction of the *Chemistry* codes. But still the sum of those four sectors represents a similar percentage over the total of codes (77%).

One of the optional questions we ask of the users is if the code was analysed before their request. From 62% answers, less than 20% of the codes were previously analysed. That may mean that a high percentage of the owners of codes previously analysed do not request POP2 services maybe because they consider they do not need to use POP services or, in general, that they consider they do not need to periodically analyse the code. But it also identifies a large number of codes that have never been analysed and where the POP service may play a relevant role.

Figure 9 plots the profile of the code with respect to its mode of execution. This is also an optional question where we got replies only in 45% of the cases. More than 75% of the codes run as standalone, but 15% of the codes are usually executed as part of a coupled application (which is a very frequent scenario for Earth Science codes).

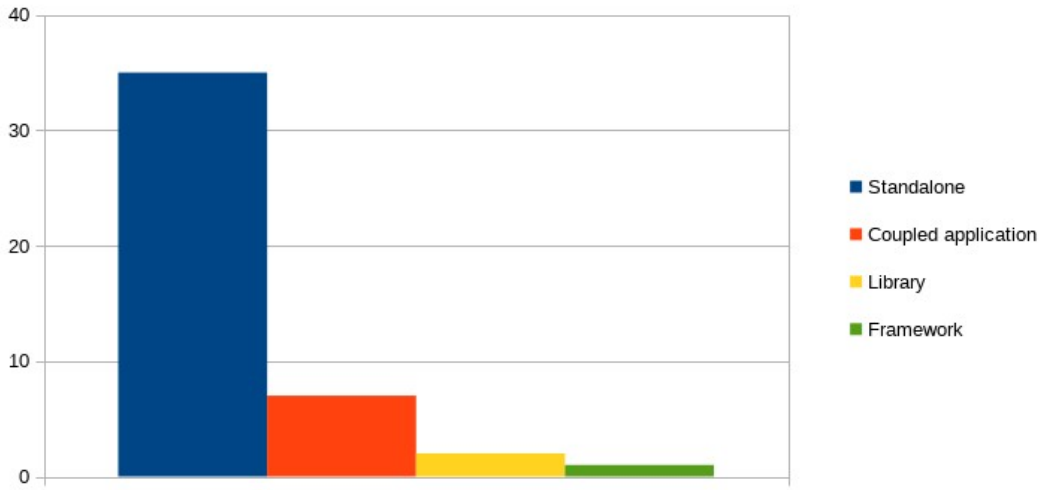

Figure 9: Code profile

To characterize the potential distribution of the code and the impact of improving them, Figure 10 plots the code license. With a population of 44 answers, more than 60% of the codes are distributed with a free license or no license, and only 20% have a commercial license. 16% of the codes are limited to internal use.

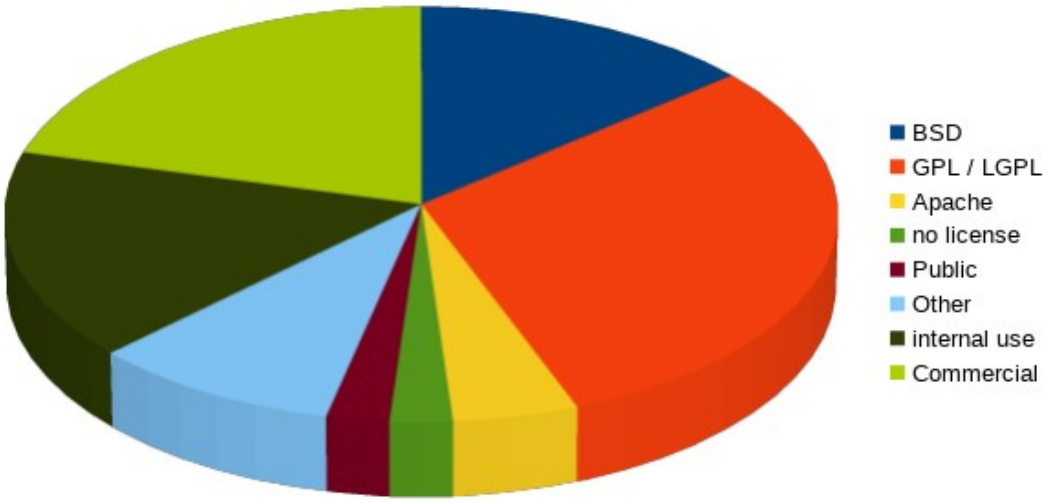

Figure 10: Code license

2.1 Code Programming

The following plots target to characterize the code with respect to the programming model and the programming language. We need to clarify that users specify as programming model all the existing variants of the code and that in many cases the analysis is focused to the version most frequently used. For instance, typically codes have a version that can run on accelerators and that may be still under development so the user is not interested to analyse this part of the code.

Figure 11 plots the parallel programing models used by the codes. 85% of the codes use MPI to be able to run in distributed memory architectures. Nevertheless, only 45% are pure MPI codes and 40% also support multithreading and/or kernel offload to accelerators. A total of 46% of the codes have support for threads, mainly through OpenMP but also using POSIX threads or even both OpenMP and POSIX threads concurrently. Finally, only 16% of the codes have support to run on accelerators which is typically combined with MPI. Despite TBB is a multithreading programming model, we did not include them on the *threads* because neither JSC nor BSC tools support that programming model.

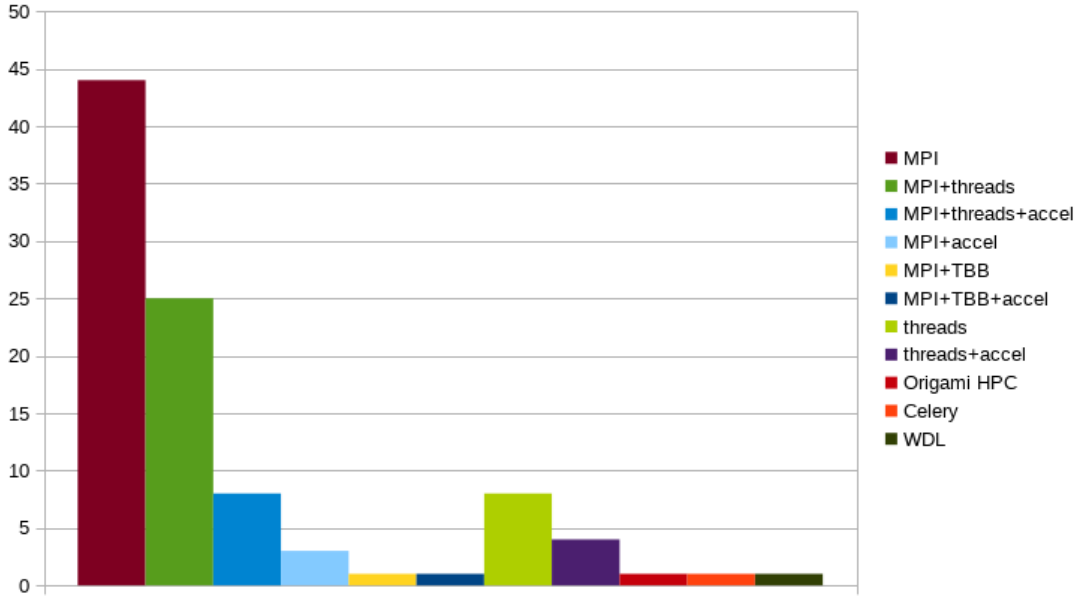

Figure 11: Parallel programming model

Comparing with the statistics collected during POP CoE project, the percentage of *MPI+threads* have been reduced while the percentage of codes that support accelerators is very similar.

To check if there is a correlation between the scientific area and the programing model used, Figure 12 plots the most frequent programming model per area (discarding areas or programming models with just one code). We can see that while pure MPI codes apply in all the sectors, the pure share memory approach only appears in the traditional sectors (*Physics*, *Engineering* and *Earth*). Hybrid approaches apply to all sectors except *Aerospace* and *Chemistry*.

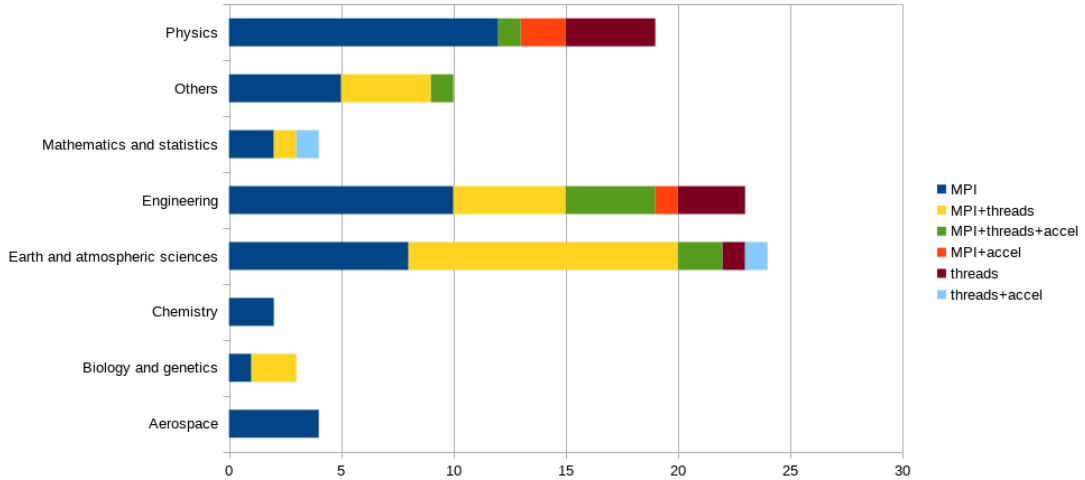

Figure 12: Programming model per scientific area

The distribution with respect to the programming language is plotted in Figure 13. Pure C++ codes have the higher percentage with 33% of the codes, while pure Fortran codes represents 23%. C++ or C is used in 71% of the codes while Fortran contributes in 42%. Finally, 22% of the codes have Python and pure Python codes are 5%.

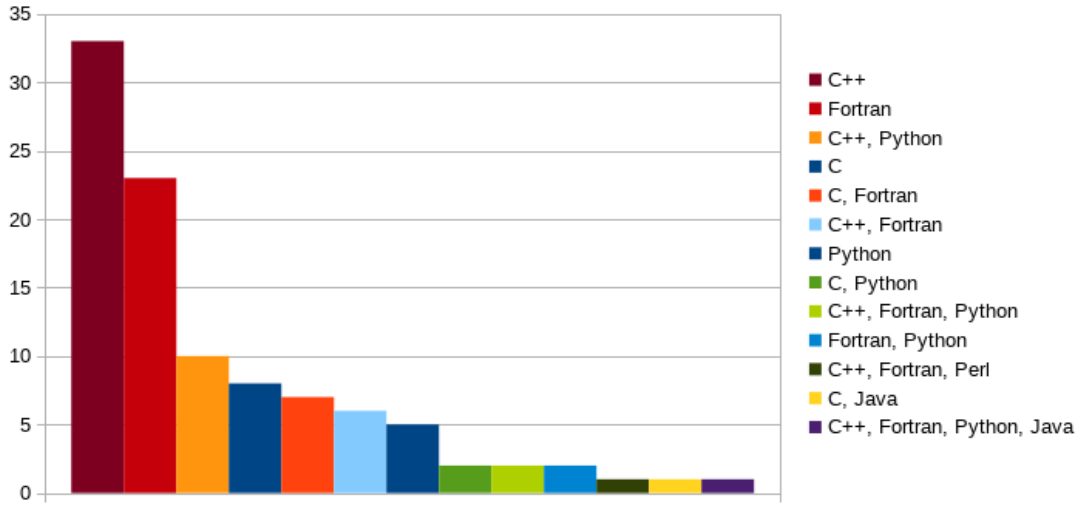

Figure 13: Programming language

Looking at the distribution per scientific area, pure C++ is spread over most of the sectors while pure Fortran code is concentrated in the 3 scientific areas with more codes (*Earth*, *Engineering* and *Physics*).

Comparing the obtained values with the metrics measured at the end of the predecessor POP CoE project, pure C++ and pure Fortran codes have swapped their scores and the percentage of codes that use Python have significantly increased.

2.2 Code Scaling

We collect four measures of the scaling of the codes. Three of them are collected through the form where the user has to specify the number of cores that are typically used in production runs and in development runs, as well as up to which number of cores he/she is satisfied with the performance achieved. The fourth metric corresponds to the largest run that has been analysed in the assessment. It is important to remark that is the user who determines the scale that has to be analysed. Figure 14 compares these four metrics classifying the values with respect to their order of magnitude.

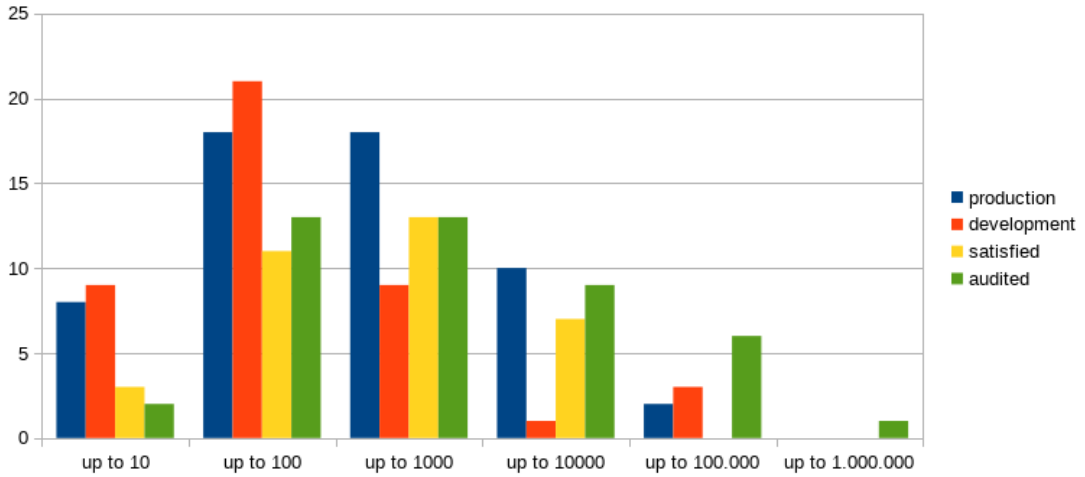

Figure 14: Scales comparison

The production runs are frequently in the range of a few thousand cores while the development runs typically use one order of magnitude fewer resources. Both the scale the user is satisfied with the performance and the maximum scale analysed have a similar frequent range between 100 and 10.000 cores. We think it is interesting to remark the codes that were analysed with a range of cores between 10.000 and 1.000.000 cores (with 309696 being the largest case analysed), bigger than their range in production. That indicates studies that were used to validate the performance at a larger scale that the one typically used.

To better compare the four scale metrics, the next two figures plot the metrics only for the 11 cases where we have the four values. Figure 15 plots the metrics on a linear scale, and as a couple of values are much larger than the rest, Figure 16 plots the same values in logarithmic scale. Analysing these figures we can identify very different approaches. The most remarkable case may be study #5 where the assessment has been done with a much bigger scale than the scale they currently use in production runs. This same approach can be seen in studies #1, #3, #4 and #8 despite with a smaller difference. It verifies that when more resources are available the codes are able to run correctly at larger scale, and potentially get faster results, though not necessarily efficiently. On the other extreme, in studies #2. #6 and #11, the assessment has been done with a scale smaller than the scale user is satisfied with the performance. Finally, a third approach is reflected in study #9 where the assessment has been done in a scale smaller than the scale used in production but bigger than the scale where the user is satisfied.

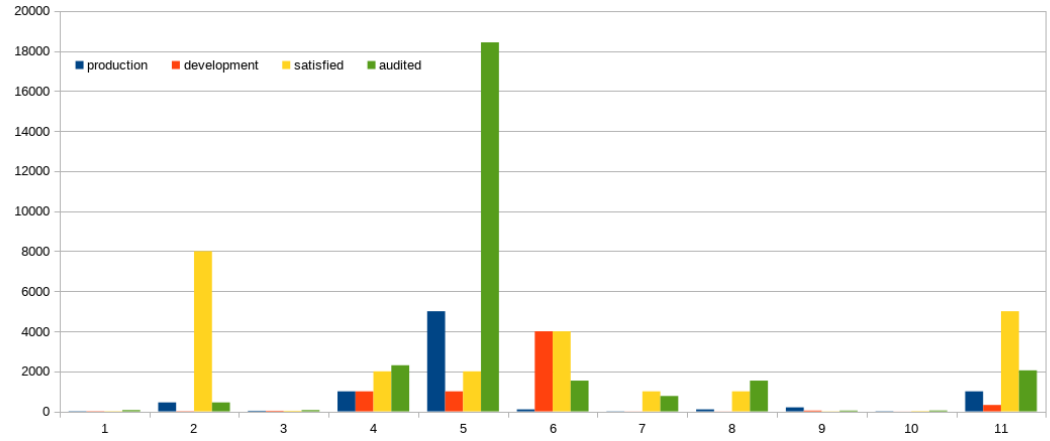

Figure 15: Scaling metrics

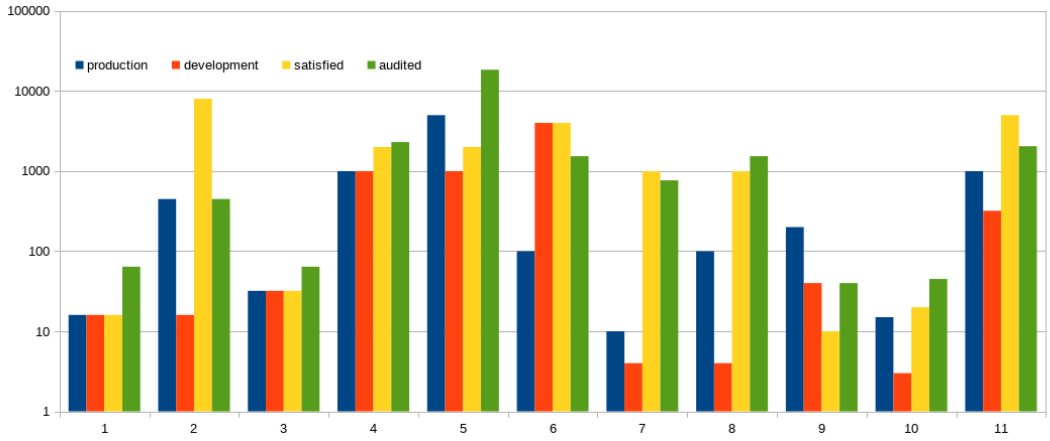

Figure 16: Scaling metric (logarithmic scale)

To provide a more detailed view of the scales analysed, Figure 17 plots the largest core count used on each of the codes. The scale of the plot has been truncated to 20.000 and the light blue bar that reaches the maximum corresponds to the largest execution with 309.696 cores. We can see the value for 9 codes that were audited with more than 5.000 cores. On the other extreme, we should notice that many of the codes that are audited with up to 100 cores were either pure thread based codes or applications that use accelerators.

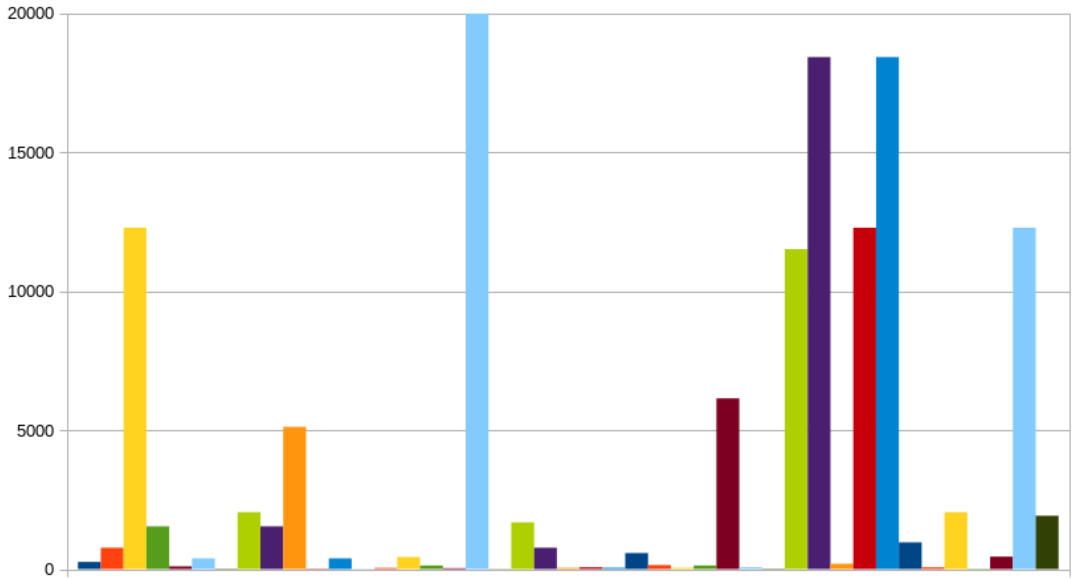

Figure 17: Largest audited run

One of the goals that it is common to all the HPC CoEs is to move toward exascale. We have contacted all the CoEs to offer to them assessment campaigns to audit their codes. The two options offered were periodic assessments of their codes and/or workshops to introduce them to the tools and to support them applying to their codes. The ChEESE CoE responded to an initial POP campaign and had ten of their codes assessed, whereas other CoEs (ESiWACE, CompBioMed and EXCELLERAT) have requested assessments of codes on a more ad hoc basis. EoCoE-II preferred to continue with the workshops initiative started in the framework of the initial POP and EoCoE projects. E-CAM also requested our participation at one of their training events.

To measure the impact of other CoEs in the scale of POP2 studies, we compared the average and maximum core counts grouping the studies in two sets: the assessments to CoEs codes and all the other studies. Figure 18: Comparing the scale of CoEs studies (logarithmic) plots these values (in logarithmic scale) confirming that the assessments to CoE codes have a significantly larger scale. The average number of cores is close to 18x with respect to the requests that are not from the CoEs, and the maximum value is 25x bigger when we focus on the CoEs assessments.

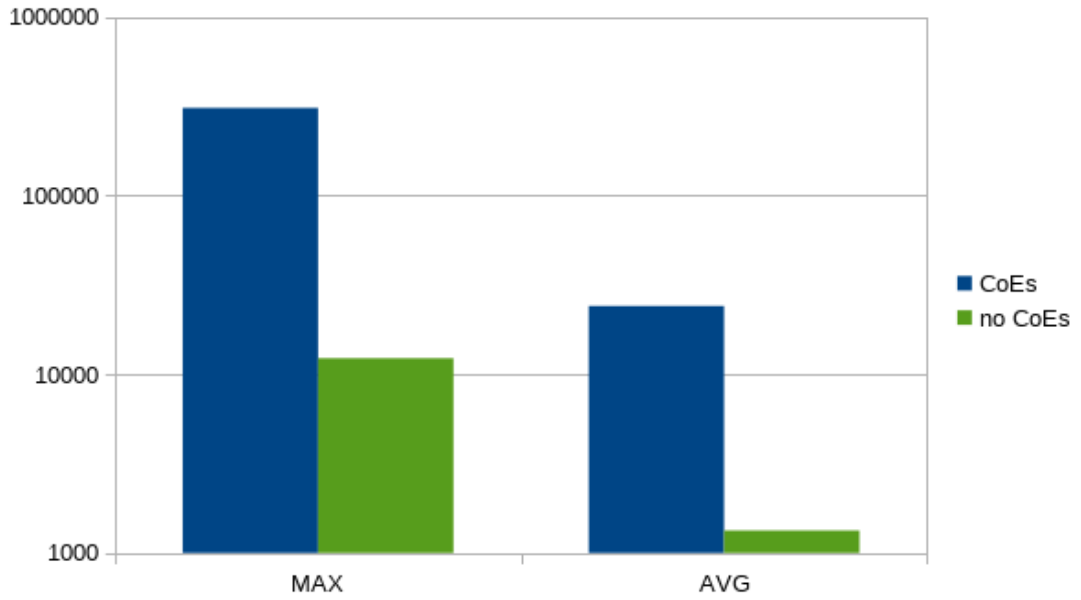

Figure 18: Comparing the scale of CoEs studies (logarithmic)

Comparing these values with the data collected in the precursor POP CoE project, the average scale is 4x larger, while the largest scale reported within POP was 239,615 cores.

To complement the analysis of the assessments scale we should consider the answer to other related questions. Covering if runs are usually single executions or many instances concurrently, 48% of the responses usually run multiple executions at the same time (42 answers). The need to execute many instances suggests reducing the scale of each run to fit on the available resources. With respect to the platform used, 60% of the codes are typically executed on the user local system that may also limit the maximum scale they can target for their production runs. Finally, 76% of the 55 answers use strong scaling mode where the problem size is maintained when increasing the number of resources, and that approach also influences the scale as the range of cores that make sense to use would be determined and limited by the input.

3 Efficiencies

As it was done in the predecessor POP CoE project, the first step of the performance assessment is to identify the structure of the application and to determine the focus of analysis (FoA), that is the relevant region(s) to focus the analysis on, discarding for instance initialization and finalization phases. After selecting the FoA, we use an efficiency model to determine the loss of performance on a small set of key factors.

The efficiency model is a hierarchy of multiplicative factors and it can be split in two main components: the contribution from the parallelisation itself (based on the time spent in the parallel runtime and its distribution among

processes/threads) and the scaling of the computations (supporting both weak and strong scaling approaches). While the parallel efficiency can be measured independently for each execution, the scaling of the computations requires measuring the application with at least two different core-counts.

The efficiencies are measured as a value between 0 and 1, the higher the better, which can be also expressed as a percentage from 0% to 100%. An efficiency value of 80% on a given factor means that the code is losing 20% of the maximum performance that the factor can achieve.

Generally, inefficiencies identified in smaller scale executions tend to grow as scales get larger. The efficiency analysis at different scales allows identifying both the factor(s) that limits the scaling as well as the factors that reduce the performance at all scales.

There is a lot of heterogeneity in the collected efficiency data as scales are very arbitrary across assessments. The results are reflecting the executions the users were interested more than the real performance that can be achieved by the codes.

To analyse the efficiencies collected, we considered five categories listed from worst to best:

- *Bad*. This category corresponds to values up to 60%. When a given factor has such a low value of efficiency it is indicating severe performance problems. The executions should be run on a lower scale or with a larger input case, because the resources are being underutilized.

- *Poor* category groups values between 60% and 75%. The loss of performance is still high. Code optimizations must ensure targeting to improve the factors that are in this range.

- *Fair* is a category with efficiencies above 75% but not higher than 85%. In this case, despite the value for the efficiency starts to be acceptable, the factor indicates that there is still place for improvement.

- *Good* category groups the efficiencies in the range between 85% and 100%. Despite the assessment may still identify potential improvements, the analysed execution achieves a commendable performance with respect to that key factor.

- *Super* is the last category for percentages higher than 100%. These values can only be seen in the computation scaling and its components. The most frequent scenario is related with IPC improvements when increasing the scale. That is typically the case of strong scaling mode as the work per process is reduced and it may improve the use of the cache or reduce the required memory bandwidth.

Figure 19 plots the distribution over these categories for the main efficiency factors we measured in the studies at their largest scale (which has previously

been shown to vary dramatically). Transfer and serialization efficiency factors have fewer occurrences reflecting the cases where they cannot be distinguished from communication efficiency (purely multithreaded codes, hybrid codes or codes that use accelerators). There are also cases where tracing (required to distinguish them) wasn't done, either because it wasn't possible (or practical) to do so and/or worth the extra effort/cost because the communication efficiency from profiles was over 95%.

As it can be expected, global efficiency is the metric with worst values because it accumulates the loss from all other efficiencies. We identify a significant number of codes with *Poor* or *Bad* values for global efficiency and parallel efficiency. We also confirm that the *Super* category only appears on the scaling metrics. Comparing between key factors we can infer that the main loss of performance is related with the parallelization and it is primarily related with global load balance and data transfer with a much lower impact of the serialization component.

Focusing on the communication factors, we can see that the communication efficiency has lower values than its components serialization and transfer, suggesting that some of the codes with poor communication efficiency correspond to the cases mentioned before where we did not differentiate between serialization and transfer.

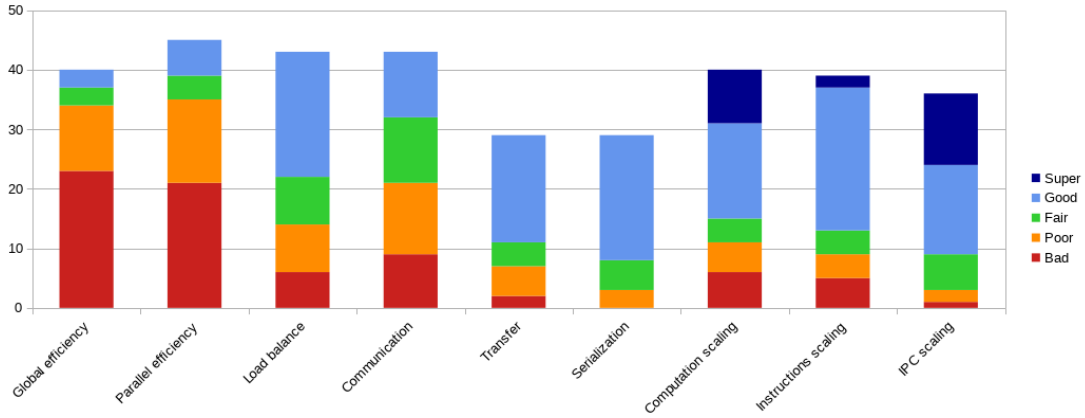

Figure 19: Key factor efficiencies classification

To validate this preliminary insight as well as to further analyse the correlation between the metrics, in the next figures we correlate the categories of a given key factor with the average value reported by its child metrics.

Figure 20: Global efficiency analysis correlates the categories of the global efficiency metric with its descendant's parallel efficiency and computation scaling. We can see that for all the categories, the parallel efficiency reports a lower value than the computation scaling. The *Bad* category is highly correlated with a low parallel efficiency (average lower than 50%) and the codes that reach the two higher categories (*Good* and *Fair*) have a

computation scaling higher than 100%. Similar insight was collected from the initial POP CoE project.

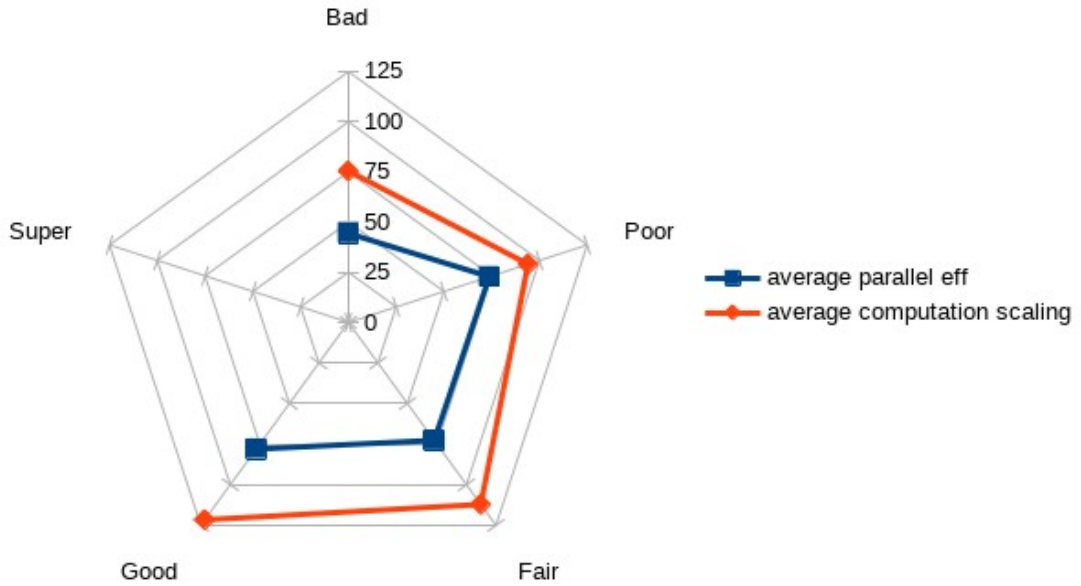

Figure 20: Global efficiency analysis

Figure 21 correlates the parallel efficiency with load balance and communication efficiencies. In this case the two metrics seems to have very similar impact on the parallel efficiency. The biggest difference is on the Bad category where the communications have a lower average value (60% vs. 68%). We can intuitively suspect that the correlation with the communications efficiency is higher as the codes classified in the Good category have average communication efficiency higher than its load balance efficiency.

In general, codes suffering more load balance problems have a better parallel efficiency, while when the communications degrade more than the imbalance, the penalty in the parallel efficiency is higher. Part of this effect can be caused by the fact that the communication efficiency is accumulating the impact of two key factors as we will see on the next figure. This is just the opposite insight identified in the first POP CoE project where imbalance problems had a higher penalty on the parallel efficiency.

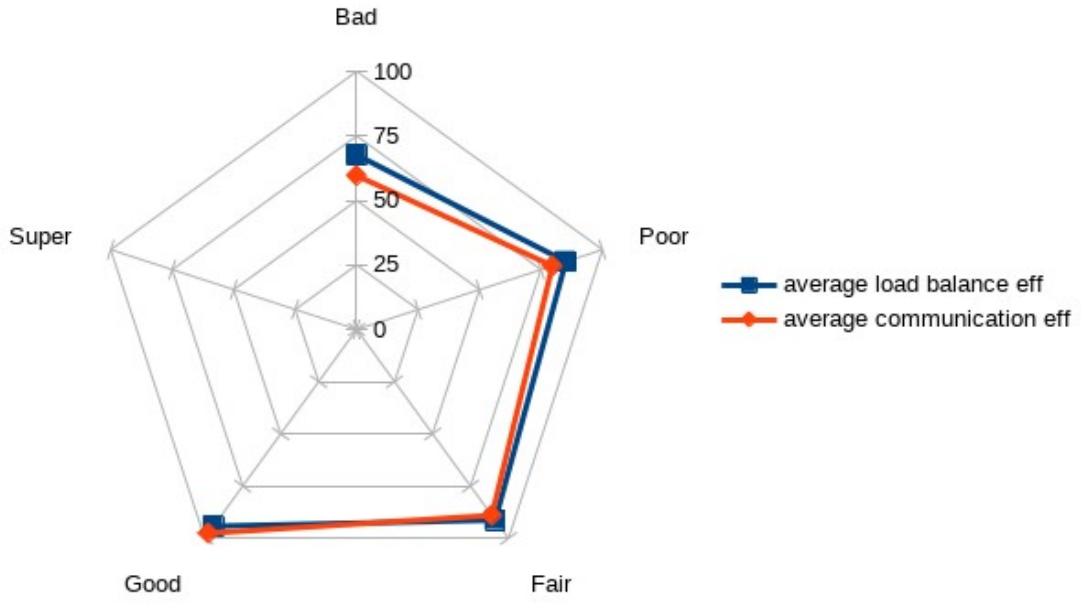

Figure 21: Parallel efficiency analysis

Figure 22 analyses the communication efficiency and its components transfer and serialization. Again, the two child metrics seems to have a very similar impact on the communication efficiency except for the Bad category where the transfer efficiency is significantly lower (68% vs. 83%). We can also see that the serialization efficiency reports "good" values in all the communication efficiency categories with average values from 83% to 98%. A similar insight about the weight of the components was collected in POP CoE project despite there we also had few codes with severe serialization problems that had the lowest values of communication efficiency.

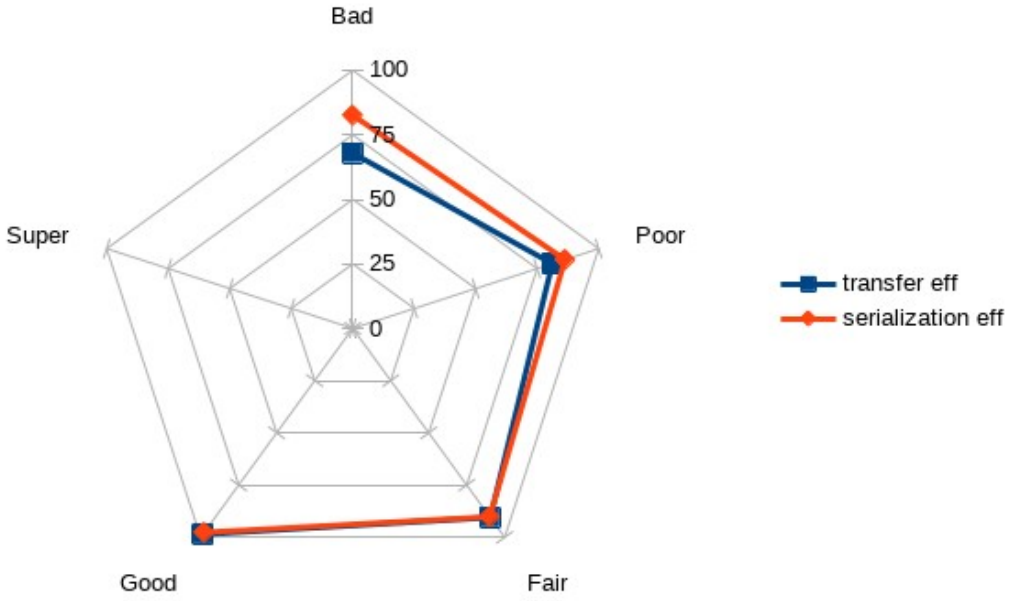

Figure 22: Communication efficiency analysis

Finally, Figure 23 analyses the scaling of the computations and its correlation with the scaling of both instructions and IPC. Except for the *Poor* category, the instructions scaling seems to have a higher influence in the computation scaling. That seems reasonable as the increase of scale may also increase the number of instructions due to some code replication while as we mentioned before, IPC may benefit from increasing the scale when strong scaling. In fact, the two higher categories (*Good* and *Super*) are showing that effect as the IPC scaling efficiencies are higher than 100%.

On the other extreme, the scaling analysis for pure OpenMP codes (or in general thread based codes) is limited to a single shared-memory compute node. Increasing the load of the node (and the load of its sockets) is typically paid with a reduction of the IPC due to the sharing of resources within a socket. That may explain the behaviour identified in the *Poor* category.

Focusing on the instructions scaling, except for the *Bad* category that has an average value of 56%, all other categories report reasonable efficiencies from 81% to 98%. That seems to indicate the problem of code replication has mainly two modes: either very severe or acceptable.

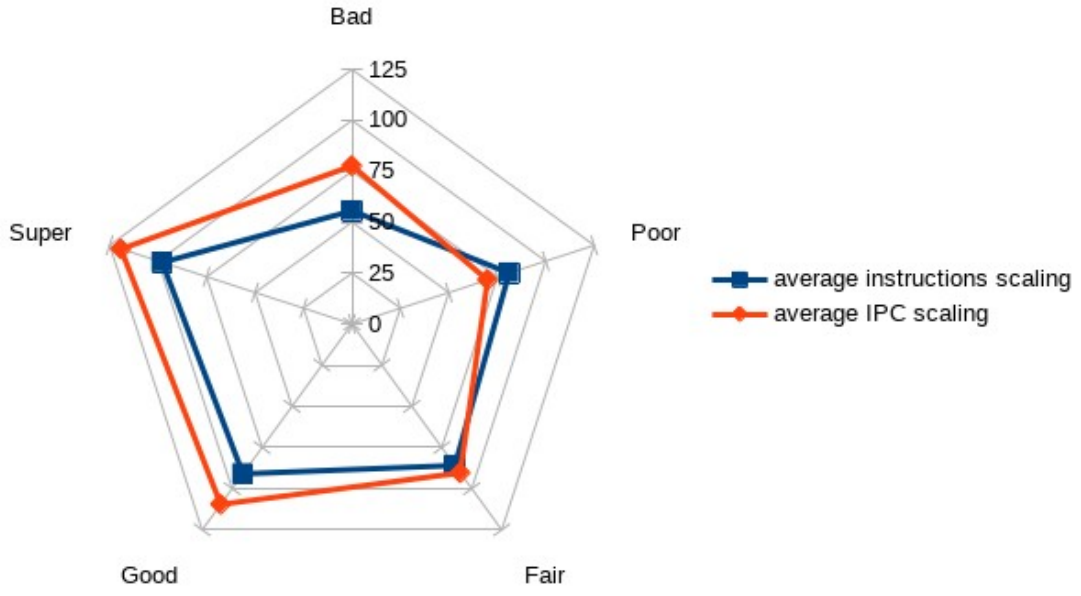

Figure 23: Computation scaling analysis

4. Findings and recommendations

This section briefly summarizes the main findings as well as the recommendations suggested in the assessments. In many cases, when the measurement does not provide the relevant performance data, the recommendation was to investigate further the aspect(s) detected.

Table 1 classifies the findings and their recommendations based on the topic they refer to. The rows are ordered by the number of instances included between parentheses.

The two most frequent topics correspond to load balance (mainly computational load imbalance) and computation scaling (related with code replication). Problems of file I/O are also relatively frequent, verifying the relevance to include I/O efficiency in the hierarchy.

General recommendations on parallel programming as well as recommendations specific for the different programming models have fewer occurrences (between 3 and 6) despite all of them totaling 19 occurrences (same number than the first two topics).

Finally, we also identify problems related with the usage of external libraries like PETSc or MKL.

Topic	**Findings**	**Recommendations**
Load balance (19)	Computational imbalance (9)	Use OpenMP
		Improve initial work distribution
		Use DLB
		Fix bug on task distribution
		Replace mesh partitioner
		Parallelise serial part
	Load imbalance (5)	
	Serialization (4)	Explore alternative scheduling
		Overlap comps and comms
	Imbalanced coupled application (1)	Optimize build system for cross-module cases

Computations (19)	Code replication (7)	
	Regions with poor IPC (6)	Improve remote memory access
		Increase data reuse
	Limited by memory bandwidth (3)	
	Poor IPC scaling (2)	
	Poor computations scaling (1)	Fix problems sharing L3 cache
I/O (9)	I/O problem (9)	Reduce number open calls
		I/O more scalable
		Check other filesystems
		Dedicated thread for I/O
		Avoid creating 1 file per process in the same directory
		Change application writing log
		Parallelize I/O
Parallel programming (6)	Unnecessary MPI calls (2)	Eliminate barriers
		Reuse mpi_comm_rank and mpi_comm_size values
	Schemes fixing MPI per node (2)	Relax the fixing
	Creating huge amount of threads (1)	OpenMP refactoring
	Race condition (1)	PoC
MPI (6)	MPI scalability limited by collectives (4)	Investigate origin communication pattern
	Data transfer problem (2)	Improve computation/communication ratio
		Reorder communication calls

OpenMP (4)	OpenMP threads idling	Overlap computation and communications
		Change scheduling
	Poor MPI+OpenMP scaling	Increase regions parallelised with OpenMP
Accelerators (3)	Poor CUDA parallelization	Improve functions executed in the CPU, concurrent executions, code refactoring
	CUDA limited scaling	Improve memory copy
External libraries (4)	inefficiencies using an external library (4)	Adjust parameters for a better performance

Table 1: Finding and recommendations

5. Recommendations for tools developers

The POP2 CoE consortium uses performance tools extensively and it is a good framework to identify recommendations for tools developers. As the tools most extensively used in the assessments are the ones developed in the consortium, some of the recommendations are treated as requests that are being implemented in WP8 (Tools and Methodology). This deliverable does not include that list of detailed requests as they are reported in D8.1 (First report on methodology development and tool improvement), but a brief overview of the recommendations.

We may separate the recommendations in the two main phases of the performance assessment where the tools are used: data collection and data analysis.

With respect to data collection, and excluding the difficulties that may be raised when installing the instrumentation tools (as those are mitigated after you get familiarised with the framework), the main problem we face is to support all the potential configurations that can be used by parallel programs. For instance, the infrequent scenario of codes that use simultaneously OpenMP and POSIX threads, or programs that combine Python and CUDA. While supporting some of them is straightforward (like combining Python and CUDA instrumentation), others may expose more difficulties like combining OpenMP constructs with explicit calls to the POSIX threads library that in fact is the layer underlying the OpenMP threads implementation. Being able to automatically detect if there is missing data that is not being collected or if there is some inconsistency on the data is as important as difficult to achieve.

With respect to the data analysis, some analysts consider important to automate the process as much as possible while others (like the one writing this document) believe it is important to carefully examine the data. Fortunately, within the consortium we have two set of tools that differ in this aspect and while Scalasca provides a more automatic analysis, Paraver targets to facilitate browsing and exploring the details captured.

Acronyms and Abbreviations

Each term should be bulleted with a definition.
Below is an initial list that should be adapted to the given deliverable.

- BSC – Barcelona Supercomputing Center
- D – deliverable
- HLRS – High Performance Computing Centre (University of Stuttgart)
- HPC – High Performance Computing
- IT4I - Vysoka Skola Banska - Technicka Univerzita Ostrava
- Juelich – Forschungszentrum Juelich GmbH
- KPI – Key Performance Indicator
- M – Month
- MS – Milestones
- POP – Performance Optimization and Productivity
- RWTH Aachen – Rheinisch-Westfaelische Technische Hochschule Aachen
- USTUTT (HLRS) – University of Stuttgart
- UVSQ - Université de Versailles Saint-Quentin-en-Yvelines
- WP – Work Package

List of Figures