D5.2 POP Analysis Report
Version 1.0

Document Information

Contract Number	824080
Project Website	www.pop-coe.eu
Contractual Deadline	M36, November 2021
Dissemination Level	Public
Nature	Report
Author	Judit Gimenez (BSC)
Contributor(s)	
Reviewer	Samir Ben Chaabane (TERATEC)
Keywords	Performance assessments, Evolution and analysis, Analysis report

Notices:
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".

© 2015 POP Consortium Partners. All rights reserved.

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	Sameer Ben Chaabane	Reviewed version
V0.3	Judit Gimenez	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 second half of the project and characterizes the cases analysed during the full POP2 project, summarizing findings and recommendations provided to the customers. Some of the metrics are compared with the experience collected on the predecessor POP CoE project, and with the previous report from May 2020.

As a reference of the activity of the assessment services work package, during the 3 years of POP2 we have had 193 new services; until November 9th, 136 studies have been completed and there are 6 services in reporting state that probably will be completed before the end of the third year; there are also 26 services in progress that will be completed in the coming weeks/months. These sum up to a total of 168 services either completed or in progress overpassing the KPI defined for the assessments work package of 140 studies completed or in progress. Between 40 to 50% of the users have requested a second service (including not only the assessments but also the proof-of-concepts), and in multiple cases the relationship has been extended to a third service. A total of 30 codes from 10 CoEs have been audited by POP2, close to half of them with multiple assessments summing up a total of 50 services. Only around a 10% of the services generated during POP2 have been cancelled and a significant percentage of these cancellations were invalid requests.

The annexes of the deliverable are a list of the services and the reports produced in the second 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 3 years of the POP2 project. These services are provided free of charge to developers and users of parallel codes with the goal to help them 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 had not yet started, but also to demonstrate our commitment to the Centre of Excellence. Even though during that period the volume of work was significantly reduced, we maintained the possibility for users 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 November 9, 2021 (at the time writing this deliverable), we have 217 assessment services, 193 studies originated since the start of the POP2 project. Most of the assessment services correspond to initial audits (183 studies) while 34 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 the three years, 32 assessment studies have been cancelled, most of them after a long period where the user did not reply to our requests, few of them because the user moved to a different department or company. There are few cases of invalid requests such as training requests through the web form, requests to analyse not parallel codes or duplicated requests. Close to one third of the cancelled studies were studies from the original POP project that were delayed until the start of POP2.

In POP2, KPIs as well as milestones combine the studies from both WP5 and WP6 (Proof-of-Concept). We use as a reference the studies either completed or in progress because in the original POP CoE project, this 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 initially agreed that 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. This ratio was re-evaluated in the second half of the project, slightly increasing the WP5 studies from 135 to 140.

This deliverable focuses on four main aspects: the evolvement of the assessment services (section 2), the characterization of the services provided (section 3), a summary of the findings and recommendations (section 4) and the assessments of the CoEs applications (section 5). In the rest of this document, we use the term “executed” to refer to the studies either completed or in progress that is the metric used in the KPI.

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 the project. To evaluate the number of studies with respect to the work plan, we include as reference a linear distribution (blue line). The vertical line splits the chart in the two periods of 18 months. The three metrics plotted classify the studies based on their grade of completion: *completed* corresponds to studies already finalised, *executed* groups the studies completed and the ones in progress, and *total* adds the studies requested that are not yet started or that are waiting the user. In this figure we do not explicitly include the studies that were cancelled but we can see their impact when the lines go down from the previous month.

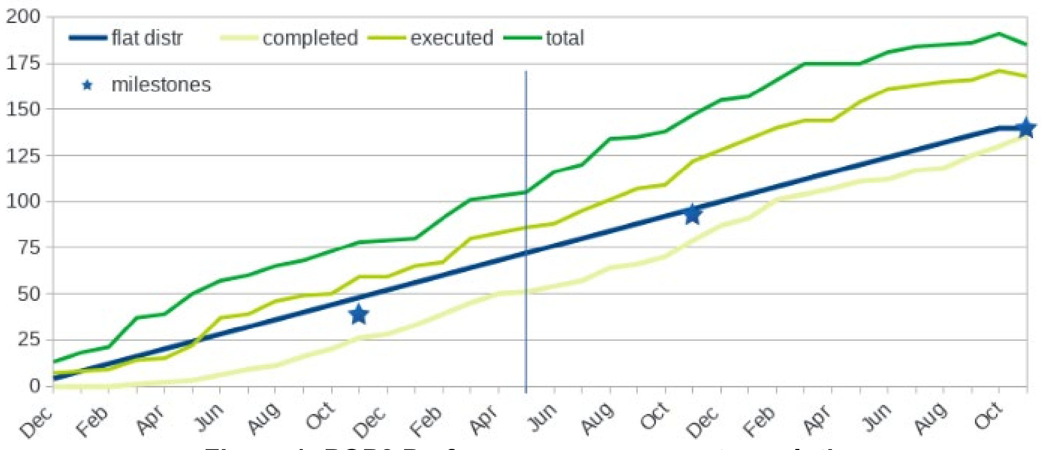

Figure 1: POP2 Performance assessments evolution

Focusing on the second part of the project, we can see that during the full period the linear distribution is situated between the completed and the executed studies, getting closer to the completed studies.

During the full 3 years, the total number of studies has been always significantly higher than the planed one, 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.

The assessments work package contributes in three milestones: MS2, MS5 and MS7. To validate the achievements with respect to the milestones, the plot includes the 3 milestones isolating the contribution of WP5 to reach the final target of 140 studies completed or in progress at M36. First thing we can see is that all of them are close to the linear distribution but, with a higher slope. Milestone MS2 was planned for M12 with a global goal of 50 studies (39 from WP5). Considering only WP5 goal, the milestone was reached in advance at M8 but if we consider only the studies that are on an advanced progress, the milestone was achieved at M13. Milestone MS5 was scheduled for M24 with a global goal of 120 studies (93 from WP5). Considering only WP5 goal, this milestone was reached at M20, but again if we do not consider the studies that are not close to completion, the milestone was reached on its scheduled date. Finally, MS7 is scheduled at M36 with a global goal of 180 studies (140 from WP5). Considering only the studies either completed or very close to completion, the milestone is also reached in advance, at M35.

Figure 2 plots the distribution of the POP2 Performance Assessments with a more detailed classification of their state. 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 (referenced in the previous paragraph). We can also see the evolution of the number of cancelled studies, 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).

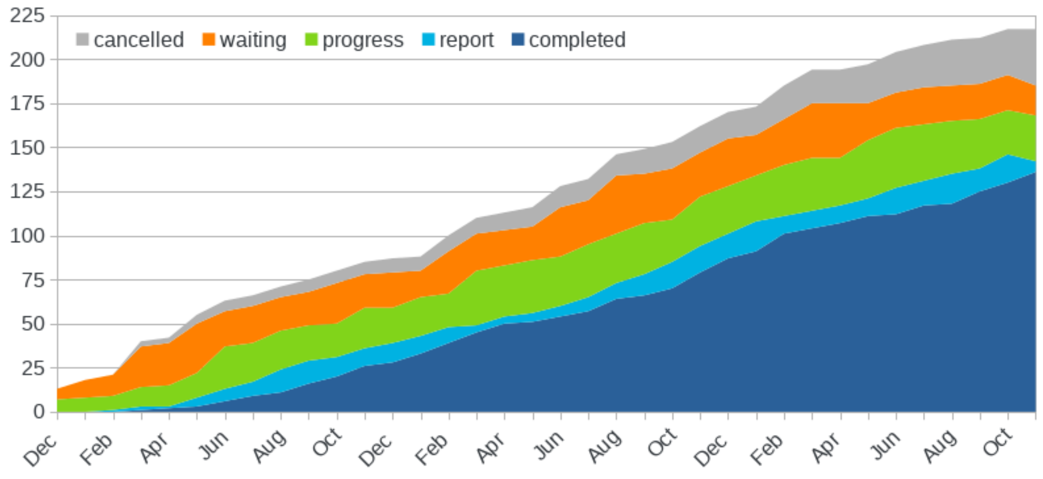

Figure 2: Distribution of the POP2 Performance assessments

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 initial end of the project, with a uniform linear distribution the target would be to reach 100% with the completed, reporting and in progress studies. This means the orange part of the bar should start above 100%.

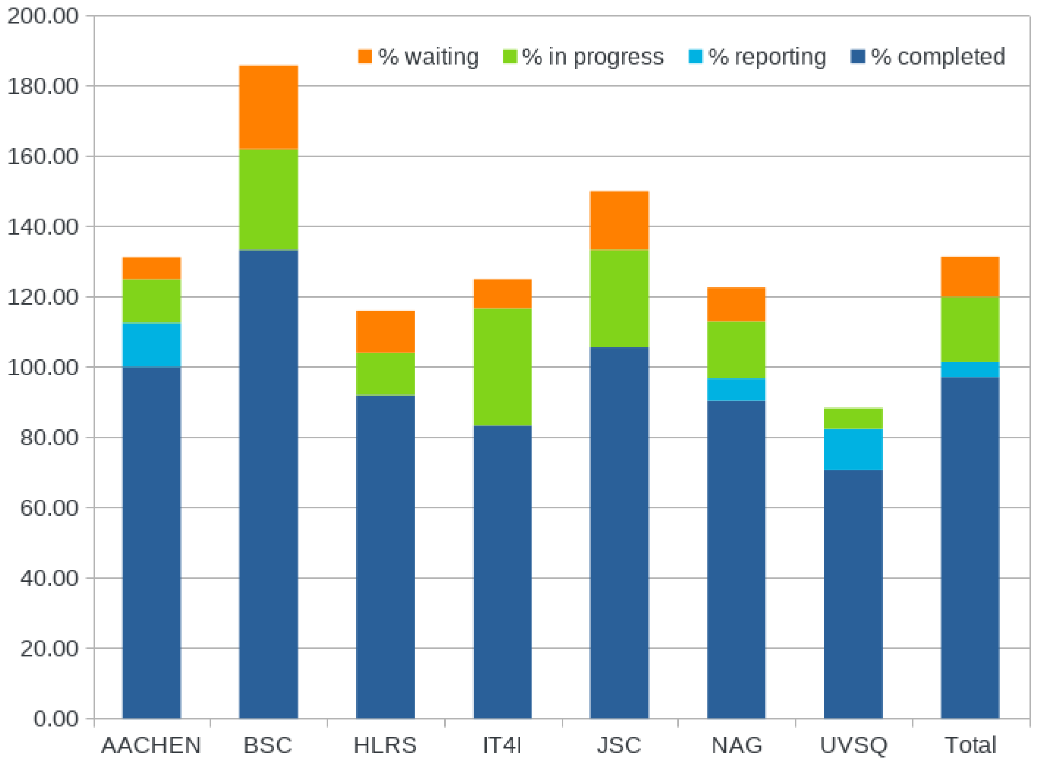

Figure 3: POP2 Performance assessments per partner

We can see that despite initially expecting a very similar bar for each of the partners, the differences are significant.

Three partners have reached 100% (or more) of their goal when considering only the completed studies. BSC has the higher bar and the completed studies are 133%, reaching 186% of the planned assignment. This is partially due to the fact that as leaders of WP5 we assumed studies when it was not possible to assign to other partners. JSC has completed 106% of the planned assignment and the total assignment is 150%. As it was already reported in D5.1, around 25% of the POP service requests for the JSC user community are charged to other projects and not to the POP2 project. AACHEN reached 100% with completed studies and the total assignment is 131%.

NAG is very close to 100% when considering only the studies completed and reporting (97%), and goes up to 113% including the studies in progress. Both IT4I and HLRS reached also their goal when including the work in progress with a percentage of 117% and 104% respectively. With the same metric,

UVSQ has only achieved 88%, due to some recent cancellations as well as to an assessment that was moved to Proof-of-Concept. It's the only partner that has a global assignment lower than the target.

Finally, the consortium as a whole has completed 97% of the targeted studies and reached 101% of the KPI even if we only consider the studies that are either completed or reporting. This value goes up to 120% if we add the studies in progress and the total assignment is of 131% of the initial plan. 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.

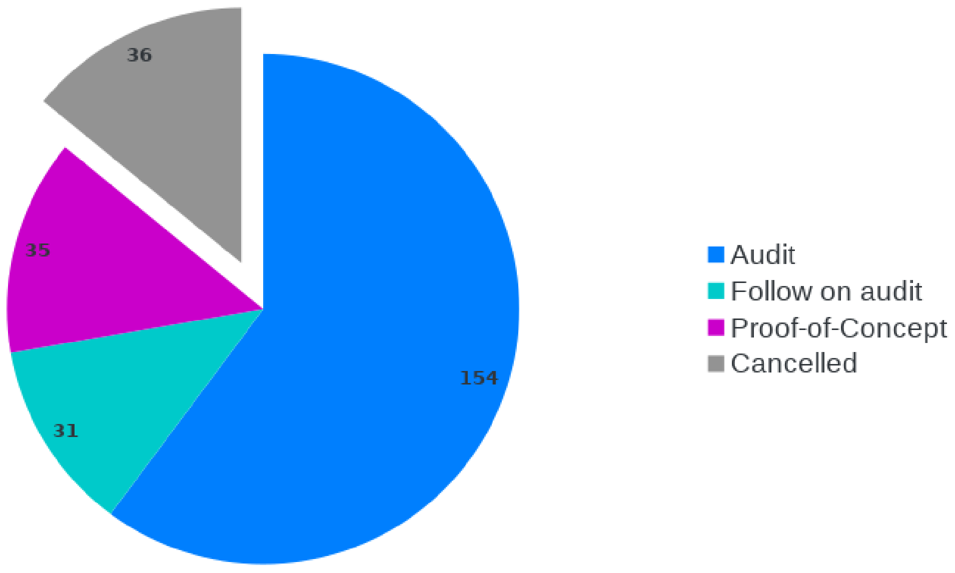

Figure 4: POP2 Studies (WP5+WP6)

We can see that the continuation studies are very similarly distributed between Follow on Audits and Proof-of-Concept studies with 13% and 15% of the total studies.

Without considering the 14% of the studies that have been cancelled, the numbers indicate that 43% of the Audits are extended with a second service. Considering that the total number of studies completed when writing this deliverable is 136, the ratio of second services goes up to almost 50% and seems a very high percentage considering that for some of the codes the assessment reports a good performance where there is no need for improvement.

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 update the analysis reported in deliverable D5.1, characterising the assessments carried out during the 3 years of POP2 project. As in the previous deliverable, 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). We include the cancelled studies when the data is available. And for the scaling analysis we introduce 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 user profile with respect to the code. Figure 5 plots the distribution on the 3 roles we identify in the request form. More than 75% 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 13% (25 users) 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 both the previous deliverable and the first POP CoE project.

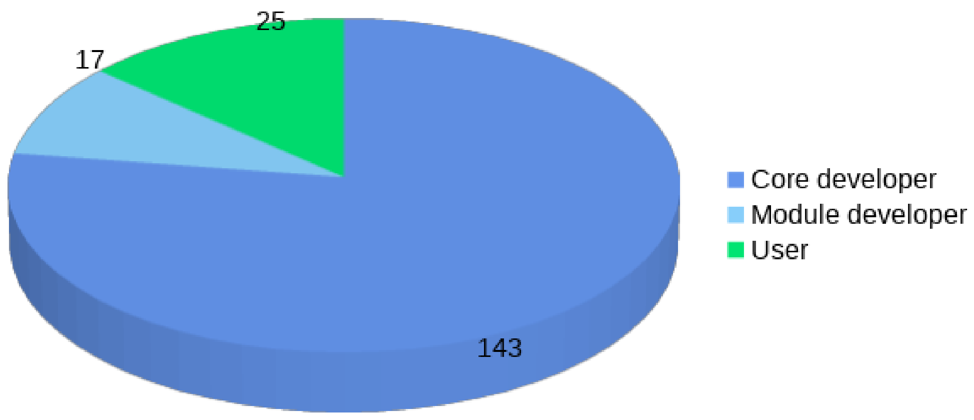

Figure 5: User role

A second aspect that we consider relevant to remark is the profile of the users with respect to their familiarity with performance tools. With the same distribution that in D5.1, 71% of the users that replied to that question (126 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 that the user considers most relevant to focus on in the assessment. Figure 6 plots the classification of the service requests. Close to 65% of the users request a performance check, indicating they are interested in a global analysis of the code. This percentage has been reduced from the previous deliverable where the value was close to 75%. That reduction is reflected as an increase from 11% to 25% in the requests that selects either to identify areas of improvements or analyse the efficiency of the parallel execution.

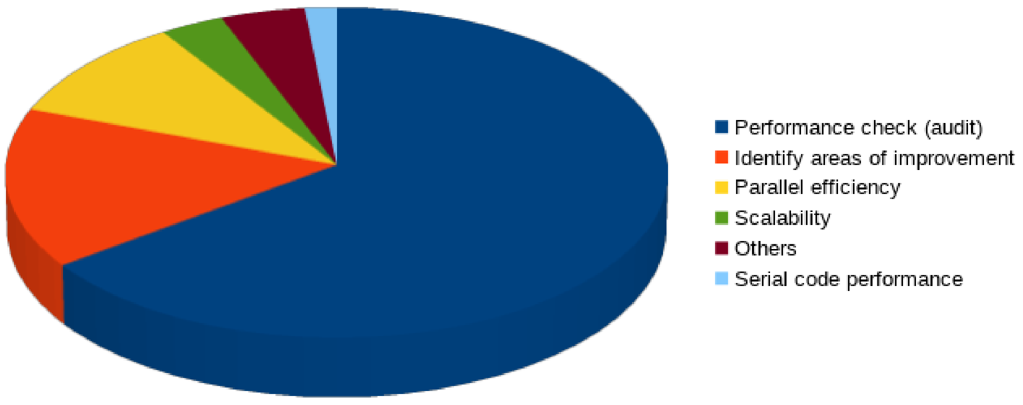

Figure 6: Service request

The last aspect we analyse from 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. Almost 70% of the requests come after a direct contact with one of the POP partners. Word of mouth and project partners sum up close to 23%. Grouping the sources of social media, website, email and news, the percentage is much lower (only 7%) indicating the need for a direct contact either from a POP partner or from some person that is already aware of the POP CoE services. The distribution is quite similar to the one observed in the previous analyses.

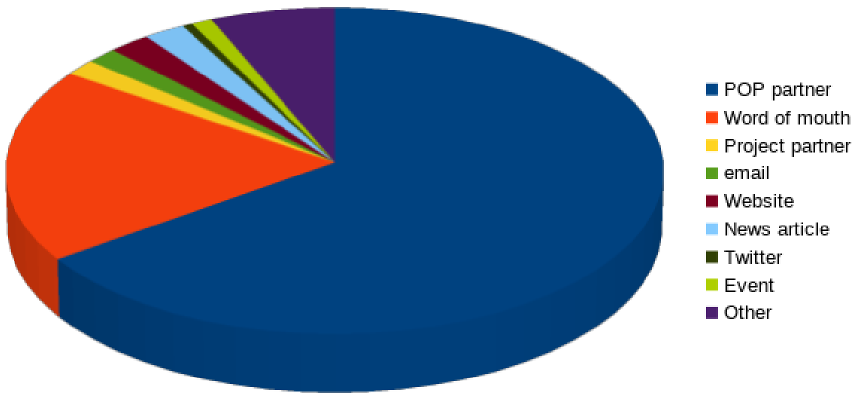

Figure 7: Source of contacts

2 Code

The first characterisation 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. Around 63% of the codes are distributed between the areas of *Earth*, *Engineering* and *Physics*. Between 7 and 8% are the rates of both *Aerospace* and *Chemistry*. All other areas represent less than 5% except *Others* that is close to 6%.

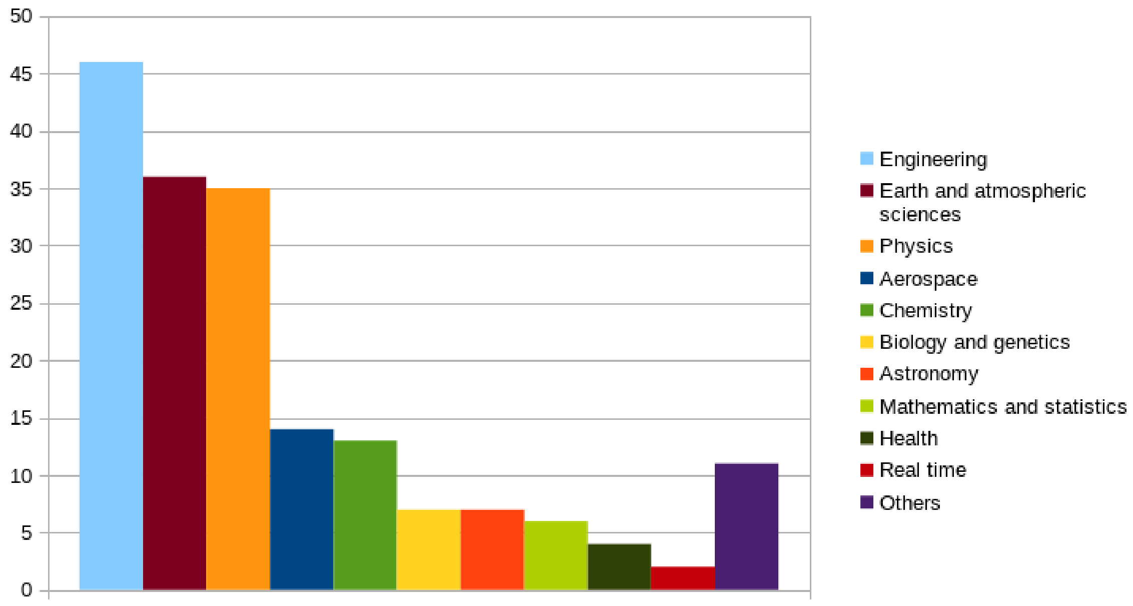

Figure 8: Code scientific/technical area

The results are similar to the percentages measured half way in POP2, with a reduction of the weight of the top 3 areas from 73% to 63%, with a higher impact from *Earth* and *Physics*; and an increase of the contribution of both *Aerospace* and *Chemistry.* Comparing the results with the initial POP CoE project, there is a very significant increase in the number of *Earth Science* codes and still some reduction of the *Chemistry* codes. While in both previous analyses, the sum of the traditional sectors (*Physics*, *Engineering, Earth* and *Chemistry*) represented a percentage of around 77%, this percentage is reduced down to 70%.

One of the optional questions we ask users is if the code was analysed before their request. From 128 answers, 75% of the codes have not been 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 is playing 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 90 replies. Close to 80% of the codes run as standalone, but 10% of the codes are usually executed as part of a coupled application, which is a very frequent scenario for Earth Science codes. In fact, both the percentage of coupled codes and the percentage of codes from Earth Science have decreased compared to D5.1.

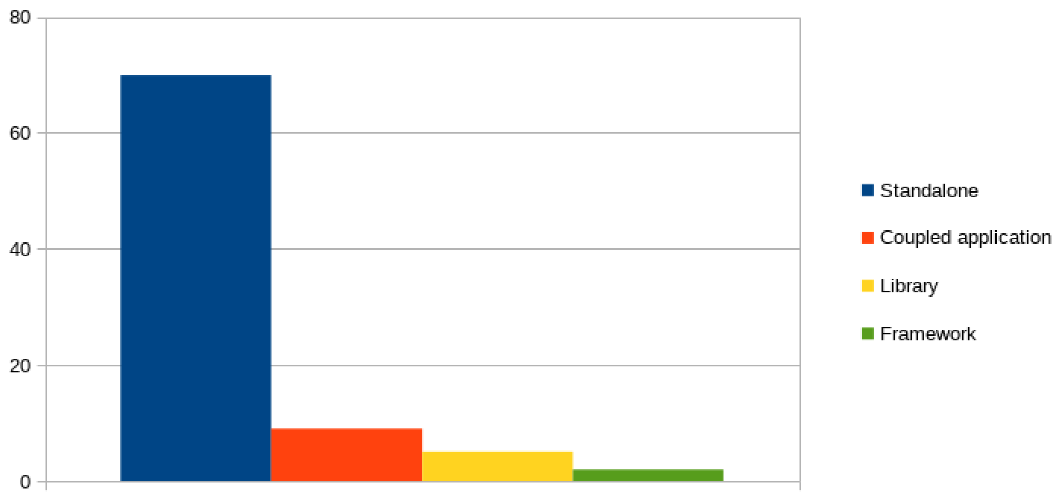

Figure 9: Code profile

To characterize the potential distribution of the code and the impact of improving them, Figure 10 shows the type of code licensing. With a population of 86 answers, close to 60% of the codes are distributed with a free license or no license, and only 23% have a commercial license. An 8% of the codes are limited to internal use, some of them because they are still under development. These percentages were similar in the previous analyses.

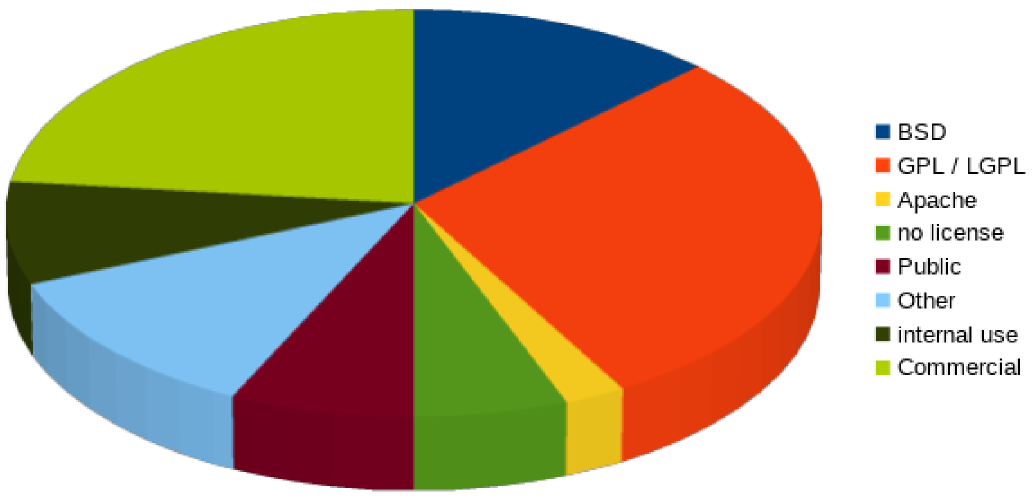

Figure 10: Code license

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 the programming models of all the existing variants of the code and in many cases the analysis focuses to the version most frequently used. For instance, codes that have a version that can run on accelerators may be still under development so the user is not interested to analyse this part of the code. Nevertheless, and just for this aspect, we have detected a relevant increase of requests to analyse codes running with accelerators, during the last two years.

Figure 11 plots the parallel programming models used by the codes. 83% of the codes use MPI to be able to run in distributed memory architectures. Nevertheless, only 46% of the codes are pure MPI codes and 37% also support multithreading and/or kernel offload to accelerators. A total of 45% of the codes have support for threads, mainly through OpenMP but also using POSIX threads or even both OpenMP and POSIX threads concurrently. Somehow surprisingly, 13% cannot run on multiple nodes as they are programmed using threads, and 5% extended the parallelization with accelerators instead targeting distributed memory (indicating the target platform is a single computer that usually includes an accelerator). Finally, still only 16% of the codes have support to run on accelerators which is typically combined with MPI. All these percentages are very similar to the ones obtained in the previous study. 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.

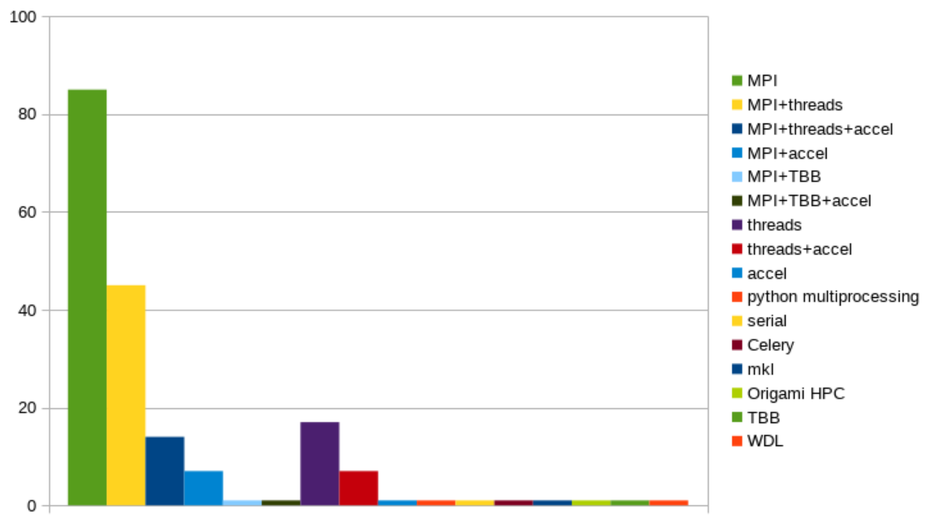

Figure 11: Parallel programming model

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

To check if there is a correlation between the scientific area and the programming model used, Figure 12 plots the most frequent programming model per area (discarding areas or programming models with just one code). Bluish colours are used to group codes that use MPI while orange-like colours group shared memory codes. We can see that both pure MPI codes and pure shared memory apply in almost all the sectors except *Aerospace*. We can also confirm that alternatives with support to accelerators increase its percentage in almost all the sectors.

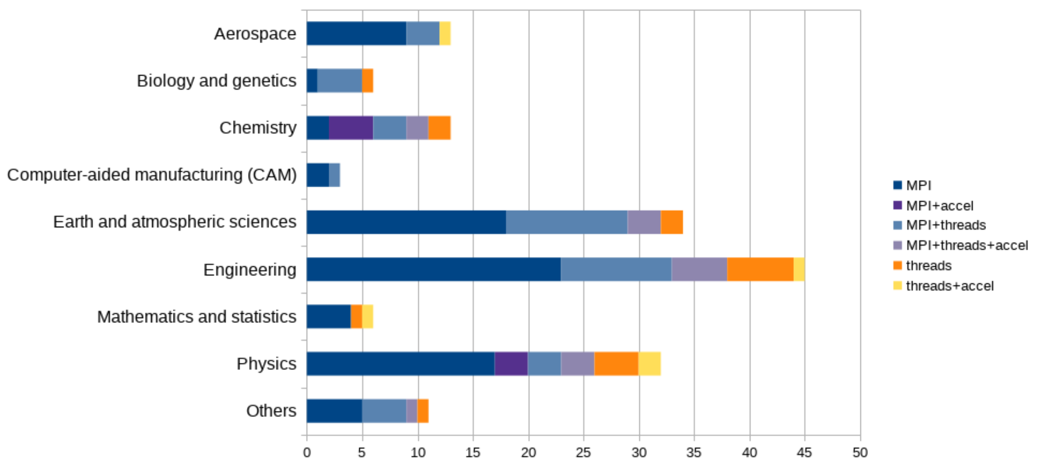

Figure 12: Programming model per scientific area

In that sense, it is surprising that there are no codes with support to accelerators in the *Biology and genetics* area.

The distribution with respect to the programming language is plotted in Figure 13. Pure C++ codes have the higher percentage with 30% of the codes, similar to the previous analysis; while pure Fortran codes increases their percentage from 23% to 29%. C++ or C is used in 62% of the codes while Fortran contributes in 52%. Finally, 45% of the codes use also Python and pure Python codes are 4%.

If we compare the distribution with the previous statistics half way in the POP2 project, the higher increase is in Fortran codes as well as in the codes that include Python (but not for pure Python codes).

The increase of Fortran codes gets a distribution closer to the one measured in the predecessor POP project where pure Fortran was the most frequent scenario. The increase of codes with Python is an evolution we have seen in the full life of the CoE.

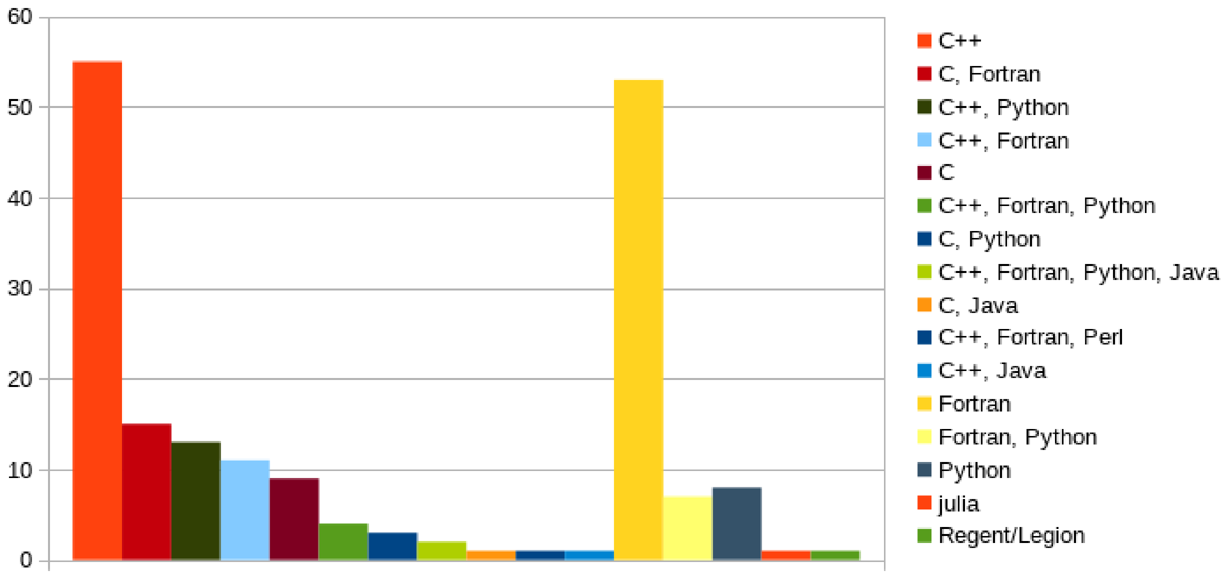

Figure 13: Programming language

Figure 14 plots the most frequent programming language per area (discarding combinations with just one code). Light blue colours are used for C++/C codes and green for pure Fortran codes. Yellowish colours are used to mark the codes that include Python. We can see that in many sectors there is a balance between C/C++ and Fortran (this is clearer in the areas with a bigger population). We can also see that Python is spread over almost all the areas.

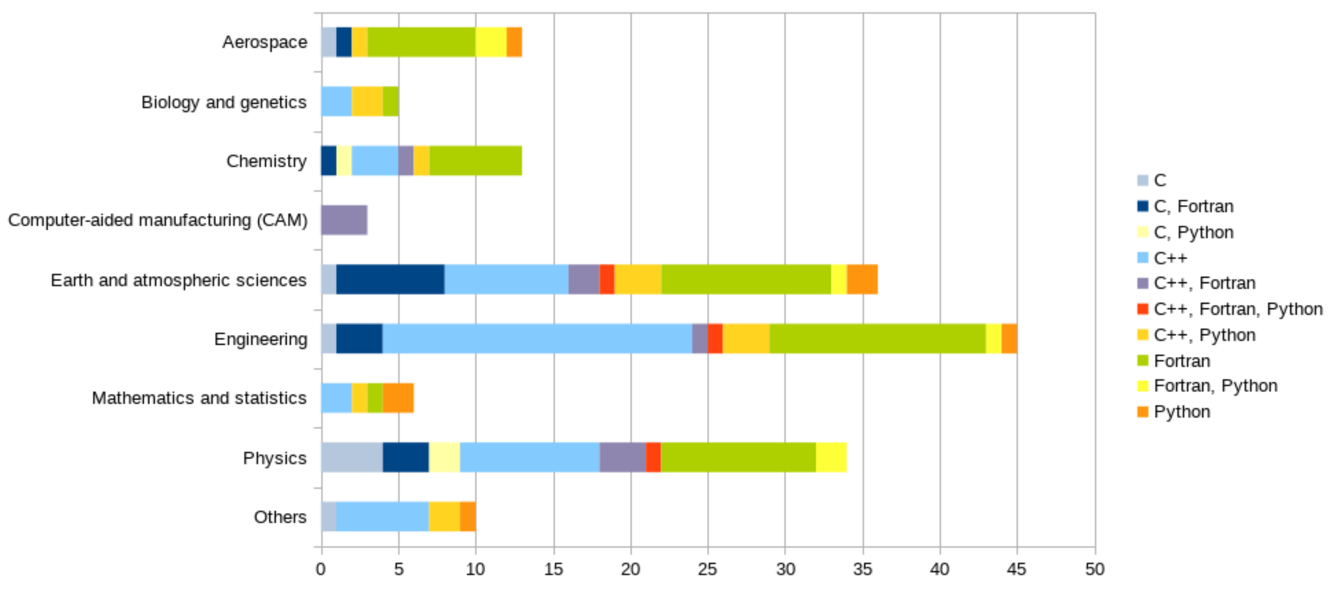

Figure 14: Programming language per scientific area

2.2 Code Scaling

We collect four metrics for 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 the user is who determines the scale that has to be analysed. Figure 15 compares these four metrics classifying the values with respect to their order of magnitude.

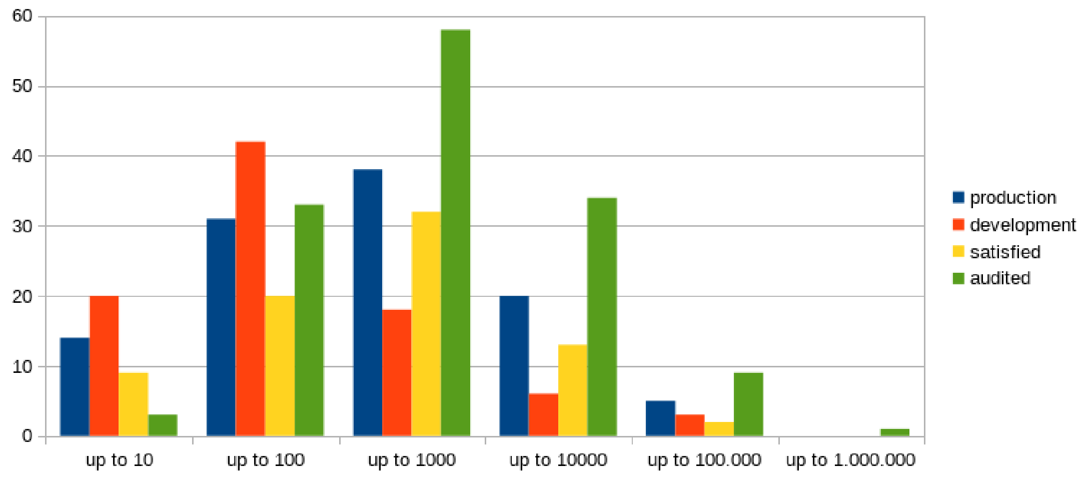

Figure 15: Scales comparison

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. The scale that the user is satisfied with the performance is similar to the production runs as it may be expected, even if this question obtained a lower number of answers (76 versus 108).

Finally, the audited runs are also in the range from 100 to 10.000 cores with a significant increase of the studies that analysed runs between 101 and 1000. 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. The percentage of studies audited in the range between 10001 and 10.000 has also been increased with respect to the previous report. As it was written in D5.1, we think it is interesting to remark that we analysed 10 codes with a range of cores between 10.000 and 1.000.000 cores (with 309696 being the largest case analysed), higher than their range in production. That indicates that these studies were used to validate the performance of the codes at a larger scale that the one typically used.

Focusing on the 3 metrics provided in the form classifying the data provided based on which value (production, development or satisfied) is larger. Figure 16 plots the result of this classification. We can see that 27% of the users that provided the 3 metrics are satisfied with the scaling larger than the one used in production or development. In fact, for many of these cases, this value is significantly higher. In our opinion this category is grouping two user profiles: users that do not have enough resources to do larger runs and users that misunderstood the question, because if they are satisfied with a much larger scale than the one they use, they will not feel the need to improve the code performance. Next ranking is for the users that give similar values for all the metrics and correspond to 25% of the cases and users that are satisfied with the scaling of the production runs (close to 7%). These categories will correspond to users that want to increase the scaling and feel the need to improve the performance first. A small 1.7% of users are satisfied with the development runs that are larger than the production scale, indicating it may be a code under development. All other groups, representing almost 16% of the users that provided us with that data, are not satisfied with the scaling of their production runs.

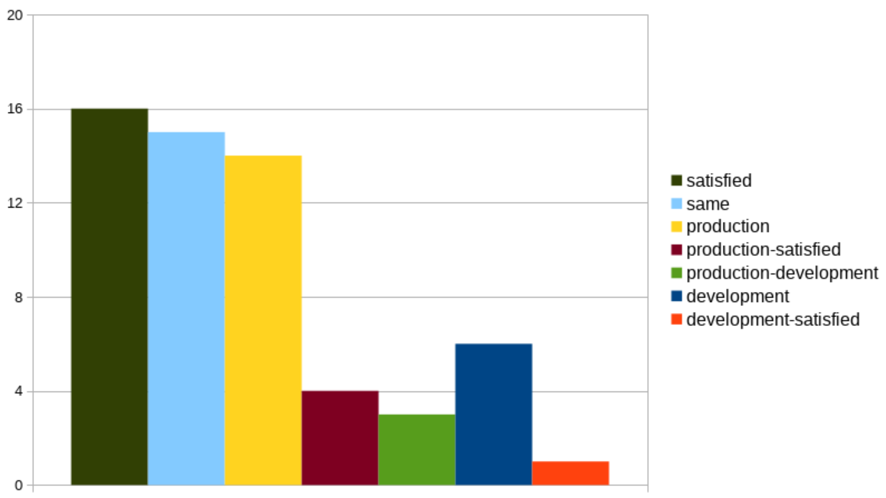

Figure 16: Scaling metrics

To better compare the four metrics, the next four figures (Figure 17 to Figure 20) focus on the studies we obtained the four metrics and plot their values depending on its range.

Going from small to large scales, Figure 17 plots the metrics for the studies with values up to 500 cores. We can identify very different behaviours in this group: users that benefit from the analysis to explore the scaling on a much larger scale than the one they are using or for which they can be satisfied (most frequent scenario), users that have very similar values for the 4 metrics and users that requested us to analyse a scale smaller than the one they told us they are using in production or development but which is usually higher than the one that they consider achieve a satisfying performance.

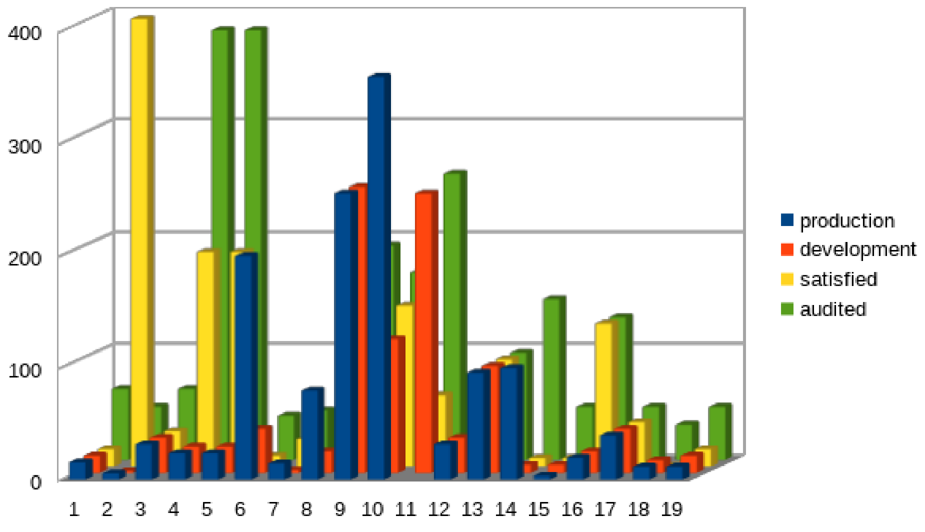

Figure 17: Scaling metrics (up to 500 cores)

Figure 18 focuses in the range from 500 to 1000 cores. We have included in this plot two studies that despite they usually use a scale smaller to 1000 cores, the audit targeted up to 1536 cores. In case #3, the production and development runs use less than 100 cores, in case #5, the production runs are close to 900 cores. In both cases the users defined the limit for satisfied on 1000 cores but they selected a larger scale when auditing, probably to explore the bottlenecks on these configurations.

In this group we can see that development runs generally use very small number of cores (except for #6) and they are satisfied with the performance of a larger scale that the one used in production. Around half of them requested to audit their codes at a scale larger than the one for which they are satisfied, but the other half wanted us to focus on a shorter scale.

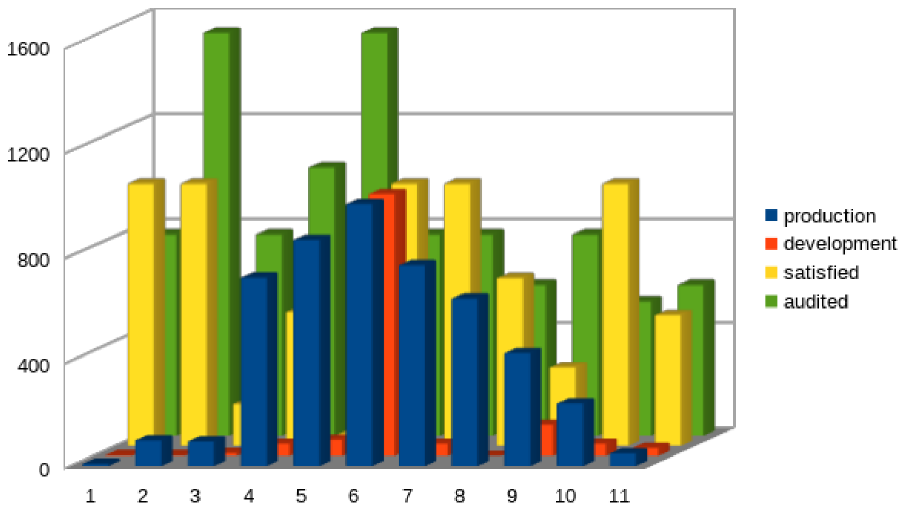

Figure 18: Scaling metrics (up to 1000 cores)

Figure 19 focus in the range from 1000 to 9000 cores. We can see that most of the studies correspond to codes that use much less cores in production than the value specified in satisfied section, for almost all of them the largest audited run was an intermediate value. There are also a couple of cases where the scale used in development is significantly higher than the one used in production.

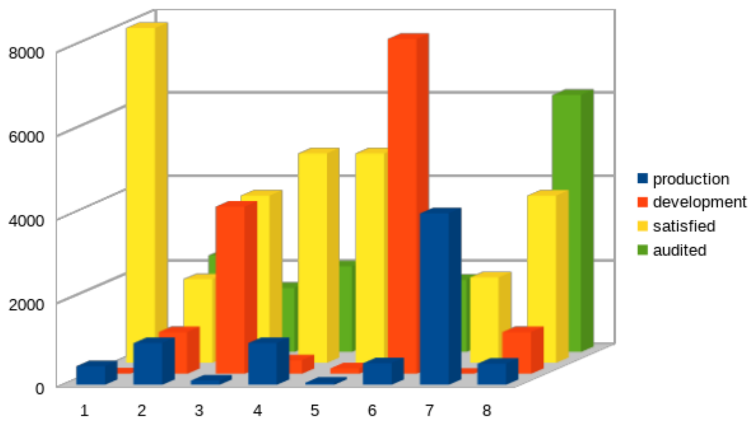

Figure 19: Scaling metrics (up to 9000 cores)

Case #8 corresponds to one of these studies where the study targeted a scale larger than all the other metrics.

Finally, Figure 20 focuses in the range from 10000 cores. Surprisingly, half of the studies that reported us some scale larger than 10000 cores, requested us to audit no more than 500 cores. For only two cases we audited a larger scale than the one they are satisfied with the performance. Checking the data, we detected that for some of these codes, the user wanted us to analyse a different version of the code.

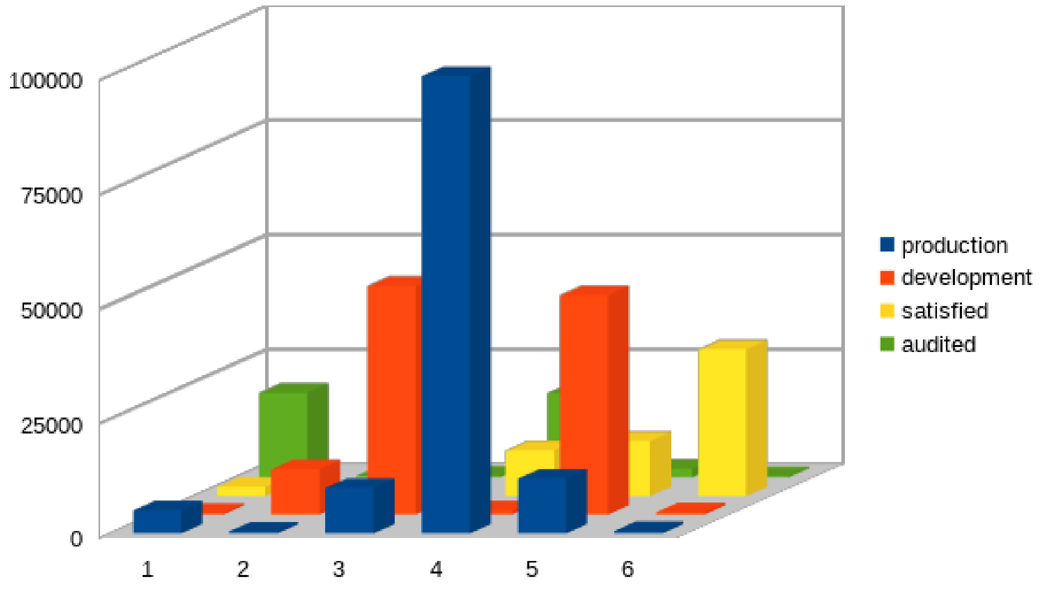

Figure 20: Scaling metrics (from 10000 cores)

To complement the analysis of the assessments scale we should also consider the answer to other related questions. This includes if runs are usually single executions or use many instances concurrently. 41% of the responses usually run multiple executions at the same time (80 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, 57% of the 87 codes are typically executed on the user local system that may also limit the maximum scale they can target for their production runs. Finally, 79% of the 113 answers use strong scaling mode where the problem size is maintained when increasing the number of resources. This approach also influences the scale as the range of cores that make sense to use, would be determined and limited by the input.

Comparing the scaling values with the data collected in the precursor POP CoE project, the average number of cores is similar, while the largest scale reported within POP was 239,615 cores and in POP2 is 309,696.

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 initialisation and finalisation 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 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 that were of more interest for the users, than the real performance that can be achieved by the codes.

To analyse the collected efficiencies, and as it was done in the previous deliverable, 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 optimisations 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 21 plots the distribution over these categories for the main efficiency factors measured in our 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%.

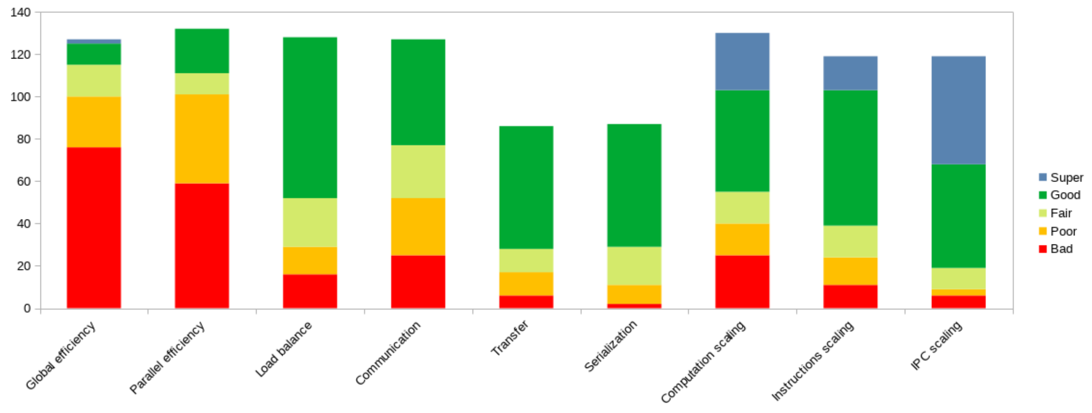

Figure 21: Key factor efficiencies classification

As it can be expected, global efficiency is the metric with the worst values because it accumulates the losses 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 and in two studies at the global efficiency level meaning that overcompensates all the loss of the parallel efficiency.

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 or that the combination of the two components impacted on the communication efficiency.

Finally with respect to the computation scaling, despite the higher percentage is classified as good or super, there are some of them with very bad scaling of the computations with a higher contribution from the instructions scaling, identifying applications that suffer from code replication or increase of instructions with the scale.

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 22 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. Similar insight was collected from the previous analyses. The *Bad* category is highly correlated with a low parallel efficiency (average lower than 50%) and it is the category with the lower average computation scaling. In all the other categories the scaling of the computation is close to 100% or higher compensating the loss due to the parallel runtime. Interestingly, for the two cases classified as Super, the average parallel efficiency is lower than the average value for the Good category. One of these cases corresponds to a parallelization from GPUs to CPUs that produced a drastic reduction of the computation time.

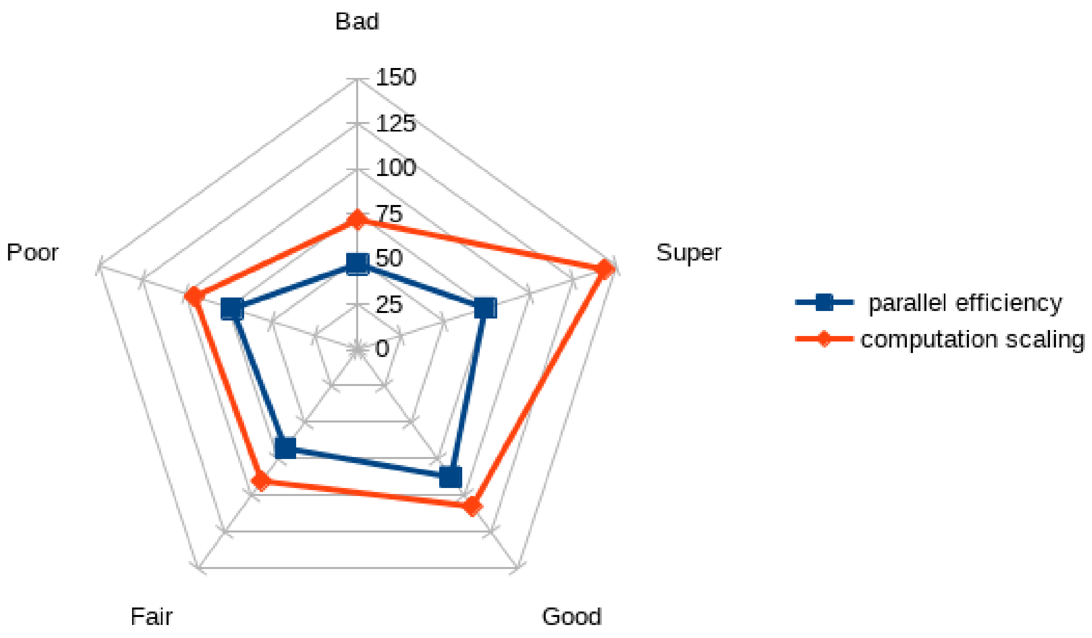

Figure 22: Global efficiency analysis

Figure 23 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 (59% vs. 70%). We can intuitively suspect that the correlation with the communications efficiency is higher because the codes classified in the Good category have average communication efficiency higher than their load balance efficiency.

In general, codes with more load balancing problems have a better parallel efficiency, while when the communications degrade more than the imbalance, the penalty for 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 the same situation we reported in D5.1 but it is just the opposite insight identified in the first POP CoE project where imbalance problems had a higher penalty on the parallel efficiency.

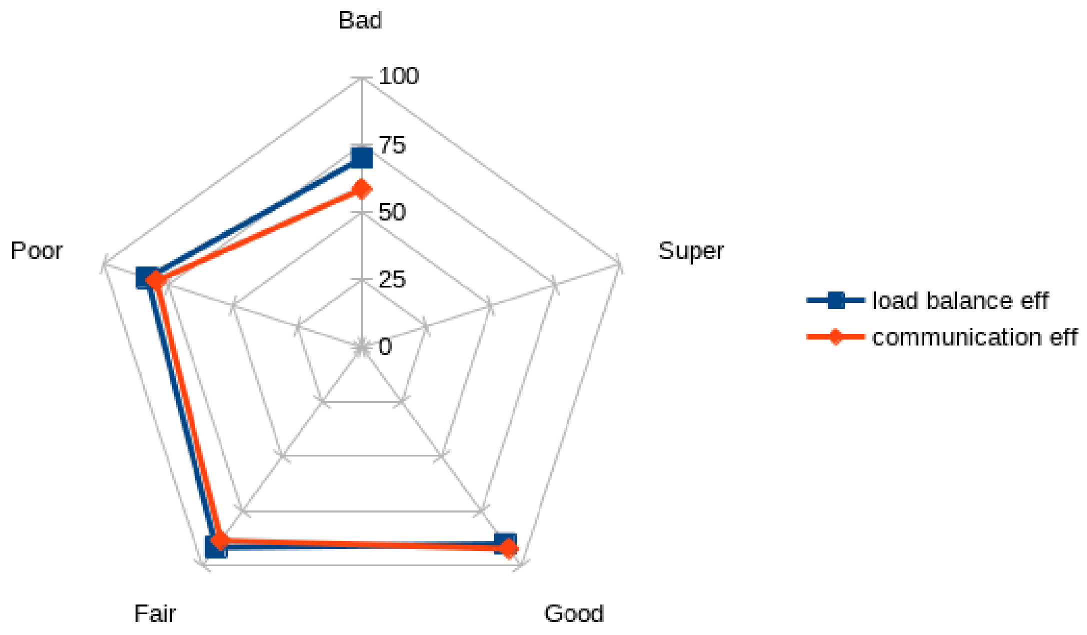

Figure 23: Parallel efficiency analysis

Figure 24 analyses the communication efficiency and its components transfer and serialization. Again, the two child metrics seem to have a very similar impact on the communication efficiency except for the Bad category where the transfer efficiency is significantly lower (65% vs. 74%). We can also see that the serialization efficiency reports “good” values in all the communication efficiency categories with average values from 74% to 97%. A similar insight about the weight of the components was collected in POP CoE project despite the fact that we also had few codes with severe serialization problems that had the lowest values of communication efficiency.

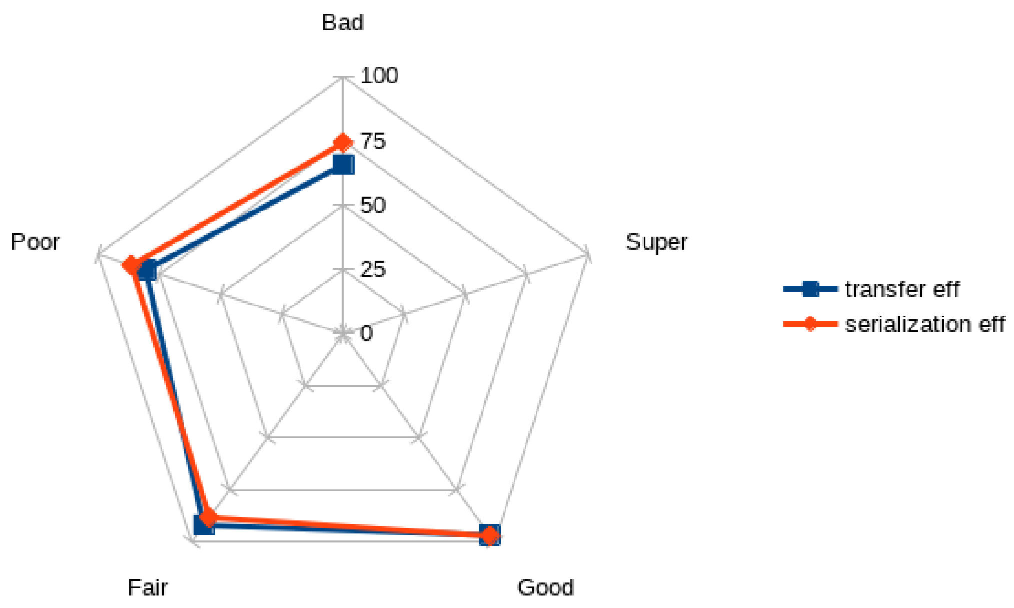

Figure 24: Communication efficiency analysis

Lastly, Figure 25 analyses the scaling of the computations and its correlation with the scaling of both instructions and IPC. In all the categories 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 scale increasing when using 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 reflected as 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, the *Bad* and *Poor* categories have an average value of 70% and 75% respectively all other categories report quite good efficiencies from 89% to 98%. That seems to indicate the problem of code replication usually is not very severe or even acceptable.

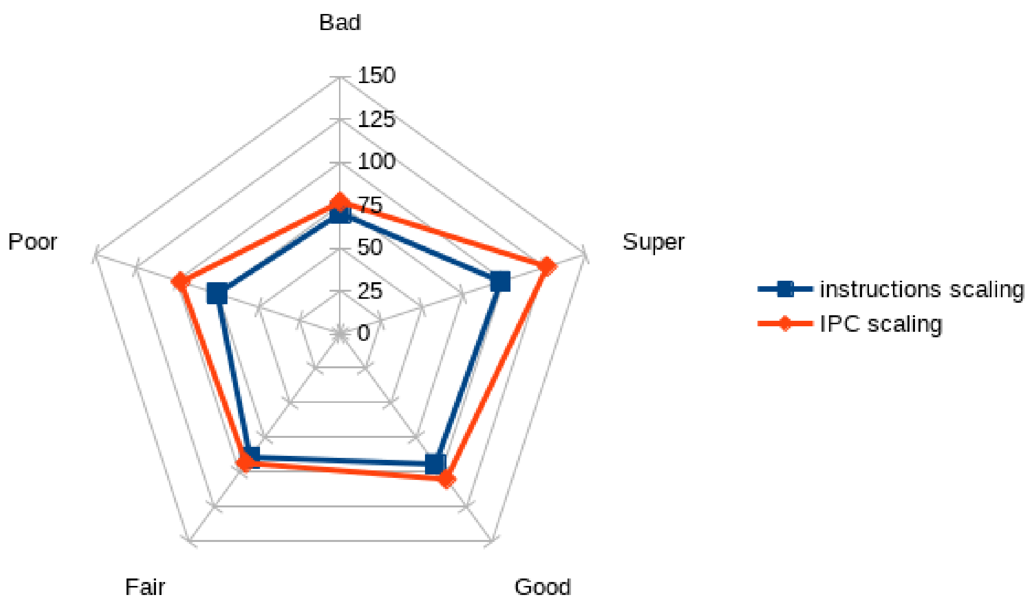

Figure 25: Computation scaling analysis

A third analysis of the efficiencies was done focusing on the top 10 codes with highest global efficiency and the ones with highest parallel efficiency. First thing we can see is that half of the codes are common to both lists. The number of cores goes from few tens to close to twenty thousand, and we do not detect any correlation. Comparing the average value for the parallel efficiency and its child metrics for both sets, the numbers are quite close being lower for the top 10 global efficiency codes. The biggest difference is for parallel efficiency (as it accumulates all their child differences) going down from 94.9% to 88.3% when we focus on the higher global efficiency. But as expected, the main difference is reflected on the computational scaling with a huge difference that compensates the lower parallel efficiency going up from 80.8% to 130.8%. Maybe not so expected, for half of the studies the instruction scaling is higher than the IPC scaling. Based on that comparison, we may infer that to achieve a good scaling it is not only important to work on a good parallelization but even more important to guarantee that there is no code replication and that we can benefit from some IPC improvement when increasing the scale.

We have done a similar analysis with the studies with lowest efficiencies. In this scenario differences are bigger despite six studies are common to both lists. Starting with the number of cores, 40% of the applications with lower scores for global efficiency were analysed with less than 100 cores, while most of the runs with lower parallel efficiency were assessed with a range from one thousand to ten thousand cores. The applications with lower global efficiency that do not have a very low parallel efficiency use to suffer severe problems of code replication except one case where the problems where on a reduction of IPC. Two of the codes also had a bad parallel efficiency while the other two had a parallel efficiency of 70% and 89%. Looking at the codes with worst parallel efficiency, most of them suffer communications problems confirming the insight obtained analysing Figure 23.

4. Findings and recommendations

This section has the goal to characterize the feedback and insight provided by the assessment studies based on the main findings and recommendations suggested as a result of the study. In many cases, when the measurement did not provide the relevant performance data, the recommendation was to investigate further the detected aspect(s). For that reason, and as the previous deliverable already reported globally on the feedback provided in the assessments (most of them were first audit), this deliverable focuses on the insight from the 24 follow-on activities that have been completed during the 3 years of POP2.

Most of the follow-on studies have the goal to analyse a new code version. In many cases that new version was a result of the feedback from a previous assessment. The improvement obtained not only depends on the complexity of the code and its initial performance, but also on the time and effort dedicated by the code owner to implement the improvements. And while some users wanted a quick evaluation after short term modifications, other users prefer to dedicate more time before asking for the second analysis. For those reasons, the improvement goes from a non-negligible 15-20% time reduction in the iterative part to a 2x or 3x improvement. In a high percentage of cases the goal was to improve the scaling with an impact that is more difficult to quantify with a single number. The code improvements that the users implemented faced many different aspects, the most frequent are: work distribution, code granularity, overlapping of computations and communications, parallelization of serial phases, changes in the MPI primitives or changes in the OpenMP scheduling.

Some follow-on studies targeted to analyse the performance on a different platform than the initial audit and in some cases combined with a new version of the code. The most frequent scenario has been to evaluate a new version with GPUs support. In general, for all the studies, during the last year we have seen a significant increase in the number of requests to analyse applications using GPUs. As an example of this trend, during the first ChEESE campaign only one GPU code was analysed, while in the second campaign more than half of the codes were assessed for a GPU platform. The most frequent sources of inefficiencies in GPU based codes are related with: the serialization between the MPI communications (or some CPU computation) and the kernels execution, the ratio between the memory copy and the kernel execution when the scale is increased, and the waste of the CPUs for most of the execution.

Finally, the third scenario for the follow-on assessments is to analyse a different set-up, to compare different code versions or to investigate what is the best number of OpenMP threads for each MPI rank. Due to the diversity of the studies, it is not possible to identify relevant common aspects that characterize them with respect to the previous sets.

Focusing on the findings for the follow-on assessments and considering the 24 studies, we can see that as it was reported in the previous deliverable, the two most frequent topics correspond to load balance and computation scaling while problems of file I/O are much less frequent in the follow-on studies.

With respect to the IPC, in a significant percentage of the studies the analysis identified that the improvement on the IPC scaling was compensating the loss experienced in other efficiency metric (like instructions scaling or transfer efficiency). But there is not a common trend on the IPC evolution with respect to the initial study, in some follow-on assessments we can find cases where the IPC is worse than in the previous study while in others it is improved.

It is important to remark the impact in the performance achieved caused by external factors like the NUMA effect or the system noise that use to be ignored when programming and running a parallel code

5. Assessing CoEs applications

One of the goals that it is common to all the HPC CoEs is to move toward Exascale. The collaboration between CoEs facilitates the path to Exascale, and in the case of POP we can help other CoEs to understand the performance, scaling and bottlenecks of their codes, while the CoEs bring us the possibility to analyse codes ready to run at a larger scale.

As reported in the previous deliverable during the first year of POP2 we contacted all the CoEs to offer 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 we are using and to support them applying the POP methodology to their codes. On that period, only the ChEESE CoE requested us to have an initial POP campaign, whereas other CoEs (ESiWACE, CompBioMed and EXCELLERAT) 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.

During the second year and a half, the collaboration with other CoEs has been increased including multiple assessments' campaigns. We already completed a second campaign for ChEESE to evaluate the impact of the improvements (many of them based on our previous feedback) and a first campaign for CoEC. We are currently working on a campaign for NOMAD2 and we have agreed to schedule a campaign for MAX that will start before the end of the year. Additionally, we continued with isolated requests of code assessments to other CoEs. Finally, due to the Covid-19 there has been a significant reduction of training workshops (details are reported as dissemination).

Figure 26, presents the number of studies done to the different CoEs. We can see that the top 3 correspond to the CoEs for which we performed or we are performing an assessment campaign.

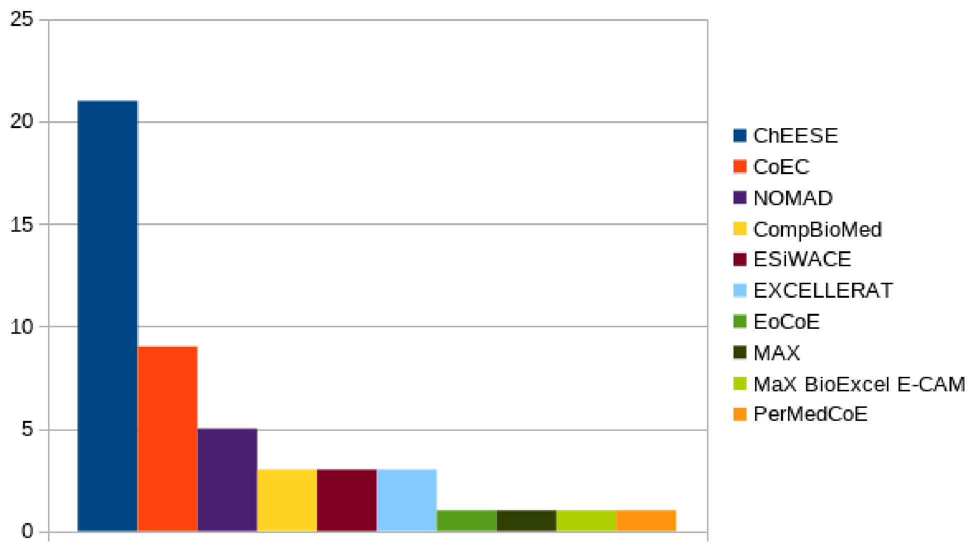

Figure 26: Assessments to CoEs

To provide more details, Table 1 lists the codes from other CoEs that have been analysed or are being analysed by POP2. In parentheses we include the number of studies per code.

CoE	Codes (#assesments)
ChEESE	ASHEE (3), ExaHYPE (1), FALL3D (3), Landslide-HySEA(*), PARODY_PDAF (1), Salvus (3), SeiSol (3), specfem3D (3), Tsunami-HySEA (2), xshells (2)
CoEC	Alya (1), AVBP (1), CIAO (1), CLIO (1), Disco (1), Nek5000 (2), OpenFOAM (1), YALES2 (1)
NOMAD	ABINIT, exciting, FHI-aims, GPAW (2)
CompBioMed	Dealampps (1), HemeLB (2)
ESiWACE	IFS (2), NEMO (1)
EXCELLERAT	Alya (1), AVBP (1), Nek5000 (1)
E-CAM	CP2K (1)
EoCoE	1D-NEGF
MAX	BigDFT
PerMedCoE	PhysyCell

Table 1: List of CoE codes assessed

The study of Landslide-HySEA was cancelled because it shares the core computation with Tsunami-HySEA and they considered the insight provided for that code was directly applicable to the first one.

Looking at the table we can detect three codes that have been analysed within the scope of two different CoEs: Alya, AVBP and Nek5000, reflecting the fact that there are multiple codes that belong to more than one CoE.

Without considering the analysis of the same code for different CoEs as follow-on activities because the user is different, we can see that close to half of the 30 codes analysed have multiple assessments: two assessments have been done for 27% of the codes, and a total of three studies for 20% of the codes. Those ratios are higher than the average value for POP2 assessments, indicating the usefulness of the insight provided by the POP studies to the CoEs.

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 27 plots these values (in logarithmic scale) confirming that the assessments to CoEs' codes have a significantly larger scale. The average number of cores is close to 14x 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. Comparing the average number of cores of all the studies with respect to the average size without the CoEs studies, there is an increase close to 5x reflecting the relevant weight of the studies for CoEs. In fact, close to one third of the studies carried out by POP2 correspond to studies to other CoEs. The increase on the scale can be explained not only because CoEs goal is to run on larger scales but also that CoEs have easily access to large computing systems.

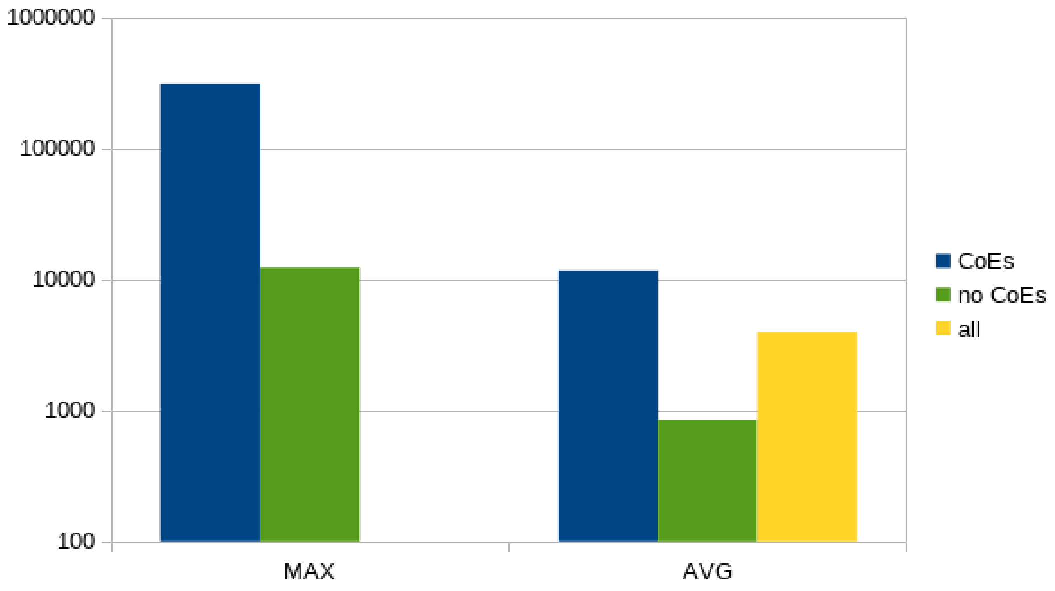

Figure 27: Comparing the scale of CoEs studies (logarithmic)

Acronyms and Abbreviations

- BSC – Barcelona Supercomputing Center
- D – deliverable
- FoA – Focus of Analysis
- 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