D4.4 POP Analysis Report

Document Information

Contract Number	676553
Project Website	www.pop-coe.eu
Contractual Deadline	M30, March 2018
Dissemination Level	Public (except for the annexes)
Nature	Report
Author	Judit Gimenez (BSC)
Contributor(s)	
Reviewer	Sally Bridgwater (NAG)
Keywords	performance assessments, studies statistics, analysis and recommendations

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 "676553".

Change Log

Version	Author	Description of Change
V0.1	J. Gimenez	Initial Draft (uncomplete)
V0.3	J. Gimenez	Completed draft with improvements as suggested by internal review
V1.0	J. Gimenez	Final version

Table of Contents

Executive Summary

This deliverable summarizes the services provided by the Analysis Work Package (WP4) of the POP project. The analysis Work Package is the framework for two of the main services provided by the POP Centre of Excellence: The Performance Audits and the Performance Plans.

The deliverable describes the work done since the last deliverable (Sept 2017) and characterizes the cases analysed during the whole project lifetime. In this final document we include statistics on timings of the analysis and our performance implementing the WP4 services. The Performance Plans are summarized with respect to the findings and recommendations provided to the customers. The report also includes some recommendations for tool developers mainly based on the collected experiences from the POP consortium. The annexes of the deliverable include a list of the services provided during the first two years of the CoE and the reports produced during the last six months. Due to confidentiality issues of some users, those annexes are not public.

1. Introduction

This deliverable describes the work done during the last six months of the project and characterizes the Performance Audit and the Performance Plan services carried out by the POP CoE partners during the whole lifetime of the project. The services are available free-of-charge to developers and users of parallel codes with the objective of providing useful insight regarding the behaviour of their applications.

As of 12th March, 2018 (at the time writing this deliverable), we had 188 requests for WP4 services. This means 38 new requests since D4.3 was written in early September 2017. For the total 188 requests which includes all of the work since the project started, 141 correspond to Performance Audits and 25 to Performance Plans. The missing 22 requests have been cancelled because the user was overloaded, they did not reply to our mails for an extended period of time, they moved to a different institution or company, the tools do not support analysis of the code, or it was not possible to agree and sign an NDA on the terms required by the user. 21 of the cancelled requests were Performance Audit studies and one of them for a Performance Plan. Comparing with the numbers reported in D4.3, during the last 6 months of the project, we had 25 new Performance Audits and 4 new Performance Plans.

As this is the last deliverable of the work package we also evaluate the performance of the assessments work package analysing the timings for the services and the throughput we achieved.

2. Performance Assessments

This section describes the evolution of the requests and their status as well as their distribution within the consortium.

Figure 1 plots the evolution of the POP assessments during the reporting period and Figure 2 plots its evolution since the project start.

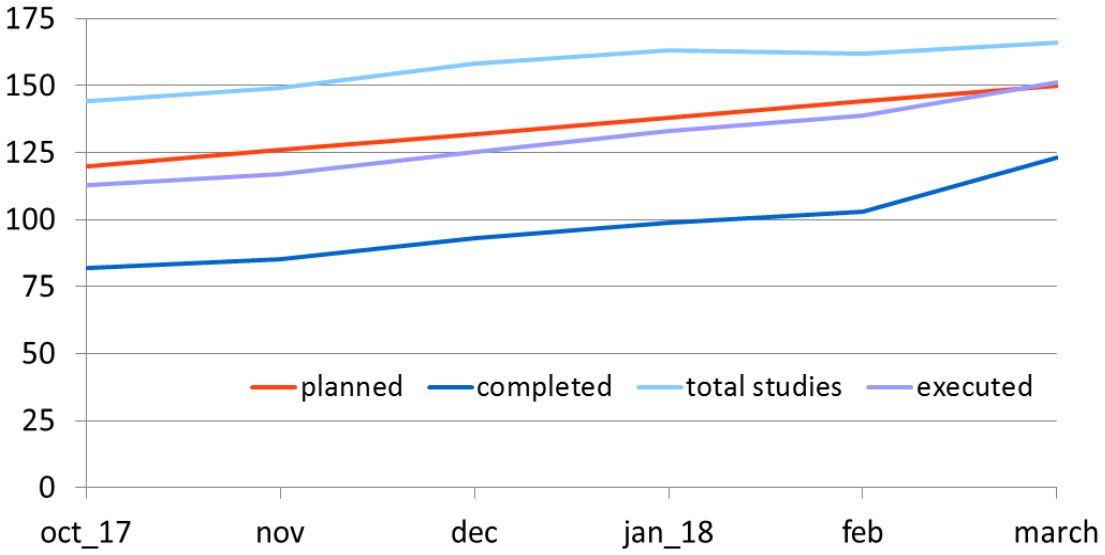

Figure 1: POP assessments evolution w.r.t plan (last six months)

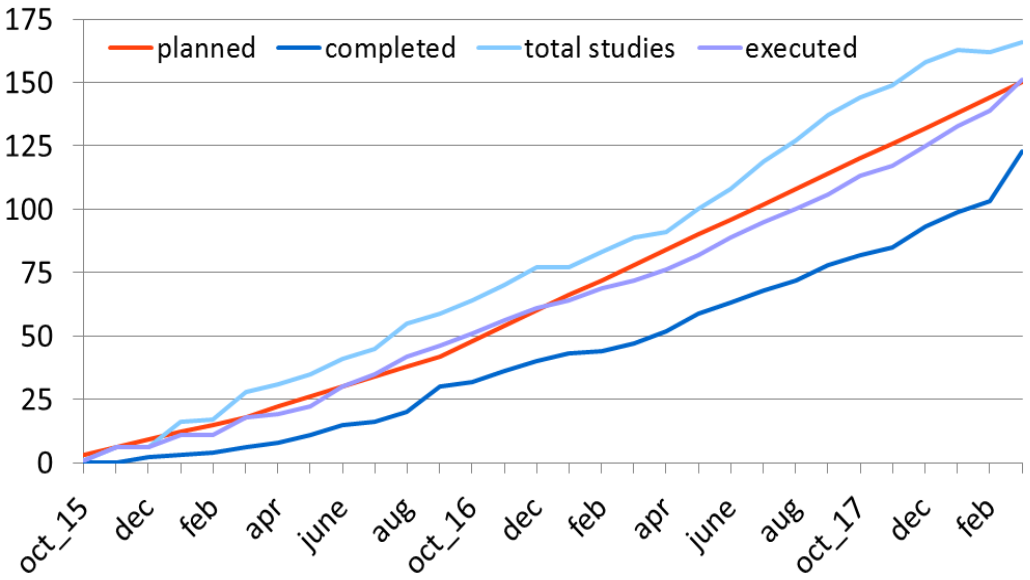

Figure 2: POP assessments evolution w.r.t plan (full project)

Comparing the total number of studies with respect to the plan, we can see that except for the first months of the project, the total number of studies has

been always higher than the planned value and the gap between them has been increasing (with few months as exceptions).

On the other hand, the number of assessments completed is always significantly lower. We consider the amount of studies planned to be a very ambitious target. Except for the first months 6 studies were required each month, implying a very tight schedule as we allocated one month for the Performance Audits and between one to three months for the Performance Plans.

As we justified in D4.2 and D4.3, one of the reasons to have a delay completing the studies is that most of them require more time than initially planned. This is because of delays mainly caused by the POP user. We have found that these delays are much more likely during the initial phase of the study (due to delays in providing sources or traces, in signing NDAs or in specifying the input cases). For this reason, we think that the executed assessments, including both completed and progressing studies, serve as a much better indicator of progress. We can see that in Figure 2 the executed metric is close to the planned value, although since December 2016 year it has been slightly lower than the planned value but that has been recovered in March this year.

During the POP project we had to cancel 22 services. Figure 3 plots the distribution of the cancelled studies compared with the total number during the project lifetime. We can see that as percentage the ratio has been always quite small despite there has been a significant increase of cancelled services during the last months. The reason was to be able to confirm to WP3 the number of required services to reach the project goal of 150 assessments.

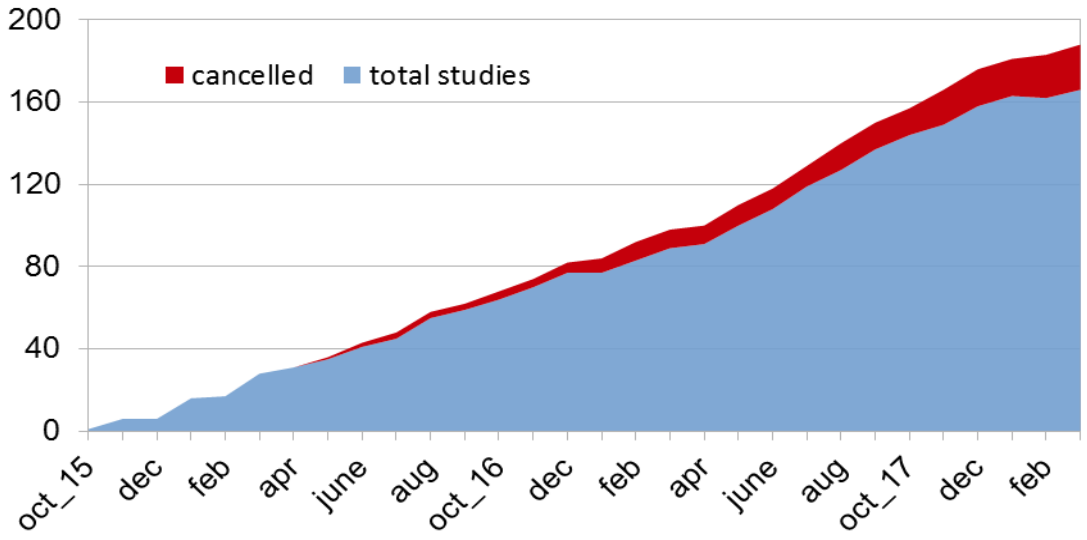

Figure 3: POP assessments (cancelled vs. total)

At the time of writing this deliverable, the current distribution of the 166 assessments is as follows: 123 completed, 4 being reported to the user, 24

doing measurements of the code and analysing the data and 15 are not yet started requests (either new requests or still waiting some input from the user).

Figure 4 plots the assessments distribution per partner for each of the states**.** As not all the partners have the same effort and budget, we agreed on a weighted value for the total number of studies per partner. The plotted line is an estimator of the planned value per partner considering the resources assigned in WP4.

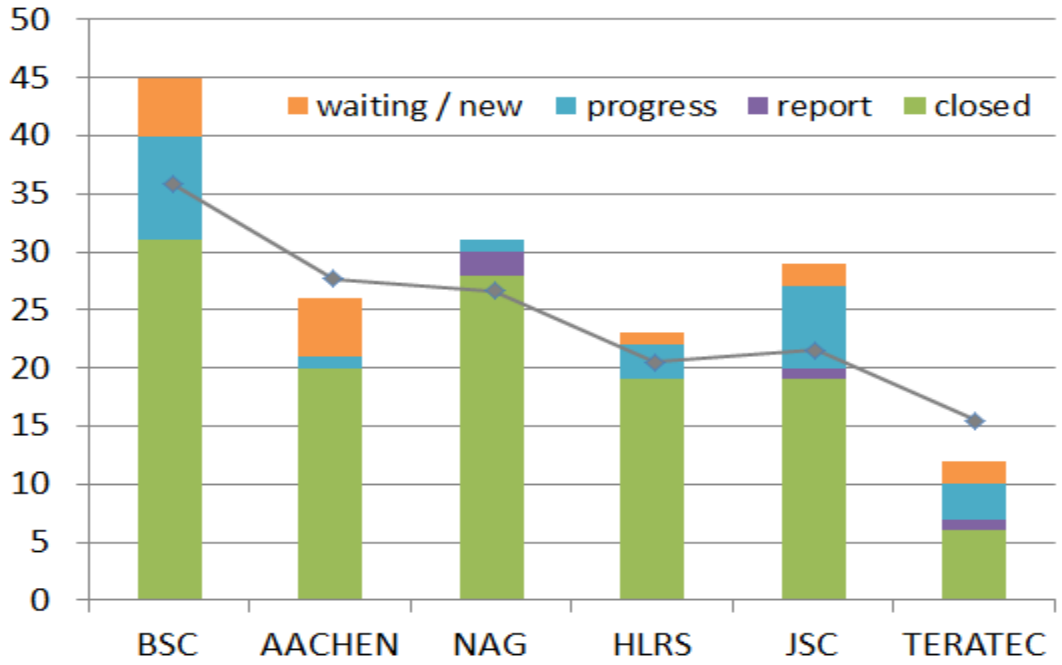

Figure 4: POP assessments per partner (plotted line represents the planned value)

We can see that for most of the partners the total number of assessments is higher than he planned value, with the exception of AACHEN and TERATEC/CNRS. The value for AACHEN is very close to the target and the reason for being lower is due to the cancellation of some tickets during the last months. In the case of CNRS (a third party from TERATEC) they started to work on WP4 on 1 October, 2016 and they have not been able to assume the work required to recover from the initial delay. Nevertheless, taking into account that they have been working for only one year and a half, their percentage fits the expected rate.

Even though the project was very ambitious with respect to the number of services, NAG has completed the assigned number of assessments. In the case of BSC, JSC and HLRS the planned value is achieved if we include part of the studies that are currently in progress. As we have the compromise to keep working with our own resources after the project ends, we expect to be able to present a number of completed studies closer to the original target during the final project review.

3. Performance Audits

The Performance Audit is the primary service and may be considered a kind of health check for the codes. There is no need for the customer to feel the code is running inefficiently for them to request POP to audit its performance, and indeed a Performance Audit is recommended prior to any planned modification. Codes are diagnosed based on a set of well-defined efficiency metrics and the efficiency achieved on different aspects (e.g. parallelization, load balance, IPC, data transfer) against which we recommend areas for improvement. Although we always try to cover a minimum set of common analyses in every Performance Audit, we can also tailor the Audit according to customer needs and / or topics of interest such as serial code performance, scalability, or communications.

To extract insight from the experience of the assessment services in the POP project, we have updated the statistics we presented in D4.3 to cover the full POP framework. There have been no major differences with the increase of the population, but we remark if there have been differences with respect to the conclusions in D4.3.

1 Performance Audit characterization

This section characterizes the audit requests with respect to the performance service required, the profile of the user (e.g. country, sector) and the profile of the code (such as the programing model and language used). This characterization is applied to the 141 audits, as this information is mostly collected on the request service form. Some of the metrics also include the cancelled studies increasing the population to 161.

.....1.1 Performance service

The web form to request an audit offers the alternative to select a focus for the Performance Audit. Figure 5 characterizes the requests received with respect to the main focus selected by the user. There have been no changes from D4.3: 60% of the requests selected the basic performance check and 17% asked for us to identify areas of improvement. Parallel efficiency is identified as the main concern, followed by scalability. None of the requests selected the communications option (maybe considering it part of scalability or parallel efficiency).

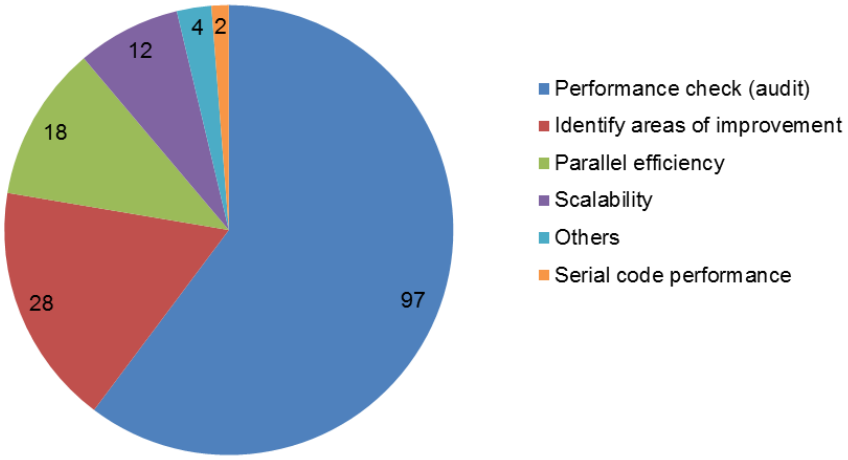

Figure 5: Performance service requested

In the detailed service questionnaire, we ask users if they can use their own platform or they would like to use PRACE/POP resources. From the 118 users that replied to this question, 58% selected to use their own resources, and 41% of them also wanted to install the tools and collect the performance data. From the 42% that wanted to use PRACE/POP platforms, the percentage that wanted to collect the data sharply reduced to 20%. We should clarify that within the 46% of the users that wanted to use a PRACE platform there were some existing PRACE users, so that it is their typical production machine.

Figure 6 plots the answers collected to the question "How did you find out about POP?". Except for very few cases that specified the website or the news, most of the cases are concentrated on a POP partner, word of mouth and other. We identified multiple cases where the contact was done through a POP partner but because the person filling the form was different from the one previously contacted; the classification is either other or word of mouth. Often people used other to specify the name of their POP contact.

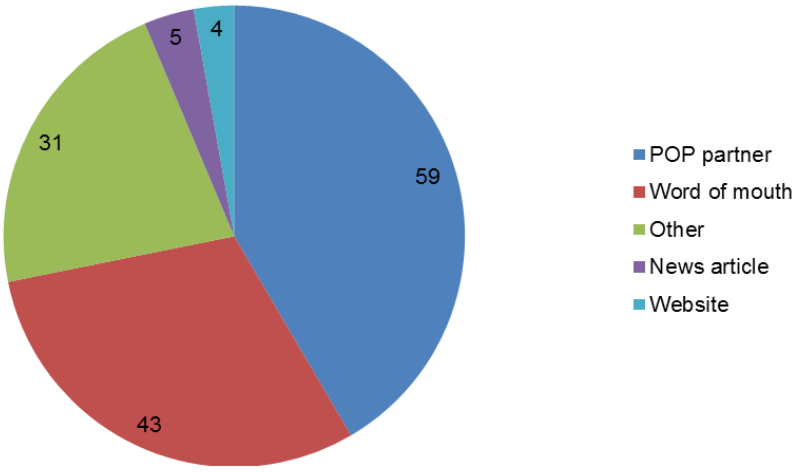

Figure 6: How did you found out about POP?

.....1.2 Code

This section tries to characterize the codes WP4 has been working with. From the 116 answers collected from the users, 67% of the codes were not analysed prior to this POP Performance Audit. This indicates a large percentage of codes that benefit from the POP free service to obtain a first analysis of their code. 112 users replied to the question about the scaling approach and 70% use strong scaling. Although that percentage may not be a surprise; it implies there may be more scalability issues. Finally, we inquired about the codes that are open source, so a bigger community may benefit if some of the POP recommendations are implemented and 61 of 118 studies are for open source codes which is just over 50%.

In Figure 7 we analyse the distribution of the requests with respect to the scientific/technical area of the code as specified by the user. As we reported in the deliverables of the first and second years, the most dominant areas are Engineering, Physics, Earth/Atmospheric and Chemistry summing up the same percentage than the second year (77% of the requests). The number of requests from relatively new fields in the HPC sector like health, data analytics or finance sums up less than a 5%.

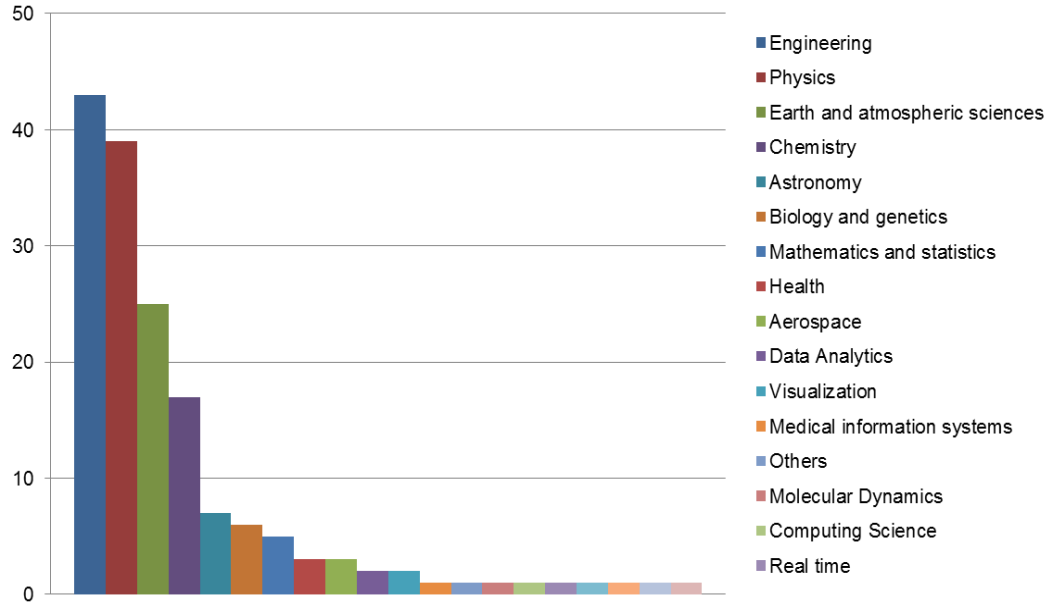

Figure 7: Code scientific/technical area

Figure 8 characterizes the audits with respect to the parallel programing model. Due to the large number of combinations, the statistic has been simplified grouping them by levels. The threads level includes OpenMP, Pthreads and OmpSs and it is dominated by OpenMP (93%). The accelerators level includes both CUDA and OpenCL with 80% of the cases corresponding to CUDA. Finally, others include programing models like Python multiprocess, MAGMA, Fortran co-arrays, Charm++, TBB, GASPI or Matlab.

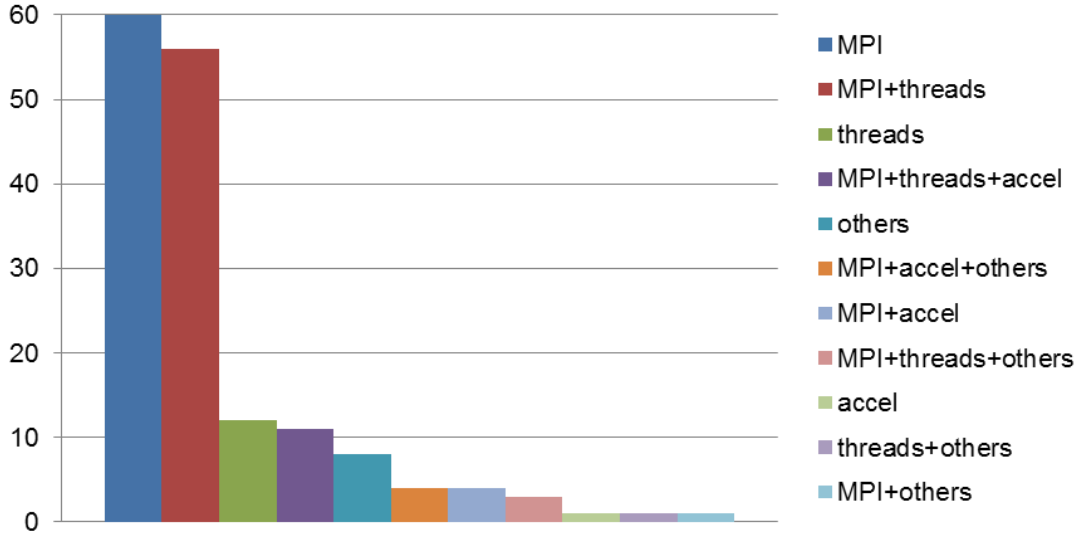

Figure 8: Parallel programming model

As expected, the codes are dominated by MPI followed by MPI+OpenMP with a similar ratio than previous years (close to 71% of the requests). Although 16% of the codes supported accelerators, in most of these cases the user requested us to focus our analysis without the CUDA capability.

There are also no surprises with respect to the most dominating programming languages (see Figure 9). Fortran, C++ and mixed Fortran and C represent 72% of the codes (very similar to previous years). Pure Fortran or Fortran mixed with other languages is used in 107 of the 161 codes. As we already identified, Python-related requests are higher than initially expected even though the ratio has decreased a little bit in the last period (Python contributes in 19 of the codes). In many cases, Python is used for serial pre/post-processing as part of a workflow, and not for the parallel computation. Finally, in others we classify 4 studies using TCL, Matlab, Perl and Octave.

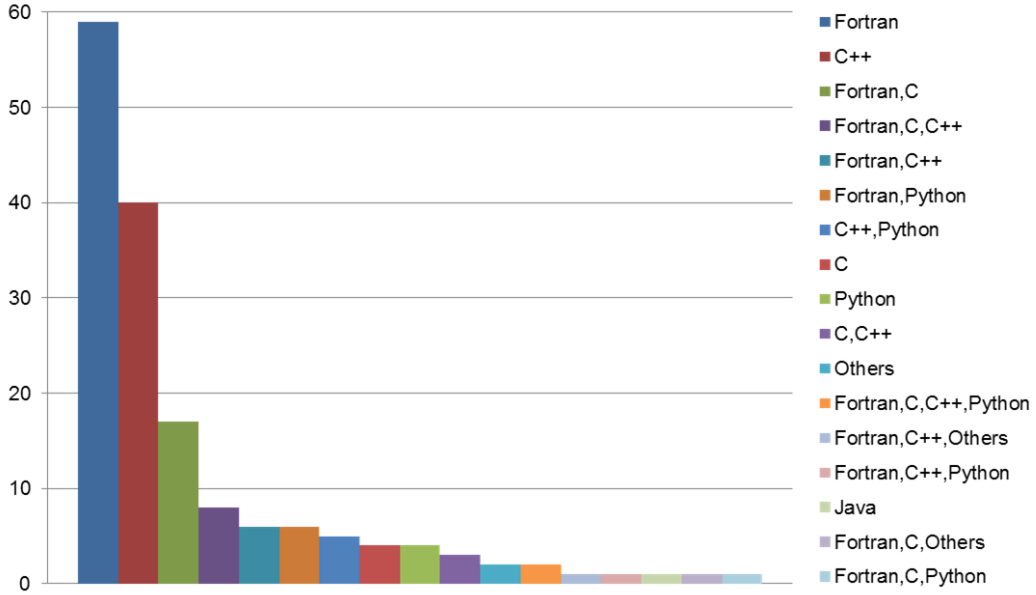

Figure 9: Programming language

Figure 10 correlates the programming language with respect to the code sectors. In this plot we eliminated both programming languages combinations and sectors with only one or two occurrences. As in the previous reports, Fortran concentrates in the traditional sectors of engineering, physics, chemistry, earth/atmospheric applications and astronomy. These sectors are the only ones with enough population to consider the distribution between languages may be representative. Pure C++ codes are a significant percentage in physics and engineering with few representative cases in most of the other sectors (except astronomy and aerospace). Finally, few Python cases appear in most of the sectors (except astronomy, biology and health).

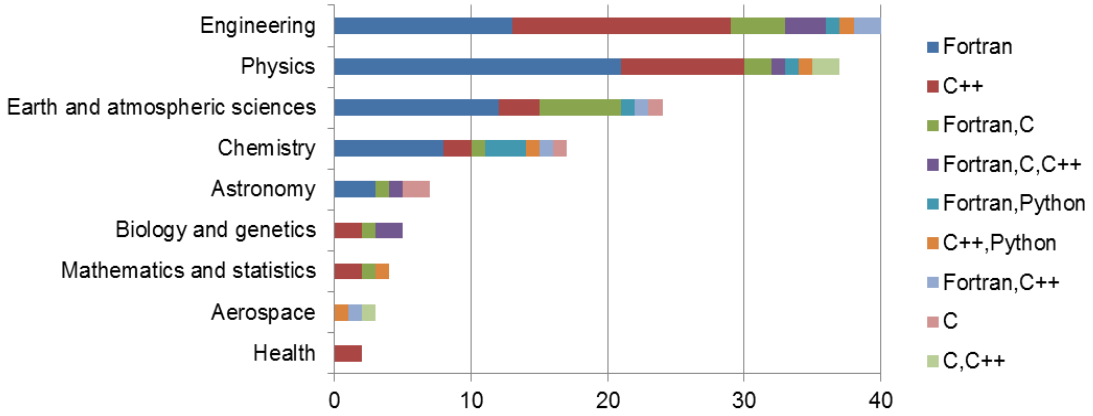

Figure 10: Programing language used on the main code sectors

In a similar way, Figure 11 correlates the parallel programming model to application sectors. As in D4.3, in Engineering, Physics and Earth/atmospheric sciences, the codes' most frequent paradigm is MPI or MPI+thread. There Chemistry more frequent use is pure MPI codes. Pure thread parallelisation has few occurrences in 6 of the 9 sectors. Finally, the codes with support to use accelerators are also spread on 6 of the sectors being significantly more relevant in Chemistry.

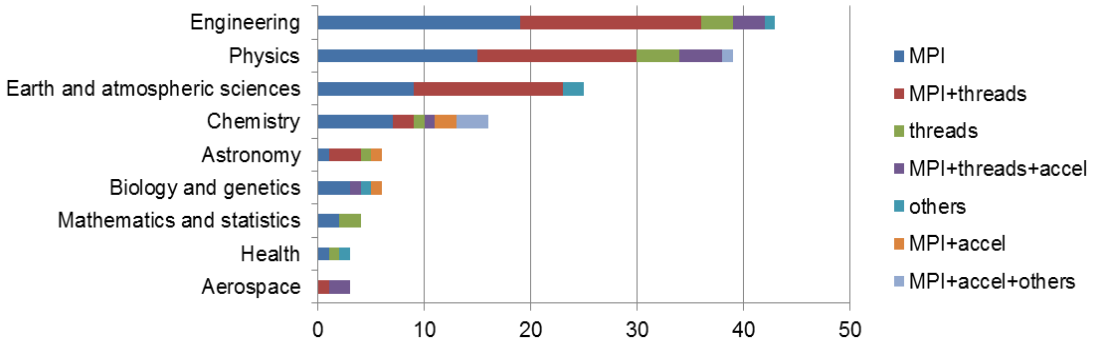

Figure 11: Parallel programing model used on the main code sectors

To complete this section, we looked at the usage of resources. As part of the second questionnaire, we request users to specify the number of processes they allocate to their jobs as well as the maximum number of cores where

they consider the code has good performance. Figure 12 correlates the number of cores customers used in production runs with the code sectors. In the figure we include the minimum, average and maximum values reported, classified per sector and including only the sectors from the previous analysis, and including as "All" (dashed line) the metrics for all the studies we have collected this data. The scale of this plot is logarithmic base 2 due to the large range of values.

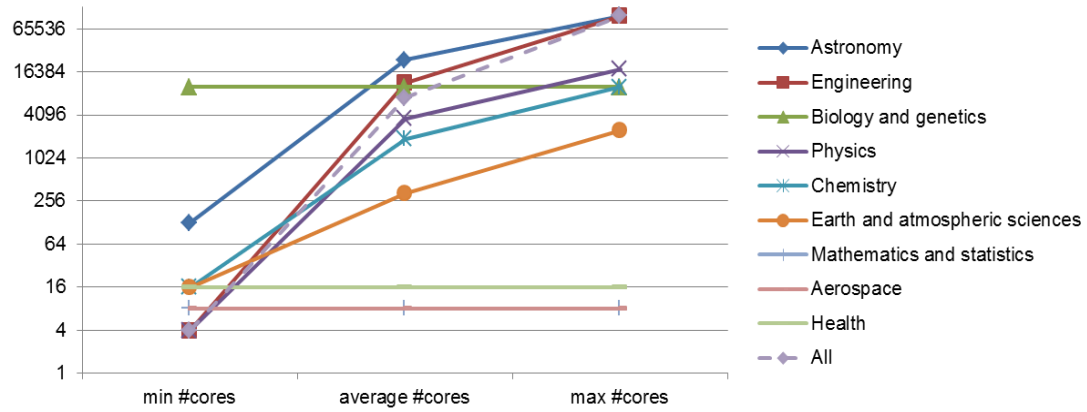

Figure 12: Number of cores used per code sector

As a difference with the previous report, now 4 of the 9 sectors have a constant number of cores. These sectors have a small population between 3 and 6 codes, but for instance the population of Astronomy is 7 codes and we already detect variability between 128 and 10000 cores. As in D4.3 the largest case for most of the sectors ranges between a few thousand and hundred thousand cores.

Figure 13 correlates the number of cores the users consider they have good performance, with the code sectors. In the figure we include the minimum, average and maximum values reported classified per sector and including only the sectors from the previous analysis (except the Health sector that none of the 3 users filled this field) and including as "All" (dashed line) the metrics for all the studies we have collected this data. The scale of this plot is logarithmic base 2 to cover the large range of values. The Y scale is the same than the previous figure to facilitate comparison.

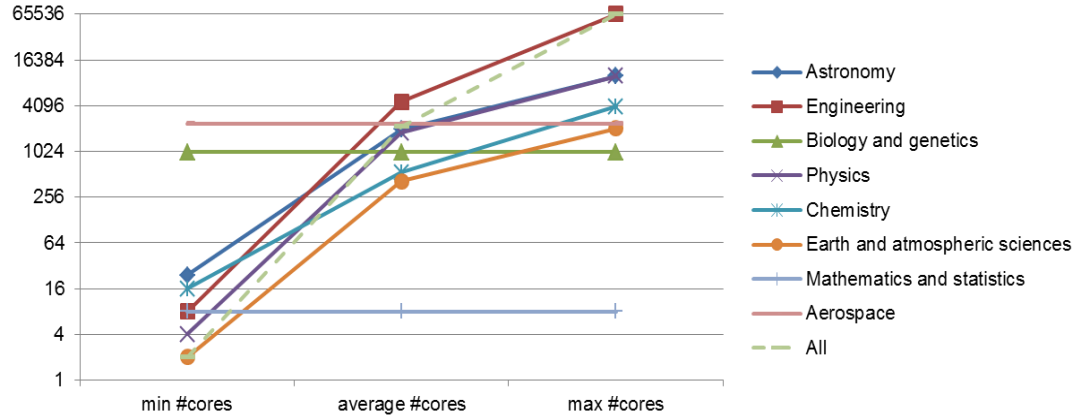

Figure 13: Number of cores with good performance per code sector

In general, the values of this plot are smaller than on the previous figure, showing the users typically execute with a higher number of resources even though they feel the efficiency is not good. For more easy comparison we plot the difference between the two previous figures in Figure 14. We subtract the number of cores that achieve good performance (as perceived by the user) from the number of cores typically used in production. A positive number means the executions are done with configurations the user consider inefficient and negative values means the user keeps in a range where they consider the code to be performing efficiently.

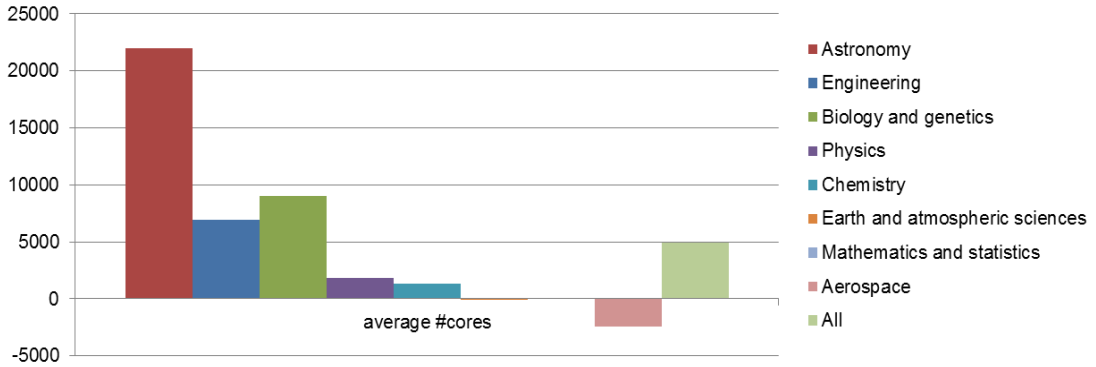

Figure 14: Typical run vs. performing run (per code sector)

As already detected comparing the previous figures, most of the sectors reported to us that they typically run with scales that they consider do not achieve a good performance. On one side this indicates POP services may improve this situation supporting them to identify the bottlenecks to optimise the applications, but on the other side it seems to indicate that most of the users are happy with the small extra gains even though they are aware they run inefficiently. Only few sectors (Aerospace, Mathematics and Earth / atmospheric sciences) select the scale of their executions taking into account their perception of the achieved performance.

.....1.3 User

In this section we characterize the POP users. The first classification is with respect to the organization making the request and it is reported in Figure 15. Similar to previous reports, the academic and research profile dominate the requests served by the CoE. Industrial users represent 24% of the audits (previously 23% and 26%), and they are mainly SMEs. We recognize that it is more difficult to engage industry because of NDAs or other restrictions, and in the specific case of SMEs sometimes their difficulties to assign personnel. Typically, there are more delays when working with industry and around 30% of the cancelled studies were from companies that despite their interest were not able to allocate personnel for a long period of time.

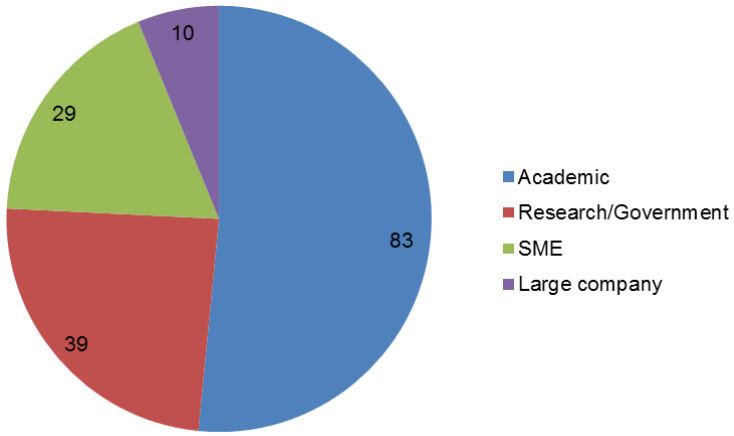

Figure 15: User profile

Figure 16 characterises users based on their role with respect to the code. Similar to previous reports, 70% of the requests are from core developers of the code; this guarantees a potential higher impact because it is in their own hands to follow the recommendations provided in the audit. 14% of requests are from users of the code, where POP service is a means to get insight on the potential performance problems to later check with the code developers.

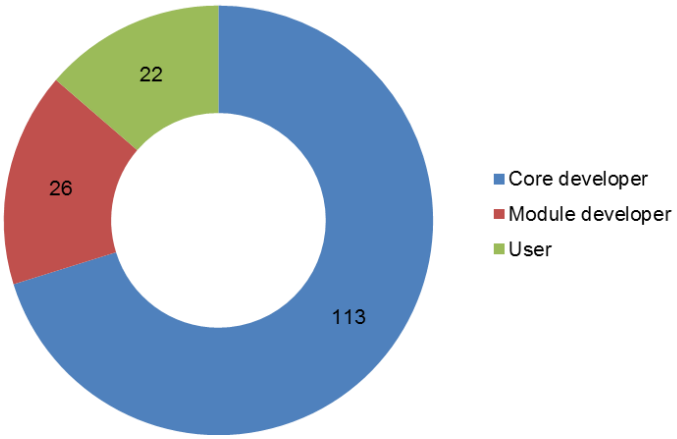

Figure 16: Contribution to the code

We looked at the correlation of the contribution with respect to the user profile, showing a very similar distribution, except for the ISVs where all the requests were from core developers as can be expected.

Figure 17 correlates the code area with respect to the requesting user institution. Focusing on the sectors with a relevant number of studies, we can see that almost all the profiles are covered in all the sectors (except for the large companies). We consider it is important to reach both industrial and academic/research profile on the more relevant sectors.

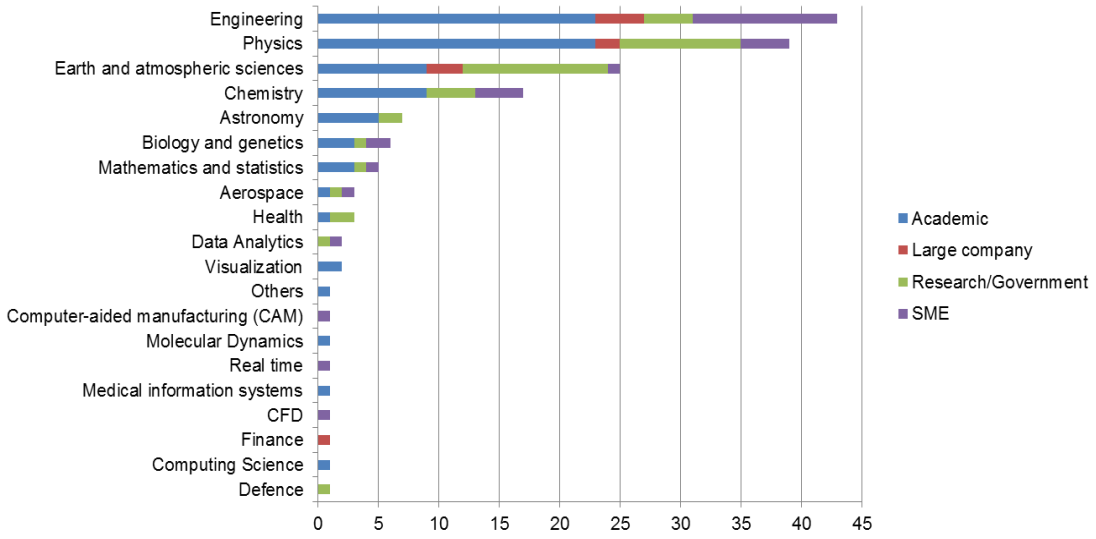

Figure 17: Institution profile per code area

In this last report we include also plots to look at the distribution over the different code sectors of the Large companies (Figure 18) and SMEs (Figure 19).

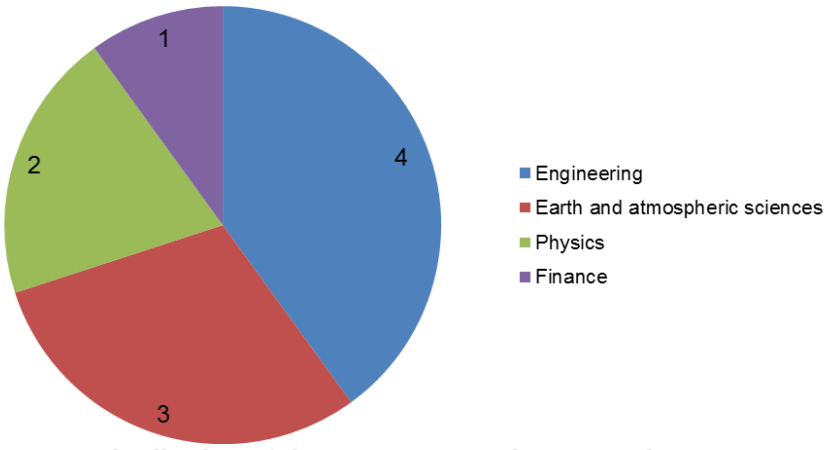

Figure 18: Distribution of the Large companies per code area

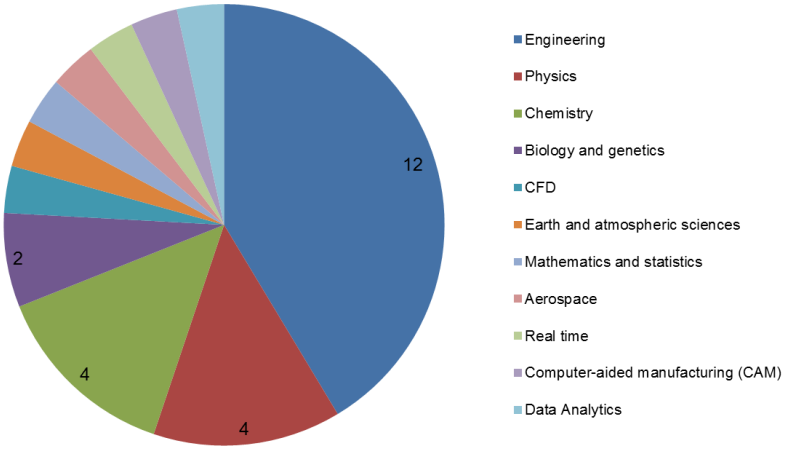

Figure 19: Distribution of the SME per code area

In both plots we can see that the largest percentage corresponds to the engineering sector. But while the large companies are concentrated in four sectors, the SMEs are spread over a very wide range of sectors.

Figure 20 summarises the answers collected to the question about users previous knowledge of performance tools classified by the user profile. 74% of the users did not have previous knowledge about performance tools. This ratio goes up to the 82% of the codes that were not analysed before and down to 55% for the codes previously analysed. By user profile, we can see that research centres have a higher percentage of knowledge (36%) and the lowest ratio is in the SMEs (17%). Surprisingly, the ratio of knowledge in large companies is a little bit larger than in the academic users (29% vs. 25%), however, the number of large companies may be too small to extract the trend.

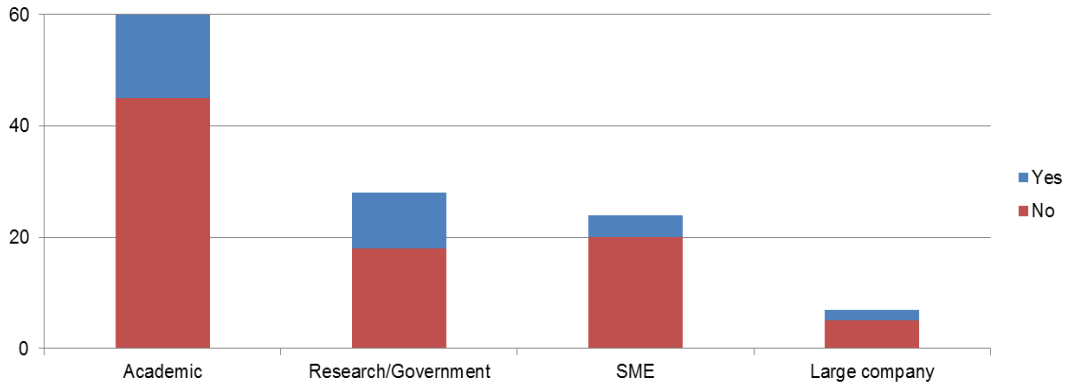

Figure 20: Previous knowledge about performance tools per user profile

Figure 21 plots the country of the requesting user's institution. Although most of the requests are from POP partner countries (United Kingdom, Germany, France and Spain), 25.5% of the requests are from other European countries or countries associated to the EU H2020 programme (similar ratio than reported in D4.2 and D4.3). Also, as in the previous reports, the highest percentage corresponds to United Kingdom with 34% of the audit requests.

Correlating the country with the user profile, we identified that the country with the higher number of SMEs is United Kingdom (9) and with the higher number of large companies is France (4). 34% of the SMEs and 20% of the large companies were from countries external to the consortium. With respect to the academic users, 69% of them were either from United Kingdom or Germany and 26% from countries outside the consortium. Finally, the research centres have been mainly spread between the 4 consortium countries but 20% of them were from other countries.

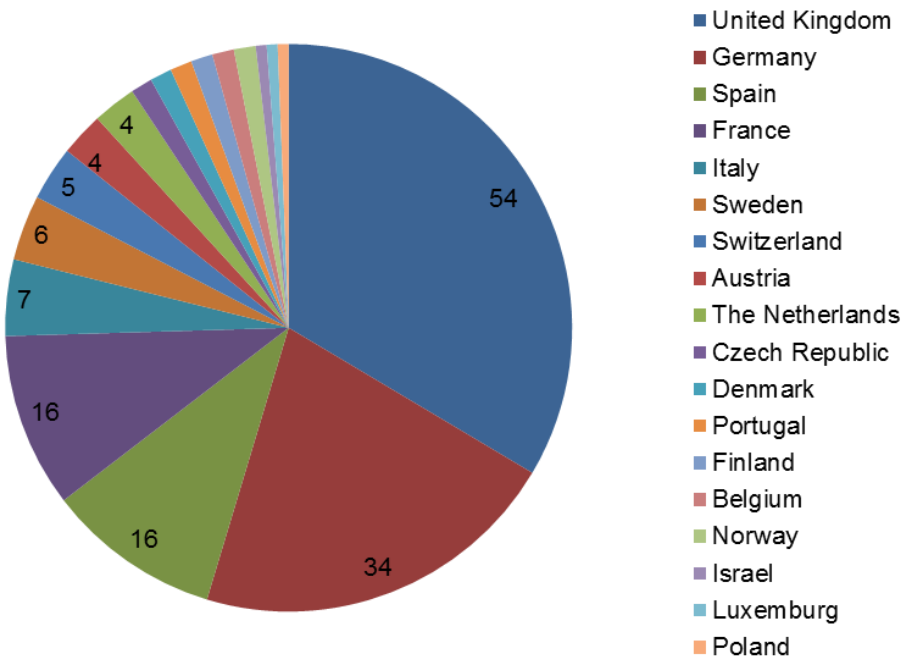

Figure 21: Institution's country

Finally, most of the requests (89%) are from institutions external to the POP consortium. The 16 internal requests correspond to studies carried out during the first year or the last period. All of them were from different departments from the POP partner, 5 of them correspond to RWTH Aachen, 2 to HLRS, 4 to JSC, 3 to BSC and 2 to TERATEC/CNRS.

2 Performance Audit results analysis

This section summarizes the results from the completed Performance Audits. When writing this deliverable, there are 106 completed audits, however, some of the data is not available for all of them.

Figure 22 characterizes the assessments with respect to the number of cores in the largest execution run the user is satisfied with the performance, and the largest run audited.

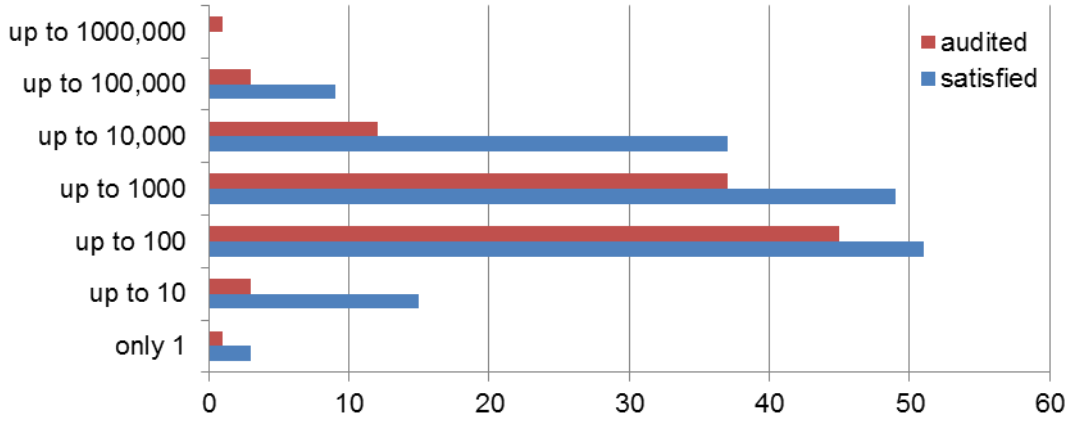

Figure 22: Code and audits scalability (#cores)

The data collected included 164 entries about the largest performing run (we included data from not completed audits to increase the population) and 102 for the largest runs audited. As reported in the last deliverable in most of the cases we have been looking at runs between ten and one thousand cores that also have the highest ratios for the number of cores where the user considers the performance is still good. As a difference, there has been a significant increase in the percentage of codes the user consider is well performing with the range of a few thousand cores. There has been also an increase of large runs audited, the largest one with 239,615 cores.

Figure 23 compares these two metrics for the cases where we collected both values. Due to the large range, the Y scale is on logarithmic base 2. The dotted lines connecting each series are included to facilitate the readability of the plot.

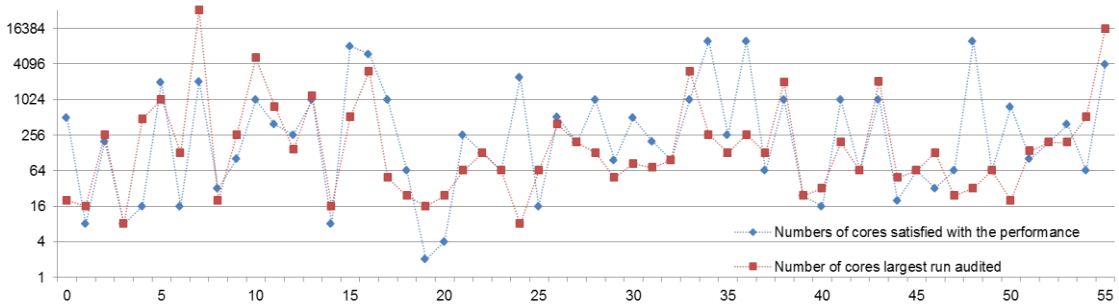

Figure 23: Code and audits scalability comparison

As in the previous report, only for few cases both metrics report the same value. There are cases where we analysed smaller executions and cases where the scale was significantly increased. The selection of the number of cores analysed is always determined by the user. In this sense it is important to remark that while some users wanted to use POP to analyse their most frequent execution, or an execution they consider it was efficient, other users decided to look to a worst-case scenario analysing a configuration where they experience performance problems.

Figure 24 classifies the efficiencies we have observed in the studies. We measure the parallel efficiency considering the time the application is executing user code over the total time (considering that time spent in the MPI or OpenMP parallel runtimes is an overhead that the code has to pay to run in parallel) and it is composed from factors like load balance, transfer and serialization. For the studies where we have multiple core counts, we also measure the IPC and instructions scalability and their impact on the computational efficiency. Finally, the global efficiency combines parallel and computational efficiencies and it corresponds to the time efficiency measured when computing the achieved speed-up. In the studies where we had multiple executions, we selected the data of the largest run analysed.

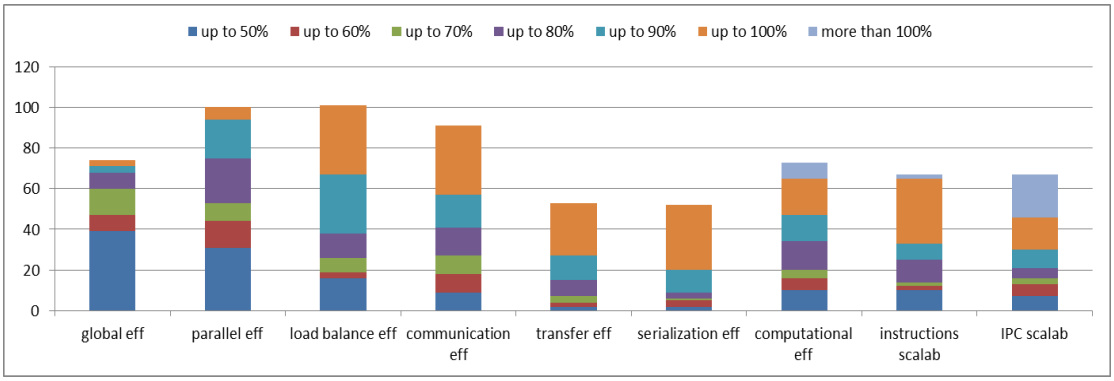

Figure 24: Efficiencies measured

We consider very good and good efficiencies ratios higher than 90% and 80% respectively. In these cases, although there is some space for improvement, there is not an important need. On the other hand, metrics like computational efficiency or IPC efficiency can be higher than 100%. The most frequent reason is when using strong scaling and the IPC increases when more processes are allocated because the data per process is reduced improving cache usage.

Due to the high difference on the number of instances measured for each of the efficiencies, Figure 25 plots the ranges of efficiencies from the previous figure as a percentage.

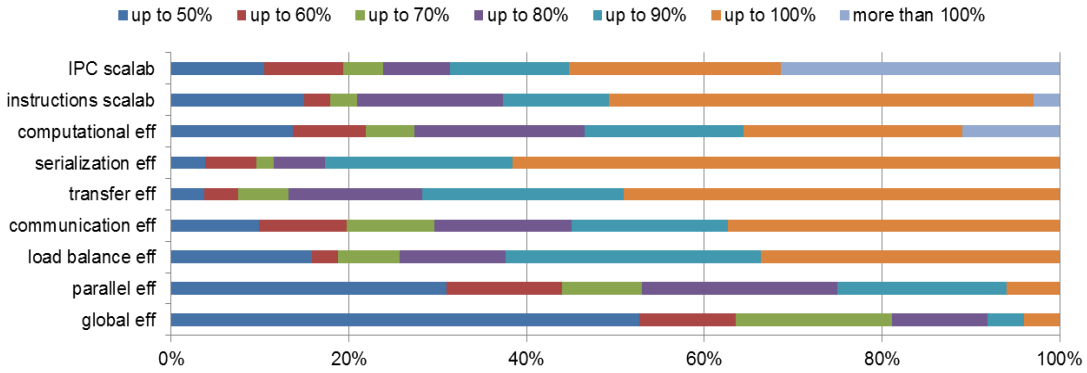

Figure 25: Efficiencies as percentage

If we analyse the parallel efficiency as it is the metric we can compute for almost all the studies, the percentage of codes analysed that require improvement to run efficiently in parallel has been increased from 66% to 75%. Close to one third of the codes have a parallel efficiency below 50% meaning that less than half of the time is dedicated to the code computations. Considering that the analysis has been focused on a region representing a key kernel, generally omitting initialisation and file I/O where efficiencies are lower, these codes are running really inefficiently. With respect to the component of the parallel efficiency, load balance reports significantly worst values than serialization and transfer, but the combination of both in the communication efficiency reports a higher percentage of codes with efficiency between 60 and 80%.

When we analyse the global efficiency, the values are even worst, indicating problems scaling the computations. Conversely, we can see that for close to one third of the codes the IPC improves when increasing the scale (more than half of the codes had an IPC bigger than 90%)

The efficiency model is computed as a hierarchy. The next plots (from Figure 26 to Figure 29) correlate each efficiency with its immediate lower level. For instance, in Figure 26 the global efficiency is correlated with both parallel efficiency and computational efficiency to identify which of the factors has a higher impact. In all of these plots we only included the cases we have the value for all the plotted metrics. As in Figure 23 the dotted lines connecting each series are included to facilitate the readability of the plot.

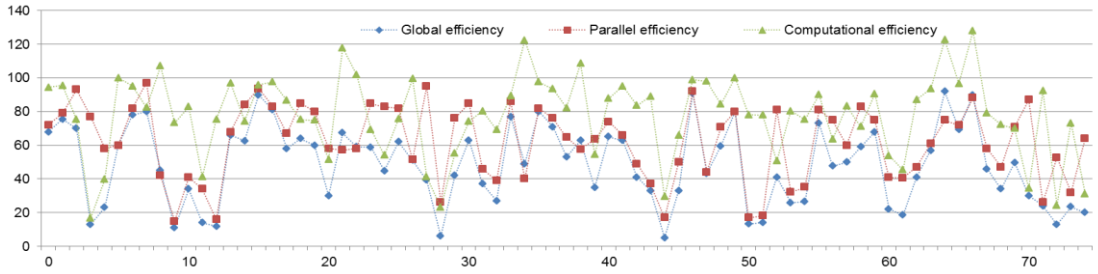

Figure 26: Global efficiency analysis

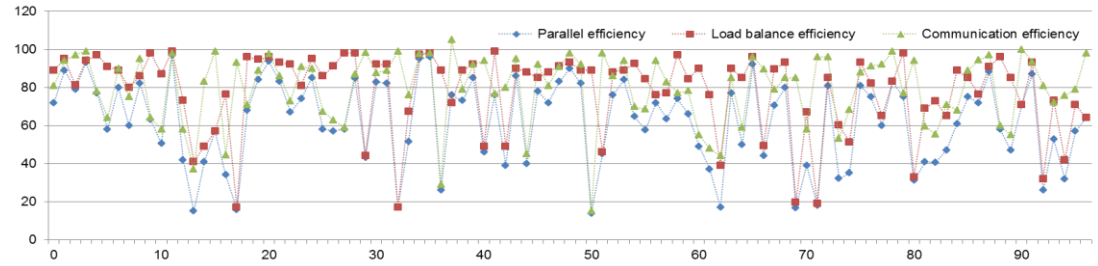

Figure 27: Parallel efficiency analysis

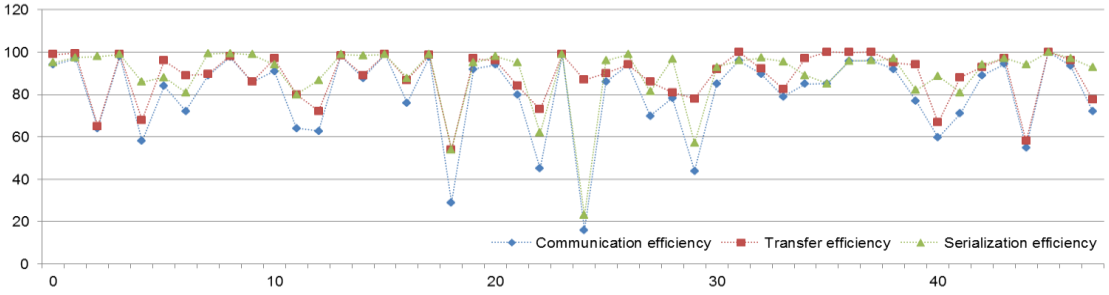

Figure 28: Communication efficiency analysis

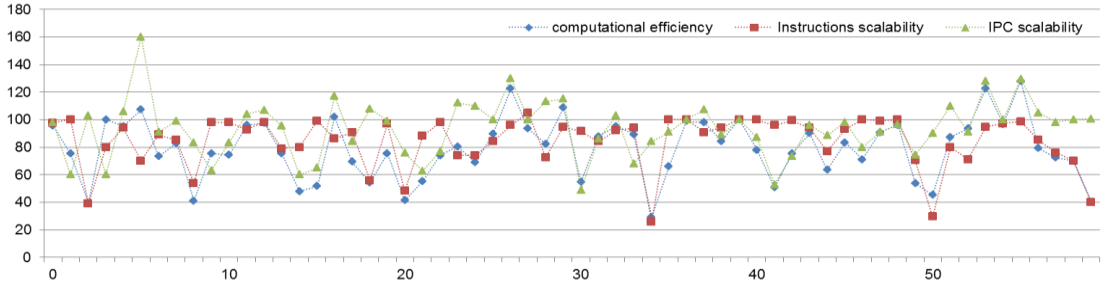

Figure 29: Computational efficiency analysis

Looking at these plots, we can observe:

- Very frequently parallel efficiency is lower than computational efficiency.
- Global efficiency lower than 40% use to be correlated with poor parallel efficiency, in some cases with combination of poor computational scalability.
- Parallel efficiency is usually worse when it is caused by load balance (with some exceptions). When it is due to communication, the impact is usually smaller. That may be because users are more conscious about the impact of communication than the impact of load balance.
- Communication efficiency is bigger than 80% in a large percentage of the codes.
- Loss of communication efficiency is mostly caused by data transfer (high volume of data or high number of communications with respect to the computation).
- Worst cases for communication efficiency (lower than 50%) have serialization problems.
- In many cases lower computational efficiency is caused by poor instructions scalability. In strong scaling, this reflects code replication.
- In some cases, computational efficiency benefits from an increase in the IPC when increasing the scale that compensates the increase of instructions or even the decreasing parallel efficiency.

We analysed at coarse level, the IPC achieved by the applications when this metric is available from the performance data. As boundary we used a value of 1 to consider whether the IPC is acceptable or low. From our experience, most of the codes have to be able to achieve this value or even higher values above 1.5 depending on the specific machine architecture. The average minimum IPC measured is just around that boundary of 1 instruction per cycle. The average IPC measured is around 2.6.

To complete this section, we grouped the efficiencies and the IPC achieved with the parallel paradigm and the programming language used to identify if we can detect some correlation. In Figure 30 we combine the main parallel programing models with the efficiencies factors related to the parallelization as well as the global efficiency.

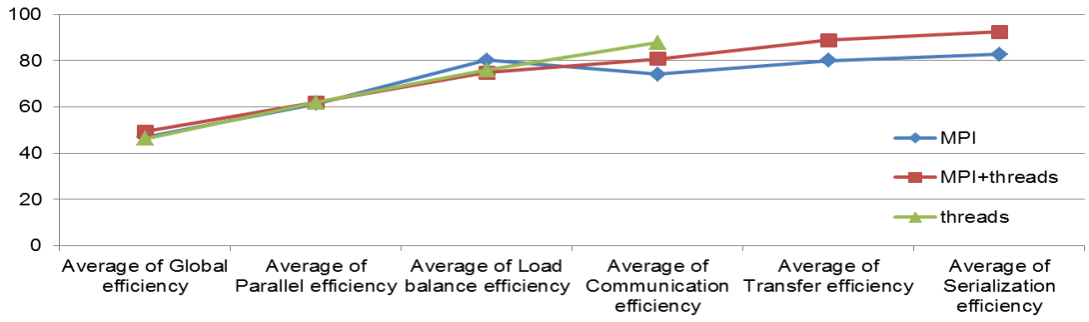

Figure 30: Average efficiencies vs. parallel programing model

We can see a very similar trend for all the paradigms where the most noticeable difference is pure MPI has on average a better load balance and worse communication factors, while the threaded versions report more problems of imbalance.

In Figure 31 we combine the most frequent programing languages with the efficiency factors correlated with the computations as well as the global efficiency and the parallel efficiency.

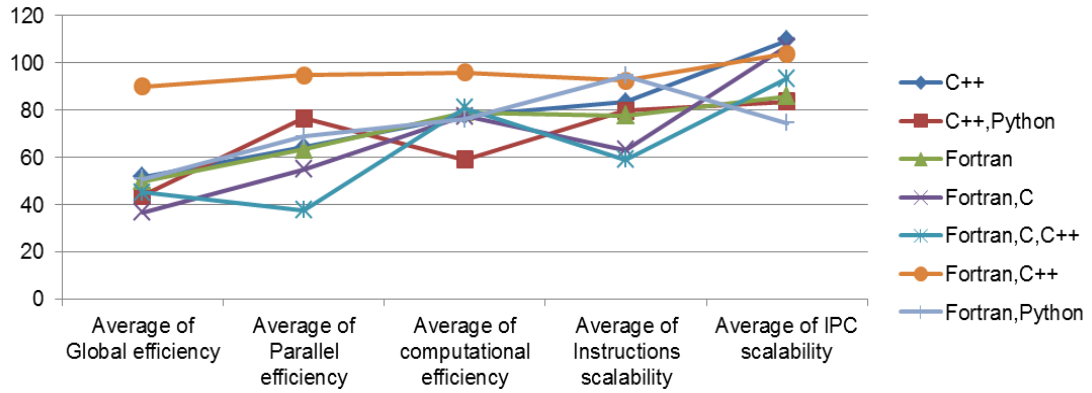

Figure 31: Average efficiencies vs. programing language

In this case we observe much more variability in the data that may indicate a potential correlation. But we also observe the scoring changes for almost each of the efficiencies. The only combination that seems to achieve the highest efficiencies are codes programed in “Fortran,C++”. If we focus on the instructions scalability, “Fortran,Python” gets the best score but also the lower score for IPC scalability. Surprisingly, C++ codes achieve the best score forIPC scalability maybe because they are newer codes (last report it was Fortran codes). The impact of Python does not seem to be relevant as it has very different rations when it is combined with C++ or Fortran.

To complete this correlation in Figure 32 we evaluate the IPC on the most frequent mixtures of programming languages.

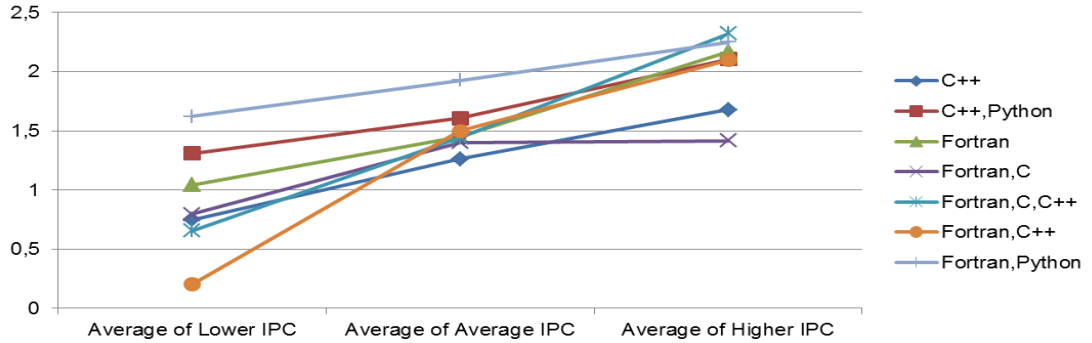

Figure 32: IPC vs. programing language

In this plot we also see significant variability between the different programing languages. As already identified in the previous figure, the higher IPCs correspond to codes in C++. We also see there is larger variability in lower IPC values.

Although we cannot consider there to be a real correlation between the parallel paradigm, the programing language and the achieved efficiencies, we have identified some significant variability, mainly correlated with the programing language.

Based on the findings and recommendations from the Performance Audit, some of our customers requested follow-on work. This has resulted in 24 Performance Plans, 22 Proofs of Concept and 10 new audits for a different code (or a different version of the same code) from the same user. The sum is 56 studies. As a percentage over the 123 completed services, 45% of the studies maintained collaboration with POP CoE. If we consider that only 75% of the codes audited required an additional POP service (percentage of the code with a limited parallel efficiency), the ratio rises up to 61%.

4. Performance Plans

The Performance Plan is a secondary service performed after the initial Audit Service. The Performance Plan aims to identify the root cause of issues previously identified in the Performance Audit as well as to quantify and qualify potential approaches to address these issues.

In the last period of the POP project we started 25 Performance Plans and one of them has been cancelled (second year report there were 21 Performance Plans). At time of writing, we have completed 17 of them (4 since last report).

In this section we summarize the goals of the Performance Plans as well as their results. Due to the small number of new and completed Performance Plans, the summary is very similar to the report in D4.3.

In many of the Performance Plans, the user requested multiple targets. The most frequent goal of the POP Performance Plans has been a deeper analysis of load balance issues identified in the Performance Audit. As we have seen, codes suffering from load imbalance usually have a very poor parallel efficiency. Up to now, 37% of the Performance Plans targeted to provide more details about the load balance of the application (it was 43% in D4.3). To evaluate the impact of the input on the code performance/efficiency and to analyse a different version of the code has been requested each one in 21% of the studies.

Table 1 summarizes the target of the Performance Plans as well as the main recommendation and findings when they have been covered by one of the studies completed.

Target	Findings and Recommendations
Load balance	• Caused by bad domain decomposition (based on #elements, not in the amount of work) • Caused by heterogeneous platform • Caused by unbalance in the amount of work and a delay to get new work from the master • Caused by variations in the IPC • Optimize some subroutines identified • Eliminate zero-sized messages (communications imbalanced) • Finer uniform grid instead coarser random grid • Revised version of the routine gave better balance • Data distribution in the sparse input matrix, changes in the OpenMP parallelisation • PoC to implement optimization suggestions • May be improved using a load balancer (f.i as provided by Charm++ runtime)

Impact of input set	• Impact on parallel efficiency (5-15%). The more balanced cases have lower transfer efficiencies and lower parallel efficiency (may indicate network contention). • Large impact based on the input data. Bigger penalty for instructions scalability issues and some inputs impact on the transfer efficiency • Huge impact but all of them with similar problems
Compare with another version of the code	• New version reduces communications but increases imbalance • Reduce extra computation copying data • Unexpected reductions on data transfer efficiency • PoC to implement optimization suggestions
Evaluate compiler optimizations	• Small improvement adding compiler flags measured
MPI+OpenMP evaluation	• Eliminate code replication • Use larger number of threads to reduce impact of communications limited scalability • PoC to eliminate IPC reduction when filling the socket • PoC to use a task-based approach to overlap communication/computation • Decrease of load balance due to the increase of idle time in the threads • Large OpenMP overheads when scaling • Study to remove critical section
communication efficiency / internode communication	• Change order of point to point communications to reduce contention • Pack large number of collectives (MPI_Bcast, MPI_Allreduce) • High sensitivity to network bandwidth
Vectorization / serial performance	• Only 0.9% of the time is spent in the 10 vectorised loops • PoC to parallelise/vectorise 3 most consuming loops identified
Unexpected variability on #instructions	• Identified non-deterministic behaviour of the application

Table 1: Summary of the assessments recommendations

5. WP4 performance: throughput and timings

As this is the final WP4 deliverable we consider it to be important to evaluate the performance achieved by the assessments work package. The work done is a service offered to external users, so we have no real control on the timings that in many cases are affected by user delays. A second factor that may affect our performance is the number of new services provided by WP3 or as WP4 follow ups along the time (as we have reported, the total number of services is significantly higher than planned) as well as the number of requests cancelled. But even then, the analysis of the codes may take more or less time depending on many specificities of each case (complexity of the code, problems collecting the performance data, need to investigate further a given symptom…). To analyse our own performance, we measured the throughput of studies we have been able to achieve and compared them with the plan, and we had a look at the timings for the different phases we defined in the analysis.

To measure the WP4 throughput we analysed the number of completed studies per month. If we look at the time distribution, there is a huge variability that typically is compensated between consecutive months. For that reason, we consider it is more relevant to look at some statistics like the average or the mode to characterize its value. We have computed these metrics for the whole project, for the last two years to eliminate a potential effect by the first months of the CoE and for the last 18 months where we targeted 6 studies per month. Table 2 includes the statistics analysed.

	Full project	Last two years	Last 18 months
Average	4.10	4.88	5.17
Mode	4.00	4.00	4.00
Median	4.00	4.00	4.00
Planned	5.00	5.50	6.00
Deviation (avg vs. plan)	18%	11%	14%

Table 2: WP4 throughput

Despite we have been able to increase the average, all the other indicators report a throughput of 4 studies completed per month. We computed the deviation with respect to the plan using the average value. We can see that if we eliminate the initial months the deviation is smaller, despite it increases a little bit when we increase the target to 6 studies per month.

To try to identify potential reasons for the deviation, we analysed the number of new requests available per month. As in the previous metric, if we look at the time distribution, there is a large variability compensated between consecutive months. Again, we look at some statistics like the average or the mode to characterize its value. We have computed this metric for the same periods than the previous metric. The statistics are reported in Table 3.

	Full project	Last two years	Last 18 months
Average	5.53	5.78	5.94
Mode	4.00	4.00	5.00
Median	5.50	6.00	6.00
Planned	5.00	5.50	6.00

Table 3: WP4 new services

We can see there is a significant variability depending on the statistic we use. Based on the median, the number of new services is equal or bigger than the target. If we look at the average, the number of new services available has been bigger than the requirements (or very close if we consider only the last 18 months). Only the mode shows lower values, indicating that may have been some months with a lower number of new services, but as we have seen in Figure 2 the total number of services has been bigger than planned since January 2017.

From our experience, the main limitation has been caused by the reported delays during the assessments in most of the cases caused by the users. These delays affected all the phases of the studies but they were especially frequent at the beginning of the study where we need the users to define the input case and to provide us with either the binary or source code and inputs or the collected traces. To measure the duration of the different phases of the WP4 services, we used the information stored in the TRAC tickets. Unfortunately, in many cases the tickets were not updated frequently enough and, in several cases some of the phases were not reported in the tickets at all. Also, some of us considered the phases in a sequence were you never go back to a previous state, while others move back and forward between states (for instance back to waiting user in the middle of the analysis or back to analysing while writing the report).

Considering that the available data has some already known error, we measured the average duration of the different phases distinguishing between the two types of services. The obtained values are reported in Table 4.

Duration (months)	Performance Audit	Performance Plan
Waiting user	1.98	2.90
Tracing	1.83	2.26
Analysing	1.34	1.85
Writing report	1.36	1.37
Reporting to customer	0.69	1.12
Full study	5 months	5.4 months

Table 4: Average phase duration

Maybe the clearest conclusion is that there has been not an important difference between Performance Audits and Performance Plans as we initially expected. This matches with our experiences where plans have been focused on comparing scenarios or machines and where the audits already covered a

quite complete diagnosis of the applications performance. There are phases where the duration may match our expectations considering the delays caused by the user (like the analysis, tracing and reporting phases) while there is a clear deviation in the phase writing report as this does not depend on the user, and the average duration should be two or three weeks.

To complete this section, we analysed the distribution of the duration of the studies. We have included in the same plot the Performance Audits and the Performance Plans as the previous table already confirmed us there have been not major differences between them. We have not done the analysis at the phase level as we detected significant deviations. For that reason, we only analysed the full study.

Figure 33 **Error! Reference source not found.**plots a histogram of the duration in months for the completed services based on the information stored in the TRAC system and identifying the SMEs services. The category of 12 months groups studies that took 12 month or longer. We can see that despite the average duration being between 5 and 6 months, a significant percentage of the services took more than 6 months and only few of them were completed in the initial schedule of one month. If we focus on the SMEs, we can see that most of the SMEs studies last between 2 and 5 months and that they are either quick or take a very long time.

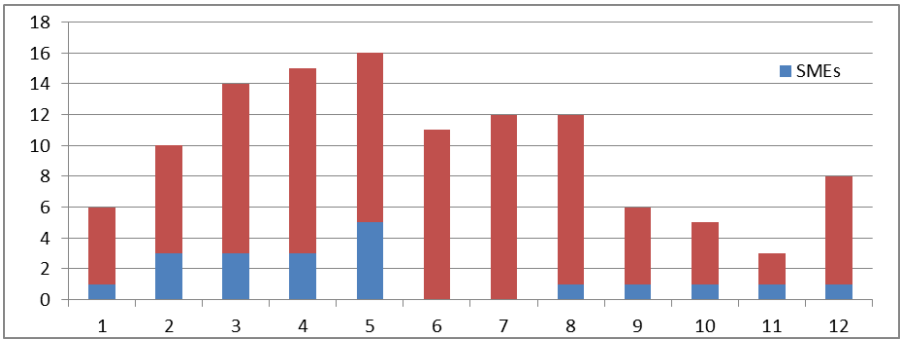

Figure 33: Histogram of the services duration (in months)

6. Recommendations for tools developers

The recommendations for tools developers are very similar to the ones reported in the last deliverable. Some partners have sent a list of detailed suggestions to the tool developers, but as these are very specific, in this deliverable we only give a brief summary.

The POP project uses performance tools extensively and it is a good framework to identify recommendations for tools developers. After 30 months using different performance tools intensively, the POP partners that are not developers of their own tools have been collecting experience about the performance tools developed by partners within the consortium.

Due to their expertise in the area of performance analysis, most of the comments concentrate on the instrumentation phase. The fact that the POP analyst is not familiar with the code emphasises the need to have a robust and complete instrumentation tool that adapts to the specific situation of the code and provides information that the user may not share with us. For instance, one problem difficult to identify is for codes that spawn a process that is not detected by the tracing. As the analyst does not know about this process, it is difficult to detect that there is missing data.

We have also identified multiple POP user cases that expose the tools to new scenarios like Global Arrays or Matlab. Although such codes are uncommon, we should consider if it makes sense to develop support for some of them in future.

Despite those problems, in around half of the studies the data was collected easily and only in less than 10% it was not possible to use the tools provided by the consortium partners. For the rest of the cases, the support of the tools developers facilitated the correct data collection.

The feedback on the visualization tools is more of wish list for new functionalities or changes in the tool behaviour. Sometimes, these are very specific of the user experience and personal preferences and what can be seen as desirable for one user, a different user may not like it.

Acronyms and Abbreviations

- BSD – Berkeley Software Distribution
- CoE – Centre of Excellence
- CUDA - Compute Unified Device Architecture
- HPC – High Performance Computing
- GPL – GNU General Public License
- IPC – Instructions per Cycle
- MPI – Message Passing Interface
- NFS – Network File System
- OmpSs – OpenMP Superscalar
- OpenCL – Open Computing Language
- OpenMP – Open Multi-Processing
- PAPI – Performance Application Programming Interface
- POP – Performance Optimization and Productivity
- PRACE – Partnership for Advanced Computing in Europe
- Pthreads – POSIX Threads
- ROI – Region of interest
- WP – Work Package
- WP4 – Work Package 4
- WPL – Work Package Leader