D4.3 Second-year report on Analysis

Document Information

Contract Number	676553
Project Website	www.pop-coe.eu
Contractual Deadline	M24, September 2017
Dissemination Level	Public (except for the annexes)
Nature	Report
Author	Judit Gimenez (BSC)
Contributor(s)	
Reviewer	Brian Wylie (JSC)
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 reports on 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 during the second year of the project and characterizes the cases analysed during the first two years of the project, summarizing the Performance Plans findings and recommendations provided to the customers. It 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 in the second year. Due to confidentiality issues of some users, those annexes are not public.

1. Introduction

This deliverable describes the work done during the second year of the project and characterizes the Performance Audit and the Performance Plan services carried out by the POP project partners during the first two years of the CoE. 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 6th September 2017 (at the time writing this deliverable), we had 150 requests for WP4 services. This means 87 new requests since D4.2 were written in September 2016. For the total 150 requests that include the work from both years, 116 correspond to Performance Audits and 21 to Performance Plans. The missing 13 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. Comparing with the numbers reported in D4.2, during the second year, we had 63 new Performance Audits and 15 new Performance Plans. As we will discuss in detail below, we consider the progress of the WP4 services is on schedule based on the project plan and we have successfully reached milestone MS3 that targeted 66 assessments at PM24 as we have already 78 completed services.

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. 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 there has been an increase of new requests during the last months that have not yet started.

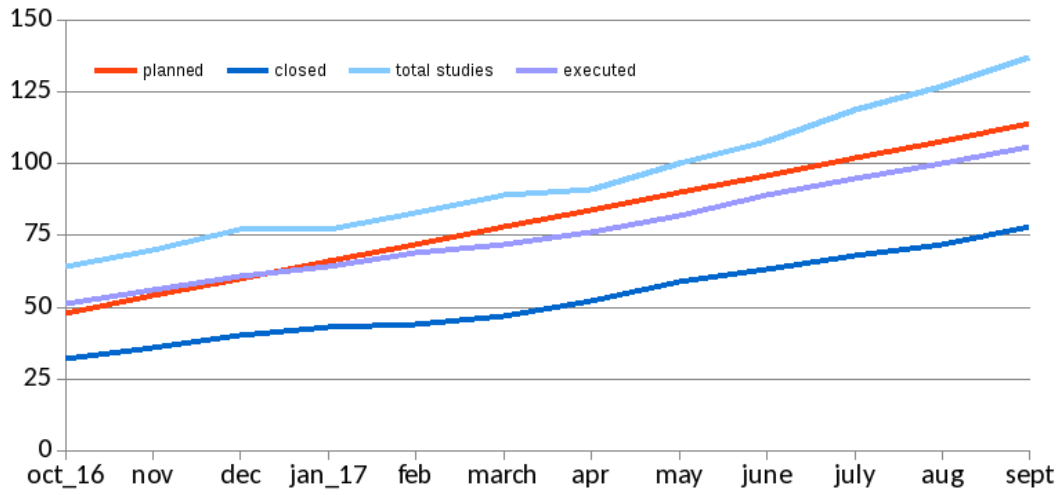

Figure 1: POP assessments evolution w.r.t plan (2nd year)

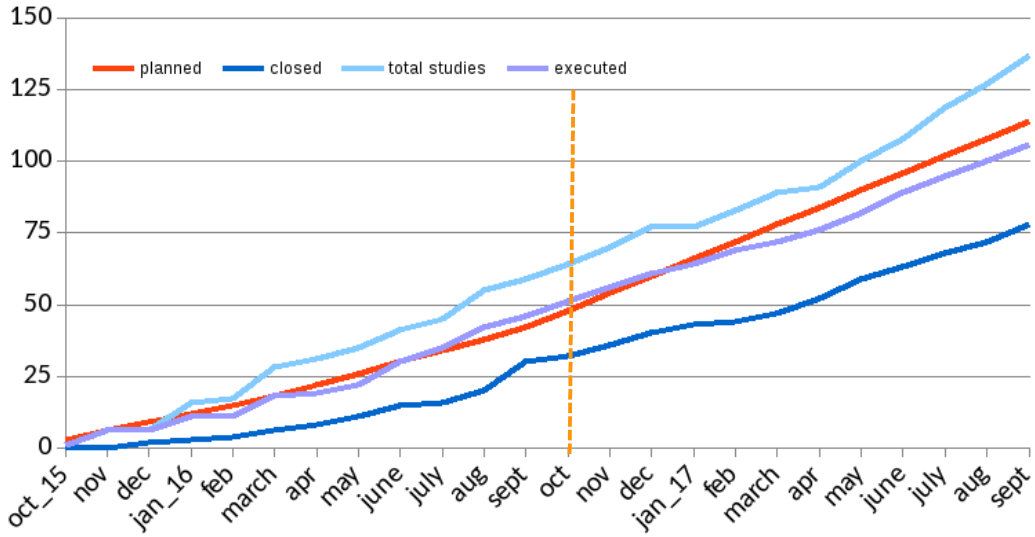

Figure 2: **POP assessments evolution w.r.t plan (both years)**

On the other hand, the number of assessments closed as completed is always significantly lower. We consider the amount of studies planned is very ambitious targeting during this second year 6 studies 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, one of the reasons to have a delay closing the tickets is that most of the studies require more time than initially planned. This is because of delays in most of the cases caused by the service requester. 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 the plot the executed metric is close to the planned value, although since December last year it has been slightly lower than the planned value.

Looking at the plot we can notice that during the last months, the number of executed assessments per month is very similar with a value around 5 and 6 studies per month. This value is close to the planned value for this second year (6 studies per month) and for that reason the distance between planned and executed is almost constant during that period.

At the time of writing this deliverable, the current distribution of the 137 assessments is as follows: 78 completed, 9 being reported to the user, 19 doing measurements of the code and analysing the data and 31 are not yet started requests (either new requests or still waiting some input from the user). The milestone for the end of September was to have 66 assessments completed and this goal was reached between June and July this year.

Figure 3 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 for project month 24 considering the resources assigned in WP4.

We can see that for most of the partners the total number of assessments is higher than he planned value, with the exception of TERATEC. CNRS, a third party from TERATEC started to work on WP4 on 1 October, 2016 and they are still recovering from the initial delay.

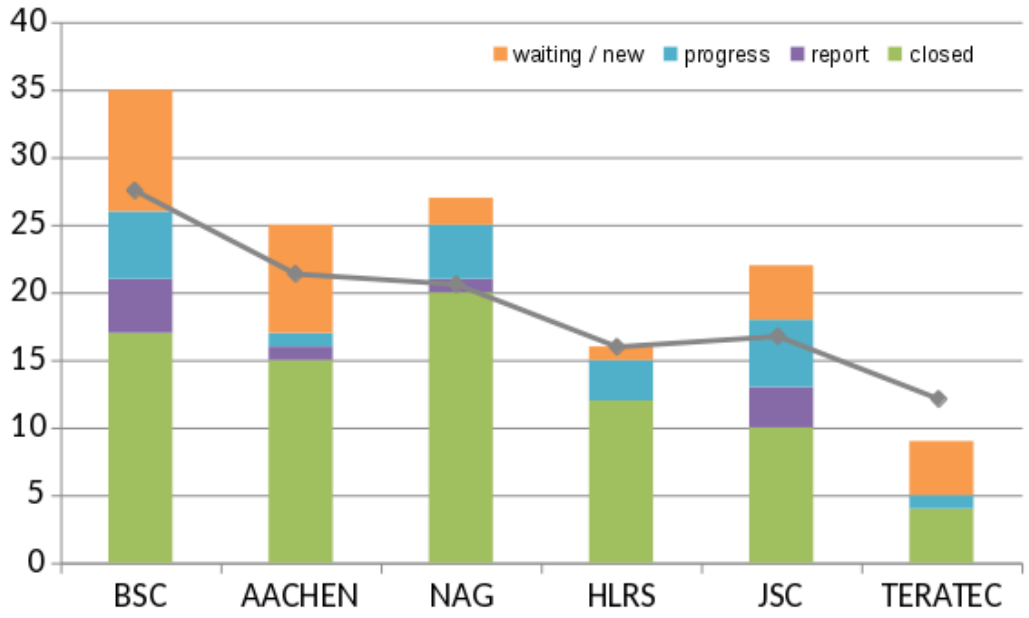

Figure 3: POP assessments per partner (plotted line represents an estimator of the planned value for month 24)

For BSC, AACHEN and HLRS, the planned value is on the orange region that corresponds to the services delayed by the user (or recently assigned). NAG and JSC are closer to their targets and in fact their current total assignment already reached their target values. For most of the partners the total assignment is close to the total value targeted except for HLRS and TERATEC.

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 to feel the code is running inefficiently 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.

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 116 audits, as this information is mostly collected on the request service form, while some of the metrics included the cancelled studies increasing the population to 129.

3.1.1 Performance service

The web form to request an audit offers the alternative to select a focus for the Performance Audit. Figure 4 characterizes the requests received with respect to the main focus selected by the user. We can see that 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).

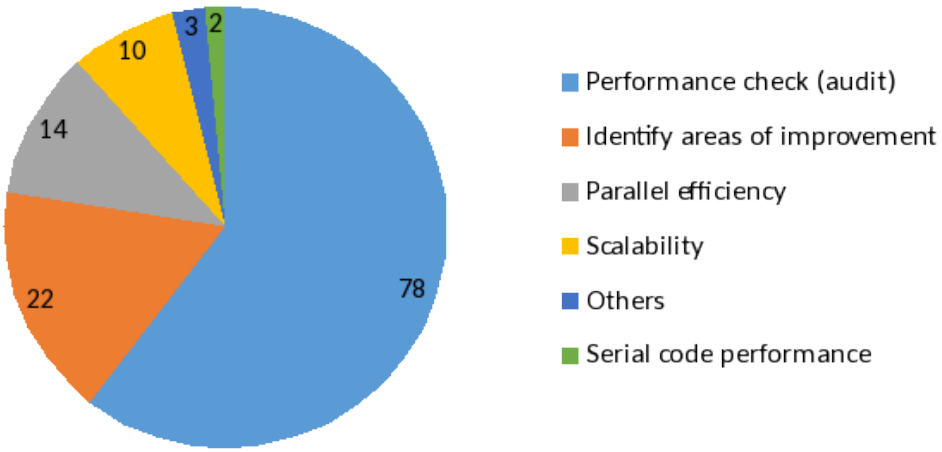

Figure 4: Performance service requested

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 97 users that replied to this question, 54% selected to use their own resources, and half of them also wanted to install the tools and collect the performance data. From the 46% 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 5 plots the answers collected to the question "How did you found out about POP?". We can see that 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 as the person filling the form was different from the one previously contacted; the classification is either other or word of mouth.

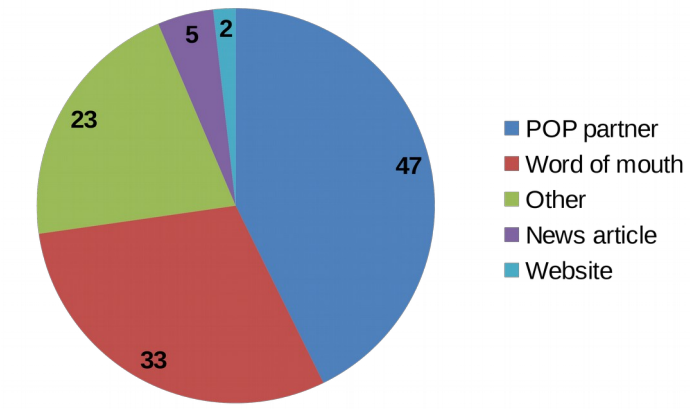

Figure 5: How did you found out about POP?

3.1.2 Code

This section tries to characterize the codes WP4 has been working with. From the 94 answers collected from the users, 64% of the codes were not analysed before the POP Performance Audit. This indicates a significant percentage of codes that benefit from the POP free service to obtain a first analysis of their performance. 87 of users replied to the question about the scaling approach and from them, 72% use strong scaling although that percentage may not be a surprise; it implies there may be more scalability issues with the largest core-counts used. 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 54 of 100 studies are for open source codes.

In Figure 6 we see the distribution of the requests with respect to the scientific/technical area of the code as specified by the user. As in the analysis of the first year, Engineering, Physics, Earth/Atmospheric and Chemistry are the most dominant areas summing up a similar percentage (77% of the

requests). The number of requests from relatively new fields in the HPC sector like health and data analytics is increasing slowly as the project continues.

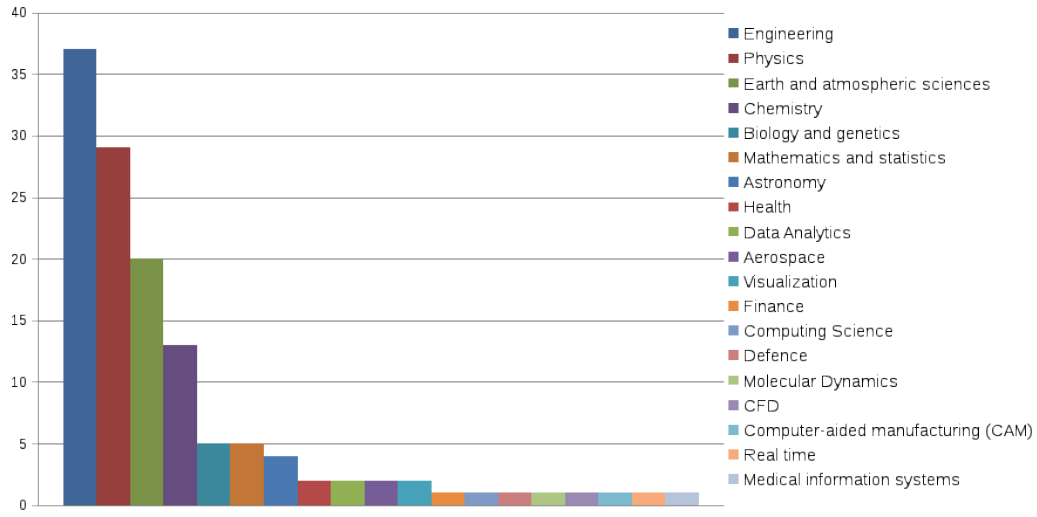

Figure 6: Code scientific/technical area

Figure 7 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 (96%). The accelerators level includes both CUDA and OpenCL with 77% of the cases corresponding to CUDA. Finally, others include programing models like Python multiprocess, MAGMA, Fortran co-arrays, Charm++, TBB or Matlab....

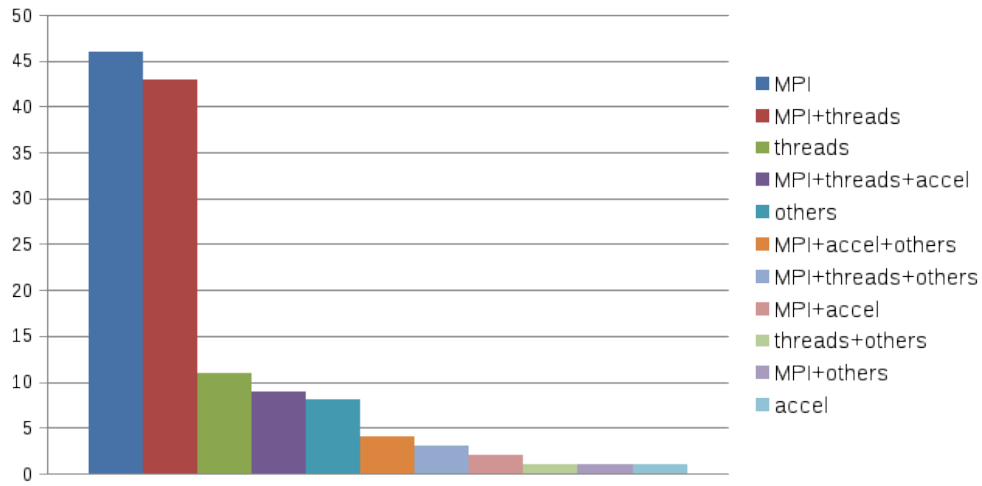

Figure 7: Parallel programming model

As expected, the codes are dominated by MPI followed by MPI+OpenMP but the ratio has been reduced from last year. In total, close to 70% of the requests use MPI or mixed MPI+OpenMP. Although there is a notable percentage of the codes that support CUDA, in many 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 8). Fortran, C++ and mixed C and Fortran represent 73% of the codes (very similar to the first year ratio). Pure Fortran or Fortran mixed with other languages is used in 85 of the 129 codes. As we already identified last year, Python-related requests are higher than initially expected and its ratio has increased a little bit (Python contributes in 17 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.

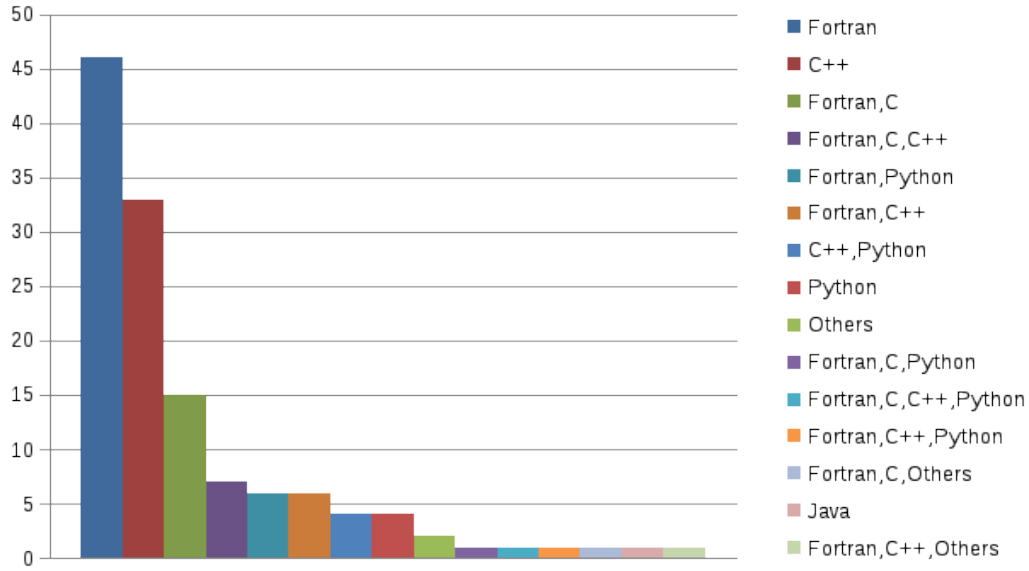

Figure 8: Programming language

Figure 9 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. We can see that Fortran concentrates in the traditional sectors of engineering, physics, chemistry, earth/atmospheric applications and astronomy. Pure C++ codes are a significant percentage in physics and engineering with few representative cases in most of the other sectors. Finally few Python cases appear in most of the sectors (except astronomy and biology). Although the population starts to be representative for some of the sectors, the sample size is too small from some sectors to extract conclusions.

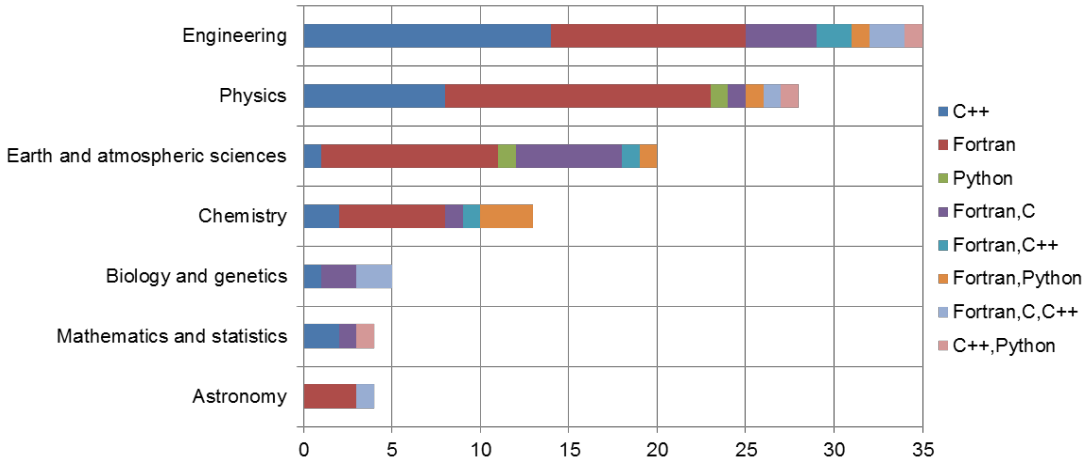

Figure 9 Programing language used on the main code sectors

In a similar way, Figure 10 correlates the parallel programming model to each of the previously mentioned application sectors. We can see that for Engineering, Physics and Earth/atmospheric sciences, the codes' most frequent paradigm is MPI or MPI+thread. Like the statistics from the first year, Chemistry more frequent use is pure MPI codes. Pure thread parallelisation has few occurrences on the most traditional HPC sectors, and there have been no cases in Earth/atmospheric sciences, Biology and Astronomy. Finally, the codes with support to use accelerators are spread on many of the sectors being more relevant in Physics and Chemistry.

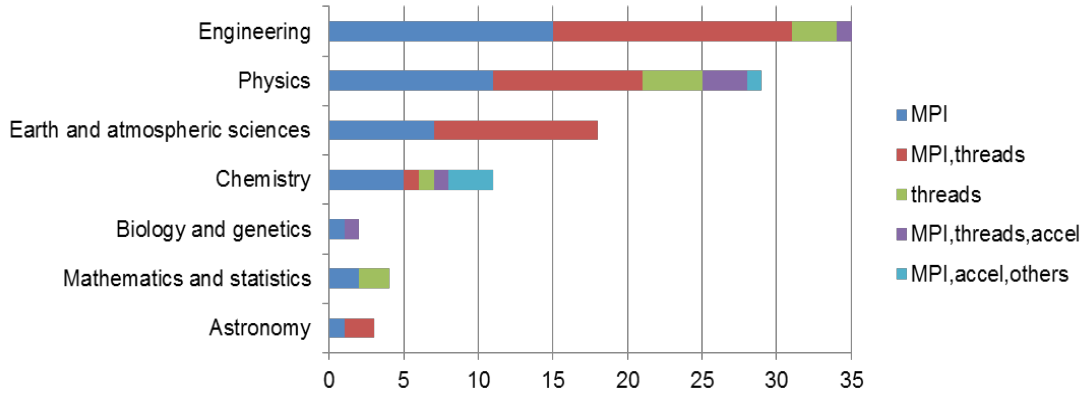

Figure 10 Parallel programing model used on the main code sectors

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 11 correlates the number of cores used 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" 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.

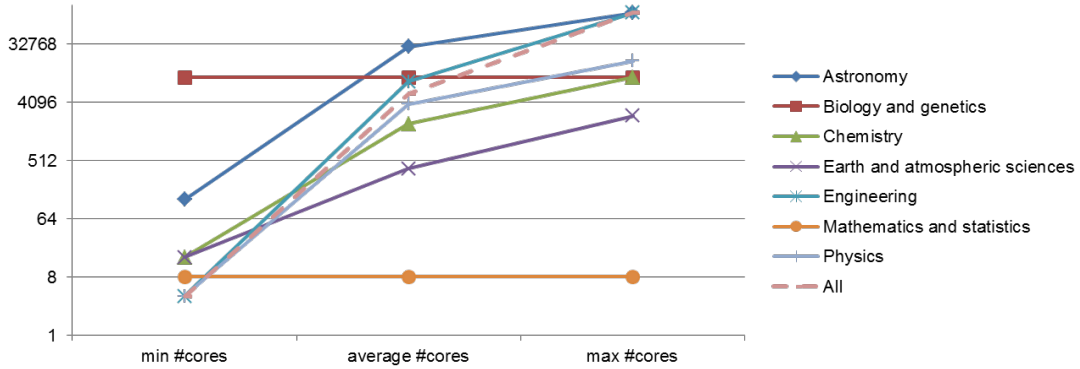

Figure 11 Number of cores used per code sector

In general, most of the sectors show a wide range between the minimum and maximum values reported except for Biology where all the values are 10000 and Mathematics with a constant value of 8 cores. Those sectors have a population of 6 and 5 codes respectively, significantly smaller than the other sectors of the plots. The largest case for most of the sectors range between a few thousand and hundred thousand cores.

Figure 12 correlates the number of cores that have a 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 and including as "All" 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.

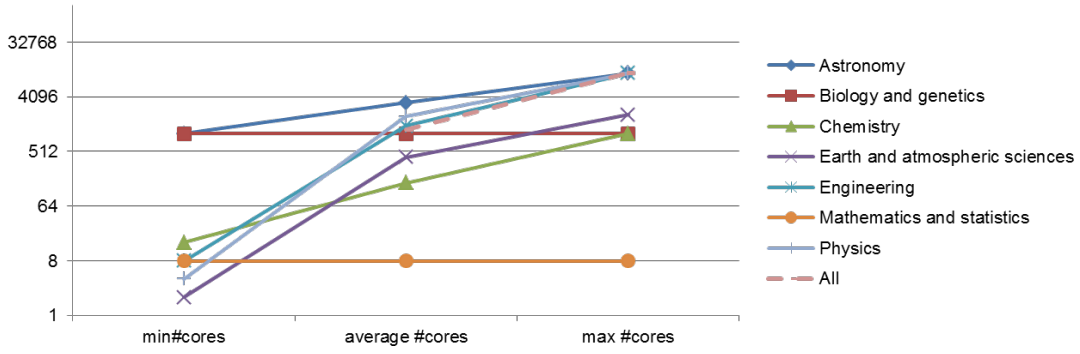

Figure 12 Number of cores with good performance per code sector

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.

3.1.3 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 13. Similar to last year, the academic and research profile dominate the requests

served by the CoE. Industrial users represent 26% of the audits (last year it was 23%), 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 some of the cancelled studies were from companies that despite their interest were not able to allocate personnel for a long period of time.

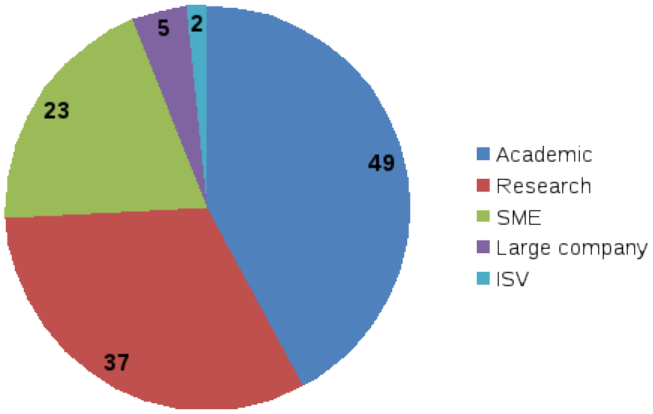

Figure 13 User profile

Figure 14 characterises users based on their role with respect to the code. 72% of the requests are from core developers of the code; guarantee a potential higher impact because it is in their own hands to follow the recommendations provided in the audit. 13% 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.

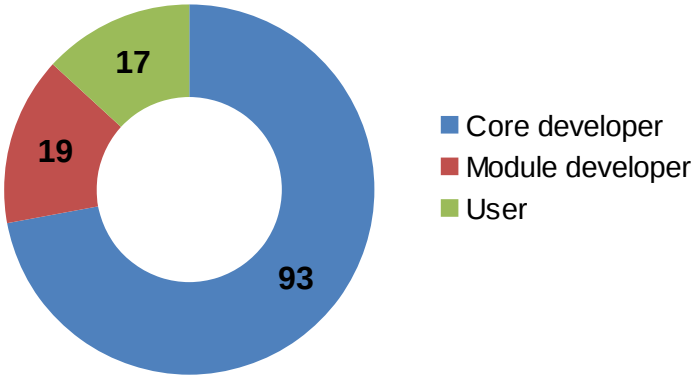

Figure 14: Contribution to the code

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 15 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. The limited number of studies in some sectors suggests we cannot extract conclusions.

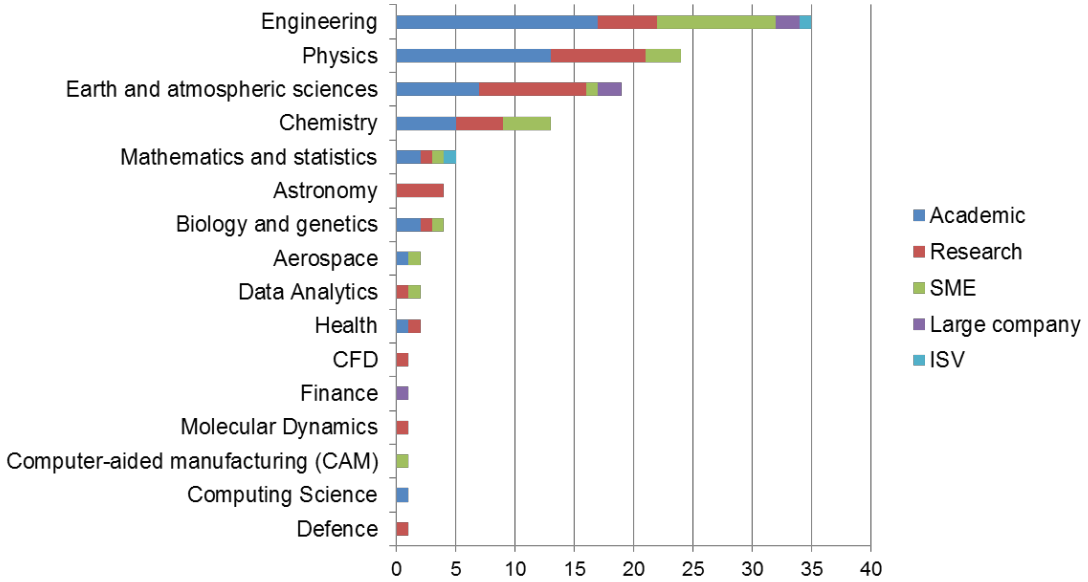

Figure 15: Institution profile per code area

Figure 16 reports the answers collected to the question about their previous knowledge about performance tools classified by the user profile. 70% of the users did not have previous knowledge about performance tools (this ratio is close to the 64% of the codes that were not analysed before). By user profile, we can see that research centres and ISVs have a higher percentage of knowledge and the lowest ratio is in the SMEs.

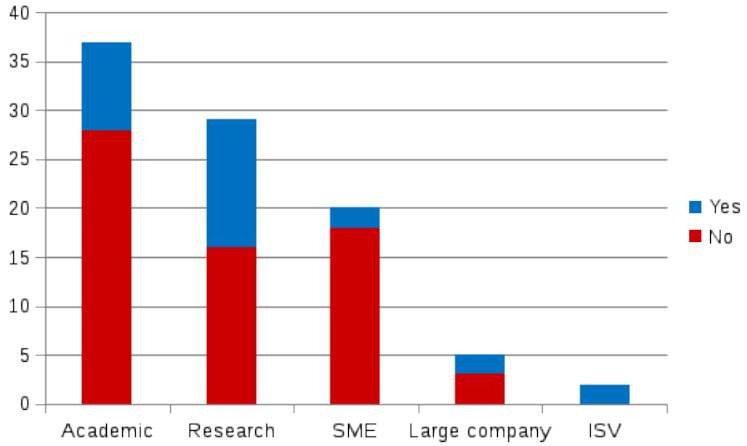

Figure 16: Previous knowledge about performance tools per user profile

Figure 17 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), close to 23% of the requests are from other European countries or countries associated to the EU H2020 programme (same ratio

than reported in D4.2). The highest percentage corresponds to United Kingdom with 35% of the audit requests.

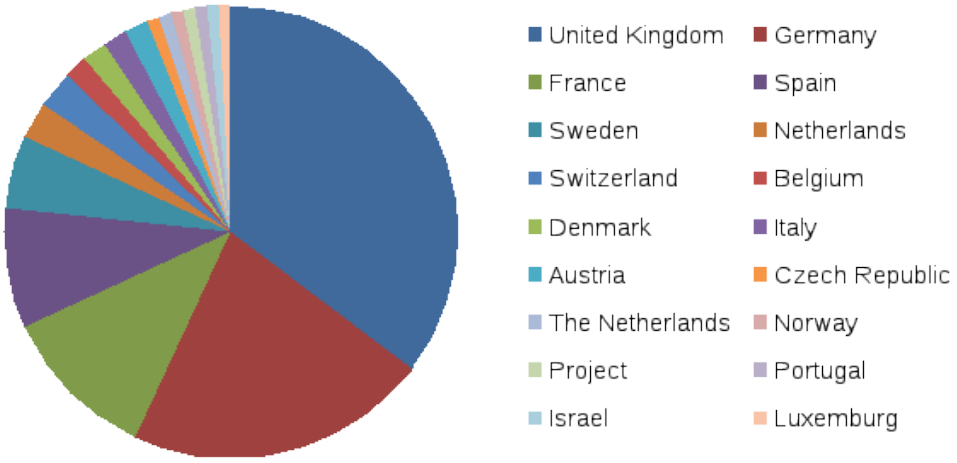

Figure 17: Institution's country

Finally, most of the requests (90%) are from institutions external to the POP consortium. The 10 internal requests correspond to studies carried out during the first year. All of them were from different departments of the POP partner, 6 of them correspond to RWTH Aachen, 2 to HLRS, 1 to JSC and 1 to BSC.

3.2 Performance Audit results analysis

This section summarizes the results from the completed Performance Audits. When writing this deliverable, there are 78 completed audits, however, some of the data is not available for all of them.

Figure 18 characterize 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.

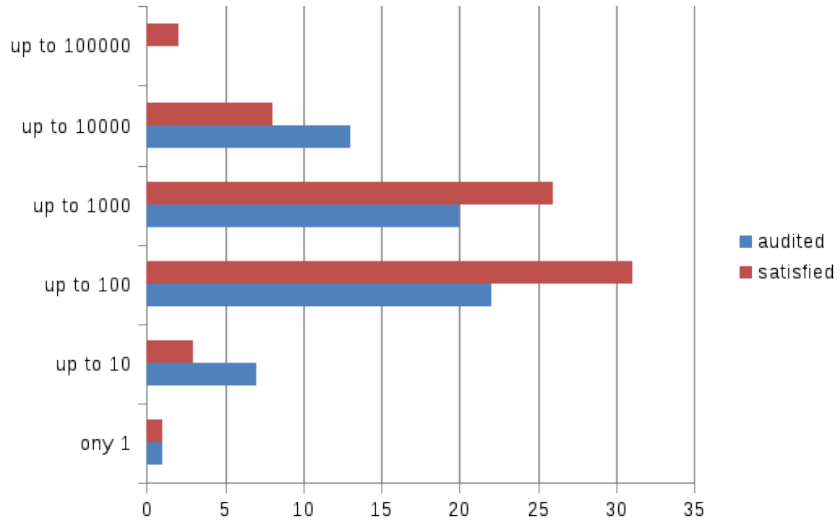

Figure 18: Code and audits scalability (#cores)

The data collected included 63 entries about the largest performing run and 71 for the largest runs audited. We can see that 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.

Figure 19 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. We can see that only for few cases both metrics report the same value. There are cases were 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 user 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.

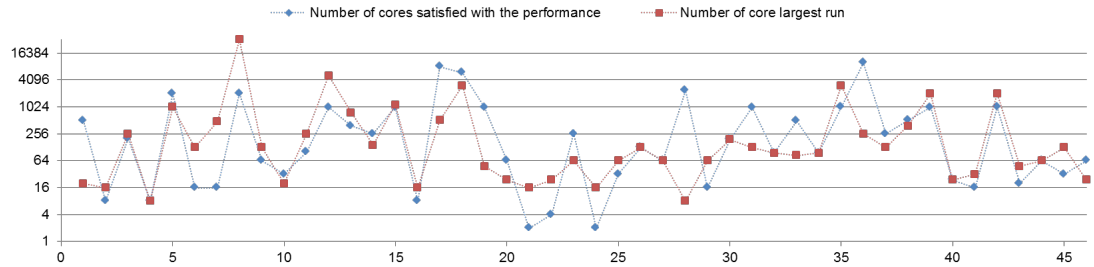

Figure 19: Code and audits scalability comparison

Figure 20 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.

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% because, for instance, when using strong scaling the IPC may increase when more processes are allocated and the computations per process are reduced improving cache usage.

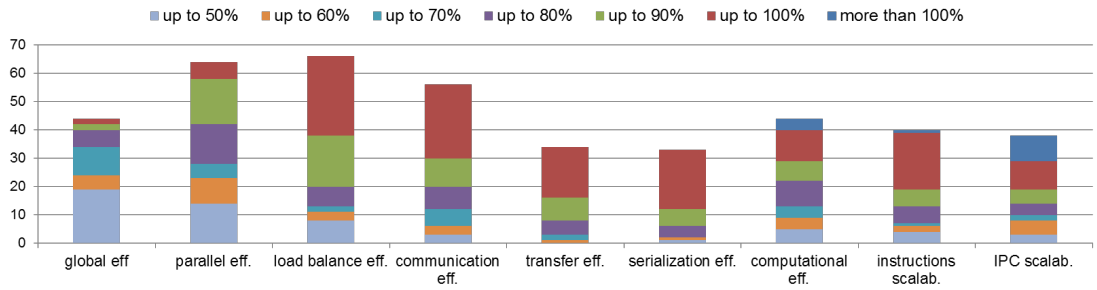

Figure 20: Efficiencies measured

Focusing on parallel efficiency as it is the metric we can compute for almost all the studies, similar to last year, 66% of the codes analysed require improvement to run efficiently in parallel. There are 22% with parallel efficiency below 50% meaning that less than half of the time is dedicated to 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.

The efficiency model is computed as a hierarchy. Next plots (from Figure 21 to Figure 24) correlate each efficiency with its immediate lower level. For instance, in Figure 21 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 19 the dotted lines connecting each series are included to facilitate the readability of the plot.

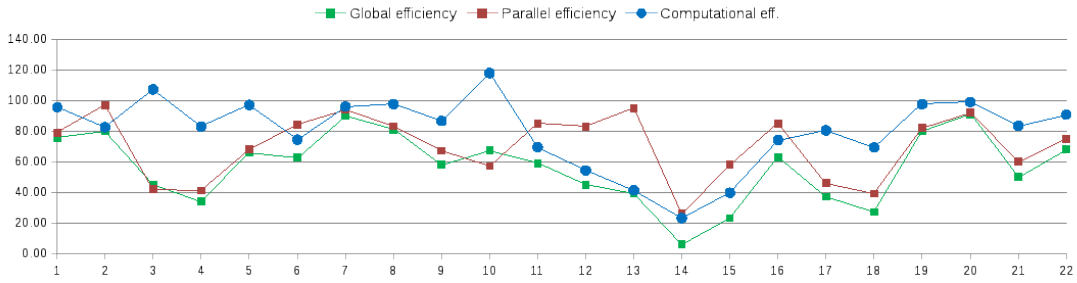

Figure 21: Global efficiency analysis

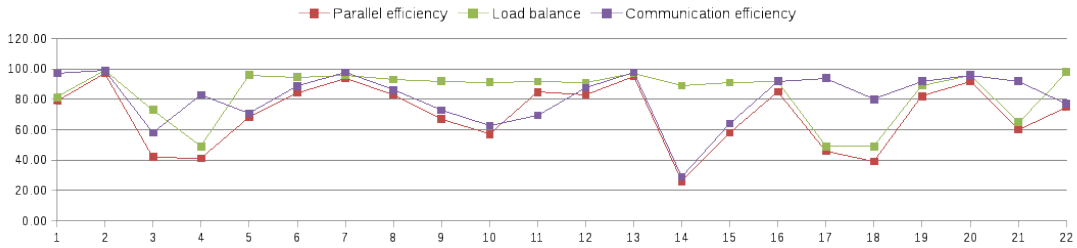

Figure 22: Parallel efficiency analysis

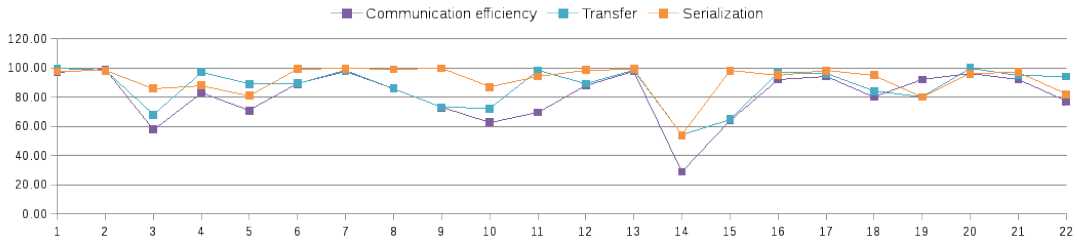

Figure 23: Communication efficiency analysis

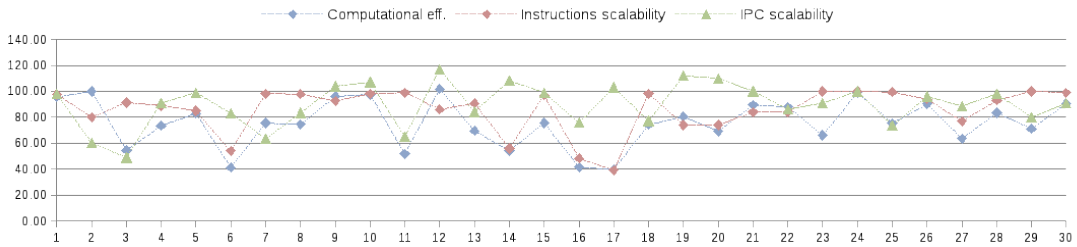

Figure 24: Computational efficiency analysis

Looking at these plots, we can observe:

- Frequently parallel efficiency is lower than computational efficiency.
- 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.
- Load balance is often either very good or very bad.
- Communication efficiency is bigger than 70% in most of cases.
- Loss of communication efficiency is mostly caused by data transfer (high volume of data or high number of communications with respect to the computation).
- In many cases lower computational efficiency is caused by poor instructions scalability. In strong scaling, that reflects code replication.
- In some cases computational efficiency benefits from an increase in the IPC when increasing the scale.

We analysed at very coarse level, the IPC achieved by the applications when this metric is available on the performance data. As boundary we used a value of 1 to consider 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 1.36 and the average higher IPC is 1.94.

We analysed if there was any correlation between the number of cores used and the efficiencies achieved, and as it may be expected there is no correlation and the efficiency is more a characteristic of the code. Finally, we have combined the average efficiencies and IPC achieved with the parallel

paradigm and the programming language used to try to identify if there is some correlation. In Figure 25 we combine the main parallel programing models with the efficiencies factors related to the parallel efficiency as well as the global efficiency. We can see a very similar trend for all the paradigms where the most noticeable difference is that the codes which can use accelerators have lower global efficiency but higher communication related efficiencies. Pure MPI has in average a better load balance and worse communication factors, while the threaded versions have more problems of imbalance.

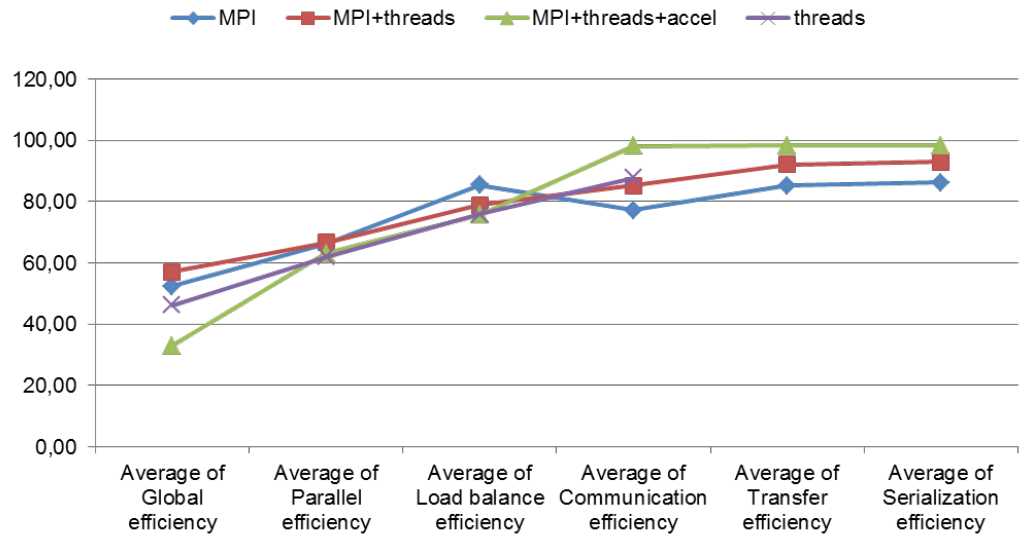

Figure 25: Average efficiencies vs. parallel programing model

In Figure 26 we combine the most frequent programing languages used with the efficiency factors correlated with the computations as well as the global efficiency. In this case we observe more variability in the data with the lower efficiencies for the codes programmed in Fortran and Fortran+C. A potential reason for that is that they correspond to older codes as both groups of codes have problems on instructions scalability indicating potential code replication. On the other hand, programs using C++ either standalone or with Fortran or Fortran and C achieve the highest efficiencies.

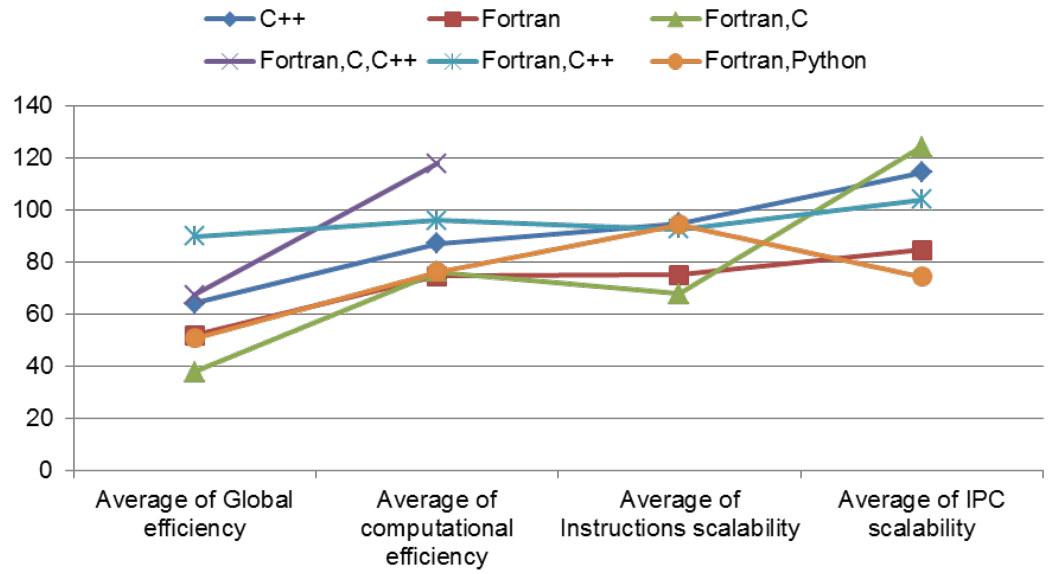

Figure 26: Average efficiencies vs. programing language

To complete this correlation in Figure 27 we combine the IPC with the most frequent mixtures of programming languages. In this plot we also see significant variability between the different programing languages. The higher IPCs correspond to codes in Fortran and Python and in general pure Fortran or Fortran combined with other languages. On the other side, codes that have some part in C++ or C report lower values for IPC. That meets our expectations.

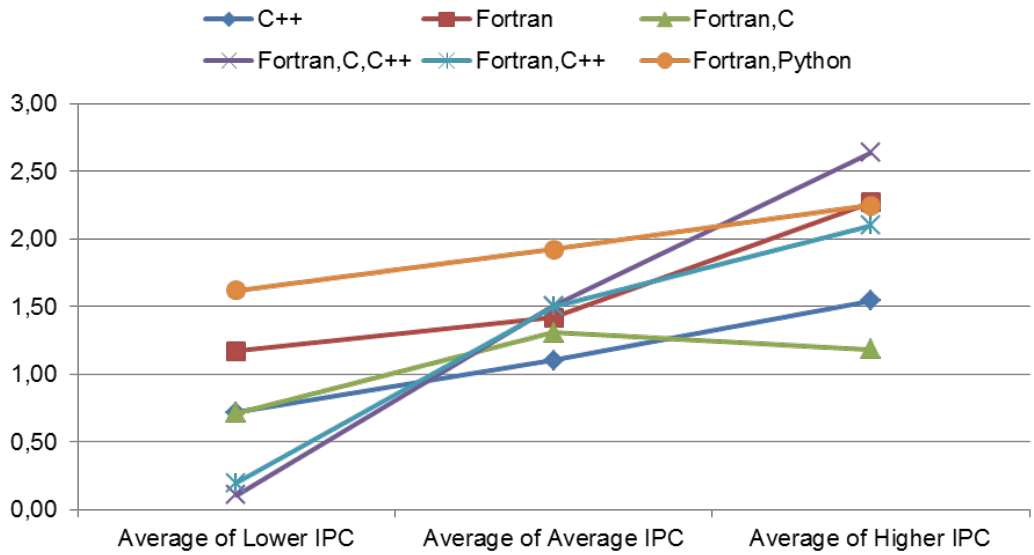

Figure 27: IPC vs. programing language

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 language.

Based on the findings and recommendations from the Performance Audit, some of our customers continued working with us. This has resulted in 21 Performance Plans, 14 Proofs of Concept and 6 new audits for a different code of the same user. The sum is 41 studies for 27 users. As a percentage over the 78 completed services, 35% of the users and 53% of the studies maintained collaboration with the POP CoE. If we consider that only 66% of the codes audited require an additional POP service (percentage of the code with a limited parallel efficiency), the ratios raise up to 53% of the users and 80% of the studies.

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 first two years of the POP project we received 21 requests for Performance Plans (last year report there were 6 Performance Plans). At time of writing, we have completed 13 of them. In this section we summarize the goals of the Performance Plans as well as their results.

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, 43% of the Performance Plans targeted to provide more details about the load balance of the application. To evaluate the impact of the input on the code performance/efficiency and to analyse a different version of the code have been requested in 24% and 14% of the studies respectively.

From the 13 completed Performance Plans, 54% of the reports explicitly suggested to continue the collaboration in the scope of a Proof of Concept (in two cases suggesting different alternatives). That offer generated 5 Proof of concepts, which is a good ratio (71%).

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

Target	Findings and Recommendations
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
communication efficiency / internode communication	• Change order of point to point communications to reduce contention • Pack large number of collectives (MPI_Bcast, MPI_Allreduce)
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
Memory access pattern / memory usage	Work in progress

Table 1: Summary of Audit recommendations

5. Recommendations for tools developers

The POP project uses performance tools extensively and it is a good framework to identify recommendations for tools developers. After two years 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 40% 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.

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