D4.2 First-year report on Analysis
Version 1.0

Document Information

Contract Number	676553
Project Website	www.pop-coe.eu
Contractual Deadline	M12, September 2016
Dissemination Level	Public
Nature	Report
Author	Judit Gimenez (BSC)
Contributor(s)	
Reviewer	Christian Terboven (RWTH-Aachen)
Keywords	performance assessments, first-year report, 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
V0.2	C. Terboven	Minor corrections and additions
V0.3	J. Gimenez	Improvements as suggested by internal review + update statistics and figures
V1.0	J. Gimenez	Final version

Table of Contents

Executive Summary

This deliverable reports on the services provided by the Analysis Work Package (WP4) during the first year 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 and characterizes the cases analysed during the first year of the project, summarizing the findings and recommendations provided to the customers. It also includes the first recommendations for tool developers based on customer feedback. The annexes of the deliverable include a list of the services provided and the reports produced in this period.

1. Introduction

This deliverable describes and characterizes the Performance Audit and the Performance Plan services carried out by the POP project partners during the first year 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 23 September 2016 (at the time of closing this deliverable), we have 63 requests for WP4 services: 53 Performance Audits and 6 Performance Plans. Four requests were cancelled because the user did not reply to our mails for an extended period of time or because he/she moved to a different institution or company. We have 35 services in process or currently being communicated to the customer. As we will discuss in detail below, the progress of the WP4 services is generally on schedule based on the project plan. As annex of this deliverable we include the list of assessments for the first year.

We have implemented the operational procedures described in D4.1 to manage the WP4 services as planned and experience dictates that they are working well. Moreover, we are using TRAC to store and check the status of the requests.

Our overall objective is to provide high quality POP services to all users at all institutions. Part of how we determine the quality of the service provided is based on a comparison of the final audit report results. At the beginning of the project, we used a high level template defined in D4.1 to produce these reports; however, in reviewing the initial audit reports, we found that there was too much variation in the measuring of metrics and the level of detail reported. For this reason, we took a step back and reviewed the audit report template together in order to come up with a more uniform approach which we believe has subsequently improved our overall service.

2. Performance Audits

The Performance Audit is the primary service and may be considered a kind of health check for the codes. 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 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.

We provide two complementary views in this deliverable: first, the evolution of the requests and their status during the first twelve months of the project, and second, a characterization of the codes and users we have been working with as well as the results obtained.

2.1 Evolution and status

The POP DoA targets the completion of 42 assessments in the first year with a planned distribution of 3 new studies for the first 6 months and 4 new studies for the second 6 months.

Figure 1 plots the evolution of the POP assessments during the reporting period. Comparing the studies with respect to the plan, we can see that since January 2016, the total number of studies is significantly higher than the planned value.

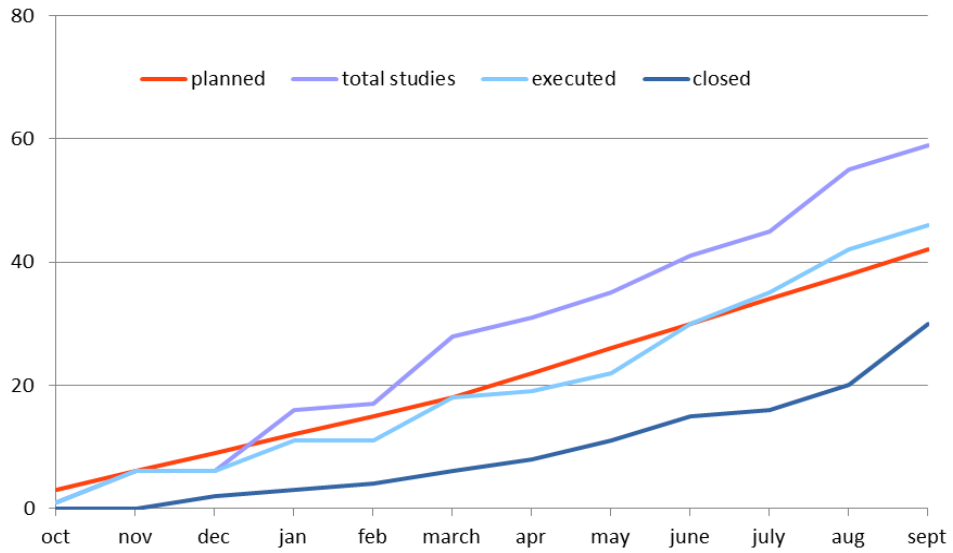

Figure 1: POP assessments evolution w.r.t plan

On the other hand, the number of assessment closed is significantly lower for the full period. The reason 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 user. We have found that these delays are much more probable during the initial phase of the study (due to delays in providing sources or traces, in signing NDAs or in selecting the input cases). We realized this early on in the project and noted it in the deliverable D4.1. For this reason, we think that the executed assessments, including both closed and in progress 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 and even a little bit higher for the last months.

At the time of writing this deliverable, the current distribution of the 59 assessments is as follows: 30 closed, 5 being reported to the user, 11 instrumenting the code and analysing the data and 13 are new requests or still awaiting some input from the user. The milestone for the end of September is to have 42 assessments completed. As of today we consider 35 services either closed or currently being reported to the user. We expect to complete more than half of the 11 remaining on-going services in the coming weeks, which will bring us quite close to our original target.

Figure 2 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 12 considering the resources assigned in WP4.

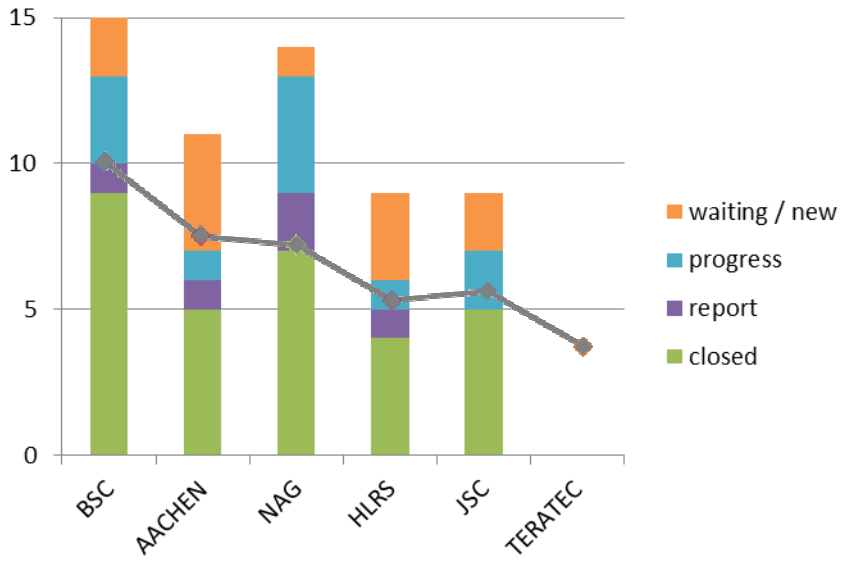

Figure 2: POP assessments per partner (plotted line represents an estimator of the planned value for month 12)

We can see that for most of the partners the sum of closed and reporting cases is close to the planned value, with the exception of TERATEC. CNRS, a third party from TERATEC required a formal third-party agreement between CNRS and TERATEC before they could hire an engineer to perform the assessments. The agreement was signed on 20 May, but was only able to complete a hire on 1 October, 2016.

This delay had minimal impact on the overall project. The rest of the consortium has assumed all the services requested, and there was no delay introduced in any service due to the fact one of the partners was not able to contribute during the first year.

2.2 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 53 audits as this information is mostly collected on the request service form.

2.2.1 Performance service

The web form to request an audit offers the alternative to select a focus for the Performance Audit. Figure 3 characterizes the requests received with respect to the main focus selected by the user. We can see that almost half of the requests selected the basic performance check and close to a 25% asked for us to identify areas of improvement. Scalability is identified as the main concern, followed by parallel efficiency and serial performance. None of the request selected the communications option (that may be considered part of scalability or parallel efficiency).

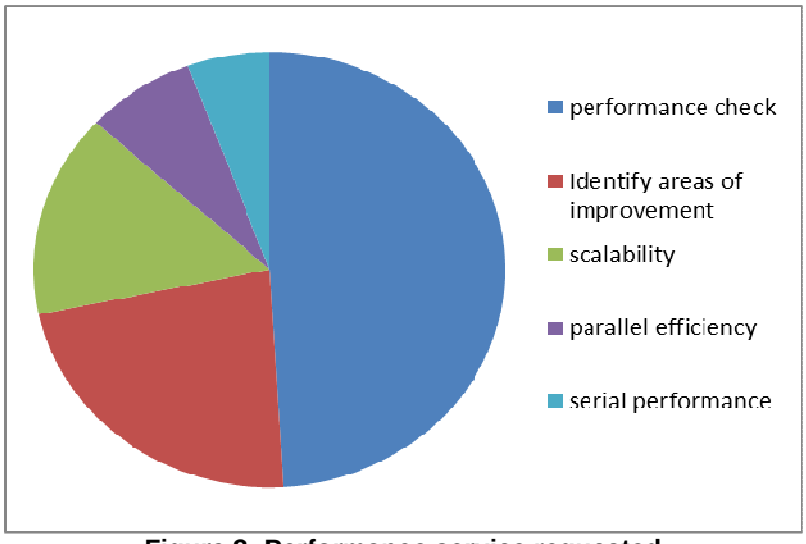

Figure 3: Performance service requested

2.2.2 Code

In Figure 4 we see the distribution of the requests with respect to the scientific/technical area of the code as specified by the user. Engineering, Physics, Earth/Atmospheric and Chemistry are the most dominating areas covering 80% of the requests. There are fewer requests from the relatively new fields in the HPC sector like medical and data analytics.

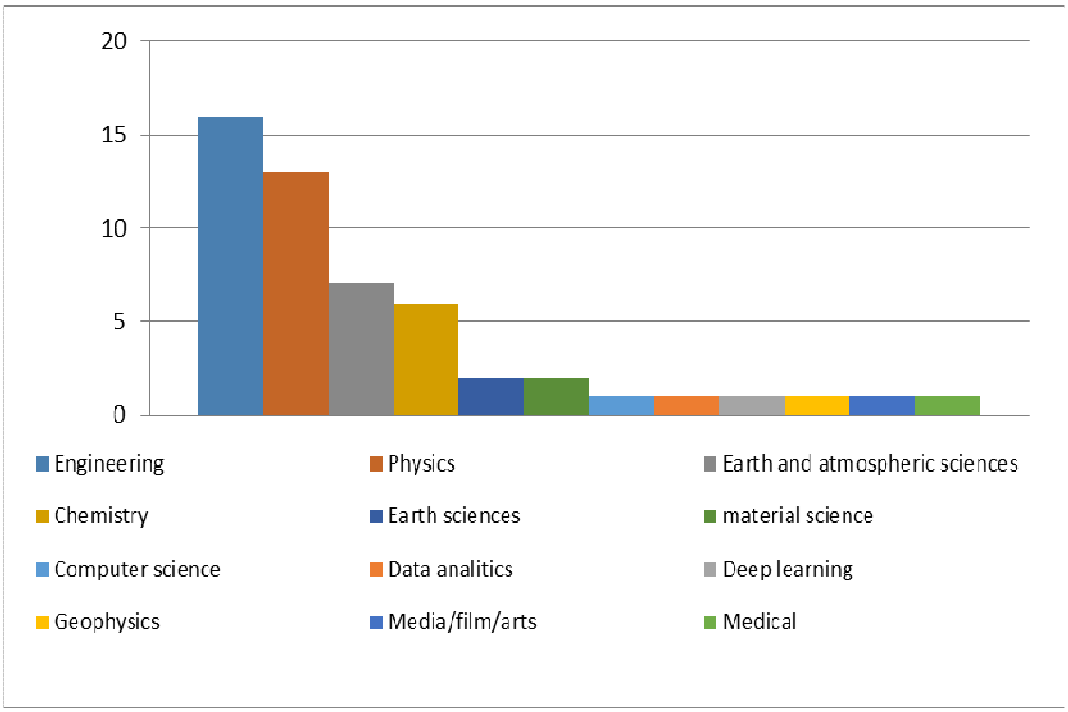

Figure 4: Code scientific/technical area

Figure 5 characterizes the audits with respect to the parallel programing model. The label MPI(+OpenMP) has been used to specify codes that are MPI+OpenMP where we only analysed the pure MPI executions (requested by the user). As expected, the codes are dominated by MPI followed by OpenMP. In total, 49 of the 53 requests use MPI, OpenMP or mixed MPI+OpenMP.

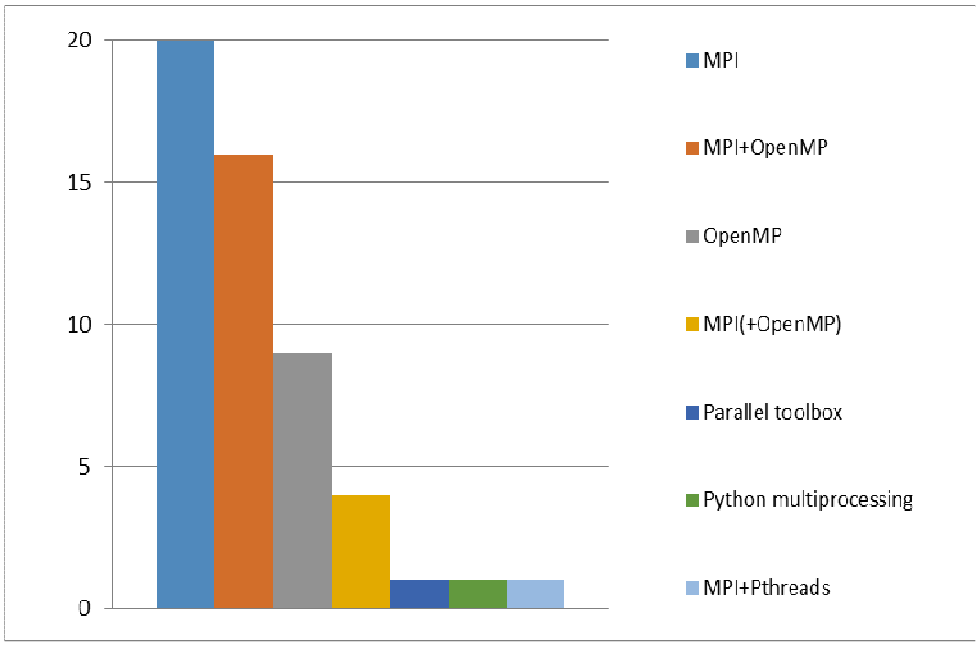

Figure 5: Parallel programming model

There are also no surprises with respect to the most dominating programming languages (see Figure 6). Fortran, C++ and mixed C and Fortran represent 79% of the codes. Pure Fortran or Fortran mixed with other languages is used in 34 of the 53 codes. Only Python-related requests are higher than expected (Python contributes in 5 of the codes).

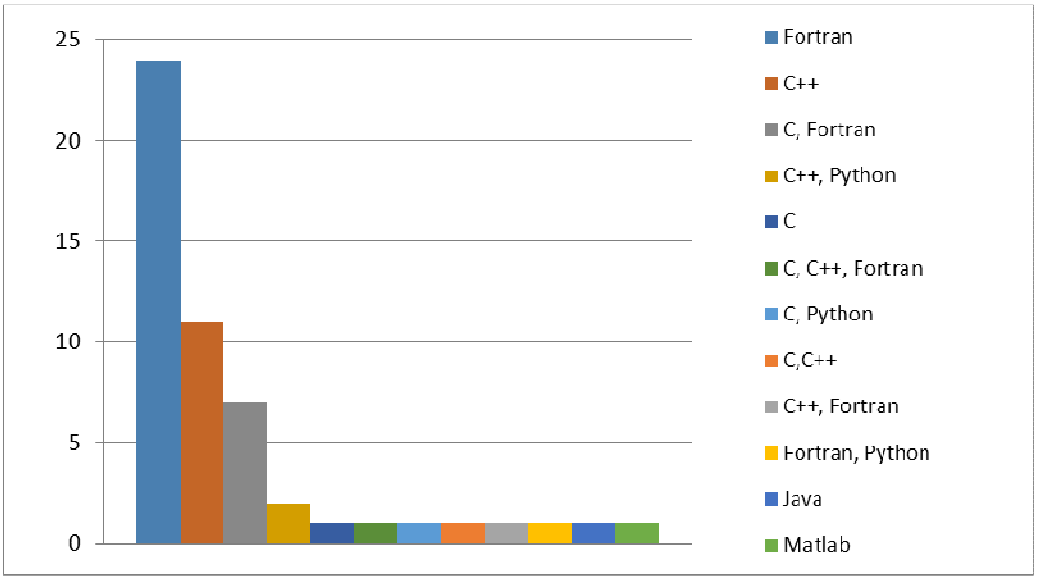

Figure 6: Programming language

Figure 7 correlates the programming language with respect to the code sectors. In this plot we eliminated both programming languages and sectors with just one occurrence.

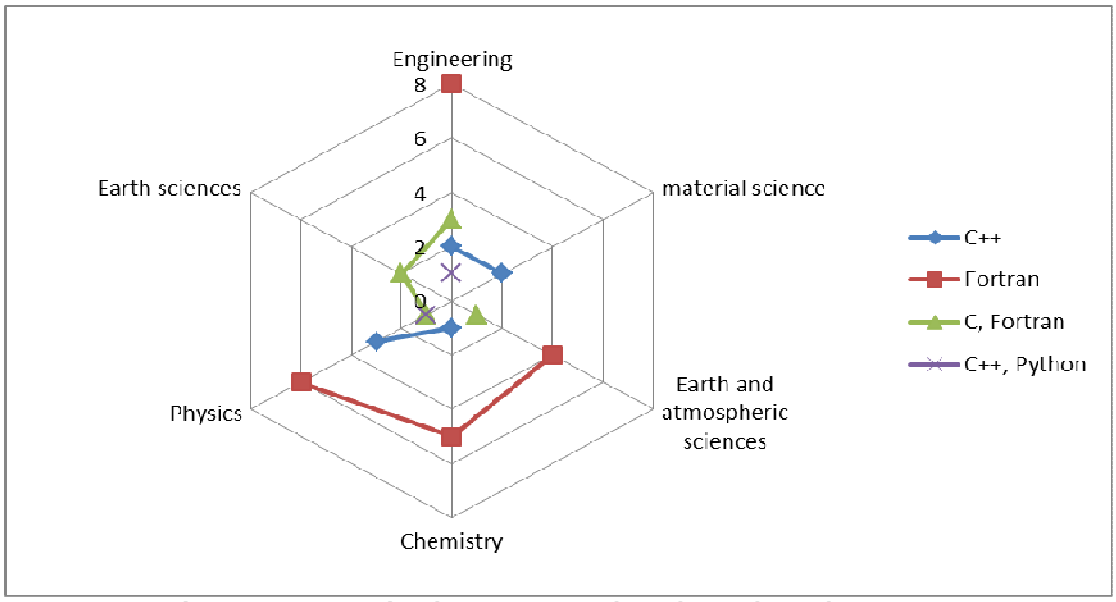

Figure 7 Programing language used on the main code sectors

We can see that Fortran dominates in the traditional sectors of engineering, physics, chemistry and earth/atmospheric applications. Pure C++ codes appear in material science, physics, chemistry and engineering but with only a few representative cases per area. We currently believe that the sample size is too small to be able to extract real observations and that we may need to wait until the end of the second year or even the end of the POP project in order to be able to draw real conclusions.

In a similar way, Figure 8 correlates the parallel programing model to each of the previously mentioned application sectors. We can see that for the engineering codes the most frequent paradigm is MPI+OpenMP while in Chemistry we see more frequent use of pure MPI codes. Pure OpenMP is rarely used when looking across these sectors.

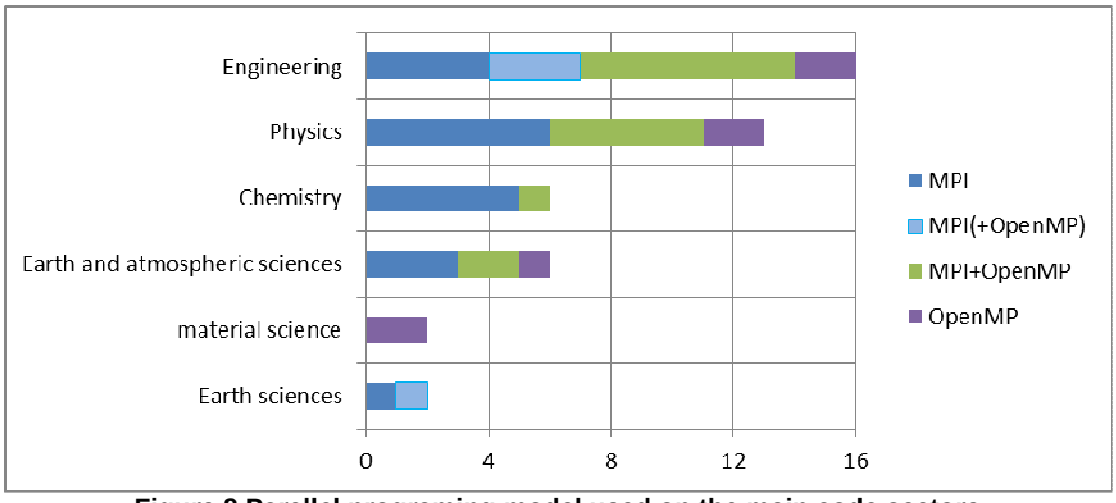

Figure 8 Parallel programing model used on the main code sectors

2.2.3 User

Twelve of the 53 audits have been requested by industries. This represents 23% of the studies. We consider this to be a significant percentage despite the fact that we would like to see even more requests from industry. However, we recognize that it is more difficult to engage industry because of NDAs or other restrictions with respect to disseminating the results. Obtaining best cases from non-industrial institutions will provide us very good material when targeting industries. Industries would not only benefit from the studies they directly request but also from the improvements implemented in open source codes that they use. In addition to the industrial requests, there have been 4 requests from governmental institutions and 37 requests from universities and research centres.

Figure 9 correlates the code area with respect to the user institution. In the plot the data has been expressed as a percentage, not considering the number of studies.

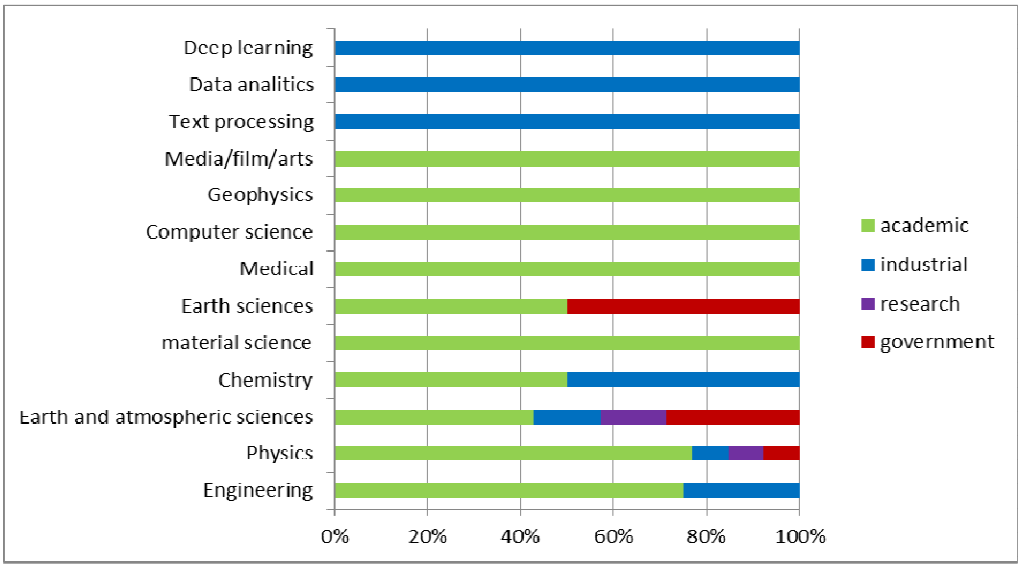

Figure 9: Institution profile per code area

It is a little bit early to extract conclusions because of the low number of studies on some of the areas. But we can see there are code areas dominated by just one of the profiles. For instance, all the requests from data analytics, deep learning and text processing belong to industrial companies while computer science, geophysics, material science, medical and media/film/arts had all the requests from academic institutions. There have also been industrial requests in other areas such as chemistry, earth and atmospheric sciences, physics and engineering.

Figure 10 characterizes the user with respect to the sector of the institution or department requesting the service. It is interesting to compare this plot with Figure 4 where the distribution is made with respect to the code sector. For

instance, Materials is the most dominating sector of the institutions while there were only two requests for codes classified by the users as material science.

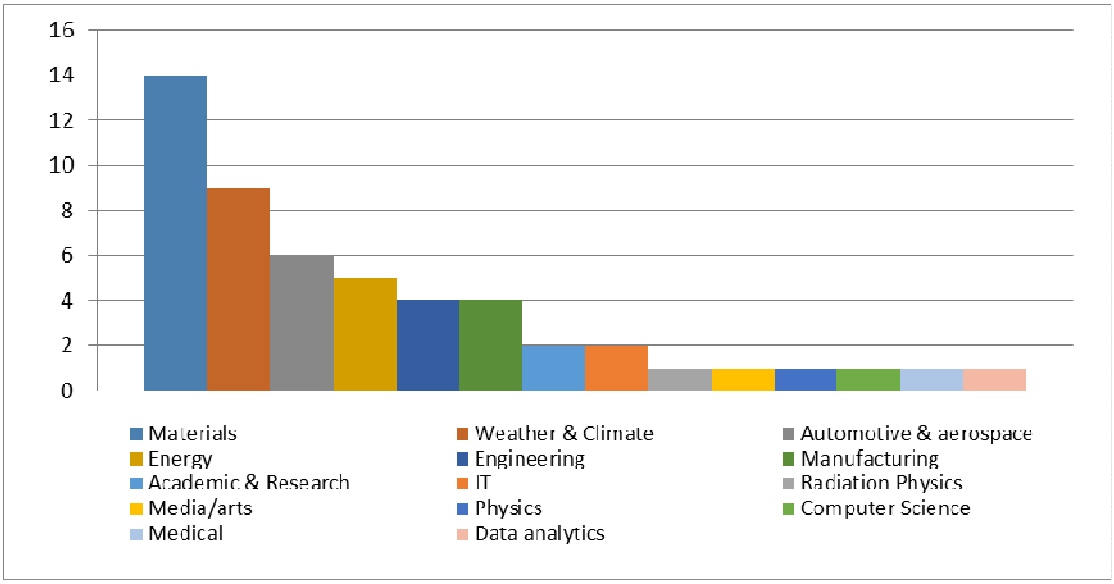

Figure 10: Institution/department scientific/technical area

Figure 11 plots the country of the user's institution. Despite most of the requests are from POP partner's countries (United Kingdom, Germany, Spain and France), close to a 23% of the requests are from other European countries.

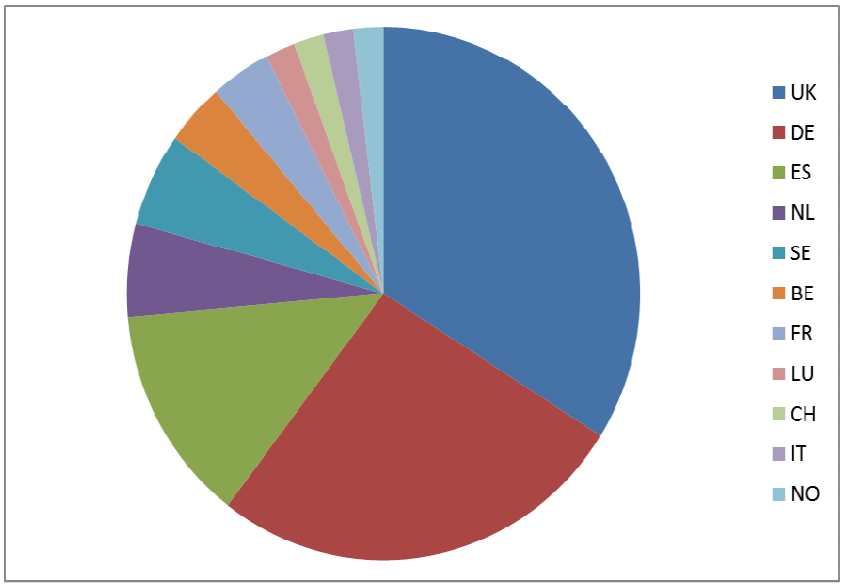

Figure 11: Institution's country

Finally, most of the requests (81%) are from institutions external to the POP consortium. The 10 internal requests (all of them from different departments than that of the POP partner) 6 of them correspond to RWTH Aachen University, 2 to HLRS, 1 to JSC and 1 to BSC.

2.3 Performance Audit results analysis

This section summarizes the results from the 29 closed Performance Audits.

Figure 12 characterize the assessments with respect to the larger run analysed. We can see that in most of the cases we have been looking at runs between few tens and few hundred cores. There are two exceptions of OpenMP codes where the user requested to look at the serial performance, one of them from industry.

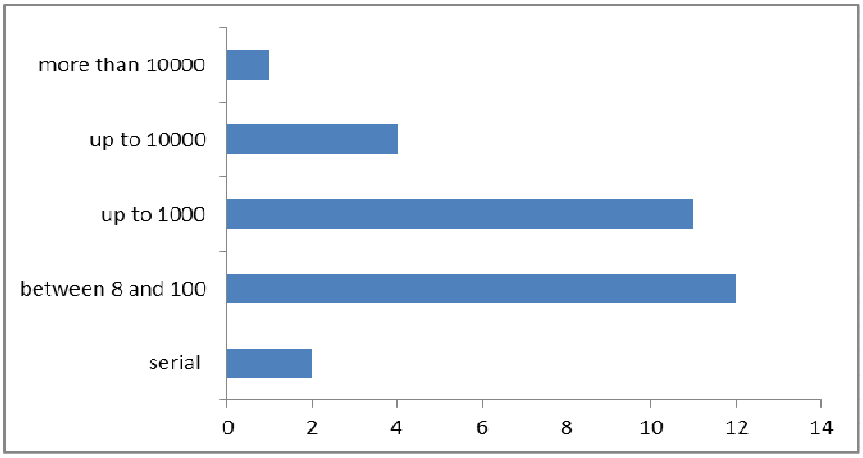

Figure 12: Assessments scalability (#cores)

Figure 13 classifies the parallel efficiency we have observed in the studies. We measure the parallel efficiency considering the time the application is executing in user code over the total time (considering that the time spent in the MPI or OpenMP parallel runtimes is an overhead that the code has to pay to run in parallel).

We have considered as very good and good efficiencies ratios higher than 90% and 80% respectively. In these cases, despite there is some space for improvement, there is not an important need. On the other side we have the serial analysis of the OpenMP codes where we do not have information on its efficiency. The rest of the levels (acceptable and bad) that represent a 67% of the codes analysed do require an improvement to run efficiently in parallel.

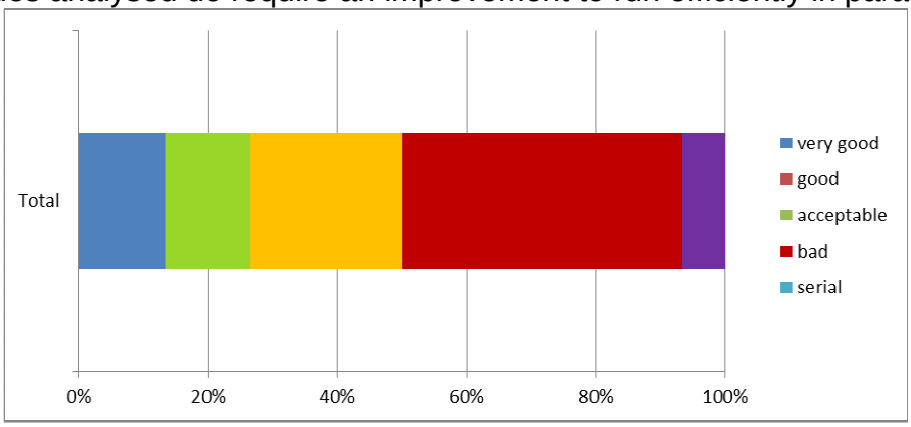

Figure 13: Parallel efficiency

Looking at the reasons for the efficiency loss, in the case of the codes that achieve an acceptable parallel efficiency, the most frequent reason has been problems of data transfer (high volume of data or high number of communications with respect to the computation). If we focus on the codes that had a bad efficiency, most of the codes present problems of load imbalance (the work is not well distributed causing delays on the synchronization points). We had some severe cases where the load balance efficiency metric went down to 50%. In some of the cases of bad performance there was a combination of load balance and data transfer problems.

Figure 14 analyses, at very coarse level, the IPC achieved by the applications when this metric is available on the performance data. This IPC is not weighted with respect to the duration of the different regions, but looks at the overall range of IPC measurements. 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.

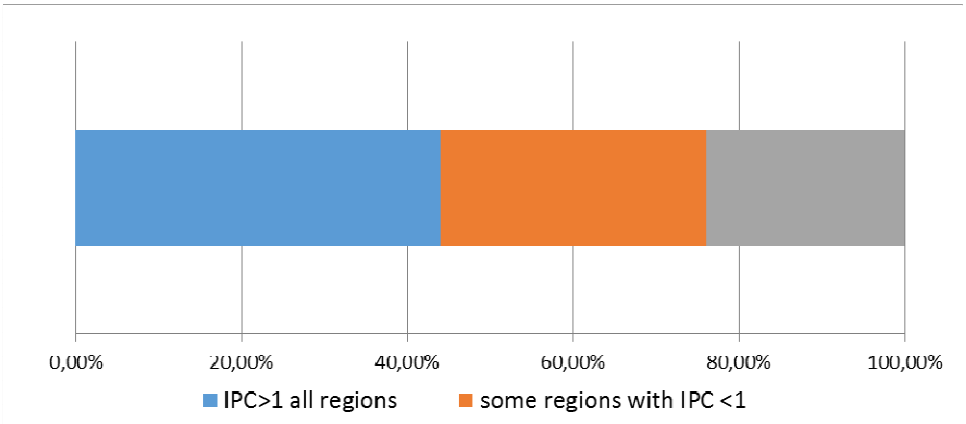

Figure 14: IPC achieved

We can see that 44% of the codes had an IPC bigger than 1 for all the computing regions. A 32% of the codes have some regions with an IPC lower than 1 that may be still acceptable depending on the weight of the bad performing region. Finally, a 24% of the codes reported poor IPC for all the regions denoting a real need to improve the IPC to improve the use of the resources.

In multiple studies, we detected problems of IPC reduction when scaling. Most of them are correlated with the multicore sharing of some cache levels. In other cases, IPC is improved with the scale compensating potential degradations of the parallel efficiency.

Table 1 and Table 2 summarize the findings and recommendations reported to the users of POP audits during the first 12 months. We have classified them with respect to which component (parallel programing paradigm, serial code execution or machine used). When more data is collected, we will be able to

do a deeper analysis including for instance the frequency of the different symptoms.

	Findings
MPI	Bad MPI pattern
	Large MPI times
OpenMP	Poor OpenMP scaling
Serial	Instructions imbalance
	IPC generating imbalance
	IPC reduces / increases with scale
	Code replication
	Function with unexpected variability between runs
	Too many calls for a function
	Identified regions to parallelise
machine	Clock frequency variations
	Degradation filling nodes
	Noisy machine

Table 1: Summary of Audit findings

	Recommendations
MPI	Overlap computations and communications/packing
	Improve MPI patterns
	Advance receive calls
	Add OpenMP to reduce stress in MPI collectives
	Reduce number of collectives
Serial	Optimize serial code
	Improve cache reuse
	Improve memory usage
	Apply vectorization
	Reduce divisions multiplying by inverse
	Improve load balance
	Use new compiler flags, optimized libraries

Table 2: Summary of Audit recommendations

The average time required to perform the assessments (from the start of the work until the study is tagged as closed) is 2 months varying from less than 1 month to 3 months and a half.

Based on the results we obtained in the Performance Audit, we were able to convince some of our customers to continue working with us. This has resulted in 6 Performance Plans and 5 Proof of Concept. As a percentage, close than 19% of the studies resulted in an extension of service (one of the studies was extended to both a Performance Plan and a Proof of Concept). We consider this is a good ratio for the first year as it represents close to one third of the codes analysed that require an improvement. We will try to increase this ratio to 50% of the codes that require an improvement for next year.

3. 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 fin the Performance Audit as well as to quantifies and qualifies potential approaches to address these issues.

In first year of the POP project we received 6 requests for Performance Plans and have started work on all of them. At time of writing, we have completed only one of them.. In this section, we briefly describe our progress for all performance plans, and we provide results (or preliminary results) when applicable.

3.1 Ateles Performance plan (HLRS)

Ateles is a Finite Element code which uses the Discontinuous Galerkin scheme on top of a distributed parallel octree mesh data structure. The main issue of Ateles code, identified during the audit, is load imbalance in the parallelization. The focus of the performance plan is an in depth analysis of these load imbalances in the MPI parallelization of the application.

Figure 15 captures how the imbalances prevent scalability of Ateles on the Hazel Hen system for the given use case. The upper panel shows time lines for 24 MPI processes, where green is application code, red is MPI calls, and black lines show MPI Point-to-point communication. The lower panel shows the fraction of processes in MPI and Application code respectively on the same time line. Most MPI processes wait on rank 0 for the first two phases, and on process 2 for the third phase.

In order to study the load imbalance in the MPI parallelization, we used SCORE-P for creating profiles and traces. By using Cube we analysed profiles for two configurations of Ateles. The following metrics were calculated: the total time, the aggregated total time for all MPI processes, the aggregated computation time and MPI time. We found, that the application running with this input data sets is clearly MPI-communication-bound.

We used Vampir to visualize the trace for the first configuration. Repeated time intervals (segments) were identified in the timeline. Further analysis was focused on subroutines in these segments. Execution and MPI communication times in an execution segment for each important subroutine were calculated. We used Vampir to investigate the communication imbalance. The message size distribution, the communication matrix view with a number of messages, and with an aggregate message volume were visualized.

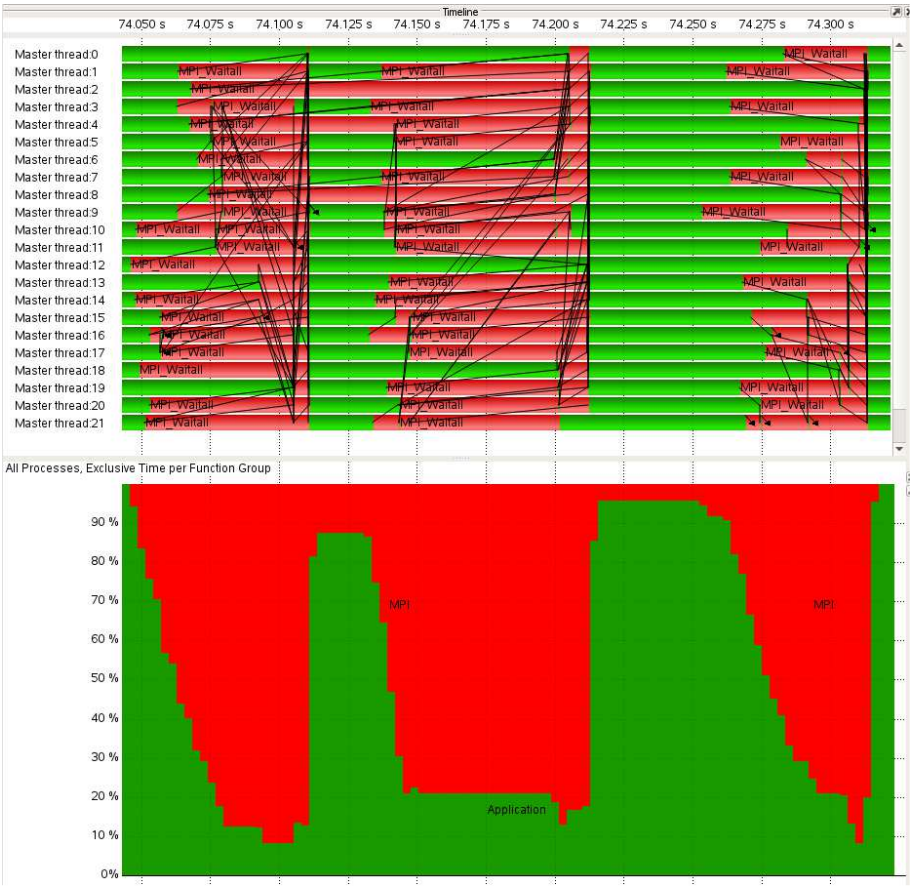

Figure 15: Application code load imbalance and its effect on MPI

The reason of the computation and communication imbalance is the bad domain decomposition. Here it becomes obvious, that the assumption that all elements have the same computational costs is not valid and by this the load distribution by equal number of elements along a space filling curve does not work: different elements trigger different behaviour inside the application.

Performance analysis revealed four basic functions at this point in the code. The optimization of the function showing the worst load imbalance may provide up to 20% and optimizations for all four functions could give in total a 36% improvement.

Improvement of the load balance will most likely come along with improving the MPI collective calls after each time step. While not noticeable for the smaller first configuration, it may provide already additional 6% improvement for the larger second configuration.

Beside the computational load imbalance, the application suffers from communication imbalance. There is a difference in the number of send messages as well as the message volume. Also we see a very high number of zero byte messages. These messages should be prevented as they stress not only the network but also increase the time of the MPI_Waitall calls.

3.2 OpenNN Performance plan (BSC)

The OpenNN audit targeted the platform available at the user site. The user is a Spanish SME and their platform an i7-4790 with 1 socket of 4 cores that can run up to 8 threads. The parallel efficiency was very good on that scale, but one of our recommendations was to check the scaling in large node sizes.

The audit also detected a significant variability on the total number of instructions due to different number of invocations of one of the parallel routines. This behaviour was not expected by the user who wanted our help to investigate it further.

The first step has been to install the code in MareNostrum III machine. The first input analysed was the airfoil case that it is distributed with the sources. We identified a poor scaling efficiency of this test case even with 2 processes with respect to the serial execution. Looking at the traces we detected the number of iterations was increasing with the number of threads causing the scalability problem observed. Artenics sent us a modified source of this example as a new input case. With this input the scaling has improved significantly. The efficiency of both case studies are plotted in Figure 16.

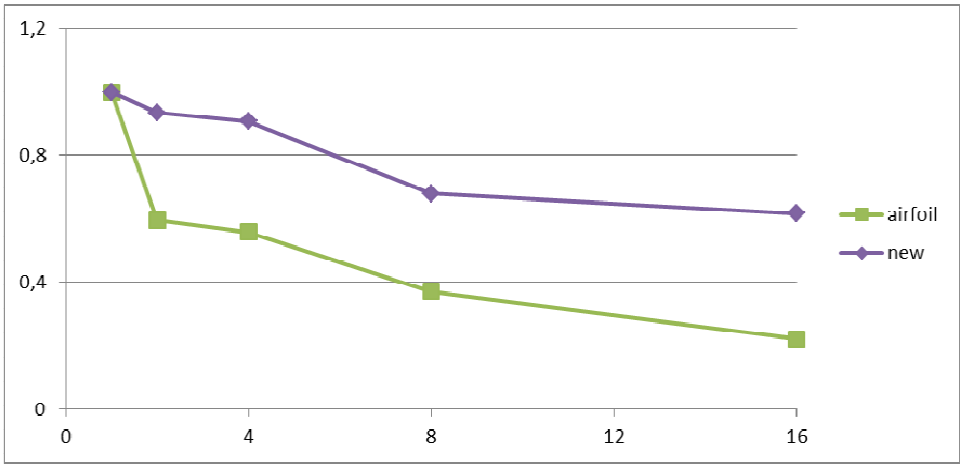

Figure 16: Global efficiency @ MareNostrum III

Most of the performance degradation we see in the new input case with more than 4 threads is due to a reduction in the IPC probably caused by the sharing of cache resources within a socket (each compute node is composed of two 8-core Intel Xeon processors).

Both cases tested have the variability on the number of calls of the parallel loop, so they can be used to check the problem identified in the performance audit. We have detected some randomness on the executions that we are currently trying to eliminate with the support of the code owner to have a fair comparison of the different runs and to check if the non-deterministic behaviour of the code is the source of the variability on the number of invocations.

This performance plan is progressing very slowly because of the small size of the company.

3.3 ICON Performance plan (BSC)

While JSC was auditing the scalability of the ICON code, the user requested BSC to analyse the performance and communication at node level. The main concerns from the user are the usage of cache memory and the potential for vectorization of the code. He provided a real size input of a pure atmospheric simulation with a reduced number of time steps.

As the audit has been done with the Scalasca framework, the approach of the performance plan is to use Paraver flexibility to look in detail the behaviour of the different application phases and use the Dimemas simulator to analyse the code sensitivity to the network parameters.

The initial analysis targeted the balance and the communications and we have requested a new tracefile to the user enabling sampling to analyse the cache usage. The preliminary results of potential improvements identified are:

- Internal phases inside the iterations show an unbalanced region where only rank 0 is computing and other ranks wait in an MPI_Bcast. It seems this may correspond to an I/O that may be interesting to overlap as it takes approximately 10 out of 66ms.

- MPI_Irecv calls represent an 8-9% of the time with an average duration of 15 microseconds. Dimemas simulations indicate ICON could be sensitive to network latency. Using persistent calls and emitting all the Irecv in one step should reduce the cost of performing each Irecv separately.

- Asynchronous communication patterns with the receive call after the computation are used for both small and large messages (bigger than 16KB). Moving the MPI_Irecv before the computation would start the rendez-vous reducing the delay of the MPI calls for large messages.

- Each rank communicates first with its neighbour with the lowest id, thus causing end-point contention that was confirmed by Dimemas simulations. An optimized scheduling or even some randomness on the order would eliminate this effect.

- There is a computing region of ten milliseconds at the beginning of each internal step of the iteration (just after calling the close of NetCDF). This region has a 15% of imbalance due to different amount of instructions, and it would be interesting to parallelize it with OpenMP. It may also be interesting to consider its vectorization. To obtain more insight we are waiting that the user sends us the new trace activating sampling.

Figure 17 shows a zoom of the MPI calls and the number of instructions for the first 3 internal phases inside one of the iteration for the first 32 ranks. The yellow region correspond to the broadcast described in the first bullet of preliminary results and the dark blue area on the bottom image identifies the 10 milliseconds computation referred on the last bullet.

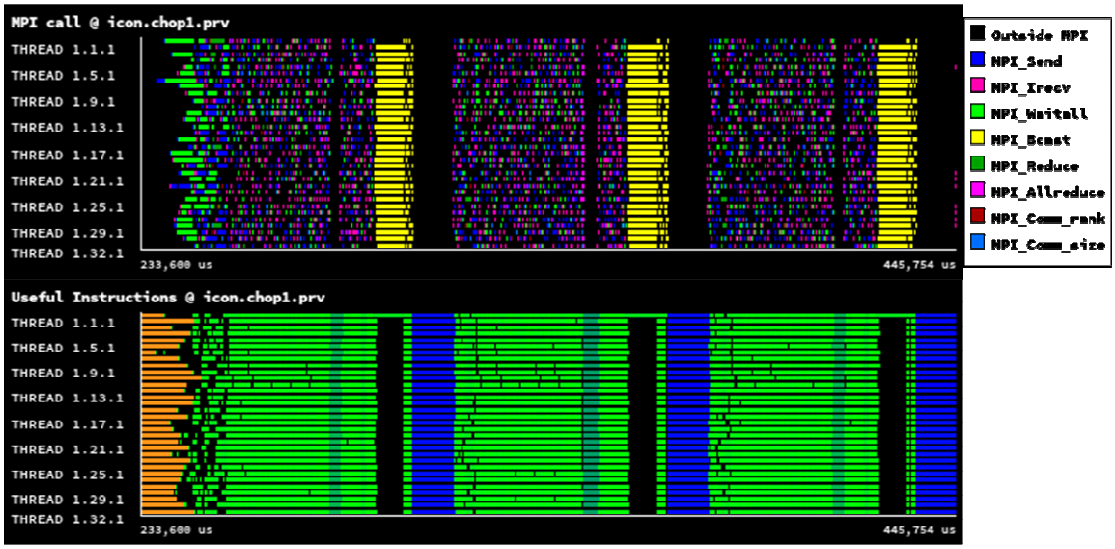

Figure 17: ICON internal phases inside iterations

3.4 SHEMAT Performance plan (RTWH Aachen)

The performance audit of the SHEMAT suite identified two performance issues in the application. First, the application suffered from computational imbalances among the processes, and second, the application performed a lot of MPI communication.

The computational imbalance among the processes occurred in the formapproxjacobian function of the application. The computational imbalance was not because of imbalance in the number of instructions executed by the processes (work imbalance), but due to the rate at which these instructions were executed. Figure 18 shows the snapshot of the application trace (a) and the instructions per second for the same region of the trace (b). It is clear that the region with a lower instruction rate takes more time, causing imbalance among the processes.

The difference in instructions rate points to an imbalance in IPC, which can be the result of difference in number of cache misses. The performance plan will look at these regions to identify the cause of lower IPC. If possible, it will also try to recommend any remedies to address the problem.

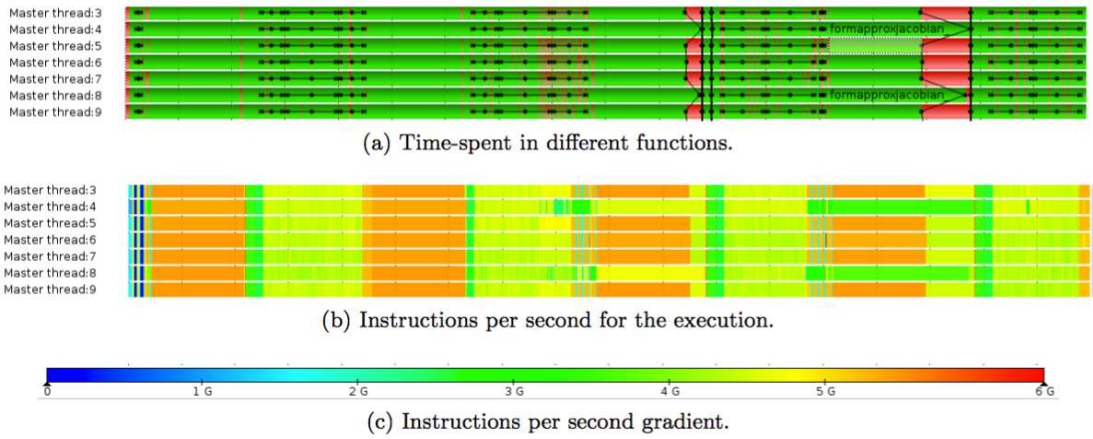

Figure 18: Time spent and instruction rate snapshot of trace of SHEMAT suite

The second performance problem identified in the application was the high rate of MPI calls, especially the MPI_Allreduce() call. These calls not only take up a significant amount of application time, but may also hinder the future scalability of the application. Table 3 shows the MPI calls called very frequently in the application, and the percentage of the total execution time they occupy. It is clear that MPI_Allreduce() takes a significant amount to time, especially when scaled to larger number of processes. Note: some of the time in MPI_Allreduce is due to the waiting time induced by computational imbalance.

The performance plan will look at the reasons of high frequency of MPI calls, and if possible, suggest changes to reduce them.

	Percentage time				Occurrence			
	12	24	36	48	12	24	36	48
MPI time	5.14 %	10.94 %	18.97 %	20.29 %				
MPI_Allreduce	4.23 %	8.85 %	16.72 %	16.65 %	38556	82008	127188	185520
MPI_Waitany	0.32 %	0.24 %	0.25 %	0.86 %	73000	155824	242112	443360
MPI_Iprobe	0.23 %	0.68 %	0.55 %	0.27 %	3700399	10424709	9402048	4797867
MPI_Test	0.22 %	0.62 %	0.50 %	0.24 %	3700183	10424277	9401400	4797003

Table 3: Frequently called MPI functions and their percentage of total time

3.5 GS2 Performance plan (NAG)

The findings in the POP Performance Audit on the GS2 application indicated two main areas of investigation. First, to investigate the poor instruction scalability, it was found that the instructions Scalability decreases to 53.9% going from four nodes up to 48 nodes. This means that the number of instructions executed almost doubles which severely impacts on performance and scalability. Shown in Figure 19, the computational scalability decreases significantly as the number of processes increases and is the main contributor to the poor scalability of the application and that is mainly due to the instruction scalability.

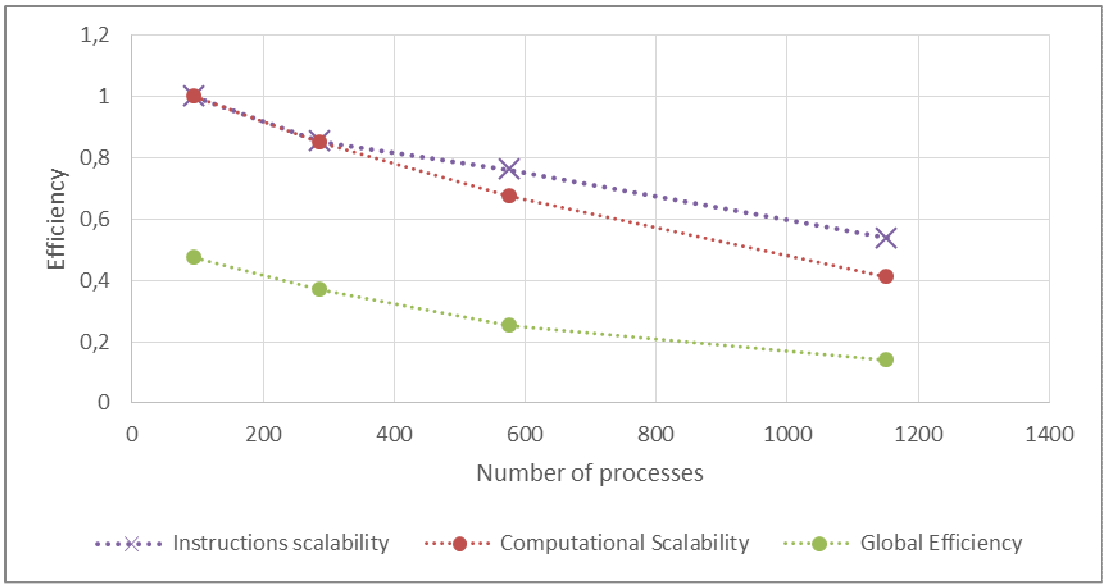

Figure 19: Degradation of the global efficiency and its correlation with the instructions

The audit did not have enough data to investigate where in the application this increase in instructions was. The performance plan will locate the areas in the code where this dominates and through investigation of the source code and discussion with the developers the potential for reducing this will be assessed and the effect this would have on performance estimated.

Second, to compare the data in the audit, that was performed on a smaller problem size than normal, with the full production problem size and contrast the conclusions. The approach for this will be to use the standard audit methodology on the larger problem size, due to the increased amount of data this would have taken too long for an audit. Depending on the similarities or differences further investigation may be made. We will also confirm whether the instruction scalability is as significant and located in the same regions.

Finally, an investigation into the vectorization of the code was requested. The vectorization will be investigated by using the appropriate compiler flags, due to the large size of the code base only key functions will be investigated for vectorization. These will be located by frequency of use and number of instructions completed. The potential for improvements and how they would be achieved will be assessed.

3.6 EPW Performance plan (JSC)

The POP Performance Audit of EPW identified load imbalance as the primary inefficiency, even at the fairly small scale of 48 processes on two compute nodes. During the main calculation, the right-hand two-thirds of Figure 20, four processes performed no work and nine others were significantly under loaded. A different load imbalance issue, combined with some serial code, affected the initial calculation in the left-hand third of Figure 20. Resolving these load

imbalance issues, and improving scalability, is the goal of the Performance Plan started in September.

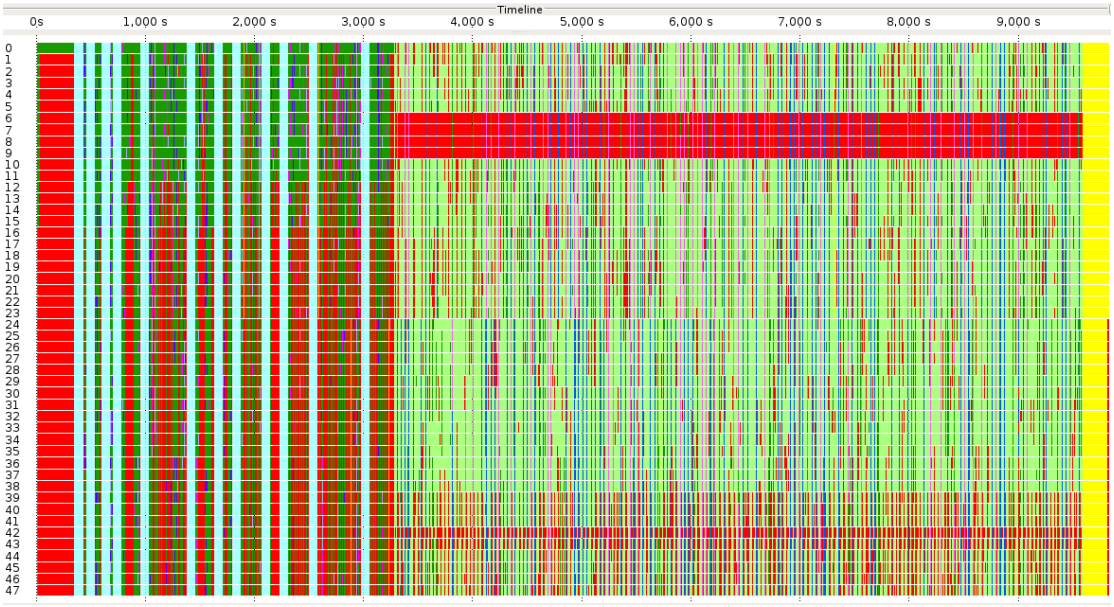

Figure 20: Load imbalance with 48 processes

The first aspects for investigation are how the load balance changes with a finer (higher-resolution) grid and with larger numbers of processes. Since certain grid-points involve less work than others, different partitioning/distribution schemes will be investigated to reduce load imbalance.

4. Recommendations for tools developers

The POP project does an extensive use of the tools and it is a good framework to identify recommendations for tools developers. From one side we have the recommendations based on the methodology and the usage of the tools for the POP studies from people expert on the tools. We also have recommendations and suggestions based on the experience and questions by some partners that did not had a previous expertise using the project tools. Both sources of recommendations are complementary and useful to improve the performance tools.

With respect to the general recommendations, maybe the most important is that Python is getting more important, both as a middleware gluing compute kernels together, and as a programming language itself. Tools have to be prepared and improve Python support. For instance, in the case of Extrae (BSC instrumentation) there is a previous support for Python with some limitations to intercept the multi-process module, which currently limits its applicability to some of the studies that use that module. This recommendation is not specific for the tools of the consortium but to the performance tools developers' community.

With respect to the methodology being used in the audits, we identified a need to automatically compute the efficiency metrics. BSC that had been using these metrics over the last years has already a python script to automatize this process while JSC is working to add these metrics on the automatic Scalasca analysis. We also identified the need to check the clock frequency as part of the metrics computed in the methodology, as in several cases some partners have identified a significant variability.

Finally there have been many questions and comments from the partners that started to use the tools developed by JSC and BSC. They are very useful to identify potential functionalities to improve the tools. As examples, the partners recommended to implement some mechanism to validate the tools installation on a new system and to improve documentation for advanced users.