D3.3 First report on second level services
Version 1.0

Document Information

Contract Number	101143931
Project Website	www.pop-coe.eu
Contractual Deadline	M12
Dissemination Level	PU
Nature	R
Authors	José Gracia (USTUTT)
Contributors	Joachim Jenke (RWTH Aachen), Ondřej Vysocký (IT4I)
Reviewers	Christian Terboven (RWTH Aachen)
Keywords	POP second-level services, Proof of Concept, Advisory Study, Cor-rectness Check, Energy Efficiency Audit

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101143931. The JU receives support from the European Union's Horizon Europe research and innovation programme and Spain, Germany, France, Portugal, and the Czech Republic.

Change Log

Version	Author	Description of Change
v0.1	José Gracia	Initial version of the POP LaTeXtemplate
v0.2	Joachim Jenke	Add Correctness Check and SLS example
v0.3	José Gracia	Add Proof of Concept and Advisory Study
v0.4	Ondřej Vysocký	Add Energy Efficiency Audit
v0.5	José Gracia	Executive summary
v0.8	José Gracia	Update after internal review
v1.0	José Gracia	Released to the EC

Contents

Executive Summary

This document describes second-level services of the project POP3.

While the main service remains performance assessments, POP3 offers another four services with the aim to either transfer skills and expertise to customers (Proof of Concept, Advisory Study), or else to assess other characteristics of the customer's code such as Correctness Check and Energy Efficiency Audit.

All four services are defined in separate sections each. In particular, we define the objectives and results, followed by a description of the service's workflow and procedures.

We conclude the document with a brief description of a completed second-level services as an example for this kind of activity.

1 Introduction

The primary service activity of POP3 is performance assessments or audits. In addition, POP3 offers four distinct second-level services (SLS). Broadly speaking, SLS either aim to train and assist customers in implementing recommendations from the assessments, or else are concerned with other aspects which are not related to code performance as such.

In particular, during *Proof of Concept* activities, POP staff demonstrate on the customer's code how to implement suggested optimizations and evaluate their impact on performance, while in *advisory studies*, POP staff takes an advisory role and the customers implements the optimizations, etc. *Energy efficiency audits*, on the other hand, explore how to optimise the energy consumed by the application. Finally, *correctness checks* aim to assert that an application uses MPI and OpenMP formally correct.

Second level services may be recommended by POP staff as a follow-up to a performance assessment, or may be requested by customers.

The following chapters define for each SLS the objectives and outcomes, and then detail the service workflows and procedures.

2 Service Definition: Proof of Concept

2.1 Objectives and Outcome

POP performance assessments may recommend optimizations that require advanced HPC techniques. Not every customer will posses such expert skills and knowledge; they will thus not be able to implement recommendations on their own.

In some cases, we envisage that customers will benefit from some help on the initial optimization steps. The objective is to show them the best programming practices, steering the refactoring efforts in directions to more productively improve the code. The objective is not to take over their code development role, but to accompany them during initial steps of applying a new approach or programming model feature and provide very immediate feedback of the performance impact as the refactoring is being carried out.

The aim of the Proof of Concept activity is thus to transfer the necessary skills from POP HPC experts to the customer by hands-on demonstration of relevant HPC techniques on the customer's code. Customers will be enabled to do similar optimizations in the future on their own. We also expect, that such skill sets will disseminate throughout the customers's organization and beyond.

The concrete outcome of the Proof of Concept activity is

1. a report which justifies and explains the employed techniques and further presents the results, and

2. an optimized version of the customer code.

Both are essentially authored by POP staff with support by the customer.

2.2 Service Workflow and Procedures

Proof of concept activities fall into three consecutive phases:

- the elaboration of a Proof of Concept plan,

- the implementation of optimizations as outlined in the plan, and
- a final report which justifies and explains the employed techniques, and presents results.

2.2.1 Proof of Concept plan

Key point for any Proof of Concept work is a very close interaction with the customer whose knowledge about the algorithms, code structure and usage scenarios are essential for the success of this service. The customer thus has to nominate a direct technical contact in order to keep communication overhead as low as possible.

Starting point of the actual service activity is the recommendations from a POP performance audit. POP expert and the customer will assess the feasibility and potential gain of all recommendations.

The next step is to define the *scope of the Proof of Concept activity*, i.e., to select a subset of recommendations which shall be addressed given the resource constraints on POP and customer side. Typically, each activity will address one major recommendation only, but addressing a few related minor issues is done as well. Also, if the same technique needs to be applied in several places of the code, POP staff will do optimizations on the code only in one or a few exemplary places. The aim is to teach and demonstrate techniques, not a assume responsibility for a complete optimization of the customer's code.

Also, POP staff will suggest a suitable *performance metric to monitor progress* and to assess the outcome. Obtaining this metric will typically entail some element of recording performance traces or profiles as is done in performance assessments.

Finally, POP staff prepare a document called *Proof of Concept Plan* which contains the following sections:

1. Background
2. Previous assessments and recommendations
3. Scope of the activity
   - 3.1. Addressed recommendations and code refactoring
   - 3.2. Use-case and evaluation metrics
   - 3.3. Target system

2.2.2 Proof of Concept implementation

The Proof of Concept implementation step performs the actual work of this service, implementing the suggestions based on the Proof of Concept plan. It will include regular interaction between the customer and POP staff to showcase the progress to the customer as well as obtaining feedback from the customer, helping with the implementation and ensuring good customer satisfaction. This is obviously important in the case of the result being implemented directly into the original application, but also in the case of kernel extraction and mock-up.

If necessary, this implementation phase may include additional elements of performance analysis to better understand the specific performance issue at hand.

Along with the implementation the progress will be monitored with the selected performance metric.

2.2.3 Proof of Concept reporting

Due to the close interaction of POP staff and the customer's technical contact, the customer is always up to date with intermediate steps and results. The monitoring based on the predefined performance metric minimizes the risk of miss-optimisations in an early stage and will allow direct feedback about the current status.

At the end of the activity, the POP expert will prepare a document called *Proof of Concept Report* which contains the following sections:

1. Background
2. Previous assessments and recommendations
3. Scope of the activity
   - 3.1. Addressed recommendations and code refactoring
   - 3.2. Use-case and evaluation metrics
   - 3.3. Target system
4. Implementation
   - 4.1. *Subsections as necessary*
5. Conclusions

The first three sections of this report, are identical as in a *Proof of Concept Plan*. In fact, we expect these to be a literal copy in most cases and only updated when necessary. Then follows the main section Implementation. It report which code modifications have been done to achieve the expected goal, in particular emphasising which feature of the particular programming model have been exploited. Progress and final result are assessed according to the metrics specified in the plan above. The final section summarises the activity, states which goals have been reached and which have not. In the later case, a brief justification or explanation is given. If possible this section should also state whether the particular programming model and techniques were sufficiently adequate and expressive to achieve the goal.

In addition, the customer will receive any code modifications done by the POP expert during the activity.

3 Service Definition: Advisory Study

3.1 Objectives and Outcome

The service Advisory Study is very similar in spirit as the Proof of Concept. Both aim to train the customer in the usage of advanced HPC techniques. The main difference is, that the customer takes a much more active role and does the actual code modifications while the POP expert takes a advisory role only. Customers will be enabled to do similar optimizations in the future on their own. We also expect, that such skill sets will disseminate throughout the customers's organization and beyond.

The concrete outcome of the Advisory Study activity is

1. a report which summarises the employed techniques and results, and
2. an optimized version of the customer code.

The report is jointly authored by POP staff and the customer. The modification of code is done by the customer.

3.2 Service Workflow and Procedures

Advisory Study activities fall into the same three consecutive phases as Proof of Concept activities. Namely, planning, implementation, and reporting.

The planning phase is identical to the Proof of Concept. Starting from the recommendations, customer and POP expert define the scope of the activity and jointly prepare a document called *Advisory Study Plan*. This document has the same structure as the Proof of Concept Plan:

1. Background
2. Previous assessments and recommendations
3. Scope of the activity
    3.1. Addressed recommendations and code refactoring
    3.2. Use-case and evaluation metrics
    3.3. Target system

During the implementation phase, the POP experts mostly takes an advisory role only. Code modifications are done by the customer after discussions with the POP expert. The POP expert suggest specific modifications and, if necessary, provides (training) material or simple usage examples. POP experts will teach the customer how to obtain performance traces and determine POP metrics to monitor progress. If necessary, the POP expert will do a re-analysis of customer-provided performance traces to assess the effectiveness of optimisations when POP metrics are not sufficiently specific or adequate. Code optimization and evaluation are repeated as necessary.

Finally, after conclusion of the implementation phase, At the end of the activity, the POP expert and the customer will jointly prepare a document called *Proof of Concept Report* which contains the following sections:

1. Background
2. Previous assessments and recommendations
3. Scope of the activity
    3.1. Addressed recommendations and code refactoring
    3.2. Use-case and evaluation metrics
    3.3. Target system
4. Summary of implementation and results
5. Conclusions

Again, the document is very similar to a Proof of Concept Plan. The main difference is that the forth section is only a summary of the implementation activities and a presentation of the final results. Detailed justification and explanation of the techniques involved are not necessary, as the customer has done this rather than the POP expert.

If possible, the customer is encouraged to share any code modifications with POP. These might be used in other POP activities, such as input for programming patterns and best practices in the Co-Design workpackage, or even as examples in training activities.

4 Service Definition: Energy Efficiency Audit

4.1 Objectives and Outcome

The goal of the energy efficiency analysis is to compare various hardware platforms for energy consumption and energy efficiency when executing the assessed application. The assessment should provide insight into the hardware behavior and compare the reached energy efficiency to the platform's peak efficiency.

When using a system, which allows controlling hardware power management knobs, the study may show power-runtime trade-off and identify hardware configuration for each instrumented part of the application that brings maximum energy savings without a performance penalty or with a predefined penalty.

4.2 Service Workflow and Procedures

In POP, to express the energy efficiency of an application or its region, we will use the metric defined in Equation

$$EnergyEfficiency = \frac{averagePerformance}{averagePower}$$

where averagePerformance can be expressed in a number of floating-point operations per second. However, some algorithms use an alternative unit, which expresses the algorithm efficiency better. One example is a number of lattice updates per second in the case of Lattice-Boltzmann method-based algorithms since each implementation may require a different number of operations to obtain the lattice update value.

The algorithm-specific performance metric can be used if reported by the application, otherwise the number of executed floating-point instructions is measured using vendor-specific performance counters. Availability of these counters is limited by hardware vendor and CPU/GPU model. E.g. AMD EPYC CPUs allow to monitor event EVENT_RETIRED_SSE_AVX_FLOPS, which does not distinguish between SSE and AVX instructions, but no separate monitoring of each instruction set.

An essential requirement to perform energy efficiency analysis is the availability of a power monitoring system exposing power consumption at the moment of reading or energy consumption to user space. The relation between power and energy is

$$Energy = Power \times Time$$

where Power is Watts [W], Time in seconds [s], and energy in Joules [J]. Energy is often presented in Watt-Hours [Wh], which equals $1Wh = 3600J$.

Since power consumption is not constant, the power monitoring system reports energy consumption but samples power and accumulates these measurements to a single counter. Thus, energy is rather expressed as

$$Energy = \frac{\sum powerSamples}{samplingFrequency}$$

Various HPC systems provide a range of power monitoring systems, each having a specific power sampling frequency, precision, and power domains, which are measured. To obtain comparable measurements from several different HPC systems, we always report compute node energy consumption. The power monitoring of compute nodes is typically provided by out-of-band monitoring (using IPMI, redfish, or similar), which works at a low frequency and is

active GPUs	0/8	1/8	2/8	3/8	4/8	5/8	6/8	7/8	8/8
GPU0 [W]	55	54	54	54	54	56	398	397	398
GPU1 [W]	52	52	51	51	51	397	398	396	397
GPU2 [W]	51	51	51	50	50	51	51	55	398
GPU3 [W]	52	52	51	51	51	52	52	398	399
GPU4 [W]	53	60	398	398	398	398	398	398	398
GPU5 [W]	52	398	397	398	398	397	398	397	398
GPU6 [W]	55	54	54	57	396	398	398	398	398
GPU7 [W]	52	52	51	398	398	394	394	393	398
CPU 0 [W]	92	94	92	94	96	97	96	99	99
CPU 1 [W]	95	95	99	99	98	98	102	100	102
CPU+GPU [W]	609	962	1298	1650	1990	2338	2685	3031	3385
Node (iLo) [W]	1129	1490	1842	2210	2570	2930	3290	3650	4010
Node baseline [W]	*520*	*528*	*544*	*560*	*580*	*592*	*605*	*619*	*625*

Table 1: Node power baseline of IT4Innovations Karolina accelerated partition (2 AMD EPYC 7763 (280 W) + 8 Nvidia A100 (400 W).

usually available to administrators only. In the majority of systems, a monitoring of some compute components is available, which uses significantly higher sampling frequency and can be read by the user (exposed in-band). To overcome the limitation, we combine information from both in-band and out-of-band monitoring to construct a Node power baseline, which expresses the difference in power readings from these monitoring systems for the specific node utilization. Table 4.2 presents the node power baseline of an accelerated node of EuroHPC system Karolina.

The maximum energy efficiency is reached when the hardware is delivering its peak performance, which is limited by the memory or the compute components. It can be expressed using a roofline model of the specific hardware. When the hardware is executing a computation, it uses the highest possible frequency for the executed instruction set until a power or thermal capping system activates and reduces the frequency to maintain the power and temperature limit. The maximum frequency of computing units is not beneficial to bring higher performance when executing a memory-bound workload while the power consumption is high.

Besides expressing the energy efficiency of an application and its selected regions, we monitor additional metrics explaining to us the behavior of the hardware.

- CPU core frequency
- CPU uncore (Data Fabric) frequency (Intel, AMD)
- GPU streaming multiprocessor frequency
- power limit
- power capping system activity
- temperature
- vectorization ratio
- since arithmetic intensity is typically not possible to measure using hardware performance counters, we use an alternative metric, Computational intensity

$$Computational\ Intensity = \frac{Instructions\ executed}{L3\ cache\ misses}$$

The application profile consists of a power consumption timeline and a graph of application regions, each with its metrics.

In systems that have scalable frequencies of both compute units and memory subsystems, it can be possible to reduce power consumption while not impacting performance or improve performance by down-scaling one of these frequencies to shift the power budget to the other frequency. The application power timeline may show us a change in power consumption, which indicates a change in hardware usage. Thus, a different hardware configuration is required. The methodology of the application dynamic tuning comes from the Horizon 2020 project READEX [1].

These metrics and the dynamic tuning of the hardware power knobs are provided by the MERIC runtime system (developed by IT4I@VSB).

5 Service Definition: Correctness Check

5.1 Objectives and Outcome

It is a common observation that parallel codes will execute correctly most of the time, but crash or produce corrupted results at random times, under different circumstances, or in different environments/systems. Possible reasons include data races and incorrect use of parallel libraries such as MPI and OpenMP. Often, users are not even aware that their codes are erroneous: errors have not yet appeared or have gone unnoticed.

Certain types of these issues can be detected by recording data movements across threads and process boundaries at runtime. These kind of tools are not well-known and require expert skills which are rarely available outside of HPC centres. In particular recognizing false positives reported by the tools and assessing the severity of issues requires experience.

The aim of the Correctness Check activity is thus to assess the correctness of customer codes with respect to issues such as data races in OpenMP, conformance to the MPI standard, usage errors or non-portable code, and errors in hybrid MPI + OpenMP usage.

The concrete outcome of the Correctness Check activity is

1. automated reports by correctness checking tools, and

2. an assessment of the relevance and severity of reported issues.

Both are provided by POP staff.

5.2 Service Workflow and Procedures

Correctness checks to be performed as part of a second level service depend on the characteristics of the parallel application. Dependent on the language of the application, different compilers provide a different set of analyses. We distinguish

- base-language analyses performed with Sanitizers,
- OpenMP-specific data race detection performed with Archer, and
- general MPI correctness analysis performed with MUST.

[1] https://www.readex.eu/

A correctness checking report should contain enough detailed information for the application developer to reproduce the result and therefore to reiterate the analysis from time to time during code development (or ideally integrate the analysis into a CI/CD setup).

The general approach of correctness checking in POP SLS is to identify sources of undefined behavior (UB). The reported issues are definitely programming errors as they violate the base language (C/C++/Fortran) standard, the OpenMP standard or the MPI specification.

Any issues reported by the tools should be verified in the source code to provide actionable solutions, if possible.

5.2.1 Base language analysis with Sanitizers

The Sanitizers help to identify general UB (UBsan), access to invalid memory addresses and memory leaks (Asan), and reading uninitialized memory (Msan). Most Sanitizers are available for all Clang-based compilers (including aocc and icx). All sanitizers are enabled by passing `-fsanitize=address|memory|undefined` to the compile and link steps.

5.2.2 Data race analysis with Archer

Archer builds on another Sanitizer, the ThreadSanitizer. To make the analysis aware of OpenMP synchronization semantics, the Archer runtime library has been developed. The tool is part of LLVM since several years now. The library is now fully supported by many Clang-based vendor compilers including AMD, HPE/Cray and Intel. Given the more advanced Fortran frontends from the vendors than what is available in LLVM, debugging support for Fortran codes should be much better with these compilers than with flang. The necessary compiler and linker flags are `-fsanitize=thread -g -fno-omit-frame-pointer`. The latter flags are important for useful debugging output in case of reported issues. The tool can be used with any optimization flags. Using the flags also used in production is actually encouraged. With a proper installation, Archer should be loaded by default, if the application was compiled with ThreadSanitizer flag. An example execution looks like:

```
OMP_NUM_THREADS=4 ARCHER_OPTIONS=verbose=1 \
TSAN_OPTIONS=ignore_noninstrumented_modules=1 \
./application
```

The verbose option for Archer leads to an initial output that confirms that Archer is active during execution. The Tsan option is important to suppress false alerts resulting from uninstrumented libraries, such as the MPI, OpenMP, or other third-party libraries.

As a general rule, debugging can introduce runtime overheads of 10-20x. Therefore input data that reduces the base runtime to about 10 seconds or a minute are helpful. While ThreadSanitizer analysis scales well with the application scalability, it has performance issues for some memory access patterns that are unfortunately quite common in HPC/scientific computing. The performance issue does not occur up to 4 threads, but can result in severe runtime overhead for using more threads and is amplified by NUMA effects/NUMA distance of the threads.

5.2.3 MPI correctness analysis with MUST

For correctness analysis with MUST, the application code should be built with debug information. No further instrumentation is necessary, although a binary created for analysis with Archer can be used for more detailed MPI-specific memory analysis. For a basic analysis, the application can be executed just like:

```
mustrun --must:mpiexec srun -n 4 ./application
```

By default, the report is provided as an html file with additional files stored in a subdirectory. The report can also be sent by mail, if the machine executing mustrun can send out email to your mail address. Similar to the considerations above, using a small input data set that executes on a low number of processes can help to get started. But the tool has also been successfully used with up to 16k MPI processes, which requires a bit more tuning with runtime flags.

6 Example second-level services

Second-level services can be done only after an initial performance assessment service. In many cases, second-level services may not be required nor requested. Typically, SLS themselves take between 3 to 6 months to conclude. It is therefore not surprising, that only one SLS, a Correctness Check, has been completed at the time of writing this document in project month 12. Another three Advisory Studies are in progress, and several other SLS are in preparation.

Since Correctness Checks are a new type of SLS and have only been formalized recently, the following section illustrated the expectable outcome of such activities. Examples of other SLS will follow in subsequent deliverables as they are completed.

6.1 NEST Correctness Check

From analysis tool point of view, the nest-simulator [2] is a Python script that loads the compute kernels as python modules. The compute kernels are C++ code parallelized with MPI and OpenMP and compiled to a dynamically loadable library.

For this kind of applications we can apply different correctness analyses: base-language analyses, OpenMP-specific data race detection, and general MPI correctness analysis. We start with basic sanitizers like AddressSanitizer and MemorySanitizer. Given the application setup with Python as the startup skeleton, we cannot apply MemorySanitizer. For Nest, AddressSanitizer did not report any issues.

6.1.1 Data race analysis with Archer

The next analysis is OpenMP-specific data race analysis with Archer. Again, given the Python driver used by the application, a recent version of LLVM (> 16.0) is necessary for the analysis. Since the nest library is configured with CMake, the following configuration flags instrument the library code for data race detection and solve some compatibility issues with the compiler:

```
-DCMAKE_CXX_FLAGS="-fsanitize=thread -g -fno-omit-frame-pointer
    -DBOOST_NO_CXX98_FUNCTION_BASE=1 -Wno-deprecated-builtins" \
-DCMAKE_C_FLAGS="-fsanitize=thread -g -fno-omit-frame-pointer
    -DBOOST_NO_CXX98_FUNCTION_BASE=1 -Wno-deprecated-builtins" \
```

To allow data race detection with the Python driver setup, ld-preloading the ThreadSanitizer runtime library is necessary:

```
env ARCHER_OPTIONS=verbose=1 LD_PRELOAD=/path/to/libclang_rt.tsan.so
    \
    python3 ./hpc_benchmark_tsan.py
```

[2] nest-simulator.org

The archer options setting to verbose allows to verify that the tool setup is working. The initial execution of Nest with ThreadSanitizer resulted in a report of 6 distinct data races. The most surprising data race was reported for a member function marked with the `const` keyword. Subsequent code review identified the writing to a member variable marked as mutable as the cause of the data race. We suggested to different solutions: replacing the boolean by an atomic boolean member, or removing the boolean member at all, if it is not necessary.

Another reported data race involved unsynchronized pushing of elements to a vector and getting the size of the vector. As the accesses occur rarely and from a single source line, the suggested solution was to add an OpenMP critical region for the violating line.

The last important finding is related to a hand-crafted boolean vector, which was for thread-safety already padded to use int64 rather than single bits. While most accesses to the values of the vector were synchronized, in one case such synchronization was missing. The race could eventually even have caused a deadlock in the code, but given the timing and compiler optimizations, such deadlock was never observed. Different solutions to solve the data race were suggested.

The application developers immediately addressed all issues in their Github repository.

6.1.2 MPI correctness analysis with MUST

We focused the MPI-specific analysis mainly on single-threaded execution, because not all analyses were prepared for multi-threaded analysis at that time. Nevertheless, the only issue reported by MUST is related to multithreaded execution of the code, because the selected MPI threading level `MPI_THREAD_FUNNELED`, where exactly one thread is allowed to perform all calls to MPI functions, is not compatible to using MPI communication in OpenMP `single` regions. The suggested solution was to replace the OpenMP `single` regions by pairs of OpenMP `master`+`barrier` regions.

Acronyms and Abbreviations

- BSC: Barcelona Supercomputing Center
- CA: Consortium Agreement
- CAdv: Customer Advocate
- DoA: Description of Action (Annex 1 of the Grant Agreement)
- EC: European Commission
- FZJ: Forschungszentrum Jülich GmbH
- D: deliverable
- GA: General Assembly / Grant Agreement
- HLRS: High Performance Computing Centre (University of Stuttgart)
- HPC: High Performance Computing
- IPR: Intellectual Property Right
- INESC-ID: Instituto de Ennenharia de Sistemas e Computadores, Investigacao e Desenvolvimento em Lisboa
- IT4I: Technical University of Ostrava
- KPI: Key Performance Indicator
- M: Month
- MS: Milestones
- PEB: Project Executive Board
- PM: Person month / Project manager
- POP: Performance Optimization and Productivity
- R: Risk
- RV: Review
- RWTH Aachen: Rheinisch-Westfaelische Technische Hochschule Aachen
- SLS: Second level service
- TERATEC: TERATEC
- USTUTT (HLRS): University of Stuttgart
- UVSQ: Universite de Versailles Saint-Quentin-en-Yvelines
- WP: Work Package
- WPL: Work Package Leader