D7.3– Final co-design repository
Version [1.0]

Document Information

Contract Number	824080
Project Website	www.pop-coe.eu
Contractual Deadline	M36
Dissemination Level	PU - Public
Nature	R
Authors	Xavier Teruel (BSC), Manuel Rodrigues (BSC).
Contributors	Jesús Labarta (BSC), Marta García-Gasulla (BSC), Julian Morillo (BSC), Joel Criado (BSC), Christoph Niethammer (HLRS), Ramil Nabiev (HLRS), Martin Rose (HLRS), Lukas Maly (IT4I), Radim Vavrik (IT4I), Tomas Panoc (IT4I), Michael Knobloch (JSC), Jon Gibson (NAG), Jonathan Boyle (NAG), Federico Panichi (NAG), Jannis Klinkenberg (RWTH), Fabian Orland (RWTH).
Reviewers	Bernd Mohr (JSC).
Keywords	Co-design, kernels, patterns, best-practices.

Notices: The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement n^{o}824080.

Change Log

Version	Author	Description of Change
v0.1	Xavier Teruel	Initial version of the document.
v0.4	Manuel Rogrigues	Adding site improvements and populating first stage contents.
v0.9	Xavier Teruel and Manuel Rodrigues	Adding executive summary, introduction and conclusion sections. Including additional repository contents.
v1.0	Xavier Teruel	Applying reviewer's comments.

Contents

Executive Summary

This final deliverable consists of the POP Co-design repository filled with global data gathered until the end of the project. It has been published in the shape of a public website.

The project's proposal describes the distribution of the effort in terms of assigned *Person Months* per partner. Internally we have distributed the assigned effort among three types of activities: 1) management; 2) site extensions; and 3) site content. Management effort does not just involve leading the technical discussion but also providing the workpackage methodology through a set of guidelines. Site extensions activities are difficult to quantify (in terms of KPIs), but the document describes all the contributions performed during the period in that respect. Finally, for site contents, we correlate the partner's assigned effort with the global project KPI in terms of the number of micro-benchmarks (i.e., 20 at the end of the project as defined in *MS8 - Final Methodology Milestone*) to assign the partner's individual KPIs. The achieved number of kernels at the end of the project indicates the fulfillment of this milestone.

The actual contribution (sum of all the individual KPIs) has been maintained above 92% for micro-benchmarks, and above 100% in all the other elements; i.e., patterns, best-practices and other classification items. During the extension of this project (i.e., M37-M42), we plan to finalize all the reported items that were not closed at M36.

1 Introduction

This final deliverable gathers all the overall information available in the POP Co-design repository. In previous deliverables, *D7.1 Co-design repository structure* [2] and *D7.2 First co-design repository* [4] we already defined the structure and populated the repository with the interim contents at the mid-term of the project. Therefore, in this document, we report all the additions and extensions we have been working on during this last period.

Furthermore, this document also reports the progress to meet the targeted metrics of the POP2 Milestone *M8 Final methodology milestone*, as defined in the proposal of the project.

As already reported in the previous document, the repository has been published in the shape of a public website, containing a mix of downloadable codes, patterns, and best-practices. The available codes contain a set of experimental results, providing an idea of the behavioural pattern highlighted by them. To improve the website's navigability, we have included additional collections to allow accessing them by means of several criteria: programmming languages, programming models, algorithms, disciplines, and/or related metrics. The website can be visited in the following URL:

http://co-design.pop-coe.eu

Given that the website is publicly available, in the following sections we will only list the main contributions, and include a brief description of each element. The website can complement the descriptions gathered in this document and provide detailed information.

The only two additions, concerning the website structure presented in the previous deliverable, are detailed in section 3 and they include: 1) a new report collection; and 2) the addition of the hybrid POP metrics arising from *POP2 WP8 Tools and Methodology*.

1.1 Effort distribution and expected KPIs

The effort of this workpackage consists of: 1) management (i.e., *Mng.*); 2) site extensions (i.e., *Ext.*); and 3) contents to the *Resources for co-design* website (i.e., *Cont.*, it includes patterns, best-practices and kernel developments).

The effort distribution in terms of *Person Months* per partner were established in the project proposal [1]. The left-hand side part of Table 1 summarizes this information and it also includes the kind of effort according to previous categories.

Partner	*Person Months (PMs)*				*MS3: M7-M18*				*MS8: M19-M36*			
	Mng.	Ext.	Cont.	Tot.	K	P	BP	CI	K	P	BP	CI
BSC	3	6	18	27	2	2	2	2	3	3	3	0
HLRS			18	18	2	2	2	2	3	3	3	0
IT4I			9	9	1	1	1	1	1	2	2	0
JSC			9	9	1	1	1	1	1	2	2	0
NAG			18	18	2	2	2	2	3	3	3	0
RWTH			18	18	2	2	2	2	3	3	3	0
Total	3	6	90	99	10	10	10	10	14	16	16	0

Table 1: Expected effort distribution per partner: Person Months (PMs), Kernels (K), Patterns (P), Best-Practices (BP), and Classification Items (CI).

Management effort does not just involve leading the technical discussion but also providing the workpackage methodology through a set of guidelines. These guidelines were described in Section 1.2 of *D7.2 First co-design repository*, and they have also been updated and reviewed during this period. Section 2.3 contains a brief reference to these guidelines.

Even though activities carried out in site extensions are difficult to quantify, Section 2 describes all contributions in this regard.

Finally, for the website contents, we correlate the partner's assigned effort with the global project KPI in terms of the number of micro-benchmarks (i.e., 20 kernels at the end of the project) to assign individual KPIs. We reproduce the same schema that was already proposed in *D7.2 First co-design repository*, scaling the numbers[1] according to the period duration of the corresponding tasks: *T7.2 Gather repository data* and *T7.3 Kernel definitions*.

The right-hand side of Table 1 describes these individual partner's KPIs per milestone.

2 Extensions and changes in the repository structure

Although the initial specification of the repository was defined in previous deliverables, in this section, we will list a series of changes and extensions made to the initial design. These changes are motivated by the needs that we have encountered while populating the *resources for co-design* website.

[1]For this period we did not expect to contribute in *Classification Items* but just for these items arising from the reported elements (e.g., including a kernel that uses a new programming language will involve to include such programming language as a Classification Item).

2.1 Website new appearence

Given that *Resources for Co-design* is part of the POP website and therefore linked to it, we redesigned the entire website appearance and layout to reproduce the same look-and-feel. This was necessary because *Resources for Co-design* is an independent website that lives in the POP GitLab repository and was developed with a different set of tools. To achieve the same layout and appearance we had to perform substantial modifications in both the *HTML* and also *CSS* code to get the desired target. Firstly, we analyzed the style of the main POP website, by understanding its source code, and then tried to reproduce the same code patterns in the co-design website. This was done for the style of the website and also for the new components that we had to integrate, such as the main menu on the left side and the top-right navigation bar. Specifically, we had to re-code the entire *base.html* file and all the *CSS* files to address all the layout and appearance features. Furthermore, the redesigned website header and footer forced us to modify the content in files *header.html* and *footer.html*. Figures 1 and 2 highlight the differences between the old and the new version of the website. It can be seen that to achieve this new look we had to change almost every aspect of the website style and layout.

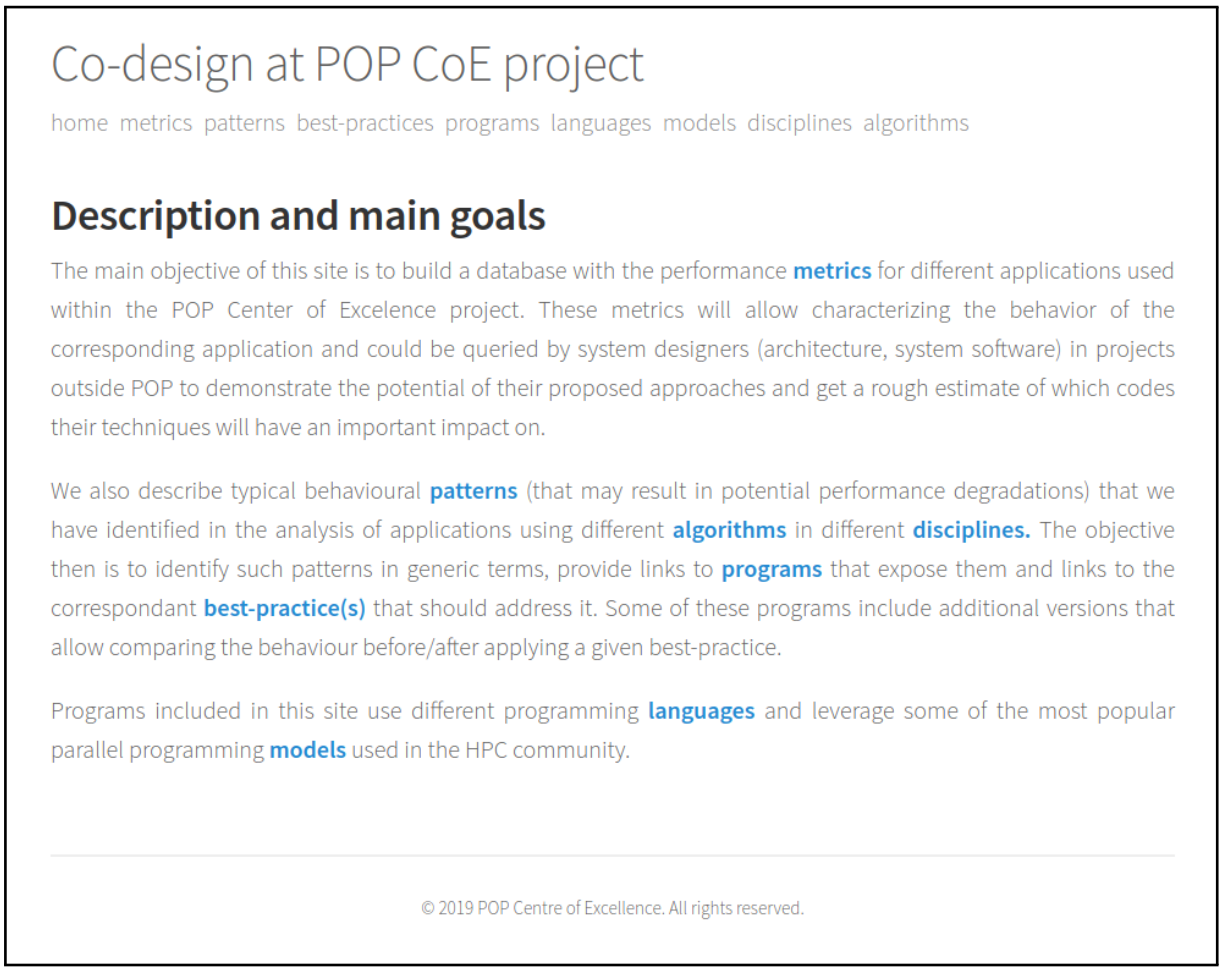


Figure 1: Resources for Co-design website old appearance and layout: the home page.

Figure 2: Resources for Co-design website new appearance and layout: the home page.

2.2 Search feature

We have included a new feature on the website: a search box to more easily find content on the website. This search tool is based on the Google programmable search engine tool. To this end, we had to create a google account associated with *Resources for Co-design* in order to configure the search engine and integrate the correspondent source code into the website. This search box is highlighted (in orange) in Figure 3. As it can be seen, this is a very simple and easy tool to use. The user just has to type the content in the search box (top right corner) and the search results are displayed in the page titled *Search Results*.

2.3 Website and repository improvements

We have re-structured and added new content to the guidelines that describe how to contribute to the Resources for Co-design Repository. Specifically, we have *deprecated* the documentation in the GitLab Wiki and added a new folder called **docs** that gathers and organizes all the information necessary to contribute with content to the repository (e.g. layout repository and kernel repository). This information is also accessible through the layout repository **readme**

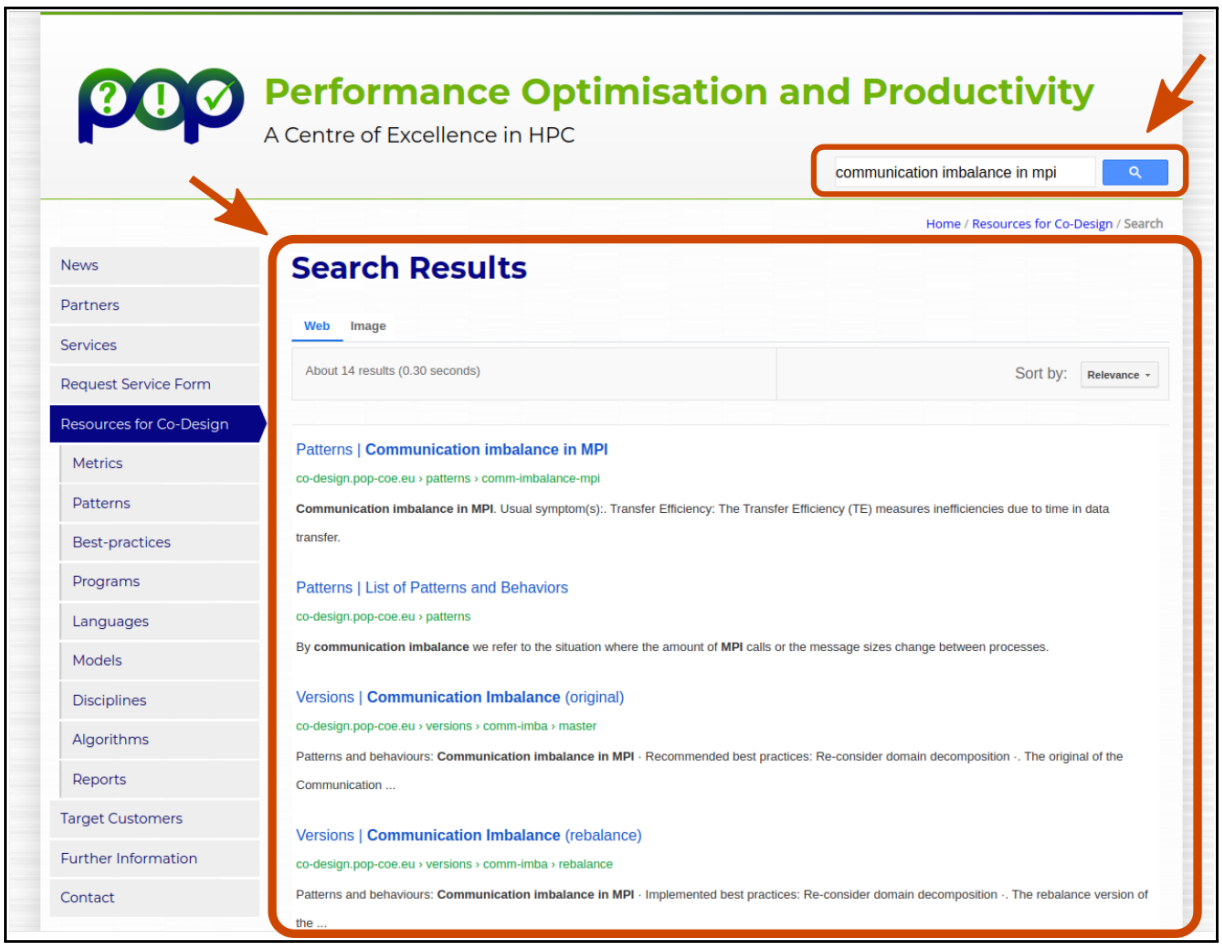


Figure 3: Resources for Co-design website new search feature.

file. Figure 4 shows where a user can find information on how to contribute to the layout and kernel repositories.

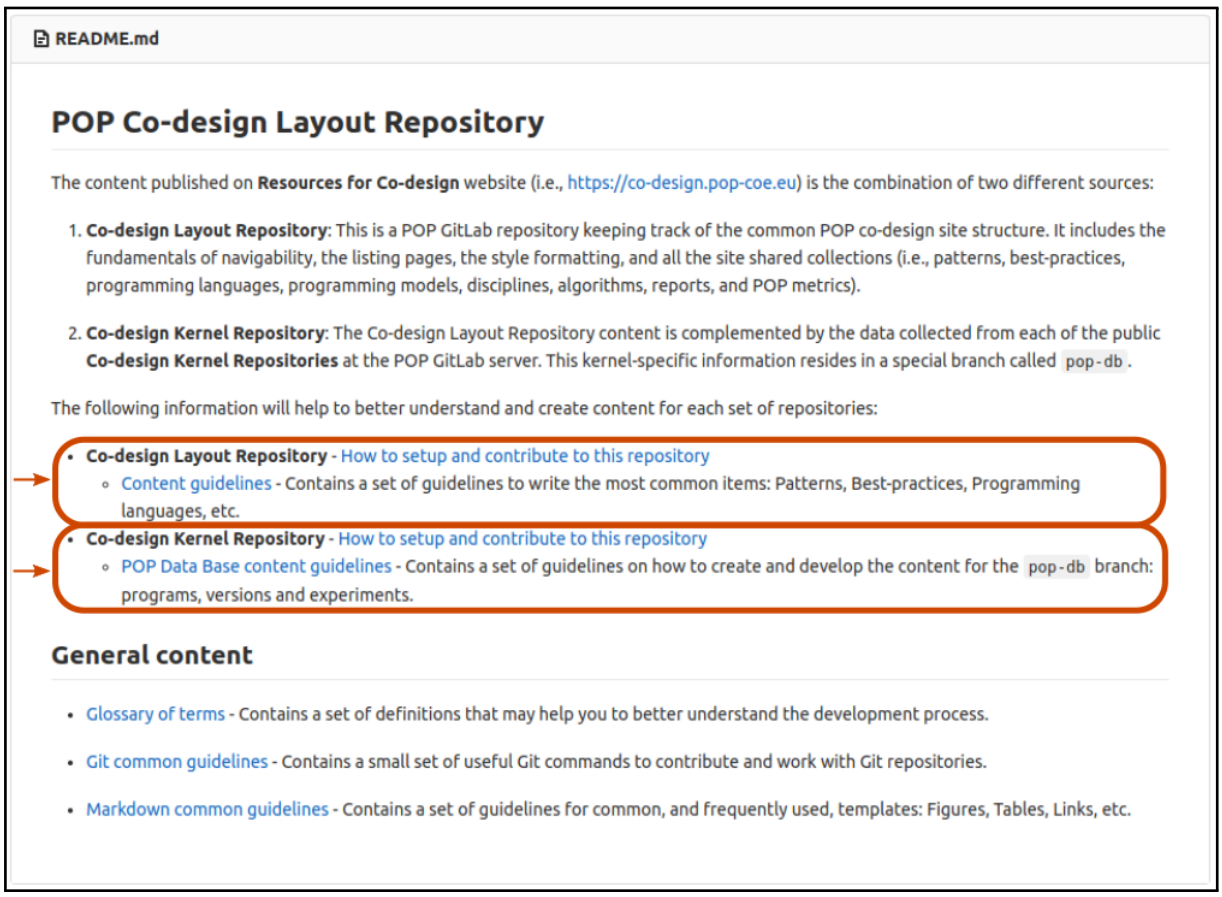
Another issue that has been improved is the GitLab CI/CD[2] pipeline. We have optimized some of the stages to reduce the overall execution time that takes to parse all the information and publish the Resources for Co-design website. We were able to reduce the execution time by half (from around 10 minutes to 5 mintues).

In the context of the new Reports collection, we developed a script that helps to gather information of all the reports that are going to be published in the website. This was required because there is a large set of reports to add and it would be a time consuming task to do all of this work manually.

Last but not least, the following set of improvements were completed in order to better visualize and relate content in the website:

- First of all, we have standardized the way information is displayed. To highlight this feature, we show in Figure 5 the Pattern *Load imbalance due to computational complexity (unknown a priori)*. Here, it is displayed in the begining of the content a grey box containing information regarding symptoms that can be used to identify this pattern (prologue). Also, at the end we can visualize another grey box with useful information regarding possible best-practices and also programs that implement this pattern (epilogue). This prologue

[2]CI/CD, as *Continuous Integration* and *Continuous Deployment*.


README.md

POP Co-design Layout Repository

The content published on **Resources for Co-design** website (i.e., https://co-design.pop-coe.eu) is the combination of two different sources:

1. **Co-design Layout Repository**: This is a POP GitLab repository keeping track of the common POP co-design site structure. It includes the fundamentals of navigability, the listing pages, the style formatting, and all the site shared collections (i.e., patterns, best-practices, programming languages, programming models, disciplines, algorithms, reports, and POP metrics).
2. **Co-design Kernel Repository**: The Co-design Layout Repository content is complemented by the data collected from each of the public **Co-design Kernel Repositories** at the POP GitLab server. This kernel-specific information resides in a special branch called `pop-db`.

The following information will help to better understand and create content for each set of repositories:

- **Co-design Layout Repository** - How to setup and contribute to this repository
  - Content guidelines - Contains a set of guidelines to write the most common items: Patterns, Best-practices, Programming languages, etc.
- **Co-design Kernel Repository** - How to setup and contribute to this repository
  - POP Data Base content guidelines - Contains a set of guidelines on how to create and develop the content for the `pop-db` branch: programs, versions and experiments.

General content

- Glossary of terms - Contains a set of definitions that may help you to better understand the development process.
- Git common guidelines - Contains a small set of useful Git commands to contribute and work with Git repositories.
- Markdown common guidelines - Contains a set of guidelines for common, and frequently used, templates: Figures, Tables, Links, etc.

Figure 4: Reources for Co-design repository guidelines.

	Met	Pat	Bep	Prog	Lan	Mod	Disc	Alg	Rep
Metrics (Met)		X		X					
Patterns (Pat)	X		X	X					X
Best-practices (Bep)		X		X					X
Programs (Pro)		X	X		X	X	X	X	X
Languages (Lan)				X					X
Models (Mod)				X					X
Disciplines (Dis)				X				X	
Algorithms (Alg)				X			X		
Reports (Rep)					X	X			

Table 2: New displayed relationships among *Resources for co-design* collections. For example, in Patterns we can see associated Metrics, Best-practices, Programs and Reports.

and epilogue pattern will apear in each collection that requires contextualization.

- Another area of improvement was relashionships between collection's items. By using internal Jekyll[3] variables we were able to create a network of relashionships and display the related content accordingly. Table 2 summarizes such displayed relationships.

[3] As introduced in D7.2 Co-design repository structure, *Jekyll* is a website generator able to transform a set of files written in Markdown language into a set of HTML pages properly linked to each other.

- Finally, we have performed a verification throughout the website and fixed a set of bugs:
  - Linking content: We added the Jekyll feature *base-url* to link content in the website. This allowed us to fix problems with broken links. As an example, we have an internal page called C-report that was not displaying images and was improved with the introduction of this fix (this is a internal page that is used to group information regarding patterns and best-practices).
  - Collections not displaying the entire content or displaying the wrong content: Here we had to fix specific if conditions in the collections layout files to display the information correctly. For example, *Resources for Co-design* website was displaying programming languages that were not associated with a given program. Also, the script that gathers all programs was not retrieving information about all the public kernels (problem with pagination feature of the GitLab API).
  - To correctly show the version's repository branch (for each program), we had to modify the version's layout source code to point each version to the respective repository branch instead of always pointing to the base branch. Also, we fixed an issue related with duplication of experiments (experiments were being shown in versions that were not related with those experiments).

3 New collections and associated contents

3.1 Collection of reports

A new collection called **Reports** was created to publically make available all the reports produced during the POP project where we have explicit permission from our customers. To this end, we have created a new repository in the POP GitLab to place all reports that are suitable for publication. In this repository we combine both POP1 and POP2 performance audits, assessments, and proof-of-concept reports. In order to filter the reports that can be published, we are following the information presented by *WP3 Costumer Advocay*, in particular, the tables available in the POP2 Wiki that indicate the possibility of publishing these documents.

In Figure 6 we can see how reports are being displayed. They are being displayed in a table format with five relevant fields: Report name, Related Project, Type of report, Programming Languages and Models.

3.2 Collection of metrics

This collection was already introduced in the previous version of the Website, so this is not a new collection. Although, we have redefined all the content in order to keep up to date with the latest developments in the specification of the POP metrics. Therefore, we have introduced two new POP metric classes: Multiplicative Hybrid metrics and Additive Hybrid metrics (Figure 7 shows a snapshot of the Multiplicative hybrid metrics and how is being displayed).

Furthermore, we have modified the description of each metric and also display which patterns and programs express that specific metric (Figure 8).

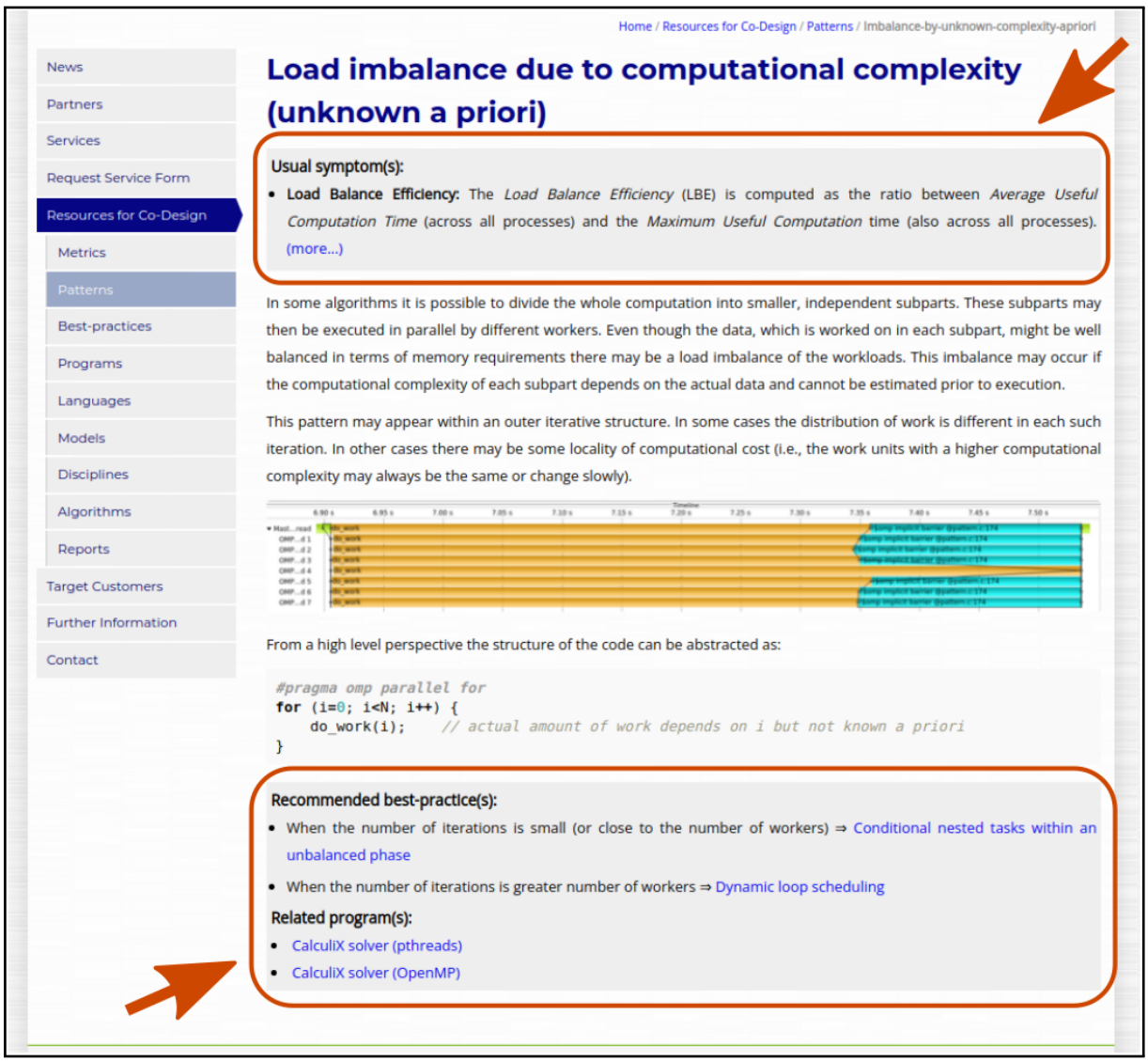


Figure 5: Reources for Co-design website: prologue and epilogue content.

4 Repository contents

4.1 New kernels

1. *For loops auto-vectorization* covers the essentials of optimizing the utilization of vector instructions to compute a given data-parallel workload. In this context, the compiler can provide valuable information about the limitations of the program and also hints on how to modify the code to fully optimize it.

2. The kernel *Python loops* is a synthetic program based on a real world HPC Python script that reproduces an inefficient way to write loop-compute algorithms in Python.

3. The *False communication-computation overlap* kernel is a synthetic program which reproduces a communication/computation pattern between several MPI processes.

4. *Alya* is a simulation code for high performance computational mechanics. Alya solves cou-

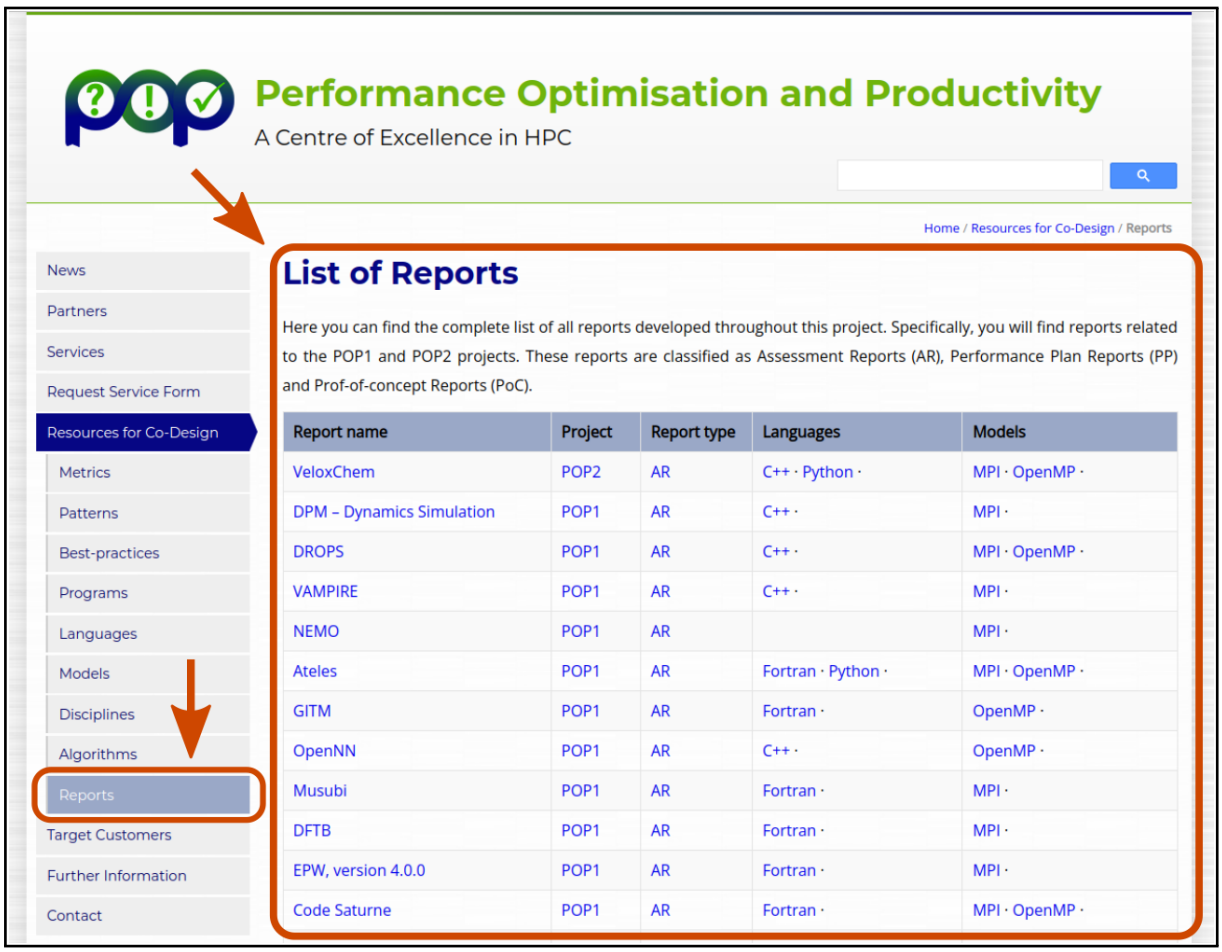


Figure 6: Resources for Co-design website: new Reports collection.

pled multiphysics problems using high performance computing techniques for distributed and shared memory supercomputers, together with vectorization and optimization at the node level. This kernel corresponds to the algebraic system assembly of a finite element code (FE) for solving partial differential equations (PDE's). The matrix assembly consists of a loop over the elements to compute element matrices and right-hand sides and their assemblies in the local system.

5. *SIFEL (SImple Finite ELements)* is an open source computer finite element (FE) code that has been developing since 2001 at the Department of Mechanics of the Faculty of Civil Engineering of the Czech Technical University in Prague. It is C/C++ code parallelized with MPI. This is the SIFEL kernel, which implement the LDL matrix factorization from a sparse matrix in symmetric skyline format and the computation of the Schur complement matrix.

6. The *Juelich KKR code family (JuKKR)* is a collection of codes developed at Forschungszentrum Juelich implementing the Korringa-Kohn-Rostoker (KKR) Green's function method to perform density functional theory calculations. Since the KKR method is based on a multiple scattering formalism it allows for highly accurate all-electron calculations. One of the main codes in the family is the KKRhost code which is used for electronic structure calculations of periodic systems. The code is written in Fortran and parallelized using a hybrid MPI + OpenMP approach. MPI ranks are logically arranged in a two-dimensional

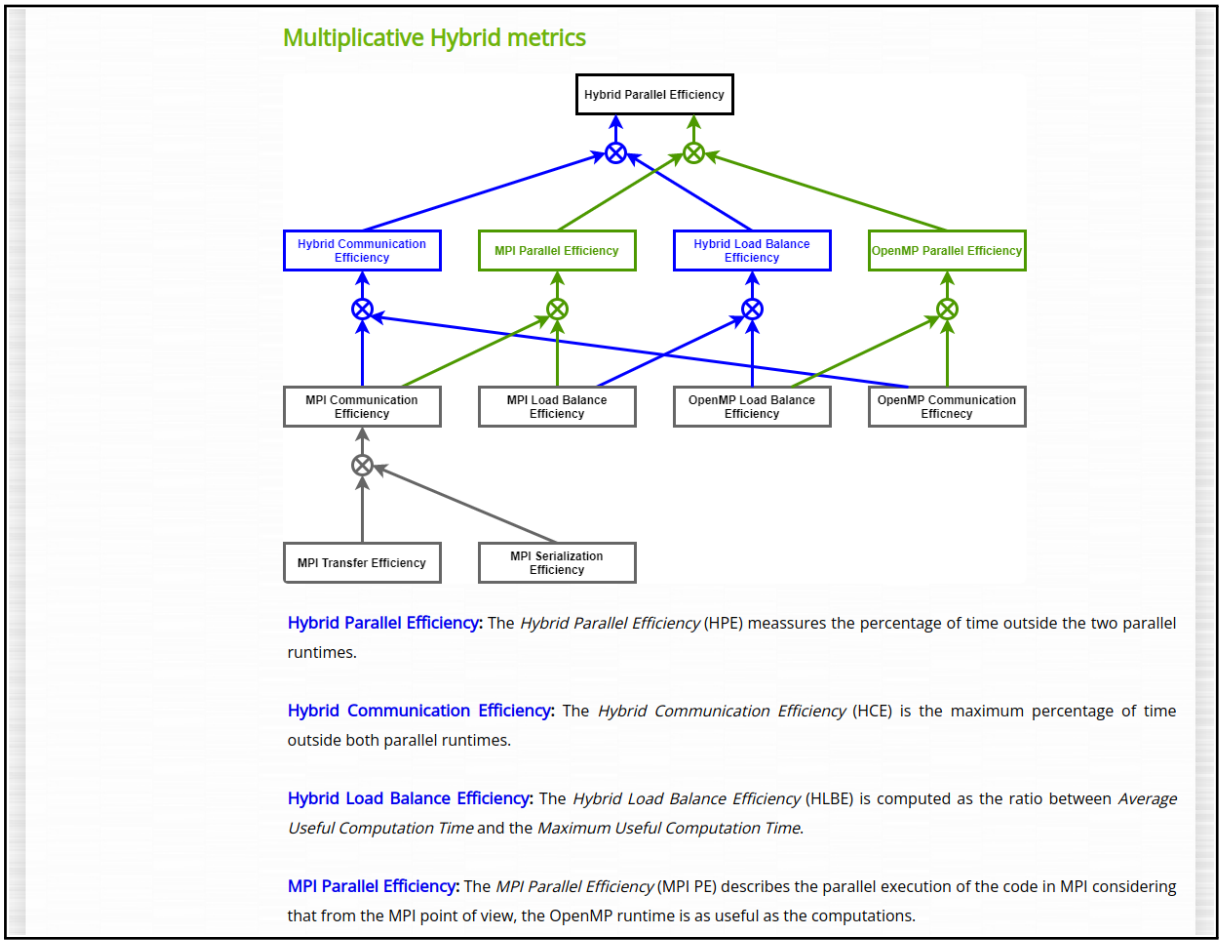


Figure 7: Resources for Co-design website: Multiplicative Hybrid metrics.

grid. In one dimension ranks are distributed among atoms and in the other dimension among energy points of the quantum system under consideration.

7. The *Communication and computation trade-off* kernel is a simple molecular dynamics code, with a relatively large amount of communication relative to the computation. The code solves Newton's second law of motion for every atom: $F = ma$. The domain decomposition is such that each MPI process has N/p atoms, where p is the number of processes (assuming this is perfectly divisible).

8. The *CPU to GPU* kernel implements the solution of the 3D diffusion equation. There are currently the following implementations: cpu_diffusion (CPU, single core), cpu_opencl_-diffusion (CPU, loops are executed in parallel using OpenMP) and opecl_diffusion (GPU, the iterations are computed on the GPU, the CPU launches kernels and manages the date transfer between MPI ranks).

9. The *BLAS tuning* kernel is a simple synthetic benchmark code that is used to show the pattern and identify the best-practice to improve the performance of multiplication of matrix-blocks. This is a code for timing parallel matrix multiplication, where all processes have access to the data in A and B, and the parallelisation is achieved by splitting B over the columns, i.e. $C_b = A * B_b$. Moreover, C_b and B_b are matrix blocks generated by splitting over the columns of matrix B.

Figure 8: Resources for Co-design website: IPC Scaling metric with related content.

10. The *Samoa* kernel stands for Space-Filling Curves and Adaptive Meshes for Oceanic And Other Applications and has been developed at TU Munich. Samoa is a PDE framework that can e.g., simulate wave propagations during tsunamis using adaptive meshes and space filling curves. It can either simulate built-in sample scenarios like radial_dam_break (perfectly balanced) and oscillating lake (imbalanced), or be used with datasets from real tsunamis by providing corresponding bathymetric and displacement data files.

11. The *OpenMP collapse* is a kernel which has been extracted from *jCFD_Genesis* application. For a nested OpenMP parallel loop, when the number of threads is larger with respect to the number of iterations and the iterations have similar length the parallel loop will display computational imbalance. This imbalance is due to a sub-optimal distribution of work among threads leading to idle cores. The inefficiency of the nested loop increases with the number of used threads.

12. The *GPU kernel* performs a naive matrix multiplication on a GPU. It is representative for many vector and matrix operations performed in computational kernels. However, in the original version the data transfer is not aligned to the word size, resulting in bad efficiency. The revised version tells the compiler that there is no pointer aliasing so it can generate code with coalesced loads and stores. This significantly increases performance, by nearly three times on a NVIDIA A100.

13. The *Access pattern bench* is a set of programs to simulate various memory access patterns that can arise in applications and have an impact on performance and efficiency. This kernel does not use any parallel programming model like MPI or OpenMP but is mainly focusing on serial access patterns investigating cache and vectorization behavior. Nevertheless, these access patterns can also appear for single processes/threads in parallel workloads.

14. The *imbalance Parallel File I/O* is a version of the existent *File I/O* kernel. This POP WP7 CoDesign code implements parallel file I/O using MPI file I/O with collective file access (all processors must participate in file access) for unstructured data where processes have different amount of data elements. Ordering of the data elements in the file is not important. The file format comes with a header including information for reading and distributing the data as they where at the moment of writing, mimicing, e.g. needs for checkpoint files.

4.2 New patterns and best-practices

1. *Inefficient user implementation of well-known math problem.* In the early stages of scientific code development, developers (possibly scientists) tend to create a naive and easy-to-read implementation of a required algorithm. Such an implementation have a great chance to be inefficient in some performance aspect.

   (a) *Leverage optimized math libraries.* The fundamentals of this best-practice lie in the rule: Do not reinvent the wheel. If a developer recognizes a well-known math routine that possibly causes a bottleneck in the program, it is recommended to do a short research on available libraries that implement the recognized routine and fit the program.

2. *Lack of iterations on an OpenMP parallel loop.* In some codes, the problem space is divided into a multi-dimensional grid on which the computations are performed. This pattern typically results in codes such as *jCFD_ Genesis*, with multiple nested loops iterating over the grid.

   (a) *Collapsing OpenMP parallel loops.* In some codes, the problem space is divided into a multi-dimensional grid on which the computations are performed. The idea behind this best-practice is to collapse the iteration space in order to being able to create more parallelism, keeping enough granularity.

3. *Inefficient Python loops.* With Python it is very easy to unconsciously produce extremely inefficient code as it is an interpreted language. One need to put special attention on the data types and sentences used in order to mitigate the interpreter's overhead since generic Python objects are several orders of magnitude slower than other alternatives. Therefore, after the prototyping phases when developing Python software, users need to identify the heaviest compute functions and apply to them the most suitable optimization.

   (a) *Usage of Numba and Numpy to improve Python's serial efficiency.* When Python applications have heavy computing functions using generic data types it is possible to drastically increase sequential performance by using Numba or Numpy. Both packages can be used at the same time or separately depending on code's needs.

4. *Spatial locality poor performance.* To achieve good performance in scientific and industrial software it is essential to review how data structures are designed inside the software and stored/loaded by the underlying programming language or machine. Although such design decisions might be beneficial in terms of readability and maintenance, it could significantly degrade performance if the way data is accessed and processed in the application does not match the data layout. There are multiple examples for a mismatch between data layout and access that might harm spatial locality of the access pattern including but not limited to the following: Arrays of Structure (AoS) or Memory layout of multi-dimensional arrays depending on the programming language.

   (a) *Spatial locality performance improvement.* This best practice recommends to align data layout and data access pattern to efficiently use available resources and take advantage of e.g., the caching behavior and vectorization units of the underlying architecture. As this best practice highly depends on the code and data structures at hand we will review the example of Array of Sturctures (AoS) that has been introduced in the pattern. In order to mitigate problems arising from the strided access pattern one option would be to adapt the data structure to better fit the access pattern like illustrated in the following code snippet.

5. *Very fine-grained tasks/chunks.* The essential purpose of parallel programming is to reduce the runtime by dividing time-demanding computations among available resources. The granularity is an important aspect that is often overlooked. The amount of work assigned to a computational core/thread has to have a reasonable size so that the overhead caused by thread/core control does not degrade the performance. When the tasks or chunks are too small, this overhead can cost the same or even more time than the computation itself. In such a case, running a program in parallel may not bring any benefit or even can consume more time and resources than running as a single thread application.

   (a) *Chunk/task grain-size trade-off (parallelism/overhead).* A trade-off between parallelism and overhead in terms of the POP metrics corresponds to balancing the Load Balance and Communication efficiency. In general, we aim to create enough chunks (parallelism) in order to utilize all available computational resources (threads). At the same time, we need to keep the related overhead low. Less overhead is reflected by higher Communication efficiency. On the other hand, scheduling of threads at the runtime is often needed in order to balance the workload effectively. This leads to better Load Balance at the cost of the lower Communication Efficiency.

6. *Low computational performance calling BLAS routines (gemm).* In many scientific applications, parallel matrix-matrix multiplications represent the computational bottleneck. In the case where each process has access to $\mathbf{A}$ and $\mathbf{B}$, a strategy to parallelize the work is to divide $\mathbf{B}$ into blocks so as to partition the computation over all the processes, and use the BLAS dgemm routine on the individual blocks. This partition may lead into a low global efficiency and a non-optimal performance.

   (a) *Tuning BLAS routines invocation (gemm).* It is best practice to write a simple code in order to test alternative partitioning of the matrix multiplication. In the case described above, where $\mathbf{B}$ is split over the columns, and $\mathbf{A}$ is not split, the size of the matrix blocks is determined by the number of processes. However, if we split $\mathbf{A}$ over the rows, whilst still splitting $\mathbf{B}$ over the columns, we have a choice of the size of the matrix blocks for $\mathbf{A}$ and of $\mathbf{B}$ in smaller blocks.

7. *High weighted communication in between ranks.* This pattern can be observed in parallel algorithms where a large fraction of the runtime is spent calculating data on individual processes and then communicating that data between processes. An example of such a parallel algorithm is molecular dynamics, where N particles interact with each other.

   (a) *Replicating computation to avoid communication.* The best-practice presented here can be used in parallel algorithms where a large fraction of the runtime is spent calculating data on individual processes and then communicating that data between processes. For such applications, the performance can be improved by replicating computation across processes to avoid communication.

8. *Writing unstructured data to linear file formats.* Many applications cope with unstructured data, which result in unbalanced data distribution over processes. A simple example for this are particle simulations where particles are moved around between processes over time. So, when the simulation shall write the global state at the end - or in-between, e.g., for a checkpoint - each process will have a different number of particles. This makes it hard to write data efficiently to a file in a contiguous way. The same pattern can also be found, e.g., in applications using unstructured meshes with different number of elements per process.

   (a) *MPI-I/O with pre-computed offsets for each processes.* Writing unstructured data with differing amounts of data per process to a contiguous file is challenging. If the ordering of the data in the file is not of importance, one approach using MPI I/O functionalities to achieve good performance is to pre-compute per process offsets into the file so that each process can then write his local data starting from this position without interfering with the other processes.

9. *Sequence of fine grain parallel loops.* This pattern applies to parallel programming models based on a shared memory environment and the fork-join execution model (e.g., OpenMP). The execution of this kind of applications is initialy sequential (i.e., only one thread starts the execution of the whole program), and just when arriving at the region of the code containing potential parallelism, a new parallel region is created and multiple threads will start the execution of the associated code. Parallel execution usually will distribute the code among participating threads by means of work-sharing directives (e.g., a loop directive will distribute the loop iteration space among all threads participating in that region).

   (a) *Collapse consecutive parallel loops.* The main idea behind collapsing parallel regions is to reduce the overhead of the fork-join phases. This technique consists on substituting a sequence of parallel work-sharing regions with a single parallel region and multiple inner work-sharing construs. It also include the possibility of removing work-sharing barriers.

   (b) *Upper level parallelisation.* When thinking about parallelizing an application, one should always try to apply parallelization on an upper level of the call tree hierarchy. The higher the level of parallelization the higher the degree of parallelism that can be exploited by the application. This best-practice shows a scenario where moving the parallelization to an upper level improves the performance significantly.

10. *Suitable programs to run on GPU.* This pattern outlines criteria that can be used to identify programs that can be implemented on GPUs using a programming model outlined

in the *Porting code to GPU (iterative kernel execution)* best-practice. This criteria can be identified as: code structure, algorithm and size of data set. Code structure: The computational problem allows partitioning of the problem using the MPI programming model. Algorithm: The algorithm must have a high degree of inherent parallelism. In order to write GPU kernels that exploit the parallelism, a deep understanding of the algorithm is necessary. Size of data set: The amount of data that is used in the computation must not exceed the amount of memory available on a single GPU.

(a) *Porting code to GPU (iterative kernel execution).* The described programming pattern is very similar to the traditional pattern used for CPU codes in the way that multiple MPI ranks run on multiple compute nodes. The difference is that the GPU is not regarded as an accelerator for certain parts of the code but as the main computing unit. Each MPI rank uses one GPU at first in order to reduce the complexity of the code and to avoid load imbalance within a single MPI rank. Each MPI rank can use multiple threads for performing computations during the execution of GPU kernels and during the data exchange among MPI ranks when the GPUs are idle.

11. *Indirect reductions on large data structure.* A pattern that appears on many codes computing on a grid of topologically connected nodes is the reduction of values on the nodes with possible multiple contributions from neighboring nodes. This pattern typically results in codes with reduction operations through indirection on a large array as depicted in the following skeleton.

    (a) *Using multidependencies.* The idea behind the use of multidependencies is to split the iteration space into tasks, precompute which of those tasks have "incompatibility" (modify at least one shared node) and use the multi dependences feature in OpenMP to achieve at runtime a scheduling effect comparable to coloring but at coarse granularity (tasks) and dynamic. Precomputing the incompatibilities is an additional code to be written and may represent an overhead. In any case, if the topology is relatively invariant this can be amortized over many iterations.

12. *GPU branch diverging.* The execution of a thread block is divided into warps with a constant number of threads per warp. When threads in the same warp follow different paths of control flow, these threads diverge in their execution such that their execution is serialized. Such a branch diverging scenario can be avoided by aligning the branch granularity to warps.

    (a) *Align branch granularity to warps.* In an optimal case all threads in a warp follow the same code path. There are several software-based optimizations available a developer can apply if manual alignment of branch granularity to warp size is not possible. Two of the techniques that can be applied are iteration delaying and branch distribution. Iteration delaying targets a divergent branch enclosed by a loop within a kernel. It improves performance by executing loop iterations that take the same branch direction and delaying those that take the other direction until later iterations. Branch distribution reduces the length of divergent code by factoring out structurally similar code from the branch paths.

13. *GPU uncoalesced memory transfer.* On CUDA compute architectures, memory transfers between global and shared memory are performed in a word size of 128 bit. Given that smaller transfers are padded to that size, copying smaller values between global and shared

memory reduces the overall transfer rate. As a consequence, it is more efficient to coalesce narrow memory references into wide ones. Even though compilers occasionally perform memory coalescing automatically, developers are usually responsible for coalesced memory transfers by manually aligning loads and stores.

(a) *Align loads and stores.* The most efficient solution to an uncoalesced memory transfer issue would be to change the code so all the data is a 128 bit word is used consecutively. However, such algorithmic changes are not always possible. But the developer can help the compiler to generate more efficient code by assuring that there is no pointer aliasing, i.e. that two pointers don't reference the same chunk of memory. This gives the compiler the freedom to apply various optimizations. On supported CUDA devises it also allows the use of the GPU read-only data cache, potentially accelerating data movement to the kernel.

In addtion, we are proposing alternatives best-practices for existing patterns (already included in the previous milestones):

1. *Taskifying communications.* This is one of the several alternatives to parallelize the packing and unpacking operations when using a Message Passing Interface. The main idea consist on encapsulating send and receives calls within an unstructured task, giving the opportunity to overlap those communication with computation or with other communication. This best-practice applies to the existent *Sequential communications in hybrid programming* pattern.

2. *Overlap computation/communication with TAMPI.* The main idea behind this best-practice is to link with the intermediate *Task Aware Message Passing Interface* (TAMPI) library in order to leverage in inherent features to overlap computation and communication and increase the amount of parallelism avoiding to block on non-asynchronous MPI services. This best-practice applies to the existent *Sequential communications in hybrid programming* pattern; once we have applied the *Taskifying communications* best-practice.

3. *Overdecomposition using OpenMP tasking.* There are multiple approaches to tackle load imbalances in an application. This best practice shows how over-decomposition with e.g., OpenMP tasking can help to reduce load imbalances as an alternative to classic loop-based solutions. This best-practice applies to the existent *Load imbalance due to computational complexity unknown a priori* pattern.

4. *Task migration among processes.* This best practice presents an approach using over-decomposition with tasks and task migration to mitigate the load imbalances between processes. This best-practice applies to the existent *Problems in dynamic load balancing* pattern, and also to *Load imbalance due to computational complexity (unknown a priori).*

And finally, we have refactored one of the existent patterns:

1. *Sequential loops.* Most of the program execution time is spent on cycles. One complication with parallelizing the sequential loop is that the body of the loop may also depend on the previous invocations of its self. This is a refactor of *Manual loop unroling*, to better capture the actual problem included in this pattern.

4.3 New programming models, disciplines and algorithms

We have added **one** new programming model:

1. *oneAPI.* oneAPI offers an open, unified programming model to simplify the development and deployment of data-centric workloads across CPUs, GPUs, FPGAs and other types of hardware architectures.

We have added **one** new discipline:

1. *Computer graphics rendering.* Rendering is used to produce a photorealistic or non-photorealistic image from a 3D model. High-level structures representing a scene are transferred into pixels accompanied by light and shading calculations of the scene's objects. The light propagation computations are complex, expensive, and this complexity grows with the size of a scene. High performance clusters provide strong resources including processors and accelerators to speed up the process and offer a lot of memory for a large data set manipulation. Rendering is used for 3D visualization (e.g., architecture, medicine, mechanical engineering), the development of animated movies, computer games, etc.

We have added **one** new Algorithm:

1. *Finite difference methods.* Finite different methods (FDM) is a general numerical method for solving ordinary differential equations (ODE) or partial differential equations (PDE) raising from physical problems in engineering analysis and design. The original differential equation is solved on equally spaced grid points where finite difference formulas are used to approximate the solution. This way, we can transform a differential equation into a system of algebraic equations to solve.

5 Conclusions

In this section, we will summarize all the contributions described in the previous sections and we will also present a snapshot of the current state of the *Resources for Co-design* website. Such information will be crossed with the expected contributions as described in Section 1.1. Finally, we will evaluate the overall success of the workpackage with respect to the proposed milestones.

Table 3 shows the current state of the *Resources for co-design* website. The table is organized per partner and kind of contribution. For each entry, we define the expected *Target*, the current state for the on-going milestone (i.e., *MS8 Status*), the contribution reported in the previous milestone (i.e., *MS3*), and the achieved value (considering *Review* and *Closed* as *Fulfil*[4]). The bottom part of the table aggregates these values in order to evaluate the overall success.

The aggregated *Fulfil* value regarding the number of kernels (i.e., 24) indicates the fulfillment of *MS8 - Final Methodology Milestone*, having surpassed the 20 micro-benchmarks as defined in the project proposal [1].

With respect to the aggregated value of the *Target* kernels (i.e., 24), a total of 14 (58%) are already closed, and 10 (42%) are already in review (what actually means that all of them have been already published in the site). It is important to remark that one of these kernels is not generating a new entry on the site, but extending one existing kernel with new versions.

[4]When a kernel reaches the *Review* state, it means it is already published in the site but we are still carrying out a *peer review* (i.e., quality check pass). When any other item reaches this state it means there is a formal definition of the element but it is being reviewed/discussed on the technical calls.

PARTNER	Item	Target	*MS8 Status*				MS3	Fulfil
			Pending	Progress	Review	Closed		
BSC	Kernels	5	0	0	1	2	2	5
	Patterns	14	0	1	1	0	5	16
	Best-practices		0	4	0	0	5	
	Classification		0	0	0	1	4	
HLRS	Kernels	5	0	0	2	1	2	5
	Patterns	14	0	1	1	1	2	14
	Best-practices		0	1	1	1	3	
	Classification		0	0	0	0	5	
IT4I	Kernels	2	0	0	1	0	1	2
	Patterns	8	0	0	0	2	1	17
	Best-practices		0	0	0	2	0	
	Classification		0	0	0	2	10	
JSC	Kernels	2	0	0	1	0	1	2
	Patterns	8	0	2	0	0	1	4
	Best-practices		0	2	0	0	1	
	Classification		0	0	0	0	2	
NAG	Kernels	5	0	0	3	0	2	5
	Patterns	14	0	0	0	3	2	14
	Best-practices		0	0	0	3	4	
	Classification		0	0	0	0	2	
RWTH	Kernels	5	0	0	2	1	2	5
	Patterns	14	0	0	0	1	2	14
	Best-practices		0	1	0	3	4	
	Classification		0	0	0	0	4	
ALL	$\sum$ Kernels	24	0	0	10	4	10	24
	$\sum$ Patterns	72	0	4	2	7	13	79
	$\sum$ Best-practices		0	8	1	9	17	
	$\sum$ Classification		0	0	0	3	27	

Table 3: Current state of the *Resources for co-design* site at the delivery of this document.

The rest of the contributions (Patterns, Best-practices and Classification items) have been grouped into a single target. The main reason is to allow easily to shift some effort from one category to another and also give some flexibility to partner's proposals. With respect to the aggregated *Target* pages (i.e., 72), a total of 76 (105%) are already closed, 3 (4%) are in review, and 12 (16%) are currently in progress.

Most of the partners have achieved their individual KPIs (i.e., *Target* vs. *Fulfil*). The most remarkable positive deviation appears in the IT4I entry, that contributes with 9 extra pages with respect to the original plan. Such contribution was done at the beginning of the project when including the programming model descriptions into the website. In the other hand, JSC still misses 2 patterns and 2 best-practices, but all these issues are already in *progress* and there is a firm comittment to make them *ready* for discussion in the short-term (during the project extension).

Regardless of such deviations, the actual global contribution has reached 100% in the case of the kernels, and above 100% in all the other cases. During the extension of the project (i.e.,

M37-M42) we plan to finalize all the reported items in this document that were not closed at M36 yet.

5.1 Contribution complexities

One of the main issues when accounting for the actual effort devoted to the workpackage is to take into account the differences (in terms of complexity) among kernels, patterns/best-practices and classification items.

The first observation would be that working on a kernel involves creating a repository, developing multiple versions of the code, documenting such entries, and also reporting some results (e.g., a performance comparison among versions); so it is one of the most time-consuming activities within the workpackage.

A second observation would be that describing either a Pattern or a Best-practice also involves developing the specific ideas behind the problem or the solution, demonstrating potential penalties or improvements, linking the contents with actual applications, etc. So, there is also a greater complexity developing these entries than describing any other classification item. In fact, the average discussion period for patterns/best-practice is around 30 days (i.e., since the ready-discussion label is assigned until the execute-publish label finally appears), while on the other side, the corresponding discussion period for a classification item is around 12 days.

So, the first conclusion we can infer about contribution complexities is:

Kernels >= Patterns/Best-practices >= Classification Items.

It could be difficult to set a ratio among the different kinds of contributions, but one approximation could be including the number of subitems within the kernel development. That is, to sum all the components arising from the creation of a kernel: kernel's description, versions, and experiments. This approach also links with another concern on kernel development. The cases in which the baseline code already exists but we want to create alternative versions implementing a different best-practice. Then, we can just count the subitems we are actually including in the existing kernel.

Table 4 shows the kernel decomposition in subitems and it also accumulates the total number of pages of the rest of the contributions per partner basis. The table summarizes the actual effort devoted to this workpackage that contributes with 56 different code versions to the *Resources for co-design* site.

Considering the differences between Patterns and Best-practices *versus* other classification items will imply looking for a ratio that could normalize the corresponding efforts. However, and for the purpose of this document, it is sufficient to remark such differences.

The total contribution of this work package with respect to micro-benchmarks could be summarized as: 23 kernel repositories, 57 kernel's versions and 43 experiment reports associated with them.

PARTNER	Program	Kernel				Kernel	Non-Kernel	Total
		D	V	E	T			
BSC	FFTXlib	1	2	2	5	24	16	40
	Communication Imbalance	1	2	2	5			
	False Comm-Comp Overlap	1	2	2	5			
	For Loops Auto-vectorization	1	2	2	5			
	Alya Assembly	1	2	1	4			
HLRS	DuMuX/DUNE	1	3	0	4	22	14	36
	Rank DLB	1	2	1	4			
	Python Loops	1	4	4	9			
	CPU to GPU	1	1	1	3			
	Imbalance Paralell I/O (v)	0	1	1	2			
IT4I	BEM4I miniApp	1	2	1	4	13	17	30
	SIFEL Kernel	1	3	5	9			
JSC	JuPedSim	1	3	0	4	9	4	13
	GPU Kernel	1	2	2	5			
NAG	Parallel File I/O	1	7	1	9	28	14	42
	OpenMP Critical	1	3	2	6			
	Comm-Comp Trade-off	1	1	2	4			
	OpenMP Collapse	1	2	1	4			
	BLAS Tuning	1	2	2	5			
RWTH	Calculix Solver	1	2	2	5	27	14	41
	Calculix I/O	1	2	2	5			
	JuKKR KLoop	1	2	2	5			
	Samoa	1	3	3	7			
	Access Pattern Bench	1	2	2	5			
	$\sum$ Items	23	57	43	123	123	79	202

Table 4: Total effort in number of pages (Kernel and non-Kernel). Kernel: Description (D), Versions (V), Experiments (E), and Total (T) page kernels.

References

[1] POP Consortium. Performance Optimisation and Productivity: A Centre of Excellence in Computing Applications, 2018.

[2] POP Consortium. D7.1 - Co-design repository structure, 2019.

[3] POP Consortium. D6.1 - First Report on Proof-of-Concept, 2020.

[4] POP Consortium. D7.2 - First co-design repository, 2020.