POP: Resources for Co-design

Software Co-Design Actions in European Flagship HPC Codes

Xavier Teruel, BSC

EU H2020 Centre of Excellence (CoE)

Grant Agreement No 824080

1 December 2018 – 30 November 2021

Extended to 31 May 2022

Outline

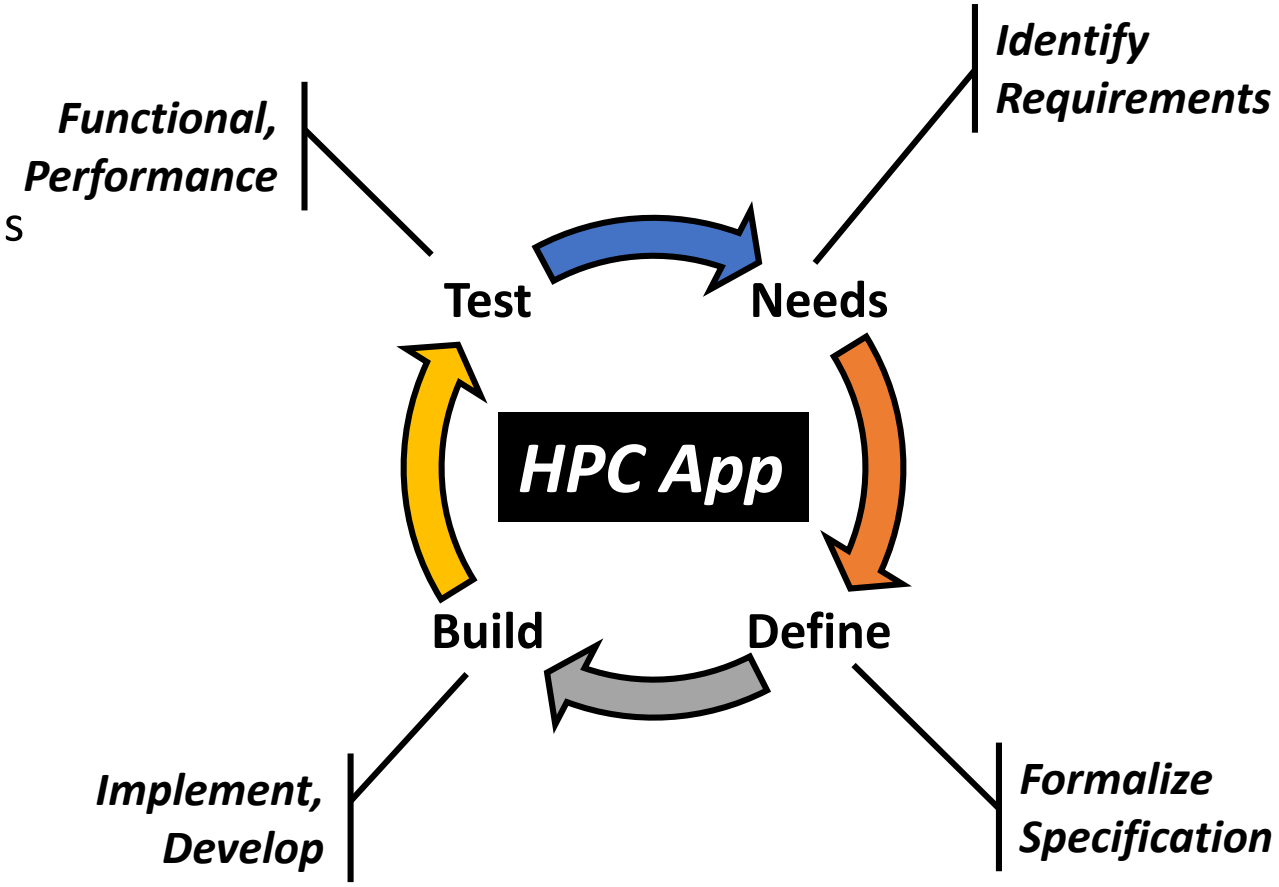
1. Our definition of “resources for co-design”
   - Incremental definition: design, co-design, and resources for co-design
   - Disclaimer: based on our vision of co-design

2. Brief description of co-design within the POP project
   - Activities work-flow
   - Figures on co-design activities

3. Demo: navigating the co-design site
   - Roadmap: metrics, patterns, best-practices, kernels, and experiments

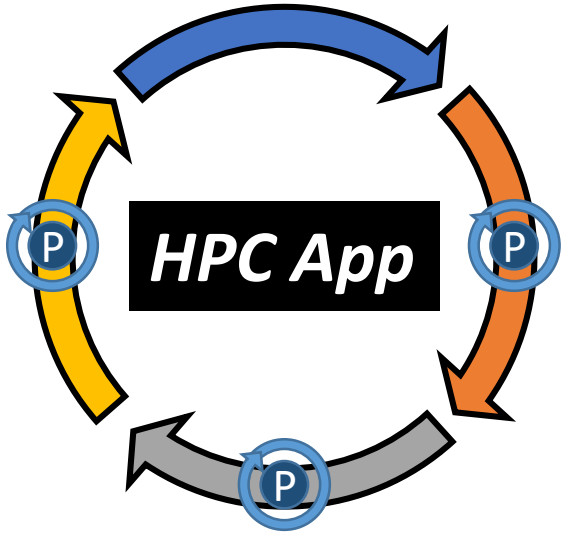
Design

- **Needs:** problem to solve? project requirements? limitations? goal?
- **Define:** analyze, synthesize, specify, ideate.
- **Build:** develop [*a prototype*], makes the ideas real.
- **Test:** does it work [*as desired*]? does it solve the need?



Design & patterns

- **Needs:** problem to solve? project requirements? limitations? goal?
- **Define:** analyze, synthesize, specify, ideate.
- **Build:** develop [*a prototype?*], makes the ideas real.
- **Test:** does it work [*as desired*]? does it solve the need?

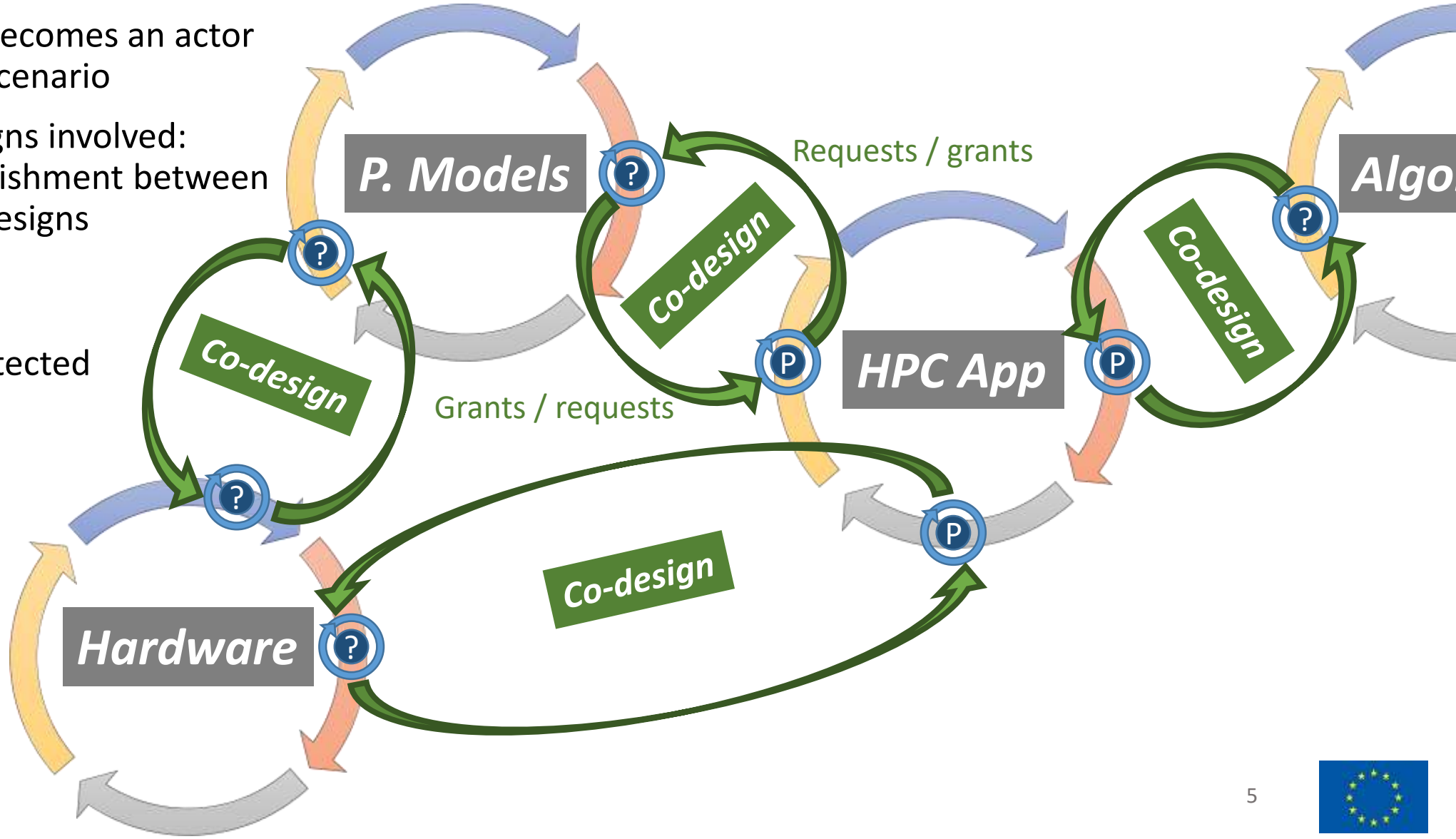


Design Pattern: ***Commonly occurring problems***

- Pattern → best-practice (*design patterns*)
- Provides an extra level of abstraction
- It allows re-usability among different designs

Co-design

- Our design becomes an actor in this new scenario
- Several designs involved: dialog establishment between 2 different designs
  - *Requests*
  - *Grants*
- Based on detected patterns



Resources

- **Patterns** (*problems*)

  Sequence of operations, memory accesses, communications and/or synchronizations that perform general algorithmic steps appearing in many different programs → They may result in potential performance degradations.

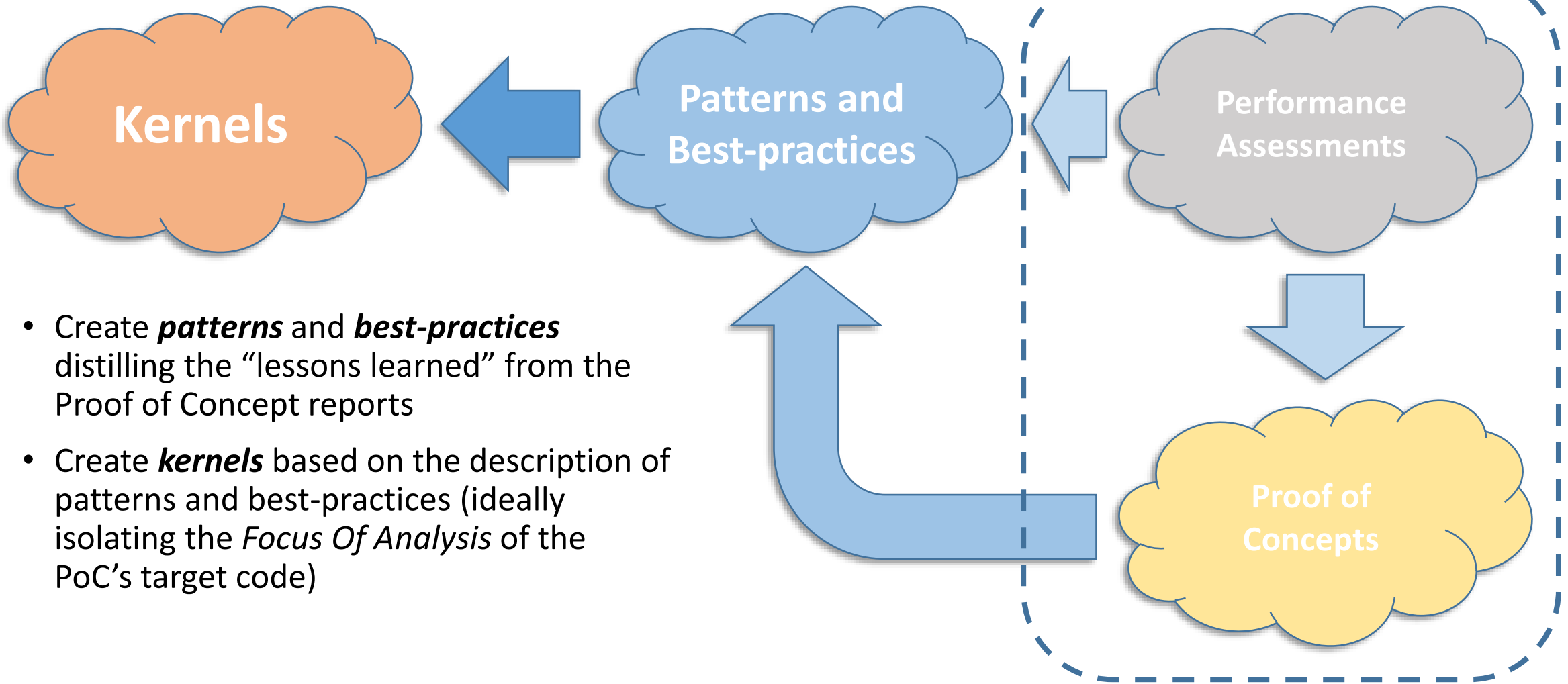
- **Best-practices** (*solutions*)

  Recommendations that will address the holistic space form of the application; refactoring (using or proposing) new features in the system software or hardware architecture.

- **Co-design kernels** (codes)

  Exemplify behaviours by means of **kernels** (with their corresponding metrics). Synthesize potential **problems** that may arise from HPC applications; propose **solutions** to such problems. Decorate descriptions with hw/sw **co-design** hints.

POP activity work-flow (co-design)



- Create ***patterns*** and ***best-practices*** distilling the "lessons learned" from the Proof of Concept reports
- Create ***kernels*** based on the description of patterns and best-practices (ideally isolating the *Focus Of Analysis* of the PoC's target code)

Figures of POP co-design

- A set of collections including **Resources** (patterns, best-practices, and kernels) and **Descriptive items** (metrics, languages, models, algorithms, disciplines, and reports).

Resources (abstract)	#
Patterns	20
Best-practices	26

Resources (programs)	#
Kernels's programs	23
Kernel's versions	57
Experiments	43

Resources (programs)	#
Prog. languages	4
Prog. models	9
Algorithms	5
Disciplines	11
Metrics	32
POP reports	116

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

Use case: parallelizing our application

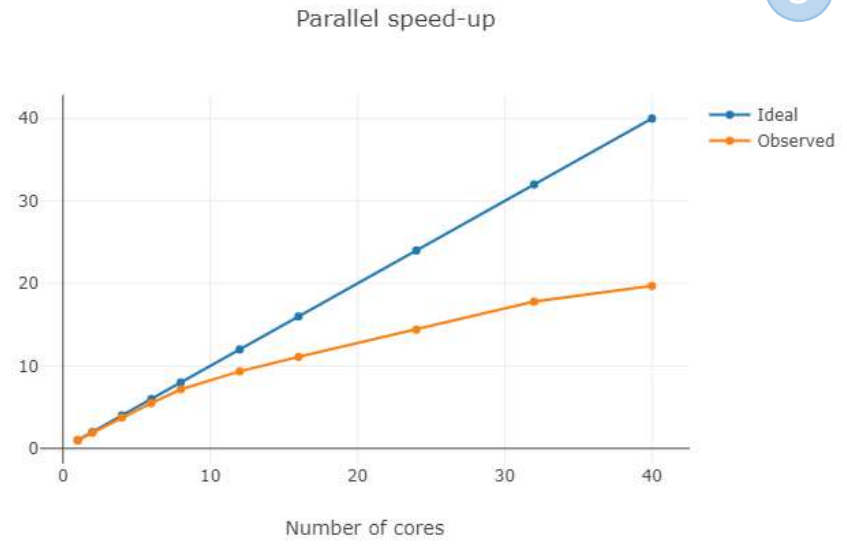
1

```
for(int i=0;i<SIZE;i++) {
   for(int j=0;j<SIZE;j++) if(i!=j) acc[i] = f1(...);
}

for(int i=0;i<SIZE;i++) pos[i] = f2(...);

for(int i=0;i<SIZE;i++) vel[i] = f3(...);
```

2

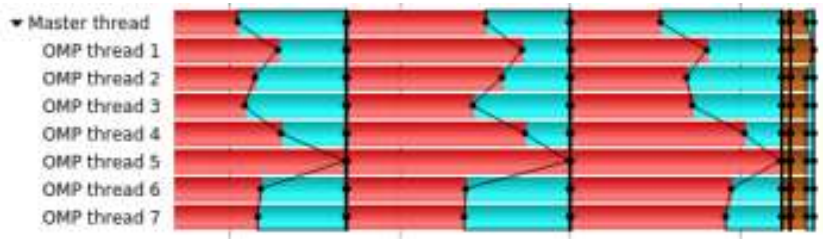
```
#pragma omp parallel for
for(int i=0;i<SIZE;i++) {
   for(int j=0;j<SIZE;j++) if(i!=j) acc[i] = f1(...);
}
#pragma omp parallel for
for(int i=0;i<SIZE;i++) pos[i] = f2(...);
#pragma omp parallel for
for(int i=0;i<SIZE;i++) vel[i] = f3(...);
```

3



4

	2	4	6	8
Global efficiency	80%	73%	61%	68%
Parallel efficiency	80%	73%	72%	72%
Load Balance	80%	73%	72%	72%
Communication efficiency	100%	100%	100%	100%
Computation scalability	100%	100%	85%	95%
IPC scalability	100%	103%	89%	101%
Instruction scalability	100%	98%	95%	94%



Co-design: landing page

Home / Resources for Co-Design

- News
- Partners
- Services
- Request Service Form
- Resources for Co-Design
  - Metrics
  - Patterns
  - Best-practices
  - Programs
  - Languages
  - Models
  - Disciplines
  - Algorithms

Description and main goals

The main objective of this site is to build a database with the performance metrics for different applications used within the POP Center of Excelence project. These metrics will allow characterizing the behavior of the corresponding application and could be queried by system designers (architecture, system software) in projects outside POP to demonstrate the potential of their proposed approaches and get a rough estimate of which codes their techniques will have an important impact on.

We also describe typical behavioural patterns (that may result in potential performance degradations) that we have identified in the analysis of applications using different algorithms in different disciplines. The objective then is to identify such patterns in generic terms, provide links to programs that expose them and links to the correspondant best-practice(s) that should address it. Some of these programs include additional versions that allow comparing the behaviour before/after applying a given best-practice.

Programs included in this site use different programming languages and leverage some of the most popular parallel programming models used in the HPC community.

Co-design: list of metrics

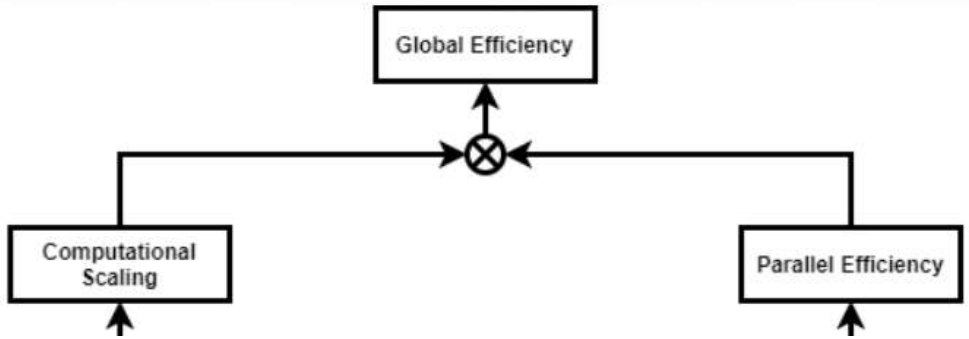
News
Partners
Services
Request Service Form
Resources for Co-Design
Metrics
Patterns
Best-practices
Programs
Languages
Models
Disciplines
Algorithms

List of Metrics

POP has defined a methodology for analysis of parallel codes to provide a quantitative way of measuring relative impact of the different factors inherent in parallelisation. The the methodology uses a hierarchy of metrics each one reflecting a common cause of inefficiency in parallel programs. These metrics then allow comparison of parallel performance (e.g., over a range of thread/process counts, across different machines, or at different stages of optimisation and tuning) to identify which characteristics of the code contribute to inefficiency.

The metrics are then calculated as efficiencies between 0 and 1, with higher numbers being better. In general, we regard efficiencies above 0.8 as acceptable, whereas lower values indicate performance issues that need to be explored in detail.

Standard metrics



Co-design: load balance

News
Partners
Services
Request Service Form
Resources for Co-Design
Metrics
Patterns
Best-practices
Programs
Languages
Models

Load Balance Efficiency

The *Load Balance Efficiency* (LBE) is computed as the ratio between *Average Useful Computation Time* (across all processes) and the *Maximum Useful Computation* time (also across all processes).

$$LBE = \frac{AverageUsefulComputationTime}{MaximumComputationTime}$$

In order to fully understand the formulas, you may also visit the glossary of the metrics terms.

Related programs: CalculiX solver · OMP Collapse ·

Related patterns: Code replication · Contention on shared resources · Frequency reduction by power governors · Load imbalance due to computational complexity (unknown a priori) · Inefficient user implementation of well-known math problem · Locality differences among threads · Lack of iterations on an OpenMP parallel loop · Operating System Noise ·

Co-design: load imbalance due to…

News
Partners
Services
Request Service Form
Resources for Co-Design
Metrics
Patterns
Best-practices
Programs
Languages
Models
Disciplines
Algorithms
Reports

Load imbalance due to computational complexity (unknown a priori)

Usual symptom(s):

- **Load Balance Efficiency:** The *Load Balance Efficiency* (LBE) is computed as the ratio between *Average Useful Computation Time* (across all processes) and the *Maximum Useful Computation* time (also across all processes). (more...)

In some algorithms it is possible to divide the whole computation into smaller, independent subparts. These subparts may then be executed in parallel by different workers. Even though the data, which is worked on in each subpart, might be well balanced in terms of memory requirements there may be a load imbalance of the workloads. This imbalance may occur if the computational complexity of each subpart depends on the actual data and cannot be estimated prior to execution.

This pattern may appear within an outer iterative structure. In some cases the distribution of work is different in each such iteration. In other cases there may be some locality of computational cost (i.e., the work units with a higher computational complexity may always be the same or change slowly).

Co-design: conditional nested tasks…

News
Partners
Services
Request Service Form
Resources for Co-Design
Metrics
Patterns
Best-practices
Programs
Languages
Models
Disciplines
Algorithms
Reports

Conditional nested tasks within an unbalanced phase

Pattern addressed: Load imbalance due to computational complexity (unknown a priori)
In some algorithms it is possible to divide the whole computation into smaller, independent subparts. These subparts may then be executed in parallel by different workers. Even though the data, which is worked on in each subpart, might be well balanced in terms of memory requirements there may be a load imbalance of the workloads. This imbalance may occur if the computational complexity of each subpart depends on the actual data and cannot be estimated prior to execution. (more...)

Required condition: When the number of iterations is small (or close to the number of workers)

In other cases N may be small (close or equal to the number of cores) and the granularity of `do_work` may be large. This happens for example in the Calculix application from which the kernel in the figure below shows an excerpt. In that case, the individual `do_work` computations correspond to the solution of independent systems of equations with iterative methods that may have different convergence properties for each of the systems.

Asuming sufficient granularity in `do_work`, it may be possible to use nesting and parallelize its internal work. In the following code we are parallelizing an inner loop within the `do_work` function:

```
void do_work(int i)
{
```


Co-design: Calculix solver program

News
Partners
Services
Request Service Form
Resources for Co-Design
Metrics
Patterns
Best-practices
Programs
Languages
Models
Disciplines
Algorithms
Reports

CalculiX solver (OpenMP)

Version's name: CalculiX solver (OpenMP) **; a version of the CalculiX solver program.**

Repository: [home] **and version downloads:** [.zip] [.tar.gz] [.tar.bz2] [.tar]

Patterns and behaviours: Load imbalance due to computational complexity (unknown a priori) ·

Implemented best practices: Conditional nested tasks within an unbalanced phase ·

This program represents the behavior of the *GMRES* solver found in the CalculiX application. However, in contrast to the original program this version now uses the *OpenMP* programming model instead of the pthread one. It solves the first non-symmetric linear system occuring in the very first timestep of simulation of airflow through a bend pipe.

The structure of the program is given by the following (pseudo) code:

```
gmres_serial(int i)
{
    // get data chunk based on i
    // perform GMRES solver on data chunk in serial
}
```

```
#pragma omp parallel num_threads(NUM_CPUS)
{
    int thread_num = omp_get_thread_num();
```

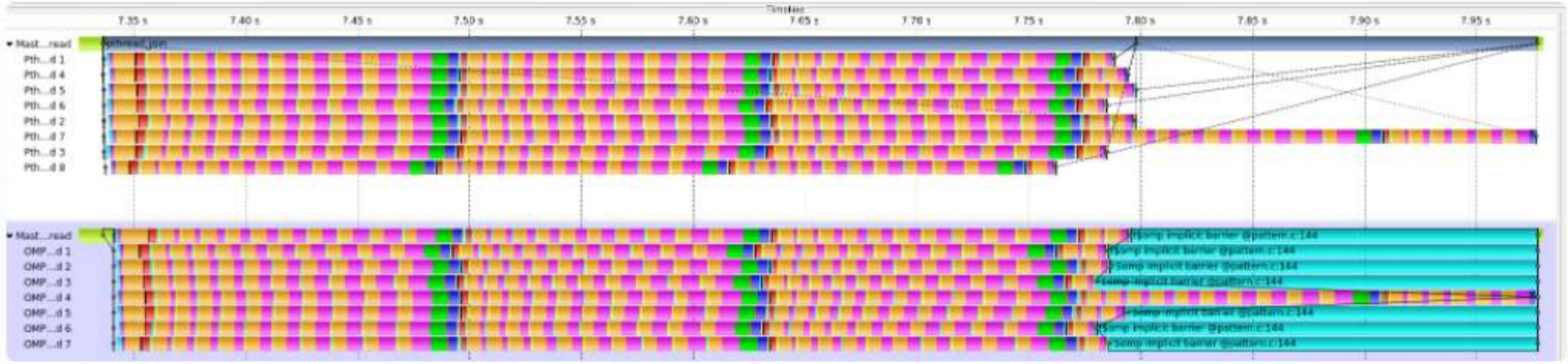

Co-design: Calculix's experiments

News
Partners
Services
Request Service Form
Resources for Co-Design
Metrics
Patterns
Best-practices
Programs
Languages
Models
Disciplines
Algorithms
Reports

CalculiX-solver trace analysis of OpenMP version

Pattern comparison

The following figure shows a comparison between a trace of the pthread version and the OpenMP version of the CalculiX-solver kernel.



ScoreP trace comparison of the CalculiX-solver kernel pthread version (top) and the OpenMP version (bottom).

In both cases the same non-symmetric linear system is solved using GMRES. The OpenMP version reveals the same pattern

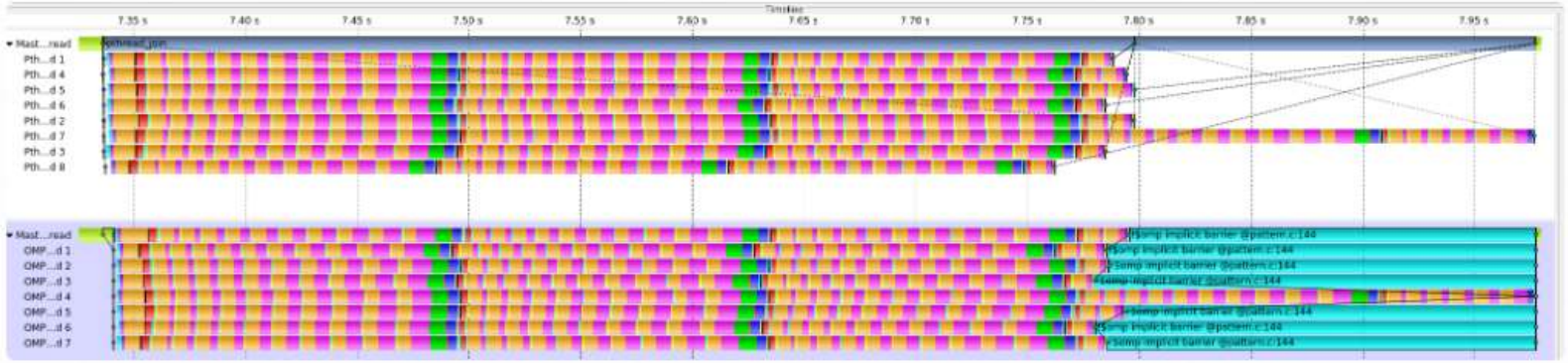
Co-design: Calculix's experiments

News
Partners
Services
Request Service Form
Resources for Co-Design
Metrics
Patterns
Best-practices
Programs
Languages
Models
Disciplines
Algorithms
Reports

CalculiX-solver trace analysis of OpenMP version

Pattern comparison

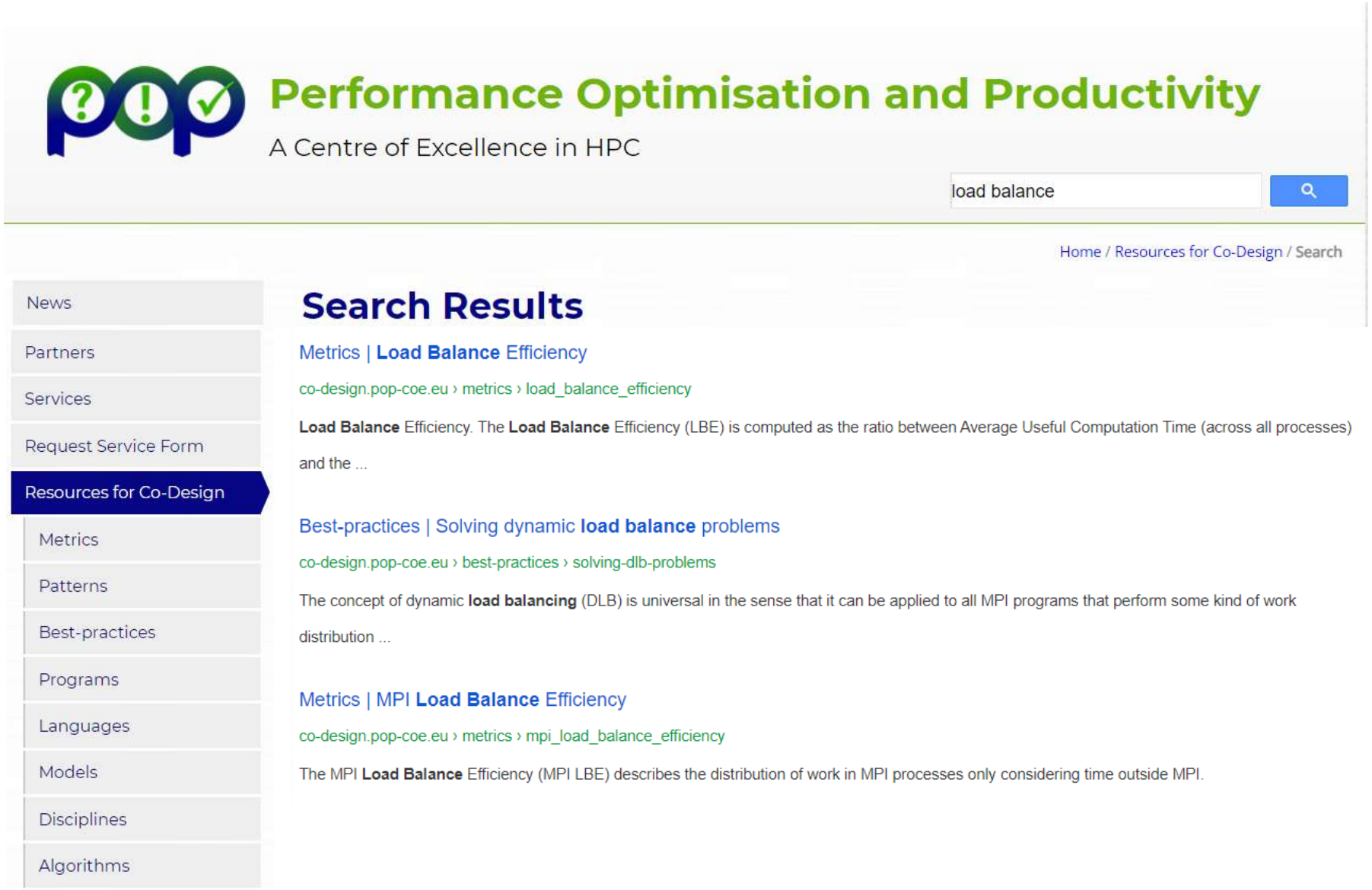
The following figure shows a comparison between a trace of the pthread version and the OpenMP version of the CalculiX-solver kernel.



ScoreP trace comparison of the CalculiX-solver kernel pthread version (top) and the OpenMP version (bottom).

In both cases the same non-symmetric linear system is solved using GMRES. The OpenMP version reveals the same pattern

Co-design: Search results



Co-design: POP reports

Home / Resources for Co-Design / Reports

- News
- Partners
- Services
- Request Service Form
- Resources for Co-Design
  - Metrics
  - Patterns
  - Best-practices
  - Programs
  - Languages
  - Models
  - Disciplines
  - Algorithms
  - Reports

List of Reports

Here you can find the complete list of all reports developed throughout this project. Specifically, you will find reports related to the POP1 and POP2 projects. These reports are classified as Assessment Reports (AR), Performance Plan Reports (PP) and Prof-of-concept Reports (PoC).

Report name	Project	Report type	Languages	Models
VeloxChem	POP2	AR	C++ · Python ·	MPI · OpenMP ·
DPM – Dynamics Simulation	POP1	AR	C++ ·	MPI ·
DROPS	POP1	AR	C++ ·	MPI · OpenMP ·
VAMPIRE	POP1	AR	C++ ·	MPI ·
NEMO	POP1	AR		MPI ·
Ateles	POP1	AR	Fortran · Python ·	MPI · OpenMP ·
GITM	POP1	AR	Fortran ·	OpenMP ·
OpenNN	POP1	AR	C++ ·	OpenMP ·
Musubi	POP1	AR	Fortran ·	MPI ·

Conclusions

- Summary: definition of co-design; description of the POP project; figures of co-design activities; navigating on the co-design site
- Co-design requirements
  - Establish a **dialog** among different design processes, provide a way to **abstract** common “problems” (e.g., design patterns), and have **empathy** with the other related design activities
- Next steps
  - More resources: more content, improve navigability, re-shape current content
  - Receive/process feedback about the site: contact us; but considering to open more interactive ways to do it
  - From *webinar* to *discussion rooms*: open discussions about co-design activitiy itself; also consider meta co-design activities (today!)

Performance Optimisation and Productivity

A Centre of Excellence in HPC

Contact:

https://www.pop-coe.eu

mailto:pop@bsc.es

@POP_HPC

And don't forget to visit:

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

Talk: *https://www.youtube.com/watch?v=g_hTeNCXP2Q*

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 676553 and 824080.

Acknowledgements

The MEEP project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 946002. The JU receives support from the European Union's Horizon 2020 research and innovation programme.