



**Barcelona  
Supercomputing  
Center**

*Centro Nacional de Supercomputación*

# Co-design, from a buzzword to a reality, an EPI success story

Marta Garcia-Gasulla

Barcelona Supercomputing Center

20 January 2025

HIPEAC25 – Barcelona

# Disclaimer

➤ Almost all the slides of this presentation come from...

- Jesus Labarta (BSC)
- Filippo Mantovani (BSC)

# Who is who? What is what?

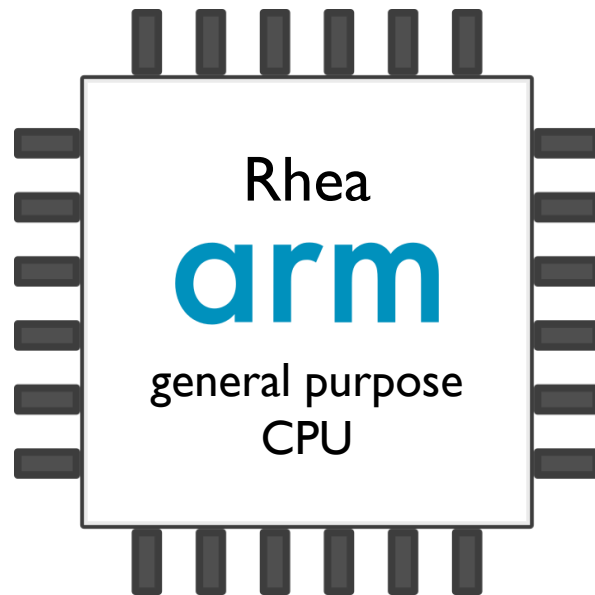


**Barcelona  
Supercomputing  
Center**

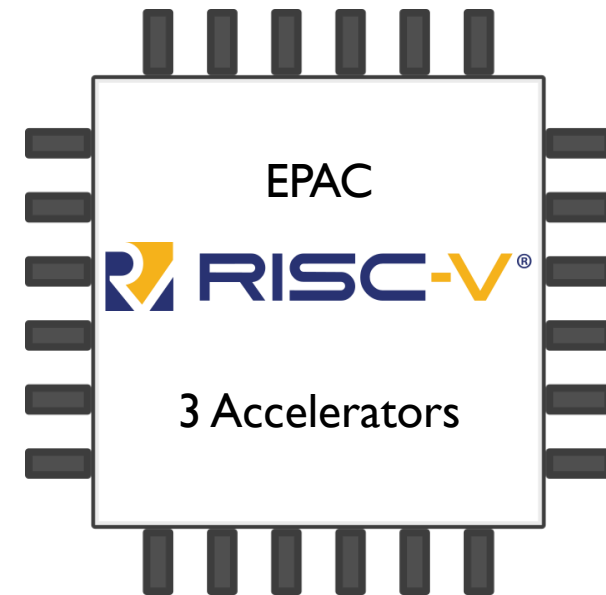
*Centro Nacional de Supercomputación*

# EPI Main Objective

- To develop European microprocessor and accelerator technology
- Strengthen competitiveness of EU industry and science



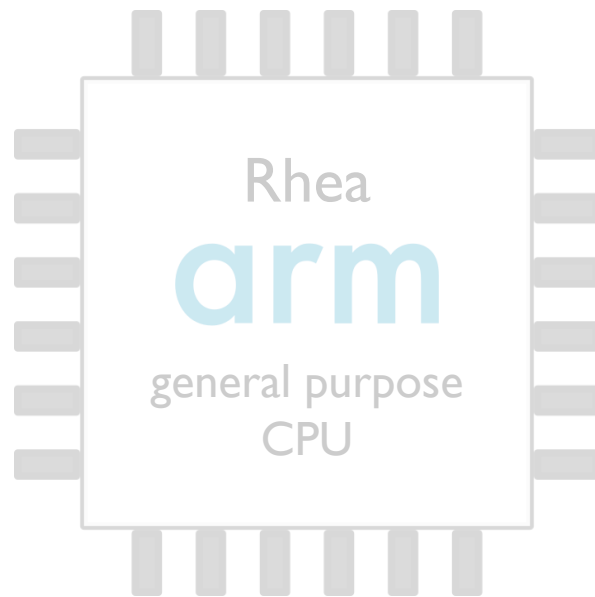
SiPearl, Atos, CEA, UniBo,  
E4, UniPi, P&R



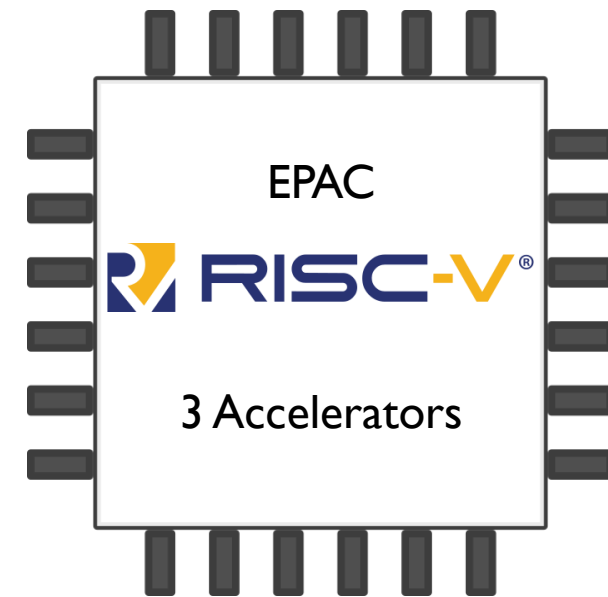
BSC, SemiDynamics, EXTOLL, FORTH,  
ETHZ, UniBo, UniZG, Chalmers, CEA, E4

# EPI Main Objective

- To develop European microprocessor and accelerator technology
- Strengthen competitiveness of EU industry and science



SiPearl, Atos, CEA, UniBo,  
E4, UniPi, P&R



BSC, SemiDynamics, EXTOLL, FORTH,  
ETHZ, UniBo, UniZG, Chalmers, CEA, E4



# EPAC: EPI Accelerator v1.5

## VEC tile

General purpose RISC-V CPU  
Avispado Core (16 kI\$, 32 kD\$)  
with dedicated VPU  
Up to 256 DP element vector length



## L2-HN tile

Distributed L2 cache (256 kB/slice) and  
Coherence Home Node



## VRP tile

General purpose RISC-V CPU  
supporting variable precision  
arithmetic up to 256 bit elements



Physical design by  Fraunhofer

Prototype board integration by 

## STX tile

RISC-V many-core machine learning  
accelerator targeting stencil and  
tensor arithmetics.



## CHI NoC and SerDes

On-chip high-speed network based  
on multiple CHI cross points (XP).

Off-chip link based on SerDes.



# EPAC: EPI Accelerator v1.5

## VEC tile

General purpose RISC-V CPU  
Avispado Core (16 kI\$, 32 kD\$)  
with dedicated VPU  
Up to 256 DP element vector length



## L2-HN tile

Distributed L2 cache (256 kB/slice) and  
Coherence Home Node



## VRP tile

General purpose RISC-V CPU  
supporting variable precision  
arithmetic up to 256 bit elements



Physical design by  Fraunhofer

Prototype board integration by 

## STX tile

RISC-V many-core machine learning  
accelerator targeting stencil and  
tensor arithmetics.



## CHI NoC and SerDes

On-chip high-speed network based  
on multiple CHI cross points (XP).

Off-chip link based on SerDes.



# What is special in EPAC-VEC?

- Boots Linux
- The scalar in-order RISC-V core can release several requests of cache lines to the main memory
- The core is connected to a Vector Processing Unit (VPU) with very wide vector registers (16kb)

**intel** – AVX512

512 bits per vector (8 DP elements)

**arm** – SVE

Up to 2048 bits per vector (32 DP elements)

**NEC** / **RISC-V**

16384 bits per vector  
(256 DP elements)



# How do I program EPAC - VEC?



## ➤ Autovectorization

- Leave it to the compiler

## ➤ `#pragma omp simd` (aka “Guided vectorization”)

- Relies on vectorization capabilities of the compiler
  - Usually works but gets complicated if the code calls functions
- Also usable in Fortran

## ➤ C/C++ builtins (aka “Intrinsics”)

- Low-level mapping to the instructions
- Allows embedding it into an existing C/C++ codebase
- Allows relatively quick experimentation

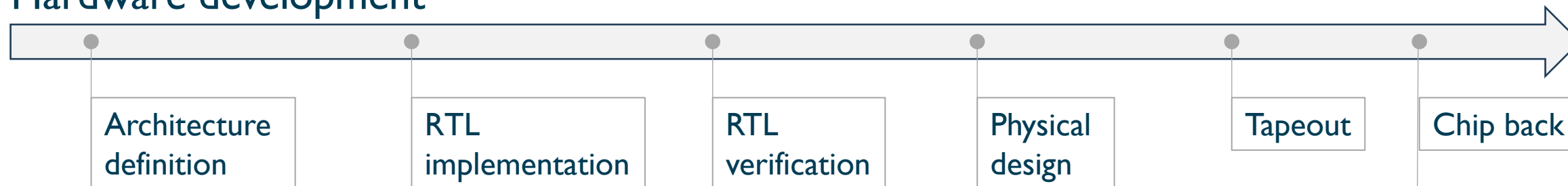


## ➤ Assembler

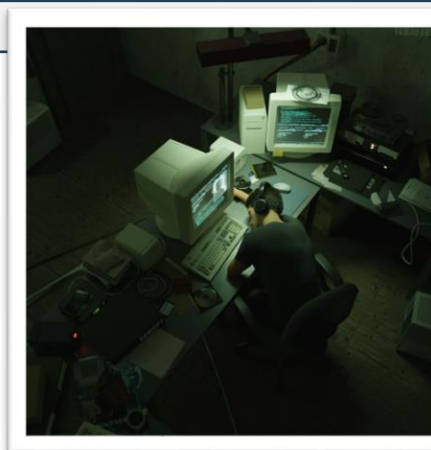
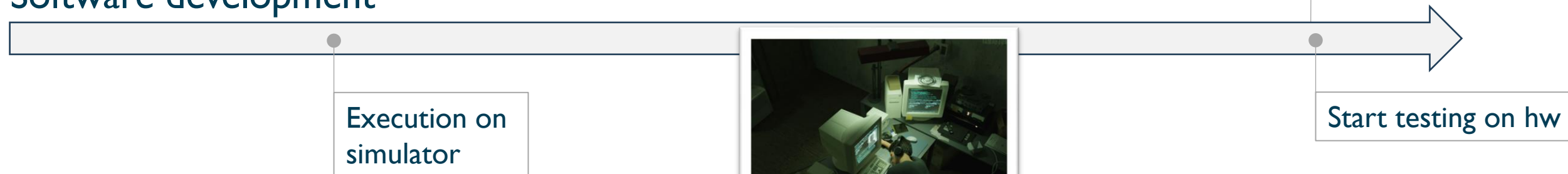
- Always a valid option but not the most pleasant

# What to do until the hardware is ready?

## Hardware development



## Software development

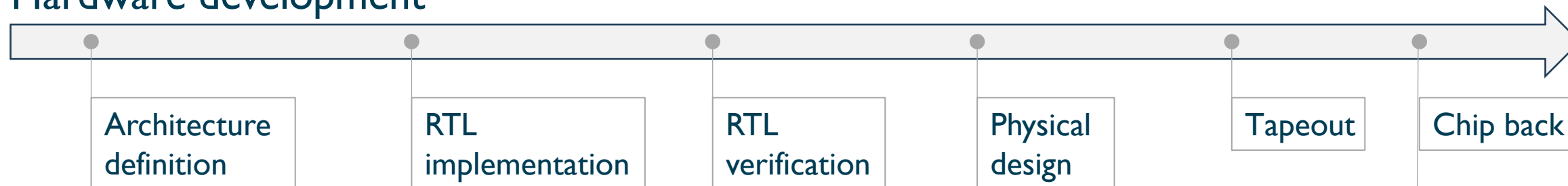


Wake up Neo...  
Follow the Software Development Vehicles

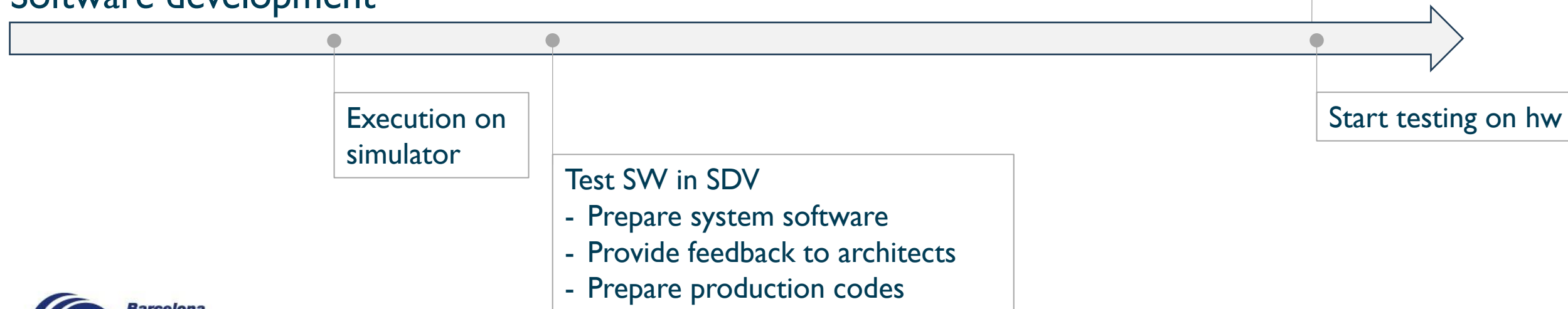


# What to do until the hardware is ready?

## Hardware development



## Software development



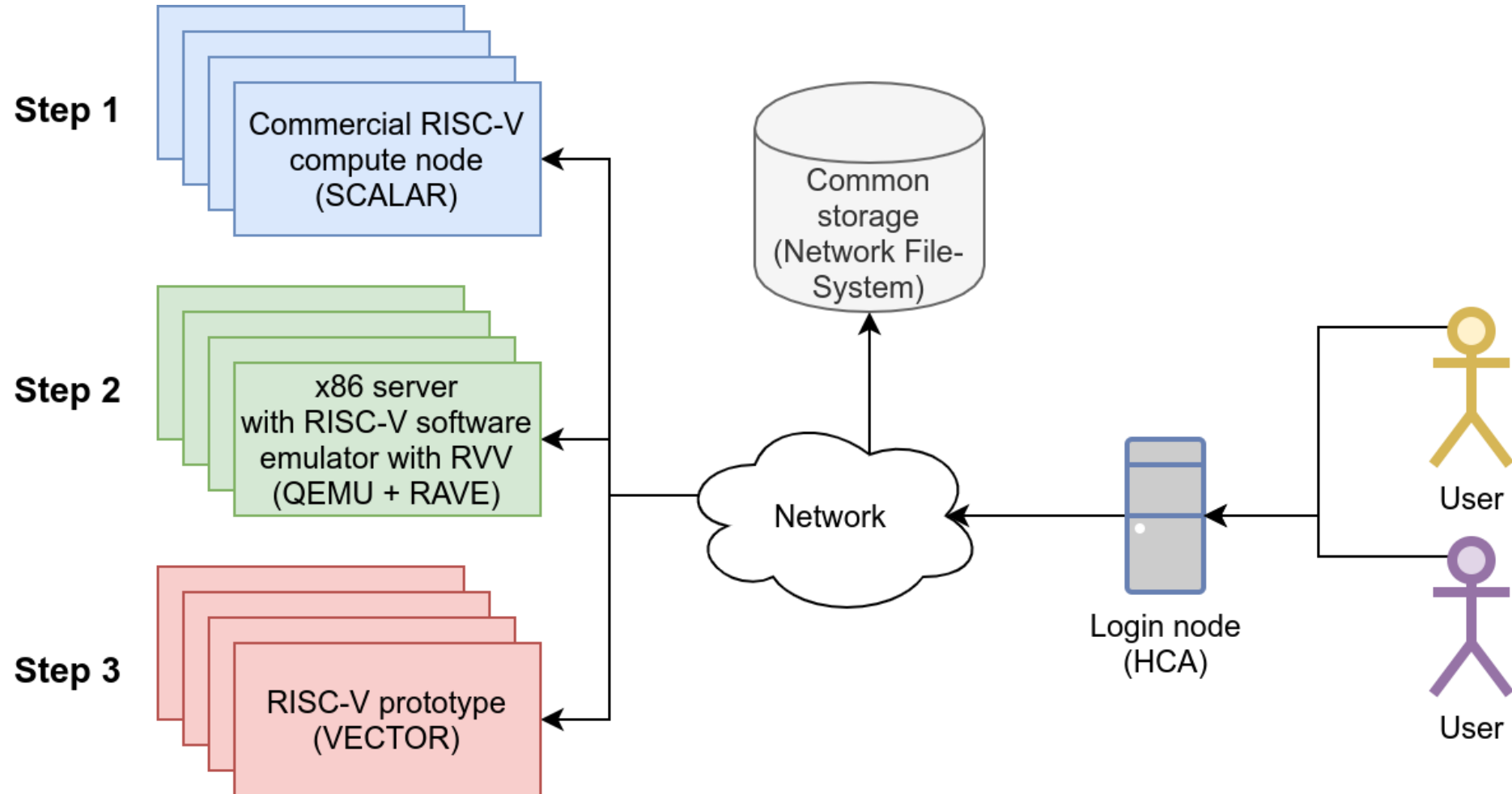
# Software Development Vehicles (SDV)



**Barcelona  
Supercomputing  
Center**

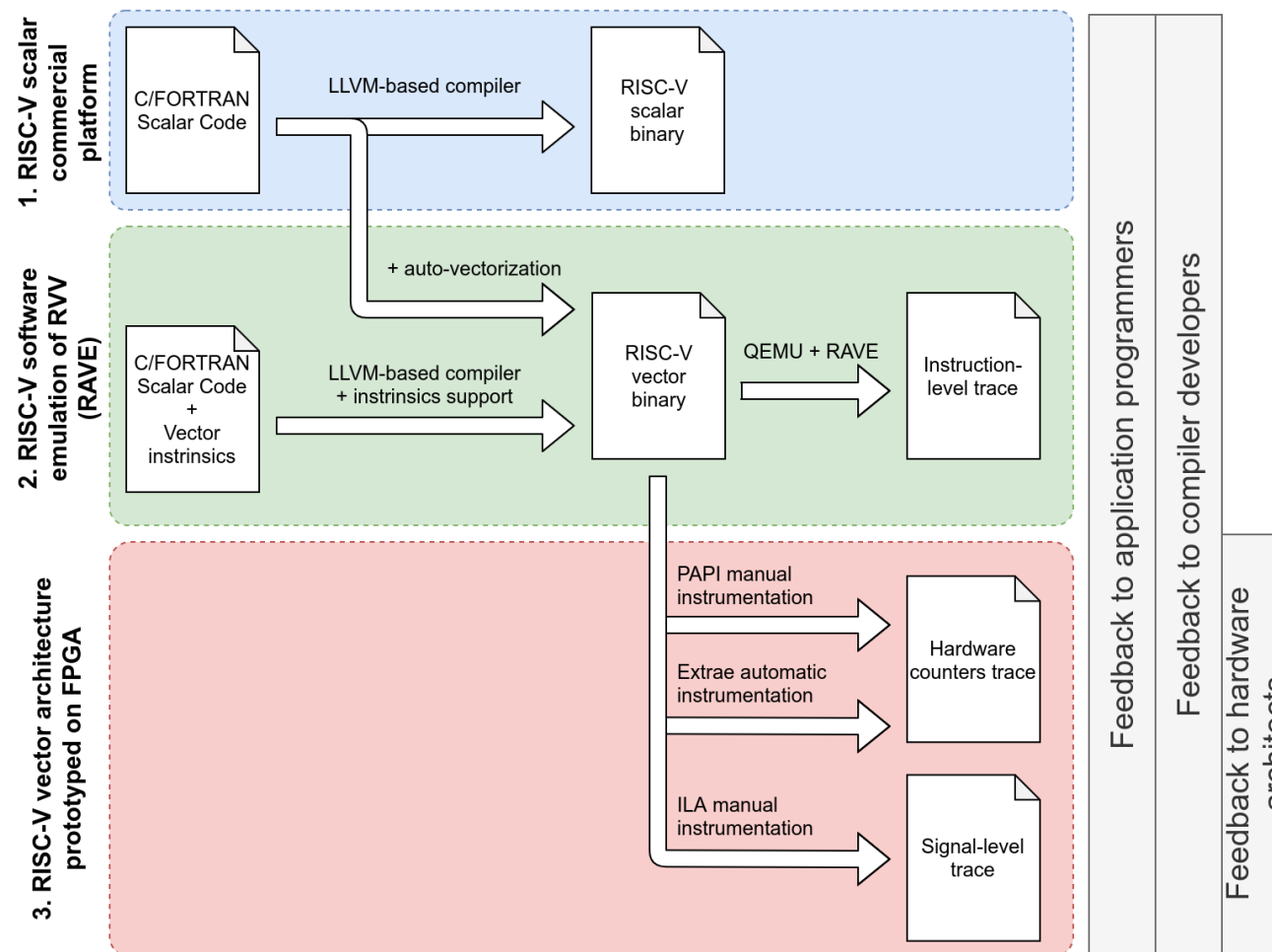
*Centro Nacional de Supercomputación*

# Software Development Vehicles (SDV)





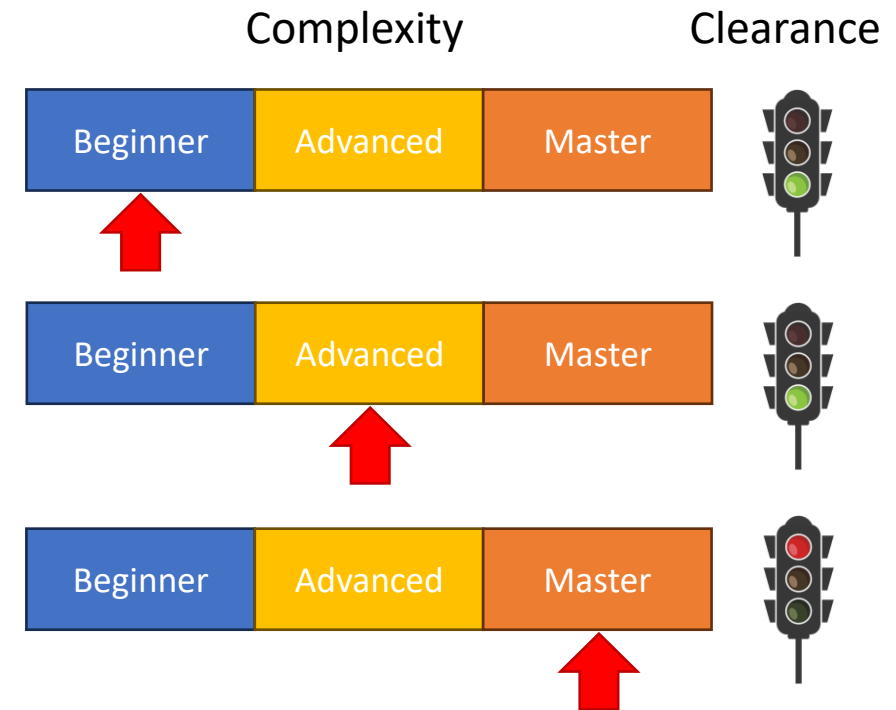
# Co-design with SDV



# Software Development Vehicles (SDV)

## ➤ 3 Steps:

- **1<sup>st</sup> step:** Run in a commercial RISC-V platform (scalar CPU)
- **2<sup>nd</sup> step:** RISC-V software emulation supporting RVV (RAVE)
- **3<sup>rd</sup> step:** Run on VEC mapped into FPGA

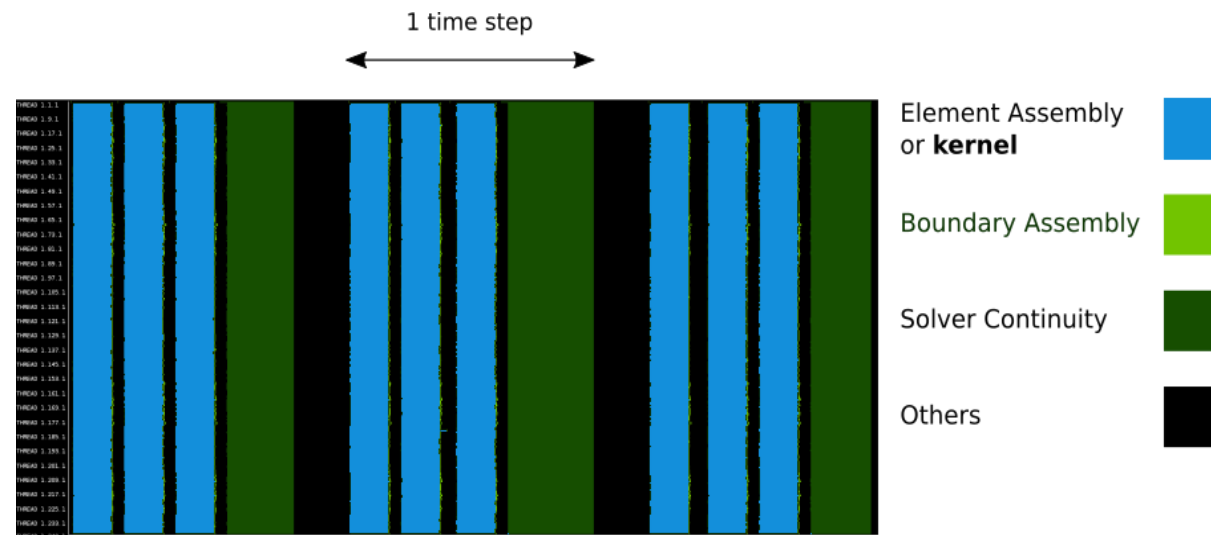


# Optimization of a CFD code for vector architectures



# Vectorization of a real CFD code (Alya)

- Alya is a multiphysics code, developed in Fortran and parallelized with MPI
- “VECTOR\_SIZE” is a parameter defined in the code at compile time to exploit vector units
  - Allocates data structures in a vector-friendly way
  - Values under study → [16, 64, 128, 240, 256, 512]
- Current work focuses on incompressible flow
  - Mini-app representative of the element assembly
- We divided the mini-app in “phases”
  - Phases are regions of codes with one or more loops
  - We are interested in loops because is where there is potential for vectorization
  - 8 phases identified: P1+P2+P3+P4+P5+P6+P7+P8 = mini-app
- Study and optimization focus on the autovectorization capabilities
  - No intrinsics → portability is preserved

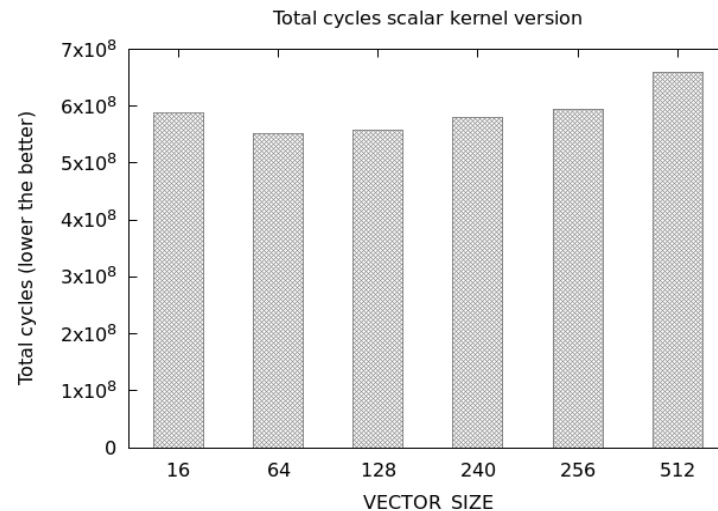




# 1<sup>st</sup> step: Run on commercial RISC-V platforms (scalar CPU)

- Phases taking longer (6,3,7,4) correspond to compute intensive regions
- Phases lasting less (5,2,8,1) are memory bound regions
- VECTOR\_SIZE parameter has almost no influence on the execution (5% coefficient of variation)

Phase	1	2	3	4	5	6	7	8
% of total cycles	1,29%	3,33%	19,80%	14,45%	3,49%	40,99%	14,68%	1,96%



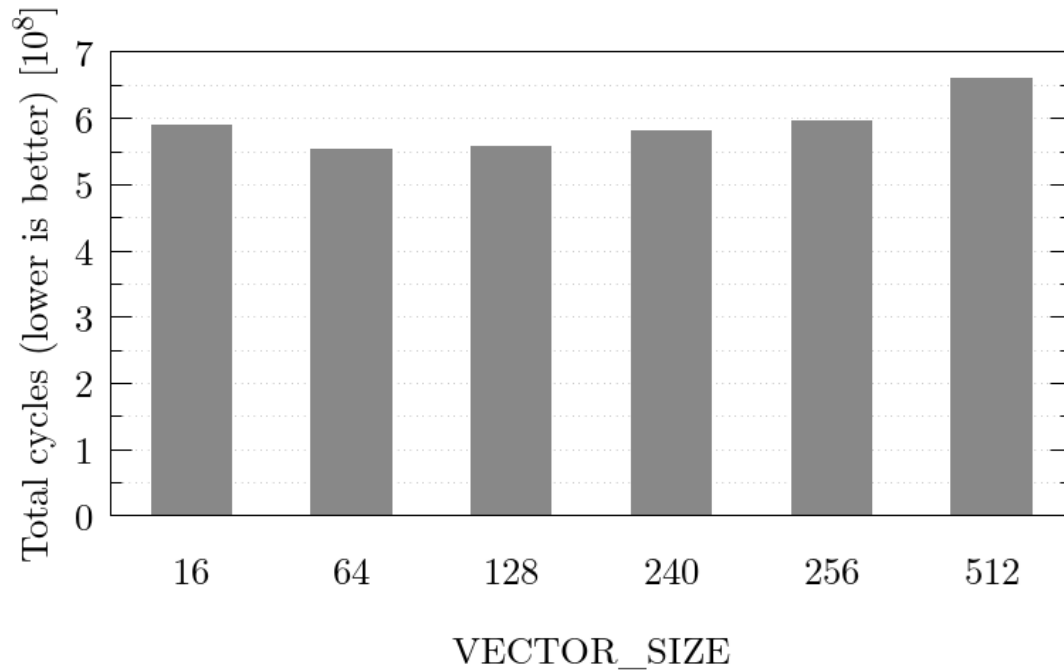
Commercial RISC-V platform (scalar CPU)



## 2<sup>nd</sup> step: Enabling auto-vectorization

- Auto-vectorization results without touching any line of code
- VECTOR\_SIZE parameter strongly influences when executing with vectors

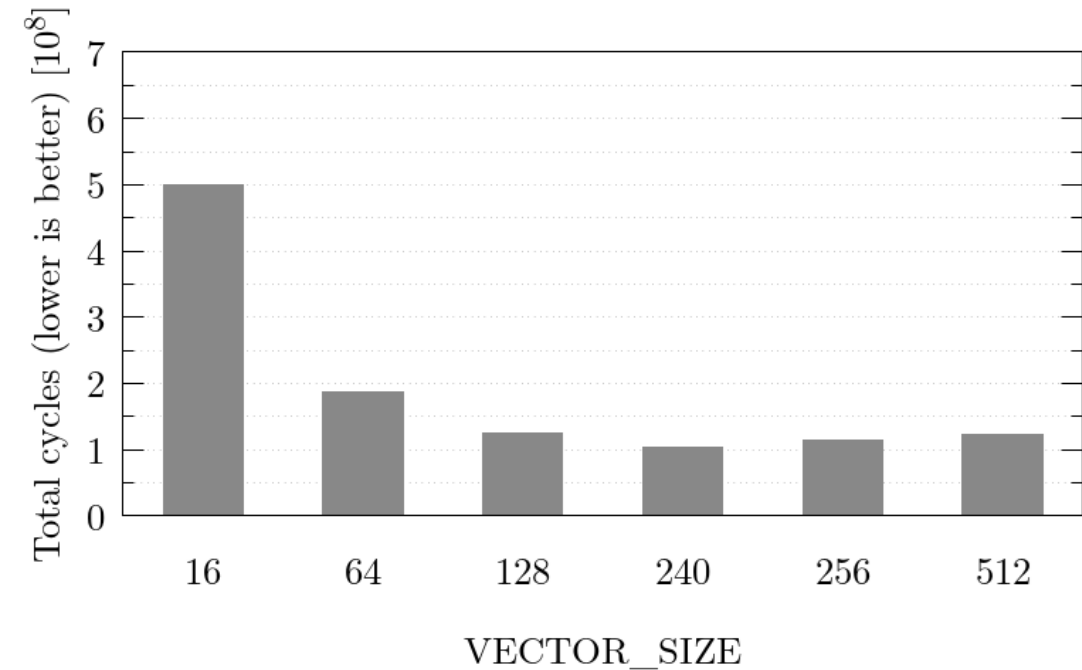
Total cycles scalar kernel version



Enabling  
Compiler  
Auto-vec



Total cycles vector kernel version



# 2<sup>nd</sup> step: Emulation supporting RVV (RAVE)

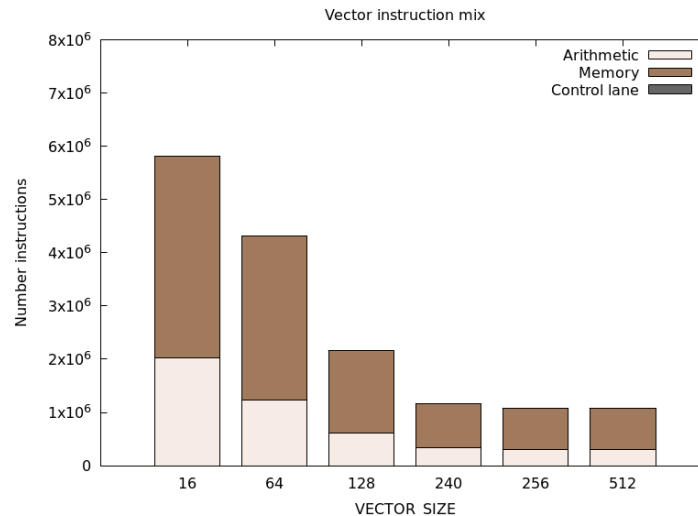
## % Vector instruction

	Phase							
VECTOR_SIZE	1	2	3	4	5	6	7	8
16	0,00%	0,00%	1,84%	0,00%	0,00%	0,95%	24,64%	0,00%
64	0,00%	0,00%	12,73%	17,37%	17,86%	21,58%	25,87%	0,00%
128	0,00%	0,00%	16,05%	16,80%	17,94%	20,39%	25,23%	0,00%
240	0,00%	0,00%	15,31%	16,45%	16,82%	19,90%	23,90%	0,00%
256	0,00%	0,00%	15,36%	16,21%	15,88%	19,78%	24,23%	0,00%
512	0,00%	0,00%	16,65%	18,19%	18,47%	21,82%	26,20%	0,00%

## For Phase 6

AVL	Number vector instructions
16	$14.3 \times 10^5$
64	$19.1 \times 10^5$
128	$9.6 \times 10^5$
240	$5.1 \times 10^5$
256	$4.7 \times 10^5$
256	$4.7 \times 10^5$

30,00%
15,00%
0,00%



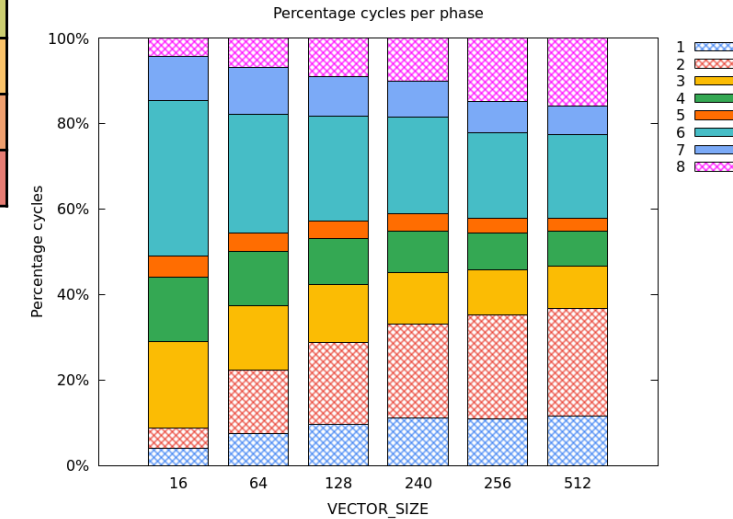
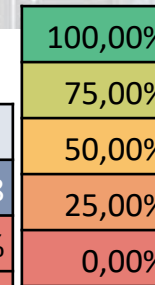
## Analysis of % of vector instructions:

- Higher VECTOR\_SIZE helps the compiler to insert more vector instructions
- Higher VECTOR\_SIZE → Increases AVL → reduces the total number of vector instructions
- 70% of vector instructions are memory type

# 3<sup>rd</sup> step: Run on VEC mapped into FPGA

## % Vector instruction

	Phase							
VECTOR_SIZE	1	2	3	4	5	6	7	8
16	0,00%	0,00%	1,84%	0,00%	0,00%	0,95%	24,64%	0,00%
64	0,00%	0,00%	12,73%	17,37%	17,86%	21,58%	25,87%	0,00%
128	0,00%	0,00%	16,05%	16,80%	17,94%	20,39%	25,23%	0,00%
240	0,00%	0,00%	15,31%	16,45%	16,82%	19,90%	23,90%	0,00%
256	0,00%	0,00%	15,36%	16,21%	15,88%	19,78%	24,23%	0,00%
512	0,00%	0,00%	16,65%	18,19%	18,47%	21,82%	26,20%	0,00%



## Analysis of % of vector cycles:

- High vCPI → we are computing several elements per instruction (GOOD)
- AVL == VECTOR\_SIZE → the more elements we process per vector instruction, the less vector instructions we execute (GOOD)

VECTOR_SIZE	vCPI	AVL	Number vector instructions
16	9.71	16	$14.3 \times 10^5$
64	23.39	64	$19.1 \times 10^5$
128	28.56	128	$9.6 \times 10^5$
240	41.19	240	$5.1 \times 10^5$
256	43.10	256	$4.7 \times 10^5$
512	45.30	256	$4.7 \times 10^5$

Phase 2 is not vectorized and accounts for 30% of the time with VECTOR\_SIZE=512

vCPI, AVL and # vector instructions phase 6

# Example of optimization: phase 2 aka VEC2

## Problem

- Compiler unable to vectorize loop, not sure of VECTOR\_DIM value

```
subroutine nsi_miniapp(VECTOR_DIM, pnode, pgaus, list_elements)
```

```
1 loop not vectorized: unsafe dependent memory operations in loop.
```

## Solution

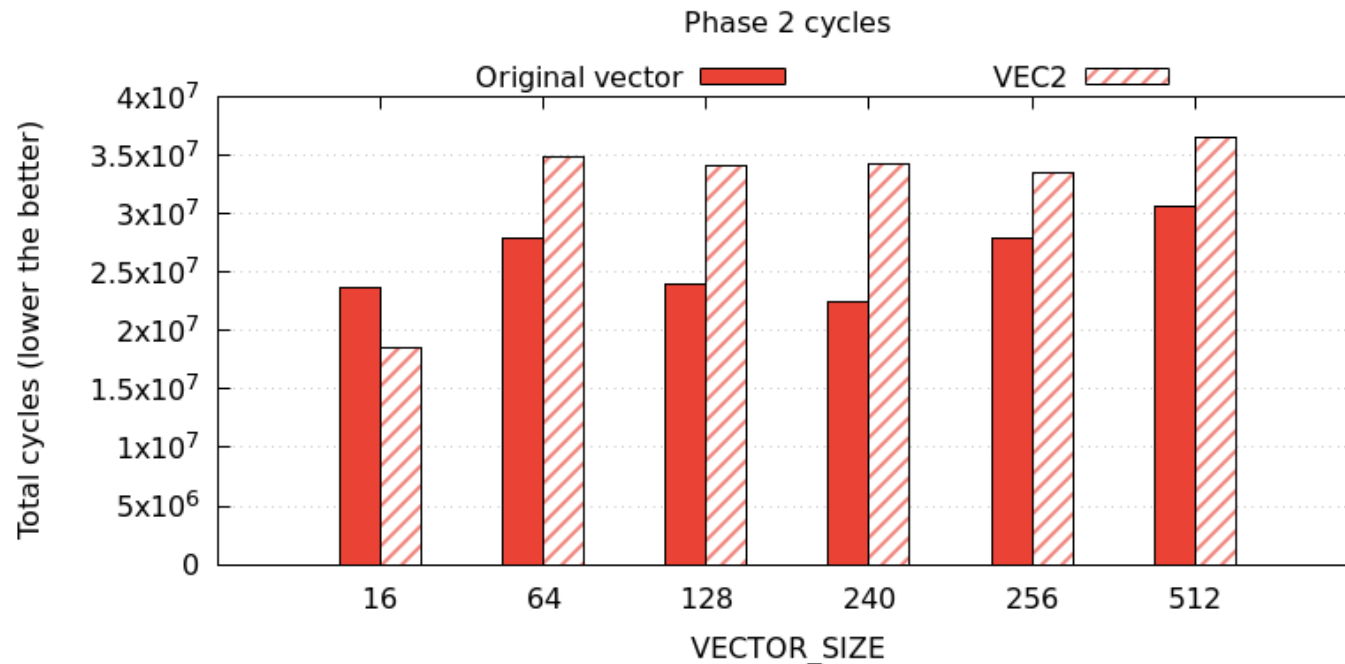
- We know VECTOR\_DIM value

```
integer(ip), parameter :: VECTOR_DIM = VECTOR_SIZE
```

# Optimization - VEC2

## ➤ Enabled vectorization in phase 2

- Performance get worst instead of improving
- AVL of vector instructions is low! ⚠  
We are not taking advantage of the full-VL. Why?





# Optimization - VEC2+VL

## Problem

- pnode comes from input, we do not know its value
- Experimentally found  $pnode \ll VECTOR\_DIM$

```
do ivect = 1, VECTOR_DIM  
  do inode = 1, pnode  
    !WORK  
  end do  
end do
```

## Solution

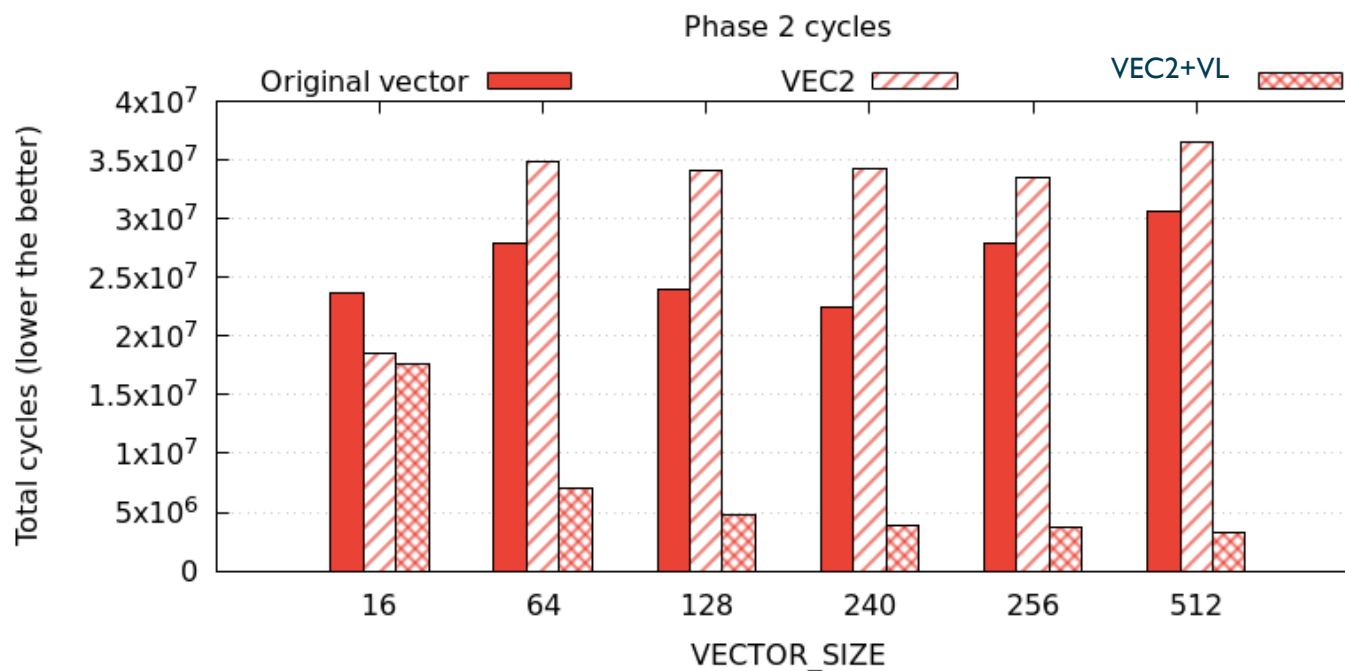
- Swap induction variables

```
do inode = 1, pnode  
  do ivect = 1, VECTOR_DIM  
    !WORK  
  end do  
end do
```

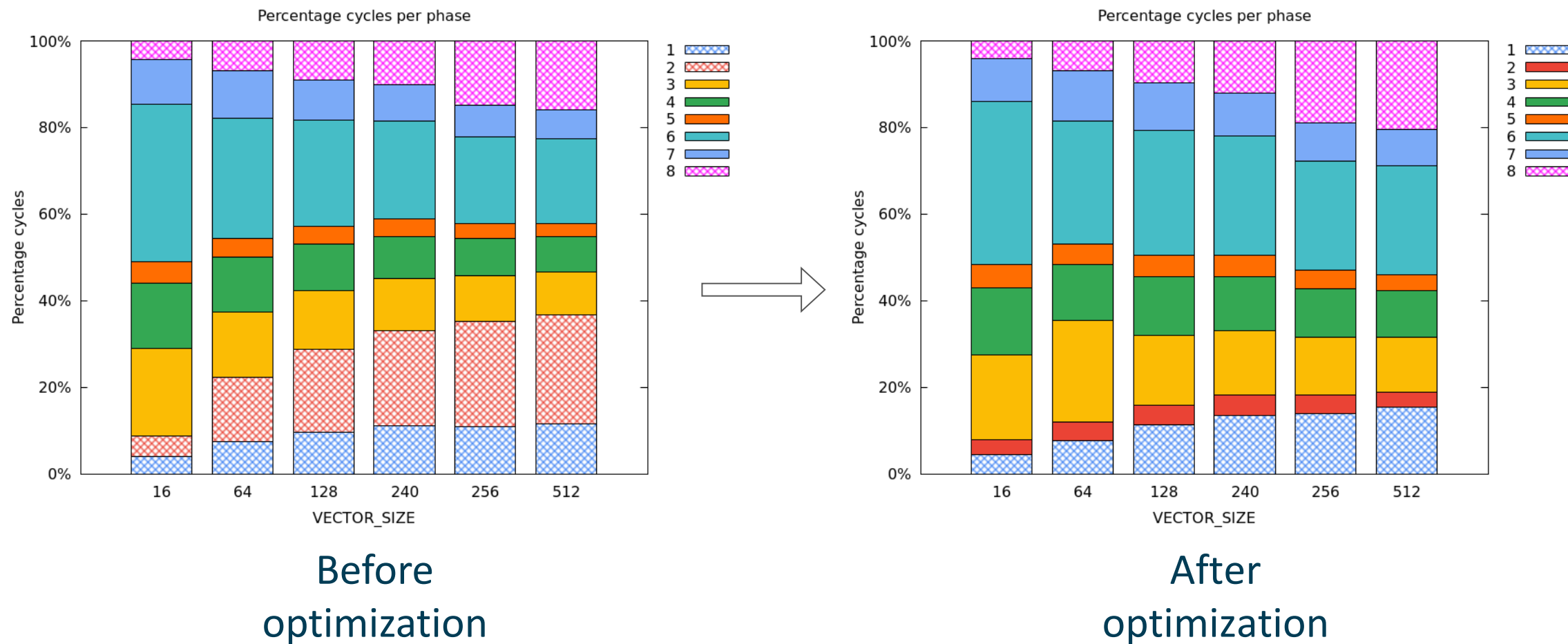
# Optimization VEC2+VL: results

## ➤ Improved AVL vectorization in phase 2

- Vector instructions running with  $AVL == VECTOR\_SIZE$

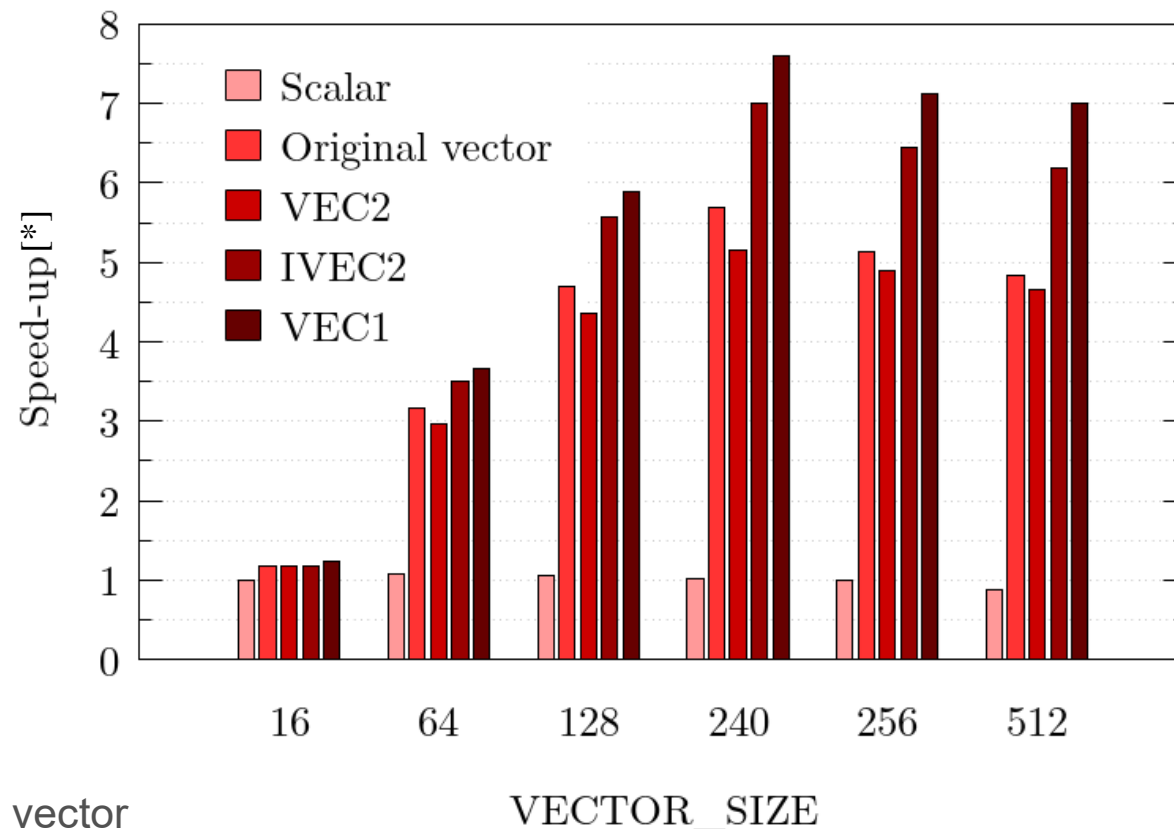


# Alya preliminary results - VEC2+VL



# Evaluation: RISC-V vector prototype

- After a detailed study and manual optimizations, we achieve a peak of 7.6x speedup (VEC1)
- Code remains portable  
No intrinsics!

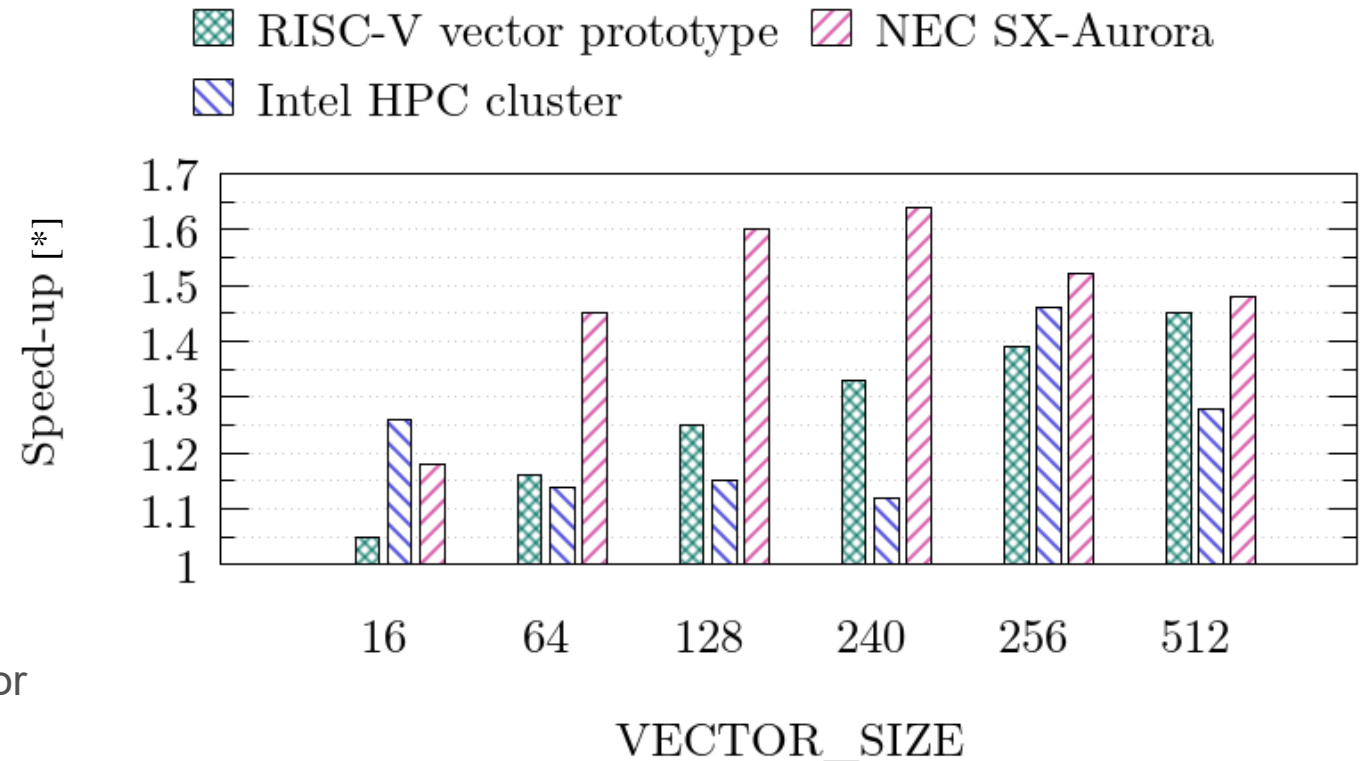


[\*] Speed-up defined as: scalar VECTOR\_SIZE<sub>16</sub> / optimized vector

# Portability across other HPC platforms

## ➤ Optimizations portable to other architectures

- “Traditional” cluster (Intel x86)
- Long-vector architecture (NEC SX-Aurora)



[\*] Speed-up defined as: vanilla vector / optimized vector



# Take home message

- EPI is developing:
  - Arm-based CPU (not part of this talk/workshop)
  - RISC-V-based Accelerator
  
- We focus on the RISC-V vector accelerator (VEC) that:
  - Can be self-hosted
  - Support variable vector length
  - Is vector length agnostic
  - Uses long vectors (256 DP elements, 32x larger than x86)
  - Boots linux (it is programmed as a regular core)

# Take home message

- While RTL is becoming actual hardware, EPI develops tools for boosting the co-design cycle
  - Software and Hardware prototypes (aka Software Development Vehicles)
- We can leverage SDVs to:
  - Influence hardware design
  - Improve compiler autovectorization and system-software support
  - Study and improve vectorization of real scientific HPC codes
- We leveraged the EPI SDVs to study and improve vectorization of a complex CFD code (Alya) written in Fortran
- Vectorization techniques improve performance on EPAC – VEC and are portable
- Similar studies are on going for several scientific codes part of EU CoEs



Blancafort, Marc, et al. "Exploiting long vectors with a CFD code: a co-design show case." 2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2024.

# Thank you!



**Barcelona  
Supercomputing  
Center**

*Centro Nacional de Supercomputación*

HIPEAC25 - Co-design, from a buzzword to a reality, an EPI success story – Marta Garcia-Gasulla

# EPAC 1.5 architecture summary

