
EPI Forum
Paris, 6-7 October, 2025
European
Processor
Initiative
epi

European Processor Initiative

RISC-V in High-End HPC

Filippo Mantovani, Computer Science Department, Barcelona Supercomputing Center (Spain)

Mario Kovač, Department of Control and Computer Engineering, Univ. of Zagreb (Croatia)

Nikolaos Dimou, Institute of Computer Science at FORTH (Greece)

RISC-V in High-End HPC

Filippo Mantovani, Computer Science Department, Barcelona Supercomputing Center (Spain)

Mario Kovač, Department of Control and Computer Engineering, Univ. of Zagreb (Croatia)

Nikolaos Dimou, Institute of Computer Science at FORTH (Greece)

What is RISC-V?

DISCLAIMER: Apple users may not fully understand this slide 🤷‍♀️

What is RISC-V?

Core Instruction Formats

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

RV32I Base Integer Instructions

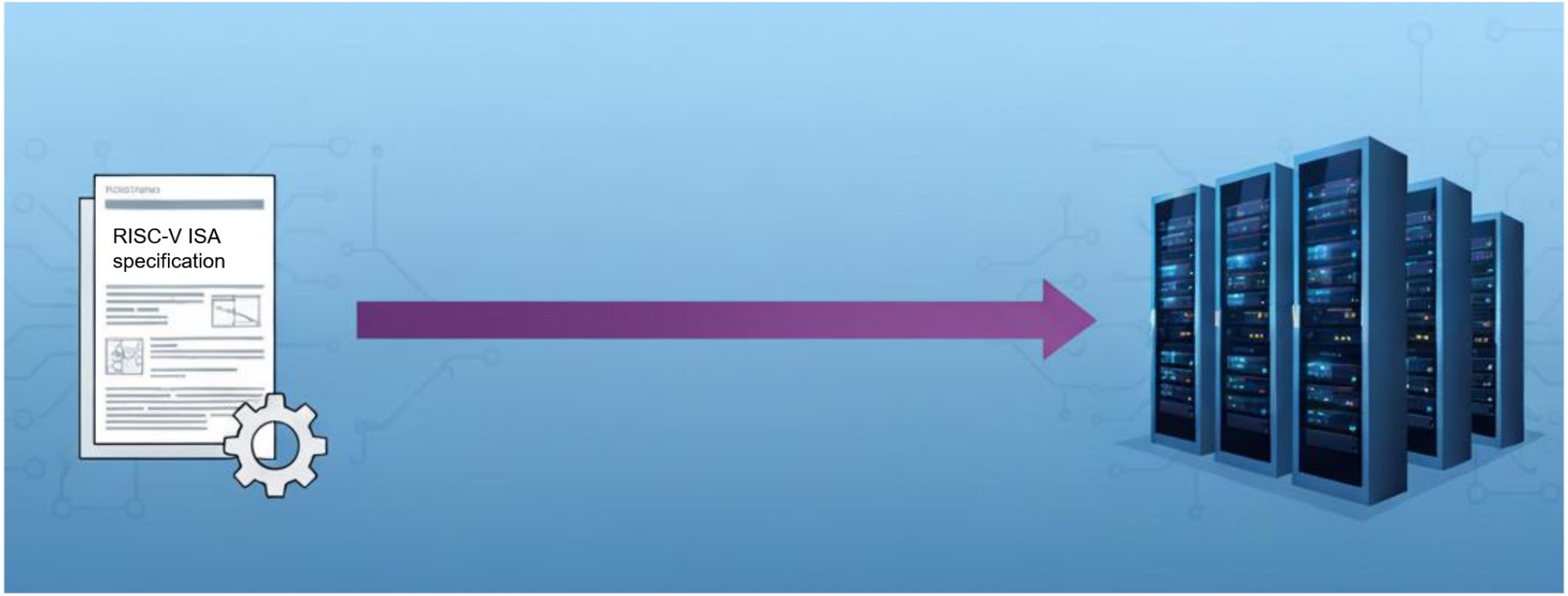
Inst	Name	FMT	Opcode	funct3	funct7	Description (C)	Note
add	ADD	R	0110011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0110011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0110011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0110011	0x6	0x00	rd = rs1 | rs2	
and	AND	R	0110011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0110011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0110011	0x5	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0110011	0x5	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2	0x00	rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3	0x00	rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011	0x0		rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x4		rd = rs1 ^ imm	
ori	OR Immediate	I	0010011	0x6		rd = rs1 | imm	
andi	AND Immediate	I	0010011	0x7		rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	imm[5:11]=0x00	rd = rs1 << imm[0:4]	
srli	Shift Right Logical Imm	I	0010011	0x5	imm[5:11]=0x00	rd = rs1 >> imm[0:4]	
srai	Shift Right Arith Imm	I	0010011	0x5	imm[5:11]=0x20	rd = rs1 >> imm[0:4]	msb-extends
slti	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltiu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≥	B	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1 + imm	
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall	Environment Call	I	1110011	0x0	imm=0x0	Transfer control to OS	
ebreak	Environment Break	I	1110011	0x0	imm=0x1	Transfer control to debugger	

- Instruction Set Architecture (ISA)
  - Simple and modular
- Research project started at Berkeley in 2010
  - Ratified ISA in 2014
- RISC-V (pronounced “risc five”, as it is the fifth generation of RISC ISA at Berkeley)

Waterman, Andrew., Patterson, David A.. **The RISC-V Reader: An Open Architecture Atlas**. United States: Strawberry Canyon LLC, 2017.

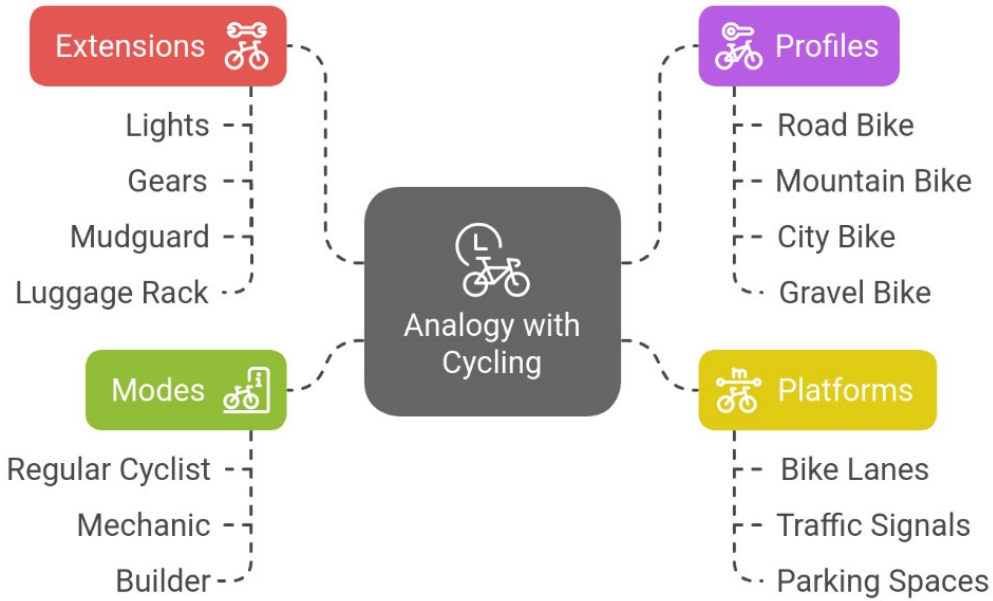
Patterson, David A.., Hennessy, John L.. **Computer Organization and Design RISC-V Edition: The Hardware Software Interface**. Netherlands: Elsevier Science, 2017.

How do we go from ISA specs to HPC?



There are steps taken by the community…

- **Extensions**
  - Optional set of instructions that can be added to the base instruction set to enhance functionality.
  - E.g., like floating-point arithmetic (F), vector processing (V), or atomic operations (A).
- **Profiles**
  - Predefined combination of base instructions + extensions
  - It defines a standard configuration for a category of RISC-V processors.
  - It ensures software compatibility across different implementations.
- **Platforms**
  - It defines a complete system environment, including hardware, firmware, and software conventions.
  - It specifies how a RISC-V processor interacts with peripherals, boot sequences, operating system expectations, etc.
- **Modes**
  - Different levels of access to system resources:
  - User Mode (U) – For normal applications, limited access.
  - Supervisor Mode (S) – For OS kernels, can manage memory and processes.
  - Machine Mode (M) – The highest privilege level, used for low-level firmware and system initialization.



...and there are steps taken by brave teams

Today you will learn about a couple of those brave teams…
the ones taking part in the EPI Stream 3 “RISC-V related activities”

EPAC: EPI Accelerator v1.5

GF22FDX, 27 mm2, 0.3 Btr Tape out Mar 2023, Bring up Oct 2023

VEC tile

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

VRP tile

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

STX tile

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

L2-HN tile

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

CHI NoC and SerDes

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

Off-chip link based on SerDes.

Physical design by Fraunhofer
Prototype board integration by E4 COMPUTER ENGINEERING

A set of silicon-ready European IP's

- RISC-V powered cores and compute units
  - VEC: Self hosted RISC-V CPU
    + wide VPU (256 DP elements) including
    - Avispado: 1st gen in-order core by Semidynamics
    - Atrevido: 2nd gen out-of-order core by Semidynamics
    - Vitruvius: 1st gen VPU, supporting RVV 0.7.1 by BSC
    - EVA: 2nd gen VPU, supporting RVV 1.0 by BSC
  - STX by Fraunhofer and ETH Zurich
  - VRP by CEA
  - KVX by Kalray
  - FPU by Univ. of Zagreb
  - IEEE/Posit VMAC Unit by IST
- Uncore technology
  - NoC by Extoll
  - High-speed SERDES and Chip2Chip link by Extoll
  - L2 cache and home node by FORTH and Chalmers
  - IOtile by Univ. of Bologna and Pisa
  - PCIe by Eviden
    - eFPGA by Menta
  - Ziptillion by ZeroPoint
- Integration of test-chips on boards
  for bring-up and testing by FORTH
  - Board for EPAC 1.5 by E4 and SECO
  - Board for EUPILOT-VEC by E4 and LINKS Foundation

Demonstrated on FPGA

Demonstrated on ASIC test-chip

A set of silicon-ready European IP's

- RISC-V powered cores and compute units
  - VEC: Self hosted RISC-V CPU + wide VPU (256 DP elements) including 3
    - Avispado: 1st gen in-order core by Semidynamics
    - Atrevido: 2nd gen out-of-order core by Semidynamics
    - Vitruvius: 1st gen VPU, supporting RVV 0.7.1 by BSC
    - EVA: 2nd gen VPU, supporting RVV 1.0 by BSC
  - STX by Fraunhofer and ETH Zurich
  - VRP by CEA
  - KVX by Kalray
  - FPU by Univ. of Zagreb 1
  - IEEE/Posit VMAC Unit by IST
- Uncore technology
  - NoC by Extoll
  - High-speed SERDES and Chip2Chip link by Extoll
  - L2 cache and home node by FORTH and Chalmers
  - IOtile by Univ. of Bologna and Pisa
  - PCIe by Eviden
    - eFPGA by Menta
  - Ziptillion by ZeroPoint
- Integration of test-chips on boards for bring-up and testing by FORTH 2
  - Board for EPAC 1.5 by E4 and SECO
  - Board for EUPILOT-VEC by E4 and LINKS Foundation

Demonstrated on FPGA

Demonstrated on ASIC test-chip

FAUST, the FPU powering EPAC chips

Mario Kovač, Department of Control and Computer Engineering, Univ. of Zagreb (Croatia)

Introduction

- FAUST © is a high-performance, pipelined, customizable floating-point unit (FPU) designed for integration into RISC-V cores with vector processing capabilities
- Compliant with IEEE 754-2019 and RISC-V Vector extension (RVV) v1.0
- Designed with a focus on minimizing its silicon area while maintaining a rich feature set and full compliance with the aforementioned standards with numerous customization capabilities to address various customer needs.
- Integrated, taped out and post-silicon validated as part of EPAC1.0, EPAC1.5 and VEC chips.

EPAC 1.0

FAUST WAS FIRST TAPEOUT AS A PART OF EUROPEAN PROCESSOR INITIATIVE EPAC CHIP

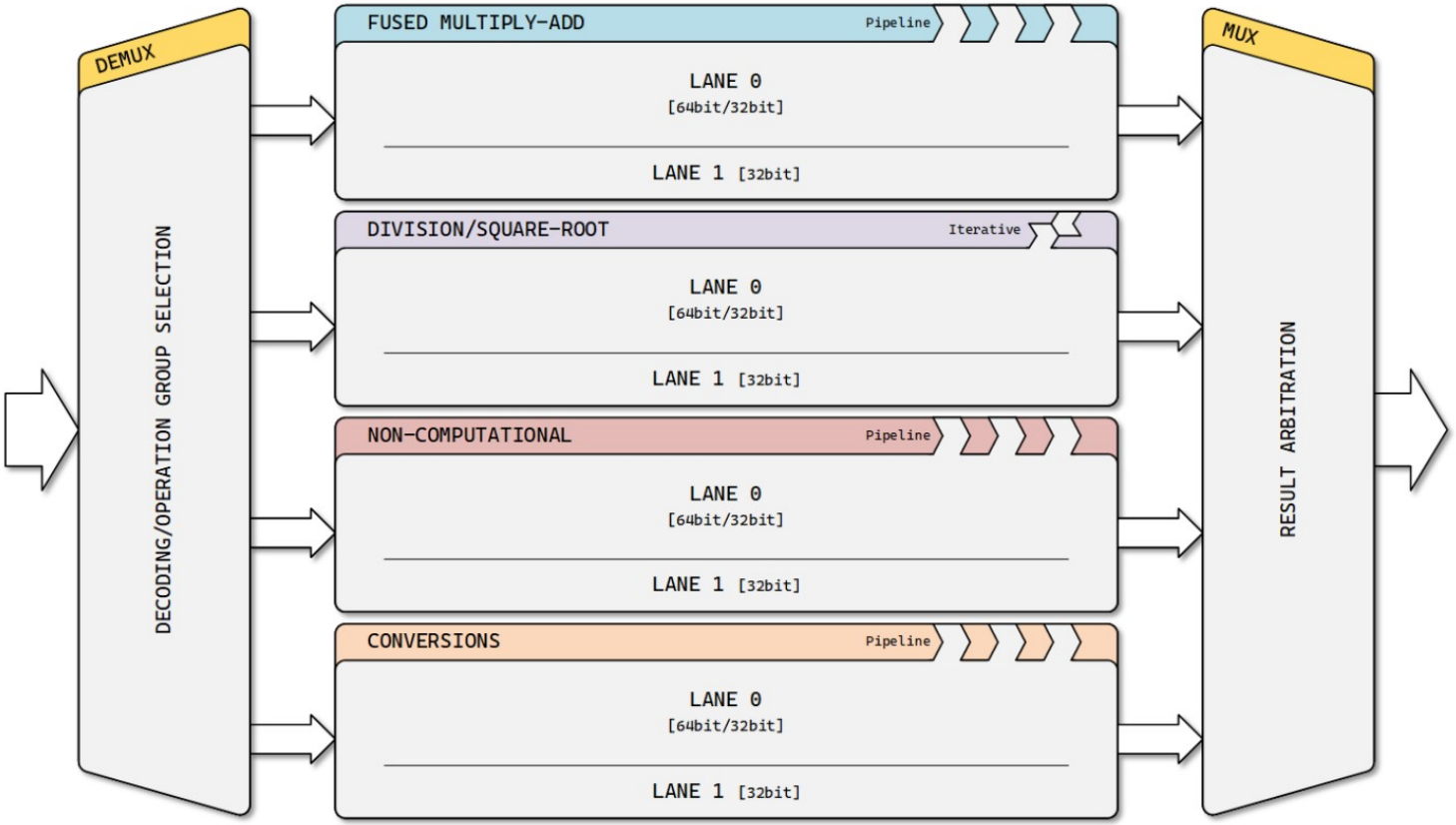
EPAC 1.5

WE MADE SOME IMPROVEMENTS AND INCLUDED THOSE IN EPAC 1.5

Features

- Supports multiple floating-point formats and rounding modes, subnormal numbers
- Aspects of the architecture adjustable by configuration parameters:
  - A choice of in-order or out-of-order execution
  - Internal pipeline setup
    - Number of pipeline stages
    - Placement of pipeline registers
    - Instantiation of skid buffers
  - Computation algorithm choice
  - Operations included
  - Lane configurations
  - Interface to VPU/CPU
  - …

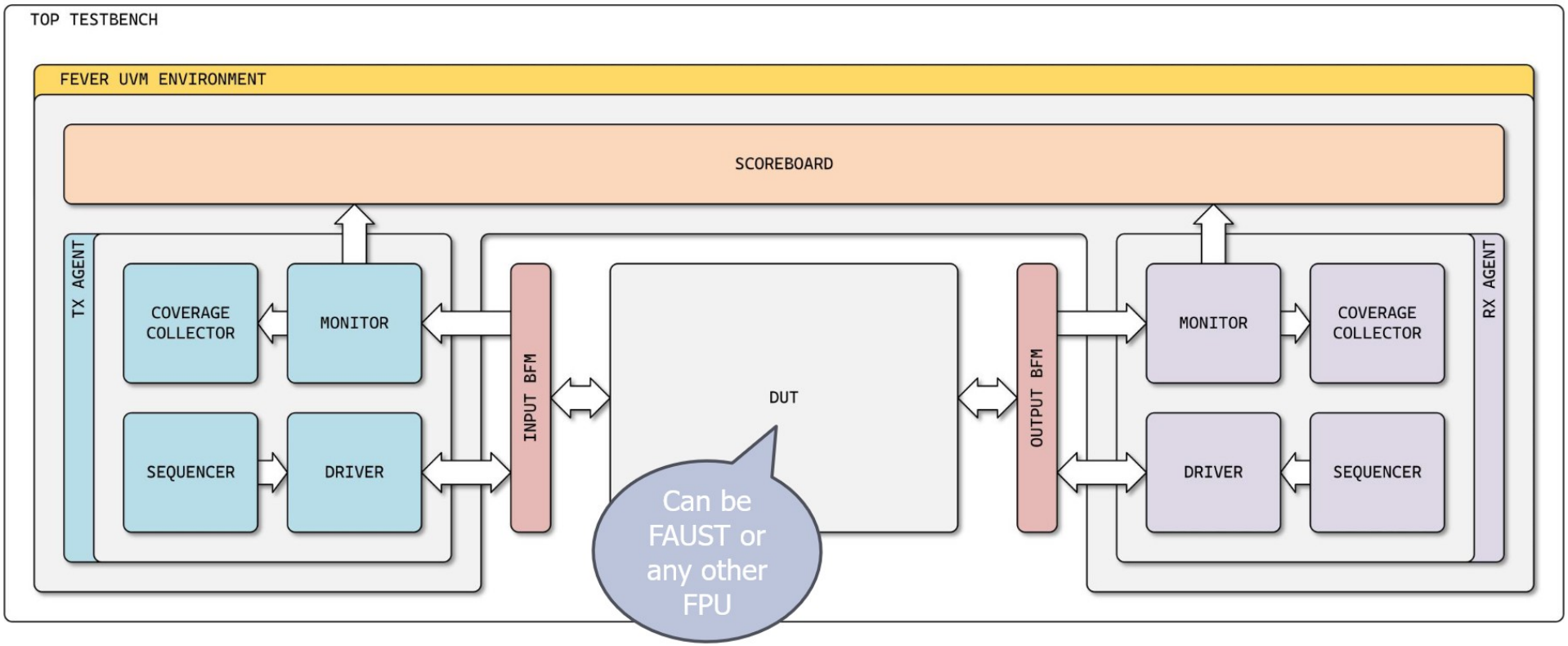
Top-level


DEMUX
DECODING/OPERATION GROUP SELECTION
FUSED MULTIPLY-ADD
Pipeline
LANE 0
[64bit/32bit]
LANE 1 [32bit]
DIVISION/SQUARE-ROOT
Iterative
LANE 0
[64bit/32bit]
LANE 1 [32bit]
NON-COMPUTATIONAL
Pipeline
LANE 0
[64bit/32bit]
LANE 1 [32bit]
CONVERSIONS
Pipeline
LANE 0
[64bit/32bit]
LANE 1 [32bit]
MUX
RESULT ARBITRATION

FEVER© Verification environment

- UVM-based environment
- FEVER reference model – extension of the open-source Berkley SoftFloat C library
  - IEEE 754-2019 compatibility
  - Additional operations required for full RISC-V Vector extension v1.0 support
- The verification environment supports all operations for *binary16*, *binary32,* and *binary64* operands.
- A SystemVerilog DPI wrapper integrates FEVER into the UVM scoreboard to evaluate UUT's outputs
- This wrapper uses a random number generator to randomize input operands and incorporates corner cases

Verification environment



Verification environment

- As part of the UVM environment, several tests were developed to verify the correct functionality of the FPU.
- The verification environment utilizes coverpoints and covergroups to ensure comprehensive testing of all possible combinations of operations, rounding modes, input widths, and other input variables.
- SystemVerilog Assertions (SVAs) were integrated into our verification environment to ensure Faust's adherence to required rules and properties

ADVANCED Release – FAUST 2.0

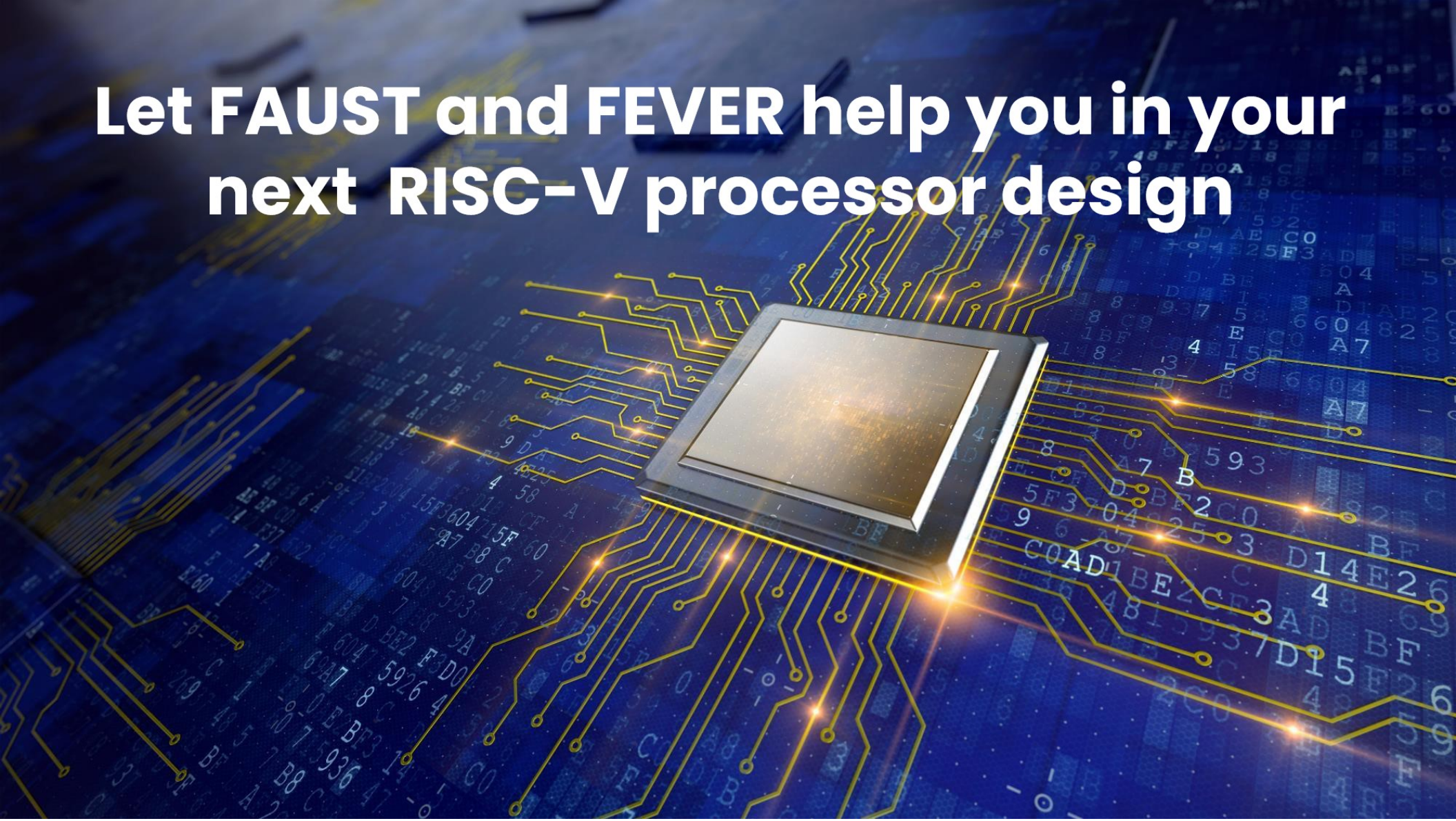
- Achieved significant area reduction while preserving a comprehensive feature set and full standards compliance
- Configurable manual placement of pipeline registers to ensure an optimally segmented pipeline, eliminating the need and sub-optimal results from automatic retiming from synthesis tools
- Multiple algorithm choices for multiplication and division/square-root, offering different area-speed tradeoffs

PROACTIVE
COMPUTE

proactivecompute.com

Follow the news …

EPI forum, Paris 6-7 October 2025


Let FAUST and FEVER help you in your next RISC-V processor design

European Processor Initiative

Test-chips and prototype platforms

Nikolaos Dimou, Institute of Computer Science at FORTH (Greece)

EPAC: EPI Accelerator v1.5

GF22FDX, 27 mm2, 0.3 Btr Tape out Mar 2023, Bring up Oct 2023

VEC tile

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

STX tile

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

VRP tile

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

L2-HN tile

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

Physical design by Fraunhofer
Prototype board integration by E4 COMPUTER ENGINEERING

CHI NoC and SerDes

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

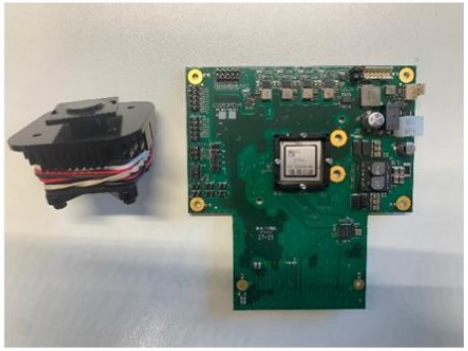
Off-chip link based on SerDes.

EPAC 1.5 Prototype: Components

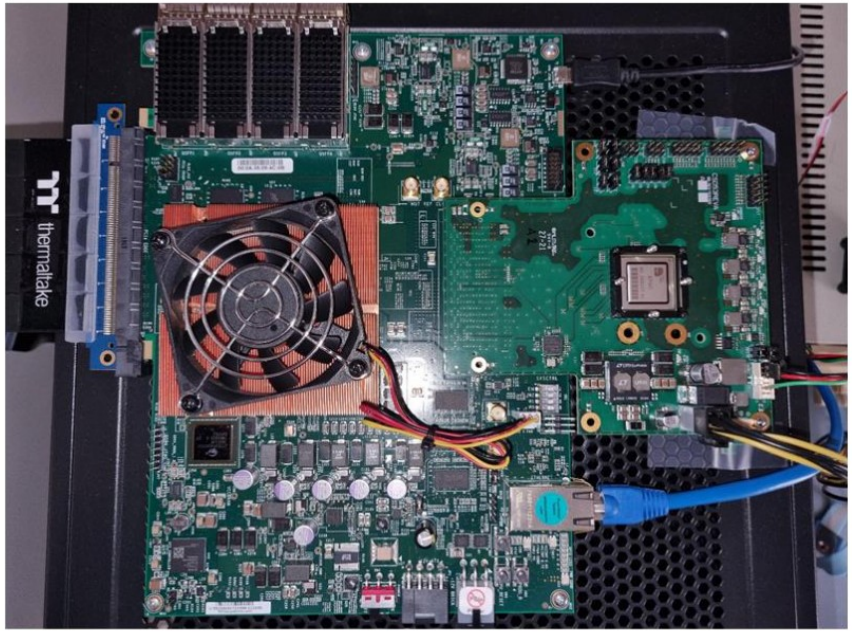
EPAC 1.5 CHIP

EPAC Daughtercard

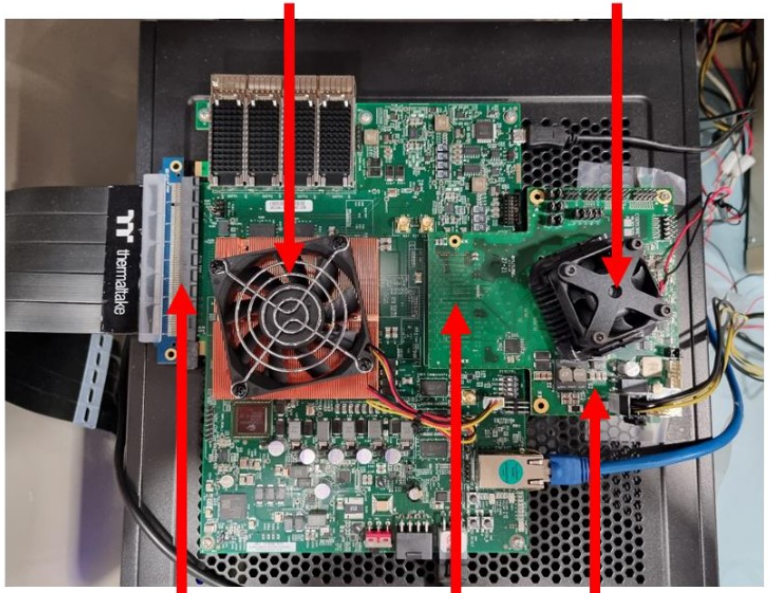
CHIP + Daughtercard

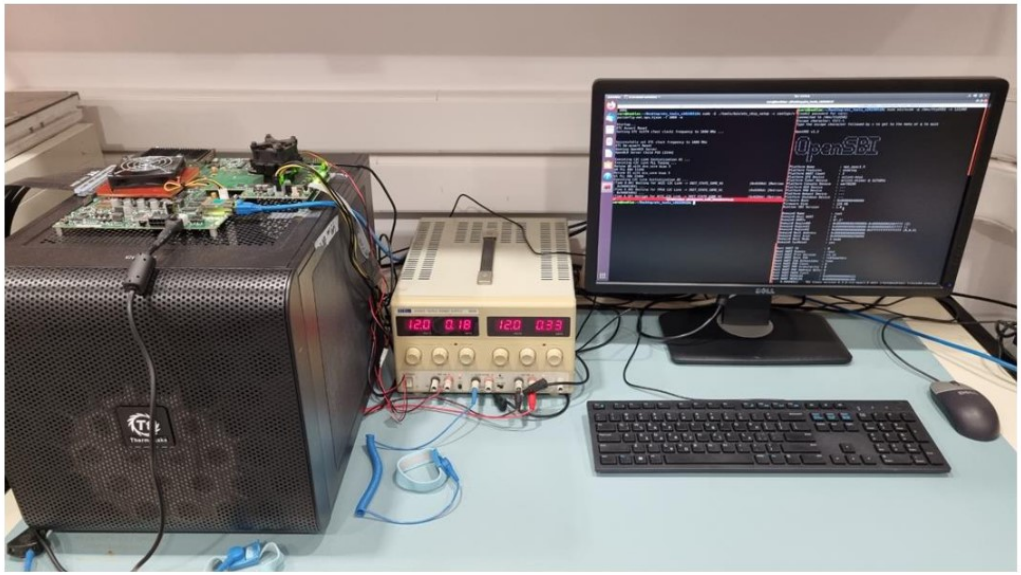


CHIP + Daughtercard + VCU128 FPGA (DDR4 + HBM + PCIe + Ethernet)



EPAC 1.5 Prototype: Full Setup





Bring-Up Setup at FORTH's Lab

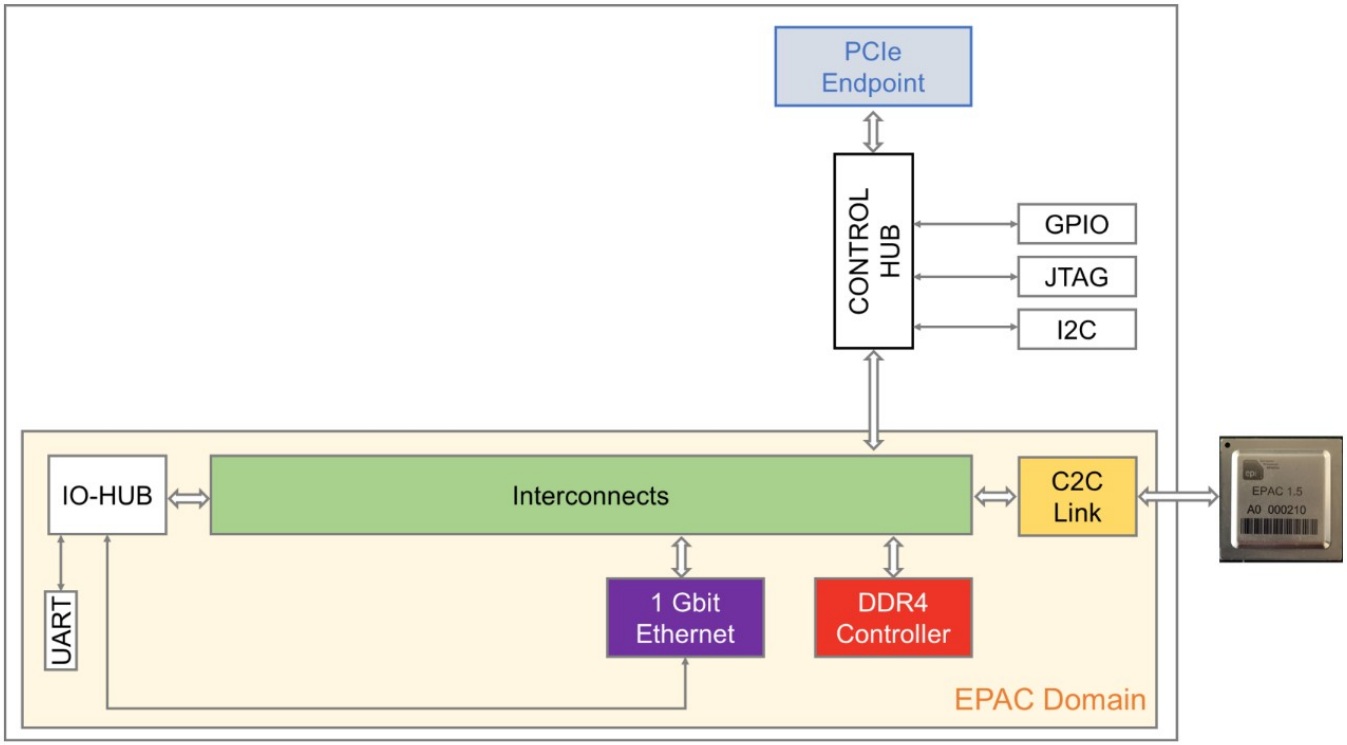
EPAC 1.5 FPGA Design

Key Components:

- PCIe EP Controller
- DDR4 Controller
- Chip-2-Chip Link
- 1Gb Eth. Controller
- Interconnects

Custom FPGA Logic:

- Low Latency / High Bandwidth Memory Accesses
- Protocol Conversions

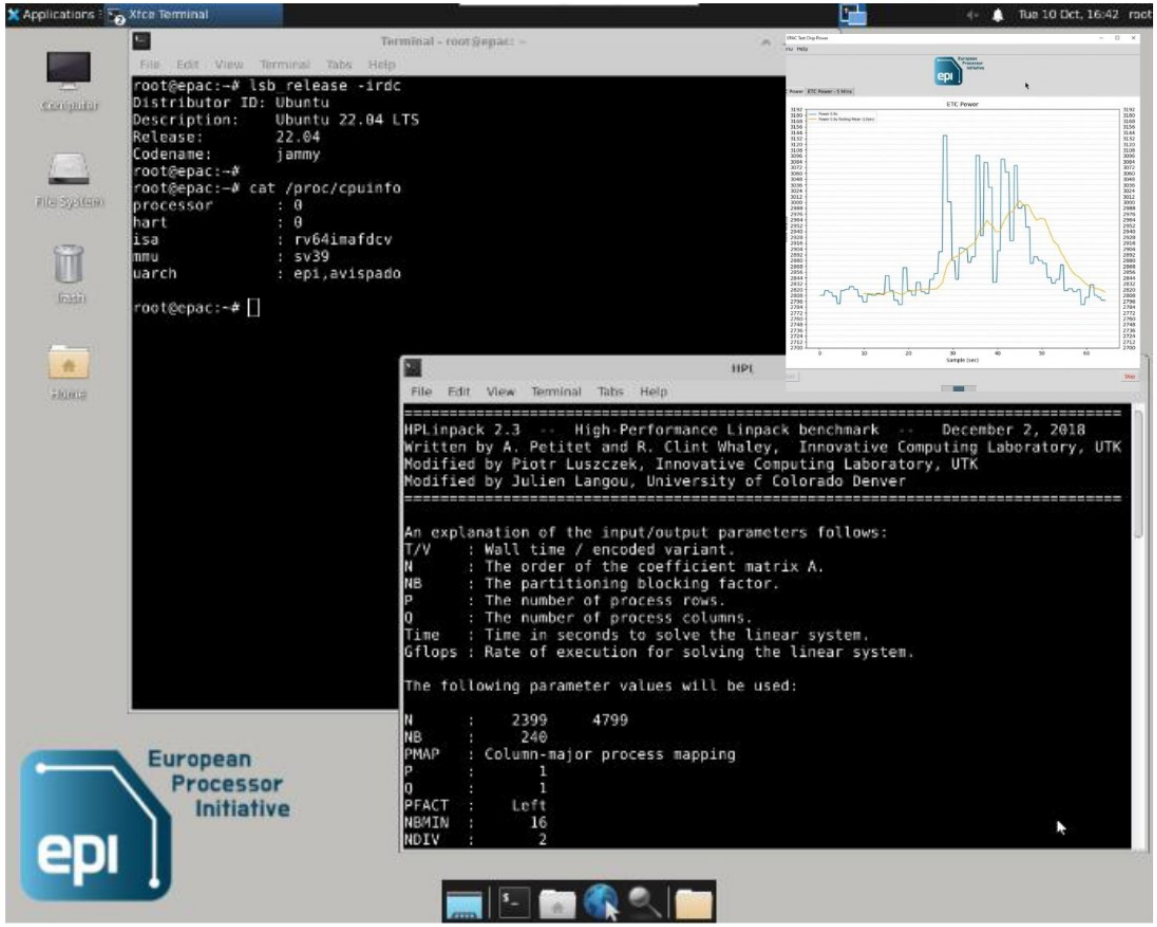


EPAC 1.5 Linux Boot

- Linux 5.7 + RVV 0.7.1
- Ubuntu 22.04 LTS
  - NFS Root FS
  - SSH
  - GUI via VNC
- Like a desktop ☺

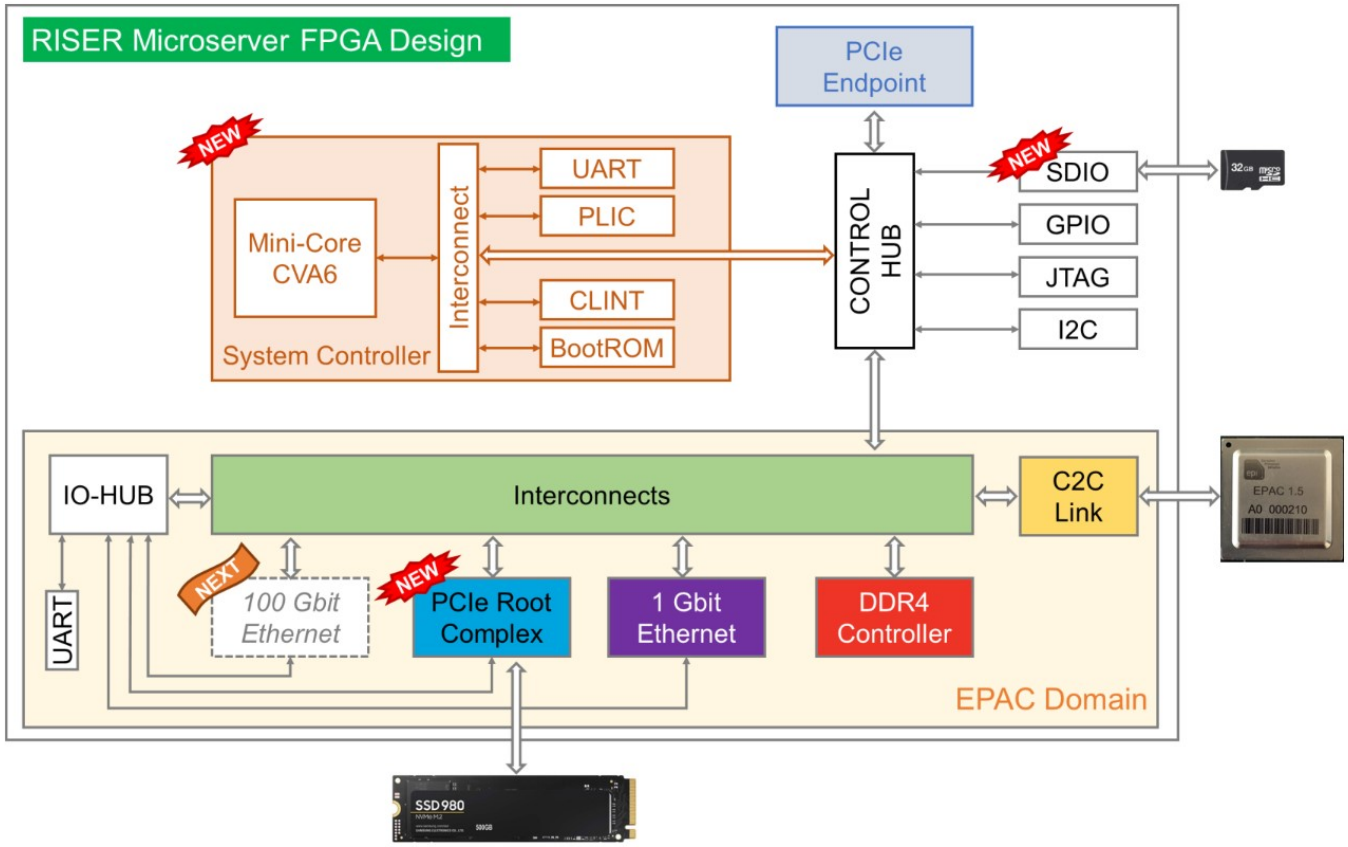
- Running LINPACK
  - Vectorized RVV 0.7.1

- Real-time power measurements



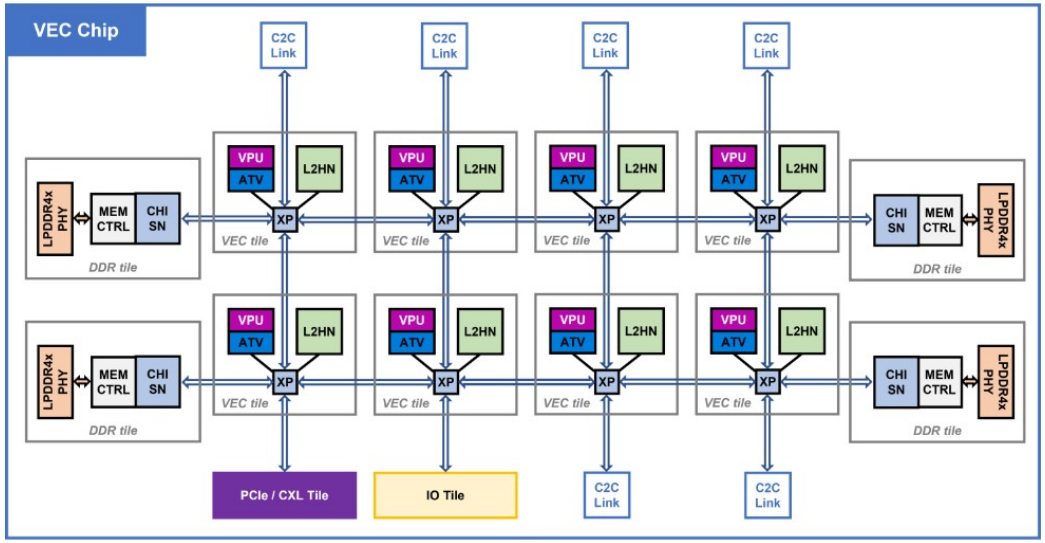
EPAC 1.5: From Accelerator to Microserver

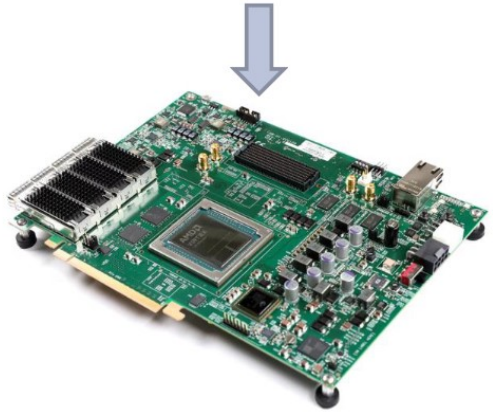
- Riser project: RISC-V for Cloud Servers
- Extending the FPGA design for the Microserver
  - System Controller
  - SDIO Controller
  - PCIe Root Complex
  - 100Gb Eth. (WIP)



EPAC-VEC SDV FPGA Prototype

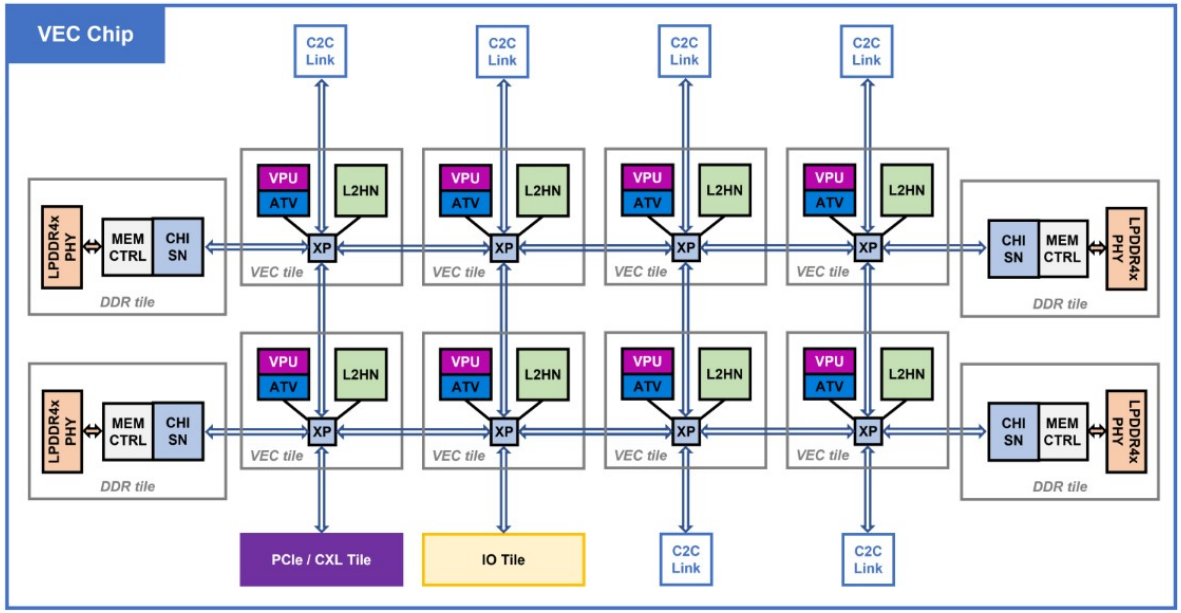
- FPGA equivalent of Chip prototype
  - Use the actual RTL
- Prepare SW for the actual chip
- Run bigger workloads than in RTL simulation
  - Linux boot
  - Real apps to characterize design
  - Unveil corner cases





EPAC-VEC FPGA SDV Challenges

- VEC design too big
  - 8 x VEC Tiles
  - 4 x DDR Tiles
  - C2C, PCIE, IO-Tile hosting Fabric Controller
- Hard-blocks / proprietary IPs not portable
  - PCIe, LPDDR4x PHY, FLL
- Challenges:
  - FPGA Area restrictions
  - Parts to port to maintain a working system
  - How to emulate boot sequence
  - Pre-silicon Validation



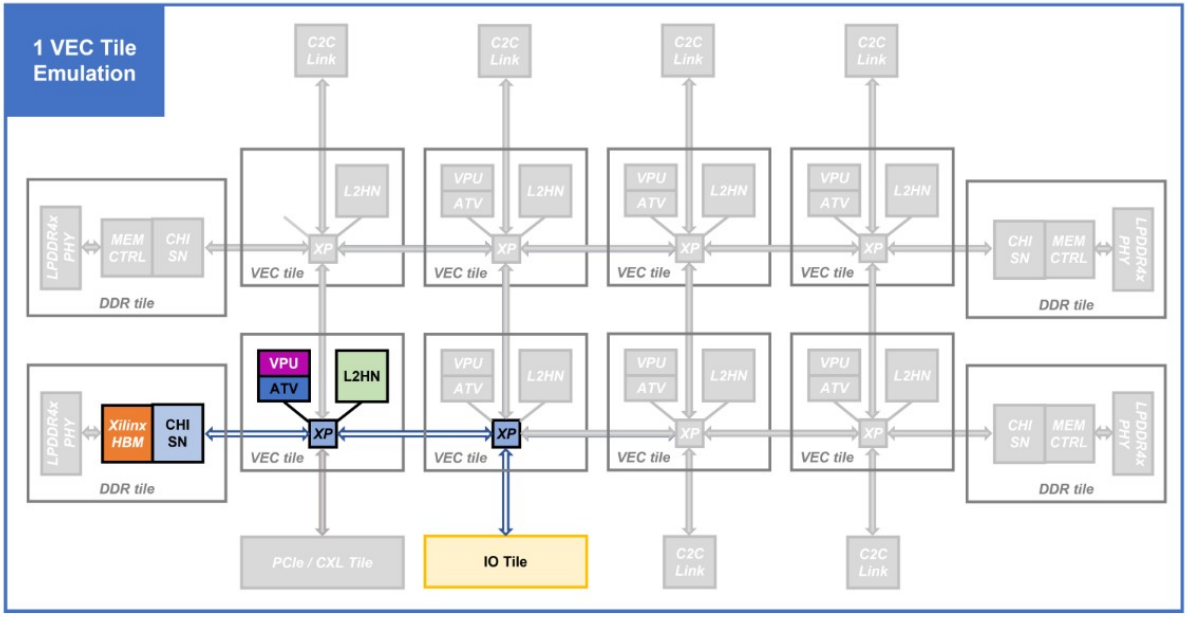
Single core FPGA SDV

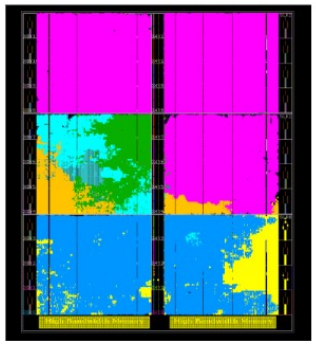
Minimal components for Linux boot:

- VEC Tile (AVS, VPU, L2HN, XP)
- IO Tile (JTAG, DM, PLIC, CLINT, UART)
- Modified DDR Tile
  - Remove LPDDR4x PHY + MC
  - Replace with HBM Hard-Block

FPGA Glue Logic to enable:

- Preloading firmware in memories
- Executing boot sequence



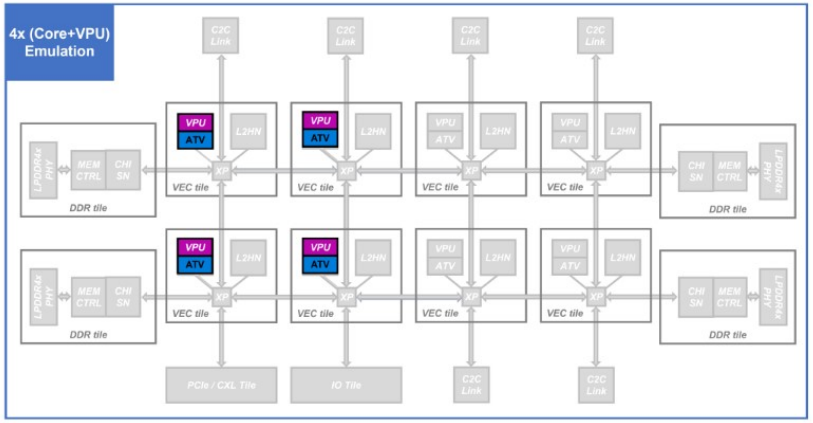


- VPU [~36% LUTs]
- ATV [~21% LUTs]
- L2HN [~4% LUTs]
- XP [~2.5% LUTs]
- IO tile [~3% LUTs]
- FPGA glue [~4.5% LUTs]
- **Total [>70% LUTs]**

Multi core FPGA SDV

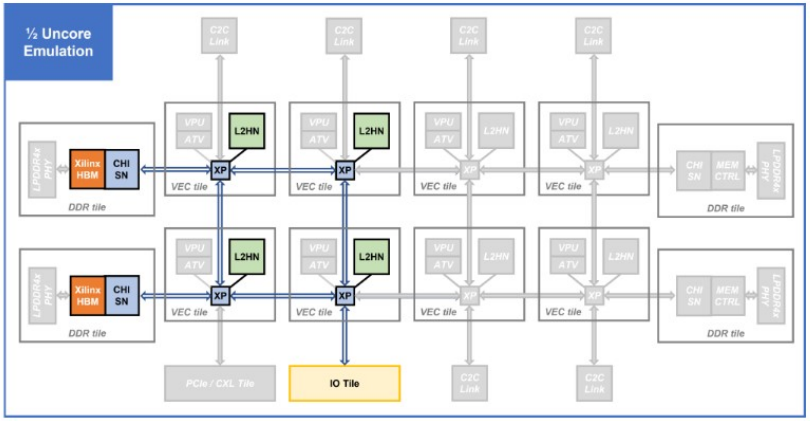
- **Partition design** to **multiple FPGAs** due to area restrictions

- Limiting factor: AVS + VPU size
  - **Variant A**: **up to 4 AVS** in one FPGA
  - **Variant B**: **1 AVS+VPU** in one FPGA



- **Half-sized Uncore** in one FPGA **for both variants**

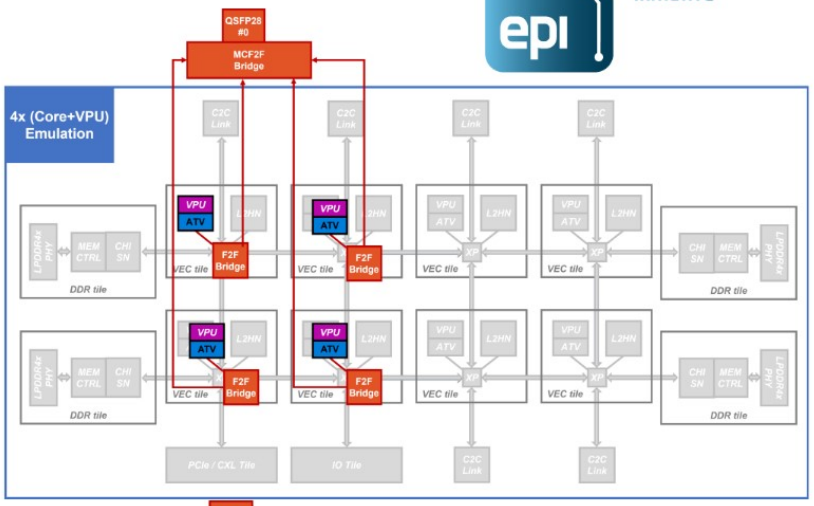
- Custom **FPGA2FPGA bridge**
  - For transferring CHI,AXI, interrupts across FPGAs



Multi core FPGA SDV

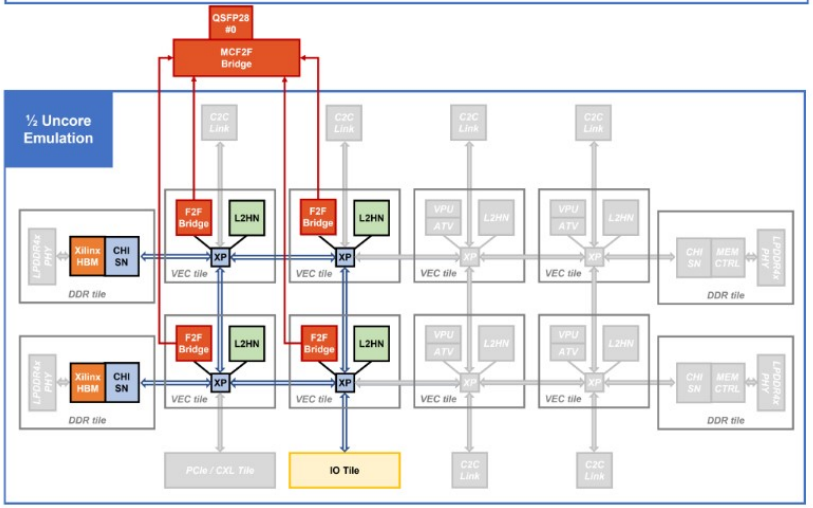
- **Partition design** to **multiple FPGAs** due to area restrictions

- Limiting factor: AVS + VPU size
  - **Variant A**: **up to 4 AVS** in one FPGA
  - **Variant B**: **1 AVS+VPU** in one FPGA



- **Half-sized Uncore** in one FPGA **for both variants**

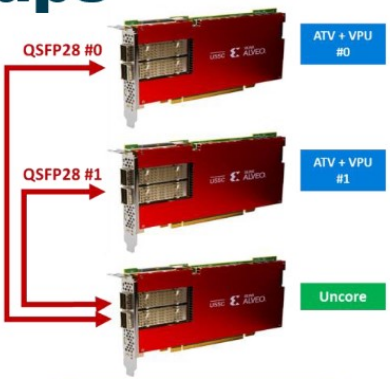
- Custom **FPGA2FPGA bridge**
  - For transferring CHI,AXI, interrupts across FPGAs



FPGA SDV setups

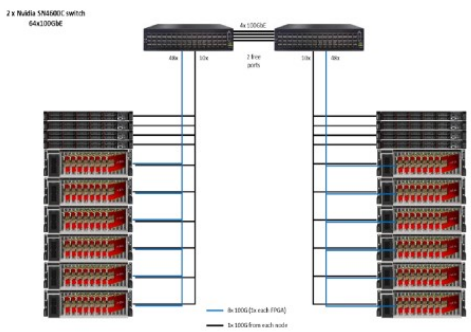
Xilinx
Alveo U55c

Single
Core + VPU



3x Alveo U55c
Interconnected via QSFP

Dual
Core + VPU
or
Quad
Core



Makinote
96x Alveo U55c

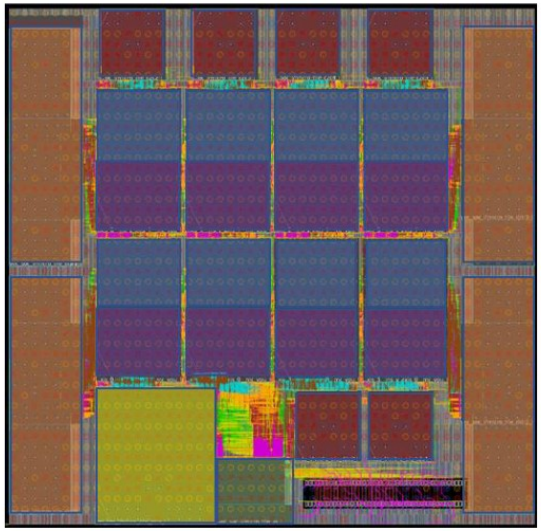
Multi-node
Single
Core + VPU

5x Xilinx VCU128
Interconnected via QSFP

Quad
Core + VPU

EUPILOT Tape Outs

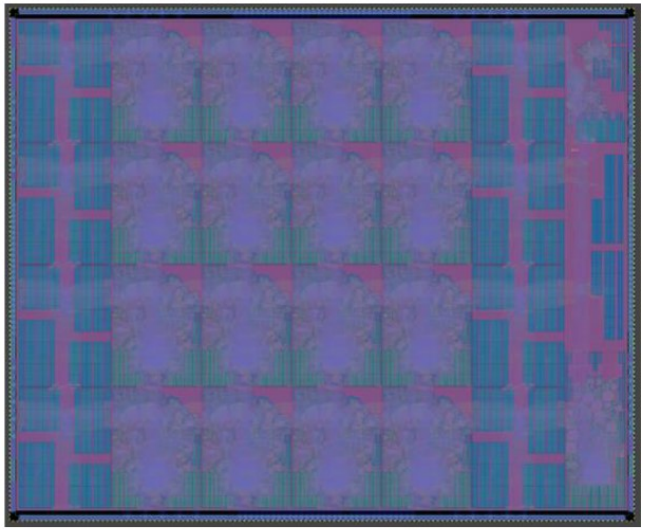
VEC



8 x OoO RV64 + wide VPU
targeting HPC

GF 12nm FinFET

MLS



Many core based on RV32 core
targeting ML and Stencils

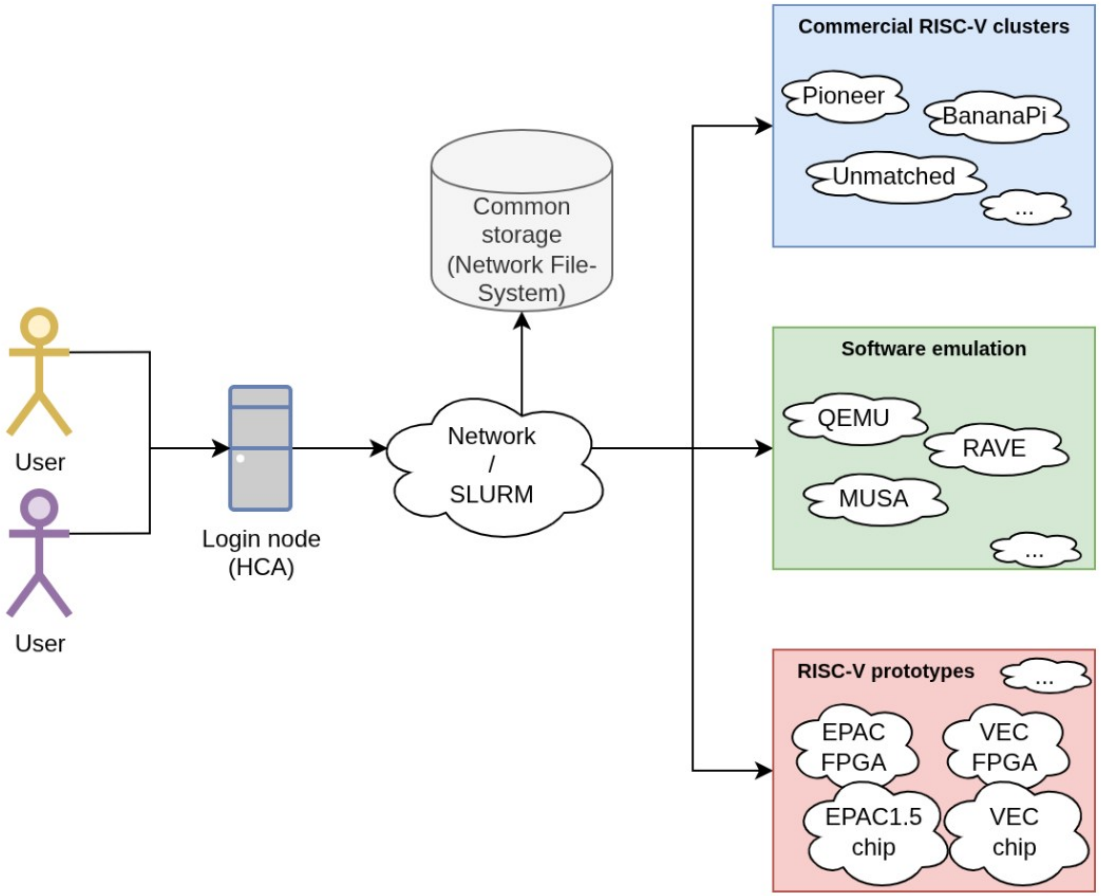
TSMC 7nm

European Processor Initiative

RISC-V EPAC in action

Filippo Mantovani, Computer Science Department, Barcelona Supercomputing Center (Spain)

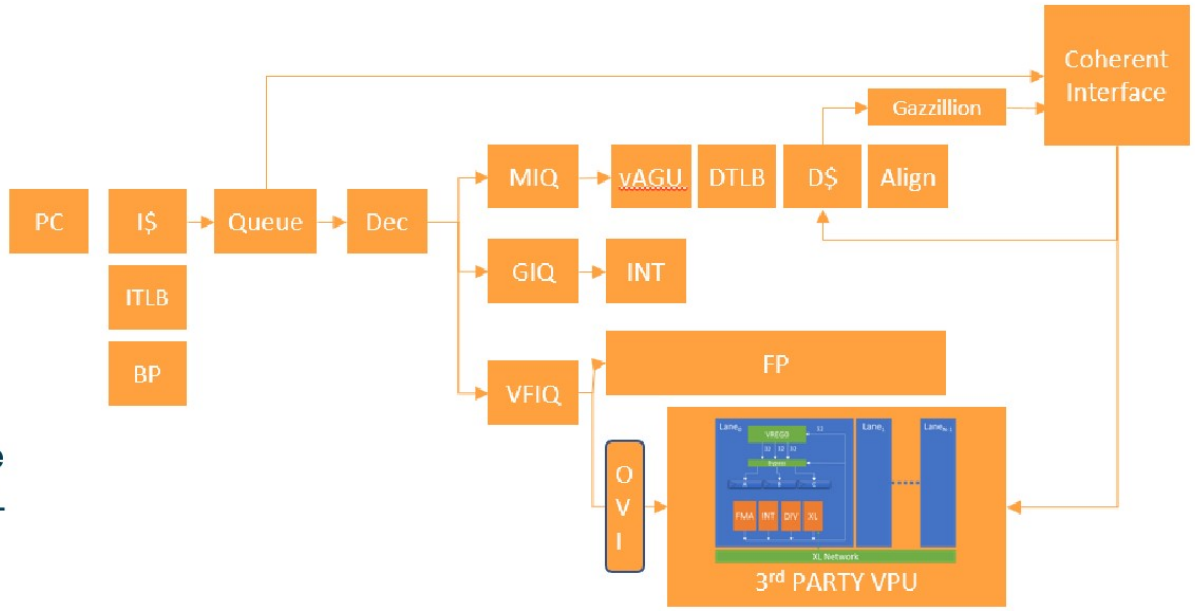
Integration in a data-center like environment



Scalar RISC-V core (Avispado)

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

  - 16 kB instruction cache
  - 32 kB L1 data cache
  - 1 MB L2 cache
  - Decodes RVV v0.7 vector extension
  - Cache coherent (CHI)
  - Vector memory accesses (vle, vlse, vlxe processed by a dedicated queue (MIQ/L



Courtesy: semidynamics silicon design and verification services

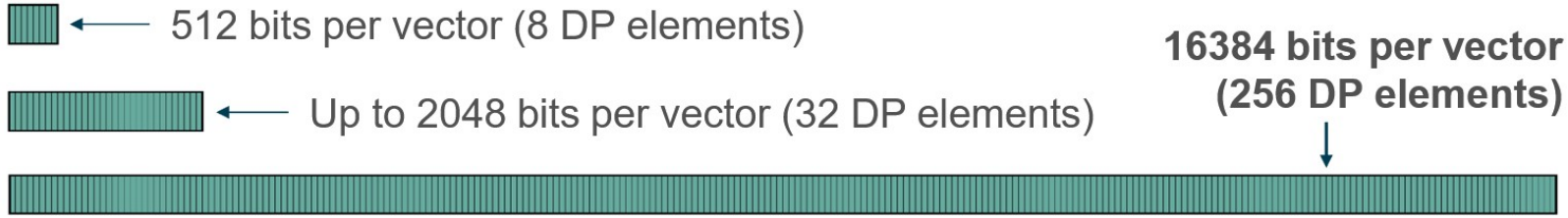
VPU with Long Vector Length (VL) support

intel – AVX512 ← 512 bits per vector (8 DP elements)

arm – SVE ← Up to 2048 bits per vector (32 DP elements)

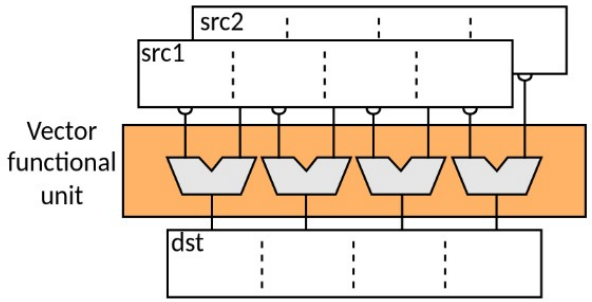
NEC / RISC-V

16384 bits per vector (256 DP elements)



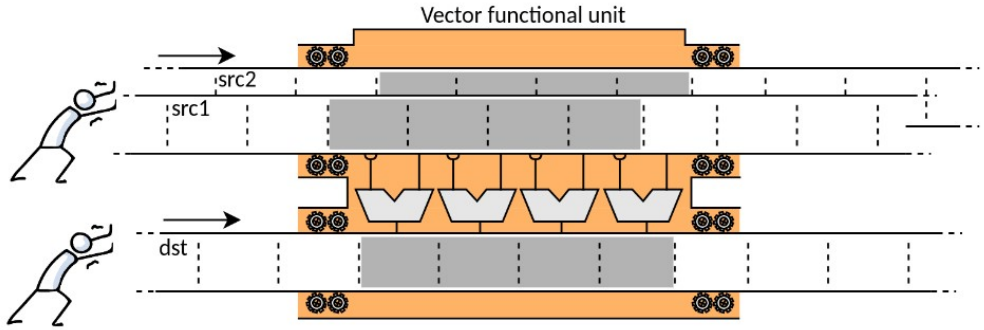
Short VL

- As many Functional Units as VL.
- Vector instructions executed in 1 cycle



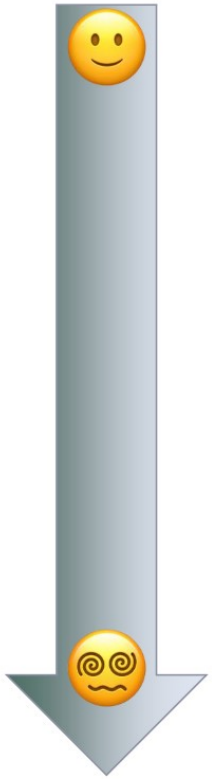
Long VL

- Cannot afford (area, power, cost) hundreds of Functional Units
- Vector instructions are executed on multiple cycles



F. Minervini, et al. "Vitruvius+: An Area-Efficient RISC-V Decoupled Vector Coprocessor for High Performance Computing Applications" [TACO-2022-50]

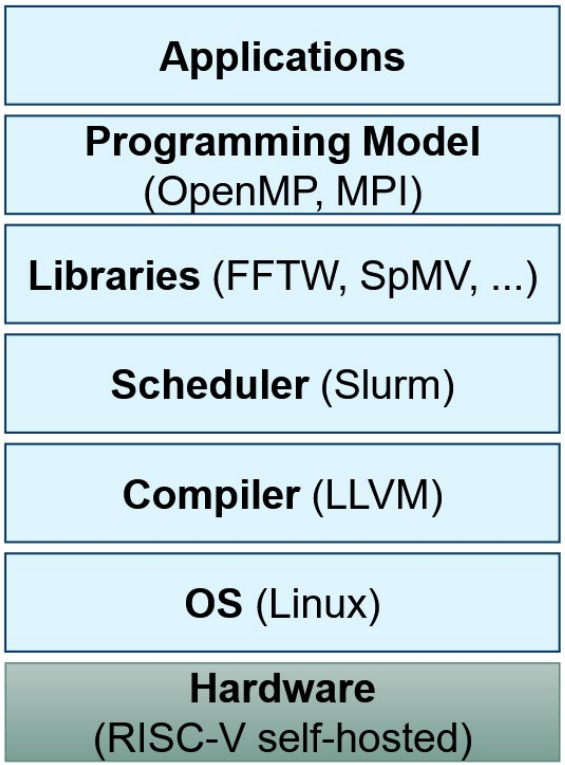
How do I program EPAC - VEC?



- Auto-vectorization
  - 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++/FORTRAN 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

How do I use EPAC - VEC?

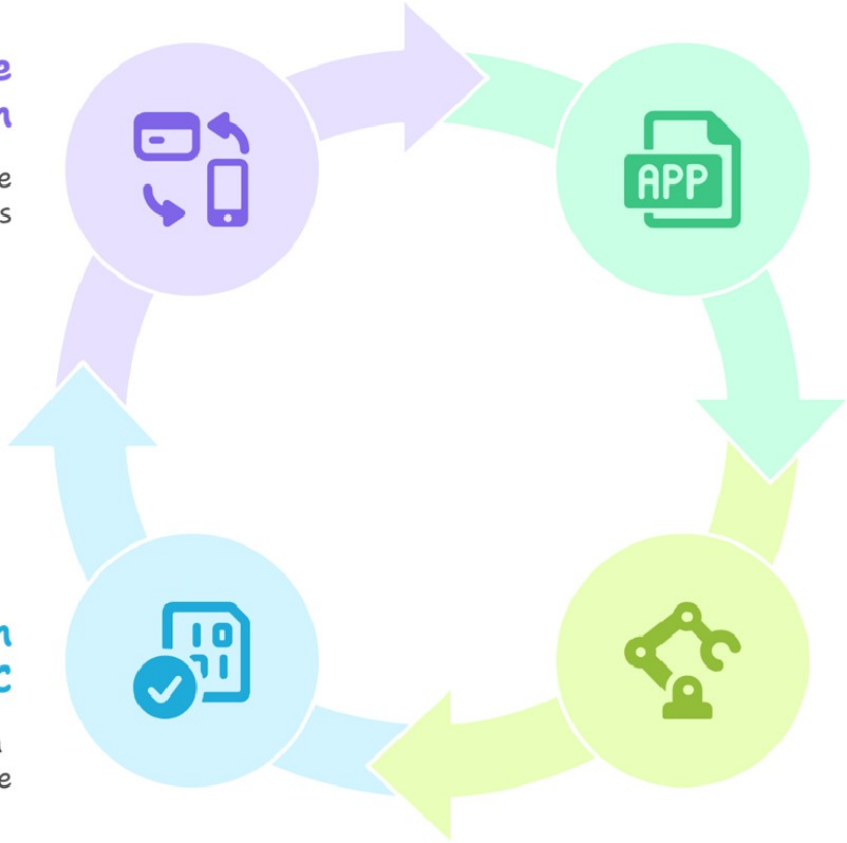
- Like a standard HPC system!
- Compile your code
  - We give you a compiler
- Link libraries
- Write/Submit a job script
  - SLURM
- Wait for the results
- Analyse execution traces and study how well your code is vectorized



Co-design methodology

Cross-Architecture Comparison

Comparing performance across architectures



App Preparation

Preparing and porting apps to RISC-V

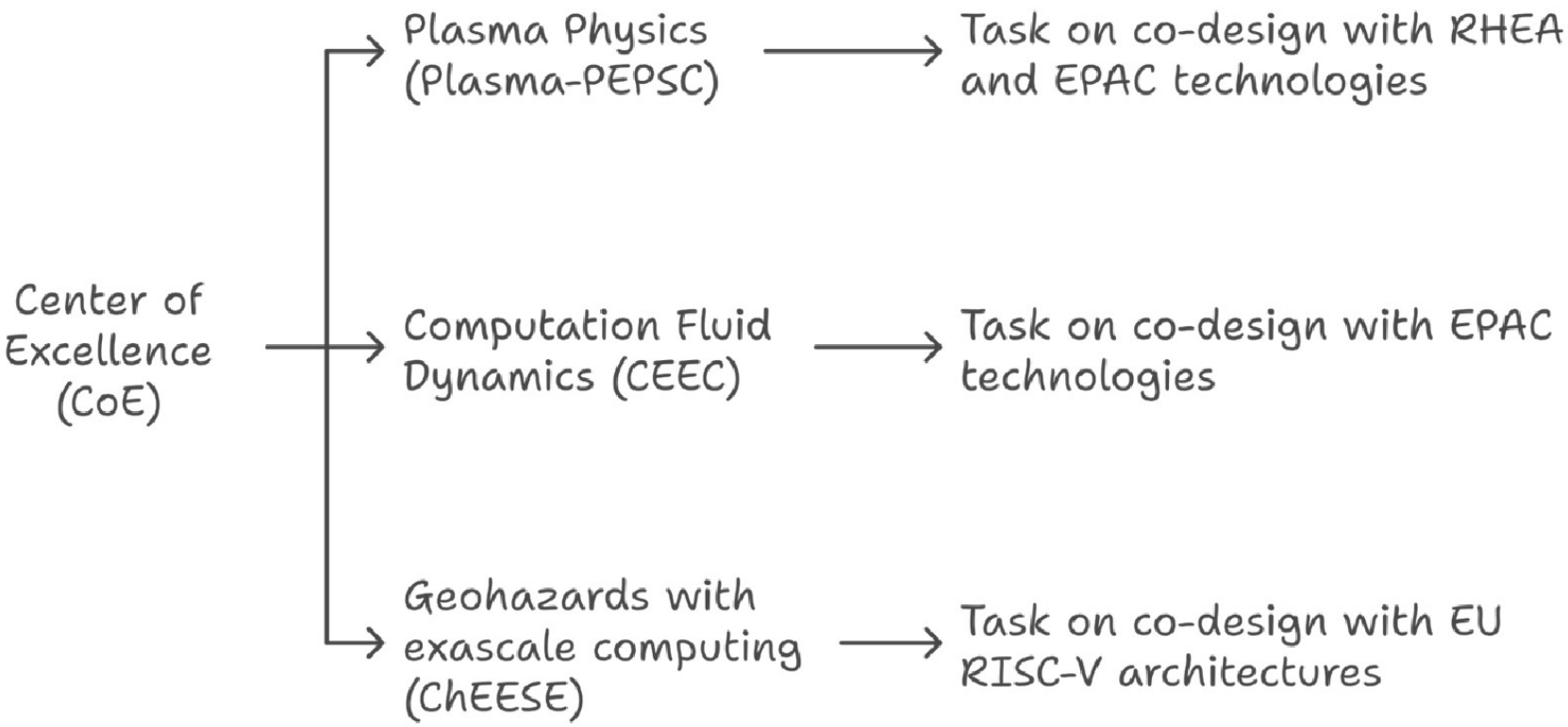
Auto-Vectorization

Optimizing code for long vectors using emulators

Execution on EPAC-VEC

Running code on EPAC-VEC hardware

Co-design methodology in action: CoEs



CEEC: Fluid dynamic

Application	1. App. preparation			2. Vector compilation & emulation			3. Execution on EPAC-VEC		4. Cross-architecture comparison
	Code	Input	Scalar RISC-V run	Compile w EPI-LLVM with auto-vectorization	Emulate (QEMU + RAVE)	Analyze & Optimize	Execute	Analyze & Optimize	
Alya (mini-app fluid matrix assembly)	✅	✅	✅	✅	✅	✅	✅	✅	✅
Alya (mini-app solid matrix assembly)	✅	✅	✅	✅	✅	✅	✅	✅	✅
FLEXI	✅	✅	✅	✅	✅	⚙️	✅		
WaLBerla	✅	✅	✅	✅	✅	✅	✅	✅	⚙️
SOD2D	✅	✅	✅	✅	✅	⚙️	✅	⚙️	
NEKO	✅	✅	✅	✅	⚙️				
NekRS	✅	✅	✅	🚧					

✅ Done ⚙️ Work in progres 🚧 Potential road block due to just in time compilation

Plasma-PEPSC: Plasma physics

Application	1-Scalar RISC-V	2-Emulated RISC-V Vector			3-FPGA RISC-V Vector (EPAC-VEC)		4-Study portability
	Compile & run on EPI clusters	Compile with LLVM/EPI compiler	Emulate (vehave / qemu)	Analyze & optimize	Execute	Analyze & optimize	Compare with other architectures
BIT1	✅	✅	✅	✅	✅	🏗	
GENE	✅	✅	✅	🏗	🚧 (*)		
GENE-X	✅	✅	✅	🚧 (**)	✅	🚧 (**)	
PIConGPU	✅	✅	✅	🏗			
Vlasiator	✅	✅	✅	✅	✅	✅	🏗

(*): flang compilation errors

(**): major refactor of the main operator

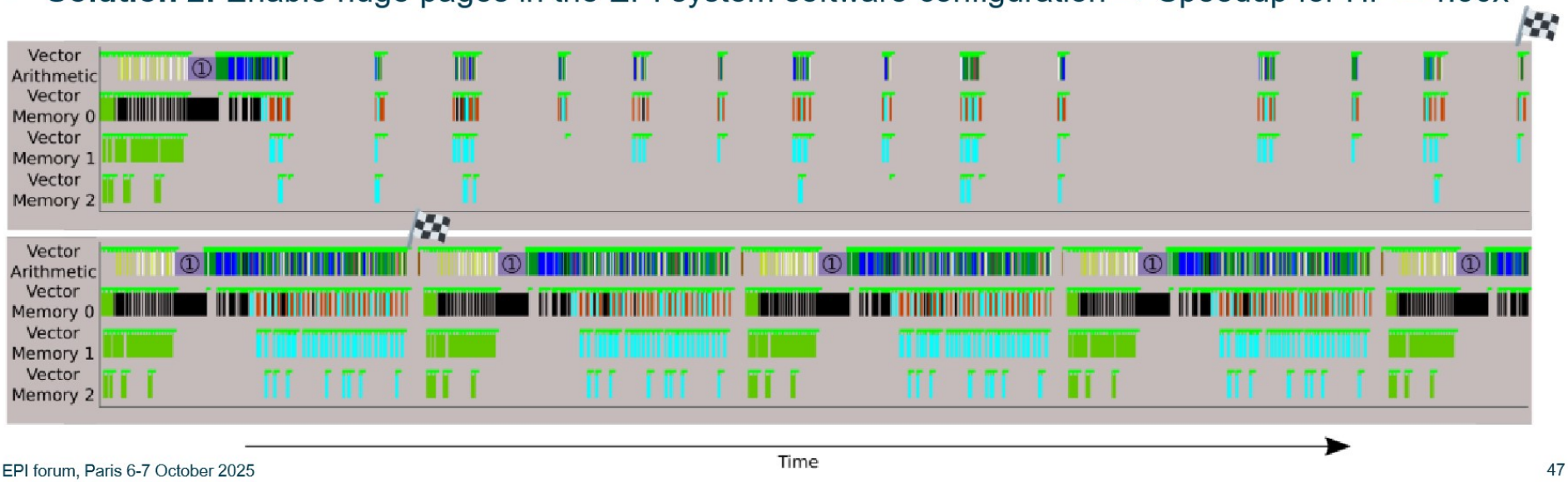
ChEESE: Earth Science

App	1-Scalar RISC-V	2-Emulated RISC-V Vector			3-FPGA RISC-V Vector (EPAC-VEC)		4-Arch comparison
	Compile & run	Compile (Vectorize)	Emulate	Analyze & optimize	Execute	Analyze & optimize	
SPECFEM3D	✅ (*)						
HySEA	✅ (*)	✅ (*)	✅ (*)	🏗️	🏗️		
SeisSol	✅	✅	✅	🏗️	✅	🏗️	🏗️
FALL3D	✅ (*)	✅ (*)	✅ (*)	✅ (*)	✅ (*)	✅ (*)	✅ (*)
Tandem							
PTatin3D							
Xshells	✅ (*)	✅ (*)	✅ (*)	🏗️	✅ (*)	🏗️	

(*): on mini-app

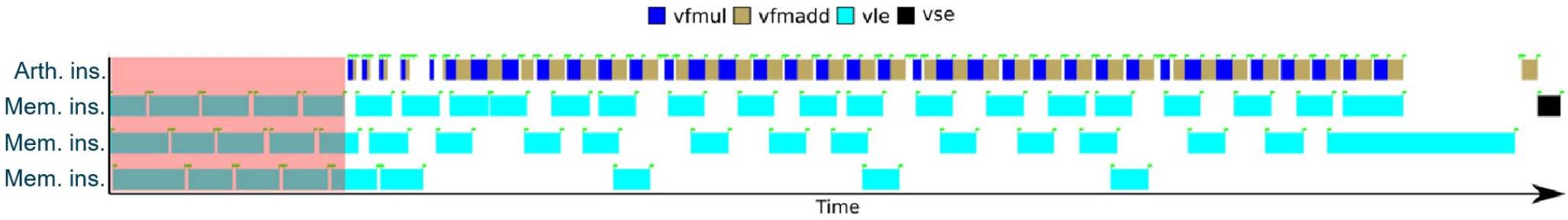
Highlight: feedback to the code owners

- Spying the pipeline of the vector processing unit of an iteration of WaLBerla we notice gaps.
- The data layout of the application is such that each set of particles to be processed are placed far apart in memory, forcing a page fault in the system.
- **Solution 1:** Modify the data layout in the code
- **Solution 2:** Enable huge pages in the EPI system software configuration → Speedup for HP = 4.56x

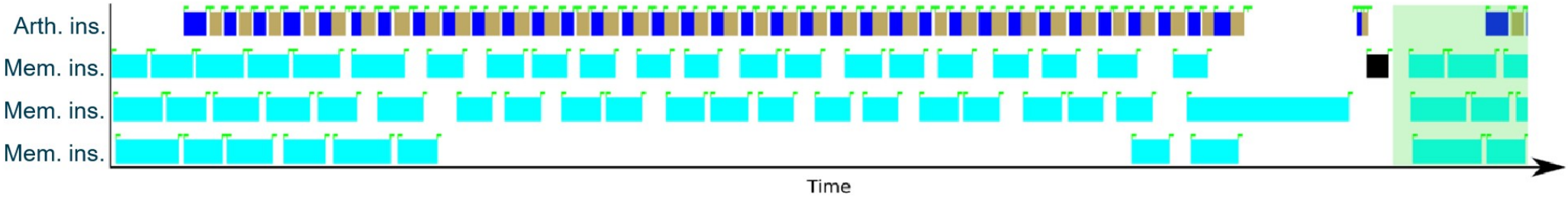


Highlights: feedback to the EPI compiler

- Looking at the instruction scheduling of a kernel of Alya (solidz).
- The default scheduling do not fully overlap independent arithmetic and memory vector instructions:



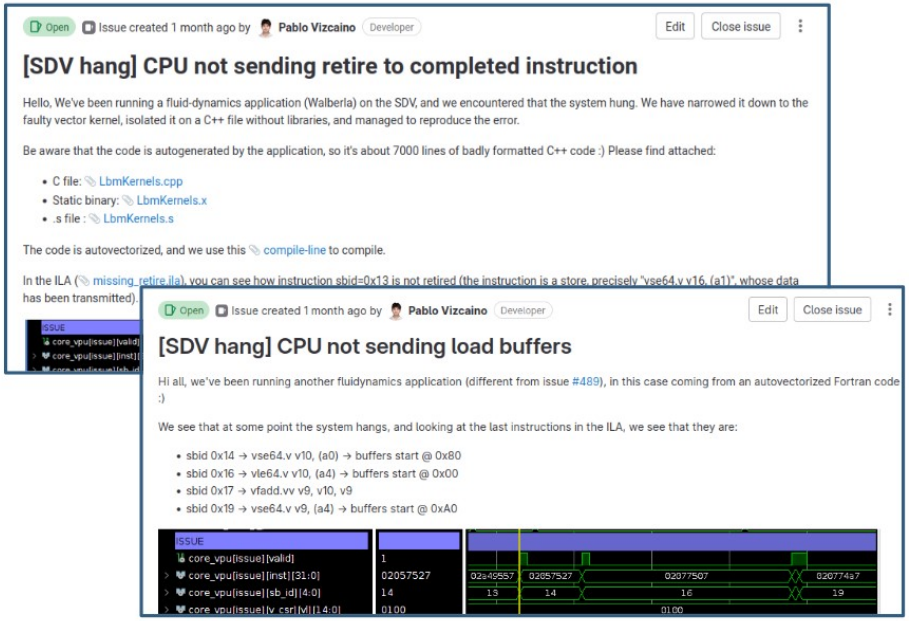
- Enabling a more interleaved scheduling allows for a speed-up of 1.15x



Highlight: feedback to the hardware developers

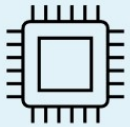
- Chip design can not be fixed after tapeout process has started
  - Before freezing the chip design heavy test is critical

- CEEC code fragments have been used to test RTL before freezing the design for tapeout
  - They served to isolate and solve at least two hardware malfunctioning



Growing impact on tools, vectorization analysis and co-design methodology

European Processor Initiative

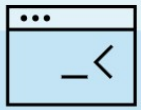


THE EUPILOT
RTL integration effort
SDV common infrastructure

dare
RTL integration effort
SDV common infrastructure
Compiler and libraries for RISC-V

Riser
Integration of EPAC technology for cloud solutions

higher
Prototyping RISC-V based hardware and software for cloud



pop
Performance analysis methods and code improvements for VEC

Multiscale
Porting and deployment of ESSI package manager

esiwace
Porting IFS dwarfs to RISC-V and studying their vectorization

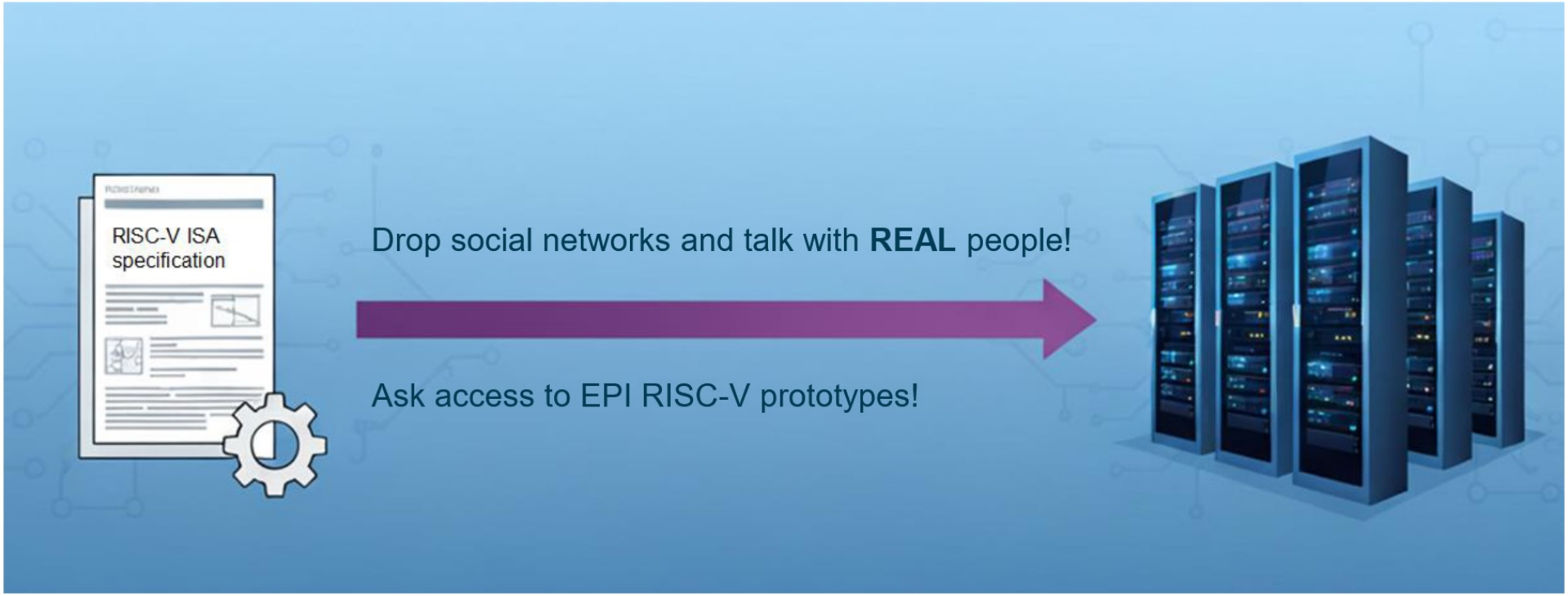
PLASMA PEPSC
Analysis of Vlasiator and BIT1

CEEC
Study of Alya and WaLBerla

ChEESE
Study of MiniFALL3D and Seissol

Are we arrived?

Challenge yourself: feel and experiment where we are!



References

- 📝 Mantovani, Filippo, et al. "Software Development Vehicles to enable extended and early co-design: a RISC-V and HPC case of study." International Conference on High Performance Computing. Cham: Springer Nature Switzerland, 2023. https://arxiv.org/abs/2306.01797
- 📝 Vizcaino, Pablo, et al. "Short reasons for long vectors in HPC CPUs: a study based on RISC-V." Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis. 2023. https://arxiv.org/abs/2309.06865
- 📝 Vizcaino, Pablo, et al. "RAVE: RISC-V Analyzer of Vector Executions, a QEMU tracing plugin." arXiv preprint arXiv:2409.13639 (2024). https://arxiv.org/abs/2409.13639
- 📝 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. https://arxiv.org/abs/2411.00815

- 📰 https://www.eetimes.com/examining-the-top-five-fallacies-about-risc-v/
- 🎞️ https://www.youtube.com/watch?v=iFIcJFcOJKk

Google Scholar

EPI PARTNERS

EPI FUNDING

This project has received funding from the European High Performance Computing Joint Undertaking (JU) under Framework Partnership Agreement No 800928 and Specific Grant Agreement No 101036168 EPI-SGA2.

The JU receives support from the European Union's Horizon 2020 research and innovation programme and from Croatia, France, Germany, Greece, Italy, Netherlands, Portugal, Spain, Sweden, and Switzerland.

SPONSORED BY THE

With funding from the:

Federal Ministry of Research, Technology and Space

REPUBLIC OF CROATIA Ministry of Science and Education

The EPI-SGA2 project, PCI2022-132935 is also co-funded by MICIU/AEI /10.13039/501100011033 and by the UE NextGenerationEU/PRTR