Software Development Vehicles to enable extended and early co-design:

a RISC-V and HPC case of study

Filippo Mantovani*, Pablo Vizcaino, Fabio Banchelli, Marta Garcia-Gasulla, Roger Ferrer, Jesus Labarta
Barcelona Supercomputing Center (BSC)

Giorgos Ieronymakis, Nikos Dimou, Vassilis Papaefstathiou
FORTH-ICS (Greece)

Context

European Processor Initiative

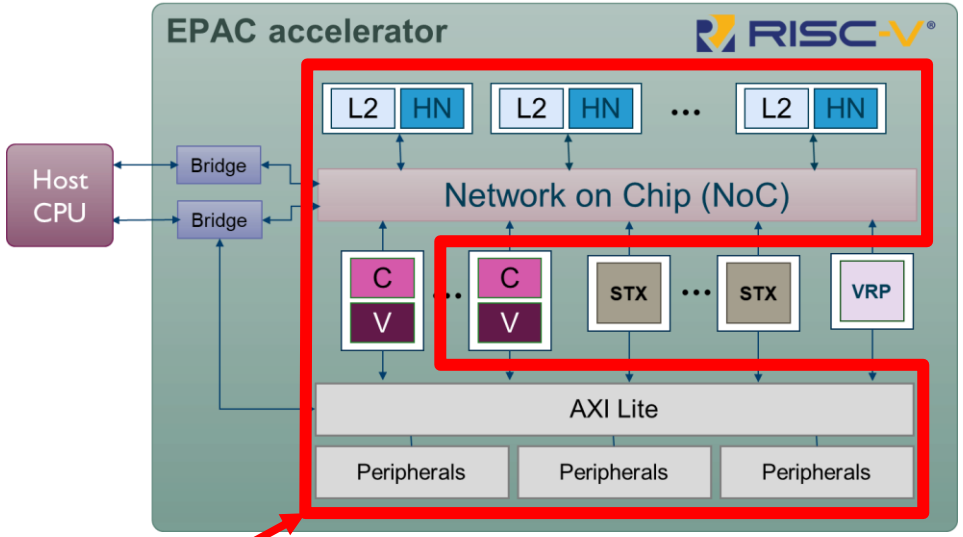
General Purpose Processor (RHEA – Arm based)

EPI Accelerators (EPAC – RISC-V based)

Out of the scope of this talk

Contact points:
SiPearl, Atos, CEA,
Univ. of Bologna, E4,
Univ. of Pisa, et al.

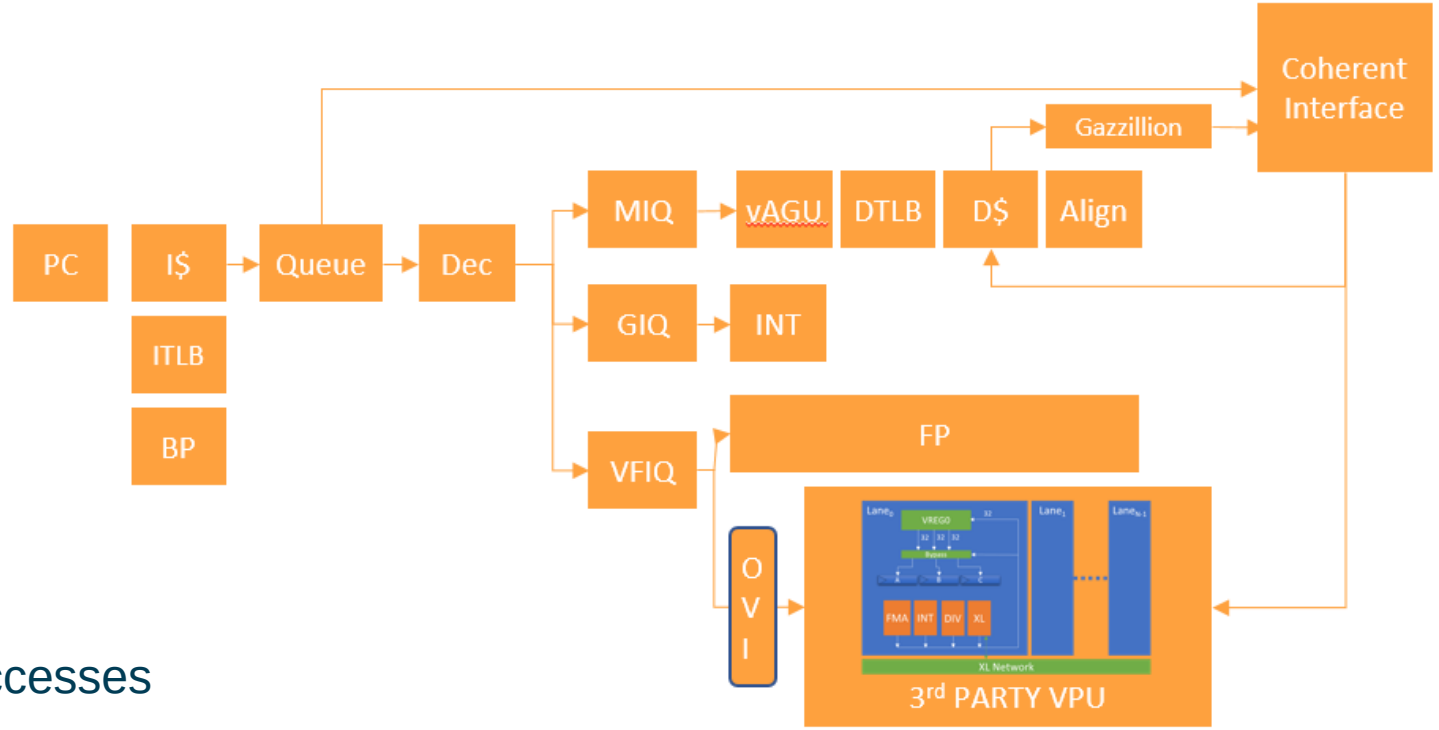
https://www.european-processor-initiative.eu/



Focus of this talk!

EPAC-VEC: RISC-V core “Avispado”

riscv64gcv



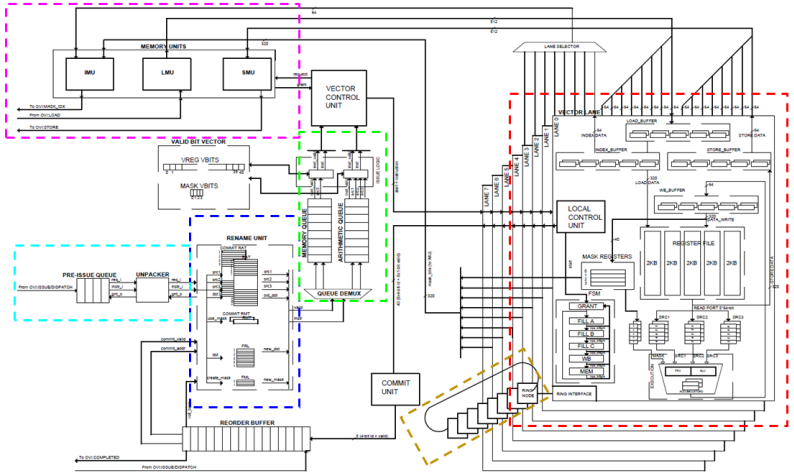
- 16 kB instruction cache
- 32 kB data cache
- Decodes v0.7, v1.0 vector extension
- Full hardware support for unaligned accesses
- Cache coherent (CHI)
- Vector memory accesses (vle, vlse, vlxe, vse, …) processed by a dedicated queue (MIQ/LSU)

Courtesy: semidynamics silicon design and verification services

EPAC-VEC: Vector processing unit “Vitruvius”

Architecture	Vector register size (1 cell = 1 double element)																	
Intel AVX512	D1	D2	D3	D4	D5	D6	D7	D8										
Arm Neon	D1	D2																
Arm SVE @ A64FX	D1	D2	D3	D4	D5	D6	D7	D8										
NEC Aurora SX	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15	D16	...	D256
RISC-V EPAC Vec	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15	D16	...	D256

- Implementation
  - Long vectors: 256 DP elements
    - #Functional Units (FUs) << Vector Length (VL)
    - 1 vector instruction can take several (32) cycles
  - 8 Lanes per core
    - FMA/lane: 2 DP Flop/cycle
  - 40 physical registers, some out of order
  - Vector length agnostic (VLA) programming and architecture

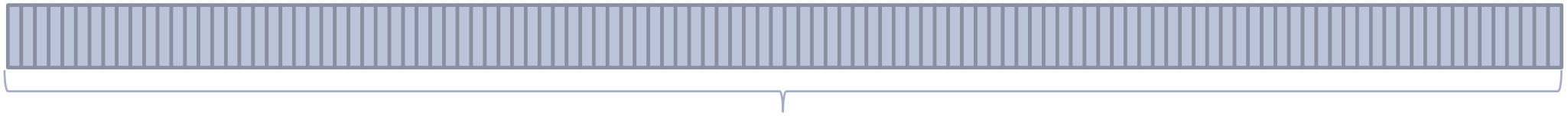


F. Minervini, et al. “Vitruvius+: An Area-Efficient RISC-V Decoupled Vector Coprocessor for High Performance Computing Applications” [TACO-2022-50]

How wide is “your” vector?

Intel AVX 512

RISC-V EPAC VEC

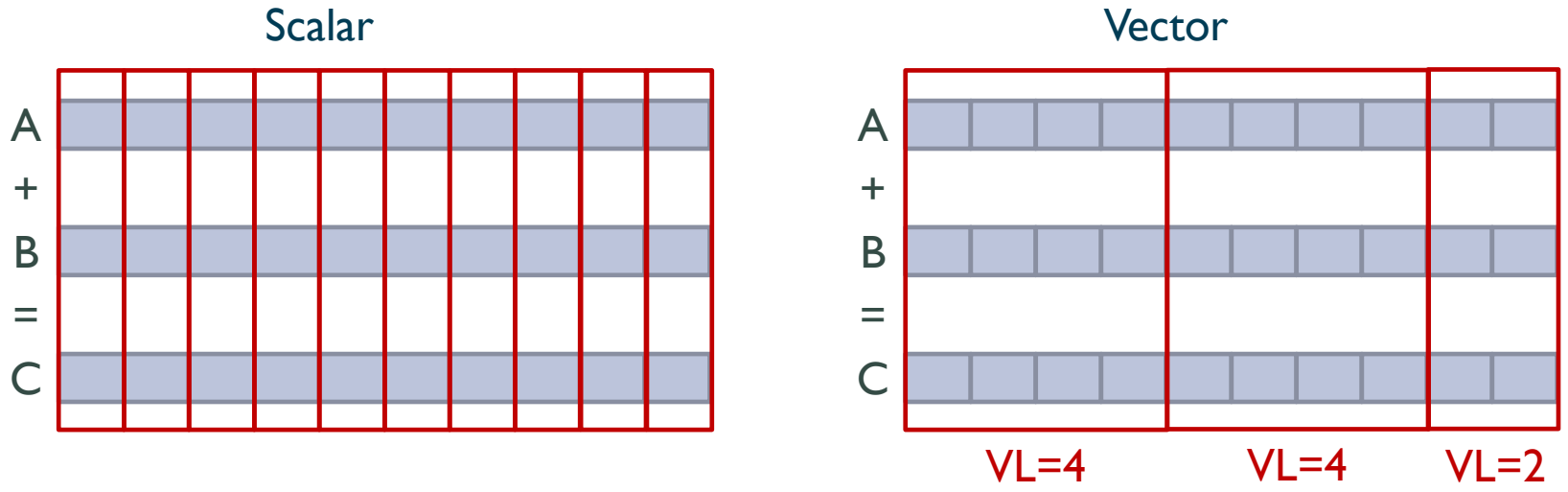


256 DP elements

16 kbits

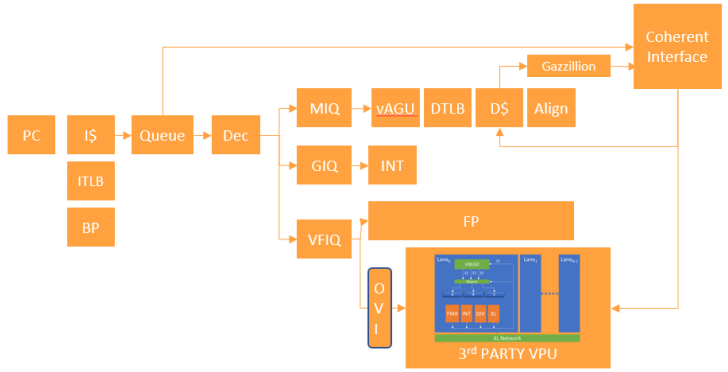
What's special?

- 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)



What's special? – part 2

- Preserve a linear, scalar, portable program
  - No need to think about "kernels to offload to the accelerator"
- Vector instructions
  - Less instructions (including scalar instructions for controlling a loop)
- Several cycles for a single vector instruction
  - Enables overlap: other functional units can do useful work meanwhile
  - Makes easier to keep all functional units busy
- Vector accelerator
  - Launch a vec. Instruction ~= Launch a kernel (in GPU terms)
  - i.e., a few cycle for decoding vs. several cycles for firing up a thread
- Coalescing on load instructions
  - Compared to scalar flow, pay overhead of load instruction start-up only once
  - Saturate the memory bandwidth with less cores

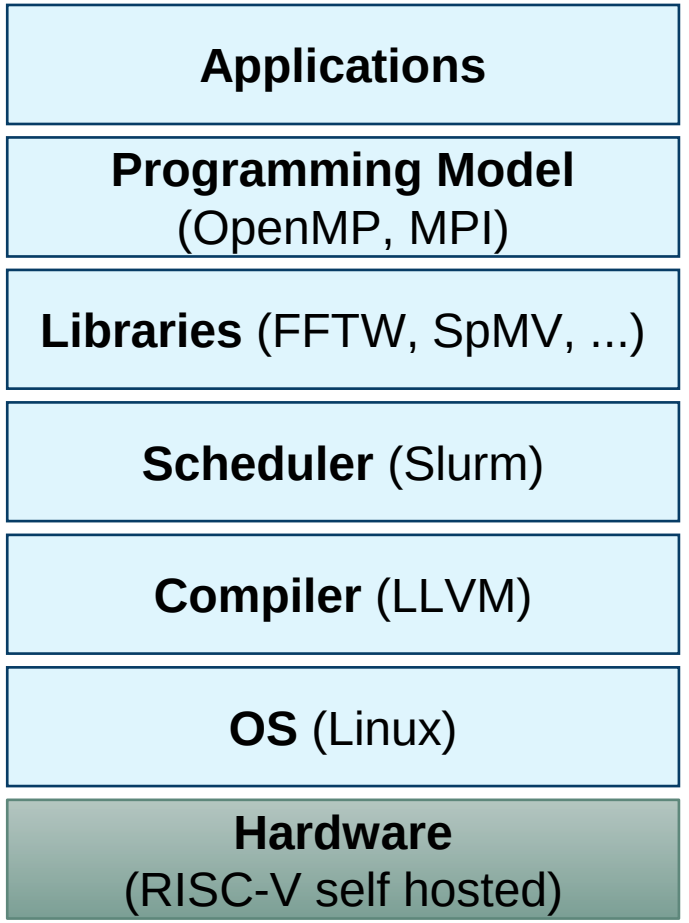


We have tools to measure each of these effects

How do I program it?

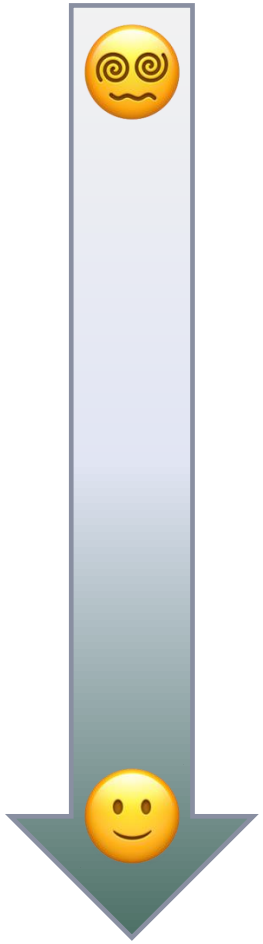
Like a standard HPC system!

- 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



How to take advantage of the V-extension?

- **Assembler**
  - Always a valid option but not the most pleasant
- **C/C++ builtins (intrinsics)**
  - Low-level mapping to the instructions
  - Allows embedding it into an existing C/C++ codebase
  - Allows relatively quick experimentation
- **#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
- **Autovectorization**
  - Leave it to the compiler



What do we do while hardware becomes ready?

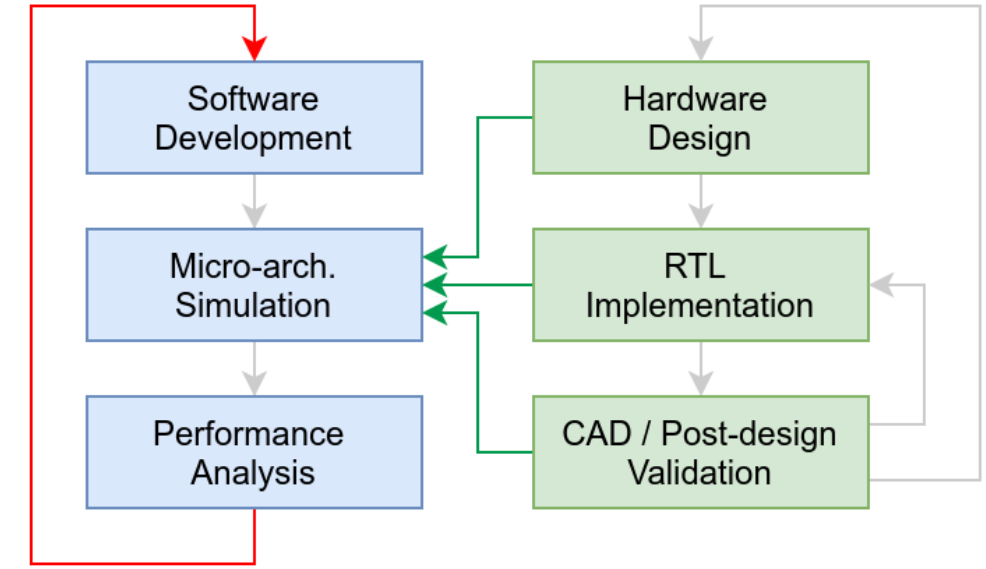
Co-design!!!

What do you mean by co-design?

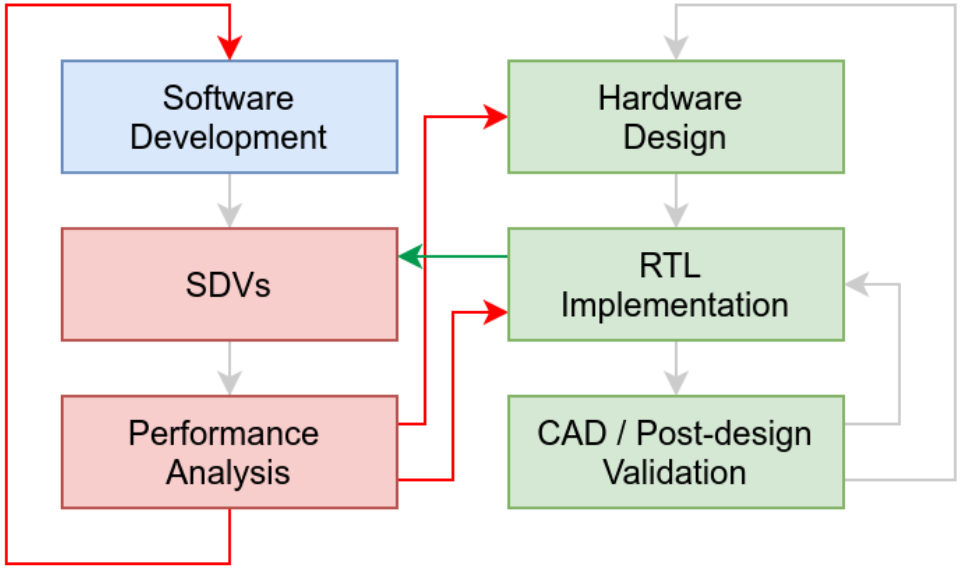
Co-design with EPAC-VEC

- Influence design decisions
  - Architecture definition / implementation
  - System software (e.g., compiler, libraries)
  - Scientific applications

SDV: Software Development Vehicles



A.

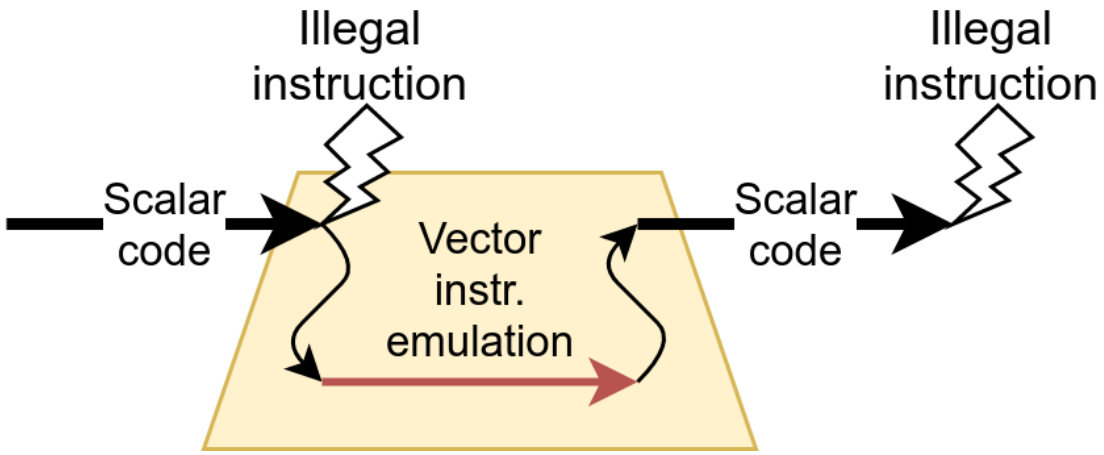


B.

Software Development Vehicles (SDV)

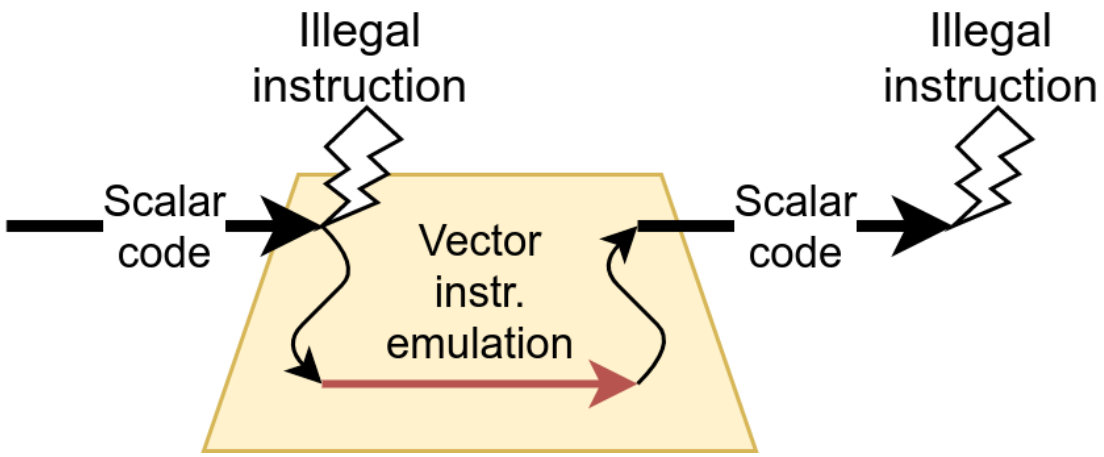
Software emulator: Vehave

- Compile using the RISC-V Vector extension (RVV) Compiler
- Obtain a binary with vector instructions
- Run in a commercial RISC-V platform (scalar CPU)
- Obtain a trace with detailed information about the vectorization



Commercial RISC-V platform (scalar CPU)

Software emulator: Vehave



PROS:

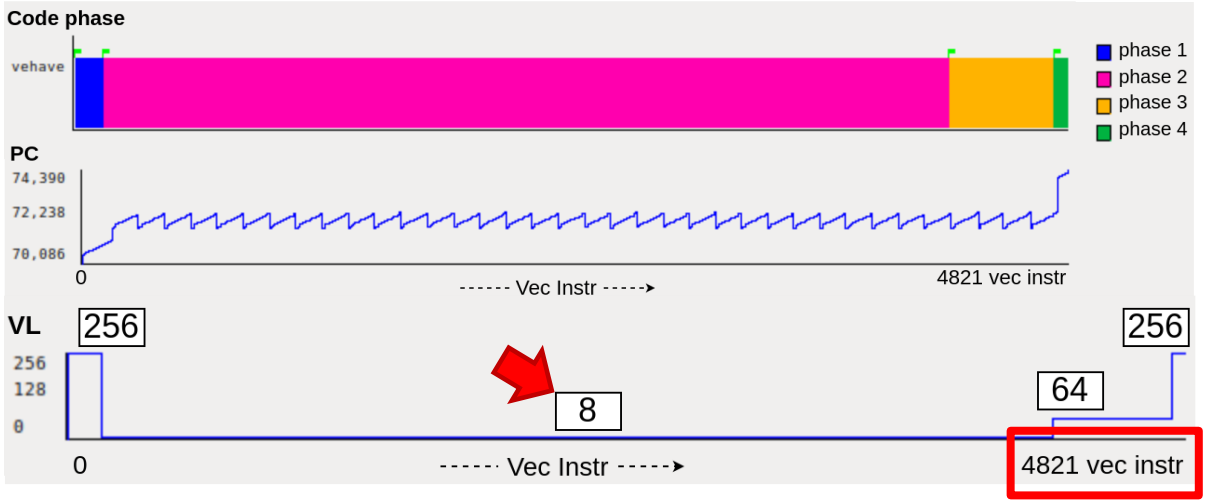
- Useful to understand the potential vectorization of the code
- Easy to use and accessible with no need of hardware infrastructure
- It supports RVV-0.7 and RVV-1.0
- Output compatible with Paraver

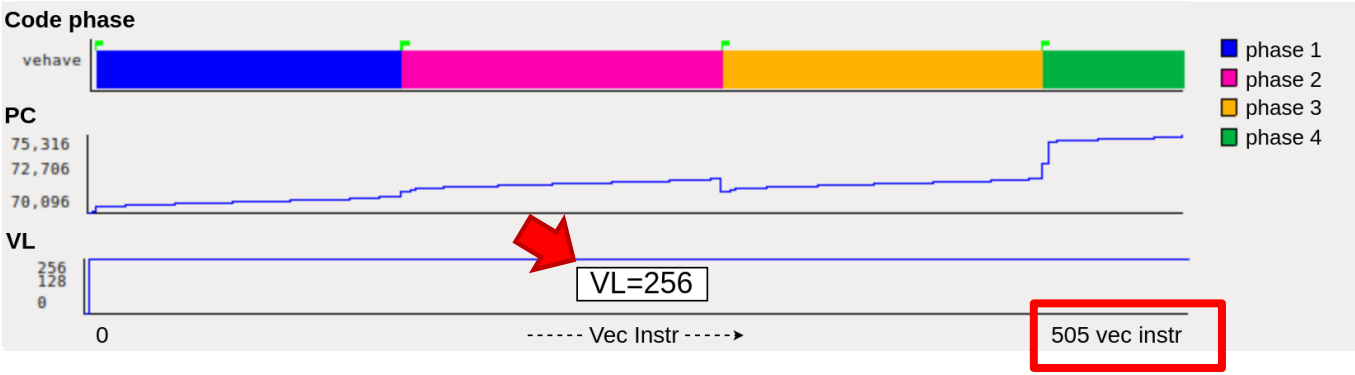
CONS:

- Slow
- No information about performance (no timing)

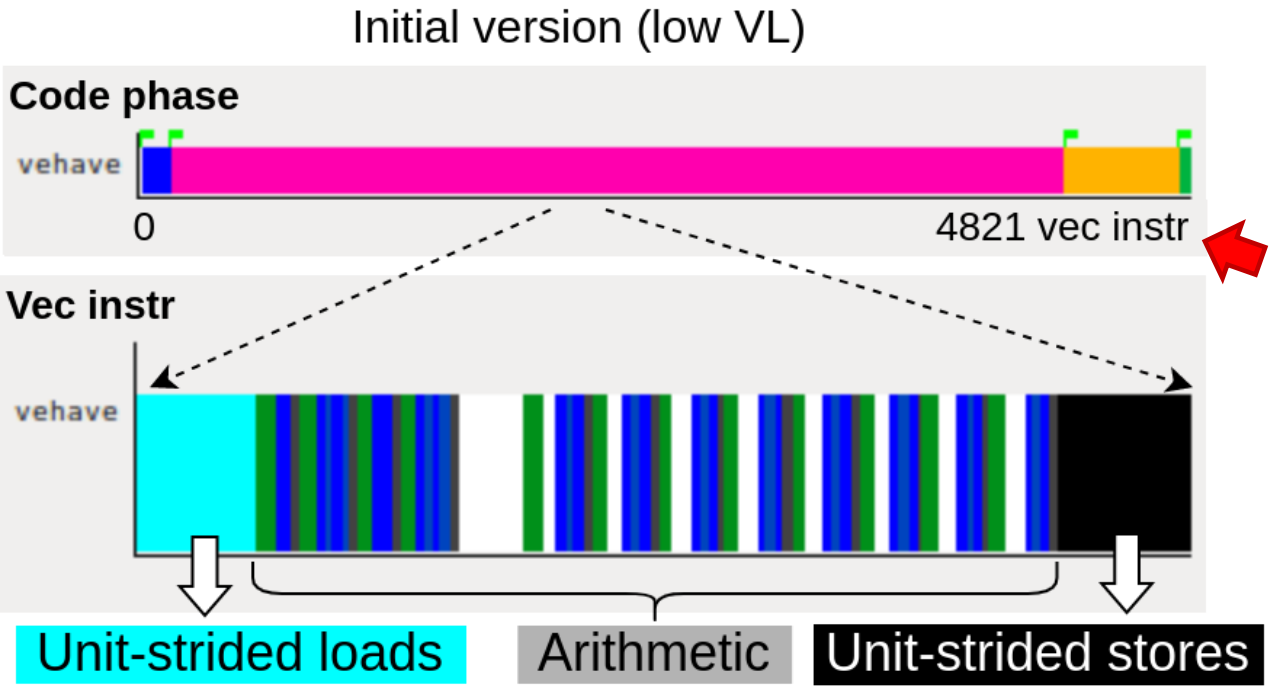
1st step: study with Vehave

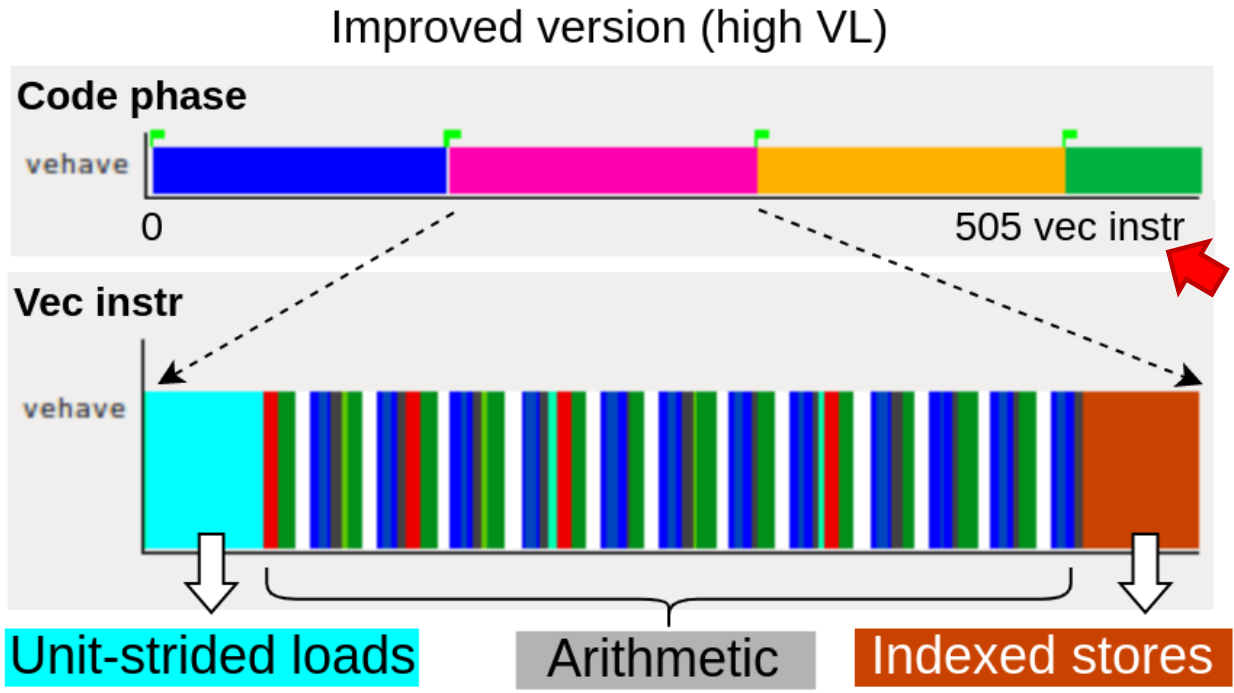
- Compile an application
  - Relying on autovectorization, with intrinsics, pragmas or assembly
  - Study the output of the compiler
- Run with emulation enabled
  - Collect execution traces
- Visualize and study traces
  - Is the code vectorized?
  - Which kind/how many vector instructions?
  - Which vector length is used?
  - Is there a way to write a “vector friendly” code?
  - Can the compiler “do better”?





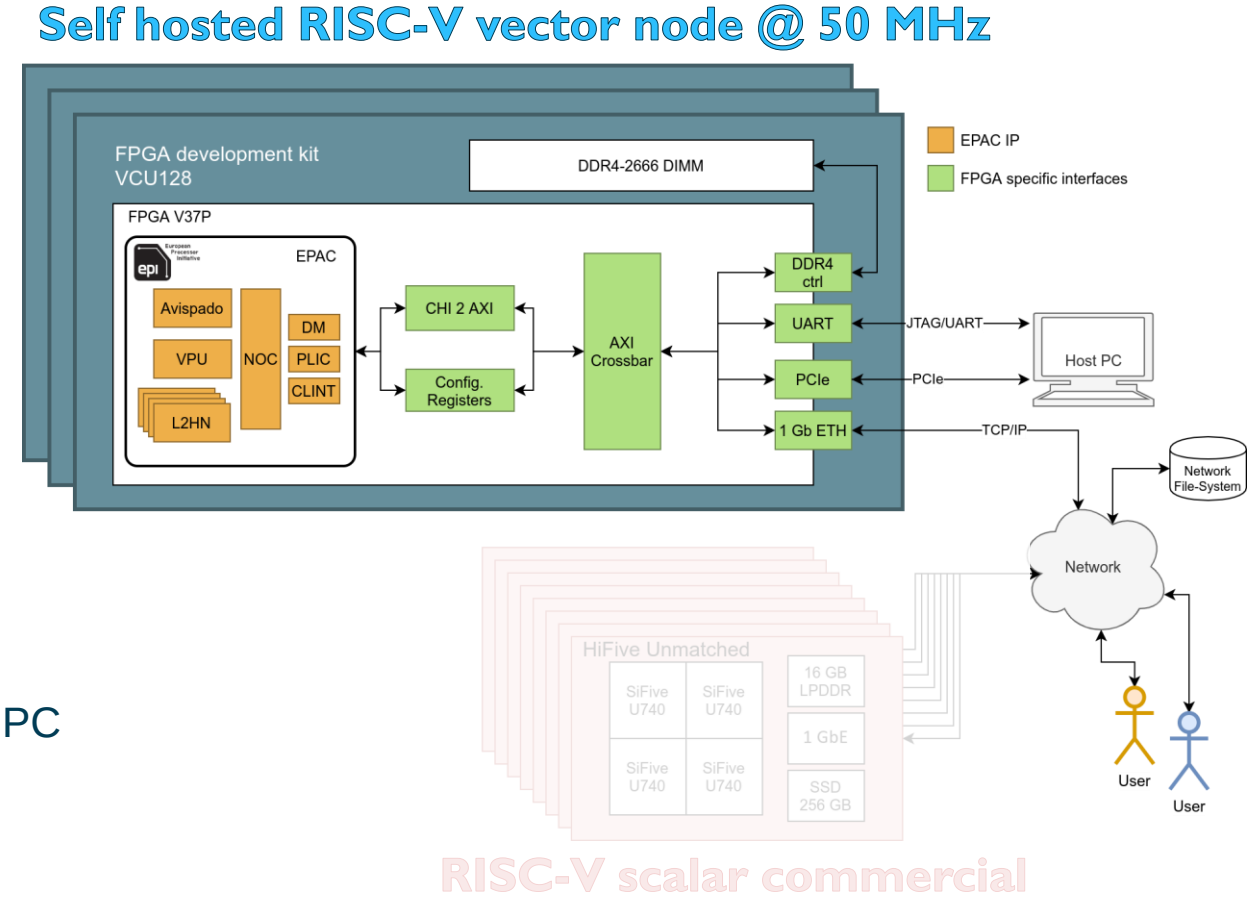
1st step: study with Vehave





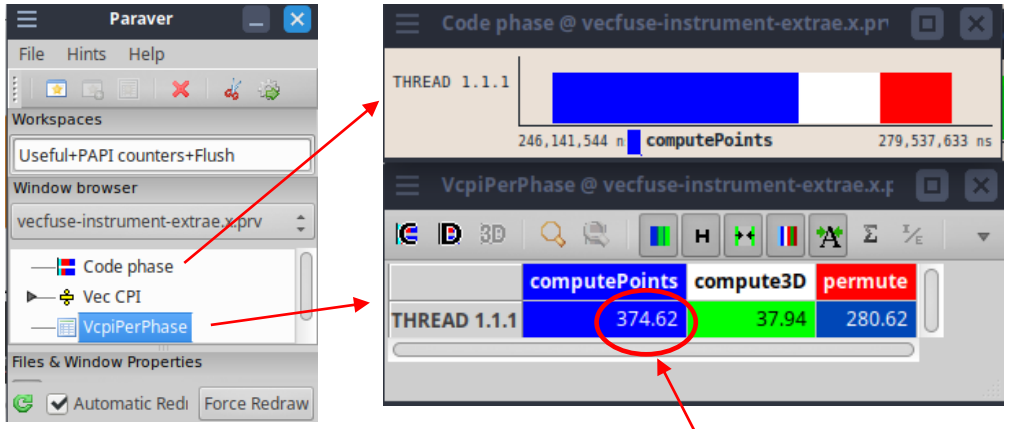
FPGA-based Software Development Vehicles

- Same RTL of EPAC mapped into FPGA
  - One tile (i.e., single core)
  - Running at 50 MHz
- Full HPC software stack and execution environment
  - Binary compatibility
  - Shared storage (NFS)
  - Multi-user / Multi-node (via MPI)
  - Standard debug and performance analysis tools for HPC



2nd STEP: Study on FPGA-SDV

- Compile an application
  - Relying on autovectorization, with intrinsics, pragmas or assembly
  - Study the output of the compiler
- Run natively on real hardware
  - Full support for I/O, syscalls, hw counter, etc
  - Collect execution traces
- Visualize and study traces
  - Which “vector CPI” do we achieve?
  - What are the most time-consuming phases?
  - How are we accessing the memory?
  - Can we overlap computation and memory accesses?



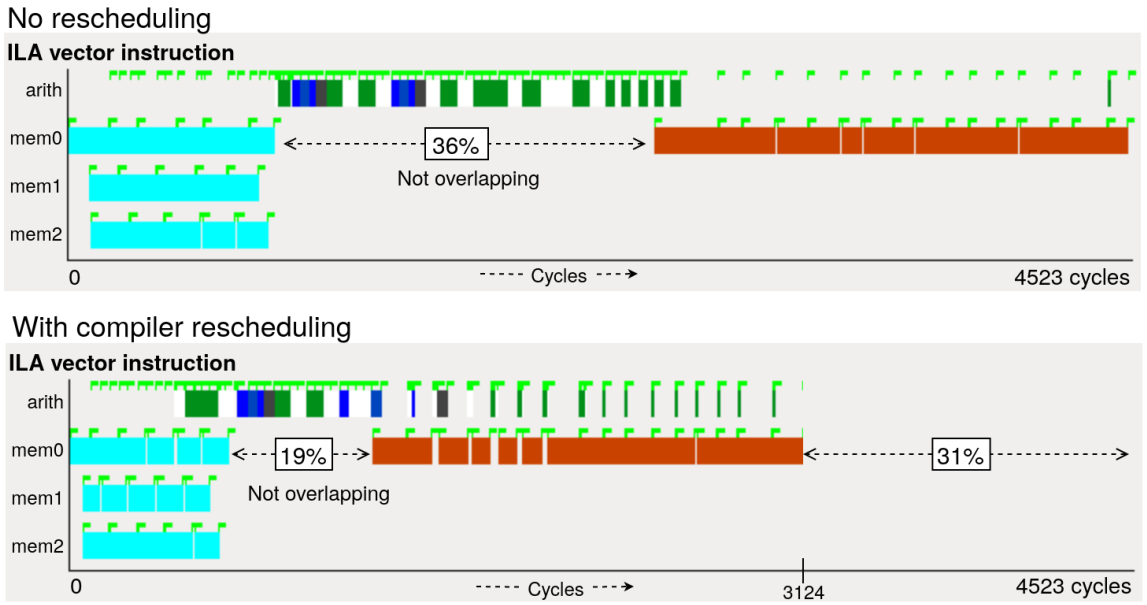
$$vCPI = \frac{\text{cycles VPU works}}{\text{\#vector instructions}}$$

fastest vCPI with vl=256 is 35 cycles

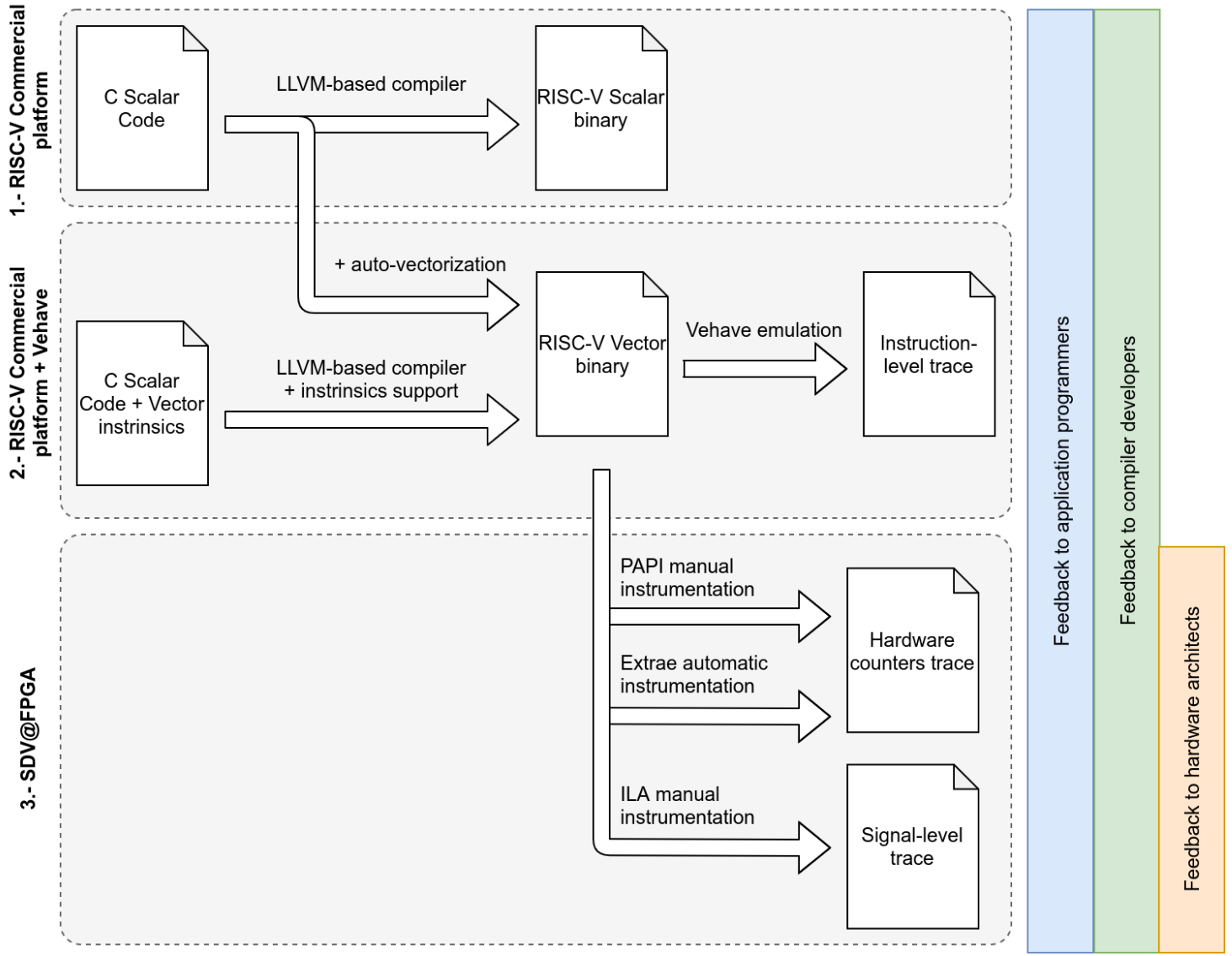
computePoints uses slow instructions
(374 cycles per vector instruction, on average!)

3rd step: signal analysis on FPGA-SDV Integrated Logic Analyser

- Fine grained analysis (at level of instructions) is possible
- Graphical representation of timelines
- In depth study can help highlighting:
  1. Low usage of the vector unit
     - Feedback to the **code developer**
  2. Suboptimal saturation or resources (FU, mem)
     - Feedback to the **RTL implementation team**
  3. Suboptimal overlap of instructions
     - Feedback to the **compiler team** (improve scheduling)



Co-design with FPGA-SDV



We can leverage SDVs to:

- Influence hardware design
- Improve compiler autovectorization and system-software support
- Study and prepare codes and libraries for long-vector architectures

References

- SDV tutorial: https://www.fz-juelich.de/en/ias/jsc/news/events/2023/ias-seminar-filippo-mantovani
- 📝 Gómez, Constantino, Filippo Mantovani, Erich Focht, and Marc Casas. "Efficiently running SpMV on long vector architectures." In Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 292-303. 2021.
- 📝 Vizcaino, Pablo, Filippo Mantovani, Roger Ferrer, and Jesus Labarta. "Acceleration with long vector architectures: Implementation and evaluation of the FFT kernel on NEC SX-Aurora and RISC-V vector extension." Concurrency and Computation: Practice and Experience (2022): e7424.
- 📰 https://www.eetimes.com/examining-the-top-five-fallacies-about-risc-v/
- 📽 https://www.youtube.com/watch?v=iFlcJFcOJKk

Acknowledgment and contacts

- Stream 3 leader and responsible for SDV:
  - Filippo Mantovani filippo.mantovani@bsc.es
- EPI General Manager:
  - Etienne Walter etienne.walter@atos.net

@EuProcessor

European Processor Initiative

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.