

RAVE: RISC-V Analyzer of Vector Executions

a QEMU tracing plugin

Pablo Vizcaino, Filippo Mantovani, Jesus Labarta, Roger Ferrer

HiPEAC - Workshop: RISC-V: the cornerstone ISA for the next generation of HPC infrastructures

Barcelona, January 21st 2025



Outline

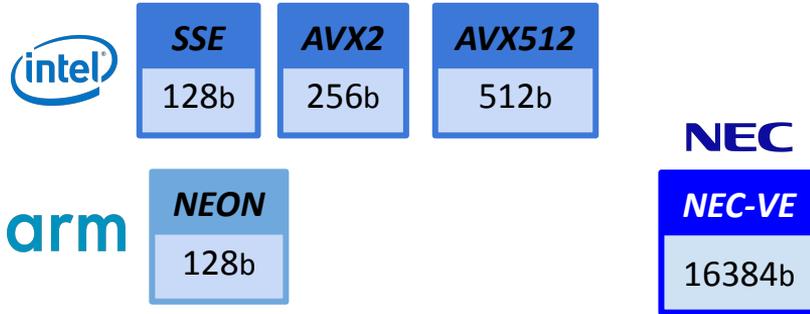
- Background on RVV
- EPAC chip
- Emulation environment
 - Why do we need **RAVE**?
 - How does **RAVE** work?
- Use cases

Motivation

- **RVV** (RISC-V Vector extension) is RV's bet for High Performance Computing

Motivation

- **RVV** (RISC-V Vector extension) is RV's bet for High Performance Computing



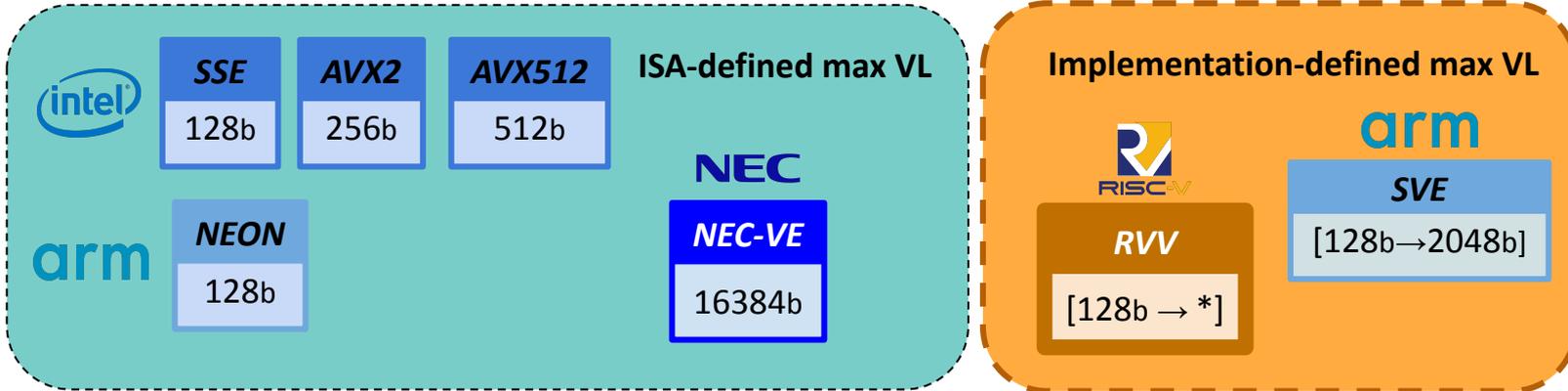
Motivation

- **RVV** (RISC-V Vector extension) is RV's bet for High Performance Computing



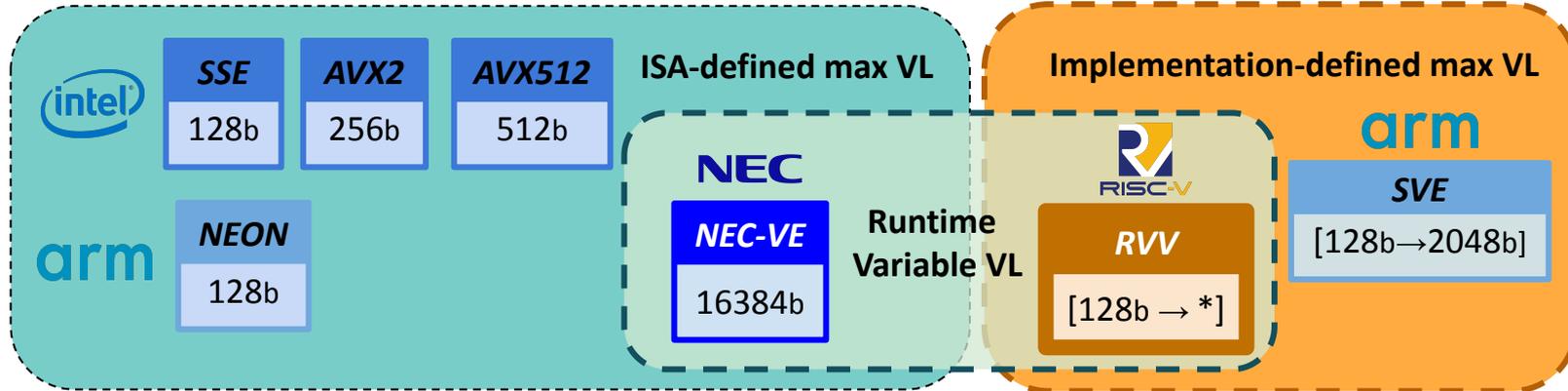
Motivation

- **RVV** (RISC-V Vector extension) is RV's bet for High Performance Computing



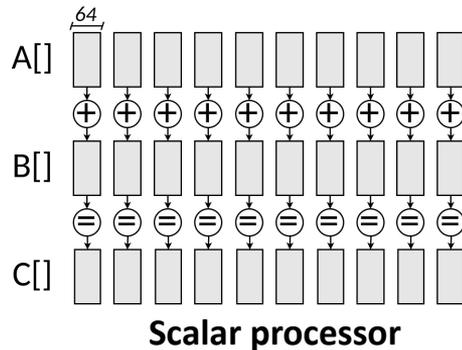
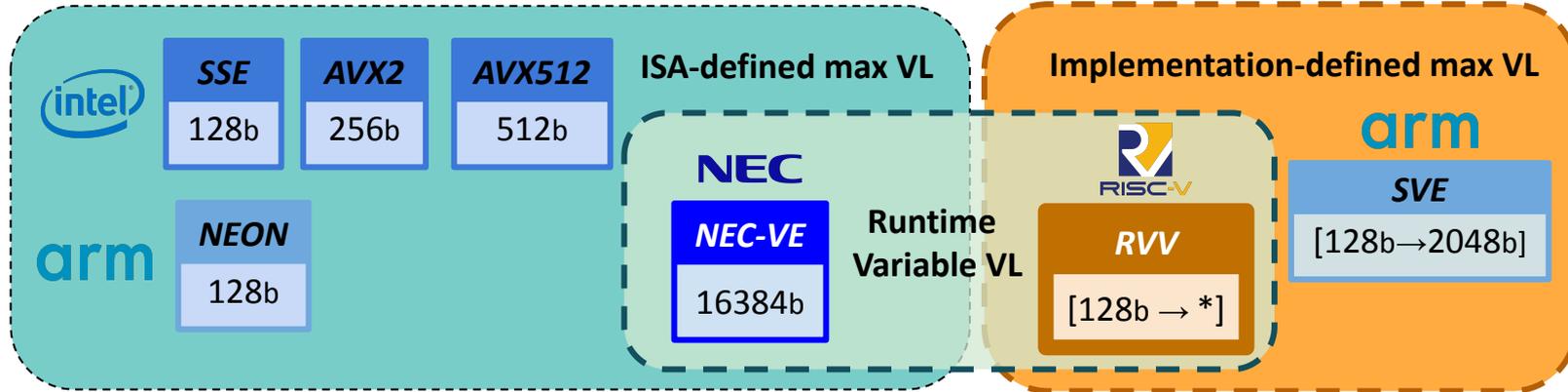
Motivation

- **RVV** (RISC-V Vector extension) is RV's bet for High Performance Computing



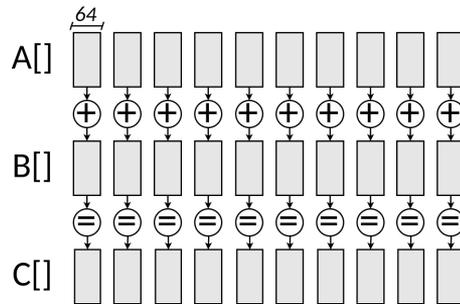
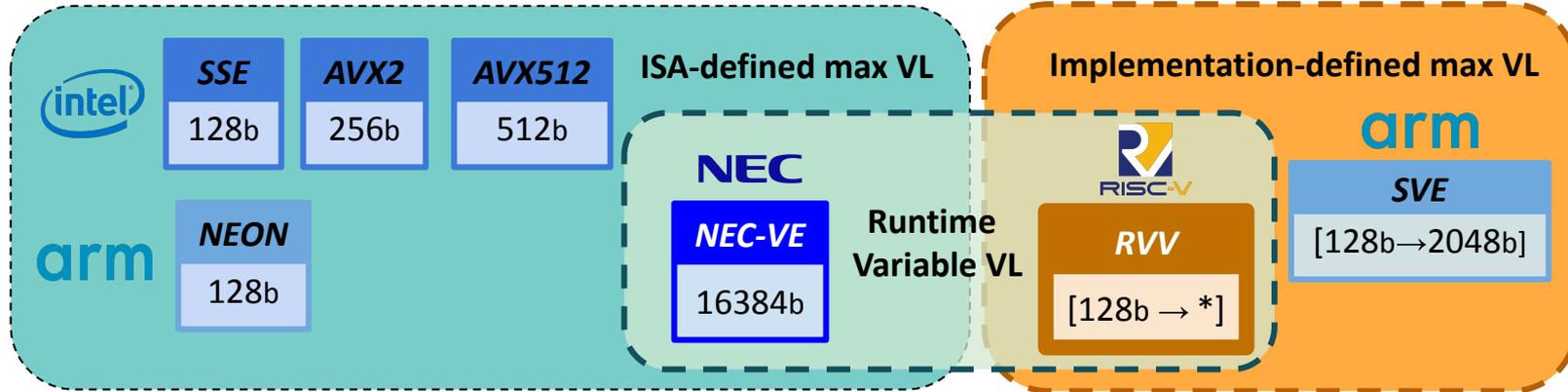
Motivation

- **RVV** (RISC-V Vector extension) is RV's bet for High Performance Computing

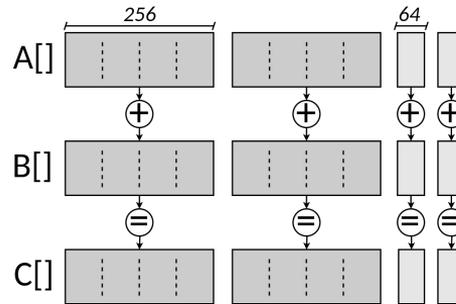


Motivation

- **RVV** (RISC-V Vector extension) is RV's bet for High Performance Computing



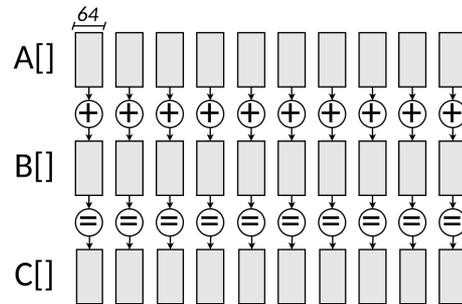
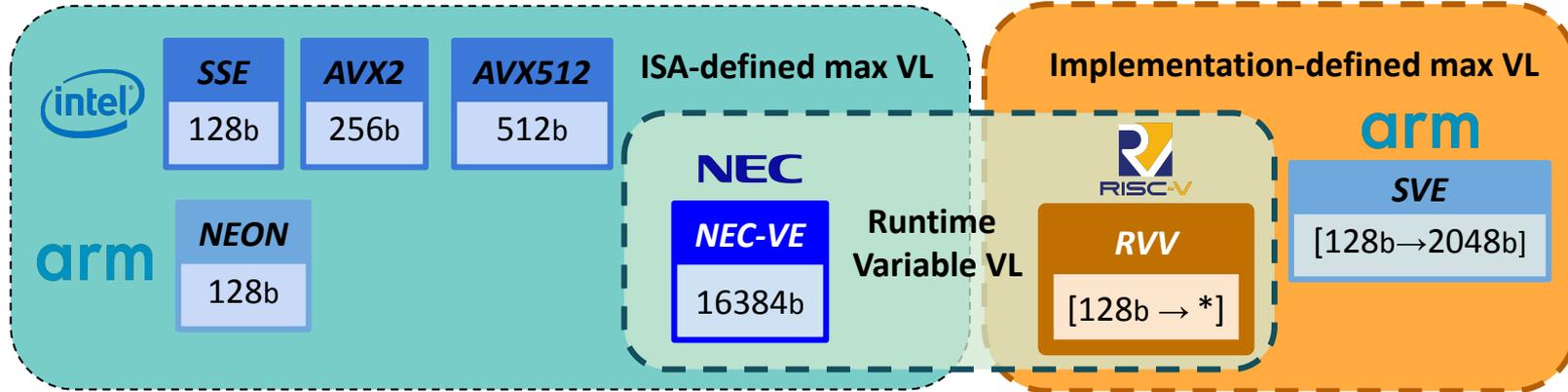
Scalar processor



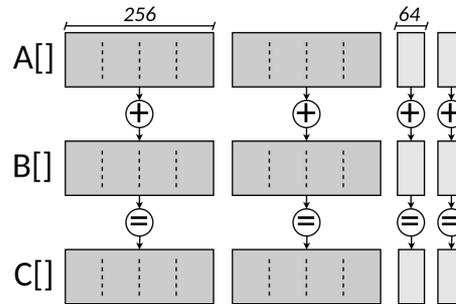
SIMD (e.g., AVX2)

Motivation

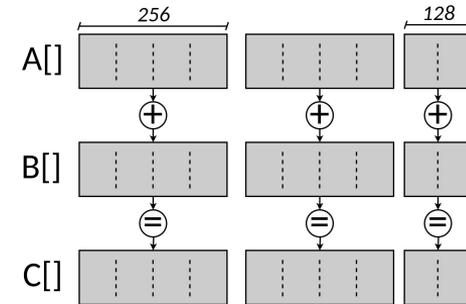
- **RVV** (RISC-V Vector extension) is RV's bet for High Performance Computing



Scalar processor



SIMD (e.g., AVX2)

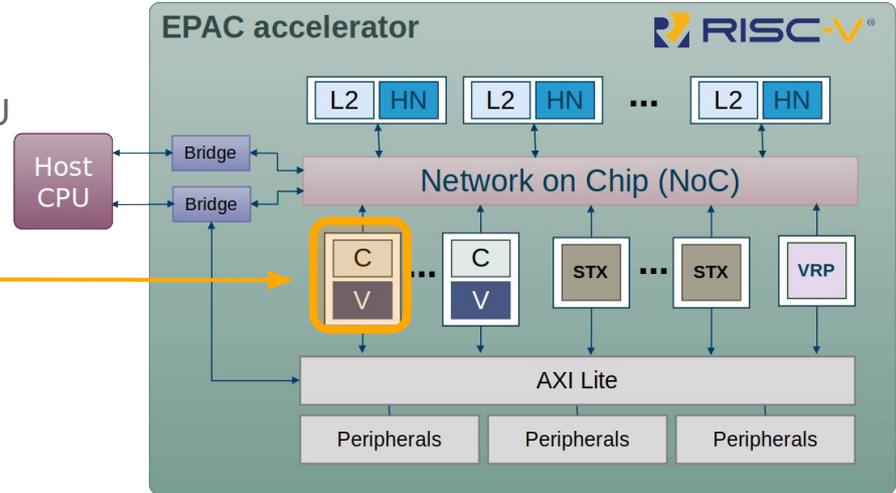


Variable VL (e.g., RVV)

Who is implementing this technology?

European Processor Initiative (EPI)

- **Rhea:** arm-based general purpose CPU
- **EPAC:** European Processor Accelerator
 - Based on **RISC-V**
 - Many tiles: VRP, STX, **VEC**



Very large vector length:



— **AVX512**  ← 512 bits per vector (8 DP elems)

arm – SVE

 ← Up to 2048 bits per vector (16 DP elems)

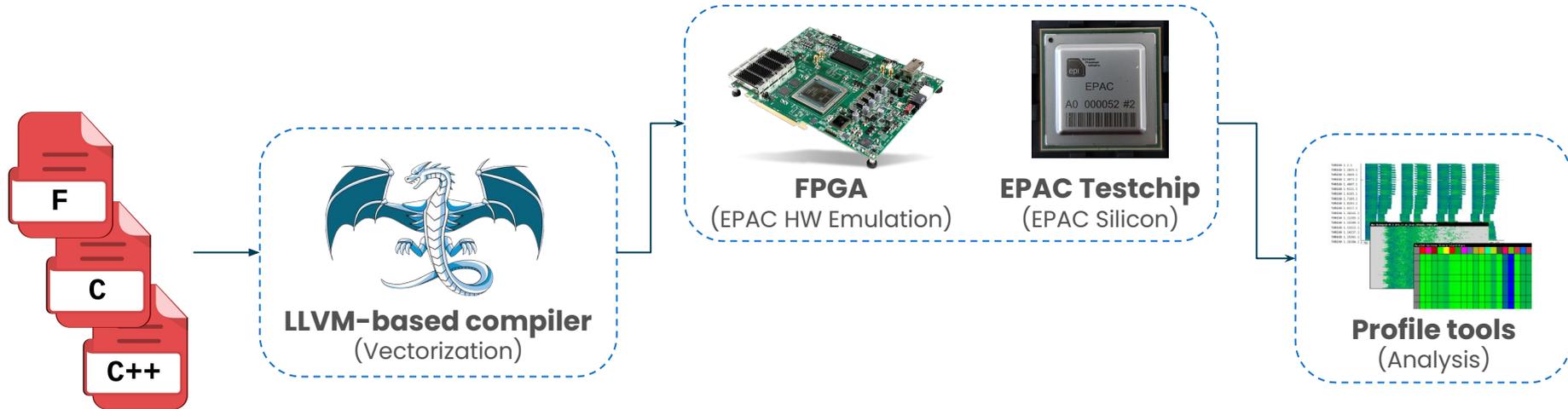
NEC



 ← **16384 bits per vector (256 DP elems)**

How can you develop code for this accelerator?

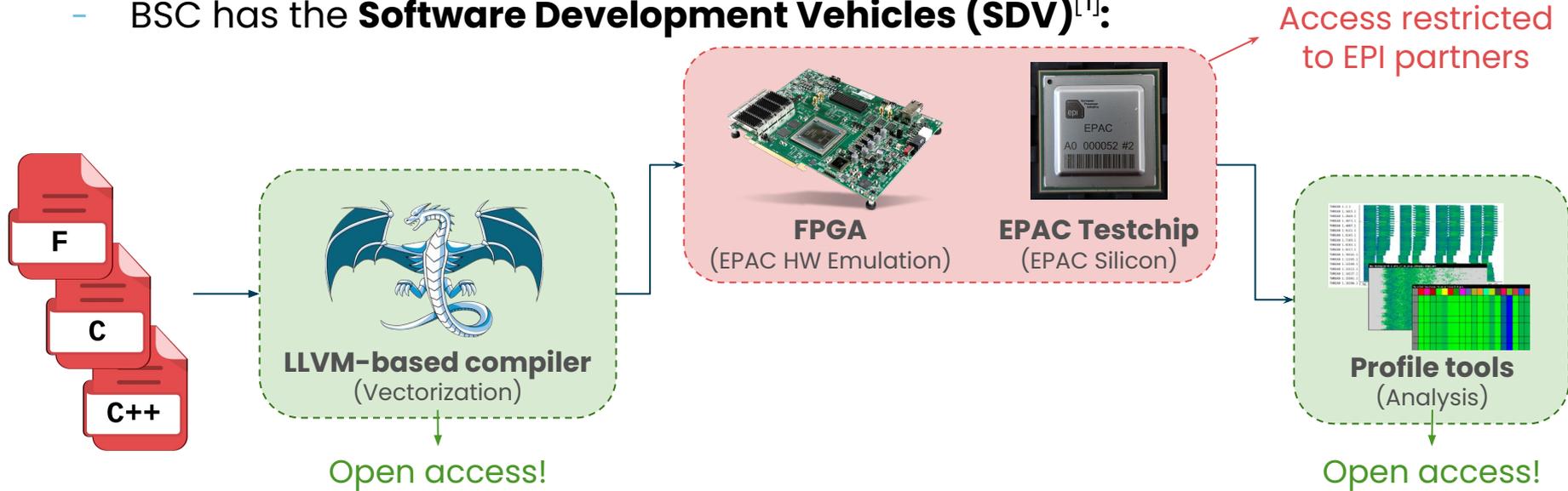
- BSC has the **Software Development Vehicles (SDV)**^[1]:



[1] Filippo Mantovani et al. (2023, May). Software Development Vehicles to enable extended and early co-design: a RISC-V and HPC case of study. In International Conference on High Performance Computing (pp. 526-537). Cham: Springer Nature Switzerland.

How can you develop code for this accelerator?

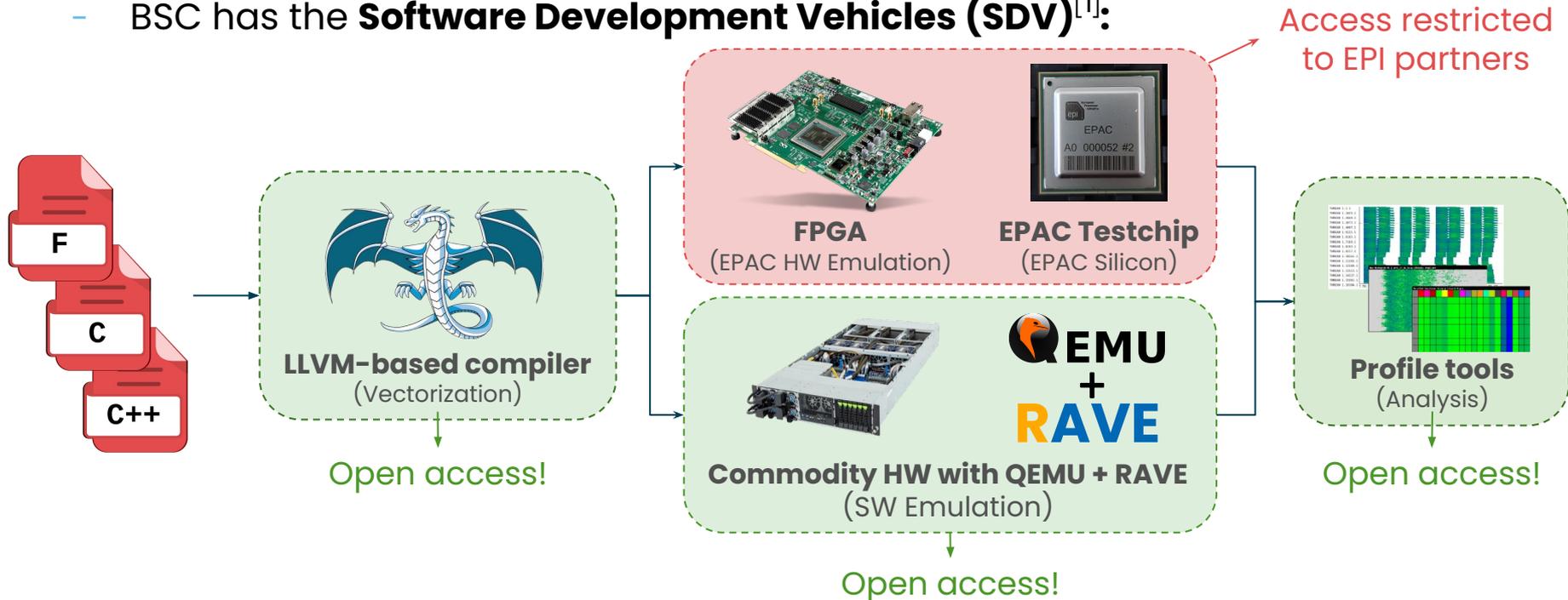
- BSC has the **Software Development Vehicles (SDV)**^[1]:



[1] Filippo Mantovani et al. (2023, May). Software Development Vehicles to enable extended and early co-design: a RISC-V and HPC case of study. In International Conference on High Performance Computing (pp. 526-537). Cham: Springer Nature Switzerland.

How can you develop code for this accelerator?

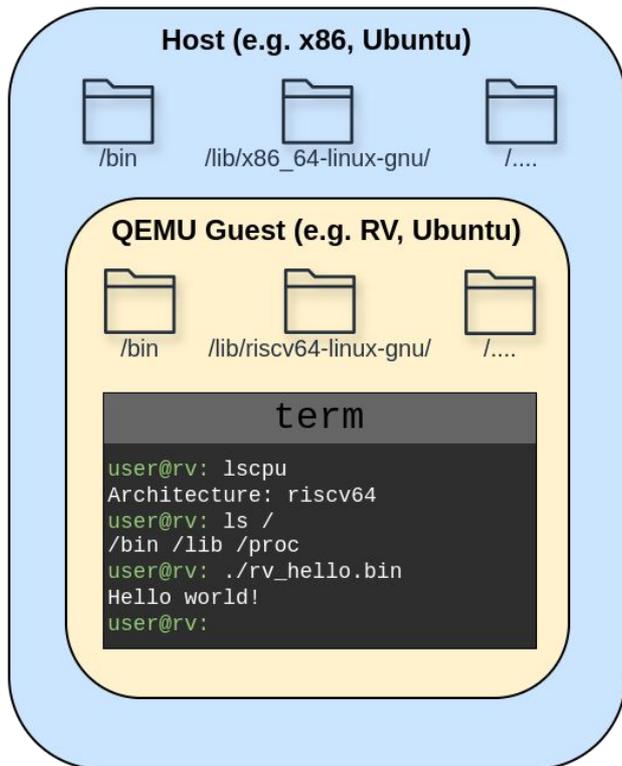
- BSC has the **Software Development Vehicles (SDV)**^[1]:



[1] Filippo Mantovani et al. (2023, May). Software Development Vehicles to enable extended and early co-design: a RISC-V and HPC case of study. In International Conference on High Performance Computing (pp. 526-537). Cham: Springer Nature Switzerland.

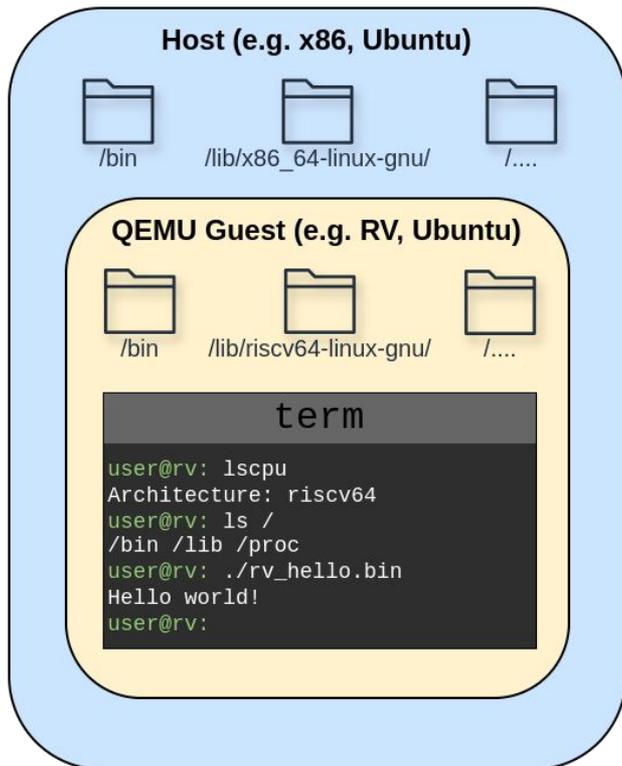
What is QEMU? a software emulator

System-level emulation

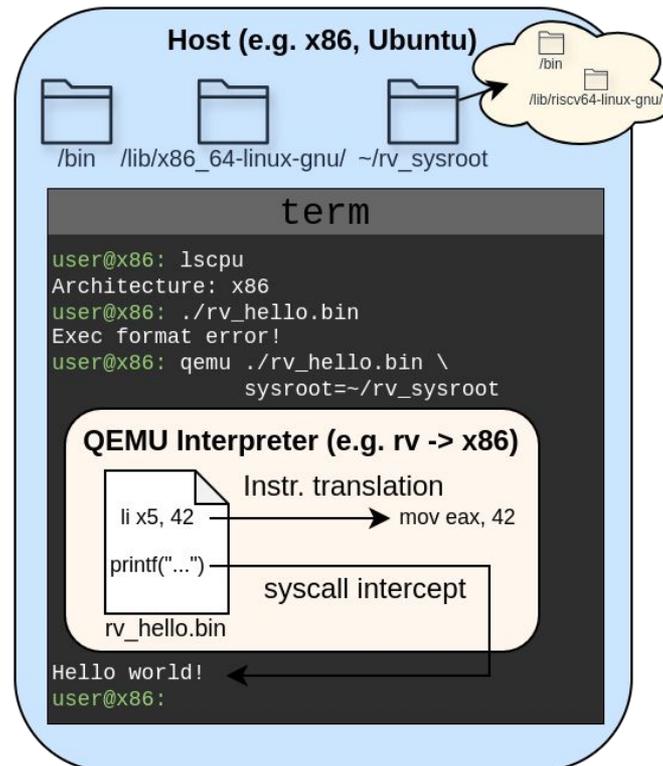


What is QEMU? a software emulator

System-level emulation

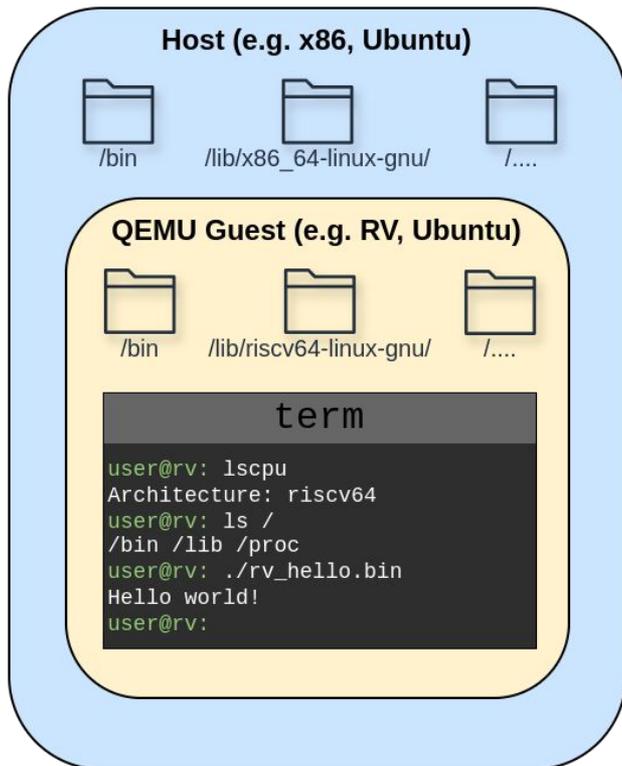


User-level emulation

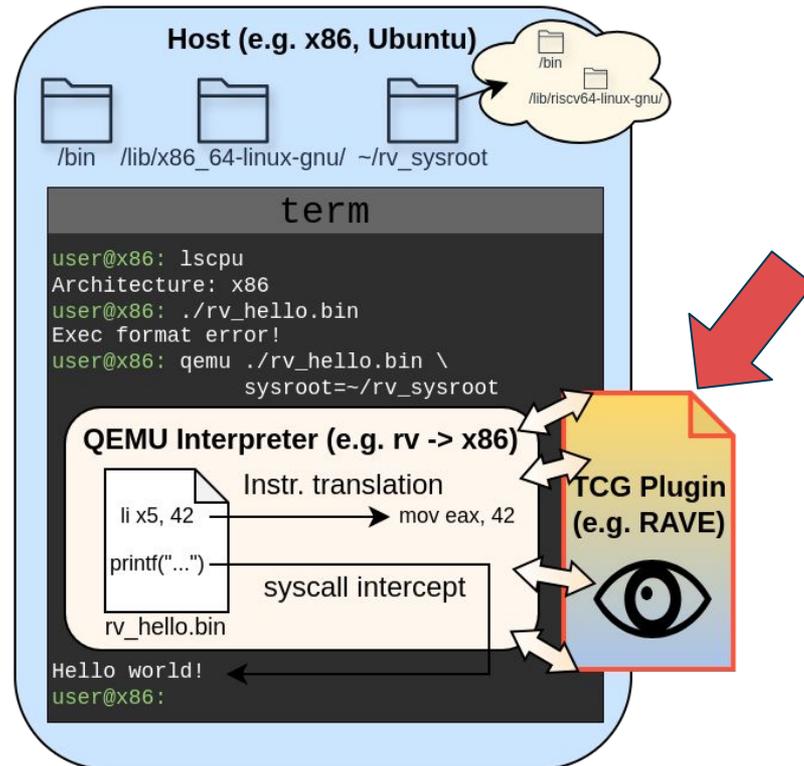


What is **RAVE**? an analysis/profiling plugin

System-level emulation



User-level emulation



What is **RAVE** useful for?

- **RAVE monitors** and **counts** metrics such as:
 - Number of emulated scalar and vector **instructions** (*you can compute Vec.Mix*)
 - Divided by type (Memory, Arithmetic, Mask, stride type, SEW, ...)
 - Average Vector Length (**VL**)
 - Number of **bytes load/stored** with scalar/vector instructions
 - Program Counter (**PC**)

- **RAVE** provides:
 - **API** called for user application to instrument regions of interest
 - Generation of **reports/logs** at the end of the emulation
 - Generation of **Paraver traces** (BSC's format for traces)

What is **RAVE** useful for?

- **RAVE monitors** and **counts** metrics such as:
 - Number of emulated scalar and vector **instructions** (*you can compute Vec.Mix*)
 - Divided by type (Memory, Arithmetic, Mask, stride type, SEW, ...)
 - Average Vector Length (**VL**) → ⚠ *Remember. Variable VL!*
 - Number of **bytes load/stored** with scalar/vector instructions
 - Program Counter (**PC**)

- **RAVE** provides:
 - **API** called for user application to instrument regions of interest
 - Generation of **reports/logs** at the end of the emulation
 - Generation of **Paraver traces** (BSC's format for traces)

Controlling the trace with the **RAVE** API

- We add instrumentation mechanisms, to define regions of interest.
- We work with tuples of Events and Values:

```
int main(){
  rave_name_event(1000, "Code Region")
  rave_name_value(1000, 1, "Ini")
  rave_name_value(1000, 2, "Compute")

  double array1[256], array2[256], array3[256];

  rave_event_and_value(1000, 1)
  ini_vectors(array1, array2, array3);
  rave_event_and_value(1000, 0)

  rave_event_and_value(1000, 2)
  for(int i=0; i<256; ++i)
    array3[i] += array1[i] + array2[i];
  rave_event_and_value(1000, 0)
};
```

Define event **1000** = "Code Region"
 Value **1** = "Ini"
 Value **2** = "Compute"

Enclose first region with value **1** ("Ini")

Enclose second region with value **2** ("Compute")

Controlling the trace with the RAVE API

- We add instrumentation mechanisms, to define regions of interest.
- We work with tuples of Events and Values:

```

int main(){
  rave_name_event(1000, "Code Region")
  rave_name_value(1000, 1, "Ini")
  rave_name_value(1000, 2, "Compute")

  double array1[256], array2[256], array3[256];

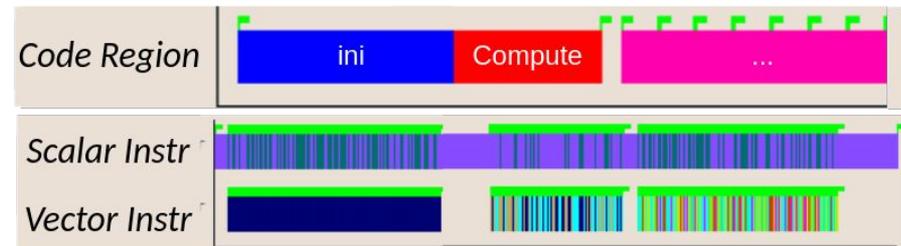
  rave_event_and_value(1000, 1)
  ini_vectors(array1, array2, array3);
  rave_event_and_value(1000, 0)

  rave_event_and_value(1000, 2)
  for(int i=0; i<256; ++i)
    array3[i] += array1[i] + array2[i];
  rave_event_and_value(1000, 0)
};
  
```

Define event 1000 = "Code Region"
 Value 1 = "Ini"
 Value 2 = "Compute"

Enclose first region with value 1 ("Ini")

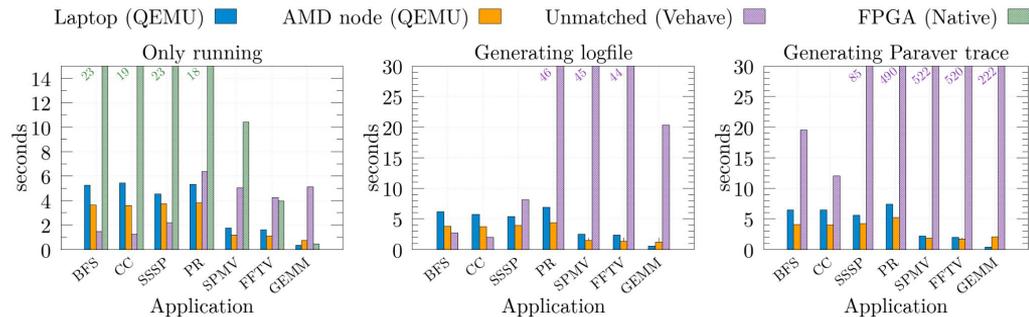
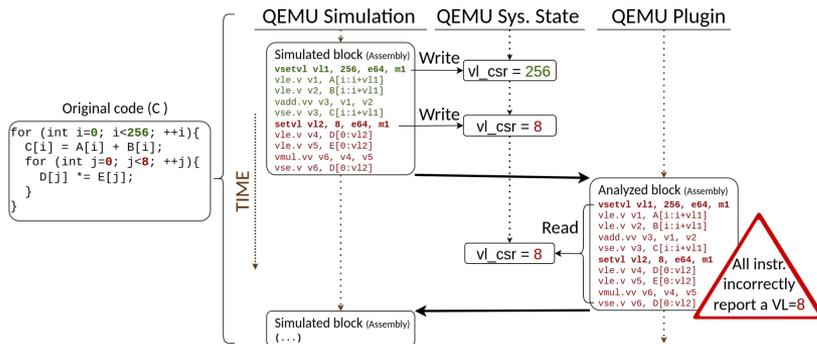
Enclose second region with value 2 ("Compute")



Before continuing....

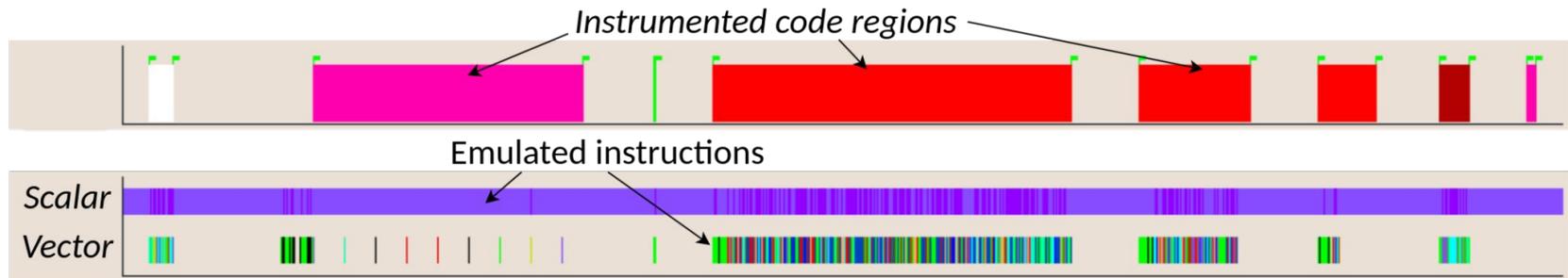
- If you want to know more about the **RAVE** internals :

 **RAVE: RISC-V Analyzer of Vector Executions, a QEMU tracing plugin** <https://arxiv.org/abs/2409.13639>



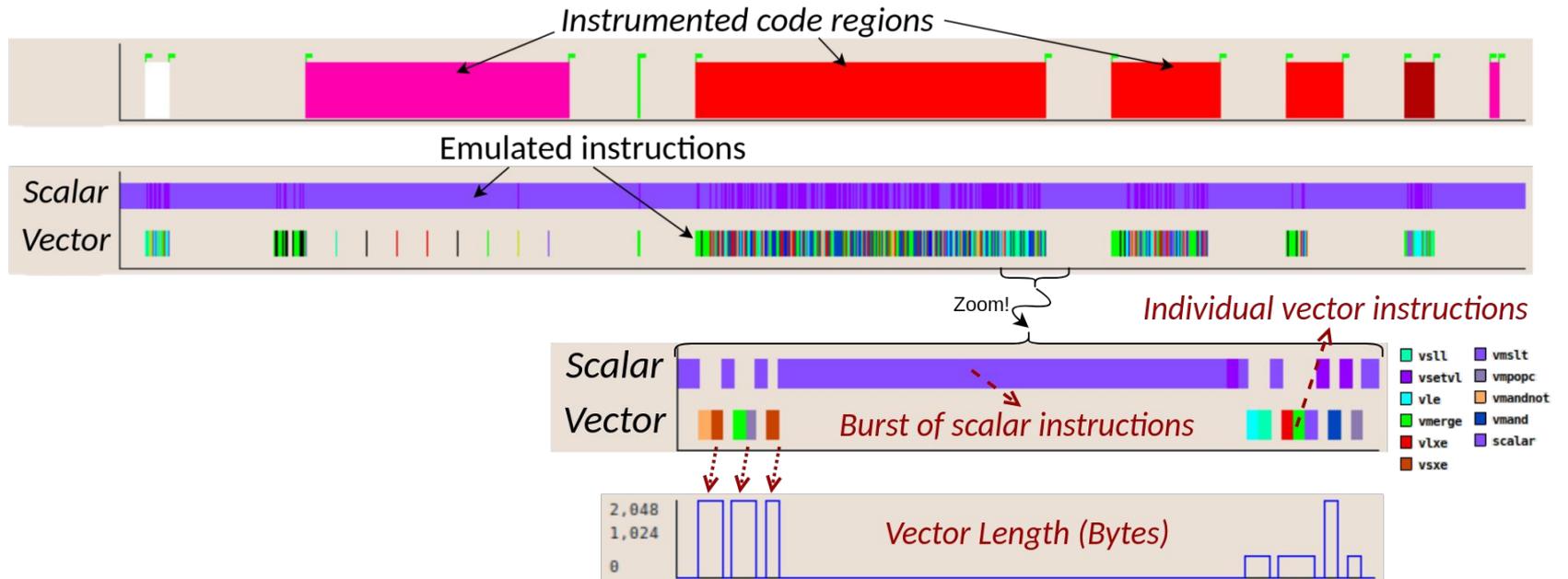
RAVE Use case: Emulation Trace (I)

- We emulated a Breadth First Search (**BFS**) code vectorized with **RVV**



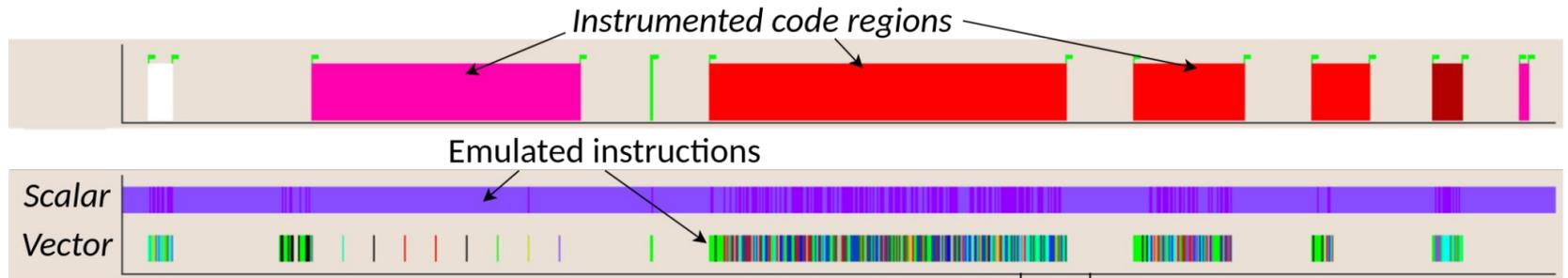
RAVE Use case: Emulation Trace (I)

- We emulated a Breadth First Search (**BFS**) code vectorized with **RVV**



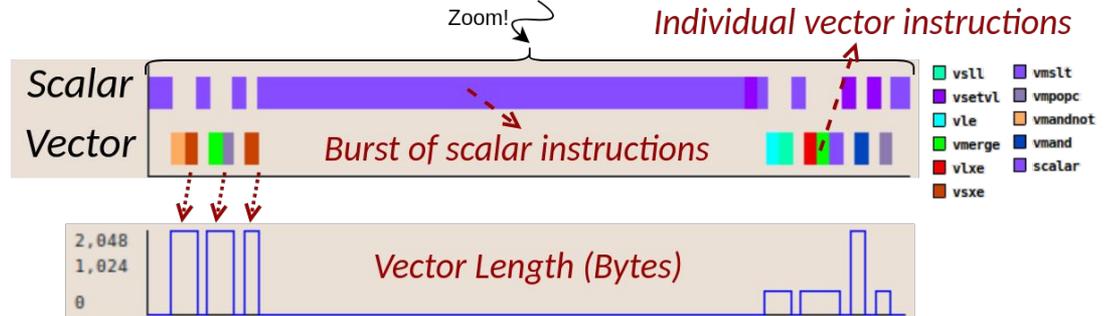
RAVE Use case: Emulation Trace (I)

- We emulated a Breadth First Search (**BFS**) code vectorized with **RVV**



Prop. of Scalar and Vector instructions

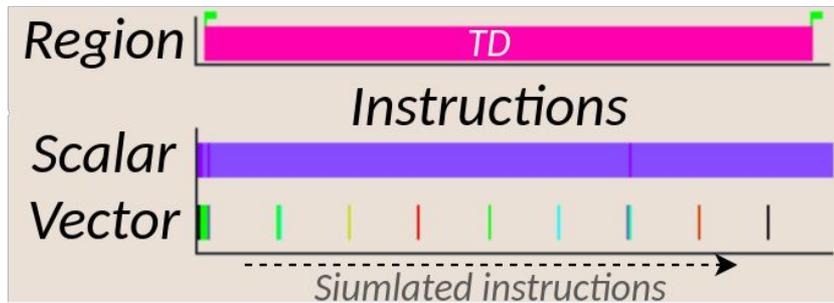
	Init	BU	TD	Bit->Q (c)	Q->Bit (e)
qemu_scalar	0.62	0.63	1.00	0.81	0.74
qemu_vector	0.38	0.37	0.00	0.19	0.26



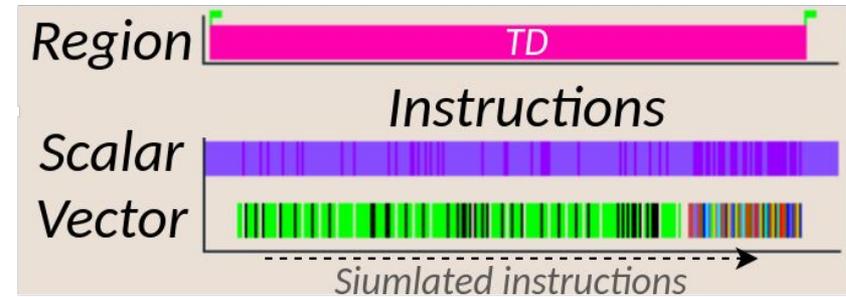
RAVE Use case: Emulation Trace (II)

Use trace insight to improve vectorization:

Before increasing TD Vectorization



After increasing TD Vectorization



	Init	BU	TD	Bit->Q (c)
qemu_scalar	0.62	0.63	1.00	0.81
qemu_vector	0.38	0.37	0.00	0.19

	Init	BU	TD	Bit->Q (c)
qemu_scalar	0.62	0.63	0.84	0.81
qemu_vector	0.38	0.37	0.16	0.19

RAVE Use case: Emulation Trace (III)

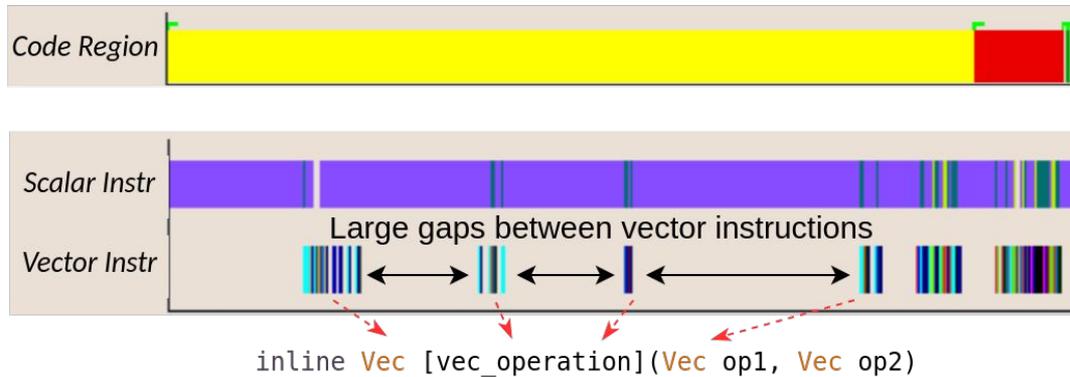
- We emulated a Plasma-Physics application called Vlasiator ^[1]
- The code was not initially designed with long vectors in mind:

[1] <https://www.helsinki.fi/en/researchgroups/vlasiator>

RAVE Use case: Emulation Trace (III)

- We emulated a Plasma-Physics application called Vlasiator ^[1]
- The code was not initially designed with long vectors in mind:

Vanilla Version

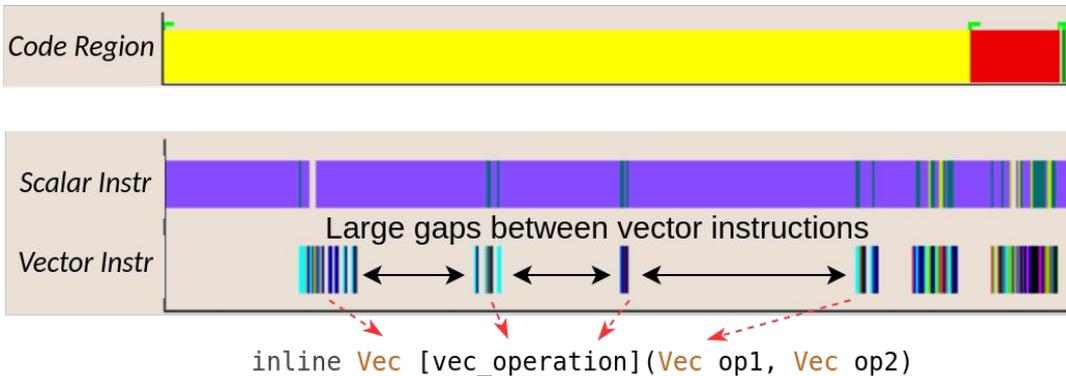


[1] <https://www.helsinki.fi/en/researchgroups/vlasiator>

RAVE Use case: Emulation Trace (III)

- We emulated a Plasma-Physics application called Vlasiator ^[1]
- The code was not initially designed with long vectors in mind:

Vanilla Version



Scalar gaps not seen on the C++ code

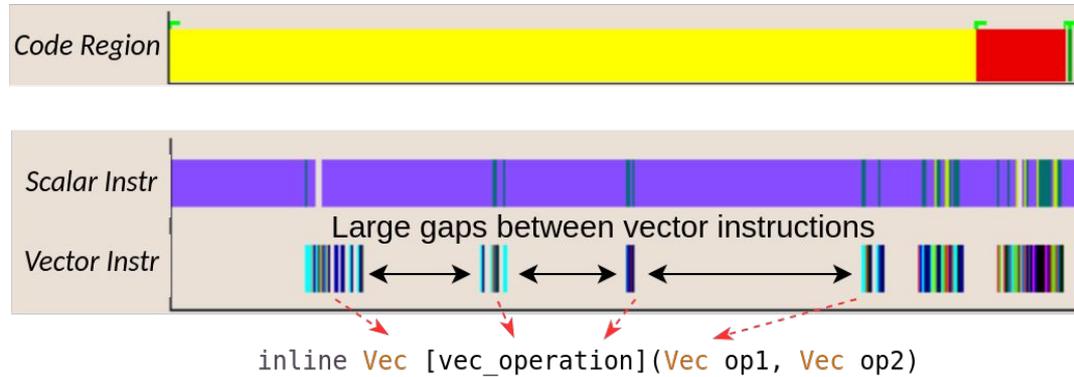
Added by the compiler. Copies!!

[1] <https://www.helsinki.fi/en/researchgroups/vlasiator>

RAVE Use case: Emulation Trace (III)

- We emulated a Plasma-Physics application called Vlasiator ^[1]
- The code was not initially designed with long vectors in mind:

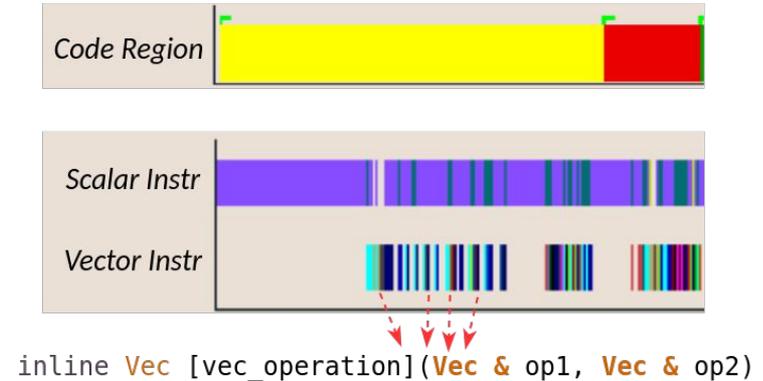
Vanilla Version



Scalar gaps not seen on the C++ code

Added by the compiler. Copies!!

After eliminating scalar copies



*Removed by passing
operands per reference*

[1] <https://www.helsinki.fi/en/researchgroups/vlasiator>

RAVE Use case: Console report

You can also obtain vectorization metrics on a console report:

```
your-machine$ rave ./bfs -f graph.el
(...)
Reg. #3: Event 1000(code_region), Value 3(BU)
  tot_instr: 38872
  scalar_instr: 15818 (40.69 %)
  vsetvl_instr: 5236 (13.47 %)
  SEW 64 vector_instr: 17818 (45.84 %)
    avg_VL: 255.60 elements
    Arith: 2466 (13.84 %)
      FP: 0 (0.00 %)
      INT: 2466 (100.00 %)
    Mem: 3142 (17.63 %)
      unit: 1573 (50.06 %)
      strided: 0 (0.00 %)
      indexed: 1569 (49.94 %)
    Mask: 8171 (45.86 %)
    Other: 4039 (22.67 %)
```

RAVE Use case: Console report

You can also obtain vectorization metrics on a console report:

```
your-machine$ rave ./bfs -f graph.el
(...)
Reg. #3: Event 1000(code_region), Value 3(BU)
  tot_instr: 38872
  scalar_instr: 15818 (40.69 %)
  vsetvl_instr: 5236 (13.47 %)
  SEW 64 vector_instr: 17818 (45.84 %)
    avg VL: 255.60 elements
    Arith: 2466 (13.84 %)
      FP: 0 (0.00 %)
      INT: 2466 (100.00 %)
    Mem: 3142 (17.63 %)
      unit: 1573 (50.06 %)
      strided: 0 (0.00 %)
      indexed: 1569 (49.94 %)
    Mask: 8171 (45.86 %)
    Other: 4039 (22.67 %)
```

RAVE Use case: Console report

You can also obtain vectorization metrics on a console report:

<pre> your-machine\$ rave ./bfs -f graph.el (...) Reg. #3: Event 1000(code_region), Value 3(BU) tot_instr: 38872 scalar_instr: 15818 (40.69 %) vsetvl_instr: 5236 (13.47 %) SEW 64 vector_instr: 17818 (45.84 %) avg VL: 255.60 elements Arith: 2466 (13.84 %) FP: 0 (0.00 %) INT: 2466 (100.00 %) Mem: 3142 (17.63 %) unit: 1573 (50.06 %) strided: 0 (0.00 %) indexed: 1569 (49.94 %) Mask: 8171 (45.86 %) Other: 4039 (22.67 %) </pre>	<p>Reduction in Mask and Other Vec Instructions</p> 	<pre> your-machine\$ rave ./bfs_no_if -f graph.el (...) Reg. #3: Event 1000(code_region), Value 3(BU) tot_instr: 44780 scalar_instr: 21866 (48.83 %) vsetvl_instr: 9556 (21.34 %) SEW 64 vector_instr: 13358 (29.83 %) avg VL: 254.77 elements Arith: 2481 (18.57 %) FP: 0 (0.00 %) INT: 2481 (100.00 %) Mem: 3028 (22.67 %) unit: 1454 (48.02 %) strided: 0 (0.00 %) indexed: 1574 (51.98 %) Mask: 4992 (37.37 %) Other: 2857 (21.39 %) </pre>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Conclusions

- We developed a plugin for **QEMU** targeting the **RISC-V Vector Extension**
- **RAVE** allows to study vectorized applications with fine-grain detail:
 - Instruction Mix, Vector Length, ...
- **RAVE** is already being used by performance analysts at BSC to study HPC applications
- Future work includes:
 - Multi-core emulation (OMP and MPI)
 - Automatic instrumentation of user functions
 - Adding a timing model

Try it yourself!

<https://repo.hca.bsc.es/gitlab/pvizcaino/rave>



Acknowledgment

This research has received funding from the European High Performance Computing Joint Undertaking (JU) under Framework Partnership Agreement No 800928 (European Processor Initiative) 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. The EPI-SGA2 project, PCI2022-132935 is also co-funded by MCIN/AEI /10.13039/501100011033 and by the UE NextGenerationEU/PRTR.



Financé par



Financé par l'Union européenne



Don't hesitate to contact me at pablo.vizcaino@bsc.es !