



**Barcelona  
Supercomputing  
Center**  
*Centro Nacional de Supercomputación*



# The RISC-V “accelerator” in EPI

**Prof. Jesús Labarta**  
BSC & UPC

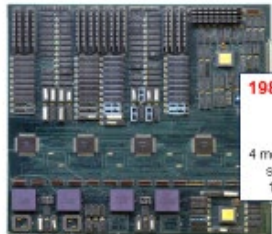
ACM Summer School  
Barcelona, August 30<sup>th</sup> 2022

# Disclaimer

- About myself



## What I did



1988

4 i386  
2 T800  
5x5 xbar  
4 memory banks  
shared cache  
13 layer PCB

```
void Cholesky(float *A) {  
  int i, j, k;  
  for (k=0; k<NT; k++) {  
    spotrf (A[k*NT+k]);  
    for (i=k+1; i<NT; i++)  
      strsm (A[k*NT+k], A[k*NT+i]);  
    for (i=k+1; i<NT; i++) {  
      for (j=k+1; j<NT; j++)  
        apgemv (A[k*NT+i], A[k*NT+j], A[j*NT+i]);  
      sayrk (A[k*NT+i], A[i*NT+i]);  
    }  
  }  
}
```

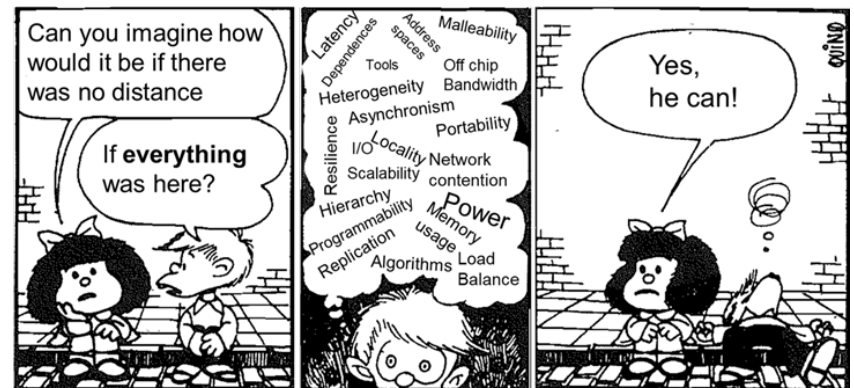
```
#pragma omp task inout ([TS])(TS)A  
void spotrf (float *A);  
#pragma omp task input ([TS])(TS)T inout ([TS])(TS)B;  
void strsm (float *T, float *B);  
#pragma omp task input ([TS])(TS)A, [TS])(TS)B; inout ([TS])(TS)C;  
void apgemv (float *A, float *B, float *C);  
#pragma omp task input ([TS])(TS)A inout ([TS])(TS)C;  
void sayrk (float *A, float *C);
```

El abuelo cebolleta  
ataca de nuevo

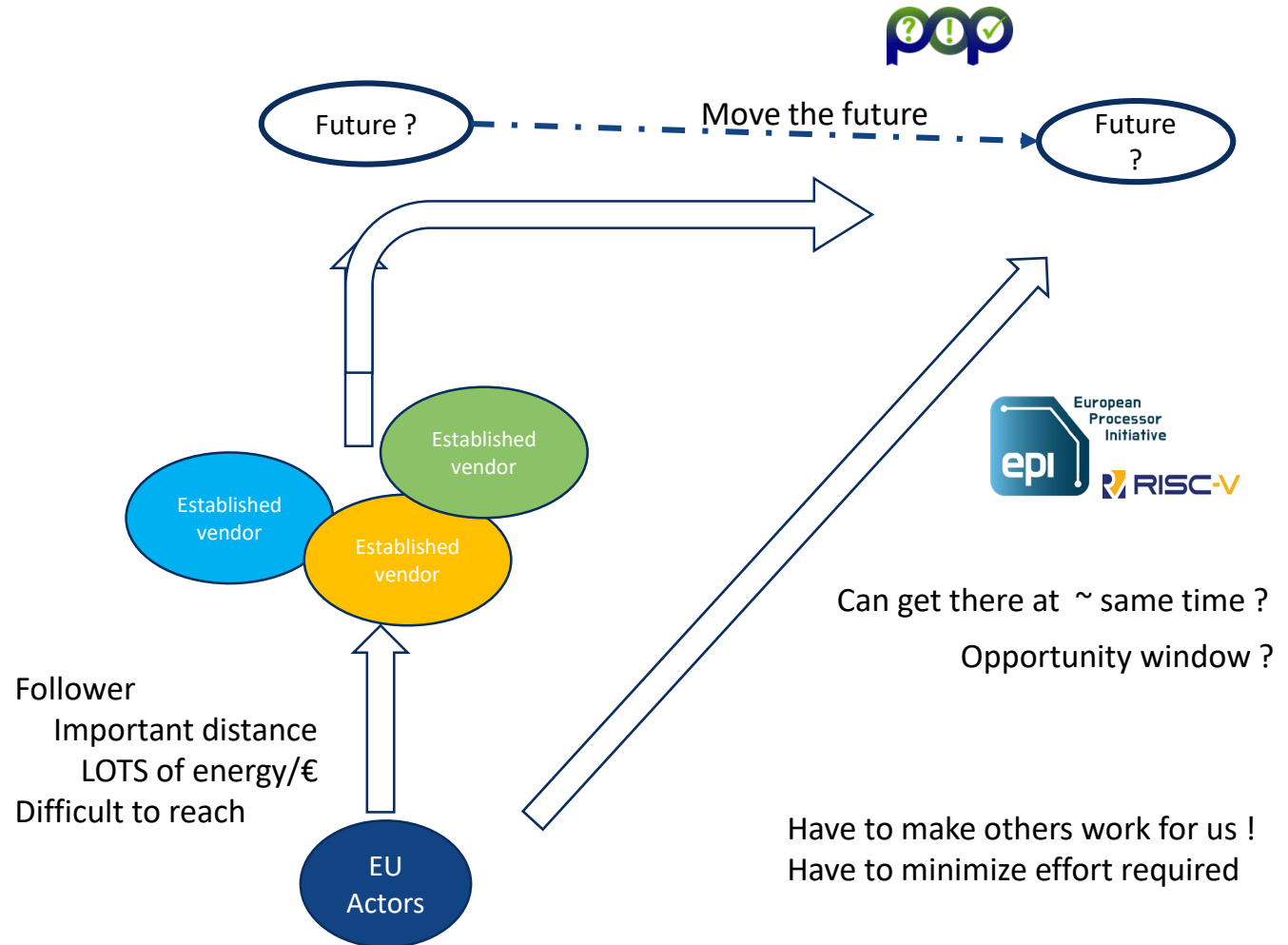


# The EPI FPA Objective

- **Components** (low power microprocessor technologies) ...
  - ARM based SoC
  - RISC-V based accelerator
- ... to be combined to target
  - HPC
  - HPDA
  - Emerging
    - Automotive
    - ...



# The importance of a vision



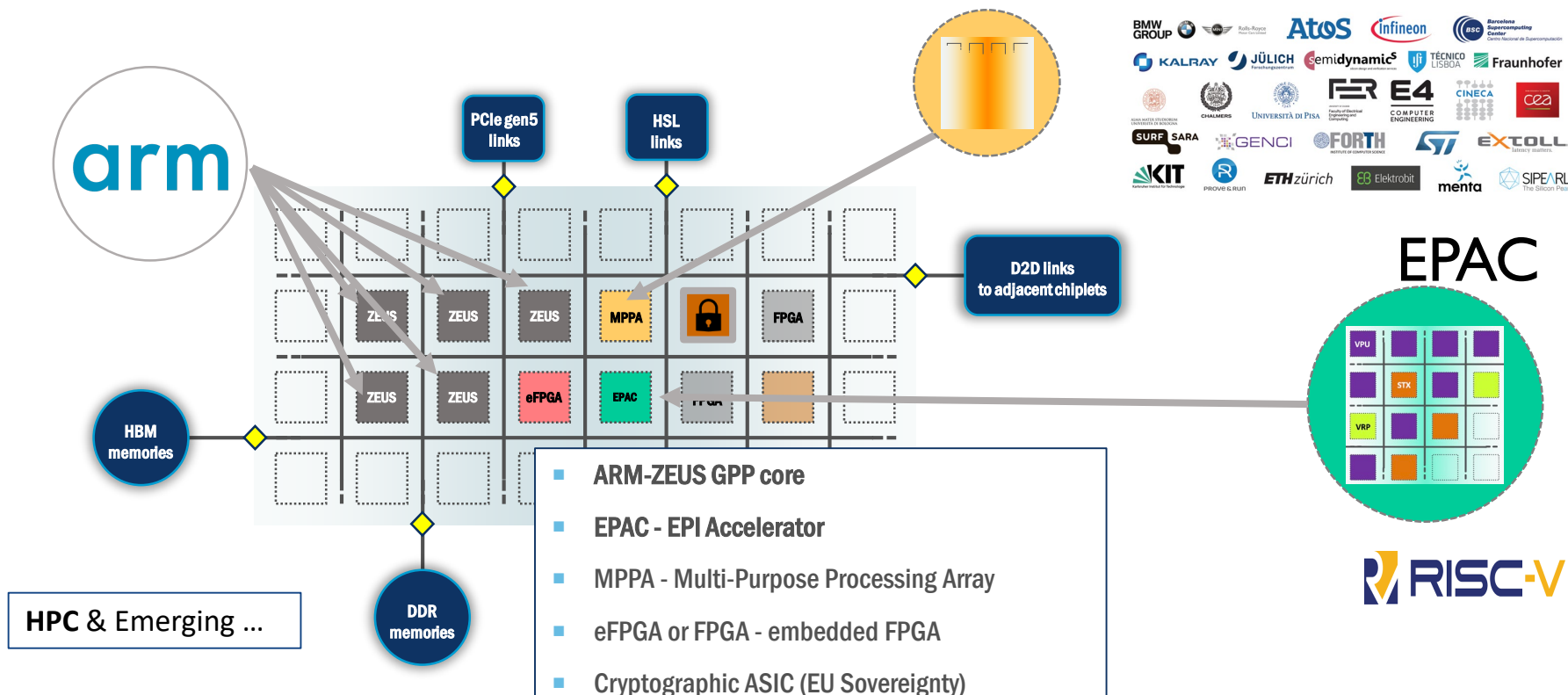
Thoughts from 2016



# Three streams

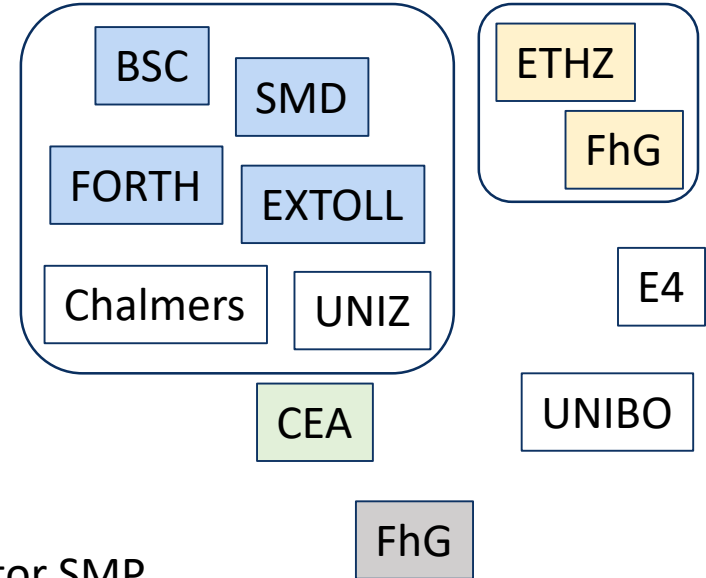
- General purpose
  - ARM SVE
  - BULL: System integrator → chip integrator → SiPearl
- Accelerator
  - RISC-V
  - EU design: BSC, Semidynamics, EXTOLL, ETHZ, UNIBO, Chalmers, ...
- Automotive
  - Infineon, ...

# EPAC within EPI



# Visions and collaborations

- STX:
  - Specific Accelerator devices
    - AI
    - Stencil
- RVV
  - ISA is important, RISC-V Vector
  - “Accelerator”
    - Easier entry, focus
    - → Standard self hosted, general purpose vector SMP
- VRP:
  - Extended precision arithmetic





# Objective

- Explain vision, discuss fundamentals, philosophy or the approach, ...
- Show the available environment and how it works ...
- Want to join the effort?





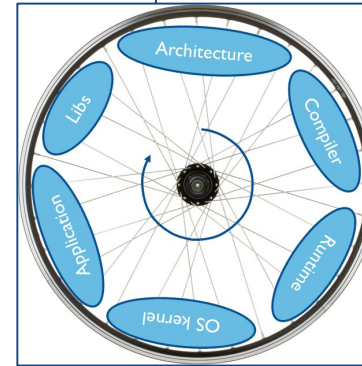
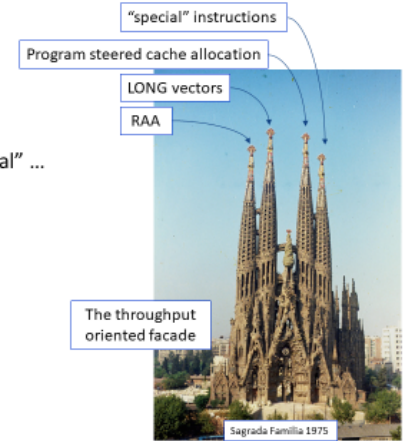
# The importance of a vision



- Holistic throughput oriented vision based on long vectors and task based models
- Hierarchical concurrency and locality exploitation
  - Not massive concurrency at a given level
  - Push behaviour exploitation to low levels
- Co-ordination between levels
- Make it all look very close to classical sequential programming to ensure productivity

## EPAC & Sagrada Familia ?

- There is something "special" ...
- ...showing the way ...
- ... sustaining the effort





**Barcelona  
Supercomputing  
Center**  
*Centro Nacional de Supercomputación*



EXCELENCIA  
SEVERO  
OCHOA

# Towards Holistic Co-design

# Holistic co-design



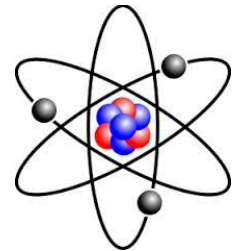
“As above, so below”

Similar concepts/mechanisms at all levels

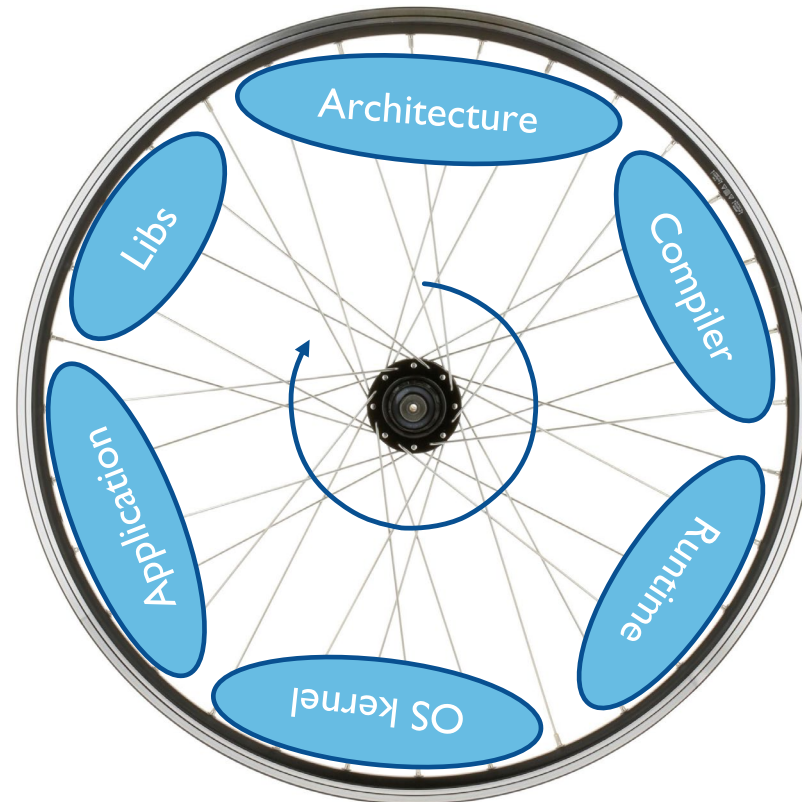
“Steered by a vision/principles”

“Steered by detailed insight”

Co-designing vs. Design



# Holistic Co-design



Best place to address an issue

Fundamentals

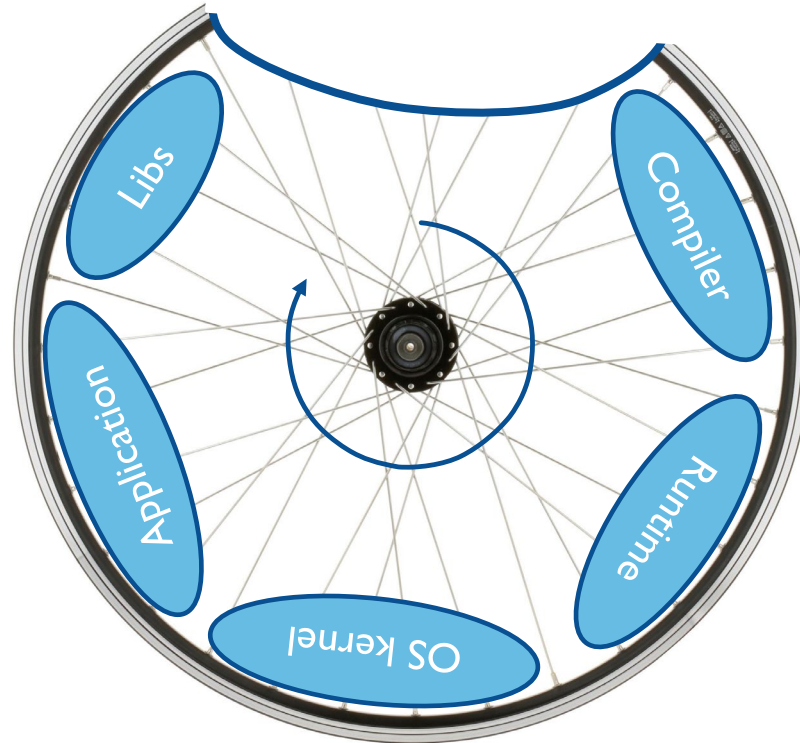
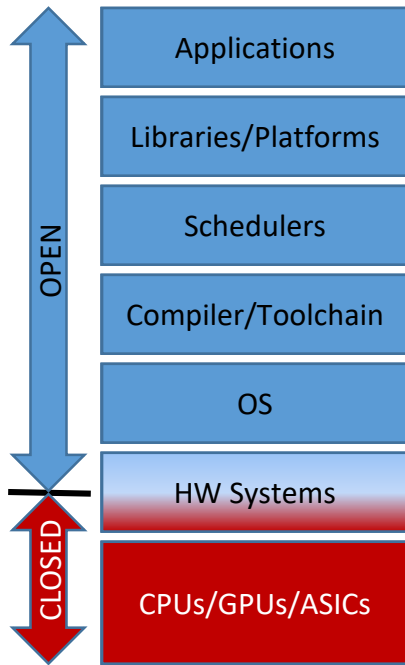
Balance

Mindset

Productivity

Efficiency

# Holistic Co-design



Best place to address an issue

Fundamentals

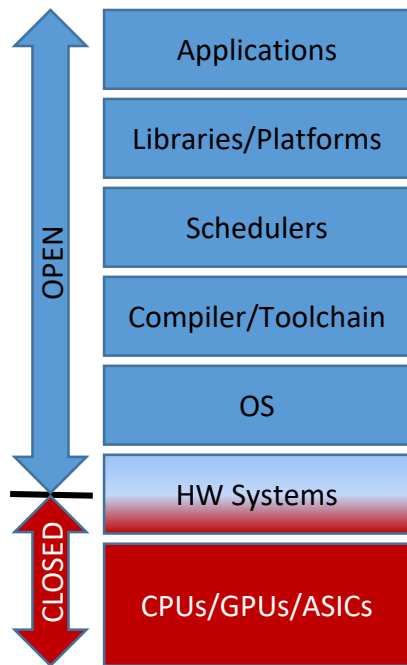
Balance

Mindset

Productivity

Efficiency

# Leverage interfaces and implementations



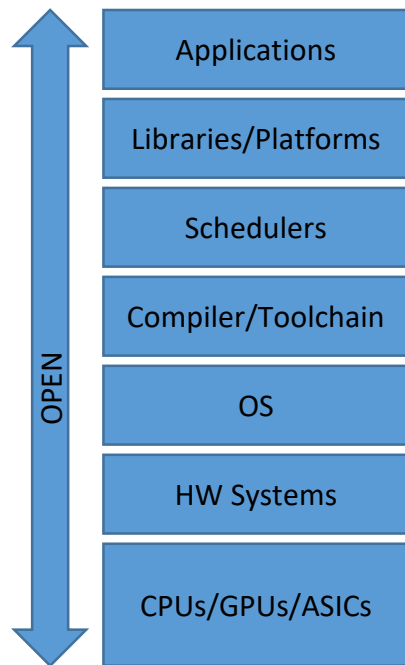
**MPI**



Leverage “standards”  
Opportunity to innovate  
and contribute



# Leverage interfaces and implementations



**MPI**



Leverage “standards”  
Opportunity to innovate  
and contribute

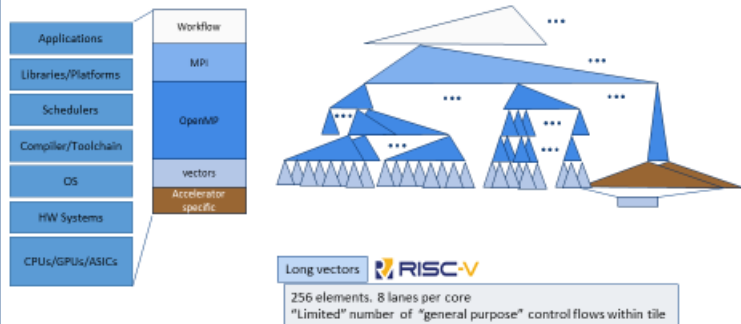


# Principles ?

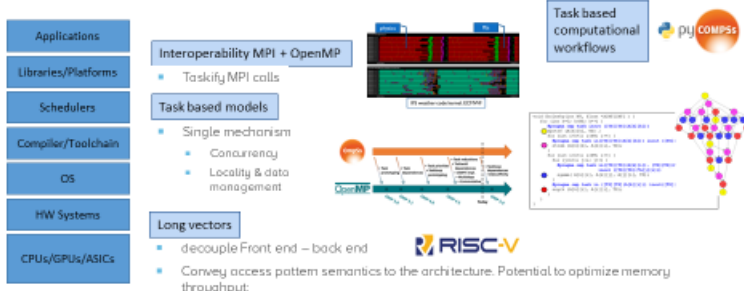
## Balanced hierarchy

Expression & exploitation of Parallelism

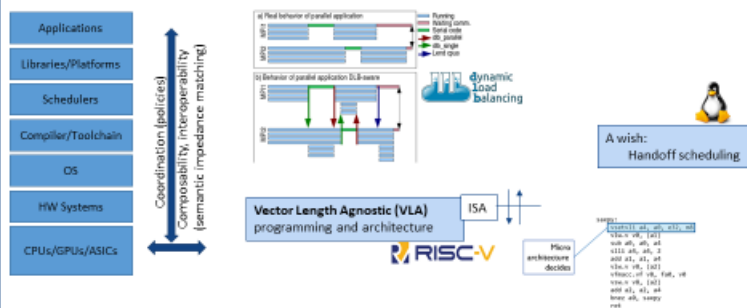
$$10^6 = 1 \times 10^6 = 10 \times 10^5 = 10^2 \times 10^2 \times 10^2$$



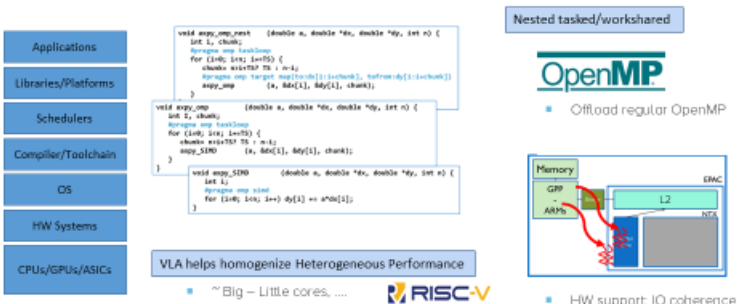
## Latency → Throughput: asynchrony and overlap



## Malleability & Coordinated scheduling



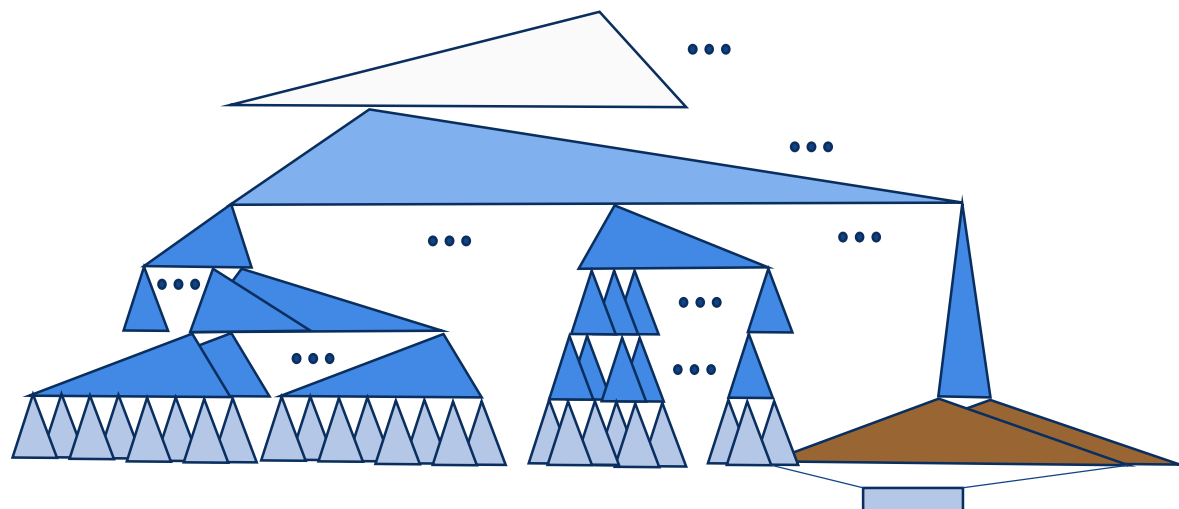
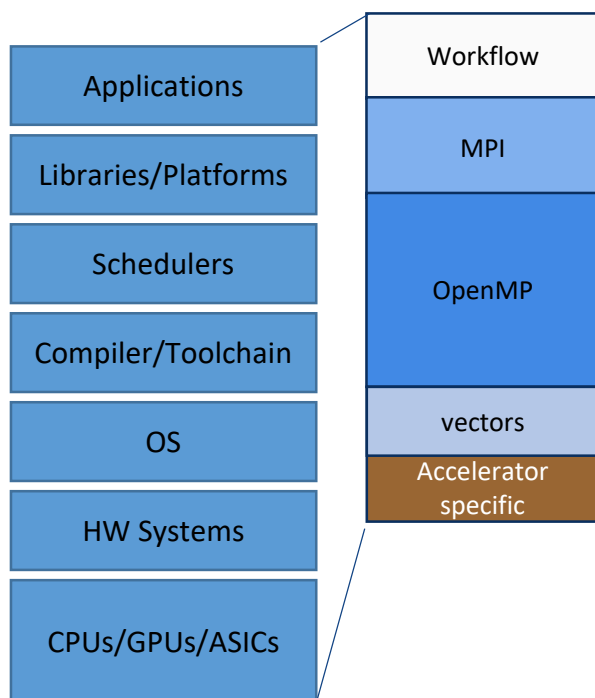
## Homogenizing Heterogeneity



# Balanced hierarchy

Expression &  
exploitation of  
Parallelism

$$10^6 = 1 \times 10^6 = 10 \times 10^5 = 10^2 \times 10^2 \times 10^2$$



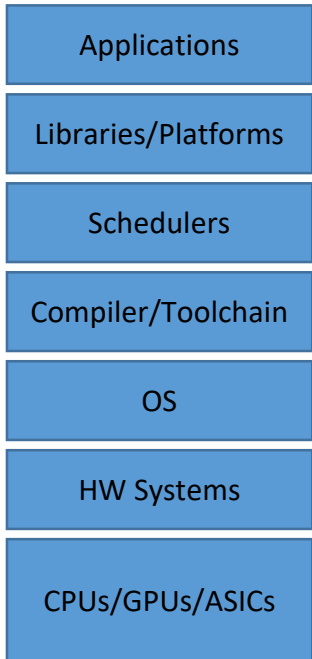
Long vectors



256 elements. 8 lanes per core

“Limited” number of “general purpose” control flows within tile

# Latency $\rightarrow$ Throughput: asynchrony and overlap



## Interoperability MPI + OpenMP

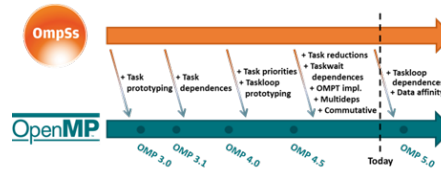
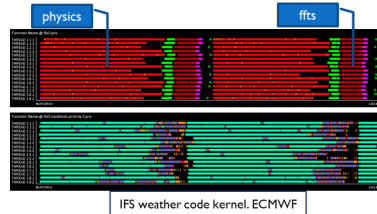
- Taskify MPI calls

## Task based models

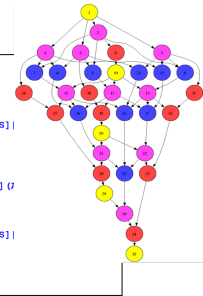
- Single mechanism
  - Concurrency
  - Locality & data management

## Long vectors

- decouple Front end – back end
- Convey access pattern semantics to the architecture. Potential to optimize memory throughput:



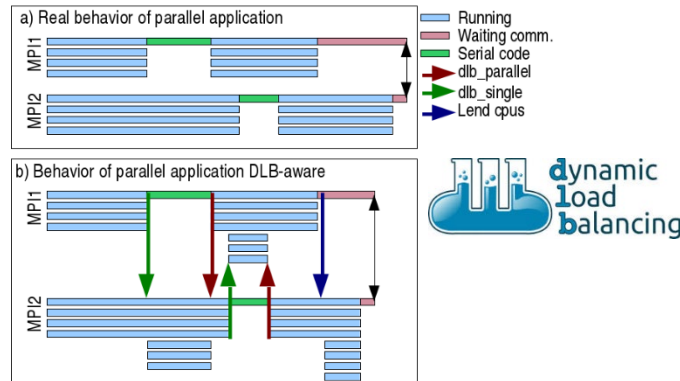
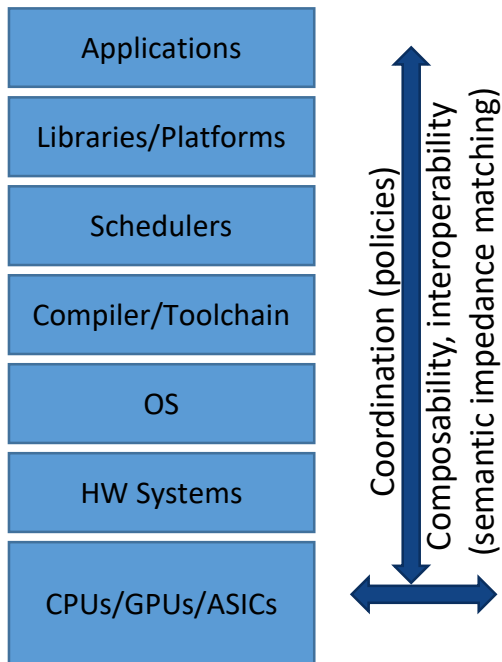
Task based  
computational  
workflows



```
void Cholesky(int NT, float *A[NT][NT]) {
    for (int k=0; k<NT; k++) {
        #pragma omp task inout ([TS] [A][k][k])
        #pragma omp taskwait ([TS])
        for (int i=k+1; i<NT; i++) {
            #pragma omp task in ([TS] [A][k][i]) inout ([TS])
            stram [A][k][i], A[i][i], TS);
        }
        for (int i=k+1; i<NT; i++) {
            for (j=k+1; j<=i; j++) {
                #pragma omp task in ([TS] [A][k][i],
                #pragma omp task in ([TS] [A][k][j], [A][j][i])
                #pragma omp task in ([TS] [A][i][i], [A][j][j], TS);
            }
            #pragma omp task in ([TS] [A][k][i]) inout ([TS])
            syyk [A][k][i], A[i][i], TS);
        }
    }
}
```



# Malleability & Coordinated scheduling



A wish:  
Handoff scheduling



Micro architecture decides

```
saxpy:
vsetvli a4, a0, e32, m8
v1w.v v0, (a1)
sub a0, a0, a4
slli a4, a4, 2
add a1, a1, a4
v1w.v v8, (a2)
vmacc.vf v8, fa0, v0
vsw.v v8, (a2)
add a2, a2, a4
bnez a0, saxpy
ret
```

# Homogenizing Heterogeneity

Applications

Libraries/Platforms

Schedulers

Compiler/Toolchain

OS

HW Systems

CPUs/GPUs/ASICs

```
void axpy_omp_nest (double a, double *dx, double *dy, int n) {  
    int i, chunk;  
    #pragma omp taskloop  
    for (i=0; i<n; i+=TS) {  
        chunk= n>i+TS? TS : n-i;  
        #pragma omp target map(to:dx[i:i+chunk], tofrom:dy[i:i+chunk])  
        axpy_omp (a, &dx[i], &dy[i], chunk);  
    }  
}
```

```
void axpy_omp (double a, double *dx, double *dy, int n) {  
    int i, chunk;  
    #pragma omp taskloop  
    for (i=0; i<n; i+=TS) {  
        chunk= n>i+TS? TS : n-i;  
        axpy_SIMD (a, &dx[i], &dy[i], chunk);  
    }  
}
```

```
void axpy_SIMD (double a, double *dx, double *dy, int n) {  
    int i;  
    #pragma omp simd  
    for (i=0; i<n; i++) dy[i] += a*dx[i];  
}
```

VLA helps homogenize Heterogeneous Performance

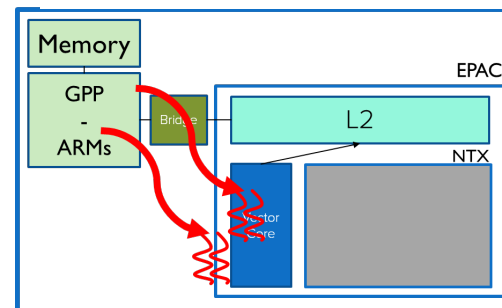
- ~ Big – Little cores, ....



Nested tasked/workshared



- Offload regular OpenMP

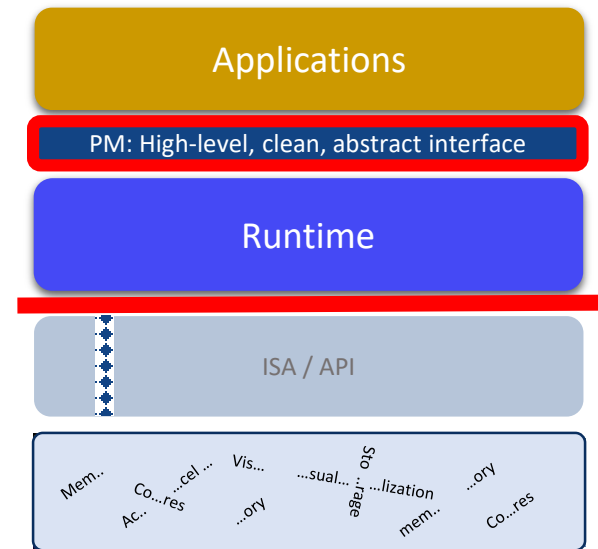


- HW support: IO coherence

# Long vectors

- Raise ISA semantic level
  - Vector instructions == tasks
  - “less words, more work”
  - The importance of ISA
- Parallelism: Asynchrony and overlap
  - Decouple Front end – back end
    - Less pressure, throughput orientation
  - OoO execution
- Locality management
  - Hierarchy concept & hints
- Osmotic membrane
  - Convey access pattern semantics to the architecture.
  - Potential to optimize memory throughput

 **RISC-V** an enabler



# Detailed analysis and Insight on behavior

Applications

Libraries/Platforms

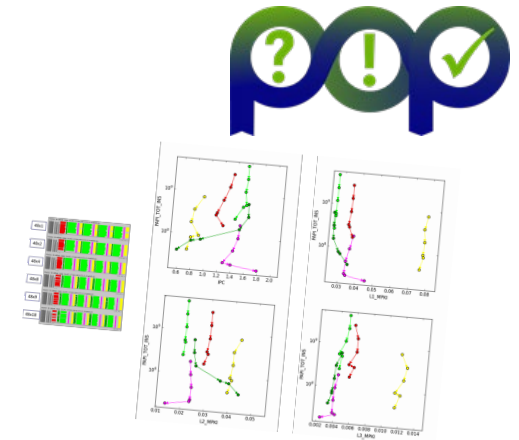
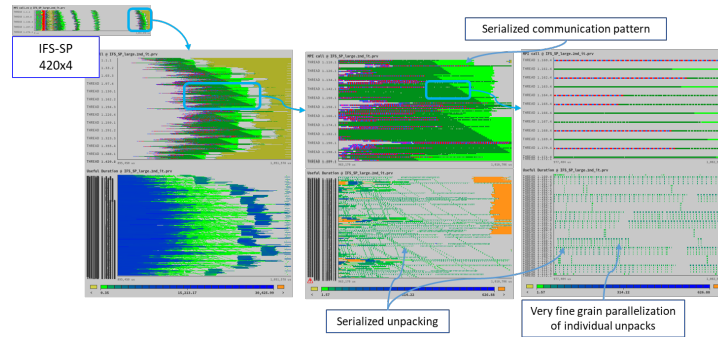
Schedulers

Compiler/Toolchain

OS

HW Systems

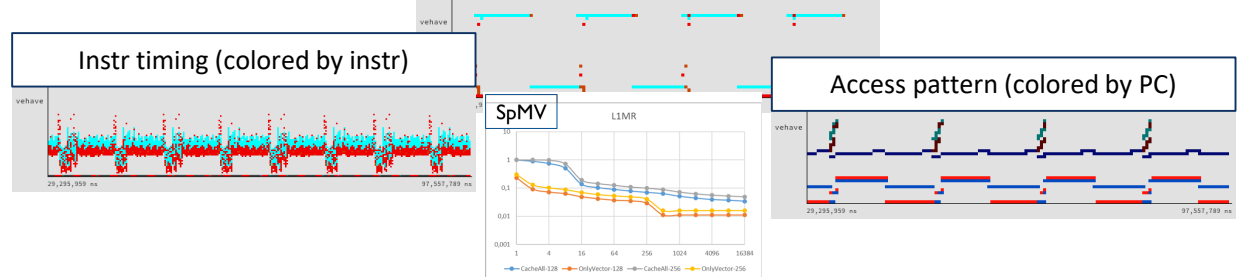
CPUs/GPUs/ASICs



LRU Stack distance (colored by instr)

Instr timing (colored by instr)

Access pattern (colored by PC)







**Barcelona  
Supercomputing  
Center**

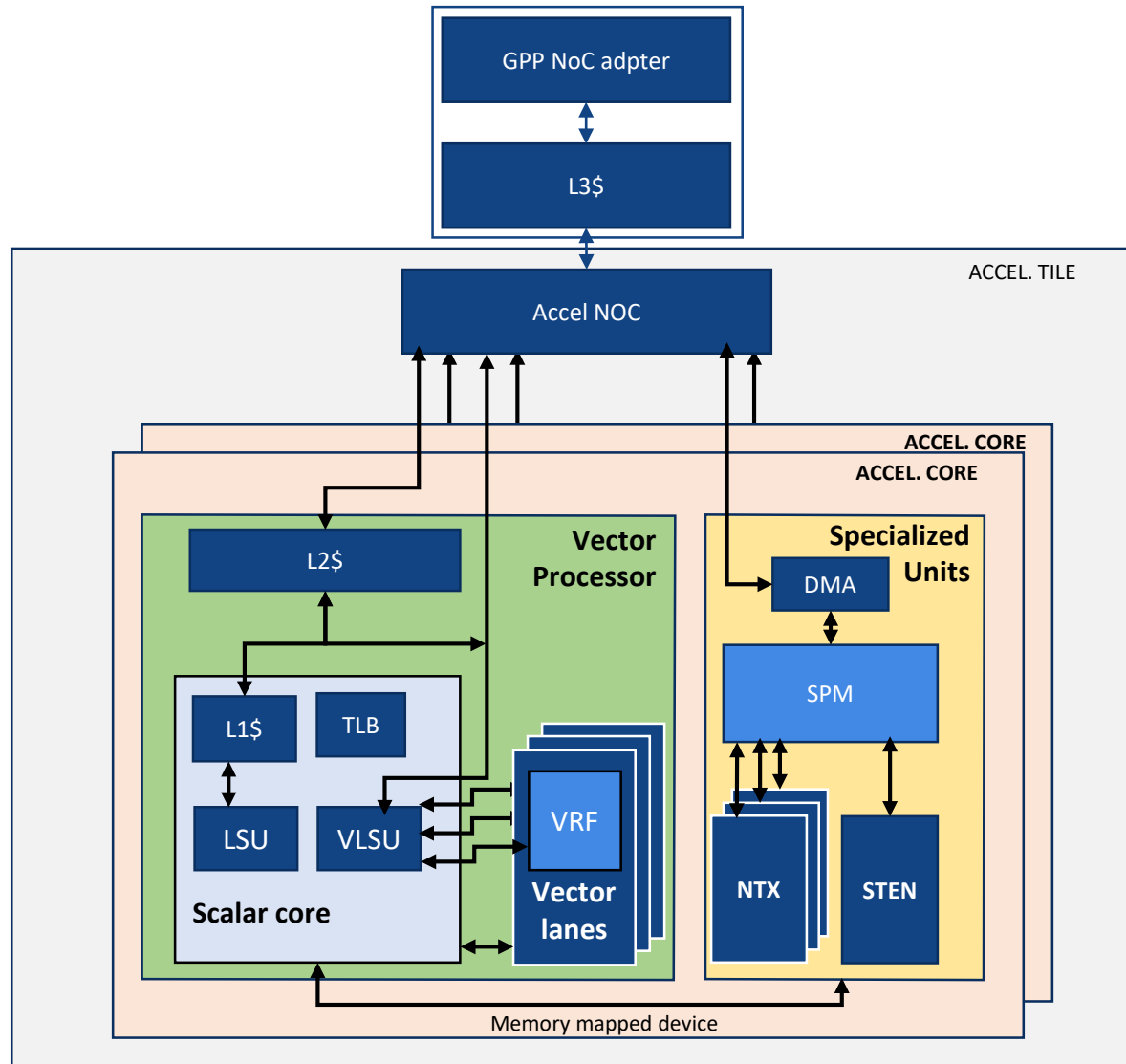
*Centro Nacional de Supercomputación*



**EXCELENCIA  
SEVERO  
OCHOA**

# EPAC overall design

# “Original EPAC Architecture”



# EPAC architecture

## RVV

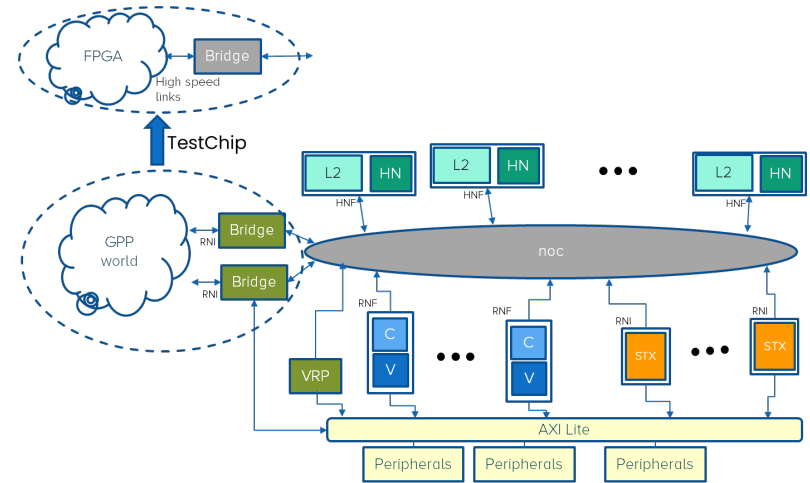
- RV64GCV (→ 8x)
  - 2 way in order core
  - Decoupled VPU
    - 8 lanes
    - Long vectors (256 DP elements)
- → 128 MSHR
- L1 - MESI coherency
- CHI interface NoC
  - 1 line / cycle
- L2\$: 256KB/module
  - Allocation control mechanisms
- No in tile L3\$

## STX

- DL and stencil specific accelerators
- Extensions to planned NTX
  - Programmable address generators →
  - lightweight RISC-v core + fat FPU + (Streaming Semantics & FREP)

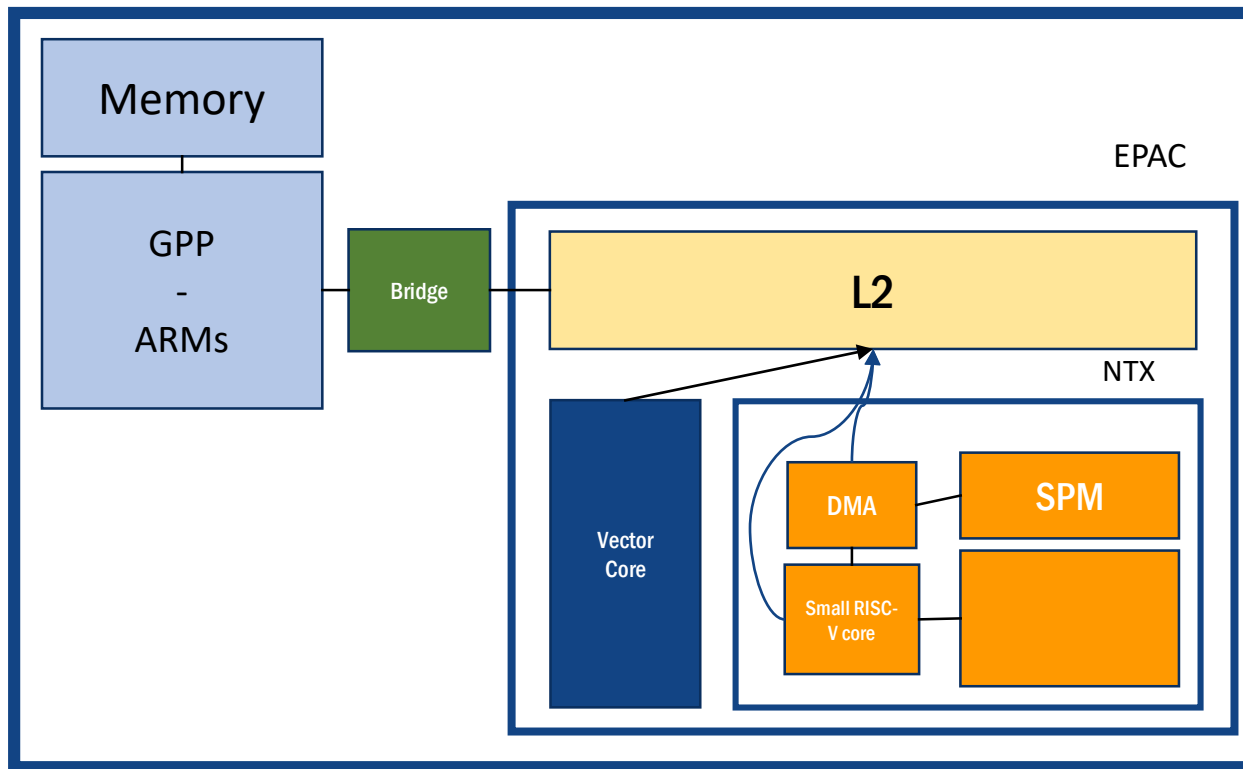
## VRP

- Variable precision processors



# EPI: A highly heterogeneous/hierarchical system

EPI

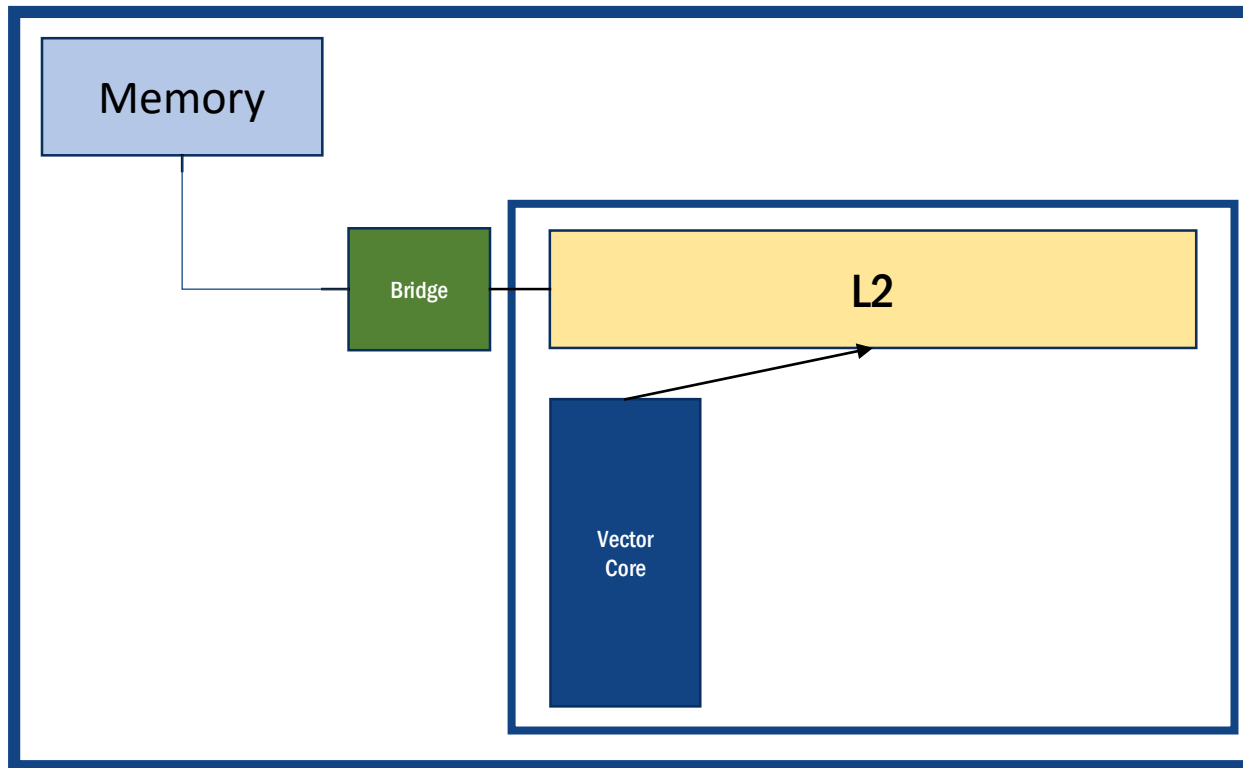


# EPI: A highly heterogeneous/hierarchical system



# Focus on ...

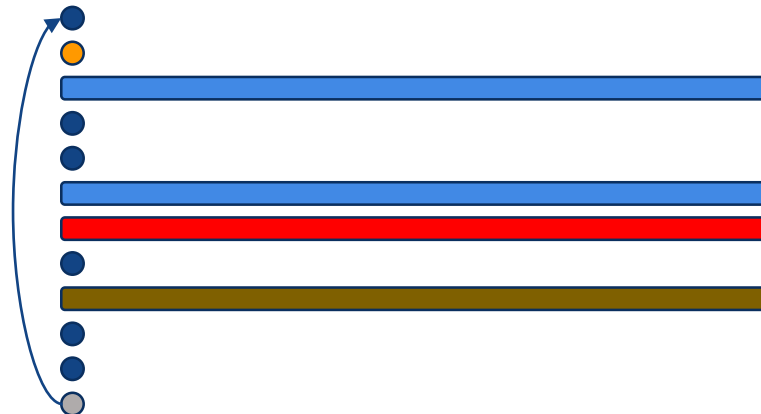
- RISC-V Long Vector
- Accelerator .... or not



# RVV VLA programming

```
void axpy_intrinsics (double a, double *dx, double *dy, int n) {
    int i;
    int gvl = __builtin_epi_vsetvl(n, __epi_e64, __epi_m1);
    __epi_1xf64 v_a = __builtin_epi_vbroadcast_1xf64(a, gvl);

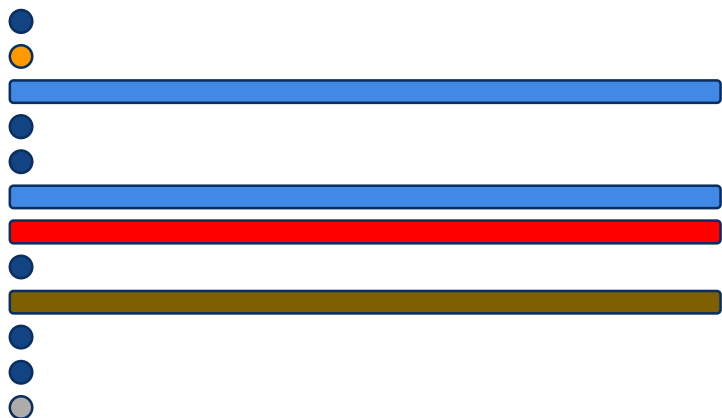
    for (i=0; i<n; ) {
        gvl = __builtin_epi_vsetvl(n - i, __epi_e64, __epi_m1);
        __epi_1xf64 v_dx = __builtin_epi_vload_1xf64(&dx[i], gvl);
        __epi_1xf64 v_dy = __builtin_epi_vload_1xf64(&dy[i], gvl);
        __epi_1xf64 v_res = __builtin_epi_vfmacc_1xf64(v_dy, v_a, v_dx, gvl);
        __builtin_epi_vstore_1xf64(&dy[i], v_res, gvl);
        i += gvl;
    }
}
```



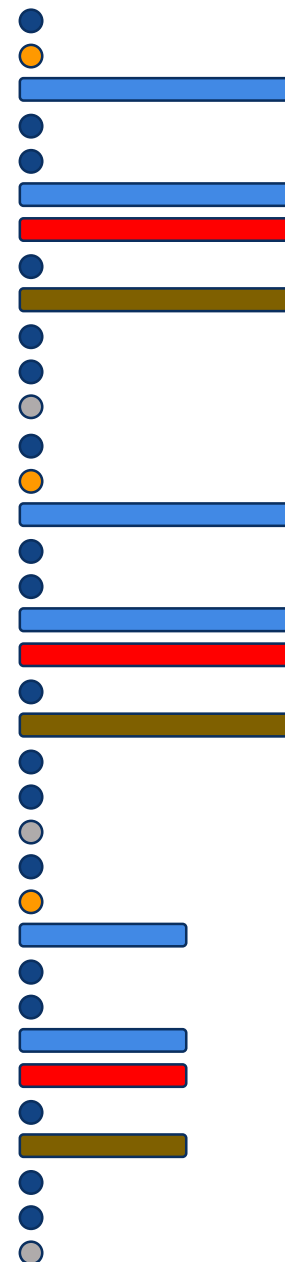


# RVV VLA programming

Execution on ideal machine



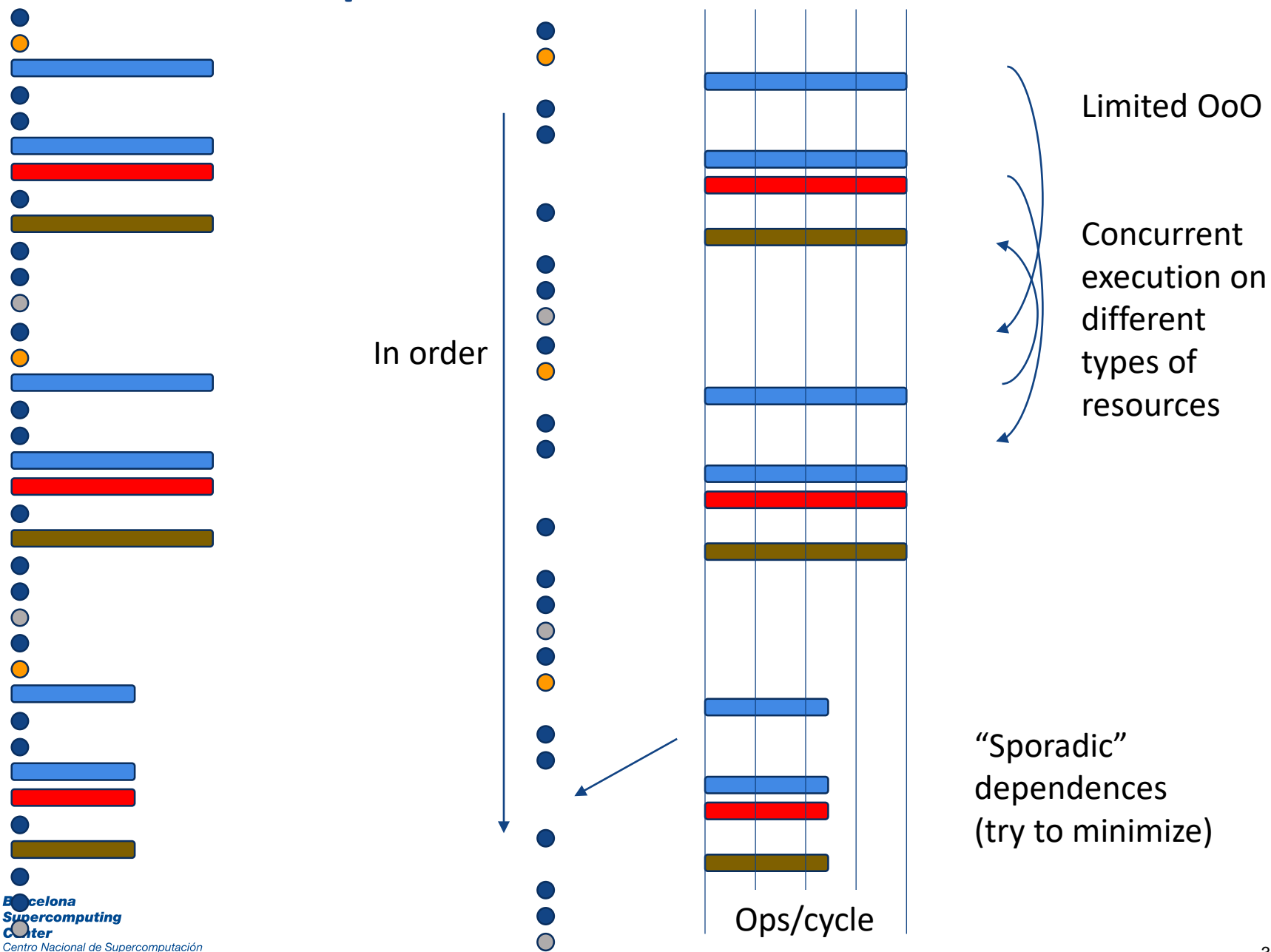
Execution on  
practical machine  
(finite MAXVL)



```
void axpy_intrinsics (double a, double *dx, double *dy, int n) {
    int i;
    int gvl = __builtin_epi_vsetvl(n, __epi_e64, __epi_m1);
    __epi_1xf64 v_a = __builtin_epi_vbroadcast_1xf64(a, gvl);

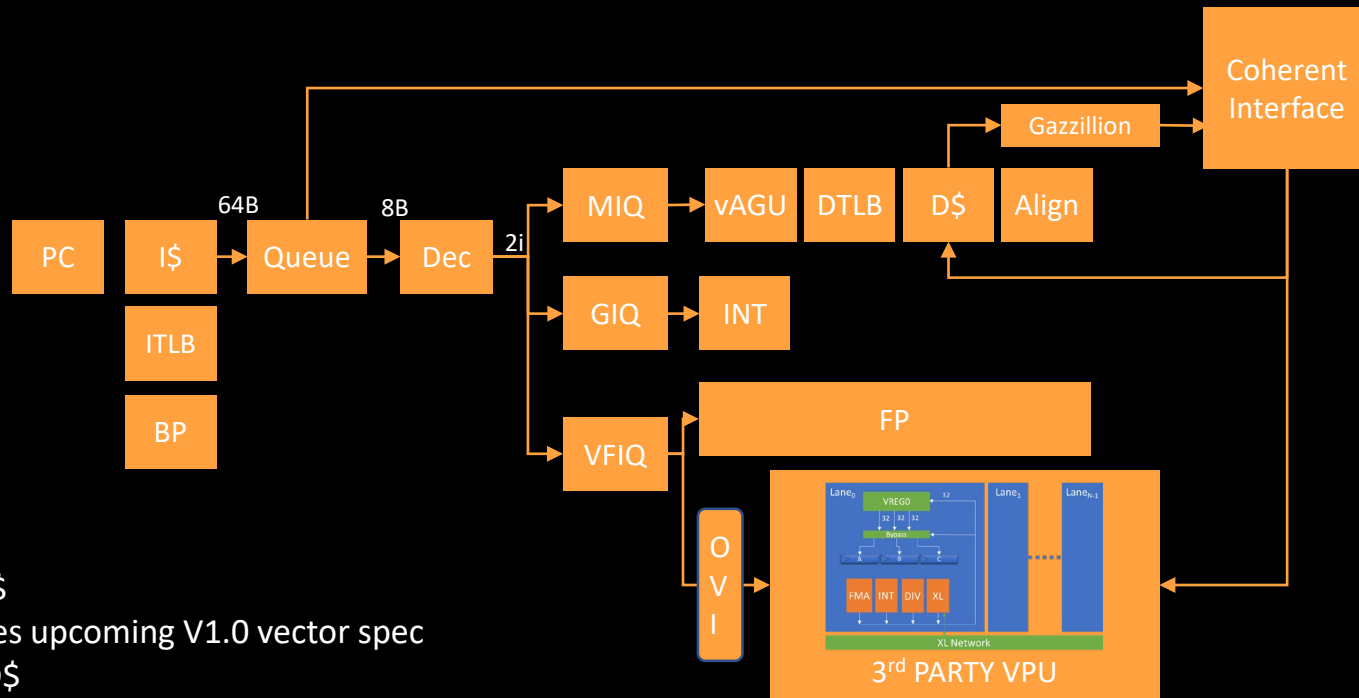
    for (i=0; i<n; ) {
        gvl = __builtin_epi_vsetvl(n - i, __epi_e64, __epi_m1);
        __epi_1xf64 v_dx = __builtin_epi_vload_1xf64(&dx[i], gvl);
        __epi_1xf64 v_dy = __builtin_epi_vload_1xf64(&dy[i], gvl);
        __epi_1xf64 v_res = __builtin_epi_vfmacc_1xf64(v_dy, v_a, v_dx, gvl);
        __builtin_epi_vstore_1xf64(&dy[i], v_res, gvl);
        i += gvl;
    }
}
```

# EPAC decoupled execution



# AVISPADO 220 with VPU

RISCV64GCV

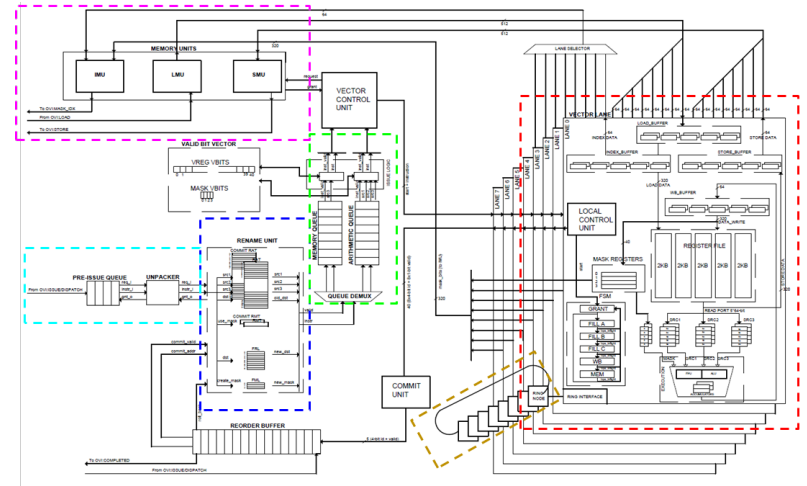


- SV48
- 16KB I\$
- Decodes upcoming V1.0 vector spec
- 32KB D\$
- Full hardware support for unaligned accesses
- Coherent (CHI)
- Vector Memory (vle, vlse, vlxe, vse, ...) processed by MIQ/LSU

Courtesy R. Espasa.

# VPU: A processor in itself

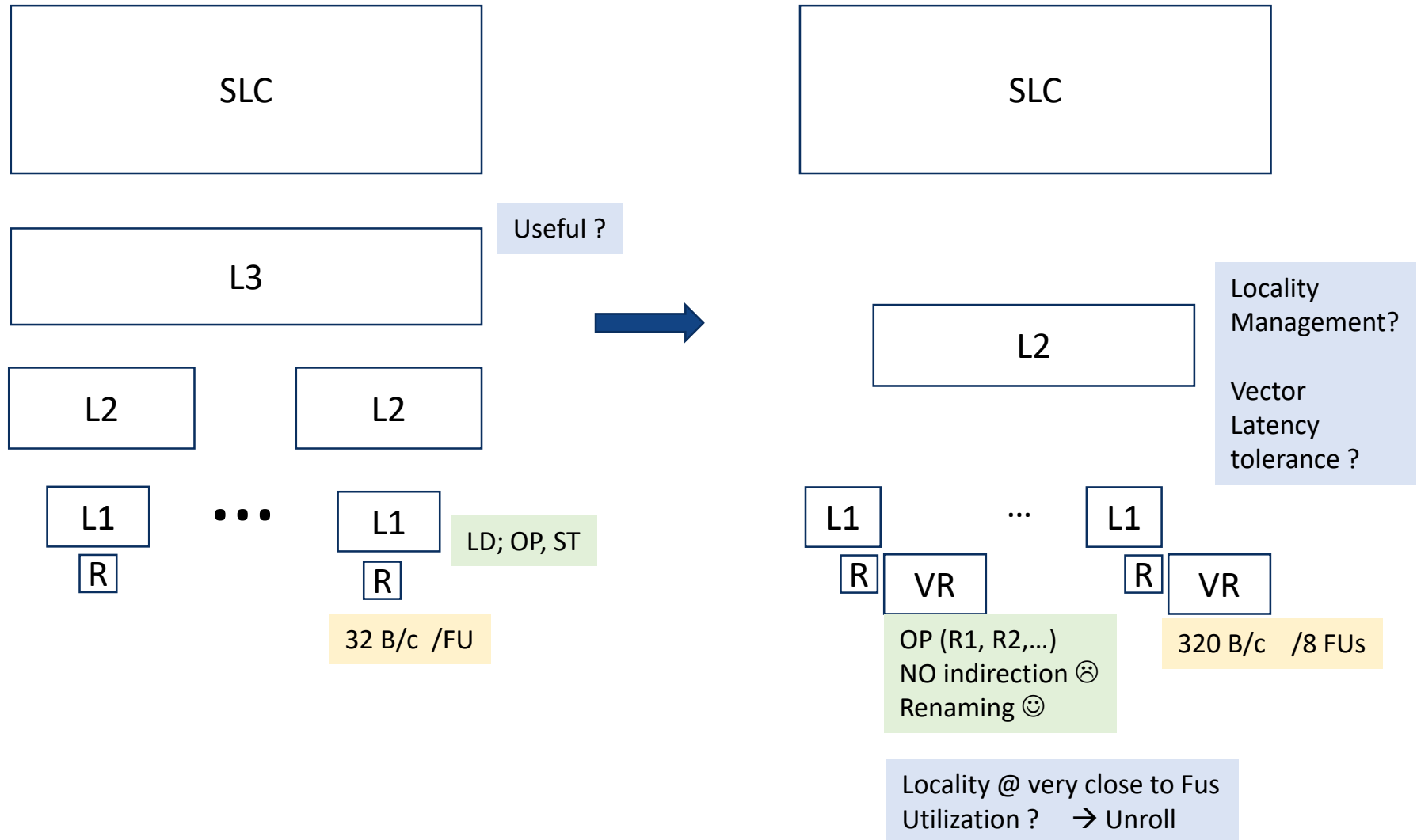
- Hierarchical “accelerator” integration
  - Program & data served by scalar core (Coherence; ~punch tape program 😊)
  - Fine grain “offloading” of “vector tasks” (directly hardware supported)
  - Homogenized heterogeneity under single “standard” ISA interface defining program order
- Implementation
  - **#FUs << VL** (lanes=8, VL=256)
  - Some OoO
    - Resources to overlap?
      - L/S, FU, shuffling
    - Renaming
      - 40 physical registers
  - Single ported register file
    - Large state
    - 5 banks/lane providing sufficient bandwidth for 1 op/cycle (latency/BW trading)
  - Data shuffling: directional ring



“Vitruvius: An Area-Efficient Decoupled Vector Accelerator for High Performance Computing”

F. Minervini, O. Palomar. RISC-V Summit 2021

# A bit on state hierarchy & locality



A more efficient way to use “comparable” amount of resources with less control flows ?



**Barcelona  
Supercomputing  
Center**  
*Centro Nacional de Supercomputación*



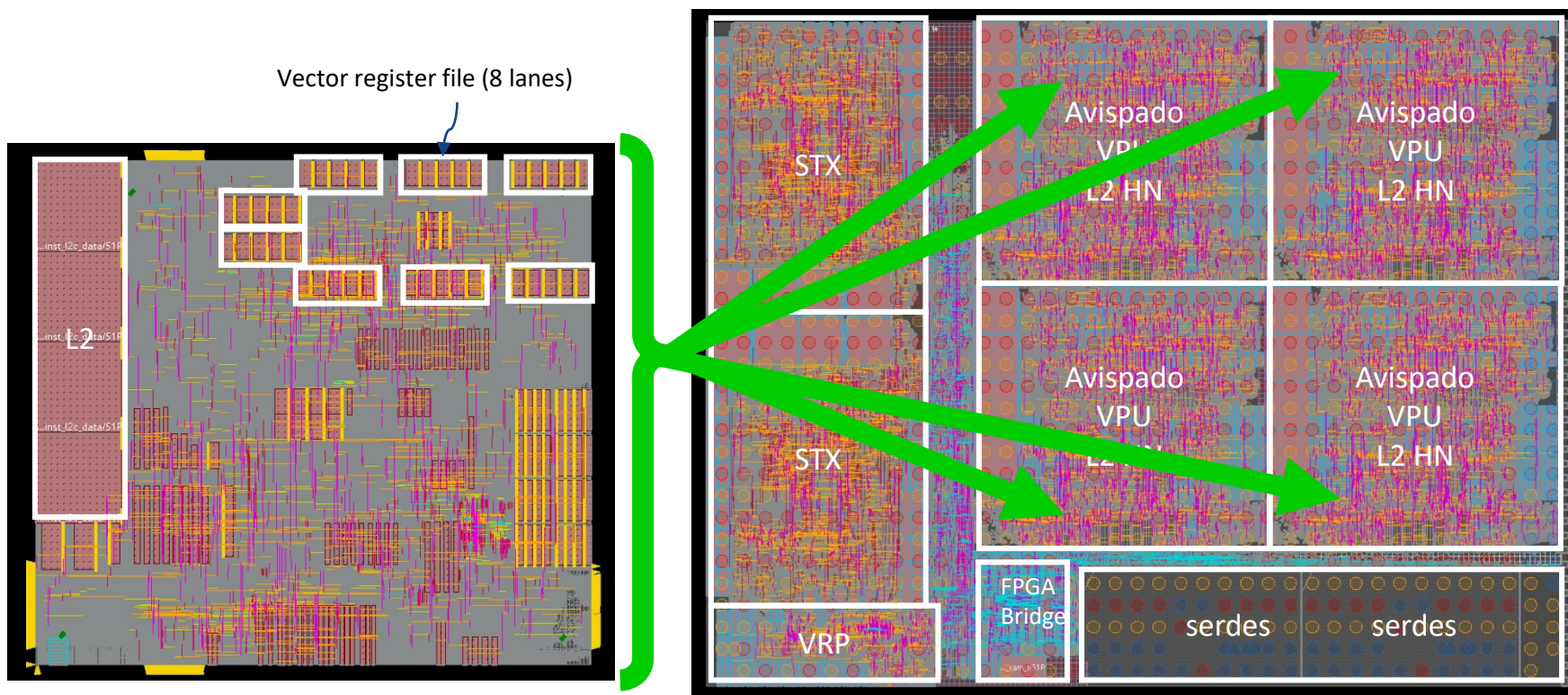
**EXCELENCIA  
SEVERO  
OCHOA**

# Test Chip & Software Development Vehicles (SDVs)



# Physical design

- 4 “VPU microtiles”
  - 2.517 mm<sup>2</sup> each





# EPAC Test Chip




```
happy@epac$ axpy 1024
Running AXPY Scalar with 1024 array elements
init time: 45060 cycles

axpy scalar reference time 23555 cycles

done
Result ok !!!
happy@epac$ vaxpy 1024
Running AXPY Vector with 1024 array elements
init time: 45043 cycles

axpy vector time 932 cycles

done
Result ok !!!
happy@epac$
```

~25x   
while only 8x FPUs  
→ Long vectors !!  
→ Memory Bandwidth

```
epac@EPAC:~/Desktop/etc_tools-master_v20211206/etc_tools-m
[sudo] password for epac:
Waiting for server ...
Waiting for server ...
Waiting for server ...
Waiting for server ...
Waiting for server ...
Waiting for server ...
Waiting for server ...
Connection with server established!
EPAC JTAG Console Client 0.1
Connecting to JTAG Console [3] ...
Press CTRL+A for exit

| Welcome to EPAC TC Bring-Up Shell |

user@epac$ jpeg_benchmark

JPEG scalar reference time: 255667444 cycles

done
user@epac$
```

VRP

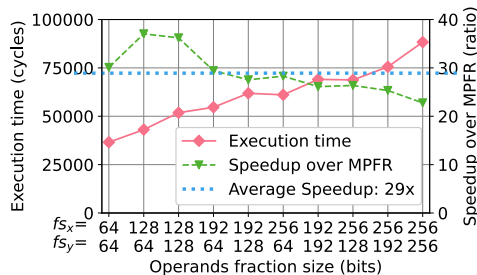
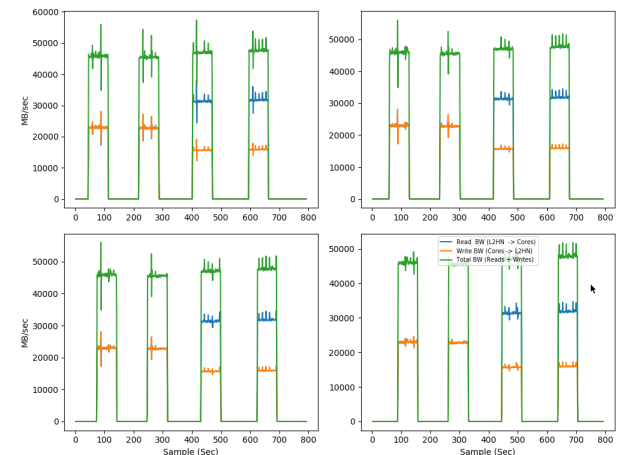


Table 1 Latency and energy per instruction

Instruction	Latency (cycles)		Energy (pJ/instruction)	
	Min	Max	Min	Max
VPADD/VPSUB	11	33	378.72	507.70
VPMUL	12	37	307.83	415.66
VPCMP	5	8	243.60	248.98
VPLOAD	16	42	604.99	1550.30
VSTORE	20	30	1023.88	1795.26
FADD	6	-	117.76	-
FMUL	6	-	126.70	-

Min and max values use operands with significant sizes of 64b and 256b, respectively

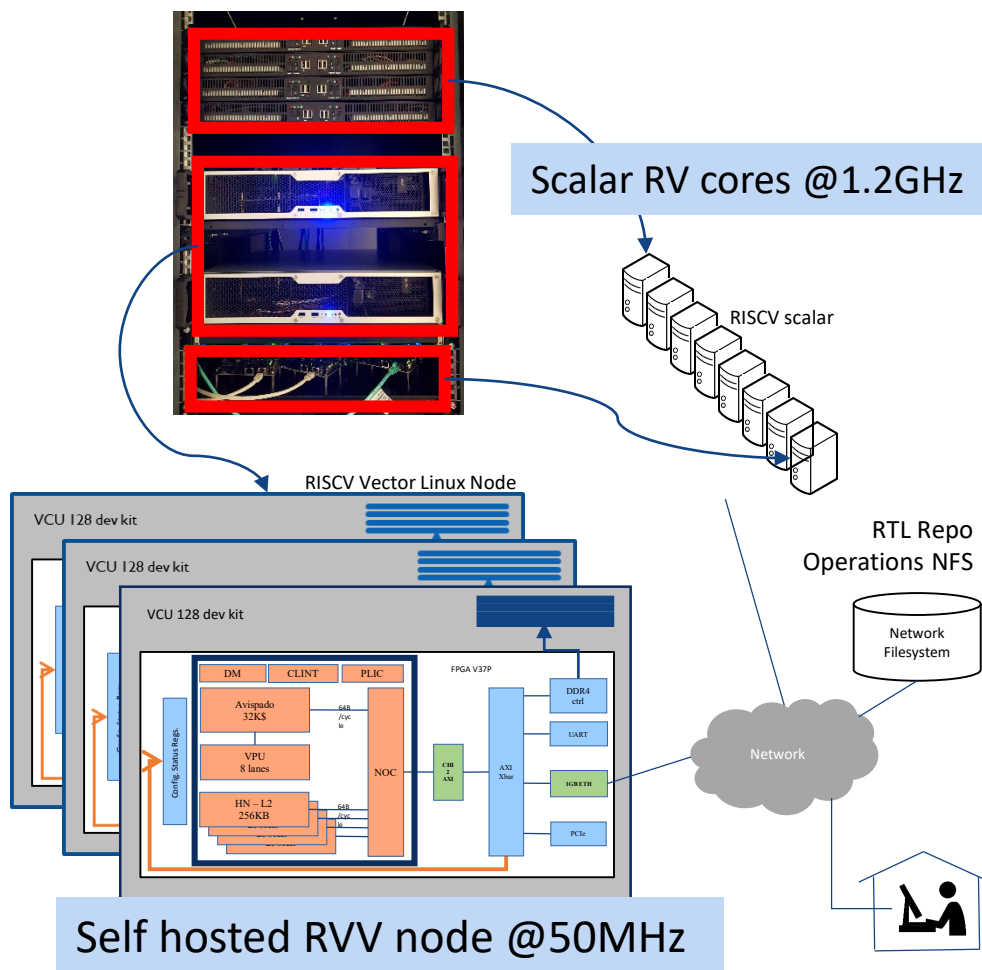
copy scale add triad



# RVV @ FPGA & ecosystem

- HPC software stack @ Commercially available RISC-V platforms
  - SLURM, MPI, OpenMP, BSC tools, RVV software emulator
- EPI SDV platforms
  - Linux
  - Test user codes @ real RTL
  - Give to EPI partners and external users early access to EPI technology
- Holistic CI/CD framework
  - HW & SW
  - Functionality & performance

Contact : [filippo.mantovani@bsc.es](mailto:filippo.mantovani@bsc.es)



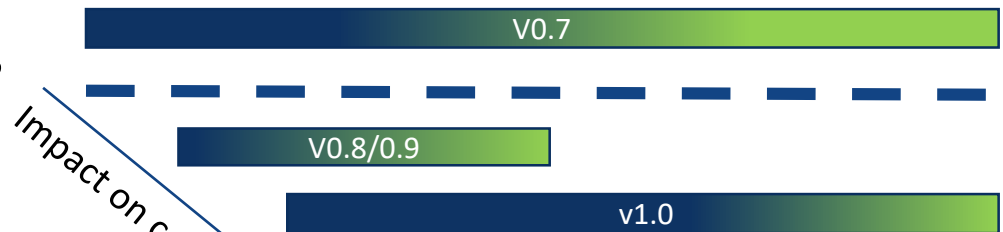
# RISC-V Vector extension (RVV) Compiler

- LLVM support for the evolution of the RISC-V Vector (RVV) Extension



- Intrinsics
- Vectorization/SIMD clauses
- Autovectorization

Support EPAC RTL  
SDV@FPGA / Test Chip



Support EPAC RTL  
SDV3/EPAC2.0

```
void SpMV_vec(double *a, long *ia, long *ja, double *x, double *y, int nrows) {
    for (int row = 0; row < nrows; row++) {
        int nnz_row = ia[row + 1] - ia[row];
        unsigned long gvl; // granted vector lengths
        int idx = ia[row];
        __epi_1xf64 v_a, v_x, v_prod, v_partial_res;
        __epi_1xi64 v_idx_row, v_three;
        y[row]=0.0;
        v_partial_res = __builtin_epi_vbroadcast_1xf64(0.0, gvl);
        for(int colid=0; colid<nnz_row; ) { //blocking on MAXVL
            gvl = __builtin_epi_vsetvl(nnz_row - colid, __epi_e64,
                v_a = __builtin_epi_vload_1xf64(&a[idx+colid], gvl);
                v_idx_row = __builtin_epi_vload_1xi64(&ja[idx+colid], gvl);
                v_three = __builtin_epi_vbroadcast_1xi64(3, gvl);
                v_idx_row = __builtin_epi_vsl_1xi64(v_idx_row, v_three, gvl)
                v_x = __builtin_epi_vload_indexed_1xf64(x, v_idx_row, g
                v_prod = __bu
                v_partial_res =
            colid += gvl;
        }
        y[row] += __builtin_e
    }
}
```

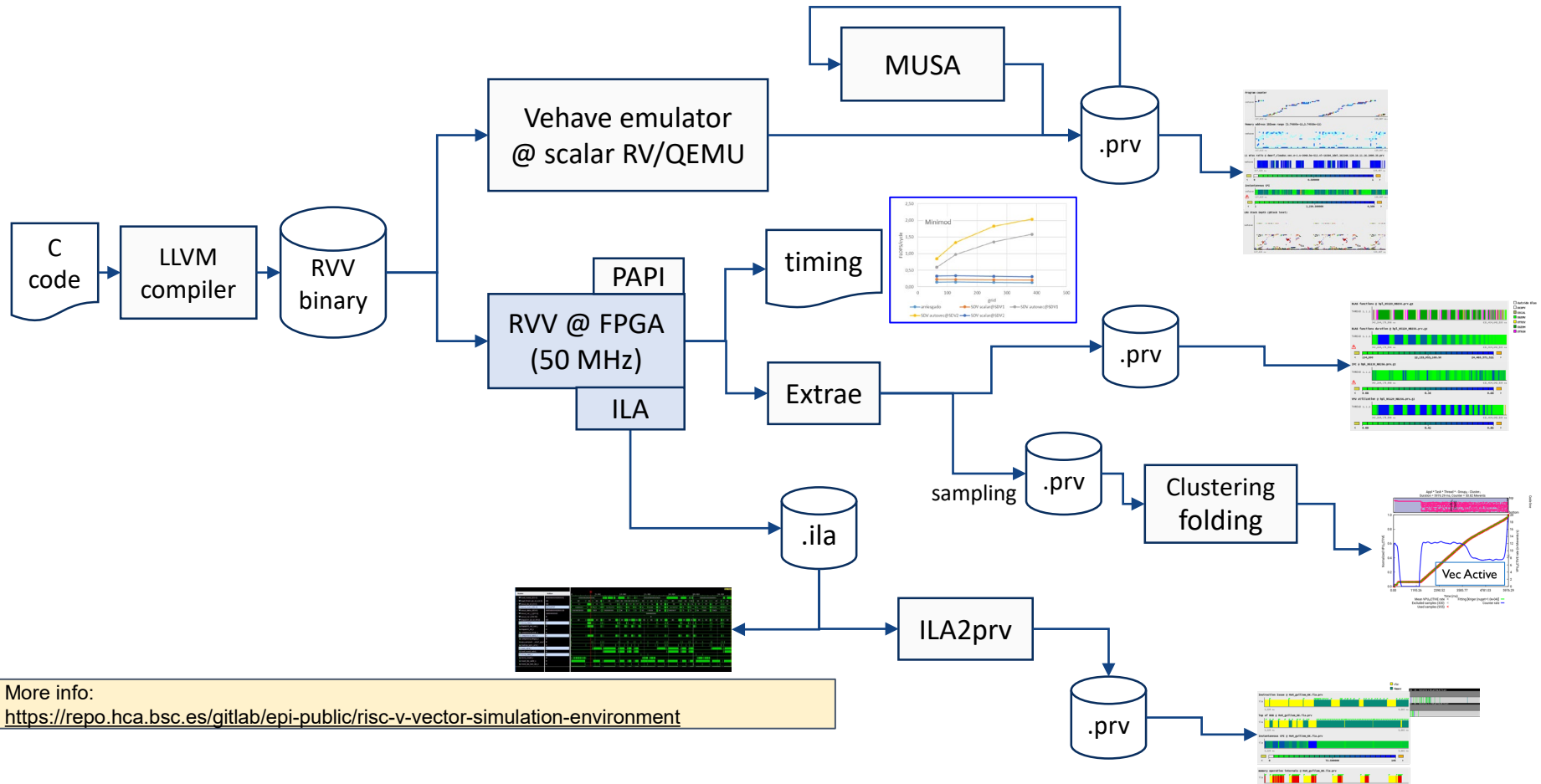
```
void target_inner_3d (...) {
    for (i) {
        for (j) {
            #pragma clang loop vectorize(enable)
            for (k) {
                lap = LAP;
                v[IDX3_1(i,j,k)] = 2.f*u[IDX3_1(i,j,k)]
                    -v[IDX3_1(i,j,k)]+vp[IDX3(i,j,k)]*lap;
            }
        }
    }
}
```

```
float complex A[n][n];
float complex templ, temp2;
...
for (int j = 0; j < n; j++) {
    if (x[j] != ZERO || y[j] != ZERO) {
        templ = alpha * conjunction(creal(y[j]), cimag(y[j]));
        temp2 = conjunction(creal(alpha * x[j]), cimag(alpha * x[j]));
        #pragma clang loop vectorize(assume_safety)
        for (int i = 0; i <= j - 1; i++) A[i][j] = A[i][j] + x[i] * templ + y[i] * temp2;
        A[j][j] = creal(A[j][j]) + creal(x[j] * templ + y[j] * temp2);
    } else A[j][j] = creal(A[j][j]);
}
```

# SDV flows

App development & Data acquisition

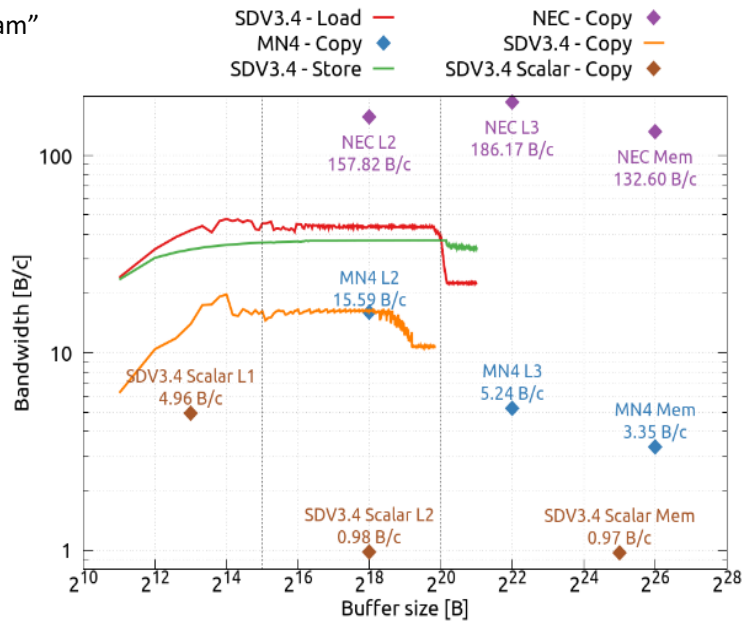
Analysis



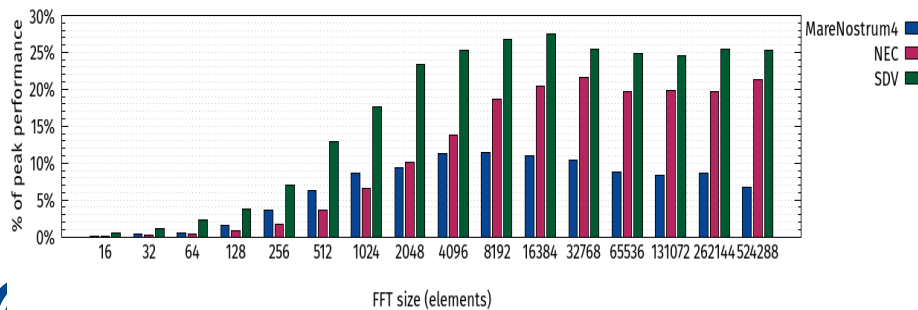
# SDV@FPGA – vector performance

- EPAC – RVV vs. state of the art

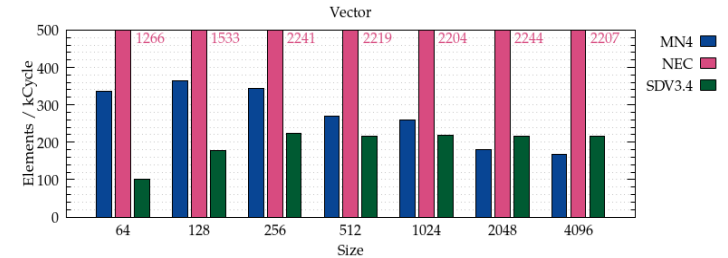
“Stream”



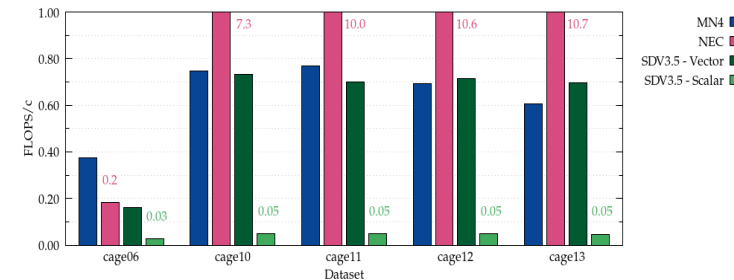
FFT



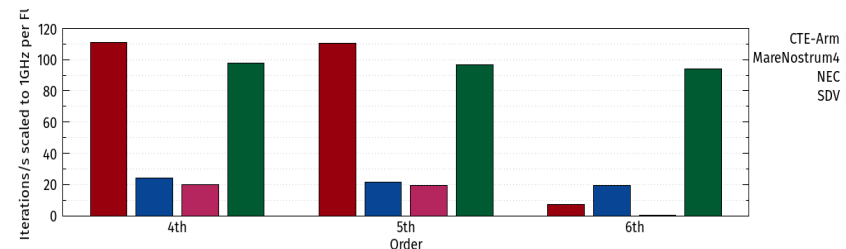
Jacobi 2D



SpMV

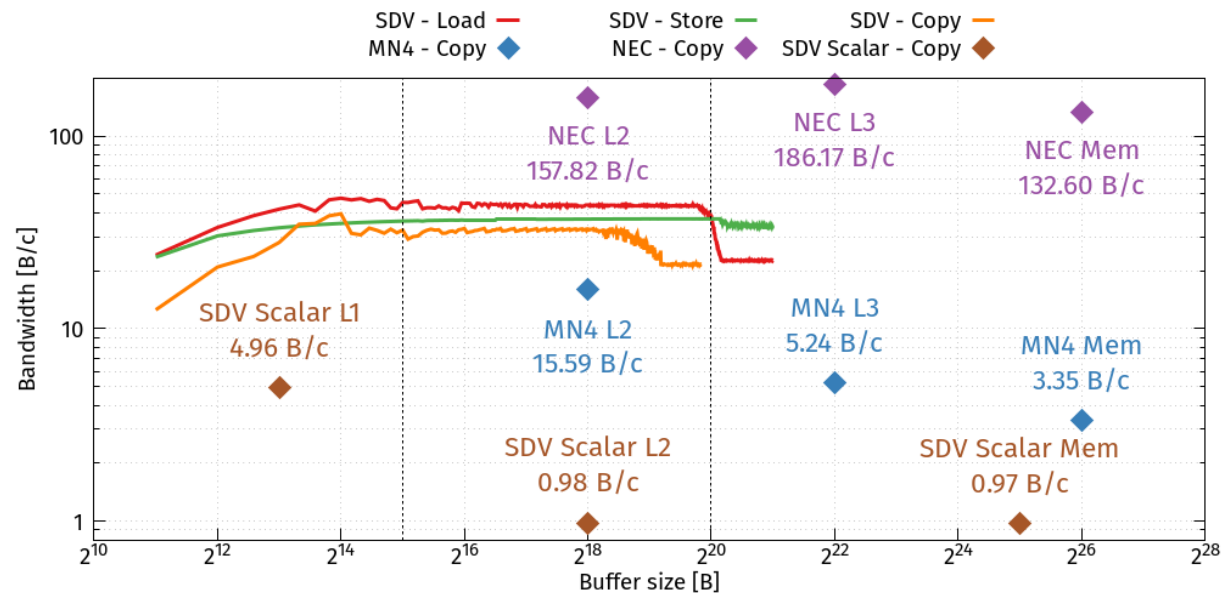
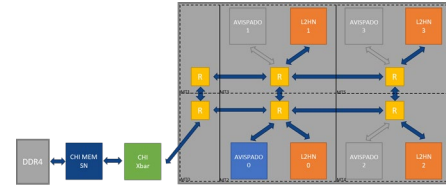


HACC



# Observed Memory Bandwidth

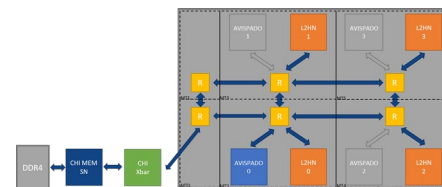
- Single core Load, Store, Copy
- Comparison to scalar and other architectures
- Comparison to their architectural peak



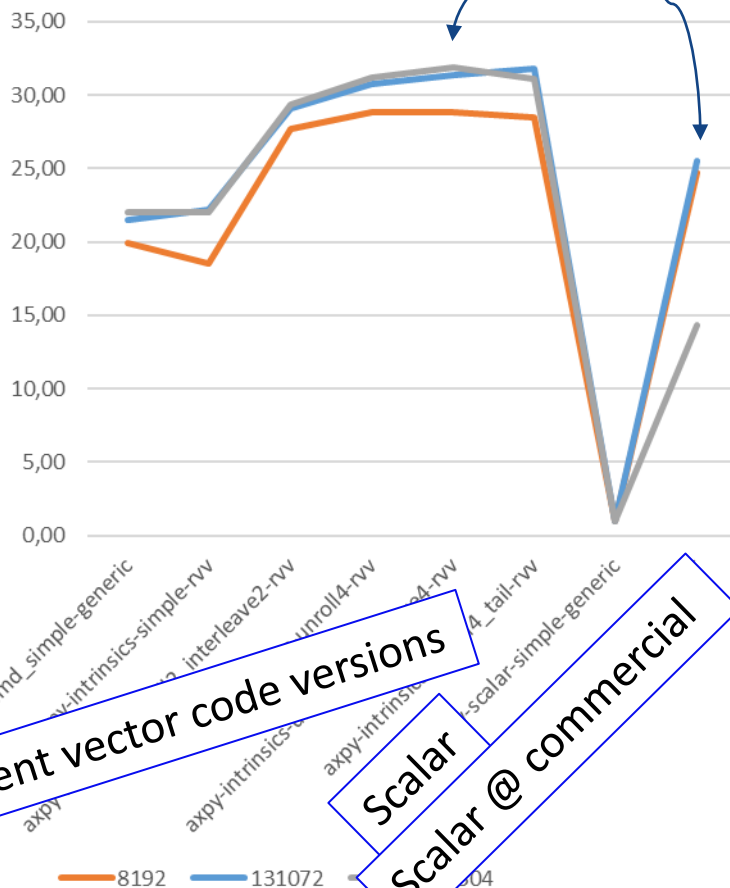
# Continuous Co-design

Axpy

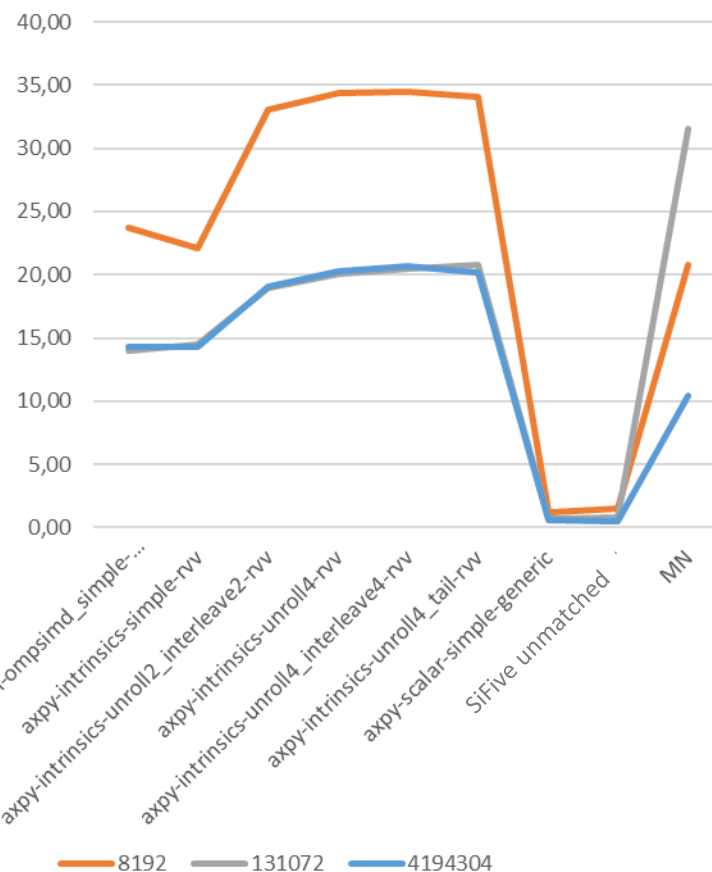
FPGA 50 MHz vs ASIC 1 GHz



Speedup vs base EPAC scalar

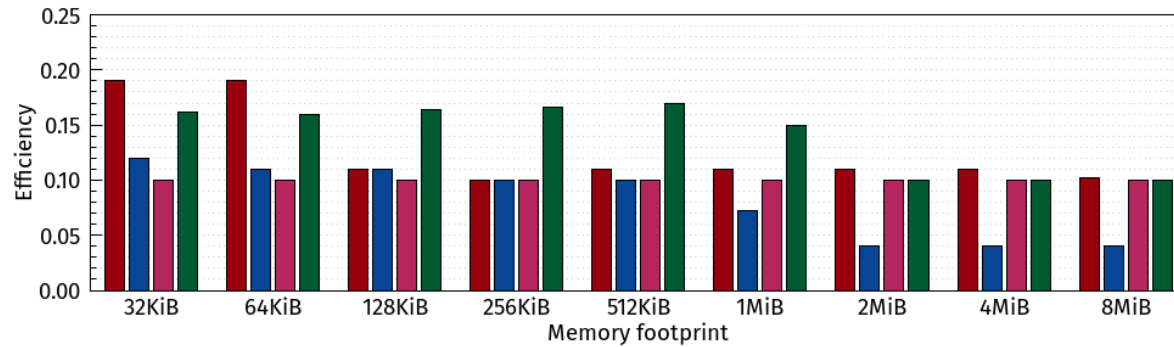
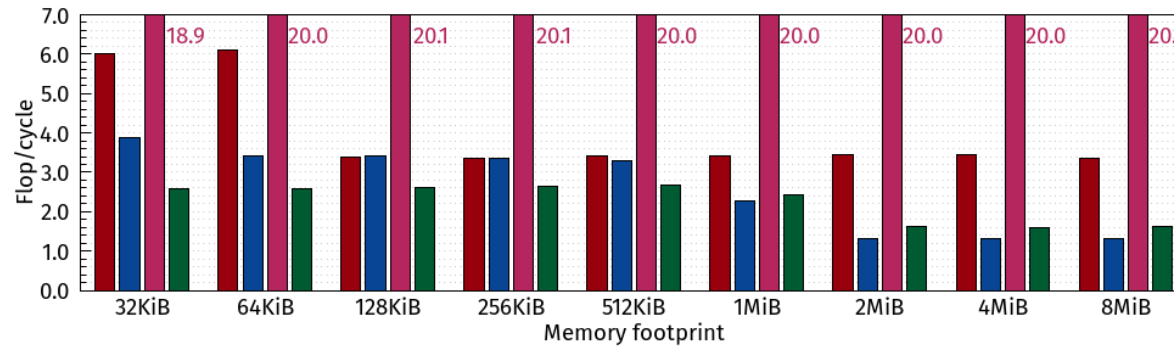
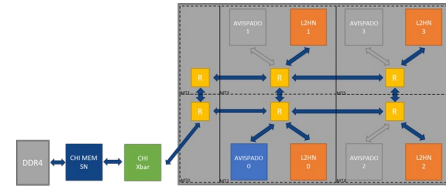


Bytes/cycle



# Continuous Co-design

- Daxpy



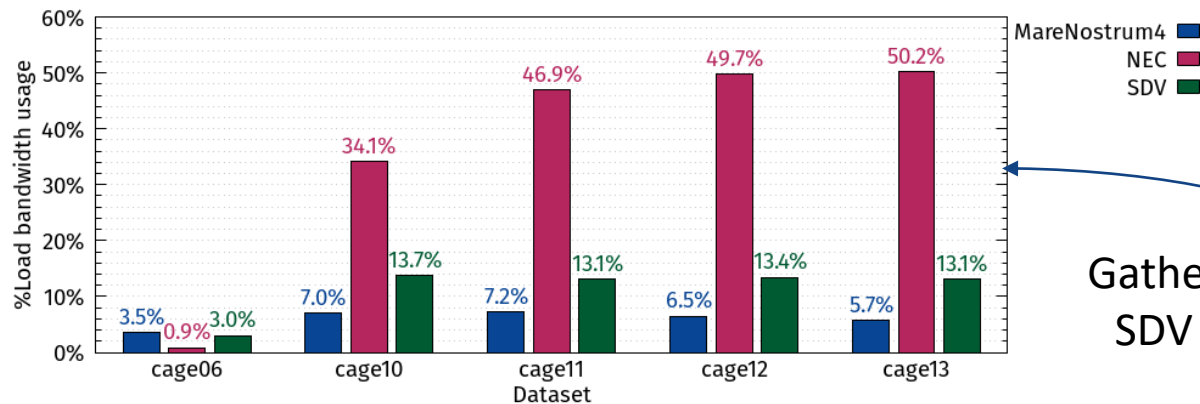
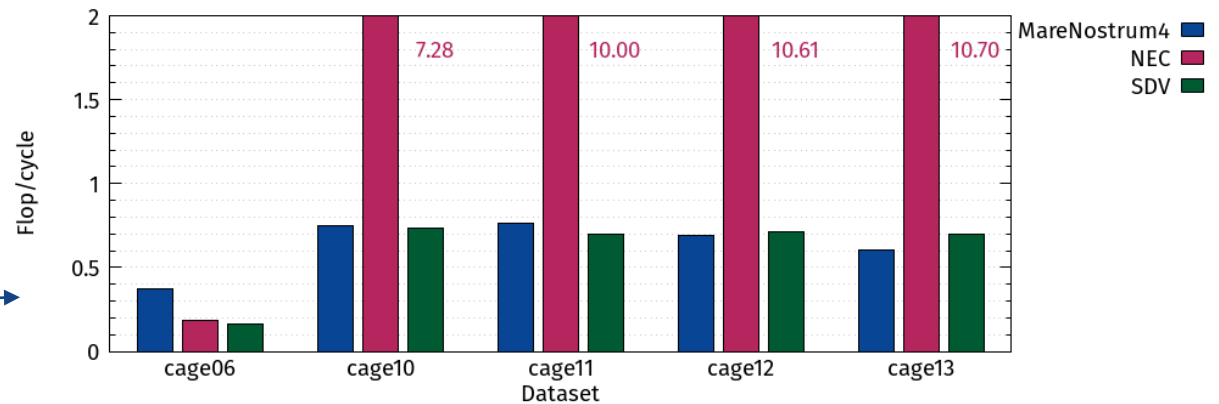
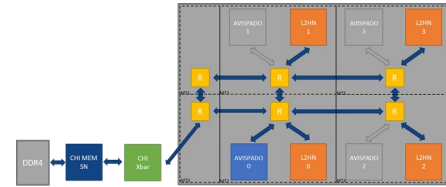
CTE-Arm  
MareNostrum4  
NEC  
SDV

CTE-Arm  
MareNostrum4  
NEC  
SDV



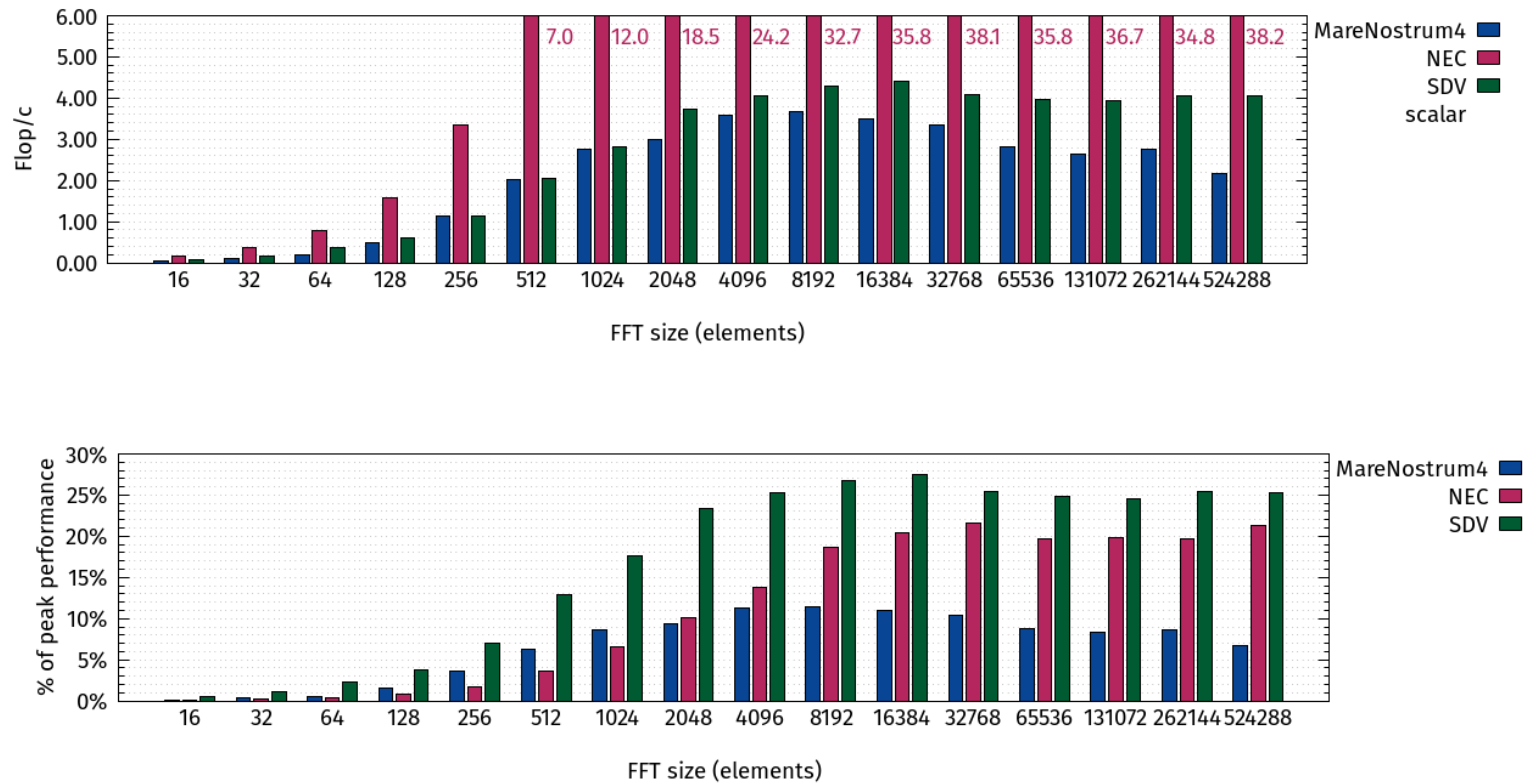
# Continuous Co-design

- SpMV
  - MKL / NLC / Ellpack based implementation



# Continuous Co-design

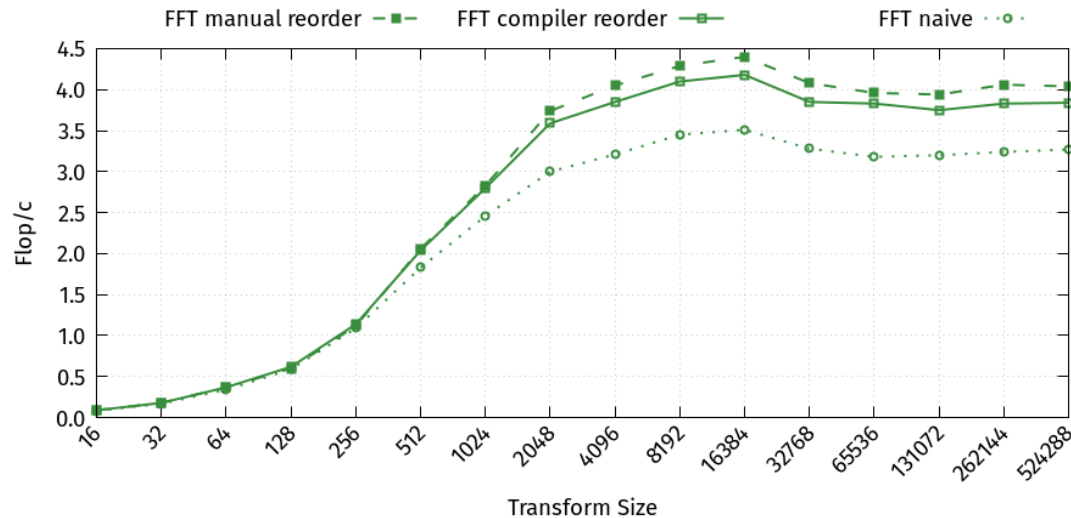
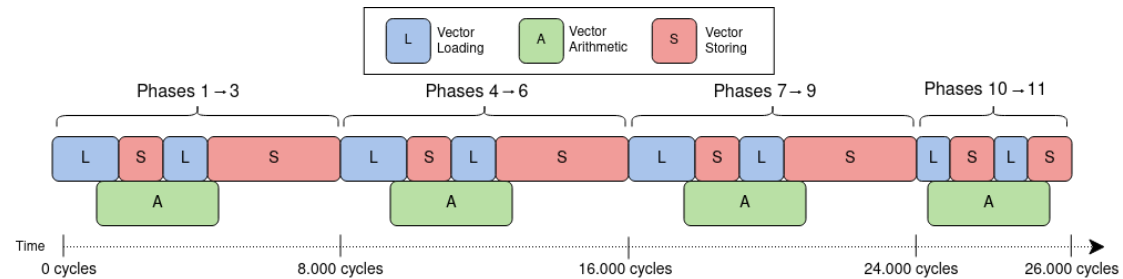
- FFT



# Continuous Co-design

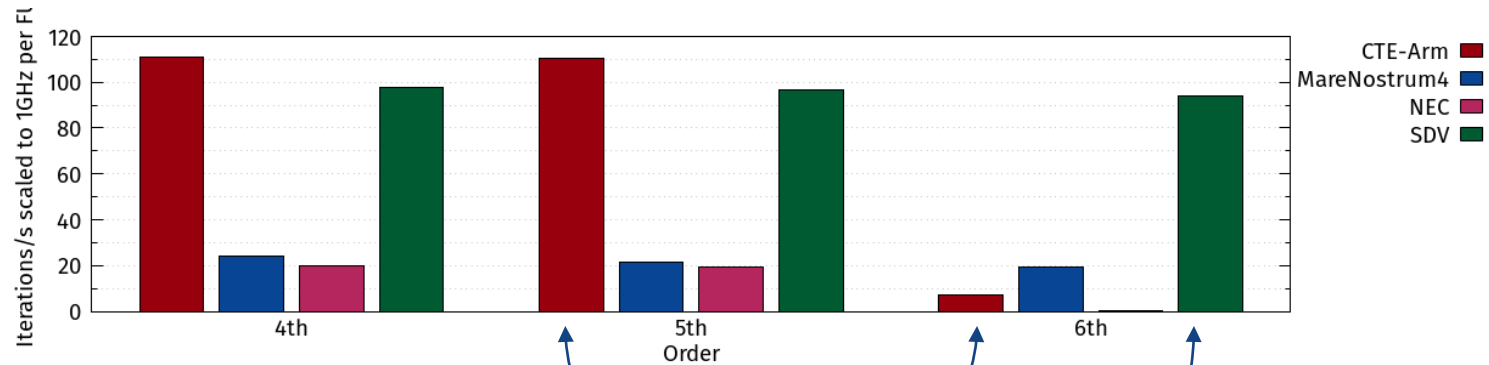
- FFT

Instruction  
scheduling



# Continuous Co-design

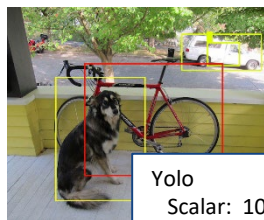
- HACC



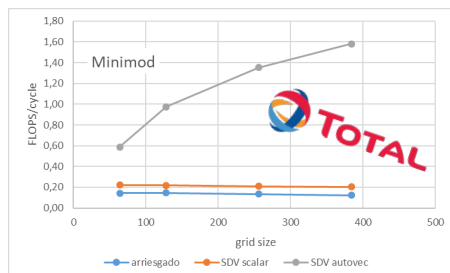
Why?

Compiler vectorization:  
Relevant technique?

# Vector in HPC and beyond

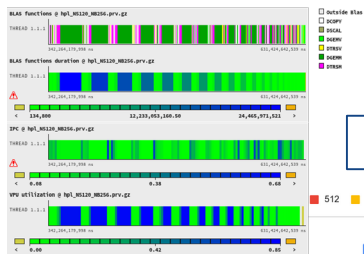


Yolo  
Scalar: 1035.505 s  
Vector: 29.377 s

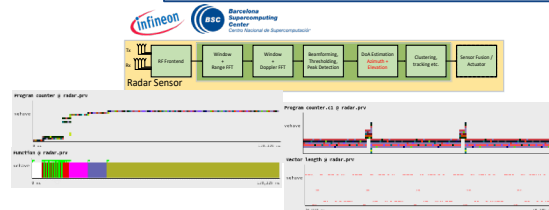
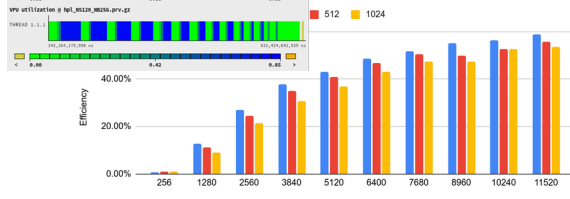


Bolt

Intel (2.5GHz): 0.0303 s  
Unmatched (1.2GHz): 0.1634 s  
SDV Scalar (50MHz): 5.8462 s  
SDV Vector (50MHz): 3.7104 s



Linpack



@vehave. WIP: backporting vectorization to 0.7

THE EUPILOT

- GROMACS
- EC-EARTH
- oneDNN



• BLIS



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

• Ginkgo



- Climate dwarfs
- TFLite
- Containers
- PyCOMPSs
- Spark

- Pytorch
- PostgreSQL
  - Sorting



**Barcelona  
Supercomputing  
Center**  
*Centro Nacional de Supercomputación*



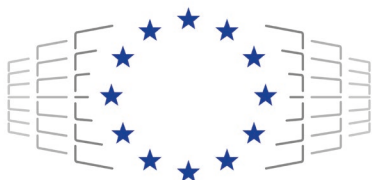
**EXCELENCIA  
SEVERO  
OCHOA**

# Thanks

# EPI PARTNERS



# EPI FUNDING



**EuroHPC**  
Joint Undertaking

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



Federal Ministry  
of Education  
and Research



Fundação  
para a Ciência  
e a Tecnologia

**VINNOVA**  
Sweden's Innovation Agency



REPUBLIC OF CROATIA  
Ministry of Science and  
Education



Swedish  
Research  
Council



Financé  
par



**Barcelona  
Supercomputing  
Center**

Centro Nacional de Supercomputación