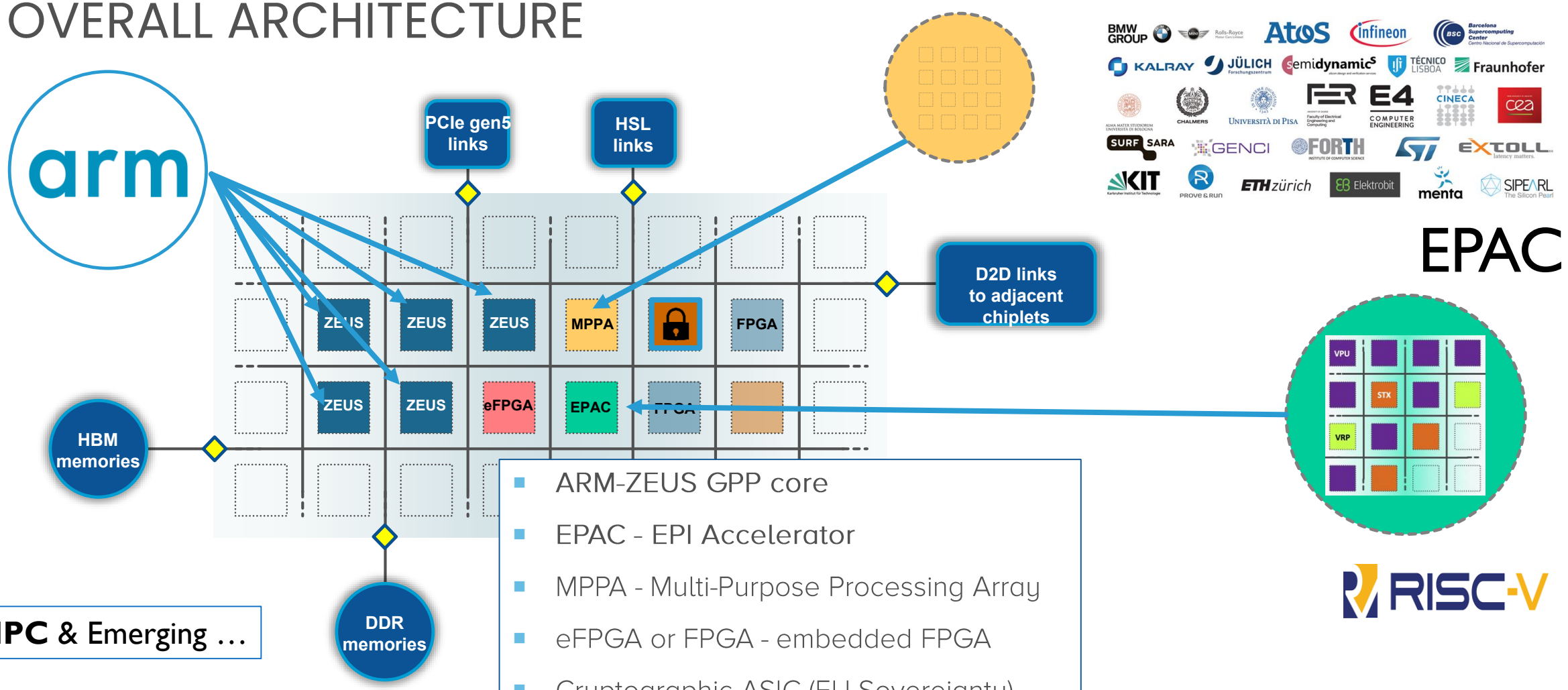
THE RISC-V VECTOR PROCESSOR IN EPI

JESUS LABARTA (@BSC.ES)

OVERALL ARCHITECTURE

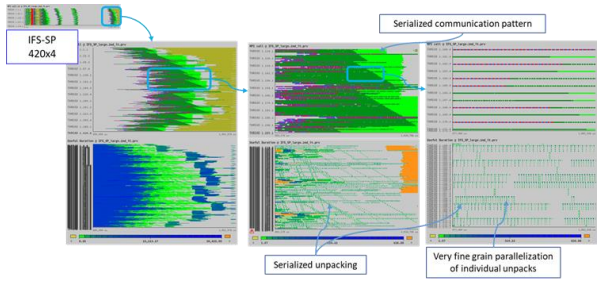


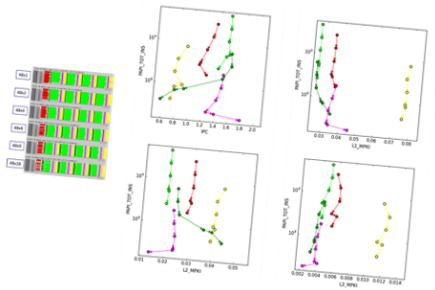
- ARM-ZEUS GPP core
- EPAC - EPI Accelerator
- MPPA - Multi-Purpose Processing Array
- eFPGA or FPGA - embedded FPGA
- Cryptographic ASIC (EU Sovereignty)

HPC & Emerging …

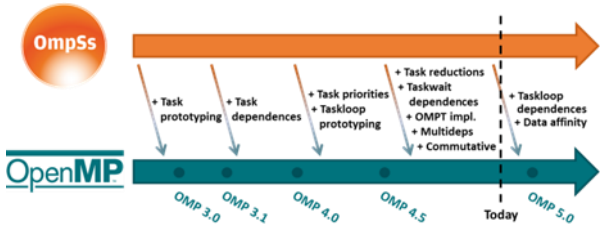
VISION CONTEXT

Insight on behavior





The programmer interface

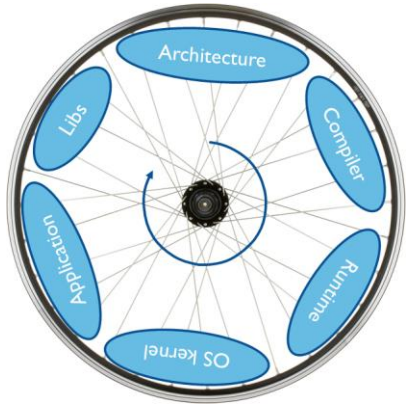


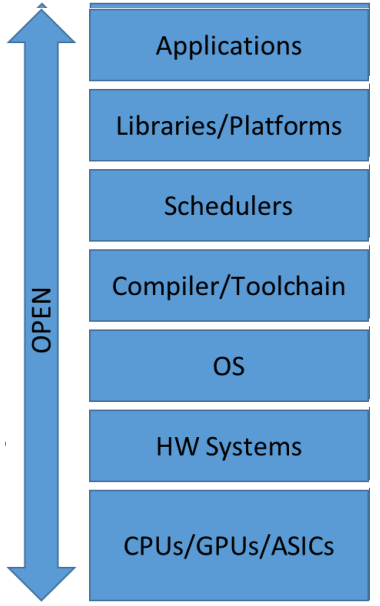
RISC-V

Leverage standards
Opportunity to innovate

- Balanced hierarchy
- Latency →throughput: asynchrony and overlap
- Malleability & coordinated scheduling
- Homogenize heterogeneity

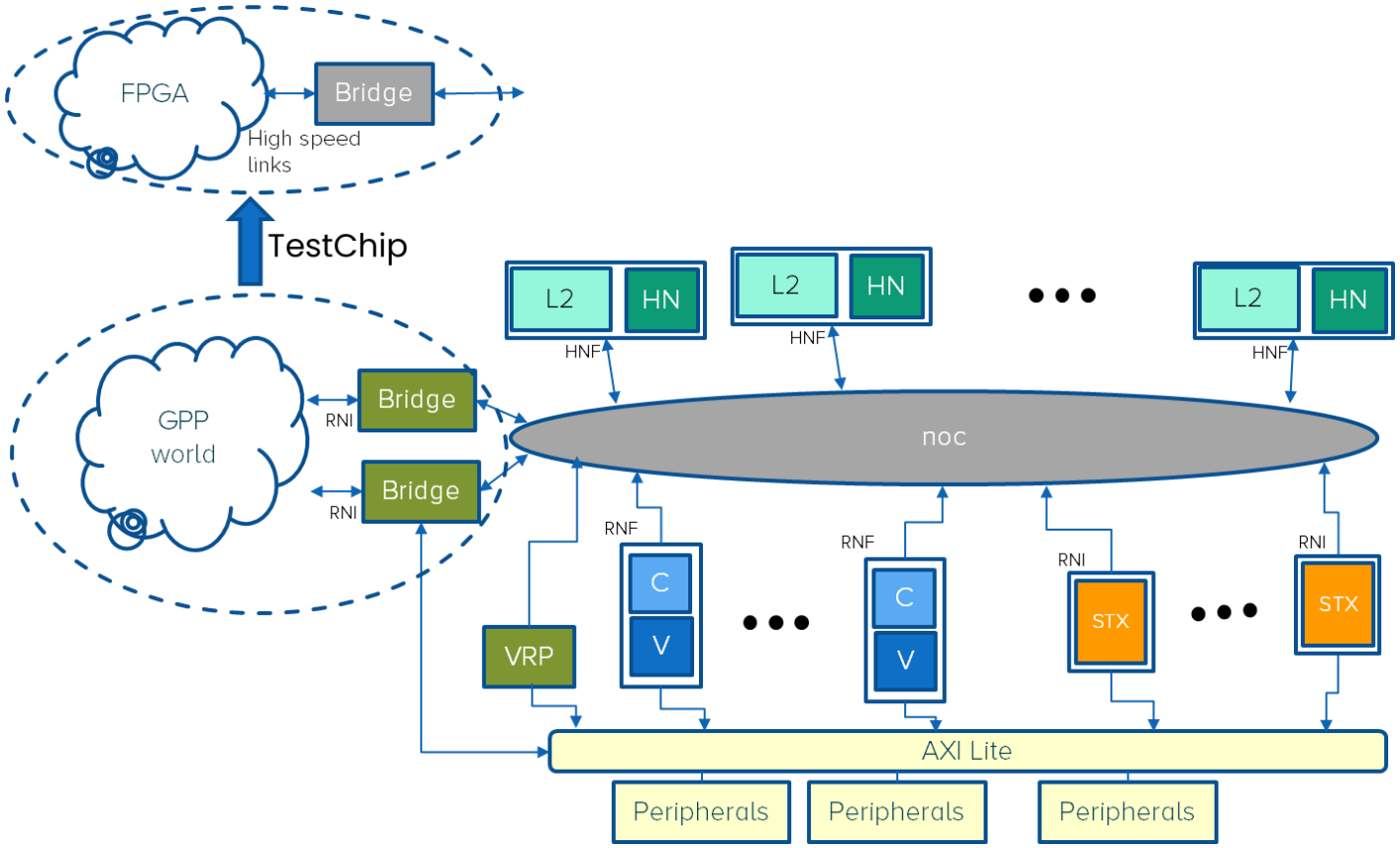
Holistic Co-design





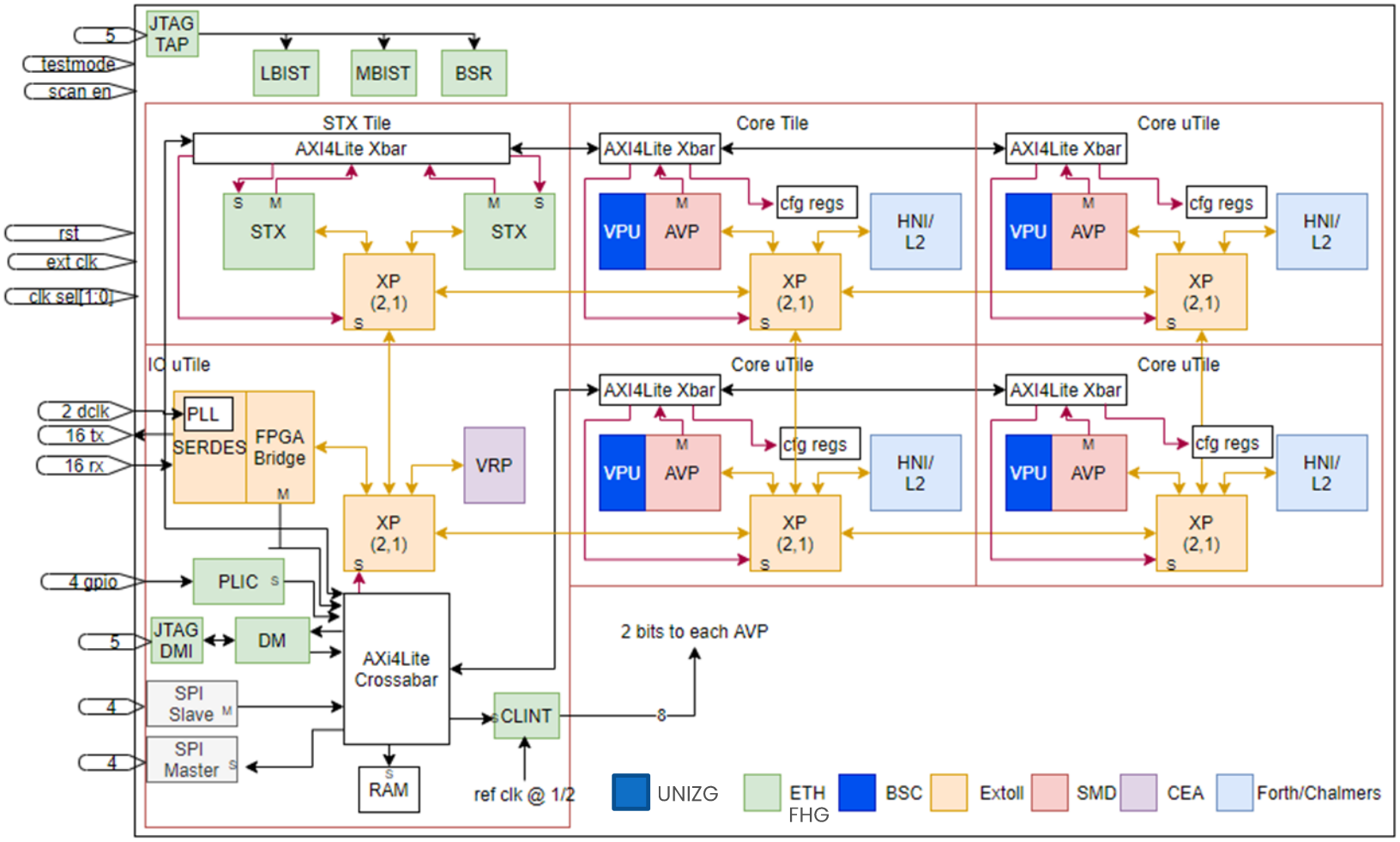
EPAC

- RV64GCV (→ 8x)
  - 2 way in order core
  - Decoupled VPU
    - 8 lanes
    - Long vectors (256 DP elements)
  - L1 - MESI coherency
- CHI interface NoC
  - 1 line / cycle (high B/F)
- L$2: 256KB/module
  - Allocation control mechanisms
- Linux

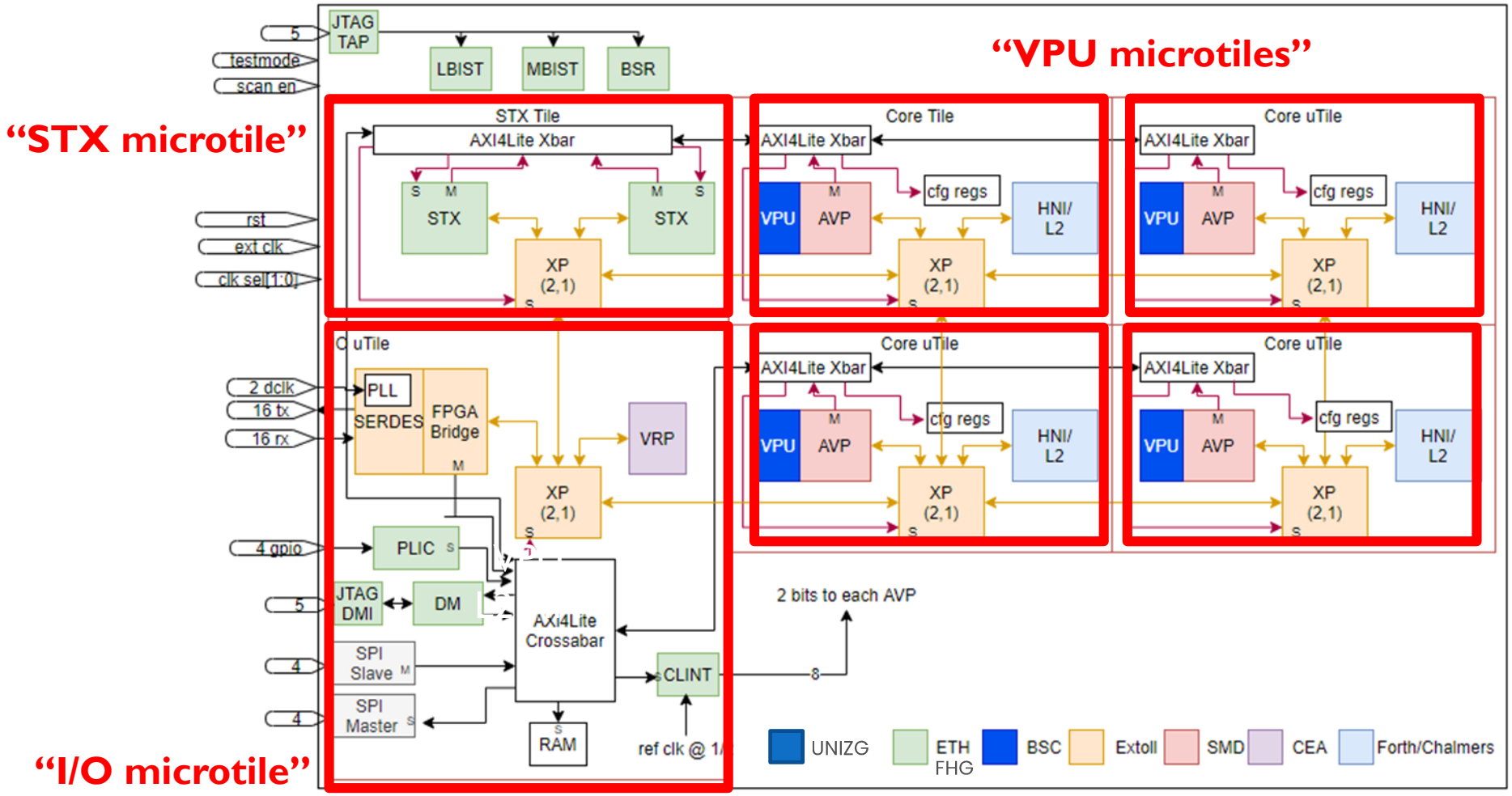


- STX: DL and stencil specific accelerators
- VRP: variable precision processors

EPAC TEST CHIP BLOCK DIAGRAM



EPAC TEST CHIP BLOCK DIAGRAM

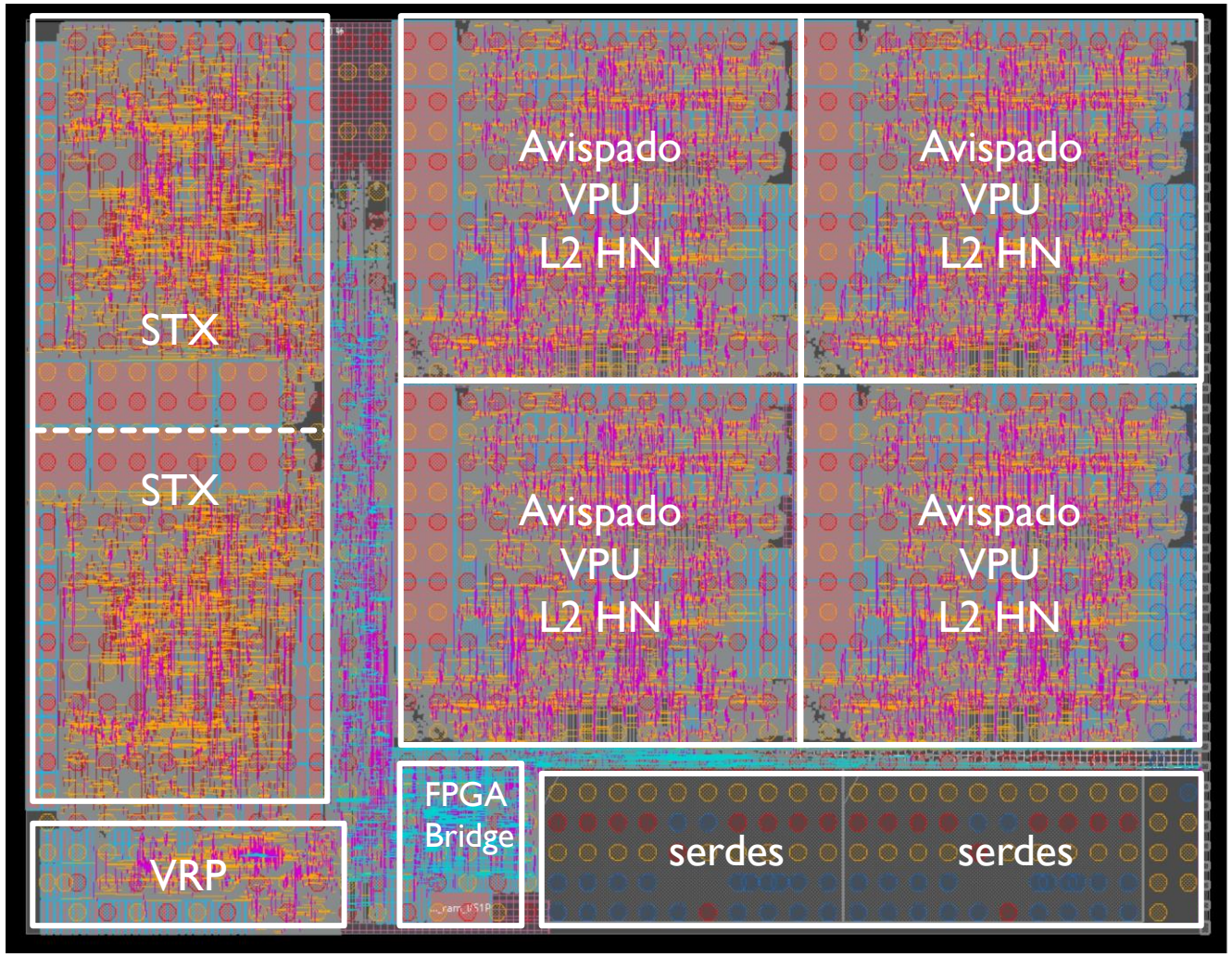


PHYSICAL DESIGN

- Final Top level chip floorplan
- Total area:

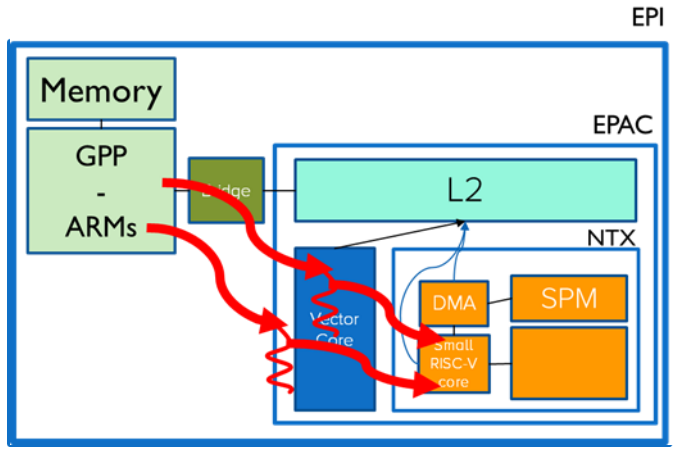
  5943 X 4593 um^2

  (27.297 mm^2)



PROGRAMMING MODEL

- MPI + OpenMP
  - Offloading
  - Tasks
  - SIMD
  - Specific language extensions



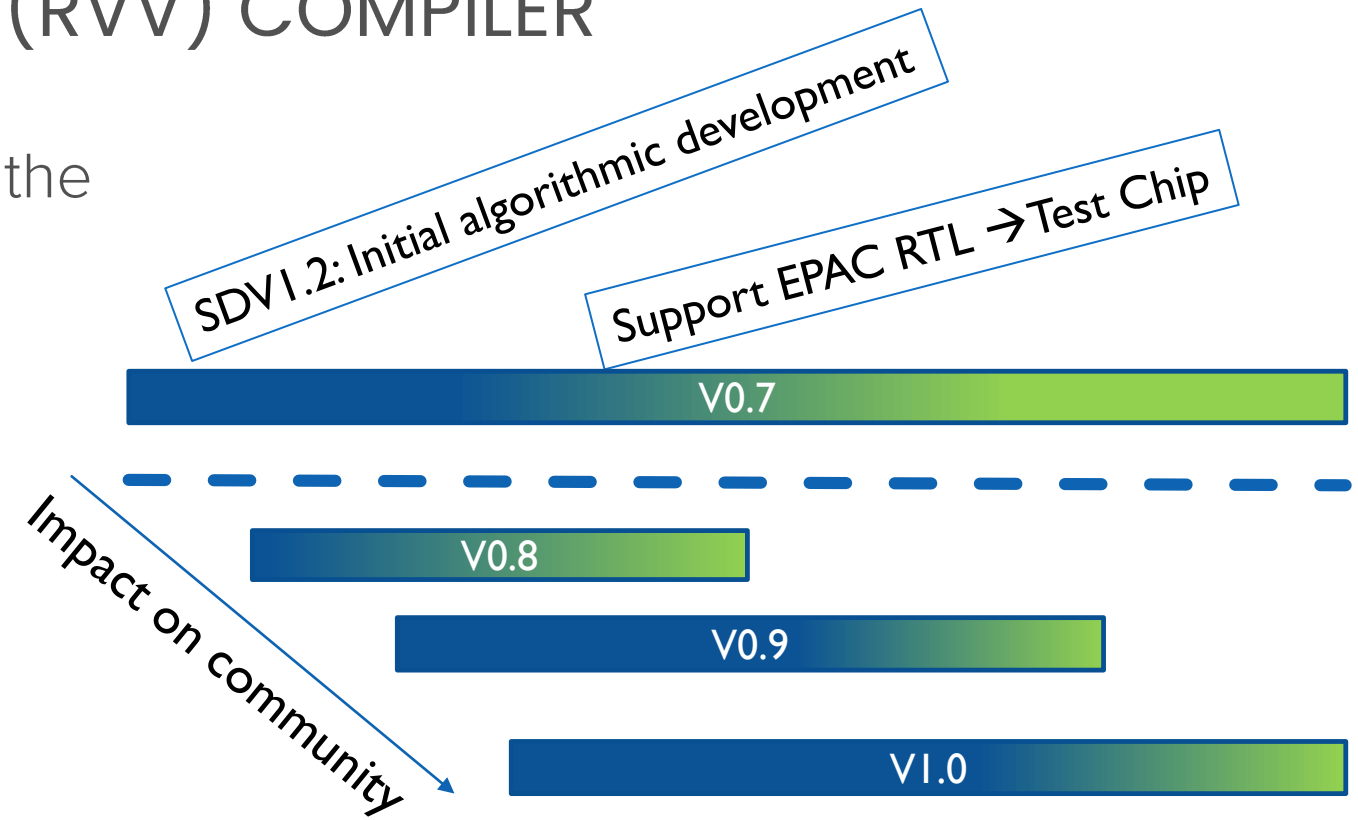

```
void axpy_omp_nest_3  (double a, double *dx, double *dy, int n) {
   int i, chunk;


   #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_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];
   }
}
```

```
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;
    }
}
```

RISC-V VECTOR EXTENSION (RVV) COMPILER

- LLVM support for the evolution of the RISC-V Vector (RVV) Extension
  - Intrinsics
  - Backend
  - Vectorization
- Cooperation
  - RISC-V Vector → SiFive
  - Other vector and VLA arch's → ARM, NEC

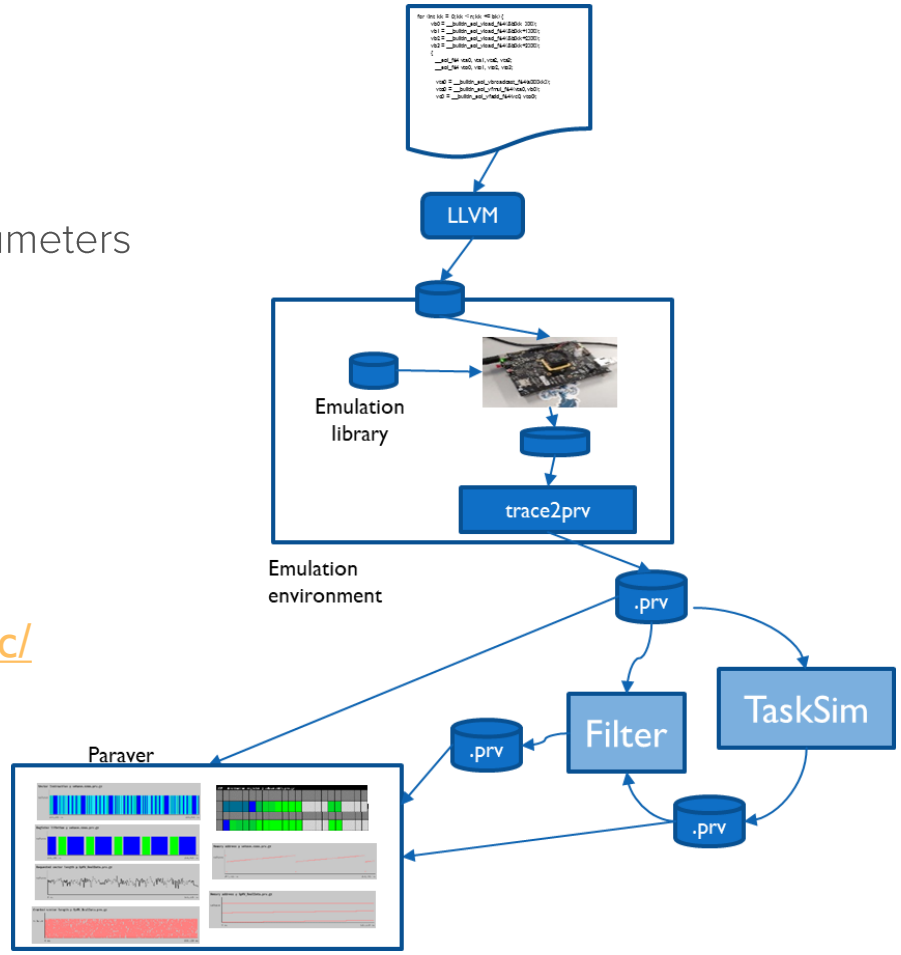


co-design
- Algorithm
- ISA
- Microarchitecture

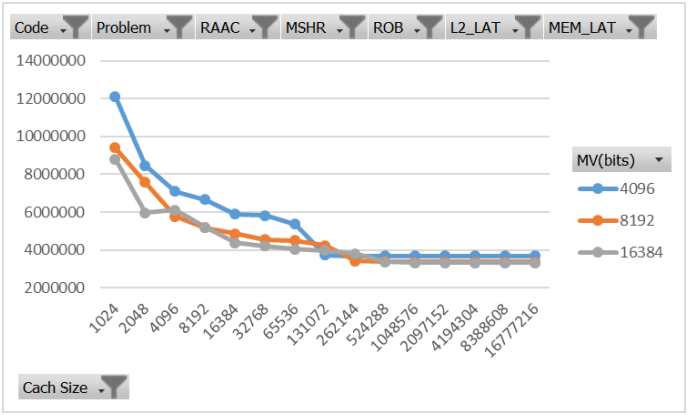
SDV 3.0 ++

SDV1.2: VECTOR SOFTWARE EMULATION

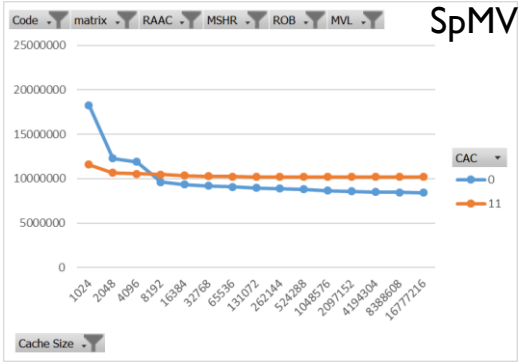
- Functional
- Timing model
  - Microarchitectural parameters
  - OoO, Decoupled
  - Locality management



Swaptions



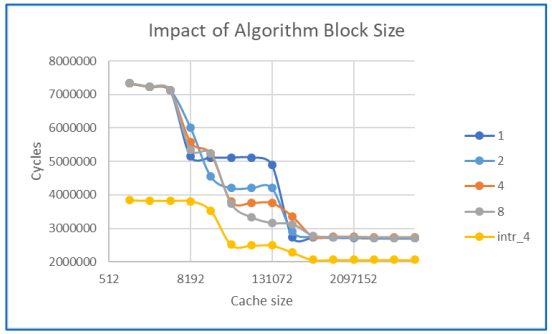
SpMV



https://ssh.hca.bsc.es/epi/ftp/
https://ssh.hca.bsc.es/epi/ftp/doc/

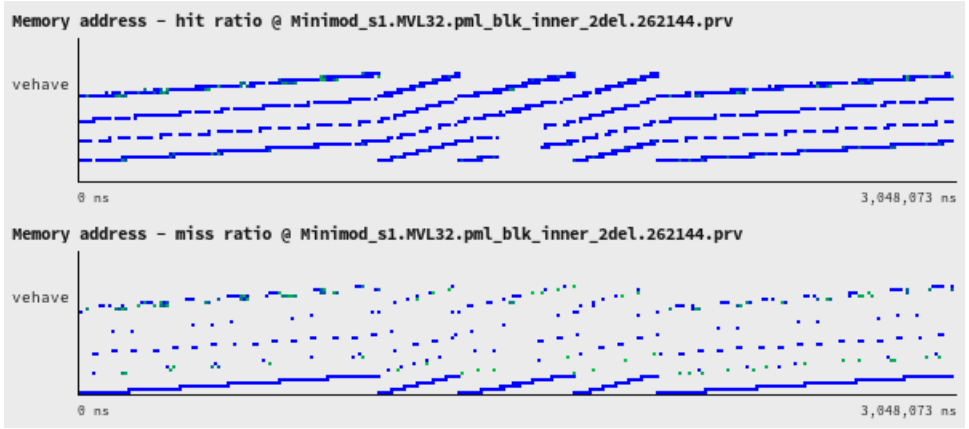
SDV1.2: VECTOR SOFTWARE EMULATION

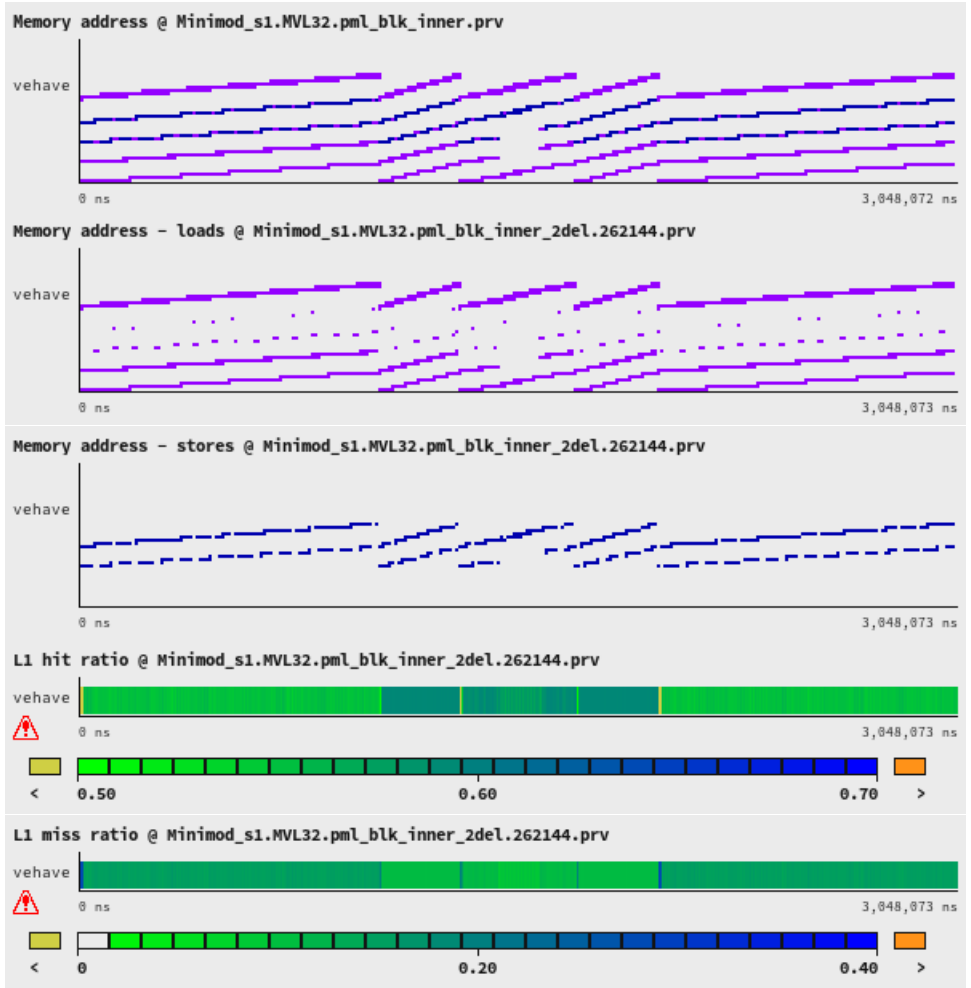
- Detailed analysis & insight



TOTAL

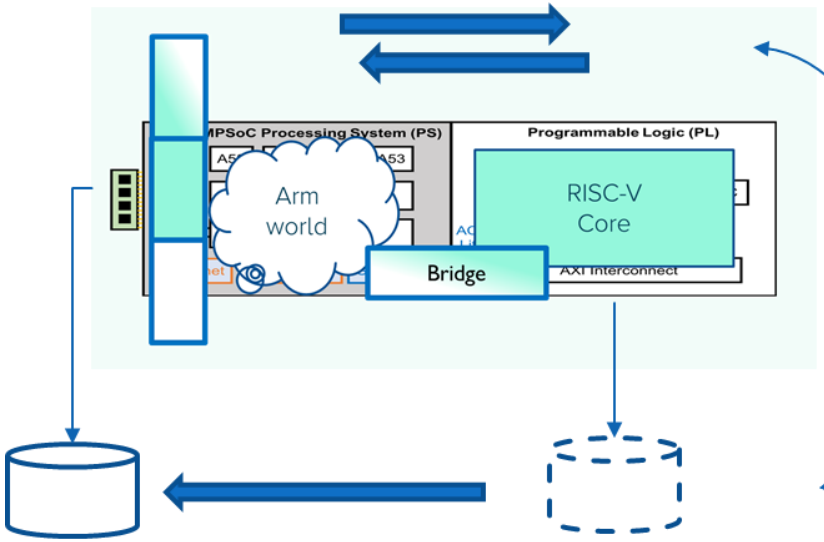
256K cache





SDV2.1: HETEROGENEOUS PLATFORM (ARM + RISC-V)

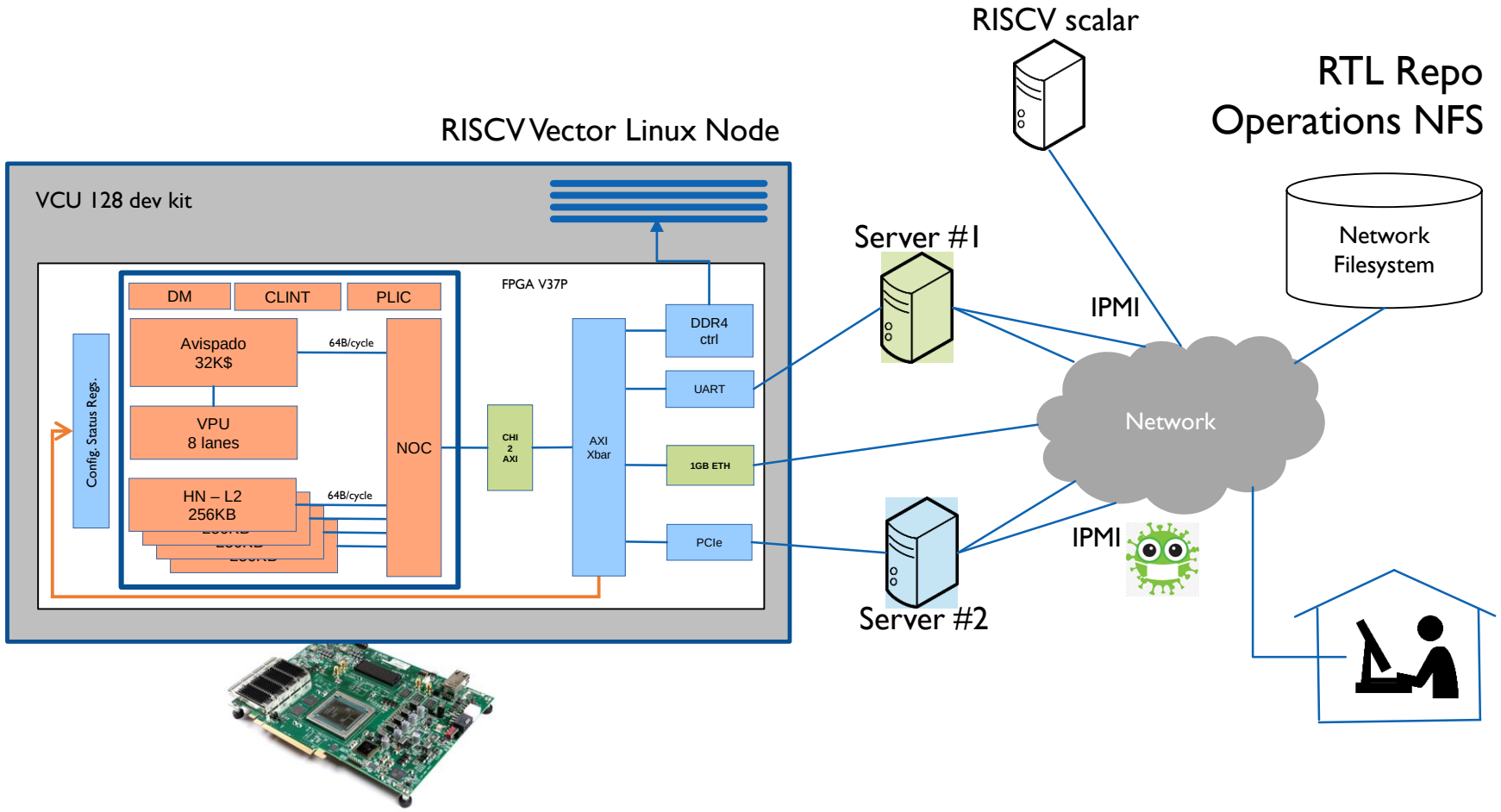
- Linux Boot on Arm and RISC-V
  - kernel version updates
  - Tracking mainline, and contributing to ongoing patch testing and review (eg. SV48, huge pages)
- OpenMP offloading
  - Asynchronous calls (via service thread pool)
  - Thread teams
  - Additional functionality and better stability (eg. C++, library calls, more microbenchmarks).
- Reverse offloading
  - access to host-side I/O devices
  - Works for Linux process on RISC-V side
  - WIP for offloaded tasks




```
#on RISC-V side
$mkfs.ext4 -b 4096 /dev/vda
$mount -t ext4 /dev/vda /mnt/scratchfs
```

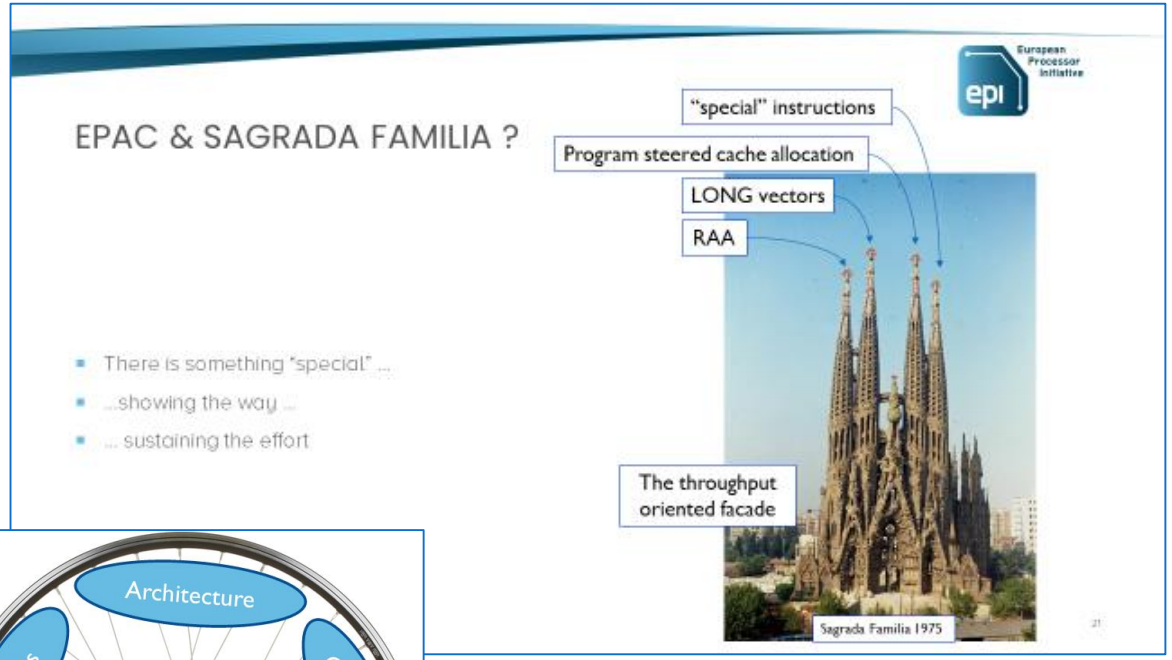
```
void main (…)
{ …
   gemTarget(A, B, C, size);
… }

void gemTarget(double *A, double *B, double *C, long S) {
  #pragma omp target map(to:A[0:S*S], B[0:S*S], S) \
                     map(from: C[0:S*S])
  {    for(int i = 0; i < size; i++)
         for(int j = 0; j < size; j++)
           for(int k = 0; k < size; k++)
             C[i*size + j] += A[i*size + k]*B[k*size + j];
       fp = fopen ("/mnt/scratchfs/output_matrix.txt", "w+");
       for (int i=0; i<size*size; i++) fprintf(fp, "%lf ", C[i]);
  }
}
```

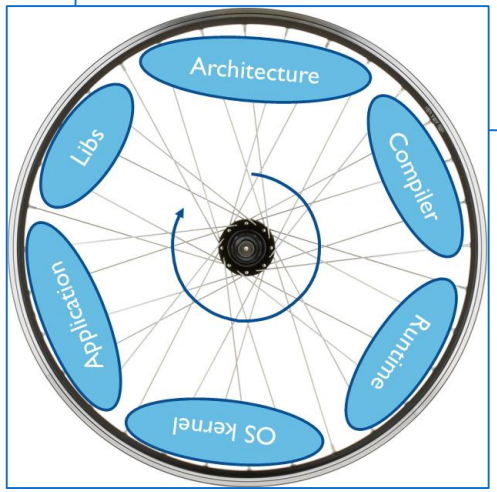
SDV3.2


RISCV scalar
RTL Repo
Operations NFS
RISCV Vector Linux Node
VCU 128 dev kit
FPGA V37P
DM
CLINT
PLIC
Avispado
32K$
64B/cycle
VPU
8 lanes
HN – L2
256KB
64B/cycle
Config. Status Regs.
NOC
CHI
2
AXI
AXI
Xbar
DDR4
ctrl
UART
1GB ETH
PCIe
Server #1
IPMI
Network
Filesystem
Network
IPMI
Server #2

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
- High bandwidth per control flow
- Push behaviour exploitation to low levels
- Co-ordination between levels
- Make it all look as classical sequential programming to ensure productivity





THE RISC-V VECTOR PROCESSOR IN EPI

JESUS LABARTA (@BSC.ES)