



**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación



Software Development Vehicles (SDVs)

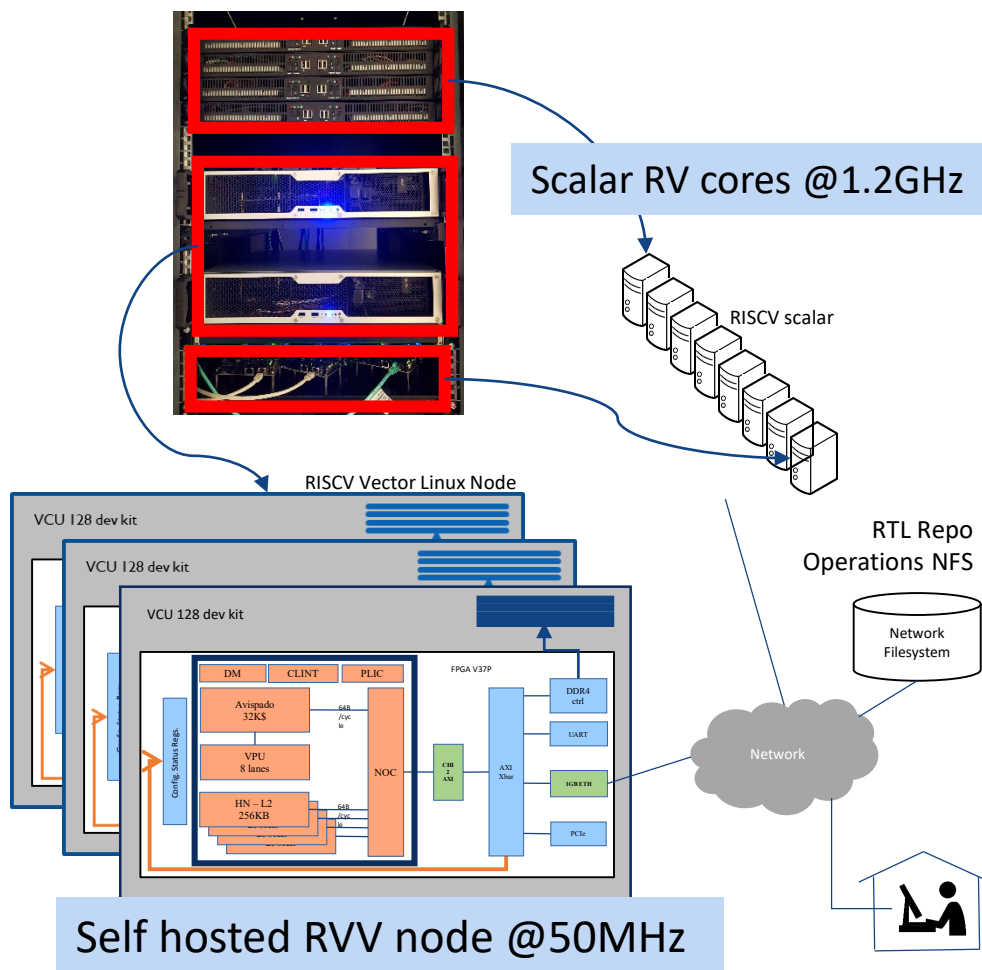
Prof. Jesús Labarta
BSC & UPC

ACM Summer School
Barcelona, August 30th 2022

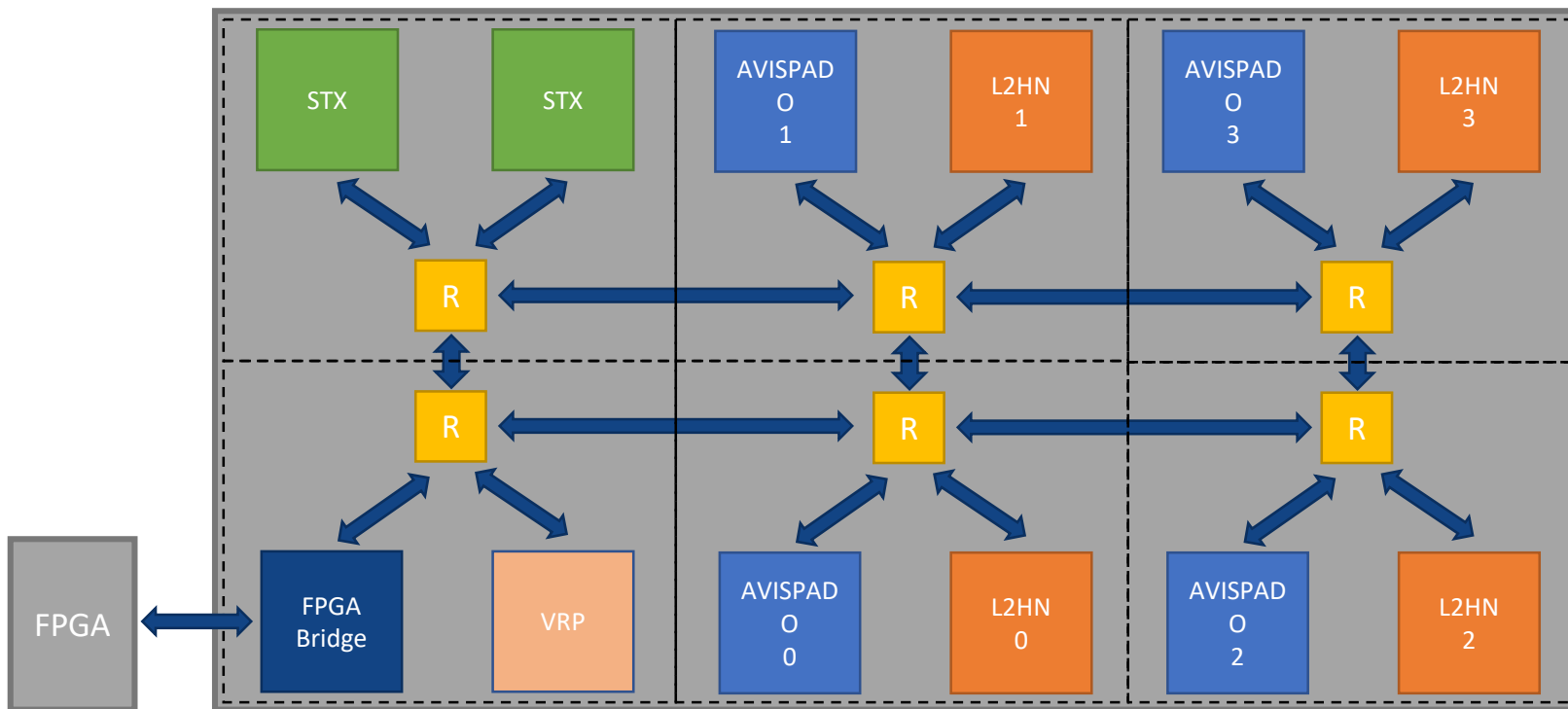
RVV @ FPGA & ecosystem

- 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

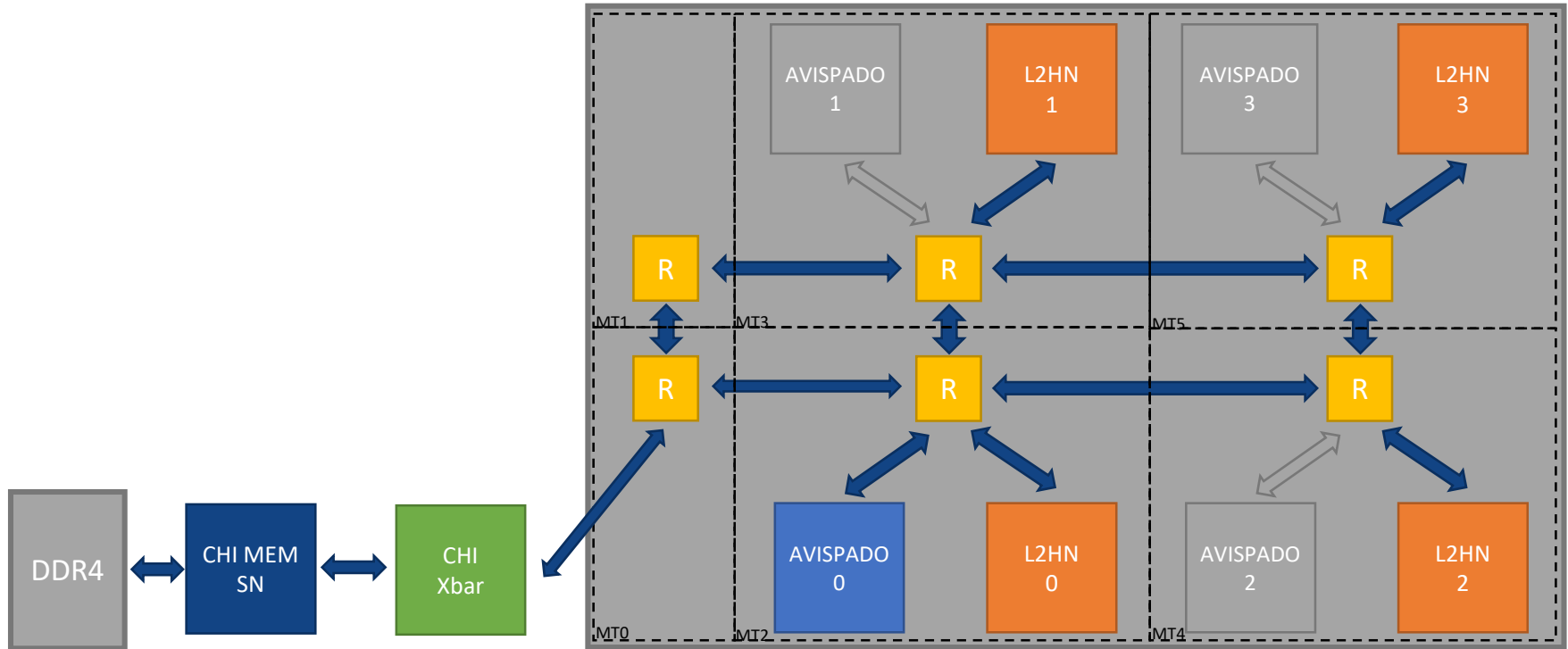


EPAC Test Chip



Courtesy V. Papaefstathiou

Single Core SDV



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
    }
}
```

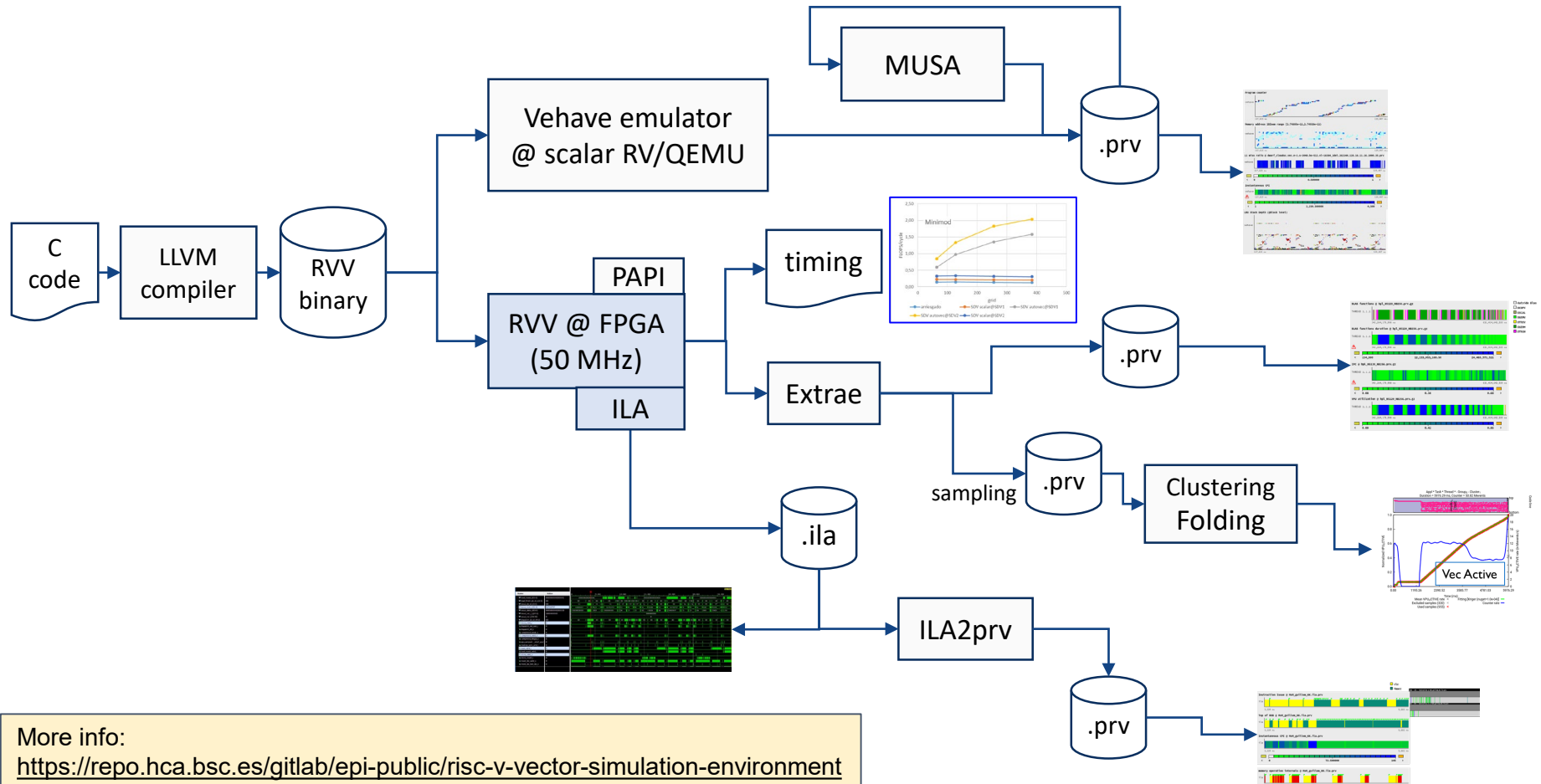
```
void target_inner_3d (...) {
    for (i) {
        for (j) {
            #pragma clang loop vectorize(enable)
            for (k) {
                lap = LAP;
                v[IDX3_1(i,j,k)] = 2.f*u[IDX3_1(i,j,k)]
                    -v[IDX3_1(i,j,k)]+vp[IDX3(i,j,k)]*lap;
            }
        }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

```
float complex A[n][n];
float complex temp1, temp2;
...
for (int j = 0; j < n; j++) {
    if (x[j] != ZERO || y[j] != ZERO) {
        temp1 = 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] * temp1 + y[i] * temp2;
        A[j][j] = creal(A[j][j]) + creal(x[j] * temp1 + y[j] * temp2);
    } else A[j][j] = creal(A[j][j]);
}
```

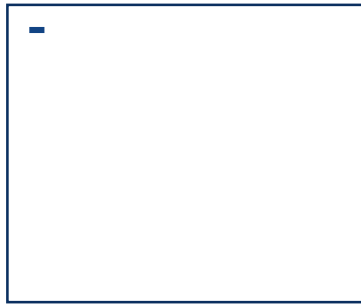
SDV flows

App development & Data acquisition

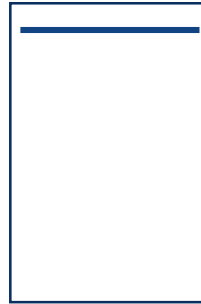
Analysis



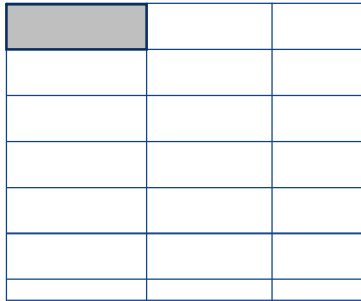
Matrix multiply



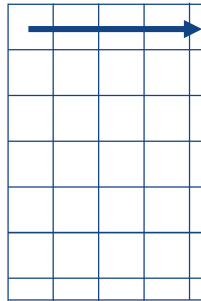
=



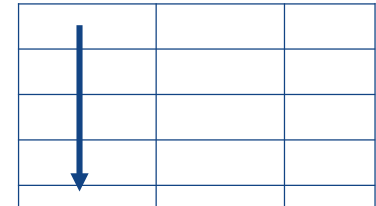
x



=



x



Matrix multiply (Scalar)

```
static void matmul_reference ( int M, int K, int N,  
                             double (*c)[N], double (*a)[K], double (*b)[N]) {  
    for (int i = 0; i < M; i++) {  
        for (int j = 0; j < N; j++) {  
            for (int k = 0; k < K; k++) {  
                c[i][j] += a[i][k] * b[k][j];  
            }  
        }  
    }  
}
```

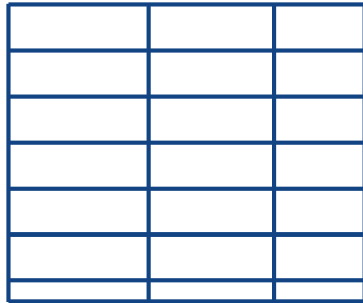
```
static void matmul_blocked(int M, int K, int N, int BS,  
                           double (*c)[N], double (*a)[K], double (*b)[N]) {  
    for (int II = 0; II < M; II+=BS) {  
        for (int JJ = 0; JJ < N; JJ+=BS) {  
            for (int KK = 0; KK < K; KK+=BS) {  
  
                int isup = II+BS > M? M:II+BS;  
                int jsup = JJ+BS > N? N:JJ+BS;  
                int ksup = KK+BS > K? K:KK+BS;  
                for (int i = II; i < isup; i++) {  
                    for (int j = JJ; j < jsup; j++) {  
                        for (int k = KK; k < ksup; k++) {  
                            c[i][j] += a[i][k] * b[k][j];  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```


Matrix multiply (vectorized)

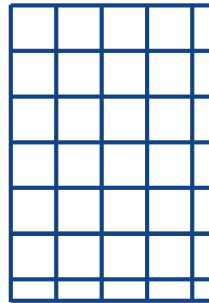
```
static void matmul_autovec_red(int M, int K, int N, double (* restrict c)[N],
                               double (* restrict a)[K], double (* restrict b)[N]) {
    for (int i = 0; i < M; i++) {
        for (int j = 0; j < N; j++) {
            #pragma clang loop vectorize(enable)
            for (int k = 0; k < K; k++) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}
```

```
static void matmul_autovec_rows(int M, int K, int N, double (*c)[N],
                                double (*a)[K], double (*b)[N]) {
    for (int i = 0; i < M; i++) {
        for (int k = 0; k < K; k++) {
            #pragma clang loop vectorize(enable)
            for (int j = 0; j < N; j++) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}
```

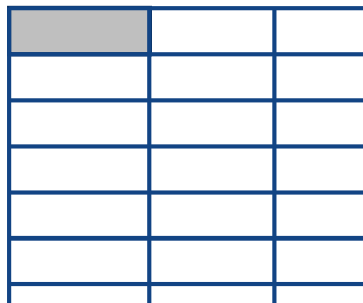
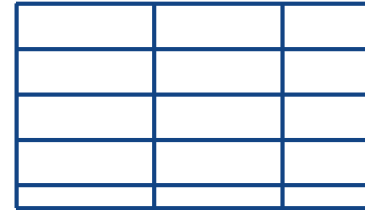
Matrix multiply



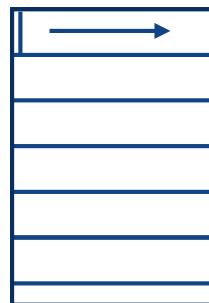
=



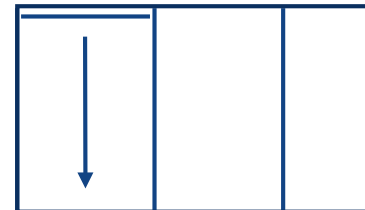
x



=



x



← v | →

Matrix multiply (intrinsics)

```

void matmul_intrinsics(int M, int K, int N, int bi, int bj, int bk,
    double (* restrict c)[N], double (* restrict a)[K], double (* restrict b)[N]) {
    assert(bi == 24); assert(bk == 1); ...
    int ii;
    for (int jj = 0; jj < N; ) {    // system selected column block size (C & B)
        unsigned long gvl = __builtin_epi_vsetv1(N-jj, __epi_e64, __epi_m1);
        for (int ii = 0; ii < M-(bi-1); ii += bi) {
            __epi_1xf64 vc0, vc1, ..., vc23;
            __epi_1xf64 vb0, vb1;
            vc0 = __builtin_epi_vload_1xf64(&c[ii][jj], gvl);
            vc1 = __builtin_epi_vload_1xf64(&c[ii+1][jj], gvl);
            ...
            vc23 = __builtin_epi_vload_1xf64(&c[ii+23][jj], gvl);

            for (int kk = 0; kk < K; kk += bk) {

                vb0 = __builtin_epi_vload_nt_1xf64(&b[kk][jj], gvl);
                FMA( vc0, vb0, a[ii][kk], gvl );
                FMA( vc1, vb0, a[ii+1][kk], gvl );
                ...
                FMA( vc23, vb0, a[ii+23][kk], gvl );
            }
            __builtin_epi_vstore_1xf64(&c[ii][jj], vc0, gvl);
            __builtin_epi_vstore_1xf64(&c[ii+1][jj], vc1, gvl);
            ...
            __builtin_epi_vstore_1xf64(&c[ii+23][jj], vc23, gvl);
        }
        jj += gvl;
    }
}

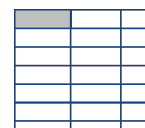
```



Matrix multiply (intrinsics)

Double buffer to
overlap load latency

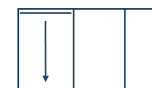
```
void matmul_intrinsics(int M, int K, int N, int bi, int bj, int bk,
    double (* restrict c)[N], double (* restrict a)[K], double (* res
    assert(bi == 24); assert(bk == 1); ...
    int ii;
    for (int jj = 0; jj < N; ) {    // system selected column block size (C & B)
        unsigned long gvl = __builtin_epi_vsetv1(N-jj, __epi_e64, __epi_m1);
        for (ii = 0; ii < M-(bi-1); ii += bi) {
            __epi_1xf64 vc0, vc1, ..., vc23;
            __epi_1xf64 vb0, vb1;
            vc0 = __builtin_epi_vload_1xf64(&c[ii][jj], gvl);
            vc1 = __builtin_epi_vload_1xf64(&c[ii+1][jj], gvl);
            ...
            vc23 = __builtin_epi_vload_1xf64(&c[ii+23][jj], gvl);
            vb1 = __builtin_epi_vload_nt_1xf64(&b[0][jj], gvl);
            for (int kk = 0; kk < K; kk += bk) {
                vb0 = vb1;
                if (kk < K-1) vb1 = __builtin_epi_vload_nt_1xf64(&b[kk+1][jj], gvl);
                FMA( vc0, vb0, a[ii][kk], gvl );
                FMA( vc1, vb0, a[ii+1][kk], gvl );
                ...
                FMA( vc23, vb0, a[ii+23][kk], gvl );
            }
            __builtin_epi_vstore_1xf64(&c[ii][jj], vc0, gvl);
            __builtin_epi_vstore_1xf64(&c[ii+1][jj], vc1, gvl);
            ...
            __builtin_epi_vstore_1xf64(&c[ii+23][jj], vc23, gvl);
        }
        jj += gvl;
    }
}
```



=



x





**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación



**EXCELENCIA
SEVERO
OCHOA**

SDV@FPGA

Time executions

SDV Allocation and access

```
$ salloc -p fpga-sdv -N 1 -t 01:00:00
salloc: Granted job allocation 133526
salloc: Waiting for resource configuration
salloc: Nodes pickle-2 are ready for job
```

Can take a few minutes
allocate pickle node
load bitstream
boot linux

```
$ ssh fpga-sdv-2
Welcome to Ubuntu 21.04 (GNU/Linux 5.7.0-yarvt-sdv3-minimal-network+ riscv64)
```

```
$ ssh pickle-2
Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.4.0-107-generic x86_64)
```

```
$ salloc -p arriesgado-ubuntu -n1 -t01:00:00
salloc: Granted job allocation 133528
salloc: Waiting for resource configuration
salloc: Nodes arriesgado-3 are ready for job
```

```
$ ssh arriesgado-3
Welcome to Ubuntu 21.04 (GNU/Linux 5.11.10-meep riscv64)
```

fpga-sdv-2
(RV – vector)

Pickle-2
(x86)

Arriesgado-3
(RV – scalar)

Timing apps

```
#include <sys/time.h>
#include <stddef.h>
#include "elapsed_time.h"
```

```
long long get_time() {
    struct timeval tv;
    gettimeofday(&tv, NULL);
    return (tv.tv_sec * 1000000) + tv.tv_usec;
}
```

// Returns the number of seconds elapsed between the two specified times

```
float elapsed_time(long long start_time, long long end_time) {
    return (float) (end_time - start_time) / (1000 * 1000);
}
```

```
#include <time.h>
#include <stdint.h>
#include <sys/time.h>
#include "elapsed_time.h"
...
start_time = get_time();
matmul_intrinsics(m, k, n, bi, bj, bk, c, a, b);
end_time = get_time();
time_intr = elapsed_time(start_time, end_time);
....
```

Matrix Multiply 396 x 396 x 396, with MAXVL = 240

Current VLMAX with SEW 64 = 256 ... Setting VLMAX to 240

reference scalar -	time: 16.403316, FLOPS/cycle: 0.15143068
ikj scalar -	time: 23.824690, FLOPS/cycle: 0.10426014
OpenBLAS scalar -	time: 3.602275, FLOPS/cycle: 0.68955466
autovectorized columns -	time: 1.571810, FLOPS/cycle: 1.58032169
vectorized intrinsics -	time: 0.173931, FLOPS/cycle: 14.28132652
vectorized intrinsics -	time: 0.175617, FLOPS/cycle: 14.14422019
vectorized intrinsics -	time: 0.176709, FLOPS/cycle: 14.05681370
vectorized intrinsics -	time: 0.174191, FLOPS/cycle: 14.26001038
vectorized intrinsics -	time: 0.177054, FLOPS/cycle: 14.02942266
vectorized intrinsics -	time: 0.173232, FLOPS/cycle: 14.33895228

...



**Barcelona
Supercomputing
Center**

Centro Nacional de Supercomputación



**EXCELENCIA
SEVERO
OCHOA**

SDV@FPGA

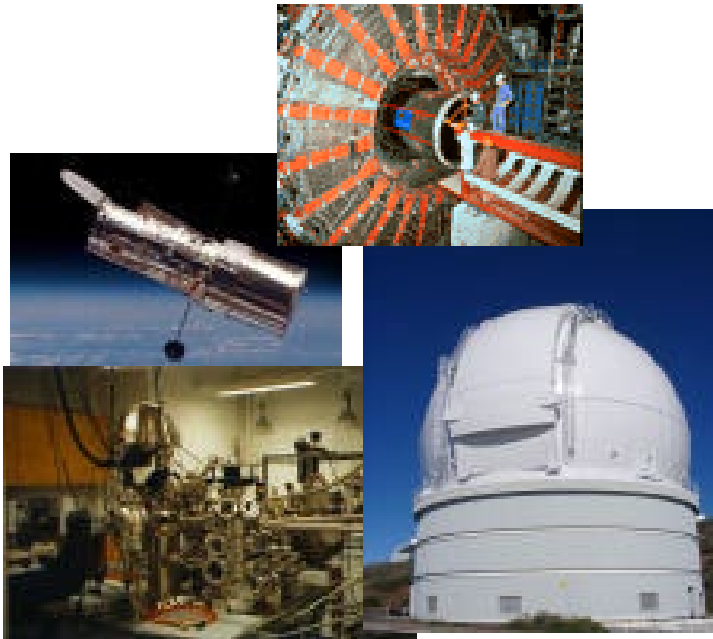
Standard Extrae instrumentation

Extræ @ SDV

- The standard BSC tools instrumentation package (Extræ) is installed on the SDV
 - Instrumentation + sampling
 - Hardware counters
- Generates paraver traces that can be analyzed with the standard Paraver and analytics tools.

Tools in ...

- **Science**



- Advances linked to capability of observation and measurement

- **Computer Science**

- `printf()`
- timers

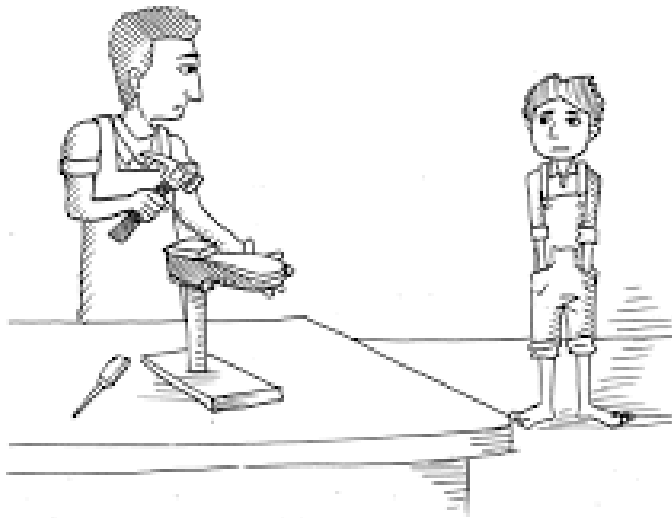
- Performance analysis:
 - A lot of speculation

- We see $\int_a^b f(t)dt$

- we talk about $f(t)$

Shoemaker's son ...

...always goes barefoot



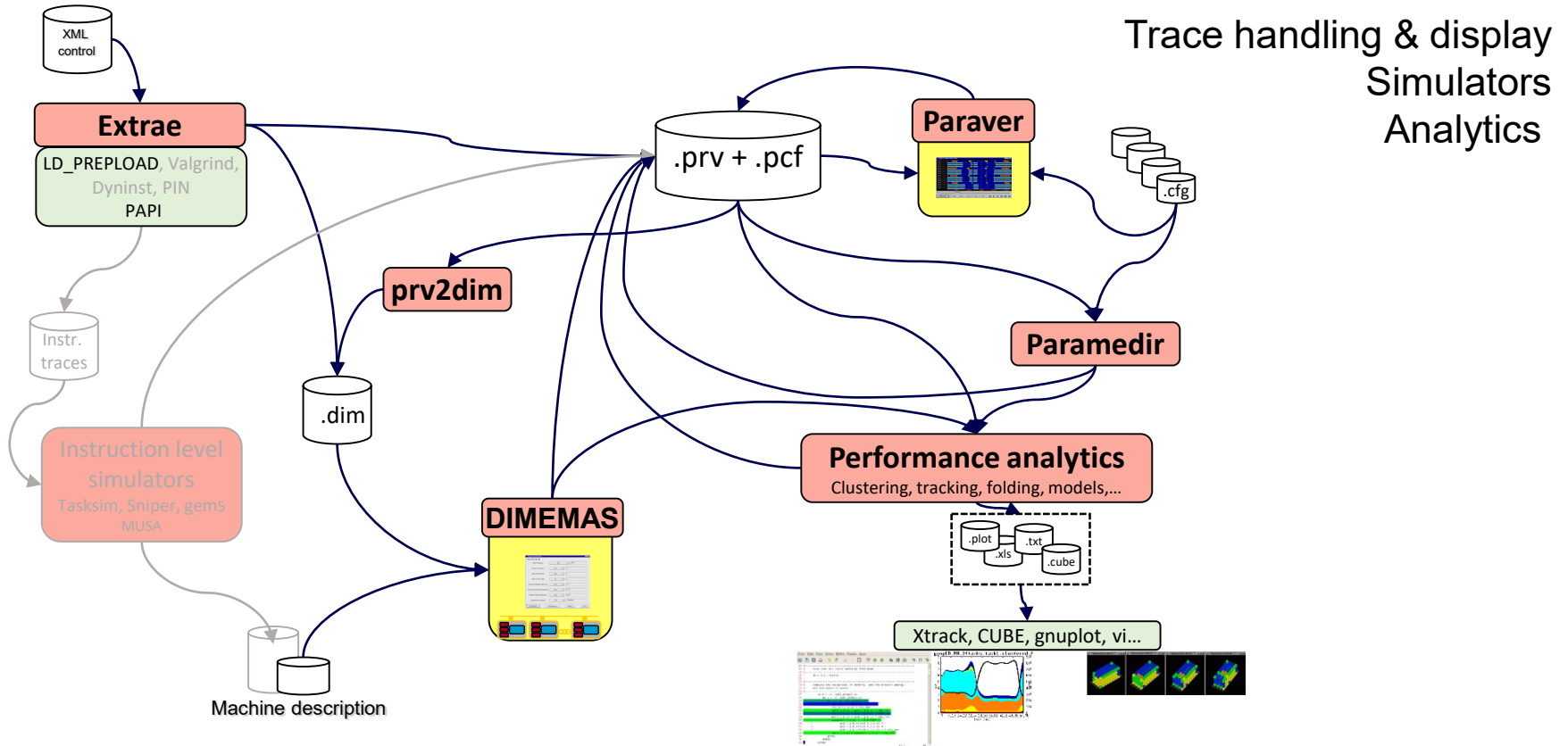
En casa del herrero ...



... cuchara de palo

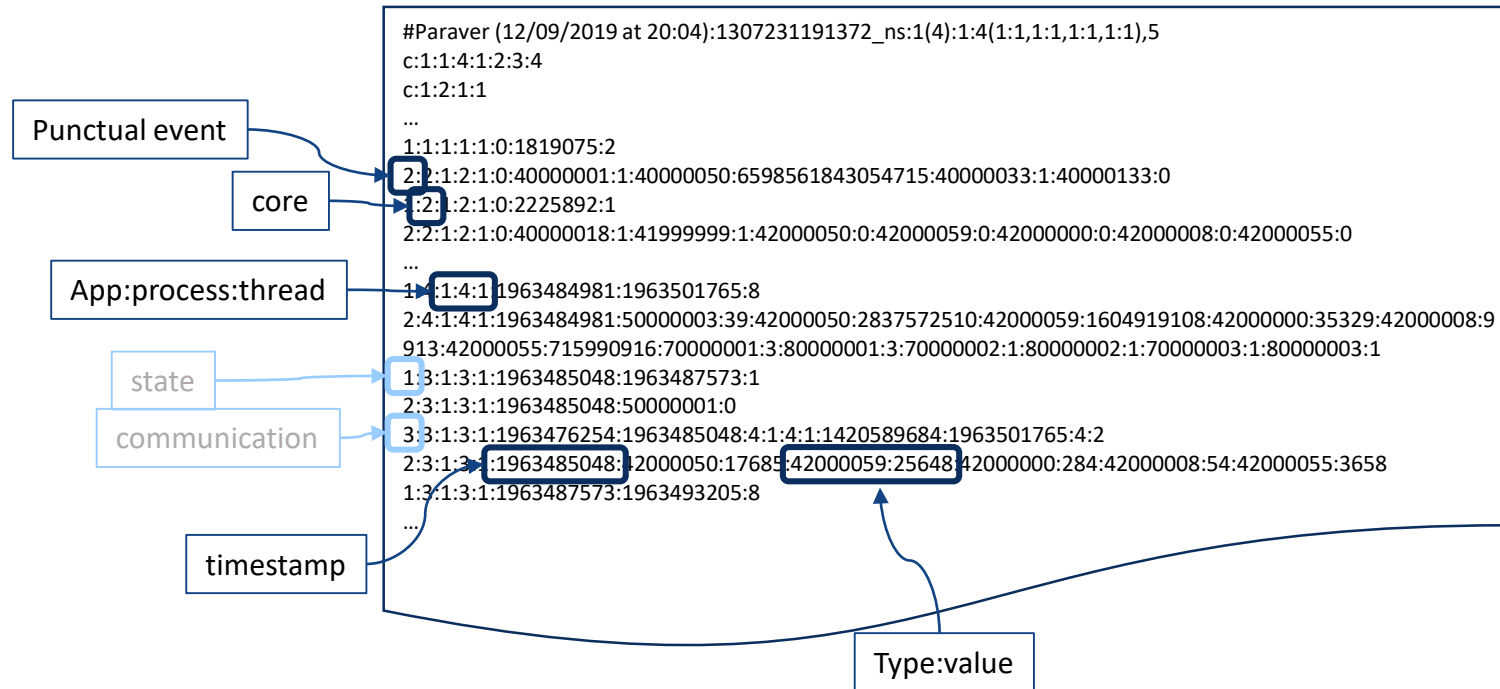


BSC – tools framework



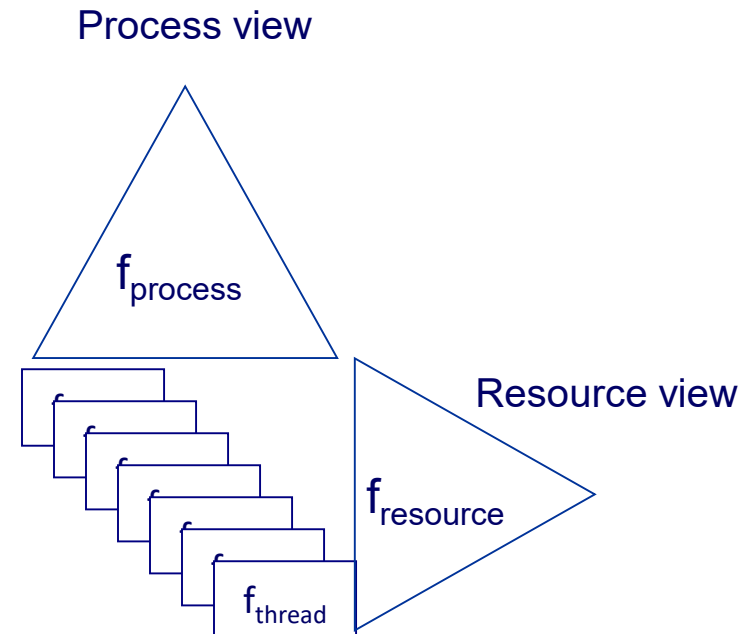
Paraver trace format

- .prv



Paraver data model

- “Time”
- Perspective:
 - “Process model”
 - “Thread”, “task”, “application”, “workload”
 - “Resource model”
 - “CPU”, “node”, “system”

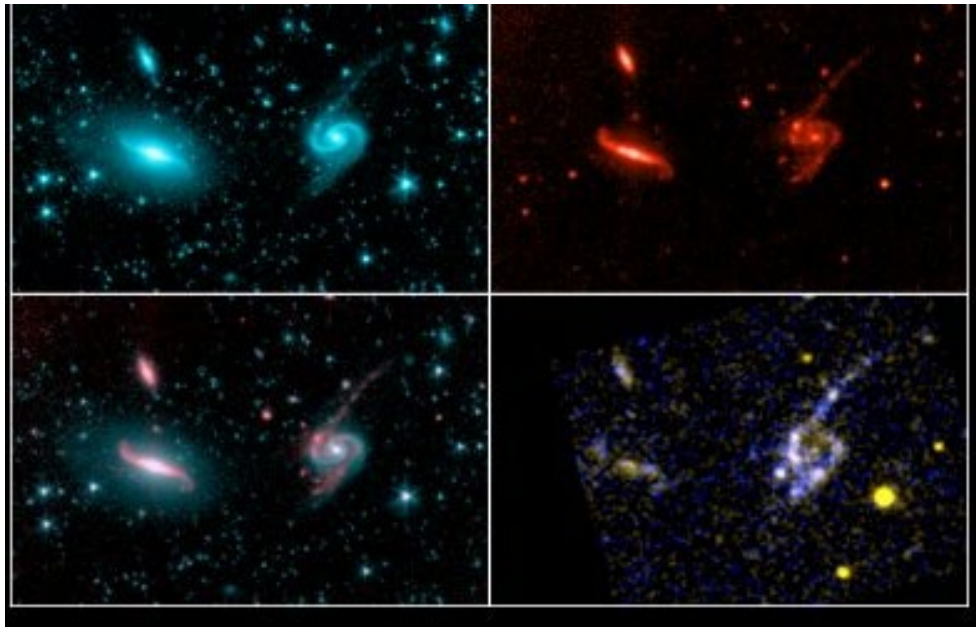


What is Paraver

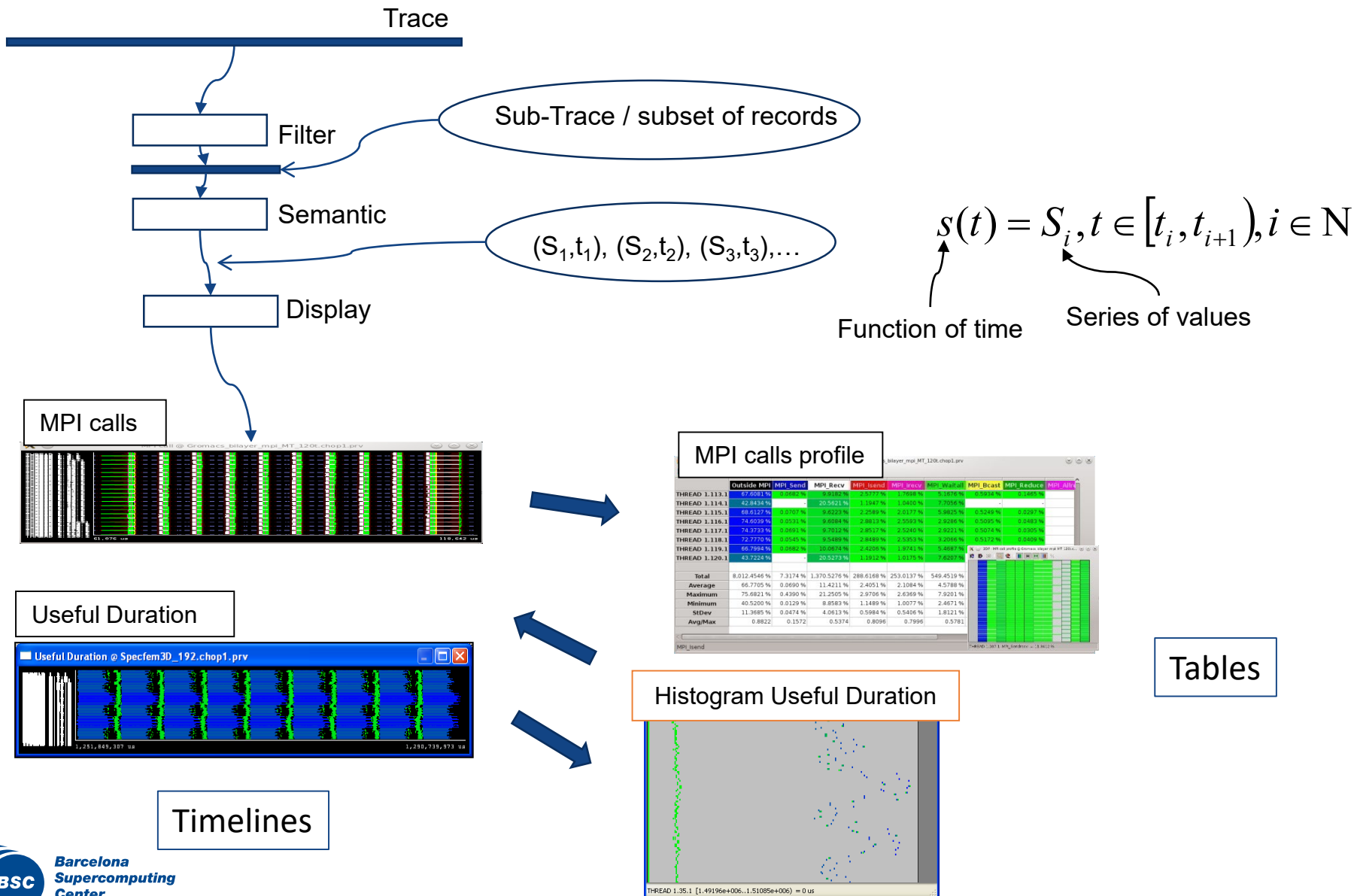
- A browser ...
- ...to manipulate (visualize, filter, cut, combine, ...)
- ... sequences of time-stamped events (traces) ...
- ... with a multispectral philosophy ...
- ... and a mathematical foundation ...
- ... that happens to be mainly used for **performance analysis**

Multispectral imaging

- Different looks at one reality
 - Different spectral bands (light sources and filters)
- Highlight different aspects
 - Can combine into false colored but highly informative images



Paraver mathematical foundation



Manual instrumentation

```
#include "extrae_user_events.h"
...
extrae_type_t type = 1000;
int nvalues = 11;
extrae_value_t values[11] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
char * description_values[11] = {"End", "Malloc", "Init data", "Ref. MxM",
    "Blocked MxM", "auto vec. red. MxM", "auto vec. columns",
    "vectorized intrinsics", "Guillem", "OpenBLAS", "deJens" };
...
Extrae_init();
Extrae_define_event_type (&type, "Code region",
    &nvalues, values, description_values);
...

Extrae_eventandcounters (1000, 1);

...

Extrae_eventandcounters (2000, loop_control_variable);

Extrae_fini();
```

`$EXTRAE_HOME/include/extrae_user_events.h`

```
<counters enabled="yes">
  <cpu enabled="yes" starting-set-distribution="1">
    <set enabled="yes" domain="all" changeat-time="50s">
      PAPI_TOT_INS, PAPI_TOT_CYC, CSR_VL, PAPI_L1_DCM, PAPI_TLB_DM
    </set>
    <set enabled="yes" domain="all" changeat-time="50s">
      PAPI_TOT_INS, PAPI_TOT_CYC, CSR_VL, VPU_COMPLETED_INST, VPU_ACTIVE
    </set>
    <set enabled="yes" domain="all" changeat-time="50s">
      PAPI_TOT_INS, PAPI_TOT_CYC, CSR_VL, VPU_MEM_INST, VPU_ARITH_INST
    </set>
    <set enabled="yes" domain="all" changeat-time="50s">
      PAPI_TOT_INS, PAPI_TOT_CYC, CSR_VL, VPU_FP_INST, VPU_FP_FMA_INST
    </set>
    <set enabled="yes" domain="all" changeat-time="50s">
      PAPI_TOT_INS, PAPI_TOT_CYC, CSR_VL, VPU_ROB_STALL, VPU_ROB_EMPTY
    </set>
  </cpu>

  <sampling enabled="yes" type="default" period="30m" variability="30m" />
</counters>
```

List of hardware counters:

<https://repo.hca.bsc.es/gitlab/epi-public/risc-v-vector-simulation-environment#read-hardware-counters>

Instrumented run

```
#!/bin/bash

source /apps/riscv/extrae/3.8.3/etc/extrae.sh
export LD_LIBRARY_PATH=/apps/riscv/papi-like/development/lib:$LD_LIBRARY_PATH

export EXTRAE_CONFIG_FILE=extrae.xml

export M_K_N=396
./MxM ${M_K_N} ${M_K_N} ${M_K_N} 18 256 1
```

Generates Paraver trace (.prv, .pcf, .row)

views

routine

Hwc set

IPC

Vector instruction mix(%)

VPU utilization (%)

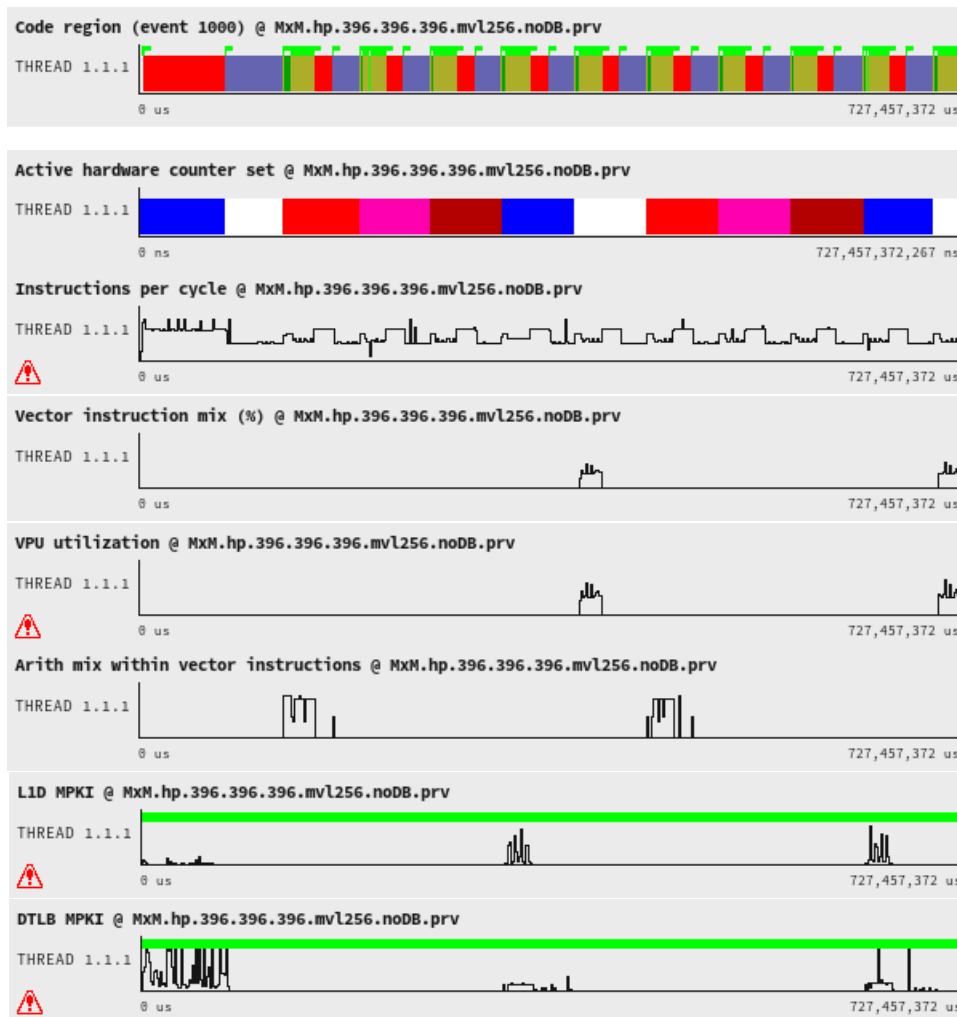
Arith. mix within vector (%)

L1MPKI

DTLP MPKI

Routine profile

- ☐ End
- ☐ Init data
- ☒ Ref. MxM
- ☒ auto vec. columns
- ☒ OpenBLAS
- ☒ ikj
- ☒ Compare Results



Init data	Ref. MxM	auto vec. columns	vectorized intrinsics	OpenBLAS	ikj	Compare
2,849,027 us	22,206,238.829 us	1,588,706.793 us	188,760.277 us	3,891,924.869 us	26,669,797.981 us	122,3

views

Zoom

routine

Hwc set

IPC

Vector instruction mix(%)

VPU utilization (%)

Arith. mix within vector (%)

L1MPKI

DTLP MPKI

Code region (event 1000) @ MxM.hp.396.396.396.mvl256.noDB.prv



■ auto vec. columns
■ vectorized intrinsics
■ OpenBLAS
■ ikj
■ Compare Results

Instructions per cycle @ MxM.hp.396.396.396.mvl256.noDB.prv



Vector instruction mix (%) @ MxM.hp.396.396.396.mvl256.noDB.prv



VPU utilization @ MxM.hp.396.396.396.mvl256.noDB.prv



views

Zoom (zoom)

routine

Hwc set

IPC

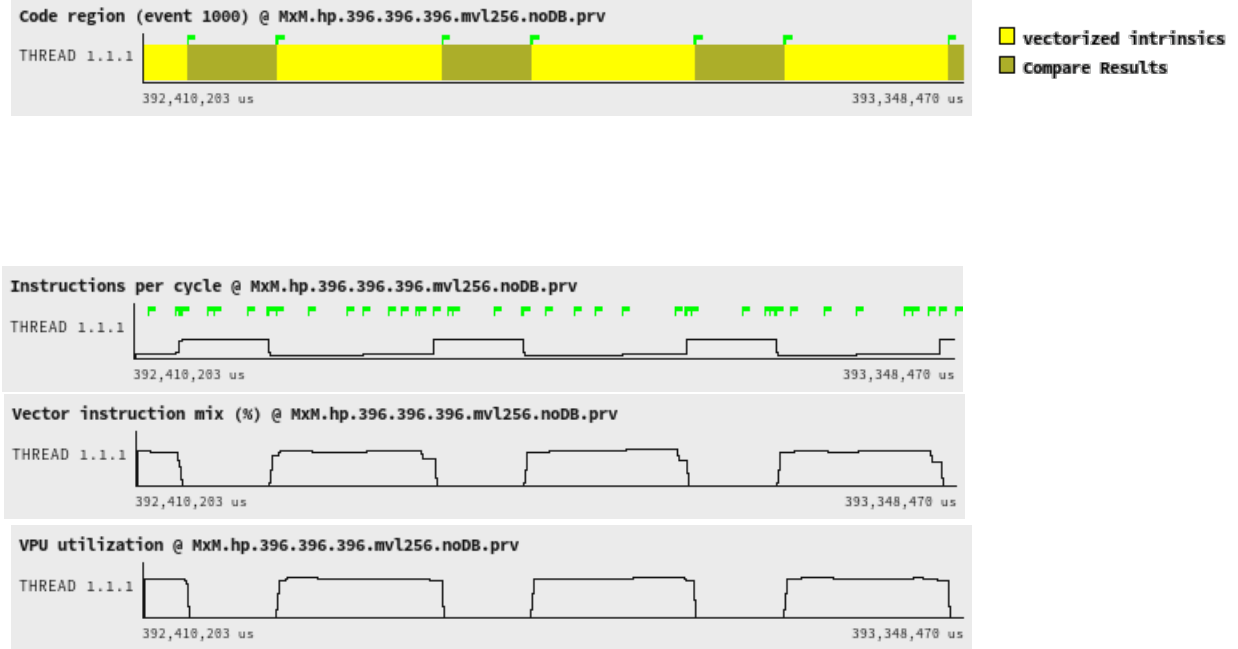
Vector instruction mix(%)

VPU utilization (%)

Arith. mix within vector (%)

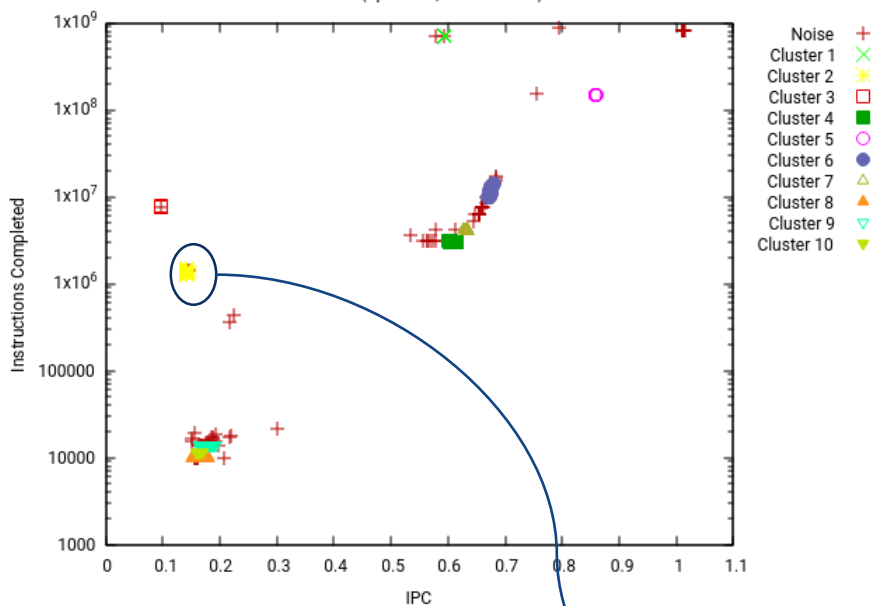
L1MPKI

DTLP MPKI



Analytics

Cluster Analysis Results of trace 'MxM.hp.396.396.mvl256.noDB.prv'
DBSCAN (Eps=0.01, MinPoints=8)

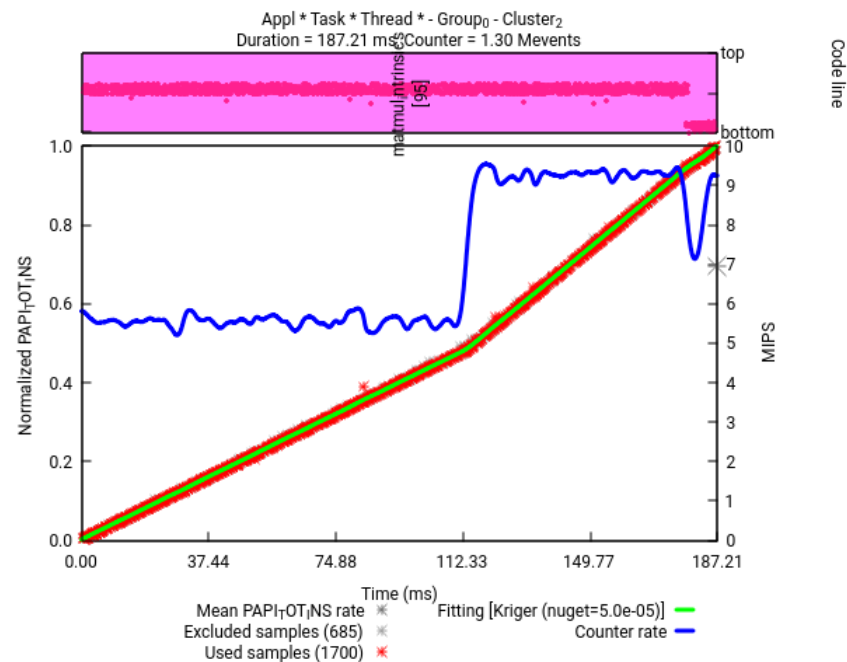


- Ref. MxM
- auto vec. columns
- vectorized intrinsics
- OpenBLAS
- ikj
- Compare Results

Code region (event 1000) @ MxM.hp.396.396.mvl256.noDB.prv



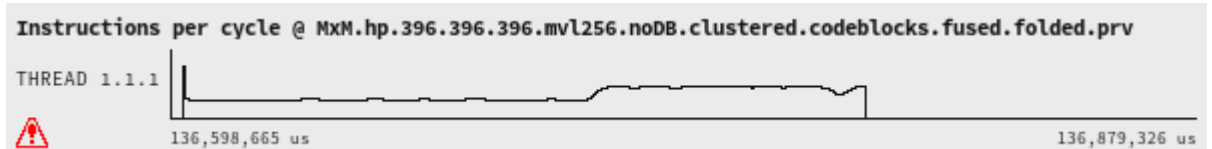
Cluster ID @ MxM.hp.396.396.mvl256.noDB.clustered.prv #1



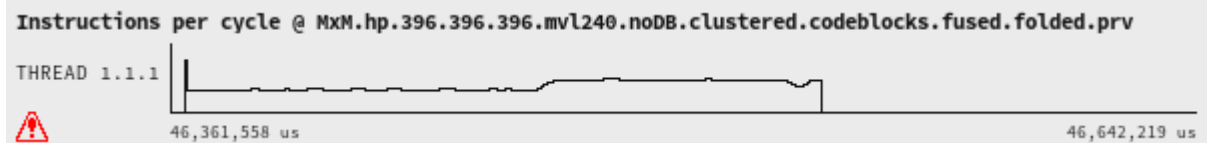
Analytics

- Impact of MAXVL

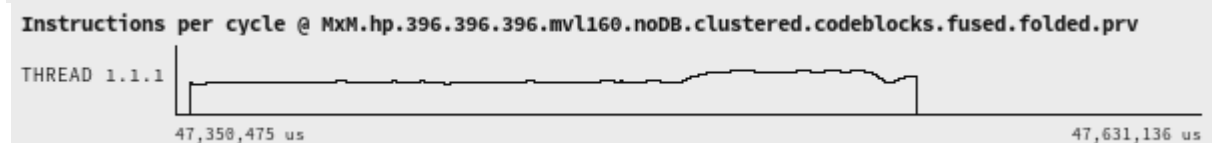
MVL 256



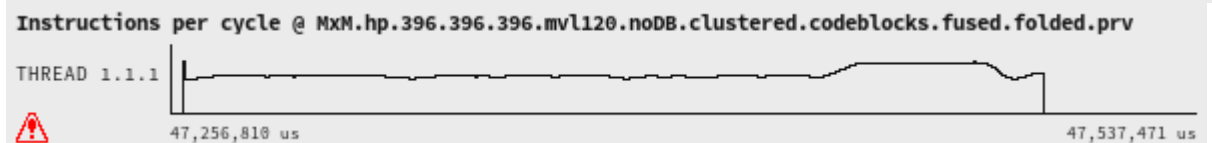
MVL 240



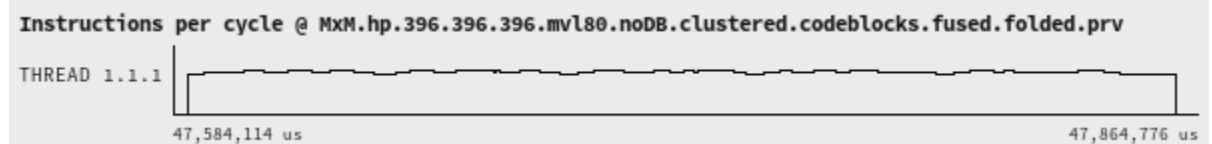
MVL 160



MVL 120



MVL 80





**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación

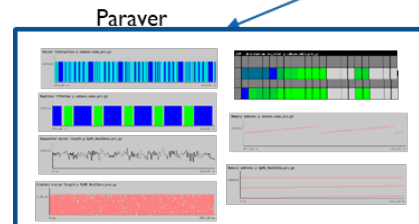
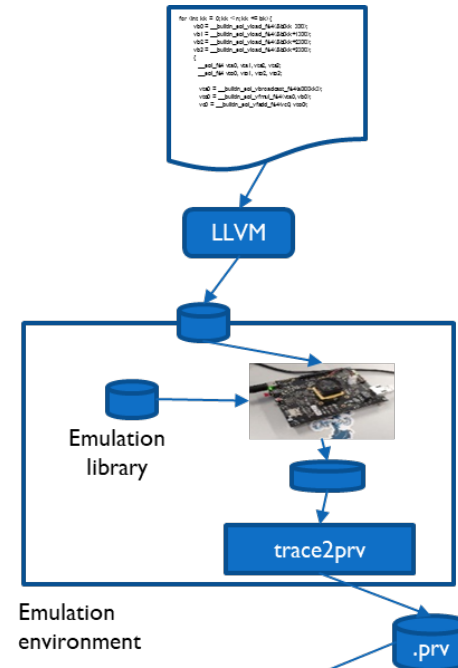


**EXCELENCIA
SEVERO
OCHOA**

Vehave

RVV Software emulation

- RISC-V **vector** instructions emulator
- Source code:
 - C + Vector Intrinsics / Autovectorized
- RISC-V 0.7 & 1.0
- Parameterized MAXVL
- Emits trace of vector instructions sequence
 - Opcode
 - Vector length
 - Register use
 - Memory address
- Highly detailed analytics in Paraver



- <https://repo.hca.bsc.es/gitlab/epi-public/risc-v-vector-simulation-environment>

How to compile and run

Makefile

```
#EPI_VERSION=  
EPI_VERSION=-0.7  
  
VEHAVE_DIR ?=/apps/riscv/vehave/EPI${EPI_VERSION}/development/include/vehave  
  
CC=clang  
CFLAGS=-mepi -O2 -fno-vectorize ... -I$(VEHAVE_DIR)
```

API

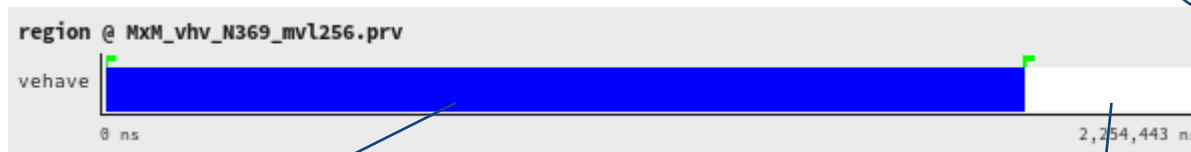
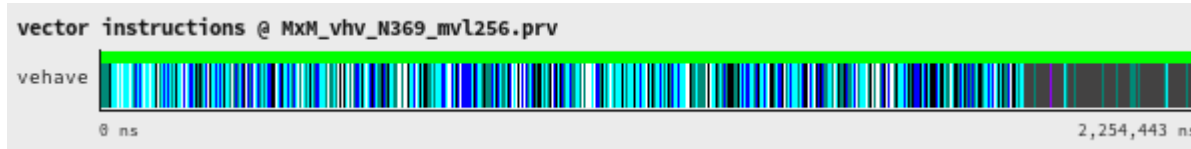
```
#include <vehave-control.h>  
...  
vehave_disable_tracing();  
...  
vehave_enable_tracing();  
...  
vehave_trace(2000, 1);  
...  
vehave_trace(2000, 1);  
...  
vehave_disable_tracing();  
...
```

execution

```
VEHAVE_HOME=/apps/riscv/vehave/EPI-0.7/development  
export VEHAVE_TRACE_SINGLE_THREAD=1  
export VEHAVE_TRACE_FILE="vehave.trace"  
export VEHAVE_DEBUG_LEVEL=0  
export VEHAVE_VECTOR_LENGTH=16384  
  
LD_PRELOAD=${VEHAVE_HOME}/lib64/libvehave.so my_code ...
```

Views: instructions & PC

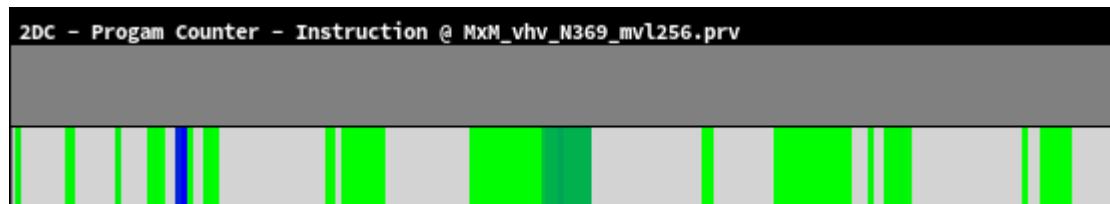
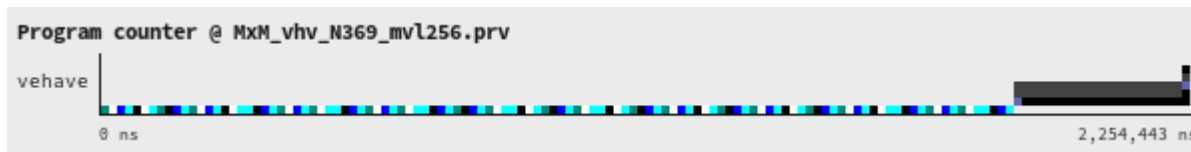
- vfadd
- vsetvl
- vsetvli
- vfmul
- vle
- vse
- vfmacc



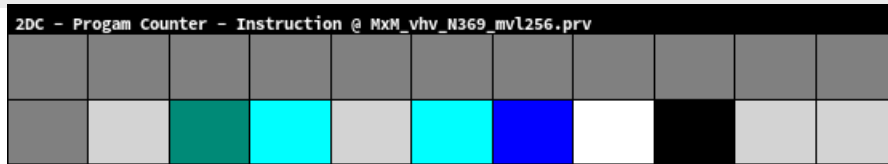
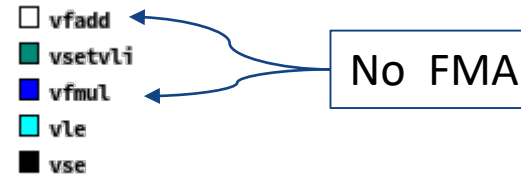
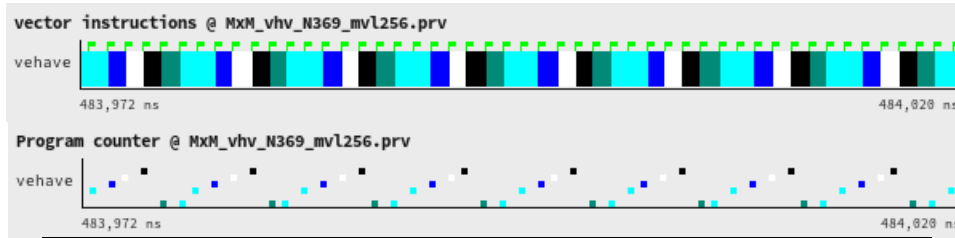
Actually
instruction

vfadd	vsetvli	vfmul	vle	vse
313,632	313,632	313,632	627,264	313,632

vsetvl	vsetvli	vle	vse	csrr	vfmv	vmv	vfmacc
816	26,166	13,464	792	1,632	4	816	313,632



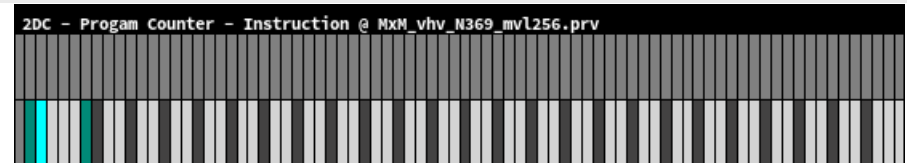
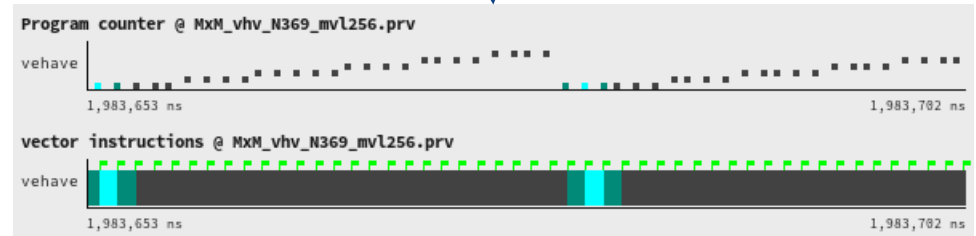
Views: instructions & PC



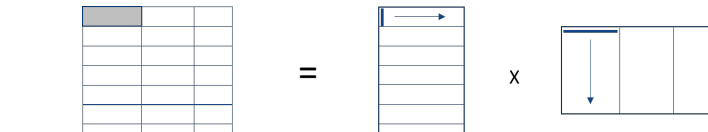
```
static void matmul_autovec_rows(int M, int K, int N, double (*c)[N],
                                double (*a)[K], double (*b)[N]) {
    for (int i = 0; i < M; i++) {
        for (int k = 0; k < K; k++) {
            #pragma clang loop vectorize(enable)
            for (int j = 0; j < N; j++) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}
```



footprint : Unroll



2 scalar instructions between fmaccs

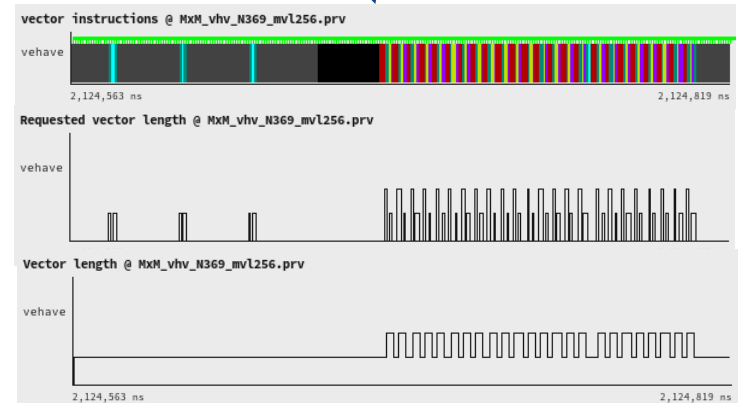
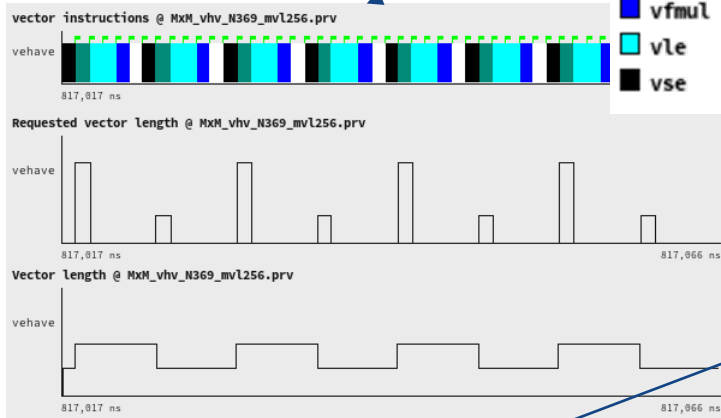
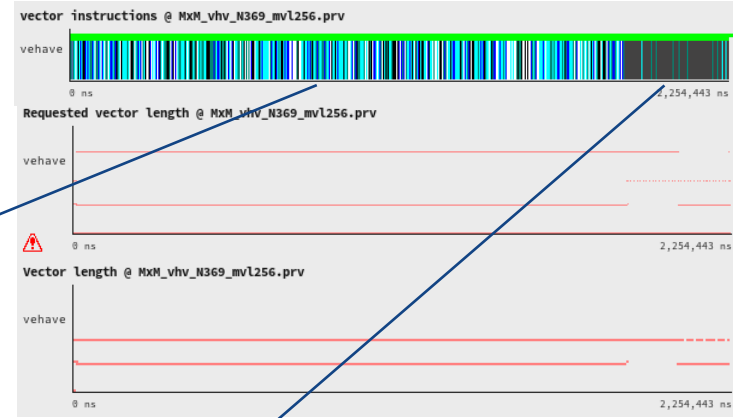


Views: vector length

```
static void matmul_autovec_rows(int M, int K, int N, double (*c)[N],
                                double (*a)[K], double (*b)[N]) {
    for (int i = 0; i < M; i++) {
        for (int k = 0; k < K; k++) {
            #pragma clang loop vectorize(enable)
            for (int j = 0; j < N; j++) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}
```



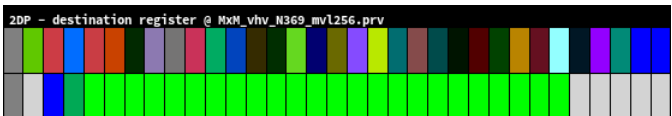
☐ vfadd
☒ vsetvli
☒ vfmul
☒ vle
☒ vse



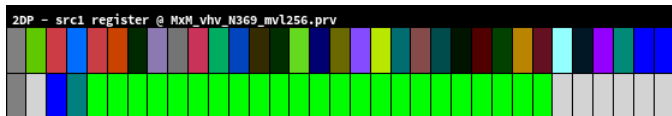
Views: registers

reg

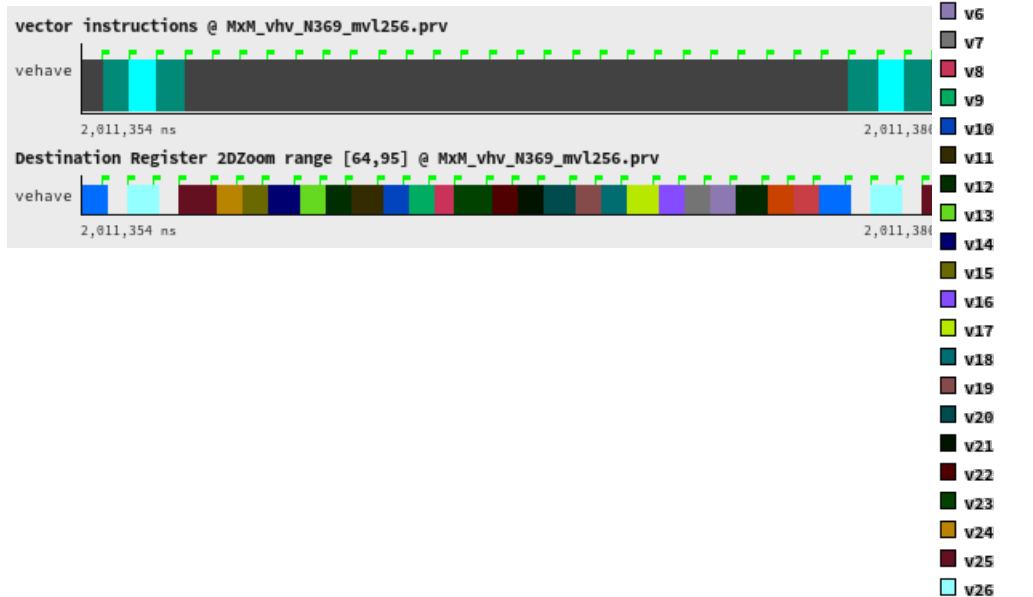
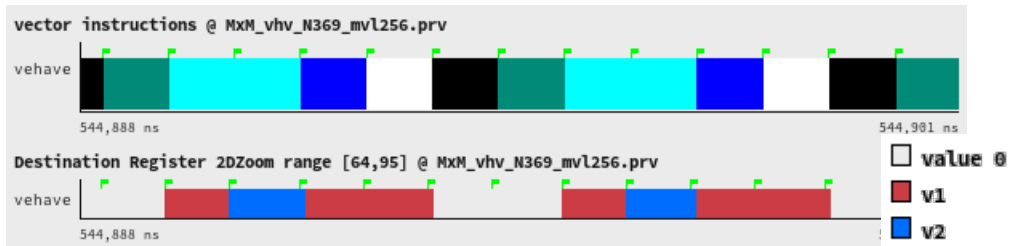
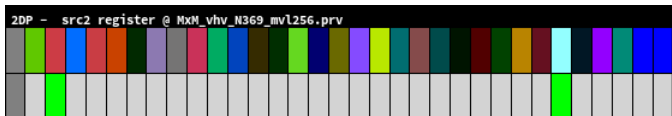
dst



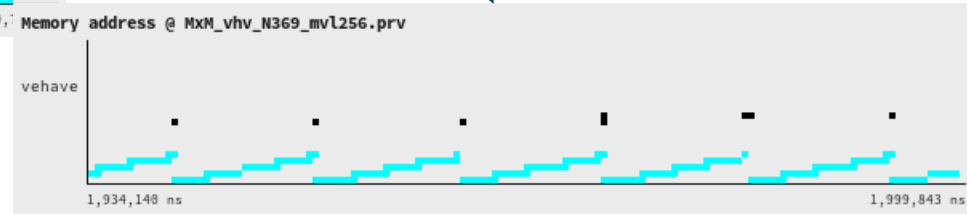
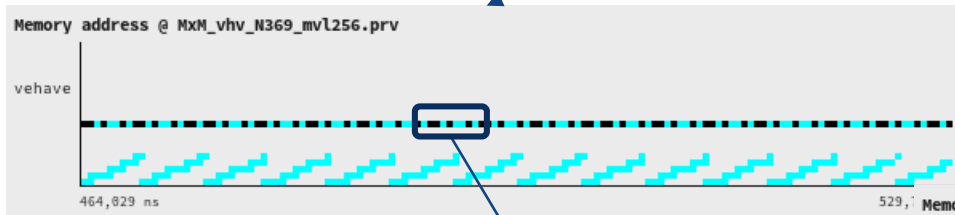
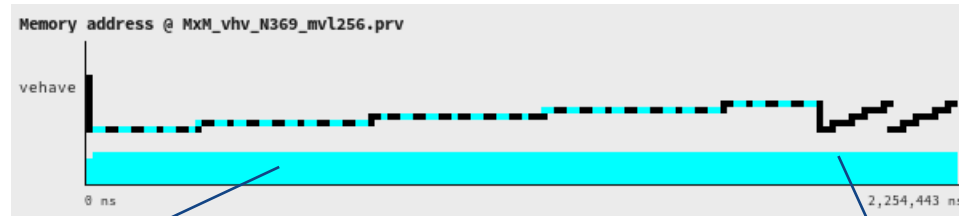
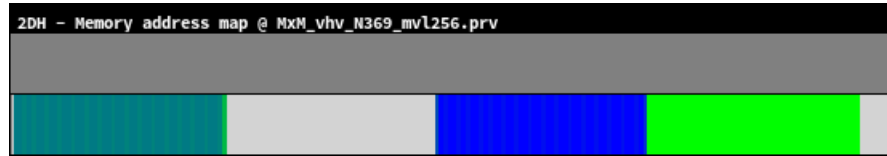
src1



src2



Views: memory addresses



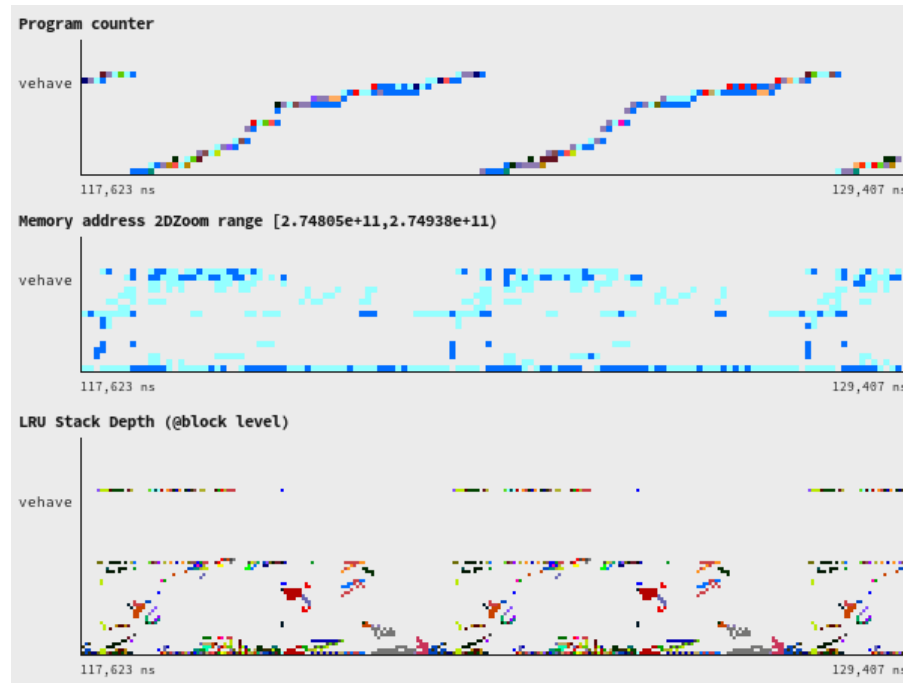
Views: other examples

Cloud_sc_microphysics miniapp

PC

Memory address

LRU stack depth



vehave

- Not ...
 - ... emulating scalar instructions
 - ... timing information
- But still ...
 - ... a LOT of useful insight on the instruction level program characteristics !!!
 - Instruction mix and sequence
 - Vector lengths
 - Scalar instructions between vector instructions
 - Memory address pattern
 - Register usage
 - ... for different MAXVL
 - ... to steer application development, architecture dimensioning & compiler



**Barcelona
Supercomputing
Center**

Centro Nacional de Supercomputación

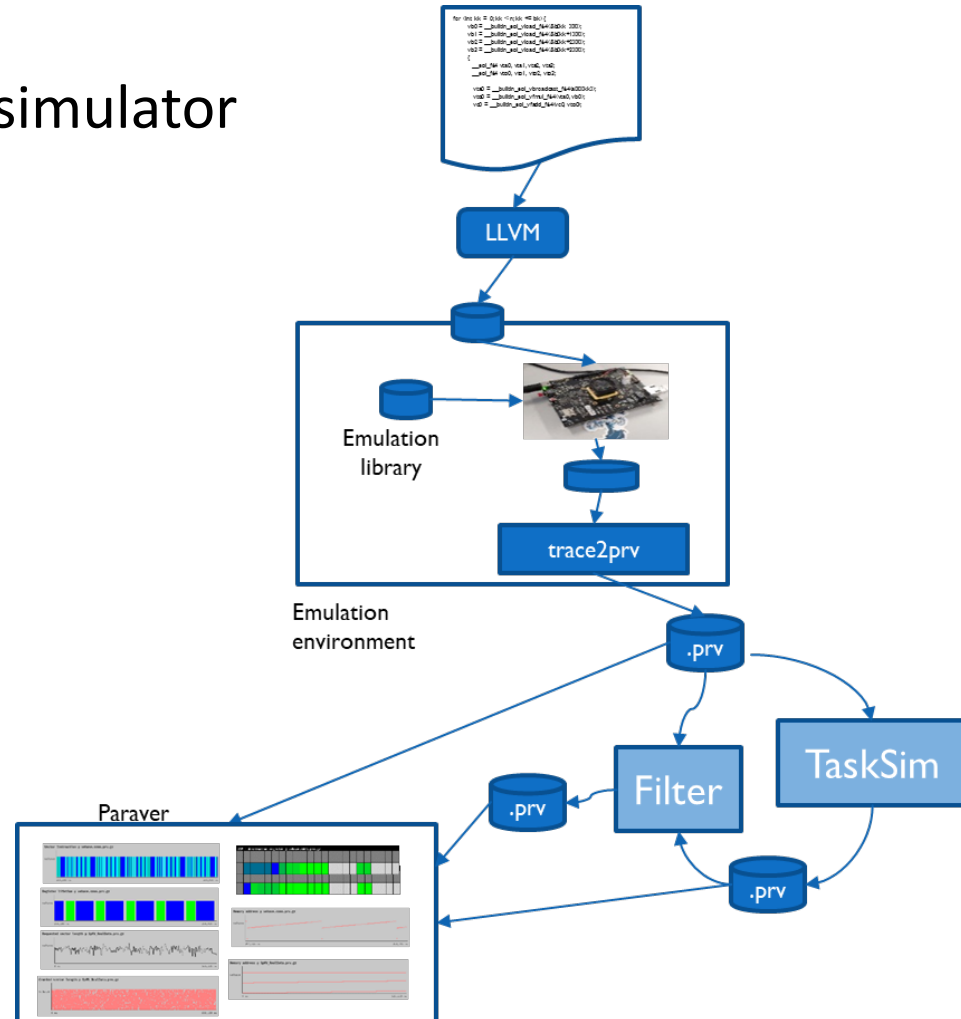


**EXCELENCIA
SEVERO
OCHOA**

MUSA

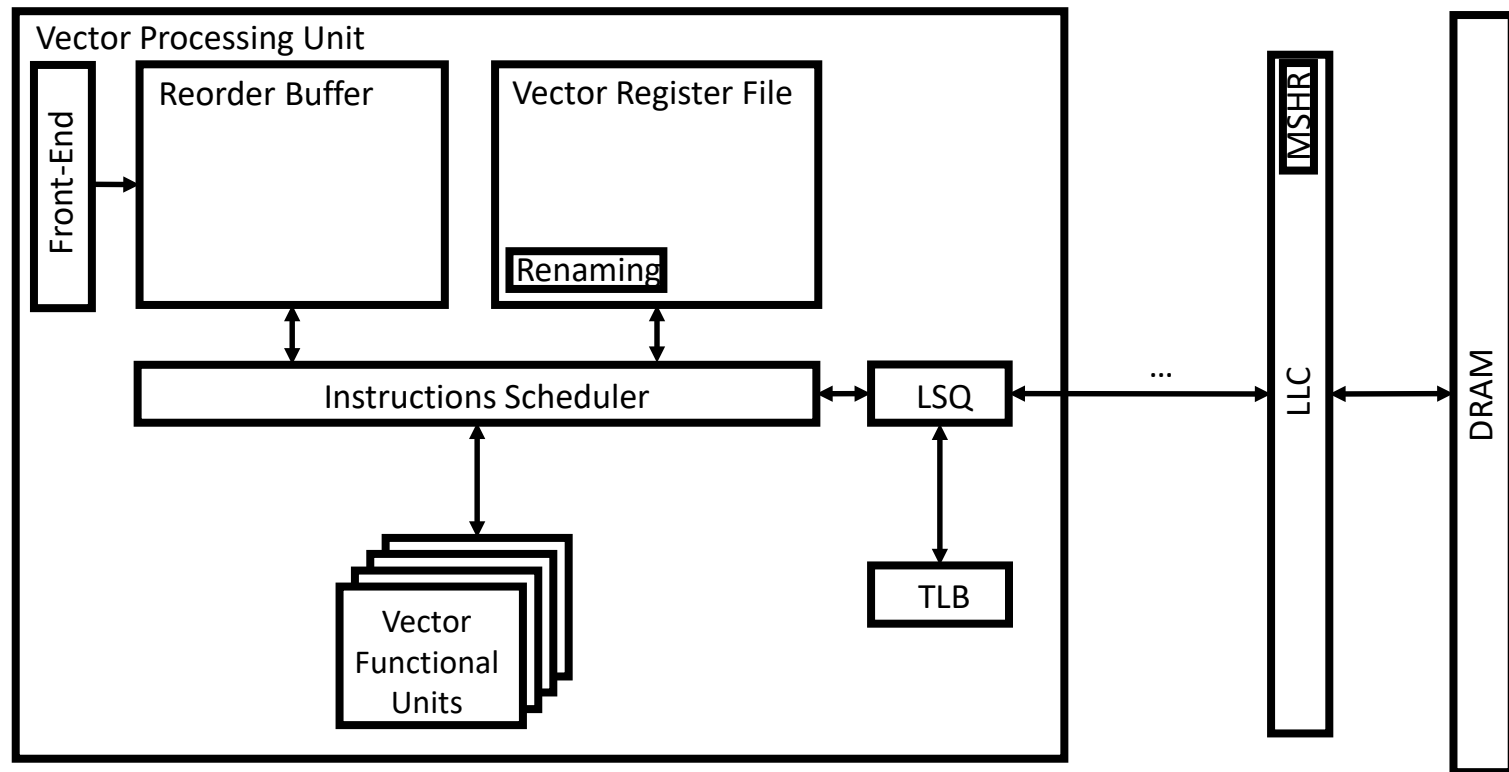
RVV Software emulation

- Trace driven (we have .prv trace) simulator
- Timing model
 - Instruction timing
 - Cache sizes and policies
 - Latencies
 - B/F
 - ...

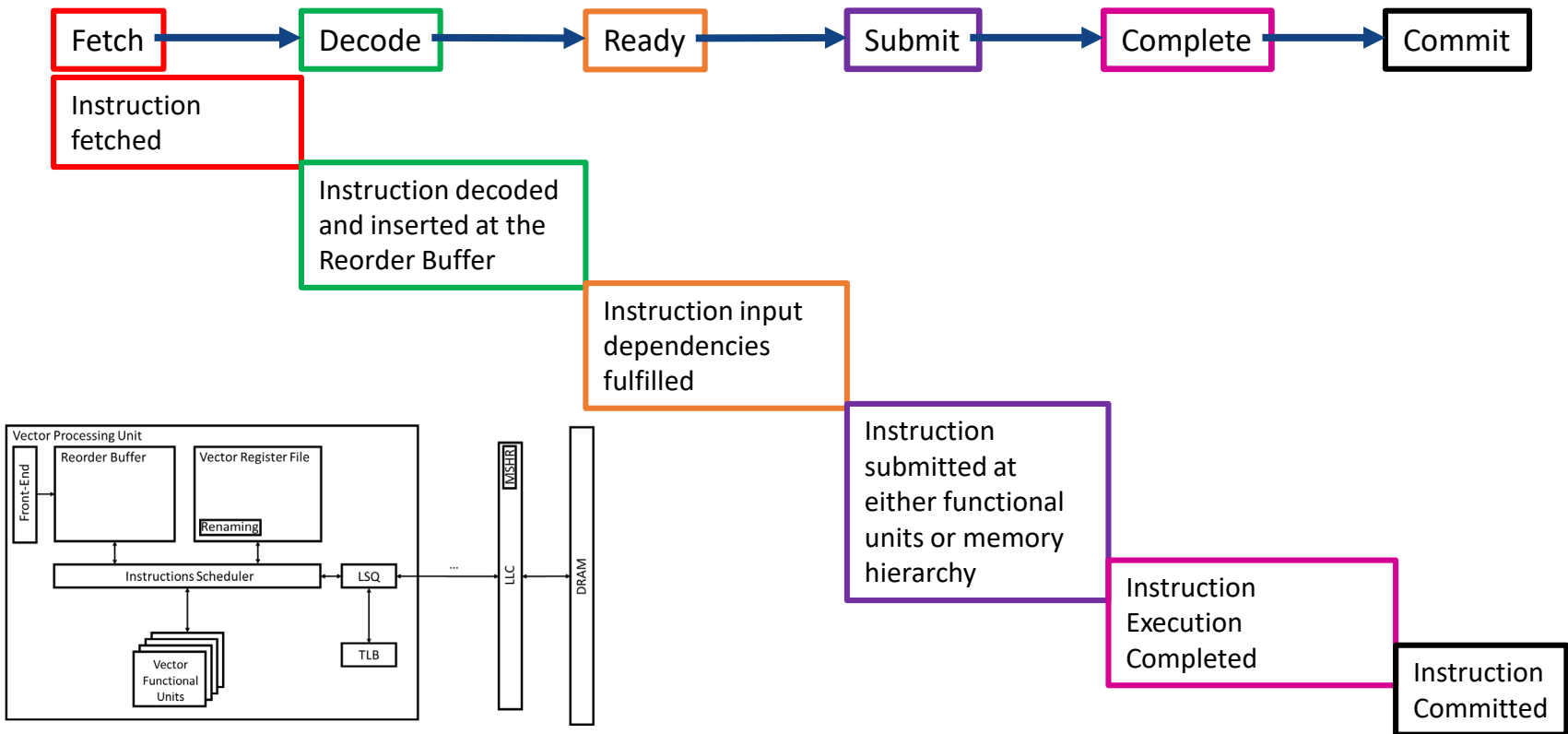


- <https://repo.hca.bsc.es/gitlab/epi-public/risc-v-vector-simulation-environment>

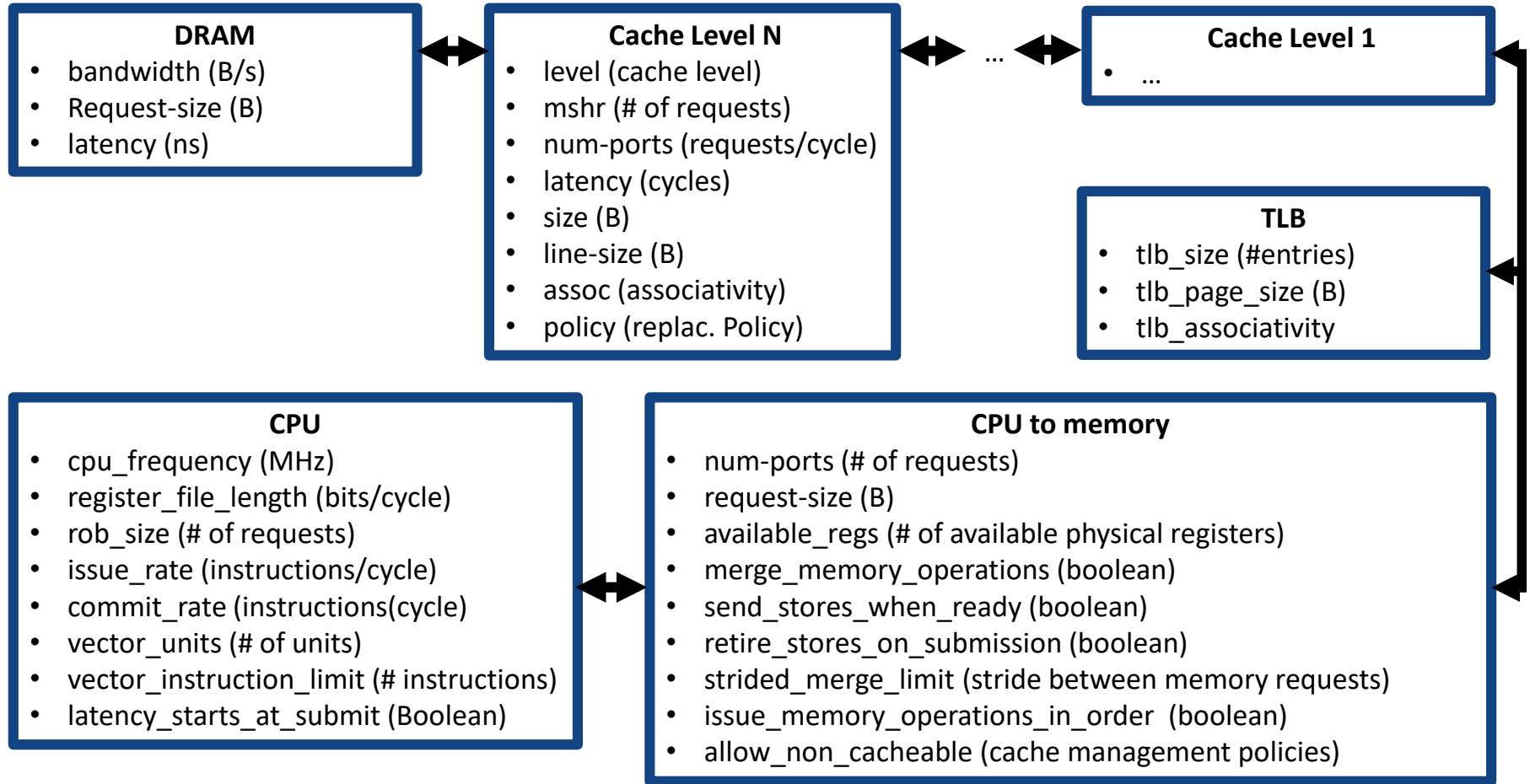
Timing Model's Vector Unit and Memory Hierarchy



Pipeline



Architecture parameters



Architecture parameters

Machine.conf

```
...
latency_file = file.lat
...
disable_paraver_trace = 0
...
[MemCPU]
rob_size = 8
...
available_regs = 40

[DL1Cache]
mshr = DL1MSHR
latency = 25
size = 32768
...
[DL1MSHR]
size = 16
...
[Memory]
bandwidth = 64000000000 # [bytes/s]
request-size = 64
latency = 120 # [ns]
```

```
# All instructions lines are "opcode base_
# Instructions that depend on the vector l
# With a "(granted_vector_length * single_
# scalar factors 0.3125, 2.8125, 3.515625
# latency: 40cycles, 360 cycles, 450cycles
ALU_FP_INST # Latency := base_latency + sc
vadd        25    1.25
vfadd       25    1.25
vfddiv      40    14.06
vfmacc      25    1.25
# All not spefcified instructions will hav
default     25    1.25
...
LU_FP_INST_NON_BLOCKING # Latency := base_
er_file_width
vfmv        20    1.25
vfmv.v.f    20    1.25

# Instructions that depend only on the vec
ALU_NON_FP_INST # Latency := base_latency
vrgather    0     1

# Instructions that have a constant latenc
NON_ALU_INST # Latency := base_latency
vsetvli     1     0
```

run

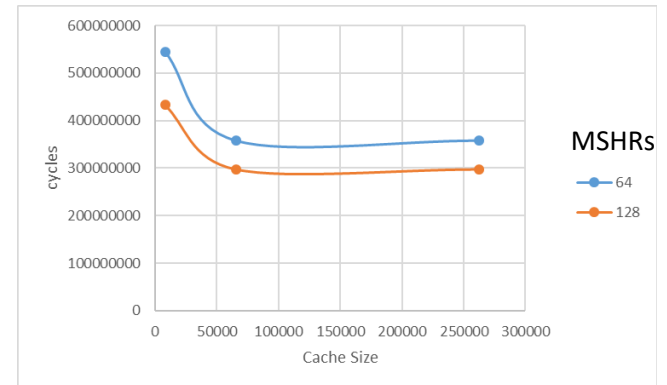
execute

```
tasksim.sh machine.conf output_name trace.prv
```

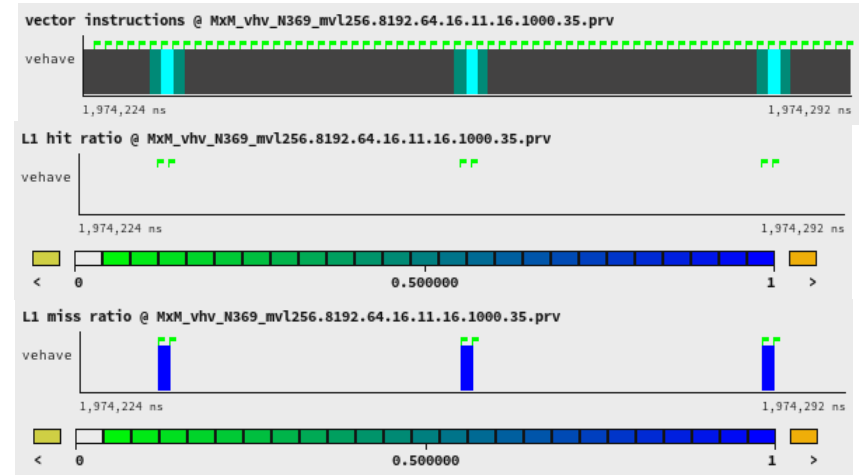
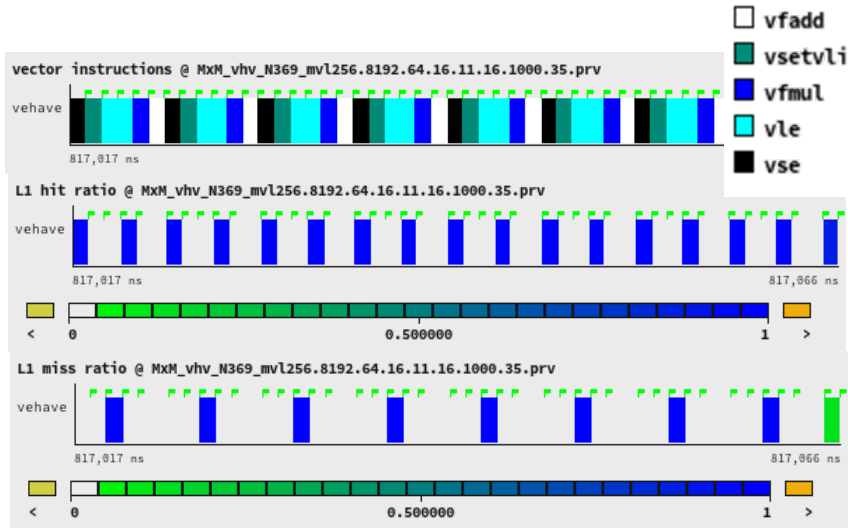
output_name.stats

```
...  
Final cycle count: 297162600  
...  
527358016:mem_written_bytes:CacheL1:0  
...  
379343:stalled cycles by a full MSHR:CacheL1:0  
...  
118009:write_miss:CacheL1:0  
7919208:write_hit:CacheL1:0  
157014:read_half_miss:CacheL1:0  
8121960:read_miss:CacheL1:0  
7899408:read_hit:CacheL1:0  
...  
137395:Number of write requests:DRAM:0  
8239969:Number of read requests:DRAM:0
```

+ output_name {.prv, .pcf, .row}



Views: cache behavior

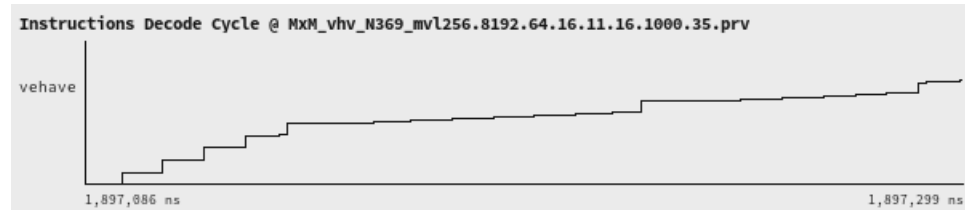


```
static void matmul_autovec_rows(int M, int K, int N, double (*c)[N],
                                double (*a)[K], double (*b)[N]) {
    for (int i = 0; i < M; i++) {
        for (int k = 0; k < K; k++) {
            #pragma clang loop vectorize(enable)
            for (int j = 0; j < N; j++) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}
```

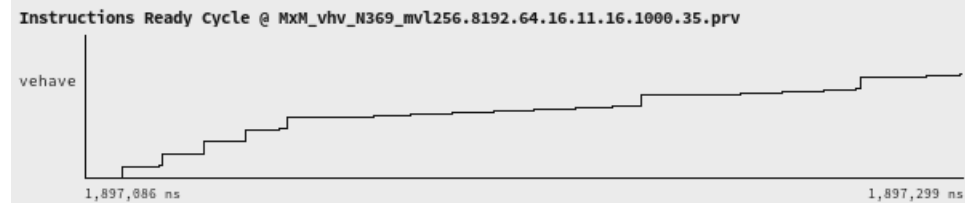


views: instruction execution timing

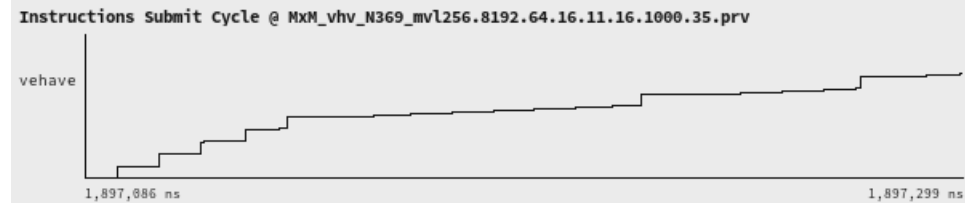
Decode



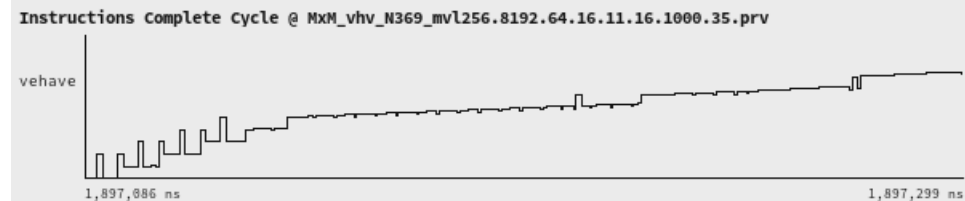
Ready



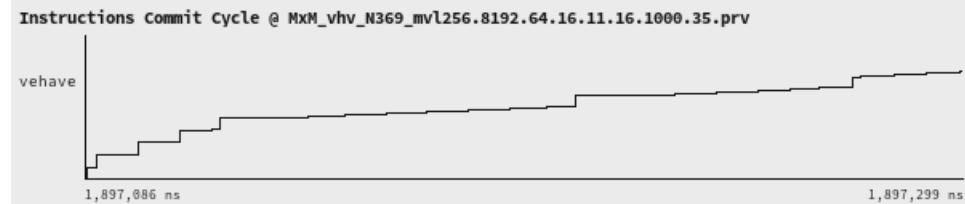
Submit



Complete



Graduate



views: instruction execution timing

Time in ROB
(commit – decode)

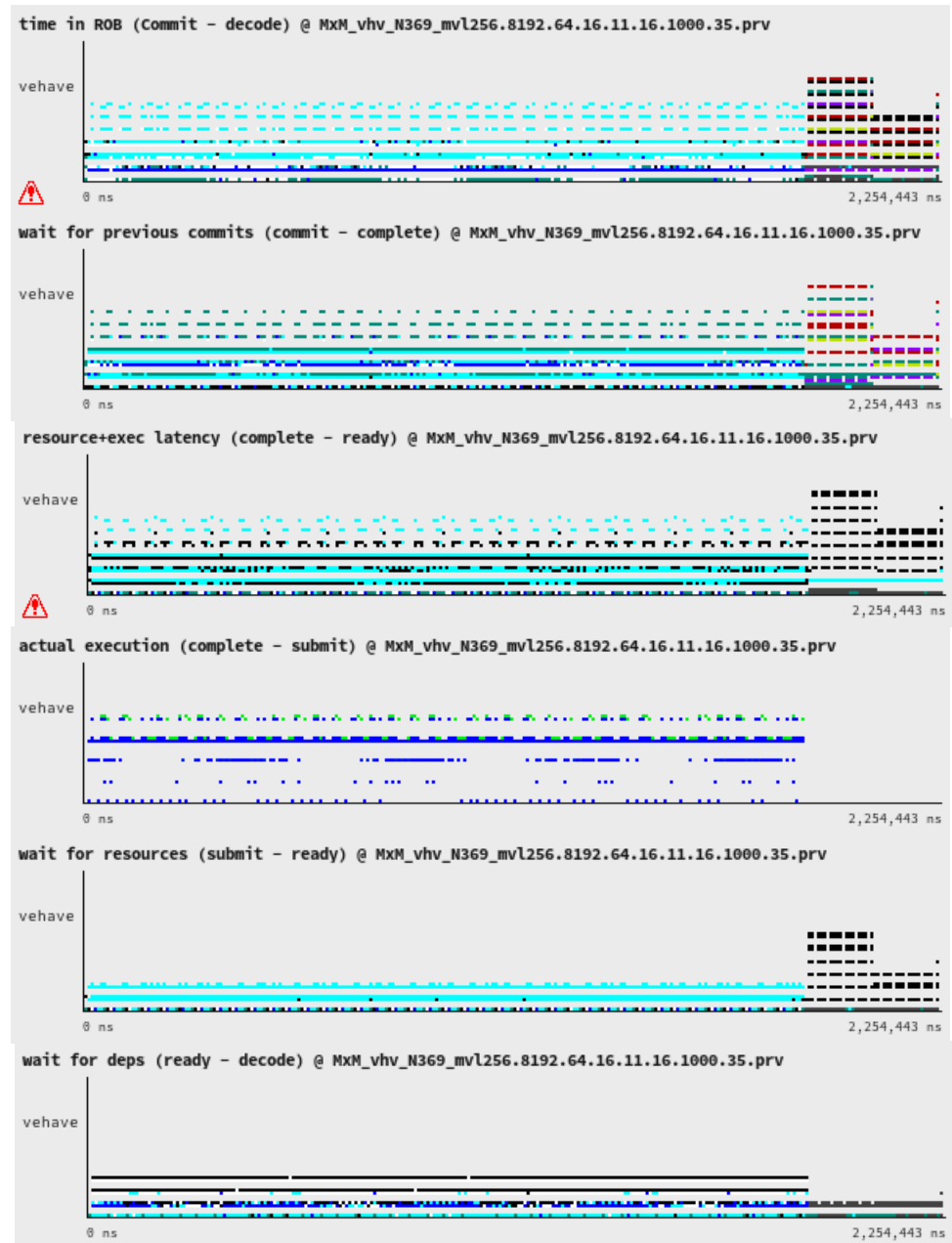
Wait for previous commits
(commit – complete)

Resource + exec latency
(complete – ready)

Actual execution
(complete – submit)

Wait for resources
(submit – ready)

Wait for deps
(Ready – Decode)



views: instruction execution timing

Time in ROB
(commit - decode)

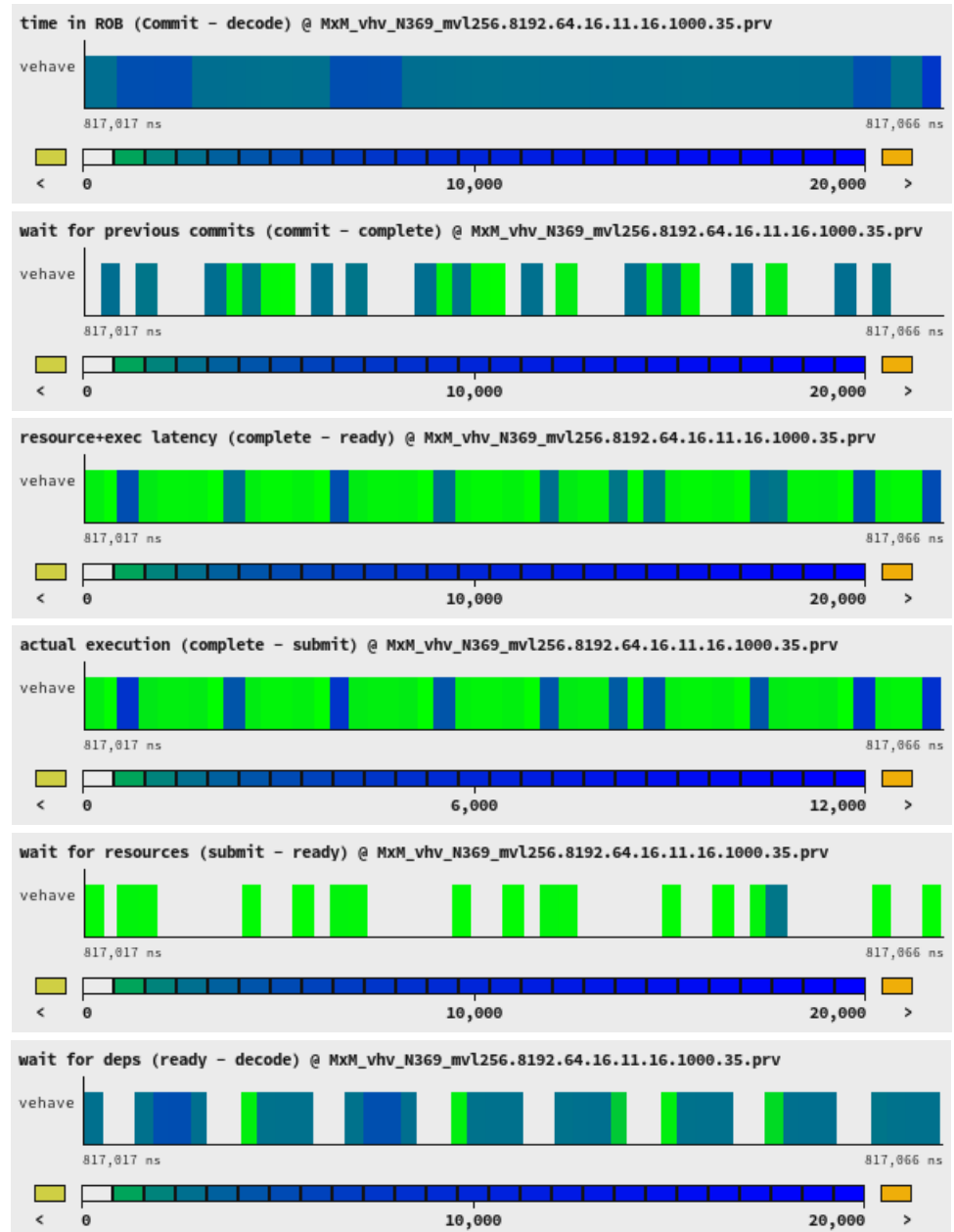
Wait for previous commits
(commit - complete)

Resource + exec latency
(complete - ready)

Actual execution
(complete - submit)

Wait for resources
(submit - ready)

Wait for deps
(Ready - Decode)



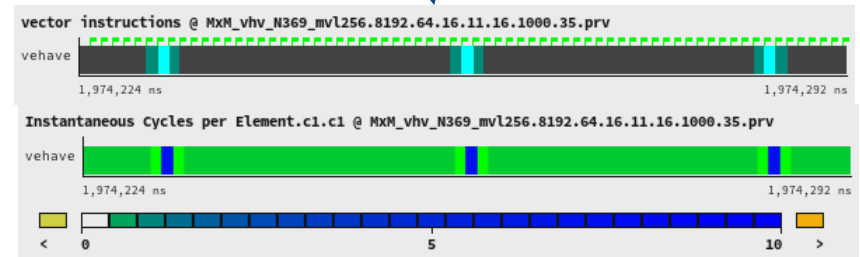
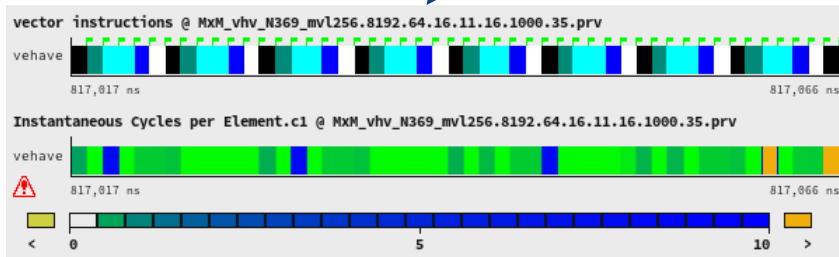
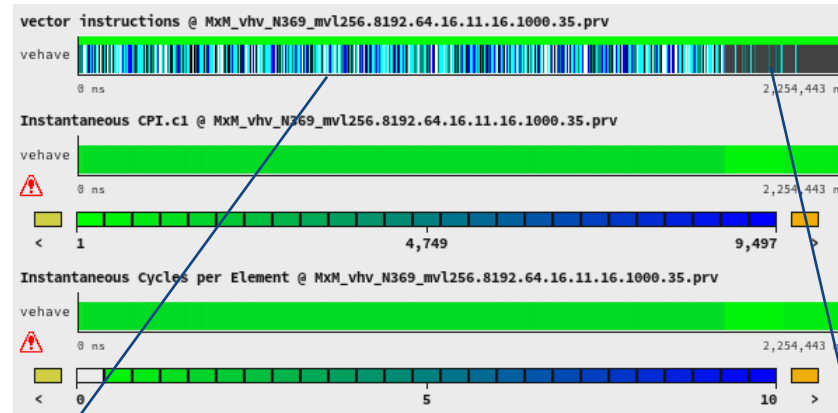
Views: “instantaneous” metrics

Instructions

CPI

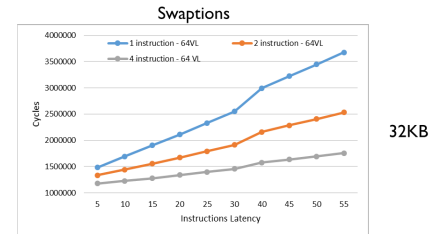
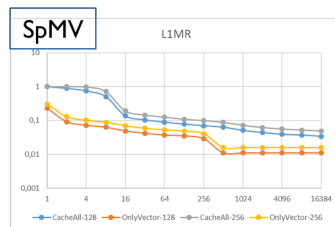
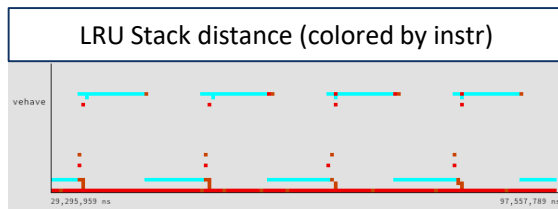
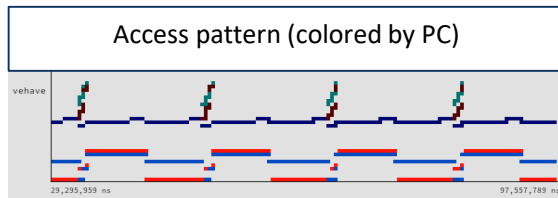
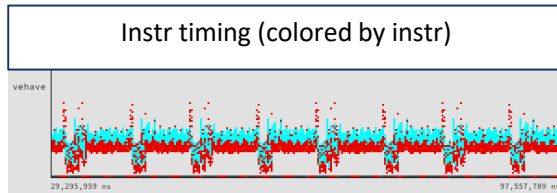
Cycles per Element

Derived from
execution, vl, ...
events

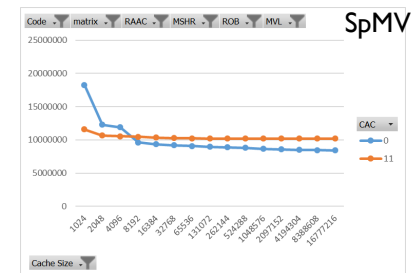
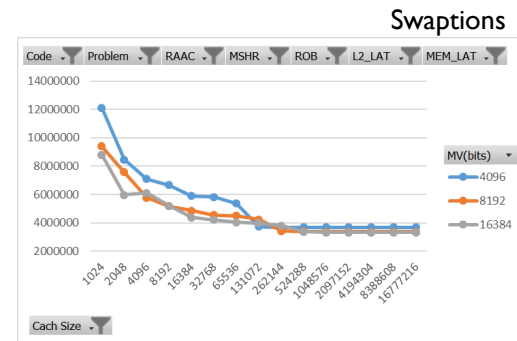


RVV Software emulation

- Detailed analysis & insight



32KB



RVV Software emulation

- Yolo: AI

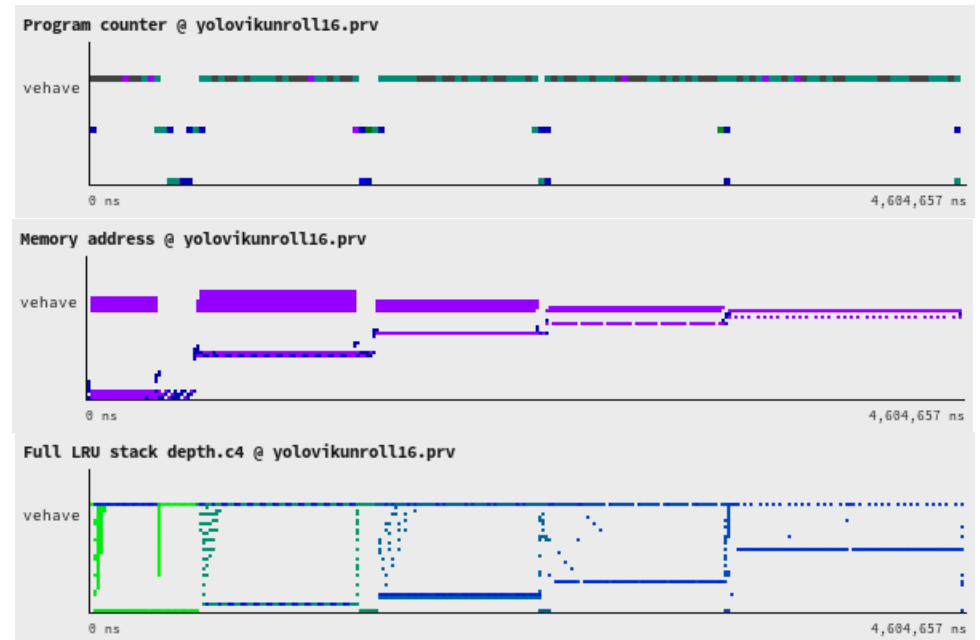
Program counter

Instruction? Order? Footprint?
Dependences? ROB size?

Memory address

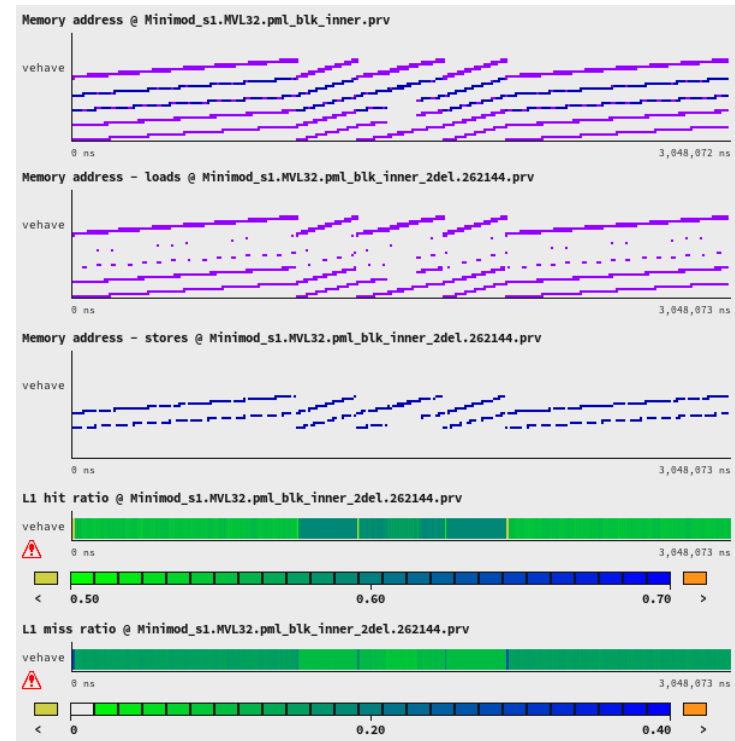
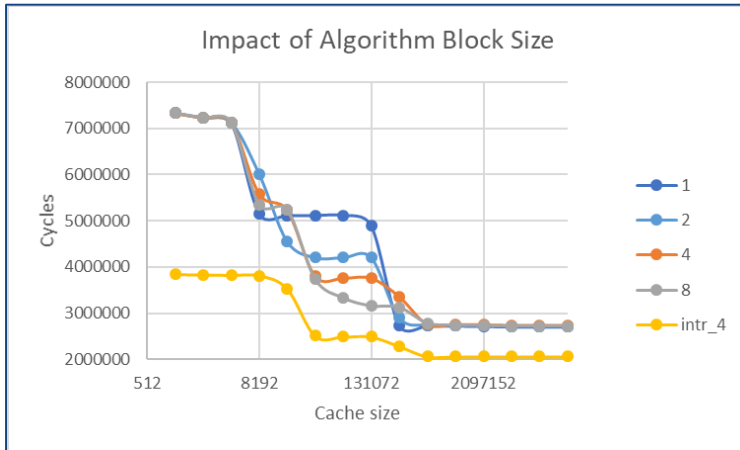
Cache management hints

LRU stack depth

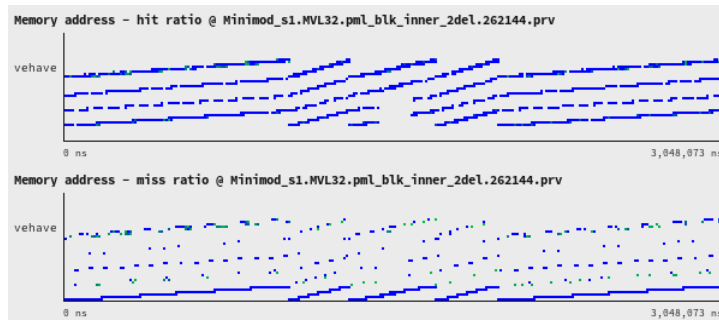


RVV Software emulation

- Geophysics



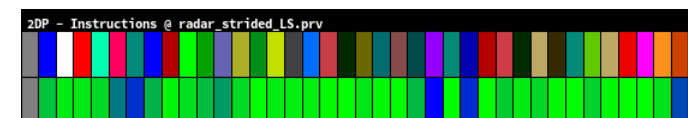
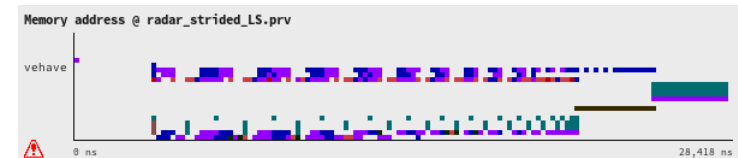
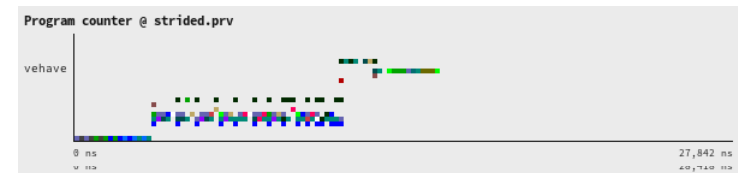
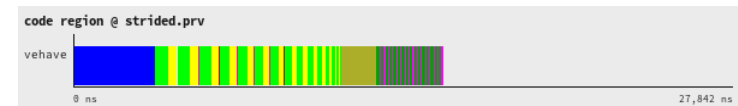
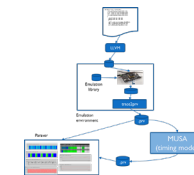
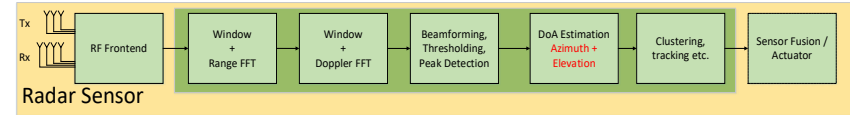
256K cache



RVV Software emulation



- Radar miniapp
 - Directives + automatic vectorization
 - Incremental way
- Steering compiler optimizations
 - Complex data types
 - Indexed \rightarrow strided
 - Reuse through registers
 - Avoid optimizations generating extremely short vector lengths
- Steering code refactoring
 - With productivity in mind !
 - Interchange, collapse, ...





**Barcelona
Supercomputing
Center**

Centro Nacional de Supercomputación



**EXCELENCIA
SEVERO
OCHOA**

SDV@FPGA

Logic Analyzer

ILA waveforms → .prv

Armed trigger (0..32)

Offset of trigger in
emitted trace (0..32K)

```
#Arm trigger: from pickle-X
sudo /apps/riscv/fpga-sdv/ila2prv/sdv-ila.sh 6 16384 file.ila

# Run program in fpga-sdv-X
# file.ila will be generated when trigger macro hit by the program

#Convert .ila trace to .prv: from pickle-X
module load llvm/EPI-0.7-development-cross
/apps/riscv/fpga-sdv/ila2prv/ila2prv file.ila .
```

fpga-sdv-2
(RV – vector)

Pickle-2
(x86)

Generates file.prv, .pcf, .row

<https://repo.hca.bsc.es/gitlab/epi-public/risc-v-vector-simulation-environment/-/wikis/Obtaining-ILA-traces>

SDV parameter sweep

- MAXVL

```
# change MAXVL from fpga-sdv-X  
./change_vl 128
```

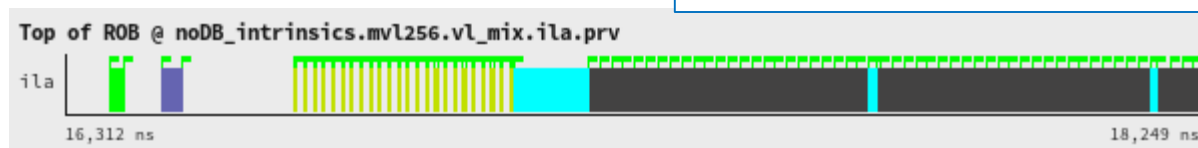
```
# API
```

- Memory latency

```
# change latency: from pickle-X  
sudo /apps/riscv/fpga-sdv/latency_changer/latency_changer 32
```

ILA views: instructions

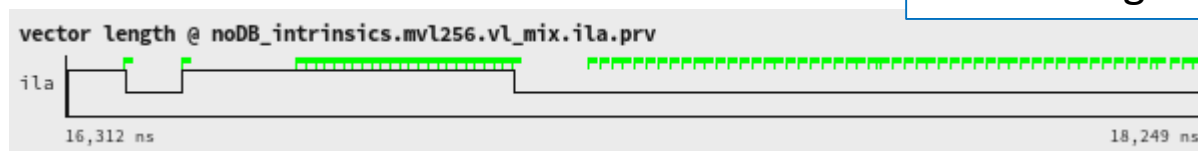
Instruction @ Top of ROB



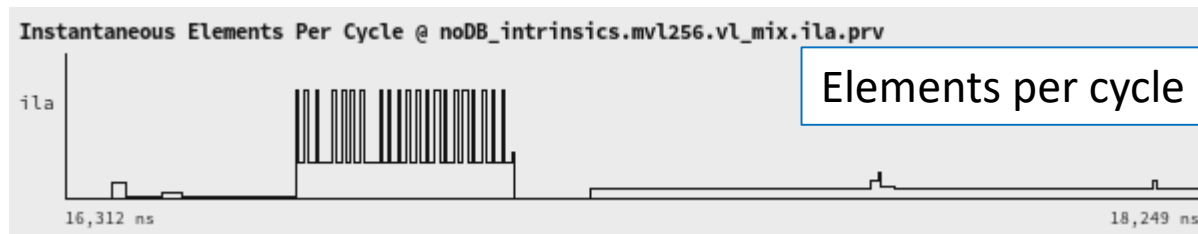
value 0
vor
vle
vfmv
vmv
vfmaccc

“cycles”

Vector length



Elements per cycle



Derived from vector length
and duration of instruction
at top of ROB

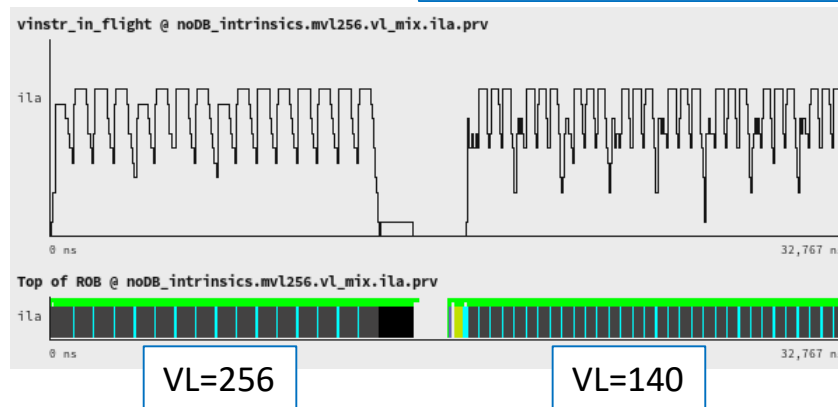
Why very efficient vmv ? Smart renaming based implementation

Profiles: #instr, avg “duration”, ...

	vor	vle	vse	vfmv	vmv	vfmaccc
ila	25 ns	14.404 ns	59.375 ns	35 ns	4 ns	24.792 ns

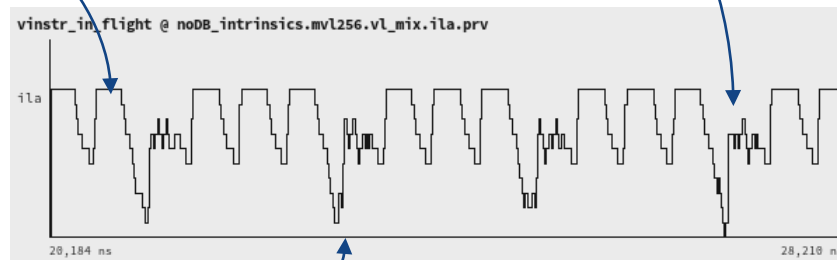
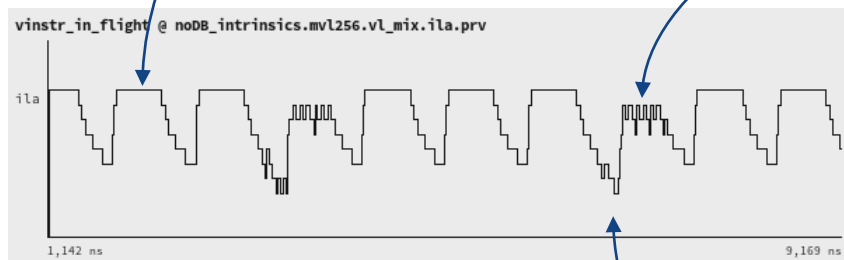
ILA views: execution

Instructions in flight



Shorter plateaus
No issue periods

Pulsations:
Less 1 every 4 iterations ?



Closer to idle VPU?

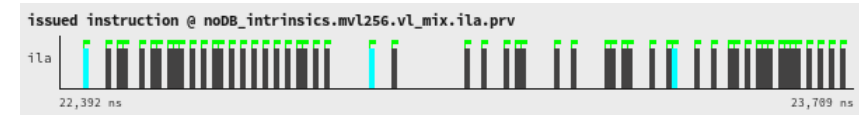
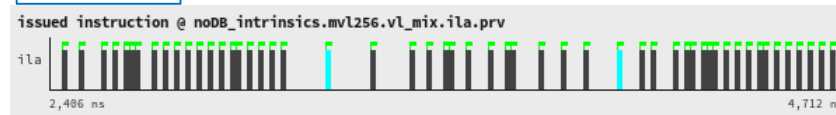
Why lower @ lower VL ?
Danger if larger memory latency?

ILA views: execution

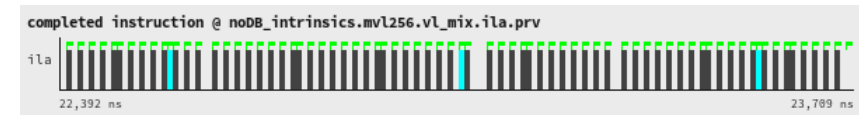
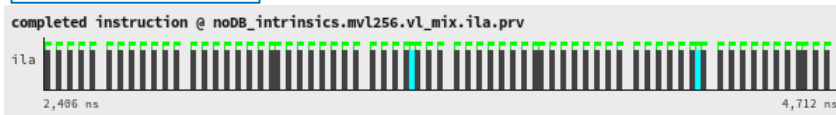
Top of ROB



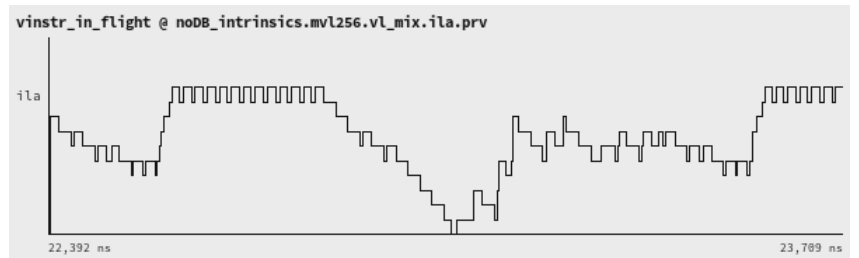
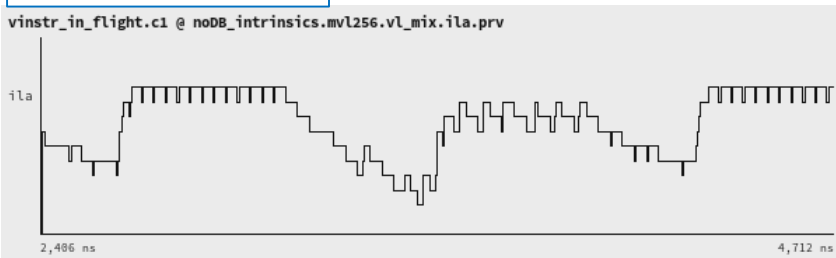
Issued



Graduated



instr in flight



VL=256

Why? Important?
FE issue? Cache issue?
Need more info ?

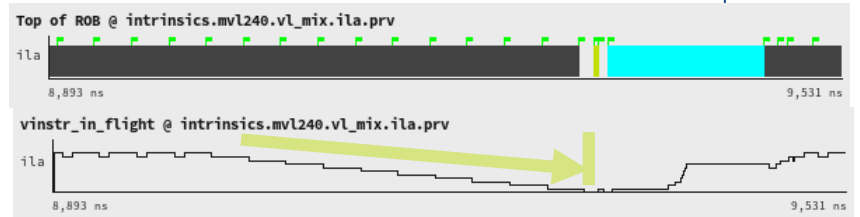
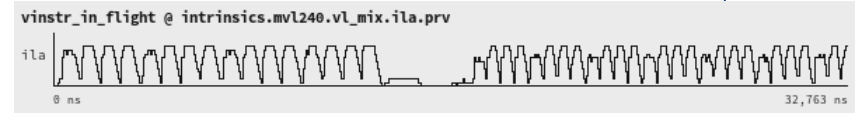
VL=140

← Not same time scale →

ILA views: execution

Double buffer to
overlap load latency

```
void matmul_intrinsics(int M, int K, int N, int bi, int bj, int bk,
    double (* restrict c)[N], double (* restrict a)[K], double (* restrict b)[N]) {
    assert(bi == 24); assert(bk == 1); ...
    int ii;
    for (int jj = 0; jj < N; ) { // system selected column block size (C & B)
        unsigned long gvl = __builtin_epi_vsetvli(N-jj, __epi_e64, __epi_m1);
        for (ii = 0; ii < M-(bi-1); ii += bi) {
            __epi_1xf64 vc0, vc1, ..., vc23;
            __epi_1xf64 vb0, vb1;
            vc0 = __builtin_epi_vload_1xf64(&c[ii][jj], gvl);
            vc1 = __builtin_epi_vload_1xf64(&c[ii+1][jj], gvl);
            ...
            vc23 = __builtin_epi_vload_1xf64(&c[ii+23][jj], gvl);
            vb1 = __builtin_epi_vload_nt_1xf64(&b[0][jj], gvl);
            for (int kk = 0; kk < K; kk += bk) {
                vb0 = vb1;
                if (kk < K-1) vb1 = __builtin_epi_vload_nt_1xf64(&b[kk+1][jj], gvl);
                FMA( vc0, vb0, a[ii][kk], gvl );
                FMA( vc1, vb0, a[ii+1][kk], gvl );
                ...
                FMA( vc23, vb0, a[ii+23][kk], gvl );
            }
            __builtin_epi_vstore_1xf64(&c[ii][jj], vc0, gvl);
            __builtin_epi_vstore_1xf64(&c[ii+1][jj], vc1, gvl);
            ...
            __builtin_epi_vstore_1xf64(&c[ii+23][jj], vc23, gvl);
        }
        jj += gvl;
    }
}
```



```
.Ltmp76:
    #DEBUG_VALUE: vc22 <- undef
    .loc 1 103 12                                # mxm_intrinsics.c:103:12
    add    a1, a1, s2
    fld    ft0, 0(a1)

.Ltmp77:
    #DEBUG_VALUE: vta <- undef
    vfmacv vf      v2, ft0, v27

.Ltmp78:
    #DEBUG_VALUE: kk <- $x9
    #DEBUG_VALUE: vc23 <- undef
    .loc 1 0 12 is_stmt 0                        # mxm_intrinsics.c:0:12
    rdvtype a1
    rdvl    a3
    vsetvli zero, zero, e64,m1
    vmv.v.v v27, v26
    vsetvli zero, a3, a1
    add     a1, zero, s1
```

Differences between mental
models and actual behavior

ILA views: Memory

Top of ROB

Memory op. execution

Memory Op “Latency”

Data strobos (line)



Variability in data arrival cycles ?
Can be reduced?



**Barcelona
Supercomputing
Center**

Centro Nacional de Supercomputación

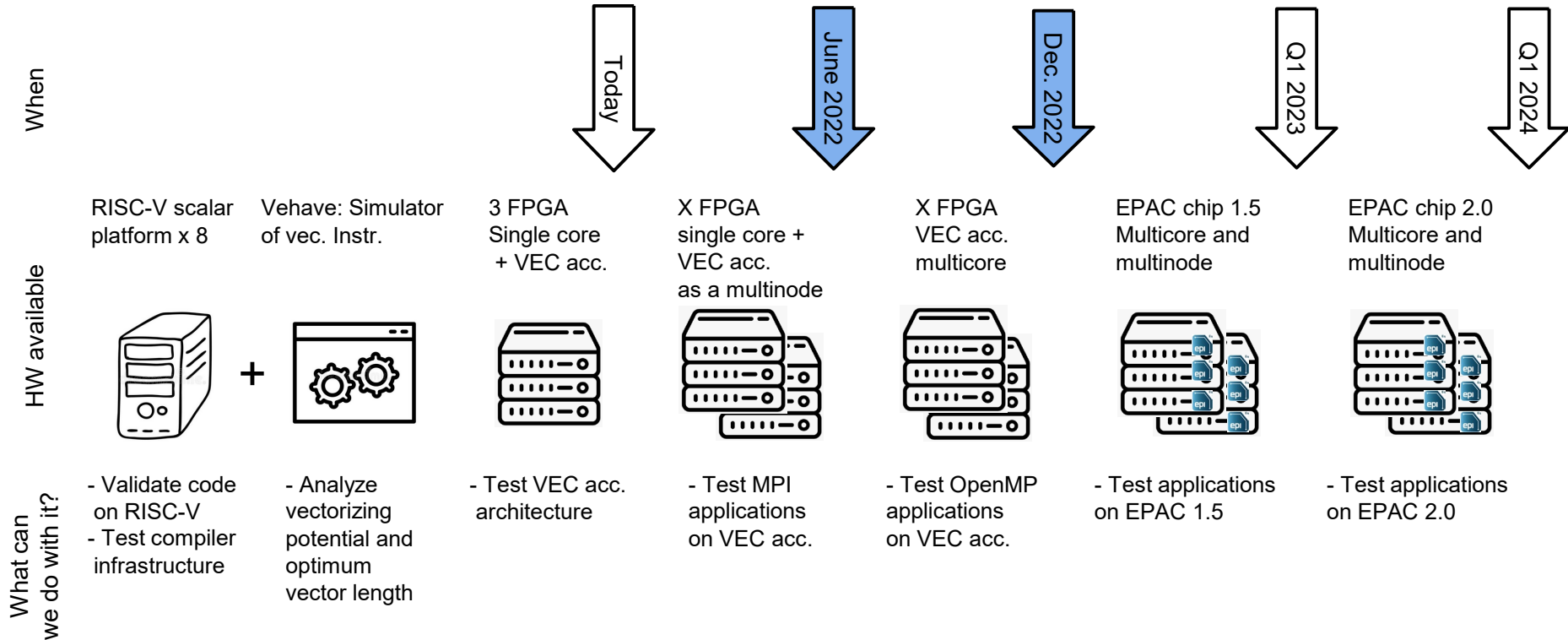


**EXCELENCIA
SEVERO
OCHOA**

SDV@FPGA

Roadmap

EPI SDV roadmap





**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación



**EXCELENCIA
SEVERO
OCHOA**

Thanks

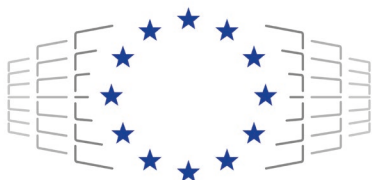
EPI PARTNERS



EVIDEN



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