RISC-V in HPC
A look into tools for performance monitoring

ISC25, Hamburg

Fabio Banchelli * (BSC)

Rafel Albert Bros Esqueu (BSC)

Tiago Rocha (INESC-ID IST)

Nuno Roma (INESC-ID IST)

Pedro Tomás (INESC-ID IST)

Nuno Neves (INESC-ID IST)

Filippo Mantovani (BSC)

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

European Processor Initiative

Context

About - me

- Last year PhD student at Barcelona Supercomputing Center (BSC)
- Working at BSC since 2017
- Contributed to EU projects
  - Mont-Blanc 3
  - European Processor Initiative
- Advisor of the NotOnlyFLOPs team at the Student Cluster Competition
  - Always running with “weird” hardware!
  - ISC22 we brought a cluster made of 12 HiFive Unmatched

About - European Processor Initiative

- **Goals**
  - Promote European technology in the HPC space
  - Strengthen competitiveness of EU industry and science
- **Outcomes**
  - General purpose CPU based on Arm (Rhea)
  - Accelerator based on RISC-V (EPAC)
- **Much more than making hardware**
  - System software (compiler + libraries) targeting RISC-V “V” extension
  - FPGA-based prototype of EPAC
  - Commercial RISC-V platforms such as HiFive Unmatched, Milk-V Pioneer, etc.
  - Performance analysis tools and methodology

Software Development Vehicles (SDVs)

About - FPGA-based prototype of EPAC (VEC)

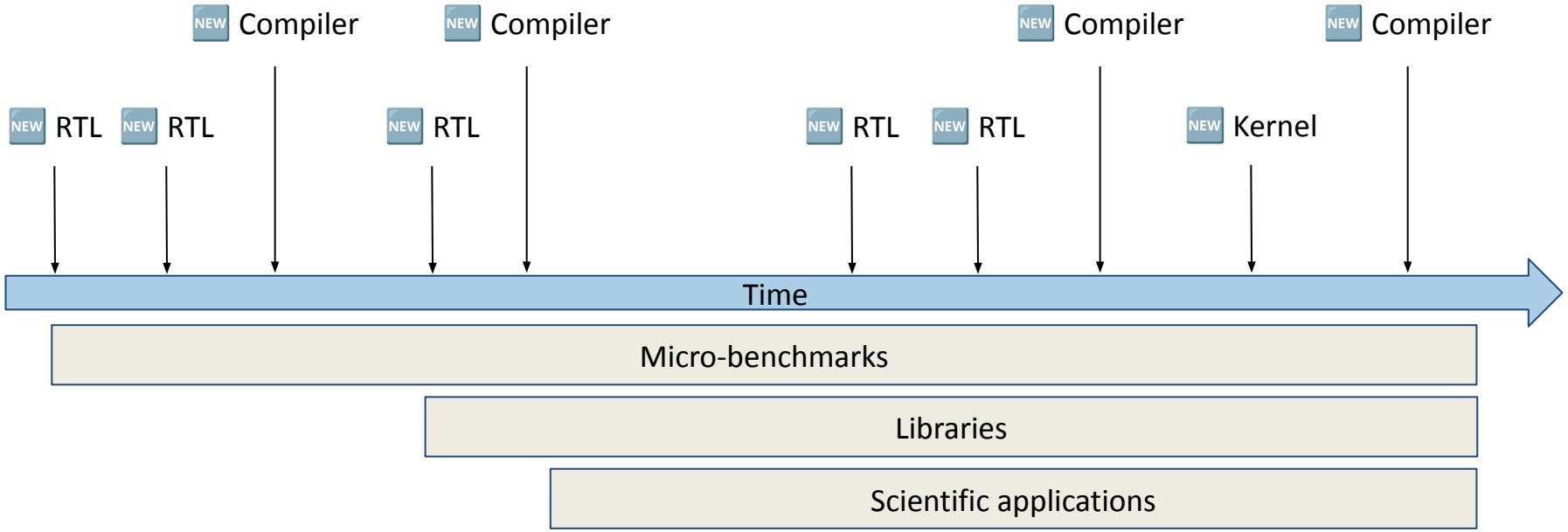
- Hardware emulation of the chip
  - Cycle accurate
  - Low clock frequency (50MHz)

- Self-hosted system
  - Running standard Linux Ubuntu
  - Traditional command line interface

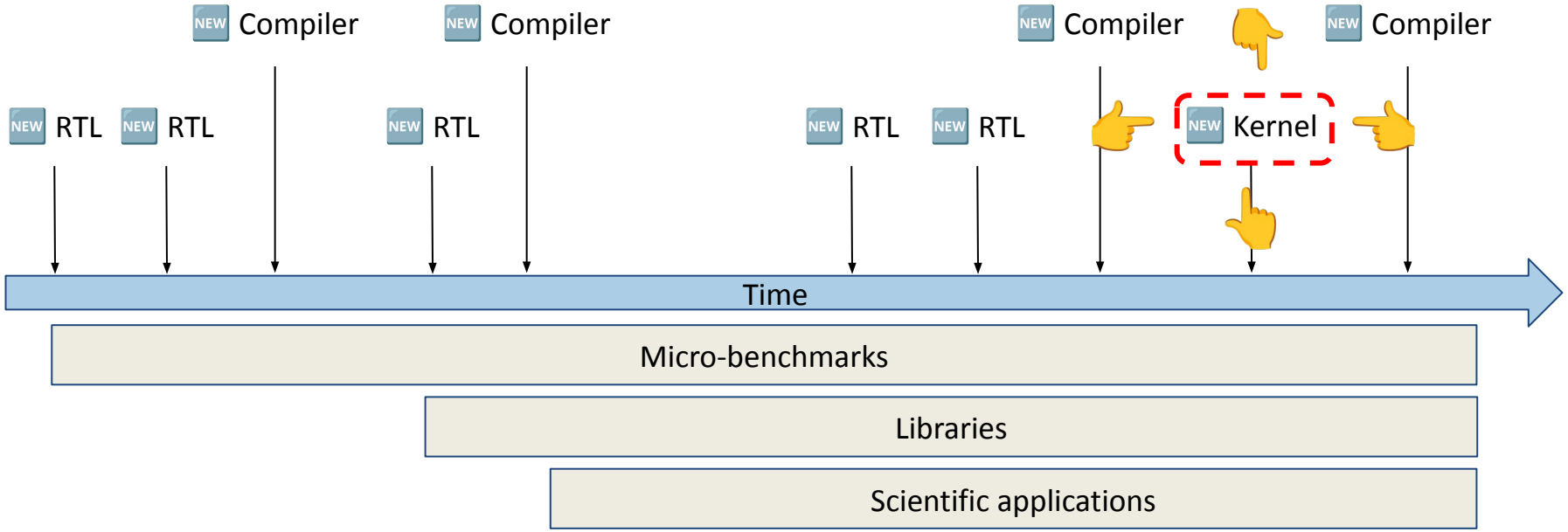
- HPC cluster experience
  - Job allocation via SLURM
  - Access via SSH
  - Disk partitions mounted via NFS

Motivation

Ecosystem that evolves through time



Ecosystem that evolves through time



The one instruction that we need

- All our micro-benchmarks measured cycles and instructions with raw ASM

```
asm volatile("rdcycle" /* ... */);
```

Kernel 5.7

- Old and reliable
- Basis for our benchmarking methodology
- System “works”, but requires multiple kernel patches

Kernel 6.10

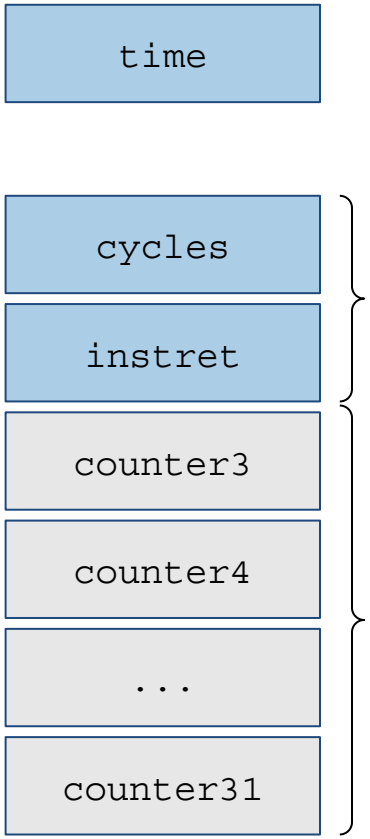
- New kernel in town
- It includes performance improvements :)
- Requires less patches to maintain

- Triggers **Illegal Instruction**

A prime on Hardware Performance Monitor (HPM)

The Divine Comedy

Counters



Real-time counter exposed as a memory-mapped register

Two fixed counters

Up to 29 configurable counters (implementation defined)

Mandatory

Optional

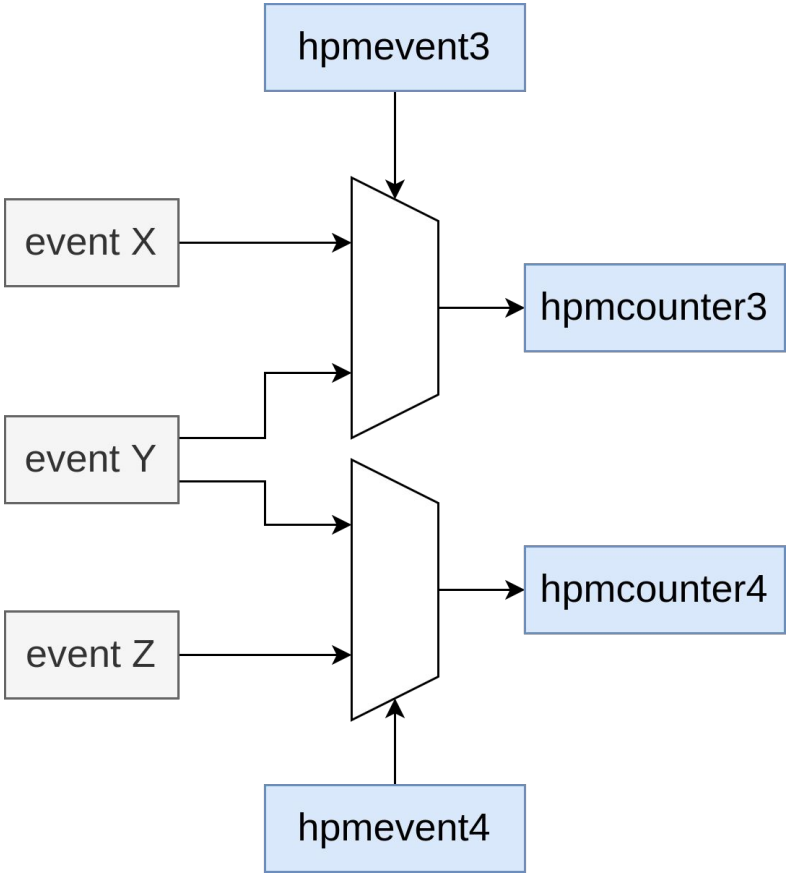
Event selectors

hpmevent

- Event selector
- Implementation defined meaning

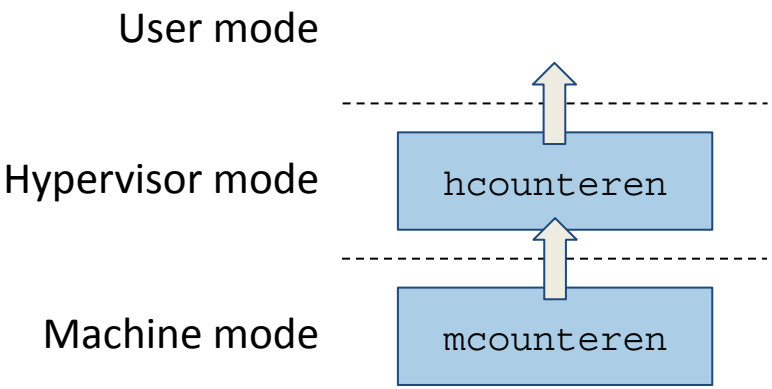
hpmcounter

- Counter registers
- Contains value (occurrences) of the selected event



Privilege levels

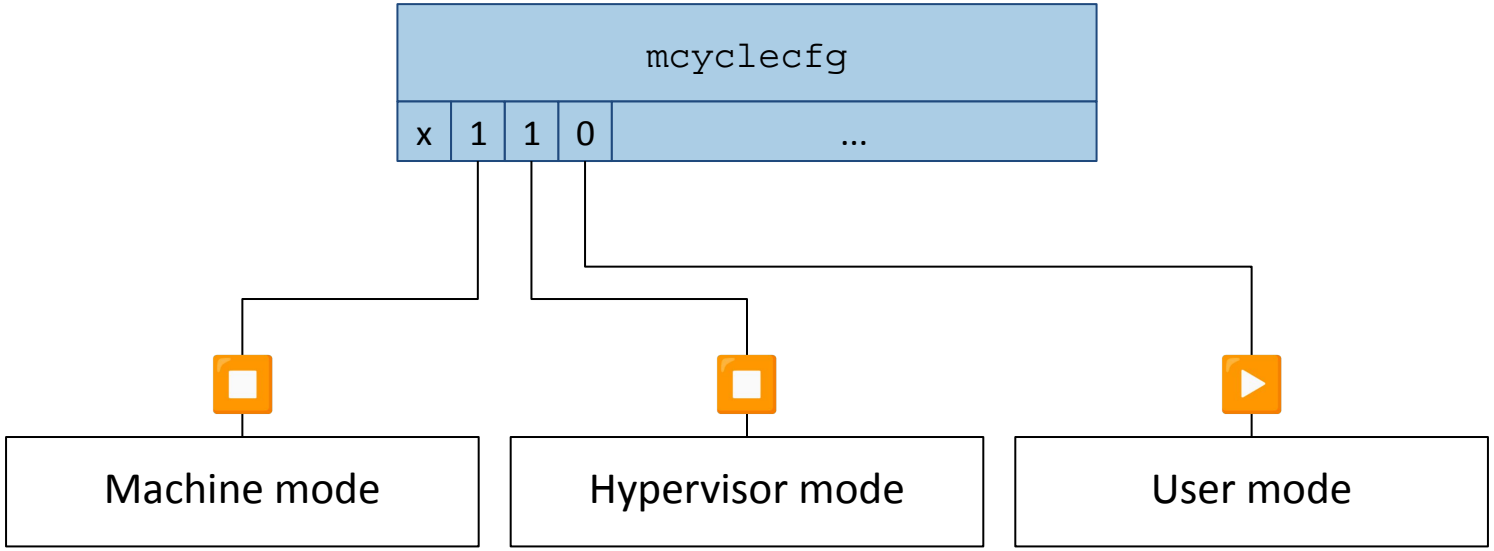
- Access to hardware counters is restricted depending on execution “mode”
- `[m,h,s]counteren` control access to next privilege level (or mode)
- Trying to read a counter from a level without enough permissions triggers an **Illegal Instruction**



Mode filtering

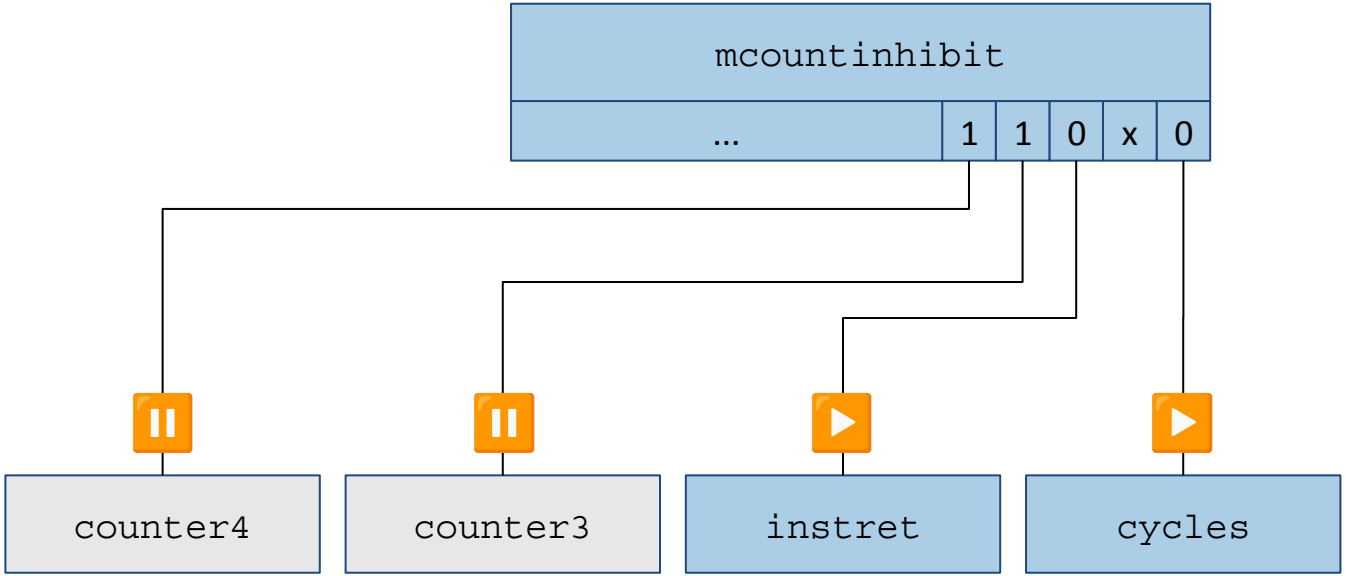
Smcntrpmf

- Filter events based on execution mode
- One register to control cycles and one to control instructions



Counter inhibit

- `mcountinhibit` pauses counters
- Each bit corresponds to one counter
- Only accessible from machine mode



Code to read hardware counters

Raw assembly

```
uint64_t start_instr, end_instr;

asm volatile("csrr %[dst], instret\n" : [dst]"=r"(start_instr));
for(int i=0; i<N; ++i){
  //Your computation
}
asm volatile("csrr %[dst], instret\n" : [dst]"=r"(end_instr));

int instructions = end_instr - start_instr;
```

Perf events

```
struct perf_event_attr pe;
int fd;
long long instructions;

memset(&pe, 0, sizeof(pe));
// Configuration of pe struct
// You really do not want to know the details…
pe.config = PERF_COUNT_HW_INSTRUCTIONS;

fd = perf_event_open(&pe, 0, -1, -1, 0);

ioctl(fd, PERF_EVENT_IOC_RESET, 0);
ioctl(fd, PERF_EVENT_IOC_ENABLE, 0);
for(int i=0; i<N; ++i){
  //Your computation
}
ioctl(fd, PERF_EVENT_IOC_DISABLE, 0);
read(fd, &instructions, sizeof(instructions));
```

PAPI

```
int eventset = -1;
long long instructions;

PAPI_create_eventset(&eventset)
PAPI_add_named_event(eventset, "PAPI_TOT_INS");

PAPI_start(eventset);
for(int i=0; i<N; ++i){
  //Your computation
}
PAPI_stop(eventset, &instructions);
```

Comparing methods

Raw assembly

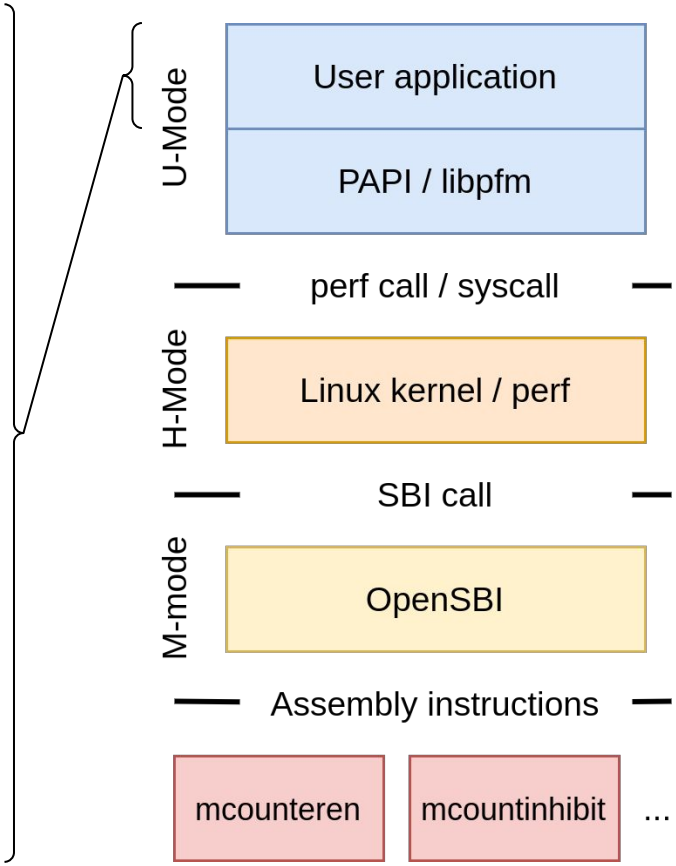
- Simplest code / Lowest overhead
- Architecture dependent (not portable)

perf

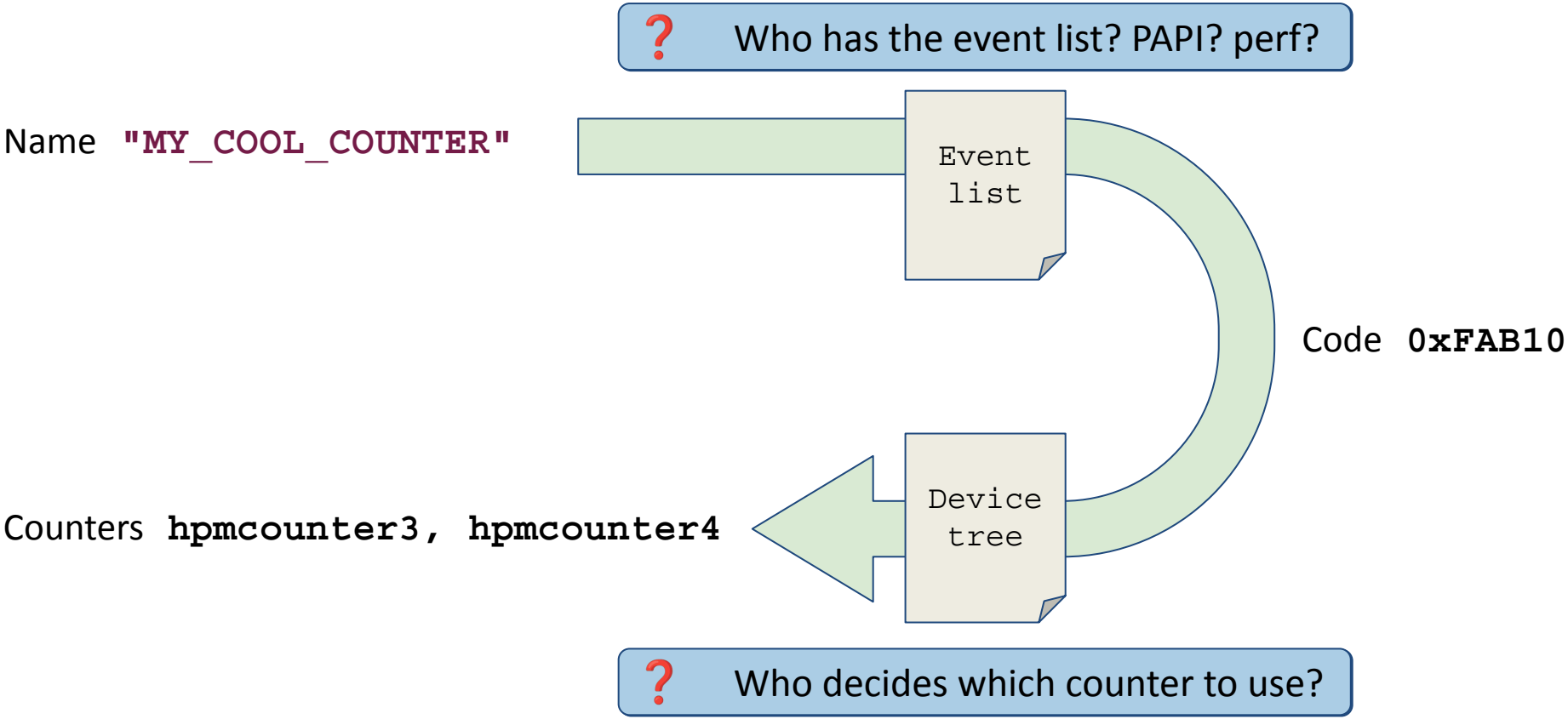
- Least user-friendly API
- Included in the Linux kernel (theoretically portable)

PAPI

- High-level abstraction / Highest overhead
- Most portability



Translation of event names



Going low-level

Raw assembly

The roadblock

- To read counters with raw assembly from U-mode, `[m,h,s]counteren` must be configured
- Linux perf driver relies on controlling `[m,h,s]counteren`

```
static void riscv_pmu_update_counter_access(void *info) {
  if (sysctl_perf_user_access == SYSCTL_LEGACY)
    csr_write(CSR_SCOUNTEREN, 0x7);
  else
    csr_write(CSR_SCOUNTEREN, 0x2);
}
```

A possible solution

- Patch the kernel to allow two modes of operation: `perf` and user access

```
static void riscv_pmu_update_counter_access(void *info) {
  if (sysctl_perf_user_access == SYSCTL_LEGACY) {
    csr_write(CSR_SCOUNTEREN, 0x7);
    sbi_ecall(SBI_EXT_PMU, SBI_EXT_PMU_COUNTER_START, /* ... */);
  } else {
    csr_write(CSR_SCOUNTEREN, 0x2);
    sbi_ecall(SBI_EXT_PMU, SBI_EXT_PMU_COUNTER_STOP, /* ... */);
  }
}
```

Assessment

Benefits

- Users have direct access to hardware counters
- Lowest possible overhead

Limitations

- Requires patching and recompiling the whole kernel
- Introduces a mode of operation that breaks `perf`
- There is no mechanism to configure event selectors from U-mode!

Possible alternative (future work)

- Implement functionality as a kernel module
- Similar work done for Arm: https://github.com/jerinjacobk/armv8_pmu_cycle_counter_el0/tree/master

Going high-level

PAPI library

The roadblock

- The software layer that PAPI relies on (libpfm4) does not recognize our hardware
- There's no definition of the supported events
  - EPAC
  - Milk-V Pioneer
  - BananaPi

Previous work

- Ground work to integrate all the pieces together (OpenSBI, perf_events, libpfm, etc.)
- PAPI support for the HiFive Unmatched

Supporting RISC-V Performance Counters Through Linux Performance Analysis Tools
https://ieeexplore.ieee.org/document/10265733

~~A possible~~ The solution

- Add functionality to recognize target platforms
- Go through the documentation and add the event listings for each system

😊 How hard could it be?

The guessing game!

```
$ cat /proc/cpuinfo
processor : 0
hart      : 4
isa       : rv64imafdc
mmu       : sv39
uarch     : sifive,u74-mc
```

```
$ cat /proc/cpuinfo
processor : 0
hart      : 2
isa       : rv64imafdcv
mmu       : sv39
mvendorid : 0x5b7
marchid   : 0x0
mimpid    : 0x0
```

```
$ cat /proc/cpuinfo
processor : 0
hart      : 0
isa       : rv64imafdcv
mmu       : sv39
uarch     : epi,avispado
```

```
$ cat /proc/cpuinfo
processor  : 0
hart       : 0
model name : Spacemit(R) X60
isa        : rv64imafdcv_sscofpmf_sstc_svpbmt_zicbom_zicboz_zicbop_zihintpause
mmu        : sv39
mvendorid  : 0x710
marchid    : 0x8000000058000001
mimpid     : 0x1000000049772200
```

The bookworm game!

- Going through documentation provided by the vendors (or community…)
- Need to define the list of events and to which counters can they be mapped to

```
static const riscv_entry_t riscv_epi_epac_avispado_pe[] = {
  // Vector Unit
  {.name = "VPU_COMPLETED_INST",
   .code = 0x020,
   .desc = "Number of finished instructions at the VPU" },
  {.name = "VPU_ISSUED_INST",
   .code = 0x040,
   .desc = "Number of issued instructions at the VPU" },
  // ...
}
```

Event codes

Mapped to counters 3 & 4

How do we tell the software?

Device tree

Assessment

Benefits

- Users have a portable way to read hardware counters
- Enables other performance analysis tools at BSC (eg. Extrae)

Limitations

- Multiple software layers before getting to the hardware → Overhead?
- Relying on event description provided by vendor
- No standard `/proc/cpuinfo` output

Ongoing work

- Sorting out implementation bugs
- Code available in github: https://github.com/hpc-ulisboa/RISC-V-PAPI

Comparing methods

Methodology

- Small kernel with exactly 17 instructions
- Only reading instret counter

- Increasing `iterations` parameter
- Comparing expected and measured

```
asm volatile(
  "1: \n"

  "add  %[o], %[o], %[i] \n"
  /* ... */
  "add  %[o], %[o], %[i] \n"

  "addi %[it], %[it], -1 \n"
  "bnez %[it], 1b \n"
  : [o]"+r"(a)
  : [it]"r"(iterations), [i]"r"(b)
);
```

Results

Low overhead

High overhead

[UPDATE!] Measured - Expected		Machine			
Method	Exp. Instructions	Banana Pi	EPAC	Pioneer	Unmatched
csrr	1,80E+01		6		5
	1,80E+02		6		6
	1,80E+03		6		6
	1,80E+04		6		6
	1,80E+05		15974		6
	1,80E+06		67850		6
	1,80E+07		619599		30961
	1,80E+08		6031365		268276
	1,80E+09		61171912		2723433
perf	1,80E+01	2719	2321	2613	1784
	1,80E+02	2719	2321	2613	1784
	1,80E+03	2719	2321	2613	1784
	1,80E+04	2719	2321	2613	1784
	1,80E+05	2719	11514	2613	1784
	1,80E+06	2719	71820	2613	1784
	1,80E+07	15176	594111	13387	27330
	1,80E+08	178615	5992007	164422	273632
	1,80E+09	1551683	60269205	1220320	2724128
PAPI	1,80E+01	4126	3895	3899	4051
	1,80E+02	4126	3895	3899	4051
	1,80E+03	4126	3895	3899	4051
	1,80E+04	4126	3895	3899	4051
	1,80E+05	4126	12353	3899	4051
	1,80E+06	4126	67234	3899	4051
	1,80E+07	24225	602994	14845	24615
	1,80E+08	207973	6070970	169415	276478
	1,80E+09	1662211	61283426	1237343	2565450

Results - Comparing methods

- Raw ASM (csrr) has lowest overhead
- Both perf and PAPI considerable overhead (1.7k and 4k)

- All methods skyrocket when measured region exceeds 18M instructions
  - Currently investigating
  - Special case to handle overflow?
  - OS preemption (hypervisor) being counted?

[UPDATE!] Measured - Expected		
Method	*Exp. Instructions*	Unmatched
csrr	1,80E+01	5
	1,80E+02	6
	1,80E+03	6
	1,80E+04	6
	1,80E+05	6
	1,80E+06	6
	1,80E+07	30961
	1,80E+08	268276
	1,80E+09	2723433
perf	1,80E+01	1784
	1,80E+02	1784
	1,80E+03	1784
	1,80E+04	1784
	1,80E+05	1784
	1,80E+06	1784
	1,80E+07	27330
	1,80E+08	273632
	1,80E+09	2724128
PAPI	1,80E+01	4051
	1,80E+02	4051
	1,80E+03	4051
	1,80E+04	4051
	1,80E+05	4051
	1,80E+06	4051
	1,80E+07	24615
	1,80E+08	276478
	1,80E+09	2565450

Results - Mode filtering

- BananaPi implements `Smcntrpmf` (mode filtering)
- Still observing increase in overhead with more than 18M instructions

- Software is not leveraging mode filtering!?

- Currently improving software to acknowledge filtering when available

[UPDATE!] Measured - Expected		*Machine*
Method	*Exp. Instructions*	Banana Pi
csrr	1,80E+01	
	1,80E+02	
	1,80E+03	
	1,80E+04	
	1,80E+05	
	1,80E+06	
	1,80E+07	
	1,80E+08	
	1,80E+09	
perf	1,80E+01	2719
	1,80E+02	2719
	1,80E+03	2719
	1,80E+04	2719
	1,80E+05	2719
	1,80E+06	2719
	1,80E+07	15176
	1,80E+08	178615
	1,80E+09	1551683
PAPI	1,80E+01	4126
	1,80E+02	4126
	1,80E+03	4126
	1,80E+04	4126
	1,80E+05	4126
	1,80E+06	4126
	1,80E+07	24225
	1,80E+08	207973
	1,80E+09	1662211

Results - Comparing with other architectures

- Comparing against Intel Sapphire Rapids CPU
- Dramatically less overhead compared to implementation on RISC-V
- Currently investigating origin of overhead
  - Translation layers for events
  - Mode filtering
  - Other implementation tricks?

[UPDATE!] Measured - Expected			
Method	Exp. Instructions	Unmatched	x86
csrr	1,80E+01	5	
	1,80E+02	6	
	1,80E+03	6	
	1,80E+04	6	
	1,80E+05	6	
	1,80E+06	6	
	1,80E+07	30961	
	1,80E+08	268276	
	1,80E+09	2723433	
perf	1,80E+01	1784	12
	1,80E+02	1784	12
	1,80E+03	1784	12
	1,80E+04	1784	12
	1,80E+05	1784	12
	1,80E+06	1784	12
	1,80E+07	27330	17
	1,80E+08	273632	65
	1,80E+09	2724128	546
PAPI	1,80E+01	4051	806
	1,80E+02	4051	806
	1,80E+03	4051	806
	1,80E+04	4051	806
	1,80E+05	4051	806
	1,80E+06	4051	806
	1,80E+07	24615	811
	1,80E+08	276478	860
	1,80E+09	2565450	1343

Summary of current status

- Reading counters with ASM (CSR read) is our preferred way...
  - But discouraged by spec
  - There are security reasons
  - Missed opportunity of giving options to the user

- Reading counters with PAPI enables analysis of complex codes
  - Has a non-negligible overhead
  - Lots of pieces to put together!

Legend
✔ - Done
🛠 - Work in progress
😴 - Pending
❓ - More research to be done
❌ - Blocked

			CSR read			perf			PAPI		
	DTB with PMU	Kernel with perf_user_access	Read cycles	Read instret	Read hpmcounter	Read cycles	Read instret	Read hpmcounter	Read cycles	Read instret	Read hpmcounter
Arriesgado	✔	✔	✔	✔	❓	✔	✔	✔	✔	✔	✔
Pioneer	✔	🛠	❌	❌	❌/❓	✔	✔	✔	✔	✔	✔
BananaPi	✔	😴	❌	❌	❌/❓	✔	✔	✔	✔	✔	✔
EPAC1.5	😴	✔	✔	✔	❓	✔	✔	❌	✔	✔	❌

Conclusions

Conclusions

- Software Development Vehicles
  - Ecosystem with hardware, software, and methodology
  - Commercial RISC-V boards and EPAC prototype

- High-Performance Monitor specification
  - Counters , event selectors, privilege levels, etc.
  - Aligning with upstream made us realize we had to refine measurement methodology

- Low-level performance monitoring
  - Simplest way to implement and lowest overhead
  - Discourage for security reason
  - Missed opportunity for research and performance analysis

- High-level performance monitoring
  - Most portability and highest overhead

https://github.com/hpc-ulisboa/RISC-V-PAPI

RISC-V in HPC
A look into tools for performance monitoring

ISC25, Hamburg

Fabio Banchelli * (BSC)

Rafel Albert Bros Esqueu (BSC)

Tiago Rocha (INESC-ID IST)

Nuno Roma (INESC-ID IST)

Pedro Tomás (INESC-ID IST)

Nuno Neves (INESC-ID IST)

Filippo Mantovani (BSC)

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

European Processor Initiative

End