DSA Monitoring Framework for HW/SW Partitioning of Application Kernels leveraging VPs

Christoph Hazott, Daniel Große
Institute for Complex Systems (ICS)
Web: jku.at/ics
Email: christoph.hazott@jku.at

JOHANNES KEPLER UNIVERSITY LINZ

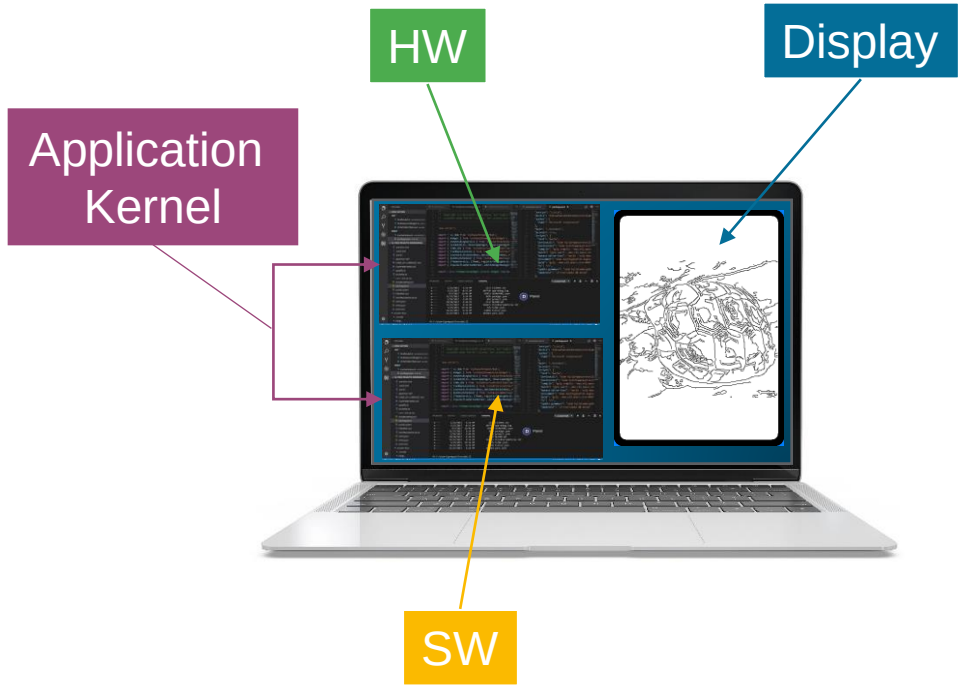
Agenda

- Motivation, Domain Specific Architectures & Background on VPs
- Ingredients of Proposed approach
- Solution:
  - Host-to-SW Memory Hierarchy
  - Proposed Framework
- Experiments
- Conclusion

Motivation and DSAs

- Moore's Law
  - Still driving electronic industry
  - Permanent innovation on all levels necessary to keep speed

- Domain Specific Architectures (DSAs)
  - Fall into class of heterogeneous architectures
  - Integrate specific HW accelerators extracted from SW to meet system performance requirements
  - HW/SW partitioning through application kernels
    - Hotspot functions invoked frequently in loops



Background: Virtual Prototypes

A **Virtual Prototype** (VP) is an executable SW model of a HW system that runs on a host computer.

A VP is binary compatible to the physical HW.

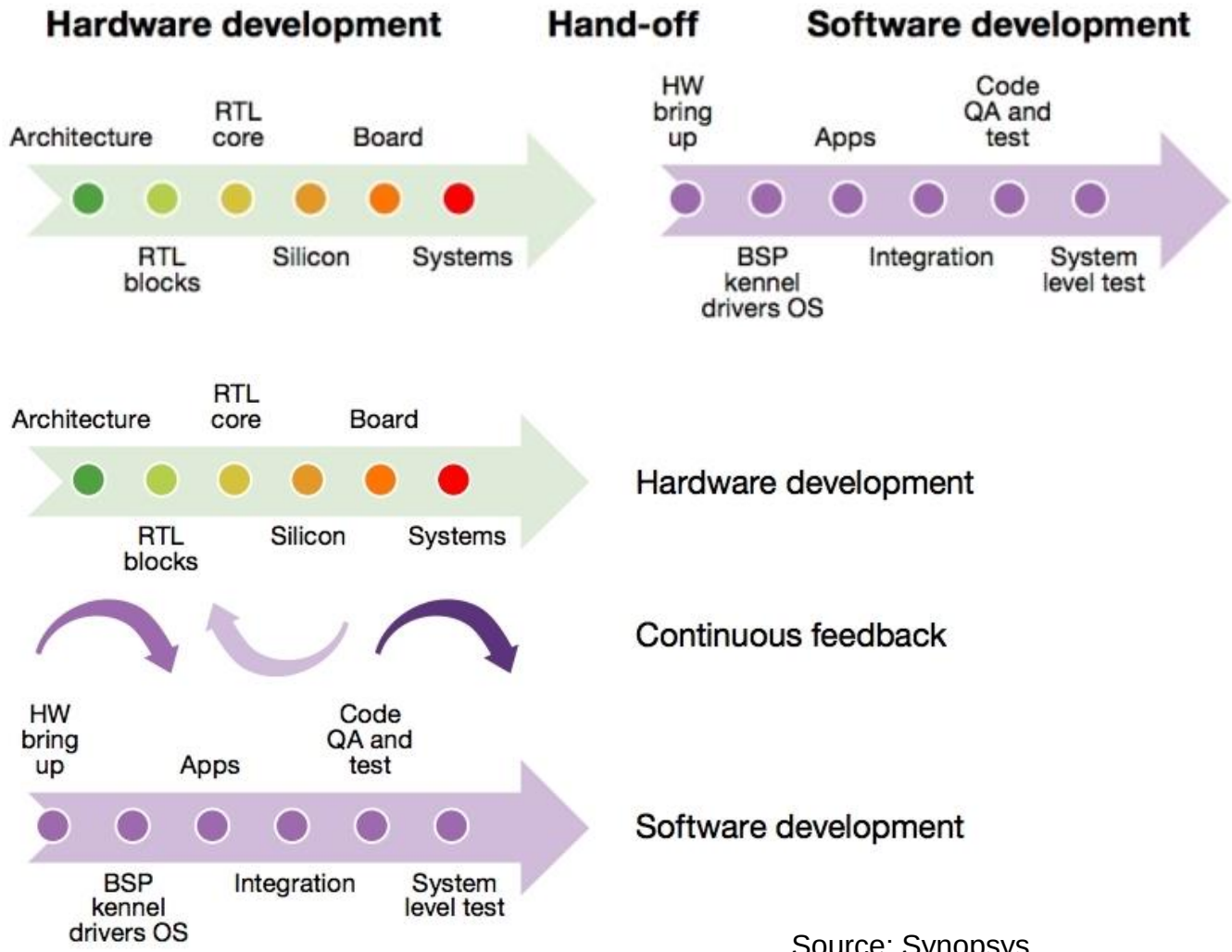
- Industrial-proven, widely used by semiconductor global players
- VPs are modeled in SystemC
  - C++ class library
  - IEEE1666-2011 Standard
  - Transaction Level Modeling (TLM) for abstraction

SYSTEMC™

accellera SYSTEMS INITIATIVE™

Background: Why Virtual Prototypes?

Traditional Development Flow



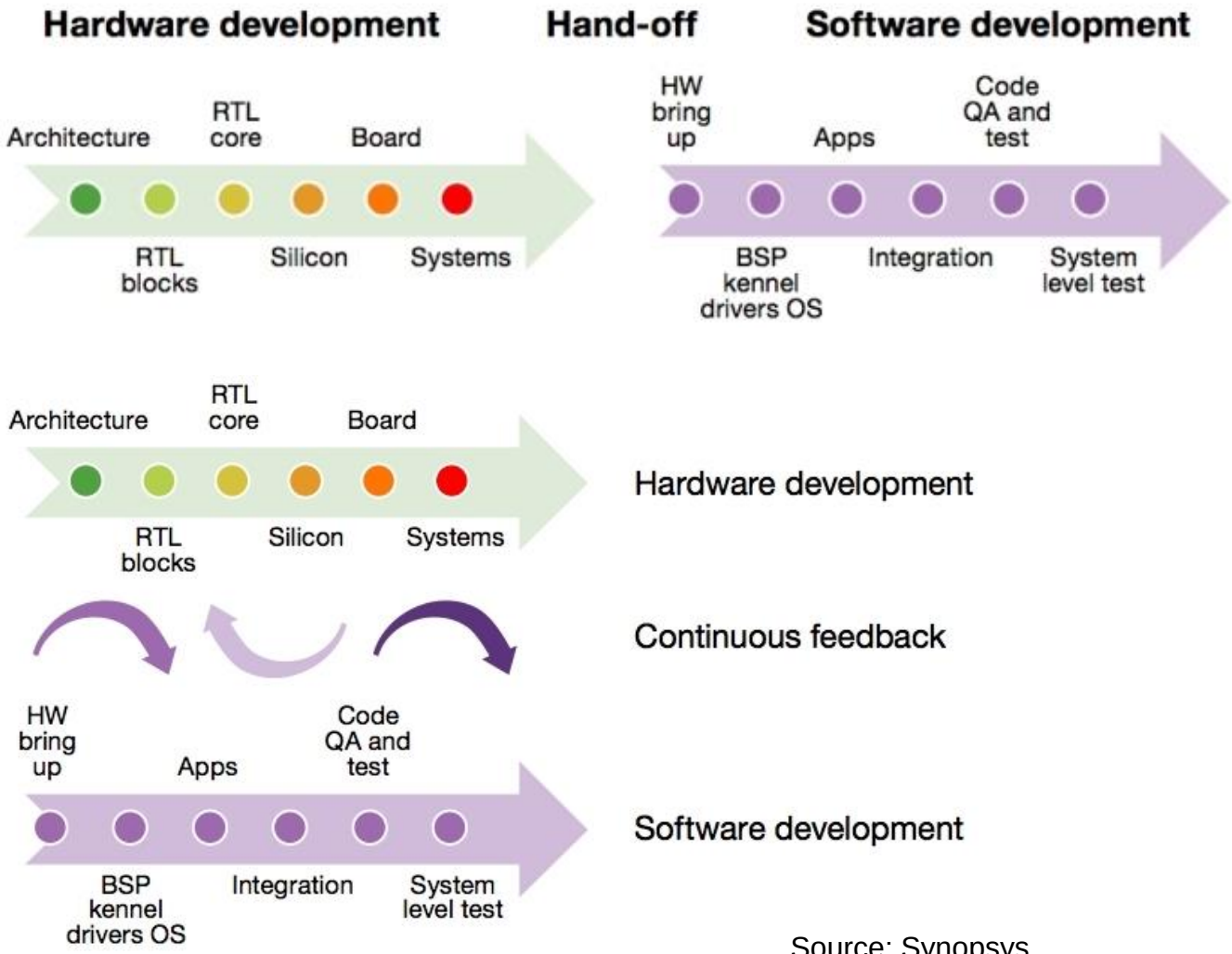
Parallel HW and SW Development based on VP

Source: Synopsys

Profiling with Virtual Prototypes?

- Classical Profiling
  - HW and SW separately
  - Joined in later processing of result
  - Changes to source code or binary

- Proposed Approach
  - Taking external perspective that encompasses HW and SW, facilitating a holistic view as unified system



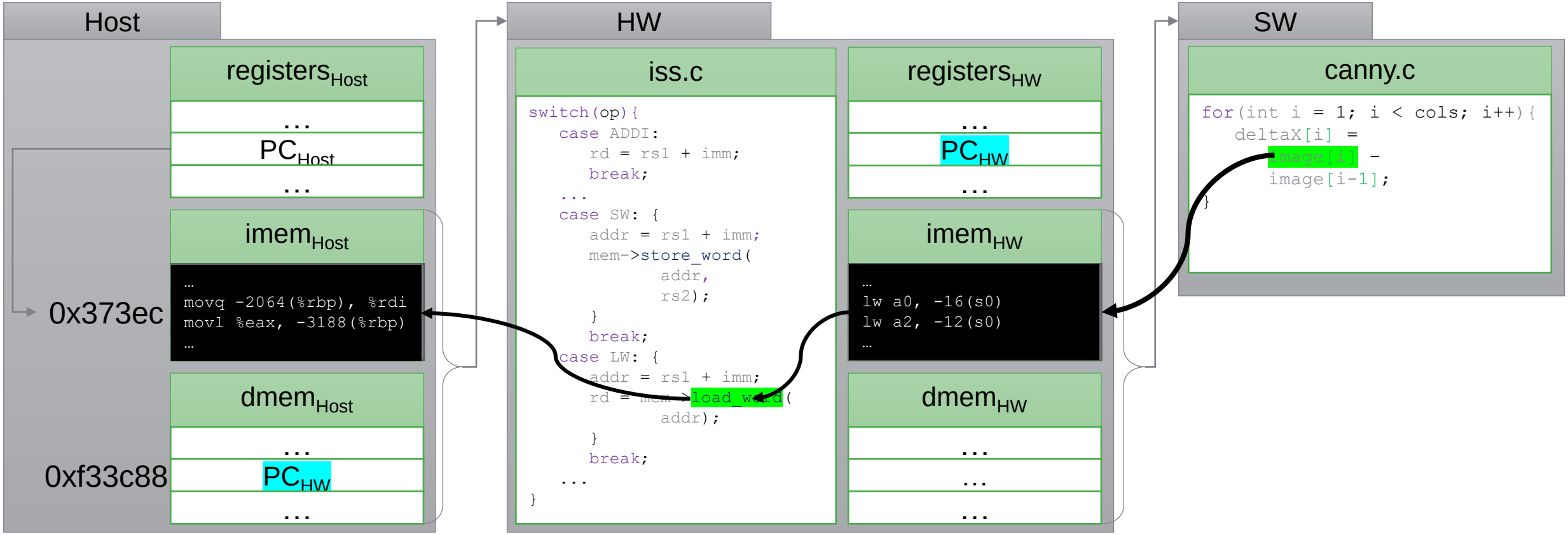
Source: Synopsys

Ingredients of Proposed Approach

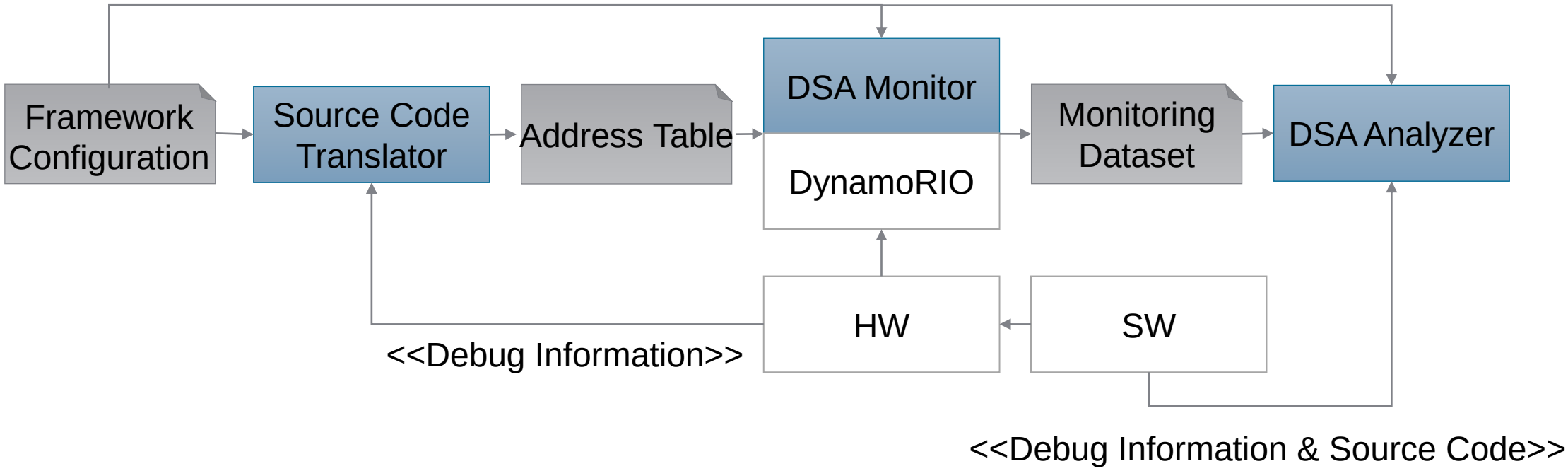
- Framework leveraging observability of VPs
- Monitoring via runtime code manipulation of VP binary running on host
  - Maintain high simulation performance
- Specialized monitors for SW kernels
  - Low data amount

- **Requirements**:
  - No modification of SW running on VP
  - No modification of VP source
- **Idea**:
  - Understand Host-to-SW memory hierarchy

Solution: Host-to-SW Memory Hierarchy



Proposed Framework



Source Code Translator

- From configuration

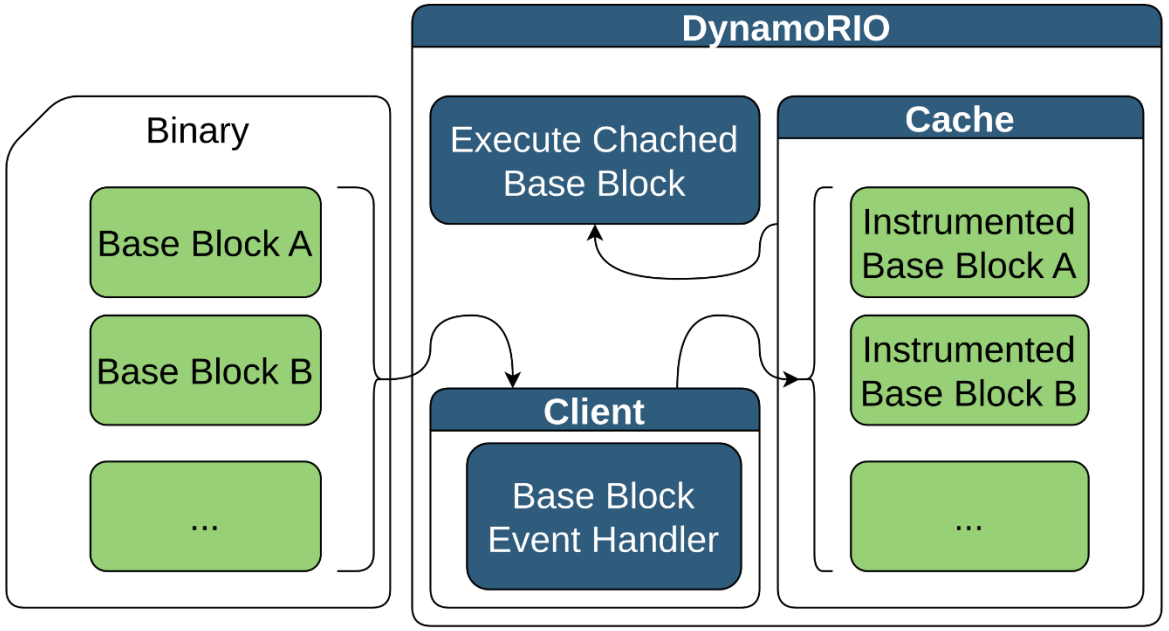
```
1 HW:'riscv_vp'
2 ...
3 PC_HW:'PC_VP'
4 ...
5 MEM_READ_HW:'/riscv_vp/memory.h:76'
6 ...
```

- To address table

```
1 PC_HW:'0xf33c88'
2 ...
3 MEM_READ_HW:'0x373ec'
4 ...
```

DynamoRIO

- DynamoRIO as runtime code manipulation system
  - Supports code transformation while executing
  - Exports interface for building additional tools
  - Powerful instruction manipulation library
- Design Goals
  - Efficient
    - Near-native performance
  - Transparent
    - Match native behavior
  - Comprehensive
    - Control every instruction, in any application
  - Customizable
    - Adapt to satisfy disparate tool needs



Src: https://github.com/DynamoRIO/dynamorio/releases/download/release_6_1_0/DynamoRIO-tutorial-mar2016.pdf

DSA Monitor

- Instrumenting for monitoring PC_{HW}

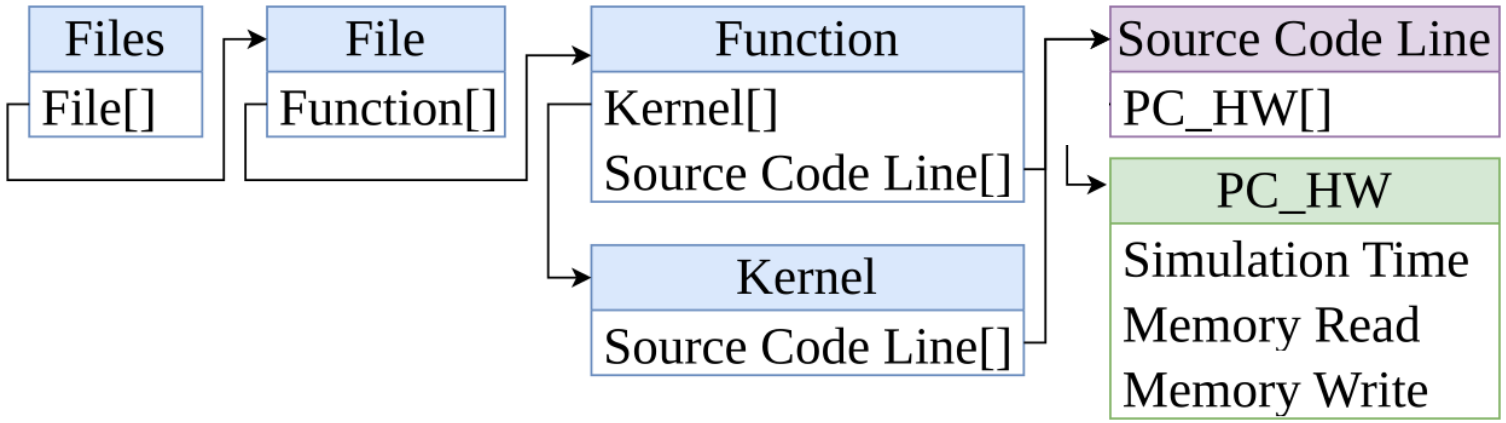
```
1  if(instr_writes_memory(instr)){
2      addr = opnd_get_addr(
3              instr_get_dst(instruction, i)
4              );
5
6      if(addr == pc_hw) {
7          dr_insert_clean_call(
8              ..., clean_call_pc_hw, ...);
9      }
10 }
```

- Instrumenting for monitoring HW memory access

```
1  pc_host = instr_get_app_pc(instr);
2  if (monitor_pc_host[pc_host]) {
3      dr_insert_clean_call(...,
           clean_call_hw_mem_read, ...);
4  }
```

DSA Analyzer

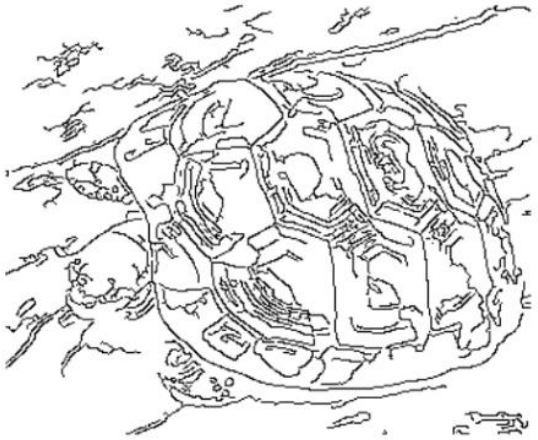
- Data structure containing monitoring results used for analysis



Experiments: Canny Edge Detection

- SW: Canny Edge Detection
  - Gaussian **smoothing**
  - Computing **derivatives**
  - Computing **magnitude** of gradient
  - Performing non-maximal **suppression**
  - Applying **hysteresis**

(a) Original



(b) Result

SW Application Kernel: Excerpts from Canny SW

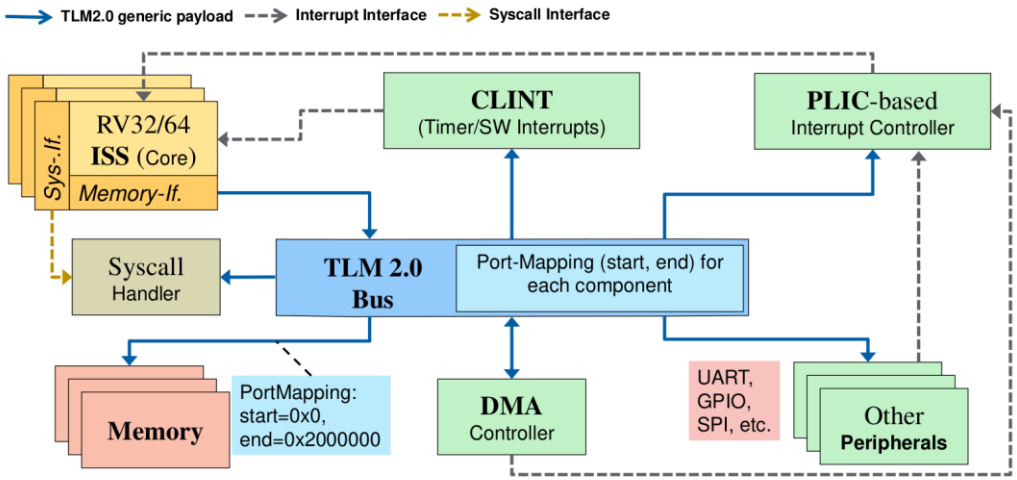
- Nested SW kernels implementing gaussian filter for smoothing image rows

```
1  for(c=0;c<cols;c++) {
2      for(r=0;r<rows;r++) {
3          ...
4          for(rr=(-center);rr<=center;rr++) {
5              row = r + rr;
6              if(row >= 0 && row < rows) {
7                  ...
8              }
9          }
10         smoothedim[r*cols+c] = ...
11     }
12 }
```

RISC-V

- RISC-V: Open and royalty-free ISA
- Focus on simplicity and modularity
- Base Integer Instruction Set
  - Mandatory
  - 32, 64 and 128 bit configurations
  - ~40 Instructions
- Extensions:
  - M .. Multiply/Divide
  - A .. Atomic
  - F, D, Q .. Floating Point (Single, Double, Quad)
  - C .. Compressed

RISC-V VP++



- Open source on GitHub
  - https://github.com/ics-jku/riscv-vp-plusplus
- Key features:
  - SystemC TLM-2.0
  - Bare metal configurations, including:
    - SiFive HiFive1 - FE310
    - GD32VF103VBT6 microcontroller (Nuclei N205) including UI
  - Linux RV32 and RV64, single and quad-core VPs (SiFive FE540)
  - Support for *RISC-V "V" Vector Extension* (RVV) version 1.0
  - Full integration of GUI-VP, which enables simulation of interactive graphical Linux applications
  - Based on RISC-V VP introduced in 2018*
- More information: http://www.systemc-verification.org

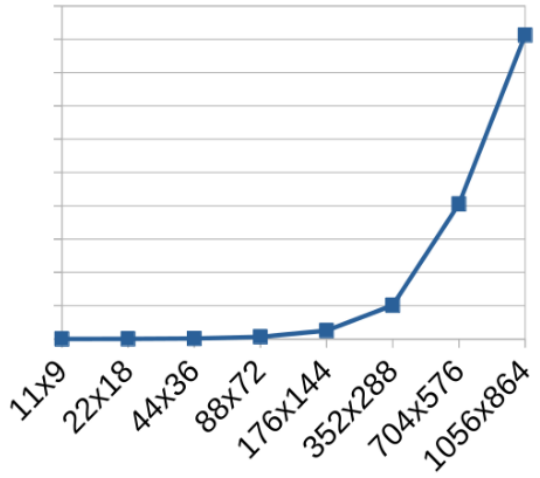
*Vladimir Herdt, Daniel Große, Hoang M. Le, and Rolf Drechsler. Extensible and configurable RISC-V based virtual prototype. In *FDL*, 2018.

Results

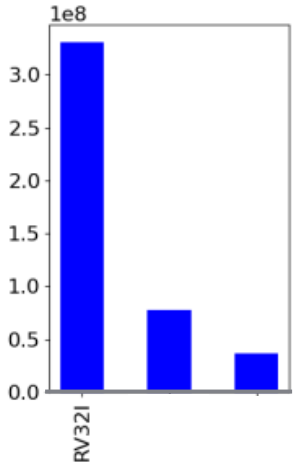
- Costs and Scalability of Monitoring

	11x9	22x18	44x36	88x72	176x144	352x288	704x576	1056x864
Executed RISC-V instructions	2,918,759	7,350,190	25,637,385	98,984,371	395,274,305	1,584,436,030	6,502,028,739	14,746,998,363
Host time - no monitoring [min]	0.01	0.03	0.09	0.32	1.28	4.97	20.66	51.84
Host time - monitoring [min]	0.04	0.06	0.17	0.60	2.39	9.20	38.02	93.34
Overhead factor	3.45	2.19	1.91	1.84	1.86	1.85	1.84	1.80
Size of monitoring dataset [MB]	34	85	294	1,229	4,608	18,432	74,752	168,960

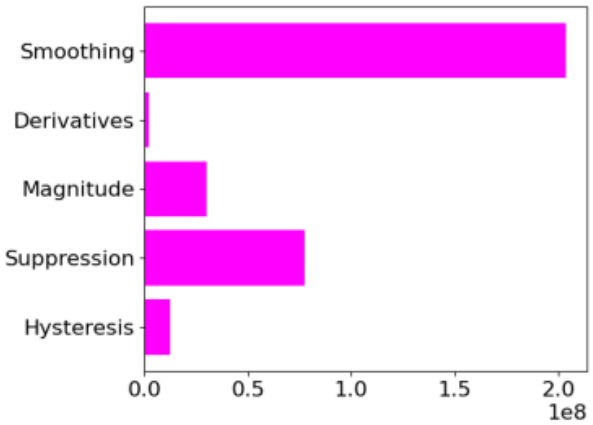
- Results scale according to the number of pixels
- Starting from a resolution of ~88x72px, overhead factor stabilizes at ~1.85
- Results stabilize at 88x72px



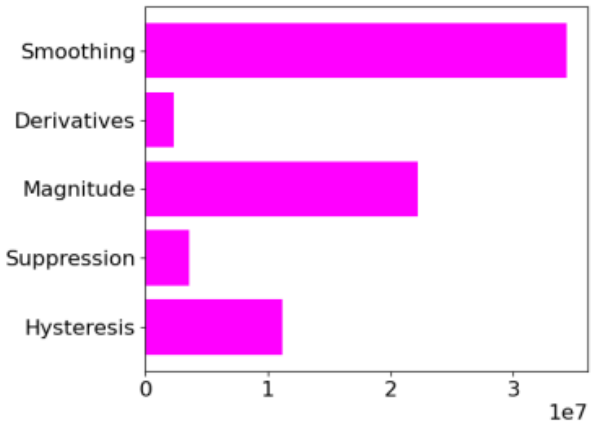
HW/SW Partitioning: Simulation Time



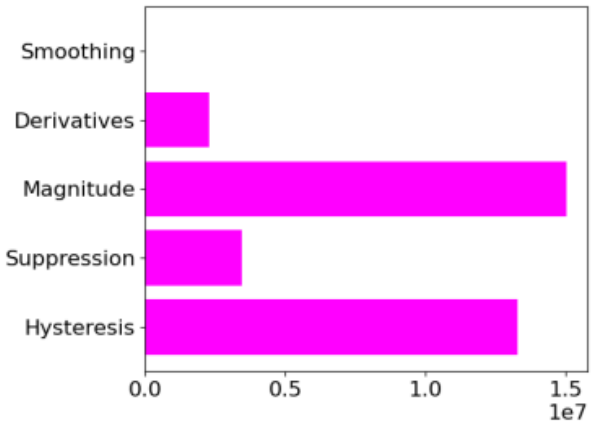
(a) Overall



(b) RV32I

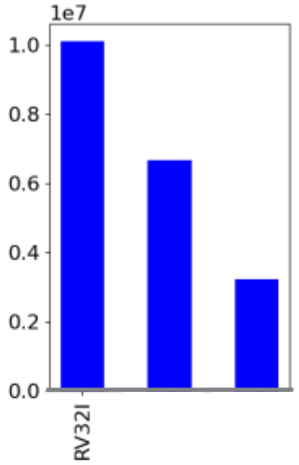


(c) +MAFC extensions

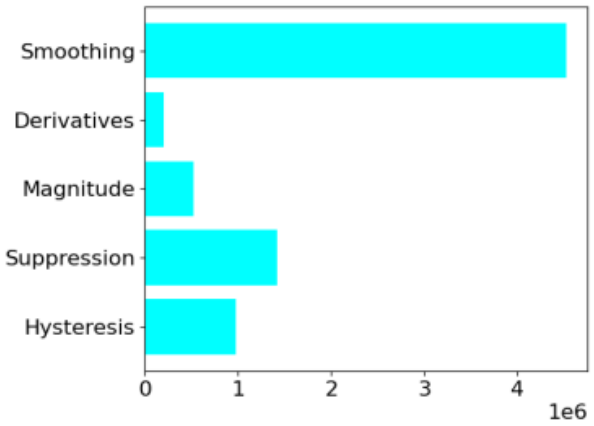


(d) +HW smoothing

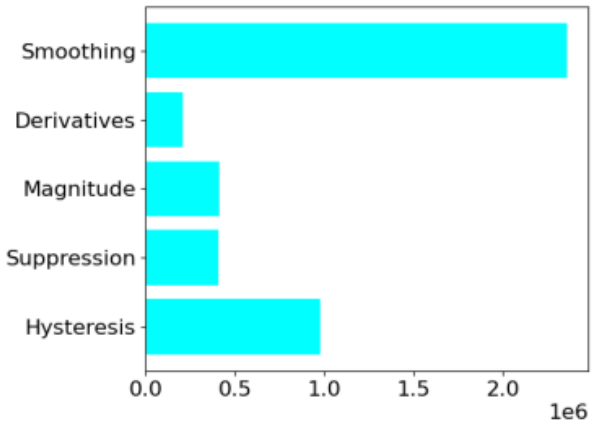
HW/SW Partitioning: Memory Accesses



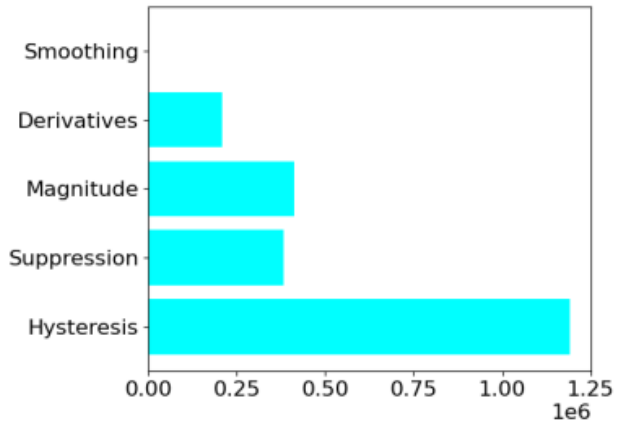
(a) Overall



(b) RV32I



(c) +MAFC extensions



(d) +HW smoothing

HW/SW Partitioning: Acceleration

- Performance improved by a factor of ~**8.67**

Final FPS results for canny

	RV32I	+MAFC	+HW
Kernel 0 [ms]	329.71	77.28	37.88
FPS	3	12	26

Conclusions

- Novel monitoring approach
  - Host-to-SW memory hierarchy
  - Leveraging dynamic binary instrumentation to insert monitors
  - Low simulation overhead
  - Low data amount

- Framework
  - User interaction via source code
  - Graphs for HW/SW interactions

DSA Monitoring Framework for HW/SW Partitioning of Application Kernels leveraging VPs

Christoph Hazott, Daniel Große
Institute for Complex Systems (ICS)
Web: jku.at/ics
Email: christoph.hazott@jku.at

JOHANNES KEPLER UNIVERSITY LINZ