KALRAY

Tracing massively parallel processor array

Tracing Summit 2013

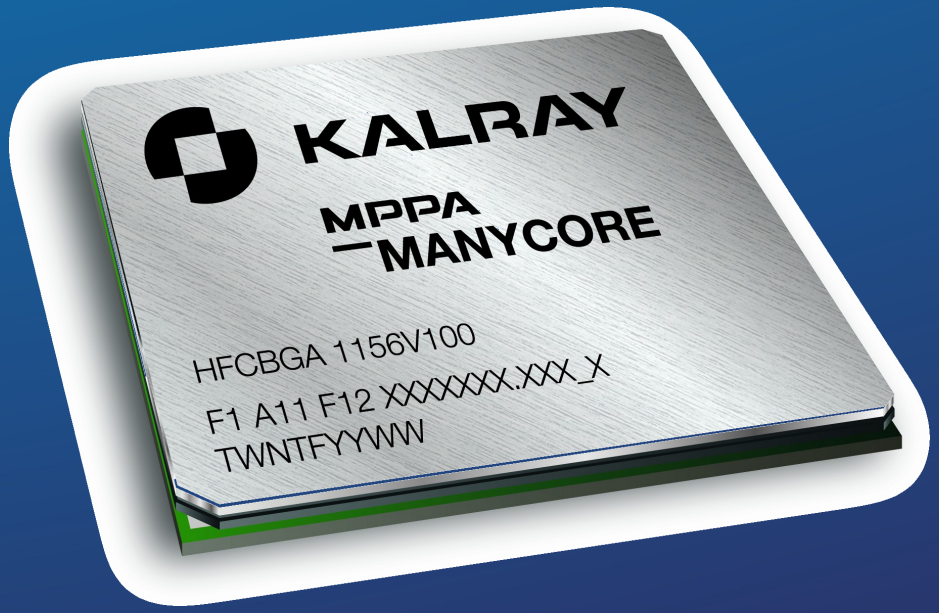
AGILE
PERFORMANCE

Agenda

- Hardware description
- Software trace system
- Trace visualization
- Use case #1
- Use case #2

Agenda

- **Hardware description**
- Software trace system
- Trace visualization
- Use case #1
- Use case #2

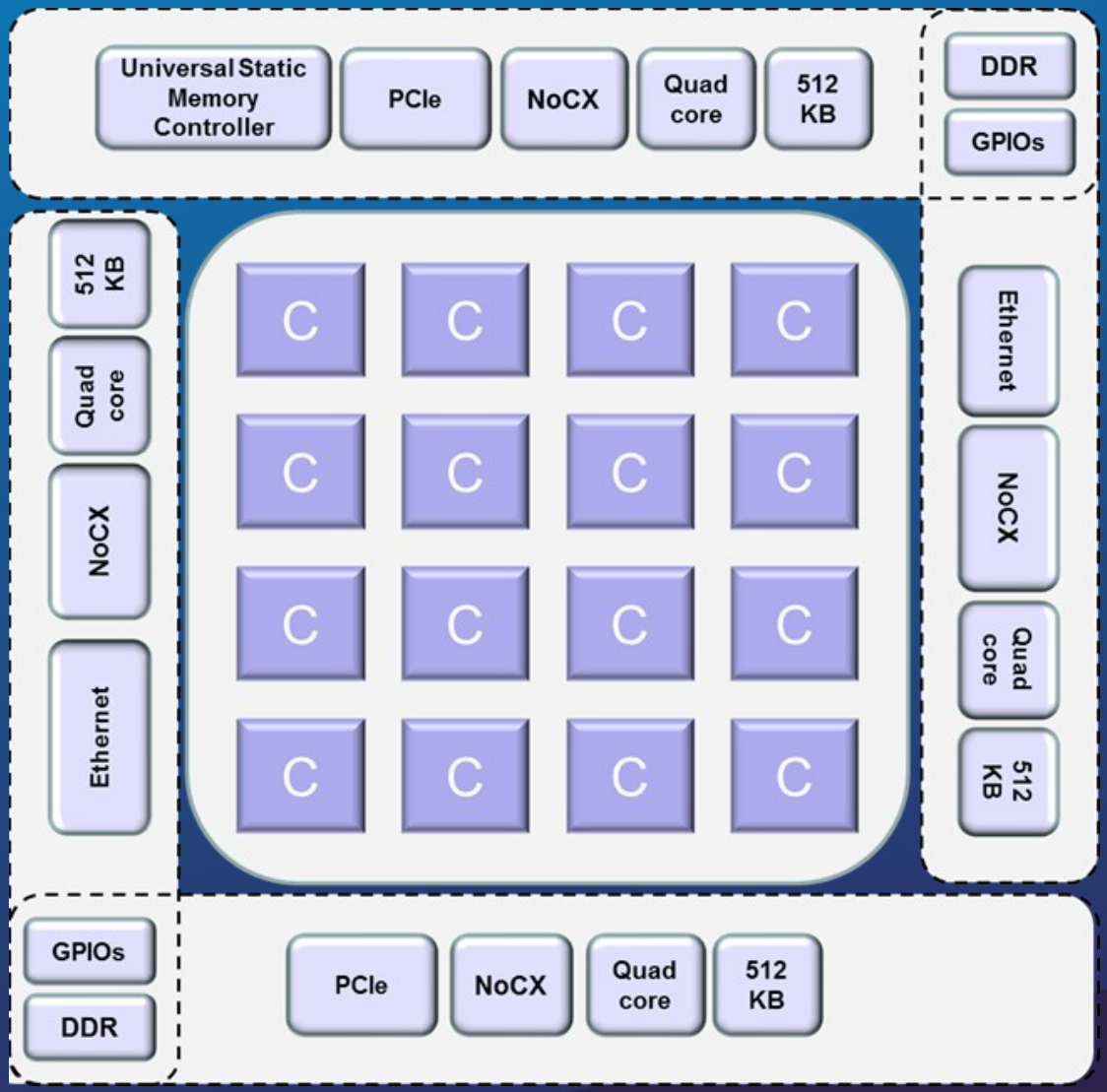


Hardware description: MPPA key features

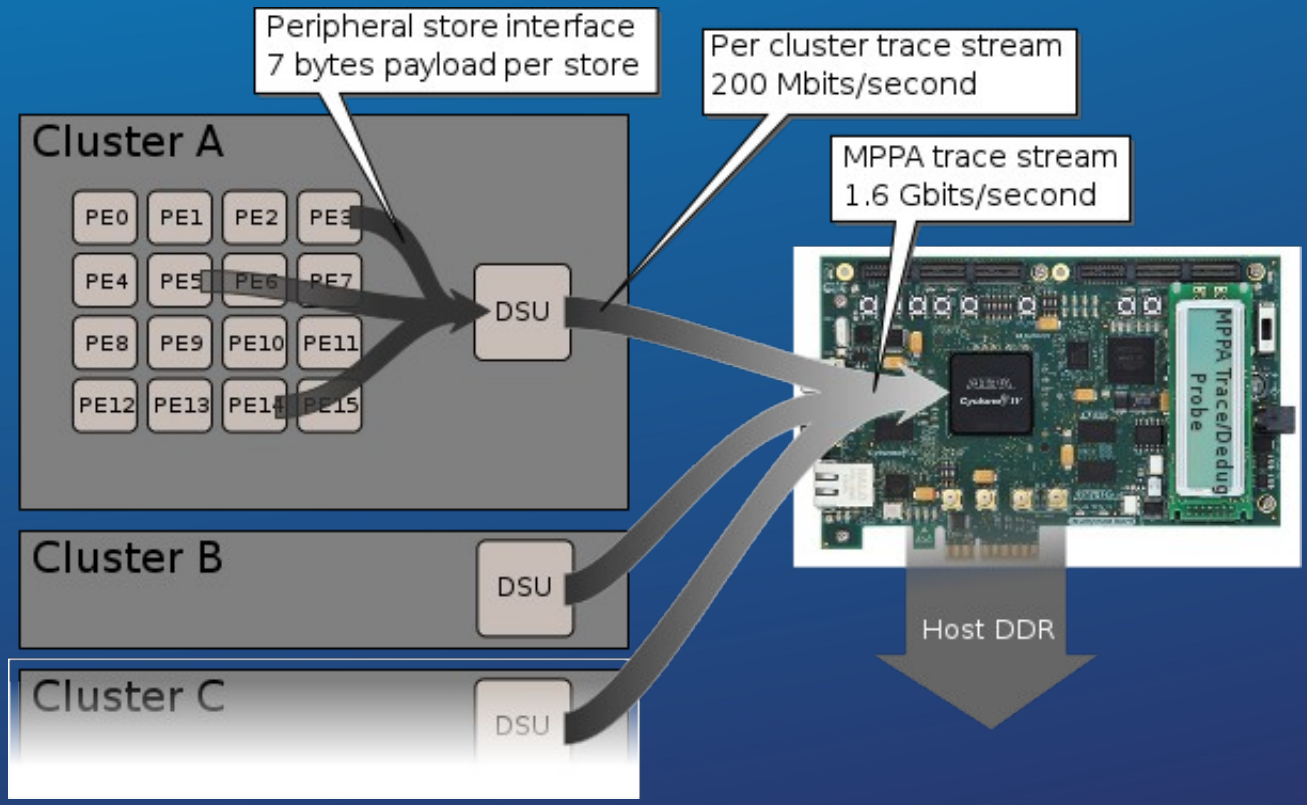
MPPA 256 (Multi Purpose Processor Array)

- 16 compute clusters of 16+1 cores
- 4 IO Clusters of 4 cores
- High bandwidth network on chip
- High speed standard interfaces
- Typical consumption 5W
- 28nm CMOS technology
- 230 GFLOPS @400MHz

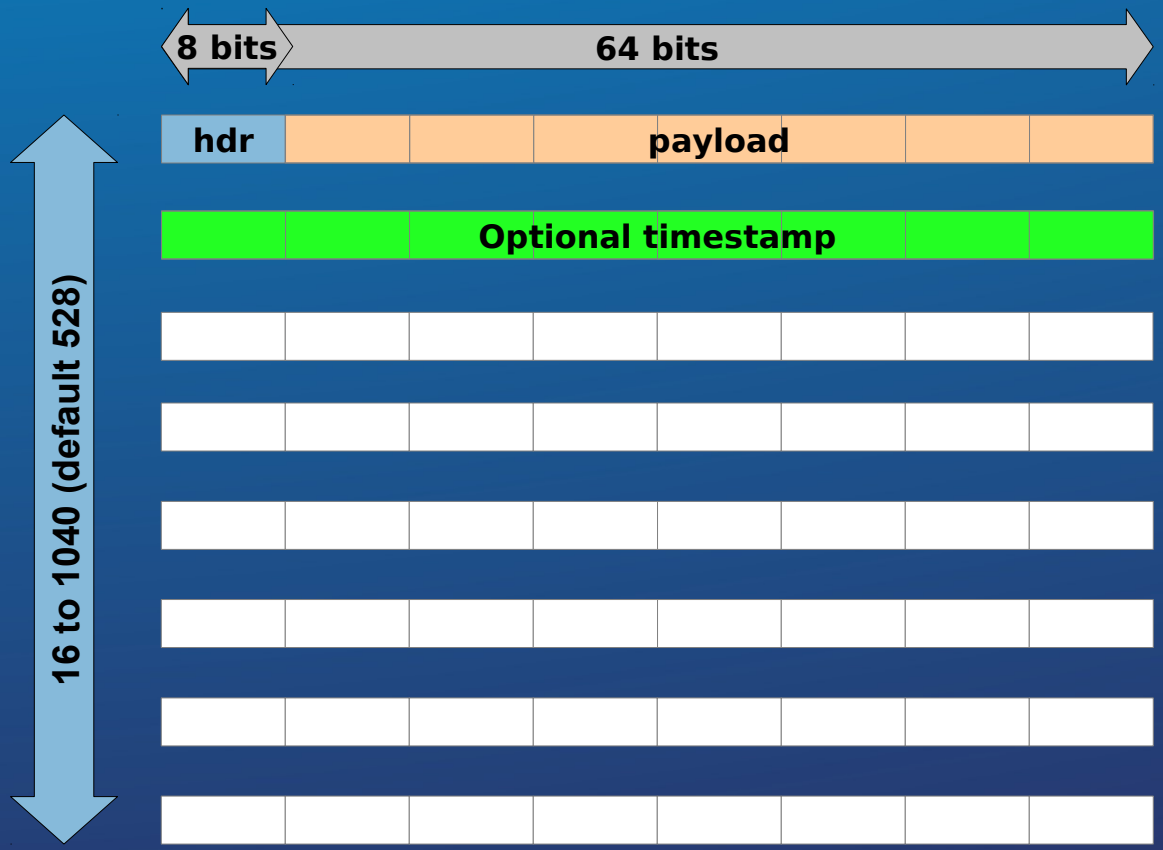
Hardware description: block diagram



Hardware description: Debug System Unit



Hardware description: Trace FIFO



- Memory mapped register
- Blocking or non-blocking
- HW detect lost messages (when non-blocking)
- HW timestamping

Agenda

- Hardware description
- **Software trace system**
- Trace visualization
- Use case #1
- Use case #2

Software trace system: details (1/2)

Static code instrumentation

- Defined at compile time

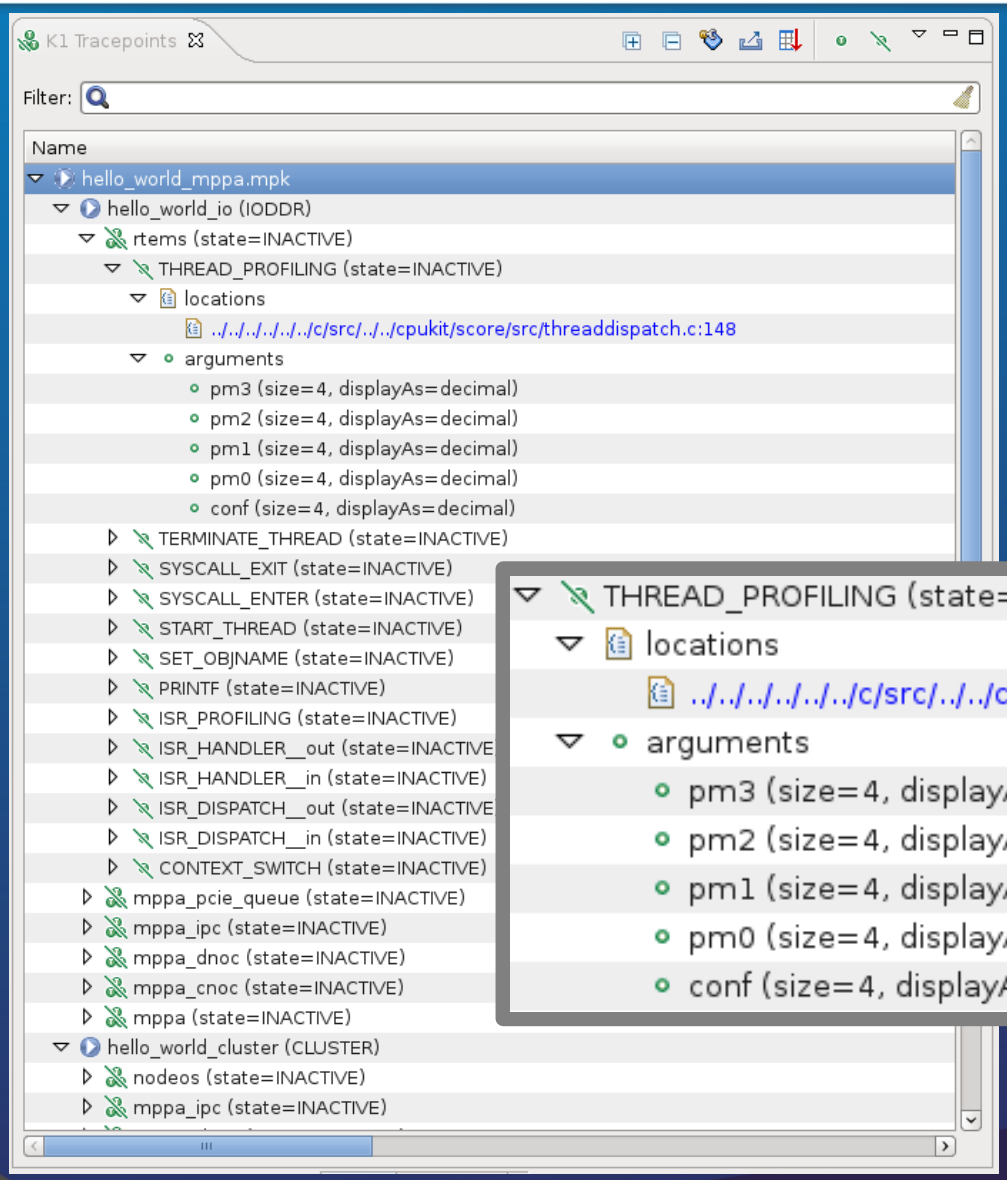
Compatible with LTT-ng

- code compiled for x86 generates UST LTT-ng tracepoints
- code compiled for MPPA generates MPPA tracepoints
- possibility to display both in the same viewer

MPPA tracepoints can be (de)activated

- today: by patching the elf file
- tomorrow: on the fly, during execution

Software trace system: details (2/2)



Software trace system: example (1/2)

```
/* test-trace.h */

/* Declare a tracepoint named "test/tp1", with the following
arguments:
- a int argument displayed in decimal format */
MPPA_DECLARE_TRACEPOINT(test, tp1, (
            MPPA_TRACEPOINT_DEC_FIELD(int,value1)) )

/* Declare a tracepoint named "test/tp4", with the following
arguments:
- a string argument
- a long long argument displayed in hexadecimal format */
MPPA_DECLARE_TRACEPOINT(test, tp4, (
            MPPA_TRACEPOINT_STRING_FIELD(char *, str),
            MPPA_TRACEPOINT_HEX_FIELD(long long,llvalue)))
```

Software trace system: example (2/2)

```
/* test-trace.c */
#include "test-trace.h"

int main (int argc, char ** argv, char ** env) {
    int i=1;
    unsigned int val1=1;
    unsigned long long llval = 0x0123456789ABCDEF;
    char * str="string for trace test";
    mppa_trace_init(); // usually done by OS.
    mppa_tracepoint(test,tp1, val1);
    mppa_tracepoint(test,tp4, str, llval);
    return 0;
}
```

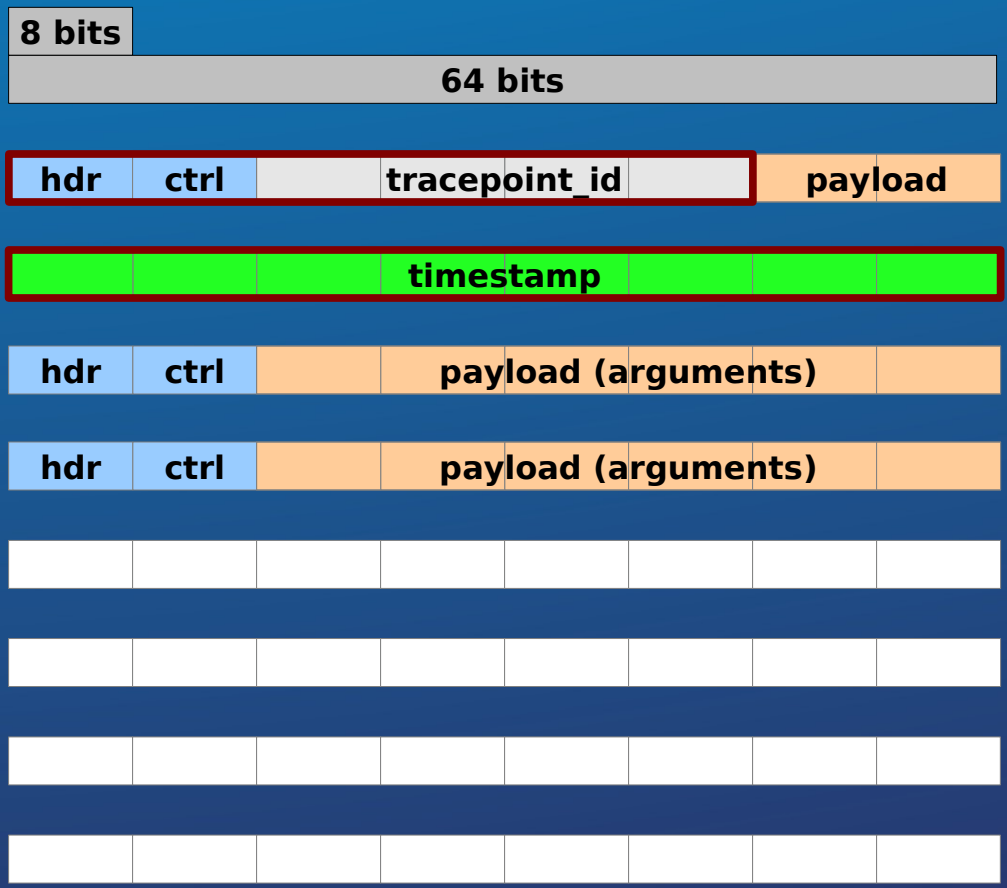
Software trace system: intrusiveness

Typical intrusiveness

- 1 cycle when a tracepoint is inactive
- 6 cycles for a simple tracepoint
- around +3 cycles for each additional argument

- 80 to 100 bytes of code size overhead.

Software trace system: FIFO Usage



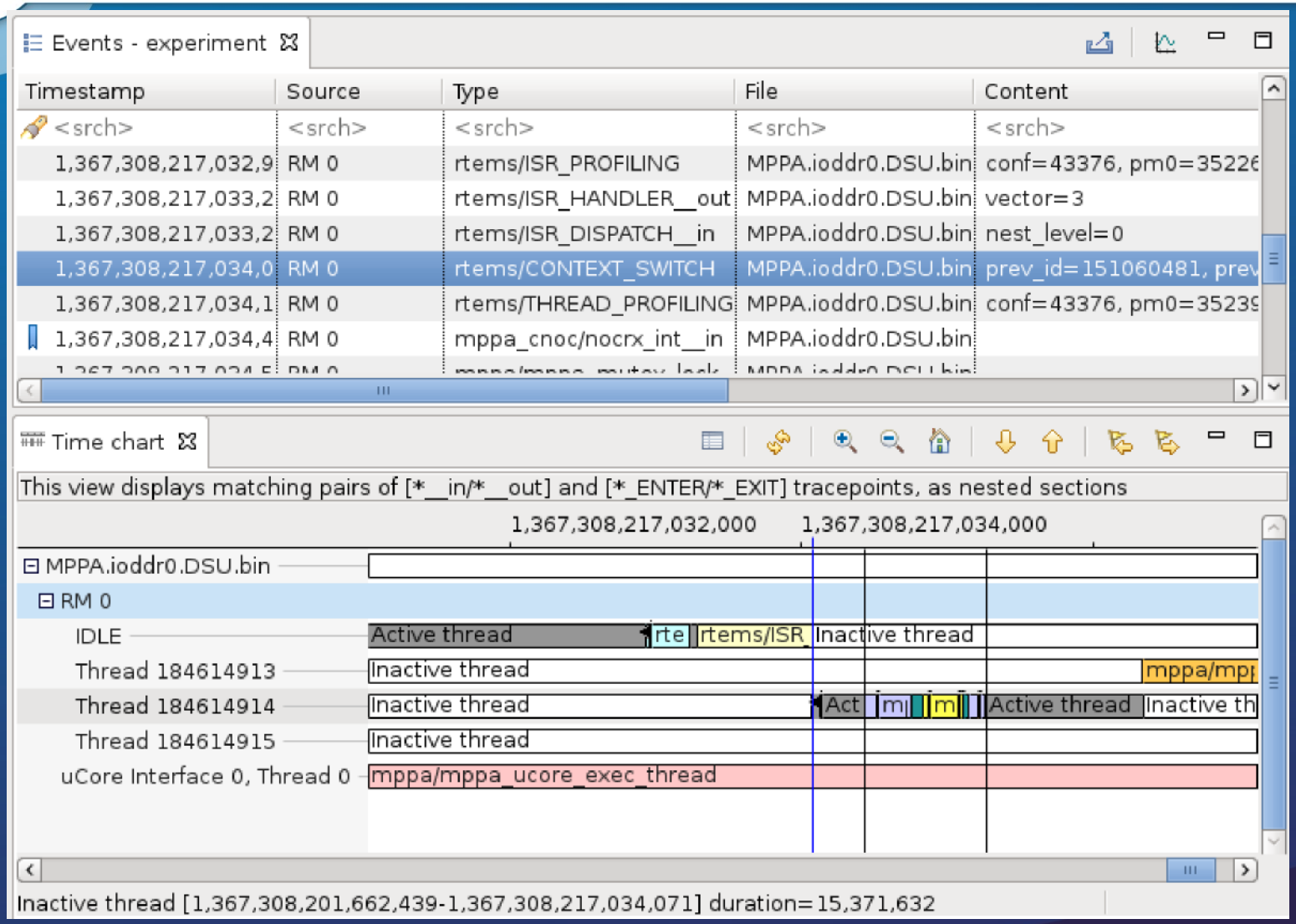
Legend:

- Minimal tracepoint
- Padded optional arguments

Agenda

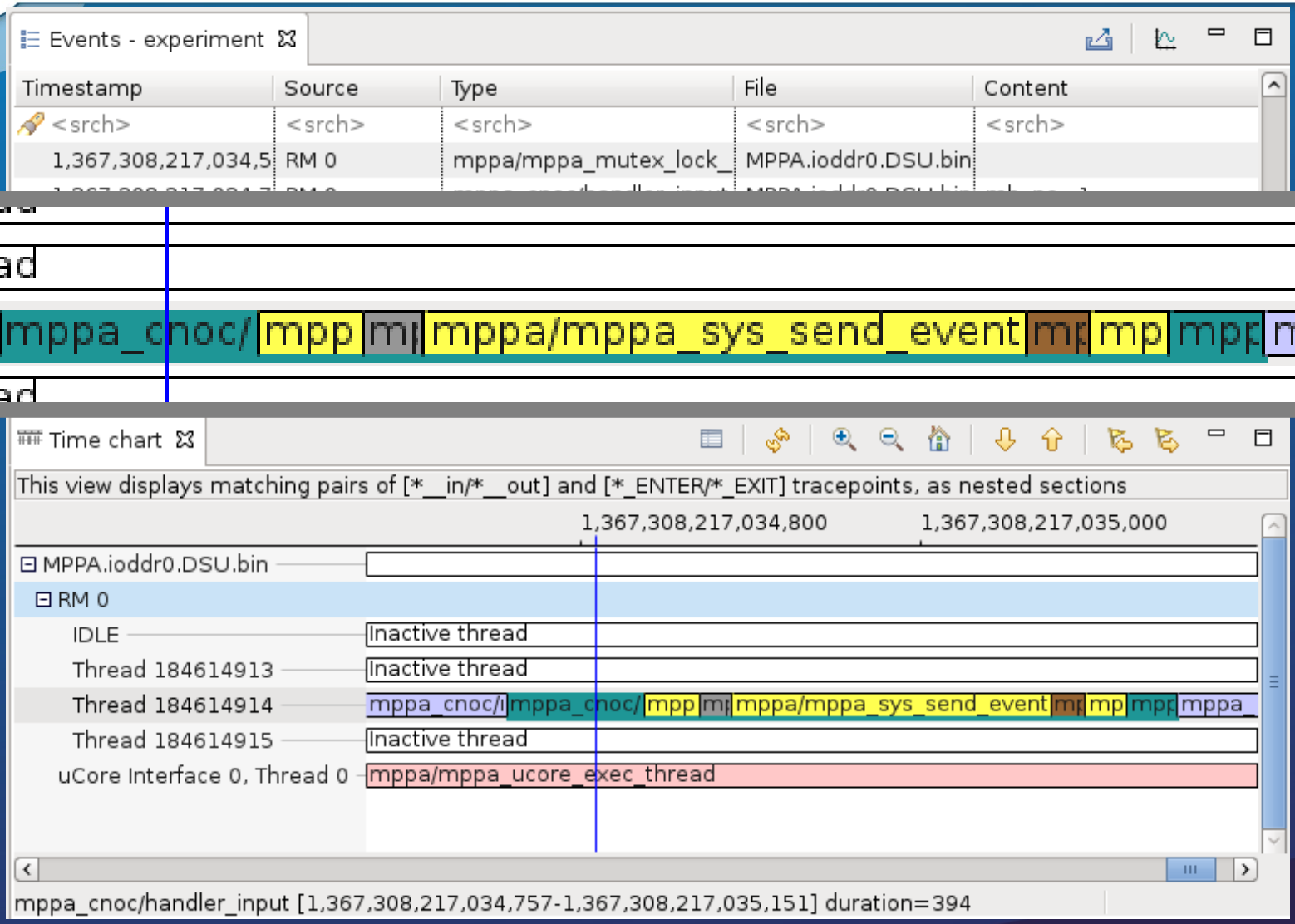
- Hardware description
- Software trace system
- **Trace visualization**
- Use case #1
- Use case #2

Trace visualization (1/4)



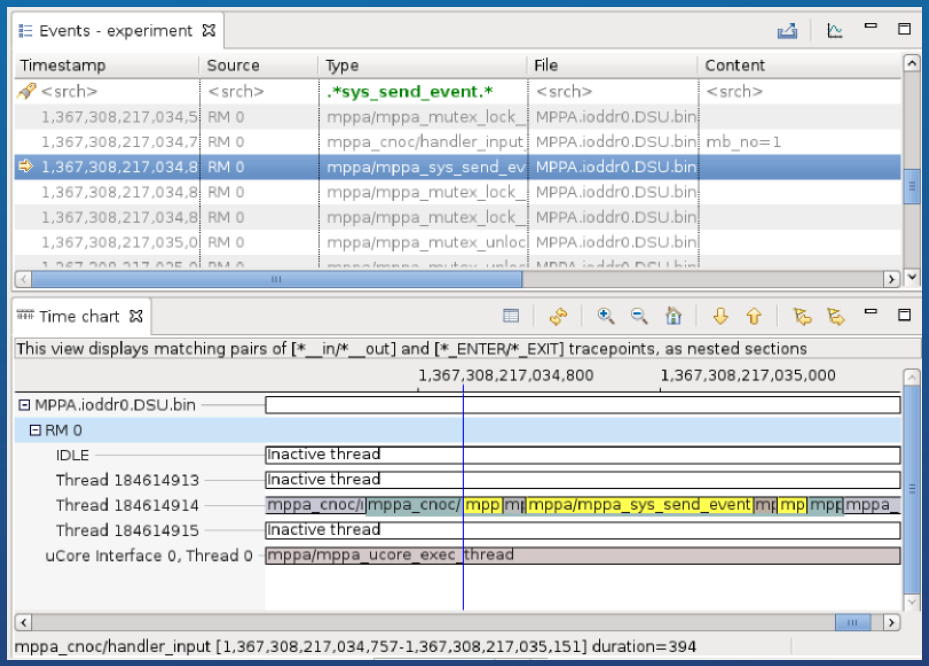
Programming-model aware

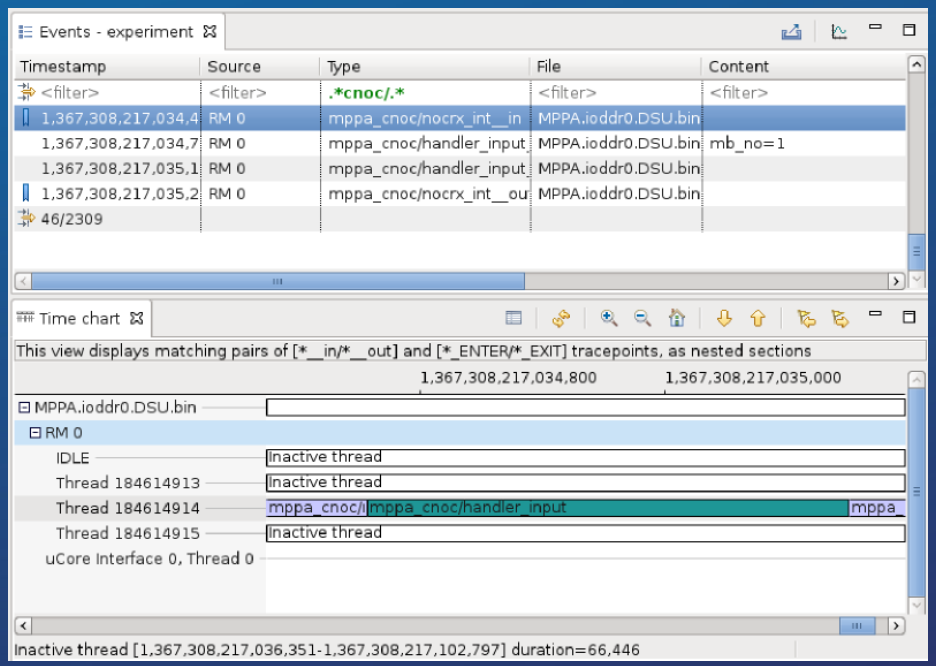
Trace visualization (2/4)



Z-order

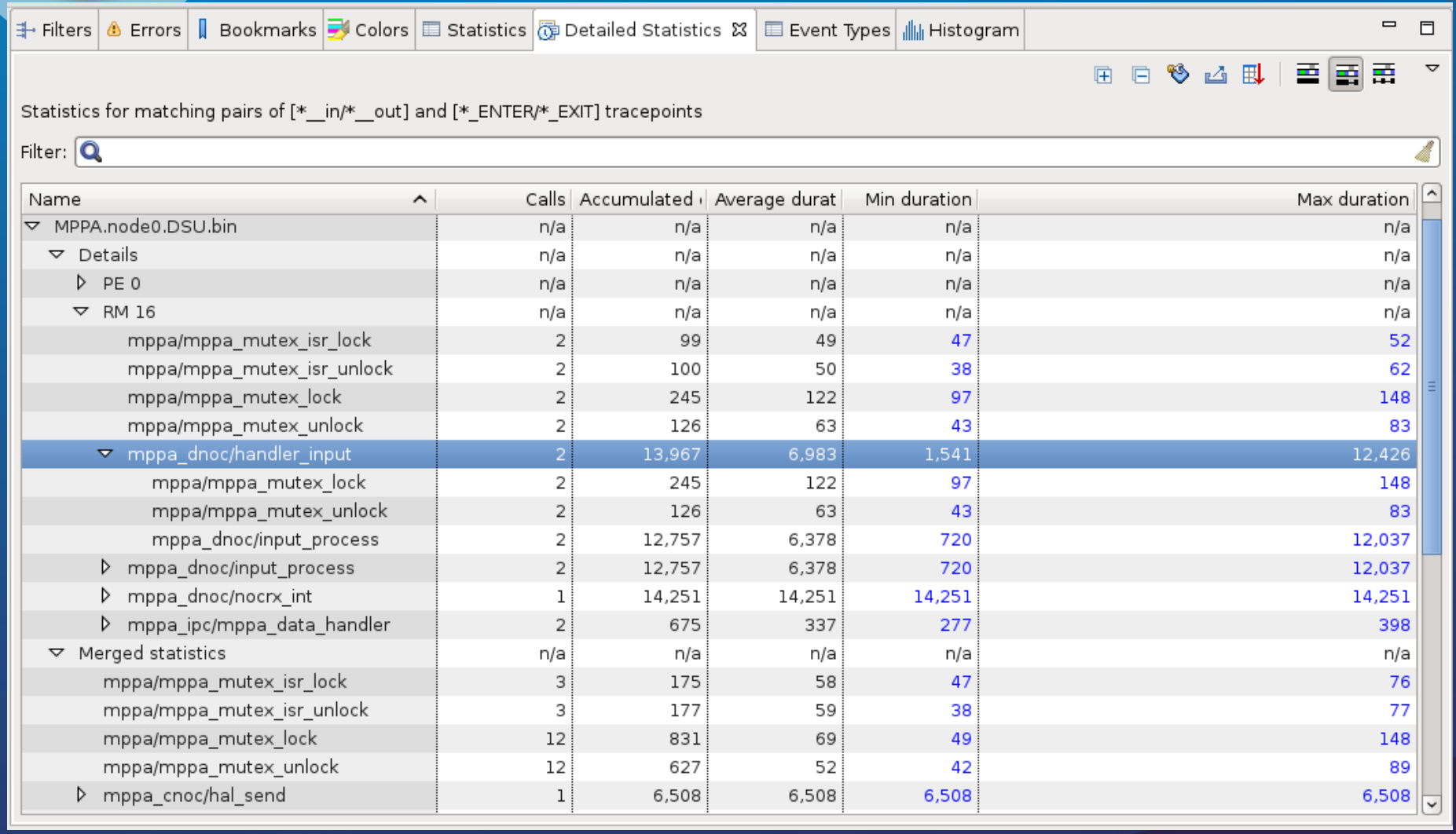
Trace visualization (3/4)





Search and filter

Trace visualization (4/4)

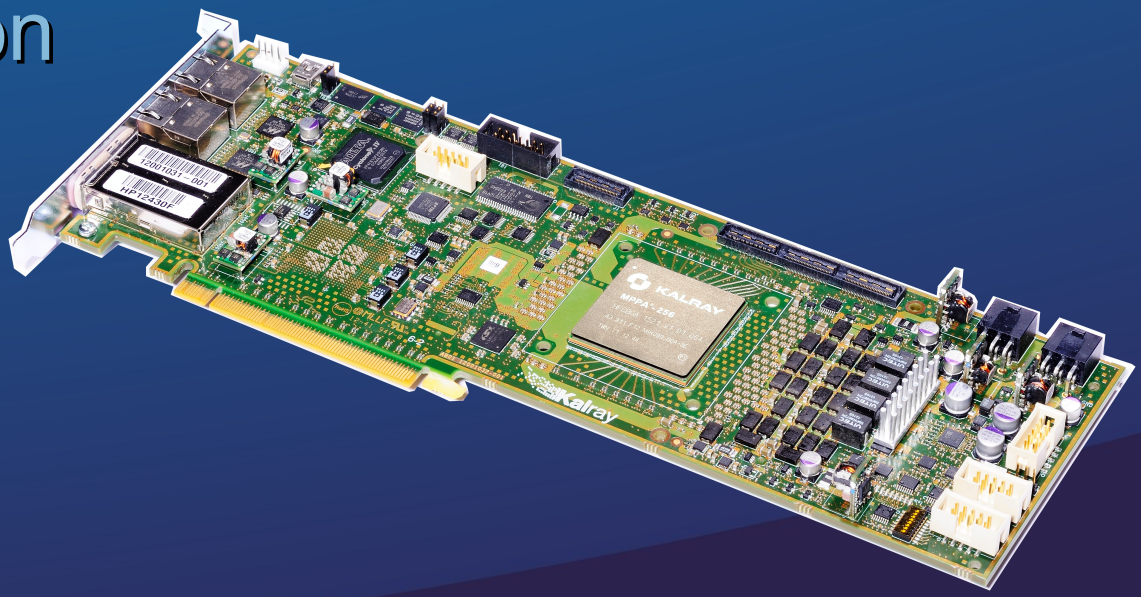


Name	Calls	Accumulated	Average durat	Min duration	Max duration
MPPA.node0.DSU.bin	n/a	n/a	n/a	n/a	n/a
Details	n/a	n/a	n/a	n/a	n/a
PE 0	n/a	n/a	n/a	n/a	n/a
RM 16	n/a	n/a	n/a	n/a	n/a
mppa/mppa_mutex_isr_lock	2	99	49	47	52
mppa/mppa_mutex_isr_unlock	2	100	50	38	62
mppa/mppa_mutex_lock	2	245	122	97	148
mppa/mppa_mutex_unlock	2	126	63	43	83
mppa_dnoc/handler_input	2	13,967	6,983	1,541	12,426
mppa/mppa_mutex_lock	2	245	122	97	148
mppa/mppa_mutex_unlock	2	126	63	43	83
mppa_dnoc/input_process	2	12,757	6,378	720	12,037
mppa_dnoc/input_process	2	12,757	6,378	720	12,037
mppa_dnoc/nocrx_int	1	14,251	14,251	14,251	14,251
mppa_ipc/mppa_data_handler	2	675	337	277	398
Merged statistics	n/a	n/a	n/a	n/a	n/a
mppa/mppa_mutex_isr_lock	3	175	58	47	76
mppa/mppa_mutex_isr_unlock	3	177	59	38	77
mppa/mppa_mutex_lock	12	831	69	49	148
mppa/mppa_mutex_unlock	12	627	52	42	89
mppa_cnoc/hal_send	1	6,508	6,508	6,508	6,508

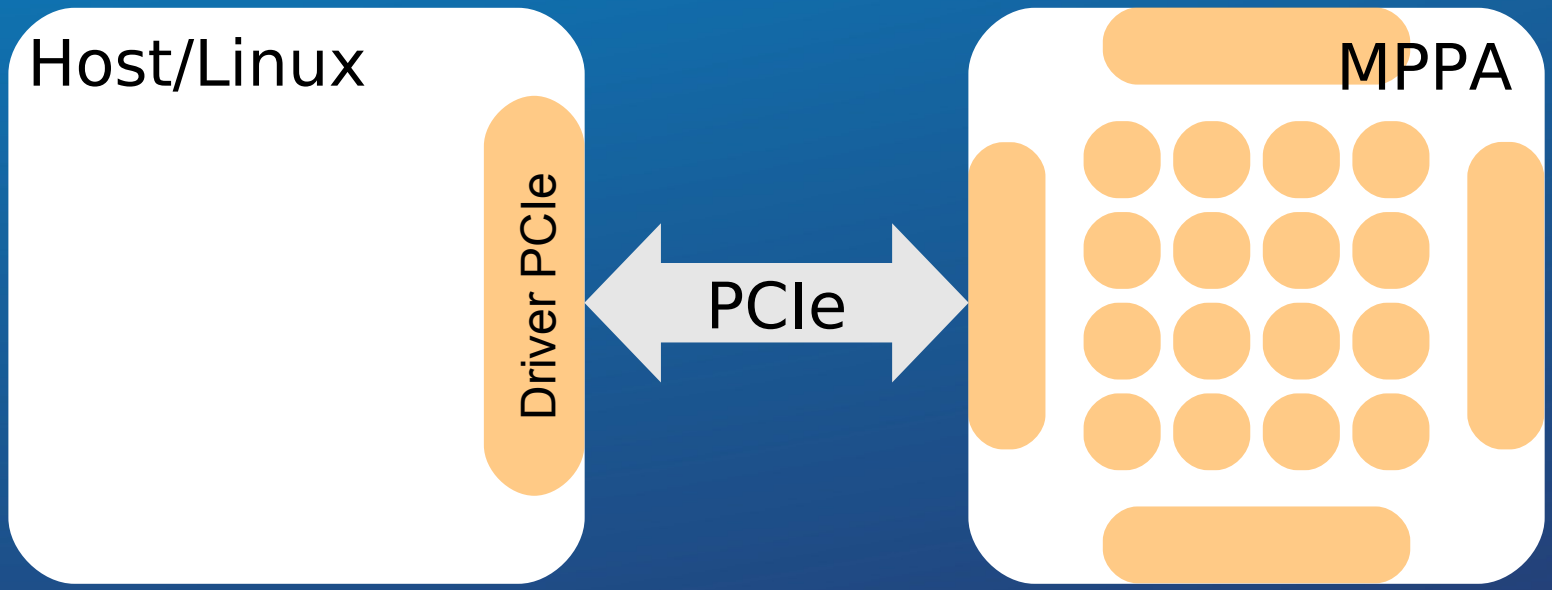
Statistics

Agenda

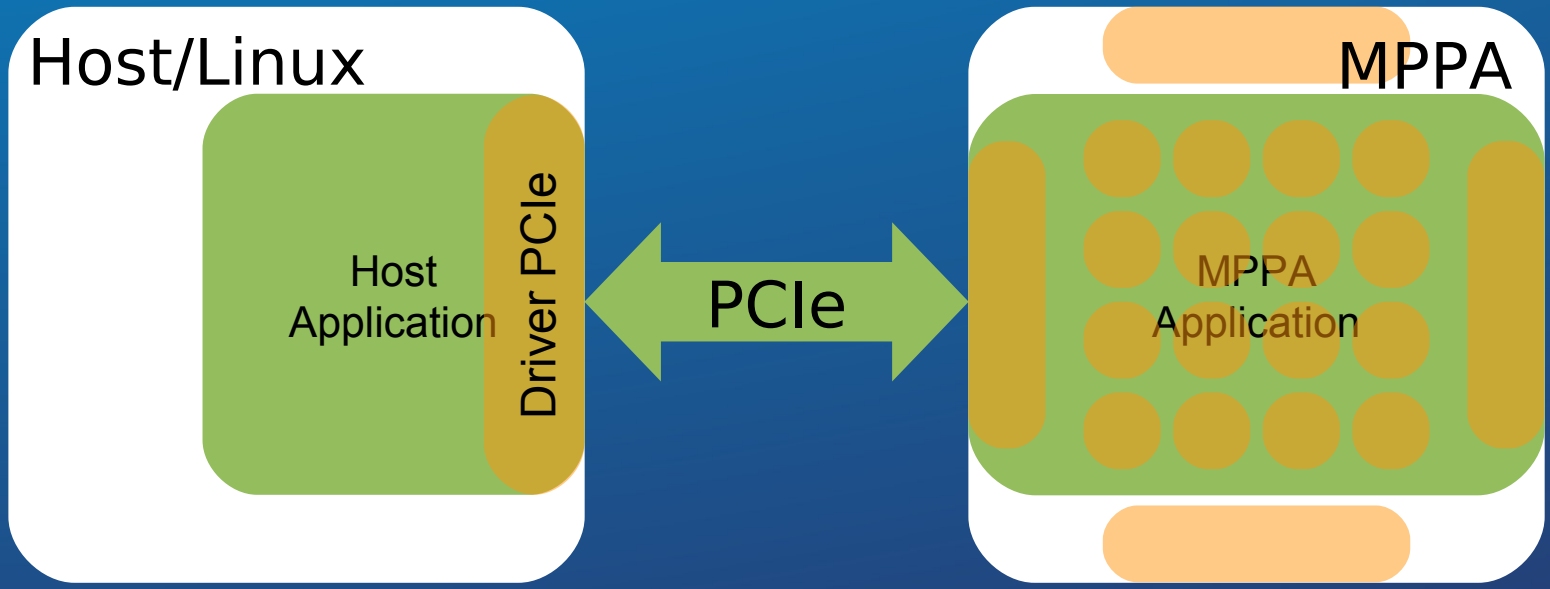
- Hardware description
- Software trace system
- Trace visualization
- **Use case #1**
- Use case #2



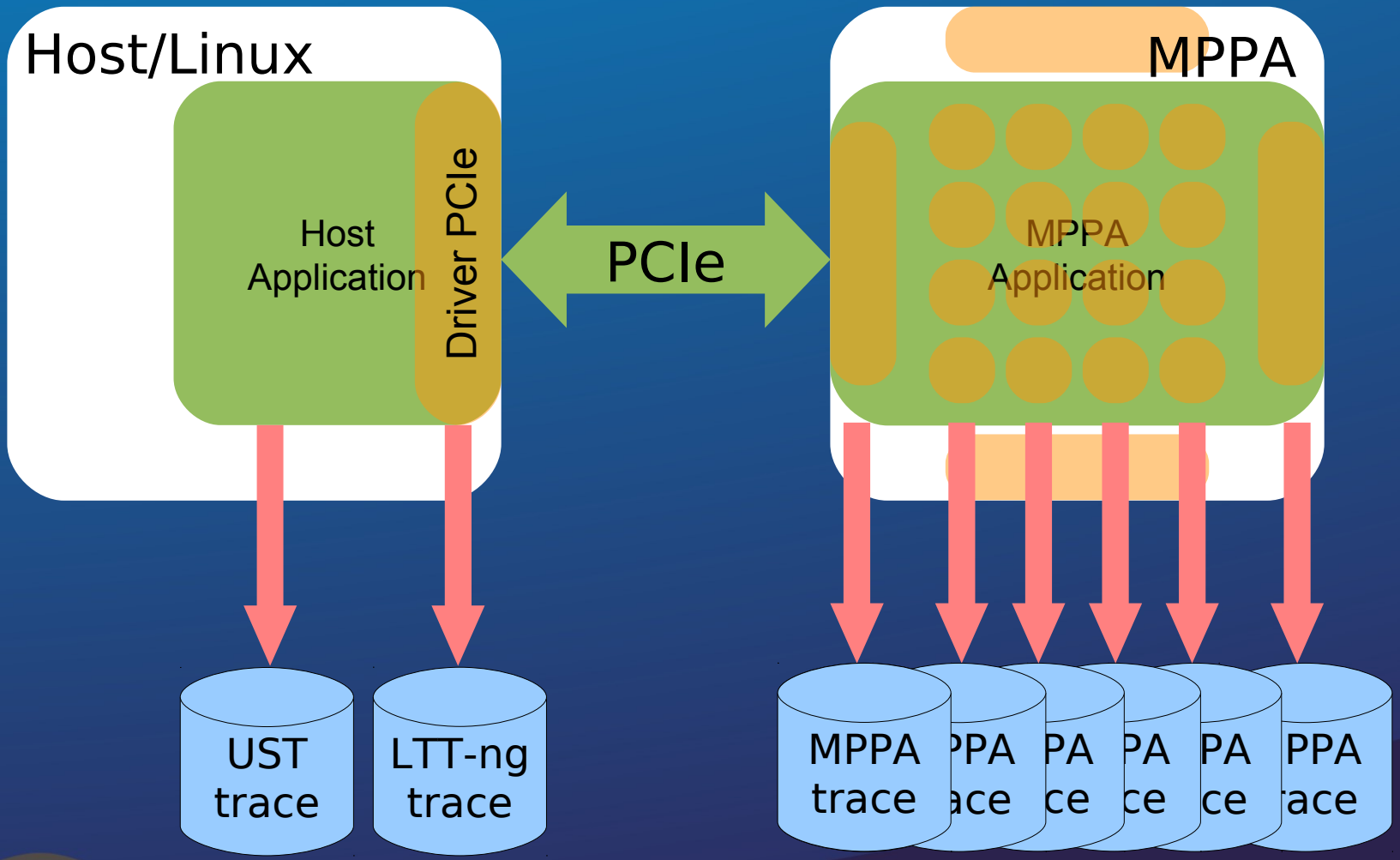
Use case #1: PCIe application (1/4)



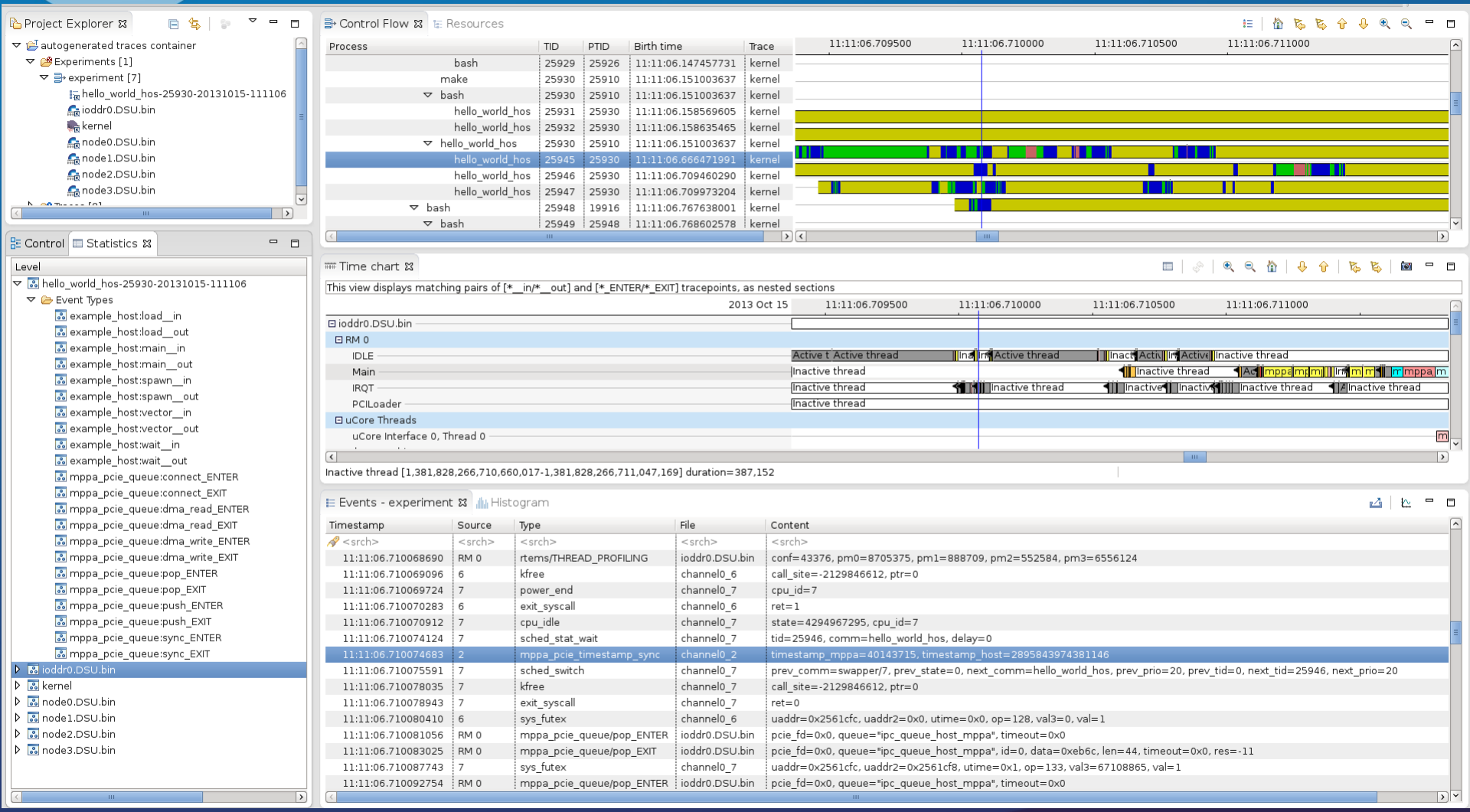
Use case #1: PCIe application (2/4)



Use case #1: PCIe application (3/4)



Use case #1: PCIe application (4/4)

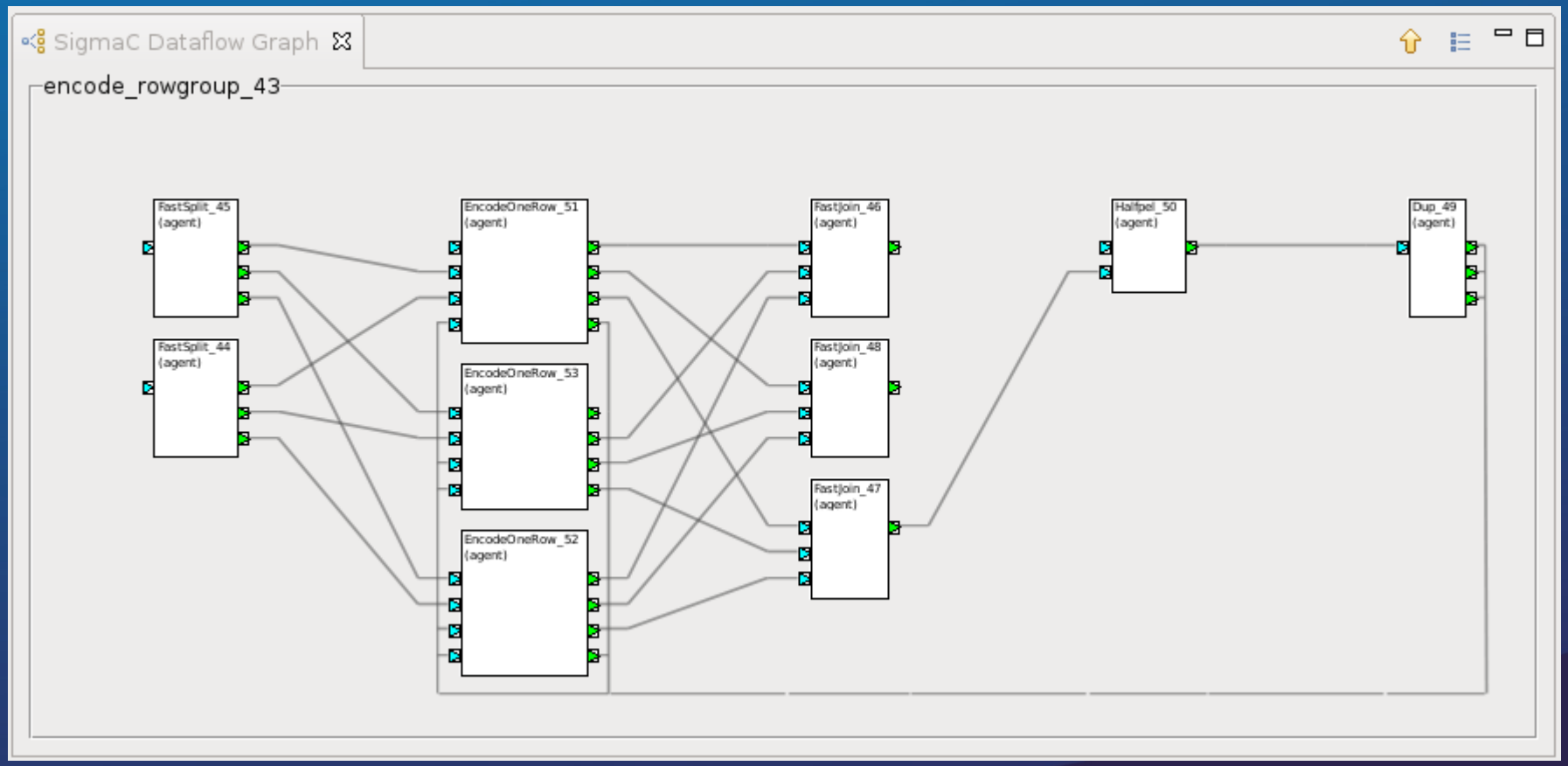


Agenda

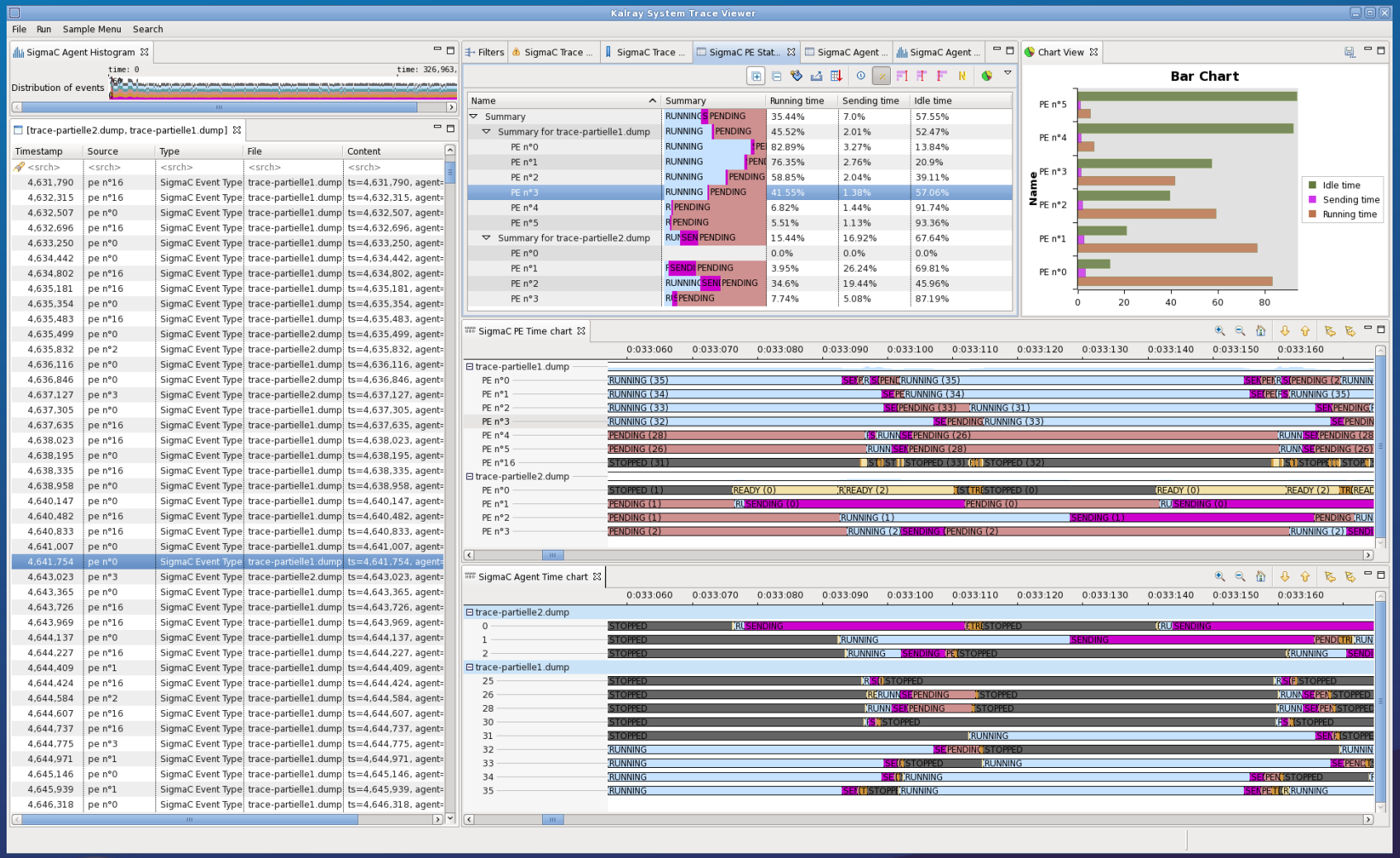
- Hardware description
- Software trace system
- Trace visualization
- Use case #1
- **Use case #2**

Use case #2: Dataflow programming

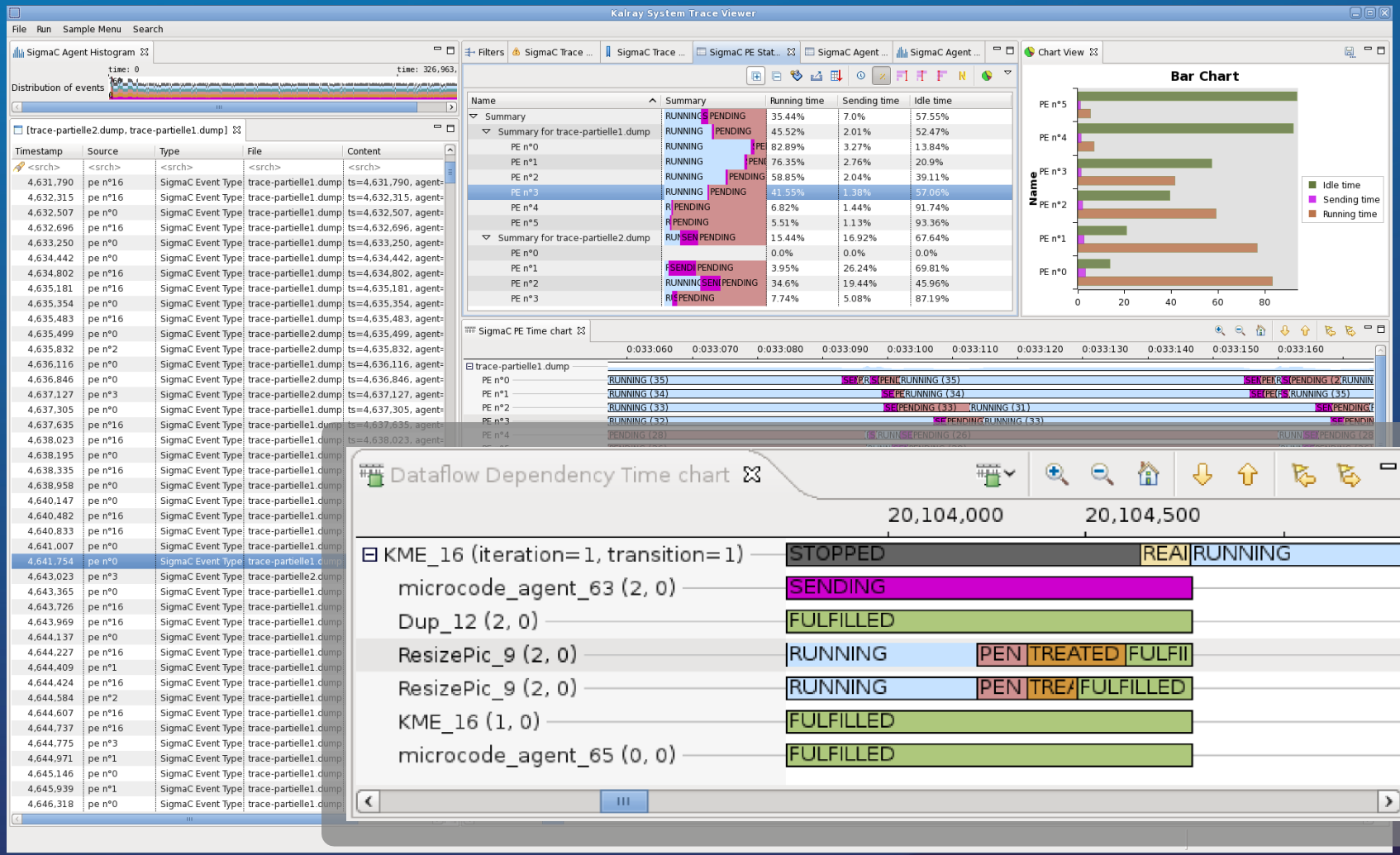
SigmaC:
A C-based parallel dataflow programming model



Dataflow-aware traces



Dataflow traces: dependencies tracking



Conclusion

Tracing all parts of the application is easy

- MPPA clusters, Host kernel and user space

Visualizing all traces in one tool is even easier

- Widely used by application and library developers to debug, verify, profile...

Future plans

- Ftrace-like instrumentation of our libraries
- Controlling & visualizing traces at runtime (from debugger)

Contributions

- All changes made in TMF (Eclipse Trace Management Framework) integrated upstream

KALRAY

Xavier Raynaud
xavier.raynaud@kalray.eu

http://www.kalray.eu

More info:

Embedded Linux Conf.
Thursday, October 24th 4:30pm
Going Linux on Massive Multicore – Marta Rybczynska
http://sched.co/1dvaTmD