Tracing and Sampling for Real-Time partially simulated Avionics Systems

Raphaël Beamonte Michel Dagenais

Computer and Software Engineering Department
École Polytechnique de Montréal
C.P.6079, Station Downtown, Montréal, Québec, Canada, H3C 3A7

DORSAL Project Meeting
December 5, 2012

Introduction

- Tracing:
  - Study runtime behavior
  - Can be used to measure latency = fundamental for RT debug
- Tracer requirements:
  - Low-overhead
  - Consistant maximum latency
- Contribution:
  - Methodology and tool to measure real-time latencies (NPT)
  - Application of NPT to measure LTTng-UST latency
  - Improvements to the real-time behavior of LTTng

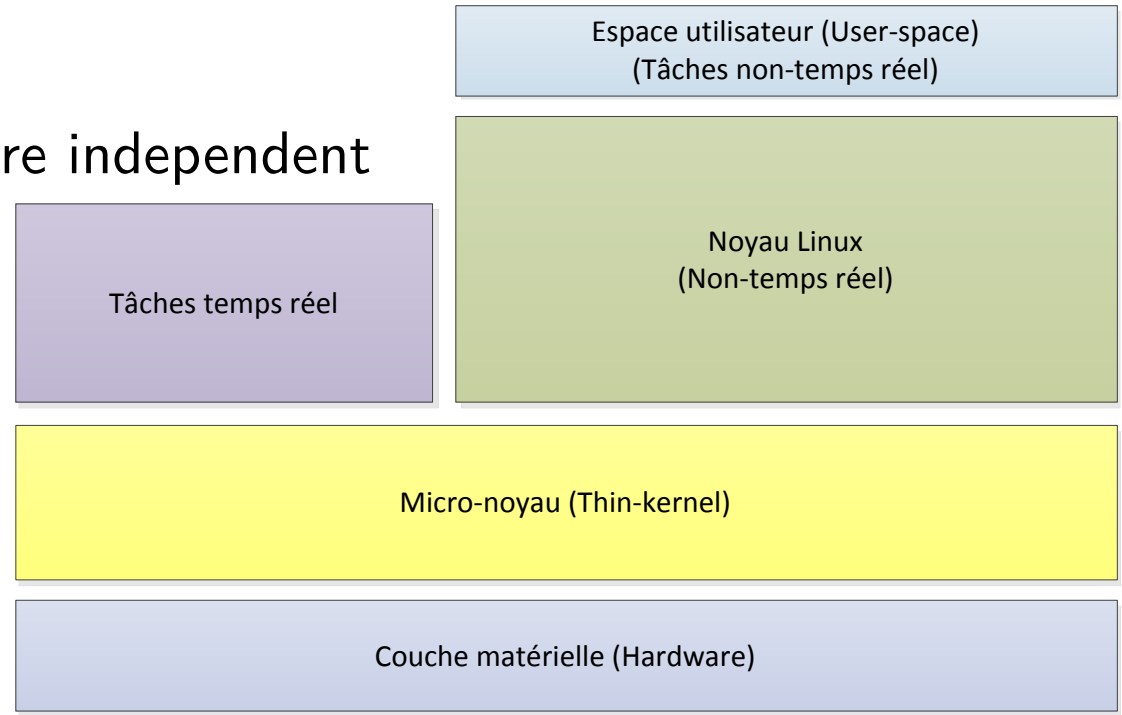
Table of Contents

RTOS: Xenomai vs. Linux

Xenomai:

- ADEOS thin-kernel
- Interrupt management – non-RT cannot preempt RT
- Hard Real Time ?

- RT and non-RT tasks are independent
- Full Linux platform ?



RTOS: Xenomai vs. Linux

Why using the Linux kernel ?

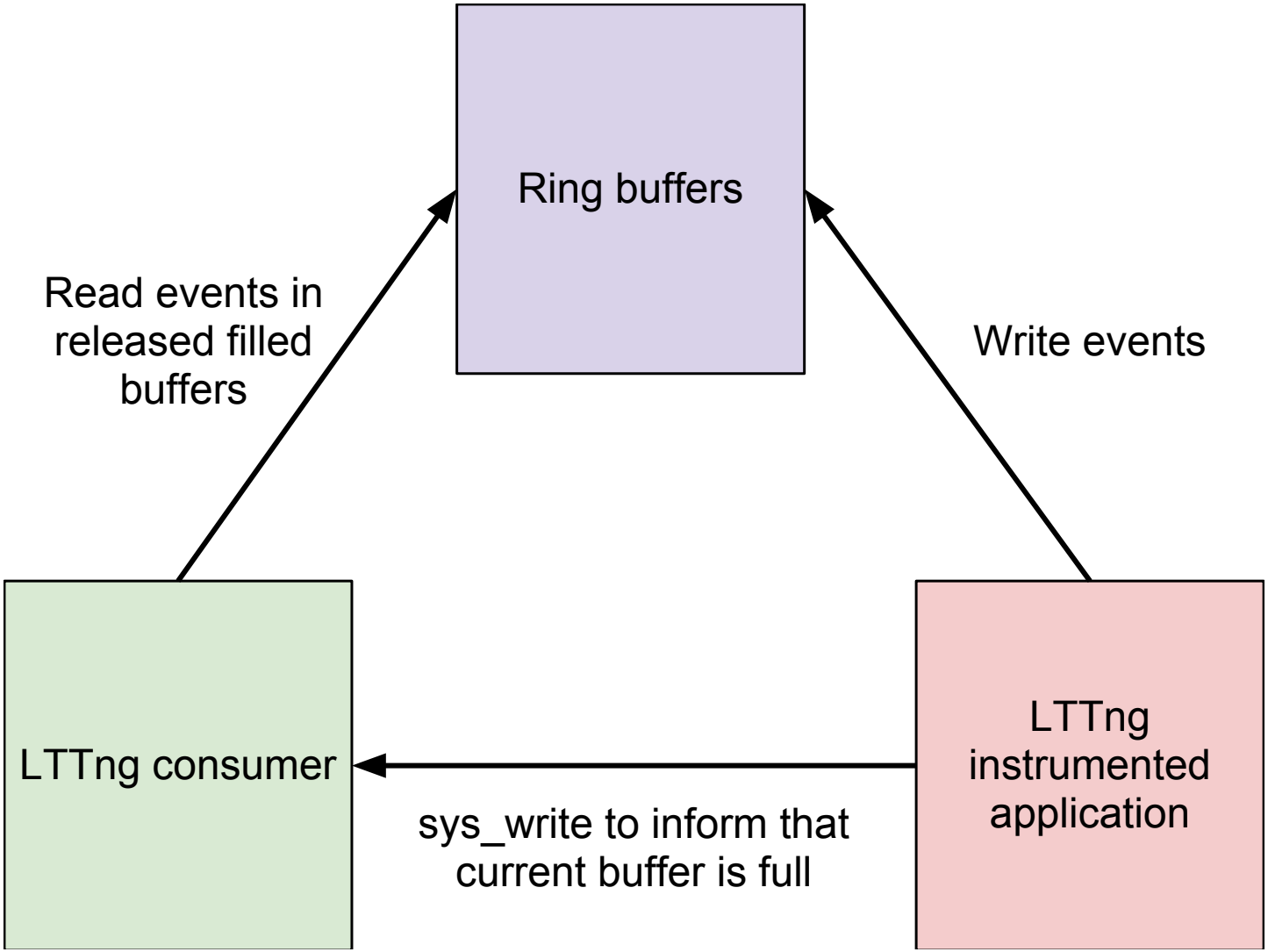
- Able to do Soft Real Time, can reach Hard Real Time :
  - BIOS configuration: would you use hyperthreading ?
  - Kernel configuration: PREEMPT_RT patch, which is more and more integrated to the standard kernel
  - Software configuration: interrupts redirection, cpu shielding...
- The power of the community

The Linux Tracing Toolkit next-generation, LTTng

Why LTTng is pertinent for RT applications ?

- Both userspace and kernel tracers (same clock source)
- Statically compiled tracepoints
- External process to consume events
- Arbitrary event types (Common Trace Format)
- Per-CPU ring buffers
- Important tracing variables protected by RCU

How LTTng-UST consumer works (simplified version)



Test environment

Hardware:

CPU	Intel® Core™ i7 CPU 920 2.67 GHz
RAM	3×2 GiB DDR3 at 1 067 MHz
Motherboard	Intel DX58SO

Kernels:

Standard	debian Linux kernel 3.2.0-3-amd64 package version 3.2.21-3
RT	debian Linux kernel 3.2.0-3-rt-amd64 package version 3.2.23-1

System verification

hwlatdetect (`hwlat_detector`): no hardware latency detected during one hour.

```
hwlatdetect:  test duration 3600 seconds
   parameters:
        Latency threshold: 10us
        Sample window:     1000000us
        Sample width:      500000us
     Non-sampling period:  500000us
        Output File:       None

Starting test
test finished
Max Latency: 0us
Samples recorded: 0
Samples exceeding threshold: 0
```

Why NPT ?

- What we have with known tools:
  - `cyclictest`: runs periodic tasks and calculates discrepancy between desired and real period
  - `preempt-test`: verify if higher priority tasks can preempt lower ones
- What we want:
  - A high-priority process that should not stop
  - No latency during the run of this process (no preemption)

How NPT works ?

- Sets CPU affinity
- Sets RT priority
- Locks process memory into RAM to disable swapping
- Disables local IRQs

- Non-stop loops to calculate statistics with `rdtsc`

- Re-enables local IRQs
- Prints computed statistics

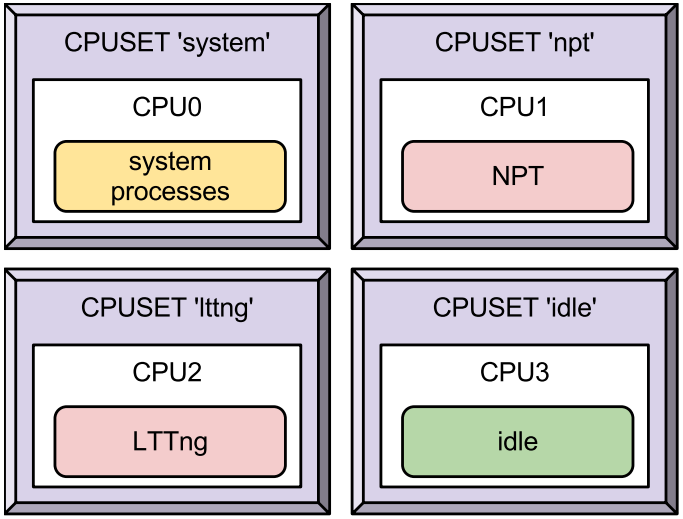
Algorithm of NPT's main loop

1: $i \leftarrow 0$
2: $t_0 \leftarrow$ **read** *rdtsc*
3: $t_1 \leftarrow t_0$
4:
5: **while** $i \leq cycles\_to\_do$ **do**
6: $\quad i \leftarrow i + 1$
7: $\quad duration \leftarrow (t_0 - t_1) \times cpuPeriod$
8:
9: $\quad$ CalculateStatistics(*duration*)
10: $\quad t_1 \leftarrow t_0$
11: $\quad t_0 \leftarrow$ **read** *rdtsc*
12: **end while**

Algorithm of NPT's main loop

1: $i \leftarrow 0$
2: $t_0 \leftarrow$ **read** *rdtsc*
3: $t_1 \leftarrow t_0$
4: **tracepoint** *nptstart*
5: **while** $i \leq$ *cycles_to_do* **do**
6: $\quad i \leftarrow i + 1$
7: $\quad$ *duration* $\leftarrow (t_0 - t_1) \times$ *cpuPeriod*
8: $\quad$ **tracepoint** *nptloop*
9: $\quad$ CalculateStatistics(*duration*)
10: $\quad t_1 \leftarrow t_0$
11: $\quad t_0 \leftarrow$ **read** *rdtsc*
12: **end while**
13: **tracepoint** *nptstop*

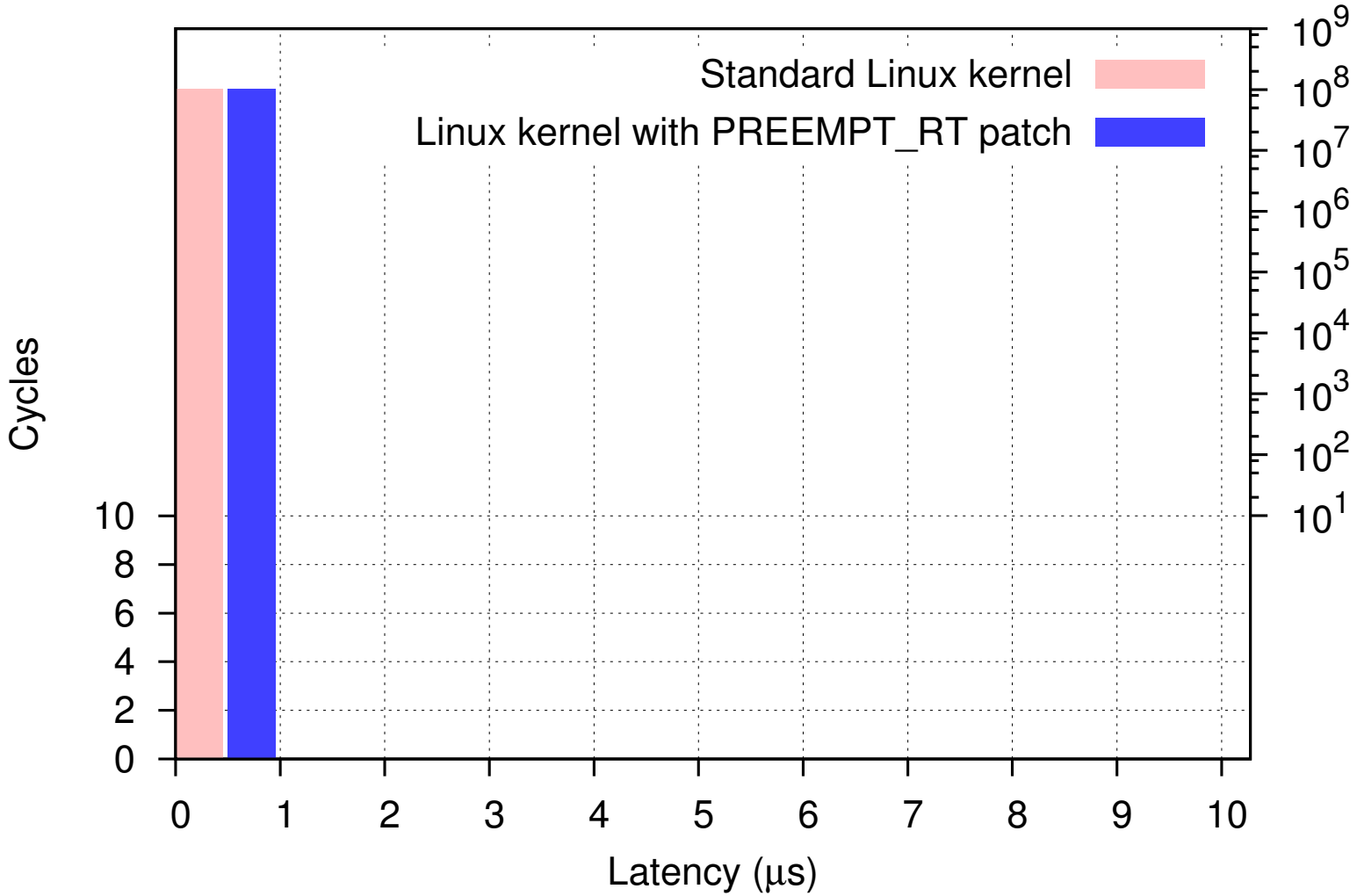
The test procedure



- Shield CPUs (`cpusets`)
- Run NPT for 10^8 loops:
  - Without tracing
  - With LTTng kernel tracing alone
  - With LTTng-UST tracing alone
  - With LTTng-UST and kernel tracing
- Do it on:
  - Standard kernel
  - PREEMPT_RT patched kernel

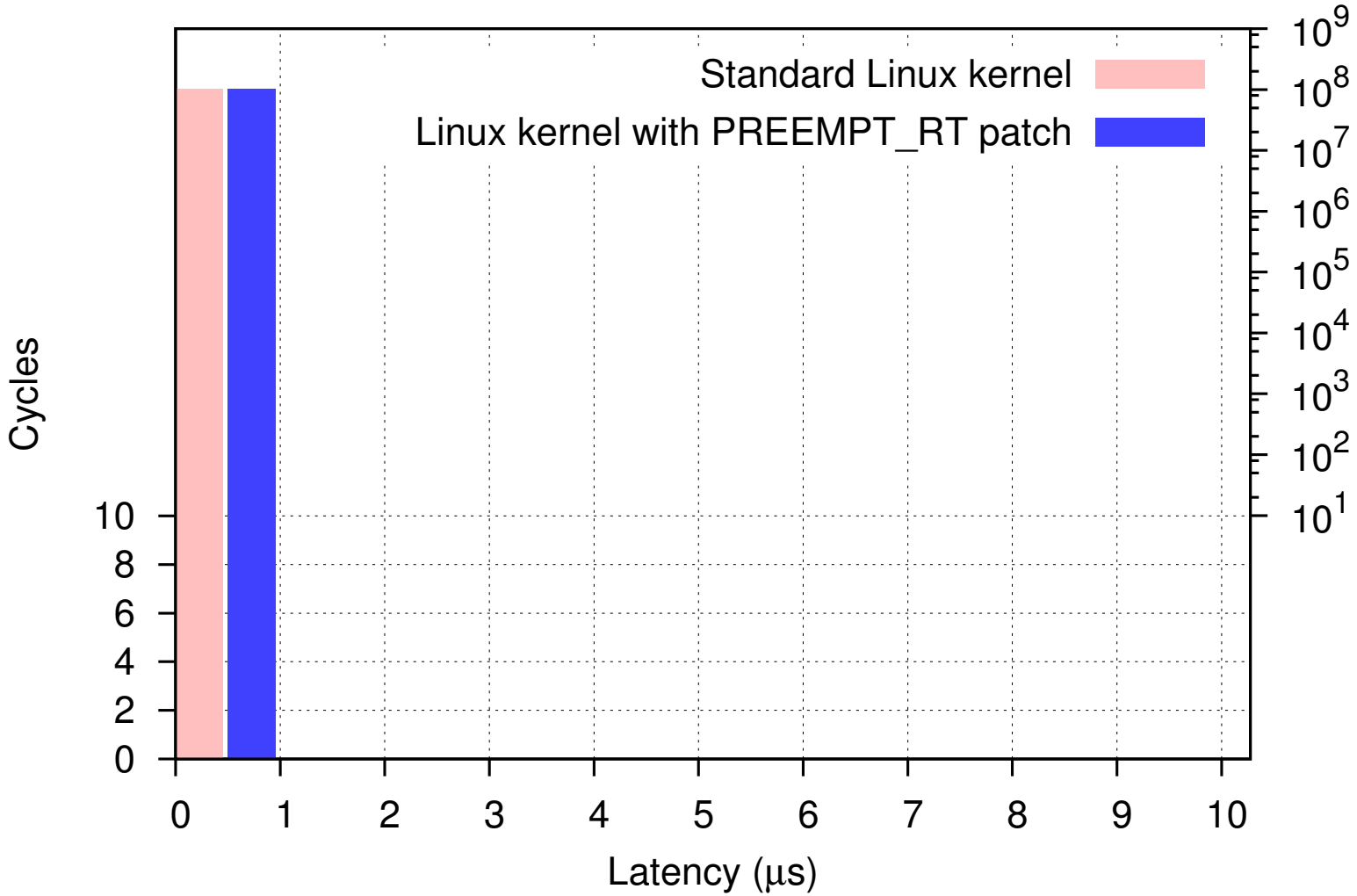
Latency results without tracing

Histogram generated by NPT for 10^8 cycles on standard and RT kernels



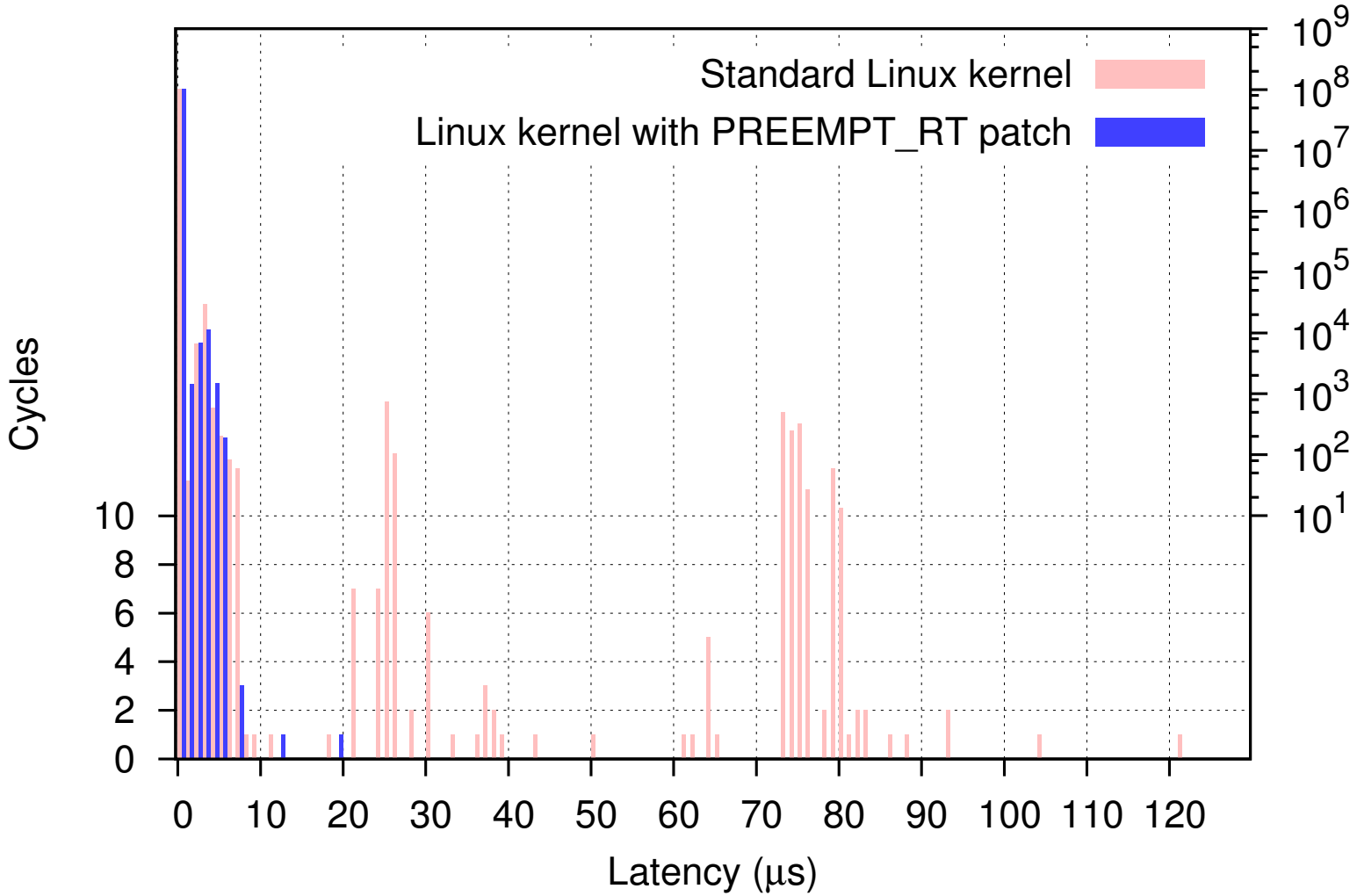
Latency results with LTTng kernel tracing

Histogram generated by NPT for 10^8 cycles on standard and RT kernels



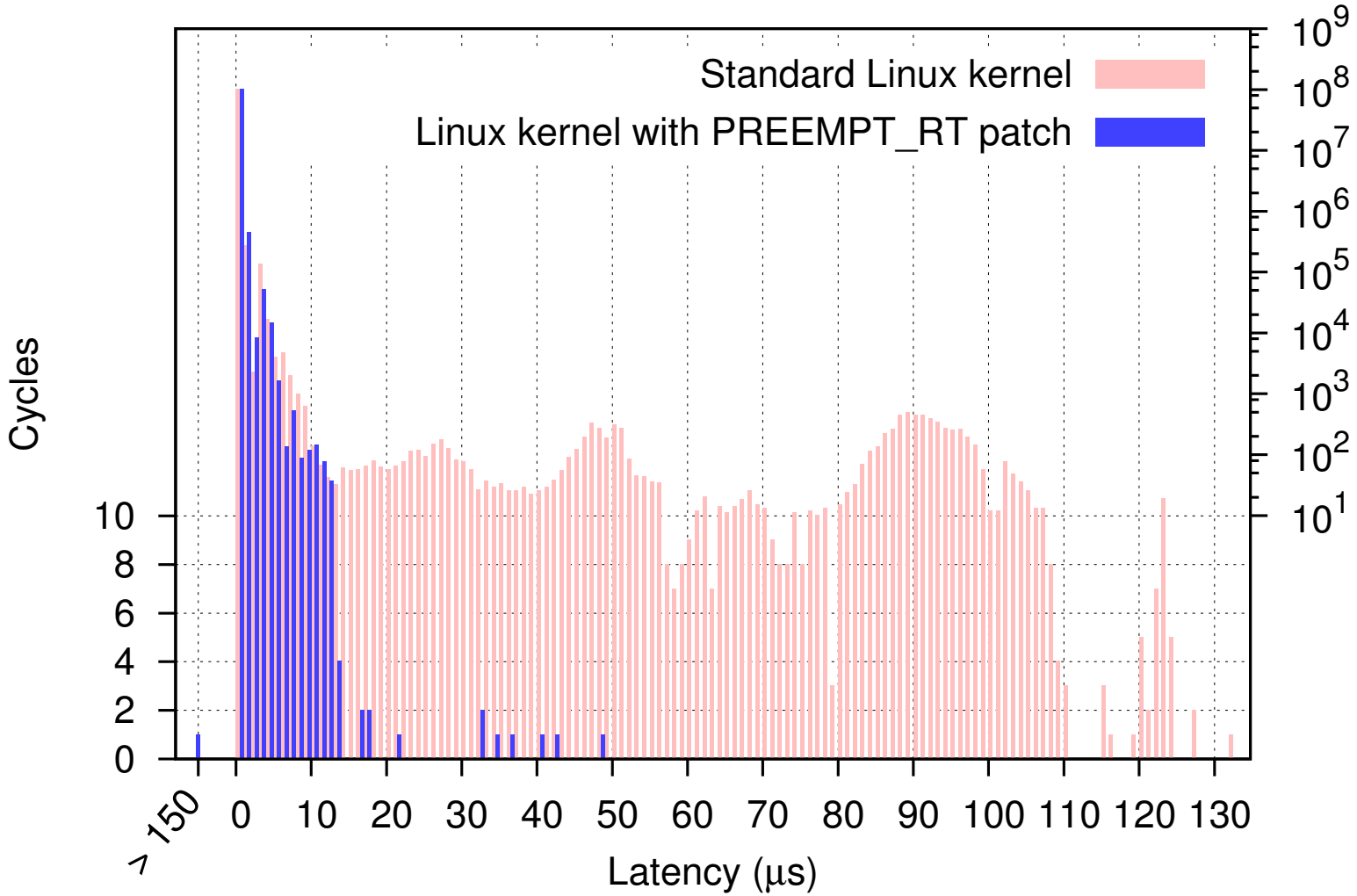
Latency results with LTTng-UST tracing

Histogram generated by NPT for 10^8 cycles on standard and RT kernels



Latency results with LTTng-UST and kernel tracing

Histogram generated by NPT for 10^8 cycles on standard and RT kernels



Identify the source of the latency

Problem

Latency added by the LTTng-UST tracing synchronization

Proposed solution

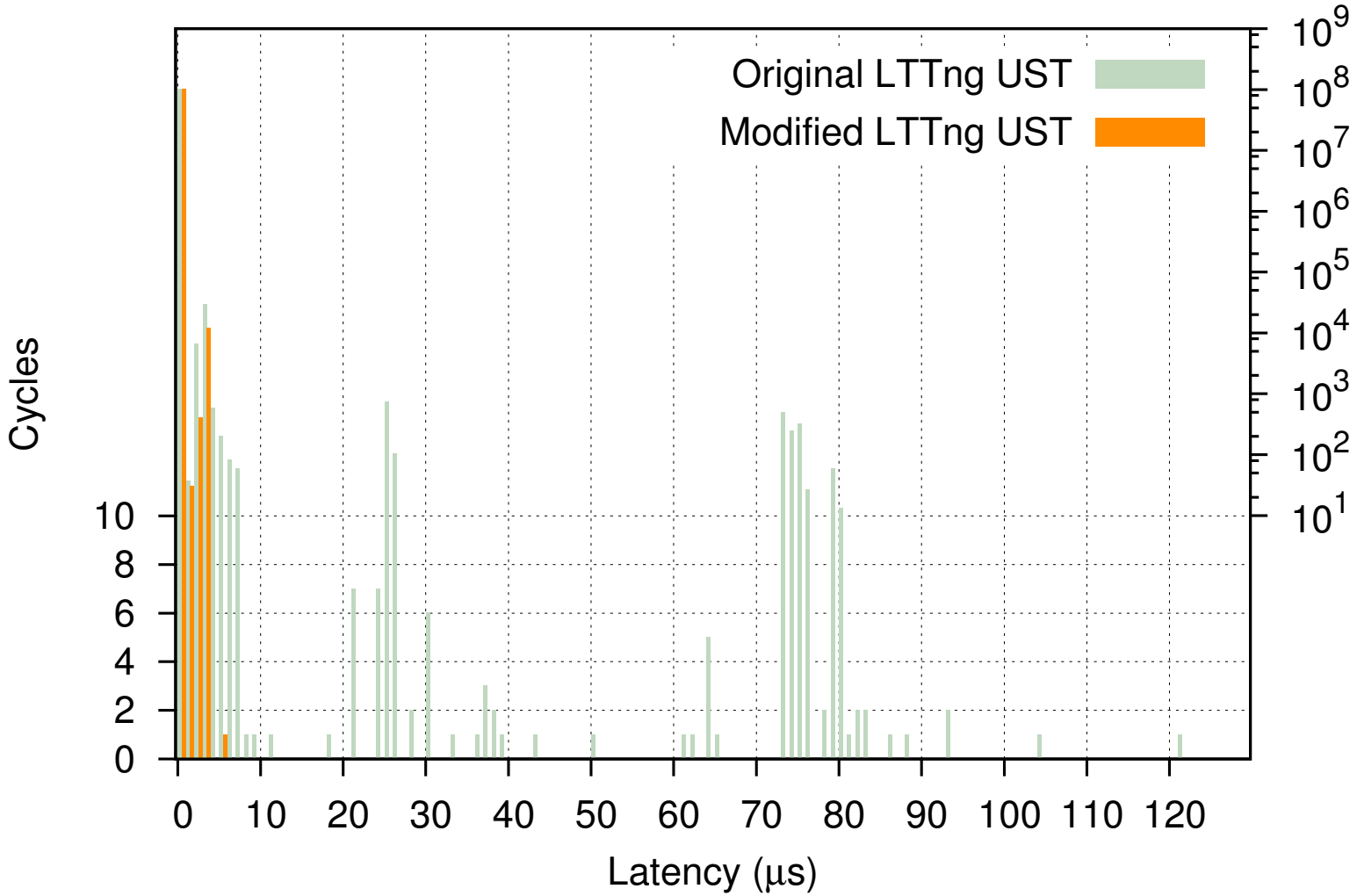
Removing synchronization between instrumented application and LTTng consumer:

- The consumer will now poll to verify if a buffer is full
- Permanent polling (100% CPU use) and `usleep`-timed polling = same performances (CPU shielding)

Removing LTTng-UST `getcpu` system call

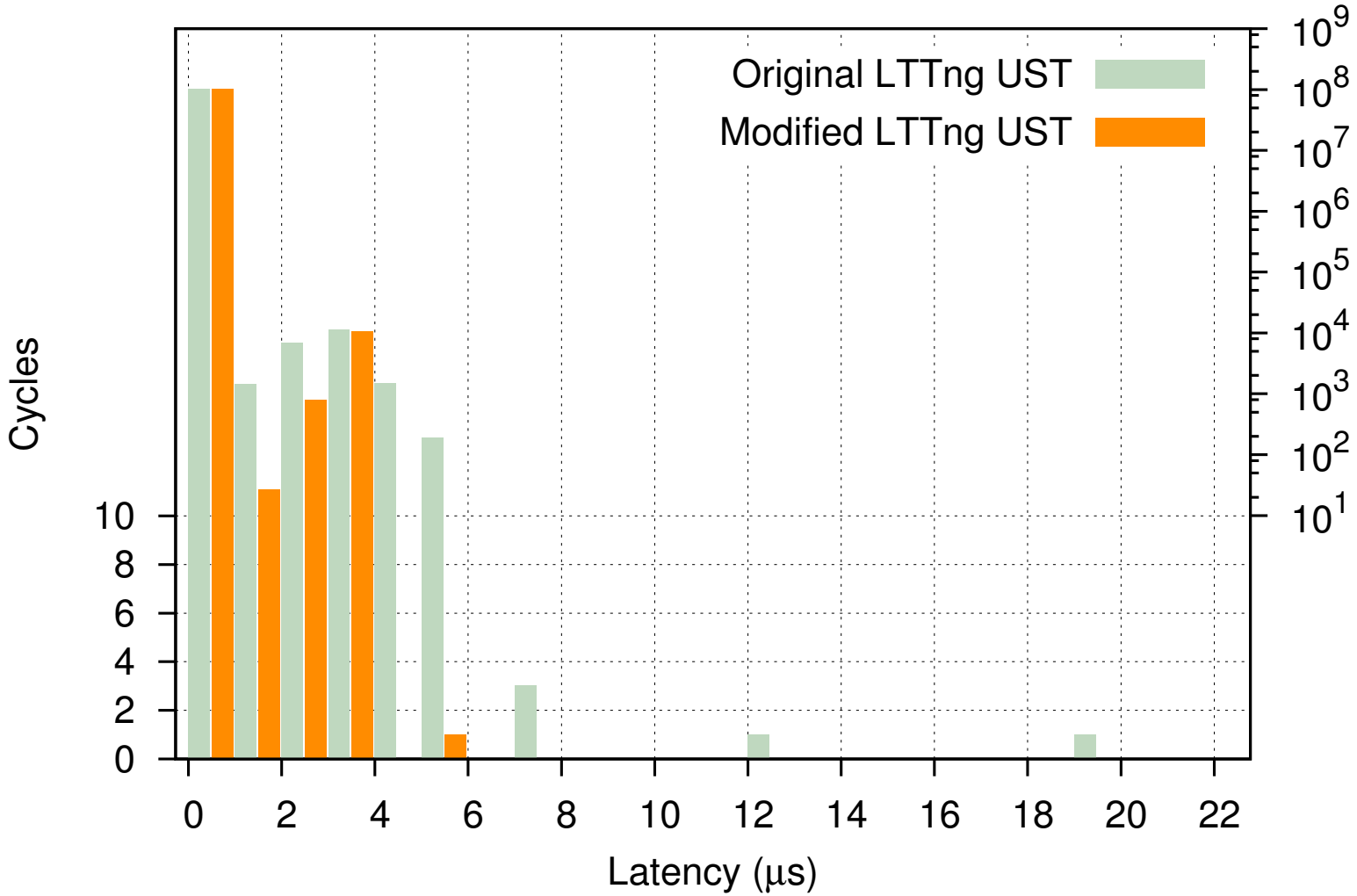
Latency results on the standard kernel

Histogram generated by NPT for 10^8 cycles with original and modified LTTng-UST



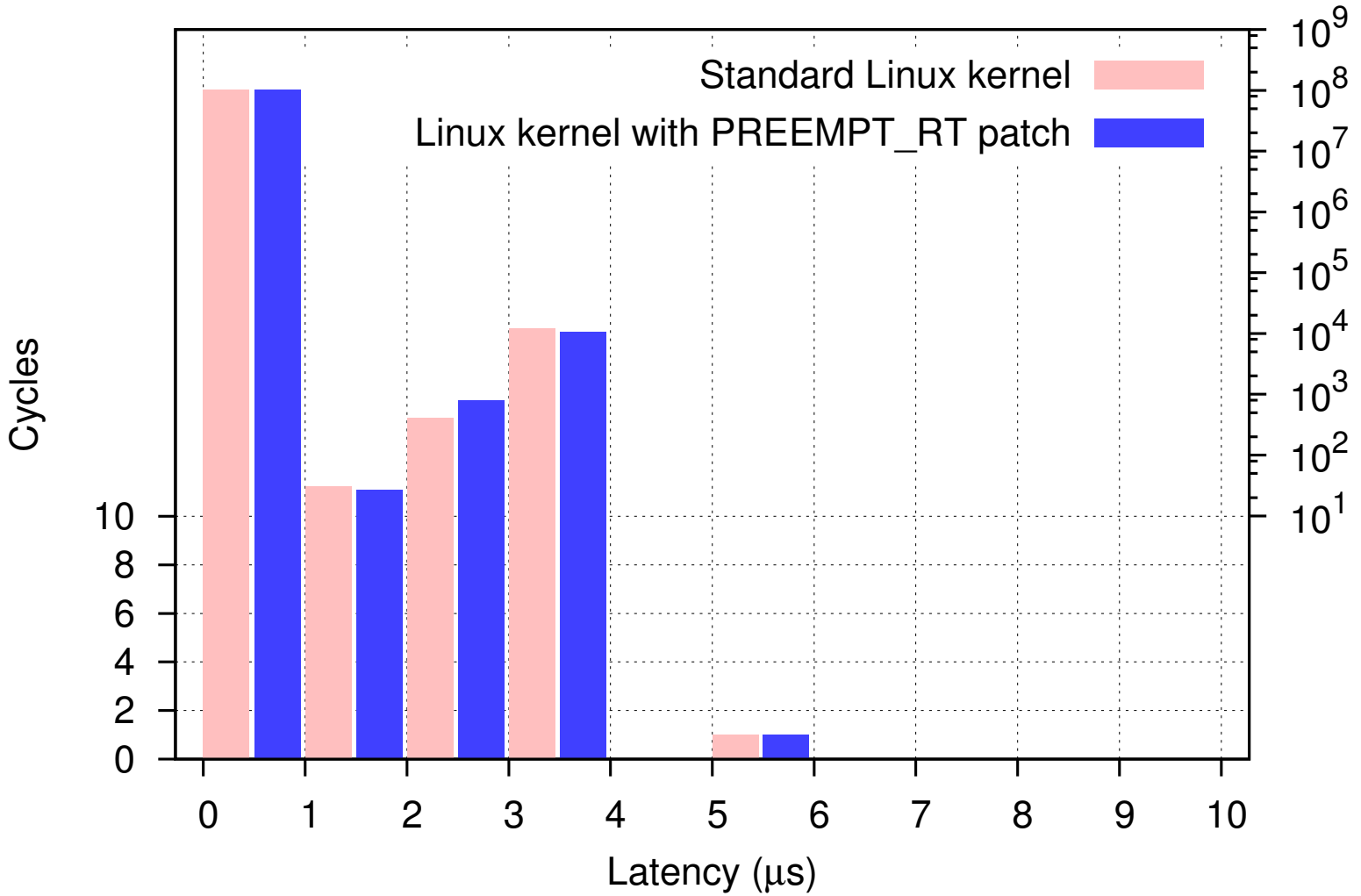
Latency results on the RT kernel

Histogram generated by NPT for 10^8 cycles with original and modified LTTng-UST



Latency results with modified LTTng-UST

Histograms generated by NPT for 10^8 cycles on standard and RT kernels



Numeric comparison

Statistics per cycles, in nanoseconds, generated by NPT on both standard and RT kernels for both the original and modified versions of LTTng-UST

	Latencies in *ns*			
Kernel	standard		RT	
LTTng	original	modified	original	modified
Minimum	287	198	289	197
Mean	317	220	322	206
Maximum	121 744	5 847	19 837	5 088
Variance	74.778	1.186	1.813	1.027
Deviation	273	34.44	42.58	32.05

Conclusion

- Non-Preempt Test tool
- Effects of LTTng tracing on both standard and RT kernels
- Modified LTTng according to our observations
- Latency is currently as low as 5 μs on both standard and PREEMPT_RT patched kernels

Future Work

- Continue LTTng-UST latency hunt
- Integrate our changes in the main branch
- Study the real-time behavior in non shielded environments
- Clarify the process of CPU isolation with respect to per-CPU kernel tasks such as RCU reclamation, timer update, ...

Thank you. Any question ?

LTTng `www.lttng.org`
mailing list: `lttng-dev@lttng.org`

NPT: `git.dorsal.polymtl.ca/?p=npt.git`

Slides: `www.dorsal.polymtl.ca/~rbeamonte/dorsal-pm-dec2012.pdf`