Hardware-assisted software tracing

Adrien Vergé

DORSAL lab
Computer Engineering Departement
December 11, 2013

POLYTECHNIQUE
MONTRÉAL

LE GÉNIE
EN PREMIÈRE CLASSE

Introduction

Software tracing has non-zero side-effects:

- interrupts
- system events
  - cache flushing, etc.
- may add some latency

Hardware tracing:

- observes the system: almost zero overhead
- provides dedicated resources (buffers...)

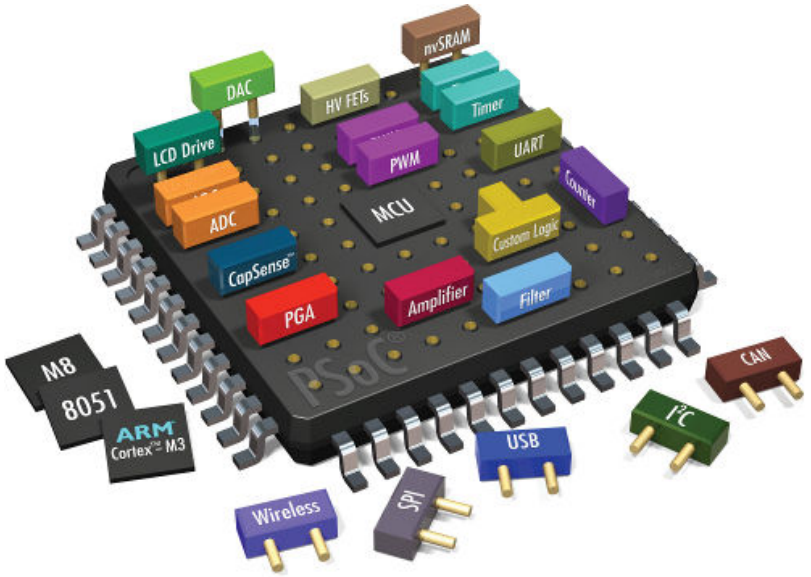
Hardware-assisted tracing

Use of hardware components

- on chip

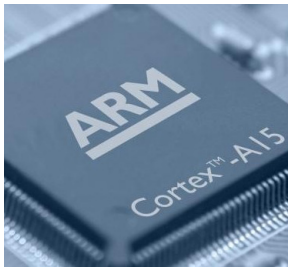
Advantages:

- dedicated circuit for tracing
- very detailed info
  - from the source
- customizable events



source: arm.com

Studied platforms

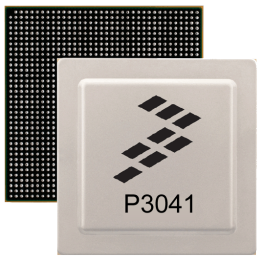


source: arm.com

ARM (*Coresight*)

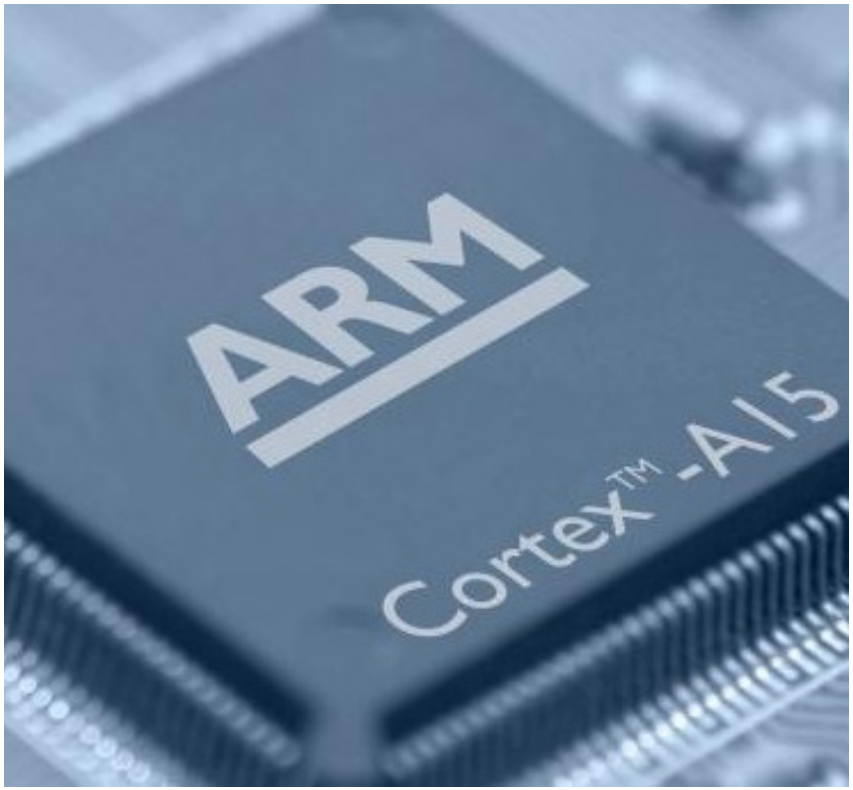
source: intel.com

Intel (*BTS*)

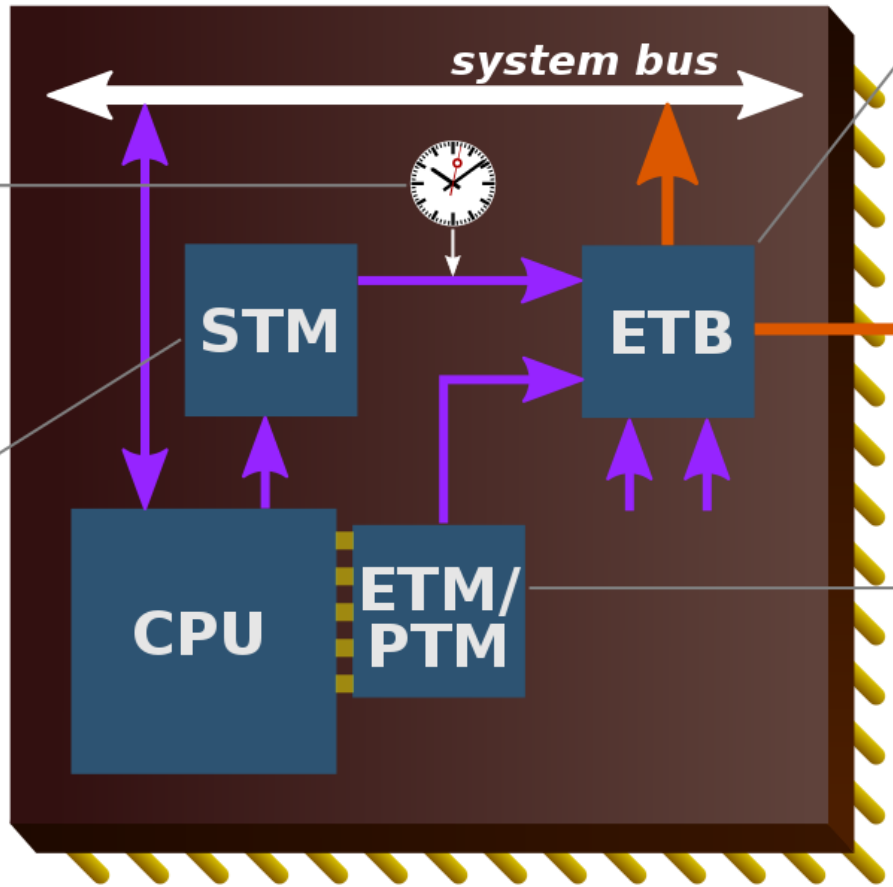


source: freescale.com

Freescale
(*Performance Monitor*)


ARM
Cortex™-A15

Coresight and LTTng



automatic timestamping

cycle-precise

System Trace Macrocell

stores software events

many "channels"

Embedded Trace Buffer

stores trace generated by different sources

retrieved on-chip or off-chip

JTAG

Program Trace Macrocell

stores execution path

(executed instructions sequence)

Coresight: STM

System Trace Macrocell

- dedicated hardware to store software events
- timestamped
  - correlated with other events
  - cycle-precise
- multisource trace in a single stream
  - optimized bandwidth

Coresight: ETM/PTM

Embedded/Program Trace Macrocell

- monitors the core's internal busses
  - no burden on performance
- hardware triggers
  - start tracing only when needed
- hardware filters
  - output only what needed
- data compression
  - complete sequence of executed instructions: approx. 1 bit / cycle

Coresight: ETB

Embedded Trace Buffer

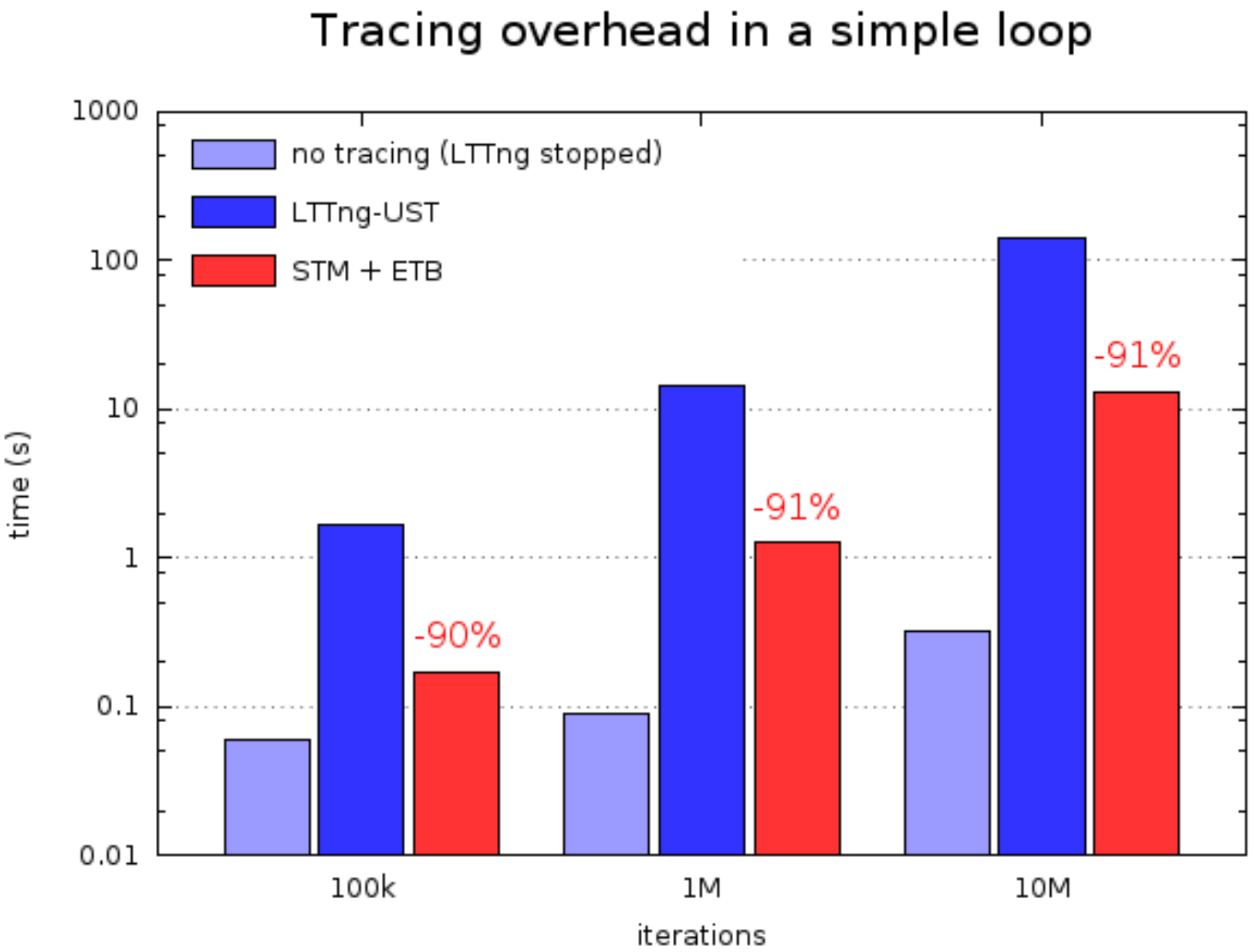
- dedicated buffer (on Pandaboard: 8 kiB)
- gives time to defer tracing
- multiplex data from different sources
- on-chip and off-chip connection (JTAG)

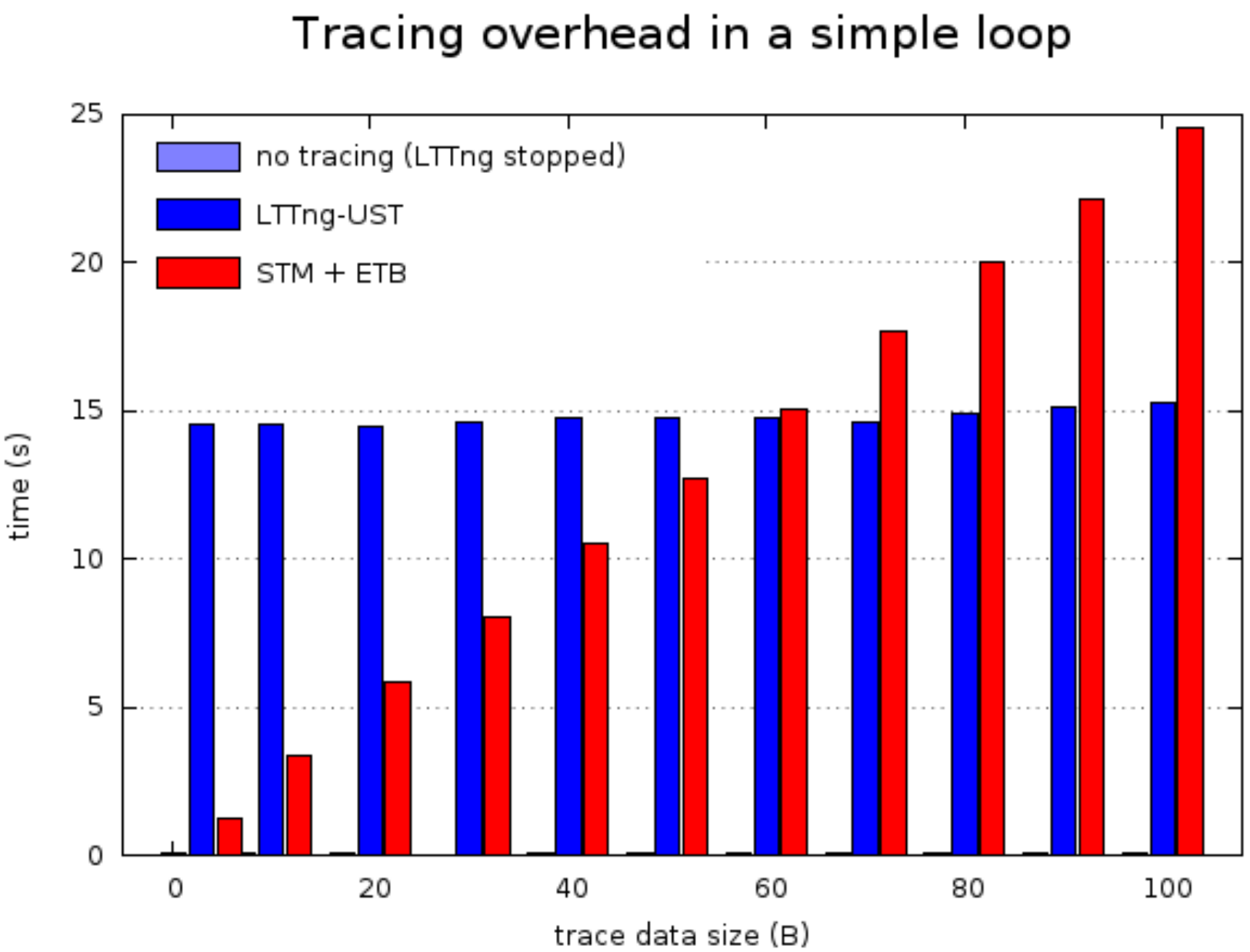
STM + ETB: results

Compare to LTTng-UST

```
for (i = 0; i < 1000000; i++)
     tracepoint(int, smallvalue);
```

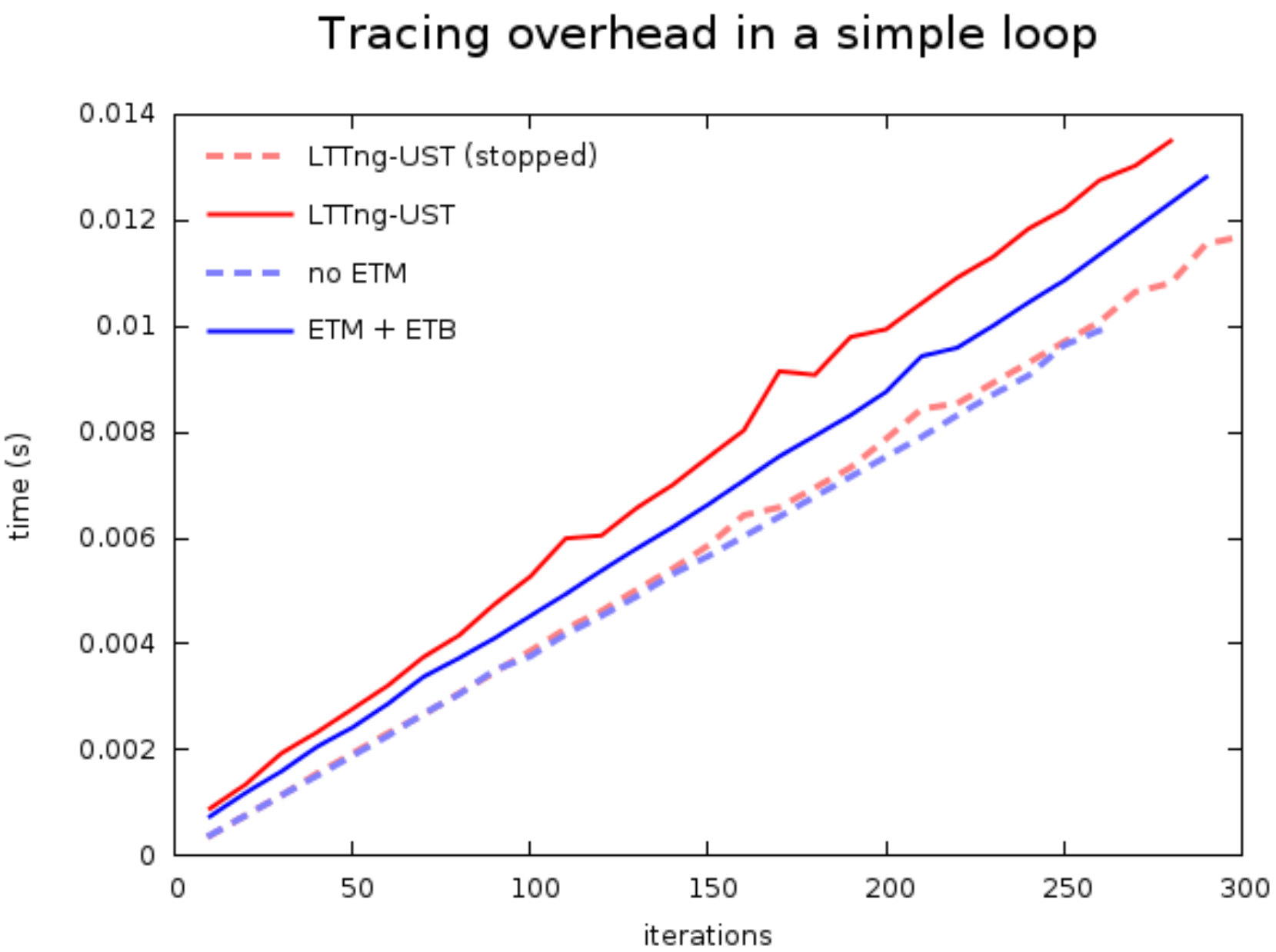
```
for (i = 0; i < 1000000; i++)
     tracepoint(string, "a longer message");
```


Tracing overhead in a simple loop
1000
100
10
1
0.1
0.01
time (s)
no tracing (LTTng stopped)
LTTng-UST
STM + ETB
-90%
-91%
-91%
100k
1M
10M
iterations


Tracing overhead in a simple loop
no tracing (LTTng stopped)
LTTng-UST
STM + ETB
time (s)
trace data size (B)
0
5
10
15
20
25
0
20
40
60
80
100

ETM + ETB: results

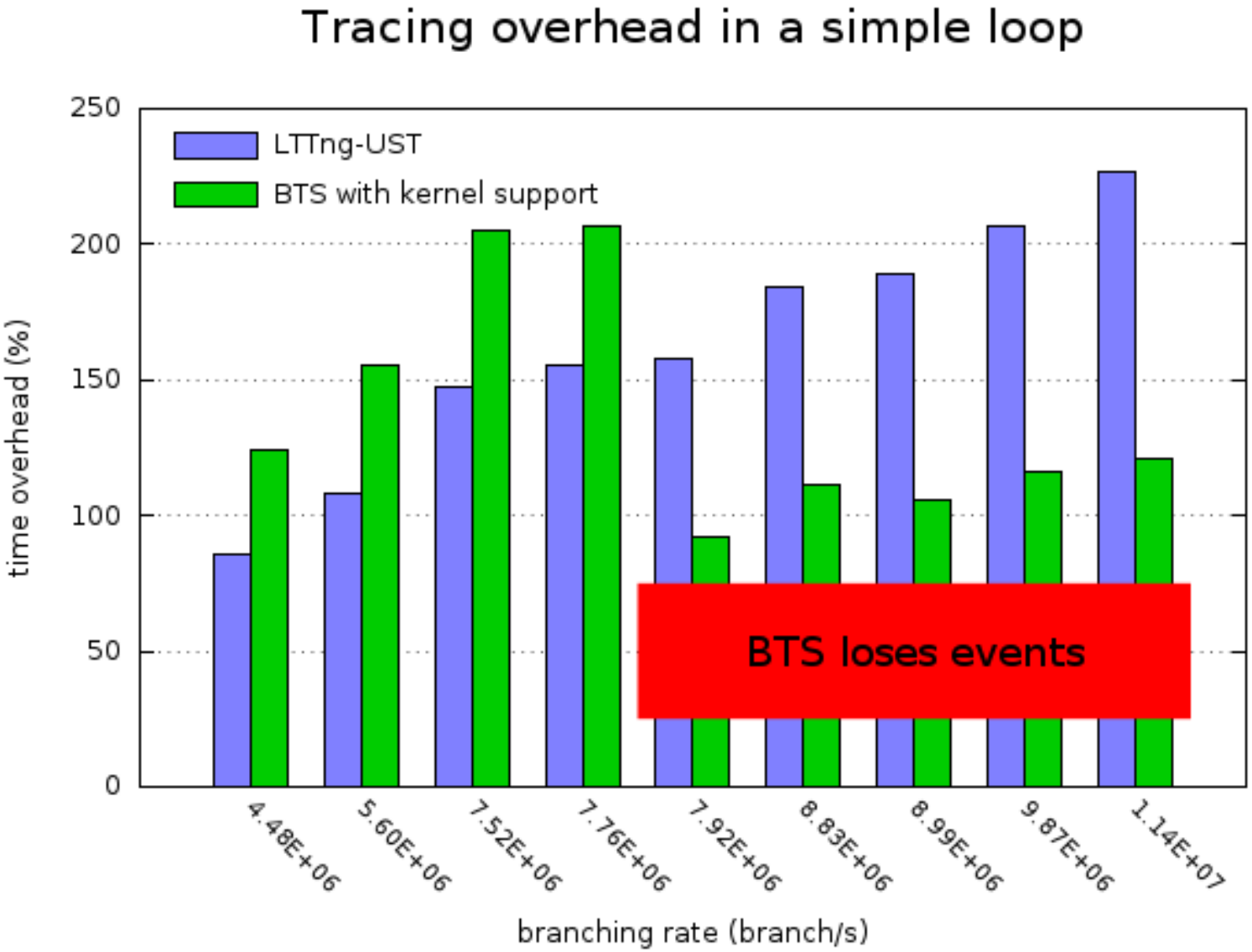
- equivalent in LTTng: empty tracepoint
  - cannot embed custom data
- extra info from hardware
- requires info on the program symbols
  - to set address matching trigger
- ETB: 512 bytes
  For now, benchmark on small loops
  - avoid overflow


Tracing overhead in a simple loop
LTTng-UST (stopped)
LTTng-UST
no ETM
ETM + ETB
time (s)
iterations
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
0
50
100
150
200
250
300


intel
Xeon® 7500

Intel: BTS (Branch Trace Store)

- traces executed instructions
- logs EVERY branch
- stores everything in RAM
  - 24 bytes / branch
  - intense bus usage
- need to drain to disk
  - intense disk usage



md5sum:
7.4E+06

sha256sum:
3.70E+07

gcc:
6.70E+08

Intel: Processor Trace

2014?

2015?

- traces executed instructions
- triggering, filtering and compression capabilities
  - similar to Coresight ETM
- not yet on silicon
- already generators and decoders
  - open-source

Freescale

Freescale: Performance Monitor

Freescale processors

- PowerPC architecture

Performance Monitor

- collection of counters
- hundreds of events
  - cache misses, type of instructions, branches...

Nexus module

- program trace
- data trace
- taken interrupts...

Conclusion

ARM	Intel	Freescale
lightweight tracepoints STM + ETB ✓	*branch tracing* BTS ✗	work in progress...
lightweight branch tracing ETM + ETB ✓	*lightweight and configurable branch tracing* **to come...**	

Questions?