Can mainstream tracing meet embedded needs?

A view of debugging and performance measurement from an embedded perspective. An overview of needs, constraints, problems and solutions.

Frank Rowand, Sony Network Entertainment

August 30, 2012

120827_1747

Why I am here...

“It would be great to also have feedback from Sony users, would it be possible?”

What is embedded?

From a Sony perspective...


Carl Zeiss
SONY
Cyber-shot
2,6-6,3/4,5-22,5
16.2
MEGA PIXELS


SONY
Euphoric
James Labrie
Static Impulse


SONY
α
77


Recently played
2012
Winterland
10:10


SONY
10528


SONY
HD
AVCHD
HANDYCAM
HDR-CX210
5.3 MEGA PIXELS
Carl Zeiss
Vario-Tessar


SON
XCA
CXD900


SONY
Carl Zeiss
Cyber-shot
16.2
MEGA PIXELS
2.6-6.3/4.5-22.5


SONY


MVeX
00:00:00
Miranda Kaleido-X
NFL
NETWORK

$170

SMP processor, 100's Mhz

small RAM for Linux
(application gets priority)

$65,000

many processors

2.5 TB / hour

What is embedded?

From a Sony perspective...

- There is obviously not one answer
- Many different product categories
- Range of devices in each category
- Many different product teams

The resulting tracing needs will thus not be unified, and are likely to be contradictory!

Contradiction

I need both:

- feature X
- feature !X

Contradiction Example

A product development team is strongly adverse to rebuilding kernel (very costly process).

A sufficiently non-invasive performance tool used by the Sony distro team requires kernel rebuilds to alter measurement options.

==> That performance tool is not very useful for the product team, but is required by the distro team to shave microseconds off latency to meet the requirements of the product team.

Can mainstream tracing meet embedded needs?

- There is not one set of needs and requirements

- There are many disparate, conflicting needs

What is embedded?

From another perspective...

Some Common ELC topics

- power management
- memory usage
- boot time
- bring up

These items are consistent with the common Sony embedded concerns.

Translated to goals

- minimize power usage (max battery run time, min energy cost)
- minimize memory usage (min cost, size, power)
- minimize boot time (usability, min power)
- constraint: minimize hardware & software cost
  - low processor frequency
  - small memory size
  - sometimes real-time
  - sometimes time to market

Needs

- Data Collection
- Data Analysis
- Data Visualization

Needs

- Performance Measurement, Tuning & Analysis
- Performance Debugging
- Hardware Debugging
- Kernel Debugging
- Application Debugging

Environment

- mostly during development
    - the focus of this talk

- different needs at deployment time
    - used by product support team, not end user

What is Measured (examples)

- latency
- locks
- code path execution cost
- memory usage
- cause and effect
- scheduling
- network
- I/O
- bus traffic

Latency

- interrupts off
- preemption disabled
- task switch
- lock contention
- overall (eg cyclictest)

Code Execution Cost

- instruction count
- cycle count
- I-cache, D-cache statistics
  - hit
  - miss
  - stall cycles

scheduling

- task switch cost
- task switch latency
- process migration

Debugging

- events leading up to the failure
- state of system at failure

Power Usage

- wake ups
- wake up sources
- power state transitions
- related latencies

Types of Data

- event traces

- aggregate and accumulated data
  - count
  - minimum
  - average
  - maximum
  - percentiles
  - variance, standard deviation
  - histogram
  - other creative metrics and relationship data

Environment

- Linux kernel with UNIX-like userspace
- Linux kernel with Android userspace
- Linux kernel with dual userspace

Environment

- processor architectures:
  - ARM
  - MIPS
  - x86
  - PowerPC

Environment

- system architectures:
  - big endian
  - little endian
  - uniprocessor
  - SMP
  - big.LITTLE (anticipated)
    (will aggregate data reflect processor type?)
  - DSP

Environment

Development system typically different architecture than target system.

aka. cross development

Problems

Need to collect data from early in boot process

- from beginning of kernel boot
- from reset, including bootloader

Problems

Memory constrained

- the camera example is not the smallest memory system of interest
- trace duration typically limited
- mixed stack size feature (4k vs 8k) to reduce memory usage
- one product uses memplug to move memory between Linux and DSP as needed

Problems

Overhead

- Heisenberg, aka observer effect
- cpu
- cache
- memory
- bus traffic

Problems

Overhead Examples

- Even custom lite tracing often needs to be invoked in reduced functionality mode to avoid causing the application to miss deadlines, which would cause the product to not function.

- Product team funded process migration trace tool because use of existing trace points was too expensive.

Problems

Some product development teams reluctant to rebuild the kernel and application then reload on the device due to long cycle time.

- Acceptable to reboot with updated kernel command line.

- Acceptable to enable and disable tools via /proc, etc

Solutions - mainline / community

- kernelshark
- ltt
- ftrace
- perf
- /proc/whatever (eg lock_stat)
- miscellaneous

Solutions - in house

- lite latency tracer
- code path tracer
- process migration tracer
- bus monitor
- jtag trace

Solutions - data analysis

- custom analysis tools
- custom graphing tools
- spread sheet programs
- kernelshark
- ltt
- perf

Solutions

No single answer.

Use the right tool for the right job.

Can mainstream tracing meet embedded needs?

An anecdotal answer:

Some of my conference talks have examples of my use of tracing and performance tools.

My ELC 2008 talk

First 90 slides show useful data and problem solving, partially using existing tools.

Next 9 slides show a problem with existing tools.

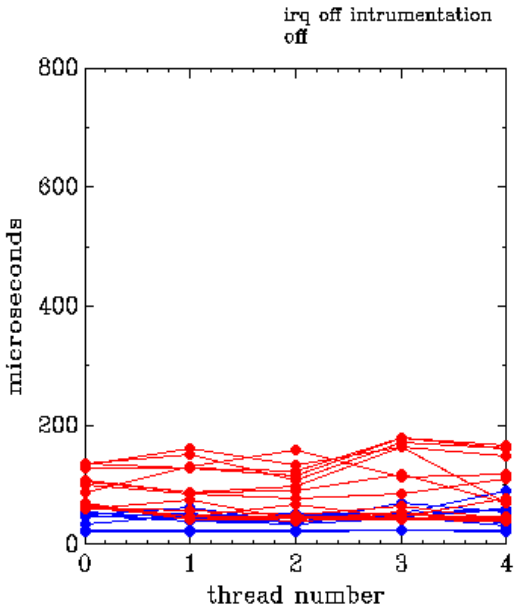
The next slide is tx49 cyclictest latency data (blue is average, red is max), for three cases:

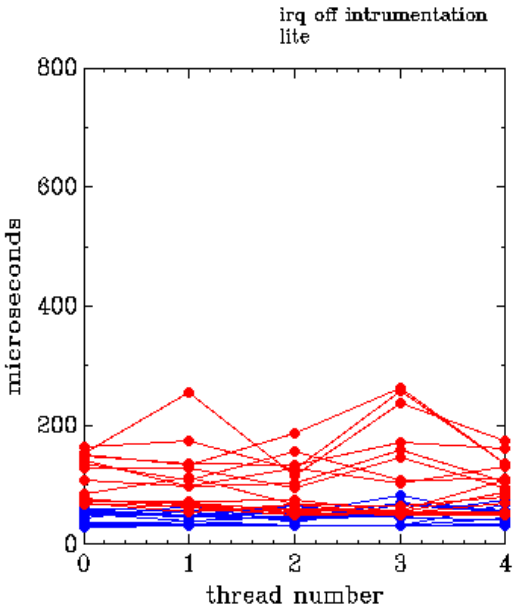
- no latency tracing enabled
- trace-lite enable
- preempt-rt patch latency tracing enabled

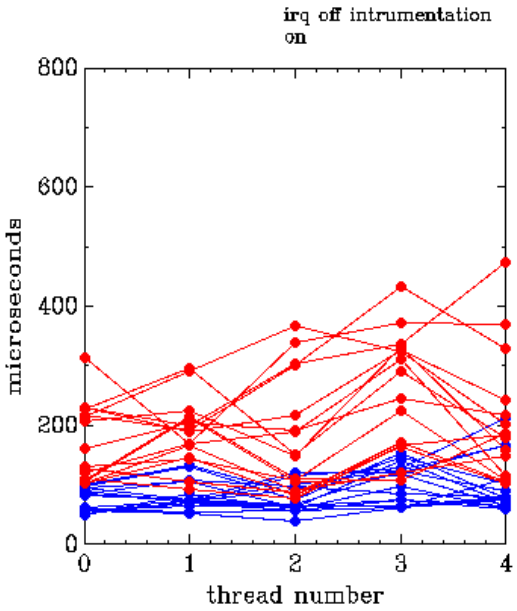
The data shows the large performance impact of enabling latency tracing.

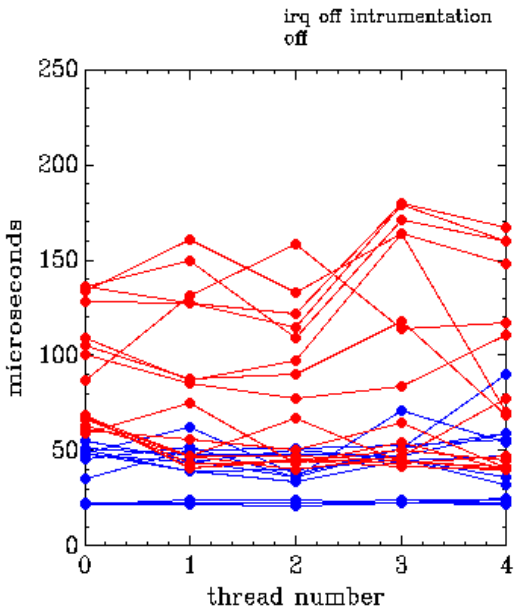
Each individual graph contains multiple lines, where each line is the result of a single test run.

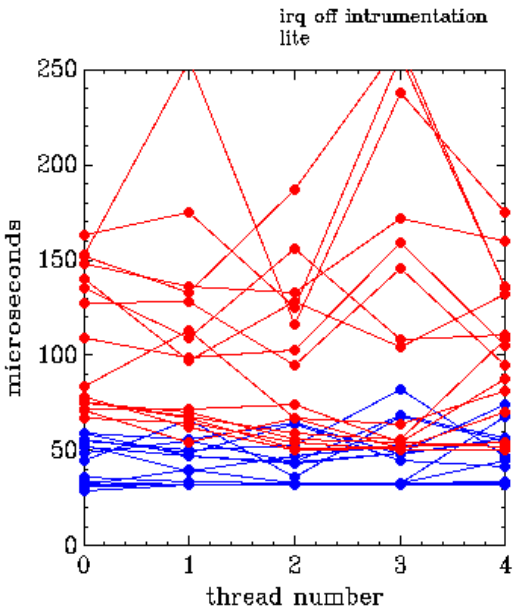
½ of the tests have no “ls” background load.

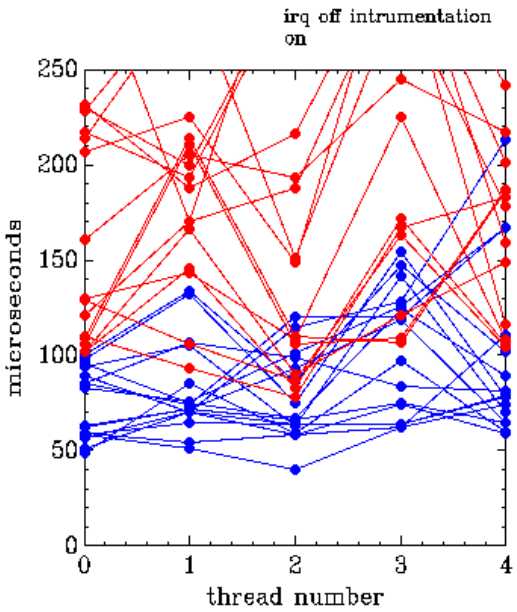

irq off intrumentation
off
microseconds
thread number
0
200
400
600
800
0
1
2
3
4


irq off intrumentation
lite
microseconds
thread number
0
200
400
600
800
0
1
2
3
4


irq off intrumentation
on
microseconds
thread number
0
200
400
600
800
0
1
2
3
4


irq off intrumentation
off
microseconds
thread number
0
50
100
150
200
250
0
1
2
3
4


irq off intrumentation
lite
microseconds
thread number
0
50
100
150
200
250
0
1
2
3
4


irq off intrumentation
on
microseconds
thread number
0
50
100
150
200
250
0
1
2
3
4

The next slides are tx49 interrupts disabled time for two cases:

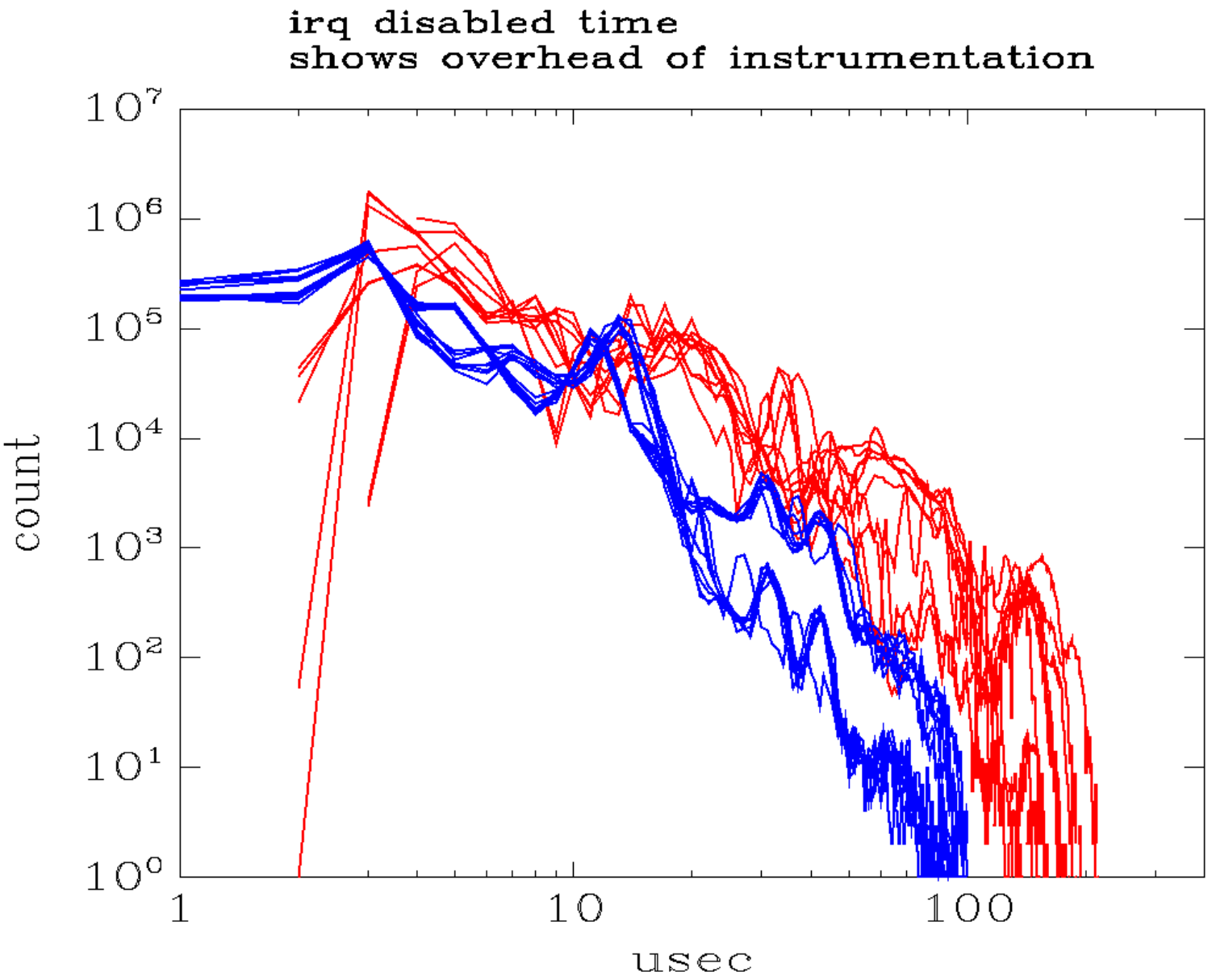
red:

preempt-rt patch latency tracing enabled

blue:

trace-lite enabled

The data again shows the large performance impact of enabling latency tracing.


irq disabled time
shows overhead of instrumentation
count
usec
10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^0
1
10
100

My ELCE 2008 talk

A somewhat jaded conclusion...

What Does It All Mean?

Frank's Law of Performance Tools

The performance metric that you need to answer the current question

- is not available from any existing source or tool

- or is not presented in a meaningful manner

You will need to write a new tool or leverage an existing tool.

My LinuxCon Japan 2011 talk

Tools that I found useful:

- cyclictest
- perf sched
- perf PMU
- perf trace
- ftrace
- KernelShark
- ltt
- /proc/lock_stat
- hwlat_detector

My ELC 2011 talk

ftrace function graph was useful

My ELC 2011 talk

Most required data not available from existing tools.

Added PMU data collection to lite latency tracer.

PMU data collection for specific code paths.

May have been able to leverage perf for PMU data but did not try because it was easy to leverage lite tracer.

Enhanced /proc/lock_stat
- filter (measure only during specific code paths)
- add histogram data

Can mainstream tracing meet embedded needs?

Anecdotal conclusion:

Sometimes: yes

Often: no