C-state latency measurement infrastructure

Linux Plumbers Conference 2019

Artem Bityutskiy,
Intel Finland
Agenda

- Introduction
- What and how measured
- Why useful
- Visual
- Current implementation
- Challenges
Introduction

• Measure **C-state** latency
• Wake Up Latency Tracer (WULT)
• Plan:
  • Open source (Github, GPL)
  • Kernel drivers → upstream
Goal

Get early feedback on WULT kernel drivers

- How do we transform them into something upstreamable
Agenda

- Introduction
- **What and how measured**
- Why useful
- Visual
- Current implementation
- Challenges
1. Arm delayed IRQ
2. Request C-state

Arm delayed IRQ \hspace{1cm} \texttt{mwait} \hspace{1cm} \text{Launch Time}
3. IRQ happens

Arm delayed IRQ  \textit{mwait}  Launch Time

\begin{itemize}
\item IRQ
\end{itemize}
4. Exiting C-state

Arm delayed IRQ    \text{mwait}    \text{Launch Time}    \text{Time After Idle}

\text{IRQ}
4. C-state exit latency

C-state Exit Latency = Time After Idle – Launch Time
Agenda

- Introduction
- What and how measured
- Why useful
- Visual
- Current implementation
- Challenges
Why useful?

• Improve Linux PM QoS subsystem
  • Improve intel_idle driver

• Analyze the HW platform
  • No need for expensive lab equipment
Agenda

- Introduction
- What and how measured
- Why useful
- Visual
- Current implementation
- Challenges
Agenda

- Introduction
- What and how measured
- Why useful
- Visual
- Current implementation
  - Delayed interrupts
  - High level design
- Challenges
Delayed interrupts

- Intel I210 PCIe NIC
- On-board 8ns resolution clock
- Host can read the clock
- Host can arm delayed interrupt
Why not timers?

- Intel: P-unit is aware of timers
  - Pre-wakes and removes latency
- May not be the case for other platforms
- External IRQ source is required
Agenda

- Introduction
- What and how measured
- Why useful
- Visual
- Current implementation
  - Delayed interrupts
  - **High level design**
- Challenges
Main kernel components

wult.ko

API

wult_igb.ko

I210 NIC
Standard driver

wult.ko

API

wult_igb.ko ↔ I210 NIC

unbind

igb
Armer

wult.ko

Armer kthread
- Pick delay
- Arm
- Wait
- Repeat

API

wult_igb.ko

I210 NIC
**Debugfs**

- **Armer kthread**
  - Pick delay
  - Arm
  - Wait
  - Repeat

- **Debugfs**
  - enable
  - delay_min
  - delay_max
  - … etc …

---

**API**

- **wult.ko**
  - Armer kthread
  - Debugfs

- **wult_igb.ko**

- **I210 NIC**
Zoom out

```
API

wult.ko

  Debugfs

  Armer

wult_igb.ko

<=>

I210 NIC
```
Idle flow

- wult.ko
- debugfs
- Armer

API

- wult_igb.ko
- I210 NIC

Idle task

- cpuidle
  - target_state->enter()

- intel_idle

CPU0
- mwait <state>
Interposer

- wult.ko
- wult_igb.ko
- Debugfs
- Armer
- Interposer

API

- I210 NIC

Idle task

- cpuidle
- target_state->enter()

intel_idle

CPU0

- mwait <state>
Interposed idle flow

- **wult.ko**
- **Debugfs**
- **Armer**
- API
- **wult_igb.ko**
- **Interposer**
- **Idle task**
- **cpuidle**
- **target_state->enter()**
- **intel_idle**
- **CPU0**
- **mwait <state>**
- **I210 NIC**
Interposer details

- **Interposer**
  - Get data before idle
  - `target_state->enter()`
  - Check wake reason
  - Get data after
  - `trace_printk()`

- **API**
  - `intel_idle`
    - CPU0
      - `mwait <state>`
    - `cpuidle`
      - `target_state->enter()`

- **Debugfs**
- **Armer**
- **wult.ko**
- **wult_igb.ko**
- **I210 NIC**
Comments?

- `wult.ko`
- `Debugfs`
- `Armer`
- `Interposer`
- `API`
- `wult_igb.ko`
- `I210 NIC`
- `Idle task`
- `cpuidle`
- `target_state->enter()`
- `intel_idle`
- `CPU0`
- `mwalt <state>`
Agenda

- Introduction
- What and how measured
- Why useful
- Visual
- Current implementation
- Challenges
  - The placement
  - C-state statistics
  - User-space interfaces
The placement

• Where in the kernel tree this stuff belongs to?
  • Proposal: drivers/idle/wult
Agenda

- Introduction
- What and how measured
- Why useful
- Visual
- Current implementation
- Challenges
  - The placement
  - Measurement data
  - User-space interfaces
Measurement data

- Data sent to userspace includes:
  - Launch Time
  - Time before and after idle
  - Delta TSC / APERF / MPERF
  - C-state counters delta:
    - CC1, CC1E, CC3, CC6, CC7
    - PC2, PC3, PC6, PC7, PC8, PC9, PC10
Challenges

- Architecture-dependent
- C-states are platform/model dependent
  - E.g., servers have only few
- Today: use 0 for non-existing C-states
- Optimal: more C-state awareness in Kernel
  - Turbostat needs this too
Agenda

- Introduction
- What and how measured
- Why useful
- Visual
- Current implementation
- Challenges
  - The placement
  - Measurement data
- User-space interfaces
Debugfs

• Today: debugfs for configuring
• Upstream
  • Sysfs ?
  • /sys/class/wult ?
trace_printk

- **Today:** `trace_printk()`: /sys/kernel/debug/tracing/trace
- **Upstream**
  - Plan: use dynamic tracepoints
  - Tom Zanussi’s patches
  - Hook to existing tracepoints
    - In `cpuidle.c` around ‘->enter()’
Thank you!
Backup
1. NIC sends MSI message at Launch Time
2. MSI reaches PCIe root complex
3. Core starts wakes up at Time After Idle

C-state Exit Latency = 1 + 2 + 3 = Time After Idle – Launch Time