« UNDERSTANDING EMBEDDED LINUX BENCHMARKING USING KERNEL TRACE ANALYSIS »

ALEXIS MARTIN
INRIA / LIG / UNIV. GRENOBLE, FRANCE
alexis.martin@inria.fr
We do Need Benchmarking!

- **Benchmark**: a **standard** or point of **reference** against which things may be **compared** or assessed.
  (new Oxford American Dictionary)

- Benchmarking **computer systems**:
  - **Assess** performance in different execution settings
  - **Compare** computer systems

- **Performance** criteria:
  - speed, latency, bandwidth, power consumption, memory used, …

→ **Critical** step in system design
Benchmarking is Challenging

- Benchmarking construction is **difficult**
- There are **many different** benchmarks available
  - 3D rendering, DBMS test, NAS…
- In some cases benchmark is **nonexistent**
- Major motivation for using a benchmark is **popularity**
- The behavior of tests is **not necessarily known**
Understand What We Benchmark

- **Identify** what is measured and how
- **Interpret** results
- Draw a **profile**
- **Compare** different benchmarks

→ **Help** to **chose** the right benchmark
1. **Execute** benchmark application (UDOO+Phoronix)

2. **Record** a trace from this execution (LTTng)

3. **Analyze** the traces (Framesoc + TraceCompass)

4. Draw a **profile** and **compare** benchmarks
Phoronix Test Suite for Benchmarking

- **Phoronix Test Suite (PTS)** is an **open-source** platform ([openbenchmarking.org](http://openbenchmarking.org))
  - It contains **various** tests (over **170**)
  - PTS is **cross-platform** (i686, x86_64, ARM, PowerPC)
  - It includes every **mechanism** for **automated** tests
  - Result **sharing** for statistics and platform **comparisons**

- Tests are classified into **families**:

<table>
<thead>
<tr>
<th>System</th>
<th>Processor</th>
<th>Network</th>
<th>Memory</th>
<th>Graphics</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td># tests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>79</td>
<td>1</td>
<td>2</td>
<td>53</td>
<td>12</td>
</tr>
</tbody>
</table>
Benchmark Selection

• Select 10 tests from 5 different families
• Use « recommended » tests from PTS
  • Calculated from most used tests

- **system**
  - idle, pybench, phpbench

- **processor**
  - scimark2, ffmpeg, compress-gzip

- **network**
  - network-loopback

- **memory**
  - stream, ramspeed

- **disk**
  - dbench
The Test Platform

- **UDOO** development board ([udoo.org](http://udoo.org))
- **i.MX 6 Quad** ARM CPU (A9) @1GHz + 1 coprocessor (Cortex-M3)
- 1GB RAM, WiFi, Gigabit ethernet, HDMI, microSD, SATA
- Touchscreen, camera, GPIO
- **Debian** ARM kernel ([armmp 3.16](http://armmp.org))
Tracing With LTTng

- **LTTng** ([lttng.org](http://lttng.org)) open-source tracing framework:
  - Trace **engine**:
    - **kernel-space**: kprobes & kernel tracepoints
    - **user-space**: user implemented tracepoints
  - **Viewing** and **analyzing**: Trace compass (eclipse)

- Trace only the **kernel** to **avoid** benchmark code **modifications**
Trace Properties

- Number of events (in Million)
  - 0M
  - 1M
  - 10M
  - 100M
  - 1000M

- Duration (in minutes)
  - 0'
  - 4'
  - 8'
  - 12'
  - 16'
  - 20'
  - 24'
  - 28'
  - 32'
  - 36'
  - 40'

- Categories
  - system
  - processor
  - network
  - memory
  - disk
Trace Properties

- Number of events (in Million):
  - 0M
  - 1M
  - 10M
  - 100M
  - 1000M

- Duration (in minutes):
  - 0'
  - 4'
  - 8'
  - 12'
  - 16'
  - 20'
  - 24'
  - 28'
  - 32'
  - 36'

Legend:
- system (red)
- processor (yellow)
- network (green)
- memory (blue)
- disk (purple)

Example data point: 10', 45M
What does the Given Family Mean?

- Phoronix gives us a family *without* explanations
- Families are related to *kernel functionalities*
- **Compute** family:
  - *Biggest number* of events?

→ We want to check if the *announced* family *corresponds* to the *computed* one
Assigning Family to Events

- Memory
- Network
- Disk
- Processor
- System
Assigning Family to Events

- mm_page_alloc
- mm_page_free
- kmem_cache_alloc
- ...

- memory
- processor
- network
- disk
- system

Kernel
Assigning Family to Events

- **memory**:
  - mm_page_alloc
  - mm_page_free
  - kmem_cache Alloc
  - ... 

- **network**:
  - rpc_bind_status
  - sock_rcvqueue_full
  - net_dev_xmit
  - ... 

- **disk**:
  - scsi_eh_wakeup
  - jbd2_commit_locking
  - block_rq_insert
  - ... 

- **processor**:
  - power_cpu_idle
  - timer_init
  - htimer_expire
  - ... 

- **system**:
  - workqueue_activate_work
  - sched_switch
  - rcu_utilization
  - ... 

- **Kernel**

---

Understanding Embedded Linux Benchmarking Using Kernel Trace Analysis - Alexis Martin, ELC 2015
Family Distribution

- System
- Processor
- Network
- Memory
- Disk

Event distribution %

0 10 20 30 40 50 60

System: idle, pybench, phpbench, scimark2, ffmpeg, compress-gzip, network-loopback, stream, ramspeed, dbench
Processor: idle, pybench, phpbench, scimark2, ffmpeg, compress-gzip, network-loopback, stream, ramspeed, dbench
Network: idle, pybench, phpbench, scimark2, ffmpeg, compress-gzip, network-loopback, stream, ramspeed, dbench
Memory: idle, pybench, phpbench, scimark2, ffmpeg, compress-gzip, network-loopback, stream, ramspeed, dbench
Disk: idle, pybench, phpbench, scimark2, ffmpeg, compress-gzip, network-loopback, stream, ramspeed, dbench
Family Distribution

Event distribution %

- idle
- pybench
- pbhbench
- scimark2
- ffmpeg
- compress-gzip
- network-loopback
- stream
- ramspeed
- dbench

System
Processor
Network
Memory
Disk
Family Distribution

- System
- Processor
- Network
- Memory
- Disk

Event distribution %

- idle
- pybench
- phpbench
- scimark2
- ffmpeg
- compress-gzip
- network-loopback
- stream
- ramspeed
- dbench
Family Distribution

Event distribution %

- Red: System
- Yellow: Processor
- Green: Network
- Blue: Memory
- Purple: Disk

ChartData: [image]
Family Distribution

Event distribution %

System Processor Network Memory Disk

Event distribution %

idle pybench phpbench scimark2 ffmpeg compress-gzip network-loopback stream ramspeed dbench
Family Distribution

<table>
<thead>
<tr>
<th>Event distribution %</th>
<th>idle</th>
<th>pybench</th>
<th>phpbench</th>
<th>scimark2</th>
<th>ffmpeg</th>
<th>compress-gzip</th>
<th>network-loopback</th>
<th>stream</th>
<th>ramspeed</th>
<th>dbench</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processor</td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Understanding Embedded Linux Benchmarking Using Kernel Trace Analysis - Alexis Martin, ELC 2015
Family Distribution is not Enough

- **Computed** family = **announced** family?
  - 5 matches over 10

- **Kernel function** is **different** from one to another benchmark
  - No relation between announced and calculated families

- We trace **only kernel** part
  → Check the **distribution** of time during which the kernel is used
Kernel-time vs. User-time

Time spent in kernel mode

Time distribution %

0 25 50 75 100

idle pybench phpbench scimark2 ffmpeg compress-gzip network-loopback stream ramspeed dbench

✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔ ✔
Kernel-time vs. User-time

Time spent in kernel mode

- idle
- pybench
- phpbench
- scimark2
- ffmpeg
- compress-gzip
- network-loopback
- stream
- ramspeed
- dbench

Time distribution %

- 0%
- 25%
- 50%
- 75%
- 100%
Kernel-time vs. User-time

Time spent in kernel mode

<table>
<thead>
<tr>
<th>Application</th>
<th>Time Distribution %</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>2</td>
</tr>
<tr>
<td>pybench</td>
<td>65</td>
</tr>
<tr>
<td>phpbench</td>
<td>65</td>
</tr>
<tr>
<td>scimark2</td>
<td>78</td>
</tr>
<tr>
<td>ffmpeg</td>
<td>63</td>
</tr>
<tr>
<td>compress-gzip</td>
<td>31</td>
</tr>
<tr>
<td>network-loopback</td>
<td>29</td>
</tr>
<tr>
<td>stream</td>
<td>78</td>
</tr>
<tr>
<td>ramspeed</td>
<td>65</td>
</tr>
<tr>
<td>dbench</td>
<td>7</td>
</tr>
</tbody>
</table>
Kernel-time vs. User-time

Time spent in kernel mode

Long time spent in kernel mode → Right computed family

Short time spent in kernel mode → Wrong computed family
Kernel-time vs. User-time

Time spent in kernel mode

<table>
<thead>
<tr>
<th>Program</th>
<th>Time Distribution %</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>2</td>
</tr>
<tr>
<td>pybench</td>
<td>49</td>
</tr>
<tr>
<td>phpbench</td>
<td>48</td>
</tr>
<tr>
<td>scimark2</td>
<td>47</td>
</tr>
<tr>
<td>ffmpeg</td>
<td>63</td>
</tr>
<tr>
<td>compress-gzip</td>
<td>29</td>
</tr>
<tr>
<td>network-loopback</td>
<td>31</td>
</tr>
<tr>
<td>stream</td>
<td>78</td>
</tr>
<tr>
<td>ramspeed</td>
<td>65</td>
</tr>
<tr>
<td>dbench</td>
<td>7</td>
</tr>
</tbody>
</table>

**Long time** spent in kernel mode → **Right** computed family

**Short time** spent in kernel mode → **Wrong** computed family
Do We Observe More Than the Benchmark?

• Big **stack** of programs for **running** those benchmarks:
  ‣ ssh
    ‣ custom bash script
      ‣ LTTng
      ‣ Phoronix
        ‣ Benchmark

• Analyze **overhead induced** by those programs

→ Observe events by **processes**
Time Spent by Processes

- Application
- Phoronix
- LTTng
- Swapper
- Other

Idle: 0%
Pybench: 40%
Phpbench: 10%
Scimark2: 10%
Ffmpeg: 10%
Compress-gzip: 10%
Network-loopback: 10%
Stream: 10%
Ramspeed: 10%
Dbench: 10%

Phoronix ✔
LTTng ✔
Swapper ✔
Other ✔
Time Spent by Processes

- Application
- Phoronix
- LTTng
- Swapper
- Other

- Application
- Phoronix
- LTTng
- Swapper
- Other

idle
pybench
phpbench
scimark2
ffmpeg
compress-gzip
network-loopback
stream
ramspeed
dbench
Time Spent by Processes

- Application
- Phoronix
- LTTng
- Swapper
- Other

<table>
<thead>
<tr>
<th>Application</th>
<th>Phoronix</th>
<th>LTTng</th>
<th>Swapper</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>pybench</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>phpbench</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>scimark2</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>ffmpeg</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>compress-gzip</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>network-loopback</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>stream</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>ramspeed</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>dbench</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
</tbody>
</table>

Time distribution %
Time distribution %

- Application
- Phoronix
- LTTng
- Swapper
- Other

Swapper = \textit{idle}

Phoronix: \textit{low} intrusion
Event Distribution by Processes

Event distribution %

- Application
- Phoronix
- LTTng
- Swapper
- Other

Application
Phoronix
LTTng
Swapper
Other
LTTng produces a huge number of events.
Analysis of LTTng Overhead

• Not easy to get only events from the benchmark
  • Names depend on benchmark
  • Some benchmarks are not only a single program
    • several instances of the same program
    • network-loopback = cat + dd + netcat
• Overhead comes mainly from LTTng
• LTTng overhead is easy to remove from trace
  • Get events from process by name and extract it
  ➔ Overhead removed, we observe only the benchmark
Understanding Embedded Linux Benchmarking Using Kernel Trace Analysis

- Alexis Martin, ELC 2015
Conclusion

• Benchmark results:
  • Better understanding of benchmarking programs
    • Profile the kernel use (families, duration)
    • What can impact the performance
  • Most used benchmarks on phoronix are very different
    • Different profiles for similar tests

• Intrusiveness of used tools:
  • Phoronix is not intrusive
  • LTTng produces many kernel events
    • Constant profile (memory + disk)
    • We know how to remove this overhead for the analysis

→ Generic way to analyze benchmarks
Acknowledgment

• This work was done and funded within the SoC-TRACE project (link)
  • French ministry of industry
  • Inria, UJF, STMicroelectronics, ProbaYes

• Framesoc tool is an outcome of this project (soctrace-inria.github.io/framesoc/)
  • Framework for the management and analysis of traces
Thank You!