« UNDERSTANDING EMBEDDED LINUX BENCHMARKING USING KERNEL TRACE ANALYSIS »

ALEXIS MARTIN
INRIA / LIG / UNIV. GRENOBLE, FRANCE
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**:
  - **Asses** 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
Work Summary

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)
  - 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></th>
<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>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

<table>
<thead>
<tr>
<th>System</th>
<th>tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>idle, pybench, phpbench</td>
</tr>
<tr>
<td>Processor</td>
<td>scimark2, ffmpeg, compress-gzip</td>
</tr>
<tr>
<td>Network</td>
<td>network-loopback</td>
</tr>
<tr>
<td>Memory</td>
<td>stream, ramspeed</td>
</tr>
<tr>
<td>Disk</td>
<td>dbench</td>
</tr>
</tbody>
</table>
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.3.16))
Tracing With LTTng

- **LTTng** (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)

Duration (in minutes)

- system
- processor
- network
- memory
- disk

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

Kernel
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
Family Distribution

- System
- Processor
- Network
- Memory
- Disk

Event distribution %

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

System Distribution: [Graph showing distribution across different benchmarks and system components]
Family Distribution

Event distribution %

- System
- Processor
- Network
- Memory
- Disk

0 10 20 30 40 50 60

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

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

Event distribution %

- System
- Processor
- Network
- Memory
- Disk

Event distribution %

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

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

![Family Distribution Diagram]

- **System**
- **Processor**
- **Network**
- **Memory**
- **Disk**

Event distribution %

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

0 25 50 75 100

Time distribution %

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

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

Time spent in kernel mode

<table>
<thead>
<tr>
<th>Task</th>
<th>Time Spent in Kernel Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>100%</td>
</tr>
<tr>
<td>pybench</td>
<td>65%</td>
</tr>
<tr>
<td>phpbench</td>
<td>63%</td>
</tr>
<tr>
<td>scimark2</td>
<td>78%</td>
</tr>
<tr>
<td>ffmpeg</td>
<td>63%</td>
</tr>
<tr>
<td>compress-gzip</td>
<td>78%</td>
</tr>
<tr>
<td>network-loopback</td>
<td>65%</td>
</tr>
<tr>
<td>stream</td>
<td>100%</td>
</tr>
<tr>
<td>ramspeed</td>
<td>65%</td>
</tr>
<tr>
<td>dbench</td>
<td>100%</td>
</tr>
</tbody>
</table>
Kernel-time vs. User-time

Time spent in kernel mode

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Time distribution %</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>2</td>
</tr>
<tr>
<td>pybench</td>
<td>15</td>
</tr>
<tr>
<td>phpbench</td>
<td>25</td>
</tr>
<tr>
<td>scimark2</td>
<td>50</td>
</tr>
<tr>
<td>ffmpeg</td>
<td>75</td>
</tr>
<tr>
<td>compress-gzip</td>
<td>78</td>
</tr>
<tr>
<td>network-loopback</td>
<td>85</td>
</tr>
<tr>
<td>stream</td>
<td>90</td>
</tr>
<tr>
<td>ramspeed</td>
<td>95</td>
</tr>
<tr>
<td>dbench</td>
<td>100</td>
</tr>
</tbody>
</table>

- ✔️ indicates a high percentage of time spent in kernel mode.
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

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
Time Spent by Processes

- Application
- Phoronix
- LTTng
- Swapper
- Other

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

✔  ✔  ✔  ✔  ✔
Understanding Embedded Linux Benchmarking Using Kernel Trace Analysis

- Alexis Martin, ELC 2015

**Time Spent by Processes**

- Application
- Phoronix
- LTTng
- Swapper
- Other

**Time distribution %**

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

- ✔
- ✔
- ✔
- ✔
- ✔

17 Understanding Embedded Linux Benchmarking Using Kernel Trace Analysis - Alexis Martin, ELC 2015
Time Spent by Processes

- Application
- Phoronix
- LTTng
- Swapper
- Other

Swapper = **idle**

Phoronix: **low** intrusion
Event Distribution by Processes

- Application
- Phoronix
- LTTng
- Swapper
- Other

Event distribution %

0 20 40 60 80 100

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

Event distribution %

- System
- Processor
- Network
- Memory
- Disk

Event distribution %

- idle
- pybench
- phpbench
- scimark2
- ffmpeg
- compress-gzip
- network-loopback
- stream
- ramspeed
- dbench
Understanding Embedded Linux Benchmarking Using Kernel Trace Analysis

- Alexis Martin, ELC 2015

LTTng Overhead Profile

- Event distribution %
- System
- Processor
- Network
- Memory
- Disk

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

Stable using of kernel events
Real Benchmark Profile

Event distribution %

- System
- Processor
- Network
- Memory
- Disk

Event distribution %

System
Processor
Network
Memory
Disk

idle
pybench
phpbench
scimark2
ffmpeg
compress-gzip
network-loopback
stream
ramspeed
dbench
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** (for long benchmarks)
  • 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!