Multi-core scheduling optimizations for soft real-time applications
a cooperation aware approach

Lucas De Marchi

sponsors:

co-authors:
Liria Matsumoto Sato
Patrick Bellasi
William Fornaciari
Wolfgang Betz
Agenda

• Introduction
  • Motivation
  • Objectives

• Analysis

• Optimization description

• Experimental results

• Conclusions & future works
Introduction – motivation
Introduction – motivation

• SMP + RT
  - Multiple processing units inside a processor
  - Determinism

• Parallel Programming Paradigms
  - Data Level Parallelism (DLP)
    - Competitive tasks
  - Task Level Parallelism (TLP)
    - Cooperative tasks
Introduction – motivation

- DLP
  - T0 → T1 → T2 → T3

- TLP
  - T0 → T1 → T2

Characterization:
- Synchronization
- Communication
Introduction – motivation

- Linux RT scheduler (mainline)
  - Run as soon as possible (based on prio)
    ⇔ Use as many CPUs as possible
  - Ok for DLP!

**But, what about TLP?**
**Anyway, why do we care about TLP?**
Objectives

• Study the behavior of RT Linux scheduler for cooperative tasks
• Optimize the RT scheduler
• Smooth integration into mainline kernel
  ▪ Don't throw away everything
Analysis – benchmark

- Simulation of a scenario where SW replaces HW
- Multimedia-like
- Mixed workload: DLP + TLP
- Challenge: map $N$ tasks to $M$ cores optimally

![Diagram showing a network of nodes and arrows with labels](image-url)
Analysis – metrics

- Throughput
  - Sample mean time
- Determinism
  - Sample variance
Analysis – metrics

• Influencing factors
  • Preemption-disabled
  • IRQ-disabled

• Locality of wake-up

Intel i7 920:
4 cores
2.8 GHz
Analysis – locality of wake-up

• Migrations

   a) migration patterns (wave0, mixer1 and mixer2)

   wave0: 112010101010101010101030111301010101010131010321010033
   wave1: 033333333333333333333313202133333333333303333202332121
   wave2: 02202222222222222222223302222222222222222222213322330
   wave3: 1111010101010101010101011110101010101010101010101010101002
   mixer0: 023033333333333333333333333202333333333333333333322233
   mixer1: 10330220101010101010101010101033010101010101010101010101010101
   mixer2: 2302010101010101010101012020101010101010101010101010101010101021
   monitor: 11112222222222222222222231112222222222222222222233322303
Analysis – locality of wake-up

• Migrations

  b) occasional migrations

  wave0: 302121331102303323303311032320011021111111120101010101
  wave1: 22330301333321201012103212101233321032023033333333333
  wave2: 033332232302003111201211330021322233323330220222222222
  wave3: 110211120211101030031121003030120101101111111010101010
  mixer0: 32221030111032023131030302121312001322032023033333333
  mixer1: 001003321202220131103203212130320331201031033022010101
  mixer2: 230101020201202020121020021010130101201202302010101010
  monitor: 11302213333331313120321333032222323312311111122222222
Analysis – locality of wake-up

- Cache-miss rate measurements

<table>
<thead>
<tr>
<th></th>
<th>1 CPU</th>
<th>2 CPUs</th>
<th>4 CPUs</th>
<th>Increase (1 - 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Load</strong></td>
<td>7.58%</td>
<td>8.99%</td>
<td>9.44%</td>
<td>+24.5%</td>
</tr>
<tr>
<td><strong>Store</strong></td>
<td>9.29%</td>
<td>9.78%</td>
<td>11.62%</td>
<td>+25.1%</td>
</tr>
</tbody>
</table>
Analysis – conclusion

- Why do we care about TLP?
  - Common parallelization technique
- What about TLP?
  - Current state of Linux scheduler is not as good as we want
Solution – benchmark

**Abstraction:**

One application level **sends** data to another
Solution – benchmark

**Abstraction:**
One application level sends data to another

**Reality:**
shared buffers + synchronization

Serialization (task cooperation)
Parallelization (task competition)
Solution – benchmark

• **Abstraction:**
  One application level sends data to another

• **Reality:**
  shared buffers + synchronization

• **Dependencies:**
  Define dependencies among tasks in the opposite way of data flow
If data do not go to tasks, then the tasks go to where data were produced.

Make tasks run on the same CPU of their dependencies.
Measurement tools

- Ftrace
- sched_switch tool (Carsten Emde)*
- gtkwave
- perf
- adhoc scripts

* http://www.osadl.org/Single-View.111+M5d51b7830c8.0.html
Solution – task-affinity

Dependency followers

Task-affinity:

Selection of the CPU in which a task (e.g. m0) executes takes into consideration the CPUs in which its dependencies (e.g. w0 and w1) ran last time.
**Solution – task-affinity implementation**

- 2 lists inside each `task_struct`:
  - `taskaffinity_list`
  - `followme_list`

- 2 system calls to add/delete affinities:
  - `sched_add_taskaffinity`
  - `sched_del_taskaffinity`
Experimental results

Cache-miss rates

- Measurements without and with task-affinity

More data/instruction hit the L1 cache

less cache lines are invalidated
Experimental results

What exactly to evaluate?

• Cache-miss rate is not exactly what we want to optimize

• Optimization objectives:
  • Lower the time to produce a single sample
  • Increase determinism on production of several samples
Experimental results
Average execution time of each task

![Graph showing execution times for wave0, wave1, wave2, wave3, mixer0, mixer1, mixer2.](image)
Experimental results
Variance of execution time of each task

---

**Task-affinity**

**Mainline**

<table>
<thead>
<tr>
<th>Task</th>
<th>Variance (µs)²</th>
</tr>
</thead>
<tbody>
<tr>
<td>w0</td>
<td>0.22</td>
</tr>
<tr>
<td>w1</td>
<td>0.31</td>
</tr>
<tr>
<td>w2</td>
<td>0.29</td>
</tr>
<tr>
<td>w3</td>
<td></td>
</tr>
<tr>
<td>m0</td>
<td>0.36</td>
</tr>
<tr>
<td>m1</td>
<td>0.83</td>
</tr>
<tr>
<td>m2</td>
<td>1.45</td>
</tr>
<tr>
<td>mixer0</td>
<td>0.61</td>
</tr>
<tr>
<td>mixer1</td>
<td>2.19</td>
</tr>
<tr>
<td>mixer2</td>
<td>0.47</td>
</tr>
</tbody>
</table>
Experimental results
Production time of each single sample

- Results obtained for 150,000 samples
Experimental results
Production time of each single sample

- Empiric repartition function
- Real-time metric (normal distribution):
  - average + 2 * standard deviation
mainline
Max: 114
Min: 29
Avg: 37.8
Variance: 3.22

task-affinity
Max: 51
Min: 35
Avg: 38.0
Variance: 0.21
## Experimental results

### Summary

<table>
<thead>
<tr>
<th></th>
<th>Average</th>
<th>Variance</th>
<th>Real-time Metrics</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>mainline</td>
<td>37.826</td>
<td>3.225</td>
<td>41.42</td>
<td>~15x</td>
</tr>
<tr>
<td>taskaffinity</td>
<td>38.038</td>
<td>0.214</td>
<td>38.96</td>
<td>5.94%</td>
</tr>
</tbody>
</table>

\~15x
Conclusion & future works

- Average execution time is almost the same
- Determinism for real-time applications is improved

Future works:
- Better focus on temporal locality
- Improve task-affinity configuration
- Test on other architectures
- Clean up the repository
Conclusion & future works

- Still a Work In Progress
- Git repository:
  - git://git.politreco.com/linux-lcs.git
- Contact:
  - lucas.demarchi@profusion.mobi
  - lucas.de.mar.chi@gmail.com
Q & A
Solution – Linux scheduler
Dependency followers

- **Linux scheduler**: Change the decision process of the CPU in which a task executes when it is woken up.

![Diagram of Linux scheduler]

- **Core Scheduler**
  - Main scheduler
  - Periodic scheduler

- **Scheduler classes**
  - RT
  - Fair
  - Idle

- **Scheduler policies**
  - RR
  - FIFO
  - Normal
  - Idle
  - Batch