Multi-core scheduling optimizations for soft real-time applications

a cooperation aware approach

Lucas De Marchi

sponsors:

co-authors:

Liria Matsumoto Sato

Patrick Bellasi

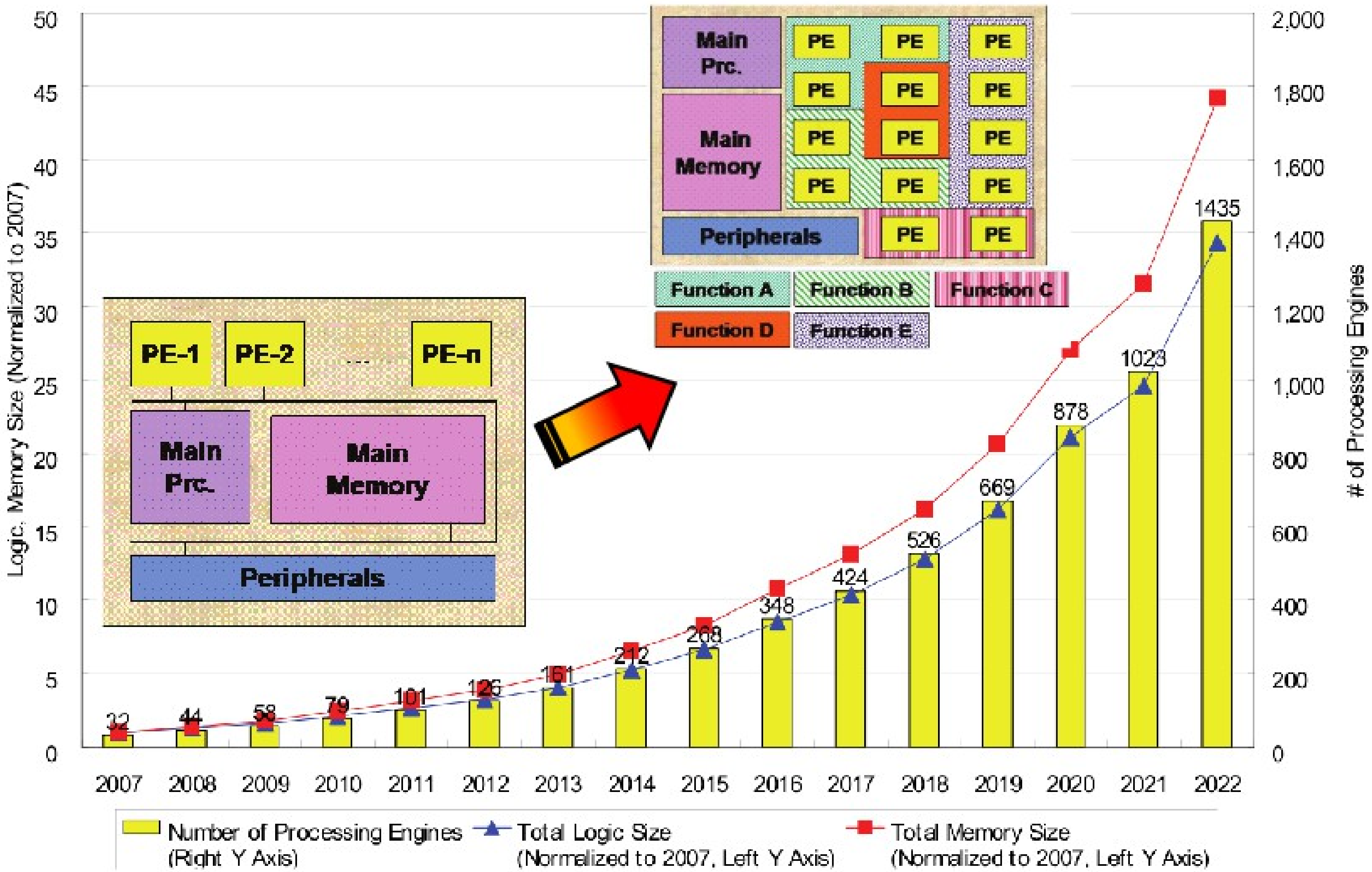
William Fornaciari

Wolfgang Betz

Agenda

- Introduction
    - Motivation
    - Objectives
- Analysis
- Optimization description
- Experimental results
- Conclusions & future works

Introduction – motivation

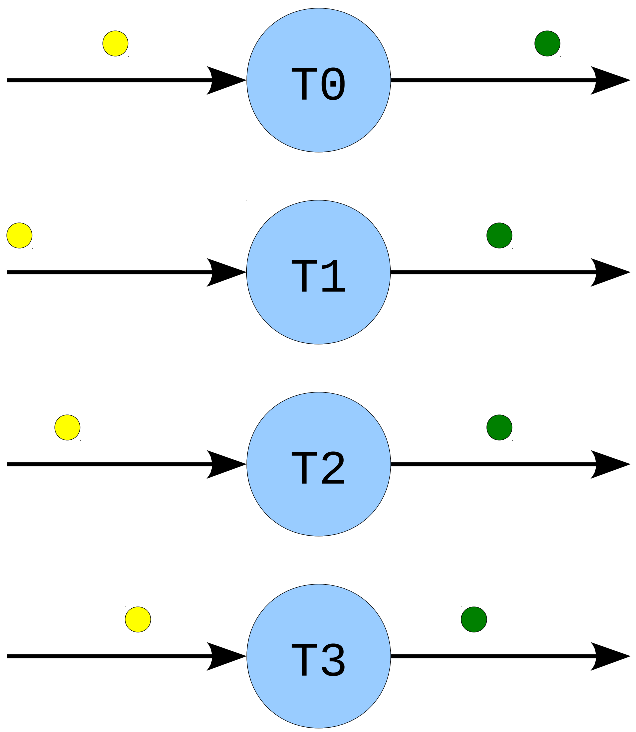


Introduction – motivation

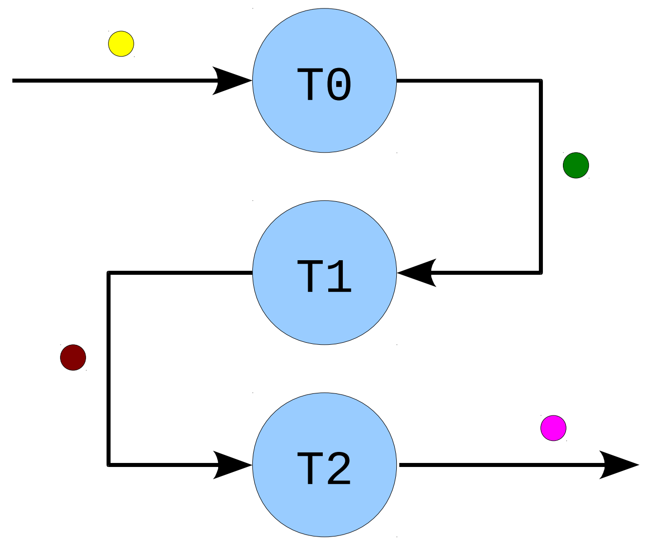
- SMP + RT
  - Multiple processing units inside a processor
  - Determinism
- Parallel Programming Paradigms
  - Data Level Parallelism (DLP)
    - Competitive tasks
  - Task Level Parallelism (TLP)
    - Cooperative tasks

Introduction – motivation

- DLP



- TLP



- 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?

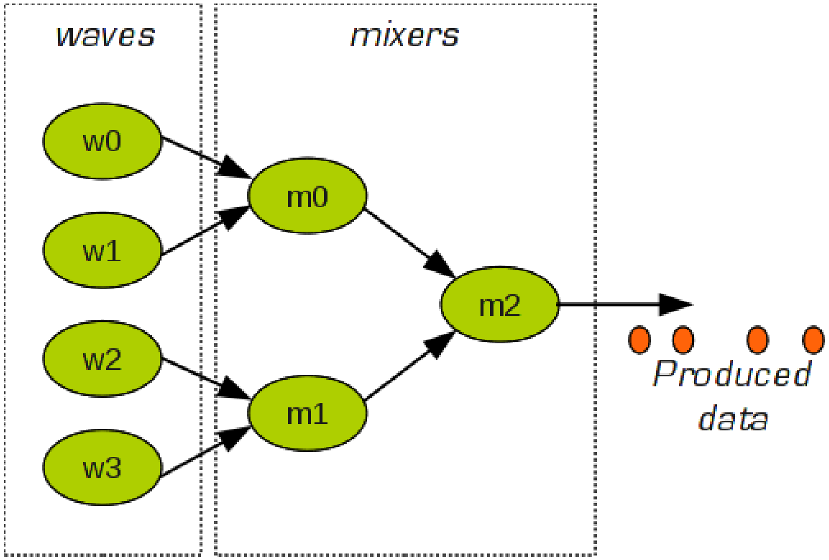
Anyway, why do we care about TLP?

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

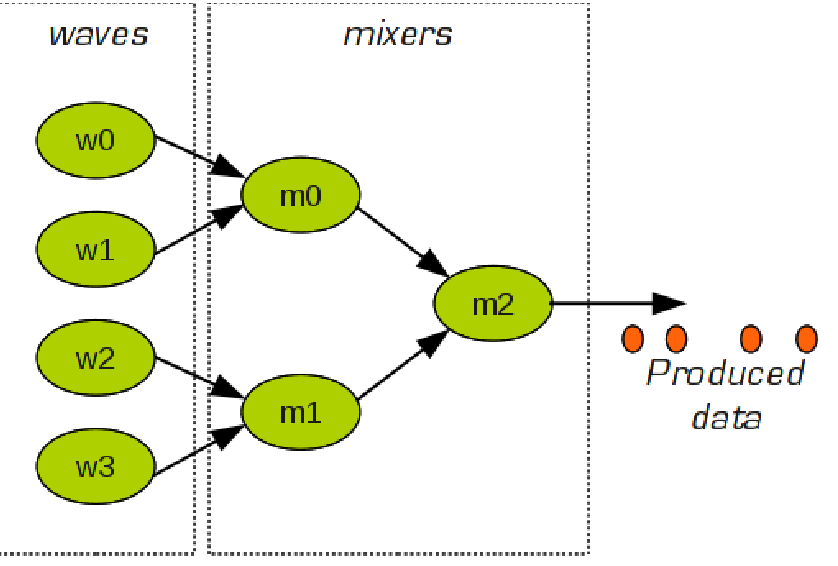
Analysis – benchmark

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



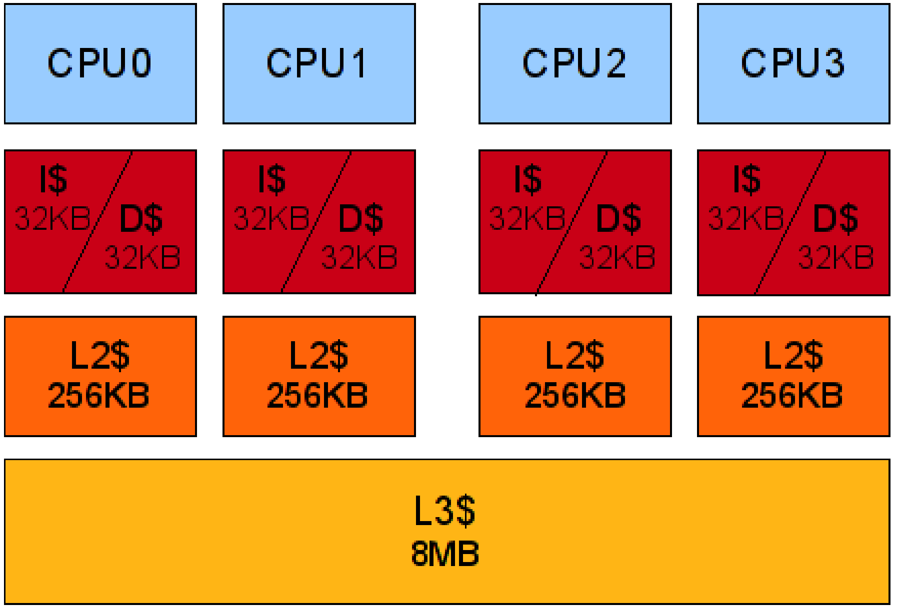
Analysis – metrics

- 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: 022022222222222222222222233302222222222222222213322330
   wave3: 111101010101010101010101011110101010101010101010101002
  mixer0: 023033333333333333333333333202333333333333333333322233
  mixer1: 103302201010101010101010101033010101010101010101010101
  mixer2: 230201010101010101010101012020101010101010101010101021
 monitor: 111122222222222222222222223111222222222222222223332230 3
```

Analysis – locality of wake-up

- Migrations

b) occasional migrations

```
   wave0: 3021213311023033233033110323200110211111112010101010101
   wave1: 2233030133332120101210321210123332103202303333333333333
   wave2: 0333322323020031112012113300213222333233302202222222222
   wave3: 1102111202111010300311210030301201011011111111010101010
  mixer0: 3222103011103202313103030212131200132203202303333333333
  mixer1: 0010033212022201311032032121303203312010310330220101010
  mixer2: 2301010202012020201210200210101301012012023020101010101
 monitor: 1130221333331313120321333032222323312311111112222222222
```

Analysis – locality of wake-up

- Cache-miss rate measurements

	1 CPU	2 CPUs	4 CPUs	Increase (1 - 4)
Load	7.58%	8.99%	9.44%	**+24.5%**
Store	9.29%	9.78%	11.62%	**+25.1%**

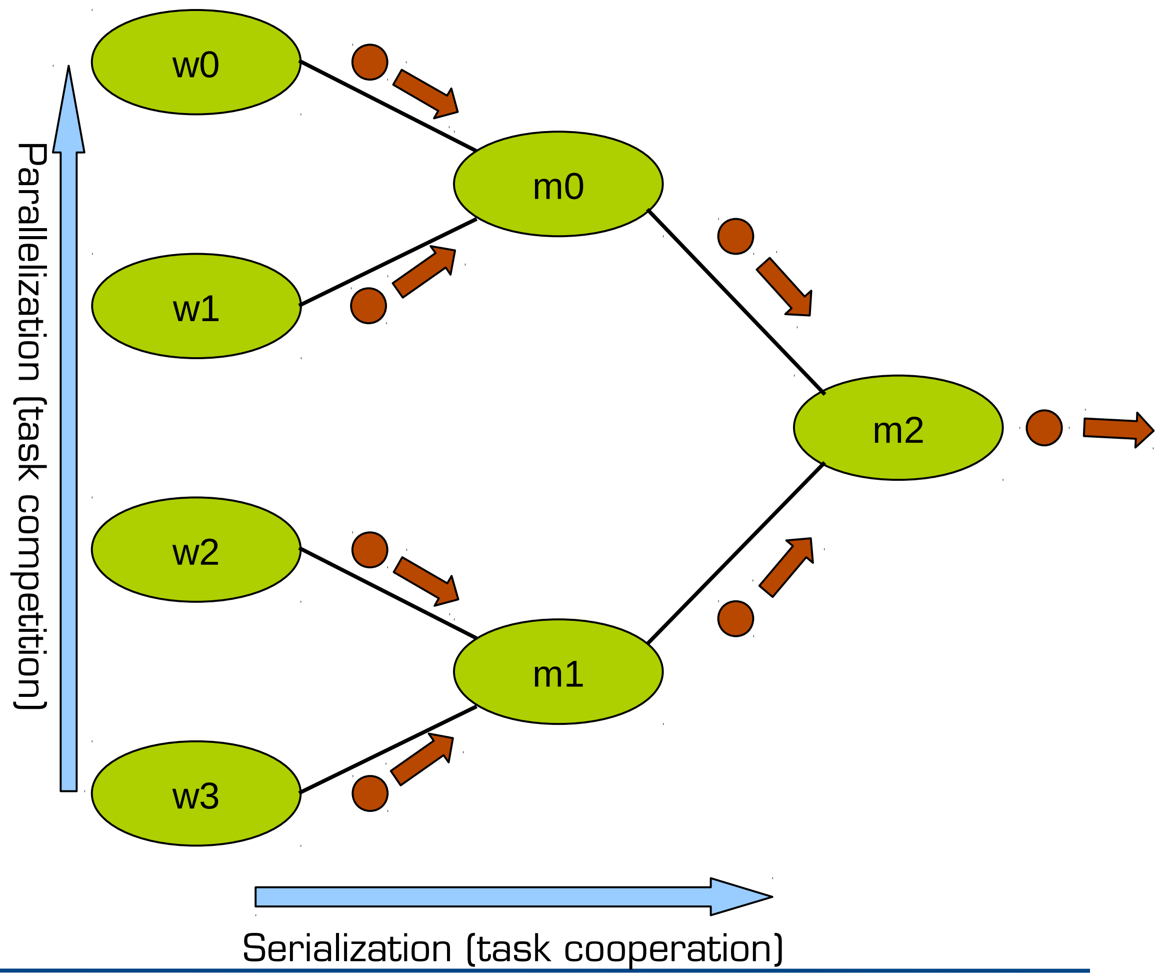
Analysis – conclusion

- 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

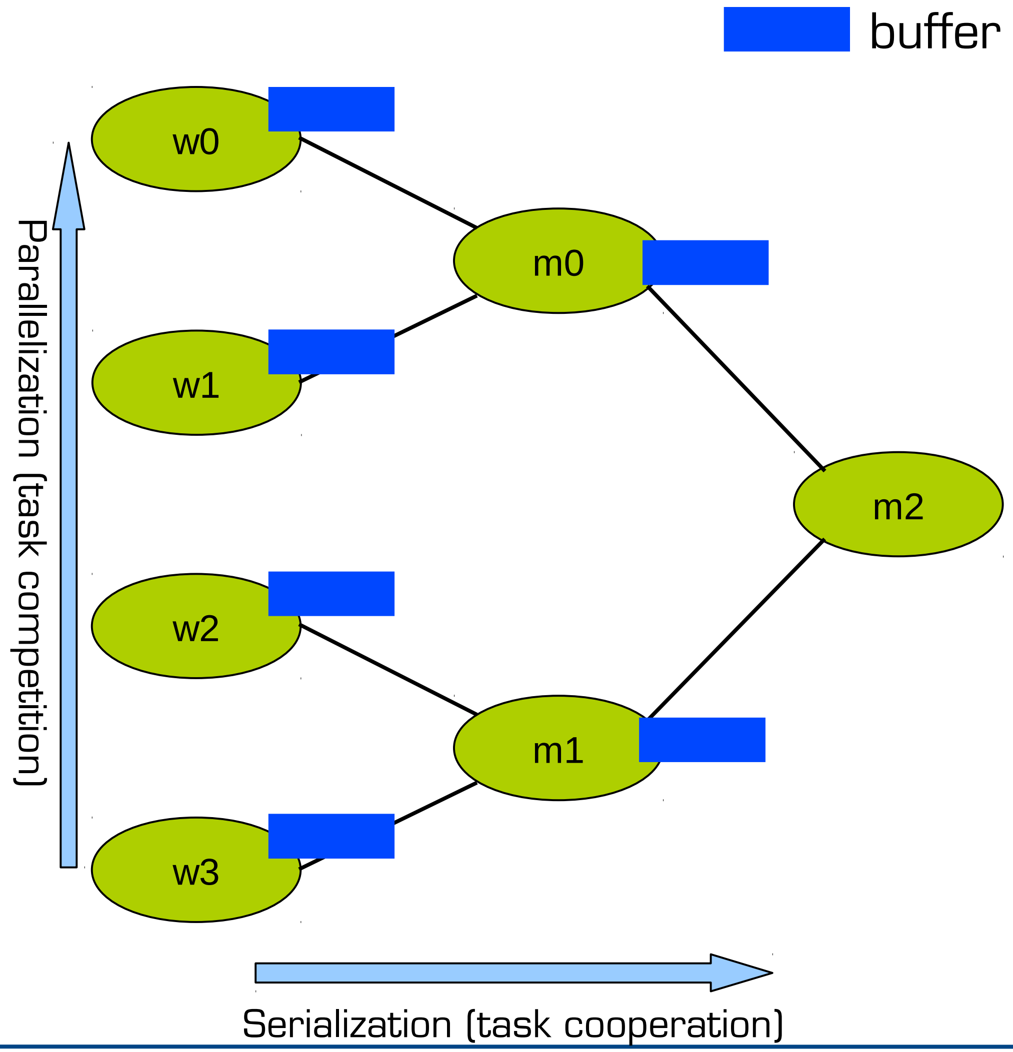
- **Abstraction:**

  One application level **sends** data to another



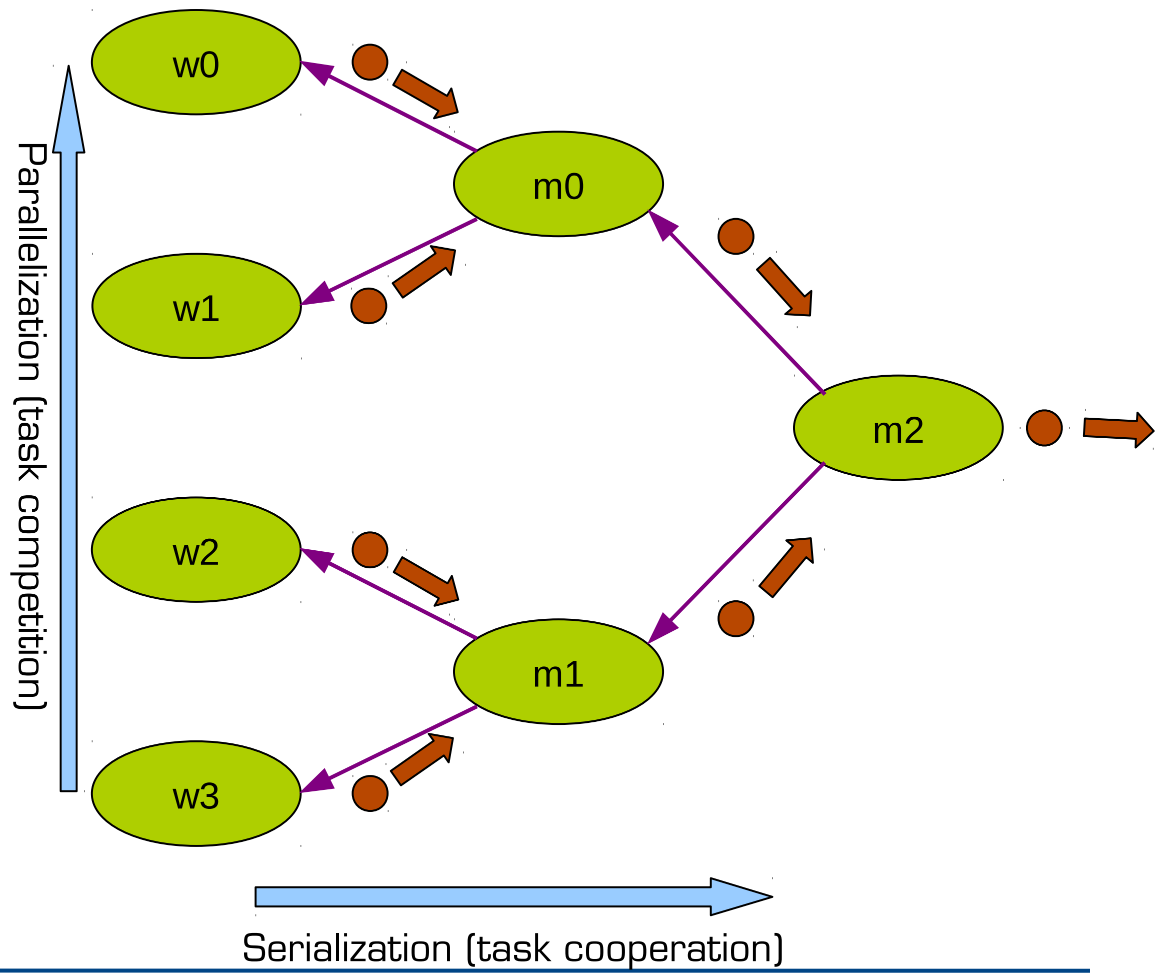
Solution – benchmark

- **Abstraction:**
  One application level **sends** data to another
- **Reality:**
  shared buffers + synchronization



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



Solution – idea

Dependency followers

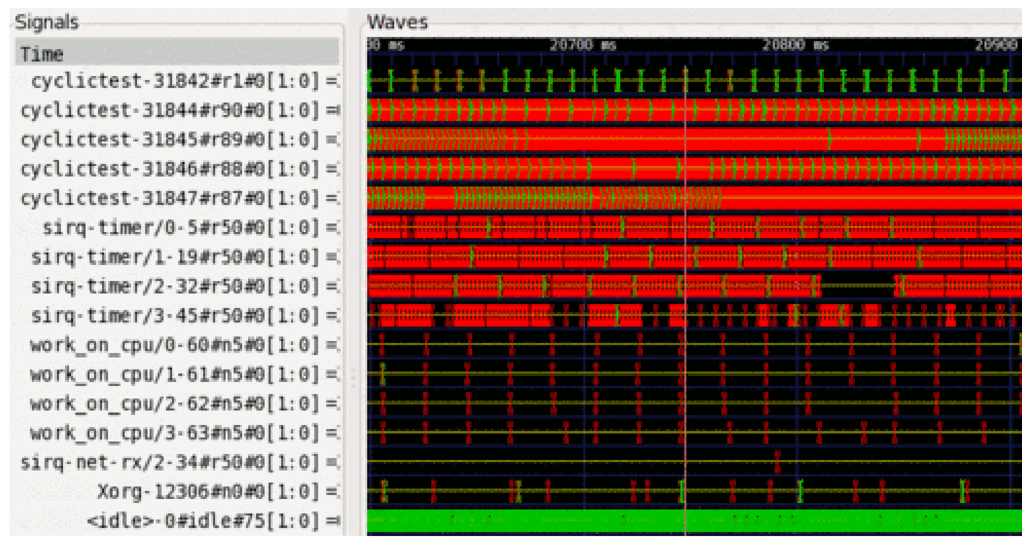
If data do not go to tasks,

then the tasks go to where data were produced

Make tasks run on same CPU of their dependencies

Measurement tools

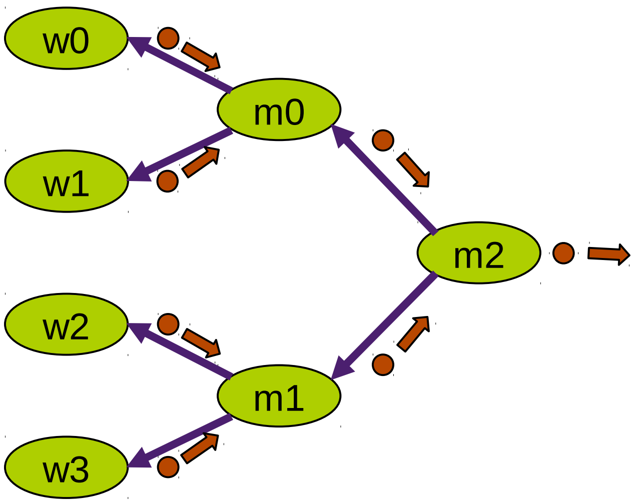
- Ftrace
- sched_switch tool (Carsten Emde)*
- gtkwave
- perf
- adhoc scripts



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

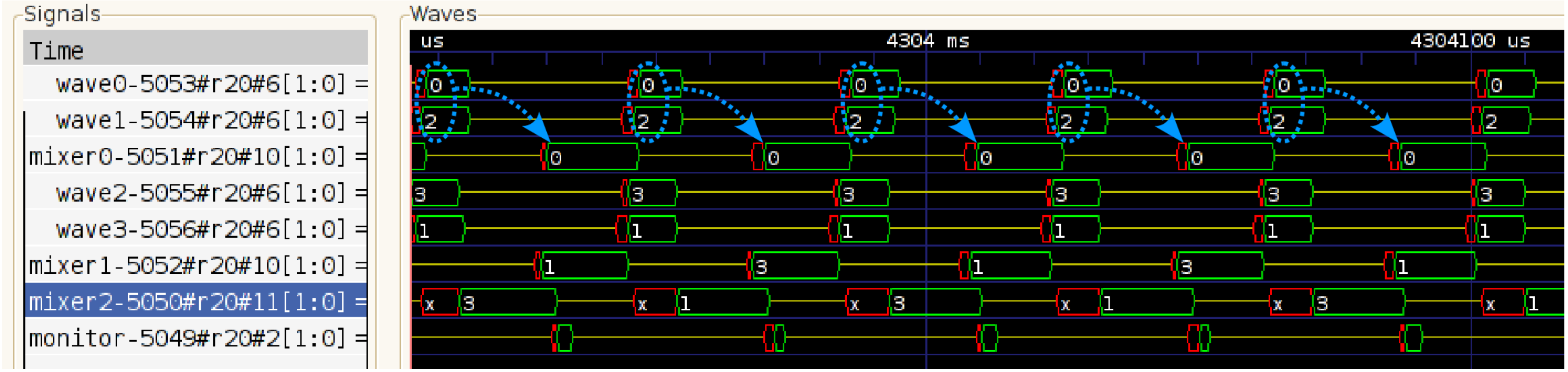
Solution – task-affinity

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.



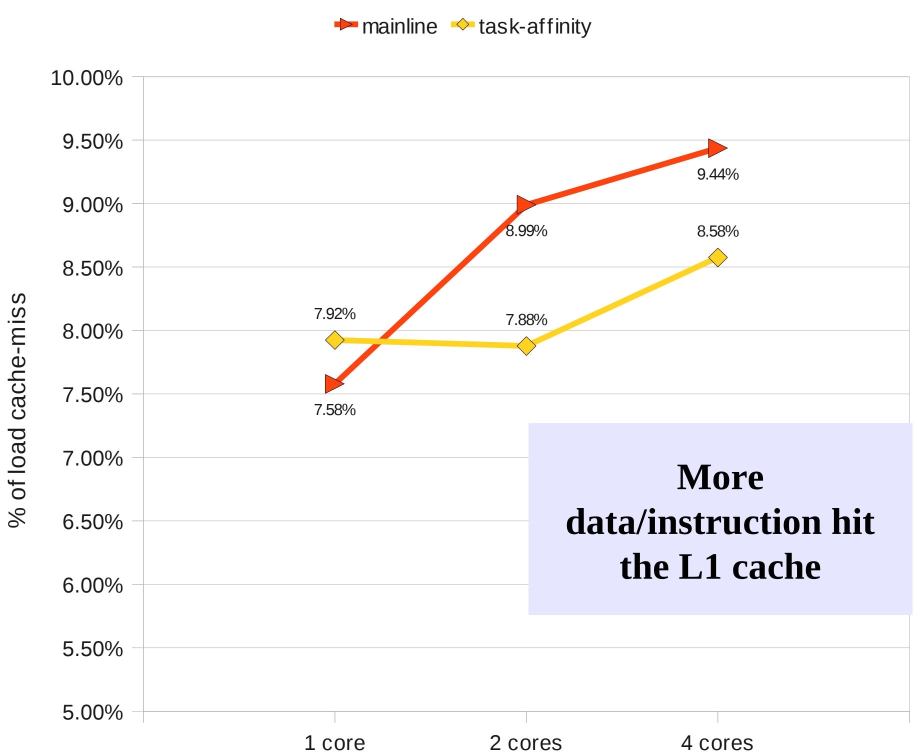
Solution – task-affinity
implementation

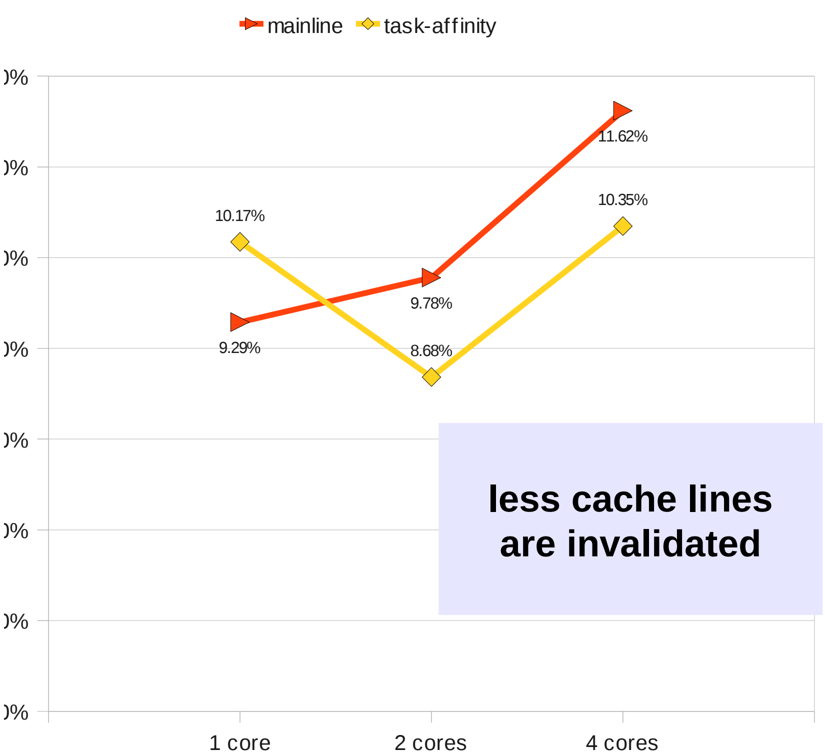
- 2 lists inside each `task_struct`:
  - `taskaffinity_list`
  - `followme_list`
- 2 system calls to add/delete affinities:
  - `sched_add_taskaffinity`
  - `sched_del_taskaffinity`

Experimental results

Cache-miss rates

- Measurements without and with task-affinity

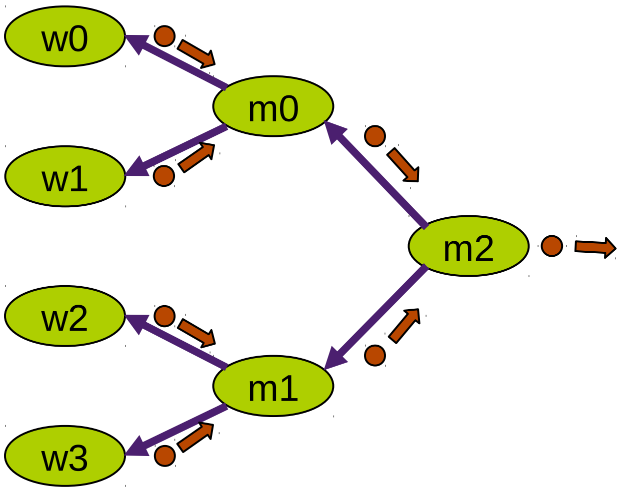




Experimental results

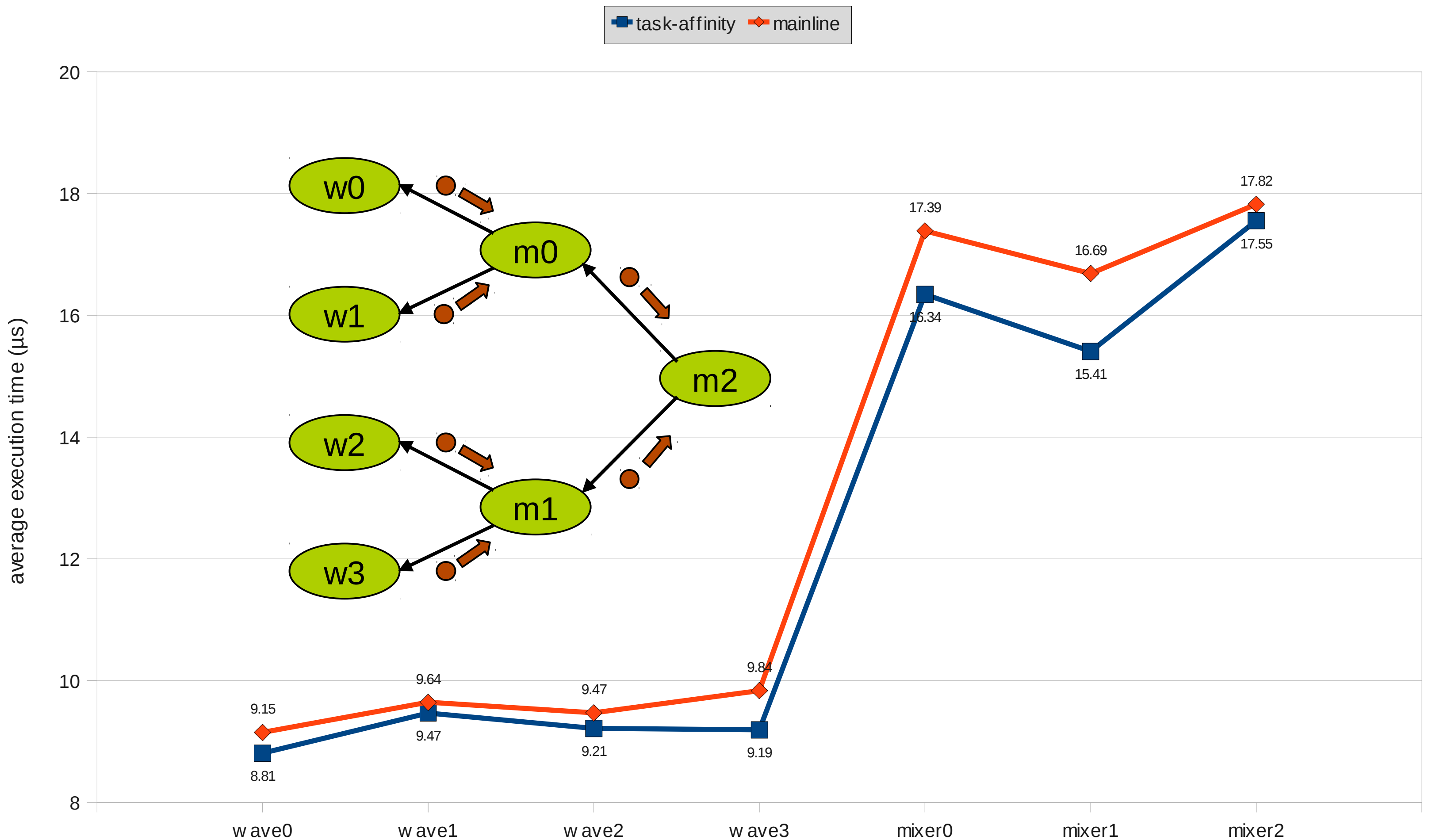
What exactly to evaluate?

- 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



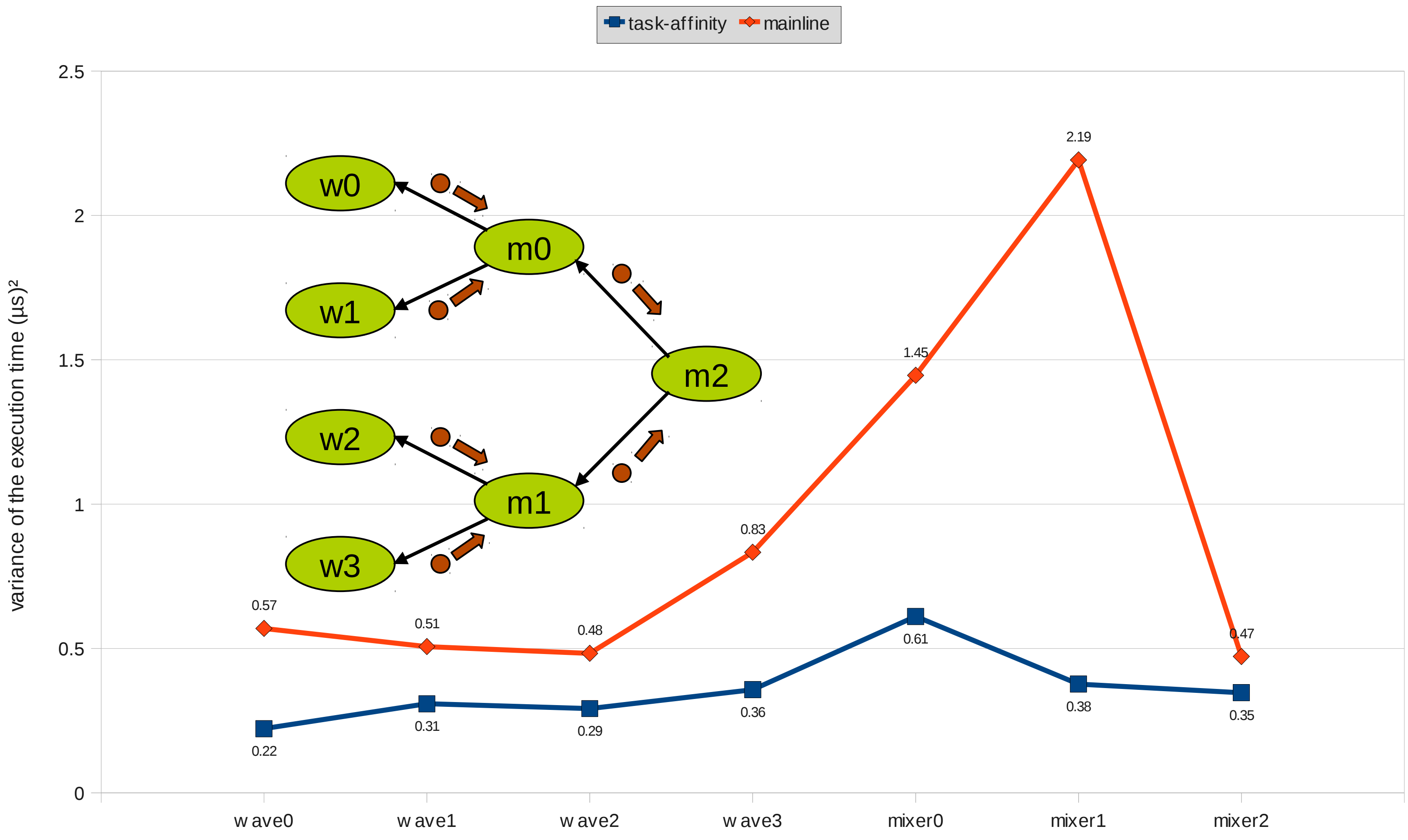
Experimental results

Average execution time of each task



Experimental results

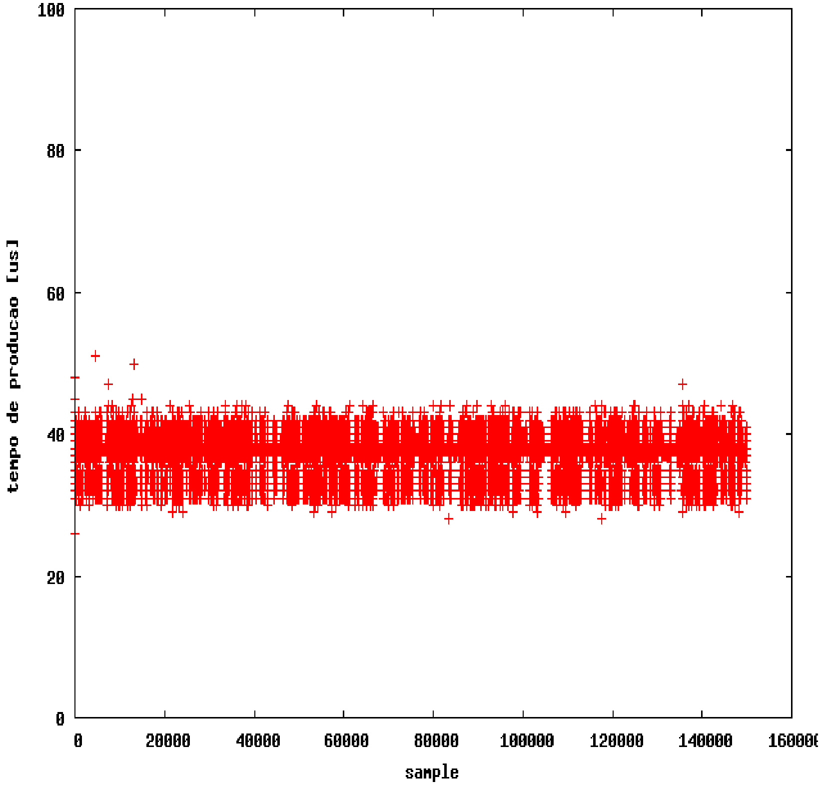
Variance of execution time of each task



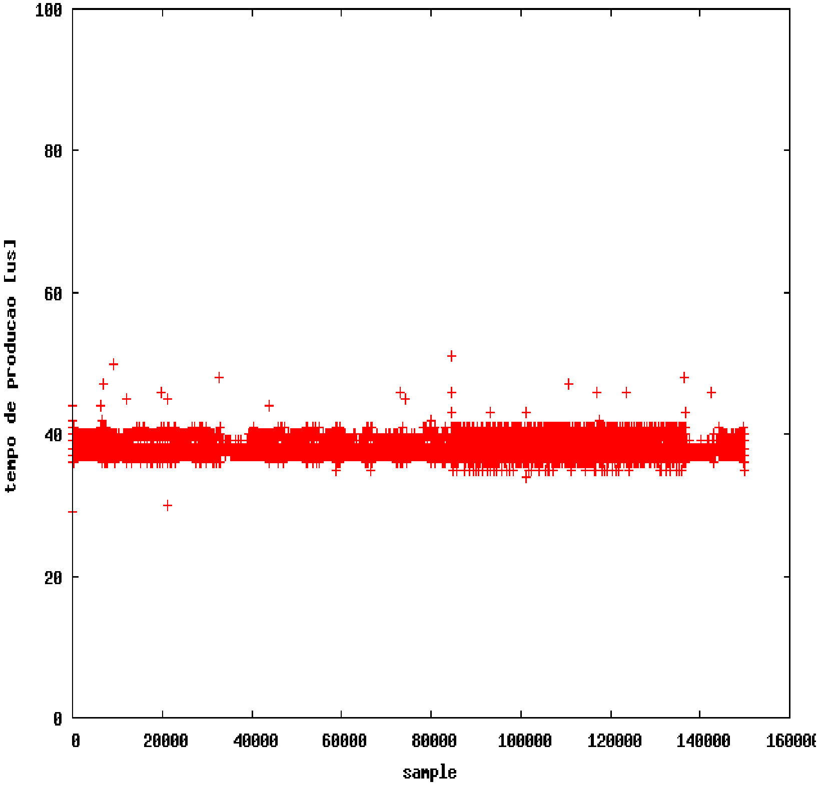
Experimental results

Production time of each single sample

- Results obtained for 150,000 samples



mainline

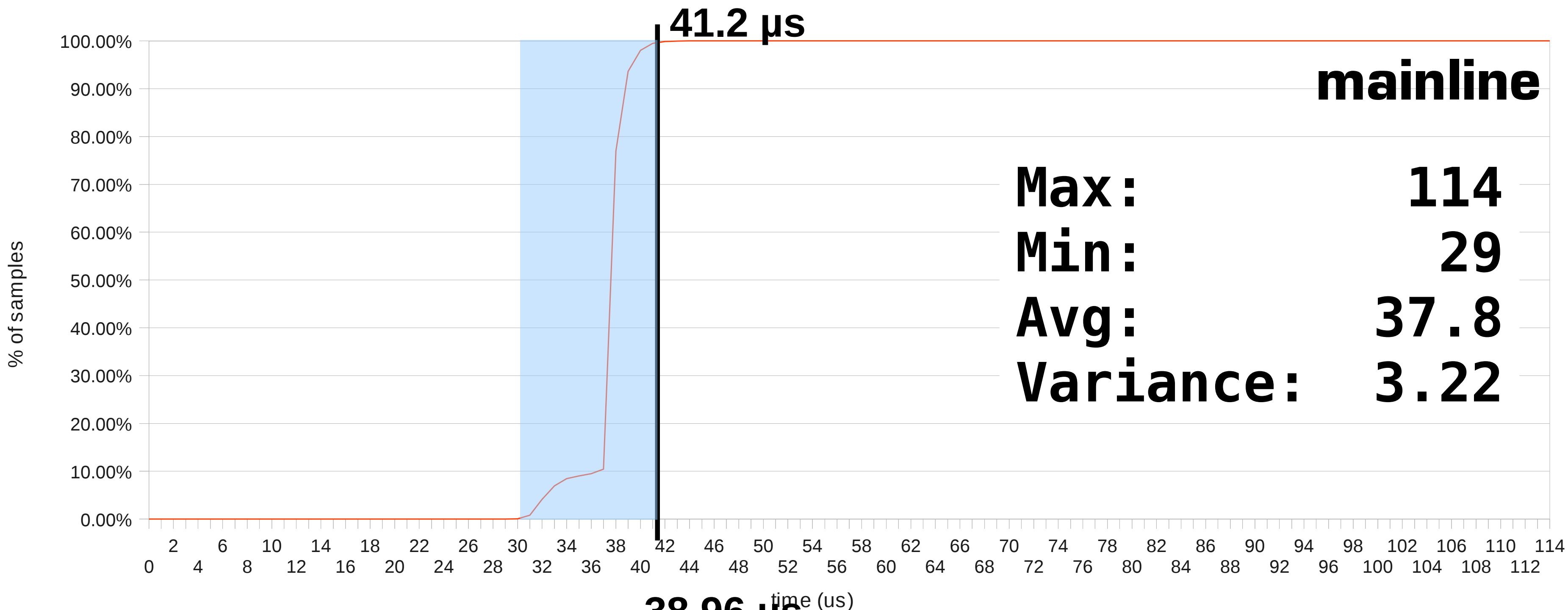


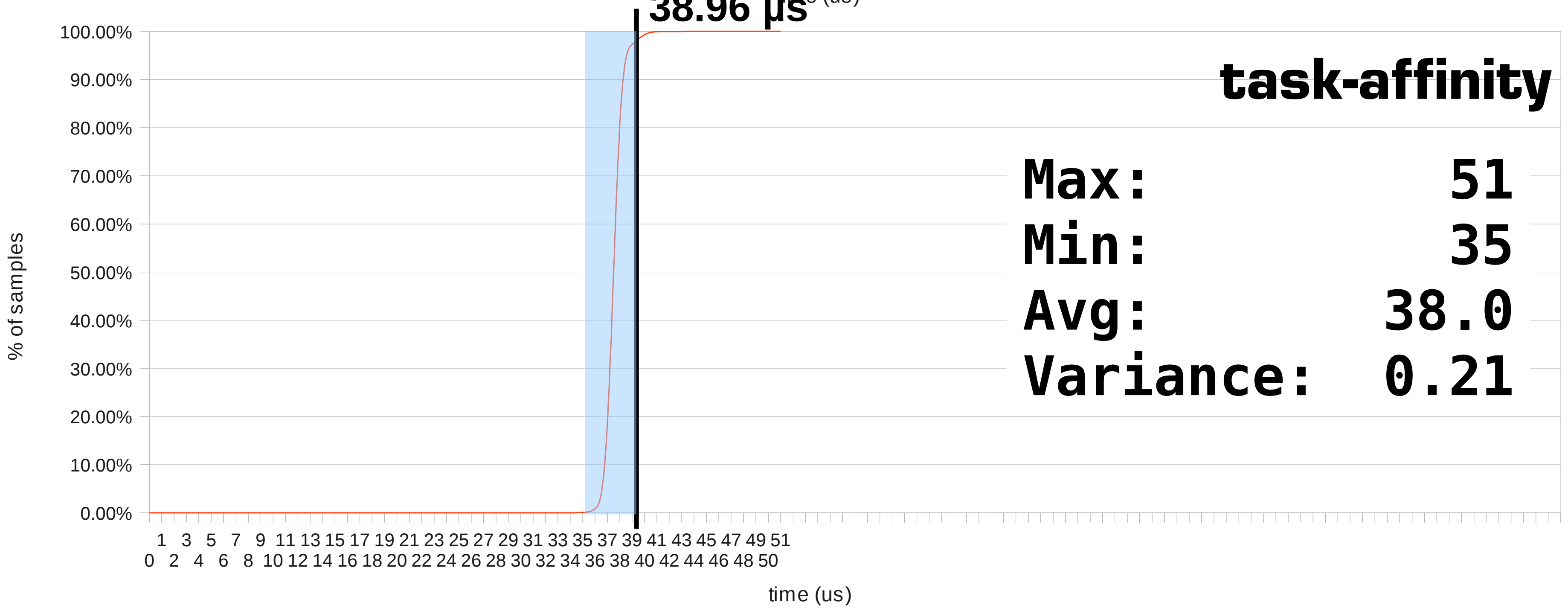
task-affinity

Experimental results

Production time of each single sample

- Empiric repartition function
- Real-time metric (normal distribution):
  - average + 2 * standard deviation


41.2 µs
mainline
Max: 114
Min: 29
Avg: 37.8
Variance: 3.22
% of samples
time (us)


38.96 µs
task-affinity
Max: 51
Min: 35
Avg: 38.0
Variance: 0.21
% of samples
time (us)

Experimental results

summary

	Average	Variance	Real-time Metrics	Speedup
mainline	37.826	3.225	41.42	–
taskaffinity	38.038	0.214	38.96	5.94%

~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.marchi@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