Multi-core scheduling optimizations for soft real-time applications

a cooperation aware approach

Lucas De Marchi

sponsors:

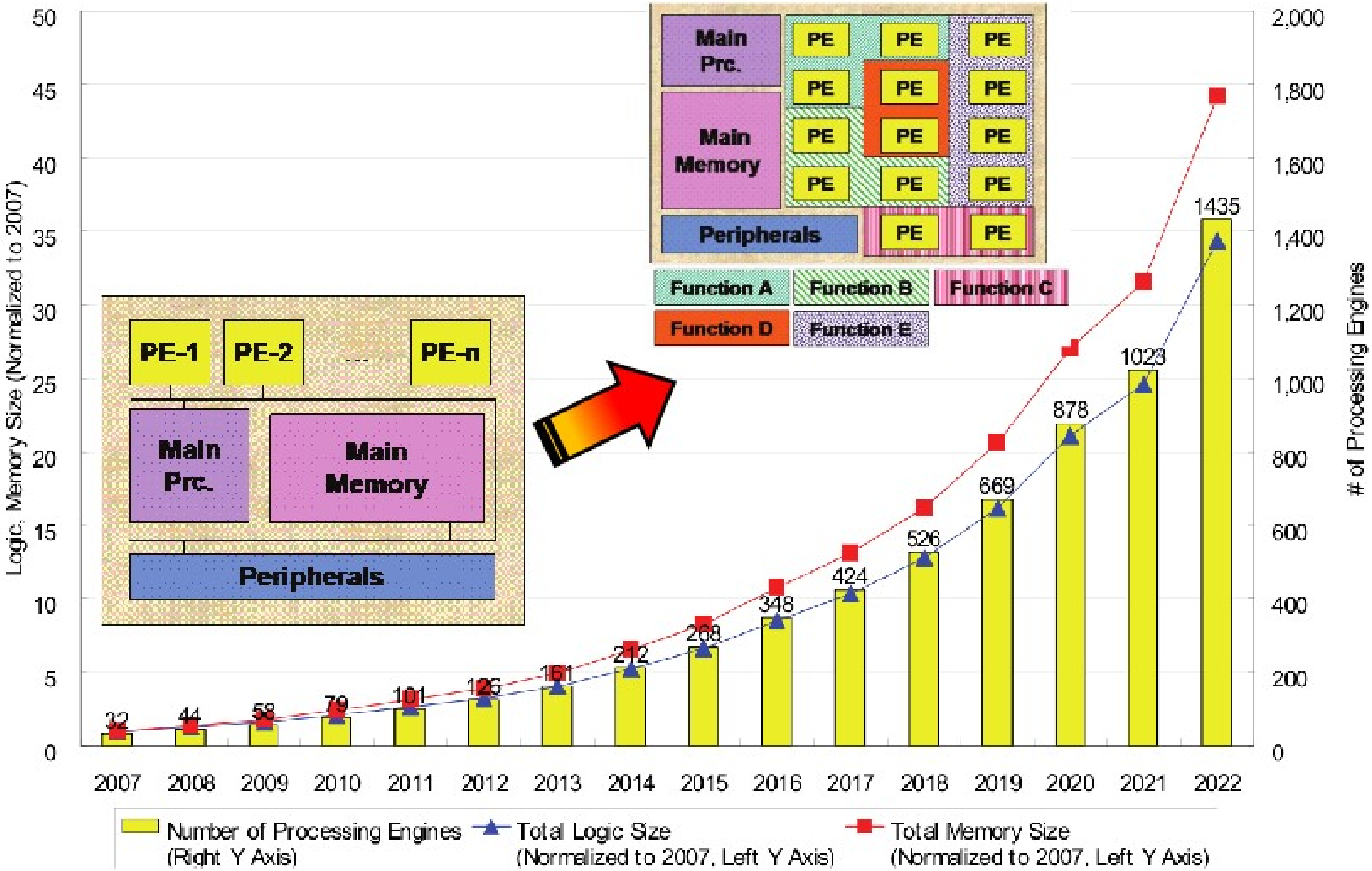
co-authors:

Patrick Bellasi
Wolfgang Betz
William Fornaciari
Liria Matsumoto Sato

Agenda

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

Introduction – motivation

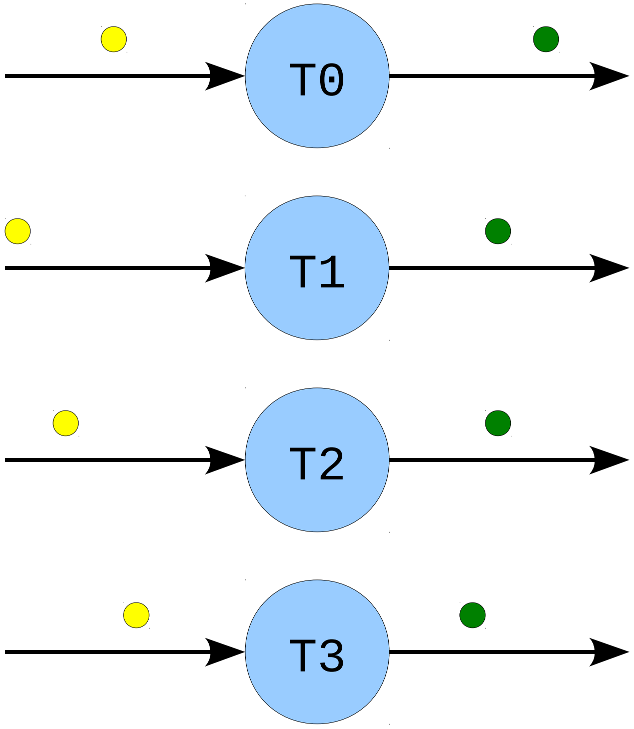


Introduction – motivation

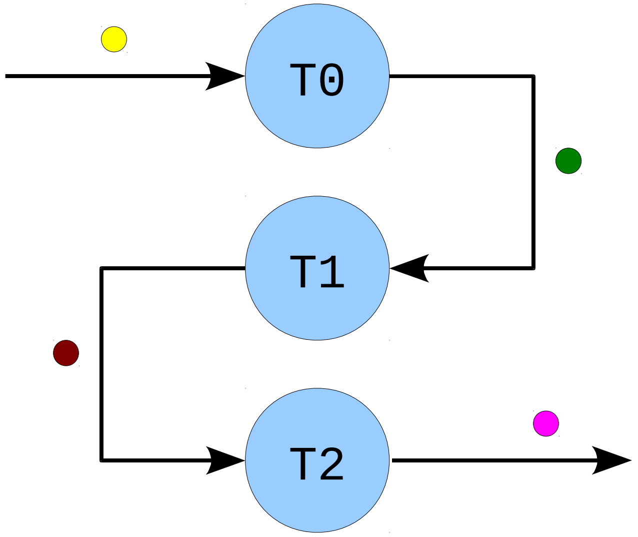
- SMP + RT
  - Multiple processing units inside a processor
  - Determinism
- Parallel Programming Paradigms
  - Data Level Parallelism (DLP)
    - Competitive tasks
  - Task Level Parallelism (TLP)
    - Cooperative tasks

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?

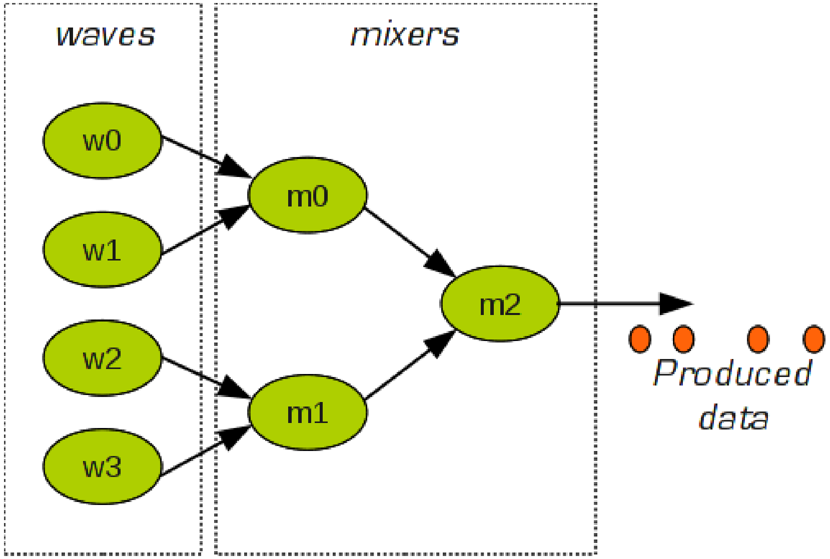
Anyway, why do we care about TLP?

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

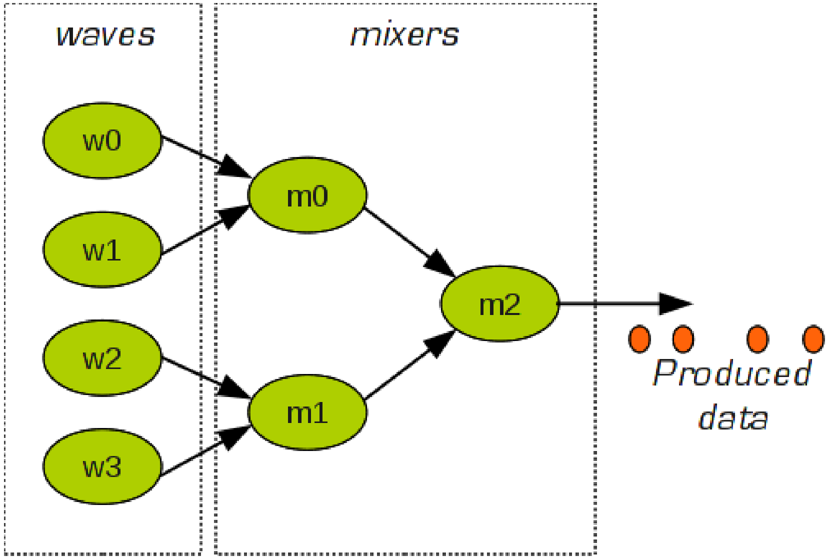
Analysis – benchmark

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



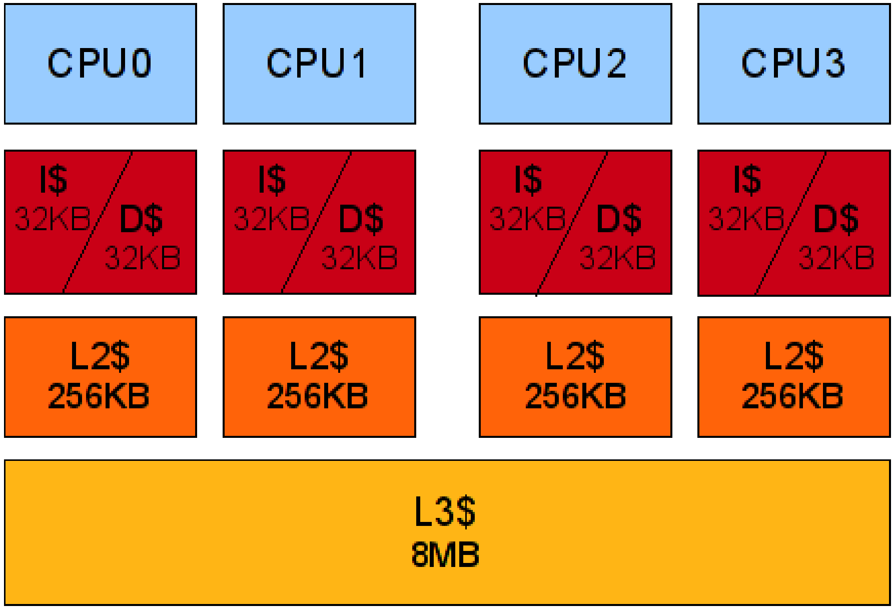
Analysis – metrics

- 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: 103302201010101010101010101010330101010101010101010101
  mixer2: 230201010101010101010101012020101010101010101010101021
 monitor: 111122222222222222222222231112222222222222222233322303
```

Analysis – locality of wake-up

- Migrations

b) occasional migrations

```
   wave0: 302121331102303323303311032320011021111111120101010101
   wave1: 223303013333212010121032121012333210320230333333333333
   wave2: 033332232302003111201211330021322233323330220222222222
   wave3: 110211120211101030031121003030120101101111111010101010
  mixer0: 322210301110320231310303021213120013220320230333333333
  mixer1: 001003321202220131103203212130320331201031033022010101
  mixer2: 230101020201202020121020021010130101201202302010101010
 monitor: 113022133333131312032133303222223233123111111222222222
```

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

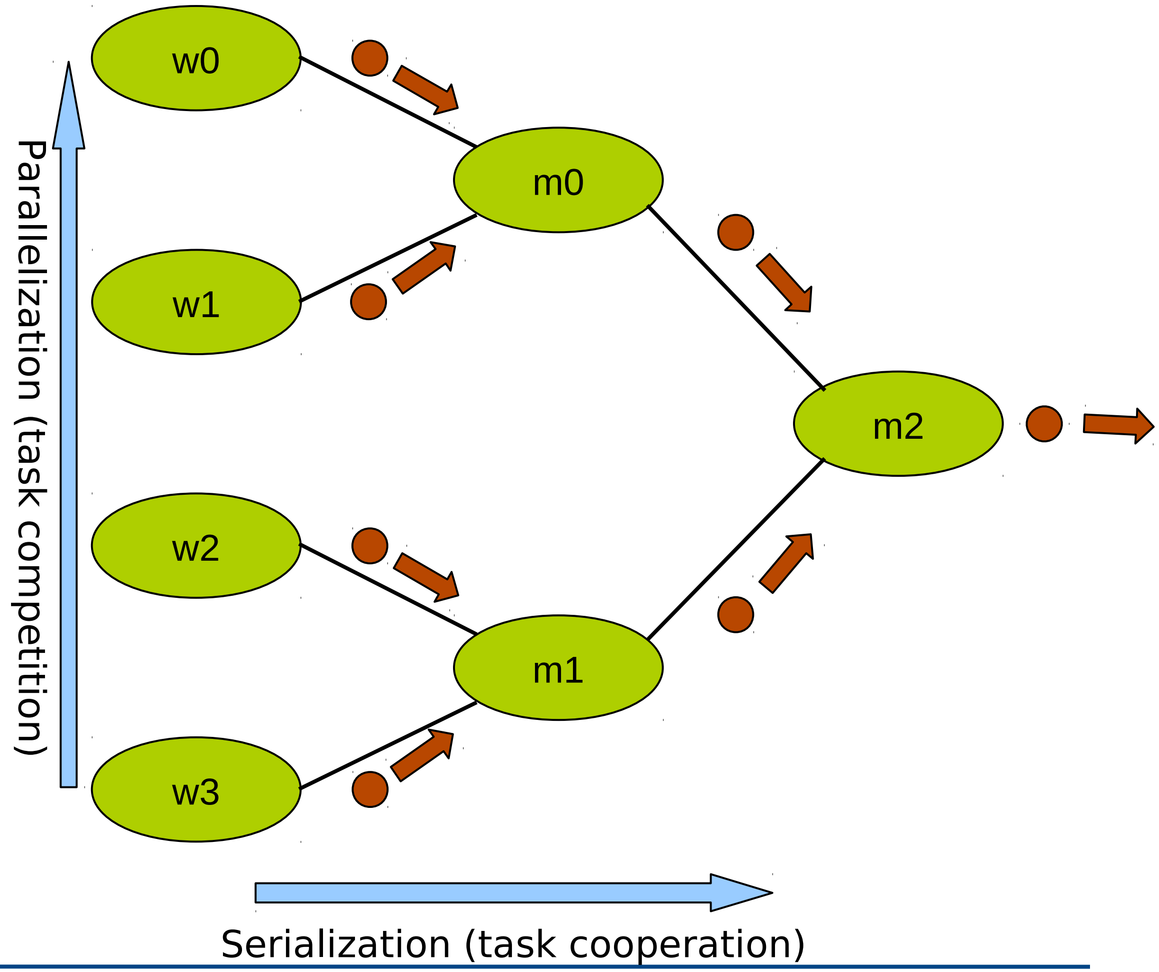
- Why do we care about TLP?
  - Common parallelization technique
- What about TLP?
  - Current state of Linux scheduler is not as good as we want

Solution – benchmark

- **Abstraction:**

  One application level

  sends data to another


w0
w1
w2
w3
m0
m1
m2
Parallelization (task competition)
Serialization (task cooperation)

Solution – benchmark

- **Abstraction:**

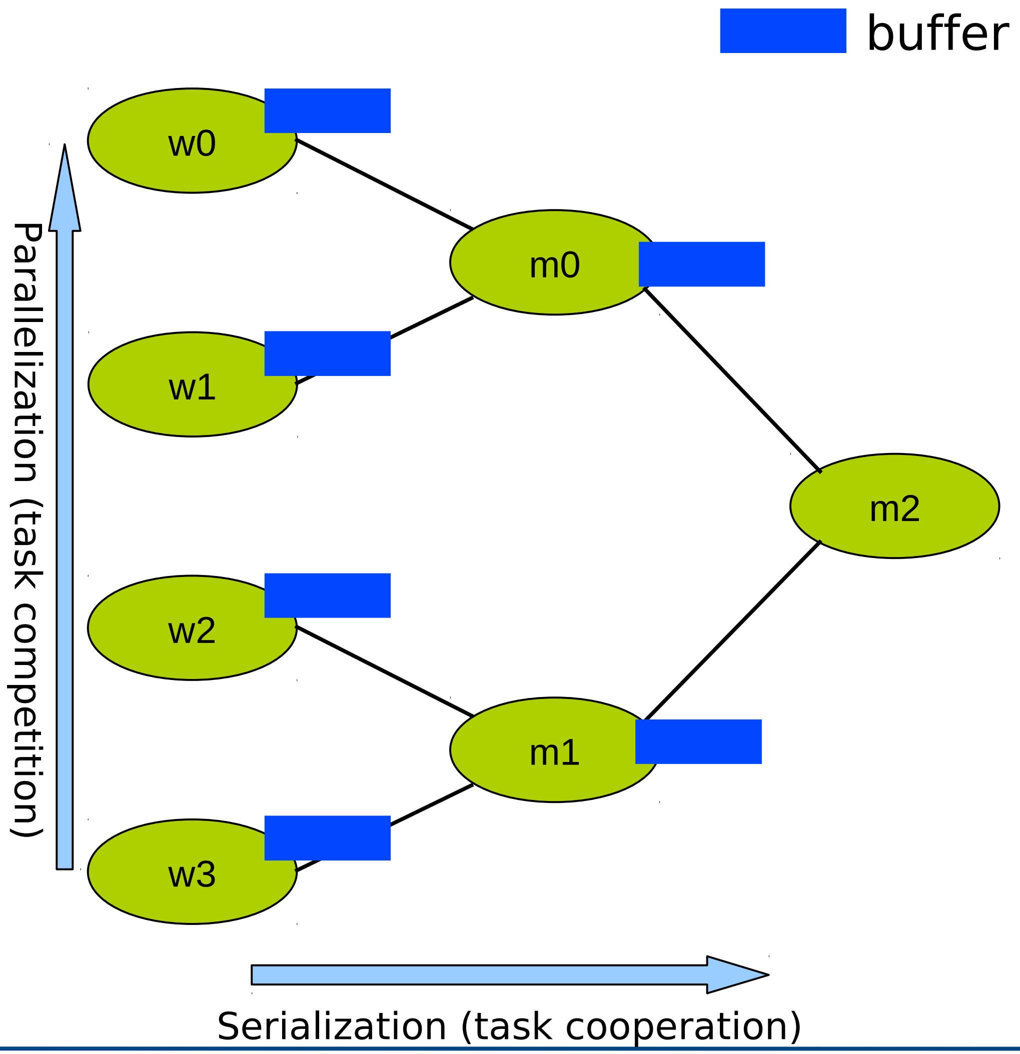
  One application level

  sends data to another

- **Reality:**

  shared buffers +

  synchronization


buffer
w0
w1
w2
w3
m0
m1
m2
Parallelization (task competition)
Serialization (task cooperation)

Solution – benchmark

- **Abstraction:**

  One application level

  sends data to another

- **Reality:**

  shared buffers +

  synchronization

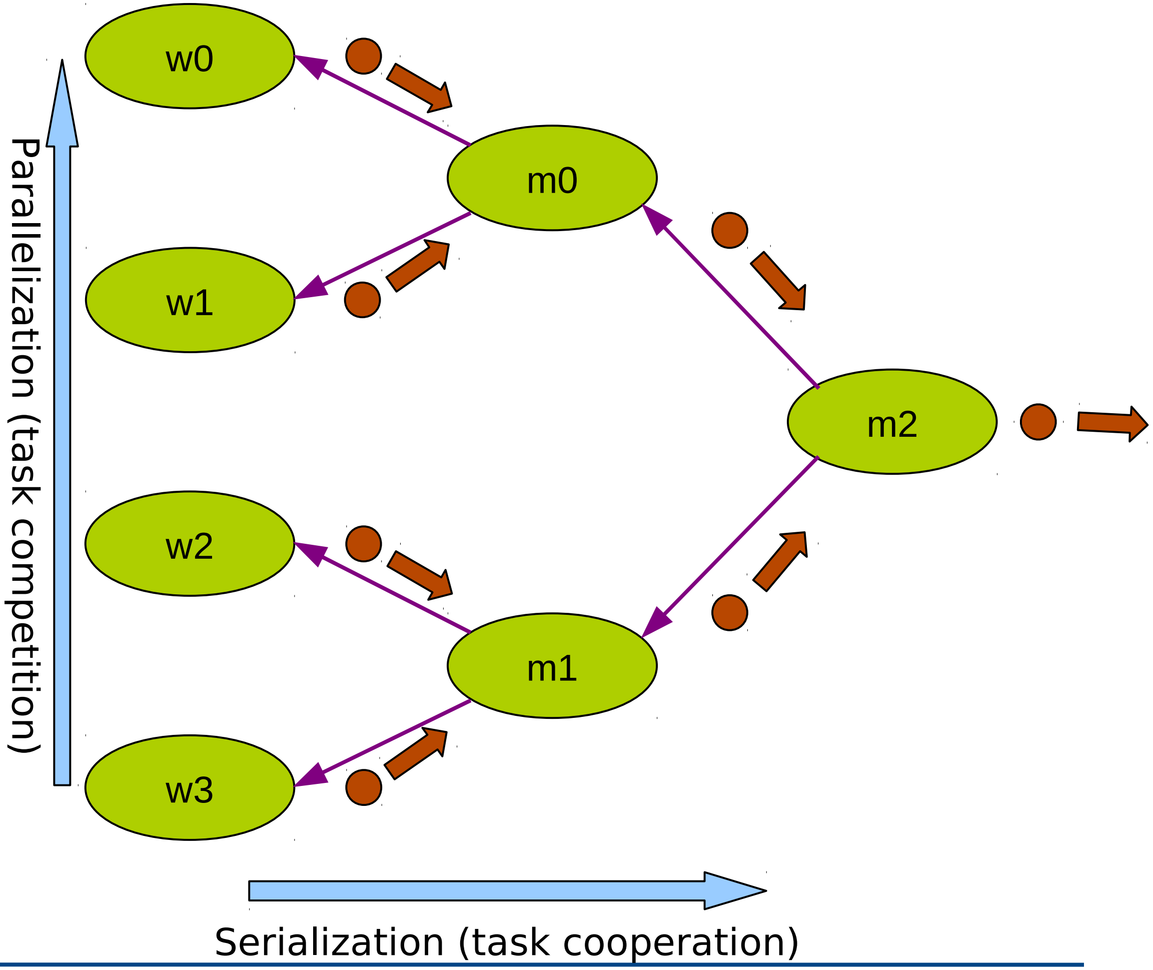
- **Dependencies:**

  Define dependencies

  among tasks in the

  opposite way of

  data flow


w0
w1
w2
w3
m0
m1
m2
Parallelization (task competition)
Serialization (task cooperation)

Solution – idea

Dependency followers

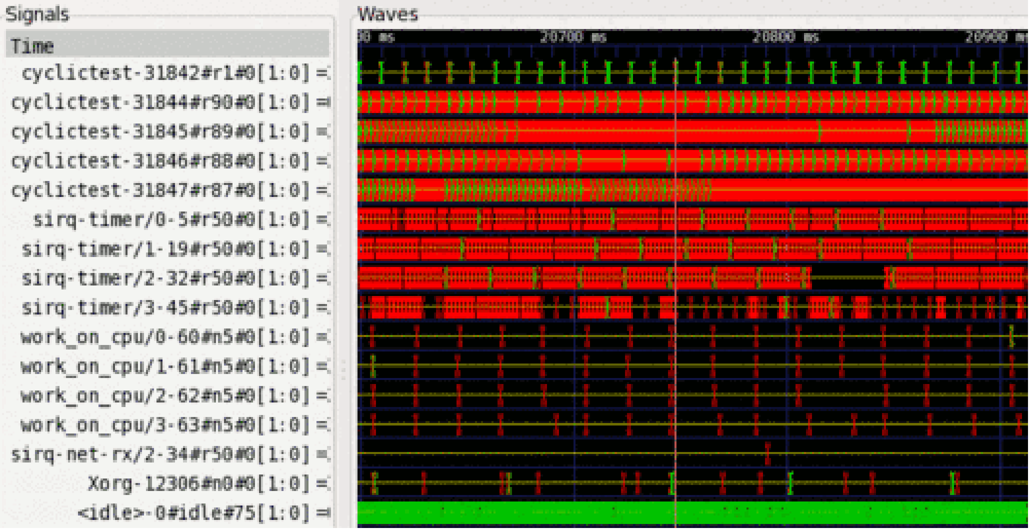
If data do not go to tasks,

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


Signals
Waves
Time
cyclictest-31842#r1#0[1:0]
cyclictest-31844#r90#0[1:0]
cyclictest-31845#r89#0[1:0]
cyclictest-31846#r88#0[1:0]
cyclictest-31847#r87#0[1:0]
sirq-timer/0-5#r50#0[1:0]
sirq-timer/1-19#r50#0[1:0]
sirq-timer/2-32#r50#0[1:0]
sirq-timer/3-45#r50#0[1:0]
work_on_cpu/0-60#n5#0[1:0]
work_on_cpu/1-61#n5#0[1:0]
work_on_cpu/2-62#n5#0[1:0]
work_on_cpu/3-63#n5#0[1:0]
sirq-net-rx/2-34#r50#0[1:0]
Xorg-12306#n0#0[1:0]
<idle>-0#idle#75[1:0]
20700 ms
20800 ms
20900 ms

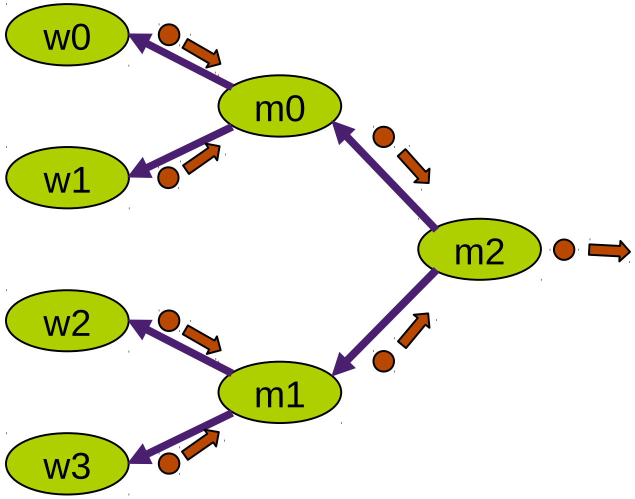
* http://www.osadl.org/Single-View.111+M5d51b7830c8.0.html

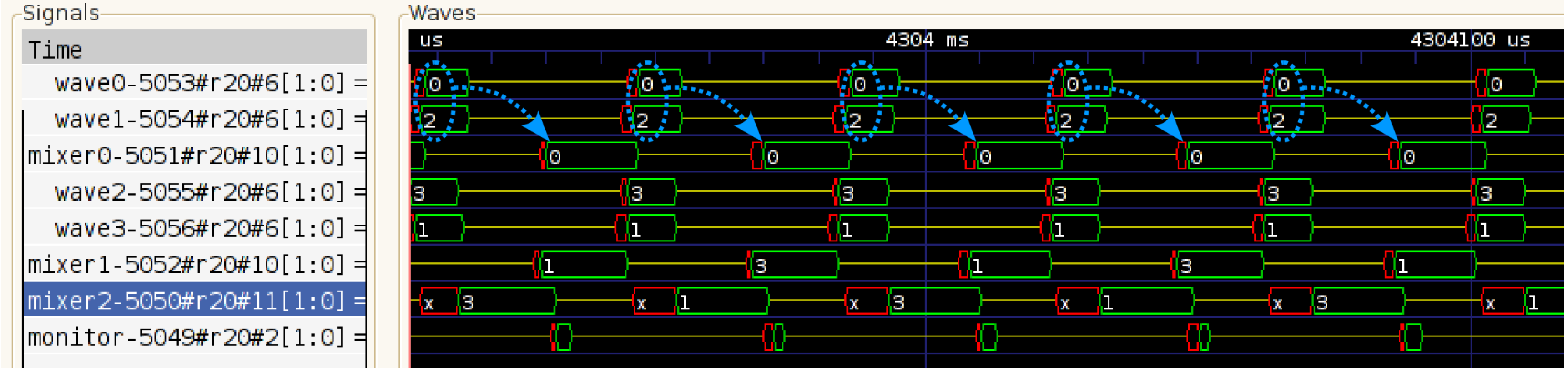
Solution – task-affinity

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.





Time
wave0-5053#r20#6[1:0] =
wave1-5054#r20#6[1:0] =
mixer0-5051#r20#10[1:0] =
wave2-5055#r20#6[1:0] =
wave3-5056#r20#6[1:0] =
mixer1-5052#r20#10[1:0] =
mixer2-5050#r20#11[1:0] =
monitor-5049#r20#2[1:0] =

Solution – task-affinity

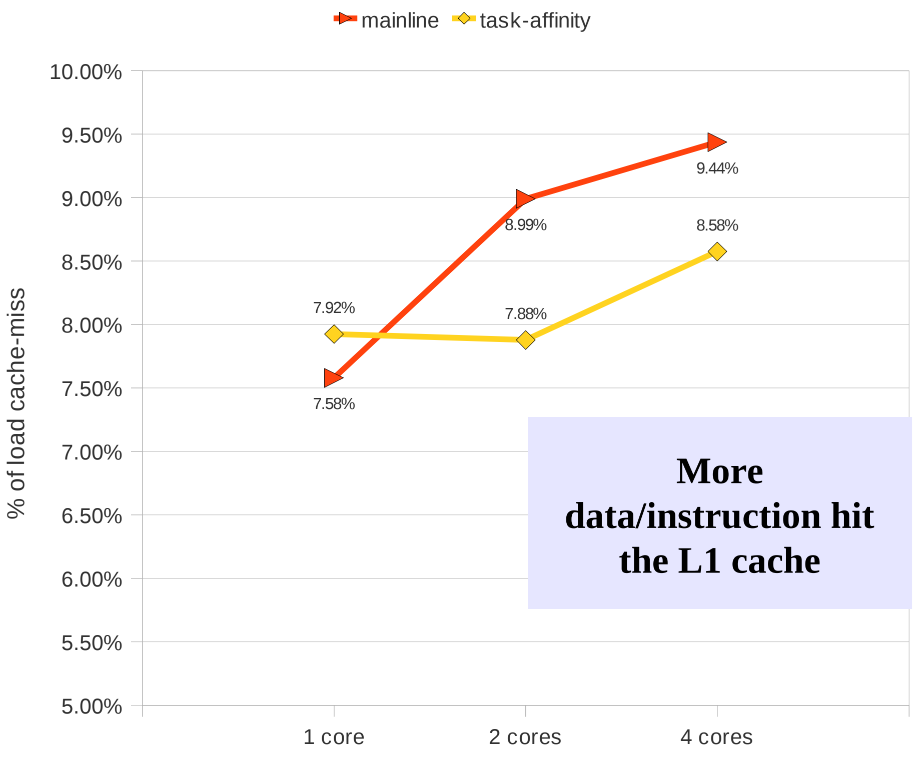
implementation

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

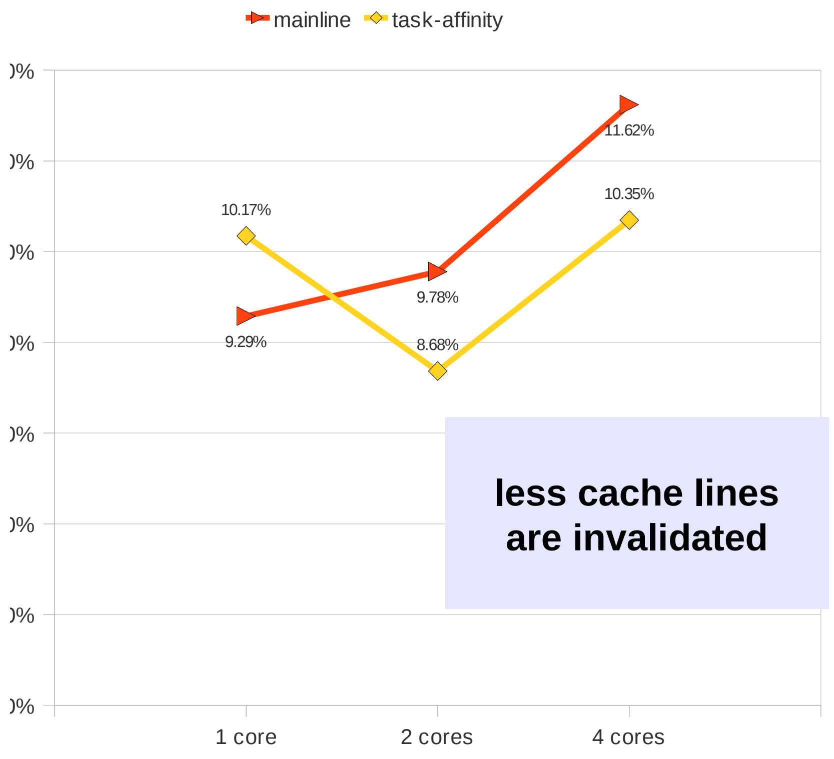
Experimental results

Cache-miss rates

- Measurements without and with task-affinity



More data/instruction hit the L1 cache

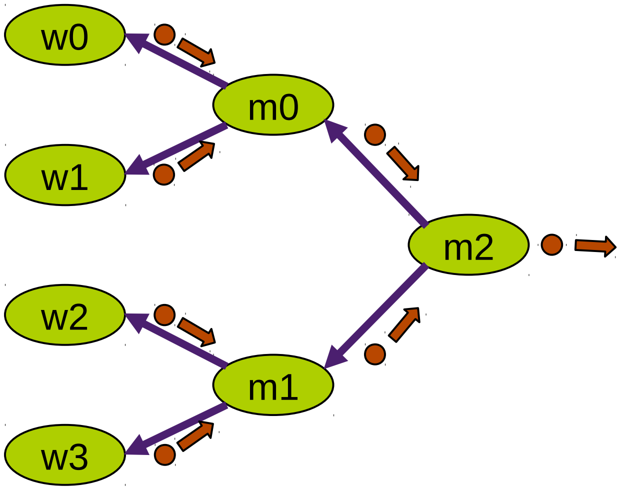


less cache lines are invalidated

Experimental results

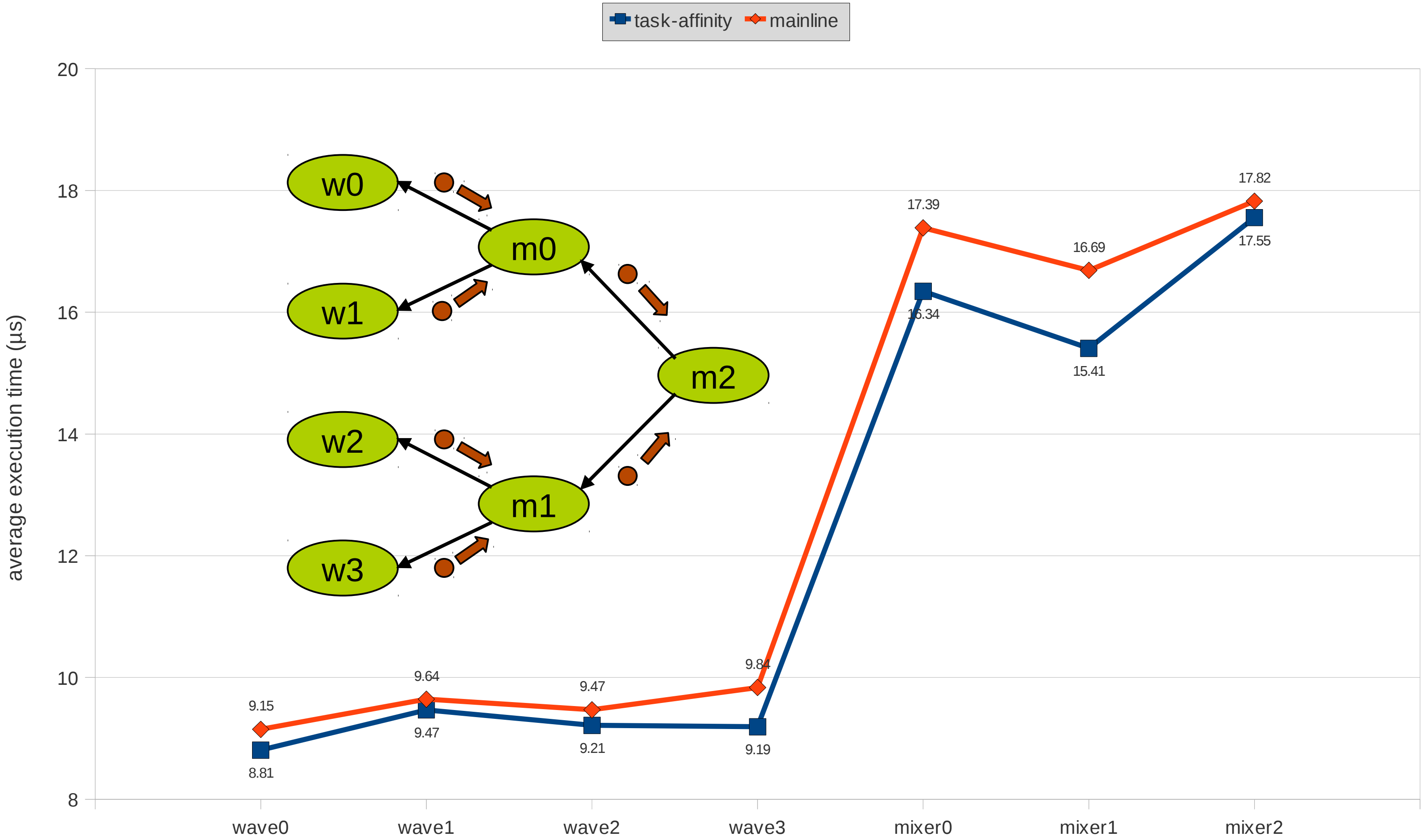
What exactly to evaluate?

- 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



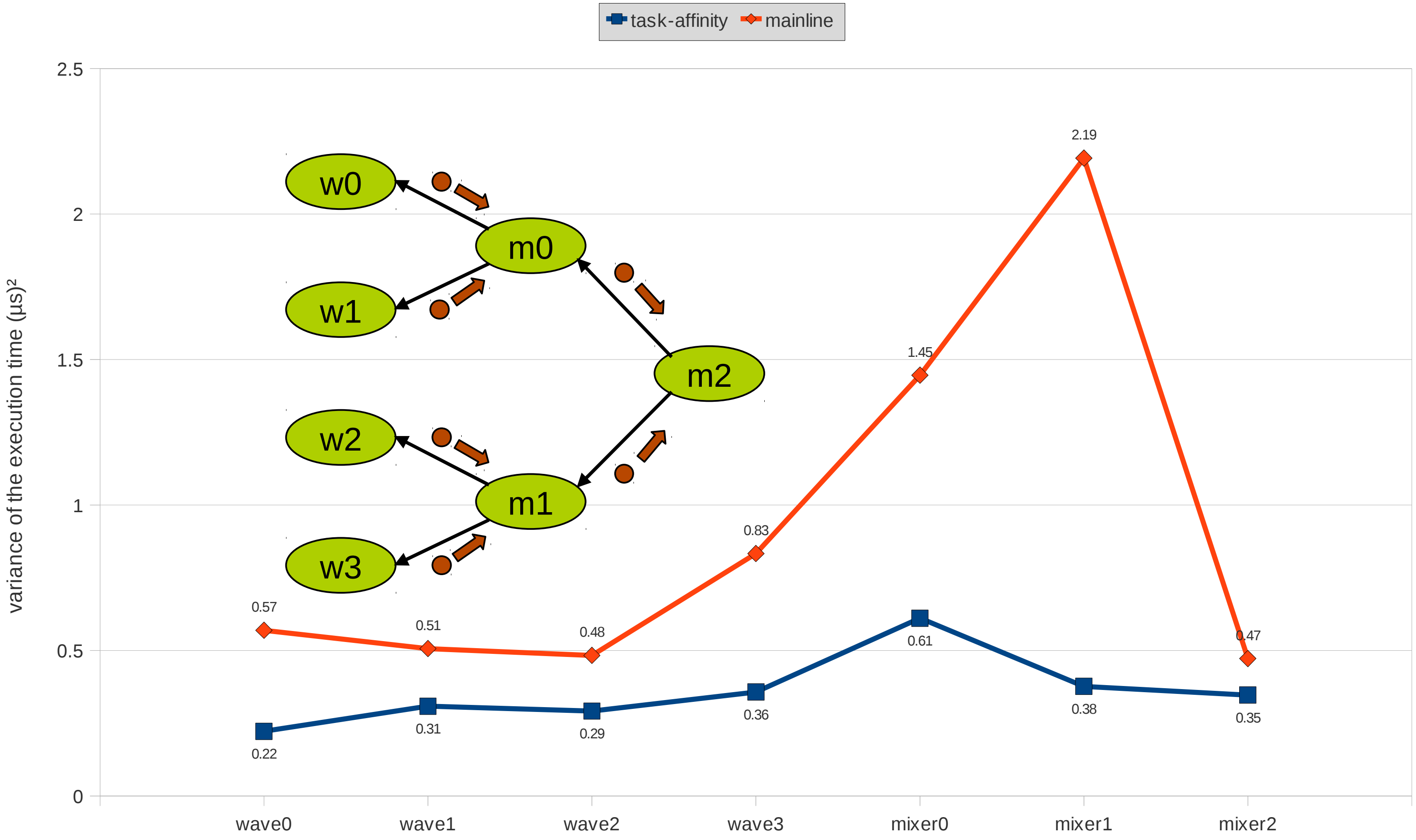
Experimental results

Average execution time of each task



Experimental results

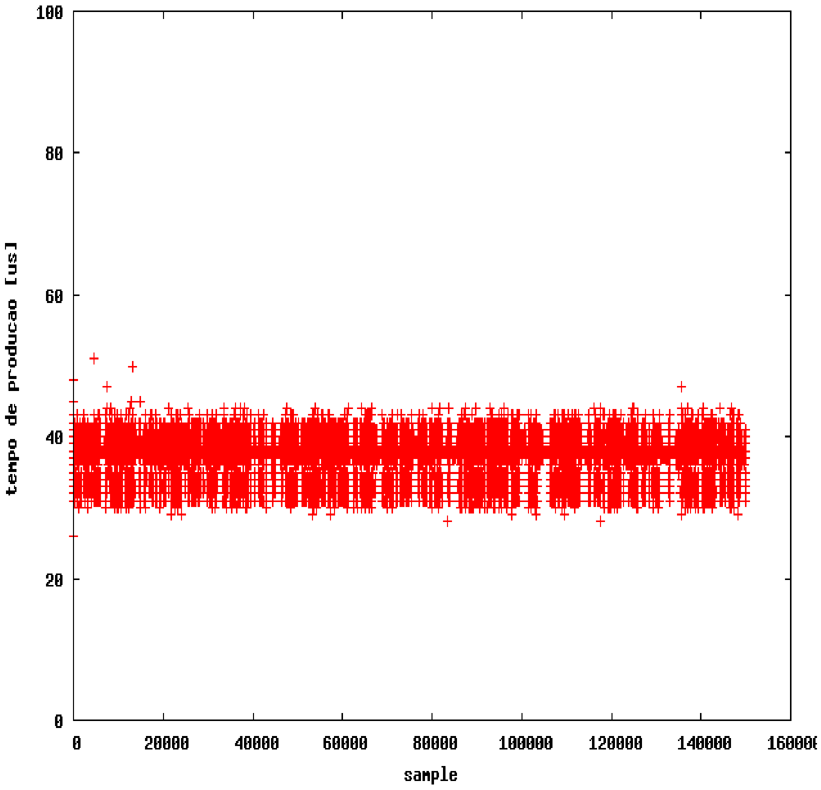
Variance of execution time of each task



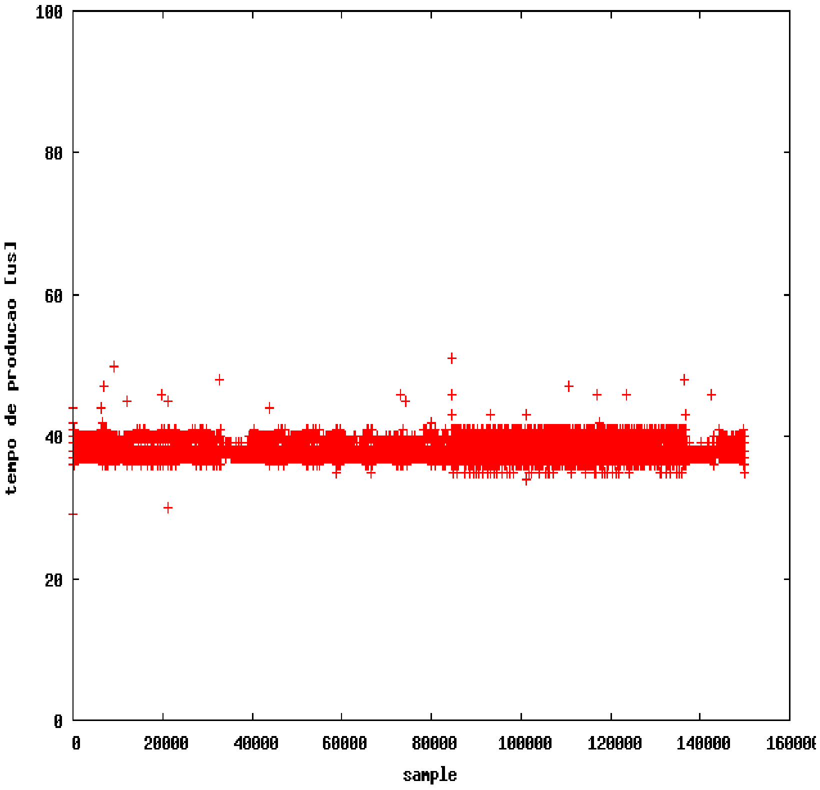
Experimental results

Production time of each single sample

- Results obtained for 150,000 samples



mainline

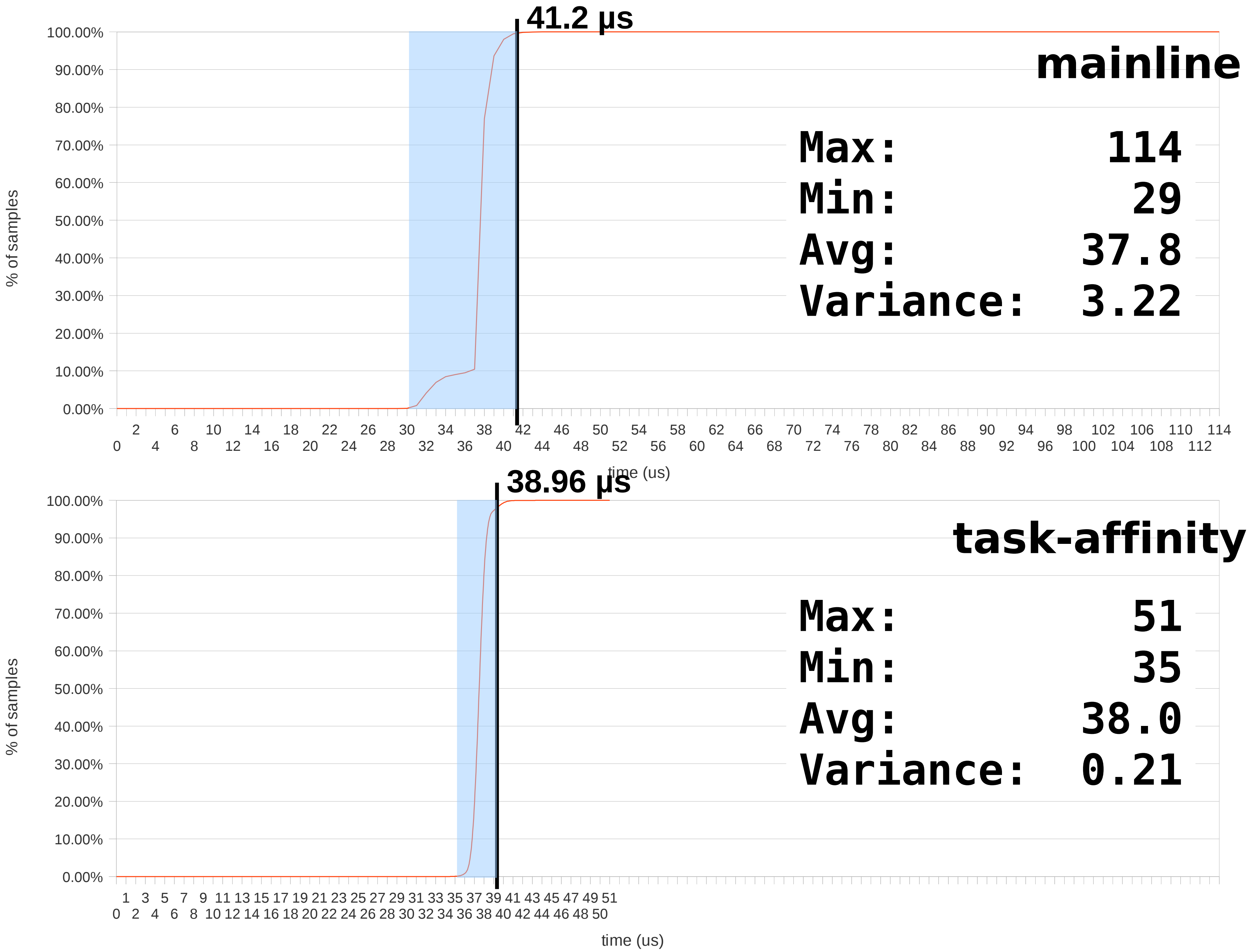


task-affinity

Experimental results

Production time of each single sample

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



mainline

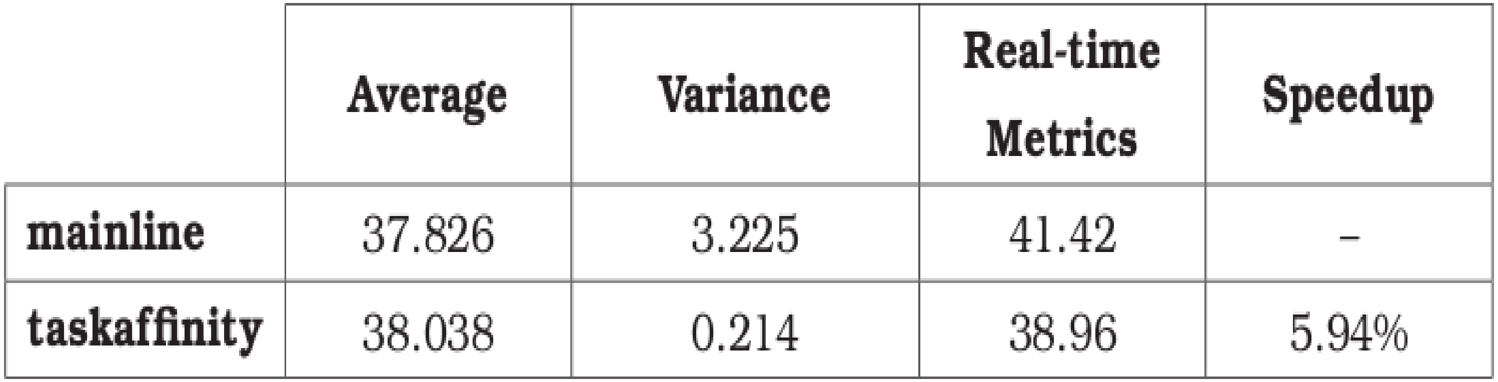
Max:	114
Min:	29
Avg:	37.8
Variance:	3.22

task-affinity

Max:	51
Min:	35
Avg:	38.0
Variance:	0.21

Experimental results

summary



	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

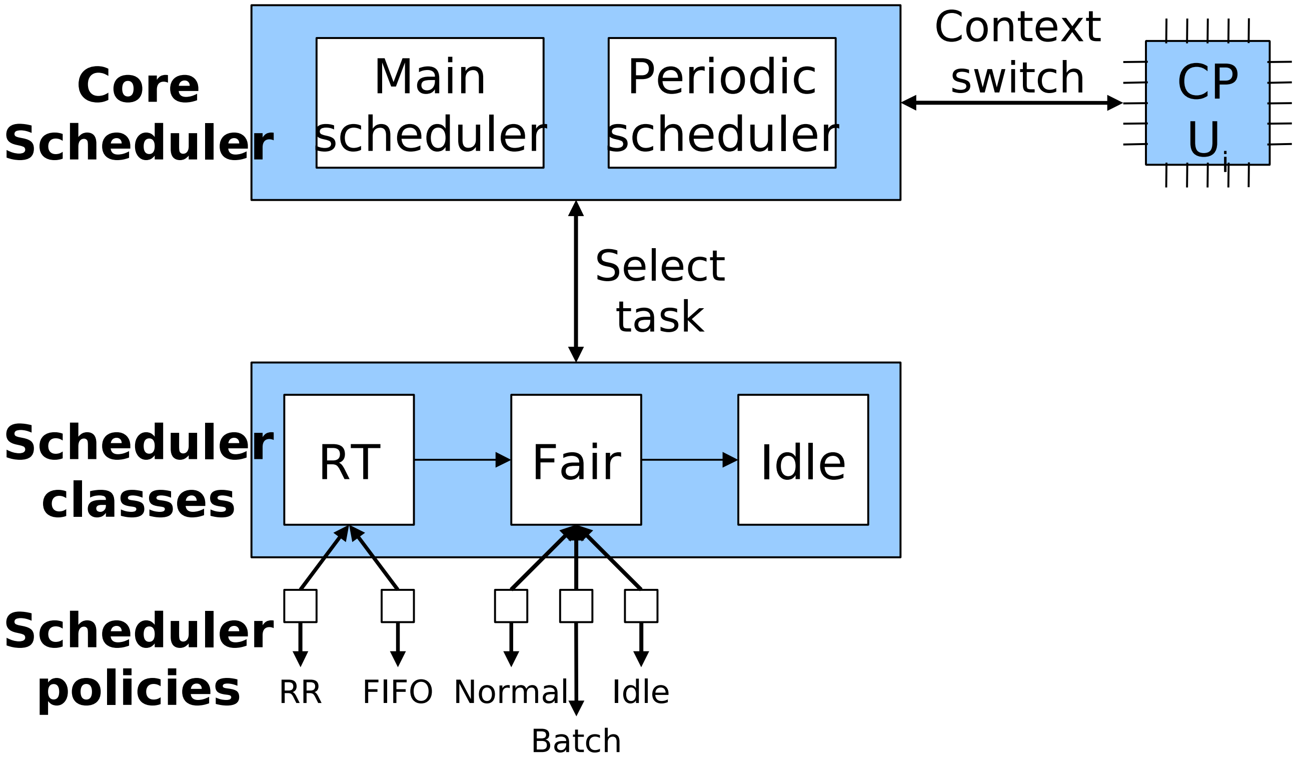
- Still a Work In Progress
- Git repository:
  - git://git.politreco.com/linux-lcs.git
- Contact:
  - lucas.demarchi@profusion.mobi
  - lucas.de.marchi@gmail.com

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



Core Scheduler
Main scheduler
Periodic scheduler
Context switch
CPU$_i$
Select task
Scheduler classes
RT
Fair
Idle
Scheduler policies
RR
FIFO
Normal
Batch
Idle