CISTER - Research Center in
Real-Time & Embedded Computing Systems

The variability of application execution times on a multi-core platform

Vincent Nélis

Patrick M. Yomsi

Luís M. Pinho

Our motivation

Memory Interference Delay Estimation for

IS

A Multi-Core Interference-Aware Schedulability Test for IMA Systems, as a Guide for SW/HW Integration

o Caccamo* and Lothar Thiele†
lliz2, mcaccamo}@cs.uiuc.edu
nzhofer, jchen, thiele}@tik.ee.ethz.ch

Soukayna M'Sirdi[1,2] Wenceslas Godard[1] and Marc Pantel[2]

veloped to compute an upper delay bound
uch
vith
can
mes
As
on.
me-
ems
eral
ride
of
ven
per
the
tive
ffic
em.
MA
of

[1]Airbus Group Inno
[2]IRIT,

Interference-free memory assignment in multi-core chips is NP-hard

Abstract—In this paper we propo
the automated integration and timi

Takeshi Tokuyama[4,d)]

NOT FAA POLICY OR GUIDANCE
DRAFT AND LIMITED RELEASE DOCUMENT
29 JANUARY 2016

items with 3D-stacked reconfigu-
nd are stacked on top of a layer of
s a vertical access port connected
nto several memory areas, where
sed by only a single IP core.
essors. Given the capacity of the
y tiles to processors. Specifically
nment where each processor has
ory blocks are orthogonally con-
that it is NP-complete to find an
the memory area of each core is

Reducing Memory Interfe
Application-Aware Me

Sai Prashanth Muralidhara
Pennsylvania State University
smuralid@cse.psu.edu

Lavany
Carnegi
Isubram

Mahmut Kandemir
Pennsylvania State Univers
kandemir@cse.psu.ed

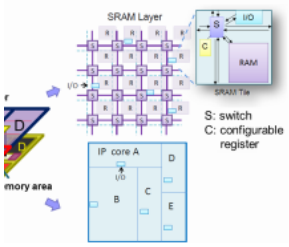
ABSTRACT

Main memory is a major shared resource among cores i
multicore system. If the interference between different ap
cations' memory requests is not controlled effectively, syst
performance can degrade significantly. Previous work aim
to mitigate the problem of interference between applicatio
by changing the scheduling policy in the memory controll
i.e., by prioritizing memory requests from applications i
way that benefits system performance.

In this paper, we first present an alternative approa
to reducing inter-application interference in the memo
system: *application-aware memory channel partition*
(MCP). The idea is to map the data of applications that
likely to severely interfere with each other to different me
ory channels. The key principles are to partition onto s
arate channels 1) the data of light (memory non-intensi
and heavy (memory-intensive) applications, 2) the data
applications with low and high row-buffer locality.

Second, we observe that interference can be further

White Paper on
Issues Associated with Interference Applied to Multicore Processors



DISCLAIMER
This draft document is being made available as a "Limited Release" document by the FAA Software and Digital Systems (SDS) Program and does not constitute FAA policy or guidance. This document is being distributed by permission by the Contracting Officer's Representative (COR). The research information in this document represents only the viewpoint of its subject matter expert authors.

Our motivation

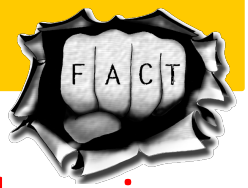
1. Processes running concurrently on different cores in a multicore environment interfere with each other on the processor shared resources.
2. The contention on these shared resources considerably slows down the execution on every core.

By **how much the execution may be slowed down** due to this interference!?

This paper answers this question **with numbers** coming from experiments.

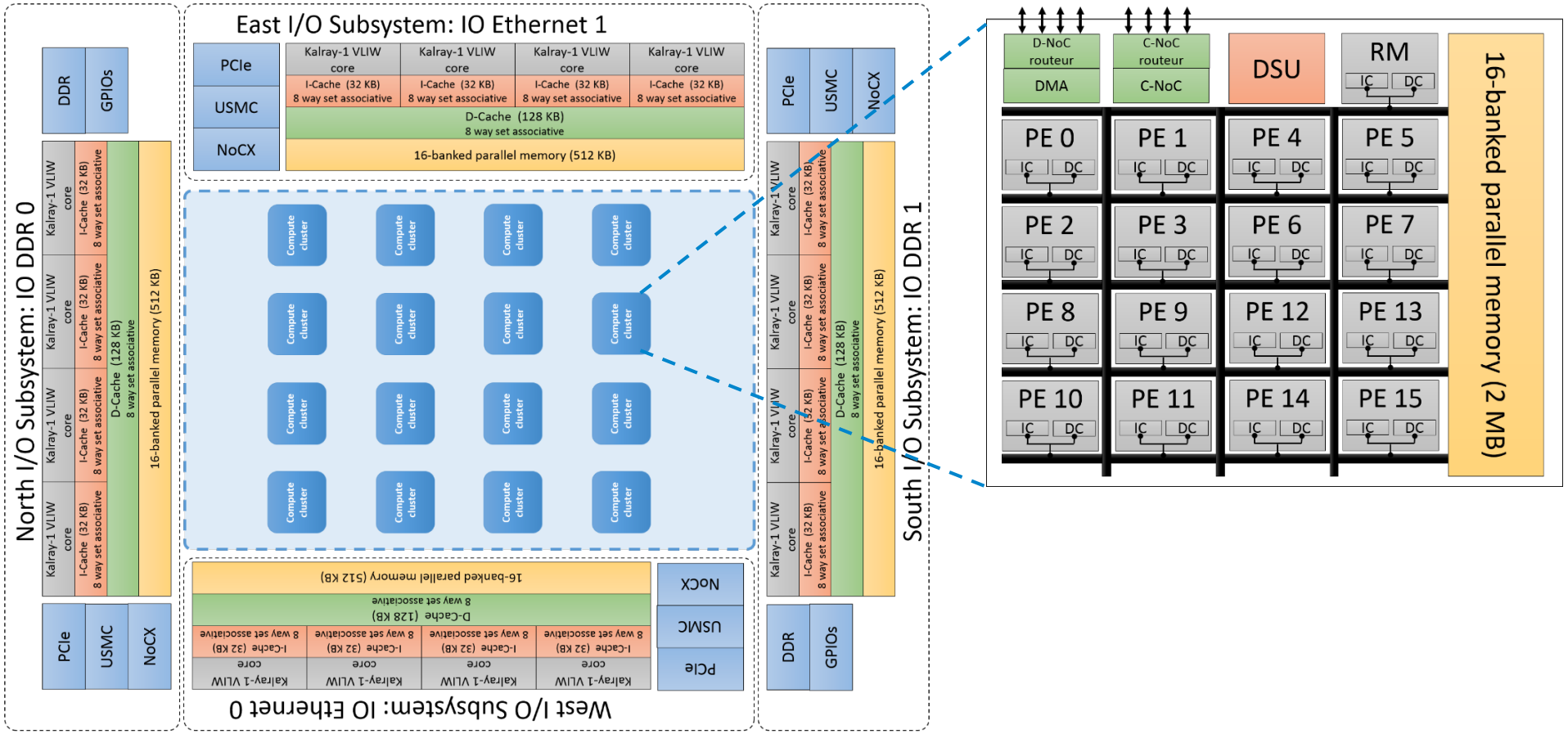
The TACLeBench benchmark suite

Good, realistic benchmark suites are essential for the evaluation and comparison of code-level timing analysis techniques.



- TACLeBench provides a freely available and comprehensive benchmark suite for timing analysis.
- TACLeBench is meant to become the standard benchmarking suite for timing analysis.
- The source codes are 100% self-contained – no dependencies to system-specific header files via #include directives exist.
- Almost all benchmarks are processor-independent and can be compiled and evaluated for any kind of target processor.

The Kalray MPPA-256



We focus on the execution in **one cluster**

Our methodology(1/2)

1. Extensive (monitored) experiments
2. Measure the execution times
3. Take the maximum/minimum observed
4. Comments on the variability (based on past experience)

Application execution time:

- "Past experience" with multicores?
- Can we improve that process?
- Can it provide insights for the computation of the WCET of the application?

Our methodology(2/2)

Application

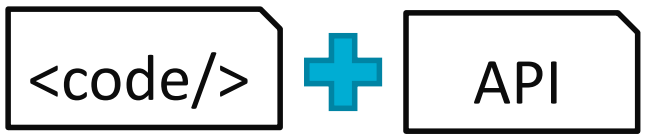
In isolation: Extract maximum intrinsic execution time (MIET)

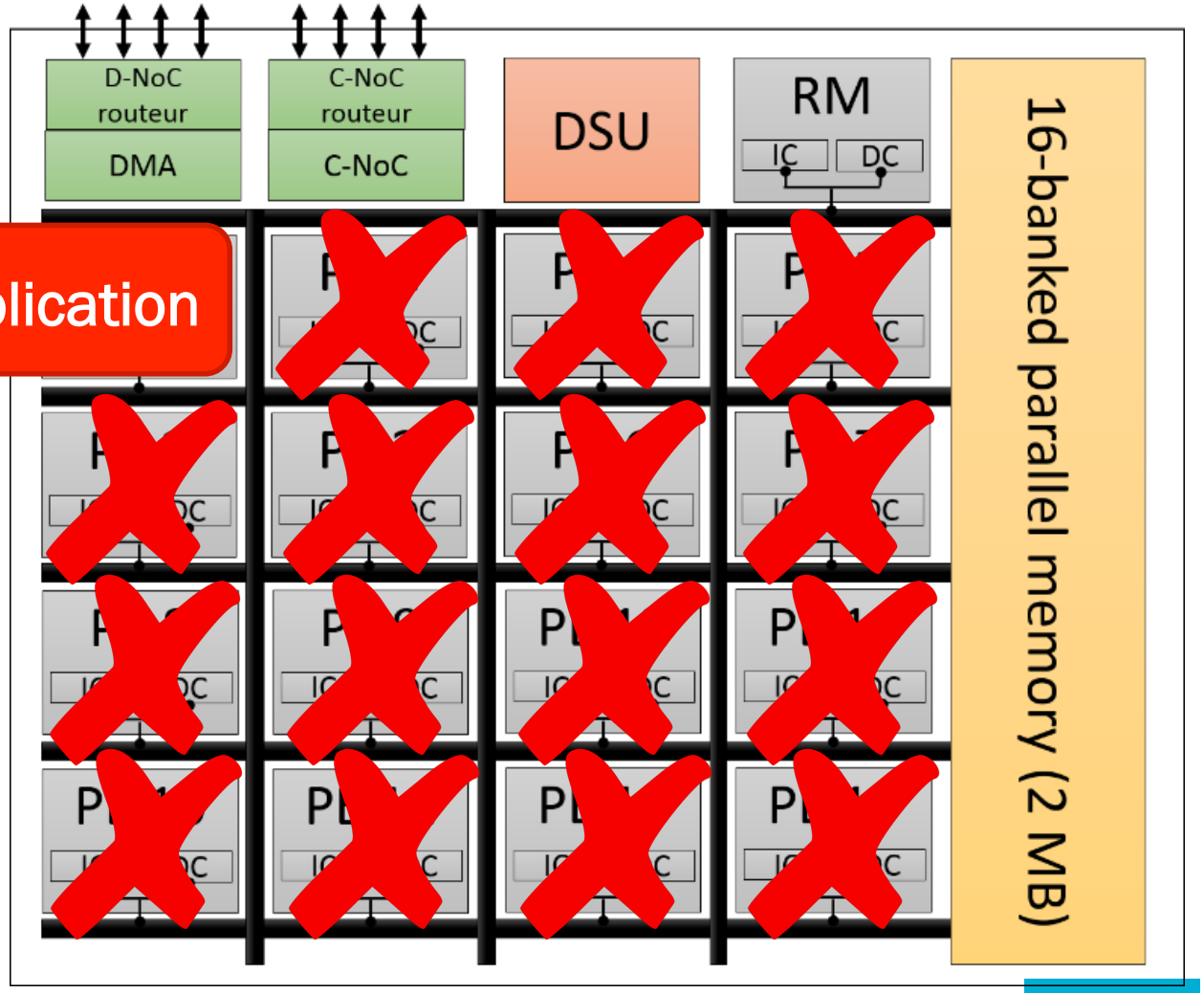
In contention: Extract maximum extrinsic execution time (MEET)

The isolation mode

TACLeBench benchmark suite

Application

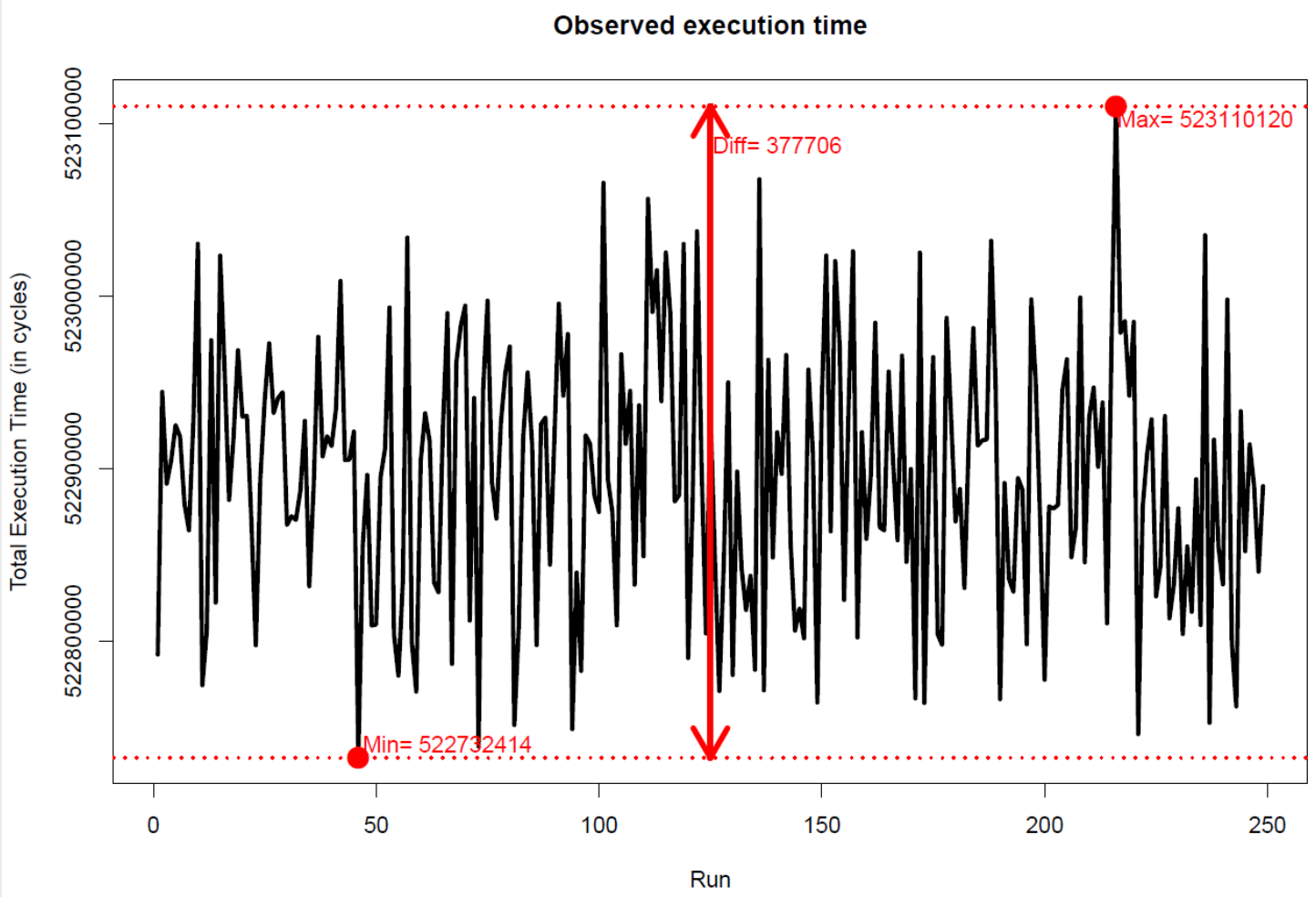




Which configuration?

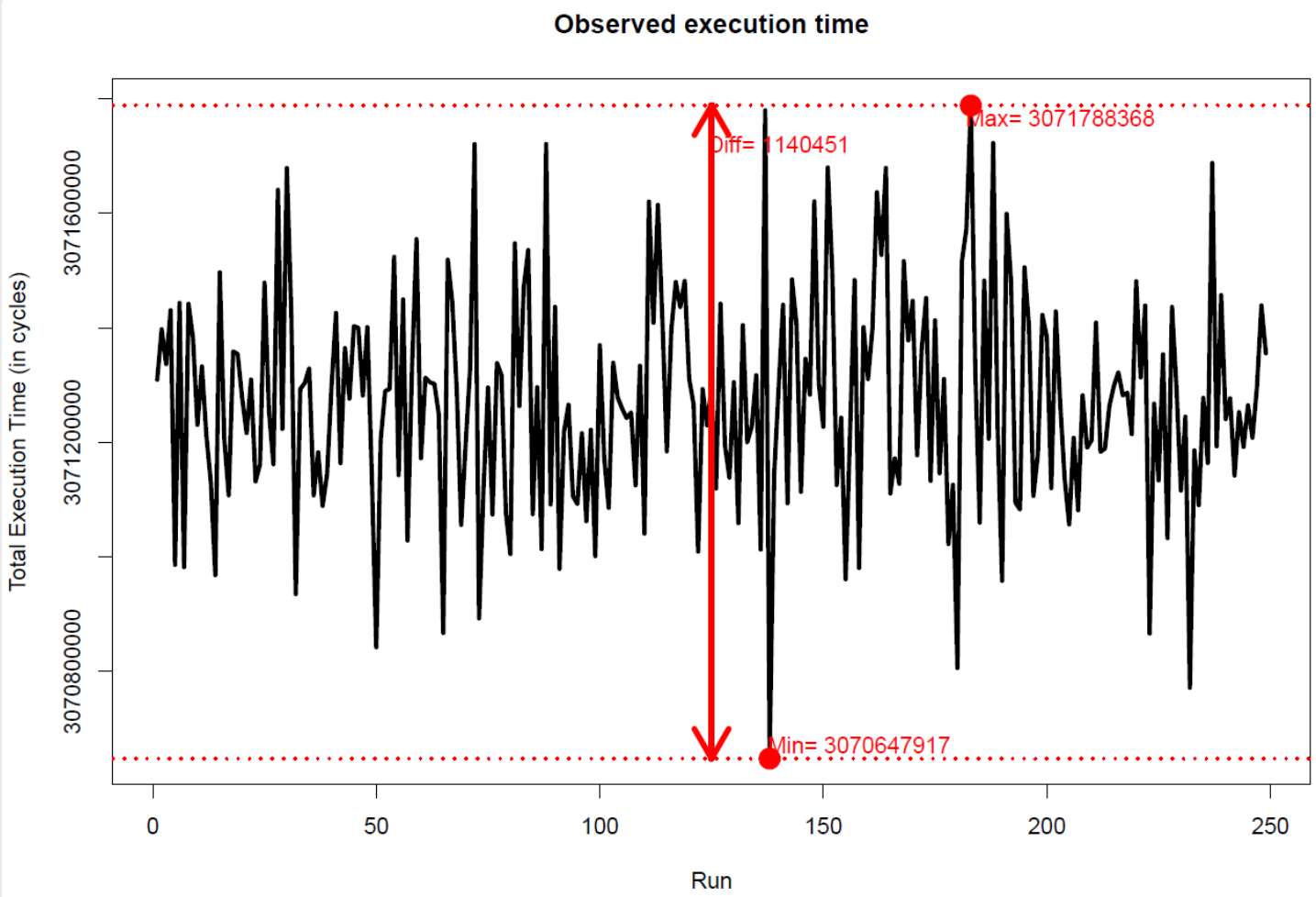
PLATFORM SETTINGS
Activate the D-cache
Invalidate the D-cache before each execution window
Stall on access
Activate the I-cache
Invalidate the I-cache before each execution
Memory mapping

The isolation mode – high-perf



Isolation High-perf:
Max = 523 M
Variation = 377 k
= 0,07 %

The isolation mode – low-perf

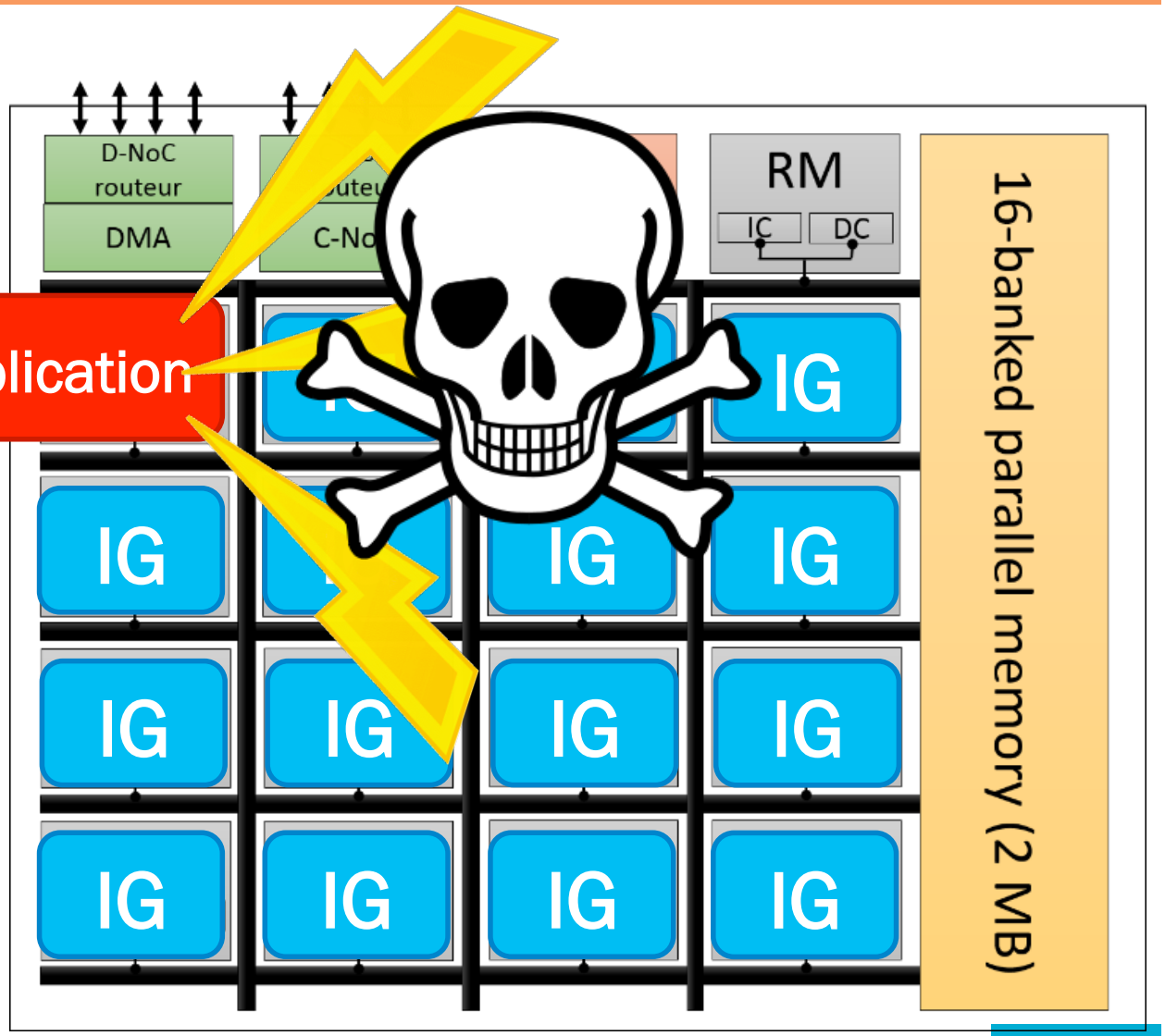


Isolation High-perf:
Max = 523 M
Variation = 377 k
= 0,07 %

Isolation Low-perf:
Max = 3.071 M
Variation = 1 M
= 0,03%

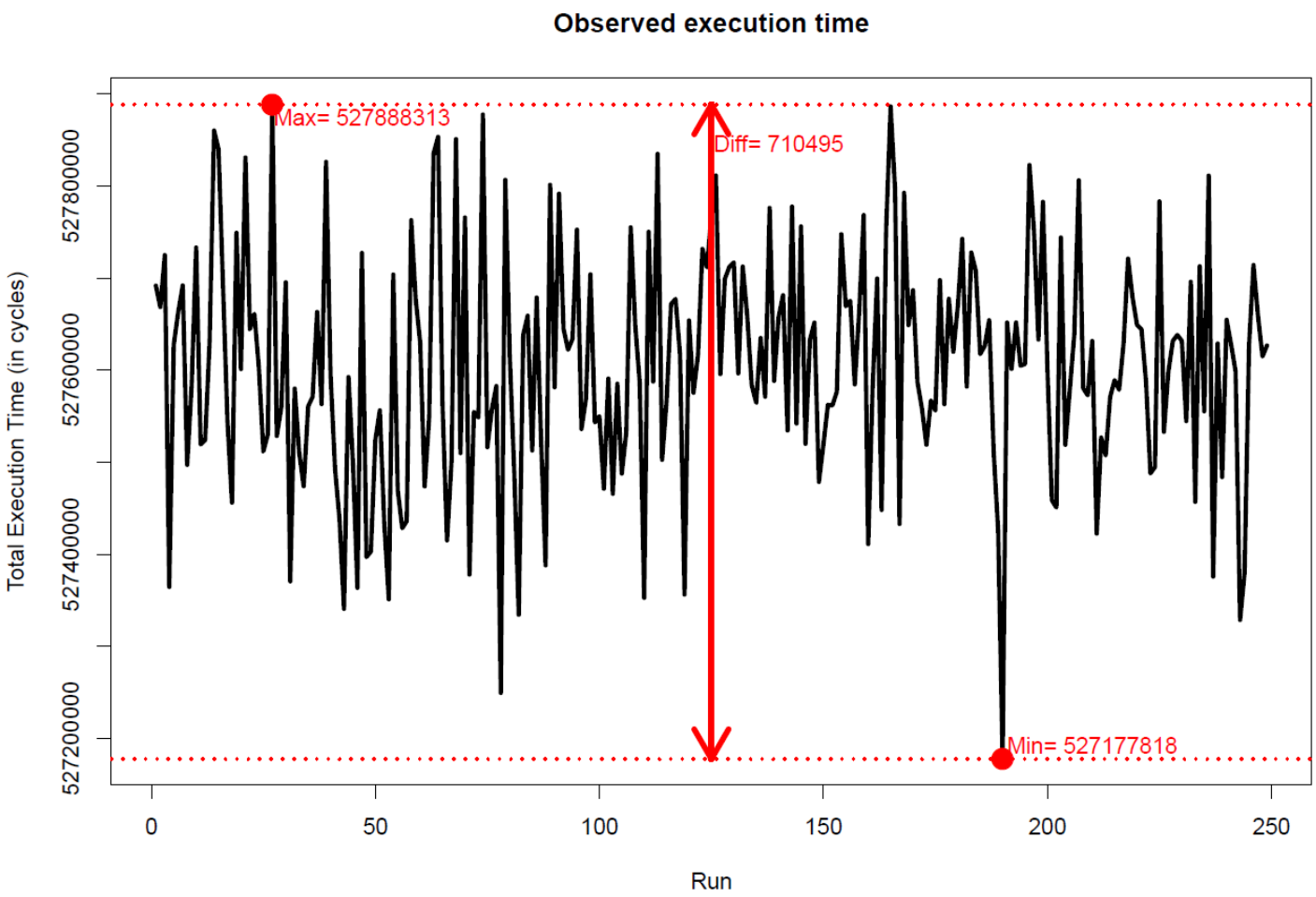
The contention mode

TACLeBench benchmark suite



<code/> + API

The contention mode – high-perf



Isolation High-perf:
Max = 523 M
Variation = 377 k
= 0,07 %

Isolation Low-perf:
Max = 3.071 M
Variation = 1 M
= 0,03%

Contention High-perf
Max = 527 M
Variation = 710 k
= 0,13%

Variability in High-perf config.

Contention mode

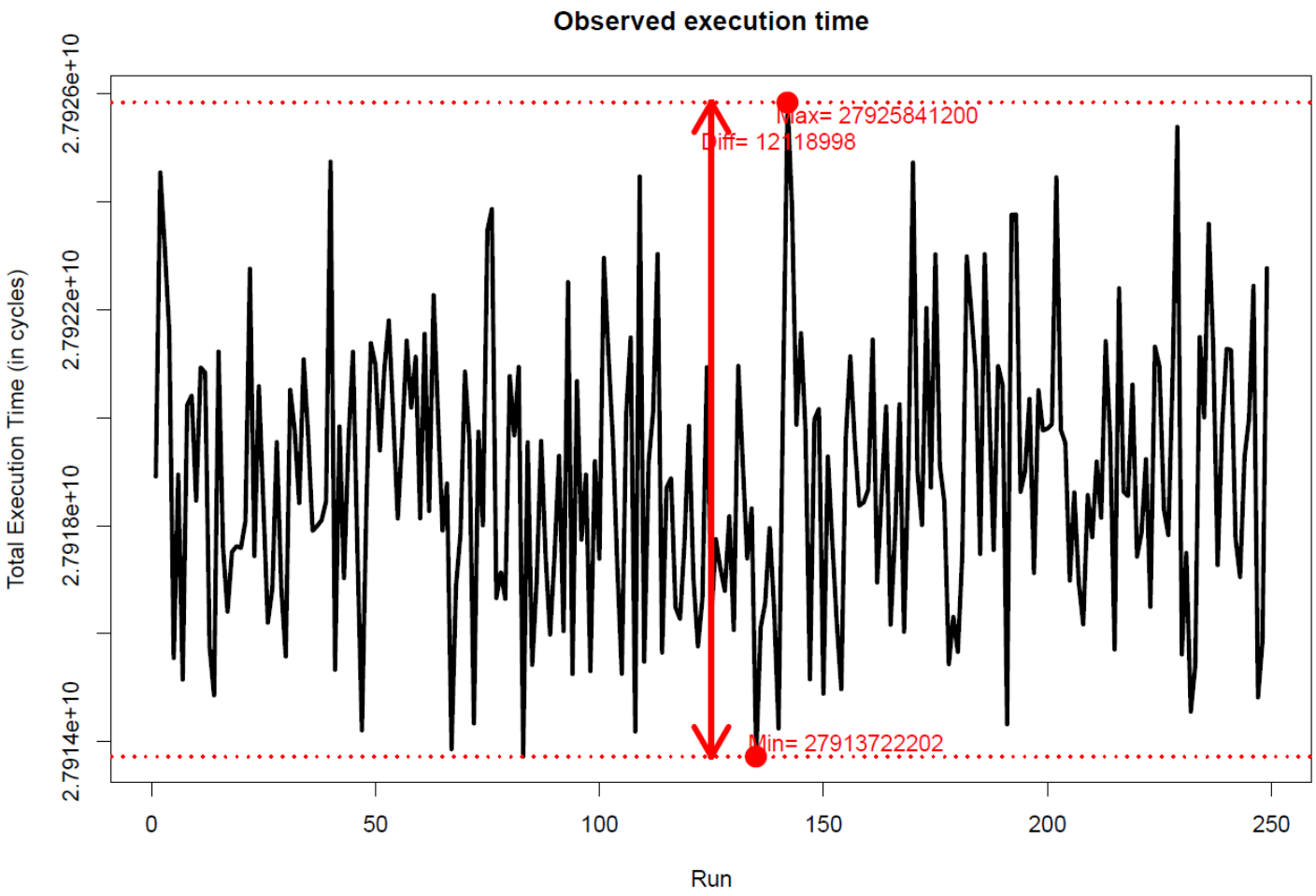
Isolation mode

Isolation High-perf:
Max = 523 M
Variation = 377 k
= 0,07 %

Isolation Low-perf:
Max = 3.071 M
Variation = 1 M
= 0,03%

Contention High-perf
Max = 527 M
Variation = 710 k
= 0,13%

The contention mode – low-perf



Isolation High-perf:
Max = 523 M
Variation = 377 k
= 0,07 %

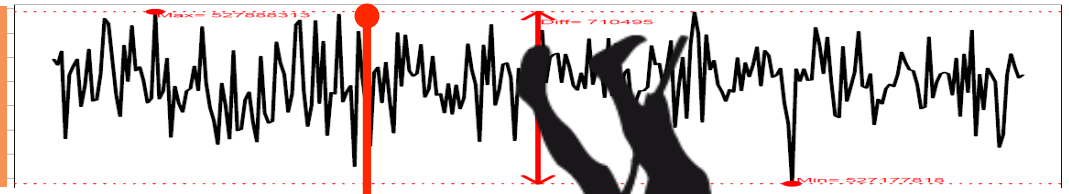
Isolation Low-perf:
Max = 3.071 M
Variation = 1 M
= 0,03%

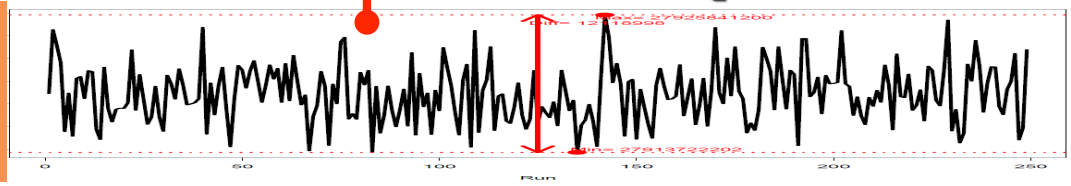
Contention High-perf
Max = 527 M
Variation = 710 k
= 0,13%

Contention Low-perf
Max = 27.925 M
Variation = 12 M
= 0,04%

Variability in Low-perf config.

Contention mode





Isolation mode

Isolation High-perf:
Max = 523 M
Variation = 377 k
= 0,07 %

Isolation Low-perf:
Max = 3.071 M
Variation = 1 M
= 0,03%

Contention High-perf
Max = 527 M
Variation = 710 k
= 0,13%

Contention Low-perf
Max = 27.925 M
Variation = 12 M
= 0,04%

Concluding remarks

From the traces obtained in isolation and contention modes, we showed that the analyzed application can actually be very sensitive to concurrent activity.

1. This work provides good insights on the amplitude of the variability of application execution times.

2. This work allows us to make recommandations on the environment set up to favor performance or predictability.

Future directions

1. What is the maximum slow-down factor that could be experienced at runtime?
2. What is the relation between the slow-down factor and the memory access pattern of the analyzed program?
3. How to totally isolate processes from each other without degrating too much the performance?

Questions?

Thank you very much for your attention!

Até Logo!