Smoothing on Dynamic Concurrency Throttling

Janaína Schwarzrock[1], Hiago Mayk G. de A. Rocha[1], Arthur F. Lorenzon[2], Antonio Carlos S. Beck[1]

[1] Federal University of Rio Grande do Sul (UFRGS), Brazil
[2] Federal University of Pampa, Brazil

Seventeenth International Workshop on Automatic Performance Tuning (iWAPT2022)

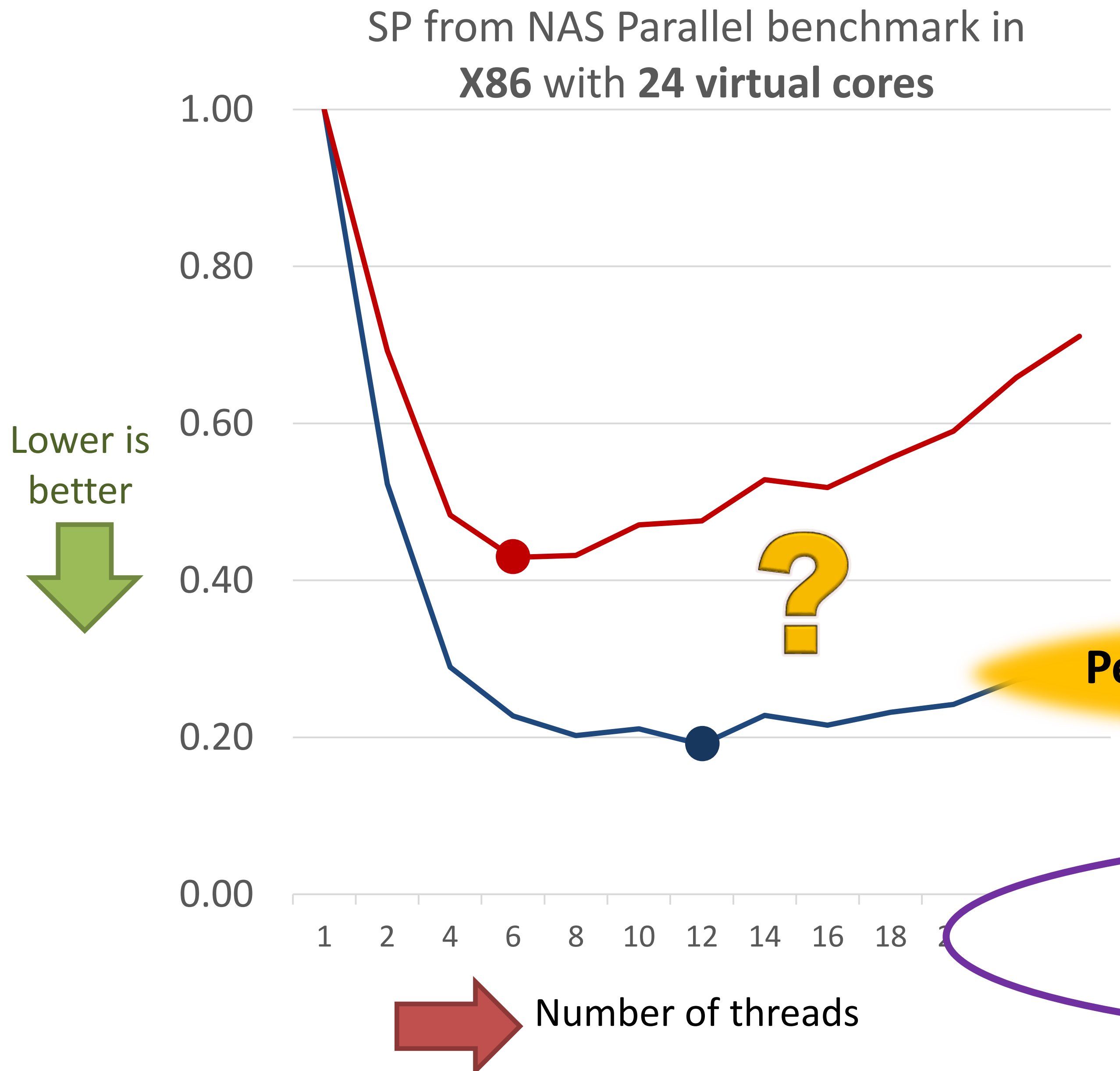
Outline

- Introduction
- Motivation
- Smoothing on DCT
- Experimental Setup
- Evaluation
- Final consideration

Outline

- **Introduction**
- Motivation
- Smoothing on DCT
- Experimental Setup
- Evaluation
- Final consideration

Parallel applications scalability



- Some applications do not scale as the number of threads increase

Execution Time

Energy consumption

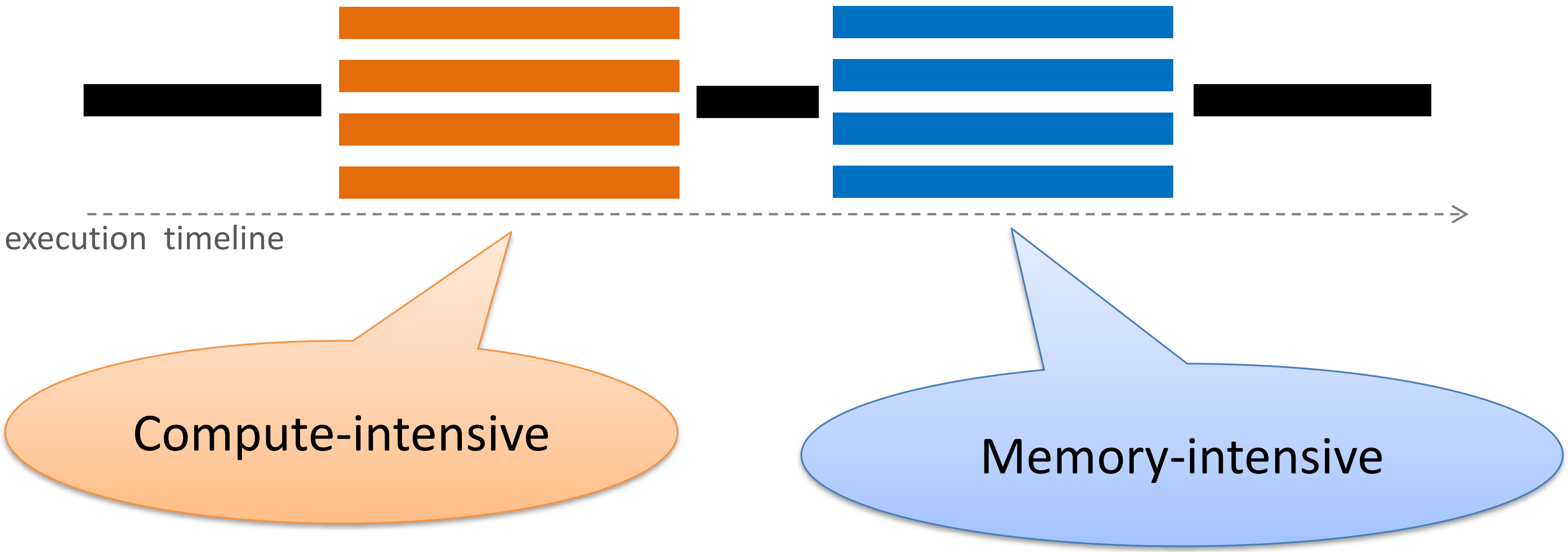
Performance or Energy?

Energy-Delay Product (EDP)

EDP = *Energy consumption * Execution Time*

Different parallel regions of an application

- Usually, a parallel application has **more** than one parallel region
- **Each parallel region may exhibit different behavior**



They may have a different optimal number of threads

It lacks adaptability

Tuning thread count approaches

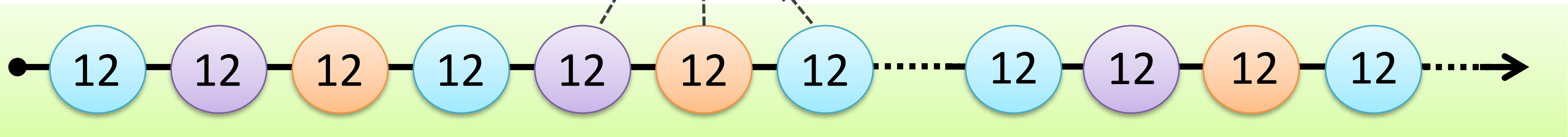
Offline

Search phase: Before execution

Static

Execution of parallel regions

Execution timeline

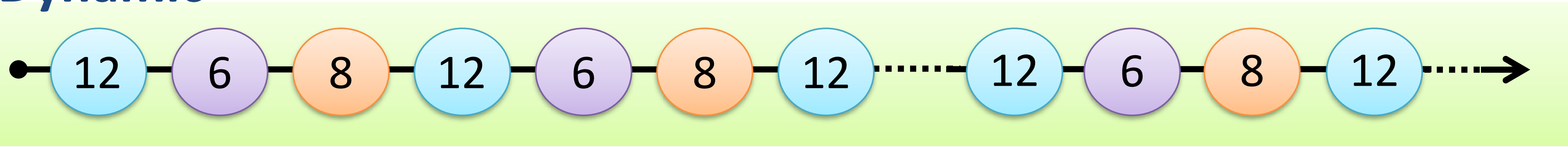


t* = 12

Dynamic

Search phase: Before execution

Execution timeline



$\mathcal{B}$ = (12, 6, 8)

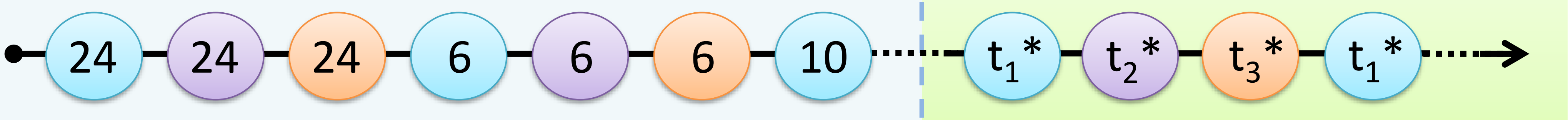
Online
Dynamic

It can adapt to any changes at run-time

Search phase: During execution

Stable phase

Execution timeline



$\mathcal{B} = (t_1^*, ..., t_k^*)$

Tuning thread count approaches

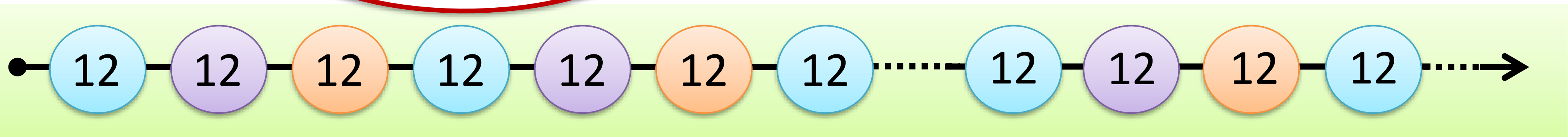
It lacks adaptability

Offline

Static

Pusukuri et al. (2011);
De Sensi (2016)

Search phase: Before execution

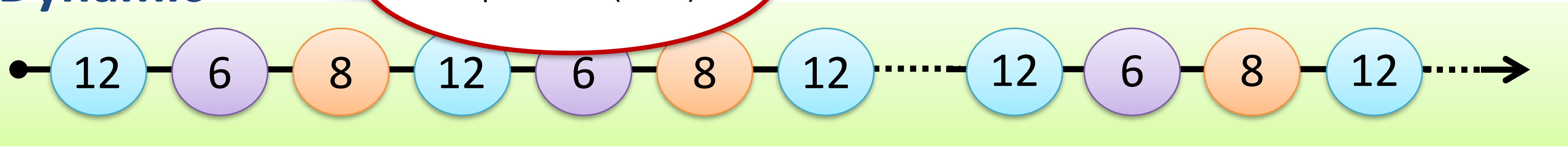


t* = 12

Dynamic

Wang et al. (2016);
Popov et al. (2019)

Search phase: Before execution



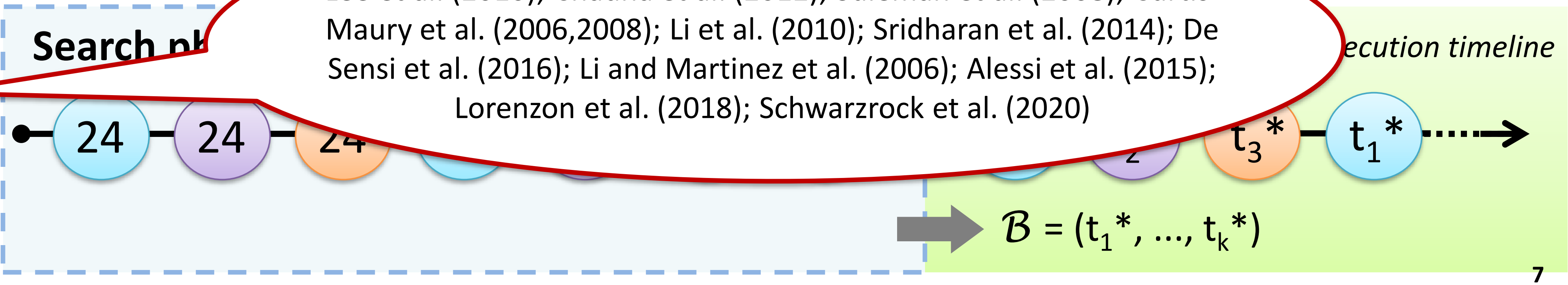
$\mathcal{B}$ = (12, 6, 8)

Online

Dynamic

It can adapt to any changes at run-time

Lee et al. (2010); Chadha et al. (2012); Suleman et al. (2008); Curtis-Maury et al. (2006,2008); Li et al. (2010); Sridharan et al. (2014); De Sensi et al. (2016); Li and Martinez et al. (2006); Alessi et al. (2015); Lorenzon et al. (2018); Schwarzrock et al. (2020)



$\mathcal{B}$ = (t_1*, ..., t_k*)

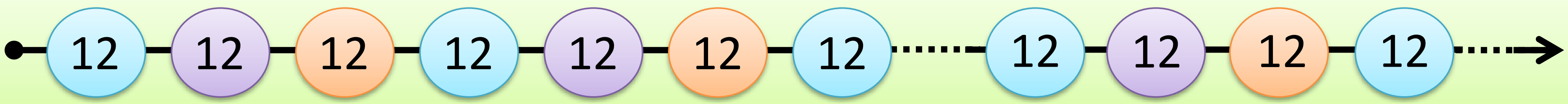
Tuning thread count approaches

Offline

Static *a single thread count that optimizes **the whole application*** **O (n)**

Upper-bound

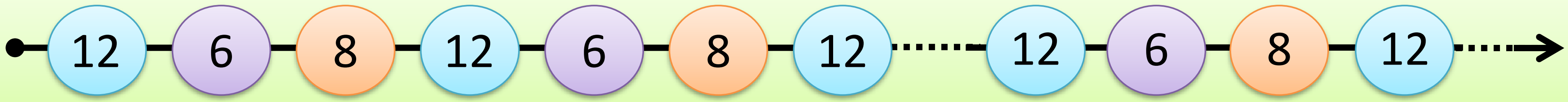
ution timeline



Search phase: Before execution → t* = 12

Dynamic *a set that optimizes **the whole application*** **O (n^k)** → unfeasible

Execution timeline



Search phase: Before execution → $\mathcal{B}$ = (12, 6, 8)

*An optimized thread count for **each parallel region individually*** **O (n) + O (n) + O (n)**

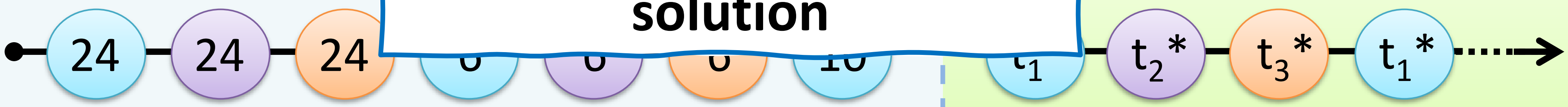
Online
Dynamic

Search phase: Duri

$\mathcal{B}$ = **Best-effort dynamic solution**

e phase

Execution timeline



→ $\mathcal{B} = (t_1^*, \ldots, t_k^*)$

Outline

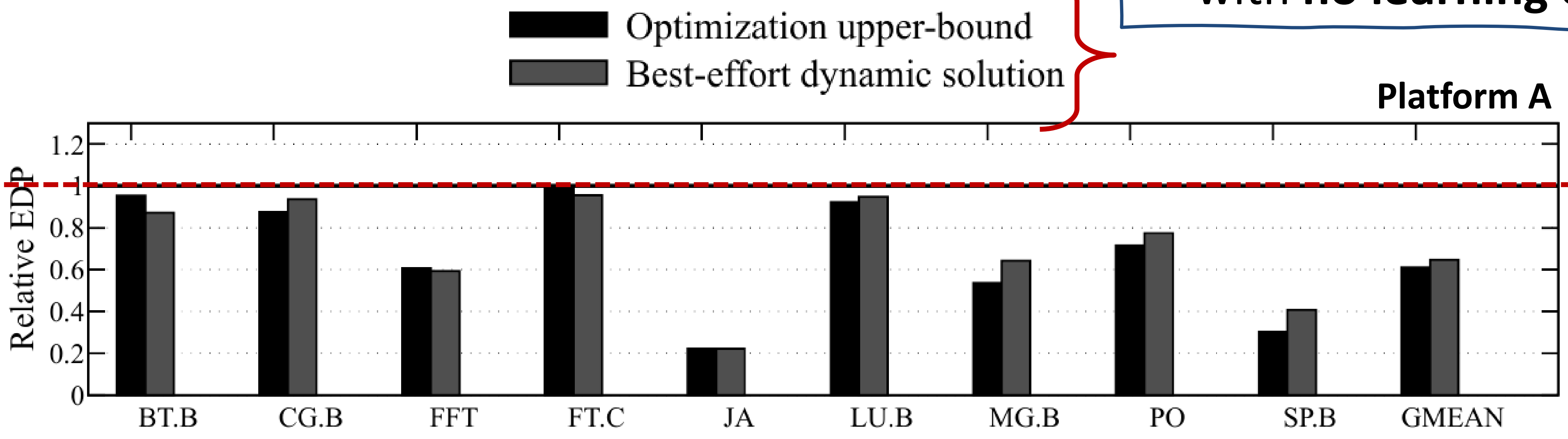
- Introduction
- Experimental Setup
- **Motivation**
- Smoothing on DCT
- Evaluation
- Final consideration

Motivation

Offline learning to get results with **no learning overhead**

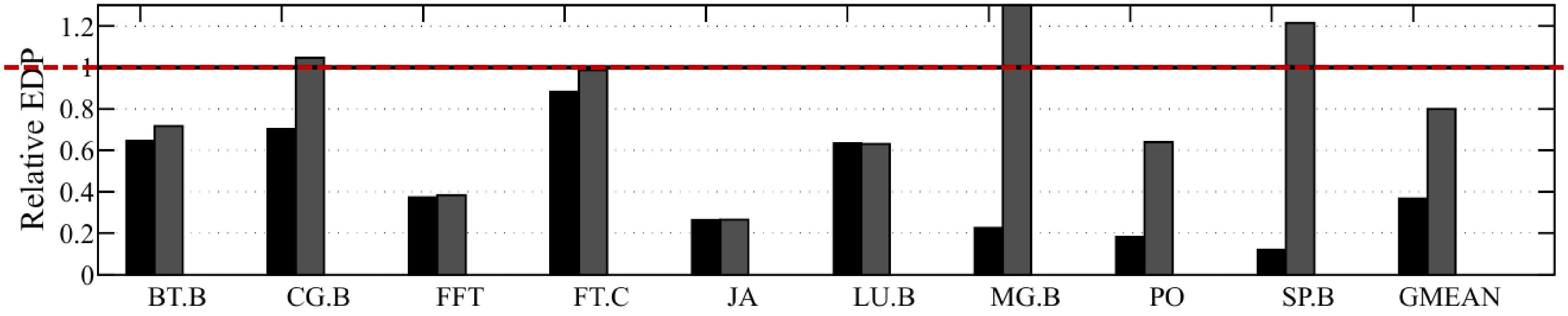
Baseline (the default execution): Execution with the **maximum number of threads**

Platform A



Parallel applications

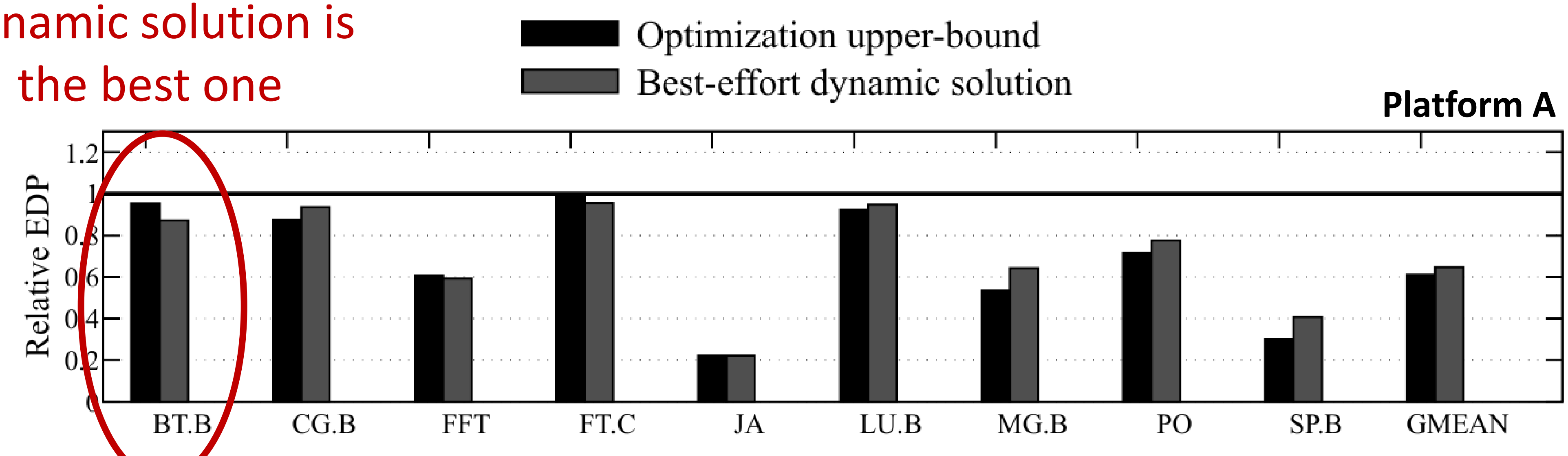
Platform B



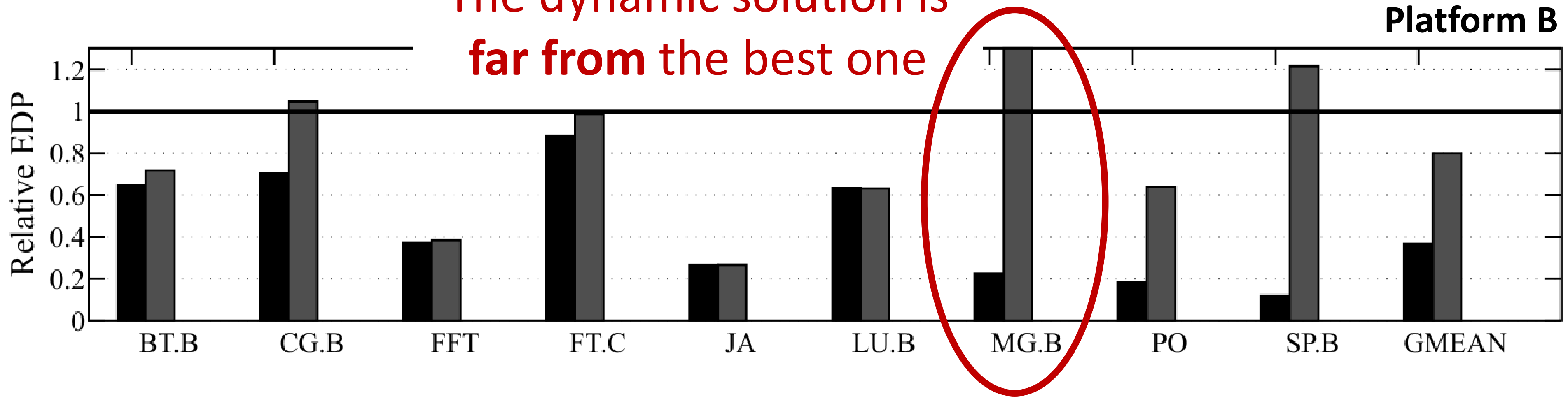
Parallel applications

Motivation

Dynamic solution is
the best one

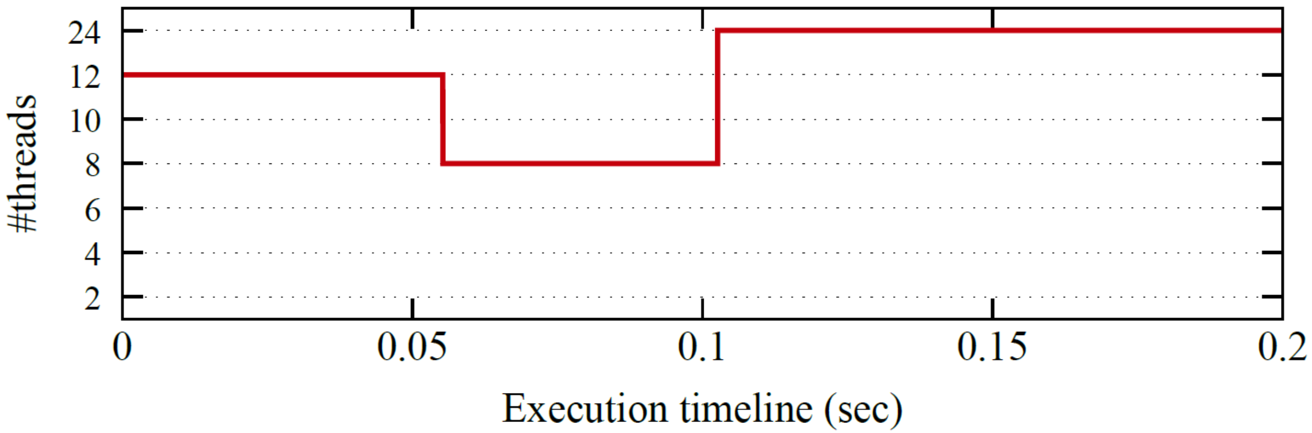


The dynamic solution is
far from the best one



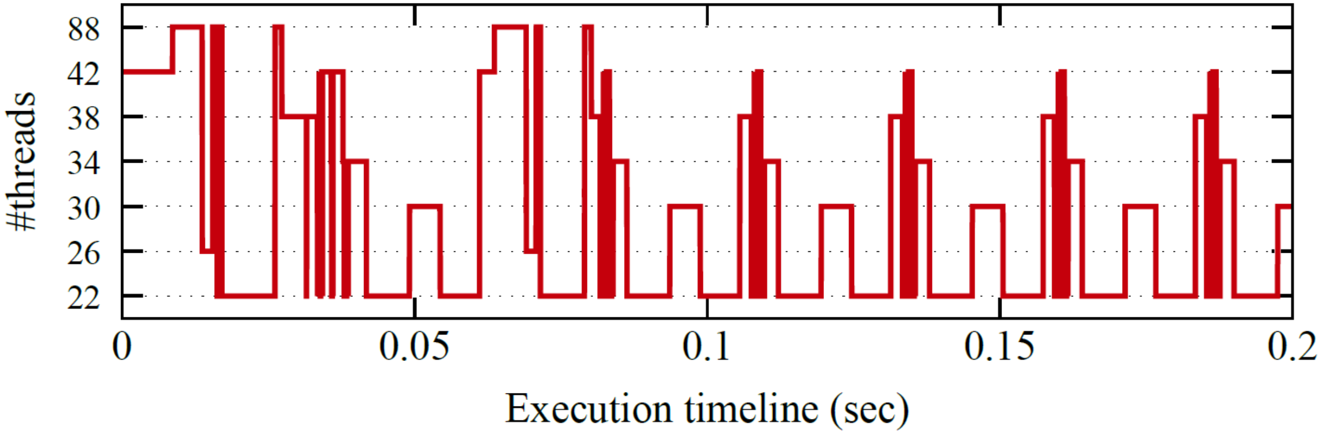
Motivation

BT on machine A



Dynamic solution is the best one

MG on machine B



The dynamic solution is **far from** the best one

When the thread count changes very often, the benefit of using the best configuration for each parallel region may not compensate for the switching cost

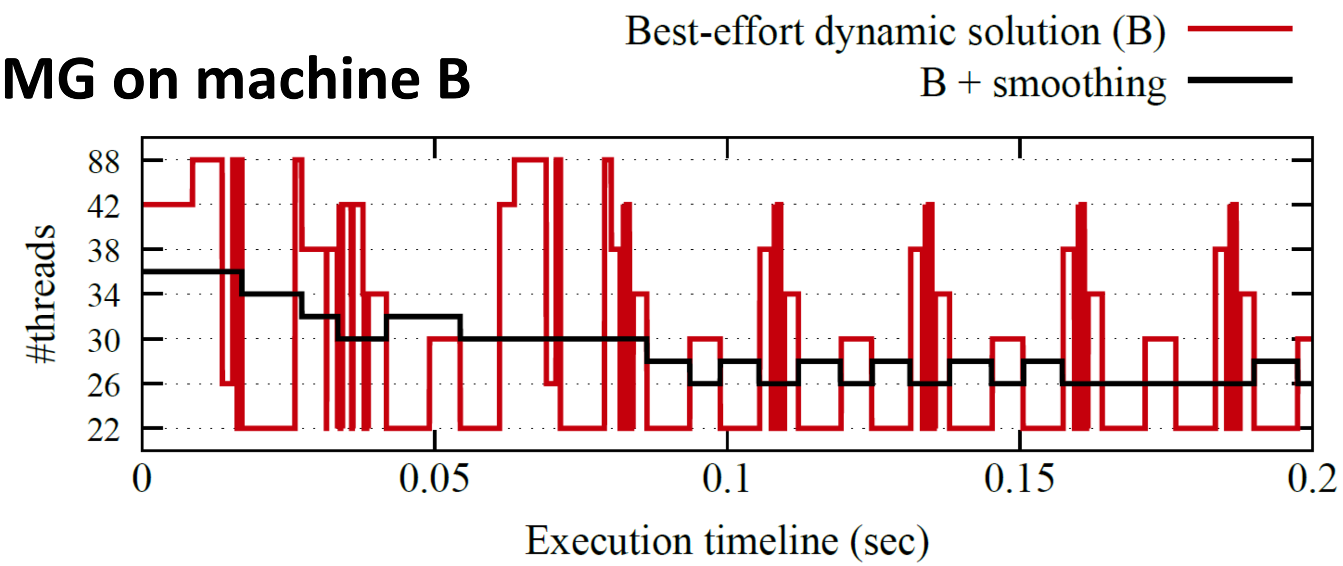
Creating/destroying/migrating threads; data warm-up (memory caches warm-up, TLB misses)

Our proposal: smoothing thread count changes

- It alleviates the switching overheads.
- Our proposal is generic and aims further to improve the optimization results of any DCT technique (offline and online).

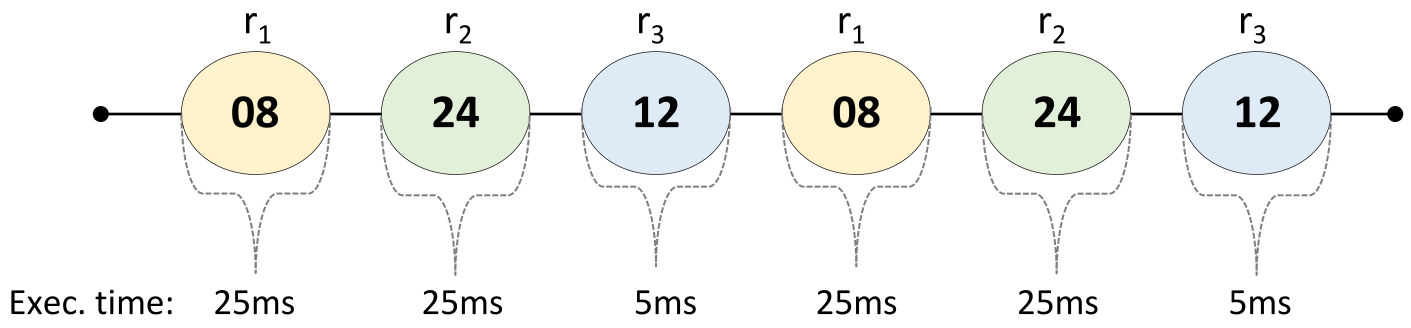
We propose a smoothing-based strategy to minimize the thread count changes

MG on machine B



Outline

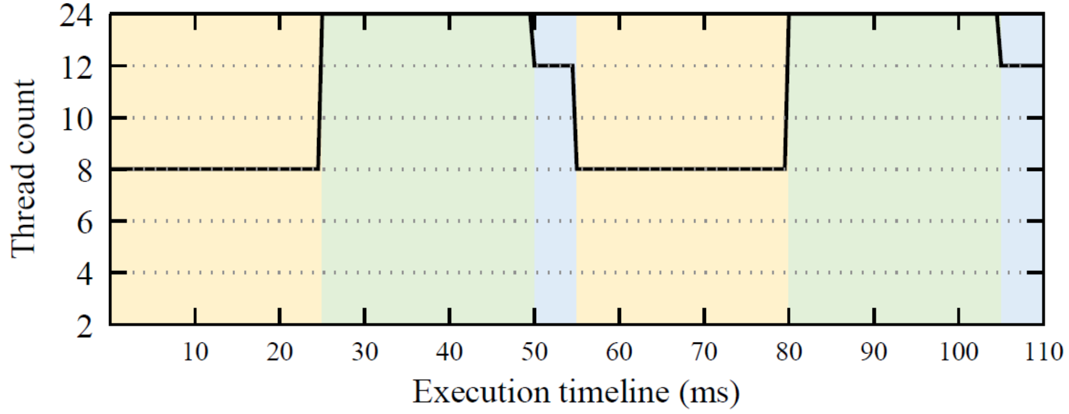
- Introduction
- Experimental Setup
- Motivation
- **Smoothing on DCT**
- Evaluation
- Final consideration



Parallel region	Best #threads
r1	08
r2	24
r3	12

$\mathcal{B}$ = (08, 24, 12)

Time series of thread count



Weighted Moving Average (WMA)

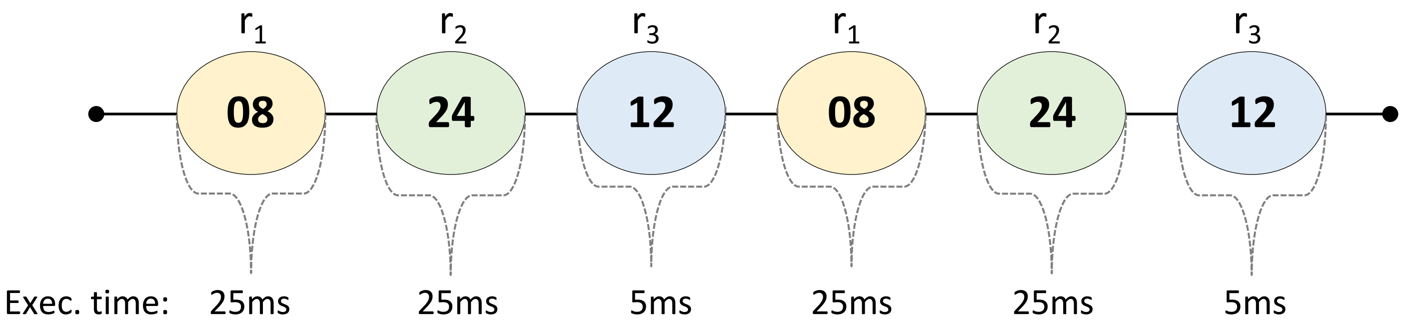
a lightweight and powerful smoothing technique

$$\mathcal{Y} = (y_1,\ y_2,\ y_3, \ldots,\ y_m)$$

$$\mathcal{E} = (w_1,\ w_2,\ w_3, \ldots,\ w_m)$$

$$\bar{y}_i = \frac{(y_i w_i) + (y_{i-1} w_{i-1}) + \cdots + (y_{i-n-1} w_{i-n-1})}{w_i + w_{i-1} + \cdots + w_{i-n-1}}$$

$$\bar{\mathcal{Y}} = (\bar{y}_1,\ \bar{y}_2,\ \bar{y}_3, \ldots,\ \bar{y}_m)$$

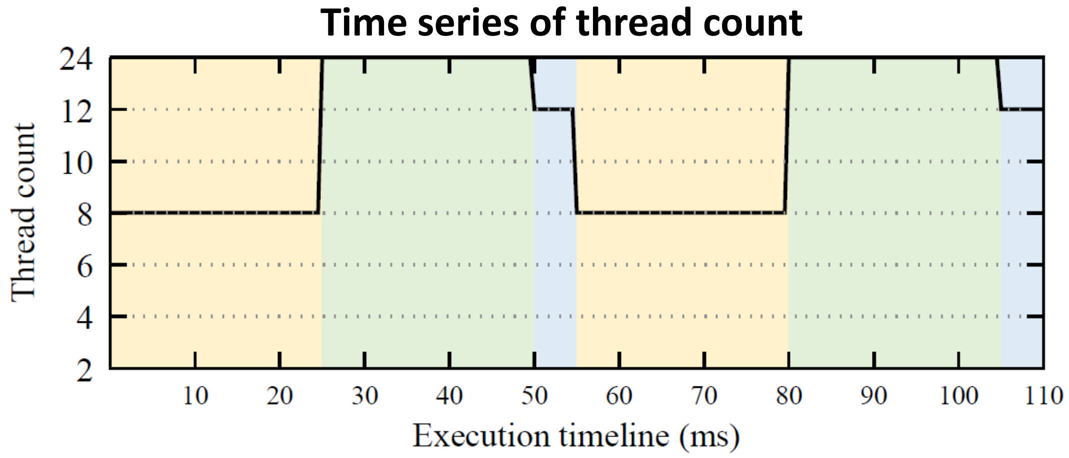


The time series (thread count):

$\mathcal{Y}$ = (08, 24, 12, 08, 24, 12)

The weights (exec. time):

$\mathcal{E}$ = (25, 25, 5, 25, 25, 5)



Weighted Moving Average (WMA)

a lightweight and powerful smoothing technique

$$\mathcal{Y} = (y_1,\ y_2,\ y_3, \ldots,\ y_m)$$

$$\mathcal{E} = (w_1,\ w_2,\ w_3, \ldots,\ w_m)$$

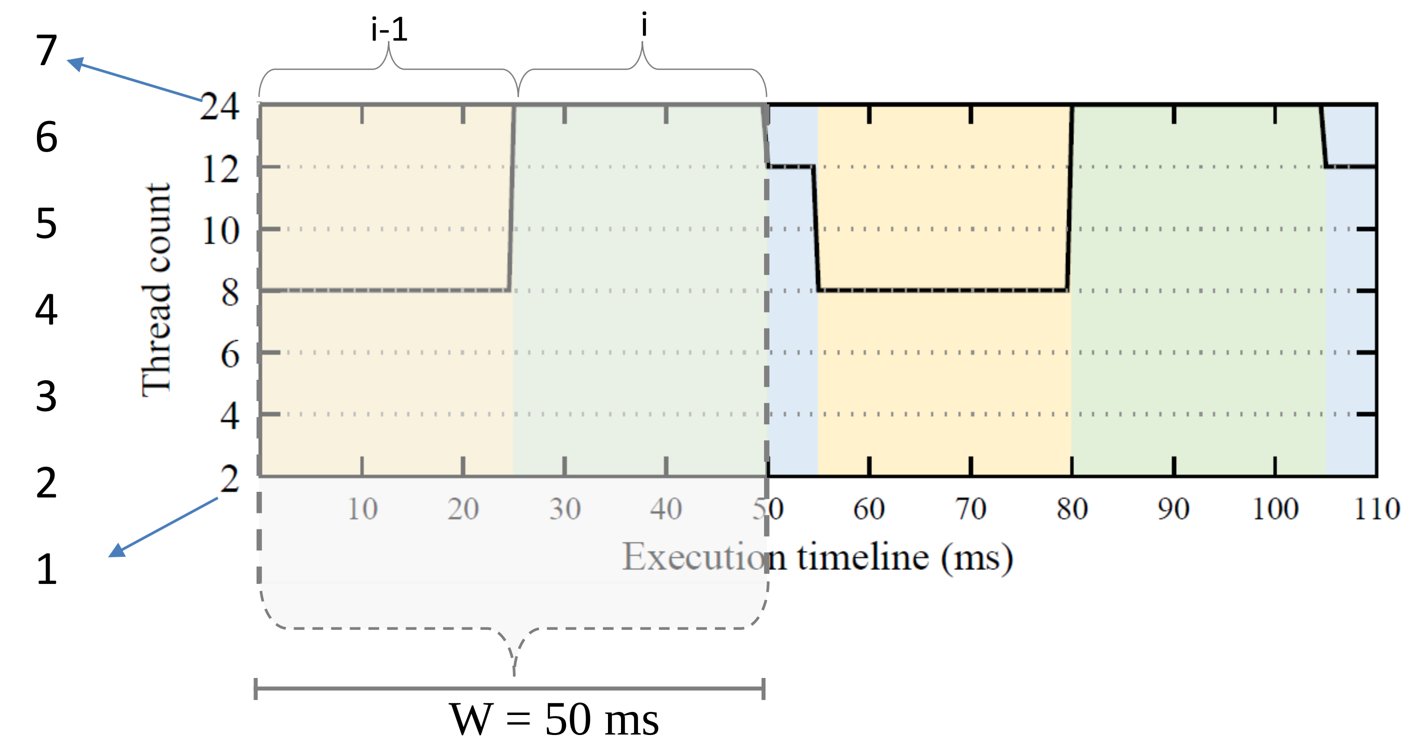
$$\bar{y}_i = \frac{(y_i w_i) + (y_{i-1} w_{i-1}) + \cdots + (y_{i-n-1} w_{i-n-1})}{w_i + w_{i-1} + \cdots + w_{i-n-1}}$$

$$\bar{\mathcal{Y}} = (\bar{y}_1,\ \bar{y}_2,\ \bar{y}_3, \ldots,\ \bar{y}_m)$$

points:

y = (04, 07, 06, 04, 07, 06)

$$\bar{\mathcal{Y}} = (\,8\,,\,12\,,\,\bar{y}_3, \ldots, \bar{y}_m)$$



Round to 6

Index 6 = 12 threads

Index best #threads

Time

$$\bar{y}_2 = \frac{(7 \times 25) + (4 \times 25)}{50} = 5.5$$

$$\bar{y}_i = \frac{(y_i w_i) + (y_{i-1} w_{i-1}) + \cdots + (y_{i-n-1} w_{i-n-1})}{w_i + w_{i-1} + \cdots + w_{i-n-1}}$$

W

Outline

- Introduction
- Motivation
- Smoothing on DCT
- **Experimental Setup**
- Evaluation
- Final consideration

Execution Environment

.Inf INSTITUTO DE INFORMÁTICA UFRGS

Machine	A	B
Processor	Intel Xeon E5-2630 (Sandy Bridge) 2.3GHz	Intel Xeon E5-2699v4 (Broadwell) 2.2 GHz
#Sockets (#nodes)	**2**	**2**
#Cores per socket	**6 (2-way SMT)**	**22 (2-way SMT)**
#Threads total	**24**	**88**
L1 cache (private)	12 x 32KB	44 x 32KB
L2 cache (private)	12 x 256KB	44 x 256KB
L3 cache (shared)	2 x 15MB	2 x 55MB
RAM Memory	2 x 16GB	2 x 128GB

- OS Linux kernel v. 4.19.0.

Thread count search space:

Machine A: `2, 4, 6, 8, 10, 12 and 24`

Machine B: `2, 4, 6, 8, .., 44 and 88`

physical cores (only **even** numbers)

the maximum number of threads

Benchmarks

- **9 OpenMP Parallel Applications written in C/C++:**

Six kernels from the NAS Parallel Benchmark:

- BT, CG, FT, LU, MG, and SP

Three applications from different domains:

- Fast Fourier Transform (FFT);
- Jacobi (JA);
- Poisson (PO).

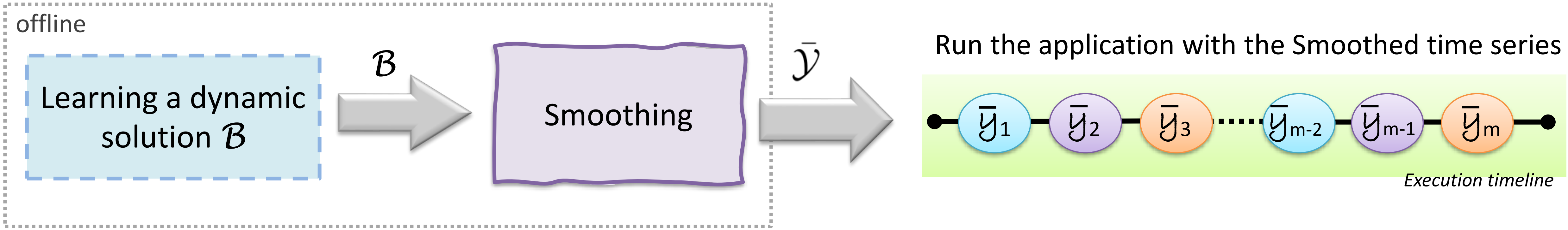
- GCC version 8.3 (OpenMP 4.5) with –O3

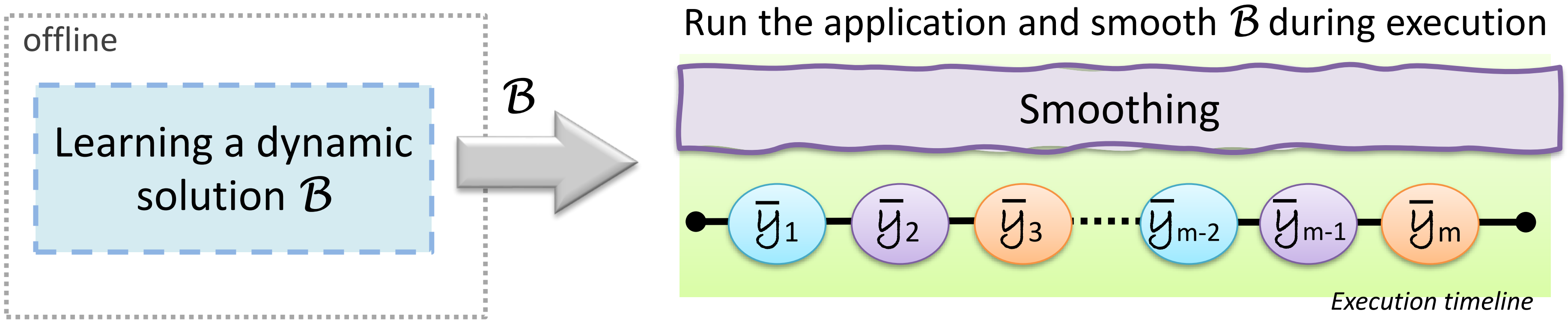
Benchmark	Input
BT	Class B
CG	Class B
FT	Class C
LU	Class B
MG	Class B
SP	Class B
FFT	Array of 10000 elements
JA	Square matrix of 8192
PO	Square matrix of 768

Outline

- Introduction
- Experimental Setup
- Motivation
- Smoothing on DCT
- **Evaluation**
- Final consideration

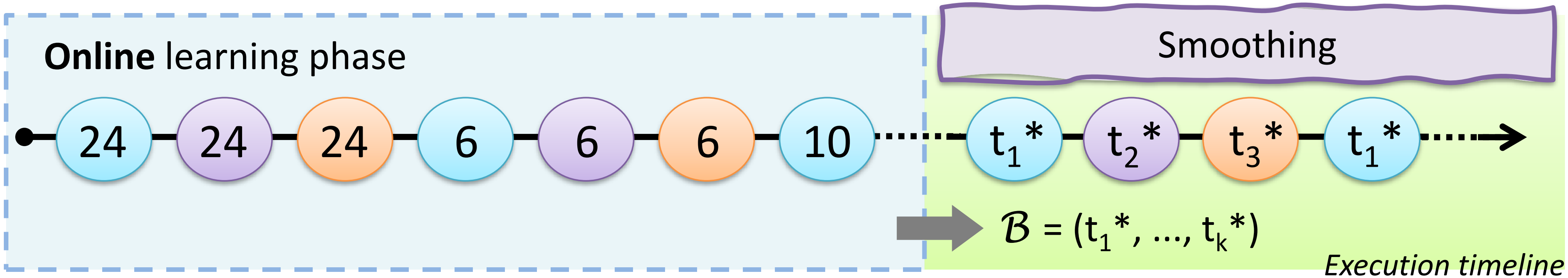
a) Evaluate the effectiveness of the smoothing technique (without online cost):





b) Evaluate the online smoothing overhead

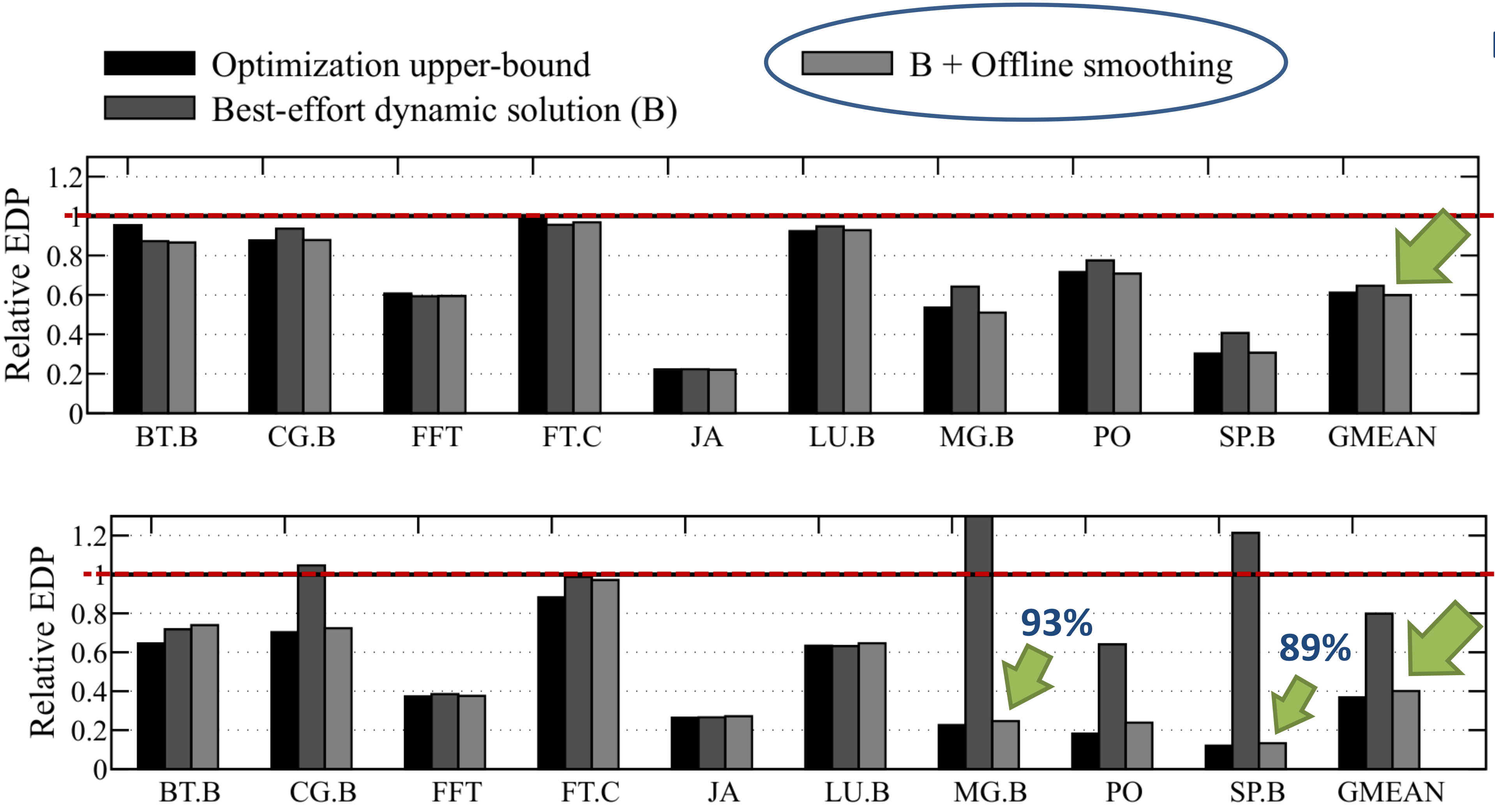
c) Evaluate the online smoothing into a DCT online learning technique



a) the effectiveness of the smoothing technique

It improves B' results

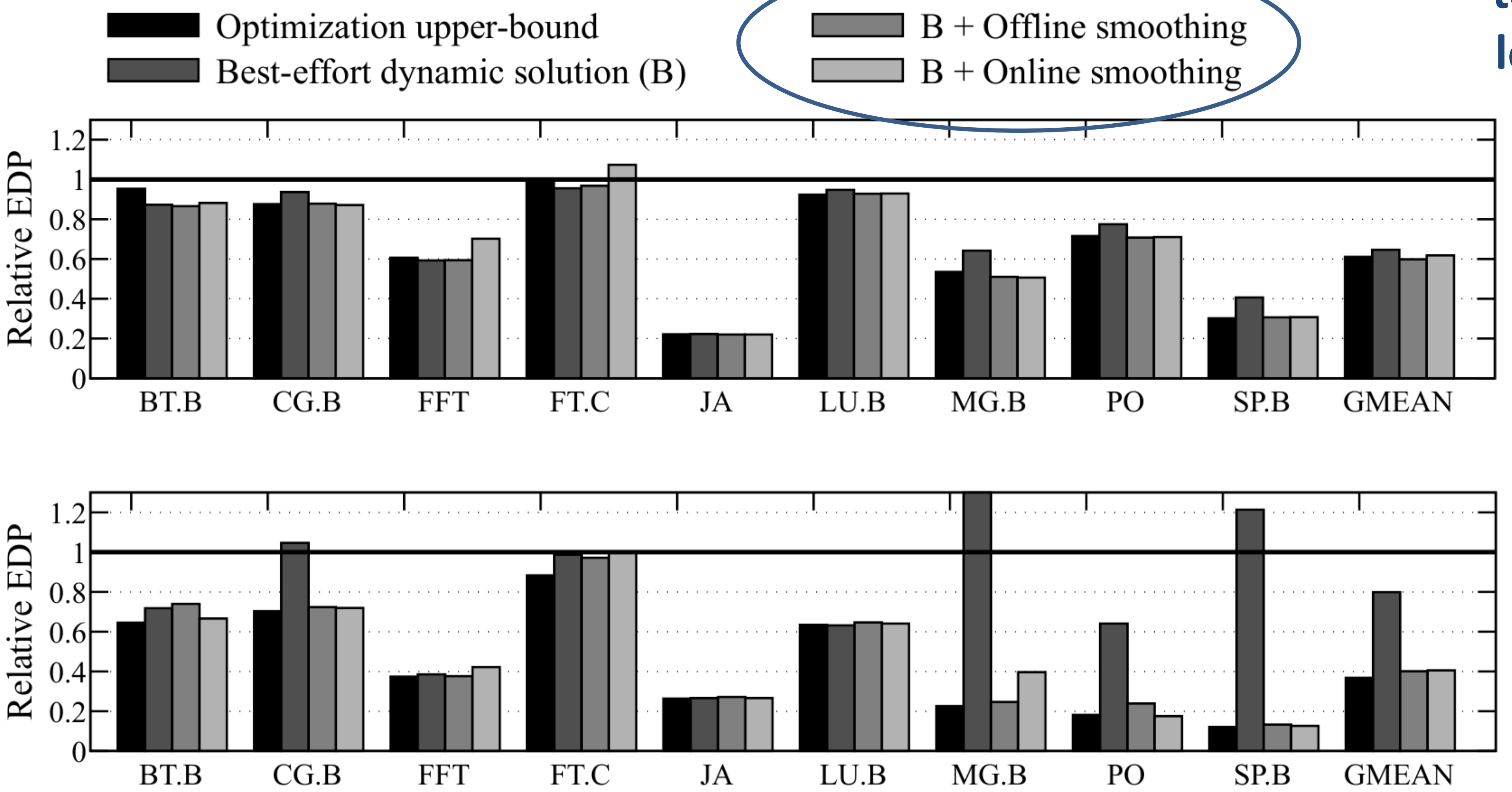
EDP optimization near the upper-bound



Platform A
(24 hw threads)

Platform B
(88 hw threads)

b) the online smoothing overhead



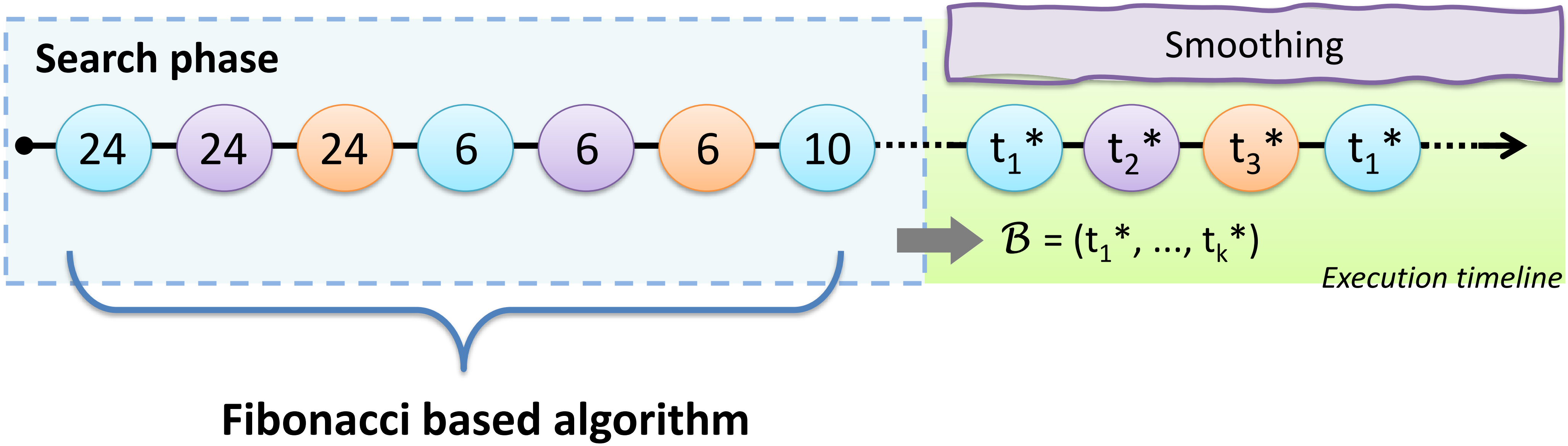
Our online smoothing technique has low overhead

Platform A
(24 hw threads)

Platform B
(88 hw threads)

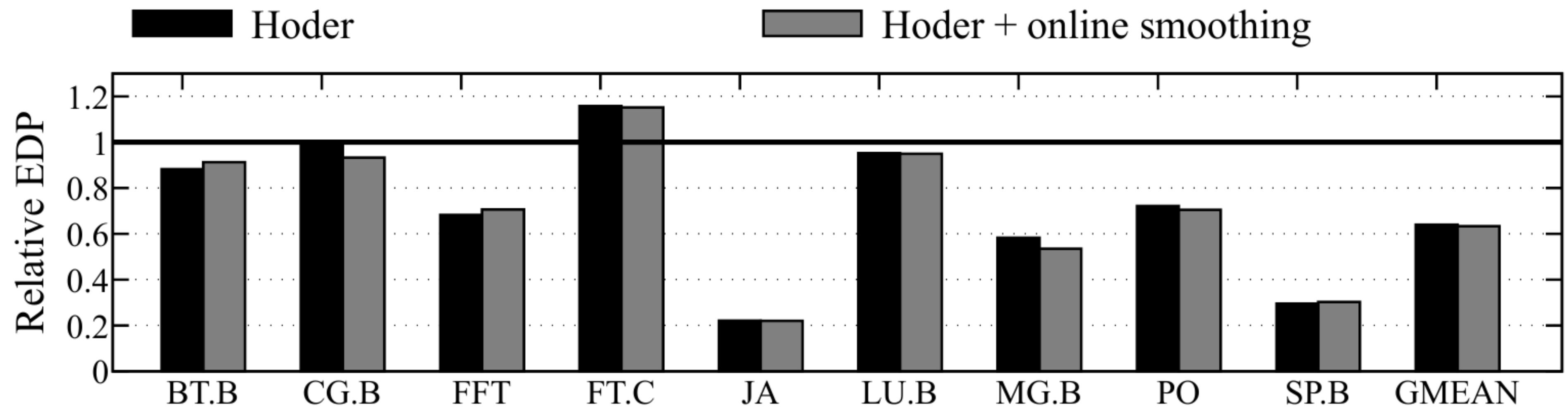
c) Smoothing into a DCT online learning technique

- Online learning DCT technique
- Hoder [1]



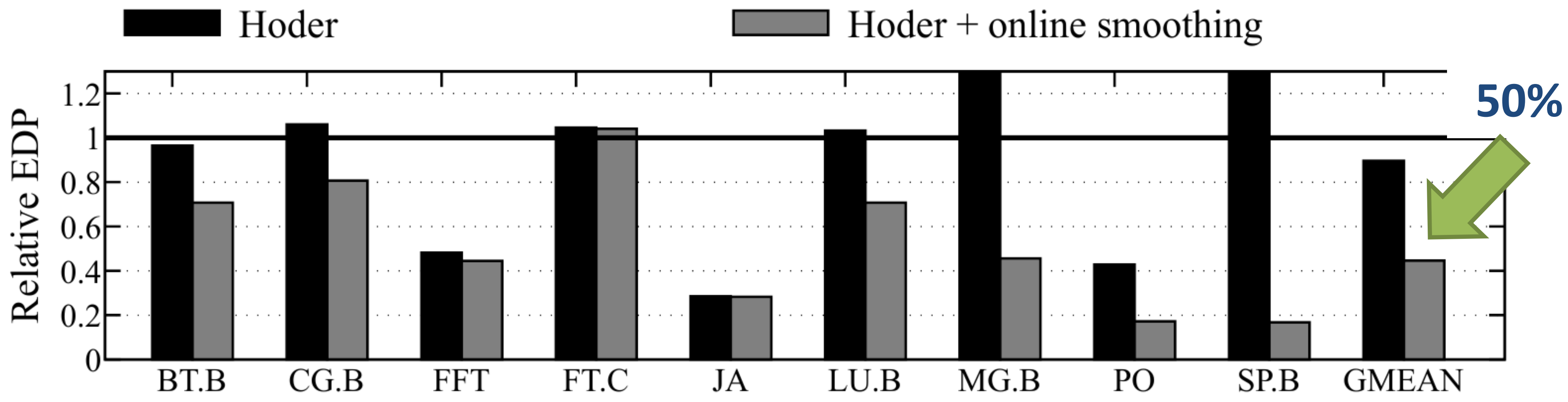
[1] J. Schwarzrock, C. C. de Oliveira, M. Ritt, A. F. Lorenzon, and A. C. S. Beck, "A runtime and non-intrusive approach to optimize edp by tuning threads and cpu frequency for openmp applications," IEEE TPDS, vol. 32, no. 7, pp. 1713–1724, 2020

c) Smoothing into a DCT online learning technique



Platform A
(24 hw threads)

large thread count
search space



Platform B
(88 hw threads)

Outline

- Introduction
- Motivation
- Smoothing on DCT
- Experimental Setup
- Evaluation
- **Final consideration**

Final consideration

- A smoothing-based strategy to further improve the optimization results of any DCT technique
- Our strategy smooths the thread count changes alleviating the switching overheads, which is generated by DCT when changing the number of threads during application execution
- Experiments on two multicore systems with nine well-known benchmarks show that our smoothing technique improves EDP results of offline and online state-of-the-art DCT techniques by up to 93% and 89% (overall mean of 22%), respectively.

Thanks for your attention!

Questions?

jschwarzrock@inf.ufrgs.br

Smoothing on Dynamic Concurrency Throttling

Janaína Schwarzrock[1], Hiago Mayk G. de A. Rocha[1], Arthur F. Lorenzon[2], Antonio Carlos S. Beck[1]

[1] Federal University of Rio Grande do Sul (UFRGS), Brazil
[2] Federal University of Pampa, Brazil