Analyzing the stability of relative performance differences between cloud and embedded environments

Rumen Kolev / Christopher Helpa

TTTechAuto

Agenda

01 Goal & Purpose

02 Concept

03 Benchmarks

04 Results

05 Summary

06 Open points & Outlook

01

Goal & Purpose

Goal & Purpose

- **Goal:** Find out how **stable** the relative performance difference factor between an **embedded CPU** and a **Cloud VM** is (PerfDiffFactor= $Runtime_{embedded}/Runtime_{VM}$)
- **Purpose:** if the performance factor somewhat stable (i.e. **low jitter**, no significant differences due to **input changes**), VMs running in a Cloud service can be used for testing, performance estimation with confidence

02

Concept

Concept (hardware)

- A **VM** running on Microsoft Azure with an underlying ARM CPU
- A Janicto processor, connected to a common processor board
- Specifications matched up closely, although **differences** in e.g. **cache** (64KB vs 32KB L1 data cache, 32MB SLC on VM), **clock frequency**

	Cloud VM	Embedded CPU
ARM CPU	Ampere Altra 64-Bit Multi-Core Processor (ARM v8.2+)	64-bit Dual-core Arm Cortex-A72 (ARMv8-A)
Number of cores	2	2
max. clock frequency	3300MHz	2000MHz
L1 cache	64KB DCache, 64KB ICache per core	32KB DCache, 48KB ICache per core
L2 cache	1MB per core	1MB shared per dual-core cluster
System-Level Cache (SLC)	32MB	-
RAM	8GiB	3.8GiB(4GiB), 512KB on-chip SRAM in MAIN domain
Operating System	Ubuntu 20.04.5 LTS, Kernel: 5.15.0-1034-azure	Arago 2021.09 (based on Yocto Linux), Kernel: 5.10.65-gdcc6bedb2c

Concept (software)

- VM OS: Ubuntu 20 LTS
- Embedded CPU OS: Yocto linux based (Arago)
- C++17
- shared object files for the C++ STD and OpenCV were shared between the two platforms
- Compiler optimization level set to –O0
- Benchmark binaries were compiled on the VM, as it uses a more stable Ubuntu LTS and then reused on the embedded CPU to avoid differences due to compilation

TTTechAuto

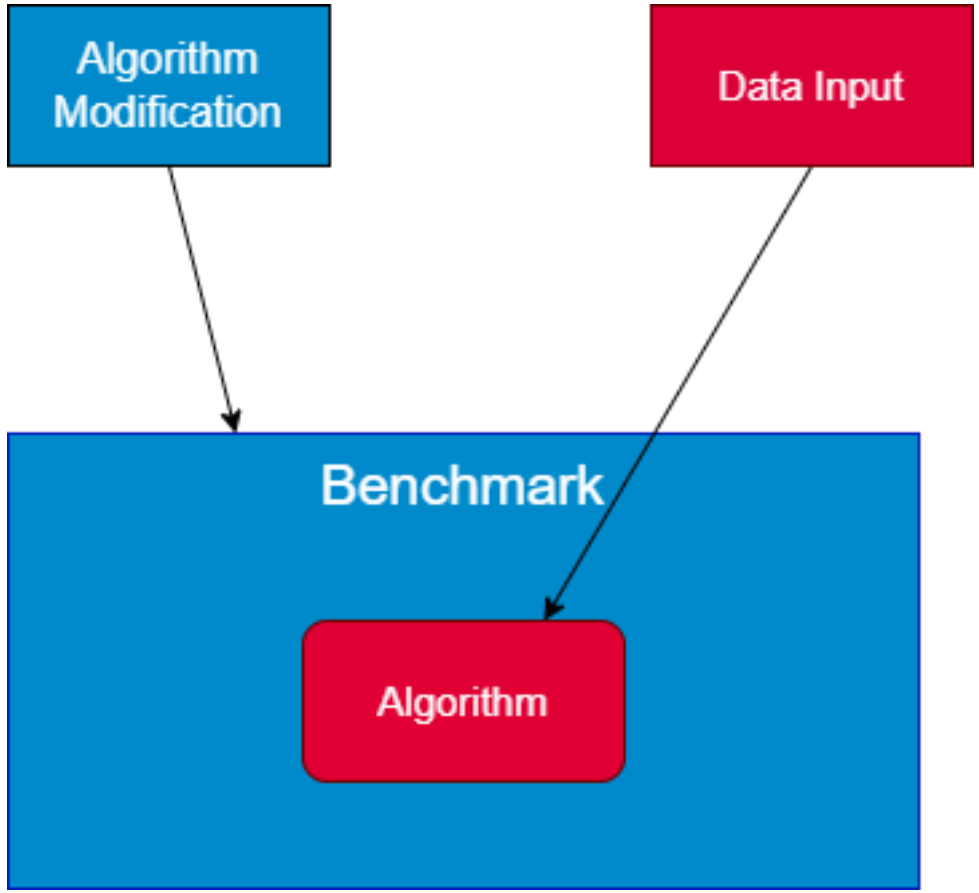
03

Benchmarks

Benchmarks

- Main requirement: **Identifiable inputs** for best-/average-/worst-case performance
- A* - Pathfinding algorithm, similar to Dijkstra's algorithm – uses a heuristic to determine which node to expand next (part of the input for our benchmarks)
- Contour detection using OpenCV
- CoreMark-PRO by EEMBC
  - JPEG/ZIP compression
  - XML parsing
  - SHA-256 Secure Hash Algorithm
  - list processing
  - matrix manipulation
  - state machine

Benchmarks

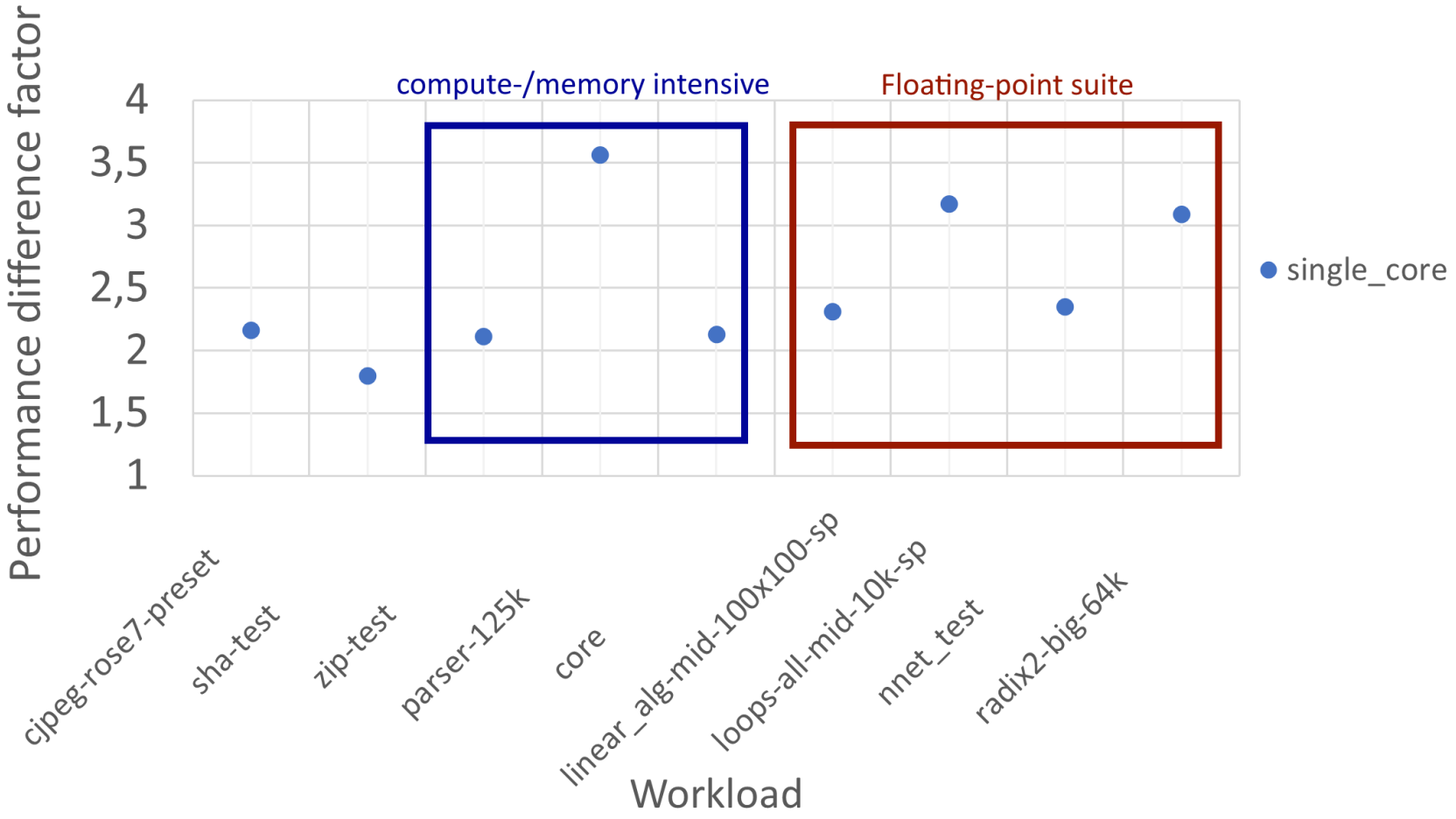


TTTechAuto

04

Results

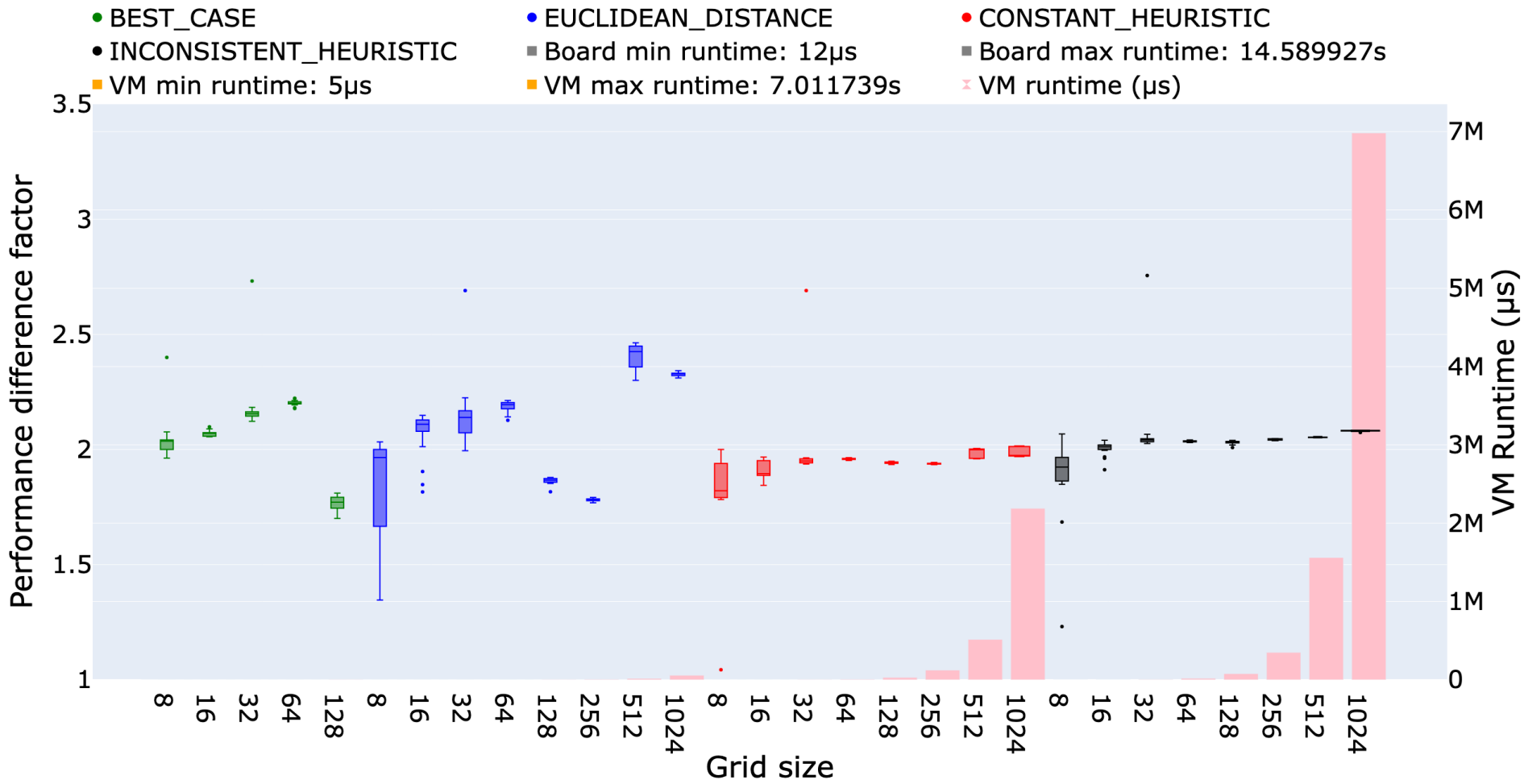
Results CoreMark-PRO



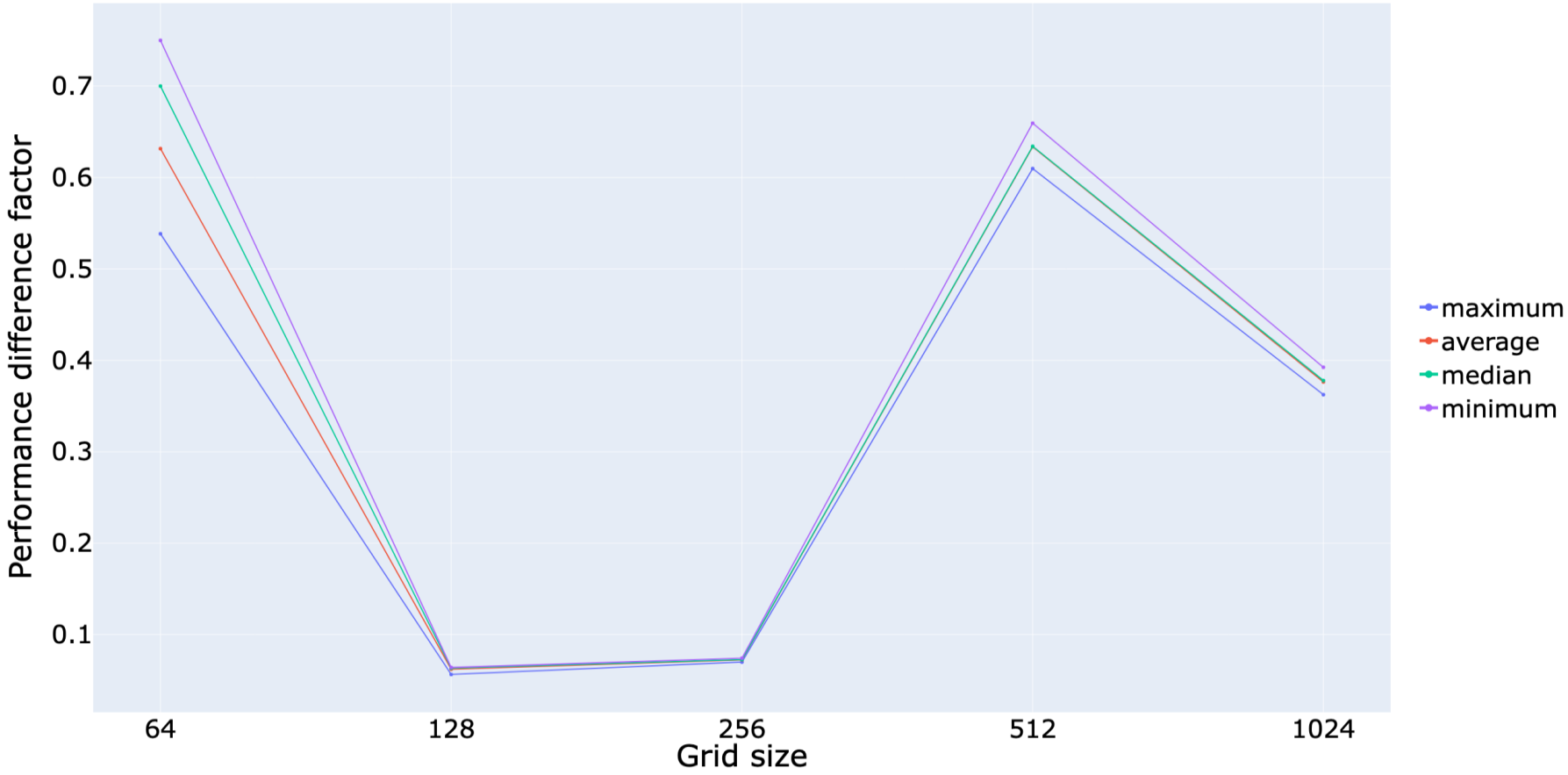
A* benchmark input variations

heuristic function	best-case	Euclidean distance	constant	inconsistent
algorithm behaviour	uses precalculated values for the actual distance to the goal cell	calculates the Euclidean distance to the goal cell	always estimates distance 0 to the goal	randomly estimates the distance between 0 and Euclidean distance to the goal (thus, admissible)
performance impact	best case	average case, good estimate, thus close to best-case	average case, bad estimate	worst-case

Results A* benchmark



Vector destruction outlier results



Contour Detection input variations

image classes			
image type	**solid-color images**	**dashcam pictures of street traffic**	**multicolored white noise**
algorithm behaviour	only image borders are considered contours	average real-life use case	very large number of neighbouring pixels with different colours => highest possible number of contours
performance impact	best case	average case	worst case

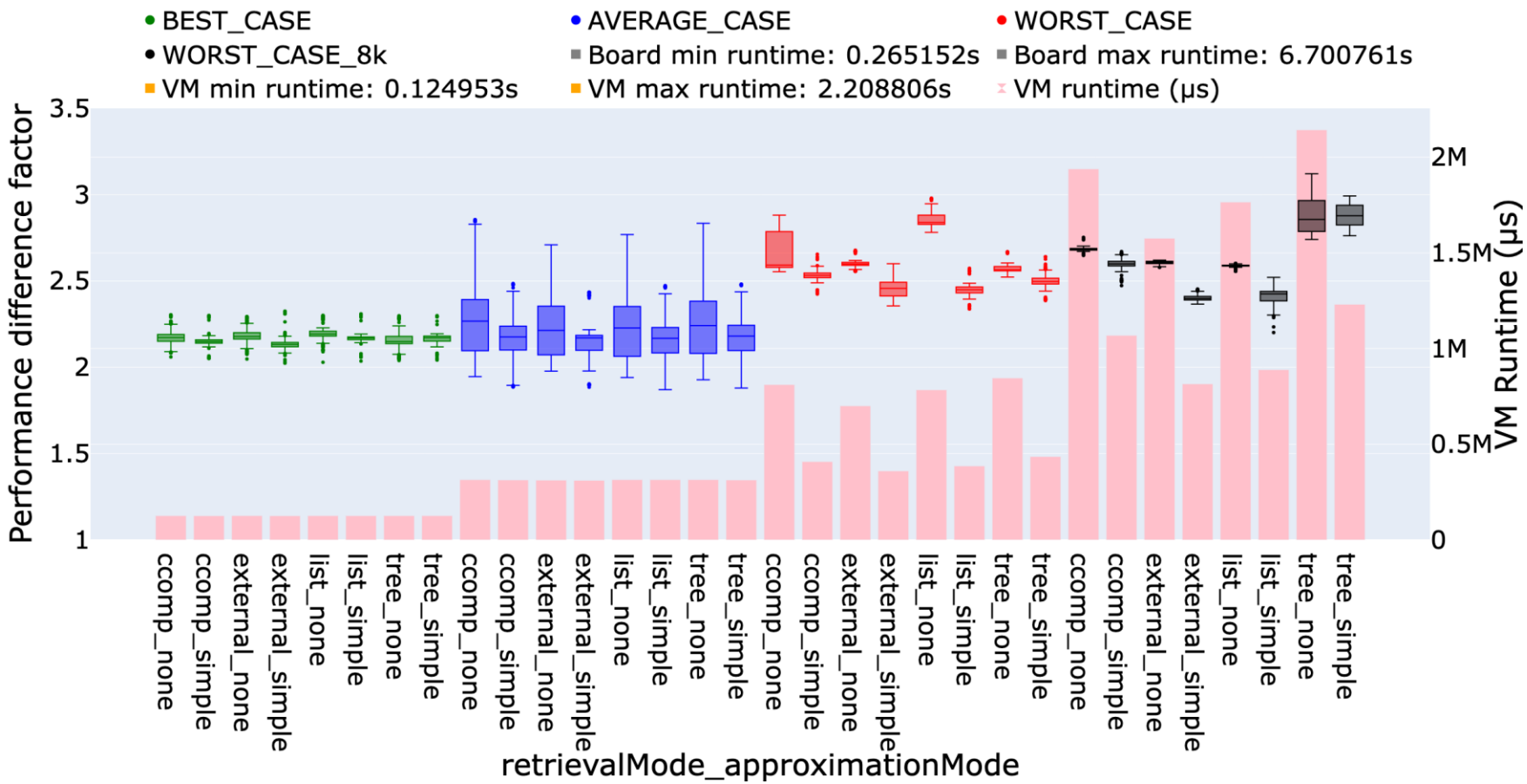
contour approximation method		
method	**CHAIN_APPROX_SIMPLE**	**CHAIN_APPROX_NONE**
performance impact	removes redundant points: less memory intensive	all the boundary points are stored: very memory intensive

contour retrieval mode				
mode	**EXTERNAL**	**LIST**	**CCOMP**	**TREE**
performance impact	least compute-/memory-intensive	no hierarchy: less memory-intensive	2-level hierarchy: slightly more memory-intensive	all hierarchy levels: most memory-/compute-intensive

Results Contour Detection

A single outlier run observed:

- Factor for 8k images lower than even the best case
- Not reproducible, even with the same binary or over 24h

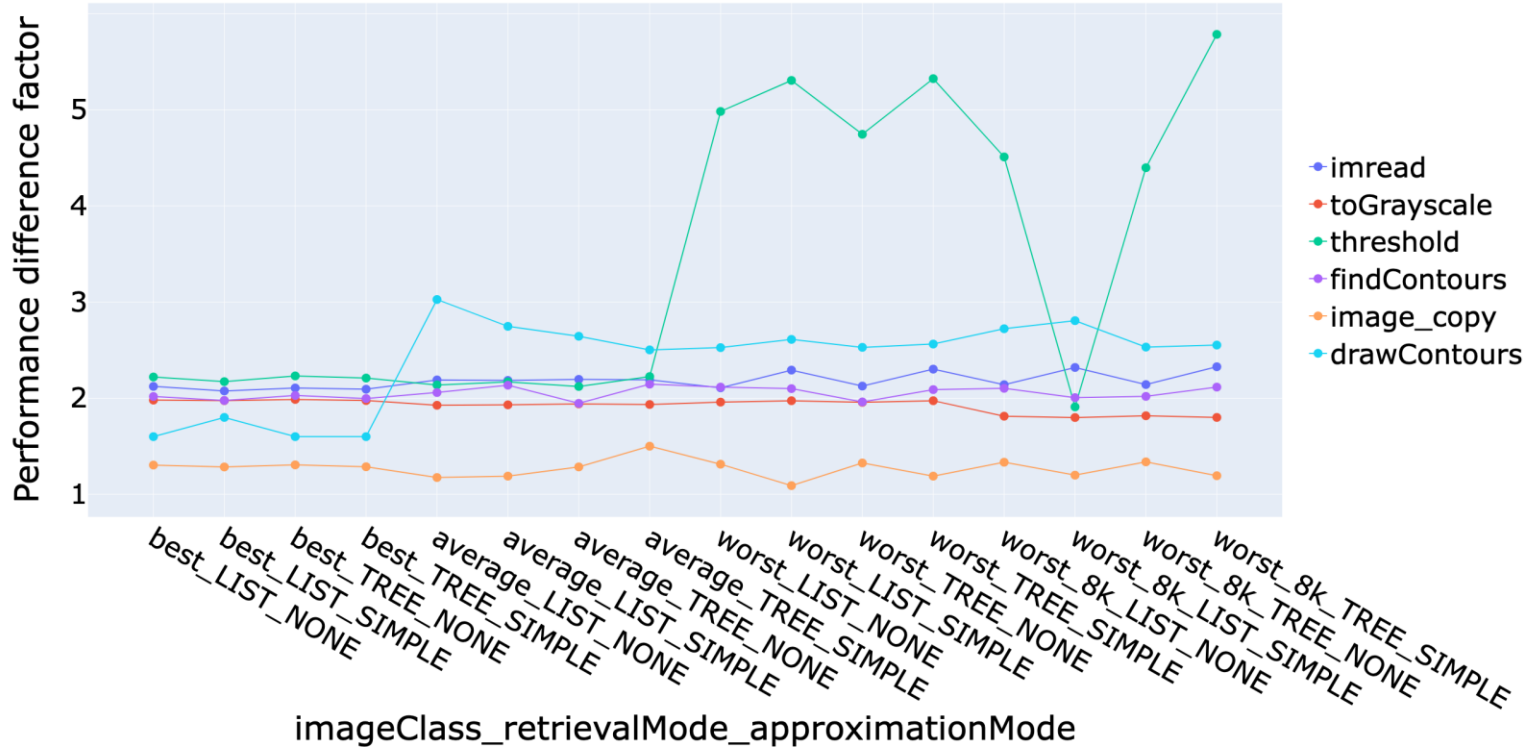


Contour Detection timing of separate sections

- **binary thresholding** section factor increase by **250%**
- **image read** function factor increase by **10%**

- **binary thresholding** section accounts for **2%** of total runtime
- **image read** function accounts for **30%** of total runtime

- OVERALL => 10% total increase in runtime
- Attributed to caching and memory model differences (esp. 32MB SLC)



TTTechAuto

05

Summary

Summary

- A* benchmark shows an **average** factor of **2.045**, with standard deviation of 0.15
- Contour Detection benchmark shows an **average** factor of **2.31**, with a standard deviation of 0.2
- CoreMark-PRO shows an **average** factor of **2.52**, with a standard deviation of 0.56
- Overall, values **between 1.8** and **3.6** were observed for the performance factor
- The factor is consistent for every specific input type, i.e., **no significant jitter** occurs due to, e.g., timing anomalies
- Difference between the avg. factors of the representative benchmarks is around **13%**

06

Open points & Outlook

Open points & Outlook

- Vector destruction has bad performance on VM -> it is possible that there are other edge cases/outliers with different behaviour
- More benchmarks can be added (floating-point, complex data structures)
- Introducing workload characterization via static and/or dynamic analysis
- different CPU:VM/Cloud combinations, esp. x86 vs ARM, other cloud providers
- Stability of the cloud performance, as motivated by the 8k images factor outlier


TTTechAuto
Thank you!