Leaps and bounds: Analysing WebAssembly's performance with a focus on bounds checking

Raven Szewczyk
University of Edinburgh
she/her

Kim Stonehouse
University of Edinburgh
she/her

Antonio Barbalace
University of Edinburgh
he/him

Tom Spink
University of St Andrews
he/him

2022-11-08

IISWC 2022

Background

WebAssembly

- Binary instruction format
- Stack-based virtual machine
- Originally designed for the Web
- Other uses: plugin sandboxing, Function-as-a-Service runtimes
- Interpreted, Just-in-Time compiled, or Ahead-of-Time compiled

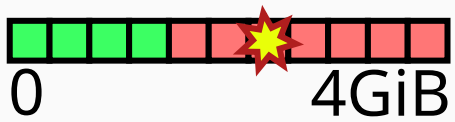
WEBASSEMBLY

Bounds Checking

WebAssembly linear memory: growable, contiguous, mutable array of bytes

Each load&store instruction is bounds checked for safety

- Pure software approach:
  ```
  if (address >= memory.size) crash;
  ```
- Virtual memory overallocation, *mprotect*, signal handlers
- UserfaultFD – user-mode page fault handling in Linux
- Intel MPX – deprecated
- Future: CHERI



Mprotect & UserfaultFD bounds checking methods

i32.load op1+const

0 < op1, const < 4GiB

mprotect

userfaultfd

Our contributions

Key Contributions

- Extensive comparison of Wasm performance on 3 ISAs
- Isolation of the impact of bounds checking
- Implementing alternative bounds checking modes into popular Wasm runtimes
- New reusable open-source benchmark harness
- Reproducing past findings on Wasm performance

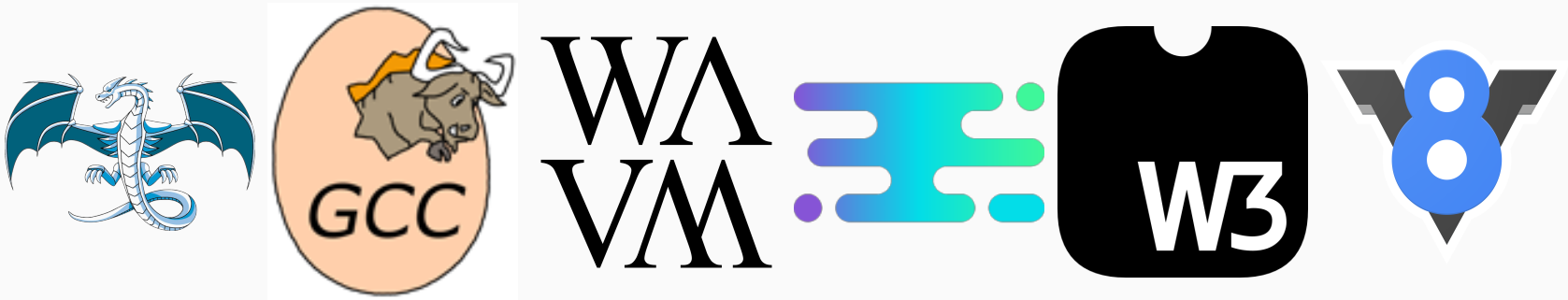
Key Results

- For Polybench/C kernels, the cost of each bounds checking method relative to no checks is roughly the same on x86-64, Armv8, and RISC-V
- Using *`mprotect()`* on Linux can cause poor multithreaded scaling, mitigated by our userfaultfd approach
- WebAssembly is fast enough for server applications with the right runtime – WAVM has 8-20% average overhead over native on x86-64.

Benchmark design

Tested runtimes

- Native Clang 13, Native GCC 11
- WAVM – LLVM-based AoT compiler
- Wasmtime – Cranelift-based AoT compiler
- Wasm3 – Threaded interpreter
- V8 Turbofan – JavaScript and Wasm JIT compiler, part of Chromium



Tested programs & bounds checking mechanisms

- Polybench/C MEDIUM – simple numerical kernels
- SPECcpu2017 Train-Rate – real-world applications[1]

- None – bounds checks are removed
- Clamp – `addr = min(addr, MEMORY_END)`
- Trap – `if (addr >= MEMORY_END) trap()`
- Mprotect
- Userfaultfd (UFFD)

[1]Subset of SPEC CPU 2017 Rate suite used: 505.mcf_r, 508.namd_r, 519.lbm_r, 525.x264_r, 531.deepsjeng_r, 544.nab_r, and 557.xz_r

Hardware

ISA	Processor	Memory	Threads	Platform
x86-64	Intel Xeon 6230R	768 GiB	16, HT off	Supermicro
Armv8	ThunderX2 CN9980	256 GiB	16, SMT off	Cavium
RV64GC	XuanTie C906	1 GiB	1	Nezha D1

- Ubuntu 22.04 LTS
- Linux 5.16, Armv8: 5.13
- Governor: `performance`
- `mitigations=off`

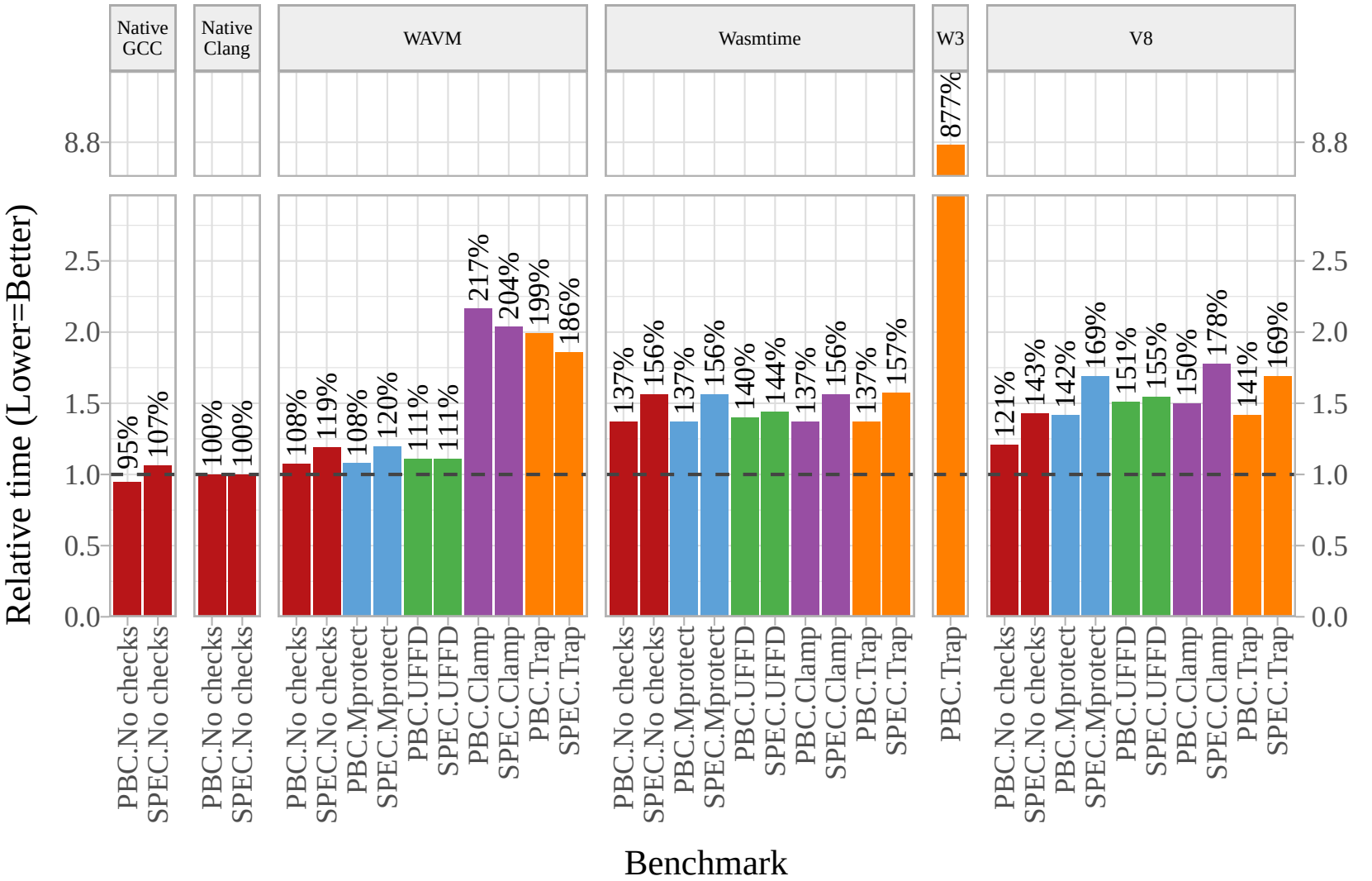
Benchmarking Harness

- ~2000 lines of C++ code
- Compiles, loads, executes Wasm modules
- Untimed warm-up and cool-down phases
- Pins threads to CPU cores
- Keeps track of CPU time, memory and perf counters[2]

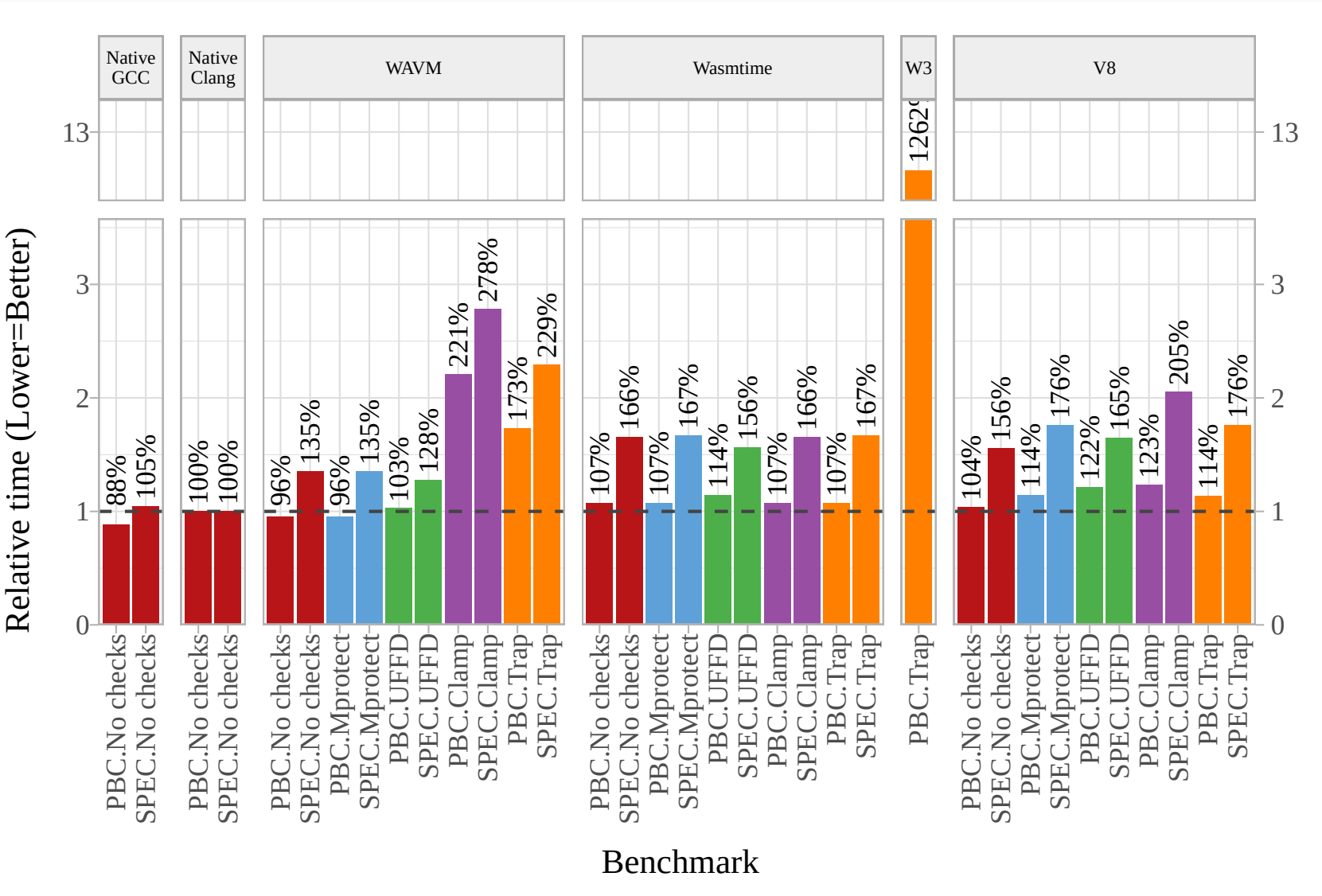
[2]Except RISC-V due to high overheads

Benchmark results

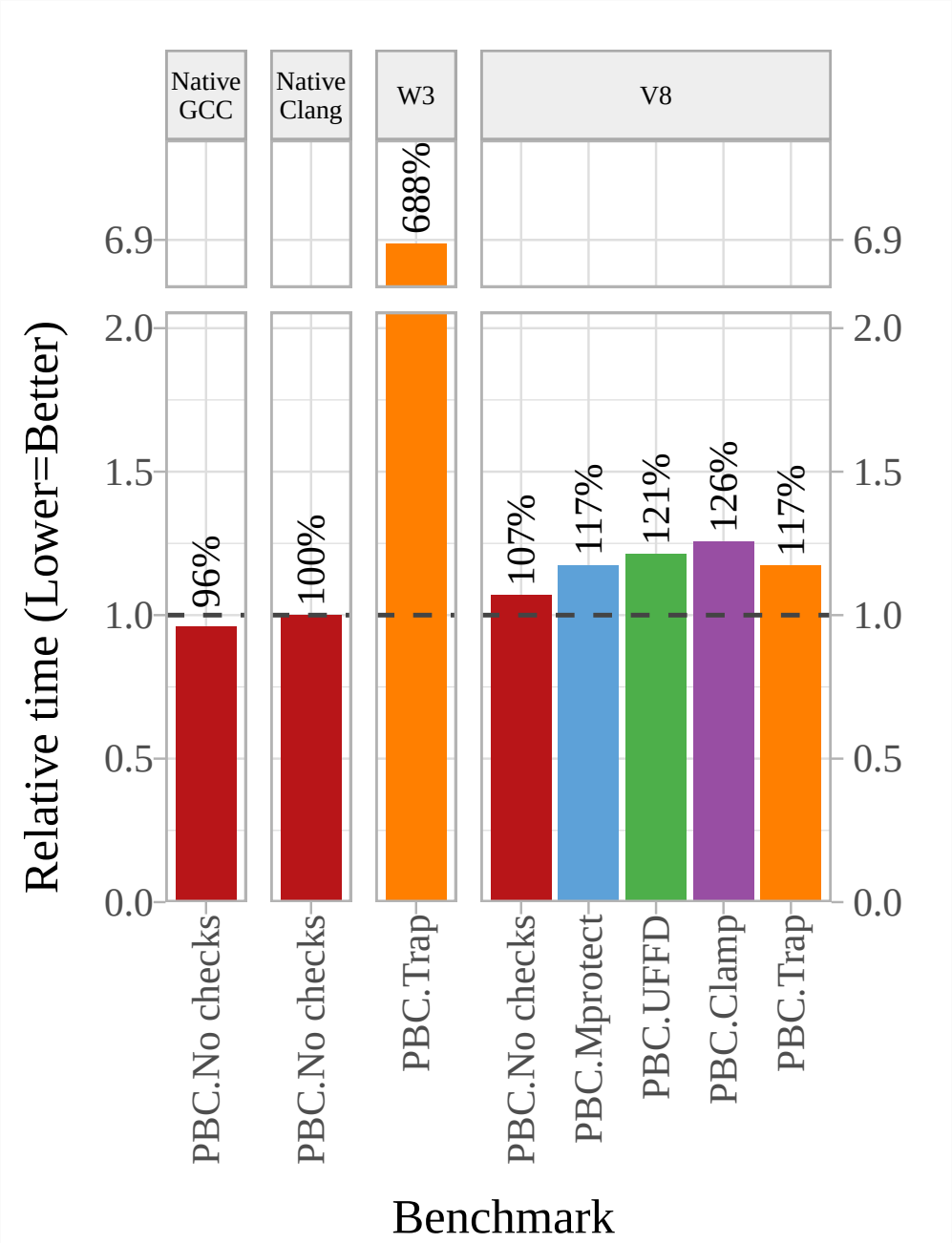
Execution time – 1 thread – x86-64



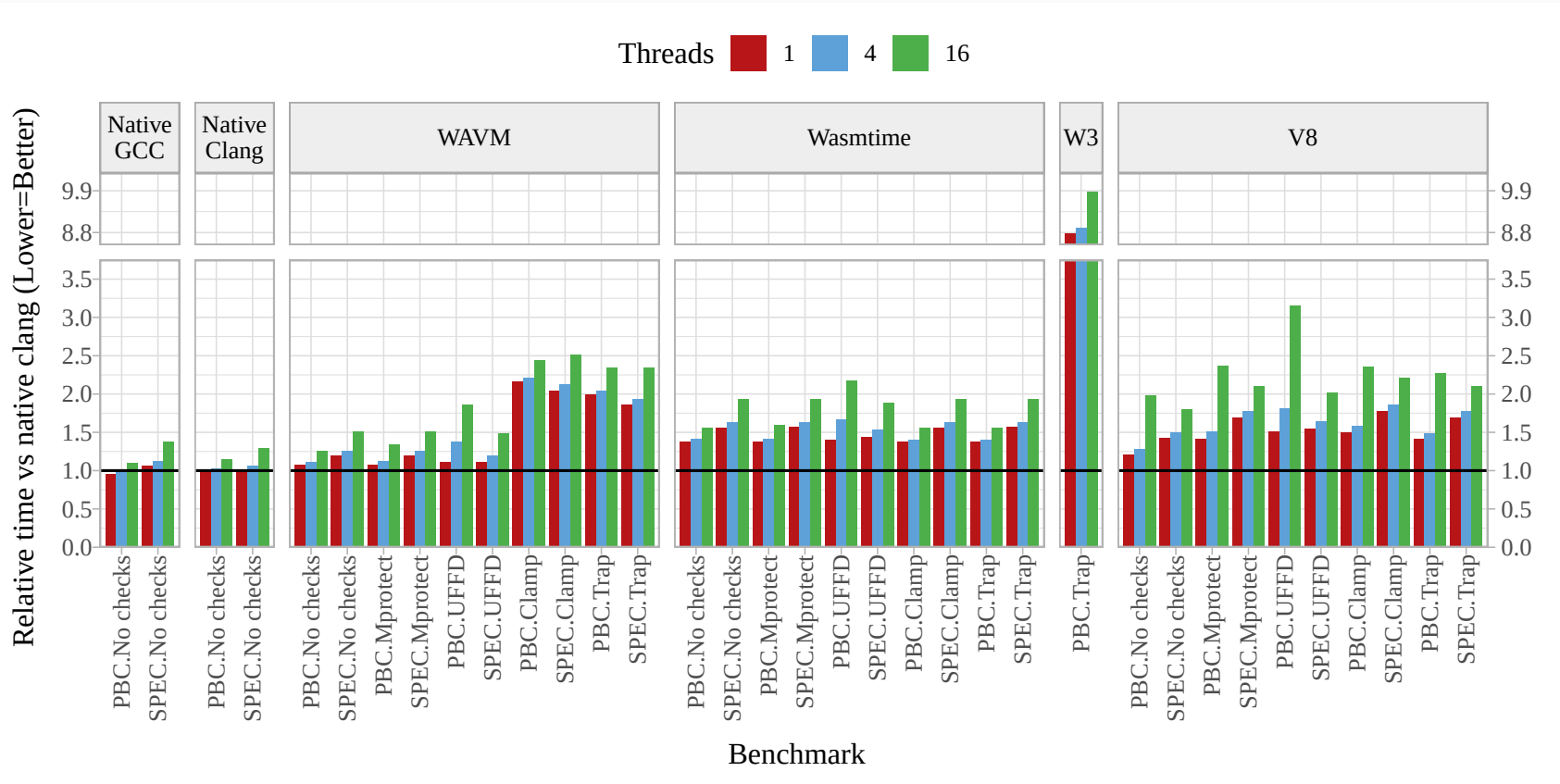
Execution time – 1 thread – Armv8



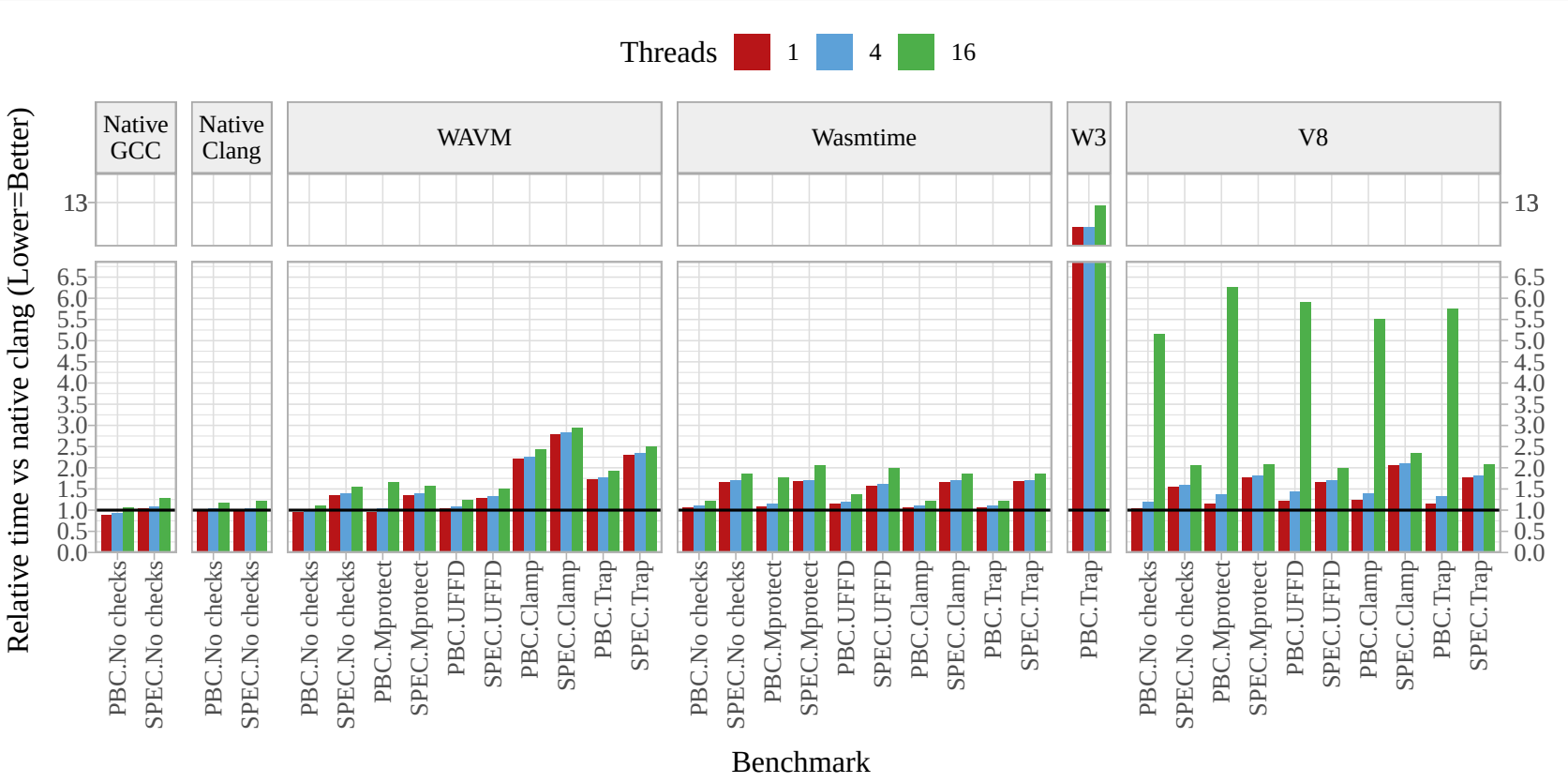
Execution time – 1 thread – RISC-V



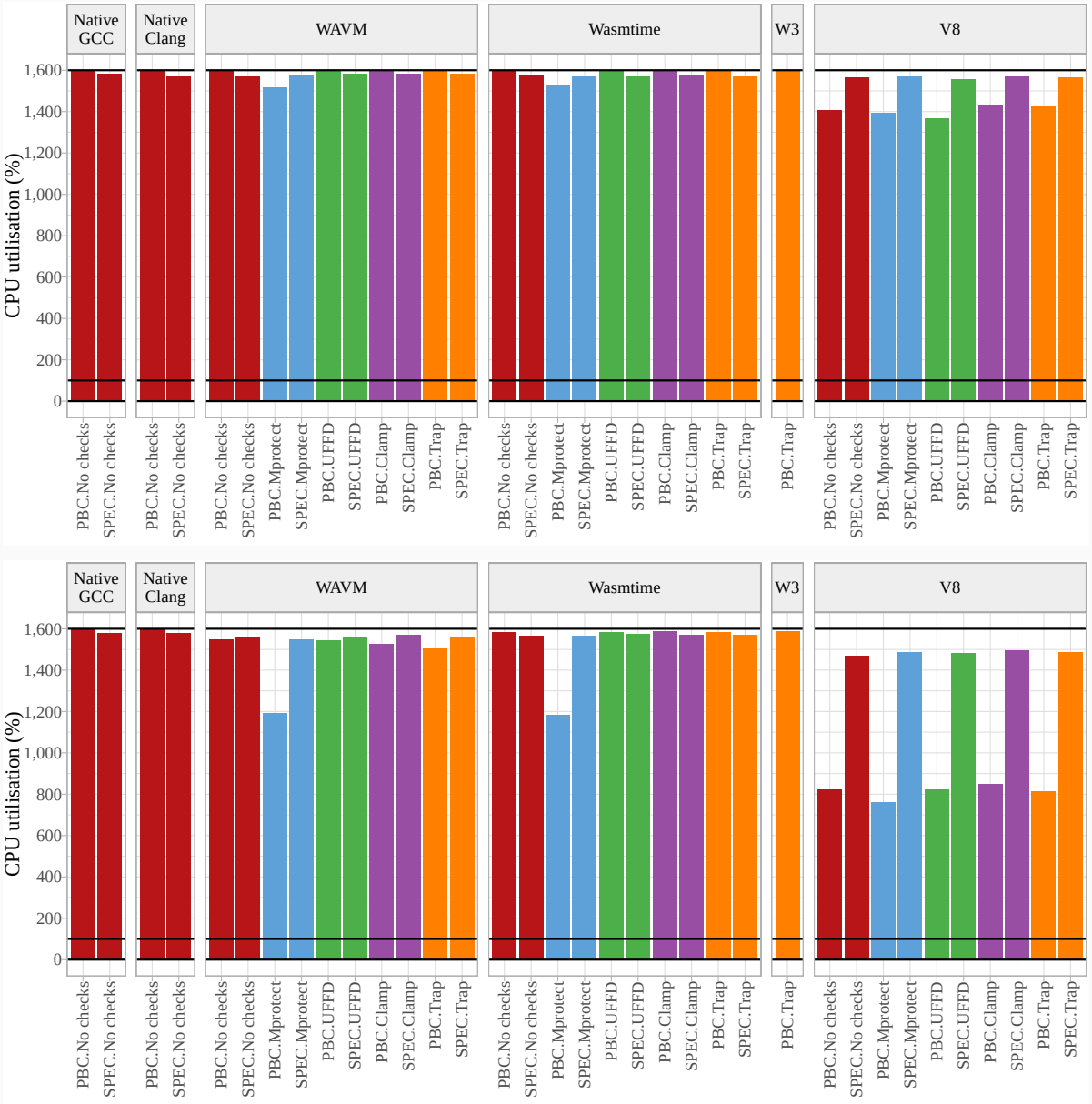
Execution time – thread scaling – x86-64



Execution time – thread scaling – Armv8



CPU load – 16 threads – x86-64 & Armv8



Summary

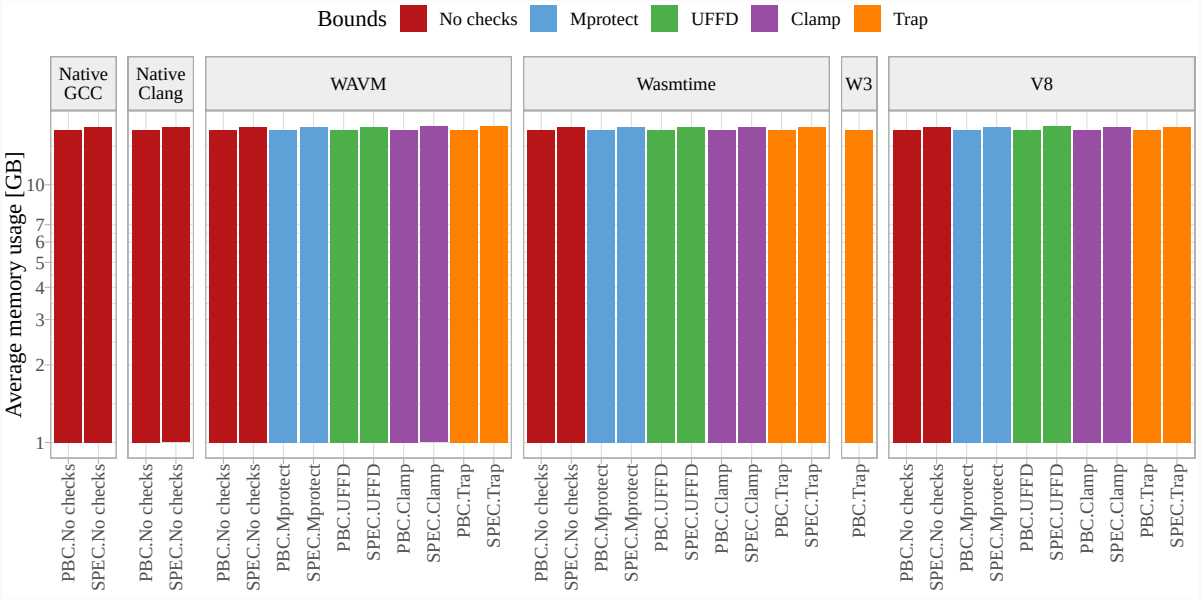
- x86-64: WAVM&Wasmtime within 20% of native performance
- Armv8: Within 35% of native
- RISC-V: V8 within 17% for Polybench/C
- *`mprotect`* can cause thread scaling issues due to kernel locking – UserfaultFD faster on Arm
- Raw benchmark data, graphs, source code available at *`https://github.com/wasmbounds/wasmbounds`*[3]

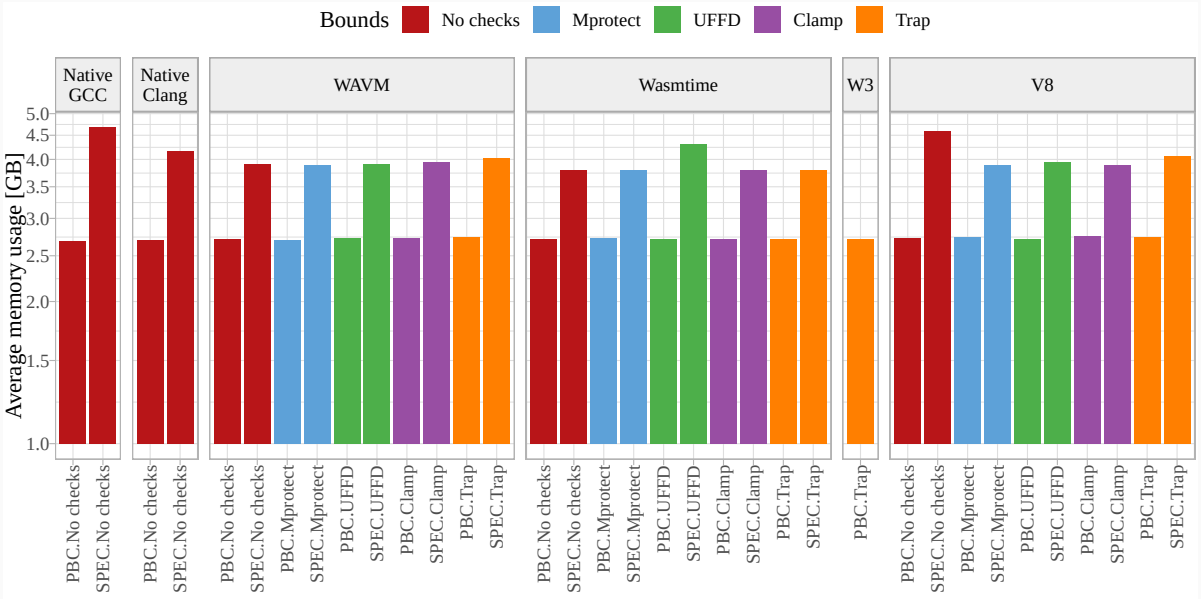
Contact me at: Raven.Szewczyk@ed.ac.uk

[3]Excluding SPECcpu2017, Archived at Zenodo: 10.5281/zenodo.7068161

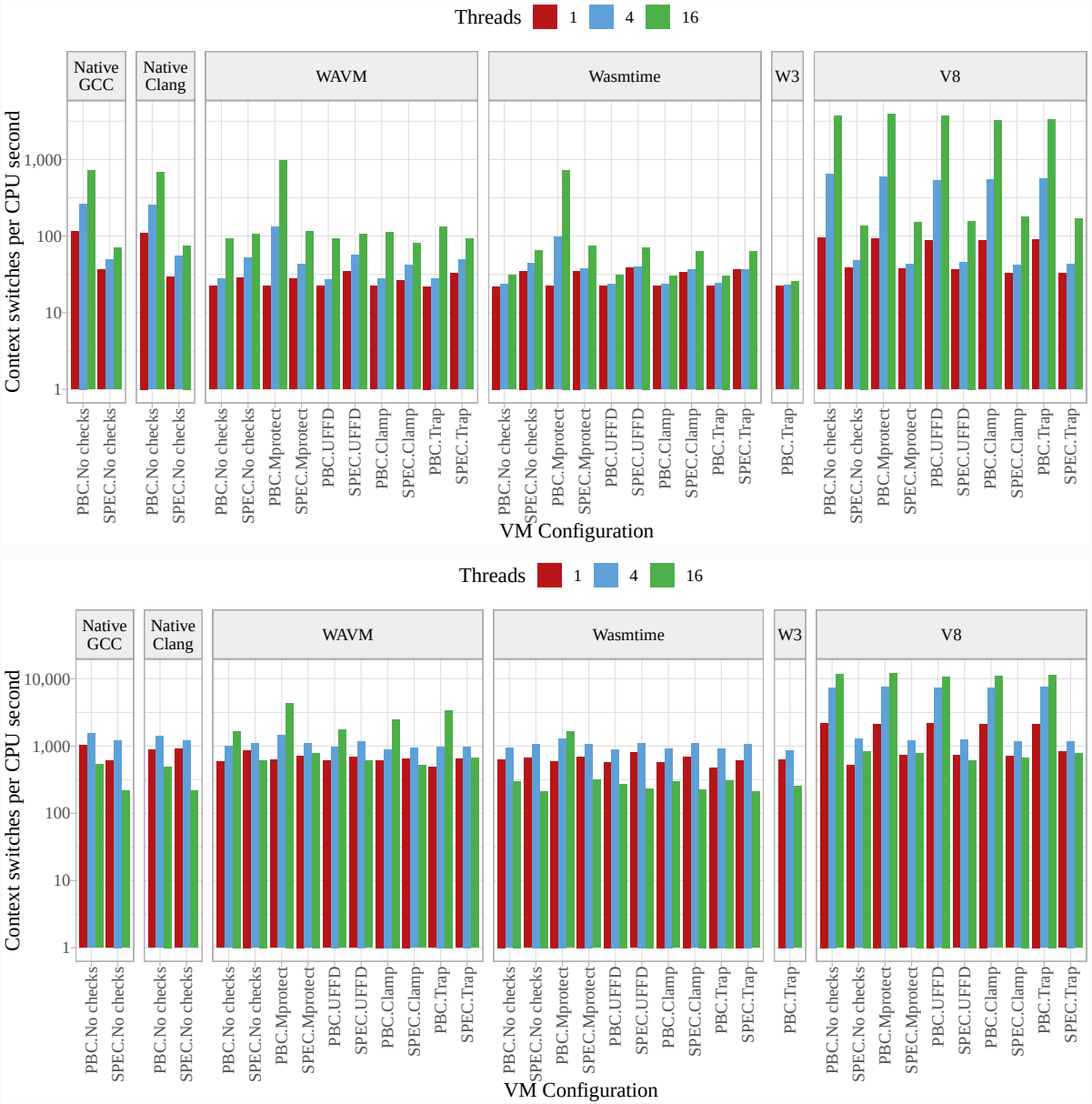
Backup slides

Average memory usage – x86-64 & Armv8

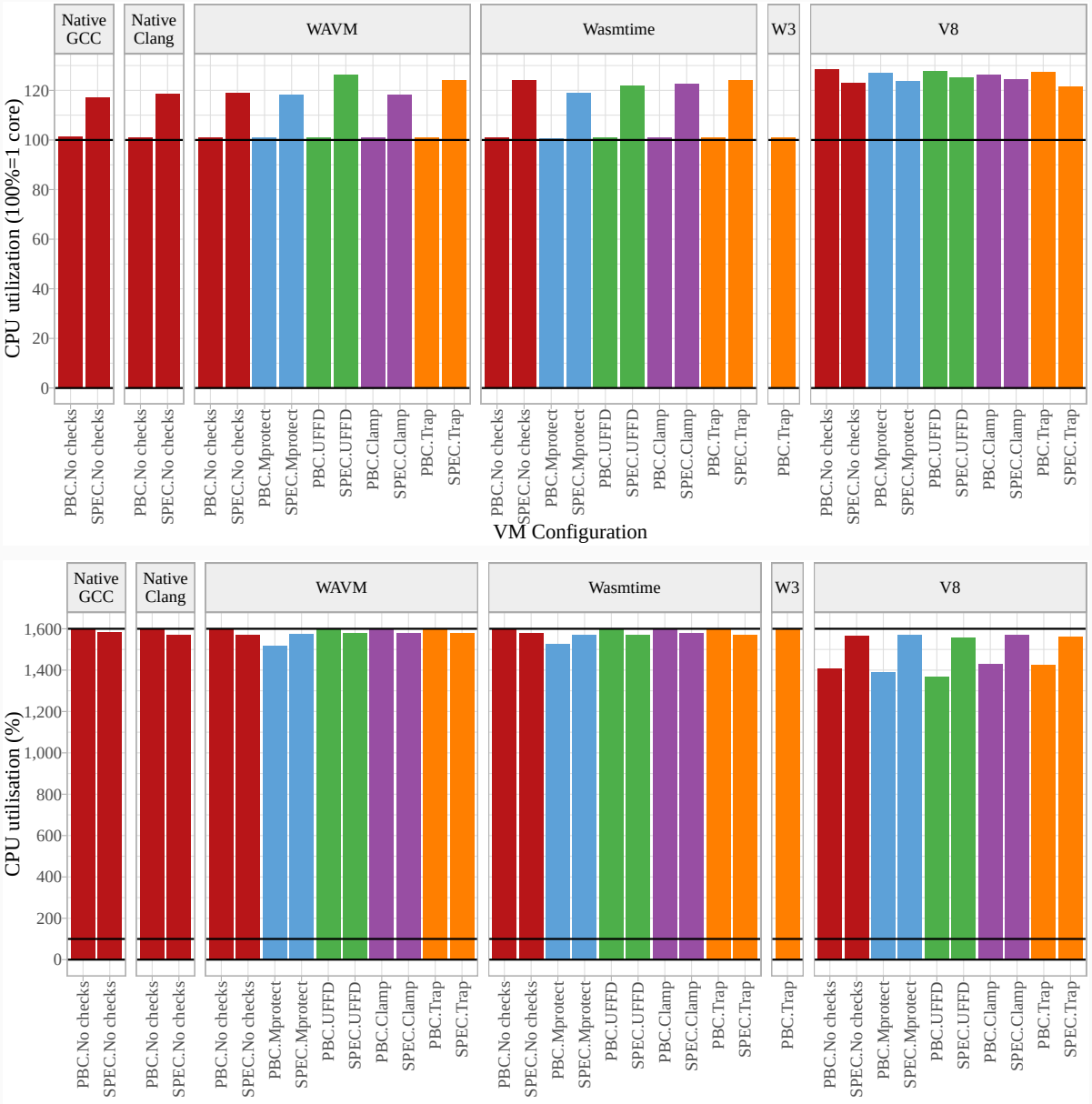




Context switches – x86-64 & Armv8



CPU load – 1 & 16 threads – x86-64



CPU load – 1 & 16 threads – Armv8