Analysis and Characterization of Performance Variability for OpenMP Runtime

Minyu Cui, Nikela Papadopoulou, Miquel Pericàs

Chalmers University of Technology

International Workshop on Runtime and Operating Systems for Supercomputers

Presentation Outline

- Introduction and motivation
- Methodology and Experimental Setup
- Experimental Results
- Conclusion & future work

Presentation Outline

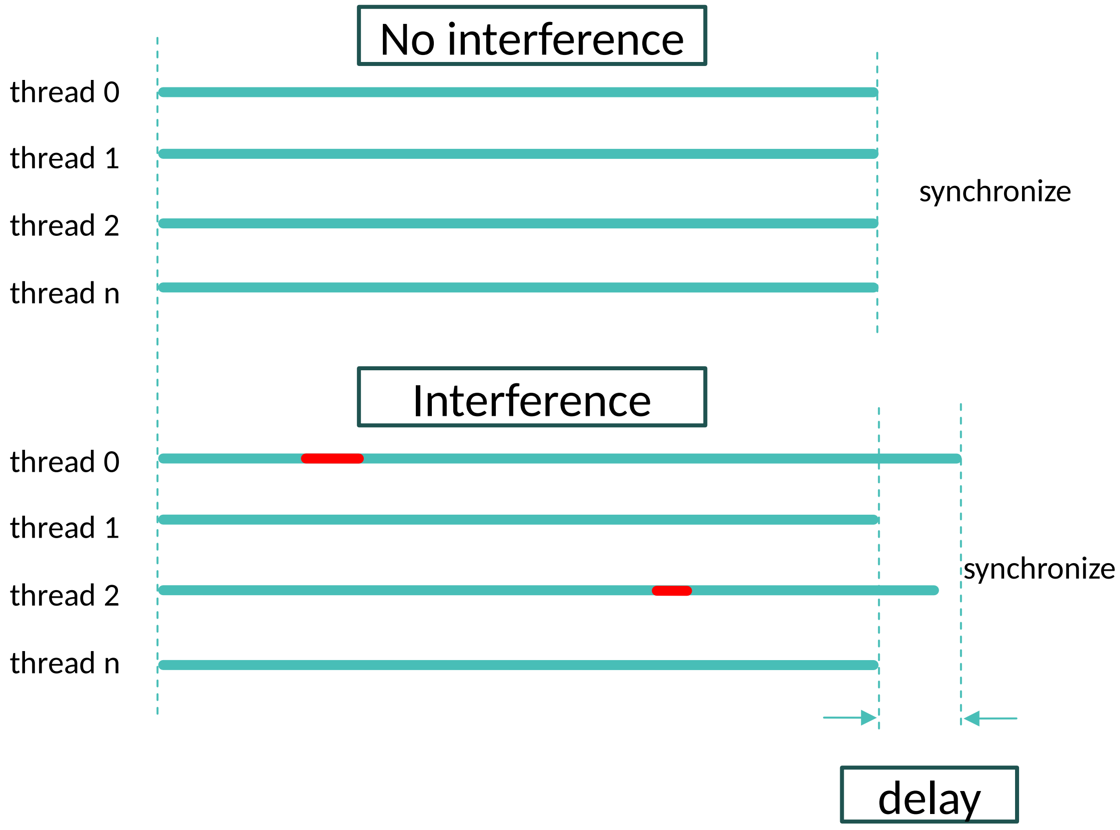
- Introduction and motivation
- Methodology and Experimental Setup
- Experimental Results
- Conclusion & future work

Introduction

- European Processor Initiative (EPI) project is developing several multicores based on ARM-SVE and RISC-VV[1]
- OpenMP is the main intra-node programming model
  - ✓ *#pragma omp parallel*
- Performance stability is a major concern
  - ✓ operating system (OS) activities
  - ✓ contention and interference on shared resources
  - ✓ random thread delays
  - ✓ more cores may lead to higher variability

✓ Our long term goal: tune Linux+OpenMP to reduce the impact of variability in upcoming EPI multicores

  - ✓ 1st step is to perform a characterization study

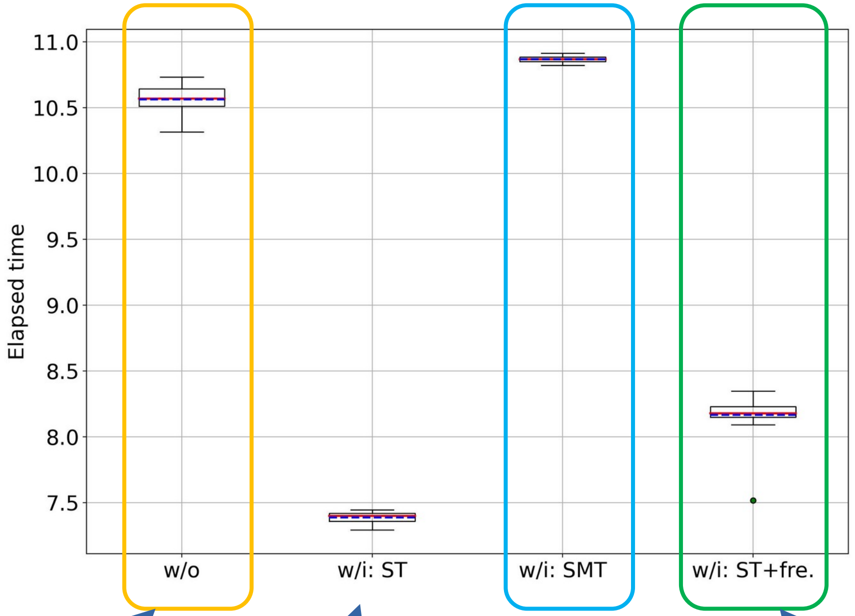


[1] *Please visit the EPI-EUPilot-EUPEX booth #213 for more information :-)*

Impact of variability on large multicores

- Example: LULESH mini-app case study
- Dual socket AMD Zen2 with 128 cores total (256 HW threads)
- What causes increased execution time and/or variability?

128 threads



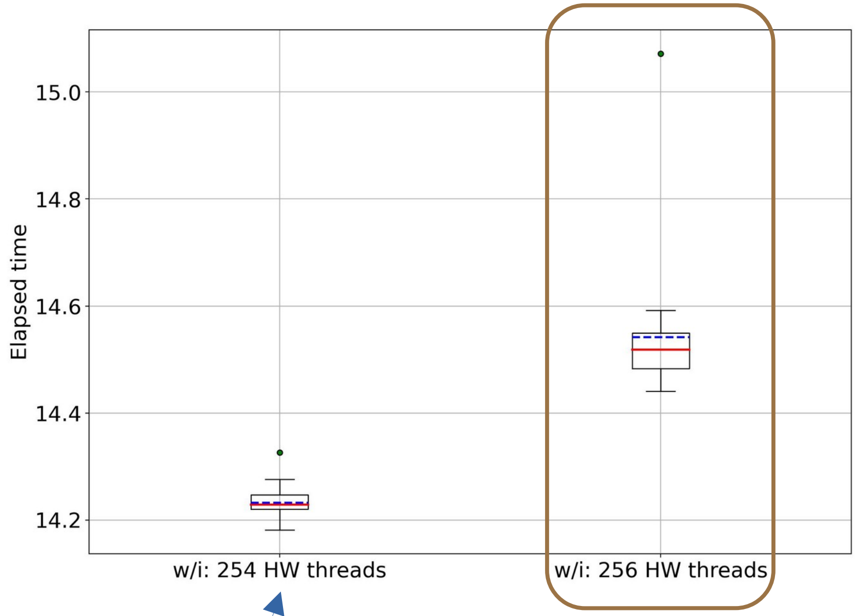
No thread pinning

with thread pinning

SMT on 64 cores (only 1 socket)

With pinning and background “logging” thread

~256 threads



SMT with 254 threads (one idle core)

SMT with 256 threads (no idle cores)

Related work and Goals of our work

- Methodologies to reduce performance variability
  - ✓ Applying thread-pinning
  - ✓ Reserving threads for system activities (e.g., daemons, interrupts, …)
  - ✓ Disable dynamic frequency management
- Kernel-level tools to measure and analyze OS noise
  - Lo2s, LTTng-Noise, osnoise

https://hpc-wiki.info/hpc/Lo2s

https://lttng.org/

https://docs.kernel.org/trace/osnoise-tracer.html

Presentation Outline

- Introduction and motivation
- Methodology and Experimental Setup
- Experimental Results
- Conclusion & future work

Methodology

- Run on production environments, using x86 system as proxies
  - no OS-level analysis tools, instead we rely on statistical analysis
  - Based on microbenchmarks to evaluate OpenMP constructs
  - Evaluate impact of thread pinning, thread mapping (eg SMT), background processes and DVFS
- Thread-pinning
  - ✓ OMP_NUM_THREADS
  - ✓ OMP_PLACES
  - ✓ OMP_PROC_BIND (*close*)
- Simultaneous multithreading (SMT) + Idle cores
  - ✓ Leave the extra hardware thread per physical core idle
  - ✓ Or: leave one core idle for system activities
- Frequency logging on a separate core
  - ✓ a background Python script to record the frequencies of all cores

Experimental setup

- Two hardware platforms

Platform/Attribute	Dardel	Vera
CPU model	AMD EPYC™ 7742 (Zen2)	Intel Xeon Gold 6130 (Skylake)
Number of CPU cores	64, 2-way SMT	16, no SMT
Number of NUMA nodes	8	2
Number of processors (sockets)	2	2
Linux distribution	SUSE Linux Enterprise Server 15 SP3 OS	Rocky Linux release 8.7
Linux kernel version	5.3.18-150300.59.76_11.0.53-cray_shasta_c	4.18.0

256 logical cores

32 logical cores

gcc v7.5.0

gcc v8.5.0

- OpenMP benchmarks - 10 runs for each benchmark
  - ✓ EPCC OpenMP microbenchmarks v3.0 (*schedbench*, *syncbench*) – each with 100 internal repetition
  - ✓ BabelStream v4.0 (OpenMP)
- Experiment configurations
  - ✓ **ST** - at most one hardware thread per physical core is used to run the benchmarks
  - ✓ **MT** (on Dardel) - both two hardware threads of the core are used to run the benchmarks

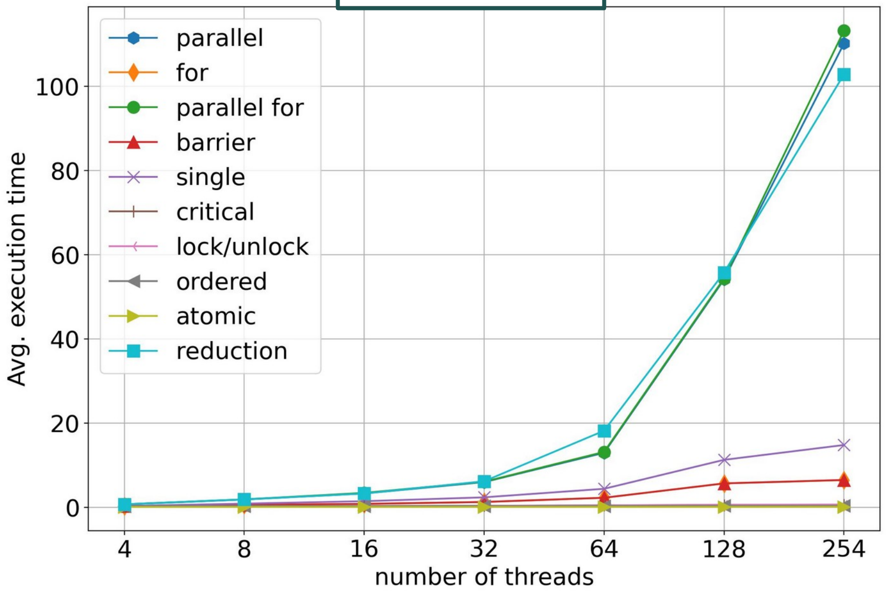
Presentation Outline

- Introduction and motivation
- Methodology and Experimental Setup
- Experimental Results
- Conclusion & future work

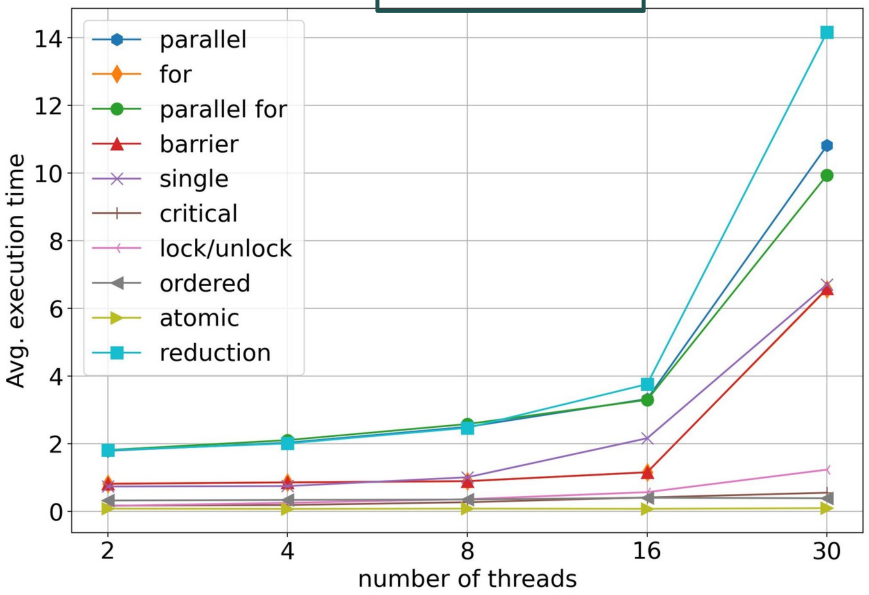
OpenMP scalability

- Scalability of average execution time (usec) after thread-pinning (plots: syncbench)
  - ✓ More HW threads -> higher average execution time for *schedbench* and *syncbench*
  - ✓ More HW threads -> lower average execution time for *BabelStream (shown in paper)*

Dardel



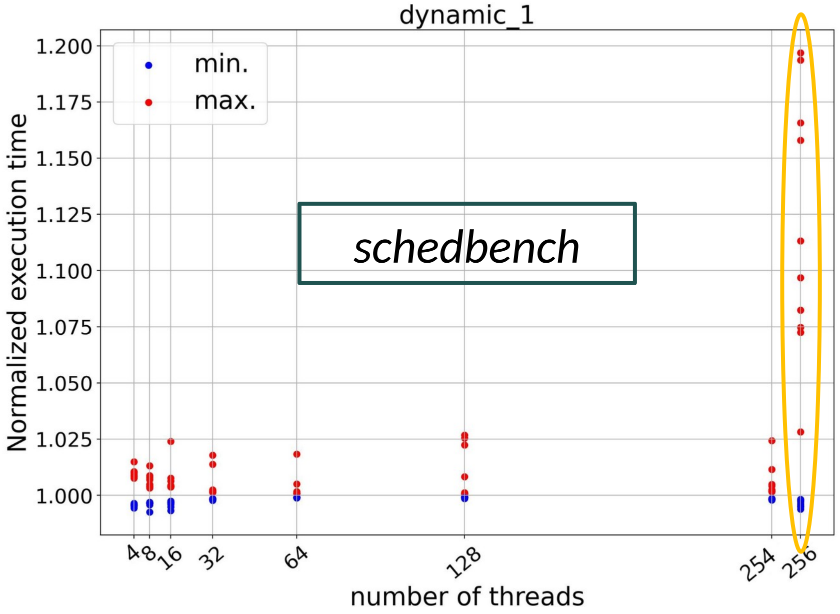
Vera

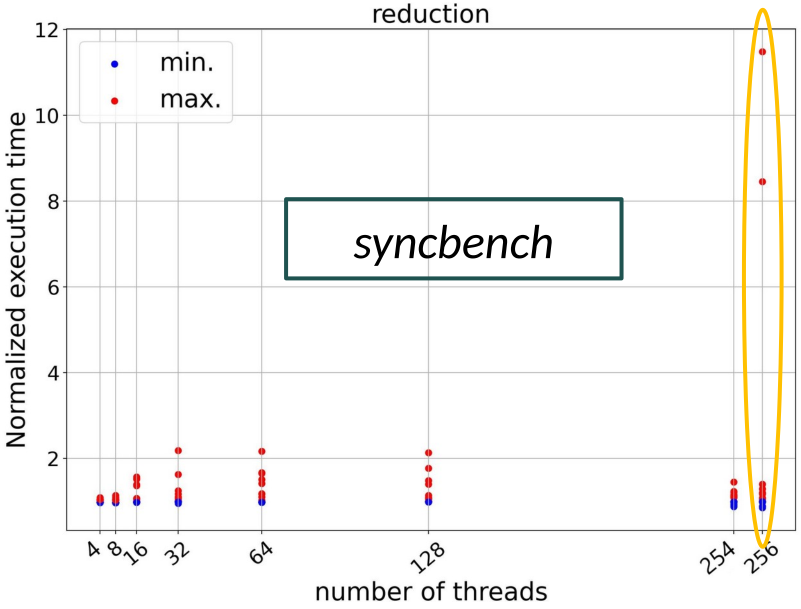


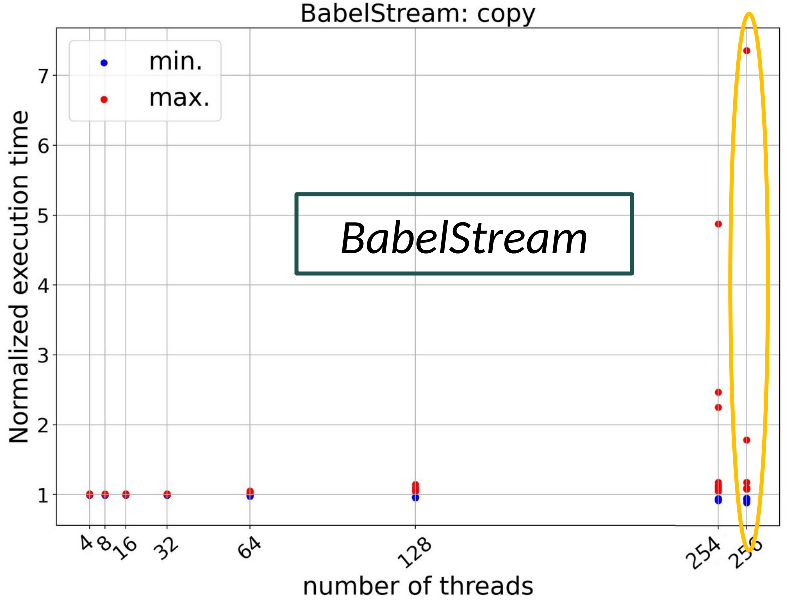
Higher HW thread count -> higher synchronization overheads
Particularly bad across sockets

OpenMP variability

- Scalability of performance (execution time) variability after thread-pinning on Dardel



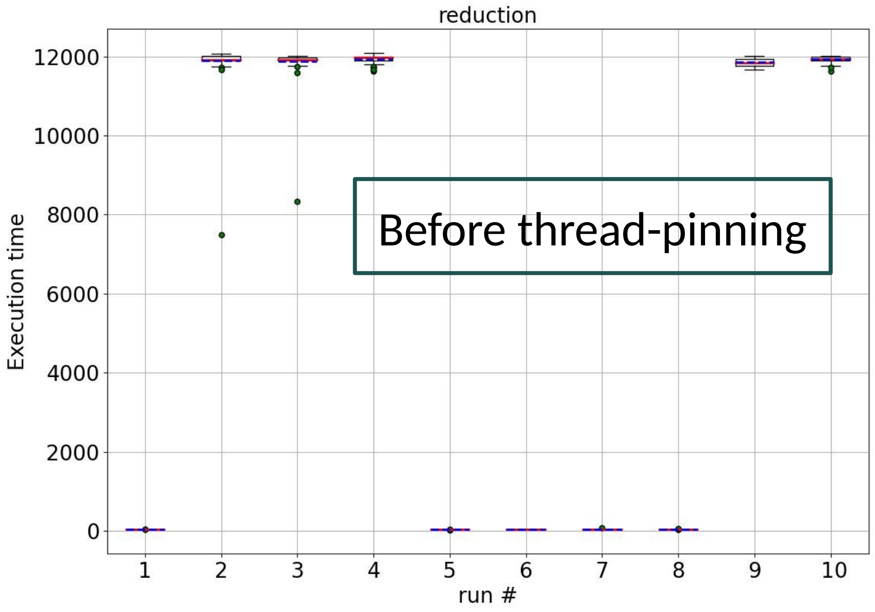


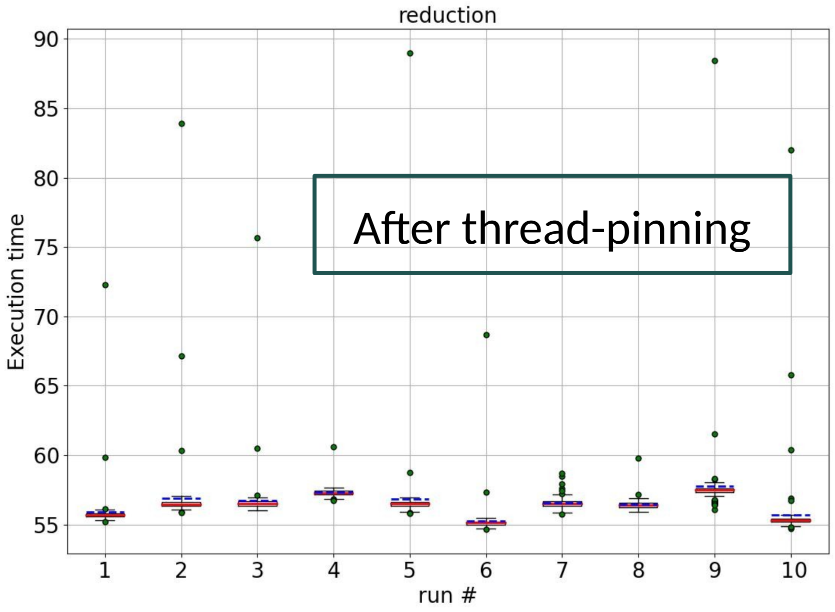


Higher HW thread count -> higher performance variability (syncbench + babelstream)
All HW threads -> significantly worse performance stability

Thread Pinning

- *Syncbench* after thread-pinning on Dardel (case study: reduction)
  - ✓ Lower run-to-run variability
  - ✓ Lower variations between the 100 internal repetitions

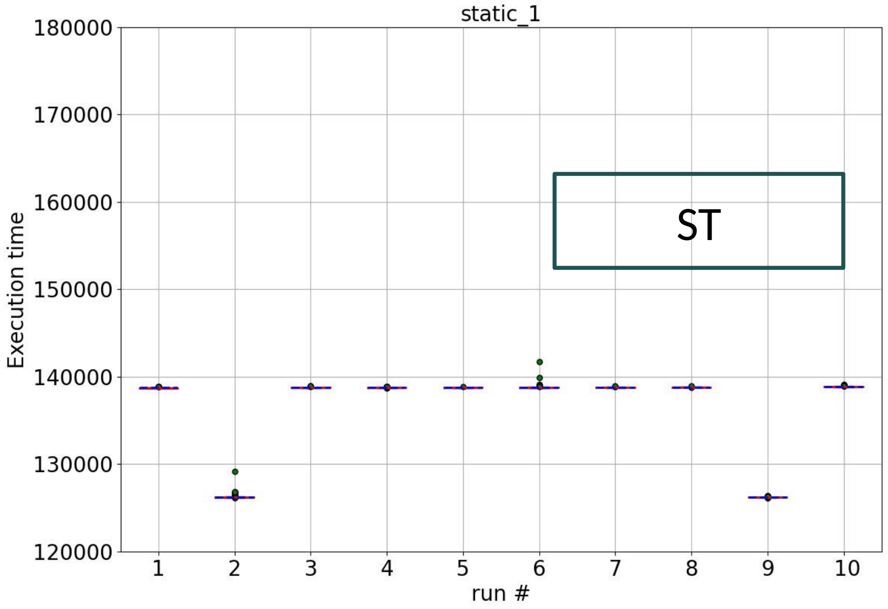




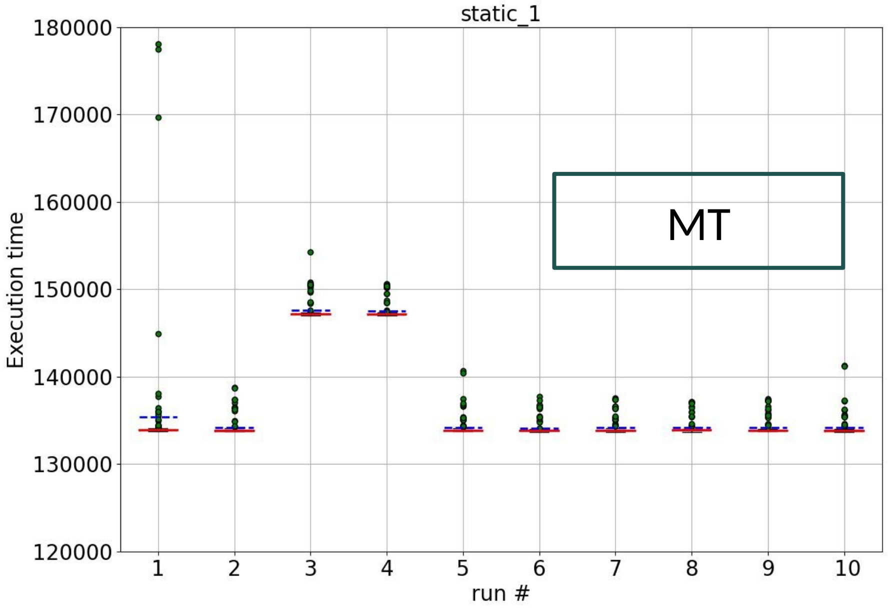
Lower performance variability after thread-pinning

Thread mapping and SMT

- MT (on Dardel) - both hardware threads of the core are used to run the benchmarks
- Plots show schedbench with 128 threads, using **both** sockets in both cases (ST and MT)
- Higher performance variability when using additional threads implemented by SMT to run benchmarks



64+64 cores w/ single threading

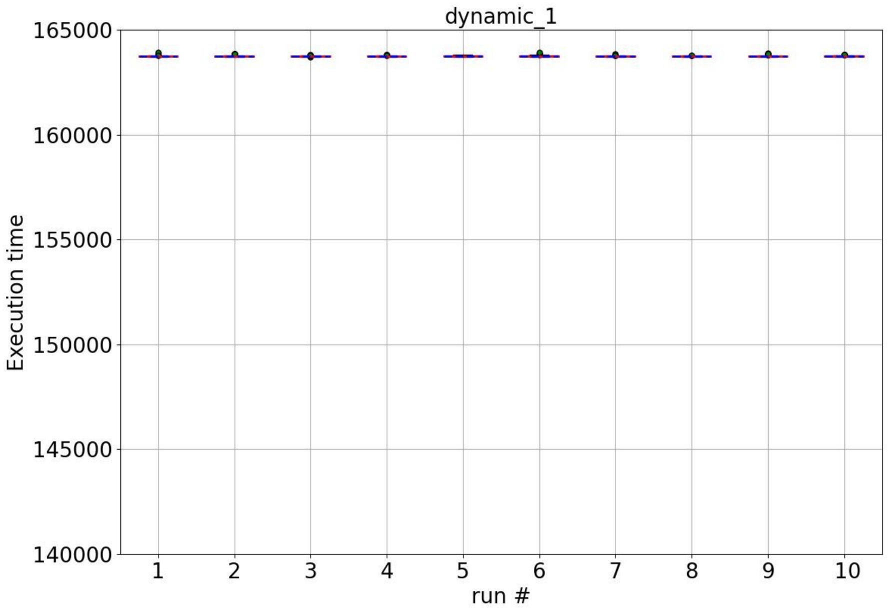


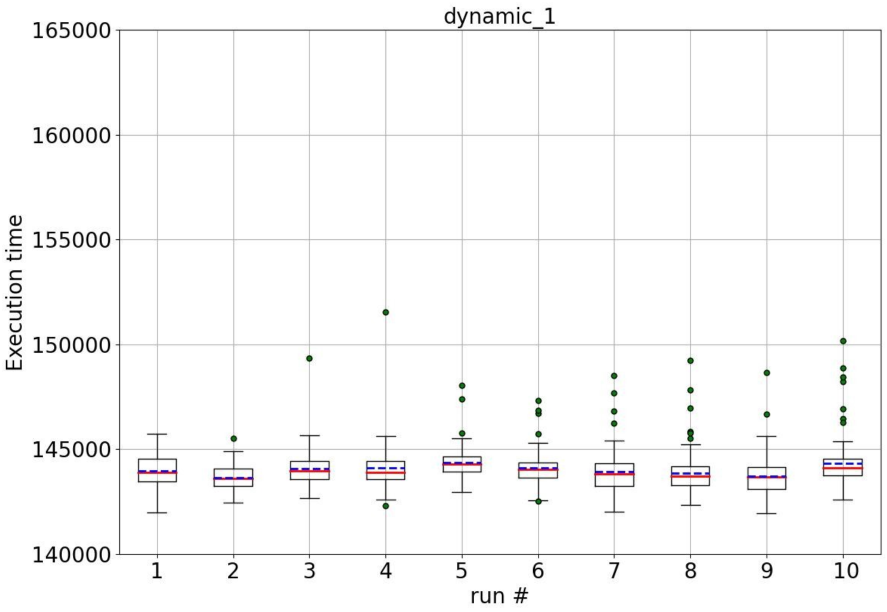
32+32 cores w/dual threading

Leaving the second thread in SMT for system activities results in better performance stability

Effects of Frequency Variation

- Frequency logging on a separate core on Vera
- Schedbench, 16 threads in one socket (left), 16 threads across two sockets (right)

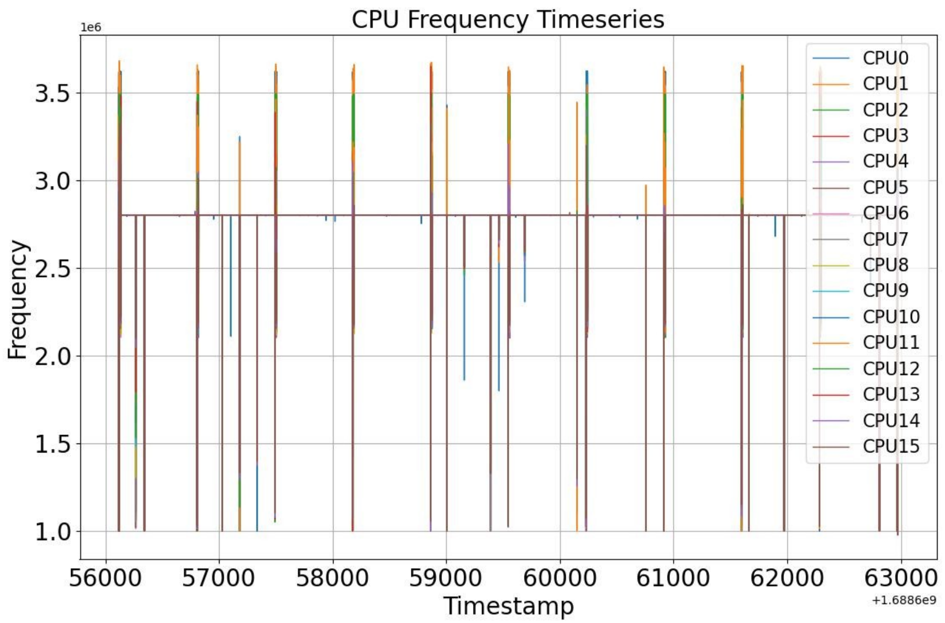


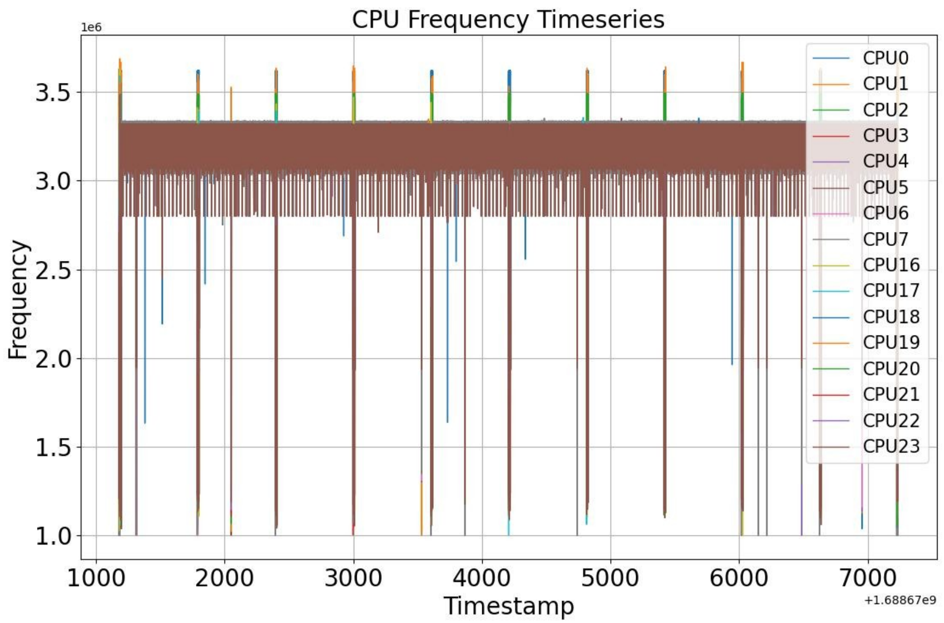


Higher performance variability due to frequency variation on Vera

Effects of Frequency Variation

- Frequency logging on a separate core on Vera
- Schedbench, 16 threads in one socket (left), 16 threads across two sockets (right)





Higher performance variability due to frequency variation on Vera

Presentation Outline

- Introduction and motivation
- Methodology and Experimental Setup
- Experimental Results
- Conclusion & future work

Conclusion

- Experimental study of performance stability for OpenMP runtime on large multicores
- High variability when all resources (all SMT threads are used)
  - Leaving one core or the second SMT thread idle considerably reduces variability
  - For example, avoid interference from the frequency logger and benchmark running on the same core
- In general, MT showed higher variability compared to ST with same number of threads.
- Frequency scaling drivers seem to add more variability when system is not close to power cap
  - ✓ Needs to be studied further
- Thread-pinning seems to be in general a good idea

Future work

- Correlate larger OpenMP applications with the results of this study
- Other compiler, e.g. LLVM runtime, and different runtime parameters e.g. OMP_WAIT_POLICY or different reduction algorithms
- Extend analysis to ARM-SVE platforms, in particular A64FX and Graviton3.
- Profiling of OS noise with kernel level tools
- Develop joint Linux+OpenMP methodology to mitigate performance variability in EPI cores

Thank you

Any questions?