TECHNISCHE UNIVERSITÄT DRESDEN

Center for Information Services and High Performance Computing

Detection and Visualization of Performance Variations to Guide Identification of Application Bottlenecks

Matthias Weber et al.
Presenter: Ronny Brendel
PSTI Workshop, Philadelphia, 16th August, 2016

TECHNISCHE UNIVERSITÄT DRESDEN

Contents

TECHNISCHE UNIVERSITÄT DRESDEN

Introduction

- Complexity of HPC systems is ever-increasing
- This creates challenges performance analysis

- Analysis techniques with different granularities and goals exist
  - Detailed execution recordings are well-suited for detecting performance variation across processes and/or time

- Automatic problem search ↔ visualization-based analysis

- We provide a new visualization-based approach for detecting performance problems

Introduction

- Assumptions:
  - Processes exhibit similar runtime behavior – SPMD
  - Processes execute the same code repeatedly – iterations

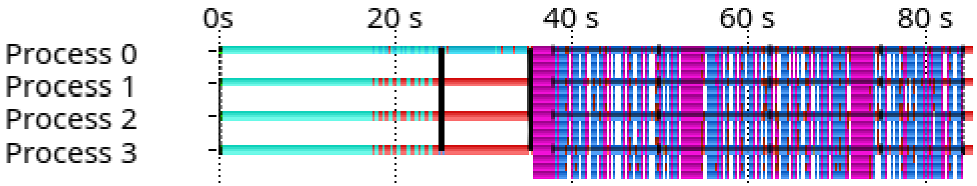
  - The duration of iterations should be similar between processes as well as between iterations on the same process

- If iterations vary in duration, this might indicate a performance problem (runtime imbalance / performance variation)

- Our approach detects such imbalances and highlights iterations with notably higher duration

Introduction

- We use execution traces [1,2] as the basis of analysis
  - Time-stamped events, in particular function enter & exit
- Timeline-based visualizations [3-5]
- Post-mortem analysis



- Approach:
  1. Identify dominant functions
  2. Compare runtime of them across iterations and processes
  3. Visualize these differences

TECHNISCHE UNIVERSITÄT DRESDEN

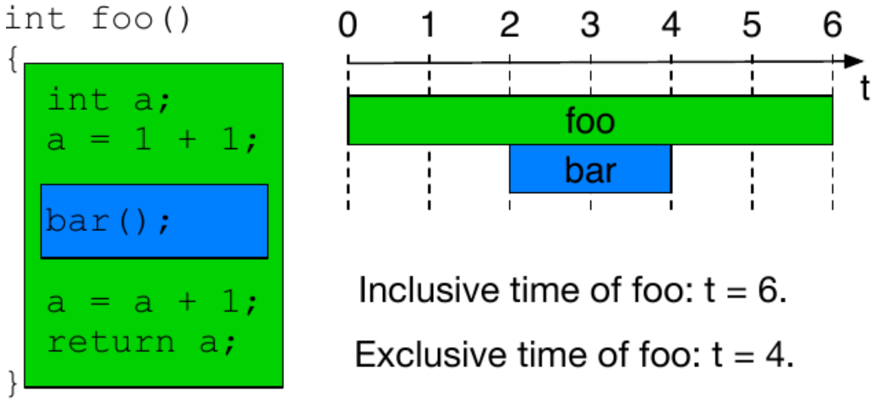
Contents

Identify Time-Dominant Functions

- Goal: Identify recurring parts of an application execution to then compare the runtime of these *segments*

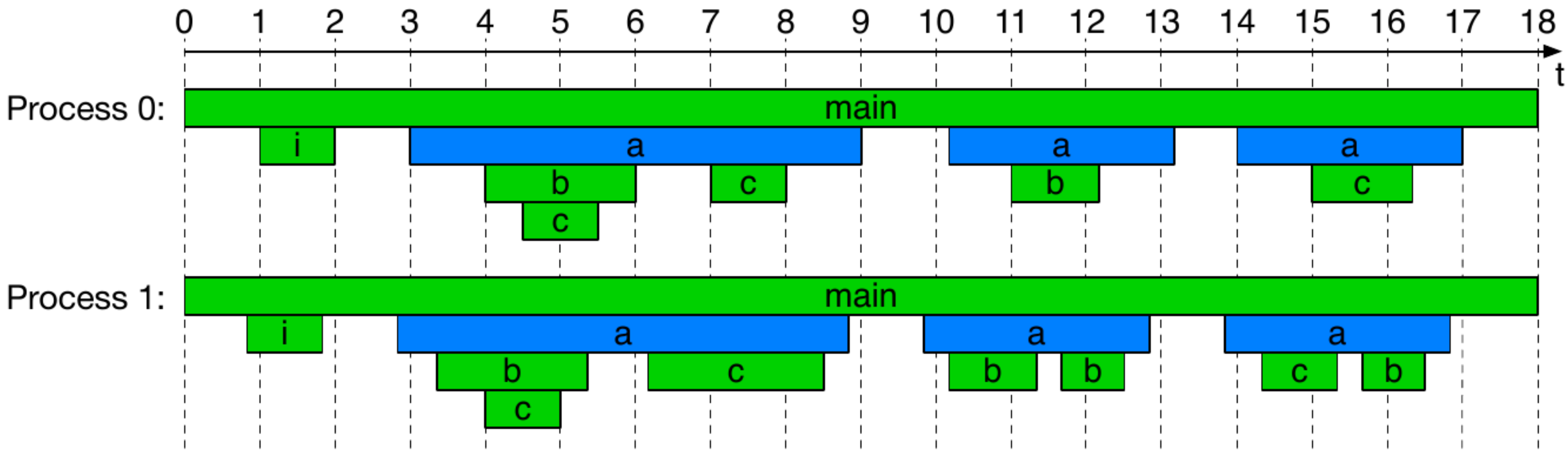
- What are suitable segments?

- Functions with a large inclusive time
  - Inclusive time is the time spent in a function including time spent in subfunctions



Identify Time-Dominant Functions

- Taking the function with just the largest inclusive time doesn‘t work, for example:



- Time-dominant function:= Function with the highest aggregated inclusive time which is called at least 2*p* times, where *p* is the number of processes

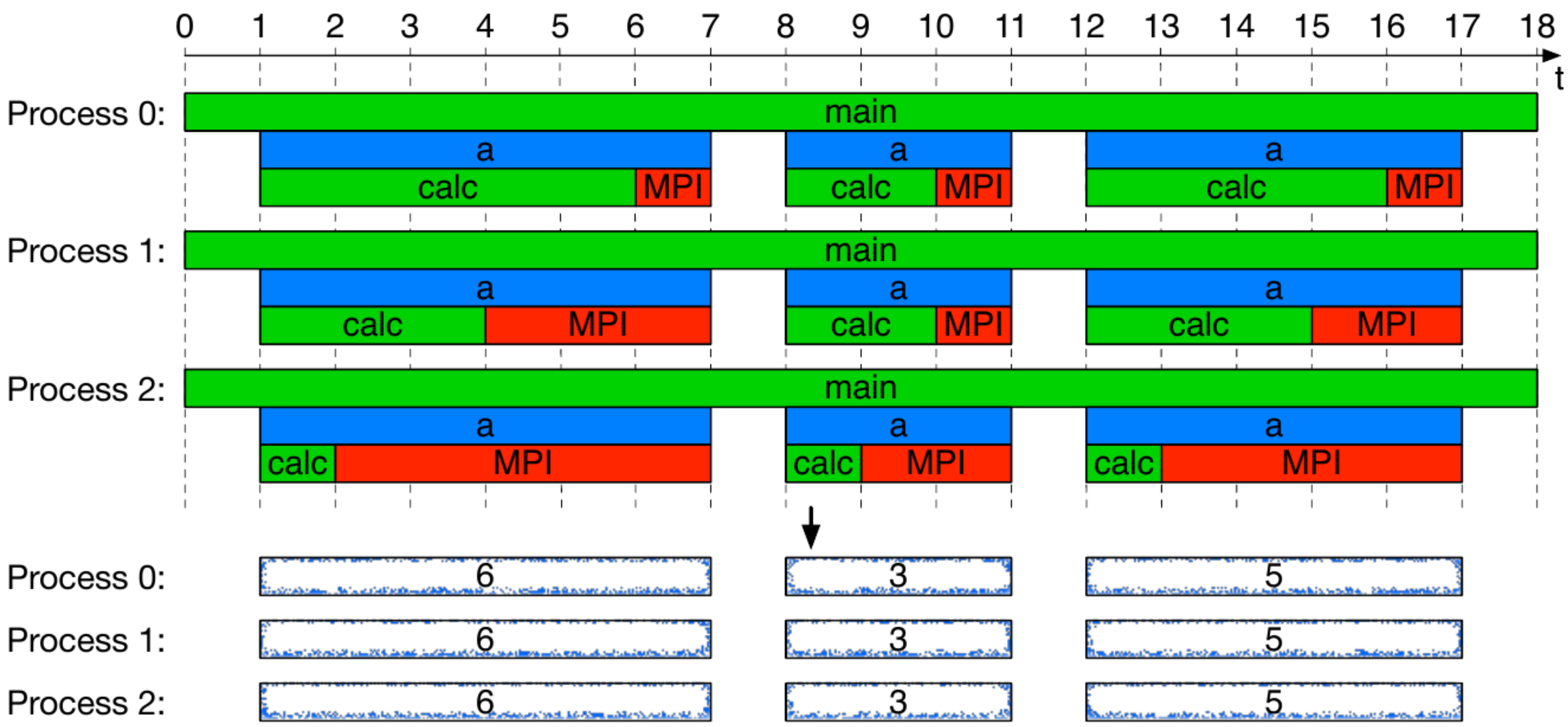
Analyze Runtime Imbalances

- Goal: Detect shifts in execution time of segments

- Assumptions:
  - If an application slows down, likely the time-dominant function runs longer
  - Outlier behavior likely impacts the runtime of the time-dominant function

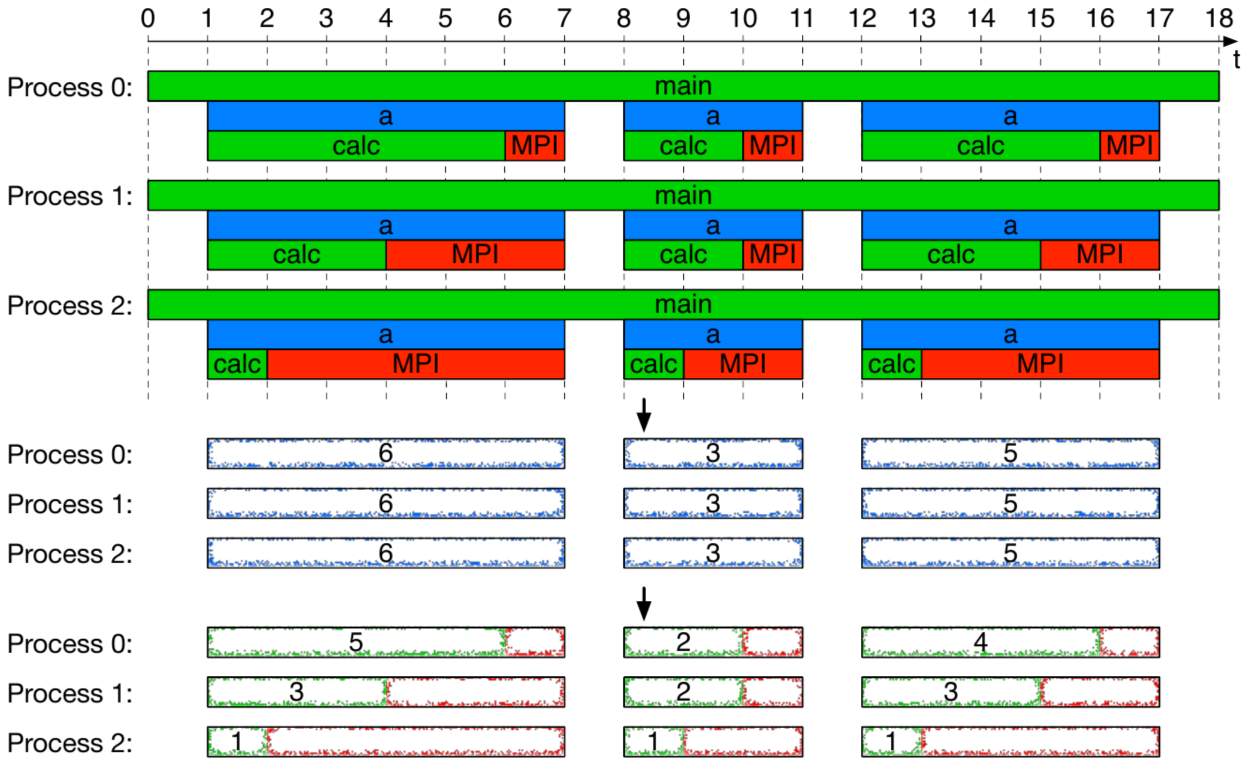
Analyze Runtime Imbalances

- Directly comparing segments has a shortcoming:
  - Included Communication time can even out variations



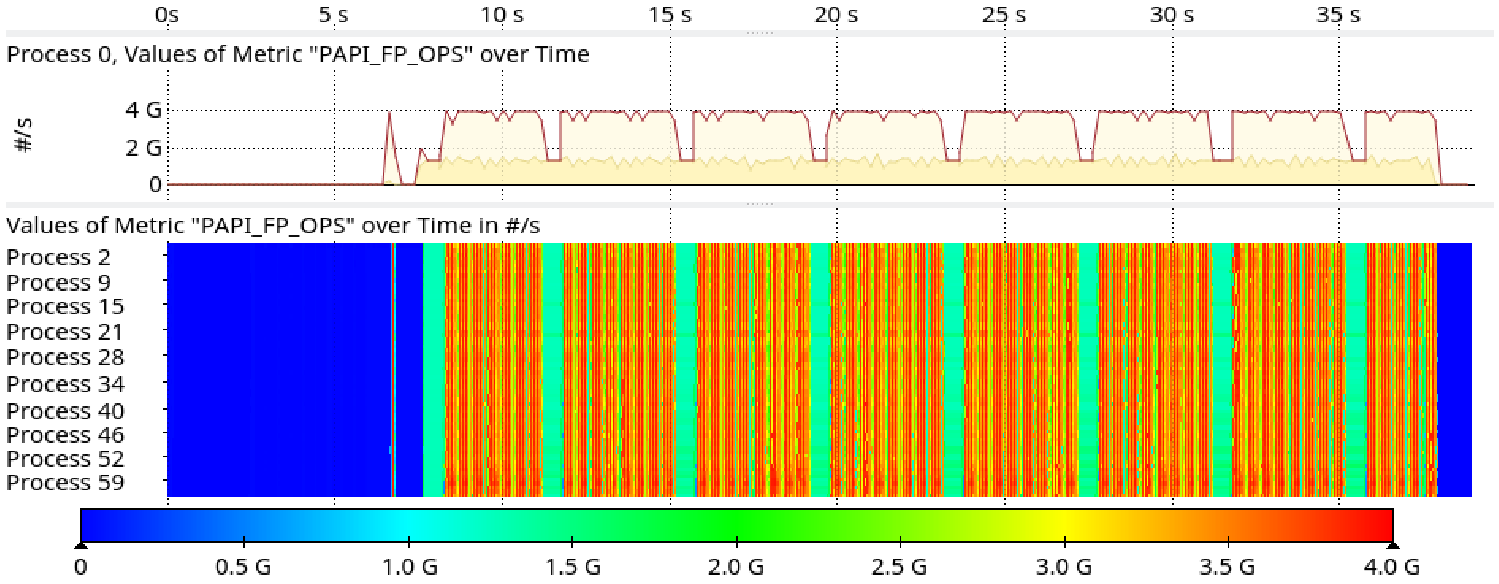
Analyze Runtime Imbalances

- Therefore, ignore synchronization time
  - *Synchronisation-oblivious segment time (SOS-time)*



Visualize Runtime Imbalances

- Implemented in Vampir [5]
- Present SOS-time as a per-process counter

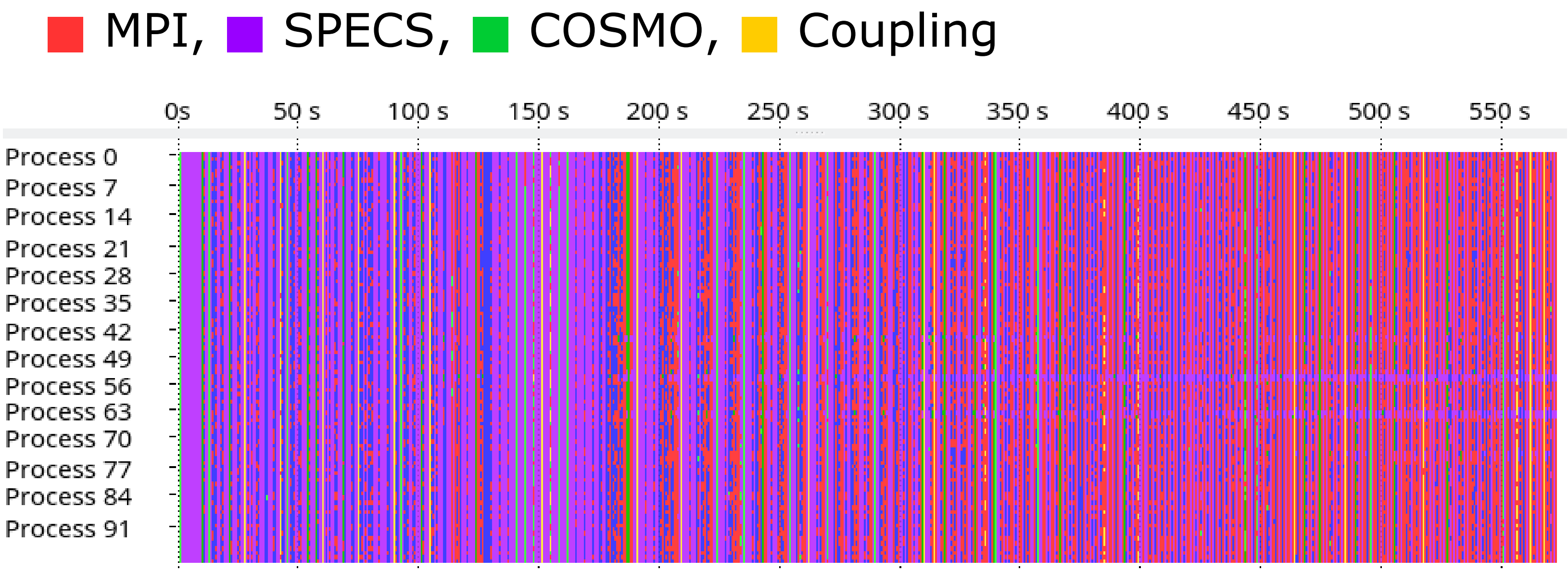


TECHNISCHE UNIVERSITÄT DRESDEN

Contents

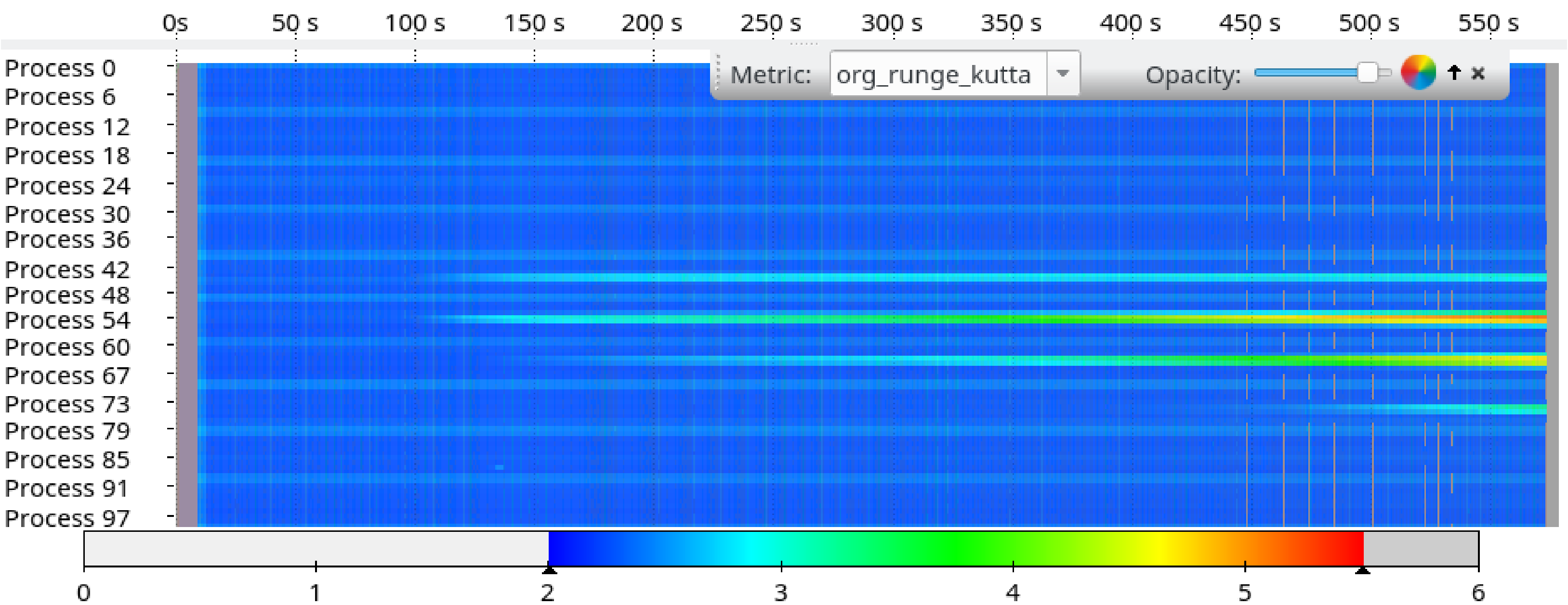
Load-Imbalance

- COSMO-SPECS [6]:
  - COSMO: Regional weather forecast model
  - SPECS: Cloud Micro-physics simulation

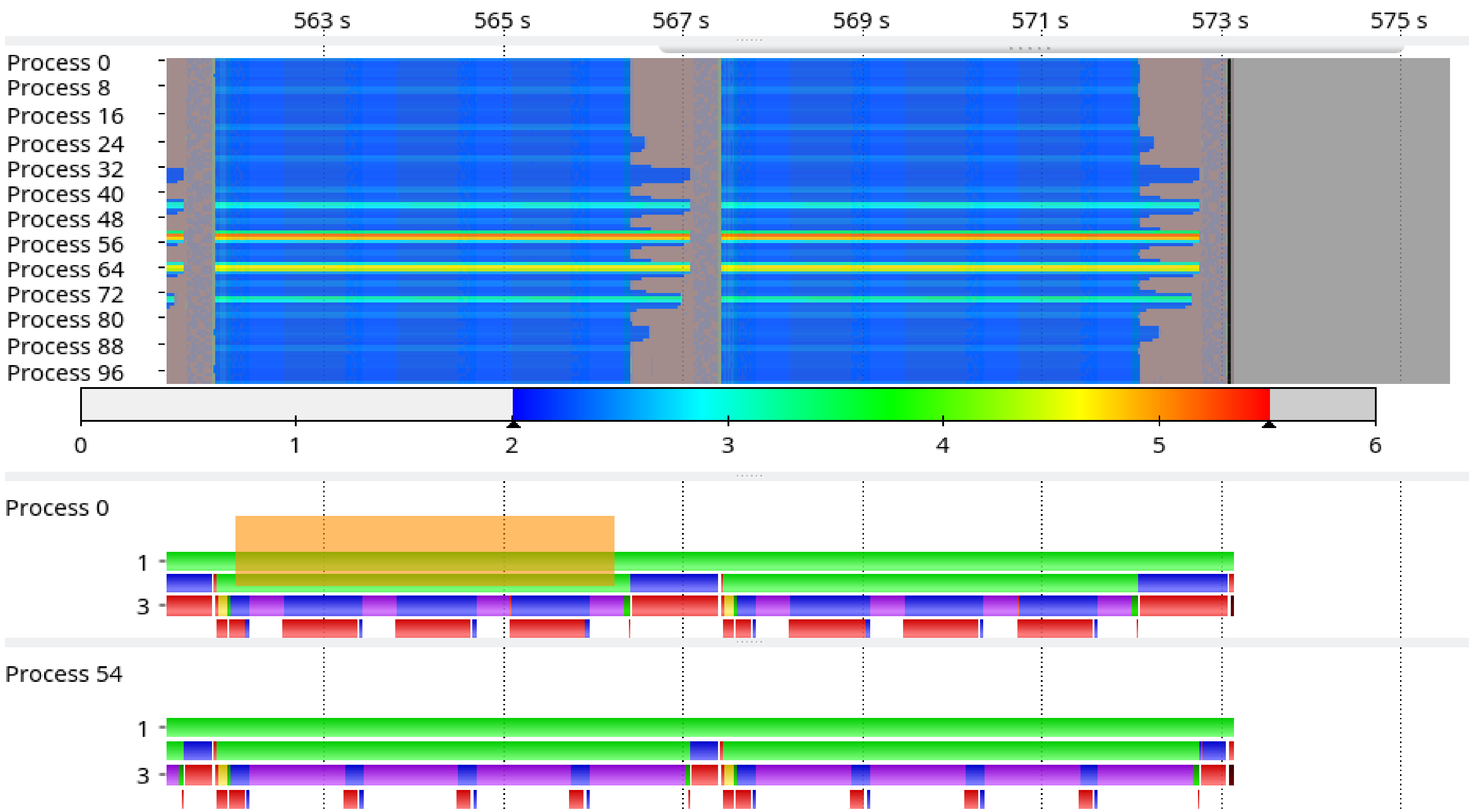


Load-Imbalance

- COSMO and SPECS use the same static domain decomposition
  - Cloud microphysics workload heavily depends on cloud shape



Load-Imbalance

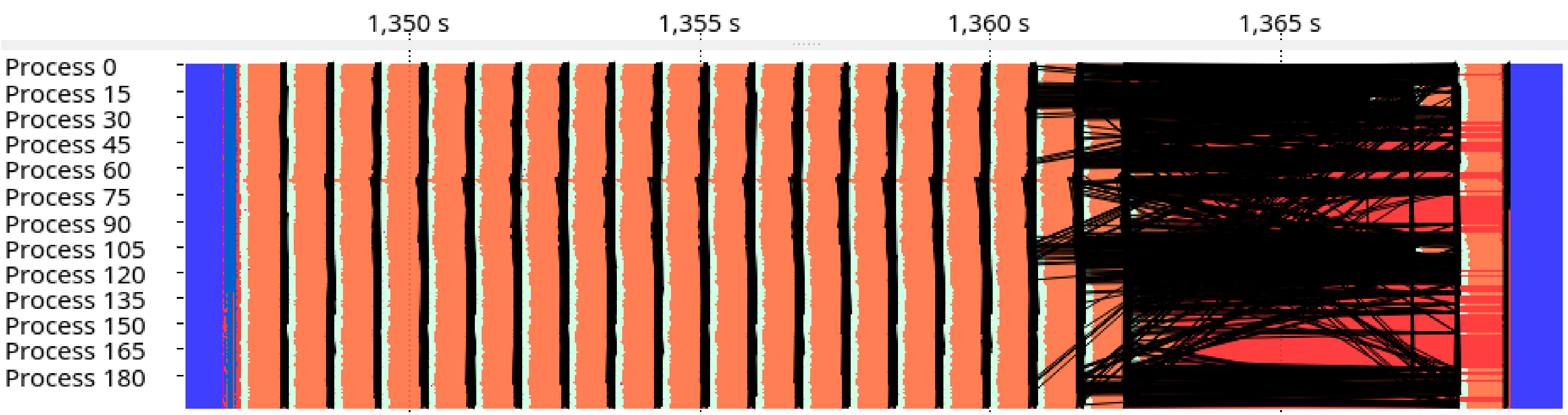


Process Interruption

- COSMO-SPECS+FD4 [7]: Load-balancing for COSMO-SPECS

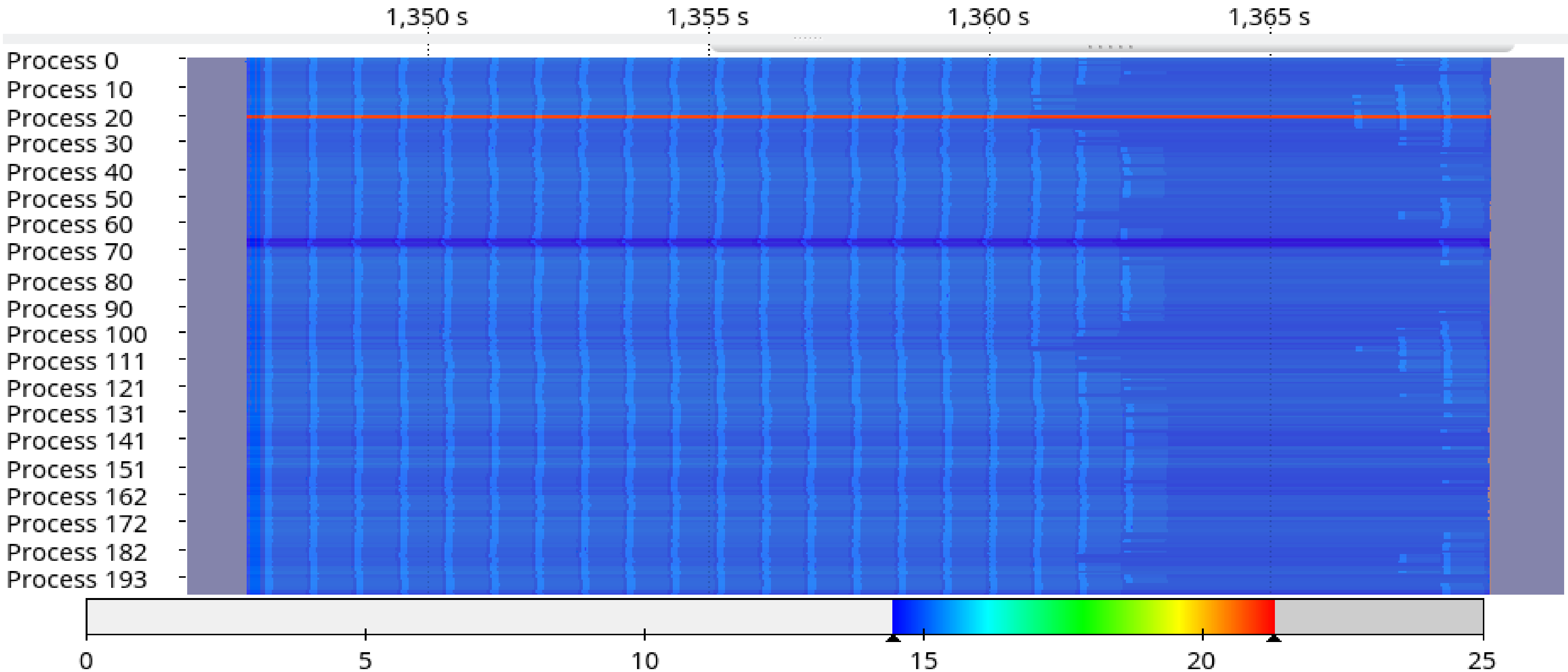
- First analysis detected that only few iterations are slow
- Second run only recorded slow iterations. Focus on one of them

■ MPI, ■ Dropped, ■■ SPECS, / messages



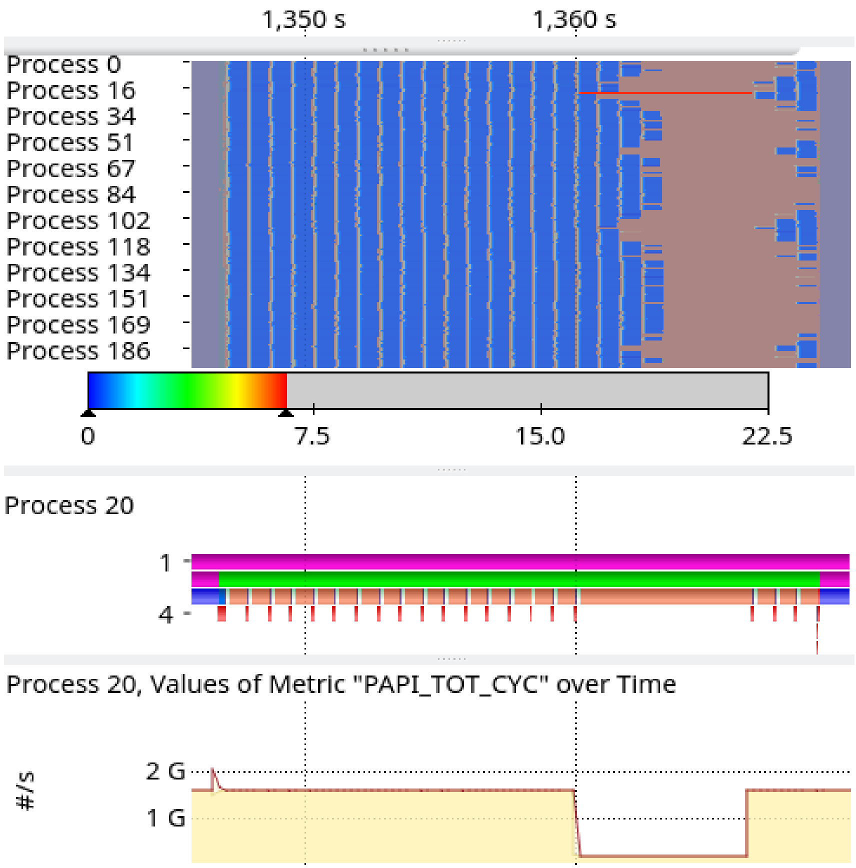
Process Interruption

- Process 20's time-dominant function has a larger SOS-time
- But where exactly is the time spent? → Refine by picking a different function for the metric



Process Interruption

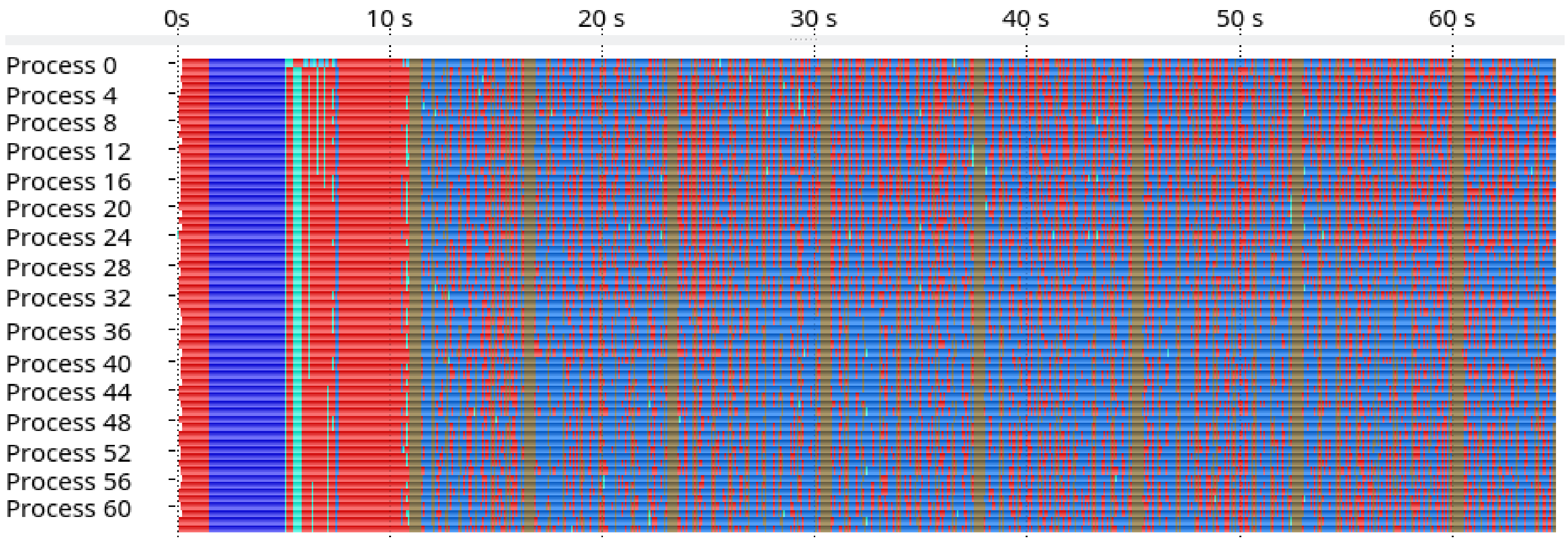
- One sub-iteration is very slow
- The total number of cycles per second during its runtime is ~150M/s vs 1500M/s in other iterations
  → Process is interrupted
- Operating system influence



Floating-Point Exception

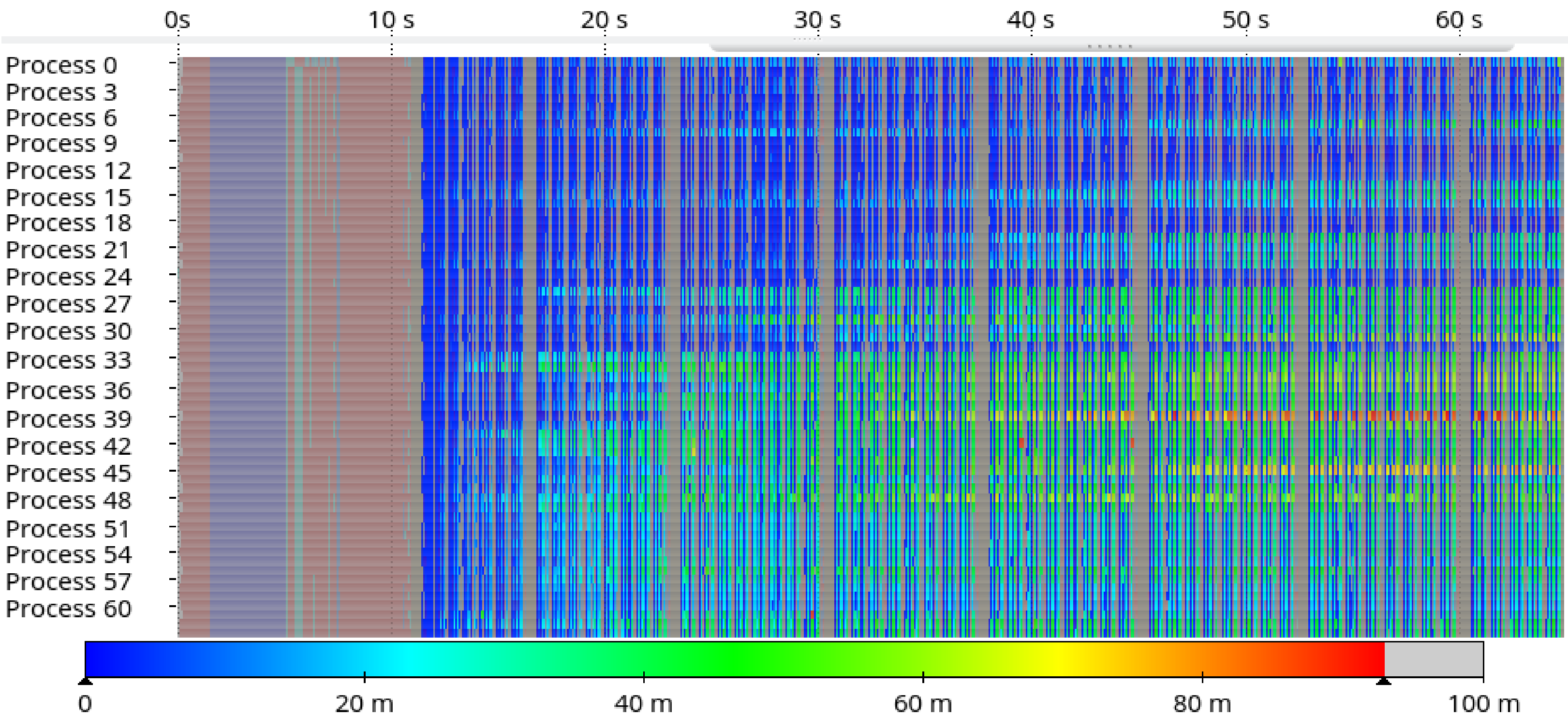
- WRF [8]:
  - Benchmark case: 12km CONUS

MPI, dynamical core, physical parameterization



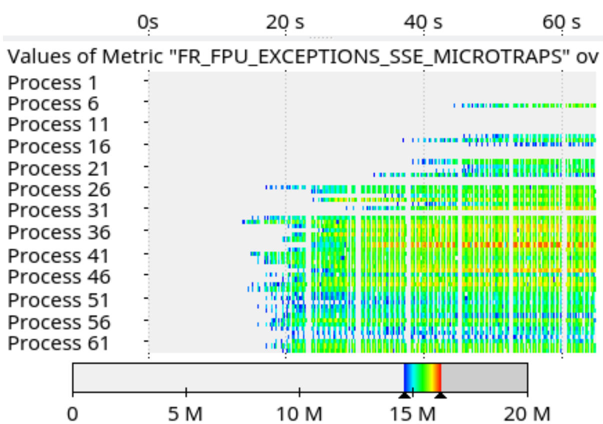
Floating-Point Exception

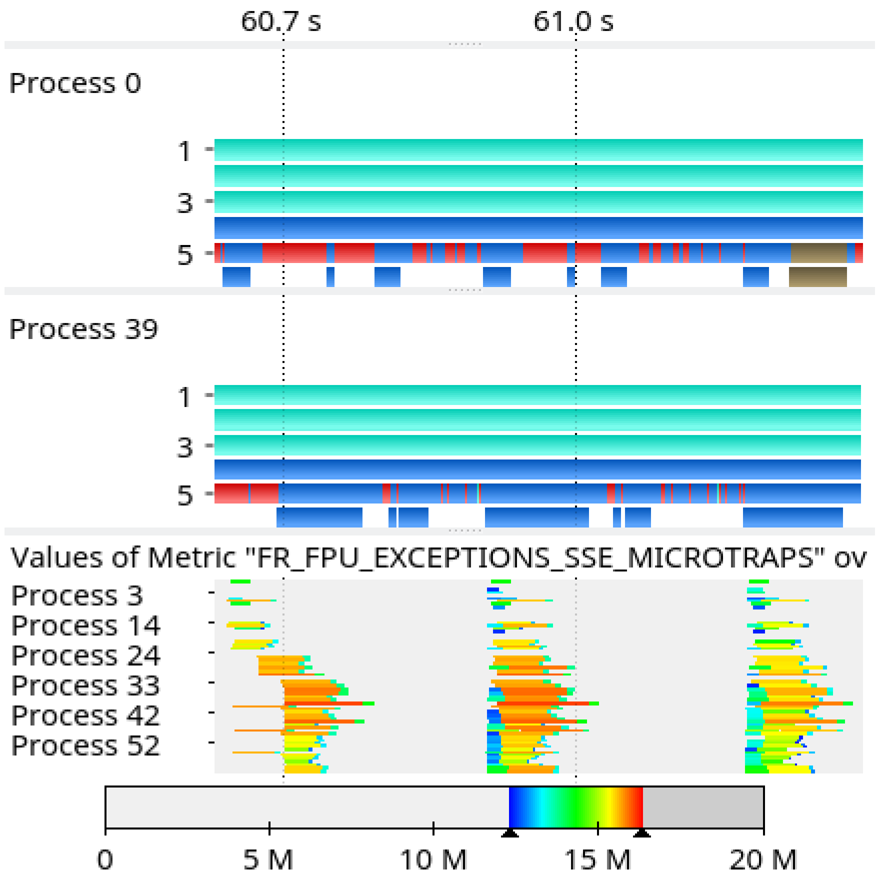
- Varying runtime of the time-dominant function across processes
- Process 39 stands out



Floating-Point Exception

- The function which takes longer is floating-point-intensive
- Number of floating-point exceptions is very high on slow processes



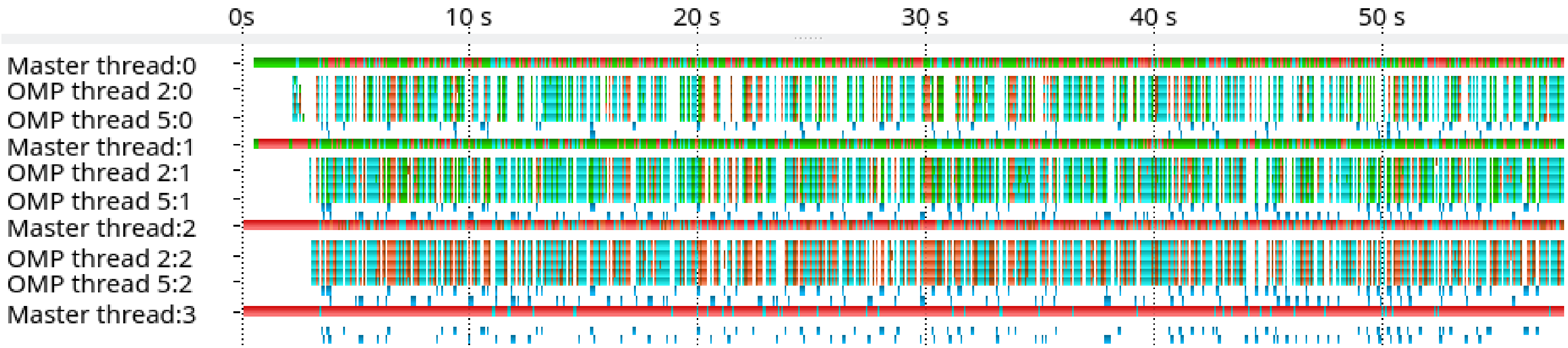


Conclusion

- Effective, light-weight approach that facilitates visual analysis of performance data, i.e. helps find runtime imbalances
  - First, identifies the recurring function with the largest impact on overall program runtime
  - Second, calculates the execution time for each invocation of this function, excluding synchronization time
  - Highlights performance variations by visualizing this synchronization-oblivious segment time
- We demonstrated its effectiveness with three real-world use cases

Future Work

- Use structural clustering [9] to only compare processes doing similar work (e.g. categorize processing elements into process, thread, CUDA thread, ...)



TECHNISCHE UNIVERSITÄT DRESDEN

References

- [1] M. S. Müller, A. Knüpfer, M. Jurenz, M. Lieber, H. Brunst, H. Mix, and W. E. Nagel. Developing Scalable Applications with Vampir, VampirServer and VampirTrace. In Parallel Computing: Architectures, Algorithms and Applications, ParCo 2007, Forschungszentrum Jülich and RWTH Aachen University, Germany, 4-7 September 2007, pages 637–644, 2007.
- [2] A. Knüpfer, C. Rössel, D. Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony, W. E. Nagel, Y. Oleynik, P. Philippen, P. Saviankou, D. Schmidl, S. Shende, R. Tschüter, M. Wagner, B. Wesarg, and F. Wolf. Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir. In Tools for High Performance Computing 2011, pages 79–91. Springer Berlin Heidelberg, 2012.

TECHNISCHE UNIVERSITÄT DRESDEN

References

- [3] V. Pillet, J. Labarta, T. Cortes, and S. Girona. PARAVER: A Tool to Visualize and Analyze Parallel Code. In Proceedings of WoTUG 18: Transputer and occam Developments, pages 17–31, March 1995.
- [4] Intel Trace Analyzer and Collector. http://software.intel.com/en-us/articles/intel-trace-analyzer, Aug. 2016.
- [5] H. Brunst and M. Weber. Custom Hot Spot Analysis of HPC Software with the Vampir Performance Tool Suite. In Proceedings of the 6th International Parallel Tools Workshop, pages 95–114. Springer Berlin Heidelberg, September 2012.

TECHNISCHE UNIVERSITÄT DRESDEN

References

- [6] V. Grützun, O. Knoth, and M. Simmel. Simulation of the influence of aerosol particle characteristics on clouds and precipitation with LM-SPECS: Model description and first results. Atmospheric Research, 90(24):233–242, 2008.
- [7] M. Lieber, V. Grützun, R. Wolke, M. S. Müller, and W. E. Nagel. Highly Scalable Dynamic Load Balancing in the Atmospheric Modeling System COSMO-SPECS+FD4. In Proc. PARA 2010, volume 7133 of LNCS, pages 131–141, 2012.

TECHNISCHE UNIVERSITÄT DRESDEN

References

- [8] G. Shainer, T. Liu, J. Michalakes, J. Liberman, J. Layton, O. Celebioglu, S. A. Schultz, J. Mora, and D. Cownie. Weather Research and Forecast (WRF) Model Performance and Profiling Analysis on Advanced Multi-core HPC Clusters. In 10th LCI International Conference on High-Performance Clustered Computing, 2009.
- [9] Brendel, R., et al. Structural Clustering: A New Approach to Support Performance Analysis at Scale. No. LLNL-CONF-669728. Lawrence Livermore National Laboratory (LLNL), Livermore, CA, 2015.