Bill Williams
IAK, ZIH

Trace-based Performance Analysis with Score-P and Vampir

OLCF Frontier User Training

25 August, 2023

What are Score-P and Vampir?

Score-P: *measurement* system, primarily instrumentation-based

Vampir: scalable *trace visualizer* for post-mortem analysis

When are these the right tools?

You have a performance problem

You suspect the problem is complicated:

- Interaction between forms of parallelism
- Problem has dynamic behavior over space and/or time

You want to be able to *filter* your results at a fine-grained level

All the pieces matter!

How to use them?

Score-P: module load correct version for your PrgEnv and toolchain
- Caution: the module is toolchain-specific!
- Works as a prefix on compile/link lines or as a compiler wrapper
- Currently: PrgEnv-amd, `module load scorep-amdclang`

Vampir: multiple methods of use
- Local client, start a server @ OLCF
- Local client, transfer or mount trace directory
- Client at OLCF, use VNC/X11 forwarding
- Directions available with load message from `module load vampir`

The standard workflow

1. Collect *reference* data so you can evaluate instrumentation overhead
2. *Build* your application with Score-P
3. Run your instrumented application, which will by default collect a *profile*
4. *Score* this profile with the scorep-score tool, which will tell you what regions are responsible for how much instrumentation overhead
5. Build a *filter file* with scorep-score and manual editing to remove uninteresting/high-overhead regions from the instrumentation output
6. Configure your next run to use this filter and produce a *trace*
7. Visualize the trace data in Vampir!

Using Vampir

Main types of views: *timeline* and *summary*

- Timelines present tracing data, location on Y-axis and time on X-axis
- Summaries present profiling data, aggregated over the currently selected time window
  - Example: “how much time do I spend in MPI functions during this phase?”

Customization for better analysis

Almost everything can be edited and saved!

- *Function groups* can be created to aggregate related pieces of code in the visualization: e.g. all particle—particle interaction calculations in the same group with a chosen color
- Custom *derived metrics* can be calculated based on existing metrics

Building instrumented applications

Plain makefiles: `scorep <instrumenter flags> <compiler> <original args>`, `scorep <instrumenter flags> <linker> <original args>`

autotools/cmake: use *compiler wrappers* `scorep-compiler <instrumenter flags> <original args>`

— allows `SCOREP_WRAPPER=off` for configure/CMake invocations

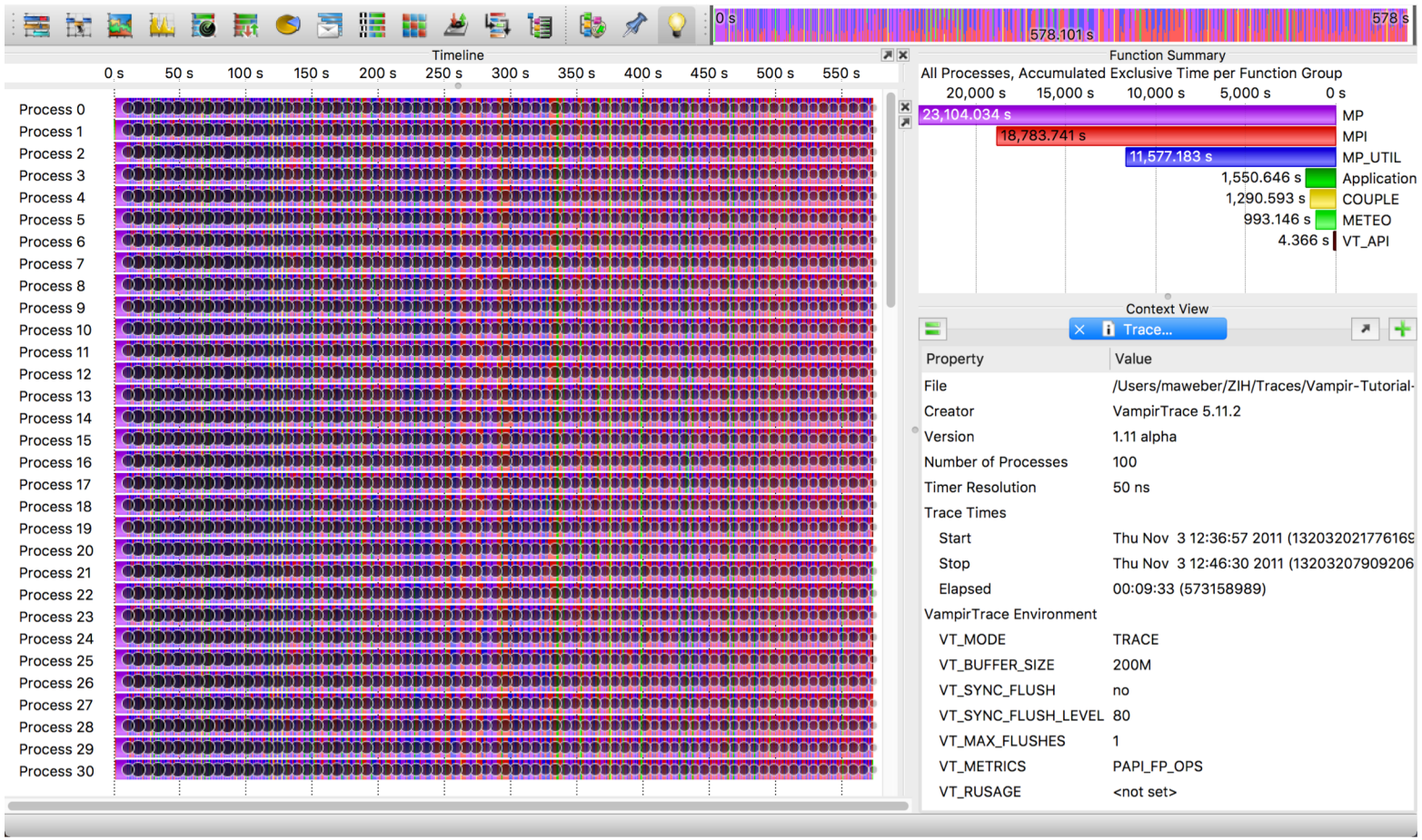
TECHNISCHE UNIVERSITÄT DRESDEN

ZIH

DRESDEN concept

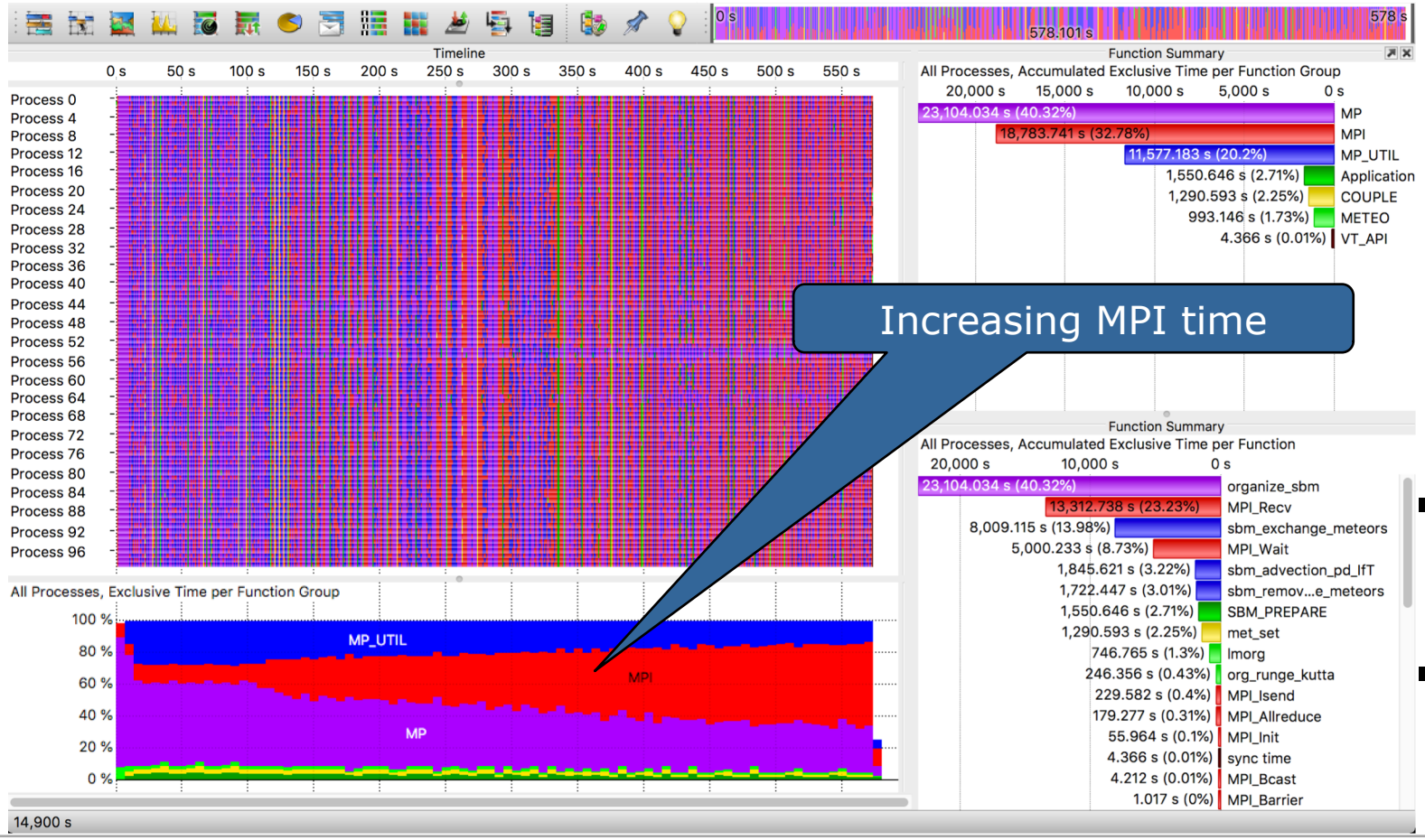
Vampir case study: dynamic load imbalance with COSMO-SPECS

COSMO-SPECS



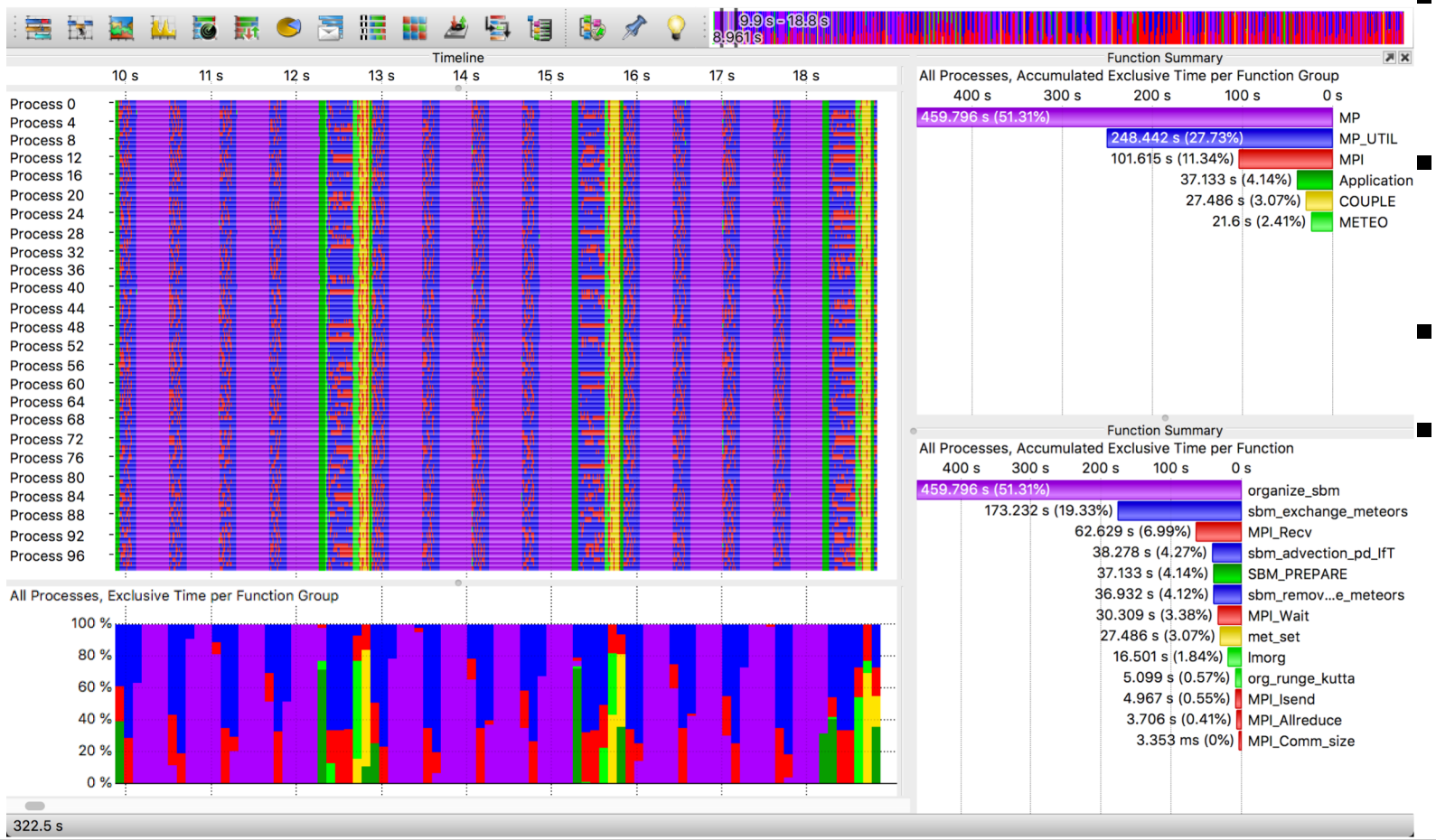
- Weather forecast code COSMO-SPECS
- Run with 100 processes
- COSMO: weather model (METEO group)
- SPECS: microphysics for accurate cloud calculation (MP and MP_UTIL group)
- Coupling of both models done in COUPLE group

COSMO-SPECS



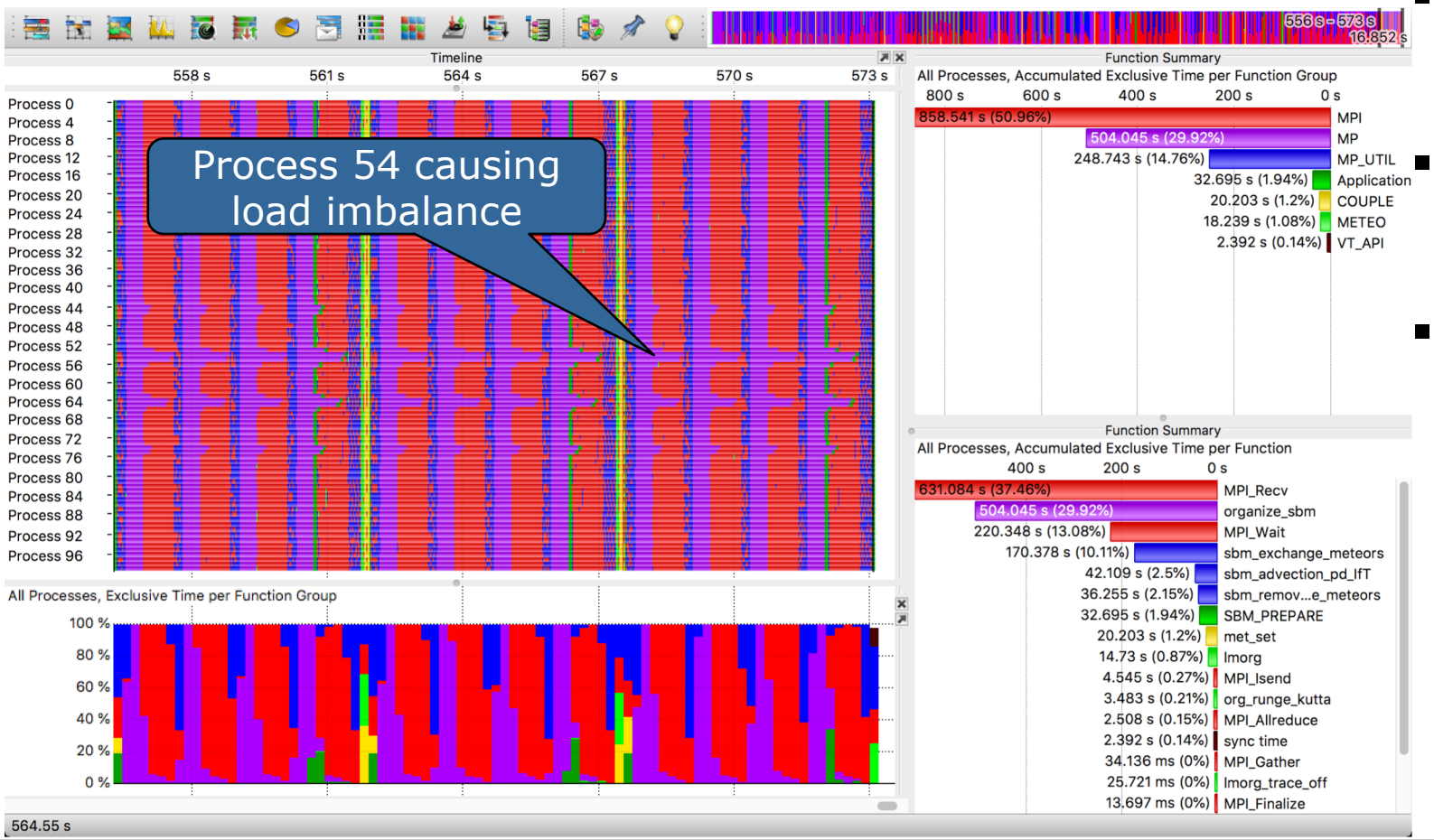
- Compared to METEO, MP and MP_UTIL are very compute intensive, however this is due to more complex calculations and no performance issue
- Problem: >32% of time spent in MPI
- MPI runtime share increases throughout the application run

COSMO-SPECS



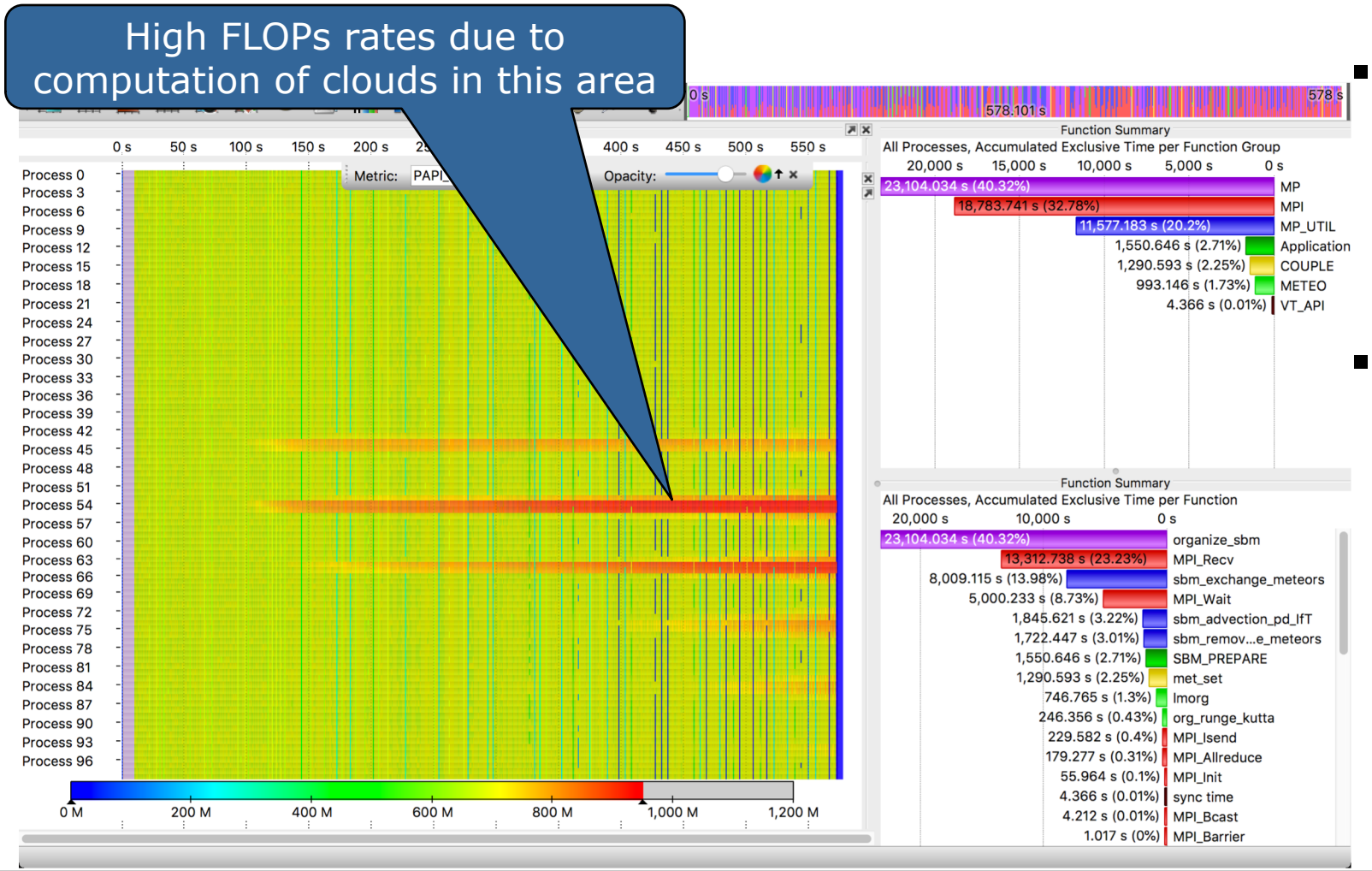
- Zoom into the first three iterations
- MP/MP_UTIL perform four sub-steps in one iteration
- Low MPI time share
- Everything is balanced and looks okay

COSMO-SPECS



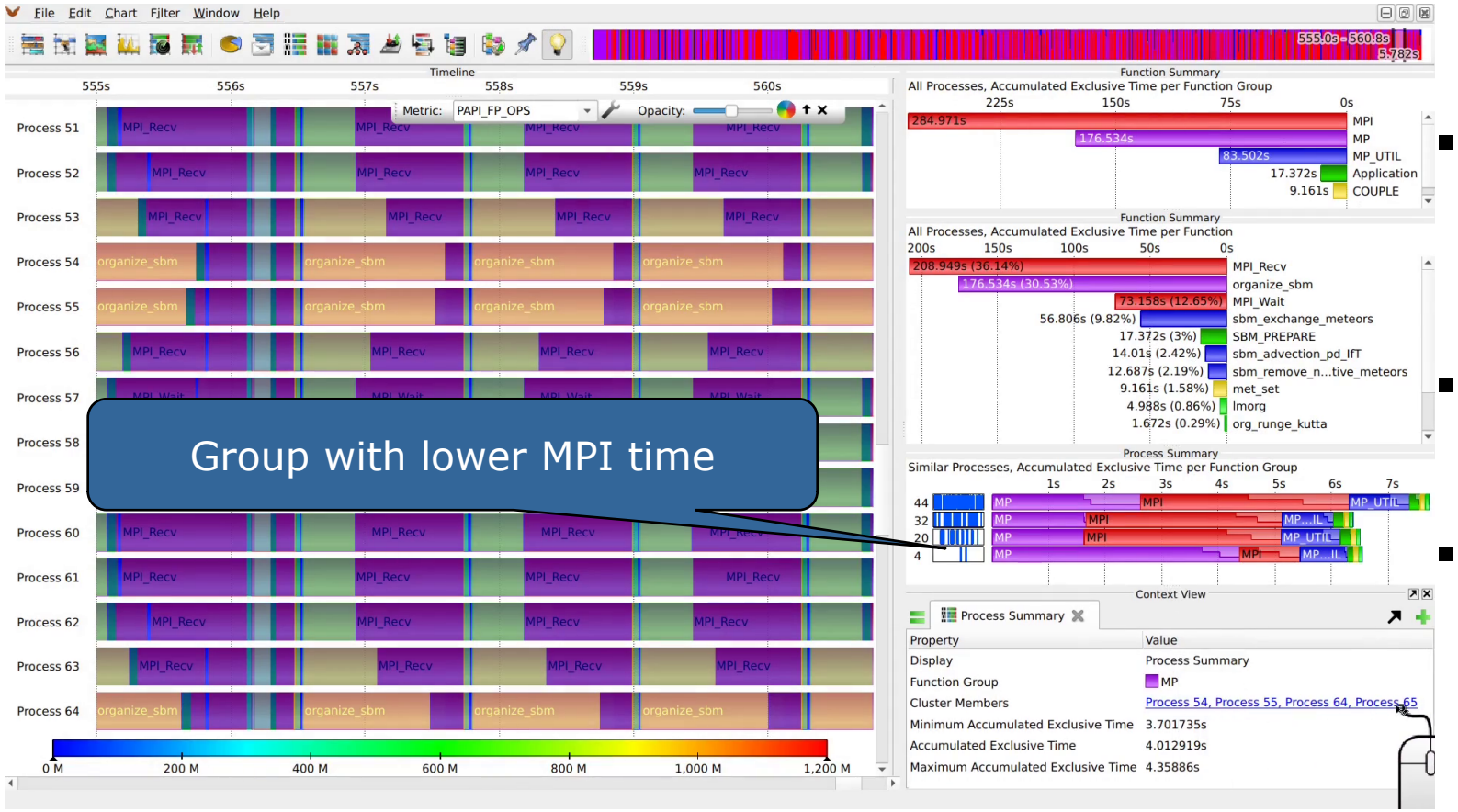
- Zoom into the last three iterations
- Very high MPI time share (>50%)
- Large load imbalance caused by MP functions around **`Process 54`** and **`Process 64`**

COSMO-SPECS



- **`PAPI_FP_OPS`** counter showing higher FLOPs rates on processes causing the imbalance
- Reason for imbalance: Static grid used for distribution of processes. Depending on the weather, expensive cloud computations (MP group) may be only necessary on some processes

COSMO-SPECS



- Process Summary helps finding outliers
- Groups processes by their behavior (similar call/duration profile)
- Number of expected groups is variable
- In this case 4 yields the best results

Further info and support

Local installation support: ask OLCF first!

- What is best practice for this installation?
- Where do I find XYZ?
- Can we get support for another toolchain installed?
- Can we get feature X enabled?

Vendor support: support@score-p.org, service@vampir.eu

- Bug reports
- Feature requests
- Other problems OLCF support can't help with

Further training options:

- (Hopefully) half-day session at OLCF coming soon
- Tutorials at SC/ISC
- VIHPS tuning workshops: 1 week of bring-your-own-code performance tools training
- Slides from VIHPS training available at https://www.vi-hps.org/training/course-material/index.html