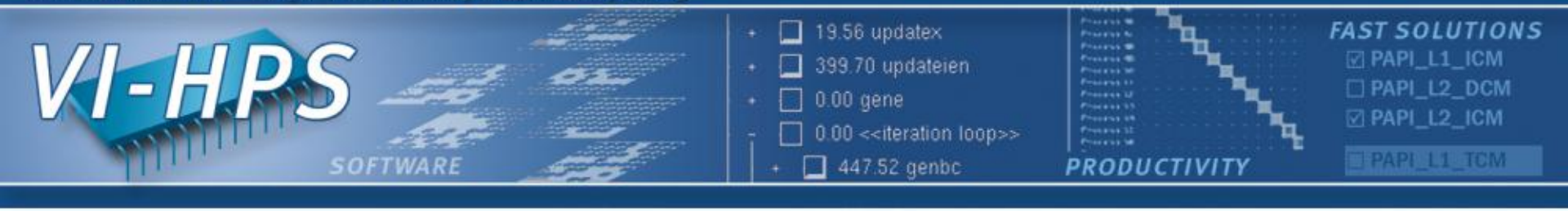


Event Tracing with VAMPIR

Ventsislav Petkov

Technische Universität München

petkovve@in.tum.de

Overview

- Introduction
- Vampir Displays
- Advanced topics
  - GPU Support
  - Scalability
- Finding Performance Bottlenecks
- Conclusion & Outlook

Program Instrumentation

- Detect run-time events (points of interest)
- Pass information to run-time measurement library

Profile Recording

- Collect aggregated information (Time, Counts, …)
- About program and system entities
  - functions, loops, basic blocks per process/thread

Trace Recording

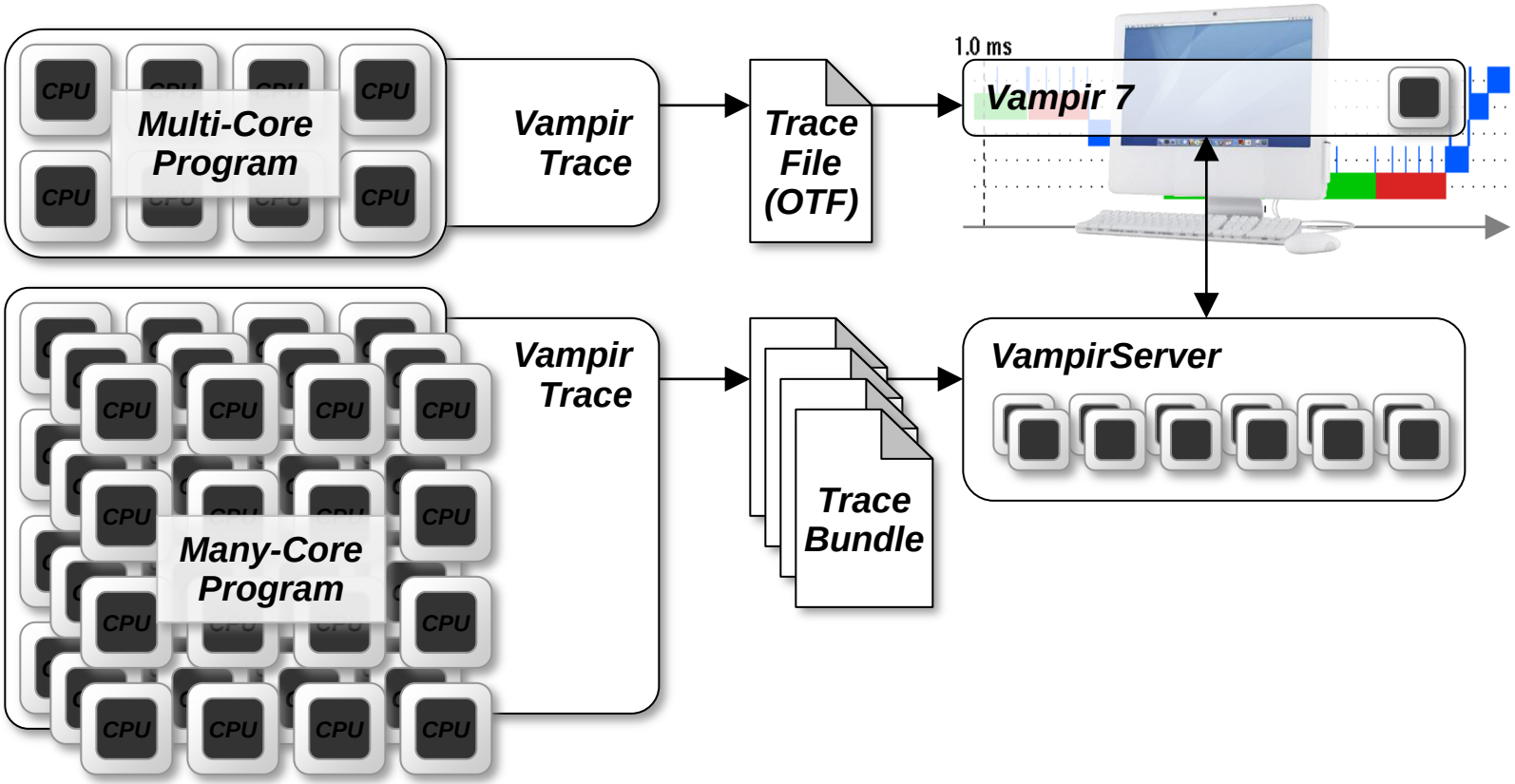
- Individual event records
- Precise time stamp, process/thread ID
- Event specific information

Trace Visualization

- Alternative and supplement to automatic analysis
- Show dynamic run-time behavior visually
- Provide statistics and performance metrics
  - global timeline for parallel processes/threads
  - process timeline plus performance counters
  - statistic summary display
  - communication statistics, more …
- Interactive browsing, zooming, selecting
  - adapt statistics to zoom level (time interval)
  - also for very large and highly parallel traces

Vampir Toolset Architecture

Scalable parallel Vampir visualization architecture



Overview

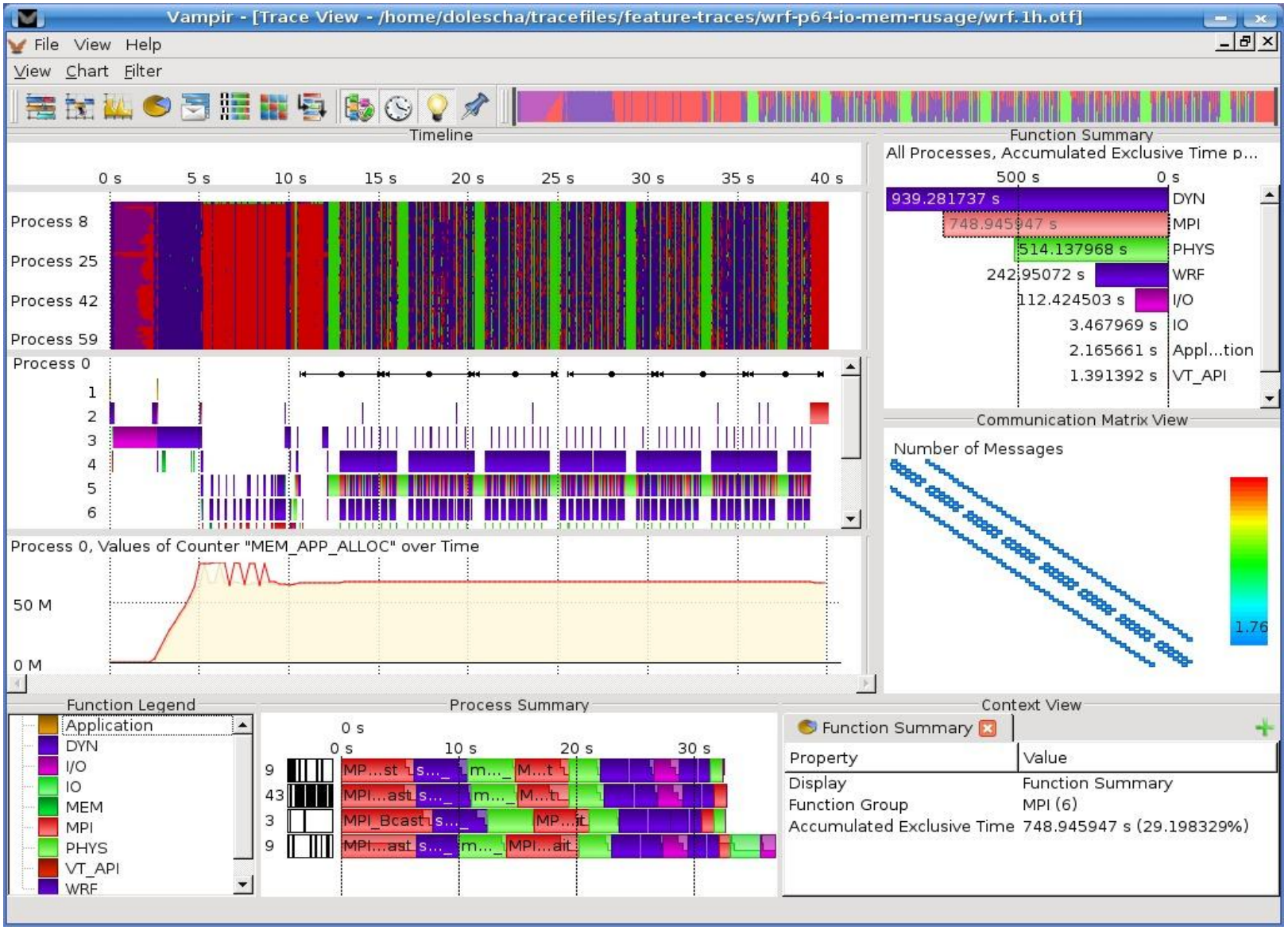
- Introduction
- Vampir Displays
- Advanced topics
  - GPU Support
  - Scalability
- Finding Performance Bottlenecks
- Conclusion & Outlook

Vampir Displays

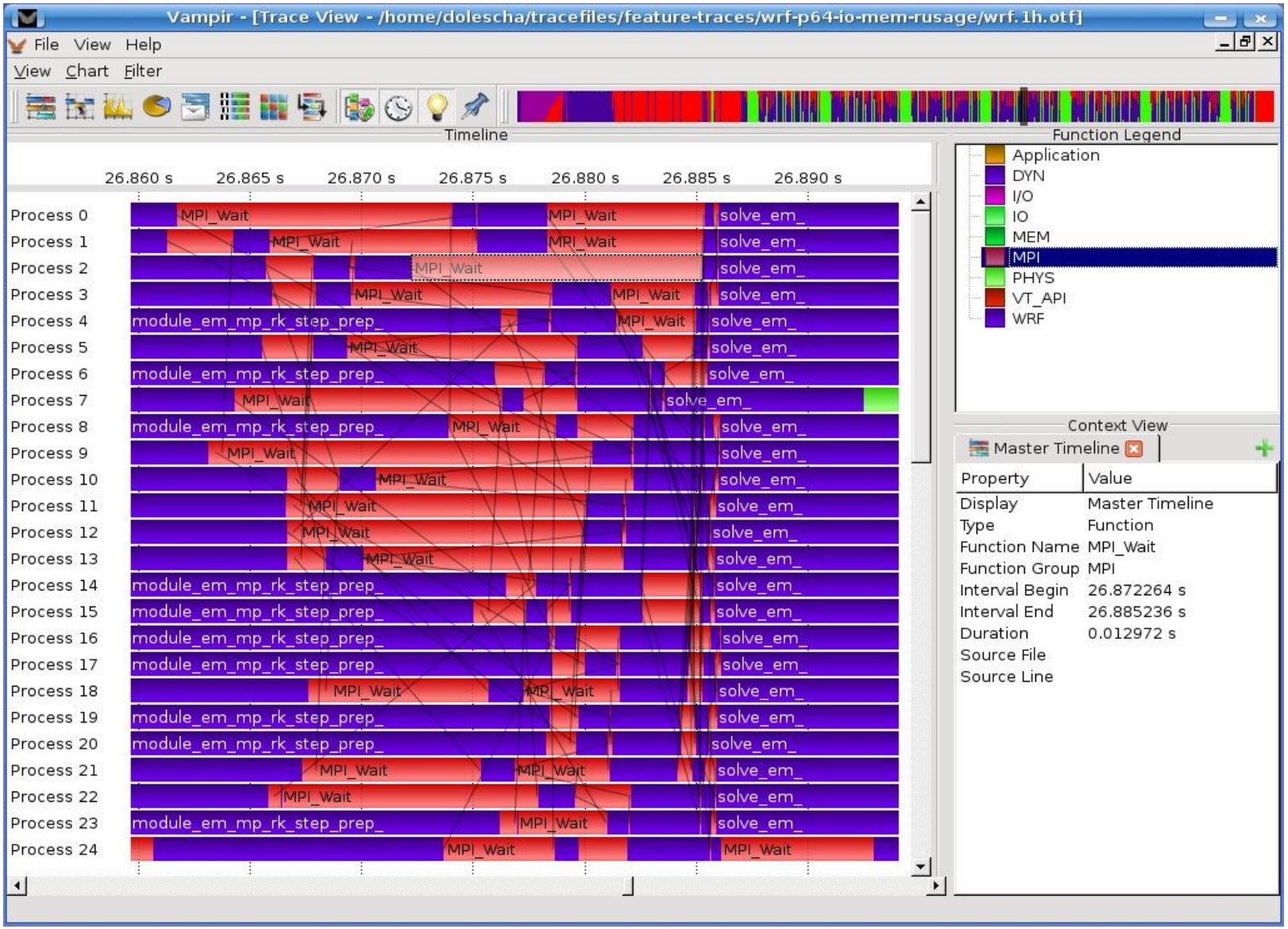
The main displays of Vampir:

- Master Timeline
- Process and Counter Timeline
- Function Summary
- Message Summary
- Process Summary
- Communication Matrix
- Call Tree

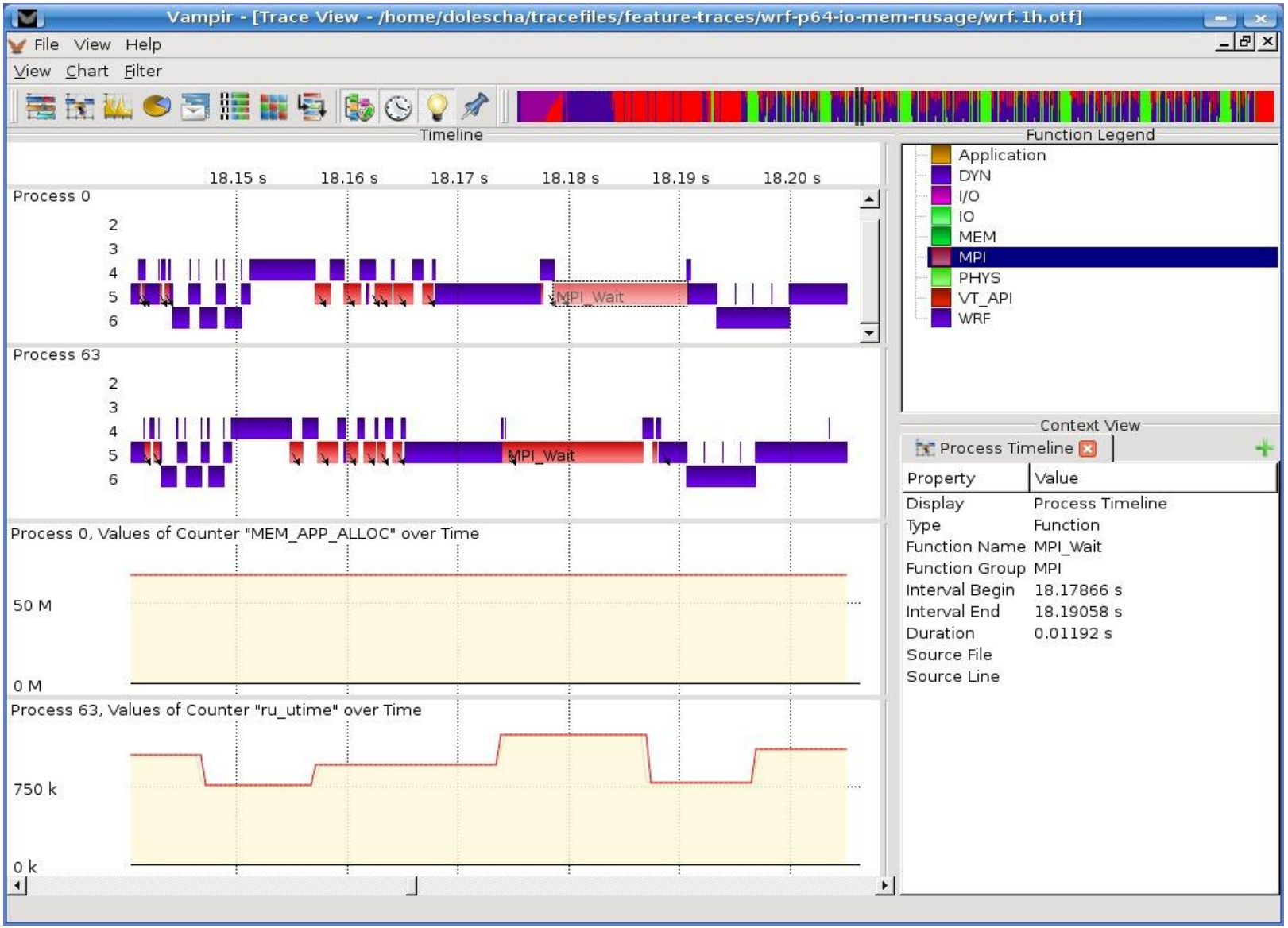
Vampir 7 Display Overview



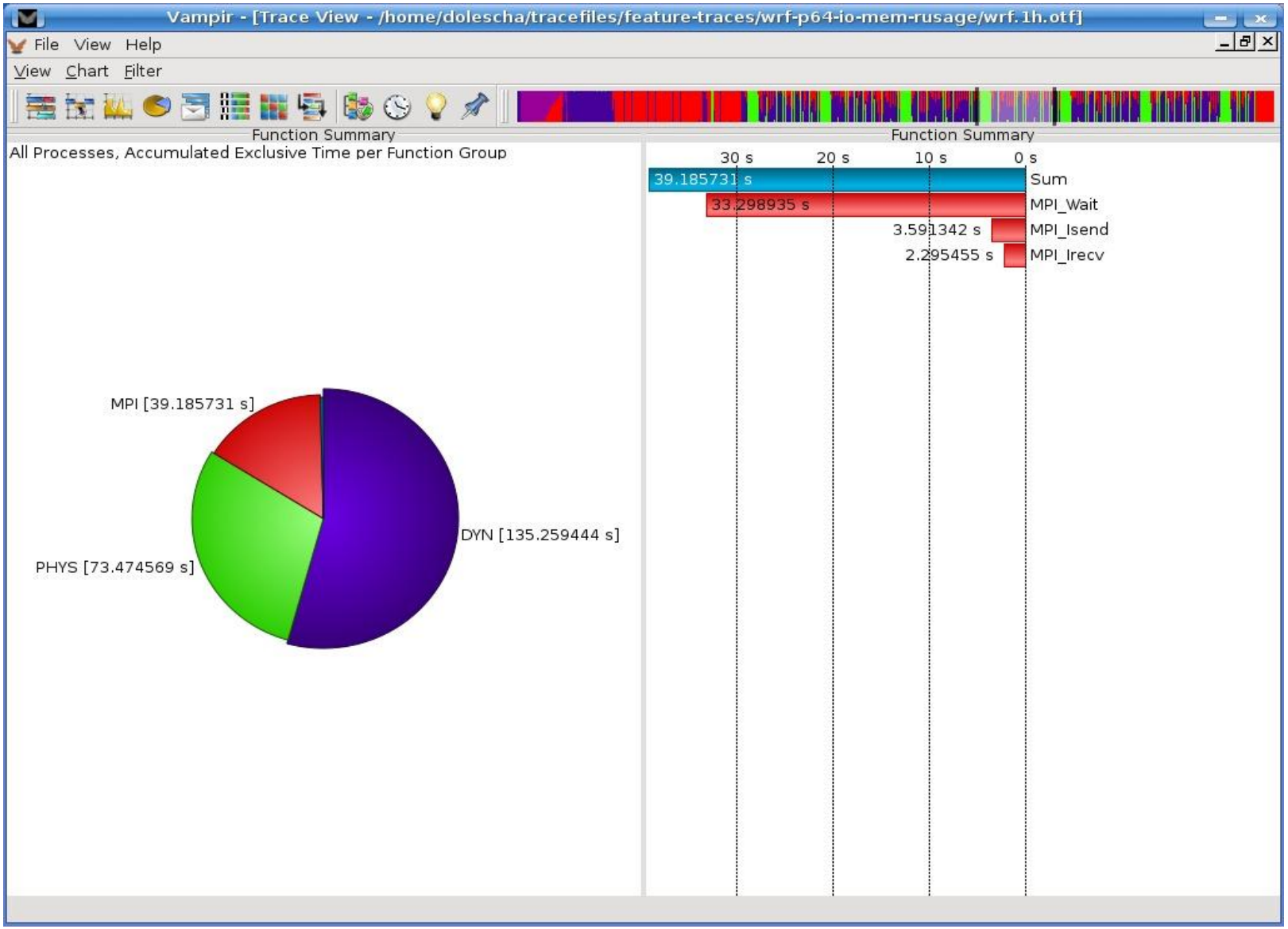
Master Timeline



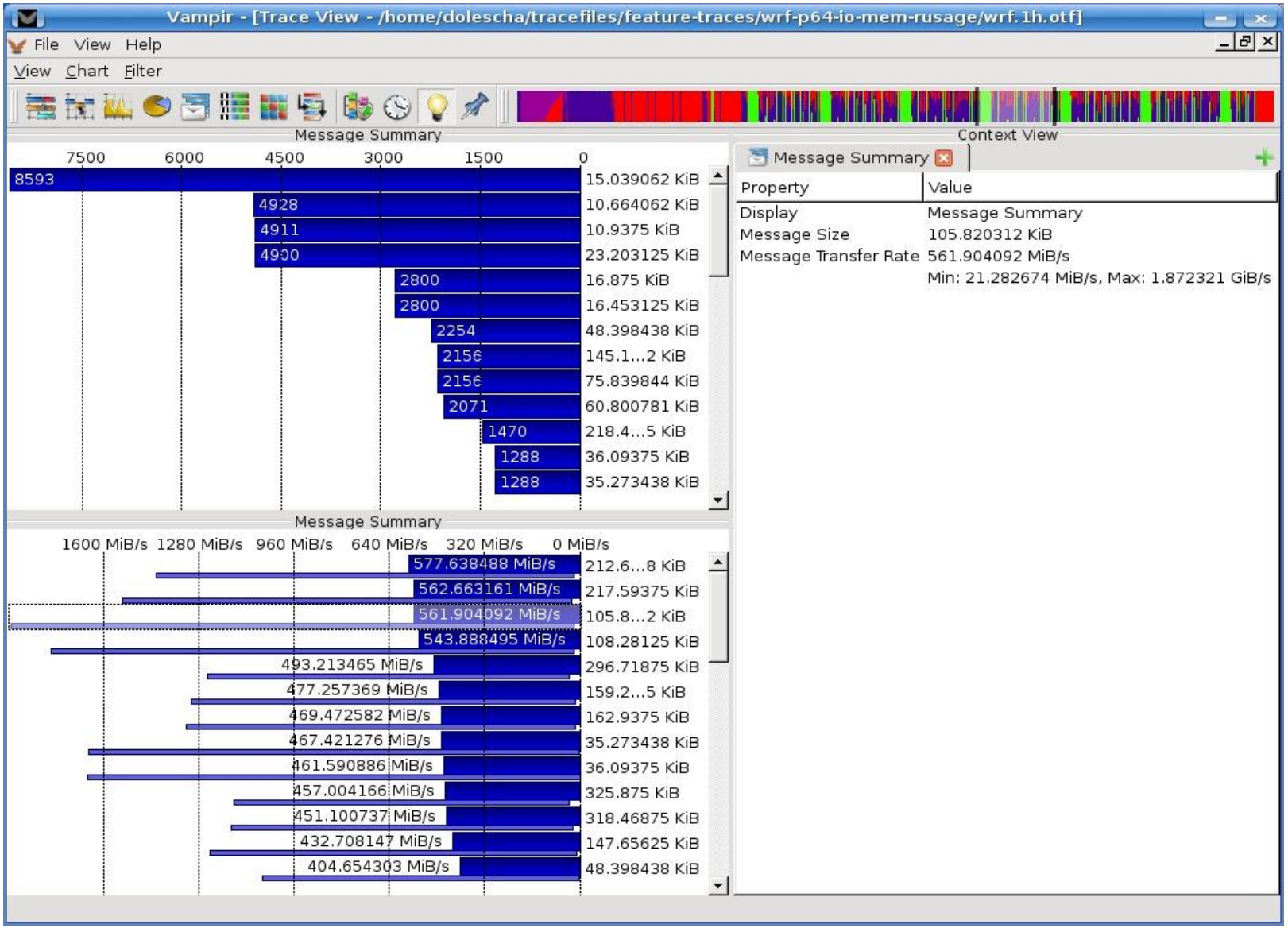
Process and Counter Timelines



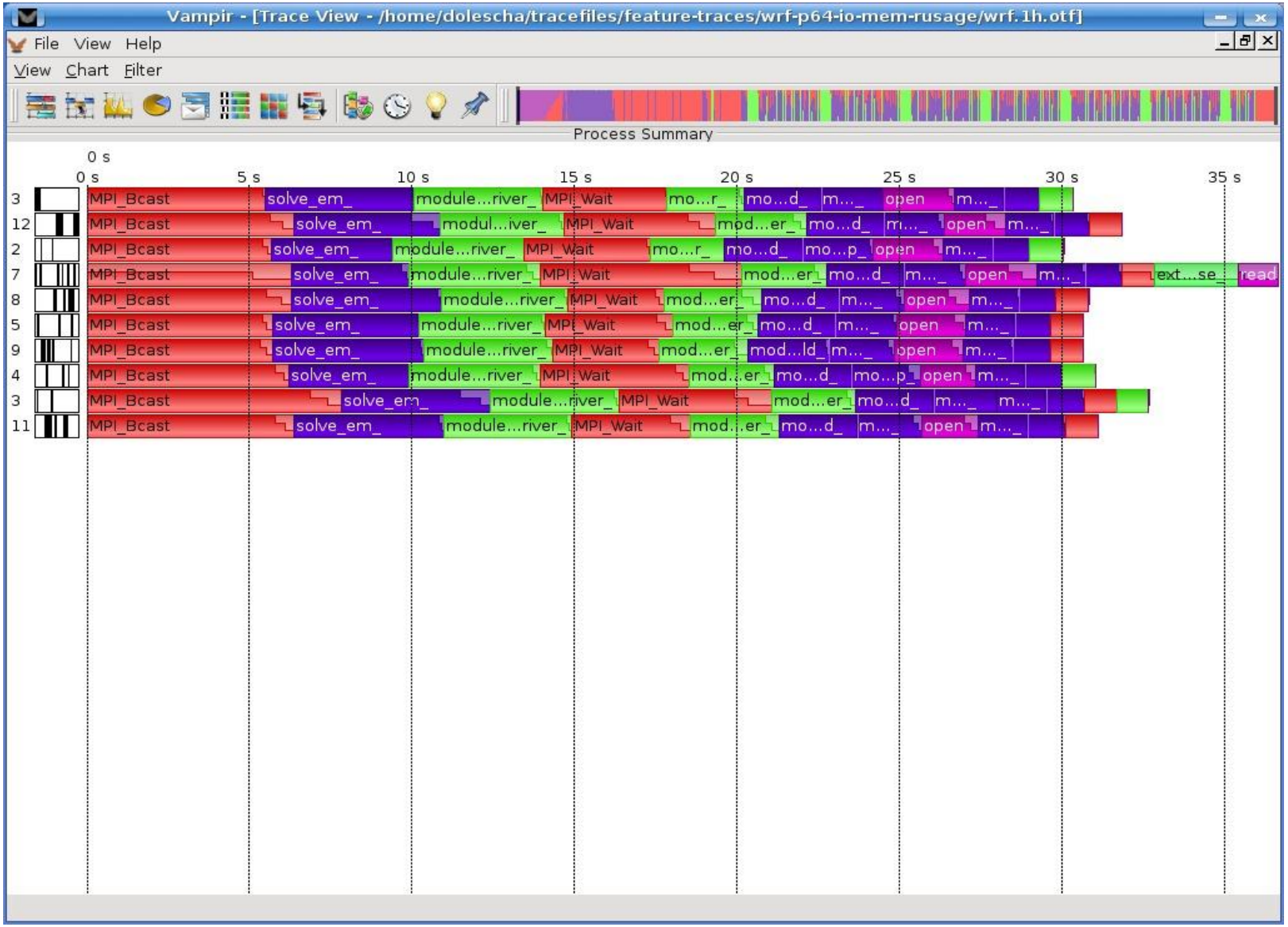
nction Summary



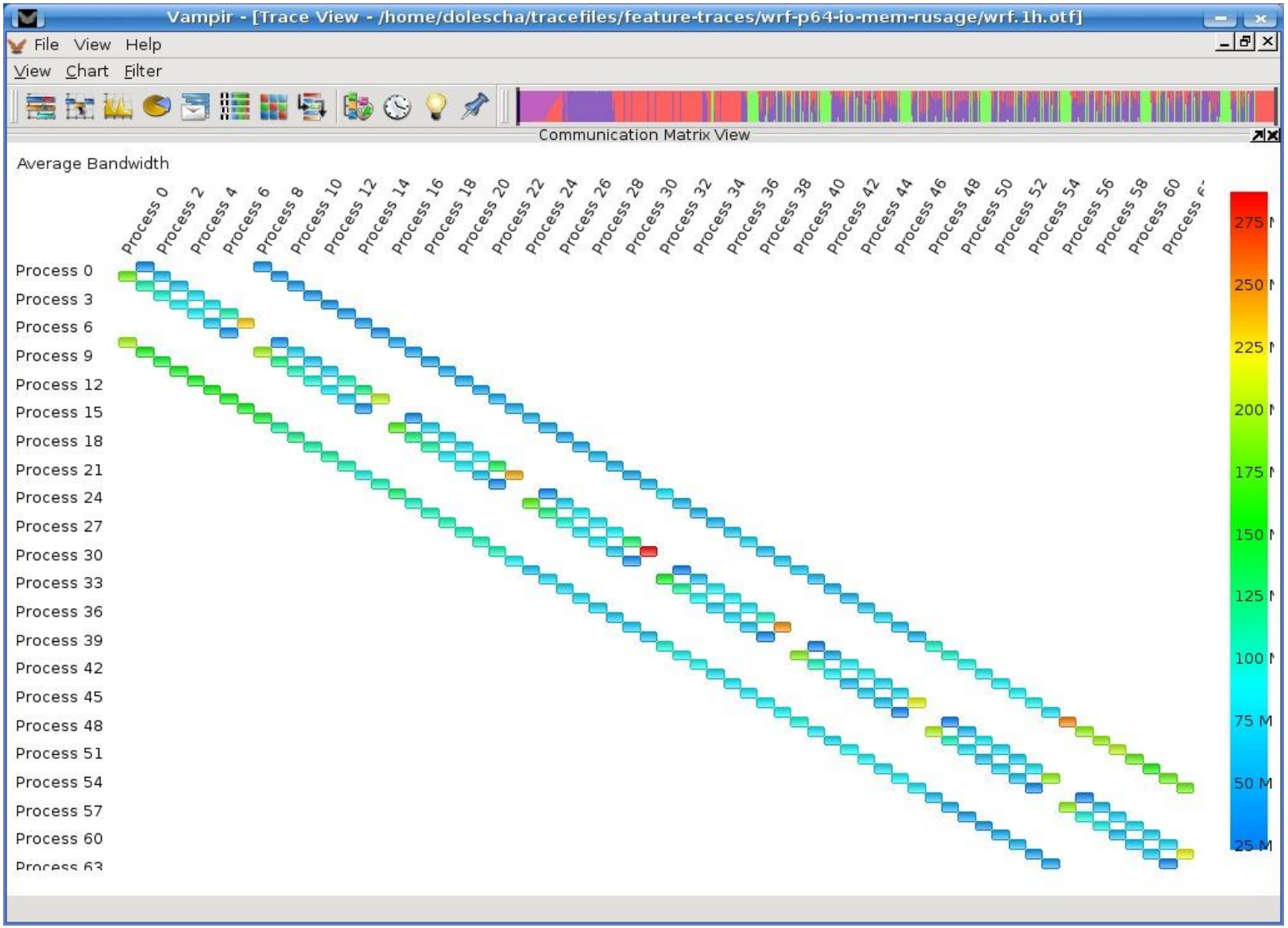
ssage Summary



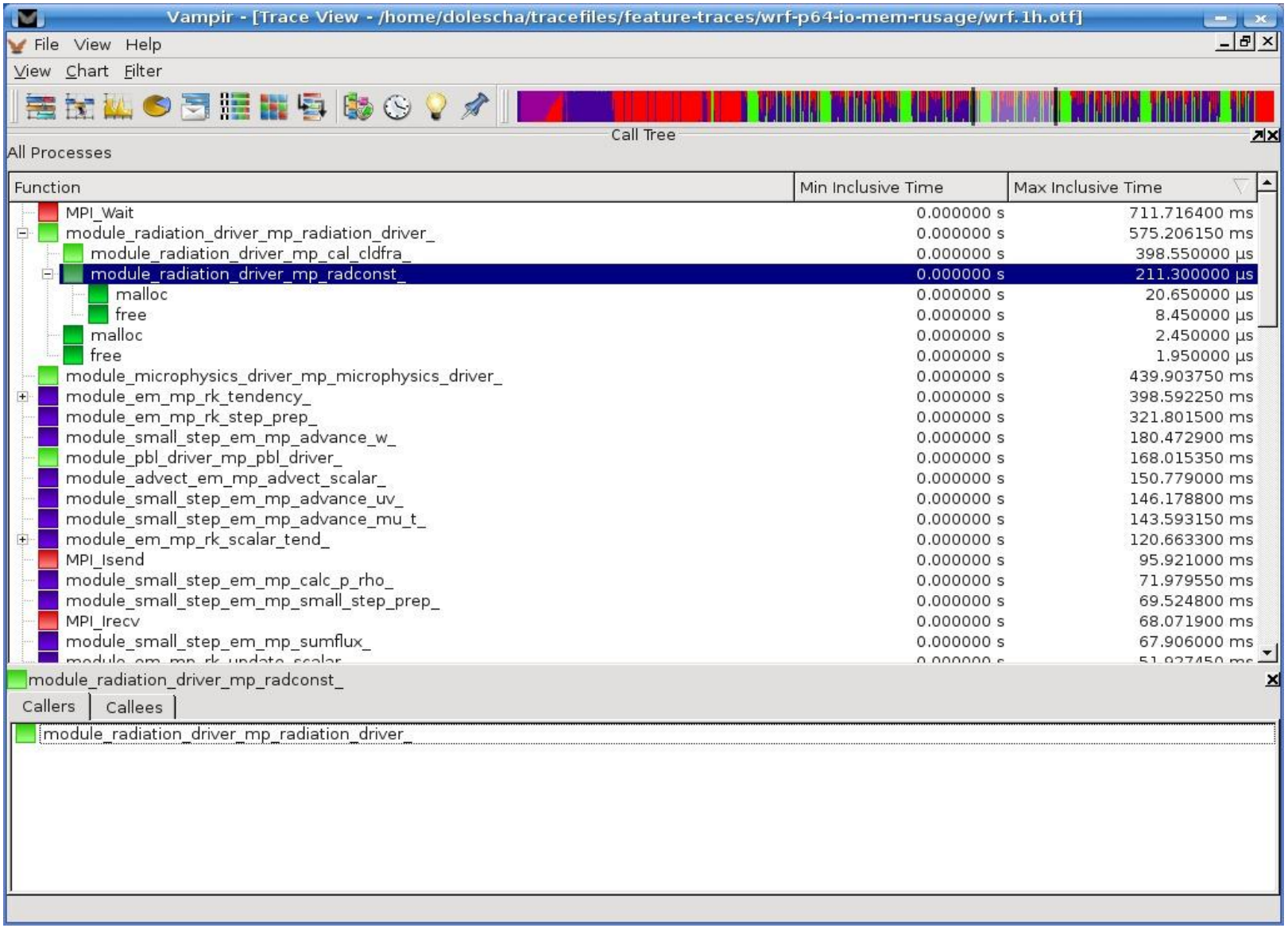
Process Summary



mmunication Matrix



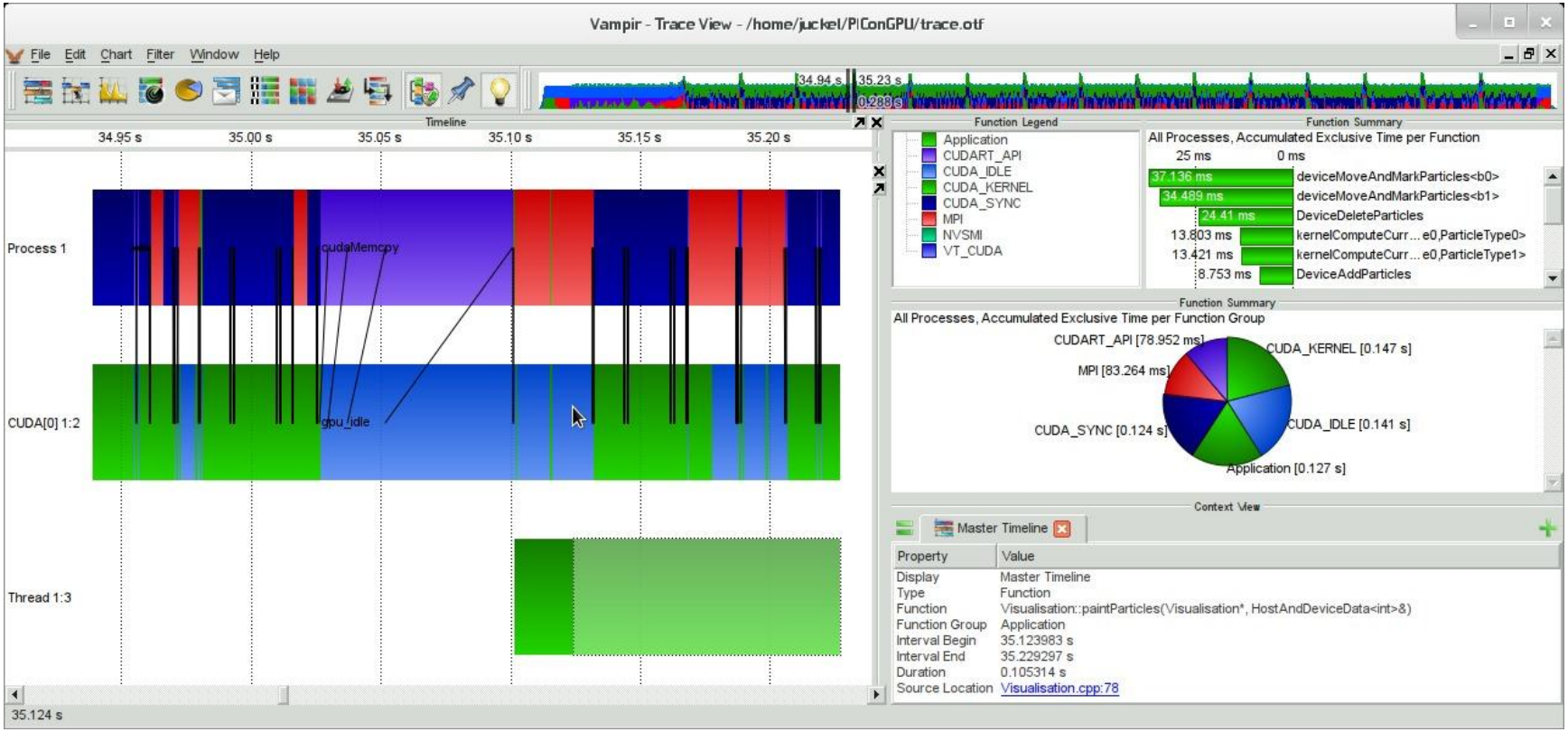
l Tree


Vampir - [Trace View - /home/dolescha/tracefiles/feature-traces/wrf-p64-io-mem-rusage/wrf.1h.otf]
File View Help
View Chart Filter
Call Tree
All Processes
Function
Min Inclusive Time
Max Inclusive Time
MPI_Wait 0.000000 s 711.716400 ms
module_radiation_driver_mp_radiation_driver_ 0.000000 s 575.206150 ms
module_radiation_driver_mp_cal_cldfra_ 0.000000 s 398.550000 µs
module_radiation_driver_mp_radconst_ 0.000000 s 211.300000 µs
malloc 0.000000 s 20.650000 µs
free 0.000000 s 8.450000 µs
malloc 0.000000 s 2.450000 µs
free 0.000000 s 1.950000 µs
module_microphysics_driver_mp_microphysics_driver_ 0.000000 s 439.903750 ms
module_em_mp_rk_tendency_ 0.000000 s 398.592250 ms
module_em_mp_rk_step_prep_ 0.000000 s 321.801500 ms
module_small_step_em_mp_advance_w_ 0.000000 s 180.472900 ms
module_pbl_driver_mp_pbl_driver_ 0.000000 s 168.015350 ms
module_advect_em_mp_advect_scalar_ 0.000000 s 150.779000 ms
module_small_step_em_mp_advance_uv_ 0.000000 s 146.178800 ms
module_small_step_em_mp_advance_mu_t_ 0.000000 s 143.593150 ms
module_em_mp_rk_scalar_tend_ 0.000000 s 120.663300 ms
MPI_Isend 0.000000 s 95.921000 ms
module_small_step_em_mp_calc_p_rho_ 0.000000 s 71.979550 ms
module_small_step_em_mp_small_step_prep_ 0.000000 s 69.524800 ms
MPI_Irecv 0.000000 s 68.071900 ms
module_small_step_em_mp_sumflux_ 0.000000 s 67.906000 ms
module_radiation_driver_mp_radconst_
Callers
Callees
module_radiation_driver_mp_radiation_driver_

Overview

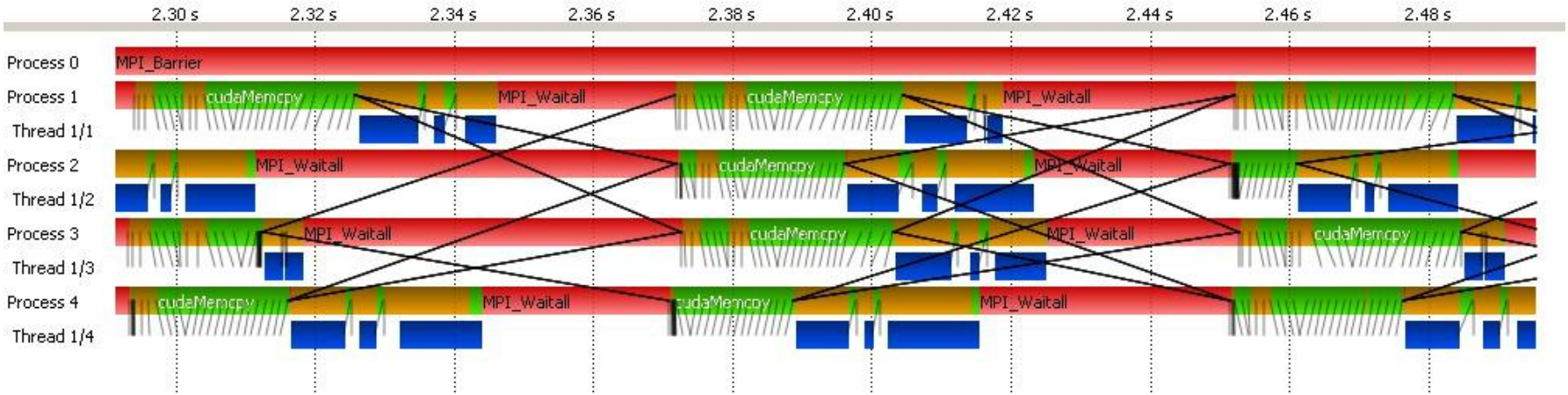
- Introduction
- Vampir Displays
- Advanced topics
  - GPU Support
  - Scalability
- Finding Performance Bottlenecks
- Conclusion & Outlook

GPU Tracing (CUDA)



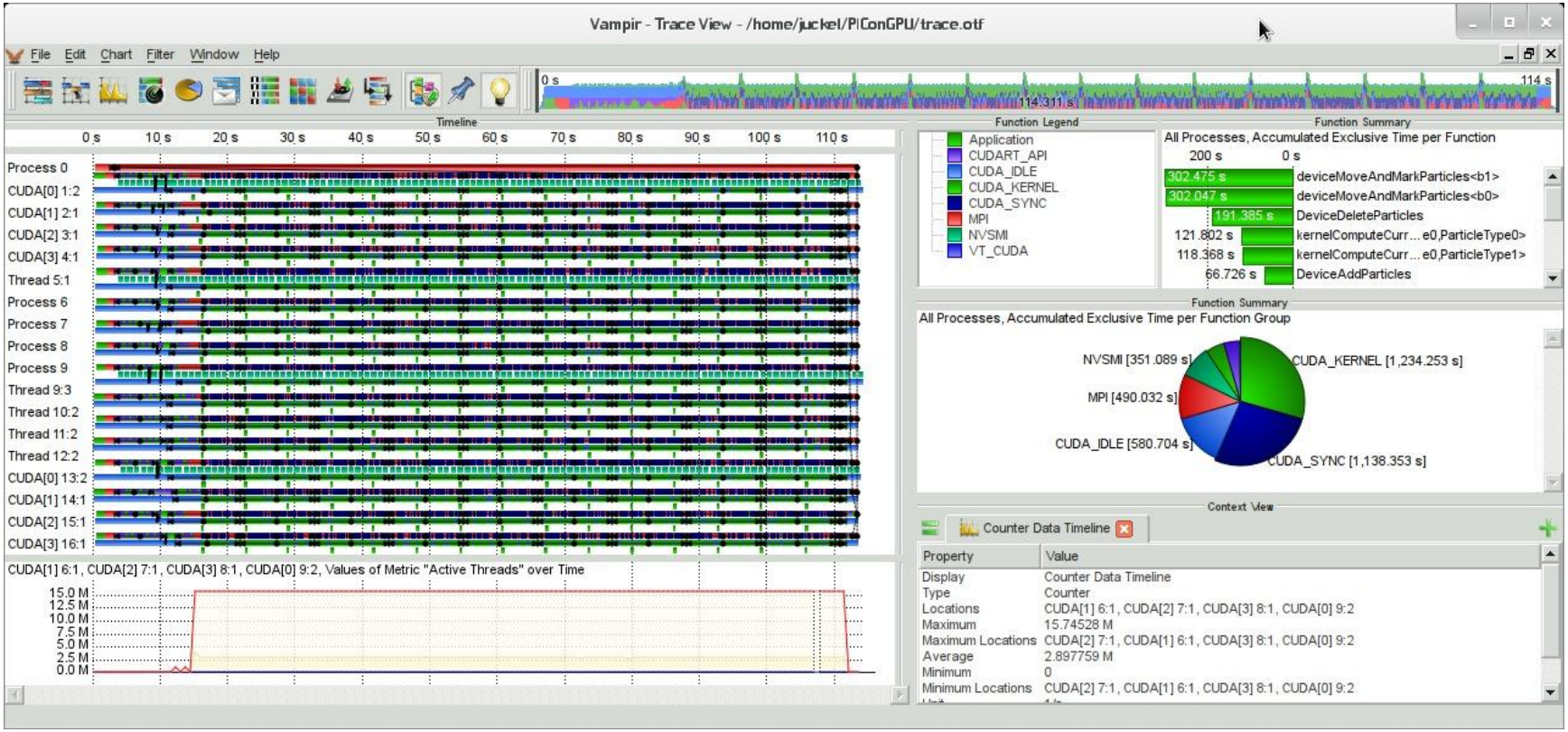
- Full support for GPGPU computing, CUDA and OpenCL
- Shows Kernel invocations, idle time, and DMA transfers

MPI + CUDA Tracing



- Support for hybrid modes: GPGPU + MPI + threads
  - Function invocations in host processes (Process X) and threads
  - Kernel invocations in CUDA threads (Thread x/y)
  - Host-GPU interactions via CUDA API (light arrows)
  - Host-Host interactions via MPI (bold arrows)

MPI + CUDA + Pthreads



- Details about kernel invocations
- Derived counter values, e.g. about GPU execution rate

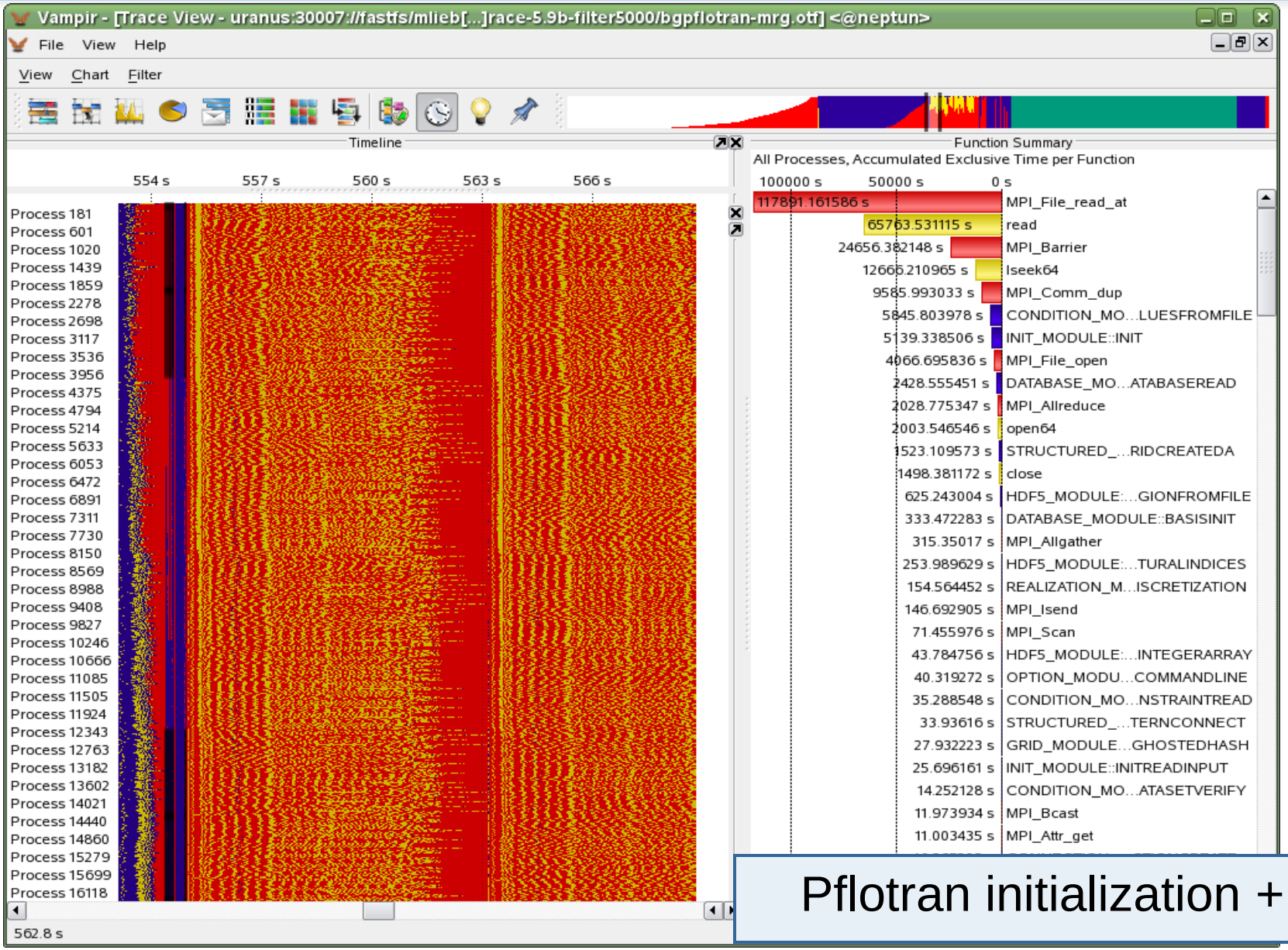
Vampir Scalability

- Currently no scalability limits provided enough analysis processes and enough distributed memory
- Full-scale ORNL Jaguar trace opened with 20,000 VampirServer analysis processes
- Vampir → find ways to show that much data

Scalability Features in Vampir GUI

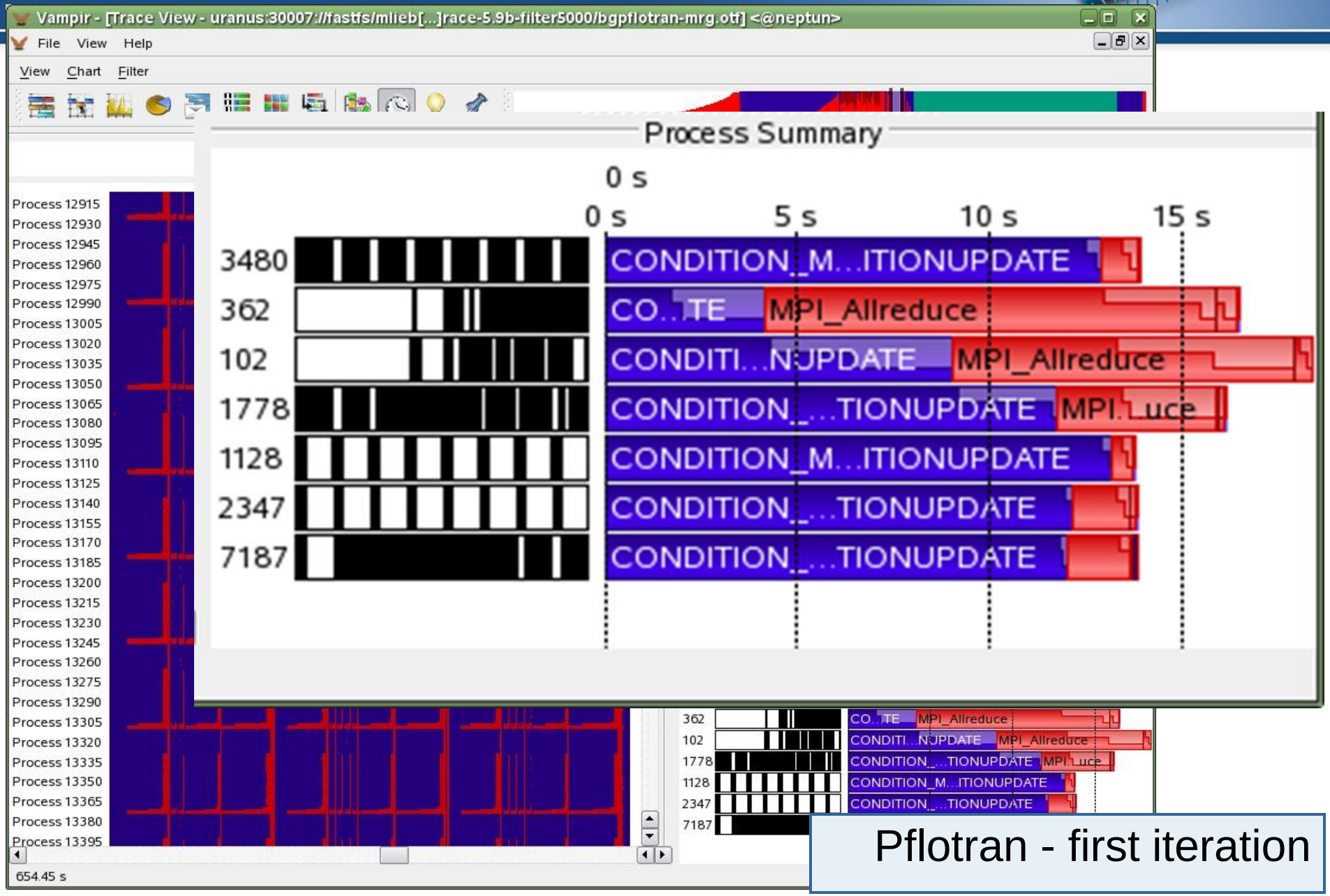
- Compact timelines (Fit to chart height)
  - In master timeline and performance radar
  - Visualization of more processes than vertical pixels
- Clustering
  - Allows detection of groups with similar behavior and outliere
- Performance radar
  - Highlighting performance conditions in a global timeline

Scalability Feature I: Fit to chart height



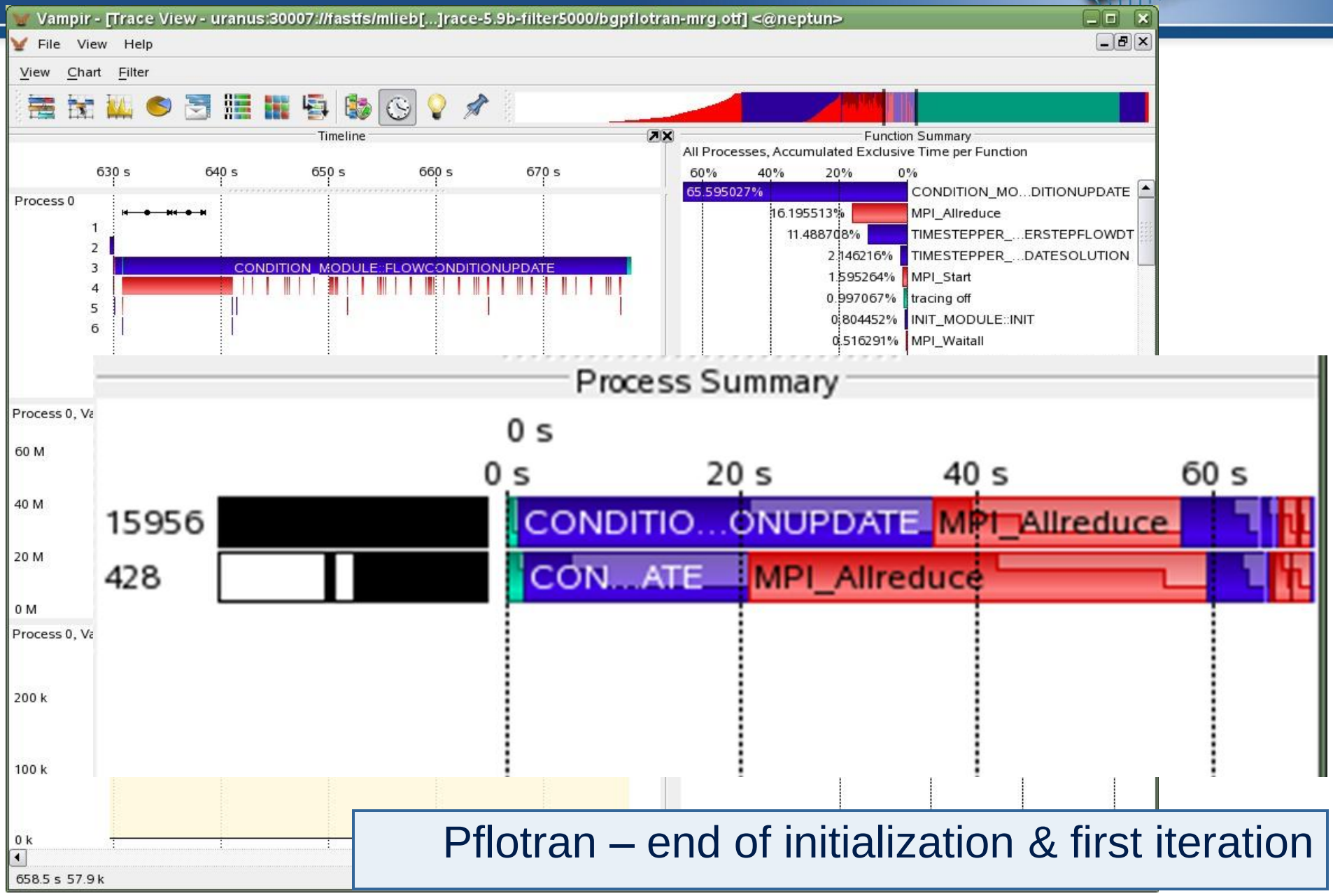
Pflotran initialization + I/O

Scalability feature II: Clustering



Pflotran - first iteration

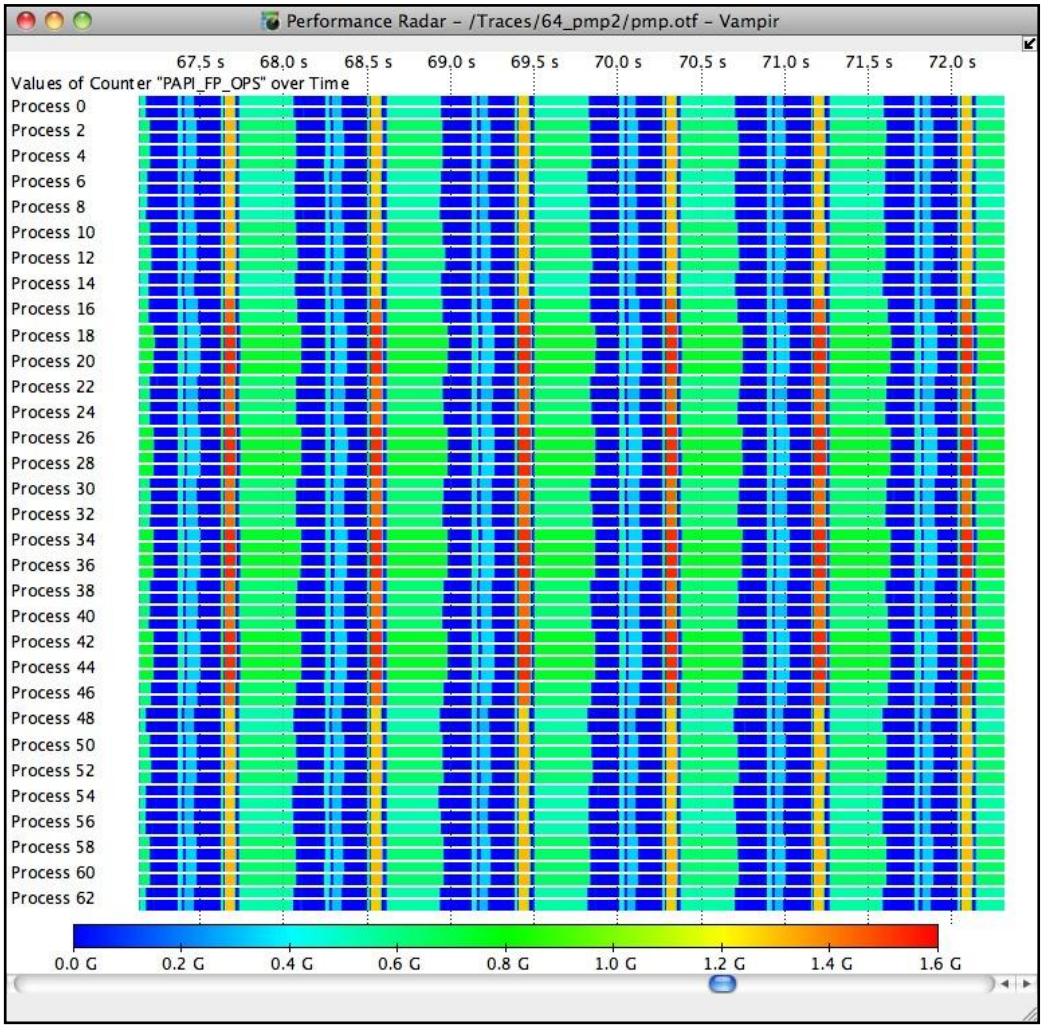
Scalability feature II: Clustering



Pflotran – end of initialization & first iteration

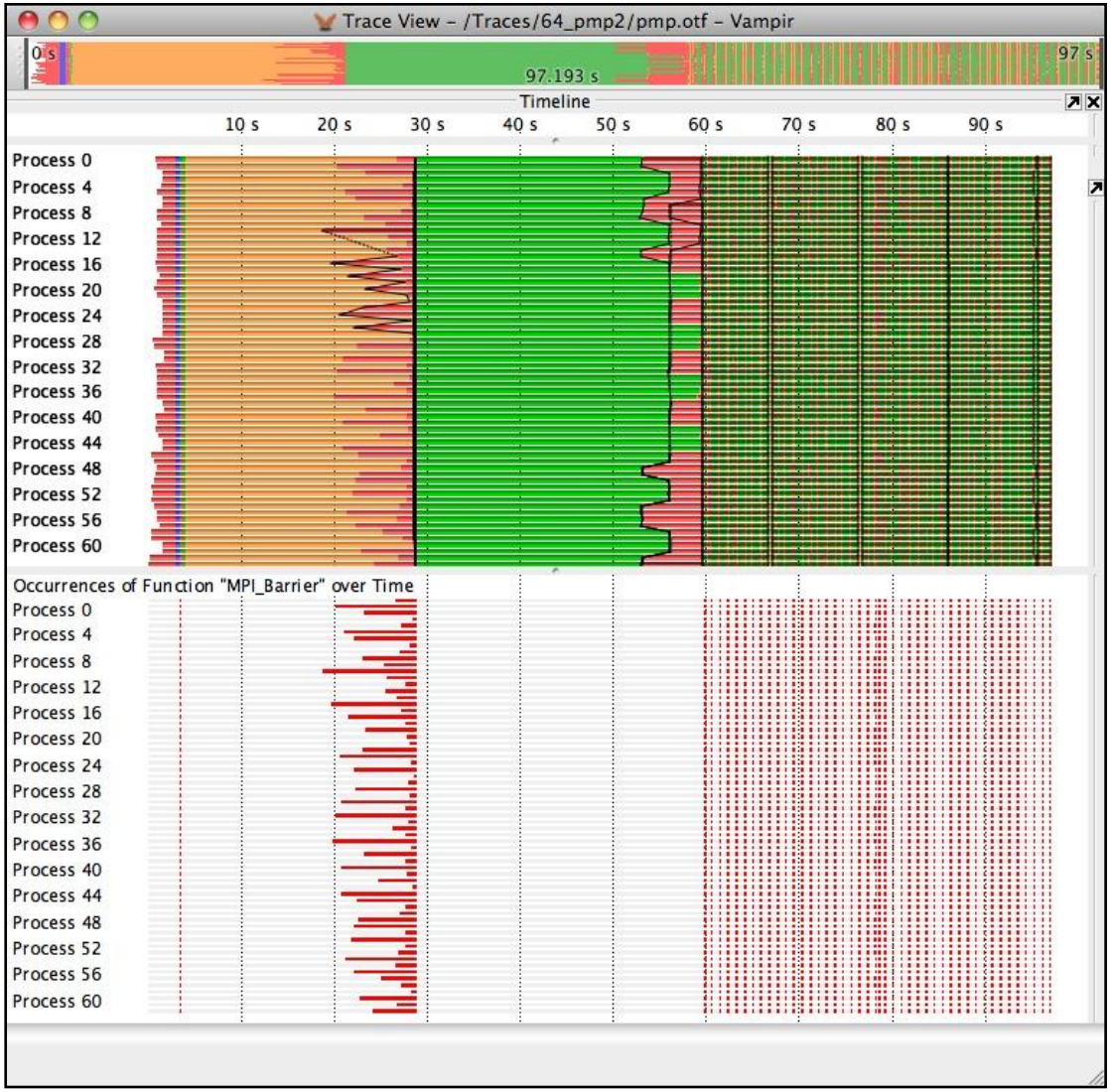
Scalability Feature III: Performance Radar

- Identification of relevant spots with a heat map
- Performance counters and properties derived from trace events
- Arithmetics on counter data and event properties



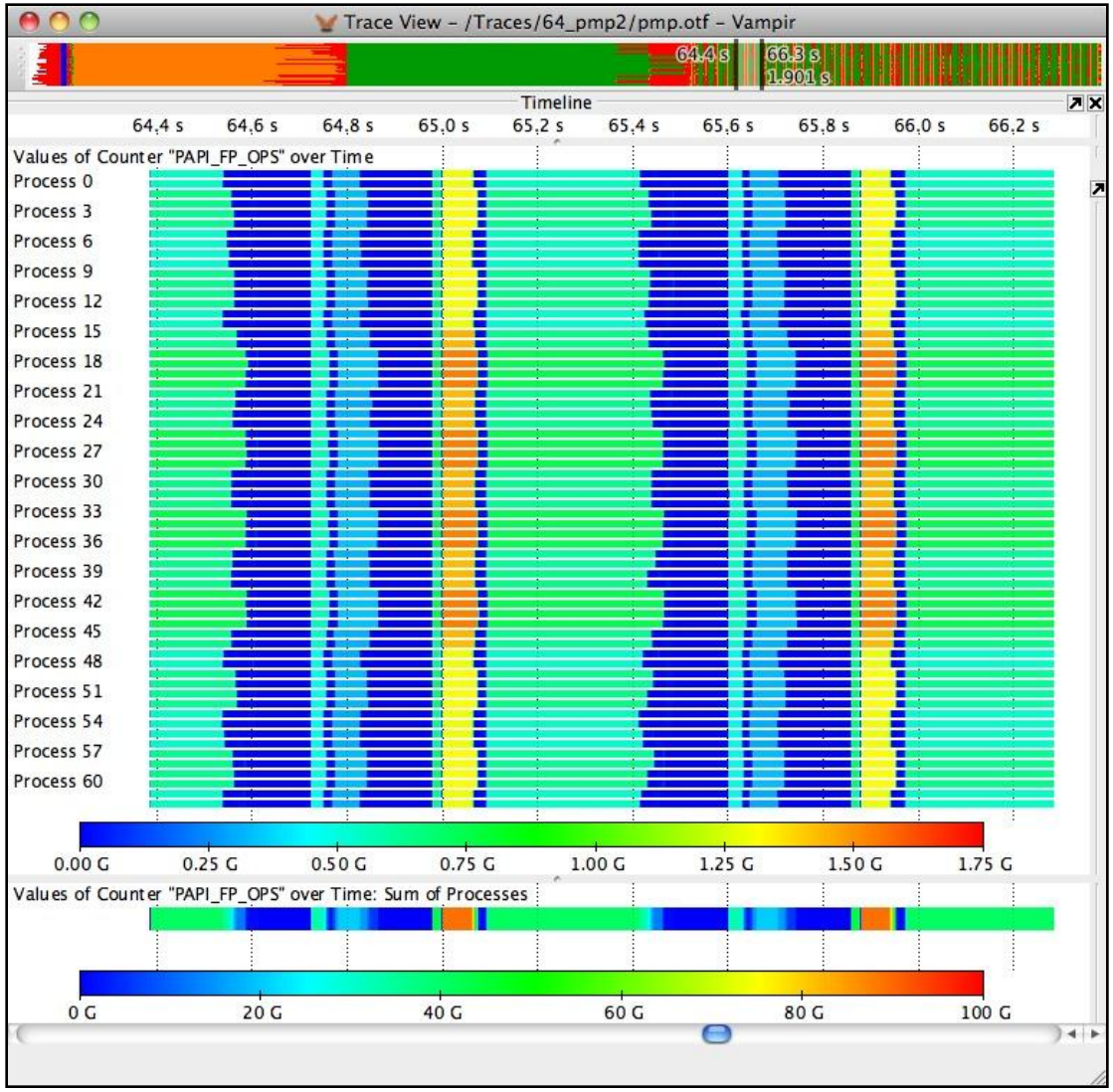
Scalability Feature III: Performance Radar

- Easily reveals where given functions occur, e.g. MPI_Barrier
- Shows density of function calls

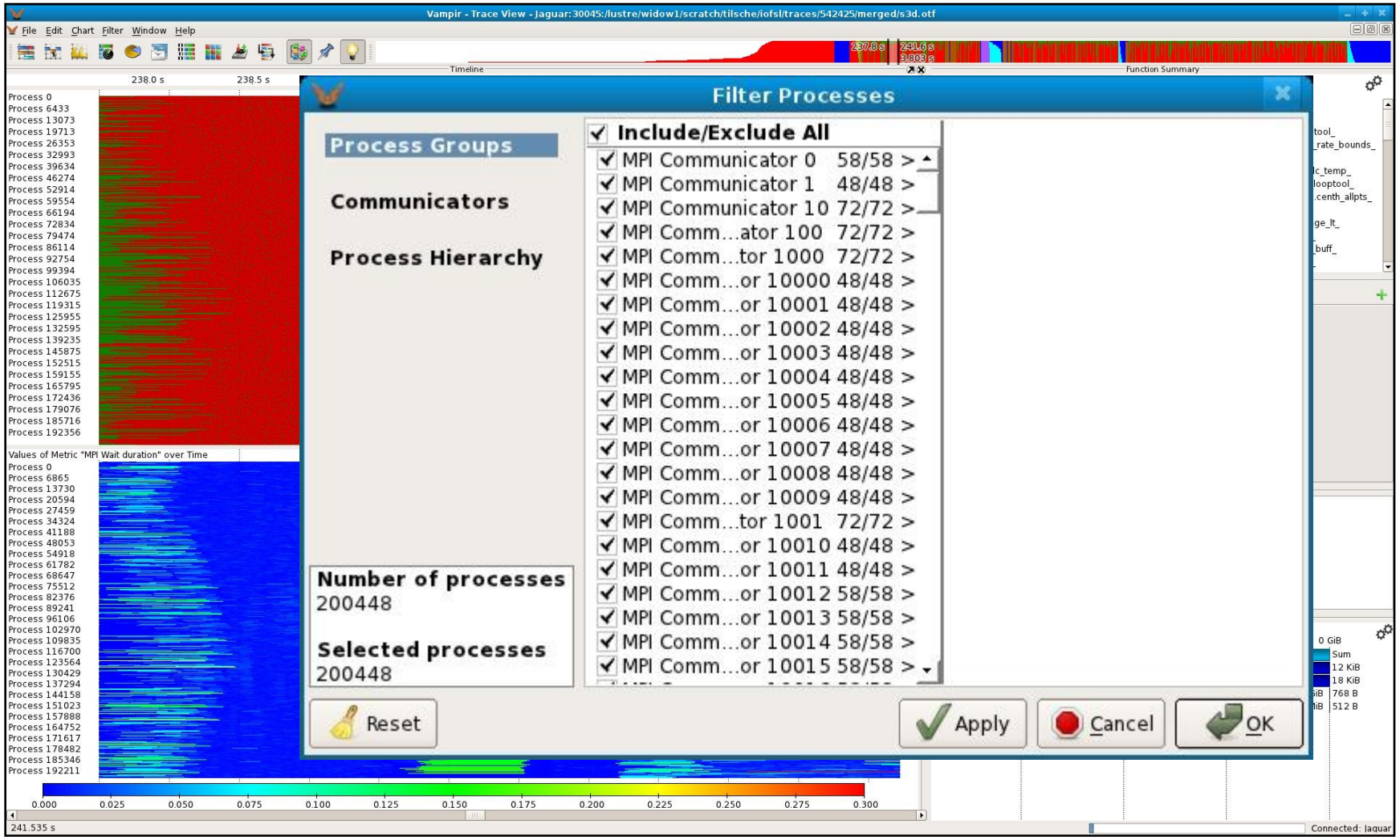


Scalability Feature III: Performance Radar

- Allows global sum over all processes, ranks and threads



Scalability Example: 200,000+ Processes in Vampir



Overview

- Introduction
- Vampir Displays
- Advanced topics
  - GPU Support
  - Scalability
- Finding Performance Bottlenecks
- Conclusion & Outlook

Vampir trace visualization

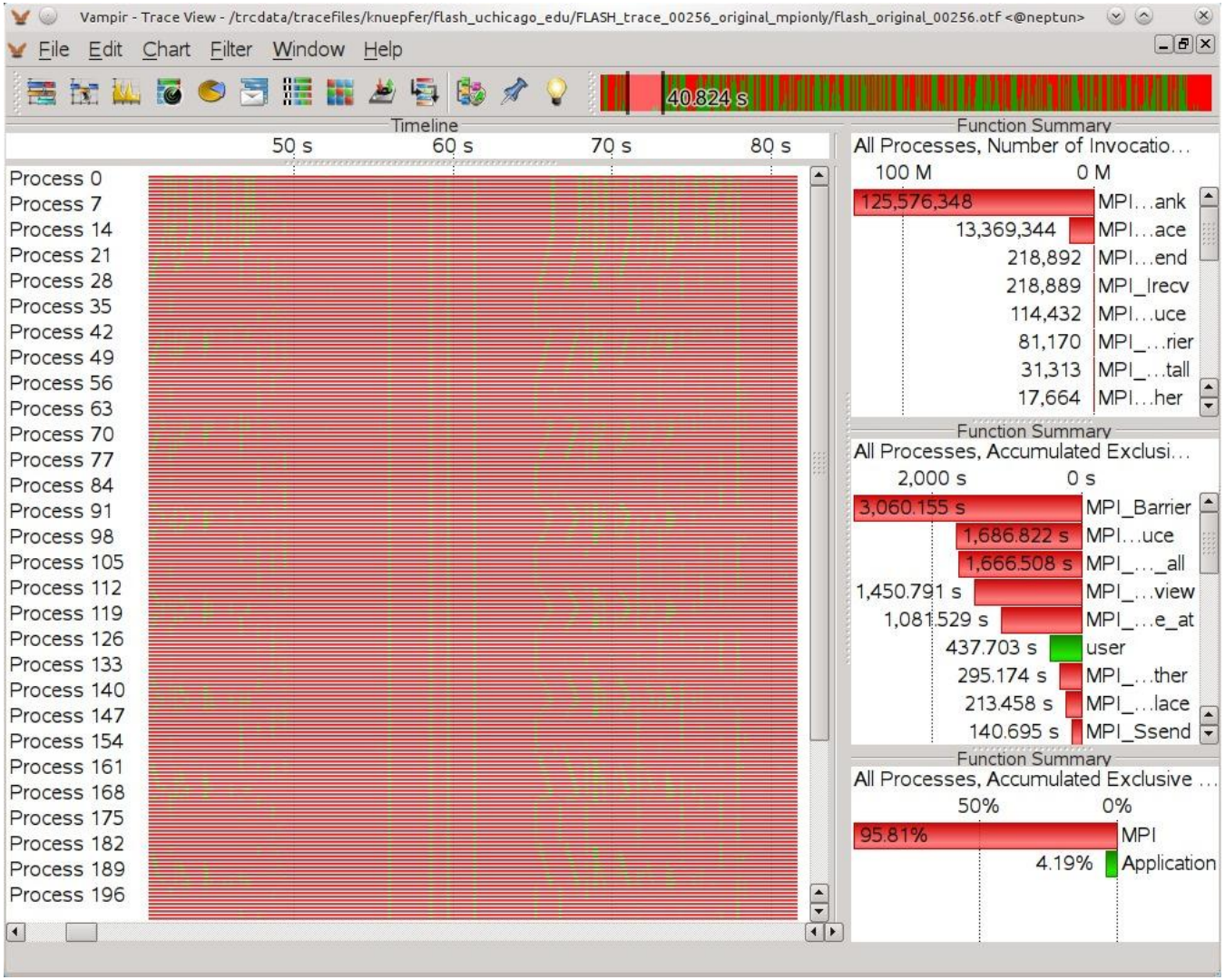
- several displays with many options
- identify essential parts of an application (initialization, main iteration, I/O, finalization)
- identify important components of the code (serial computation, MPI P2P, collective MPI, OpenMP)
- make a hypothesis about performance problems
- consider application's internal workings if known
- select the appropriate displays
- use statistic displays in conjunction with timelines

Bottlenecks in Communication

- Communication as such (dominating over computation)
- Late sender, late receiver
- Point-to-point messages instead of collective communication
- Unmatched messages
- Overcharge of MPI's buffers
- Bursts of large messages (bandwidth)
- Frequent short messages (latency)
- Unnecessary synchronization (barrier)
- The above usually result in a high MPI time share

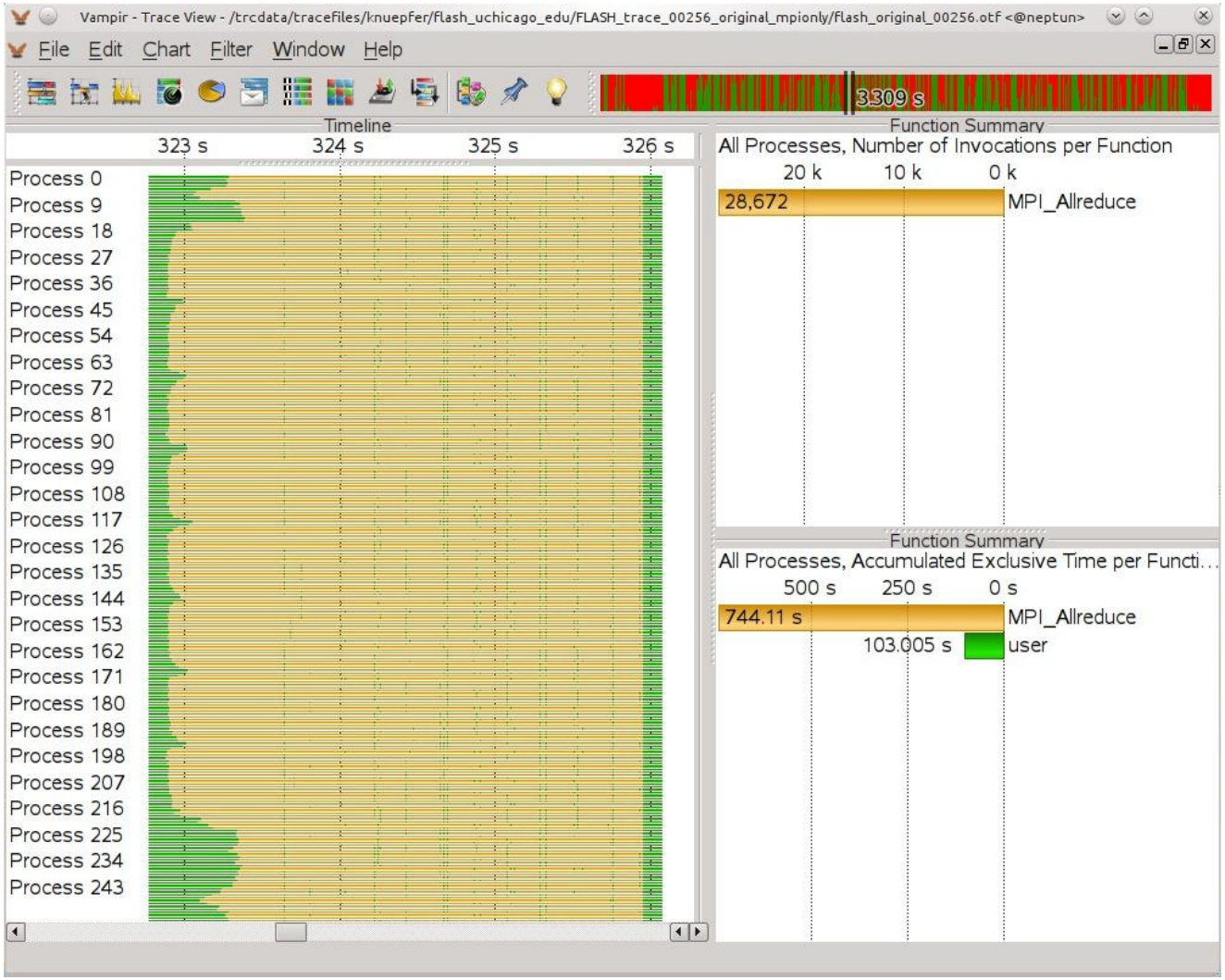
Bottlenecks in Communication: Example 1

Too much runtime share for MPI all-together



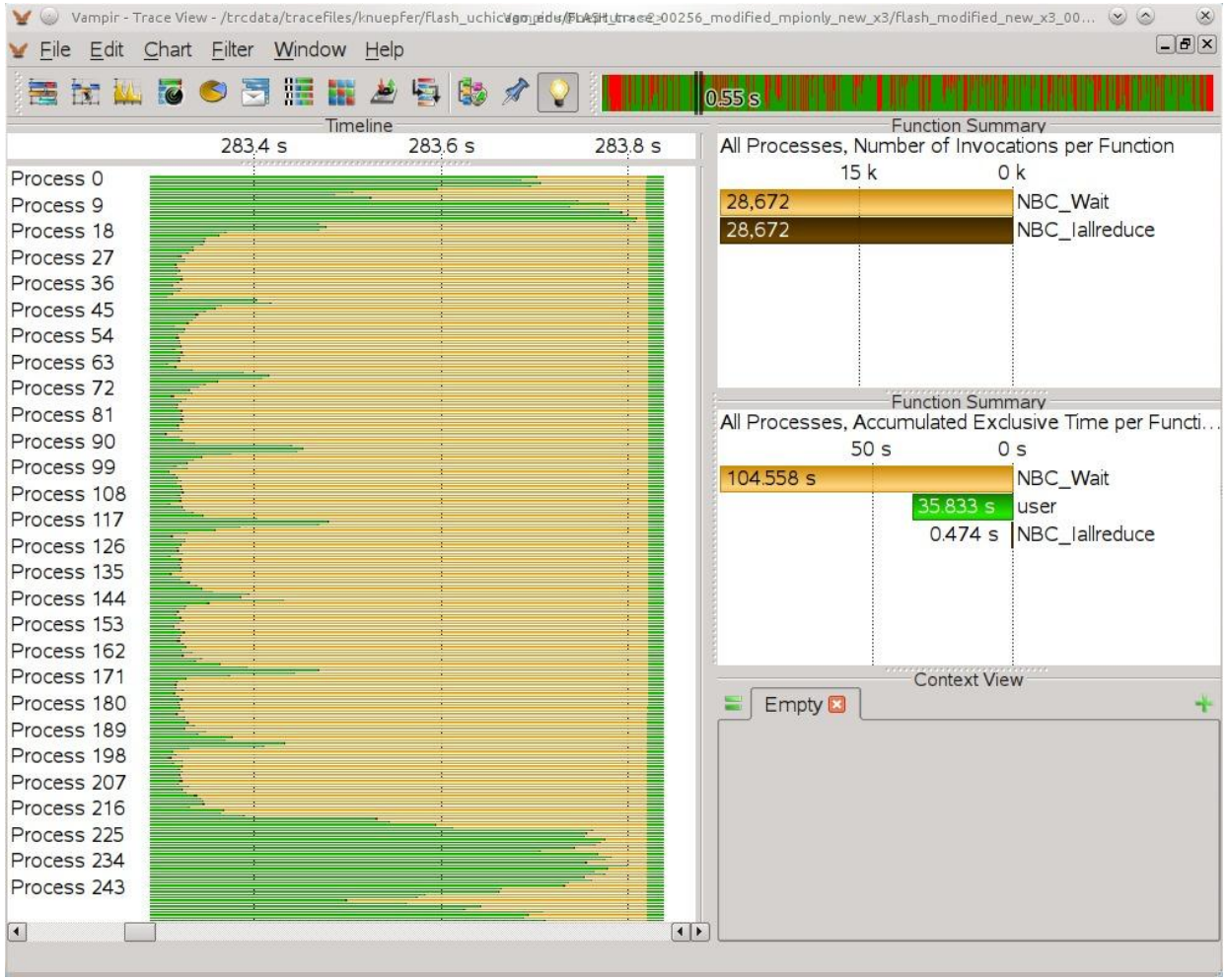
Bottlenecks in Communication: Example 2

Long bursts of MPI_AllReduce operations



Bottlenecks in Communication: Example 2

Long bursts of MPI_AllReduce operations
→
Replace by NBC non-blocking allreduce + wait



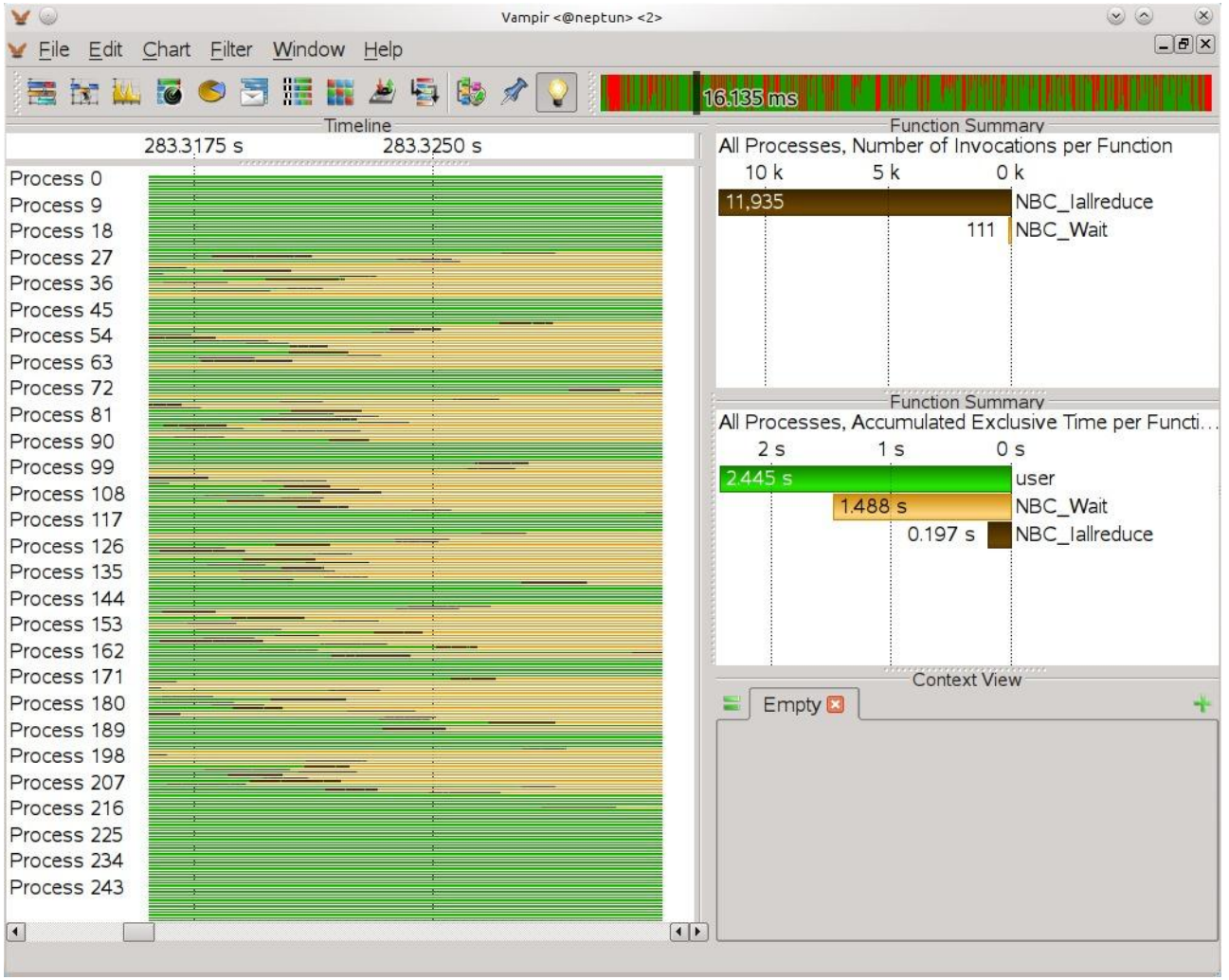
Bottlenecks in Communication : Example 2

Long bursts of MPI_AllReduce operations

→

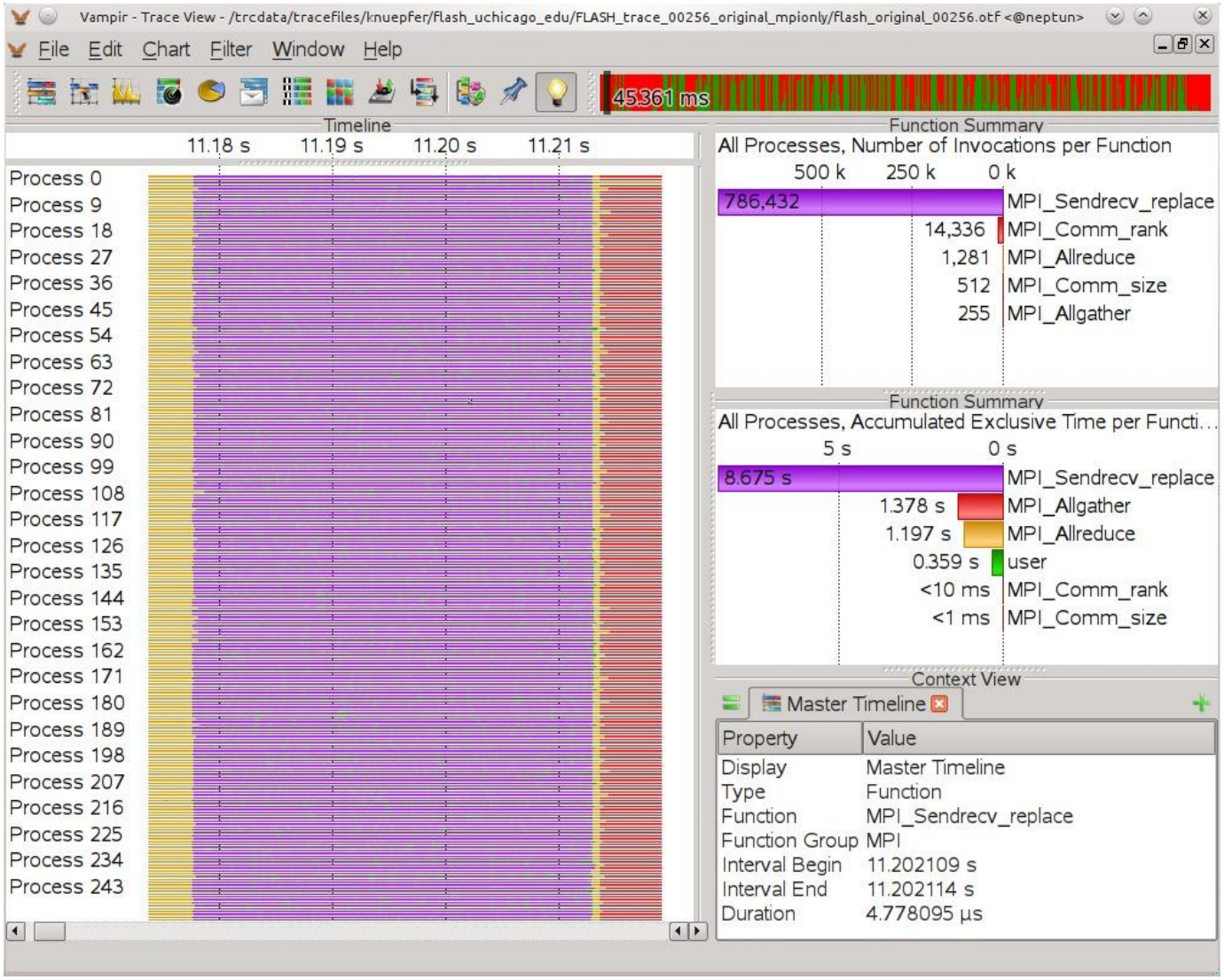
Replace by NBC non-blocking allreduce + wait

(zoom)



Bottlenecks in Communication: Example 3

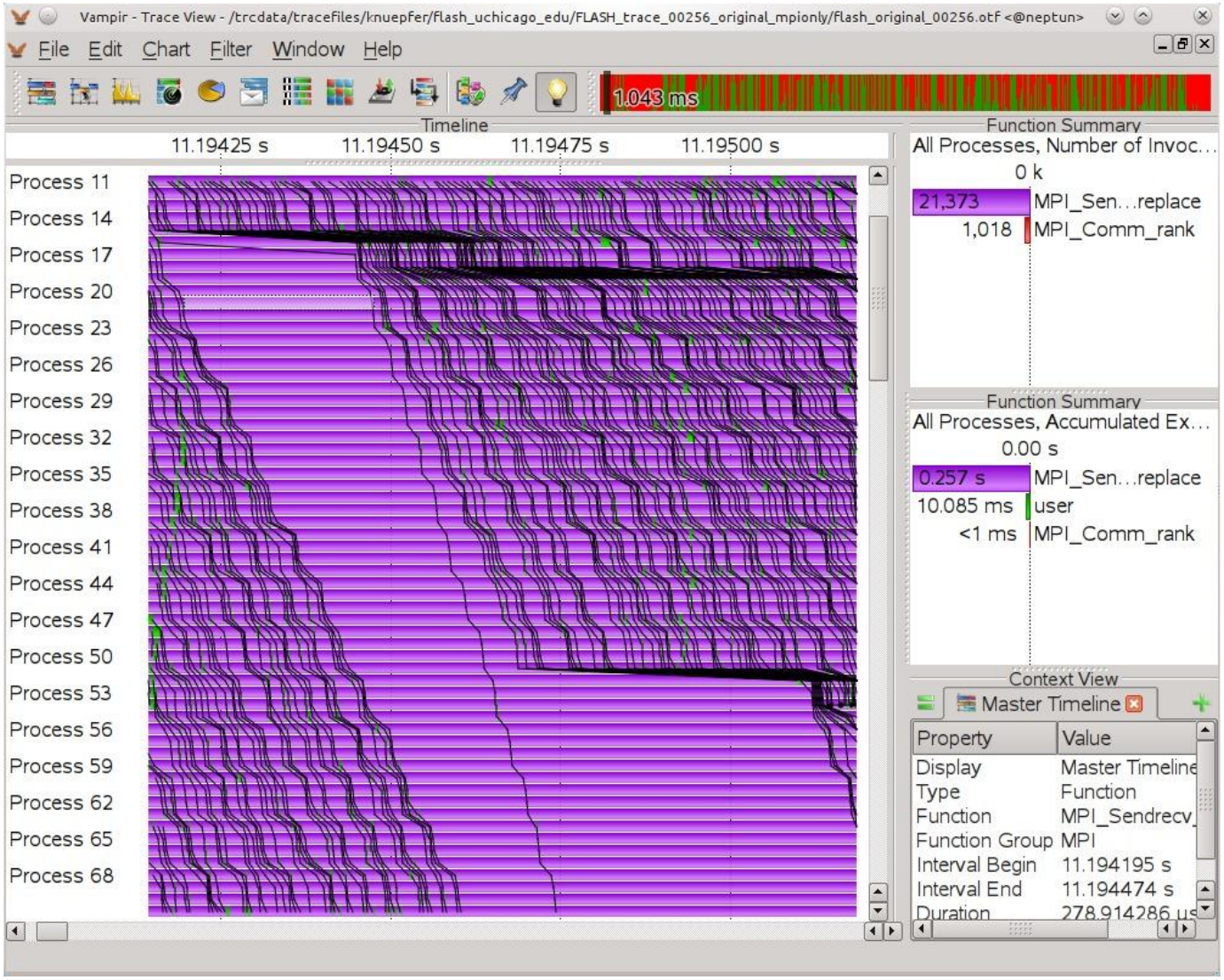
Chains of MPI Sendrecv replace



Bottlenecks in Communication: Example 3

Chains of MPI Sendrecv replace (zoom)

Propagate delays over successive MPI ranks

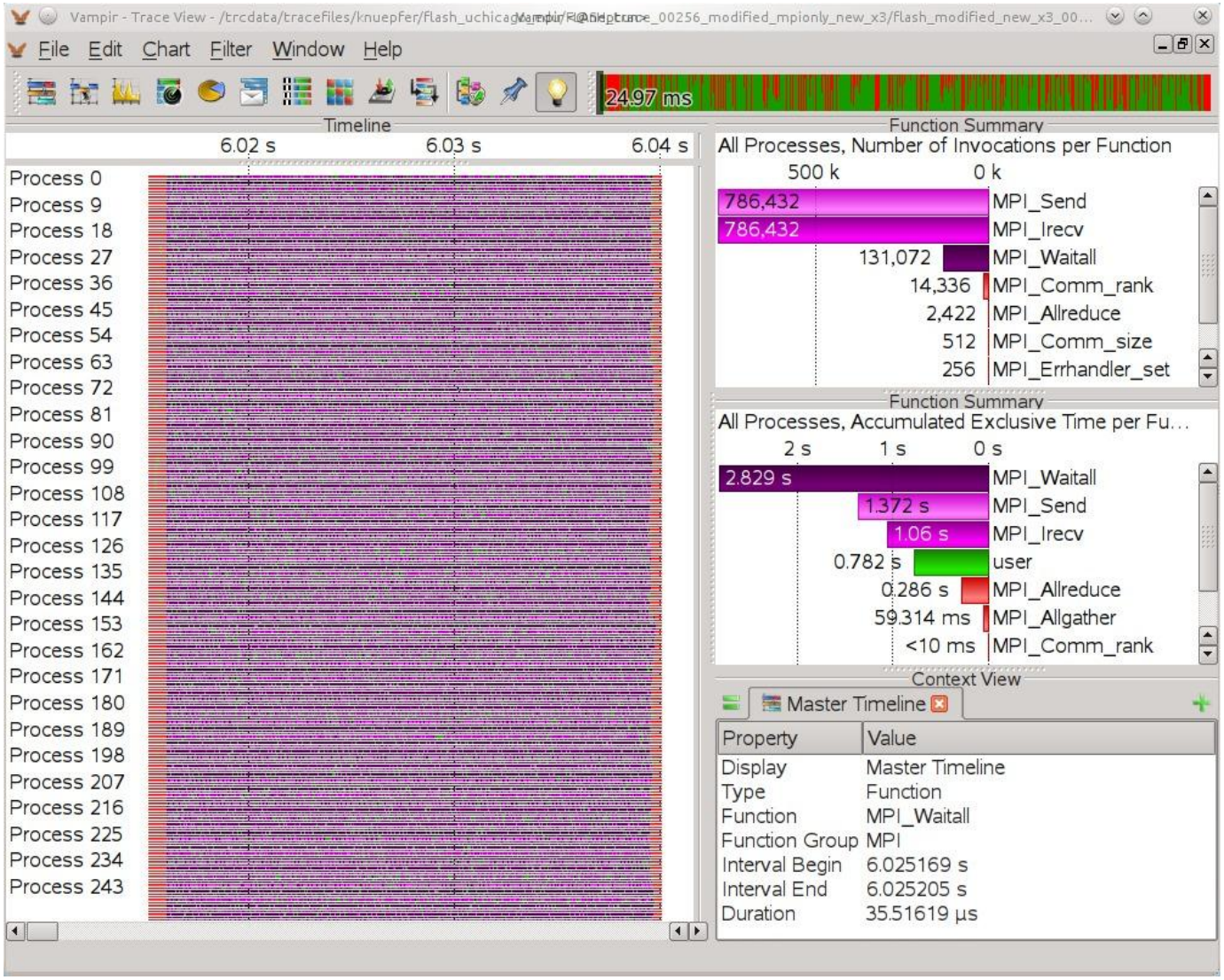


Bottlenecks in Communication: Example 3

Chains of MPI Sendrecv_ replace

→

Replace by faster over-lapping series of Send, Irecv, and Waitall

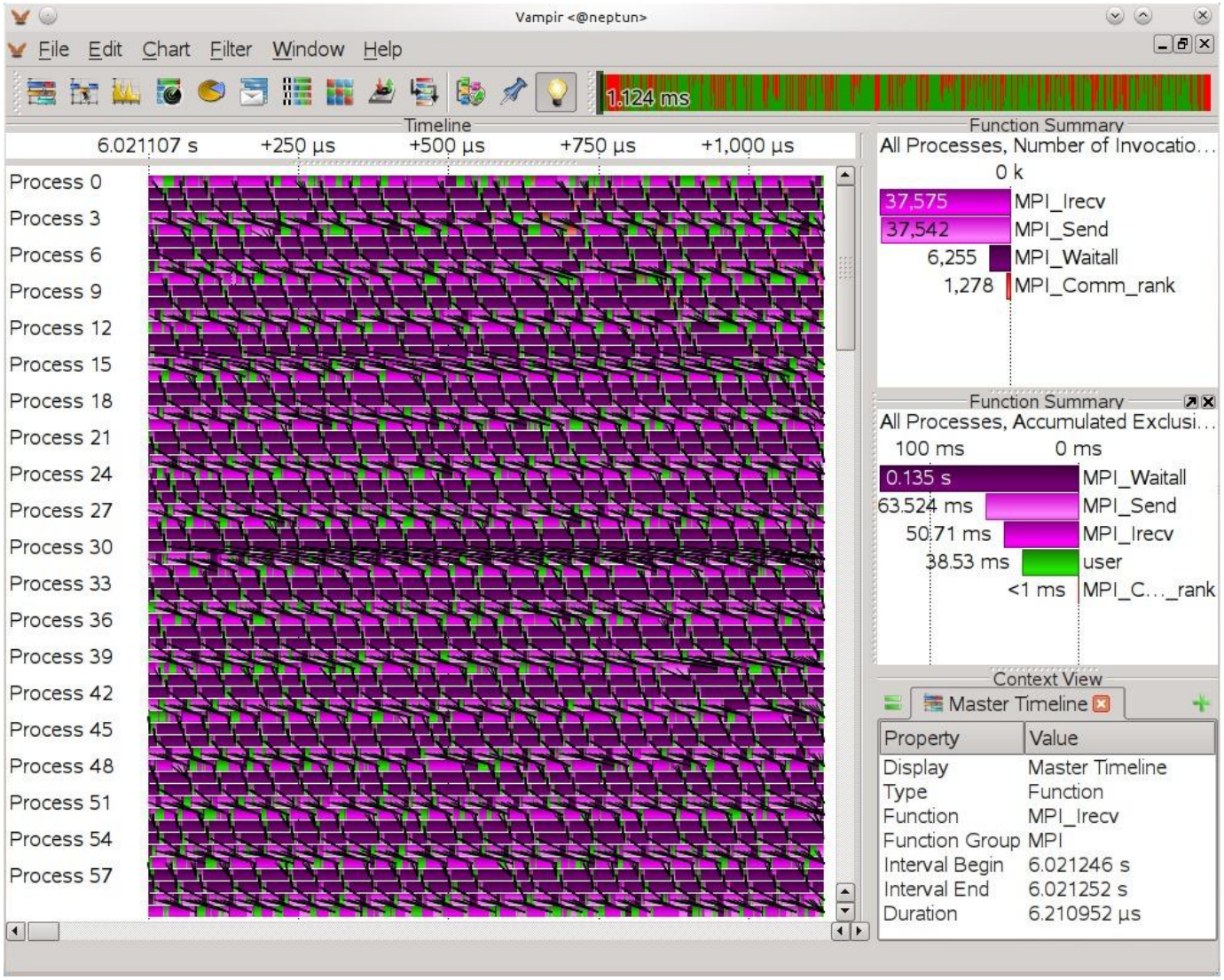


Bottlenecks in Communication: Example 3

Chains of MPI Sendrecv_ replace

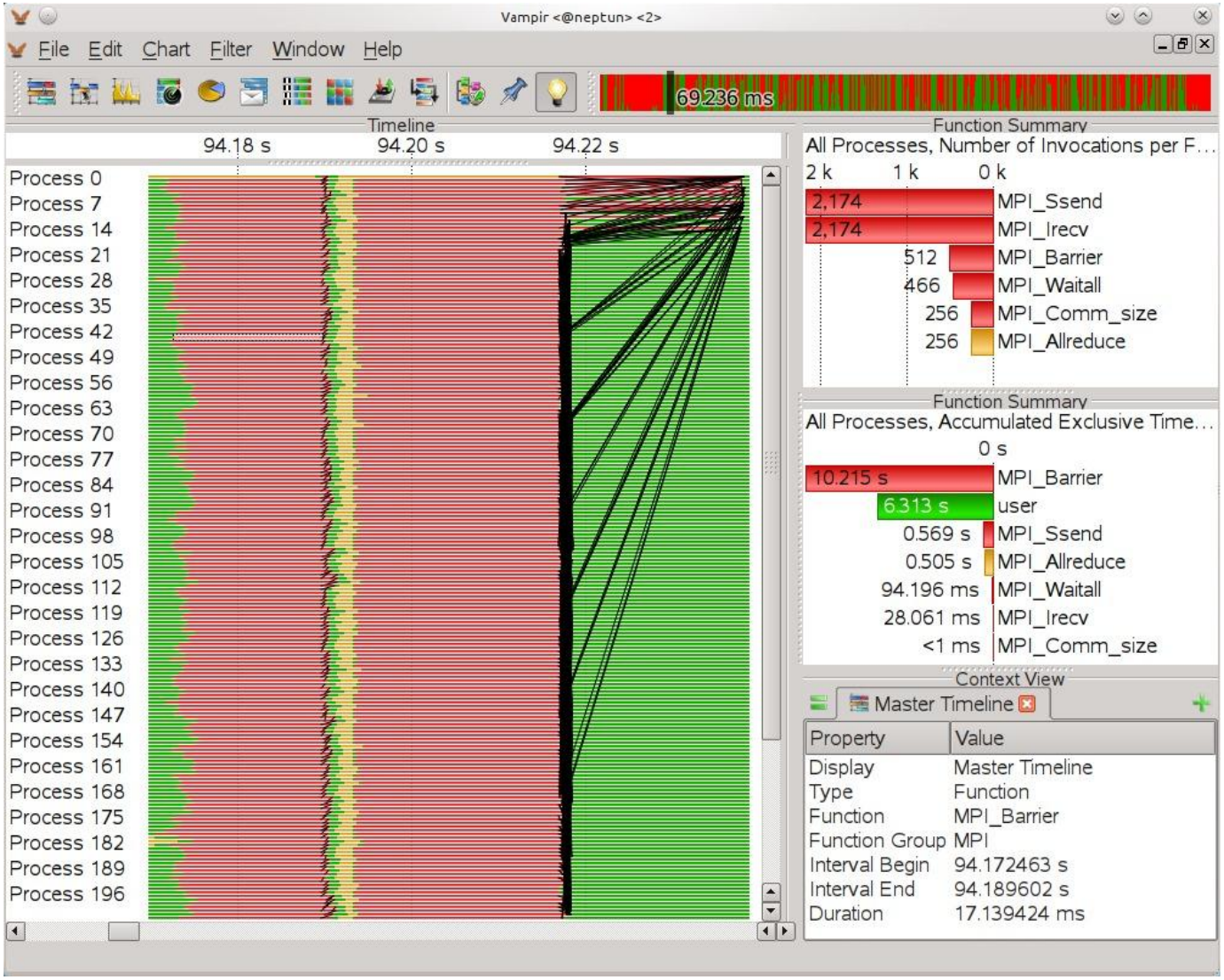
→

Replace by faster over-lapping series of Send, Irecv, and Waitall (zoom)



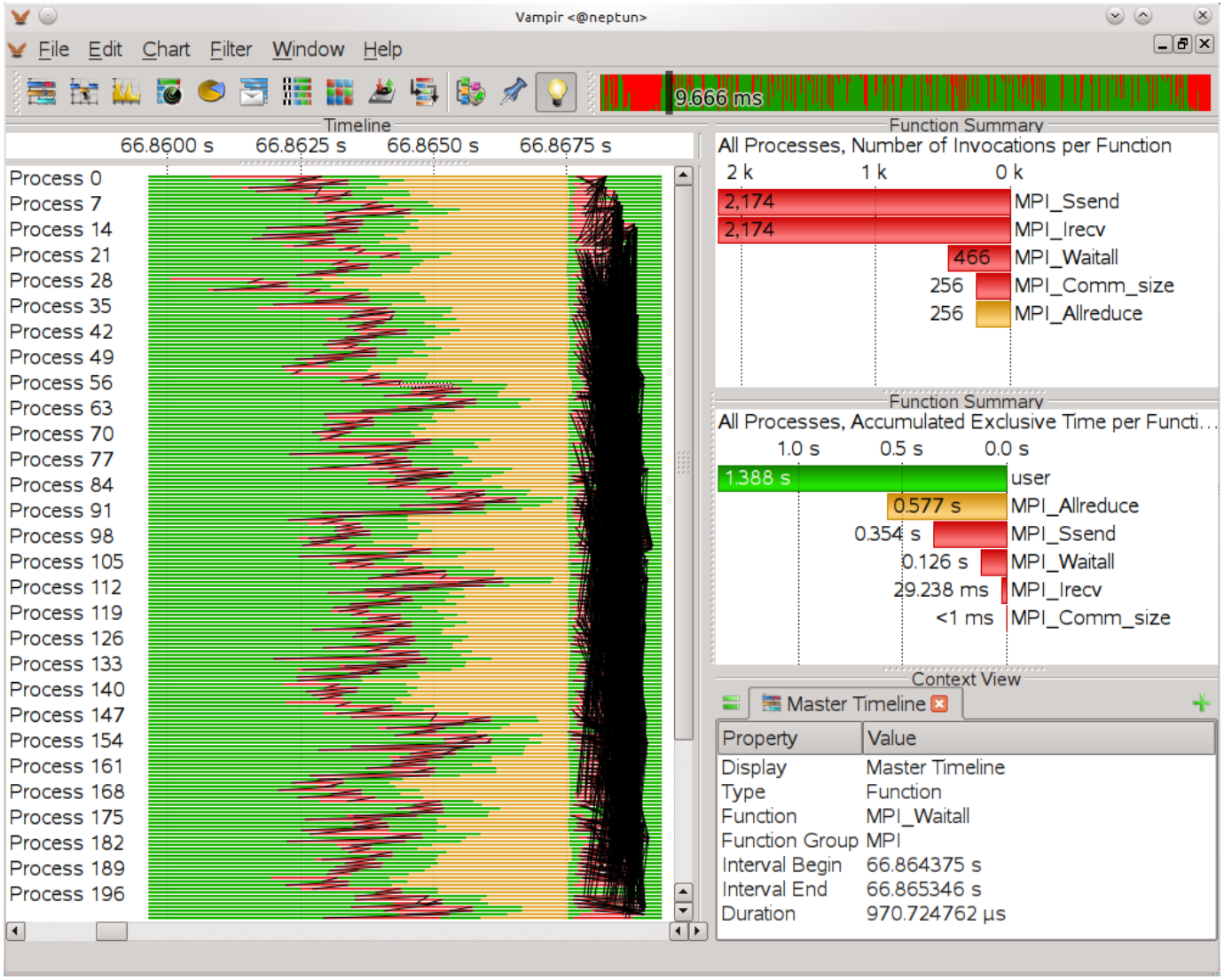
Bottlenecks in Communication: Example 4

Many MPI barriers are unneeded



Bottlenecks in Communication: Example 4

Many MPI barriers are unneeded
→
Remove barriers if possible, following operations start less regular but are faster.



Bottlenecks in Computation

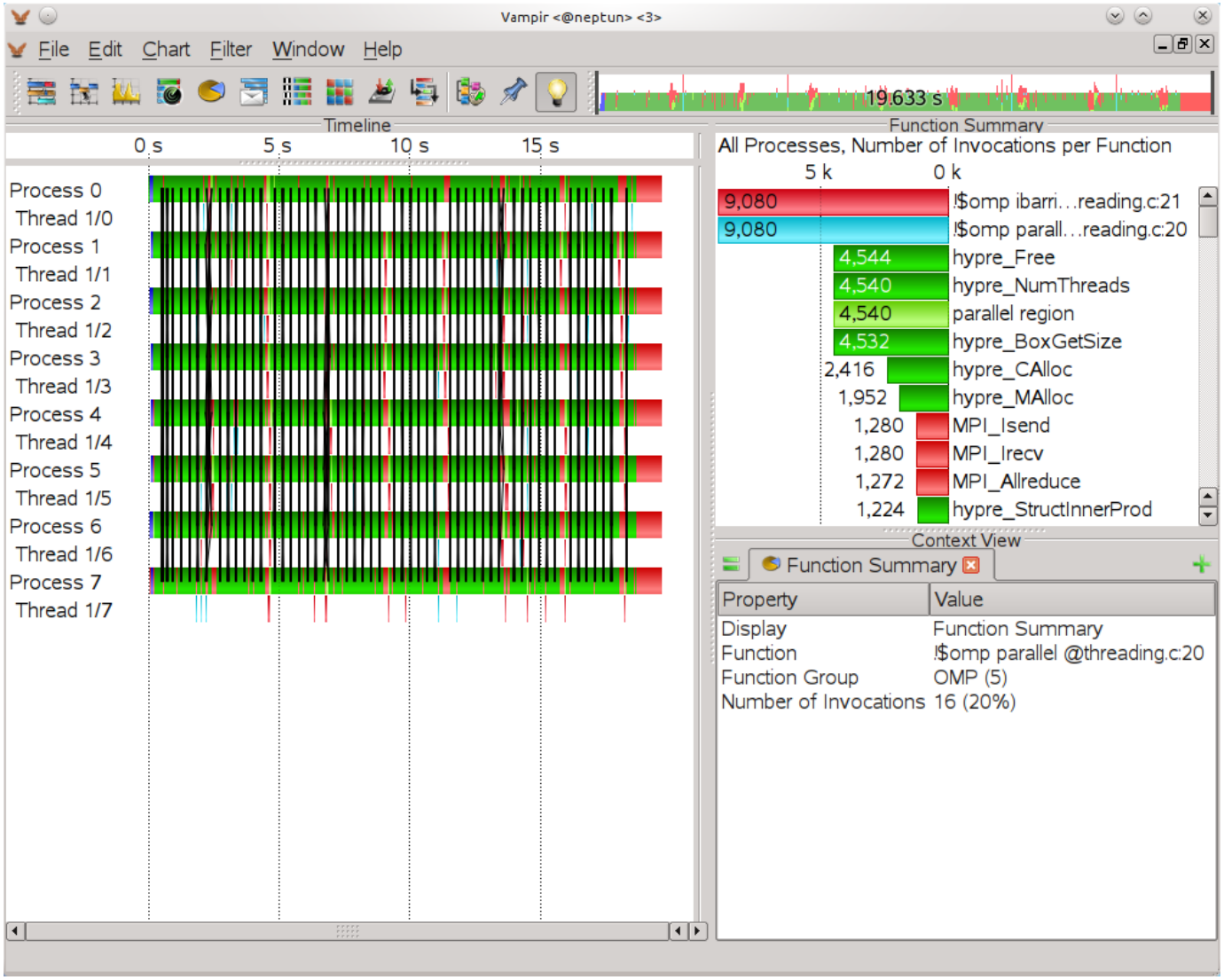
- Unbalanced computation (single late comer)
- Strictly serial parts of program (idle threads)
- Frequent tiny function calls, sparse loops
- Memory bound computation
  - Inefficient L1/L2/L3 cache usage, TLB misses
  - Detectable via HW performance counters
- I/O bound computation
  - Slow input/output, sequential I/O
  - I/O load imbalance
- Exception handling

Bottlenecks in Computation: Example 1

Mostly idle OpenMP threads

→

Low parallel efficiency



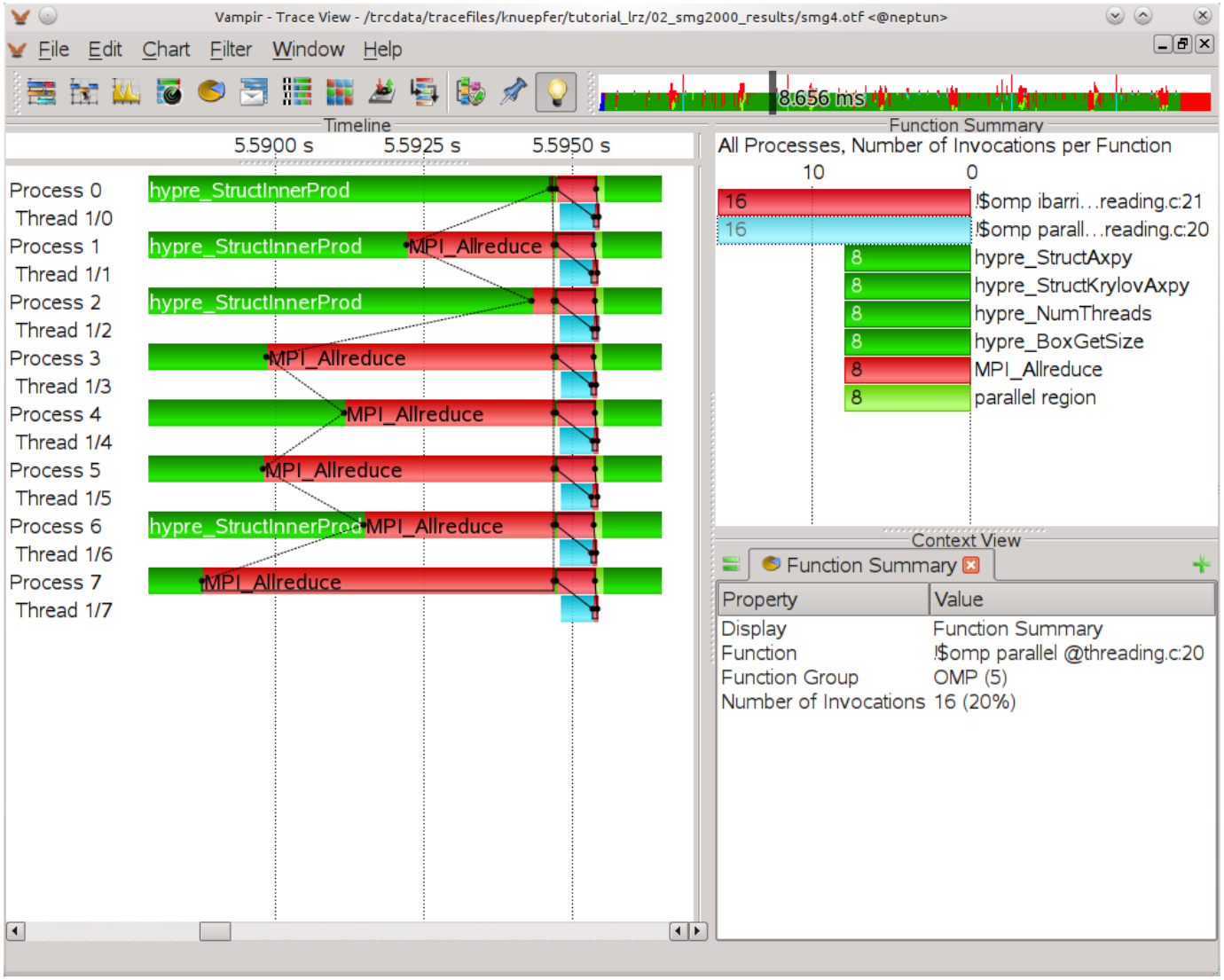
Bottlenecks in Computation: Example 1

Mostly idle OpenMP threads

→

Low parallel efficiency

(zoom)

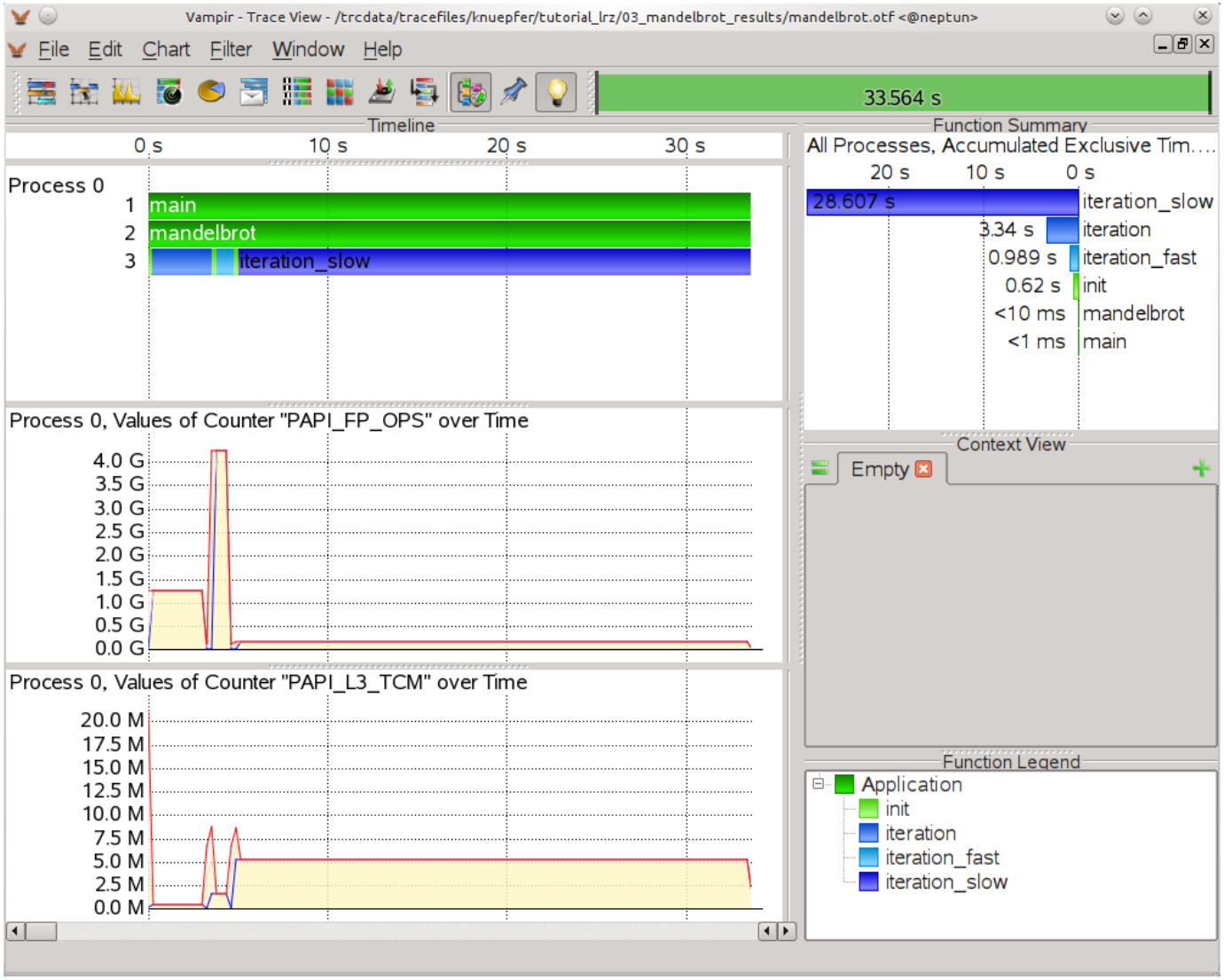


Bottlenecks in Computation: Example 2

High rate of FPop/s with low rate of L3 cache misses

vs.

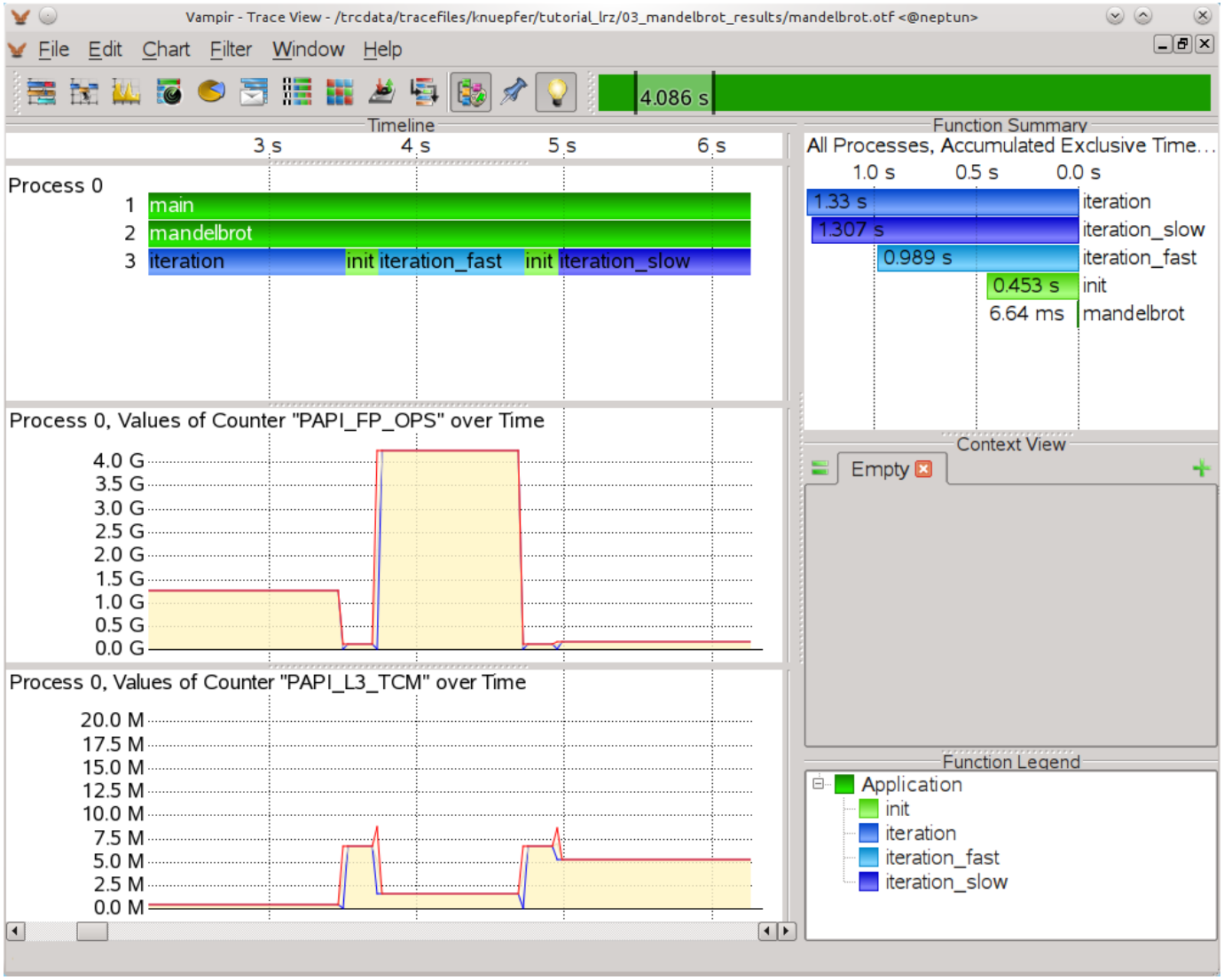
low FPop/s rate due to a high L3 miss rate.



Bottlenecks in Computation: Example 2

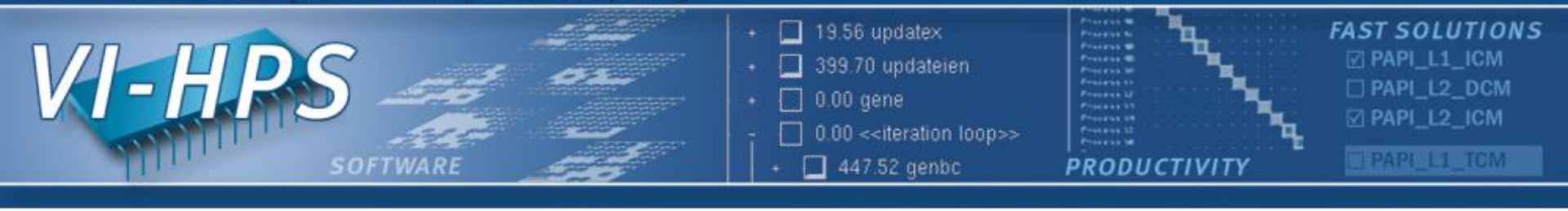
High rate of FPop/s with low rate of L3 cache misses
vs.
low FPop/s rate due to a high L3 miss rate.

(zoom)



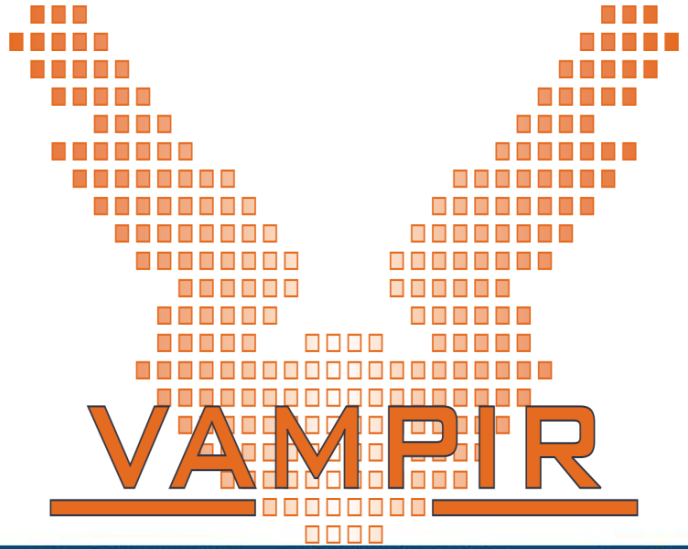
Overview

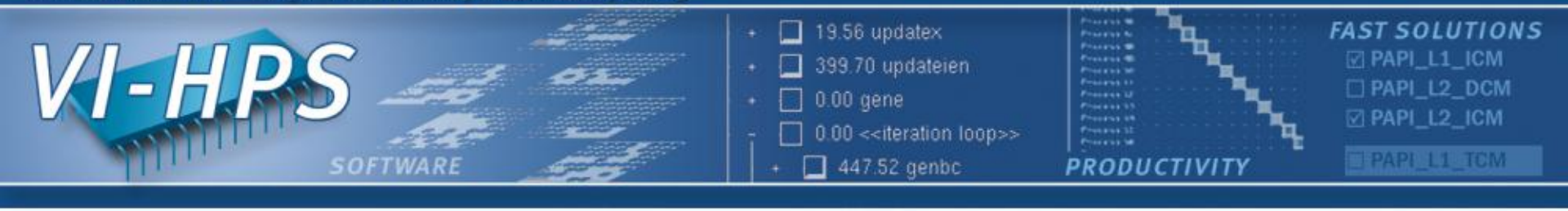
- Introduction
- Vampir Displays
- Advanced topics
  - GPU Support
  - Scalability
- Finding Performance Bottlenecks
- Conclusion & Outlook



Matthias Jurenz, Andreas Knüpfer, Ronny Brendel, Matthias Lieber, Jens Doleschal, Jens Domke, Holger Mickler, Daniel Hackenberg, Michael Heyde, Thomas Ilsche, Guido Juckeland, Robert Dietrich, Frank Winkler, Michael Kluge, Matthias Müller, Holger Brunst, Ronald Geisler, Reinhard Neumann, Bert Wesarg, Rene Widera, Thomas Ilsche, Matthias Weber, Thomas William, Hartmut Mix,

and Wolfgang E. Nagel





Vampir and VampirTrace are
available at http://www.vampir.eu and
http://www.tu-dresden.de/zih/vampirtrace/ ,
support via vampirsupport@zih.tu-dresden.de