Virtual Institute – High Productivity Supercomputing

VAMPIR & VAMPIRTRACE INTRODUCTION AND OVERVIEW

8th VI-HPS Tuning Workshop at RWTH Aachen
September, 2011

Tobias Hilbrich and Joachim Protze

Slides by: Andreas Knüpfer, Jens Doleschal,

ZIH, Technische Universität Dresden

Overview

Part I: Welcome to the Vampir Suite

- Introduction
- Event Trace Visualization
- Vampir & VampirServer
- The Vampir Displays
    - Timeline
    - Process Timeline with Performance Counters
    - Summary Display
    - Message Statistics
- VampirTrace
    - Instrumentation & Run-Time Measurement

Part II: Hands On

Why bother with performance analysis?

- Well, why are you here after all?
- Efficient usage of expensive and limited resources
- Scalability to achieve next bigger simulation

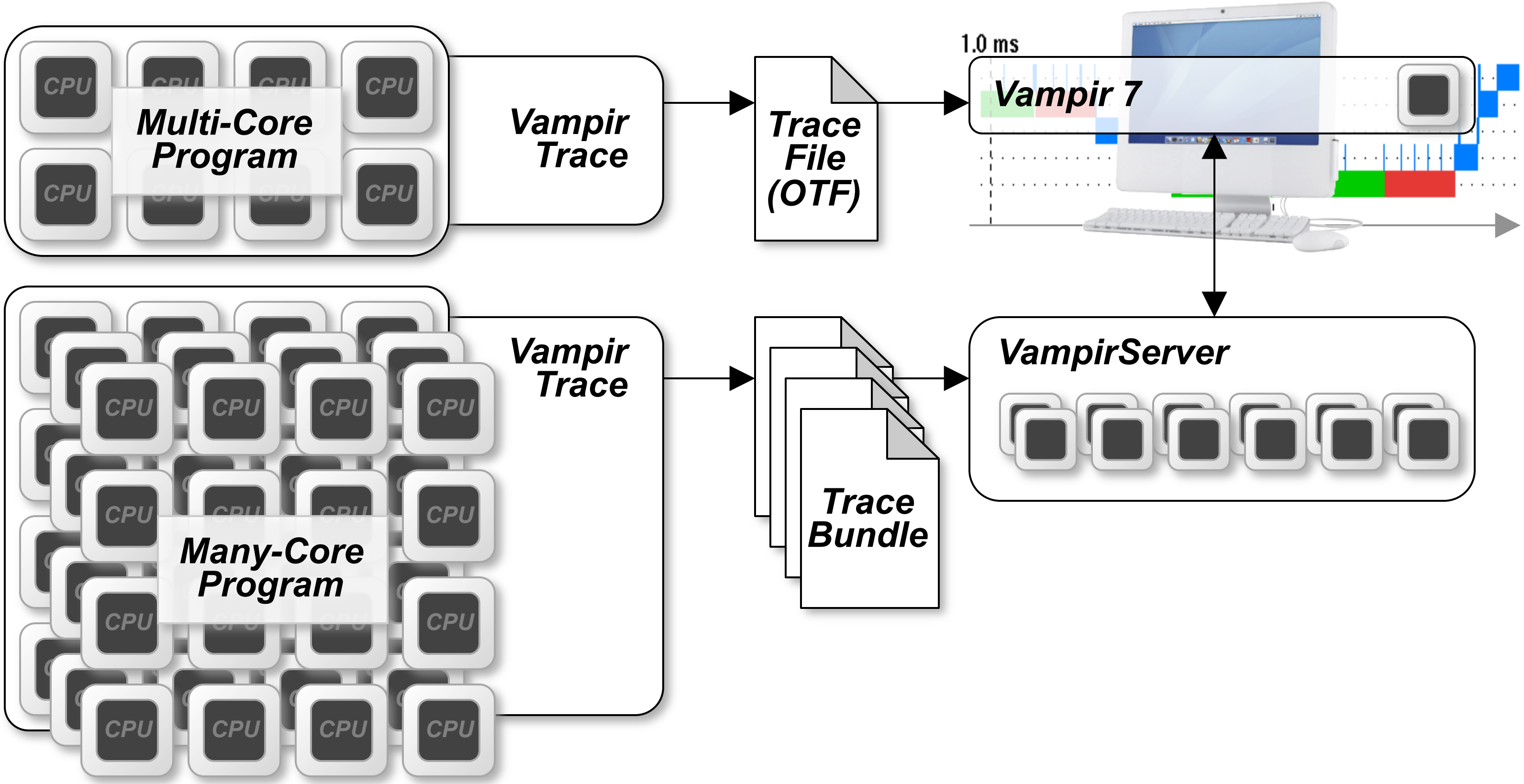
Profiling and Tracing

- Have an optimization phase
  - just like testing and debugging phase
- Use tools!
- Avoid *do-it-yourself-with-printf* solutions, really!

Trace Visualization

- Alternative and supplement to automatic analysis
- Show dynamic run-time behavior graphically
- Provide statistics and performance metrics
  - Global timeline for parallel processes/threads
  - Process timeline plus performance counters
  - Statistics summary display
  - Message statistics
  - more
- Interactive browsing, zooming, selecting
  - Adapt statistics to zoom level (time interval)
  - Also for very large and highly parallel traces

Vampir Toolset Architecture



Usage order of the Vampir Performance Analysis Toolset

1. Instrument your application with VampirTrace
2. Run your application with an appropriate test set

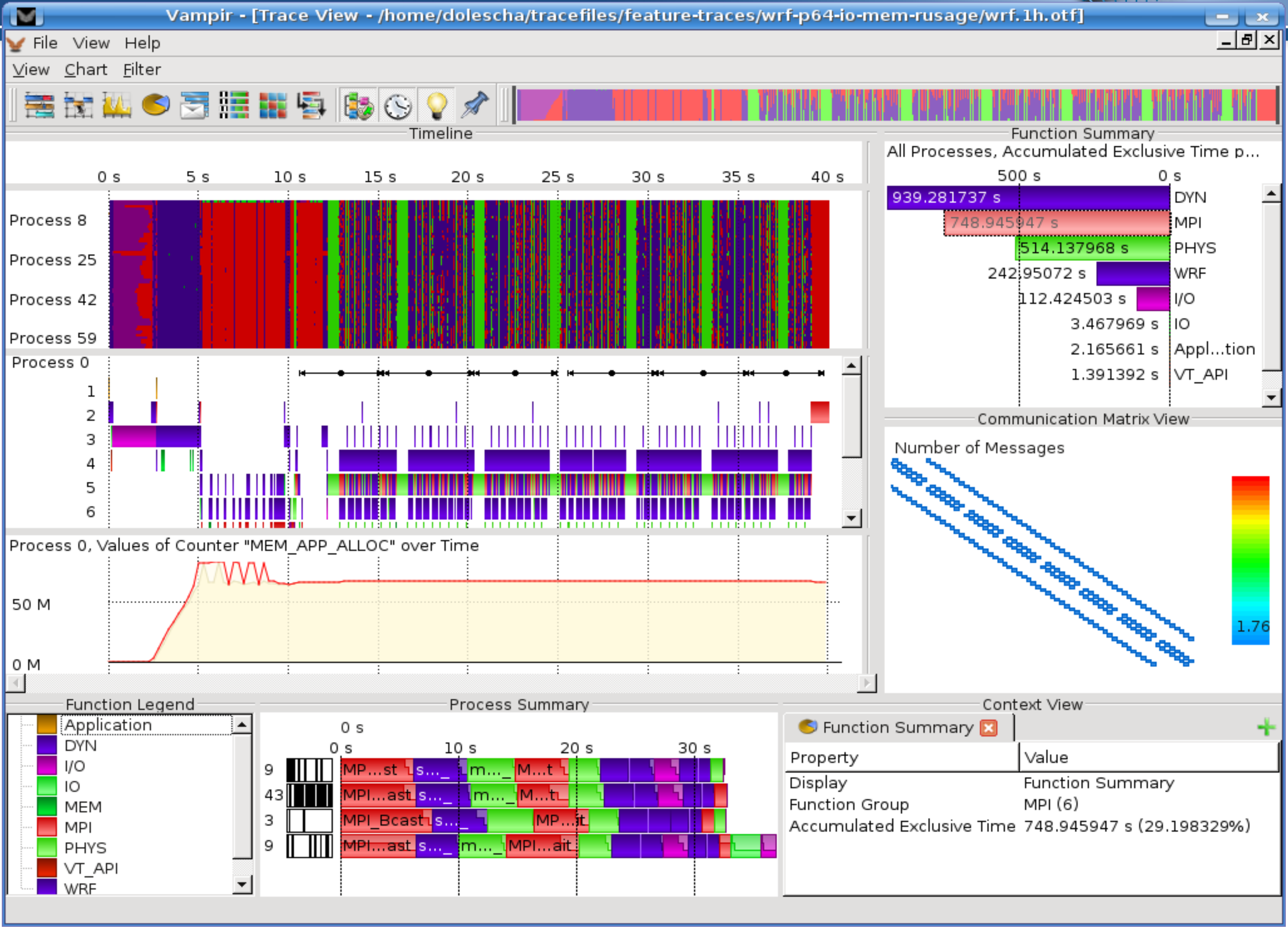
3. Analyze your trace file with Vampir
   - Small trace files can be analyzed on your local workstation
     1. Start your local Vampir
     2. Load trace file from your local disk
   - Large trace files should be stored on the cluster file system
     1. Start VampirServer on your analysis cluster
     2. Start your local Vampir
     3. Connect local Vampir with the VampirServer on the analysis cluster
     4. Load trace file from the cluster file system

Vampir Displays

The main displays of Vampir:

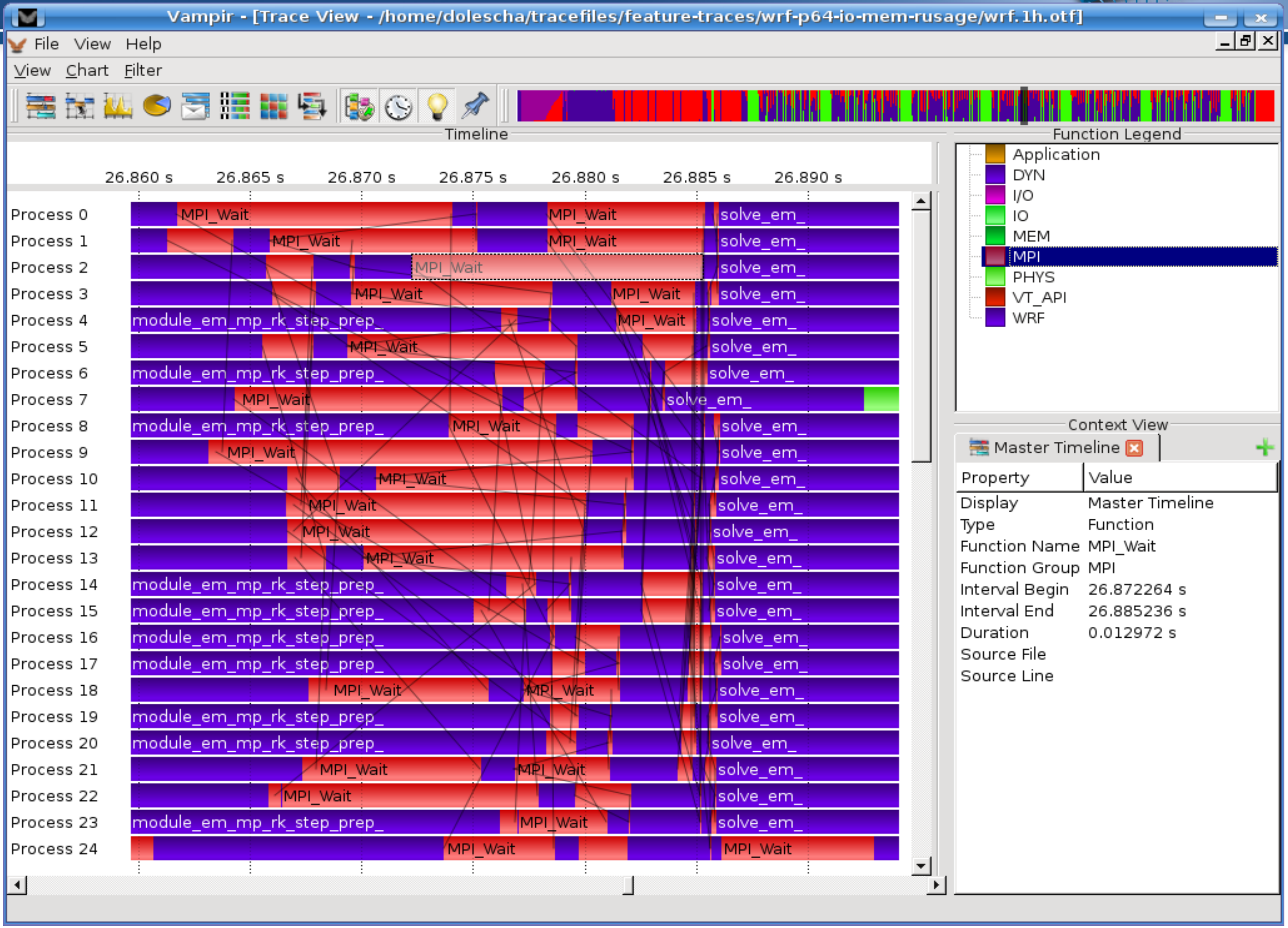
- Master Timeline (Global Timeline)
- Process and Counter Timeline
- Function Summary
- Message Summary
- Process Summary
- Communication Matrix
- Call Tree

Vampir 7: Displays for a WRF Trace with 64 Processes



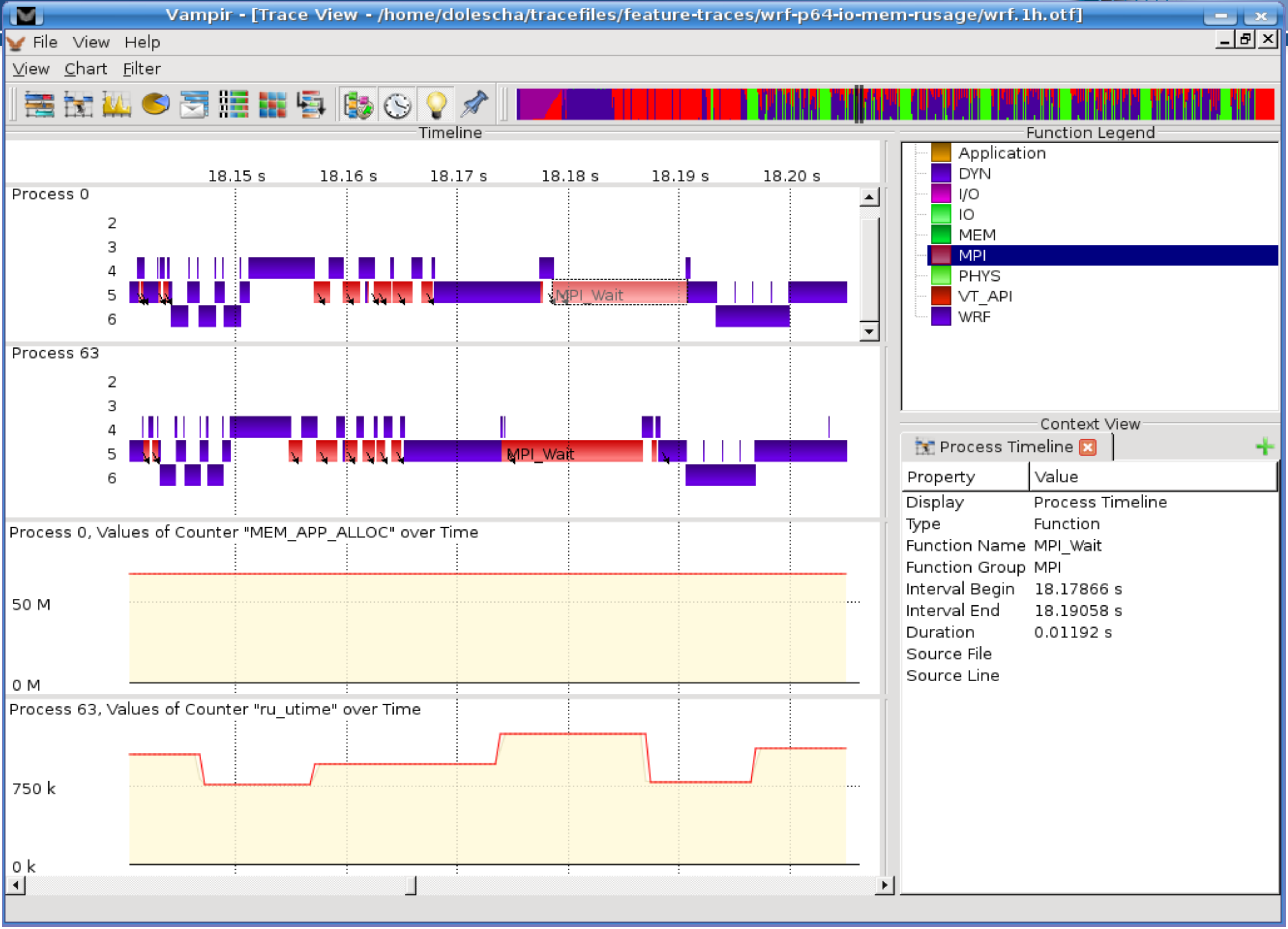
Property	Value
Display	Function Summary
Function Group	MPI (6)
Accumulated Exclusive Time	748.945947 s (29.198329%)

Master Timeline (Global Timeline)



Property	Value
Display	Master Timeline
Type	Function
Function Name	MPI_Wait
Function Group	MPI
Interval Begin	26.872264 s
Interval End	26.885236 s
Duration	0.012972 s
Source File	
Source Line	

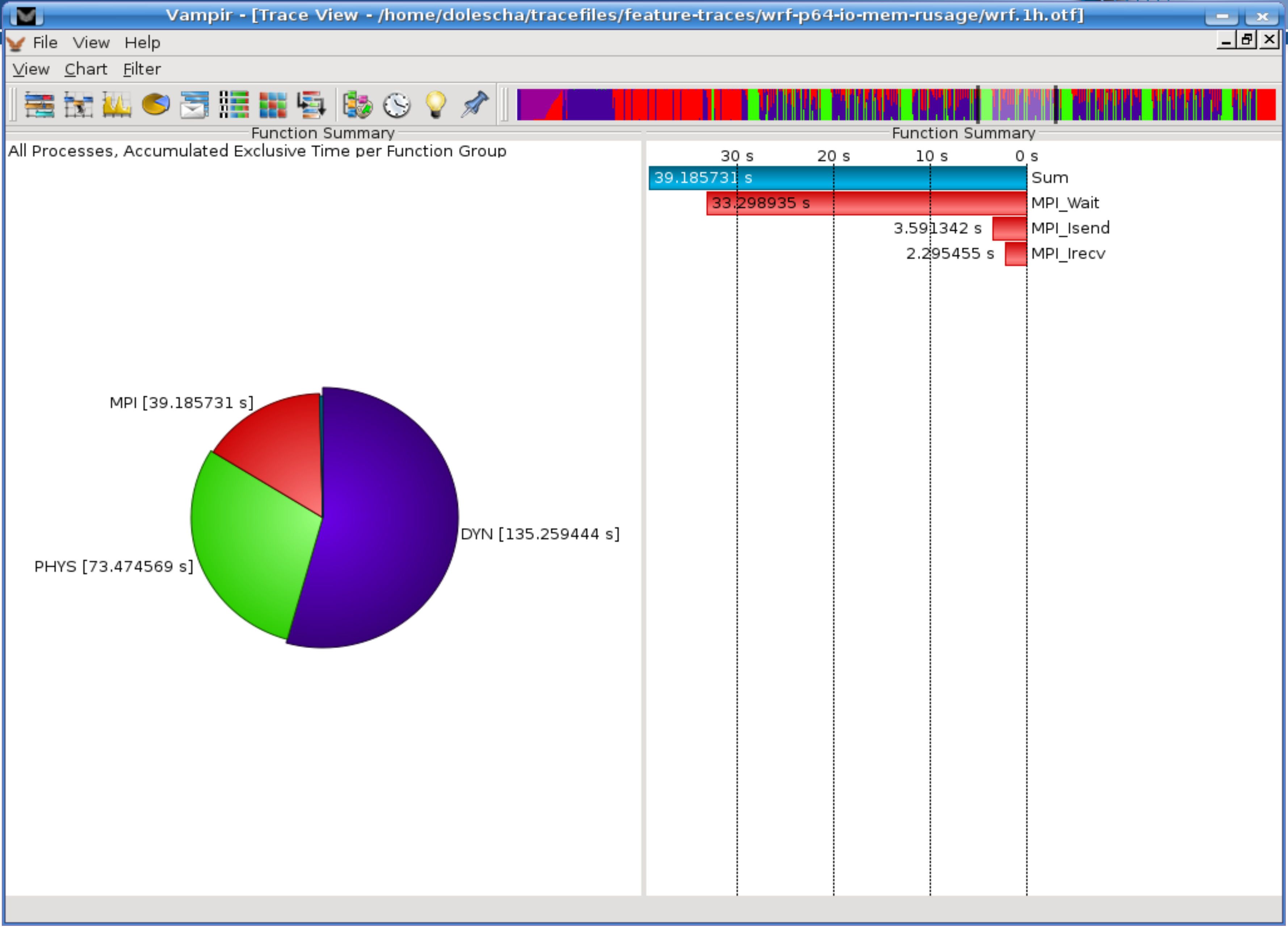
Process and Counter Timeline



Vampir - [Trace View - /home/dolescha/tracefiles/feature-traces/wrf-p64-io-mem-rusage/wrf.1h.otf]

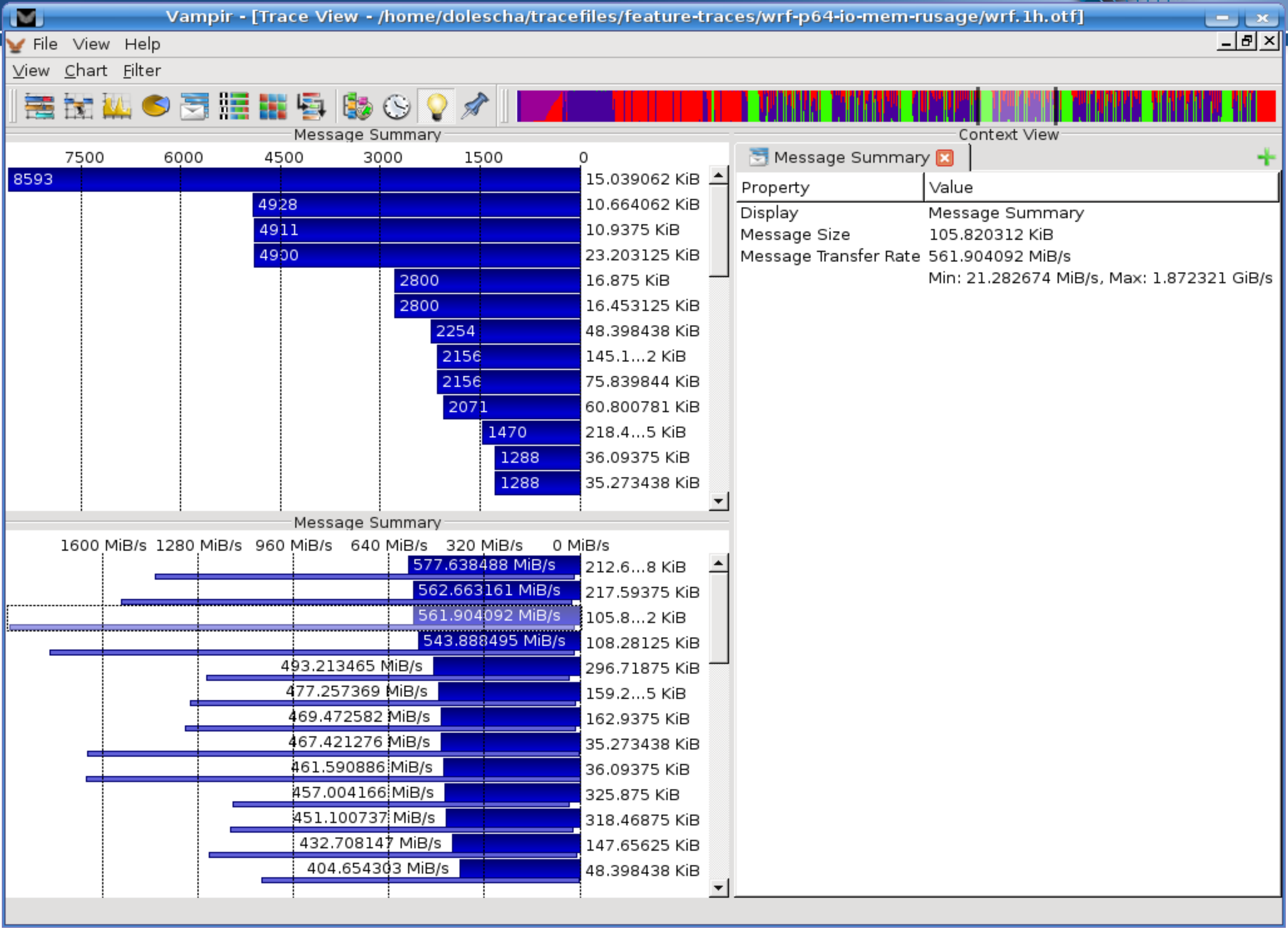
Property	Value
Display	Process Timeline
Type	Function
Function Name	MPI_Wait
Function Group	MPI
Interval Begin	18.17866 s
Interval End	18.19058 s
Duration	0.01192 s
Source File	
Source Line	

Function Summary



Vampir - [Trace View - /home/dolescha/tracefiles/feature-traces/wrf-p64-io-mem-rusage/wrf.1h.otf]

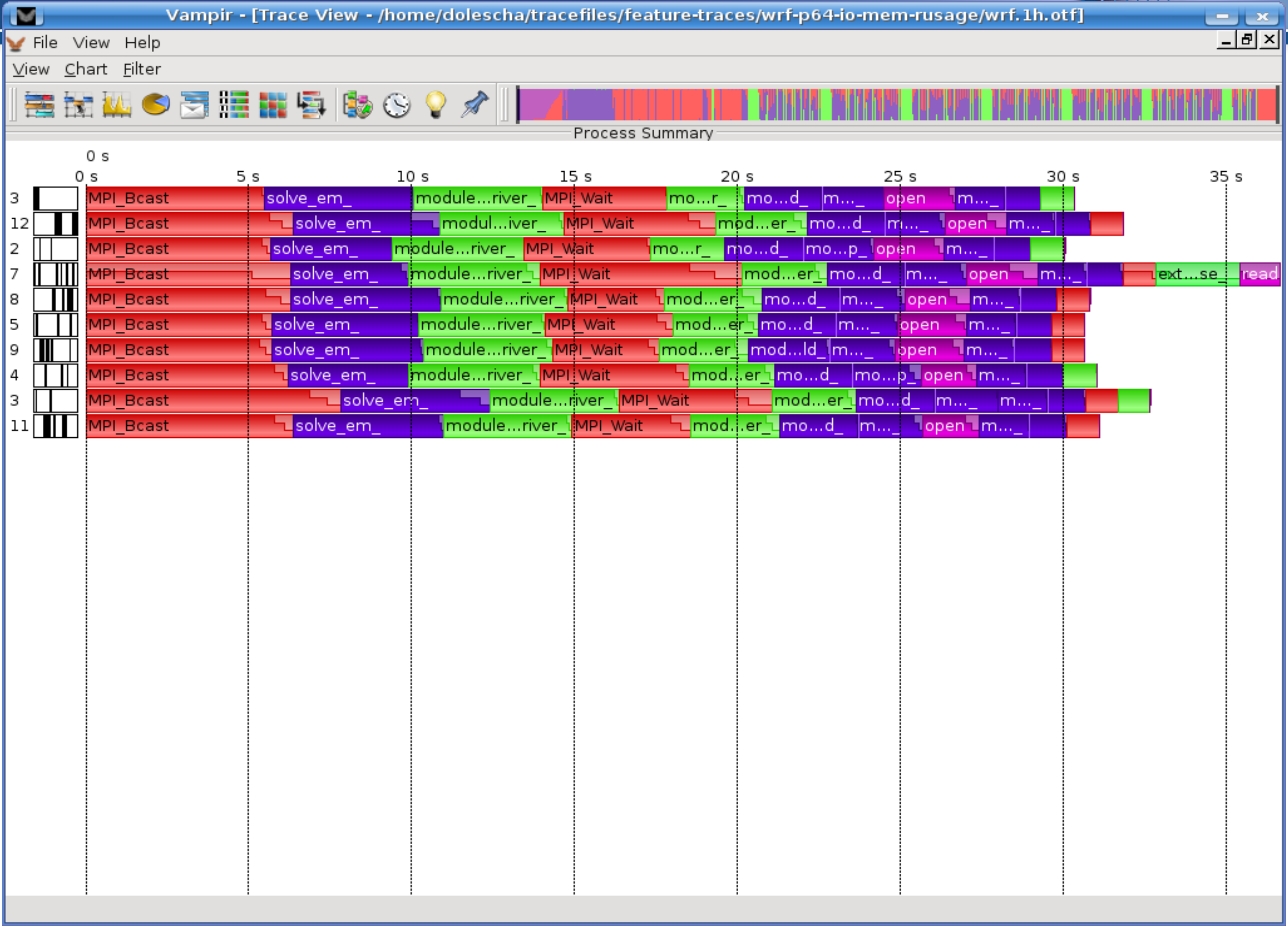
Message Summary



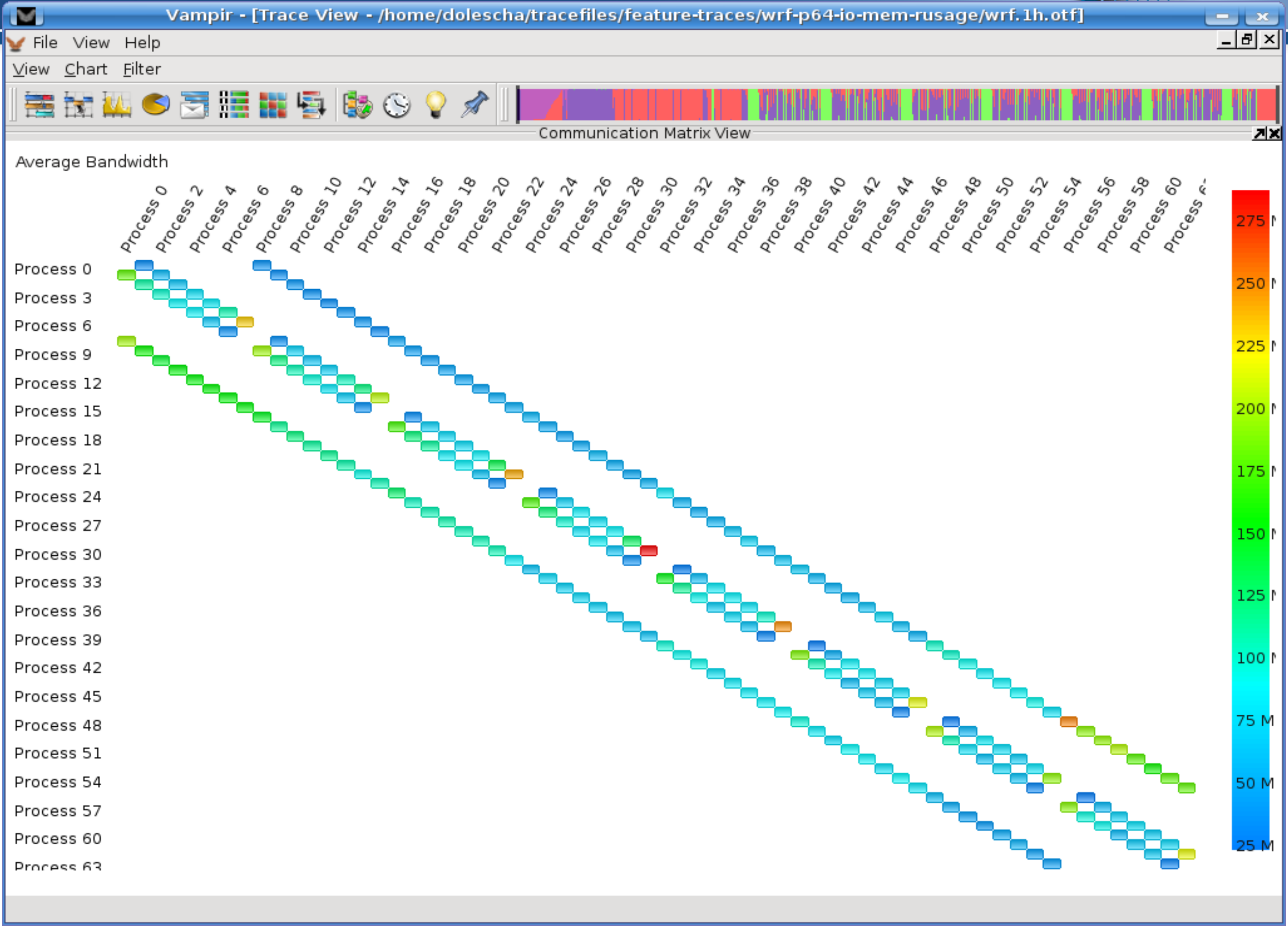
Vampir - [Trace View - /home/dolescha/tracefiles/feature-traces/wrf-p64-io-mem-rusage/wrf.1h.otf]
File View Help
View Chart Filter
Message Summary
7500 6000 4500 3000 1500 0
8593 15.039062 KiB
4928 10.664062 KiB
4911 10.9375 KiB
4900 23.203125 KiB
2800 16.875 KiB
2800 16.453125 KiB
2254 48.398438 KiB
2156 145.1...2 KiB
2156 75.839844 KiB
2071 60.800781 KiB
1470 218.4...5 KiB
1288 36.09375 KiB
1288 35.273438 KiB
Message Summary
1600 MiB/s 1280 MiB/s 960 MiB/s 640 MiB/s 320 MiB/s 0 MiB/s
577.638488 MiB/s 212.6...8 KiB
562.663161 MiB/s 217.59375 KiB
561.904092 MiB/s 105.8...2 KiB
543.888495 MiB/s 108.28125 KiB
493.213465 MiB/s 296.71875 KiB
477.257369 MiB/s 159.2...5 KiB
469.472582 MiB/s 162.9375 KiB
467.421276 MiB/s 35.273438 KiB
461.590886 MiB/s 36.09375 KiB
457.004166 MiB/s 325.875 KiB
451.100737 MiB/s 318.46875 KiB
432.708147 MiB/s 147.65625 KiB
404.654303 MiB/s 48.398438 KiB
Context View
Message Summary

Property	Value
Display	Message Summary
Message Size	105.820312 KiB
Message Transfer Rate	561.904092 MiB/s Min: 21.282674 MiB/s, Max: 1.872321 GiB/s

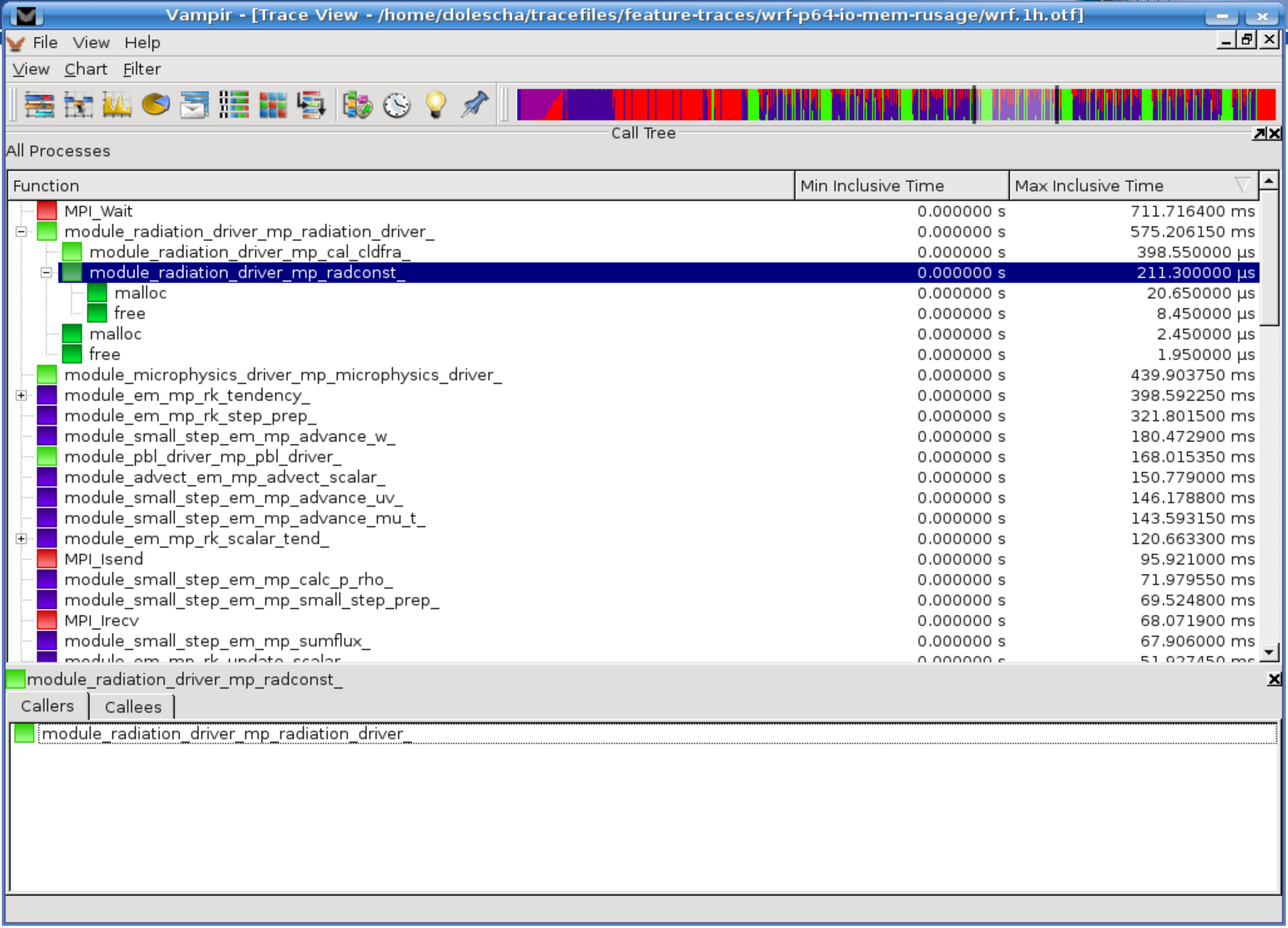
Process Summary



Communication Matrix



Call Tree



Vampir - [Trace View - /home/dolescha/tracefiles/feature-traces/wrf-p64-io-mem-rusage/wrf.1h.otf]

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_em_mp_rk_update_scalar_	0.000000 s	51.927450 ms

🟩 module_radiation_driver_mp_radconst_

🟩 module_radiation_driver_mp_radiation_driver_

Introduction: Profiling & Tracing

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
  - application, processes, threads, …

Trace Recording

- Save individual event records together with precise timestamp and process or thread ID
- Plus event specific information

Instrumentation & Measurement

- What do you need to do for it?
  - Use VampirTrace
- Instrumentation (automatic with compiler wrappers)

```
CC=icc
CXX=icpc
F90=ifc
MPICC=mpicc
```

```
CC=vtcc
CXX=vtcxx
F90=vtf90
MPICC=vtcc -vt:cc mpicc
```

- Re-compile & re-link
- Trace Run (run with appropriate test data set)

- More details later

What does VampirTrace do in the background?

- Instrumentation:
  - Via compiler wrappers
  - By underlying compiler with specific options
  - MPI instrumentation with replacement lib
  - OpenMP instrumentation with Opari
  - Also binary instrumentation with Dyninst
  - Also source2source instrumentation with PDT (Tau)
  - Partial manual instrumentation

What does VampirTrace do in the background?

- Trace Run:
  - Event data collection
  - Precise time measurement
  - Parallel timer synchronization
  - Collecting parallel process/thread traces
  - Collecting performance counters (from PAPI, memory usage, POSIX I/O calls and fork/system/exec calls, and more …)
  - Filtering and grouping of function calls

Summary

- Vampir & VampirServer
  - Interactive trace visualization and analysis
  - Intuitive browsing and zooming
  - Scalable to large trace data sizes (100GByte)
  - Scalable to high parallelism (2000 processes)
- Vampir for Linux, Windows and MacOS

- VampirTrace
  - Convenient instrumentation and measurement
  - Hides away complicated details
  - Provides many options and switches for experts
- VampirTrace is part of Open MPI since version 1.3

Vampir & VampirTrace

- Event Tracing in General

Common Event Types

- Enter/leave of function/routine/region
  - time stamp, process/thread, function ID
- Send/receive of P2P message (MPI)
  - time stamp, sender, receiver, length, tag, communicator
- Collective communication (MPI)
  - time stamp, process, root, communicator, # bytes
- Hardware performance counter values
  - time stamp, process, counter ID, value
- etc.

Profiling and Tracing

- Tracing Advantages
  - Preserve temporal and spatial relationships
  - Allow reconstruction of dynamic behavior on any required abstraction level
  - Profiles can be calculated from traces

- Tracing Disadvantages
  - Traces can become very large
  - May cause perturbation
  - Instrumentation and tracing is complicated
    - Event buffering, clock synchronization, …

Instrumentation

- Instrumentation: Process of modifying programs to detect and report events

- There are various ways of instrumentation:
  - Manually
    - Large effort, error prone
    - Difficult to manage
  - Automatically
    - Via source to source translation
    - Via compiler instrumentation
    - Program Database Toolkit (PDT)
    - OpenMP Pragma And Region Instrumenter (Opari)

Open Trace Format (OTF)

- Open source trace file format
- Available at http://www.tu-dresden.de/zih/otf
- Includes powerful libotf for reading/parsing/writing in custom applications
- Multi-level API:
  - High level interface for analysis tools
  - Low level interface for trace libraries
- Actively developed by TU Dresden in cooperation with the University of Oregon and the Lawrence Livermore National Laboratory

Practical Instrumentation

- Instrumentation with VampirTrace
  - Hide instrumentation in compiler wrapper
  - Use underlying compiler, add appropriate options

```
CC = mpicc

CC = vtcc –vt:cc mpicc
```

- Test Run
  - User representative test input
  - Set parameters, environment variables, etc.
  - Perform trace run
- Get Trace

Source Code Instrumentation

```
int foo(void* arg) {

    if (cond) {

        return 1;
    }

    return 0;
}
```

```
int foo(void* arg) {
    enter(7);
    if (cond) {
        leave(7);
        return 1;
    }
    leave(7);
    return 0;
}
```

manually or automatically