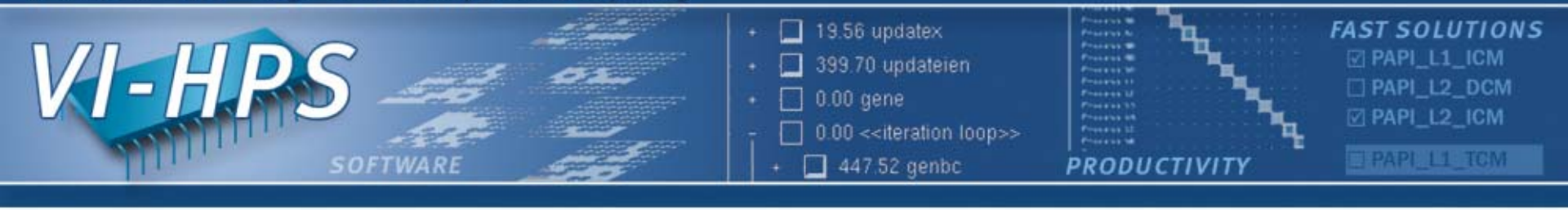


VAMPIR & VAMPIRTRACE INTRODUCTION AND OVERVIEW

6th VI-HPS Tuning Workshop at SARA, Amsterdam
May 26th – May 28th, 2010

Andreas Knüpfer, Jens Doleschal,
andreas.knuepfer@tu-dresden.de
jens.doleschal@tu-dresden.de

Overview

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

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

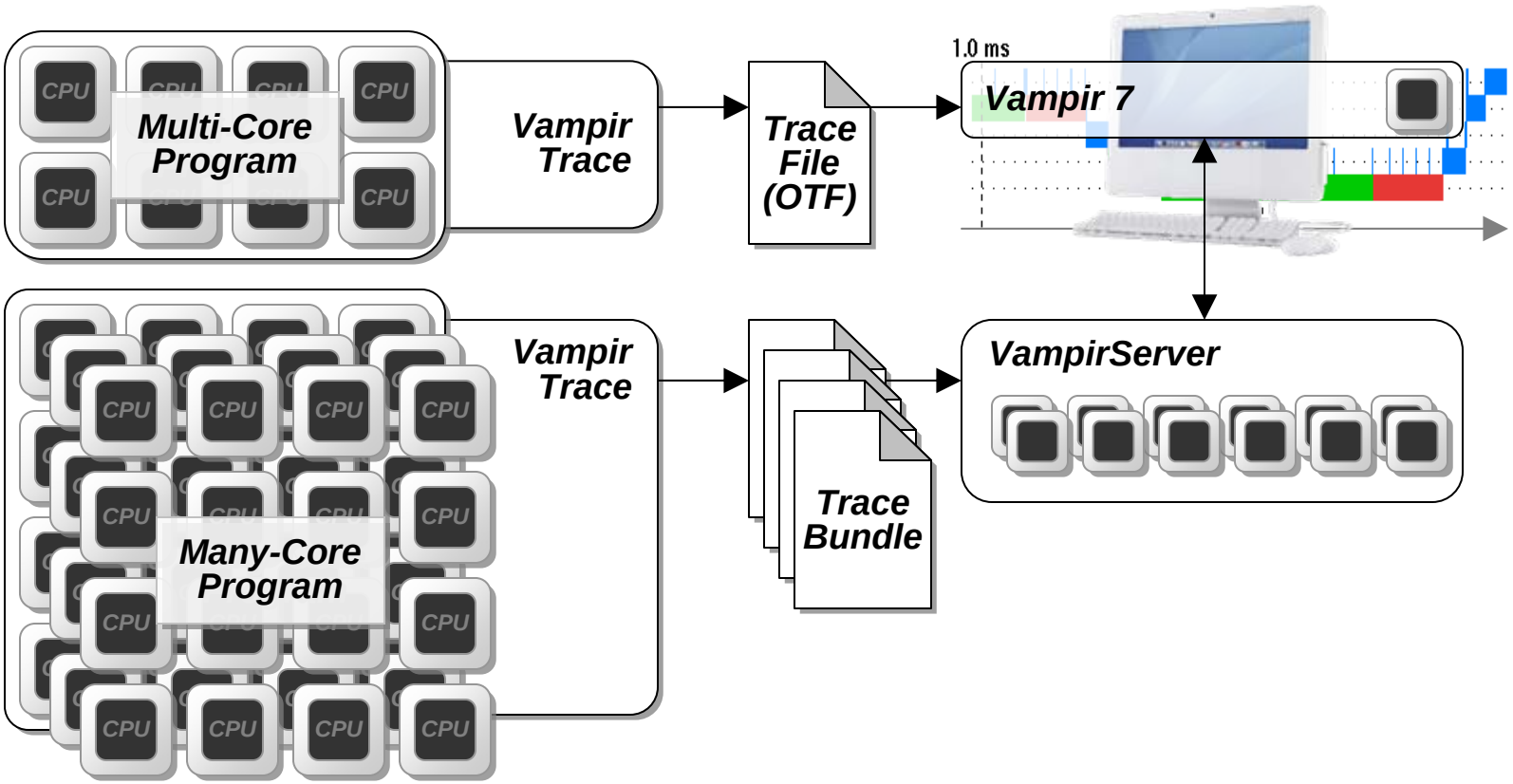
Profiling and Tracing

- 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

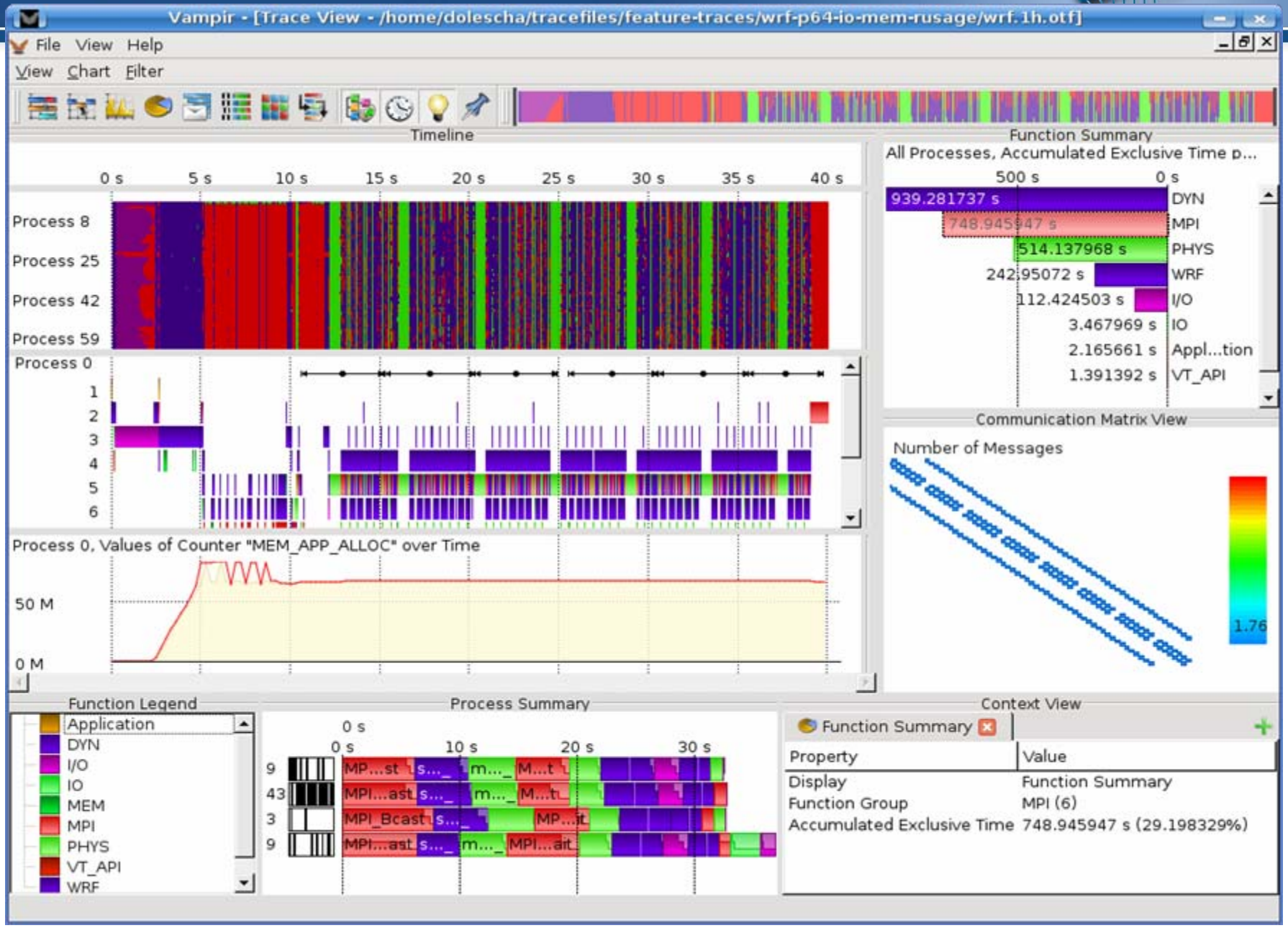
Vampir Toolset Architecture



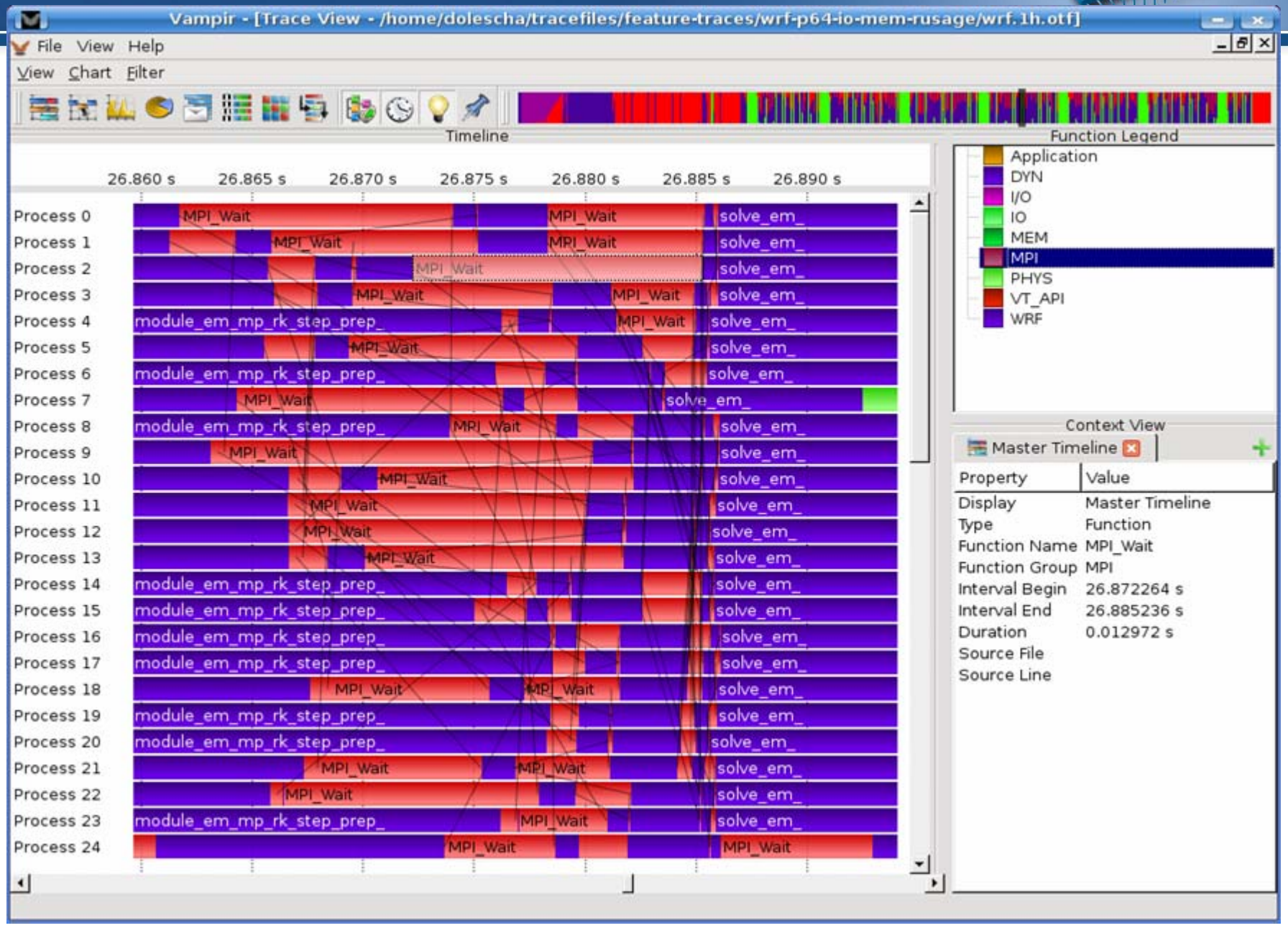
The main displays of Vampir:

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

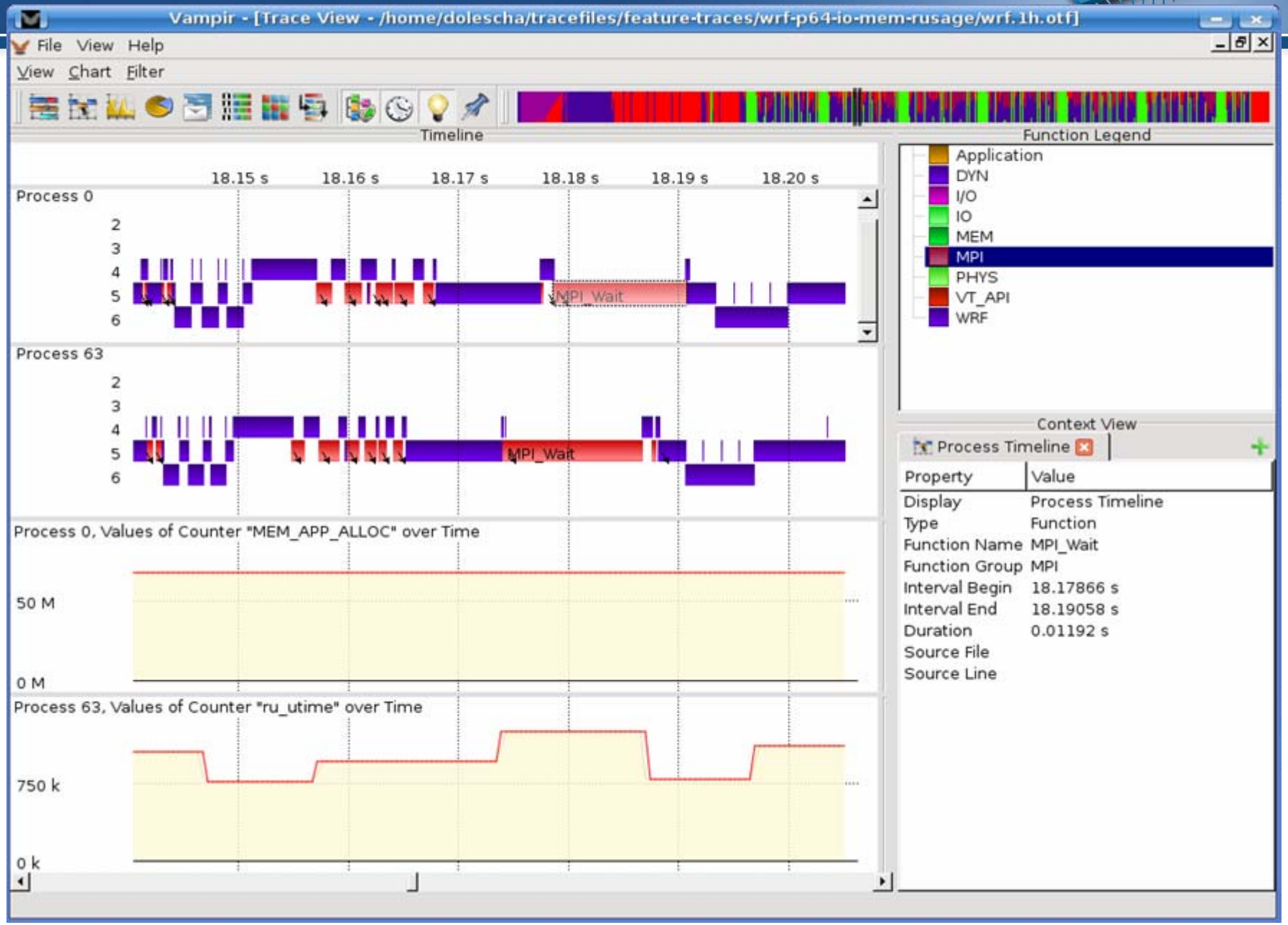
Vampir 7: Displays for a WRF Trace with 64 Processes



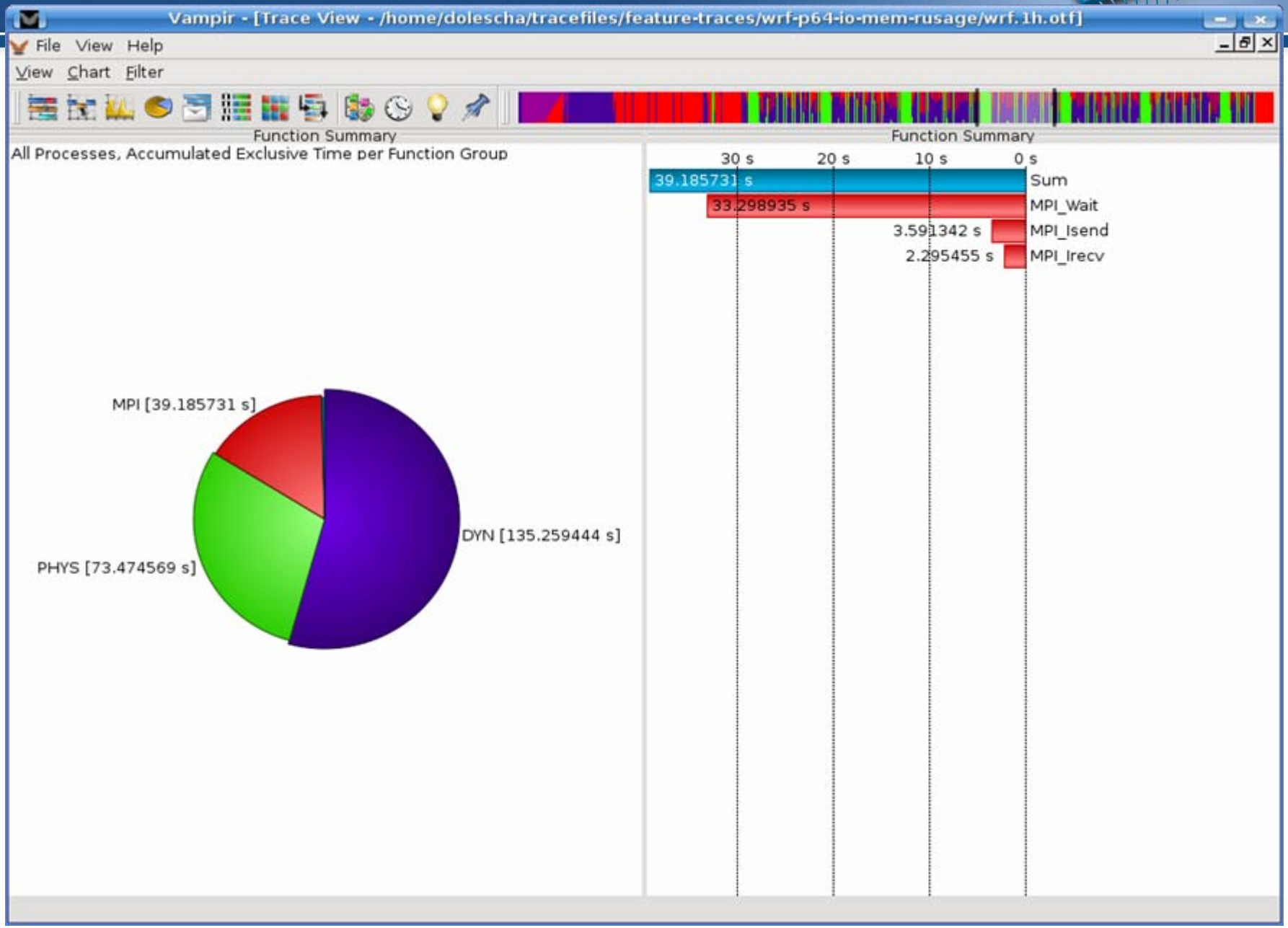
Master Timeline (Global Timeline)



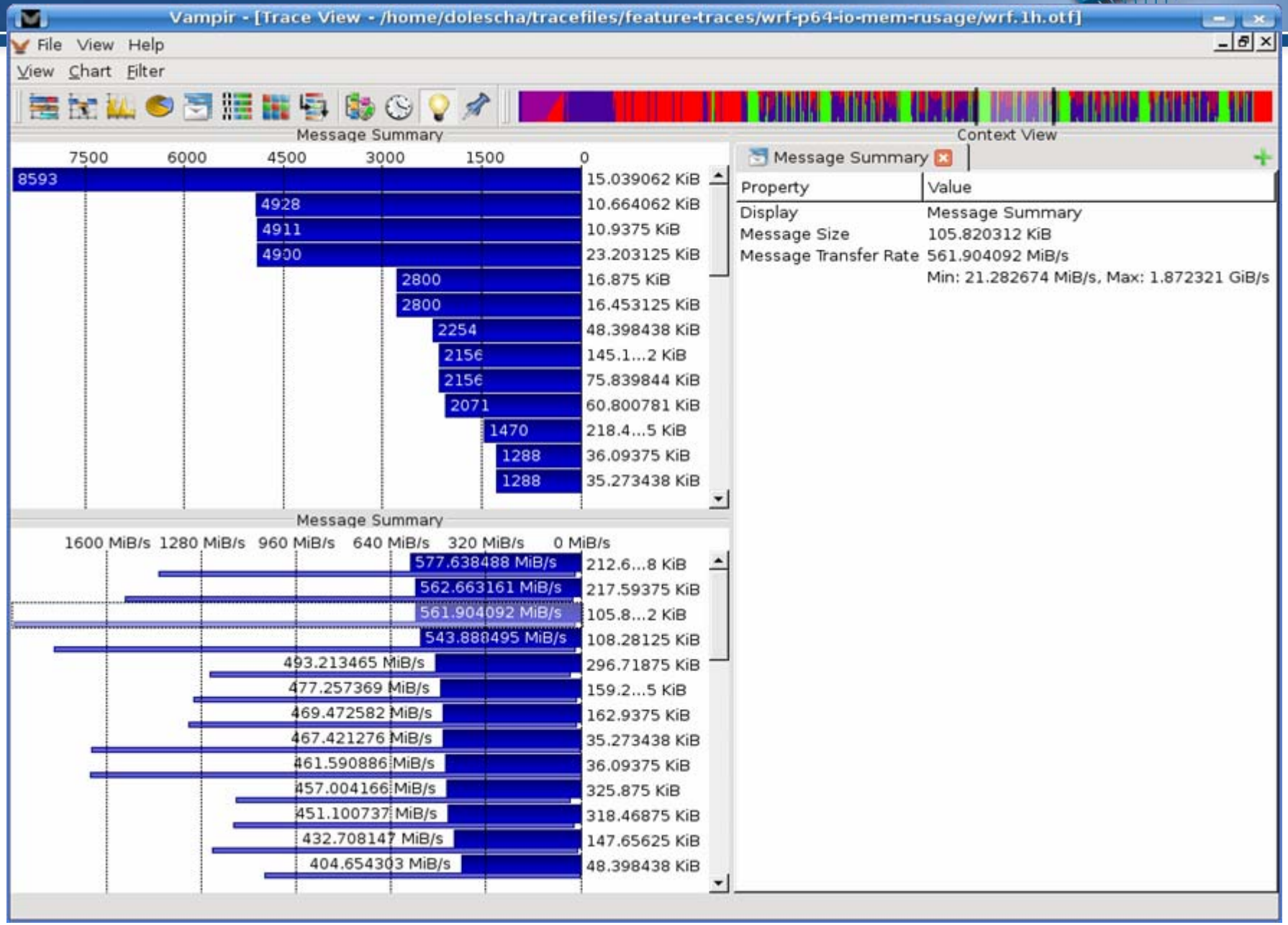
Process and Counter Timeline



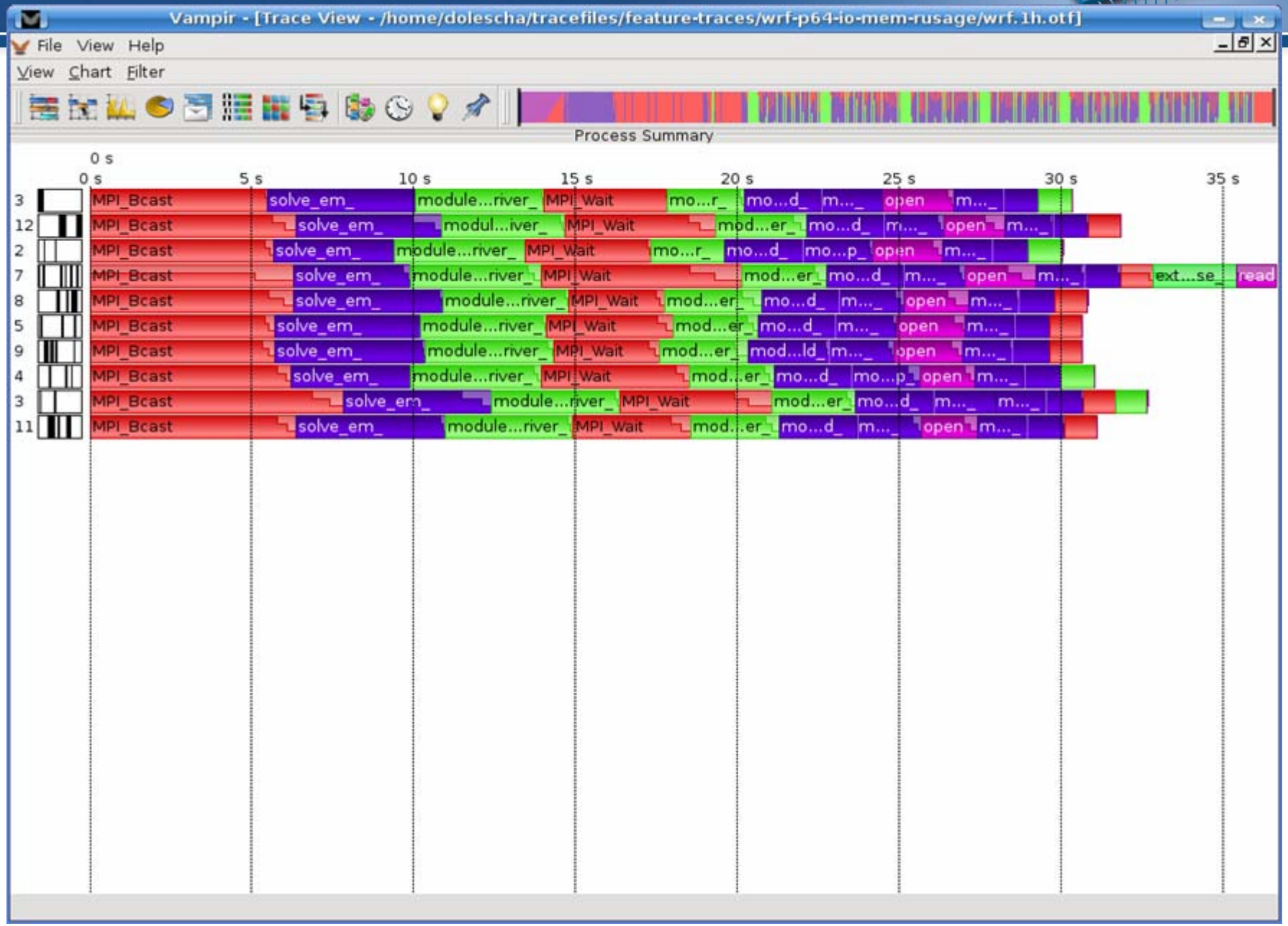
Function Summary



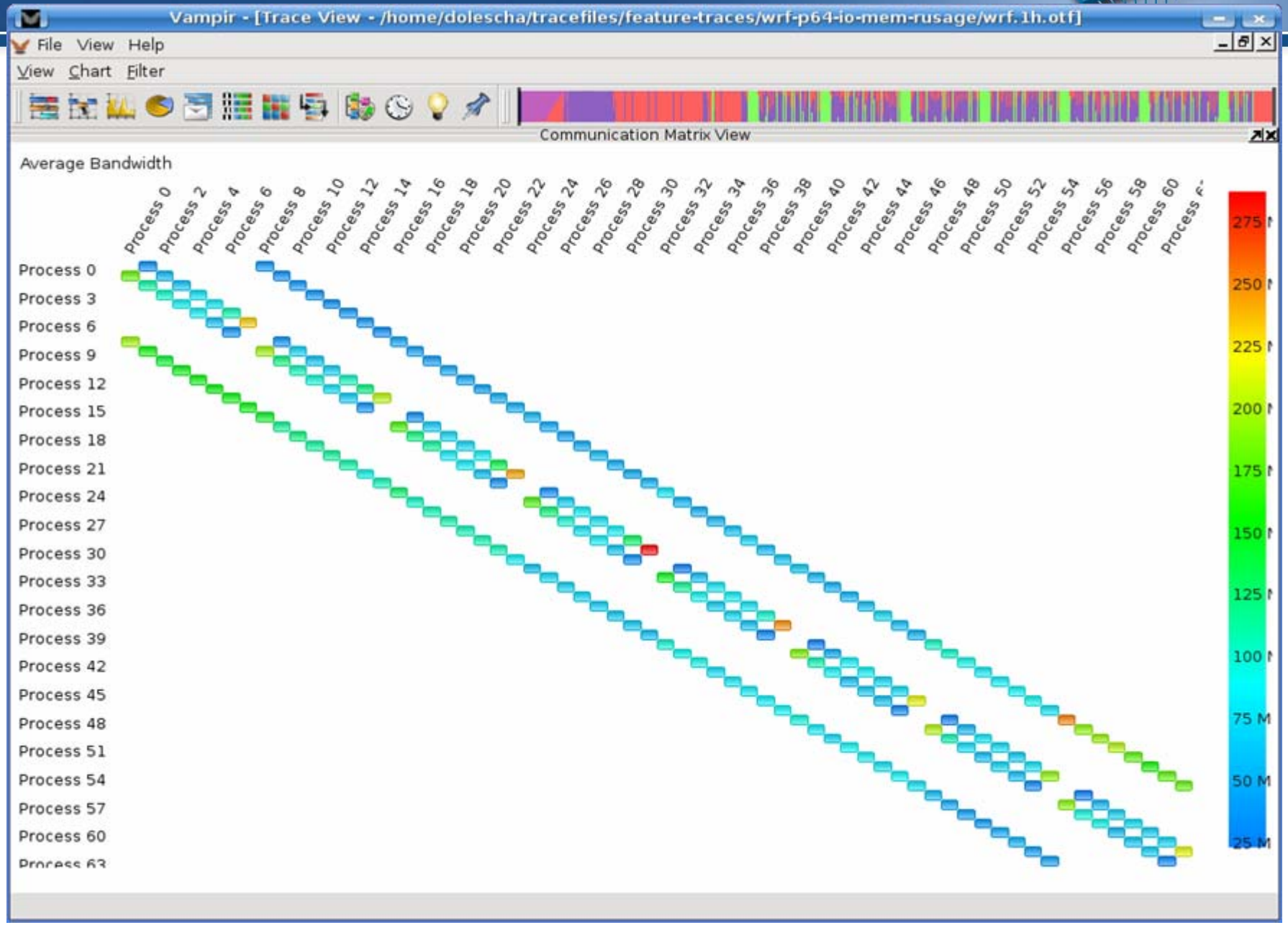
Message Summary



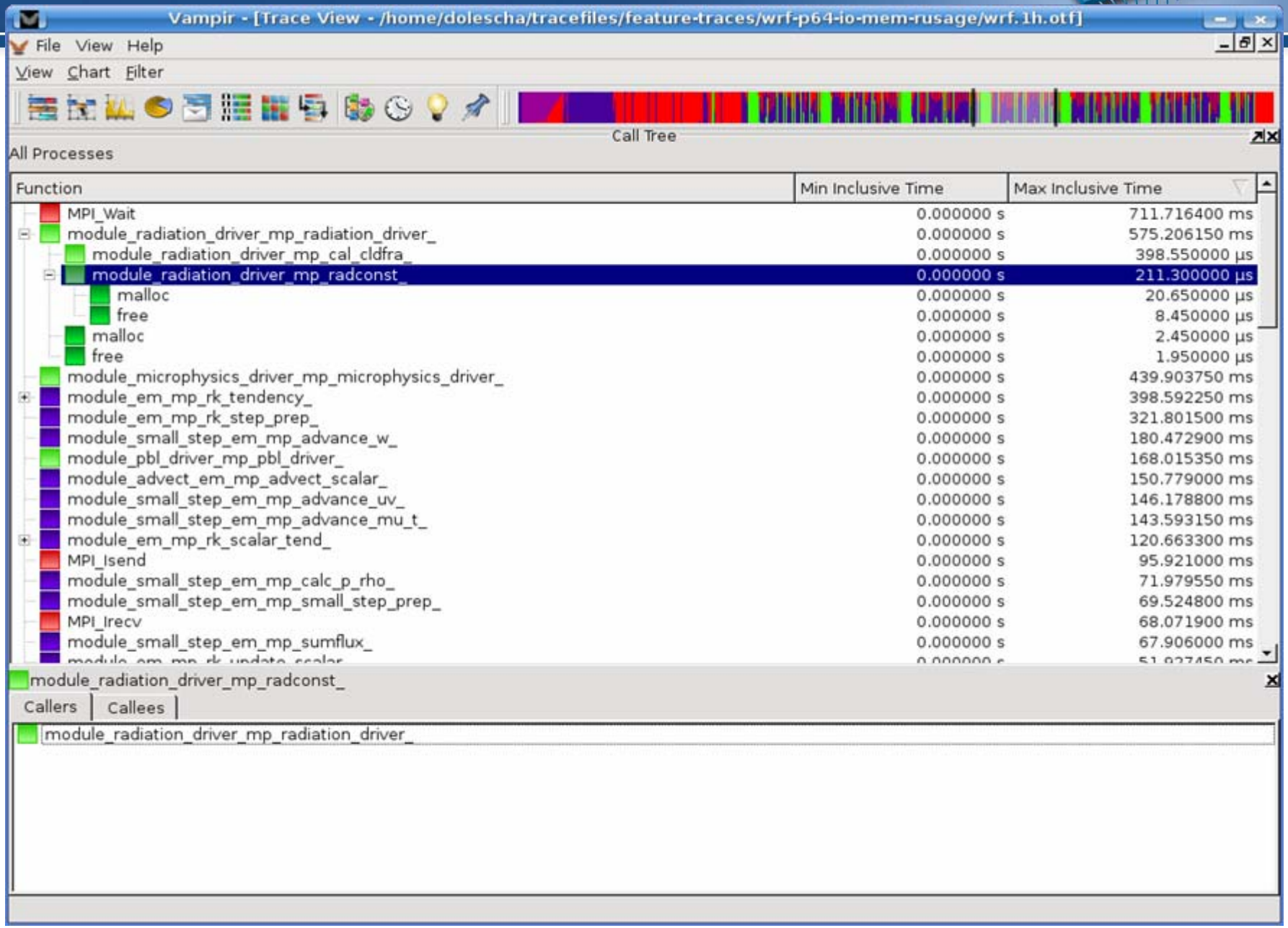
Process Summary



Communication Matrix



Call 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_

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

- 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
  - 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 Windows in progress, beta available

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

Thanks for your attention.

Acknowledgement

Staff at ZIH - TU Dresden:

Ronny Brendel, Holger Brunst, Jens Doleschal, Ronald Geisler, Daniel Hackenberg, Michael Heyde, Tobias Hilbrich, Rene Jäkel, Matthias Jurenz, Michael Kluge, Andreas Knüpfer, Matthias Lieber, Holger Mickler, Hartmut Mix, Matthias Müller, Wolfgang E. Nagel, Reinhard Neumann, Michael Peter, Heide Rohling, Johannes Spazier, Michael Wagner, Matthias Weber, Bert Wesarg

VAMPIR & VAMPIRTRACE DETAILS AND HANDS-ON

6th VI-HPS Tuning Workshop at SARA, Amsterdam
May 26th – May 28th, 2010

Andreas Knüpfer, Jens Doleschal,
andreas.knuepfer@tu-dresden.de
jens.doleschal@tu-dresden.de

Overview

- Event Tracing in General
- Hands-on: NPB 3.3 BT-MPI
- Finding Performance Bottlenecks

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.

- 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: 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

- 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

VAMPIR & VAMPIRTRACE HANDS-ON: NPB 3.3 BT-MPI

- Move into tutorial directory in your home directory

```
% cd NPB3.3-MPI
```

- Select the VampirTrace compiler wrappers

```
% gedit config/make.def
    -> comment out line 32, resulting in:
            ...
            32: #MPIF77 = mpif77
            ...
    -> remove the comment from line 38, resulting in:
            ...
            38: MPIF77 = vtf77 –vt:f77 mpif77
            ...
    -> comment out line 88, resulting in:
            ...
            88: #MPICC = mpicc
            ...
    -> remove the comment from line 94, resulting in:
            ...
            94: MPICC = vtcc  -vt:cc mpicc
            ...
```

- Build benchmark

```
% make clean; make suite
```

- Launch as MPI application

```
% cd bin.vampir; export VT_FILE_PREFIX=bt_1_initial
% mpiexec –np 16 bt_W.16

 NAS Parallel Benchmarks 3.3 -- BT Benchmark

 Size:   24x  24x  24
 Iterations:  200    dt:   0.0008000
 Number of active processes:    16

 Time step    1
 ...
 Time step  180
[0]VampirTrace: Maximum number of buffer flushes reached \
(VT_MAX_FLUSHES=1)
[0]VampirTrace: Tracing switched off permanently
 Time step  200
 ...
```

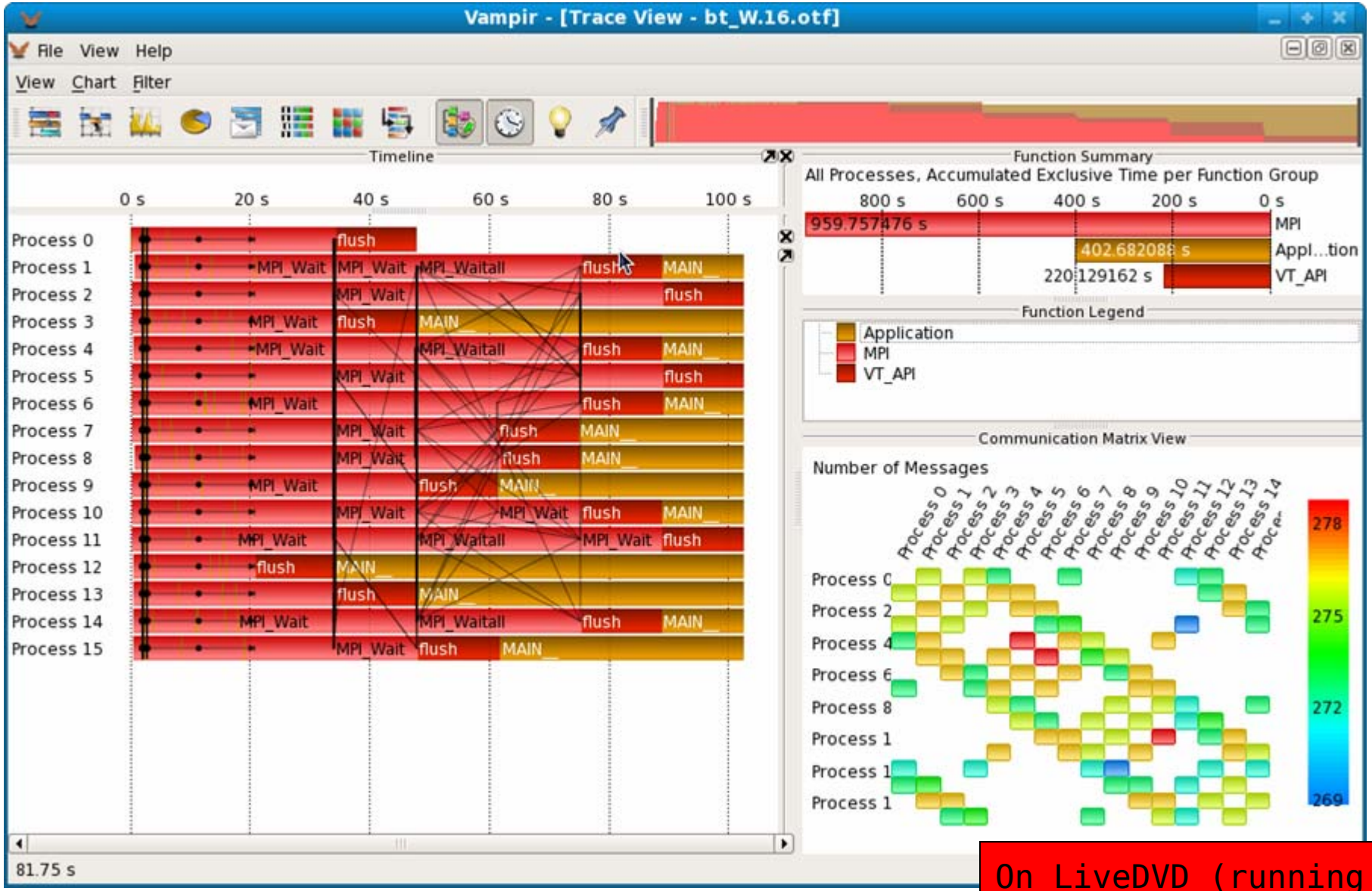
- Resulting trace files

```
% ls -alh
4,1M bt_1_initial.16
4,9K bt_1_initial.16.0.def.z
29 bt_1_initial.16.0.marker.z
12M bt_1_initial.16.10.events.z
12M bt_1_initial.16.1.events.z
11M bt_1_initial.16.2.events.z
12M bt_1_initial.16.3.events.z
...
11M bt_1_initial.16.c.events.z
12M bt_1_initial.16.d.events.z
12M bt_1_initial.16.e.events.z
12M bt_1_initial.16.f.events.z
66 bt_1_initial.16.otf
```

- Visualization with Vampir7

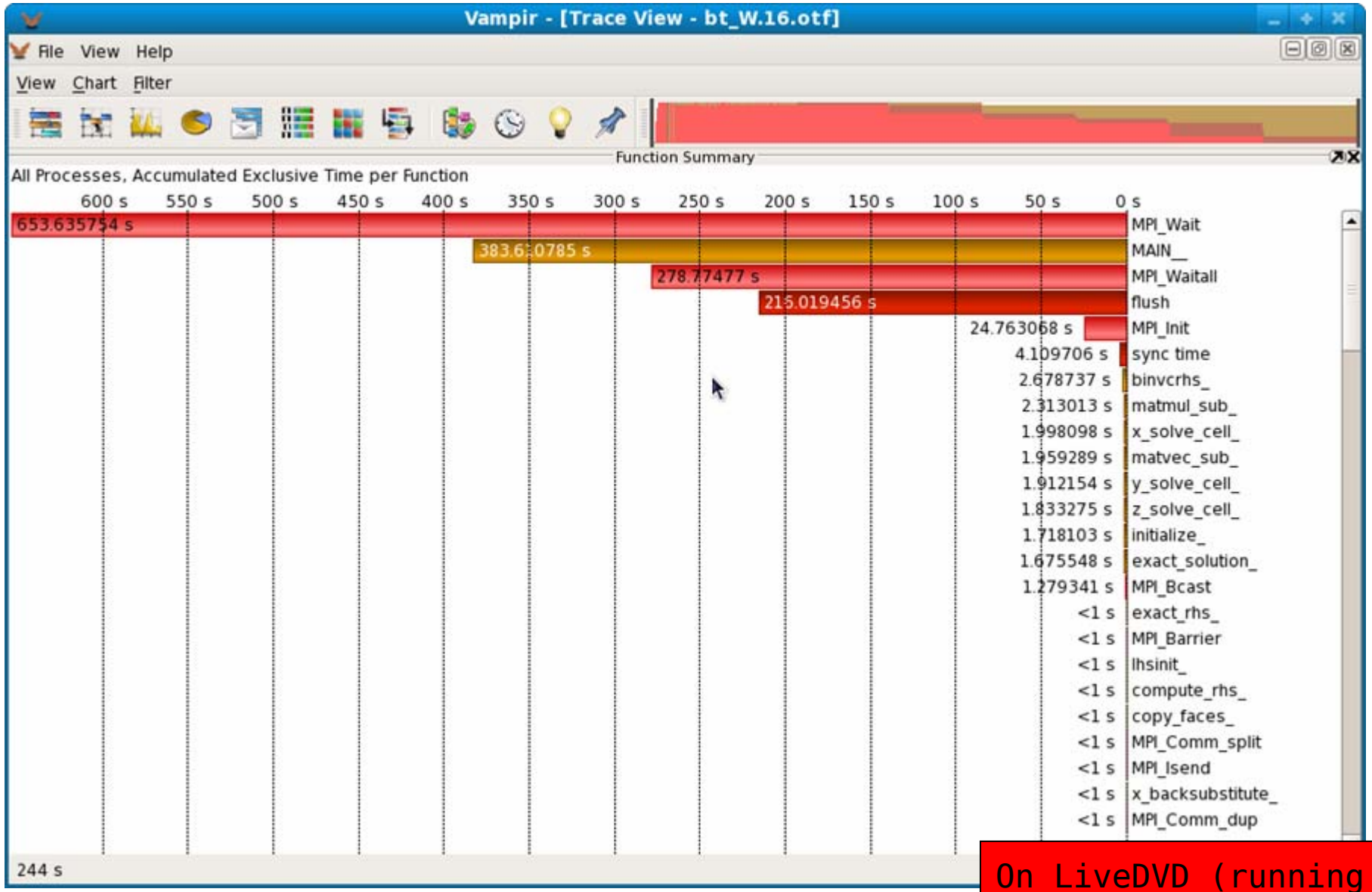
```
% vampir bt_1_initial.16.otf
```

Hands-on: NPB 3.3 BT-MPI



On LiveDVD (running in

Hands-on: NPB 3.3 BT-MPI



On LiveDVD (running in

- Decrease number of buffer flushes by increasing the buffer size

```
% export VT_MAX_FLUSHES=1 VT_BUFFER_SIZE=120M
```

- Set a new file prefix

```
% export VT_FILE_PREFIX=bt_2_buffer_120M
```

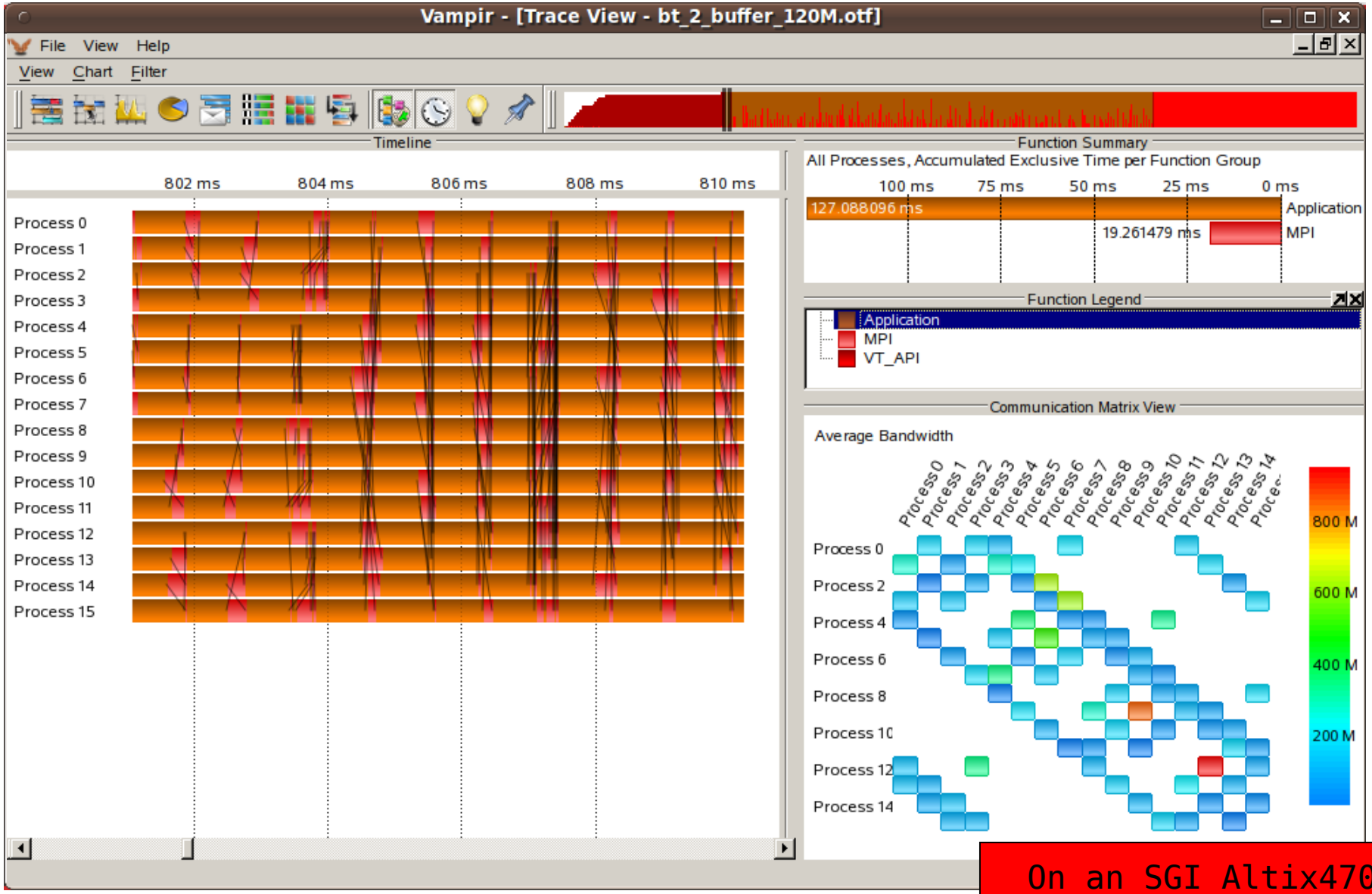
- Launch as MPI application

```
% mpiexec -np 16 bt_W.16
```

Remove the old trace first !

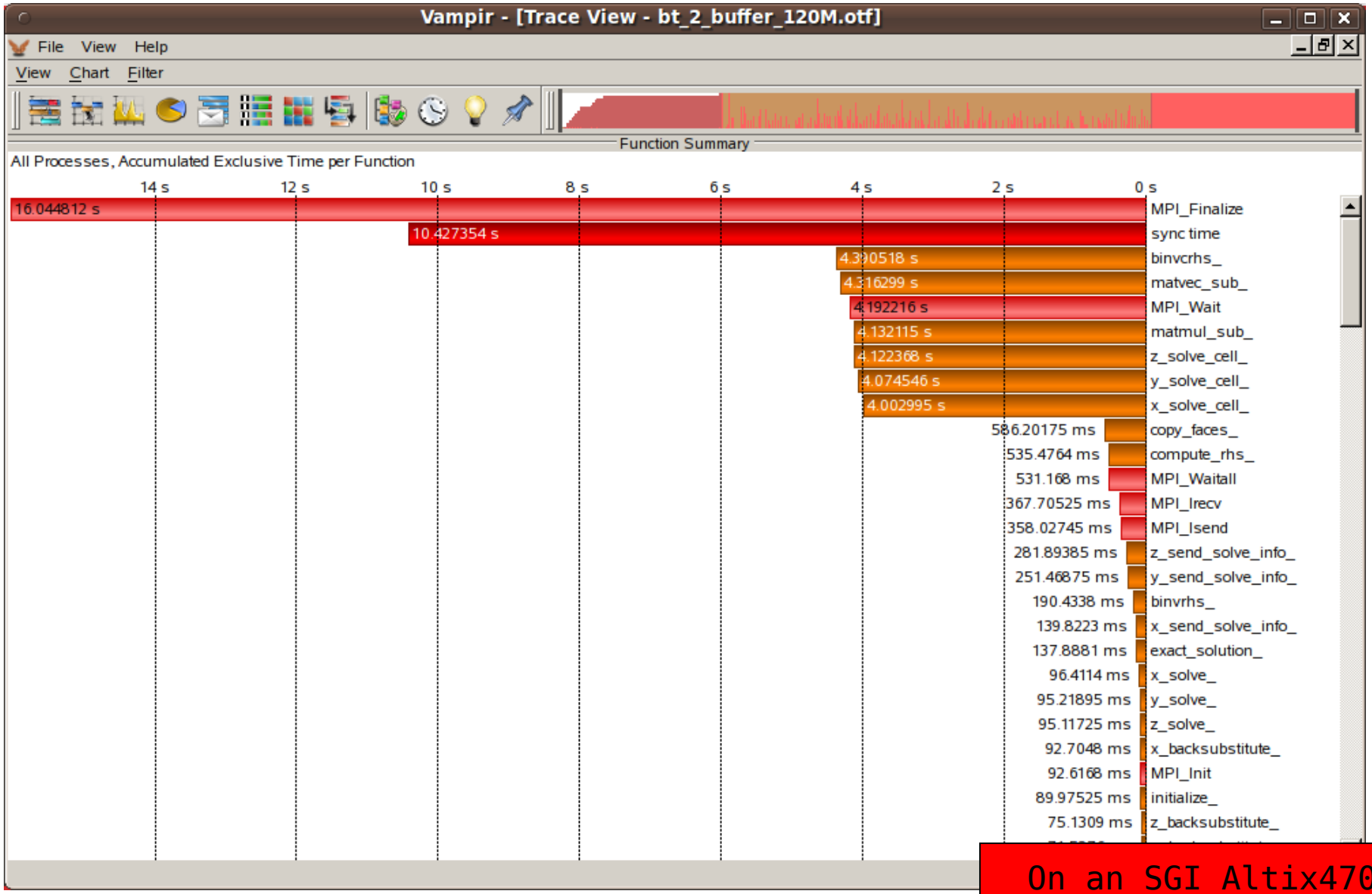
Only for laptops with at least 2GB main memory !

Hands-on: NPB 3.3 BT-MPI



On an SGI Altix4700

Hands-on: NPB 3.3 BT-MPI



On an SGI Altix4700

Function Filtering

- Filtering is one of the ways to reduce trace size
- Environment variable VT_FILTER_SPEC

```
% export VT_FILTER_SPEC = /home/user/filter.spec
```

- Filter definition file contains a list of filters

```
my_*;test_* -- 1000
debug_* -- 0
calculate -- -1
* -- 1000000
```

- See also the vtfilter tool
  - can generate a customized filter file
  - can reduce the size of existing trace files

Function Grouping

- Groups can be defined for related functions
  - Groups can be assigned different colors, highlighting different activities
- Environment variable VT_GROUPS_SPEC

```
% export VT_GROUPS_SPEC = /home/user/groups.spec
```

- Group file contains a list of associated entries

```
CALC=calculate
MISC=my*;test
UNKNOWN=*
```

- Generate filter specification file

```
% vtfilter -gen -fo filter.txt -r 10 -stats  \
  -p bt_2_buffer_120M.otf  #or bt_1_initial if you have less than 2G
% export VT_FILTER_SPEC=filter.txt
```

- Set a new file prefix

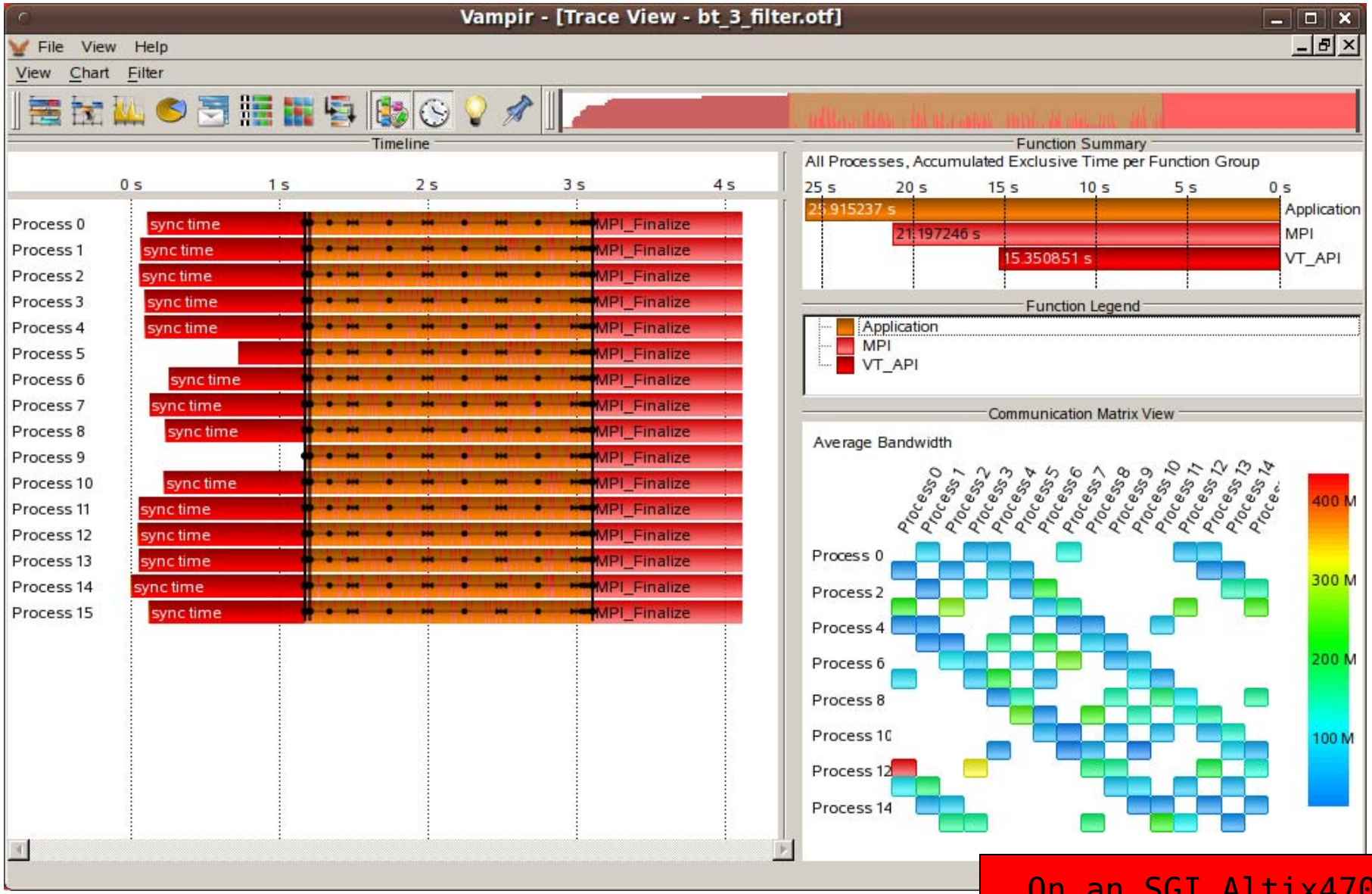
```
% export VT_FILE_PREFIX=bt_3_filter
```

- Launch as MPI application

```
% mpiexec -np 16 bt_W.16
```

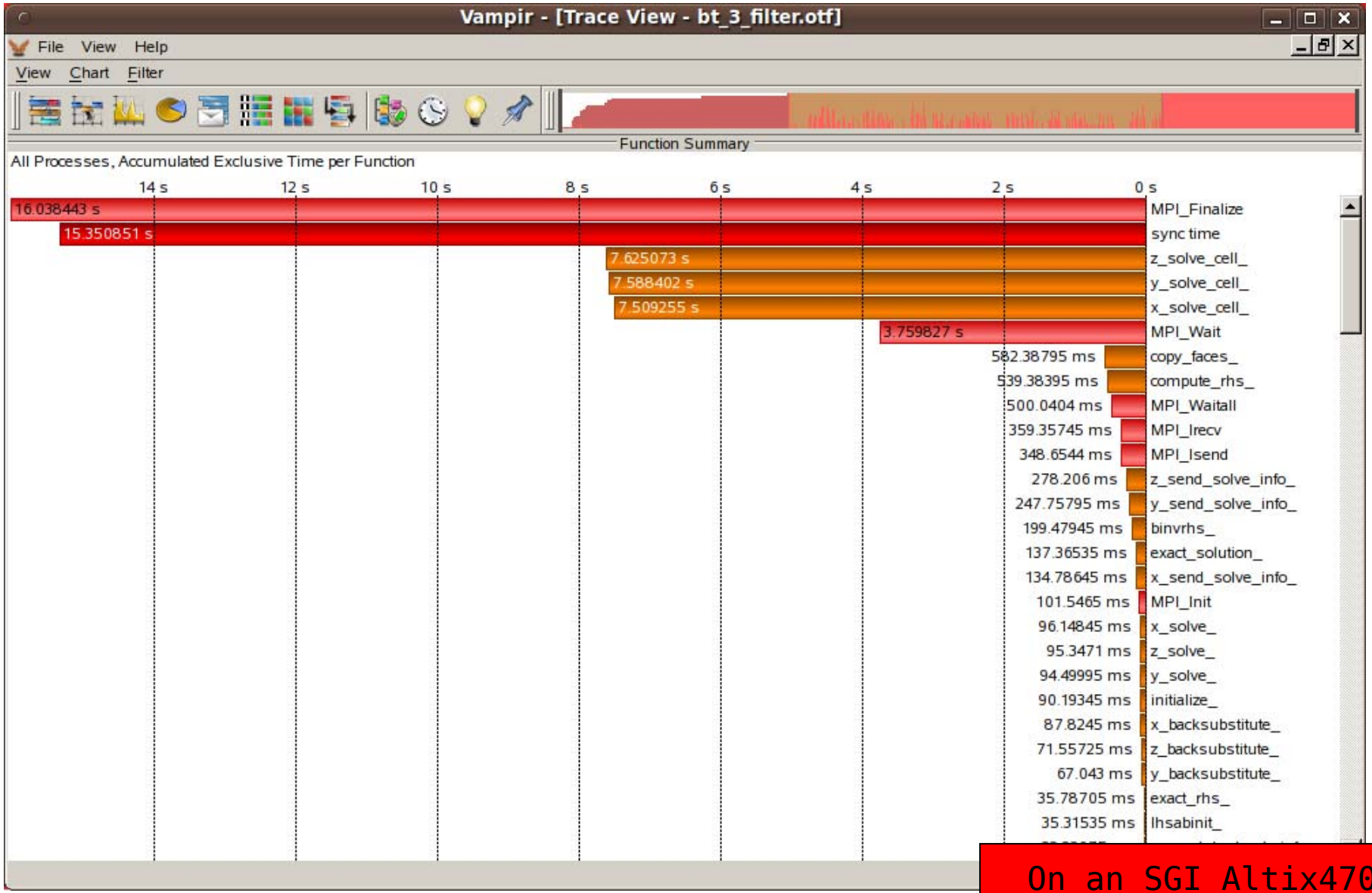
Remove the old trace first !

Hands-on: NPB 3.3 BT-MPI



On an SGI Altix4700

Hands-on: NPB 3.3 BT-MPI



On an SGI Altix4700

- PAPI counters can be included in traces
  - If VampirTrace was build with PAPI support
  - If PAPI is available on the platform
- VT_METRICS specifies a list of PAPI counters

```
% export VT_METRICS = PAPI_FP_OPS:PAPI_L2_TCM
```

- see also the PAPI commands papi_avail and papi_command_line

Memory Allocation and I/O counters

- Memory allocation counters can be recorded:
  - If VampirTrace build with memory allocation tracing support
  - If GNU glibc is used on the platform
- intercept glibc functions like “malloc” and “free”
- Environment variable VT_MEMTRACE

```
% export VT_MEMTRACE = yes
```

- I/O counters can be included in traces
  - If VampirTrace was build with I/O tracing support
- Standard I/O calls like “open” and “read” are recorded
- Environment variable VT_IOTRACE

```
% export VT_IOTRACE = yes
```

- Record PAPI hardware counters

```
% papi_avail
% papi_event_chooser PRESET PAPI_FP_OPS
% export VT_METRICS=PAPI_FP_OPS:PAPI_L2_TCM
```

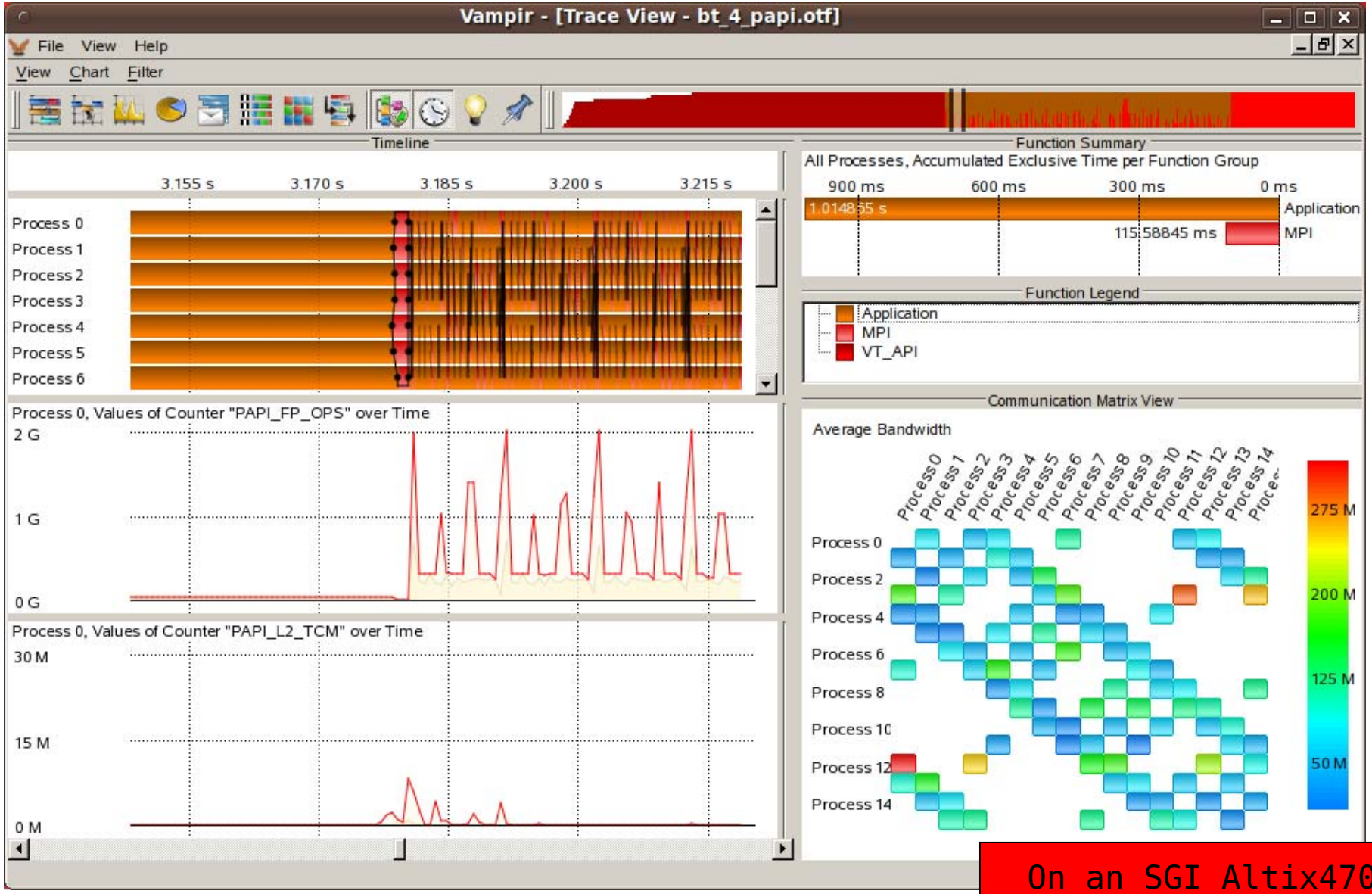
- Set a new file prefix

```
% export VT_FILE_PREFIX=bt_4_papi
```

- Launch as MPI application

```
% mpiexec -np 16 bt_W.16
```

Hands-on: NPB 3.3 BT-MPI



On an SGI Altix4700

VampirTrace Run-Time Options

- control options by environment variables:

– VT_PFORM_GDIR	Directory for final trace files
– VT_PFORM_LDIR	Directory for intermediate files
– VT_FILE_PREFIX	Trace file name
– VT_BUFFER_SIZE	Internal trace buffer size
– VT_MAX_FLUSHES	Max number of buffer flushes
– VT_MEMTRACE	Enable memory allocation tracing
– VT_MPICHECK	Enable MPI checking
– VT_IOTRACE	Enable I/O tracing
– VT_MPITRACE	Enable MPI tracing
– VT_FILTER_SPEC	Name of filter definition file
– VT_GROUPS_SPEC	Name of grouping definition file
– VT_METRICS	PAPI counter selection

Thanks for your attention.

VAMPIR & VAMPIRTRACE
Finding Performance Bottlenecks

- Trace Visualization
  - Vampir provides a number of display types
  - Each allows many different options
- Advice
  - 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

Finding Bottlenecks

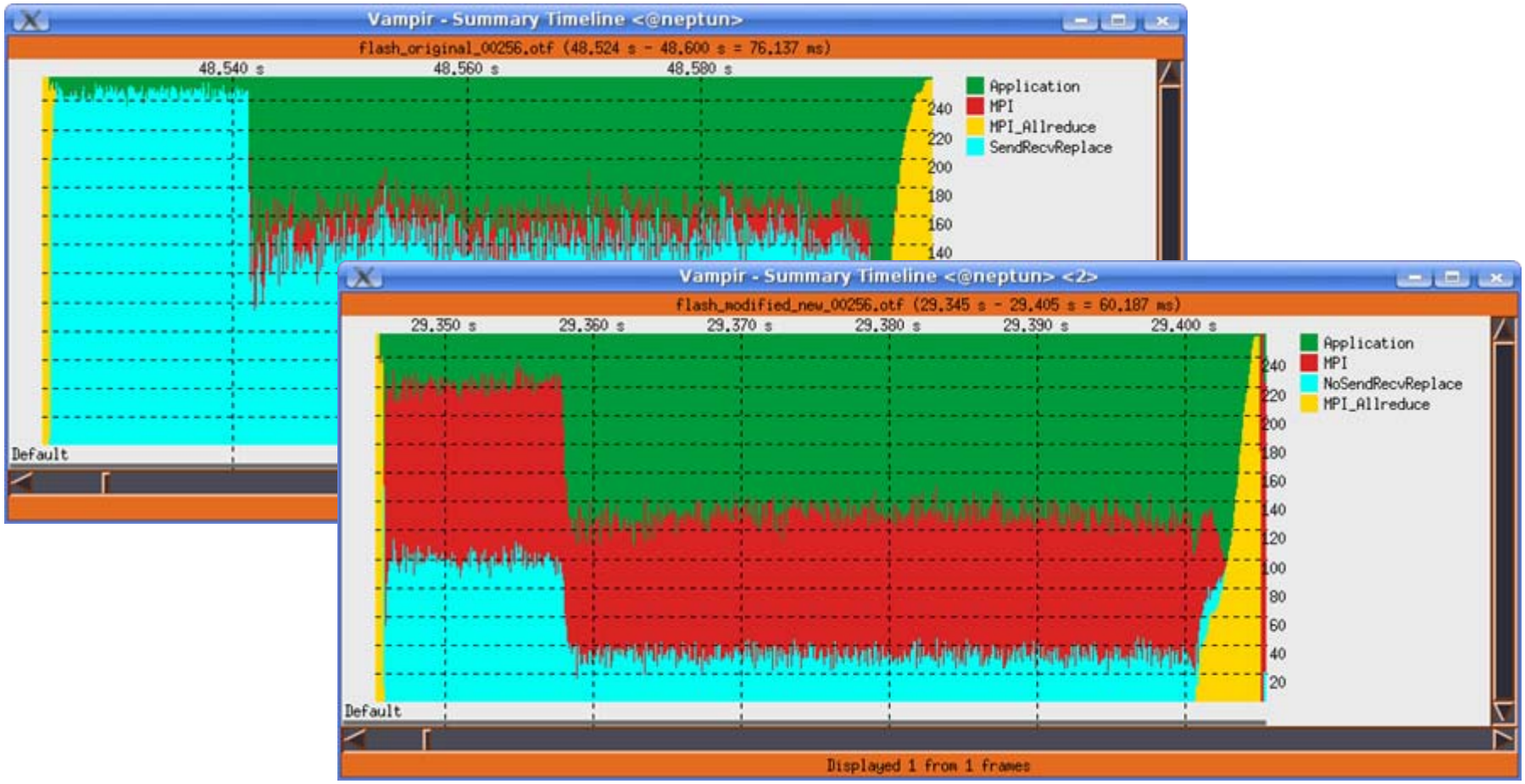
- Communication
- Computation
- Memory, I/O, etc.
- Tracing itself

Bottlenecks in Communication

- Communications 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)

All of the above usually result in high MPI time share

Bottlenecks in Communication

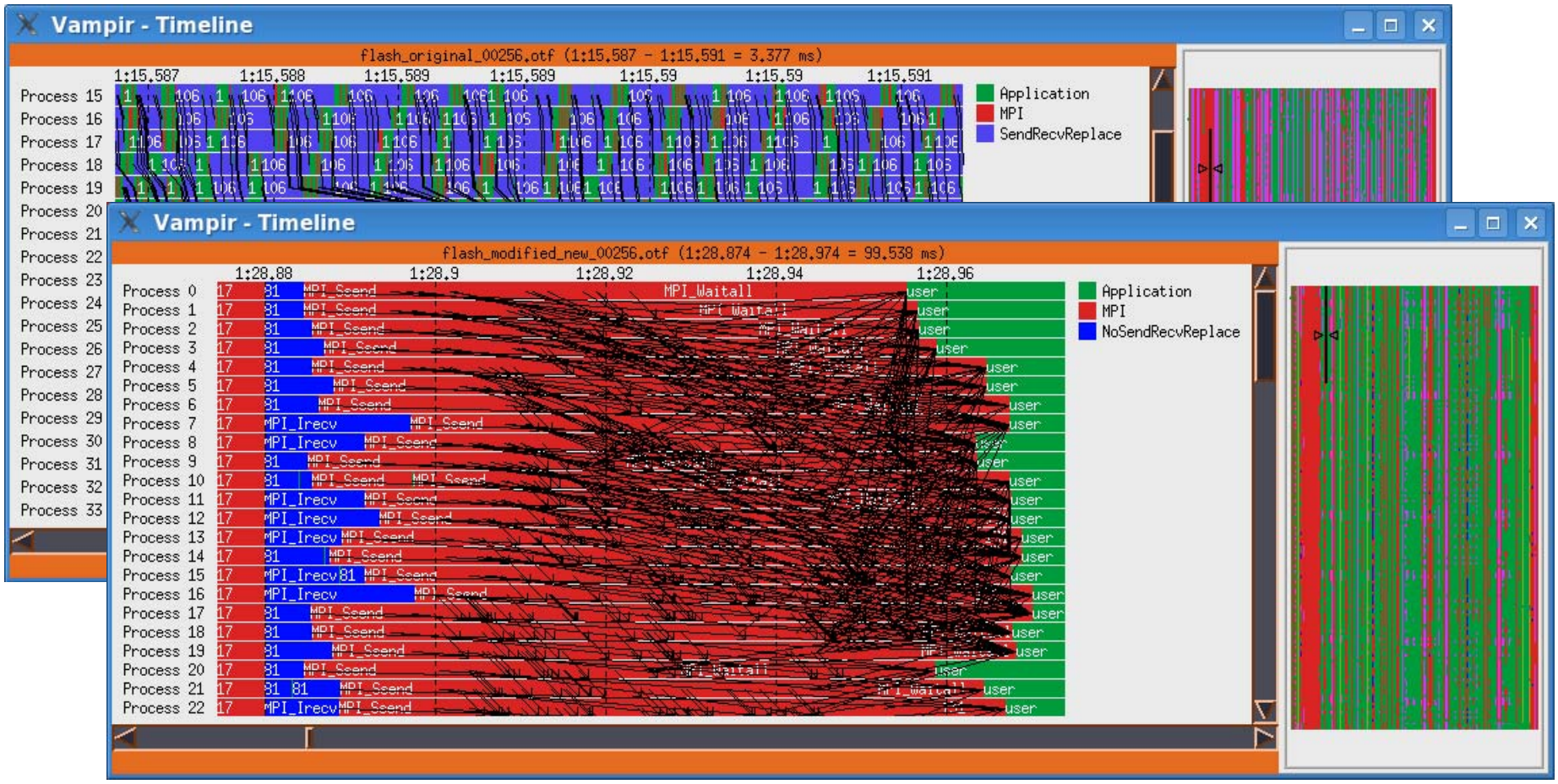


prevalent communication: MPI_Allreduce

Bottlenecks in Communication

prevalent communication: timeline view

Bottlenecks in Communication

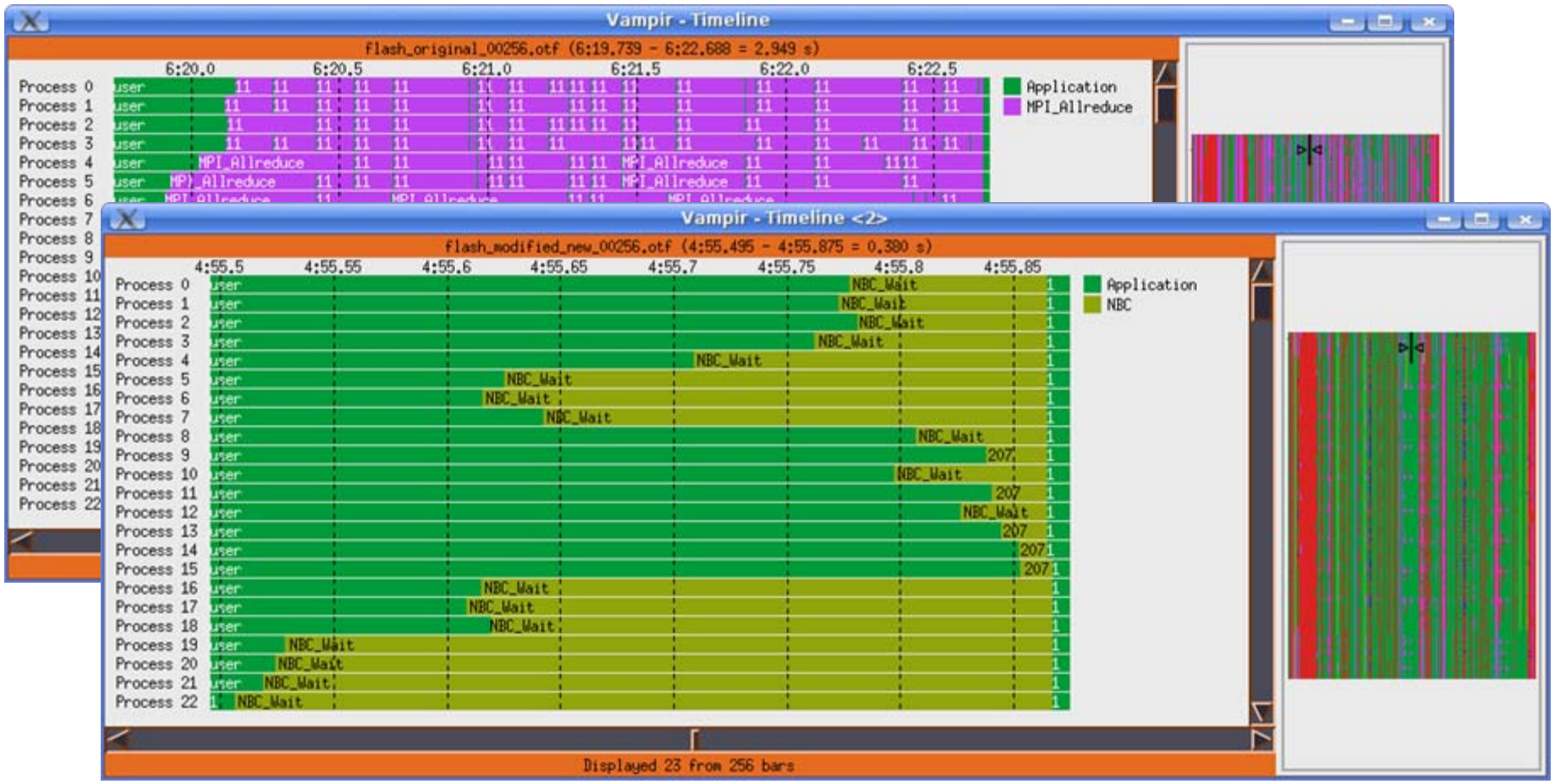


Propagated Delays in MPI_SendReceiveReplace

Bottlenecks in Communication

unnecessary MPI_Barriers

Bottlenecks in Communication

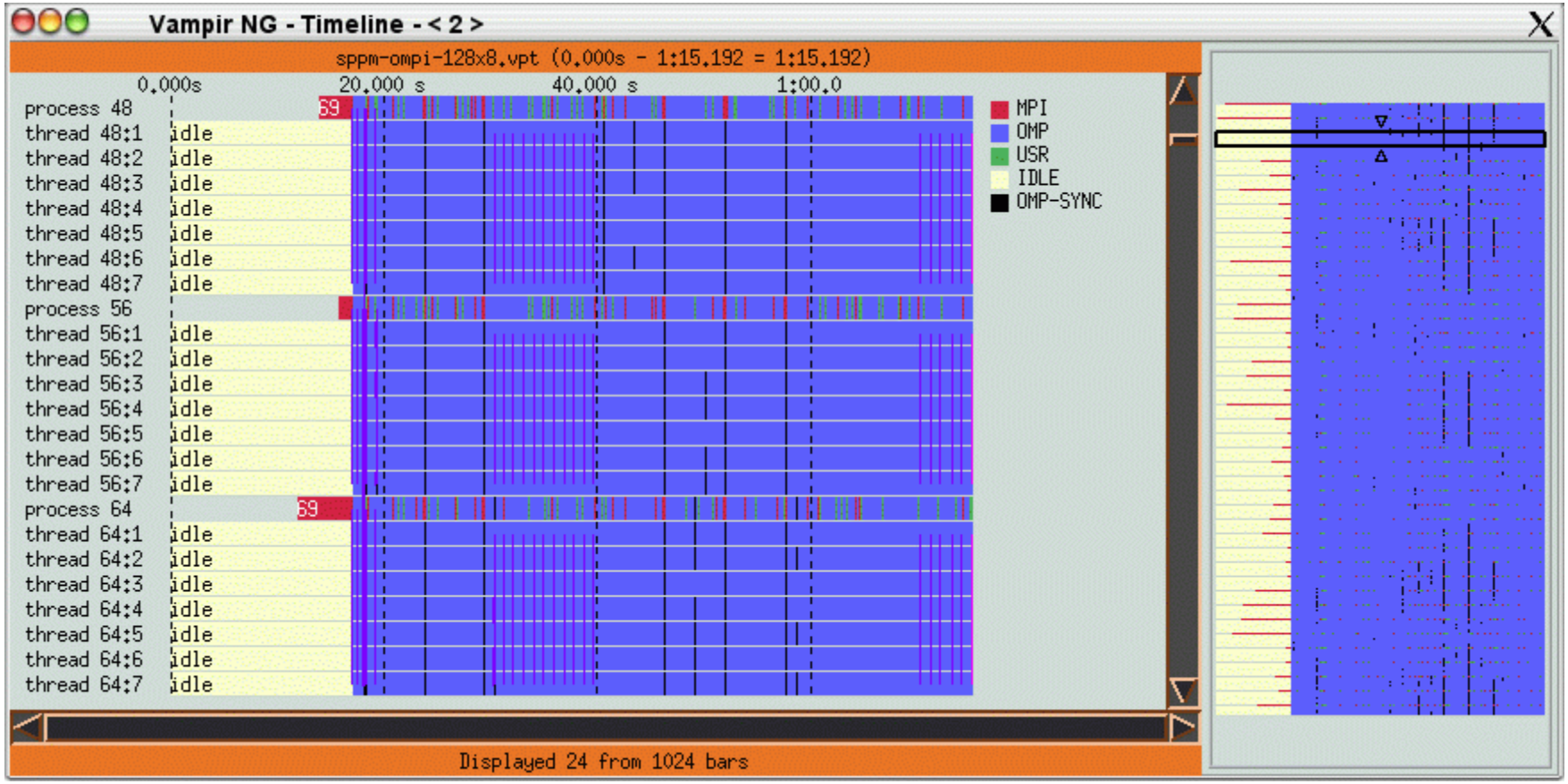


Patterns of successive MPI_Allreduce calls

Further Bottlenecks

- unbalanced computation
  - single late comer
- strictly serial parts of program
  - idle processes/threads
- very frequent tiny function calls
- sparse loops

Further Bottlenecks

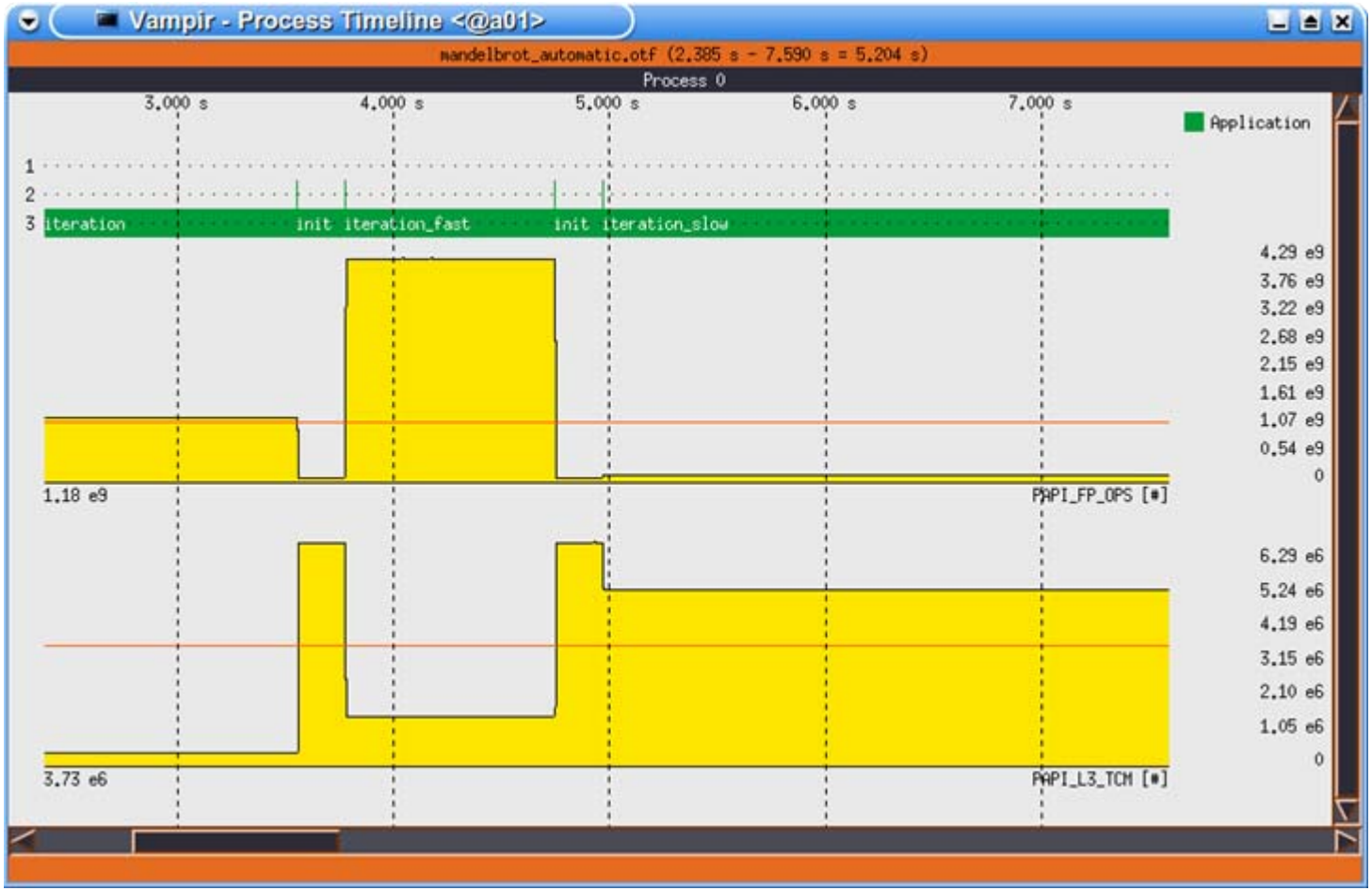


Example: Idle OpenMP threads

Bottlenecks in Computation

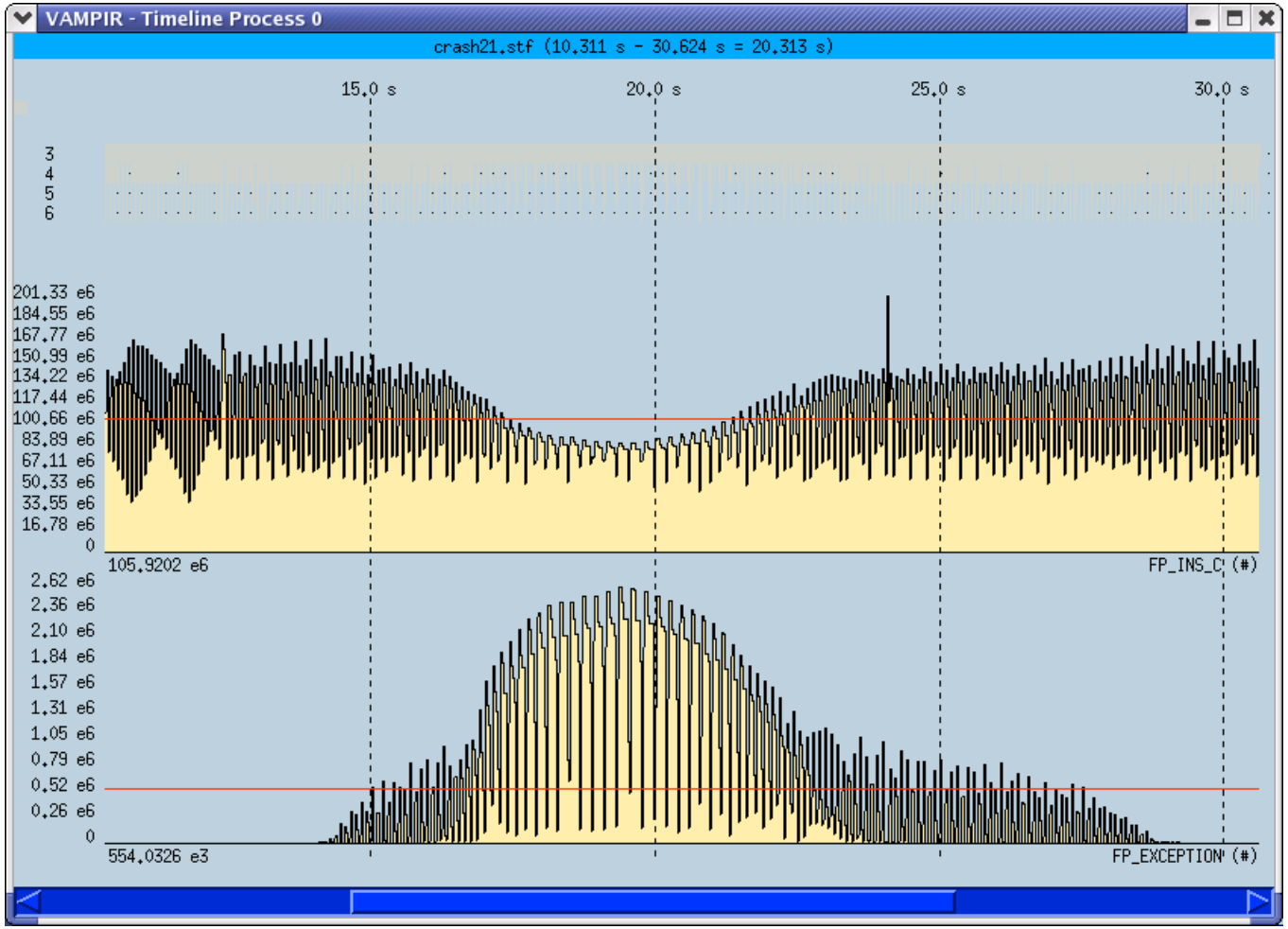
- memory bound computation
  - inefficient L1/L2/L3 cache usage
  - TLB misses
  - detectable via HW performance counters
- I/O bound computation
  - slow input/output
  - sequential I/O on single process
  - I/O load imbalance
- exception handling

Bottlenecks in Computation



low FP rate due to heavy cache misses

Bottlenecks in Computation



low FP rate due to heavy FP exceptions

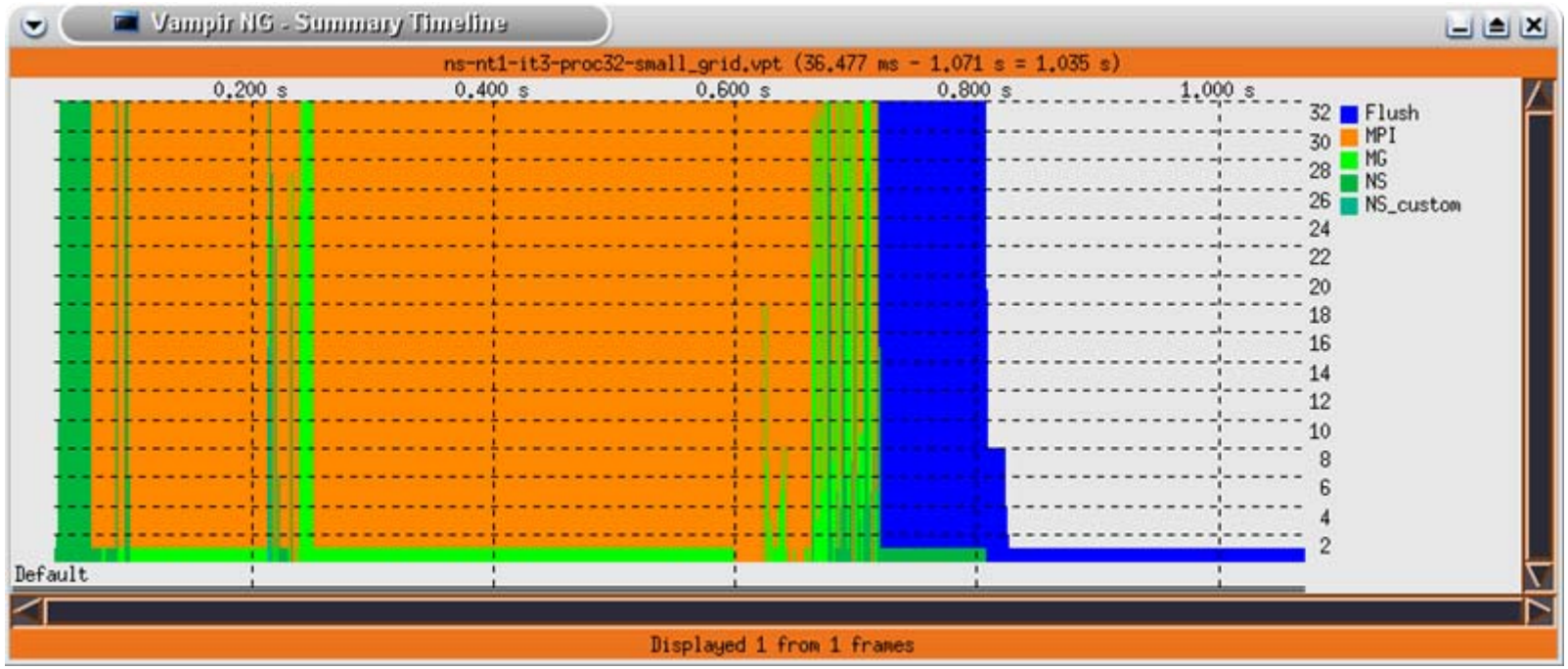
Bottlenecks in Computation

irregular slow I/O operations

Effects due to Tracing

- measurement overhead
  - especially grave for tiny function calls
  - solve with selective instrumentation
- long/frequent/asynchronous trace buffer flushes
- too man concurrent counters
- heisenbugs

Effects due to Tracing



Trace buffer flushes are explicitly marked in the trace.
It is rather harmless at the end of a trace as shown here.

Conclusions and Outlook

- performance analysis very important in HPC

- use performance analysis tools for profiling and tracing
- do not spend effort in DIY solutions,
  e.g. like printf-debugging

- use tracing tools with some precautions
  - overhead
  - data volume

- let us know about problems and about feature wishes
- vampirsupport@zih.tu-dresden.de

Vampir and VampirTraces are available at http://www.vampir.eu and http://www.tu-dresden.de/zih/vampirtrace/ ,

get support via vampirsupport@zih.tu-dresden.de

Acknowledgement:

Staff at ZIH - TU Dresden:

Ronny Brendel, Holger Brunst, Jens Doleschal, Ronald Geisler, Daniel Hackenberg, Michael Heyde, Tobias Hilbrich, Rene Jäkel, Matthias Jurenz, Michael Kluge, Andreas Knüpfer, Matthias Lieber, Holger Mickler, Hartmut Mix, Matthias Müller, Wolfgang E. Nagel, Reinhard Neumann, Michael Peter, Heide Rohling, Johannes Spazier, Michael Wagner, Matthias Weber, Bert Wesarg

Wrapper Functions

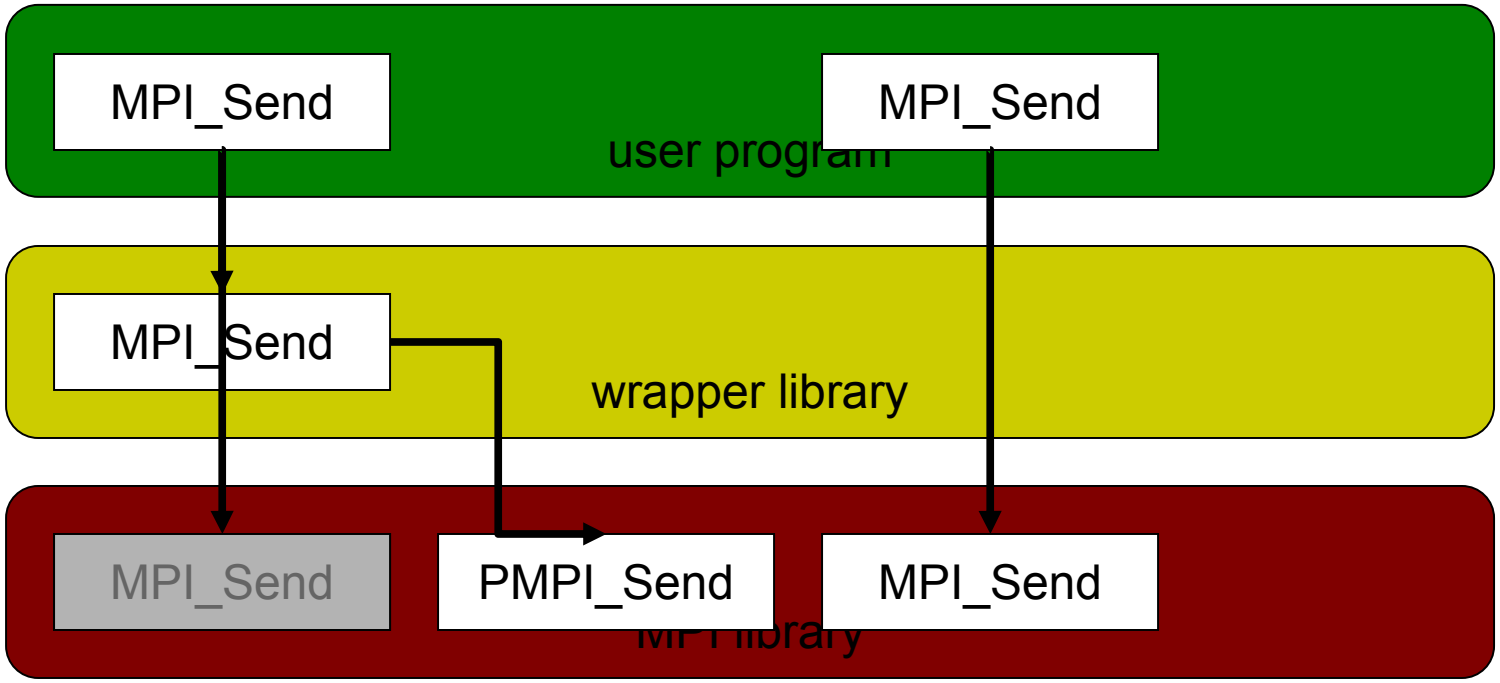
- provide wrapper functions
  - call instrumentation function for notification
  - call original target for functionality
  - via preprocessor directives:

```
#define MPI_Init WRAPPER_MPI_Init

#define MPI_Send WRAPPER_MPI_Send
```

- via library preload:
  - preload instrumented dynamic library
- suitable for standard libraries (e.g. MPI, glibc)

- Each MPI function has to names:
  - MPI_xxx and PMPI_xxx
- Replacement of MPI routines at link time



gcc –finstrument-functions –c foo.c

```
void __cyg_profile_func_enter( <args> );
void __cyg_profile_func_exit( <args> );
```

- many compilers support this: GCC, Intel, IBM, PGI, NEC, Hitachi, Sun Fortran, …
- no source code modification necessary

Dynamic Instrumentation

- modify executable in file or binary in memory
- insert instrumentation calls
- very platform/machine dependent, expensive

- DynInst project (http://www.dyninst.org)
  - common interface
  - supported platforms: Alpha/Tru64, MIPS/IRIX, PowerPC/AIX, Sparc/Solaris, x86/Linux x86/Windows, ia64/Linux