VAMPIR & VAMPIRTRACE DETAILS AND HANDS-ON

Matthias Weber,
Jens Doleschal, Andreas Knüpfer
ZIH, TU Dresden
matthias.weber@tu-dresden.de

November 2010

Overview

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

VAMPIR & VAMPIRTRACE
Event Tracing in General

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

- 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, …

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.

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: 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)

```
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

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

CC=icc	CC=vtcc
CXX=icpc	CXX=vtcxx
F90=ifc	F90=vtf90
MPICC=mpicc	MPICC=vtcc -vt:cc mpicc

- Re-compile & re-link
- Trace run
  - User representative test input
  - Set parameters, environment variables, etc.
  - Perform trace run
- Get Trace

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

Hands-on: NPB 3.3 BT-MPI

- Move into tutorial directory in your home directory

```
% cd tutorial
```

- 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 89, resulting in:
            ...
            89: #MPICC = mpicc
            ...
    -> remove the comment from line 95, resulting in:
            ...
            95: MPICC = vtcc  -vt:cc mpicc
            ...
```

Hands-on: NPB 3.3 BT-MPI – Step 1

- 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  60
[0]VampirTrace: Maximum number of buffer flushes reached \
(VT_MAX_FLUSHES=1)
[0]VampirTrace: Tracing switched off permanently
 Time step  200
 ...
```

Hands-on: NPB 3.3 BT-MPI – Step 1

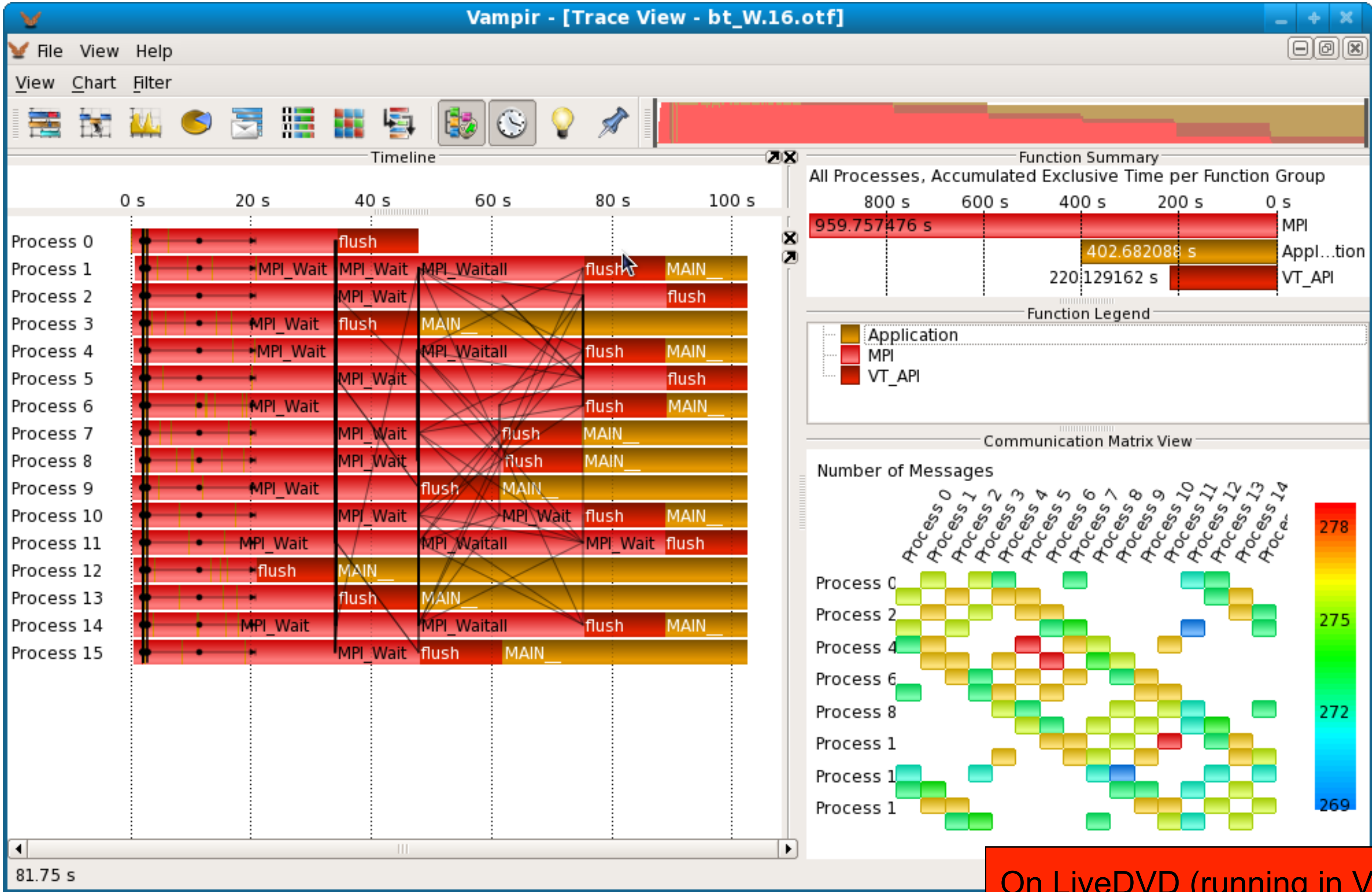
- Resulting trace files

```
% ls -alh
4,1M bt_1_initial.16
3,6K bt_1_initial.16.0.def.z
3.8M bt_1_initial.16.0.marker.z
3.8M bt_1_initial.16.10.events.z
3.8M bt_1_initial.16.1.events.z
3.8M bt_1_initial.16.2.events.z
3.8M bt_1_initial.16.3.events.z
...
3.8M bt_1_initial.16.c.events.z
3.8M bt_1_initial.16.d.events.z
3.8M bt_1_initial.16.e.events.z
3.8M bt_1_initial.16.f.events.z
66 bt_1_initial.16.otf
```

- Visualization with Vampir7

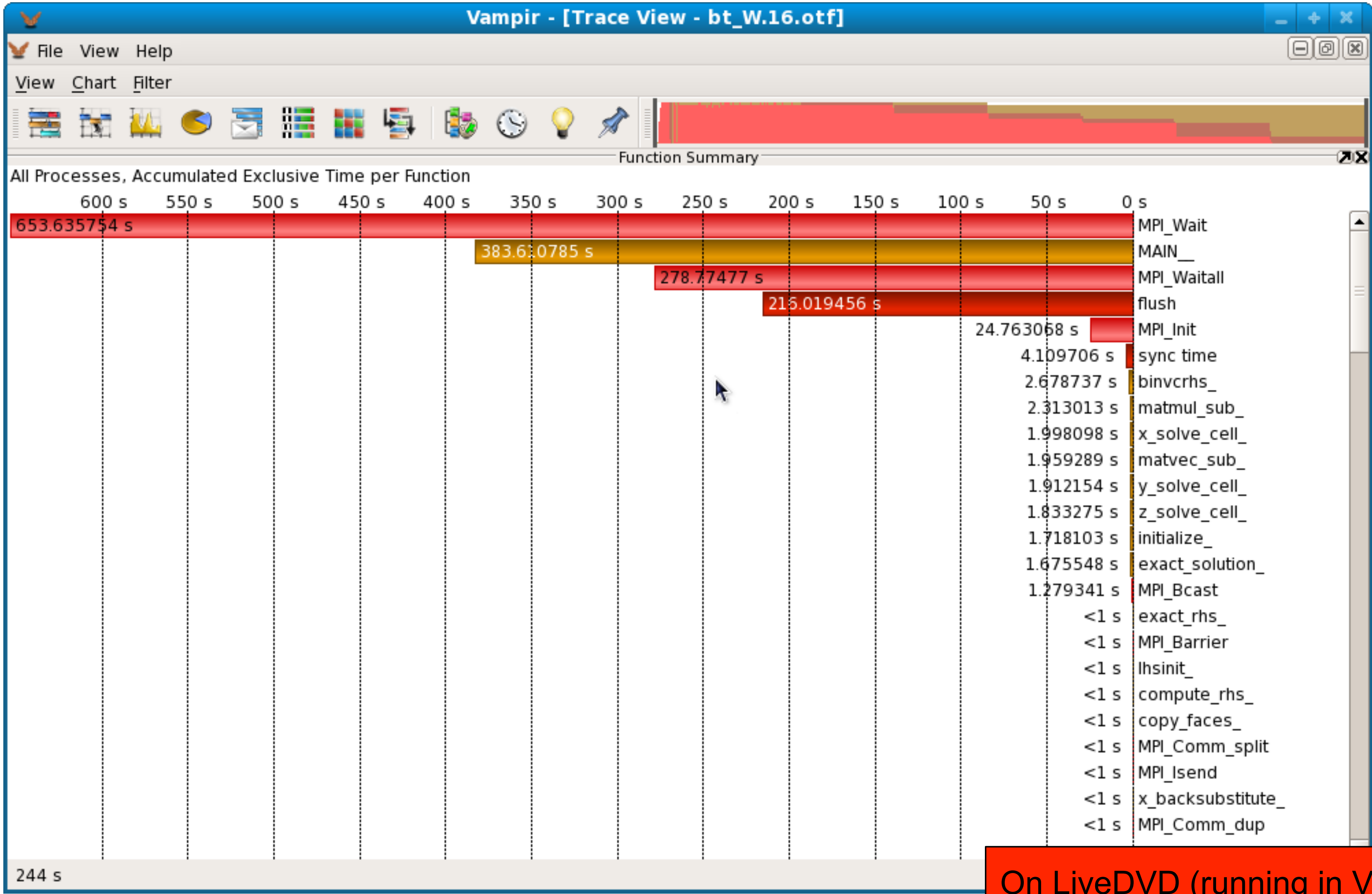
```
% vampir bt_1_initial.16.otf
```

Hands-on: NPB 3.3 BT-MPI – Step 1



On LiveDVD (running in VM)

Hands-on: NPB 3.3 BT-MPI – Step 1



On LiveDVD (running in VM)

Issues of Hands-on – Step 1

Issue:

Tracing was switched off because the internal trace buffer was too small

Result:

1. Asynchronous behavior of the application due to buffer flush of the measurement system
2. No tracing information available after flush operation

Solutions:

1. Increase trace buffer size
2. Increase number of allowed buffer flushes (not recommended)
3. Use filter mechanisms to reduce the number of recorded events
4. Switch tracing on/off if your application works in an iterative manner to reduce the number of recorded events (see the VampirTrace manual for more details)

Hands-on: NPB 3.3 BT-MPI – Step 2

- 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
```

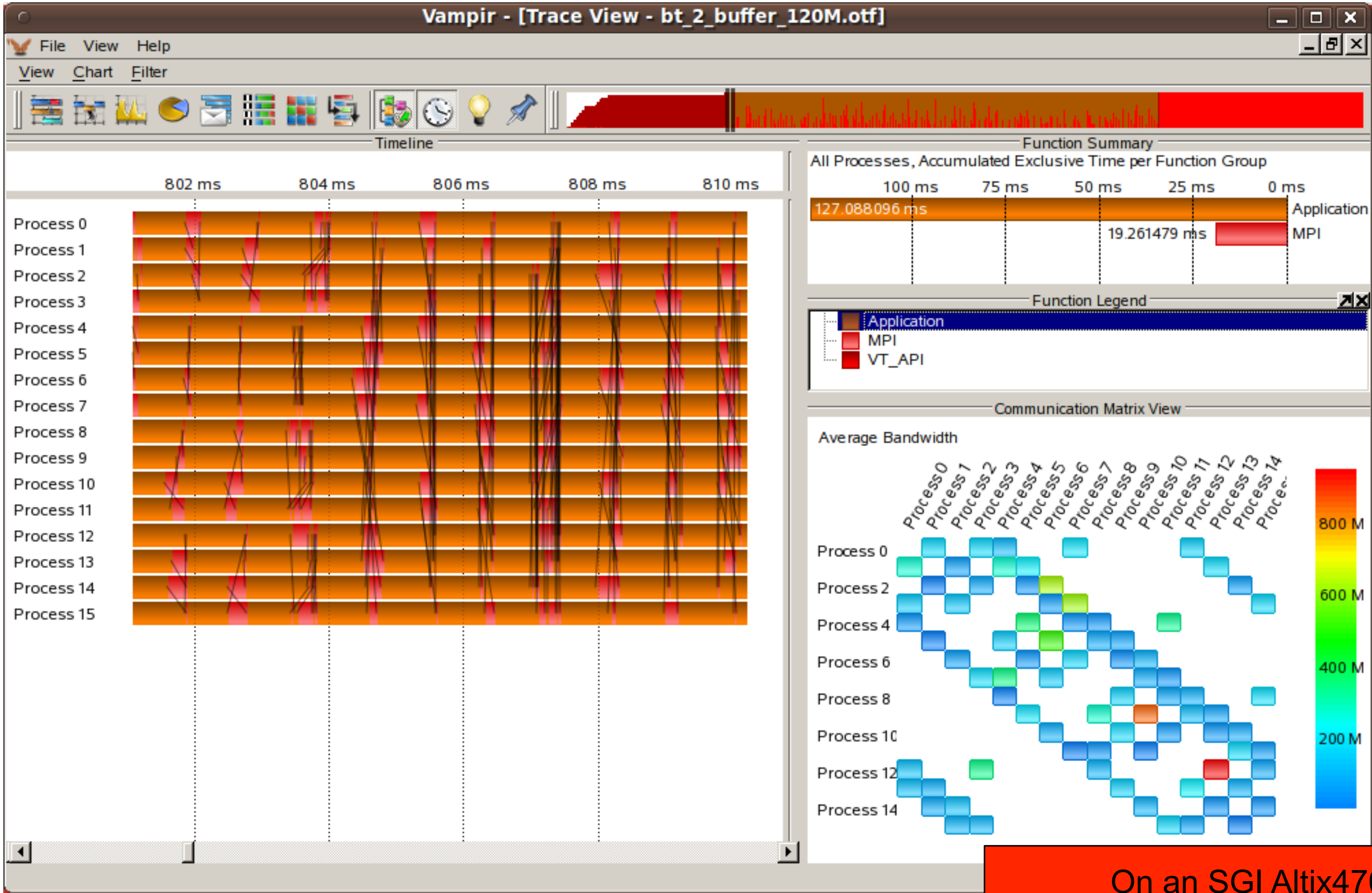
Remove the old trace first !

- Visualization with Vampir 7

```
% vampir bt_2_buffer.16.otf
```

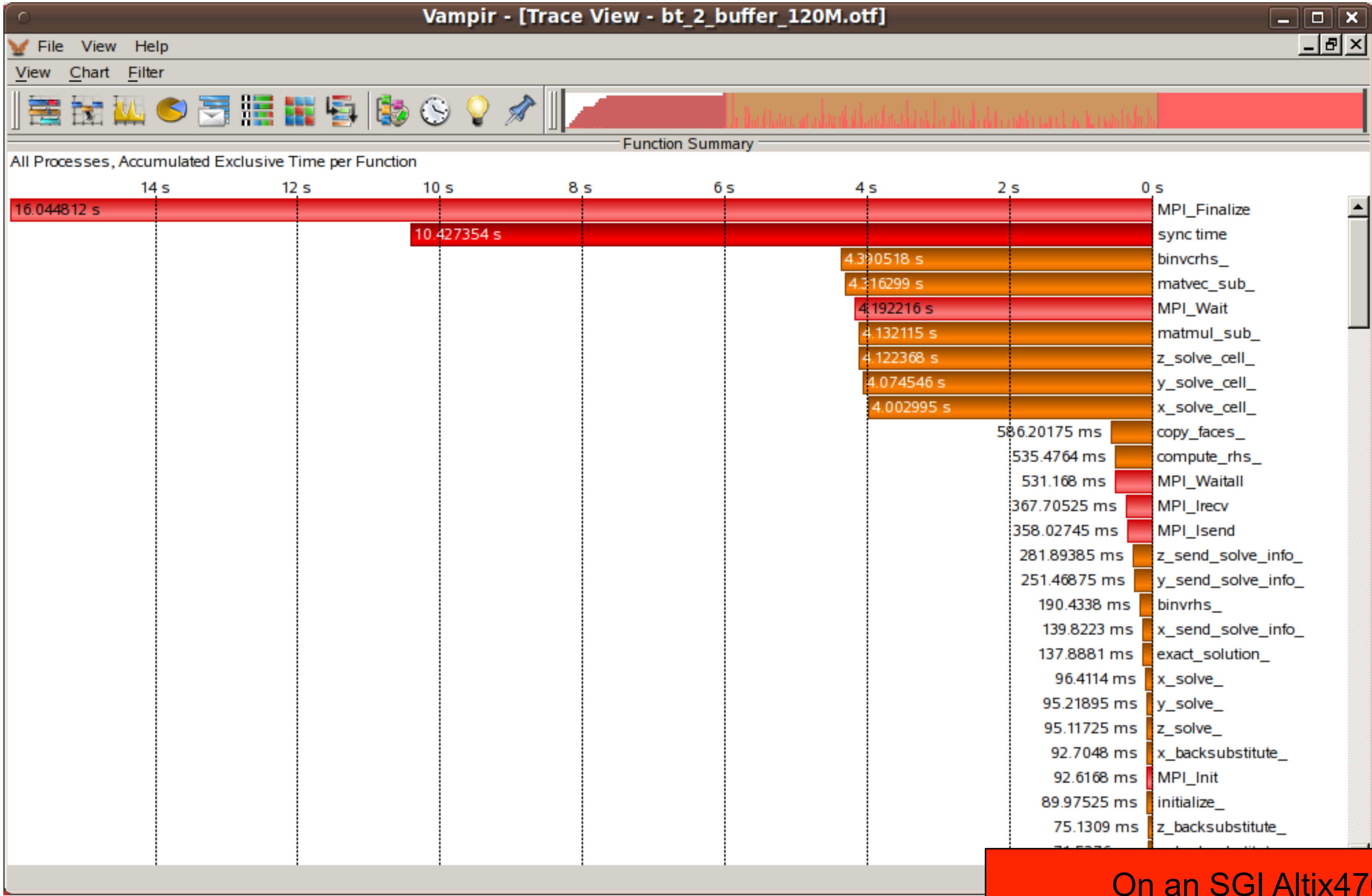
Only for laptops with at least 2GB main memory !

Hands-on: NPB 3.3 BT-MPI – Step 2



On an SGI Altix4700

Hands-on: NPB 3.3 BT-MPI – Step 2



On an SGI Altix4700

Issues of Increasing the Buffer Size – Step 2

VI-HPS

Issue:

Each function entry/exit, MPI event was recorded

Result:

Trace file becomes large even for short application runs
and may not fit into the main memory

Solutions:

1. Use filter mechanisms to reduce the number of recorded events
2. Switch tracing on/off if your application works in an iterative manner
to reduce the number of recorded events
(see the VampirTrace manual for more details)

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

Switch Tracing On/Off

- Starting and stopping of tracing should be performed with care
- Tracing has to be activated on the same call stack level as it was switched off to ensure the consistency of the trace file
- Useful if your program behaves in an iterative manner or if you are only interested in some parts of your application

```
#include “vt_user.h”
…
VT_OFF();
for( i=1; i < 100; i++ ) { do something uninteresting };
VT_ON();
…
```

- Recompile your source code with the user macro “-DVTRACE”

```
% vtcc … -DVTRACE source_code.c …
```

- Generate your filter specification and set environment

```
% gedit filter.txt
    binvcrhs*; matvec_sub*; matmul_sub* -- 0

% export VT_FILTER_SPEC=filter.txt
```

- Set a new file prefix

```
% export VT_FILE_PREFIX=bt_3_filter
```

- Launch as MPI application

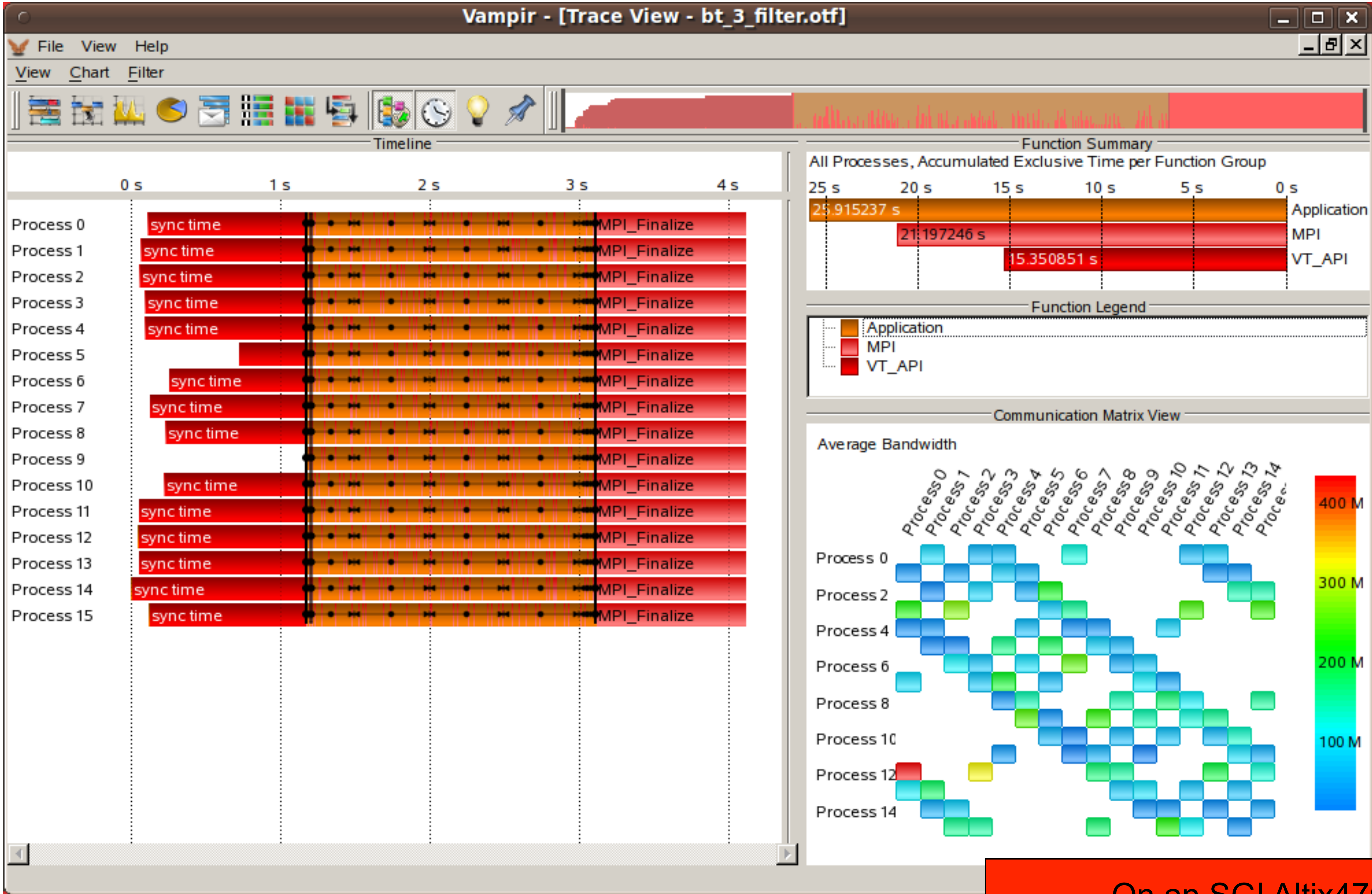
```
% mpiexec -np 16 bt_W.16
```

Remove the old trace first !

- Visualization with Vampir 7

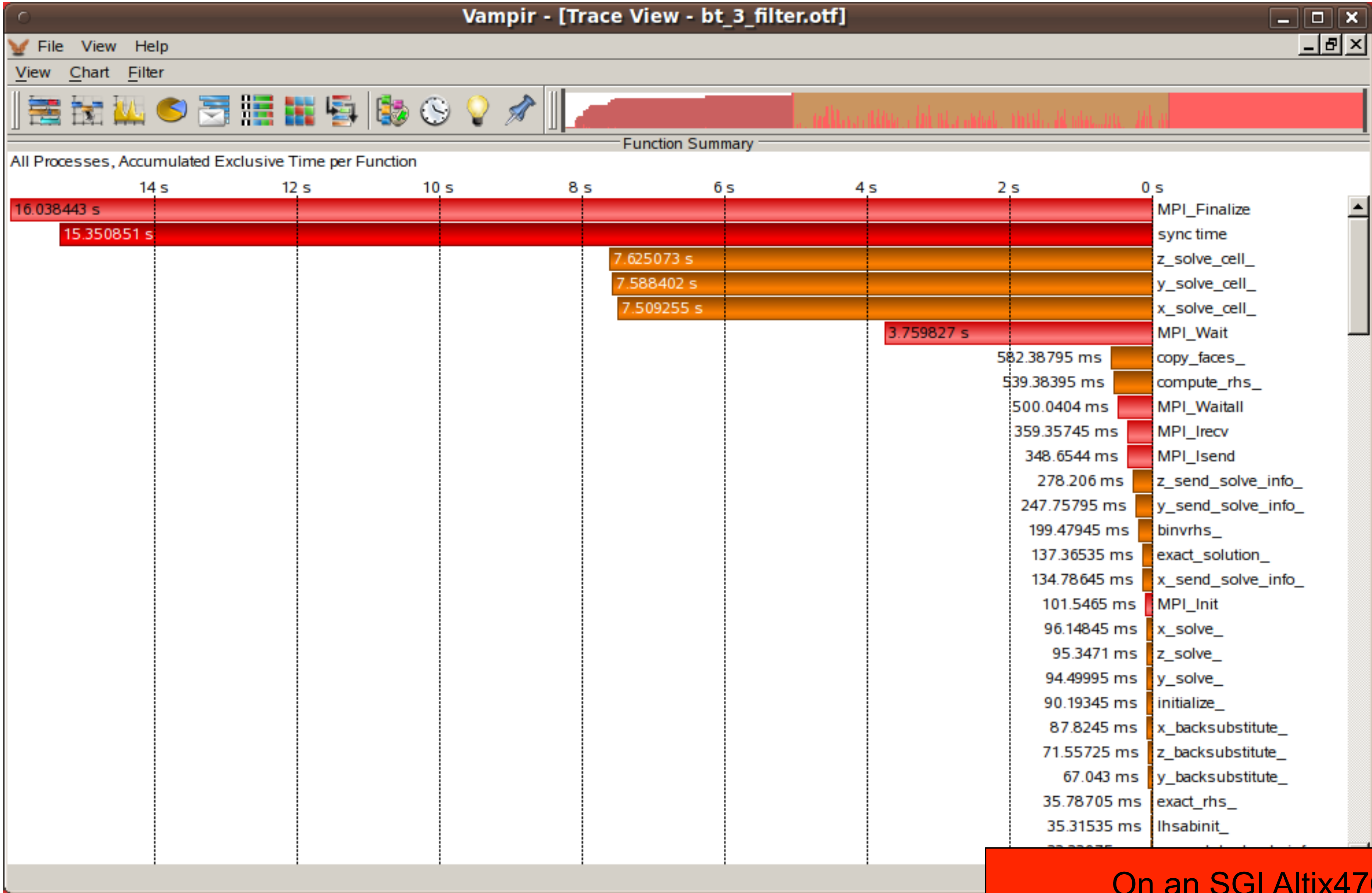
```
% vampir bt_3_filter.16.otf
```

Hands-on: NPB 3.3 BT-MPI – Step 3



On an SGI Altix4700

Hands-on: NPB 3.3 BT-MPI – Step 3



On an SGI Altix4700

Issues of VampirTrace Hands-on – Step 3

Issue:

Runtime filtering will be called for every event

Result:

Runtime filtering may increases the runtime overhead

Solutions:

1. Use manual source instrumentation (high effort, not recommended)
2. Only instrument interesting source files with VampirTrace
3. Switch tracing on/off if your application works in an iterative manner to reduce the number of recorded events (see the VampirTrace manual for more details)

However, these trace files include no information about the computational performance of your application. Therefore, in the **next step**:

Recording of hardware performance counters

PAPI

- 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
```

Hands-on: NPB 3.3 BT-MPI – Step 4

- 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

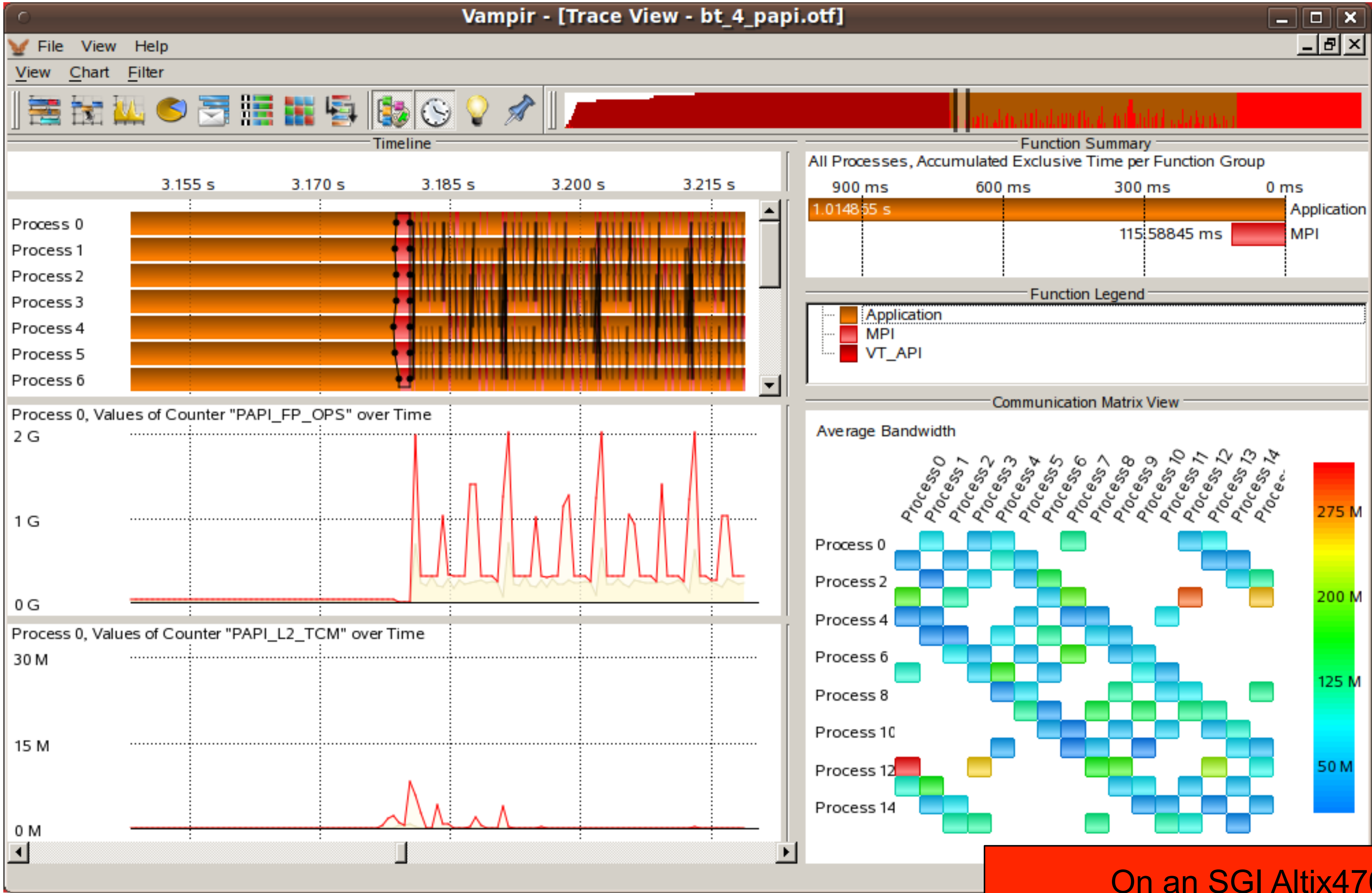
```
% mpiexec -np 16 bt_W.16
```

Remove the old trace first !

- Visualization with Vampir 7

```
% vampir bt_4_papi.16.otf
```

Hands-on: NPB 3.3 BT-MPI – Step 4



On an SGI Altix4700

Vampir Trace Files: NPB 3.3 BT-MPI and further examples

- All NPB 3.3 BT-MPI trace files of a hands-on session are located at:

```
% cd $HOME/workshop-vampirtrace/Examples/npb-bt-mpi/result_thinkpad
```

- All NPB 3.3 BT-MPI trace files created on a SGI-Altix are located at:

```
% cd $HOME/workshop-vampirtrace/Examples/npb-bt-mpi/result_altix
```

- SMG 2000 trace files with various configurations are located at:

```
% cd $HOME/workshop-vampirtrace/Examples/smg2000/
```

- Mandelbrot trace files can be found at:

```
% cd $HOME/workshop-vampirtrace/Examples/mandelbrot
```

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=*
```

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

Finding 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

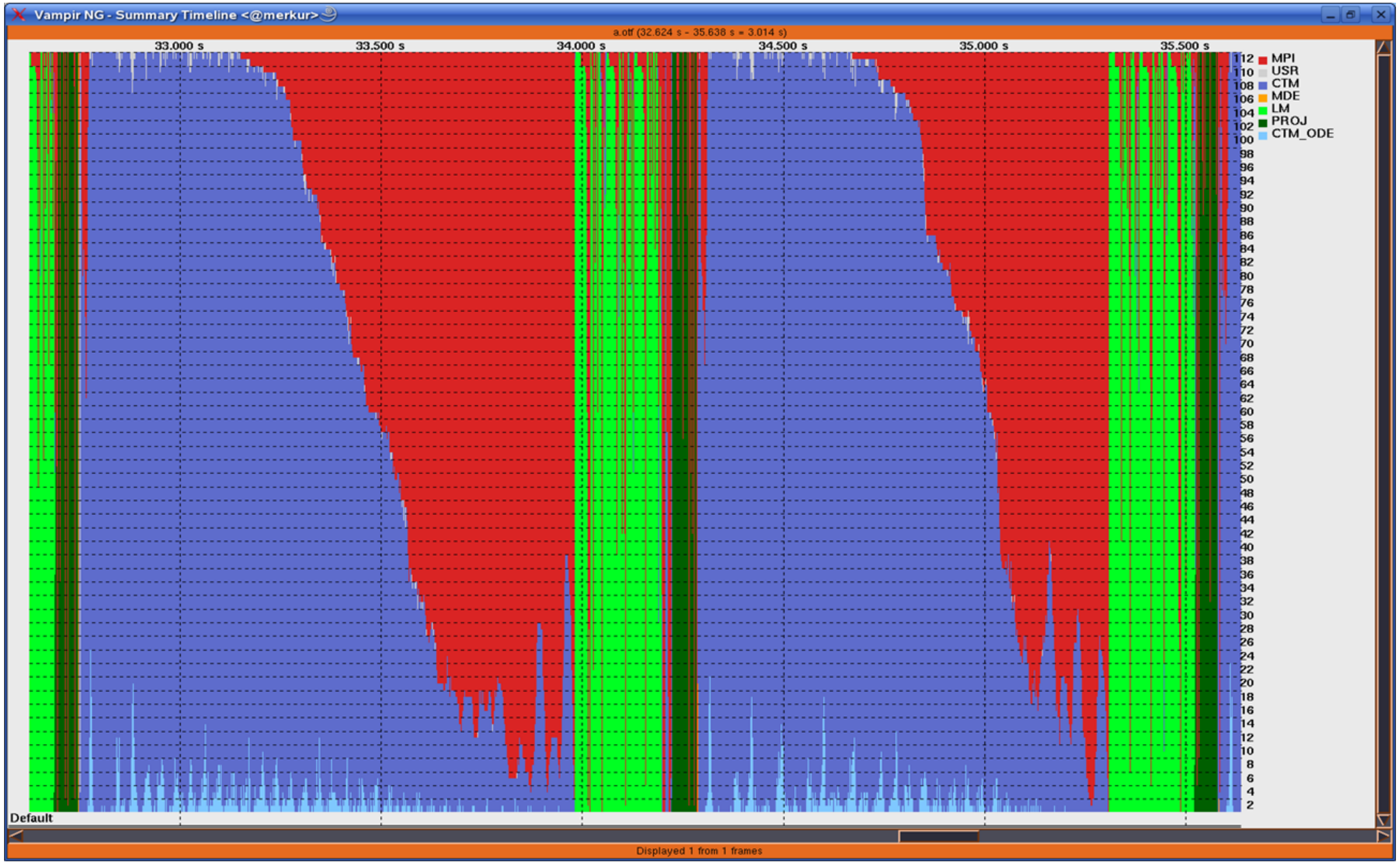
- Communication
- Computation
- Memory, I/O, etc.
- Tracing itself

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)

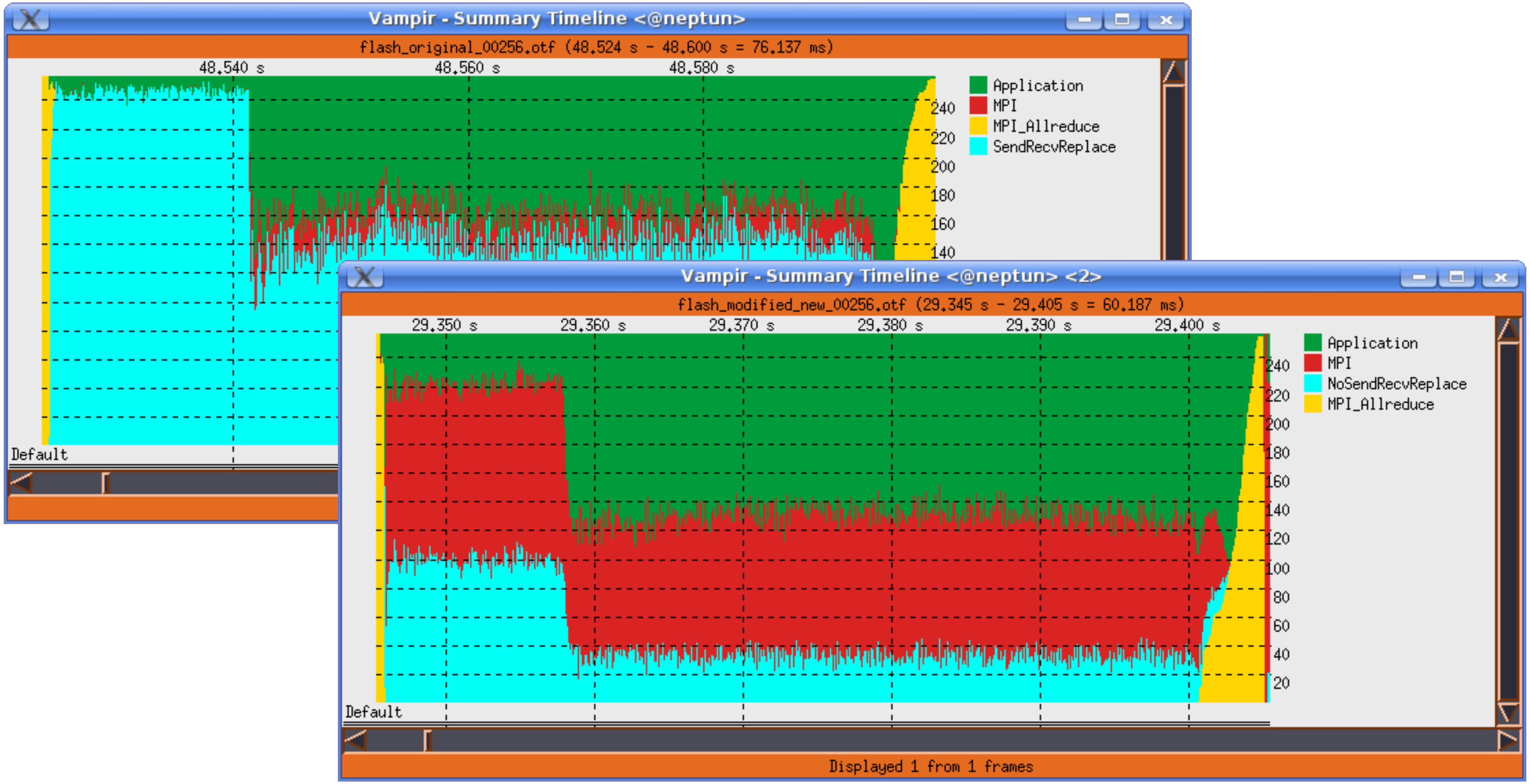
All of the above usually result in high MPI time share

Bottlenecks in Communication



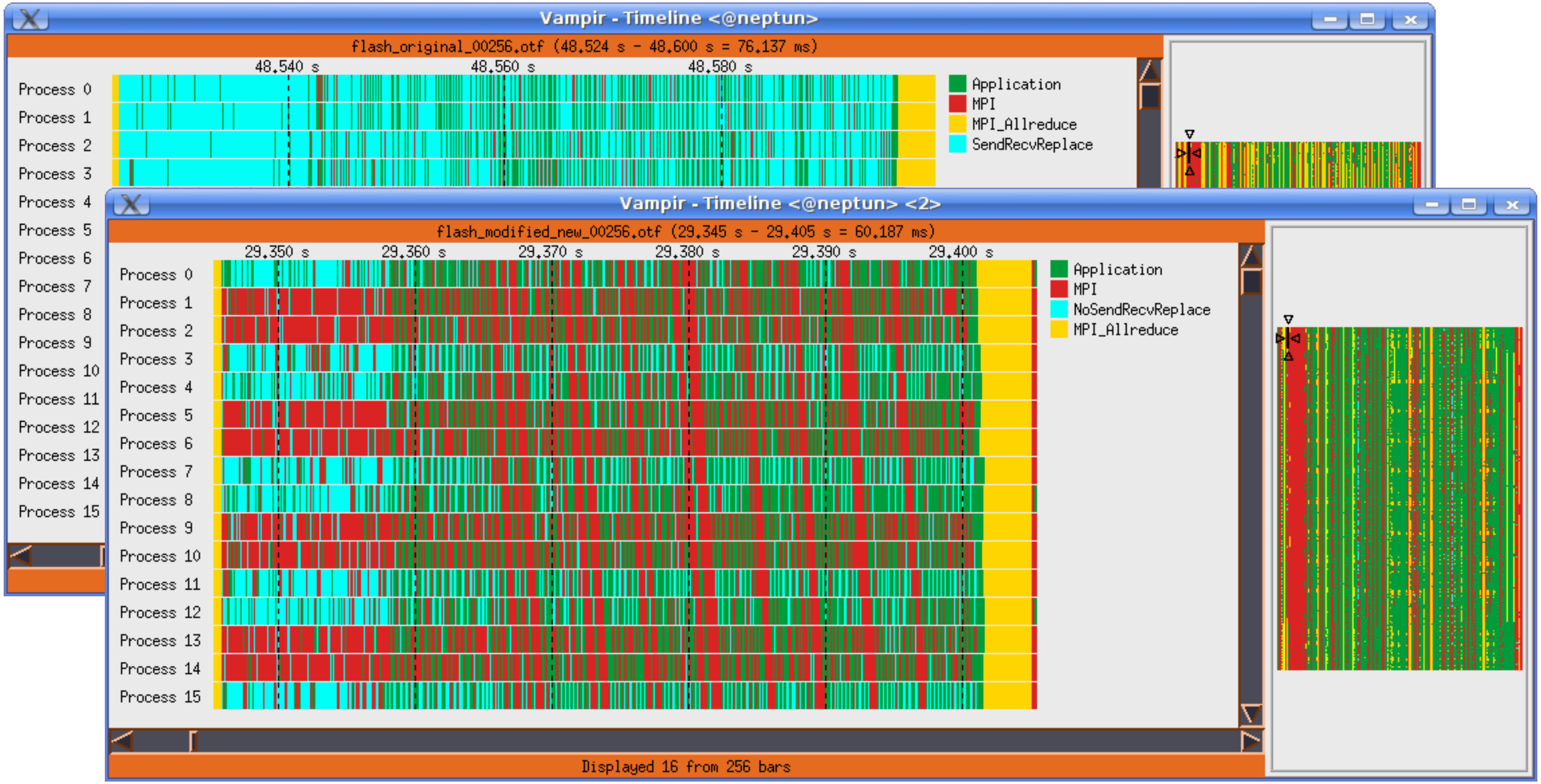
Example: prevalent communication

Bottlenecks in Communication



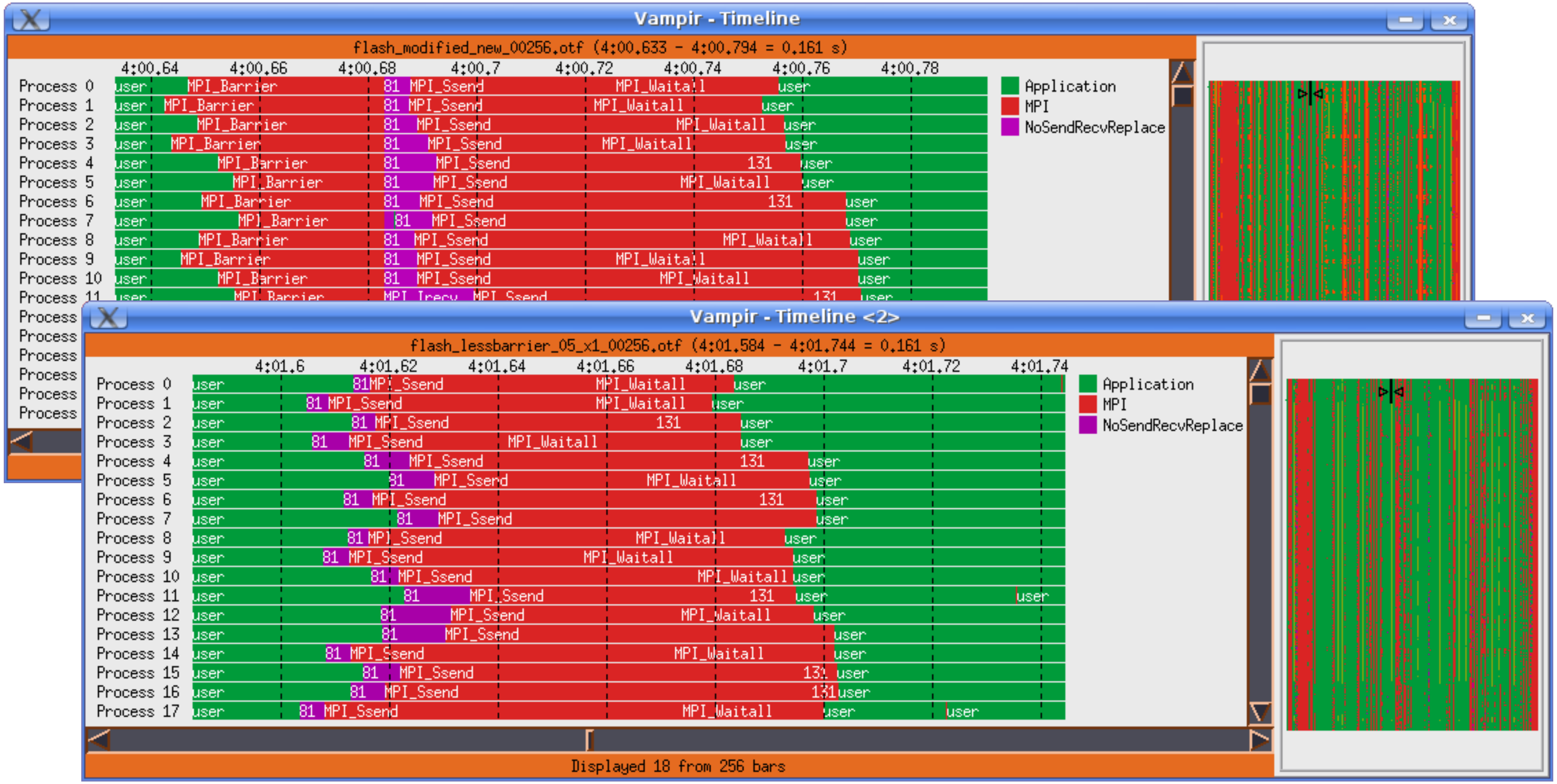
prevalent communication: MPI_Allreduce

Bottlenecks in Communication



prevalent communication: timeline view

Bottlenecks in Communication



unnecessary MPI_Barriers

Further Bottlenecks

- unbalanced computation
  - single late comer

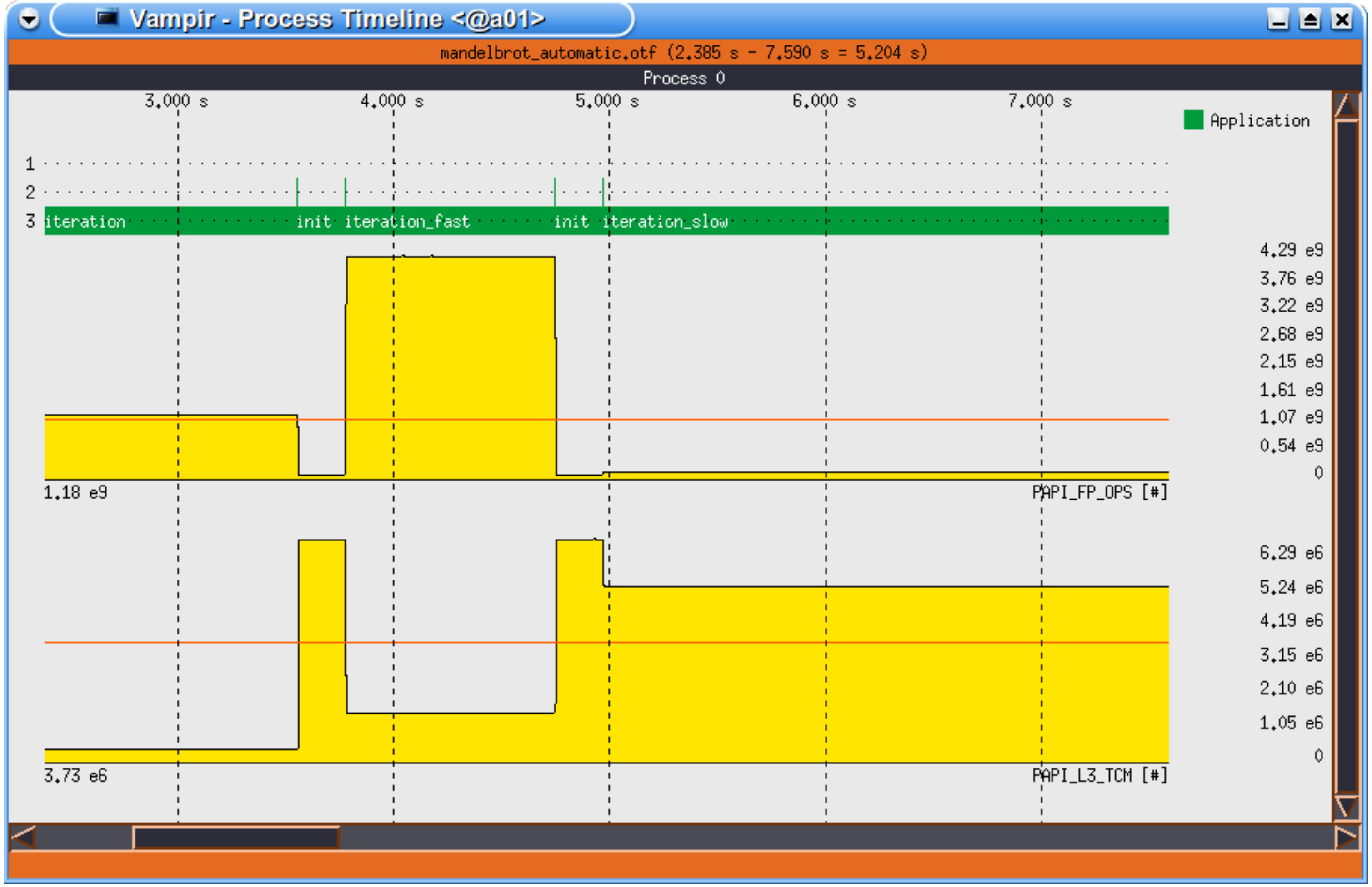
- strictly serial parts of program
  - idle processes/threads

- very frequent tiny function calls
- sparse loops

Bottlenecks in Computation

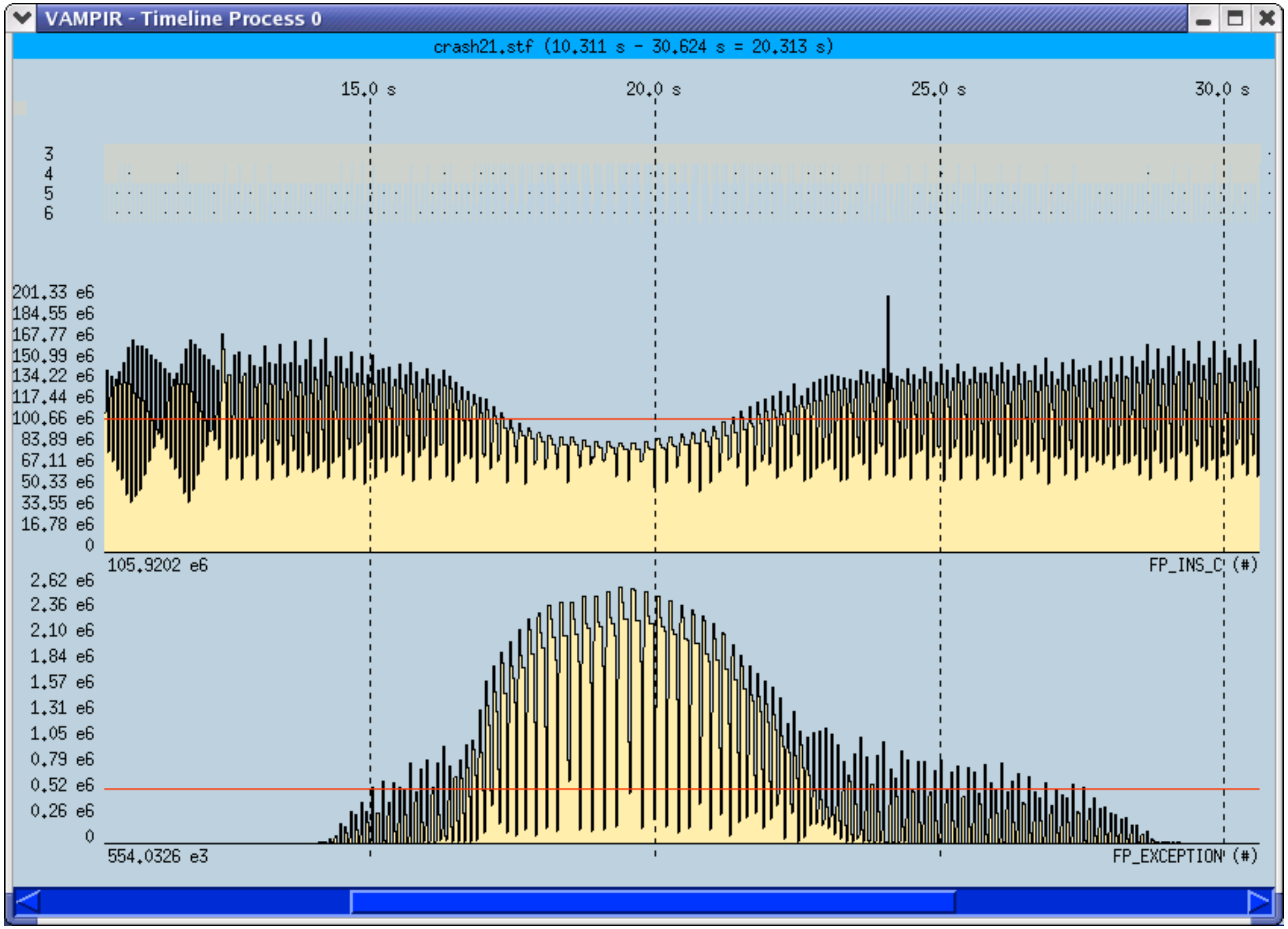
- memory bound computation
  - inefficient L1/L2/L3 cache usage
  - TLB misses
  - detectable via HW performance counters
- I/O bound computation
  - slow input/output
  - sequential I/O on single process
  - I/O load imbalance
- exception handling

Bottlenecks in Computation

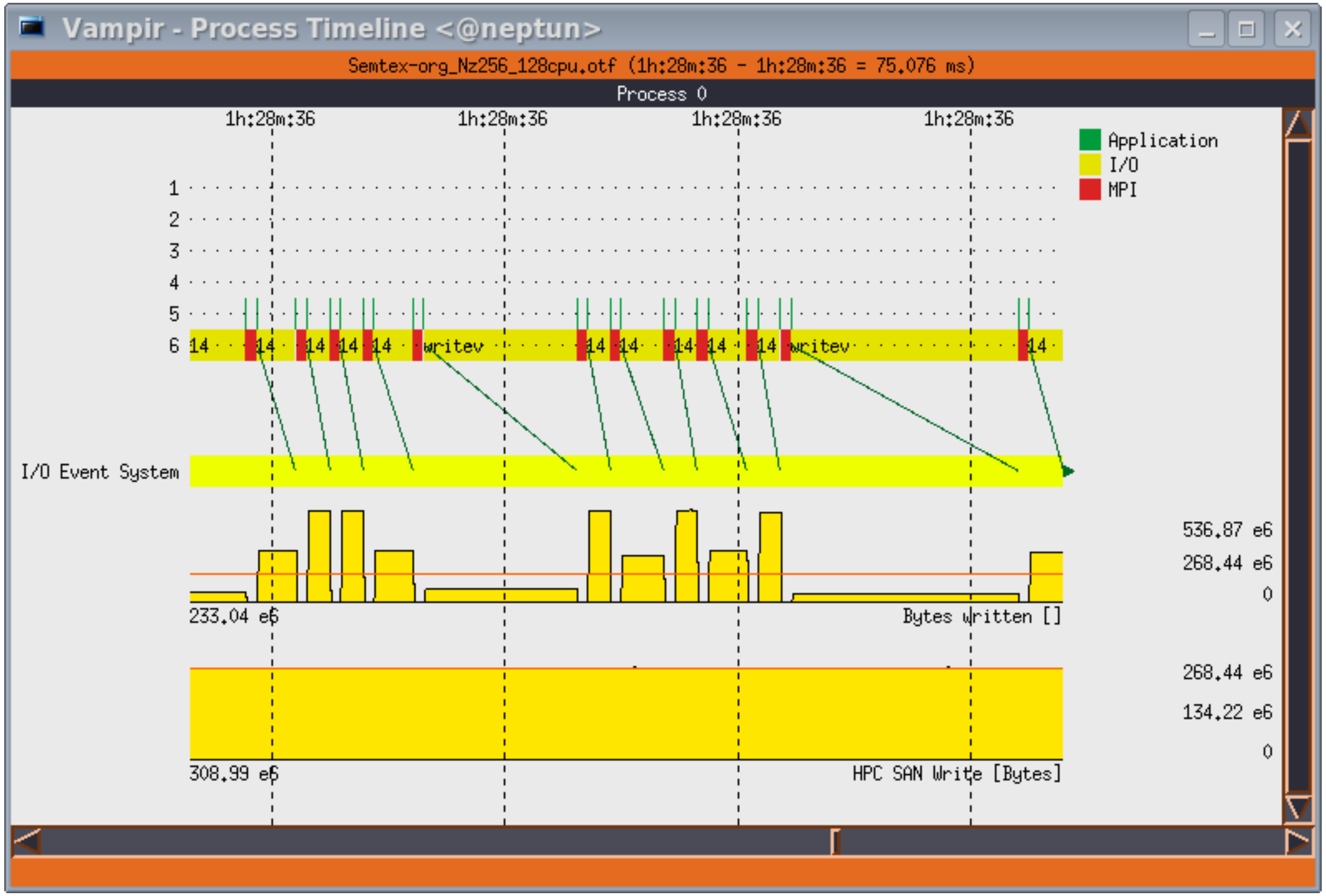


low FP rate due to heavy cache misses

Bottlenecks in Computation



low FP rate due to heavy FP exceptions



irregular slow I/O operations

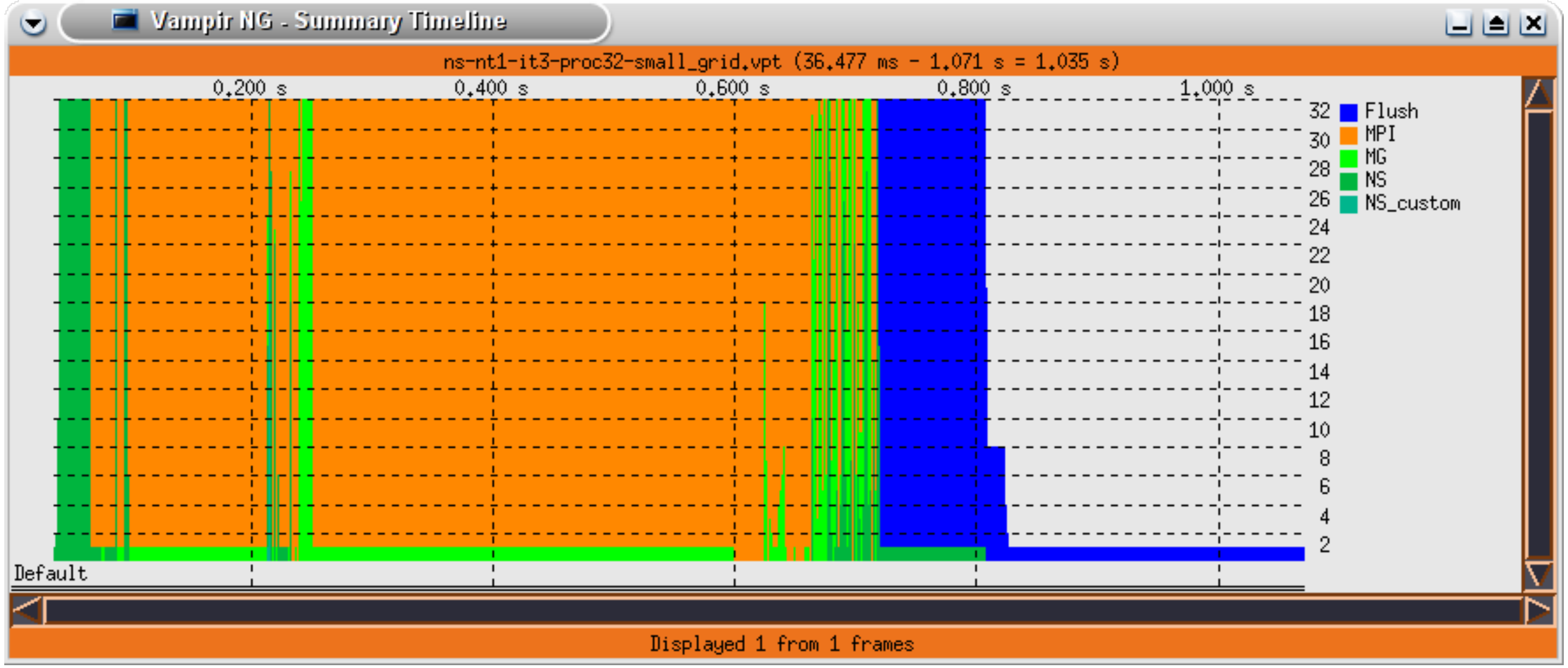
Effects due to Tracing

- 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.

- 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

- This work would have been impossible without the dedication of:
  - Matthias Lieber (Tracing & Analysis)
  - Matthias Jurenz (VampirTrace Software & Support)
  - Matthias Weber (Vampir Software & Support)

- The Vampir Team:

  Matthias Jurenz, Andreas Knüpfer, Ronny Brendel, Matthias Lieber, Jens Doleschal, Holger Mickler, Daniel Hackenberg, Michael Heyde, Guido Juckeland, Dietrich Robert, Johannes Spazier, Michael Kluge, Matthias Müller, Holger Brunst, Ronald Geisler, Reinhard Neumann, Heide Rohling, Rene Widera, Thomas Ilsche, Matthias Weber, Bert Wesarg, Hartmut Mix, Thomas William, Wolfgang E. Nagel