TECHNISCHE UNIVERSITÄT DRESDEN

Trace Based Performance Analysis at Large Scale on Jaguar and Titan

Guido Juckeland (guido.juckeland@tu-dresden.de)

ZIH
Center for Information Services &
High Performance Computing

Disclaimer

Performance tools will not automatically make you code run faster. They help you understand, what your code does and where to put in work.

Agenda

Introduction

The Vampir Workflow

VampirTrace

- Instrumentation & Run-Time Measurement

The Vampir Displays

- "Seeing" Performance Bottlenecks

Vampir on Jaguar and Titan

- Scalability & GPU Support
- How to use Vampir at OLCF

Conclusions

Introduction

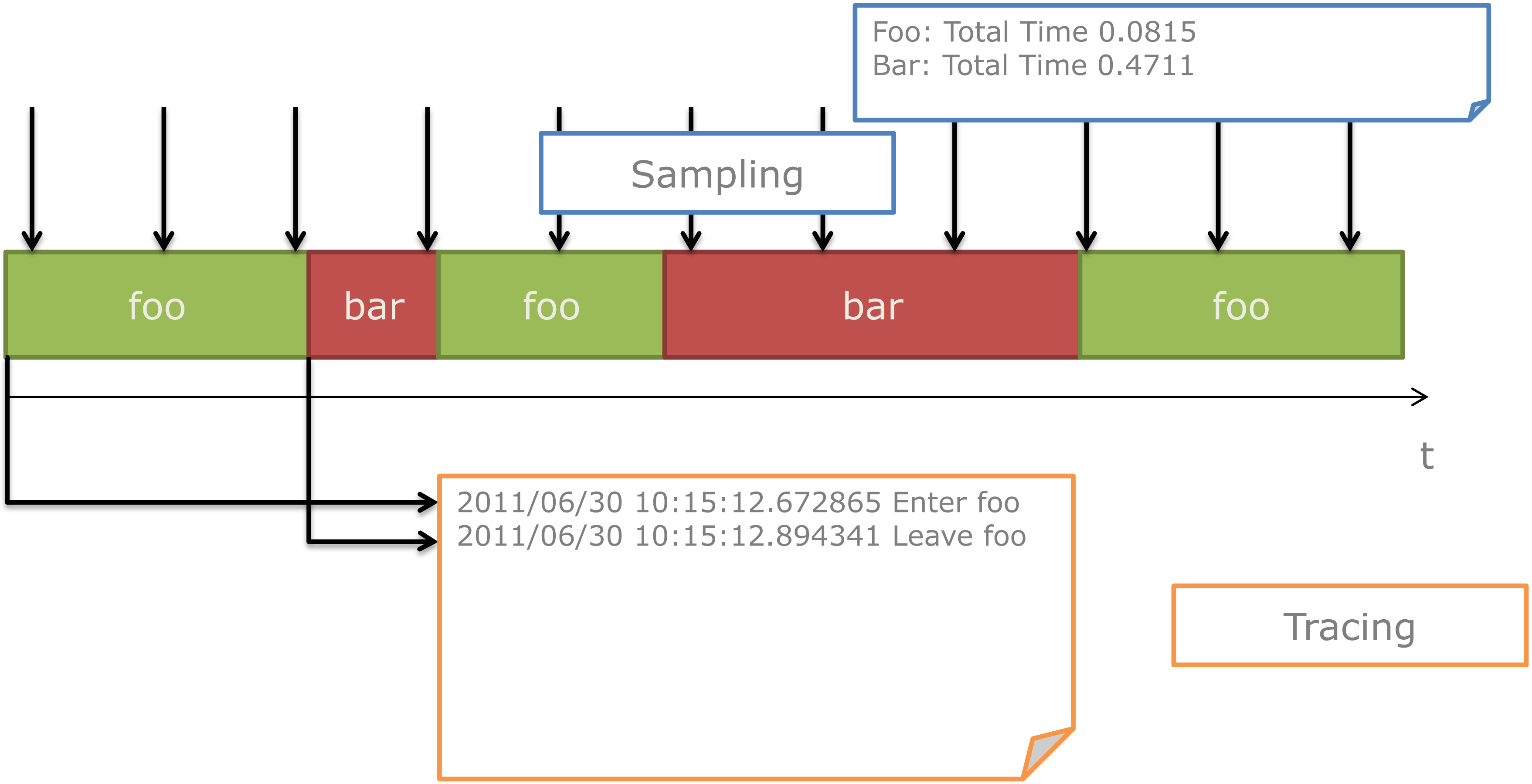
Why bother with performance analysis?

- 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!

Sampling vs. Tracing

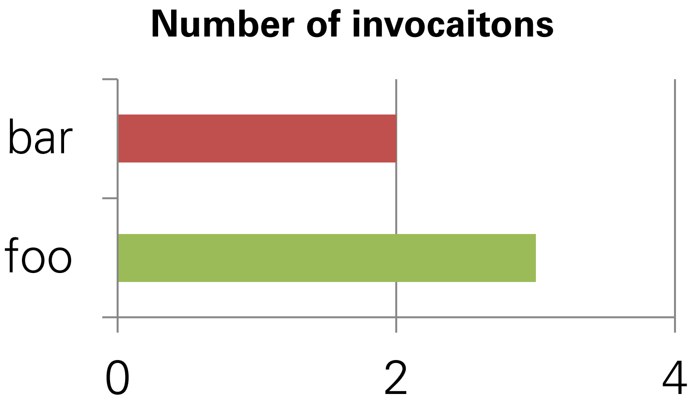


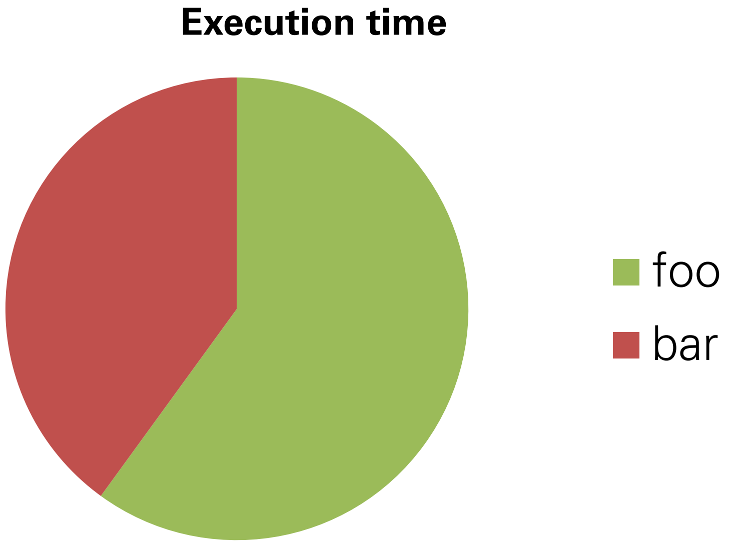
Foo: Total Time 0.0815
Bar: Total Time 0.4711

```
2011/06/30 10:15:12.672865 Enter foo
2011/06/30 10:15:12.894341 Leave foo
```

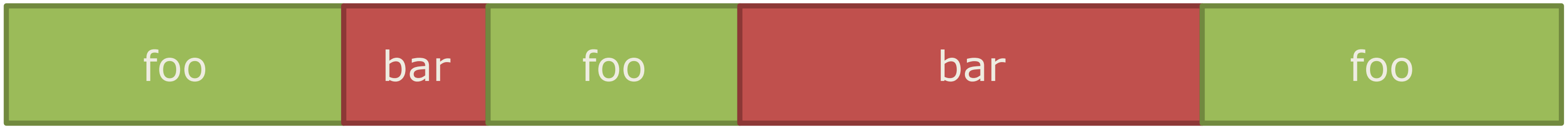
Program profiles

Statistics





Timelines



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

Instrumentation & Measurement

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

Instrumentation & Measurement

What does VampirTrace do in the background?

Trace Run:

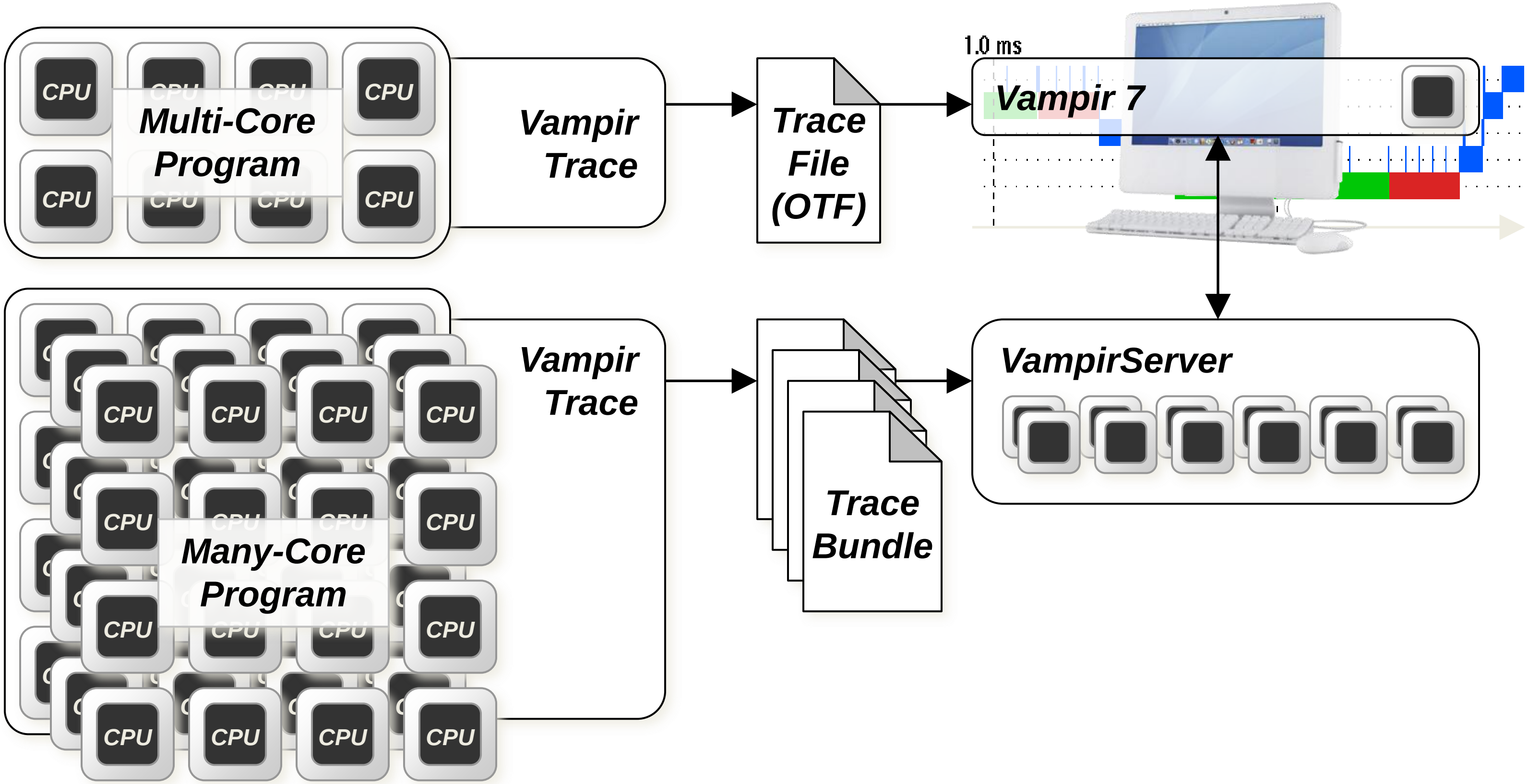
- Event data collection
- Precise time measurement
- Parallel timer synchronization
- Collecting parallel process/thread traces
- Collecting performance counters (from PAPI, memory usage, POSIX I/O calls and fork/system/exec calls, CUDA, and more …)
- Monitor GPU usage
- Filtering and grouping of function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

Event Trace Visualization

Trace Visualization

- Alternative and supplement to automatic analysis
- Show dynamic run-time behavior graphically
- Provide statistics and performance metrics
  - Master 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 Workflow

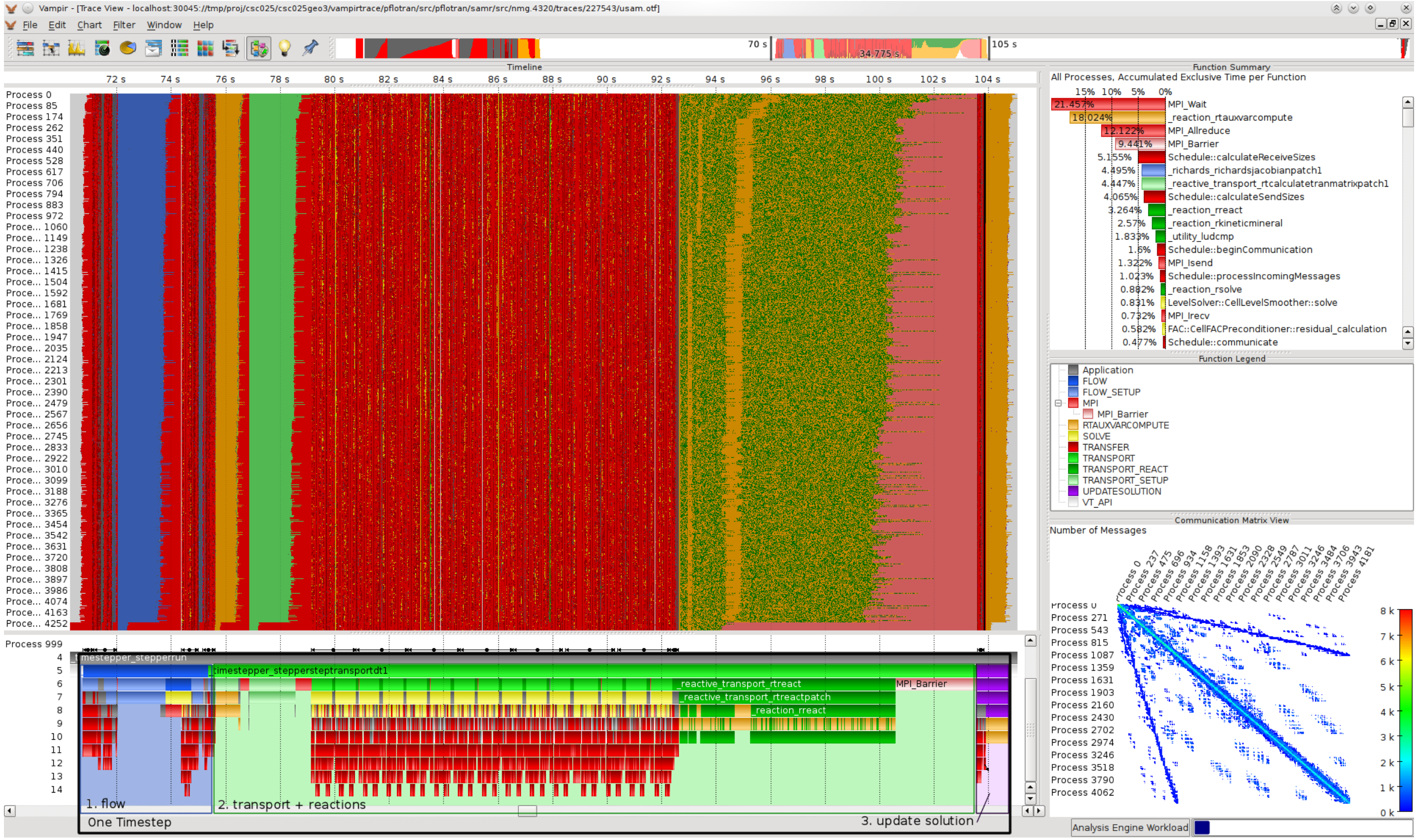


Vampir

GUI to analyze trace files (OTF)

Main concept: Timeline + statistics

GUI is QT based – available on Linux, Mac, Windows



Vampir - [Trace View - localhost:30045://tmp/proj/csc025/csc025geo3/vampirtrace/pflotran/src/pflotran/samr/src/nmg.4320/traces/227543/usam.otf]

Function Summary

All Processes, Accumulated Exclusive Time per Function

%	Function
21.457%	MPI_Wait
18.024%	_reaction_rtauxvarcompute
12.122%	MPI_Allreduce
9.441%	MPI_Barrier
5.155%	Schedule::calculateReceiveSizes
4.495%	_richards_richardsjacobianpatch1
4.447%	_reactive_transport_rtcalculatetranmatrixpatch1
4.065%	Schedule::calculateSendSizes
3.264%	_reaction_rreact
2.57%	_reaction_rkineticmineral
1.833%	_utility_ludcmp
1.6%	Schedule::beginCommunication
1.322%	MPI_Isend
1.023%	Schedule::processIncomingMessages
0.882%	_reaction_rsolve
0.831%	LevelSolver::CellLevelSmoother::solve
0.732%	MPI_Irecv
0.582%	FAC::CellFACPreconditioner::residual_calculation
0.477%	Schedule::communicate

Function Legend

- Application
- FLOW
- FLOW_SETUP
- MPI
  - MPI_Barrier
- RTAUXVARCOMPUTE
- SOLVE
- TRANSFER
- TRANSPORT
- TRANSPORT_REACT
- TRANSPORT_SETUP
- UPDATESOLUTION
- VT_API

VampirServer

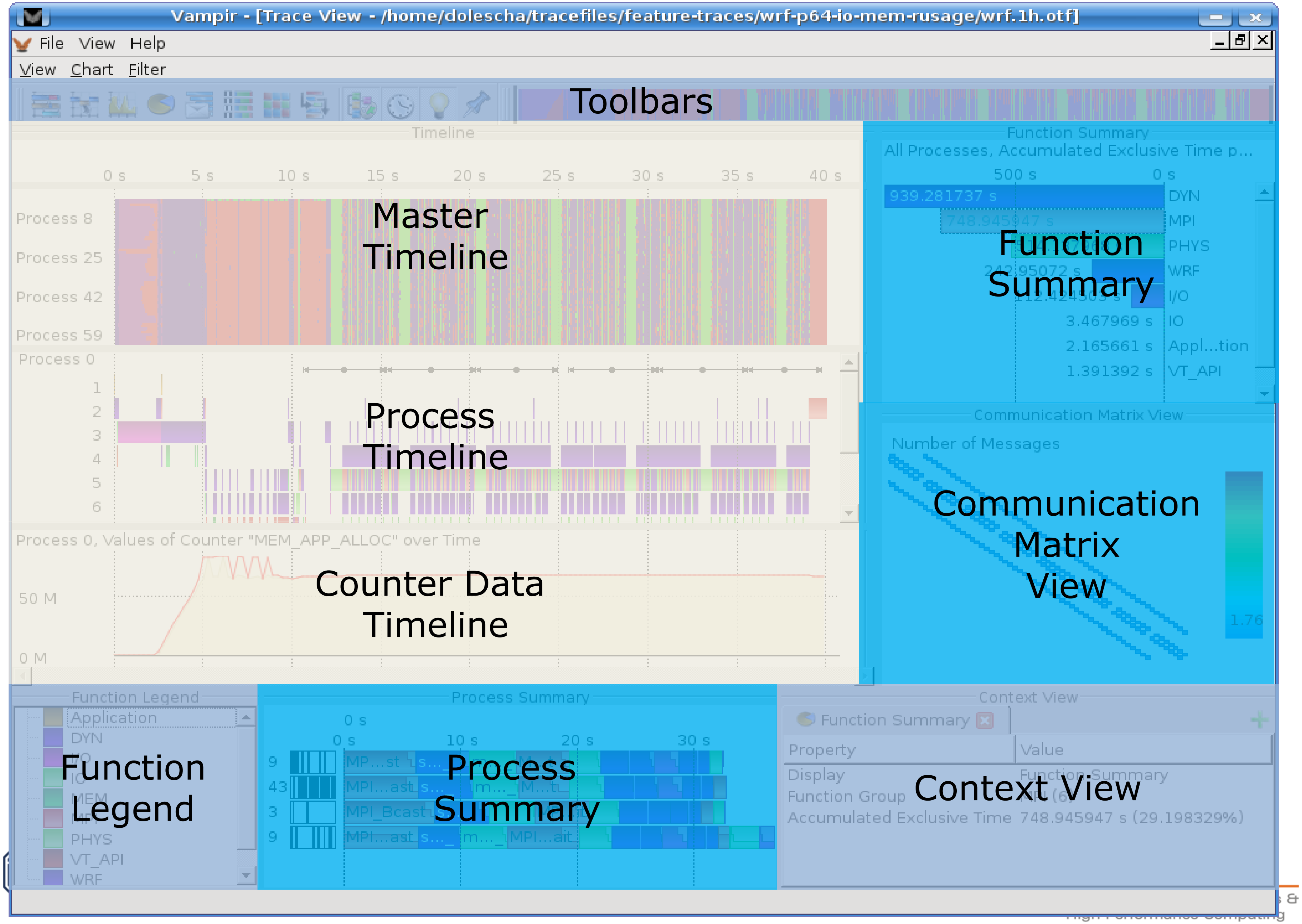
Parallel analysis engine for Vampir

- MPI
- pthreads

Scales to > 10,000 analysis processes

Loads the entire uncompressed trace into memory

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%)

Current Scalability Features in Vampir

Fit to chart height feature of master timeline and performance radar

- Allows visualization of more processes than pixels of screen are available

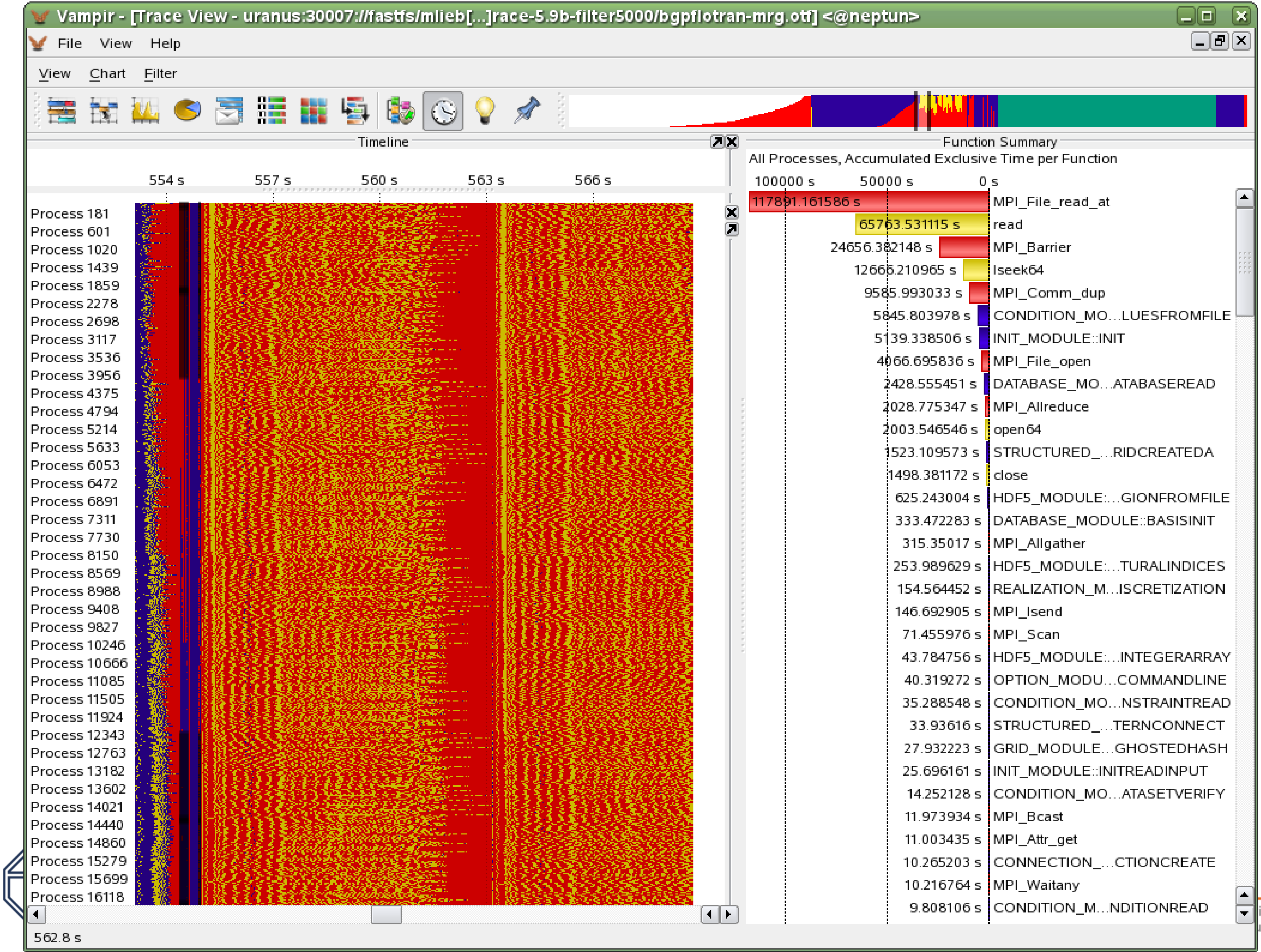
Clustering

- Allows detection of outlier processes and groups with similar behavior

Performance radar

- Highlighting performance conditions of your program in a global timeline

Scalability Feature I: Fit to chart height: Pflotran initialization + I/O

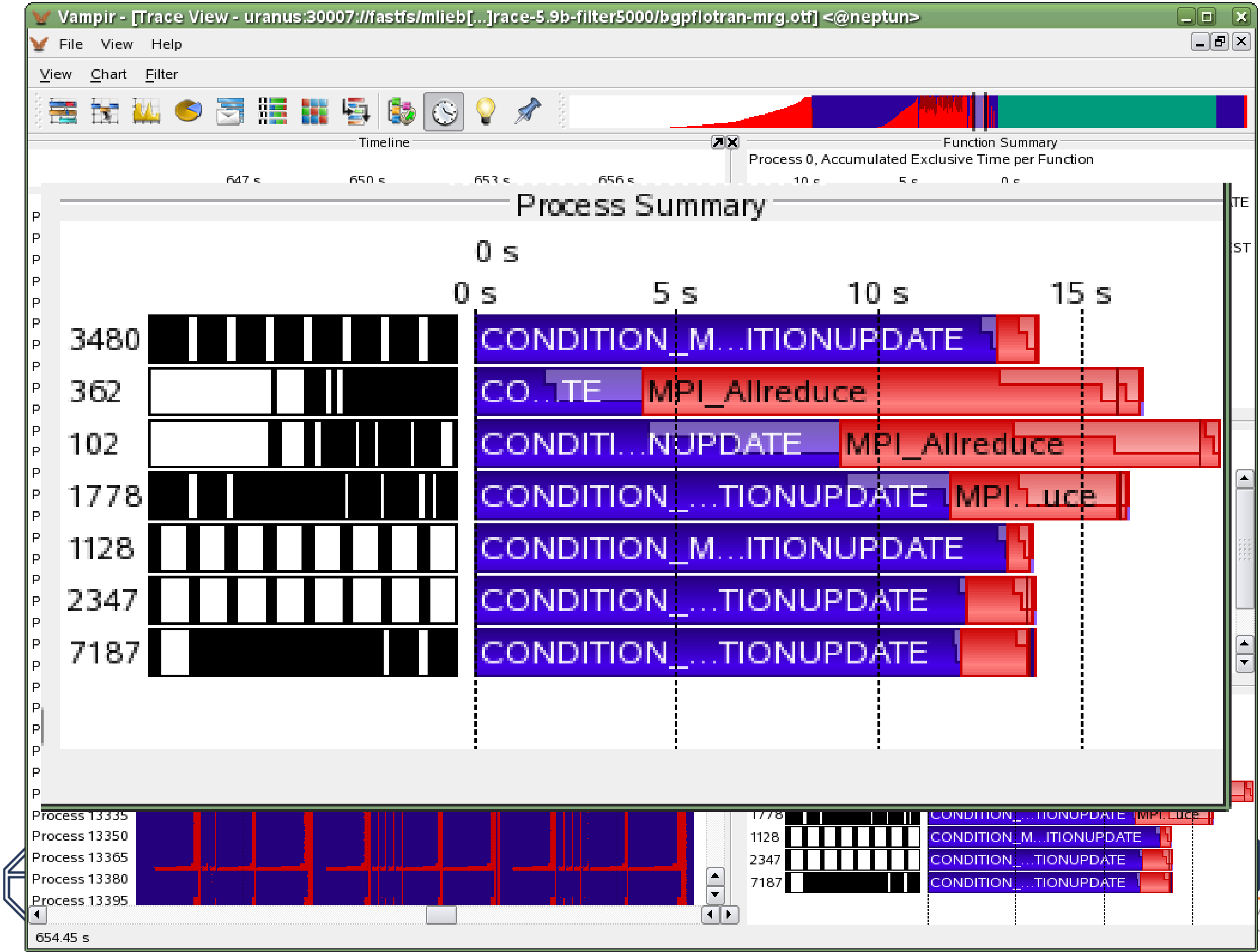


Vampir - [Trace View - uranus:30007://fastfs/mlieb[...]race-5.9b-filter5000/bgpflotran-mrg.otf] <@neptun>

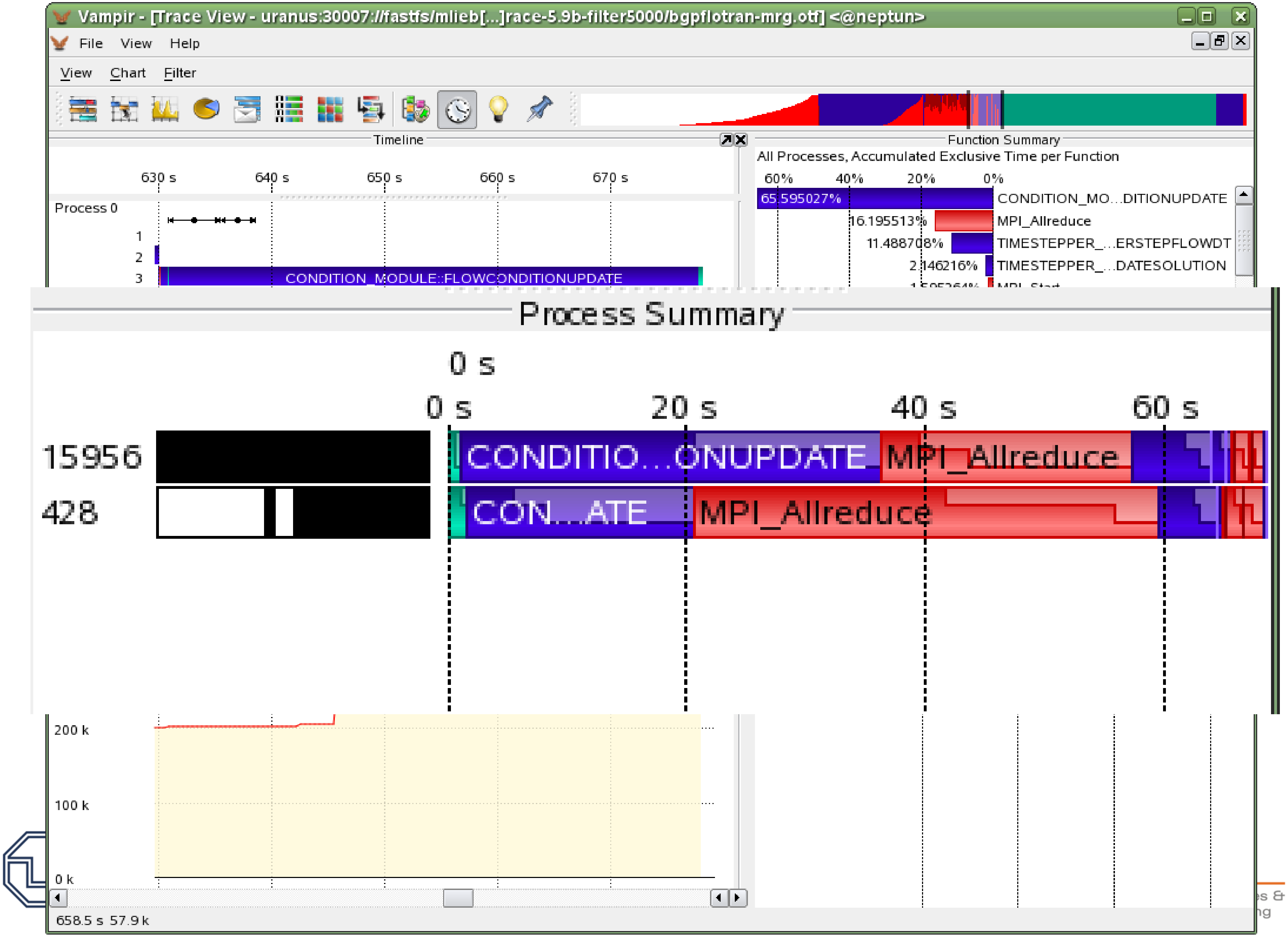
All Processes, Accumulated Exclusive Time per Function

117891.161586 s	MPI_File_read_at
65763.531115 s	read
24656.382148 s	MPI_Barrier
12665.210965 s	lseek64
9585.993033 s	MPI_Comm_dup
5845.803978 s	CONDITION_MO...LUESFROMFILE
5139.338506 s	INIT_MODULE::INIT
4066.695836 s	MPI_File_open
2428.555451 s	DATABASE_MO...ATABASEREAD
2028.775347 s	MPI_Allreduce
2003.546546 s	open64
1523.109573 s	STRUCTURED_...RIDCREATEDA
1498.381172 s	close
625.243004 s	HDF5_MODULE:...GIONFROMFILE
333.472283 s	DATABASE_MODULE::BASISINIT
315.35017 s	MPI_Allgather
253.989629 s	HDF5_MODULE:...TURALINDICES
154.564452 s	REALIZATION_M...ISCRETIZATION
146.692905 s	MPI_Isend
71.455976 s	MPI_Scan
43.784756 s	HDF5_MODULE:...INTEGERARRAY
40.319272 s	OPTION_MODU...COMMANDLINE
35.288548 s	CONDITION_MO...NSTRAINTREAD
33.93616 s	STRUCTURED_...TERNCONNECT
27.932223 s	GRID_MODULE...GHOSTEDHASH
25.696161 s	INIT_MODULE::INITREADINPUT
14.252128 s	CONDITION_MO...ATASETVERIFY
11.973934 s	MPI_Bcast
11.003435 s	MPI_Attr_get
10.265203 s	CONNECTION_...CTIONCREATE
10.216764 s	MPI_Waitany
9.808106 s	CONDITION_M...NDITIONREAD

Scalability feature II: clustering: Pflotran - first iteration



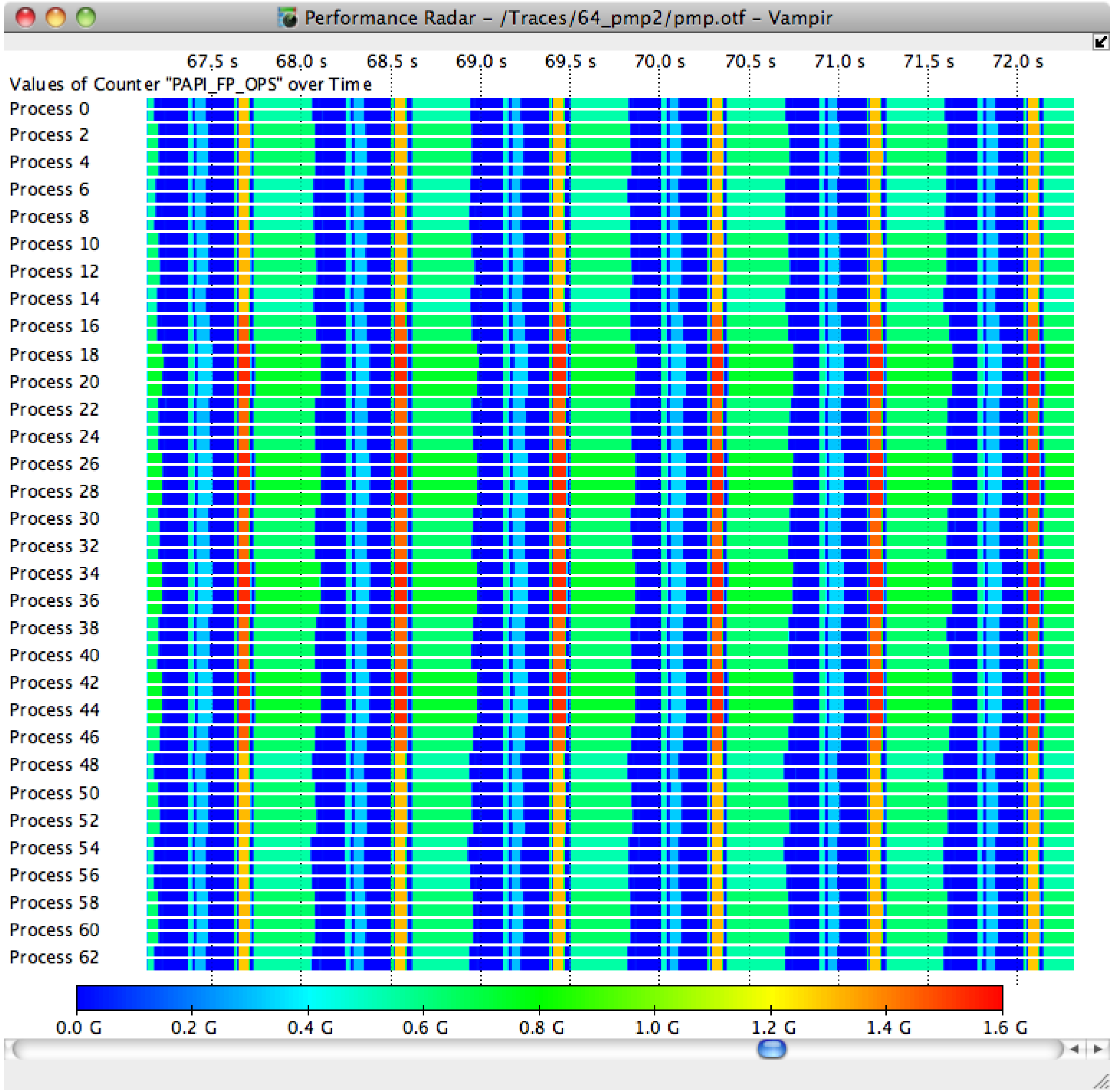
Scalability feature II: clustering: Pflotran – end of initialization & first iteration


Vampir - [Trace View - uranus:30007://fastfs/mlieb[...]race-5.9b-filter5000/bgpflotran-mrg.otf] <@neptun>
File
View
Help
View
Chart
Filter
Timeline
630 s
640 s
650 s
660 s
670 s
Process 0
1
2
3
CONDITION_MODULE::FLOWCONDITIONUPDATE
Function Summary
All Processes, Accumulated Exclusive Time per Function
60%
40%
20%
0%
65.595027%
CONDITION_MO...DITIONUPDATE
16.195513%
MPI_Allreduce
11.488708%
TIMESTEPPER_...ERSTEPFLOWDT
2.146216%
TIMESTEPPER_...DATESOLUTION
Process Summary
0 s
0 s
20 s
40 s
60 s
15956
CONDITIO...ONUPDATE
MPI_Allreduce
428
CON...ATE
MPI_Allreduce
200 k
100 k
0 k
658.5 s 57.9 k

Scalability Feature III: Performance Radar

Display objectives:

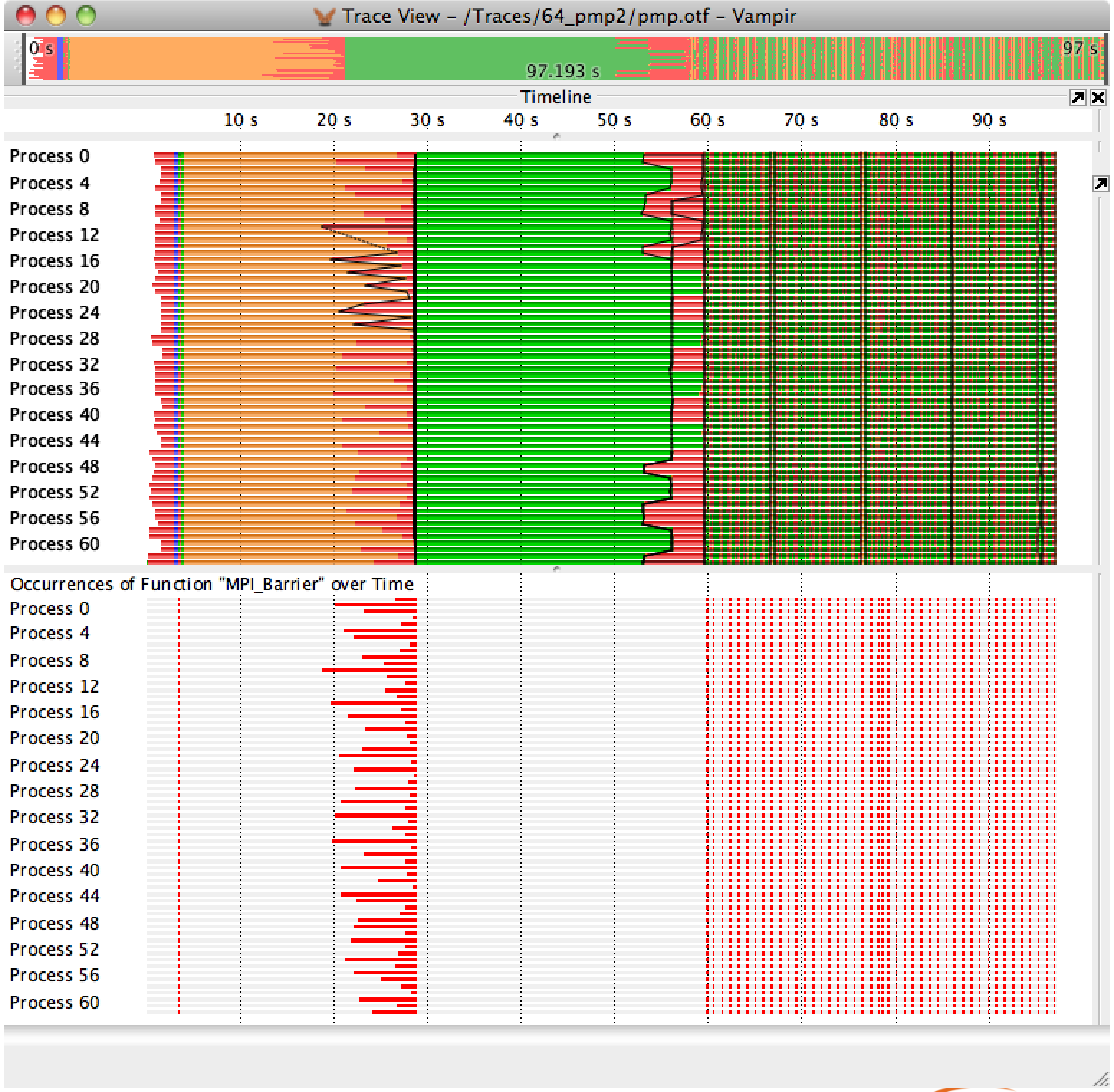
- Identification of performance relevant trace parts
- Assistance to users to navigate in trace data and to spot interesting sections
- Performance of basic arithmetics on counter data



Scalability Feature II: Performance Radar

Features:

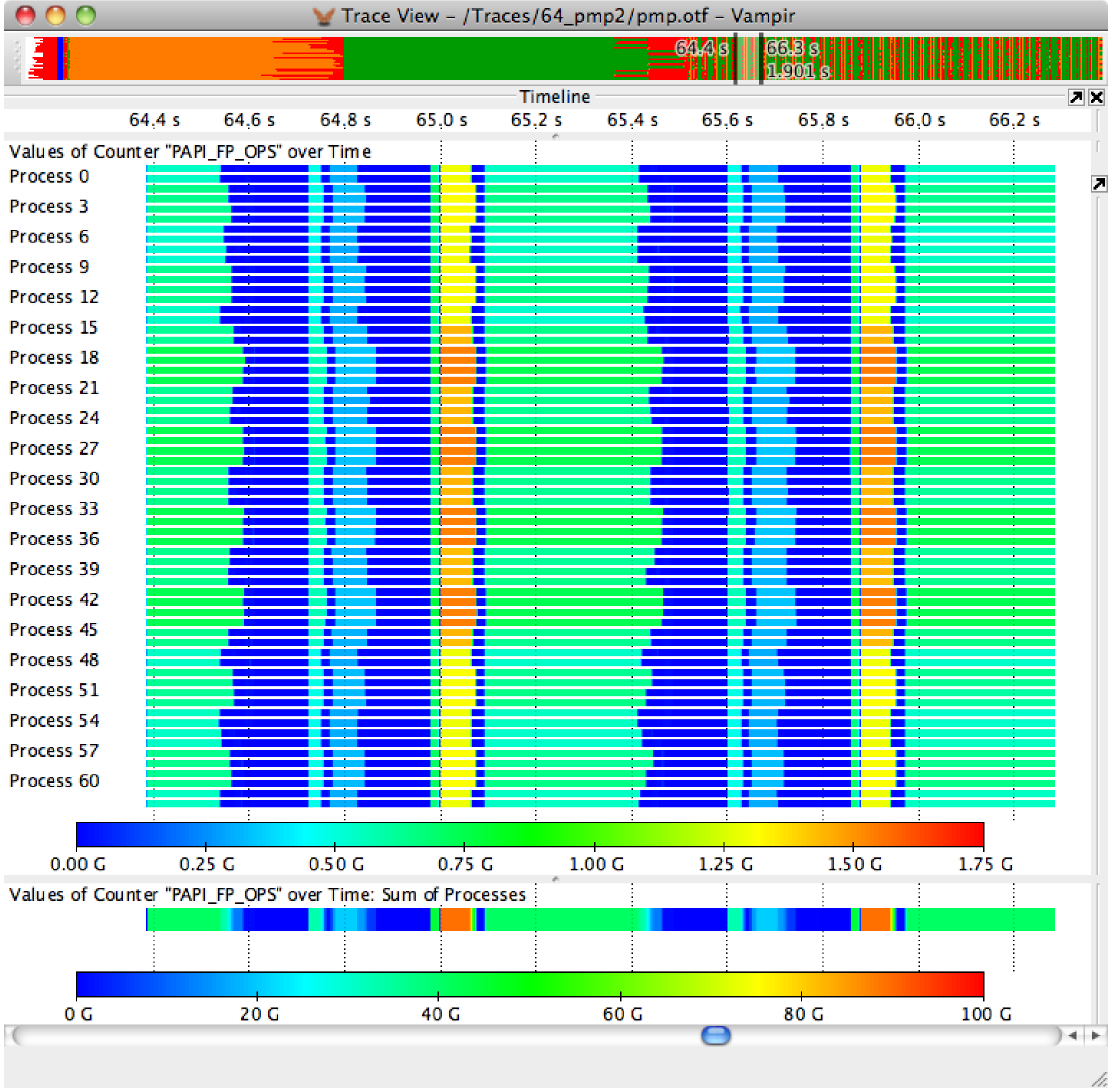
- Detection of occurrences of functions/function groups
- Visualization of call density of functions/function groups to help to find performance relevant candidates
- Construction of filter based on function occurrences over time for further usage in calculations



Scalability Feature II: Performance Radar

Features:

- Performance of arbitrary calculations on counter data, e.g. add up all floating point operations over time, differentiation of performance counter
- Support for concatenation of several calculations
- Utilization of filter in calculations, e.g. only add up FLOPS of function x



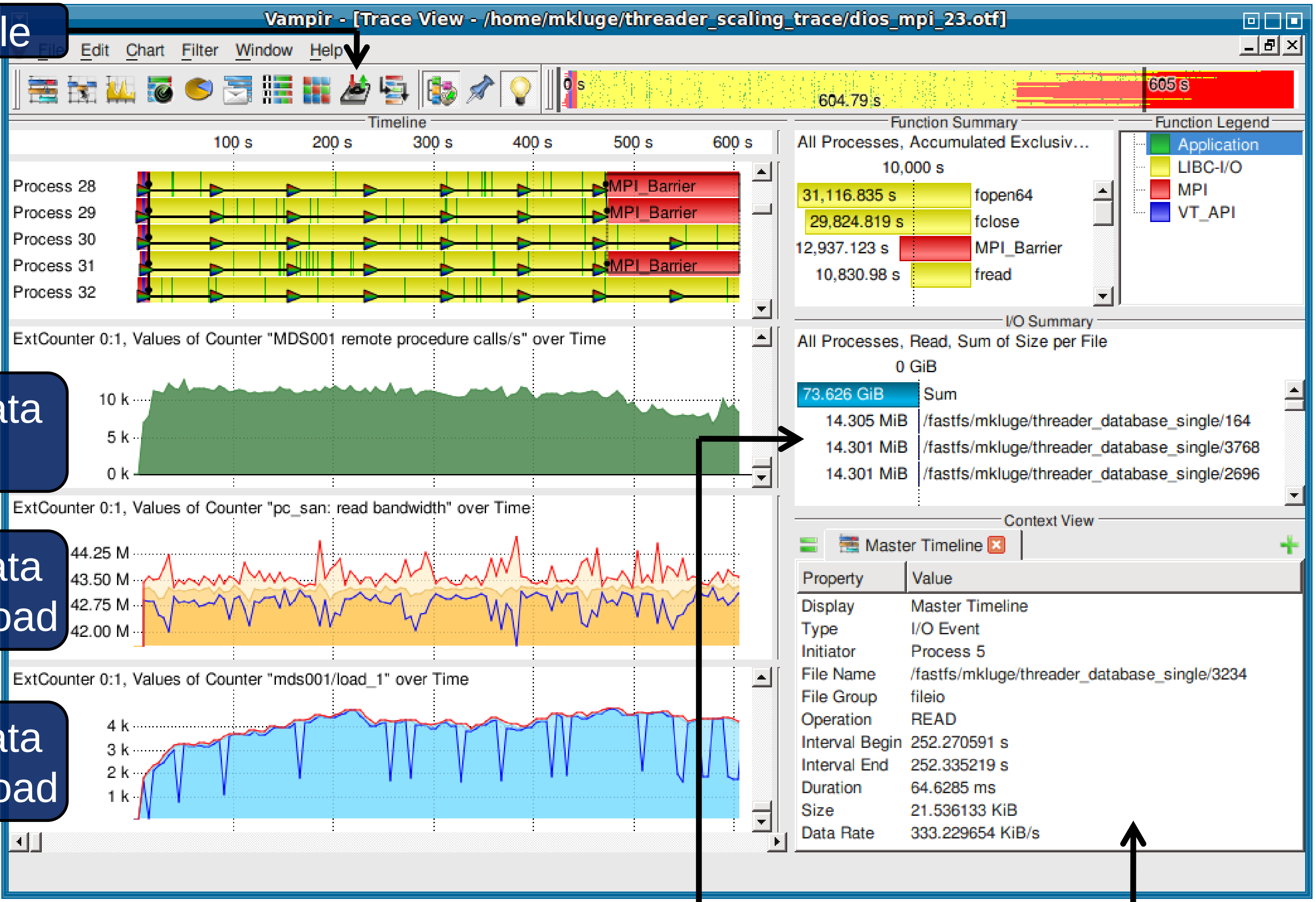
Vampir I/O Analysis

enable

metadata rate

metadata server load

metadata server load



Vampir - [Trace View - /home/mkluge/threader_scaling_trace/dios_mpi_23.otf]

Function Summary

All Processes, Accumulated Exclusiv...

10,000 s

31,116.835 s	fopen64
29,824.819 s	fclose
12,937.123 s	MPI_Barrier
10,830.98 s	fread

I/O Summary

All Processes, Read, Sum of Size per File

0 GiB

73.626 GiB	Sum
14.305 MiB	/fastfs/mkluge/threader_database_single/164
14.301 MiB	/fastfs/mkluge/threader_database_single/3768
14.301 MiB	/fastfs/mkluge/threader_database_single/2696

Property	Value
Display	Master Timeline
Type	I/O Event
Initiator	Process 5
File Name	/fastfs/mkluge/threader_database_single/3234
File Group	fileio
Operation	READ
Interval Begin	252.270591 s
Interval End	252.335219 s
Duration	64.6285 ms
Size	21.536133 KiB
Data Rate	333.229654 KiB/s

detailed per file and summary statistics

detailed per event data

Finding Performance Bottlenecks

Inefficient Communication patterns

Load imbalance / serial parts of the application

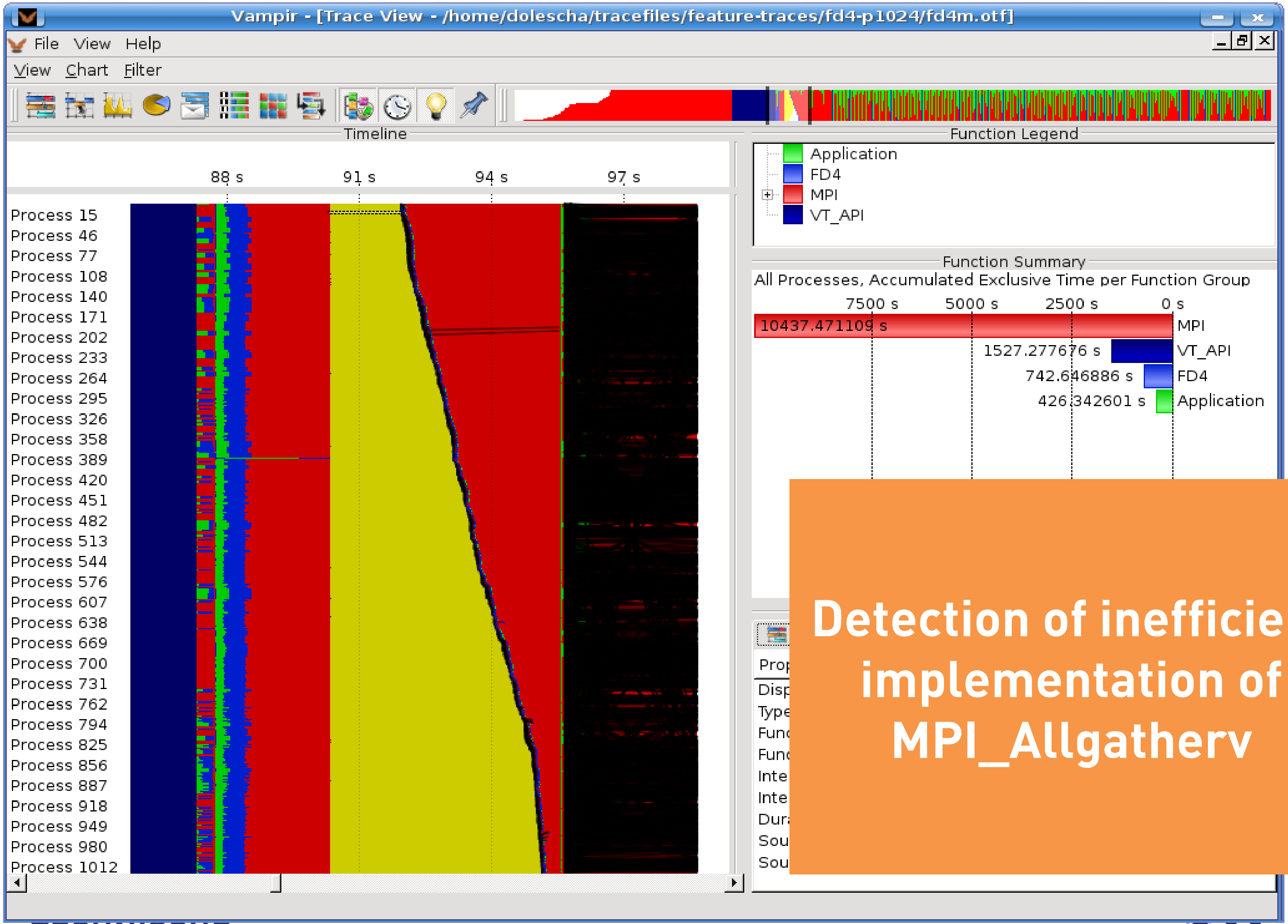
Memory bound computation

- Inefficient cache usage
- TLB misses
- Use HW counters (PAPI) to detect

I/O bottlenecks

Most time consuming function

Scalability Feature I: Fit to chart height: Inefficient MPI_Allgatherv



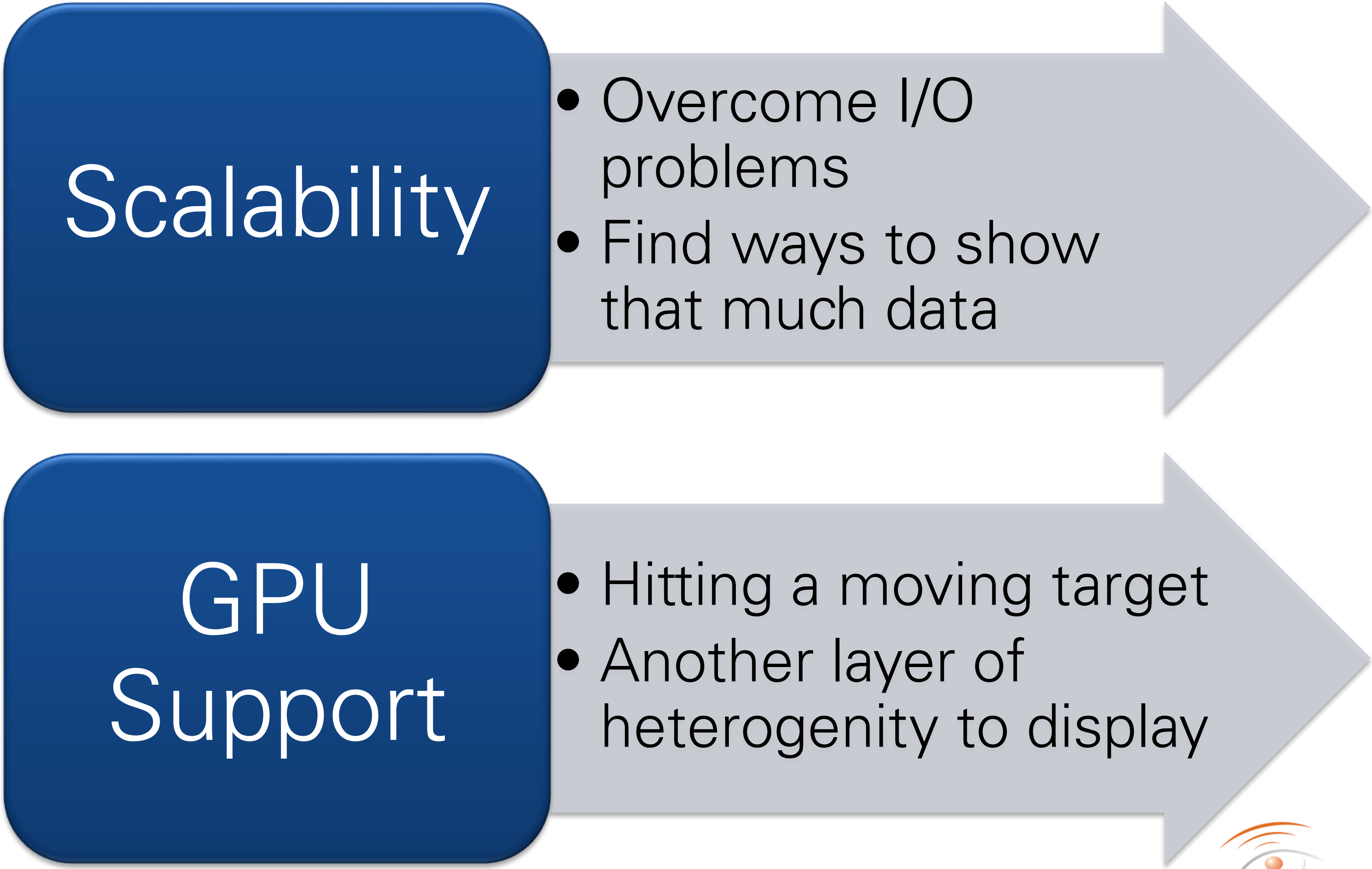
Vampir - [Trace View - /home/dolescha/tracefiles/feature-traces/fd4-p1024/fd4m.otf]

All Processes, Accumulated Exclusive Time per Function Group

10437.471109 s	MPI
1527.277676 s	VT_API
742.646886 s	FD4
426.342601 s	Application

Detection of inefficient implementation of MPI_Allgatherv

Challenges on Jaguar and Titan



Scalability

- Overcome I/O problems
- Find ways to show that much data

GPU Support

- Hitting a moving target
- Another layer of heterogenity to display

Scalability

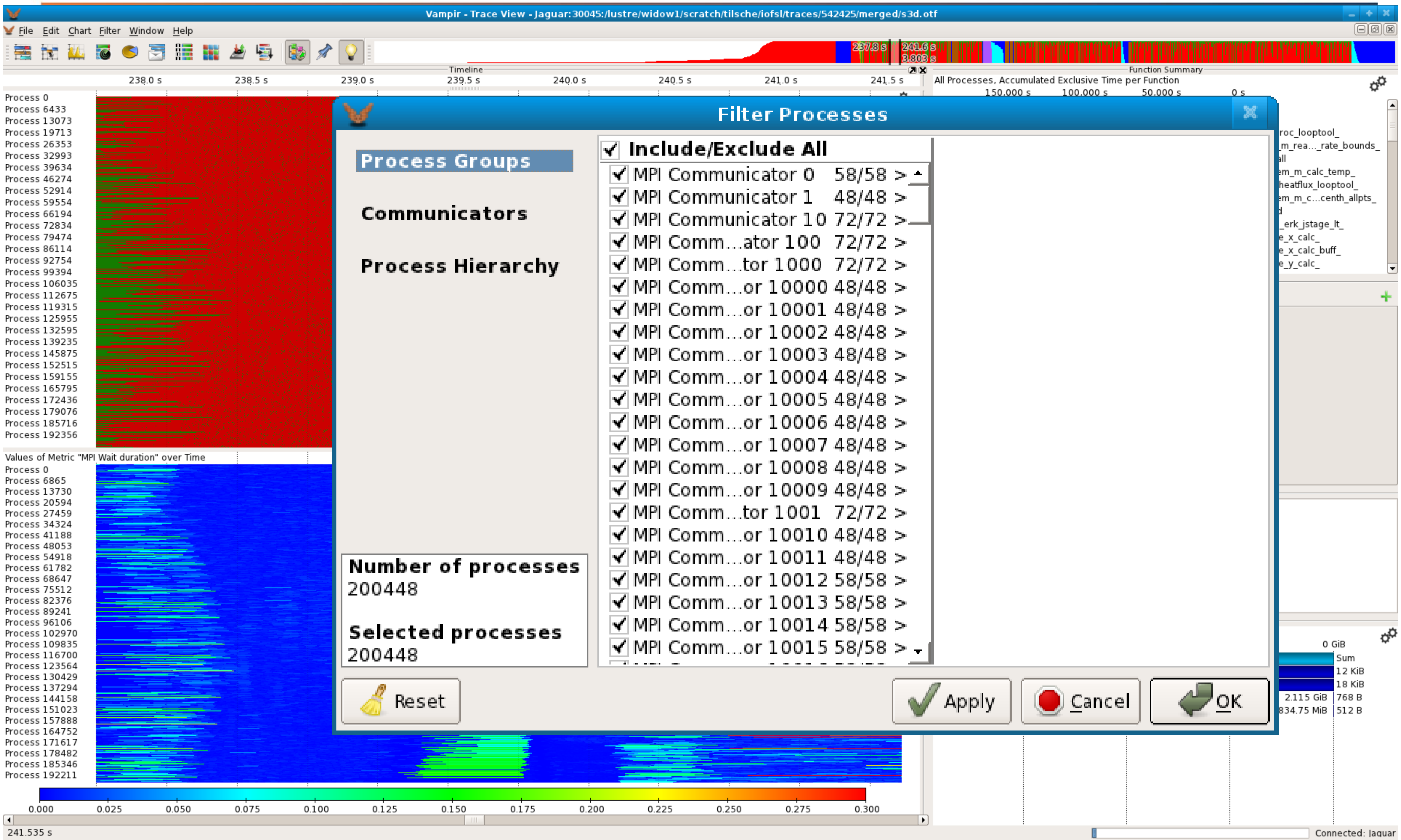
Current limits on Jaguar

- Tracing up to 8000 Processes
  - I/O Problem (too many file creates – one per process)

Prototype already working on Jaguar

- Tracing 200,000+ processes
- Opened with 20,000 VampirServer processes

200,000+ Processes in Vampir



Filter Processes

Process Groups

Communicators

Process Hierarchy

☑ **Include/Exclude All**

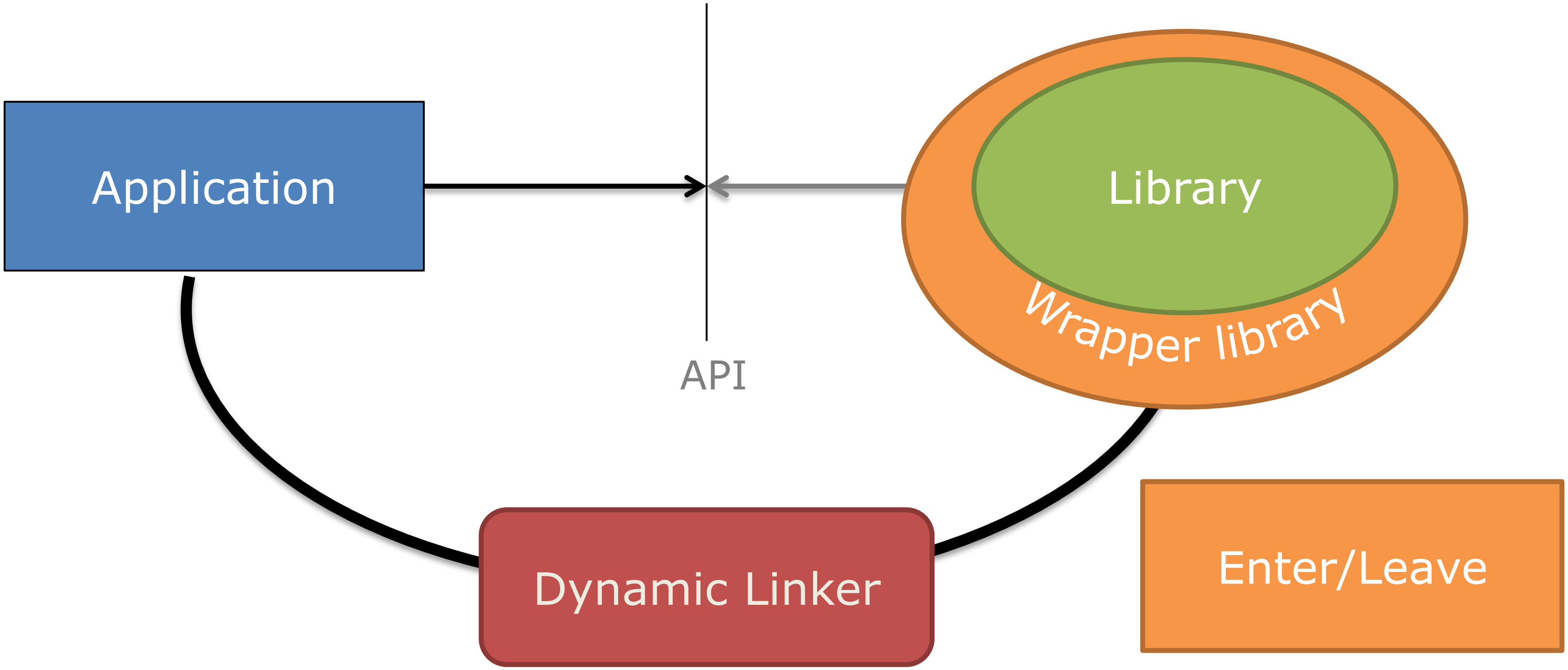
- ☑ MPI Communicator 0 58/58 >
- ☑ MPI Communicator 1 48/48 >
- ☑ MPI Communicator 10 72/72 >
- ☑ MPI Comm...ator 100 72/72 >
- ☑ MPI Comm...tor 1000 72/72 >
- ☑ MPI Comm...or 10000 48/48 >
- ☑ MPI Comm...or 10001 48/48 >
- ☑ MPI Comm...or 10002 48/48 >
- ☑ MPI Comm...or 10003 48/48 >
- ☑ MPI Comm...or 10004 48/48 >
- ☑ MPI Comm...or 10005 48/48 >
- ☑ MPI Comm...or 10006 48/48 >
- ☑ MPI Comm...or 10007 48/48 >
- ☑ MPI Comm...or 10008 48/48 >
- ☑ MPI Comm...or 10009 48/48 >
- ☑ MPI Comm...tor 1001 72/72 >
- ☑ MPI Comm...or 10010 48/48 >
- ☑ MPI Comm...or 10011 48/48 >
- ☑ MPI Comm...or 10012 58/58 >
- ☑ MPI Comm...or 10013 58/58 >
- ☑ MPI Comm...or 10014 58/58 >
- ☑ MPI Comm...or 10015 58/58 >

Number of processes
200448

Selected processes
200448

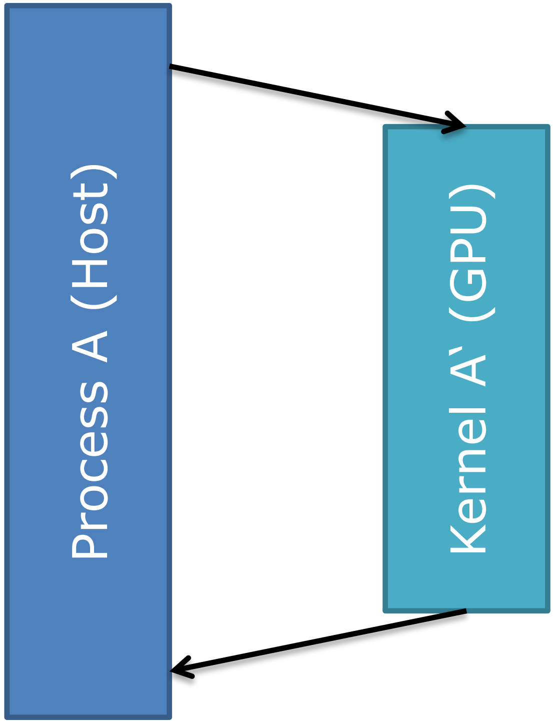
GPU Support: CUDA (+OpenCL)

Currently done with Library Wrapping

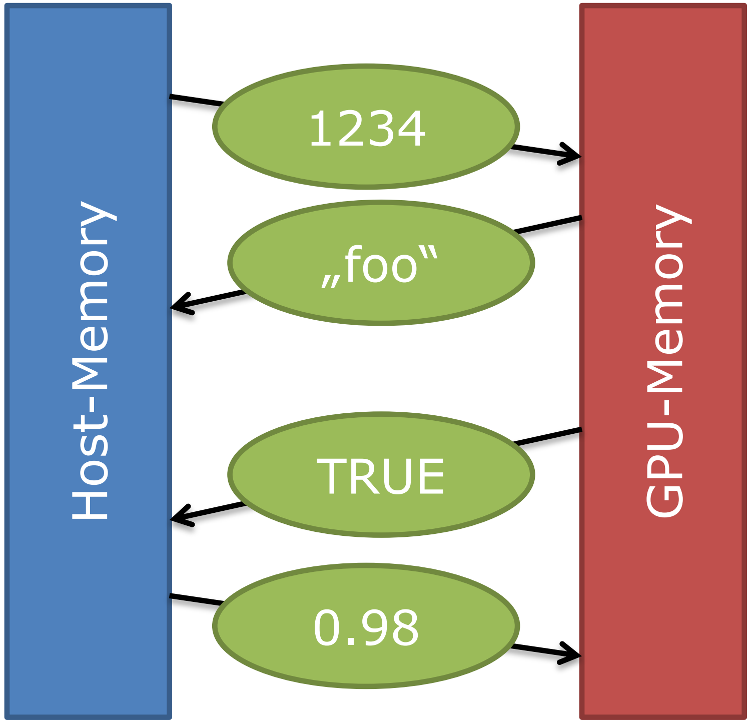


Reuse known metrics

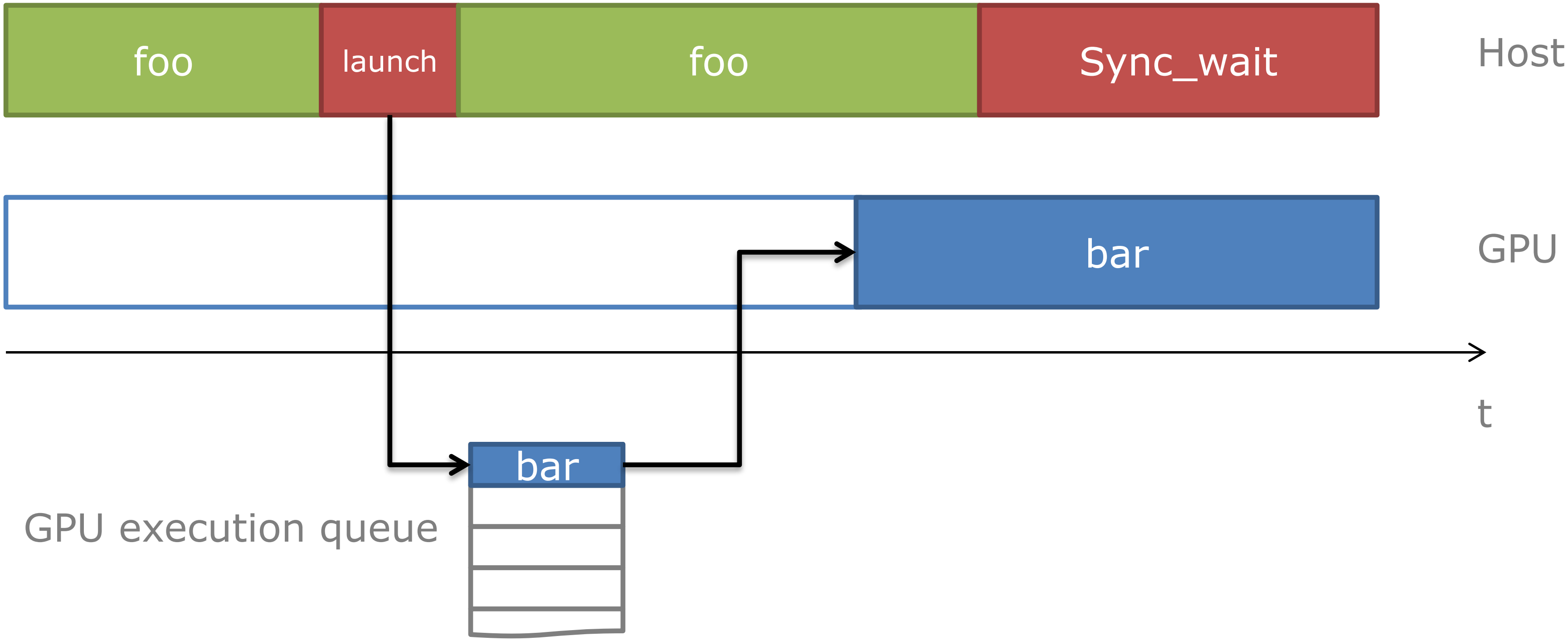
Thread = Kernel



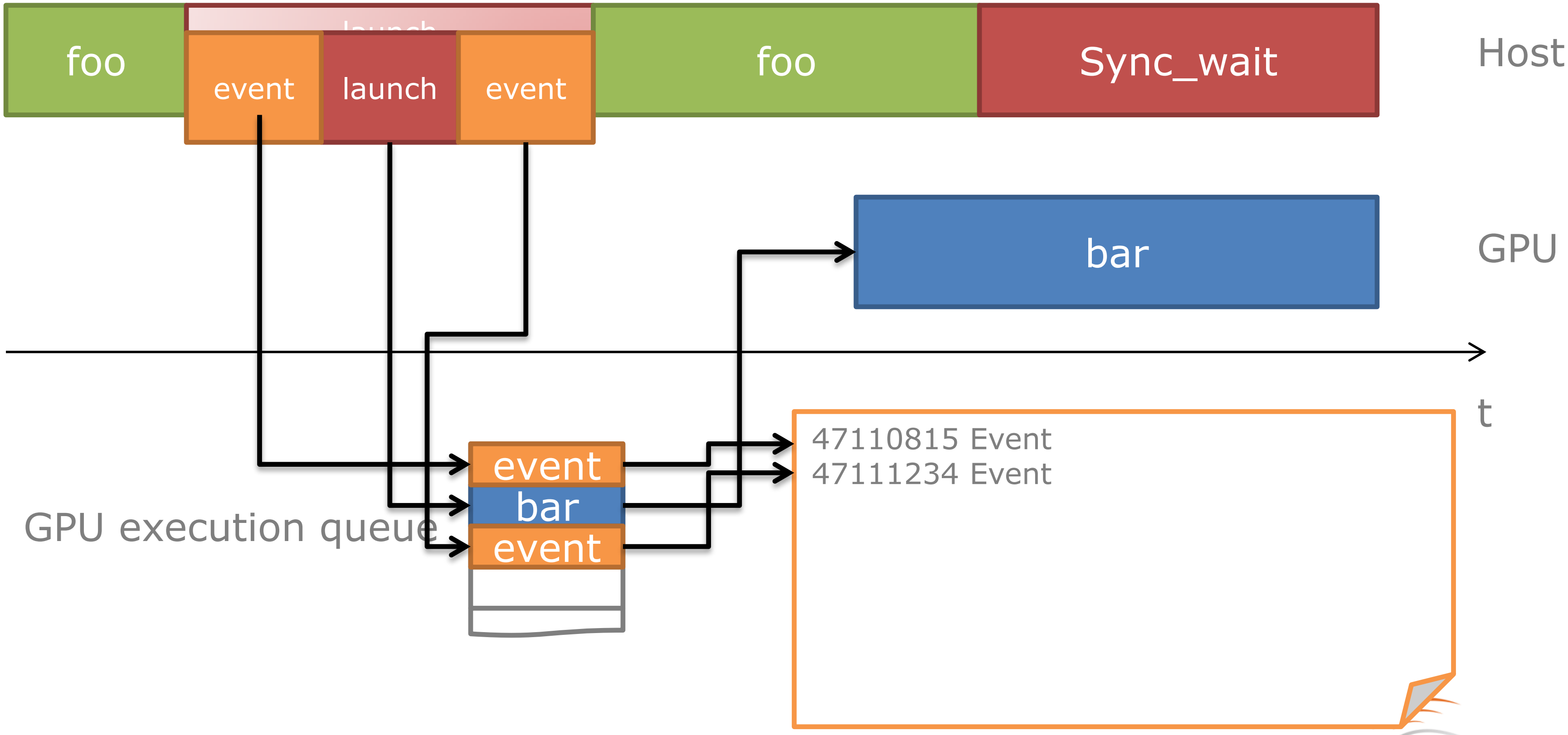
Message = cudaMemCpy



Asynchronous Events



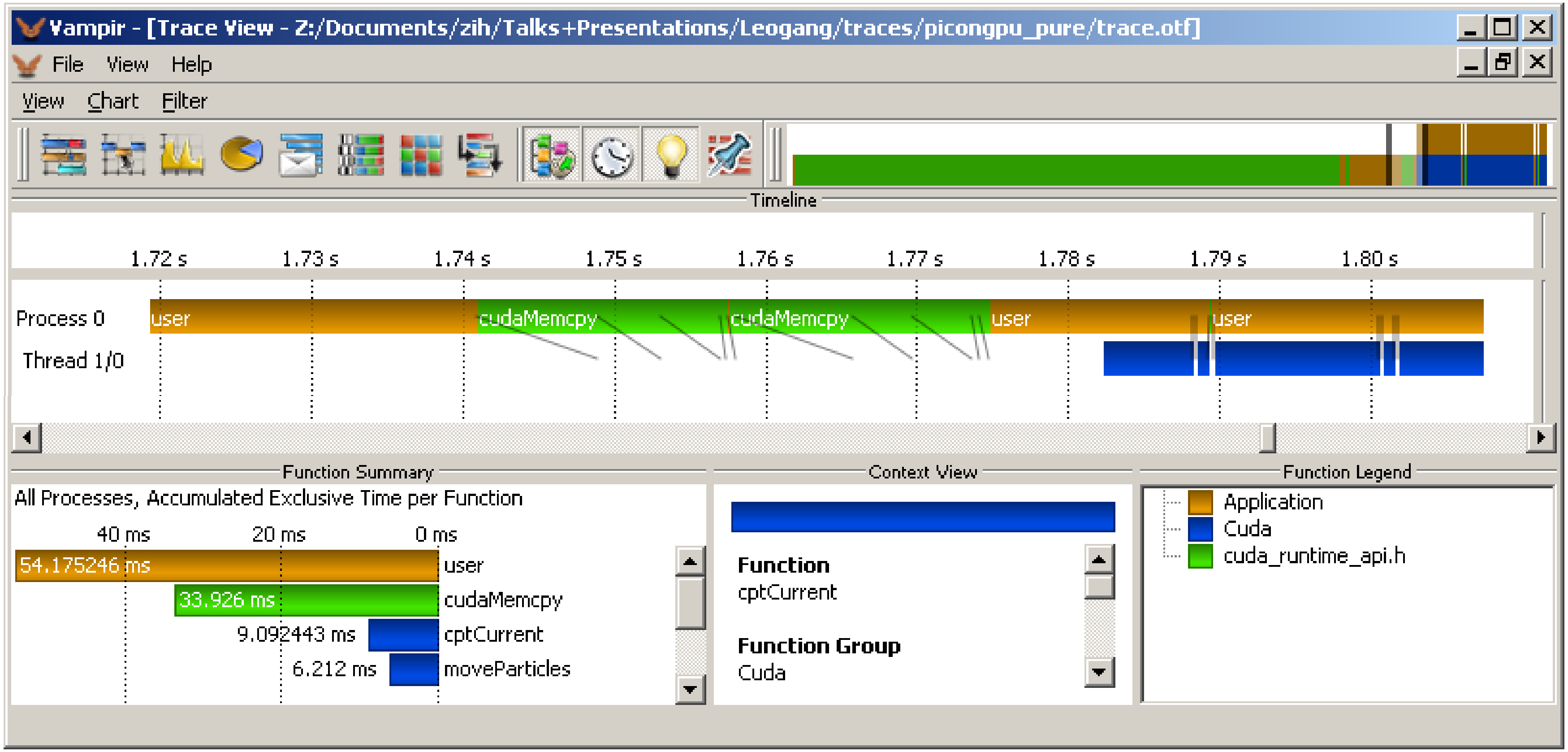
Time stamps for asynchronous events



PIConGPU – an example for a multi-hybrid application

http://picongpu.fzd.de

1 GPU

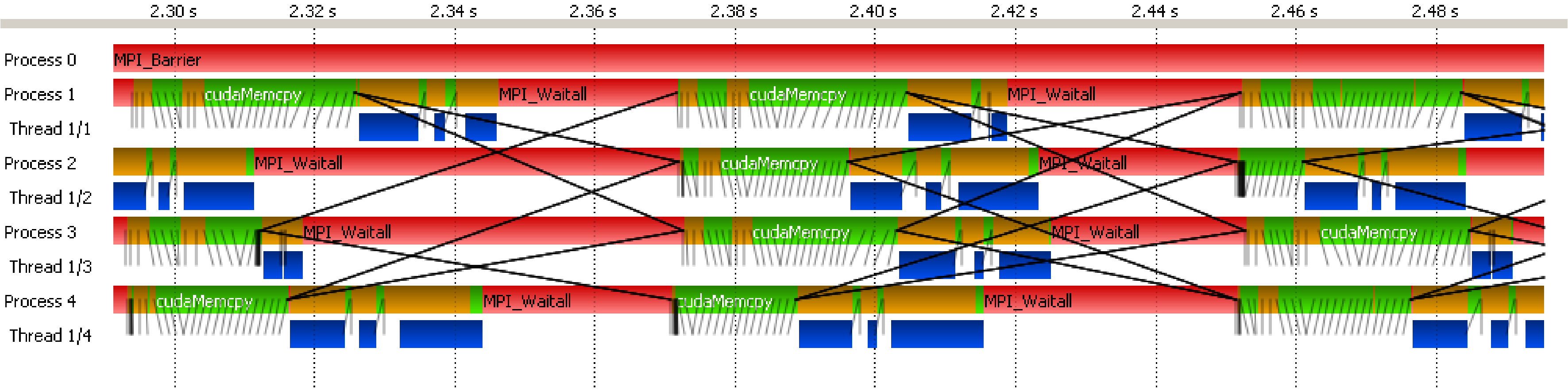


Vampir - [Trace View - Z:/Documents/zih/Talks+Presentations/Leogang/traces/picongpu_pure/trace.otf]

Function
cptCurrent

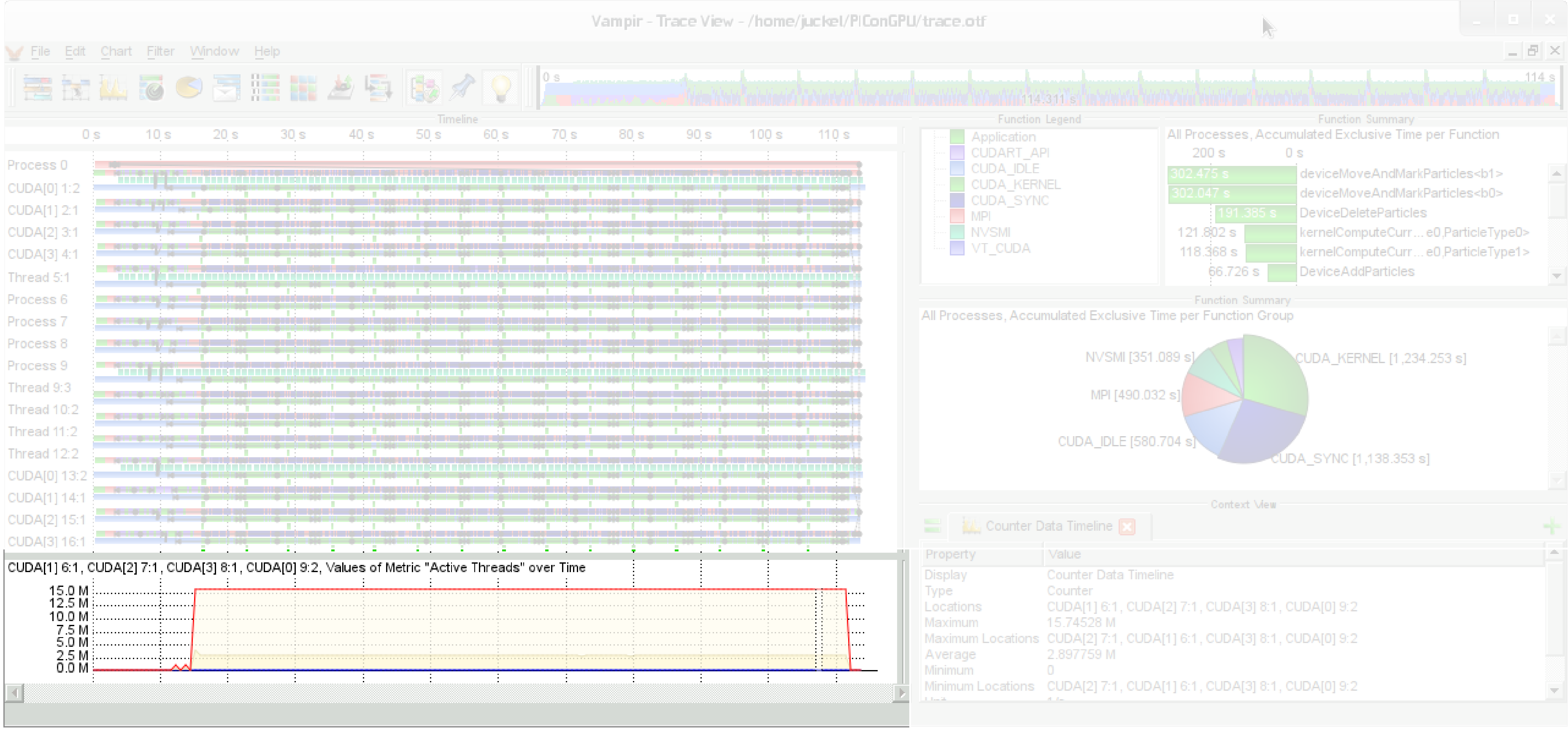
Function Group
Cuda

MPI + CUDA



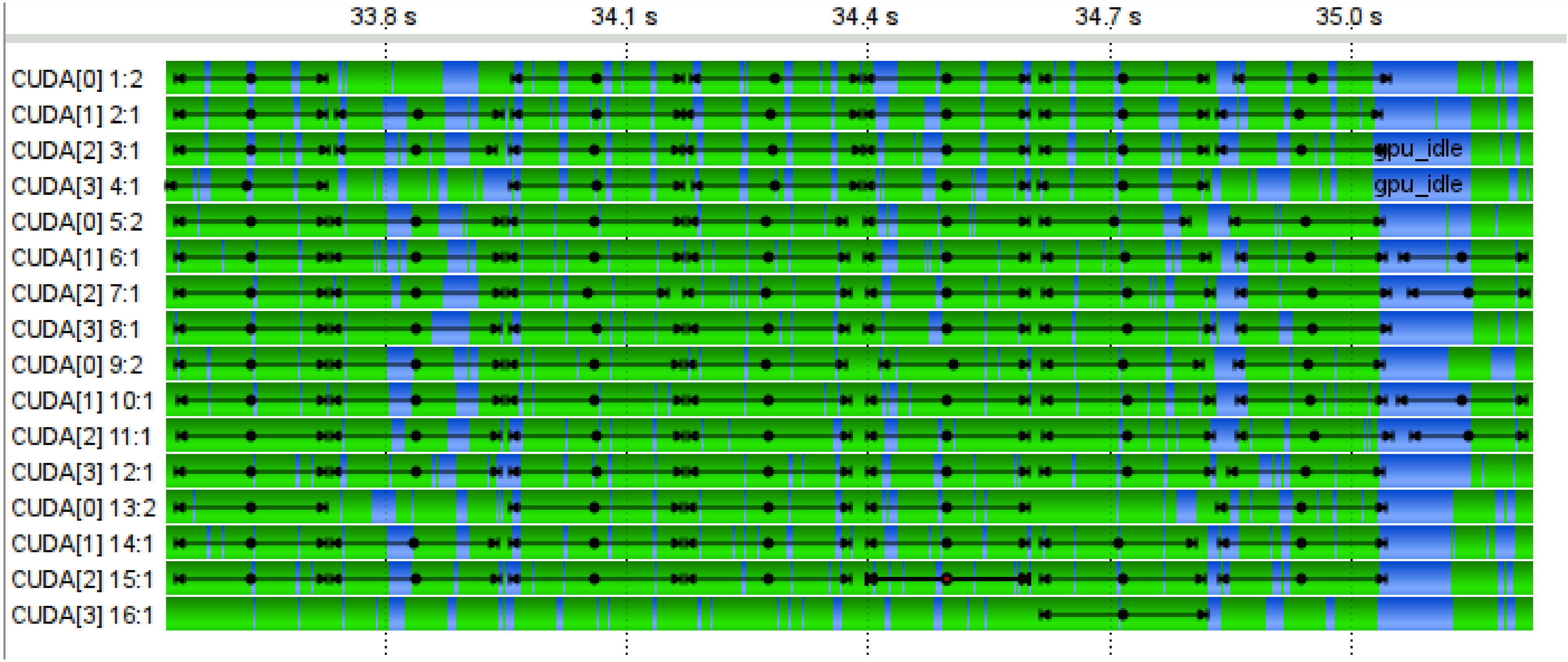
Low GPU Utilization

MPI + pthreads + CUDA


Vampir - Trace View - /home/juckel/PIConGPU/trace.otf
File Edit Chart Filter Window Help
0 s
114 s
114.311 s
Timeline
0 s 10 s 20 s 30 s 40 s 50 s 60 s 70 s 80 s 90 s 100 s 110 s
Process 0
CUDA[0] 1:2
CUDA[1] 2:1
CUDA[2] 3:1
CUDA[3] 4:1
Thread 5:1
Process 6
Process 7
Process 8
Process 9
Thread 9:3
Thread 10:2
Thread 11:2
Thread 12:2
CUDA[0] 13:2
CUDA[1] 14:1
CUDA[2] 15:1
CUDA[3] 16:1
CUDA[1] 6:1, CUDA[2] 7:1, CUDA[3] 8:1, CUDA[0] 9:2, Values of Metric "Active Threads" over Time
15.0 M
12.5 M
10.0 M
7.5 M
5.0 M
2.5 M
0.0 M
Function Legend
Application
CUDART_API
CUDA_IDLE
CUDA_KERNEL
CUDA_SYNC
MPI
NVSMI
VT_CUDA
Function Summary
All Processes, Accumulated Exclusive Time per Function
200 s
0 s
302.475 s deviceMoveAndMarkParticles<b1>
302.047 s deviceMoveAndMarkParticles<b0>
191.385 s DeviceDeleteParticles
121.802 s kernelComputeCurr…e0,ParticleType0>
118.368 s kernelComputeCurr…e0,ParticleType1>
66.726 s DeviceAddParticles
Function Summary
All Processes, Accumulated Exclusive Time per Function Group
NVSMI [351.089 s]
CUDA_KERNEL [1,234.253 s]
MPI [490.032 s]
CUDA_IDLE [580.704 s]
CUDA_SYNC [1,138.353 s]
Context View
Counter Data Timeline

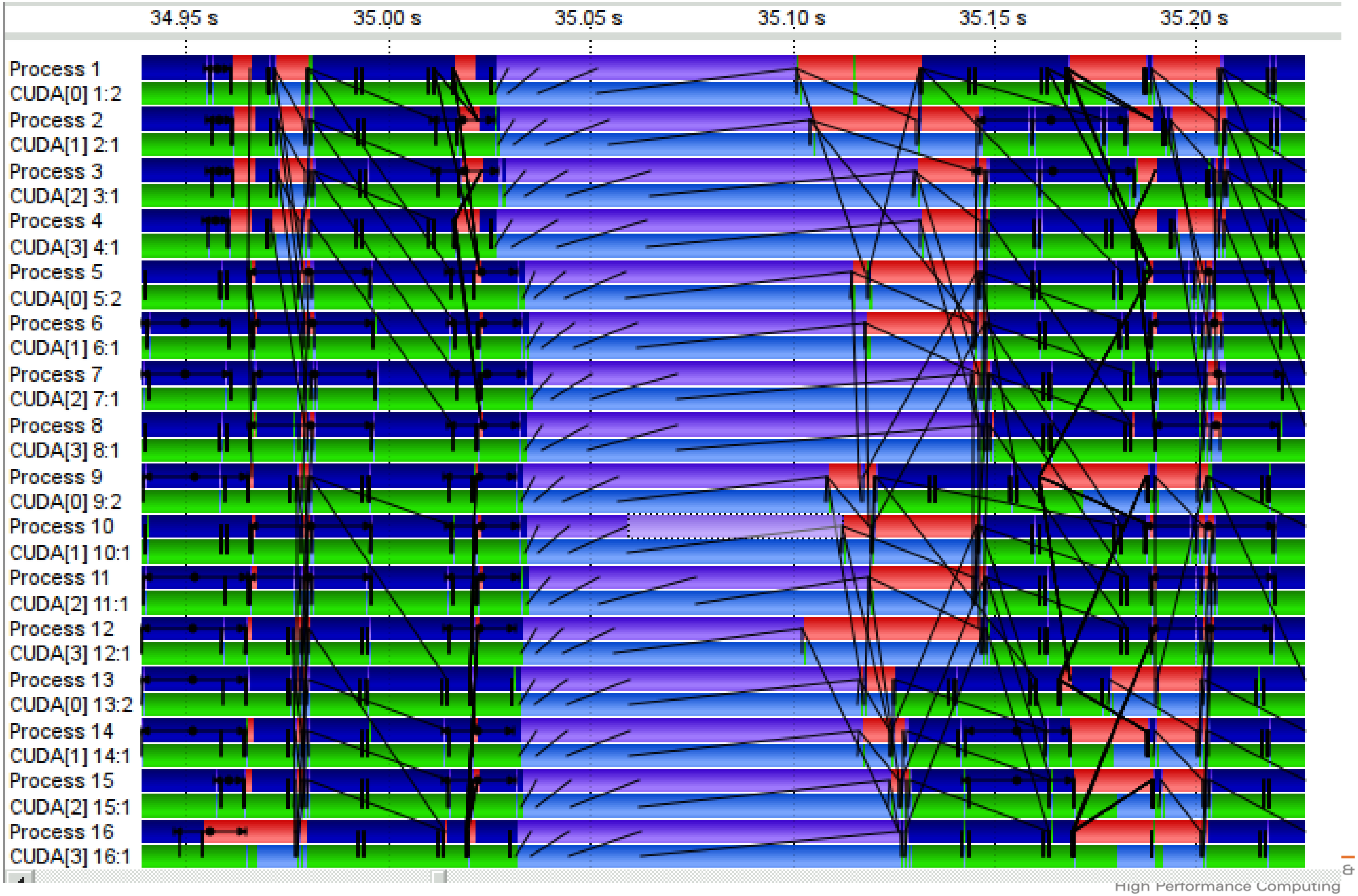
Property	Value
Display	Counter Data Timeline
Type	Counter
Locations	CUDA[1] 6:1, CUDA[2] 7:1, CUDA[3] 8:1, CUDA[0] 9:2
Maximum	15.74528 M
Maximum Locations	CUDA[2] 7:1, CUDA[1] 6:1, CUDA[3] 8:1, CUDA[0] 9:2
Average	2.897759 M
Minimum	0
Minimum Locations	CUDA[2] 7:1, CUDA[1] 6:1, CUDA[3] 8:1, CUDA[0] 9:2

What are the GPUs doing

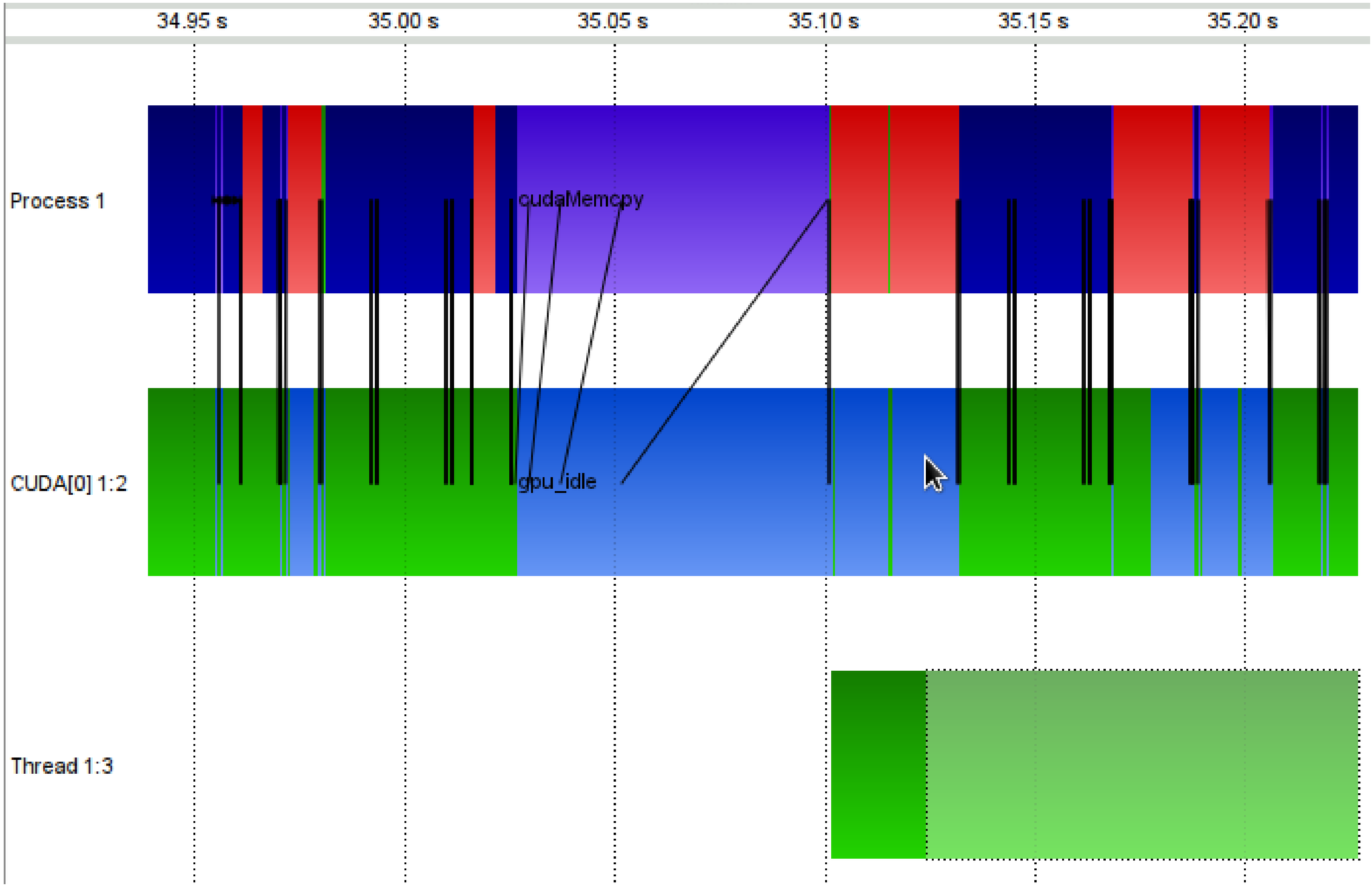


Green: Running a kernel Blue: Idle

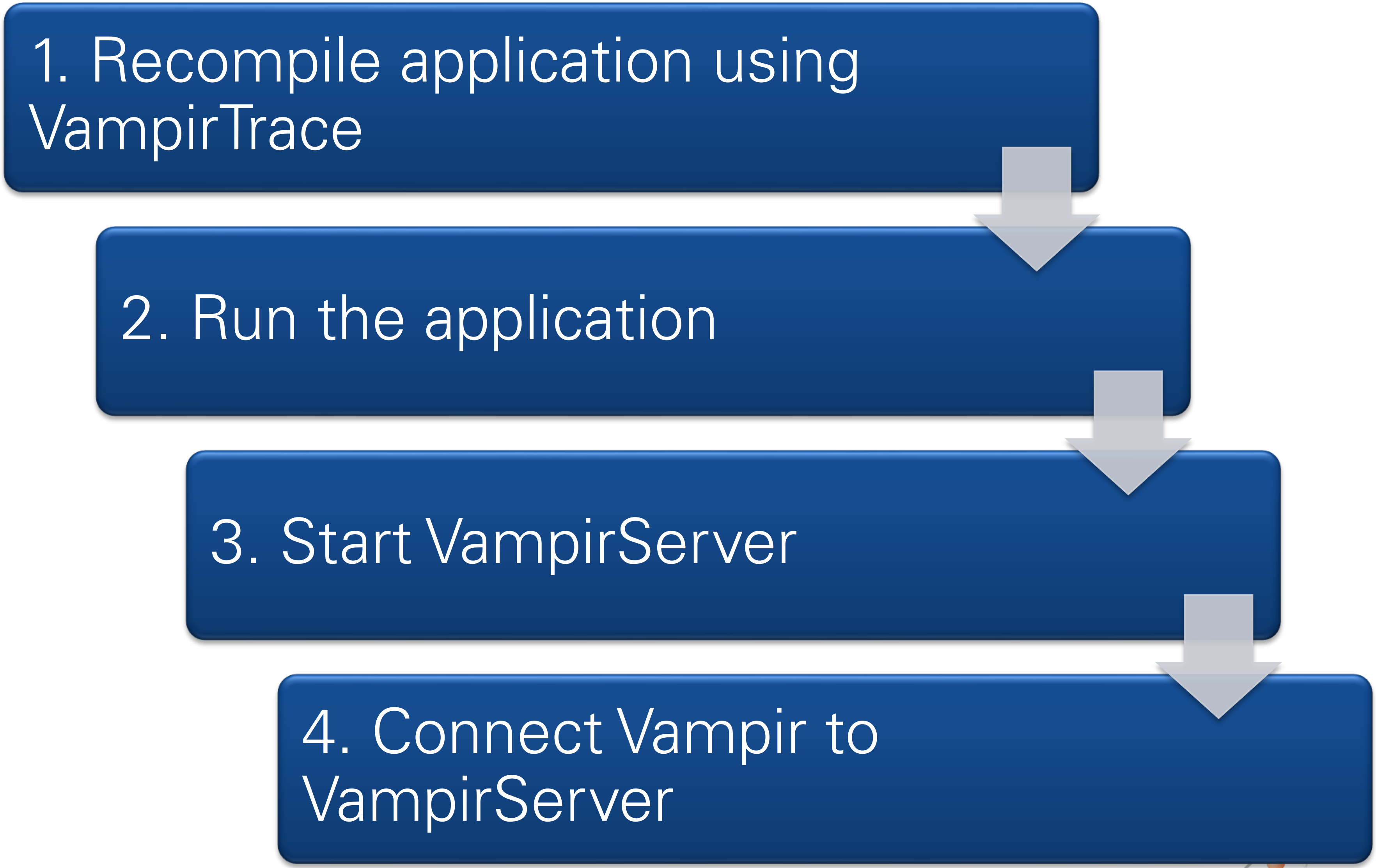
Connection to host processes



One Process & Thread Group



Fours steps to get you going at ORNL



1. Recompile application using VampirTrace

2. Run the application

3. Start VampirServer

4. Connect Vampir to VampirServer

Step 1. Recompile application using VampirTrace

First: `module load vampirtrace`

Use the appropriate compiler wrappers

- vtcc, vtCC, vtf77, vtf90
- Pick appropriate library (seq, mpi, mt, hyb)
  - e.g. `vtcc -vt:hyb` (recommended)
- Pick instrumentation type
  - `-vt:inst compinst` (default, compiler instrumentation, all functions)
  - `-vt:inst manual` (MPI, OpenMP, CUDA and manual)
- See what's going on behind `-vt:show` or `-vt:verbose`
- More details `-vt:help`

Tell your build system to use VampirTrace

- `./configure -with-CC="vtcc -vt:hyb"` …

Step 2. Running the application

Run your application as usual

Make sure the VampirTrace module is loaded

- Load module before and `qsub -V`
- `module load vampirtrace` in the runscript

Control VampirTrace by using environment variables

- `VT_PFORM_GDIR=traces`
  - `mkdir $VT_PFORM_GDIR`
  - `lfs setstripe -c 1 $VT_PFORM_GDIR`
  - place trace files in that directory
  - Use /lustre/widow1 for large runs

Step 2. Running the application (contd.)

Control VampirTrace by using environment variables

- `VT_FILTER_SPEC=default.filter`
  - `cp $VAMPIRTRACE_DIR/etc/default.filter .`
  - Reduces resulting trace size by filtering frequently called functions
- `VT_MAX_FLUSHES=20`
  - Defaults to 1
  - Allows VT to flush buffer during execution
- `VT_BUFFER_SIZE=100M`
  - Defaults to 32M
  - Increases internal buffer size
- `VT_SYNC_FLUSH=yes`
  - Use that if you have collective ops in your code
  - Avoids all processes waiting for one process to do trace I/O
  - But has overhead itsself (extra allreduce on all collective ops!)
- `VT_METRICS=PAPI_FP_OPS:PAPI_TOT_INS`

Step 3+4. Analyze small traces locally (< 100 MB)

Copy the Vampir GUI / Client to your workstation/laptop

- `scp home.ccs.ornl.gov:/sw/sources/vampir/client/vampir-7.*-i386.tar.gz .`
- `tar –xzf vampir-7.*-i386.tar.gz`
- `./vampir/bin/vampir`

Client is available for:

- Linux {i386, amd64}.tar.gz
- Windows {x86, x64}.exe
- MacOS as .dmg

Copy trace files to your workstation/laptop

- `scp -r jaguarpf.ccs.ornl.gov:/tmp/…/traces/ .`

Start Vampir on your workstation/laptop

- Open trace and enjoy!

TECHNISCHE UNIVERSITÄT DRESDEN

ZIH

Center for Information Services & High Performance Computing

Step 3. Start VampirServer

VampirServer runs on the compute nodes and needs access to the generated trace files (.otf, .z)

Currently available on jaguar, smoky, lens

```
$ module load vampir
   $ qsub -V -I -l size=120,walltime=2:0:0 -A <ACC>
   $ vngd-start.sh
```

Number of processes depends on (size of) the trace

- Try ¼ of the cores used by the application
- Running out of memory → add nodes
- Getting strange MPI errors → reduce process count

Step 4. Connect Vampir to VampirServer (Linux/Mac)

Open an SSH tunnel to the compute node

-
  ```
  jaguarpf$ vngd-start.sh
  Launching VampirServer Version 2.3.0 with  on 12 processes ...
  Found license file: /tmp/work/tilsche/.vampir/etc/lic.dat
  Running 11 analysis processes... (abort with Ctrl-C or vngd-shutdown)
  Server listens on: nid13928:30051
  ```

-
  ```
  workstation$ ssh –L 30051:nid13928:30051 jaguarpf.ccs.ornl.gov
  ```

Open Vampir and "remote open" to

- Server: `localhost`
- Port: `30051`

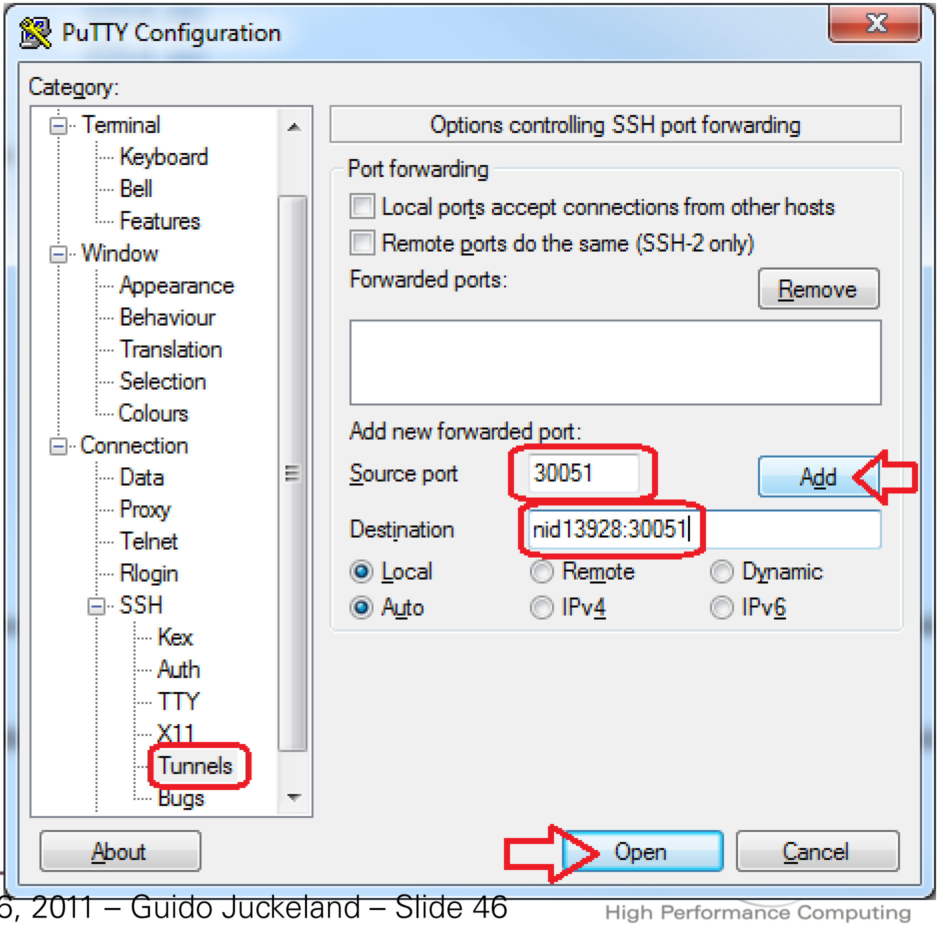
Step 4. Connect Vampir to VampirServer (Windows)

We need an SSH tunnel to the compute node

```
jaguarpf$ vngd-start.sh
Launching VampirServer Version 2.3.0 with  on 12 processes ...
Found license file: /tmp/work/tilsche/.vampir/etc/lic.dat
Running 11 analysis processes... (abort with Ctrl-C or vngd-shutdown)
Server listens on: nid13928:30051
```

Vampir "remote open" to

- Server: `localhost`
- Port: `30051`



PuTTY Configuration

Options controlling SSH port forwarding

Port forwarding

☐ Local ports accept connections from other hosts

☐ Remote ports do the same (SSH-2 only)

Forwarded ports:

Add new forwarded port:

Source port: 30051

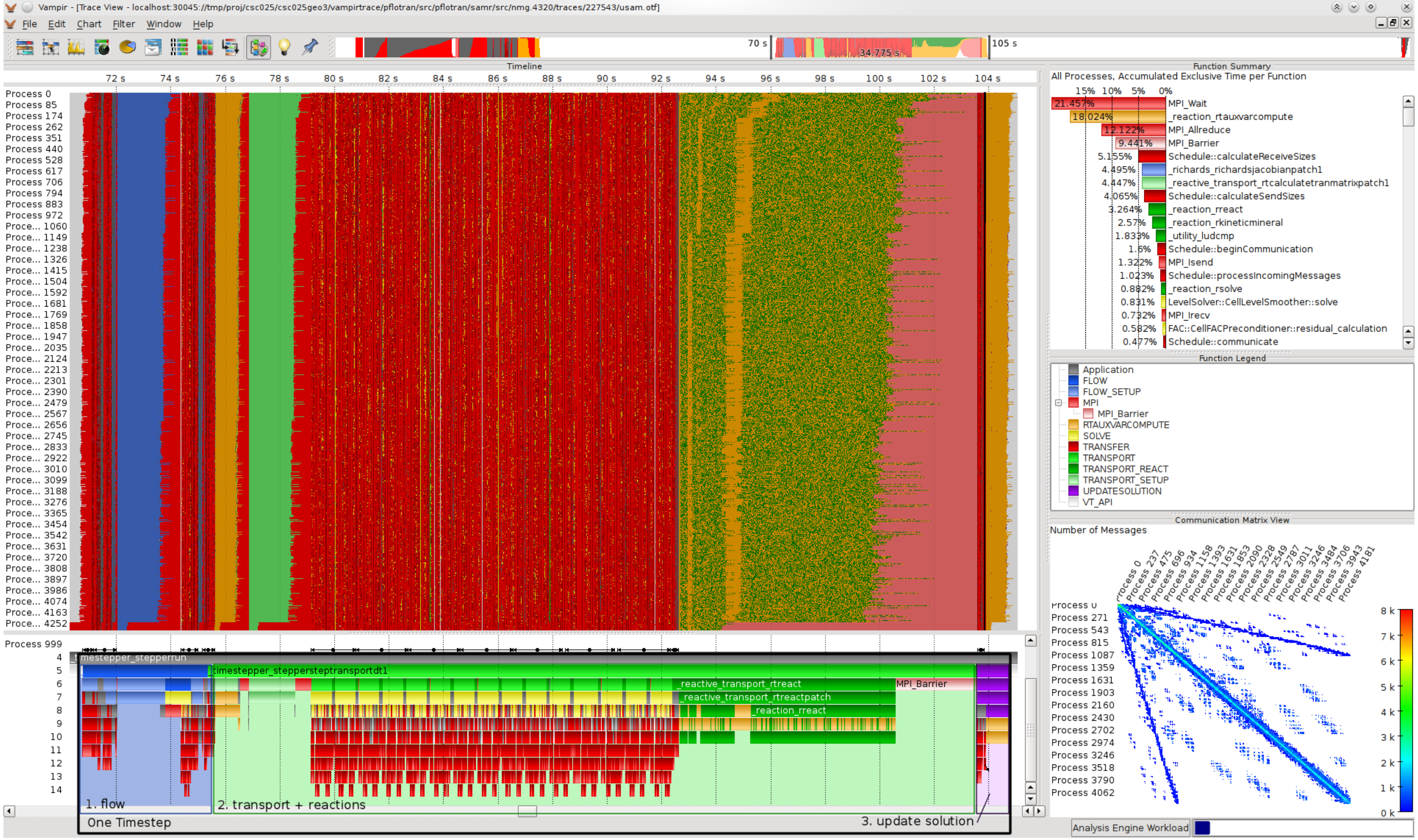
Destination: nid13928:30051

● Local ○ Remote ○ Dynamic

● Auto ○ IPv4 ○ IPv6

View your application trace:

"Play around" with Vampir to get a feeling about the features



Vampir - [Trace View - localhost:30045://tmp/proj/csc025/csc025geo3/vampirtrace/pflotran/src/pflotran/samr/src/nmg.4320/traces/227543/usam.otf]

Function Summary

All Processes, Accumulated Exclusive Time per Function

%	Function
21.457%	MPI_Wait
18.024%	_reaction_rtauxvarcompute
12.122%	MPI_Allreduce
9.441%	MPI_Barrier
5.155%	Schedule::calculateReceiveSizes
4.495%	_richards_richardsjacobianpatch1
4.447%	_reactive_transport_rtcalculatetranmatrixpatch1
4.065%	Schedule::calculateSendSizes
3.264%	_reaction_rreact
2.57%	_reaction_rkineticmineral
1.833%	_utility_ludcmp
1.6%	Schedule::beginCommunication
1.322%	MPI_Isend
1.023%	Schedule::processIncomingMessages
0.882%	_reaction_rsolve
0.831%	LevelSolver::CellLevelSmoother::solve
0.732%	MPI_Irecv
0.582%	FAC::CellFACPreconditioner::residual_calculation
0.477%	Schedule::communicate

Function Legend

- Application
- FLOW
- FLOW_SETUP
- MPI
  - MPI_Barrier
- RTAUXVARCOMPUTE
- SOLVE
- TRANSFER
- TRANSPORT
- TRANSPORT_REACT
- TRANSPORT_SETUP
- UPDATESOLUTION
- VT_API

Communication Matrix View

Number of Messages

1. flow

2. transport + reactions

3. update solution

One Timestep

Caution: Effects due to Tracing

I/O overhead (flush)

- Visibly marked in the trace
- ‘Long‘ time for I/O
- Ideally only once at the end (invisible) or during barriers
- Avoid by applying runtime filters

Measurement overhead

- Overhead on function calls
- Invisible
- Avoid instrumenting tiny frequently called functions
- Compare total runtime to get an upper bound on overhead

Summary

- **Vampir & VampirServer**
  - interactive trace visualization and analysis
  - intuitive browsing and zooming
  - scalable to “quite large” trace data sizes (1,5 TByte)
  - scalable to high parallelism (200,000 processes)
- Vampir available for Windows, Linux/Unix and Mac OS X

- **VampirTrace**
  - convenient instrumentation and measurement infrastructure
  - hides away complicated details
  - provides many options and switches for experts
- VampirTrace is part of Open MPI 1.3 and higher

Acknowledgements

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, Jens Domke, Holger Mickler, Daniel Hackenberg, Michael Heyde, Thomas Ilsche, 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

Further information about VampirTrace & Vampir

- www.nccs.gov/computing-resources/jaguar/software/?software=vampir
- vampir-7.4.0-OLCF3/doc/Manual.pdf
- www.vampir.eu
- /sw/sources/vampirtrace/5.11ornl/doc/UserManual.pdf
- www.tu-dresden.de/zih/vampirtrace

TECHNISCHE UNIVERSITÄT DRESDEN

ZIH
Center for Information Services & High Performance Computing