

# Non-Intel Tools at Claix

aiXcelerate 2017

Dirk Schmidl ([schmidl@itc.rwth-aachen.de](mailto:schmidl@itc.rwth-aachen.de))

## Simple to use tool to get an performance overview

- `mpiexec -> perf-report mpiexec ...`



```
Command: /opt/MPI/openmpi-1.10.4/linux/intel_16.0.2.181/bin/mpirun -x
OMP_NUM_THREADS=2 -np 4 jacobi.exe
Resources: 1 node (36 physical, 72 logical cores per node)
Memory: 252 GiB per node
Tasks: 4 processes, OMP_NUM_THREADS was 2
Machine: login.hpc.itc.rwth-aachen.de
Start time: Tue Dec 5 2017 09:48:42 (UTC+01)
Total time: 1 second
Full path: /rwthfs/rz/cluster/home/ds534486/Kurse/aixcelerate/C++-hyb-jacobi
```

### Summary: jacobi.exe is **Compute-bound** in this configuration

**Compute** 82.0% 

Time spent running application code. High values are usually good. This is **high**; check the CPU performance section for advice

**MPI** 18.0% 

Time spent in MPI calls. High values are usually bad. This is **low**; this code may benefit from a higher process count

**I/O** 0.0% 

Time spent in filesystem I/O. High values are usually bad. This is **negligible**; there's no need to investigate I/O performance

This application run was **Compute-bound**. A breakdown of this time and advice for investigating further is in the **CPU** section below. As little time is spent in **MPI** calls, this code may also benefit from running at larger scales.

# Allinea Performance Reports



## CPU

A breakdown of the 82.0% CPU time:

Single-core code	2.9%	
OpenMP regions	97.1%	
Scalar numeric ops	41.0%	
Vector numeric ops	0.0%	
Memory accesses	59.0%	

The per-core performance is **memory-bound**. Use a profiler to identify time-consuming loops and check their cache performance.

No time is spent in **vectorized instructions**. Check the compiler's vectorization advice to see why key loops could not be vectorized.

## I/O

A breakdown of the 0.0% I/O time:

Time in reads	0.0%	
Time in writes	0.0%	
Effective process read rate	0.00 bytes/s	
Effective process write rate	0.00 bytes/s	

No time is spent in I/O operations. There's nothing to optimize here!

## Memory

Per-process memory usage may also affect scaling:

Mean process memory usage	251 MiB	
Peak process memory usage	263 MiB	
Peak node memory usage	4.0%	

The **peak node memory usage** is very low. Running with fewer MPI processes and more data on each process may be more efficient.

## MPI

A breakdown of the 18.0% MPI time:

Time in collective calls	69.6%	
Time in point-to-point calls	30.4%	
Effective process collective rate	6.65 kB/s	
Effective process point-to-point rate	83.3 MB/s	

Most of the time is spent in **collective calls** with a **very low** transfer rate. This suggests load imbalance is causing synchronization overhead; use an MPI profiler to investigate.

The point-to-point transfer rate is **low**. This can be caused by inefficient message sizes, such as many small messages, or by imbalanced workloads causing processes to wait.

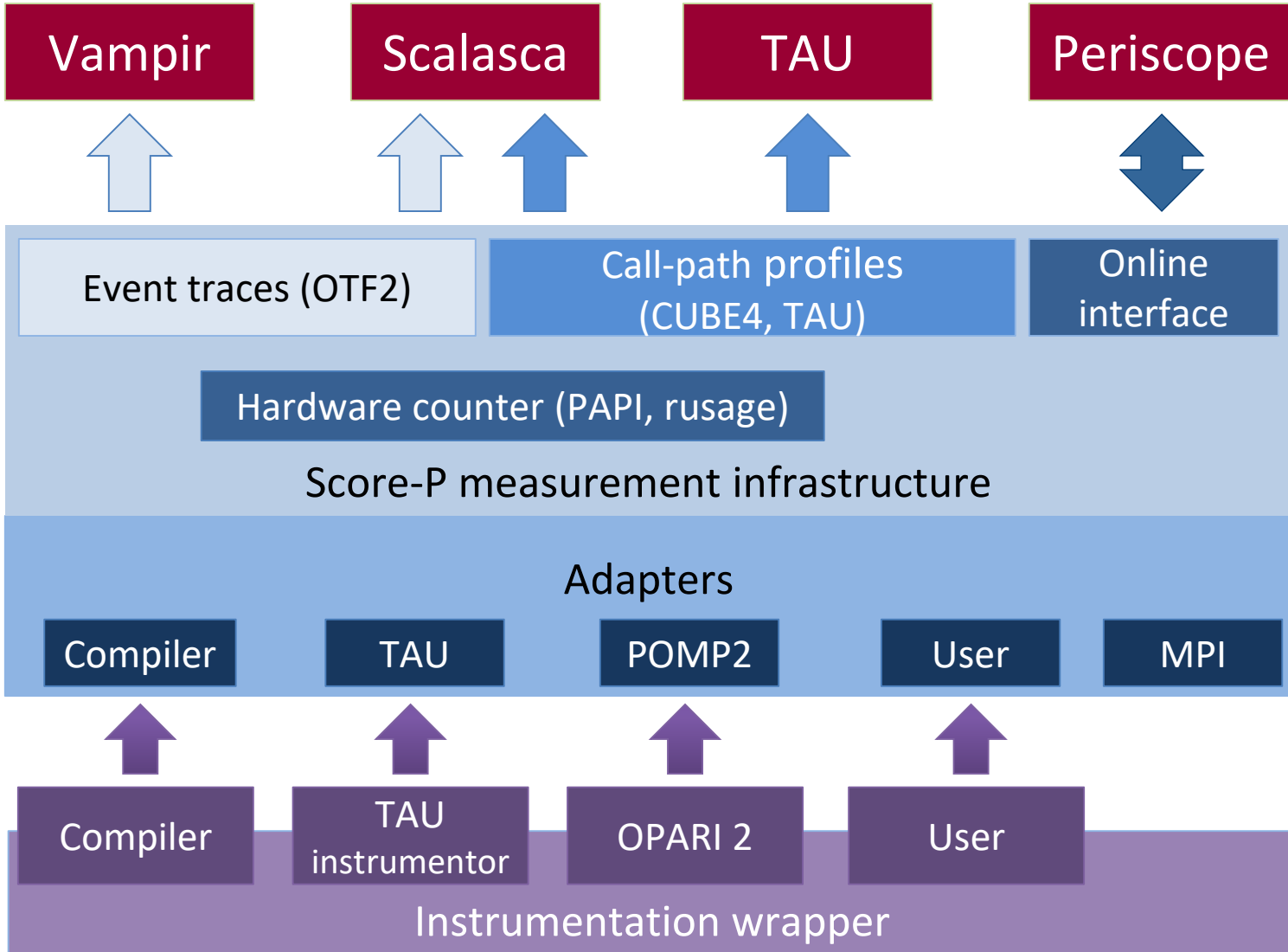
## OpenMP

A breakdown of the 97.1% time in OpenMP regions:

Computation	86.9%	
Synchronization	13.1%	
Physical core utilization	22.2%	
System load	25.0%	

**Physical core utilization** is low and some cores may be unused. Try increasing OMP\_NUM\_THREADS to improve performance.

# Score-P: A Unified Measurement System



UNIVERSITY OF OREGON

- **Supports Tracing and Profiling**
- **Uses direct instrumentation**
- **It also supports C/C++ and Fortran with MPI, OpenMP and hybrid codes.**
- **Useful for large scale applications.**

## Usage:

1. Precede your compiler command with *scorep*

`icc test.c -openmp -o a.out → scorep icc test.c -openmp -o a.out`

`mpicc test.c -openmp -o a.out → scorep mpicc test.c -openmp -o a.out`

2.a Run your application as usual to generate a profile

2.b Set `SCOREP_ENABLE_TRACING=true`, `SCOREP_ENABLE_PROFILING=false` and run the application for a trace.

3. Analyze the data in `scorep-XXXXXX`

## ■ Standard API to access hardware counters

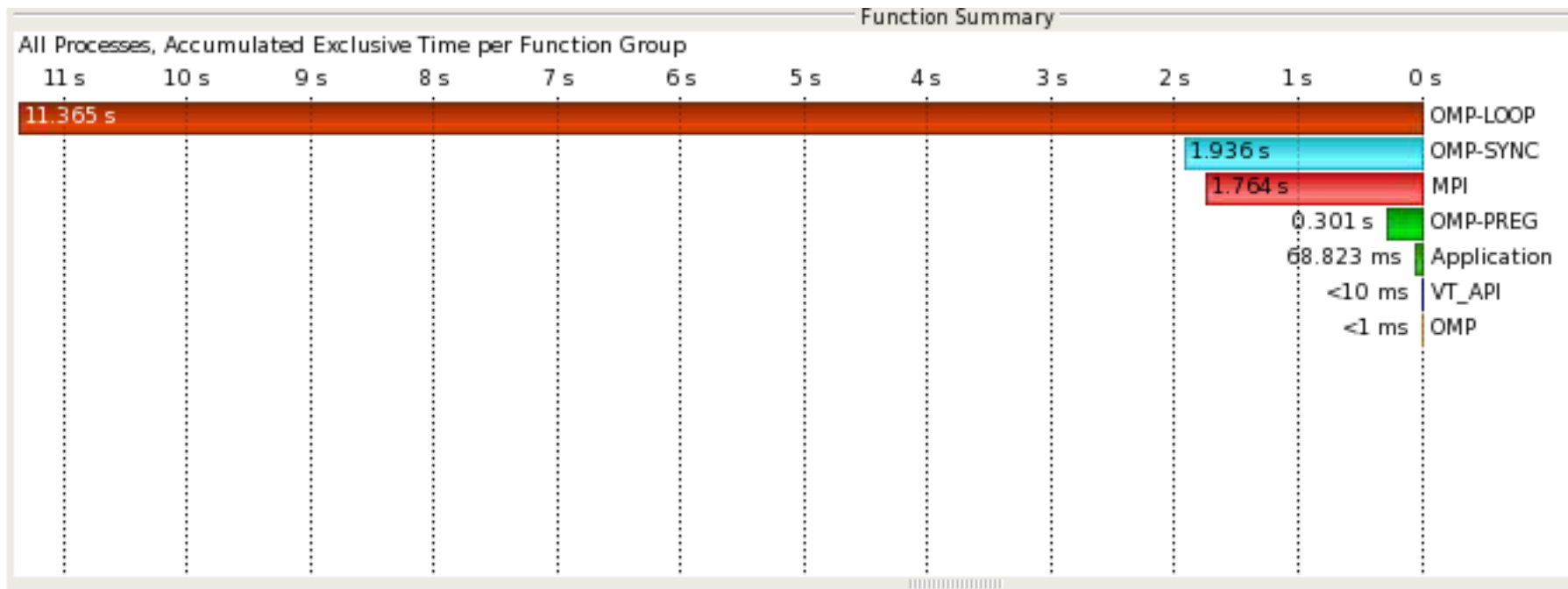
- provides access to a set of hardware counters with standardized names and over a standardized interface
- used in many tools for hardware counter access (also in Score-P)
- papi\_avail provides a list of available counters
- allows also to measure counters to get MFLOPS

Name	Code	Avail	Deriv	Description (Note)
PAPI_L1_DCM	0x80000000	Yes	No	Level 1 data cache misses
PAPI_L1_ICM	0x80000001	Yes	No	Level 1 instruction cache misses
PAPI_L2_DCM	0x80000002	Yes	No	Level 2 data cache misses
PAPI_L2_ICM	0x80000003	Yes	No	Level 2 instruction cache misses
PAPI_L3_DCM	0x80000004	No	No	Level 3 data cache misses
PAPI_L3_ICM	0x80000005	No	No	Level 3 instruction cache misses
PAPI_L1_TCM	0x80000006	Yes	Yes	Level 1 cache misses
PAPI_L2_TCM	0x80000007	Yes	No	Level 2 cache mis:
-----	-----	..	..	..

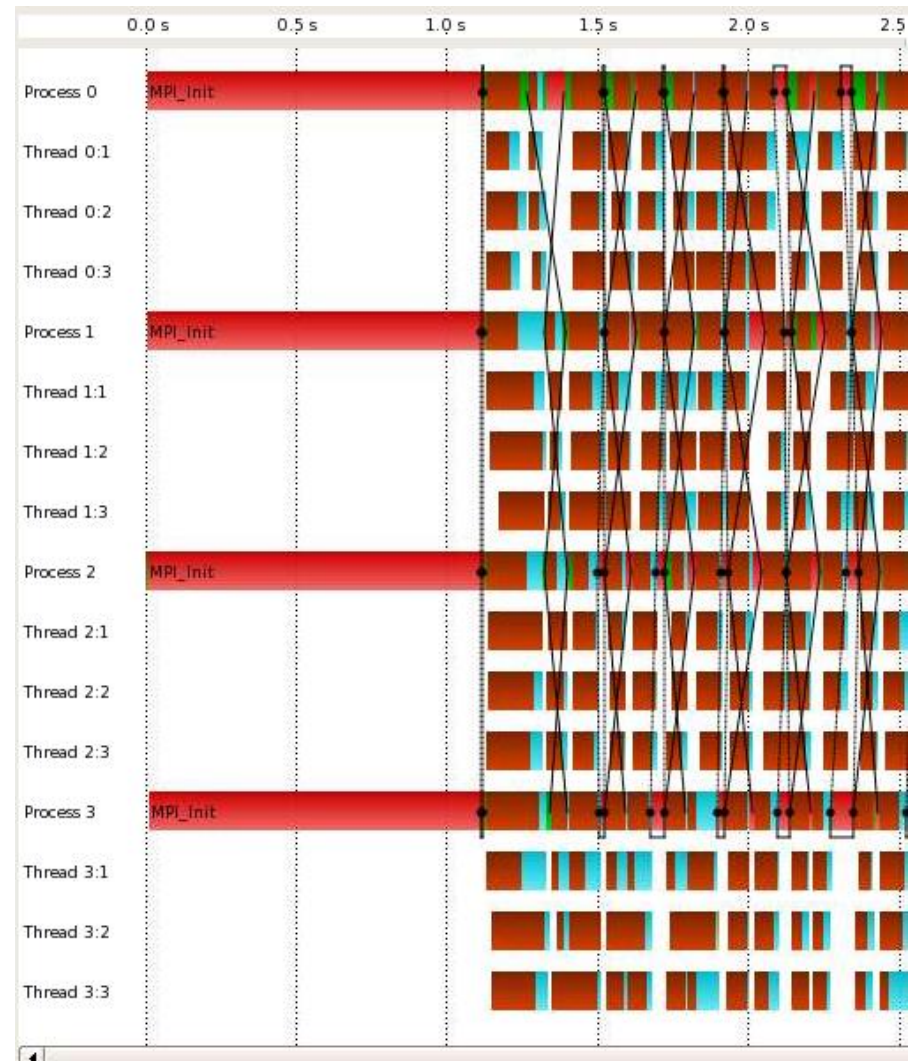
developed at:

THE UNIVERSITY of  
TENNESSEE

- Overview of the total time spend in functions.
- Time spend in MPI with sending or receiving messages can be seen.
- Change Event Category to “Function” to split the “Function Groups” and get more details.



- The Timeline gives a detailed view of all events.
- Regions and Messages of all Processes and Threads are shown.
- Zoom horizontal or vertical for more detailed information.
- Click on a message or region for specific details.

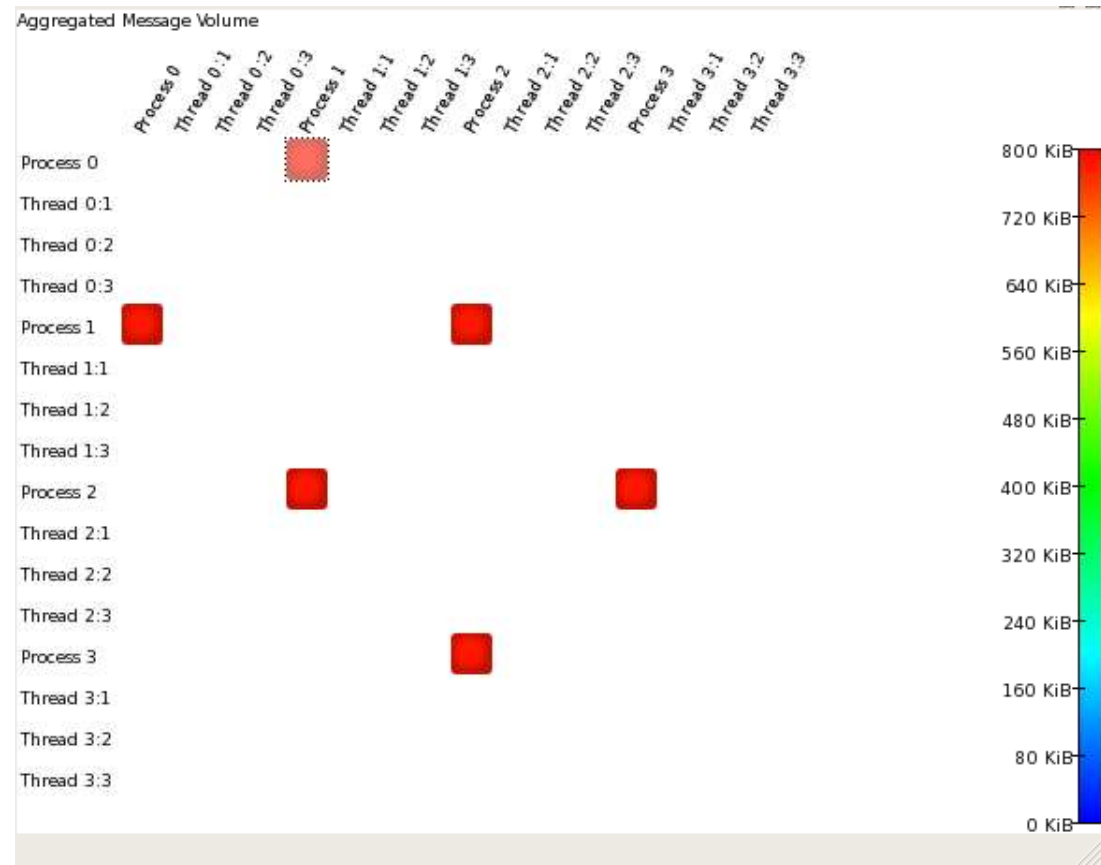




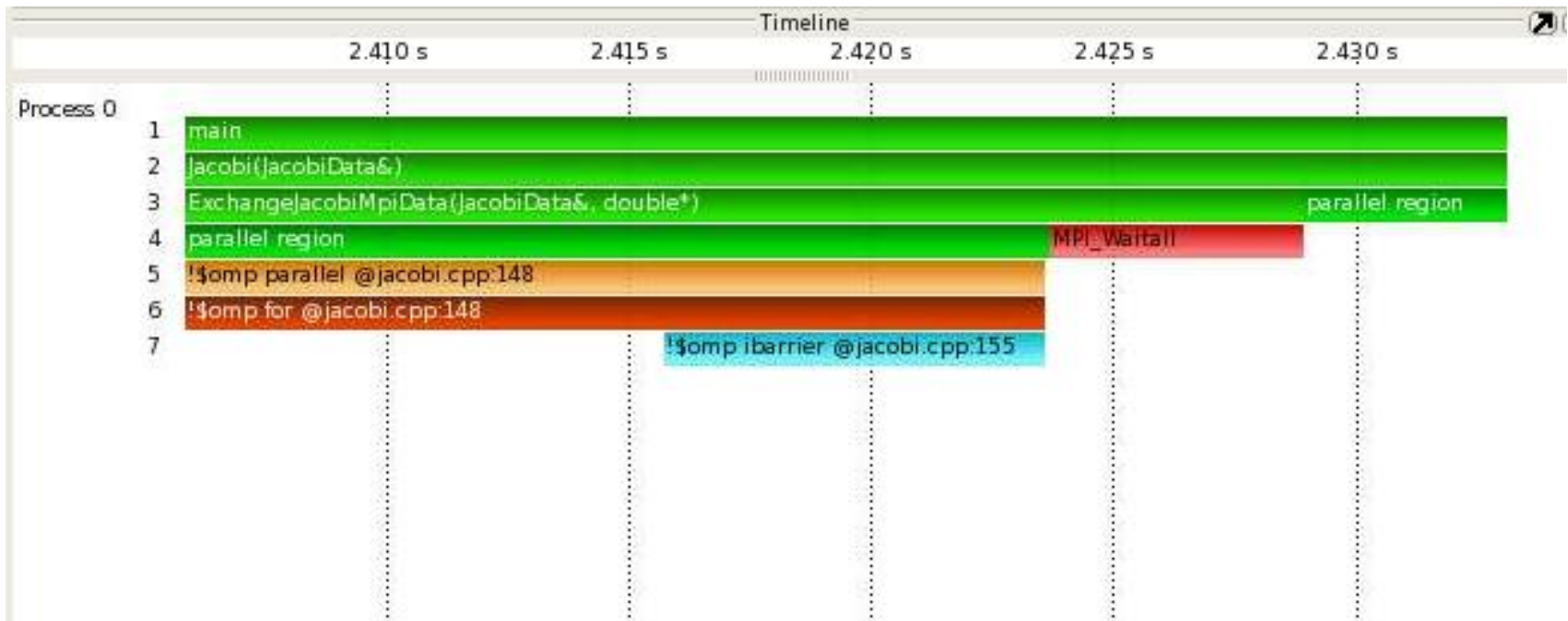
- MPI Communication Matrix for Point to Point Messages
- Overview over the communication behavior of the application
- Coupled with timeline view

## ■ Different Views:

- Aggregated message volume
- Min/Max message size
- Average message size
- Min/Max/Avg transfer time
- ....



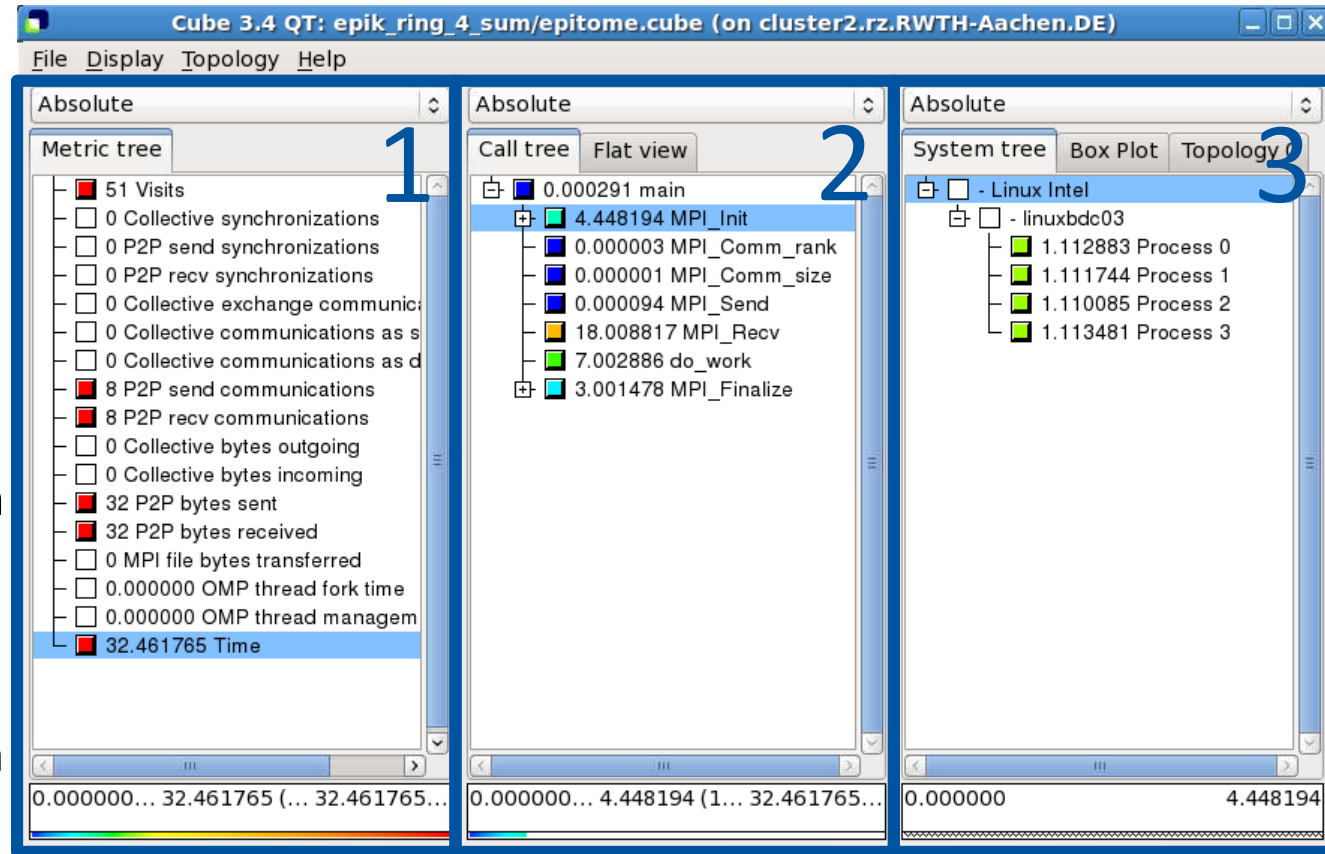
- The callstack can give even more information on the functions called on every thread/process.



1. Metric tree
2. Call tree
3. Topology tree

All views are coupled from left to right:

1. choose a metric  
-> this metric is shown for all functions
2. choose a function  
-> the right view shows the distribution over processes



Total execution time is 32 sec.



Out of these 4.4 sec. are spent in MPI\_Init().



Out of these 1.1 sec is spent by every process.

# Scalasca – Metric

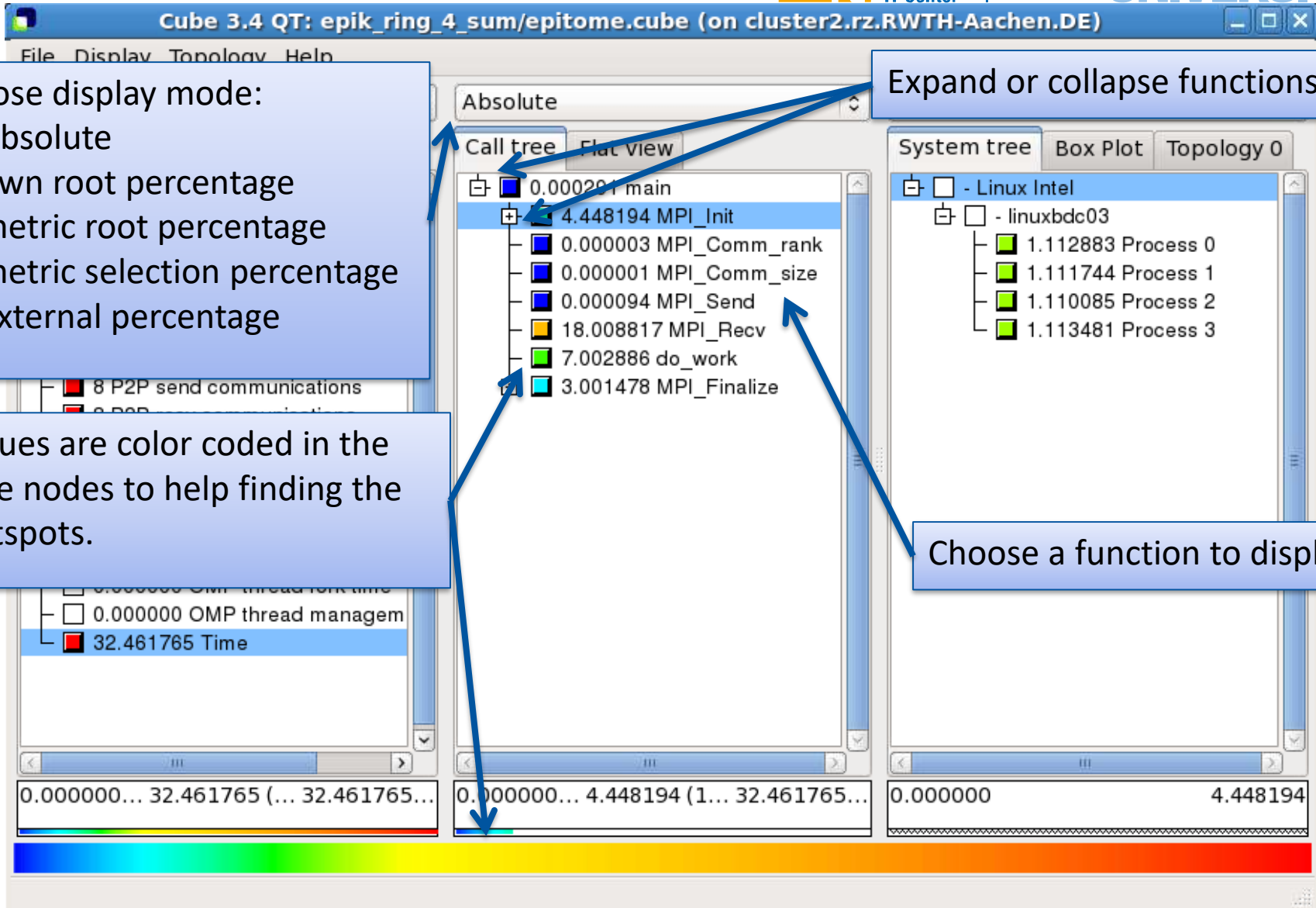
**Choose display mode:**

- Absolute
- own root percentage
- external percentage

**Choose metric to display, like time or number of visits.**

The screenshot shows the 'Metric tree' on the left with '51 Visits' and '32.461765 Time' selected. The middle panel shows a tree for '0.000291 main' with '4.448194 MPI\_Init' selected. The right panel shows a tree for '- Linux Intel' with '1.112883 Process 0' selected. The bottom of the interface features three data panels with numerical values and a color bar.

# Scalasca – Call Tree



Cube 3.4 QT: epik\_ring\_4\_sum/epitome.cube (on cluster2.rz.RWTH-Aachen.DE)

File Display Topology Help

Absolute

Call tree Flat view

- 0.000201 main
  - 4.448194 MPI\_Init
    - 0.000003 MPI\_Comm\_rank
    - 0.000001 MPI\_Comm\_size
    - 0.000094 MPI\_Send
    - 18.008817 MPI\_Recv
    - 7.002886 do\_work
    - 3.001478 MPI\_Finalize

System tree Box Plot Topology 0

- Linux Intel
  - linuxbdc03
    - 1.112883 Process 0
    - 1.111744 Process 1
    - 1.110085 Process 2
    - 1.113481 Process 3

8 P2P send communications

0.000000 OMP thread fork time

0.000000 OMP thread managem

32.461765 Time

0.000000... 32.461765 (... 32.461765...)

0.000000... 4.448194 (1... 32.461765...)

0.000000 4.448194

0.000000... 32.461765 (... 32.461765...)

0.000000... 4.448194 (1... 32.461765...)

0.000000 4.448194

Choose display mode:

- Absolute
- own root percentage
- metric root percentage
- metric selection percentage
- external percentage

Expand or collapse functions.

Values are color coded in the tree nodes to help finding the hotspots.

Choose a function to display.

# Scalasca – System View

Cube 3.4 QT: epik\_ring\_4\_sum/epitome.cube (on cluster2.rz.RWTH-Aachen.DE)

File Display Topology Help

Absolute

Call tree Flat view

- 0.000291 main
- 4.448194 MPI\_Init
- 0.000003 MPI\_Comm\_rank
- 0.000001 MPI\_Comm\_size
- 0.000094 MPI\_Send
- 18.008817 MPI\_Recv
- 7.002886 do\_work
- 3.001478 MPI\_Finalize

Absolute

System tree Box Plot Topology 0

- Linux Intel
  - linuxbdc03
    - 1.112883 Process 0
    - 1.111744 Process 1
    - 1.110085 Process 2
    - 1.113481 Process 3

0 0 Collective bytes incoming

32 32 P2P bytes sent

32 32 P2P bytes received

0 0 MPI file bytes transferred

0.000000 0.000000 OMP thread fork time

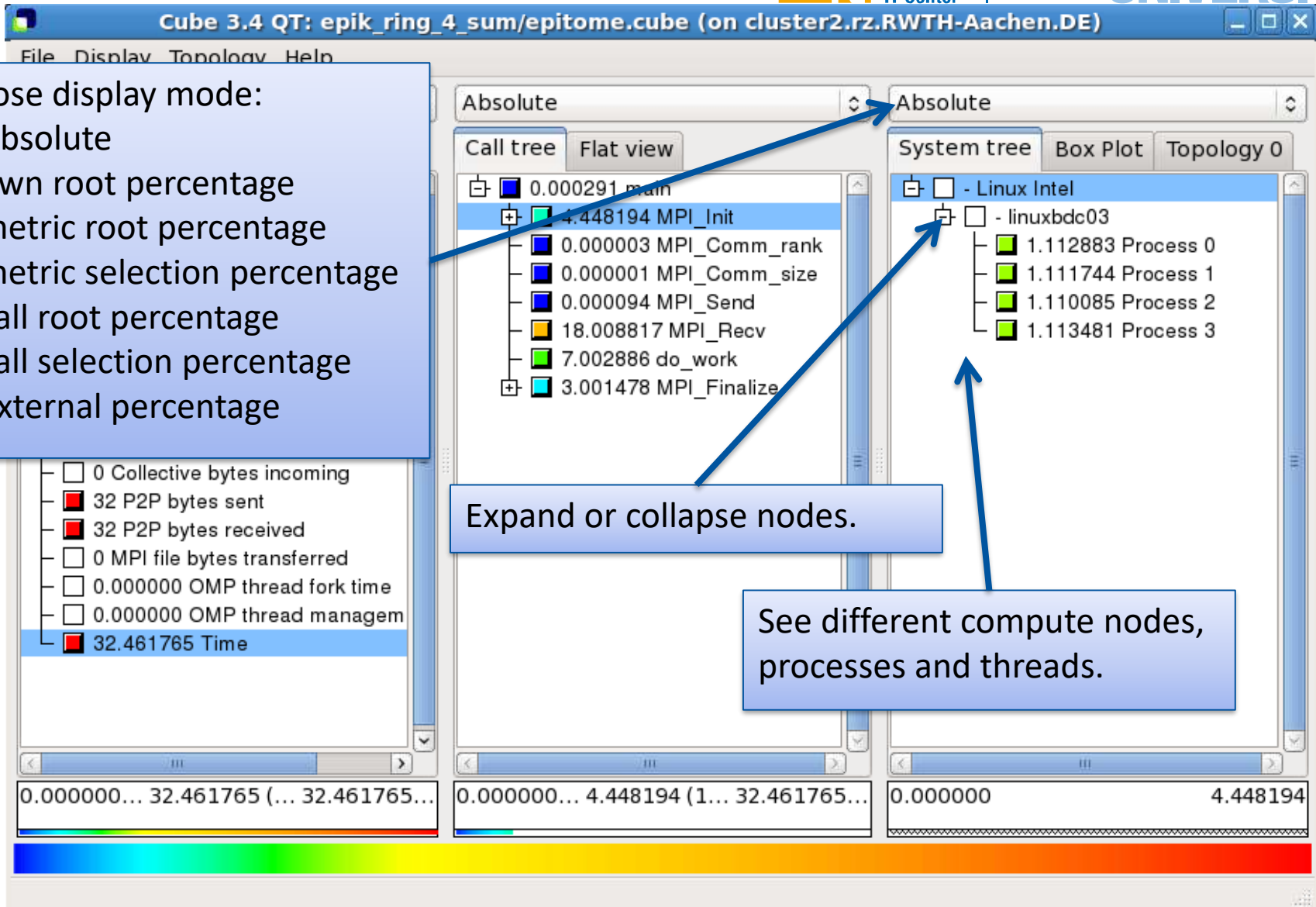
0.000000 0.000000 OMP thread managem

32.461765 32.461765 Time

0.000000... 32.461765 (... 32.461765...)

0.000000... 4.448194 (1... 32.461765...)

0.000000 4.448194



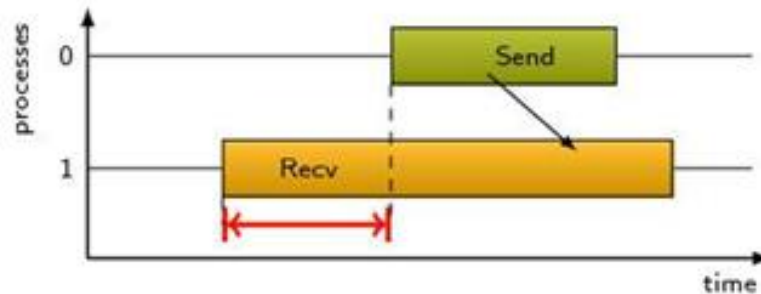
- Choose display mode:
- Absolute
  - own root percentage
  - metric root percentage
  - metric selection percentage
  - call root percentage
  - call selection percentage
  - external percentage

Expand or collapse nodes.

See different compute nodes, processes and threads.

- For large scale applications visualizing traces might be too much information.
- Aggregating everything in a profile might lose important information
- Scalasca allows to search for performance problems automatically.
- By rerunning the trace and comparing the time stamps, several situations can be automatically detected.

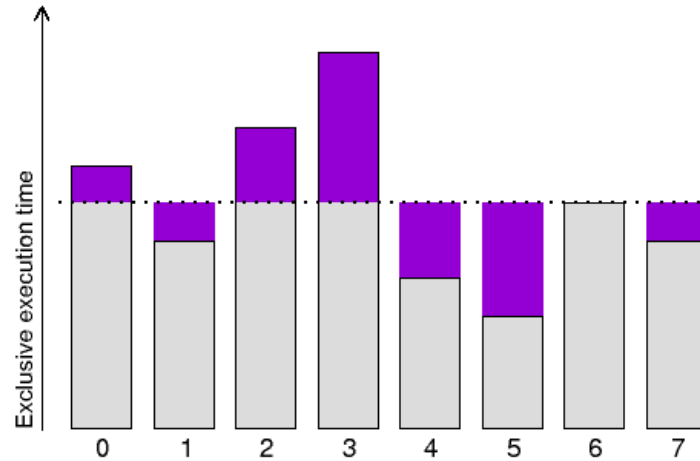
## Example: The late-sender pattern



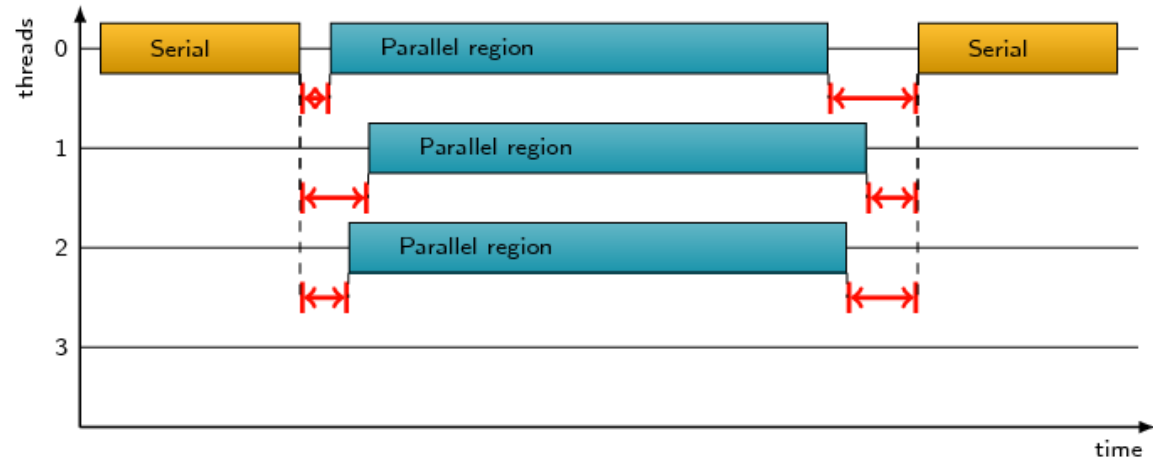
- The time lost through this problem (red arrow) is accumulated over the complete run and stored in a profile.

# Automatic Pattern Detection with Scalasca

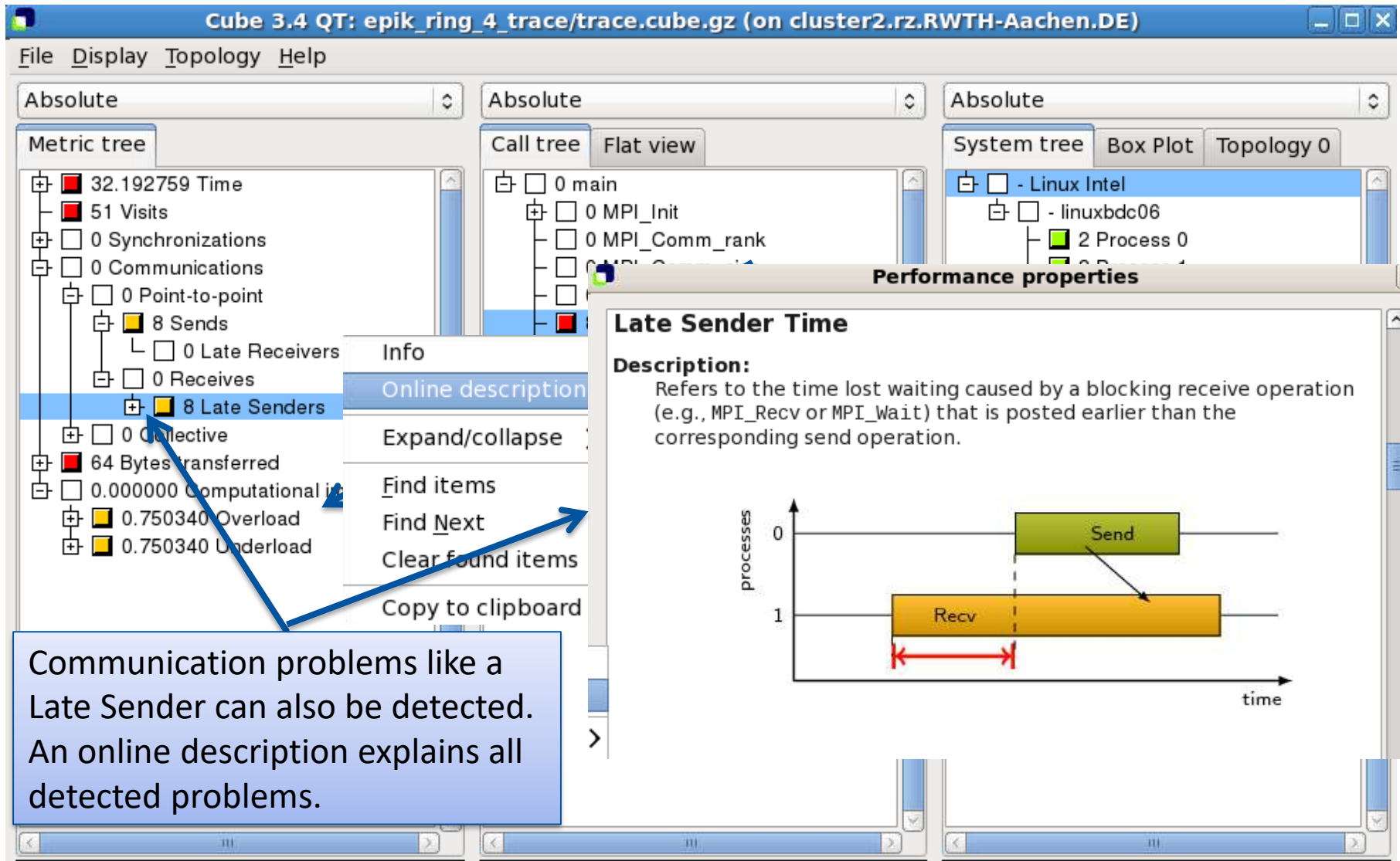
- Many patterns can be detected, like:  
Computational load imbalance:



## OpenMP management time:







Cube 3.4 QT: epik\_ring\_4\_trace/trace.cube.gz (on cluster2.rz.RWTH-Aachen.DE)

File Display Topology Help

Metric tree

- 32.192759 Time
- 51 Visits
- 0 Synchronizations
- 0 Communications
- 0 Point-to-point
  - 8 Sends
    - 0 Late Receivers
    - 0 Receives
    - 8 Late Senders
  - 0 Collective
- 64 Bytes transferred
- 0.000000 Computational
- 0.750340 Overload
- 0.750340 Underload

Call tree Flat view

- 0 main
  - 0 MPI\_Init
  - 0 MPI\_Comm\_rank

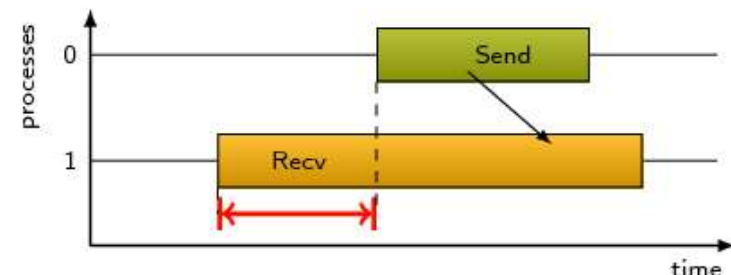
System tree Box Plot Topology 0

- Linux Intel
  - linuxbdc06
    - 2 Process 0

Performance properties

### Late Sender Time

**Description:**  
Refers to the time lost waiting caused by a blocking receive operation (e.g., MPI\_Recv or MPI\_Wait) that is posted earlier than the corresponding send operation.



processes

0

1

Send

Recv

time

Communication problems like a Late Sender can also be detected. An online description explains all detected problems.