Non-Intel Tools at Claix

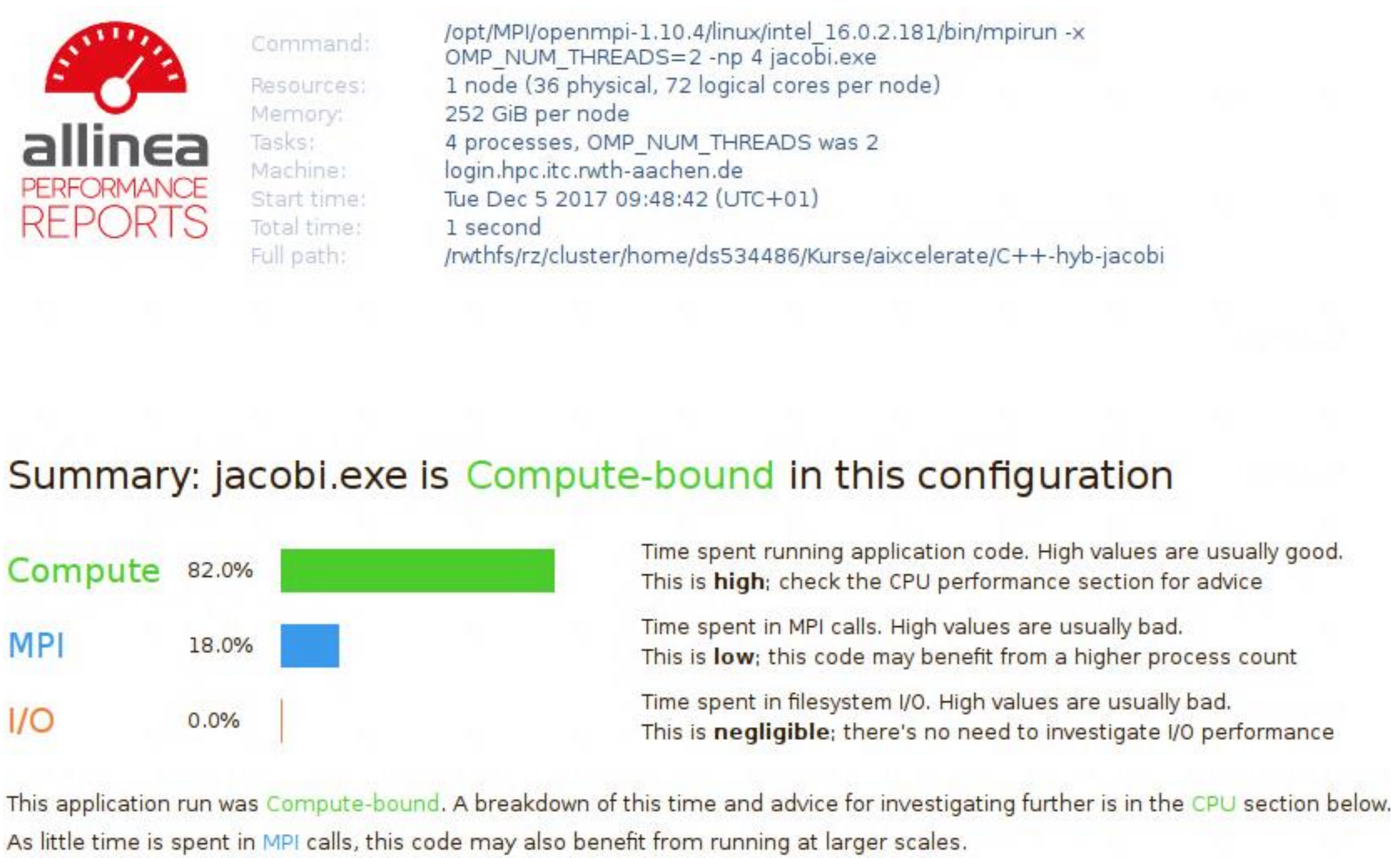
aiXcelerate 2017

Dirk Schmidl (schmidl@itc.rwth-aachen.de)

IT Center | RWTH AACHEN UNIVERSITY

Simple to use tool to get an performance overview

- **mpiexec -> perf-report mpiexec ...**



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.

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.

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!

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.

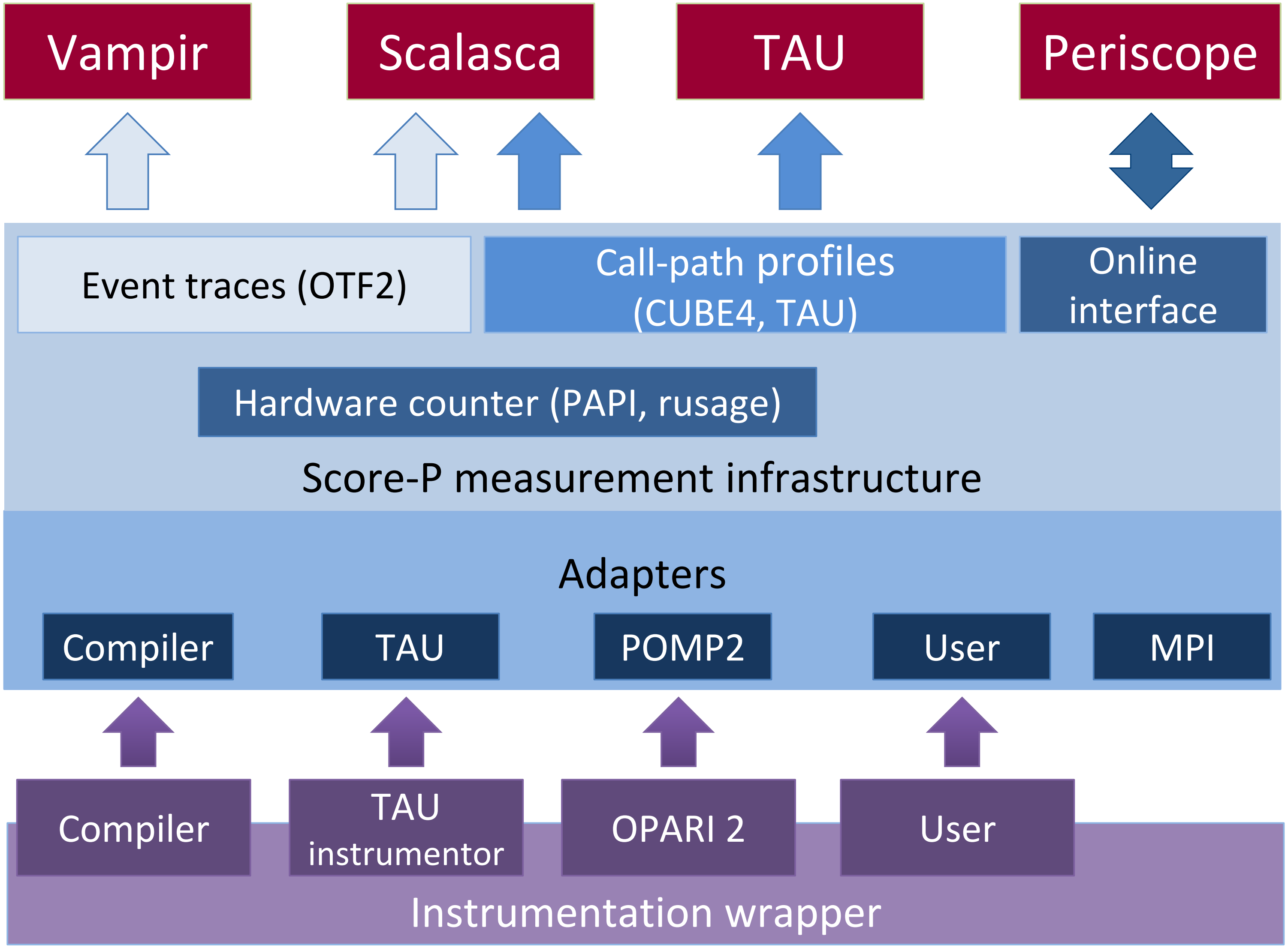
Memory

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.

Score-P: A Unified Measurement System



Score-P

- **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

Performance API (PAPI)

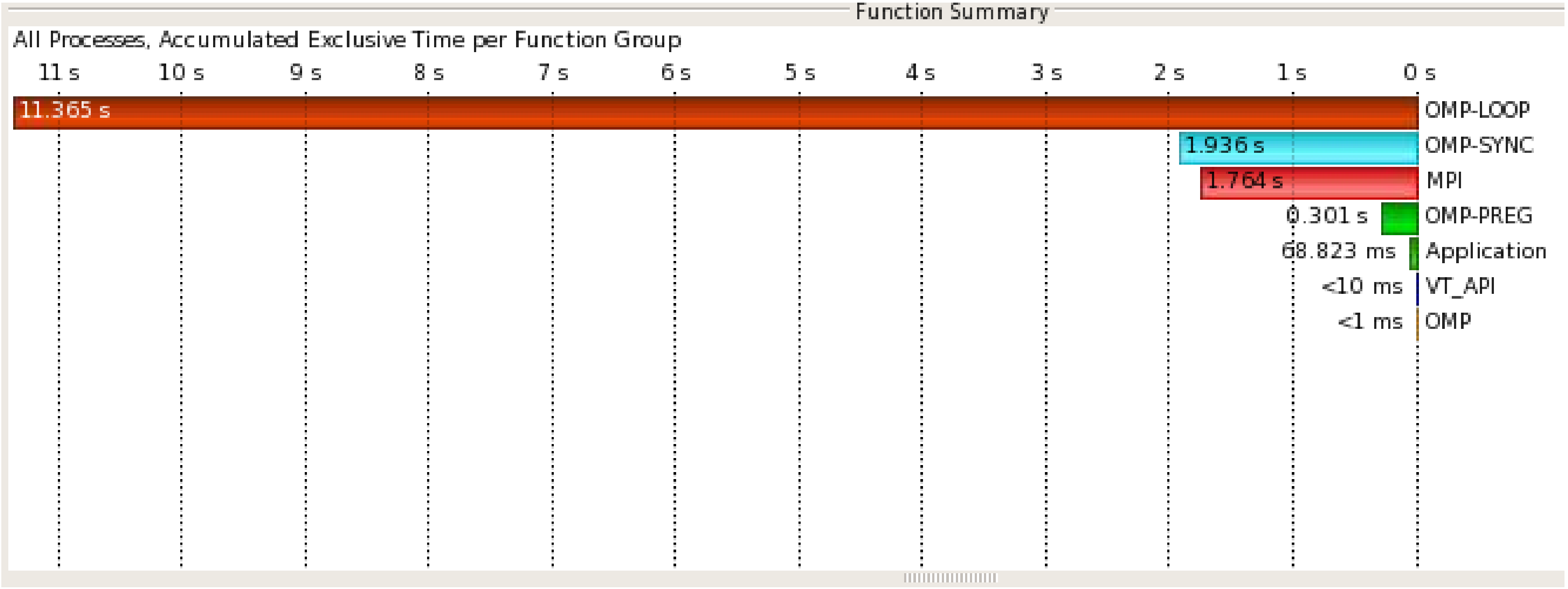
- **Standard API to access hardware counters**
  - → provides access to a set of hardware counters with standardized names and over a standardized interface
  - → used in many tools for hardware counter access (also in Score-P)
  - → papi_avail provides a list of available counters
  - → allows also 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

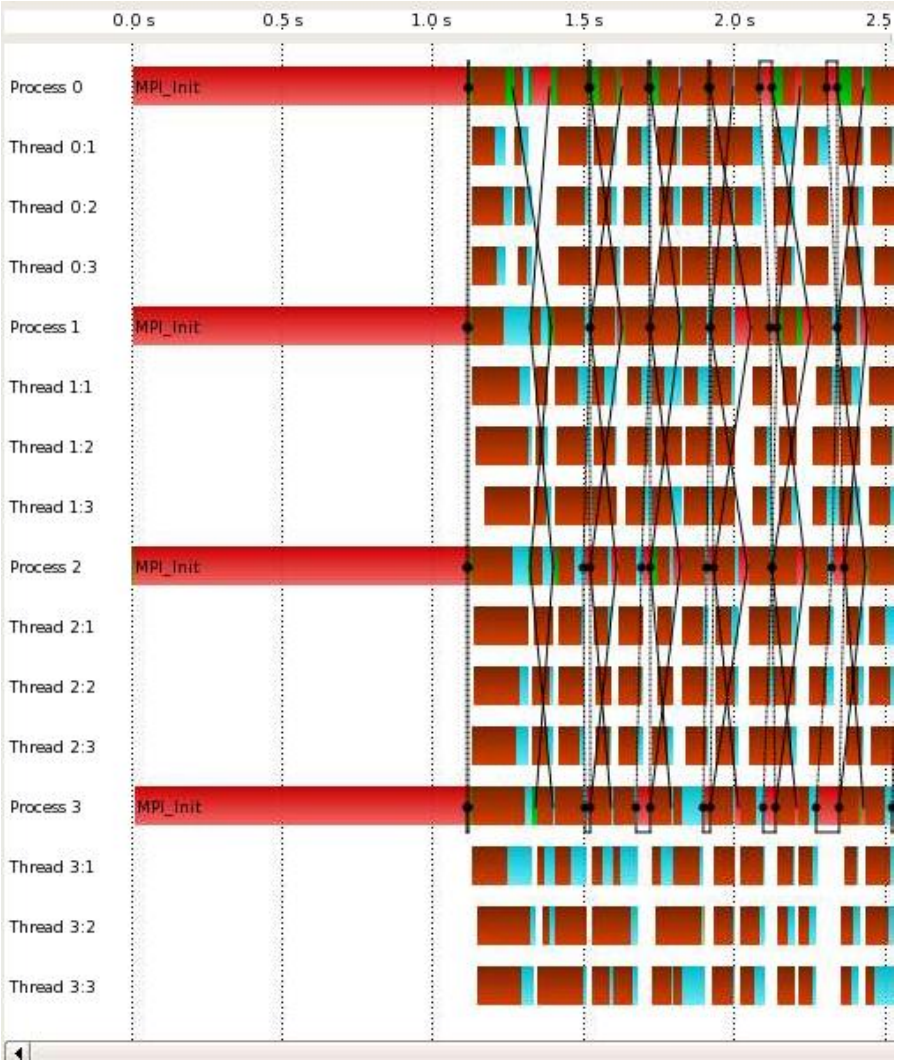
Vampir – Function Summary

- 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.



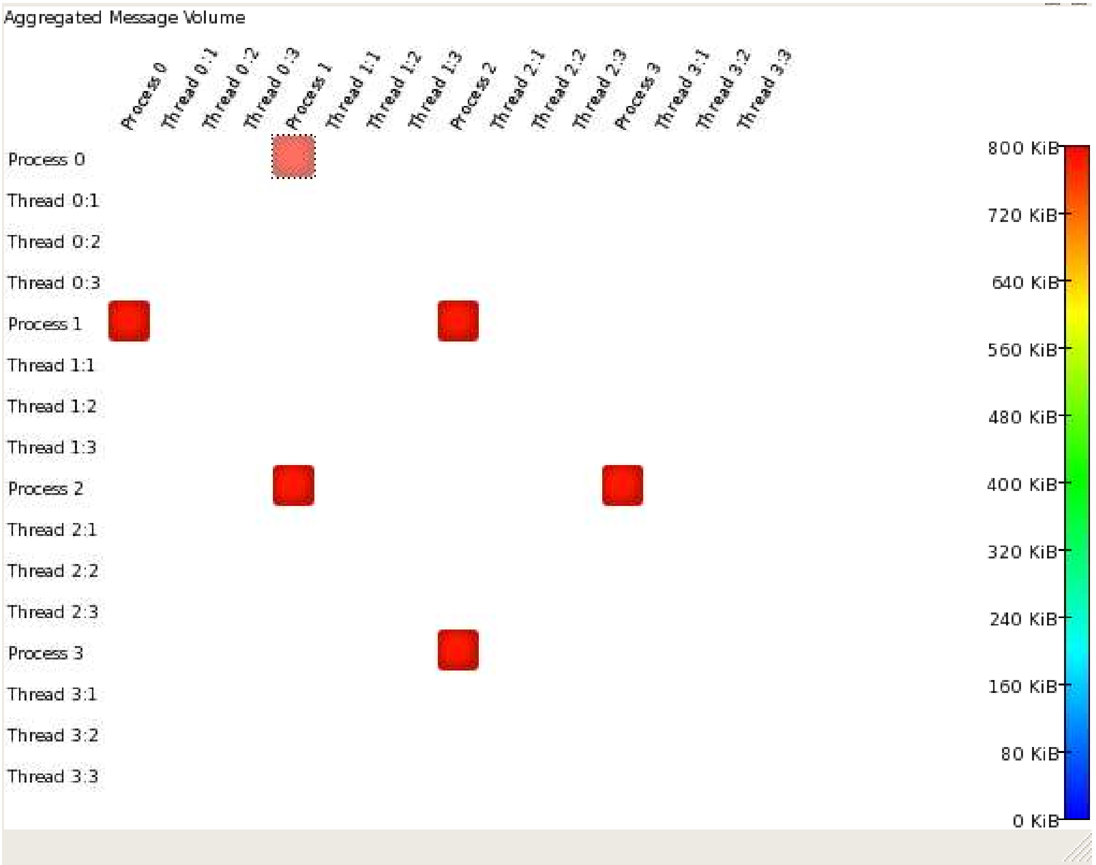
Vampir - Timeline

- **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.**



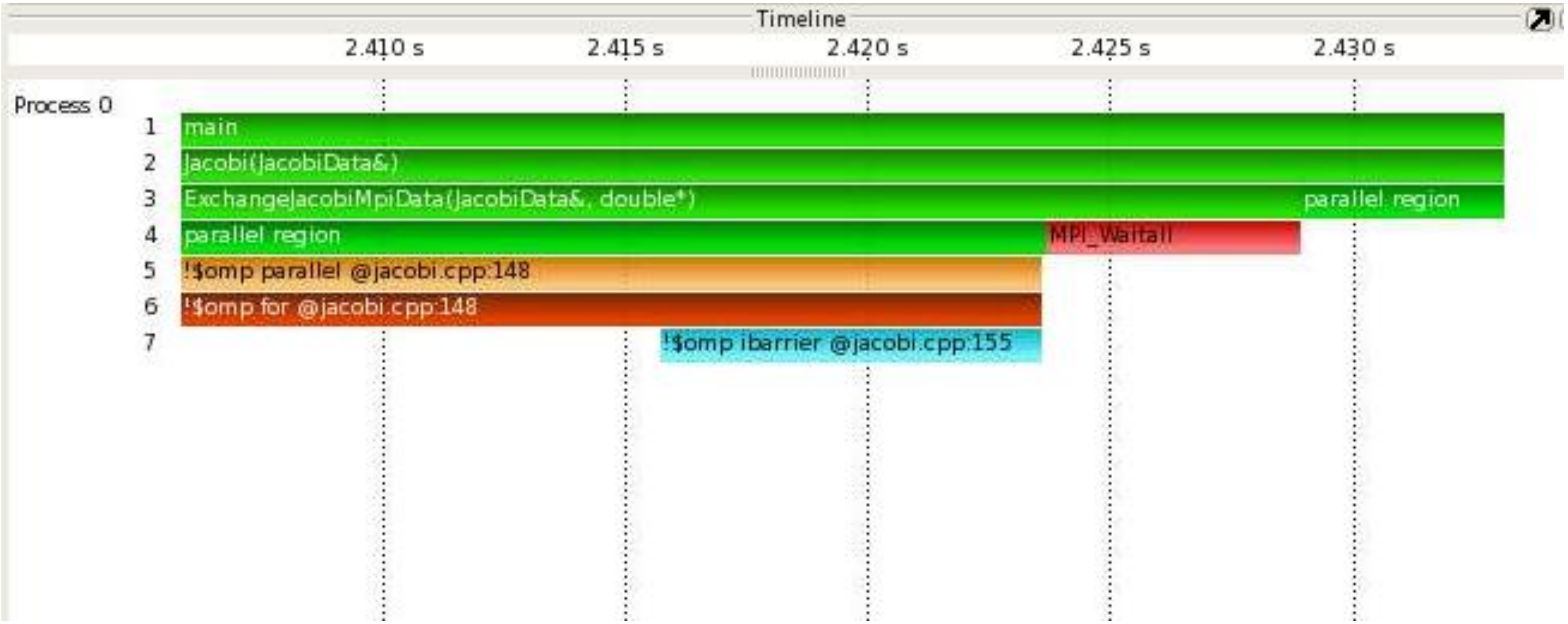
Vampir – Communication Matrix

- 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
  - → ….



Vampir – Process Timeline

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

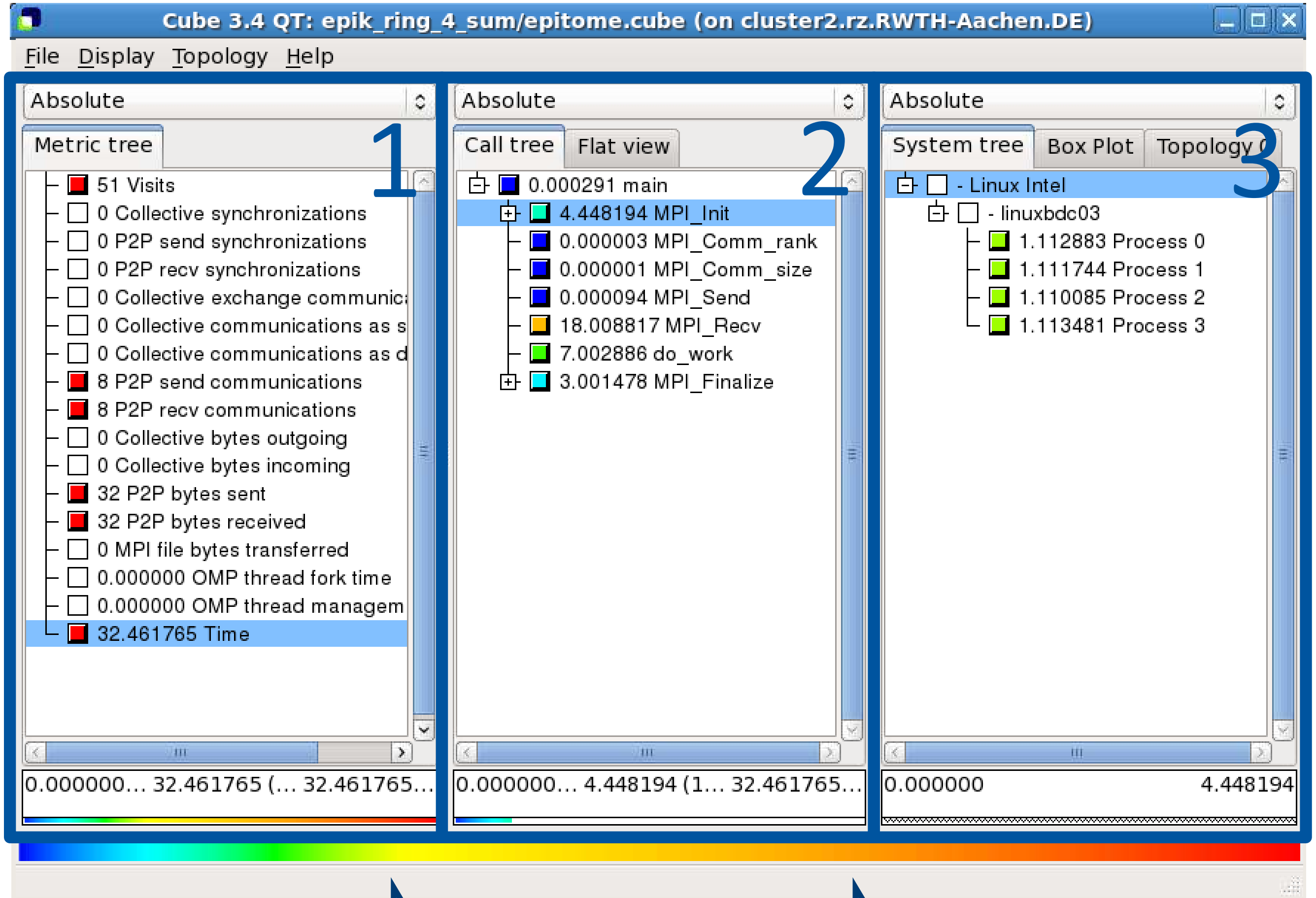


Scalasca – Cube Data Browser

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



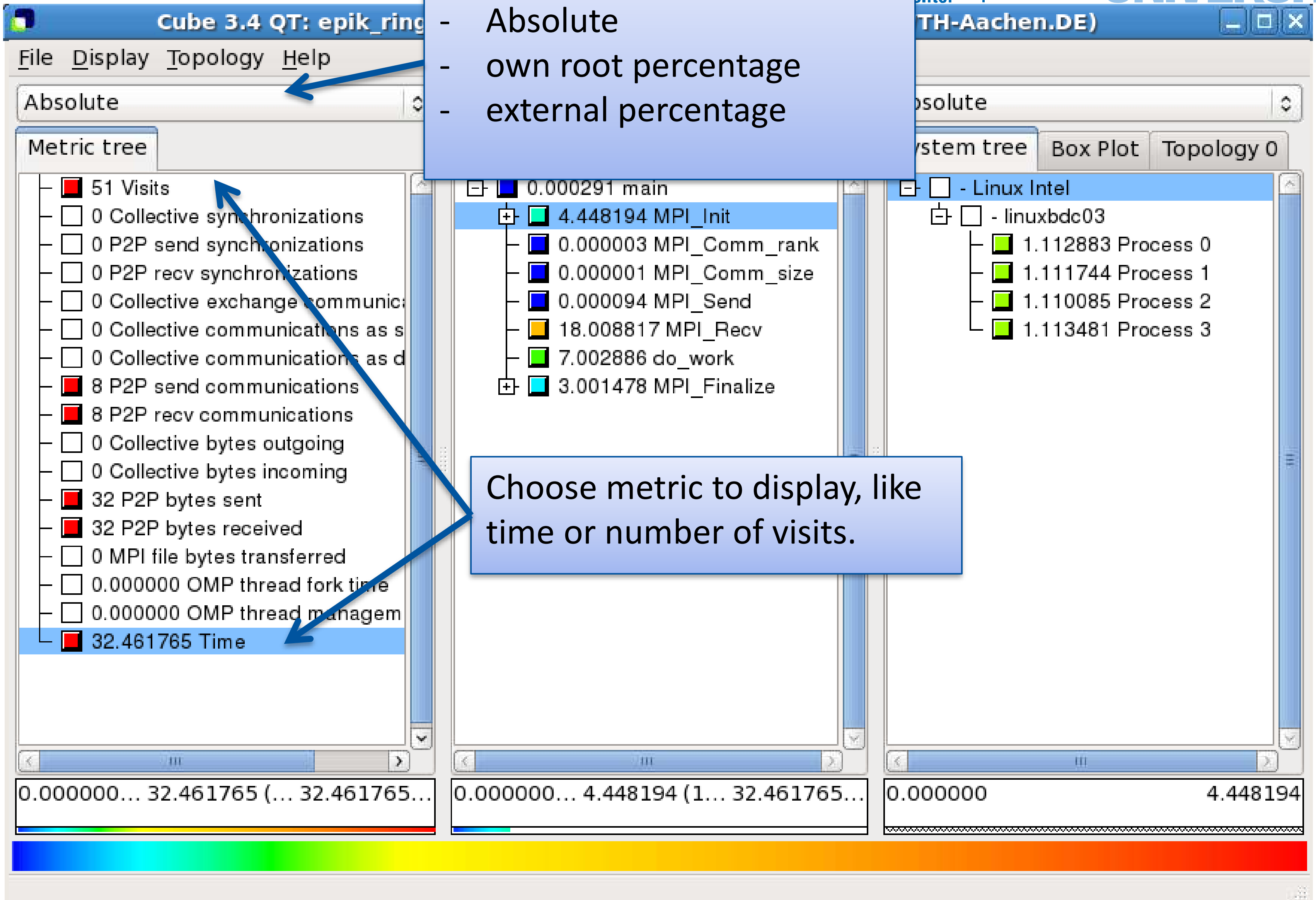
Total execution time is 32 sec.

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.

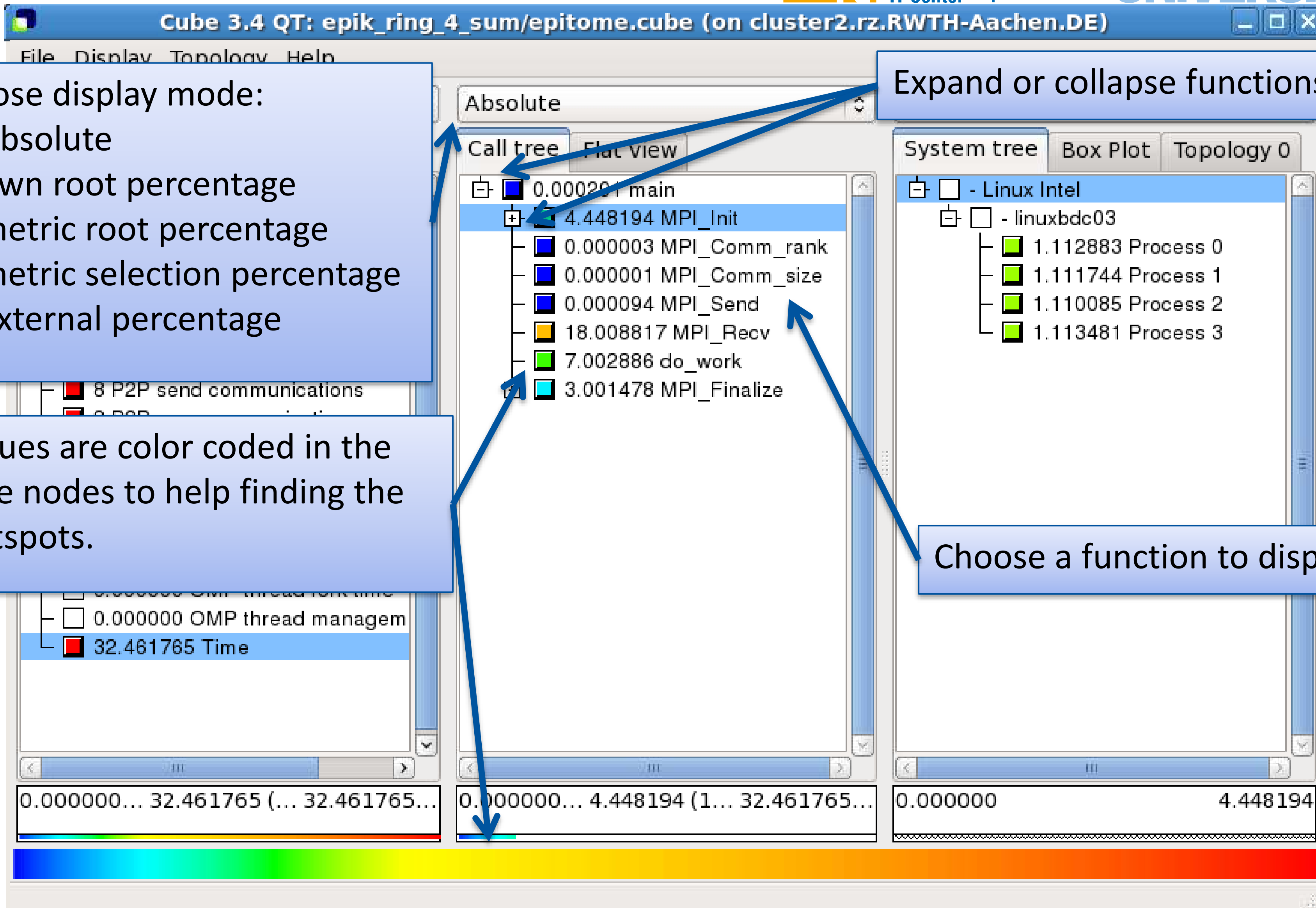
Scalasca – Call Tree

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

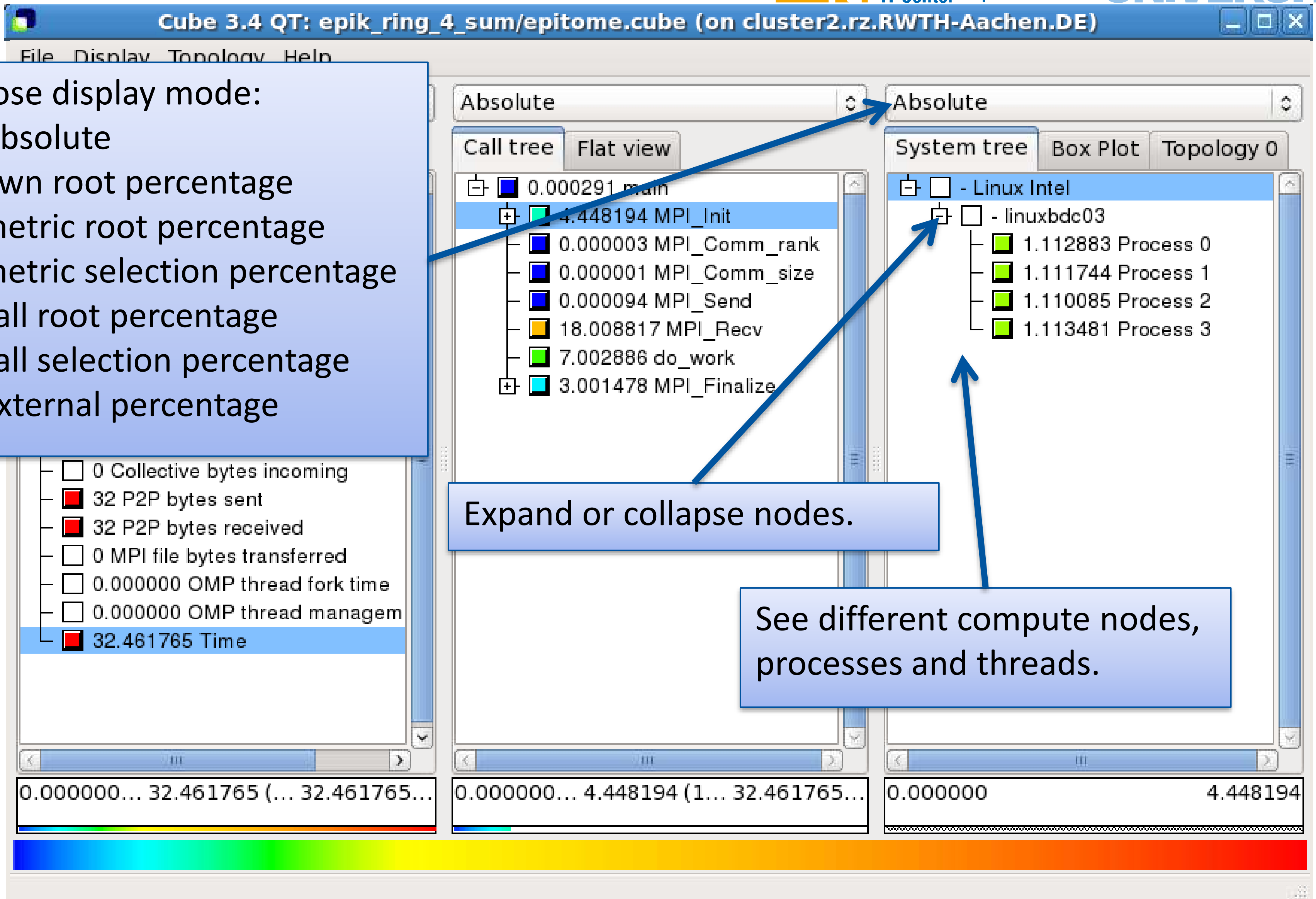
Expand or collapse functions.

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

Choose a function to display.



Scalasca – System View



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.

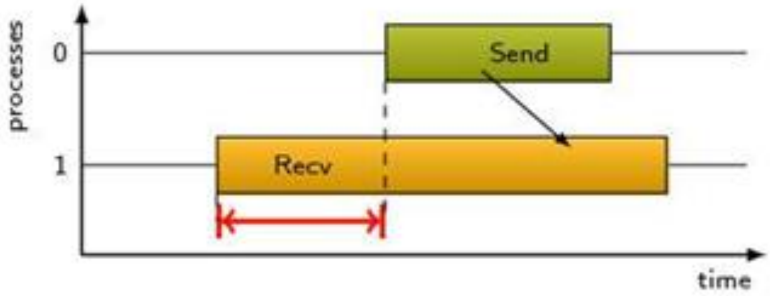
See different compute nodes, processes and threads.

Automatic Pattern Detection with Scalasca

- For large scale applications visualizing traces might be to 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.

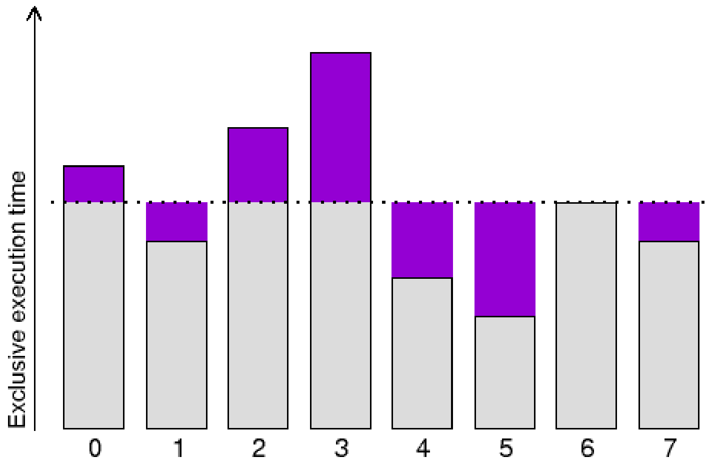
Example: The late-sender pattern



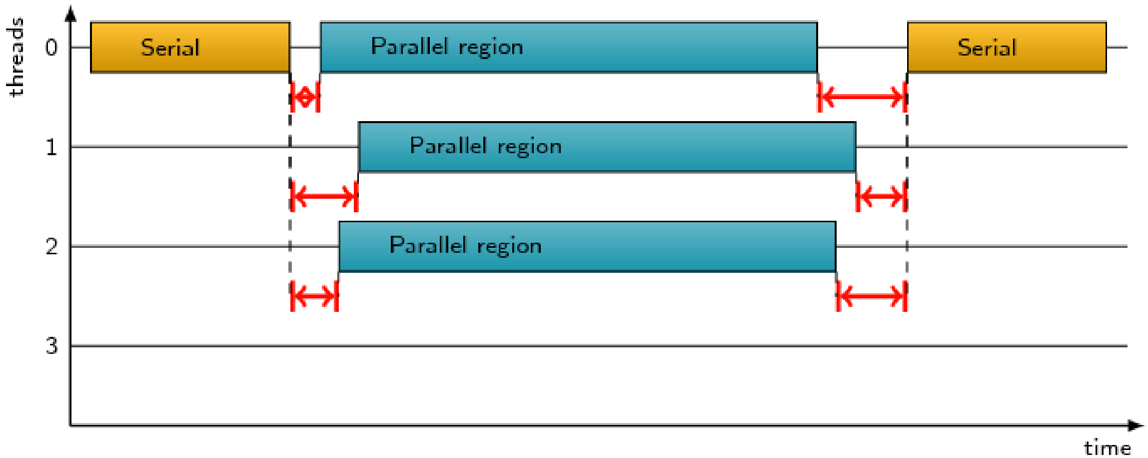
- The time lost through this problem (red arrow) is accumulated over the complete run and stored in a profile.

- **Many patterns can be detected, like:**

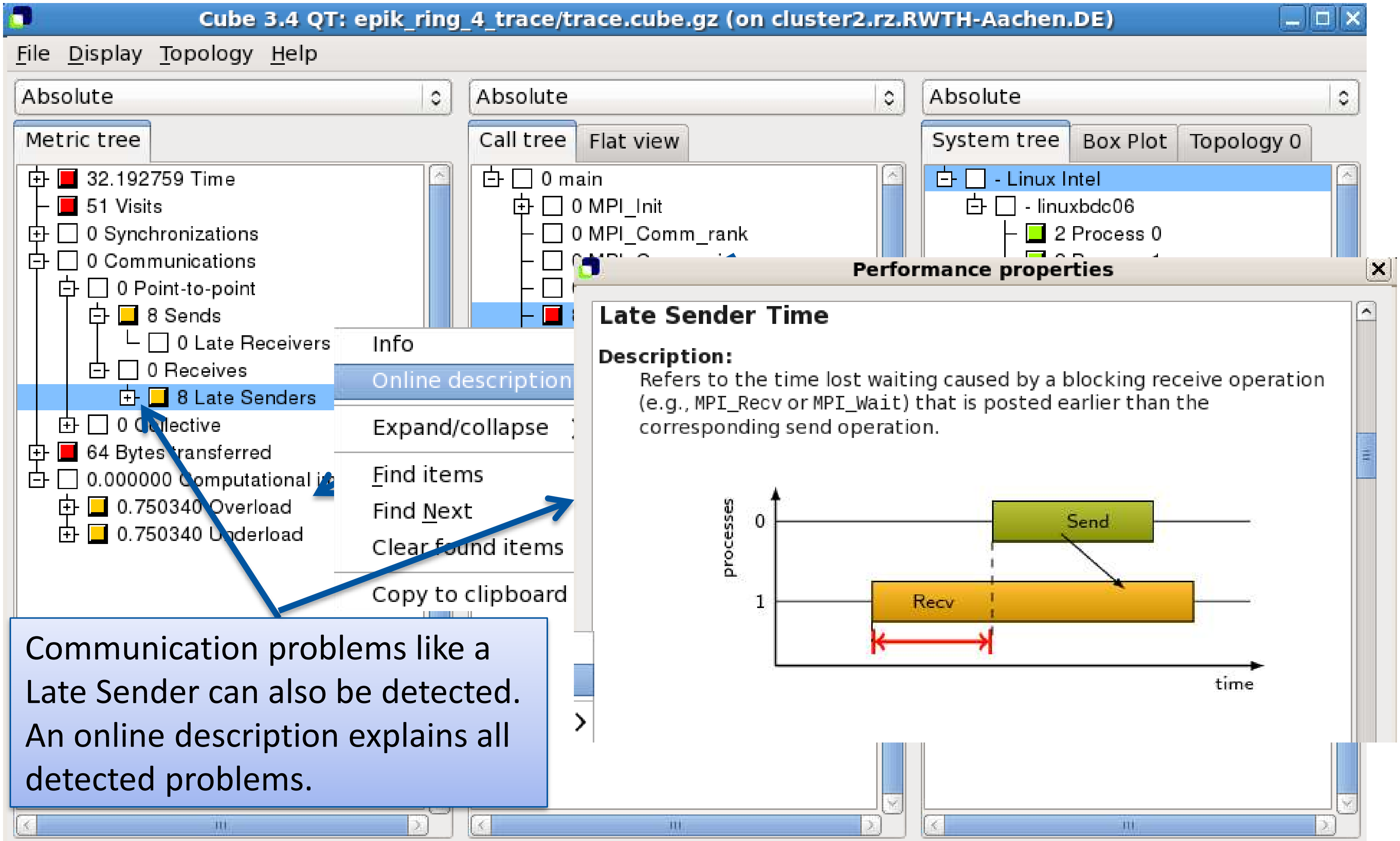
Computational load imbalance:



OpenMP management time:



Scalasca - Traces



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