

Scalable identifier unification in SCALASCA using MRNet

2008-04-30 | Markus Geimer
Jülich Supercomputing Centre

m.geimer@fz-juelich.de

Overview

Introduction to SCALASCA

Identifier unification

- Motivating example
- Details

Scalable unification using MRNet

Summary

The SCALASCA project

Overview

- Started in January 2006
- Funded by Helmholtz Initiative & Networking Fund
- Developed in collaboration with ICL/UT
- Follow-up to pioneering KOJAK project
 - *Automatic pattern-based performance analysis*

Objective

- Development of a **scalable** performance analysis toolset
- Specifically targeting **large-scale** applications

Features

Open source

Portable

- BG/L, BG/P, IBM SP & blade clusters, Cray XT3/4, SGI Altix, Sun Fire clusters (SPARC, x86-64), ...

Supports various languages & parallel programming paradigms

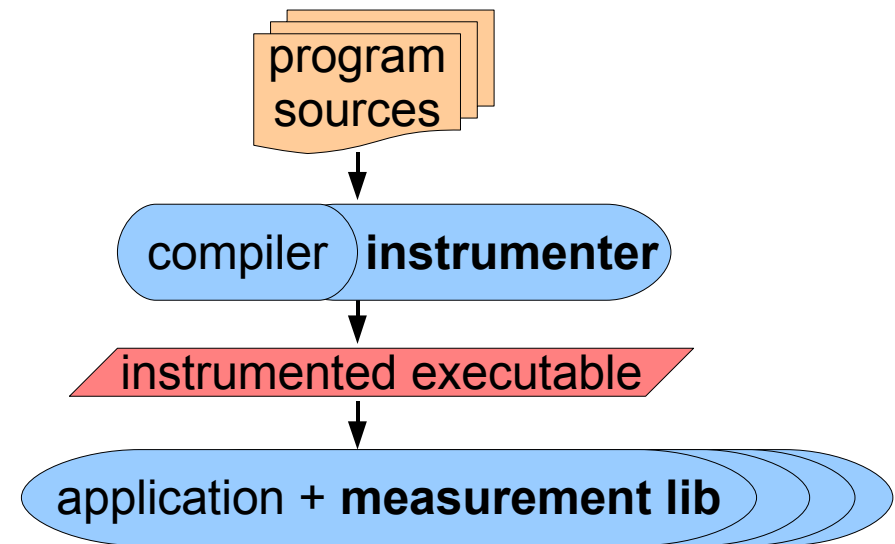
- Fortran, C, C++
- MPI, OpenMP & hybrid MPI/OpenMP

Integrated measurement & analysis toolset

- Runtime summarization (aka profiling)
- Automatic event trace analysis

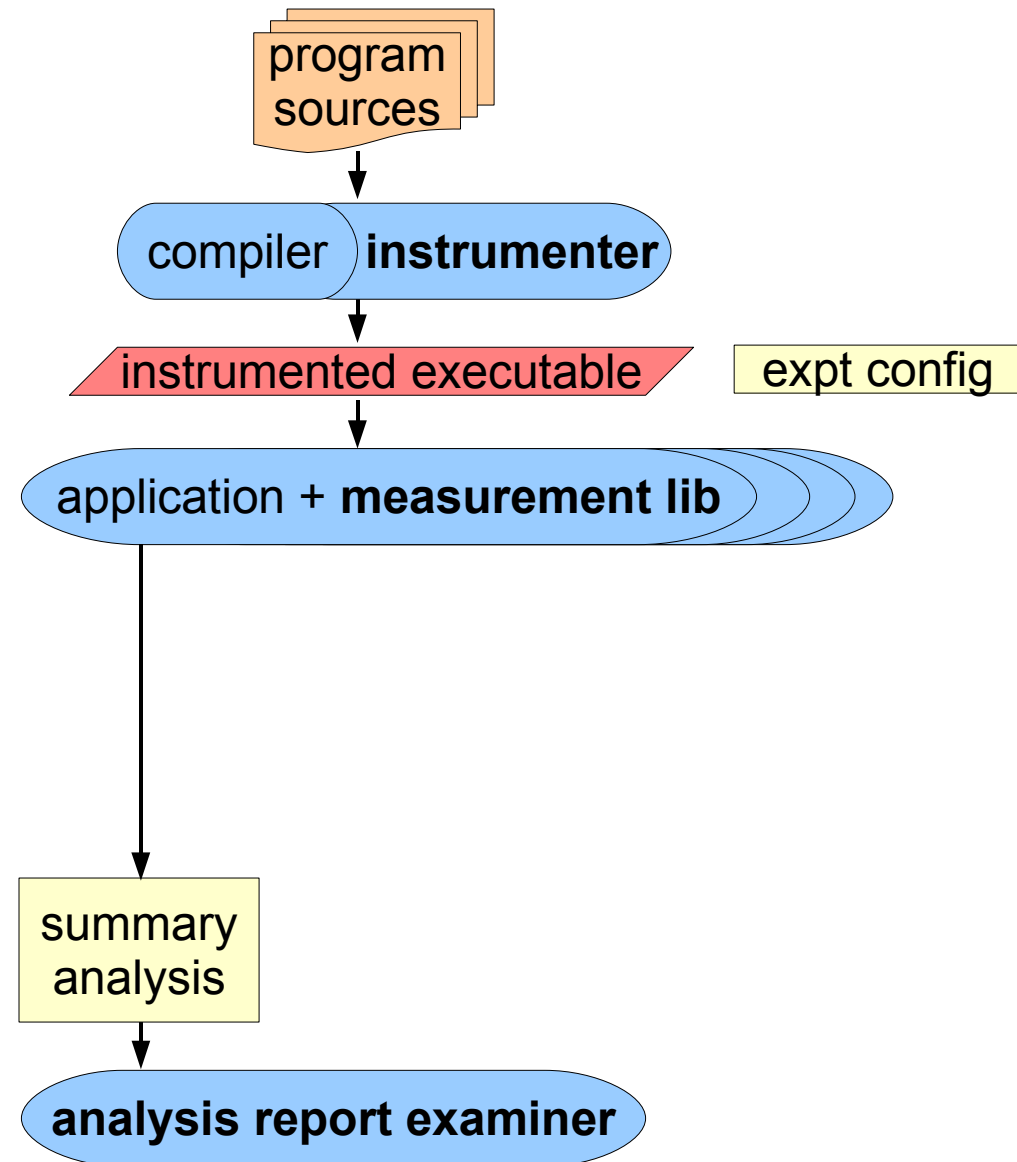
Application instrumentation

- Automatic/manual code instrumenter
 - *Processes program sources*
 - *Adds instrumentation and measurement library into application executable*
- Measurement library
 - *Exploits MPI standard profiling interface (PMPI)*
 - *Provides measurement infrastructure & instrumentation API*



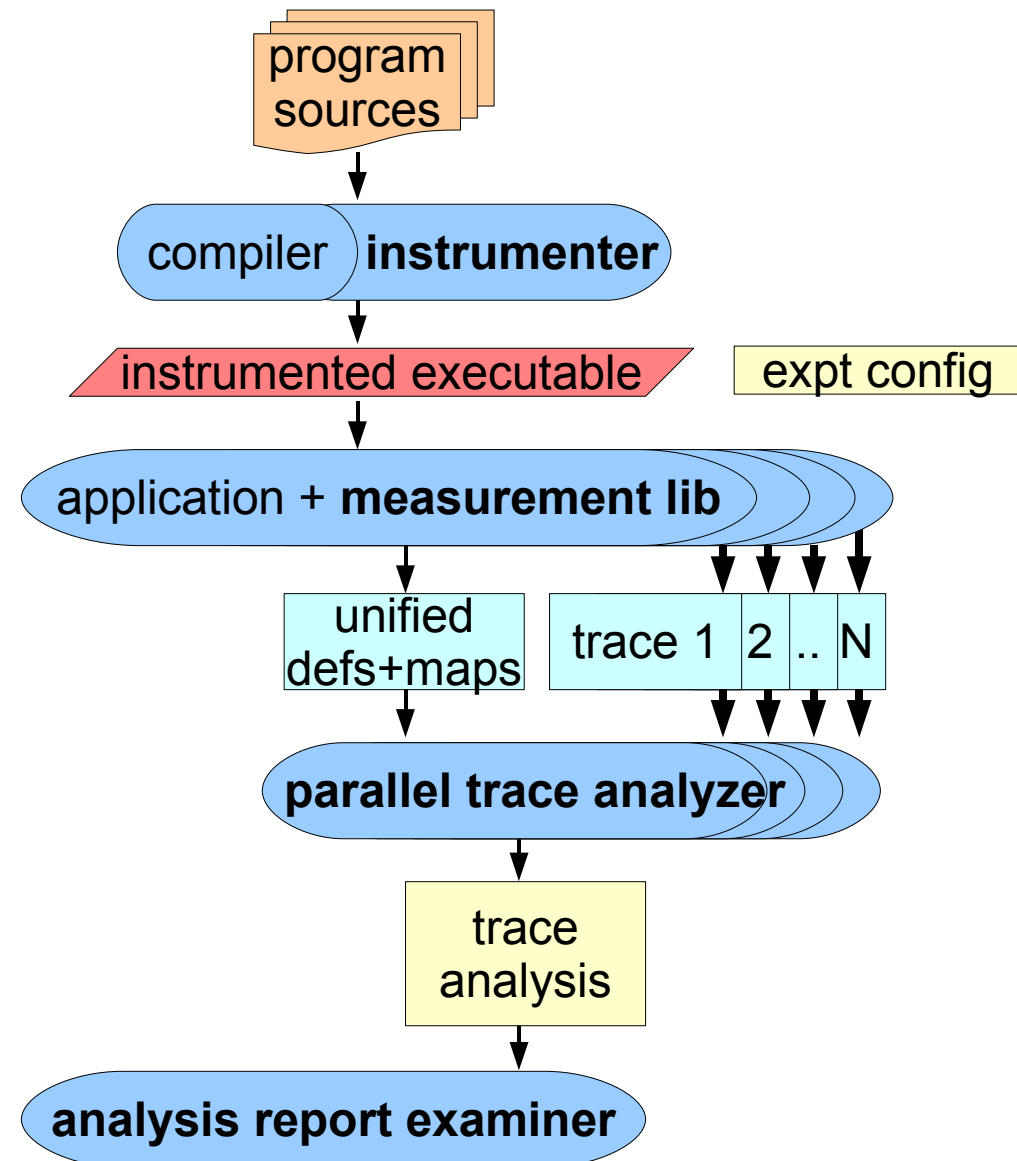
Runtime summarization

- Measurement library manages threads & events produced by instrumentation
- Measurements summarized by thread & call-path during execution
- Analysis report unified & collated at finalization
- Presentation of analysis results



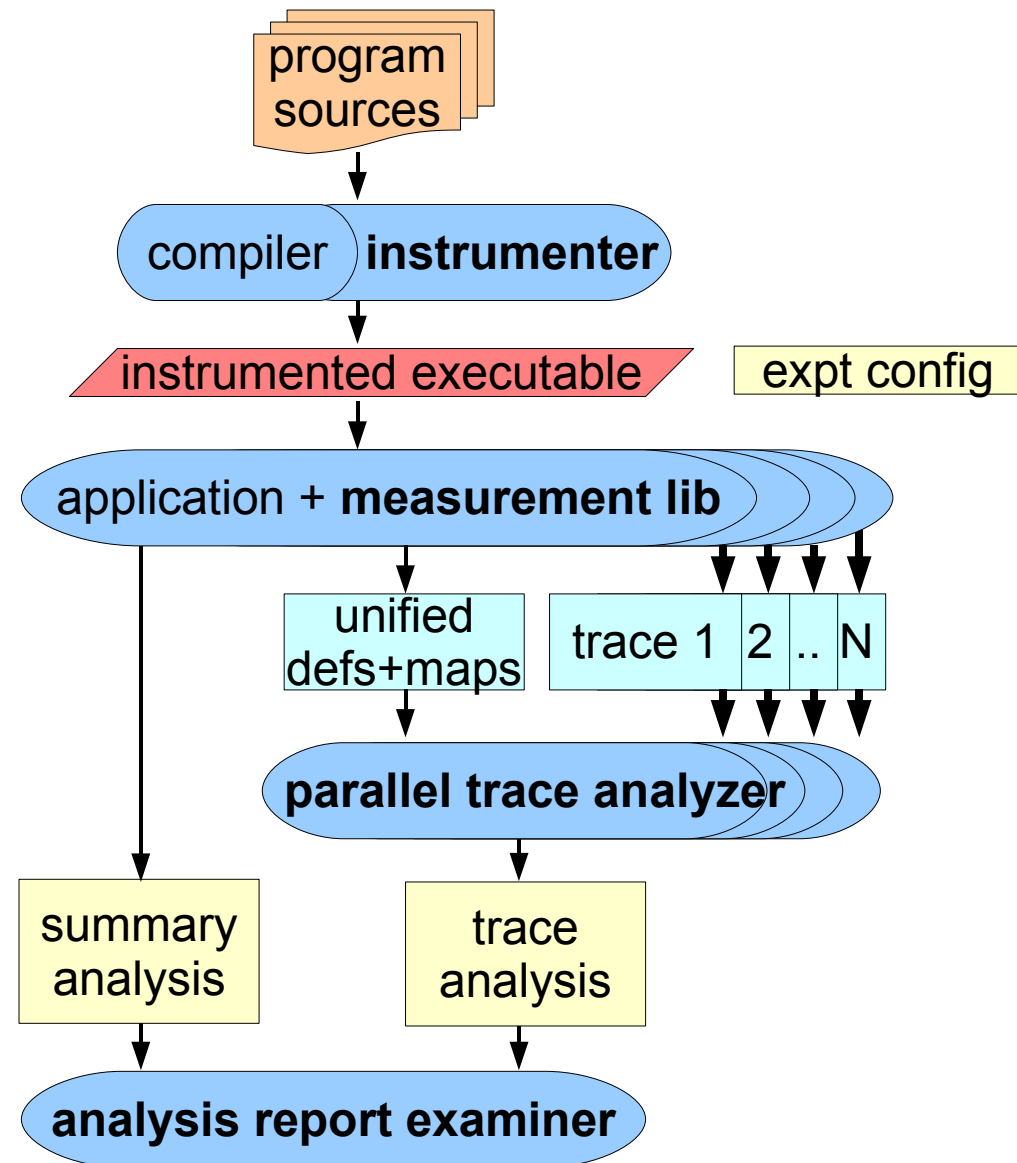
Event tracing & analysis

- Time-stamped events buffered during measurement for each thread
- Flushed to files along with unified definitions & mapping tables at finalization
- Follow-up analysis replays events and produces extended analysis report
- Presentation of analysis report

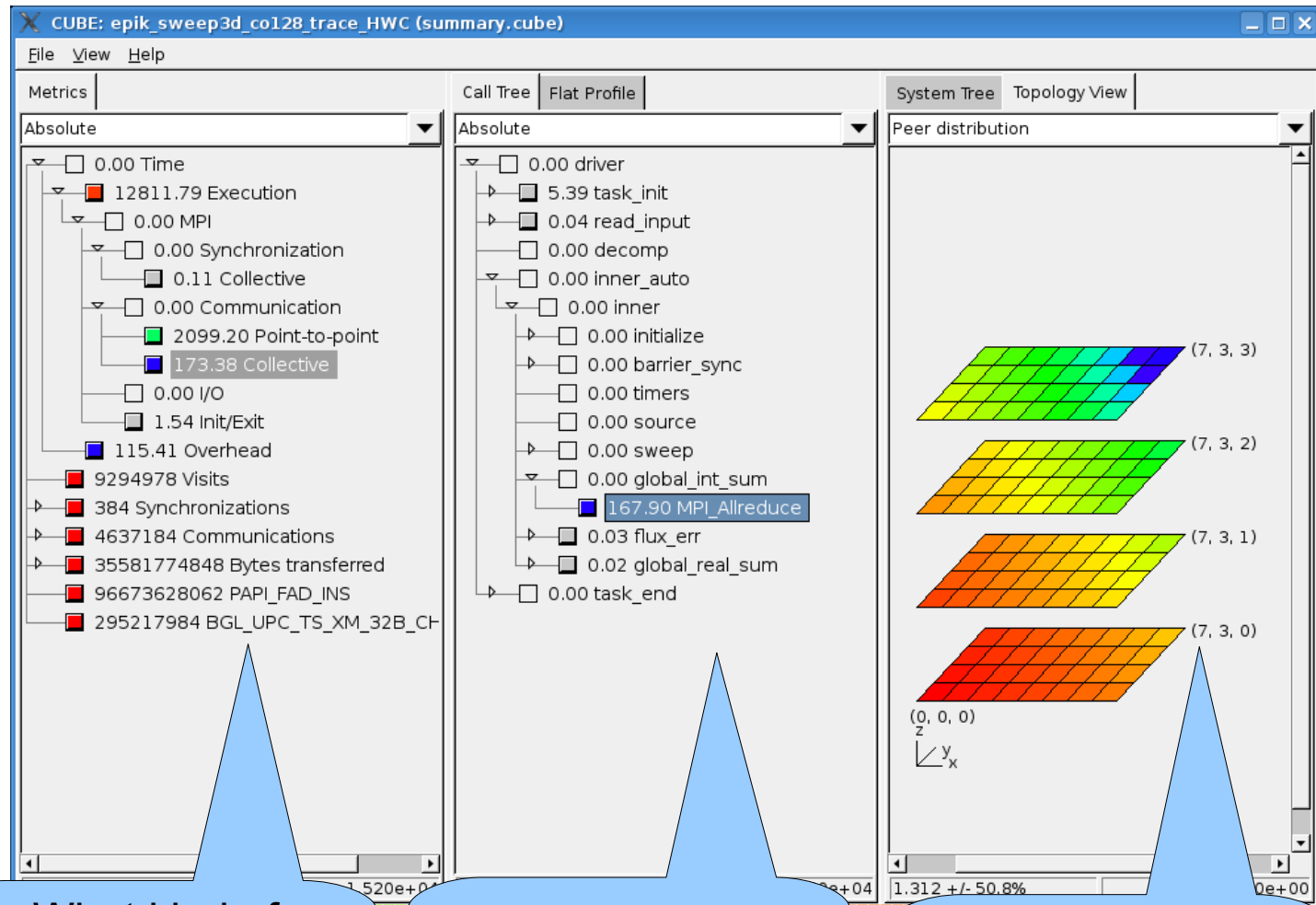


SCALASCA components

- Automatic/manual code instrumenter
- Unified measurement library supporting both
 - *runtime summaries*
 - *trace file generation*
- Parallel, replay-based trace analyzer
- Common analysis report examiner



Runtime summary report

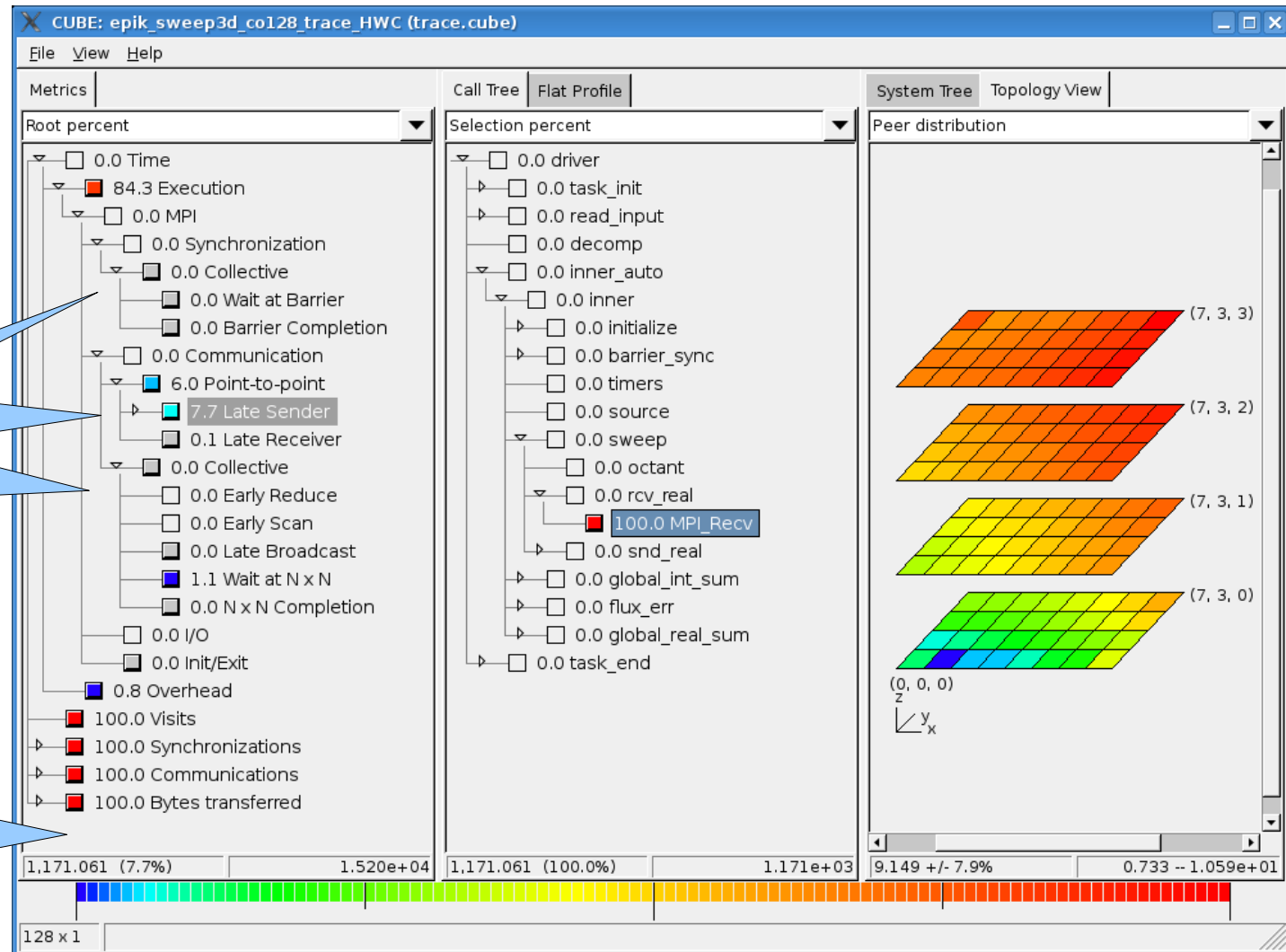


What kind of performance problem?

Where is it in the source code? In what context?

How is it distributed across the system?

Trace analysis report



Scalability test: SMG2000

ASC Purple application benchmark kernel

- Semi-coarsening multigrid solver

MPI parallel version

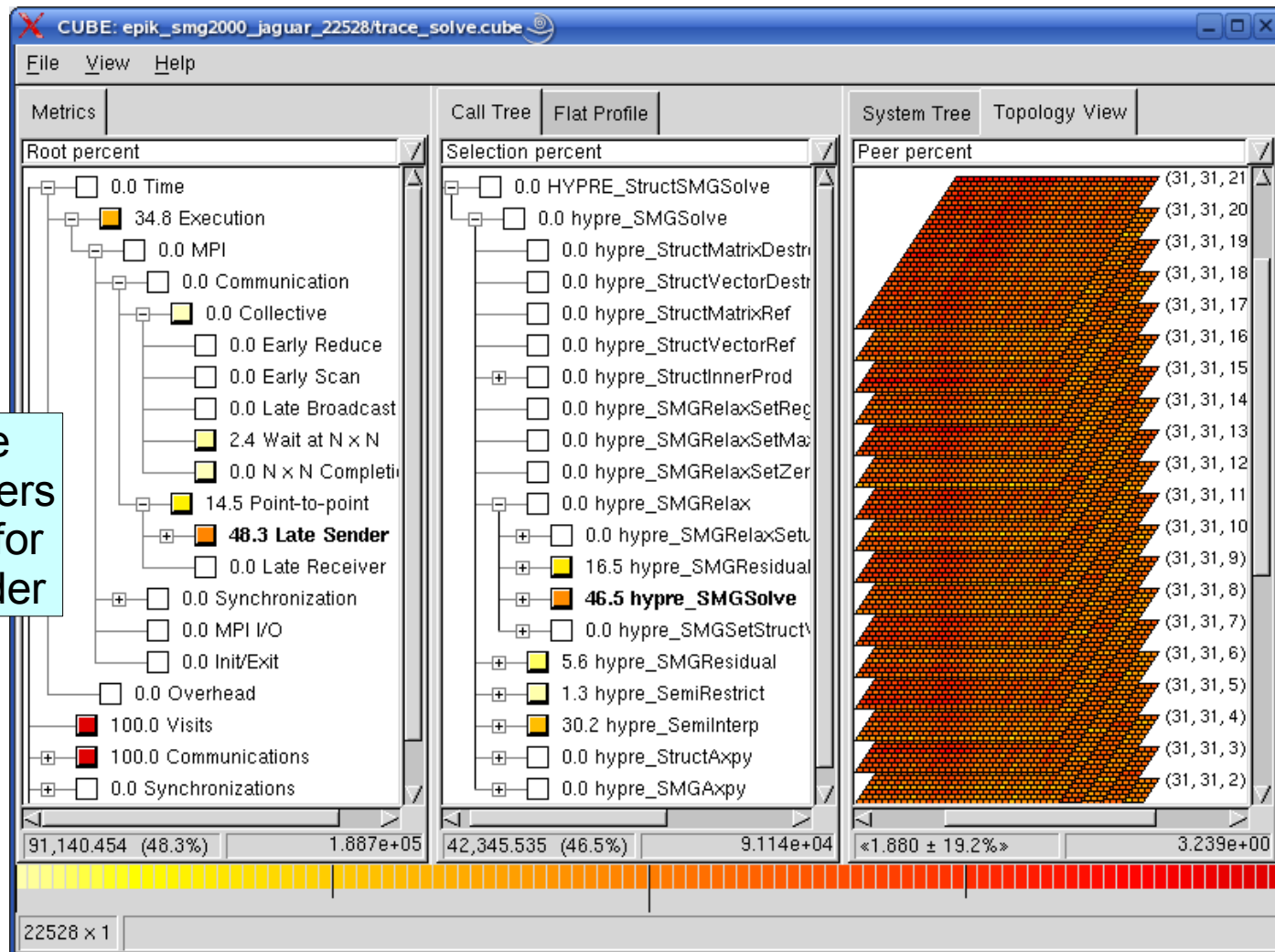
- 28,000 lines of C code in 50 files
- Performs lots of non-local communication

Test configuration

- Weak-scaling with 64x64x32 datapoints/process
- Replaced convergence test by fixed number of iterations (5)
- Full instrumentation / no filtering
- Run on Cray XT3/4 @ ORNL (jaguar) with up to 22k CPUs

Trace analysis result @ 22k CPUs

Wait time of receivers blocked for late sender

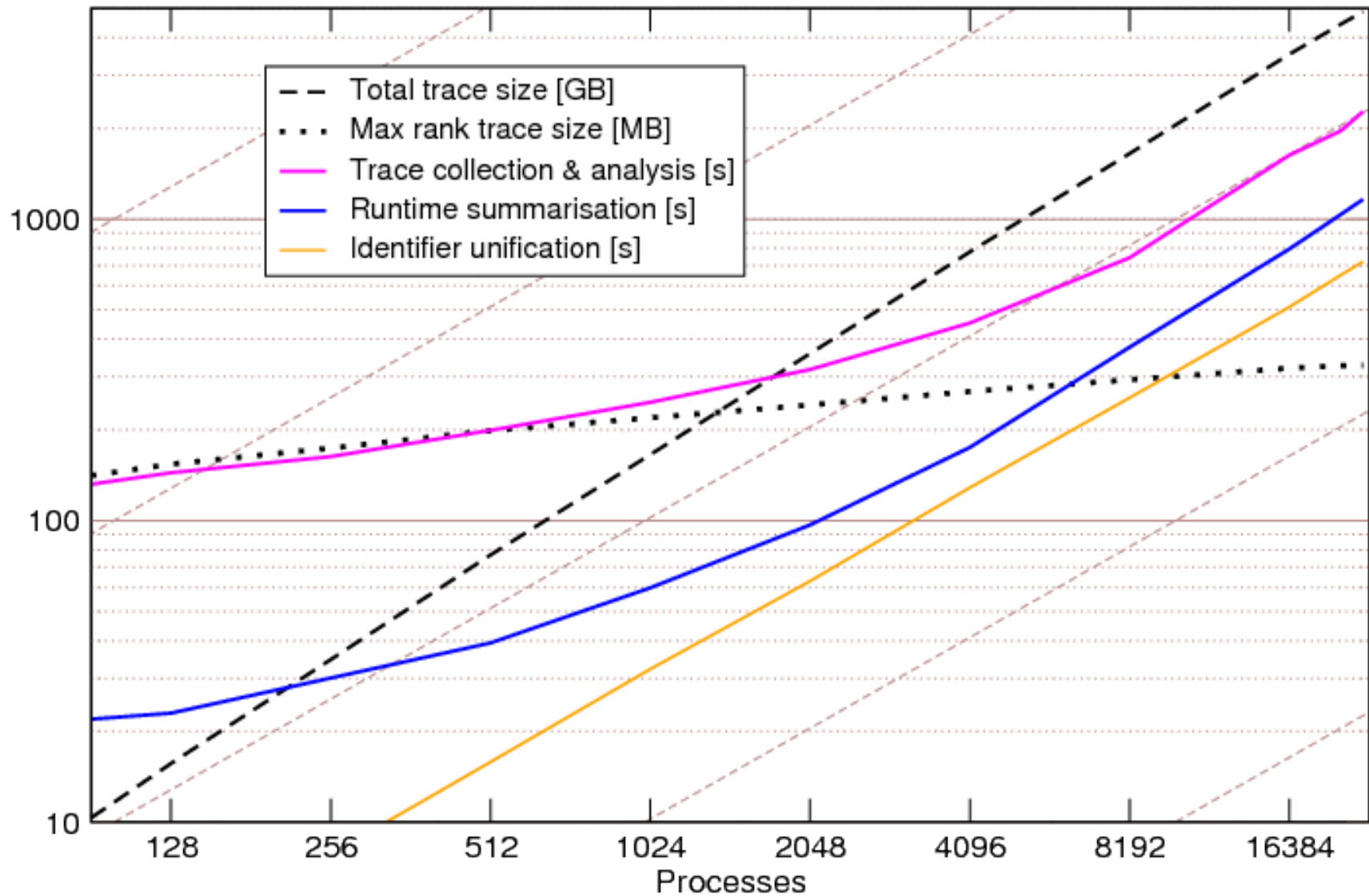


Analysis statistics @ 22k CPUs

	Summary ¹⁾	Trace
Measurement		
<i>Experiment activation</i>	50 s	224 s
<i>Identifier unification & mapping table gen.</i>	720 s	836 s
<i>Collation / Flushing</i>	392 s	272 s
<i>Number of events</i>		~426 G
<i>Buffered trace data</i>		4.9 TB
Analysis		
<i>Total trace analysis time</i>		937 s
<i>Time for replay only</i>		114 s
<i>Report size</i>	1.4 GB	1.5 GB

¹⁾ Includes 4 Opteron hardware counter metrics

Measurement & analysis scalability



Identifier unification

Definition records

- Created for various entities, e.g.,
 - *MPI communicators*
 - *Source code regions*
 - *Call-paths*
- Each definition record gets process-local numeric identifier

Identifier unification

- Needed to establish “global view”
- Eliminates duplicate definition records
- Determines local→global identifier mapping

Example

```

int main()
{
    ...
    if (rank != 0)
        foo();
    bar();
    ...
}

```

Local definitions
rank == 0

1	main()
2	r: 1 p:--
3	bar()
4	r: 3 p: 2

Local definitions
rank != 0

1	main()
2	r: 1 p:--
3	foo()
4	r: 3 p: 2
5	bar()
6	r: 5 p: 2

Global definitions



1	main()
2	r: 1 p:--
3	bar()
4	r: 3 p: 2
5	foo()
6	r: 5 p: 2

Mapping

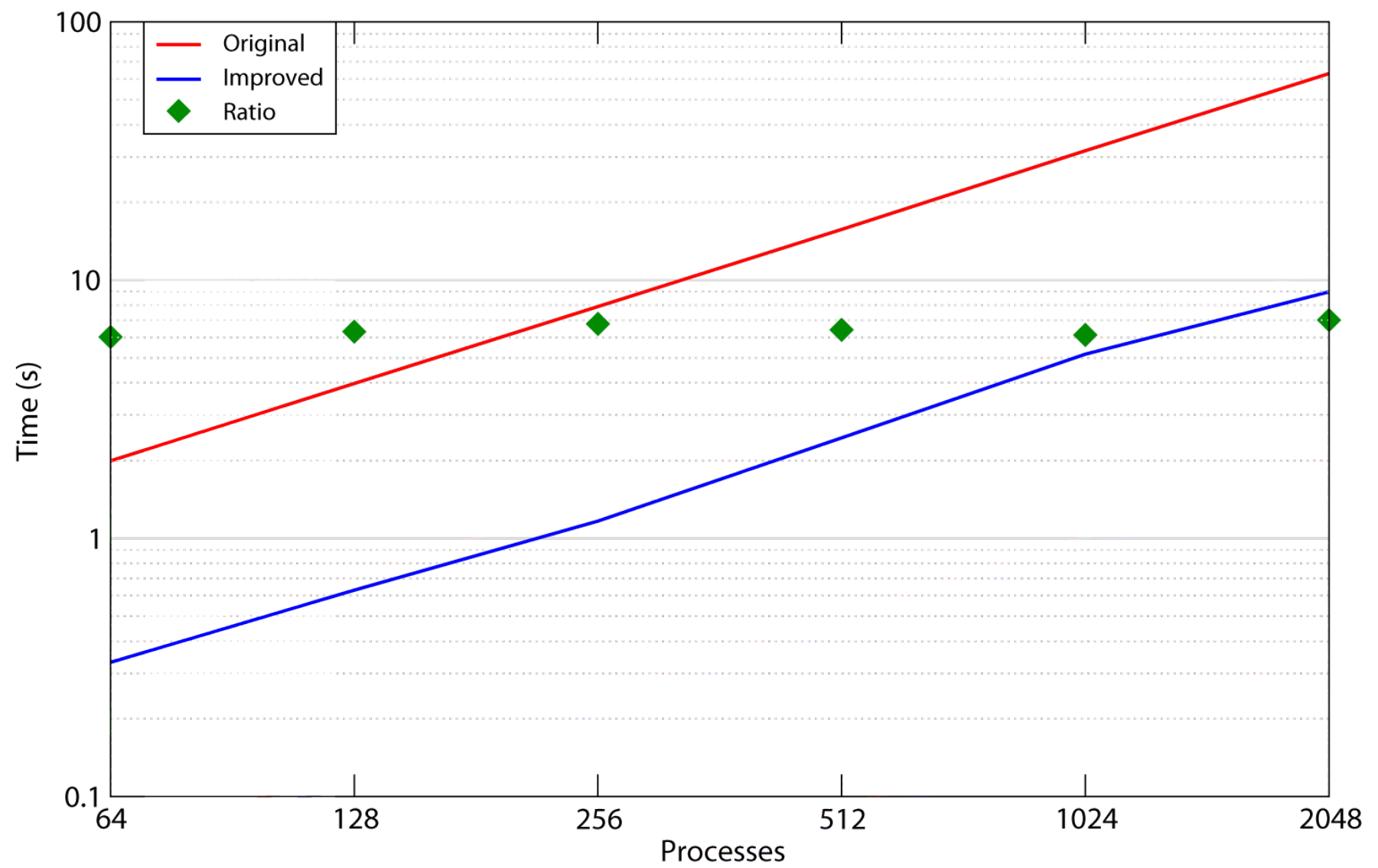
1 → 1
2 → 2
3 → 3
4 → 4

Mapping

1 → 1
2 → 2
3 → 5
4 → 6
5 → 3
6 → 4

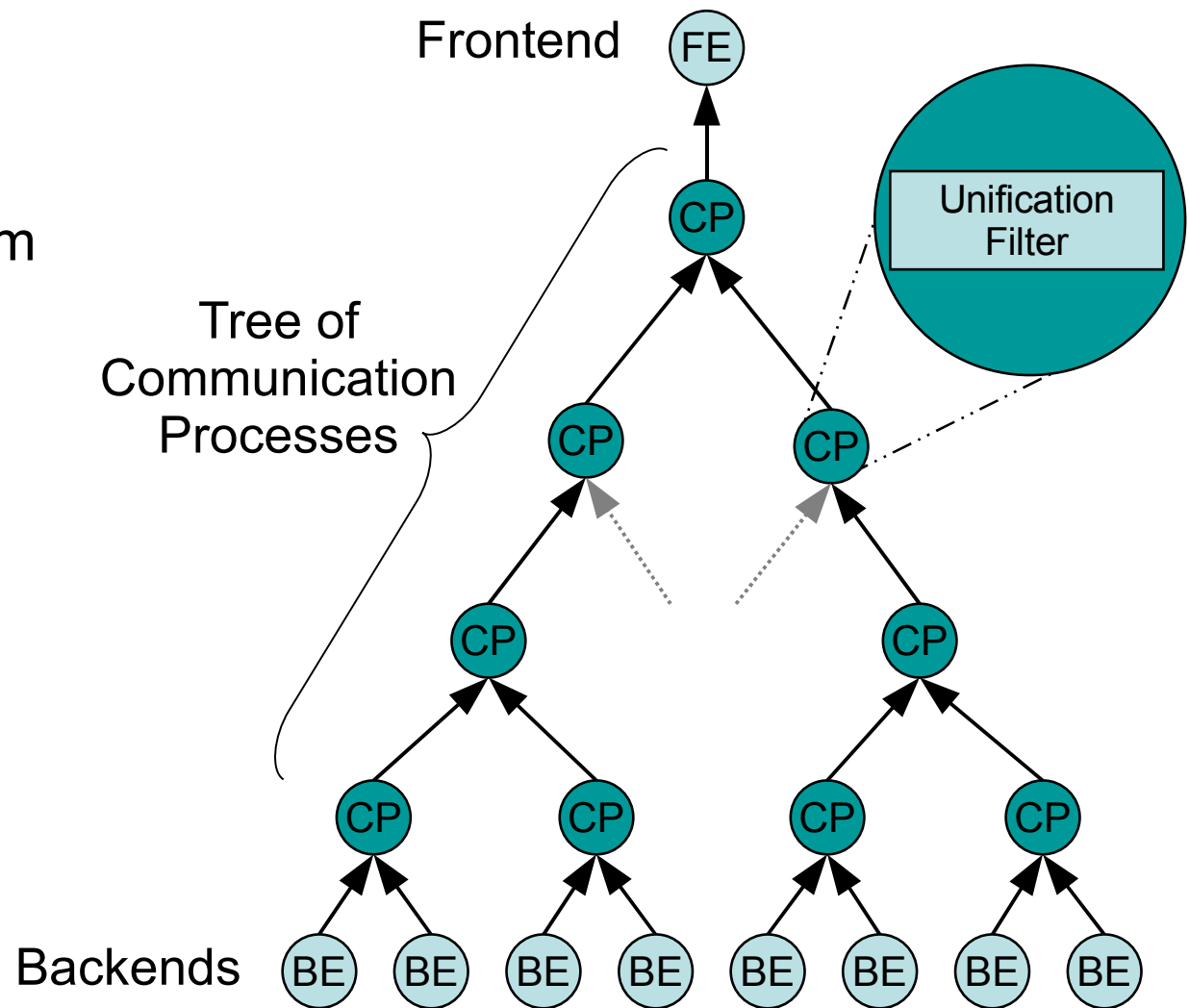
-  Region definition
-  Call-path definition

Performance after sequential optimizations (Cray XT)



Idea: Study tree-based unification using MRNet

- Provides flexible infrastructure
- Reduces complexity from $O(n)$ to $O(\log n)$
- Implement unification as filter
- Frontend writes unified global definitions & mapping tables to files



Unification filter

Input

- Vector of definition buffers from a set of processes / child nodes in the tree

Output

- Single definition buffer containing (partially) unified data

Problem

- Assigned identifiers can be different on next tree level
- Mappings generated on communication processes are not final

Mapping table generation

Algorithm outline

- Send global definition data back to application processes
- Perform unification of global definitions and process' own local definitions
- Send generated mappings to frontend

Challenge

- Aggregated mapping table data can grow large
- Collation via MRNet not straightforward
- Various alternatives to be evaluated

Results

Initial prototype

- Running since Monday 😊
- Unification of most definition records already works
- Some need closer investigation, though

Mapping file generation

- Not implemented yet
- Needs multiple passes up/down the tree
- Efficiency needs to be investigated (break-even)

Stay tuned...

Summary

SCALASCA

- Portable open-source toolset for performance analysis
- Supports runtime summarization & event trace analysis
- Has proven its scalability & usability with up to 22k CPUs

Identifier unification

- Became increasingly important at large scales
- Sequential optimizations helped somewhat

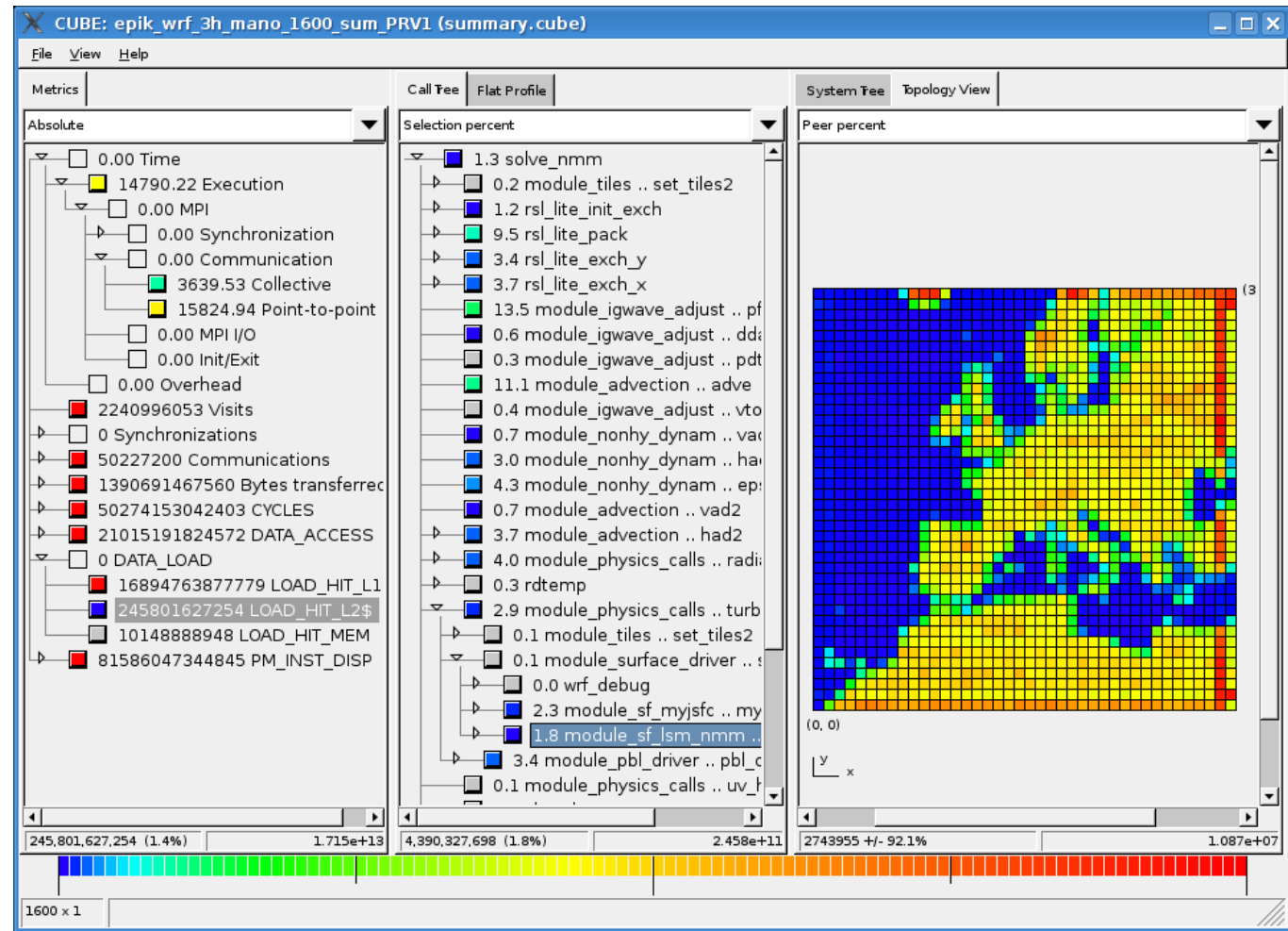
Tree-based unification using MRNet

- Initial prototype working
- Promising, but still some open issues

Thank you!

For more information
visit our website:

<http://www.scalasca.org>



WRF-NMM weather prediction code on MareNostrum @ 1600 CPUs