Lawrence Livermore
National Laboratory

JÜLICH
FORSCHUNGSZENTRUM

Supporting Performance Analysis and Optimization on Extreme-Scale Computer Systems – Current State & Future Directions

SC 2012 |

Martin Schulz – Lawrence Livermore National Laboratory
Bernd Mohr – Jülich Supercomputing Centre
Brian Wylie – Jülich Supercomputing Centre

Presenters

- Martin Schulz
  - Lawrence Livermore National Laboratory
- Bernd Mohr
  - Jülich Supercomputing Centre
- Brian Wylie
  - Jülich Supercomputing Centre

- Expertise
  - Performance tool development
    - KOJAK, Open|SpeedShop, P^{n}MPI, Scalasca, …
  - Scaling analysis techniques
  - Application optimization

Why this Tutorial? General Tool Observations

- Tool developers get access to new machines / architectures at same time as application developers
  - **Tools not available when most needed**
- **Shrinking development + install cycles of machines/architectures**
  - Faster than applications / users / tool developers can cope with
- Developing working and robust scalable tools **needs**
  - **Access** to large machines
  - Large enough **allocation** to make large test runs
- Users still think tools need to be
  - Always simple (even on large and extremely complex systems)
  - Always fast (even on large and extremely complex systems)

Goals of the Tutorial

- Overview of code development tools
- Scaling techniques for current machines
  - Profiling/Sampling
  - Tracing
  - Examples of existing tool kits
- Challenges going forward towards Exascale
  - What will have to change?
  - Where are the bottlenecks?
- Impact on tools and on tool users
  - Changes that can be expected moving forward

Let's keep this interactive

  - Ask questions as we go along

Outline

- Analysis techniques on current generation machines
  - Profiling/Sampling techniques
  - Tracing and trace analysis techniques
- New techniques/concepts introduced for Petascale
  - Hierarchical aggregation
  - Component frameworks
- The road to Exascale
  - Current developments
  - Impact on code development environments

Reference and Extra Material

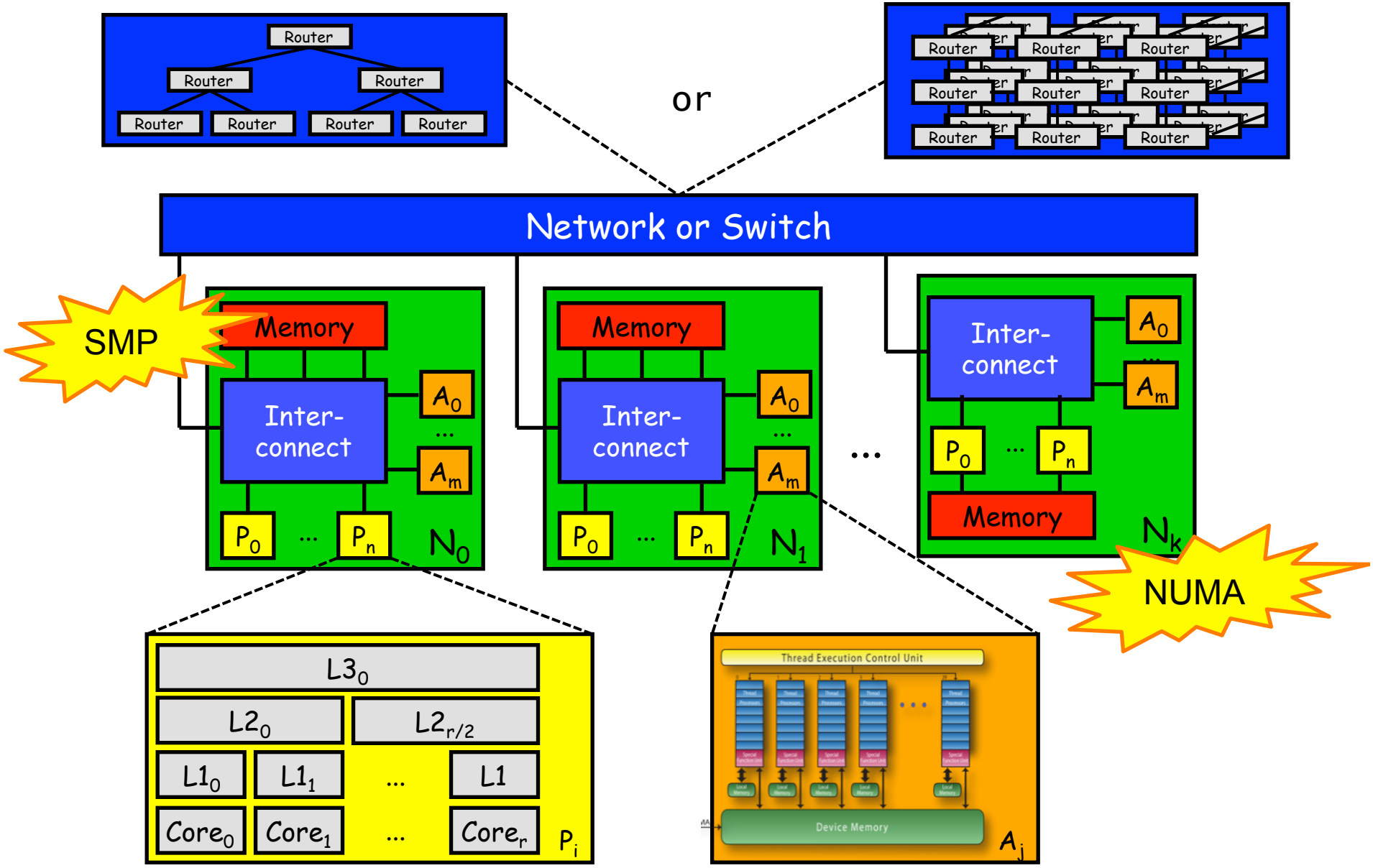
- The handout slides contain additional slides with
  - Reference material
  - More detailed and advanced material

- This slides are marked with the symbol 

- Topics which are described in more detail in the reference material are marked with 

MOTIVATION

Parallel Architectures: State of the Art



Increasing Importance of Scaling

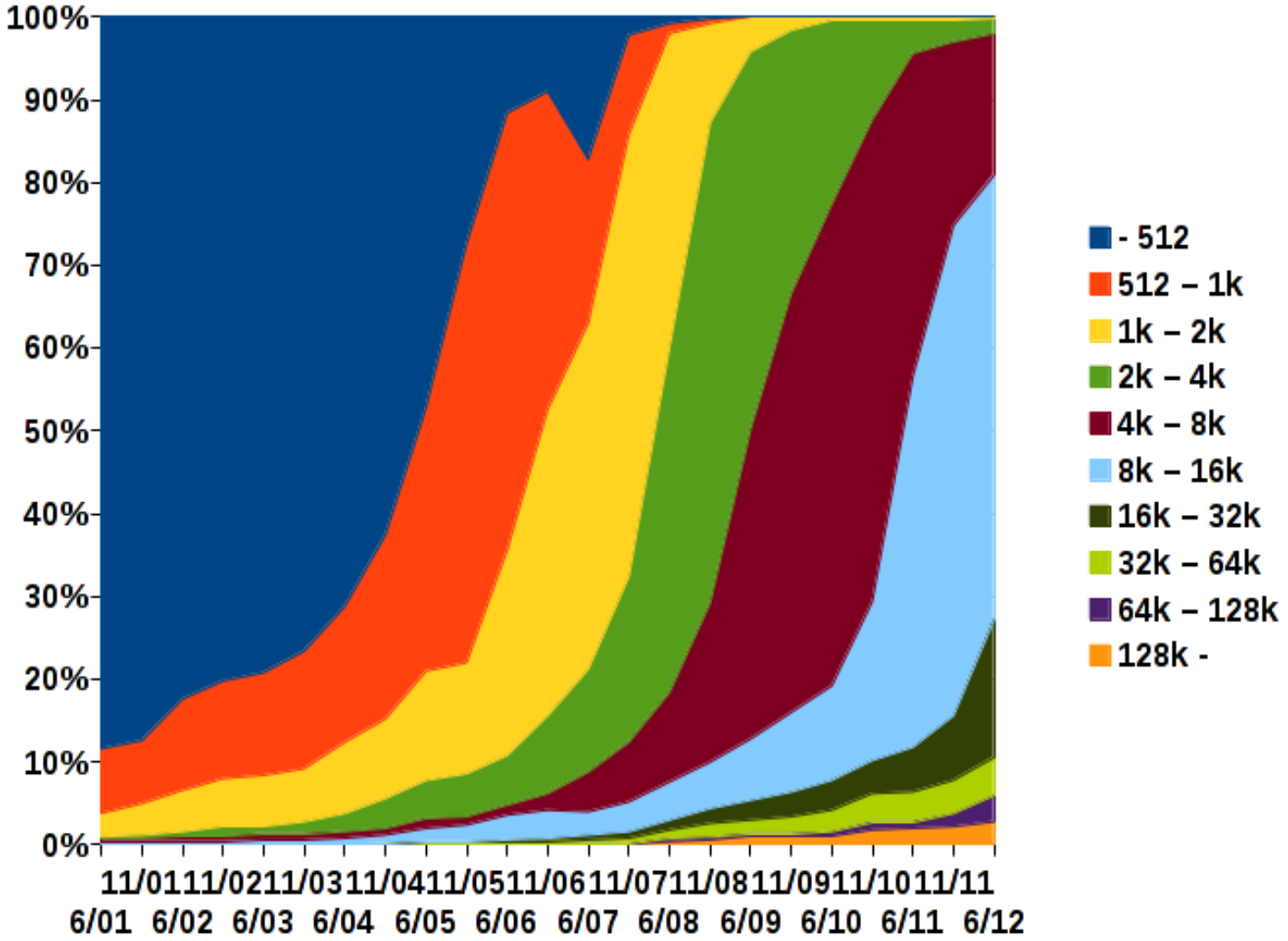
- Number of Cores share for TOP 500 June 2012

NCore	Count	Share	∑Rmax	Share	∑NCore
1025-2048	1	0.2%	122 TF	0.1%	1,280
2049-4096	9	1.8%	623 TF	0.5%	30,520
4097-8192	85	17.0%	7,500 TF	6.1%	581,728
8193-16384	268	53.6%	24,852 TF	20.1%	3,319,798
> 16384	137	27.4%	90,376 TF	73.2%	9,519,131
Total	500	100%	123,473 TF	100%	13,452,457

- **Average** system size: **26,904 cores**
- **Median** system size: **13,104 cores**

Increasing Importance of Scaling II

- Number of Cores share for TOP 500 Jun 2001 – Jun 2012



Personal Motivation JSC

- Supercomputers at Jülich Supercomputing Centre

Year	Machine	#Cores
1998	Cray T3E	1,024
2003	IBM p690 cluster	1,312
2006	IBM BlueGene/L	16,386
2007	IBM BlueGene/P	65,536
2009	Bull/SUN/Intel	26,304
2009	IBM BlueGene/P	294,912
2012	IBM BlueGene/Q	131,072 + ###,###

Personal Motivation JSC II

Jülich Supercomputing Centre Machine Hall Early 2012

72-rack BlueGene/P
(292,912 cores)

3288-node Bull/SUN
(26,304 cores)

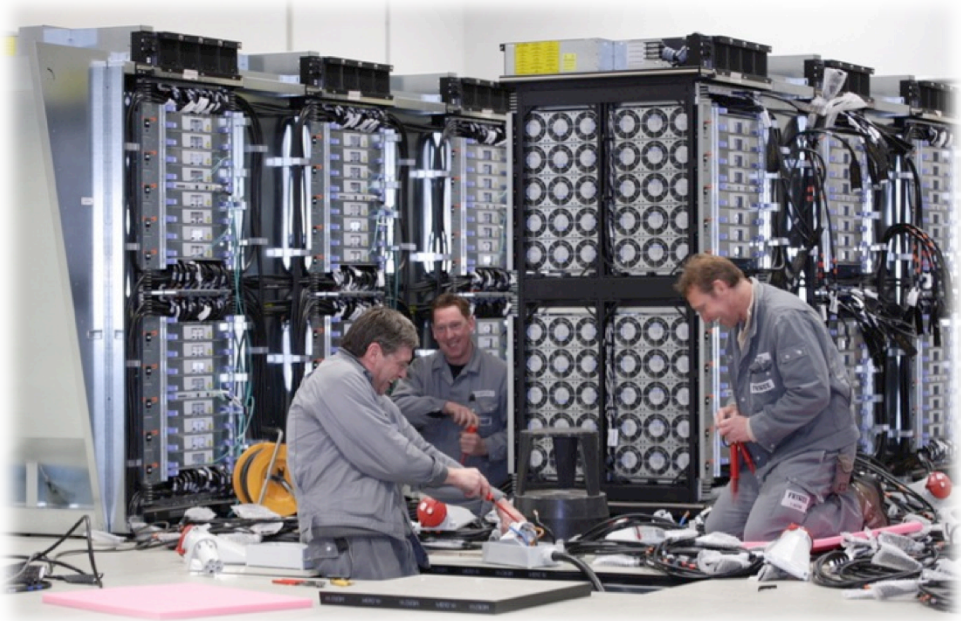
Personal Motivation JSC III

Jülich Supercomputing Centre: 294,912 core BlueGene/P

Most parallel machine of the world for 2009 – 06/2011!

JSC IBM BlueGene/P

- 32-bit PowerPC 450
  850 MHz, 4-way SMP
- 1,00 Petaflop/s peak
  0,82 Petaflop/s Linpack
  Jun09: #3
  Jun11: #12



- 144 TByte memory
- Numerous Hardware
  - 72 racks, 73,728 nodes, 294,912 cores,
  - 648 power modules, 576 link cards, 144 service cards,
  - 4,352 data cables, 288 service cables, …
- Interconnects
  - 3D-Torus, collective (tree), and barrier network
  - 10 GigaBit (I/O), 1 GigaBit (control)

JSC: Supercomputer Networks

Length of the communication cables:

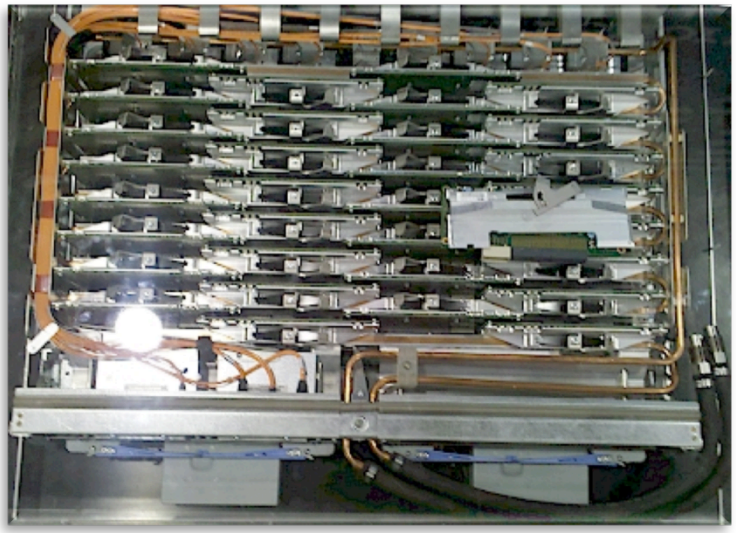
• BG/P:	23 km (copper)	
	21 km (fiber)	
• JUROPA:	20 km (mixed)	
• HPC-FF:	16 km (mixed)	
	80 km	

First 8 IBM BG/Q Racks at JSC

- Will be extended to 28 racks (458,752 cores) end of 2012

Personal Motivation, LLNL: BG/Q or Sequoia

- **Blue Gene/Q:**
  - 20PF/s peak
  - 96 racks, 98,304 nodes
  - 1.5M cores/6M threads
  - 1.5 PB memory
  - Liquid cooled
  - 5D torus interconnect
  - New technologies like HW-TM



Comparison BG/P ⇔ BG/Q

	BG/P	BG/Q
Rack	1024 nodes	1024 nodes
Processor	PPC 450, 850 MHz 4 cores per node	PPC A2, 1.6 GHz **16 cores per node** HW Support for Thread Level Speculation and Transactional Memory
Core	Double vector unit	Quad vector unit **4-way SMT**
Memory	2 or 4 GByte per node	16 GByte per node
Network	• 3D Torus • Collective network • Barrier/Interrupt • I/O network • Control Network	• 5D Torus • Collective network (part of the 5D Torus) • Barrier/Interrupt (part of 5D Torus) • I/O network • Control Network
Cooling	Air	Liquid

Livermore Computing

- **Long supercomputing tradition**
  - Four machine rooms on site
  - Simulation in support of LLNL missions
  - Capacity and capability computing
  - Tradition of Co-Design with vendors
- **BlueGene/L (~600TF) – 212,992 cores**
  - #1 machine from 2005-2007
  - #8 on the Top500 in June 2011
  - Last part decommissioned this year
- **Some other current machines**
  - **Zin** Sandy Bridge/IB ~1 PF
  - **Dawn** BlueGene/P ~500 TF
  - **Cab** Sandy Bridge/IB ~425 TF
  - **Sierra** Nehalem / IB ~261 TF
  - **Juno** Opteron / IB ~160 TF
  - **Hera** Opteron / IB ~120 TF
  - **Graph** Opteron / IB / GPU ~110 TF
  - **Hyperion** Nehalem / IB ~90 TF

Personal Motivation, LLNL: Capacity Systems

- Series of clusters
  - Commodity components
  - QDR Infiniband
  - Standardized Tri-lab software env.
  - As of last year:
    - Up to 1944 nodes / 23328 cores
    - Up to 261 Tflop/s peak
- Bought as sets of scalable units (SU)
  - 144/162 nodes each
  - Integrated IB
  - Procurement across the Tri-Labs Tri-Lab Capacity Systems (TLCC)
  - Combine several SUs into a system
- Applications
  - Small-medium jobs
  - Strictly batch scheduled MOAB/SLURM
  - Grand challenge & dedicated runs
- No longer just capacity systems (!)

Next Generation Clusters / TLCC-2

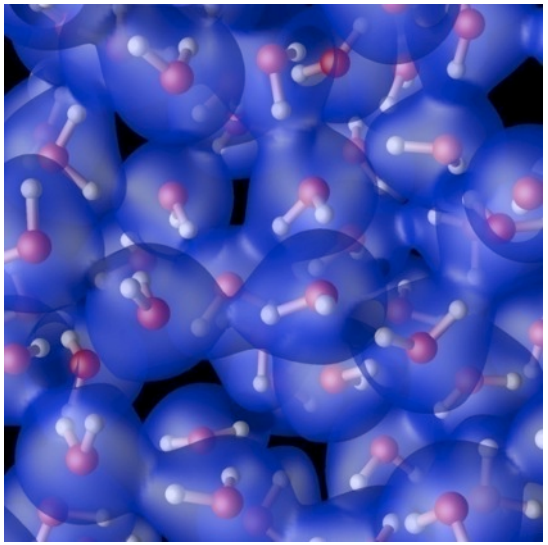
Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
15	Lawrence Livermore National Laboratory United States	Xtreme-X GreenBlade GB512X, Xeon E5 (Sandy Bridge - EP) 8C 2.60GHz, Infiniband QDR / 2011 Appro	46208	773.70	961.13	924.2

- TLCC-2 Compute Clusters
  - Again bought in Scalable Units
  - Zin/Merl/Cab
  - Largest system is almost 1 Pflop/s
- Built around Intel's Sandy Bridge
  - Dual socket / 8 cores each
  - New measurement options
  - Opportunistic Turbo Mode
    - New impacts on performance
    - Power limitations

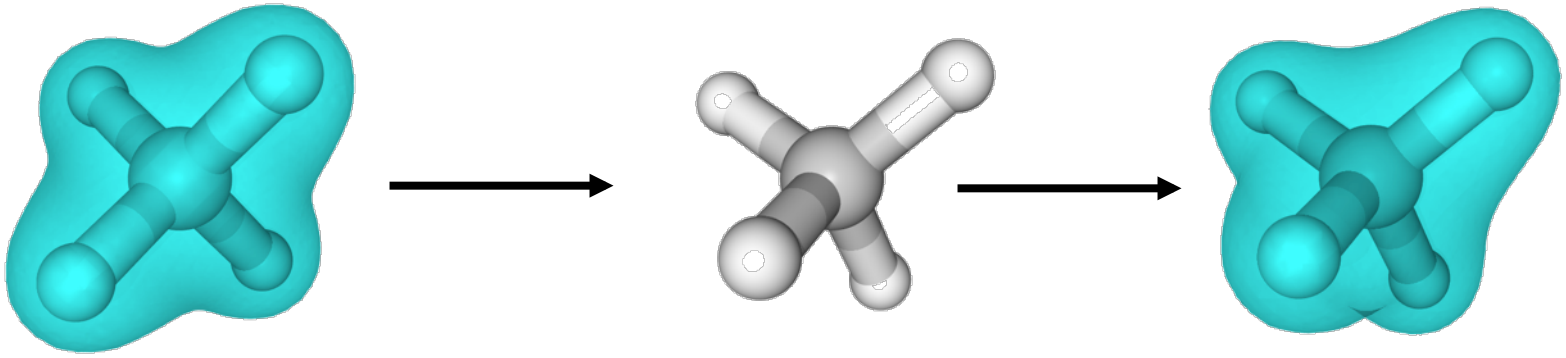
APPLICATION SCALABILITY

Example: QBox

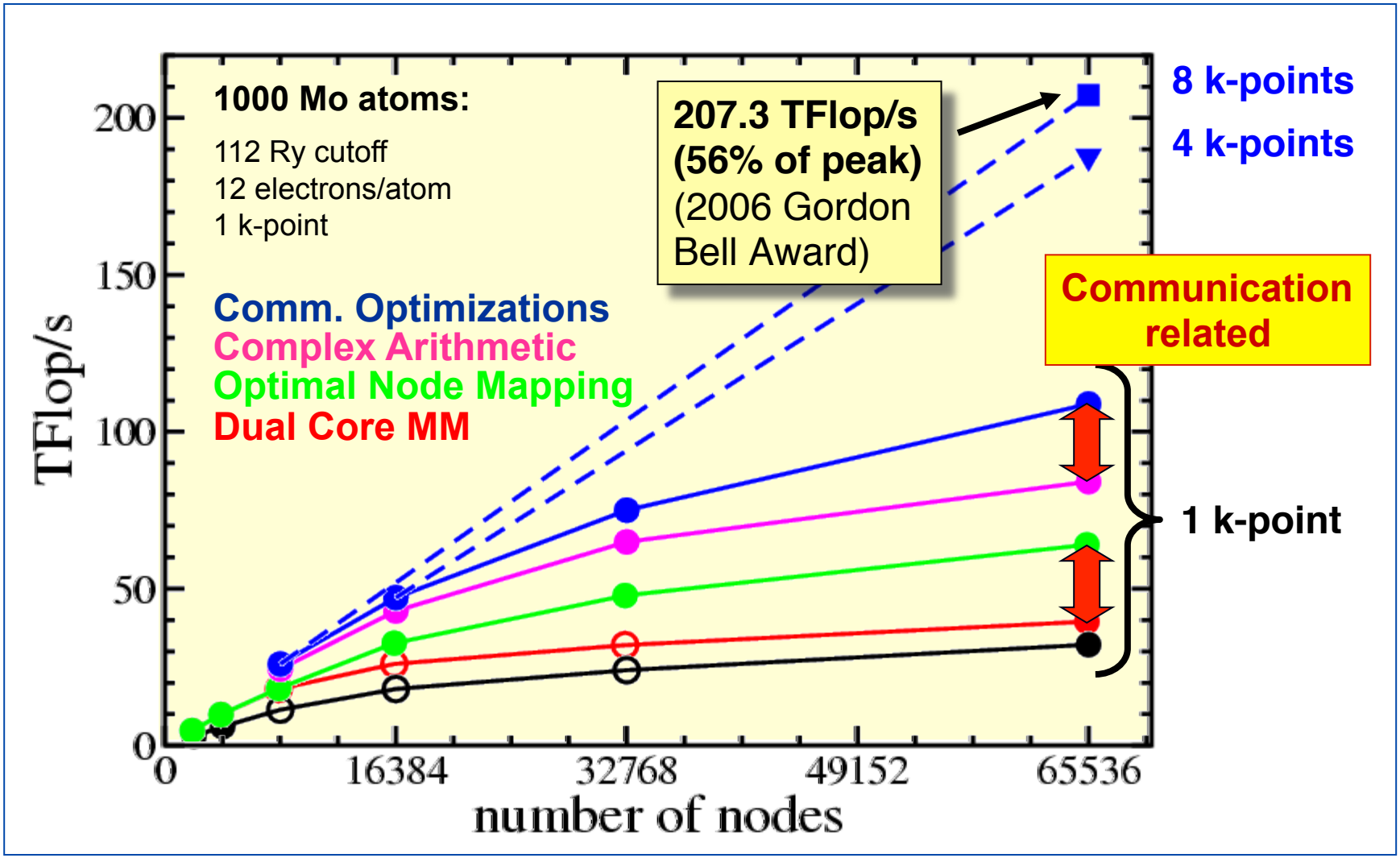
- Material Simulation
- First Principles Method
  - No empirical parameters
  - Chemically dynamic
    - Iterative process
    - Computationally intensive
- Gordon Bell Winner in 2006



Electron density surrounding water molecules, calculated from first-principles

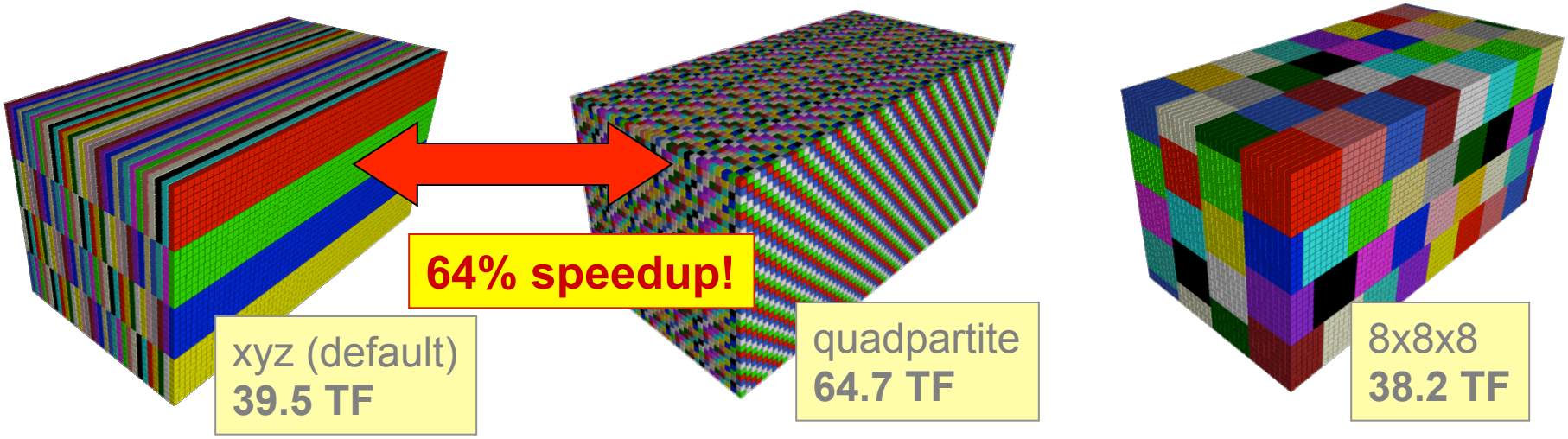


QBox Performance



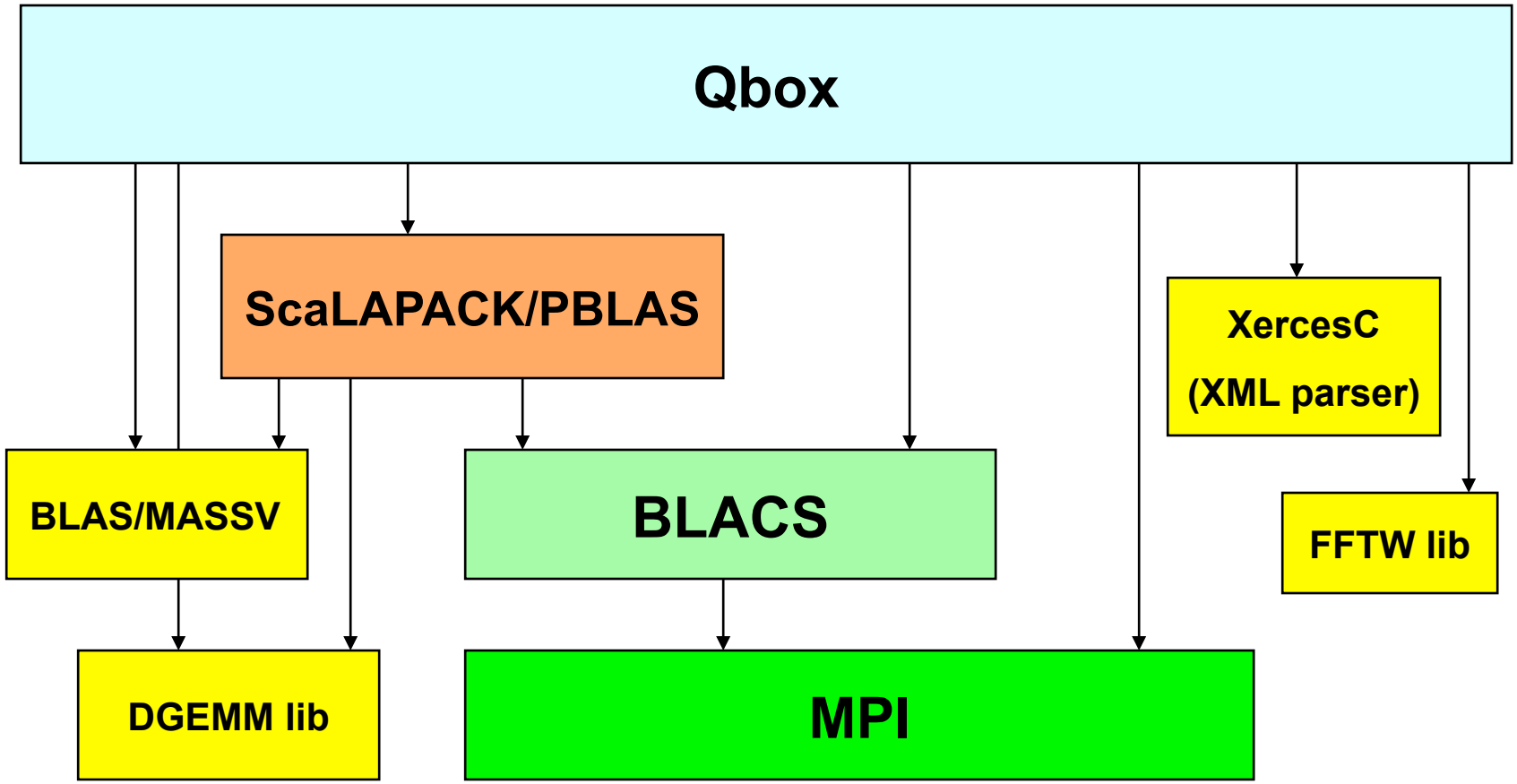
Biggest Improvements: Node Mappings

- Node mappings are essential for good performance
  - Base data structure: dense 2D matrix
  - Need to map rows/columns onto 3D torus of BG/L
- Unexpected node mapping onto 64x32x32 torus



- Large optimization potential/need
  - Most effects appeared/understood only at scale
  - But: concrete communication patterns are unknown

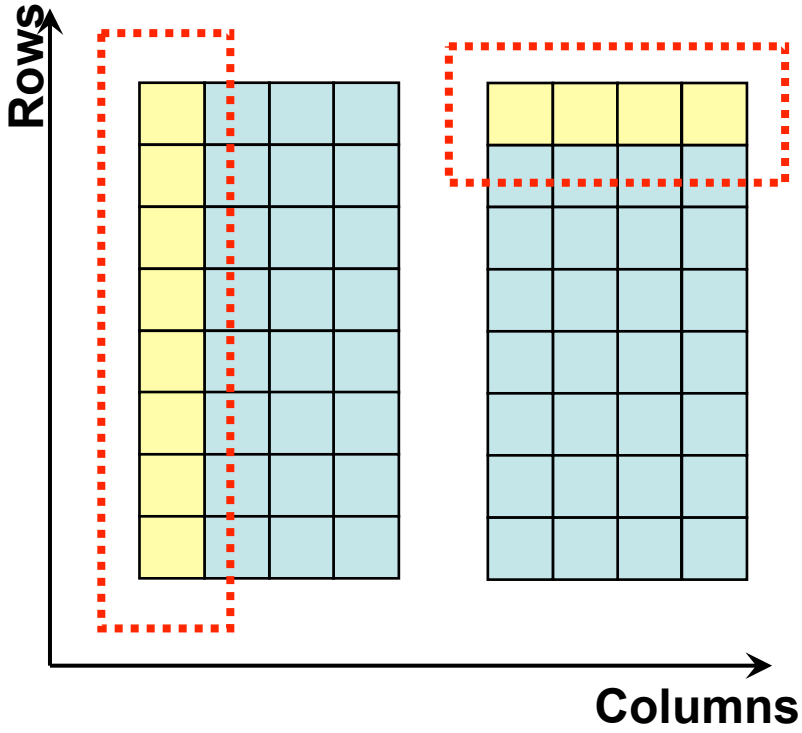
QBox Code Structure



- Implicit generation of communicators
- Frequent creation and destruction

Distinguishing Communicators in QBox

- Dense matrix
  - Row and column communicators
  - Global operations
- Row vs. column behavior
  - Need to optimize for both
  - Tradeoffs not straightforward
- Need to profile separately
  - Different operations
  - Separate optimization
- Challenges
  - Identify application behavior in black box libraries
  - Customize profiler to separate row and column behavior

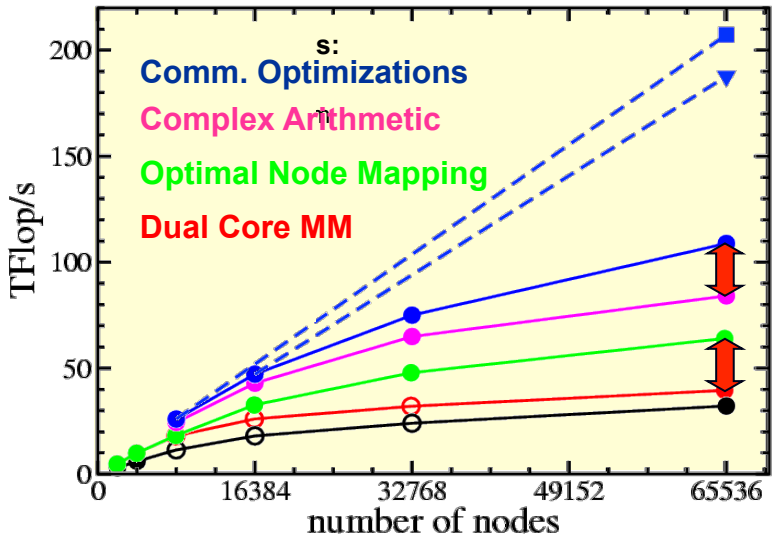


Communicator Profiling Results for QBox

Operation	Sum	Global	Row	Column
Send	317245	31014	202972	83259
Allreduce	319028	269876	49152	0
Alltoallv	471488	471488	0	0
Recv	379265	93034	202972	83259
Bcast	401042	11168	331698	58176

AMD Opteron/Infiniband Cluster

- Lessons learned from QBox
  - Node mappings are critical
  - Performance effects often show only at scale
  - Need to understand behavior and customize tool behavior
  - Need for tools to break black box abstractions



2009 Jülich BlueGene/P Extreme Scaling Workshop

- October 26 - 28, 2009
- 10 participating teams from Harvard University, MIT, ANL, Rensselaer Polytechnic Institute, University of Edinburgh, Swiss Institute of Technology, Instituto Superior Técnico (Lisbon) RZG MPG, DESY Zeuthen and JSC.
- 398 jobs with 135.6 rack days (out of 169.3 rack days provided): 80% utilization during workshop!
- All but 1 team succeeded in executing their code on full machine (294,912 cores)
- Report available at http://www.fz-juelich.de/jsc/files/docs/ib/ib-10/ib-2010-02.pdf

2009 Scaling Workshop Applications

- QCD, EPCC, UK
- Gyrokinetic Turbulence Simulation, RZG/IPP, DE
- Neutron Transport Simulations, ANL, US
  - **Gordon Bell Finalist**
- Astro-physics (particle-in-cell), UTL, PT
- Simulation of coronary arteries, EPFL, CH
- Parallel Evolutionary Biology Suite, Harvard, US
- Adaptive computational fluid dynamics, RPI, US
  - **Gordon Bell Finalist**
- Mesoscopic Particle Dynamics, JSC, DE
- QCD, Hungary/France/German HMC Collaboration
- QCD, DESY/Bonn Univ., DE

2010 Jülich BlueGene/P Extreme Scaling Workshop

- March 22 - 24, 2010
- 10 participating teams from Harvard University, ANL, CORIA (France), ETH Zurich, KIT, LRZ, ZIB, JSC, and the Universities of Marburg, Chemnitz, and Erlangen.
- 392 jobs with 138.7 rack days (out of 164 rack days provided): 84% utilization
- 6 teams succeeded in executing their code on full machine (including team that failed 2009)
- 3 teams could “only” scale to 64 racks (262,144 cores) due to the need to run on a power-of-two number of cores
- 1 team got stuck at 32 racks (131,072 cores) due to program bug
- Report available at http://www.fz-juelich.de/jsc/files/docs/ib/ib-10/ib-2010-03.pdf

2010 Scaling Workshop Applications

- Particle Flow Simulation, Erlangen Univ., DE
- Simulation of Fluid Flow and Mass Transport, Marburg Univ., DE
- Turbulent Flows in Complex Geometries, CORIA, FR
- Spectral Element Code, ETH, CH / ANL, US
- Simulation of Coronary Arteries, Harvard/EPFL
  - **Gordon Bell Finalist**
  - **George Michael Memorial PhD Fellowship (A. Peters)**
- Parallel Fast Fourier Transform, Chemnitz Unv., DE
- QCD, ZIB/LRZ, DE
- Mesoscopic Particle Dynamics, JSC, DE
- Hydrodynamic Turbulence, KIT, DE
- Large-Scale Density-Functional Calculations, JSC, DE

2011 Jülich BlueGene/P Extreme Scaling Workshop

- February 14 - 16, 2011
- 8 participating teams from KTH (SE), KAUST (SA), Princeton PPL, RZG MPG, Mickiewicz University (PL), University College London (UK), Euskal Herriko Unibertsitatea (ES), University of Heidelberg
- Selected from 15 proposals
- 308 jobs with 122 rack days (out of 157 rack days provided): 77% utilization
- Teams succeeded in executing 11(!) codes on full machine
- Report available at http://www2.fz-juelich.de/jsc/files/docs/ib/ib-11/ib-2011-02.pdf

2011 Scaling Workshop Applications

- 2 x Neuronal Network Simulations, KTH, SE
- Molecular Dynamics (BBMD), KAUST, SA
- Gyrokinetic PIC Simulations (GTC-P), Princeton PPL, US
- Eigenvalue Solver (ELPA) +
  Ab-initio Molecular Simulation, RZG MPG, DE
- Molecular Nanomagnets (QTM), Mickiewicz University, PL
- Lattice Boltzmann (LB3D), University College London, UK
- Time-dependent density functional theory (Octopus),
  Euskal Herriko Unibertsitatea, ES
- Algebraic Multigrid Solver (Muphi/DUNE-ISTL),
  University of Heidelberg, DE

Lessons from Scaling Workshops

- Applications from a wide range of subject areas scale to very large numbers of processes (O(300k))
  - Weak scaling is easiest, but good strong scaling is also possible
  - Most employ MPI, increasingly combined with OpenMP threading (and other hybrid parallelizations)
    - QCD “bare-metal” parallelizations are the exception
  - Message-passing needs to be local (close neighbours) and appropriate placement (rank-reordering) may be necessary
  - Asynchronous communication is generally beneficial
    - Even with limited overlap of computation
  - Implicit collective communication synchronizations and inherent computation/communication imbalances grow to dominate at scale
  - Effective parallel file I/O is critical
    - (see other SC tutorials for specifics)
- Each doubling of scale exposes a new performance bottleneck or bug!

PRESENT: SCALABLE PROFILING

Scalable Profiling Approaches

- **MPI wrapper profiling** with **summarization at end of run**
  - mpiP [LLNL et al.: http://mpip.sourceforge.net]
    - single text output file
    - data for all ranks
  - FPMPI-2 [ANL: http://www.mcs.anl.gov/fpmpi/]
    - **special**: Optionally **identifies synchronization time**
    - single text output file
    - count, sum, avg, min, max over ranks

  - IBM HPCT [IBM ACTC]
    - four text output files
    - rank 0 + ranks with min/median/max MPI time

Scalable Profiling Approaches II

- **Sampling-based call stack profiling**
  - hpctoolkit [Rice Univ.: http://hpctoolkit.org]
    - one output file per process
    - Scalable post-processing with separate MPI program "hpcprof_mpi"
    - Comprehensive loop and basic block analysis
  - CrayPat [Cray]
    - single binary output file
    - data for all ranks
    - Post-processing with "`pat_report`" into text report
    - Also: Apprentice2 GUI
    - Special load imbalance metrics

Scalable Profiling Approaches III

- **Sampling-based user code + MPI wrapper profiling**
  - TAU [UO: http://tau.uoregon.edu/]
    - One of many measurement options of TAU (see also next slide)
    - Invoked via "`tau_exec -ebs ...`"
  - Open|SpeedShop [Krell: http://www.openspeedshop.org/]
    - Sampling provides online and postmortem results
    - Data stored in SQL database / GUI to display
    - Time and HW counter metrics, with/without callpath
    - Focus on ease of use through convenience scripts

Scalable Profiling Approaches IV

- **Instrumentation-based user code and MPI profiling**
  - TAU [UO: http://tau.uoregon.edu/]
    - **Parallel summarization at analysis time**
    - **Scalable (3D) result displays**
    - Function, call path, or phase profiling
    - Multiple metrics (time, HW counter, memory, …)
  - Scalasca [JSC: http://www.scalasca.org]
    - **summarization at end of run** (using MPI)
    - single (but 3dim) output file (metric/call path/thread)
    - call path profiling **+ scalable 3D metrics browser**
    - multiple metrics (time, HW counter, msg count+bytes)

mpiP: Efficient MPI Profiling

- Open source MPI profiling library
  - Developed at LLNL, maintained by LLNL & ORNL
  - Available from sourceforge
  - Works with any MPI library

- Easy-to-use and portable design
  - Relies on PMPI instrumentation
  - No additional tool daemons or support infrastructure
  - Single text file as output
  - Optional: GUI viewer

Running with mpiP 101 / Experimental Setup

- mpiP works on binary files
  - Uses standard development chain
  - Use of “-g” recommended
- Run option 1: Relink
  - Specify libmpi.a/.so on the link line
  - Portable solution, but requires object files
- Run option 2: library preload
  - Set preload variable (e.g., LD_PRELOAD) to mpiP
  - Transparent, but only on supported systems

Running with mpiP 101 / Running

```
bash-3.2$ srun –n4 smg2000
mpiP:
mpiP:
mpiP: mpiP V3.1.2 (Build Dec 16 2008/17:31:26)
mpiP: Direct questions and errors to mpip-
help@lists.sourceforge.net
mpiP:
Running with these driver parameters:
  (nx, ny, nz)    = (60, 60, 60)
  (Px, Py, Pz)    = (4, 1, 1)
  (bx, by, bz)    = (1, 1, 1)
  (cx, cy, cz)    = (1.000000, 1.000000, 1.000000)
  (n_pre, n_post) = (1, 1)
  dim             = 3
  solver ID       = 0
=============================================
Struct Interface:
=============================================
Struct Interface:
  wall clock time = 0.075800 seconds
  cpu clock time  = 0.080000 seconds
```

Header

```
=============================================
Setup phase times:
=============================================
SMG Setup:
  wall clock time = 1.473074 seconds
  cpu clock time  = 1.470000 seconds
=============================================
Solve phase times:
=============================================
SMG Solve:
  wall clock time = 8.176930 seconds
  cpu clock time  = 8.180000 seconds

Iterations = 7
Final Relative Residual Norm = 1.459319e-07

mpiP:
mpiP: Storing mpiP output in [./smg2000-p.4.11612.1.mpiP].
mpiP:
bash-3.2$
```

Output File

mpiP 101 / Output – Metadata

```
@ mpiP
@ Command : ./smg2000-p -n 60 60 60
@ Version                  : 3.1.2
@ MPIP Build date          : Dec 16 2008, 17:31:26
@ Start time               : 2009 09 19 20:38:50
@ Stop time                : 2009 09 19 20:39:00
@ Timer Used               : gettimeofday
@ MPIP env var             : [null]
@ Collector Rank           : 0
@ Collector PID            : 11612
@ Final Output Dir         : .
@ Report generation        : Collective
@ MPI Task Assignment      : 0 hera27
@ MPI Task Assignment      : 1 hera27
@ MPI Task Assignment      : 2 hera31
@ MPI Task Assignment      : 3 hera31
```

mpiP 101 / Output – Overview

```
---------------------------------------------------------------
@--- MPI Time (seconds) ---------------------------------------
---------------------------------------------------------------
Task    AppTime    MPITime     MPI%
   0       9.78       1.97    20.12
   1        9.8       1.95    19.93
   2        9.8       1.87    19.12
   3       9.77       2.15    21.99
   *       39.1       7.94    20.29
---------------------------------------------------------------
```

mpiP 101 / Output – Callsites

```
---------------------------------------------------------------------
@--- Callsites: 23 ----------------------------------------------------
---------------------------------------------------------------------
 ID Lev File/Address        Line Parent_Funct                  MPI_Call
  1   0 communication.c     1405 hypre_CommPkgUnCommit         Type_free
  2   0 timing.c             419 hypre_PrintTiming             Allreduce
  3   0 communication.c      492 hypre_InitializeCommunication Isend
  4   0 struct_innerprod.c   107 hypre_StructInnerProd         Allreduce
  5   0 timing.c             421 hypre_PrintTiming             Allreduce
  6   0 coarsen.c            542 hypre_StructCoarsen           Waitall
  7   0 coarsen.c            534 hypre_StructCoarsen           Isend
  8   0 communication.c     1552 hypre_CommTypeEntryBuildMPI   Type_free
  9   0 communication.c     1491 hypre_CommTypeBuildMPI        Type_free
 10   0 communication.c      667 hypre_FinalizeCommunication   Waitall
 11   0 smg2000.c            231 main                          Barrier
 12   0 coarsen.c            491 hypre_StructCoarsen           Waitall
 13   0 coarsen.c            551 hypre_StructCoarsen           Waitall
 14   0 coarsen.c            509 hypre_StructCoarsen           Irecv
 15   0 communication.c     1561 hypre_CommTypeEntryBuildMPI   Type_free
 16   0 struct_grid.c        366 hypre_GatherAllBoxes          Allgather
 17   0 communication.c     1487 hypre_CommTypeBuildMPI        Type_commit
 18   0 coarsen.c            497 hypre_StructCoarsen           Waitall
 19   0 coarsen.c            469 hypre_StructCoarsen           Irecv
 20   0 communication.c     1413 hypre_CommPkgUnCommit         Type_free
 21   0 coarsen.c            483 hypre_StructCoarsen           Isend
 22   0 struct_grid.c        395 hypre_GatherAllBoxes          Allgatherv
 23   0 communication.c      485 hypre_InitializeCommunication Irecv
---------------------------------------------------------------------
```

mpiP 101 / Output – per Function Timing

```
---------------------------------------------------------------
@--- Aggregate Time (top twenty, descending, milliseconds) ---
---------------------------------------------------------------
Call                 Site       Time    App%    MPI%      COV
Waitall                10    4.4e+03   11.24   55.40     0.32
Isend                   3   1.69e+03    4.31   21.24     0.34
Irecv                  23        980    2.50   12.34     0.36
Waitall                12        137    0.35    1.72     0.71
Type_commit            17        103    0.26    1.29     0.36
Type_free               9       99.4    0.25    1.25     0.36
Waitall                 6       81.7    0.21    1.03     0.70
Type_free              15       79.3    0.20    1.00     0.36
Type_free               1       67.9    0.17    0.85     0.35
Type_free              20       63.8    0.16    0.80     0.35
Isend                  21         57    0.15    0.72     0.20
Isend                   7       48.6    0.12    0.61     0.37
Type_free               8       29.3    0.07    0.37     0.37
Irecv                  19       27.8    0.07    0.35     0.32
Irecv                  14       25.8    0.07    0.32     0.34
...
```

mpiP 101 / Output – per Function Message Size

```
---------------------------------------------------------------
@--- Aggregate Sent Message Size (top twenty, descending, bytes ----
---------------------------------------------------------------
Call                 Site       Count      Total       Avrg  Sent%
Isend                   3      260044    2.3e+08        885  99.63
Isend                   7        9120   8.22e+05       90.1   0.36
Isend                  21        9120   3.65e+04          4   0.02
Allreduce               4          36        288          8   0.00
Allgatherv             22           4        112         28   0.00
Allreduce               2          12         96          8   0.00
Allreduce               5          12         96          8   0.00
Allgather              16           4         16          4   0.00
```

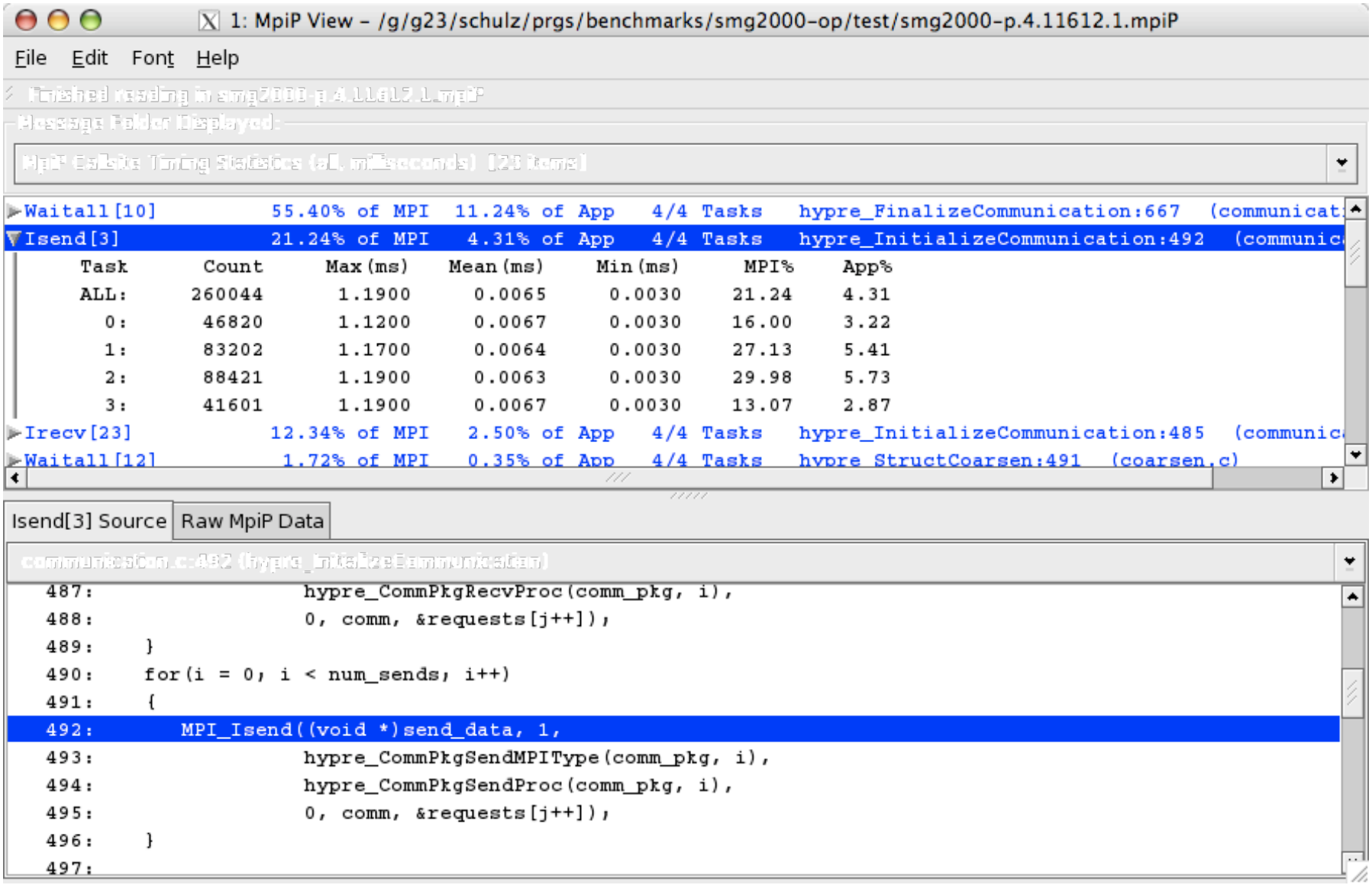
mpiP 101 / Output – per Callsite Timing

```
---------------------------------------------------------------------
@--- Callsite Time statistics (all, milliseconds): 92 ---------------
---------------------------------------------------------------------
Name                Site  Rank  Count      Max      Mean       Min   App%   MPI%
Allgather             16     0      1    0.034     0.034     0.034   0.00   0.00
Allgather             16     1      1    0.049     0.049     0.049   0.00   0.00
Allgather             16     2      1     2.92      2.92      2.92   0.03   0.16
Allgather             16     3      1        3         3         3   0.03   0.14
Allgather             16     *      4        3       1.5     0.034   0.02   0.08

Allgatherv            22     0      1     0.03      0.03      0.03   0.00   0.00
Allgatherv            22     1      1    0.036     0.036     0.036   0.00   0.00
Allgatherv            22     2      1    0.022     0.022     0.022   0.00   0.00
Allgatherv            22     3      1    0.022     0.022     0.022   0.00   0.00
Allgatherv            22     *      4    0.036    0.0275     0.022   0.00   0.00

Allreduce              2     0      3    0.382     0.239     0.011   0.01   0.04
Allreduce              2     1      3     0.31     0.148     0.046   0.00   0.02
Allreduce              2     2      3    0.411     0.178     0.062   0.01   0.03
Allreduce              2     3      3     1.33     0.622     0.062   0.02   0.09
Allreduce              2     *     12     1.33     0.297     0.011   0.01   0.04

...
```

mpiP 101 / Output – per Callsite Message Size

```
---------------------------------------------------------------------------
@--- Callsite Message Sent statistics (all, sent bytes) -------------------
---------------------------------------------------------------------------
Name                 Site Rank   Count       Max      Mean       Min       Sum
Allgather              16    0       1         4         4         4         4
Allgather              16    1       1         4         4         4         4
Allgather              16    2       1         4         4         4         4
Allgather              16    3       1         4         4         4         4
Allgather              16    *       4         4         4         4        16

Allgatherv             22    0       1        28        28        28        28
Allgatherv             22    1       1        28        28        28        28
Allgatherv             22    2       1        28        28        28        28
Allgatherv             22    3       1        28        28        28        28
Allgatherv             22    *       4        28        28        28       112

Allreduce               2    0       3         8         8         8        24
Allreduce               2    1       3         8         8         8        24
Allreduce               2    2       3         8         8         8        24
Allreduce               2    3       3         8         8         8        24
Allreduce               2    *      12         8         8         8        96
```

GUI for mpiP based on Tool Gear



Advanced Features

- Fine tuning the profiling
  - User controlled stack trace depth
  - Reduced output for large scale experiments
  - Application control to limit scope
  - Measurements for MPI I/O routines

- Controlled by MPIP environment variable
  - Set by user before profile run
  - Command line style argument list
  - Example:
    - setenv MPIP “-c –o –k 4”

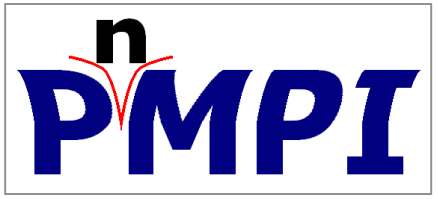
mpiP: 8192 Core Run Text Output Example

```
@ mpiP
@ Version: 3.1.1
// 10 lines of mpiP and experiment configuration options
// 8192 lines of task assignment to BlueGene topology information

@--- MPI Time (seconds) ---------------------------------------
Task    AppTime    MPITime     MPI%
   0       37.7       25.2    66.89
// ...
8191       37.6         26    69.21
   *   3.09e+05   2.04e+05    65.88

@--- Callsites: 26 --------------------------------------------
 ID Lev File/Address        Line Parent_Funct             MPI_Call
  1   0 coarsen.c            542 hypre_StructCoarsen      Waitall
// 25 similar lines

@--- Aggregate Time (top twenty, descending, milliseconds) --------
Call                 Site       Time    App%    MPI%      COV
Waitall                21   1.03e+08   33.27   50.49    0.11
Waitall                 1   2.88e+07    9.34   14.17    0.26
// 18 similar lines
```

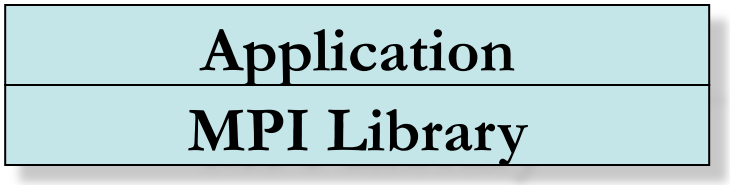
mpiP: 8192 Core Run Text Output Example (cont.)

```
@--- Aggregate Sent Message Size (top twenty, descending, bytes) --
Call                 Site       Count      Total       Avrg  Sent%
Isend                  11   845594460   7.71e+11        912  59.92
Allreduce              10       49152   3.93e+05          8   0.00
// 6 similar lines

@--- Callsite Time statistics (all, milliseconds): 212992 ---------
Name       Site Rank     Count    Max   Mean      Min   App%   MPI%
Waitall      21    0    111096    275    0.1 0.000707  29.61  44.27
//  ...
Waitall      21 8191     65799    882   0.24 0.000707  41.98  60.66
Waitall      21    * 577806664    882  0.178 0.000703  33.27  50.49
// 213,042 similar lines

@--- Callsite Message Sent statistics (all, sent bytes) -----------
Name       Site Rank     Count       Max      Mean   Min         Sum
Isend        11    0     72917 2.621e+05     851.1     8  6.206e+07
//...
Isend        11 8191     46651 2.621e+05      1029     8  4.801e+07
Isend        11    * 845594460 2.621e+05     911.5     8  7.708e+11
// 65,550 similar lines
```

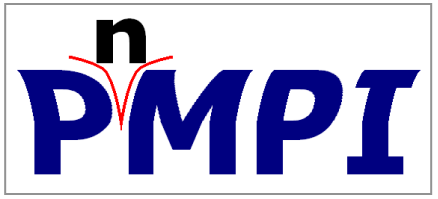
Customizing Profiling with P^{N}MPI



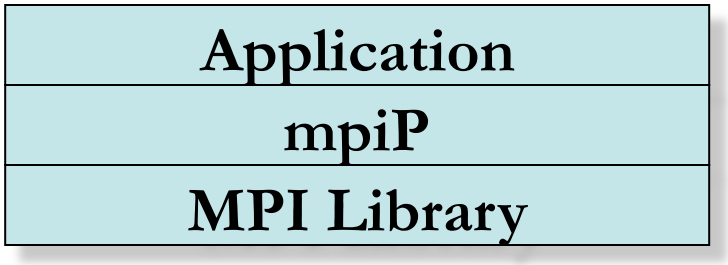
- Need to separate context
  - Qbox example
  - Changing profiler too complex



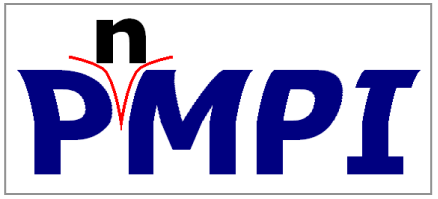
Customizing Profiling with P^{N}MPI



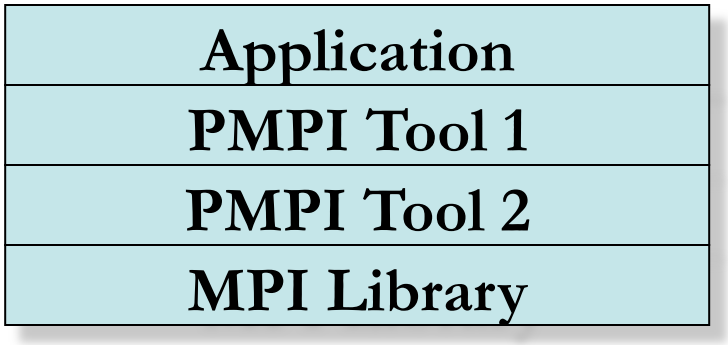
- PMPI interception of MPI calls
  - Easy to include in applications
  - Limited to a single tool



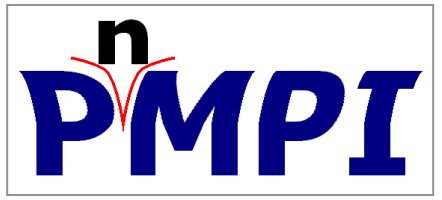
Customizing Profiling with P^{N}MPI



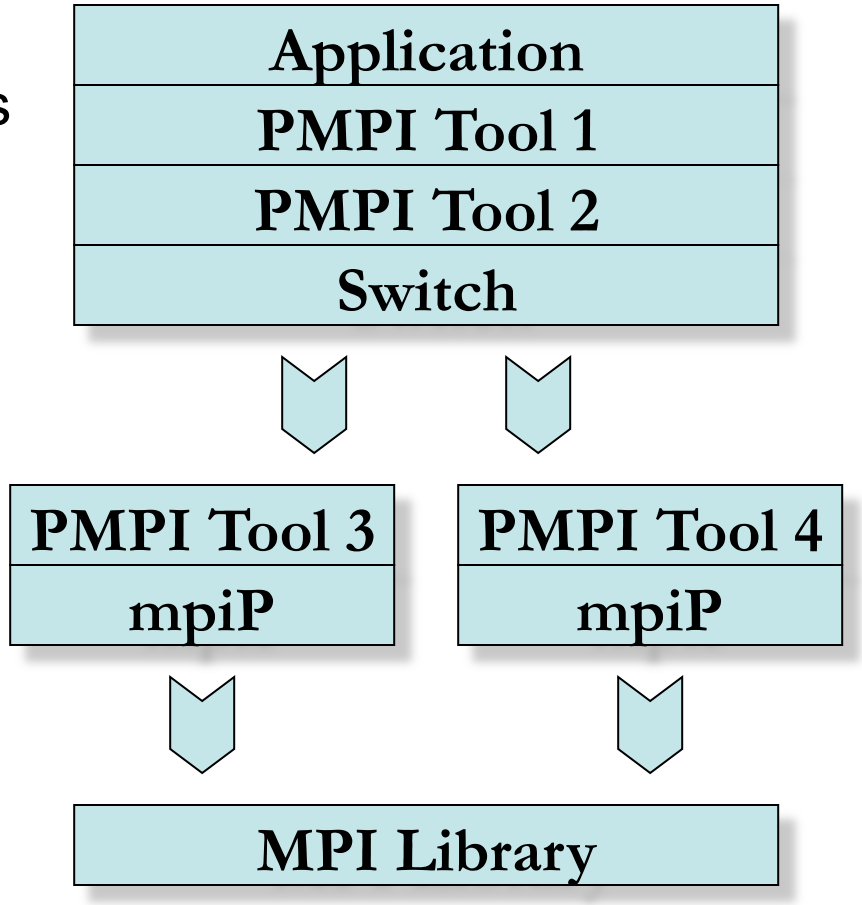
- PMPI interception of MPI calls
  - Easy to include in applications
  - Limited to a single tool
- P^{N}MPI virtualized PMPI
  - Multiple tools concurrently
  - Dynamic loading of tools
  - Configuration through text file
  - Tools are independent
  - Tools can collaborate



Customizing Profiling with P^{N}MPI

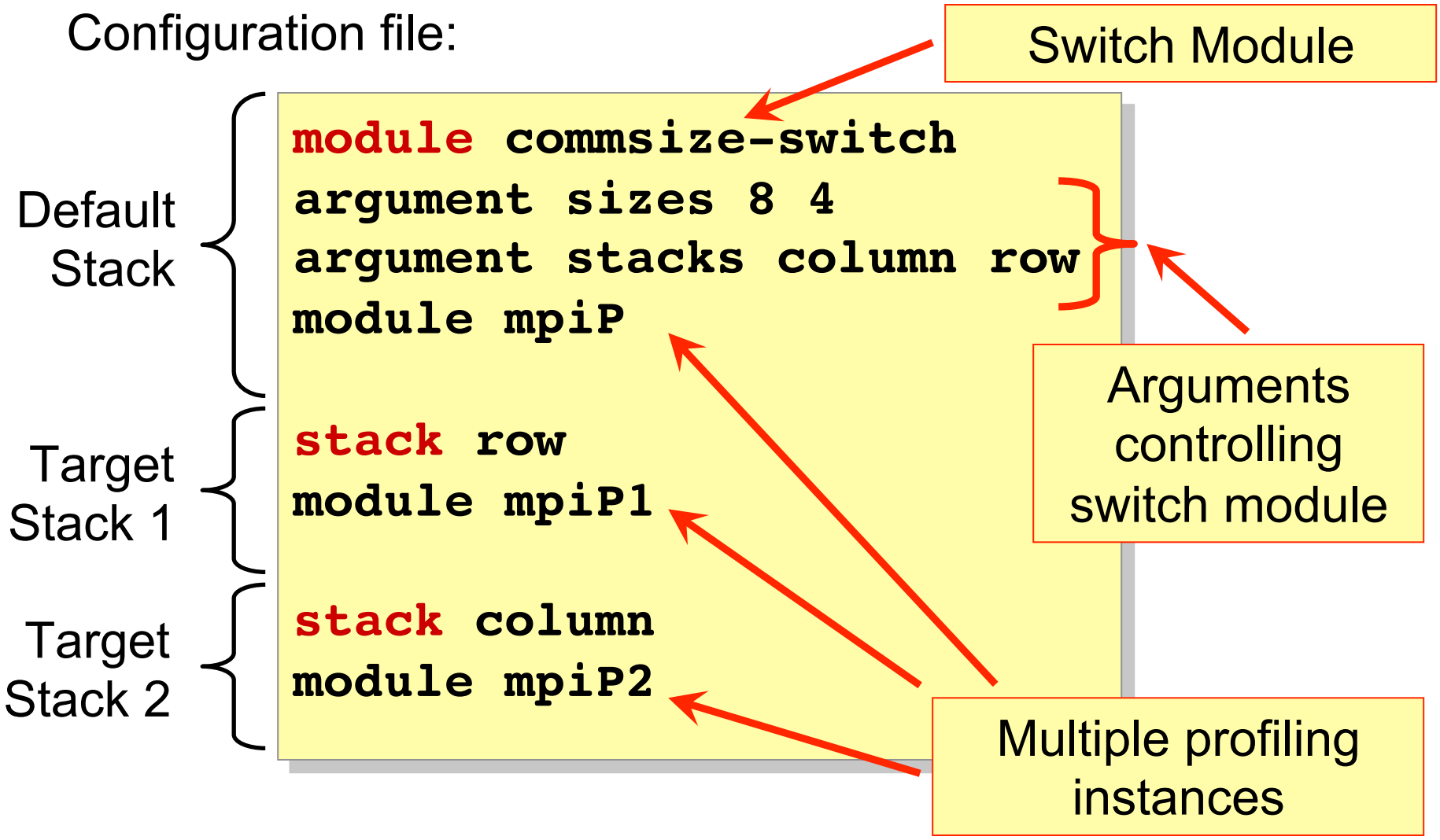


- PMPI interception of MPI calls
  - Easy to include in applications
  - Limited to a single tool
- P^{N}MPI virtualized PMPI
  - Multiple tools concurrently
  - Dynamic loading of tools
  - Configuration through text file
  - Tools are independent
  - Tools can collaborate
- Transparently adding context
  - Select tool based on MPI context
  - Transparently isolate tool instances



Example: Defining Switch Modules in P^NMPI

Configuration file:

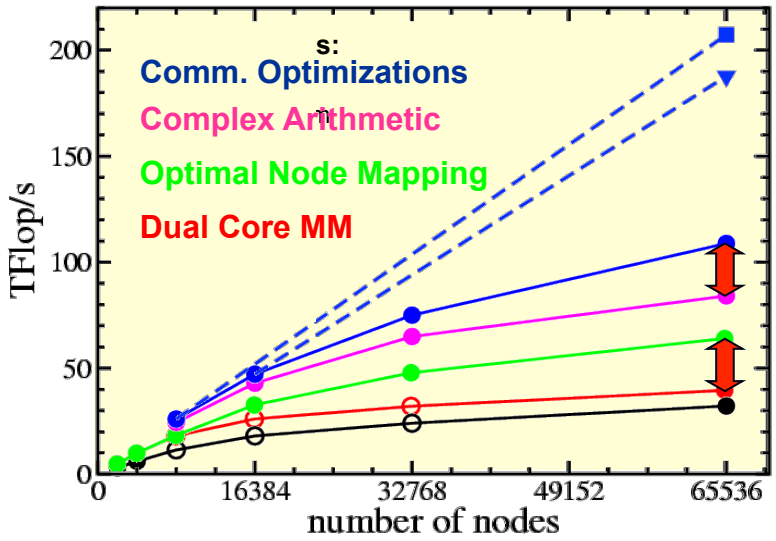


Flashback: Communicator Profiling Results for QBox

Operation	Sum	Global	Row	Column
Send	317245	31014	202972	83259
Allreduce	319028	269876	49152	0
Alltoallv	471488	471488	0	0
Recv	379265	93034	202972	83259
Bcast	401042	11168	331698	58176

AMD Opteron/Infiniband Cluster

- Lessons learned from QBox
  - Node mappings are critical
  - Performance effects often show only at scale
  - Need to understand behavior and customize tool behavior
  - Need for tools to break black box abstractions



Scalable Profiling Approaches

- **MPI wrapper profiling** with **summarization at end of run**
  - mpiP [LLNL et al.: http://mpip.sourceforge.net]
    - single text output file
    - data for all ranks
  - FPMPI-2 [ANL: http://www.mcs.anl.gov/fpmpi/]
    - **special**: Optionally **identifies synchronization time**
    - single text output file
    - count, sum, avg, min, max over ranks

  - IBM HPCT [IBM ACTC]
    - four text output files
    - rank 0 + ranks with min/median/max MPI time

MPI Profiling: FPMPI-2

- **Scalable, light-weight MPI profiling library with special features**
- **1st special:** Distinguishes between messages of different sizes within 32 message bins (essentially powers of two)
- **2nd special:** Optionally **identifies synchronization time**
  - On synchronizing collective calls
    - Separates waiting time from collective operation time
  - On blocking sends
    - Determines the time until the matching receive is posted
  - On blocking receives
    - Determines time the receive waits until the message arrives
  - All implemented with MPI calls
    - Pro: Completely portable
    - Con: Adds overhead (e.g., MPI_Send ⇨ MPI_Issend/Test)
- http://www.mcs.anl.gov/fpmpi/

FMPI2 Output Example

```
MPI Routine Statistics (FPMPI2 Version 2.1e)
Processes:      8192
Execute time:   52.77
Timing Stats: [seconds] [min/max]          [min rank/max rank]
  wall-clock: 52.77 sec 52.770000 / 52.770000     0 / 0
        user: 52.79 sec 52.794567 / 54.434833   672 / 0
         sys: 0 sec      0.000000 / 0.000000     0 / 0

                Average of sums over all processes
Routine         Calls      Time Msg Length   %Time by message length
                                          0.........1........1....
                                                     K        M
MPI_Allgather :     1  0.000242          4 0*00000000000000000000
MPI_Allgatherv:     1   0.00239         28 0000*0000000000000000000
MPI_Allreduce :    12  0.000252         96 00*000000000000000000000
MPI_Reduce    :     2     0.105          8 0*00000000000000000000
MPI_Isend     : 233568      1.84   2.45e+08 01.....1112111...00000000
MPI_Irecv     : 233568     0.313   2.45e+08 02...111112.....00000000
MPI_Waitall   :  89684      23.7
MPI_Barrier   :     1  0.000252
```

FMPI2 Output Example (cont.)

```
Details for each MPI routine
                                                  % by message length
                            (max over             0.........1........1…....
                             processes [rank])              K         M
MPI_Isend:
  Calls      :      233568       436014 [3600] 02...111122......0000000
  Time       :        1.84         2.65 [3600] 01.....1112111...0000000
  Data Sent  :    2.45e+08    411628760 [3600]
  By bin     : 1-4             [13153,116118]    [   0.0295,    0.234]
             : 5-8               [2590,28914]    [  0.00689,   0.0664]
// ...
             : 131073-262144          [8,20]    [   0.0162,   0.0357]
  Partners   :         245 max 599(at 2312) min 47(at 1023)
MPI_Waitall:
  Calls      :       89684
  Time       :        23.7
  SyncTime   :        6.07

// Similar details for other MPI routines
```

mp_profiler MPI Profiling Libraries

- Part of IBM High Performance Computing Toolkit

- PMPI wrapper library to collect runtime profiles
  - #calls and total time spent per MPI routine
  - Message size distributions for communication routines
- Text and XML output
- **Very flexible configuration** via
  - environment variables
  - C configuration function interface (for experts)

- By default
  - four output files
  - rank 0 + ranks with min/median/max MPI time

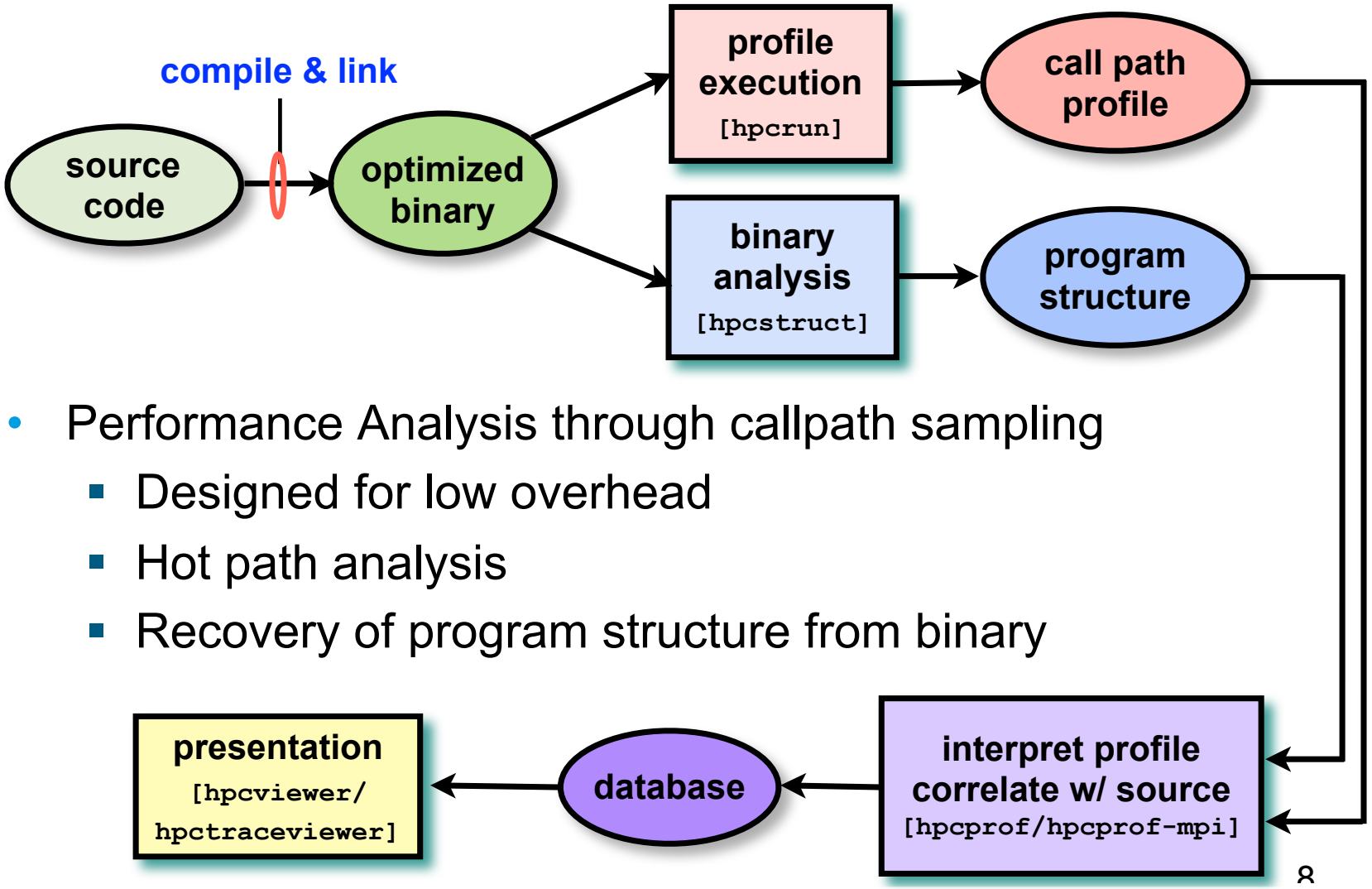
MP-Profiler: mpitrace output (rank 0)

```
elapsed time from clock-cycles using freq = 700.0 MHz

-----------------------------------------------------------
MPI Routine                #calls     avg. bytes      time(sec)
-----------------------------------------------------------
MPI_Comm_size               31696            0.0          0.009
MPI_Comm_rank               36391            0.0          0.006
MPI_Isend                  125475          585.9          0.789
MPI_Irecv                  125242          681.9          0.136
MPI_Waitall                121060            0.0         33.116
MPI_Barrier                     1            0.0          0.000
MPI_Allgather                   1            4.0          0.000
MPI_Allgatherv                  1           28.0          0.002
MPI_Allreduce                  12            8.0          2.591
-----------------------------------------------------------
total communication time = 36.650 seconds.
total elapsed time       = 49.904 seconds.
```

MP-Profiler: mpitrace output (rank 0, cont.)

```
---------------------------------------------------------------
Message size distributions:
MPI_Isend                    #calls    avg. bytes      time(sec)
                              26279           4.0          0.054
                               7582           8.0          0.018
// ...
                                  5      262144.0          0.004
// Similar details for other MPI routines

---------------------------------------------------------------
Communication summary for all tasks:
  minimum communication time = 27.538 sec for task 3600
  median  communication time = 32.563 sec for task 4027
  maximum communication time = 37.557 sec for task 63

MPI tasks sorted by communication time:
taskid  xcoord  ycoord  zcoord  procid    total_comm(sec)
  3600      16       0      14       0          27.538
// ...
    63      31       1       0       0          37.557
```

HPCToolkit / Rice University



- Performance Analysis through callpath sampling
  - Designed for low overhead
  - Hot path analysis
  - Recovery of program structure from binary

Image by John Mellor-Crummey

HPCToolkit: hpcviewer

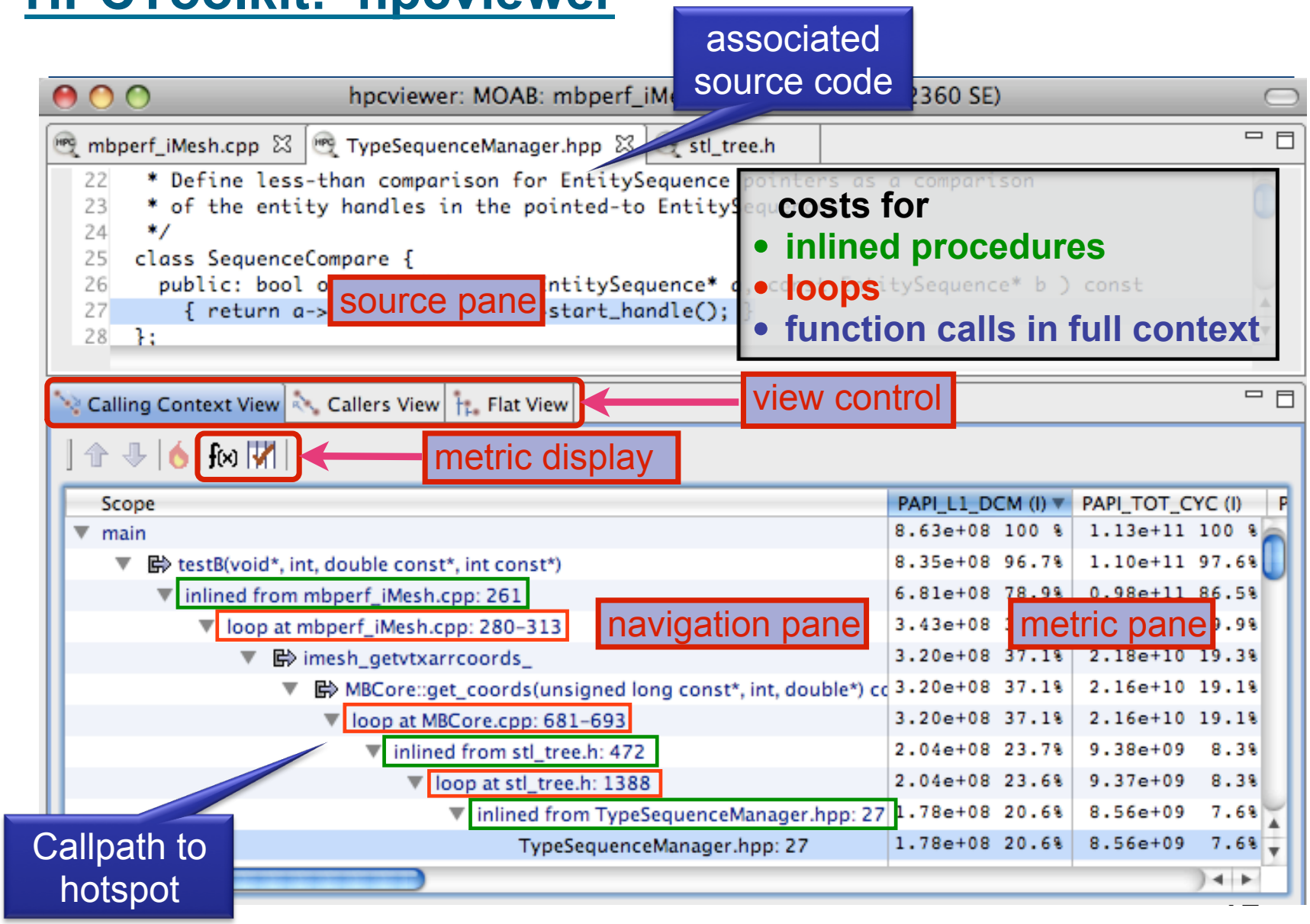
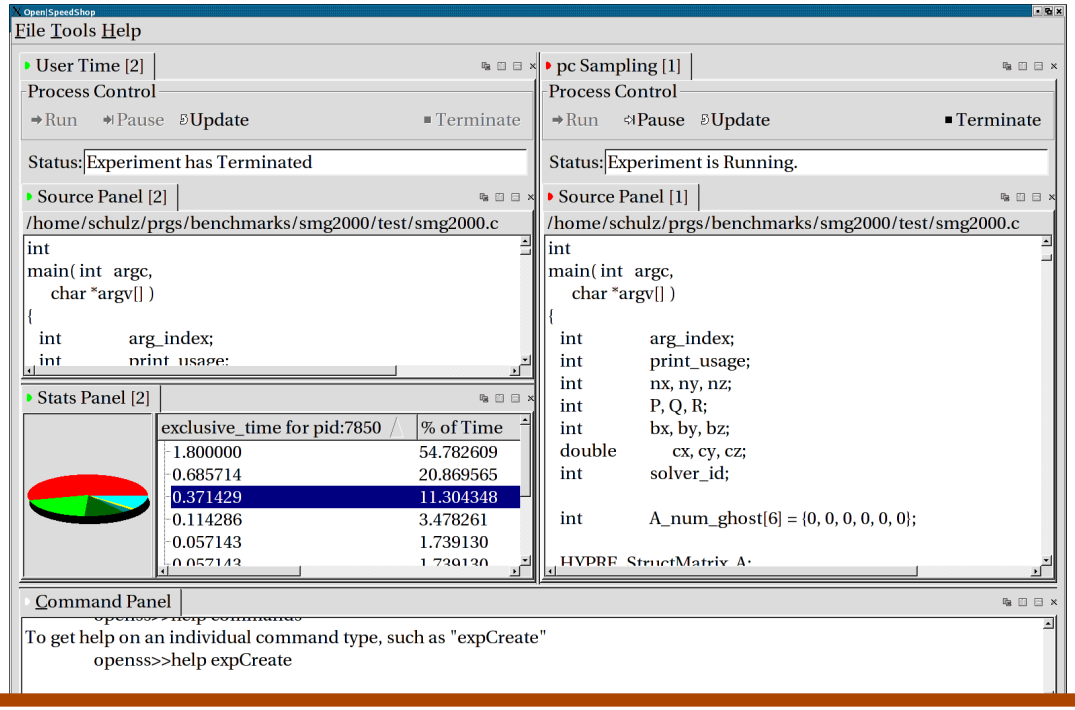


Image by John Mellor-Crummey

Open|SpeedShop

- Open Source Performance Analysis Tool Framework
  - Most common performance analysis steps **all in one tool**
  - Combines **sampling** and **tracing** techniques
  - **Extensible** by plugins for data collection and representation
  - Gathers and displays several types of performance data
- Flexible and Easy to use
  - User access through **GUI, Command Line, Python Scripting, and convenience scripts.**
- Several Instrumentation Options
  - All work on **unmodified application binaries**
  - **Offline** and **online data collection / attach** to running codes
- Supports a wide range of systems
  - Extensively used and tested on a variety of **Linux clusters**
  - New: **Cray XT/XE** and **Blue Gene/P** support

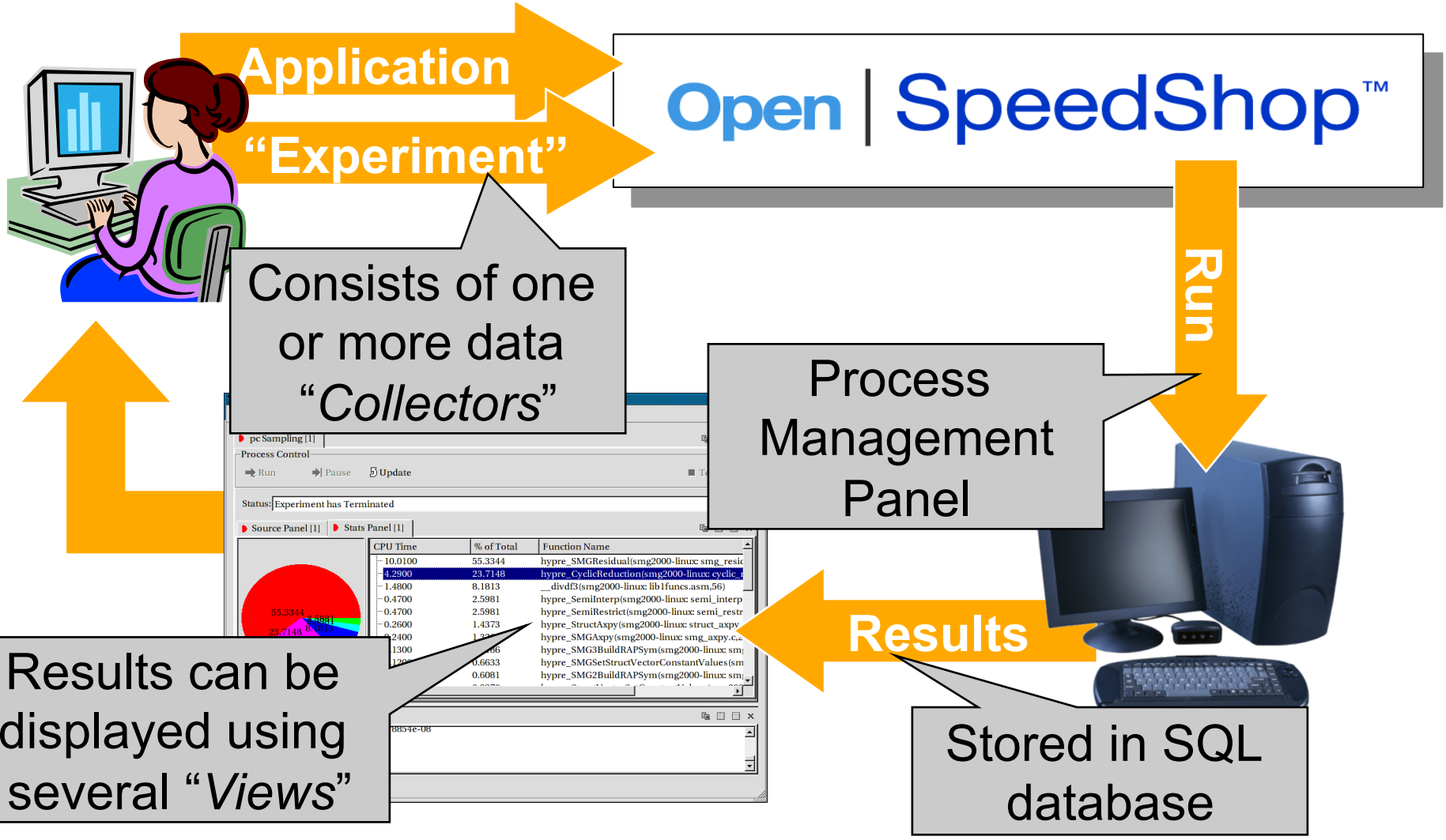
Multiple Interfaces for Additional Flexibility




```
Experiment Commands
    expAttach
    expCreate
    expDetach
    expGo
    expView

List Commands
    list -v exp
    list -v hosts
    list -v src
```

```
import openss

my_filename=openss.FileList("myprog.a.out")
my_exptype=openss.ExpTypeList("pcsamp")
my_id=openss.expCreate(my_filename,my_exptype)

openss.expGo()

My_metric_list = openss.MetricList("exclusive")
my_viewtype = openss.ViewTypeList("pcsamp”)
result = openss.expView(my_id,my_viewtype,my_metric_list)
```

Experiment Workflow



Performance Experiments

- Concept of an Experiment
  - What to measure and what to analyze?
  - Experiment type (data type to gather) is chosen by user
  - Any experiment can be applied to any application
- Variety of options
  - Sampling experiments
    - Time and HW counters, HW counter sampling
  - Tracing experiments
    - MPI, I/O, FPE
- All experiments can be applied in parallel scenarios
  - Support for MPI and threads
  - By default: all data summarized across processes/threads
  - Per process/thread display & comparisons

Making it Easy to Run on O|SS Experiments

1. Picking the experiment
   - What do I want to measure?
   - *We will start with pcsamp to get a first overview*
2. Launching the application
   - How do I control my application under O|SS?
   - *Enclose how you normally run your application in quotes*
   - ***osspcsamp*** *"mpirun –np 256 smg2000 –n 65 65 65"*
3. Storing the results
   - *O|SS will create a database*
   - *Name: smg2000-pcsamp.openss*
4. Exploring the gathered data
   - How do I interpret the data?
   - *O|SS will print a default report*
   - *Open the GUI to analyze data in detail (run: "**openss**")*

Example Run with Output

```
osspcsamp "mpirun -np 2 ./smg2000 -n 65 65 65"
[openss]: pcsamp experiment using the pcsamp experiment default sampling rate: "100".
[openss]: Using OPENSS_PREFIX installed in /opt/OSS-mrnet
[openss]: Setting up offline raw data directory in /tmp/jeg/offline-oss
[openss]: Running offline pcsamp experiment using the command:
"mpirun -np 2 /opt/OSS-mrnet/bin/ossrun "./smg2000 -n 65 65 65" pcsamp"

Running with these driver parameters:
 (nx, ny, nz)    = (65, 65, 65)
 (Px, Py, Pz)    = (2, 1, 1)
 (bx, by, bz)    = (1, 1, 1)
 (cx, cy, cz)    = (1.000000, 1.000000, 1.000000)
 (n_pre, n_post) = (1, 1)
 dim             = 3
 solver ID       = 0
=============================================
Struct Interface:
=============================================
Struct Interface:
 wall clock time = 0.049847 seconds
 cpu clock time  = 0.050000 seconds
```

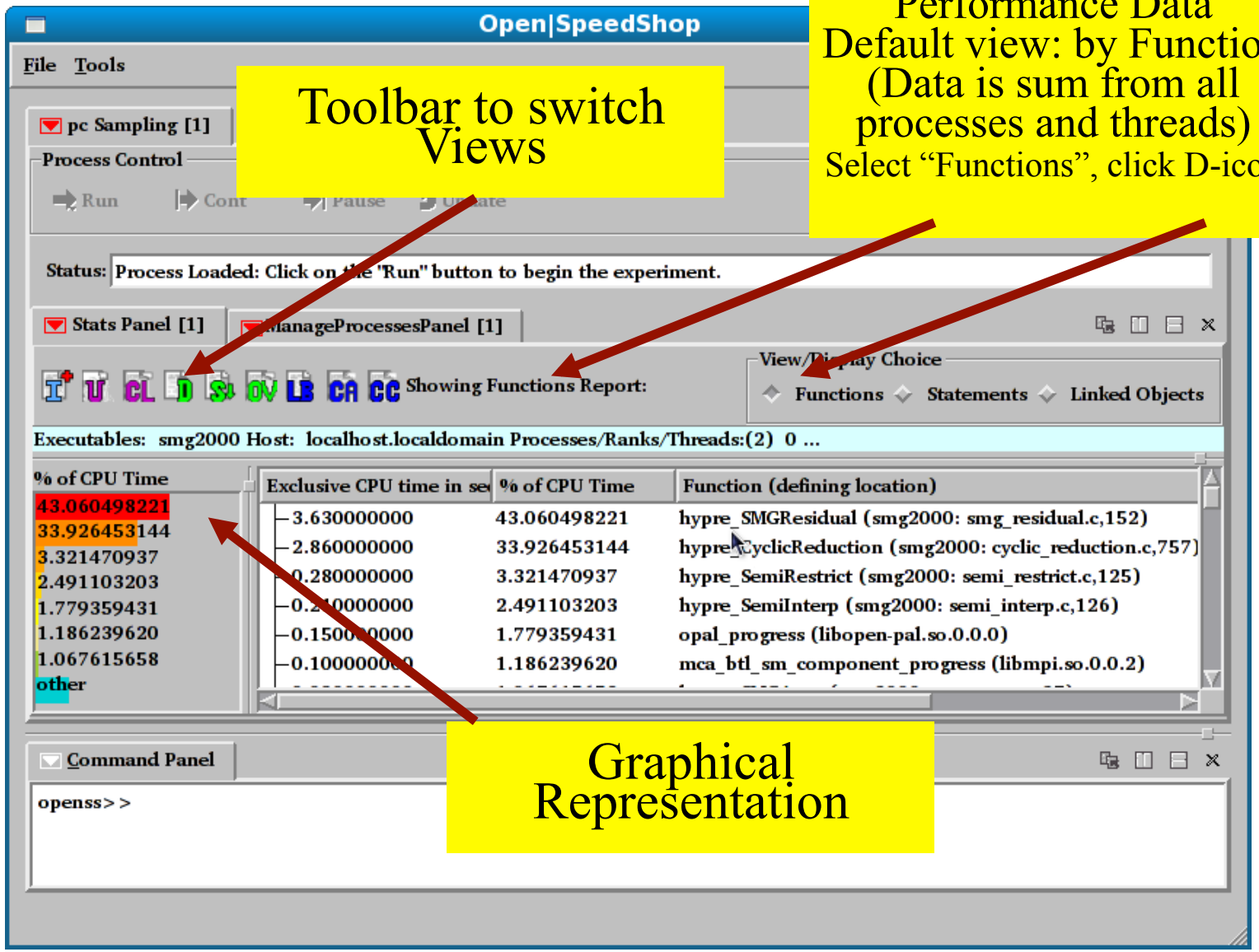
Example Run with Output

```
=============================================
Setup phase times:
=============================================
SMG Setup:
 wall clock time = 0.635208 seconds
 cpu clock time  = 0.630000 seconds
=============================================
Solve phase times:
=============================================
SMG Solve:
 wall clock time = 3.987212 seconds
 cpu clock time  = 3.970000 seconds
Iterations = 7
Final Relative Residual Norm = 1.774415e-07
[openss]: Converting raw data from /tmp/jeg/offline-oss into temp file X.0.openss

Processing raw data for smg2000
Processing processes and threads ...
Processing performance data ...
Processing functions and statements ...
```

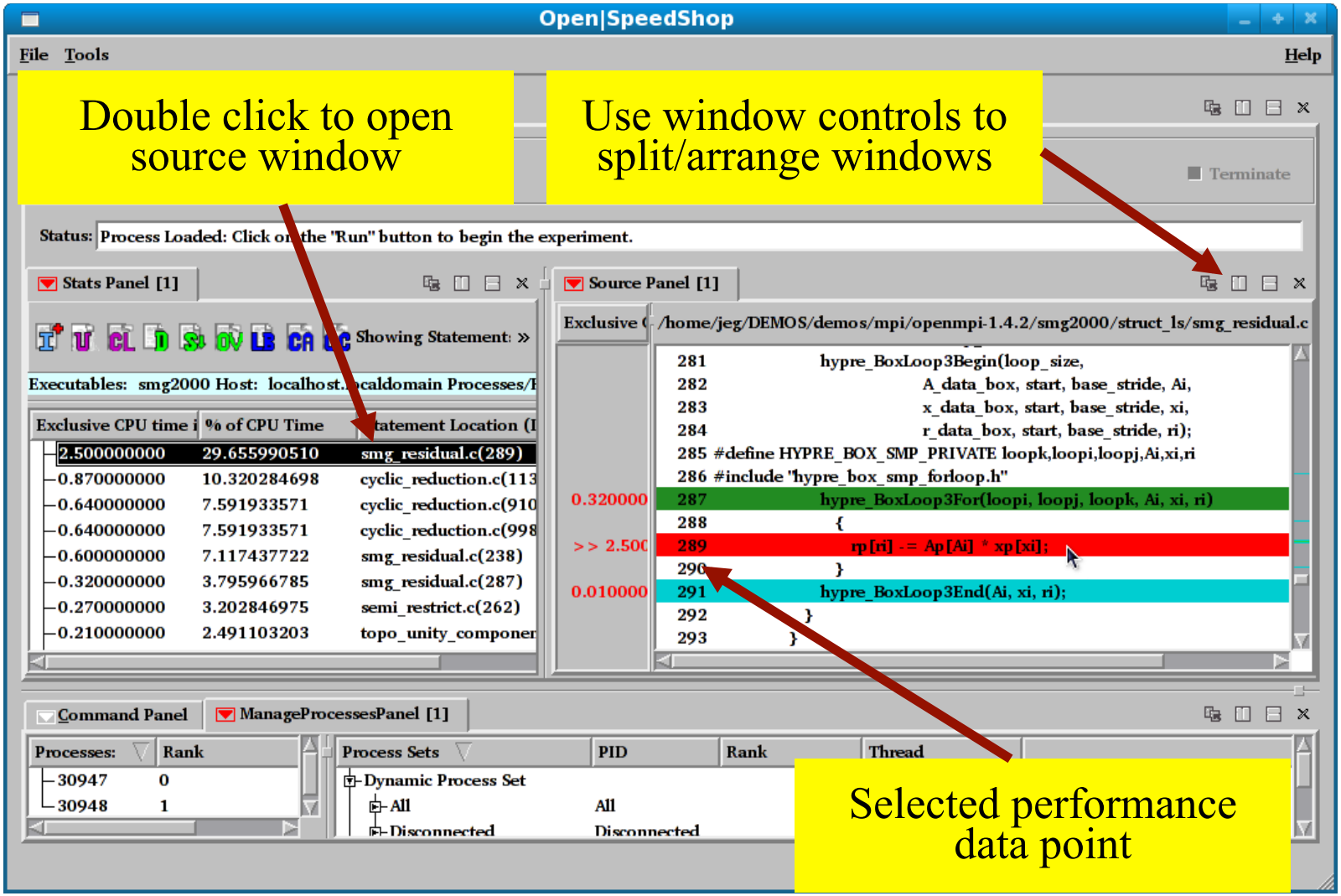
Example Run with Output

```
[openss]: Restoring and displaying default view for:
        /home/jeg/DEMOS/demos/mpi/openmpi-1.4.2/smg2000/test/smg2000-
pcsamp-1.openss
[openss]: The restored experiment identifier is:  -x 1

 Exclusive CPU time      % of CPU Time  Function (defining location)
      in seconds.
      3.630000000        43.060498221  hypre_SMGResidual (smg2000: smg_residual.c,152)
      2.860000000        33.926453144  hypre_CyclicReduction (smg2000: cyclic_reduction.c,
757)
      0.280000000         3.321470937  hypre_SemiRestrict (smg2000: semi_restrict.c,125)
      0.210000000         2.491103203  hypre_SemiInterp (smg2000: semi_interp.c,126)
      0.150000000         1.779359431  opal_progress (libopen-pal.so.0.0.0)
      0.100000000         1.186239620 mca_btl_sm_component_progress (libmpi.so.0.0.2)
      0.090000000         1.067615658  hypre_SMGAxpy (smg2000: smg_axpy.c,27)
      0.080000000         0.948991696  ompi_generic_simple_pack (libmpi.so.0.0.2)
      0.070000000         0.830367734  __GI_memcpy (libc-2.10.2.so)
      0.070000000         0.830367734 hypre_StructVectorSetConstantValues (smg2000:
struct_vector.c,537)
      0.060000000         0.711743772  hypre_SMG3BuildRAPSym (smg2000:
smg3_setup_rap.c,233)
```

Default Output Report View



Associate Source & Performance Data



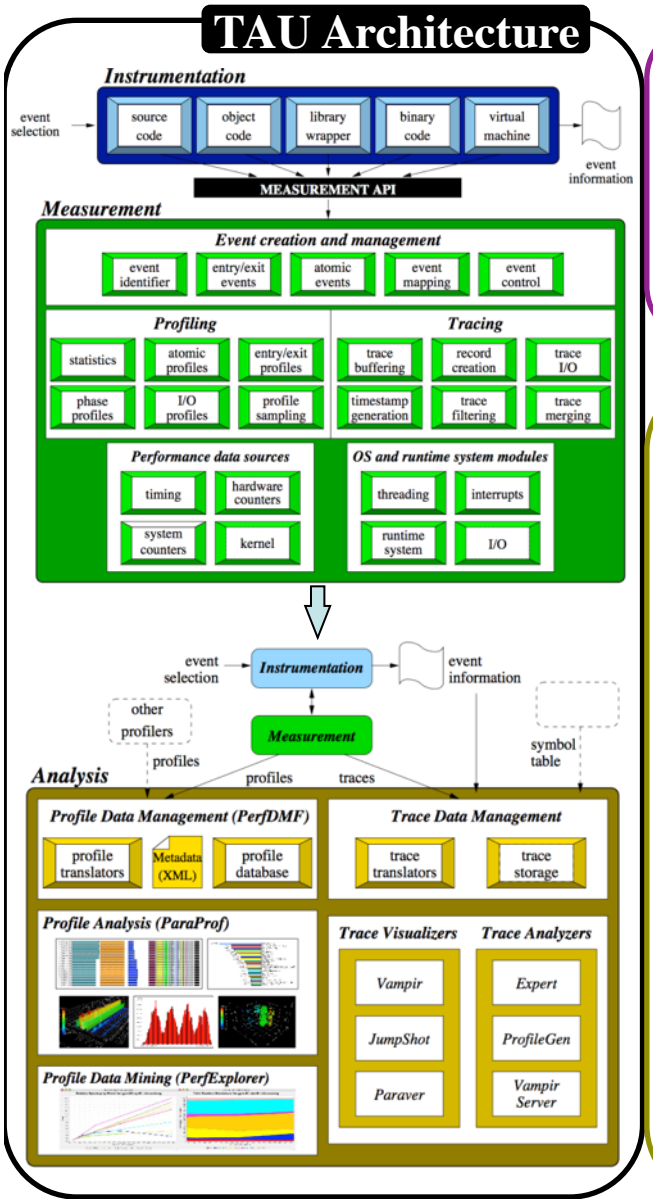
Digging Deeper with the "Swiss Army Knife" of Performance Analysis : TAU

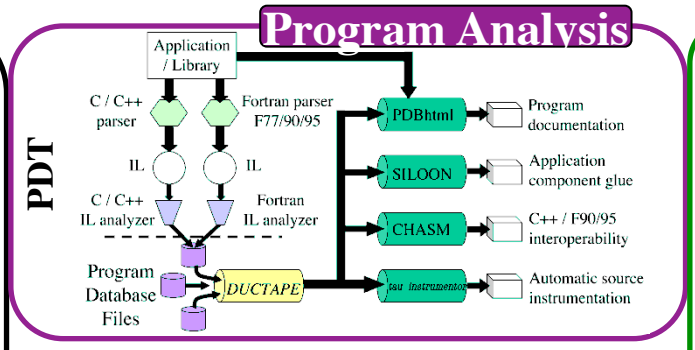
- **Very portable tool set** for instrumentation, measurement and analysis of parallel multi-threaded applications
- Instrumentation API supports choice
  - between **profiling and tracing**
  - of metrics (i.e., time, HW Counter (PAPI))
- Uses Program Database Toolkit (PDT) for C, C++, Fortran source code instrumentation
- Supports
  - Languages: C, **C++**, Fortran 77/90, HPF, HPC++, **Java**, **Python**
  - Threads: **pthreads**, Tulip, SMARTS, **Java, Win32, OpenMP**
  - Systems: UNIX/Linux + Windows + MacOS + …

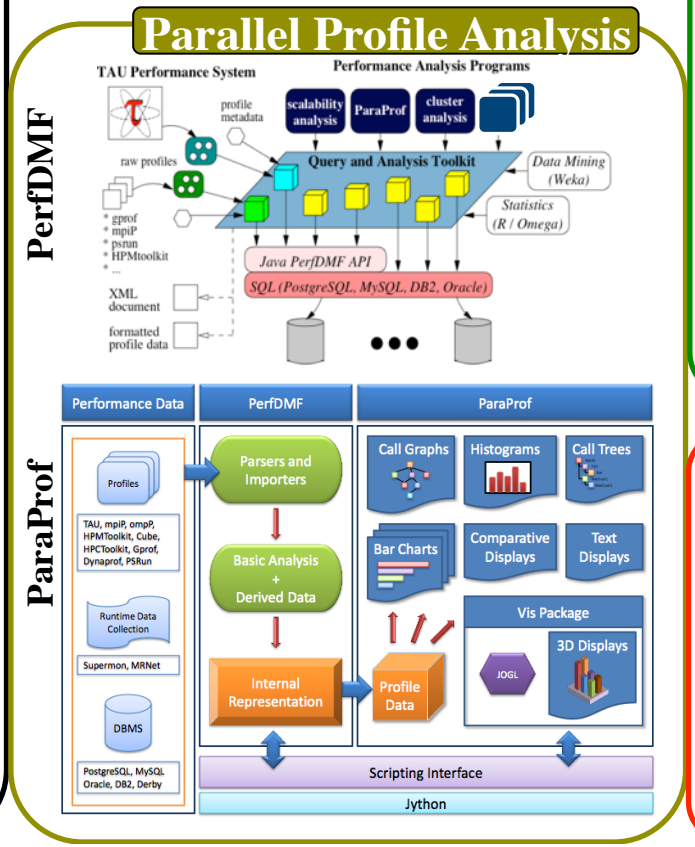
- http://tau.uoregon.edu/

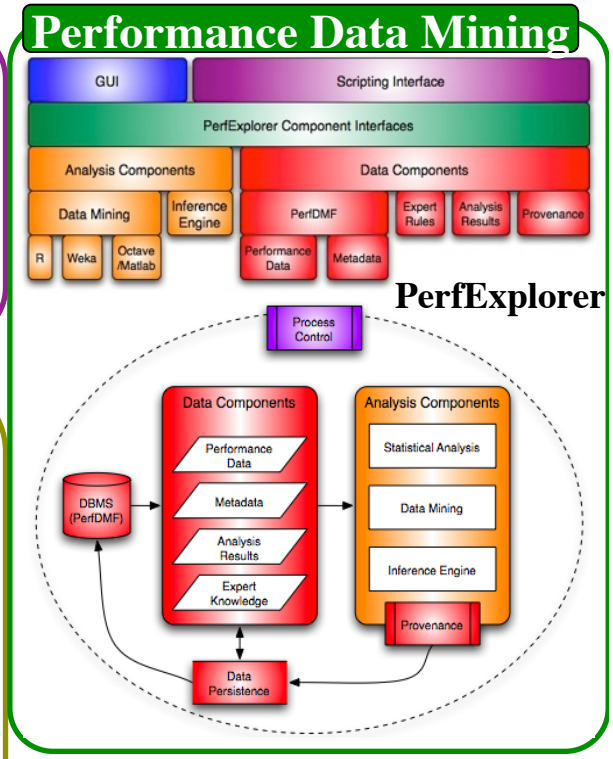
Tuning and Analysis Utilities

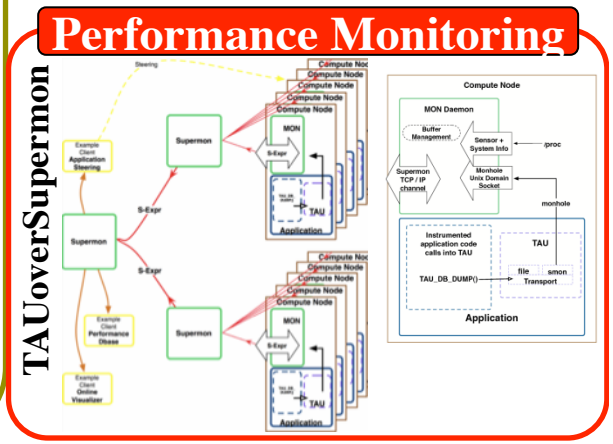
TAU Performance System Components



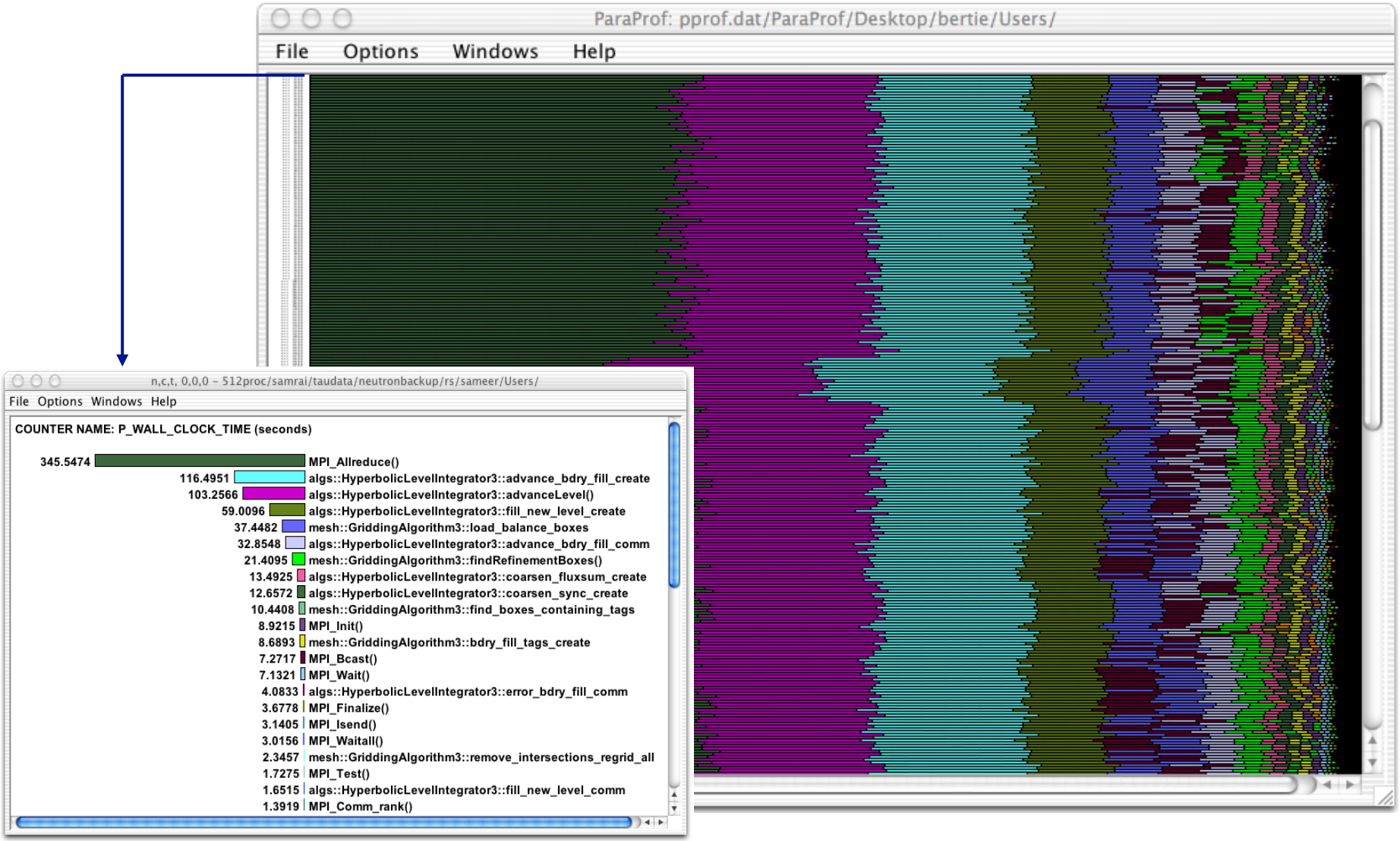




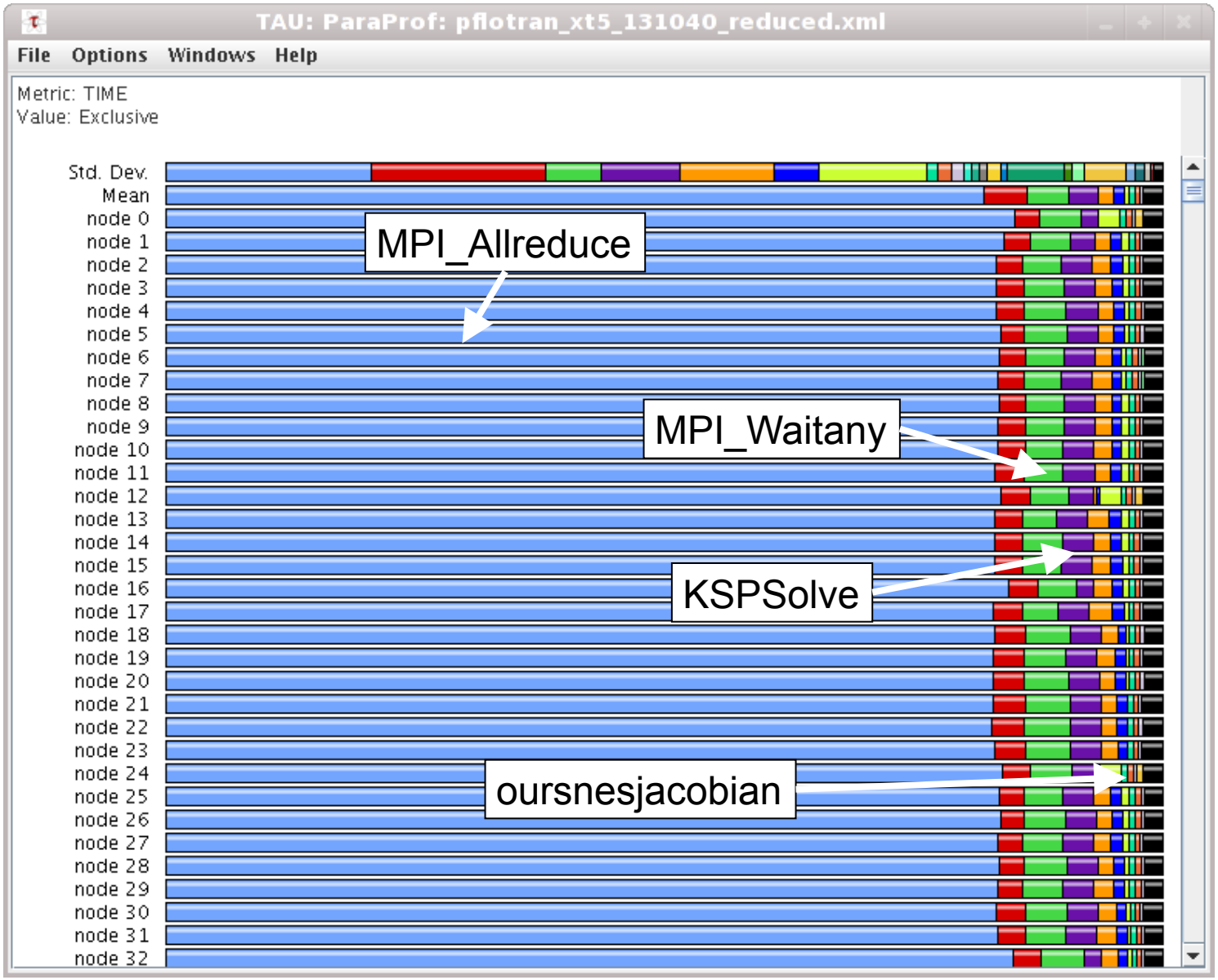




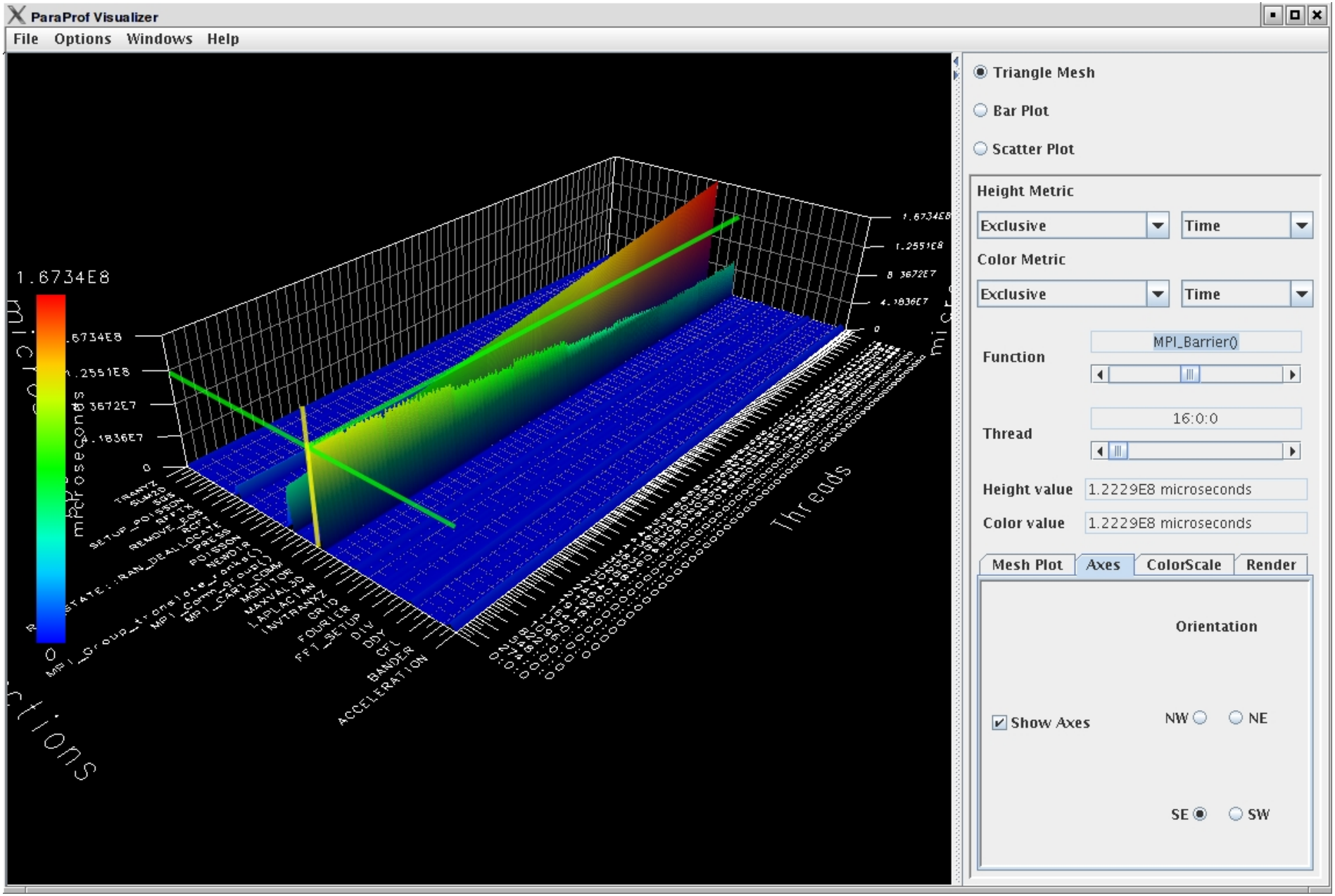
TAU Profiling, Large System



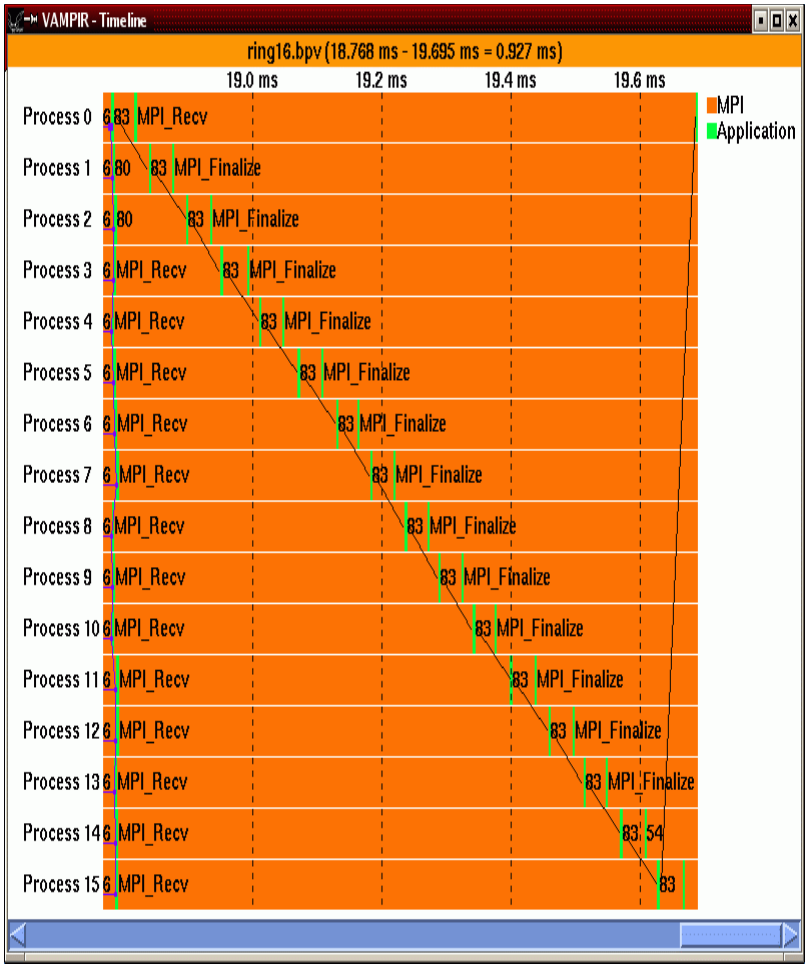
TAU Full Profile (Exclusive, 131K cores)

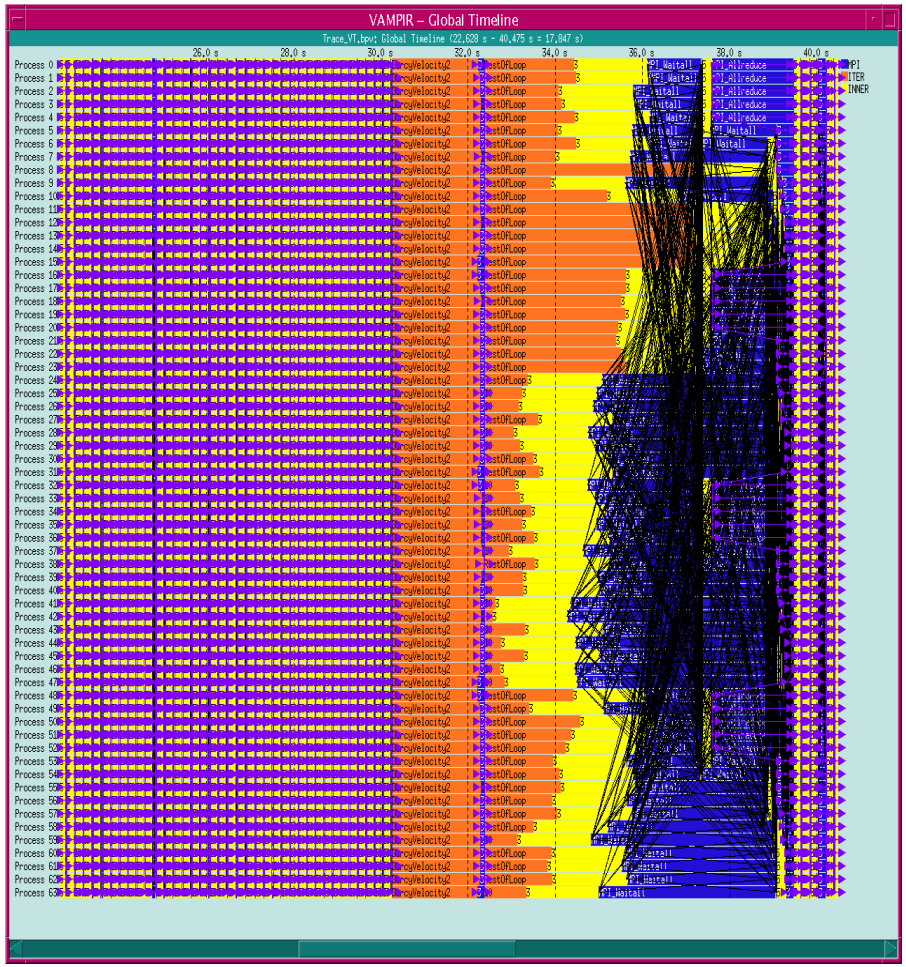


TAU ParaProf: 3D Profile, Miranda, 16K PEs



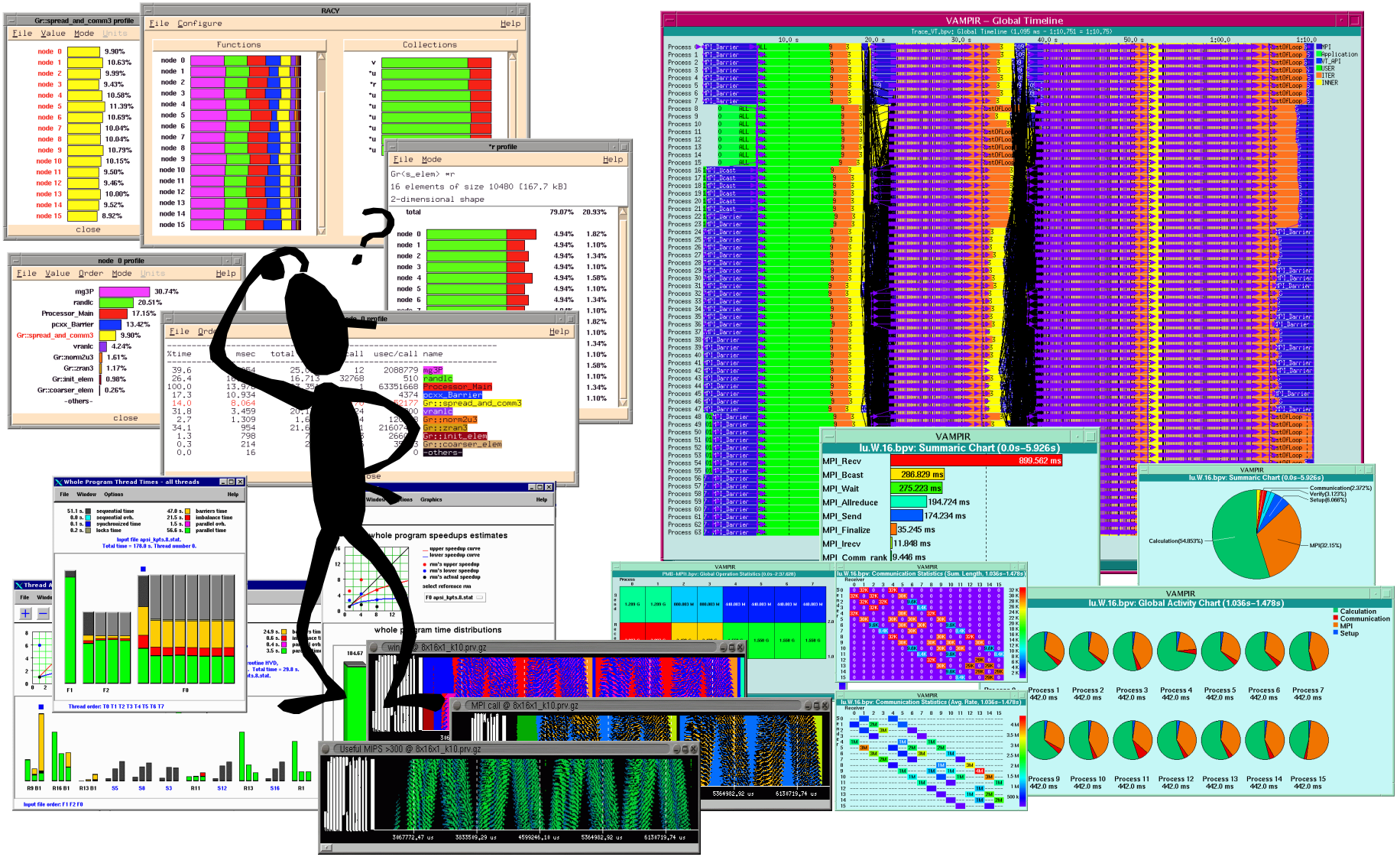
“A picture is worth 1000 words…”



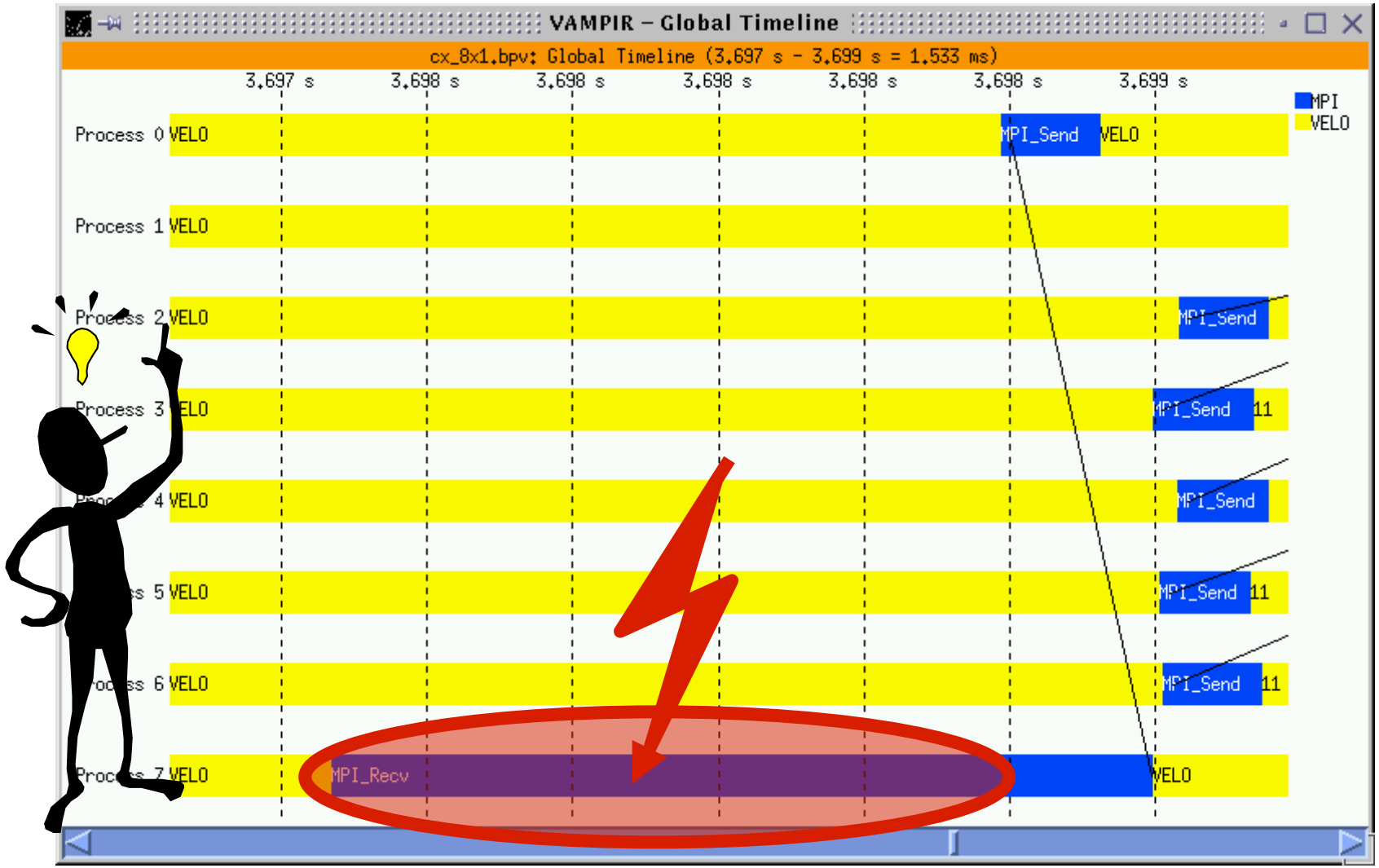


- MPI ring program
- “Real world” example

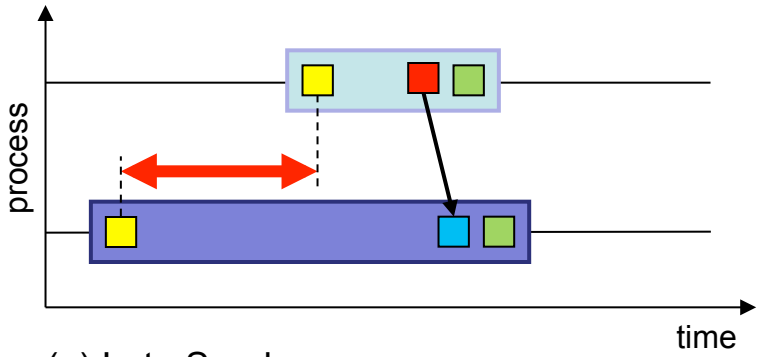
“What about 1000’s of pictures?” (with 100’s of menu options)



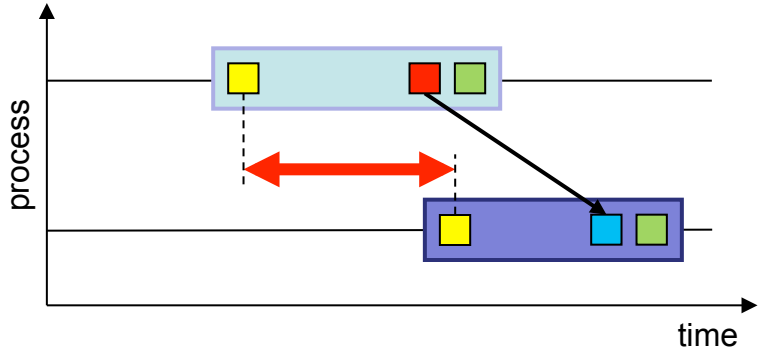
Example Automatic Analysis: Late Sender



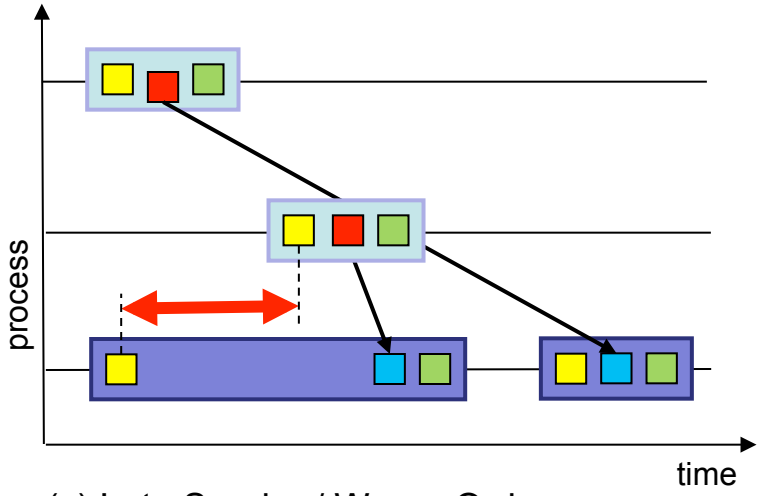
Example Patterns



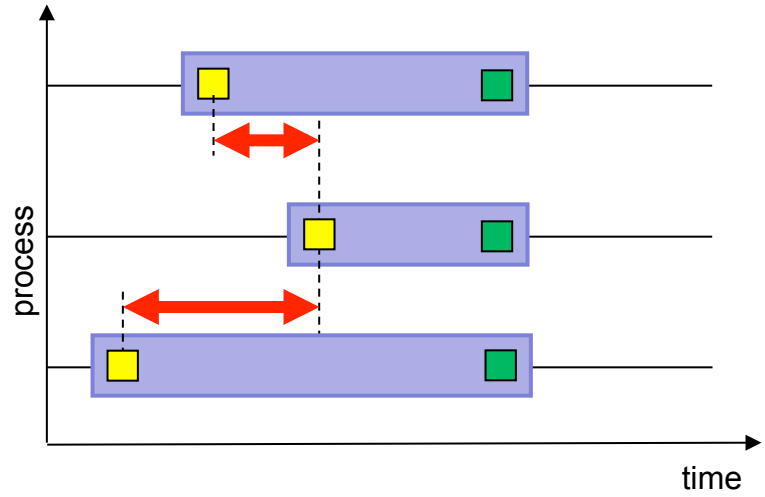
(a) Late Sender



(b) Late Receiver



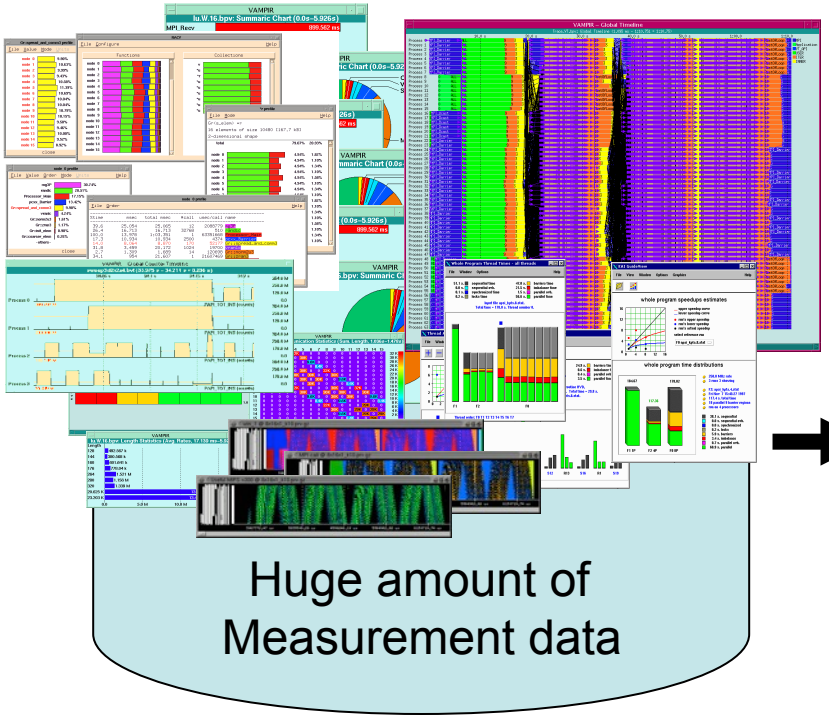
(c) Late Sender / Wrong Order



(d) Wait at N x N

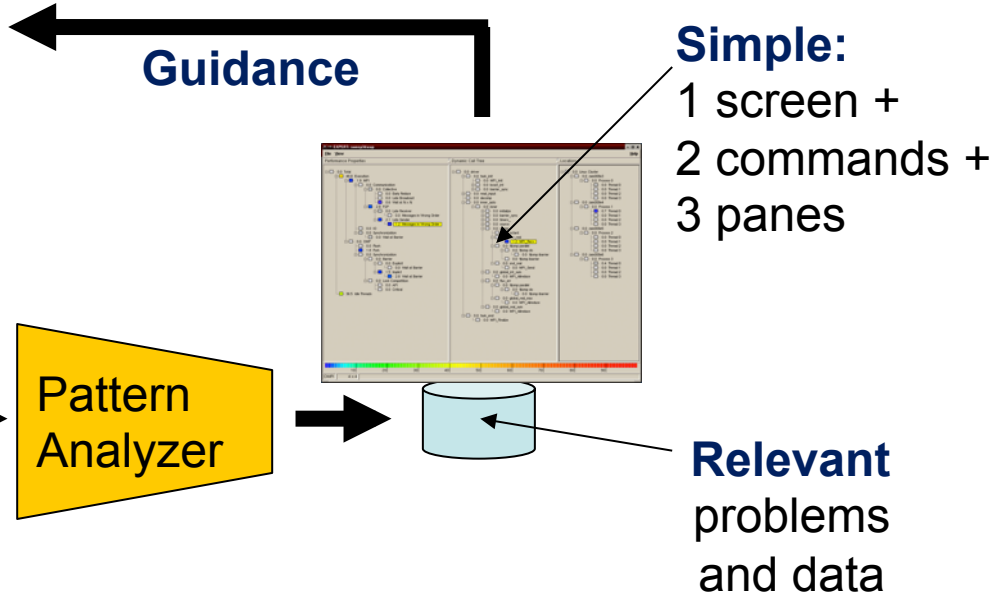
Basic Idea Automatic Performance Analysis

- "Traditional" Tool



- For non-standard / tricky cases (10%)
- For expert users

- Automatic Tool



- For standard cases (90% ?!)
- For "normal" users
- Starting point for experts

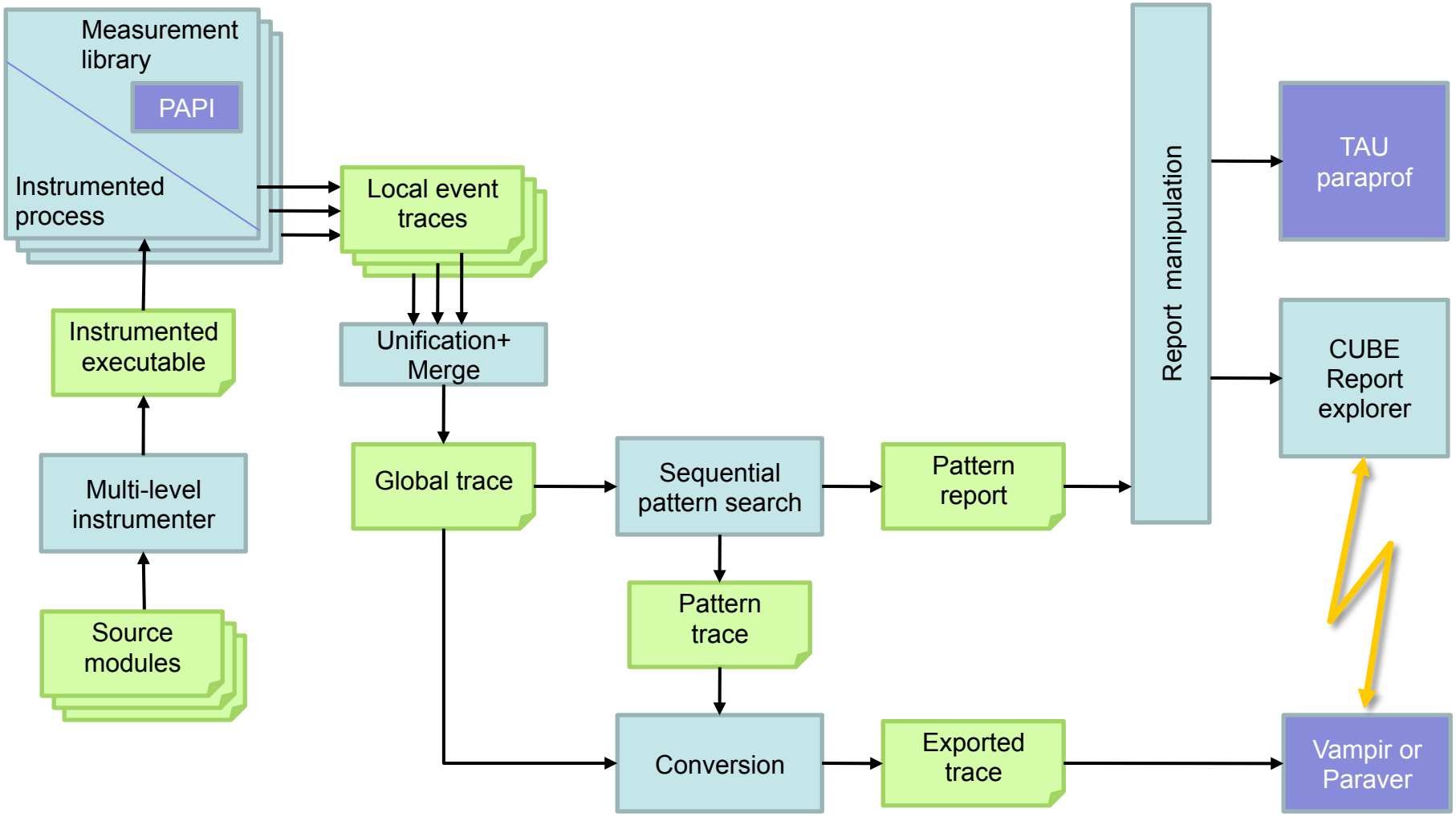
⇨ **More productivity for performance analysis process!**

The KOJAK Project

- Kit for Objective Judgement and Automatic Knowledge-based detection of bottlenecks
- Forschungszentrum Jülich
- Innovative Computing Laboratory, TN
- Started 1998

- Approach
  - **Instrument** C, C++, and Fortran parallel applications
    - Based on MPI, OpenMP, SHMEM, or hybrid
  - **Collect** event traces
  - **Search** trace for event patterns representing inefficiencies
  - **Categorize and rank** inefficiencies found
- http://www2.fz-juelich.de/jsc/kojak/

Sequential Analysis Process



= Third-party component

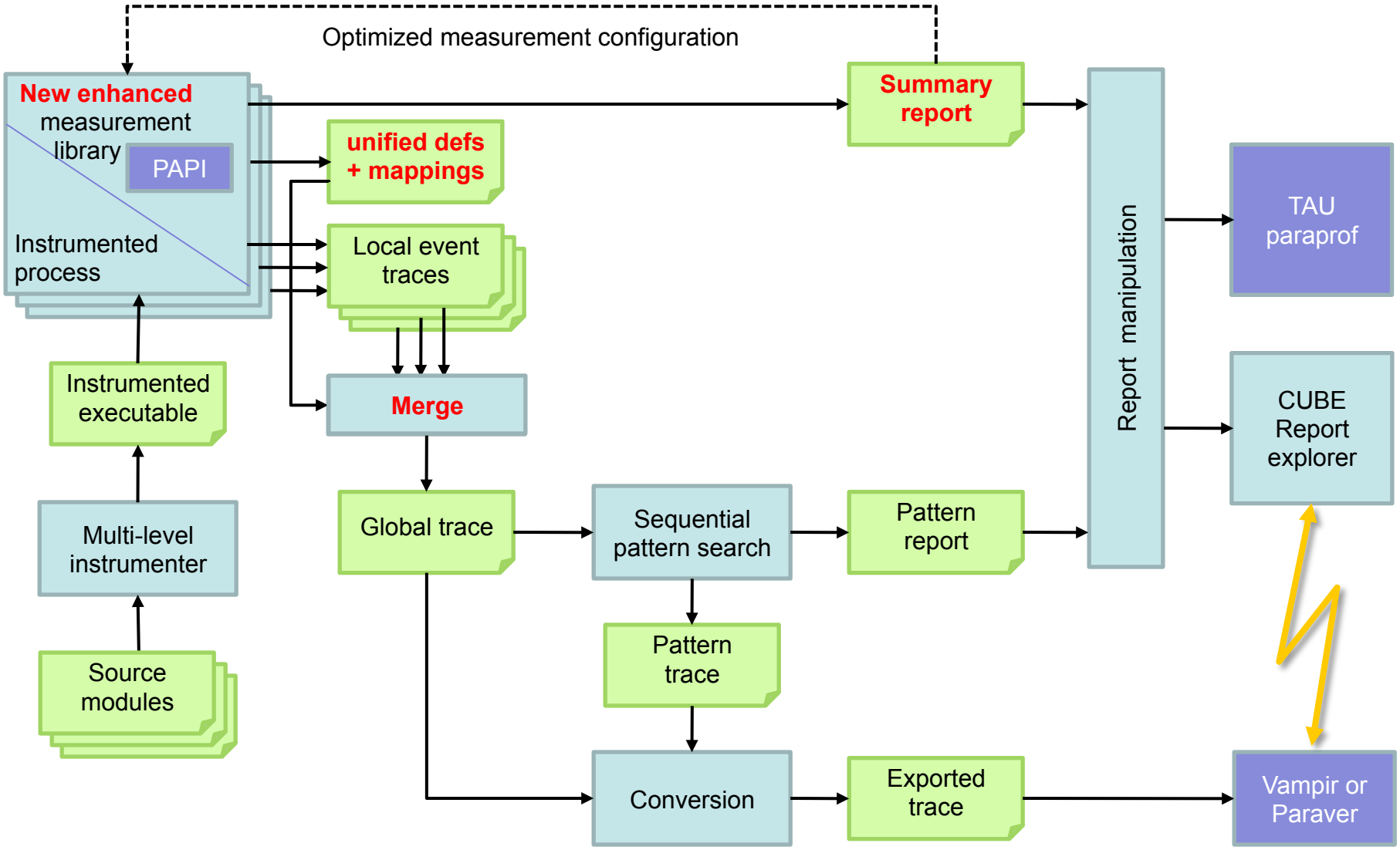
The Scalasca Project

- Scalable Analysis of Large Scale Applications
- Follow-up project to KOJAK
- Started in January 2006

- **Objective 1:** do not rely on tracing only
  - ⇨ Supports scalable **call-path profiling**
- Objective 2: develop a scalable version of KOJAK
  - ⇨ Basic idea: **parallelization of trace analysis**

- Supports MPI 2.2 (P2P, collectives, RMA, IO) and basic OpenMP (no nesting)
- http://www.scalasca.org/

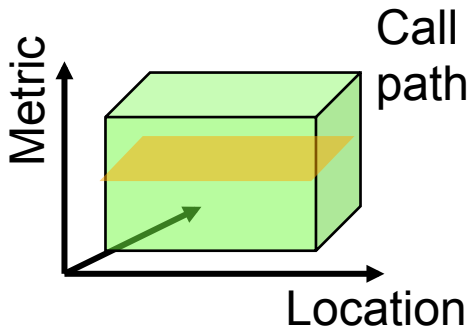
New Analysis Process I



= Third-party component

CUBE Result Browser

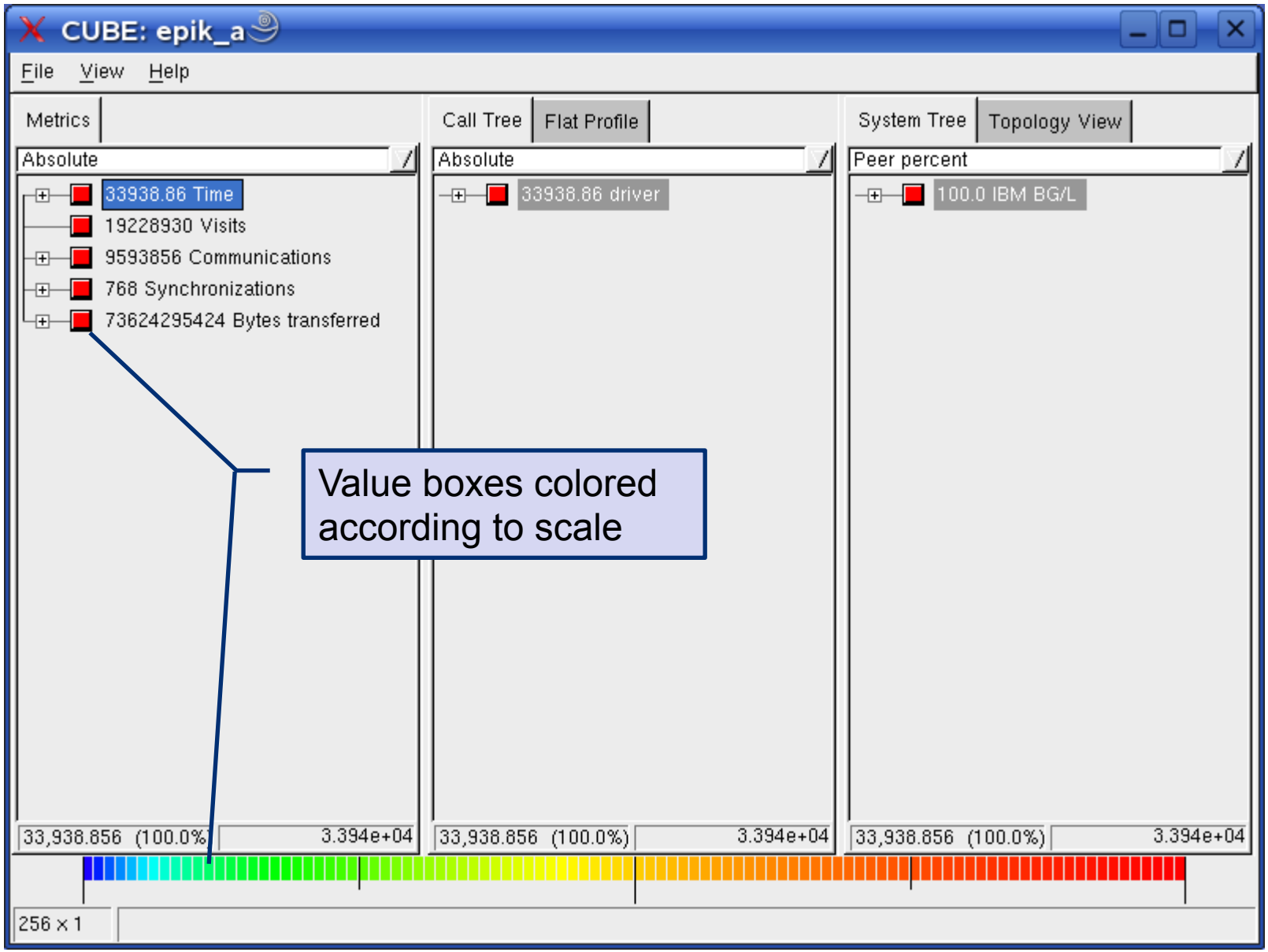
- Representation of results (3D severity matrix) along three hierarchical axes
  - Metric
  - Call tree path
  - System location



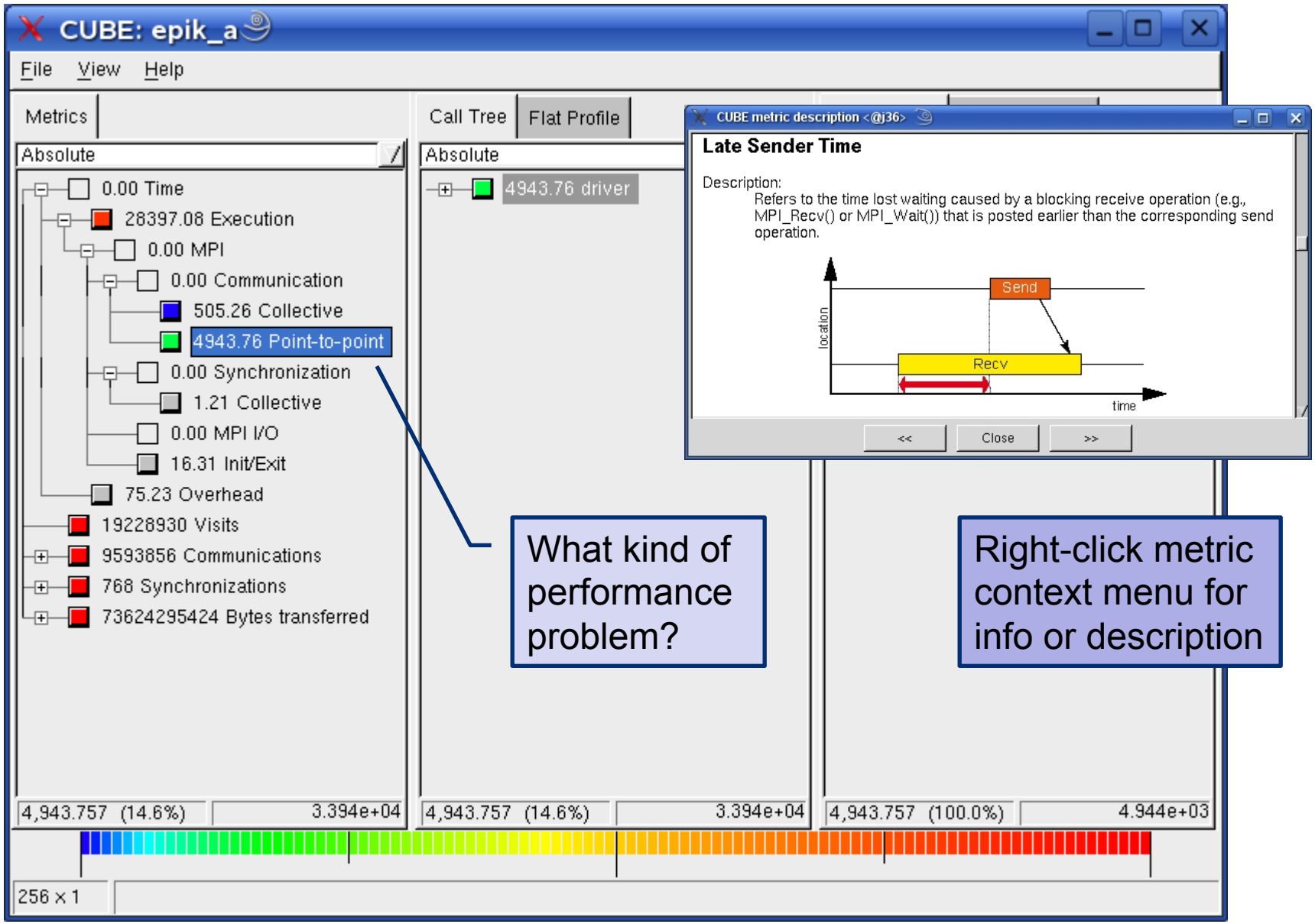
- Three coupled tree browsers

- Each node displays severity
  - As colour: for easy identification of bottlenecks
  - As value: for precise comparison

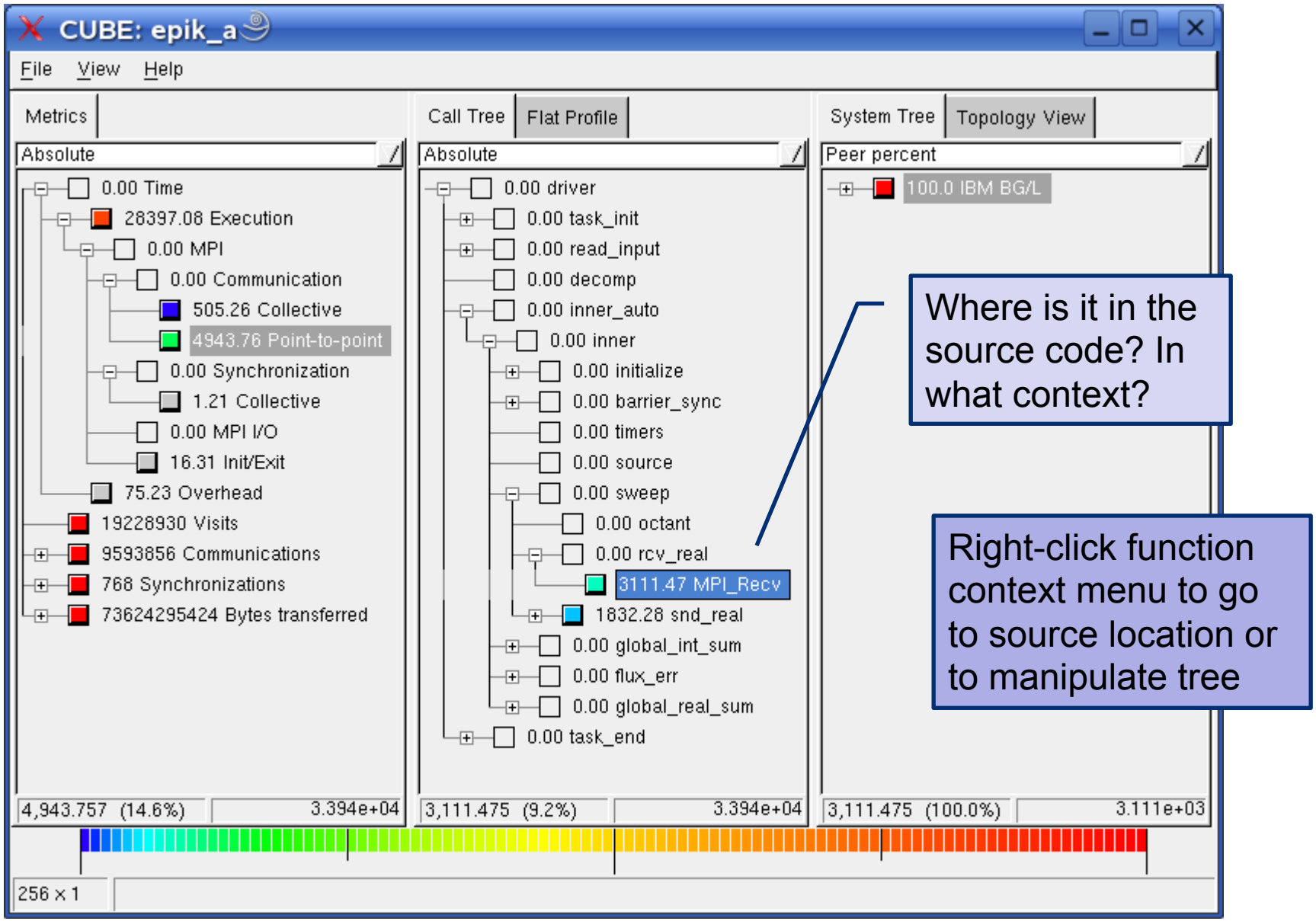
CUBE Result Browser (III)



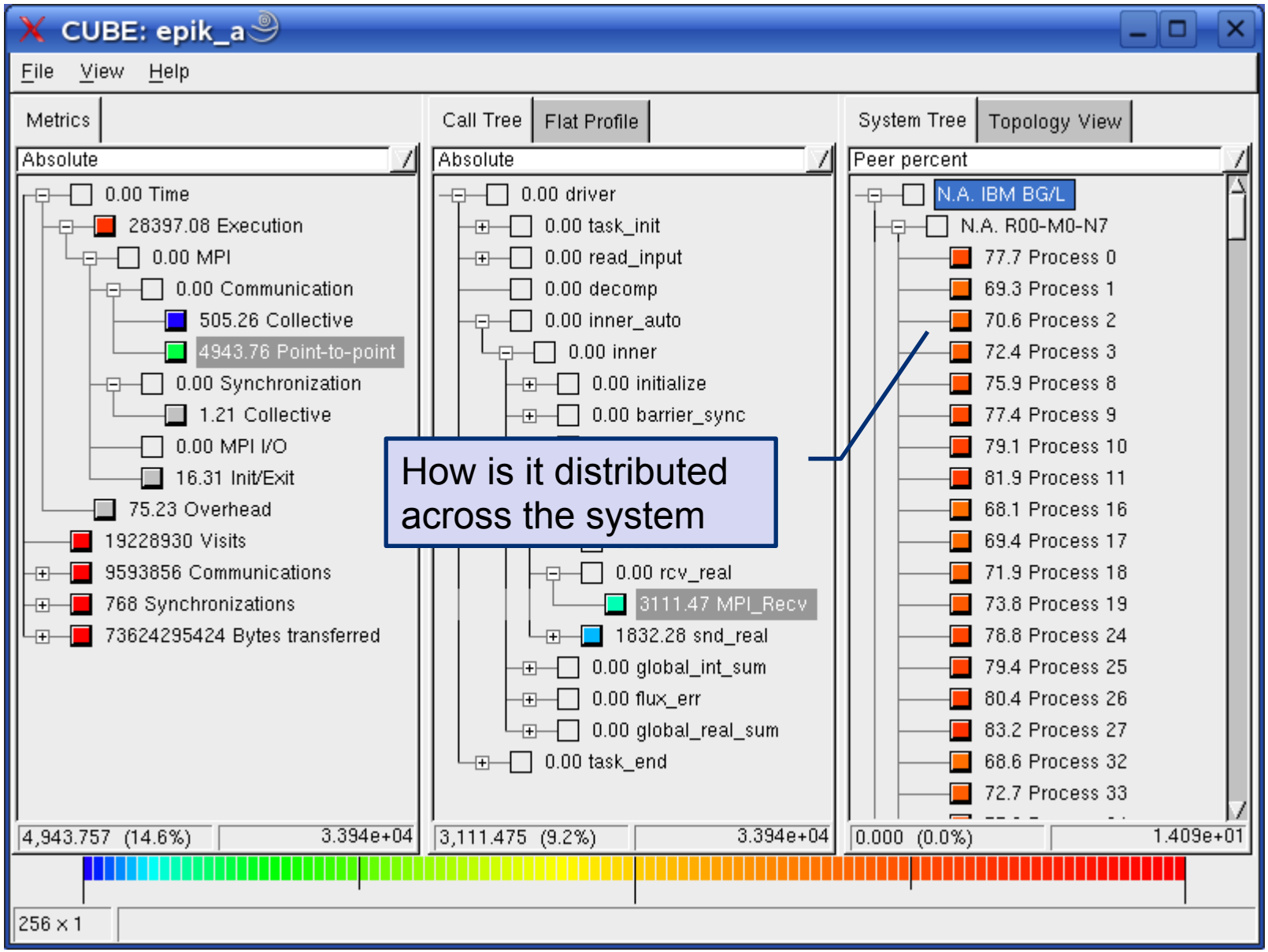
Metric Dimension



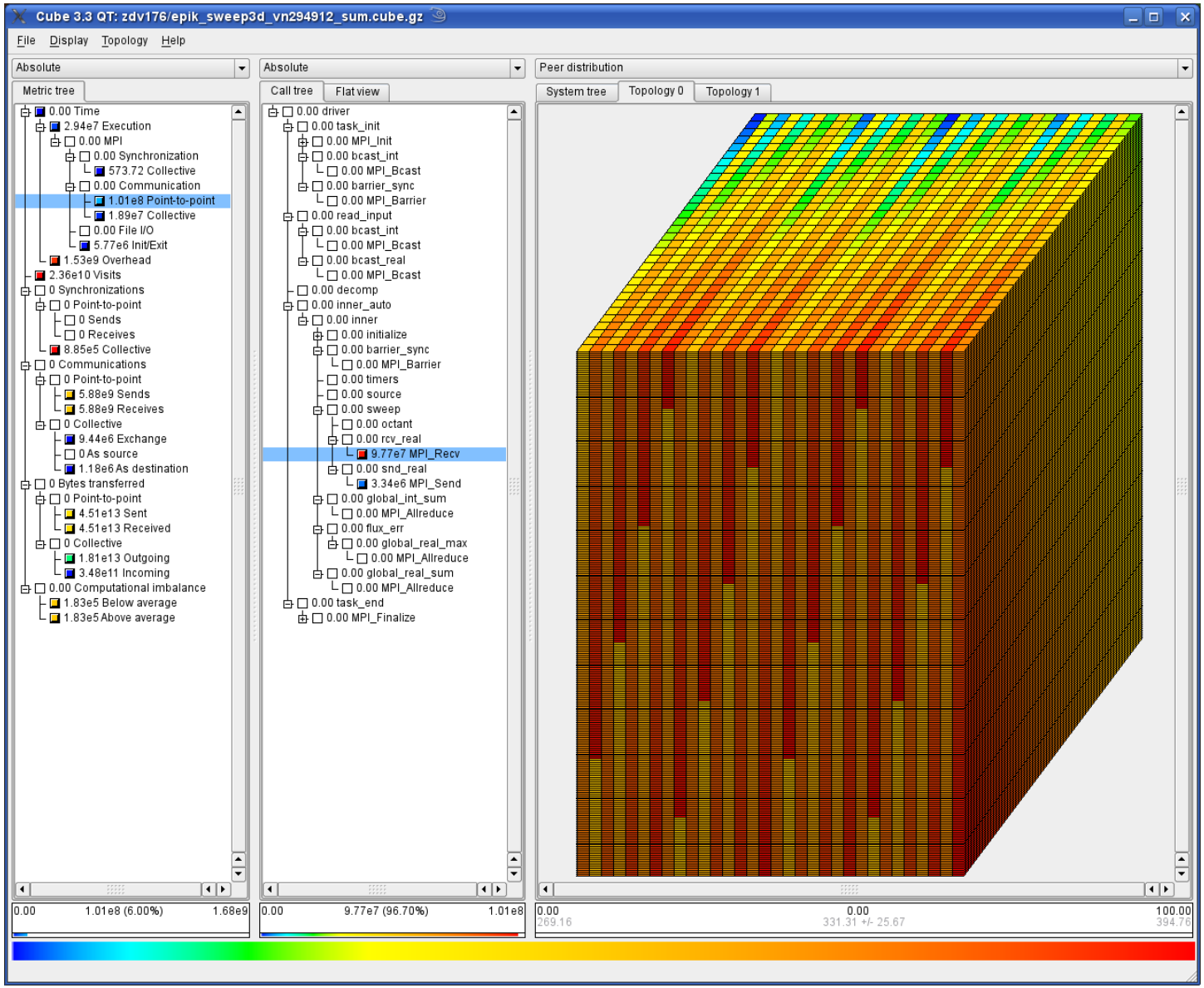
Call Tree Dimension



System Tree Dimension

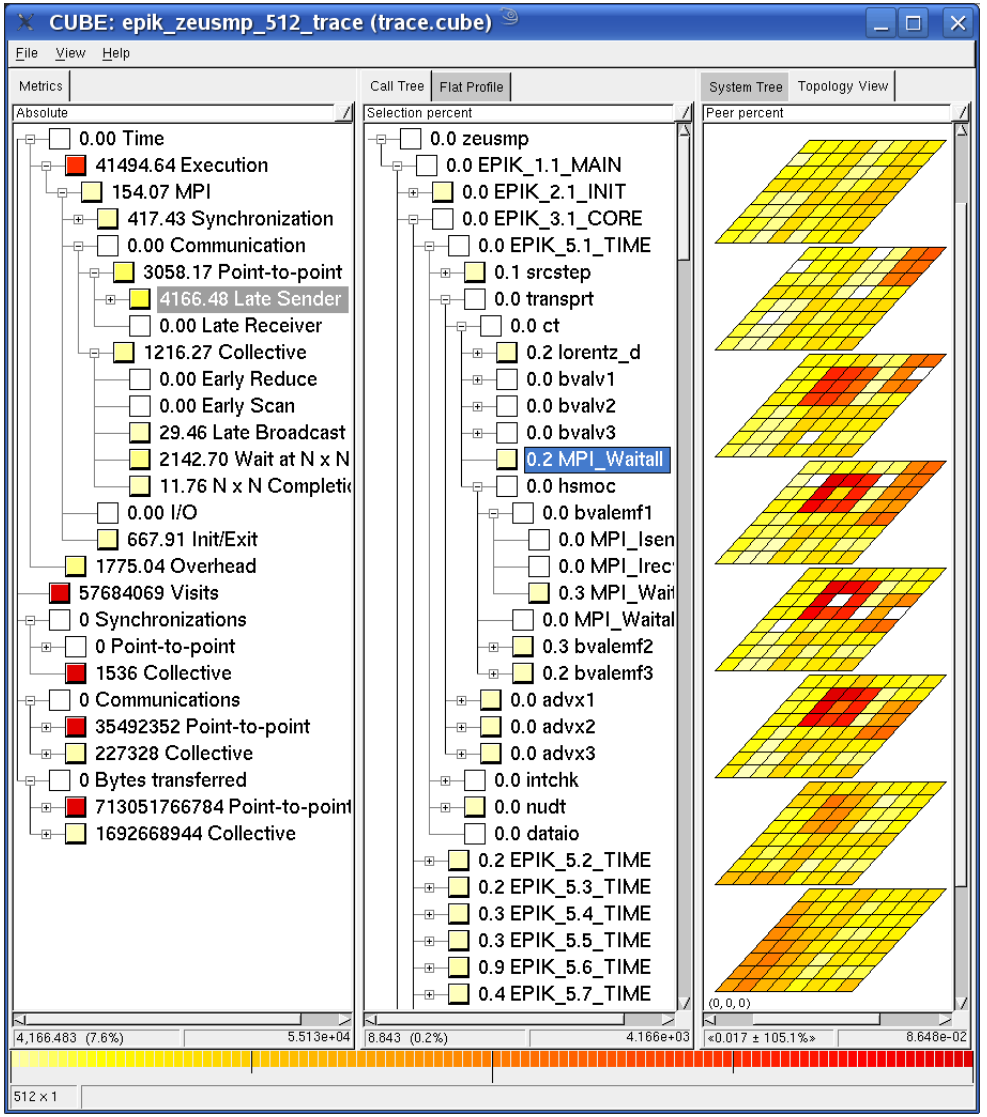


Summary analysis sweep3D@294,912



Time-series Call-path (Phase) Profiling

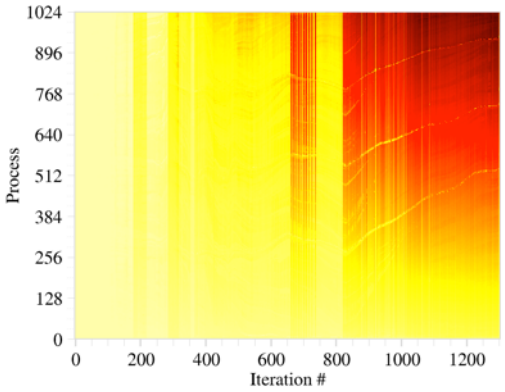
- Manual instrumentation to distinguish iterations of the main loop
- Complete **call-tree recorded for each iteration**
  - With multiple metrics collected for every call-path
- Huge growth in the amount of data collected
  - Reduced scalability



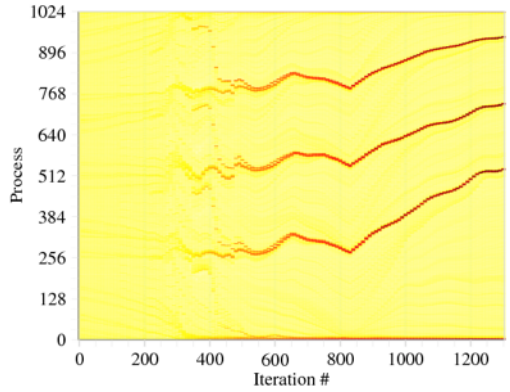
Incremental On-line Clustering

- Exploit that many iterations are very similar
  - Summarizes several iterations to their average
- On-line to save memory
- Process local to
  - Avoid communication
  - Adjust to local temporal patterns
- The number of clusters never exceeds a predefined maximum
  - Merging of the two closest ones
- Allows to identify interesting candidate iterations for in-depth analysis using tracing

PEPC n-body tree code (JSC)



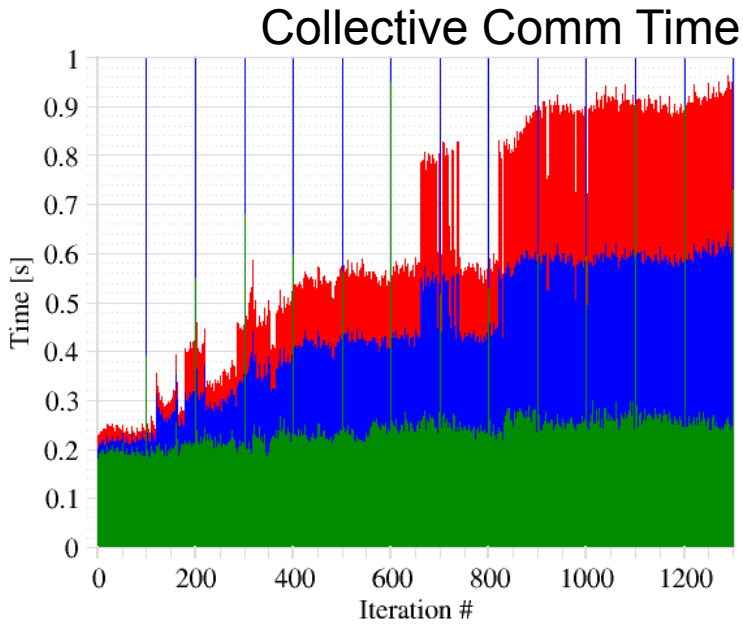
Late Sender

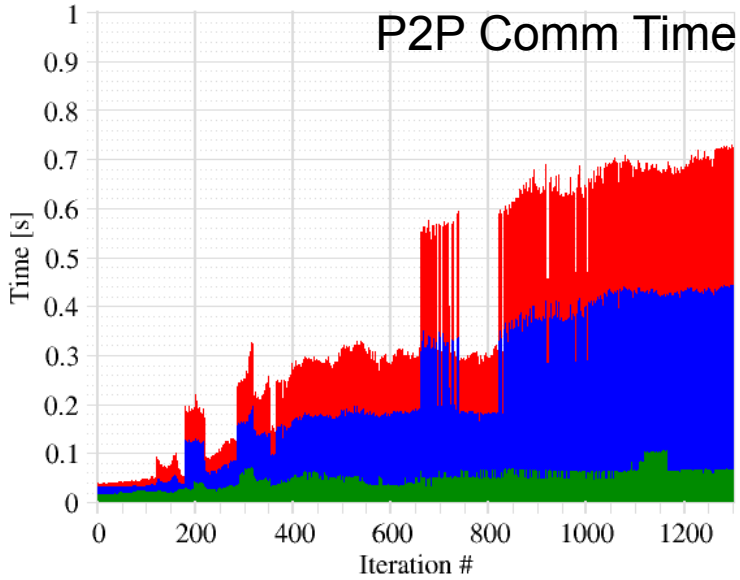


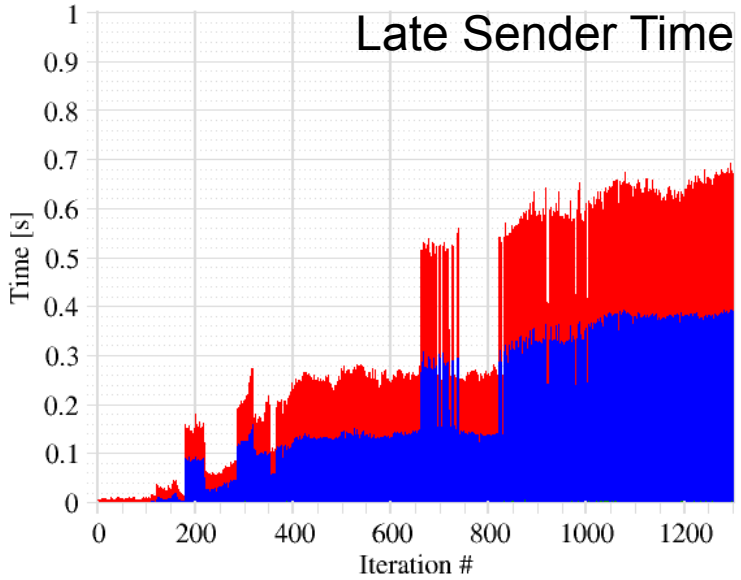
particles owned by a process

Iteration Graphs

- For each iteration:
  - *red: max process*
  - *blue: median process*
  - *green: min process*

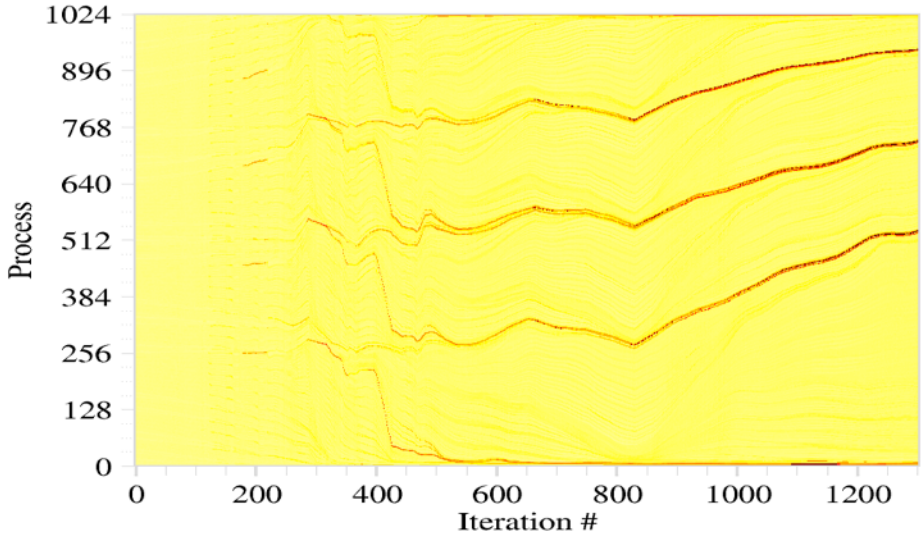




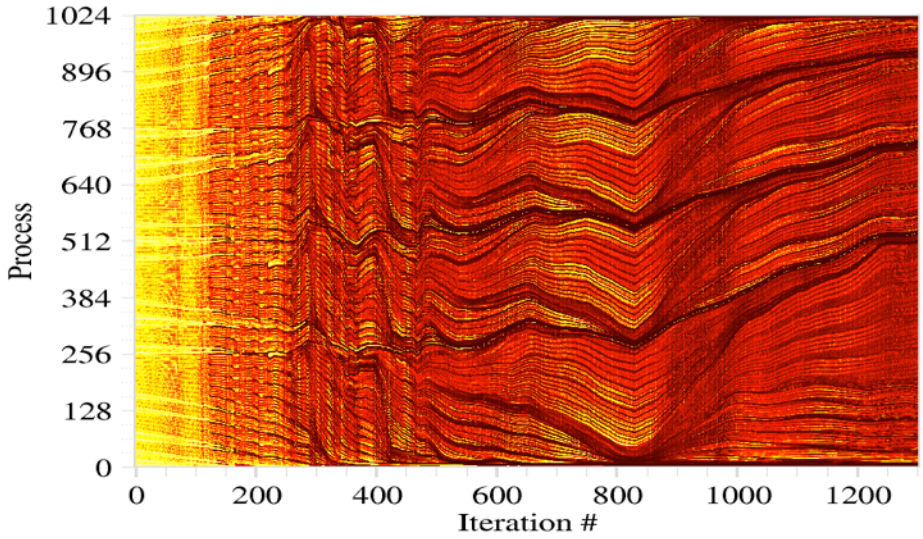


Heat Map of P2P Communication Count

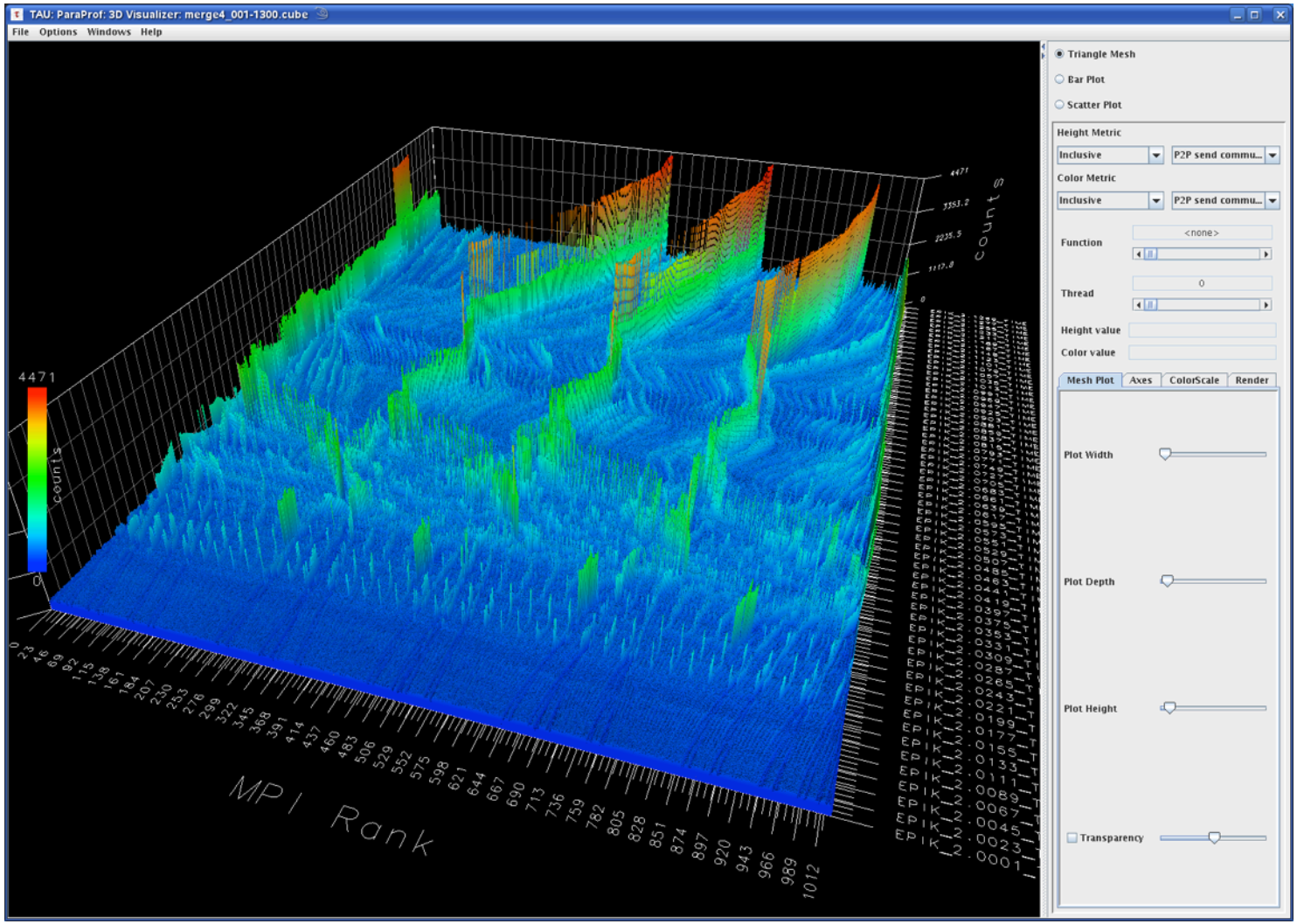
- **Linear scaling**
  - Color darkness linearly scaled between lowest and highest value



- **Histogram equalization**
  - Around equal number of pixels at each darkness level
  - Reveals the details
  - But we lose the scale information



3D Visualization



- Visualizing our data using the TAU ParaProf 3D visualizer

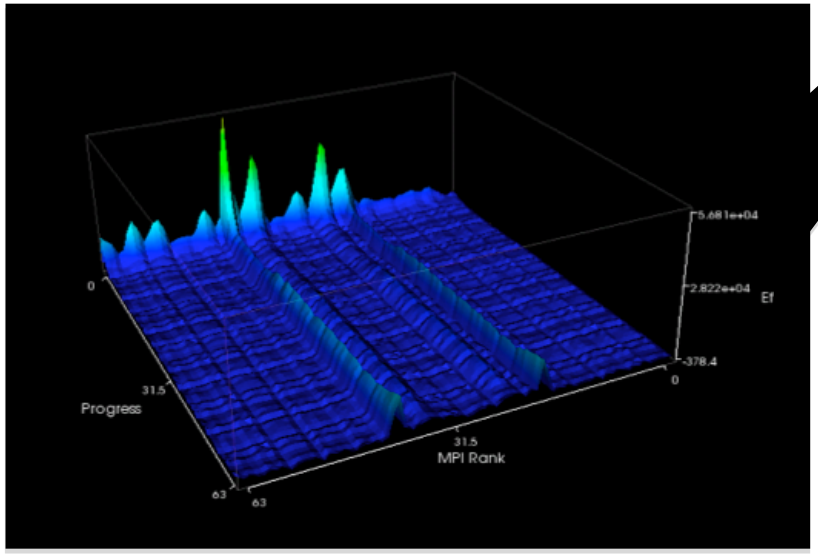
Profiling for Load Balance Analysis

- Load balance crucial for scaling
  - Small imbalances will cause large slow down
  - Impact gets larger with scale
- Requirements
  - Scalability
  - Need to understand load wrt. source code
  - Minimal impact on codes
- Libra tool set
  - Collect and visualize load data across ranks and time
  - Adaptive data compression
  - Interactive GUI and data presentation

Using Libra with the Progress/Effort Model

1. Manually instrument progress loop
   - Gives us time axis
   - Add MPI_Pcontrol
2. Collect stacktraces at MPI events
   - Mark start/end of effort regions
3. Every progress step:
   - Record cumulative time spent in effort regions

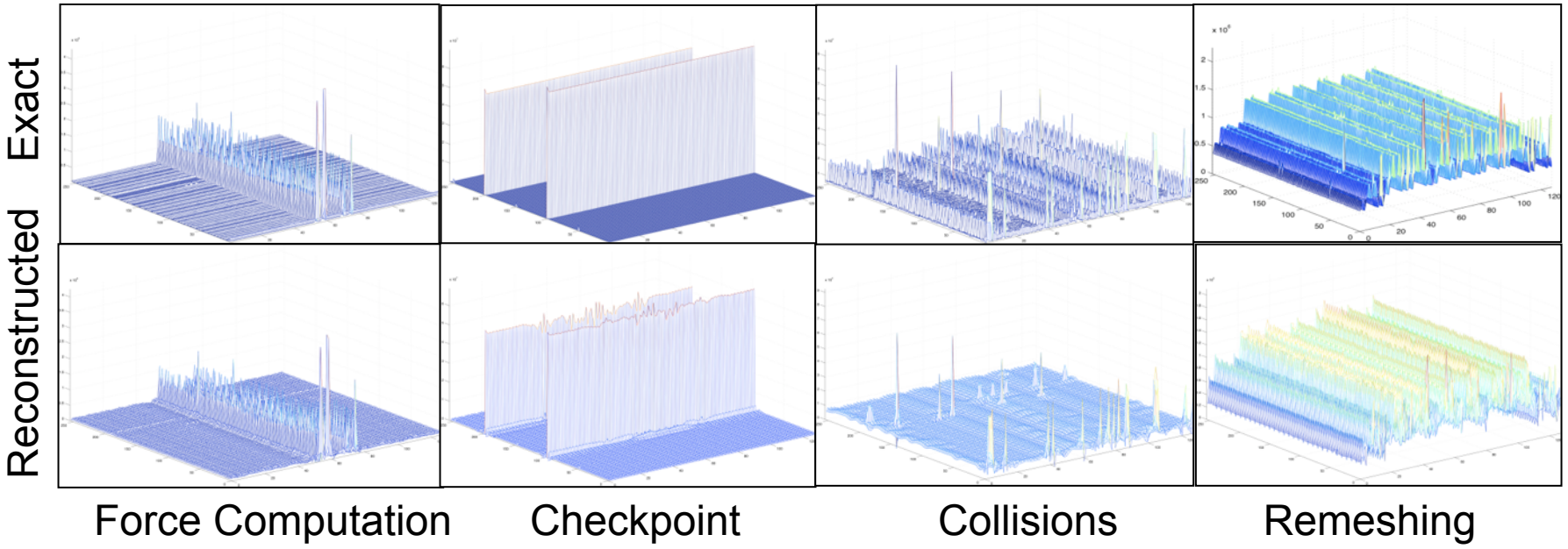
```
for (int i=0; i < max_timestep; i++) {
  MPI_Barrier();

  // convergent solver
  while (!converged) {
    // ... computational effort ...
    MPI_Allgather();
  }

  MPI_Allreduce(...);

  // .. further effort ...
  MPI_Waitall(...);

  MPI_Pcontrol(0); // progress marker
}
```



- Region per pair of dynamic callpaths
  - One surface plot per region
  - “Slice” code into different load behaviors
  - Selectable in GUI

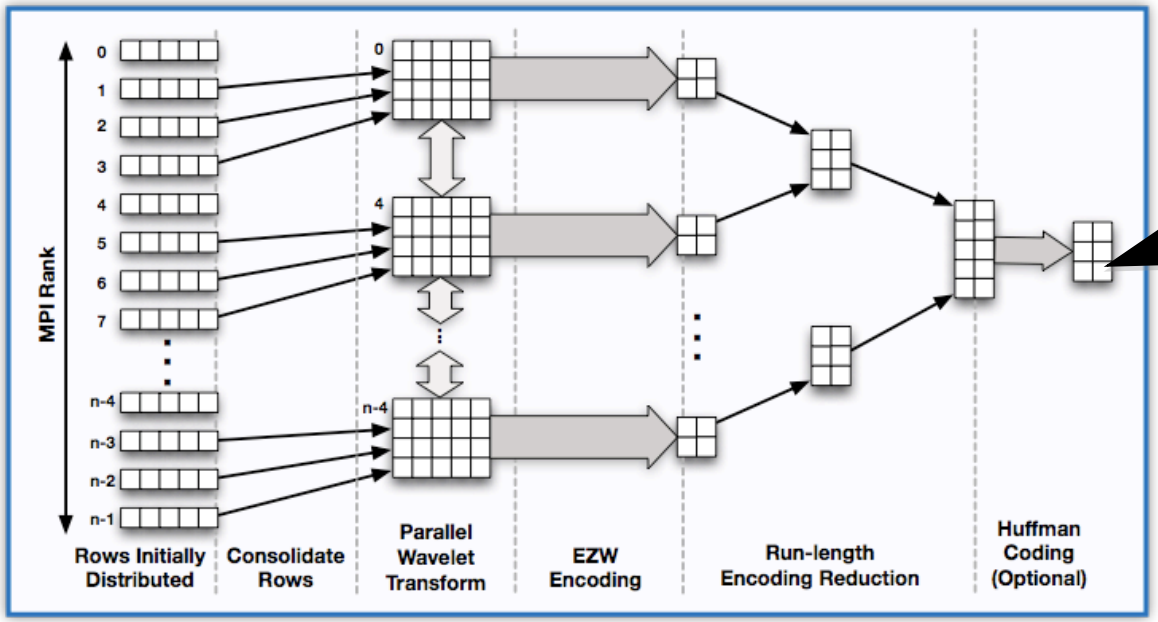
Data Compression using Wavelets

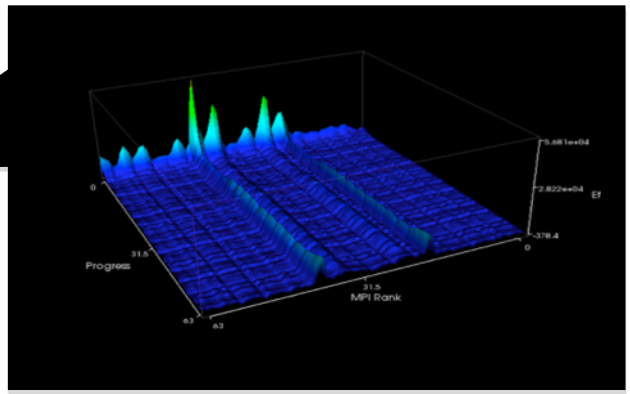
- Wavelets offer efficient compression but maintain structure
- Multi-scale representation -> level-of-detail visualization
- Compression has to be parallel itself



Example: Crystal Dislocation Dynamics Code

Libra Uses Wavelets to Achieve Scalability

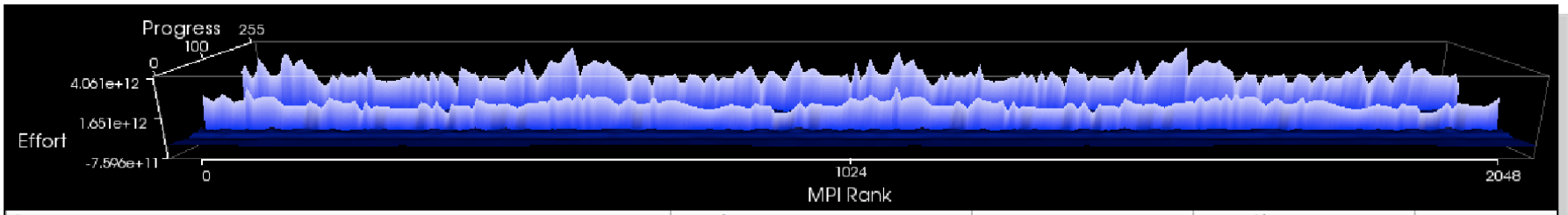




- Fast compression relies on inter-process analysis
  - ~4-5s to write 4.7 GB compressed to ~10MB on 16K cores
  - Wavelet transform uses nearest-neighbor communication
  - Interleave compression from multiple regions

Example of Using Libra

- Case study on S3D on Intrepid (ANL)
  - Symptoms: bad scaling behavior
  - Mainly seen at large scale
  - Libra directly pointed to I/O routines for checkpointing
    - “Two walls” visible in the plot
    - Each indicates one checkpoint operation
  - Problem caused by varying I/O performance



Start	TotalTime	Deviation	MeanSkew	MeanKurtosis
write_savefile(io.f90:312)	1.28868e+16 (19.98%)	7.79937e+11	3.42771	14.5355
write_savefile(io.f90:322)	8.83365e+15 (13.69%)	5.33235e+11	3.30364	13.637
write_savefile(io.f90:312)	7.97592e+15 (12.36%)	7.96813e+11	5.00885	27.2989
write_savefile(io.f90:322)	5.35586e+15 (8.30%)	5.35976e+11	5.01255	27.3279

Summary

- Several profilers for MPI exist
  - Good to gain initial overview of communication behavior
  - Typically gather data into few output files
  - Use tools like P^nMPI to transparently add context
- Sampling/Profiling tool kits
  - Open|SpeedShop: focus on easy of use
  - TAU: large variety of metrics and display options
  - Kojak/Scalasca: profiling + automatic analysis
- Specialized tool kits like Libra can help for targeted problems
- Automatic analysis will become more and more dominant
  - Data volumes are growing / manual analysis infeasible
  - Generic profiles only provide so much information
  - Look for particular bottlenecks like “late sender”
  - Clustering/outlier analysis
  - Performance analysis will itself be a parallel application!

PRESENT: SCALABLE TRACING

Scalable Tracing Approaches

- **HPCToolkit** [Rice Univ., http://www.hpctoolkit.org]
  - Tracing of call stack samples
- **SLOG-2 / Jumpshot-4** [ANL]
  - Frame-based trace data format / more scalable visualizations
- **Paraver** [BSC/UPC http://www.cepba.upc.es/paraver]
  - Automatic detection + identification of program phase structure
  - Powerful filter and summarization features
  - Scalable visualization

Scalable Tracing Approaches II

- **VampirServer** [TU Dresden, ZIH, http://www.vampir.eu]
  - Visualization client + parallel trace analysis server
  - Standalone tracing library or integrated into Open|SpeedShop
- **ScalaTrace** [NCSU, LLNL http://moss.csc.ncsu.edu/~mueller/scala.html]
  - Intra and inter process lossless trace compression
- **Scalasca** [JSC: http://www.scalasca.org]
  - Parallel automatic trace analysis
  - Scalable metrics display

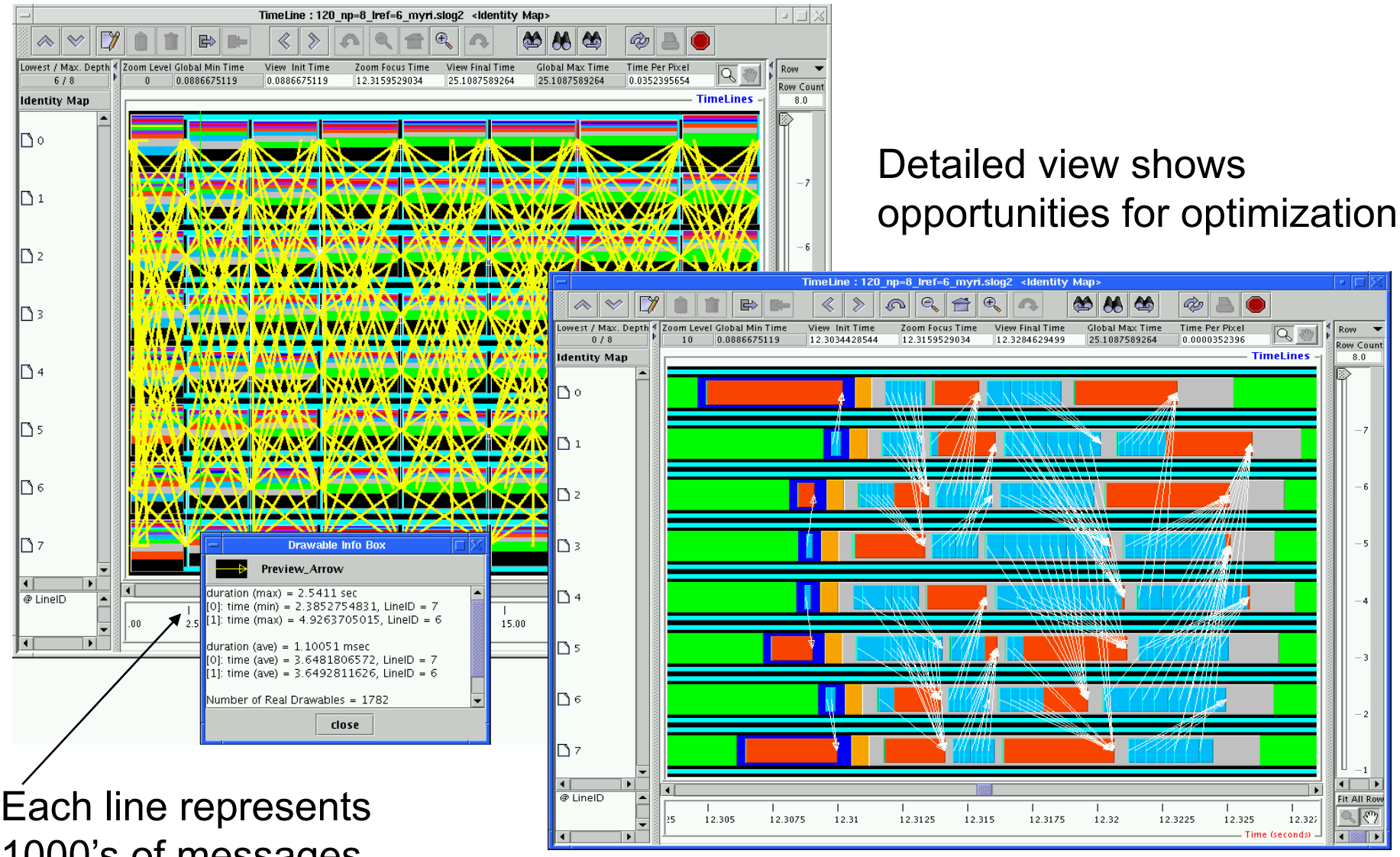
Frame-based trace data format

- Argonne National Laboratory

- **SLOG-2** trace format
  - Trace file consists of visualization-friendly **interval records** (include duration)
  - Separate **frames** for multiple size of intervals
  - To display region around a single point of time, use **index** to quickly locate and read the right frames
  - **Index based on binary tree of bounding boxes**

- Example:
  - Typically under 100ms to display a specific time interval out of 19GB trace file

Viewing Multiple Scales with Jumpshot



Detailed view shows opportunities for optimization

Each line represents 1000's of messages

1000x zoom

ScalaTrace: Intra + Inter Node Compression

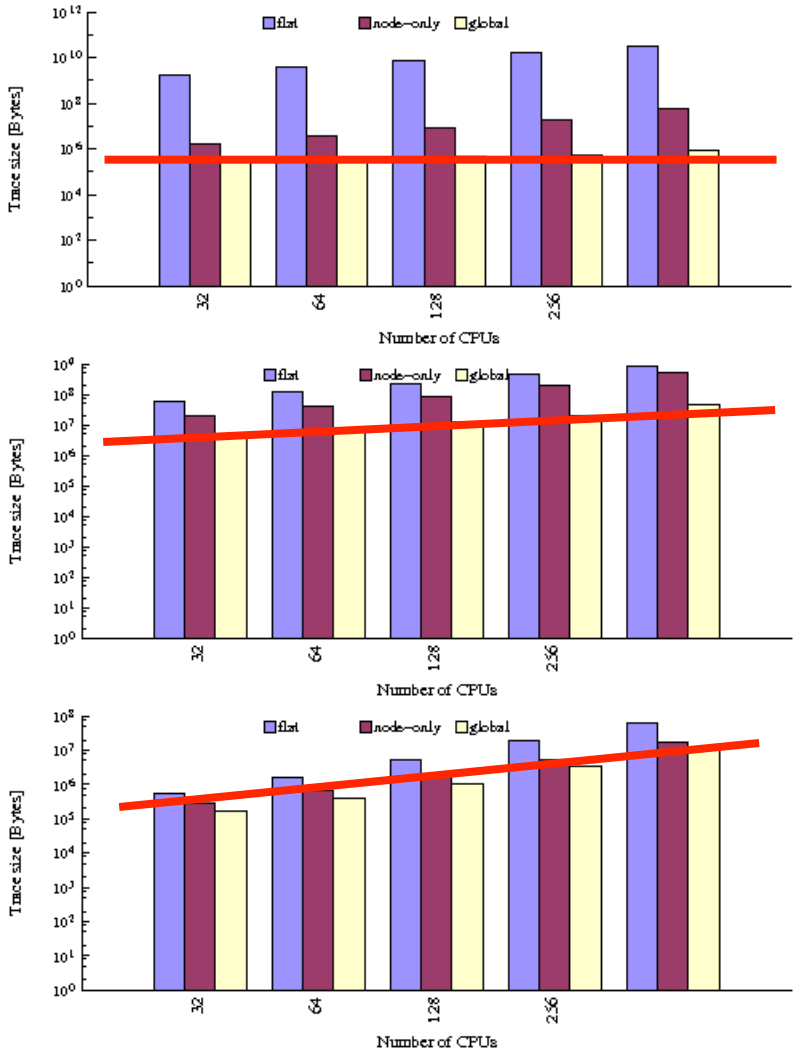
- NC State University + LLNL/CASC
- **Goal: logical replay**; less useful for performance analysis
- **Intra-node compression framework**
  - Event aggregation with special handling of MPI_Waitsome
  - Maintain structure of call sequences ⇨ stack walk signatures
  - `A B C D E C D E ⇨ (A B ((C D E), iter=2))`
- **Inter-node compression framework**
  - Make use of SPMD nature of MPI codes
  - Identify regular expressions in communication patterns
  - Match operations across nodes by manipulating parameters
    - Source / destination offsets (location independent encoding)
    - Request offsets
- Actual algorithm more complicated, see IPDPS'07 paper
- Option: compute time recording using histograms, see ICS'08 paper
- http://moss.csc.ncsu.edu/~mueller/scala.html

Results: Trace File Sizes

Log size[Bytes], 32-512 cpus

none / intra- / inter-node

- Near-constant
  - NAS EP, DT, LU,BT,FT
  - Instead of linear
- Sub-linear
  - NAS MG, CG
  - Still good
- Non-scalable
  - NAS IS, UMT2k
  - 2-4 orders of magn. smaller



Paraver

- **Performance analysis framework**

- Features
  - Detailed quantitative performance analysis
  - Concurrent **comparative analysis of several traces**
  - Fast analysis of very large traces
  - Support for hybrid MPI/OpenMP applications
  - Just a few simple displays, however „**programmable**“
    - Complex analysis and filter functions
    - Supports definition of derived metrics
    - Metric independent visualization

- http://www.cepba.upc.es/paraver

Paraver Data Reduction Features

- Accumulation of values using **software counters**
- **Powerful filtering** ⇨ expression over time, processors, states, communications, events
- **Automatic structure / phase detection**
  - Based on signal processing
    - Using wavelets (Casas: ParCo 2007)
    - Using autocorrelation functions (Casas: Euro-Par 2007)
  - Also used for cleanup:
    - Preemptions
    - Clogged systems / instrumentation overhead
    - Flushing

Paraver: Iterative pattern detection (2001)

- Periodicity detection based on distance metric

$$d(m) = sign \sum_{i=0}^{N-1} | x(i) - x(i-m) | \; 0 < m < M \wedge M \leq N$$

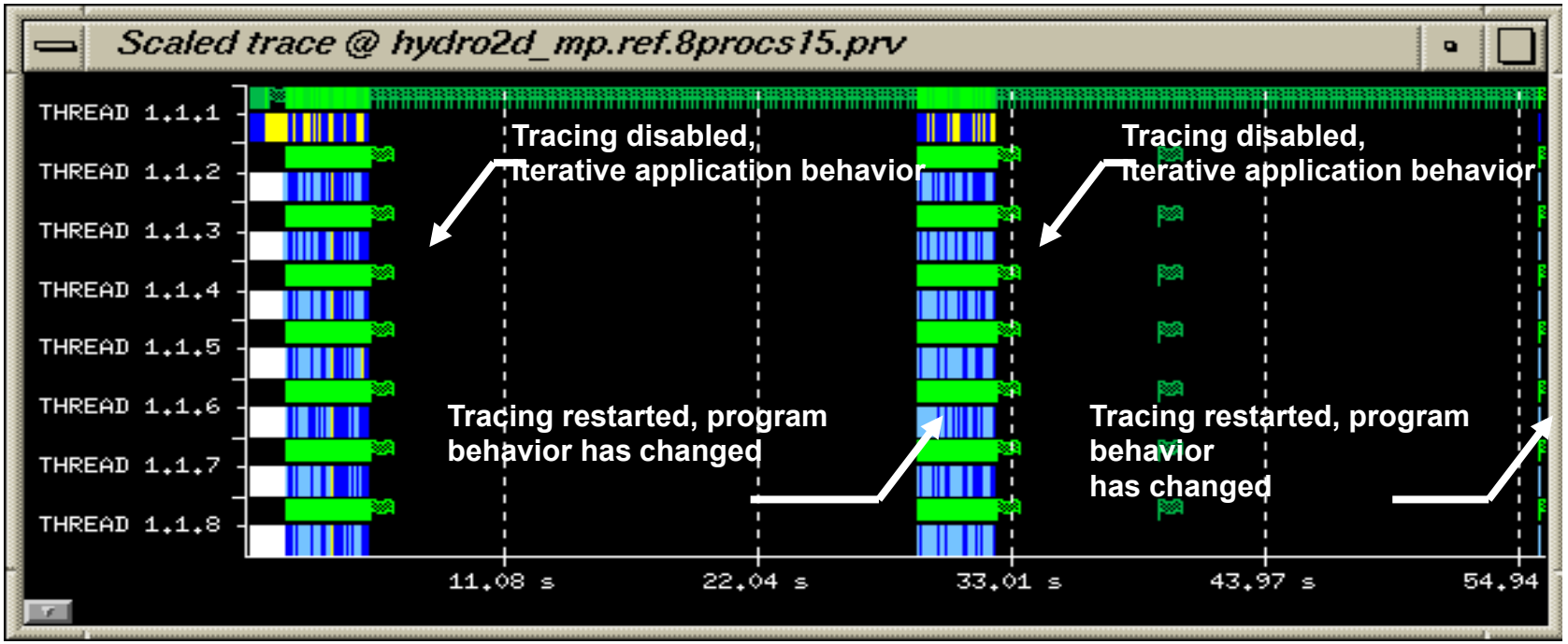
  - Applied to stream of function identifiers
  - Computes vector distance between x(i) and x(i-m)
    ::= distance between vector x(i)
    and same vector shifted by m samples

If d(m) = 0 : periodic pattern with period length m

  - Efficient detection algorithm
    with complexity O(M) per event

Example: Iterative pattern detection

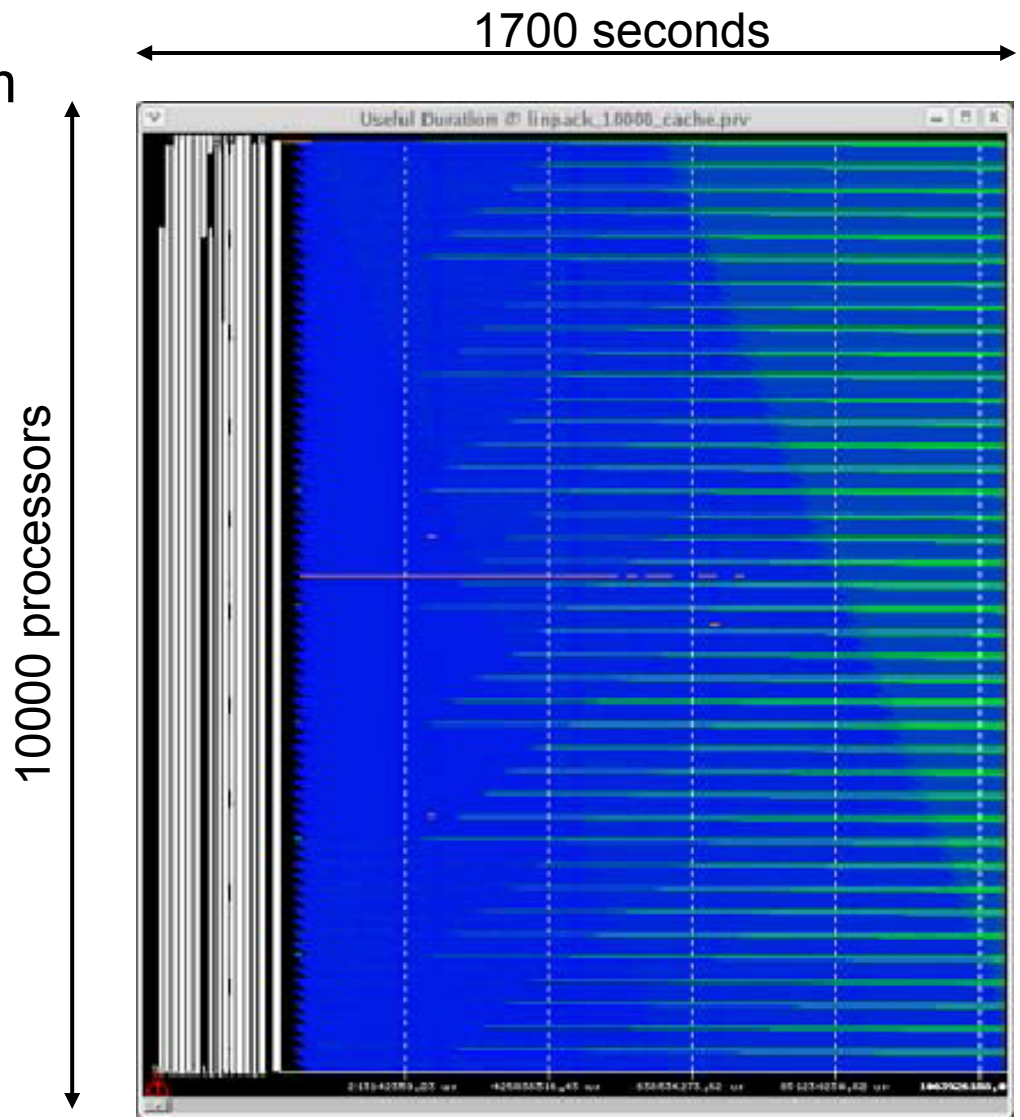
- Example: SPEC Hydro2D



- Overhead
  - Original tracing mechanism: 1-3%
  - With periodicity detection: 3-6%

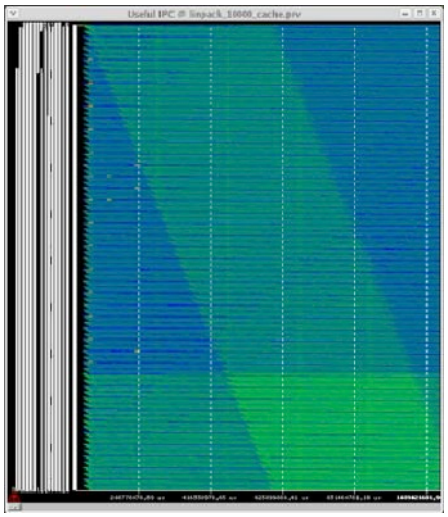
Paraver: Scalability of Display

- Linpack @ MareNostrum
- Dgemm duration

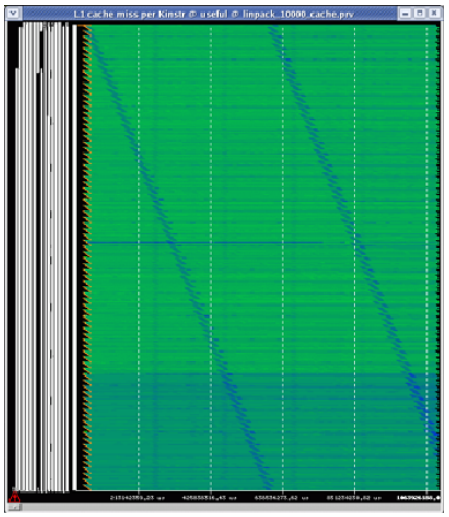


Paraver: Scalability of Display

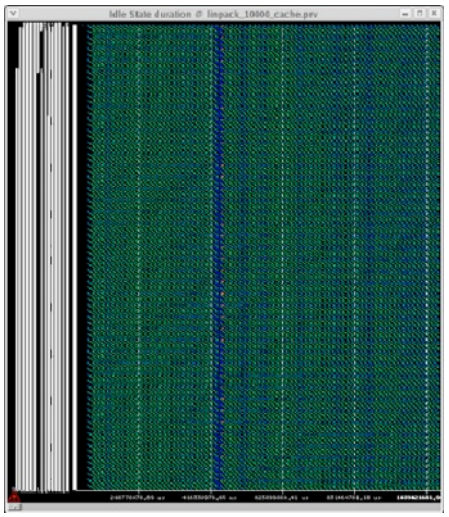
- Linpack @ MareNostrum



Dgemm
IPC



Dgemm
L1 miss ratio



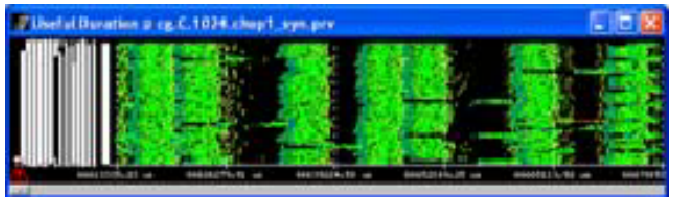
Communication
duration

Paraver: Scalability of Display

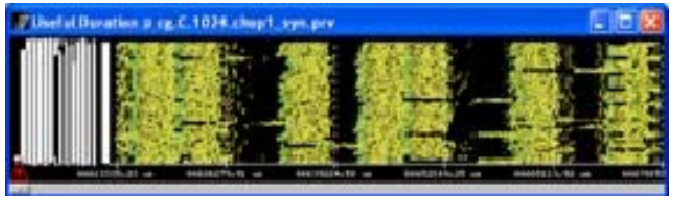
- Non-linear render

CG.C
1024 CPUs
Useful duration

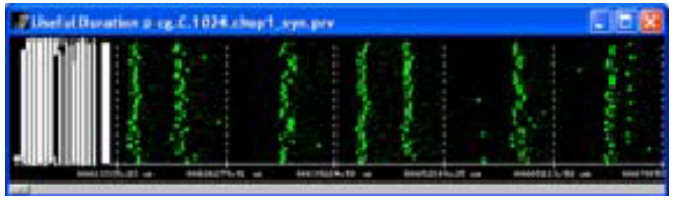
Max



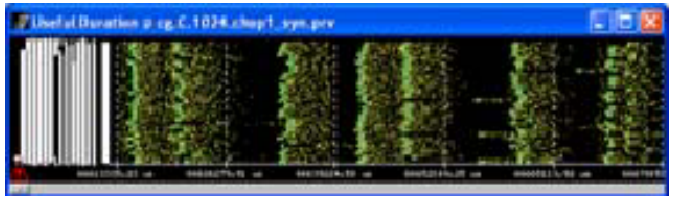
Min != 0



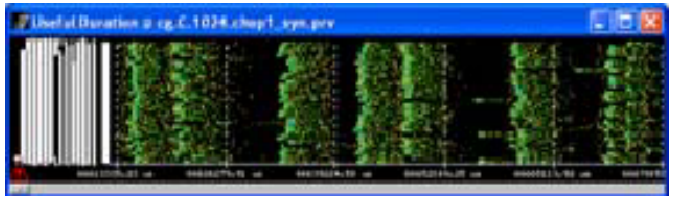
Last



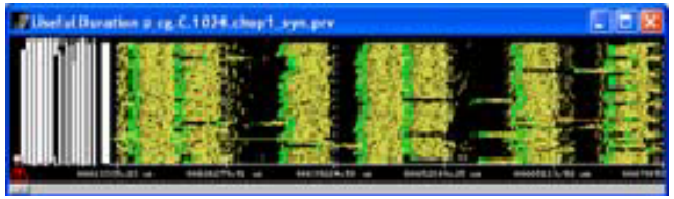
Random



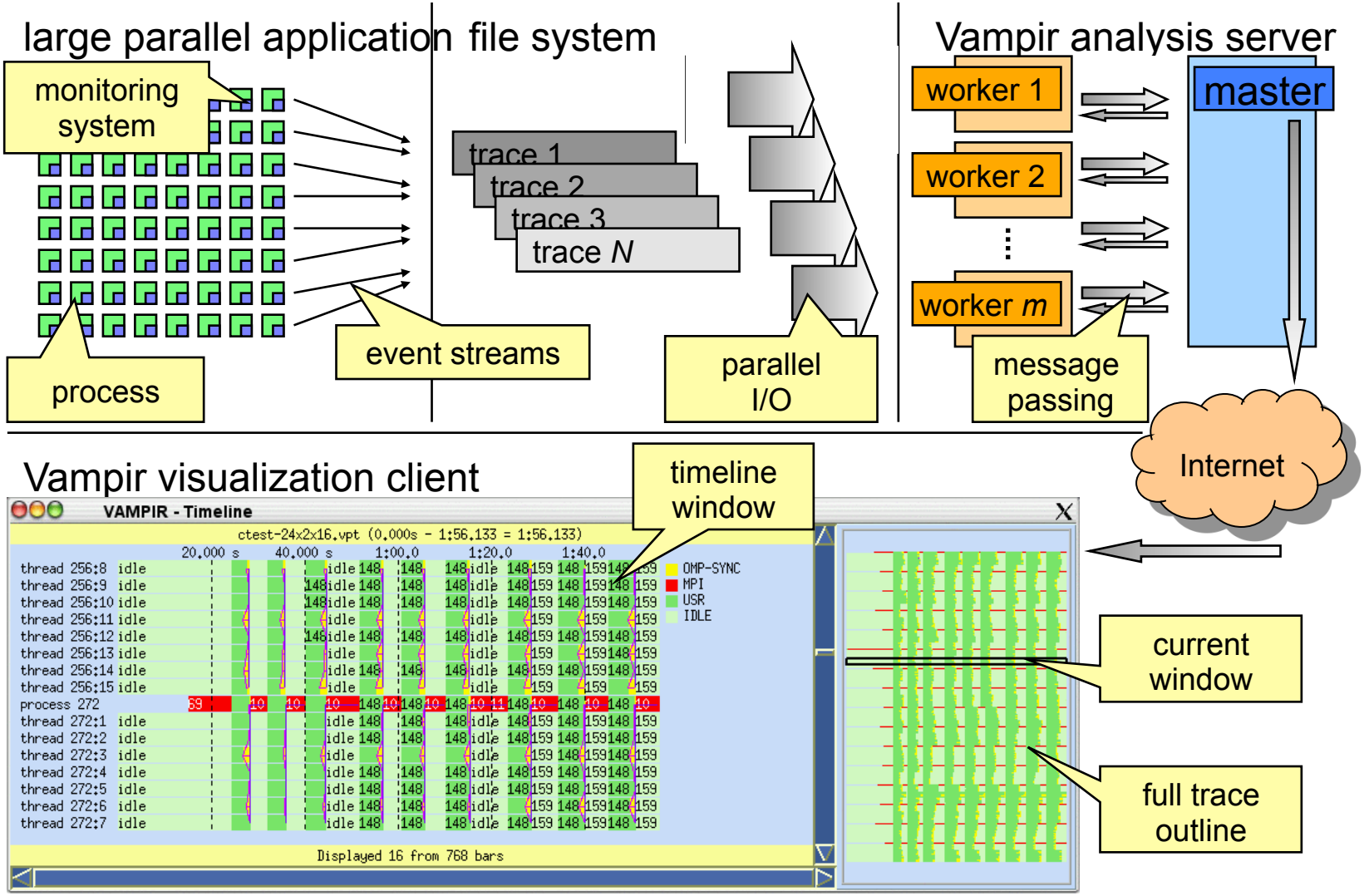
Random != 0



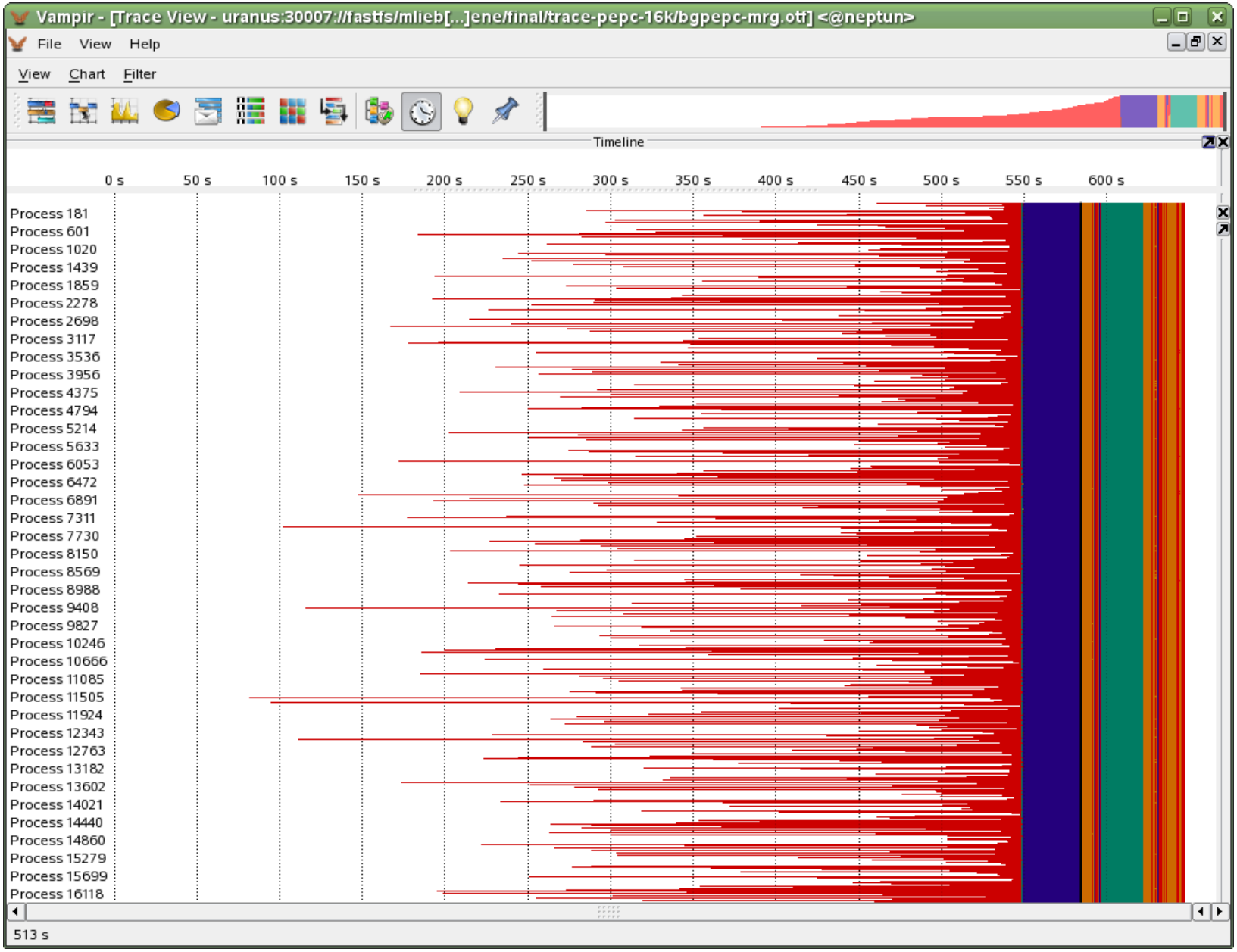
Average



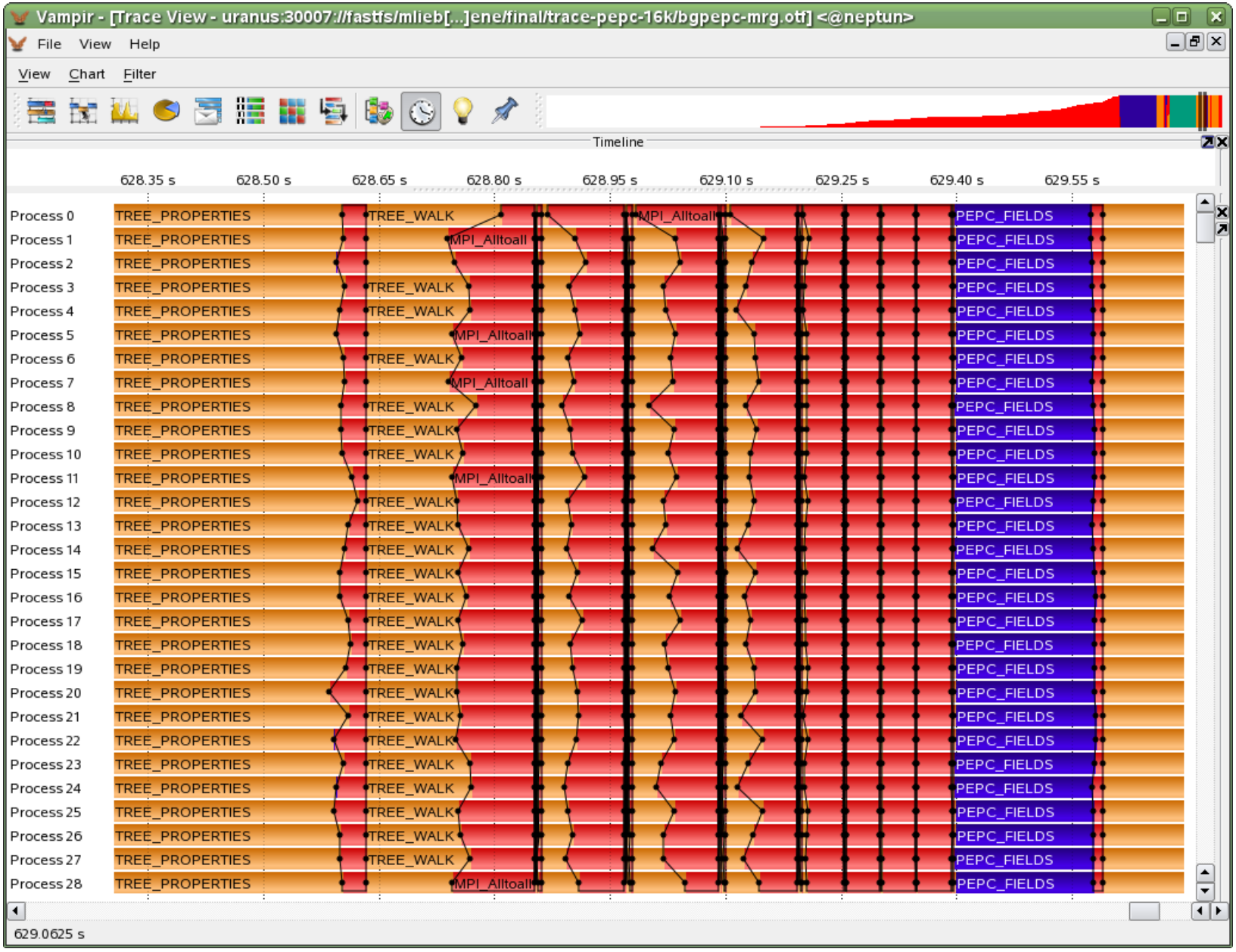
VampirServer Architecture



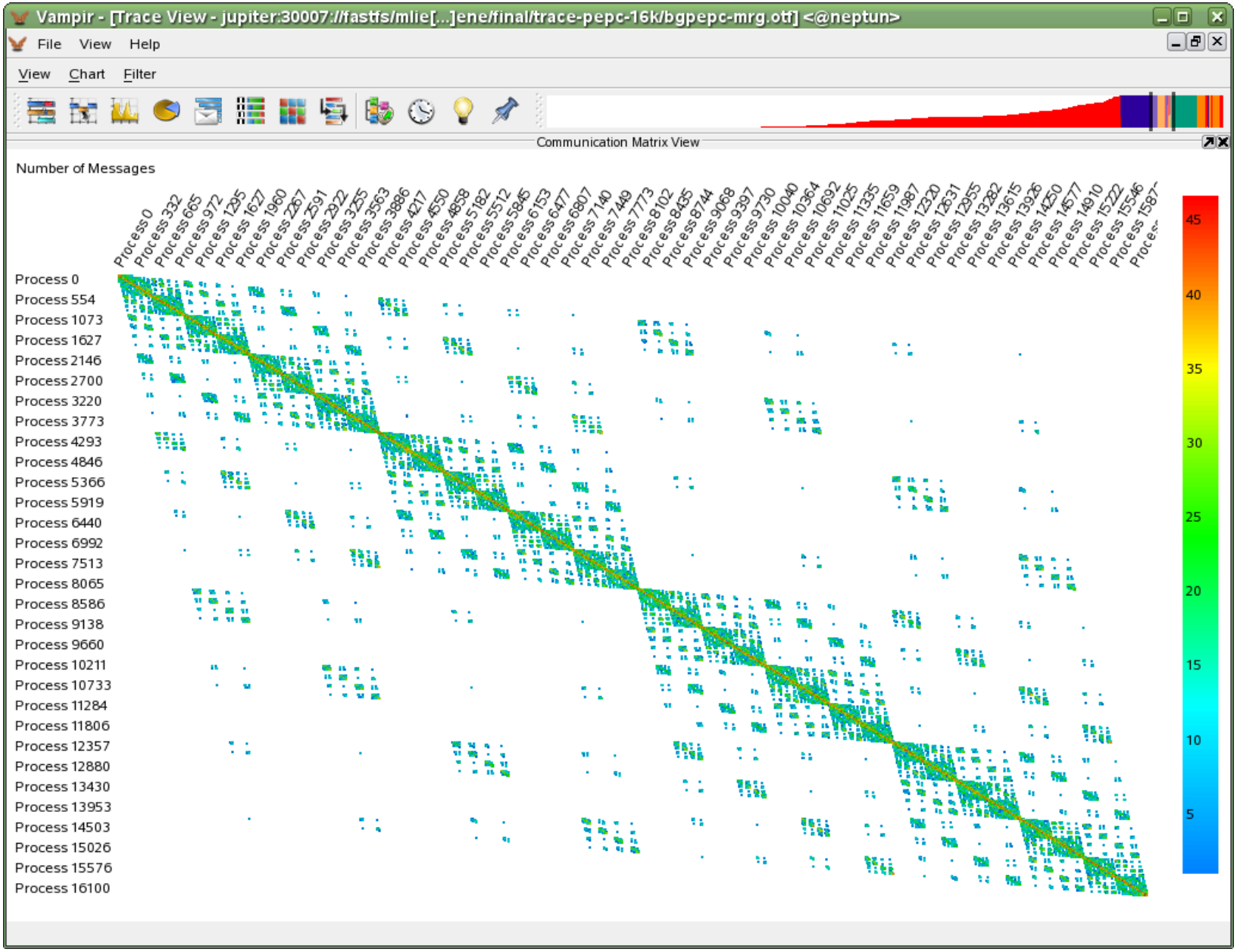
VampirServer:
PEPC, 16384 PEs, Global Timeline



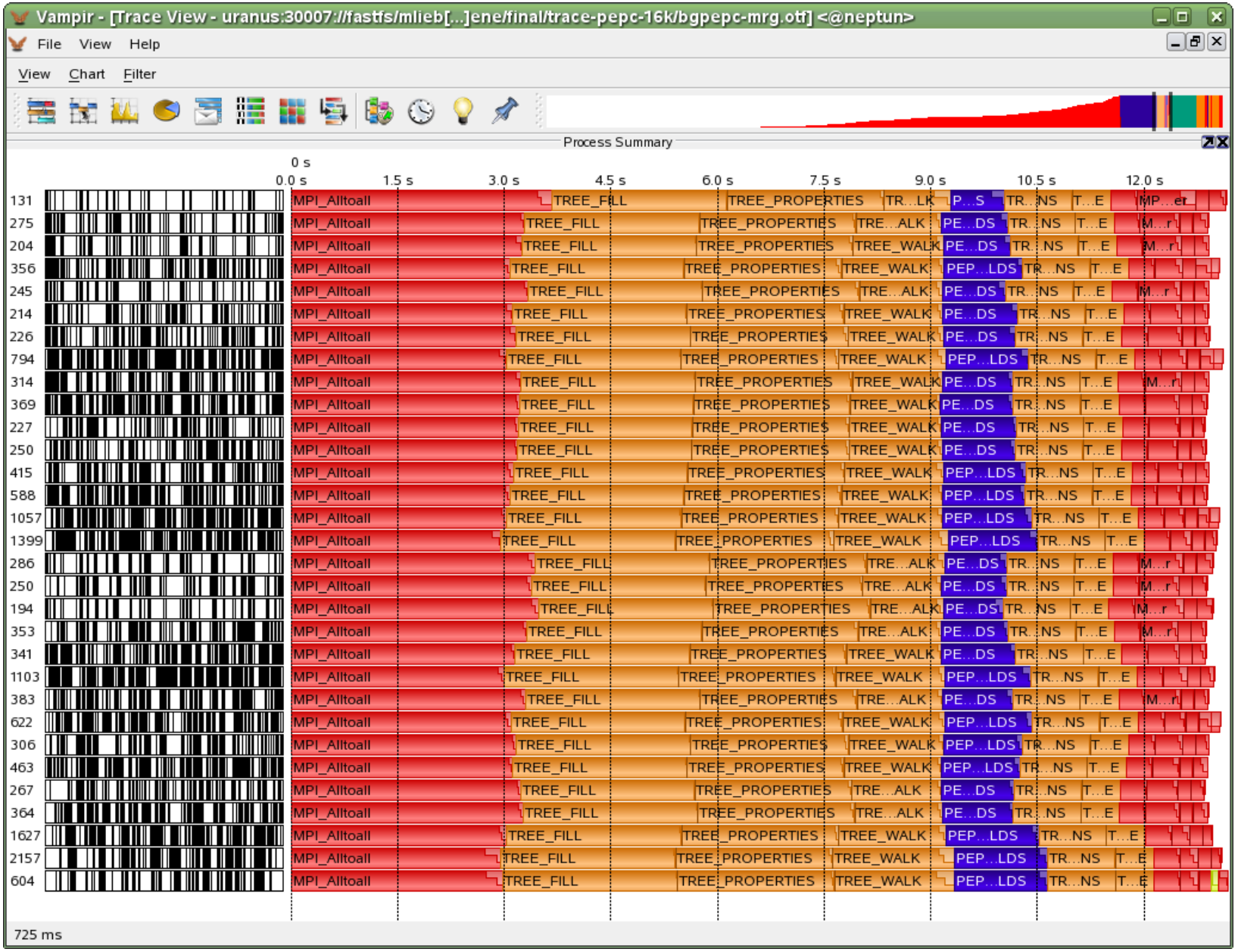
VampirServer:
PEPC, 16384 PEs, Global Timeline (zoomed)



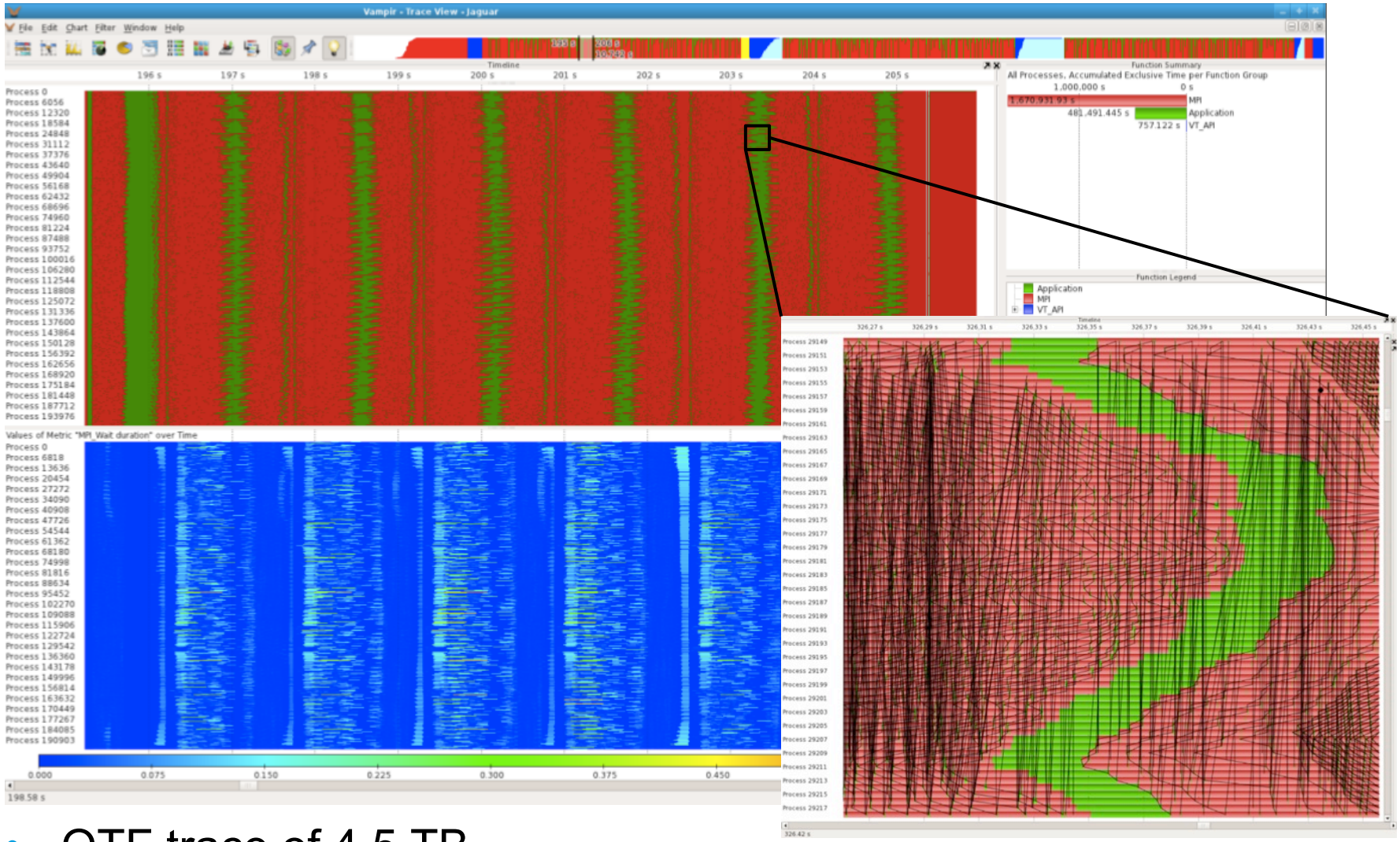
VampirServer:
PEPC, 16384 PEs, Message Statistics



VampirServer:
PEPC, 16384 PEs, Cluster Analysis



VampirServer BETA: Trace Visualization S3D@200,448



- OTF trace of 4.5 TB
- VampirServer running with 20,000 analysis processes

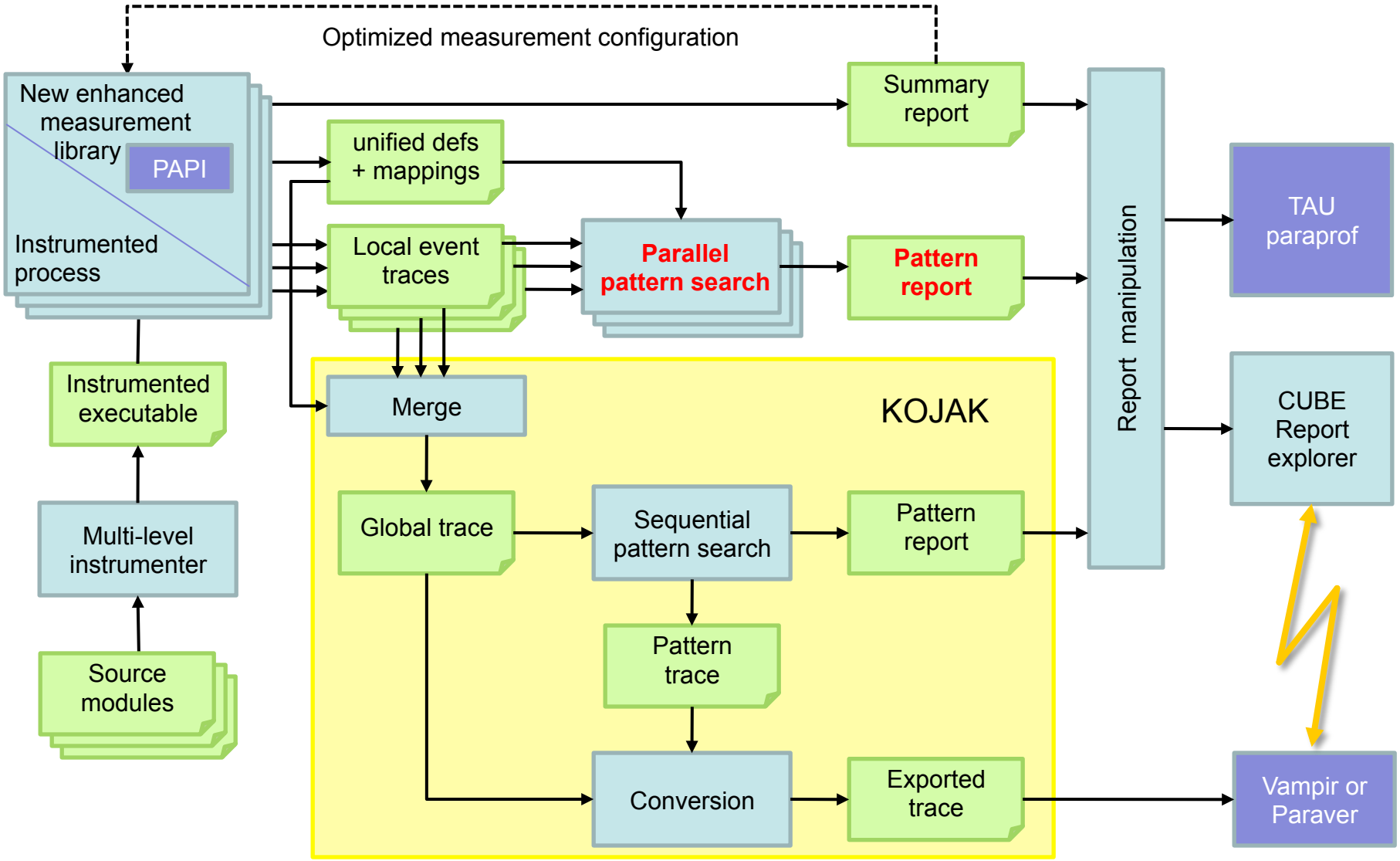
The Scalasca Project

- Scalable Analysis of Large Scale Applications
- Follow-up project to KOJAK
- Started in January 2006

- Objective 1: do not rely on tracing only
  - ⇨ Supports scalable **call-path profiling**
- **Objective 2:** develop a scalable version of KOJAK
  - ⇨ Basic idea: **parallelization of trace analysis**

- Supports MPI 2.2 (P2P, collectives, RMA, IO) and basic OpenMP (no nesting)
- http://www.scalasca.org/

New Analysis Process II

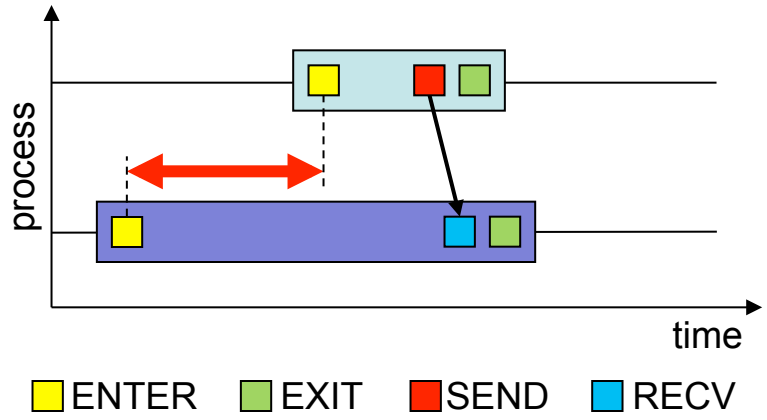


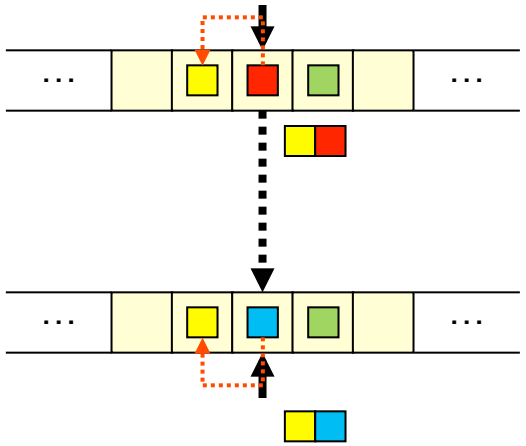
= Third-party component

Scalable Automatic Trace Analysis

- **Parallel pattern search to address wide traces**
  - As many processes / threads as used to run the application
    ⇨ Can run in same batch job!!
  - Each process / thread responsible for its "own" local trace data
- **Idea: "parallel replay" of application**
  - Analysis uses communication mechanism that is being analyzed
  - Use MPI P2P operation to analyze MPI P2P communication, use MPI collective operation to analyze MPI collectives, ...
  - Communication requirements not significantly higher and (often lower) than requirements of target application
- **In-memory trace analysis**
  - Available memory scales with number of processors used
  - Local memory usually large enough to hold local trace

Example: Late Sender





Sender

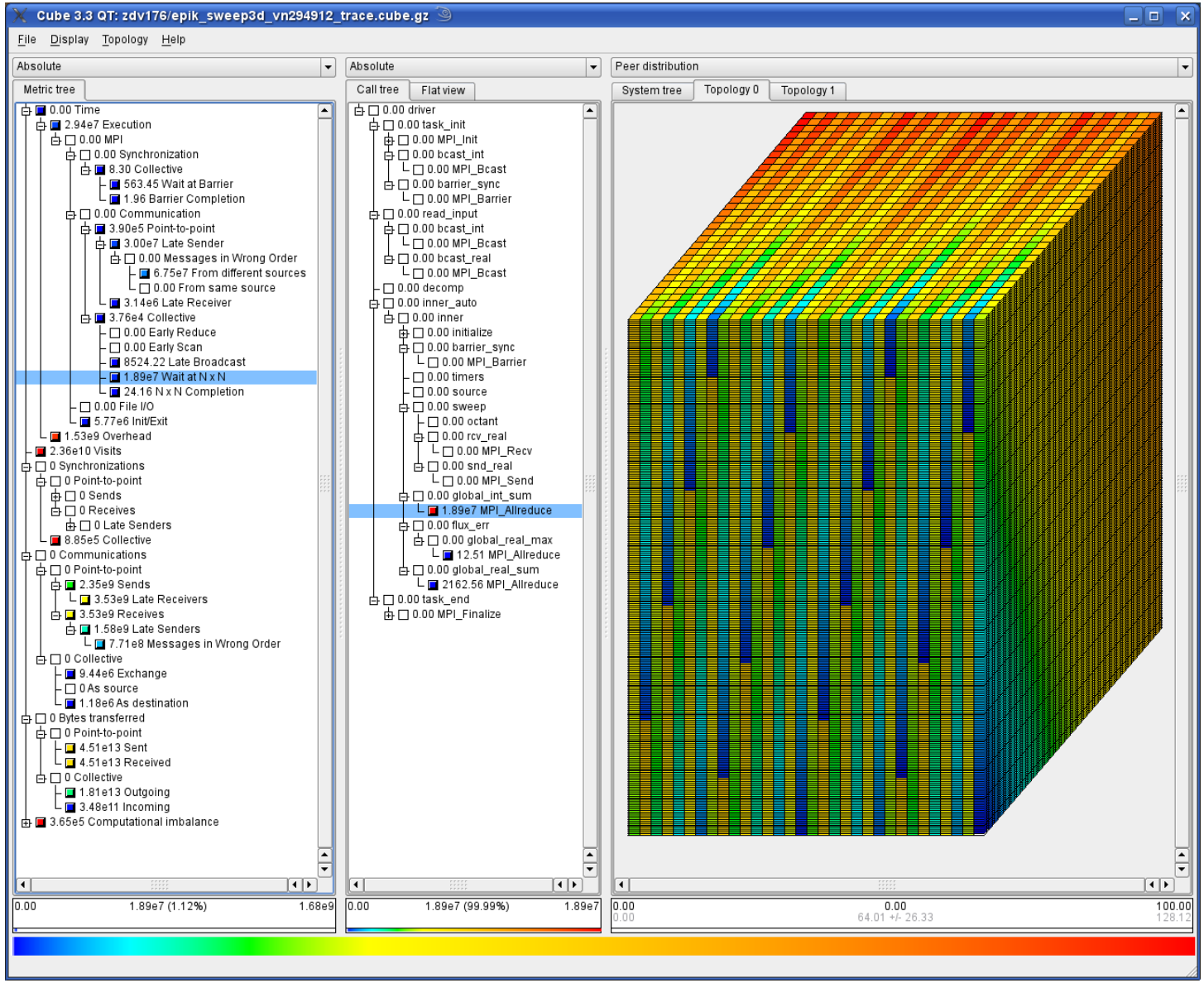
- Triggered by send event
- Determine enter event
- Send both events to receiver

Receiver

- Triggered by receive event
- Determine enter event
- Receive remote events
- Detect **Late Sender** situation
- Calculate & store waiting time

Trace analysis sweep3D@294,912

- 10 min sweep3D runtime
- 11 sec replay
- 4 min trace data write/read (576 files)
- 7.6 TB buffered trace data
- 510 billion events



Summary

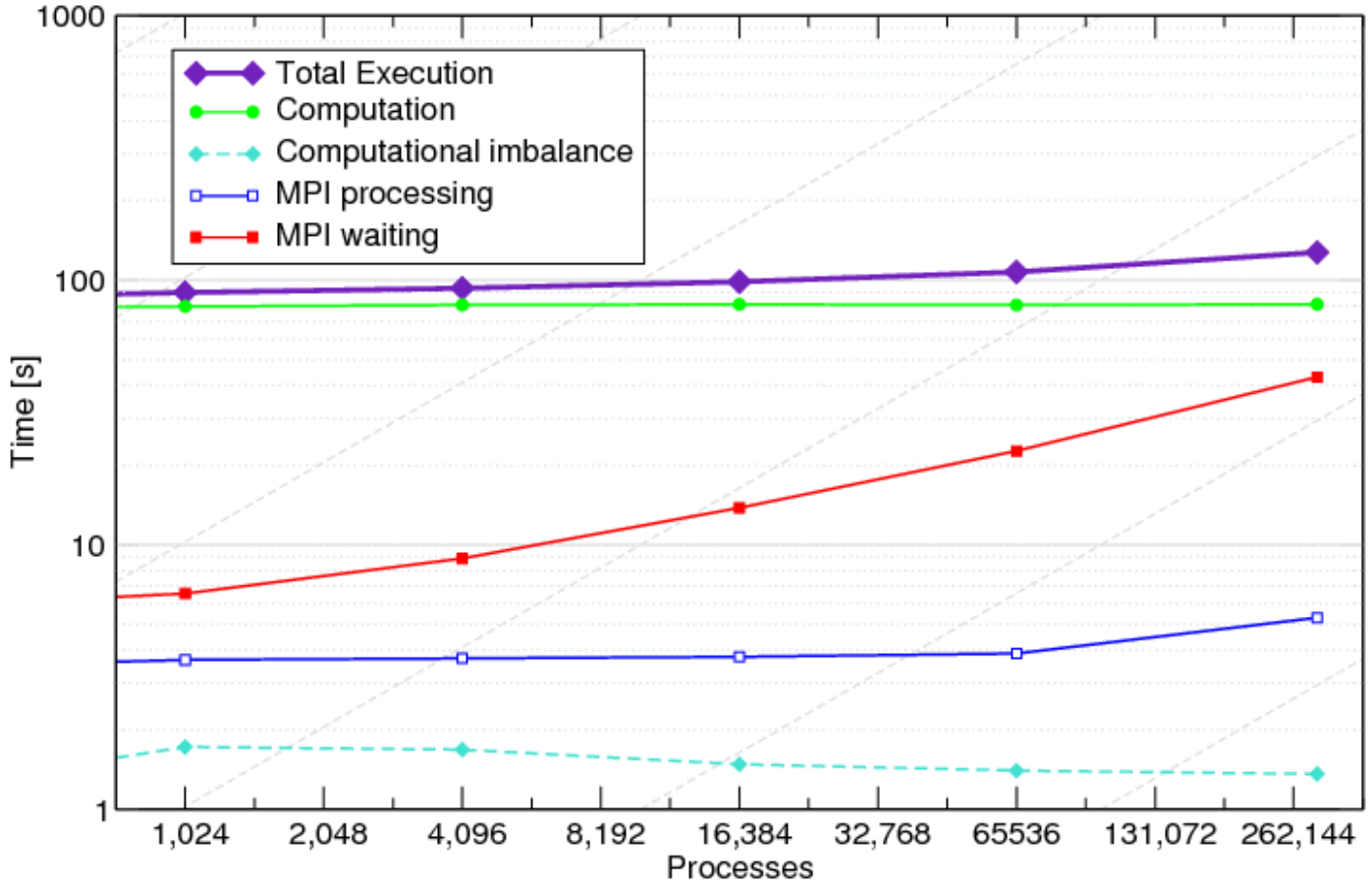
- Trace information can provide detailed insight
  - Several tool sets with their own pros and cons
  - Conversion between data formats necessary
- New techniques to reduce and display at scale
  - Parallel server/viewer architectures (VampirServer)
  - Trace folding: removing repetitive phases (Paraver)
  - Pattern detection and compact storage (ScalaTrace)
- Automatic trace analysis
  - Phase detection (to drive folding)
  - Pattern detection (e.g., “late sender” detection)
- Tools will require significant processing capabilities
  - Parallel applications themselves
  - Tool usage is no longer “free”

SCALABILITY CASE STUDIES

Scalasca case study 1: Sweep3D

- ASCI benchmark code from Los Alamos National Laboratory
  - 3-dimensional neutron transport simulation
  - direct order solve uses diagonal wavefront sweeps over grid cells combined with pipelining of blocks of *k*-planes and octants
  - execution performance extensively modeled & analyzed
- MPI parallel version using 2D domain decomposition
  - ~2,000 lines of code (12 source modules), mostly Fortran77
  - very portable, and highly scalable
  - tunable via input deck, e.g., number of *k*-planes in blocks (MK=1)
  - benchmark configuration does 12 iterations
    - flux correction 'fix-ups' applied after 7th iteration
- Run on *jugene* BG/P with up to 294,912 processes
  - Summary and trace analysis using Scalasca
  - Full automatic instrumentation, no runtime filtering

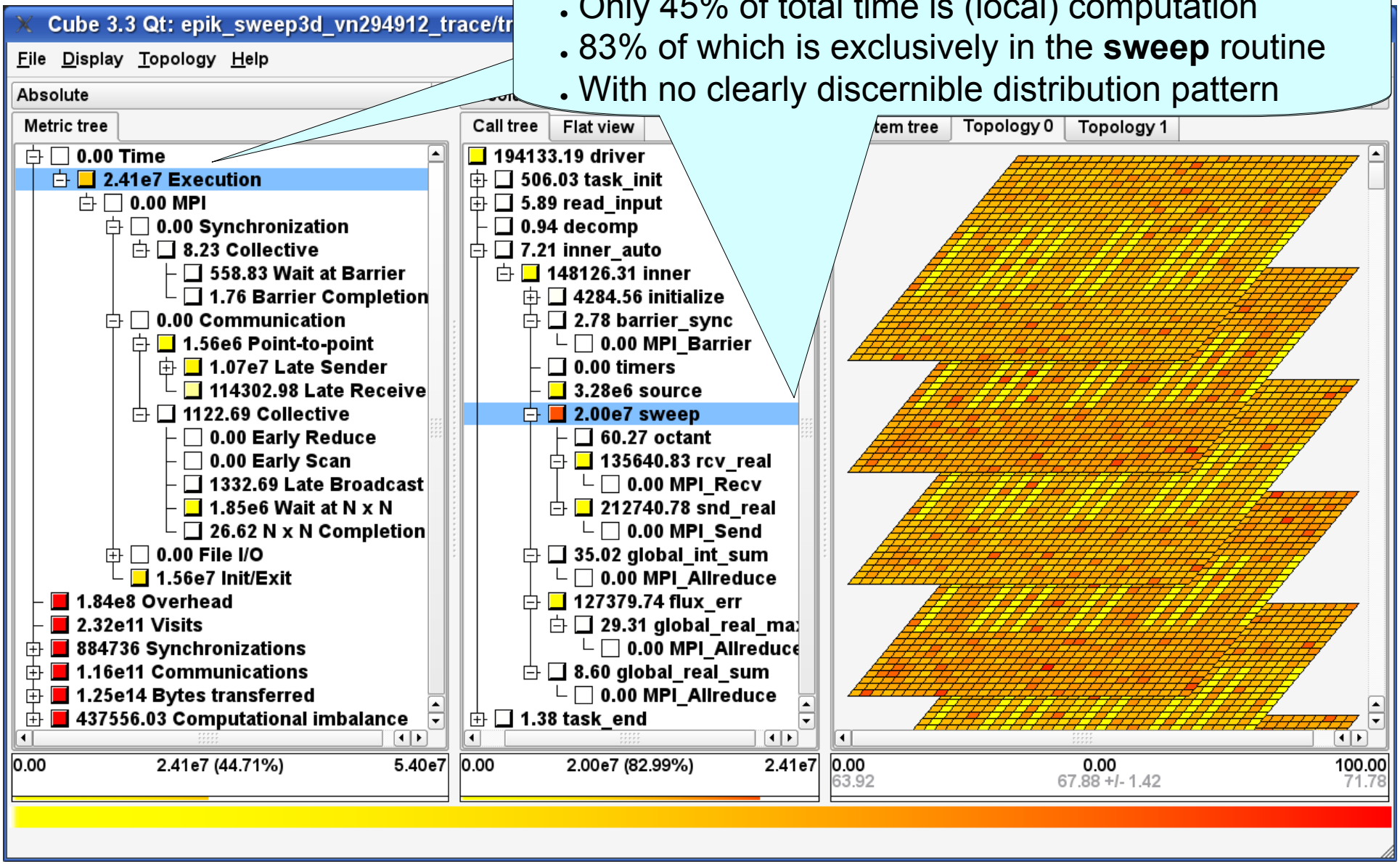
Sweep3D scaling analysis (BG/P)



- Good performance and scalability to largest scale
- Computation time constant (due to *weak scaling*)
- MPI time grows due to markedly increasing waiting times

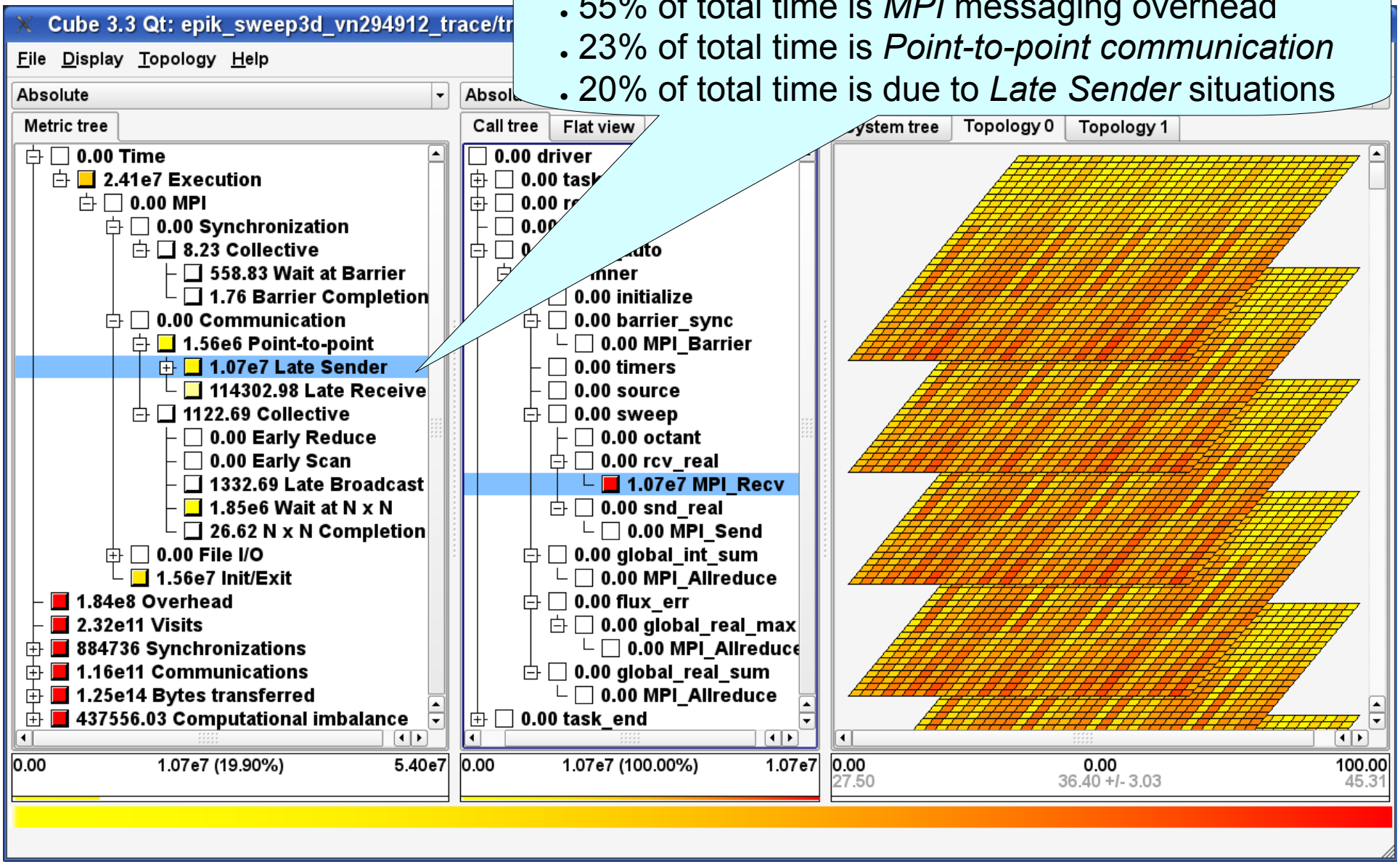
sweep computation time

- Only 45% of total time is (local) computation
- 83% of which is exclusively in the **sweep** routine
- With no clearly discernible distribution pattern



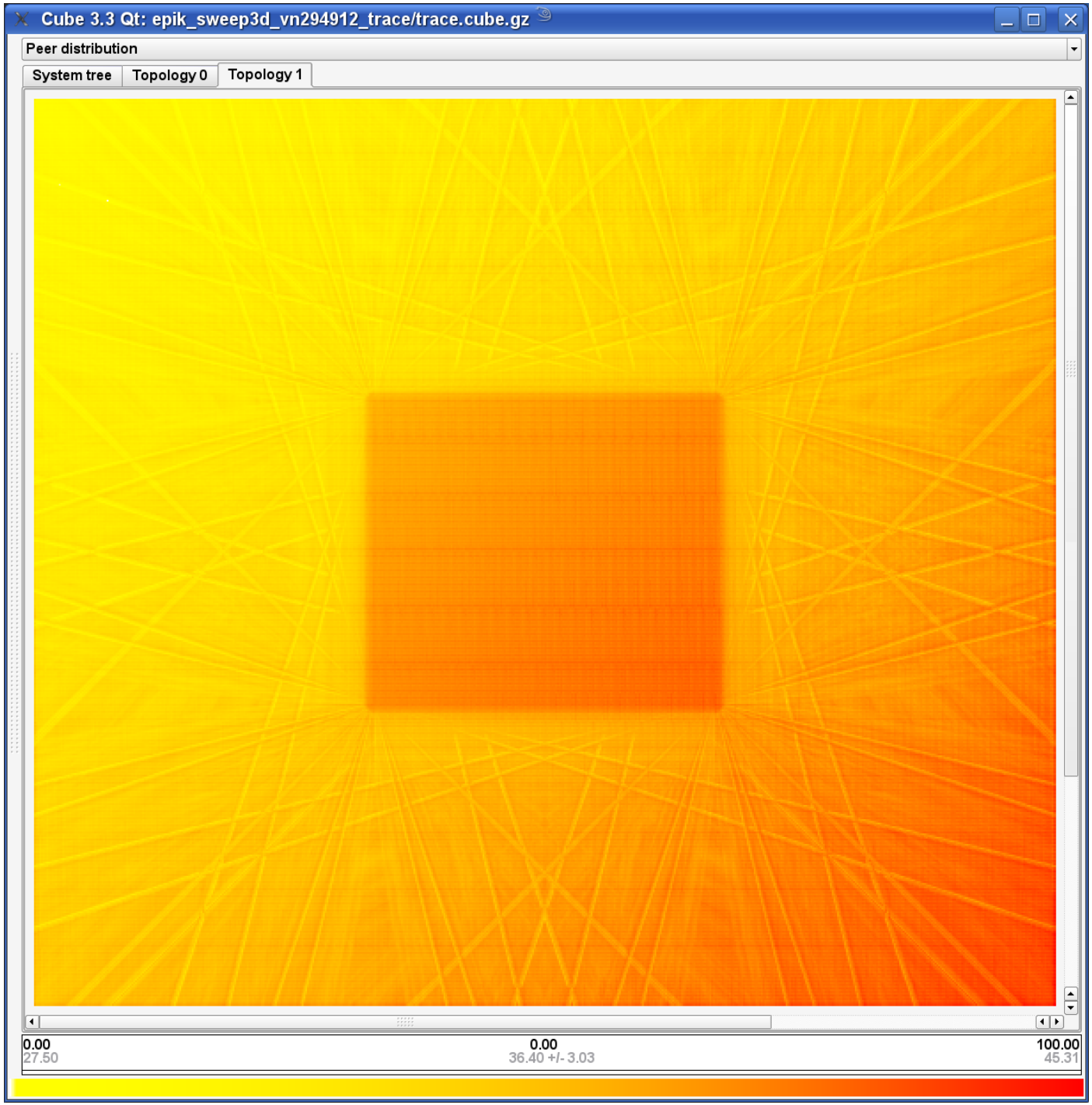
Late Sender time

- 55% of total time is *MPI* messaging overhead
- 23% of total time is *Point-to-point communication*
- 20% of total time is due to *Late Sender* situations

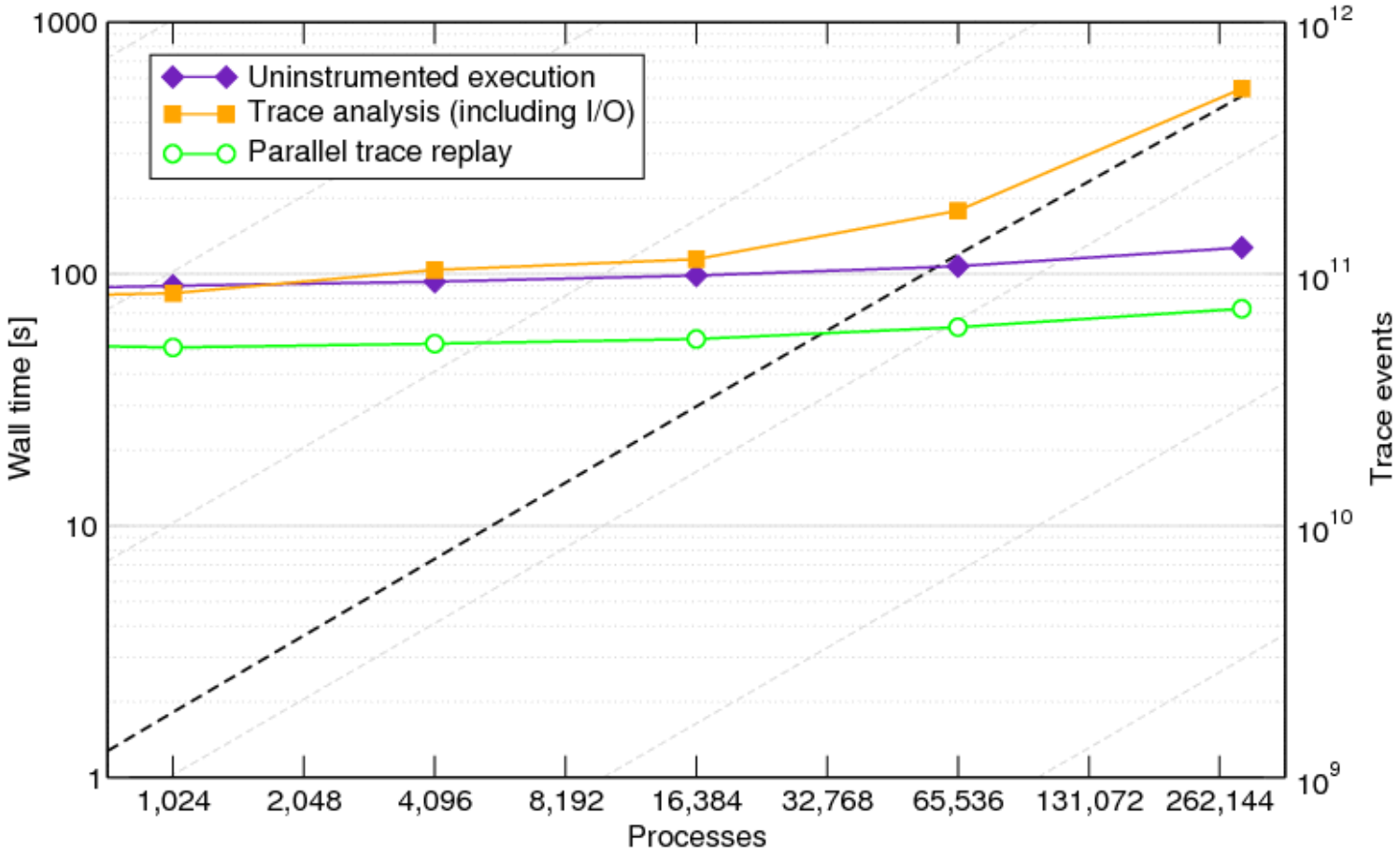


Scalasca topology view

- Application's 576x512 grid of processes
- Reveals clear pattern in the distribution of *Late Sender* metric values
- Arises from superimposing diagonal octant sweeps with imbalanced computation 'fix-ups'



Scalasca trace analysis scaling (BG/P)



- 27MB of trace event records per process
- Total trace size (---) increases to 7.6TB for 510G events
- Parallel analysis replay time scales with application execution time

Scalasca scalability issues/optimizations (Part 1)

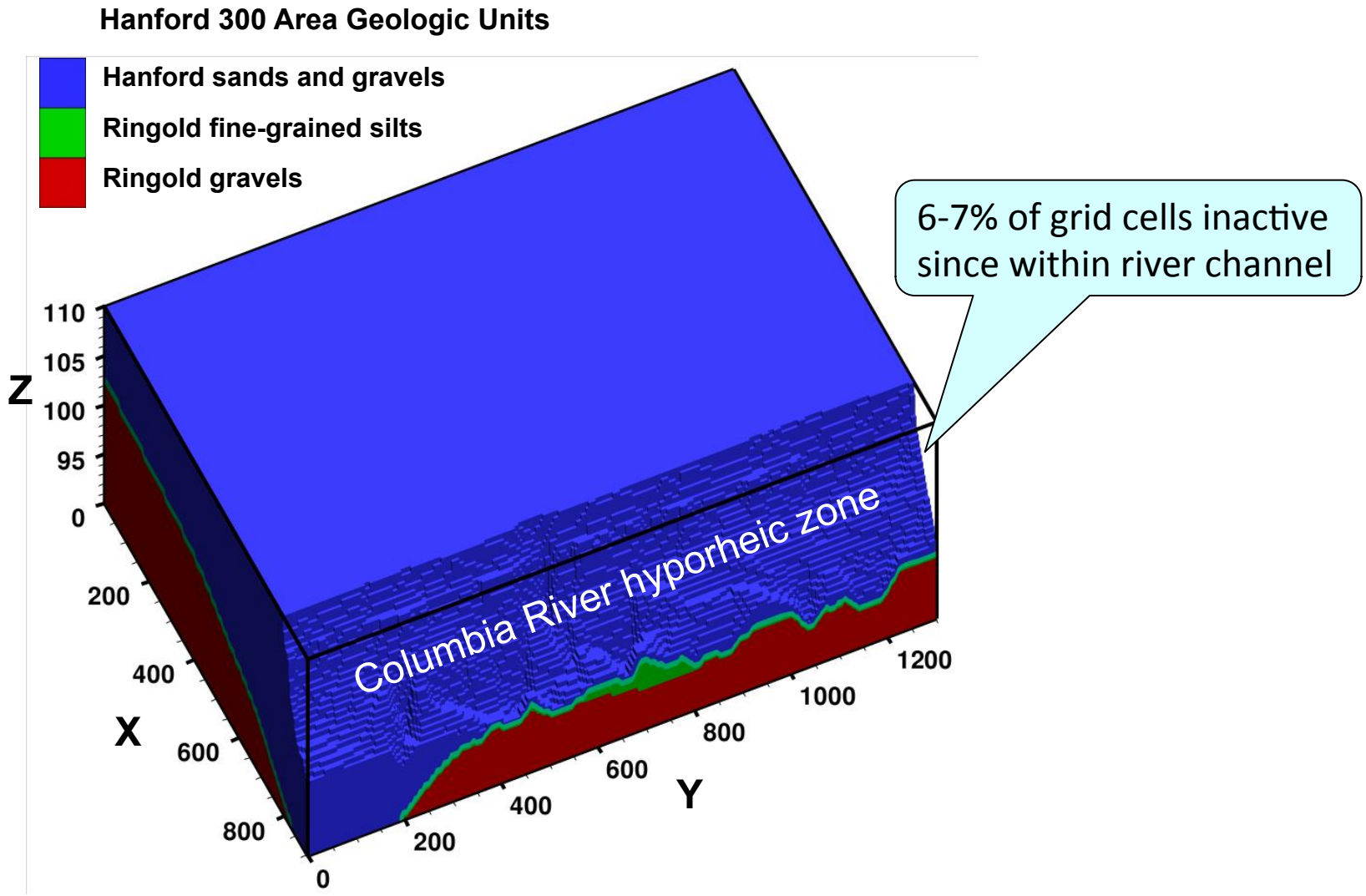
- Trace collection and analysis of Sweep3D successful at largest scale
  - Only 3% measurement dilation versus uninstrumented execution
- Creating individual traces for each process is prohibitive
  - 86 minutes to open/create files
  - Reduced to 10 minutes using *SIONlib* multi-files
    - one shared file per BG/P IONode (576 on jugene)
- Time for unification of identifiers grew linearly with num. processes
  - 40 minutes to generate unified definitions and mappings
  - Reduced to 13 seconds employing hierarchical unification scheme
- Analysis reports grow linearly with number of processes
  - Use binary format for metric values plus XML metadata
  - Store inclusive values and load them incrementally on demand
- Full analysis presentation requires large, high-resolution displays

Scalasca case study 2: PFLOTRAN

- 3D reservoir simulator developed by LANL/ORNL/PNNL
  - ~80,000 lines of Fortran9X, combining solvers for
    - PFLOW non-isothermal, multi-phase groundwater flow
    - PTRAN reactive, multi-component contaminant transport
- employs PETSc, LAPACK, BLAS & HDF5 I/O libraries
  - internal use of MPI
- "2B" input dataset run for 10 simulation time-steps
  - uses 3-dimensional (non-MPI) PETSc Cartesian grid
  - TRAN(sport) step scales much better than FLOW step
  - FLOW step generally faster, but crossover at larger scales
- Scalasca summary & trace measurements
  - IBM BG/P (*jugene*) and Cray XT5 (*jaguar*)

PFLOTRAN "2B" test case

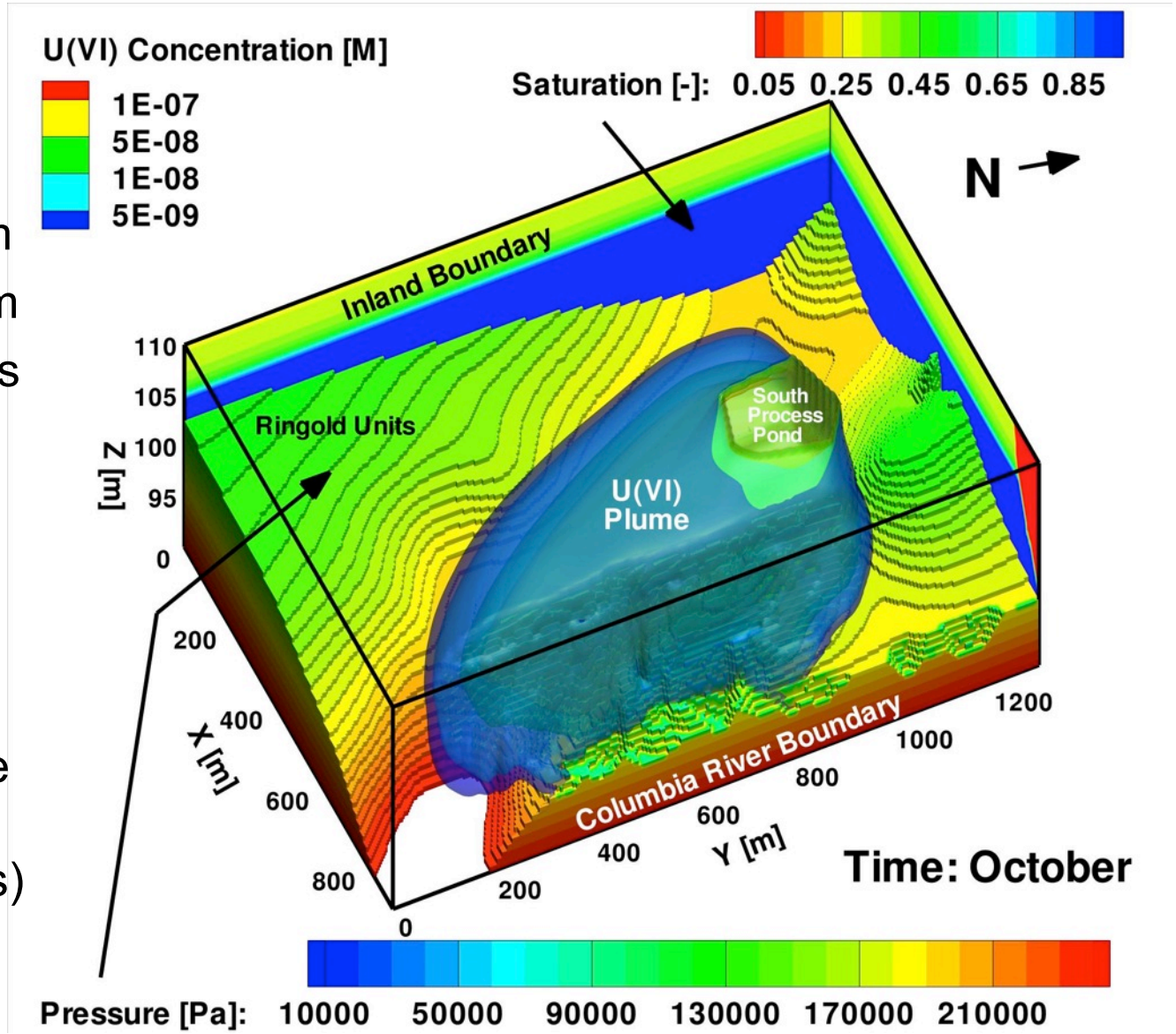
Slide courtesy of G. Hammond (PNNL)



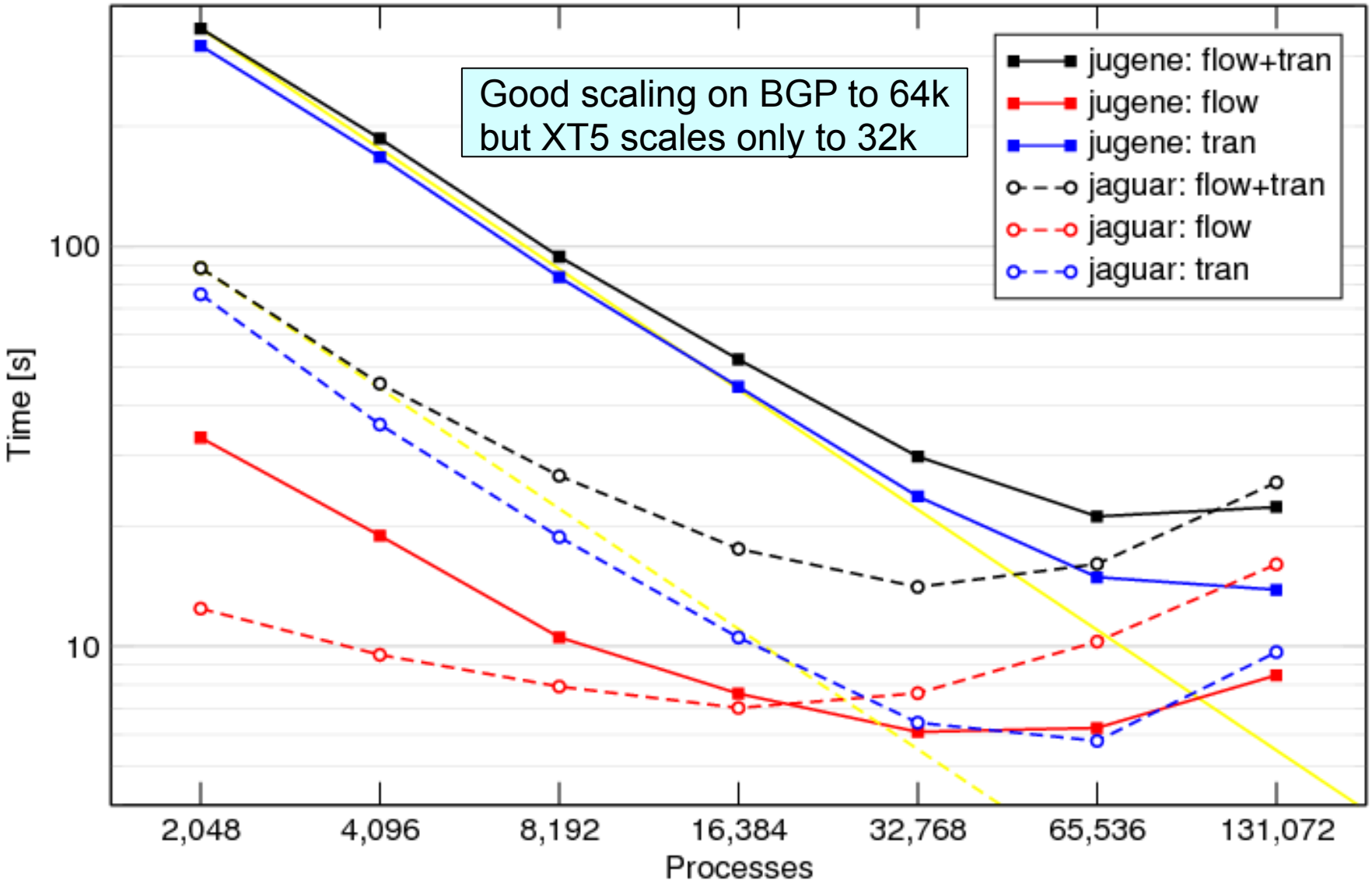
PFLOTRAN simulation of U(VI) migration

Slide courtesy of G. Hammond (PNNL)

- Hanford 300 area
- Problem domain:
  - 900x1300x20m
  - grid $\Delta x/\Delta y$ = 5m
  - 1.87M grid cells
  - 15 chemical species
  - 28M DoF total
- 1-year simulation:
  - Δ t = 1 hour
  - 5-10 hr runtime on Cray XT5 (using 4k cores)



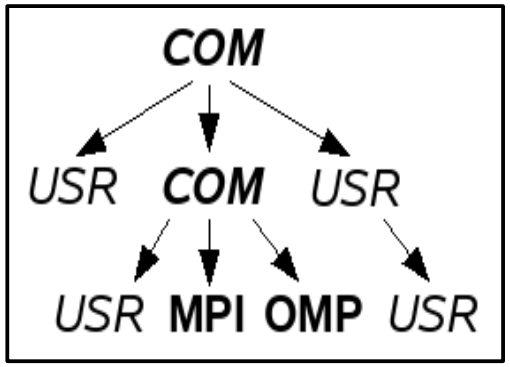
PFLOTRAN “2B” strong scalability



Scalasca usage with PFLOTRAN

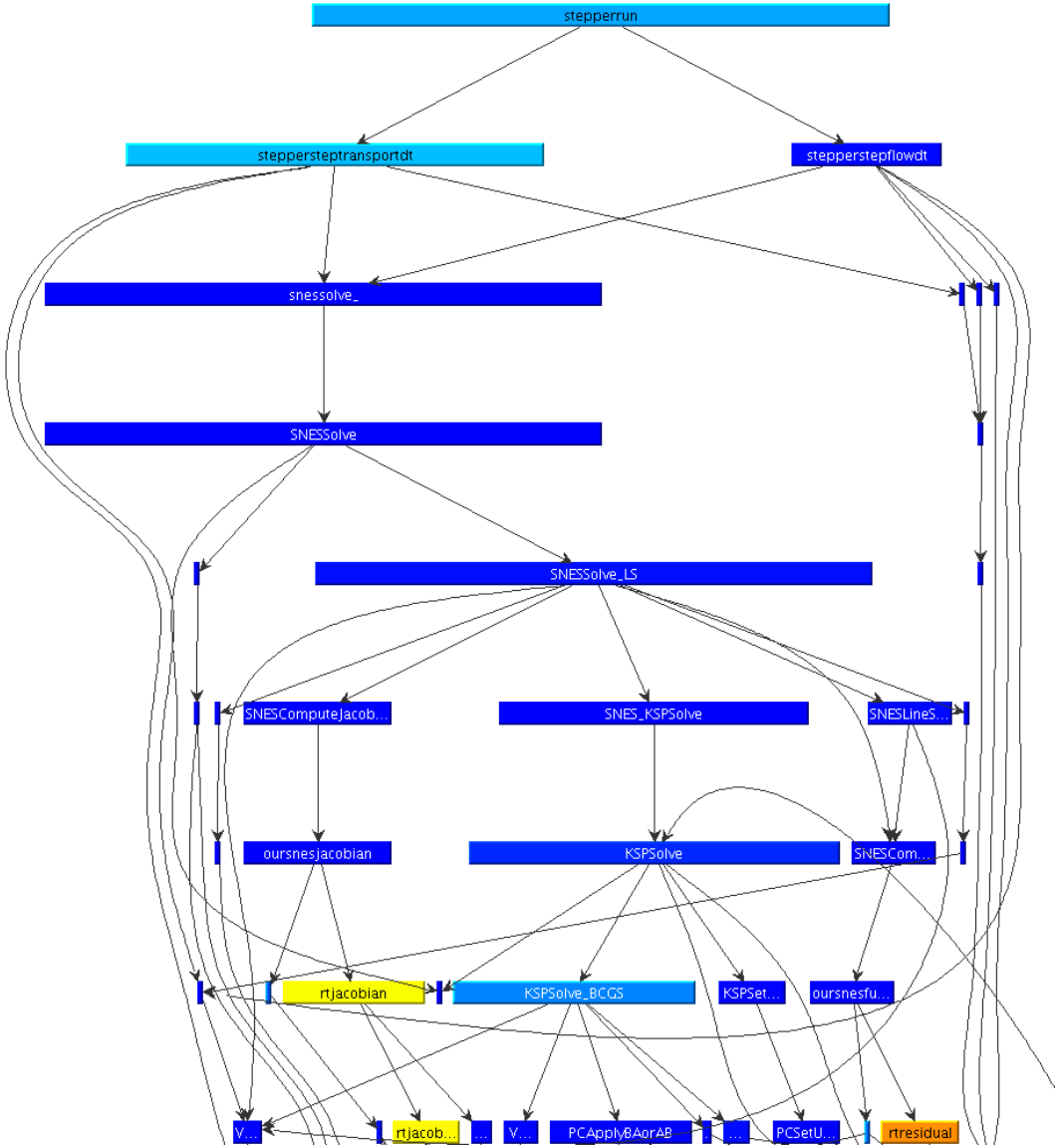
- Automatic instrumentation
  - User-level source routines instrumented by IBM XL compilers
  - Both PFLOTRAN application (Fortran) & PETSc lib (C)
  - (P)MPI routine interposition with instrumented library
- Initial summary measurements used to define filter file specifying all purely local computation routines
- Summary and trace measurement collected using filter
  - Parallel trace analysis initiated automatically on same partition
- Post-processing of analysis reports
  - Cut to extract *stepperrun* time-step loop
  - incorporation of application's 3D grid topology
- Analysis report examination in interactive GUI

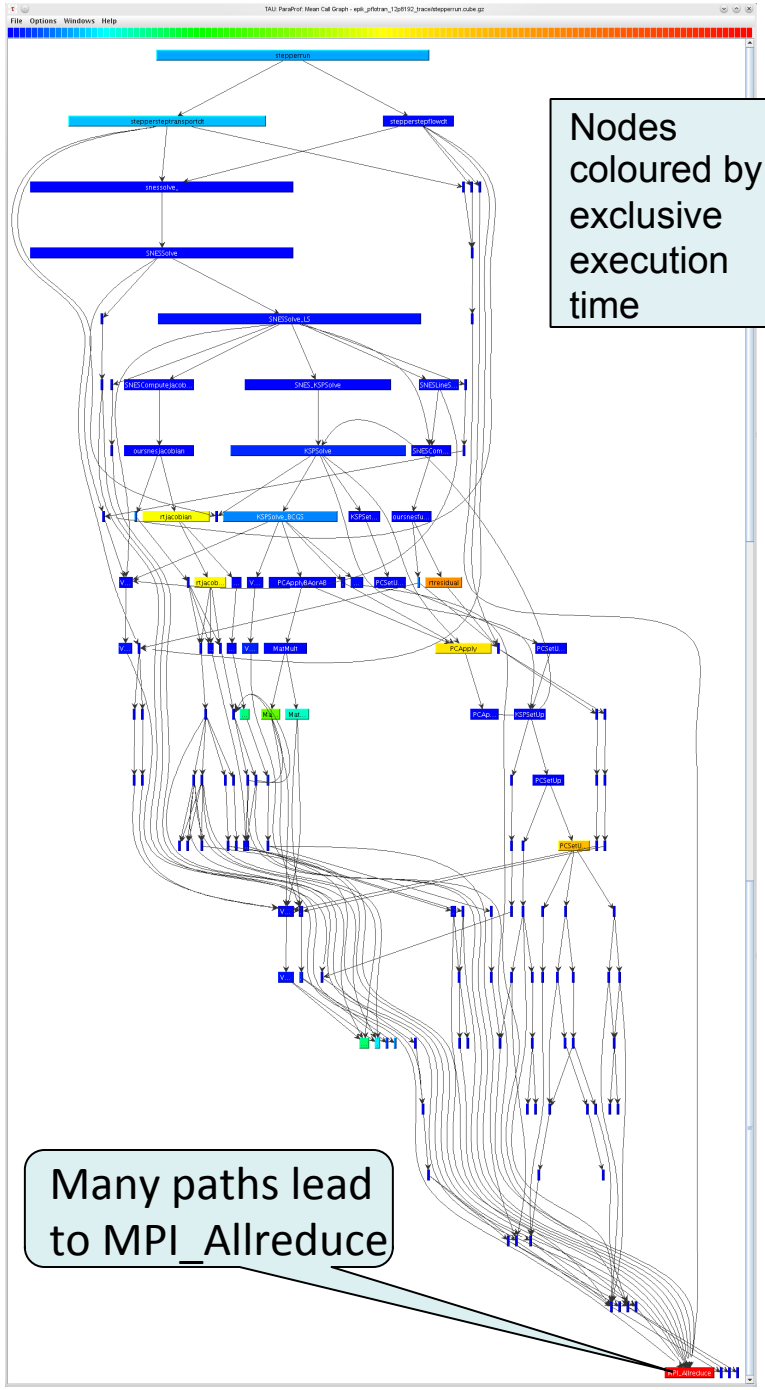
Scalasca usage with PFLOTRAN

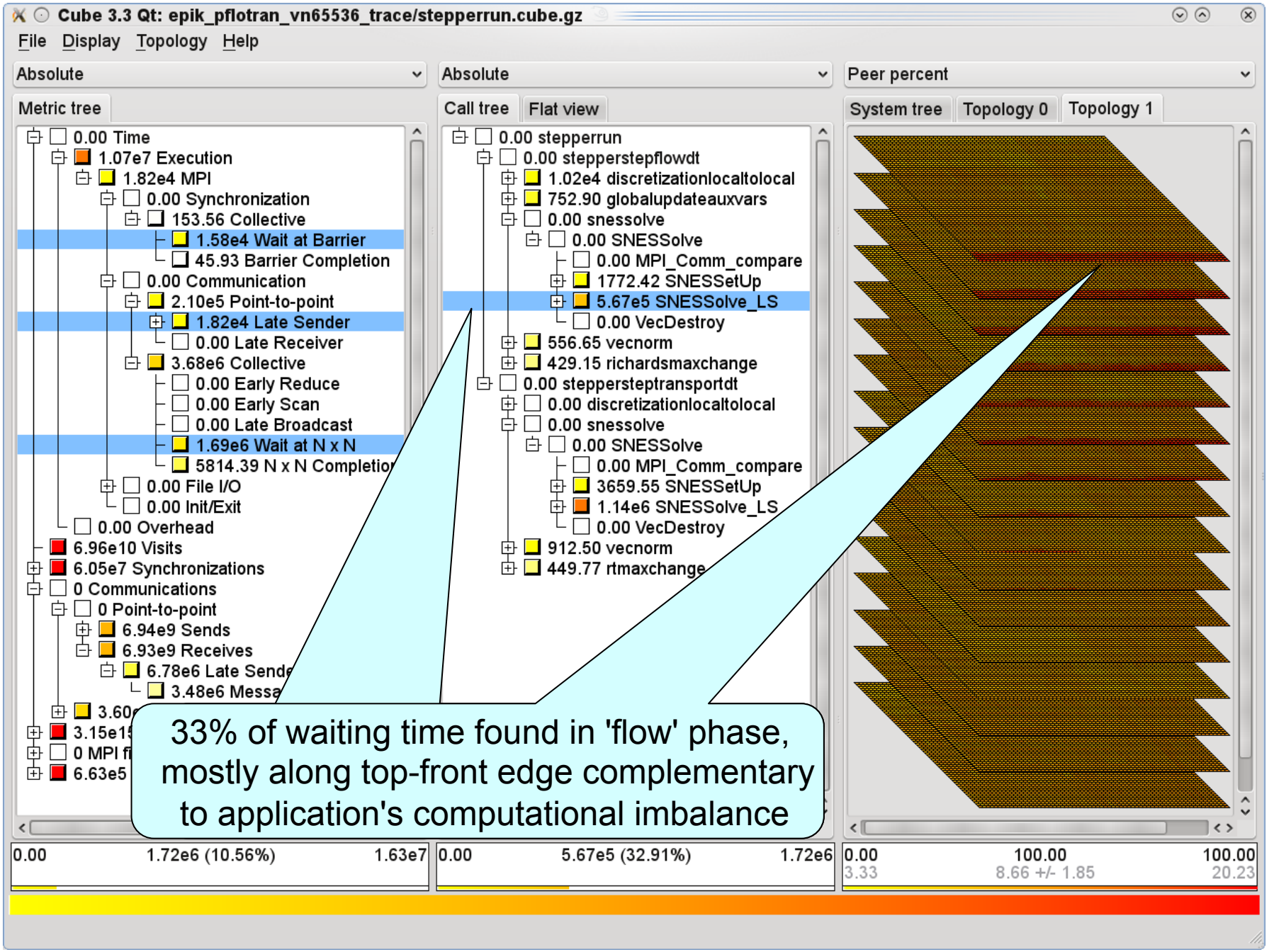


- Determined by scoring summary experiment using fully-instrumented application executable
  - 29 MPI library routines used
  - 1100+ PFLOTRAN & PETSc routines
    - most not on a callpath to MPI, purely local calculation (USR)
    - ~250 on callpaths to MPI, mixed calculation & comm. (COM)
  - Using measurement filter listing all USR routines
    - maximum callpath depth 22 frames
    - ~1750 unique callpaths (399 in FLOW, 375 in TRAN)
    - 633 MPI callpaths (121 in FLOW, 114 in TRAN)
  - FLOW callpaths very similar to TRAN callpaths

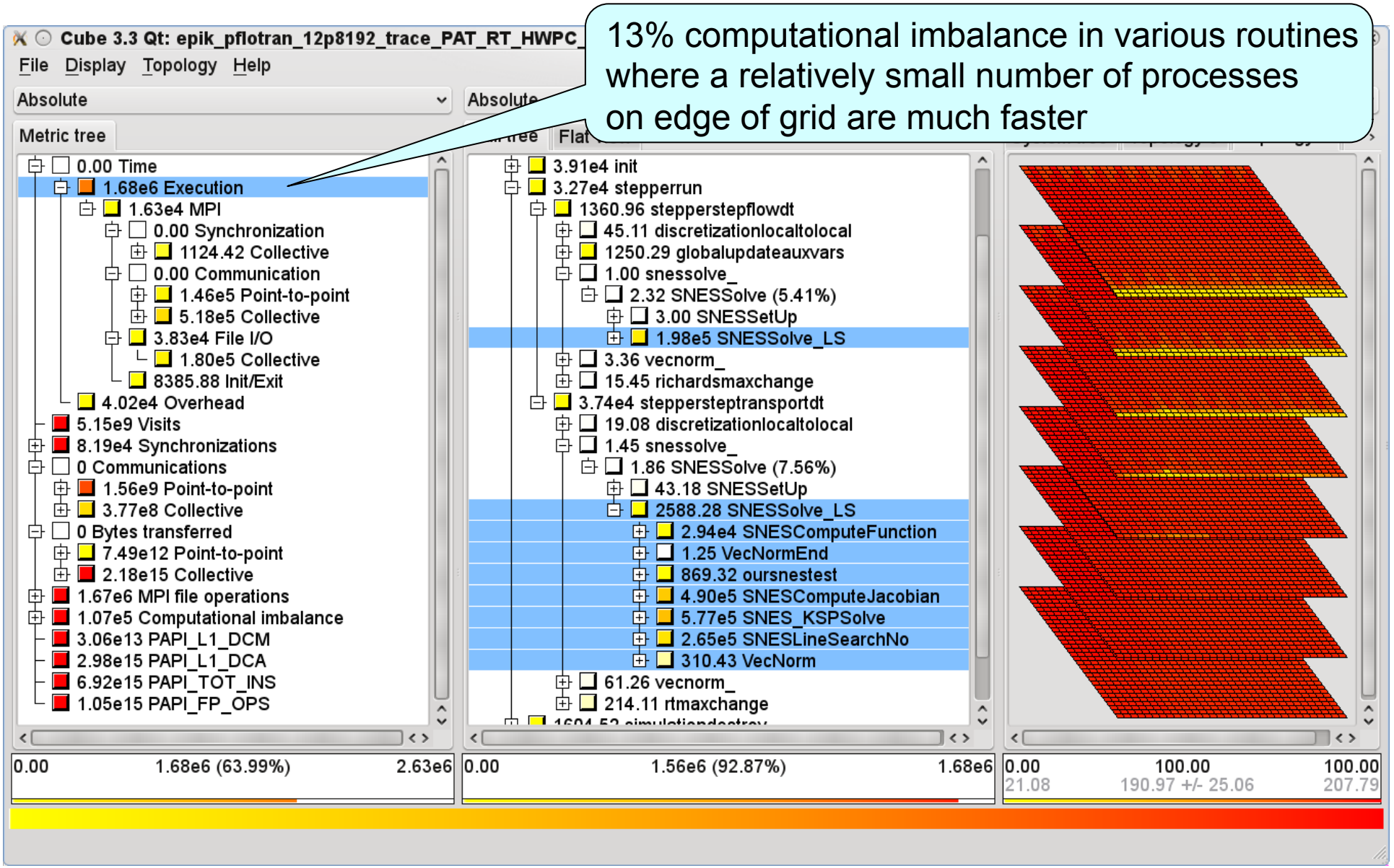
PFLOTRAN *stepperrun* graph



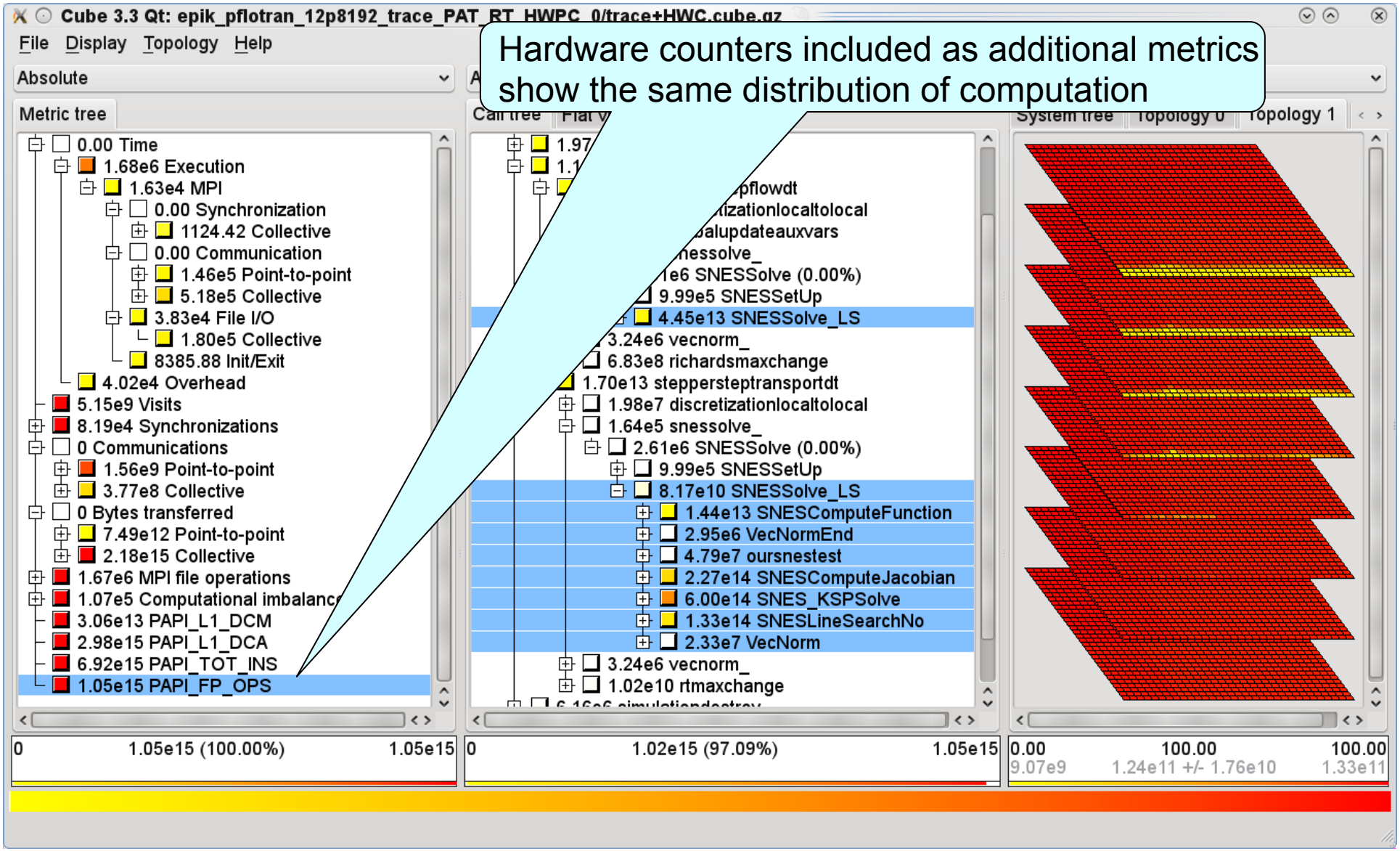



Cube 3.3 Qt: epik_pflotran_vn65536_trace/stepperrun.cube.gz
File Display Topology Help
Absolute
Absolute
Peer percent
Metric tree
Call tree
Flat view
System tree
Topology 0
Topology 1
0.00 Time
1.07e7 Execution
1.82e4 MPI
0.00 Synchronization
153.56 Collective
1.58e4 Wait at Barrier
45.93 Barrier Completion
0.00 Communication
2.10e5 Point-to-point
1.82e4 Late Sender
0.00 Late Receiver
3.68e6 Collective
0.00 Early Reduce
0.00 Early Scan
0.00 Late Broadcast
1.69e6 Wait at N x N
0.00 File I/O
0.00 Init/Exit
0.00 Overhead
6.96e10 Visits
6.05e7 Synchronizations
0 Communications
0 Point-to-point
6.94e9 Sends
6.93e9 Receives
0.00 stepperrun
0.00 stepperstepflowdt
1.02e4 discretizationlocaltolocal
752.90 globalupdateauxvars
0.00 snessolve
0.00 SNESSolve
0.00 MPI_Comm_compare
1772.42 SNESSetUp
5.67e5 SNESSolve_LS
0.00 VecDestroy
556.65 vecnorm
429.15 richardsmaxchange
0.00 steppersteptransportdt
0.00 discretizationlocaltolocal
0.00 snessolve
0.00 SNESSolve
0.00 MPI_Comm_compare
3659.55 SNESSetUp
1.14e6 SNESSolve_LS
0.00 VecDestroy
912.50 vecnorm
449.77 rtmaxchange
33% of waiting time found in 'flow' phase, mostly along top-front edge complementary to application's computational imbalance
0.00 1.72e6 (10.56%) 1.63e7
0.00 5.67e5 (32.91%) 1.72e6
0.00 100.00 100.00
3.33 8.66 +/- 1.85 20.23

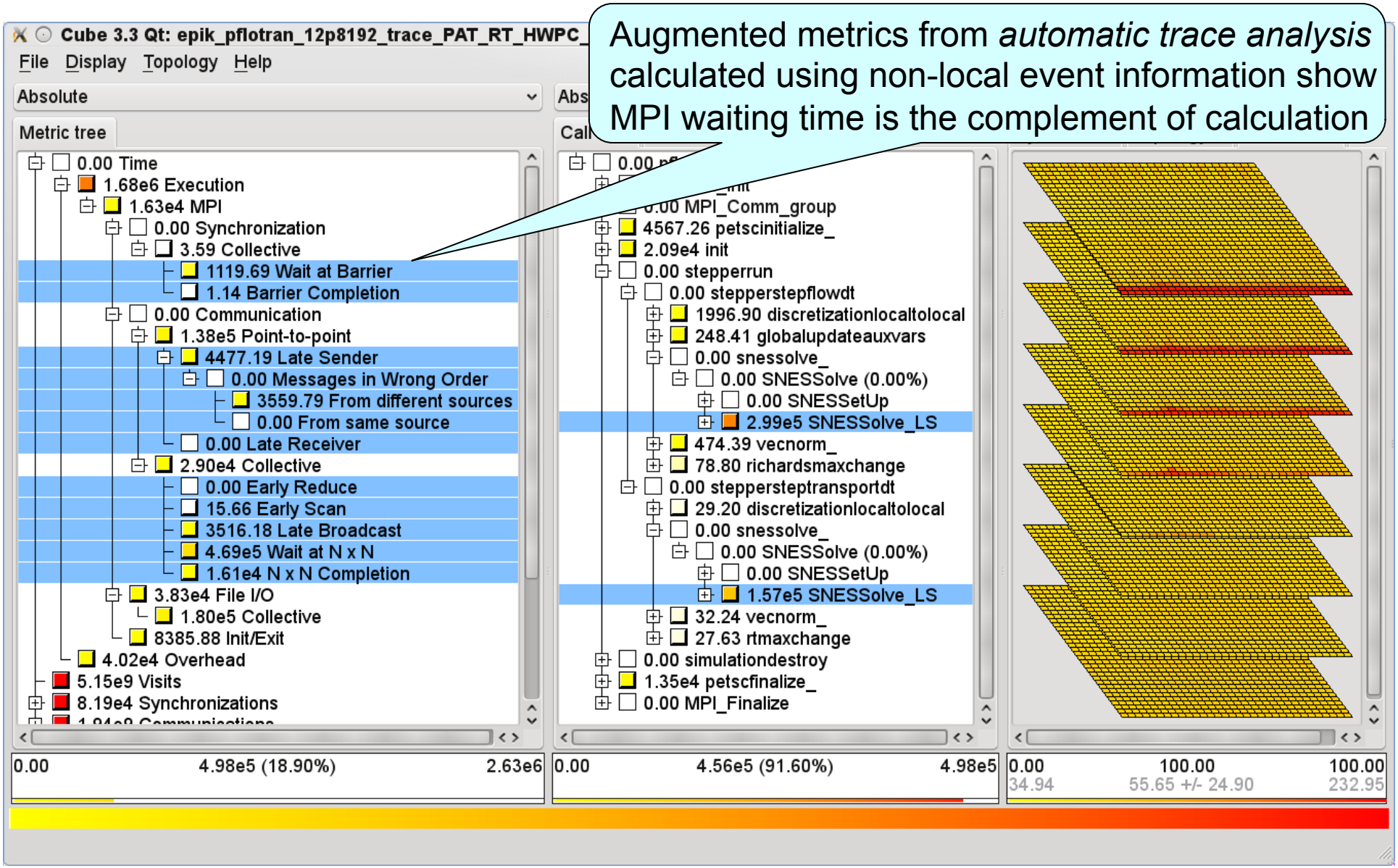
Exclusive execution time



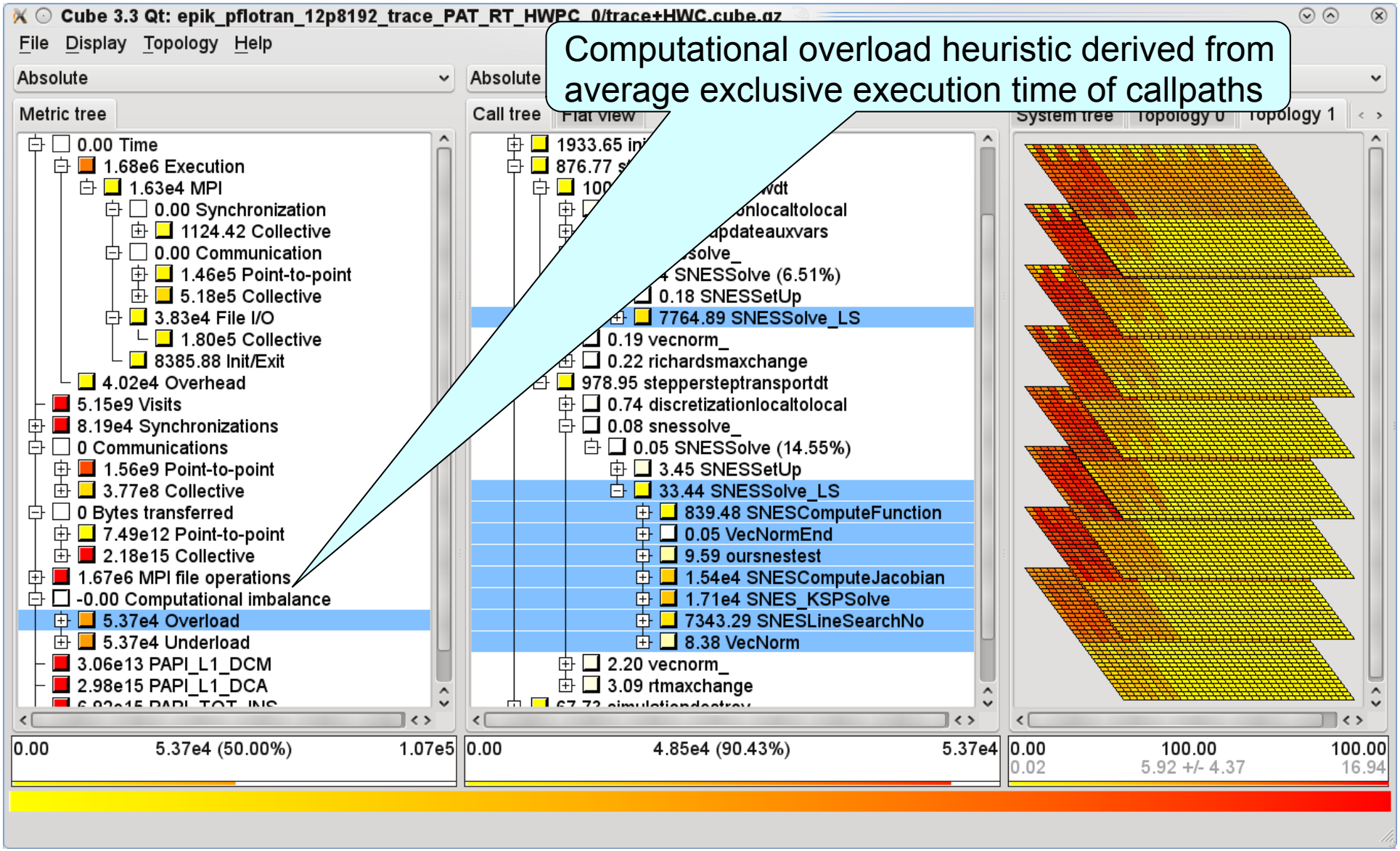
Floating-point operations



Scalasca trace analysis metrics (waiting time)



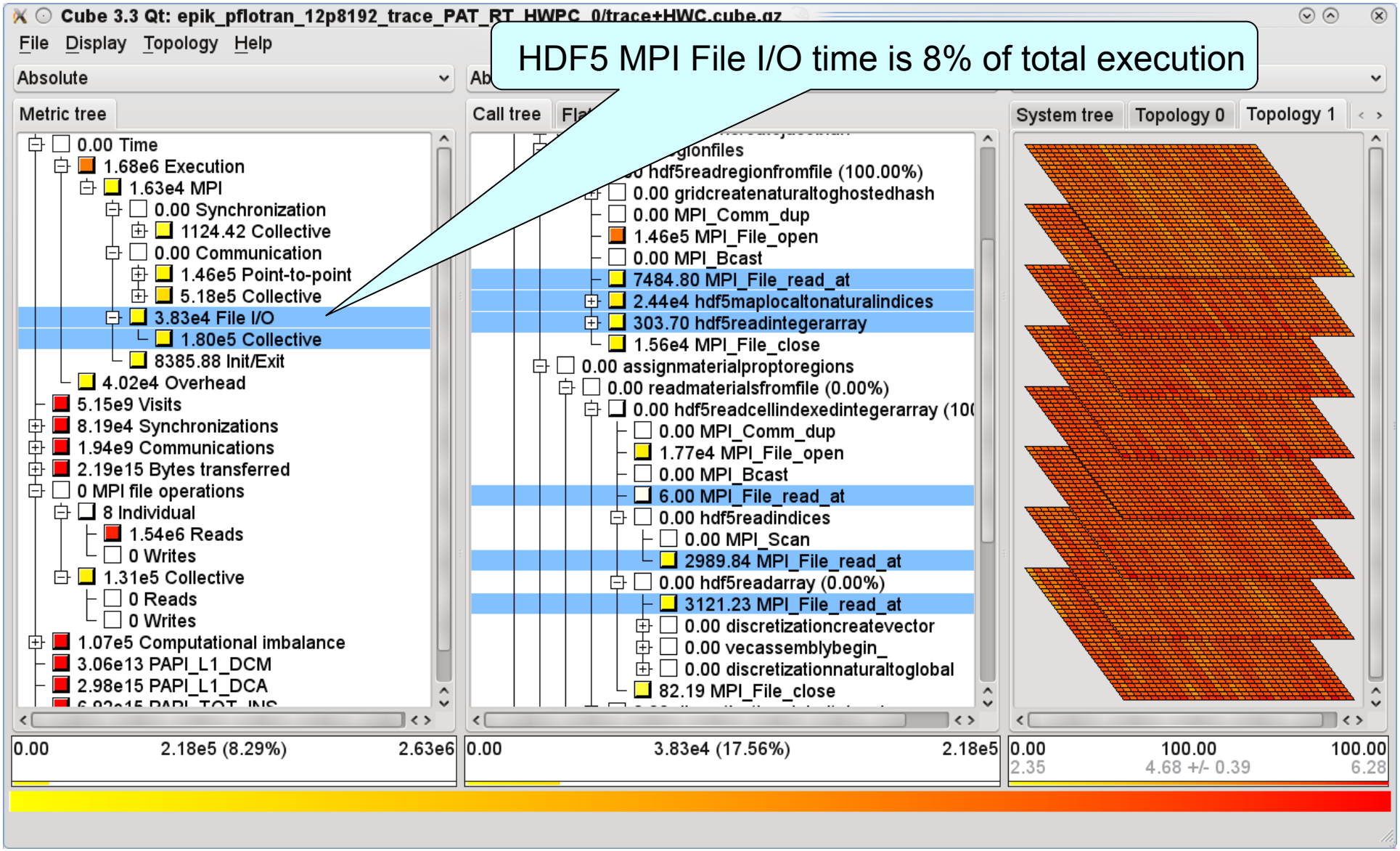
Computational imbalance: overload



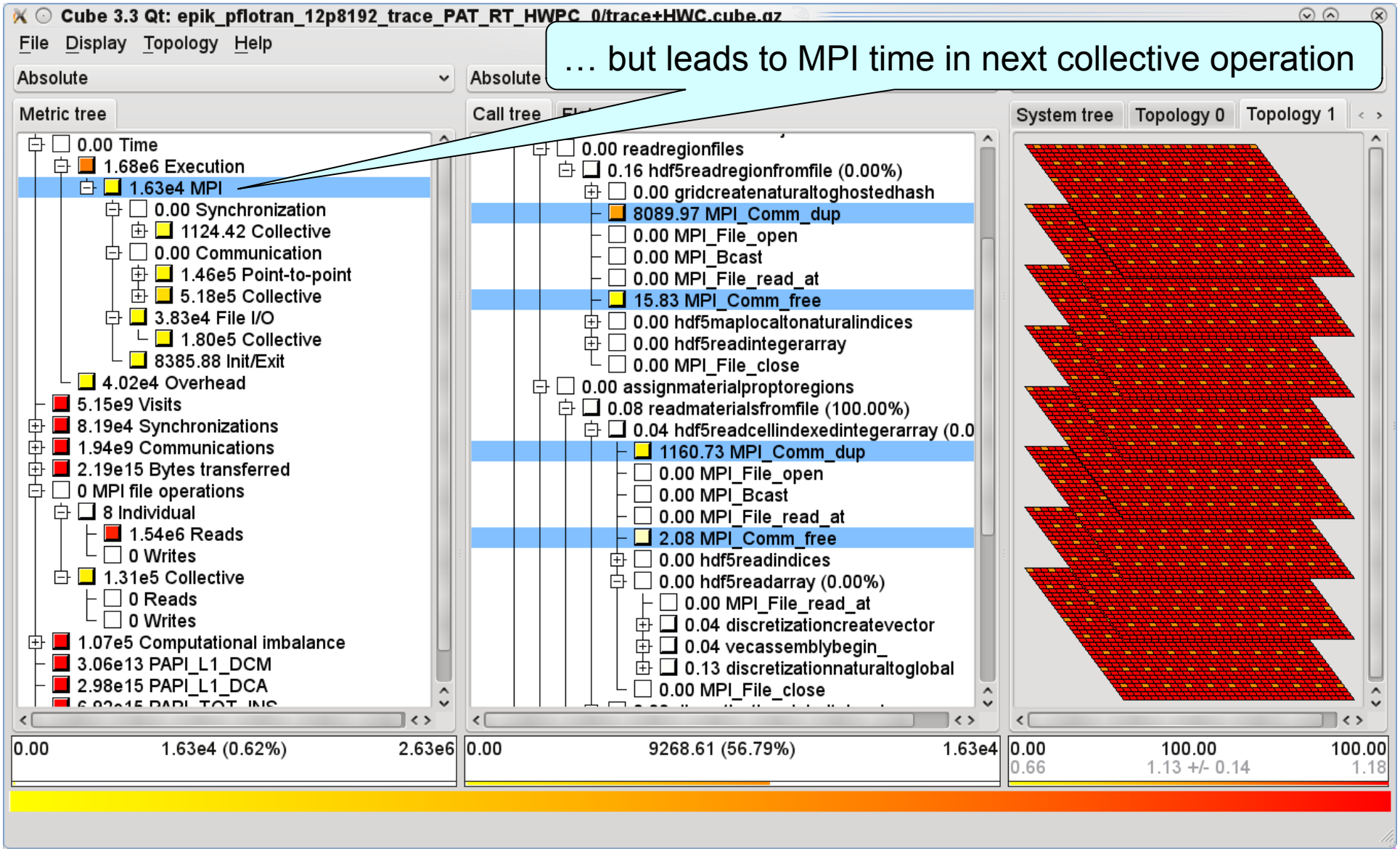
PFLOTRAN “2B” grid decomposition imbalance

- 850x1000x160 cells decomposed on 65536=64x64x16 process grid
  - x-axis: 850/64=13 plus 18 extra cells
  - y-axs: 1000/64=15 plus 40 extra cells
  - z-axis: 160/16=10
- Perfect distribution would be 2075.2 cells per process
  - but 20% of processes in each z-plane have 2240 cells
  - 8% computation overload manifests as waiting times at the next communication/synchronization on the other processes
- The problem-specific localized imbalance in the river channel is minor
  - reversing the assignments in the x-dimension won't help much since some of the z-planes have no inactive cells

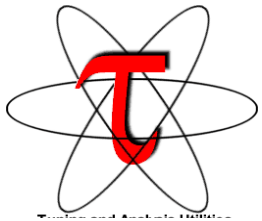
MPI File I/O time



MPI communicator duplication time



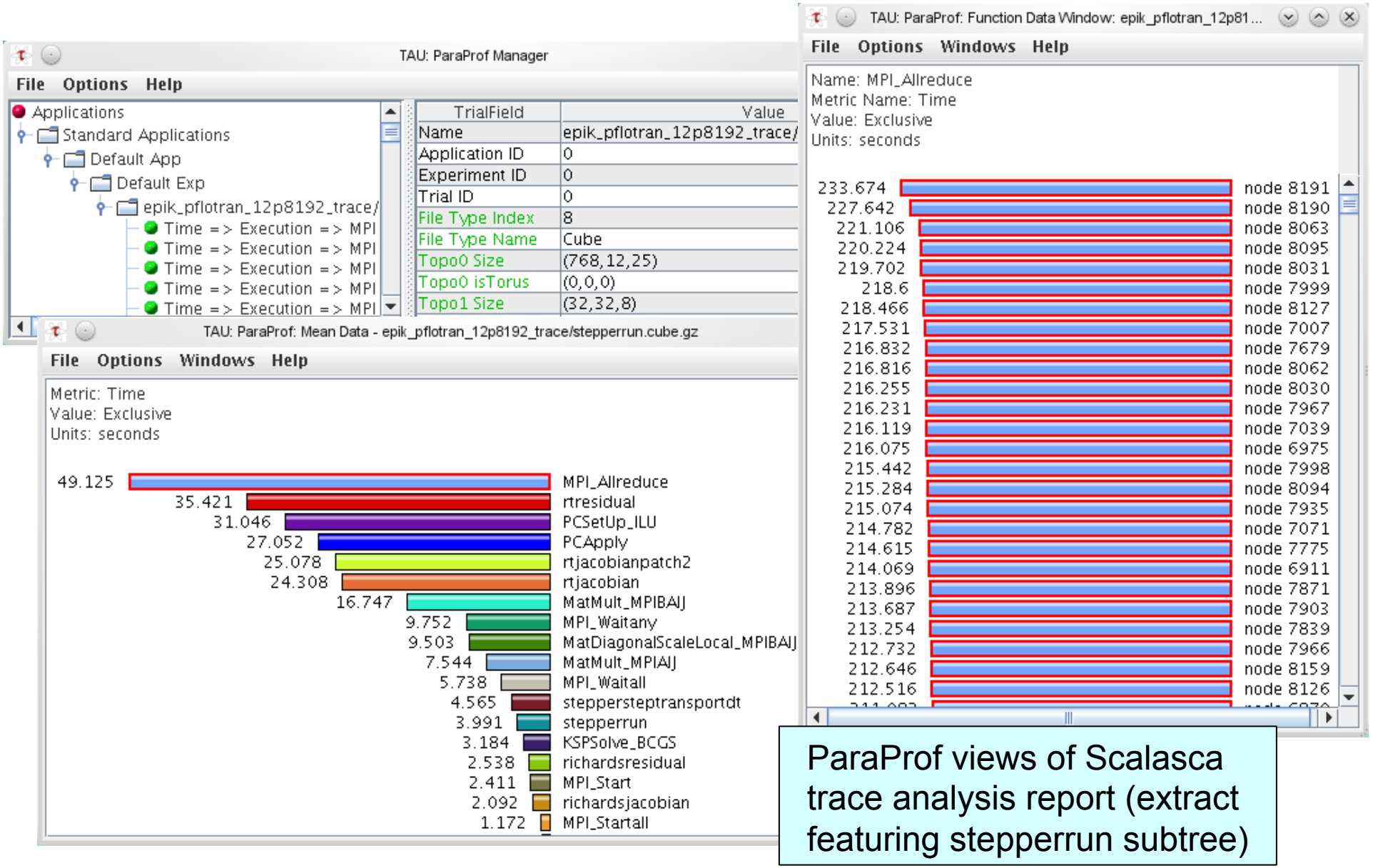
Complementary analyses & visualizations



- TAU/ParaProf can import Scalasca analysis reports
  - part of portable open-source TAU performance system
  - provides a callgraph display and various graphical profiles
  - may need to extract part of report to fit available memory

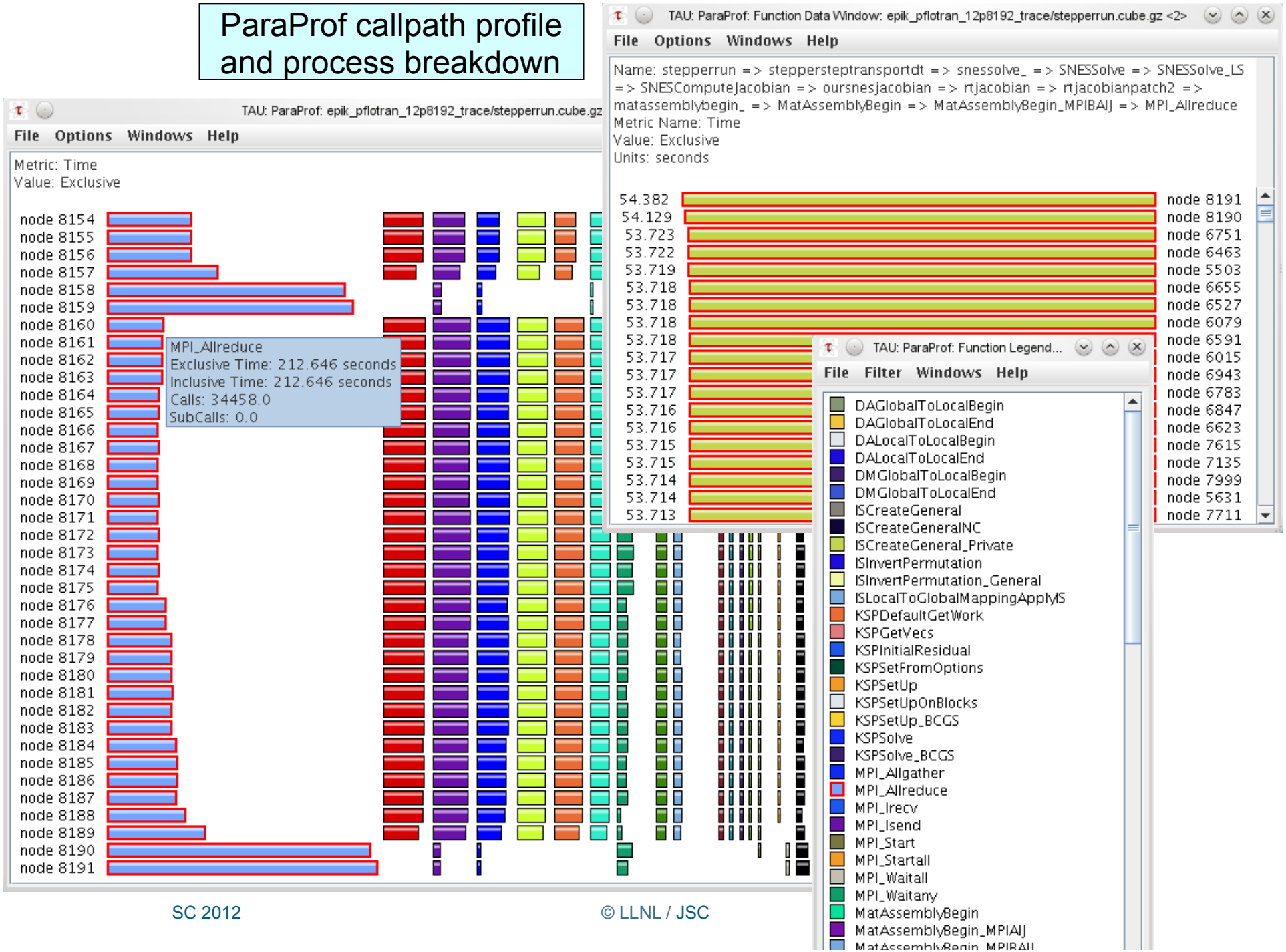
- Vampir 7 can visualize Scalasca distributed event traces
  - part of commercially-licensed Vampir product suite
  - provides interactive analysis & visualization of execution intervals
    - powerful zoom and scroll to examine fine detail
  - avoids need for expensive trace merging and conversion
    - required for visualization with Paraver & JumpShot
    - but often prohibitive for large traces

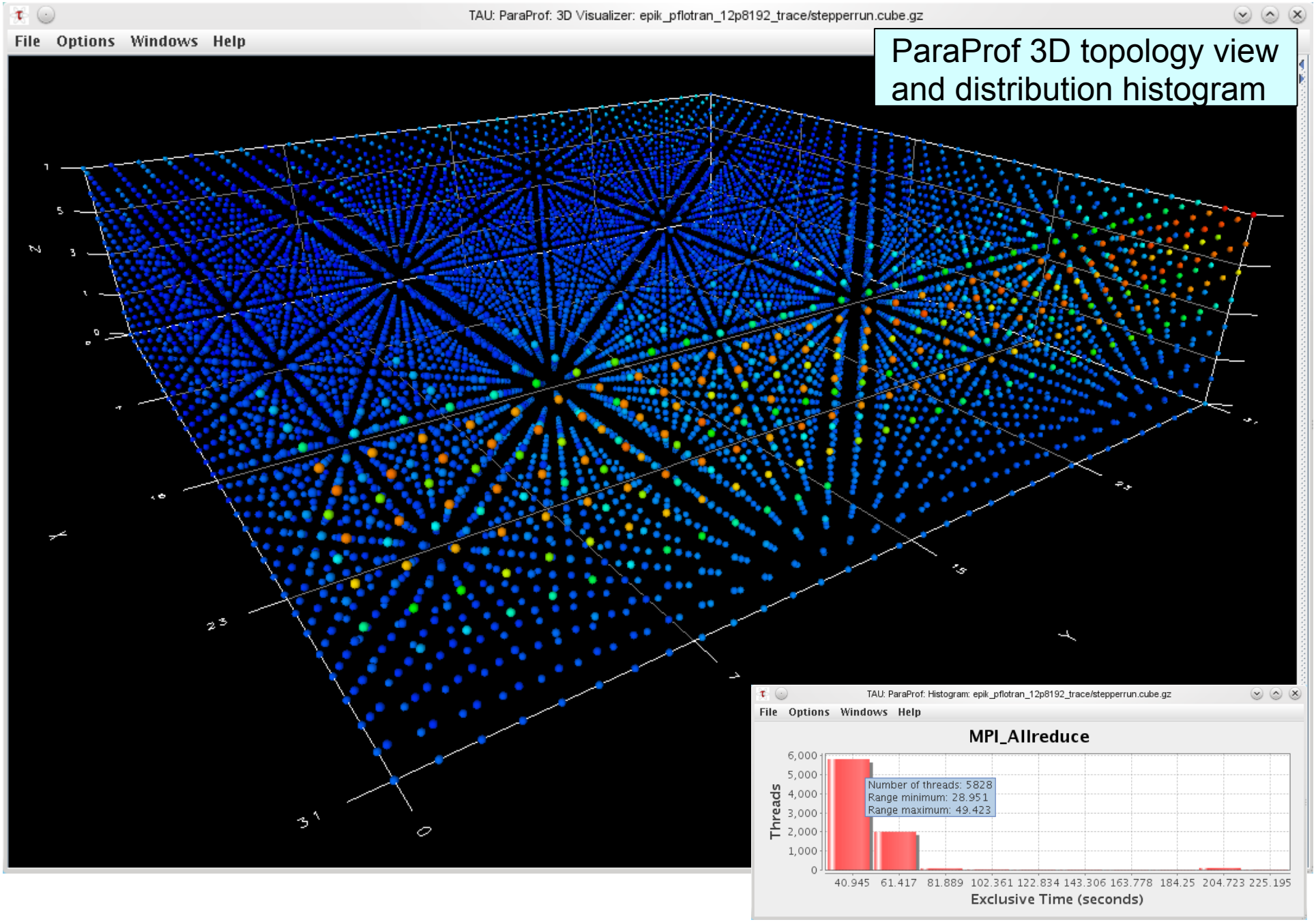
TAU/ParaProf graphical profiles



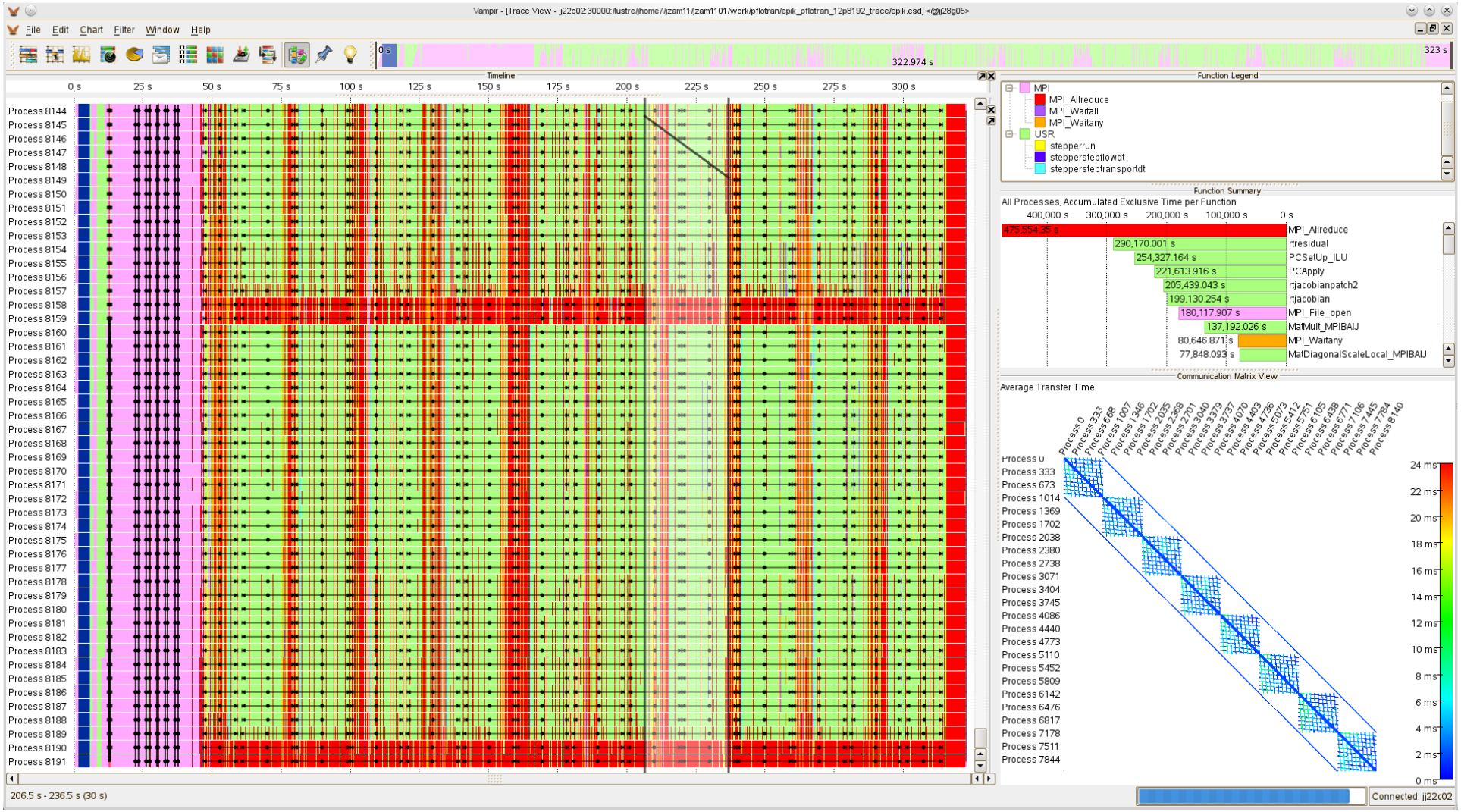
ParaProf views of Scalasca trace analysis report (extract featuring stepperrun subtree)

ParaProf callpath profile and process breakdown

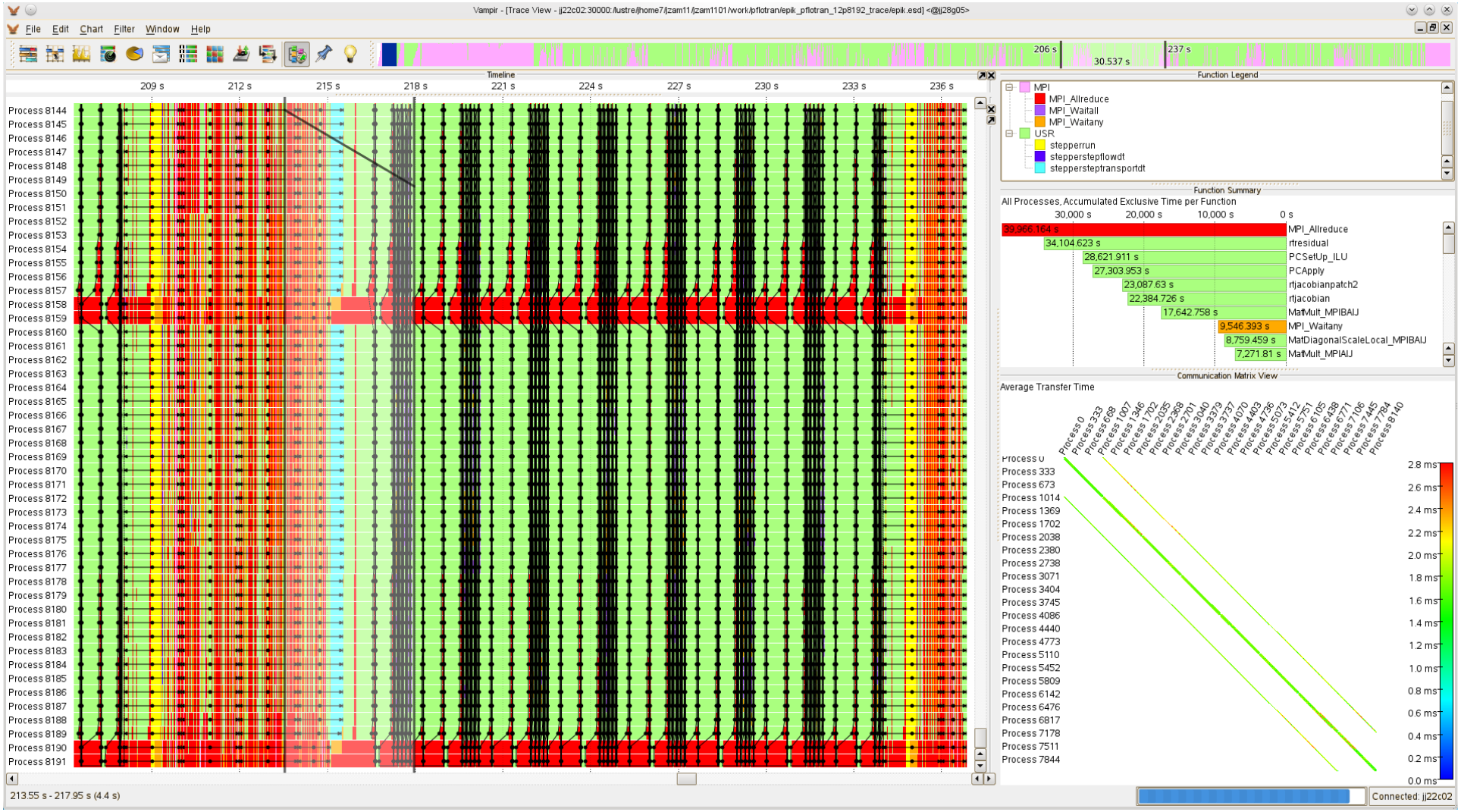



TAU: ParaProf: 3D Visualizer: epik_pflotran_12p8192_trace/stepperrun.cube.gz
File Options Windows Help
ParaProf 3D topology view and distribution histogram
TAU: ParaProf: Histogram: epik_pflotran_12p8192_trace/stepperrun.cube.gz
File Options Windows Help
MPI_Allreduce
Threads
6,000
5,000
4,000
3,000
2,000
1,000
0
Number of threads: 5828
Range minimum: 28.951
Range maximum: 49.423
40.945 61.417 81.889 102.361 122.834 143.306 163.778 184.25 204.723 225.195
Exclusive Time (seconds)

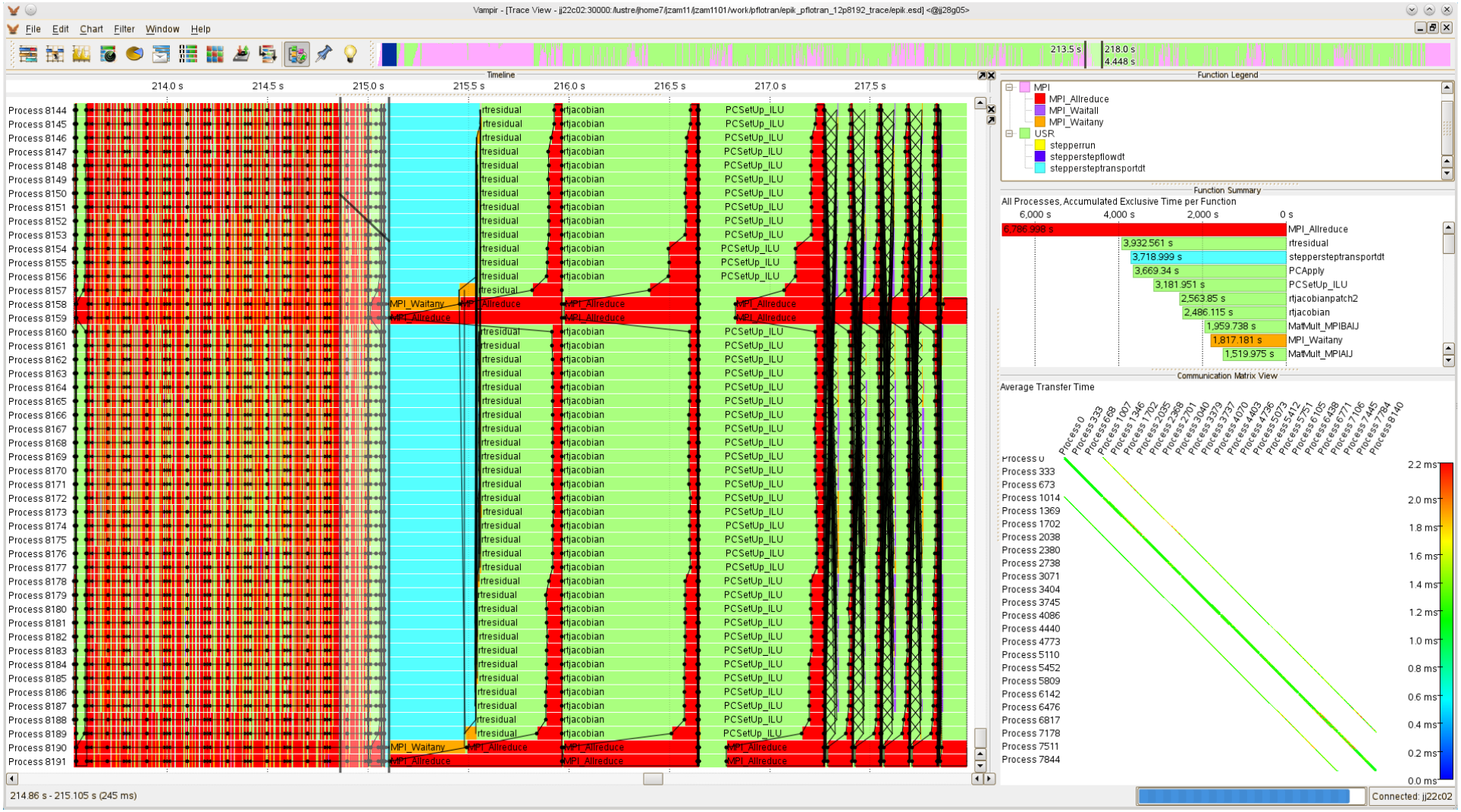
Vampir overview of execution (init + 10 steps)



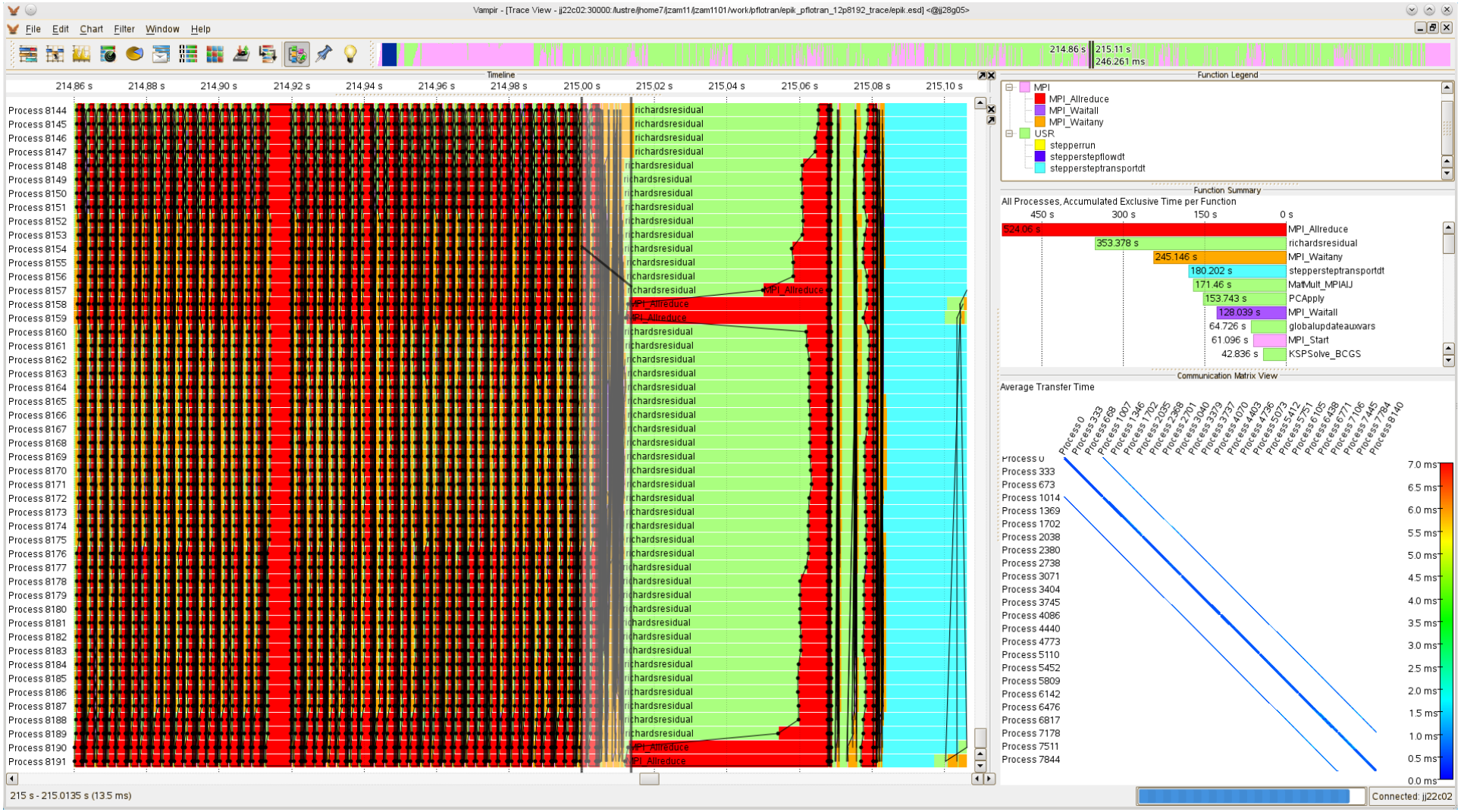
PFLOTRAN execution timestep 7 (flow & transport)



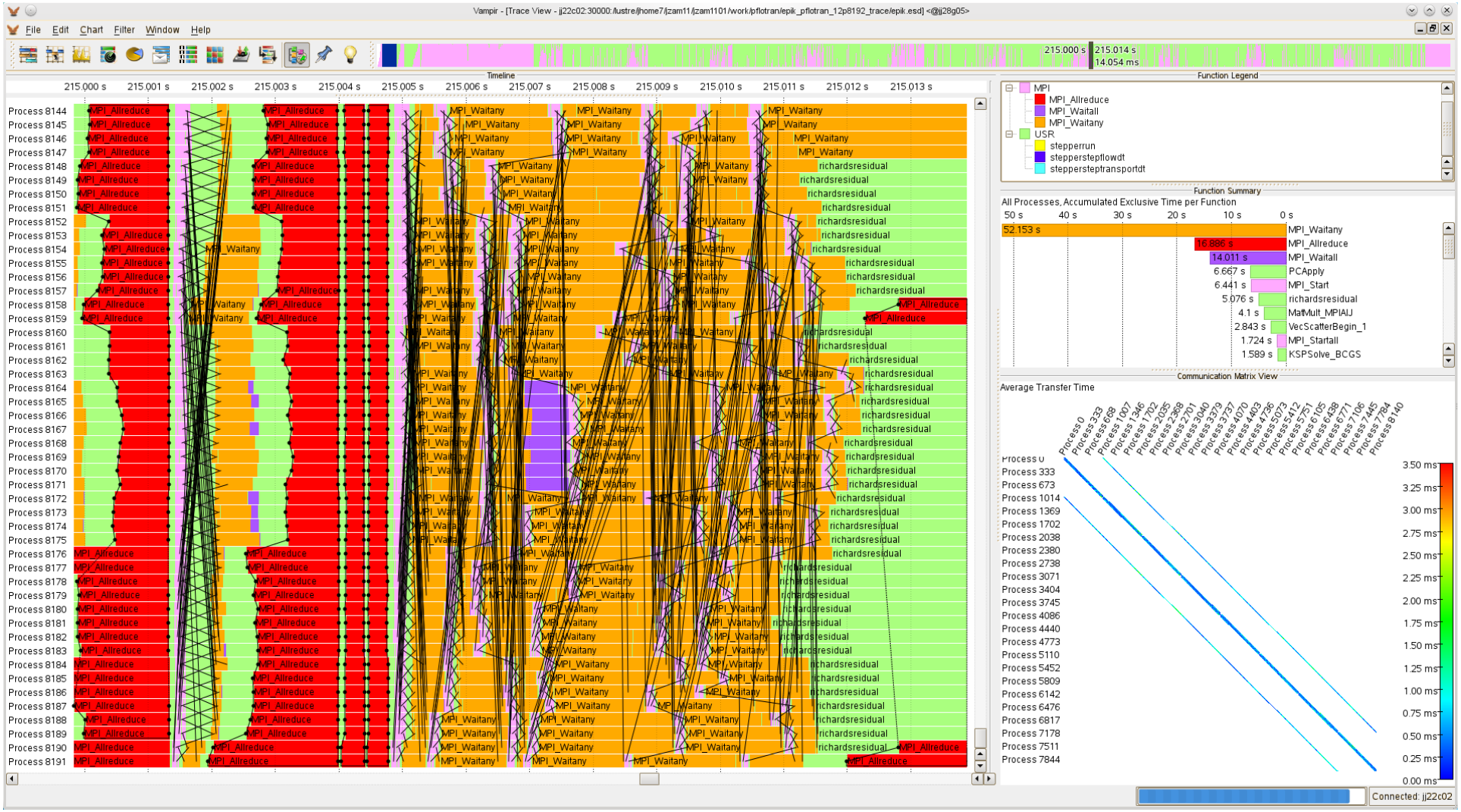
PFLOTRAN execution flow/transport transition



PFLOTRAN execution at end of flow phase



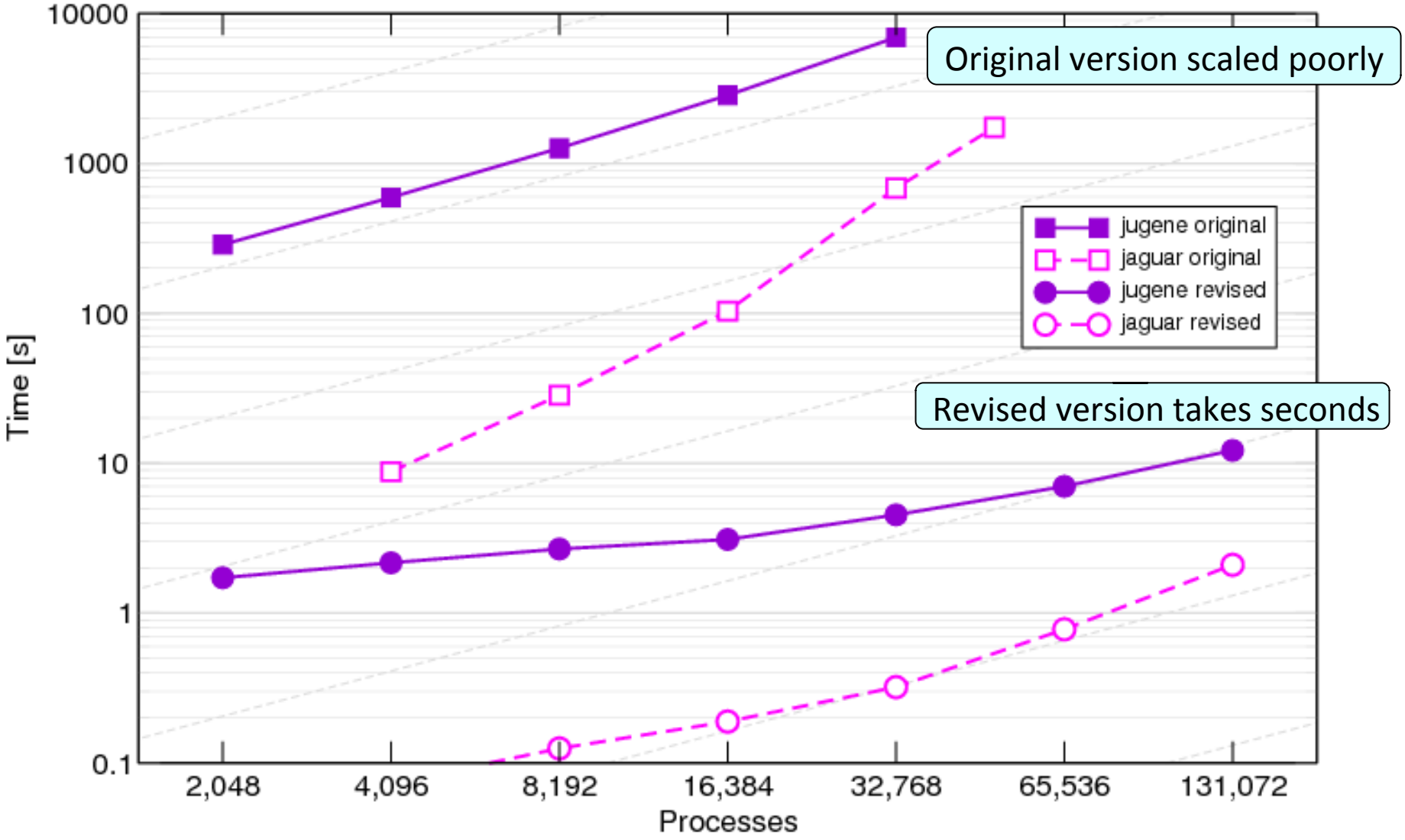
PFLOTRAN execution at end of flow phase detail



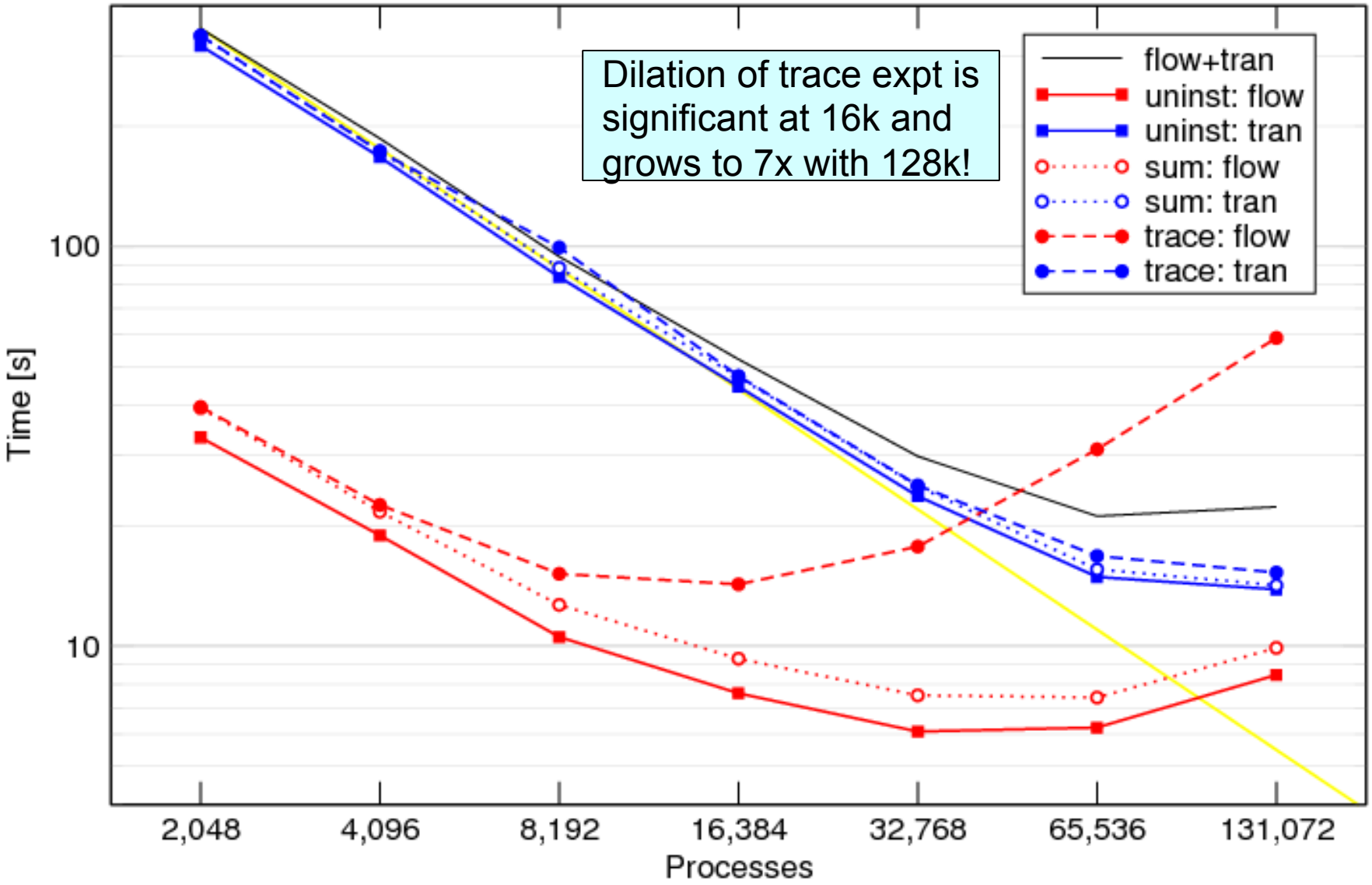
Scalasca scalability issues/optimizations (Part 2)

- PFLOTRAN (via PETSc & HDF5) uses lots of MPI communicators
  - 19 (MPI_COMM_WORLD) + 5xNPROCS (MPI_COMM_SELF)
    - time shows up in collective *MPI_Comm_dup*
  - Not needed for summarization, but essential for trace replay
    - definition storage & unification time grow accordingly
  - Original version of Scalasca failed with 48k processes
    - communicator definitions 1.42 GB, unification over 29 minutes
  - Revised version resolved scalability bottleneck
    - custom management of MPI_COMM_SELF communicators
    - hierarchical implementation of unification
  - …also eliminated growing tracing measurement dilation
    - avoid rank globalization using *MPI_Group_translate_ranks*
    - store local communicator ranks in trace event records
- Scalasca trace collection and analysis costs increase with scale

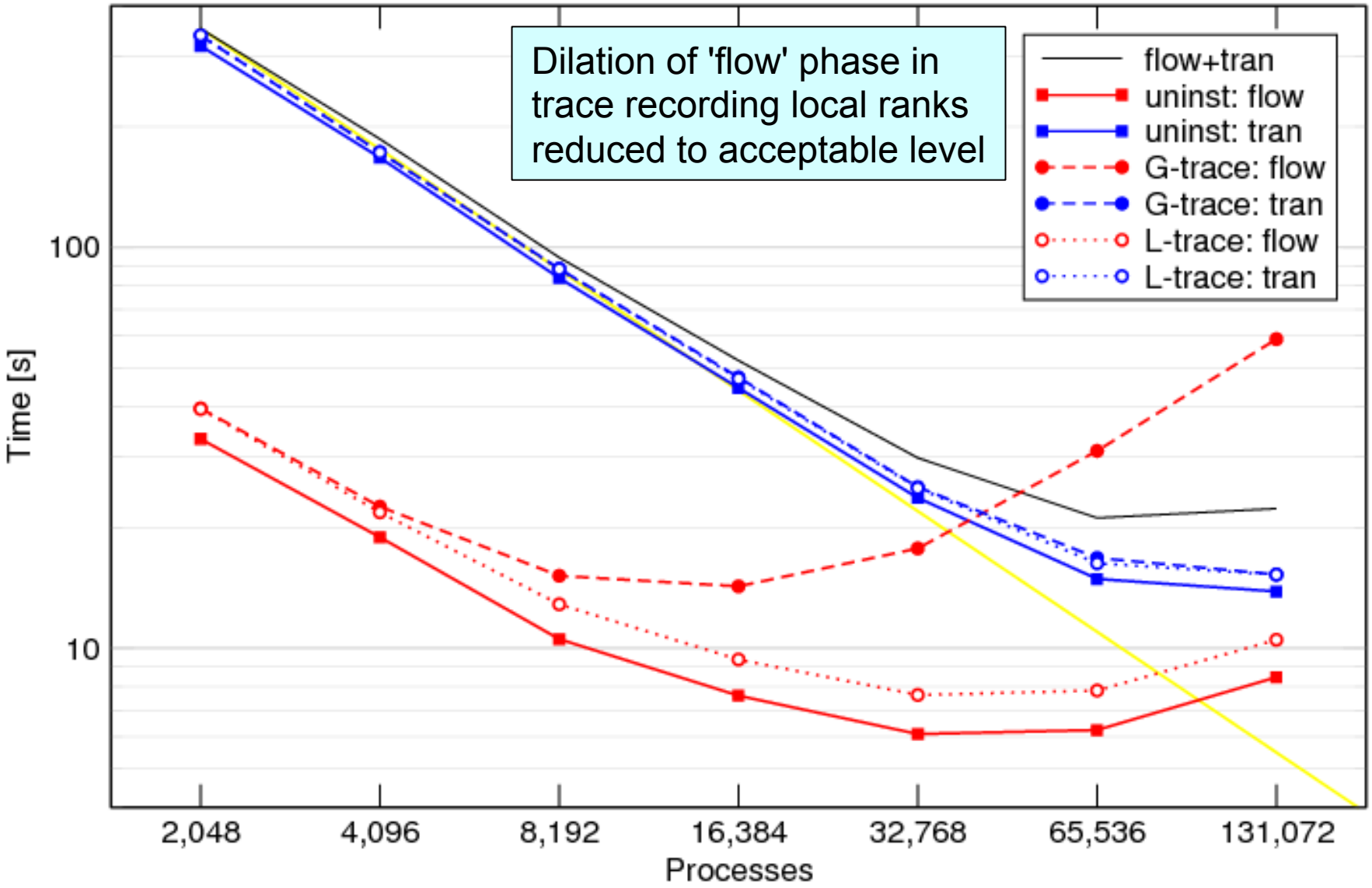
Improved unification of PFLOTRAN identifiers



PFLOTRAN measurement dilation (BG/P original)



Trace measurement dilation (BG/P)

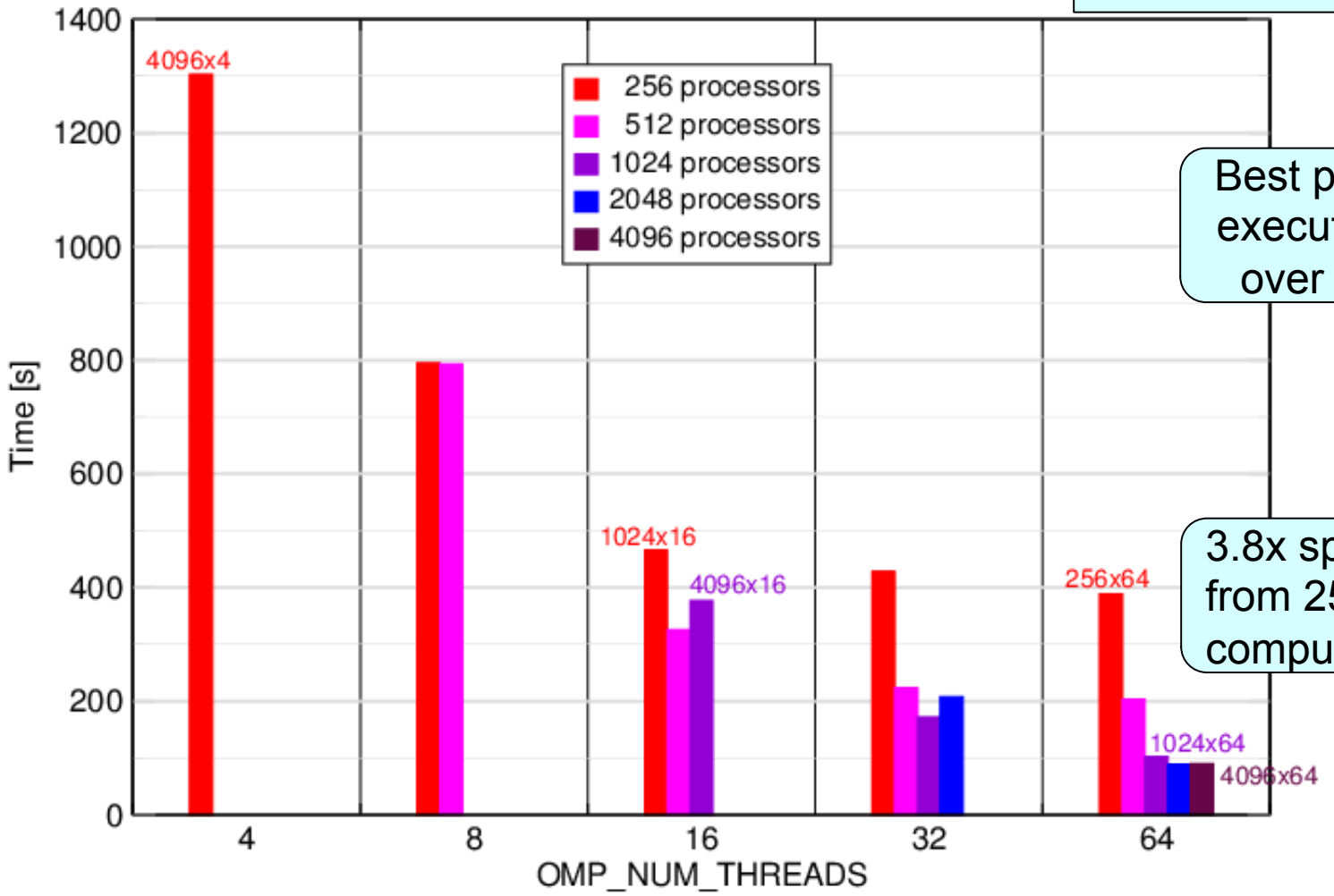


Scalasca case study 3: BT-MZ

- NPB benchmark code from NASA NAS
  - block triangular solver for unsteady, compressible Navier-Stokes equations discretized in three spatial dimensions
  - performs ADI for several hundred time-steps on a regular 3D grid and verifies solution error within acceptable limit
- Hybrid MPI+OpenMP parallel version (NPB3.3-MZ-MPI)
  - ~7,000 lines of code (20 source modules), mostly Fortran77
  - intra-zone computation with OpenMP, inter-zone with MPI
    - only master threads perform MPI (outside parallel regions)
  - very portable, and highly scalable
  - configurable for a range of benchmark classes and sizes
  - dynamic thread load balancing disabled to avoid oversubscription
- Run on *juqueen* BG/Q with up to 524,288 threads (8 racks)
  - Summary and trace analysis using Scalasca
  - Selective instrumentation by compiler & OPARI source processor

BT-MZ.E scaling analysis (BG/Q)

NPB class E problem:
64 x 64 zones
4224 x 3456 x 92 grid



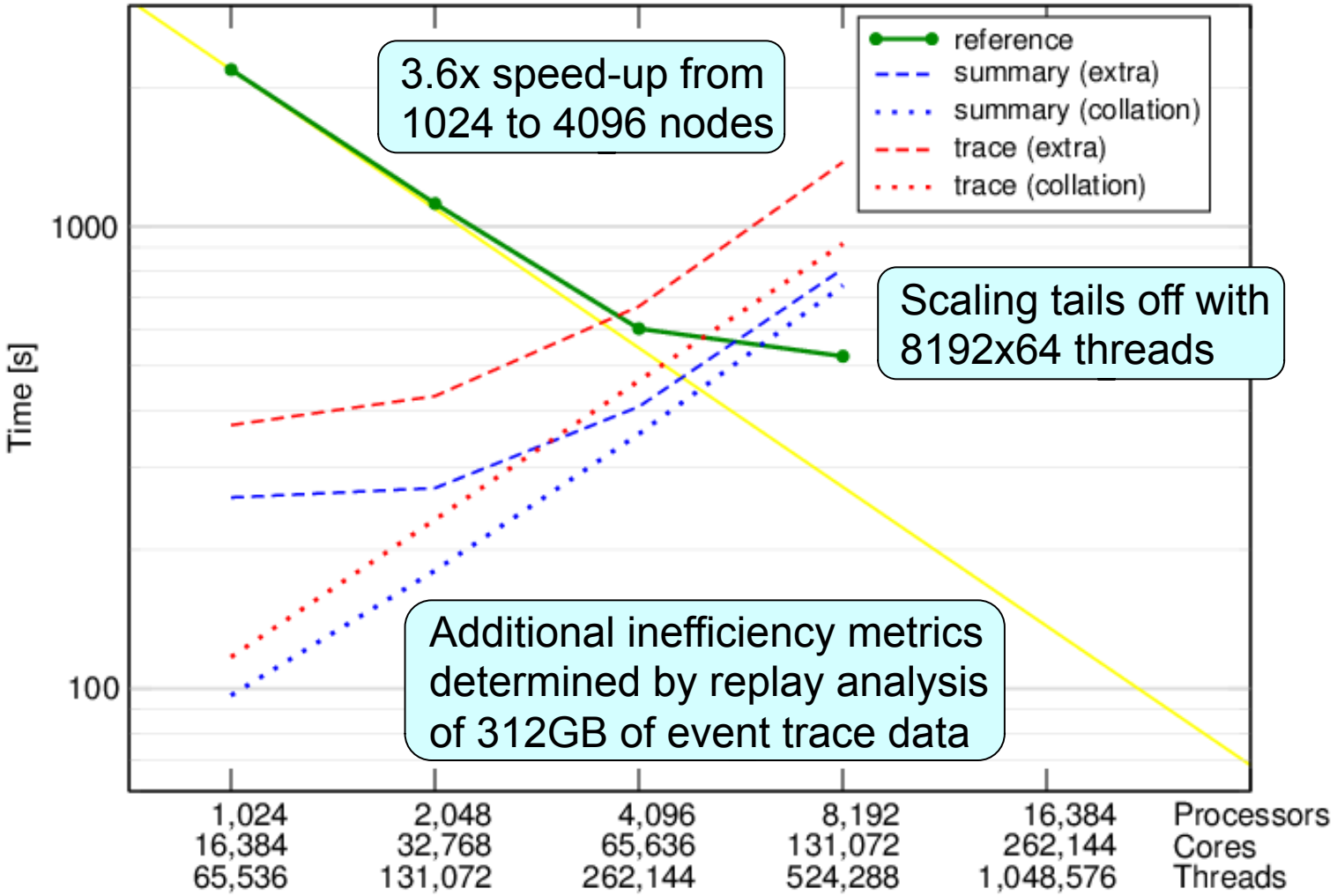
Best pure MPI execution time over 2000s

3.8x speed-up from 256 to 1024 compute nodes

- Best performance with 64 OpenMP threads per MPI process
- 55% performance bonus from exploiting all 4 hardware threads

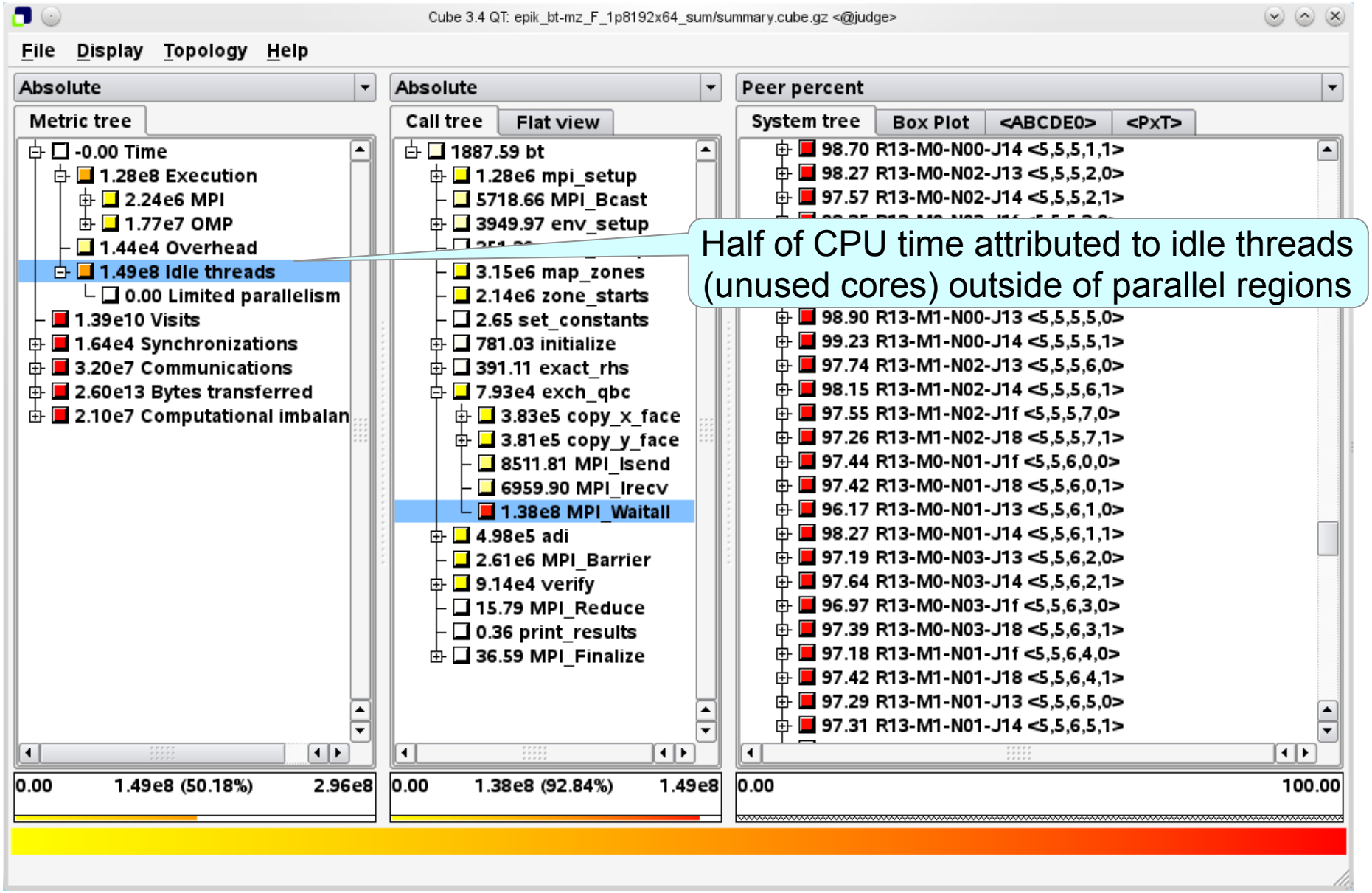
BT-MZ.F scaling analysis (BG/Q)

NPB class F problem:
128 x 128 zones
12032 x 8960 x 250 grid

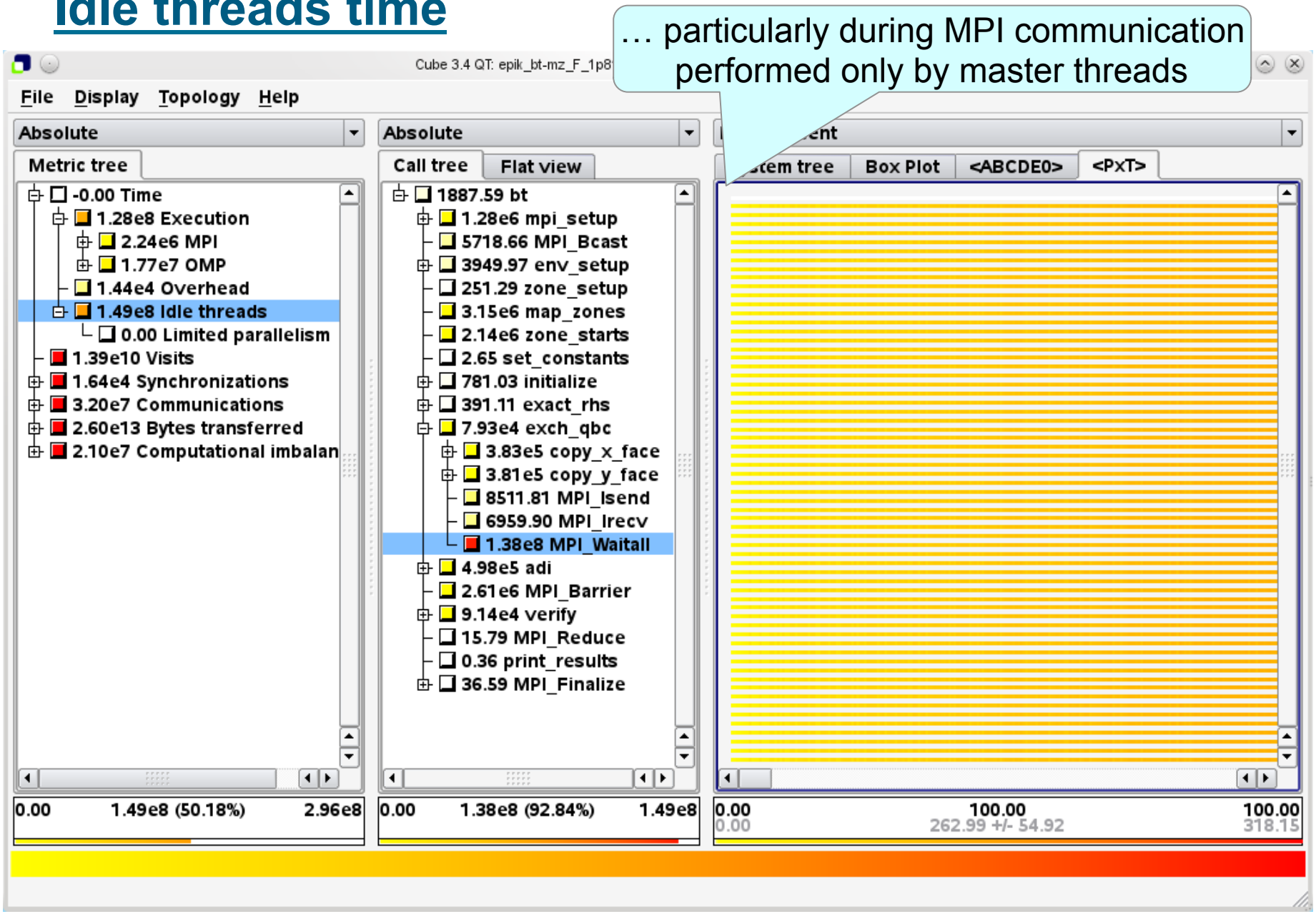


- Negligible dilation of Scalasca measurements, however, extra costs for analysis report collation proportional to total number of threads

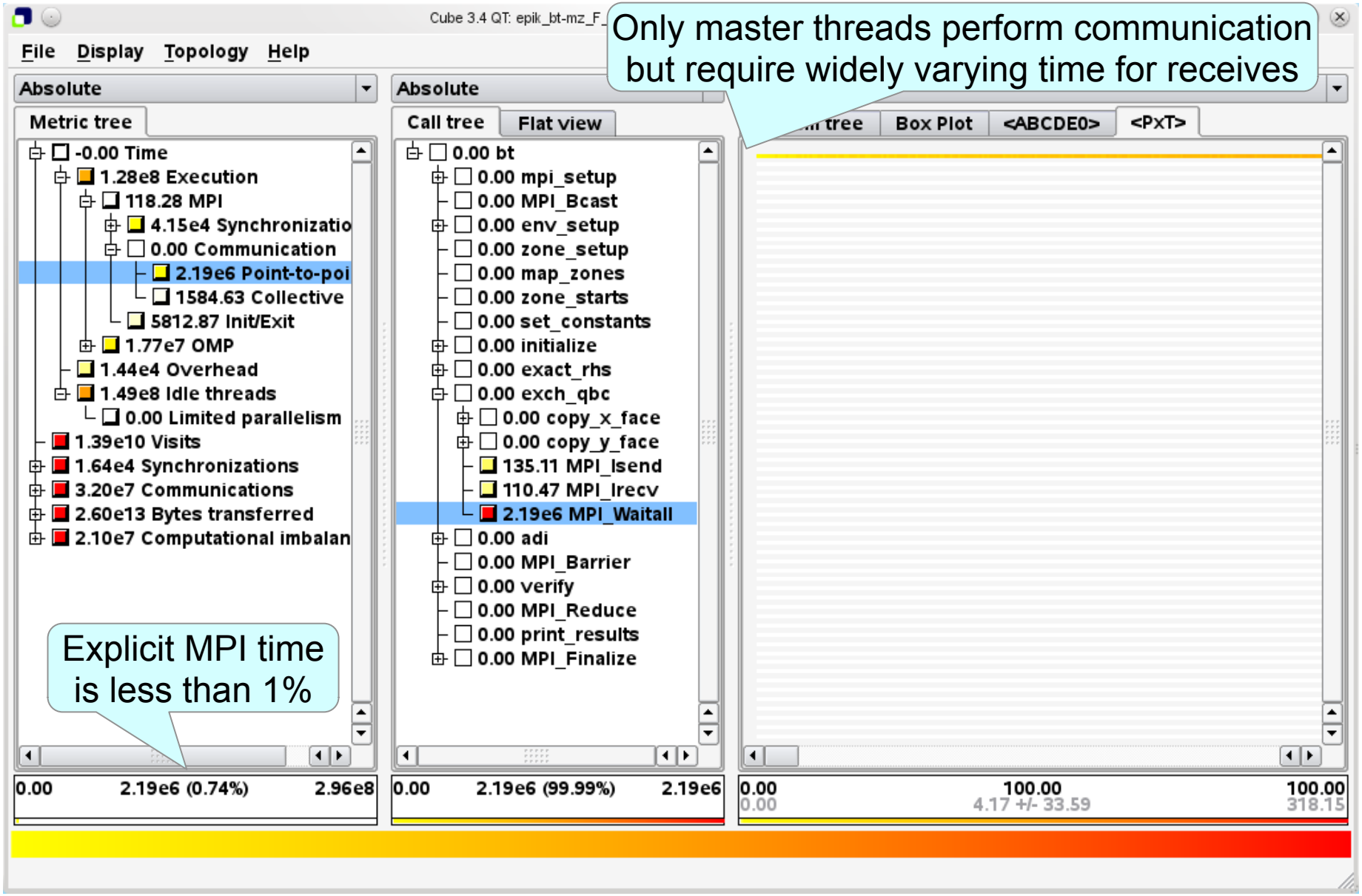
Idle threads time



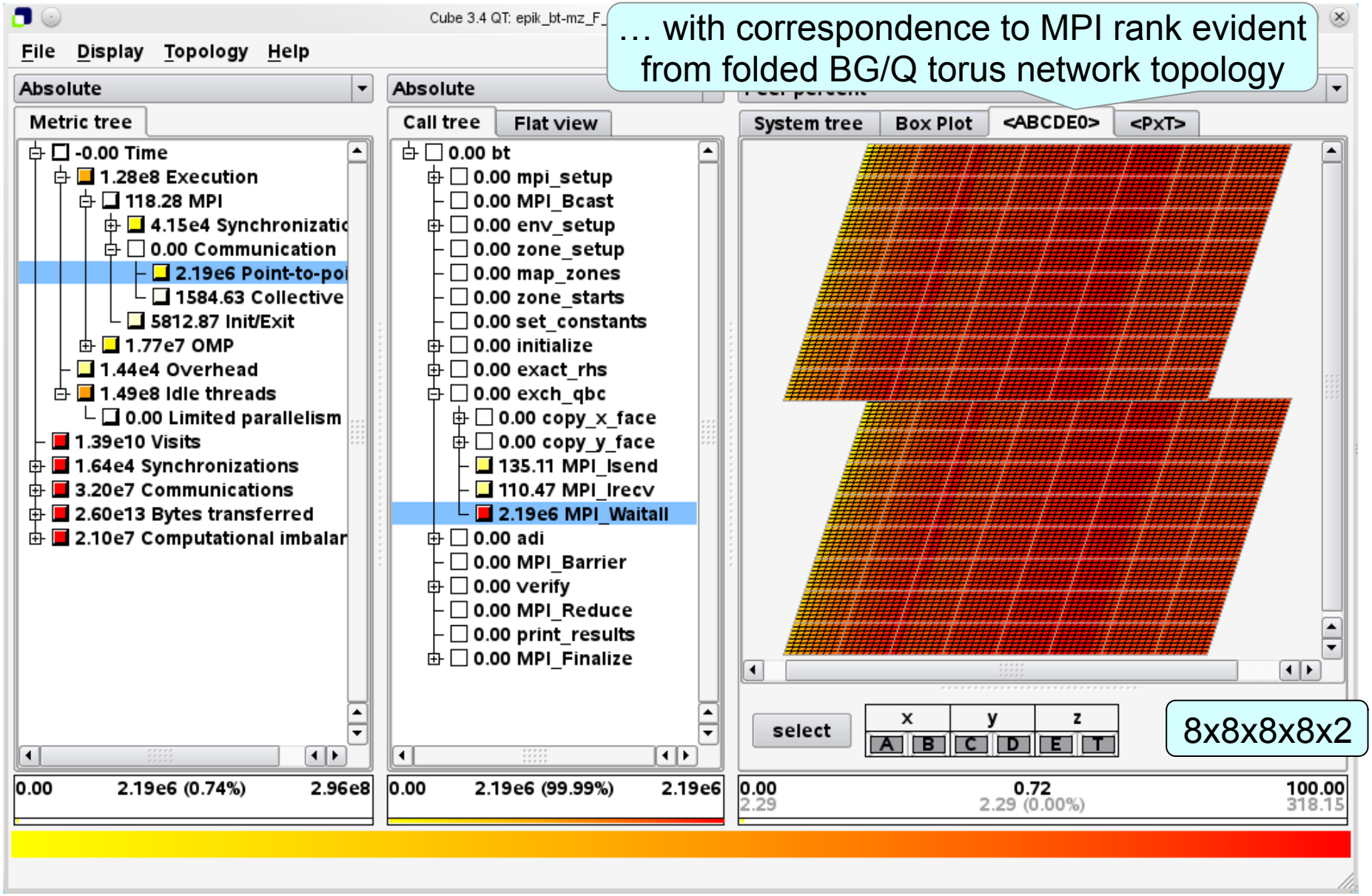
Idle threads time



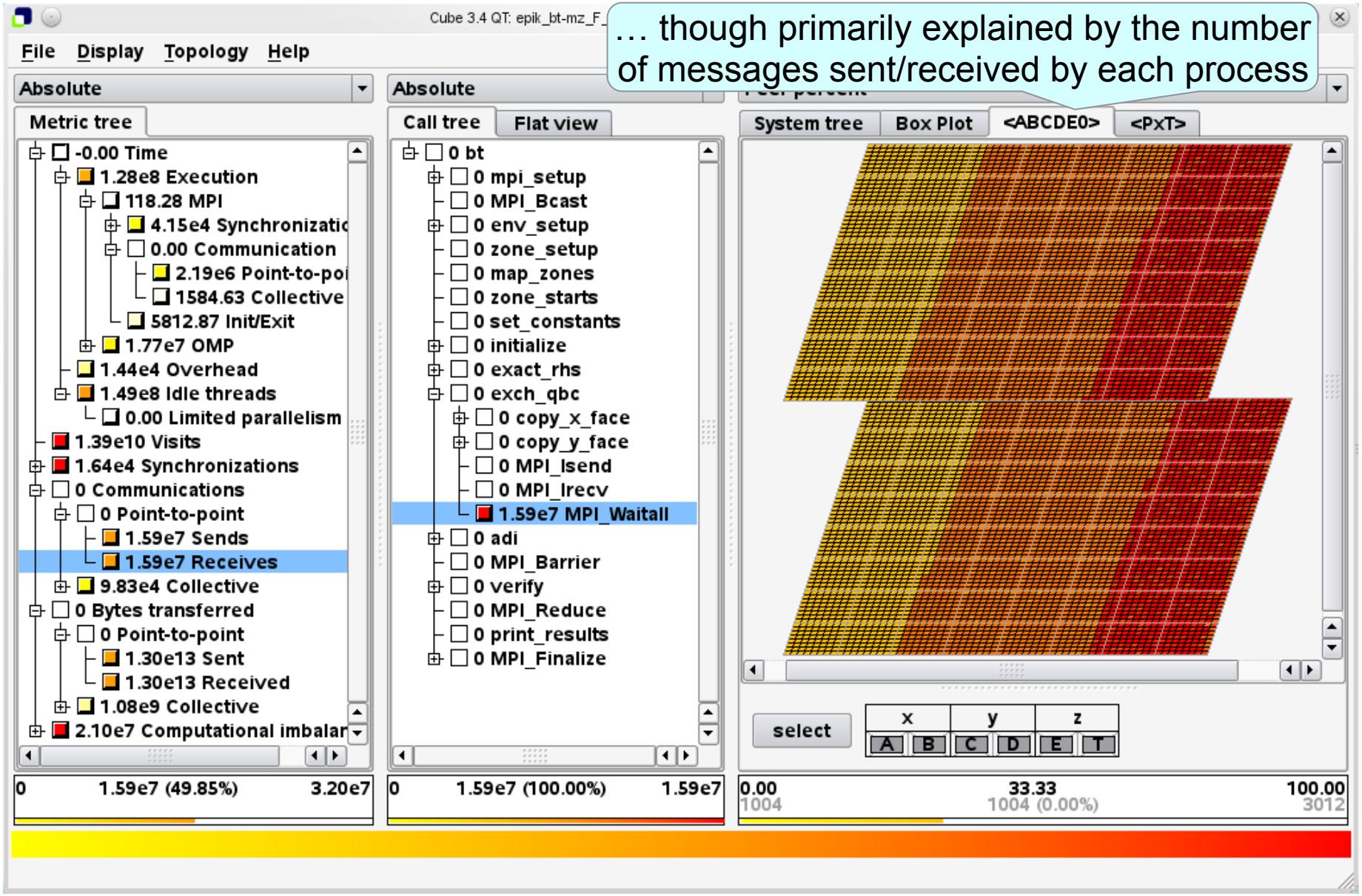
MPI point-to-point communication time



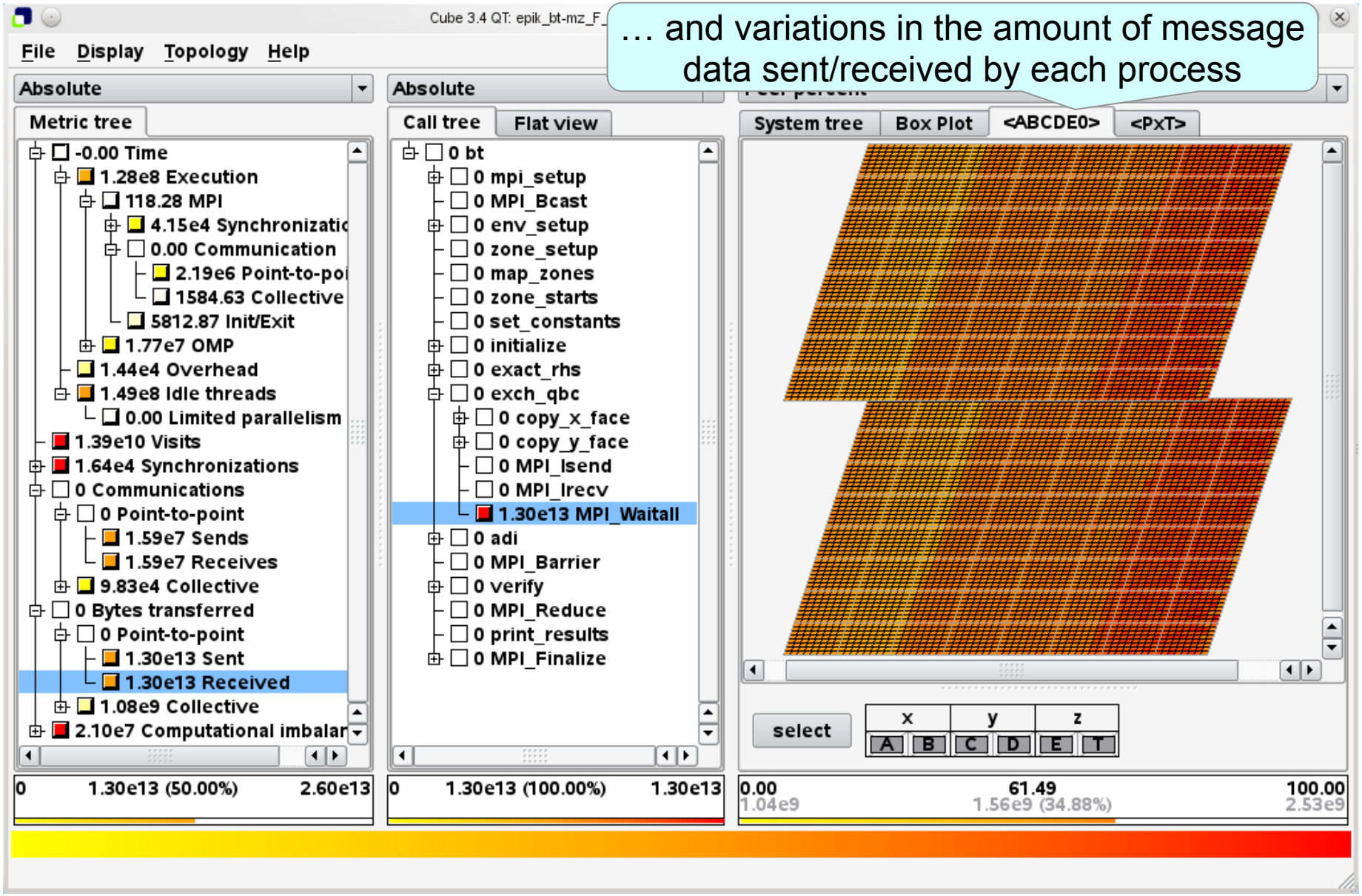
MPI point-to-point communication time



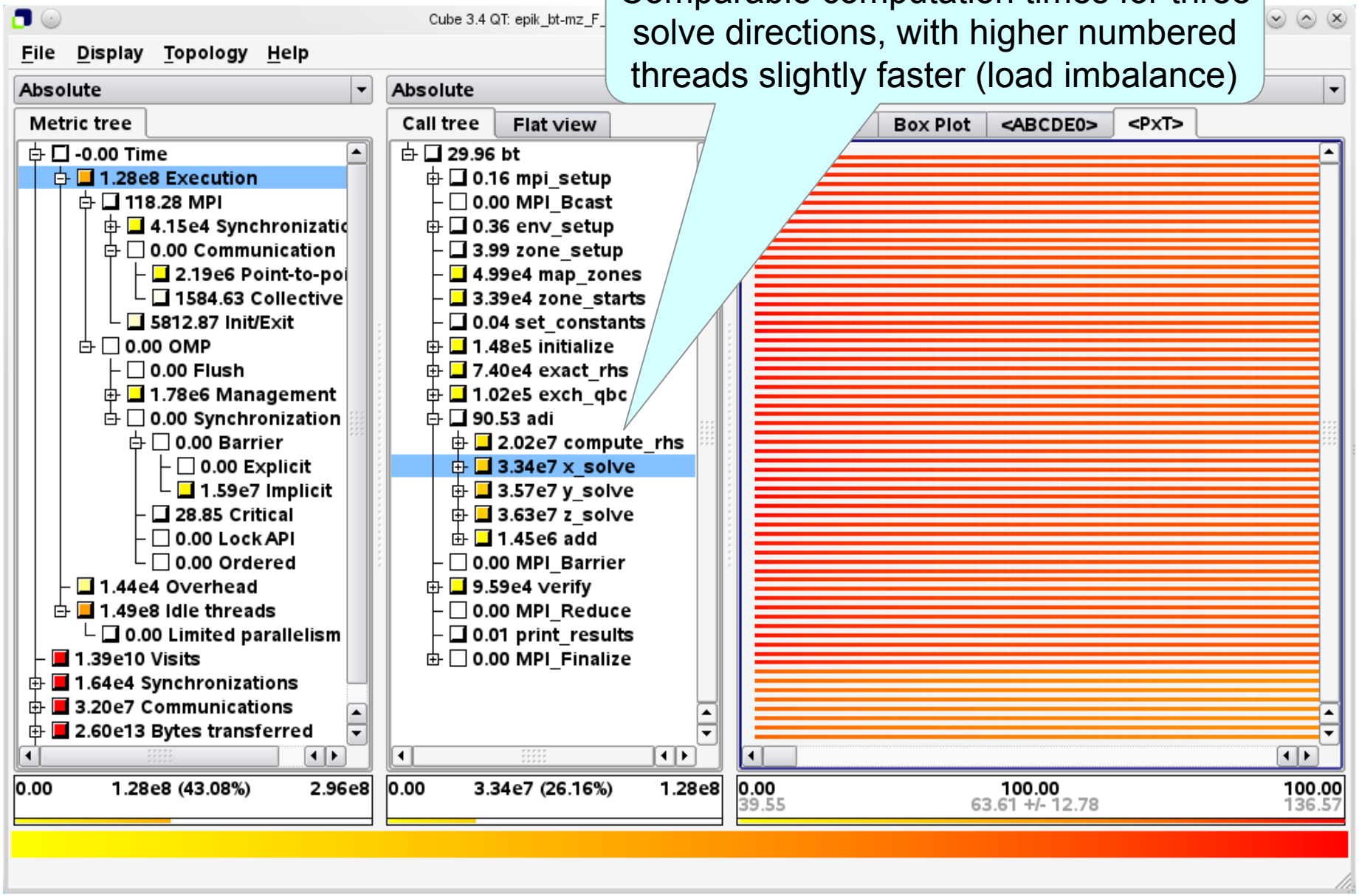
MPI point-to-point receive communications



MPI point-to-point bytes received

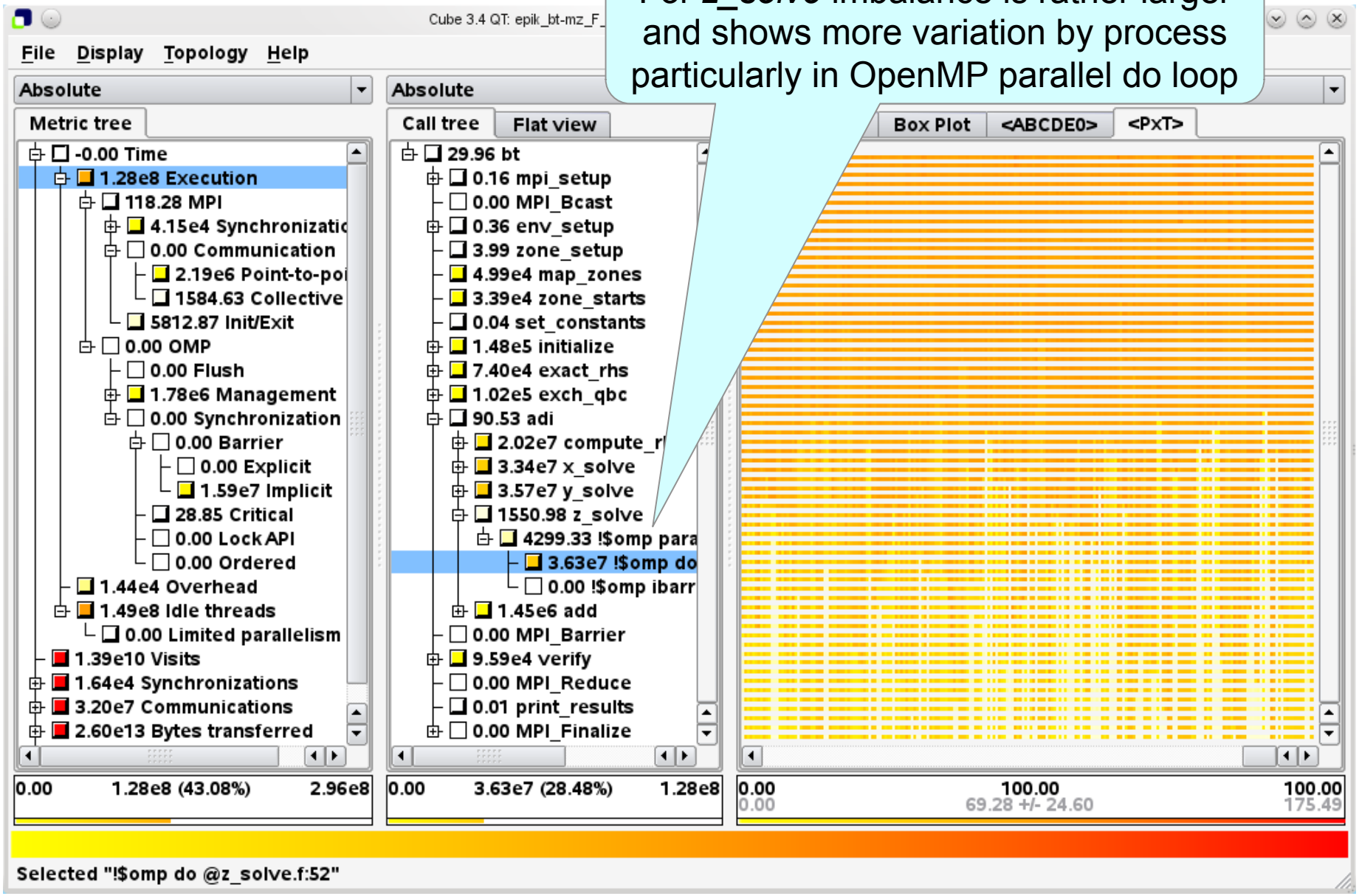


Computation time (*x*)

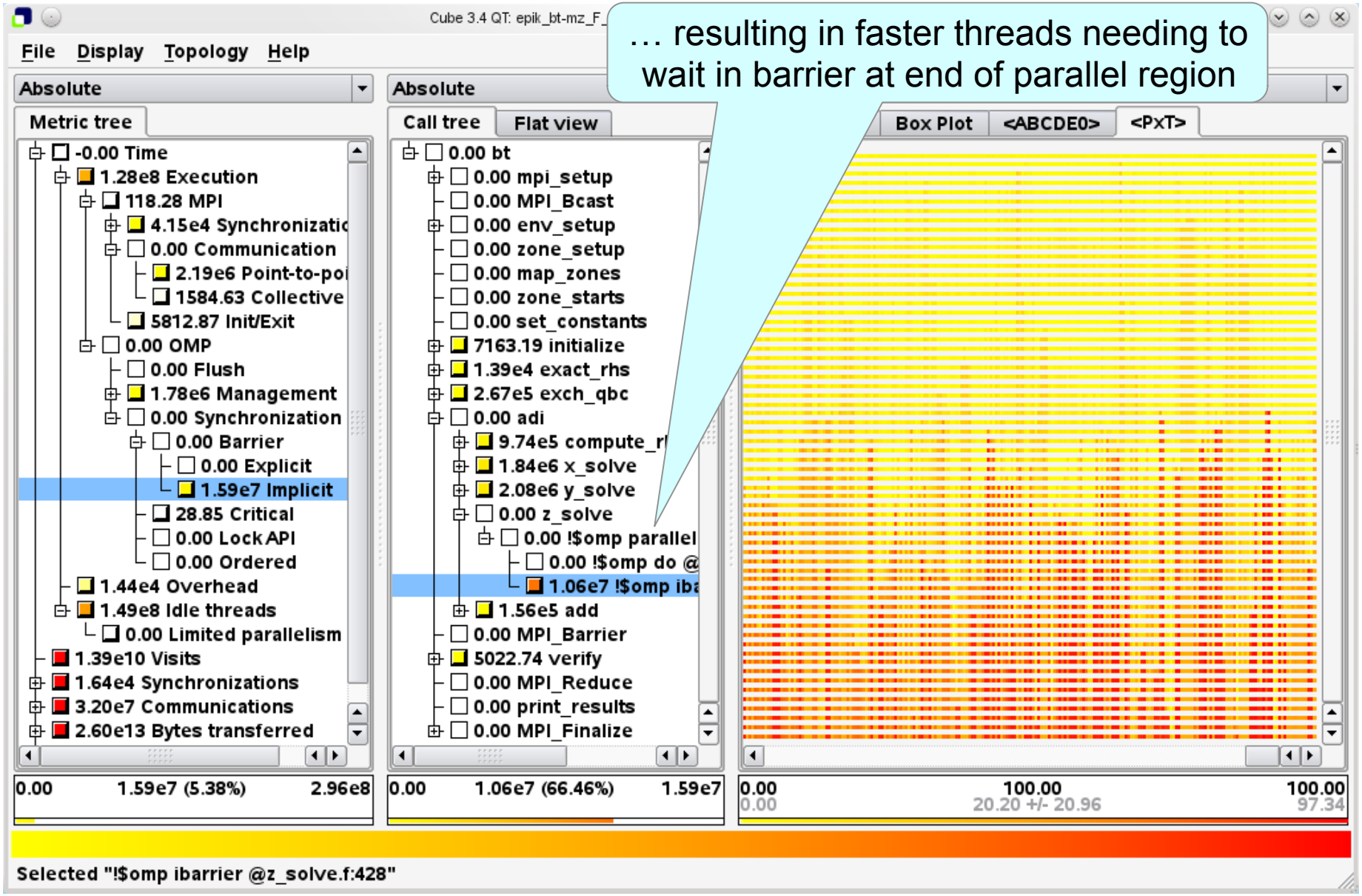


Computation time (z)

For *z_solve* imbalance is rather larger and shows more variation by process particularly in OpenMP parallel do loop

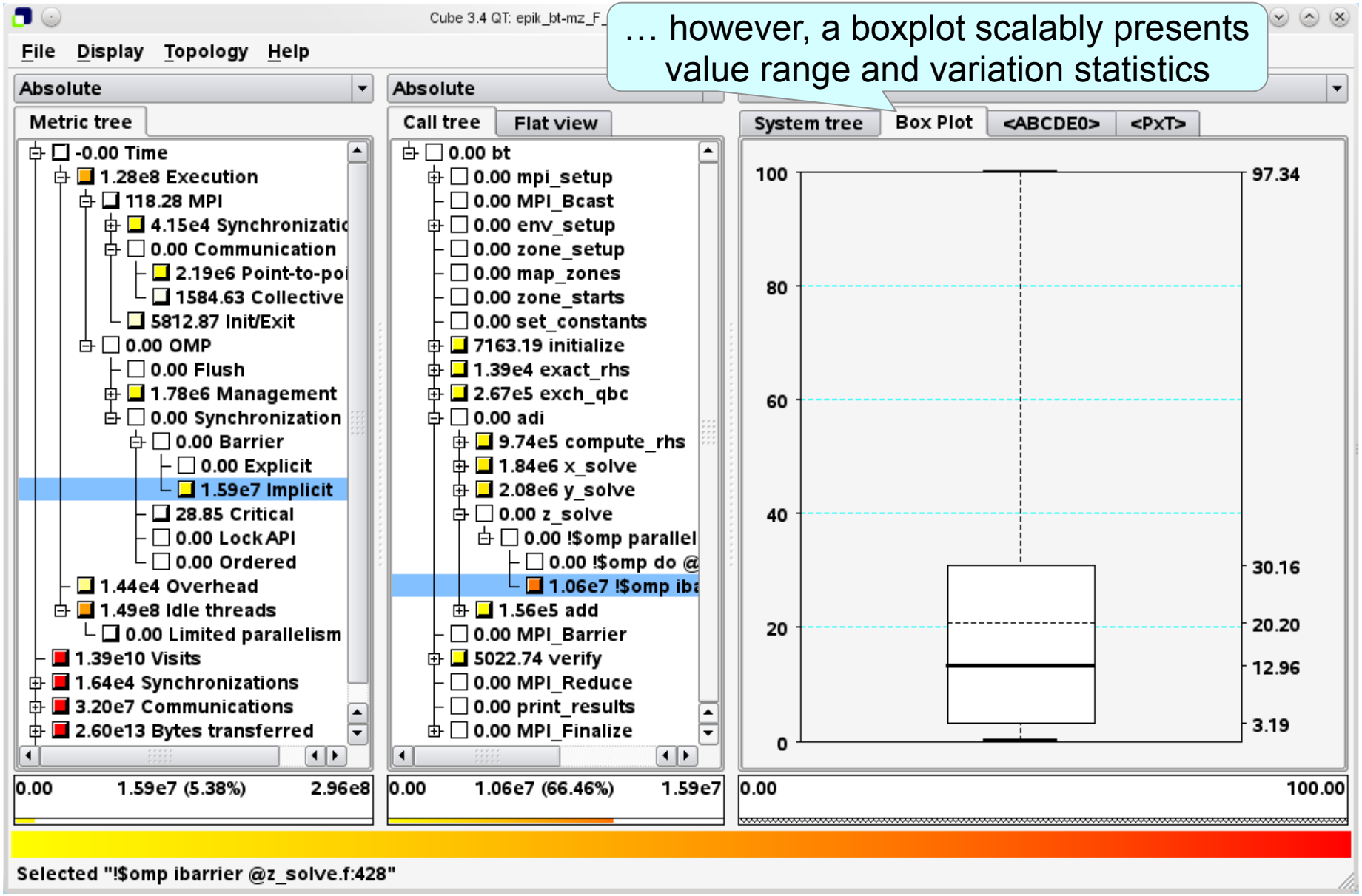


OpenMP implicit barrier synchronization time (z)

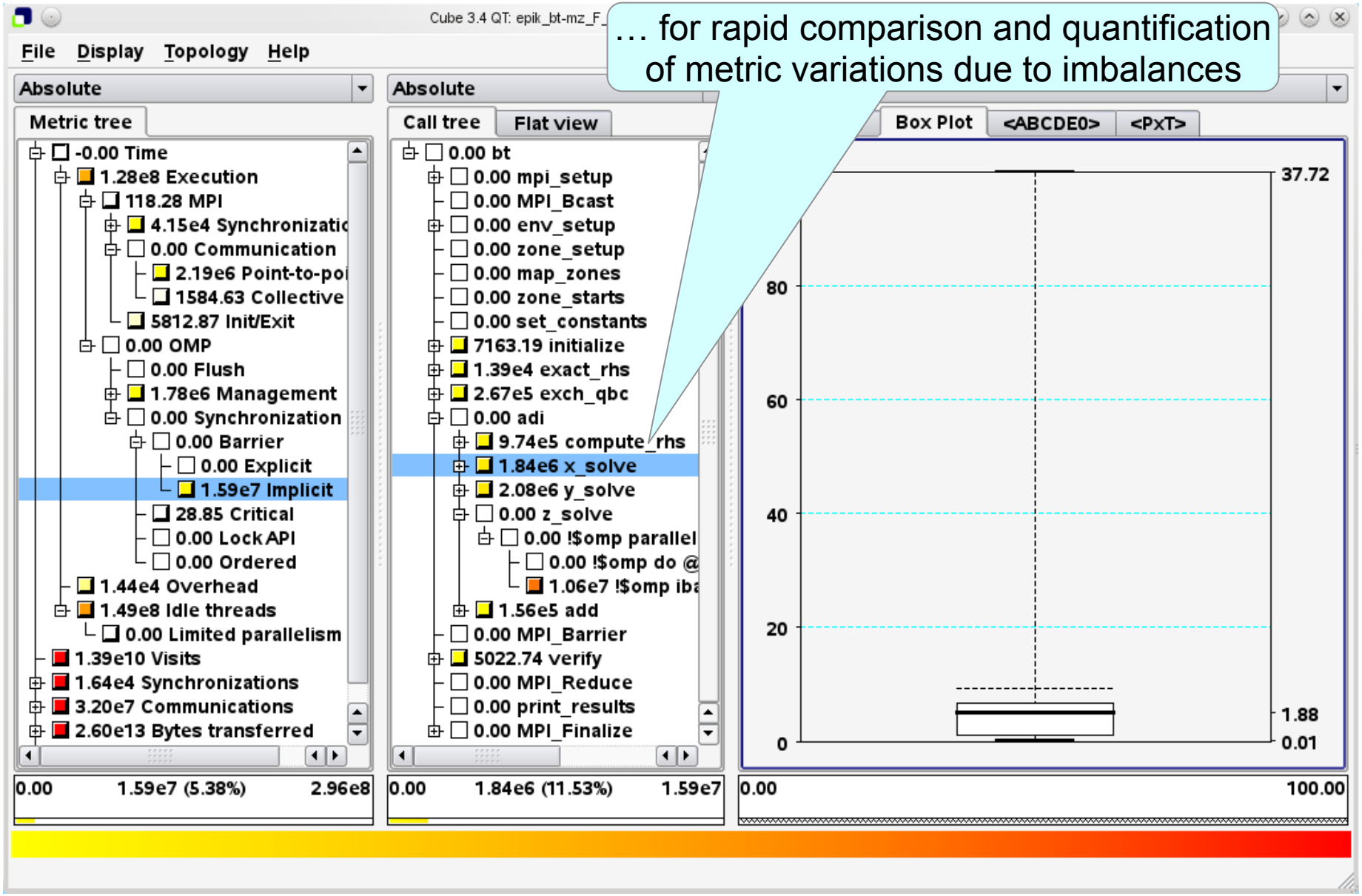


OpenMP implicit barrier synchronization time (z)

OpenMP implicit barrier synchronization time (z)



OpenMP implicit barrier synchronization time (*x*)



Scalasca scalability issues/optimizations (Part 3)

- Runtime summary analysis of BT-MZ successful at largest scale
  - 8192 MPI processes each with 64 OpenMP threads = ½ million
  - Only 3% measurement dilation versus uninstrumented execution
    - Latest XL compiler and OPARI instrumentation more efficient
    - Compilers can selectively instrument routines to avoid filtering
- Integrated analysis of MPI & OpenMP parallelization overheads
  - performance of both need to be understood in hybrid codes
  - MPI message statistics can explain variations in comm. times
- Time for measurement finalization grew linearly with num. processes
  - only 39 seconds for process and thread identifier unification
  - but 745/920 seconds to collate and write data for analysis report
- Analysis reports contain data for many more processes and threads than can be visualized (even on large-screen monitors)
  - fold & slice high dimensionality process topology
  - compact boxplot presents range and variation of values

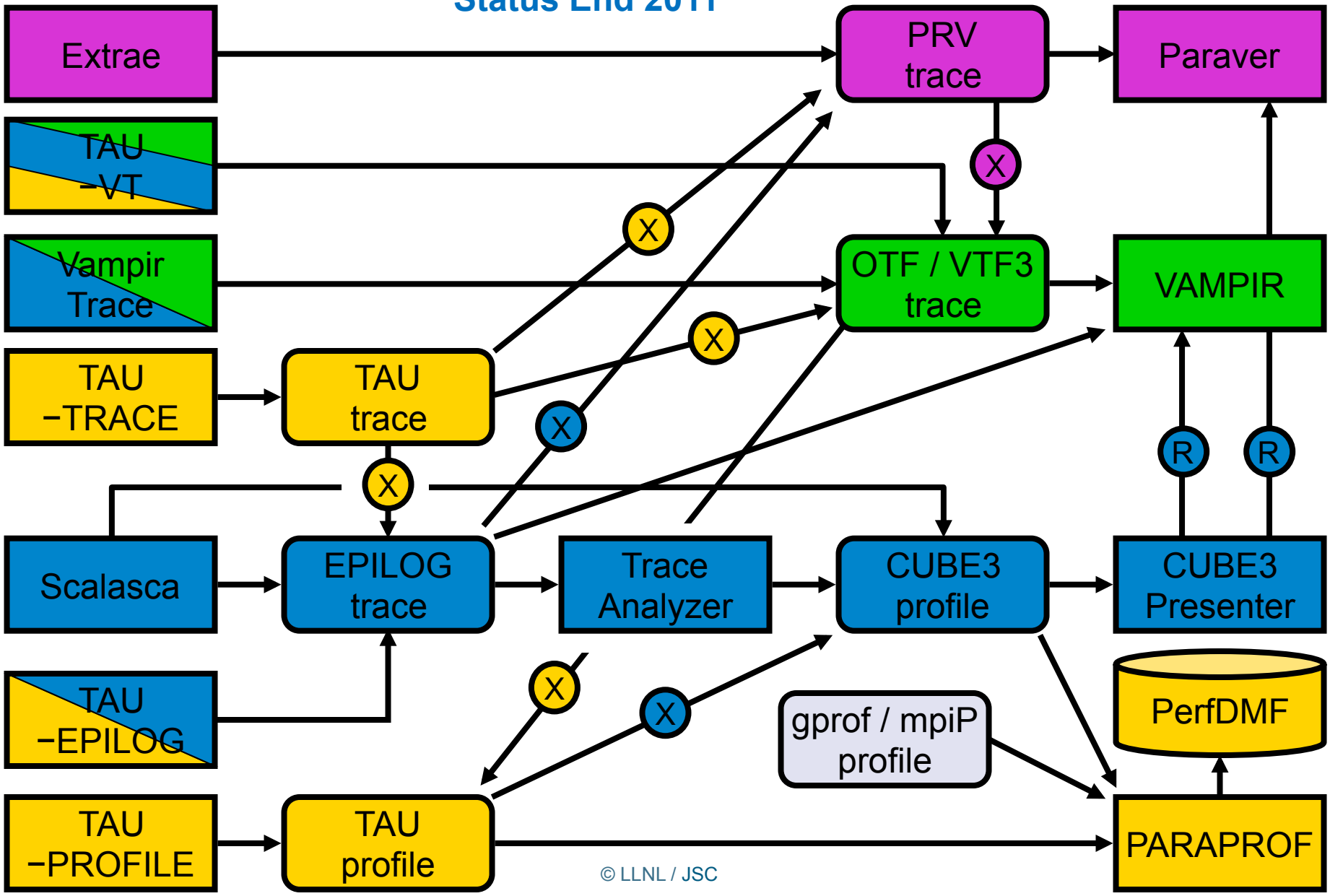
Summary / Case Studies

- Complex applications executing at very large scale provide significant challenges for performance analysis tools
- Scalability requires a range of complementary instrumentation, measurement & analysis capabilities, including:
  - Selective instrumentation & measurement filtering
  - Run-time summarization & event trace analysis
  - Parallel trace analysis without re-writing or merging of trace files
  - Shared multi-files for storing trace data
  - Use of local communicator rank identifiers in event records
  - Efficient communicator management
  - Hierarchical unification of definition identifiers
  - Compact configurable presentations of analyses and statistics for interactive exploration
- Portability and inter-operability important too

MASTERING FUTURE CHALLENGES

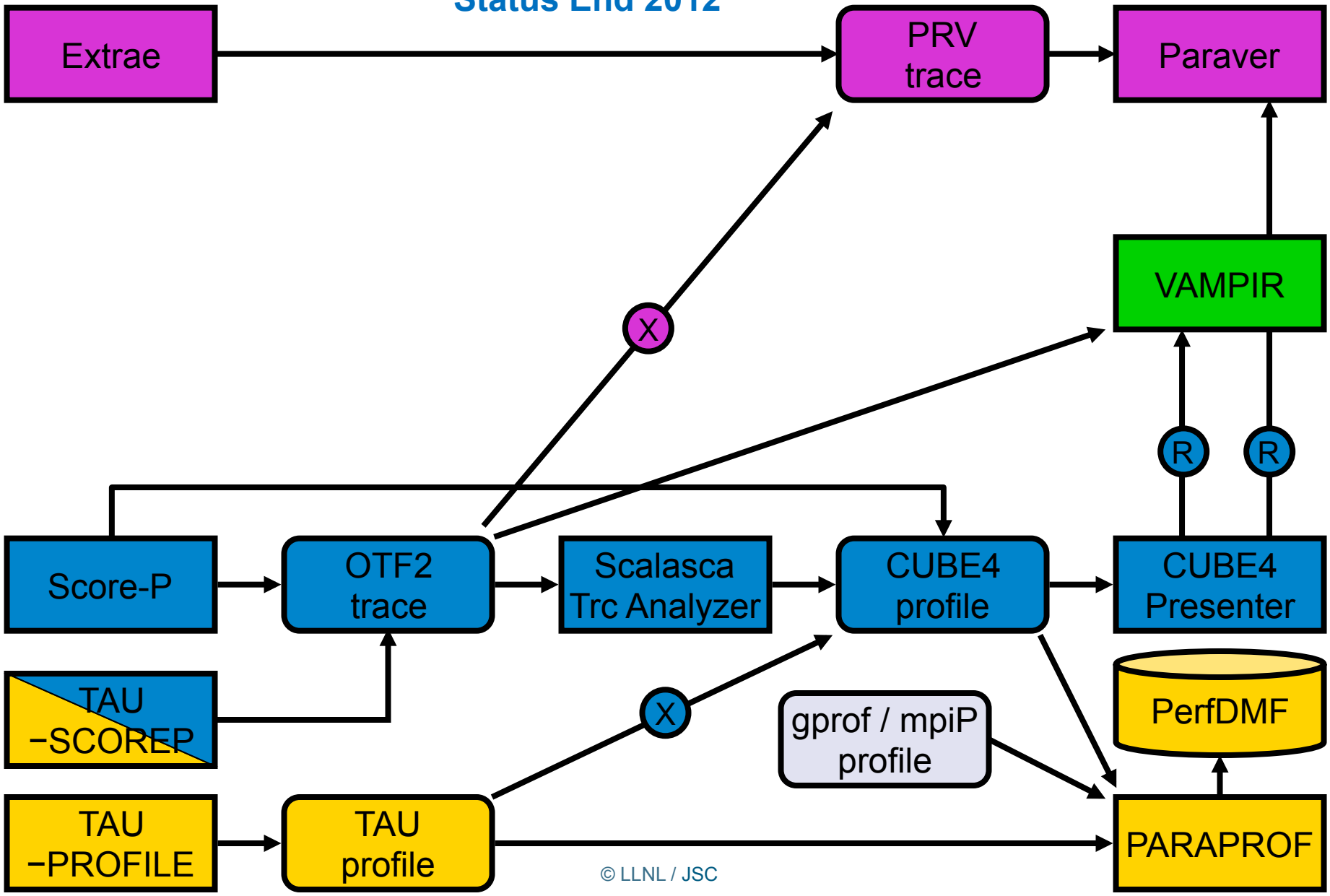
Scalasca ⇔ TAU ⇔ VAMPIR ⇔ Paraver

Status End 2011



Scalasca ⇔ TAU ⇔ VAMPIR ⇔ Paraver

Status End 2012



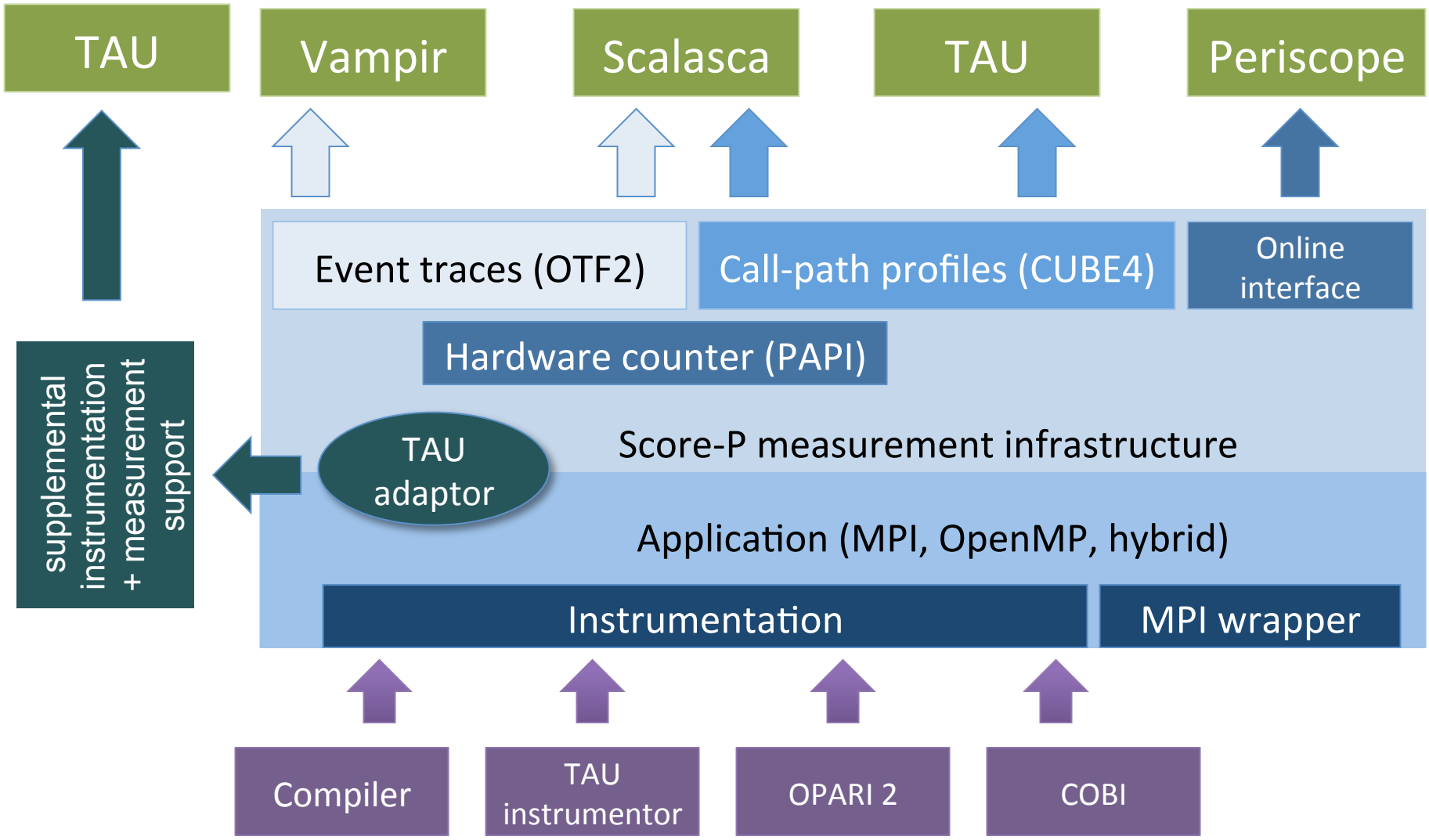
Score-P Objectives

- Make common part of Periscope, Scalasca, TAU, and Vampir a community effort
  - **Score-P measurement system**
- Save manpower by sharing resources
- Invest this manpower in analysis functionality
  - Allow tools to differentiate faster according to their specific strengths
  - Increased benefit for users
- Avoid the pitfalls of earlier community efforts
  - Start with small group of partners
  - Build on extensive history of collaboration

Score-P Design Goals

- **Functional requirements**
  - Performance data: profiles, traces
  - Initially direct instrumentation, later also sampling
  - Offline and online access
  - Metrics: time, communication metrics and hardware counters
  - Initially MPI 2 and OpenMP 3, later also CUDA and OpenCL
- **Non-functional requirements**
  - Portability: all major HPC platforms
  - Scalability: petascale
  - Low measurement overhead
  - Easy installation through UNITE framework
  - Robustness
  - Open source: New BSD license

Score-P Architecture

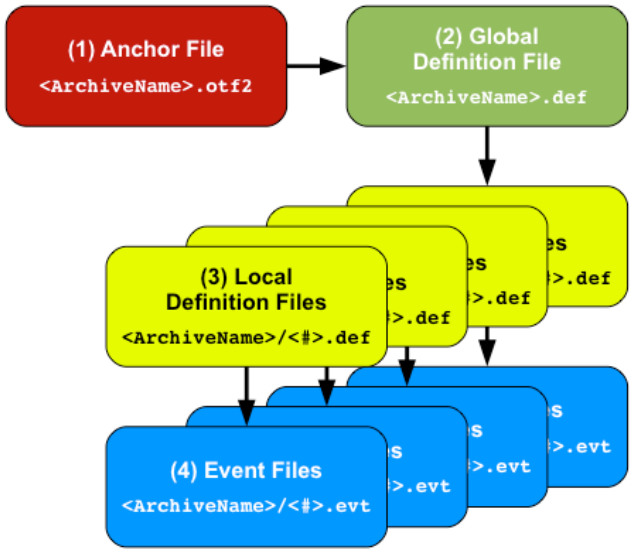


Score-P Partners

- Forschungszentrum Jülich, Germany
- German Research School for Simulation Sciences, Aachen, Germany
- Gesellschaft für numerische Simulation mbH Braunschweig, Germany
- RWTH Aachen, Germany
- Technische Universität Dresden, Germany
- Technische Universität München, Germany
- University of Oregon, Eugene, USA

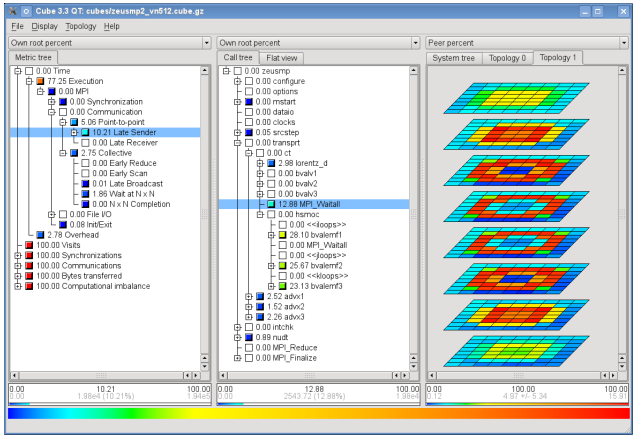
OTF-2 Tracing Format

- Successor to OTF and EPILOG
- Same basic structure as OTF, EPILOG, or other formats
- Design goals
  - High scalability
  - Low overhead (storage space and processing time)
  - Good read/write performance
    - Reduced number of files during initial writing via SIONlib
  - Compatibility reader for OTF and Epilog formats
  - Extensibility



CUBE-4 Profiling Format

- Latest version of a family of profiling formats
  - Still under development, to be released soon
- Representation of three-dimensional performance space
  - Metric, call path, process or thread
- File organization
  - Metadata stored as XML file
  - Metric values stored in binary format
    - Two files per metric:
      data + index for storage-efficient
      sparse representation



- Optimized for
  - High write bandwidth
  - Fast interactive analysis through incremental loading

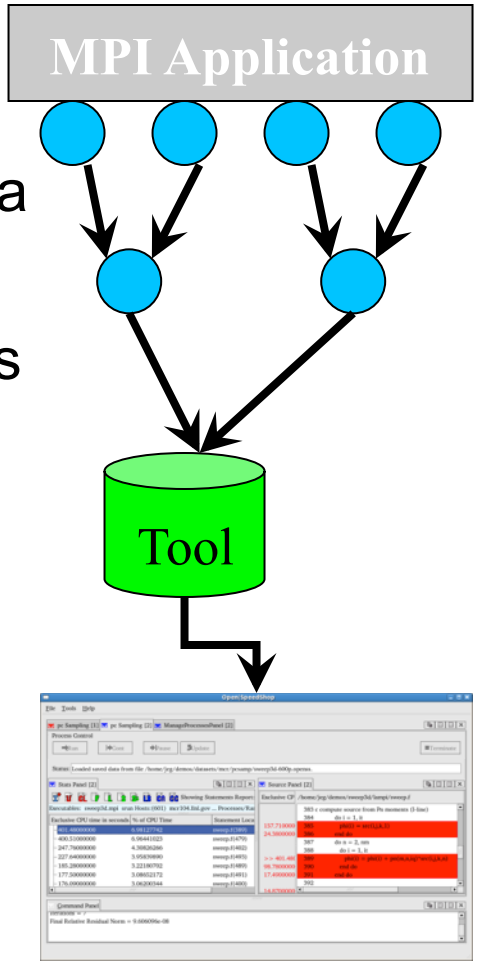
Summary: The Need for Integration

- Successful performance analysis requires to use the right tool for the right purpose
  - We need well-defined performance analysis workflows
  - We need tightly integrated tool sets

- Score-P Status and Future Plans
  - Bug fix release for ISC12 (V1.0.2)
    - http://www.score-p.org/
  - Future extensions
    - Heterogeneous computing (EU ITEA2 H4H project)
    - Time-series profiling (EU HOPSA + BMBF LMAC projects)
    - Sampling (BMBF LMAC project)

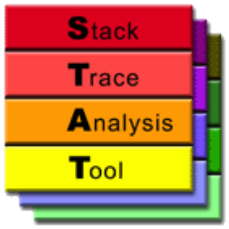
Petascale Tools Required New Developments

- Scalability requires online aggregation
  - Expected data volumes growing
  - We can no longer store all performance data
  - Current profilers are too generic
  - Need application/question specific operators
- Implementation is simple in principle, but poses difficult engineering questions
  - Scalability
  - Node location of intermediate nodes
  - Data storage
- Need to avoid re-implementations
  - Most tools face the same issue
  - Need for shared component infrastructures



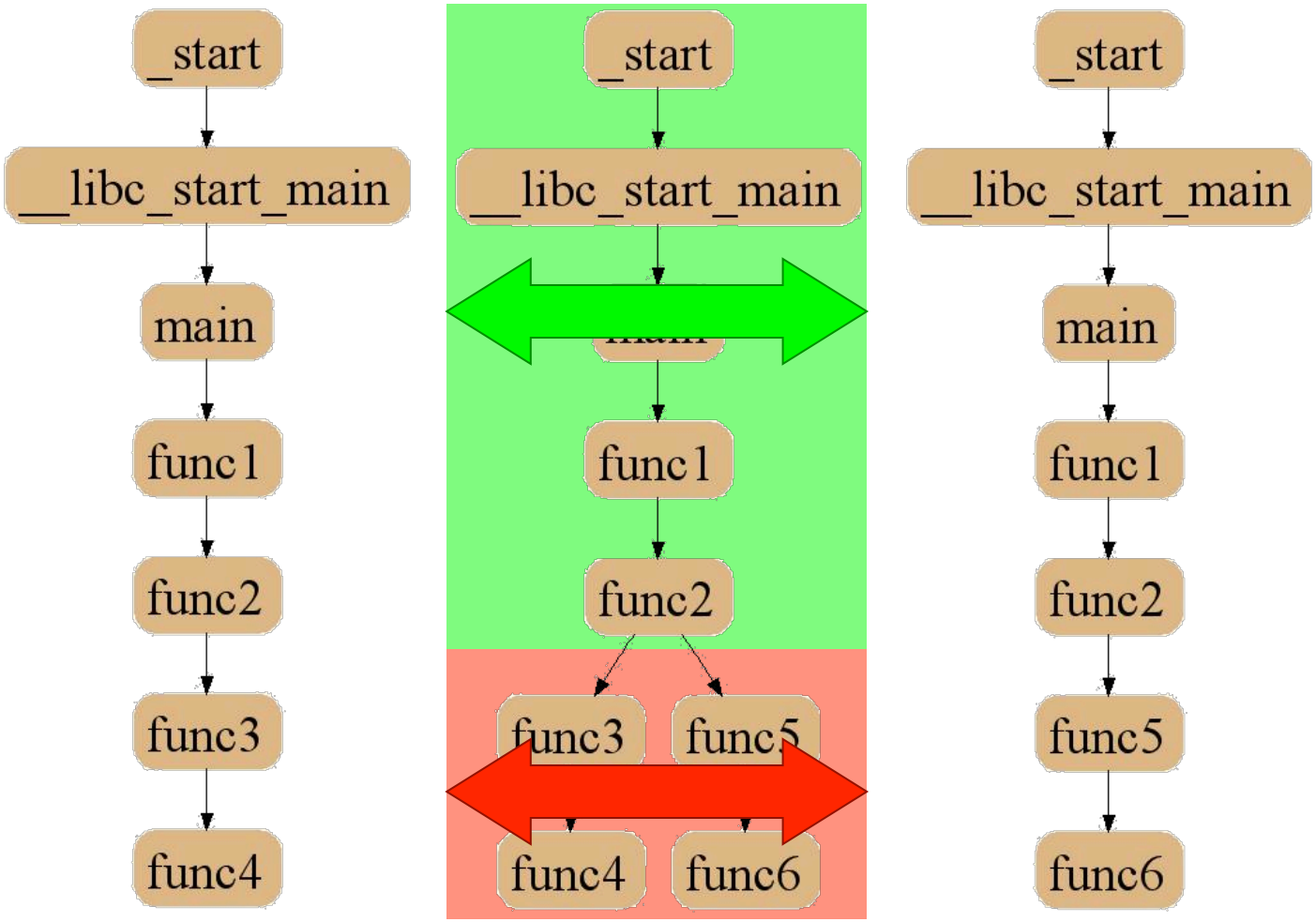
STAT: Aggregating Stack Traces for Debugging

- Existing debuggers don't scale
  - Inherent limits in the approaches
  - Need for new, scalable methodologies
- Need to pre-analyze and reduce data
  - Fast tools to gather state
  - Help select nodes to run conventional debuggers on
- Scalable tool: STAT
  - Stack Trace Analysis Tool
  - Goal: Identify equivalence classes
  - Hierarchical and distributed aggregation of stack traces from all tasks
  - Stack trace merge <1s from 200K+ cores

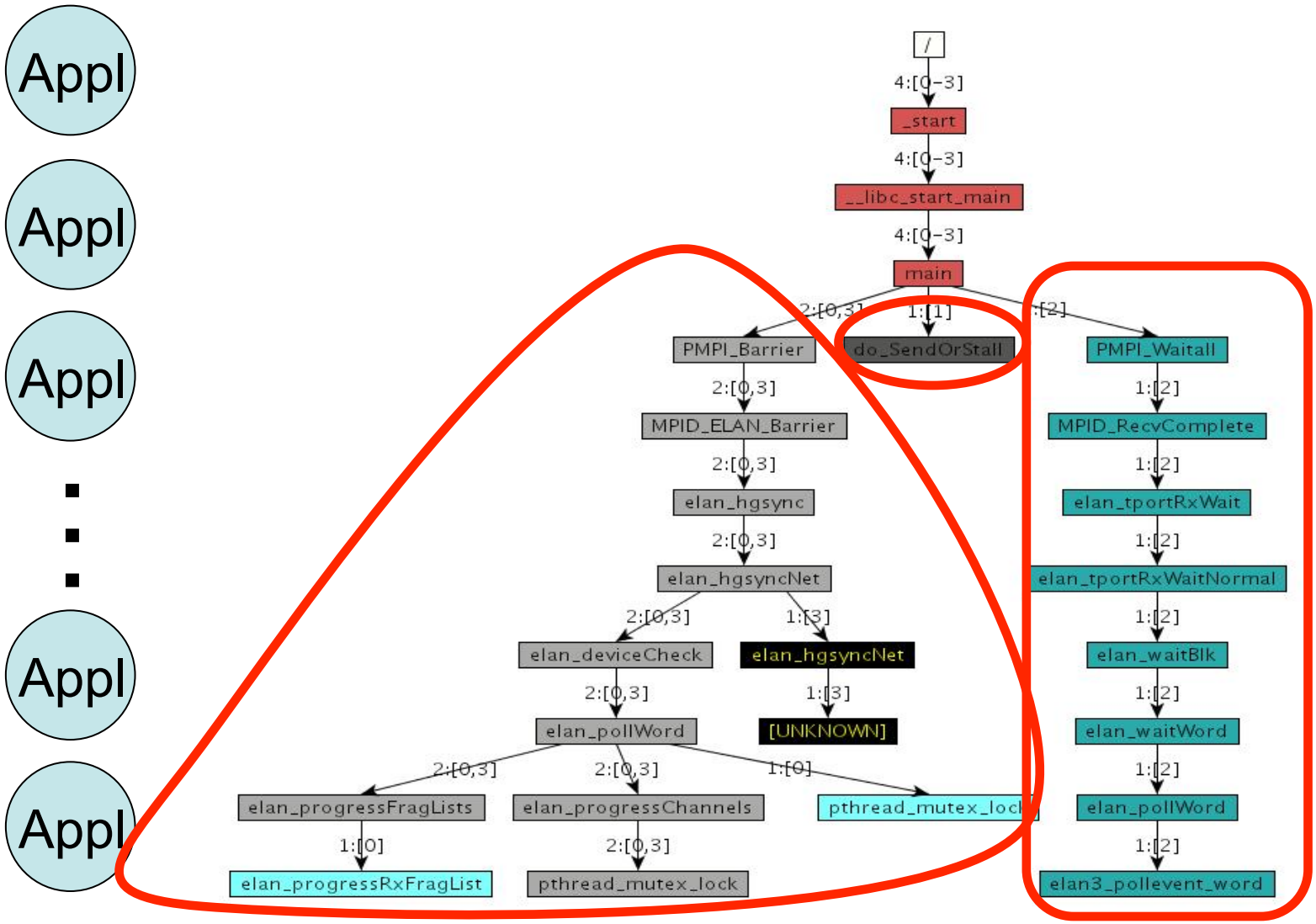


(Project by LLNL, UW, UNM)

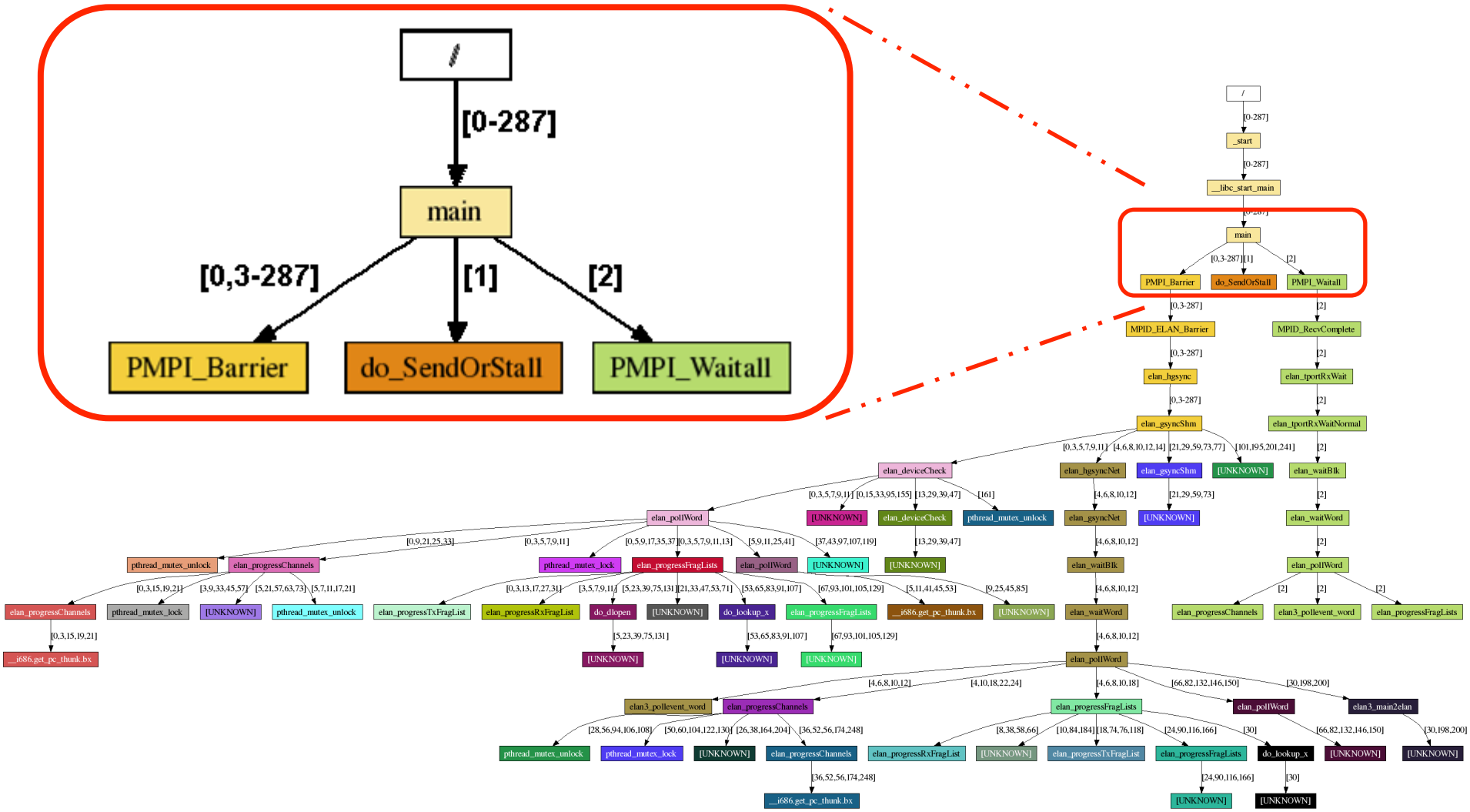
Distinguishing Behavior with Stack Traces



3D-Trace Space/Time Analysis

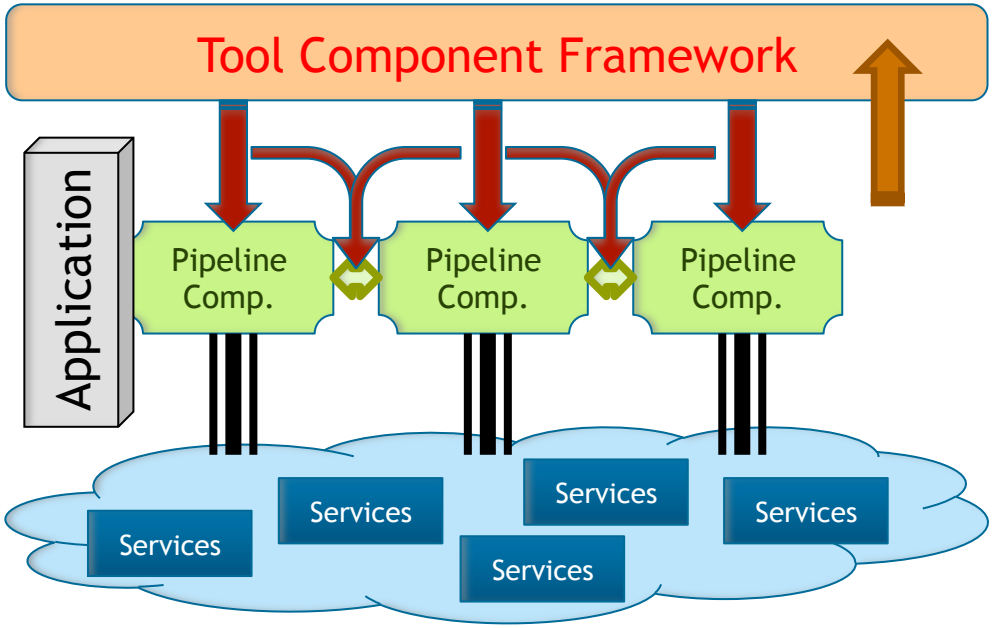


Scalable Representation

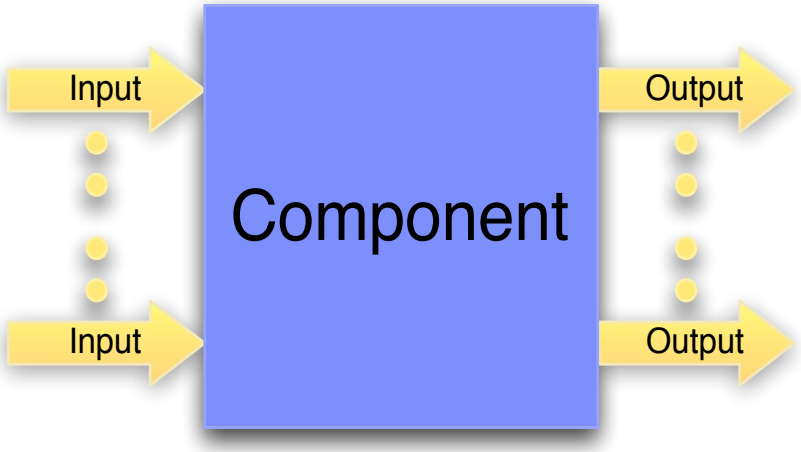


Shared Tool Frameworks

- Component Based Tool Framework (CBTF)
  - Independent components connected by typed pipes
  - Transforming data coming from the application on the way to the user
  - External specification of which components to connect
  - Each combination of components is/can be “a tool”
  - Shared services
- Partners
  - Krell Institute
  - LANL, LLNL, SNLs
  - ORNL
  - UW, UMD
  - CMU

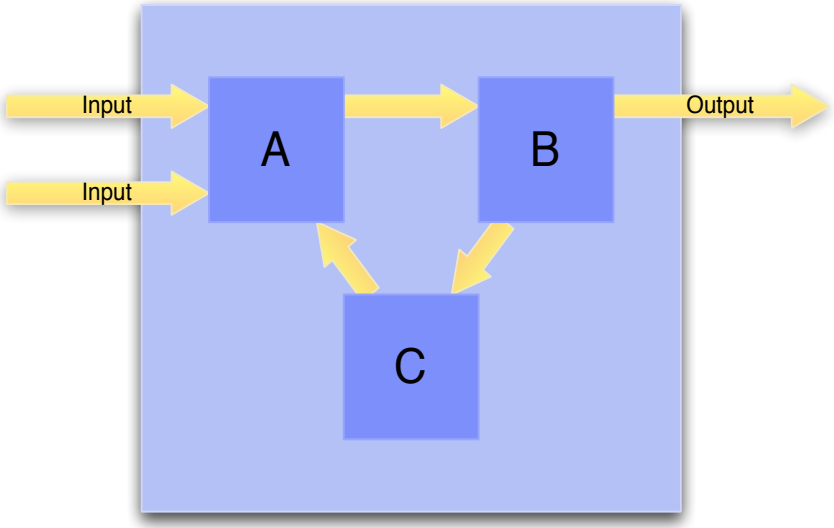


CBTF Modules



- Data-Flow Model
  - Accepts Inputs
  - Performs Processing
  - Emits Outputs
- C++ Based
- Provide Metadata
  - Type & Version
  - Input Names & Types
  - Output Names & Types
- Versioned
  - Concurrent Versions
- Packaging
  - Executable-Embedded
  - Shared Library
  - Runtime Plugin

CBTF Component Networks



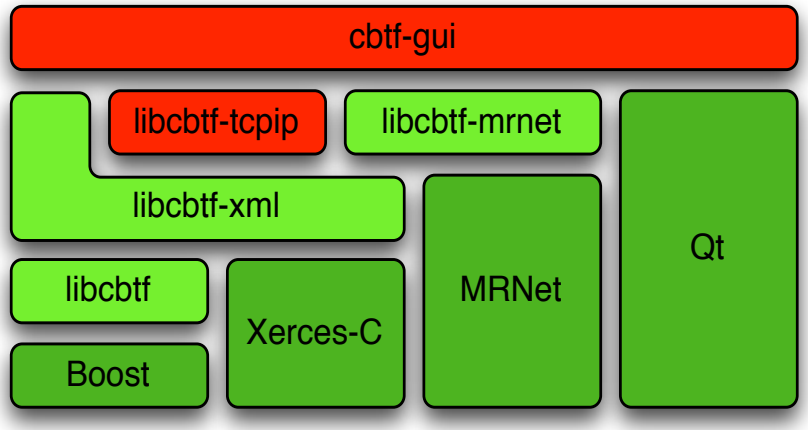
- Components
  - Specific Versions
- Connections
  - Matching Types
- Arbitrary Component Topology
  - Pipelines
  - Graphs with cycles
  - ….
- Recursive
  - Network itself is a component
- XML-Specified

Specifying Component Networks to Create New Tools

```
....
<Type>ExampleNetwork</Type>
<Version>1.2.3</Version>
<SearchPath>.:/opt/myplugins</SearchPath>
<Plugin>myplugin</Plugin>
    <Component>
      <Name>A1</Name>
     <Type>TestComponentA</Type>
    </Component>
...
  <Network>
...
     <Connection>
       <From>
         <Name>A1</Name>
         <Output>out</Output>
       </From>
       <To>
          <Name>A2</Name>
          <Input>in</Input>
        </To>
      </Connection>
...
 </Network>
```

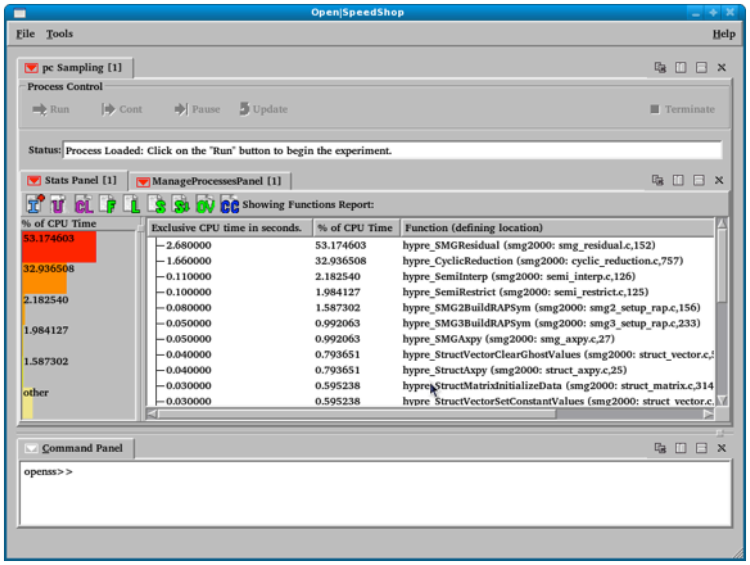
- Users can create new tools by specifying new networks
  - Combine existing functionality
  - Reuse general model
  - Add application specific details
    - Phase/context filters
    - Data mappings
- Connection information
  - Which components?
  - Which ports connected?
  - Grouping into networks
- Implemented as XML
  - User writable
  - Could be generated by a GUI

CBTF Structure and Dependencies



- Minimal Dependencies
  - Easier Builds
- Tool-Type Independent
  - Performance Tools
  - Debugging Tools
  - etc…
- Completed Components
  - Base Library (libcbtf)
  - XML-Based Component Networks (libcbtf-xml)
  - MRNet Distributed Components (libcbtf-mrnet)
- Planned Components
  - TCP/IP Distributed Component Networks
  - GUI Definition of Component Networks

Tools on Top of CBTF



- Open|SpeedShop v2.0
  - CBTF created by componentizing the existing Open|SpeedShop
  - Motivation:
    scalability & maintainability
  - Components for
    - Data collection
    - Data aggregation
    - Connection to existing GUI & DB
- Further tools in progress
  - Tools for observing memory consumption
  - GPU performance analysis
  - Tools for system administration and health monitoring

Summary: The Need for Components

- We need frameworks that enable …
  - Independently created and maintained components
  - Flexible connection of components
  - Assembly of new tools from these components by the user
- CBTF is designed as a generic tool framework
  - Components are connected by typed pipes
  - Infrastructure for hierarchical aggregation with user defined functions
  - Component specification is external through XML files
  - Tailor tools by combining generic and application specific tools
- CBTF is available as a pre-release version
  - First prototype of Open|SpeedShop v2.0 working
  - Several new tools built on top of CBTF
  - Wiki at http://ft.ornl.gov/doku/cbtfw/start

FUTURE:
PETASCALE ⇨ EXASCALE ⇨ ?

Observations

From workshop report
SDTPC Aug 2007
http://www.csm.ornl.gov/workshops/Petascale07/

- **Petascale is not terascale scaled up!**
  - More than linear increase of scale
  - Multi-core processors
    - ⇨ Multi-mode parallelism
    - ⇨ Reduced memory per core
  - Heterogeneity via HW acceleration (Cell, FPGA, GPU, …)
    - ⇨ New programming models (needed)
    - ⇨ Higher system diversity
- More emphasis on
  - **Fault-tolerance** and **performability**
  - **Automated diagnosis and remediation**

Projection for a Exascale System*

System attributes	2010	"2015"		"2018"		Difference 2010 & 2018
System peak	2 Pflop/s	200 Pflop/s		1 Eflop/sec		O(1000)
Power	6 MW	15 MW		~20 MW		
System memory	0.3 PB	5 PB		32-64 PB		O(100)
Node performance	125 GF	0.5 TF	7 TF	1 TF	10 TF	O(10) – O(100)
Node memory BW	25 GB/s	0.1 TB/ sec	1 TB/sec	0.4 TB/sec	4 TB/sec	O(100)
Node concurrency	12	O(100)	O(1,000)	O(1,000)	O(10,000)	O(100) – O(1000)
Total Concurrency	225,000	$O(10^8)$		$O(10^9)$		O(10,000)
Total Node Interconnect BW	1.5 GB/s	20 GB/sec		200 GB/sec		O(100)
MTTI	days	O(1day)		O(1 day)		- O(10)

* From http://www.exascale.org

Planning for Exa/Extreme-Scale

- International Exascale Software Project (IESP)
- European efforts
  - European Exascale Software Initiative (EESI, EESI2)
  - Exascale Innovation Center (EIC), JSC + IBM
  - Intel Exascale Labs (JSC, Paris, Leuven, BSC)
  - EU FP7 Exascale projects (DEEP, MontBlanc, CRESTA)
- Projects and Planning by the US Department of Energy
  - Office of Science projects
  - Exascale Co-Design Centers
  - ASC Planning and Co-Design efforts
- DOD/NSA's Advanced Computing Initiative

IESP

- **International Exascale Software Project**
- International collaboration
  - Started Apr 2009
- http://www.exascale.org/

- Objectives
  - Develop international exascale (system) software roadmap

IJHPCA, Feb 2011, http://hpc.sagepub.com/content/25/1/3

  - Investigate opportunities for international collaborations and funding
  - Explore governance structure and models for IESP

IESP Roadmap Components

4.1 Systems Software

- 4.1.1 Operating systems
- 4.1.2 Runtime Systems
- 4.1.3 I/O systems
- 4.1.4 Systems Management
- 4.1.5 External Environments

4.2 Development Environments

- 4.2.1 Programming Models
- 4.2.2 Frameworks
- 4.2.3 Compilers
- 4.2.4 Numerical Libraries
- 4.2.5 Debugging tools

4.3 Applications

- 4.3.1 Application Element: Algorithms
- 4.3.2 Application Support: Data Analysis and Visualization
- 4.3.3 Application Support: Scientific Data Management

4.4 Crosscutting Dimensions

- 4.4.1 Resilience
- 4.4.2 Power Management
- 4.4.3 Performance Optimization
- 4.4.4 Programmability

see IJHPCA, Feb 2011, http://hpc.sagepub.com/content/25/1/3

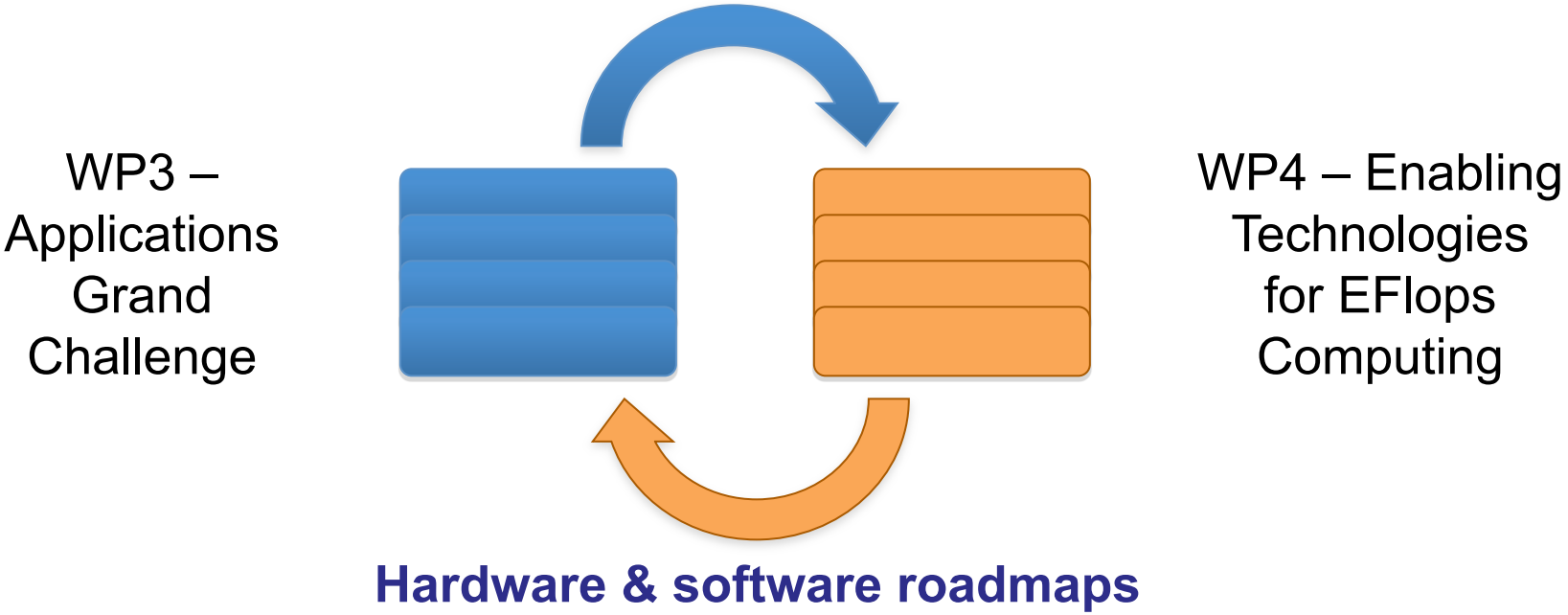
EESI

- **European Exascale Software Initiative**
- EU FP7
  - Funded Jun 2010 to Nov 2011
  - Funded Sep 2012 to Mar 2015
- http://www.eesi-project.eu/

- Objectives
  - Develop European exascale system and application software vision and roadmap
  - Investigate Europe's strengths and weaknesses
  - Identify sources of competitiveness for Europe
  - Investigate and propose programs in education and training for the next generation of computational scientists
- EU Roadmap and other reports available at website

General Structure

WP2 : Link with IESP and other projects



WP5 : Communication - Dissemination

Reports

- http://www.eesi-project.eu/pages/menu/publications/final-report-recommendations-roadmap.php
  - Final report Roadmap and Recommendations
  - Summary report Application Grand Challenges work groups
  - Summary report Enabling Technologies working groups

- http://www.eesi-project.eu/pages/menu/publications/working-group-reports.php
  - 8 detailed reports from the Application Grand Challenges and Enabling Technologies working groups

- http://www.eesi-project.eu/pages/menu/publications/investigation-of-hpc-initiatives.php
  - Survey international Exascale activities

EIC

- **Exascale Innovation Center**
- Collaboration with IBM (Germany)
  - Started Apr 2010

- Objectives
  - Energy-efficient architectures
  - Scalable storage and I/O
  - Exascale characteristics (Hardware and Software)

ECL

- **ExaCluster Laboratory**
- Collaboration with Intel (Germany) and ParTec
  - Started June 2010

- Objectives
  - Research current challenges in systems management software for large heterogeneous supercomputer systems
  - Research on open exascale runtime system software and software tools.

European FP7 Exascale Research Projects

- DEEP – Dynamical Exascale Entry Platform
  - http://www.deep-project.eu
  - Regular cluster combined with booster (cluster of Intel MIC)
  - Dynamic assignments of booster nodes to cluster nodes
- MontBlanc
  - http://www.montblanc-project.eu
  - Cluster build out of low-power ARM CPUs and mobile GPUs
- CRESTA
  - http://www.cresta-project.eu
  - Enabling a key set of co-design applications for Exascale
  - Building and exploring systemware for Exascale platforms

Exascale Efforts at DOE / ASCR

- Range of projects funded through several FOAs
  - Architecture
  - X-Stack and X-Stack 2 – software infrastructure
  - Data Analysis
  - Resilient Solvers
- Exascale Co-Design centers focus on particular applications
  - Joint efforts on architecture, software, physics
  - Broad teams across multiple labs
- Three Co-Design centers funded at $4M/year for 5 years
  - Material design at Extreme Scale (lead by LANL)
  - Next generation reactor design (lead by ANL)
  - Combustion (lead by SNLs)
- Regular PI meetings, some joint with DOE/NNSA

Planning Efforts at ASC/NNSA

- Series of workshops
  - Held internally at first, later with external participation
  - Activities joint with Office of Science
- ASC has established eight working groups
  - Applications (Bert Still, LLNL)
  - Libraries and Solvers (Dana Knoll, LANL)
  - Programming Models (Pat McCormick, LANL)
  - Systems Software (Ron Minich, SNLs)
  - Hardware/Architectures (Paul Henning, LANL)
  - I/O (Lee Ward, SNLs)
  - Visualization and Data Analysis (Jim Ahrens, LANL)
  - Tools (Martin Schulz, LLNL)
- FastForward program targeted at industry participants
  - Managed by LLNL, run by the “E7” Laboratories
- Separate Co-Design activities targeted at ASC codes

ASC Report: Exascale Needs for Tools

1. We need to broaden Petascale tools **(Development)**
   - Petascale will be capacity computing
   - Broader user base, essential for many UQ problems
   - Need to harden tools for use by end users
   - Integration across tools (and other infrastructures)
   - Maintenance support to ensure sustainability
2. We need scale tools to Exascale **(Research)**
   - Focus on expert users / code teams
   - Specialized tools that analyze particular problems
   - Tools themselves need to be parallel and scalable
   - Tools need to tolerate faults
   - Will require machine resources / no longer free

ASC Report: Exascale Needs for Tools (cont.)

- Challenges in providing new capabilities
  - Scalability of measurement, analysis, and presentation
    - Incl. new metrics: memory, power, …
  - Turning information into insight
    - Despite flood and complexity of data from billions of threads
  - Dealing with new programming methodologies
    - Heterogeneous systems/architectures (HW and SW)
    - Coupled systems and applications
  - "What if" tools for Co-Design
- Challenges for tool implementations
  - Quick design of prototype tools for new scenarios
  - Getting right interfaces with the right abstractions
    - To SSW, HWA, Apps, Libraries, Runtimes, Compilers, …
  - Resiliency for tools and tool infrastructures

ASC Report: Crosscutting Issues for Tool Design

- Programming Models
  - Interfaces into the programming models and their runtimes
  - Code refactoring to transition to new programming models
  - Development tool chain, incl. compilers and libraries (like MPI)
- Architectures and Networks
  - Additional sensors, counters, hardware structure, …
  - Computational & network resources (shared with I/O, Visualization)
  - Use application characterization for co-design
  - Virtual prototyping
- System Software
  - Dynamic resource (co-)allocation
  - Information and configuration flow tool <-> runtime
  - Integrated runtime and resource management systems
- I/O, Storage, Visualization
  - Shared resources & infrastructure

FINAL THOUGHTS

Acknowledgements: Dagstuhl Seminar 10181

- Program Development for Extreme-Scale Computing
- 2nd to 7th May, 2010
- http://www.dagstuhl.de/10181/
- 45 participants
- Organizers:
  - B.Miller (U. Wisc)
  - J.Labarta (BSC)
  - M.Schulz (LLNL)
  - B.Mohr (JSC)

Tool Scaling Challenge (2010!)

- 3 months ahead
  - Provide two "scalable" applications
    - Pflotran (US), PEPC (EU)
  - Provide compute time on Jaguar (ORNL) and Jugene (JSC)
  - Request to apply tools on runs on at least 10,000 cores
- **Results**
  - Many succeeded: DDT (Allinea), OSS (Krell/Tri-Lab), Libra (LLNL), Cray Tools (Cray), Paraver (BSC), Vampir (TUD), TAU (UO), Scalasca (JSC)
  - Periscope (TUM) "only" ran at 8000
  - Most tools presented demos for 2000-4000 cores showing slides for the large case
  - TAU showed on-line(!) demo on 12,000 cores

Tools for HPC

- We **can** do performance analysis on the terascale, however…
  - Parallel Computing (PC?)
    might have reached the masses …

  - but remember, we do
    High Performance Computing (HPC!)

⇨ We need integrated teams / simulation labs / end stations / ..

⇨ To get integrated, customized tool support

⇨ Tools will no longer be “free” / analysis does cost resources

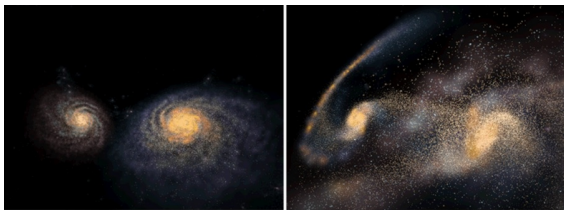
⇨ Tool community needs to build up interoperable, reusable tool components implementing the various basic technologies

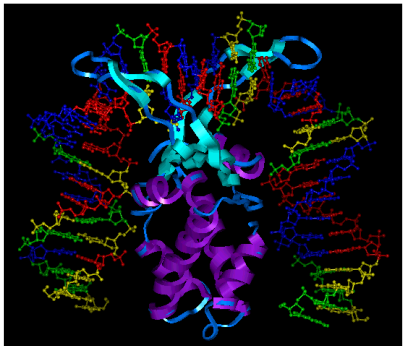
BACKUP
Very Quick Introduction to Parallel Programming and Performance Measurements

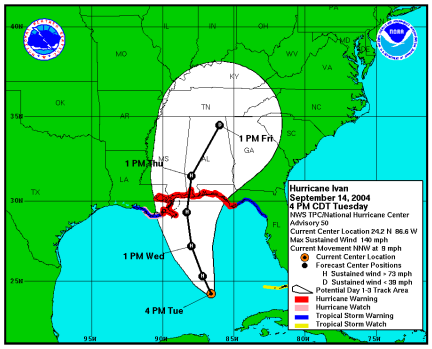
SC 2012 | Martin Schulz – Lawrence Livermore National Laboratory
Bernd Mohr – Jülich Supercomputing Centre
Brian Wylie – Jülich Supercomputing Centre

High-performance computing

- **Computer simulation augments theory and experiments**
  - Needed whenever real experiments would be too large/small, complex, expensive, dangerous, or simply impossible
  - Became third pillar of science
- **Computational science**
  - Multidisciplinary field that uses advanced computing capabilities to understand and solve complex problems
- Challenging applications
  - Protein folding
  - Climate / weather modeling
  - Astrophysics modeling
  - Nano-scale materials
  - . . .







⇨ **Realistic simulations need enormous computer resources (time, memory) !**

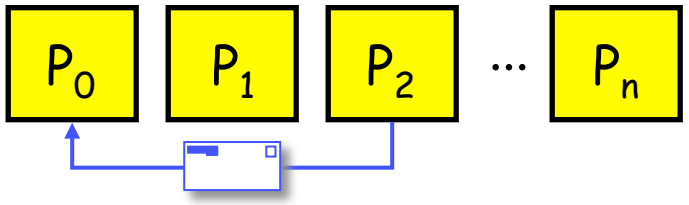
Programming Models: MPI

- MPI: **M**essage **P**assing **I**nterface
- De-facto **standard** message passing interface
  - MPI 1.0 in 1994
  - MPI 1.2 in 1997
  - MPI 2.0 in 1997
  - MPI 2.1 in 2008
  - MPI 2.2 in 2009
  - MPI 3.0 in 2012

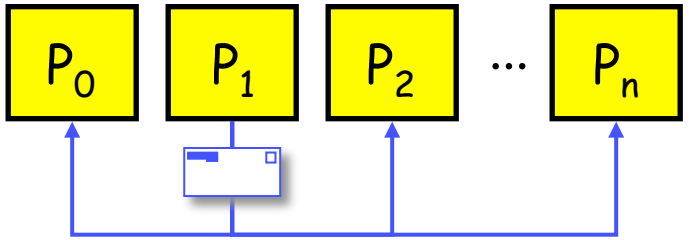
- **Library interface**
- Language bindings for Fortran, C, C++, [Java]
- Typically used in conjunction with SPMD programming style
- http://www.mpi-forum.org

MPI Functionality

- **Point-to-point** communication (between 2 processes)



- **Collective** communication (between a group of processes)



- **Barrier** synchronization
- Management of **communicators**, data types, topologies

- **One-sided** communication
- **Parallel I/O**
- F90 and C++ support
- Process creation

} MPI 2.0

Programming Models: OpenMP

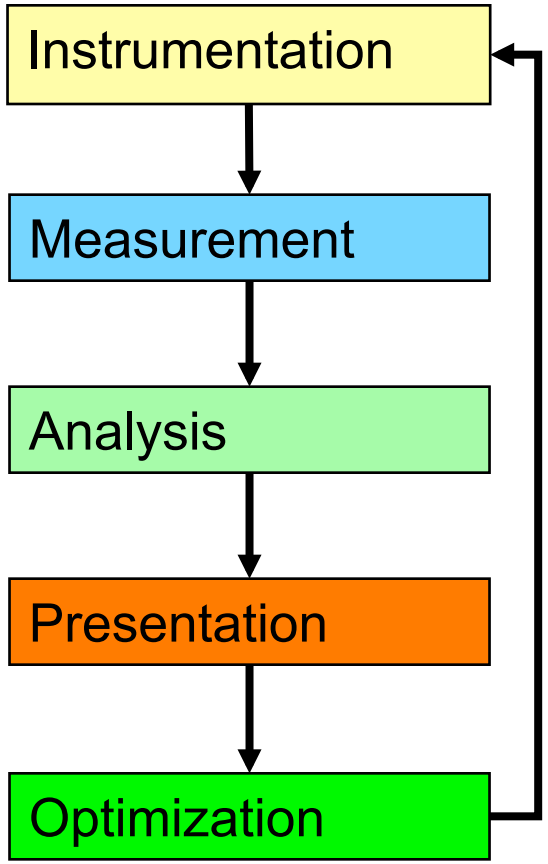
- OpenMP: **Open** specification for **M**ulti **P**rocessing
- De-facto **standard** programming interface for portable **shared memory** programming
  - ⇨ **Does NOT work on distributed memory systems!**
- Based on **Directives** for Fortran 77/90 and **pragmas** for C/C++, library routines and environment variables
- Explicit (not automatic) programming model
  - ⇨ **Does NOT check correctness of directives!**
  - OpenMP 1.0 in 1997 (Fortran) and 1998 (C/C++)
  - OpenMP 2.0 in 2000 (Fortran) and 2002 (C/C++)
  - OpenMP 2.5 in 2005 (C/C++/Fortran)
  - OpenMP 3.0 in 2008
  - OpenMP 3.1 in 2011
- http://www.openmp.org

OpenMP Functionality

- Directives/pragmas
  - **Parallel regions** (execute the same code in parallel)
  - **Parallel loops** (execute loop iterations in parallel)
  - **Parallel sections** (execute different sections in parallel)
  - Execution by exactly one single or master thread
  - Shared and private data
  - **Reductions**
  - **Synchronization** primitives (Barrier, Critical region, Atomic)

- New in OpenMP 3.0
  - Asynchronous **task** creation and execution

Performance Measurement Cycle



- Insertion of extra code (probes, hooks) into application
- Collection of data relevant to performance analysis
- Calculation of metrics, identification of performance problems
- Transformation of the results into a representation that can be easily understood by a human user
- Elimination of performance problems

Performance Measurement

- **Two dimensions**
  - **When** performance measurement is triggered
    - **Externally** (asynchronous) ⇨ indirect measurement
      - Sampling
        - » Timer interrupt
        - » Hardware counters overflow
    - **Internally** (synchronous) ⇨ direct measurement
      - Code instrumentation
        - » Automatic or manual instrumentation
  - **How** performance data is recorded
    - **Profile** ::= Summation of events over time
      - run time summarization (functions, call sites, loops, …)
    - **Trace file** ::= Sequence of events over time

Measurement Methods: Profiling I

- Recording of **aggregated information**
  - Time
  - Counts
    - Calls
    - Hardware counters
- **about program and system entities**
  - Functions, call sites, loops, basic blocks, …
  - Processes, threads

Measurement Methods: Tracing

- Recording **information about** significant points (**events**) during execution of the program
  - Enter/leave a code region (function, loop, …)
  - Send/receive a message ...
- Save information in **event record**
  - Timestamp, location ID, event type
  - plus event specific information
- **Event trace** := stream of event records sorted by time

- Can be used to reconstruct the **dynamic behavior**

  ⇨ Abstract execution model on level of defined events

Event tracing

Process A

```
void foo() {
  trc_enter("foo");
  ...
  trc_send(B);
  send(B, tag, buf);
  ...
  trc_exit("foo");
}
```

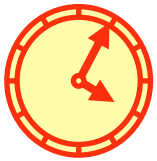
instrument

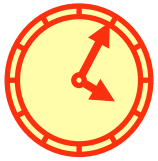
Process B

```
void bar() {
  trc_enter("bar");
  ...
  recv(A, tag, buf);
  trc_recv(A);
  ...
  trc_exit("bar");
}
```

MONITOR

MONITOR





synchronize(d)

Local trace A

...		
58	ENTER	1
62	SEND	B
64	EXIT	1
...		

1	foo
...	

Local trace B

...		
60	ENTER	1
68	RECV	A
69	EXIT	1
...		

1	bar
...	

Global trace

...			
58	A	ENTER	1
60	B	ENTER	2
62	A	SEND	B
64	A	EXIT	1
68	B	RECV	A
69	B	EXIT	2
...			

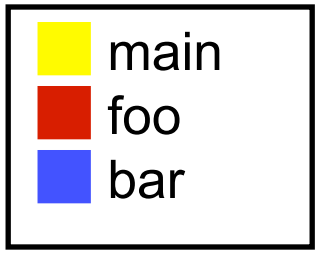
merge

unify

1	foo
2	bar
...	

Event Tracing: “Timeline” Visualization

1	foo
2	bar
3	...



...			
58	A	ENTER	1
60	B	ENTER	2
62	A	SEND	B
64	A	EXIT	1
68	B	RECV	A
69	B	EXIT	2
...