SCALASCA:
Scalable performance **a**nalysis of **la**rge-**sc**ale parallel **a**pplications

Brian J.N. Wylie

John von Neumann Institute for Computing
Forschungszentrum Jülich
B.Wylie@fz-juelich.de

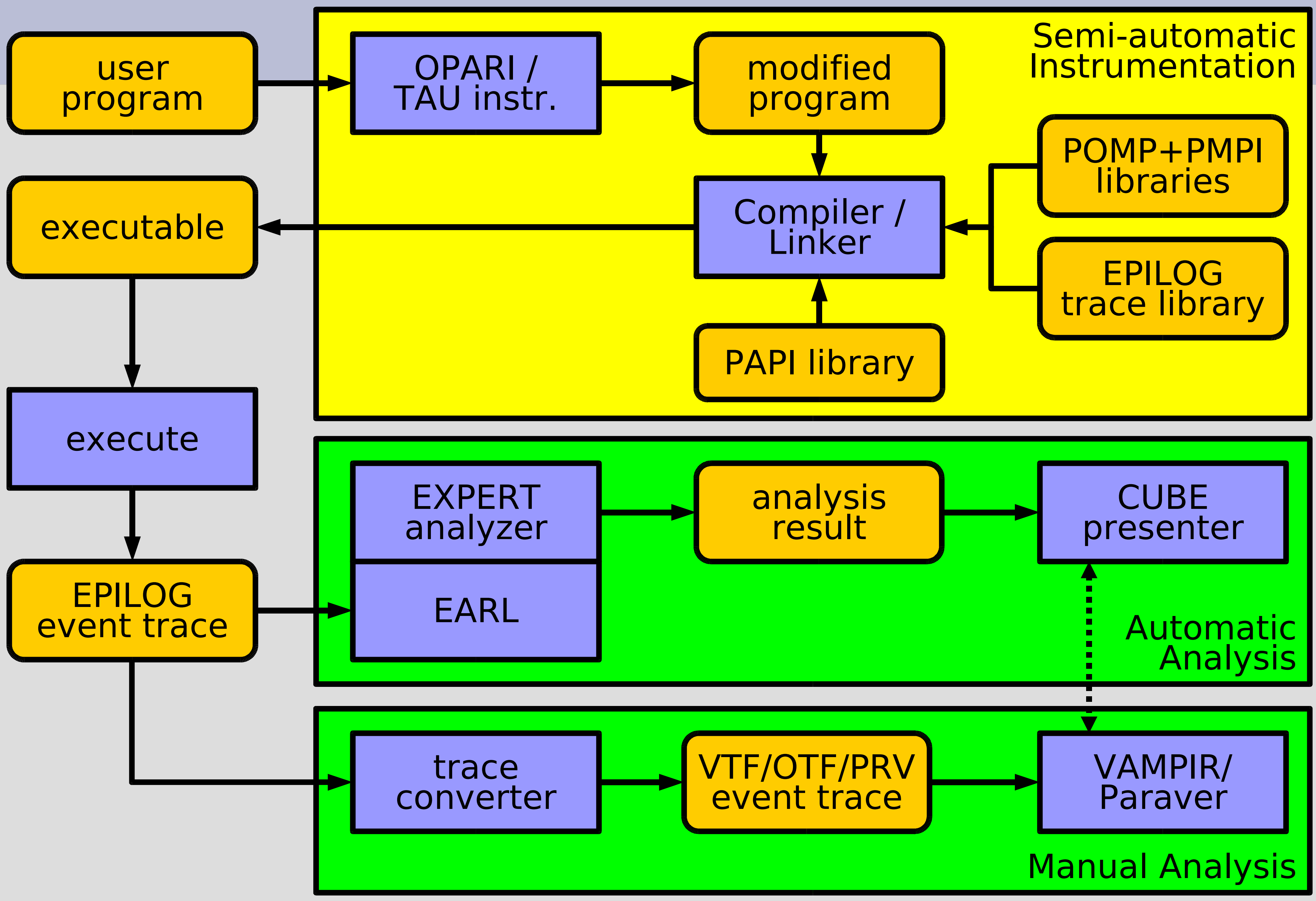
Outline

- KOJAK automated event tracing & analysis
  - New performance tool requirements
  - Successor project focussing on scalability
- Scalable runtime measurement
  - Usability & scalability improvements
  - Integration of summarisation & selective tracing
- Scalable measurement analysis
  - Process local traces in parallel
  - Parallel event replay impersonating target
- Demonstration of improved scalability
  - SMG2000 on IBM BlueGene/L & Cray XT3
- Summary

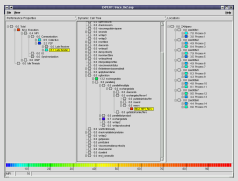
The KOJAK project

- Kit for Objective Judgement & Automatic Knowledge-based detection of bottlenecks
  - Forschungszentrum Jülich
  - University of Tennessee
- Long-term goals
  - Design & implementation of a portable, generic & automatic performance analysis environment
- Focus
  - Event tracing & inefficiency pattern search
  - Parallel computers with SMP nodes
  - MPI, OpenMP & SHMEM programming models

KOJAK architecture

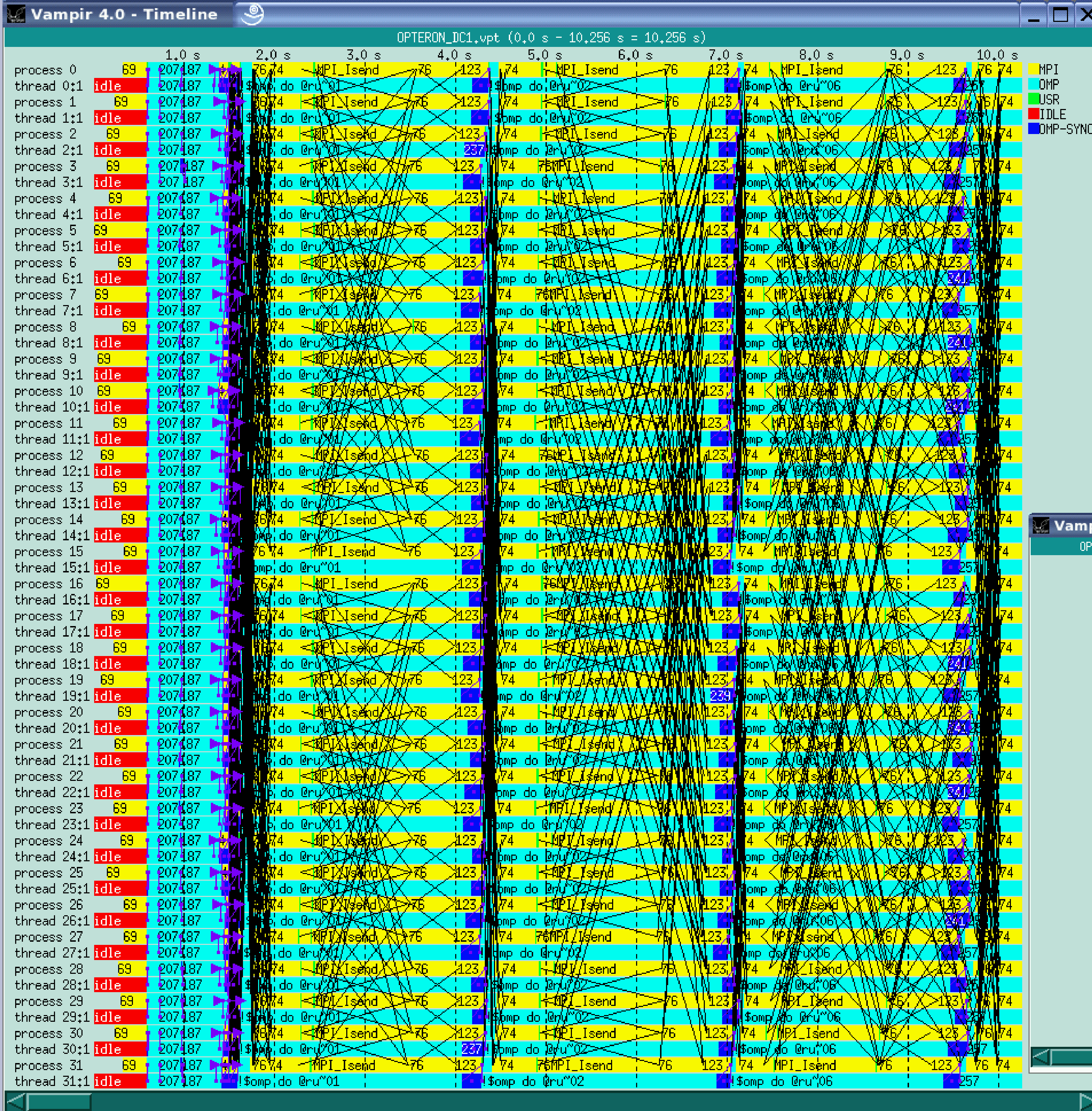


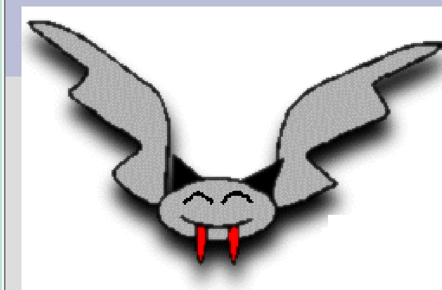
KOJAK tool components

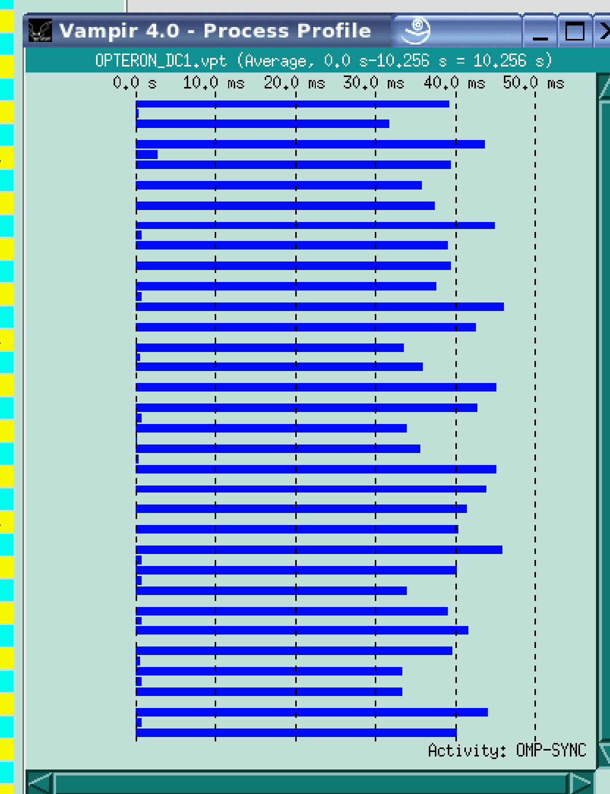


- Instrument user application
  - EPILOG tracing library API calls
  - User functions and regions:
    - Automatically by TAU source instrumenter
    - Automatically by compiler (GCC,Hitachi,IBM,NEC,PGI,Sun)
    - Manually using POMP directives
  - MPI calls: Automatic PMPI wrapper library
  - OpenMP: Automatic OPARI source instrumentor
  - Record hardware counter metrics via PAPI
- Analyze measured event trace
  - Automatically with EARL-based EXPERT trace analyzer and CUBE analysis result browser
  - Manually with VAMPIR (via EPILOG-VTF3 converter)

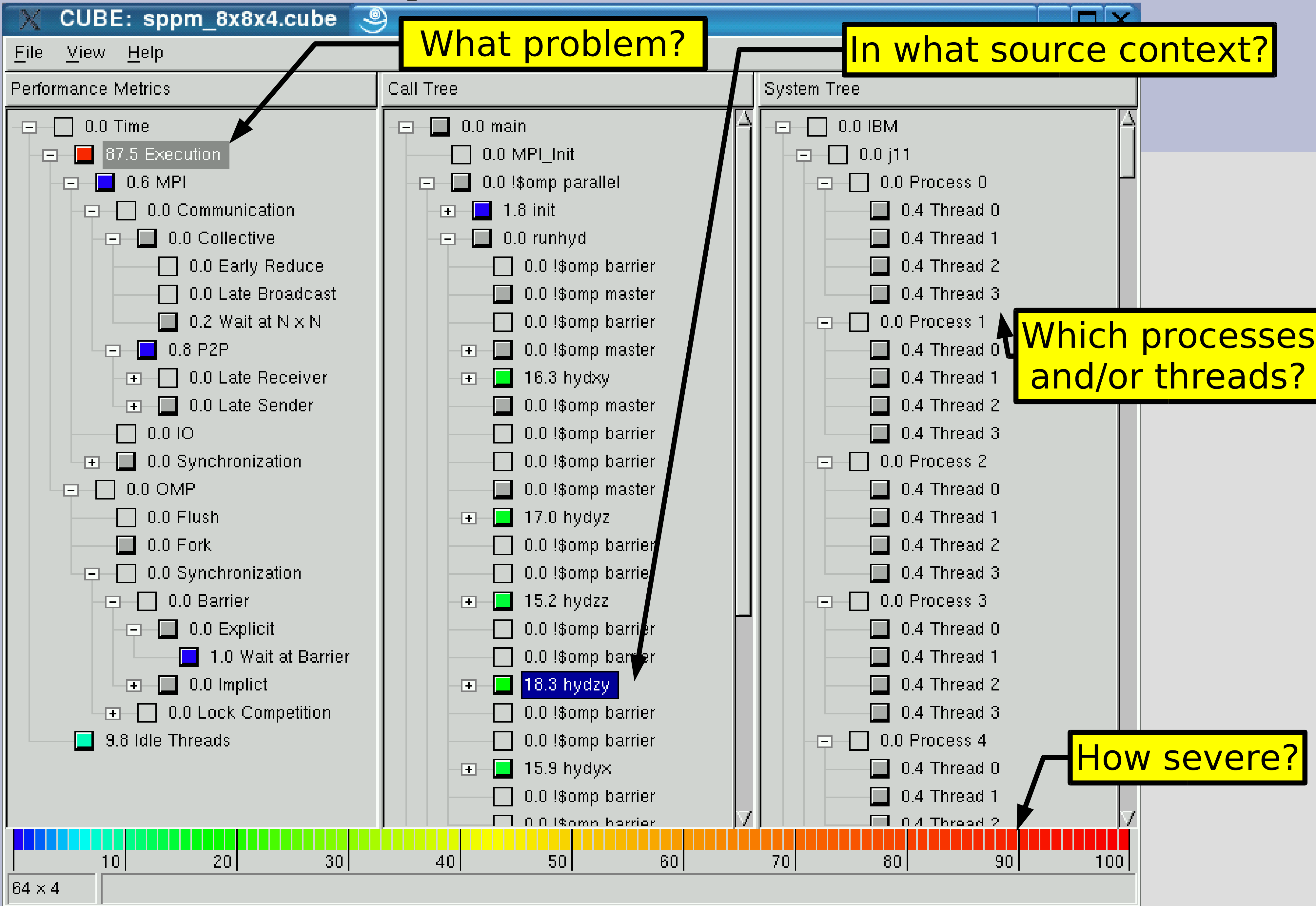
KOJAK/ VAMPIR







CUBE analysis browser



KOJAK supported platforms

- Full support for instrumentation, measurement, and automatic analysis
  - Linux IA32, IA64 & IA32_64 clusters (incl. XD1)
  - IBM AIX POWER3 & 4 clusters (SP2, Regatta)
  - Sun Solaris SPARC & x64 clusters (SunFire, …)
  - SGI Irix MIPS clusters (Origin 2K, 3K)
  - DEC/HP Tru64 Alpha clusters (Alphaserver, …)

- Instrumentation and measurement only
  - IBM BlueGene/L
  - Cray XT3, Cray X1, Cray T3E
  - Hitachi SR-8000, NEC SX

The SCALASCA project

Scalable performance **a**nalysis
of **la**rge-**sc**ale parallel **a**pplications

- Scalable performance analysis
  - Scalable performance measurement collection
  - Scalable performance analysis & presentation
- KOJAK follow-on research project
  - funded by German Helmholtz Association (HGF) for 5 years (2006-2010)
- Ultimately to support full range of systems
  - Initial focus on MPI on BlueGene/L

SCALASCA design overview

- Improved integration and automation
  - Instrumentation, measurements & analyses
- Parallel trace analysis based on replay
  - Exploit distributed processors and memory
  - Communication replay with measurement data
- Complementary runtime summarisation
  - Low-overhead execution callpath profile
  - Totalisation of local measurements
- Feedback-directed selective event tracing and instrumentation configuration
  - Optimise subsequent measurement & analysis

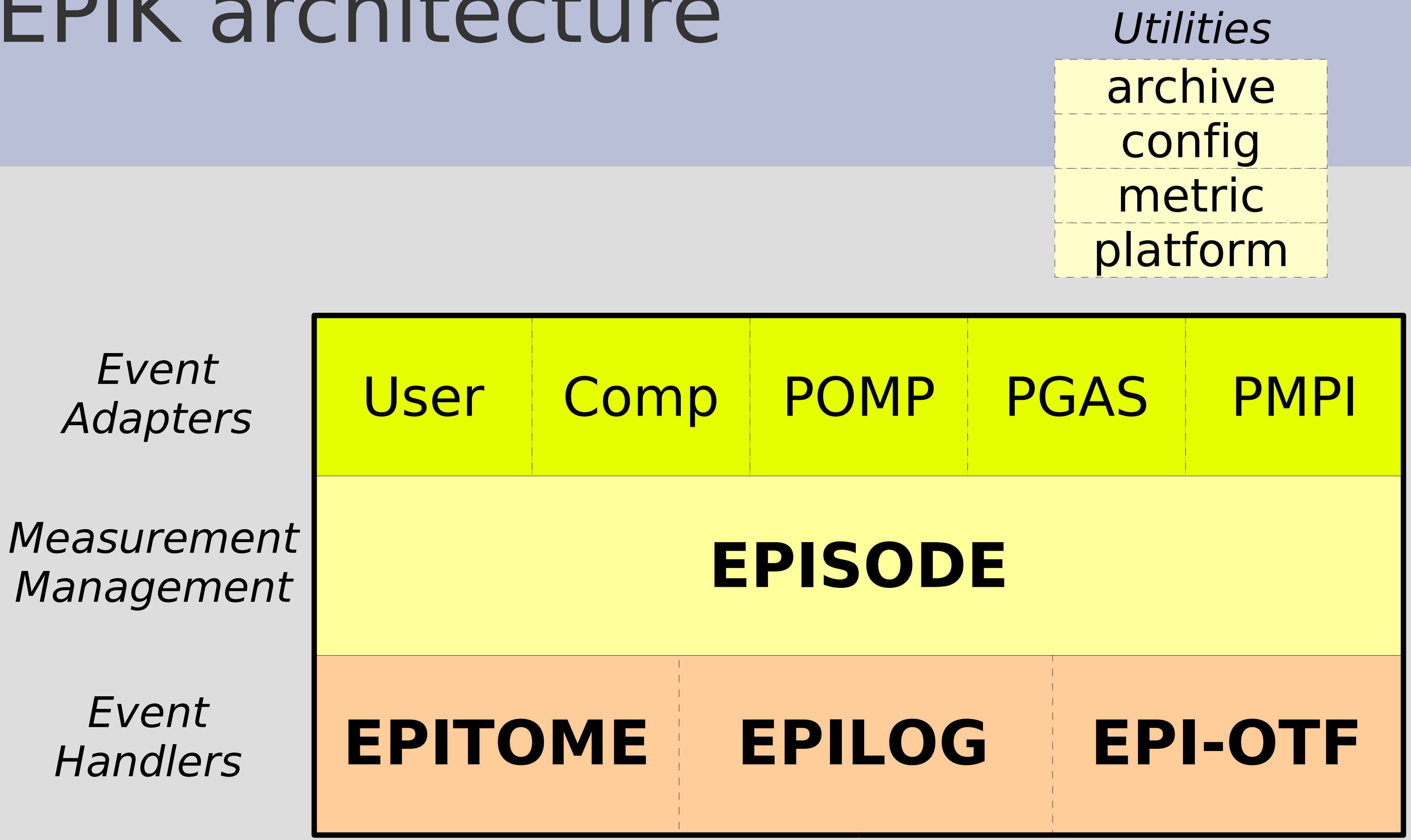
SCALASCA Phase 1

- Exploit existing OPARI instrumenter
- Re-develop measurement runtime system
  - Ameliorate scalability bottlenecks
  - Improve usability and adaptability
- Develop new parallel trace analyser for MPI
  - Use parallel processing & distributed memory
  - Analysis processes mimic subject application's execution by replaying events from local traces
  - Gather distributed analyses
- Direct on-going CUBE re-development
  - Library for incremental analysis report writing

EPIK measurement system

- Revised runtime system architecture
  - Based on KOJAK's EPILOG runtime system and associated tools & utilities
  - EPILOG name retained for tracing component
- Modularised to support both event tracing and complementary runtime summarisation
  - Sharing of user/compiler/library event adapters and measurement management infrastructure
- Optimised operation for scalability
- Improved usability and adaptability

EPIK architecture



EPIK components

- Integrated runtime measurement library incorporating
  - EPIK: Event preparation interface kit
    - Adapters for user/compiler/library instrumentation
    - Utilities for archive management, configuration, metric handling and platform interfacing
  - EPISODE: Management of measurements for processes & threads, attribution to events, and direction to event handlers
  - EPILOG: Logging library & trace-handling tools
  - EPI-OTF: Tracing library for OTF [VAMPIR]
  - EPITOME: Totalised metric summarisation

EPIK scalability improvements

- Merging of event traces only when required
  - Parallel replay uses only local event traces
  - Avoids sequential bottleneck and file re-writing
- Separation of definitions from event records
  - Facilitates global unification of definitions and creation of (local−global) identifier mappings
  - Avoids extraction/re-write of event traces
  - Can be shared with runtime summarisation
- On-the-fly identifier re-mapping on read
  - Interpret local event traces using identifier mappings for global analysis perspective

EPIK usability improvements

- Dedicated experiment archive directory
  - Organises measurement and analysis data
  - Facilitates experiment management & integrity
  - Opacity simplifies ease-of-use
- File compression/decompression
  - Processing overheads more than compensated by reduced file reading & writing times
  - Bonus in form of smaller experiment archives
- Runtime generation of OTF traces [MPI]
  - Alternative to post-mortem trace conversion, developed in collaboration with TU Dresden ZIH

Automatic analysis process

- Scans event trace sequentially
  - If trigger event: call search function of pattern
  - If match:
    - Determine call path and process/thread affected
    - Calculate **severity** ::= percentage of total execution time "lost" due to pattern
- Analysis result
  - For each pattern: distribution of severity
    - Over all call paths
    - Over machine / nodes / processes / threads
  - CUBE presentation via 3 linked tree browsers
    - Pattern hierarchy (general ⇨ specific problem)
    - Region / Call tree
    - Location hierarchy (Machine/Node, Process/Thread)

Analysis patterns (examples)

Profiling Patterns

Total	Total time consumed
Execution	User CPU execution time
MPI	MPI API calls
OMP	OpenMP runtime
Idle threads	Unused CPU time during sequential execution

Complex Patterns

MPI/ Late Sender	Receiver blocked prematurely
MPI/ Late Receiver	Sender blocked prematurely
Messages in wrong order	Waiting for a message from a particular sender while other messages already available in queue
MPI/ Wait at N x N	Waiting for last participant in N-to-N operation
MPI/ Late Broadcast	Waiting for sender in broadcast operation
OMP/ Wait at Barrier	Waiting in explicit or implicit barriers

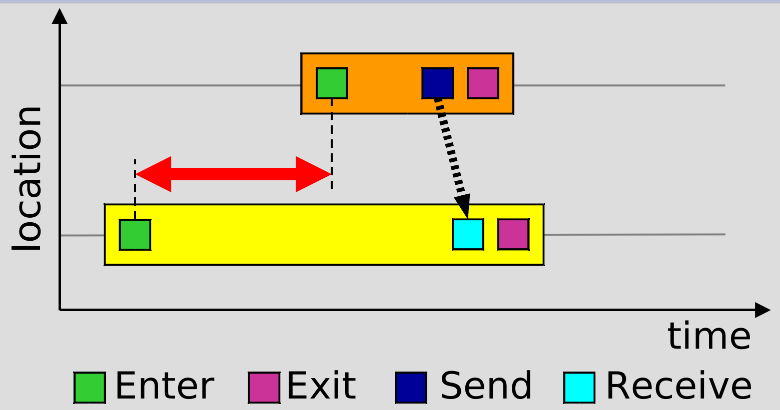
Initial implementation limitations

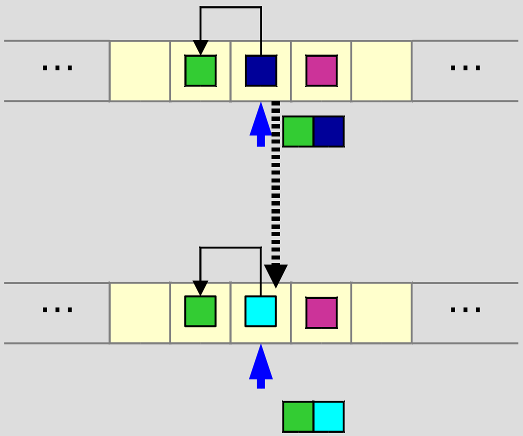
- Event traces must be merged in time order
  - Merged trace file is large and unwieldy
  - Trace read and re-write strains filesystem
  - Processing time scales very poorly
- Sequential scan of entire event trace
  - Processing time scales poorly with trace size
  - Requires a windowing and re-read strategy, for working set larger than available memory
- Only practical for short interval traces and/or hundreds of processes/threads

Parallel pattern analysis

- Analyse individual rank trace files in parallel
  - Exploits target system's distributed memory & processing capabilities
  - Often allows whole event trace in main memory
- *Parallel Replay* of execution trace
  - Parallel traversal of event streams
  - Replay communication with similar operation
  - Event data exchange at synchronisation points of target application
- Gather & combine each process' analysis
  - Master writes integrated analysis report

Example performance property: *Late Sender*





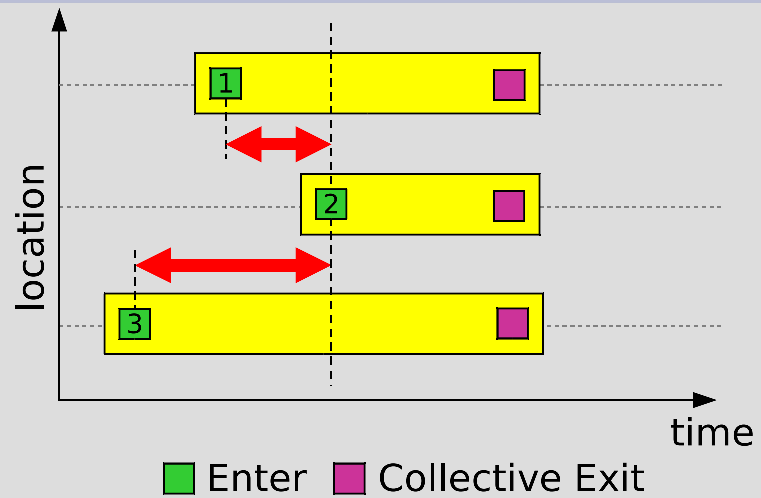
Sender:

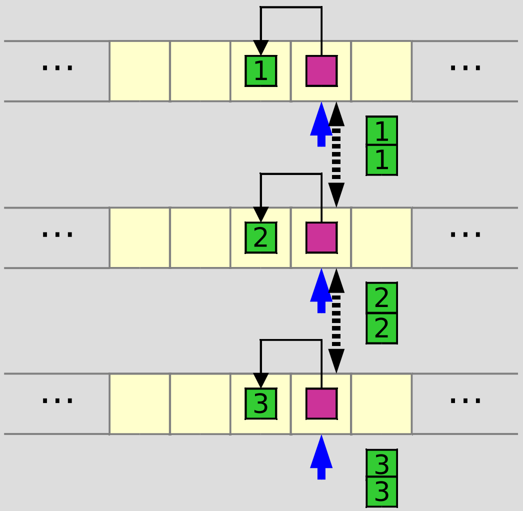
- Triggered by send event
- Determine enter event
- Send both events to receiver

Receiver:

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

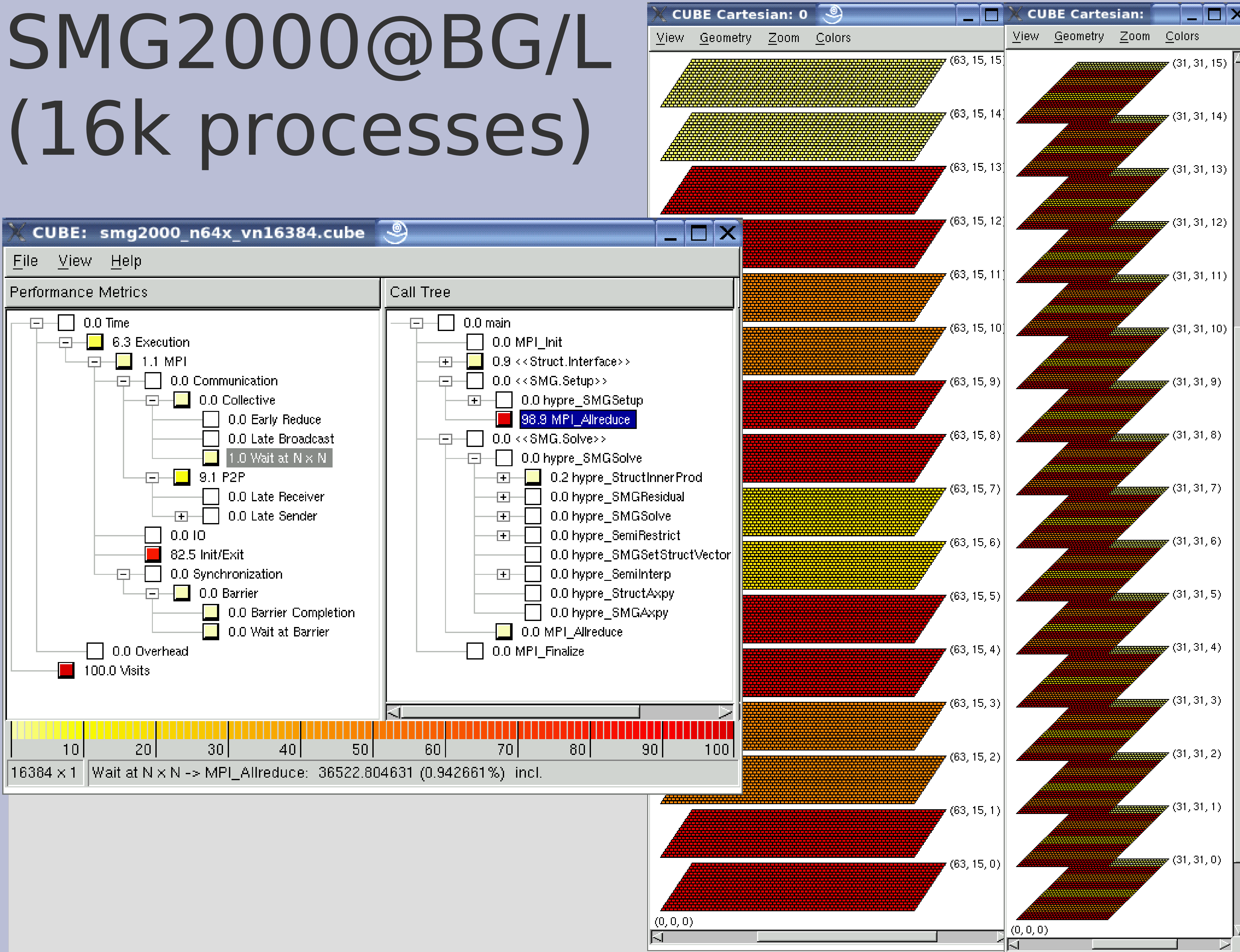
Example performance property: *Wait at N x N*





- Wait time due to inherent synchronisation in N-to-N operations (e.g., MPI_Allreduce)
  - Triggered by collective exit event
  - Determine enter events
  - Distribute latest enter event (max-reduction)
  - Calculate & store local waiting time

SMG2000@BG/L (16k processes)



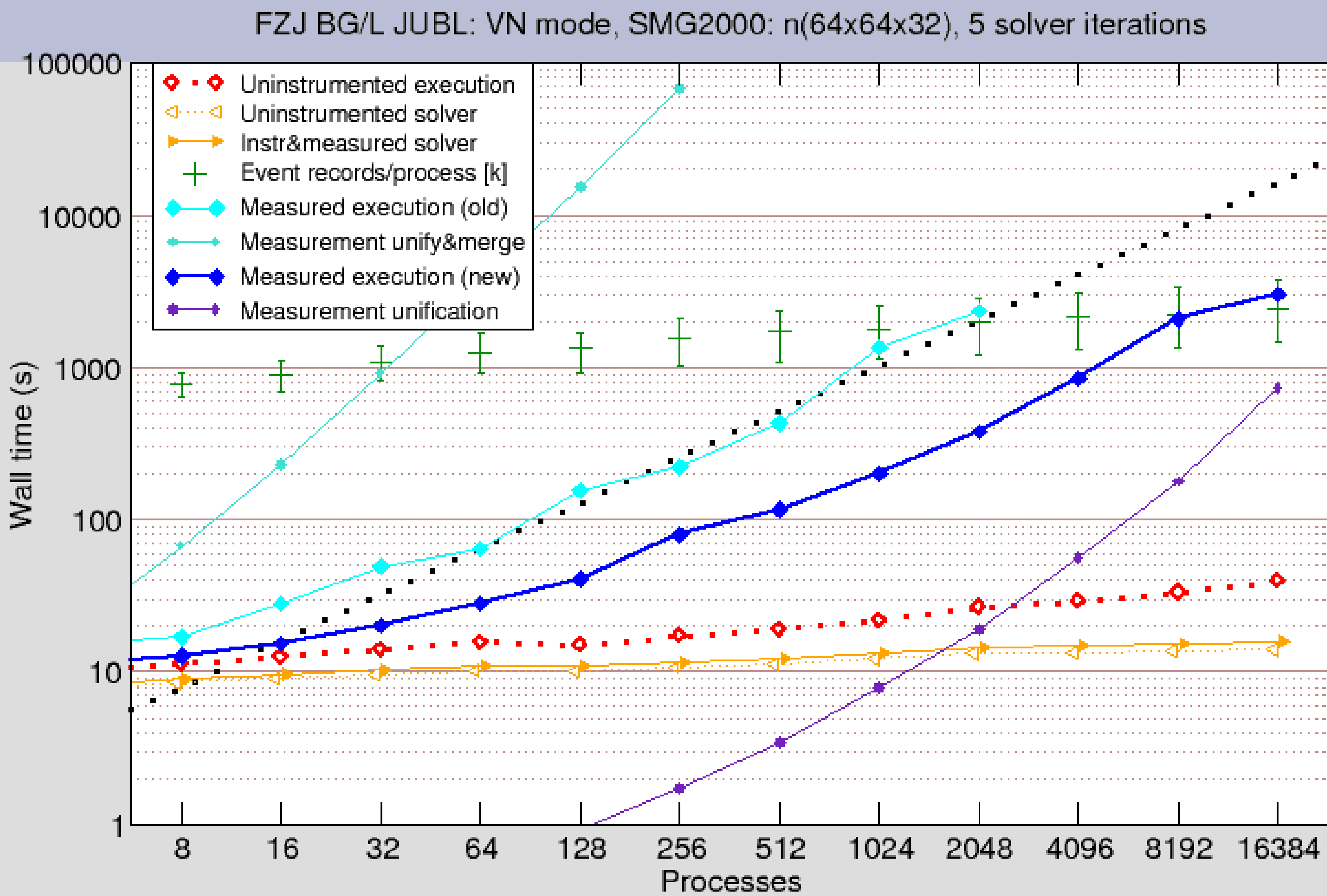
Jülicher BlueGene/L (JUBL)

- 8,192 dual-core PowerPC compute nodes
- 288 dual-core PowerPC I/O nodes [GPFS]
- p720 service & login nodes (8x Power5)

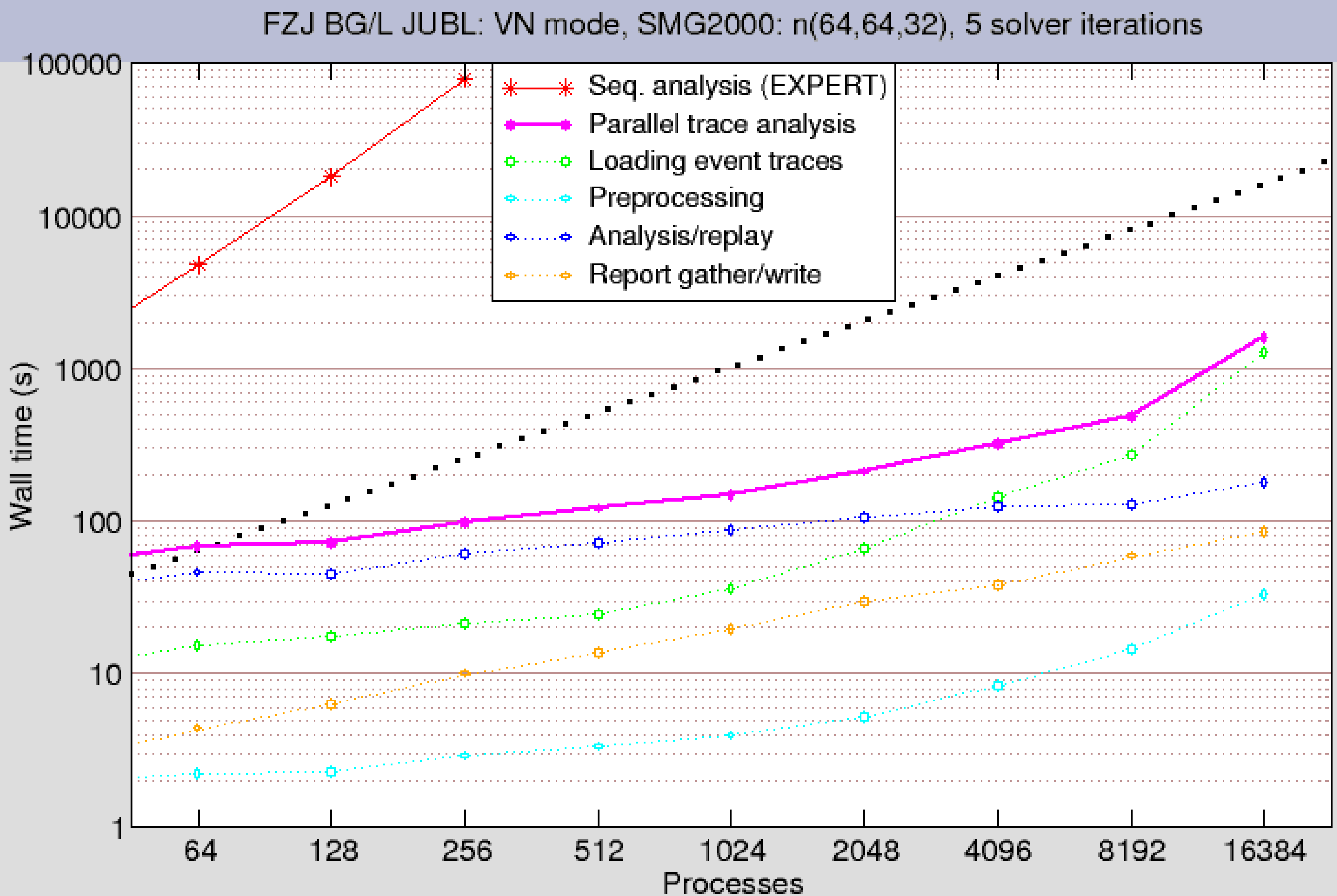
Scalability validation

- 16,384 MPI processes on Jülicher BlueGene/L
  - Running ASC SMG2000 benchmark [64x64x32]
  - Fixed problem size per process: weak scaling
- Traces collected in 100MB memory buffers written directly into experiment archive
  - 40,000 million event records
  - 100GB of compressed event trace data
  - <15% measurement dilation
- Early analyser prototype unified identifiers, replayed events in parallel (on the same system), and produced analysis report
  - Sequential analysis impractical at this scale!

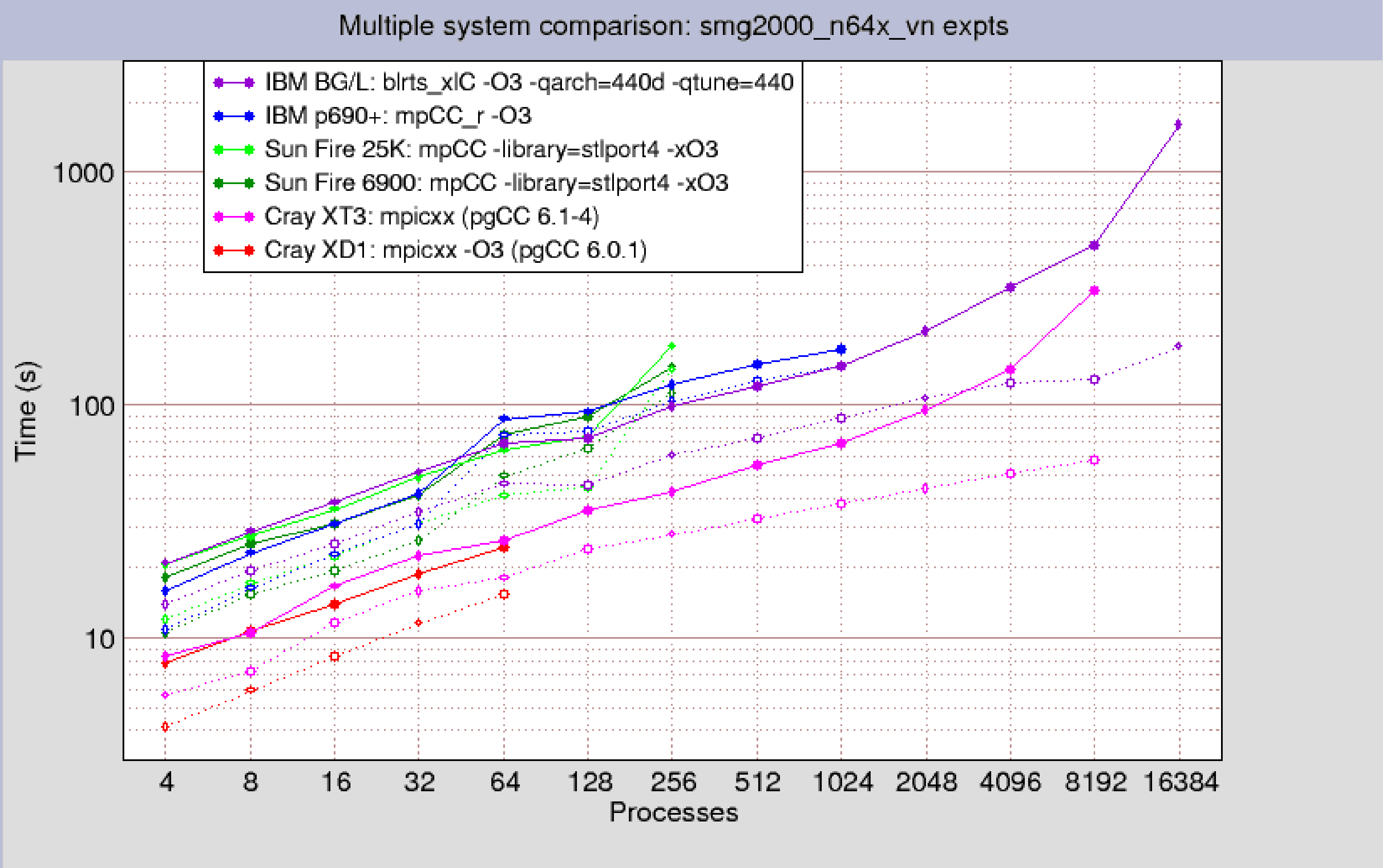
Measurement: SMG2000@BG/L



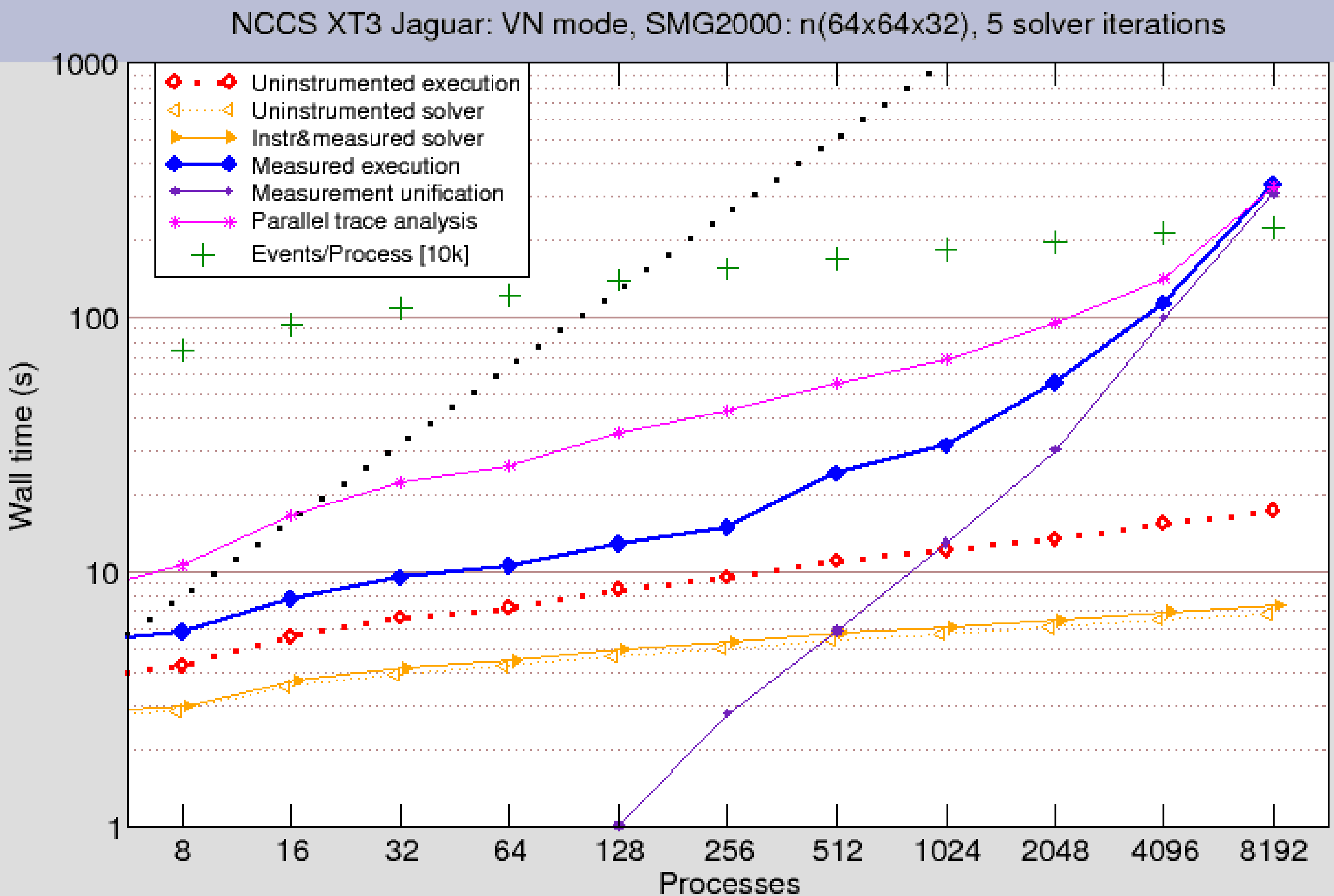
Scout analysis: SMG2000@BG/L



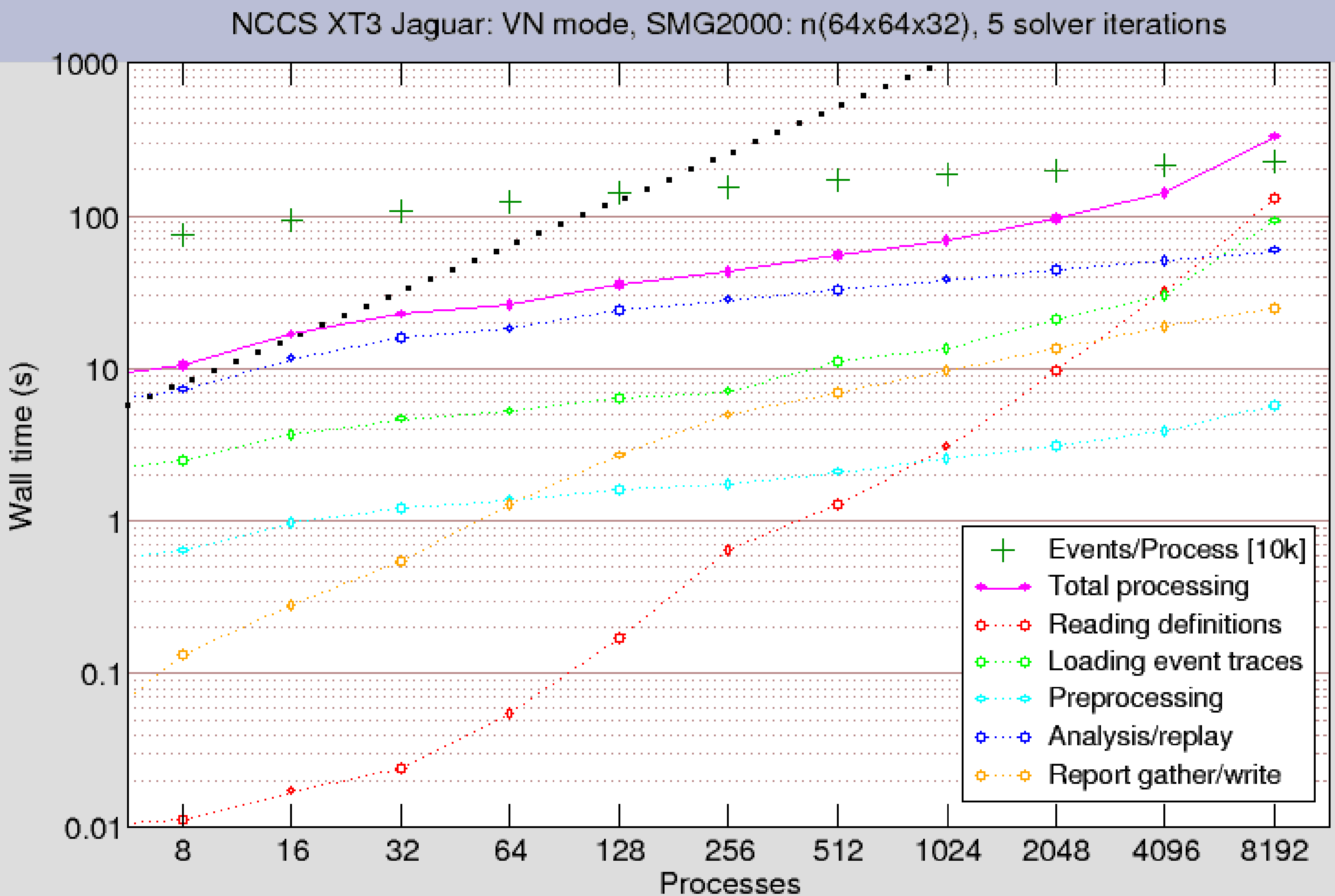
Scout analysis: SMG2000



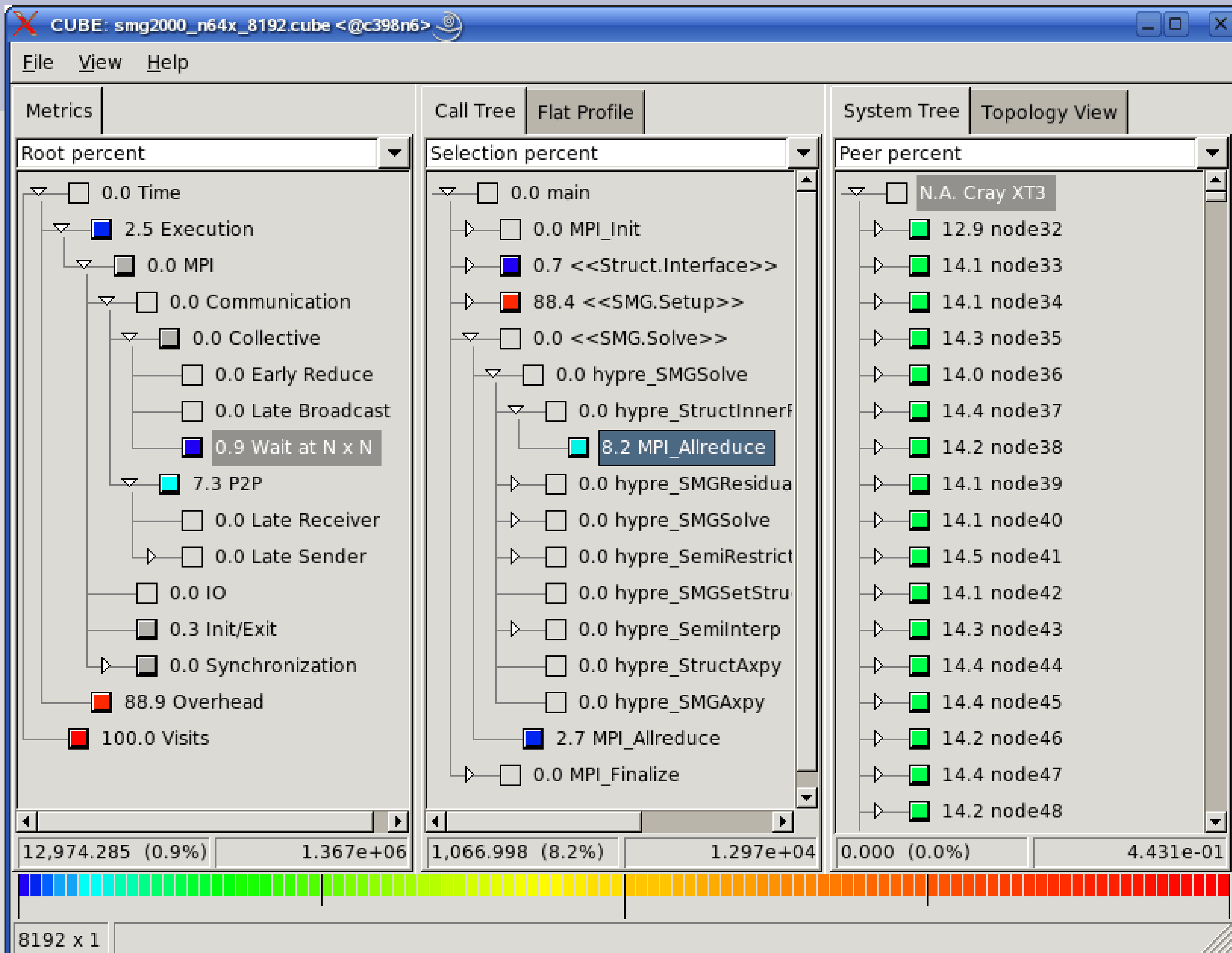
Measurement: SMG2000@XT3



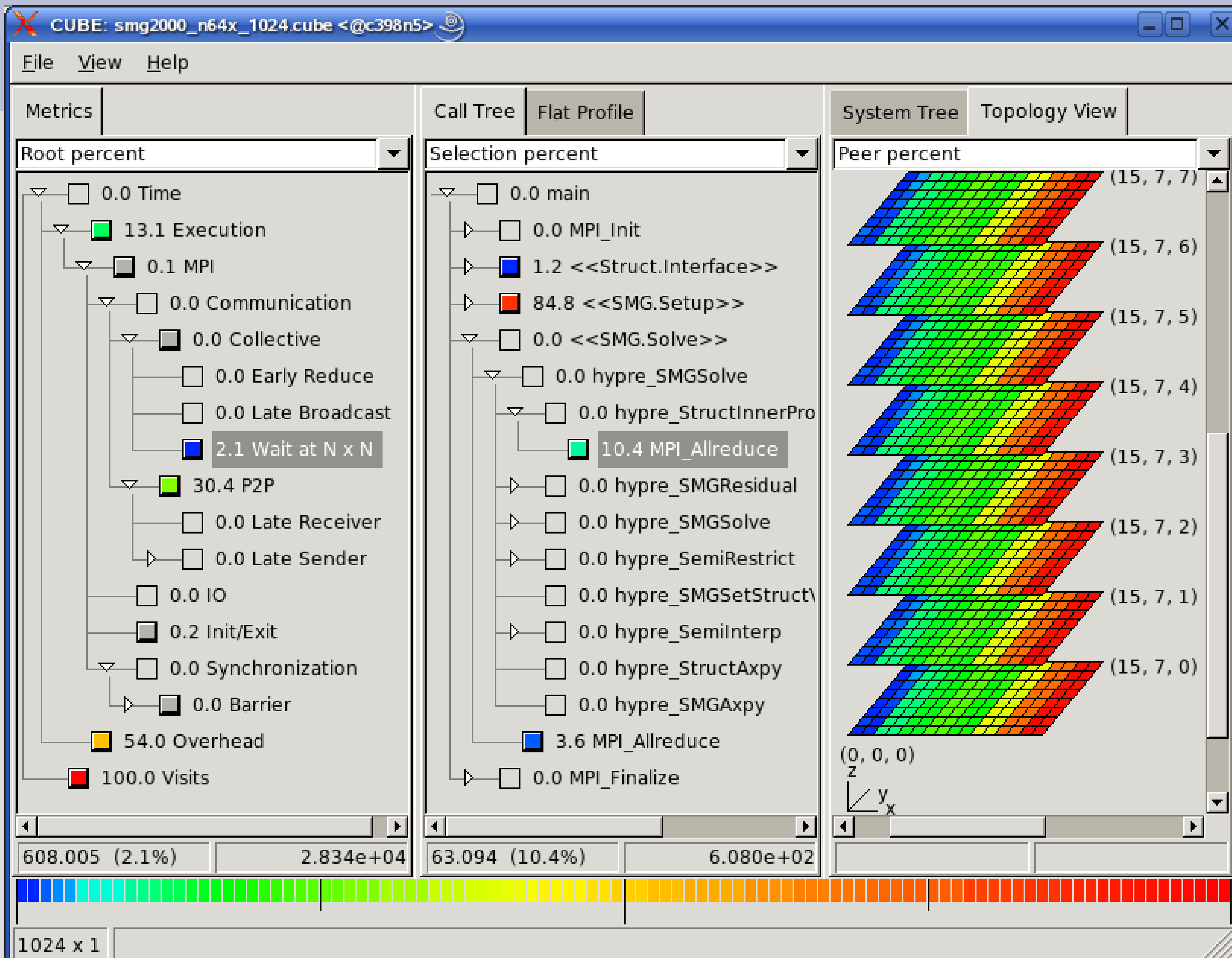
Scout analysis: SMG2000@XT3



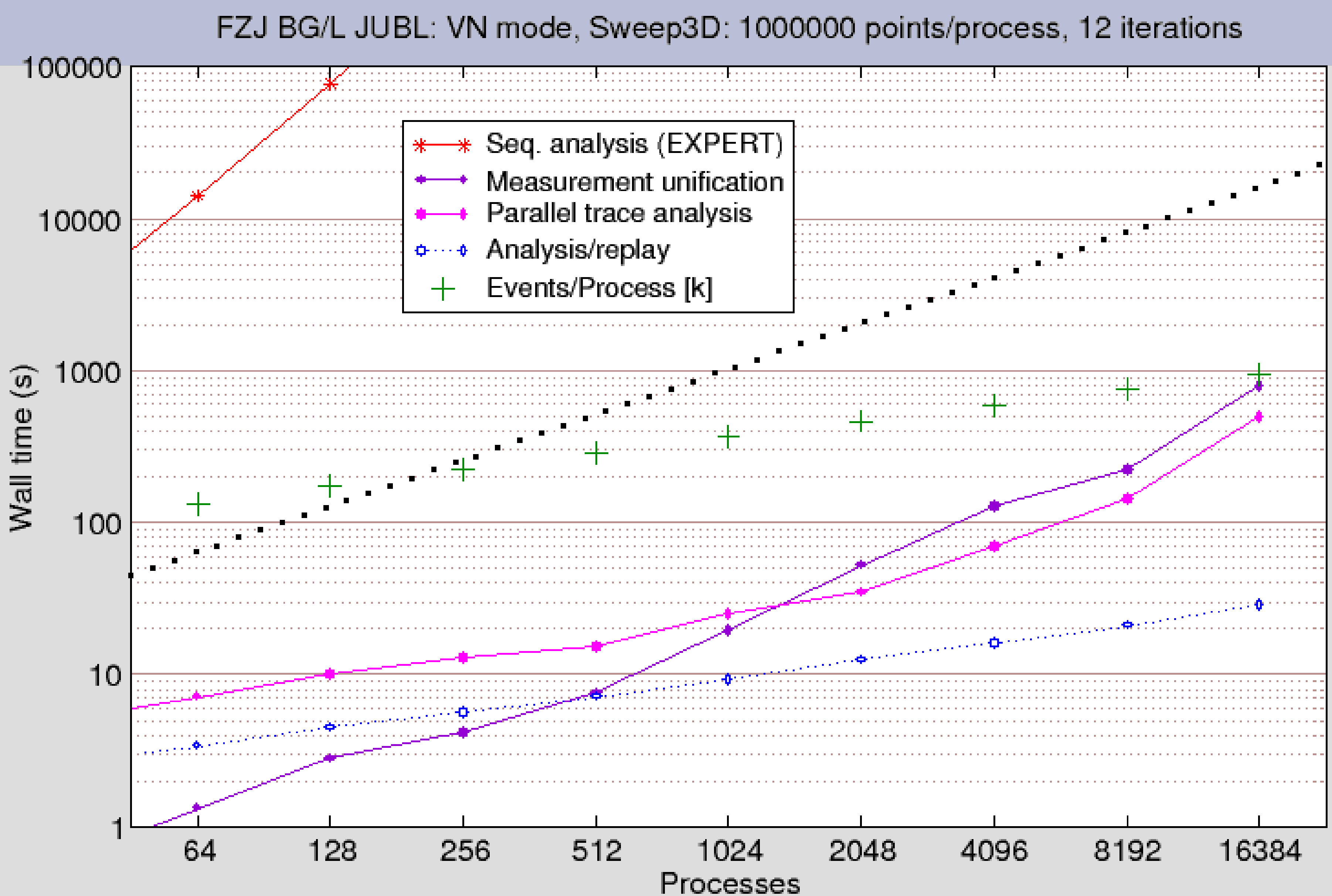
SMG2000@XT3 (8192 processes)



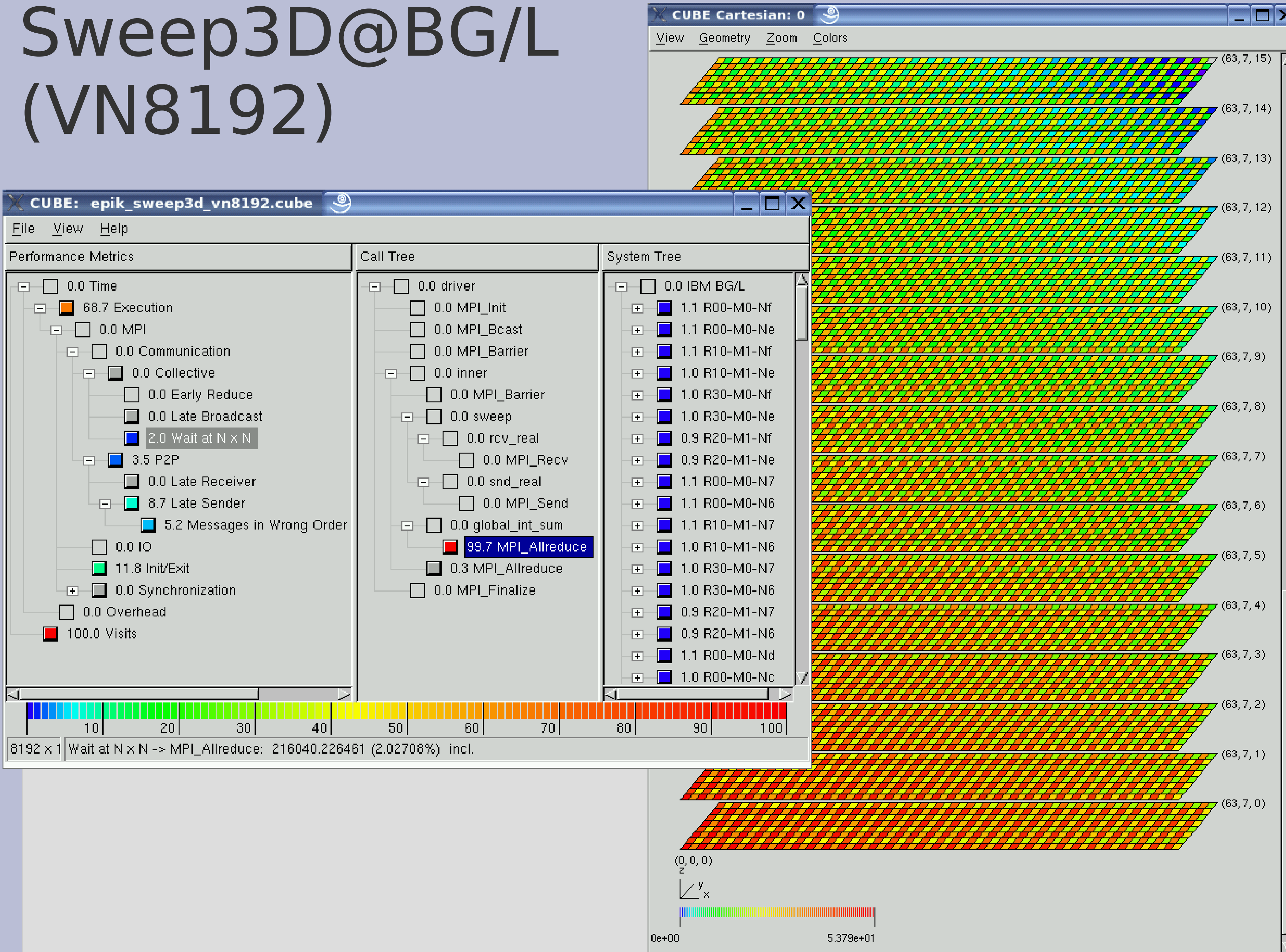
SMG2000@XT3 (1024 processes)



Scout analysis: Sweep3D@BG/L



Sweep3D@BG/L (VN8192)



SCALASCA work in progress

- Parallel/distributed analysis infrastructure
  - Runtime unification of local identifiers
- Prepare a technology preview release
  - Target: Dec 2006

- Runtime callpath tracking
  - Callpath measurement summarisation
- Generalise parallel replay/analysis
  - OpenMP (and OMP/MPI hybrid), MPI-2 RMA, ...
- Improving runtime configurability
- Improving analysis explorer [CUBE3]

SCALASCA future plans

- Develop selective source instrumenter
- Develop selective runtime event tracing
  - start-up and/or during execution
  - e.g., communication events
- Feedback-directed configuration of instrumentation and/or measurement
  - based on profile and/or trace analysis
- Support for PGAS programming paradigms
  - SHMEM, GA, CAF, UPC, ...
- Scalable analysis data structures (cCCGs)
- Extend analyses
- ...

Summary

- KOJAK supports automated execution analysis of most important HPC/cluster platforms, program languages & paradigms
- SCALASCA is investigating improvements which primarily focus on scalability
  - Enhanced trace collection & parallelised analysis
  - Scaling demonstrated to 16k MPI processes
- Performance analysis previously impractical at extreme scale is being made accessible

Further information

Automatic Performance Analysis with KOJAK

- available under BSD open-source licence
- sources, documentation & publications:
  http://www.fz-juelich.de/zam/kojak/
- mailto: kojak@fz-juelich.de

Scalable performance **a**nalysis of **la**rge-**sc**ale parallel **a**pplications

http://www.scalasca.org/

- mailto: scalasca@fz-juelich.de

References

- *Automatic performance analysis of hybrid MPI/OpenMP applications*,
  Wolf & Mohr, J. Systems Arch. 49(10-11), Nov. 2003
- *Large event traces in parallel performance analysis*,
  Wolf et al, Proc. CACS, LNI P-81, 2006
- *Integrated runtime measurement summarisation and selective event tracing*,
  Wylie et al, Proc. PARA'06 (to appear)
- *A platform for scalable parallel trace analysis*,
  Geimer et al, Proc. PARA'06 (to appear)
- *Scalable parallel trace-based performance analysis*,
  Geimer et al, Proc. EuroPVM/MPI'06, LNCS 4192

KOJAK/SCALASCA team

- Felix Wolf
  - RWTH Aachen Junior Professor
- Daniel Becker
- Christoph Geile
- Markus Geimer
- Björn Kuhlmann
- Bernd Mohr
- Brian Wylie