Scalable performance analysis of large-scale parallel applications

Brian Wylie & Markus Geimer
Jülich Supercomputing Centre
scalasca@fz-juelich.de

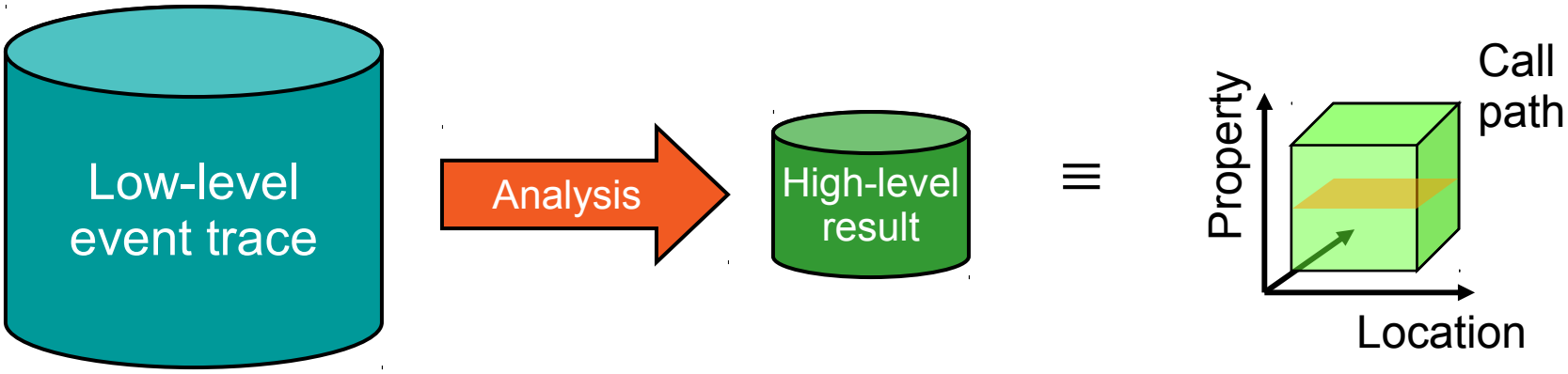
October 2010

Performance analysis, tools & techniques

- Profile analysis
  - Summary of aggregated metrics
    - per function/callpath and/or per process/thread
  - Most tools (can) generate and/or present such profiles
    - but they do so in *very* different ways, often from event traces!
  - e.g., gprof, mpiP, ompP, ***Scalasca***, TAU, Vampir, ...
- Time-line analysis
  - Visual representation of the space/time sequence of events
  - Requires an execution trace
  - e.g., Vampir, Paraver, JumpShot, Intel TAC, Sun Studio, ...
- Pattern analysis
  - Search for event sequences characteristic of inefficiencies
  - Can be done manually, e.g., via visual time-line analysis
  - or automatically, e.g., KOJAK, ***Scalasca***, Periscope, ...

Automatic trace analysis

- Idea
  - Automatic search for patterns of inefficient behaviour
  - Classification of behaviour & quantification of significance



  - Guaranteed to cover the entire event trace
  - Quicker than manual/visual trace analysis
  - Parallel replay analysis exploits memory & processors to deliver scalability

The Scalasca project

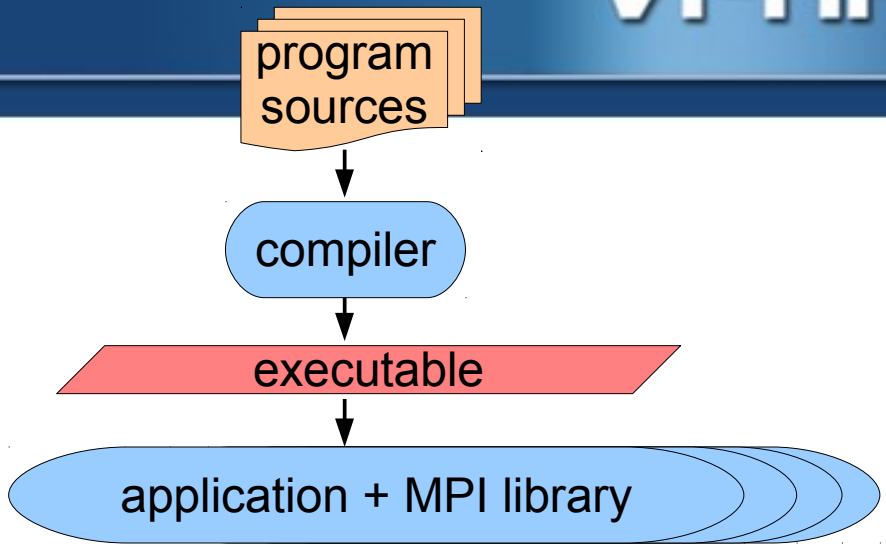
- Overview
  - Helmholtz Initiative & Networking Fund project started in 2006
  - Headed by Prof. Felix Wolf (JSC/RWTH/GRS-Sim)
  - Follow-up to pioneering KOJAK project (started 1998)
    - Automatic pattern-based trace analysis
- Objective
  - Development of a **scalable** performance analysis toolset
  - Specifically targeting **large-scale** parallel applications
    - such as those running on BlueGene/P or Cray XT with 10,000s to 100,000s of processes
- Latest release November 2010: Scalasca v1.3.2
  - Download from www.scalasca.org
  - Available on POINT/VI-HPS Parallel Productivity Tools DVD

Scalasca features

- Open source, New BSD license
- Portable
  - IBM BlueGene P & L, IBM SP & blade clusters, Cray XT, SGI Altix, NEC SX, SiCortex, Solaris & Linux clusters, ...
- Supports parallel programming paradigms & languages
  - MPI, OpenMP & hybrid OpenMP/MPI
  - Fortran, C, C++
- Integrated instrumentation, measurement & analysis toolset
  - Automatic and/or manual customizable instrumentation
  - Runtime summarization (aka profiling)
  - Automatic event trace analysis
  - Analysis report exploration & manipulation

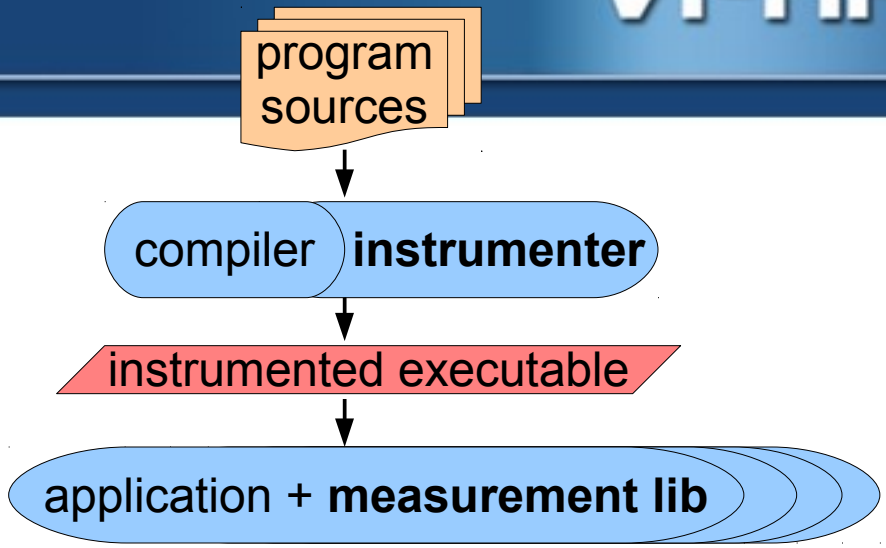
Generic MPI application build & run

- Application code compiled & linked into executable using MPICC/CXX/FC
- Launched with MPIEXEC
- Application processes interact via MPI library



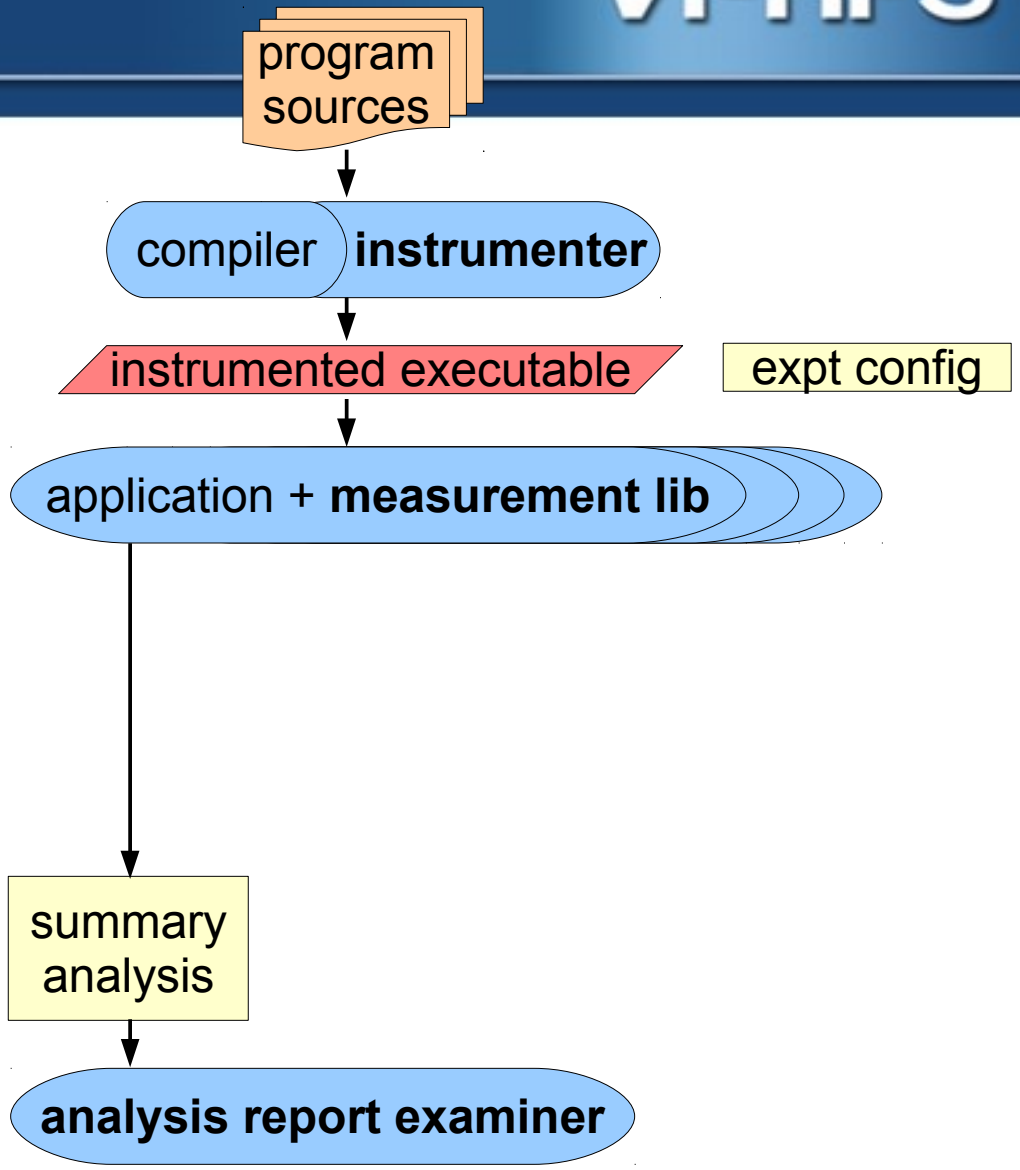
Application instrumentation

- Automatic/manual code instrumenter
- Program sources processed to add instrumentation and measurement library into application executable
- Exploits MPI standard profiling interface (PMPI) to acquire MPI events



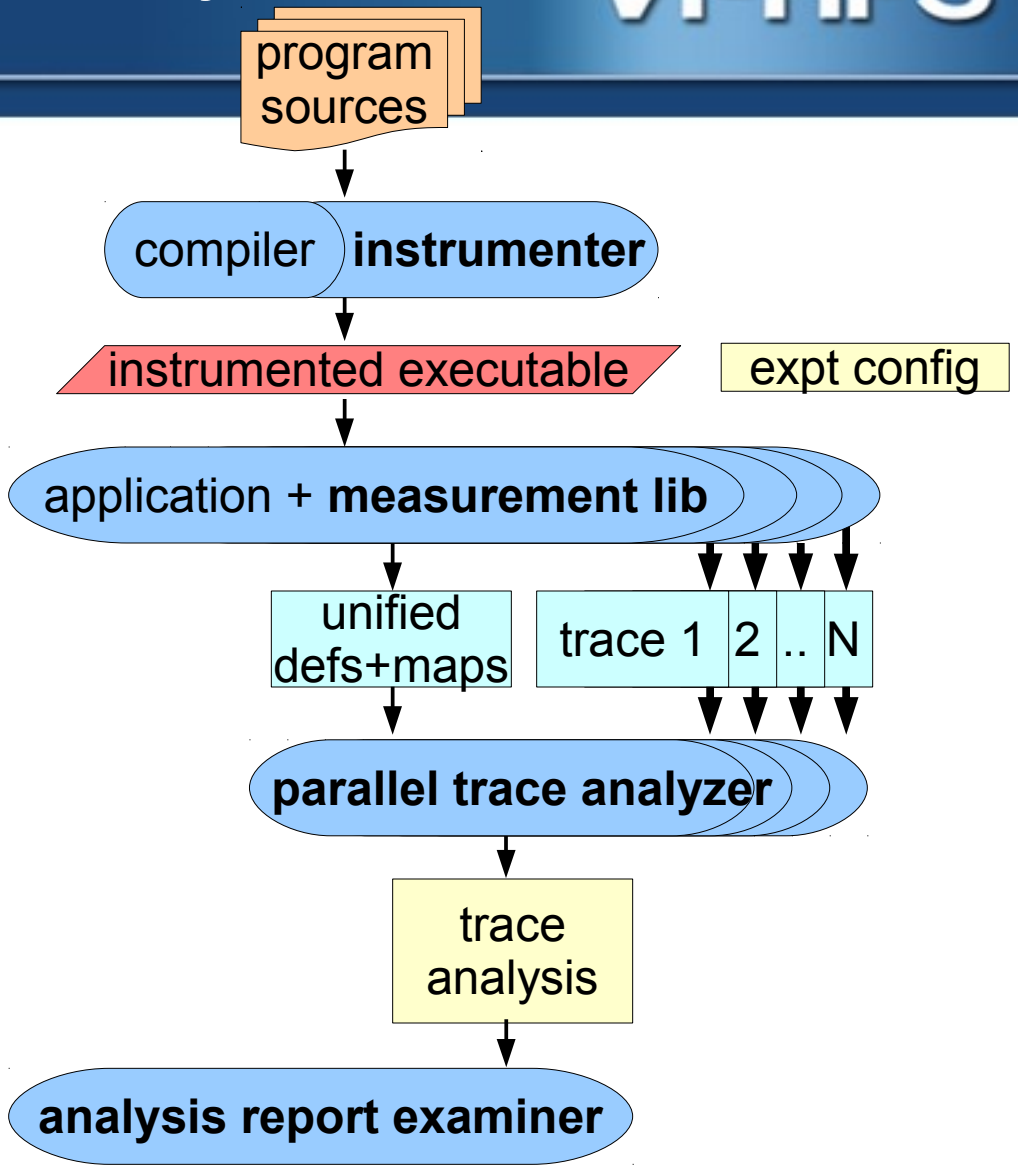
Measurement runtime summarization

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



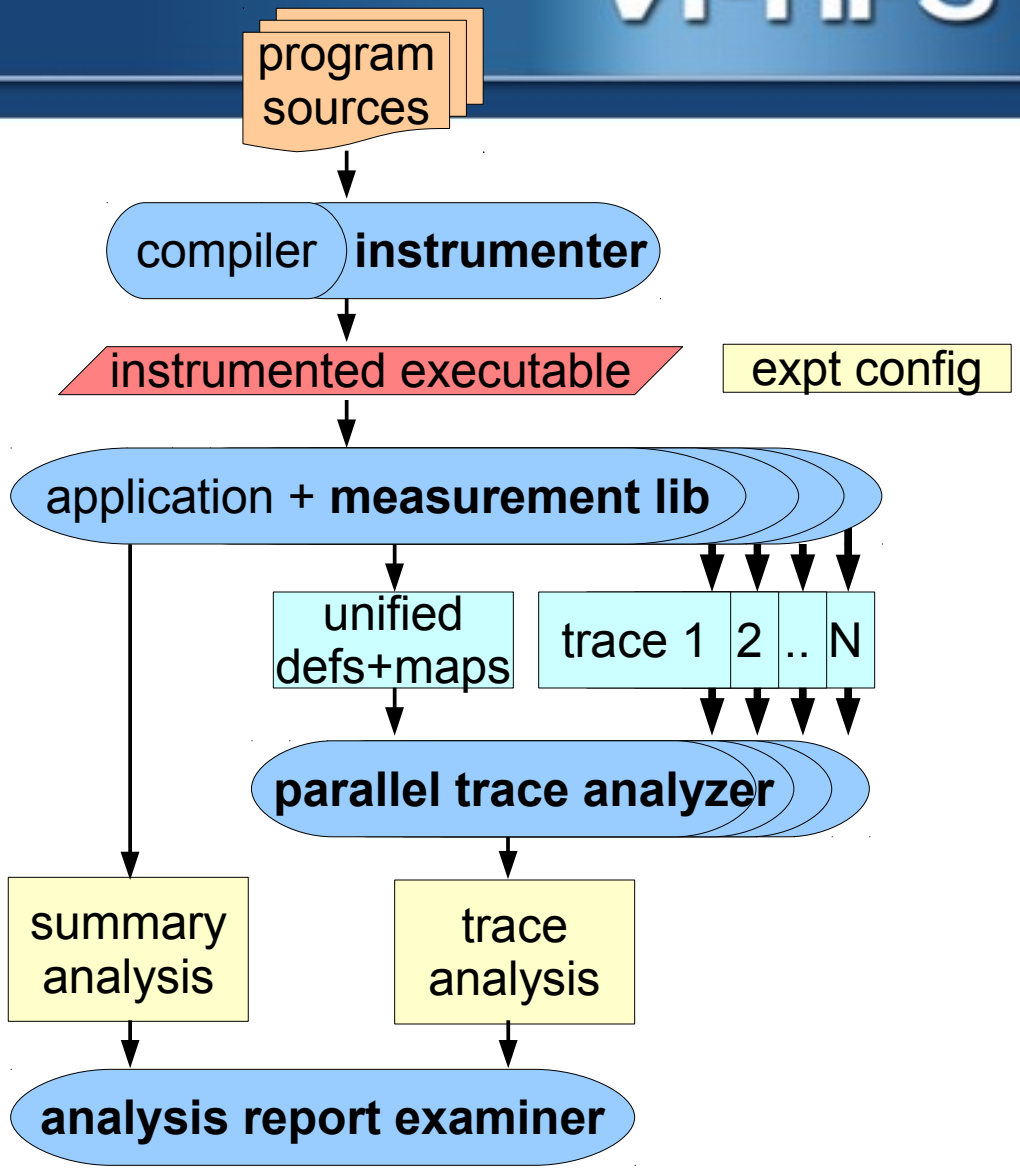
Measurement event tracing & analysis

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



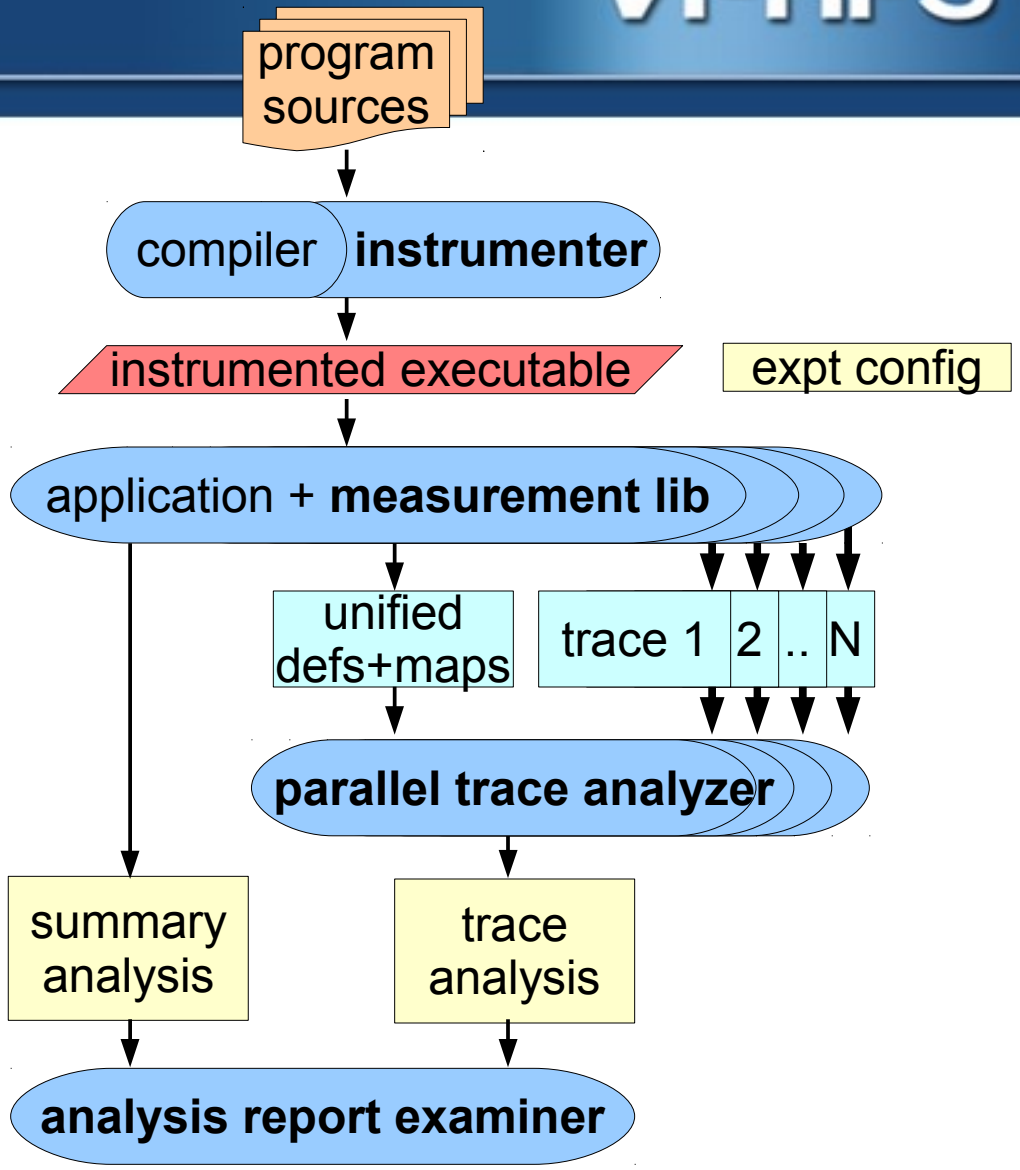
Generic parallel tools architecture

- Automatic/manual code instrumenter
- Measurement library for runtime summary & event tracing
- Parallel (and/or serial) event trace analysis when desired
- Analysis report examiner for interactive exploration of measured execution performance properties



Scalasca toolset components

- Scalasca instrumenter = SKIN
- Scalasca measurement collector & analyzer = SCAN
- Scalasca analysis report examiner = SQUARE



scalasca

- One command for everything

  % **scalasca**

  Scalasca 1.3

  Toolset for scalable performance analysis of large-scale apps

  usage: scalasca [-v][-n] {action}

  1. prepare application objects and executable for measurement:
     scalasca *-instrument* <compile-or-link-command> # ***skin***
  2. run application under control of measurement system:
     scalasca *-analyze* <application-launch-command> # ***scan***
  3. post-process & explore measurement analysis report:
     scalasca *-examine* <experiment-archive|report> # ***square***

  [-h] show quick reference guide (only)

EPIK

- Measurement & analysis runtime system
  - Manages runtime configuration and parallel execution
  - Configuration specified via EPIK.CONF file or environment
    - epik_conf reports current measurement configuration
  - Creates experiment archive (directory): **epik_<*title*>**
  - Optional runtime summarization report
  - Optional event trace generation (for later analysis)
  - Optional filtering of (compiler instrumentation) events
  - Optional incorporation of HWC measurements with events
    - via PAPI library, using PAPI preset or native counter names
- Experiment archive directory
  - Contains (single) measurement & associated files (e.g., logs)
  - Contains (subsequent) analysis reports

OPARI

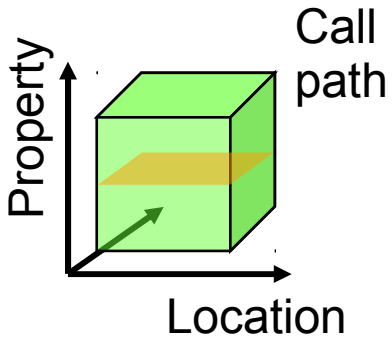
- Automatic instrumentation of OpenMP & POMP directives via source pre-processor
  - Parallel regions, worksharing, synchronization
  - Currently limited to OpenMP 2.5
    - No special handling of guards, dynamic or nested thread teams
  - Configurable to disable instrumentation of locks, etc.
  - Typically invoked internally by instrumentation tools
- Used by Scalasca/Kojak, ompP, TAU, VampirTrace, etc.
  - Provided with Scalasca, but also available separately
    - OPARI 1.1 (October 2001)
    - OPARI 2.0 currently in development

CUBE3

- Parallel program analysis report exploration tools
  - Libraries for XML report reading & writing
  - Algebra utilities for report processing
  - GUI for interactive analysis exploration
    - requires Qt4 or wxGTK widgets library
    - can be installed independently of Scalasca instrumenter and measurement collector/analyzer, e.g., on laptop or desktop
- Used by Scalasca/Kojak, Marmot, ompP, PerfSuite, etc.
  - Analysis reports can also be viewed/stored/analyzed with TAU Paraprof & PerfExplorer
  - Provided with Scalasca, but also available separately
    - CUBE 3.3.1 (November 2010)

Analysis presentation and exploration

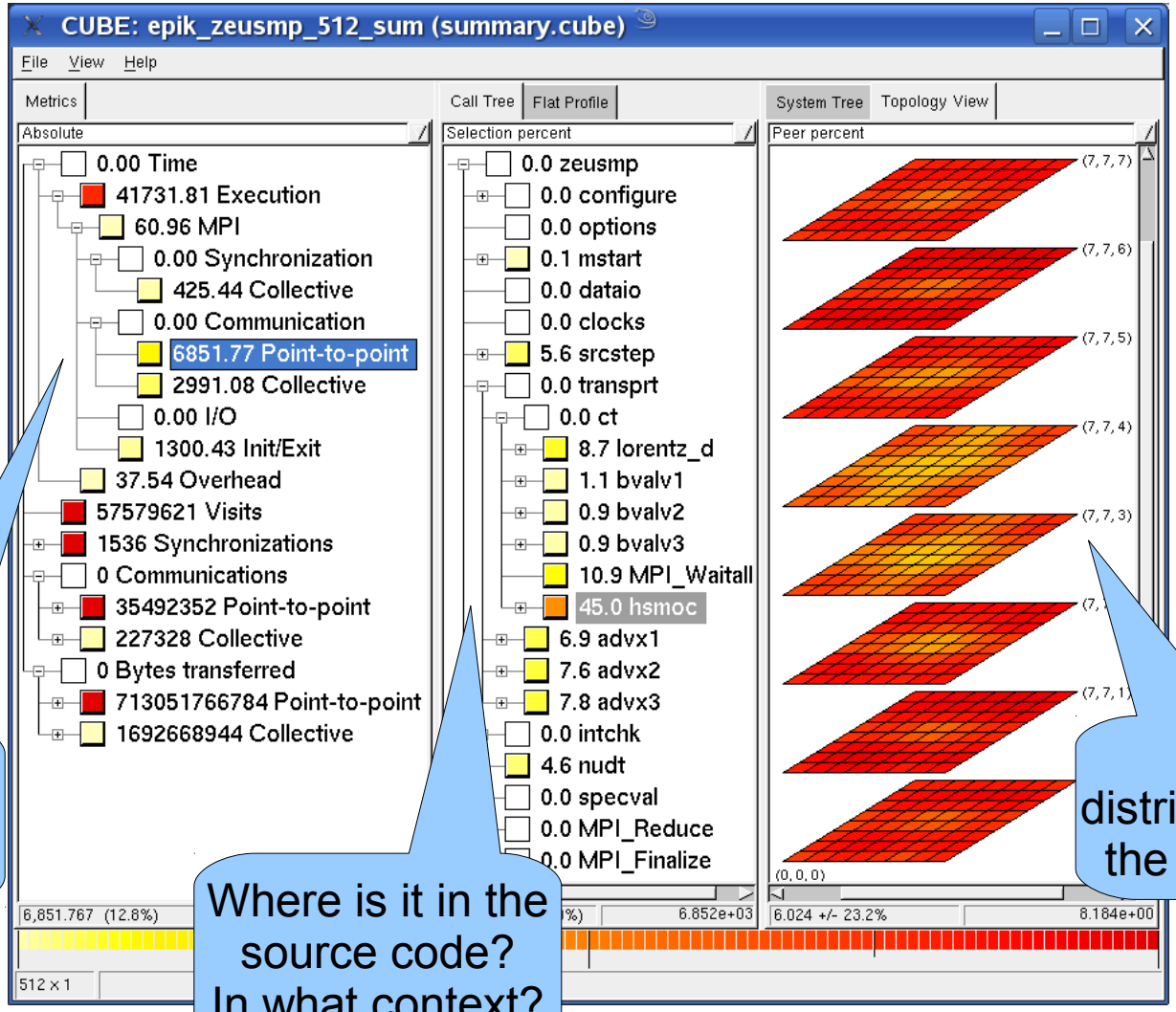
- Representation of values (severity matrix) on three hierarchical axes
  - Performance property (metric)
  - Call-tree path (program location)
  - System location (process/thread)



- Three coupled tree browsers

- CUBE3 displays severities
  - As value: for precise comparison
  - As colour: for easy identification of hotspots
  - Inclusive value when closed & exclusive value when expanded
  - Customizable via display mode

Scalasca analysis report explorer (summary)

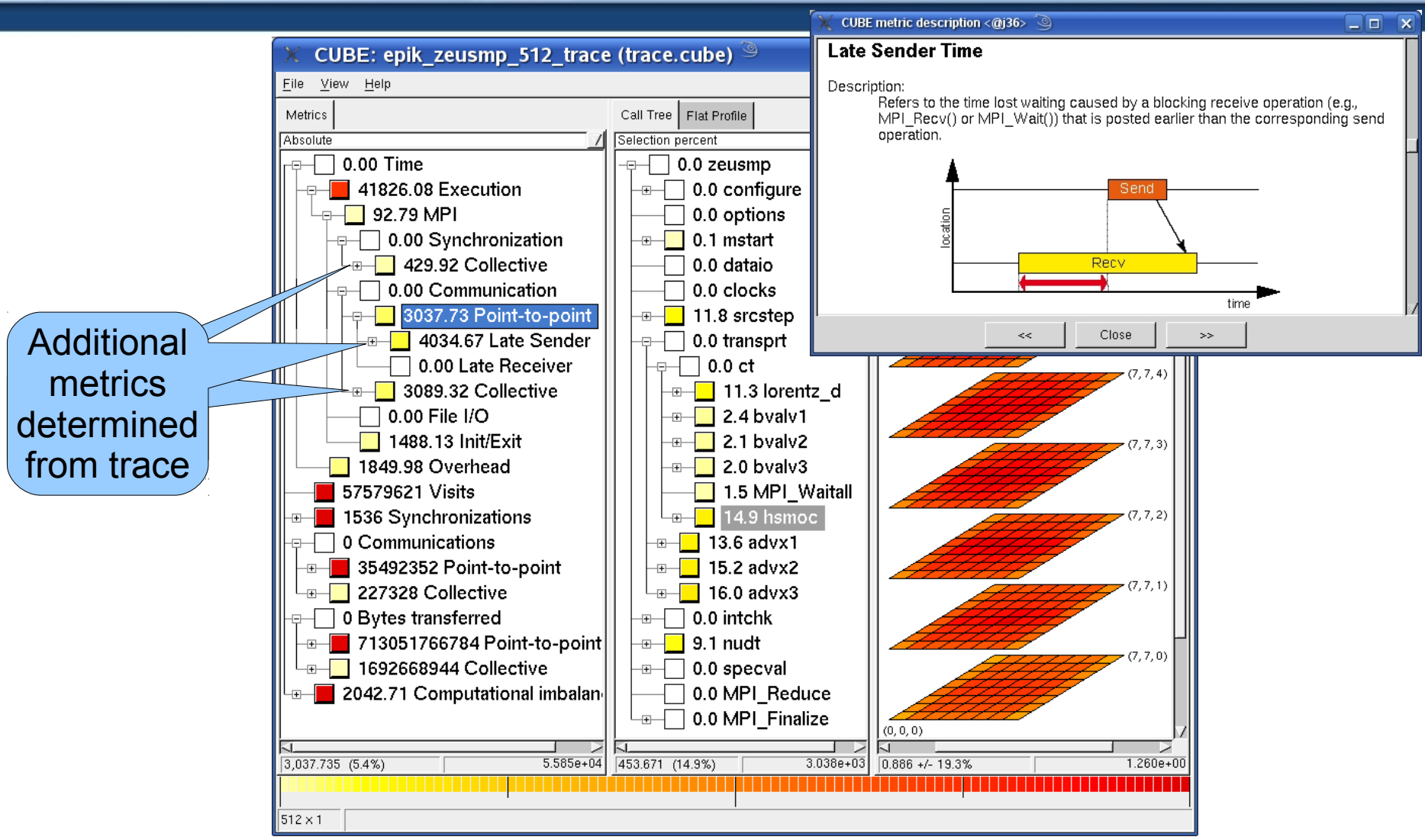


What kind of performance problem?

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

How is it distributed across the processes?

Scalasca analysis report explorer (trace)

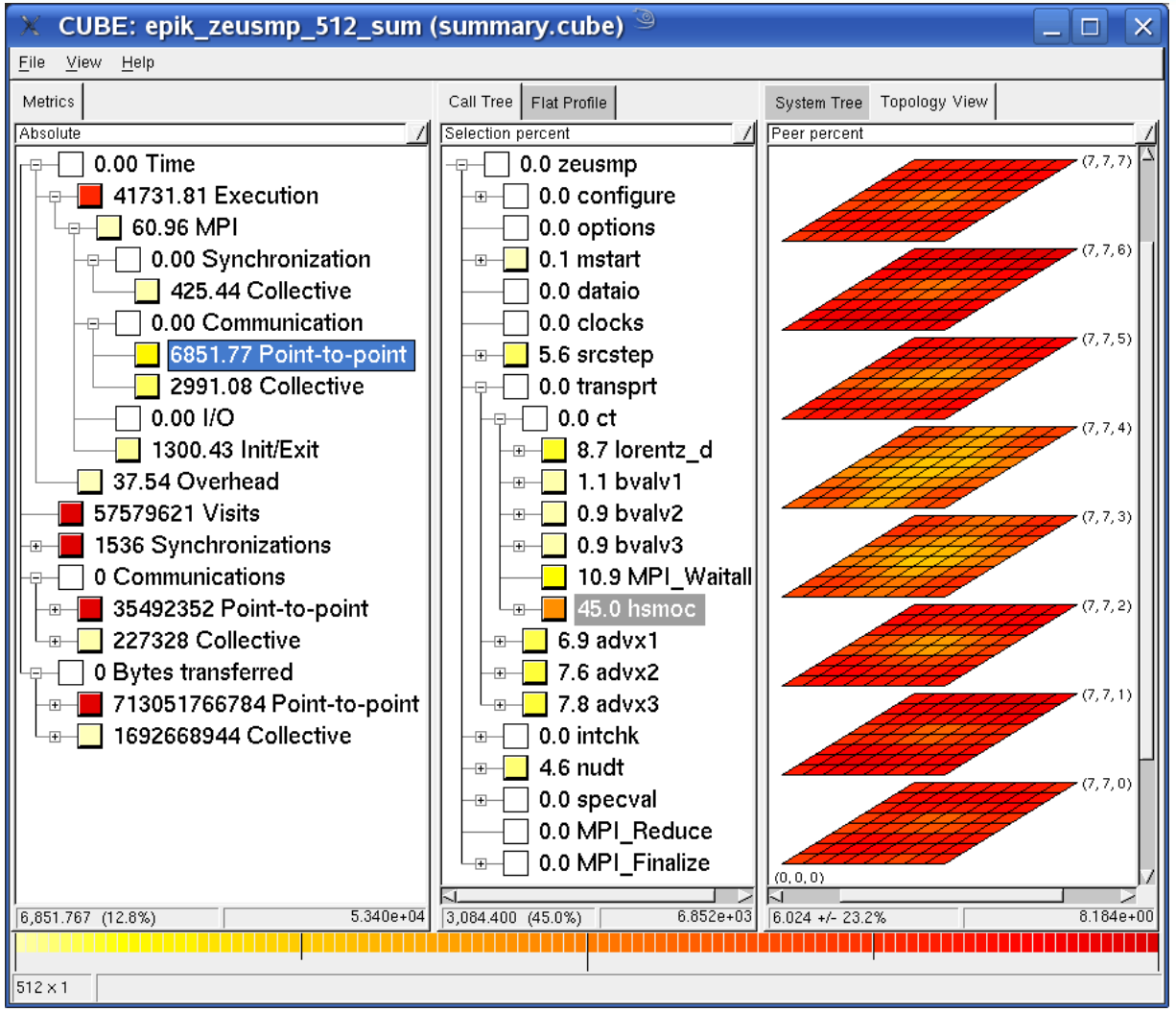


ZeusMP2/JUMP case study

- Computational astrophysics
  - (magneto-)hydrodynamic simulations on 1-, 2- & 3-D grids
  - part of SPEC MPI2007 1.0 benchmark suite (132.zeusmp2)
  - developed by UCSD/LLNL
  - >44,000 lines Fortran90 (in 106 source modules)
  - provided configuration scales to 512 MPI processes
- Run with 512 processes on JUMP
  - IBM p690+ eServer cluster with HPS at JSC
- Scalasca summary and trace measurements
  - ~5% measurement dilation (full instrumentation, no filtering)
  - 2GB trace analysis in 19 seconds
  - application's 8x8x8 grid topology automatically captured from MPI Cartesian

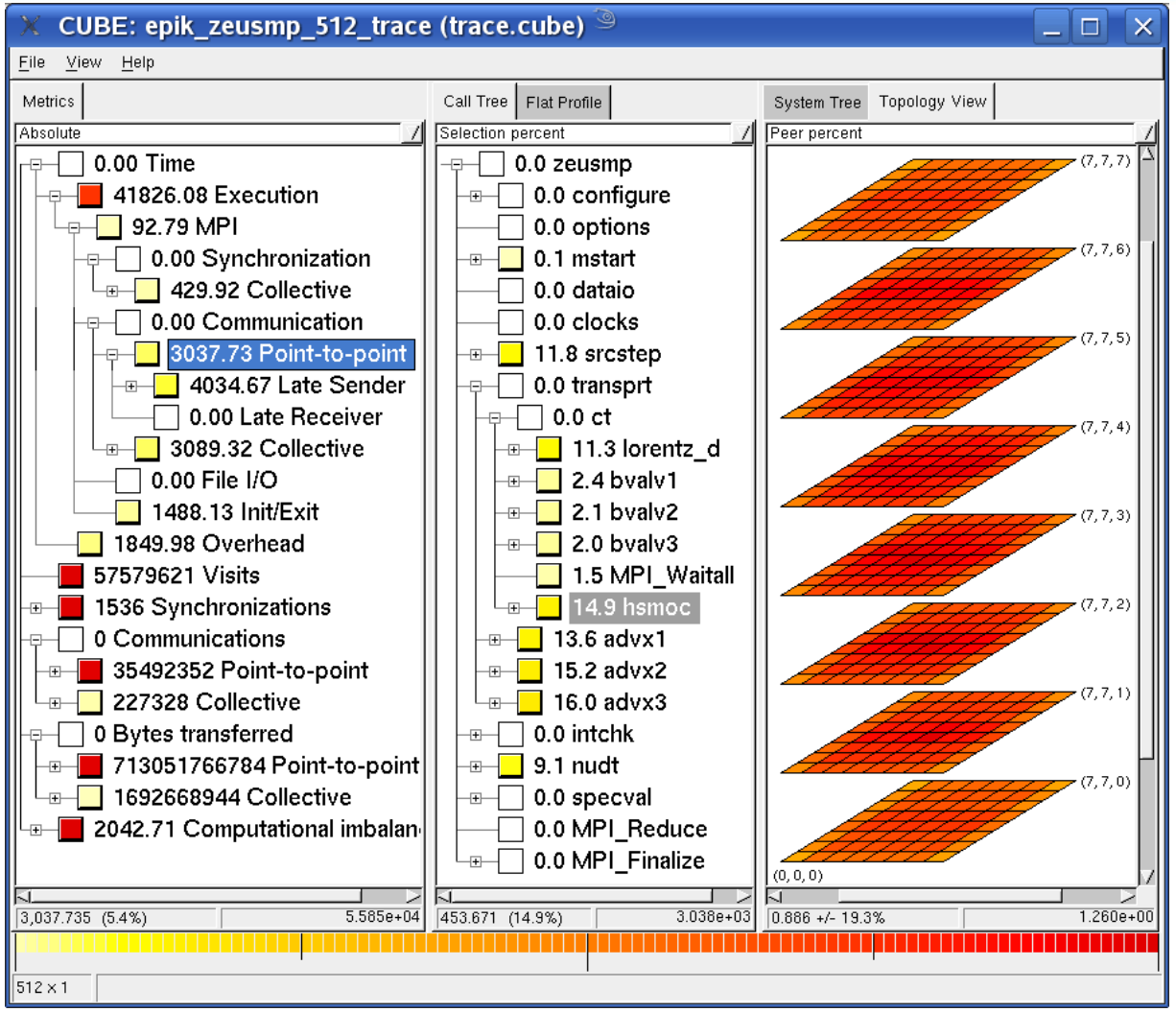
Scalasca summary analysis: zeusmp2 on jump

- 12.8% of time spent in MPI point-to-point communication
- 45.0% of which is on program callpath `transprt/ct/hsmoc`
- With 23.2% std dev over 512 processes
- Lowest values in 3rd and 4th planes of the Cartesian grid



Scalasca trace analysis: zeusmp2 on jump

- MPI point-to-point communication time separated into transport and Late Sender fractions
- Late Sender situations dominate (57%)
- Distribution of transport time (43%) indicates congestion in interior of grid



Scalasca 1.3 functionality

- Automatic function instrumentation (and filtering)
  - CCE, GCC, IBM, Intel, PathScale & PGI compilers
  - optional PDToolkit selective instrumentation (when available) and manual instrumentation macros/pragmas/directives
- MPI measurement & analyses
  - scalable runtime summarization & event tracing
  - only requires application executable re-linking
  - P2P, collective, RMA & File I/O operation analyses
- OpenMP measurement & analysis
  - requires (automatic) application source instrumentation
  - thread management, synchronization & idleness analyses
- Hybrid OpenMP/MPI measurement & analysis
  - combined requirements/capabilities

Scalasca 1.3 added functionality

- Improved configure/installation
- Support for using PDToolkit to instrument sources
  - selective instrumentation of source files and routines
- Consistent instrumentation selection
  - automatic (compiler/pdt) and/or manual (pomp/user)
- Measurement configuration of MPI event wrappers
  - specify desired categories of events, e.g., P2P, COLL, RMA
- MPI RMA (one-sided communication) analysis
- Improved OpenMP (and hybrid) measurement & analysis
  - specify desired number of threads: ESD_MAX_THREADS
  - consistent automatic analyses of traces
- Improved documentation of analysis reports

Scalasca interoperability

- Instrumentation
  - Separate OpenMP instrumenter (OPARI) distribution
  - Scalasca source instrumentation via TAU/PDToolkit
  - Adapter for VT manual instrumentation macros
  - TAU instrumentation with Scalasca measurement libraries
- Trace utilities
  - Trace conversion utilities for VT/OTF, Paraver, JumpShot
  - Vampir visualization of Scalasca traces (without conversion)
- Analysis report utilities
  - Separate report generation/manipulation library and GUI (CUBE3) distribution
  - Alternative presentation with TAU Paraprof/PerfExplorer
- Part of Unified Tool Environment (UNITE) bundle