VI-HPS

scalasca

Scalable performance analysis of large-scale parallel applications

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

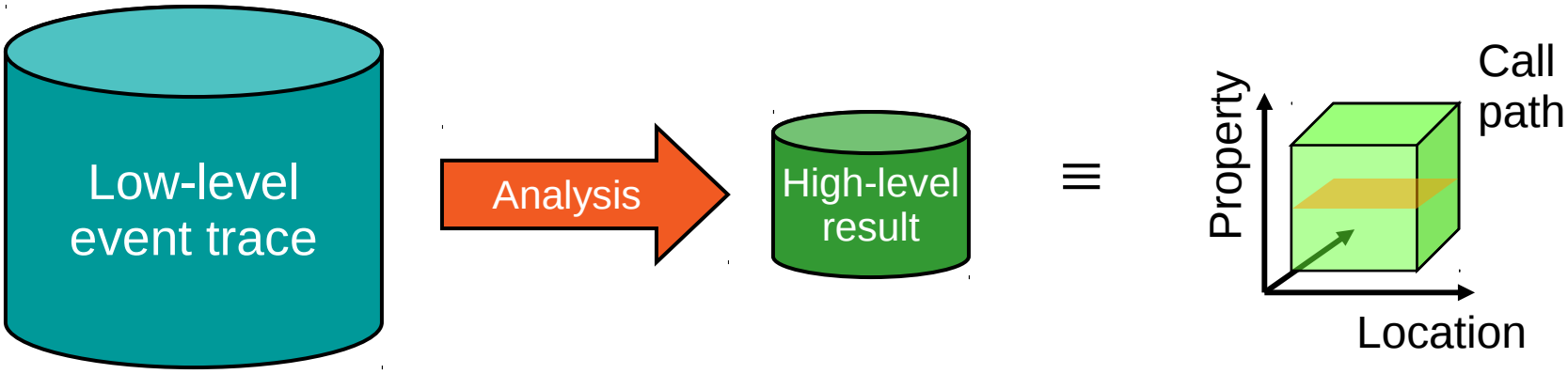
April 2012

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 Bernd Mohr (JSC) & Felix Wolf (GRS)
  - 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/Q or Cray XT/XE/XK with 10,000s to 100,000s of processes
- Latest release February 2012: Scalasca v1.4.1
  - Download from www.scalasca.org
  - Available on POINT/VI-HPS Parallel Productivity Tools DVD

Scalasca features

- Open source, New BSD license
- Portable
  - Cray XT, IBM BlueGene, IBM SP & blade clusters, NEC SX, SGI Altix, 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

Scalasca support & limitations

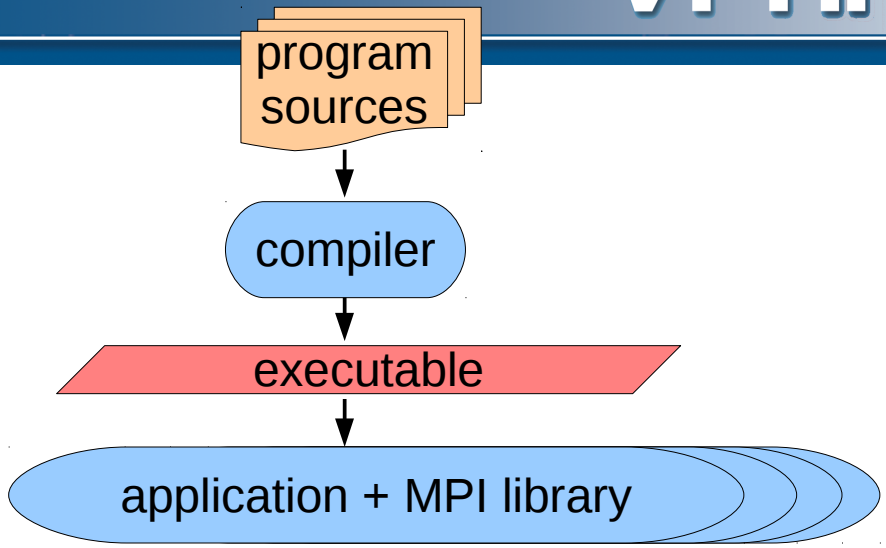
- MPI 2.2 apart from dynamic process creation
  - C++ interface deprecated with MPI 2.2
- OpenMP 2.5 apart from nested thread teams
  - partial support for dynamically-sized/conditional thread teams*
  - no support for OpenMP used in macros or included files
- Hybrid OpenMP+MPI
  - partial support for non-uniform thread teams*
  - no support for MPI_THREAD_MULTIPLE
  - no trace analysis support for MPI_THREAD_SERIALIZED (only MPI_THREAD_FUNNELED)

* Summary & trace measurements are possible, and traces may be analyzed with Vampir or other trace visualizers

  - automatic trace analysis currently not supported

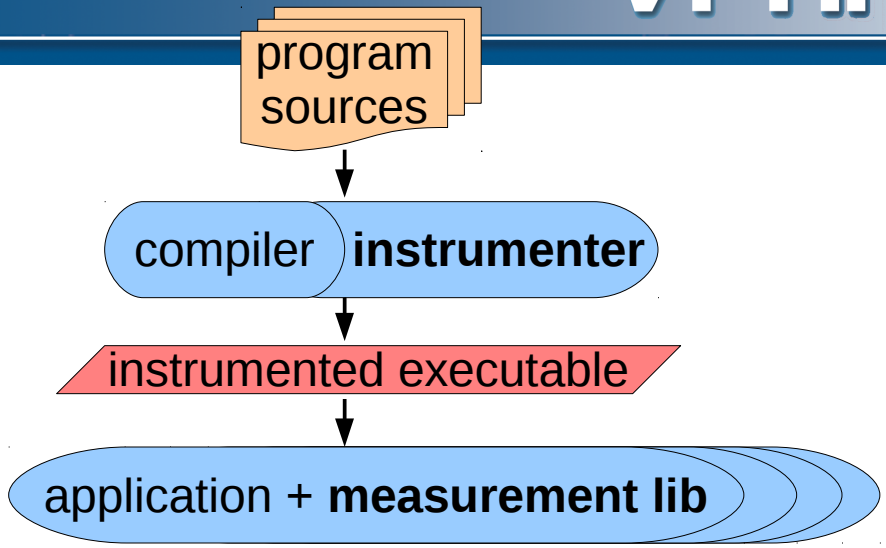
Generic MPI application build & run

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



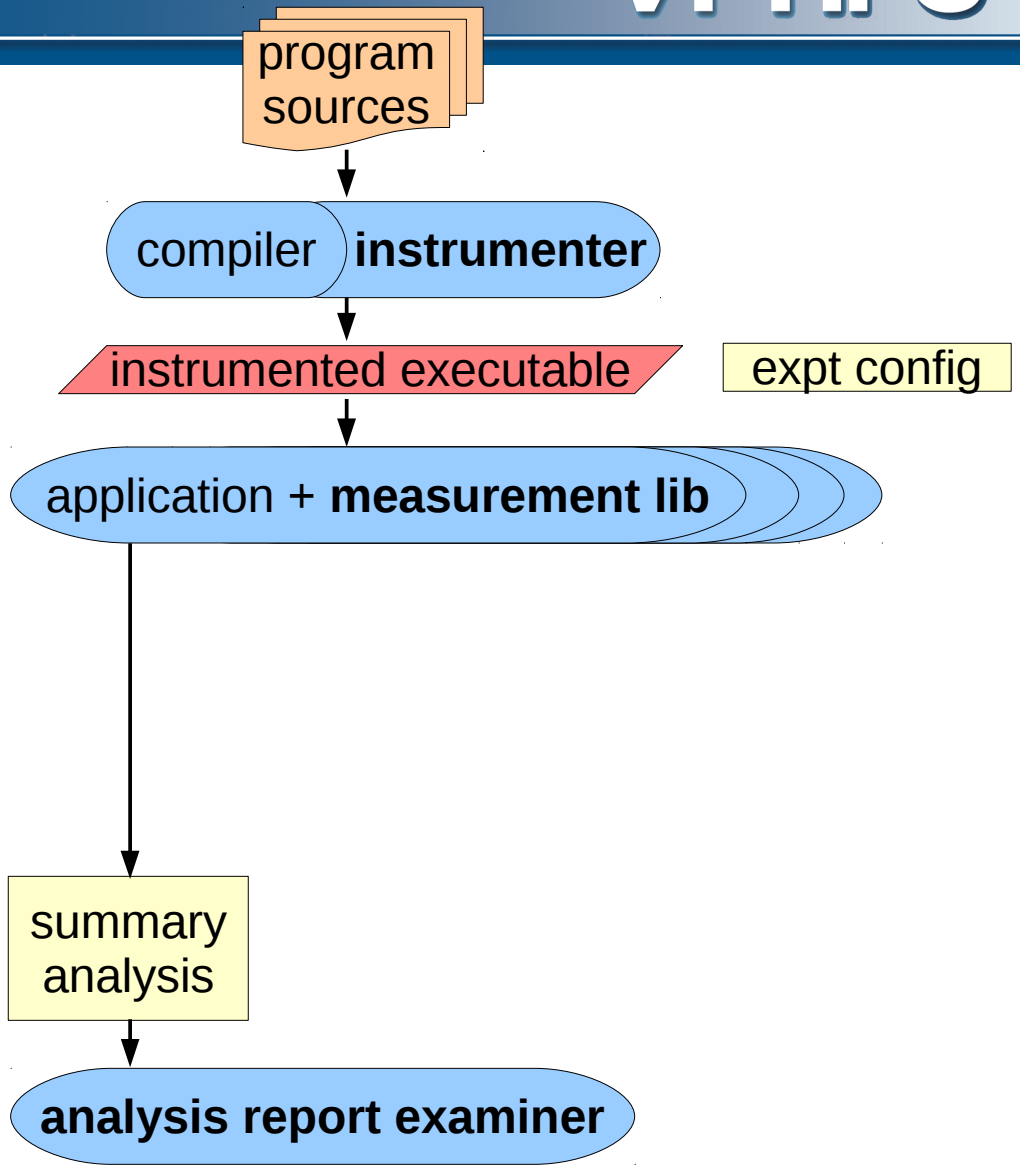
Application instrumentation

- 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



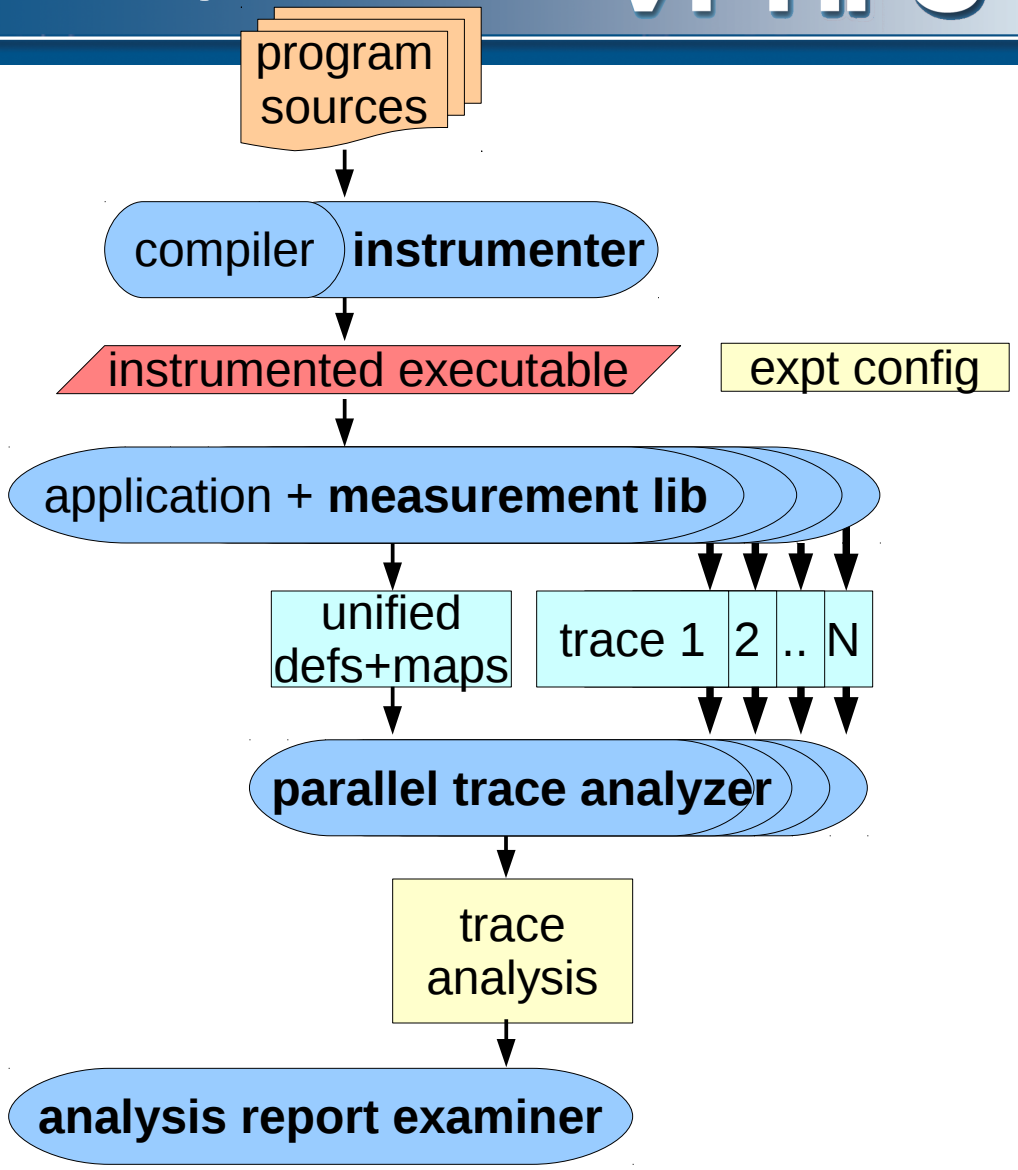
Measurement runtime summarization

- 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



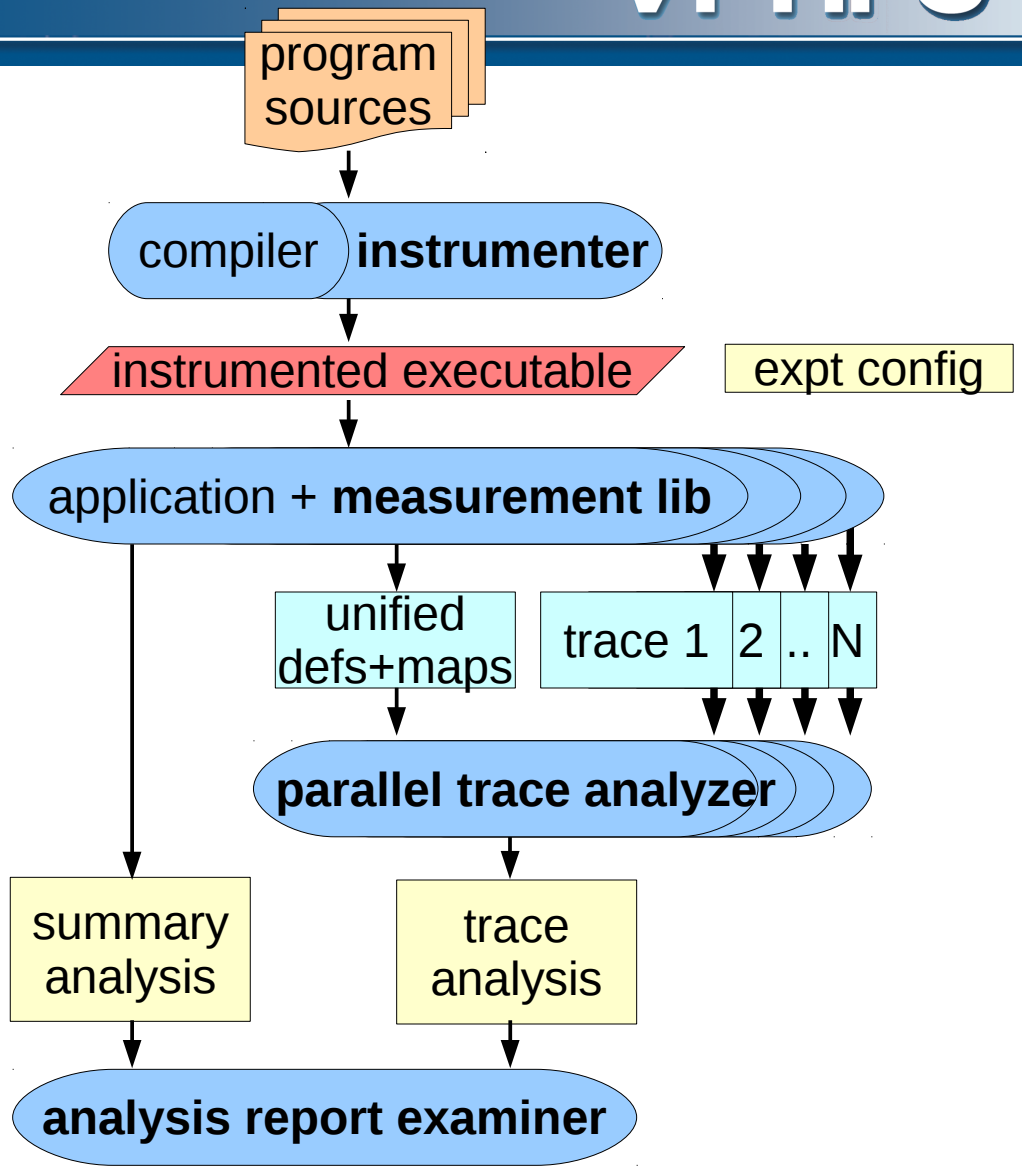
Measurement event tracing & analysis

- 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



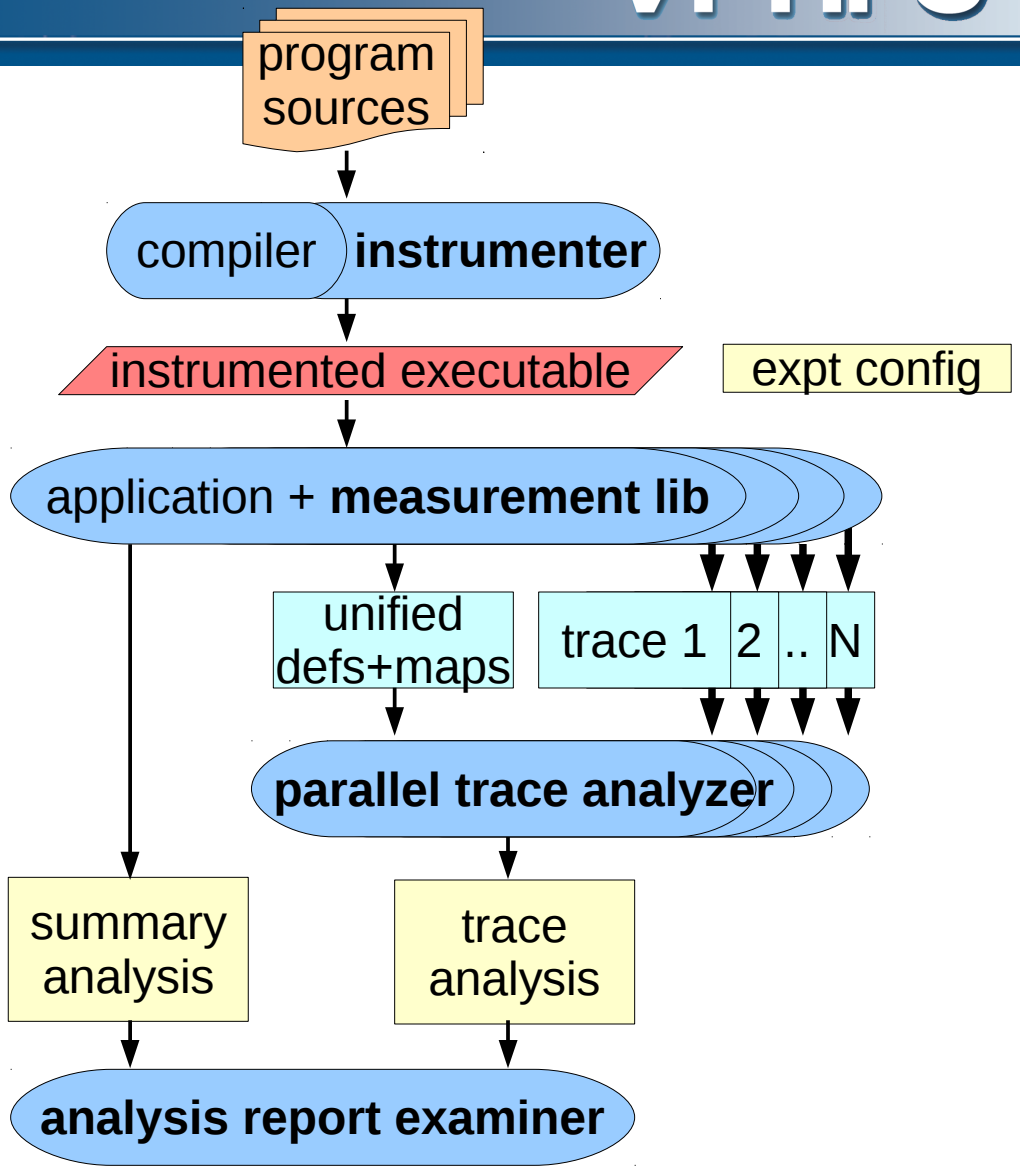
Generic parallel tools architecture

- 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



- One command for everything

  % **scalasca**

  Scalasca 1.4

  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)

- 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

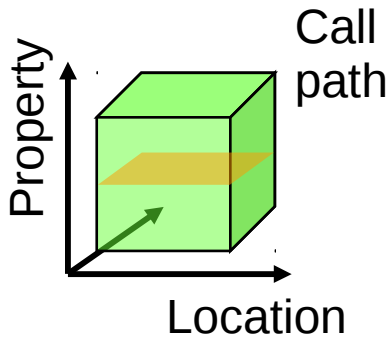
OPARI

- Automatic instrumentation of OpenMP & POMP directives via source pre-processor
  - Parallel regions, worksharing, synchronization
  - OpenMP 2.5 with OpenMP 3.0 coming
    - No special handling of guards, dynamic or nested thread teams
    - OpenMP 3.0 ORDERED sequentialization support
    - Support for OpenMP 3.0 tasks currently in development
  - Configurable to disable instrumentation of locks, etc.
  - Typically invoked internally by instrumentation tools
- Used by Scalasca/Kojak, ompP, Periscope, Score-P, TAU, VampirTrace, etc.
  - Provided with Scalasca, but also available separately
    - OPARI 1.1 (October 2001)
    - OPARI2 1.0 (January 2012)

- Parallel program analysis report exploration tools
  - Libraries for XML report reading & writing
  - Algebra utilities for report processing
  - GUI for interactive analysis exploration
    - requires Qt4 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, Score-P, etc.
  - Analysis reports can also be viewed/stored/analyzed with TAU Paraprof & PerfExplorer
  - Provided with Scalasca, but also available separately
    - CUBE 3.4.1 (January 2012)
    - CUBE 4.0 (December 2011)

Analysis presentation and exploration

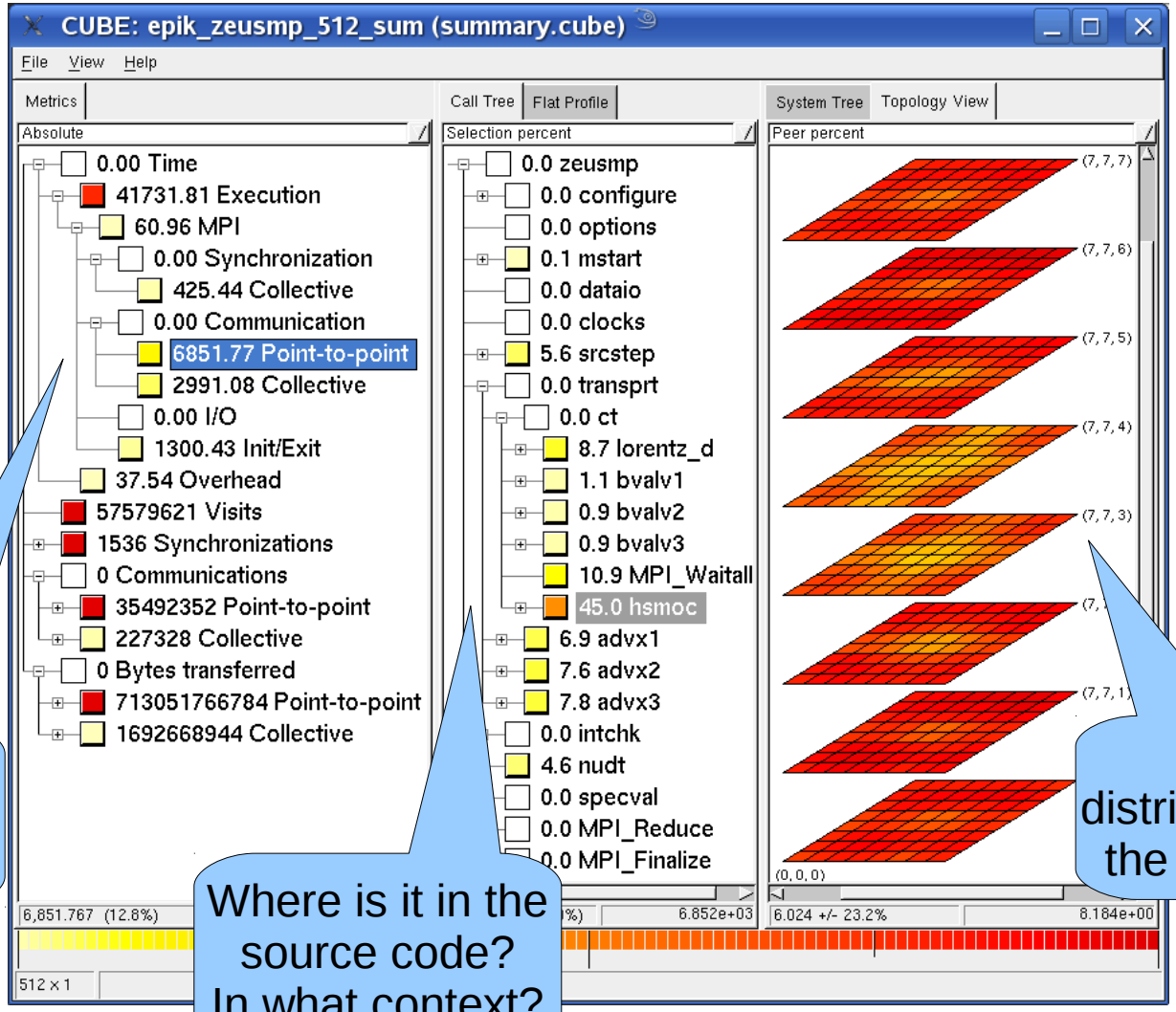
- Representation of values (severity matrix) on three hierarchical axes
  - Performance property (metric)
  - Call-tree path (program location)
  - System location (process/thread)



- Three coupled tree browsers

- CUBE 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)

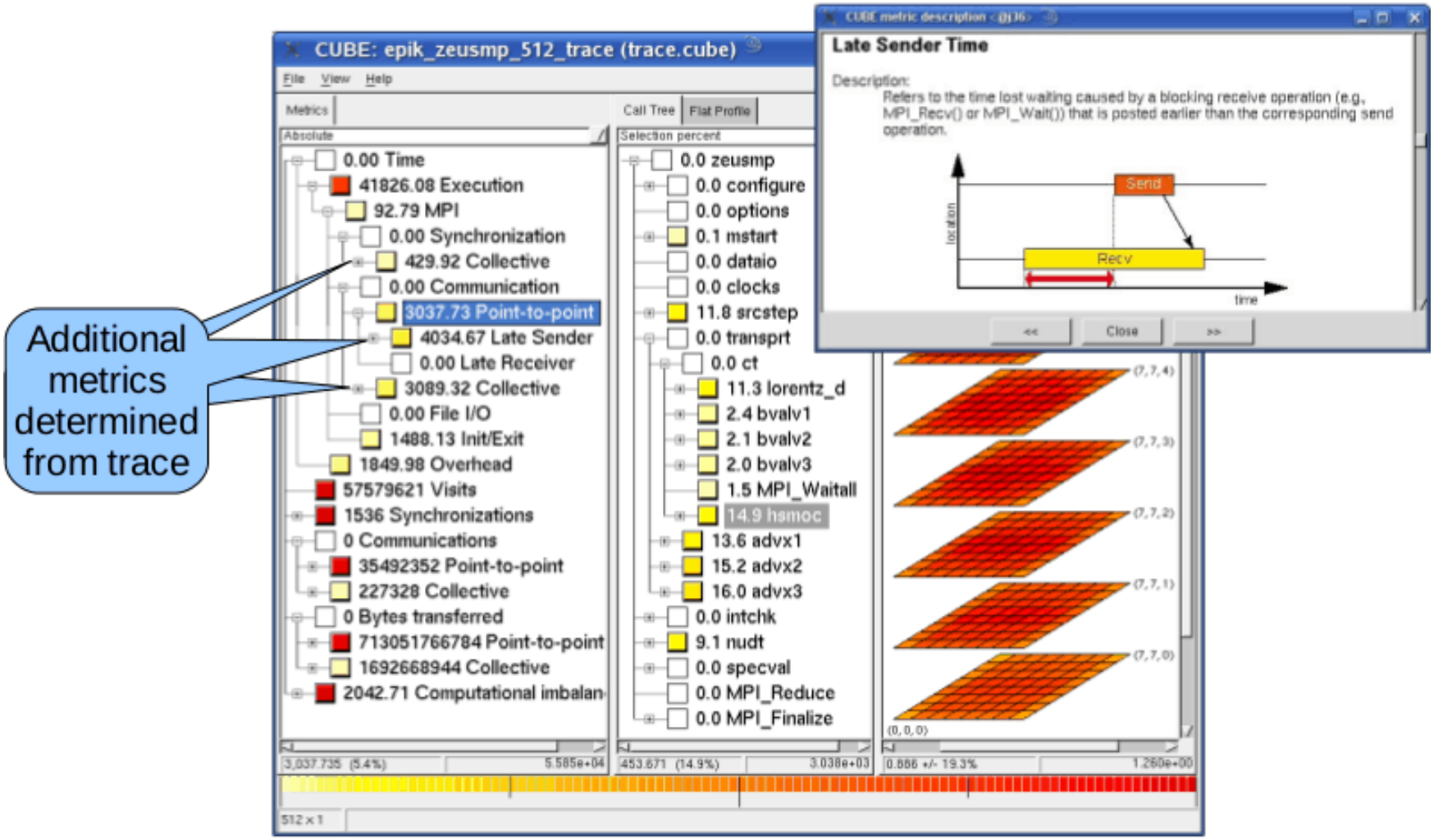


What kind of performance problem?

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

How is it distributed across the processes?

Scalasca analysis report explorer (trace)

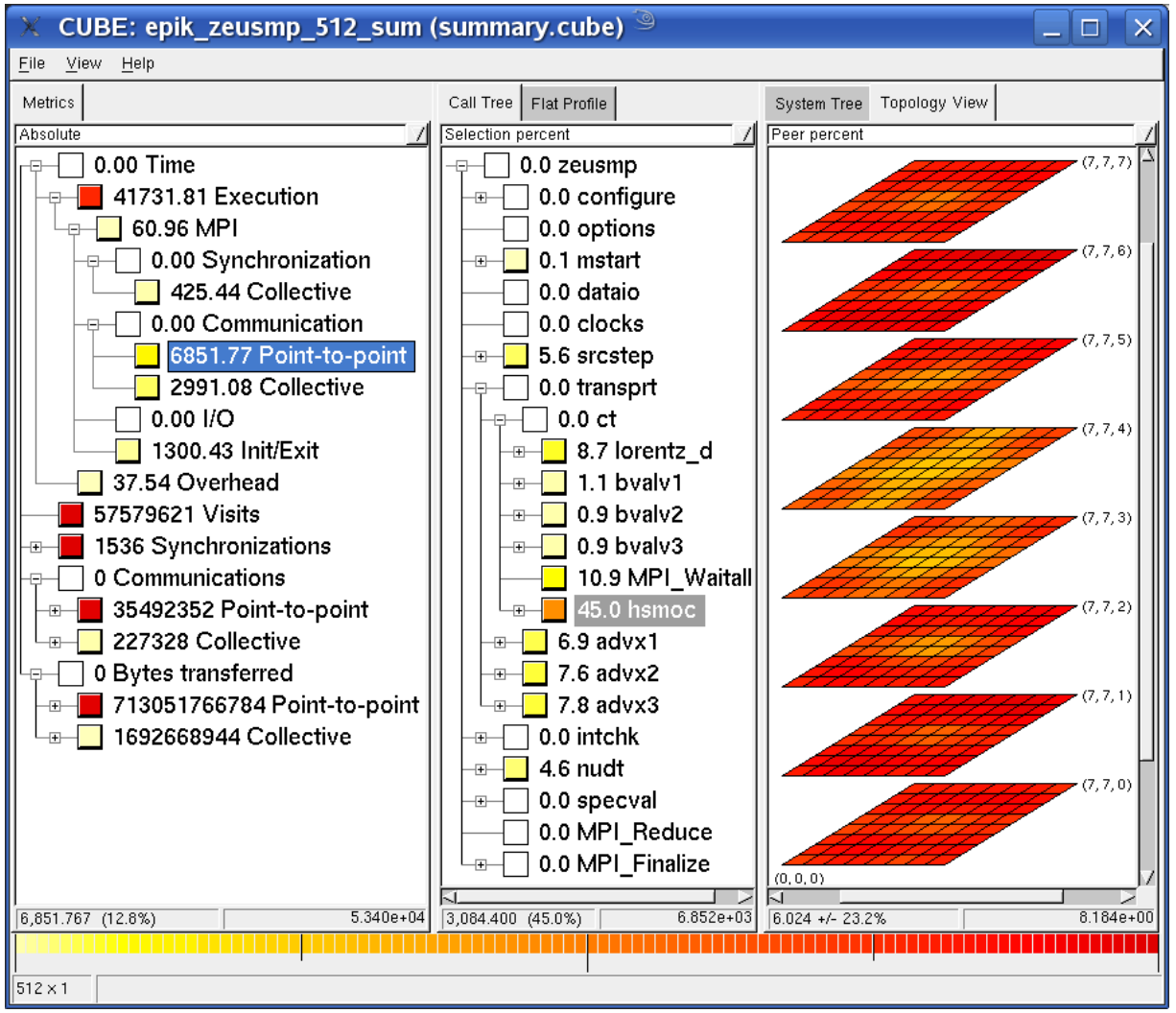


ZeusMP2/JUMP case study

- 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

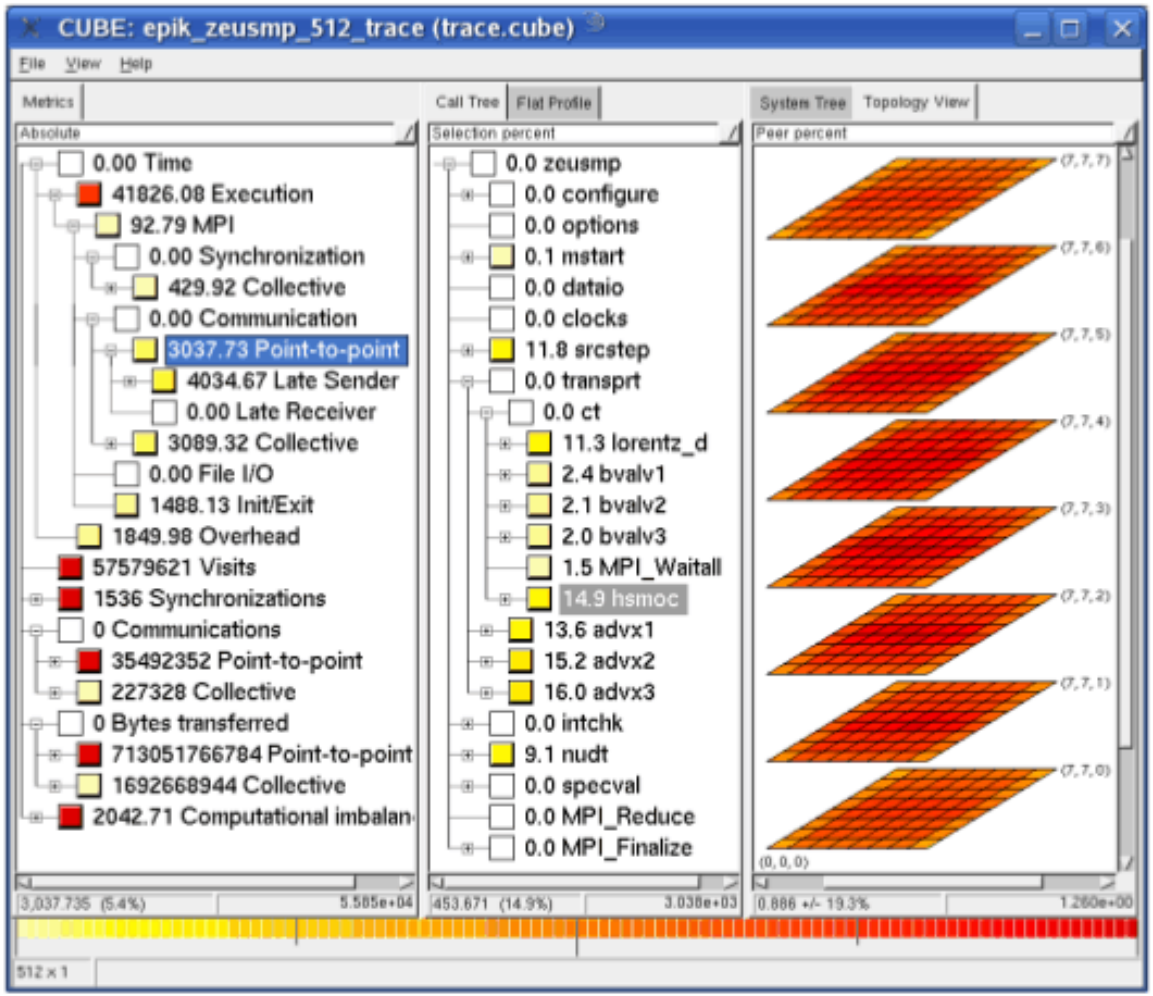
Scalasca summary analysis: zeusmp2 on JUMP

- 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 1.4 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
  - parallel trace analysis requires uniform thread teams

Scalasca 1.4 added functionality

- Improved configure/installation
- Improved parallel & distributed source instrumentation
  - OpenMP/POMP source instrumentation with OPARI2
- Improved MPI communicator management
- Additional summary metrics
  - MPI-2 File bytes transferred (read/written)
  - OpenMP-3 ORDERED sequentialization time
- Improved OpenMP & OpenMP+MPI tracefile management via SIONlib parallel I/O library
- Trace analysis reports of severest pattern instances
  - linkage to external trace visualizers Vampir & Paraver
- New boxplot and topology presentations of distributions
- Improved documentation of analysis reports

Scalasca on VEAS

- /soft/perftools/scalasca
  - link to /home/projects/scalasca subdirectories
- /home/projects/scalasca/cube-3.4.1
  - link from /soft/perftools/cube/latest
  - Qt4-based GUI for Scalasca analysis report exploration
- /home/projects/scalasca/scalasca-1.4.2rc1+sion
  - configured with PDT, PAPI & SIONlib
  - generally recommended
  - link from /soft/perftools/scalasca/latest
- /home/projects/scalasca/scalasca-1.4.2rc1-sion
  - configured with PDT, PAPI & without SIONlib
  - available as a backup in case of problems
  - link from /soft/perftools/scalasca/scalasca-regio

Scalasca issues & limitations (BG/Q): general

- Instrumentation
  - compatibilities of different compilers/libraries unknown
    - if in doubt, rebuild everything
- Measurement collection & analysis
  - runjob & qsub support likely to be incomplete
    - quote ignorable options and try different variations of syntax
    - can't use "**scan** qsub" with qsub script mode
      - use "**scan** runjob" within script instead
    - in worst case, should be able to configure everything manually
  - node-level hardware counters replicated for every thread
  - scout.hyb generally coredumps after completing trace analysis
- Analysis report examination
  - Hardware topology shows only one process per compute node (the one with the largest rank)

Scalasca issues & limitations (BG/Q): tracing

- Tracing experiments collect trace event data in trace files, which are automatically analysed with a parallel analyzer
  - parallel trace analysis requires the same configuration of MPI processes and OpenMP threads as used during collection
  - generally done automatically using the allocated partition
- By default, Scalasca uses separate trace files for each MPI process rank stored in the unique experiment archive
  - for pure MPI, data written directly into archive files
    - the number of separate trace files may become overwhelming
  - for hybrid MPI+OpenMP, data written initially to files for each thread, merged into separate MPI rank files during experiment finalization, and then split again during trace analysis
    - the number of intermediate files may be overwhelming
    - merging and parallel read can be painfully slow

Scalasca issues & limitations (BG/Q): sionlib

- Scalasca can be configured to use the SIONlib I/O library
  - optimizes parallel file reading and writing
    - avoids explicit merging and splitting of trace data files
  - can greatly reduce file creation cost for large numbers of files
  - ELG_SION_FILES specifies the number of files to be created
    - default of 0 reverts to previous behaviour with non-SION files
  - for pure MPI, try one SION file per (I/O) node
  - for hybrid MPI+OpenMP,
    set ELG_SION_FILES equal to number of MPI processes
    - trace data for each OpenMP thread included in single SION file
    - not usable currently with more than 61 threads per SION file due to exhaustion of available file descriptors

Scalasca 1.4.2rc2 on BG/Q (MPI only)

- Everything should generally work as on other platforms (particularly BG/P), but runjob & Cobalt qsub are unusual
- scalasca -instrument
  - **skin** mpixlf77 -O3 -c bt.o
  - **skin** mpixlf77 -O3 -o bt.1024 *.o
- scalasca -analyze
  - **scan -s** *mpirun* -np 1024 -mode SMP -exe ./bt.1024
    - epik_bt_smp1024_sum
  - **scan -s** *runjob* --np 1024 --ranks-per-node 16 **:** ./bt.1024
    - epik_bt_16p1024_sum
  - **scan -s** *qsub* -n 16 --mode c16 ./bt.1024
    - epik_bt_16p1024_sum (after submitted job actually starts)
- scalasca -examine
  - **square** epik_bt_16p1024_sum

Scalasca 1.4.2rc2 on BG/Q (MPI+OpenMP)

- Everything should generally work as on other platforms (particularly BG/P), but runjob & Cobalt qsub are unusual
- scalasca -instrument
  - **skin** mpixlf77_r -qsmp=omp -O3 -c bt.o
  - **skin** mpixlf77_r -qsmp=omp -O3 -o bt-mz.256 *.o
- scalasca -analyze
  - **scan -s** *mpirun* -np 256 -mode SMP -exe ./bt-mz.256 \
    -env OMP_NUM_THREADS=4
    - epik_bt-mz_smp256x4_sum
  - **scan -s** *runjob* --np 256 --ranks-per-node 16 \
    --envs OMP_NUM_THREADS=4 **:** ./bt-mz.256
  - **scan -s** *qsub* -n 16 --mode c16
    -env OMP_NUM_THREADS=4 ./bt-mz.256
    - epik_bt-mz_16p256x4_sum (after submitted job actually starts)

Scalasca 1.4.2rc2 on BG/Q: experiment names

- Scalasca experiment archive directories uniquely store measurement collection and analysis artefacts
  - experiment title prefixed with **epik_**
- Default EPIK experiment title composed from
  - executable basename (without suffix): **bt-mz**
  - ranks-per-node: **16p**
  - number of MPI ranks: **256**
  - number of OMP threads: **x4**
  - type of experiment: **sum** or **trace**
  - (+ HWC metric-list specification)
- Can alternatively be specified with **-e** command-line option or **EPK_TITLE** environment variable

Scalasca 1.4.2rc2 on BG/Q: HWC experiments

- Scalasca experiments can include hardware counters
  - specify lists of PAPI presets or native counters
  - via **-m** option or EPK_METRICS environment variable
    - EPK_METRICS=PAPI_FP_OPS:PEVT_IU_IS1_STALL_CYC
  - alternatively create a file defining groups of counters, specify this file with EPK_METRICS_SPEC and use the group name
- Available hardware counters (and PAPI presets) and supported combinations are platform-specific
- Shared counters are read and stored for each thread
- Although counters are stored in Scalasca traces, they are (currently) ignored by the parallel trace analyzers
  - storage for counters is not included in `max_tbc` estimates
  - summary+trace experiments produce combined analysis reports including measured hardware counter metrics