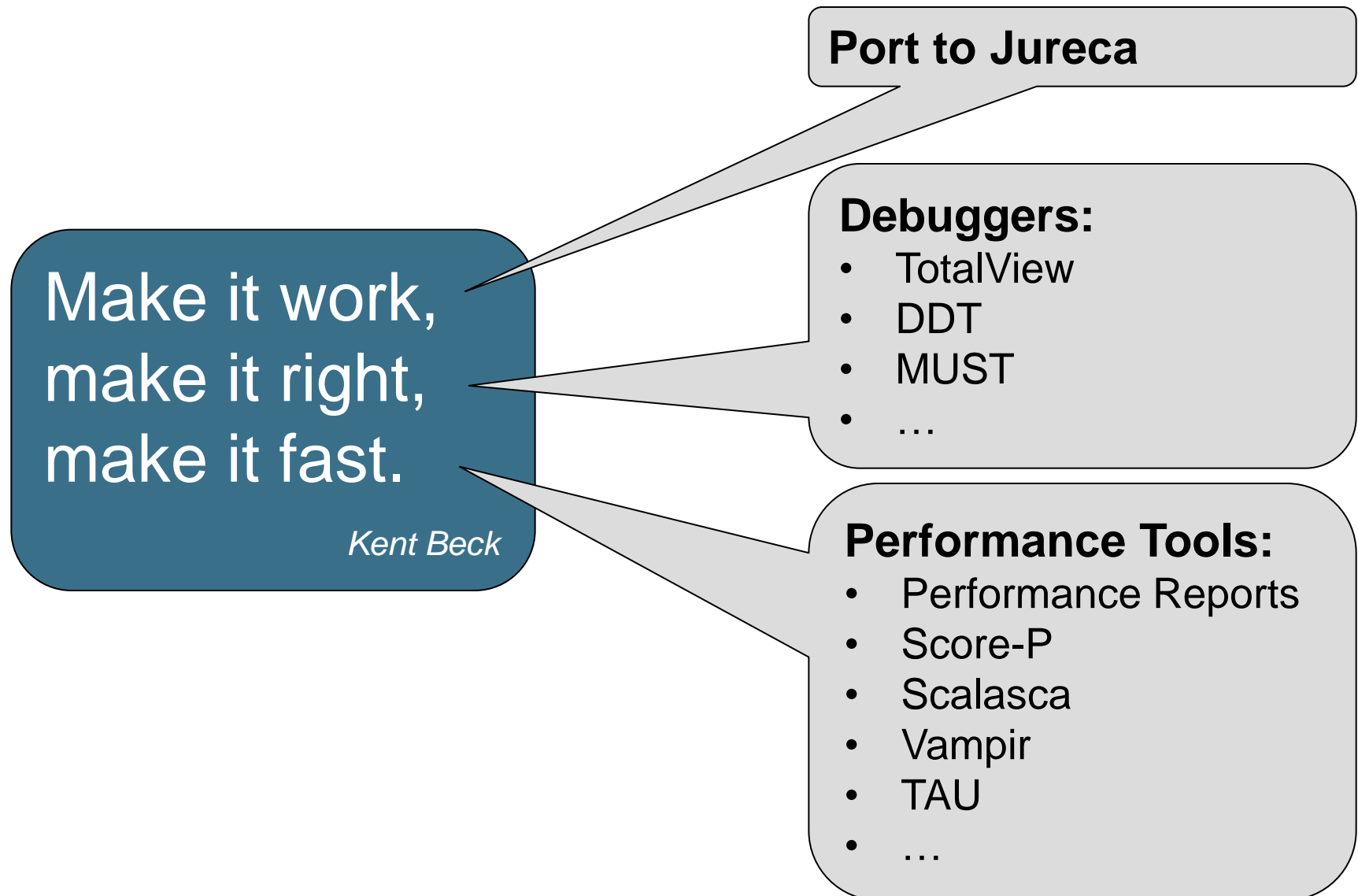


Debuggers and Performance Tools

June 2016 | Markus Geimer

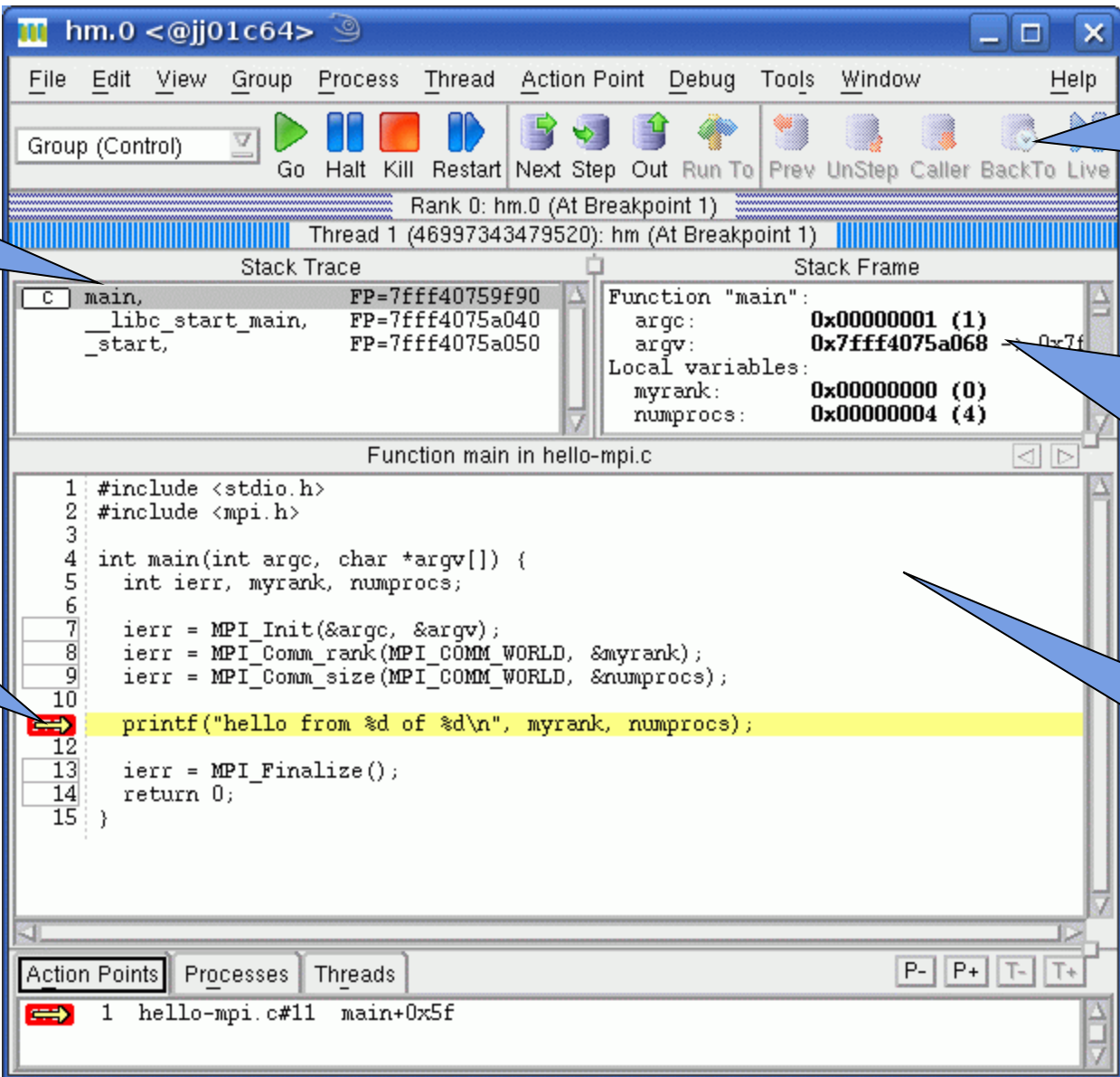


Debugging Tools



- UNIX Symbolic Debugger
for C, C++, F77, F90, PGI HPF, assembler programs
- “Standard” debugger
- Special, non-traditional features
 - Multi-process and multi-threaded
 - C++ support (templates, inheritance, inline functions)
 - F90 support (user types, pointers, modules)
 - 1D + 2D Array Data visualization
 - Support for parallel debugging (MPI: automatic attach, message queues, OpenMP, pthreads)
 - Scripting and batch debugging
 - Memory Debugging
 - CUDA and OpenACC support
- <http://www.roguewave.com>
- **NOTE:** License limited to 2048 processes (shared between all users)

TotalView: Main Window



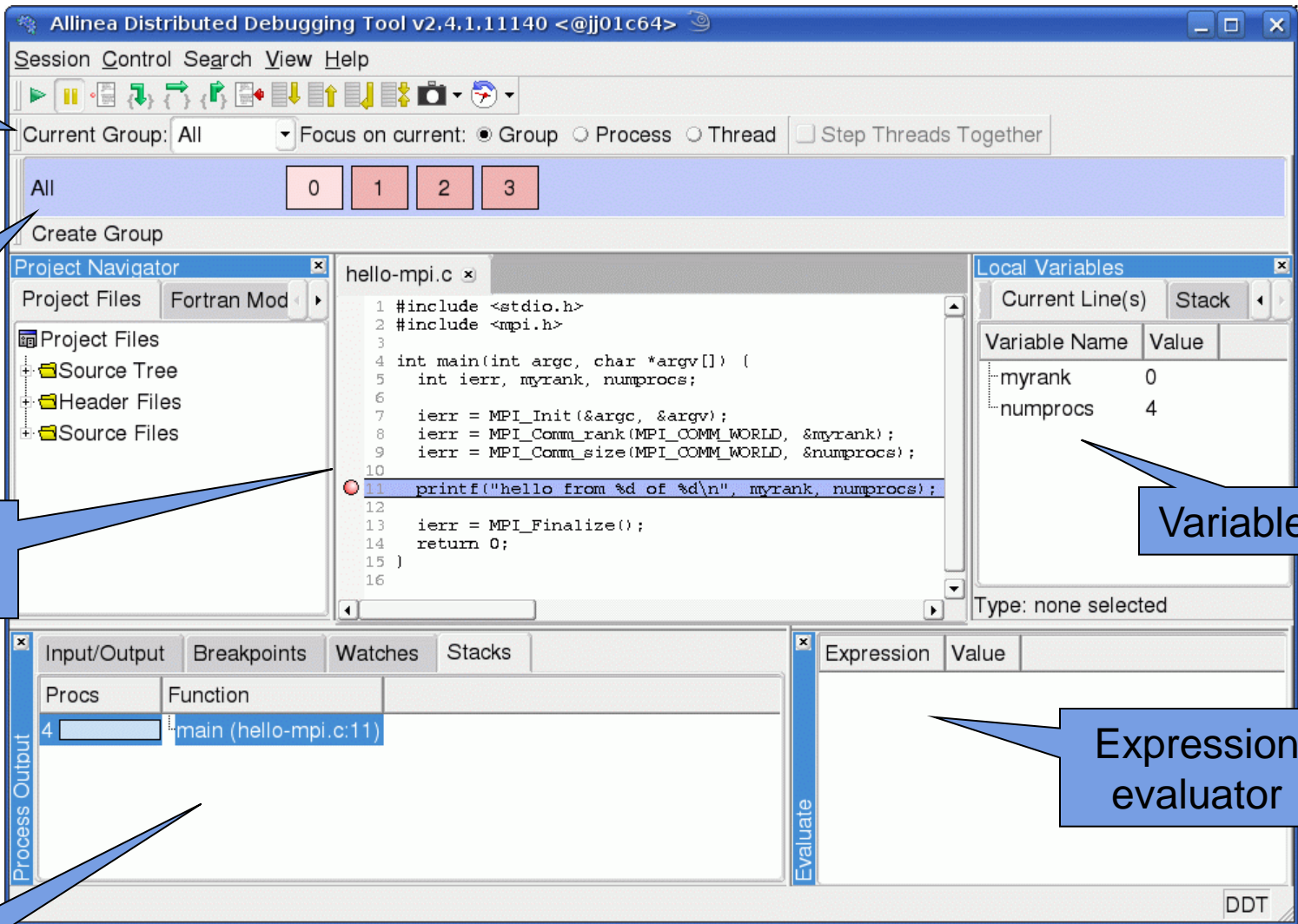
The screenshot shows the TotalView debugger interface for a process named 'hm.0'. The window title is 'hm.0 <@jj01c64>'. The menu bar includes File, Edit, View, Group, Process, Thread, Action Point, Debug, Tools, Window, and Help. The toolbar contains icons for Go, Halt, Kill, Restart, Next Step, Out, Run To, Prev, UnStep, Caller, BackTo, and Live. The status bar shows 'Rank 0: hm.0 (At Breakpoint 1)' and 'Thread 1 (46997343479520): hm (At Breakpoint 1)'. The main area is divided into several panes:

- Stack Trace:** Shows the call stack with frames for 'main', '_libc_start_main', and '_start'. Callouts point to this area with the text 'Stack trace'.
- Stack Frame:** Shows details for the selected 'main' frame, including local variables: 'argc: 0x00000001 (1)', 'argv: 0x7fff4075a068', 'myrank: 0x00000000 (0)', and 'numprocs: 0x00000004 (4)'. A callout points to this area with the text 'Local variables for selected stack frame'.
- Source Code Window:** Displays the source code for 'Function main in hello-mpi.c'. A callout points to this area with the text 'Source code window'. A red arrow icon on the left margin indicates a breakpoint at line 11, which is highlighted in yellow. A callout points to this area with the text 'Break points'.
- Toolbar for common options:** A callout points to the top toolbar with the text 'Toolbar for common options'.

At the bottom, the 'Action Points' pane shows a list of breakpoints: '1 hello-mpi.c#11 main+0x5f'. Navigation buttons 'P-', 'P+', 'T-', and 'T+' are also visible.

- UNIX Graphical Debugger for C, C++, F77, F90 programs
- Modern, easy-to-use debugger
- Special, non-traditional features
 - Multi-process and multi-threaded
 - 1D + 2D array data visualization
 - Support for **MPI parallel debugging** (automatic attach, message queues)
 - Support for **OpenMP** (Version 2.x and later)
 - Support for **CUDA** and **OpenACC**
 - Job submission from within debugger
- <http://www.allinea.com>
- **NOTE:** License limited to 64 processes (shared between all users)

DDT: Main Window



The screenshot shows the Allinea Distributed Debugging Tool (DDT) v2.4.1.11140 interface. The window title is "Allinea Distributed Debugging Tool v2.4.1.11140 <@jj01c64>". The menu bar includes "Session", "Control", "Search", "View", and "Help". The toolbar contains various icons for running, pausing, and stepping through code. Below the toolbar, there are controls for "Current Group: All", "Focus on current: Group", "Process", "Thread", and "Step Threads Together". A row of buttons labeled "All", "0", "1", "2", and "3" represents process groups. A "Create Group" button is also present. The "Project Navigator" on the left shows a tree view with "Project Files", "Source Tree", "Header Files", and "Source Files". The main editor displays the source code for "hello-mpi.c", with line 11 selected: `printf("hello from %d of %d\n", myrank, numprocs);`. The "Local Variables" panel on the right shows a table with "myrank" (0) and "numprocs" (4). The "Process Output" panel at the bottom left shows a stack trace with "main (hello-mpi.c:11)". The "Expression evaluator" panel at the bottom right has an "Evaluate" button. Callouts point to "Process controls", "Process groups", "Source code", "Variables", "Expression evaluator", and "Stack trace".

Process controls

Process groups

Source code

Variables

Expression evaluator

Stack trace

- Next generation MPI correctness and portability checker
- <http://doc.itc.rwth-aachen.de/display/CCP/Project+MUST>



- MUST reports
 - Errors: violations of the MPI-standard
 - Warnings: unusual behavior or possible problems
 - Notes: harmless but remarkable behavior
 - Further: potential deadlock detection

- Can detect errors such as
 - Memory leaks
 - Memory corruption
 - Allocation/deallocation API mismatches
 - Illegal memory accesses
 - Data races
 - Deadlocks
- Available on Jureca:
 - Valgrind
 - Intel Inspector

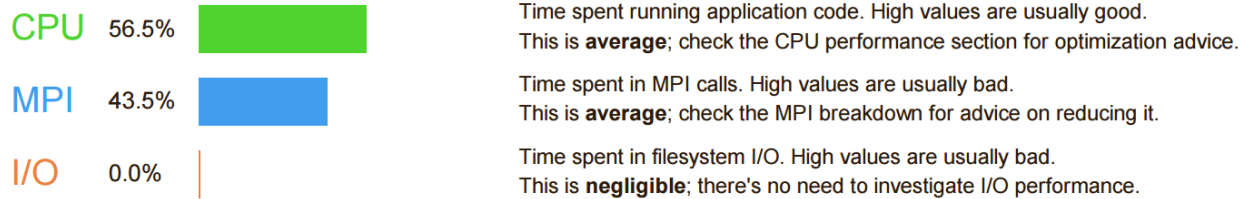
Performance Analysis Tools

- **Single page** report provides quick overview of performance issues
- Works on unmodified, optimized executables
- Shows CPU, memory, network, and I/O utilization
- Supports MPI, multi-threading, and accelerators
- Saves data in HTML, CVS, or text form
- <http://www.allinea.com/products/allinea-performance-reports>
- **Note:** License limited to 512 processes (with unlimited number of threads)

Example Performance Reports

Summary: cp2k.popt is CPU-bound in this configuration

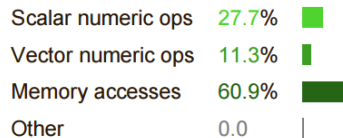
The total wallclock time was spent as follows:



This application run was CPU-bound. A breakdown of this time and advice for investigating further is in the CPU section below.

CPU

A breakdown of how the 56.5% total CPU time was spent:

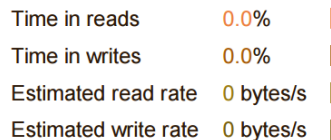


The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance.

Little time is spent in vectorized instructions. Check the compiler's vectorization advice to see why key loops could not be vectorized.

I/O

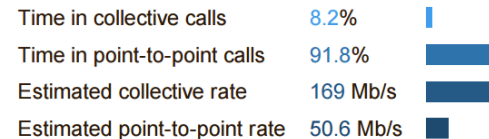
A breakdown of how the 0.0% total I/O time was spent:



No time is spent in I/O operations. There's nothing to optimize here!

MPI

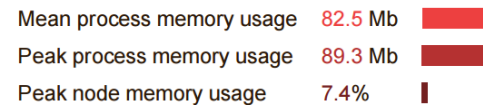
Of the 43.5% total time spent in MPI calls:



The point-to-point transfer rate is low. This can be caused by inefficient message sizes, such as many small messages, or by imbalanced workloads causing processes to wait. Use an MPI profiler to identify the problematic calls and ranks.

Memory

Per-process memory usage may also affect scaling:



The peak node memory usage is low. You may be able to reduce the total number of CPU hours used by running with fewer MPI processes and more data on each process.



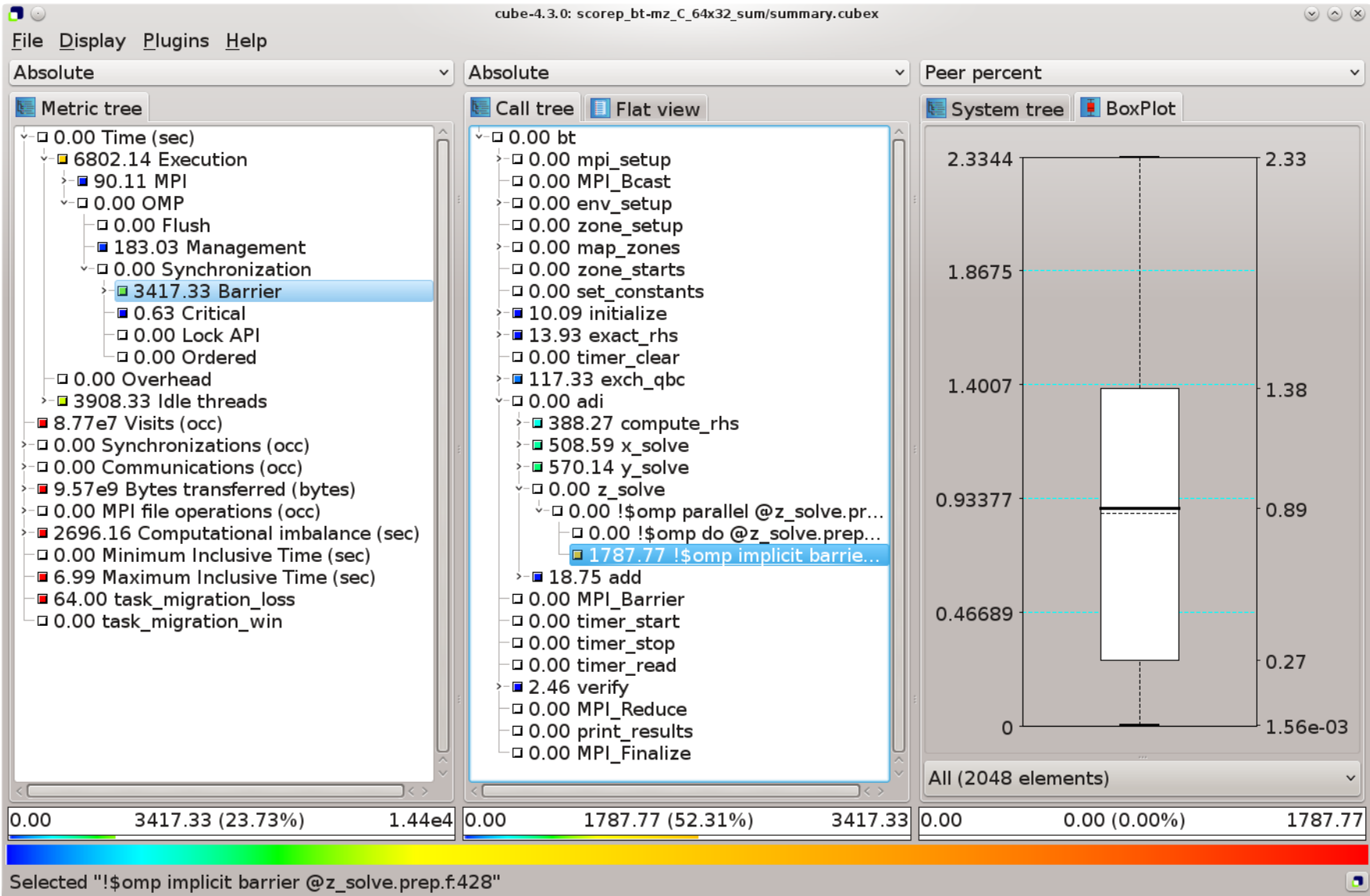
- Community instrumentation and measurement infrastructure
 - Developed by a consortium of performance tool groups



UNIVERSITY OF OREGON

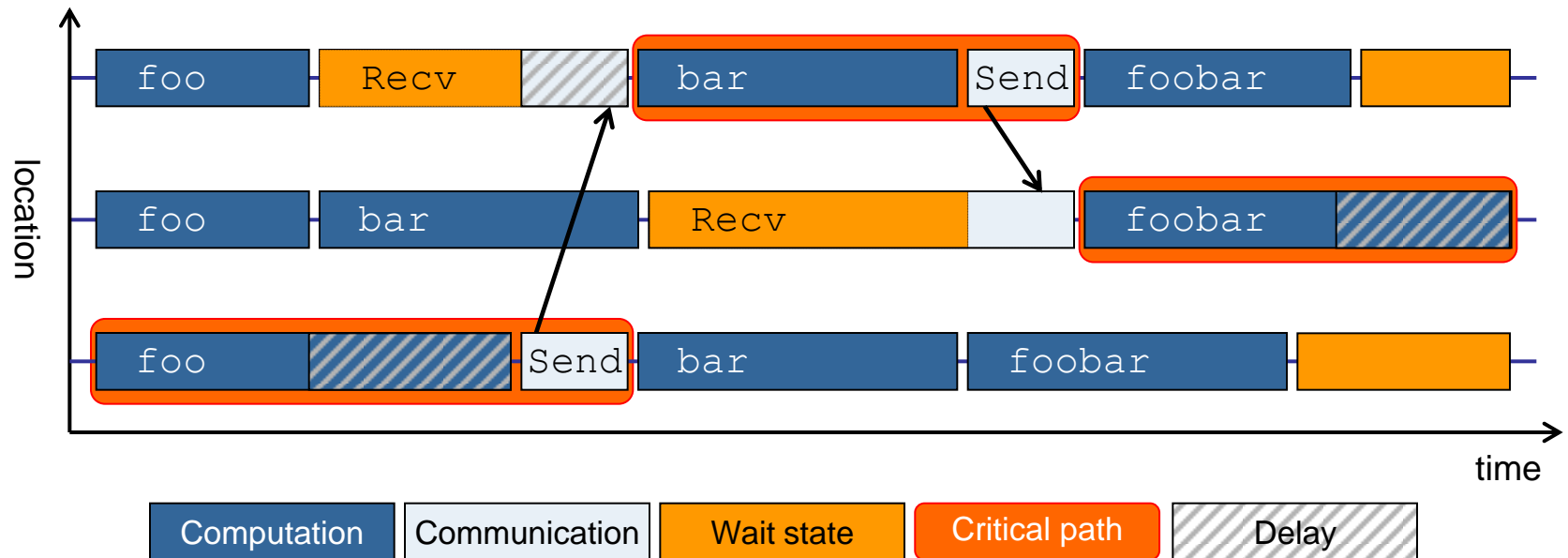
- Common data formats improve tool interoperability
- Supports C, C++, and Fortran using MPI, SHMEM, OpenMP, POSIX threads, CUDA, OpenCL and combinations
- Highly configurable:
 - Basic and advanced profile generation
 - Event trace recording
 - Using instrumentation and sampling (experimental)
- <http://www.score-p.org>

Call-path Profile: Example



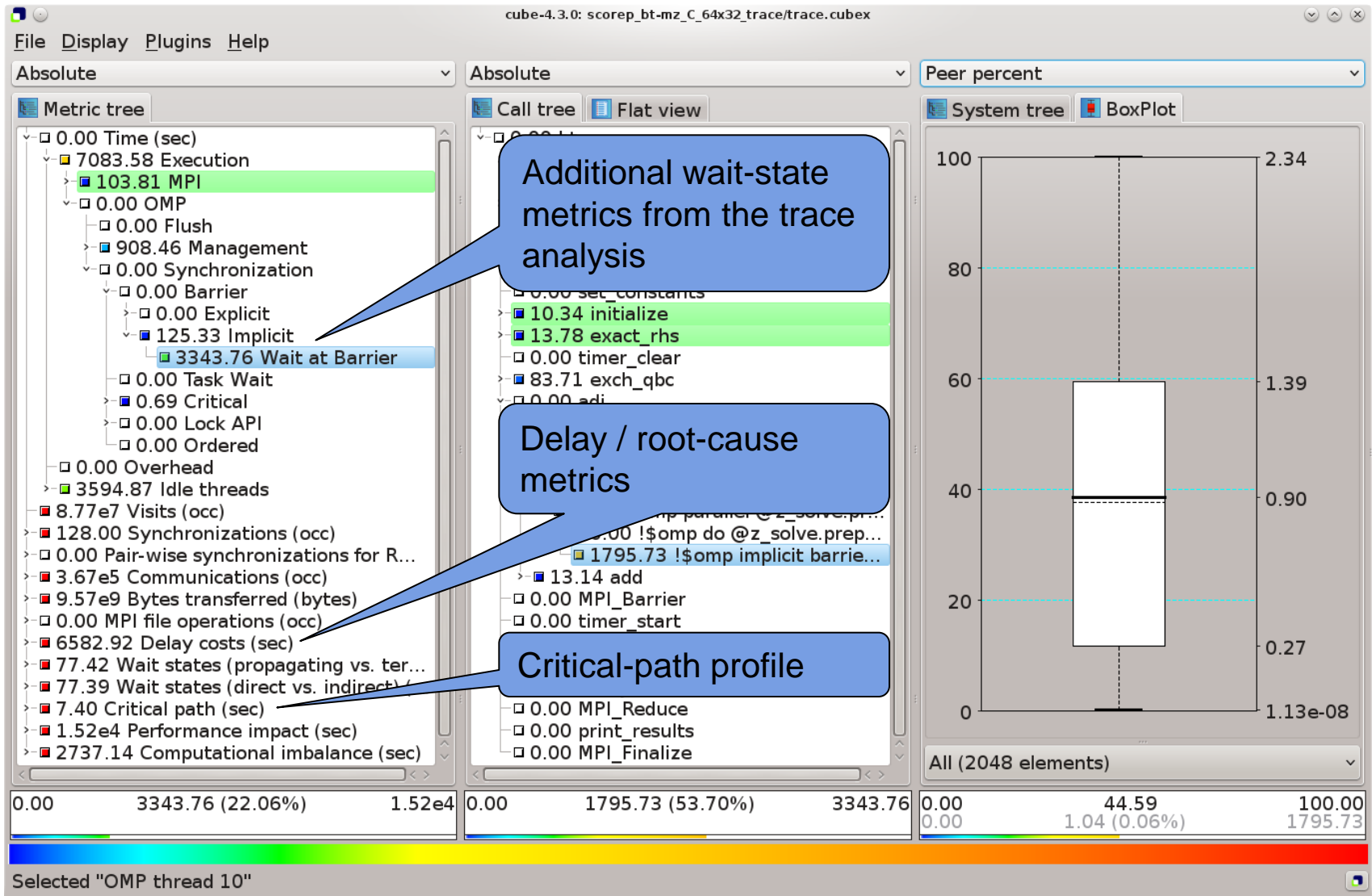
- Collection of trace-based performance analysis tools
 - Specifically designed for large-scale systems
 - Unique features:
 - Scalable, automated search for event patterns representing inefficient behavior
 - Scalable identification of the critical execution path
 - Delay / root-cause analysis
- Based on Score-P for instrumentation and measurement
 - Includes convenience / post-processing commands providing added value
- <http://www.scalasca.org>

Example: Wait-state, Critical Path & Delay Analysis

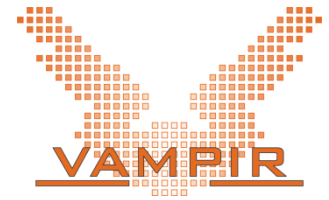


- Searches for *wait states* in communication & synchronization
- Determines a profile of the application's *critical path*
- Identifies *delays* as the root causes of wait states

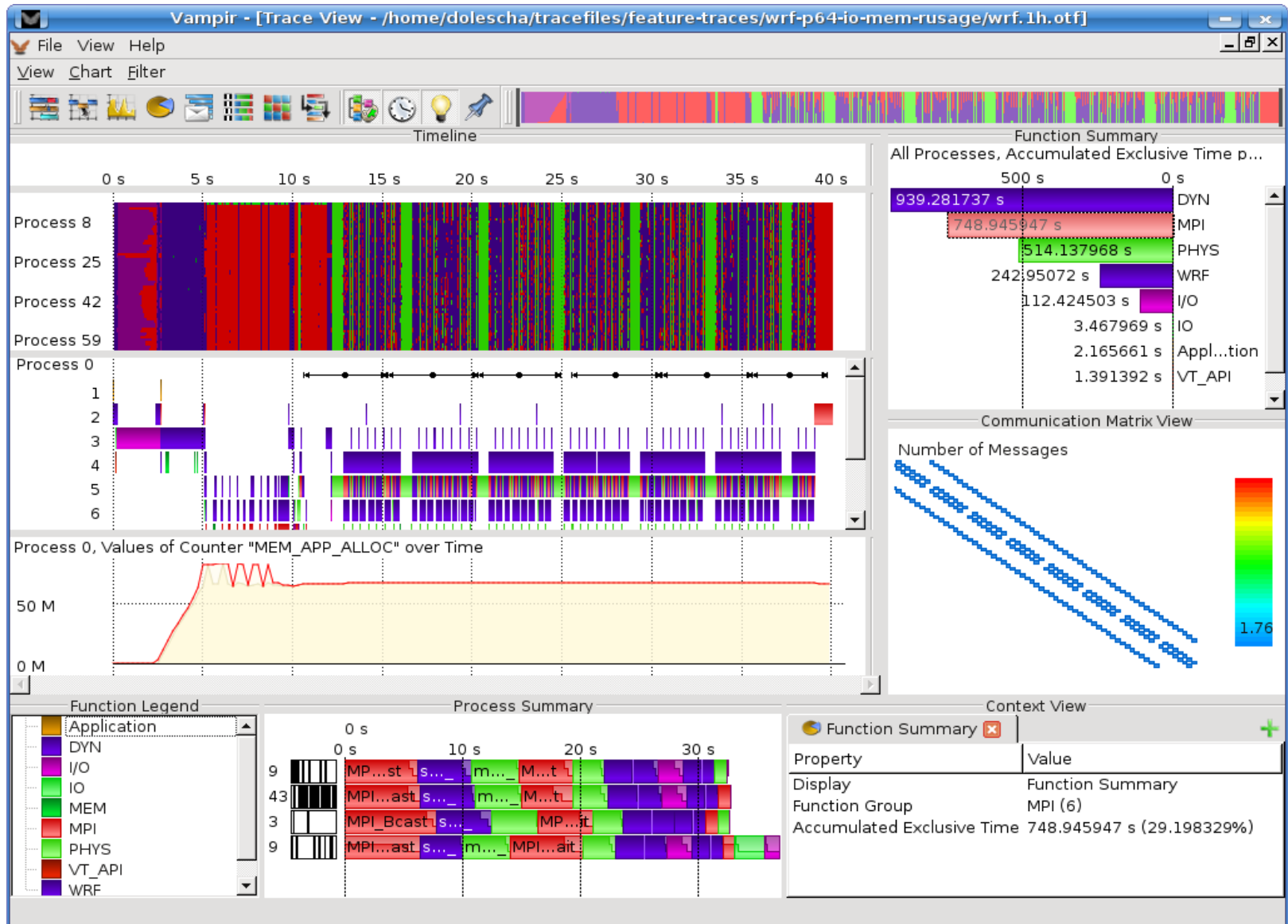
Scalasca Trace Analysis Example



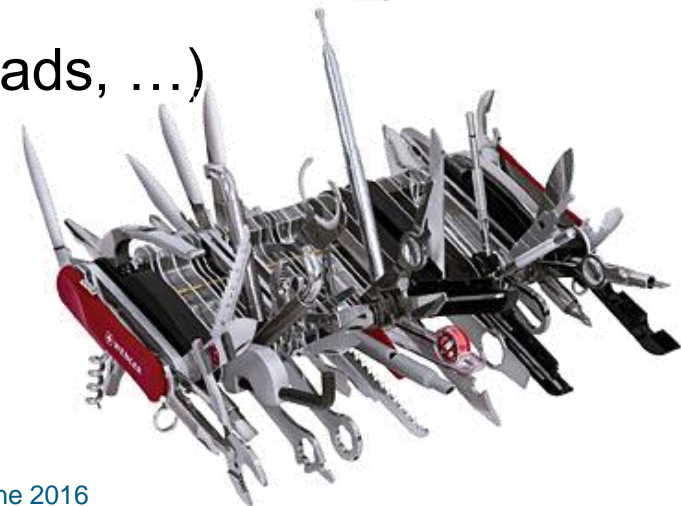
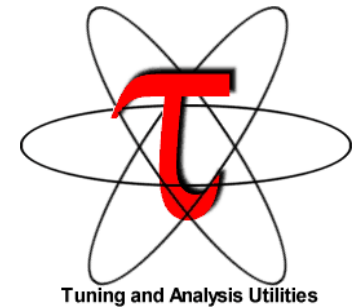
- Offline trace visualization for Score-P's OTF2 trace files
- Visualization of MPI, OpenMP and application events:
 - All diagrams highly customizable (through context menus)
 - Large variety of displays for ANY part of the trace
- <http://www.vampir.eu>
- Advantage:
 - Detailed view of dynamic application behavior
- Disadvantage:
 - Requires event traces (huge amount of data)
 - Completely manual analysis



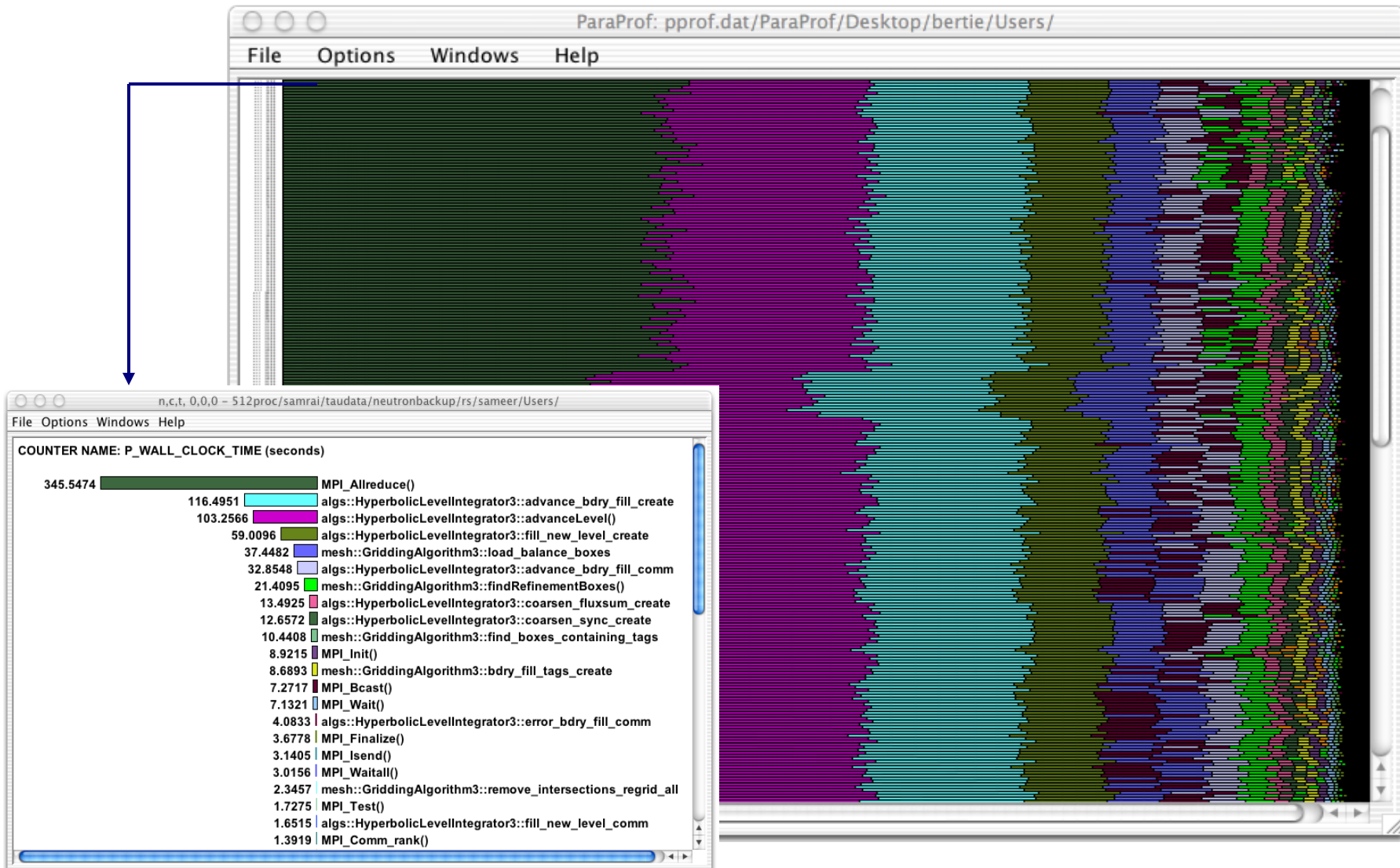
Vampir Displays



- Very portable tool set for instrumentation, measurement and analysis of parallel multi-threaded applications
- <http://tau.uoregon.edu/>
- Supports
 - Various profiling modes and tracing
 - Various forms of code instrumentation
 - C, C++, Fortran, Java, Python
 - MPI, multi-threading (OpenMP, Pthreads, ...)

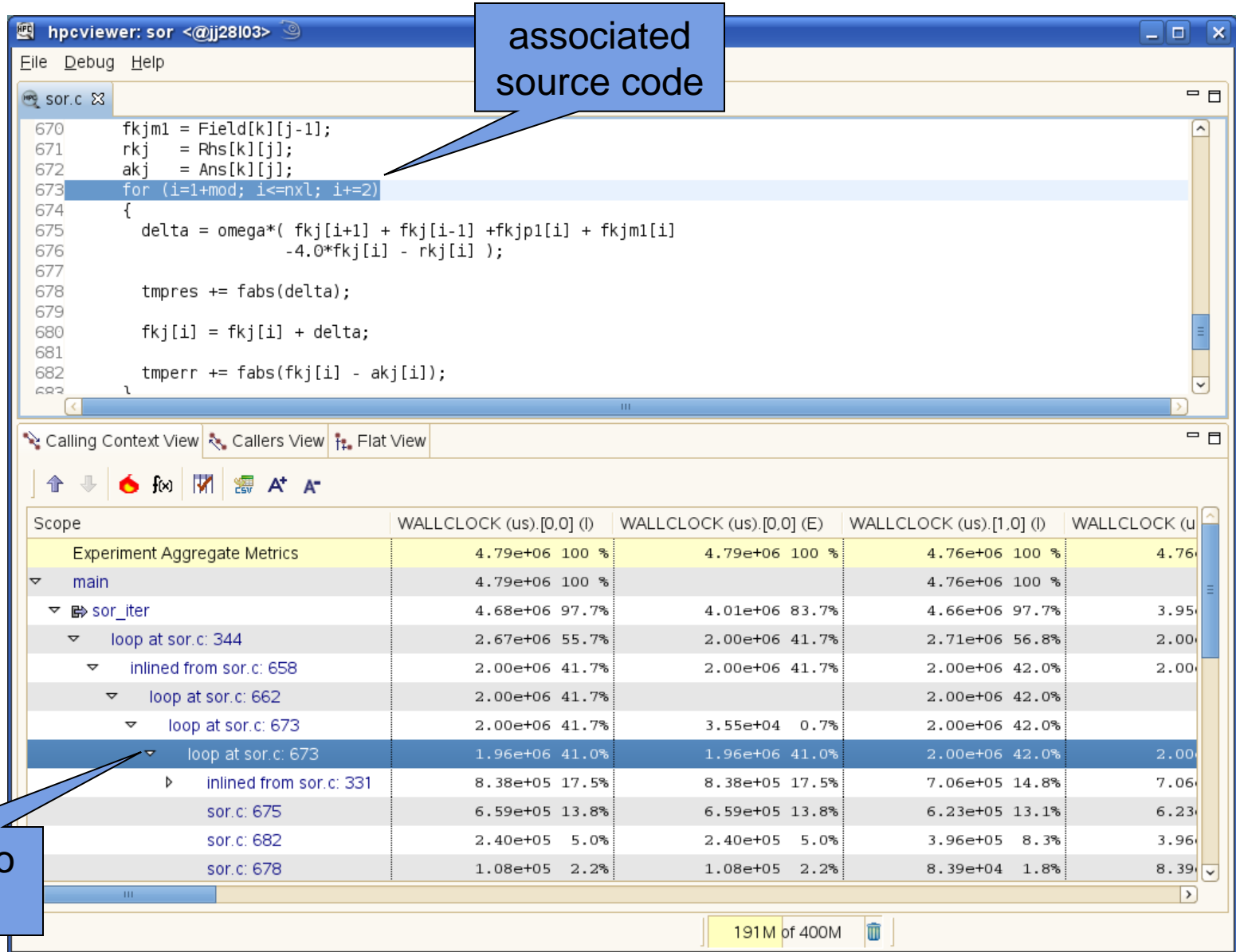


TAU: Basic Profile View



- Multi-platform sampling-based call-path profiler
- Works on unmodified, optimized executables
- <http://hpctoolkit.org>
- Advantages:
 - Overhead can be easily controlled via sampling interval
 - Advantageous for complex C++ codes with many small functions
 - Loop-level analysis (sometimes even individual source lines)
 - Supports POSIX threads
- Disadvantages:
 - Statistical approach that might miss details
 - MPI/OpenMP time displayed as low-level system calls

Example: hpcviewer



The screenshot shows the hpcviewer interface. The top window displays the source code for `sor.c`. A blue callout box labeled "associated source code" points to the `for` loop starting at line 673. The bottom window shows a callpath table with columns for Scope, WALLCLOCK (us).[0.0] (I), WALLCLOCK (us).[0.0] (E), WALLCLOCK (us).[1.0] (I), and WALLCLOCK (us).[1.0] (E). A blue callout box labeled "Callpath to hotspot" points to the "loop at sor.c: 673" entry in the table.

Scope	WALLCLOCK (us).[0.0] (I)	WALLCLOCK (us).[0.0] (E)	WALLCLOCK (us).[1.0] (I)	WALLCLOCK (us).[1.0] (E)
Experiment Aggregate Metrics	4.79e+06 100 %	4.79e+06 100 %	4.76e+06 100 %	4.76e+06 100 %
main	4.79e+06 100 %		4.76e+06 100 %	
sor_iter	4.68e+06 97.7%	4.01e+06 83.7%	4.66e+06 97.7%	3.95e+06 82.5%
loop at sor.c: 344	2.67e+06 55.7%	2.00e+06 41.7%	2.71e+06 56.8%	2.00e+06 41.7%
inlined from sor.c: 658	2.00e+06 41.7%	2.00e+06 41.7%	2.00e+06 42.0%	2.00e+06 42.0%
loop at sor.c: 662	2.00e+06 41.7%		2.00e+06 42.0%	
loop at sor.c: 673	2.00e+06 41.7%	3.55e+04 0.7%	2.00e+06 42.0%	
loop at sor.c: 673	1.96e+06 41.0%	1.96e+06 41.0%	2.00e+06 42.0%	2.00e+06 42.0%
inlined from sor.c: 331	8.38e+05 17.5%	8.38e+05 17.5%	7.06e+05 14.8%	7.06e+05 14.8%
sor.c: 675	6.59e+05 13.8%	6.59e+05 13.8%	6.23e+05 13.1%	6.23e+05 13.1%
sor.c: 682	2.40e+05 5.0%	2.40e+05 5.0%	3.96e+05 8.3%	3.96e+05 8.3%
sor.c: 678	1.08e+05 2.2%	1.08e+05 2.2%	8.39e+04 1.8%	8.39e+04 1.8%

Callpath to hotspot

- Darshan
 - I/O characterization tool logging parallel application file access
 - Shows counts of file access operations, times for key operations, histograms of accesses, etc.
- Intel Advisor
 - Vectorization optimization
- Intel Trace Analyzer and Collector
 - Graphical tool for understanding MPI application behavior
- NVIDIA Visual Profiler
 - GPU performance analysis
 - Supports CUDA and OpenACC

Remark: No Single Solution is Sufficient!



☞ *A combination of different methods, tools and techniques is typically needed!*

- Analysis
 - Statistics, visualization, automatic analysis, data mining, ...
- Measurement
 - Sampling / instrumentation, profiling / tracing, ...
- Instrumentation
 - Source code / binary, manual / automatic, ...

- **VI-HPS** (Virtual Institute - High-Productivity Supercomputing)
 - International collaboration between tool development groups
 - Organizes many tool trainings
 - Full/half-day tutorials at conferences
 - Multi-day “bring-your-own-code” tuning workshops
 - <http://www.vi-hps.org>
- **POP** (Performance Optimization and Productivity)
 - EU Centre of Excellence providing performance optimization services
 - <http://pop-coe.eu>