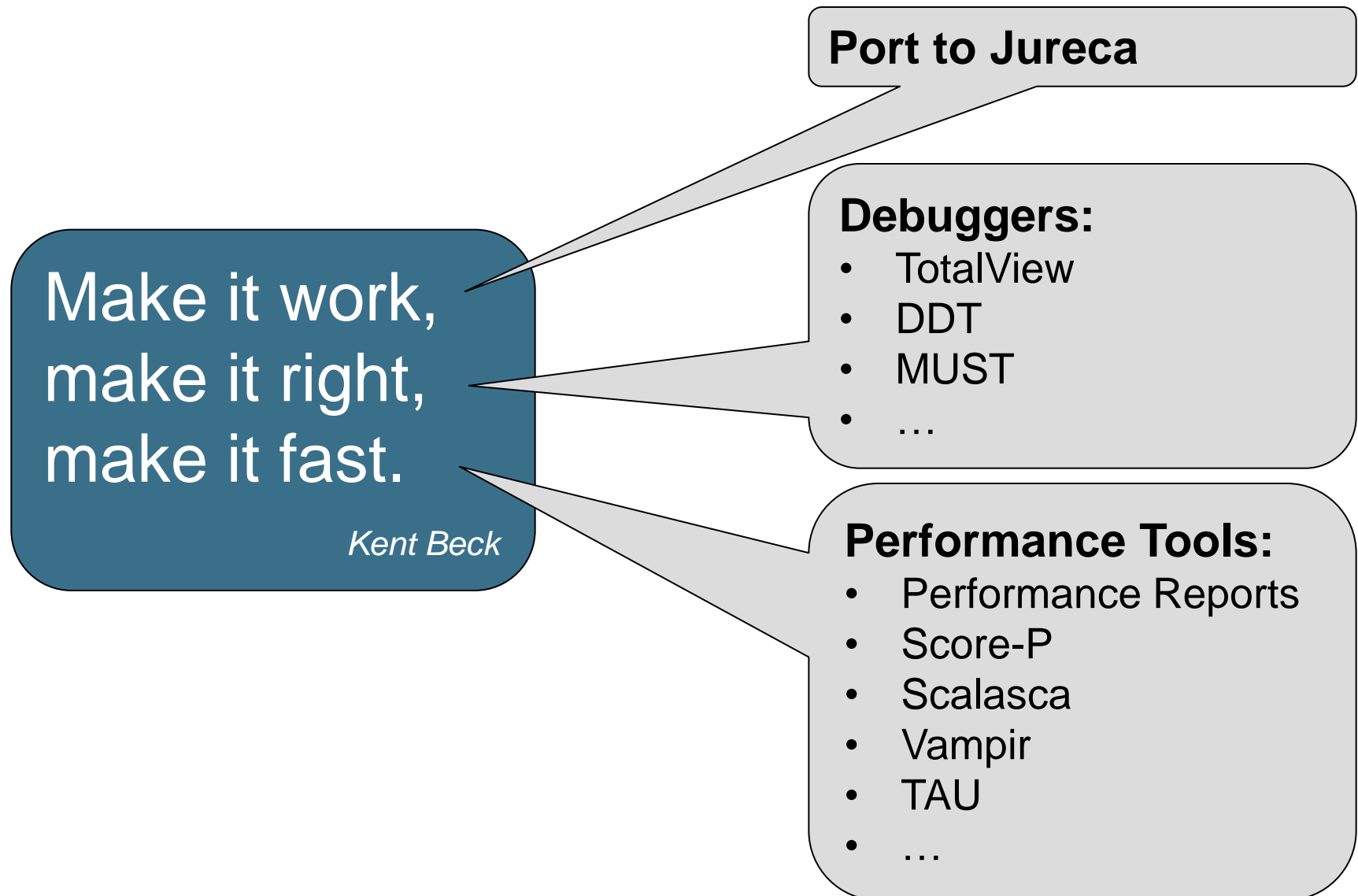


# Debuggers and Performance Tools

June 2016 | Markus Geimer

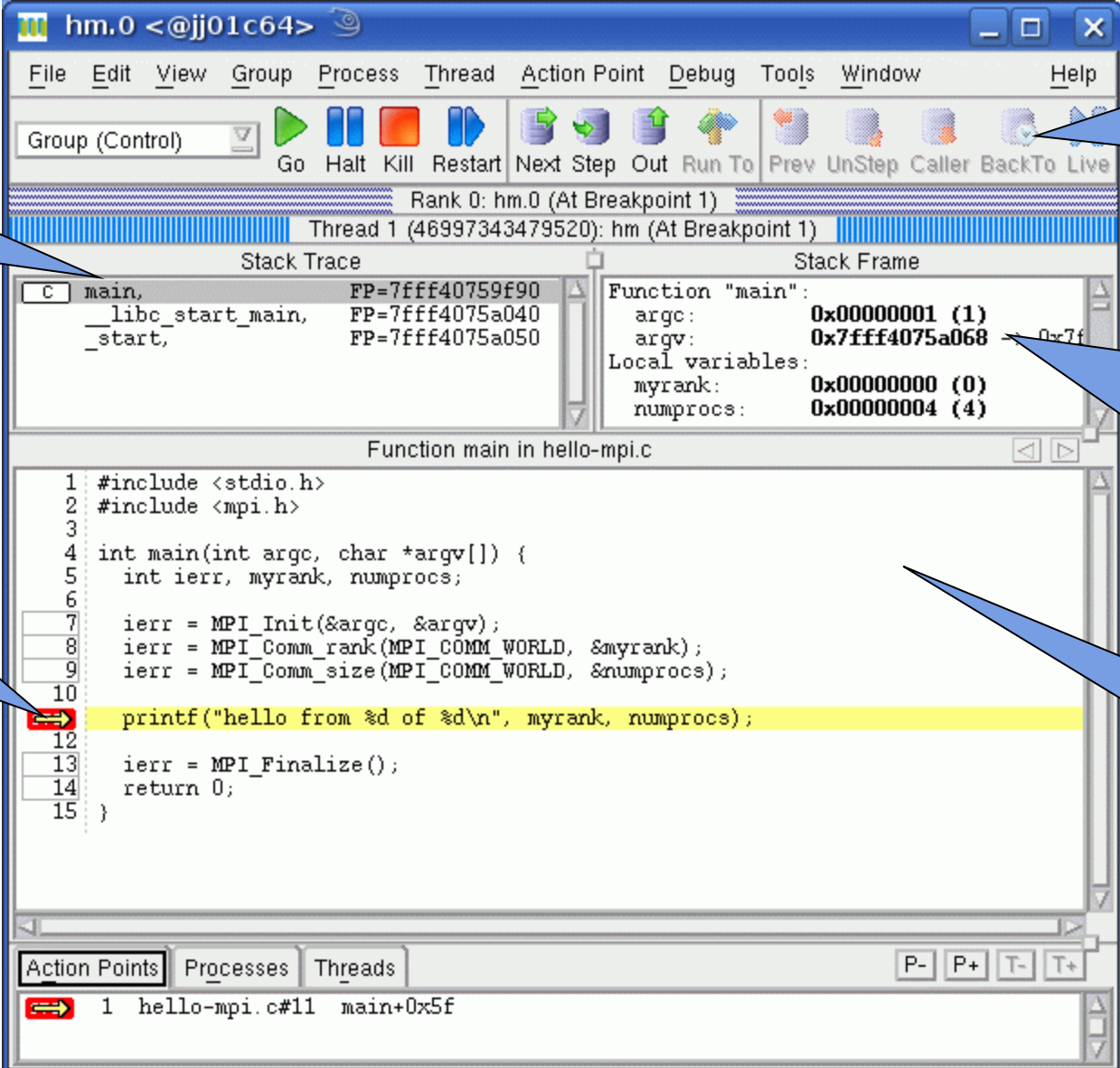


# Debugging Tools



- 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. At the top is a menu bar (File, Edit, View, Group, Process, Thread, Action Point, Debug, Tools, Window, Help) and a toolbar with icons for Go, Halt, Kill, Restart, Next Step, Out, Run To, Prev, UnStep, Caller, BackTo, and Live. Below the toolbar is a status bar showing 'Rank 0: hm.0 (At Breakpoint 1)' and 'Thread 1 (46997343479520): hm (At Breakpoint 1)'. The main window is divided into several panes. The 'Stack Trace' pane on the left shows a list of stack frames: 'main' (FP=7fff40759f90), '\_libc\_start\_main' (FP=7fff4075a040), and '\_start' (FP=7fff4075a050). The 'Stack Frame' pane on the right shows the details for the selected 'main' frame, including arguments (argc=0x00000001, argv=0x7fff4075a068) and local variables (myrank=0x00000000, numprocs=0x00000004). The 'Source code' pane at the bottom shows the source code for 'hello-mpi.c', with a breakpoint set at line 11. The 'Breakpoints' pane at the bottom left shows a list of breakpoints, with one breakpoint set at 'hello-mpi.c#11 main+0x5f'. Callouts point to various features: 'Stack trace' points to the Stack Trace pane, 'Break points' points to the Breakpoints pane, 'Toolbar for common options' points to the toolbar, 'Local variables for selected stack frame' points to the Stack Frame pane, and 'Source code window' points to the Source code pane.

Stack trace

Break points

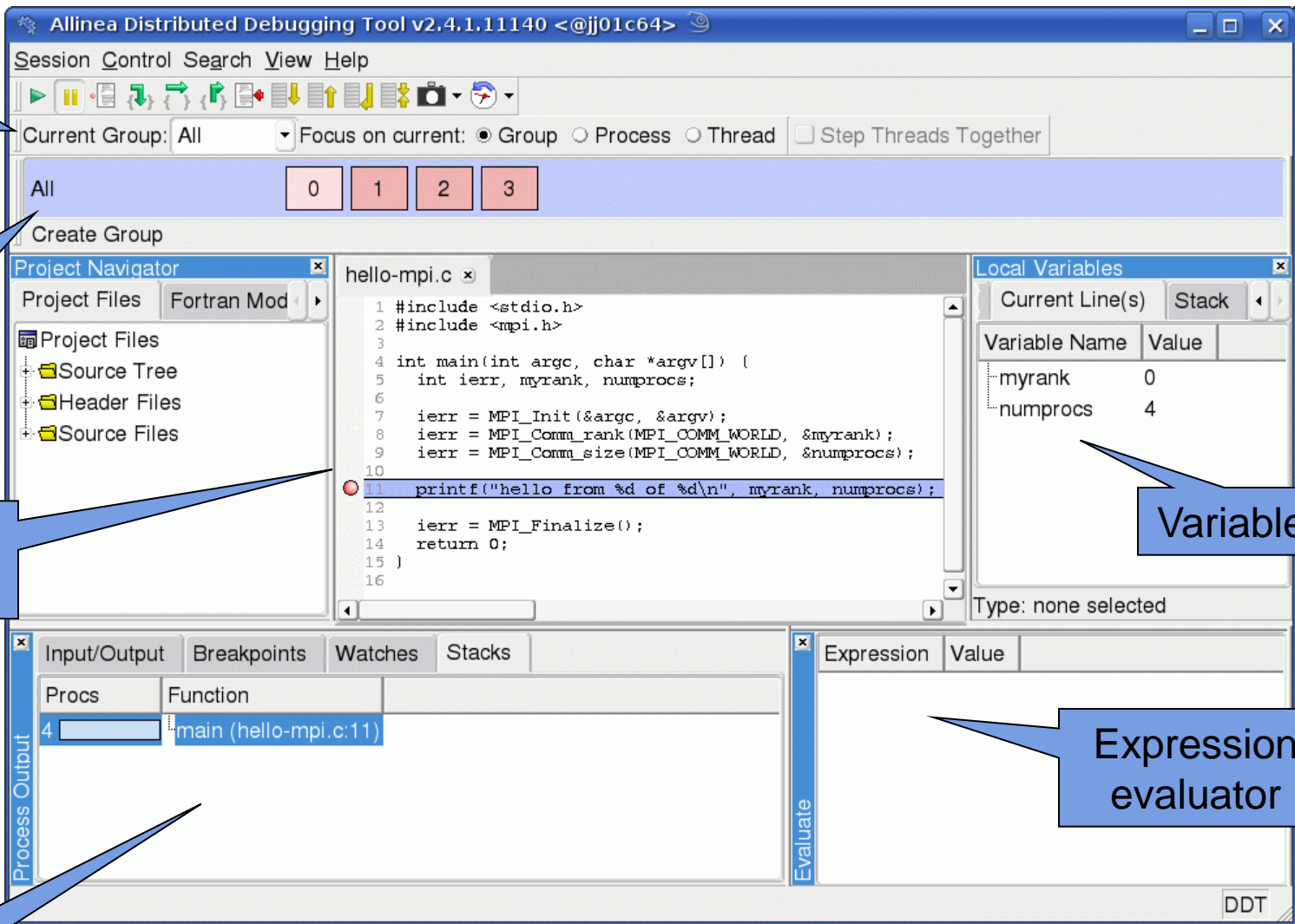
Toolbar for common options

Local variables for selected stack frame

Source code window

- 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 is divided into several panes:

- Process controls:** Located at the top, it includes a menu bar (Session, Control, Search, View, Help), a toolbar with various icons, and a section for "Current Group: All" with buttons for "0", "1", "2", and "3".
- Process groups:** A "Create Group" button is located below the process controls.
- Source code:** The central pane displays the source code for "hello-mpi.c". The code includes headers, defines a main function, and uses MPI functions. Line 11 is highlighted with a red circle.
- Variables:** The "Local Variables" pane on the right shows the current line(s) and a table of variables. The table has columns "Variable Name" and "Value".
- Expression evaluator:** The "Expression" pane on the right allows for evaluating expressions. It has columns "Expression" and "Value".
- Stack trace:** The "Input/Output" pane at the bottom left shows a table of processes and functions. The table has columns "Procs" and "Function".

Variable Name	Value
myrank	0
numprocs	4

Procs	Function
4	main (hello-mpi.c:11)

- 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

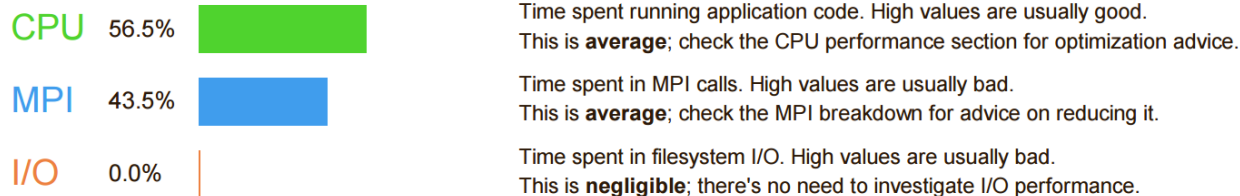
# Performance Analysis Tools

- **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

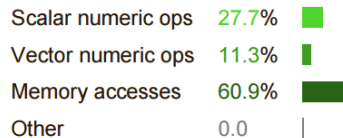
The total wallclock time was spent as follows:



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:

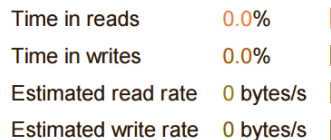


The per-core performance is **memory-bound**. Use a profiler to identify time-consuming loops and check their cache performance.

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

### I/O

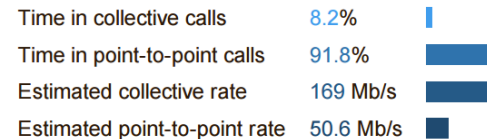
A breakdown of how the 0.0% total I/O time was spent:



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

### MPI

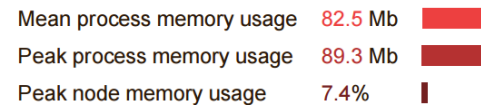
Of the 43.5% total time spent in MPI calls:



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.



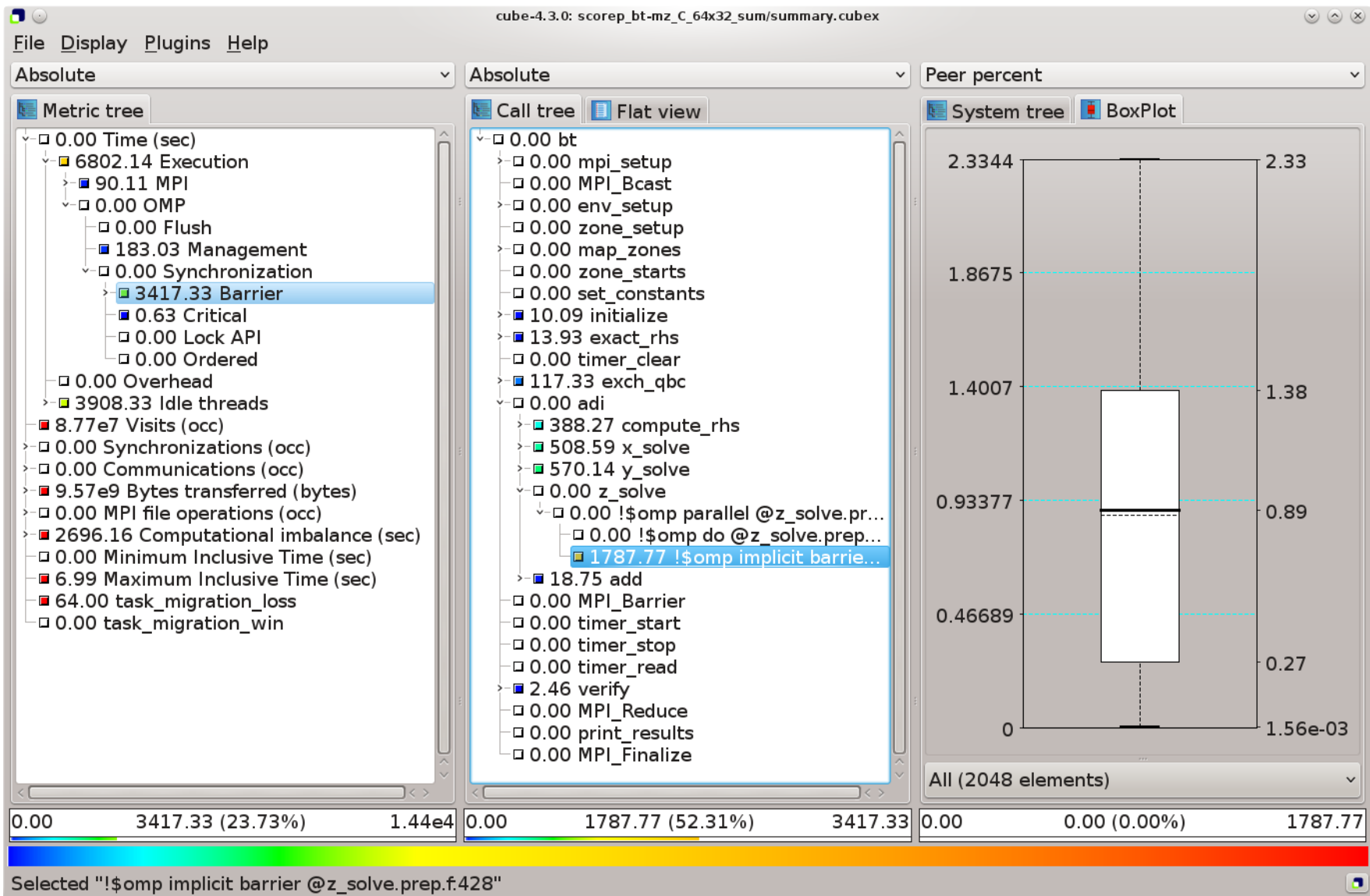
- Community instrumentation and measurement infrastructure
  - Developed by a consortium of performance tool groups



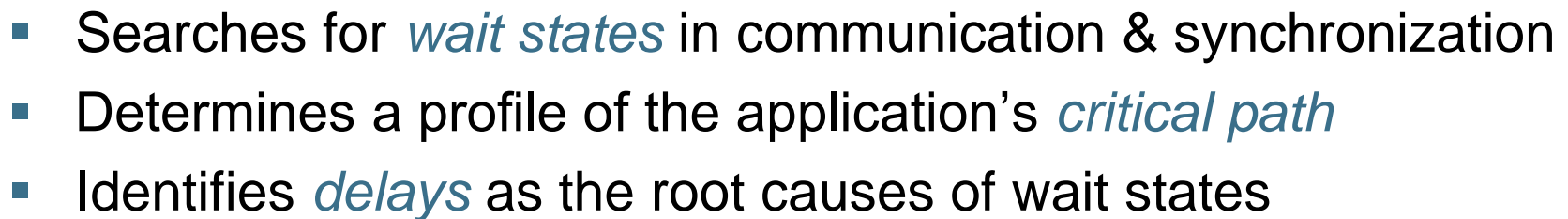
UNIVERSITY OF OREGON

- 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>

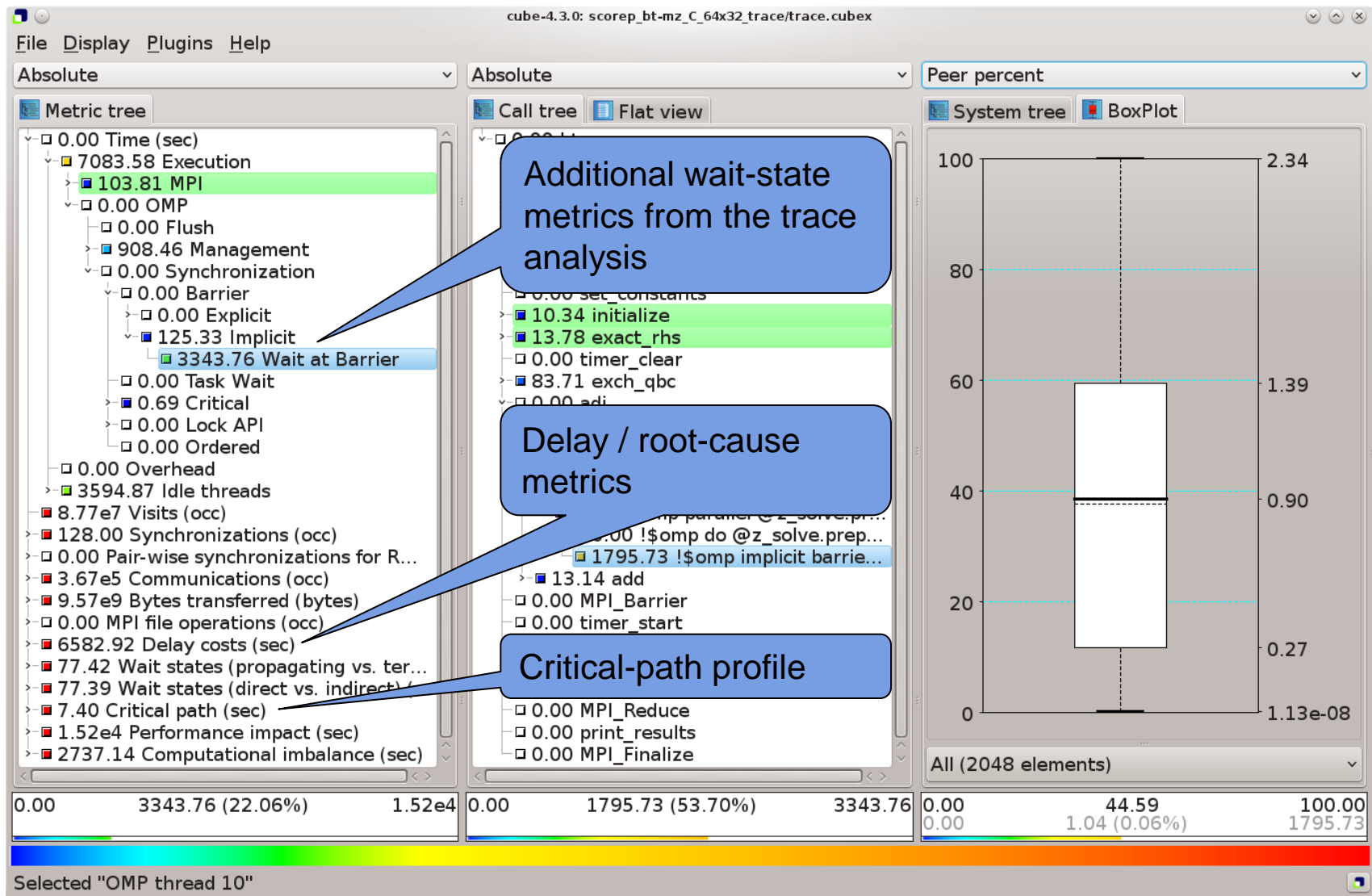
# Call-path Profile: Example

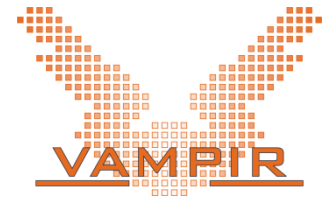


- Collection of trace-based performance analysis tools
  - Specifically designed for large-scale systems
  - Unique features:
    - Scalable, automated search for event patterns representing inefficient behavior
    - Scalable identification of the critical execution path
    - Delay / root-cause analysis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



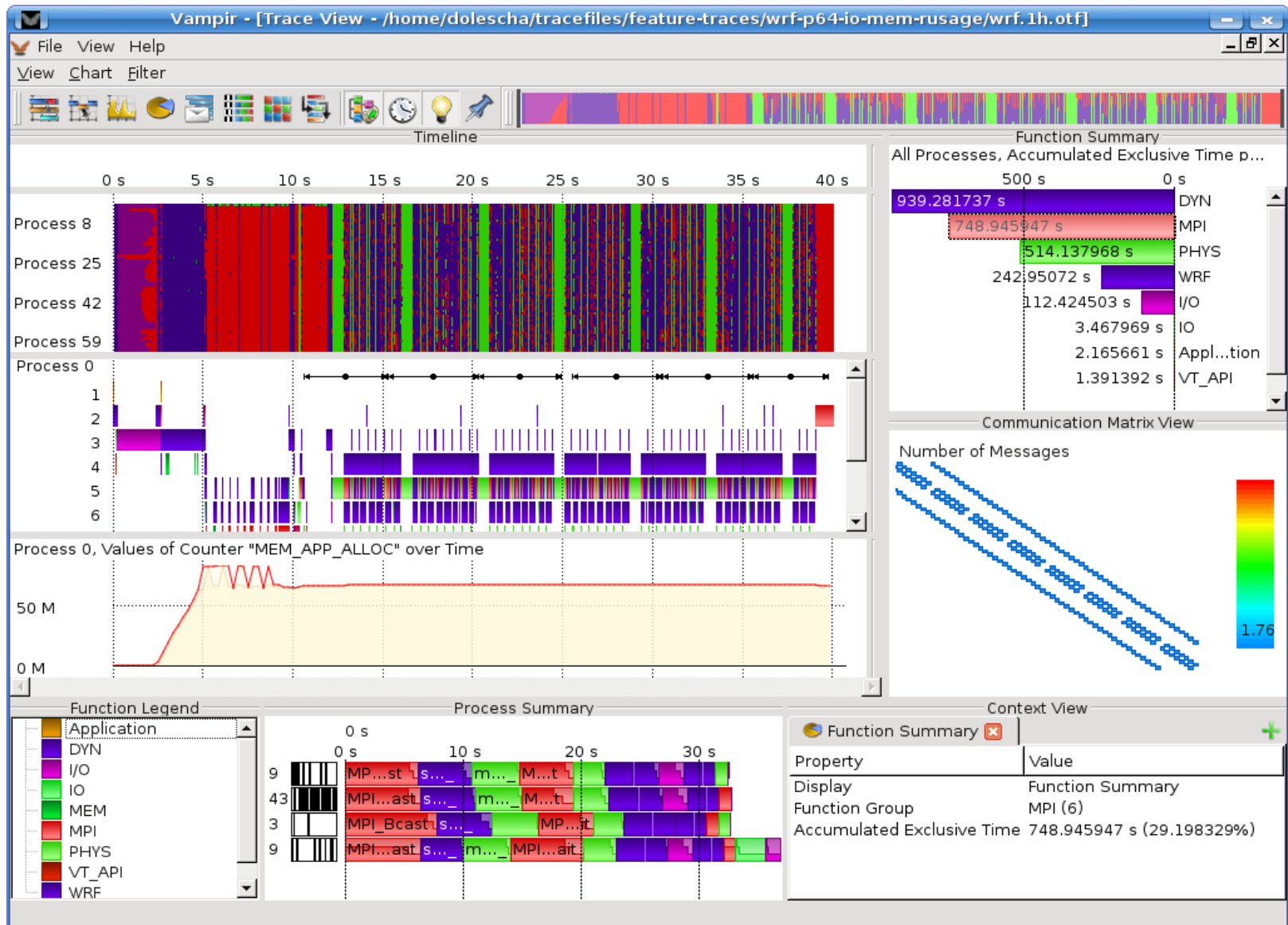
# Scalasca Trace Analysis Example



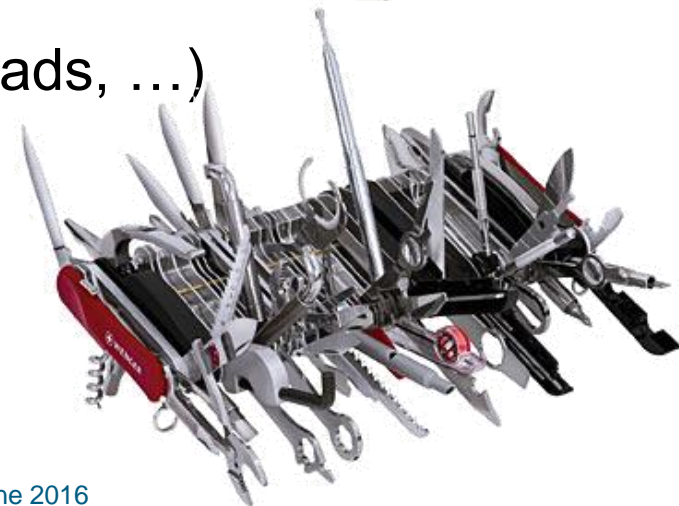
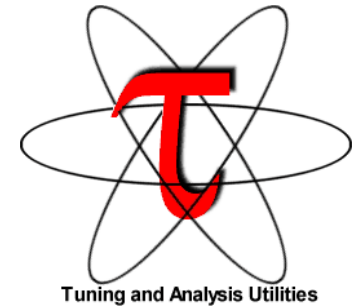


- Offline trace visualization for Score-P's OTF2 trace files
- Visualization of MPI, OpenMP and application events:
  - All diagrams highly customizable (through context menus)
  - Large variety of displays for ANY part of the trace
- <http://www.vampir.eu>
- Advantage:
  - Detailed view of dynamic application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

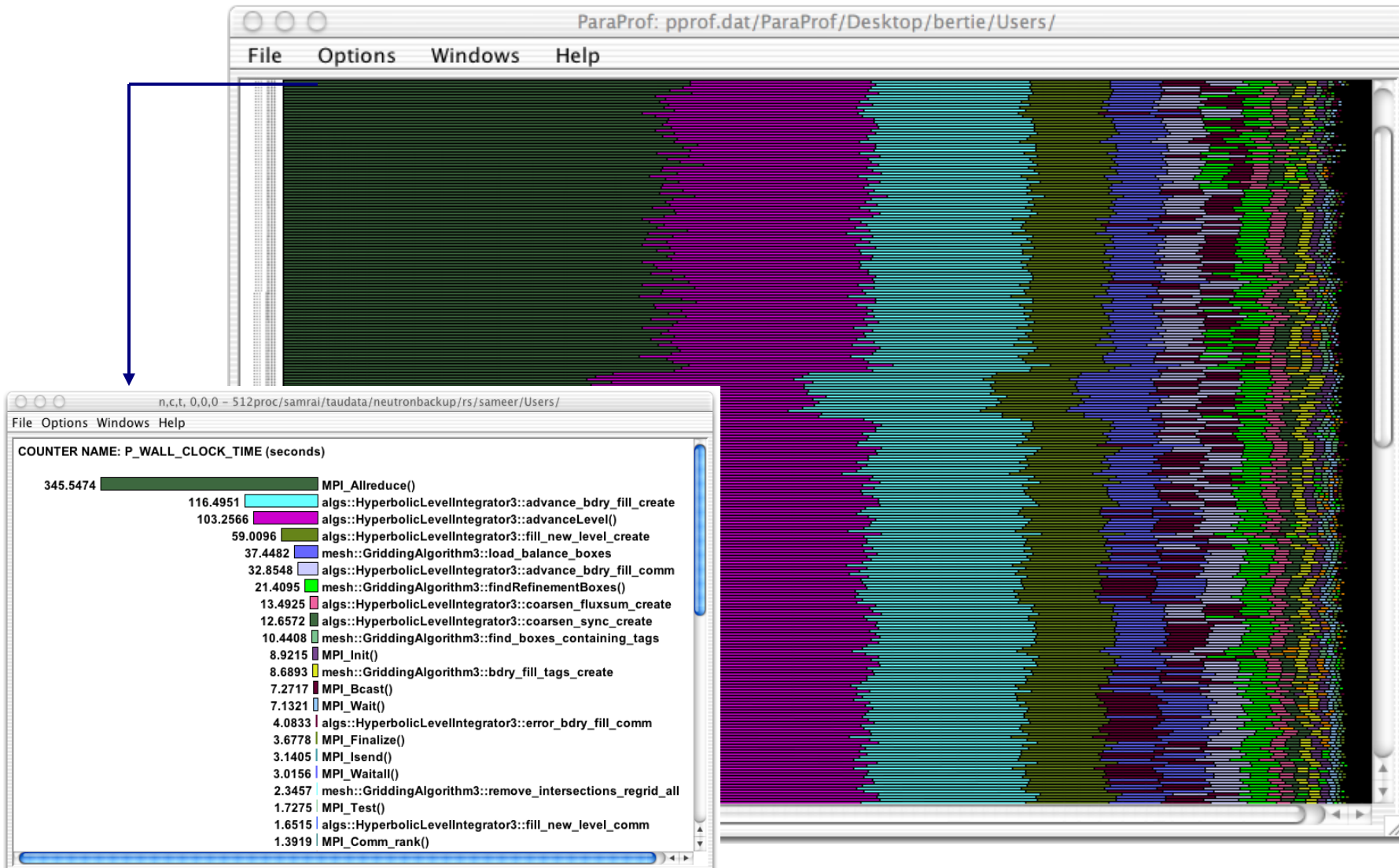
# Vampir Displays



- 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, ...)

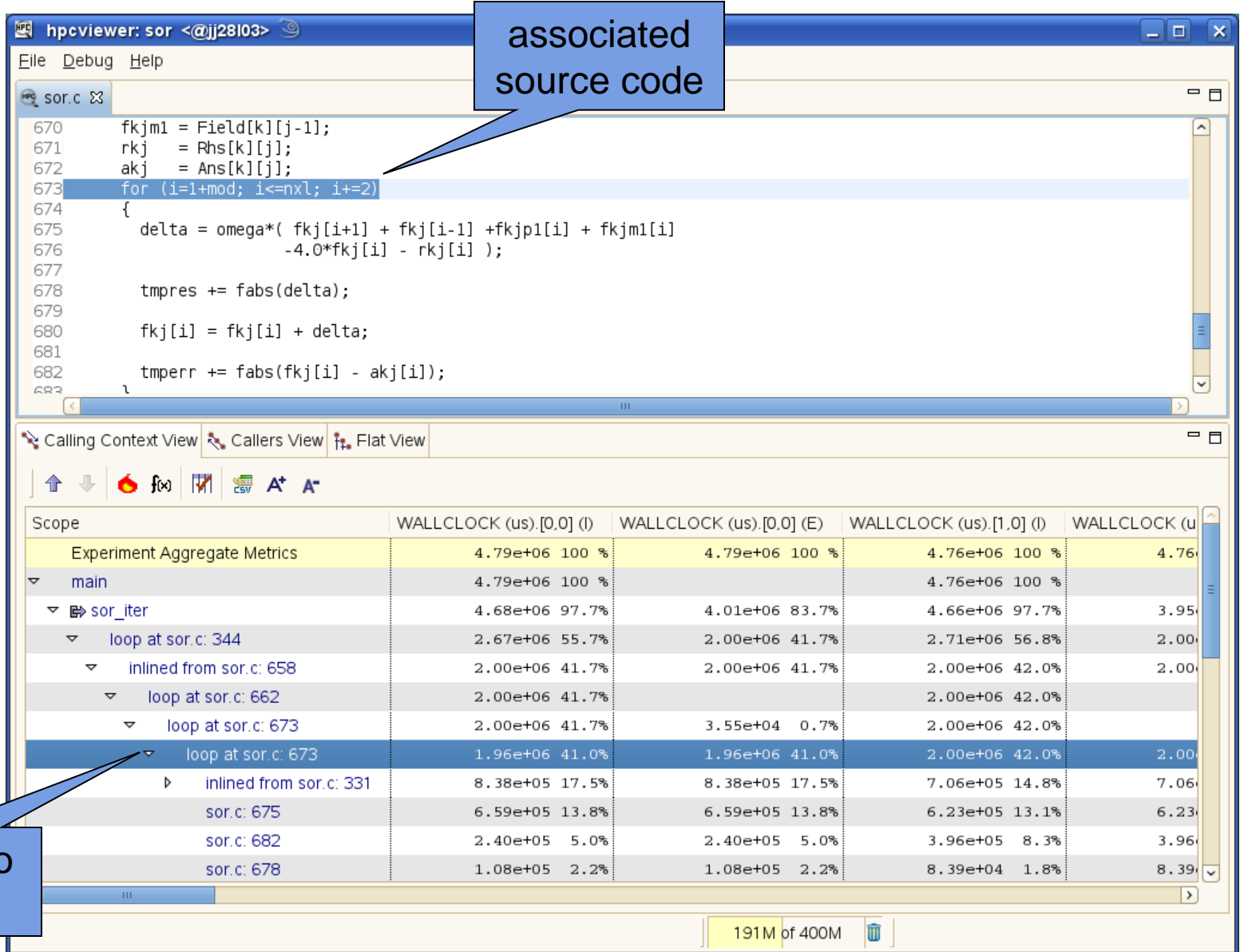


# TAU: Basic Profile View



- 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 application window. The top pane 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 pane shows the "Calling Context View" with a table of performance metrics. A blue callout box labeled "Callpath to hotspot" points to the "loop at sor.c: 673" entry in the table.

hpcviewer: sor <@jj28103>

File Debug Help

sor.c

```
670  fkjm1 = Field[k][j-1];
671  rkj   = Rhs[k][j];
672  akj   = Ans[k][j];
673  for (i=1+mod; i<=nxl; i+=2)
674  {
675      delta = omega*( fkj[i+1] + fkj[i-1] +fkjpl[i] + fkjm1[i]
676                  -4.0*fkj[i] - rkj[i] );
677
678      tmpres += fabs(delta);
679
680      fkj[i] = fkj[i] + delta;
681
682      tmperr += fabs(fkj[i] - akj[i]);
683  }
```

Calling Context View Callers View Flat View

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 %	4.79e+06 100 %	4.76e+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 83.0 %
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 41.7 %	2.00e+06 42.0 %	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 %	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 %

191M of 400M

- 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>