EuroPar'13 Tutorial:
Tools for
High Productivity Supercomputing

26 August 2013

Brian Wylie
Jülich Supercomputing Centre

Martin Schulz
Lawrence Livermore Nat'l Lab

Agenda

Time	Topic	Presenter
09:00	Introduction to VI-HPS & Linux ISO	
	Parallel application engineering & workflow	
	Extreme-scale case studies	
10:30	*Break*	
11:00	Execution monitoring, checking & debugging	
	Demo: ***MUST*** MPI correctness checking	J. Protze
12:30	*Lunch*	
14:30	Integrated application execution profile & trace analysis	
	Demo: ***Scalasca/Score-P*** instrumentation & measurement	B. Wylie
	Demo: ***Vampir*** interactive trace analysis	R. Tschüter
	Demo: ***Periscope*** on-line automated analysis	I. Compres
16:00	*Break*	
16:30	Complementary tools & utilities	
	Demo: ***O|SS*** parallel performance framework	M. Schulz
17:45	Review & discussion	
18:00	*Adjourn*	

Introduction to VI-HPS

Brian Wylie
Jülich Supercomputing Centre

Goal: Improve the quality and accelerate the development process of complex simulation codes running on highly-parallel computer systems

- Start-up funding (2006–2011) by Helmholtz Association of German Research Centres

HELMHOLTZ | ASSOCIATION

- Activities
  - Development and integration of HPC programming tools
    - Correctness checking & performance analysis
  - Training workshops
  - Service
    - Support email lists
    - Application engagement
  - Academic workshops

http://www.vi-hps.org

VI-HPS partners (founders)

Forschungszentrum Jülich

- Jülich Supercomputing Centre

RWTH Aachen University

- Centre for Computing & Communication

Technical University of Dresden

- Centre for Information Services & HPC

University of Tennessee (Knoxville)

- Innovative Computing Laboratory

VI-HPS partners (cont.)

Barcelona Supercomputing Center

- Centro Nacional de Supercomputación

German Research School

- Laboratory of Parallel Programming

Lawrence Livermore National Lab.

- Centre for Applied Scientific Computing

Technical University of Munich

- Chair for Computer Architecture

University of Oregon

- Performance Research Laboratory

University of Stuttgart

- HPC Centre

University of Versailles St-Quentin

- LRC ITACA

Productivity tools

MUST

- MPI usage correctness checking

PAPI

- Interfacing to hardware performance counters

Periscope

- Automatic analysis via an on-line distributed search

Scalasca

- Large-scale parallel performance analysis

TAU

- Integrated parallel performance system

Vampir

- Interactive graphical trace visualization & analysis

Score-P

- Community instrumentation & measurement infrastructure

Productivity tools (cont.)

KCachegrind
- Callgraph-based cache analysis [x86 only]

MAQAO
- Assembly instrumentation & optimization [x86 only]

mpiP/mpiPview
- MPI profiling tool and analysis viewer

Open MPI
- Integrated memory checking

Open|Speedshop
- Integrated parallel performance analysis environment

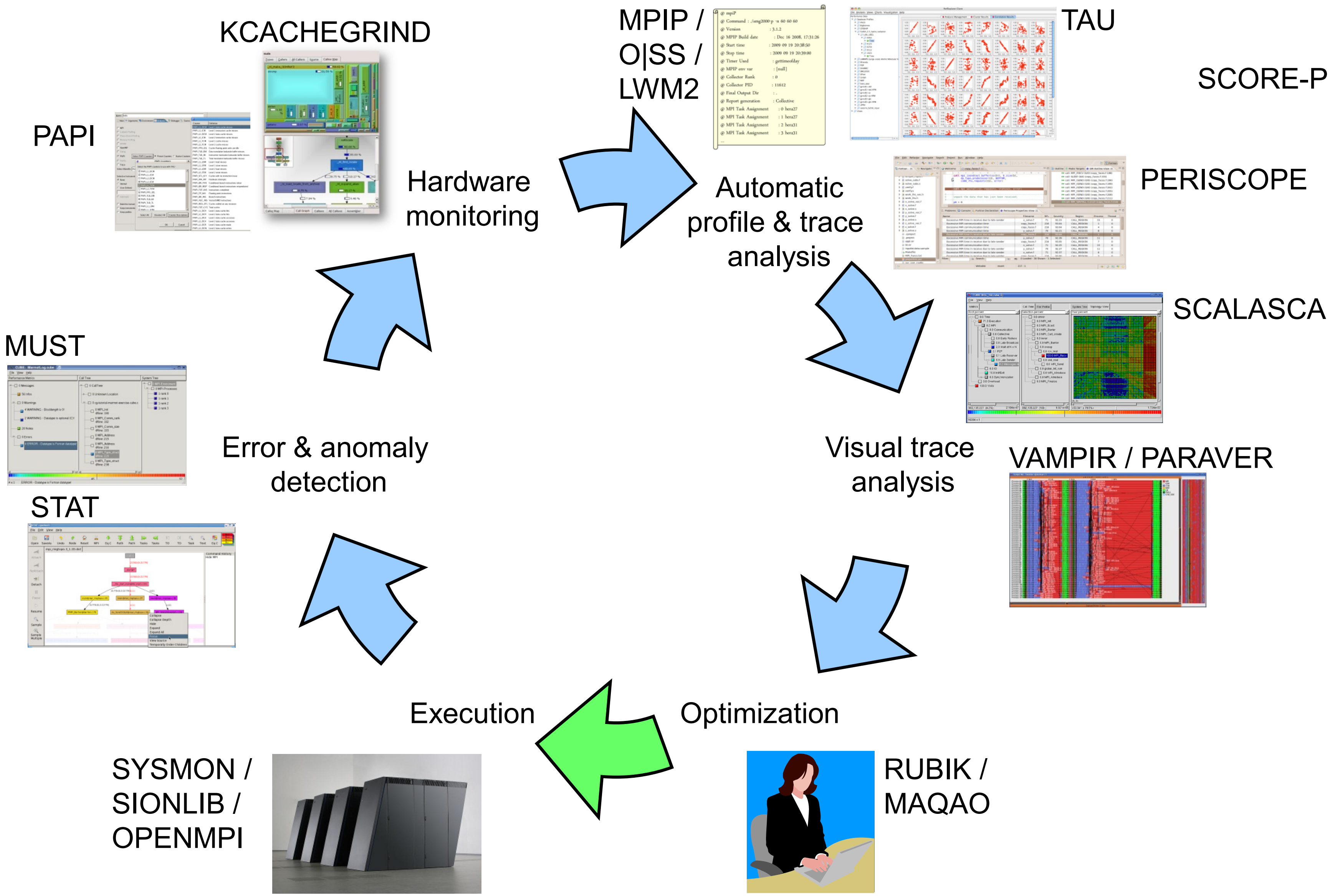
Paraver/Extrae
- Event tracing and graphical trace visualization & analysis

Rubik
- Process mapping generation & optimization [BG only]

SIONlib
- Optimized native parallel file I/O

STAT
- Stack trace analysis tools

Technologies and their integration



Tools will ***not*** automatically make you, your applications or computer systems more *productive*.

However, they can help you understand ***how*** your parallel code executes and ***when / where*** it's necessary to work on *correctness* and *performance* issues.

VI-HPS training & Tuning Workshops

- Goals
  - Give an overview of the programming tools suite
  - Explain the functionality of individual tools
  - Teach how to use the tools effectively
  - Offer hands-on experience and expert assistance using tools
  - Receive feedback from users to guide future development
- For best results, bring & analyze/tune your own code(s)!

- VI-HPS Hands-on Tutorial series
  - SC'08, ICCS'09, SC'09, Cluster'10, SC'10, SC'11, EuroMPI'12, XSEDE'13 (San Diego), **SC'13 (Denver)**
- VI-HPS Tuning Workshop series
  - 2008 (Aachen & Dresden), 2009 (Jülich & Bremen), 2010 (Garching & Amsterdam/NL), 2011 (Stuttgart & Aachen), 2012 (St-Quentin/F & Garching), 2013 (Saclay/F & Jülich)

Recent & upcoming events

- SC’13 Hands-on Tutorials (17&18 Nov 2013, Denver)
  - Score-P/Scalasca/Vampir/TAU, MUST, O|SS, Paraver
- 12th VI-HPS Tuning Workshop (7-11 Oct 2013, Jülich)
  - Hosted by Jülich Supercomputing Centre, FZJ, Germany
  - Using PRACE Tier-0 *Juqueen* BlueGene/Q system
  - Score-P, Scalasca, Vampir, TAU, Periscope, Paraver, MUST, ...
- Further events to be determined
  - (one-day) tutorials
    - With guided exercises usually using a Live-DVD
  - (multi-day) training workshops
    - With your own applications on actual HPC systems
- Check www.vi-hps.org/training for announced events
- Contact us if you might be interested in hosting an event

- Bootable Linux installation on DVD (or USB memory stick)
- Includes everything needed to try out our parallel tools on an 64-bit x86-architecture notebook computer
  - VI-HPS tools: MUST, PAPI, Score-P, Periscope, Scalasca, TAU, Vampir*
  - Also: Eclipse/PTP, TotalView*, etc.
    - * time/capability-limited evaluation licences provided for commercial products
  - GCC (w/ OpenMP), OpenMPI
  - Manuals/User Guides
  - Tutorial exercises & examples
- Produced by U. Oregon PRL
  - Sameer Shende

VI-HPS Linux Live ISO

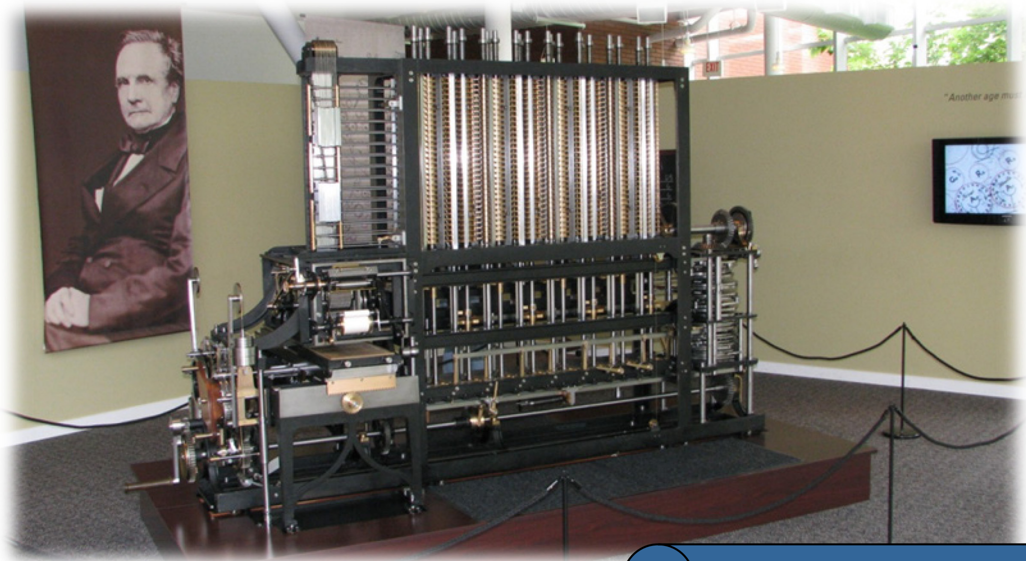
- ISO image approximately 4GB
  - download latest version from website
  - http://www.vi-hps.org/training/livedvd
  - optionally create bootable DVD or USB drive
- Boot directly from disk
  - enables hardware counter access and offers best performance, but no save/resume
- Boot within virtual machine
  - faster boot time and can save/resume state, but may not allow hardware counter access
- Boots into Linux environment for HPC
  - supports building and running provided MPI and/or OpenMP parallel application codes
  - and experimentation with VI-HPS (and third-party) tools

Introduction to Parallel Performance Engineering

Markus Geimer, Brian Wylie
Jülich Supercomputing Centre

(with content used with permission from tutorials by Bernd Mohr/JSC and Luiz DeRose/Cray)

Performance: an old problem



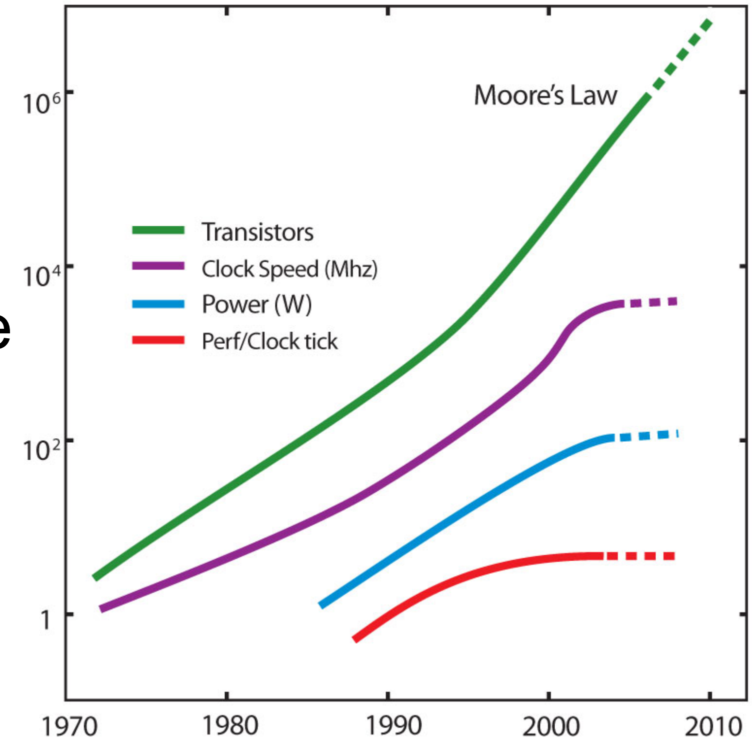
Difference Engine

"The most constant difficulty in contriving the engine has arisen from the desire to reduce the time in which the calculations were executed to the shortest which is possible."

Charles Babbage
1791 – 1871

Today: the "free lunch" is over

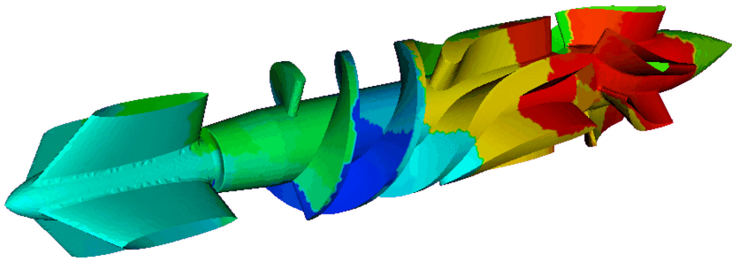
- Moore's law is still in charge, but
  - Clock rates no longer increase
  - Performance gains only through increased parallelism
- Optimizations of applications more difficult
  - Increasing application complexity
    - Multi-physics
    - Multi-scale
  - Increasing machine complexity
    - Hierarchical networks / memory
    - More CPUs / multi-core



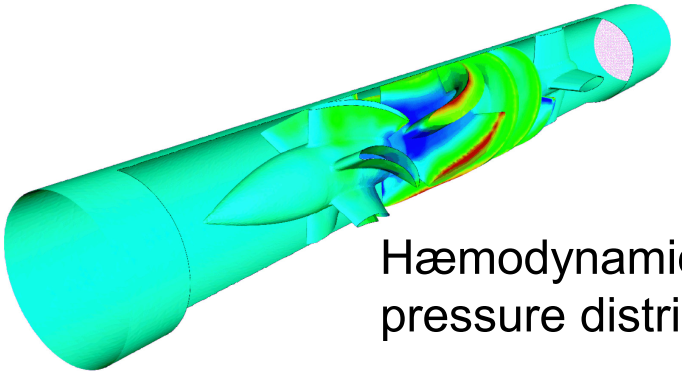
☞ Every doubling of scale reveals a new bottleneck!

Example: XNS

- CFD simulation of unsteady flows
  - Developed by CATS / RWTH Aachen
  - Exploits finite-element techniques, unstructured 3D meshes, iterative solution strategies
- MPI parallel version
  - >40,000 lines of Fortran & C
  - DeBakey blood-pump data set (3,714,611 elements)

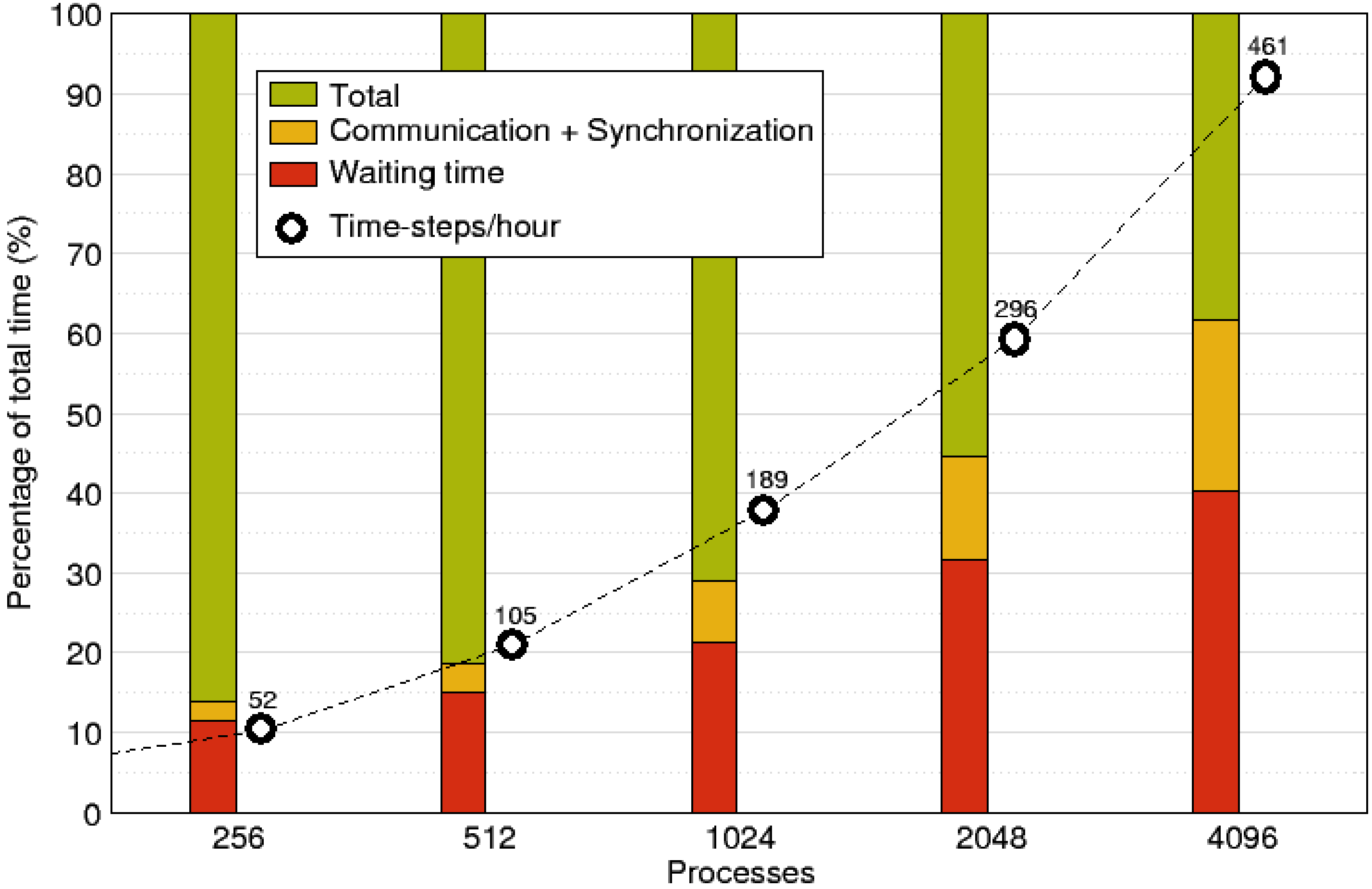


Partitioned finite-element mesh



Hæmodynamic flow pressure distribution

XNS wait-state analysis on BG/L (2007)



Performance factors of parallel applications

- "Sequential" factors
  - Computation
    - ☞Choose right algorithm, use optimizing compiler
  - Cache and memory
    - ☞Tough! Only limited tool support, hope compiler gets it right
  - Input / output
    - ☞Often not given enough attention
- "Parallel" factors
  - Partitioning / decomposition
  - Communication (i.e., message passing)
  - Multithreading
  - Synchronization / locking
    - ☞More or less understood, good tool support

Tuning basics

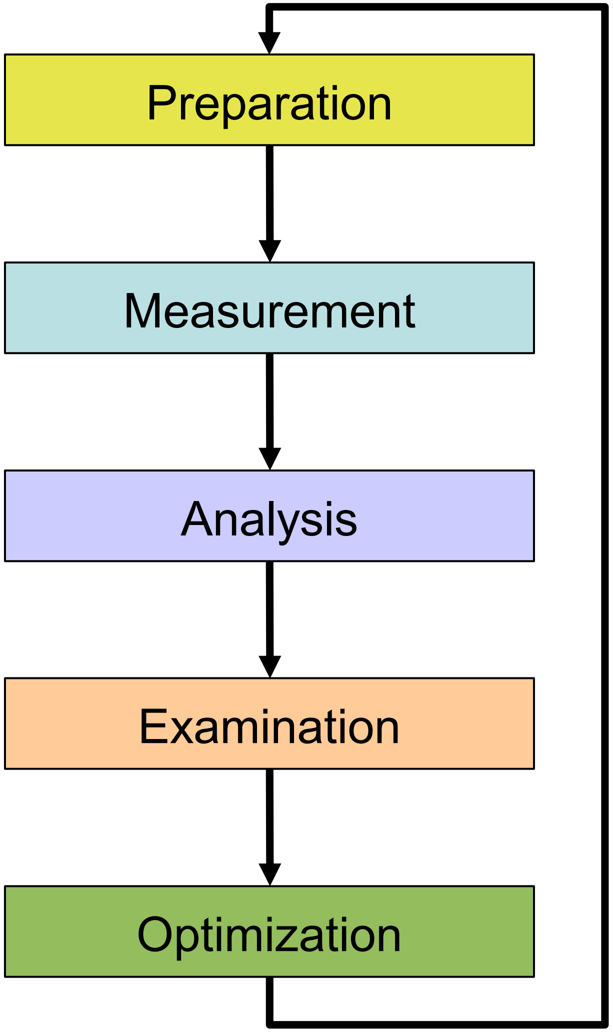
- Successful engineering is a combination of
  - The right algorithms and libraries
  - Compiler flags and directives
  - Thinking !!!
- Measurement is better than guessing
  - To determine performance bottlenecks
  - To compare alternatives
  - To validate tuning decisions and optimizations
    - ☞After each step!

However…

> "We should forget about small efficiencies, say 97% of the time: premature optimization is the root of all evil."
>
> Charles A. R. Hoare

- It's easier to optimize a slow correct program than to debug a fast incorrect one
  - ☞ *Nobody cares how fast you can compute a wrong answer...*

Performance engineering workflow



- Prepare application (with symbols), insert extra code (probes/hooks)
- Collection of data relevant to execution performance analysis
- Calculation of metrics, identification of performance metrics
- Presentation of results in an intuitive/understandable form
- Modifications intended to eliminate/reduce performance problems

The 80/20 rule

- Programs typically spend 80% of their time in 20% of the code
- Programmers typically spend 20% of their effort to get 80% of the total speedup possible for the application
  - ☞ *Know when to stop!*

- Don't optimize what does not matter
  - ☞ *Make the common case fast!*

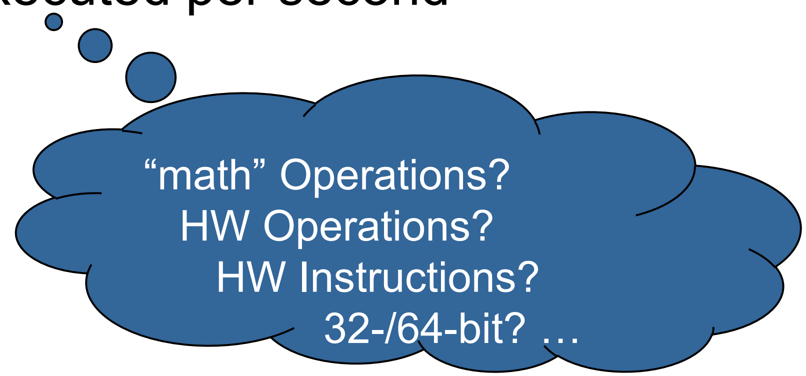
> "If you optimize everything, you will always be unhappy."
>
> Donald E. Knuth

Metrics of performance

- What can be measured?
  - A **count** of how often an event occurs
    - E.g., the number of MPI point-to-point messages sent
  - The **duration** of some interval
    - E.g., the time spent these send calls
  - The **size** of some parameter
    - E.g., the number of bytes transmitted by these calls
- Derived metrics
  - E.g., rates / throughput
  - Needed for normalization

Example metrics

- Execution time
- Number of function calls
- CPI
  - CPU cycles per instruction
- FLOPS
  - Floating-point operations executed per second

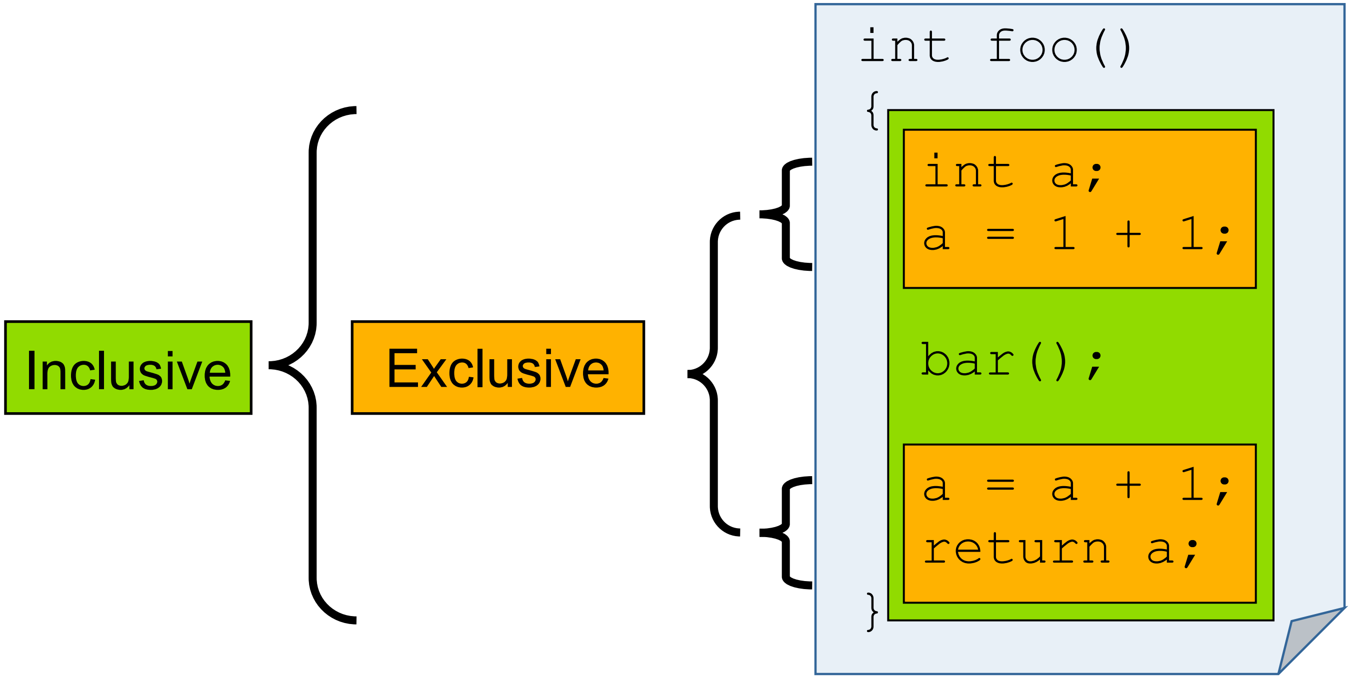


Execution time

- Wall-clock time
  - Includes waiting time: I/O, memory, other system activities
  - In time-sharing environments also the time consumed by other applications
- CPU time
  - Time spent by the CPU to execute the application
  - Does not include time the program was context-switched out
    - Problem: Does not include inherent waiting time (e.g., I/O)
    - Problem: Portability? What is user, what is system time?
- Problem: Execution time is non-deterministic
  - Use mean or minimum of several runs

Inclusive vs. Exclusive values

- Inclusive
  - Information of all sub-elements aggregated into single value
- Exclusive
  - Information cannot be subdivided further



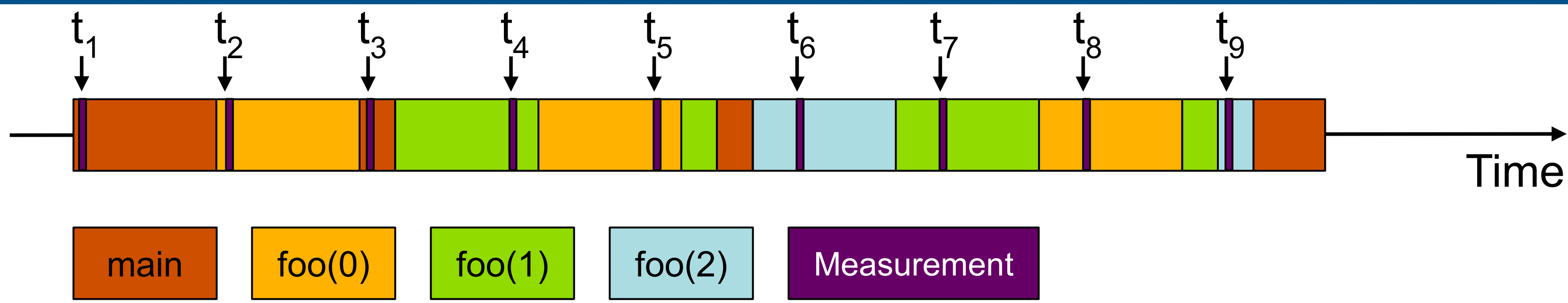
Classification of measurement techniques

- How are performance measurements triggered?
  - Sampling
  - Code instrumentation

- How is performance data recorded?
  - Profiling / Runtime summarization
  - Tracing

- How is performance data analyzed?
  - Online
  - Post mortem

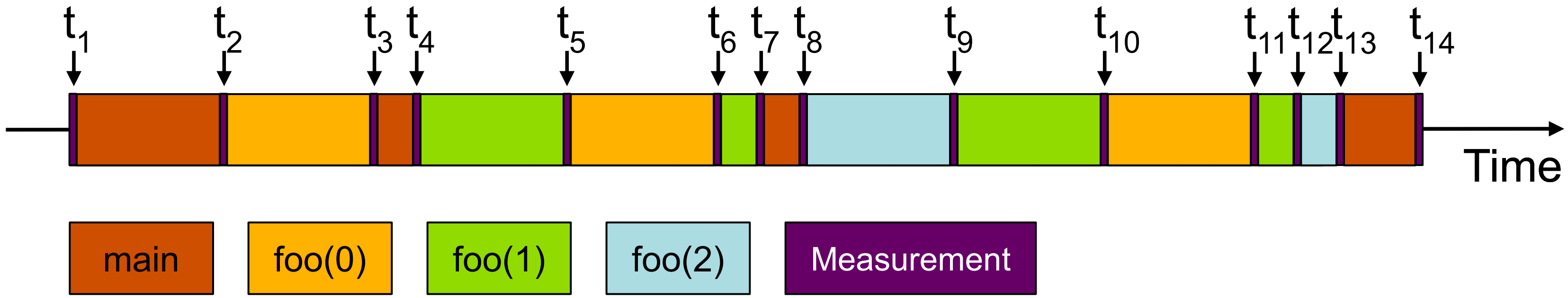
Sampling




```
int main()
{
  int i;

  for (i=0; i < 3; i++)
    foo(i);

  return 0;
}

void foo(int i)
{

  if (i > 0)
    foo(i - 1);

}
```

- Running program is periodically interrupted to take measurement
  - Timer interrupt, OS signal, or HWC overflow
  - Service routine examines return-address stack
  - Addresses are mapped to routines using symbol table information
- **Statistical** inference of program behavior
  - Not very detailed information on highly volatile metrics
  - Requires long-running applications
- Works with unmodified executables

Instrumentation




```
int main()
{
  int i;
  Enter("main");
  for (i=0; i < 3; i++)
    foo(i);
  Leave("main");
  return 0;
}

void foo(int i)
{
  Enter("foo");
  if (i > 0)
    foo(i - 1);
  Leave("foo");
}
```

- Measurement code is inserted such that every event of interest is captured **directly**
  - Can be done in various ways
- Advantage:
  - Much more detailed information
- Disadvantage:
  - Processing of source-code / executable necessary
  - Large relative overheads for small functions

Instrumentation techniques

- Static instrumentation
  - Program is instrumented prior to execution
- Dynamic instrumentation
  - Program is instrumented at runtime

- Code is inserted
  - Manually
  - Automatically
    - By a preprocessor / source-to-source translation tool
    - By a compiler
    - By linking against a pre-instrumented library / runtime system
    - By binary-rewrite / dynamic instrumentation tool

Critical issues

- Accuracy
  - Intrusion overhead
    - Measurement itself needs time and thus lowers performance
  - Perturbation
    - Measurement alters program behaviour
    - E.g., memory access pattern
  - Accuracy of timers & counters
- Granularity
  - How many measurements?
  - How much information / processing during each measurement?

☞ *Tradeoff: Accuracy vs. Expressiveness of data*

Classification of measurement techniques

- How are performance measurements triggered?
  - Sampling
  - Code instrumentation

- How is performance data recorded?
  - Profiling / Runtime summarization
  - Tracing

- How is performance data analyzed?
  - Online
  - Post mortem

Profiling / Runtime summarization

- Recording of aggregated information
  - Total, maximum, minimum, …
- For measurements
  - Time
  - Counts
    - Function calls
    - Bytes transferred
    - Hardware counters
- Over program and system entities
  - Functions, call sites, basic blocks, loops, …
  - Processes, threads

☞ *Profile = summarization of events over execution interval*

Types of profiles

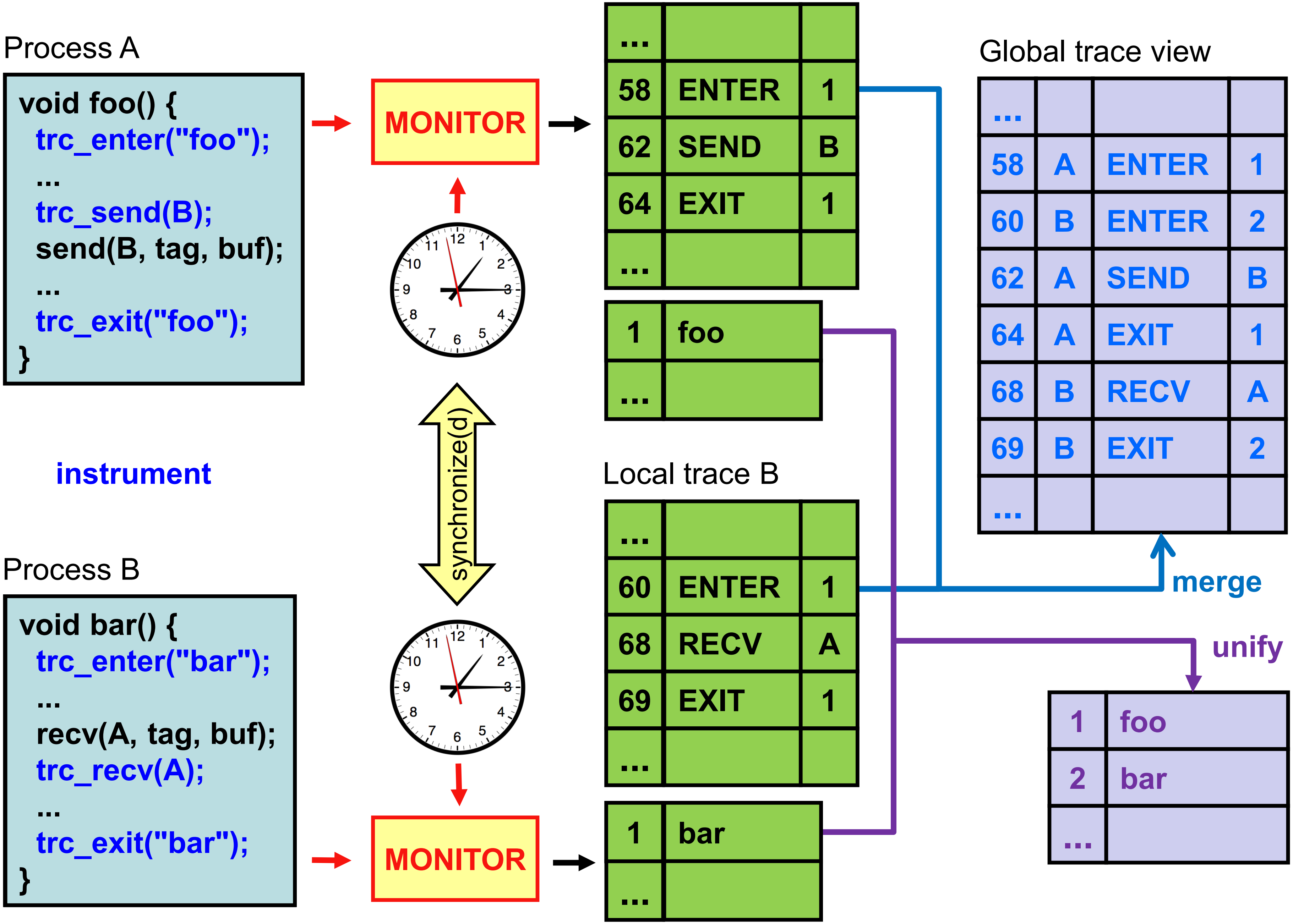
- Flat profile
  - Shows distribution of metrics per routine / instrumented region
  - Calling context is not taken into account
- Call-path profile
  - Shows distribution of metrics per executed call path
  - Sometimes only distinguished by partial calling context (e.g., two levels)
- Special-purpose profiles
  - Focus on specific aspects, e.g., MPI calls or OpenMP constructs
  - Comparing processes/threads

Tracing

- Recording information about significant points (events) during execution of the program
  - Enter / leave of a region (function, loop, …)
  - Send / receive a message, …
- Save information in event record
  - Timestamp, location, event type
  - Plus event-specific information (e.g., communicator, sender / receiver, …)
- Abstract execution model on level of defined events

☞ *Event trace = Chronologically ordered sequence of event records*

Event tracing



Tracing vs. Profiling

- Tracing advantages
  - Event traces preserve the **temporal** and **spatial** relationships among individual events (☞ context)
  - Allows reconstruction of **dynamic** application behaviour on any required level of abstraction
  - Most general measurement technique
    - Profile data can be reconstructed from event traces
- Disadvantages
  - Traces can very quickly become extremely large
  - Writing events to file at runtime causes perturbation
  - Writing tracing software is complicated
    - Event buffering, clock synchronization, ...

Classification of measurement techniques

- How are performance measurements triggered?
  - Sampling
  - Code instrumentation

- How is performance data recorded?
  - Profiling / Runtime summarization
  - Tracing

- How is performance data analyzed?
  - Online
  - Post mortem

Online analysis

- Performance data is processed during measurement run
  - Process-local profile aggregation
  - More sophisticated inter-process analysis using
    - “Piggyback” messages
    - Hierarchical network of analysis agents
- Inter-process analysis often involves application steering to interrupt and re-configure the measurement

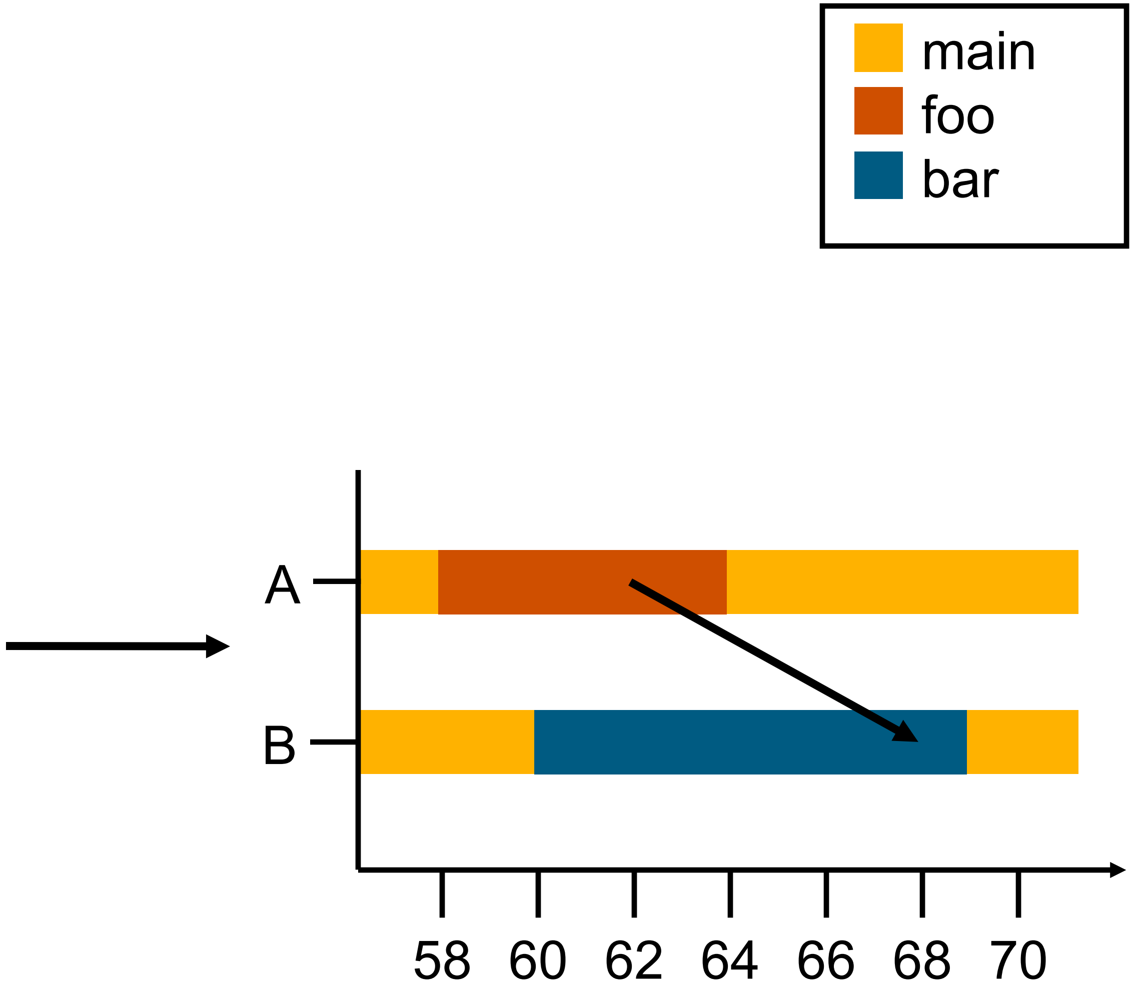
Post-mortem analysis

- Performance data is stored at end of measurement run
- Data analysis is performed afterwards
  - Automatic search for bottlenecks
  - Visual trace analysis
  - Calculation of statistics

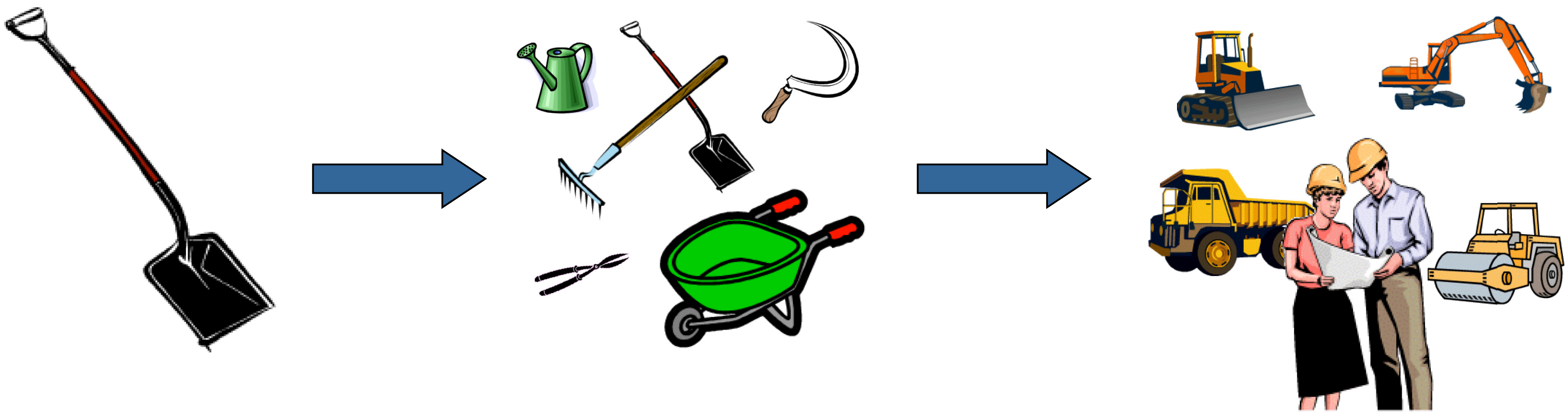
Example: Time-line visualization

1	foo
2	bar
3	...

...			
58	A	ENTER	1
60	B	ENTER	2
62	A	SEND	B
64	A	EXIT	1
68	B	RECV	A
69	B	EXIT	2
...			



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, ...

Typical performance analysis procedure

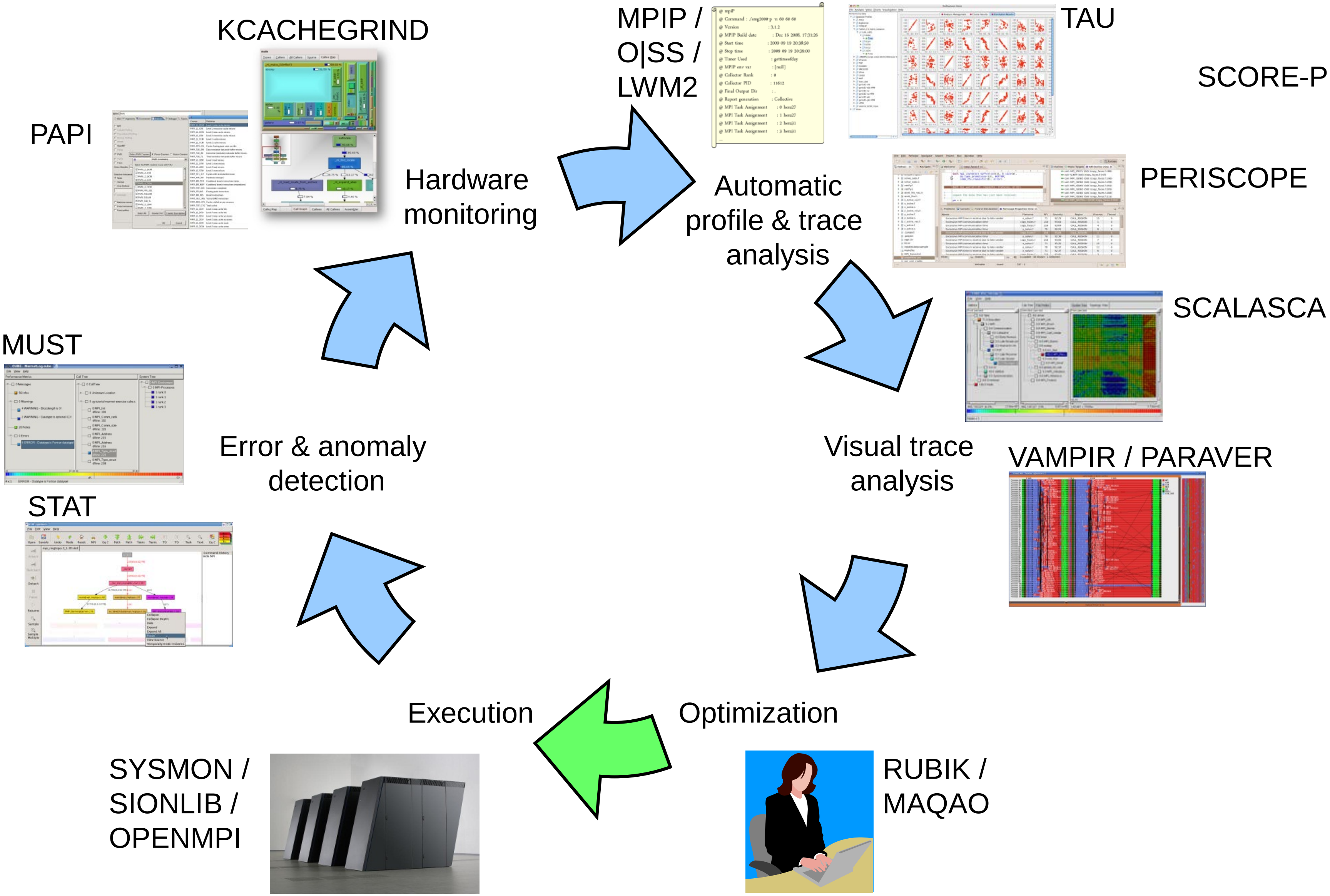
- Do I have a performance problem at all?
  - Time / speedup / scalability measurements
- What is the key bottleneck (computation / communication)?
  - MPI / OpenMP / flat profiling
- Where is the key bottleneck?
  - Call-path profiling, detailed basic block profiling
- Why is it there?
  - Hardware counter analysis, trace selected parts to keep trace size manageable
- Does the code have scalability problems?
  - Load imbalance analysis, compare profiles at various sizes function-by-function

VI-HPS productivity tools suite

Brian Wylie
Jülich Supercomputing Centre

Martin Schulz
Lawrence Livermore National Laboratory

VI-HPS technologies and their integration



- system/batchqueue monitoring (PTP/**SysMon**)
- lightweight execution monitoring/screening (**LWM2**)
- portable performance counter access (**PAPI**)
- MPI library profiling (**mpiP**)
- MPI execution outlier detection (**AutomaDeD**)
- MPI memory usage checking (**memchecker**)
- MPI correctness checking (**MUST**)
- lightweight stack trace analysis (**STAT**)
- task dependency debugging (**Temanejo**)

Integrated appl. execution profile & trace analysis

- instrumentation & measurement (**Score-P**, Extrae)
- profile analysis examination (**CUBE**, **ParaProf**)
- execution trace exploration (**Vampir**, **Paraver**)
- automated trace analysis (**Scalasca**)
- on-line automated analysis (**Periscope**)

- parallel performance frameworks (**O|SS**, **TAU**)
- performance analysis data-mining (**PerfExplorer**)
- parallel execution parametric studies (**Dimemas**)
- cache usage analysis (**kcachegrind**)
- assembly code optimization (**MAQAO**)
- process mapping generation/optimization (**Rubik**)

- parallel file I/O optimization (**SIONlib**)
- PMPI tools virtualization (P^NMPI)
- component-based tools framework (**CBTF**)

UNITE

- Uniform integrated tool environment
  - Manages installation & access to program development tools
    - based on software environment management “modules”
    - commonly used on most cluster and HPC systems
    - configurable for multiple MPI libraries & compiler suites
  - Specifies how & where tools packages get installed
    - including integrating tools where possible
  - Defines standard module names and different versions
  - Supplies pre-defined module files
  - Configurable to co-exist with local installations & policies
- Developed by JSC, RWTH & TUD
  - Available as open-source from
    http://www.vi-hps.org/projects/unite/

UNITE module setup

- First activate the UNITE modules environment

```
% module load UNITE
UNITE loaded
```

- then check modules available for tools and utilities (in various versions and variants)

```
% module avail
------ /usr/local/UNITE/modulefiles/tools ------
must/1.2.0-openmpi-gnu
periscope/1.5-openmpi-gnu
scalasca/1.4.3-openmpi-gnu(default)
scalasca/1.4.3-openmpi-intel
scalasca/2.0-openmpi-gnu
scorep/1.2-openmpi-gnu(default)
scorep/1.2-openmpi-intel
tau/2.19-openmpi-gnu
vampir/8.1
------ /usr/local/UNITE/modulefiles/utils ------
cube/3.4.3-gnu         papi/5.1.0-gnu      sionlib/1.3p7-openmp-gnu
```

UNITE module help

- then load the desired module(s)

```
% module load scalasca/1.4.3-openmpi-gnu-papi
cube/3.4.2-gnu loaded
scalasca/1.4.3-openmpi-gnu-papi loaded
```

- and/or read the associated module help information

```
% module help scalasca
Module specific help for
/usr/local/UNITE/modulefiles/tools/scalasca/1.4.3-openmpi-gnu-papi

Scalasca: Scalable performance analysis toolset
version 1.4.3 (for OpenMPI, Intel compiler, PAPI)

Basic usage:
1.Instrument application with "scalasca –instrument"
2.Collect & analyze execution measurement with "scalasca –analyze"
3.Examine analysis report with "scalasca –examine"

For more information
-See ${SCALASCA_ROOT}/doc/manuals/QuickReference.pdf
-http://www.scalasca.org
-mailto:scalasca@fz-juelich.de
```

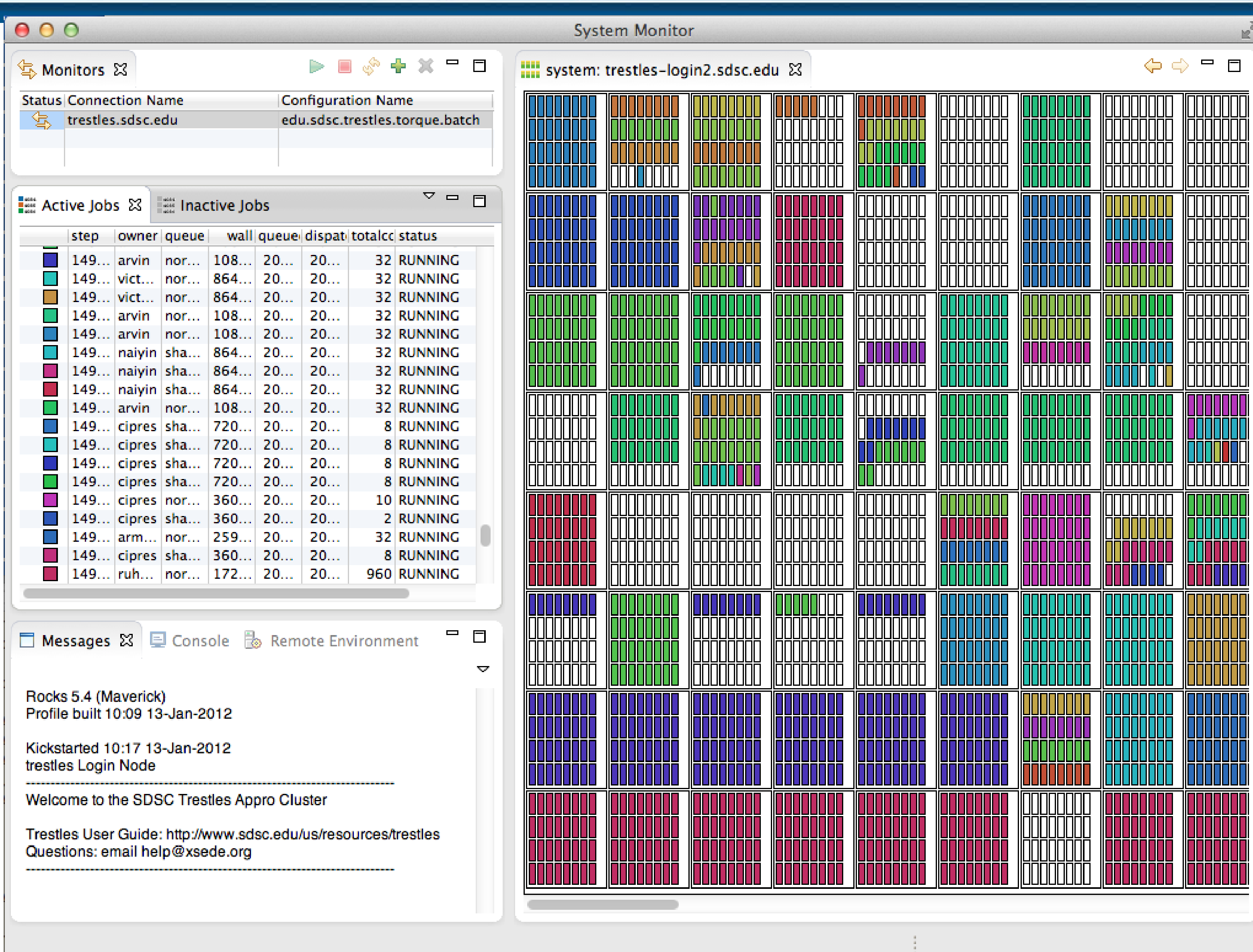
Application execution monitoring, checking & debugging

Execution monitoring, checking & debugging

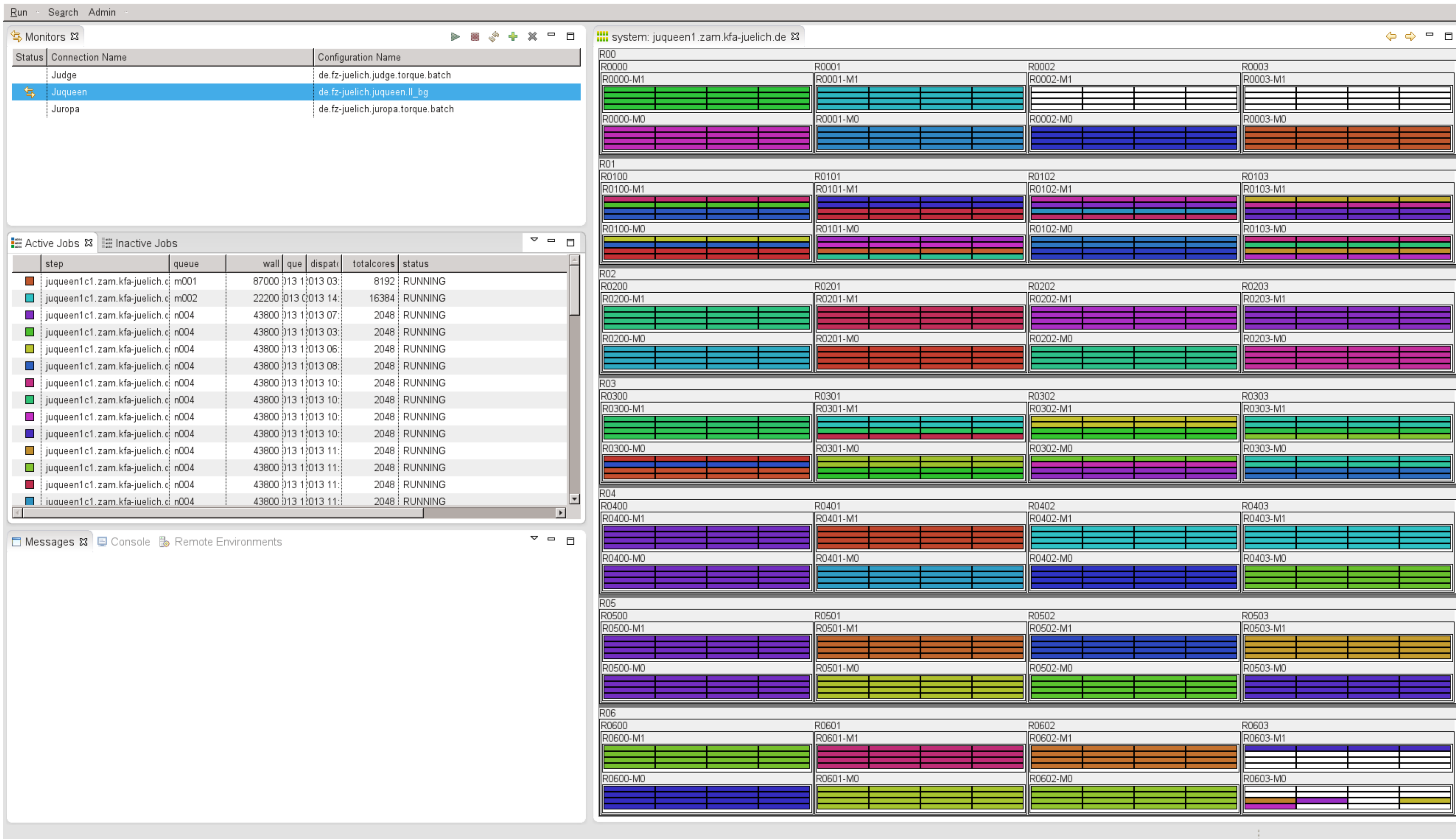
- system/batchqueue monitoring (PTP/**SysMon**)
- lightweight execution monitoring/screening (**LWM2**)
- portable performance counter access (**PAPI**)
- MPI library profiling (**mpiP**)
- MPI execution outlier detection (**AutomaDeD**)
- MPI memory usage checking (**memchecker**)
- MPI correctness checking (**MUST**)
- lightweight stack trace analysis (**STAT**)
- task dependency debugging (**Temanejo**)

- System monitor
  - Stand-alone or Eclipse/PTP plug-in
  - Displays current status of (super)computer systems
    - System architecture, compute nodes, attached devices (GPUs)
    - Jobs queued and allocated
  - Simple GUI interface for job creation and submission
    - Uniform interface to LoadLeveler, LSF, PBS, SLURM, Torque
    - Authentication/communication via SSH to remote systems
- Developed by JSC and contributed to Eclipse/PTP
  - Documentation and download from http://wiki.eclipse.org/PTP/System_Monitoring_FAQ
  - Supports Linux, Mac, Windows (with Java)

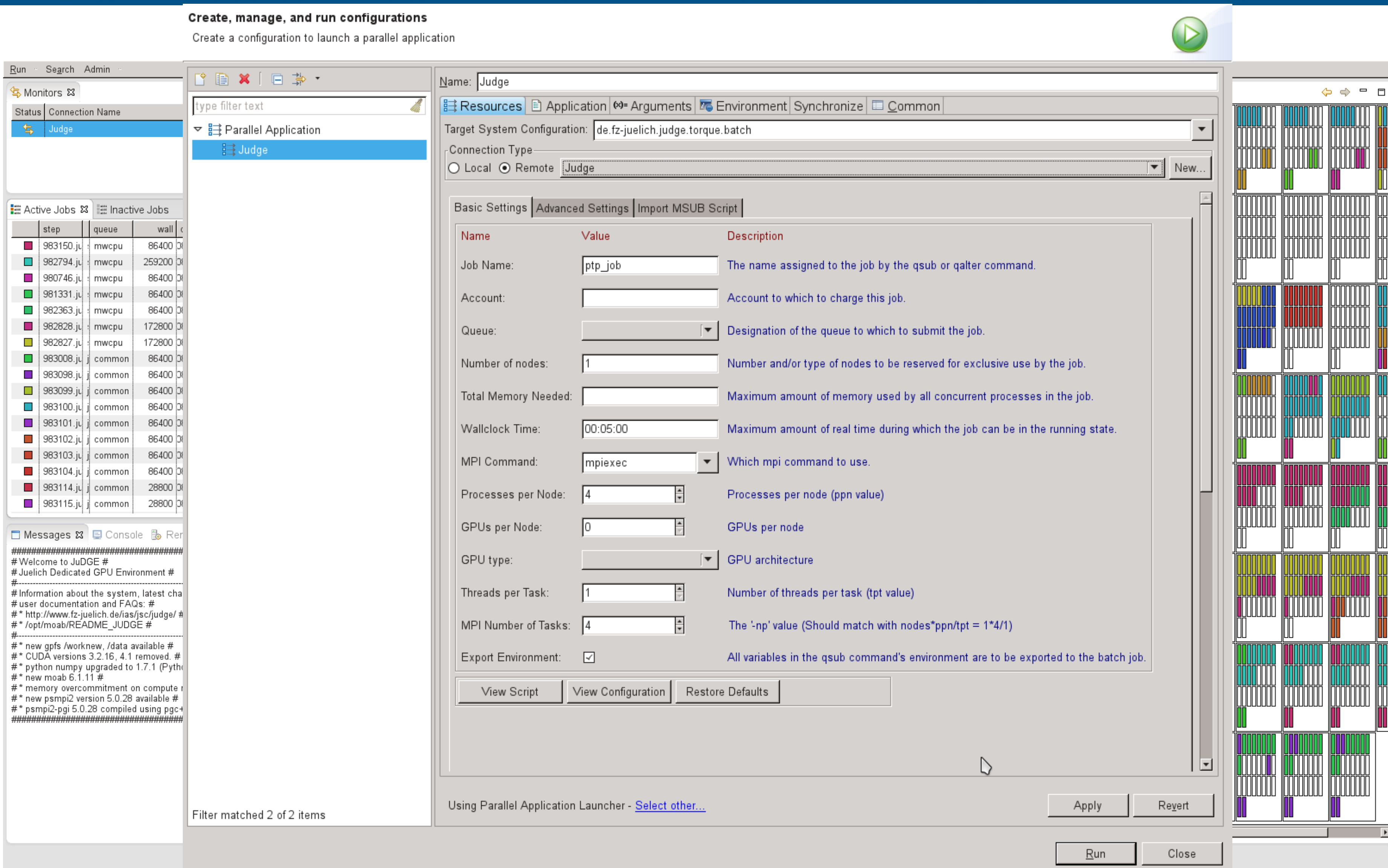
SysMon status display (Trestles@SDSC)



SysMon status display (Juqueen BG/Q)



Parallel execution launch configuration



- Light-Weight Monitoring Module
  - Provides basic application performance feedback
    - Profiles MPI, pthread-based multithreading (including OpenMP), CUDA & POSIX file I/O events
    - CPU and/or memory/cache utilization via PAPI hardware counters
  - Only requires preloading of LWM2 library
    - No recompilation/relinking of dynamically-linked executables
  - Less than 1% overhead suitable for initial performance screening
  - System-wide profiling requires a central performance database, and uses a web-based analysis front-end
    - Can identify inter-application interference for shared resources
- Developed by GRS Aachen
  - Supports x86 Linux
  - Available from http://www.vi-hps.org/projects/hopsa/tools/

LWM2 job digest report

```
**********************************************************************************
                                   Job Digest
**********************************************************************************
                          Job id:   0
            Wall clock time [s]:   35.28
               Nr. of Processes:   16
            Sampling rate [Hz]:   100
----------------------------------------------------------------------------------
                     Time spent:      Average          Minimum            Maximum
----------------------------------------------------------------------------------
              Time spent in MPI:         26.74%           4.07%              42.76%
          Time spent in MPI P2P:          0.15%           0.06%               0.31%
         Time spent in MPI Coll:          0.03%           0.00%               0.09%
          Time spent in MPI I/O:          0.00%           0.00%               0.00%
        Time spent in POSIX I/O:          0.00%           0.00%               0.00%
----------------------------------------------------------------------------------
              MPI Communication:      Average          Minimum            Maximum
----------------------------------------------------------------------------------
      Size of P2P messages [Bytes]:     110052.22         27040             162240
Size of collective messages sent [Bytes]:   11.19             0                 40
Size of collective messages recv [Bytes]:    3.81             0                 12
       P2P message frequency [/s]:        274.14         274.08             274.26
Collective invocation frequency [/s]:       0.23           0.23               0.23
         P2P bytes transfer rate [/s]:  20743502048.78  9664145454.55   53162496000.00
        Coll bytes transfer rate [/s]:     12000.00        4000.00           12000.00
----------------------------------------------------------------------------------
      Multithreading performance:      Average          Minimum            Maximum
----------------------------------------------------------------------------------
            OMP effective threads:          1.00           1.00               1.00
                Max. thread count:          1              1                  1
----------------------------------------------------------------------------------
        Sequential performance:      Average          Minimum            Maximum
----------------------------------------------------------------------------------
                             CPI:           0.77           0.57               1.05
                           FLOPS:   9507180128.38  9482099301.00      9562715666.00
                   FP Operations:          28.57%         21.25%             39.30%
----------------------------------------------------------------------------------
```

- Portable performance counter library & utilities
  - Configures and accesses hardware/system counters
  - Predefined events derived from available native counters
  - Core component for CPU/processor counters
    - instructions, floating point operations, branches predicted/taken, cache accesses/misses, TLB misses, cycles, stall cycles, …
    - performs transparent multiplexing when required
  - Extensible components for off-processor counters
    - InfiniBand network, Lustre filesystem, system hardware health, …
  - Used by multi-platform performance measurement tools
    - Score-P, Periscope, Scalasca, TAU, LWM2, Open|SpeedShop, ...
- Developed by UTK-ICL
  - Available as open-source for most modern processors
    http://icl.cs.utk.edu/papi/

PAPI preset counters (and their definitions)

- juropa$ ***papi_avail***
- Available events and hardware information.

```
--------------------------------------------
PAPI Version             : 4.1.0.0
Vendor string and code   : GenuineIntel (1)
Model string and code    : Intel(R) Xeon(R) CPU
X5570 @ 2.93GHz (26)
CPU Revision             : 5.000000
CPUID Info               : Family: 6  Model: 26
Stepping: 5
CPU Megahertz            : 1600.000000
CPU Clock Megahertz      : 1600
Hdw Threads per core     : 2
Cores per Socket         : 4
NUMA Nodes               : 2
CPU's per Node           : 8
Total CPU's              : 16
Number Hardware Counters : 16
Max Multiplex Counters   : 512
--------------------------------------------
   Name        Code    Avail Deriv Description
```

- **PAPI_L1_DCM** 0x80000000 Yes No
  Level 1 data cache misses
- **PAPI_L1_ICM** 0x80000001 Yes No
  Level 1 instruction cache misses

  ...

- Of 107 possible events, 35 are available, of which 9 are derived.

- juropa$ ***papi_avail -d***
- ...
  Symbol   Event Code Count |Short Descr.|
  |Long Description|
  |Developer's Notes|
  |Derived|
  |PostFix|
  Native Code[n]: <hex> |name|
- **PAPI_L1_DCM** 0x80000000 1 |L1D cache misses|
  |Level 1 data cache misses|
  ||
  |NOT_DERIVED|
  ||
  Native Code[0]: 0x40002028 |*L1D:REPL*|
- **PAPI_L1_ICM** 0x80000001 1 |L1I cache misses|
  |Level 1 instruction cache misses|
  ||
  |NOT_DERIVED|
  ||
  Native Code[0]: 0x40001031 |*L1I:MISSES*|
- **PAPI_L2_DCM** 0x80000002 2 |L2D cache misses|
  |Level 2 data cache misses|
  ||
  |*DERIVED_SUB*|
  ||
  Native Code[0]: 0x40000437 |*L2_RQSTS:MISS*|
  Native Code[1]: 0x40002037 |*L2_RQSTS:IFETCH_MISS*|
- ...

PAPI native counters (and qualifiers)

- juropa$ ***papi_native_avail***
- Available native events and hardware information.
- ...
- Event Code   Symbol | Long Description |

```
--------------------------------------------------------------------------------
0x40000000   UNHALTED_CORE_CYCLES  | count core clock cycles whenever the cloc |
             | k signal on the specific core is running (not halted). Alias to e |
             | vent CPU_CLK_UNHALTED:THREAD                                       |
--------------------------------------------------------------------------------
0x40000001   INSTRUCTION_RETIRED  | count the number of instructions at retire |
             | ment. Alias to event INST_RETIRED:ANY_P                           |
--------------------------------------------------------------------------------
...
```

-
```
--------------------------------------------------------------------------------
0x40000086   UNC_SNP_RESP_TO_REMOTE_HOME  | Remote home snoop response - LLC d |
             | oes not have cache line                                 |
  40000486   :I_STATE  | Remote home snoop response - LLC does not have cache  |
             | line                                               |
  40000886   :S_STATE  | Remote home snoop response - LLC has  cache line in S |
             |  state                                              |
  40001086   :FWD_S_STATE  | Remote home snoop response - LLC forwarding cache |
             |  line in S state.                                     |
  40002086   :FWD_I_STATE  | Remote home snoop response - LLC has forwarded a  |
             | modified cache line                                    |
  40004086   :CONFLICT  | Remote home conflict snoop response                 |
  40008086   :WB  | Remote home snoop response - LLC has cache line in the M s |
             | tate                                               |
  40010086   :HITM  | Remote home snoop response - LLC HITM                   |
--------------------------------------------------------------------------------
Total events reported: 135
```

PAPI counter combinations

- juropa$ ***papi_event_chooser PRESET \ PAPI_FP_OPS PAPI_DP_OPS***
- Event Chooser: Available events which can be added with given events.

  …

  Name Code Deriv Description (Note)
- **PAPI_TOT_INS** 0x80000032 No
  Instructions completed
- **PAPI_FP_INS** 0x80000034 No
  Floating point instructions
- **PAPI_TOT_CYC** 0x8000003b No
  Total cycles
- **PAPI_VEC_SP** 0x80000069 No
  Single precision vector/SIMD instructions
- **PAPI_VEC_DP** 0x8000006a No
  Double precision vector/SIMD instructions

- Total events reported: 5.

- juropa$ ***papi_command_line \ PAPI_FP_OPS PAPI_DP_OPS PAPI_L1_DCM***

- Successfully added PAPI_FP_OPS
- Successfully added PAPI_DP_OPS
- Failed adding: PAPI_L1_DCM
- because: **PAPI_ECNFLCT**

- PAPI_FP_OPS : 42142167
- PAPI_DP_OPS : 42142167
- PAPI_L1_DCM : **---------**

  Verification: Checks for valid event name.
- This utility lets you add events from the command line interface to see if they work.

- Lightweight MPI profiling
  - only uses PMPI standard profiling interface
    - static (re-)link or dynamic library preload
  - accumulates statistical measurements for MPI library routines used by each process
  - merged into a single textual output report
  - MPIP environment variable for advanced profiling control
    - stack trace depth, reduced output, etc.
  - MPI_Pcontrol API for additional control from within application
  - optional separate mpiPview GUI
- Developed by LLNL & ORNL
  - BSD open-source license
  - http://mpip.sourceforge.net/

Scenarios:

- New application development
- Analyze/Optimize external application
- Suspected bottlenecks

First goal: overview of …

- Communication frequency and intensity
- Types and complexity of communication
- Source code locations of expensive MPI calls
- Differences between processes

Basic Principle of Profiling MPI

Intercept all MPI API calls

- Using wrappers for all MPI calls

Aggregate statistics over time

- Number of invocations
- Data volume
- Time spent during function execution

Multiple aggregations options/granularity

- By function name or type
- By source code location (call stack)
- By process rank

mpiP: Efficient MPI Profiling

Open source MPI profiling library

- Developed at LLNL, maintained by LLNL & ORNL
- Available from sourceforge
- Works with any MPI library

Easy-to-use and portable design

- Relies on PMPI instrumentation
- No additional tool daemons or support infrastructure
- Single text file as output
- Optional: GUI viewer

mpiP works on binary files

- Uses standard development chain
- Use of "-g" recommended

Run option 1: Relink

- Specify libmpi.a/.so on the link line
- Portable solution, but requires object files

Run option 2: library preload

- Set preload variable (e.g., LD_PRELOAD) to mpiP
- Transparent, but only on supported systems

Running with mpiP 101 / Running

```
bash-3.2$ srun –n4 smg2000
mpiP:
mpiP:
mpiP: mpiP V3.1.2 (Build Dec 16 2008/17:31:26)
mpiP: Direct questions and errors to mpip-
help@lists.sourceforge.net
mpiP:
Running with these driver parameters:
  (nx, ny, nz)    = (60, 60, 60)
  (Px, Py, Pz)    = (4, 1, 1)
  (bx, by, bz)    = (1, 1, 1)
  (cx, cy, cz)    = (1.000000, 1.000000, 1.000000)
  (n_pre, n_post) = (1, 1)
  dim             = 3
  solver ID       = 0
=============================================
Struct Interface:
=============================================
Struct Interface:
  wall clock time = 0.075800 seconds
  cpu clock time  = 0.080000 seconds
```

Header

```
=============================================
Setup phase times:
=============================================
SMG Setup:
  wall clock time = 1.473074 seconds
  cpu clock time  = 1.470000 seconds
=============================================
Solve phase times:
=============================================
SMG Solve:
  wall clock time = 8.176930 seconds
  cpu clock time  = 8.180000 seconds

Iterations = 7
Final Relative Residual Norm = 1.459319e-07

mpiP:
mpiP: Storing mpiP output in [./smg2000-p.4.11612.1.mpiP].
mpiP:
bash-3.2$
```

Output File

mpiP 101 / Output – Metadata

```
@ mpiP
@ Command : ./smg2000-p -n 60 60 60
@ Version                  : 3.1.2
@ MPIP Build date          : Dec 16 2008, 17:31:26
@ Start time               : 2009 09 19 20:38:50
@ Stop time                : 2009 09 19 20:39:00
@ Timer Used               : gettimeofday
@ MPIP env var             : [null]
@ Collector Rank           : 0
@ Collector PID            : 11612
@ Final Output Dir         : .
@ Report generation        : Collective
@ MPI Task Assignment      : 0 hera27
@ MPI Task Assignment      : 1 hera27
@ MPI Task Assignment      : 2 hera31
@ MPI Task Assignment      : 3 hera31
```

mpiP 101 / Output – Overview

```
---------------------------------------------------------
@--- MPI Time (seconds) ----------------------------------
---------------------------------------------------------
Task    AppTime    MPITime     MPI%
   0       9.78       1.97    20.12
   1        9.8       1.95    19.93
   2        9.8       1.87    19.12
   3       9.77       2.15    21.99
   *       39.1       7.94    20.29
---------------------------------------------------------
```

mpiP 101 / Output – Callsites

```
---------------------------------------------------------------------------
@--- Callsites: 23 --------------------------------------------------------
---------------------------------------------------------------------------
 ID Lev File/Address        Line Parent_Funct                    MPI_Call
  1   0 communication.c     1405 hypre_CommPkgUnCommit           Type_free
  2   0 timing.c             419 hypre_PrintTiming               Allreduce
  3   0 communication.c      492 hypre_InitializeCommunication   Isend
  4   0 struct_innerprod.c   107 hypre_StructInnerProd           Allreduce
  5   0 timing.c             421 hypre_PrintTiming               Allreduce
  6   0 coarsen.c            542 hypre_StructCoarsen             Waitall
  7   0 coarsen.c            534 hypre_StructCoarsen             Isend
  8   0 communication.c     1552 hypre_CommTypeEntryBuildMPI     Type_free
  9   0 communication.c     1491 hypre_CommTypeBuildMPI          Type_free
 10   0 communication.c      667 hypre_FinalizeCommunication     Waitall
 11   0 smg2000.c            231 main                            Barrier
 12   0 coarsen.c            491 hypre_StructCoarsen             Waitall
 13   0 coarsen.c            551 hypre_StructCoarsen             Waitall
 14   0 coarsen.c            509 hypre_StructCoarsen             Irecv
 15   0 communication.c     1561 hypre_CommTypeEntryBuildMPI     Type_free
 16   0 struct_grid.c        366 hypre_GatherAllBoxes            Allgather
 17   0 communication.c     1487 hypre_CommTypeBuildMPI          Type_commit
 18   0 coarsen.c            497 hypre_StructCoarsen             Waitall
 19   0 coarsen.c            469 hypre_StructCoarsen             Irecv
 20   0 communication.c     1413 hypre_CommPkgUnCommit           Type_free
 21   0 coarsen.c            483 hypre_StructCoarsen             Isend
 22   0 struct_grid.c        395 hypre_GatherAllBoxes            Allgatherv
 23   0 communication.c      485 hypre_InitializeCommunication   Irecv
---------------------------------------------------------------------------
```

mpiP 101 / Output – per Function Timing

```
---------------------------------------------------------------
@--- Aggregate Time (top twenty, descending, milliseconds) ---
---------------------------------------------------------------
Call                 Site       Time    App%    MPI%     COV
Waitall                10    4.4e+03   11.24   55.40    0.32
Isend                   3   1.69e+03    4.31   21.24    0.34
Irecv                  23        980    2.50   12.34    0.36
Waitall                12        137    0.35    1.72    0.71
Type_commit            17        103    0.26    1.29    0.36
Type_free               9       99.4    0.25    1.25    0.36
Waitall                 6       81.7    0.21    1.03    0.70
Type_free              15       79.3    0.20    1.00    0.36
Type_free               1       67.9    0.17    0.85    0.35
Type_free              20       63.8    0.16    0.80    0.35
Isend                  21         57    0.15    0.72    0.20
Isend                   7       48.6    0.12    0.61    0.37
Type_free               8       29.3    0.07    0.37    0.37
Irecv                  19       27.8    0.07    0.35    0.32
Irecv                  14       25.8    0.07    0.32    0.34
...
```

mpiP 101 / Output – per Function Message Size

```
---------------------------------------------------------------------------
@--- Aggregate Sent Message Size (top twenty, descending, bytes) ------
---------------------------------------------------------------------------
Call          Site    Count     Total     Avrg  Sent%
Isend            3   260044    2.3e+08     885  99.63
Isend            7     9120   8.22e+05    90.1   0.36
Isend           21     9120   3.65e+04       4   0.02
Allreduce        4       36        288       8   0.00
Allgatherv      22        4        112      28   0.00
Allreduce        2       12         96       8   0.00
Allreduce        5       12         96       8   0.00
Allgather       16        4         16       4   0.00
```

mpiP 101 / Output – per Callsite Timing

```
---------------------------------------------------------------------------
@--- Callsite Time statistics (all, milliseconds): 92 ---------------------
---------------------------------------------------------------------------
Name          Site Rank  Count      Max      Mean       Min   App%   MPI%
Allgather       16    0      1    0.034     0.034     0.034   0.00   0.00
Allgather       16    1      1    0.049     0.049     0.049   0.00   0.00
Allgather       16    2      1     2.92      2.92      2.92   0.03   0.16
Allgather       16    3      1        3         3         3   0.03   0.14
Allgather       16    *      4        3       1.5     0.034   0.02   0.08

Allgatherv      22    0      1     0.03      0.03      0.03   0.00   0.00
Allgatherv      22    1      1    0.036     0.036     0.036   0.00   0.00
Allgatherv      22    2      1    0.022     0.022     0.022   0.00   0.00
Allgatherv      22    3      1    0.022     0.022     0.022   0.00   0.00
Allgatherv      22    *      4    0.036    0.0275     0.022   0.00   0.00

Allreduce        2    0      3    0.382     0.239     0.011   0.01   0.04
Allreduce        2    1      3     0.31     0.148     0.046   0.00   0.02
Allreduce        2    2      3    0.411     0.178     0.062   0.01   0.03
Allreduce        2    3      3     1.33     0.622     0.062   0.02   0.09
Allreduce        2    *     12     1.33     0.297     0.011   0.01   0.04

...
```

mpiP 101 / Output – per Callsite Message Size

@--- Callsite Message Sent statistics (all, sent bytes) -------------------

----

Name	Site	Rank	Count	Max	Mean	Min	Sum
Allgather	16	0	1	4	4	4	4
Allgather	16	1	1	4	4	4	4
Allgather	16	2	1	4	4	4	4
Allgather	16	3	1	4	4	4	4
Allgather	16	*	4	4	4	4	16
Allgatherv	22	0	1	28	28	28	28
Allgatherv	22	1	1	28	28	28	28
Allgatherv	22	2	1	28	28	28	28
Allgatherv	22	3	1	28	28	28	28
Allgatherv	22	*	4	28	28	28	112
Allreduce	2	0	3	8	8	8	24
Allreduce	2	1	3	8	8	8	24
Allreduce	2	2	3	8	8	8	24
Allreduce	2	3	3	8	8	8	24
Allreduce	2	*	12	8	8	8	96

...

mpiP Advanced Features

- User controlled stack trace depth
- Reduced output for large scale experiments
- Application control to limit scope
- Measurements for MPI I/O routines

Controlled by MPIP environment variable

- Set by user before profile run
- Command line style argument list
- Example: MPIP = "-c –o –k 4"

Param.	Description	Default
-c	Concise Output / No callsite data	
-f dir	Set output directory	
-k n	Set callsite stack traceback size to n	1
-l	Use less memory for data collection	
-n	Do not truncate pathnames	
-o	Disable profiling at startup	
-s n	Set hash table size	256
-t x	Print threshold	0.0
-v	Print concise & verbose output	

Controlling the Stack Trace

Callsites are determined using stack traces

- Starting from current call stack going backwards
- Useful to avoid MPI wrappers
- Helps to distinguishes library invocations

Tradeoff: stack trace depth

- Too short: can't distinguish invocations
- Too long: extra overhead / too many call sites

User can set stack trace depth

- -k <n> parameter

Challenges with mpiP at Scale

```
@ mpiP
@ Version: 3.1.1
// 10 lines of mpiP and experiment configuration options
// 8192 lines of task assignment to BlueGene topology information

@--- MPI Time (seconds) ---------------------------------------
Task    AppTime    MPITime     MPI%
   0       37.7       25.2    66.89
// ...
8191       37.6         26    69.21
   *   3.09e+05   2.04e+05    65.88

@--- Callsites: 26 -------------------------------------------
 ID Lev File/Address     Line Parent_Funct             MPI_Call
  1   0 coarsen.c         542 hypre_StructCoarsen      Waitall
// 25 similar lines

@--- Aggregate Time (top twenty, descending, milliseconds) --------
Call                 Site       Time    App%    MPI%     COV
Waitall                21   1.03e+08   33.27   50.49    0.11
Waitall                 1   2.88e+07    9.34   14.17    0.26
// 18 similar lines
```

Challenges with mpiP at Scale (cont.)

```
@--- Aggregate Sent Message Size (top twenty, descending, bytes) --
Call                     Site       Count        Total        Avrg  Sent%
Isend                      11   845594460   7.71e+11         912  59.92
Allreduce                  10       49152   3.93e+05           8   0.00
// 6 similar lines

@--- Callsite Time statistics (all, milliseconds): 212992 ---------
Name          Site Rank       Count     Max    Mean       Min    App%    MPI%
Waitall         21    0      111096     275     0.1 0.000707   29.61   44.27
//   ...
Waitall         21 8191       65799     882    0.24 0.000707   41.98   60.66
Waitall         21    *  577806664     882   0.178 0.000703   33.27   50.49
// 213,042 similar lines

@--- Callsite Message Sent statistics (all, sent bytes) -----------
Name          Site Rank       Count       Max       Mean    Min          Sum
Isend           11    0       72917 2.621e+05      851.1      8    6.206e+07
//...
Isend           11 8191       46651 2.621e+05       1029      8    4.801e+07
Isend           11    *  845594460 2.621e+05      911.5      8    7.708e+11
// 65,550 similar lines
```

Concise Output

Output file contains many details

- Users often only interested in summary
- Per callsite/task data harms scalability

Option to provide concise output

- Same basic format
- Omit collection of per callsite/task data

User controls output format through parameters

- -c = concise output only
- -v = provide concise and full output files

Limiting Scope

By default, mpiP measures entire execution

- Any event between MPI_Init and MPI_Finalize

Optional: controlling mpiP from within the application

- Disable data collection at startup (-o)
- Enable using MPI_Pcontrol(x)
  - x=0: Disable profiling
  - x=1: Enable profiling
  - x=2: Reset profile data
  - x=3: Generate full report
  - x=4: Generate concise report

Limiting Scope / Example

```
for(i=1; i < 10; i++)
{ switch(i)
  {
    case 5:
      MPI_Pcontrol(2);
      MPI_Pcontrol(1);
      break;
    case 6:
      MPI_Pcontrol(0);
      MPI_Pcontrol(4);
      break;
    default:
      break; }
  /* ... compute and communicate for one timestep ... */
}
```

Reset & Start in Iteration 5

Stop & Report in Iteration 6

mpiP Platforms

Highly portable design

- Built on top of PMPI, which is part of any MPI
- Very few dependencies

Tested on many platforms, including

- Linux (x86, x86-64, IA-64, MIPS64)
- BG/L & BG/P
- AIX (Power 3/4/5)
- Cray XT3/4/5 with Catamount and CNL
- Cray X1E

mpiP Installation

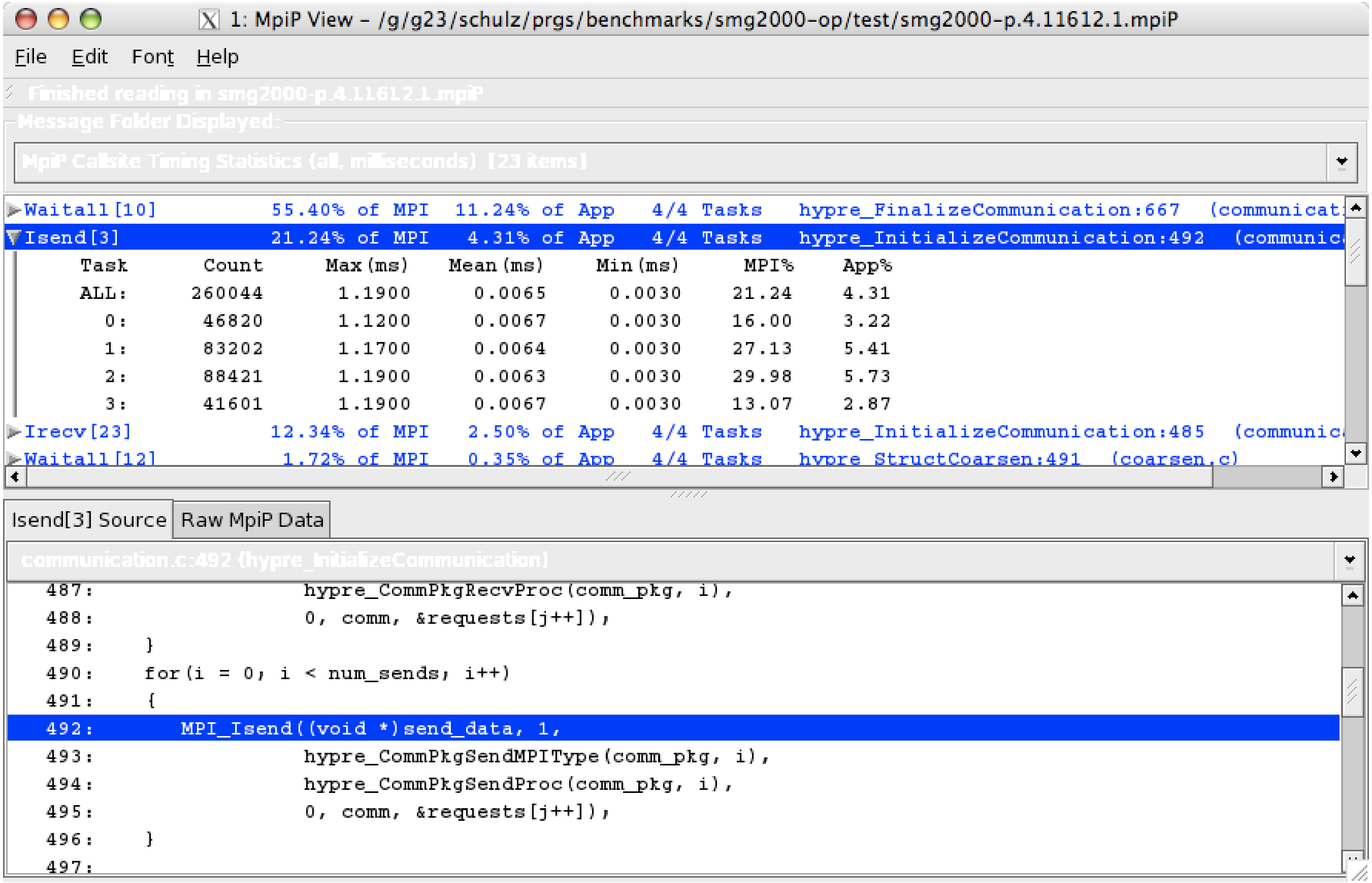
Download from http://sourceforge.net/projects/mpip
- Current release version: 3.1.2
- CVS access to development version

Autoconf-based build system with options to
- Disable I/O support
- Pick a timing option
- Choose name demangling scheme
- Build on top of the suitable stack tracer
- Set maximal stack trace depth

mpiPView: The GUI for mpiP

- Optional: displaying mpiP data in a GUI
- Implemented as part of the Tool Gear project
- Reads mpiP output file
- Provides connection to source files
- Usage process
- First: select input metrics
- Hierarchical view of all callsites
- Source panel once callsite is selected
- Ability to remap source directories

mpiPView GUI example



- Helps find memory errors in MPI applications
  - e.g, overwriting of memory regions used in non-blocking comms, use of uninitialized input buffers
  - intercepts memory allocation/free and checks reads and writes
- Part of Open MPI based on valgrind Memcheck
  - Need to be configured when installing Open MPI 1.3 or later, with valgrind 3.2.0 or later available
- Developed by HLRS
  - www.vi-hps.org/Tools/MemChecker.html

MUST

- Tool to check for correct MPI usage at runtime
  - Checks conformance to MPI standard
    - Supports Fortran & C bindings of MPI-2.2
  - Checks parameters passed to MPI
  - Monitors MPI resource usage
- Implementation
  - C++ library gets linked to the application
  - Does not require source code modifications
  - Additional process used as DebugServer
- Developed by RWTH Aachen, TU Dresden, LLNL & LANL
  - BSD license open-source initial release in November 2011 as successor to MARMOT
  - http://tu-dresden.de/zih/must/

MUST: Need for runtime error checking

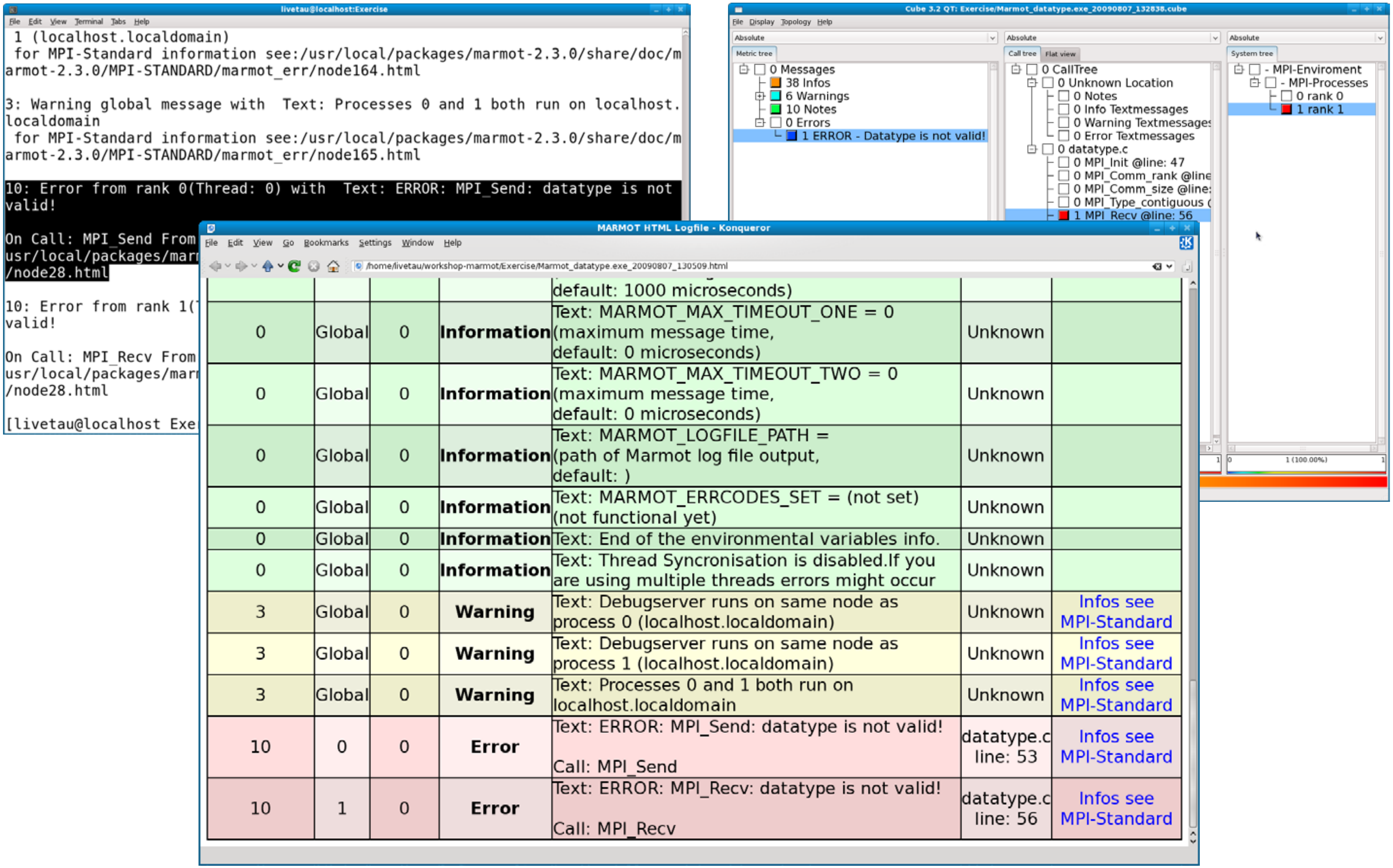
- Programming MPI is error-prone
- Interfaces often define requirements for function arguments
  - non-MPI Example: *memcpy* has undefined behaviour for overlapping memory regions
- MPI-2.2 Standard specification has 676 pages
  - Who remembers all requirements mentioned there?
- For performance reasons MPI libraries run no checks
- Runtime error checking pinpoints incorrect, inefficient & unsafe function calls

- Local checks:
  - Integer validation
  - Integrity checks (pointer validity, etc.)
  - Operation, Request, Communicator, Datatype & Group object usage
  - Resource leak detection
  - Memory overlap checks
- Non-local checks:
  - Collective verification
  - Lost message detection
  - Type matching (for P2P and collectives)
  - Deadlock detection (with root cause visualization)

MUST usage

- Compile and link application as usual
  - Static re-link with MUST compilers when required
- Execute replacing *mpiexec* with ***mustrun***
  - Extra DebugServer process started automatically
  - Ensure this extra resource is allocated in jobscript
- Add *--must:nocrash* if application doesn't crash to disable checks and improve execution performance
- View *MUST_Output.html* report in browser

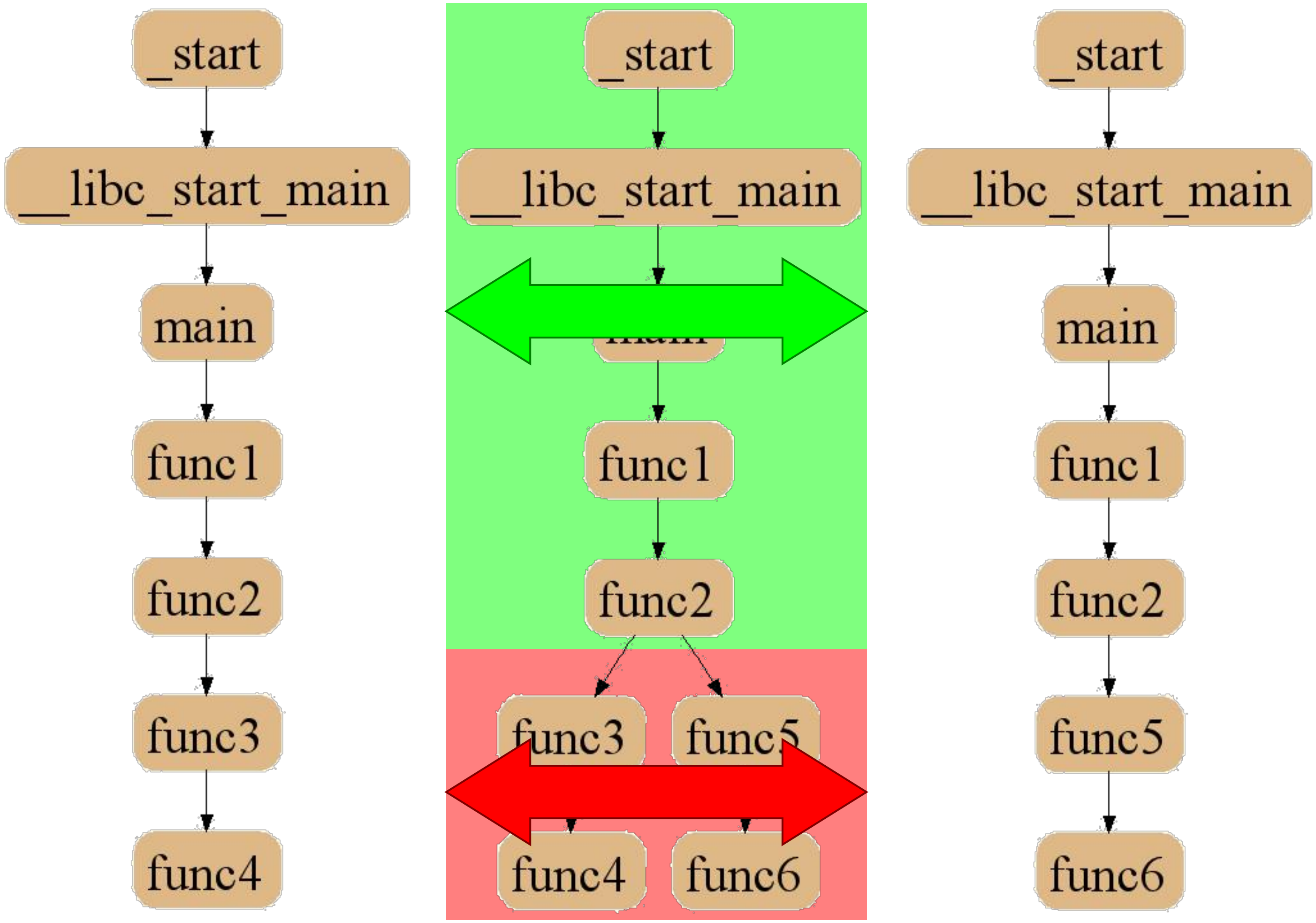
MUST/Marmot reports



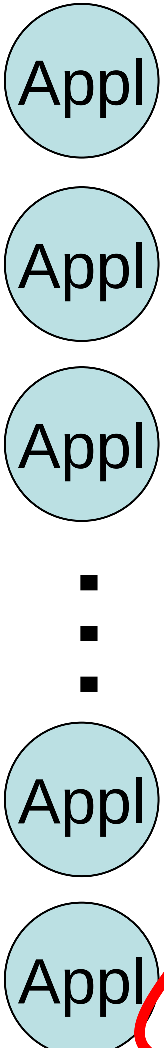
- Existing debuggers don't scale
  - Inherent limits in the approaches
  - Need for new, scalable methodologies
- Need to pre-analyze and reduce data
  - Fast tools to gather state
  - Help select nodes to run conventional debuggers on
- Scalable tool: STAT
  - Stack Trace Analysis Tool
  - Goal: Identify equivalence classes
  - Hierarchical and distributed aggregation of stack traces from all tasks
  - Stack trace merge <1s from 200K+ cores

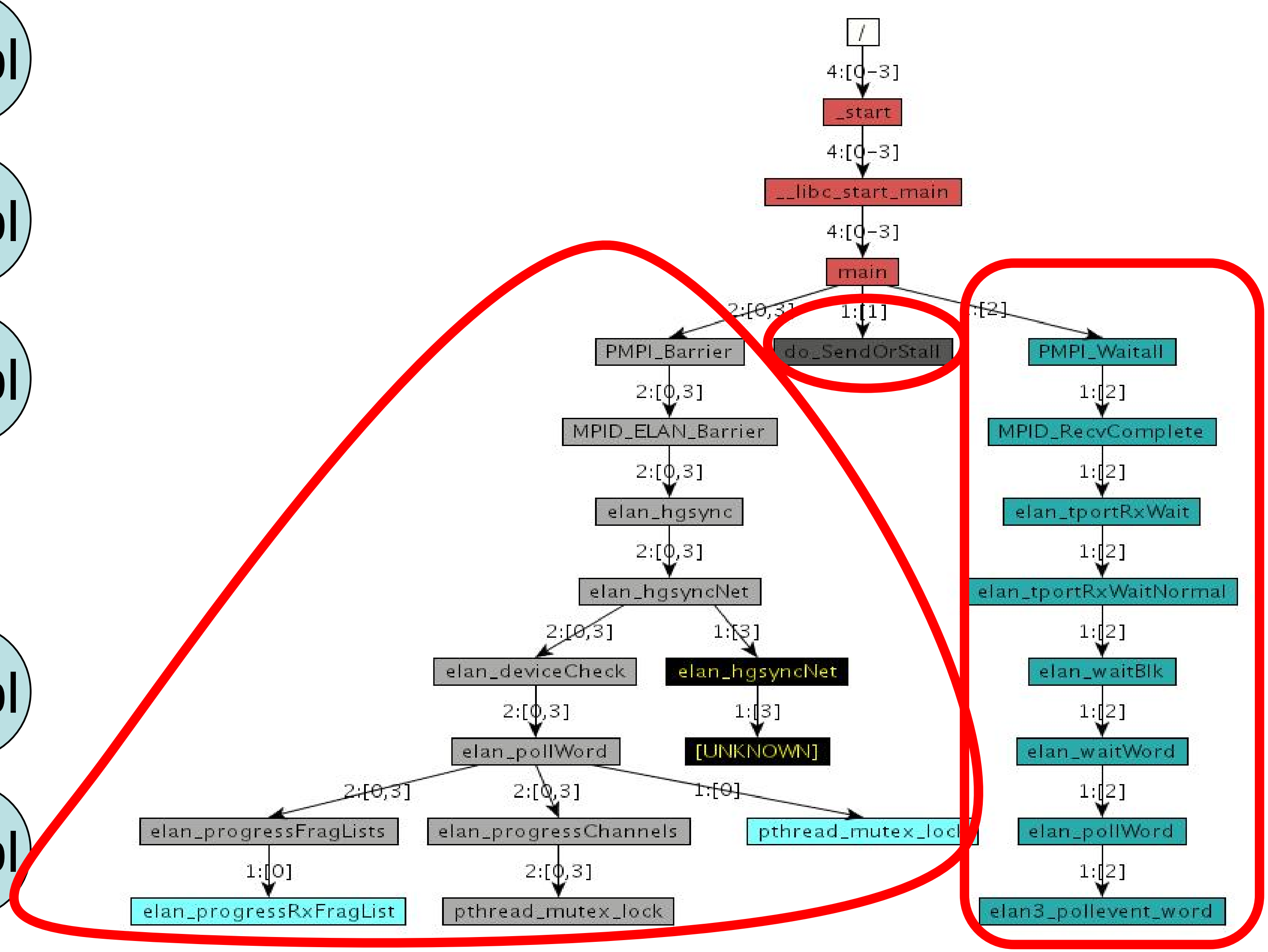
(Project by LLNL, UW, UNM)

Distinguishing Behavior with Stack Traces

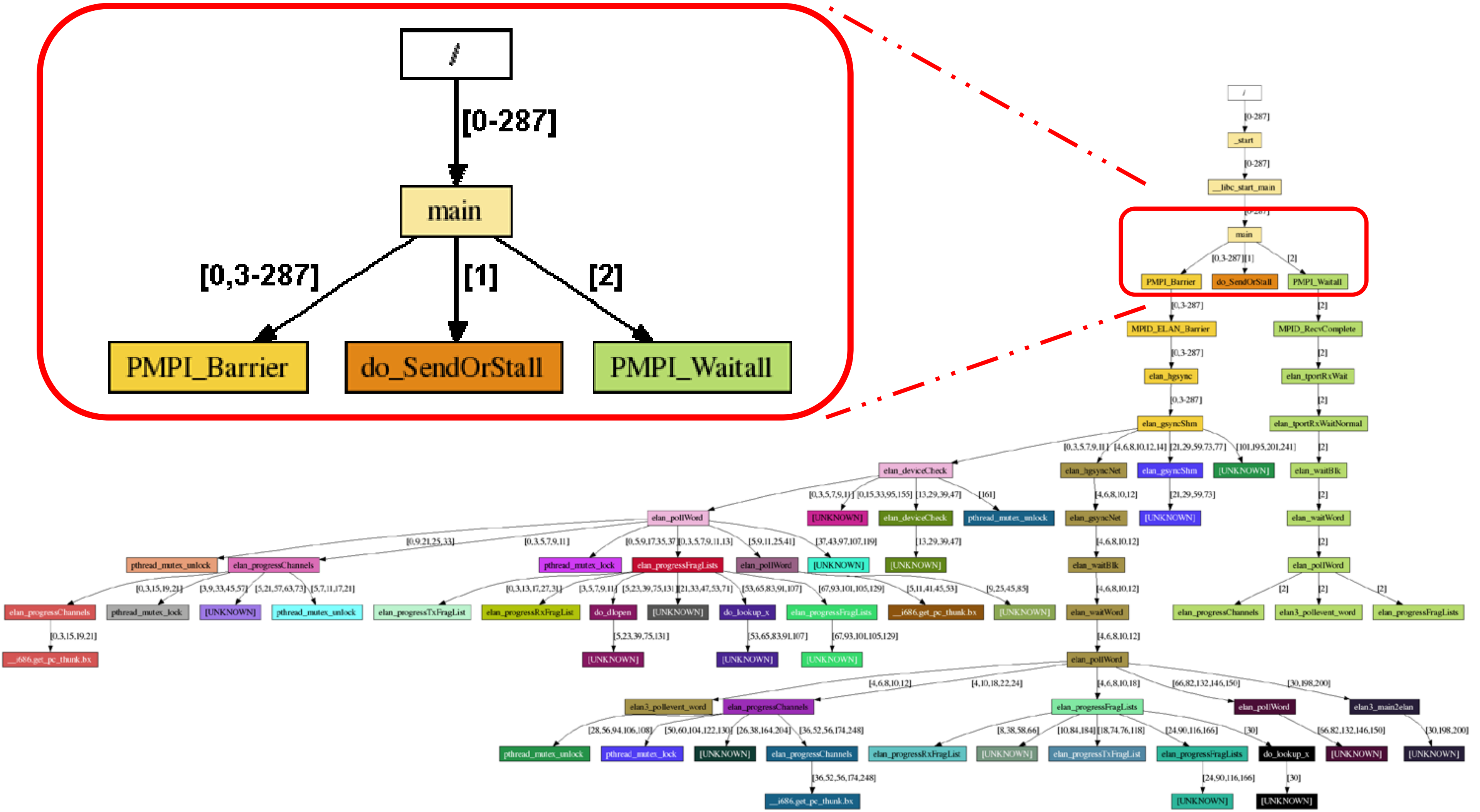


3D-Trace Space/Time Analysis

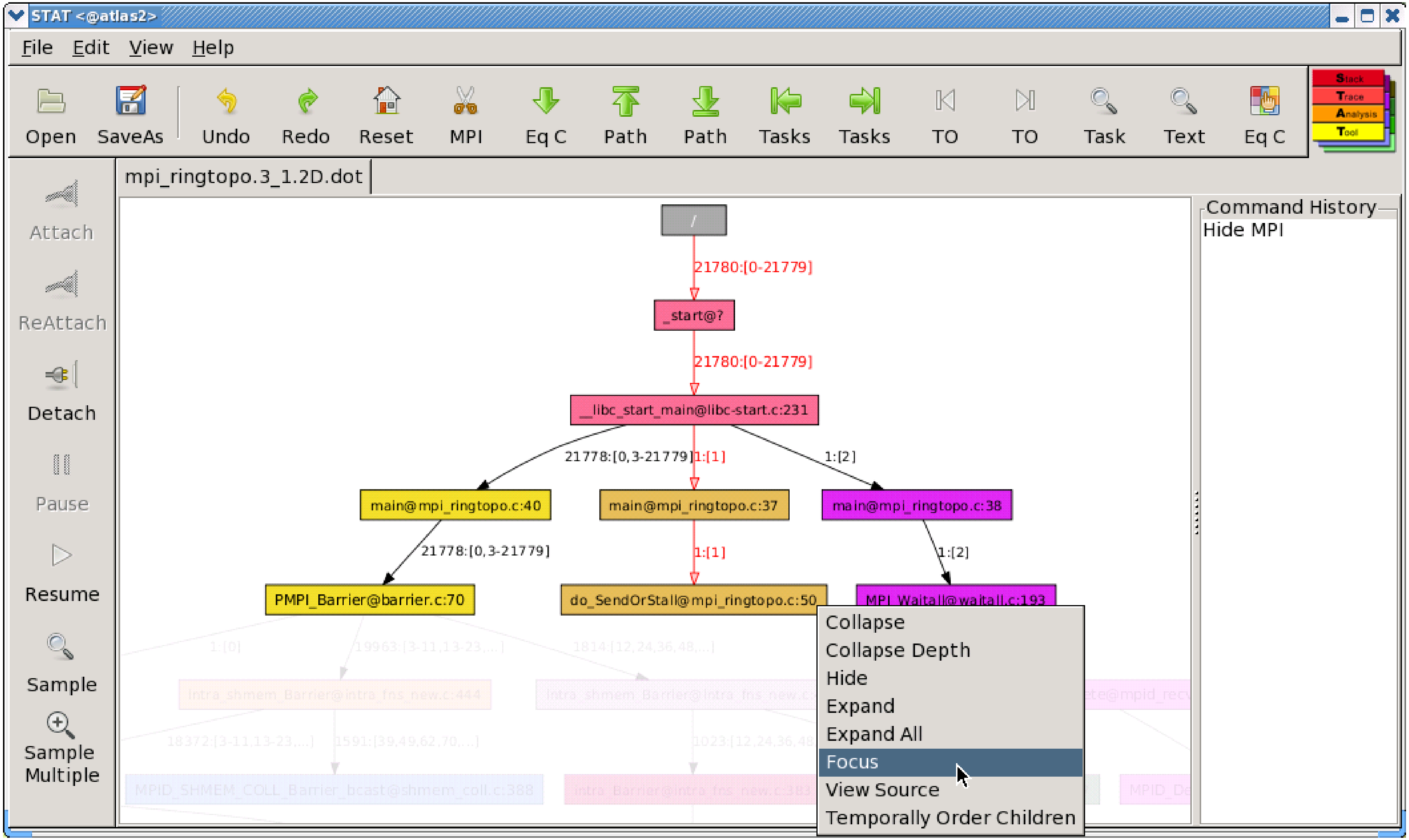




Scalable Representation



STAT GUI

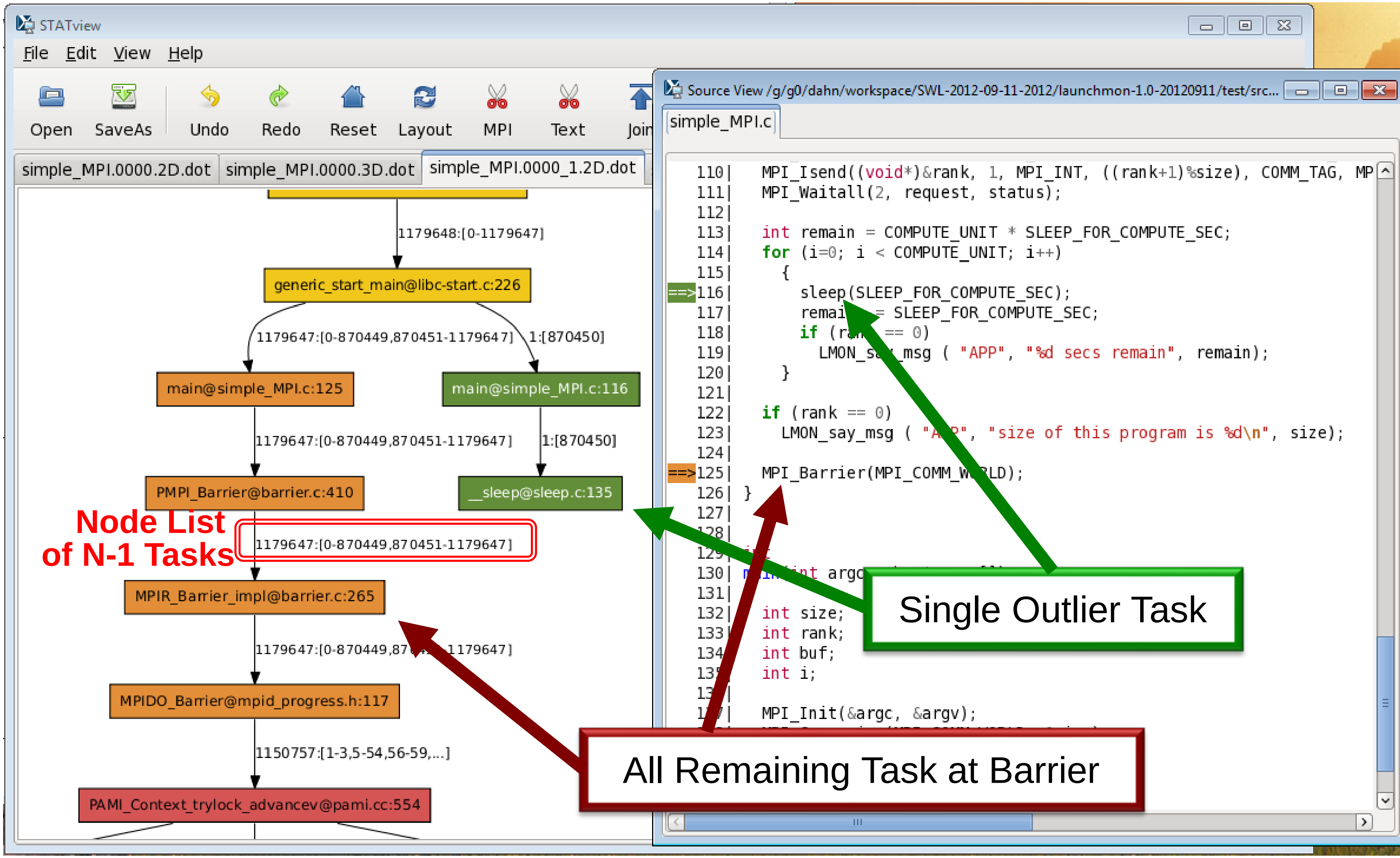


STAT <@atlas2>
File
Edit
View
Help
Open
SaveAs
Undo
Redo
Reset
MPI
Eq C
Path
Path
Tasks
Tasks
TO
TO
Task
Text
Eq C
Stack
Trace
Analysis
Tool
Attach
ReAttach
Detach
Pause
Resume
Sample
Sample Multiple
mpi_ringtopo.3_1.2D.dot
Command History
Hide MPI
/
21780:[0-21779]
_start@?
21780:[0-21779]
__libc_start_main@libc-start.c:231
21778:[0,3-21779]
1:[1]
1:[2]
main@mpi_ringtopo.c:40
main@mpi_ringtopo.c:37
main@mpi_ringtopo.c:38
21778:[0,3-21779]
1:[1]
1:[2]
PMPI_Barrier@barrier.c:70
do_SendOrStall@mpi_ringtopo.c:50
MPI_Waitall@waitall.c:193
Collapse
Collapse Depth
Hide
Expand
Expand All
Focus
View Source
Temporally Order Children

Experience with STAT

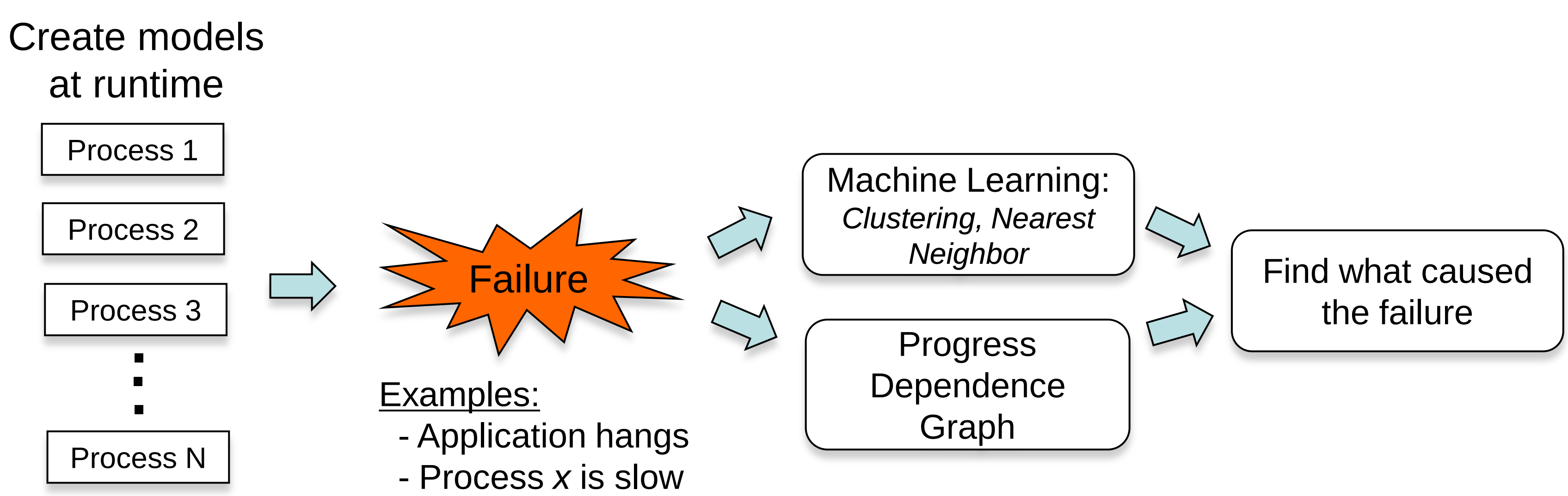
- Equivalence classes
  - Scenario 1: pick one representative for each class
  - Scenario 2: pick one “suspicious” equivalence class
  - Focus debugger on subset of processes
- Highly effective tool at large scale
  - Quick overview capturing hung tasks
  - Allows to focus interactive debugger to only a few tasks
- Typically used as first line of defense
  - Easy to use and non intrusive
  - Attach option for already running jobs
- Enables identification of software and hardware bugs
  - Detects outliers independent of cause

Case Study: STAT at > 1 MPI Million Processes



AutomaDeD: Probabilistic Bug Detection

- Goal: identify root cause a bug
  - Exploit static code and dynamic properties
  - Probabilistic anomaly detection
  - Identify least progressed task as likely culprit
  - Combine with static slicing to detect code location
- Status: release available in the next months

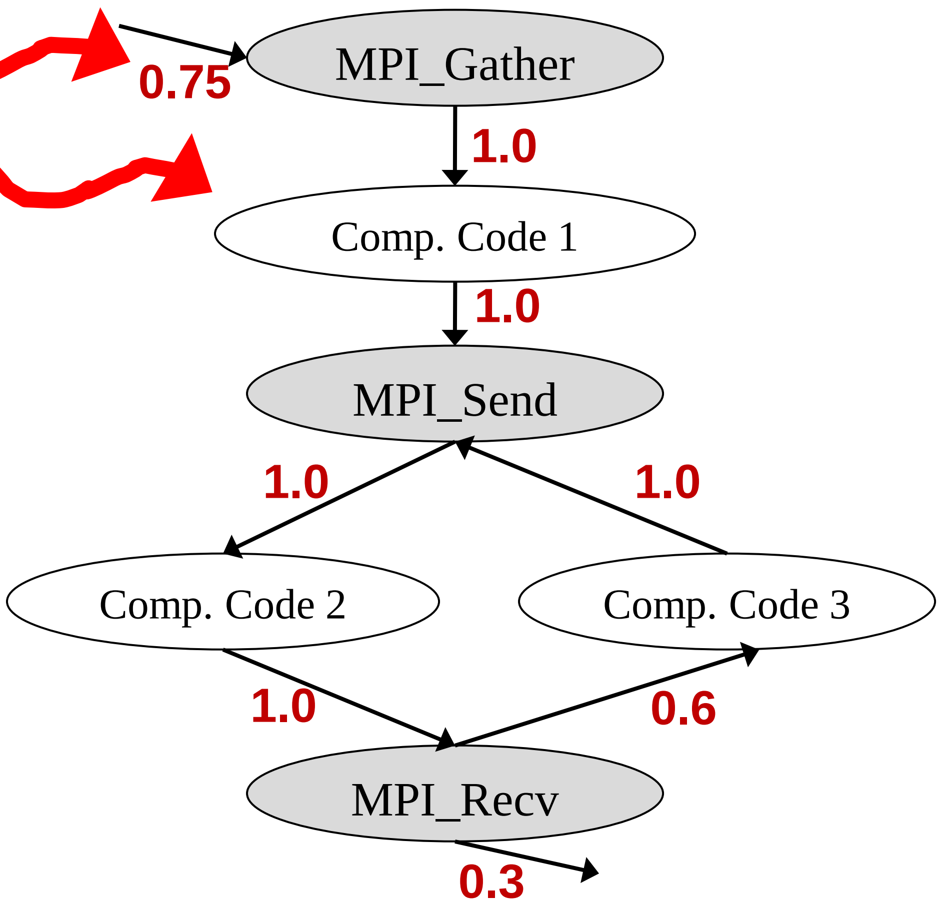


Each MPI Tasks is Modeled as a Markov Model

Sample code

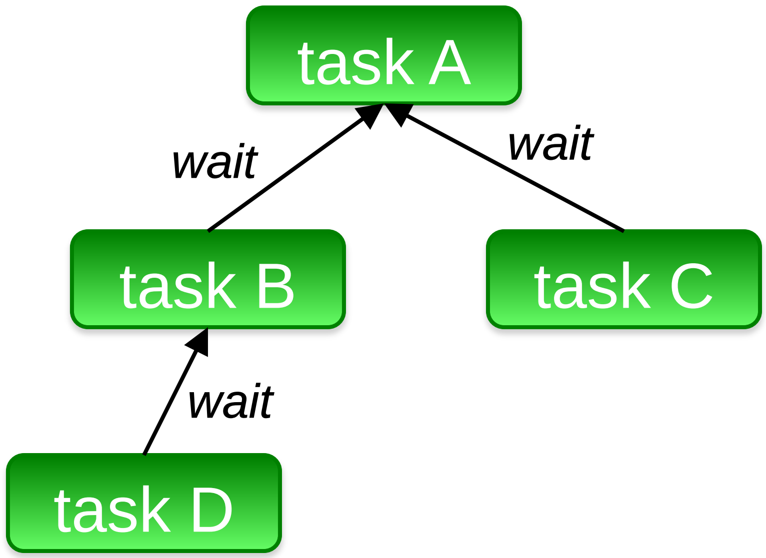
```
foo() {
   MPI_gather( )
   // Computation code
   for (…) {
      // Computation code
      MPI_Send( )
      // Computation code
      MPI_Recv( )
      // Computation code
   }
```

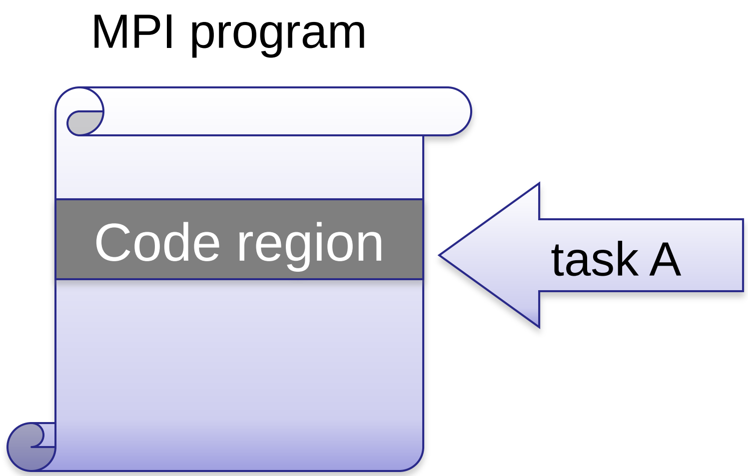
Markov Model



MPI calls wrappers:
- Gather call stack
- Create states in the model

Basis: Progress Dependence Graph

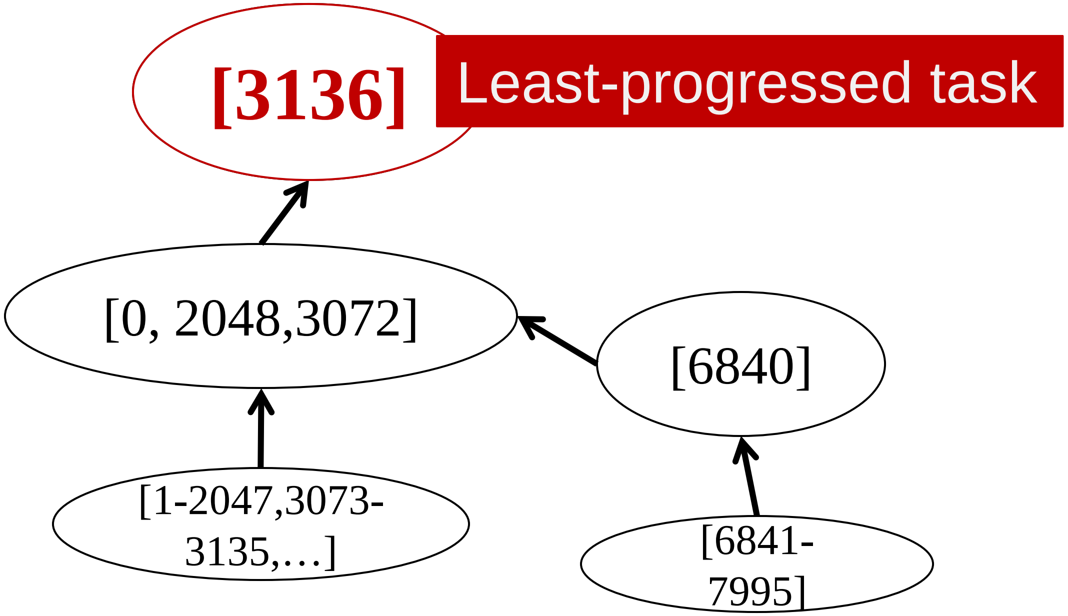




- Facilitates finding the origin of performance faults
- Allows programmer to focus on the origin of the problem:
  The least progressed task
- Distributed Algorithm to infer this from Markov models

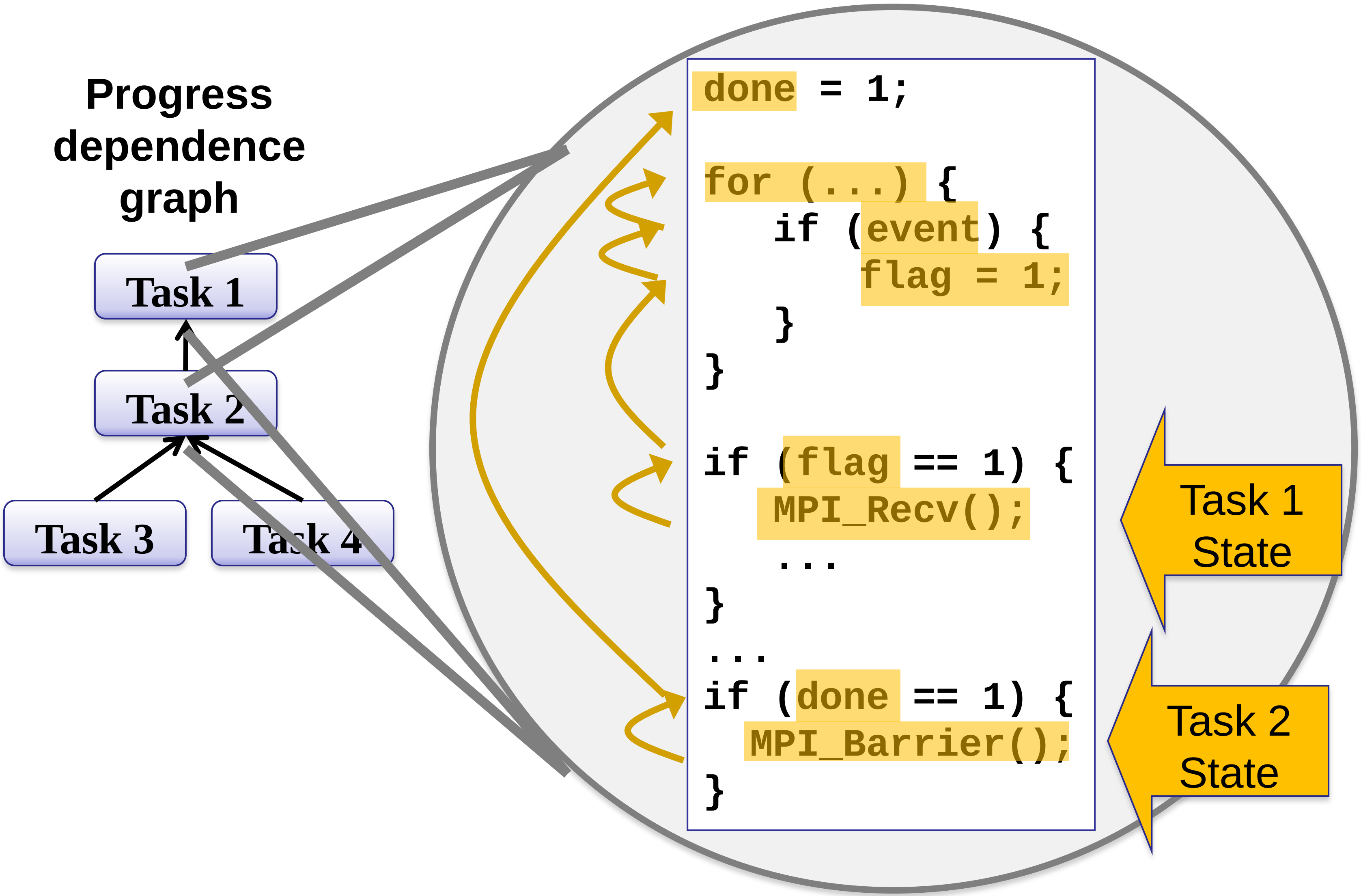
Case Study

Hang with ~8,000 MPI tasks on BlueGene/L



- AutomaDeD finds that task 3136 is the origin of the hang
  - *How did it reach its current state?*

Finding the Faulty Code Region: *Program Slicing*



Slice with Origin of the Bug

Case Study

```
dataWritten = 0
for (…) {
   MPI_Probe(…, &flag, …)
   if (flag == 1) {
      MPI_Recv()
      MPI_Send()
      dataWritten = 1
   }
   MPI_Send()
   MPI_Recv()
   // Write data
}
if (dataWritten == 0) {
   MPI_Recv()
   MPI_Send()
}
Reduce()
Barrier()
```

Least-progressed task State

Dual condition occurs in BlueGene/L

- *A task is a writer and a non-writer*

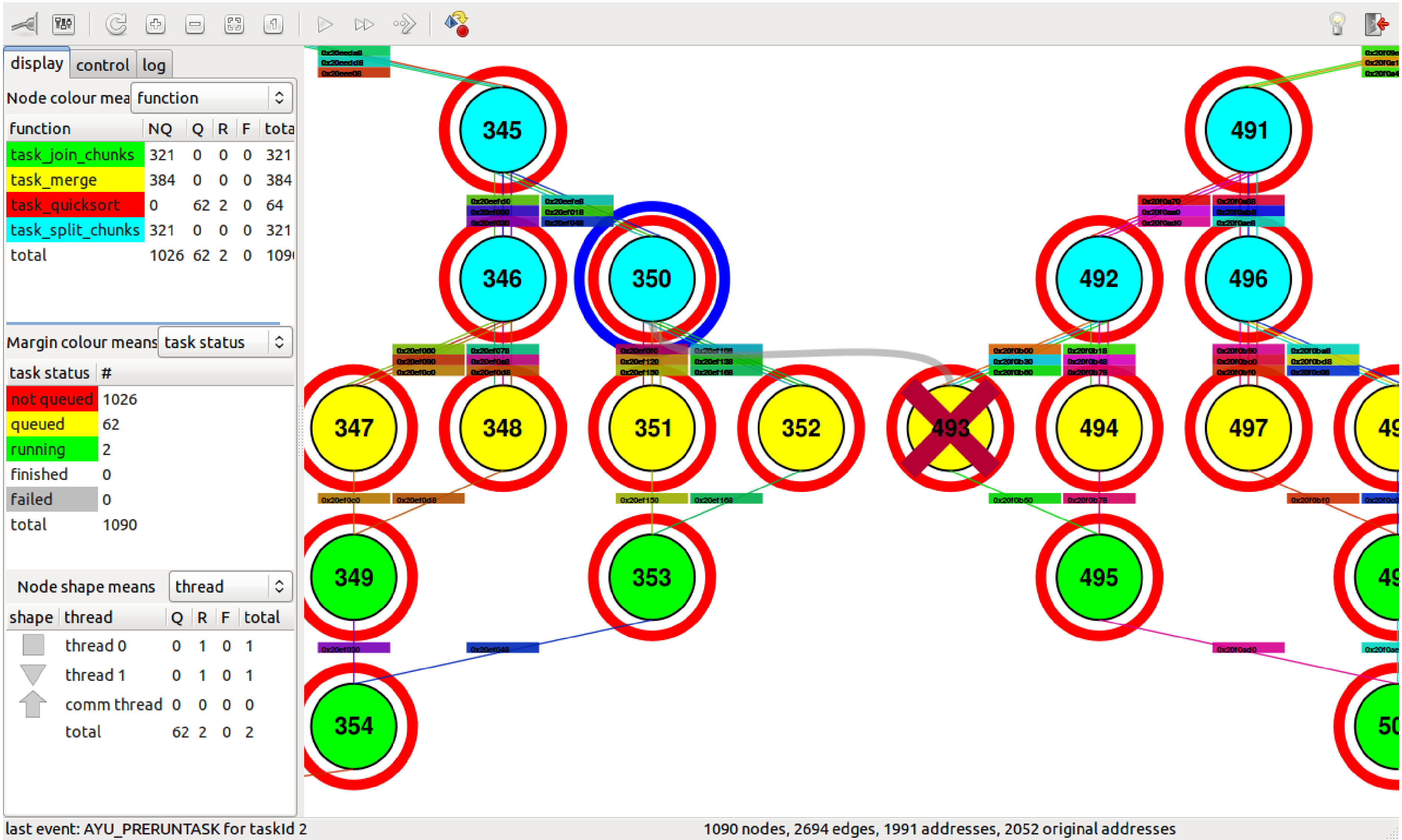
`MPI_Probe` checks for *source, tag and comm* of a message

- *Another writer intercepted wrong message*

Programmer used unique MPI tags to isolate different I/O groups

- Tool for debugging task-based programming models
  - Intuitive GUI to display and control program execution
  - Shows tasks and dependencies to analyse their properties
  - Controls task dependencies and synchronisation barriers
- Currently supports SMPSs and basic MPI usage
  - support in development for OpenMP, OmpSs, etc., and hybrid combinations with MPI
  - based on Ayudame runtime library
- Developed by HLRS
  - Available from http://www.hlrs.de/organization/av/spmt/research/temanejo/

Temanejo task debugger GUI

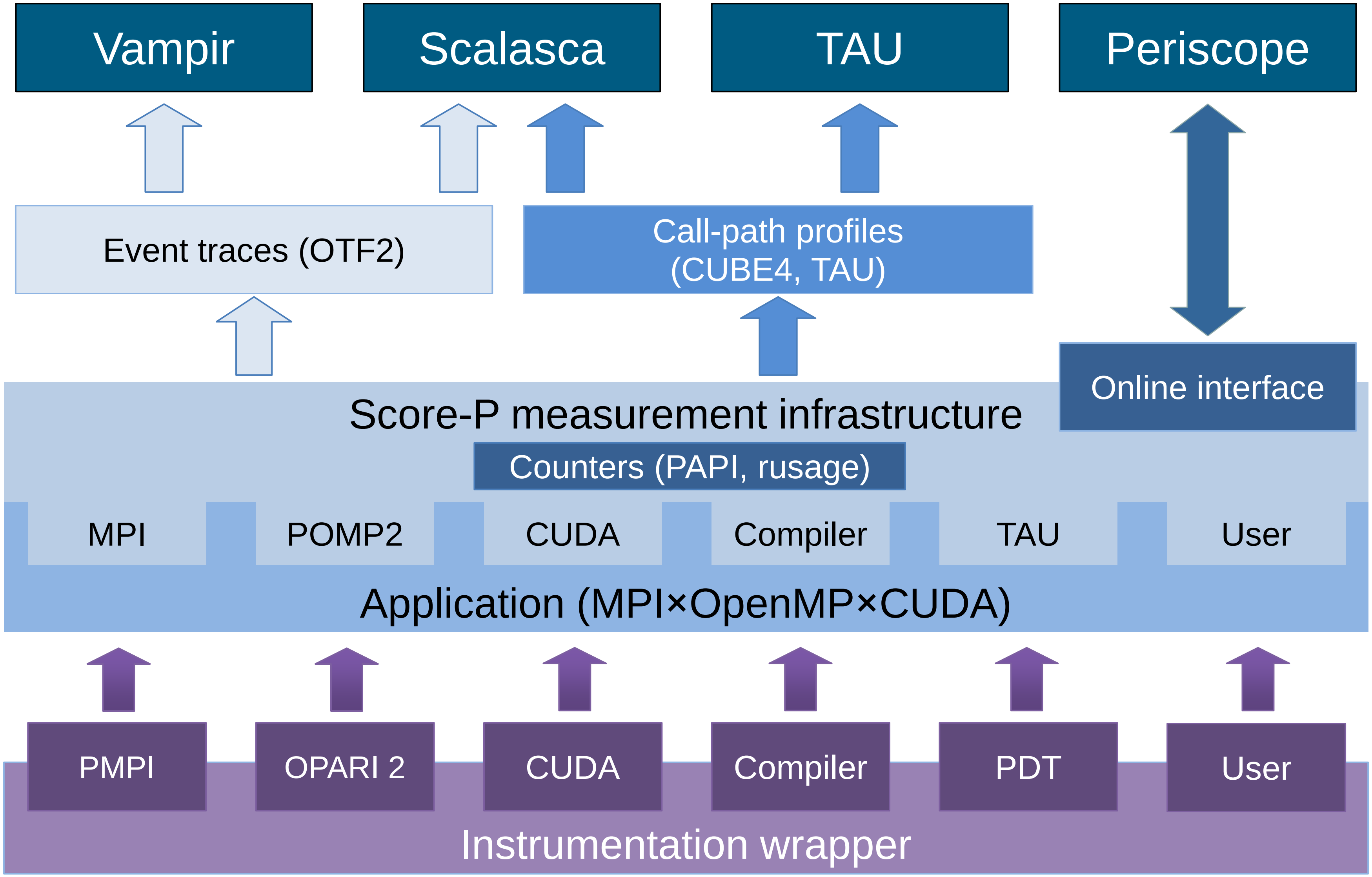


Integrated application execution profiling and trace analysis

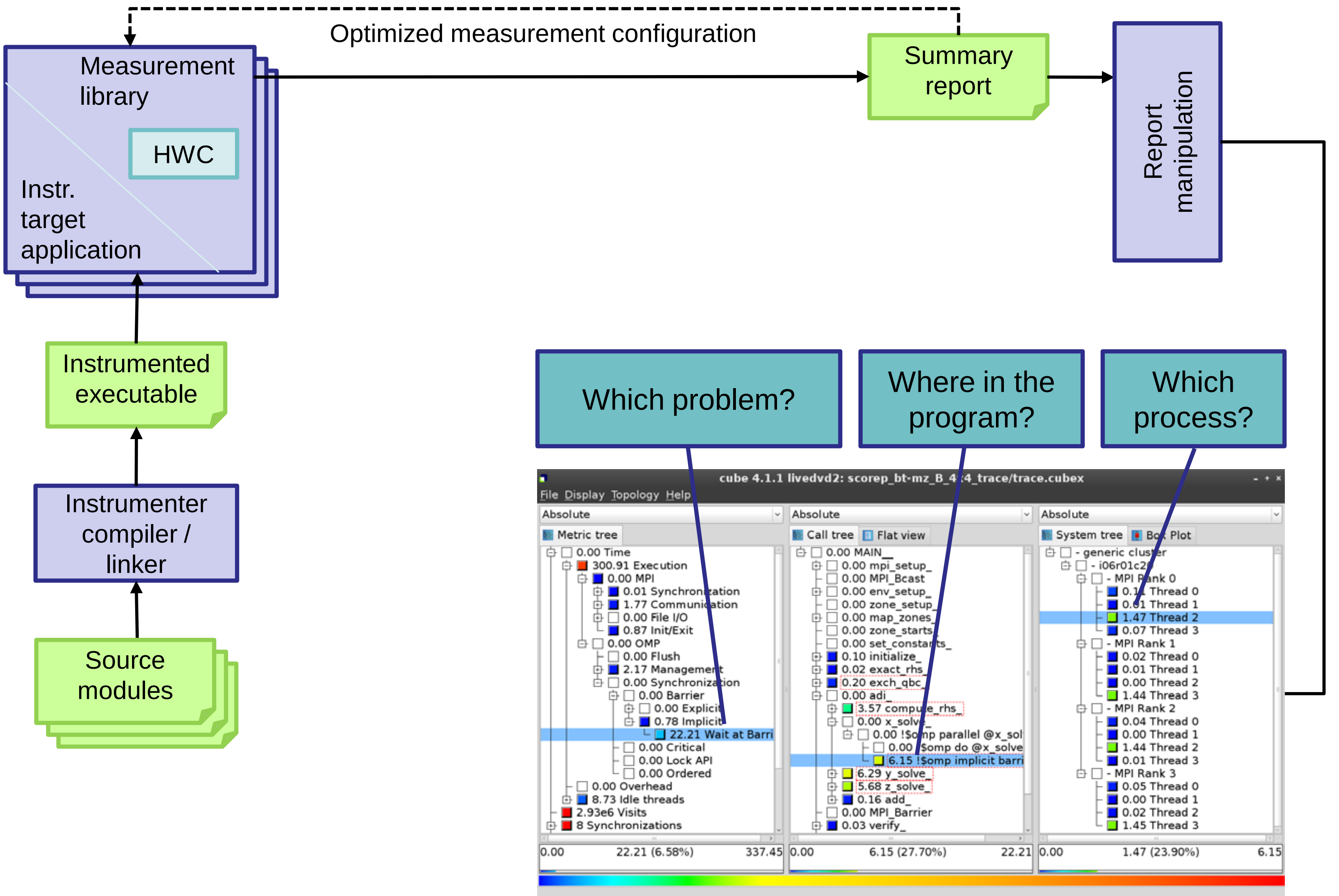
- instrumentation & measurement (**Score-P**, Extrae)
- profile analysis examination (**CUBE**, **ParaProf**)
- execution trace exploration (**Vampir**, **Paraver**)
- automated trace analysis (**Scalasca**)
- on-line automated analysis (**Periscope**)

- Scalable performance measurement infrastructure
  - Supports instrumentation, profiling & trace collection, as well as online analysis of HPC parallel applications
    - MPI, OpenMP & CUDA (including combinations)
  - Used by latest versions of Periscope, Scalasca, TAU & Vampir
  - Based on updated tool components
    - CUBE4 profile data utilities & GUI
    - OA online access interface to performance measurements
    - OPARI2 OpenMP & pragma instrumenter
    - OTF2 open trace format
- Created by BMBF SILC & US DOE PRIMA projects
  - JSC, RWTH, TUD, TUM, GNS, GRS, GWT & UO PRL
  - Available as BSD open-source from http://www.score-p.org/

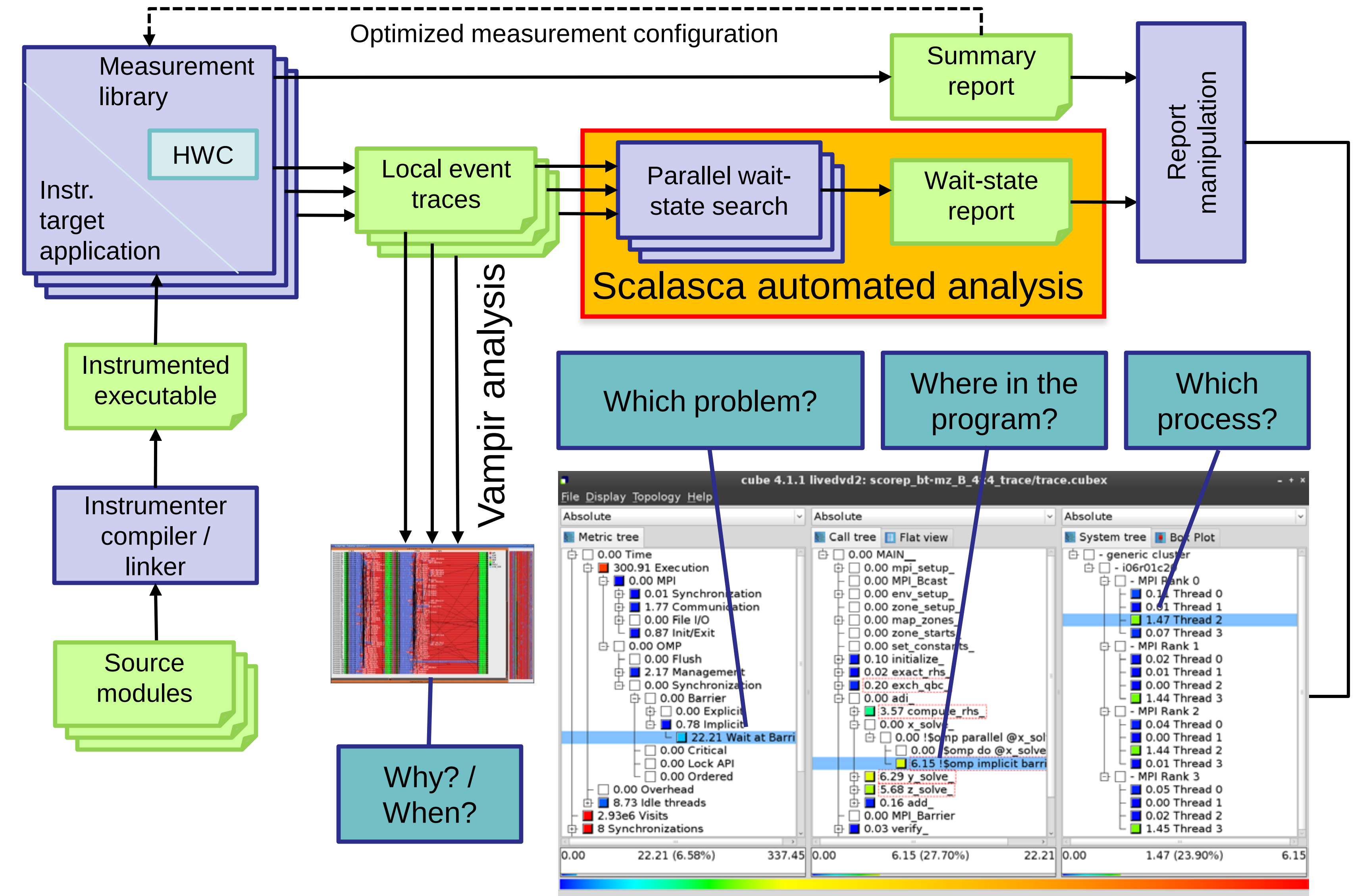
Score-P architecture



Score-P workflow (runtime summarization)



Score-P workflow (trace collection & analyses)



Preparation and initial analysis commands

- Use instrumenter as preposition for source compilation & link commands to produce instrumented executable

```
% scorep --user  mpif77 –fopenmp –O3 –c bt.f
% scorep --user  mpif77 –fopenmp –O3 –o bt_mz.4
```

- Use measurement nexus as execution preposition to configure measurement collection and analysis

```
% OMP_NUM_THREADS=4  scan –s  mpiexec –np 4  bt-mz.4
-> scorep_bt-mz_4x4_sum
```

- Score measurement to assess quality, determine routines to filter and expected trace buffer content

```
% square –s scorep_bt-mz_4x4_sum
-> scorep_bt-mz_4x4_sum/scorep.score
```

Measurement and analysis commands

- Revise measurement configuration as appropriate

```
% export SCOREP_METRIC_PAPI=PAPI_FP_OPS,PAPI_L2_DCM
% OMP_NUM_THREADS=4  scan –f scorep.filt  mpiexec –np 4  bt-mz.4
-> scorep_bt-mz_4x4_sum
```

- Collected trace automatically analysed in parallel using the same execution configuration

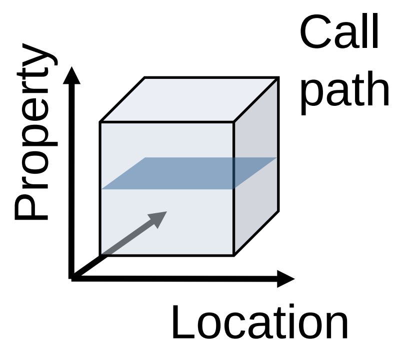
```
% OMP_NUM_THREADS=4  scan –f scorep.filt -t  mpiexec –np 4  bt-mz.4
-> scorep_bt-mz_4x4_trace
```

- Postprocess intermediate analysis report(s) to derive additional metrics and hierarchy, then explore with GUIs

```
% square scorep_bt-mz_4x4_trace
-> scorep_bt-mz_4x4_trace/trace.cubex
-> [CUBE GUI]
% vampir scorep_bt-mz_4x4_trace/traces.otf2
-> [Vampir GUI]
```

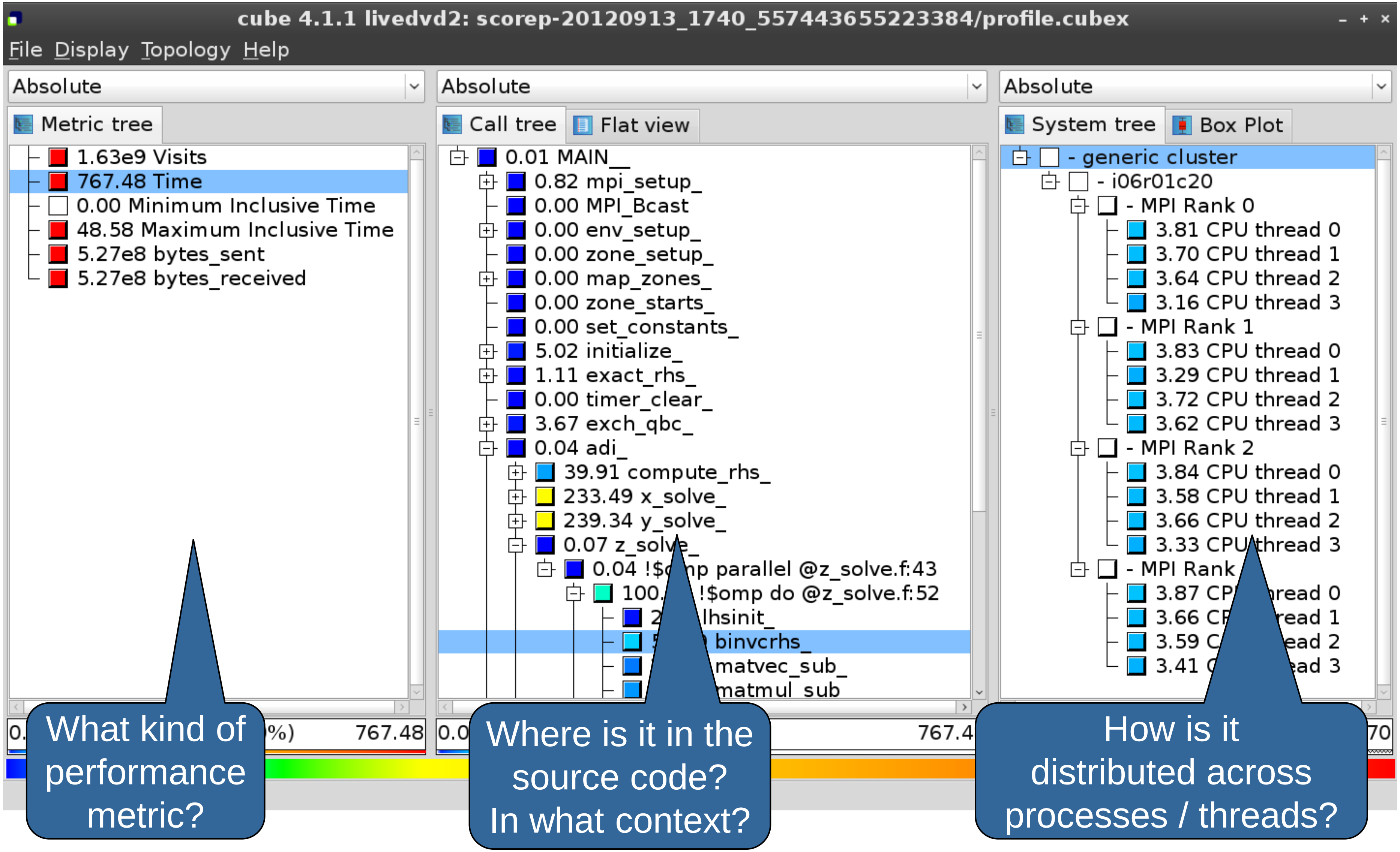
- Parallel program analysis report exploration tools
  - Libraries for XML report reading & writing
  - Algebra utilities for report processing
  - GUI for interactive analysis exploration
- Used by Score-P and Scalasca for analysis reports
  - Non-GUI libraries required by Score-P for scoring reports
  - Can also be installed independently of Score-P, e.g., on laptop or desktop, for local analysis exploration with GUI
- Developed originally as part of Scalasca toolset
  - New BSD open-source license
  - www.scalasca.org

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



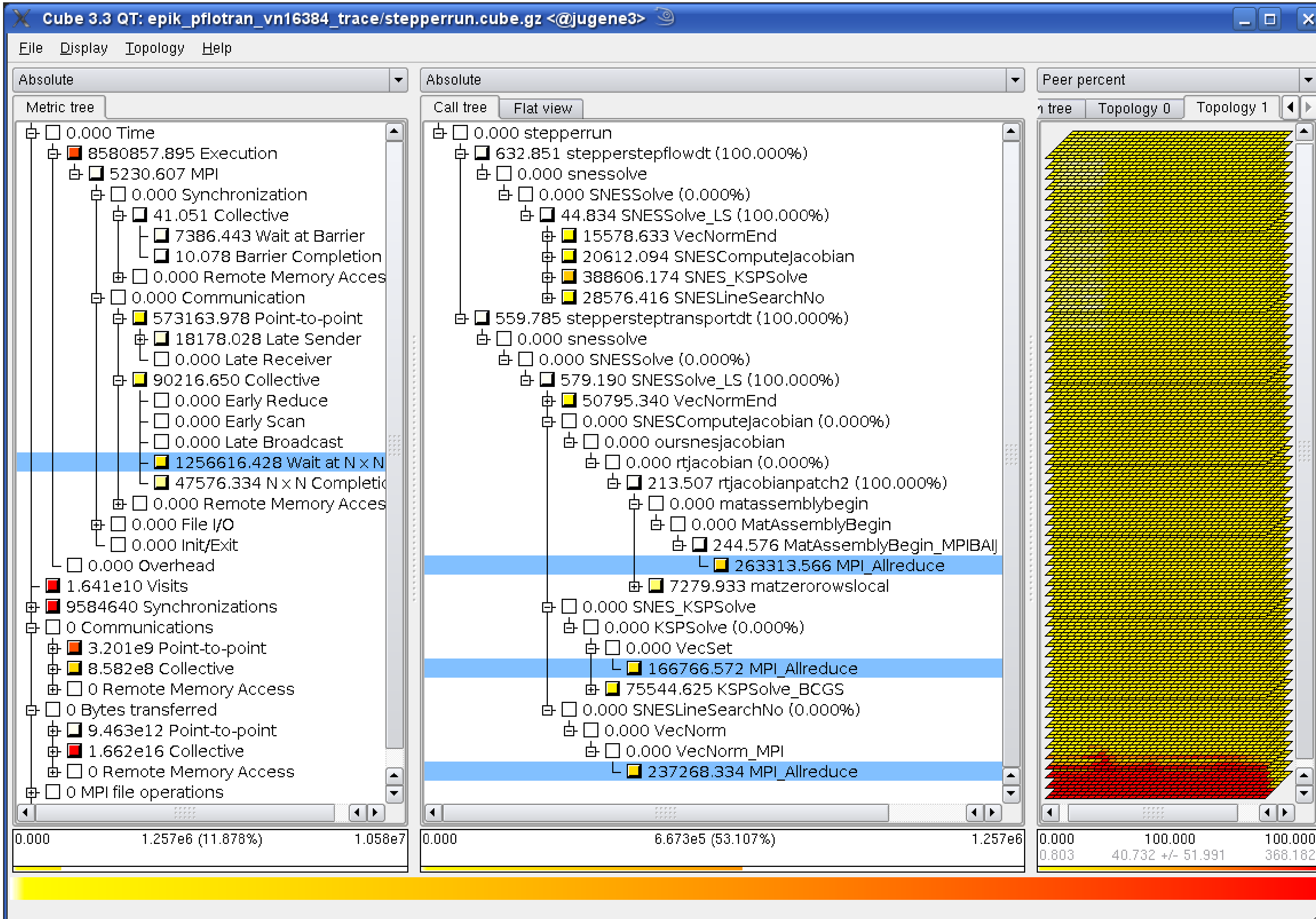
- Three coupled tree browsers

- 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 modes

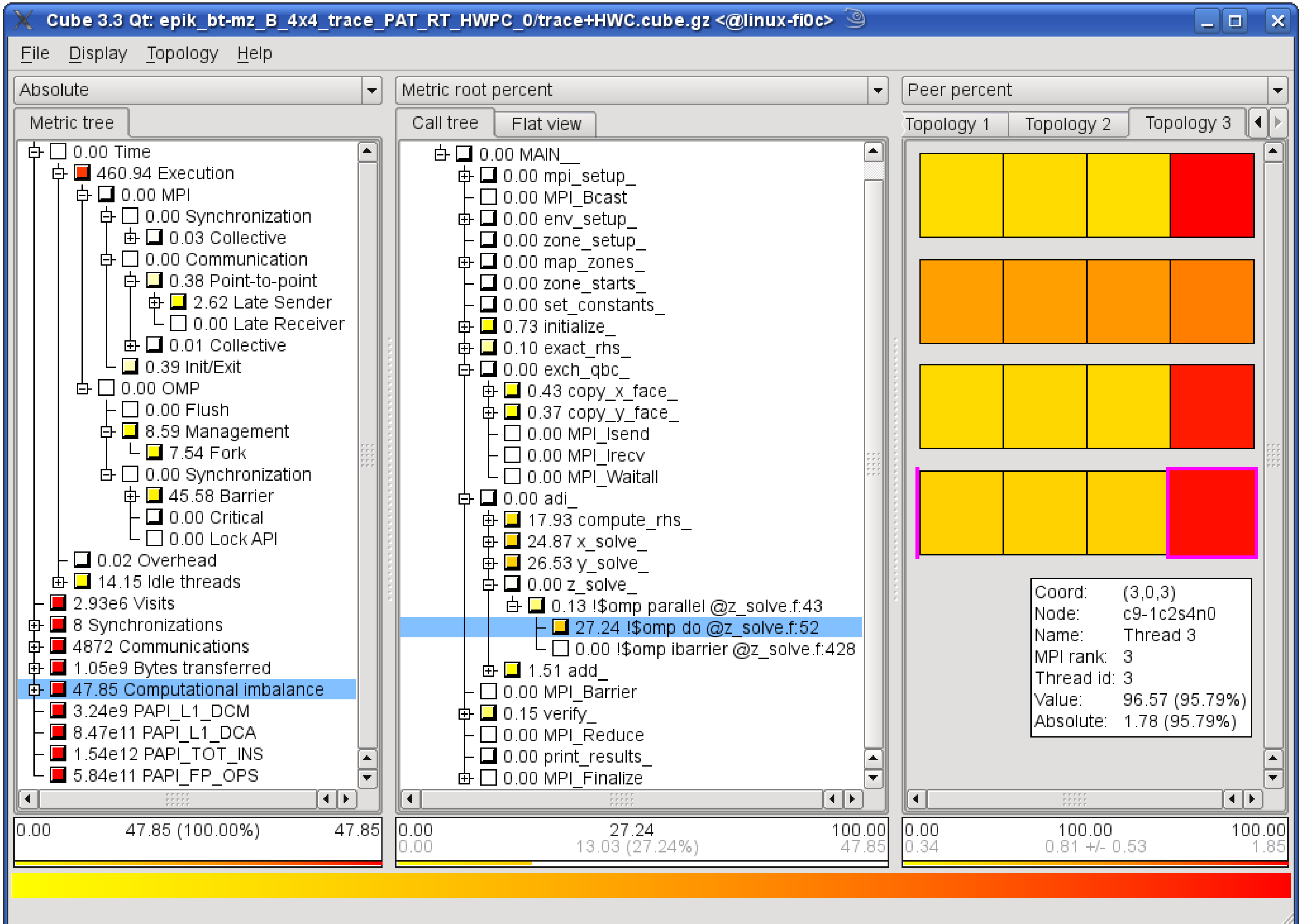

cube 4.1.1 livedvd2: scorep-20120913_1740_557443655223384/profile.cubex
File Display Topology Help
Absolute
Metric tree
1.63e9 Visits
767.48 Time
0.00 Minimum Inclusive Time
48.58 Maximum Inclusive Time
5.27e8 bytes_sent
5.27e8 bytes_received
Call tree
Flat view
0.01 MAIN__
0.82 mpi_setup_
0.00 MPI_Bcast
0.00 env_setup_
0.00 zone_setup_
0.00 map_zones_
0.00 zone_starts_
0.00 set_constants_
5.02 initialize_
1.11 exact_rhs_
0.00 timer_clear_
3.67 exch_qbc_
0.04 adi_
39.91 compute_rhs_
233.49 x_solve_
239.34 y_solve_
0.07 z_solve_
0.04 !$omp parallel @z_solve.f:43
!$omp do @z_solve.f:52
lhsinit_
binvcrhs_
matvec_sub_
matmul_sub
System tree
Box Plot
- generic cluster
- i06r01c20
- MPI Rank 0
3.81 CPU thread 0
3.70 CPU thread 1
3.64 CPU thread 2
3.16 CPU thread 3
- MPI Rank 1
3.83 CPU thread 0
3.29 CPU thread 1
3.72 CPU thread 2
3.62 CPU thread 3
- MPI Rank 2
3.84 CPU thread 0
3.58 CPU thread 1
3.66 CPU thread 2
3.33 CPU thread 3
- MPI Rank
767.48
What kind of performance metric?
Where is it in the source code? In what context?
How is it distributed across processes / threads?

- Automatic performance analysis toolset
  - Scalable performance analysis of large-scale applications
    - particularly focused on MPI & OpenMP paradigms
    - analysis of communication & synchronization overheads
  - Automatic and manual instrumentation capabilities
  - Runtime summarization and/or event trace analyses
  - Automatic search of event traces for patterns of inefficiency
    - Scalable trace analysis based on parallel replay
  - Interactive exploration GUI and algebra utilities for XML callpath profile analysis reports
- Developed by JSC & GRS
  - Open-source with New BSD license
  - http://www.scalasca.org/

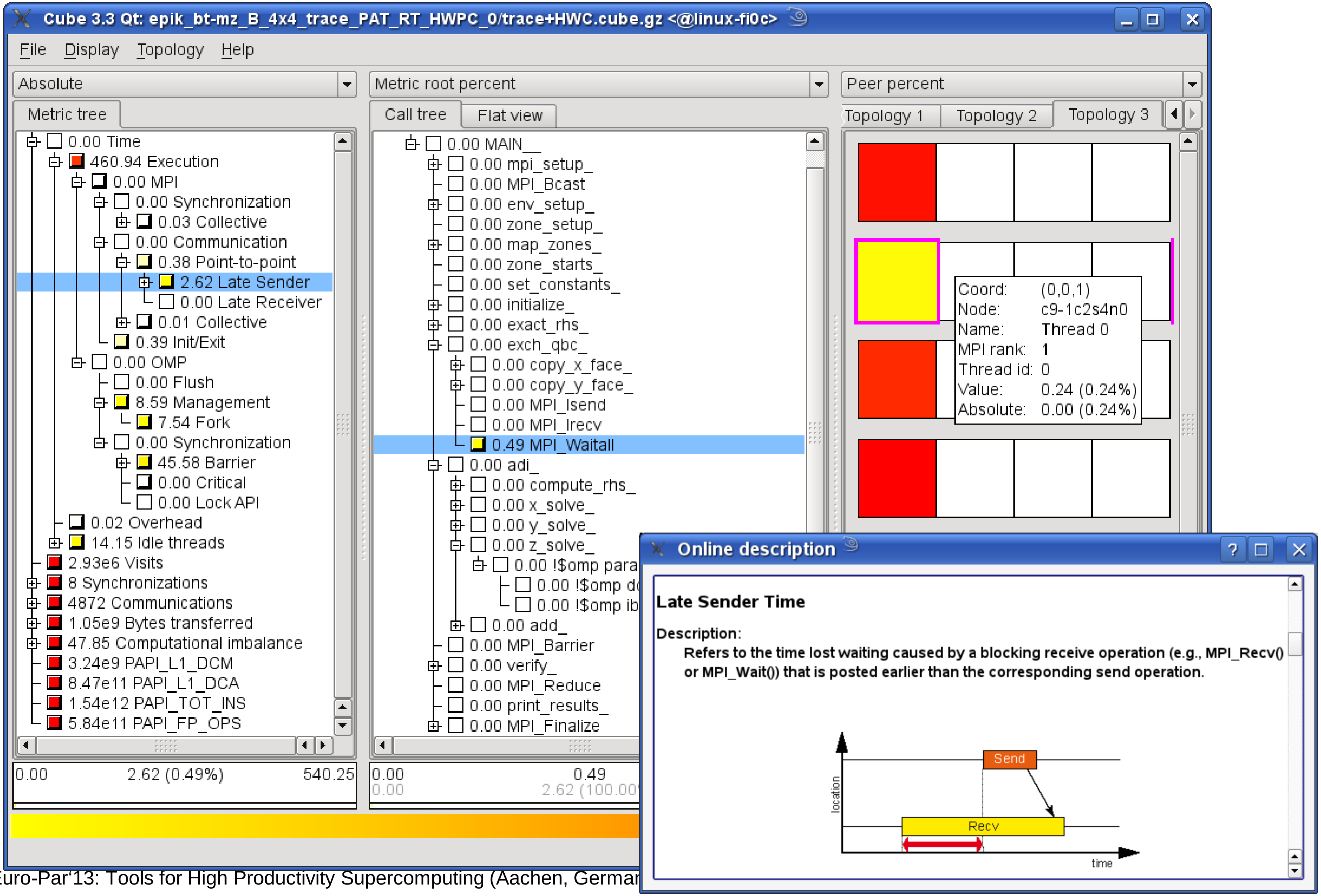
Scalasca automatic trace analysis report



Scalasca hybrid analysis report

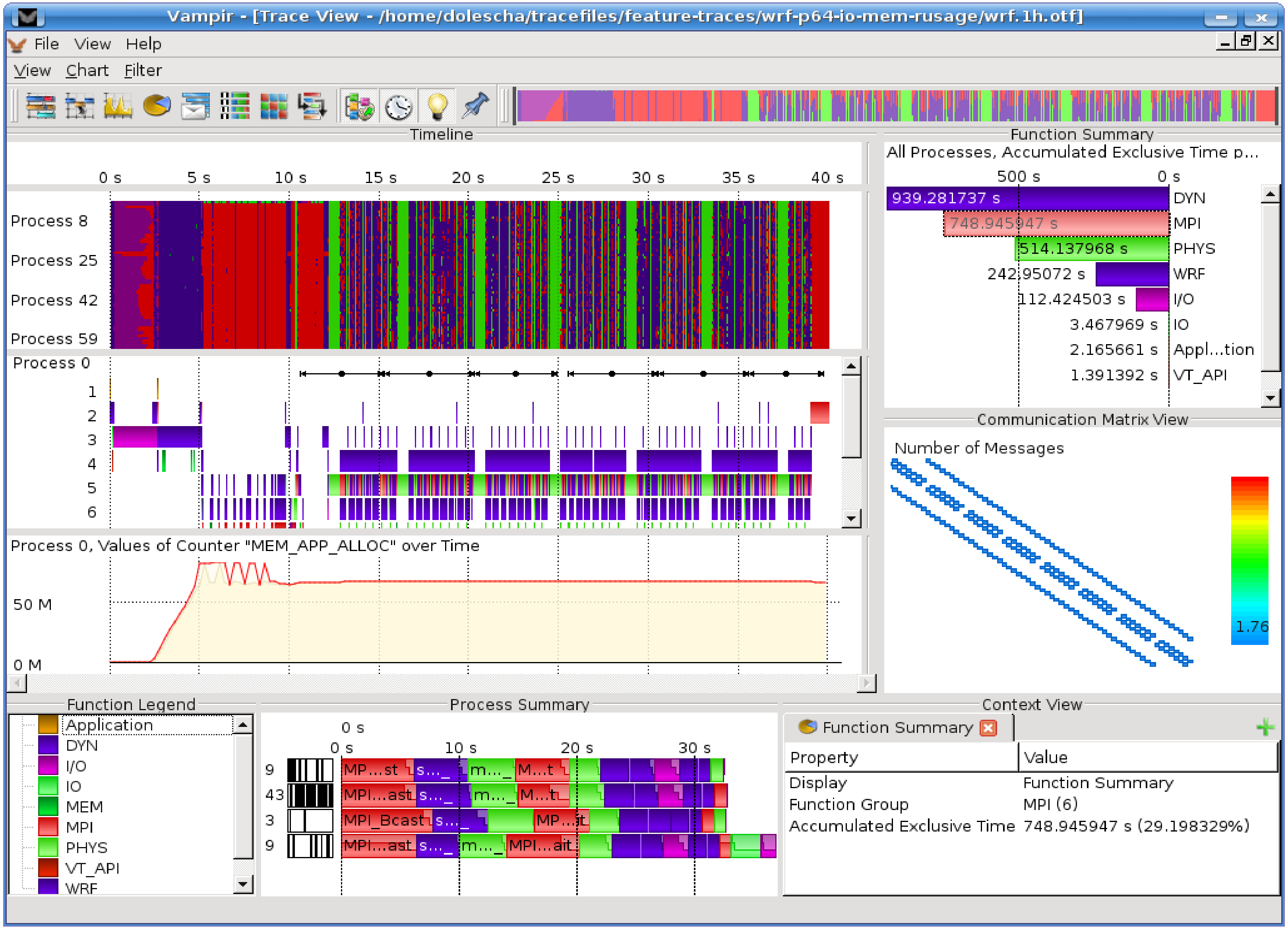


Scalasca automatic trace analysis report

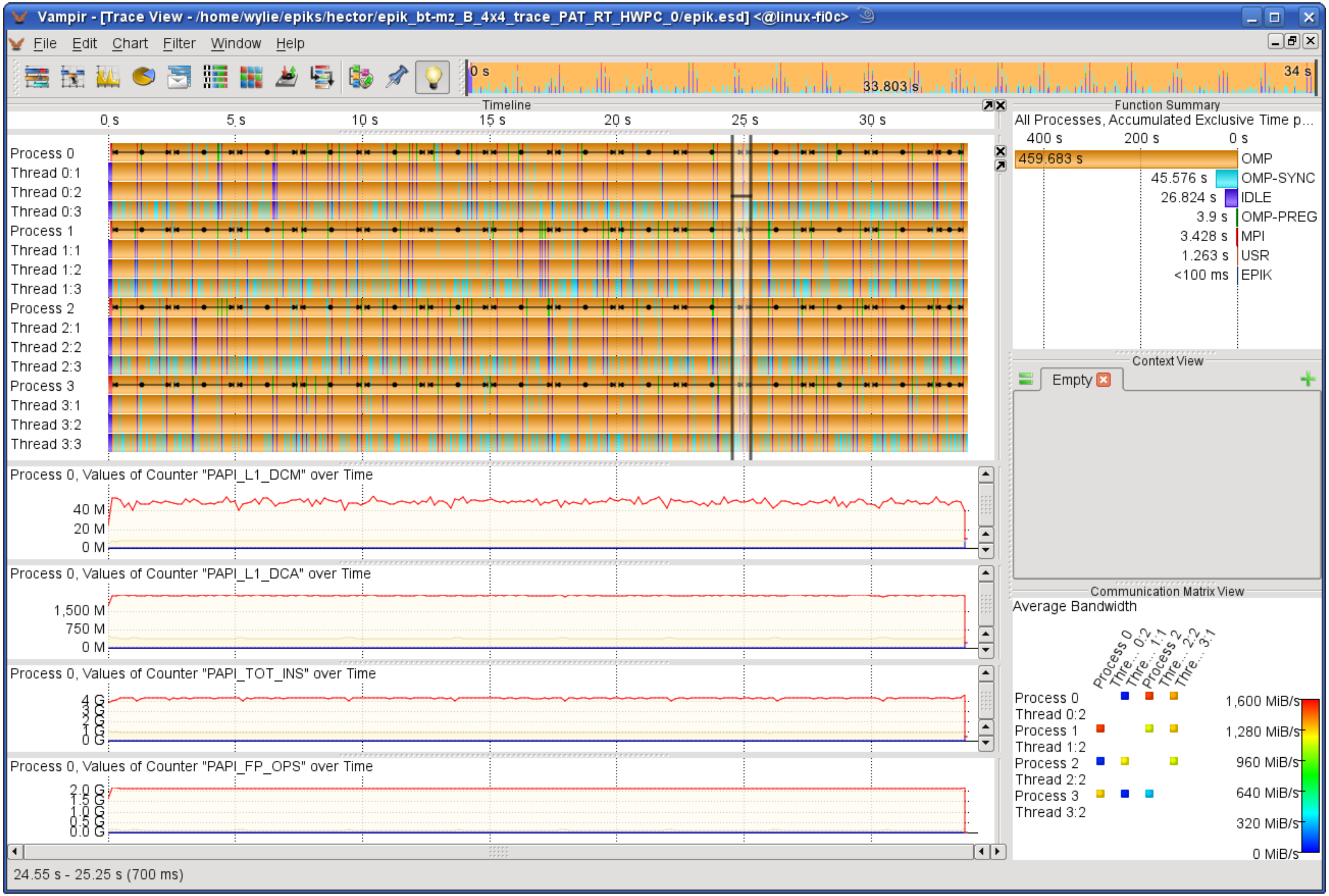


- Interactive event trace analysis
  - Alternative & supplement to automatic trace analysis
  - Visual presentation of dynamic runtime behaviour
    - event timeline chart for states & interactions of processes/threads
    - communication statistics, summaries & more
  - Interactive browsing, zooming, selecting
    - linked displays & statistics adapt to selected time interval (zoom)
    - scalable server runs in parallel to handle larger traces
- Developed by TU Dresden ZIH
  - Open-source VampirTrace library bundled with OpenMPI 1.3
  - http://www.tu-dresden.de/zih/vampirtrace/
  - Vampir Server & GUI have a commercial license
  - http://www.vampir.eu/

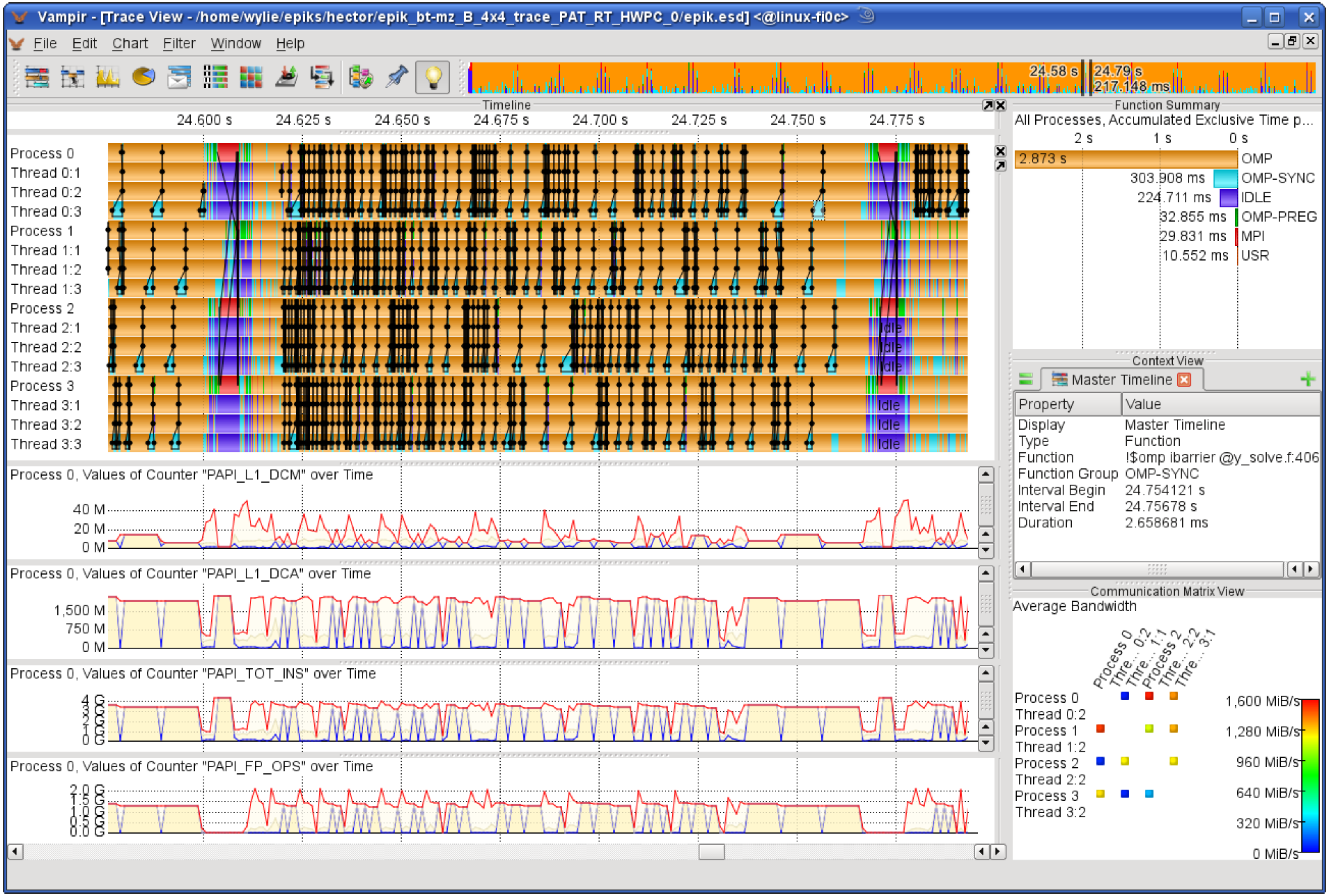
Vampir interactive trace analysis GUI



Vampir interactive trace analysis GUI



Vampir interactive trace analysis GUI (zoom)



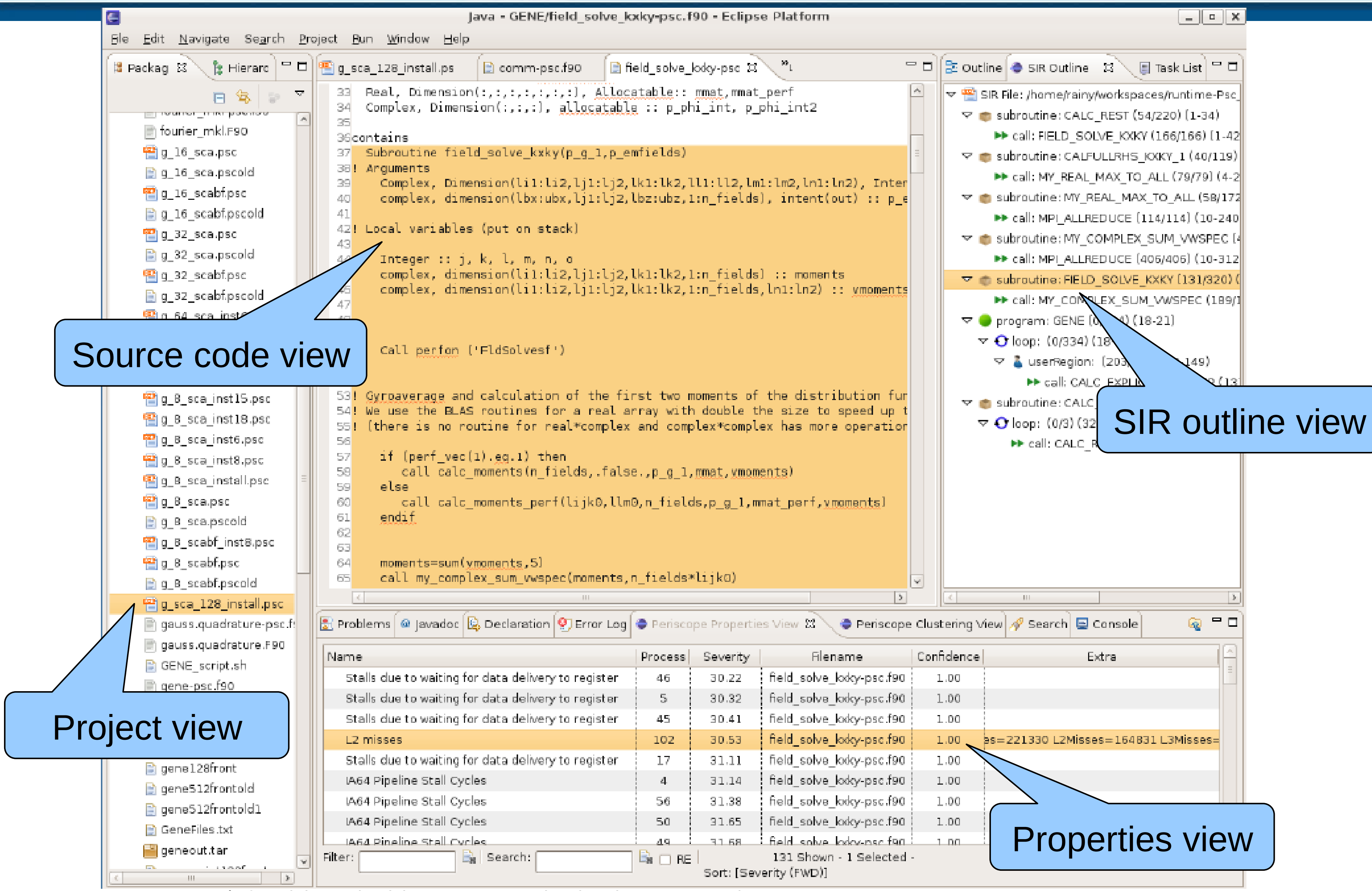
Periscope

- Automated profile-based performance analysis
  - Iterative on-line performance analysis
    - Multiple distributed hierarchical agents
  - Automatic search for bottlenecks based on properties formalizing expert knowledge
    - MPI wait states, OpenMP overheads and imbalances
    - Processor utilization hardware counters
  - Clustering of processes/threads with similar properties
  - Eclipse-based integrated environment
- Supports
  - SGI Altix Itanium2, IBM Power and x86-based architectures
- Developed by TU Munich
  - Released as open-source
  - http://www.lrr.in.tum.de/periscope

Periscope properties & strategies (examples)

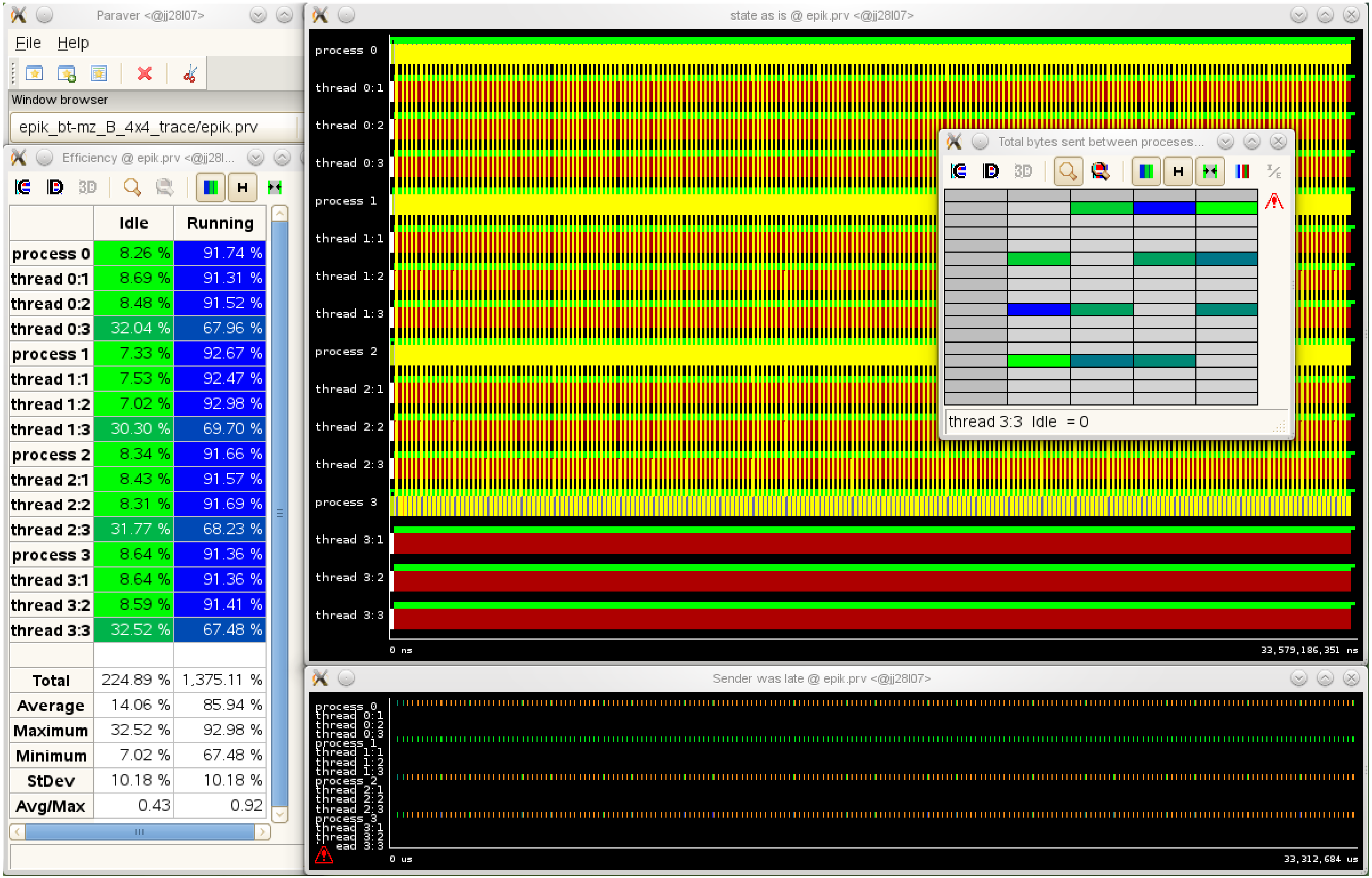
- MPI
  - Excessive MPI communication time
  - Excessive MPI time due to many small messages
  - Excessive MPI time in receive due to late sender
  - …
- OpenMP
  - Load imbalance in parallel region/section
  - Sequential computation in master/single/ordered region
  - ...
- Hardware performance counters (platform-specific)
  - Cycles lost due to cache misses
    - High L1/L2/L3 demand load miss rate
  - Cycles lost due to no instruction to dispatch
  - ...

Periscope plug-in to Eclipse environment

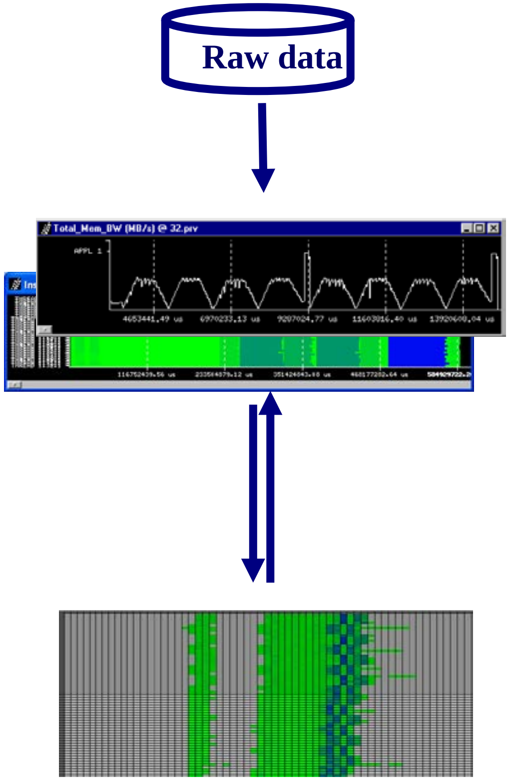


- Interactive event trace analysis
  - Visual presentation of dynamic runtime behaviour
    - event timeline chart for states & interactions of processes
    - Interactive browsing, zooming, selecting
  - Large variety of highly configurable analyses & displays
- Developed by Barcelona Supercomputing Center
  - Paraver trace analyser and Extrae measurement library
  - Dimemas message-passing communication simulator
  - Open source available from http://www.bsc.es/paraver/

Paraver interactive trace analysis GUI



Paraver interactive trace analyses



Timelines

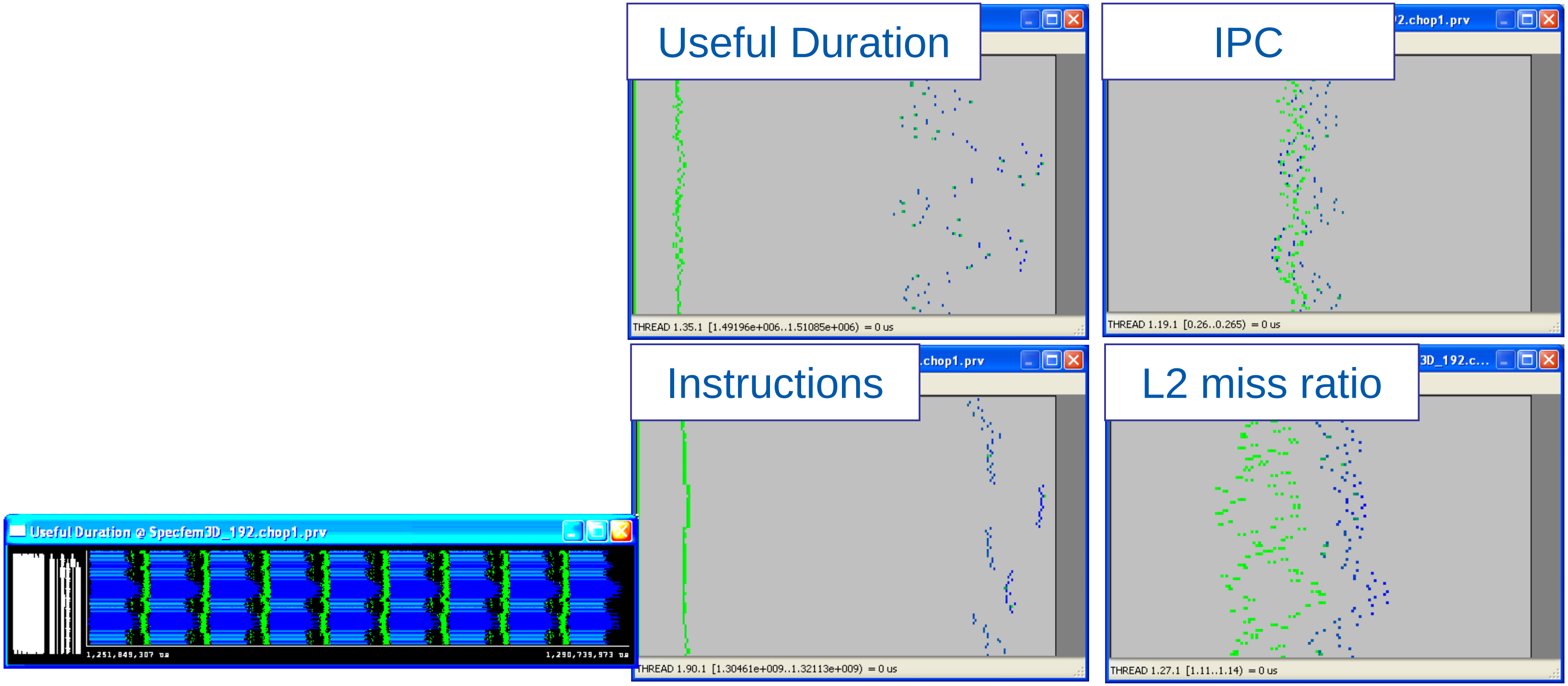
2/3D tables (Statistics)

Goal = Flexibility
- No semantics
- Programmable

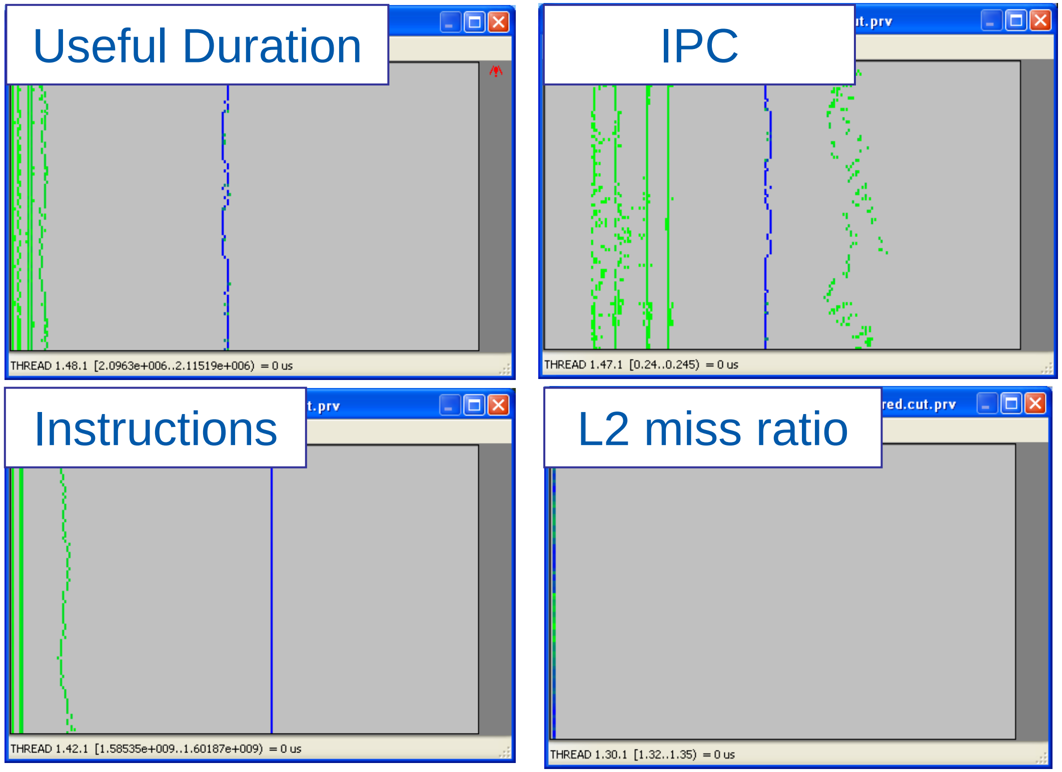
Configuration files
- Distribution
- Your own

Comparative analyses
- Multiple traces
- Synchronize scales

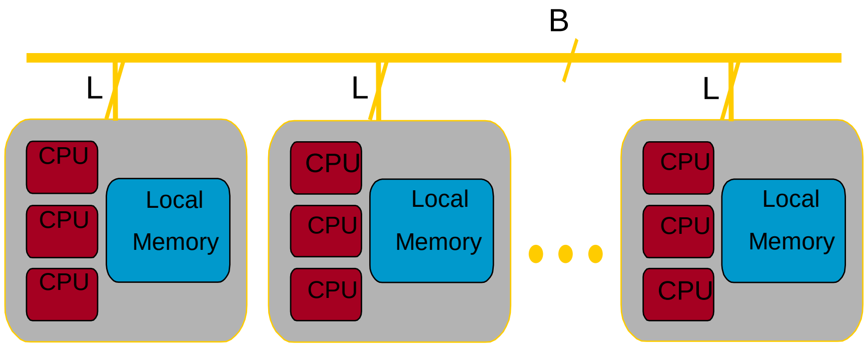
- View and measure to understand execution performance
  - Example: Imbalance in computation due to
    - IPC imbalance related to L2 cache misses – check memory access
    - Instructions imbalance – redistribute work



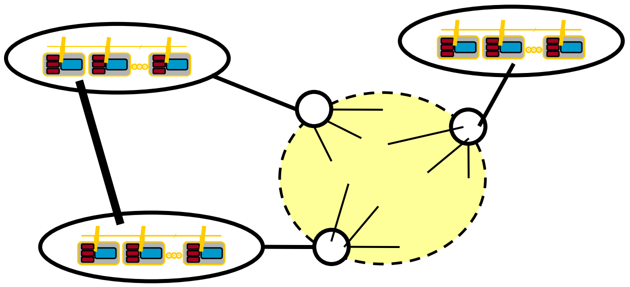
- View and measure to understand execution performance
  - Example: 6 months later



- Reads & writes Paraver traces
- Key factors influencing performance
  - Abstract architecture
  - Basic MPI protocols
  - No attempt to model details
- Objectives
  - Simple / general
  - Fast simulations
- Linear components
  - Point2point
  - CPU/block speed
- Non-linear components
  - Synchronization
  - Resources contention
- Network of SMPs / GRID



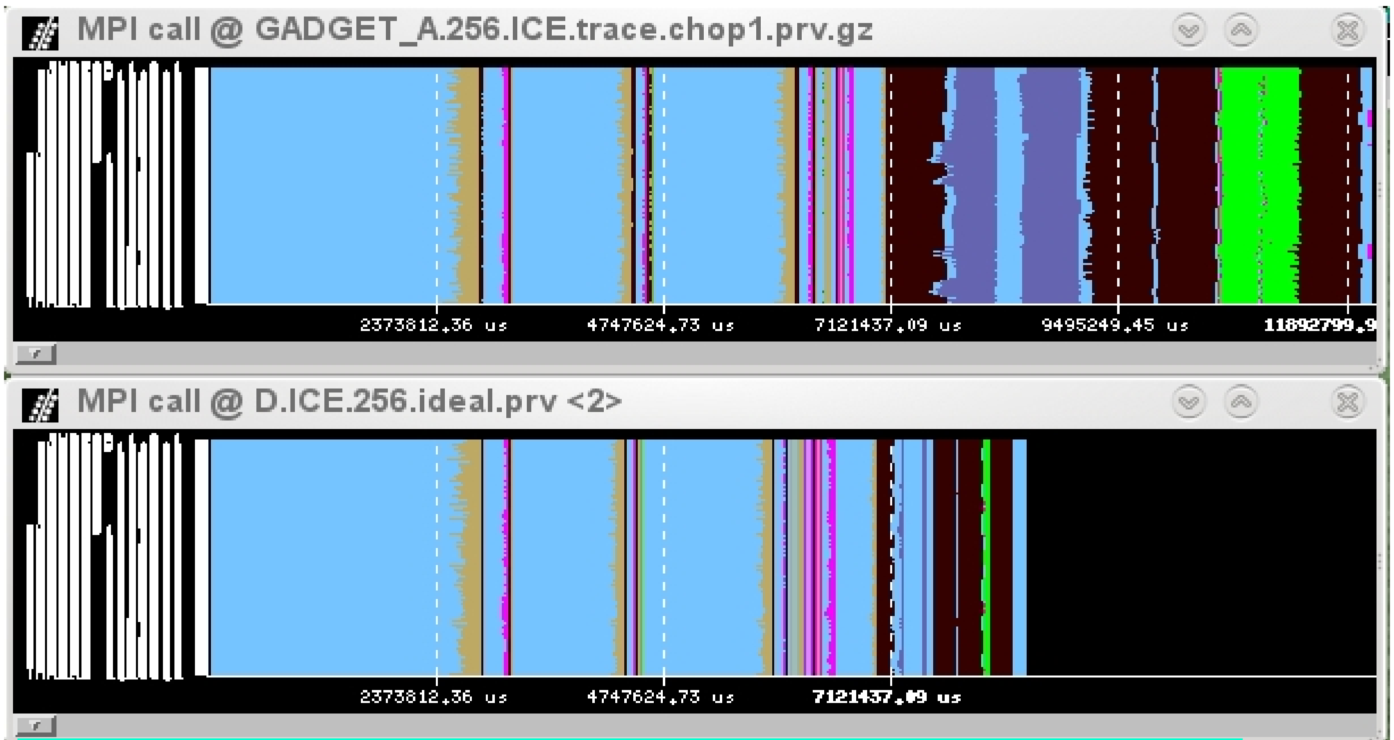
$$T = \frac{MessageSize}{BW} + L$$



Dimemas – message passing simulator

- Predictions to find application limits

Real run



Ideal network: infinite bandwidth, no latency

Time in MPI with ideal network caused by serializations of the application

Extended VI-HPS tools suite

Extended VI-HPS Tool Suite

- parallel performance frameworks (**O|SS**, **TAU**)
- performance analysis data-mining (**PerfExplorer**)
- parallel execution parametric studies (**Dimemas**)
- cache usage analysis (**kcachegrind**)
- assembly code optimization (**MAQAO**)
- process mapping generation/optimization (**Rubik**)

- parallel file I/O optimization (**SIONlib**)
- PMPI tools virtualization (P^NMPI)
- component-based tools framework (**CBTF**)

Open|SpeedShop Tool Set

- Open Source Performance Analysis Tool Framework
  - Most common performance analysis steps ***all in one tool***
  - Combines ***tracing*** and ***sampling*** techniques
  - ***Extensible*** by plugins for data collection and representation
  - Gathers and displays several types of performance information
- Flexible and Easy to use
  - User access through:
    GUI, ***Command Line***, ***Python Scripting, convenience scripts***
- Several Instrumentation Options
  - All work on ***unmodified application binaries***
  - ***Offline*** and ***online data collection*** / ***attach*** to running codes
- Supports a wide range of systems
  - Extensively used and tested on a variety of ***Linux clusters***
  - New: ***Cray XT/XE/XK*** and ***Blue Gene P/Q*** support

Central Concept: Performance Experiments

- Users pick experiments:
  - What to measure and from which sources?
  - How to select, view, and analyze the resulting data?
- Two main classes:
  - Statistical Sampling
    - Periodically interrupt execution and record location
    - Useful to get an overview
    - Low and uniform overhead
  - Event Tracing
    - Gather and store individual application events
    - Provides detailed per event information
    - Can lead to huge data volumes
- O|SS can be extended with additional experiments

Sampling Experiments in O|SS

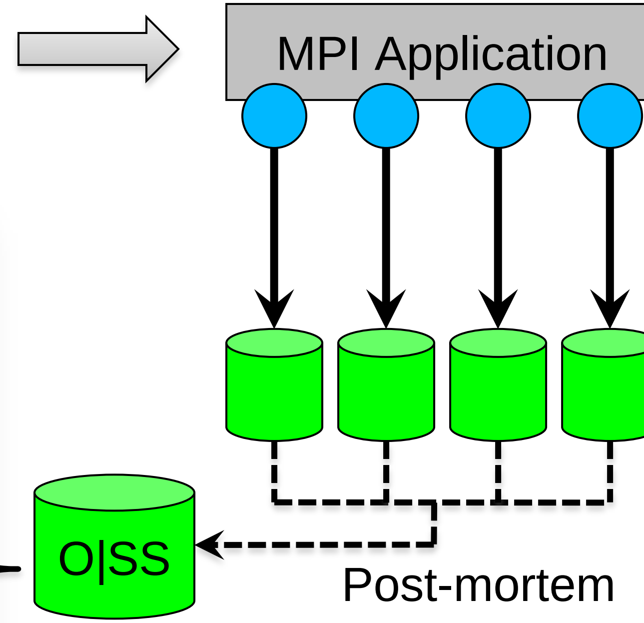
- PC Sampling (pcsamp)
  - Record PC repeatedly at user defined time interval
  - Low overhead overview of time distribution
  - Good first step, lightweight overview
- Call Path Profiling (usertime)
  - PC Sampling and Call stacks for each sample
  - Provides inclusive and exclusive timing data
  - Use to find hot call paths, whom is calling who
- Hardware Counters (hwc, hwctime, hwcsamp)
  - Access to data like cache and TLB misses
  - hwc, hwctime:
    - Sample a HWC event based on an event threshold
    - Default event is PAPI_TOT_CYC overflows
  - hwcsamp:
    - Periodically sample up to 6 counter events based (hwcsamp)
    - Default events are PAPI_FP_OPS and PAPI_TOT_CYC

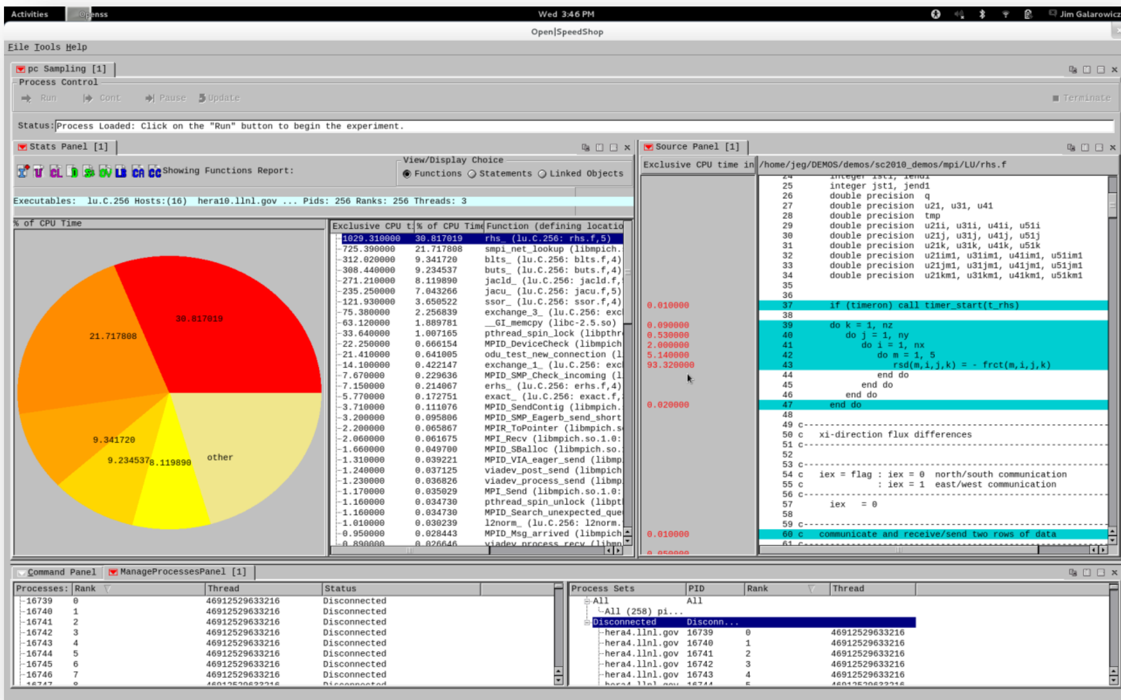
- Input/Output Tracing (io, iop, iot)
  - Record invocation of all POSIX I/O events
  - Provides aggregate and individual timings
  - Lightweight I/O profiling (iop)
  - Store function arguments and return code for each call (iot)
- MPI Tracing (mpi, mpit, mpiotf)
  - Record invocation of all MPI routines
  - Provides aggregate and individual timings
  - Store function arguments and return code for each call (mpit)
  - Create Open Trace Format (OTF) output (mpiotf)
- Floating Point Exception Tracing (fpe)
  - Triggered by any FPE caused by the application
  - Helps pinpoint numerical problem areas

- O|SS supports MPI and threaded codes
  - Automatically applied to all tasks/threads
  - Default views aggregate across all tasks/threads
  - Data from individual tasks/threads available
  - Thread support (incl. OpenMP) based on POSIX threads
- Specific parallel experiments (e.g., MPI)
  - Wraps MPI calls and reports
    - MPI routine time
    - MPI routine parameter information
  - The mpit experiment also store function arguments and return code for each call
- Specialized views
  - Load balance information (min/mean/max process)
  - Cluster analysis to detect common tasks

Open | SpeedShop™ 101

osspcsamp “srun –n4 –N1 smg2000 –n 65 65 65”





http://www.openspeedshop.org/

Example Run with Output

- osspcsamp “mpirun –np 2 smg2000 –n 65 65 65” (1/2)

```
Bash> osspcsamp "mpirun -np 2 ./smg2000 -n 65 65 65"
[openss]: pcsamp experiment using the pcsamp experiment default sampling rate: "100".
[openss]: Using OPENSS_PREFIX installed in /opt/OSS-mrnet
[openss]: Setting up offline raw data directory in /tmp/jeg/offline-oss
[openss]: Running offline pcsamp experiment using the command:
"mpirun -np 2 /opt/OSS-mrnet/bin/ossrun "./smg2000 -n 65 65 65" pcsamp"

Running with these driver parameters:
 (nx, ny, nz)    = (65, 65, 65)
 …
               <SMG native output>
…
Final Relative Residual Norm = 1.774415e-07
[openss]: Converting raw data from /tmp/jeg/offline-oss into temp file X.0.openss

Processing raw data for smg2000
Processing processes and threads ...
Processing performance data ...
Processing functions and statements …
```

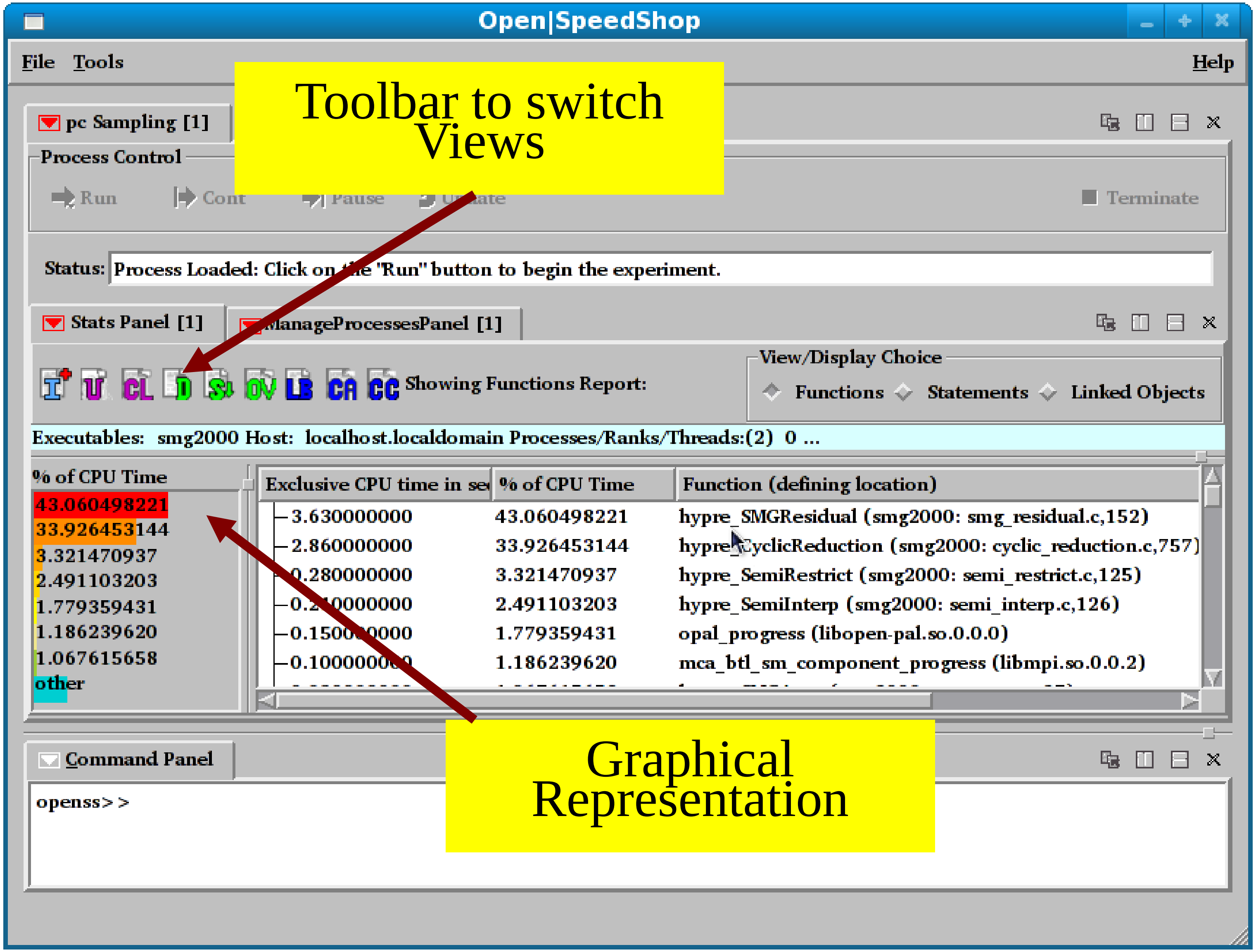
Example Run with Output

- osspcsamp “mpirun –np 2 smg2000 –n 65 65 65” (2/2)

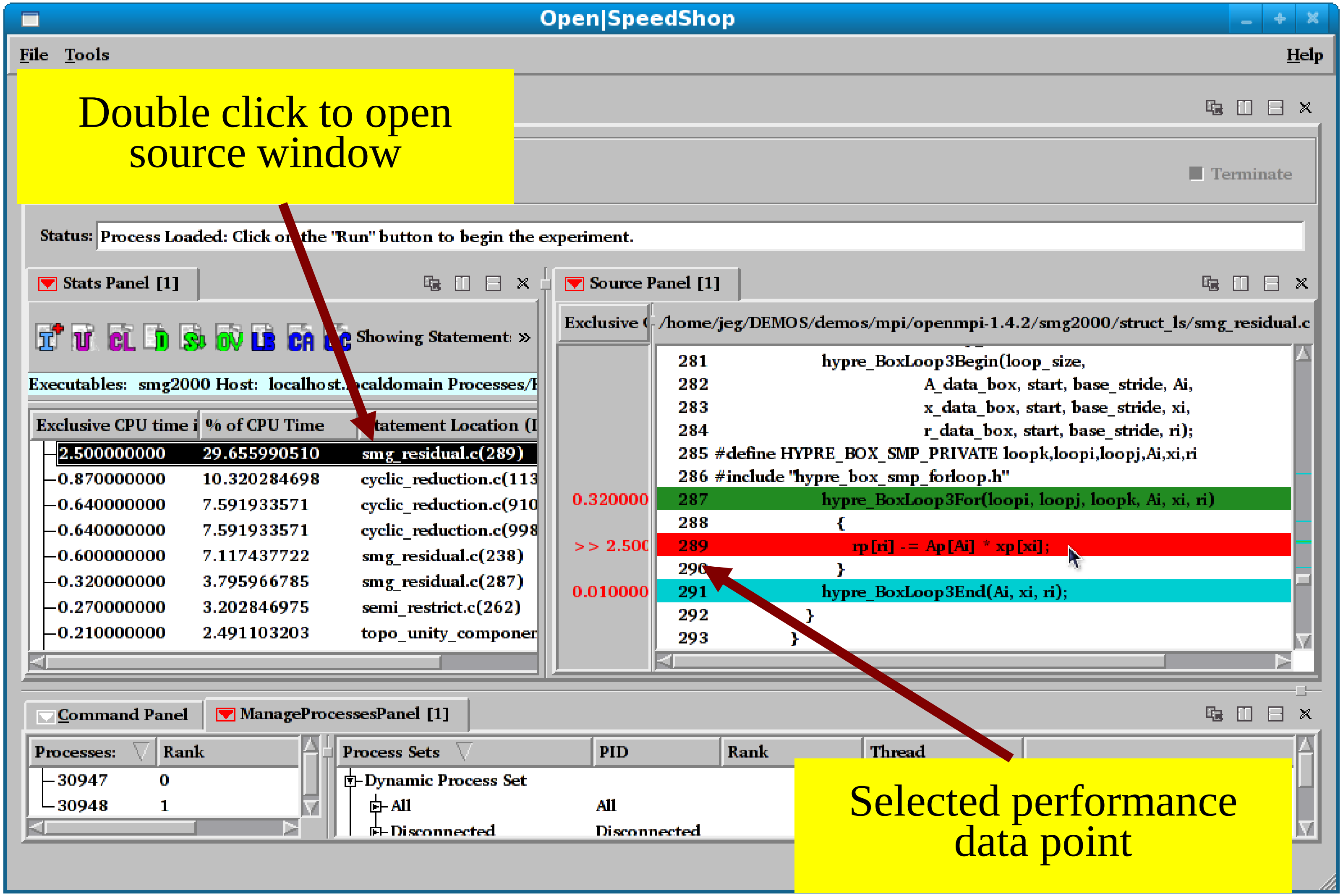
```
[openss]: Restoring and displaying default view for:
          /home/jeg/DEMOS/demos/mpi/openmpi-1.4.2/smg2000/test/smg2000-pcsamp-1.openss
[openss]: The restored experiment identifier is:  -x 1

Exclusive CPU time      % of CPU Time  Function (defining location)
    in seconds.
    3.630000000        43.060498221  hypre_SMGResidual (smg2000: smg_residual.c,152)
    2.860000000        33.926453144  hypre_CyclicReduction (smg2000: cyclic_reduction.c,757)
    0.280000000         3.321470937  hypre_SemiRestrict (smg2000: semi_restrict.c,125)
    0.210000000         2.491103203  hypre_SemiInterp (smg2000: semi_interp.c,126)
    0.150000000         1.779359431  opal_progress (libopen-pal.so.0.0.0)
    0.100000000         1.186239620 mca_btl_sm_component_progress (libmpi.so.0.0.2)
    0.090000000         1.067615658  hypre_SMGAxpy (smg2000: smg_axpy.c,27)
    0.080000000         0.948991696  ompi_generic_simple_pack (libmpi.so.0.0.2)
    0.070000000         0.830367734  __GI_memcpy (libc-2.10.2.so)
    0.070000000         0.830367734 hypre_StructVectorSetConstantValues (smg2000: struct_vector.c,537)
    0.060000000         0.711743772  hypre_SMG3BuildRAPSym (smg2000: smg3_setup_rap.c,233)
```

- View with GUI: openss –f smg2000-pcsamp-1.openss

Default Output Report View



Associate Source & Performance Data



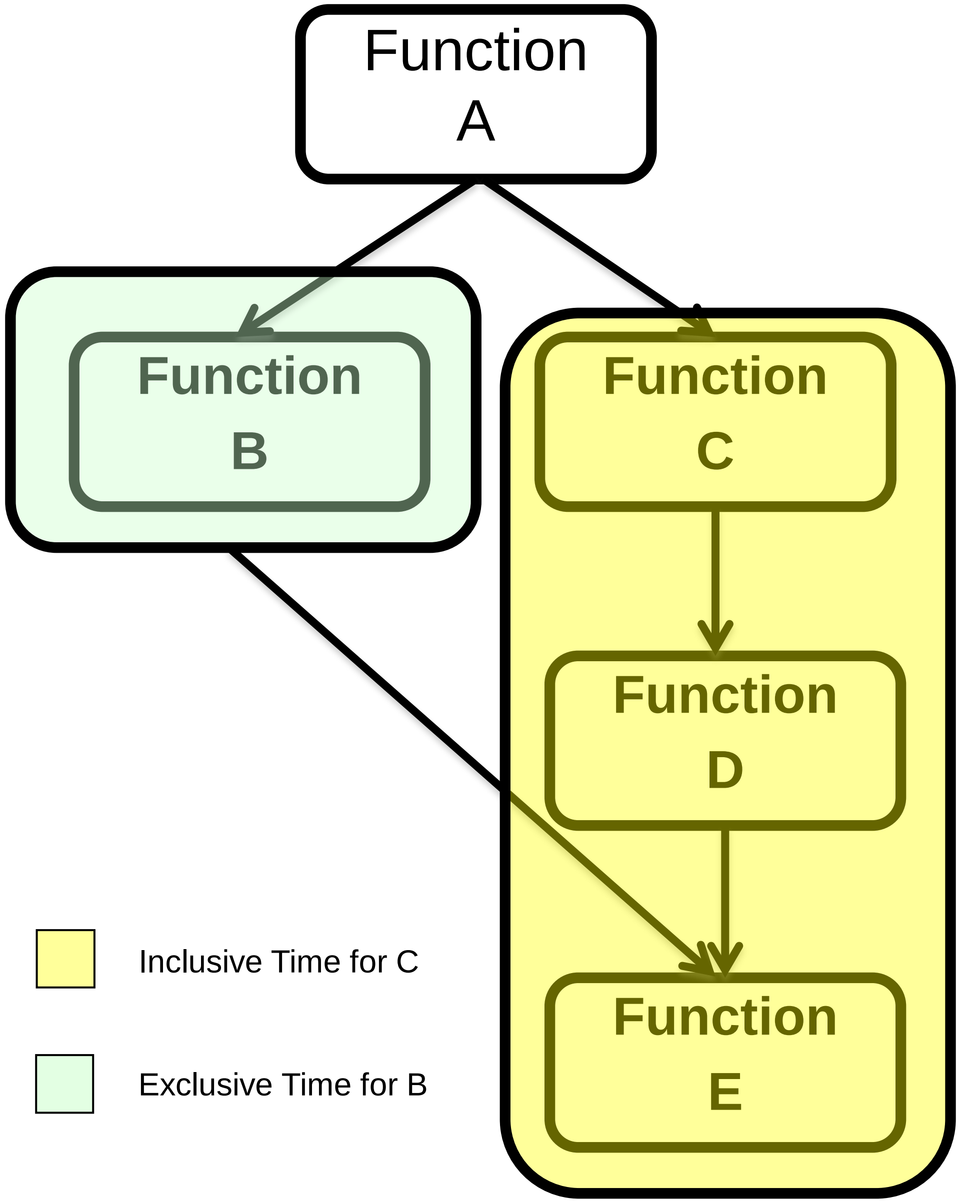
Alternative Interfaces

- Scripting language
  - Immediate command interface
  - O|SS interactive command line (CLI)
- Python module

```
Experiment Commands
    expAttach
    expCreate
    expDetach
    expGo
    expView

List Commands
    list -v exp
    list -v hosts
```

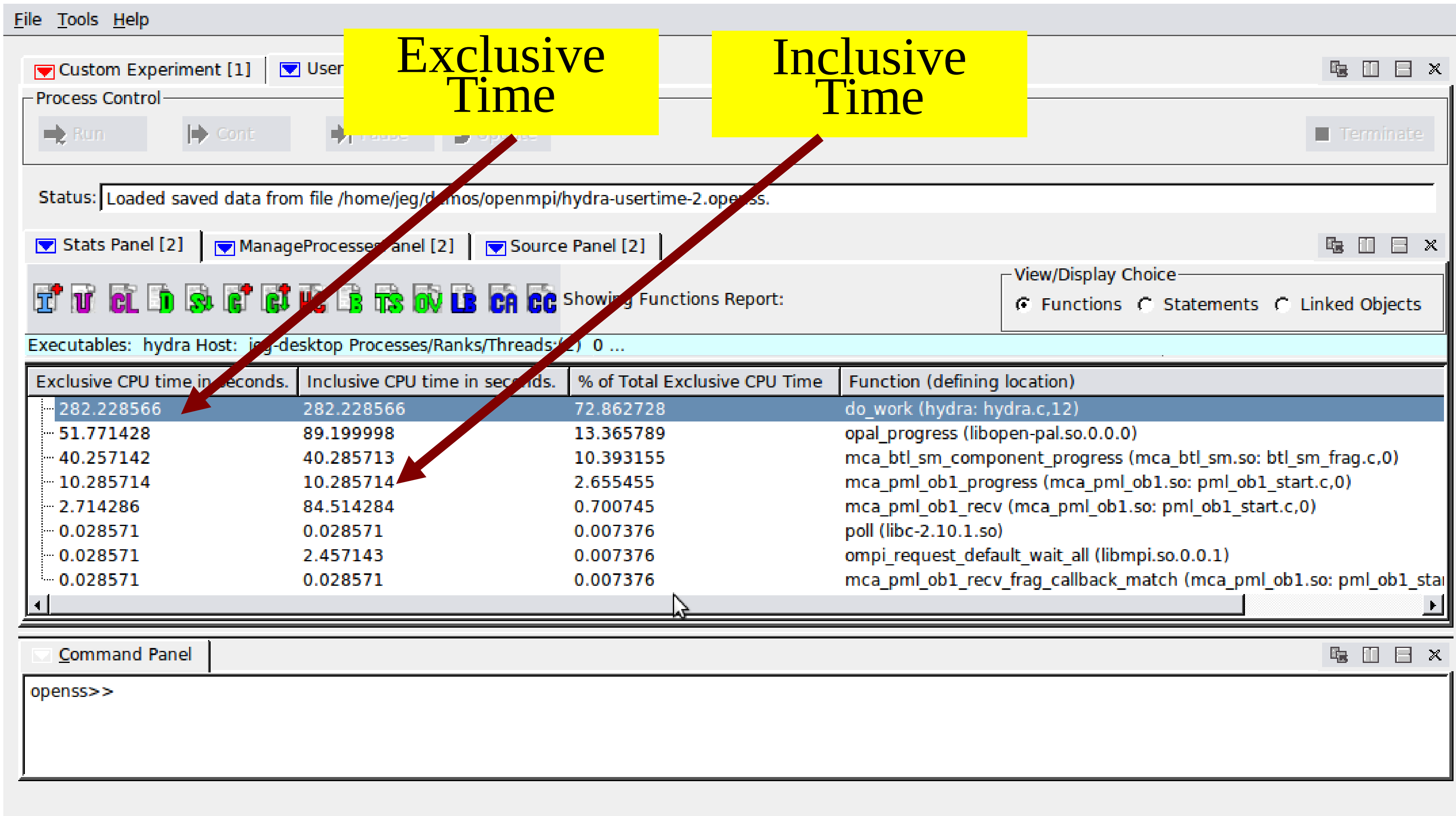
```
import openss

my_filename=openss.FileList("myprog.a.out")
my_exptype=openss.ExpTypeList("pcsamp")
my_id=openss.expCreate(my_filename,my_exptype)

openss.expGo()

My_metric_list = openss.MetricList("exclusive")
my_viewtype = openss.ViewTypeList("pcsamp”)
result = openss.expView(my_id,my_viewtype,my_metric_list)
```

Inclusive vs. Exclusive Timing

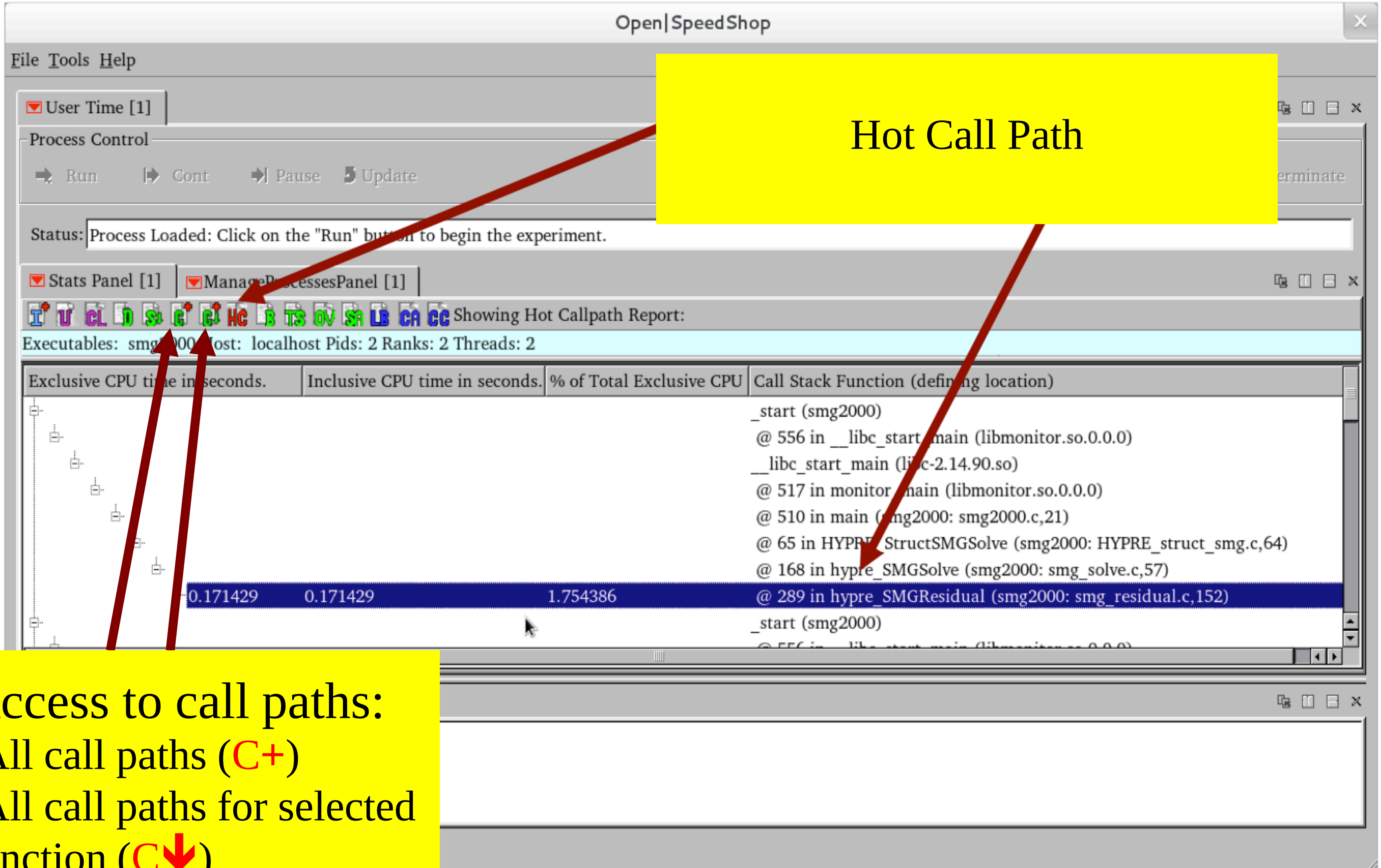


- **Usertime Experiment**
  - Gather stack traces for each sample
- **Enable calculation of inclusive/exclusive times**
  - Time spent inside a function only (exclusive)
    - See: Function B
  - Time spent inside a function and its children (inclusive)
    - See Function C and children
- **Tradeoffs**
  - Pro: Obtain additional context information
  - Con: Higher overhead/lower sampling rate

- Default View
  - Similar to pcsamp view from first example



Stack Trace Views: Hot Call Path



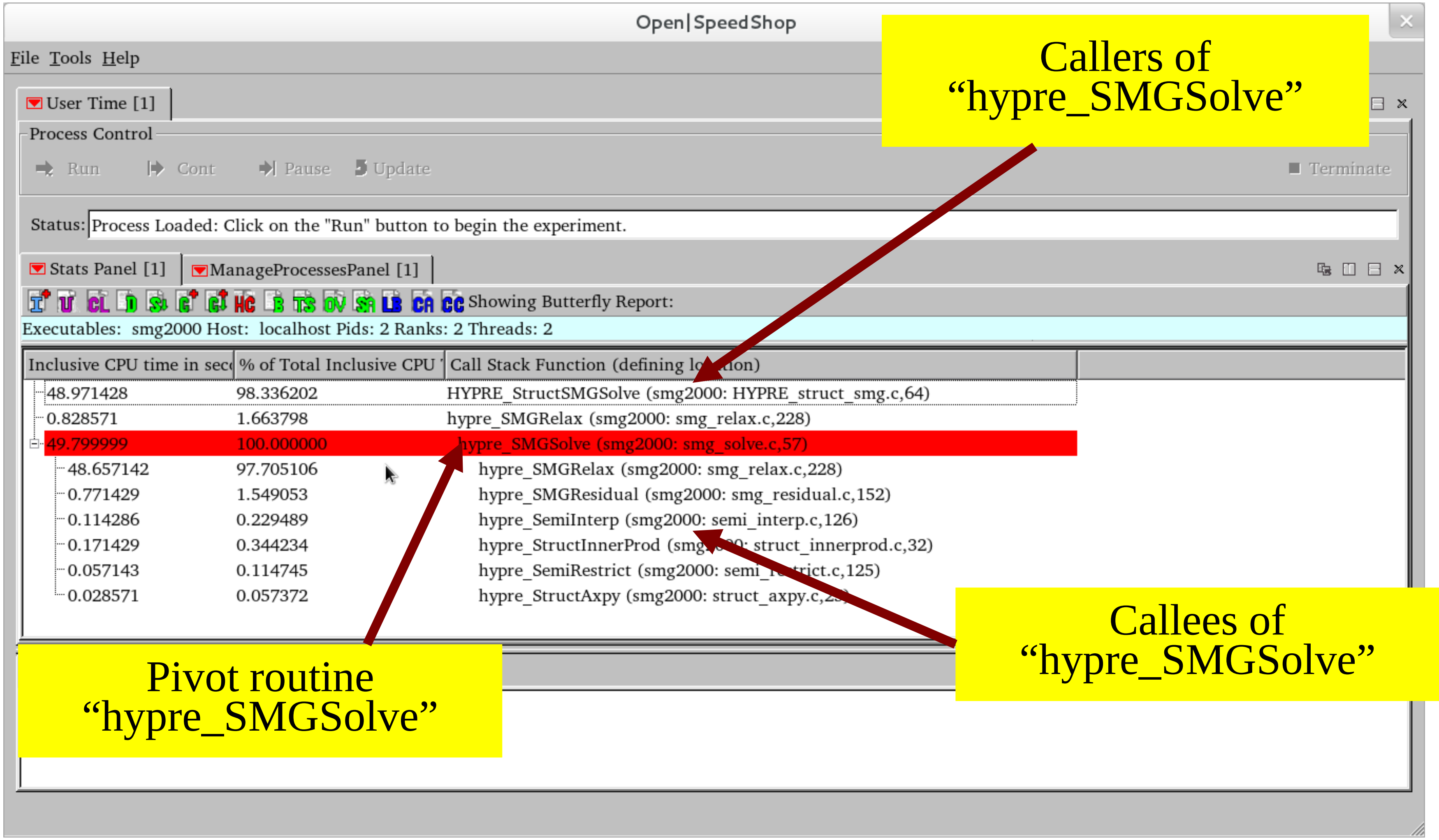
Hot Call Path

Access to call paths:
- All call paths (C+)
- All call paths for selected function (C↓)

Interpreting Call Context Data

- Inclusive versus exclusive times
  - If similar: child executions are insignificant
    - May not be useful to profile below this layer
  - If inclusive time significantly greater than exclusive time:
    - Focus attention to the execution times of the children
- Hotpath analysis
  - Which paths takes the most time?
- Butterfly analysis (similar to gprof)
  - Should be done on “suspicious” functions
    - Functions with large execution time
    - Functions with large difference between implicit and explicit time
    - Functions of interest
    - Functions that “take unexpectedly long”
  - Shows split of time in callees and callers

Stack Trace Views: Butterfly View



Comparing Performance Data

- Key functionality for any performance analysis
  - Absolute numbers often don’t help
  - Need some kind of baseline / number to compare against
- Typical examples
  - Before/after optimization
  - Different configurations or inputs
  - Different ranks, processes or threads
- Open|SpeedShop includes support to line up profiles
  - Perform multiple experiments and create multiple databases
  - Script to load all experiments and create multiple columns
- Advanced functionality in GUI
  - Arbitrary number of columns with data to compare
  - Use “CC” (Custom Comparison) button

Summary: Basic Steps

- Place **the way you run your application normally** in quotes and pass it as an argument to osspcsamp
  - Similar for any of the other experiment
  - osspcsamp “srun –N 8 –n 64 ./mpi_application app_args”
- Open|SpeedShop sends a summary profile to stdout
- Open|SpeedShop creates a database file
- Display alternative views of the data with the GUI via:
  - openss –f <database file>
- Display alternative views of the data with the CLI via:
  - openss –cli –f <database file>
- On clusters, need to set OPENSS_RAWDATA_DIR
  - Should point to a directory in a shared file system
  - More on this later – usually done in a module or dotkit file.
- Start with pcsamp for overview of performance
- Then home into performance issues with other experiments

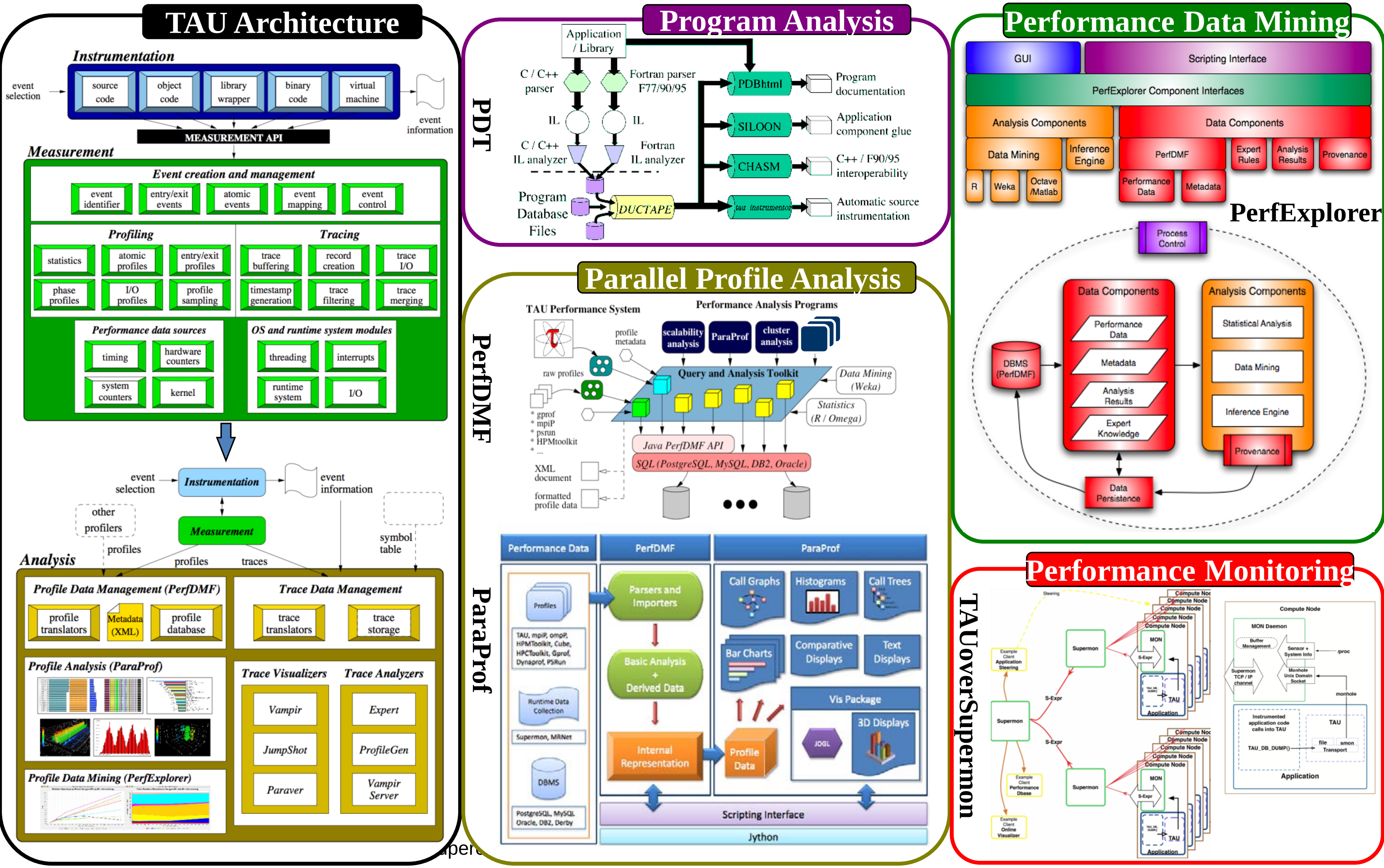
Summary: Digging Deeper

- Multiple interfaces
  - GUI for easy display of performance data
  - CLI makes remote access easy
  - Python module allows easy integration into scripts
- Dedicated views for parallel executions
  - Load balance view
  - Use custom comparison to compare ranks or threads
- Usertime experiments provide inclusive/exclusive times
  - Time spent inside a routine vs. its children
  - Key view: butterfly
- Comparisons
  - Between experiments to study improvements/changes
  - Between ranks/threads to understand differences/outliers

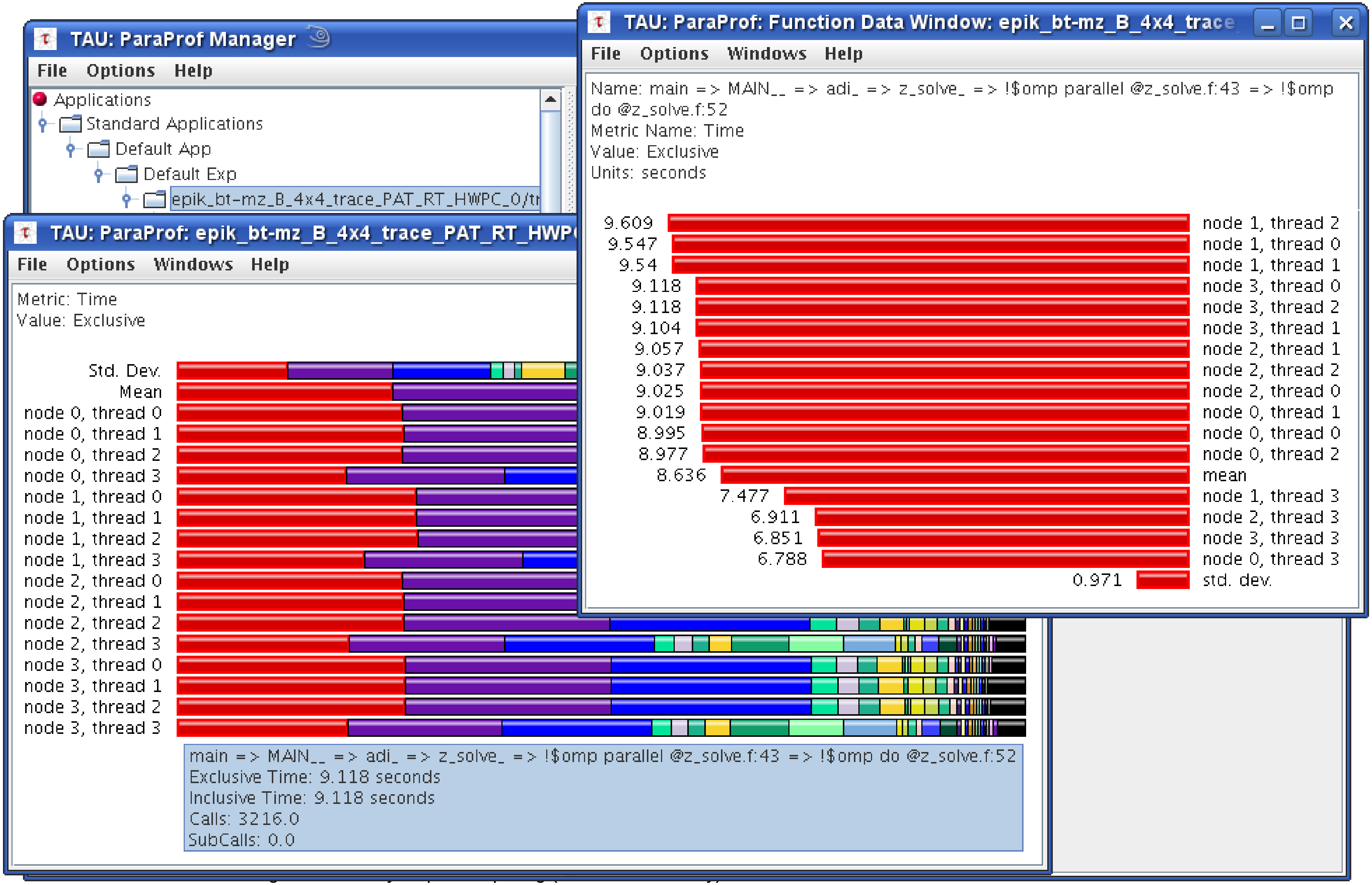
- Integrated performance toolkit
  - Instrumentation, measurement, analysis & visualization
    - Highly customizable installation, API, envvars & GUI
    - Supports multiple profiling & tracing capabilities
  - Performance data management & data mining
  - Targets all parallel programming/execution paradigms
    - Ported to a wide range of computer systems
  - Performance problem solving framework for HPC
  - Extensive bridges to/from other performance tools
    - PerfSuite, Scalasca, Vampir, ...
- Developed by U. Oregon/PRL
  - Broadly deployed open-source software
  - http://tau.uoregon.edu/

TAU Performance System components

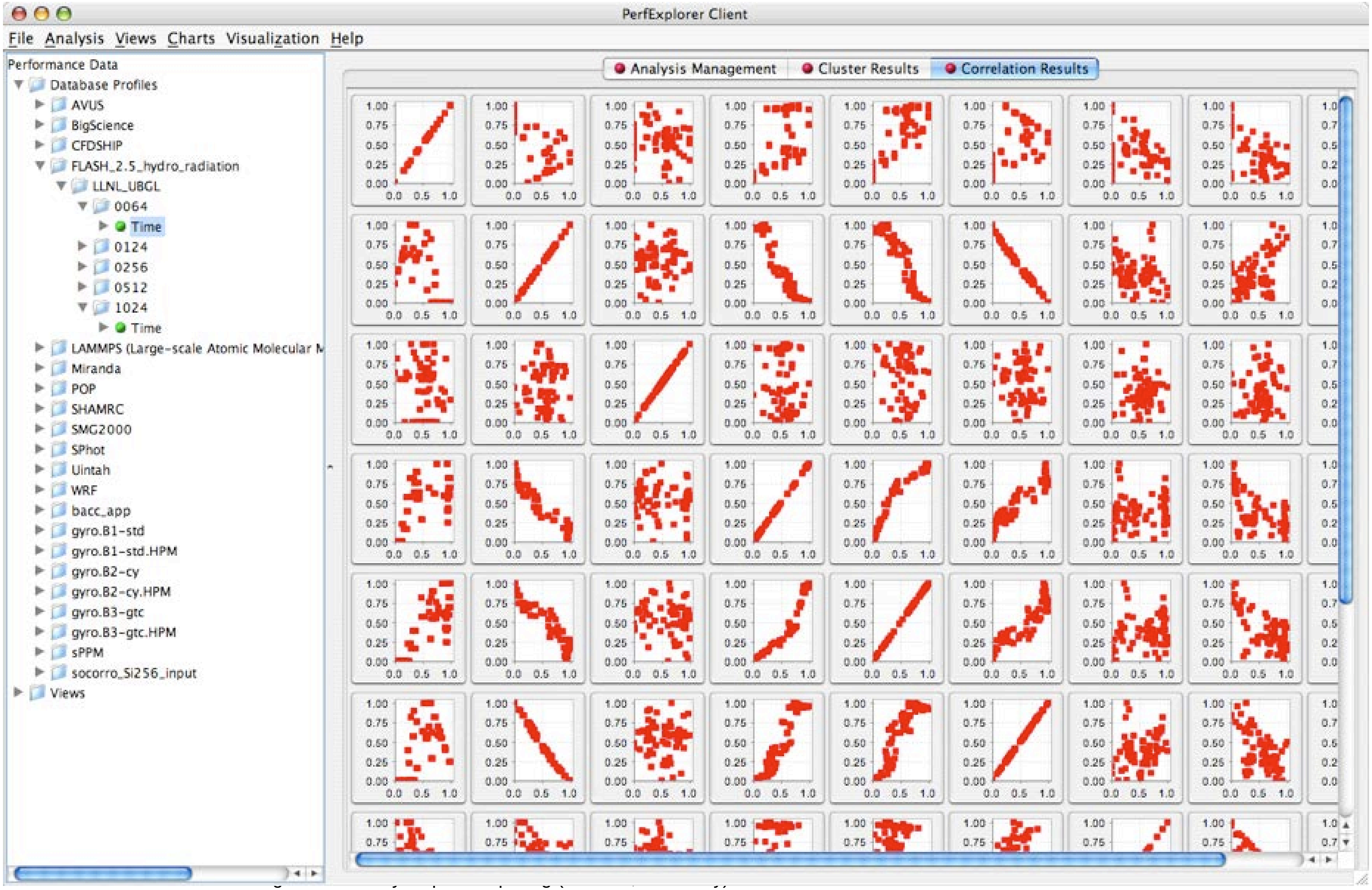
VI-HPS



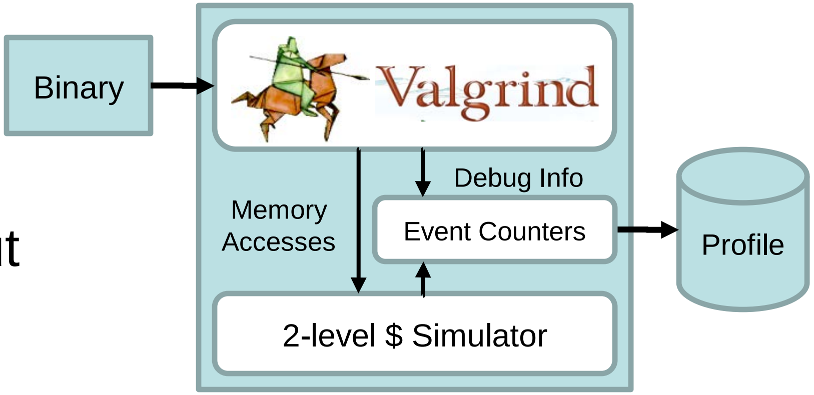
TAU ParaProf GUI displays (selected)



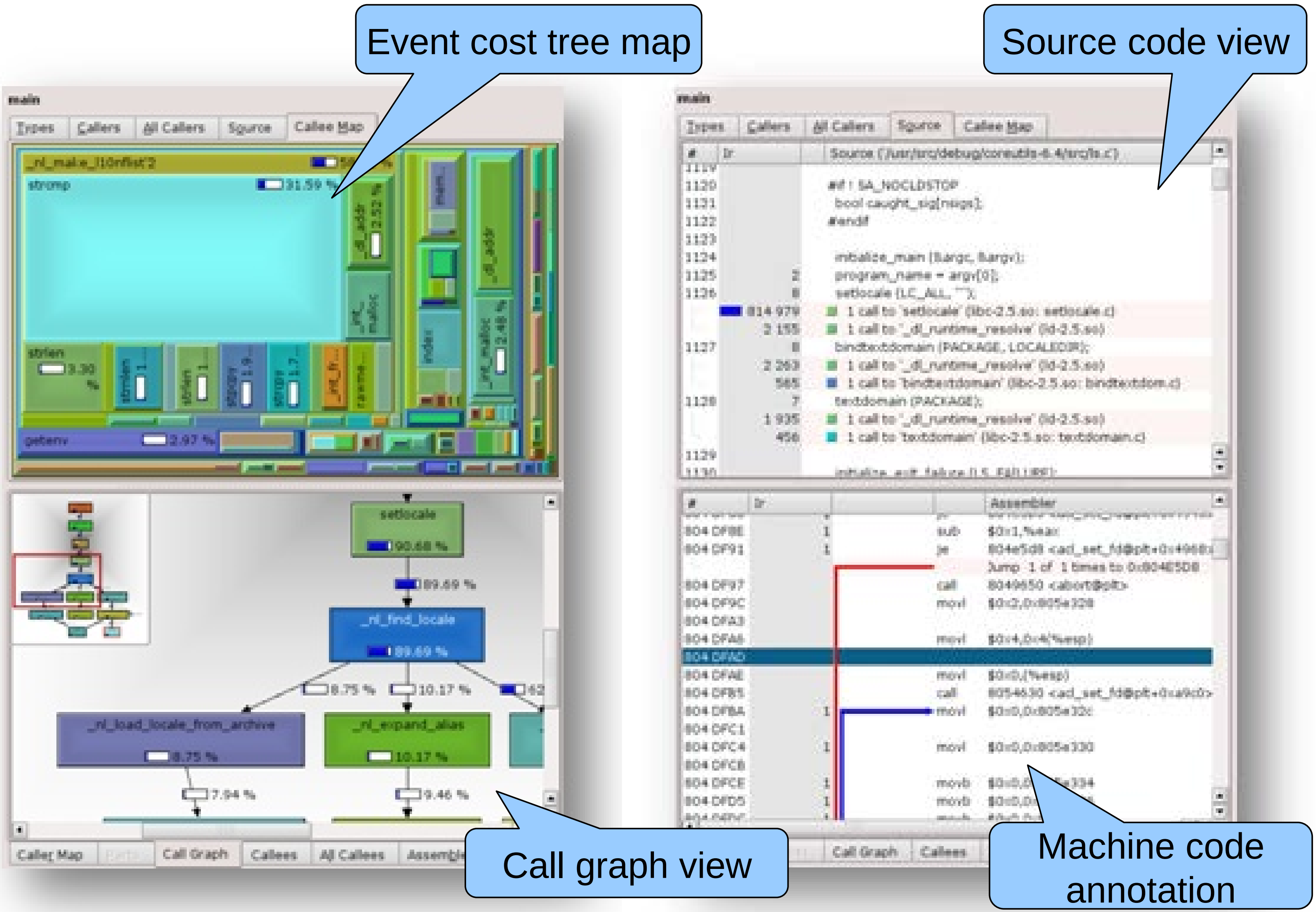
TAU PerfExplorer data mining



- Cachegrind: cache analysis by simple cache simulation
  - Captures dynamic callgraph
  - Based on valgrind dynamic binary instrumentation
  - Runs on x86/PowerPC/ARM unmodified binaries
    - No root access required
  - ASCII reports produced
- [KQ]Cachegrind GUI
  - Visualization of cachegrind output
- Developed by TU Munich
  - Released as GPL open-source
  - http://kcachegrind.sf.net/



KCachegrind GUI



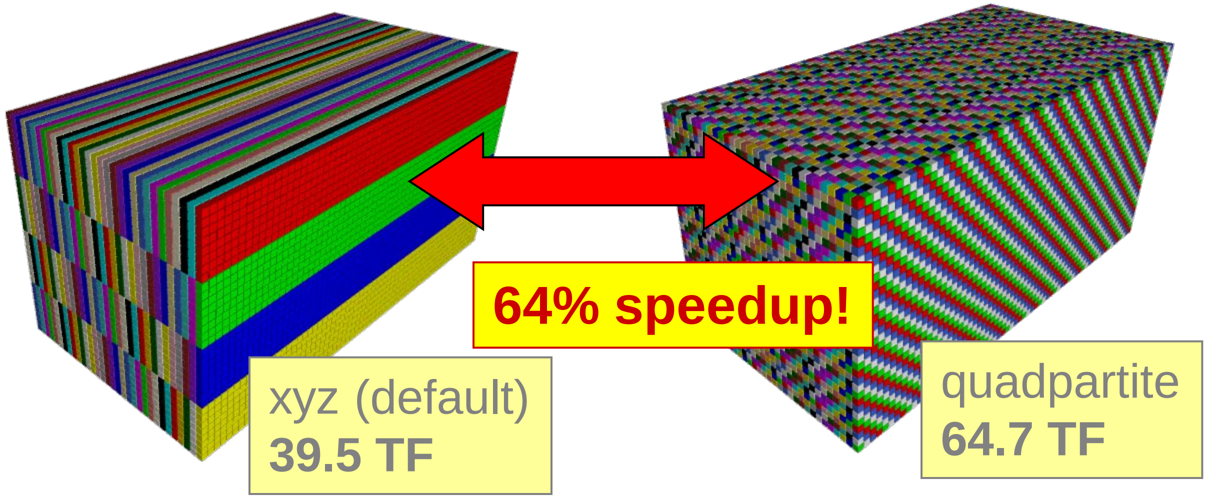
MAQAO

- Modular Assembler Quality Analyzer & Optimizer
  - Framework for binary manipulation
    - using plugins and scripting language
  - Tool exploiting framework to produce reports
    - fast prototyping and batch interface
  - STAN static performance model

Challnges with mpiP at Scale

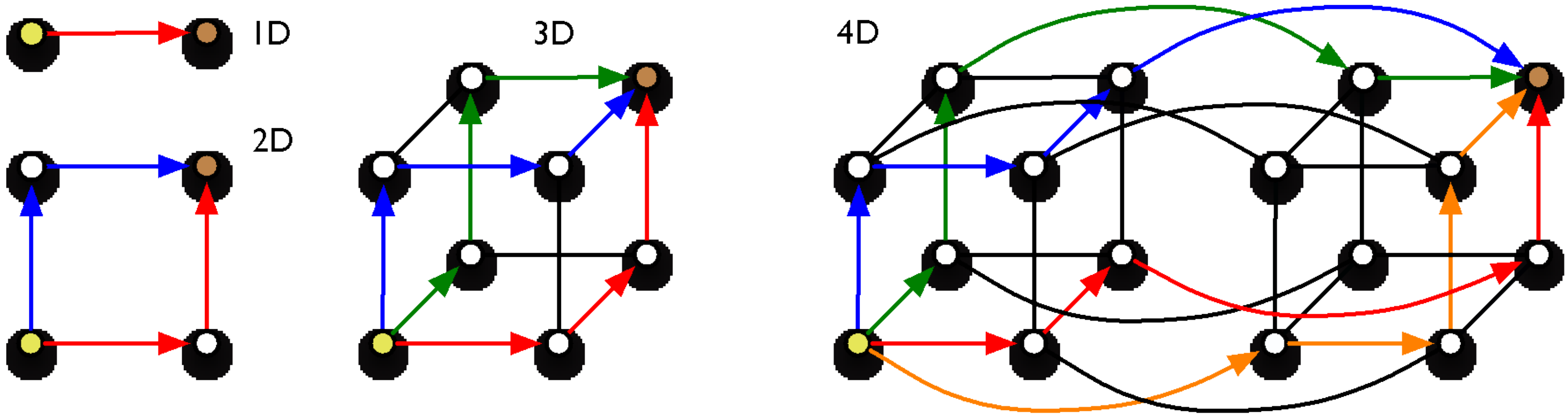
  - MIL instrumentation language for dynamic analysis
    - building custom performance evaluation tools using HWCs
    - instrumentation of functions, loops, blocks & instructions
- Developed by UVSQ Exascale Computing Research lab
  - Supports Intel x86_64 microarchitecture
  - Available from www.maqao.org

- **Network topologies getting more complex**
  - Interactions with communication topology non-trivial
  - Node placement has huge impact on performance



- **Require tools to help with defining layouts**
  - Easier specification and visualization of layouts
  - Capture basic optimization steps at an abstract level

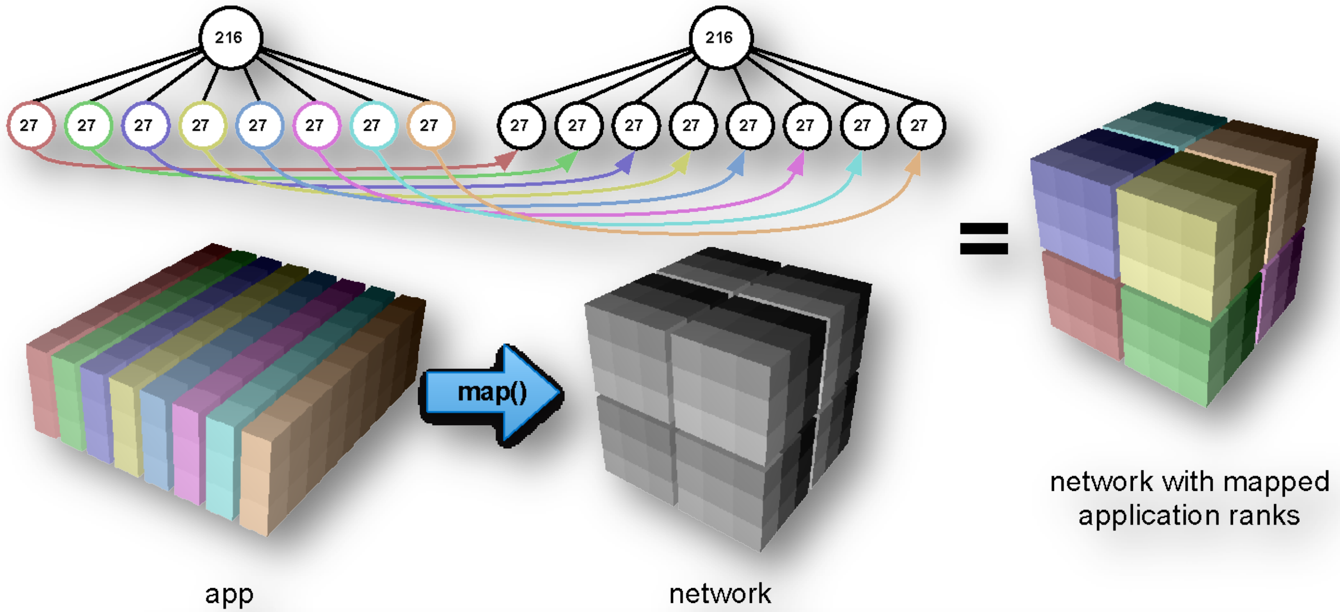
Utilizing the Full Capacity of the Torus



Black links are “spare” links that can handle extra traffic that comes through the cube.

- **Dimension independent transformations/tilting**
  - Tilting optimization allows higher bandwidth on torus links
  - Tilting is easily extended into higher dimensions (5D, etc.)

Rubik: Easy Generation of BG Mapping Files

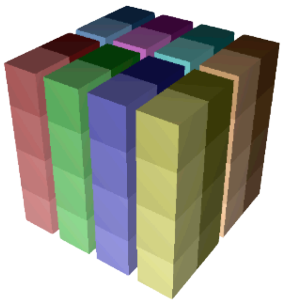



```
# Create app partition tree of 27-task planes
app = box([9,3,8])
app.tile([9,3,1])

# Create network partition tree of 27-task cubes
network = box([6,6,6])
network.tile([3,3,3])

network.map(app)   # Map plane tasks into cubes
```

http://scalability.llnl.gov/

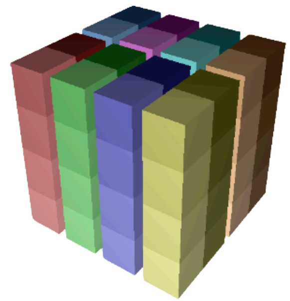
Additional Rubik Operations

div



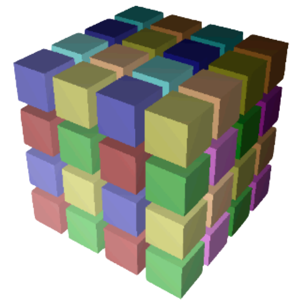
```
app = box([4,4,4])
app.div([2,1,4])
```

tile



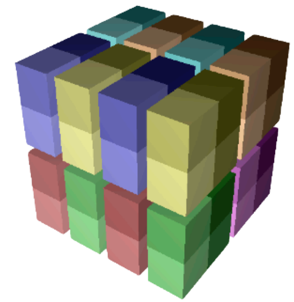
```
app = box([4,4,4])
app.tile([2,4,1])
```

mod



```
app = box([4,4,4])
app.mod([2,2,2])
```

cut

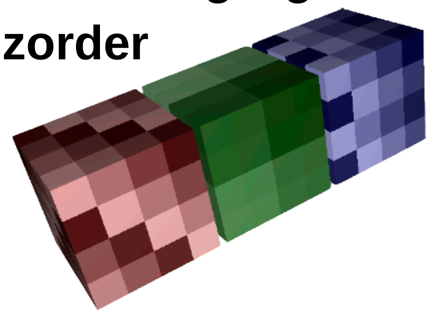


```
app = box([4,4,4])
app.cut([2,2,2],
        [div,div,mod])
```

zigzag

zorder

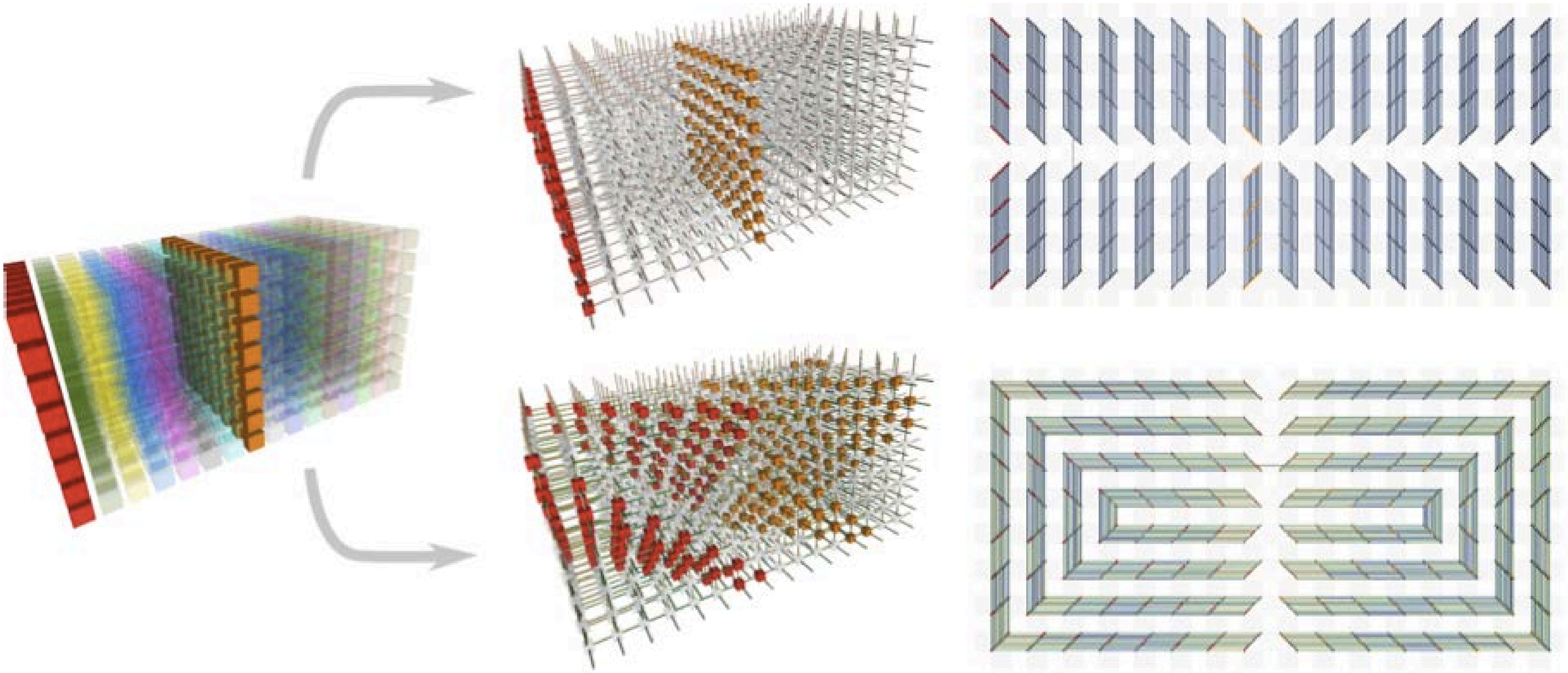
tilt



```
Z, Y, X = 0, 1, 2
net = box([12,4,4])
net.div([3,1,1])
net[0,0,0].tilt(Z,X,1)
net[0,0,0].tilt(X,Y,1)
net[1,0,0].zorder()
net[2,0,0].zigzag(Z,X,1)
net[2,0,0].zigzag(X,Y,1)
```

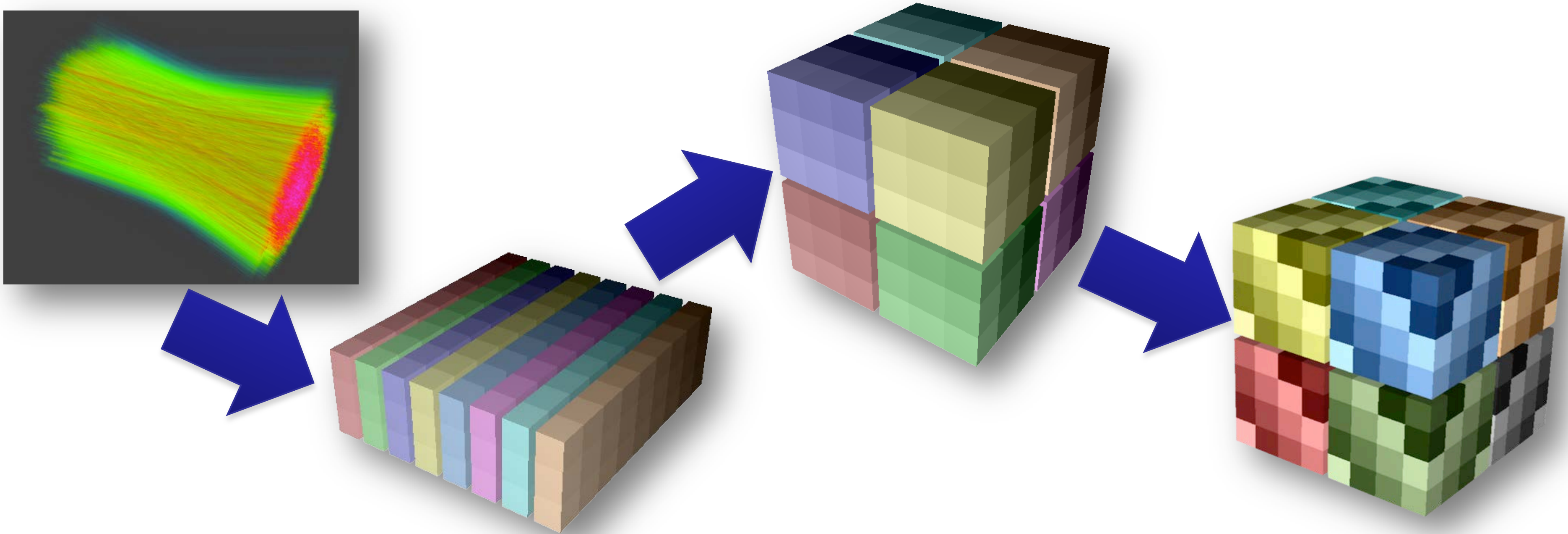
Example: an LLNL Laser Code

- **Problems setup as a series of 2D slabs**
  - During each step: X/Y phases within a slab
  - Looking at realized bandwidth for each step



Mappings for the Laser Code

- Improved bandwidth from **50 MB/s to over 201 MB/s**
- Can be implemented as a single, short Python script
- Works for any dimensionality
- Integrated visualization of mappings (for 3D)



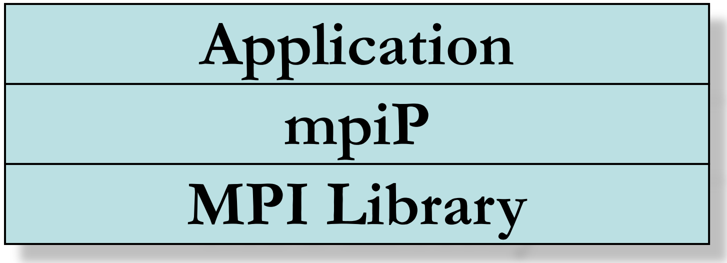
VI-HPS component technologies

- Key tool components also provided as open-source
  - Program development & system environment
    - Eclipse PTP ETFw, SysMon
  - Program/library instrumentation
    - COBI, OPARI, PDToolkit
  - Runtime measurement systems
    - P^nMPI, UniMCI
  - Scalable optimized file I/O
    - SIONlib
  - Libraries & tools for handling (and converting) traces
    - EPILOG, OTF, PEARL
  - Component Based Tool Framework (CBTF)
    - Communication framework to create custom tools

- Portable native parallel I/O library & utilities
  - Scalable massively-parallel I/O to task-local files
  - Manages single or multiple physical files on disk
    - optimizes bandwidth available from I/O servers by matching blocksizes/alignment, reduces metadata-server contention
  - POSIX-I/O-compatible sequential & parallel API
    - adoption requires minimal source-code changes
  - Tuned for common parallel filesystems
    - GPFS (BlueGene), Lustre (Cray), ...
  - Convenient for application I/O, checkpointing,
    - Used by Scalasca tracing (when configured)
- Developed by JSC
  - Available as open-source from
  - http://www.fz-juelich.de/jsc/sionlib/

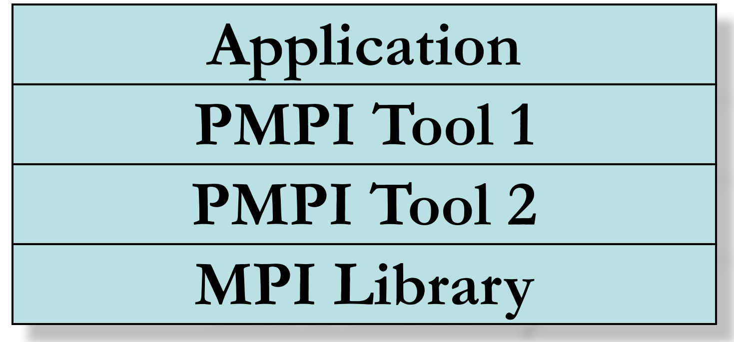
P^{N}MPI: Virtualizing PMPI Tools

- PMPI interception of MPI calls
  - Easy to include in applications
  - Limited to a single tool

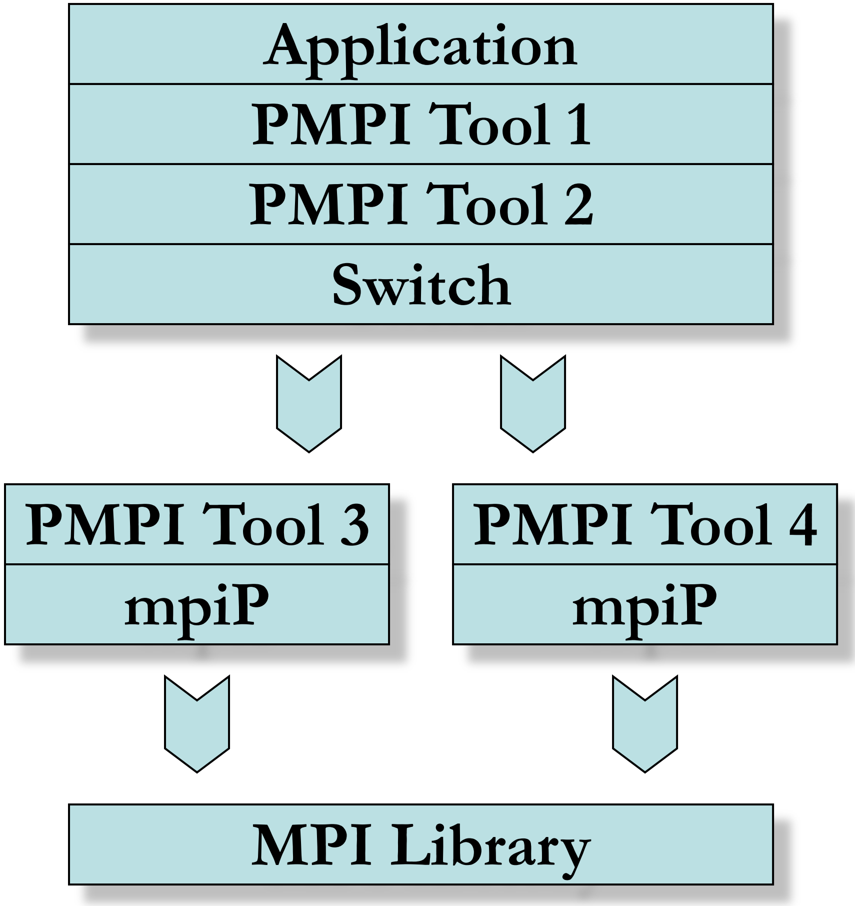


P^NMPI: Virtualizing PMPI Tools

- PMPI interception of MPI calls
  - Easy to include in applications
  - Limited to a single tool
- P^NMPI virtualized PMPI
  - Multiple tools concurrently
  - Dynamic loading of tools
  - Configuration through text file
  - Tools are independent
  - Tools can collaborate



- PMPI interception of MPI calls
  - Easy to include in applications
  - Limited to a single tool
- P^NMPI virtualized PMPI
  - Multiple tools concurrently
  - Dynamic loading of tools
  - Configuration through text file
  - Tools are independent
  - Tools can collaborate
- Transparently adding context
  - Select tool based on MPI context
  - Transparently isolate tool instances



Example: Defining Switch Modules in P^NMPI

Configuration file:

```
module commsize-switch
argument sizes 8 4
argument stacks column row
module mpiP

stack row
module mpiP1

stack column
module mpiP2
```

Default Stack: module commsize-switch … module mpiP

Target Stack 1: stack row, module mpiP1

Target Stack 2: stack column, module mpiP2

Switch Module

Arguments controlling switch module

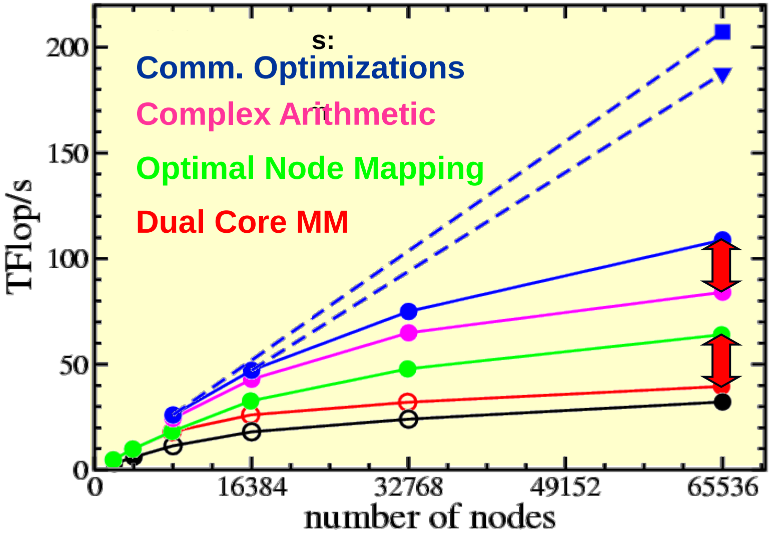
Multiple profiling instances

Communicator Profiling Results for QBox

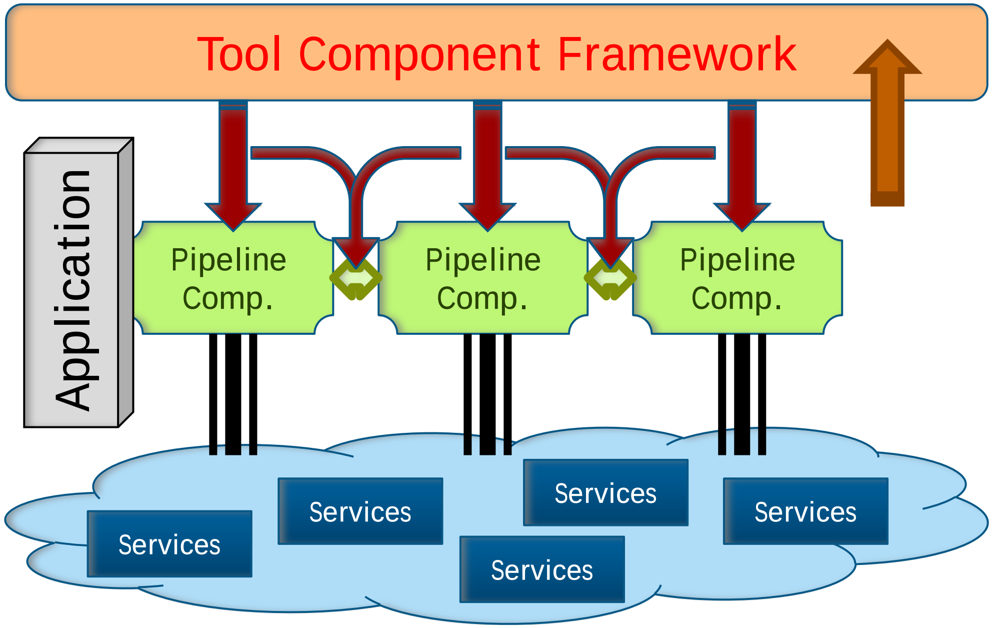
Operation	Sum	Global	Row	Column
Send	317245	31014	202972	83259
Allreduce	319028	269876	49152	0
Alltoallv	471488	471488	0	0
Recv	379265	93034	202972	83259
Bcast	401042	11168	331698	58176

AMD Opteron/Infiniband Cluster

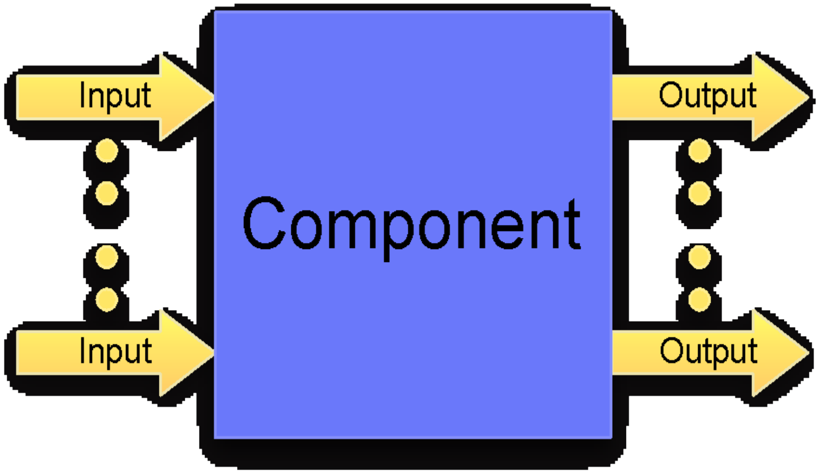
- Lessons learned from QBox
  - Node mappings are critical
  - Performance effects often show only at scale
  - Need to understand behavior and customize tool behavior
  - Need for tools to break black box abstractions



- Component Based Tool Framework (CBTF)
  - Independent components connected by typed pipes
  - Transforming data coming from the application on the way to the user
  - External specification of which components to connect
  - Each combination of components is/can be "a tool"
  - Shared services
- Partners
  - Krell Institute
  - LANL, LLNL, SNLs
  - ORNL
  - UW, UMD
  - CMU

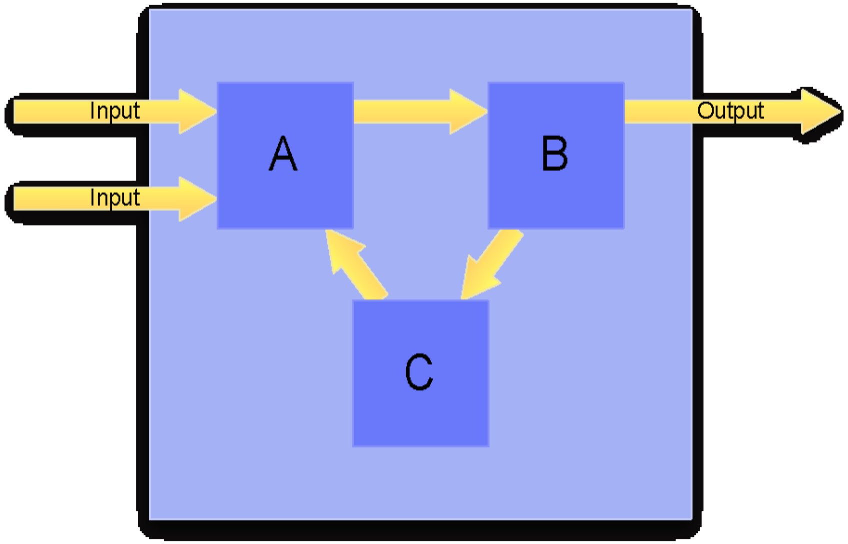


CBTF Modules



- Data-Flow Model
  - Accepts Inputs
  - Performs Processing
  - Emits Outputs
- C++ Based
- Provide Metadata
  - Type & Version
  - Input Names & Types
  - Output Names & Types
- Versioned
  - Concurrent Versions
- Packaging
  - Executable-Embedded
  - Shared Library
  - Runtime Plugin

CBTF Component Networks



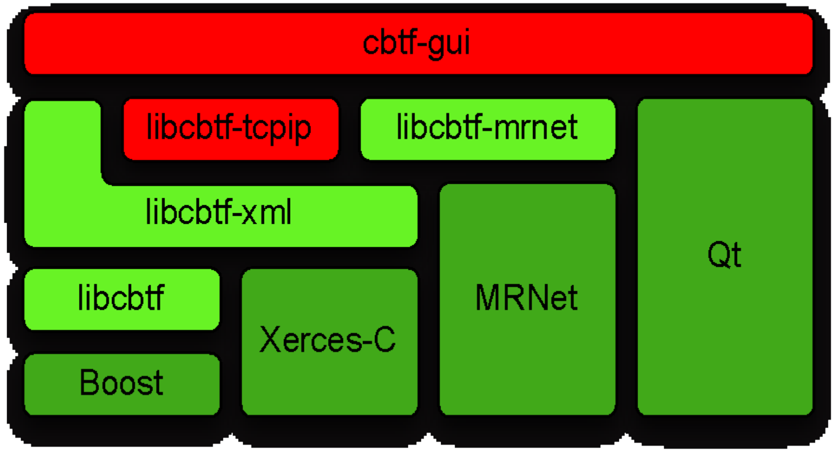
- Components
  - Specific Versions
- Connections
  - Matching Types
- Arbitrary Component Topology
  - Pipelines
  - Graphs with cycles
  - ….
- Recursive
  - Network itself is a component
- XML-Specified

Specifying Component Networks to Create New Tools

```
....
<Type>ExampleNetwork</Type>
<Version>1.2.3</Version>
<SearchPath>.:/opt/myplugins</SearchPath>
<Plugin>myplugin</Plugin>
    <Component>
      <Name>A1</Name>
     <Type>TestComponentA</Type>
    </Component>
…
 <Network>
…
    <Connection>
       <From>
         <Name>A1</Name>
         <Output>out</Output>
       </From>
       <To>
           <Name>A2</Name>
           <Input>in</Input>
        </To>
      </Connection>
…
 </Network>
```

- Users can create new tools by specifying new networks
  - Combine existing functionality
  - Reuse general model
  - Add application specific details
    - Phase/context filters
    - Data mappings
- Connection information
  - Which components?
  - Which ports connected?
  - Grouping into networks
- Implemented as XML
  - User writable
  - Could be generated by a GUI

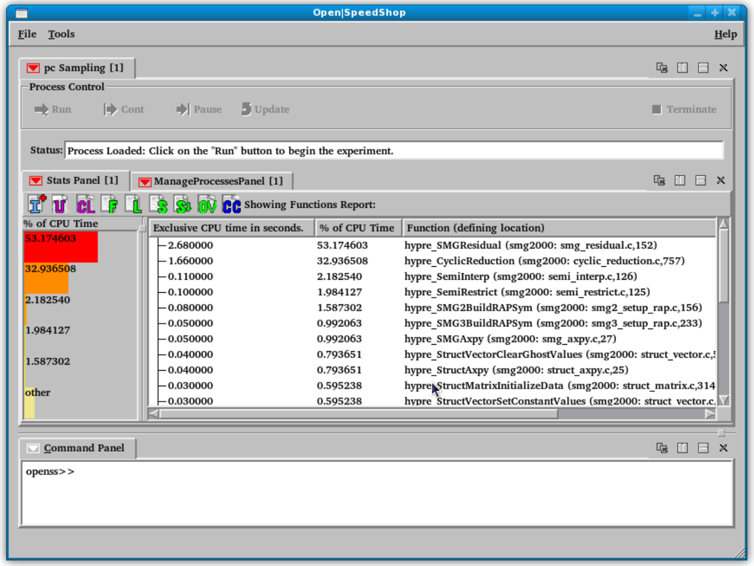
CBTF Structure and Dependencies



- Minimal Dependencies
  - Easier Builds
- Tool-Type Independent
  - Performance Tools
  - Debugging Tools
  - etc…
- Completed Components
  - Base Library (libcbtf)
  - XML-Based Component Networks (libcbtf-xml)
  - MRNet Distributed Components (libcbtf-mrnet)
- Planned Components
  - TCP/IP Distributed Component Networks
  - GUI Definition of Component Networks

Tools on Top of CBTF

- Open|SpeedShop v2.0
  - CBTF created by componentizing the existing Open|SpeedShop
  - Motivation:
    scalability & maintainability
- Extensions for O|SS in CBTF (planned for 10/13)
  - Threading overheads
  - Memory consumption
  - I/O profiling
- Further tools in progress
  - GPU performance analysis
  - Tools for system administration and health monitoring



Summary: The Need for Components

- We need frameworks that enable …
  - Independently created and maintained components
  - Flexible connection of components
  - Assembly of new tools from these components by the user
- CBTF is designed as a generic tool framework
  - Components are connected by typed pipes
  - Infrastructure for hierarchical aggregation with user defined functions
  - Component specification is external through XML files
  - Tailor tools by combining generic and application specific tools
- CBTF is available as a pre-release version
  - First prototype of Open|SpeedShop v2.0 working
  - New extensions for O|SS exploiting CBTF advantages
  - Several new tools built on top of CBTF
  - Wiki at http://ft.ornl.gov/doku/cbtfw/start
  - Code available on sourceforge

Review

Brian Wylie
Jülich Supercomputing Centre

Summary

You've been introduced to a variety of widely-available tools, and seen their basic use demonstrated

– with some guidance to apply and use the tools most effectively

- Tools provide complementary capabilities
  – computational kernel & processor analyses
  – communication/synchronization analyses
  – load-balance, scheduling, scaling, …
- Tools are designed with various trade-offs
  – general-purpose versus specialized
  – platform-specific versus agnostic
  – simple/basic versus complex/powerful

Tool selection

- Which tools you use and when you use them likely to depend on situation
  - which are available on (or for) your computer system
  - which support your programming paradigms and languages
  - which you are familiar (comfortable) with using
- also depends on the type of issue you have or suspect

- Awareness of (potentially) available tools can help finding the most appropriate tools

Workflow (getting started)

- First ensure that the parallel application runs correctly
  – no-one will care how quickly you can get invalid answers or produce a directory full of corefiles
  – parallel debuggers help isolate known problems
    - *STAT* can help reducing focus to smaller sets of processes
  – correctness checking tools can help identify other issues
  – (that might not cause problems right now, but will eventually)
    - e.g., race conditions, invalid/non-compliant usage
- Generally valuable to start with an overview of execution performance
  – fraction of time spent in computation vs comm/synch vs I/O
  – which sections of the application/library code are most costly
- and how it changes with scale or different configurations
  – processes vs threads, mappings, bindings

Workflow (communication/synchronization)

- Communication/synchronization issues generally apply to every computer system (to different extents) and typically grow with the number of processes/threads
  - *Weak scaling*: fixed computation per thread, and perhaps fixed localities, but increasingly distributed
  - *Strong scaling*: constant total computation, increasingly divided amongst threads, while communication grows
  - Collective communication (particularly of type “all-to-all”) result in increasing data movement
  - Synchronizations of larger groups are increasingly costly
  - Load-balancing becomes increasingly challenging, and imbalances increasingly expensive
    - generally manifests as waiting time at following collective ops
  - Mapping of processes / threads can also be important

Workflow (wasted waiting time)

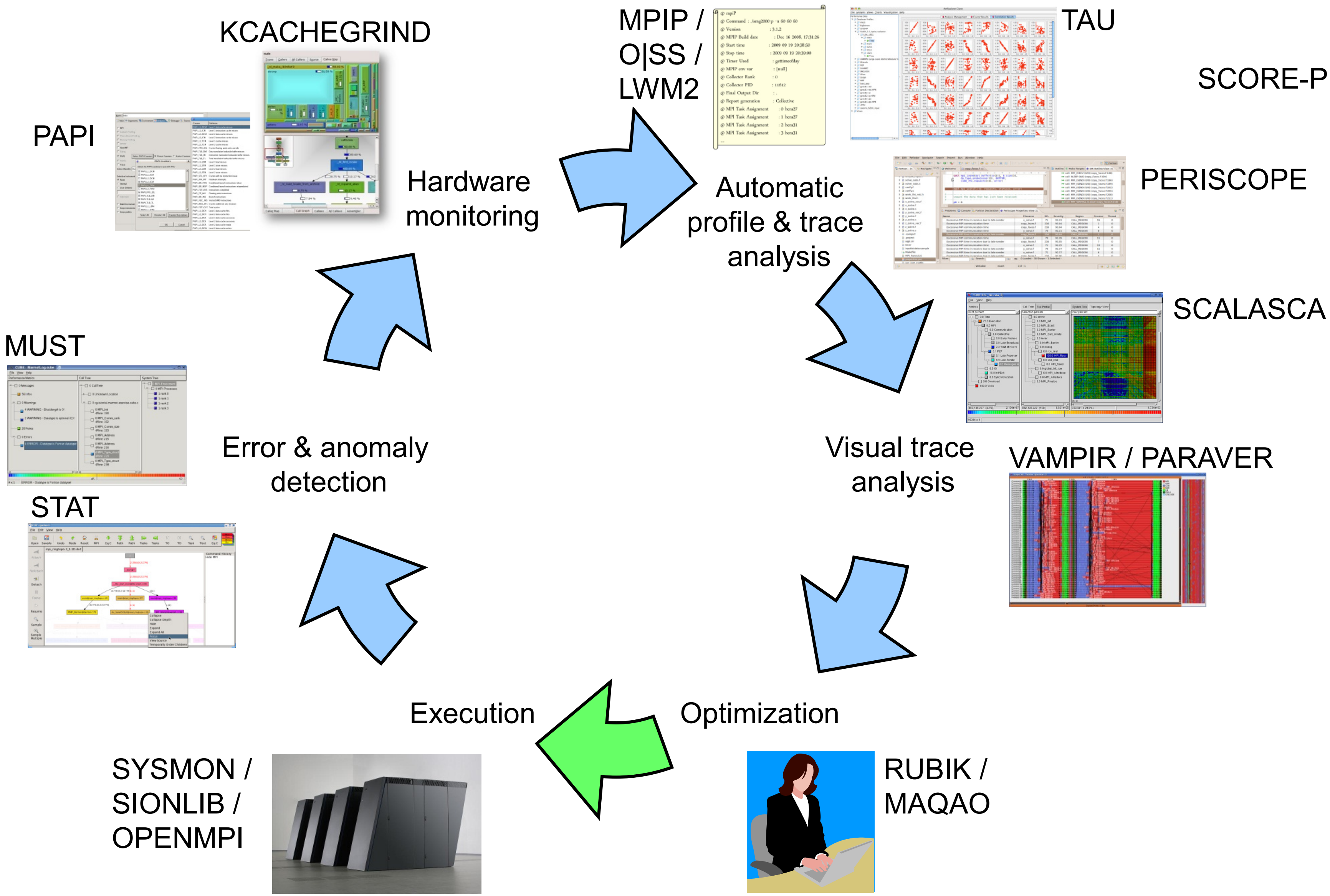
- Waiting times are difficult to determine in basic profiles
  - Part of the time each process/thread spends in communication & synchronization operations may be wasted waiting time
  - Need to correlate event times between processes/threads
    - *Periscope* uses augmented messages to transfer timestamps and additional on-line analysis processes
    - Post-mortem event trace analysis avoids interference and provides a complete history
    - *Scalasca* automates trace analysis and ensures waiting times are completely quantified
    - *Vampir* allows interactive exploration and detailed examination of reasons for inefficiencies

Workflow (core computation)

Effective computation within processors/cores is also vital

- Optimized libraries may already be available
- Optimizing compilers can also do a lot
  - provided the code is clearly written and not too complex
  - appropriate directives and other hints can also help
- Processor hardware counters can also provide insight
  - although hardware-specific interpretation required
- Tools available from processor and system vendors help navigate and interpret processor-specific performance issues

VI-HPS tools portfolio and their integration



Further information

- Website
  - Introductory information about the VI-HPS portfolio of tools for high-productivity parallel application development
    - links to individual tools sites for details and download
  - Training material
    - tutorial slides
    - latest ISO image of VI-HPS Linux DVD with productivity tools
    - user guides and reference manuals for tools
  - News of upcoming events
    - tutorials and workshops
    - mailing-list sign-up for announcements

http://www.vi-hps.org