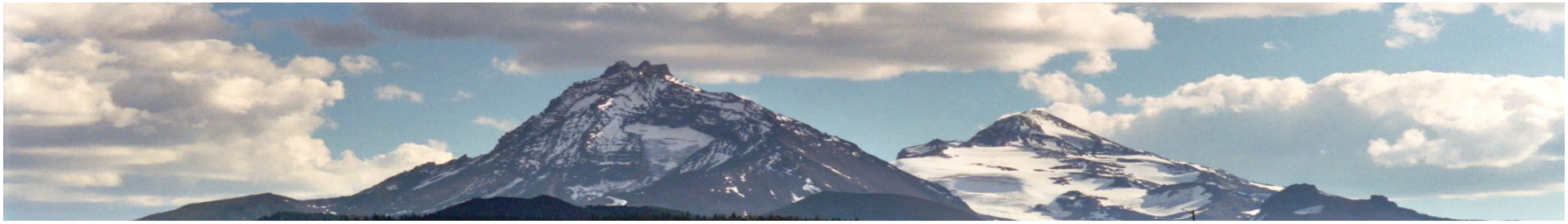
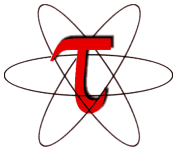


Lecture 6:
TAU Performance System

Allen D. Malony
Department of Computer and Information Science

UNIVERSITY OF OREGON

TAU Performance System®



- Tuning and Analysis Utilities (20+ year project)
- Performance problem solving framework for HPC
  - Integrated, scalable, flexible, portable
  - Target all parallel programming / execution paradigms
- Integrated performance toolkit
  - Multi-level performance instrumentation
  - Flexible and configurable performance measurement
  - Widely-ported performance profiling / tracing system
  - Performance data management and data mining
  - Open source (BSD-style license)
- Broad use in complex software, systems, applications

http://tau.uoregon.edu

TAU History

1992-1995: Malony and Mohr work with Gannon on DARPA pC++ project work. TAU is born. [parallel profiling, tracing, performance extrapolation]

1995-1998: Shende works on Ph.D. research on performance mapping. TAU v1.0 released. [multiple languages, source analysis, automatic instrumentation]

1998-2001: Significant effort in Fortran analysis and instrumentation, work with Mohr on POMP, Kojak tracing integration, focus on automated performance analysis. [performance diagnosis, source analysis, instrumentation]

2002-2005: Focus on profiling analysis tools, measurement scalability, and perturbation compensation. [analysis, scalability, perturbation analysis, applications]

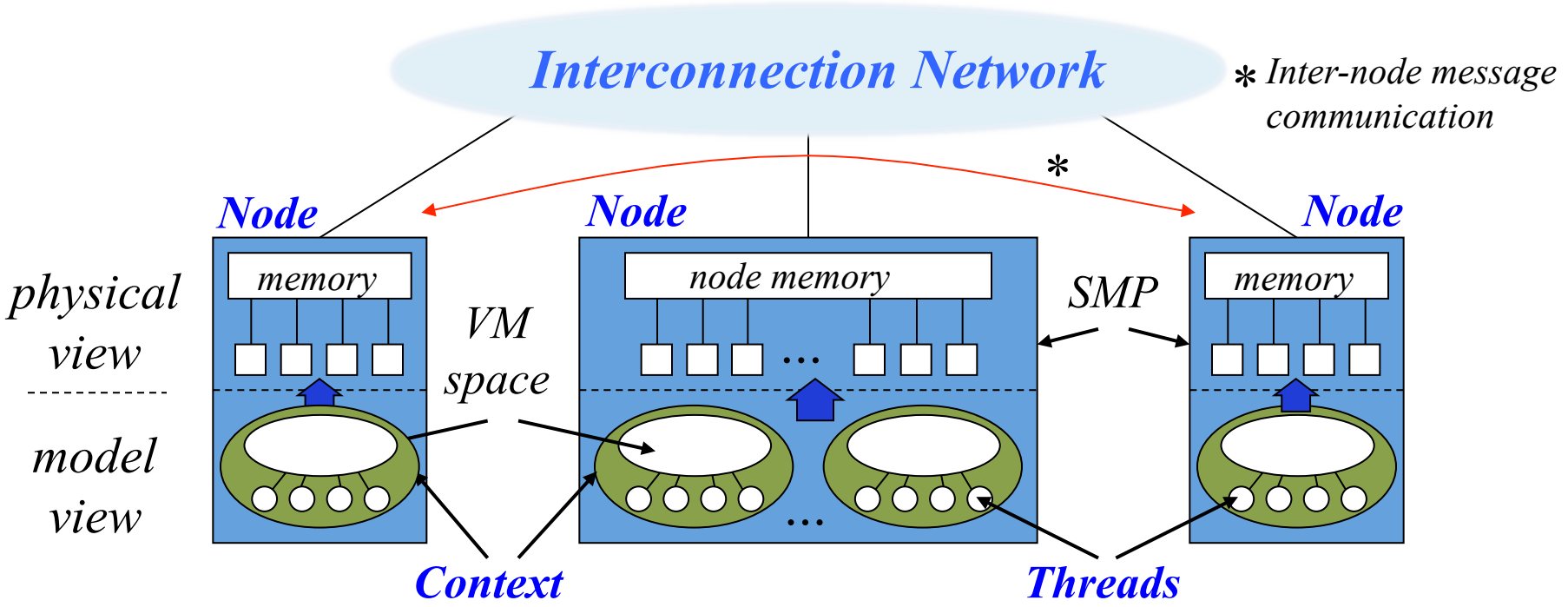
2005-2007: More emphasis on tool integration, usability, and data presentation. TAU v2.0 released. [performance visualization, binary instrumentation, integration, performance diagnosis and modeling]

2008-2011: Add performance database support, data mining, and rule-based analysis. Develop measurement/analysis for heterogeneous systems. Core measurement infrastructure integration (Score-P). [database, data mining, expert system, heterogeneous measurement, infrastructure integration]

2012-present: Focus on exascale systems. Improve scalability. Add hybrid measurement support, extend heterogeneous and mixed-mode, develop user-level threading. Apply to petascale / exascale applications. [scale, autotuning, user-level]

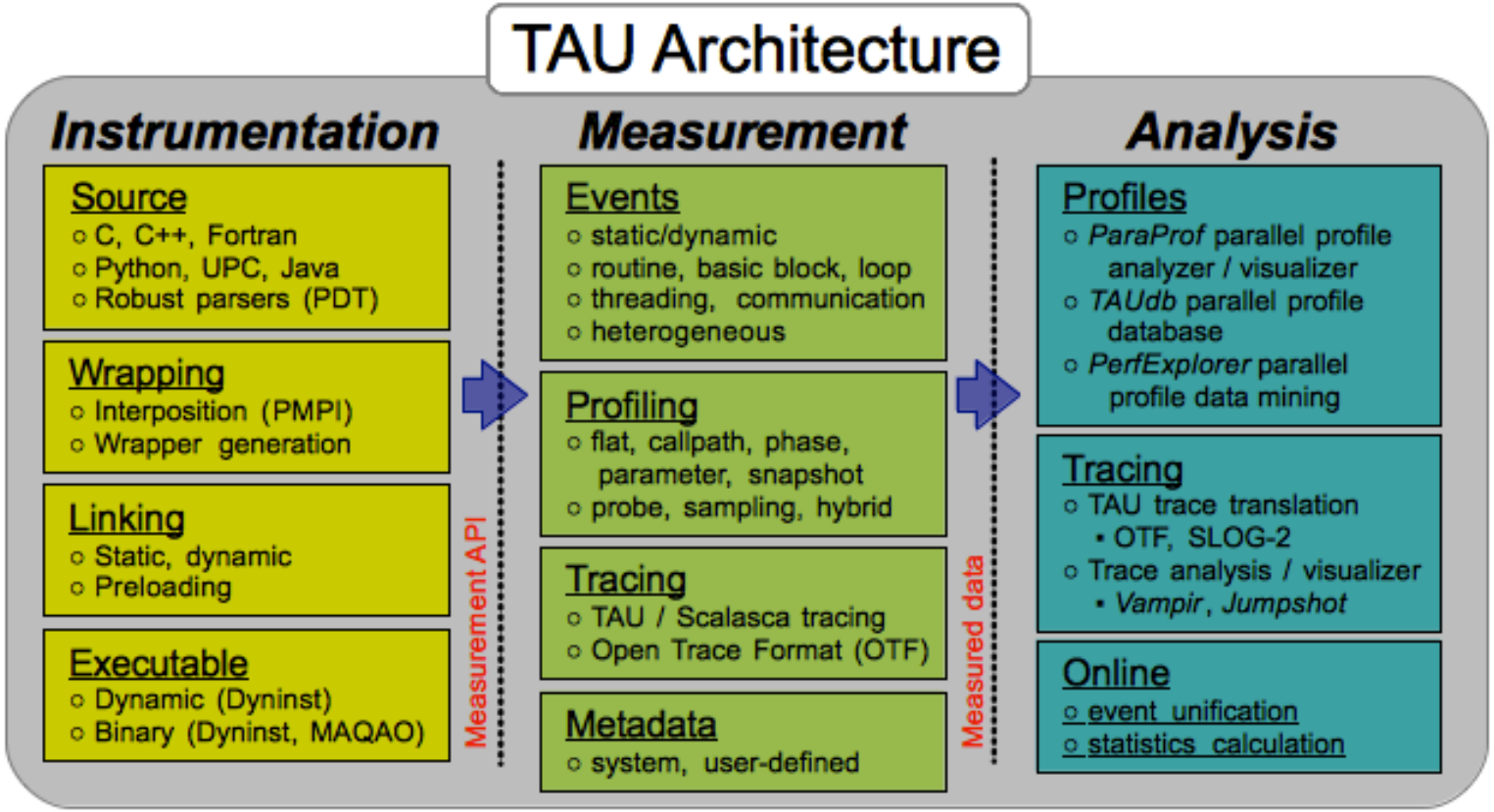
General Target Computation Model in TAU

- Node: physically distinct shared memory machine
  - Message passing node interconnection network
- Context: distinct virtual memory space within node
- Thread: execution threads (user/system) in context



TAU Architecture

- TAU is a parallel performance framework and toolkit
- Software architecture provides separation of concerns
  - Instrumentation | Measurement | Analysis



TAU Observation Methodology and Workflow

- TAU's (primary) methodology for parallel performance observation is based on the insertion of measurement probes into application, library, and runtime system
  - Code is instrumented to make visible certain events
  - Performance measurements occur when events are triggered
  - Known as probe-based (direct) measurement
- Performance experimentation workflow
  - Instrument application and other code components
  - Link / load TAU measurement library
  - Execute program to gather performance data
  - Analysis performance data with respect to events
  - Analyze multiple performance experiments
- Extended TAU's methodology and workflow to support sampling-based techniques

TAU Components

- Instrumentation
  - Fortran, C, C++, OpenMP, Python, Java, UPC, Chapel
  - Source, compiler, library wrapping, binary rewriting
  - Automatic instrumentation
- Measurement
  - Internode: MPI, OpenSHMEM, ARMCI, PGAS, DMAPP
  - Intranode: Pthreads, OpenMP, hybrid, …
  - Heterogeneous: GPU, MIC, CUDA, OpenCL, OpenACC, …
  - Performance data (timing, counters) and metadata
  - Parallel profiling and tracing (with Score-P integration)
- Analysis
  - Parallel profile analysis and visualization (ParaProf)
  - Performance data mining / machine learning (PerfExplorer)
  - Performance database technology (TAUdb)
  - Empirical autotuning

TAU Instrumentation Approach

- Direct and indirect performance instrumentation
  - *Direct* instrumentation of program (system) code (probes)
  - *Indirect* support via sampling or interrupts
- Support for standard program code events
  - Routines, classes and templates
  - Statement-level blocks, loops
  - *Interval events* (start/stop)
- Support for user-defined events
  - Interval events specified by user
  - *Atomic events* (statistical measurement at a single point)
  - *Context events* (atomic events with calling path context)
- Provides static events and dynamic events
- Instrumentation optimization

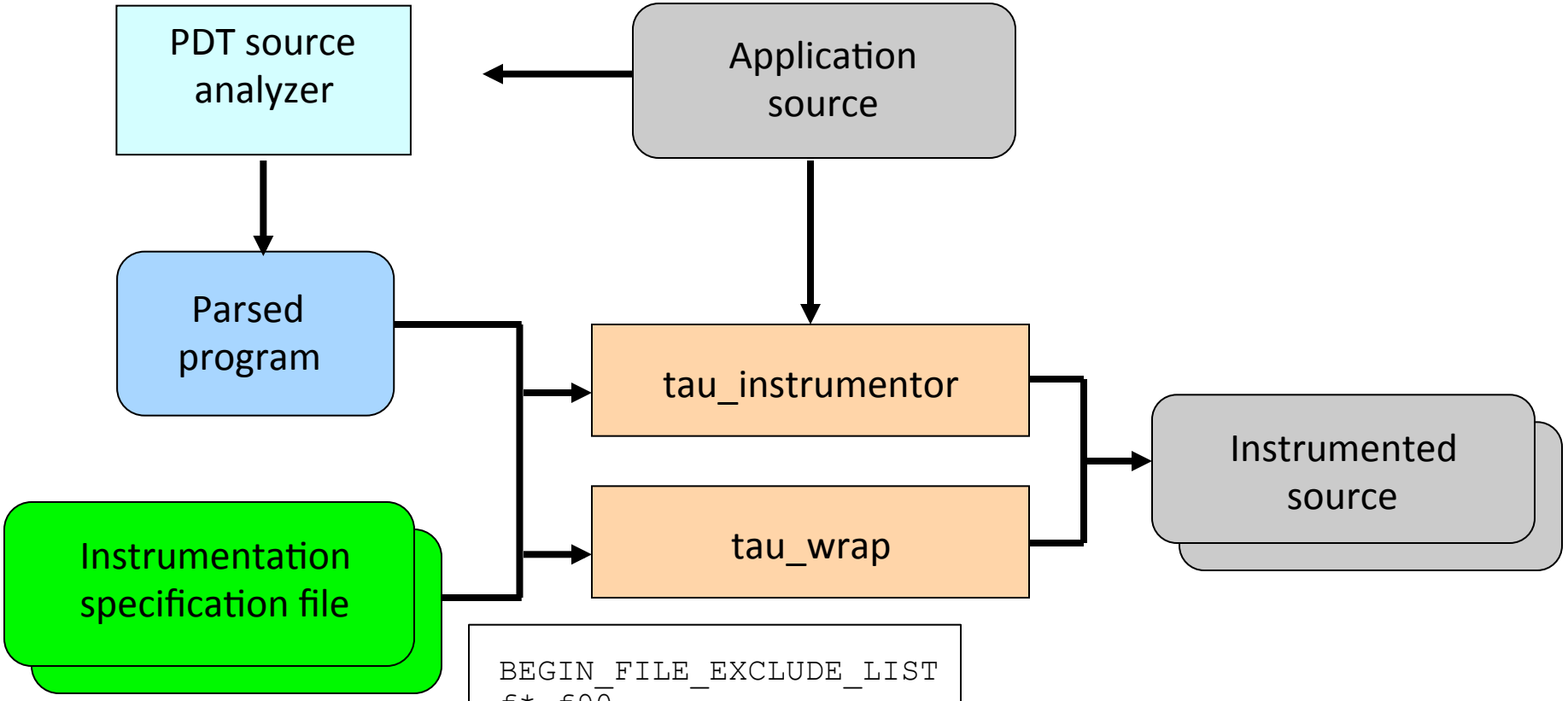
TAU Instrumentation Mechanisms

- Source code
  - Manual (TAU API, TAU component API)
  - Automatic (robust)
    - C, C++, F77/90/95, OpenMP (POMP/OPARI), UPC
  - Compiler (GNU, IBM, NAG, Intel, PGI, Pathscale, Cray, …)
- Object code (library-level)
  - Statically- and dynamically-linked wrapper libraries
    - MPI, I/O, memory, …
  - Powerful library wrapping of external libraries without source
- Executable code / runtime
  - Runtime preloading and interception of library calls
  - Binary instrumentation (Dyninst, MAQAO, PEBIL)
  - Dynamic instrumentation (Dyninst)
  - OpenMP (runtime API, CollectorAPI, GOMP, OMPT)
- Virtual machine, interpreter, and OS instrumentation

Instrumentation for Wrapping External Libraries

- Preprocessor substitution
  - Header redefines a routine with macros (only C and C++)
  - Tool-defined header file with same name takes precedence
  - Original routine substituted by preprocessor callsite
- Preloading a library at runtime
  - Library preloaded in the address space of executing application intercepts calls from a given library
  - Tool wrapper library defines routine, gets address of global symbol (*dlsym*), internally calls measured routine
- Linker-based substitution
  - Wrapper library defines wrapper interface
    - wrapper interface then which calls routine
  - Linker is passed option to substitute all references from applications object code with tool wrappers

Automatic Source-level / Wrapper Instrumentation




```
BEGIN_EXCLUDE_LIST
Foo
Bar
D#EMM
END_EXCLUDE_LIST
```

```
BEGIN_FILE_EXCLUDE_LIST
f*.f90
Foo?.cpp
END_FILE_EXCLUDE_LIST
BEGIN_FILE_INCLUDE_LIST
main.cpp
foo.f90
END_FILE_INCLUDE_LIST
```

MPI Wrapper Interposition Library

- Uses standard MPI Profiling Interface
  - Provides name shifted interface (weak bindings)
    - *MPI_Send = PMPI_Send*
- Create TAU instrumented MPI library
  - Interpose between MPI and TAU
    - `-lmpi` replaced by `-lTauMpi -lpmpi -lmpi`
  - No change to the source code, just re-link application!
- Can we interpose MPI for compiled applications?
  - Avoid re-compilation or re-linking
  - Requires shared library MPI
    - uses `LD_PRELOAD` for Linux
  - Approach will work with other shared libraries (see later slide)
  - Use TAU `tau_exec` (see later slide)

```
% mpirun -np 4 tau_exec a.out
```

Binary Instrumentation

- TAU has been a long-time user of DyninstAPI
- Using DyninstAPI's binary re-writing capabilities, created a binary re-writer tool for TAU (`tau_run`)
  - Supports TAU's performance instrumentation
  - Works with TAU instrumentation selection
    - files and routines based on exclude/include lists
  - TAU's measurement library (DSO) is loaded by `tau_run`
  - Runtime (pre-execution) and binary re-writing supported
- Simplifies code instrumentation and usage greatly!

```
% tau_run a.out -o a.inst
% mpirun -np 4 ./a.inst
```

- Support PEBIL and MAQAO binary instrumentation

Library Interposition

- Simplify TAU usage to assess performance properties
  - Application, I/O, memory, communication
- Designed a new tool that leverages runtime instrumentation by pre-loading measurement libraries
- Works on dynamic executables (default under Linux)
- Substitutes routines (e.g., I/O, MPI, memory allocation/deallocation) with instrumented calls
  - Interval events (e.g., time spent in write())
  - Atomic events (e.g., how much memory was allocated)

TAU Execution Command – tau_exec

- Uninstrumented execution

```
% mpirun -np 256 ./a.out
```

- Track MPI performance

```
% mpirun -np 256  tau_exec ./a.out
```

- Track I/O and MPI performance (MPI default)

```
% mpirun -np 256 tau_exec -io  ./a.out
```

- Track memory operations

```
% setenv TAU_TRACK_MEMORY_LEAKS   1
% mpirun -np 256 tau_exec -memory ./a.out
```

- Track I/O performance and memory operations

```
% mpirun -np 256 tau_exec -io -memory ./a.out
```

POSIX I/O Calls Supported

- Unbuffered I/O
  - *open, open64, close, read, write, readv, writev, creat, creat64*
- Buffered I/O
  - *fopen, fopen64, fdopen, freopen, fclose*
  - *fprintf, fscanf, fwrite, fread*
- Communication
  - *socket, pipe, socketpair, bind, accept, connect*
  - *recv, send, sendto, recvfrom, pclose*
- Control
  - *fcntl, rewind, lseek, lseek64, fseek, dup, dup2, mkstep, tmpfile*
- Asynchronous I/O
  - *aio_{read,write,suspend,cancel,return}, lio_listio*

Library wrapping – tau_gen_wrapper

- How to instrument an external library without source?
  - Source may not be available
  - Library may be too cumbersome to build (with instrumentation)
- Build a library wrapper tools
  - Used PDT to parse header files
  - Generate new header files with instrumentation files
  - Three methods: runtime preloading, linking, redirecting headers
- Add to `TAU_OPTIONS` environment variable:

```
-optTauWrapFile=<wrapperdir>/link_options.tau
```

- Wrapped library
  - Redirects references at routine callsite to a wrapper call
  - Wrapper internally calls the original
  - Wrapper has TAU measurement code

HDF5 Library Wrapping

```
[sameer@zorak]$ tau_gen_wrapper hdf5.h /usr/lib/libhdf5.a -f select.tau

Usage : tau_gen_wrapper <header> <library> [-r|-d|-w (default)] [-g groupname] [-i
headerfile] [-c|-c++|-fortran] [-f <instr_req_file> ]
• instruments using runtime preloading (-r), or -Wl,-wrap linker (-w), redirection of
header file to redefine the wrapped routine (-d)
• instrumentation specification file (select.tau)
• group (hdf5)
• tau_exec loads libhdf5_wrap.so shared library using –loadlib=<libwrap_pkg.so>
• creates the wrapper/ directory with -opt

NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.057            1           1          13       1236 .TAU Application
 70.8        0.875        0.875           1           0        875 hid_t H5Fcreate()
  9.7         0.12         0.12           1           0        120 herr_t H5Fclose()
  6.0        0.074        0.074           1           0         74 hid_t H5Dcreate()
  3.1        0.038        0.038           1           0         38 herr_t H5Dwrite()
  2.6        0.032        0.032           1           0         32 herr_t H5Dclose()
  2.1        0.026        0.026           1           0         26 herr_t H5check_version()
  0.6        0.008        0.008           1           0          8 hid_t H5Screate_simple()
  0.2        0.002        0.002           1           0          2 herr_t H5Tset_order()
  0.2        0.002        0.002           1           0          2 hid_t H5Tcopy()
  0.1        0.001        0.001           1           0          1 herr_t H5Sclose()
  0.1        0.001        0.001           2           0          0 herr_t H5open()
```

TAU Measurement Approach

- Portable and scalable parallel profiling solution
  - Multiple profiling types and options
  - Event selection and control (enabling/disabling, throttling)
  - Online profile access and sampling
  - Online performance profile overhead compensation
- Portable and scalable parallel tracing solution
  - Trace translation to OTF, EPILOG, Paraver, and SLOG2
  - Trace streams (OTF) and hierarchical trace merging
- Robust timing and hardware performance support
- Multiple counters (hardware, user-defined, system)
- Metadata (hardware/system, application, …)

TAU Measurement Mechanisms

- Parallel profiling
  - Function-level, block-level, statement-level
  - Supports user-defined events and mapping events
  - Support for flat, callgraph/callpath, phase profiling
  - Support for parameter and context profiling
  - Support for tracking I/O and memory (library wrappers)
  - Parallel profile stored (dumped, shapshot) during execution
- Tracing
  - All profile-level events
  - Inter-process communication events
  - Inclusion of multiple counter data in traced events

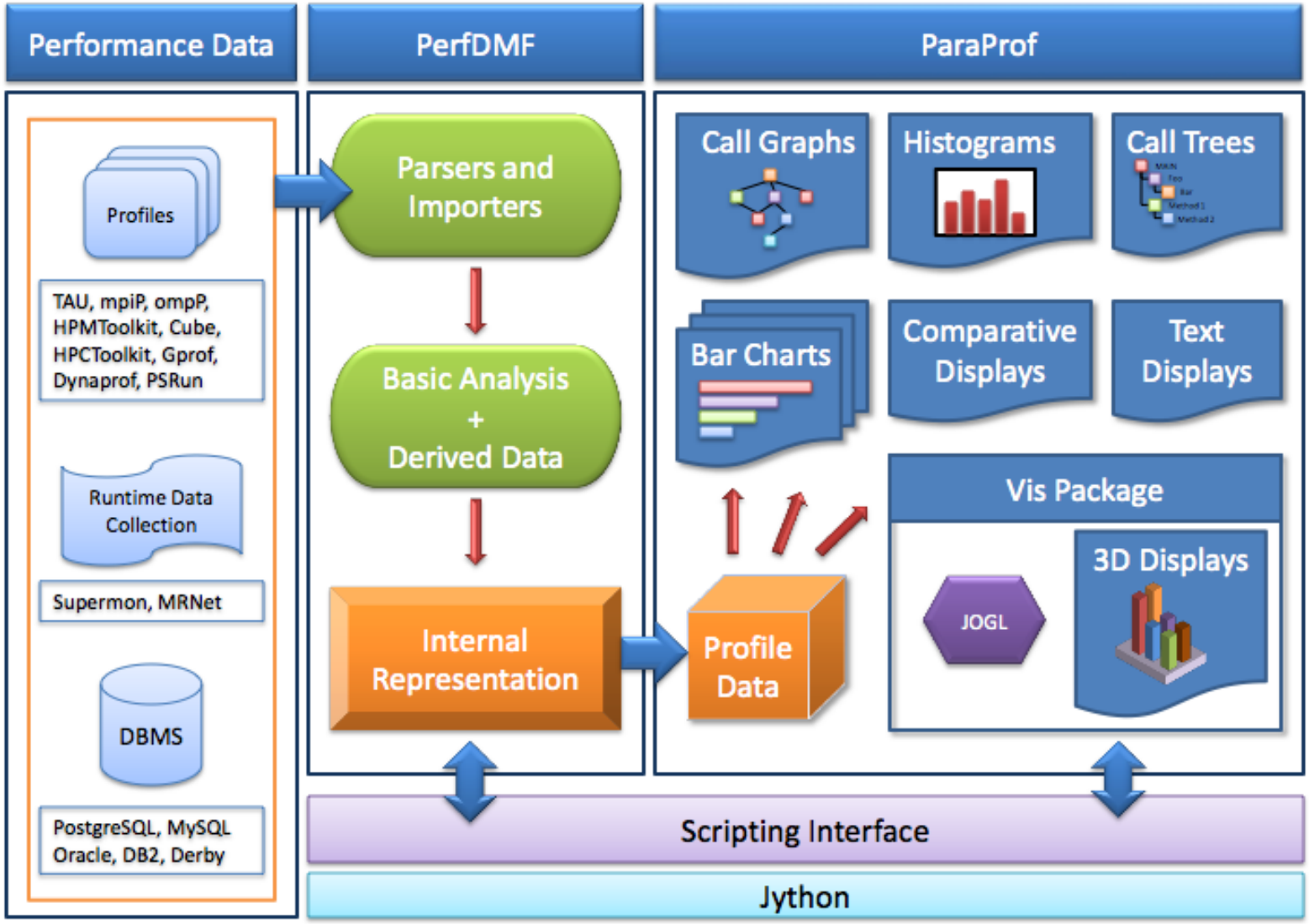
Parallel Performance Profiling

- Flat profiles
  - Metric (e.g., time) spent in an event (callgraph nodes)
  - Exclusive/inclusive, # of calls, child calls
- Callpath profiles (Calldepth profiles)
  - Time spent along a calling path (edges in callgraph)
  - "main=> f1 => f2 => MPI_Send" (event name)
  - TAU_CALLPATH_DEPTH environment variable
- Phase profiles
  - Flat profiles under a phase (nested phases are allowed)
  - Default "main" phase
  - Supports static or dynamic (per-iteration) phases
- Parameter and context profiling

Performance Analysis

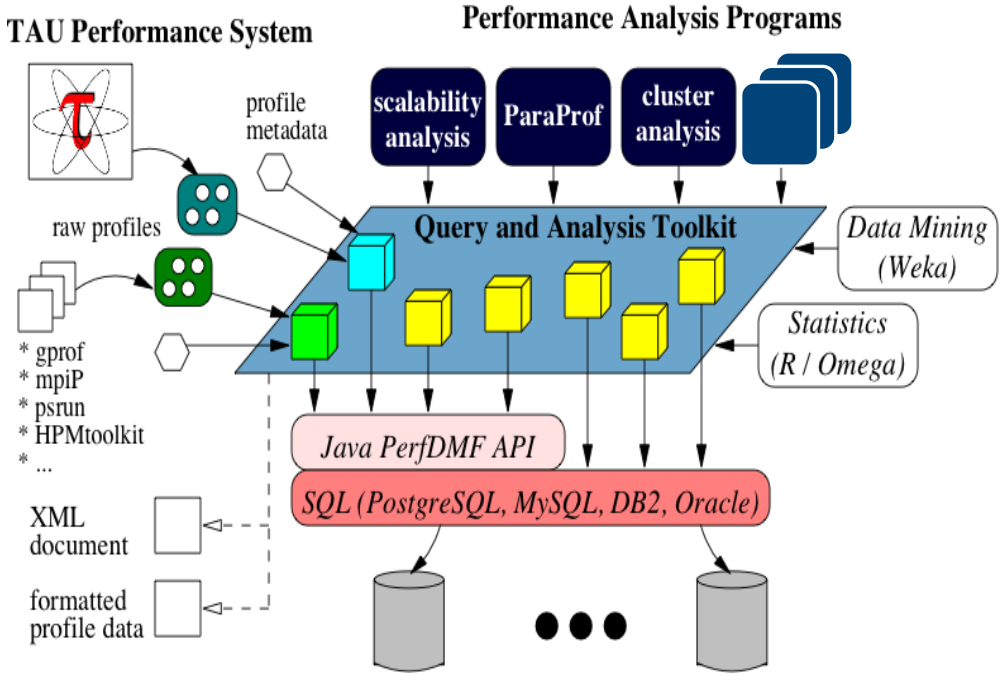
- Analysis of parallel profile and trace measurement
- Parallel profile analysis (ParaProf)
  - Java-based analysis and visualization tool
  - Support for large-scale parallel profiles
- Performance data management (TAUdb)
- Performance data mining (PerfExplorer)
- Parallel trace analysis
  - Translation to VTF (V3.0), EPILOG, OTF formats
  - Integration with Vampir / Vampir Server (TU Dresden)
- Integration with CUBE browser (Scalasca, UTK / FZJ)
- Scalable runtime fault isolation with callstack debugging
- Efficient parallel runtime bounds checking

Profile Analysis Framework



Performance Data Management (TAUdb)

- Provide an open, flexible framework to support common data management tasks
  - Foster multi-experiment performance evaluation
- Extensible toolkit to promote integration and reuse across available performance tools
  - Supported multiple profile formats: TAU, CUBE, gprof, mpiP, psrun, …
  - Supported DBMS: PostgreSQL, MySQL, Oracle, DB2, Derby, H2

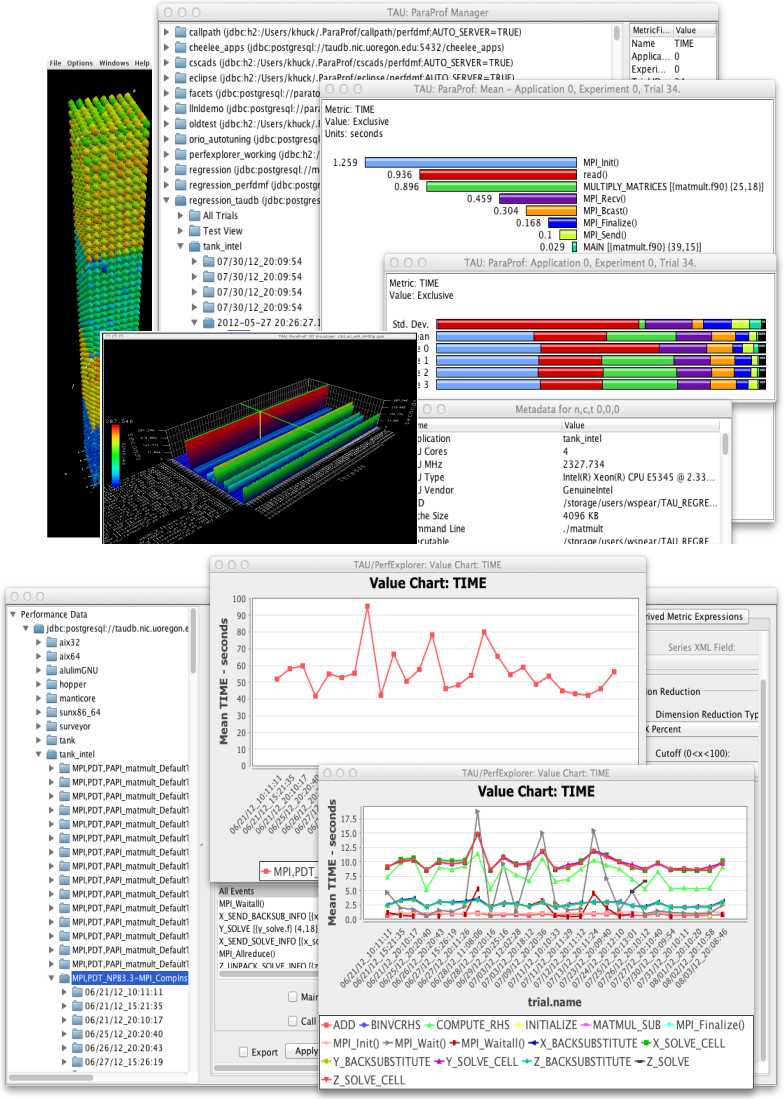


TAUdb Database Schema

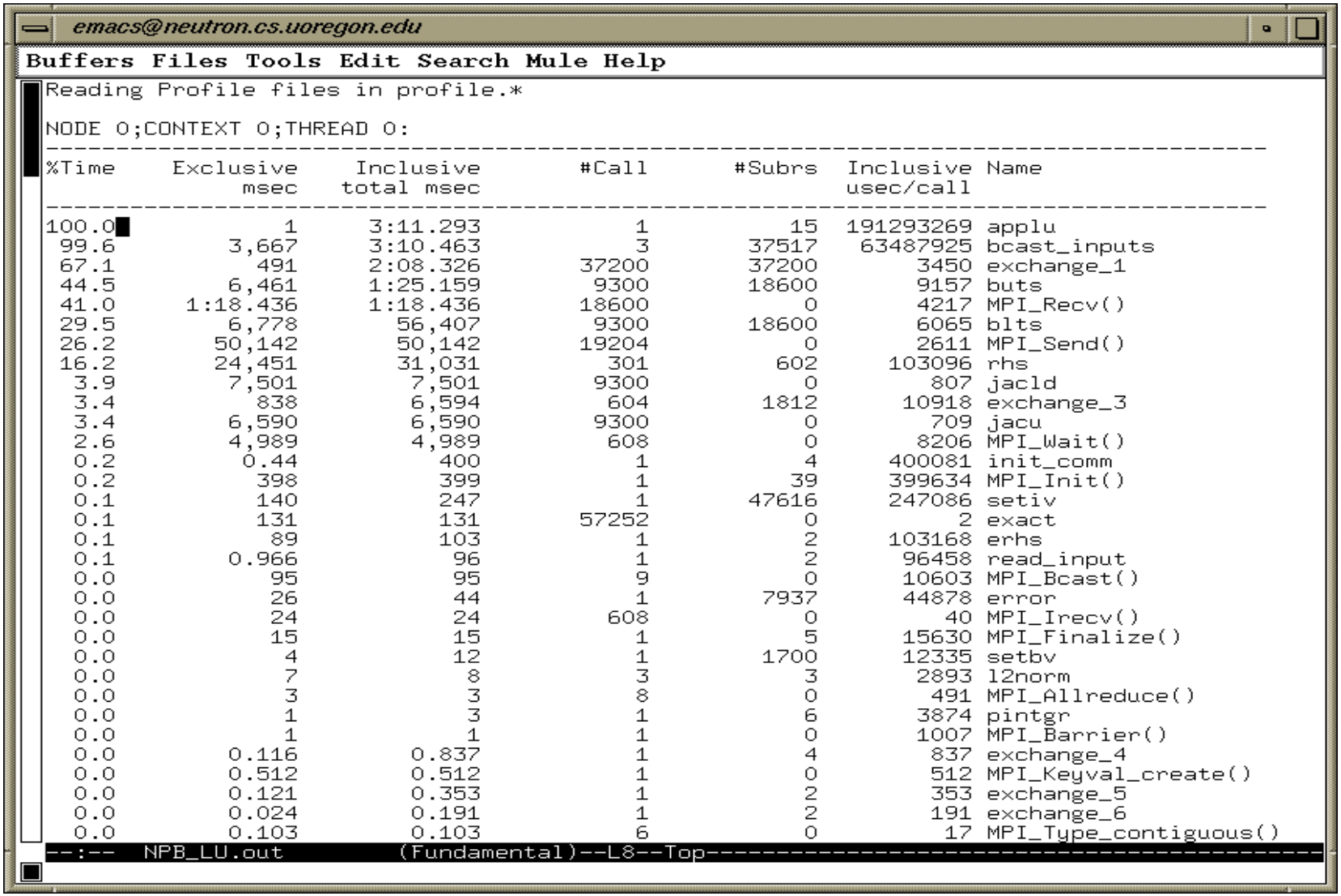
- Parallel performance profiles
- Timer and counter measurements with 5 dimensions
  - Physical location: process / thread
  - Static code location: function / loop / block / line
  - Dynamic location: current callpath and context (parameters)
  - Time context: iteration / snapshot / phase
  - Metric: time, HW counters, derived values
- Measurement metadata
  - Properties of the experiment
  - Anything from name:value pairs to nested, structured data
  - Single value for whole experiment or full context (tuple of thread, timer, iteration, timestamp)

TAUdb Tool Support

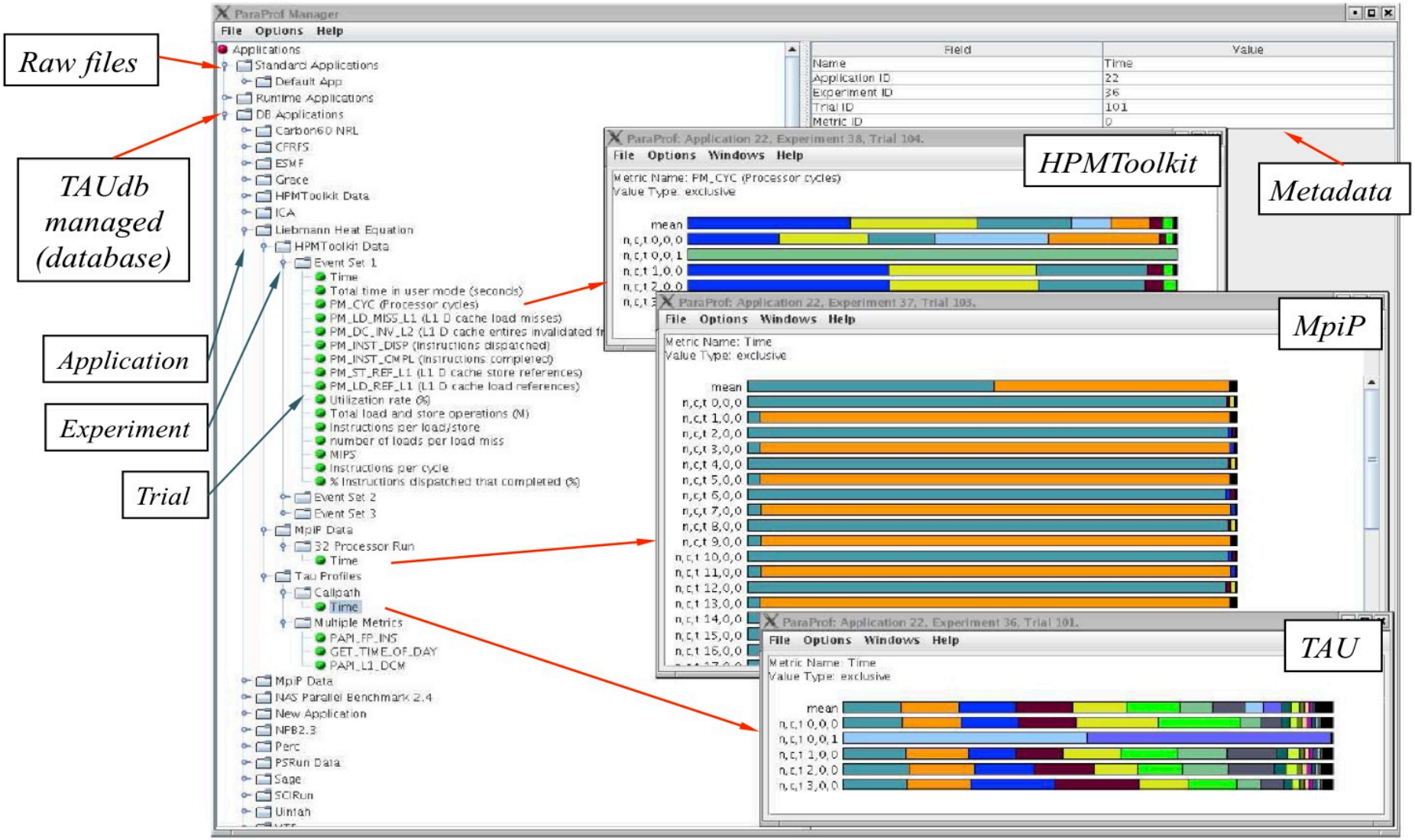
- ParaProf
  - Parallel profile analyzer
    - visual pprof
  - 2, 3+D visualizations
  - Single and comparative experiment analysis
- PerfExplorer
  - Data mining framework
    - Clustering, correlation
  - Multi-experiment analysis
  - Scripting engine
  - Expert system



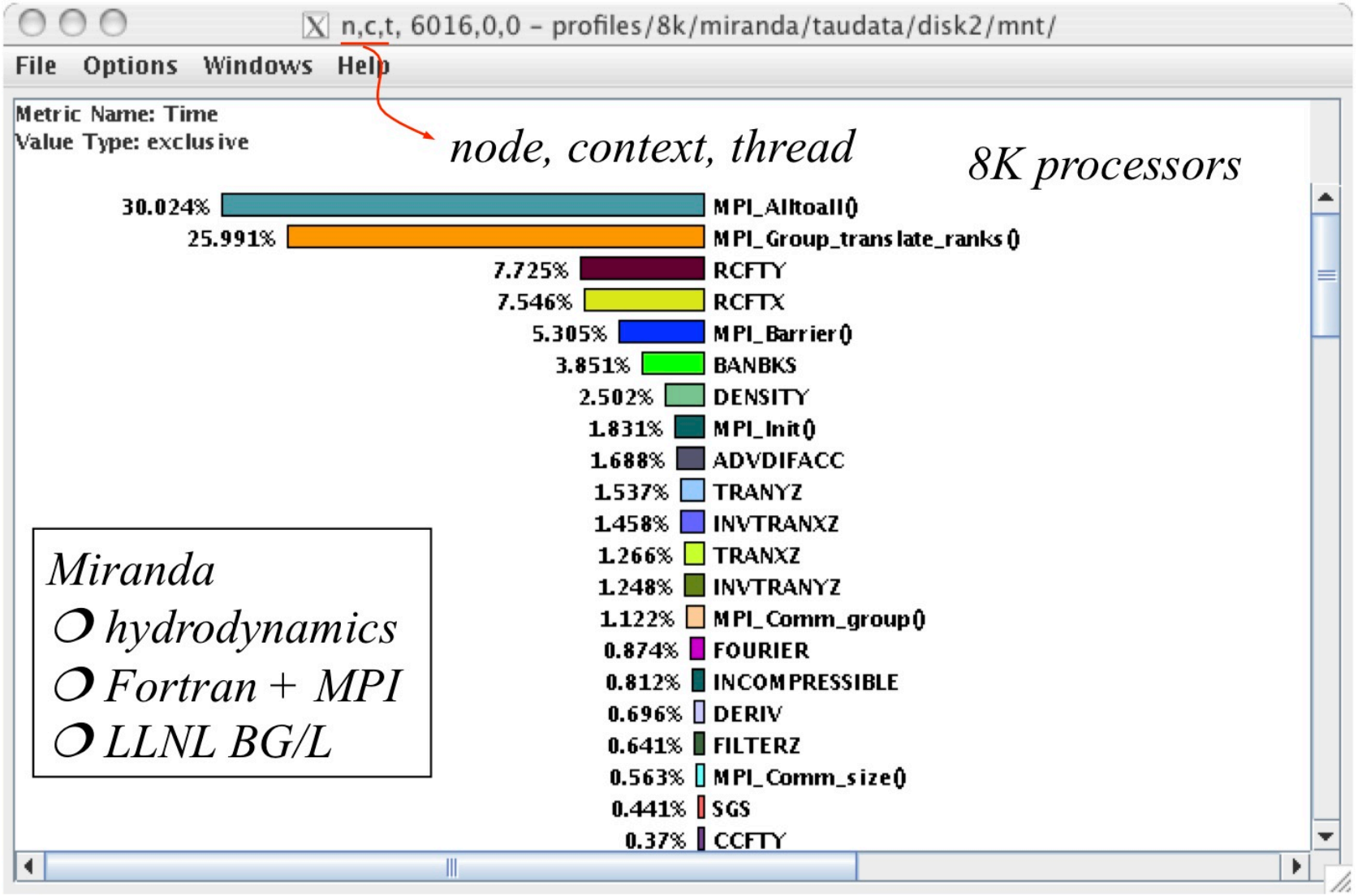
Parallel Profile Analysis – pprof



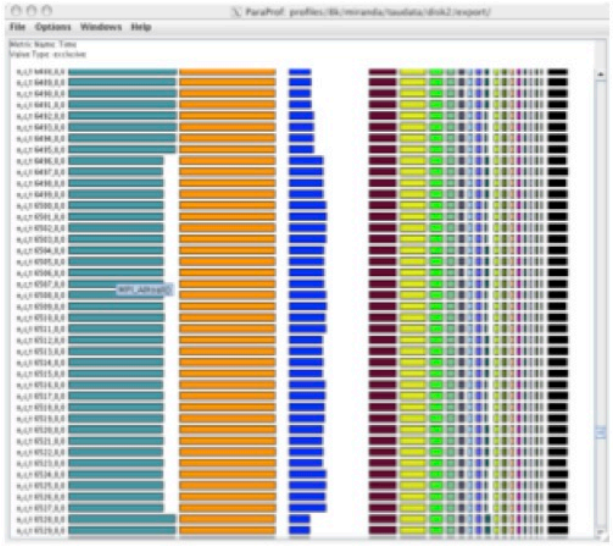
Parallel Profile Analysis – ParaProf



ParaProf – Single Thread of Execution View

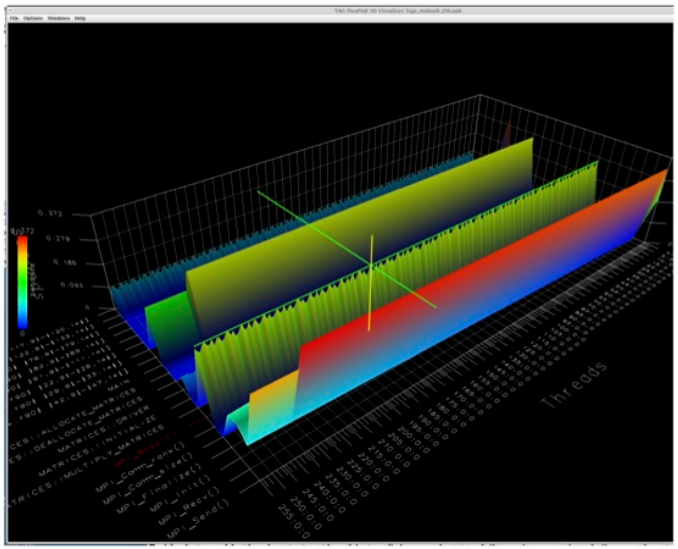


ParaProf – Full Profile / Comparative Views

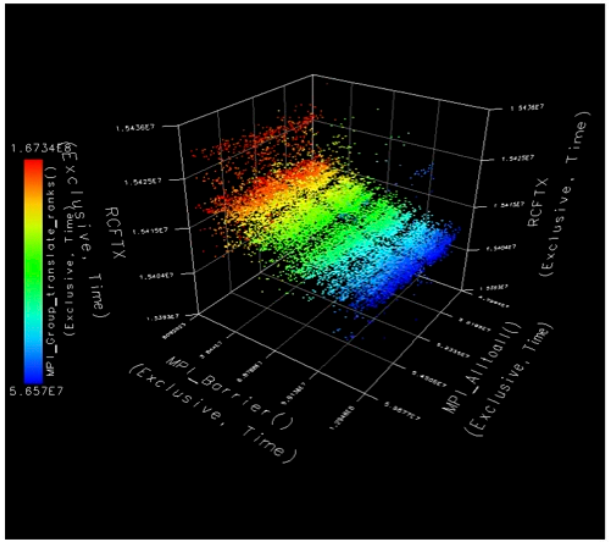


bargraph view

Full profile
- threads
- events



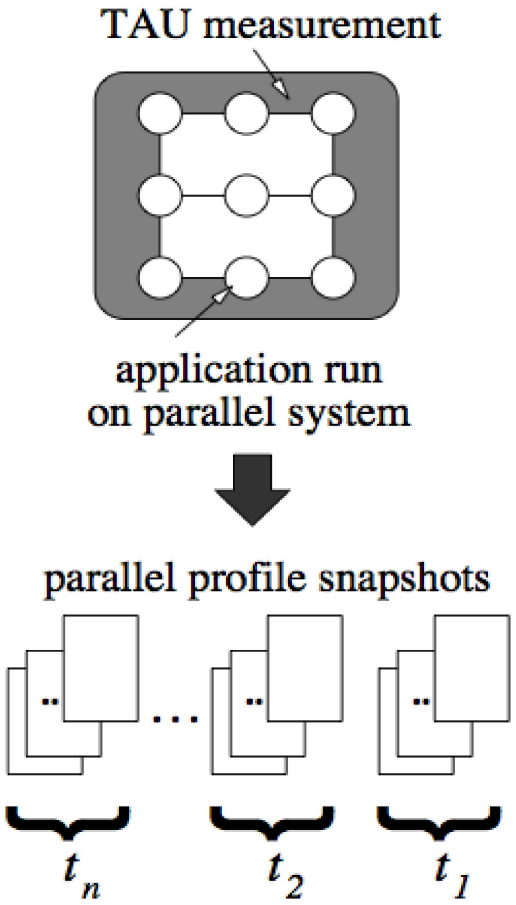
landscape view

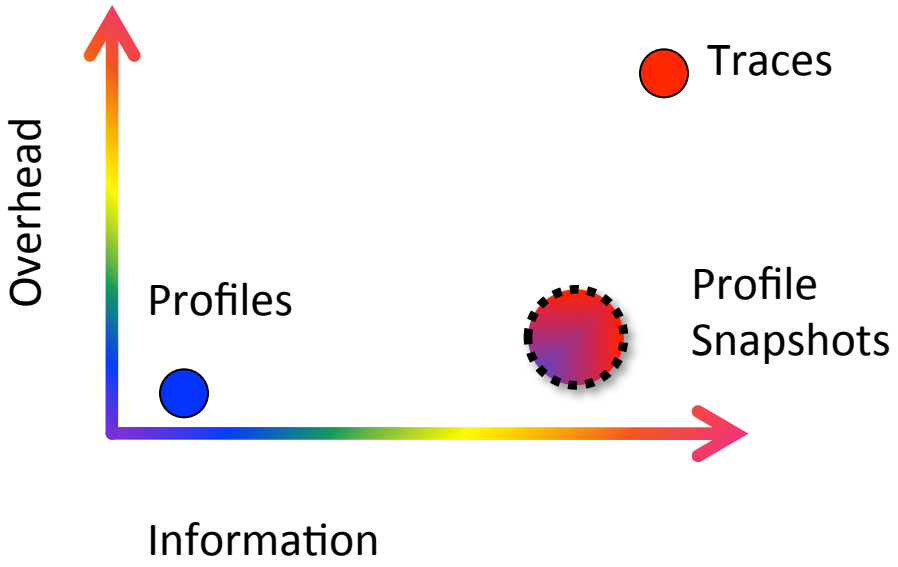


Comparative
- three event axes
- one event color

Performance Dynamics: Parallel Profile Snapshots

- Profile snapshots are parallel profiles recorded at runtime
- Shows performance profile dynamics (all types allowed)

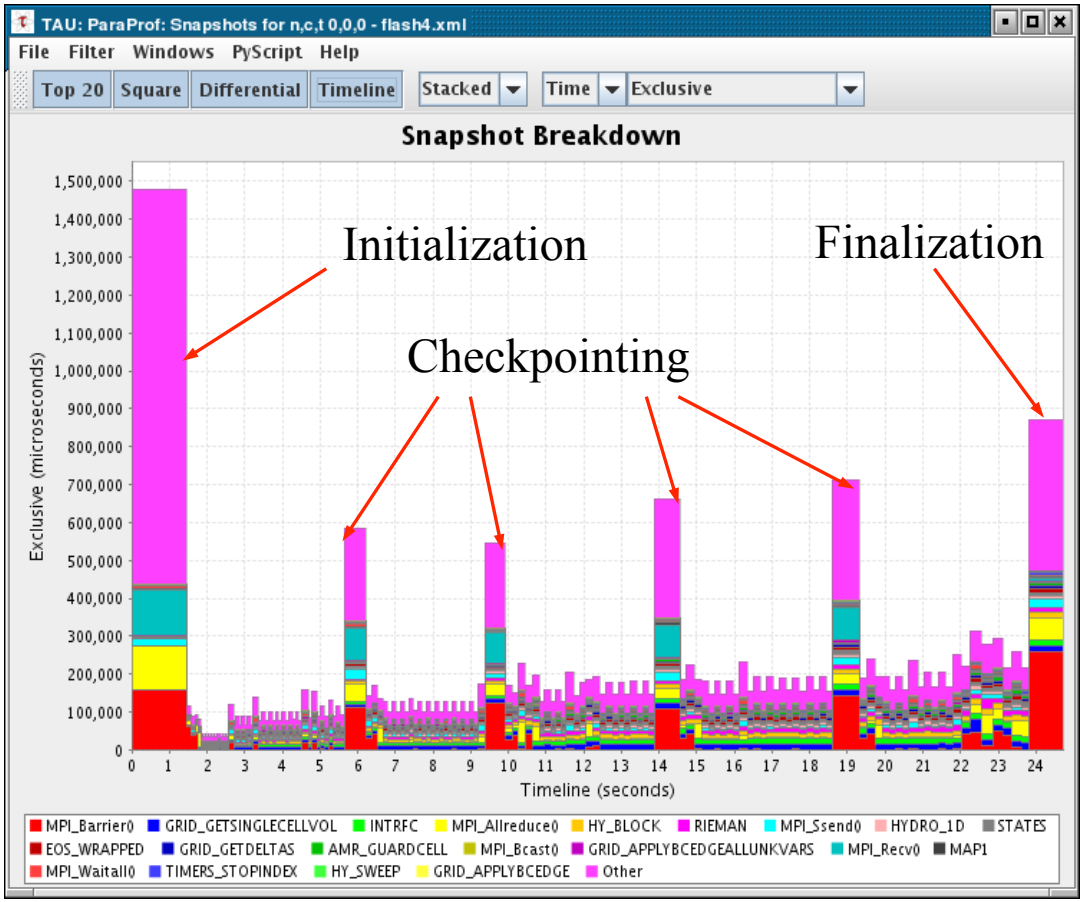




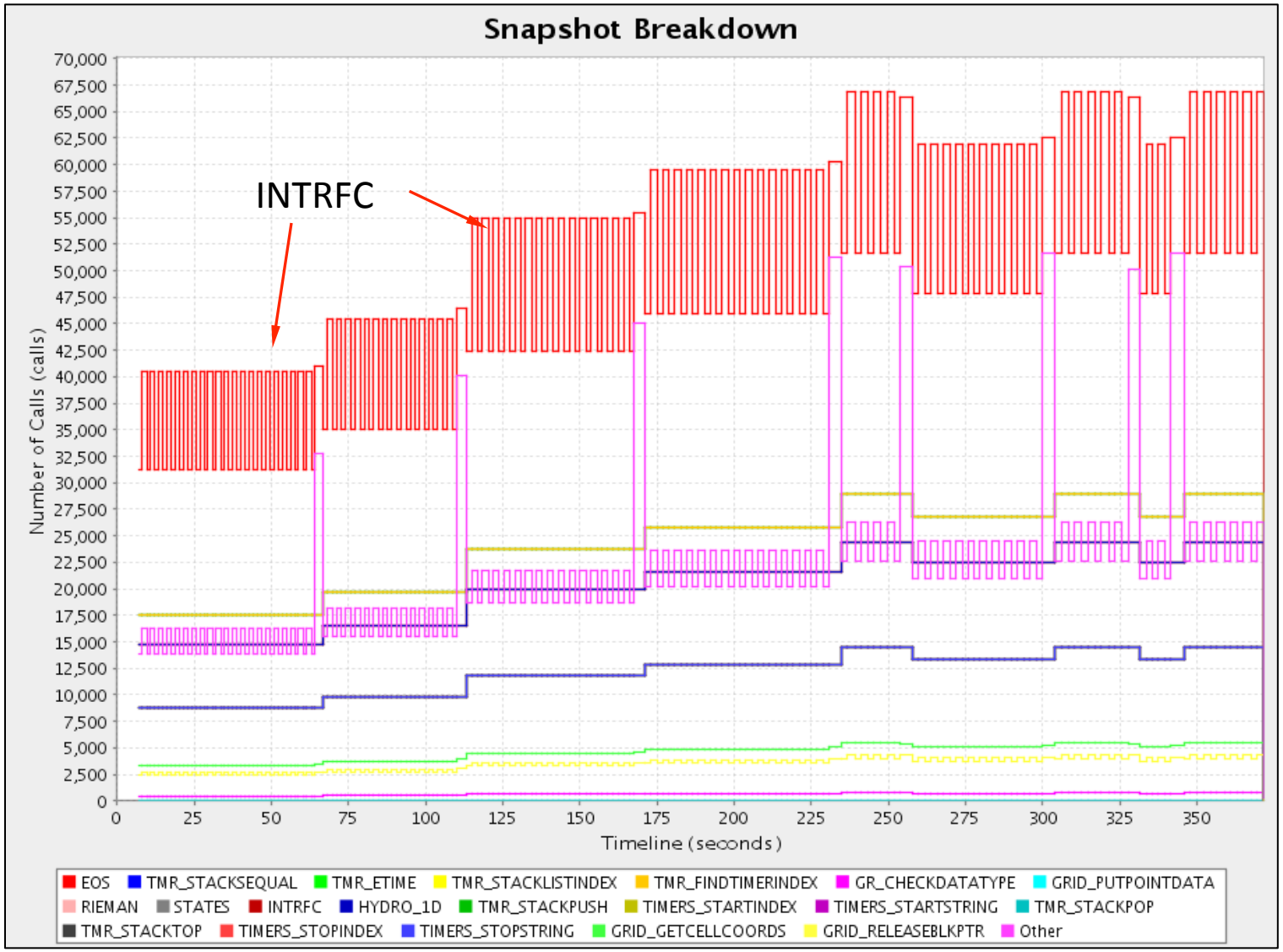
A. Morris, W. Spear, A. Malony, and S. Shende, "Observing Performance Dynamics using Parallel Profile Snapshots," European Conference on Parallel Processing (EuroPar), 2008.

Parallel Profile Snapshots of FLASH 3.0

- Simulation of astrophysical thermonuclear flashes
- Snapshots show profile differences since last snapshot
  - Captures all events since beginning per thread
  - Mean profile calculated post-mortem
  - Highlight change in performance per iteration and at checkpointing



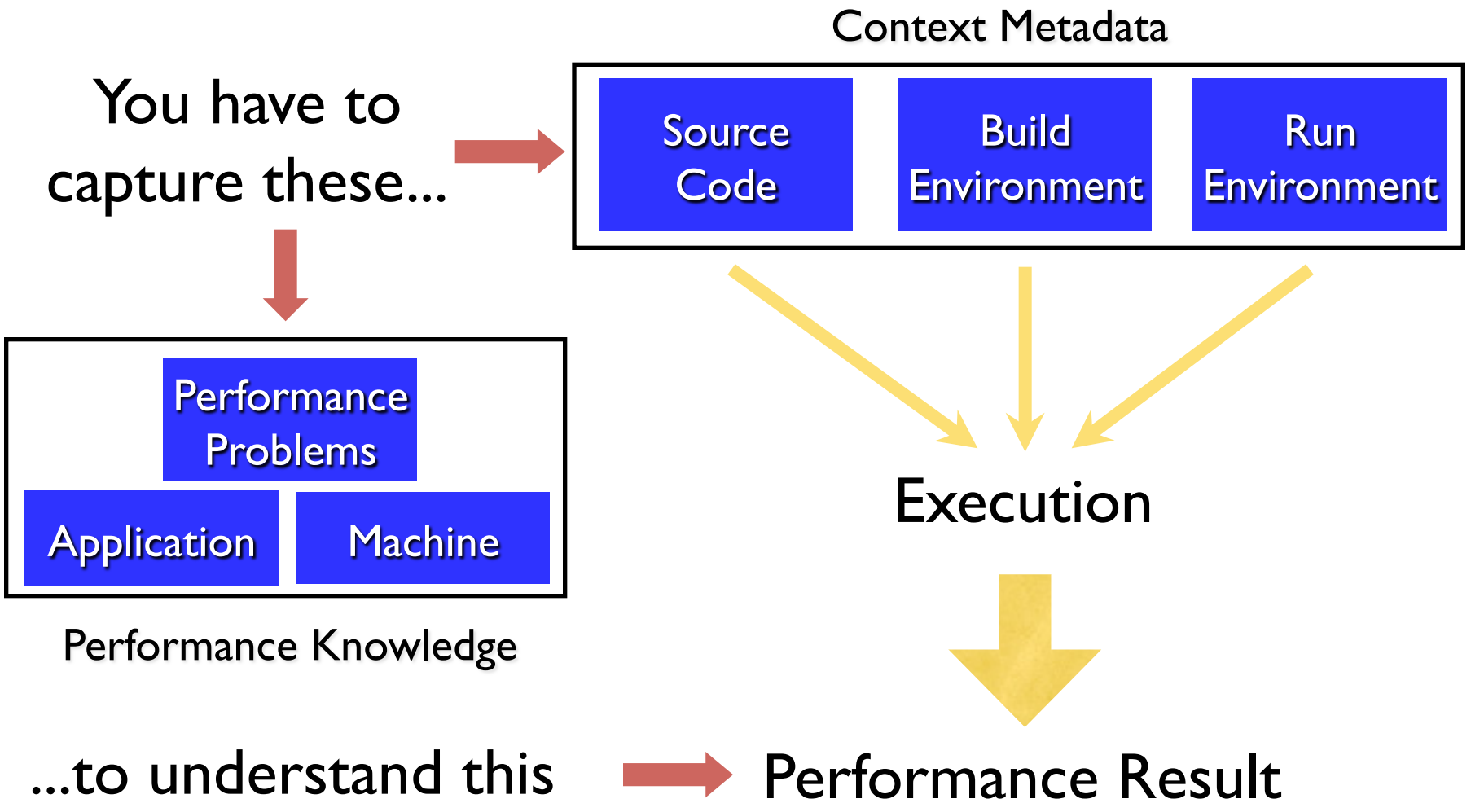
FLASH 3.0 Performance Dynamics (Periodic)



How to explain performance?

- Should not just redescribe the performance results
- Should explain performance phenomena
  - What are the causes for performance observed?
  - What are the factors and how do they interrelate?
  - Performance analytics, forensics, and decision support
- Need to add knowledge to do more intelligent things
  - Automated analysis needs good informed feedback
  - Performance model generation requires interpretation
- Build these capabilities into performance tools
  - Support broader experimentation methods and refinement
  - Access and correlate data from several sources
  - Automate performance data analysis / mining / learning
  - Include predictive features and experiment refinement

Role of Knowledge and Context in Analysis

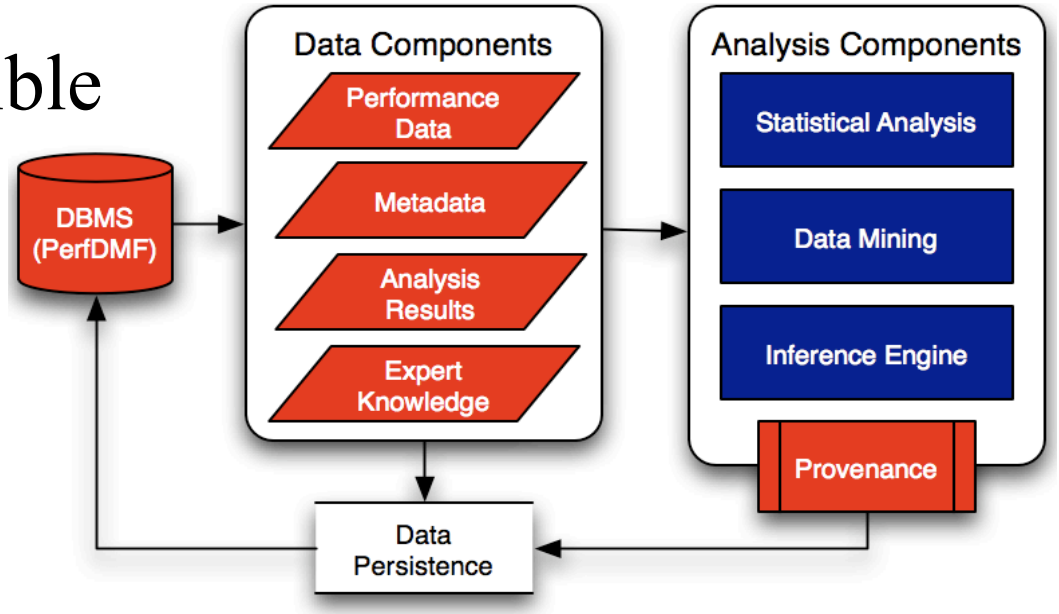


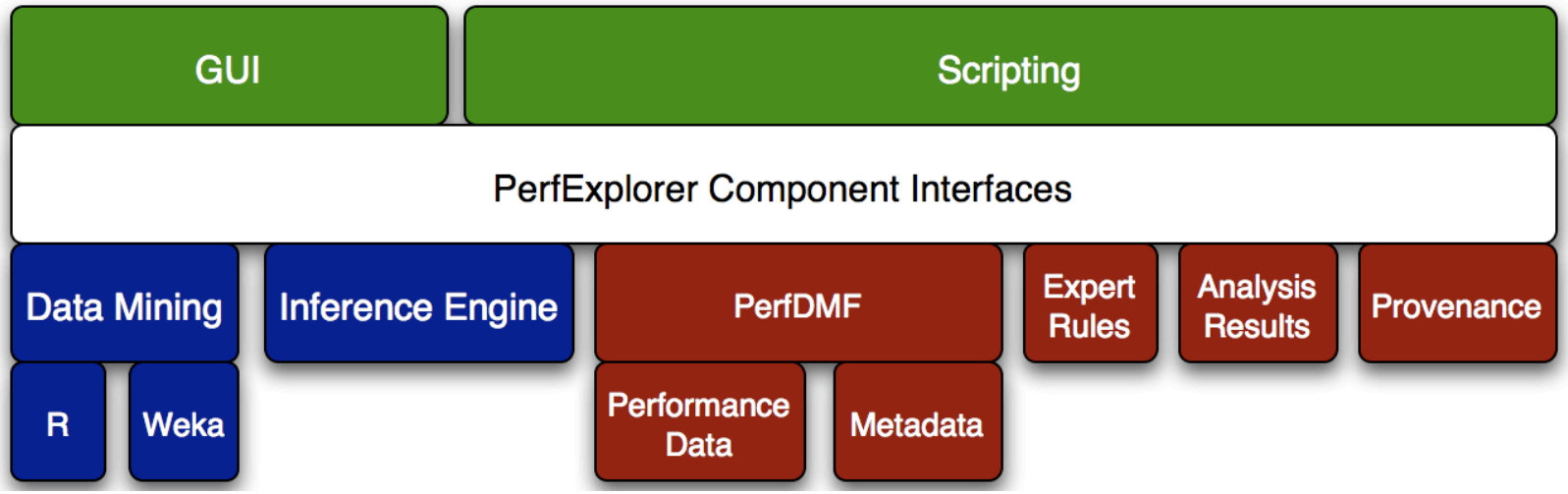
Performance Data Mining / Analytics

- Conduct systematic and scalable analysis process
  - Multi-experiment performance analysis
  - Support automation, collaboration, and reuse
- Performance knowledge discovery framework
  - Data mining analysis applied to parallel performance data
    - parametric, comparative, clustering, correlation, …
  - Integrate available statistics and data mining packages
    - Weka, R, Matlab / Octave
  - Apply data mining operations in interactive enviroment
  - Meta-analysis based on metadata collection in TAU
    - hardware/system, application, user, …

PerfExplorer Performance Data Mining

- Programmable,extensible framework to support workflow automation
- Rule-based inference for expert system analysis

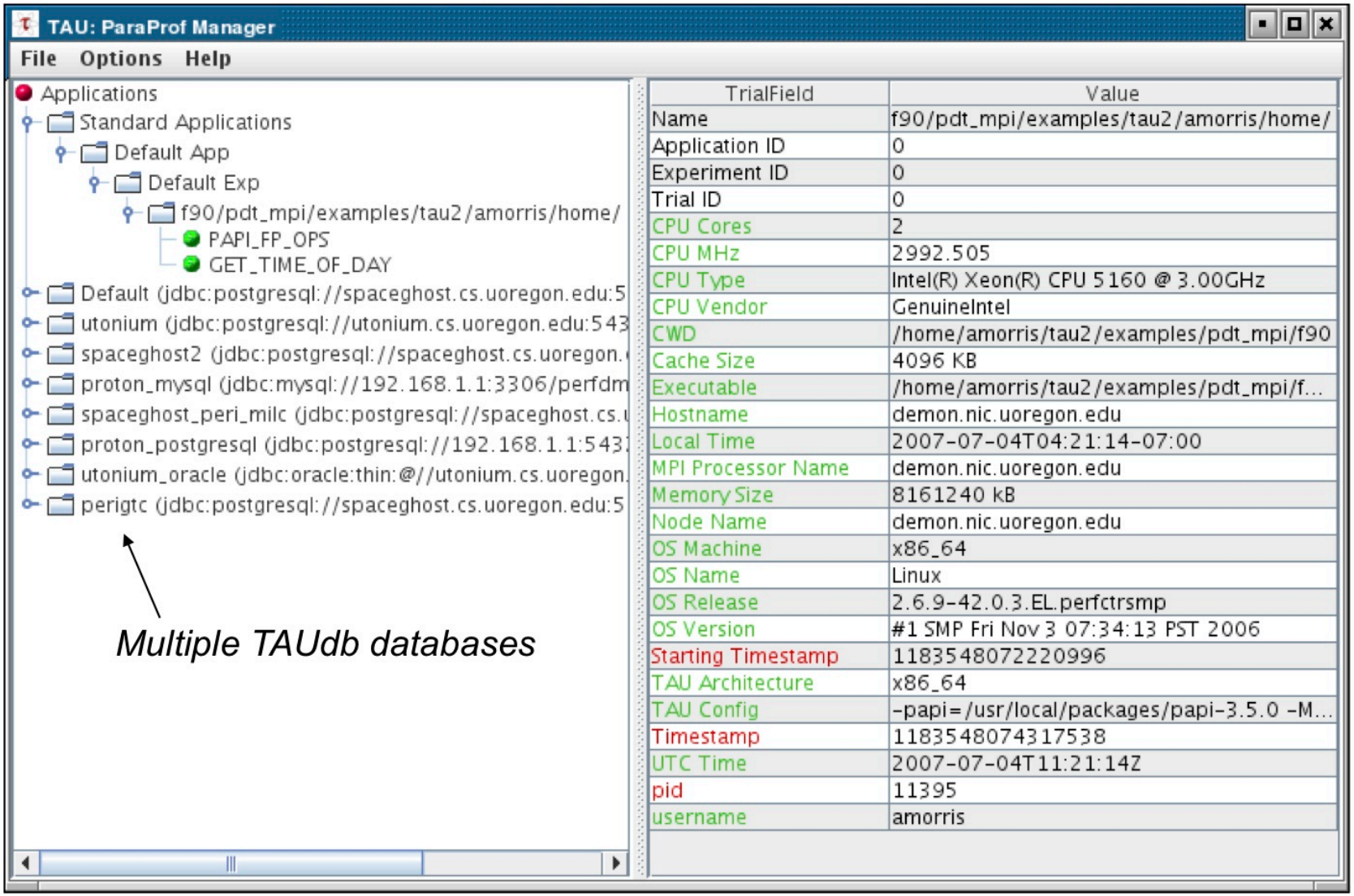




Metadata Collection

- Integration of XML metadata with parallel profile
- Three ways to incorporate metadata
  - Measured hardware/system information
    - CPU speed, memory in GB, MPI node IDs, …
  - Application instrumentation (application-specific)
    - TAU_METADATA() used to insert any name/value pair
    - Application parameters, input data, domain decomposition
  - TAUdb can load an XML file of additional metadata
    - Compiler flags, submission scripts, input files, …
- Metadata can be imported from several sources

Performance Experiment Metadata

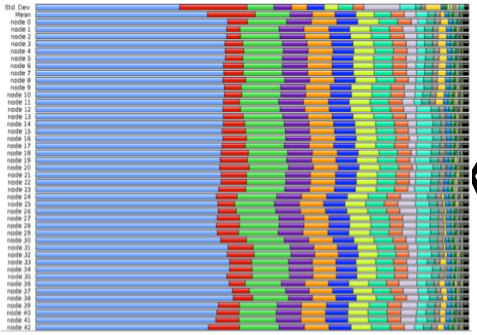


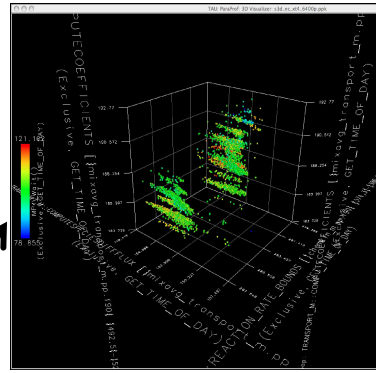
TAU Availability on New Systems

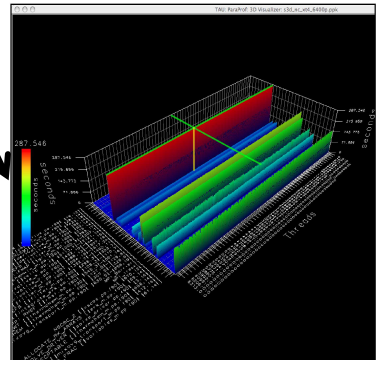
- Intel compilers with Intel MPI on Intel Xeon PhiTM (MIC)
- GPI with Intel Linux x86_64 InfiniBand clusters
- IBM BG/Q and Power 7 Linux with IBM XL compilers
- NVIDIA Kepler K20 with CUDA 5.5 with NVCC
- Fujitsu Fortran/C/C++ MPI compilers on the K computer
- PGI compilers with OpenACC support on NVIDIA systsems
- Cray CX30 Sandybridge Linux systems with Intel compilers
- AMD OpenCL libs with GNU on AMD Fusion cluster
- MPC compilers on TGCC Curie system (Bull, Linux x86_64)
- GNU on ARM Linux clusters (MontBlanc, Beacon, Stampede)
- Cray CCE compilers with OpenACC on Cray XK6, XK7
- Microsoft MPI w/ Mingw compilers on Windows Azure
- LLVM and GNU compilers under Mac OS X, IBM BG/Q

Performance Visualization

- Large performance data presents interpretation challenges
- Visualization aids in data exploration and pattern analysis
  - 3D visualization can help in identifying relations between events/metrics
- Existing tools provide "canned" views
  - TAU provides a few
    2D: bargraph, histogram
    3D: full profile, correlation

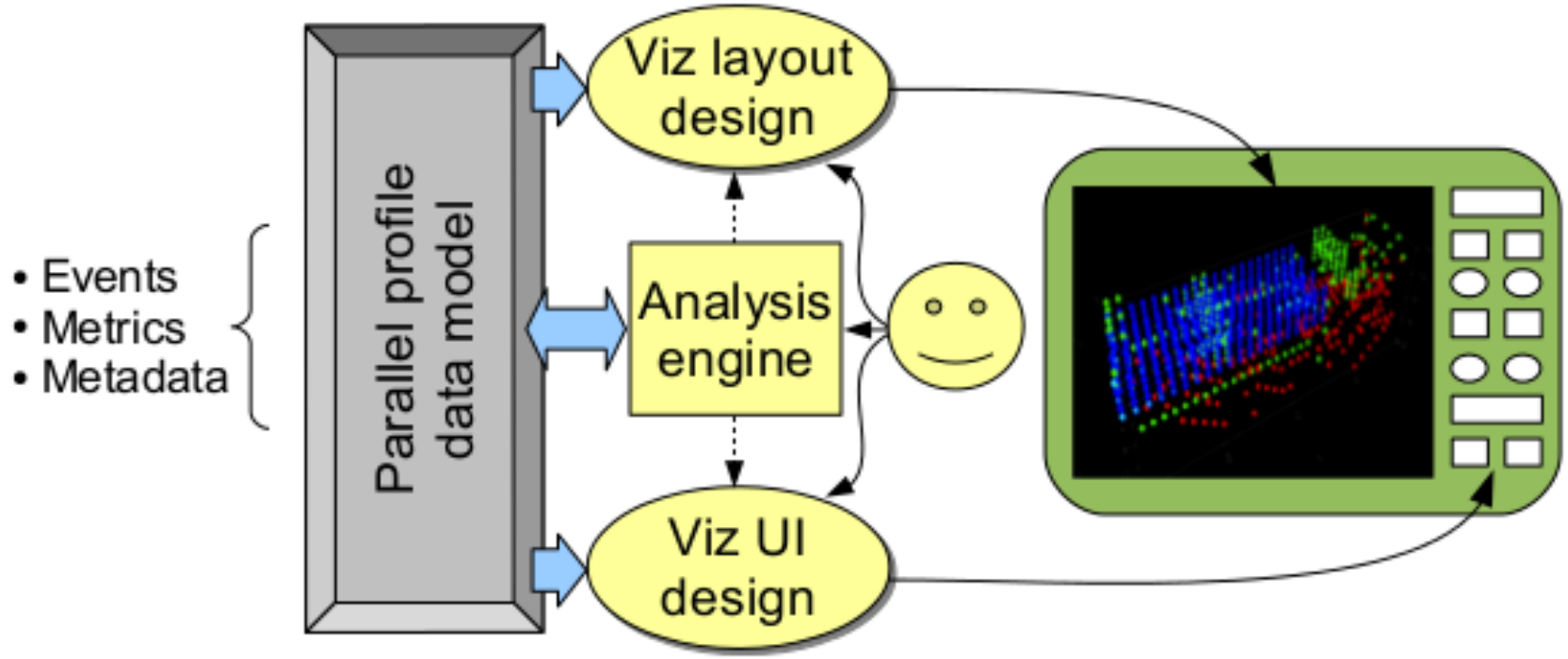






- Developing new visualizations is a challenge
  - Strategy 1: Create new view for each problem
  - Strategy 2: Use external visualization environment
- Provide high-level support to use within existing framework

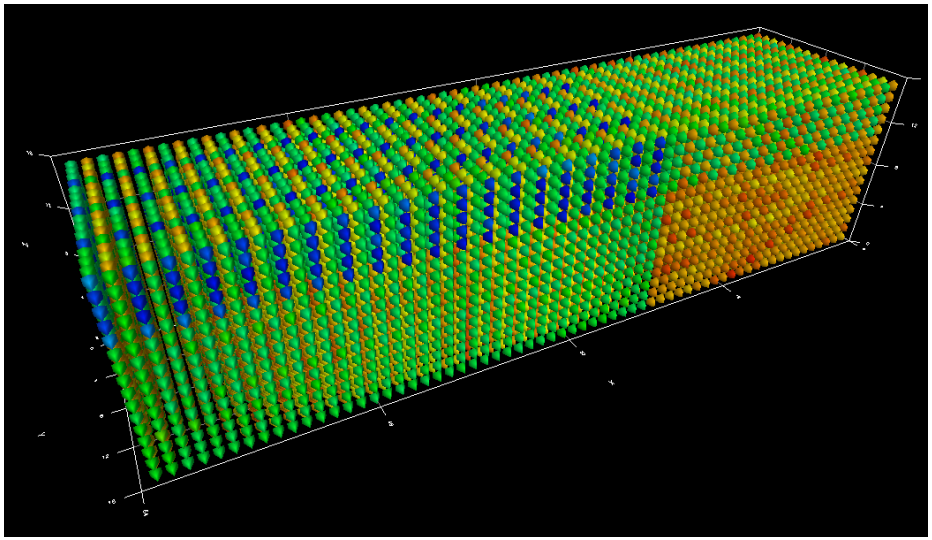
Extending Visualization Support in Profile Tool



- User defines visualization
  - Based on performance data model
- Specifies layout based on events, metrics, and metadata
- UI provides control of data binding and visualization

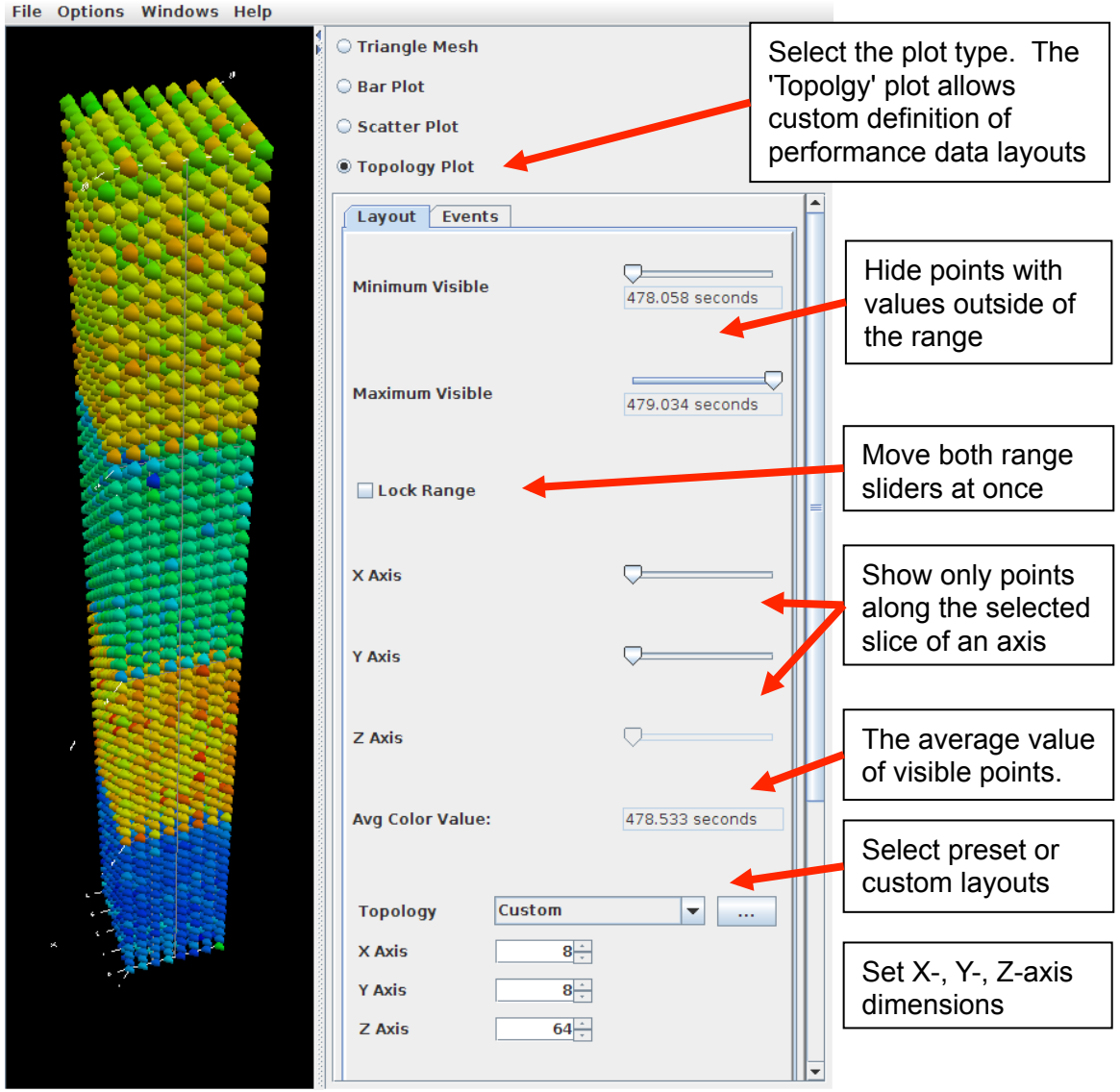
Using Process Topology Metadata

- Inspired by the Scalasca CUBE topology display
- Each point represents a thread of execution (MPI process)
  - Positioned according to the Cartesian (x,y,z,t) coordinates
- Color is determined by selected event/metric value
- Topology information can be recorded in TAU metadata
- ParaProf reads metadata to determine topology and create layout
- Sweep3D is a 3D neutron transport application
  - 16K run on BG/P
  - Color is exclusive time in the "sweep" function



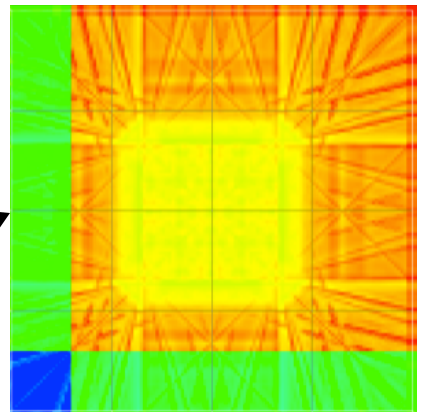
Topology Control UI

- *Layout* tab allows customization of the position and visibility of data points
- Performance event/metric data used to define color and position is selected in the *Event* tab
- Additional rendering options, such as color scale and point size are available
- 4k-core S3D run on BG/P

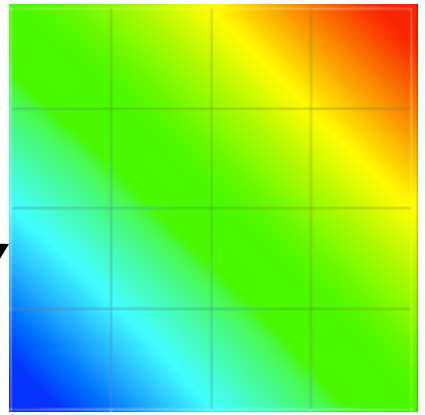


Alternate Topologies

- Certain views may hide deeper inter-process behavior
- Spatially dependent performance issues may be revealed by manipulating topology
- Sweep3D profile with alternative Cartesian mapping exposes distribution of computational effort

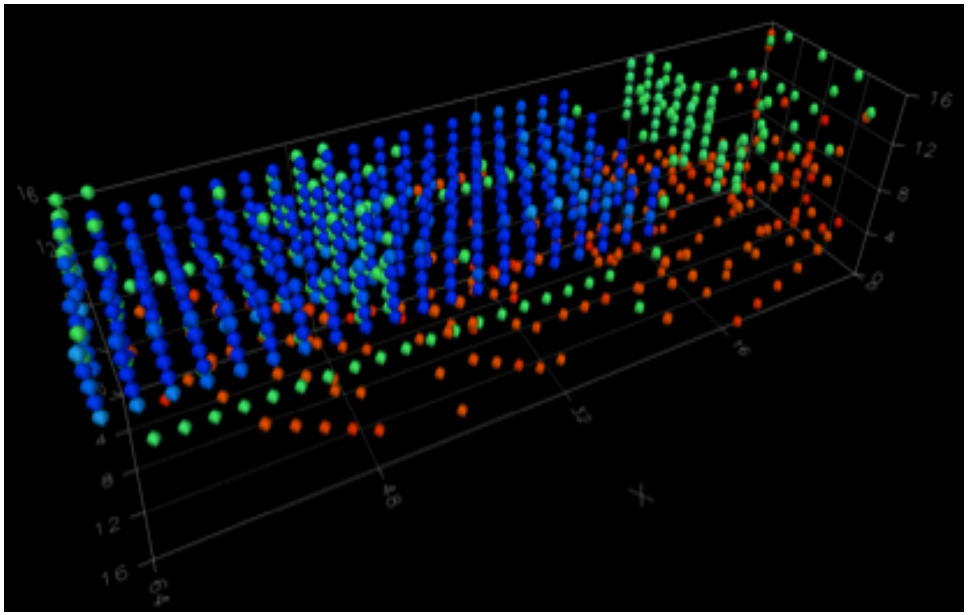


- Topology has direct effect on communication
- Visualization mapped to hardware topologies can suggest better node/rank mapping
- MPI_AllReduce() values for Sweep3D highlights waiting distribution from rank 0 (lower left) to the most distant rank (upper right)



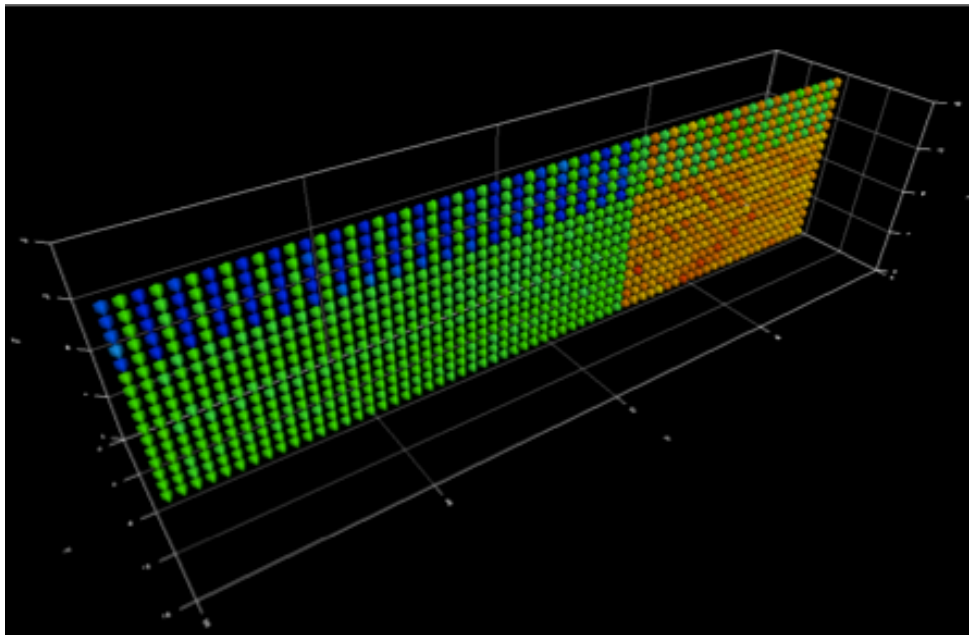
Viewing Internal Structure

- Dense topologies can hide internal structure
- Restrict visibility by color value to expose performance patterns
- ParaProf visualization UI now allows for range filtering
  - Mid-level values can be excluded
  - Remaining points are:
    - high outliers (hotspots)
    - low outliers (underutilized nodes)



Slicing to Reduce Dimensionality

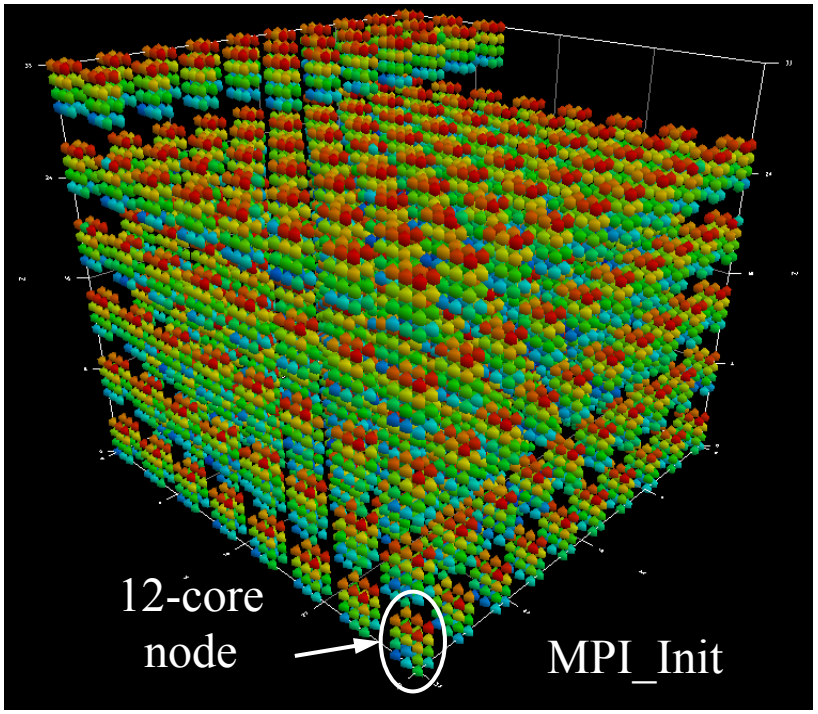
- Restrict visibility to slices along the spatial axes
- Multiple axis controls allow selection of planes, lines, or an individual point

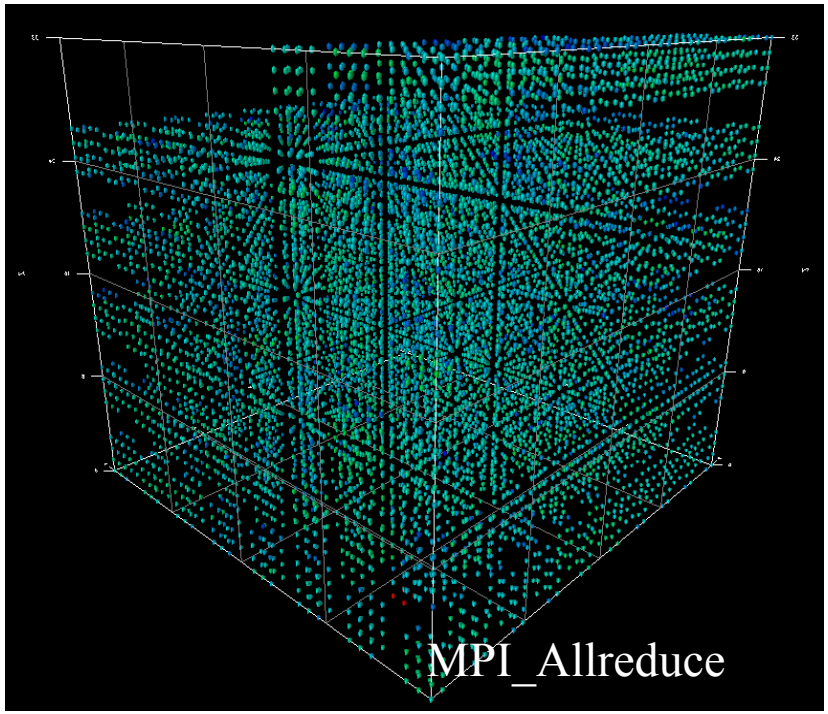


- ParaProf visualization UI provides filtering control
  - Averaging the color value for all points in the selected area

Cray XE6 Topology

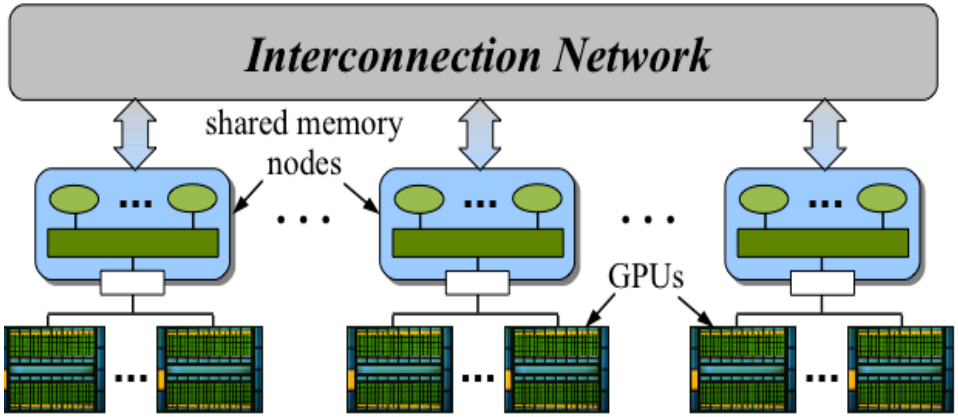
- Now have ability to capture Cray XE6 topology metadata
  - Node topology and core identity
- GCRM is a global cloud resolving mode
- Visualization of 10K execution
  - Topological layout attempts to capture node cores





Heterogeneous Parallel Performance

- Heterogeneous parallel systems are very relevant today
  - Multi-CPU, multicore shared memory nodes
  - Manycore (throughput) accelerators
  - Cluster interconnection network



- Performance is the main driving concern
- Heterogeneity is an important path to extreme scale
- Heterogeneous software technology to get performance
  - More sophisticated parallel programming frameworks
  - Integrated parallel performance tools
    - support heterogeneous performance model and perspectives

Implications for Performance Tools

- Current status quo is somewhat comfortable
  - Mostly homogeneous parallel systems and software
  - Shared-memory multithreading – OpenMP
  - Distributed-memory message passing – MPI
- Parallel computational models are relatively stable
  - Corresponding performance models are tractable
  - Parallel performance tools can keep up and evolve
- Heterogeneity creates richer computational potential
  - Greater performance diversity and complexity
- Heterogeneous system environments
- Performance tools have to support richer computation models and more versatile performance perspectives
- Will utilize more sophisticated programming and runtime

Heterogeneous Performance Views

- Want to create performance views that capture heterogeneous concurrency and execution behavior
  - Reflect interactions between components
  - Capture performance semantics of computation
  - Assimilate performance for all execution paths
- Existing parallel performance tools are CPU (host)-centric
  - Event-based sampling
  - Direct measurement (instrumentation of events)
- What perspective does the host have of other components?
  - Determines semantics of the measurement data
  - Determines assumptions about behavior and interactions
- Performance views may have to work with reduced data

TAU for Heterogeneous Performance Analysis

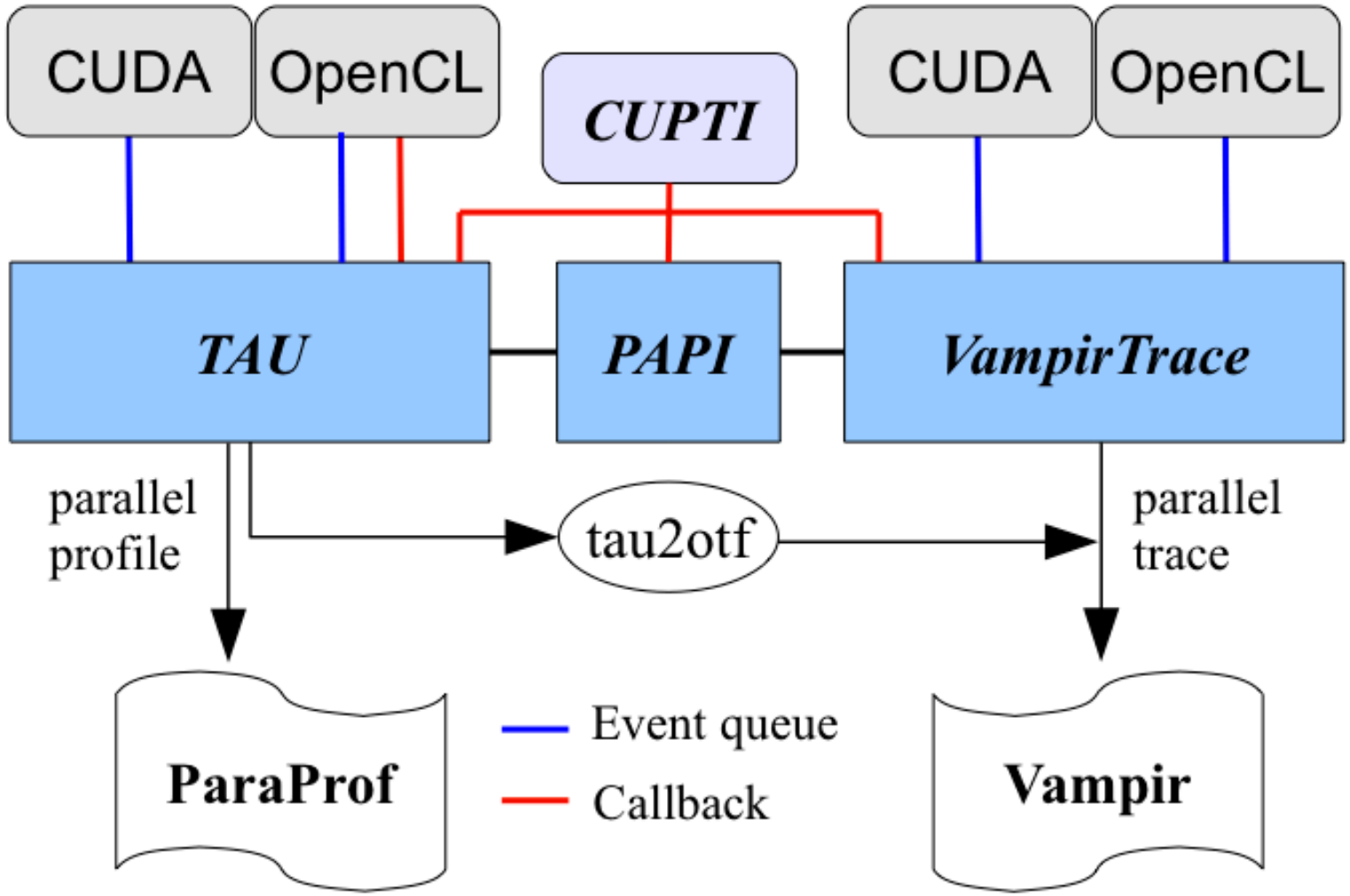
- Extend TAU for heterogeneous performance analysis
- Integrate Host-GPU support in TAU measurement
  - Enable host-GPU measurement approach
    - target CUDA and NVIDIA GPUs
    - utilize CUPTI and PAPI CUDA
    - interface with compilers (PGI)
  - Provide both heterogeneous profiling and tracing
    - contextualization of asynchronous kernel invocation
- Additional support
  - TAU wrapping of libraries (*tau_gen_wrapper*)
  - Work with library preloading *(tau_exec*)

A. Malony, et al., "Parallel Performance Measurement of Heterogeneous Parallel Systems with GPUs," International Conference on Parallel Processing (ICPP 2011), Taipei, Taiwan, 2011.

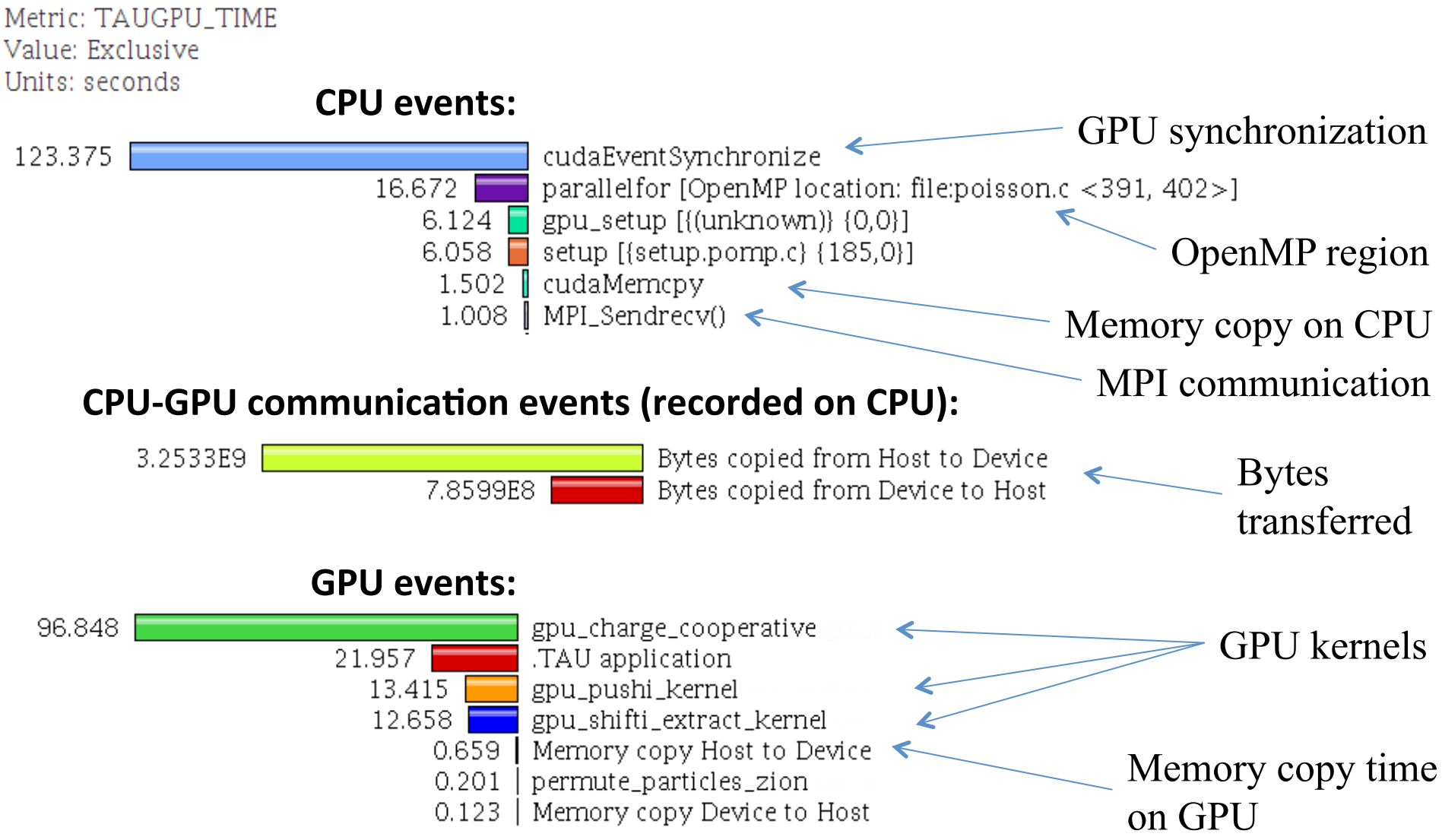
How TAU GPU Support Works

- Approach 1: CUDA library wrapping + CUDA events
  - CUDA driver or runtime API are intercepted
- Approach 2: CUDA Profiling Tools Interface (CUPTI)
  - Callback API – a callback is registered for every API call
  - Counter API -- GPU device counters (device-level)
  - Activity API – kernel and memory copy timing information
- With both of these methods, TAU records the activity that occurred on the GPU at a synchronization point
- Performance data is merged with TAU profile/trace
  - Divided between CPU and GPU
  - Each GPU stream/context presented as a separate thread

GPU Performance Tool Interoperability



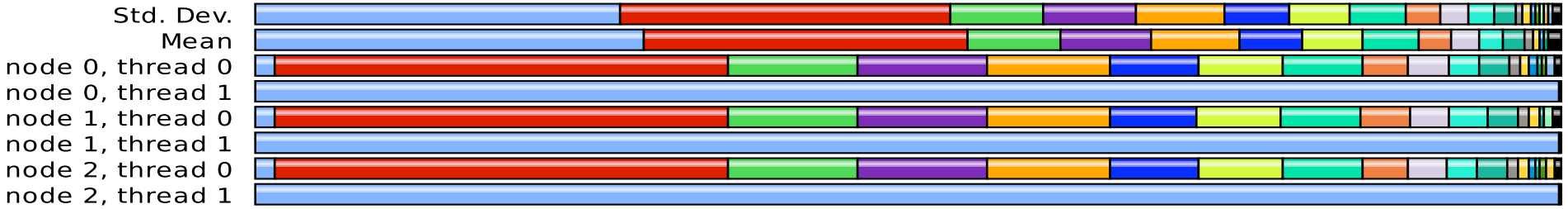
A TAU Profile with GPU Performance Data



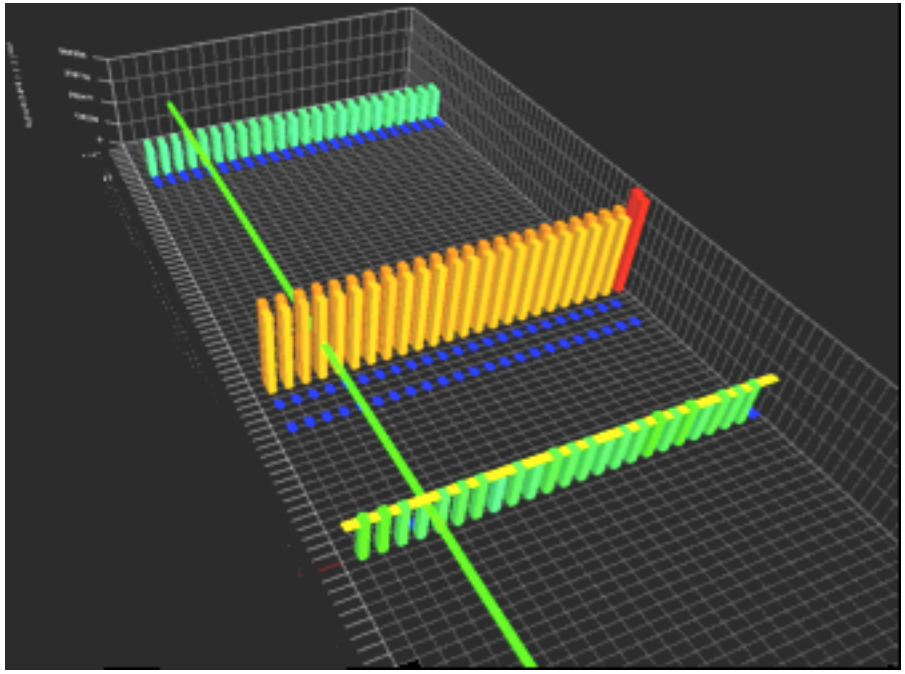
SHOC Stencil2D Parallel Profile

Metric: TAUGPU_TIME
Value: Exclusive

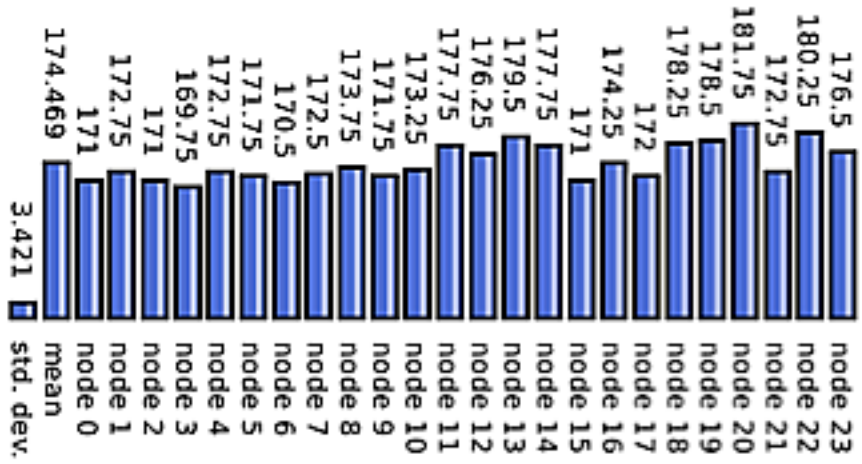
3 GPUs



24 GPUs

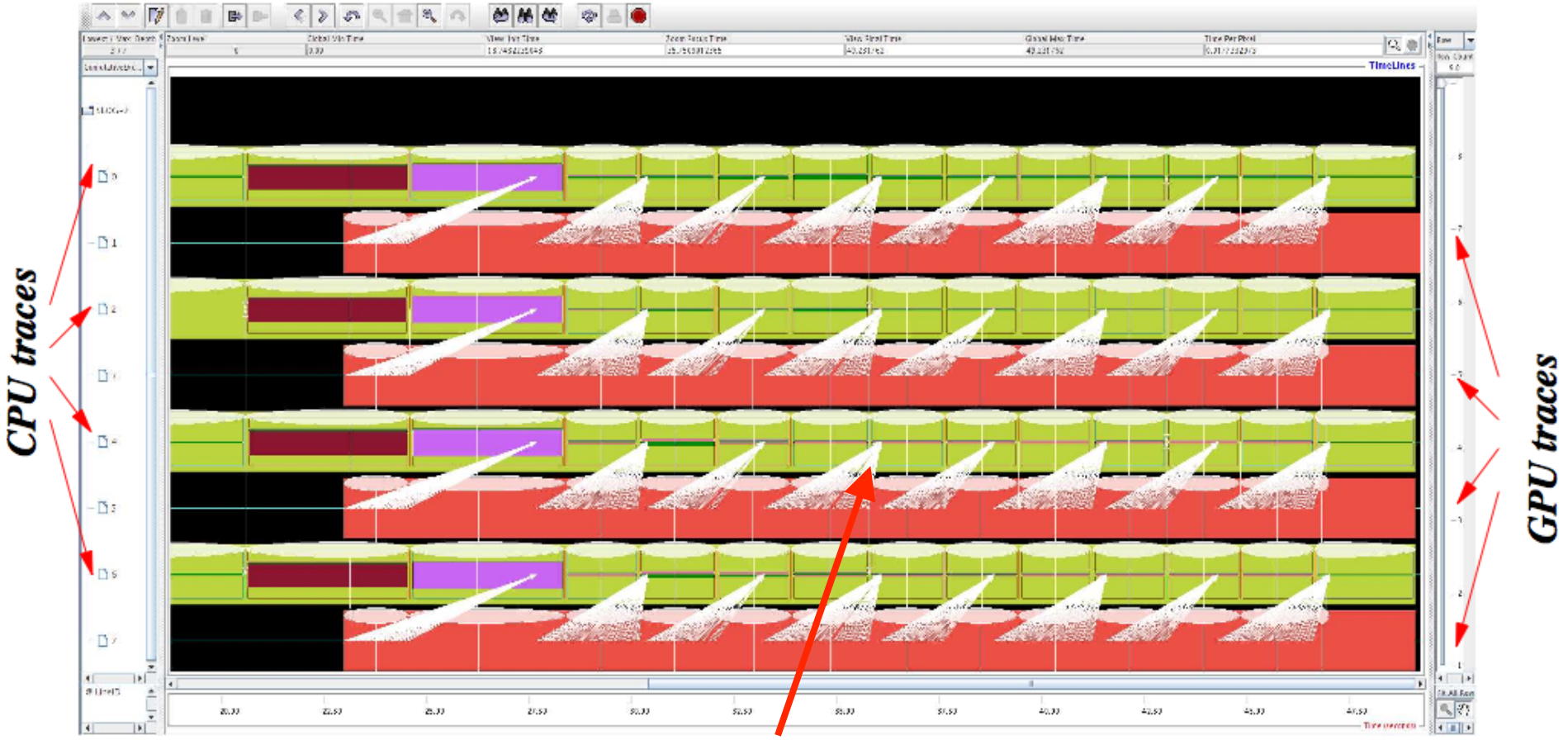


Name: StencilKernel
Metric Name: TAUGPU_TIME
Value: Exclusive
Units: microseconds



SHOC Stencil2D Trace (4 CPUxGPU)

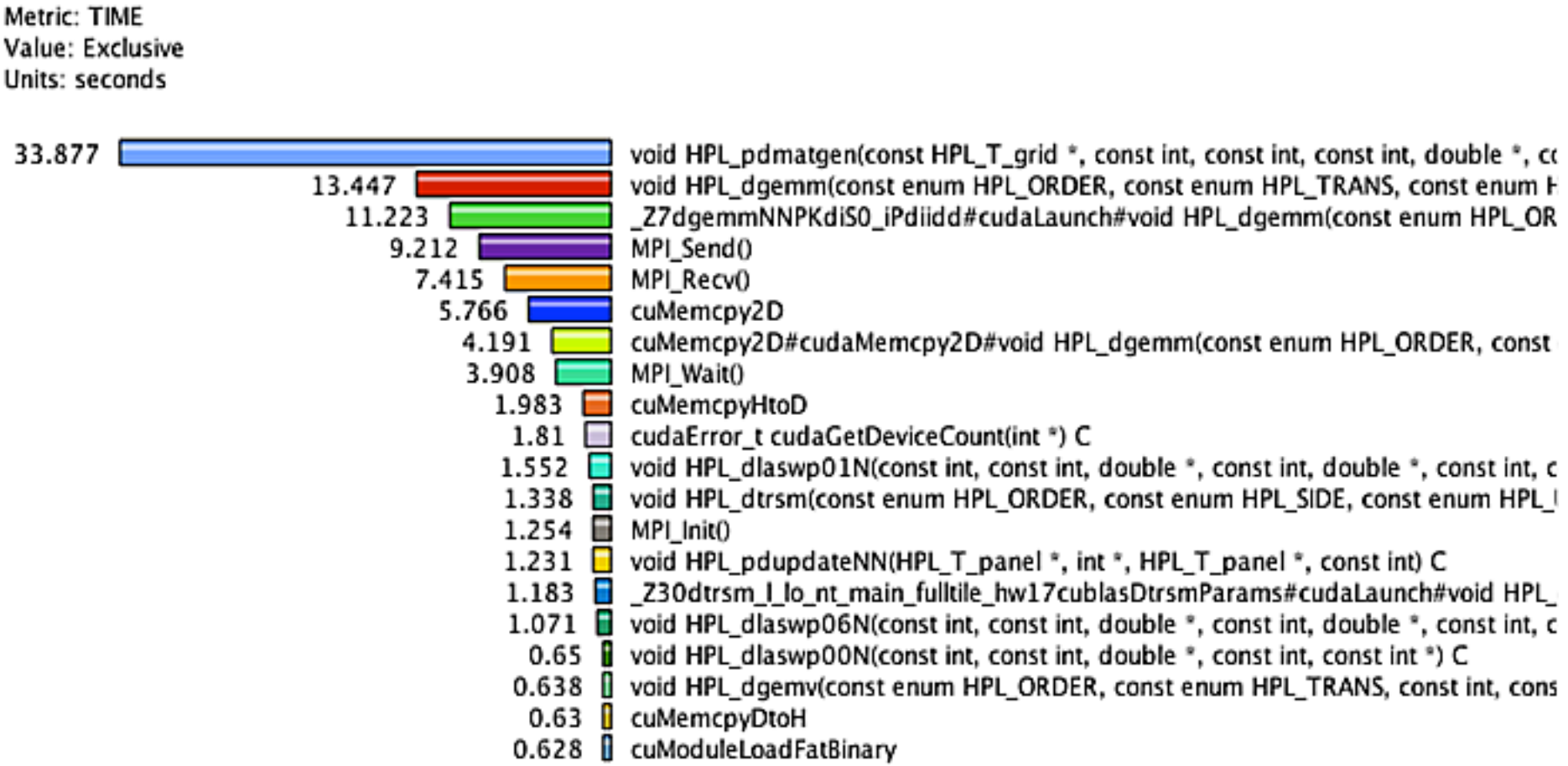
- Visualization using Jumpshot (Argonne)



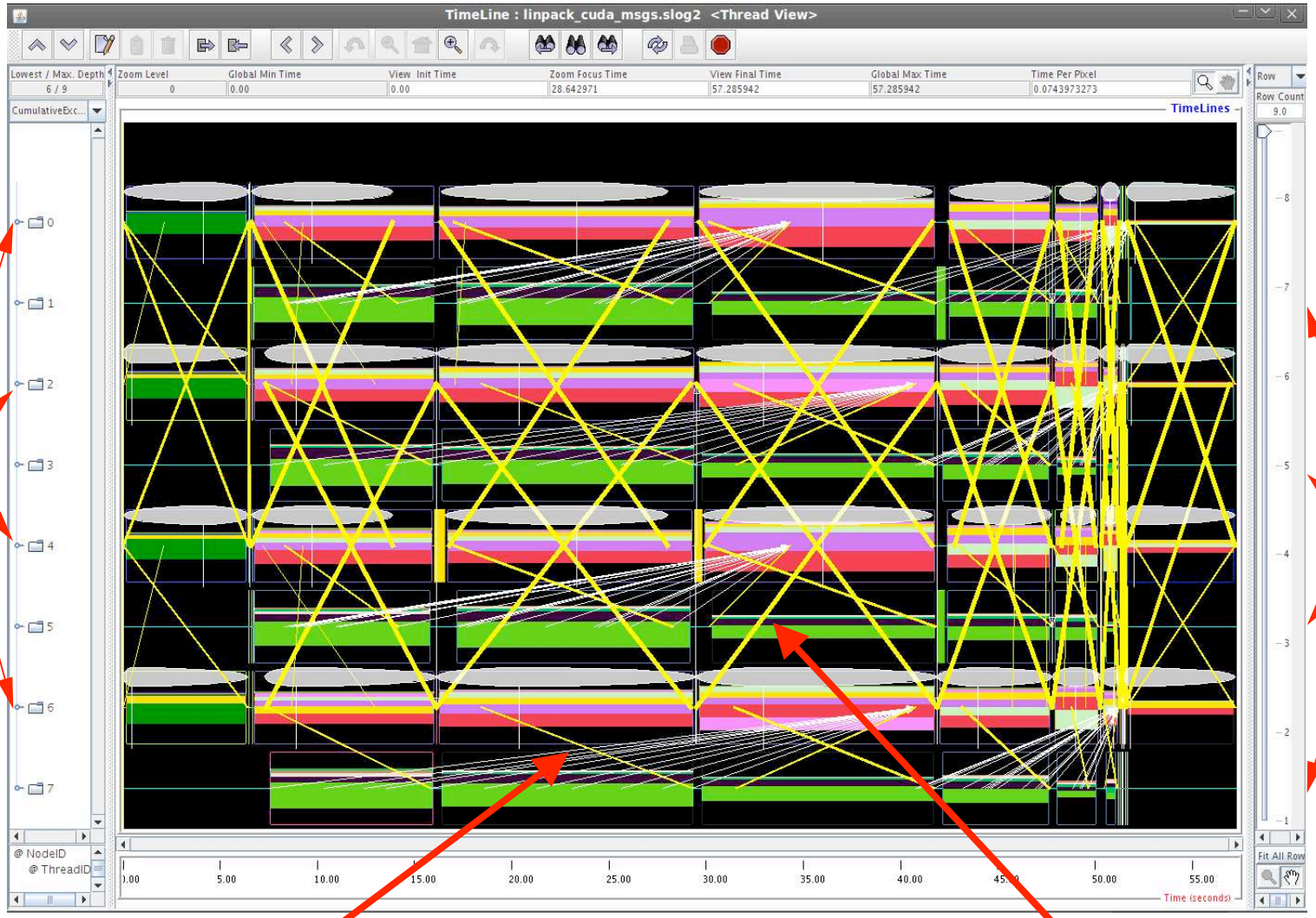
CUDA memory transfer (white)

CUDA Linpack Profile (4 CPUxGPU)

- GPU-accelerated Linpack benchmark (NVIDIA)

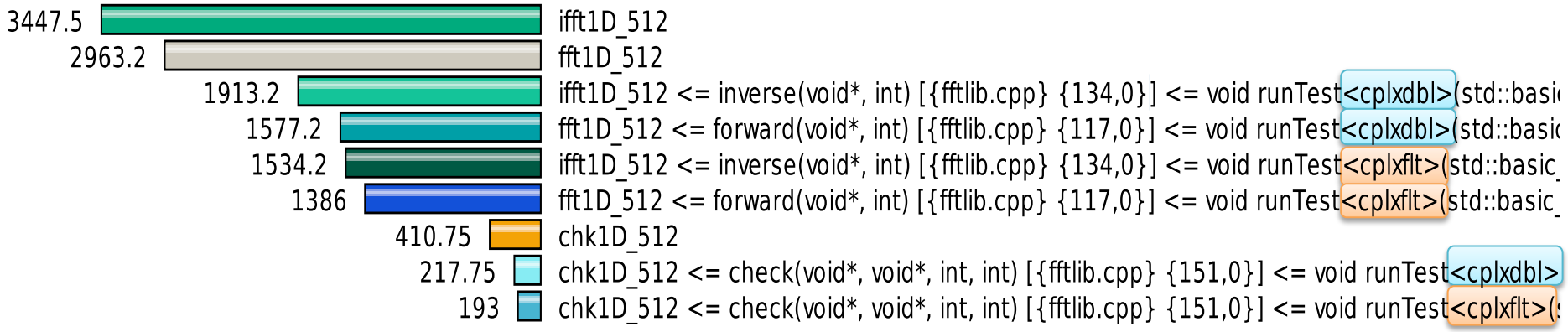


CUDA Linpack Trace



SHOC FFT Profile with Callsite Info

- Callsite information breaks up kernel execution time by where in the application the kernel was launched.
  - Callsite information links the kernels on the GPU with functions that launch them on the CPU.
  - Callsite paths can be thought of as an extension of a callpath spanning both CPU and GPU.
  - Three kernels: ifft1D_512, fft1D_512, chk1D_512
  - Callsite shows single or double precession step



TAU and OpenACC

- CUPTI works with OpenACC applications
- However the operations the compiler (PGI, Cray)does to generate code is hidden
- We can retrieve this information by integrating into the compiler directly
- Working with the PGI compiler to discover which OpenACC regions are responsible for which GPU events

PGI Compiler ACC integration

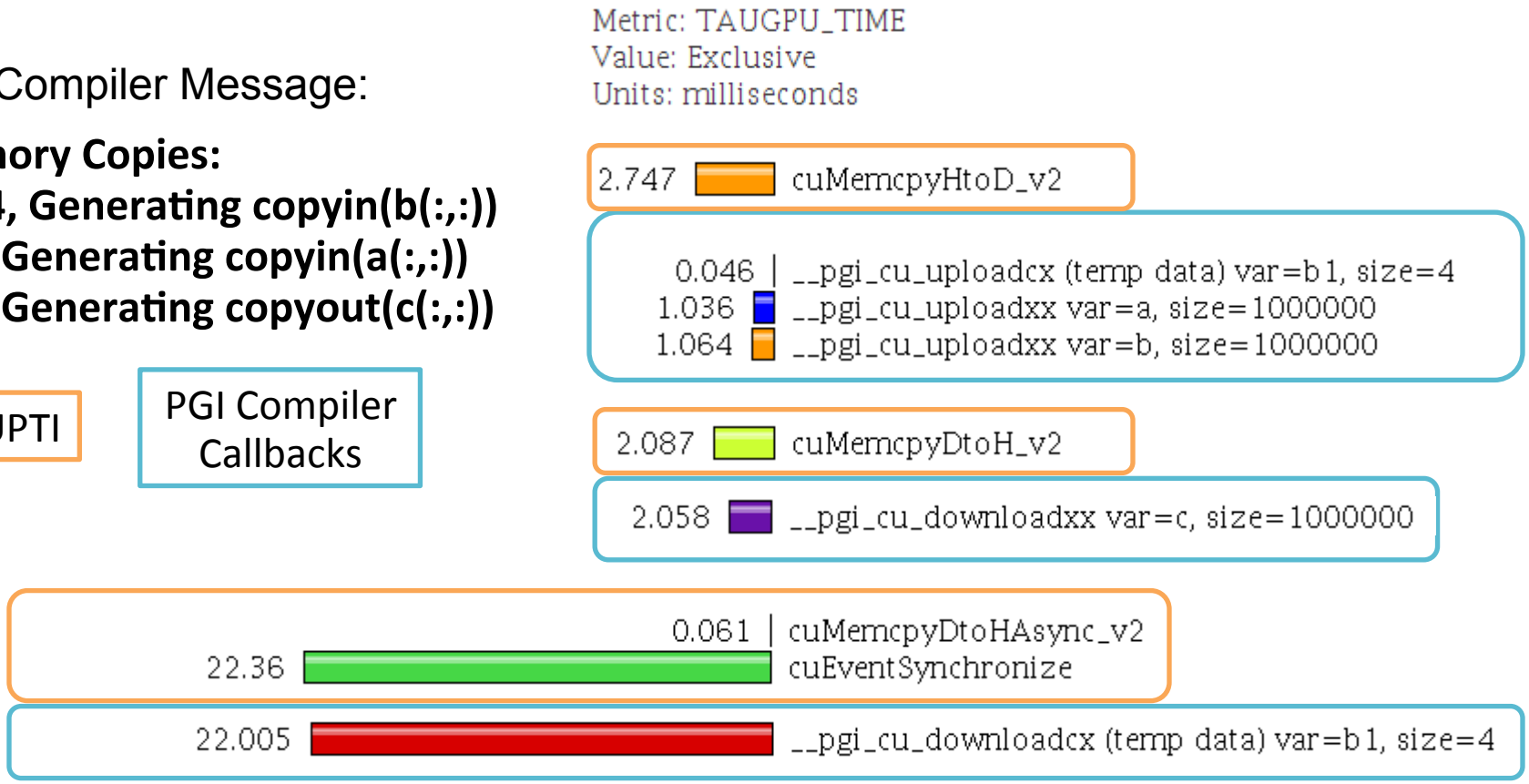
- Memory copies recorded by CUPTI are anonymous
- Collect information from the PGI compiler

PGI Compiler Message:

Memory Copies:
34, Generating copyin(b(:,:))
Generating copyin(a(:,:))
Generating copyout(c(:,:))

CUPTI

PGI Compiler Callbacks



CUDA Kernel Tracking

- Track register usage for each kernel
  - Local, shared and device registers
- Familiar technique of sharing blocks of memory on the GPU is capture by this feature
- GPU occupancy is calculated for each kernel automatically

Compare Two Matrix Multiply Cases

- Simple
  - No use of shared memory

```
multiply_matrices(float *d_a, float *d_b, float *d_c, int lda)
...
for (unsigned int j=0; j<M; j++) {
  ctemp = ctemp + d_a[idx(row,j,lda)] * d_b[idx(j,col,lda)];
}
d_c[id] = ctemp;
...
```

- Improved
  - Utilize the shared memory
  - Better use of registers

```
multiply_matrices_shared_blocks(float *d_a, float *d_b, float *d_c, int lda)
...
for (int k = 0; k < (M / bs); k++) {
  //form submatrices
  a[sub_row][sub_col] = sub_a[idx(sub_row, sub_col, lda)];
  b[sub_row][sub_col] = sub_b[idx(sub_row, sub_col, lda)];
  //wait for all threads to complete copy to shared memory.
  __syncthreads();
  //multiply each submatrix
  for (int j=0; j < bs; j++) {
    c = c + a[sub_row][j] * b[j][sub_col];
  }
  // move results to device memory.
  d_c[id] = c;
  // wait for multiplication to finish before moving onto the next submatrix
  __syncthreads();
...
```

CUDA Device Memory Profile

Use of shared memory

Increase in register usage

multiply_matrices_shared_blocks(float*, float*, float*, int)						
Block Size	256	1	256	256	256	0
Local Memory (bytes per thread)	0	1	0	0	0	0
Local Registers (per thread)	23	1	23	23	23	0
Shared Dynamic Memory (bytes)	0	1	0	0	0	0
Shared Static Memory (bytes)	2,048	1	2,048	2,048	2,048	0
multiply_matrices(float*, float*, float*, int)						
Block Size	256	1	256	256	256	0
Local Memory (bytes per thread)	0	1	0	0	0	0
Local Registers (per thread)	12	1	12	12	12	0
Shared Dynamic Memory (bytes)	0	1	0	0	0	0
Shared Static Memory (bytes)	0	1	0	0	0	0

Improved

Simple

Name ▽	Exclusive TAUGPU_TIME	Inclusive TAUGPU_TIME	Calls	Child Calls
multiply_matrices_shared_blocks(float*, float*, float*, int)	12.666	12.666	1	0
multiply_matrices(float*, float*, float*, int)	43.414	43.414	1	0
Memory copy Host to Device	2.079	2.079	2	0
Memory copy Device to Host	3.604	3.604	2	0
.TAU application	1.507	63.271	1	6

2.4x speedup by using shared memory

CUDA Kernel Occupancy

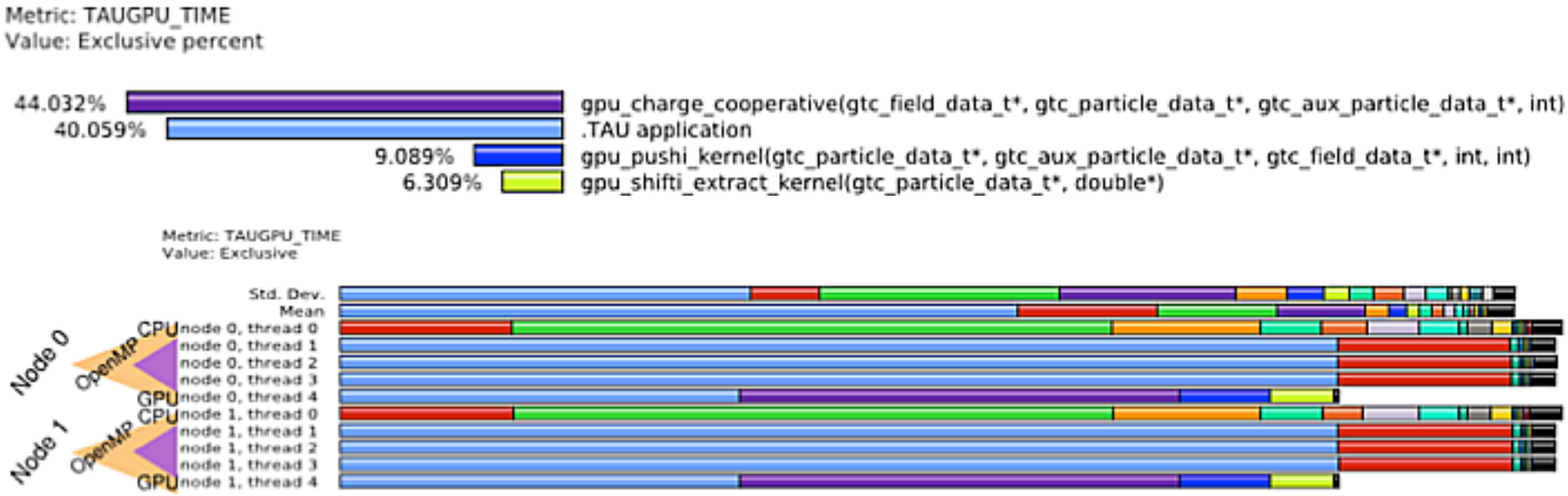
83% of 48 total wraps occupied

Block per SM limited by the number of registers used

Name	Total △	NumSamples
multiply_matrices_shared_blocks(float*, float*, float*, int)		
Allocatable Blocks Per SM given Registers used (Blocks)	5	1
Allocatable Blocks per SM given Thread count (Blocks)	6	1
Allocatable Blocks Per SM given Shared Memory usage (Blocks)	24	1
GPU Occupancy (Warps)	40	1
multiply_matrices(float*, float*, float*, int)		
Allocatable Blocks per SM given Thread count (Blocks)	6	1
Allocatable Blocks Per SM given Registers used (Blocks)	10	1
GPU Occupancy (Warps)	48	1
Allocatable Blocks Per SM given Shared Memory usage (Blocks)	1,024	1

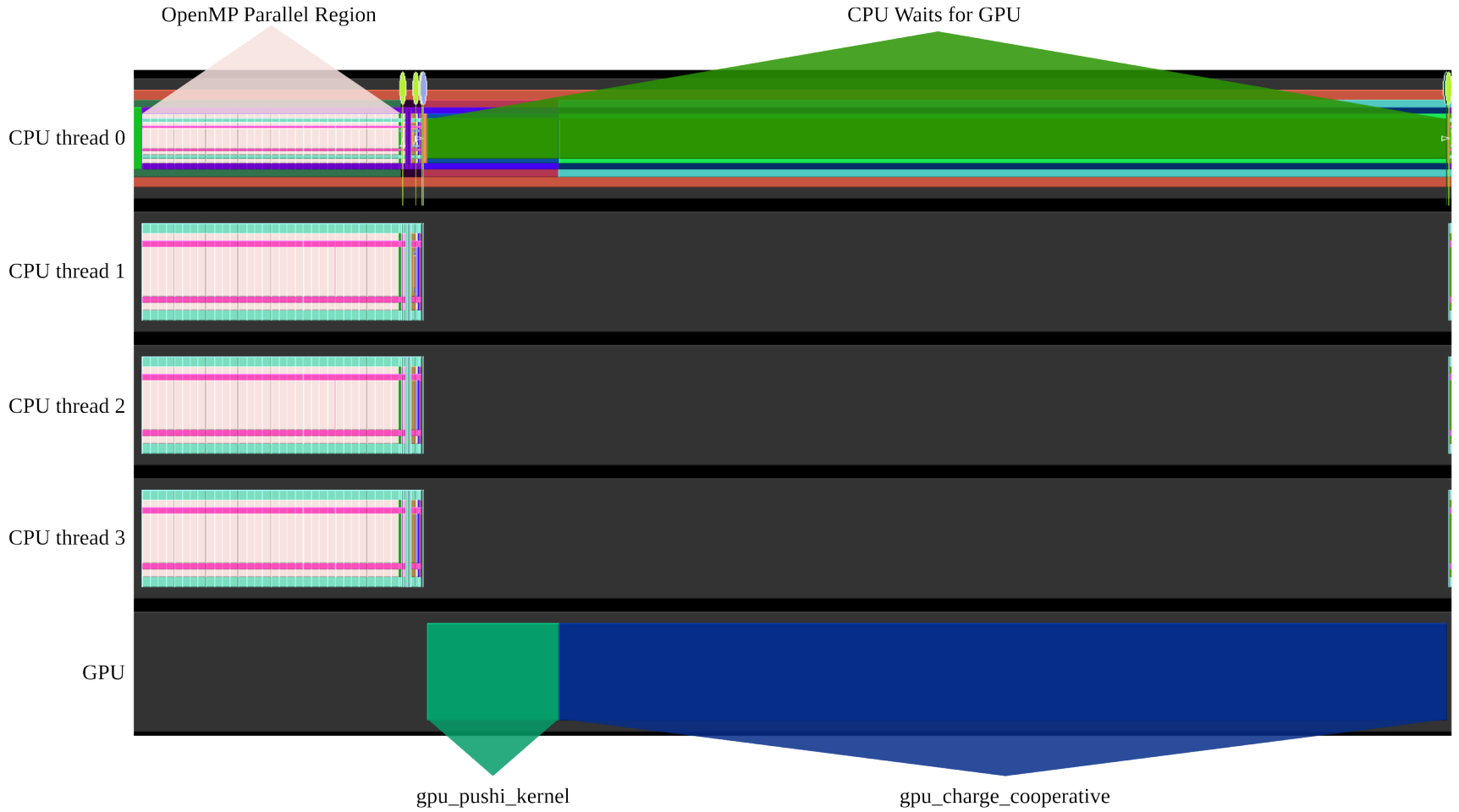
Heterogeneous Performance Case Study

- Gyrokinetic Toroidal Simulations (GTC)
  - Fusion simulation
- GTC CUDA version has been developed
  - OpenMP + CUDA
  - Three CUDA kernels



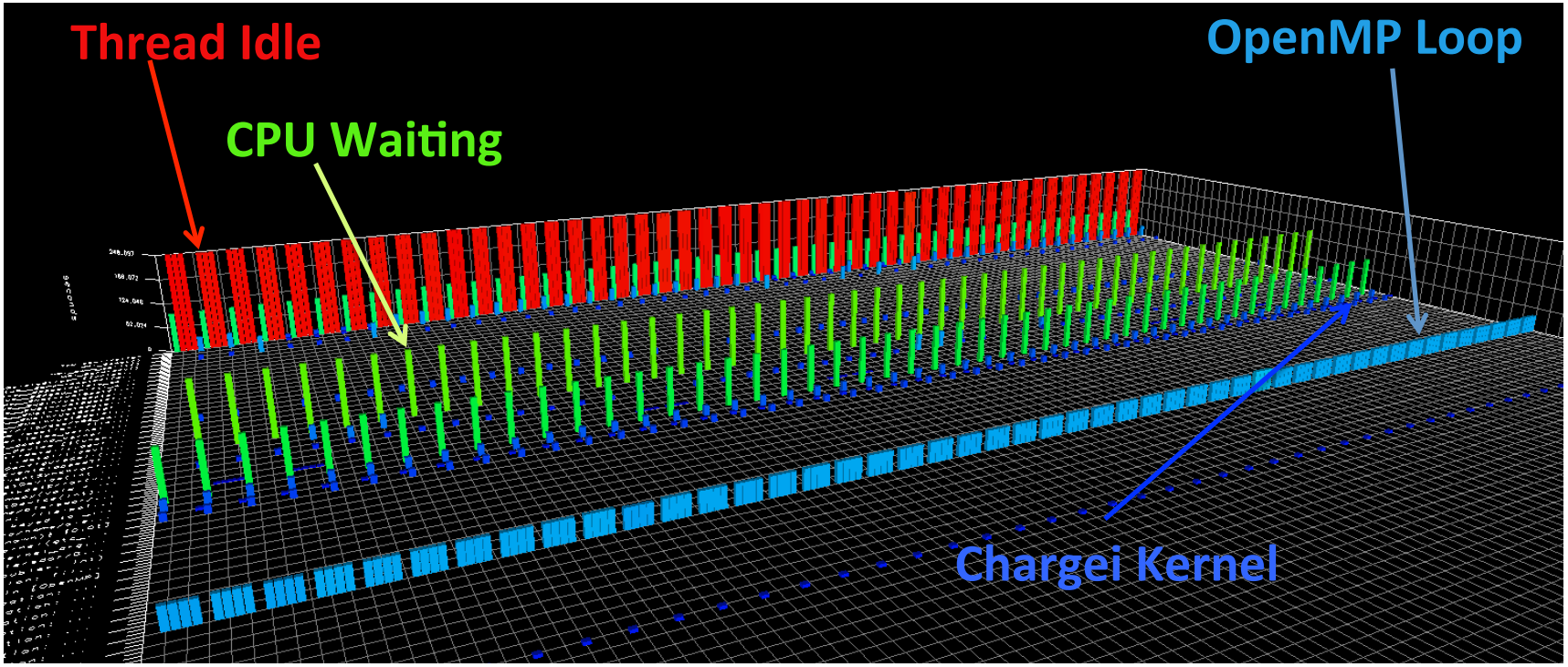
GTC Performance Trace

- GPU threads and OpenMP threads are integrated into trace



GTC on NSF Keeneland

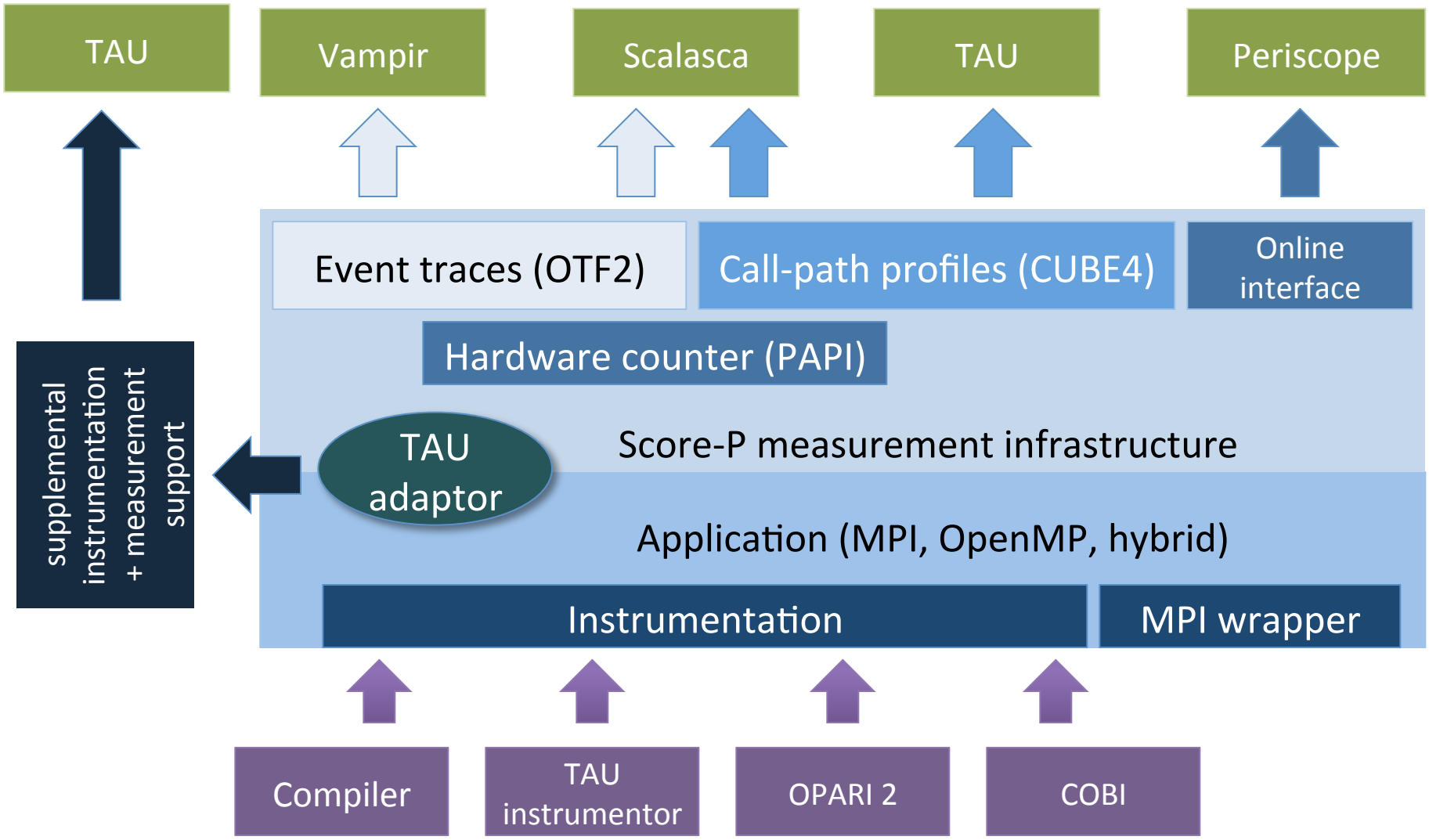
- HP SL390 GPU cluster
  - 2 Intel Xeon CPUs, 3 NVIDIA GPUs (M2070/M2090)
- GTC run on 16 nodes
  - 48 MPI ranks, 198 OpenMP threads, 48 GPUs



Common Infrastructure Integration – Score-P

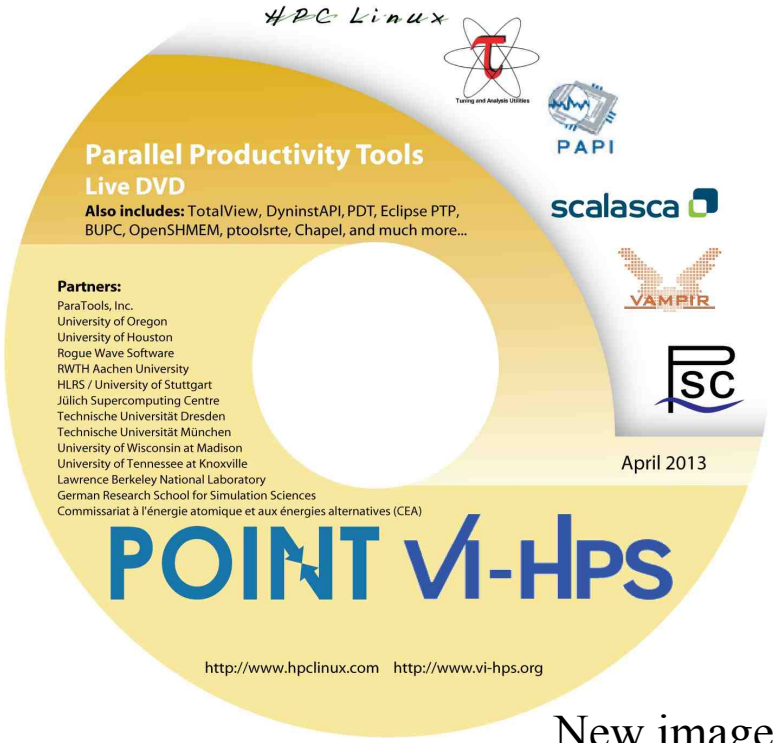
- Community effort for a common tools infrastructure
  - Starting with TAU, Periscope, Scalasca, and Vampir
  - Open for other tools and groups
- Joint development of Score-P
  - Core performance measurement infrastructure
  - MPI, OpenMP, heterogeneous
- DOE-funded PRIMA project
  - University of Oregon
  - Forschungszentrum Jülich
- BMBF-funded SILC project (multiple partners)

Score-P Architecture



For More Information …

- TAU Website
  http://tau.uoregon.edu
  - Software
  - Release notes
  - Documentation
- HPC Linux
  http://www.hpclinux.com
  - Parallel Tools "LiveDVD"
  - Boot up on your laptop or desktop
  - Includes TAU and variety of other packages
  - Include documentation and tutorial slides



New image just released at SC'13!

Short Course Outline

- Lecture 1: Introduction and Fundamentals
- Lecture 2: Methodology
- Lecture 3: Tools Technology
- Lecture 4: Tools Landscape – Part 1
- Lecture 5: Tools Landscape – Part 2
- Lecture 6: TAU Performance System
- Lecture 7: TAU Applications
- Lecture 8: Advances in TAU
- Lecture 9: Future Directions

Tuesday

Wednesday

Thursday