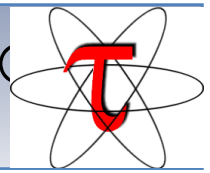
TAU PERFORMANCE SYSTEM

Sameer Shende
Alan Morris, Wyatt Spear, Scott Biersdorff
Performance Research Lab

Allen D. Malony, Kevin Huck, Aroon Nataraj
Department of Computer and Information Science
University of Oregon

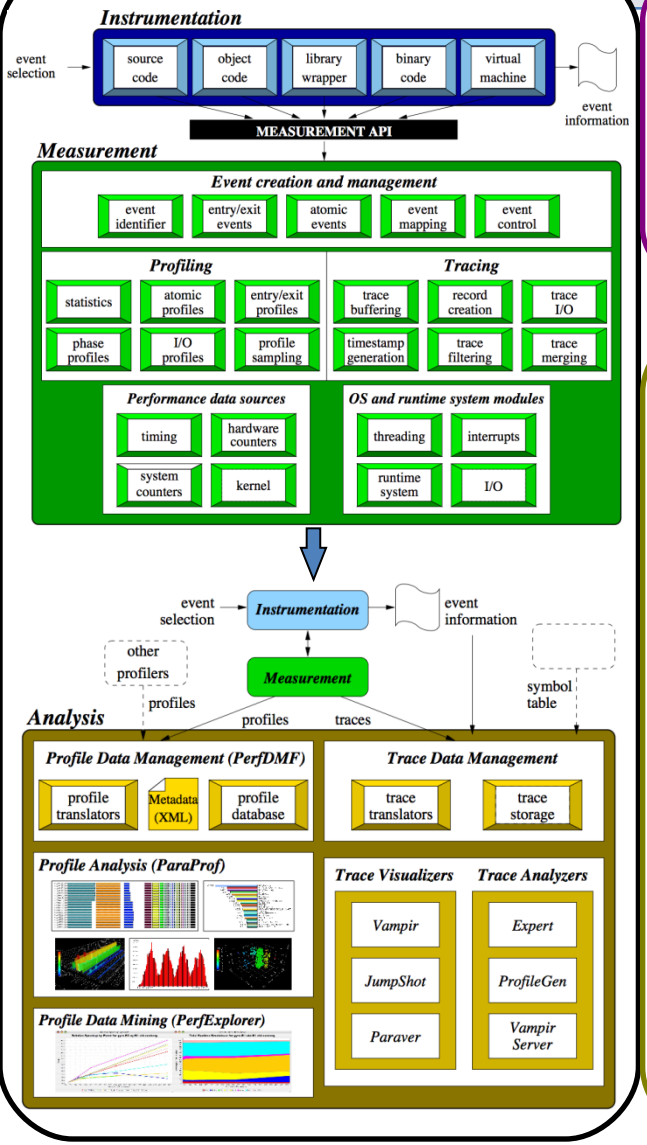
TAU Performance System®



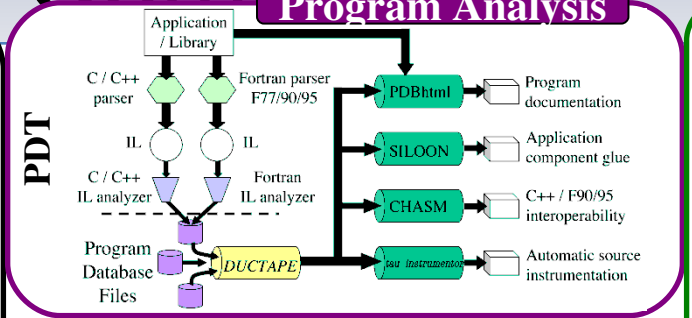
- *T*uning and *A*nalysis *U*tilities (16+ year project)
- Performance problem solving framework for HPC
  - Integrated, scalable, flexible, portable
  - Target all parallel programming / execution paradigms
- Integrated performance toolkit (open source)
  - Instrumentation, measurement, analysis, visualization
  - Widely-ported performance profiling / tracing system
  - Performance data management and data mining

TAU Performance System Components

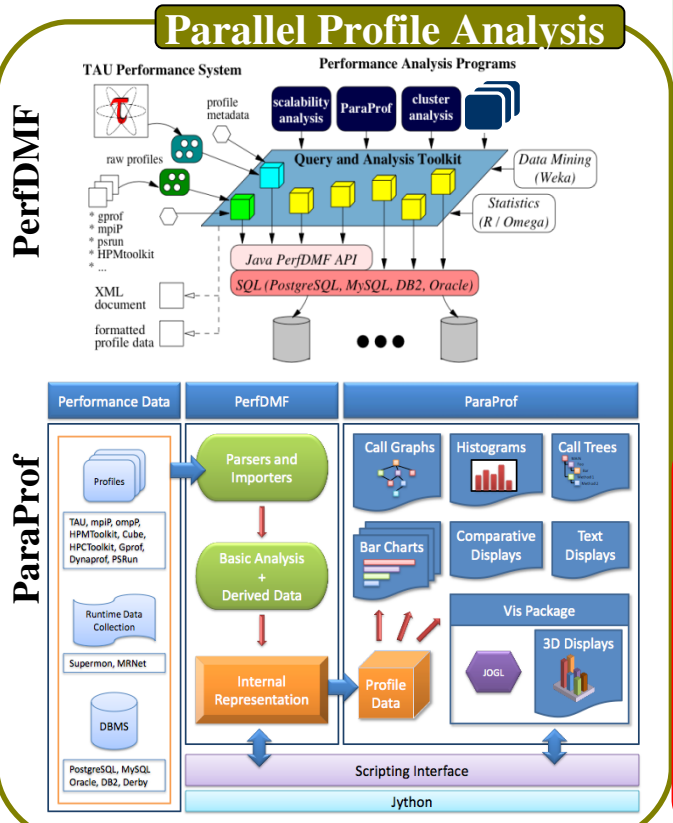
TAU Architecture



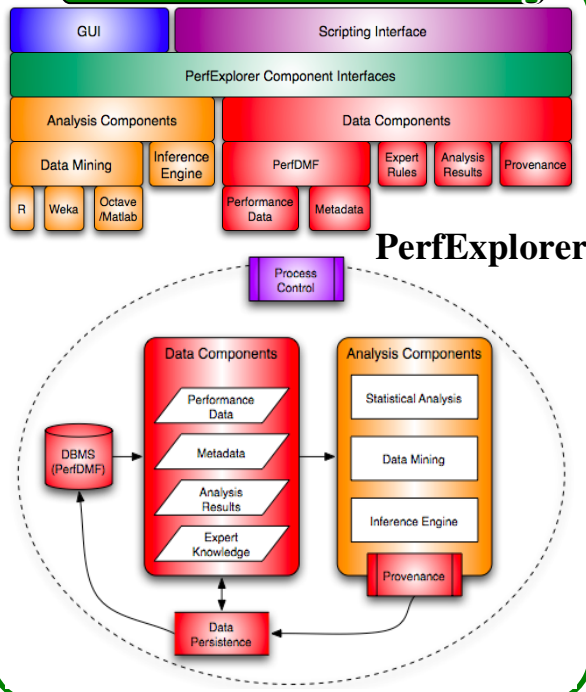
Program Analysis



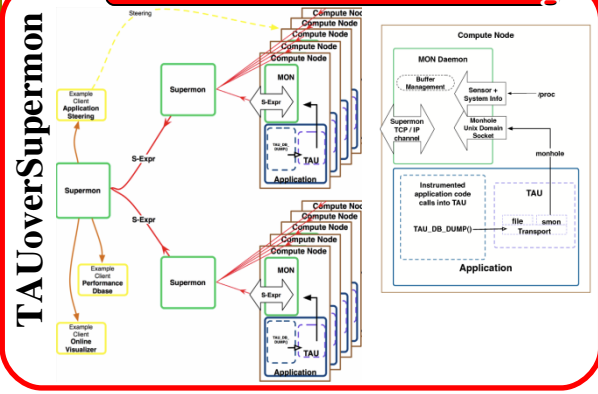
Parallel Profile Analysis



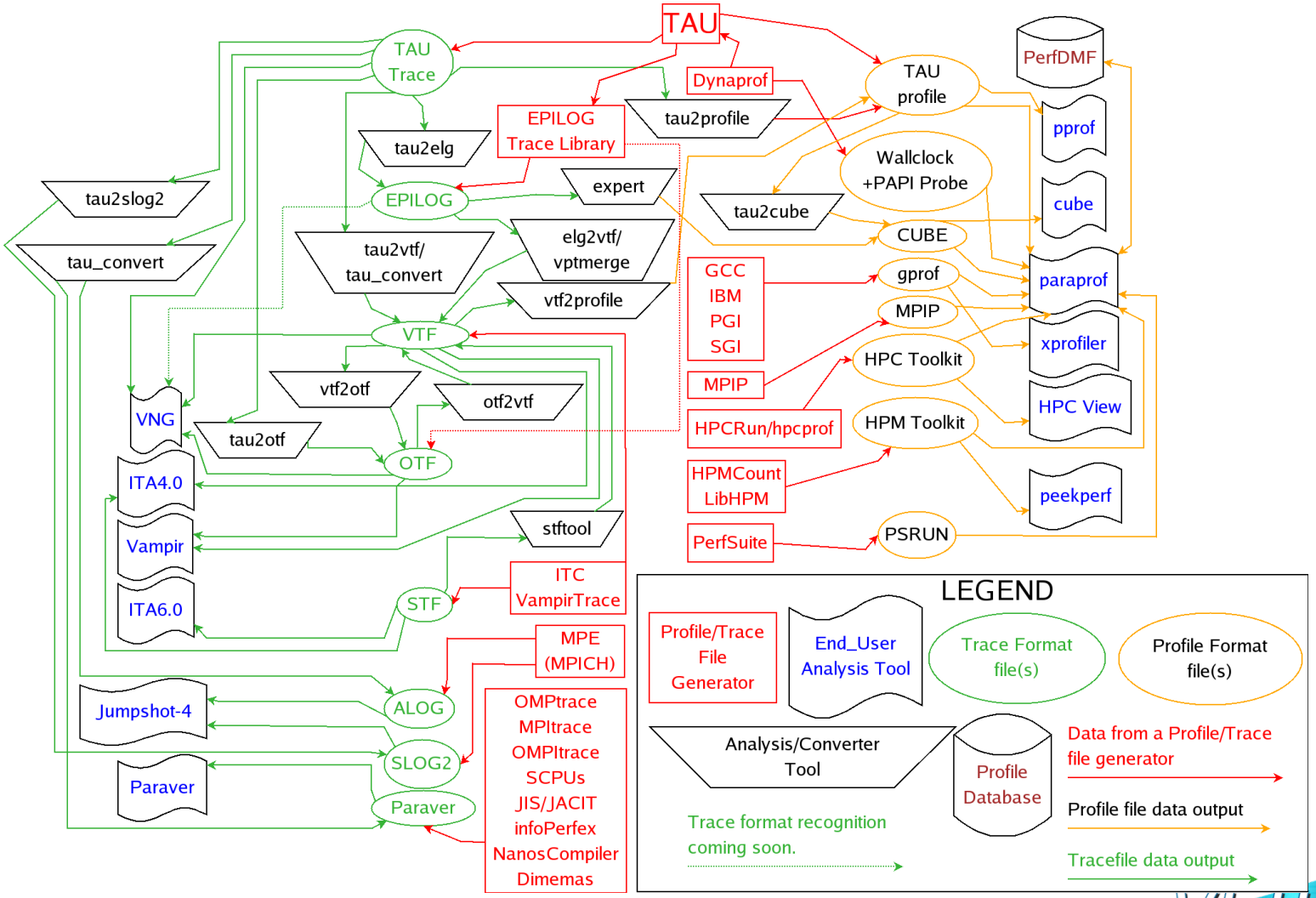
Performance Data Mining



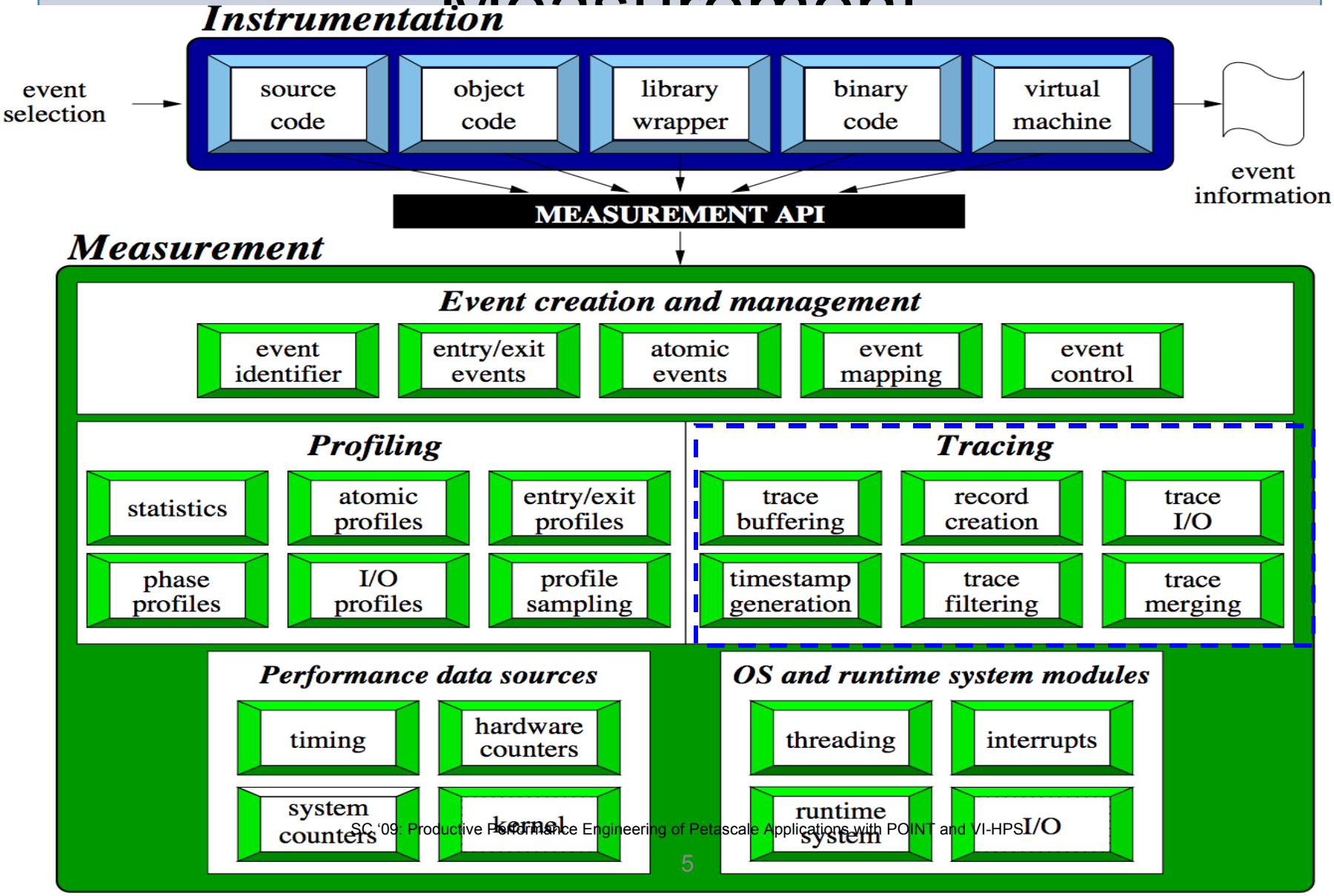
Performance Monitoring



Building Bridges to Other Tools



TAU Instrumentation / Measurement



Direct Performance Observation

- Execution actions of interest exposed as events
  - In general, actions reflect some execution state
    - presence at a code location or change in data
    - occurrence in parallelism context (thread of execution)
  - Events encode actions for performance system to observe
- Observation is direct
  - Direct instrumentation of program (system) code (probes)
  - Instrumentation invokes performance measurement
  - Event measurement: performance data, meta-data, context
- Performance experiment
  - Actual events + performance measurements
- Contrast with (indirect) event-based sampling

TAU Instrumentation Approach

- Support for standard program events
  - Routines, classes and templates
  - Statement-level blocks
  - Begin/End events (Interval events)
- Support for user-defined events
  - Begin/End events specified by user
  - Atomic events (e.g., size of memory allocated/freed)
  - Flexible selection of event statistics
- Provides static events and dynamic events
- Enables “semantic” mapping
- Specification of event groups (aggregation, selection)
- Instrumentation optimization

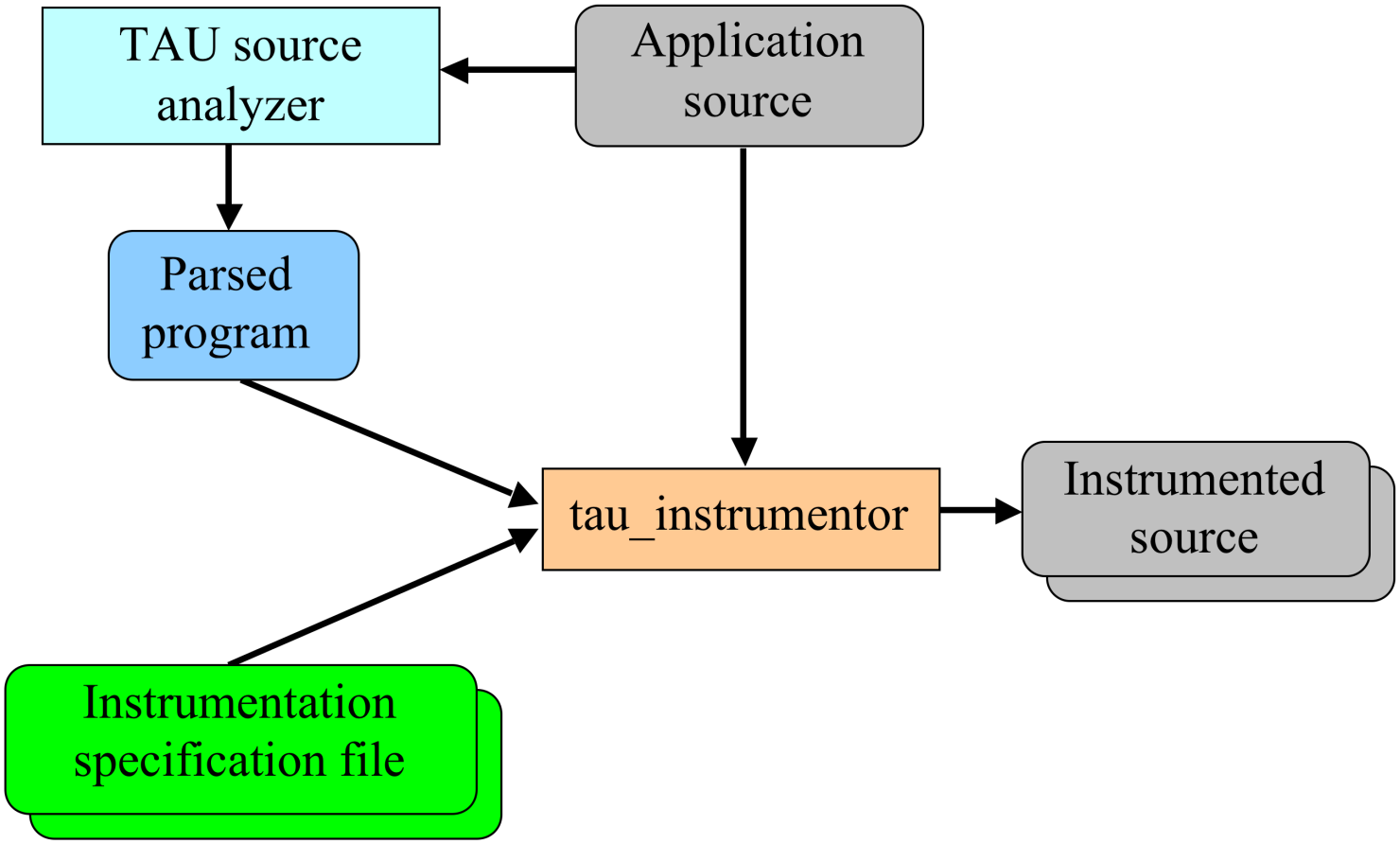
TAU Event Interface

- Events have a type, a group association, and a name
- TAU events names are character strings
  - Powerful way to encode event information
  - Inefficient way to communicate each event occurrence
- TAU maps a new event name to an event ID
  - Done when event is first encountered (get event handle)
  - Event ID is used for subsequent event occurrences
  - Assigning a uniform event ID a priori is problematic
- A new event is identified by a new event name in TAU
  - Can create new event names at runtime
  - Allows for dynamic events (TAU renames events)
  - Allows for context-base, parameter-based, phase events

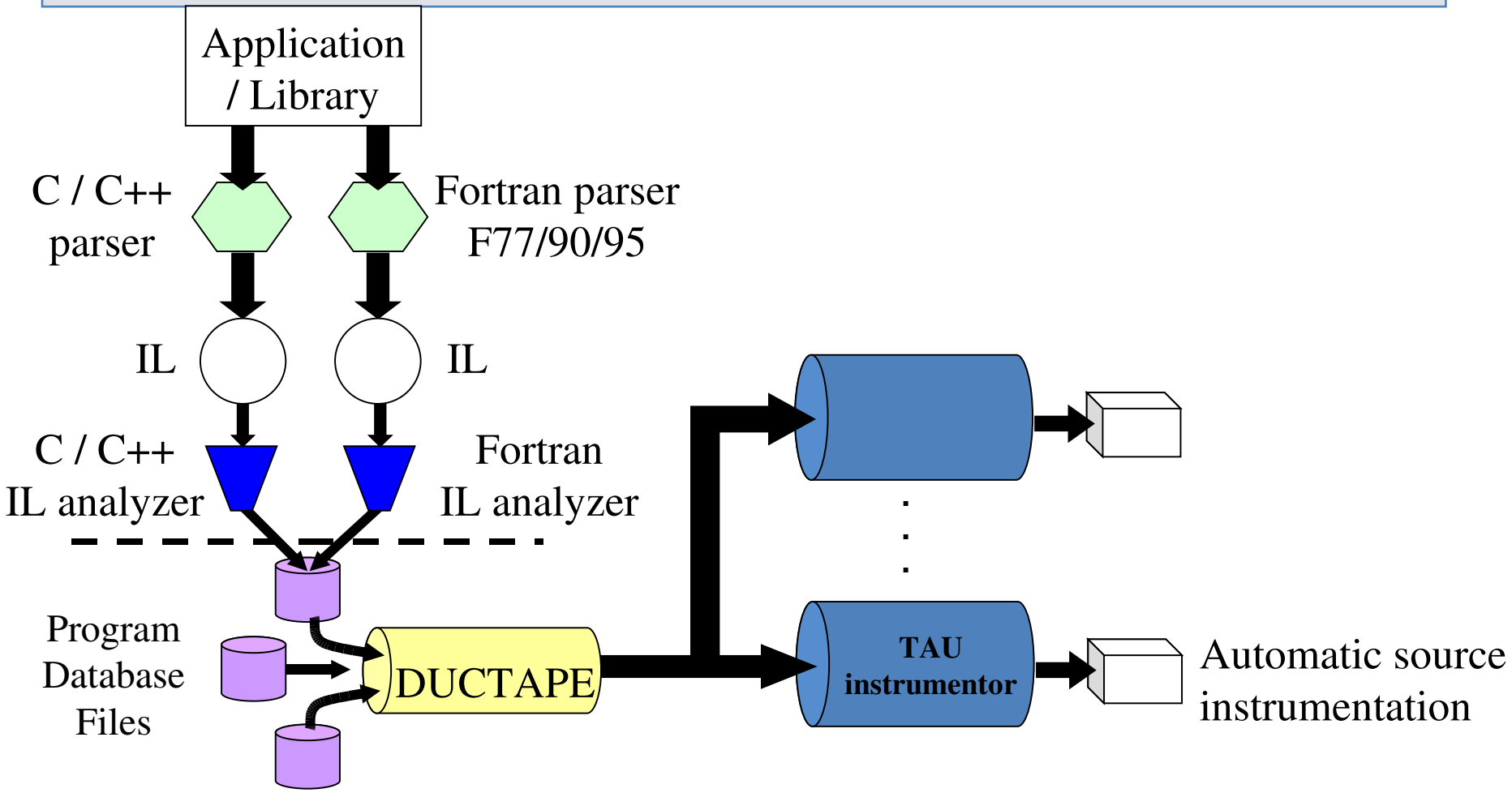
TAU Instrumentation Mechanisms

- Source code
  - Manual (TAU API, TAU component API)
  - Automatic (robust)
    - C, C++, F77/90/95 (Program Database Toolkit (PDT))
    - OpenMP (directive rewriting (Opari), POMP2 spec)
    - Library header wrapping
- Object code
  - Pre-instrumented libraries (e.g., MPI using PMPI)
  - Statically- and dynamically-linked (with LD_PRELOAD)
- Executable code
  - Binary and dynamic instrumentation (Dyninst)
  - Virtual machine instrumentation (e.g., Java using JVMPI)
- TAU_COMPILER to automate instrumentation process

Automatic Source-level Instrumentation



Program Database Toolkit (PDT)



MPI Wrapper Interposition Library

- Uses standard MPI Profiling Interface
  - Provides name shifted interface
    - MPI_Send = PMPI_Send
    - Weak bindings
- Create TAU instrumented MPI library
  - Interpose between MPI and TAU
  - Done during program link
    - -lmpi replaced by –lTauMpi –lpmpi –lmpi
  - No change to the source code!
  - Just re-link application to generate performance data

MPI Shared Library Instrumentation

- Interpose the MPI wrapper library for applications that have already been compiled
  - Avoid re-compilation or re-linking
- Requires shared library MPI
  - Uses LD_PRELOAD for Linux
  - On AIX use MPI_EUILIB / MPI_EUILIBPATH
  - Does not work on XT3
- Approach will work with other shared libraries
- Use TAU tauex
  - % mpirun -np 4 tauex a.out

Selective Instrumentation File

- Specify a list of events to exclude or include
- # is a wildcard in a routine name

```
BEGIN_EXCLUDE_LIST
Foo
Bar
D#EMM
END_EXCLUDE_LIST
BEGIN_INCLUDE_LIST
int main(int, char **)
F1
F3
END_INCLUDE_LIST
```

Selective Instrumentation File

- Optionally specify a list of files
- * and ? may be used as wildcard characters

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

Selective Instrumentation File

- User instrumentation commands
  - Placed in INSTRUMENT section
  - Routine entry/exit
  - Arbitrary code insertion
  - Outer-loop level instrumentation

```
BEGIN_INSTRUMENT_SECTION
loops file="foo.f90" routine="matrix#"
memory file="foo.f90" routine="#"
io routine="matrix#"
[static/dynamic] phase routine="MULTIPLY"
dynamic [phase/timer] name="foo" file="foo.cpp" line=22 to line=35
file="foo.f90" line = 123 code = "  print *, \" Inside foo\""
exit routine = "int foo()" code = "cout <<\"exiting foo\"<<endl;"
END_INSTRUMENT_SECTION
```

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)
- Performance measurement of I/O and Linux kernel

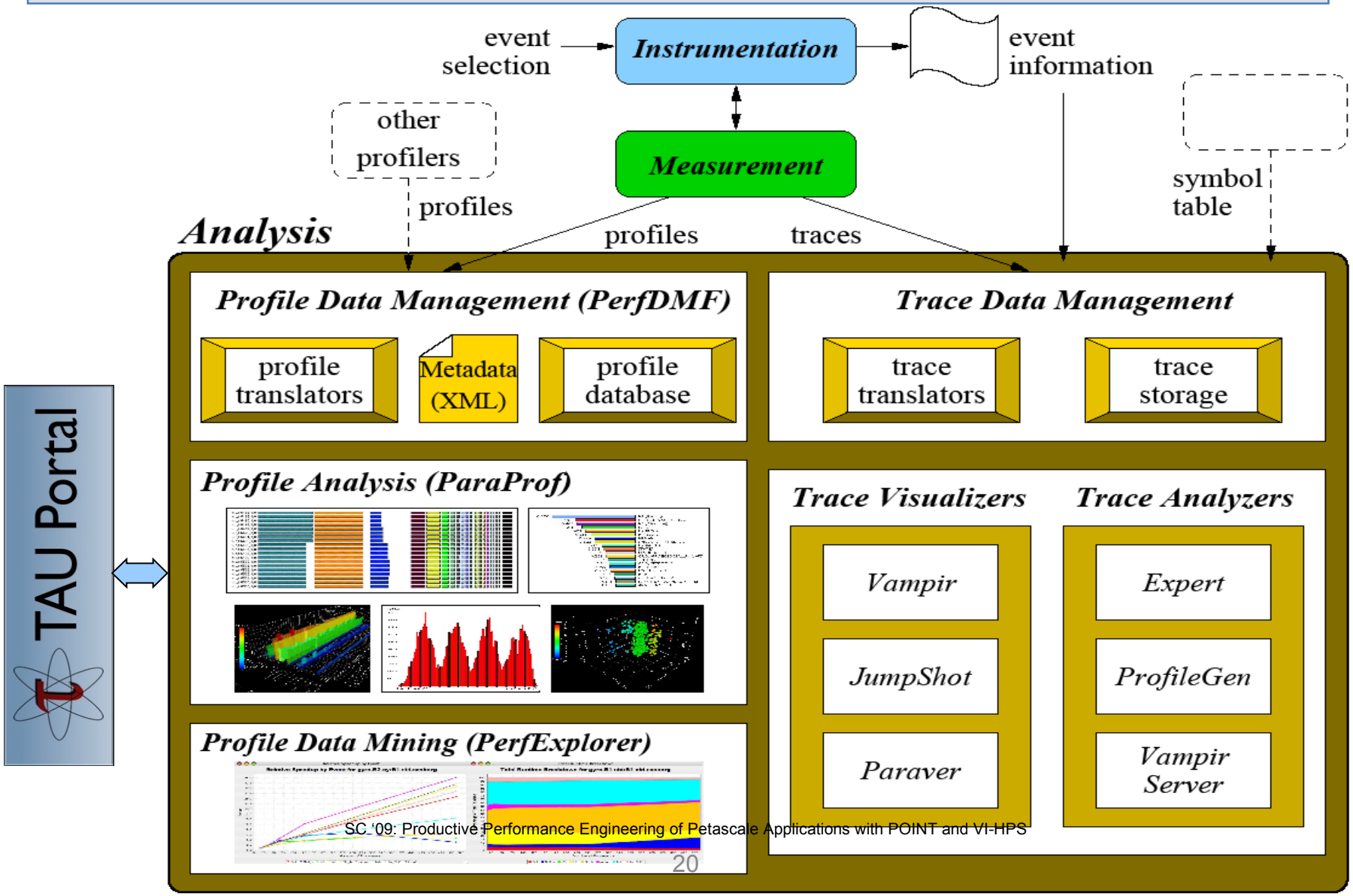
TAU Measurement Mechanisms

- 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

Types of 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

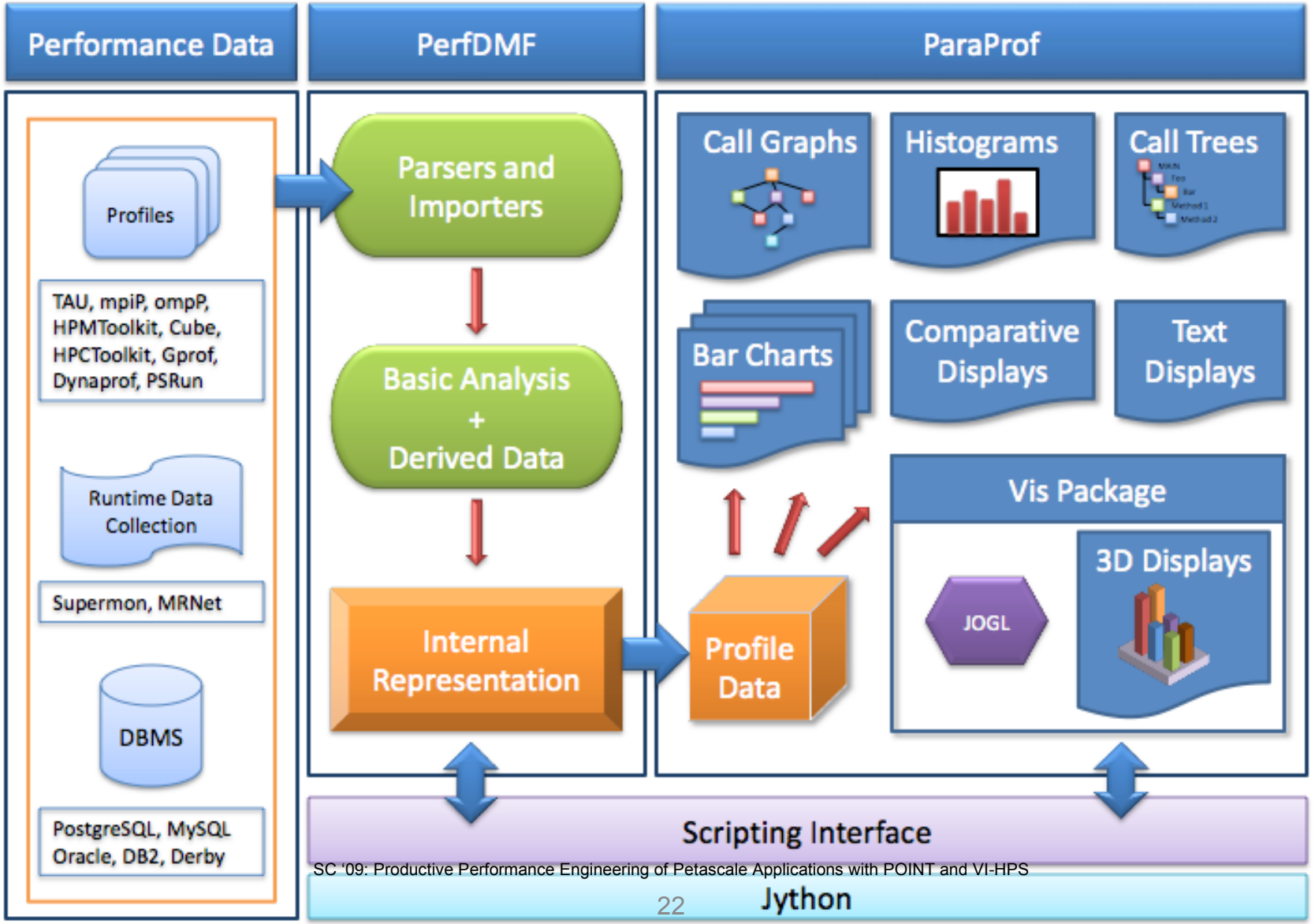
TAU Analysis



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 framework (PerfDMF)
- Parallel trace analysis
  - Translation to VTF (V3.0), EPILOG, OTF formats
  - Integration with Vampir / Vampir Server (TU Dresden)
  - Profile generation from trace data
- Online parallel analysis and visualization
- Integration with CUBE browser (Scalasca, UTK / FZJ)

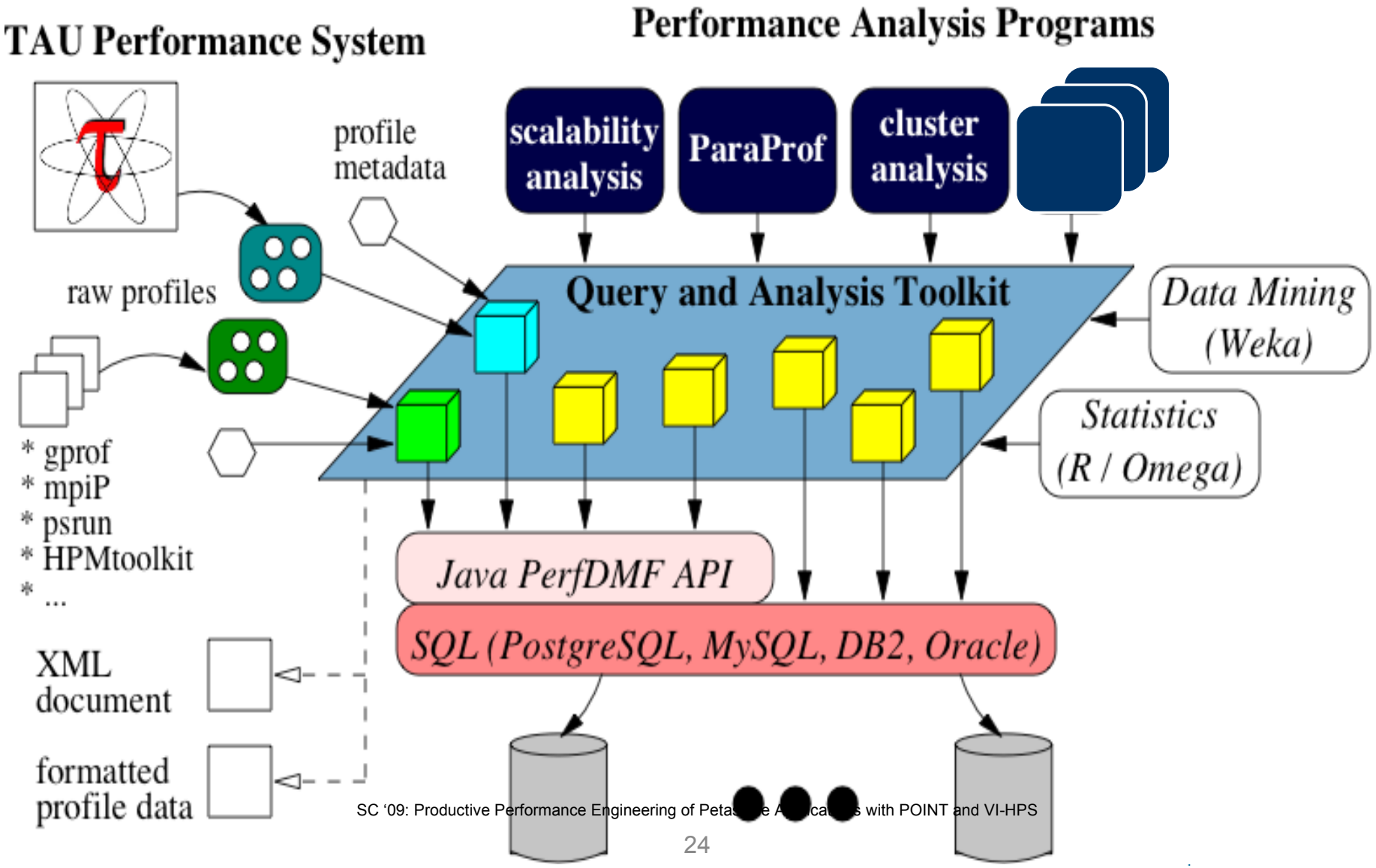
ParaProf Profile Analysis Framework



Performance Data Management

- 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 (PerfDMF)
  - Originally designed to address critical TAU requirements
  - Supported profile formats:
    TAU, CUBE (Scalasca), HPC Toolkit (Rice), HPM Toolkit (IBM), gprof, mpiP, psrun (PerfSuite), Open|SpeedShop, …
  - Supported DBMS:
    PostgreSQL, MySQL, Oracle, DB2, Derby/Cloudscape
  - Profile query and analysis API
- Reference implementation for PERI-DB project

PerfDMF Architecture



Metadata Collection

- Integration of XML metadata for each parallel profile
- Three ways to incorporate metadata
  - Measured hardware/system information (TAU, PERI-DB)
    - 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
  - PerfDMF data management tools can incorporate an XML file of additional metadata
    - Compiler flags, submission scripts, input files, …
- Metadata can be imported from / exported to PERI-DB

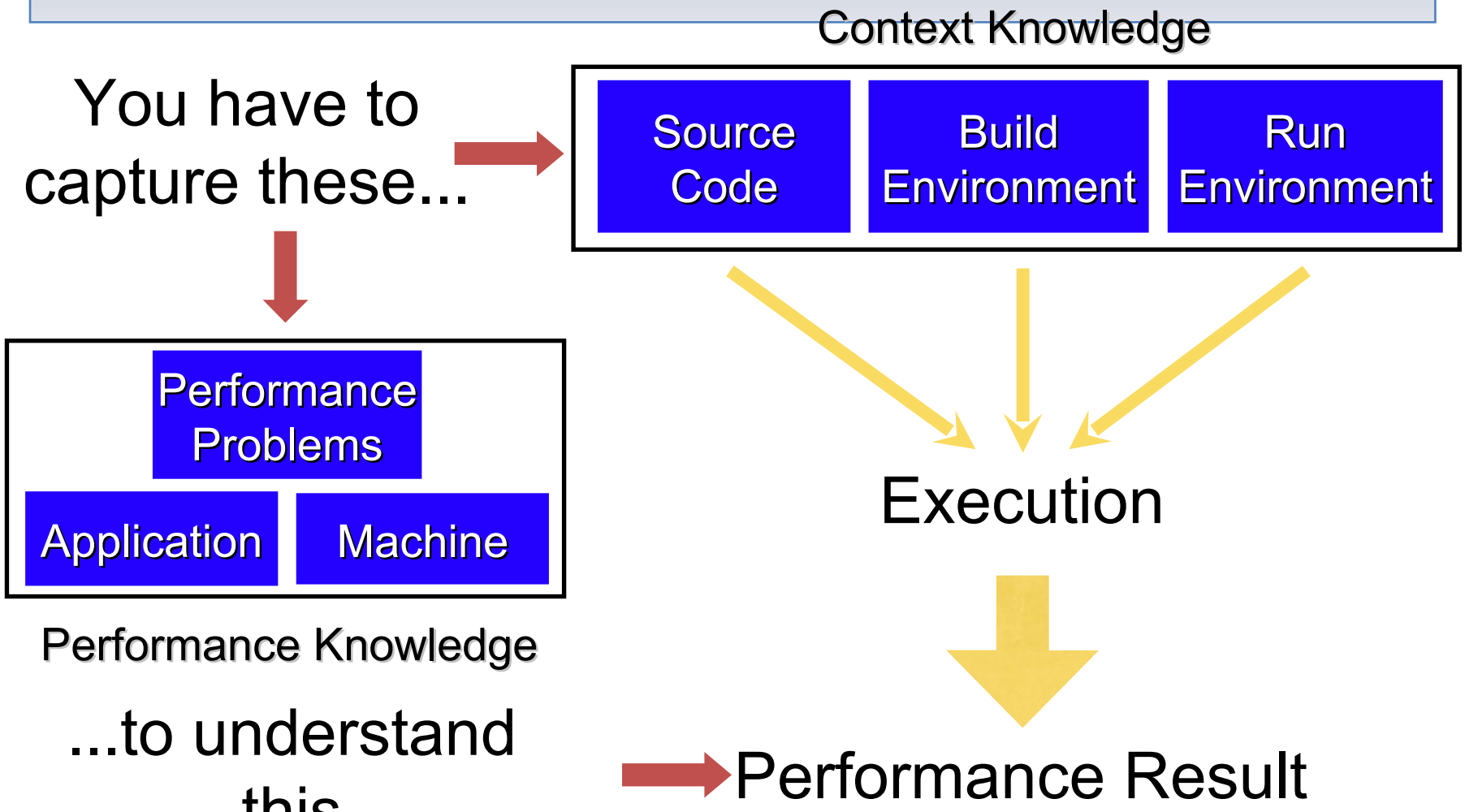
Performance Data Mining / Analytics

- 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
    - comparative, clustering, correlation, dimension reduction, …
  - Use the existing TAU infrastructure
- PerfExplorer v1 performance data mining framework
  - Multiple experiments and parametric studies
  - Integrate available statistics and data mining packages
    - Weka, R, Matlab / Octave
  - Apply data mining operations in interactive enviroment

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
    - iterative tuning, performance regression testing
  - Performance model generation requires interpretation
- We need better methods and tools for
  - Integrating meta-information
  - Knowledge-based performance problem solving

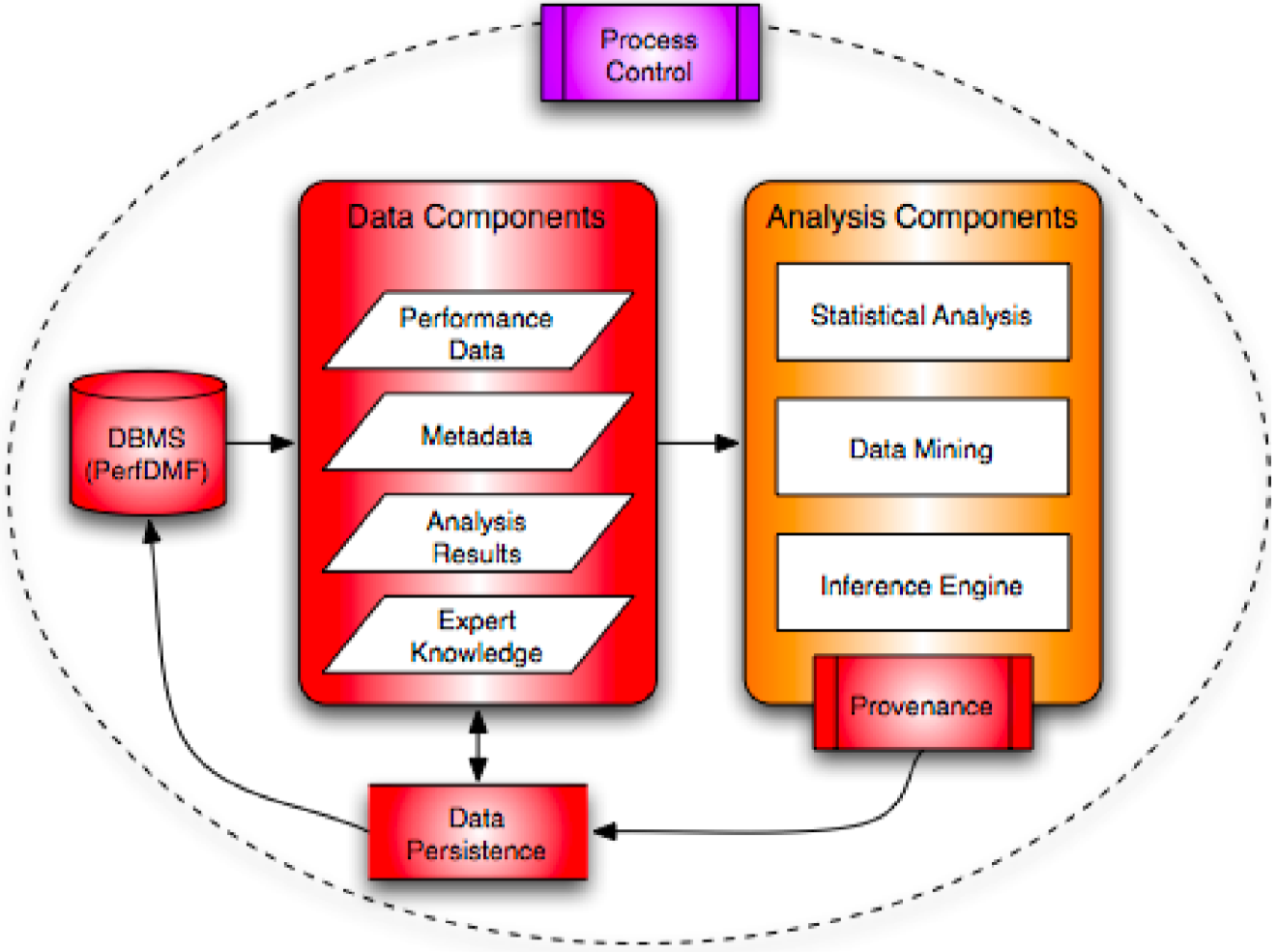
Role of Metadata and Knowledge Role



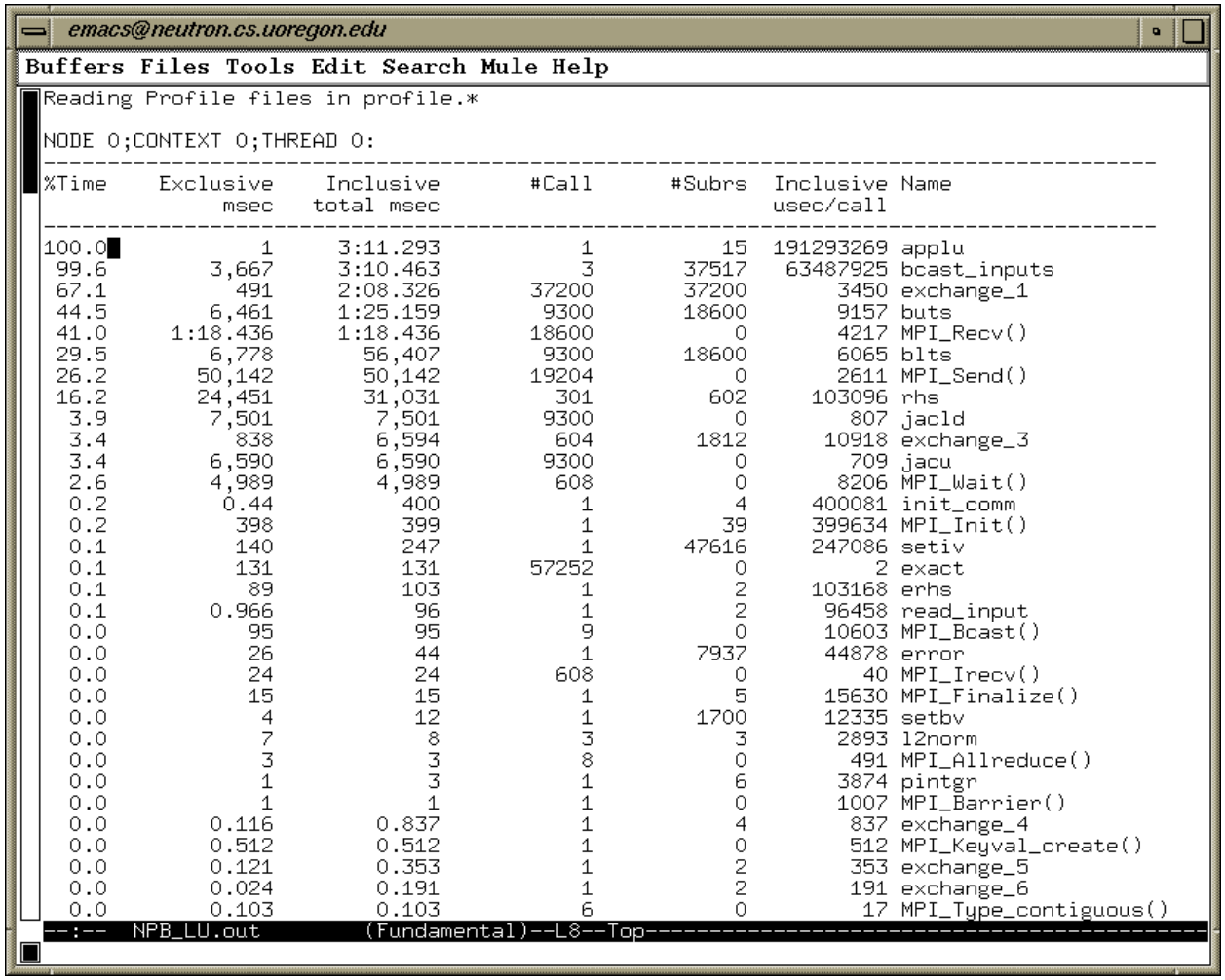
PerfExplorer v2 – Requirements

- Component-based analysis process
  - Analysis operations implemented as modules
  - Linked together in analysis process and workflow
- Scripting
  - Provides process/workflow development and automation
- Metadata input, management, and access
- Inference engine
  - Reasoning about causes of performance phenomena
  - Analysis knowledge captured in expert rules
- Persistence of intermediate analysis results
- Provenance
  - Provides historical record of analysis results

PerfExplorer v2 Architecture



Parallel Profile Analysis – pprof



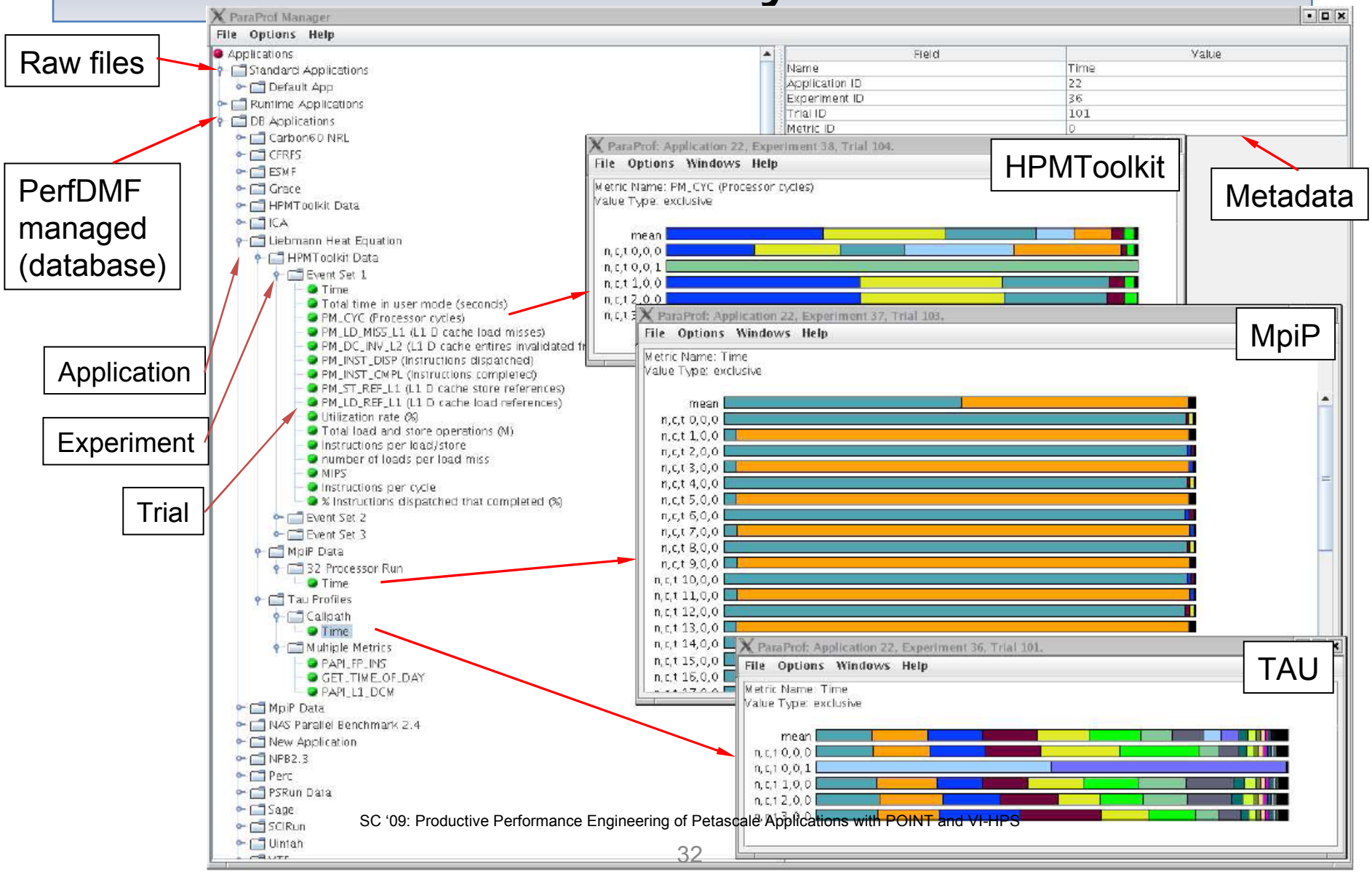
emacs@neutron.cs.uoregon.edu

Buffers Files Tools Edit Search Mule Help

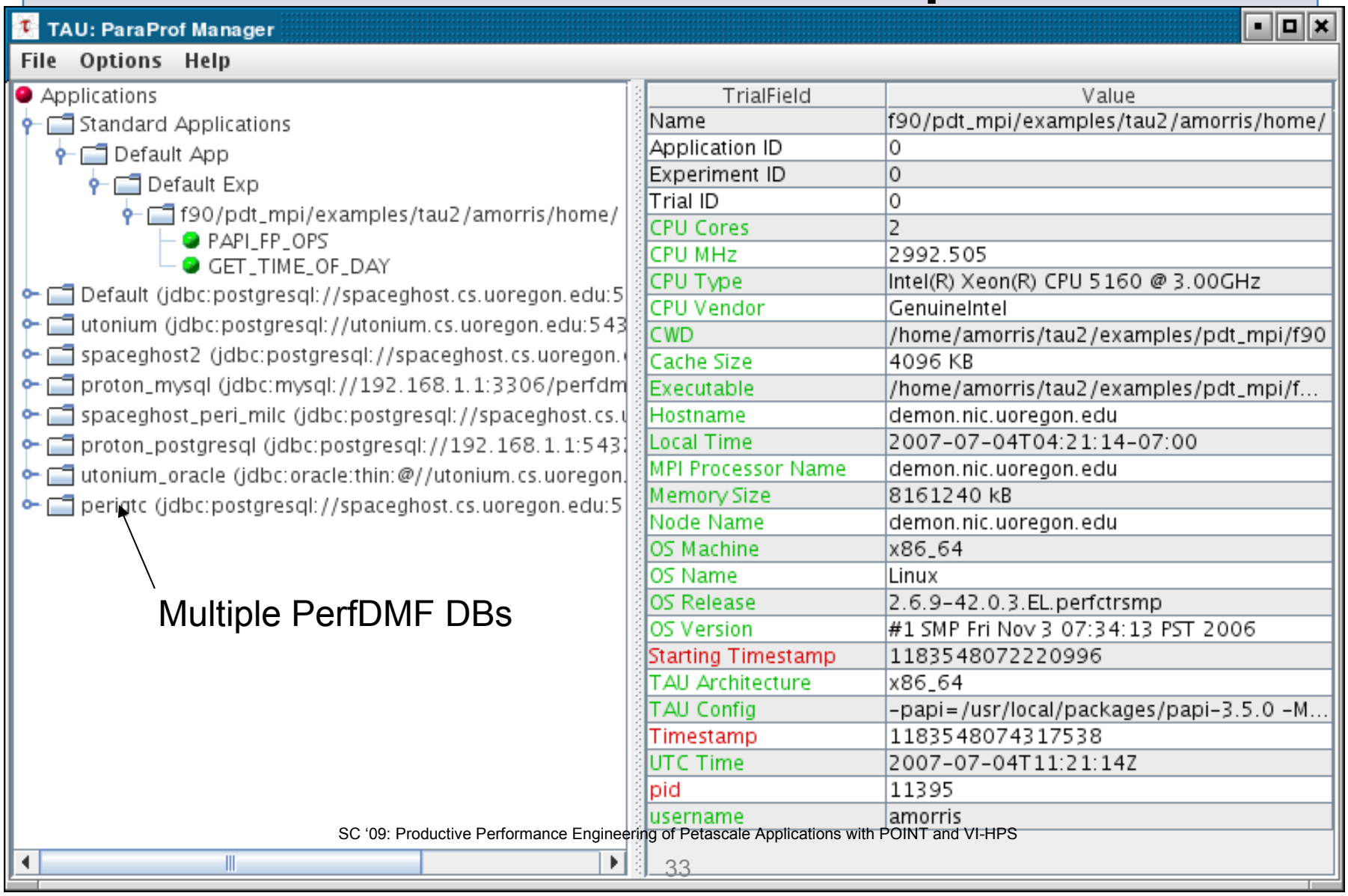
```
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0           1     3:11.293           1          15  191293269 applu
 99.6       3,667     3:10.463           3       37517   63487925 bcast_inputs
 67.1         491     2:08.326       37200       37200       3450 exchange_1
 44.5       6,461     1:25.159        9300       18600       9157 buts
 41.0    1:18.436     1:18.436       18600           0       4217 MPI_Recv()
 29.5       6,778       56,407        9300       18600       6065 blts
 26.2      50,142       50,142       19204           0       2611 MPI_Send()
 16.2      24,451       31,031         301         602     103096 rhs
  3.9       7,501        7,501        9300           0        807 jacld
  3.4         838        6,594         604        1812      10918 exchange_3
  3.4       6,590        6,590        9300           0        709 jacu
  2.6       4,989        4,989         608           0       8206 MPI_Wait()
  0.2        0.44          400           1           4     400081 init_comm
  0.2         398          399           1          39     399634 MPI_Init()
  0.1         140          247           1       47616     247086 setiv
  0.1         131          131       57252           0          2 exact
  0.1          89          103           1           2     103168 erhs
  0.1       0.966           96           1           2      96458 read_input
  0.0          95           95           9           0      10603 MPI_Bcast()
  0.0          26           44           1        7937      44878 error
  0.0          24           24         608           0         40 MPI_Irecv()
  0.0          15           15           1           5      15630 MPI_Finalize()
  0.0           4           12           1        1700      12335 setbv
  0.0           7            8           3           3       2893 l2norm
  0.0           3            3           8           0        491 MPI_Allreduce()
  0.0           1            3           1           6       3874 pintgr
  0.0           1            1           1           0       1007 MPI_Barrier()
  0.0       0.116        0.837           1           4        837 exchange_4
  0.0       0.512        0.512           1           0        512 MPI_Keyval_create()
  0.0       0.121        0.353           1           2        353 exchange_5
  0.0       0.024        0.191           1           2        191 exchange_6
  0.0       0.103        0.103           6           0         17 MPI_Type_contiguous()
--:--  NPB_LU.out       (Fundamental)--L8--Top-----------------------------------------
```

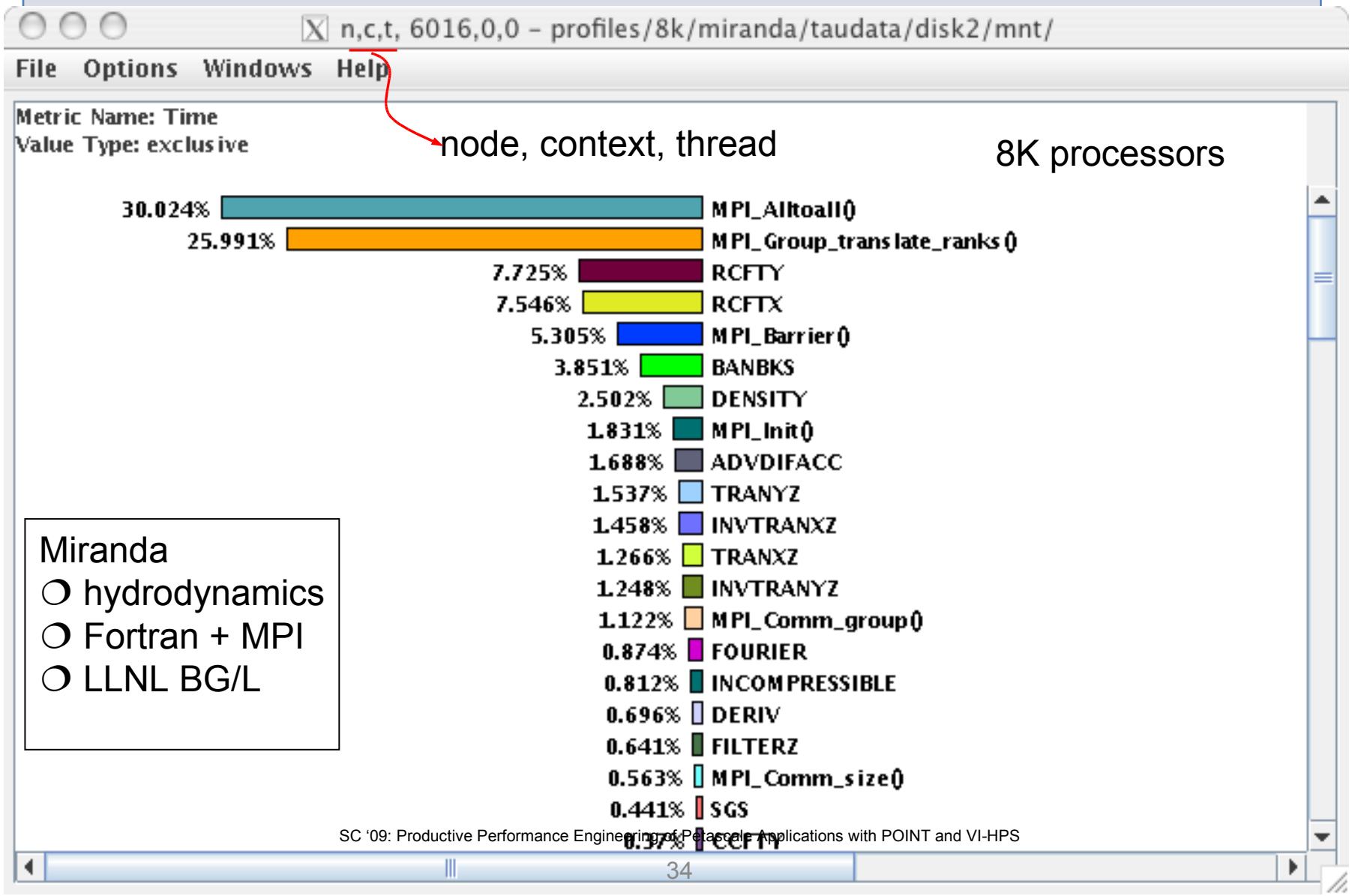
Parallel Profile Analysis – ParaProf



Metadata for Each Experiment

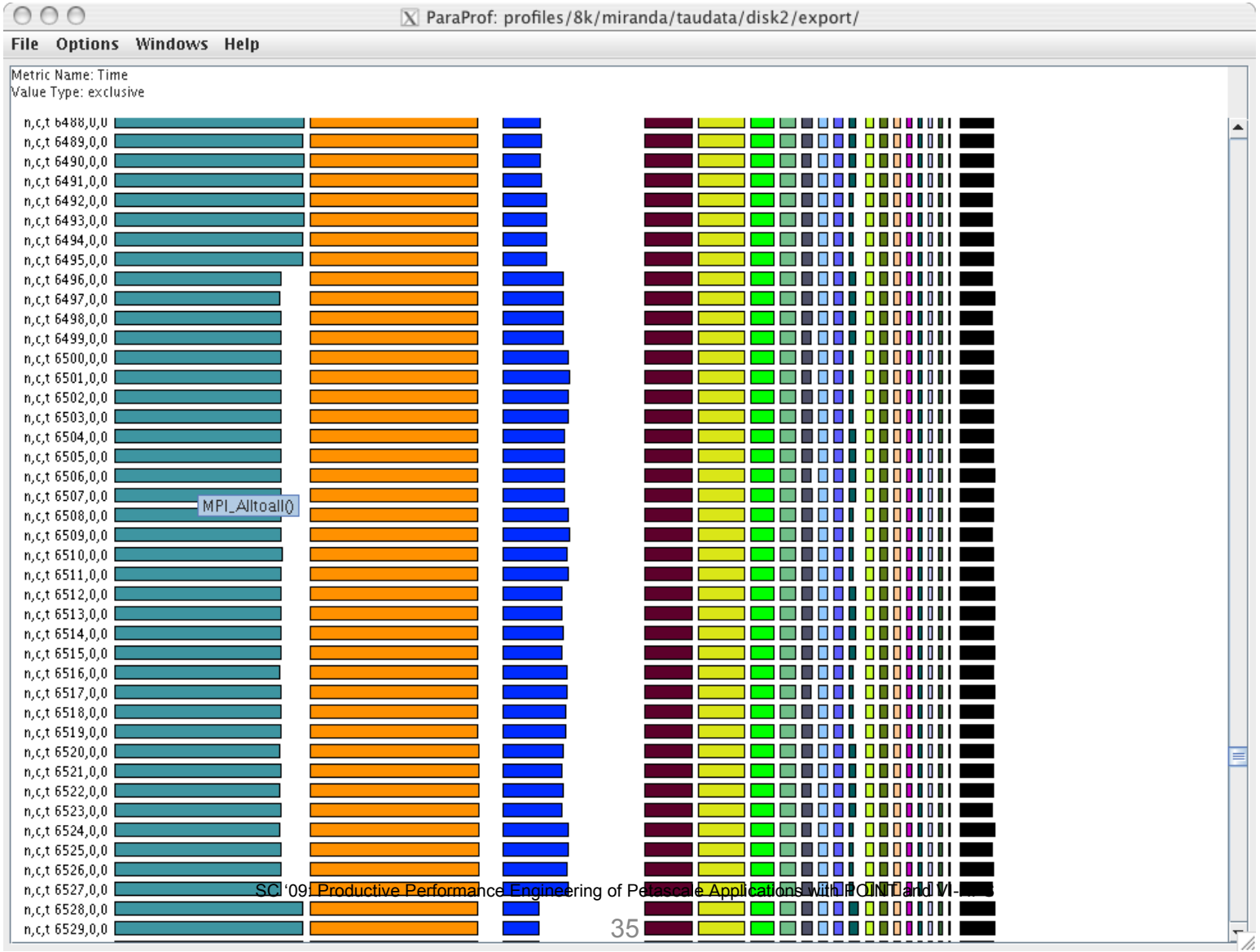


ParaProf – Flat Profile

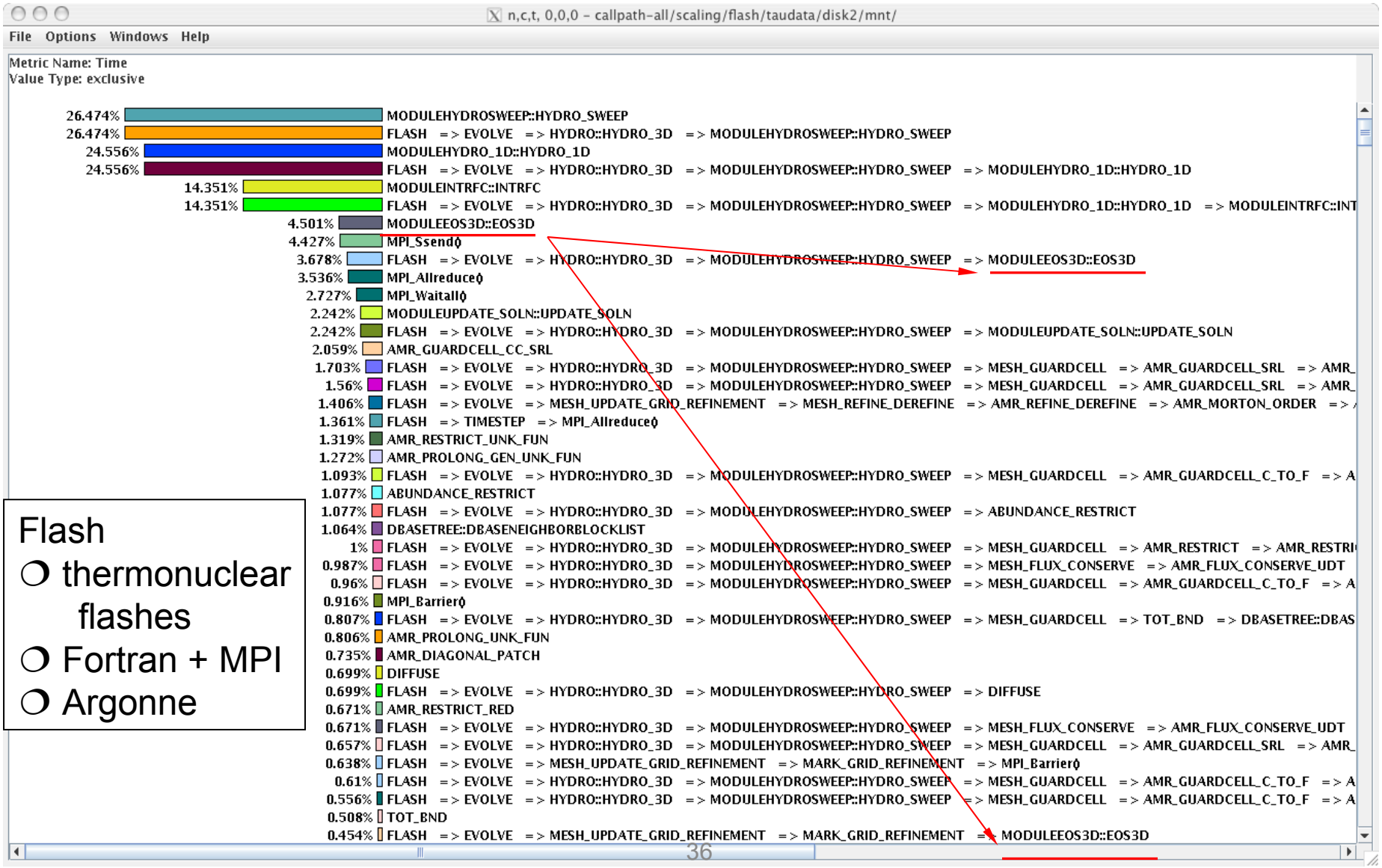


Miranda
- hydrodynamics
- Fortran + MPI
- LLNL BG/L

ParaProf – Stacked View

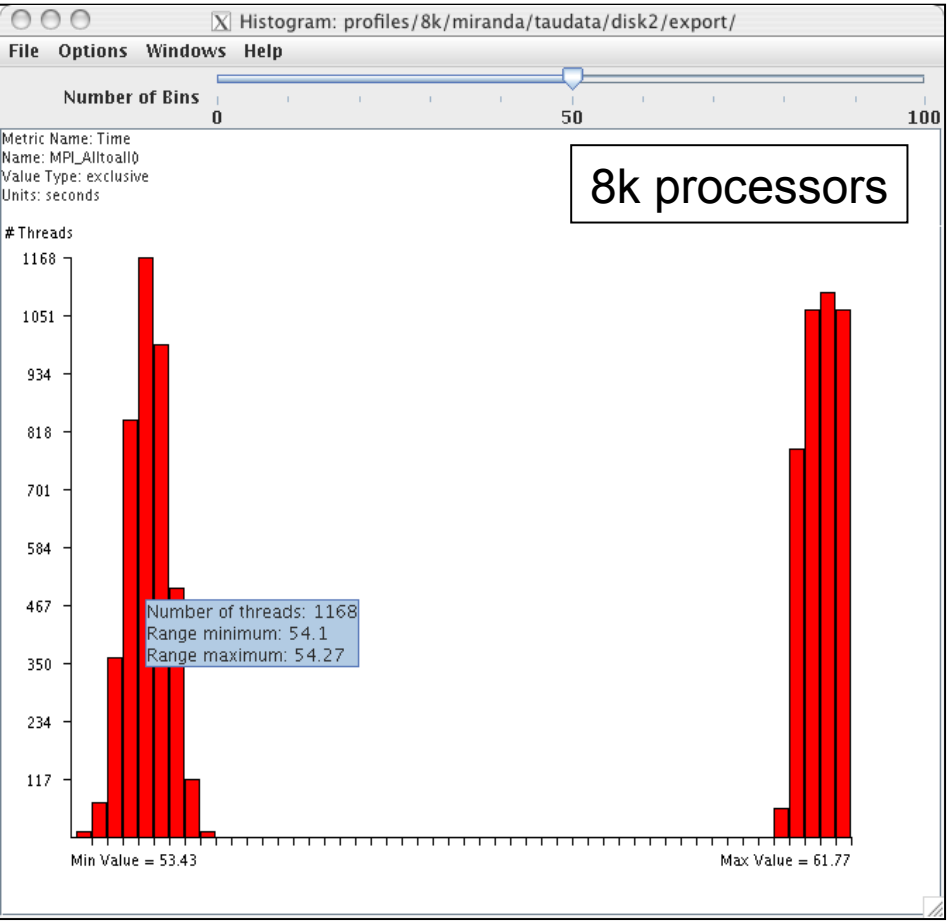


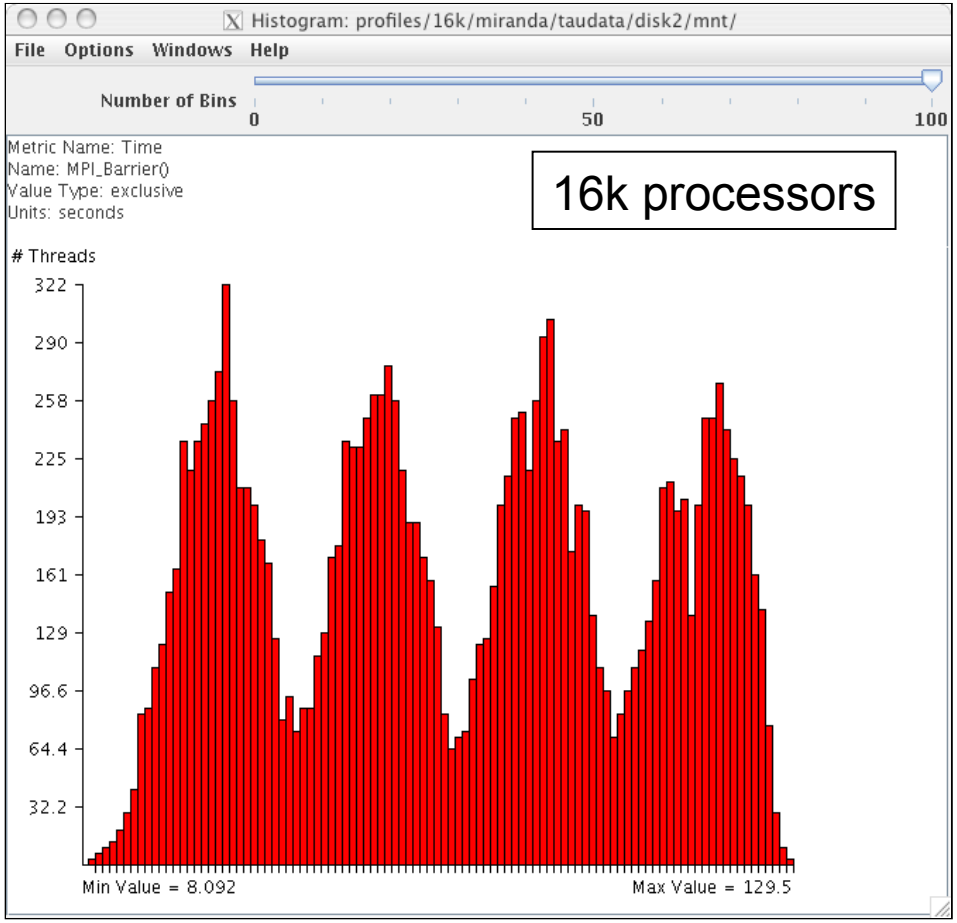
ParaProf – Callpath Profile



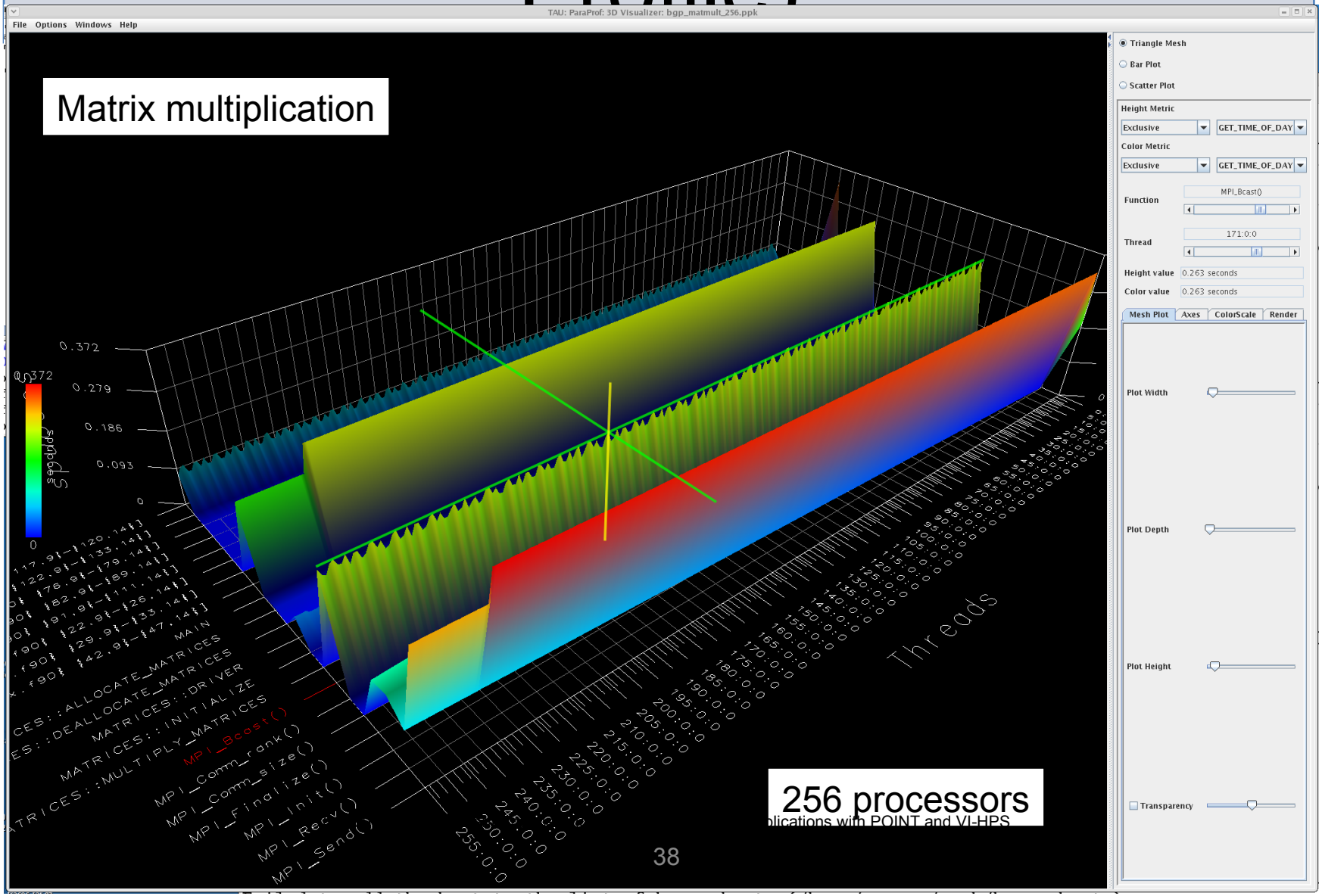
Flash
- thermonuclear flashes
- Fortran + MPI
- Argonne

ParaProf – Scalable Histogram

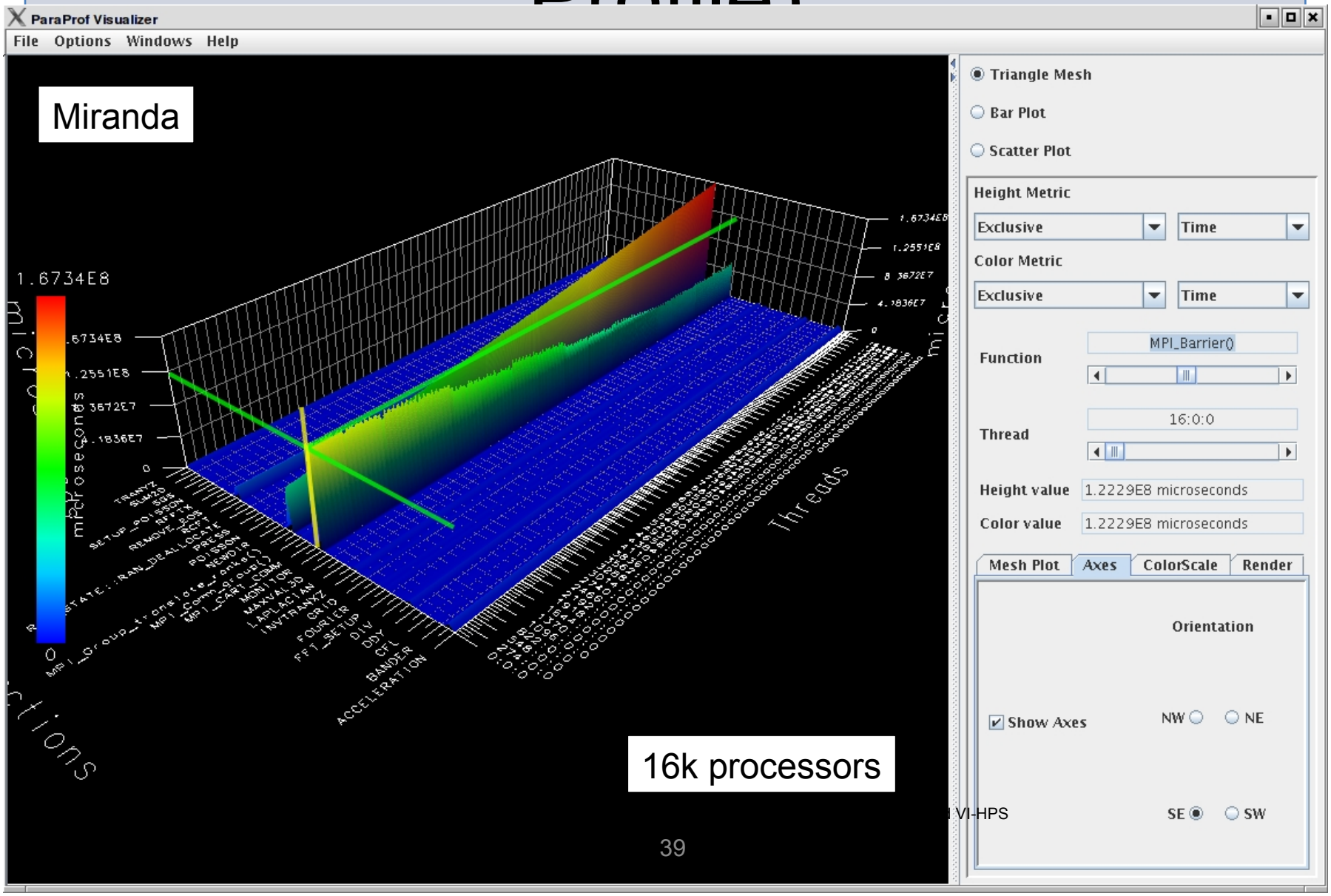




ParaProf – 3D View (Full Profile)

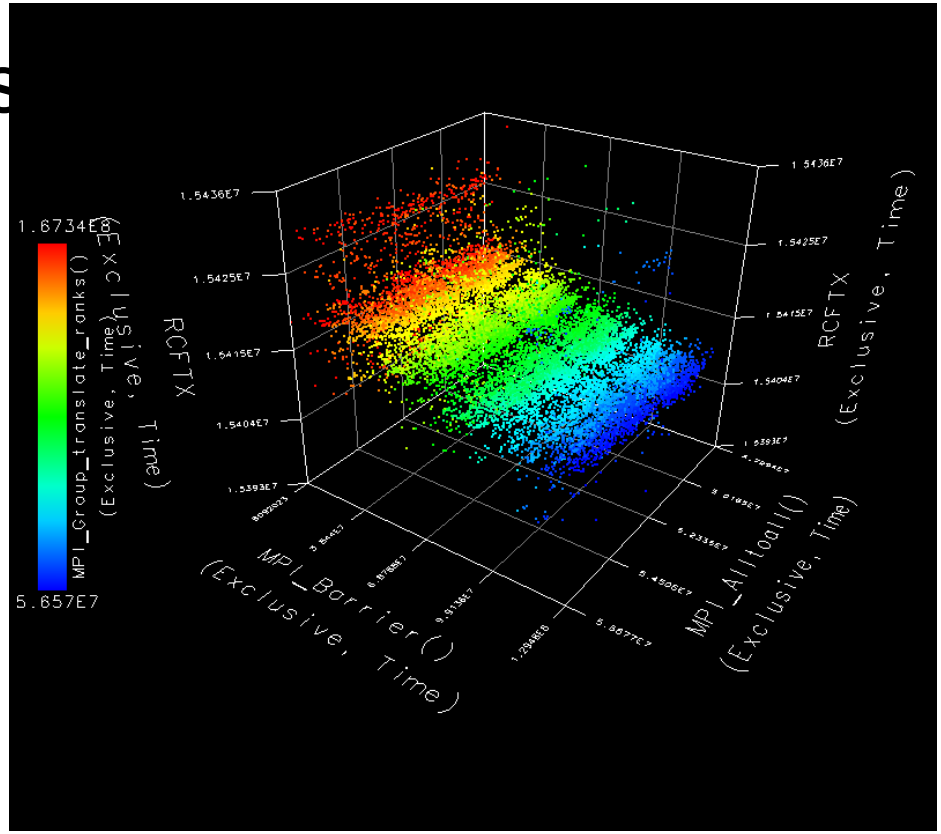


ParaProf – 3D View (Full Profile)



ParaProf – 3D Scatterplot

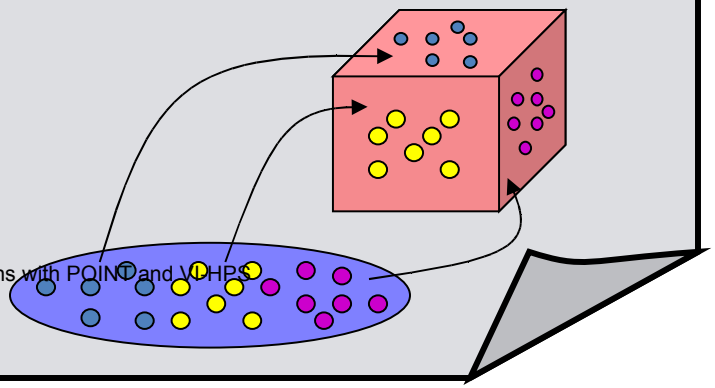
- Each point is a “thread” of execution
- A total of four metrics shown in relation
- ParaProf’s visualization library
  - JOGL
- Miranda



Performance Mapping

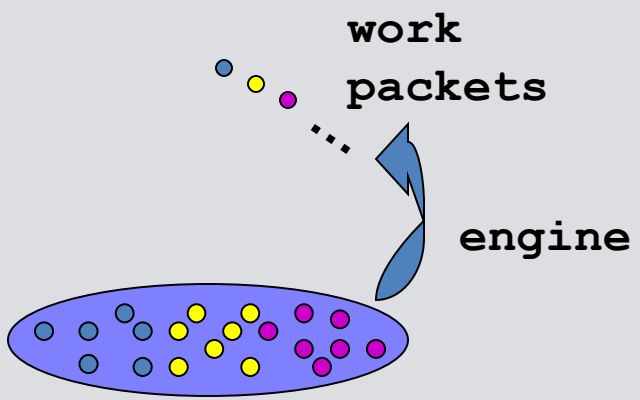
- Example: Particles distributed on cube

```
Particle* P[MAX]; /* Array of particles */
int GenerateParticles() {
  /* distribute particles over all faces of the cube */
  for (int face=0, last=0; face < 6; face++){
    /* particles on this face */
    int particles_on_this_face = num(face);
    for (int i=last; i < particles_on_this_face; i++) {
      /* particle properties are a function of face */
    P[i] = ... f(face);
      ...
    }
    last+= particles_on_this_face;
  }
}
```



Performance Mapping

```
int ProcessParticle(Particle *p) {
  /* perform some computation on p */
}
int main() {
  GenerateParticles();
  /* create a list of particles */
  for (int i = 0; i < N; i++)
    /* iterates over the list */
    ProcessParticle(P[i]);
}
```



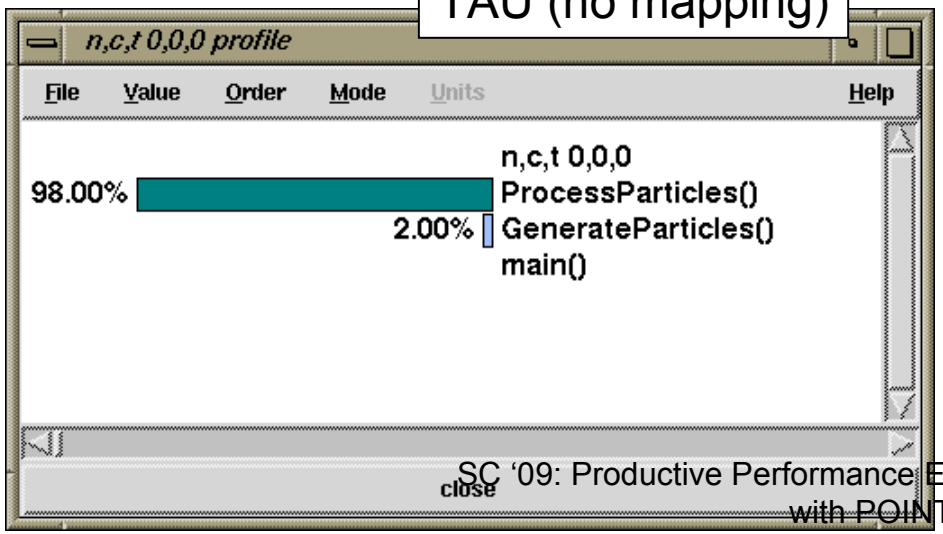
- How much time (flops) spent processing face i particles?
- What is the distribution of performance among faces?

No Mapping versus Mapping

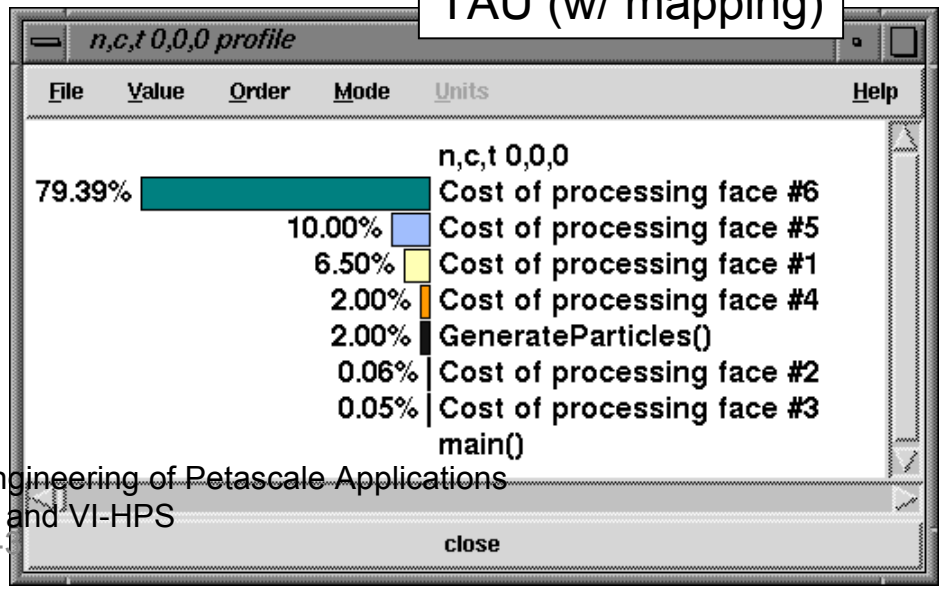
- Typical performance tools report performance with respect to routines
- Does not provide support for mapping

- TAU's performance mapping can observe performance with respect to scientist's programming and problem abstractions

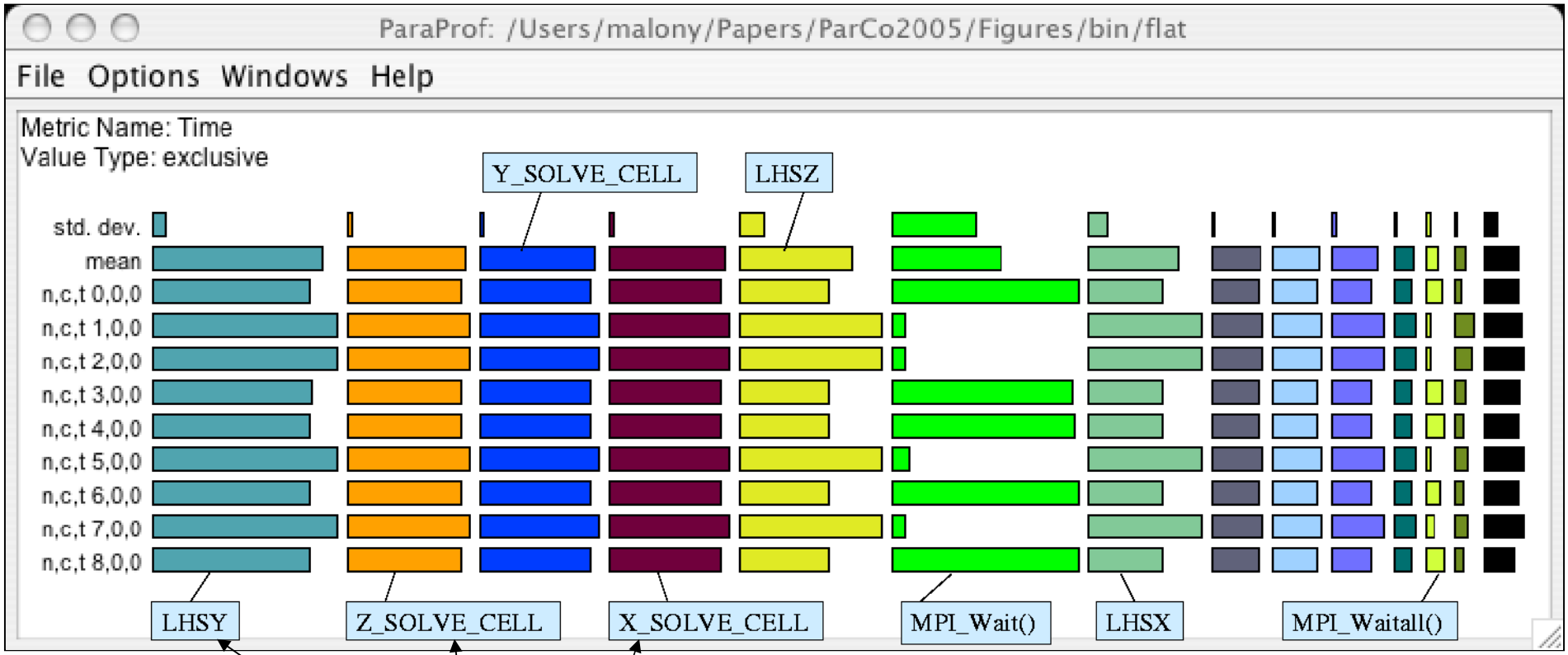
TAU (no mapping)



TAU (w/ mapping)



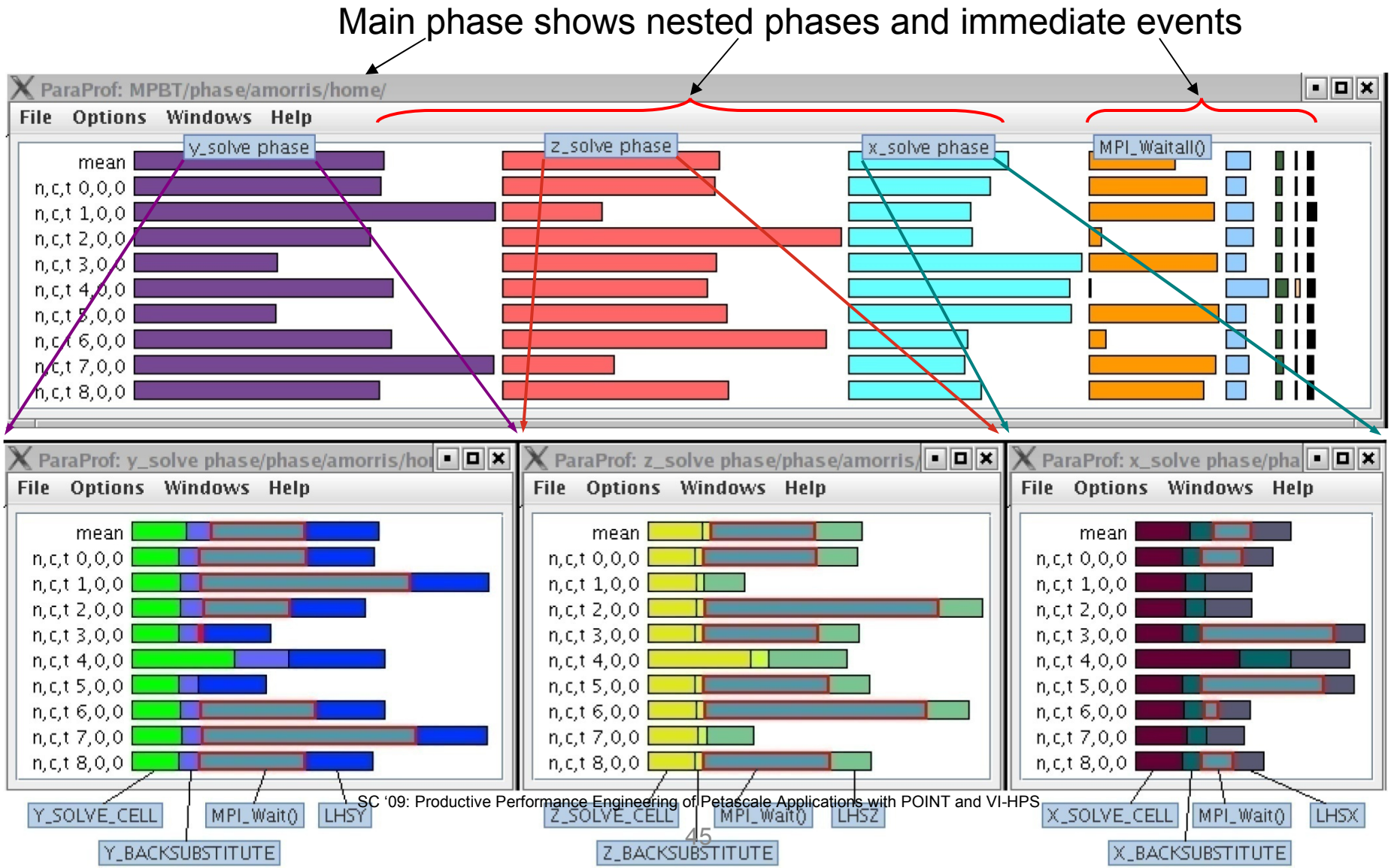
NAS BT – Flat Profile



Application routine names reflect phase semantics

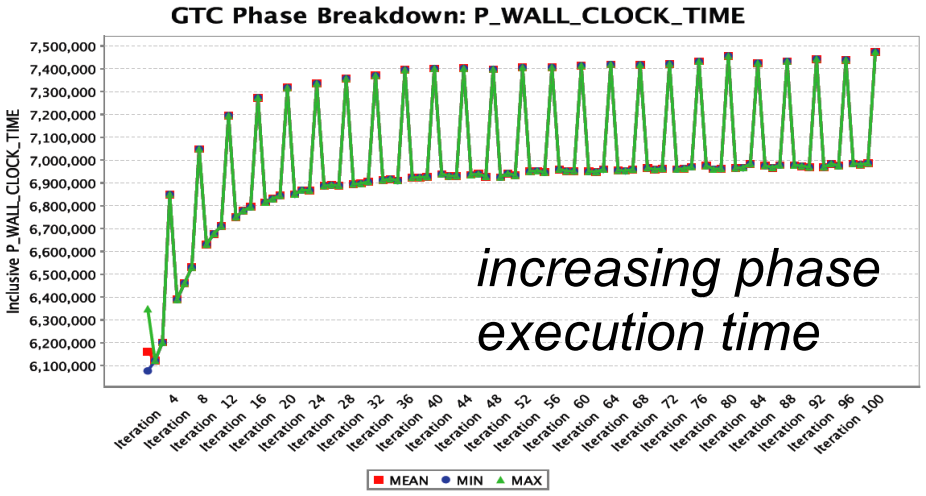
How is MPI_Wait() distributed relative to solver direction?

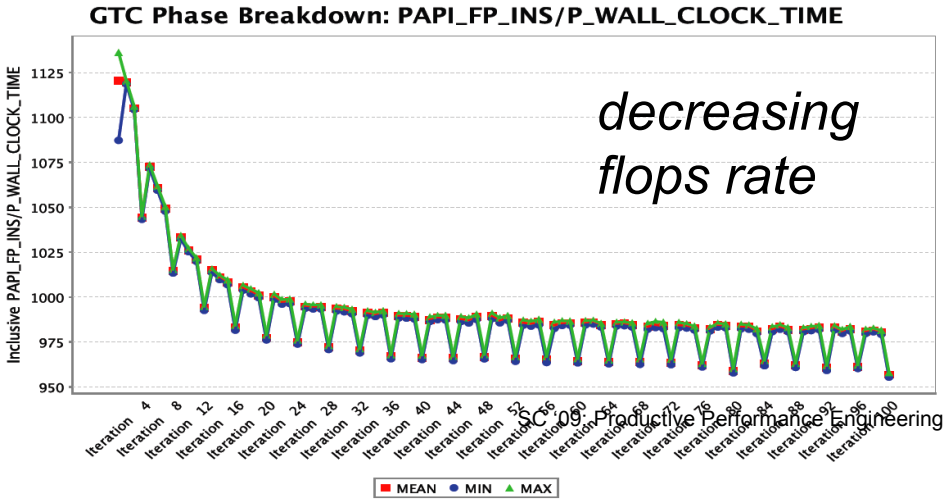
NAS BT – Phase Profile

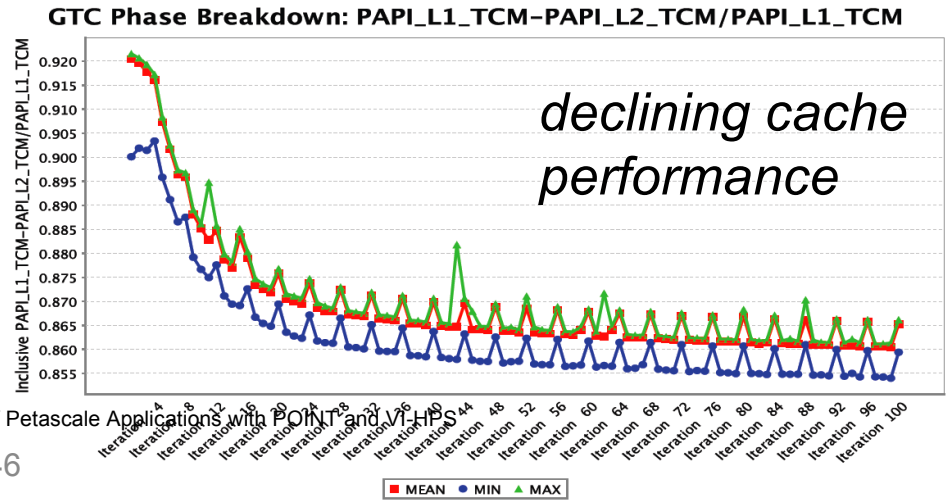


Phase Profiling of HW Counters

- GTC particle-in-cell simulation of fusion turbulence
- Phases assigned to iterations
- Poor temporal locality for one important data
- Automatically generated by PE2 python script

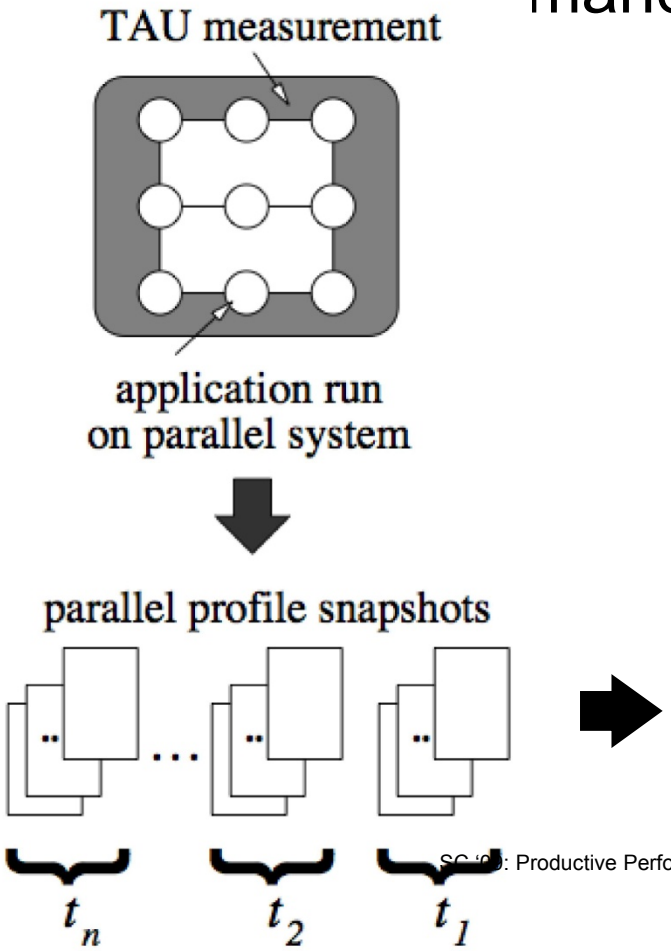


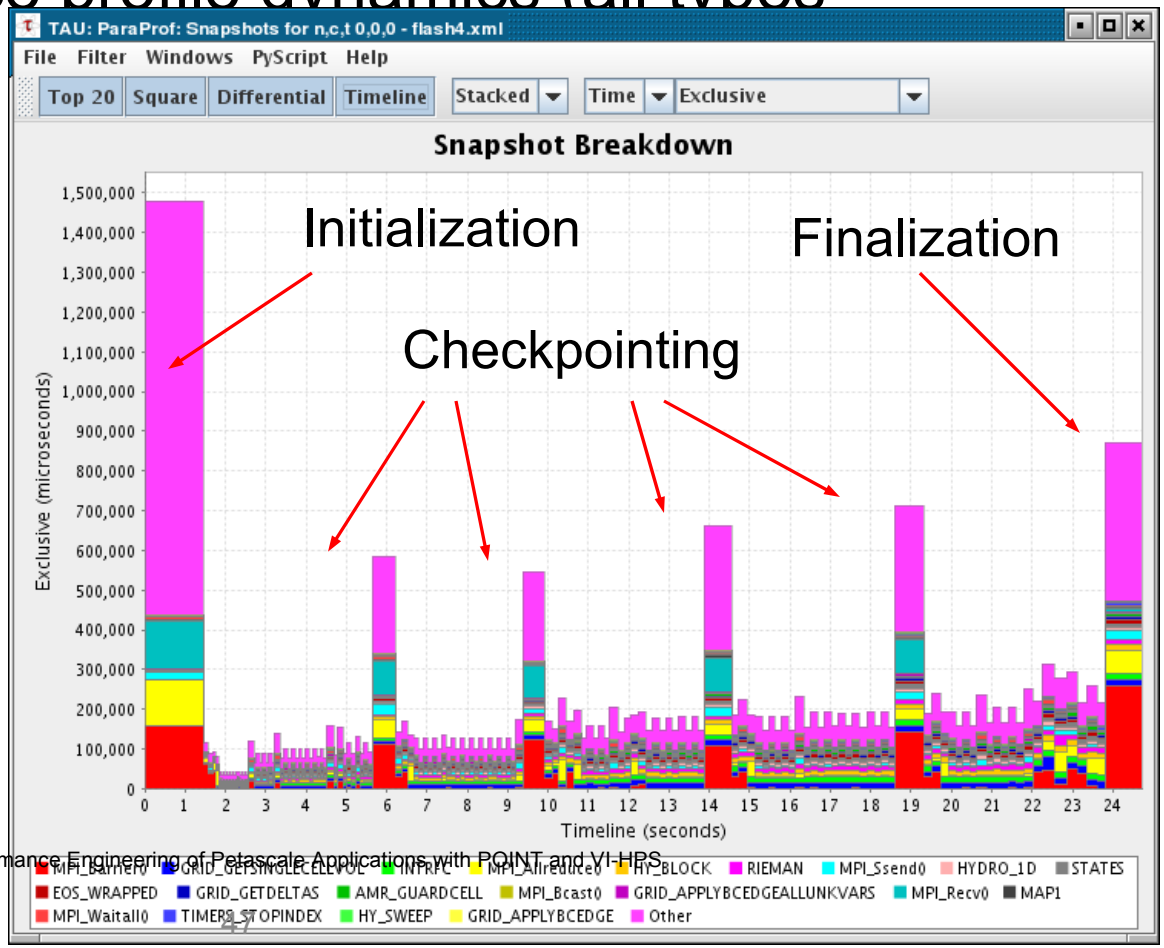




Profile Snapshots in ParaProf

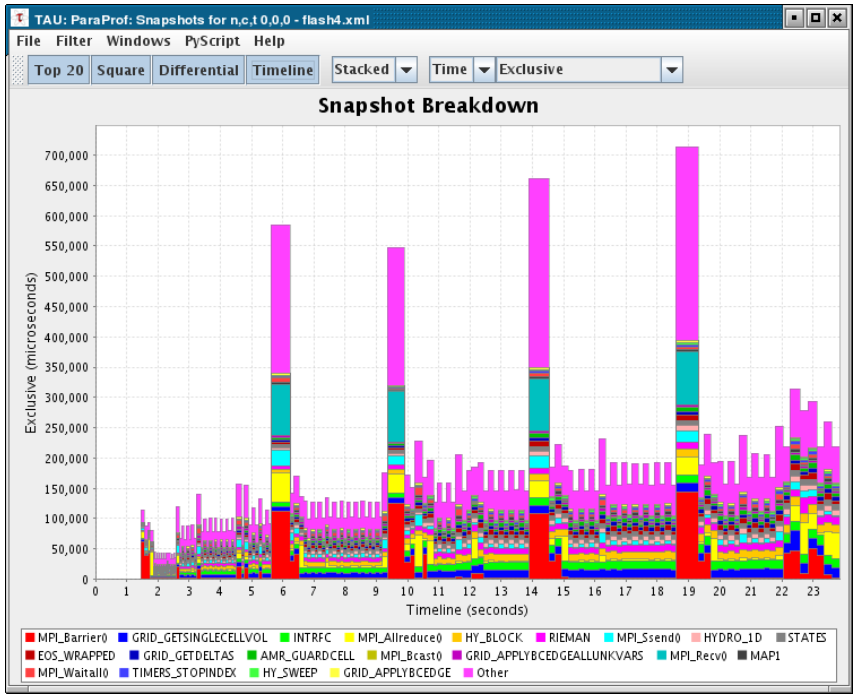
- Profile snapshots are parallel profiles recorded at runtime
- Shows performance profile dynamics (all types

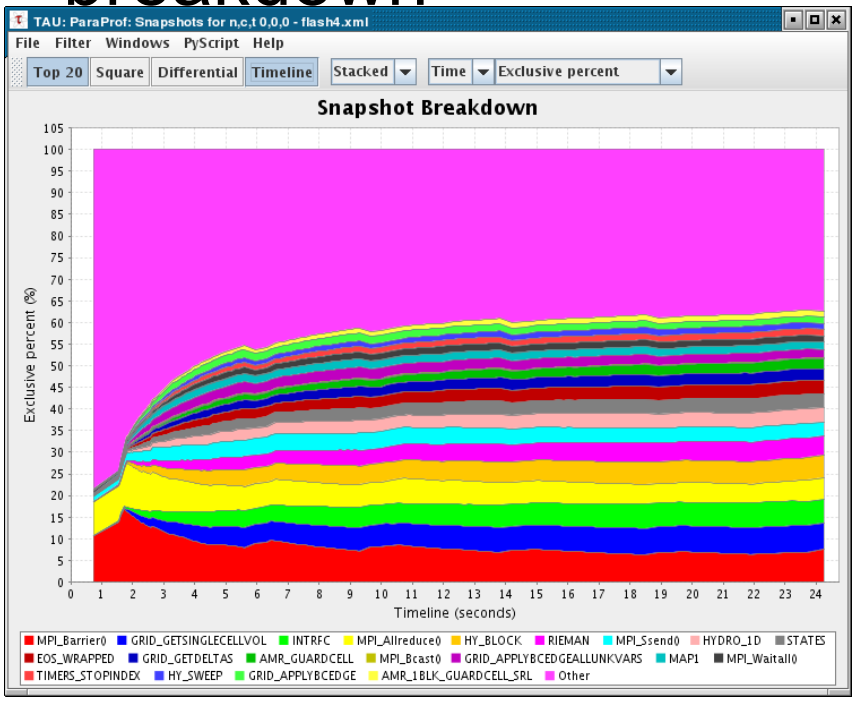




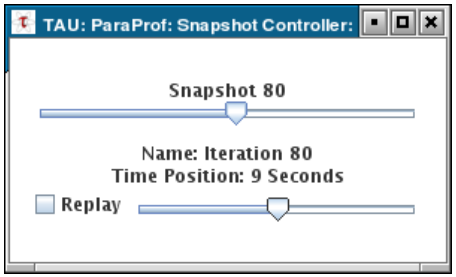
Profile Snapshot Views

- Only show main loop
- Percentage breakdown

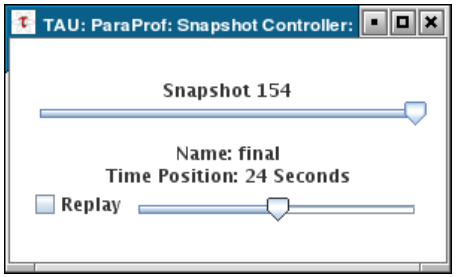


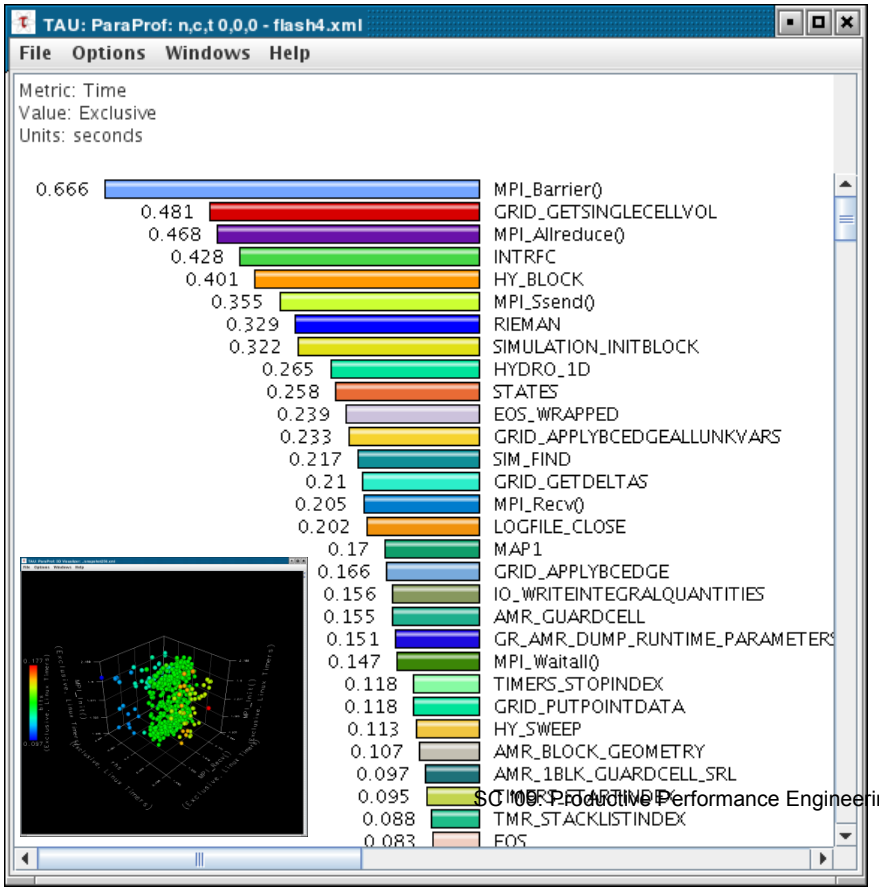


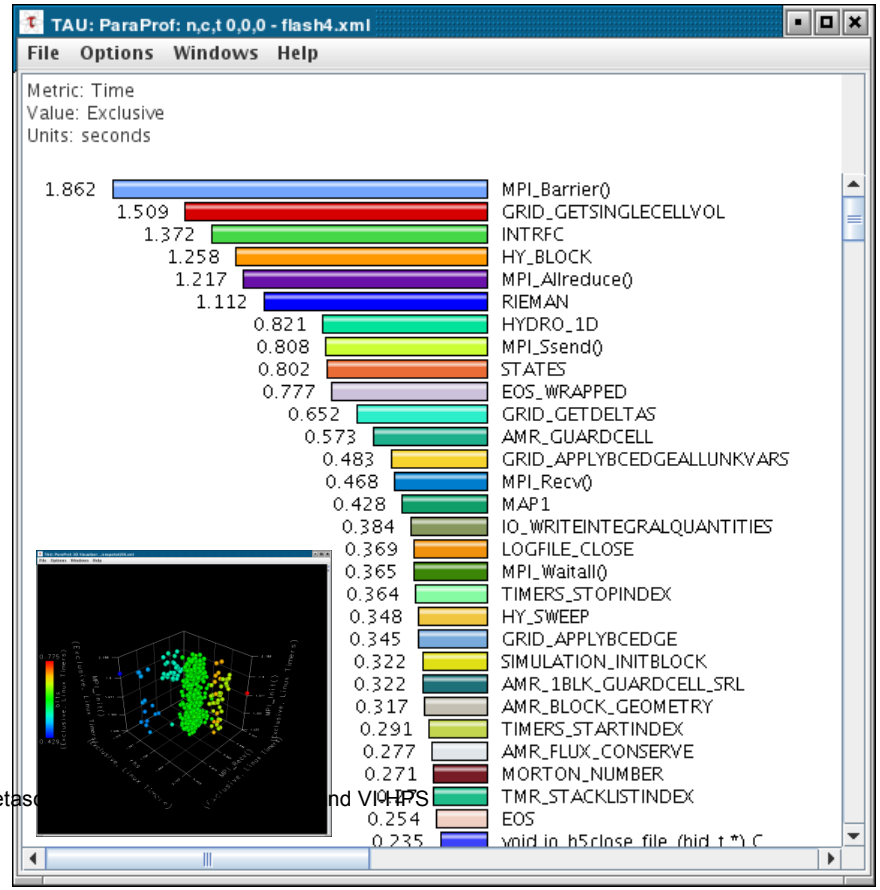
Snapshot Replay in ParaProf



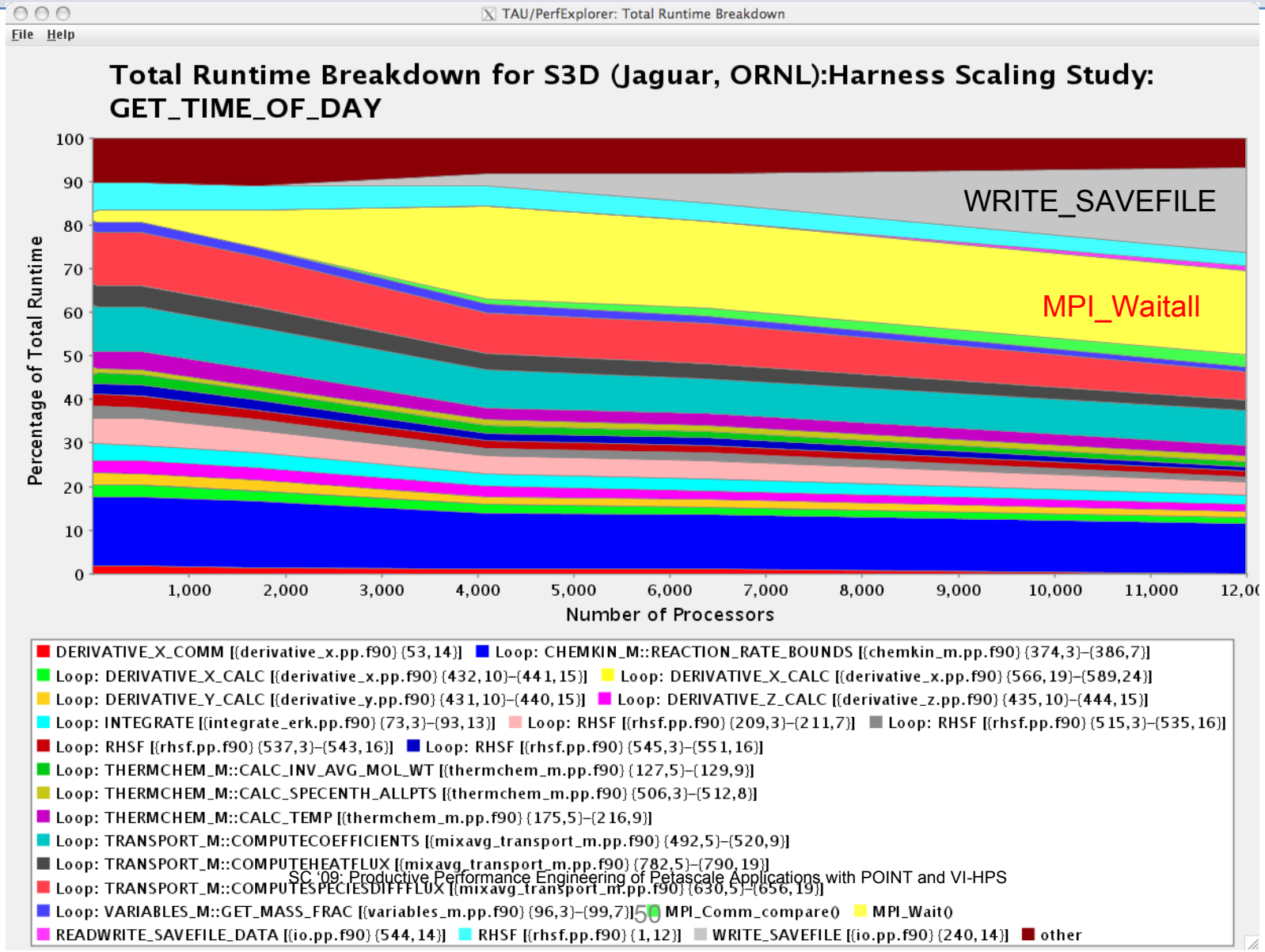
All windows dynamically update





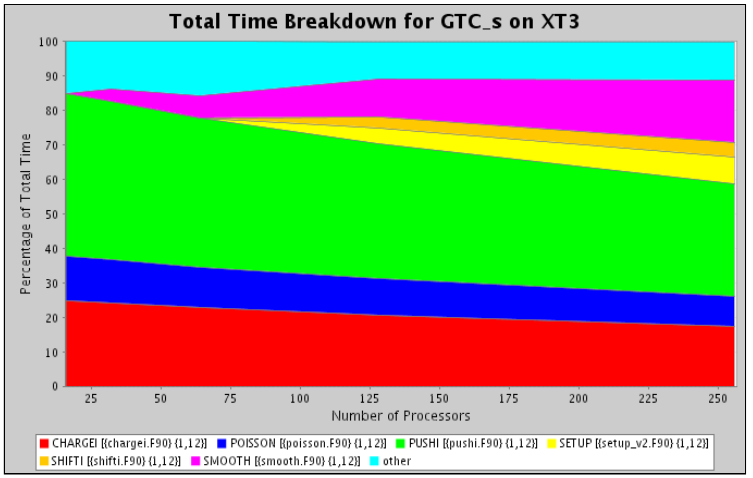


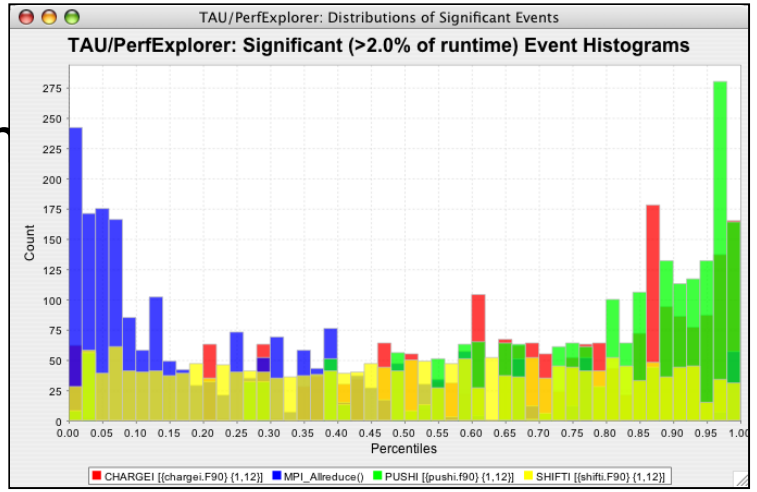
PerfExplorer – Runtime Breakdown



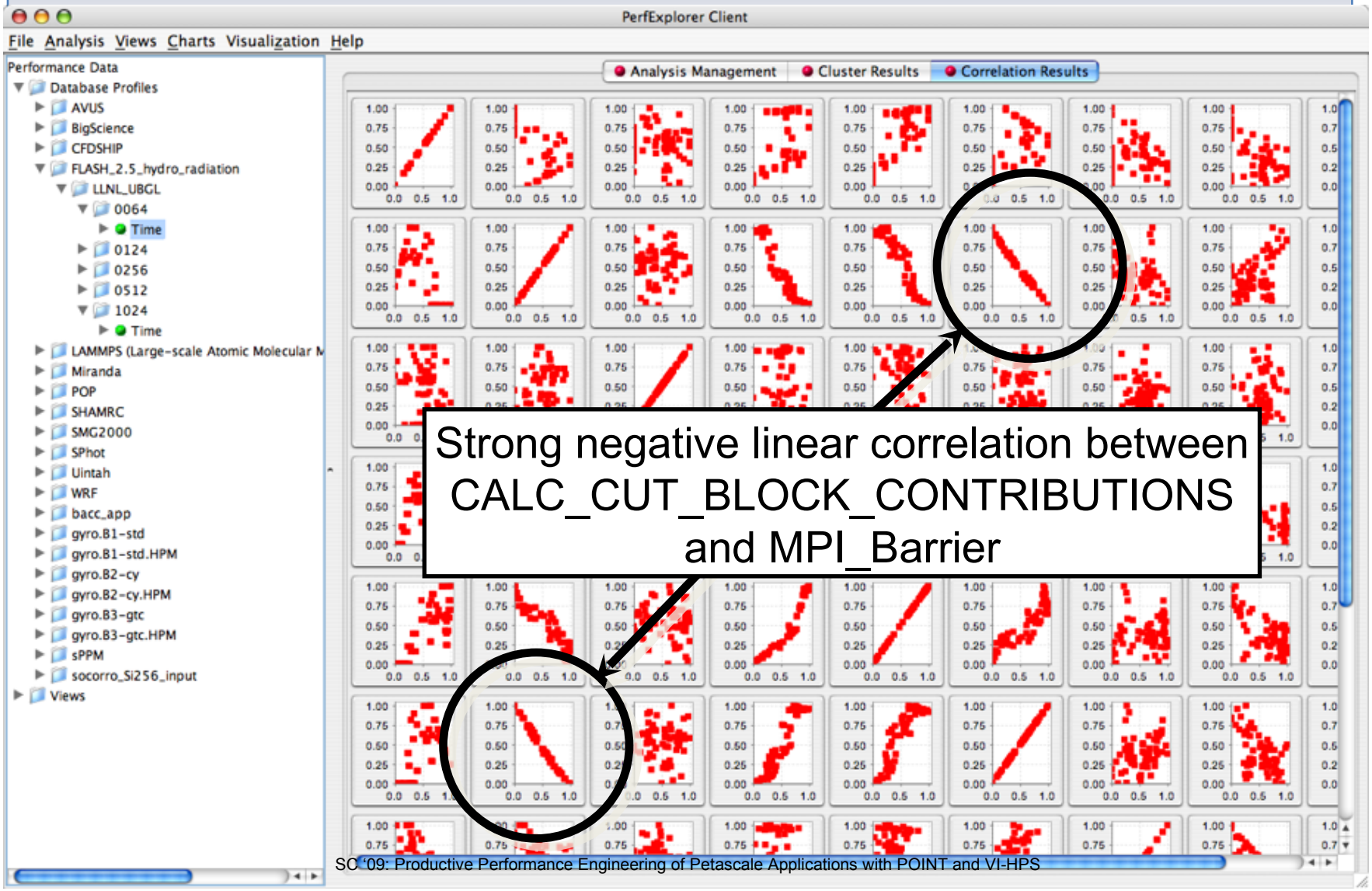
PerfExplorer – Relative Comparisons

- Total execution time
- Timesteps per second
- Relative efficiency
- Relative efficiency per event
- Relative speedup
- Relative speedup per event
- Group fraction of total
- Runtime breakdown
- Correlate events with total runtim
- Relative efficiency per phase
- Relative speedup per phase
- Distribution visualizations



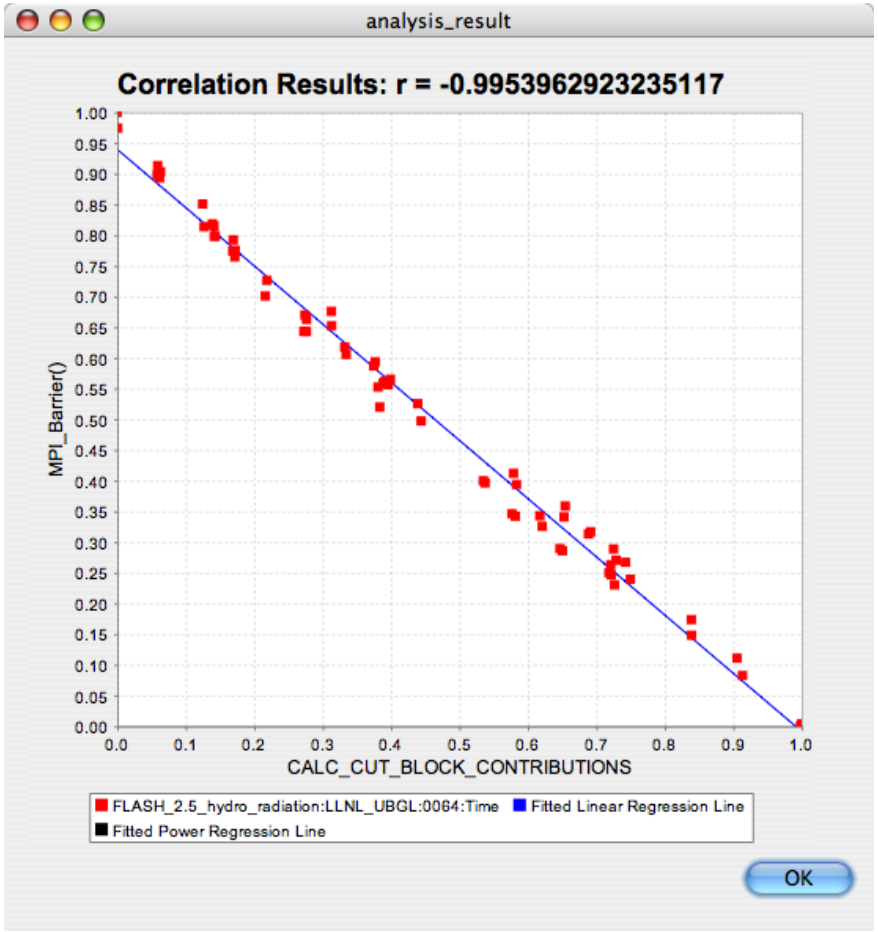


PerfExplorer – Correlation Analysis



PerfExplorer – Correlation Analysis

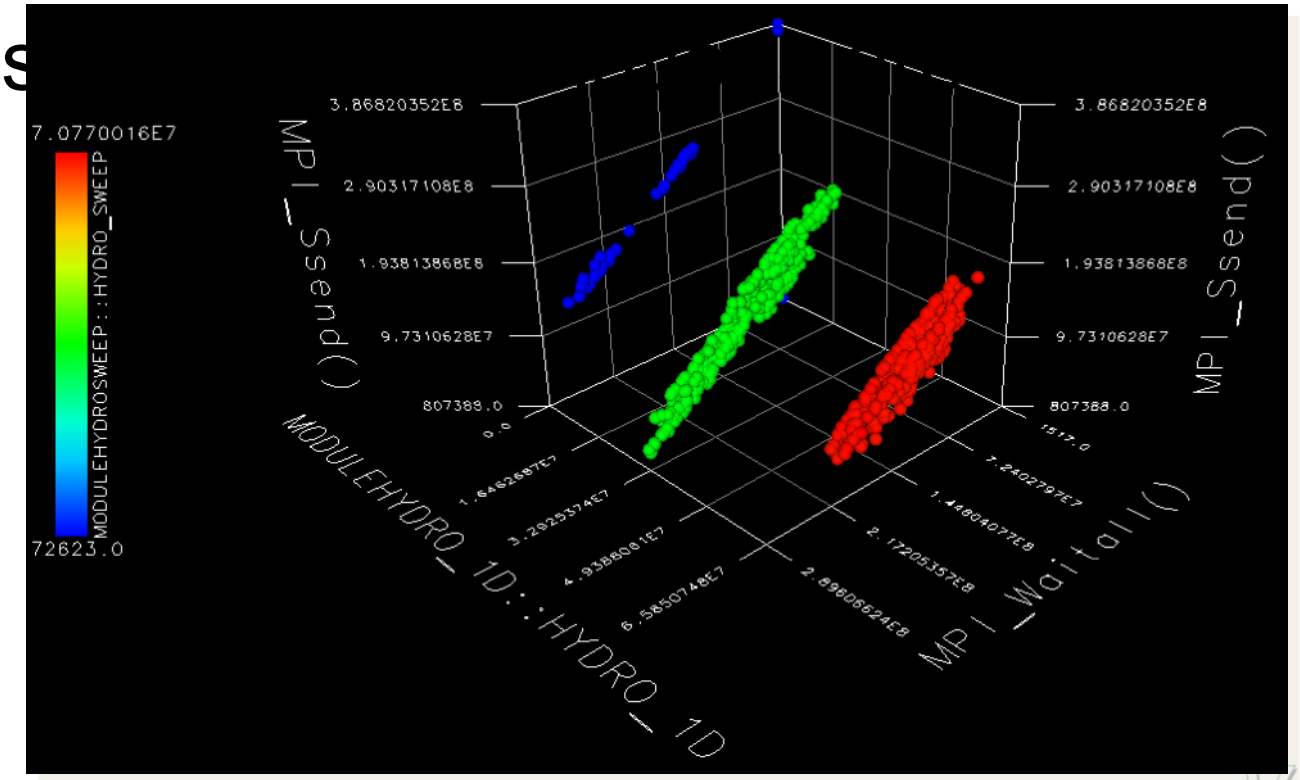
- -0.995 indicates strong, negative relationship
- As CALC_CUT_BLOCK_CONTRIBUTIONS() increases in execution time, MPI_Barrier() decreases



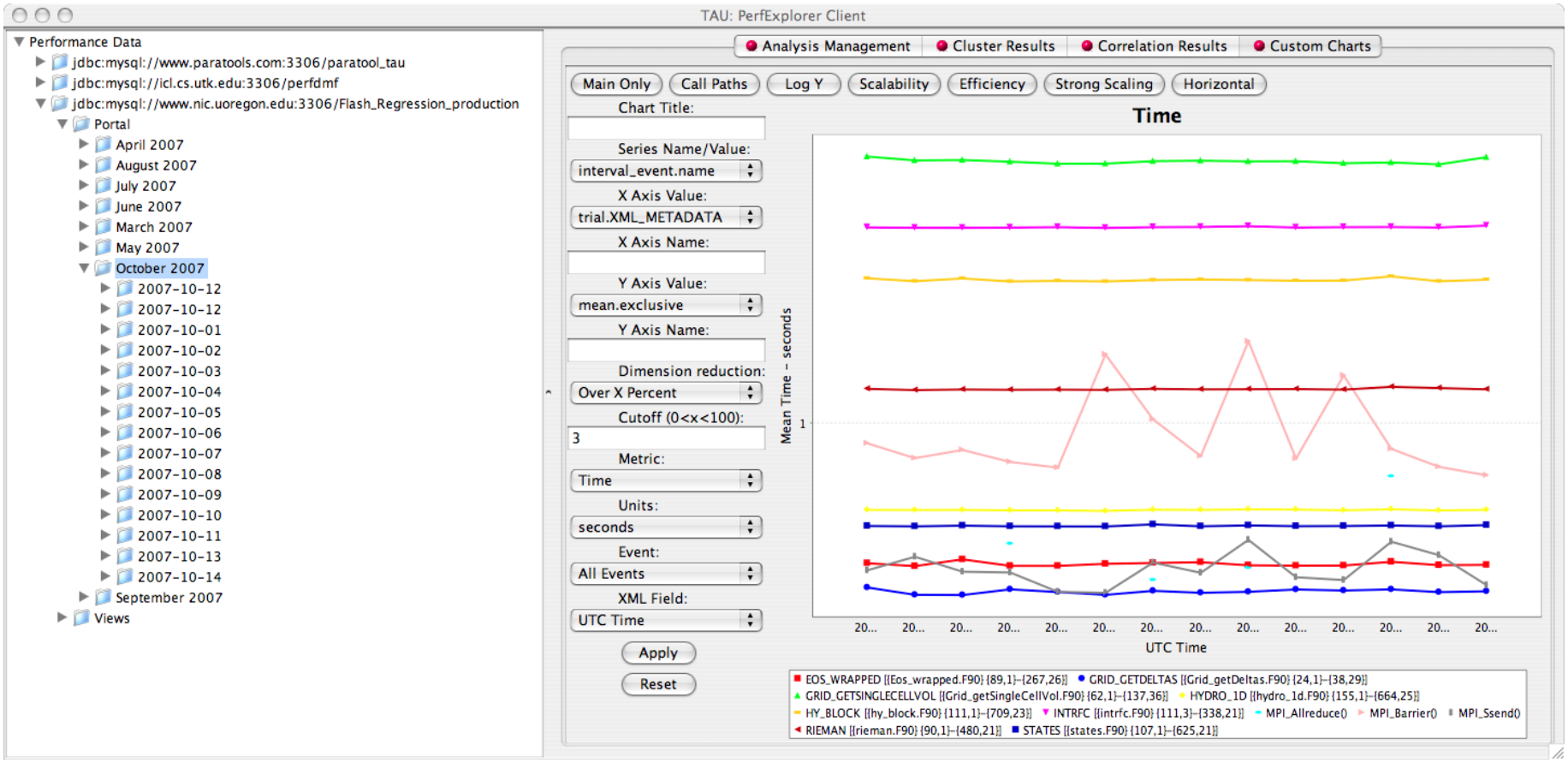
PerfExplorer – Cluster Analysis

PerfExplorer – Cluster Analysis

- Four significant events automatically selected
- Clus



PerfExplorer – Performance Regression



Other Projects in TAU

- TAU Portal
  - Support collaborative performance study
- Kernel-level system measurements (KTAU)
  - Application to OS noise analysis and I/O system analysis
- TAU performance monitoring
  - TAUoverSupermon and TAUoverMRNet
- PerfExplorer integration and expert-based analysis
  - OpenUH compiler optimizations
  - Computational quality of service in CCA
- Eclipse CDT and PTP integration
- Performance tools integration (NSF POINT project)

Using TAU

- Install TAU
  - % configure [options]; make clean install
- Modify application makefile and choose TAU configuration
  - Select TAU's stub makefile
  - Change name of compiler in makefile
- Set environment variables
  - Directory where profiles/traces are to be stored/counter selection
  - TAU options
- Execute application
  - % mpirun –np <procs> a.out;
- Analyze performance data
  - paraprof, vampir, pprof, paraver …

Application Build Environment

- Minimize impact on user's application build procedures
- Handle parsing, instrumentation, compilation, linking
- Dealing with Makefiles
  - Minimal change to application Makefile
  - Avoid changing compilation rules in application Makefile
  - No explicit inclusion of rules for process stages
- Some applications do not use Makefiles
  - Facilitate integration in whatever procedures used
- Two techniques:
  - TAU shell scripts (tau_<compiler>.sh)
    - Invokes all PDT parser, TAU instrumenter, and compiler
  - TAU_COMPILER

Configuring TAU

- TAU can measure several metrics with profiling and tracing approaches
- Different tools can also be invoked to instrument programs for TAU measurement
- Each configuration of TAU produces a measurement library for an architecture
- Each measurement configuration of TAU also creates a corresponding stub makefile that can be used to compile programs
- Typically configure multiple measurement libraries

TAU Measurement System Configuration

- configure [OPTIONS]
  - {-c++=<CC>, -cc=<cc>} Specify C++ and C compilers
  - -pdt=<dir> Specify location of PDT
  - -opari=<dir> Specify location of Opari OpenMP tool
  - -papi=<dir> Specify location of PAPI
  - -vampirtrace=<dir> Specify location of VampirTrace
  - -mpi[inc/lib]=<dir> Specify MPI library instrumentation
  - -dyninst=<dir> Specify location of DynInst Package
  - -shmem[inc/lib]=<dir> Specify PSHMEM library instrumentation
  - -python[inc/lib]=<dir> Specify Python instrumentation
  - -tag=<name> Specify a unique configuration name
  - -epilog=<dir> Specify location of EPILOG
  - -slog2 Build SLOG2/Jumpshot tracing package
  - -otf=<dir> Specify location of OTF trace package
  - -arch=<architecture> Specify architecture explicitly (bgl, xt3,ibm64,ibm64linux…)
  - {-pthread, -sproc} Use pthread or SGI sproc threads
  - -openmp Use OpenMP threads
  - -jdk=<dir> Specify Java instrumentation (JDK)
  - -fortran=[vendor] Specify Fortran compiler

TAU Measurement System Configuration

- configure [OPTIONS]
  - -TRACE Generate binary TAU traces
  - -PROFILE (default) Generate profiles (summary)
  - -PROFILECALLPATH Generate call path profiles
  - -PROFILEPHASE Generate phase based profiles
  - -PROFILEMEMORY Track heap memory for each routine
  - -PROFILEHEADROOM Track memory headroom to grow
  - -MULTIPLECOUNTERS Use hardware counters + time
  - -COMPENSATE Compensate timer overhead
  - -CPUTIME Use usertime+system time
  - -PAPIWALLCLOCK Use PAPI's wallclock time
  - -PAPIVIRTUAL Use PAPI's process virtual time
  - -SGITIMERS Use fast IRIX timers
  - -LINUXTIMERS Use fast x86 Linux timers

TAU Configuration – Examples

- Configure using PDT and MPI for x86_64 Linux

  ./configure –pdt=/usr/pkgs/pkgs/pdtoolkit-3.14
  -mpiinc=/usr/pkgs/mpich/include -mpilib=
  /usr/pkgs/mpich/lib
  -mpilibrary='-lmpich -L/usr/gm/lib64 -lgm -lpthread -ldl'

- Use PAPI counters (one or more) with C/C++/F90 automatic instrumentation for Cray CNL. Also instrument the MPI library. Use PGI compilers.

  ./configure -arch=craycnl -cc=cc -c++=CC -fortran=pgi -papi=
  /opt/xt-tools/papi/3.2.1 -mpi -MULTIPLECOUNTERS; make
  clean install

- Stub makefiles

/usr/pkgs/tau/x86_64/lib/Makefile.tau-mpi-pdt-pgi

/usr/pkgs/tau/x86_64/lib/Makefile.tau-multiplecounters-mpi-papi-pdt-pgi

Stub Makefiles Configuration Parameters

- TAU scripts use stub makefiles to select performance measurements
- Variables:
  - TAU_CXX Specify the C++ compiler used by TAU
  - TAU_CC, TAU_F90 Specify the C, F90 compilers
  - TAU_DEFS Defines used by TAU (add to CFLAGS)
  - TAU_LDFLAGS Linker options (add to LDFLAGS)
  - TAU_INCLUDE Header files include path (add to CFLAGS)
  - TAU_LIBS Statically linked TAU library (add to LIBS)
  - TAU_SHLIBS Dynamically linked TAU library
  - TAU_MPI_LIBS TAU's MPI wrapper library for C/C++
  - TAU_MPI_FLIBS TAU's MPI wrapper library for F90
  - TAU_FORTRANLIBS Must be linked in with C++ linker for F90
  - TAU_CXXLIBS Must be linked in with F90 linker
  - TAU_INCLUDE_MEMORY Use TAU's malloc/free wrapper lib
  - TAU_DISABLE TAU's dummy F90 stub library
  - TAU_COMPILER Instrument using tau_compiler.sh script

TAU Measurement Configuration

- % cd /opt/tau-2.18/x86_64/lib; ls Makefile.*
  - Makefile.tau-pdt
  - Makefile.tau-mpi-pdt
  - Makefile.tau-callpath-mpi-pdt
  - Makefile.tau-mpi-pdt-trace
  - Makefile.tau-mpi-compensate-pdt
  - Makefile.tau-multiplecounters-mpi-papi-pdt
  - Makefile.tau-multiplecounters-mpi-papi-pdt-trace
  - Makefile.tau-pthread-pdt…
- For an MPI+F90 application, you may want to start with:
  - Makefile.tau-mpi-pdt
  - Supports MPI instrumentation & PDT for automatic source instrumentation
- % setenv TAU_MAKEFILE /opt/tau-2.18/x86_64/lib/Makefile.tau-mpi-pdt

-PROFILE Option

- Generates flat profiles
  - One for each MPI process
  - It is the default option.
- Uses wallclock time
  - gettimeofday() sys call
- Calculates exclusive, inclusive time spent in each timer and number of calls

Generating a Flat Profile with MPI

```
% setenv TAU_MAKEFILE /opt/tau-2.18/x86_64
                     /lib/Makefile.tau-mpi-pdt
% set path=(/opt/tau-2.18/x86_64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)

% qsub run.job
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.

% paraprof app.ppk
```

Generating a Loop-level Profile

```
% setenv TAU_MAKEFILE /opt/tau-2.18/x86_64
                      /lib/Makefile.tau-mpi-pdt
% setenv TAU_OPTIONS '-optTauSelectFile=select.tau -optVerbose'
% cat select.tau
  BEGIN_INSTRUMENT_SECTION
  loops routine="#"
  END_INSTRUMENT_SECTION

% set path=(/opt/tau-2.18/x86_64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% qsub  run.job
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.

% paraprof app.ppk
```

Compiler-based Instrumentation

```
% setenv TAU_MAKEFILE /opt/tau-2.18/x86_64
                      /lib/Makefile.tau-mpi
% setenv TAU_OPTIONS '-optCompInst -optVerbose'
% % set path=(/opt/tau-2.18/x86_64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)

% qsub  run.job
% paraprof --pack app.ppk
  Move the app.ppk file to your desktop.
% paraprof app.ppk
```

-MULTIPLECOUNTERS Option

- Instead of one metric, profile or trace with more than one metric
  - Set environment variables COUNTER[1-25] to specify the metric
    - % setenv COUNTER1 GET_TIME_OF_DAY
    - % setenv COUNTER2 PAPI_L2_DCM
    - % setenv COUNTER3 PAPI_FP_OPS
    - % setenv COUNTER4 PAPI_NATIVE_<native_event>
    - % setenv COUNTER5 P_WALL_CLOCK_TIME …
- When used with –TRACE option, the first counter must be GET_TIME_OF_DAY
    - % setenv COUNTER1 GET_TIME_OF_DAY
    - Provides a globally synchronized real time clock for tracing
- -multiplecounters appears in the name of the stub Makefile
- Often used with –papi=<dir> to measure hardware performance counters and time
- papi_native_avail and papi_avail are two useful tools

Generate a PAPI profile

```
% setenv TAU_MAKEFILE /opt/tau-2.18/x86_64
                        /lib/Makefile.tau-multiplecounters-papi-mpi-pdt
% setenv TAU_OPTIONS '-optTauSelectFile=select.tau -optVerbose'
% cat select.tau
  BEGIN_INSTRUMENT_SECTION
  loops routine="#"
  END_INSTRUMENT_SECTION

% set path=(/opt/tau-2.18/x86_64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% setenv COUNTER1 GET_TIME_OF_DAY
% setenv COUNTER2 PAPI_FP_INS
% qsub  run.job
% paraprof --pack app.ppk
   Move the app.ppk file to your desktop.
% paraprof app.ppk
  Choose Options -> Show Derived Panel -> Arg 1 = PAPI_FP_INS,
    Arg 2 = GET_TIME_OF_DAY, Operation = Divide -> Apply, choose.
```

-PROFILECALLPATH Option

- Generates profiles that show the calling order (edges and nodes in callgraph)
  - A=>B=>C shows the time spent in C when it was called by B and B was called by A
  - Control the depth of callpath using TAU_CALLPATH_DEPTH environment variable
  - -callpath in the name of the stub Makefile name or setting TAU_CALLPATH= 1 at runtime
    (TAU v2.18.1+)

-DEPTHLIMIT Option

- Allows users to enable instrumentation at runtime based on the depth of a calling routine on a callstack
  - Disables instrumentation in all routines a certain depth away from the root in a callgraph
- TAU_DEPTH_LIMIT environment variable specifies depth
  - % setenv TAU_DEPTH_LIMIT 1
  - enables instrumentation in only “main”
  - % setenv TAU_DEPTH_LIMIT 2
  - enables instrumentation in main and routines that are directly called by main
- Stub makefile has -depthlimit in its name:
  - setenv TAU_MAKEFILE <taudir>/<arch>/lib/Makefile.tau-icpc-mpi-depthlimit-pdt

Generate a Callpath Profile

```
% setenv TAU_MAKEFILE /opt/tau-2.18/x86_64
                      /lib/Makefile.tau-callpath-mpi-pdt
% set path=(/opt/tau-2.18/x86_64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% setenv TAU_CALLPATH_DEPTH 100

NOTE: In TAU v2.18.1+ you may simply use:
% setenv TAU_CALLPATH 1
to generate the callpath profiles without any recompilation.
% qsub  run.job
% paraprof --pack app.ppk
   Move the app.ppk file to your desktop.
% paraprof app.ppk
(Windows -> Thread -> Call Graph)
```

-TRACE Configuration Option

- Generates event-trace logs, rather than summary profiles
- Traces show when and where an event occurred in terms of location and the process that executed it
- Traces from multiple processes are merged:
  - % tau_treemerge.pl
    - generates tau.trc and tau.edf as merged trace and event definition file
- TAU traces can be converted to Vampir's OTF/VTF3, Jumpshot SLOG2, Paraver trace formats:
  - % tau2otf tau.trc tau.edf app.otf
  - % tau2vtf tau.trc tau.edf app.vpt.gz
  - % tau2slog2 tau.trc tau.edf -o app.slog2
  - % tau_convert -paraver tau.trc tau.edf app.prv
- Stub Makefile has -trace in its name
  - % setenv TAU_MAKEFILE <taudir>/<arch>/lib/Makefile.tau-icpc-mpi-pdt-trace

Generate a Trace File

```
% setenv TAU_MAKEFILE /opt/tau-2.18/x86_64
                /lib/Makefile.tau-mpi-pdt-trace
% set path=(/opt/tau-2.18/x86_64/bin $path)
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
% qsub  run.job
% tau_treemerge.pl
(merges binary traces to create tau.trc and tau.edf files)
JUMPSHOT:
% tau2slog2 tau.trc tau.edf -o app.slog2
% jumpshot app.slog2
   OR
VAMPIR:
% tau2otf tau.trc tau.edf app.otf -n 4 -z
(4 streams, compressed output trace)
% vampir app.otf
(or vng client with vngd server)
```

Instrumentation Specification

```
% tau_instrumentor
Usage : tau_instrumentor <pdbfile> <sourcefile> [-o <outputfile>] [-noinline]
[-g groupname] [-i headerfile] [-c|-c++|-fortran] [-f <instr_req_file> ]
For selective instrumentation, use -f option
% tau_instrumentor foo.pdb foo.cpp -o foo.inst.cpp -f selective.dat
% cat selective.dat
# Selective instrumentation: Specify an exclude/include list of routines/files.
BEGIN_EXCLUDE_LIST
void quicksort(int *, int, int)
void sort_5elements(int *)
void interchange(int *, int *)
END_EXCLUDE_LIST

BEGIN_FILE_INCLUDE_LIST
Main.cpp
Foo?.c
*.C
END_FILE_INCLUDE_LIST
# Instruments routines in Main.cpp, Foo?.c and *.C files only
# Use BEGIN_[FILE]_INCLUDE_LIST with END_[FILE]_INCLUDE_LIST
```

Outer Loop Level Instrumentation

```
BEGIN_INSTRUMENT_SECTION
loops file="loop_test.cpp" routine="multiply"
# it also understands # as the wildcard in routine name
# and * and ? wildcards in file name.
# You can also specify the full
# name of the routine as is found in profile files.
#loops file="loop_test.cpp" routine="double multiply#"
END_INSTRUMENT_SECTION

% pprof
NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0         0.12       25,162           1           1   25162827 int main(int, char **)
100.0        0.175       25,162           1           4   25162707 double multiply()
 90.5       22,778       22,778           1           0   22778959 Loop: double multiply()[
file = <loop_test.cpp> line,col = <23,3> to <30,3> ]
  9.3        2,345        2,345           1           0    2345823 Loop: double multiply()[
file = <loop_test.cpp> line,col = <38,3> to <46,7> ]
  0.1           33           33           1           0      33964 Loop: double
multiply()[ file = <loop_test.cpp> line,col = <16,10> to <21,12> ]
```

Support Acknowledgements

- Department of Energy (DOE)
  - Office of Science
    - MICS, Argonne National Lab
  - ASC/NNSA
    - University of Utah ASC/NNSA Level 1
    - ASC/NNSA, Lawrence Livermore National La
- Department of Defense (DoD)
  - HPC Modernization Office (HPCMO)
- NSF Software Development for Cyberinfrastructure (SD
- Research Centre Juelich
- Los Alamos National Laboratory
- TU Dresden
- ParaTools, Inc.

For more information

- TAU Website:
  http://tau.uoregon.edu
  - Software
  - Release notes
  - Documentation