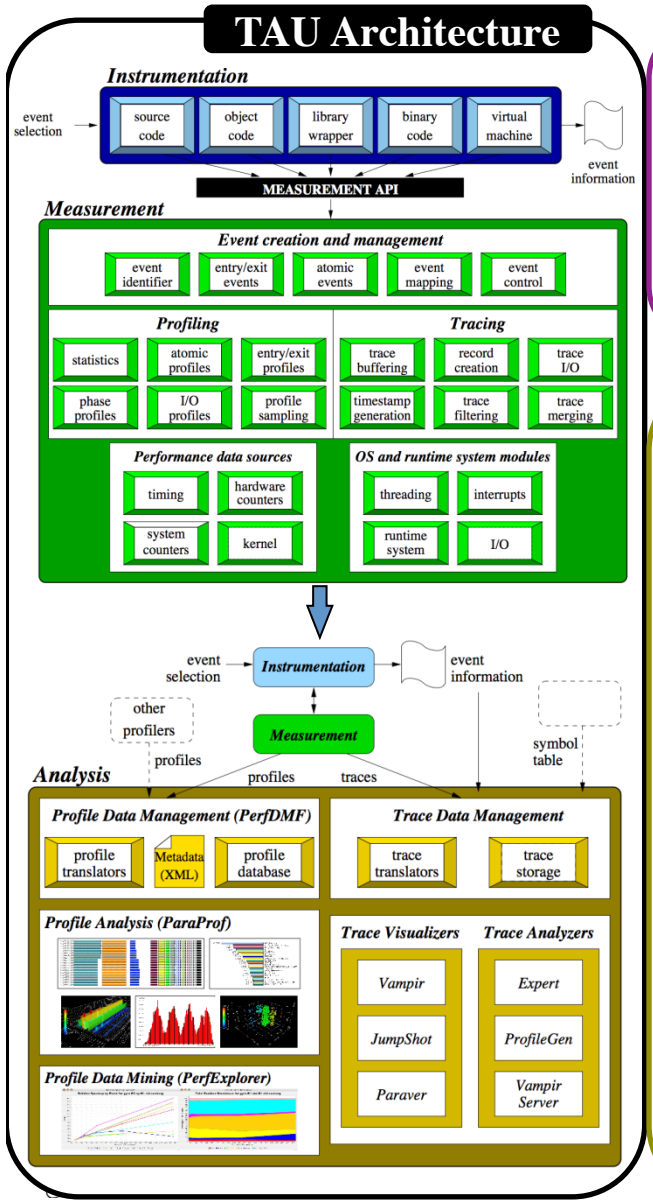
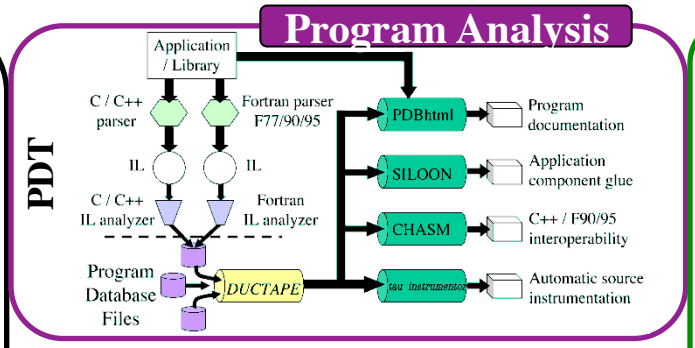
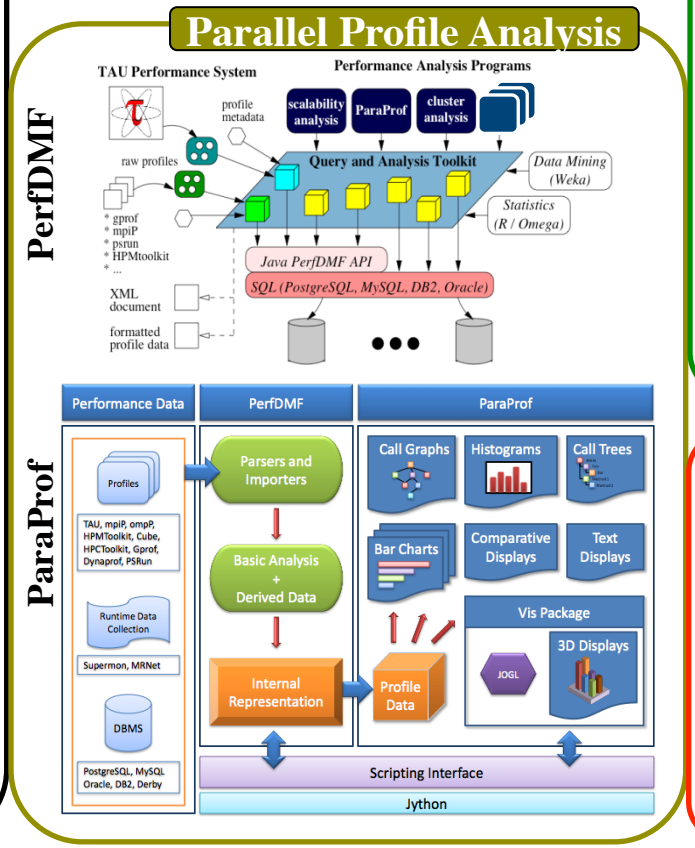
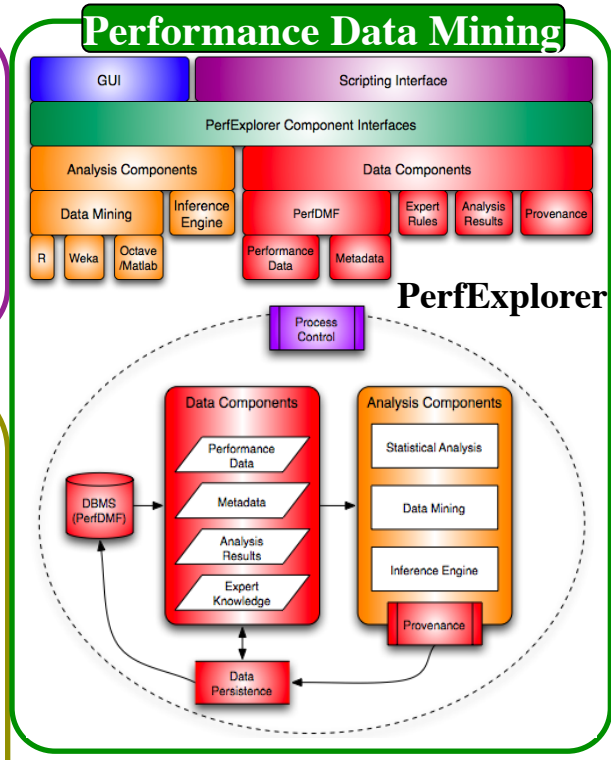
APPENDIX

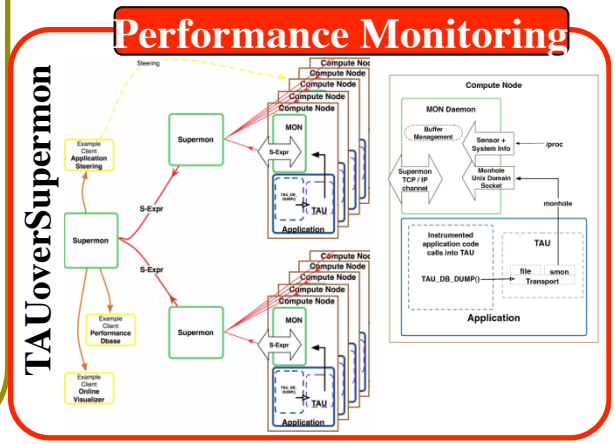
TAU Performance System Components



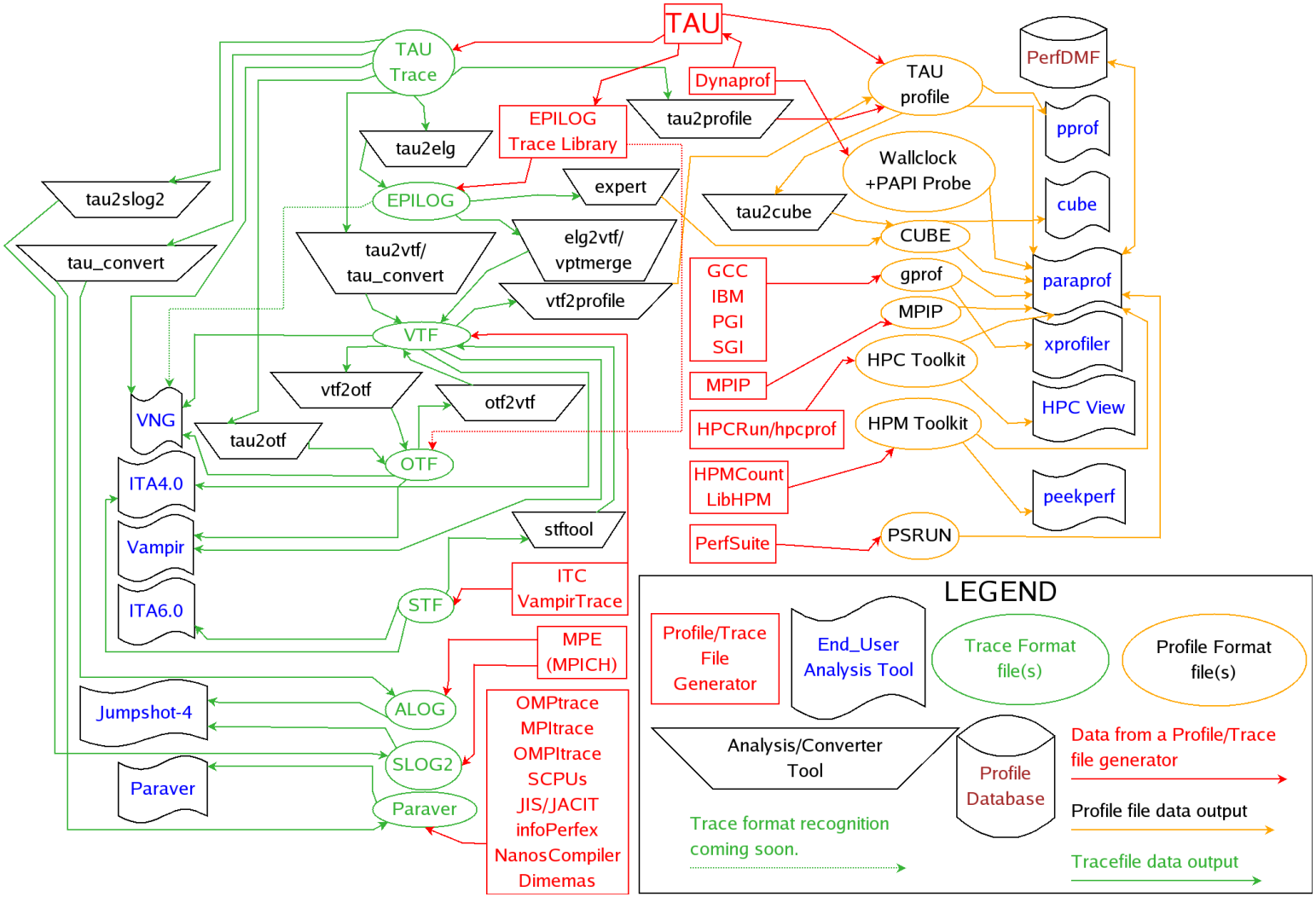








Building Bridges to Other Tools



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

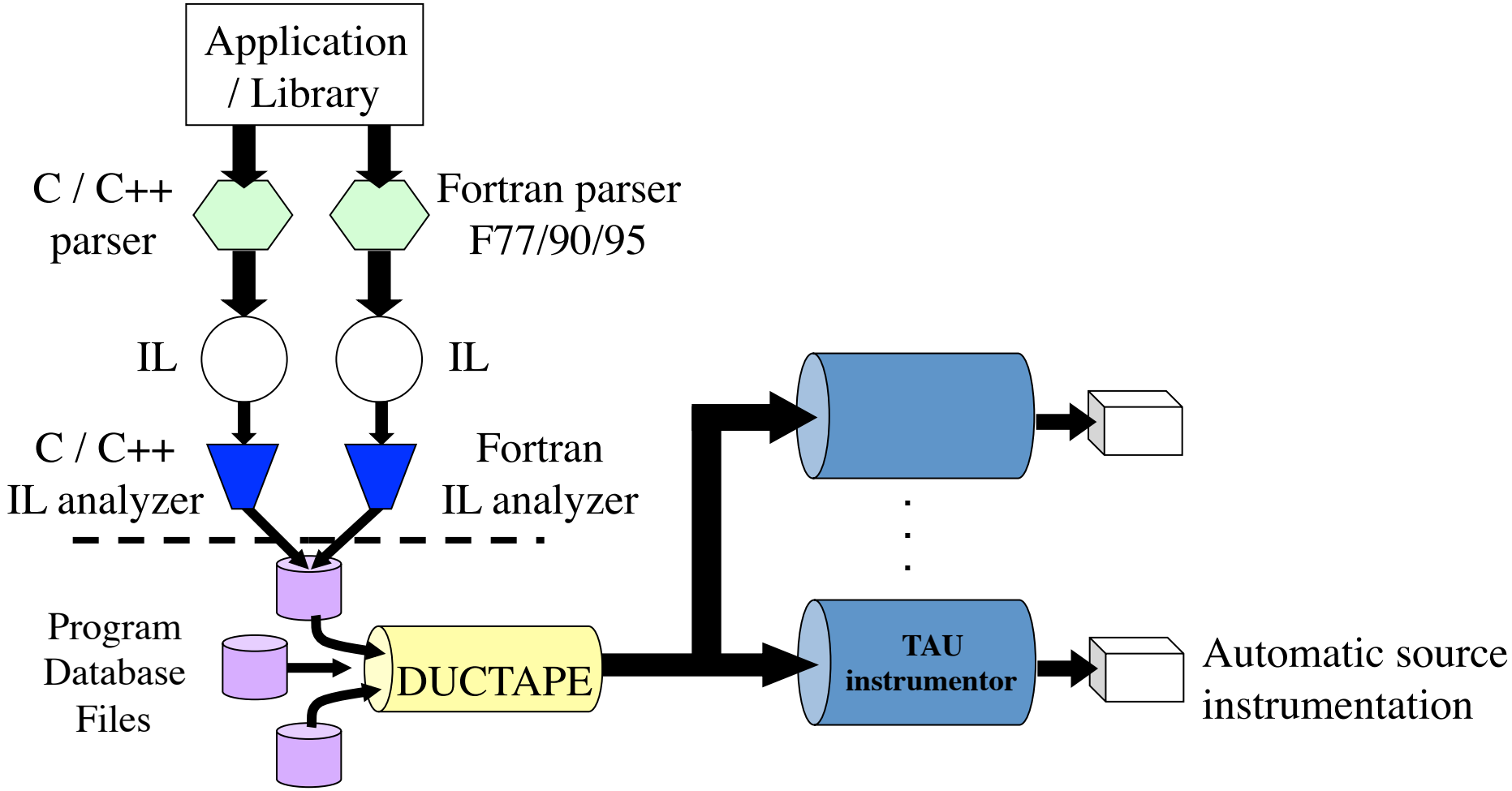
Indirect Performance Observation

- Program code instrumentation is not used
- Performance is observed indirectly
  - Execution is interrupted
    - can be triggered by different events
  - Execution state is queried (sampled)
    - different performance data measured
  - *Event-based sampling* (EBS)
- Performance attribution is inferred
  - Determined by execution context (state)
  - Observation resolution determined by interrupt period
  - Performance data associated with context for period

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

Program Database Toolkit (PDT)



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
```

Interval and Atomic Events

```
xterm
NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.187        1,105           1          44    1105659 int main(int, char **) C
 93.2        1,030        1,030           1           0    1030654 MPI_Init()
  5.9        0.879           65          40         320       1637 void func(int, int) C
  4.6           51           51          40           0       1277 MPI_Barrier()
  1.2           13           13         120           0        111 MPI_Recv()
  0.8            9            9           1           0       9328 MPI_Finalize()
  0.0        0.137        0.137         120           0          1 MPI_Send()
  0.0        0.086        0.086          40           0          2 MPI_Bcast()
  0.0        0.002        0.002           1           0          2 MPI_Comm_size()
  0.0        0.001        0.001           1           0          1 MPI_Comm_rank()
---------------------------------------------------------------------------------------

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
       365  5.138E+04      44.39   3.09E+04  1.234E+04  Heap Memory Used (KB) : Entry
       365  5.138E+04       2064  3.115E+04   1.21E+04  Heap Memory Used (KB) : Exit
        40         40         40         40          0  Message size for broadcast
---------------------------------------------------------------------------------------
                                                                    27,1          1%
```

Interval event e.g., routines (start/stop)

Atomic events (trigger with value)

% setenv TAU_CALLPATH_DEPTH 0
% setenv TAU_TRACK_HEAP 1

Atomic Events and Context Events

```
xterm
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.253        1,106           1          44    1106701 int main(int, char **) C
 93.2        1,031        1,031           1           0    1031311 MPI_Init()
  6.0            1           66          40         320       1650 void func(int, int) C
  5.7           63           63          40           0       1588 MPI_Barrier()
  0.8            9            9           1           0       9119 MPI_Finalize()
  0.1            1            1         120           0         10 MPI_Recv()
  0.0        0.141        0.141         120           0          1 MPI_Send()
  0.0        0.085        0.085          40           0          2 MPI_Bcast()
  0.0        0.001        0.001           1           0          1 MPI_Comm_size()
  0.0            0            0           1           0          0 MPI_Comm_rank()
---------------------------------------------------------------------------------------

USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue  MeanValue  Std. Dev.  Event Name
---------------------------------------------------------------------------------------
        40         40         40         40          0  Message size for broadcast
       365  5.139E+04      44.39  3.091E+04  1.234E+04  Heap Memory Used (KB) : Entry
        40  5.139E+04       3097  3.114E+04  1.227E+04  Heap Memory Used (KB) : Entry : MPI_Barrier()
        40  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : MPI_Bcast()
         1       2067       2067       2067          0  Heap Memory Used (KB) : Entry : MPI_Comm_rank()
         1       2066       2066       2066          0  Heap Memory Used (KB) : Entry : MPI_Comm_size()
         1  5.139E+04  5.139E+04  5.139E+04  0.0006905  Heap Memory Used (KB) : Entry : MPI_Finalize()
         1      57.56      57.56      57.56          0  Heap Memory Used (KB) : Entry : MPI_Init()
       120  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : MPI_Recv()
       120  5.139E+04  1.129E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : MPI_Send()
         1      44.39      44.39      44.39          0  Heap Memory Used (KB) : Entry : int main(int, char **) C
        40  5.036E+04       2068  3.011E+04  1.227E+04  Heap Memory Used (KB) : Entry : void func(int, int) C
                                                                                    4,9           1%
```

Atomic event

Context event = atomic event + executing context

% setenv TAU_CALLPATH_DEPTH 1

% setenv TAU_TRACK_HEAP 1

ParaProf – Callpath Profile

n,c,t, 0,0,0 - callpath-all/scaling/flash/taudata/disk2/mnt/

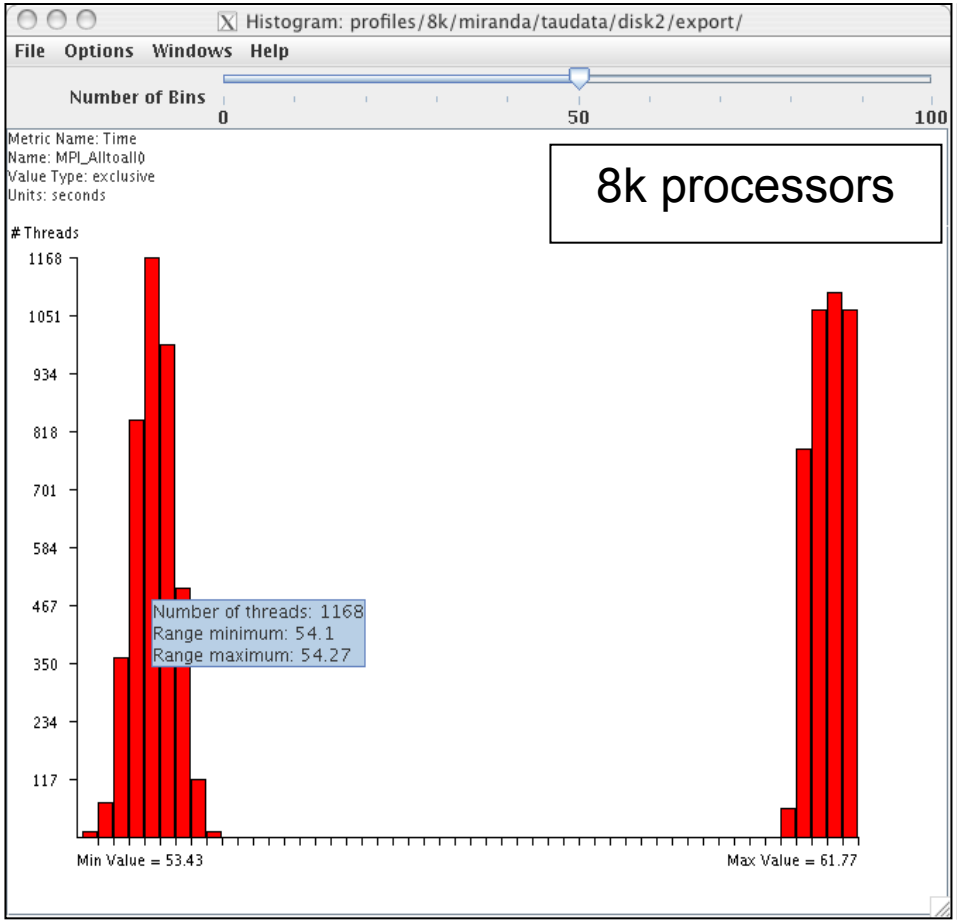
File Options Windows Help

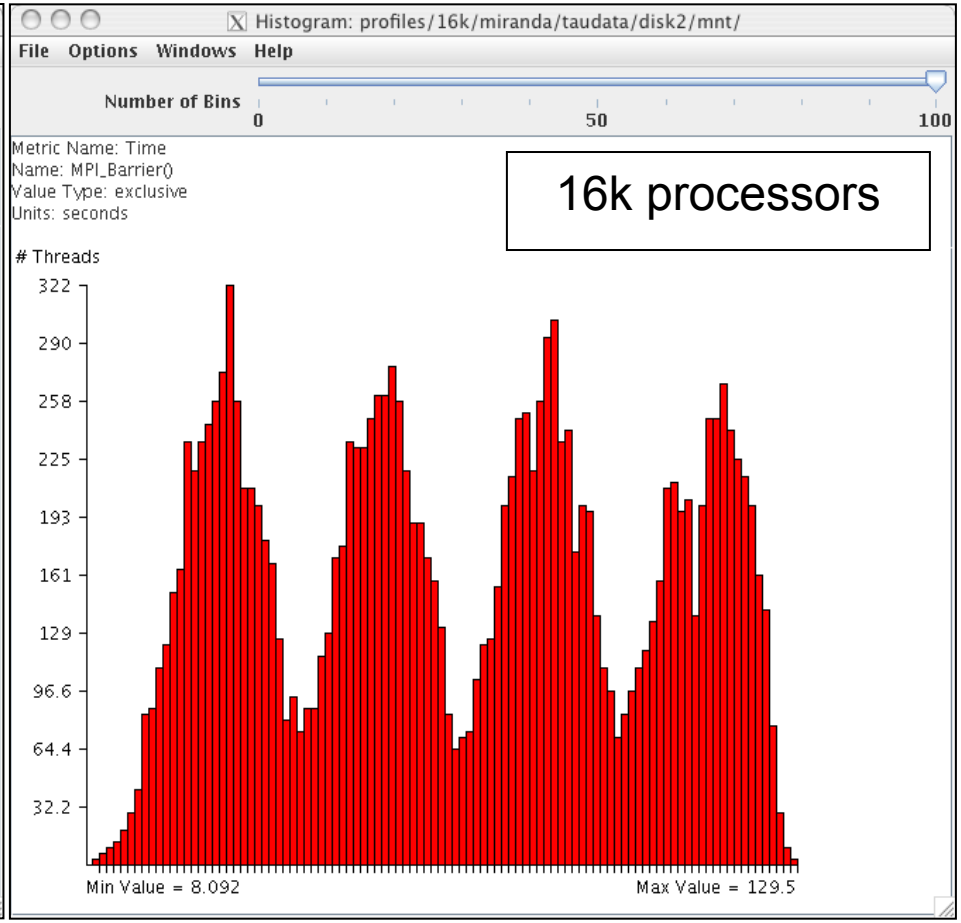
Metric Name: Time
Value Type: exclusive

26.474%	MODULEHYDROSWEEP::HYDRO_SWEEP
26.474%	FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP
24.556%	MODULEHYDRO_1D::HYDRO_1D
24.556%	FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MODULEHYDRO_1D::HYDRO_1D
14.351%	MODULEINTRFC::INTRFC
14.351%	FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MODULEHYDRO_1D::HYDRO_1D => MODULEINTRFC::INT
4.501%	MODULEEOS3D::EOS3D
4.427%	MPI_Ssend()
3.678%	FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MODULEEOS3D::EOS3D
3.536%	MPI_Allreduce()
2.727%	MPI_Waitall()
2.242%	MODULEUPDATE_SOLN::UPDATE_SOLN
2.242%	FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MODULEUPDATE_SOLN::UPDATE_SOLN
2.059%	AMR_GUARDCELL_CC_SRL
1.703%	FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_GUARDCELL => AMR_GUARDCELL_SRL => AMR_
1.56%	FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_GUARDCELL => AMR_GUARDCELL_SRL => AMR_
1.406%	FLASH => EVOLVE => MESH_UPDATE_GRID_REFINEMENT => MESH_REFINE_DEREFINE => AMR_REFINE_DEREFINE => AMR_MORTON_ORDER =>
1.361%	FLASH => TIMESTEP => MPI_Allreduce()
1.319%	AMR_RESTRICT_UNK_FUN
1.272%	AMR_PROLONG_GEN_UNK_FUN
1.093%	FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_GUARDCELL => AMR_GUARDCELL_C_TO_F => A
1.077%	ABUNDANCE_RESTRICT
1.077%	FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => ABUNDANCE_RESTRICT
1.064%	DBASETREE::DBASENEIGHBORBLOCKLIST
1%	FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_GUARDCELL => AMR_RESTRICT => AMR_RESTRI
0.987%	FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_FLUX_CONSERVE => AMR_FLUX_CONSERVE_UDT
0.96%	FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_GUARDCELL => AMR_GUARDCELL_C_TO_F => A
0.916%	MPI_Barrier()
0.807%	FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_GUARDCELL => TOT_BND => DBASETREE::DBAS
0.806%	AMR_PROLONG_UNK_FUN
0.735%	AMR_DIAGONAL_PATCH
0.699%	DIFFUSE
0.699%	FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => DIFFUSE
0.671%	AMR_RESTRICT_RED
0.671%	FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_FLUX_CONSERVE => AMR_FLUX_CONSERVE_UDT
0.657%	FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_GUARDCELL => AMR_GUARDCELL_SRL => AMR_
0.638%	FLASH => EVOLVE => MESH_UPDATE_GRID_REFINEMENT => MARK_GRID_REFINEMENT => MPI_Barrier()
0.61%	FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_GUARDCELL => AMR_GUARDCELL_C_TO_F => A
0.556%	FLASH => EVOLVE => HYDRO::HYDRO_3D => MODULEHYDROSWEEP::HYDRO_SWEEP => MESH_GUARDCELL => AMR_GUARDCELL_C_TO_F => A
0.508%	TOT_BND
0.454%	FLASH => EVOLVE => MESH_UPDATE_GRID_REFINEMENT => MARK_GRID_REFINEMENT => MODULEEOS3D::EOS3D

Flash
- thermonuclear flashes
- Fortran + MPI
- Argonne

ParaProf – Scalable Histogram

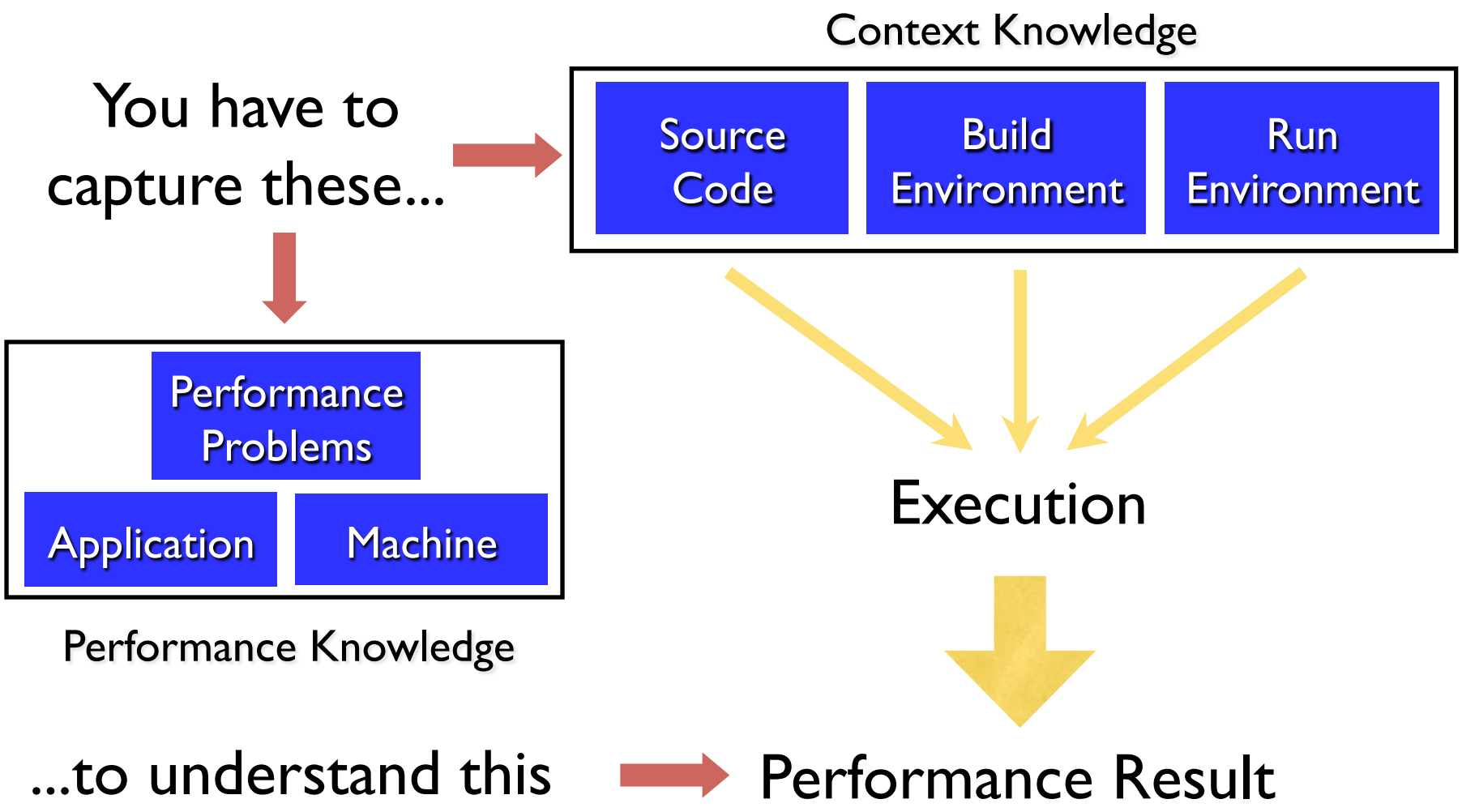




How to explain performance?

- 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



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

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

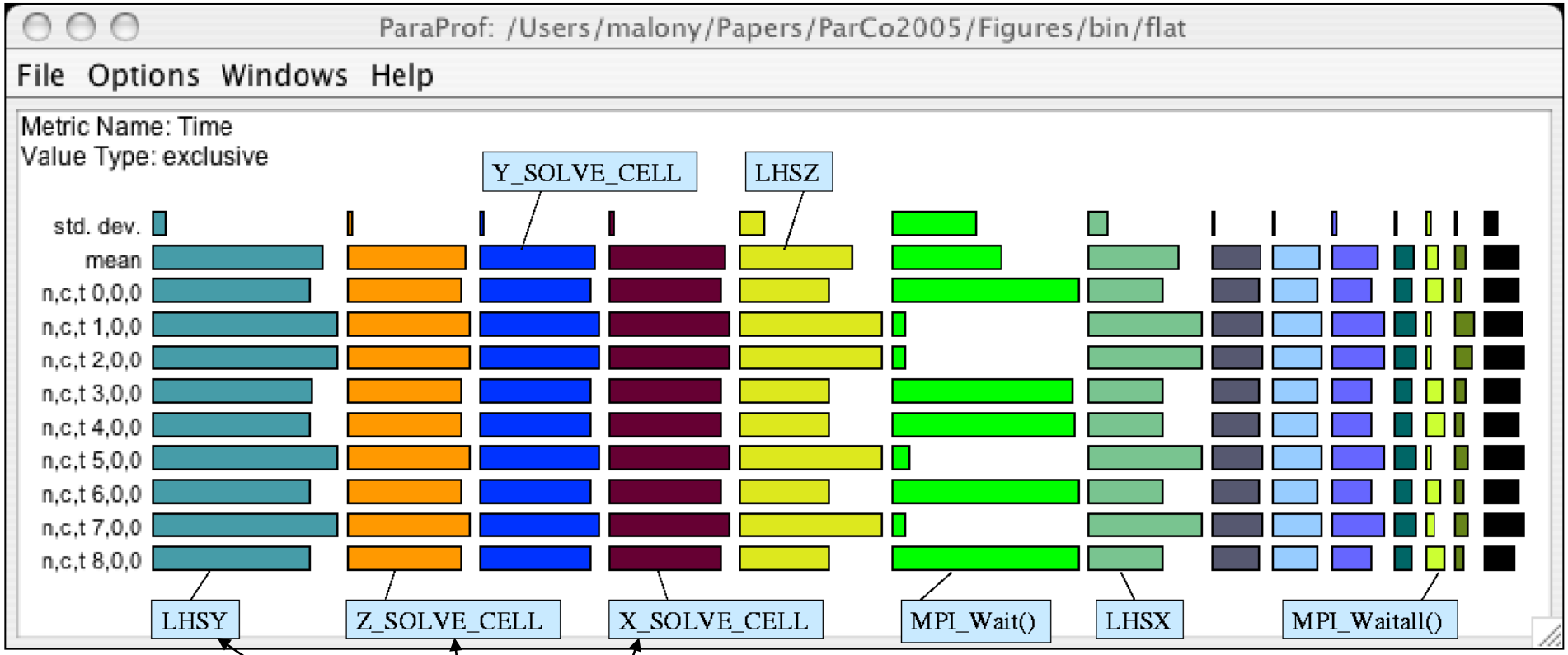
Performance Data Mining / Analytics

- 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

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

NAS BT – Flat Profile



Application routine names reflect phase semantics

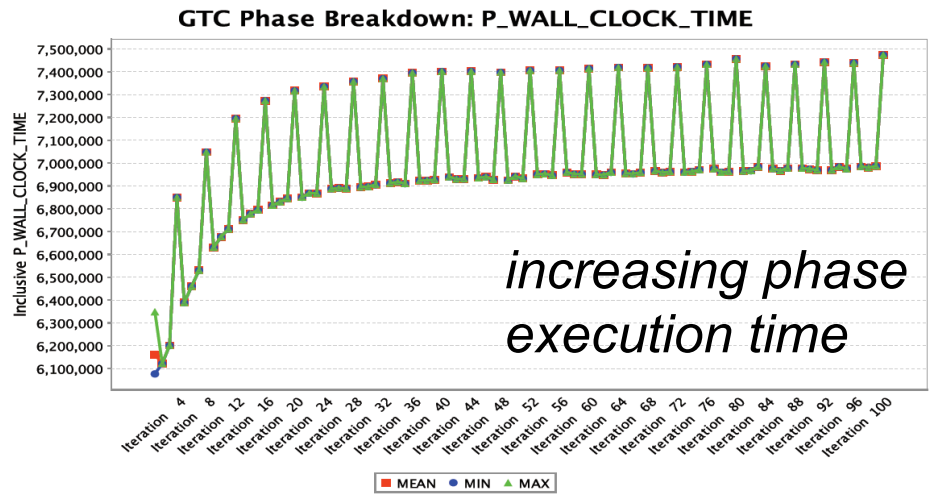
How is MPI_Wait() distributed relative to solver direction?

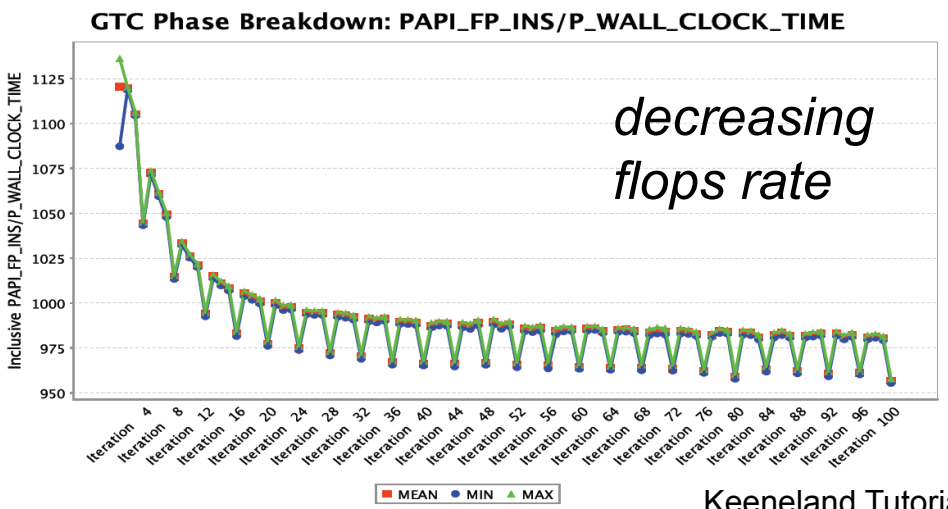
NAS BT – Phase Profile

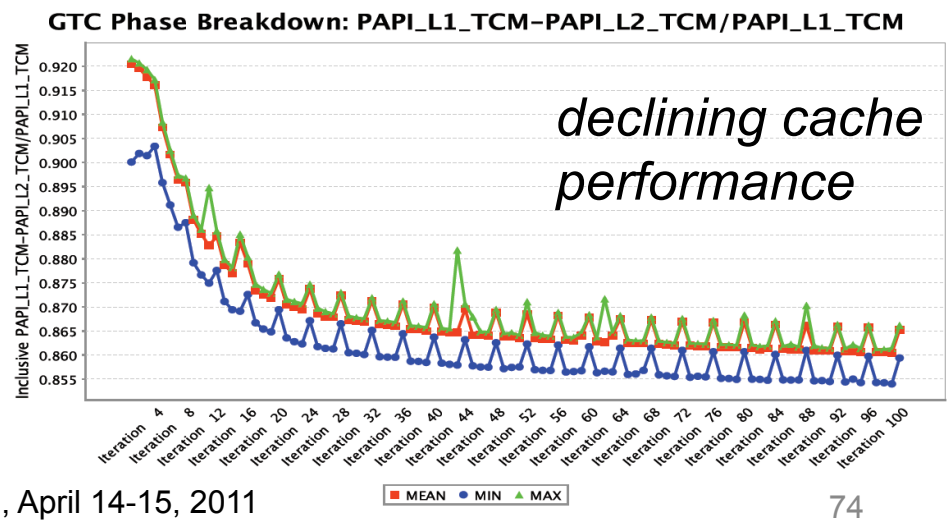
Main phase shows nested phases and immediate events

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

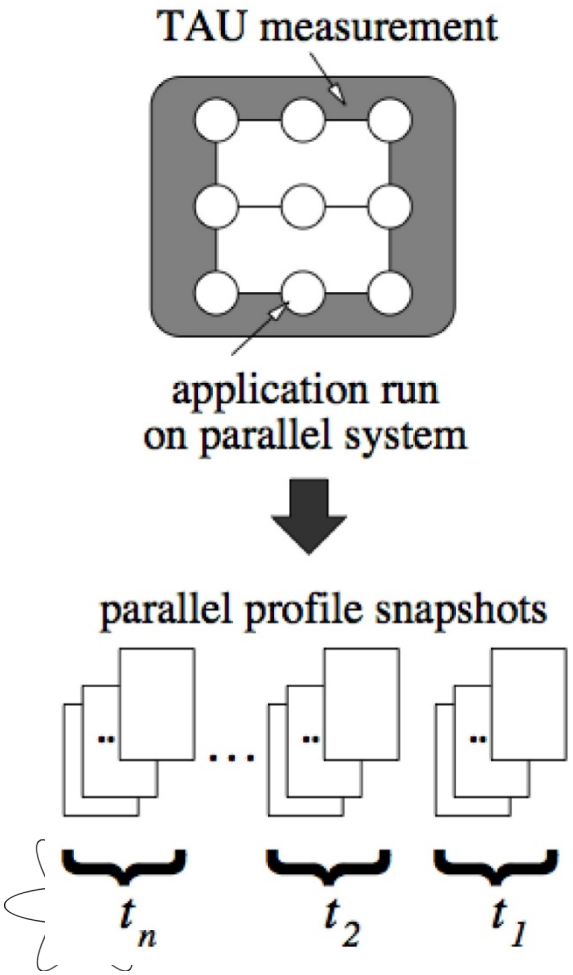


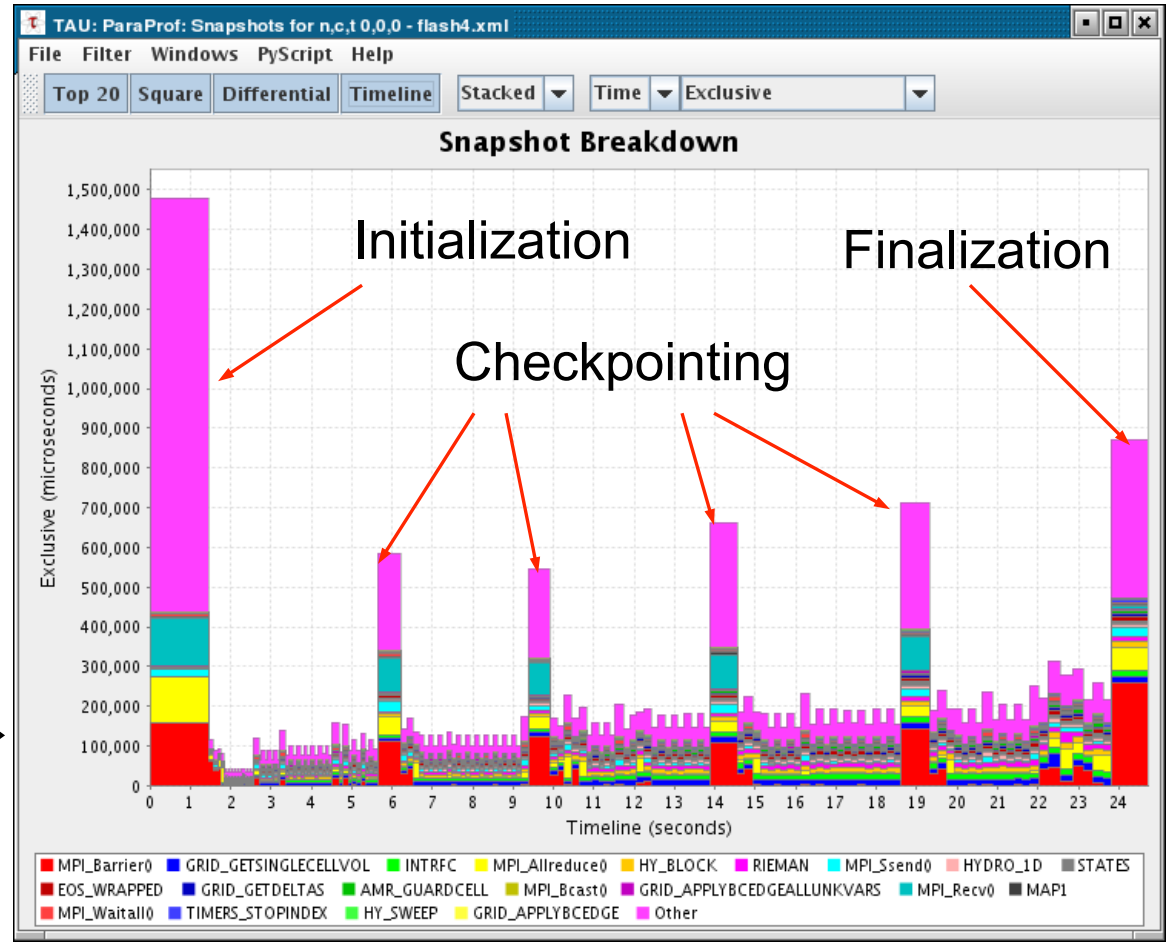




Profile Snapshots in ParaProf

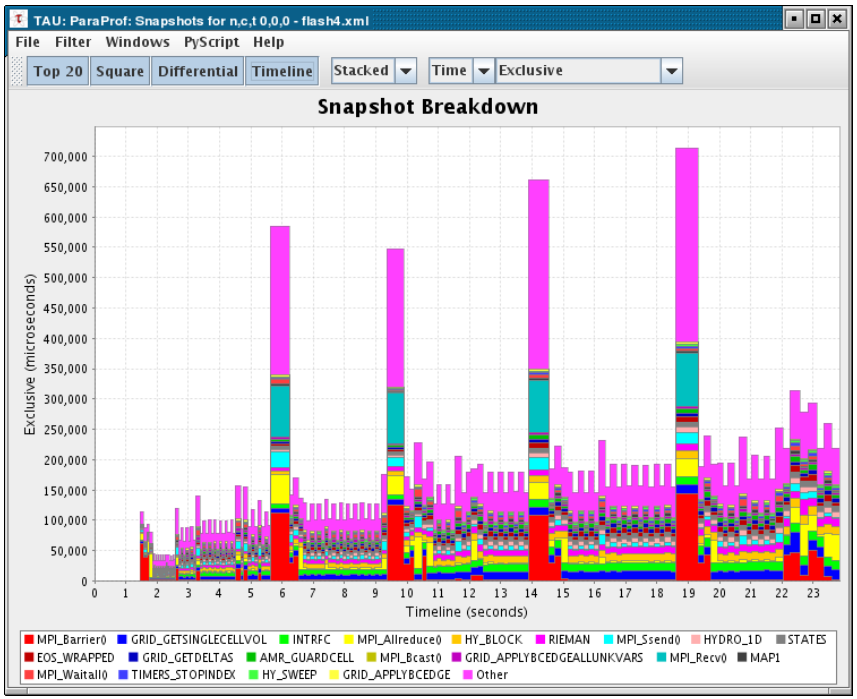
- Profile snapshots are parallel profiles recorded at runtime
- Shows performance profile dynamics (all types allowed)

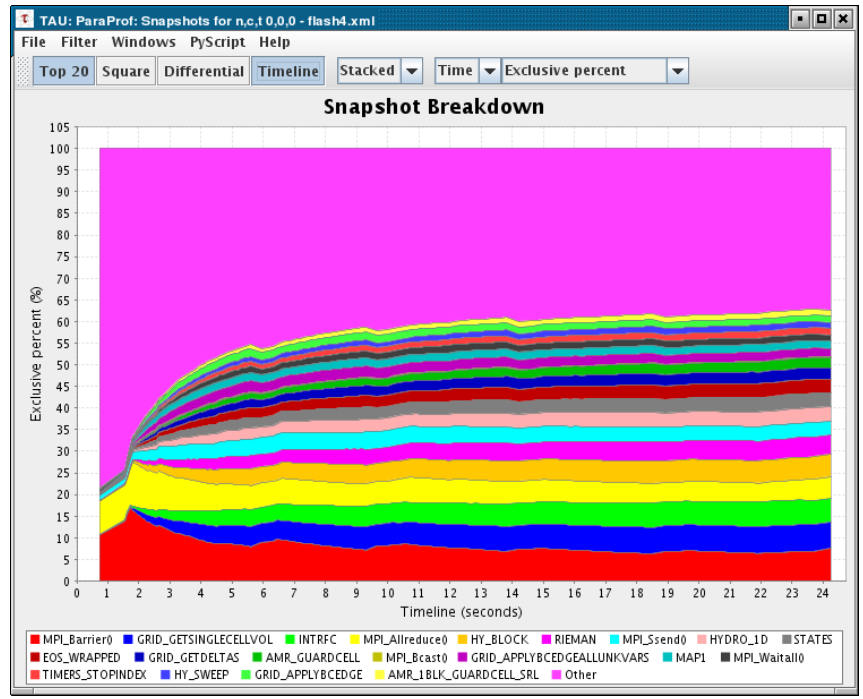




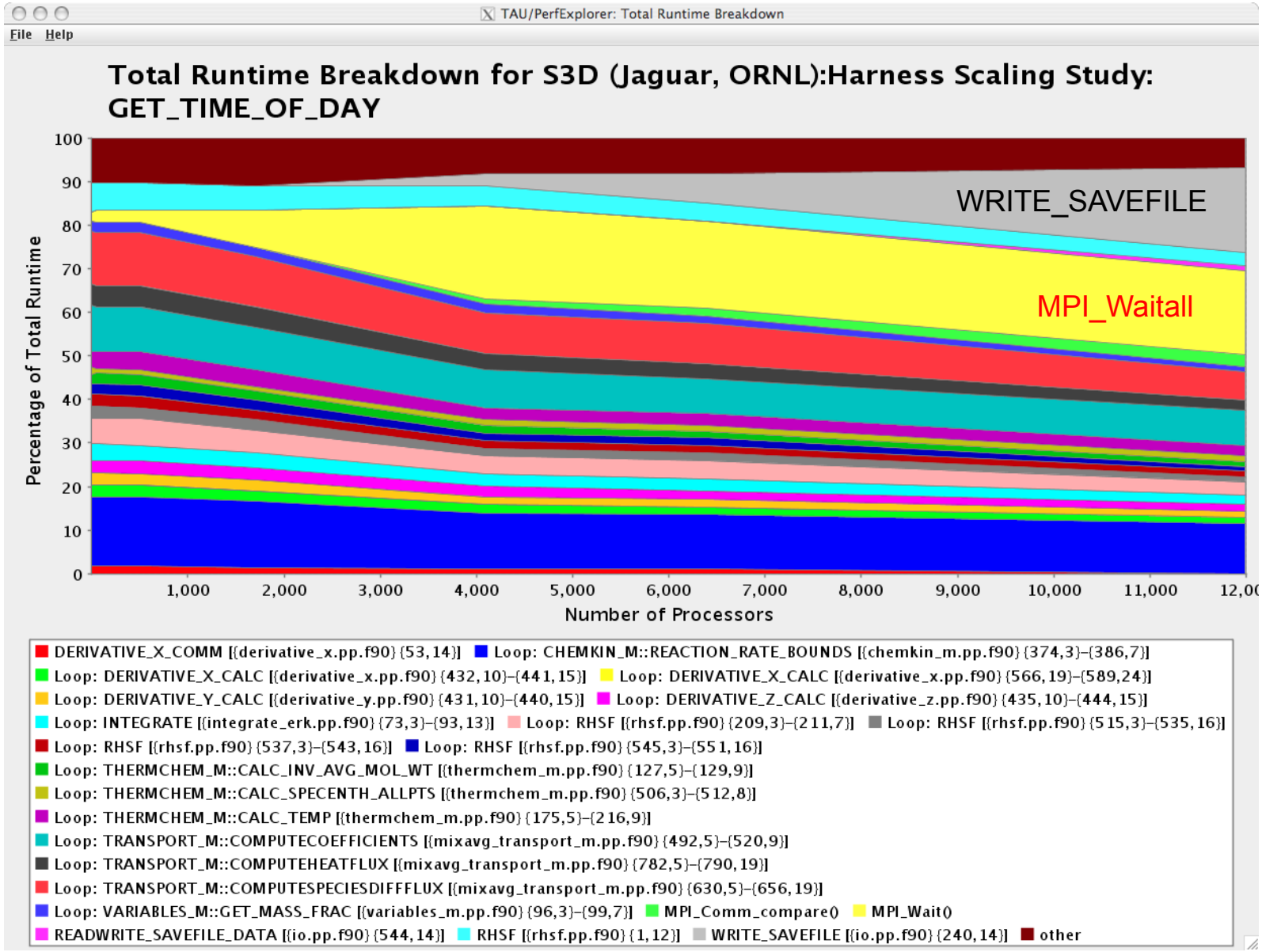
Profile Snapshot Views

- Only show main loop
- Percentage breakdown



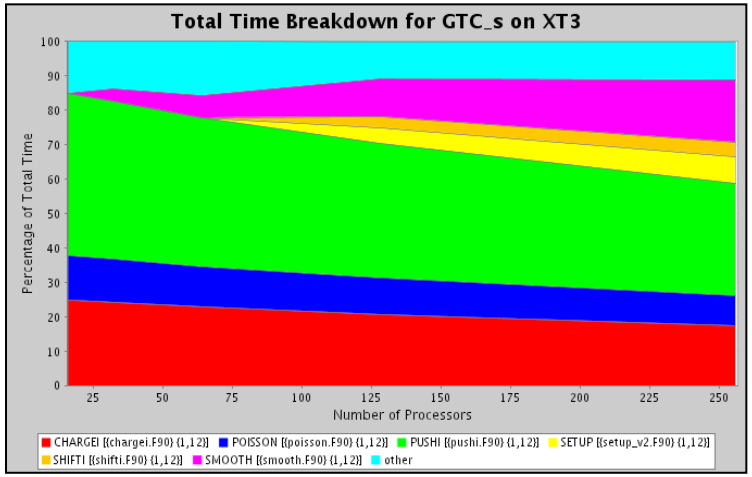


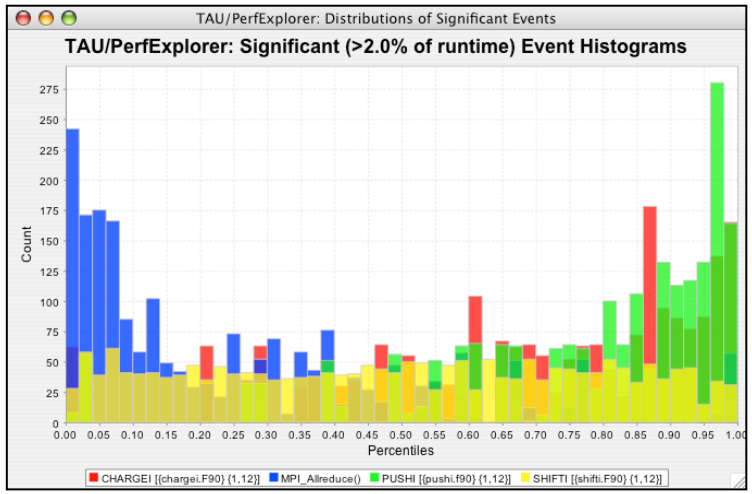
PerfExplorer – Runtime Breakdown



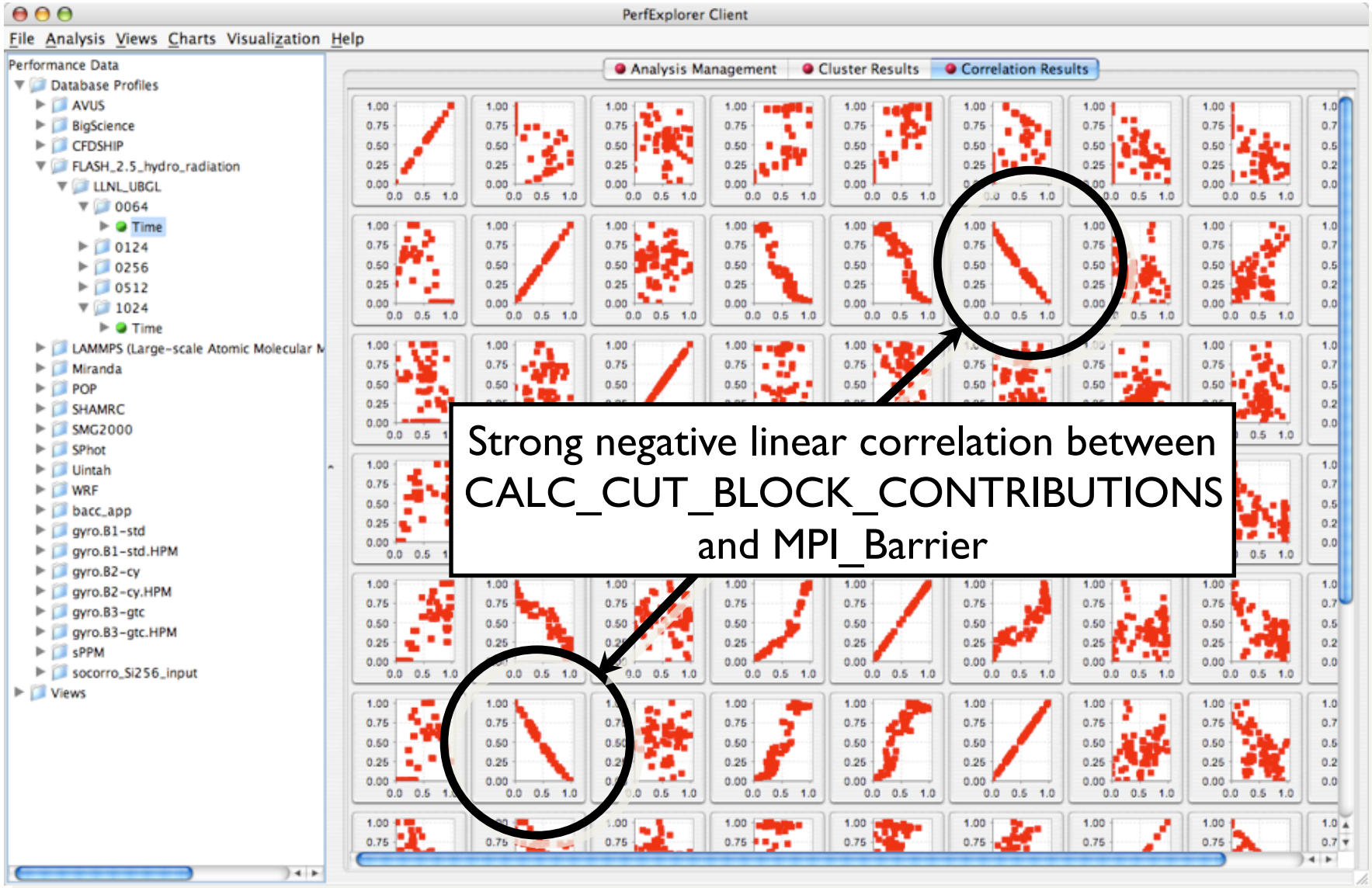
PerfExplorer – Relative Comparisons

- 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 runtime
- Relative efficiency per phase
- Relative speedup per phase
- Distribution visualizations



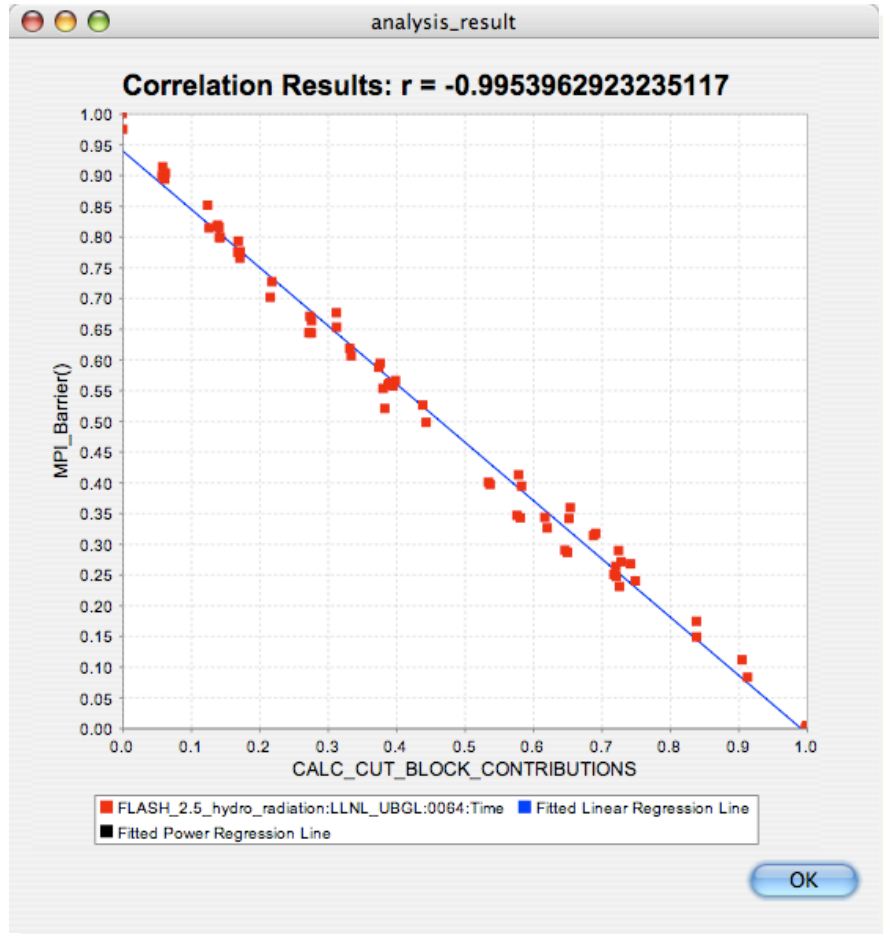


PerfExplorer – Correlation Analysis

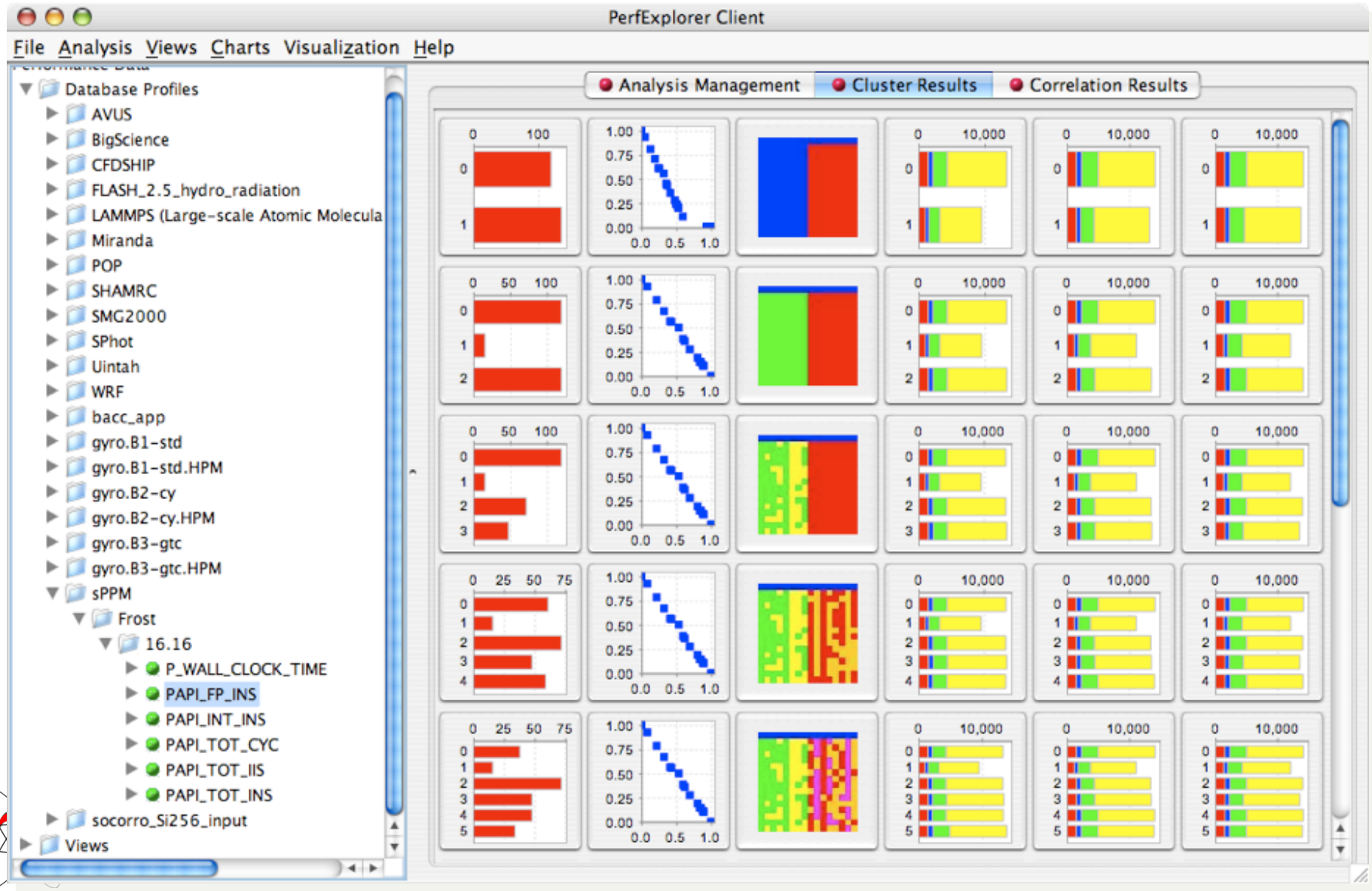


PerfExplorer – Correlation Analysis

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

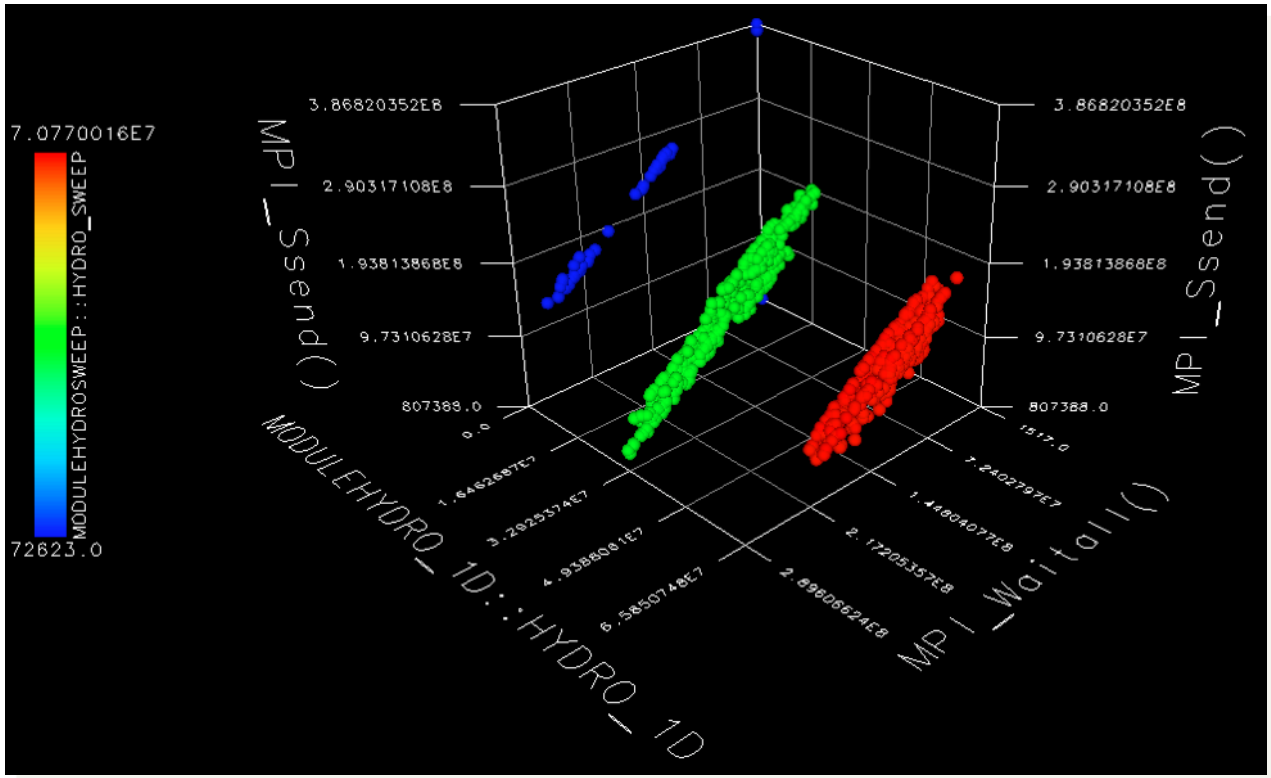


PerfExplorer – Cluster Analysis



PerfExplorer – Cluster Analysis

- Four significant events automatically selected
- Clusters and correlations are visible



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)

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
```

Rewriting Binaries

- `tau_run`
- Support for both static and dynamic executables
- Specify the list of routines to include or exclude from instrumentation
- Specify the TAU measurement library to be injected
- Simplify the usage of TAU:
  - To instrument:
    ```
    % tau_run a.out -o a.inst
    ```
  - To perform measurements and execute the application:
    ```
    % mpirun -np 4 ./a.inst
    ```
  - To analyze the data:
    ```
    % paraprof
    ```

Library Interposition/Wrapping

- `tau_exec` and `tau_wrap`
- TAU provides a wealth of options to measure the performance of an application
- Need to simplify TAU usage to easily evaluate performance properties, including I/O, memory, and communication
- Designed a new tool (`tau_exec`) that leverages runtime instrumentation by pre-loading measurement libraries
- Works on dynamic executables (default under Linux)
- Substitutes I/O, MPI, and memory allocation/deallocation routines with instrumented calls
  - Interval events (e.g., time spent in write())
  - Atomic events (e.g., how much memory was allocated)
- Measure I/O and memory usage

TAU Execution Command

- Uninstrumented execution
  - `% mpirun -np 256 ./a.out`
- Track MPI performance
  - `% mpirun -np 256  tau_exec ./a.out`
- Track I/O and MPI performance (MPI enabled by 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`
- Track GPGPU operations
  - `% mpirun -np 256 tau_exec -cuda ./a.out`

Library Wrapping

- `tau_wrap -r <library.so>`
- 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
  - Library loads the original library using the dlopen() call
- Application is instrumented
- Add the `-I<wrapper>` directory to the command line
- C pre-processor will substitute our headers
  - Redirects references at routine callsite to a wrapper call
  - Wrapper internally calls the original
  - Wrapper has TAU measurement code

HDF5 Library Wrapping

```
[sameer@zorak]$ cparse hdf5.h; tau_wrap hdf5.h.pdb hdf5.h -o hdf5.inst.c -f
select.tau -g hdf5 -r libhdf5.so; cd wrapper; make

Usage : tau_wrap <pdbfile> <sourcefile> [-o <outputfile>] [-r runtimelibname] [-g
groupname] [-i headerfile] [-c|-c++|-fortran] [-f <instr_req_file> ]
```

- instrumented wrapper library source (hdf5.inst.c)
- instrumentation specification file (select.tau)
- group (hdf5)
- tau_exec loads libhdf5_wrap.so shared library using LD_PRELOAD
- creates the wrapper/ directory

```
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()
  0.0            0            0           1           0          0 herr_t H5Tclose()
```

-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> ]
```