

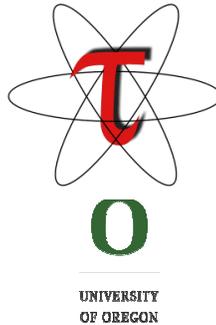
TAU Performance System

Sameer Shende

University of Oregon

sameer@cs.uoregon.edu

ACTS Workshop, LBL, Aug 24, 2007



Acknowledgements

- Dr. Allen D. Malony, Professor
- Alan Morris, Senior software engineer
- Wyatt Spear, Software engineer
- Scott Biersdorff, Software engineer
- Kevin Huck, Ph.D. student
- Aroon Nataraj, Ph.D. student
- Brad Davidson, Systems administrator

Outline



- ❑ Overview of features
- ❑ Instrumentation
- ❑ Measurement
- ❑ Analysis tools
 - Parallel profile analysis (ParaProf)
 - Performance data management (PerfDMF)
 - Performance data mining (PerfExplorer)
- ❑ Application examples

Performance Evaluation



- ❑ Profiling
 - Presents summary statistics of performance metrics
 - number of times a routine was invoked
 - exclusive, inclusive time/hpm counts spent executing it
 - number of instrumented child routines invoked, etc.
 - structure of invocations (calltrees/callgraphs)
 - memory, message communication sizes also tracked
- ❑ Tracing
 - Presents when and where events took place along a global timeline
 - timestamped log of events
 - message communication events (sends/receives) are tracked
 - shows when and where messages were sent
 - large volume of performance data generated leads to more perturbation in the program

Definitions – Profiling



□ Profiling

- Recording of summary information during execution
 - inclusive, exclusive time, # calls, hardware statistics, ...
- Reflects performance behavior of program entities
 - functions, loops, basic blocks
 - user-defined “semantic” entities
- Very good for low-cost performance assessment
- Helps to expose performance bottlenecks and hotspots
- Implemented through
 - **sampling**: periodic OS interrupts or hardware counter traps
 - **instrumentation**: direct insertion of measurement code

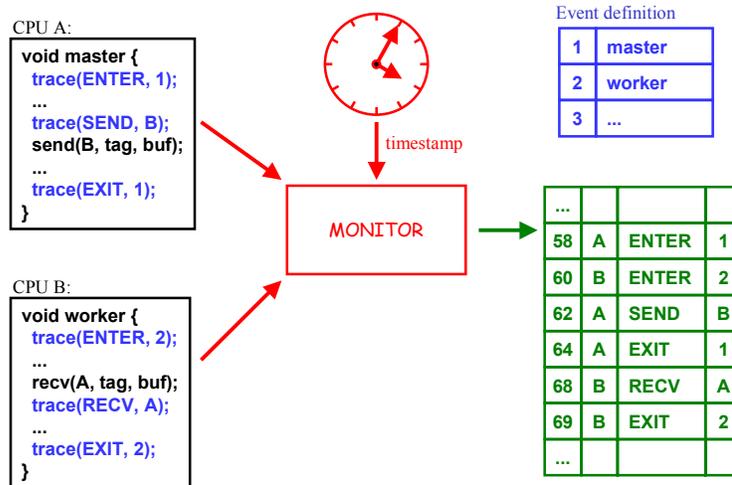
Definitions – Tracing



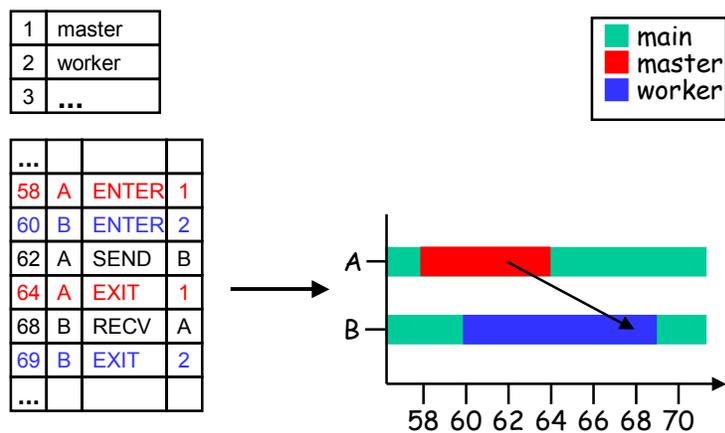
□ Tracing

- Recording of information about significant points (**events**) during program execution
 - entering/exiting code region (function, loop, block, ...)
 - thread/process interactions (e.g., send/receive message)
- Save information in **event record**
 - timestamp
 - CPU identifier, thread identifier
 - Event type and event-specific information
- **Event trace** is a time-sequenced stream of event records
- Can be used to reconstruct dynamic program behavior
- Typically requires code instrumentation

Event Tracing: *Instrumentation, Monitor, Trace*



Event Tracing: "Timeline" Visualization



Steps of Performance Evaluation



- ❑ Collect basic routine-level timing profile to determine where most time is being spent
- ❑ Collect routine-level hardware counter data to determine types of performance problems
- ❑ Collect callpath profiles to determine sequence of events causing performance problems
- ❑ Conduct finer-grained profiling and/or tracing to pinpoint performance bottlenecks
 - Loop-level profiling with hardware counters
 - Tracing of communication operations

TAU Performance System

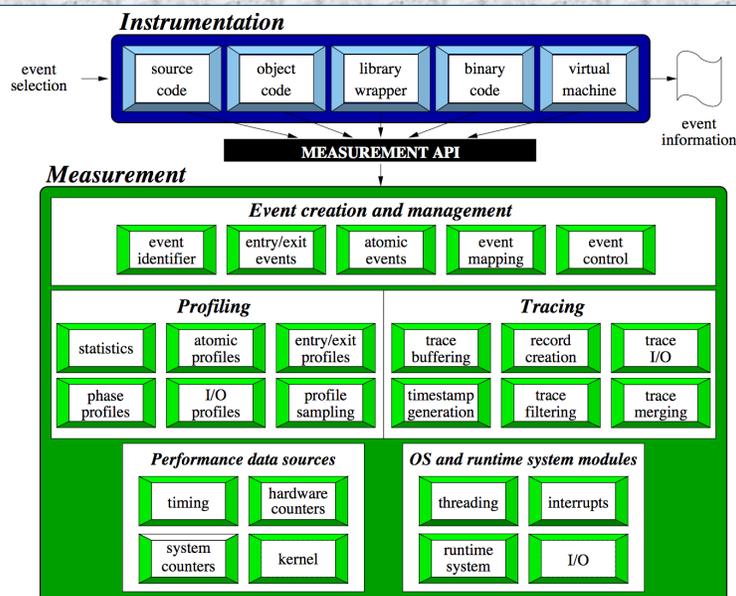


- ❑ **Tuning and Analysis Utilities** (15+ year project effort)
- ❑ **Performance system framework for HPC systems**
 - Integrated, scalable, flexible, and parallel
- ❑ Targets a general complex system computation model
 - *Entities*: nodes / contexts / threads
 - *Multi-level*: system / software / parallelism
 - Measurement and analysis abstraction
- ❑ **Integrated toolkit for performance problem solving**
 - Instrumentation, measurement, analysis, and visualization
 - Portable performance profiling and tracing facility
 - Performance data management and data mining
- ❑ **Partners**: LLNL, ANL, LANL, Research Center Jülich

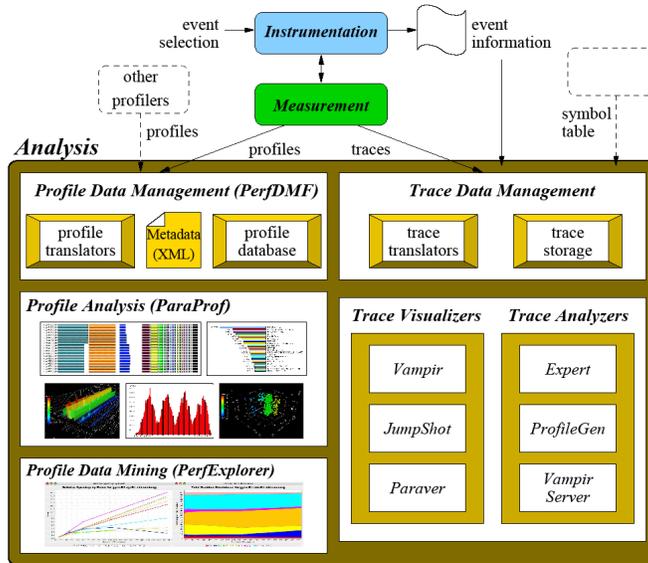
TAU Parallel Performance System Goals

- ❑ **Portable (open source) parallel performance system**
 - Computer system architectures and operating systems
 - Different programming languages and compilers
- ❑ Multi-level, multi-language performance instrumentation
- ❑ **Flexible and configurable performance measurement**
- ❑ Support for multiple parallel programming paradigms
 - Multi-threading, message passing, mixed-mode, hybrid, object oriented (generic), component-based
- ❑ Support for performance mapping
- ❑ Integration of leading performance technology
- ❑ **Scalable (very large) parallel performance analysis**

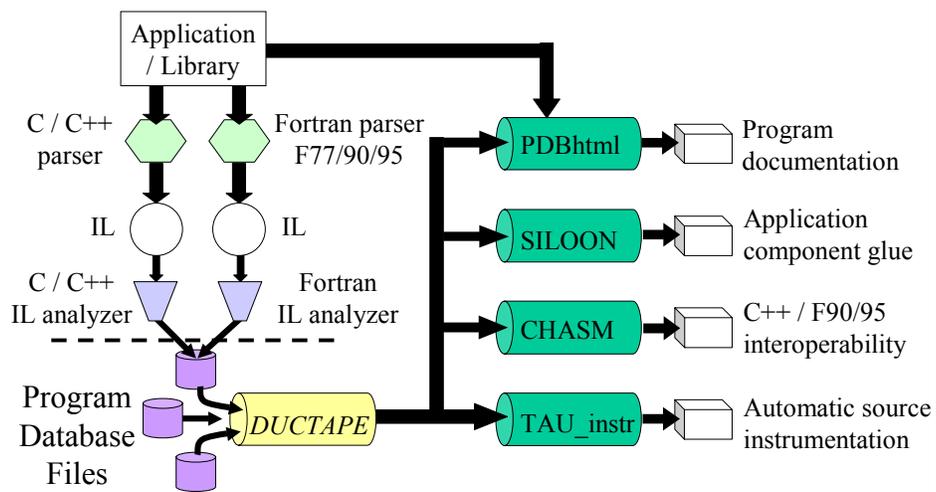
TAU Performance System Architecture



TAU Performance System Architecture



Program Database Toolkit (PDT)





- ❑ **Performance Application Programming Interface**
 - The purpose of the PAPI project is to design, standardize and implement a portable and efficient API to access the hardware performance monitor counters found on most modern microprocessors.
- ❑ Parallel Tools Consortium project started in 1998
- ❑ Developed by University of Tennessee, Knoxville
- ❑ <http://icl.cs.utk.edu/papi/>

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)
- ❑ **Object code**
 - Pre-instrumented libraries (e.g., MPI using *PMPI*)
 - Statically-linked and dynamically-linked
- ❑ **Executable code**
 - Dynamic instrumentation (pre-execution) (*DynInstAPI*)
 - Virtual machine instrumentation (e.g., Java using *JVMPI*)
- ❑ **TAU_COMPILER** to automate instrumentation process

Using TAU: A brief Introduction



- To instrument source code using PDT
 - Choose an appropriate TAU stub makefile in <arch>/lib:
% **setenv TAU_MAKEFILE**
 /opt/apps/tau/tau_latest/x86_64/lib/Makefile.tau-icpc-mpi-pdt
 - % **setenv TAU_OPTIONS '-optVerbose ...'** (see **tau_compiler.sh**)

And use **tau_f90.sh**, **tau_cxx.sh** or **tau_cc.sh** as Fortran, C++ or C compilers:

 - % **mpif90 foo.f90**

changes to

 - % **tau_f90.sh foo.f90**

- Execute application and analyze performance data:
 - % **pprof** (for text based profile display)
 - % **paraprof** (for GUI)

TAU Measurement System Configuration

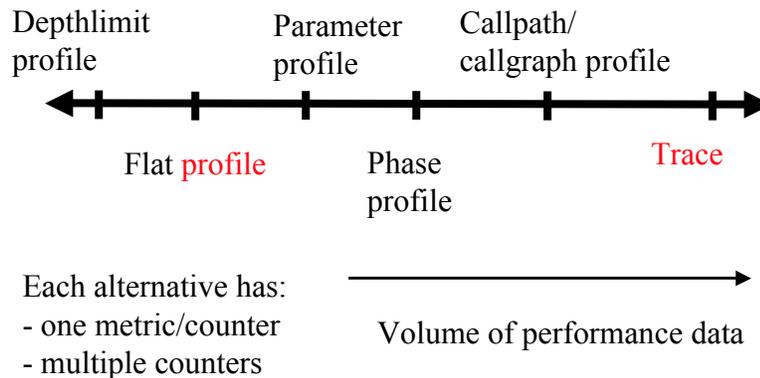


- **configure [OPTIONS]**
 - {-c++=<CC>, -cc=<cc>} Specify C++ and C compilers
 - pdt=<dir> Specify location of PDT
 - opari=<dir> Specify location of Opari OpenMP tool
 - papi=<dir> Specify location of PAPI
 - vampirtrace=<dir> Specify location of VampirTrace
 - mpi[inc/lib]=<dir> Specify MPI library 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
 - PROFILEPARAM Generate parameter 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

Performance Evaluation Alternatives



TAU Measurement Configuration – Examples



- ❑ `configure -pdt=<dir> -c++=pathCC -cc=pathcc -fortran=pathscale -mpiinc=/usr/common/ug/mvapich/pathscale/mvapich-0.9.5-mlx1.0.3/include -mpilib=/usr/common/ug/mvapich/pathscale/mvapich-0.9.5-mlx1.0.3/lib -mpilibary='-lmpich -L/usr/local/ibgd/driver/infinihost/lib64 -lvapi'`
 - on Jacquard with PDT, MPI for x86_64 and Pathscale compilers
- ❑ `./configure -papi=/opt/apps/papi-3.5.0 -MULTIPLECOUNTERS [other options]; make clean install`
 - Use PAPI counters (one or more) with C/C++/F90 automatic instrumentation for Linux. Also instrument the MPI library.
- ❑ Typically configure multiple measurement libraries
 - `.all_configs`, `.last_config` files contain all and last configuration
 - `tau_validate --html --build x86_64 >& results.html`
 - `./upgradetau /path/to/old/tau-2.16`
- ❑ Each configuration creates a unique `<arch>/lib/Makefile.tau<options> stub` makefile. It corresponds to the configuration options used. e.g.,
 - `/opt/apps/tau/tau_latest/x86_64/lib/Makefile.tau-mpi-pdt-pgi`
 - `/opt/apps/tau/tau_latest/x86_64/lib/Makefile.tau-multiplecounters-icpc-mpi-papi-pdt`

TAU Measurement Configuration – Examples



```
% cd /opt/apps/tau/tau_latest/x86_64/lib; ls Makefile.*
Makefile.tau-icpc-pdt
Makefile.tau-icpc-mpi-pdt
Makefile.tau-icpc-callpath-mpi-pdt
Makefile.tau-icpc-mpi-pdt-trace
Makefile.tau-icpc-mpi-compensate-pdt
Makefile.tau-multiplecounters-icpc-mpi-papi-pdt
Makefile.tau-multiplecounters-icpc-mpi-papi-pdt-trace
Makefile.tau-icpc-mpi-papi-pdt-epilog-trace
Makefile.tau-icpc-pdt...
```

- ❑ For an MPI+F90 application, you may want to start with:

```
Makefile.tau-icpc-mpi-pdt-pgi
  ○ Supports MPI instrumentation & PDT for automatic source instrumentation for Intel compilers
```

Configuration Parameters in Stub Makefiles

- Each TAU stub Makefile resides in `<tau>/<arch>/lib` directory
- 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
- Each stub makefile encapsulates the parameters that TAU was configured with
- It represents a specific instance of the TAU libraries. TAU scripts use stub makefiles to identify what performance measurements are to be performed.

Automatic Instrumentation

- We now provide compiler wrapper scripts
 - Simply replace `mpxlf90` with `tau_f90.sh`
 - Automatically instruments Fortran source code, links with TAU MPI Wrapper libraries.
- Use `tau_cc.sh` and `tau_cxx.sh` for C/C++

Before

```
CXX = mpCC
F90 = mpxlf90_r
CFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o ... fn.o

app: $(OBJS)
    $(CXX) $(LDFLAGS) $(OBJS) -o $@
    $(LIBS)
.cpp.o:
    $(CC) $(CFLAGS) -c $<
```

After

```
CXX = tau_cxx.sh
F90 = tau_f90.sh
CFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o ... fn.o

app: $(OBJS)
    $(CXX) $(LDFLAGS) $(OBJS) -o $@
    $(LIBS)
.cpp.o:
    $(CC) $(CFLAGS) -c $<
```

TAU_COMPILER Commandline Options

- See `<taudir>/<arch>/bin/tau_compiler.sh -help`
- **Compilation:**
 - % `mpxlf90 -c foo.f90`
 - Changes to
 - % `f95parse foo.f90 $(OPT1)`
 - % `tau_instrumentor foo.pdb foo.f90 -o foo.inst.f90 $(OPT2)`
 - % `mpxlf90 -c foo.f90 $(OPT3)`
- **Linking:**
 - % `mpxlf90 foo.o bar.o -o app`
 - Changes to
 - % `mpxlf90 foo.o bar.o -o app $(OPT4)`
- Where options OPT[1-4] default values may be overridden by the user:
 - F90 = `$(TAU_COMPILER) $(MYOPTIONS) mpxlf90`

TAU_COMPILER Options

- Optional parameters for `$(TAU_COMPILER)`: [`tau_compiler.sh -help`]
 - `-optVerbose` Turn on verbose debugging messages
 - `-optDetectMemoryLeaks` Turn on debugging memory allocations/de-allocations to track leaks
 - `-optPdtGnuFortranParser` Use `gfpars` (GNU) instead of `f95parse` (Cleanscape) for parsing Fortran source code
 - `-optKeepFiles` Does not remove intermediate `.pdb` and `.inst.*` files
 - `-optPreProcess` Preprocess Fortran sources before instrumentation
 - `-optTauSelectFile=""` Specify selective instrumentation file for `tau_instrumentor`
 - `-optLinking=""` Options passed to the linker. Typically `$(TAU_MPI_FLIBS) $(TAU_LIBS) $(TAU_CXXLIBS)`
 - `-optCompile=""` Options passed to the compiler. Typically `$(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)`
 - `-optPdtF95Opts=""` Add options for Fortran parser in PDT (`f95parse/gfpars`)
 - `-optPdtF95Reset=""` Reset options for Fortran parser in PDT (`f95parse/gfpars`)
 - `-optPdtCOpts=""` Options for C parser in PDT (`cpars`). Typically `$(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)`
 - `-optPdtCxxOpts=""` Options for C++ parser in PDT (`cxxpars`). Typically `$(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)`
 - ...

Overriding Default Options:TAU_COMPILER



```
% cat Makefile
F90 = tau_f90.sh
OBSJ = f1.o f2.o f3.o ...
LIBS = -Lappdir -lapplib1 -lapplib2 ...

app: $(OBSJ)
    $(F90) $(OBSJ) -o app $(LIBS)
.f90.o:
    $(F90) -c $<
% setenv TAU_OPTIONS '-optVerbose -optTauSelectFile=select.tau
    -optKeepFiles'
% setenv TAU_MAKEFILE <taudir>/x86_64/lib/Makefile.tau-mpi-pdt
```

Optimization of Program Instrumentation



- Need to eliminate instrumentation in frequently executing lightweight routines
- Throttling of events at runtime:
 - % **setenv TAU_THROTTLE 1**
 - Turns off instrumentation in routines that execute over 100000 times (TAU_THROTTLE_NUMCALLS) and take less than 10 microseconds of inclusive time per call (TAU_THROTTLE_PERCALL)
- Selective instrumentation file to filter events
 - % **tau_instrumentor [options] -f <file> OR**
 - % **setenv TAU_OPTIONS '-optTauSelectFile=tau.txt'**
- Compensation of local instrumentation overhead
 - % **configure -COMPENSATE**

Selective Instrumentation File



- Specify a list of routines to exclude or include (case sensitive)
- # is a wildcard in a routine name. It cannot appear in the first column.

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

- Specify a list of routines to include for instrumentation

```
BEGIN_INCLUDE_LIST
int main(int, char **)
F1
F3
END_EXCLUDE_LIST
```

- Specify either an include list or an exclude list!

Selective Instrumentation File



- Optionally specify a list of files to exclude or include (case sensitive)
- * and ? may be used as wildcard characters in a file name

```
BEGIN_FILE_EXCLUDE_LIST
f*.f90
Foo?.cpp
END_FILE_EXCLUDE_LIST
```

- Specify a list of routines to include for instrumentation

```
BEGIN_FILE_INCLUDE_LIST
main.cpp
foo.f90
END_FILE_INCLUDE_LIST
```

Selective Instrumentation File



- ❑ User instrumentation commands are placed in INSTRUMENT section
- ❑ ? and * used as wildcard characters for file name, # for routine name
- ❑ \ as escape character for quotes
- ❑ 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"
file="foo.f90" line = 123 code = " print *, \" In foo\""
exit routine = "int f1()" code = "cout <<\"Out f1\"<<endl;"
END_INSTRUMENT_SECTION
```

Using TAU



- ❑ Install TAU
 - % ./configure [options]; make clean install
- ❑ Replace the names of your compiler with tau_f90.sh, tau_cxx.sh and tau_cc.sh in your makefiles
- ❑ Set environment variables
 - Choose the measurement option and compile your code:
 - setenv TAU_MAKEFILE \$TAU/Makefile.tau-icpc-mpi-pdt
 - setenv TAU_OPTIONS '-optVerbose -optKeepFiles -optPreProcess'
 - setenv TAU_THROTTLE 1
 - At runtime to keep instrumentation overhead in check
 - At runtime, if more than one metric is measured (-multiplecounters):
 - setenv COUNTER1 GET_TIME_OF_DAY
 - setenv COUNTER2 PAPI_FP_INS
 - setenv COUNTER3 PAPI_NATIVE_<native_name>
 - Use papi_native_avail, papi_avail, and papi_event_chooser to select these preset and native event names
- ❑ Build the application, run it, analyze performance data

Compiling Fortran Codes with TAU: Tips



- ❑ If your Fortran code uses free format in .f files (fixed is default for .f), you may use:
% setenv TAU_OPTIONS '-optPdtF95Opts="-R free" -optVerbose'
- ❑ If it uses several module files, you may switch from the default Cleanscape Inc. parser in PDT to the GNU gfortran parser to generate PDB files:
% setenv TAU_OPTIONS '-optPdtGnuFortranParser -optVerbose'
- ❑ If your Fortran code uses C preprocessor directives (#include, #ifdef, #endif):
% setenv TAU_OPTIONS '-optPreProcess -optVerbose -optDetectMemoryLeaks'
- ❑ To use an instrumentation specification file:
% setenv TAU_OPTIONS '-optTauSelectFile=mycmd.tau -optVerbose -optPreProcess'
% cat mycmd.tau
BEGIN_INSTRUMENT_SECTION
memory file="foo.f90" routine="#"
instruments all allocate/deallocate statements in all routines in foo.f90
loops file="*" routine="#"
io file="abc.f90" routine="FOO"
END_INSTRUMENT_SECTION

Instrumentation of OpenMP Constructs



- ❑ **OpenMP Pragma And Region Instrumentor** [UTK, FZJ]
- ❑ Source-to-Source translator to insert **POMP** calls around OpenMP constructs and API functions
- ❑ **Done:** Supports
 - Fortran77 and Fortran90, OpenMP 2.0
 - C and C++, OpenMP 1.0
 - POMP Extensions
 - EPILOG and TAU POMP implementations
 - Preserves source code information (**#line line file**)
- ❑ **tau_ompcheck**
 - Balances OpenMP constructs (DO/END DO) and detects errors
 - Invoked by tau_compiler.sh prior to invoking Opari
- ❑ KOJAK Project website <http://icl.cs.utk.edu/kojak>



OpenMP API Instrumentation



□ Transform

- `omp_#_lock()` → `pomp_#_lock()`
- `omp_#_nest_lock()` → `pomp_#_nest_lock()`

[# = `init` | `destroy` | `set` | `unset` | `test`]

□ POMP version

- Calls `omp` version internally
- Can do extra stuff before and after call

Example: !\$OMP PARALLEL DO Instrumentation



```
call pomp_parallel_fork(d)
!$OMP PARALLEL other-clauses...
  call pomp_parallel_begin(d)
  call pomp_do_enter(d)
  !$OMP DO schedule-clauses, ordered-clauses,
             lastprivate-clauses
    do loop
  !$OMP END DO NOWAIT
  call pomp_barrier_enter(d)
  !$OMP BARRIER
  call pomp_barrier_exit(d)
  call pomp_do_exit(d)
  call pomp_parallel_end(d)
!$OMP END PARALLEL DO
call pomp_parallel_join(d)
```

Opari Instrumentation: Example

□ OpenMP directive instrumentation

```
pomp_for_enter(&omp_rd_2);
#line 252 "stommel.c"
#pragma omp for schedule(static) reduction(+: diff) private(j)
firstprivate (a1,a2,a3,a4,a5) nowait
for( i=1;i<=i2;i++) {
  for(j=j1;j<=j2;j++){
    new_psi[i][j]=a1*psi[i+1][j] + a2*psi[i-1][j] + a3*psi[i][j+1]
    + a4*psi[i][j-1] - a5*the_for[i][j];
    diff=diff+fabs(new_psi[i][j]-psi[i][j]);
  }
}
pomp_barrier_enter(&omp_rd_2);
#pragma omp barrier
pomp_barrier_exit(&omp_rd_2);
pomp_for_exit(&omp_rd_2);
```

Using Opari with TAU

Step I: Configure KOJAK/opari [Download from <http://www.fz-juelich.de/zam/kojak/>]

```
% cd kojak-2.1.1; cp mf/Makefile.defs.ibm Makefile.defs;
  edit Makefile
```

```
% make
```

Builds opari

Step II: Configure TAU with Opari (used here with MPI and PDT)

```
% configure -opari=/usr/contrib/TAU/kojak-2.1.1/opari
  -mpiinc=/usr/lpp/ppe.poe/include
  -mpilib=/usr/lib
  -pdt=/usr/contrib/TAU/pdtoolkit-3.9
```

```
% make clean; make install
```

```
% setenv TAU_MAKEFILE /tau/<arch>/lib/Makefile.tau-...opari-...
```

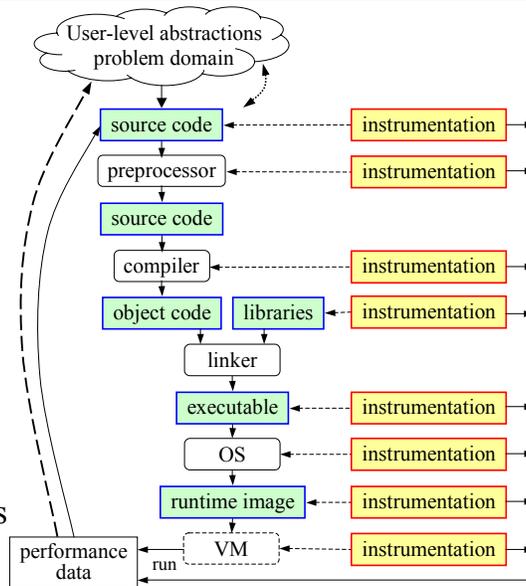
```
% tau_cxx.sh -c foo.cpp
```

```
% tau_cxx.sh -c bar.f90
```

```
% tau_cxx.sh *.o -o app
```

Multi-Level Instrumentation and Mapping

- ❑ **Multiple interfaces**
- ❑ **Information sharing**
 - Between interfaces
- ❑ **Event selection**
 - Within/between levels
- ❑ **Mapping**
 - Associate performance data with high-level semantic abstractions



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 SLOG2, OTF, EPILOG, and Paraver
 - Trace streams (OTF) and hierarchical trace merging
- ❑ Robust timing and hardware performance support
- ❑ Multiple counters (hardware, user-defined, system)
- ❑ Performance measurement for CCA component software

TAU Measurement Mechanisms



- ❑ **Parallel profiling**
 - Function-level, block-level, statement-level
 - Supports user-defined events and mapping events
 - TAU parallel profile stored (dumped) during execution
 - Support for flat, callgraph/callpath, phase profiling
 - Support for memory profiling (headroom, malloc/leaks)
 - Support for tracking I/O (wrappers, Fortran instrumentation of read/write/print calls)
- ❑ **Tracing**
 - All profile-level events
 - Inter-process communication events
 - Inclusion of multiple counter data in traced events

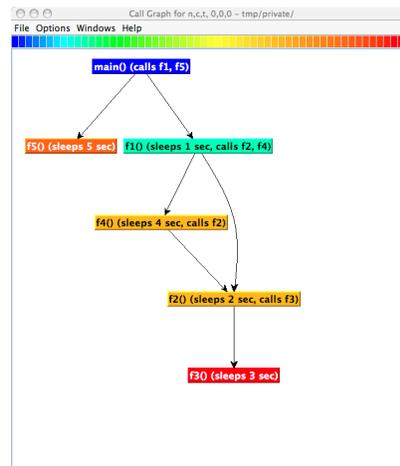
Types of Parallel Performance Profiling



- ❑ **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

-PROFILECALLPATH Configuration Option

- Generates program callgraph



Profile Measurement – Three Flavors

- Flat profiles
 - Time (or counts) spent in each routine (nodes in callgraph).
 - Exclusive/inclusive time, no. of calls, child calls
 - E.g.: MPI_Send, foo, ...
- Callpath Profiles
 - Flat profiles, **plus**
 - Sequence of actions that led to poor performance
 - Time spent along a calling path (edges in callgraph)
 - E.g., “main=> f1 => f2 => MPI_Send” shows the time spent in MPI_Send when called by f2, when f2 is called by f1, when it is called by main. Depth of this callpath = 4 (TAU_CALLPATH_DEPTH environment variable)
- Phase based profiles
 - Flat profiles, **plus**
 - Flat profiles under a phase (nested phases are allowed)
 - Default “main” phase has all phases and routines invoked outside phases
 - Supports static or dynamic (per-iteration) phases
 - E.g., “IO => MPI_Send” is time spent in MPI_Send in IO phase

-DEPTHLIMIT Configuration 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

-COMPENSATE Configuration Option



- ❑ Specifies online compensation of performance perturbation
- ❑ TAU computes its timer overhead and subtracts it from the profiles
- ❑ Works well with time or instructions based metrics
- ❑ Does not work with level 1/2 data cache misses

-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**

-PROFILEPARAM Configuration Option



- ❑ Idea: partition performance data for individual functions based on runtime parameters
- ❑ Enable by configuring with **-PROFILEPARAM**
- ❑ TAU call: TAU_PROFILE_PARAM1L (value, "name")
- ❑ Simple example:

```
void foo(long input) {
    TAU_PROFILE("foo", "", TAU_DEFAULT);
    TAU_PROFILE_PARAM1L(input, "input");
    ... }

```

Workload Characterization



- ❑ 5 seconds spent in function “foo” becomes
 - 2 seconds for “foo [<input> = <25>]”
 - 1 seconds for “foo [<input> = <5>]”
 - ...
- ❑ Currently used in MPI wrapper library
 - Allows for partitioning of time spent in MPI routines based on parameters (message size, message tag, destination node)
 - Can be extrapolated to infer specifics about the MPI subsystem and system as a whole

Workload Characterization



- ❑ Simple example, send/receive squared message sizes (0-32MB)

```
#include <stdio.h>
#include <mpi.h>
int buffer[8*1024*1024];

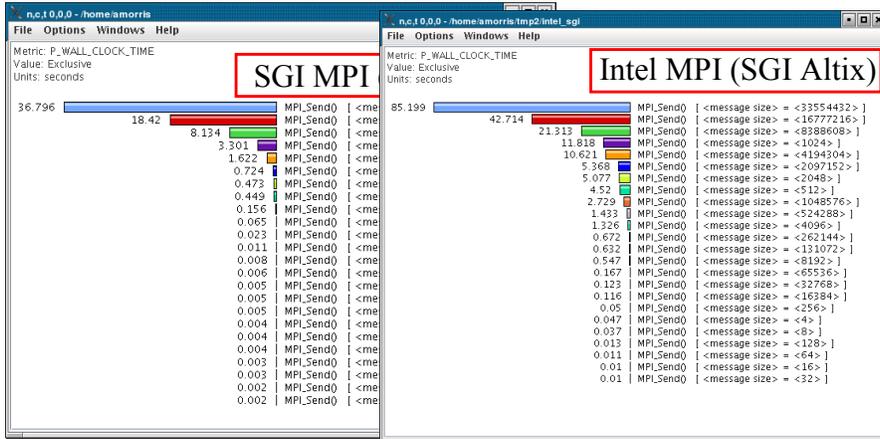
int main(int argc, char **argv) {
    int rank, size, i, j;
    MPI_Init(&argc, &argv);
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    for (i=0;i<1000;i++)
        for (j=1;j<=8*1024*1024;j*=2) {
            if (rank == 0) {
                MPI_Send(buffer, j, MPI_INT, 1, 42, MPI_COMM_WORLD);
            } else {
                MPI_Status status;
                MPI_Recv(buffer, j, MPI_INT, 0, 42, MPI_COMM_WORLD, &status);
            }
        }
    MPI_Finalize();
}
```

Workload Characterization

- Use `tau_load.sh` to instrument MPI routines (SGI Altix)

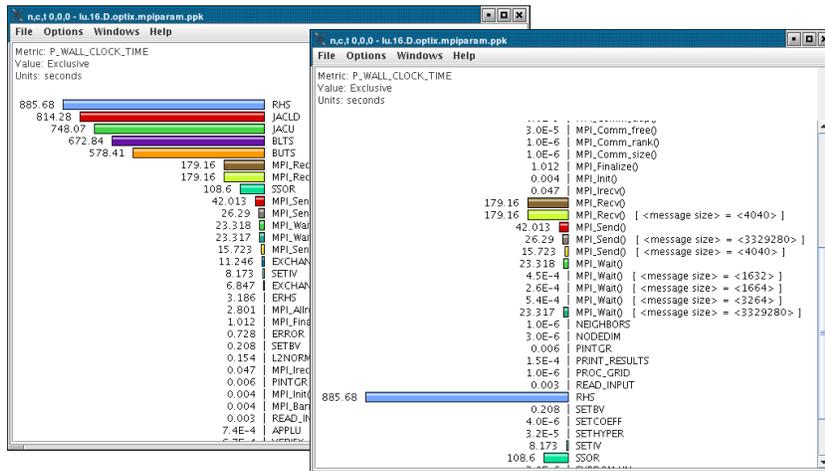
```
% icc mpi.c -lmpi
```

```
% mpirun -np 2 tau_load.sh -XrunTAU-icpc-mpi-pdt.so a.out
```



Workload Characterization

- MPI Results (NAS Parallel Benchmark 3.1, LU class D on 16 processors of SGI Altix)

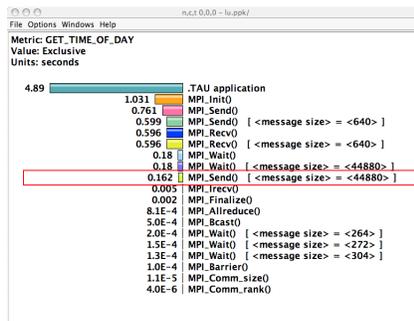


Workload Characterization

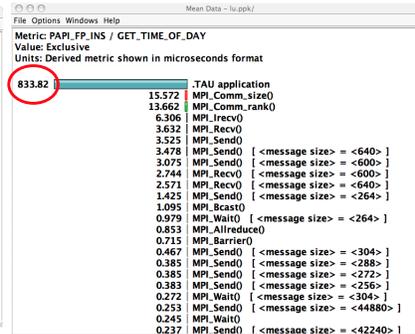
- Two different message sizes (~3.3MB and ~4K)

Name	Inclusive...	Exclusive...	Calls	Child...
MPI_Comm_free()	0	0	1	0
MPI_Comm_rank()	0	0	1	0
MPI_Comm_size()	0	0	2	0
MPI_Finalize()	1.012	1.012	1	0
MPI_Init()	0.004	0.004	1	0
MPI_Irecv()	0.047	0.047	612	0
MPI_Recv()	179.165	179.165	244,412	0
MPI_Recv() [<message size> = <4040>]	179.165	179.165	244,412	0
MPI_Send()	42.013	42.013	245,020	0
MPI_Send() [<message size> = <3329280>]	26.29	26.29	608	0
MPI_Send() [<message size> = <4040>]	15.723	15.723	244,412	0
MPI_Wait()	23.317	23.317	612	0
MPI_Wait() [<message size> = <1632>]	0	0	1	0
MPI_Wait() [<message size> = <1664>]	0	0	1	0
MPI_Wait() [<message size> = <3264>]	0.001	0.001	2	0
MPI_Wait() [<message size> = <3329280>]	23.317	23.317	608	0
NEIGHBORS	0	0	1	0
NODEDIM	0	0	1	0
PINTGR	0.008	0.006	1	6
PRINT_RESULTS	0	0	1	0

Job Tracking: ParaProf profile browser



LU spent 0.162 seconds sending messages of size 44880



It got 833.82 Mflops!

Memory Profiling in TAU



- Configuration option **-PROFILEMEMORY**
 - Records global heap memory utilization for each function
 - Takes one sample at beginning of each function and associates the sample with **function name**
- Configuration option **-PROFILEHEADROOM**
 - Records headroom (amount of free memory to grow) for each function
 - Takes one sample at beginning of each function and associates it with the **callstack** [TAU_CALLPATH_DEPTH env variable]
 - Useful for debugging memory usage on IBM BG/L.
- Independent of instrumentation/measurement options selected
- No need to insert macros/calls in the source code
- User defined atomic events appear in profiles/traces

Memory Profiling in TAU (Atomic events)



Sorted By: number of userEvents

NumSamples	Max	Min	Mean	Std. Dev	Name
252032	2022.7	1181.2	1534.3	410.04	MODULEHYDRO_ID::HYDRO_ID - Heap Memory (KB)
252032	2022.8	1181.7	1534.3	410.04	MODULEINTRFC::INTRFC - Heap Memory (KB)
104559	2023.2	331.13	1526.6	409.54	MODULEEOS3D::EOS3D - Heap Memory (KB)
63008	2022.7	1182	1534.3	410.01	MODULEUPDATE_SOLN::UPDATE_SOLN - Heap Memory (KB)
55545	2023.3	333.07	1514.2	408.31	DBASETREE::DBASENEIGHBORBLOCKLIST - Heap Memory (KB)
51374	2023	1179.4	1497.7	402.53	AMR_PROLONG_GEN_UNK_FUN - Heap Memory (KB)
42120	2022.7	1187.5	1533.5	409.83	ABUNDANCE_RESTRICT - Heap Memory (KB)
41958	2023	346.12	1514.9	408.39	AMR_RESTRICT_UNK_FUN - Heap Memory (KB)
31832	2022.8	1187.4	1534.1	409.91	AMR_RESTRICT_RED - Heap Memory (KB)
31504	2022.7	1181.8	1534.3	410.04	DIFFUSE - Heap Memory (KB)
26042	2023	1179.2	1501.9	403.61	AMR_PROLONG_UNK_FUN - Heap Memory (KB)

Flash2 code profile (-PROFILEMEMORY) on IBM BlueGene/L [MPI rank 0]

Memory Profiling in TAU



- ❑ Instrumentation based observation of global heap memory (not per function)
 - call TAU_TRACK_MEMORY()
 - call TAU_TRACK_MEMORY_HEADROOM()
 - Triggers one sample every 10 secs
 - call TAU_TRACK_MEMORY_HERE()
 - call TAU_TRACK_MEMORY_HEADROOM_HERE()
 - Triggers sample at a specific location in source code
 - call TAU_SET_INTERRUPT_INTERVAL(seconds)
 - To set inter-interrupt interval for sampling
 - call TAU_DISABLE_TRACKING_MEMORY()
 - call TAU_DISABLE_TRACKING_MEMORY_HEADROOM()
 - To turn off recording memory utilization
 - call TAU_ENABLE_TRACKING_MEMORY()
 - call TAU_ENABLE_TRACKING_MEMORY_HEADROOM()
 - To re-enable tracking memory utilization

Detecting Memory Leaks in C/C++

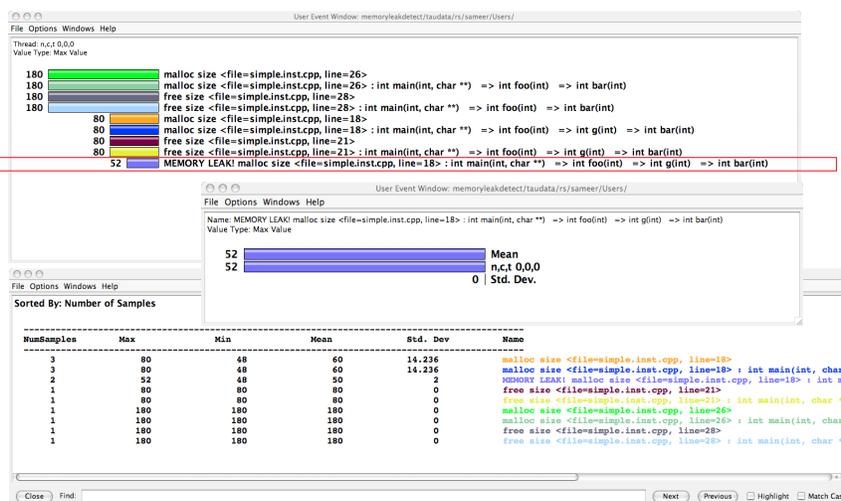


- ❑ TAU wrapper library for malloc/realloc/free
- ❑ During instrumentation, specify
 - optDetectMemoryLeaks option to TAU_COMPILER
 - % setenv TAU_OPTIONS '-optVerbose -optDetectMemoryLeaks'
 - % setenv TAU_MAKEFILE <taudir>/<arch>/lib/Makefile.tau-icpc-mpi-pdt...
 - % tau_cxx.sh foo.cpp ...
- ❑ Tracks each memory allocation/de-allocation in parsed files
- ❑ Correlates each memory event with the executing callstack
- ❑ At the end of execution, TAU detects memory leaks
- ❑ TAU reports leaks based on allocations and the executing callstack
- ❑ Set **TAU_CALLPATH_DEPTH** environment variable to limit callpath data
 - default is 2
- ❑ Future work
 - Support for C++ new/delete planned
 - Support for Fortran 90/95 allocate/deallocate planned

Detecting Memory Leaks in C/C++

```
include /opt/tau/x86_64/lib/Makefile.tau-icpc-mpi-pdt
MYOPTS = -optVerbose -optDetectMemoryLeaks
CC= $(TAU_COMPILER) $(MYOPTS) $(TAU_CXX)
LIBS = -lm
OBJS = f1.o f2.o ...
TARGET= a.out
TARGET: $(OBJS)
    $(F90) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)
.c.o:
    $(CC) $(CFLAGS) -c $< -o $@
```

Memory Leak Detection



Detecting Memory Leaks in Fortran



```
subroutine foo(x)
  integer:: x
  integer, allocatable :: A(:), B(:), C(:)

  print *, "inside foo"
  allocate(A(x), B(x), C(x))
  deallocate(A, C)
  print *, "exiting foo"

end subroutine foo

program main
  call foo(5)
end program main
```

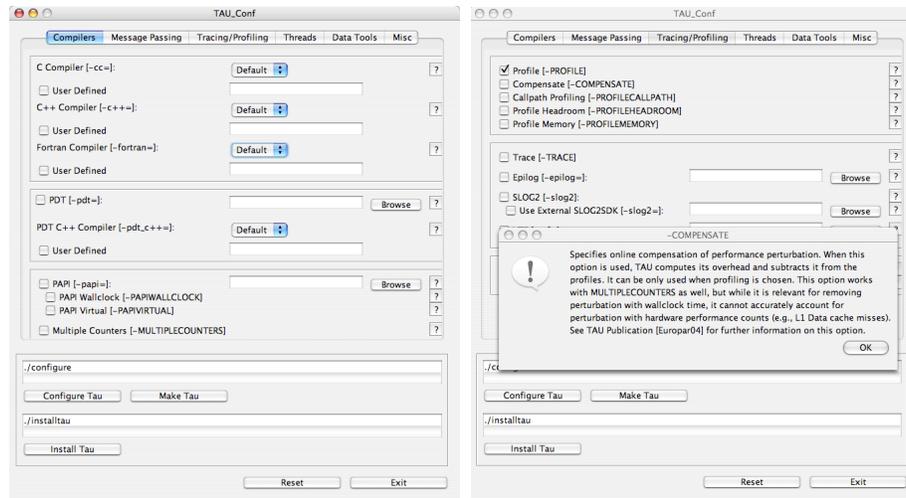
Detecting Memory Leaks in Fortran



USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0

NumSamples	MaxValue	MinValue	MeanValue	Std. Dev.	Event Name
1	5	5	5	0	MEMORY LEAK! malloc size <file=simple.f, variable=B, line=6> : MAIN => FOO
1	5	5	5	0	free size <file=simple.f, variable=A, line=7>
1	5	5	5	0	free size <file=simple.f, variable=A, line=7> : MAIN => FOO
1	5	5	5	0	free size <file=simple.f, variable=C, line=7>
1	5	5	5	0	free size <file=simple.f, variable=C, line=7> : MAIN => FOO
1	5	5	5	0	malloc size <file=simple.f, variable=A, line=6>
1	5	5	5	0	malloc size <file=simple.f, variable=A, line=6> : MAIN => FOO
1	5	5	5	0	malloc size <file=simple.f, variable=B, line=6>
1	5	5	5	0	malloc size <file=simple.f, variable=B, line=6> : MAIN => FOO
1	5	5	5	0	malloc size <file=simple.f, variable=C, line=6>
1	5	5	5	0	malloc size <file=simple.f, variable=C, line=6> : MAIN => FOO

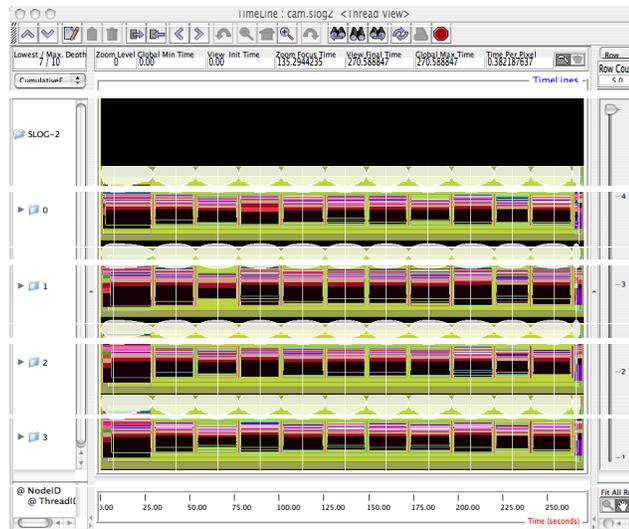
TAU_SETUP: A GUI for Installing TAU



Jumpshot

- <http://www-unix.mcs.anl.gov/perfvis/software/viewers/index.htm>
- Developed at Argonne National Laboratory as part of the MPICH project
- Rusty Lusk, PI
 - Also works with other MPI implementations
 - Jumpshot is bundled with the TAU package
- Java-based tracefile visualization tool for postmortem performance analysis of MPI programs
- Latest version is Jumpshot-4 for SLOG-2 format
 - Scalable level of detail support
 - Timeline and histogram views
 - Scrolling and zooming
 - Search/scan facility

Jumpshot



Tracing: Using TAU and Jumpshot

- ❑ Configure TAU with **-TRACE** option:
 - % `configure -TRACE -otf=<dir>`
 - % `-MULTIPLECOUNTERS -papi=<dir> -mpi`
 - % `-pdt=dir ...`
- ❑ Set environment variables:
 - % `setenv TRACEDIR /p/gml/<login>/traces`
 - % `setenv COUNTER1 GET_TIME_OF_DAY (reqd)`
 - % `setenv COUNTER2 PAPI_FP_INS`
 - % `setenv COUNTER3 PAPI_TOT_CYC ...`
- ❑ Execute application and analyze the traces:
 - % `mpirun -np 32 ./a.out [args]`

 - % `tau_treemerge.pl`
 - % `tau2slog2 tau.trc tau.edf -o app.slog2`
 - % `jumpshot app.slog2`

Performance Analysis and Visualization

- ❑ Analysis of parallel profile and trace measurement
- ❑ **Parallel profile analysis**
 - *ParaProf*: parallel profile analysis and presentation
 - *ParaVis*: parallel performance visualization package
 - Profile generation from trace data (*tau2profile*)
- ❑ Performance data management framework (*PerfDMF*)
- ❑ **Parallel trace analysis**
 - Translation to *VTF* (V3.0), *EPILOG*, *OTF* formats
 - Integration with *VNG* (Technical University of Dresden)
- ❑ Online parallel analysis and visualization
- ❑ Integration with *CUBE* browser (KOJAK, UTK, FZJ)

ParaProf Parallel Performance Profile Analysis

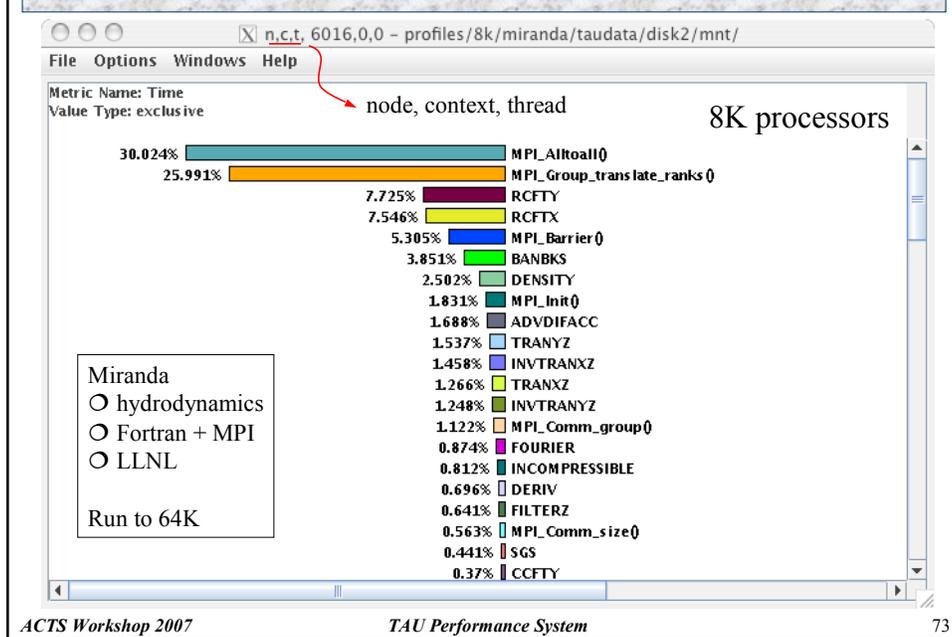
The screenshot displays the ParaProf Manager interface. On the left, a tree view shows the hierarchy of data: Applications, HPMToolkit Data, MpiP Data, and TAU. Red arrows point from labels to specific parts of the interface: 'Raw files' points to the Applications folder; 'PerfDMF managed (database)' points to the HPMToolkit Data folder; 'Application' points to the HPMToolkit Data folder; 'Experiment' points to the MpiP Data folder; and 'Trial' points to the TAU folder. The main window shows three performance profile plots for HPMToolkit, MpiP, and TAU. Each plot has a Y-axis labeled 'mean' and an X-axis labeled 'n.ct'. The HPMToolkit plot shows a mean of 0.0 and n.ct values of 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0. The MpiP plot shows a mean of 0.0 and n.ct values of 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0. The TAU plot shows a mean of 0.0 and n.ct values of 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0. A table in the top right corner shows metadata for the application, including Application ID, Benchmark ID, Trial ID, and Metric ID.

ACTS Workshop 2007

TAU Performance System

72

ParaProf – Flat Profile (Miranda, BG/L)



Terminology – Example

- For routine “int main()”:
- Exclusive time
 - 100-20-50-20=10 secs
- Inclusive time
 - 100 secs
- Calls
 - 1 call
- Subrs (no. of child routines called)
 - 3
- Inclusive time/call
 - 100secs

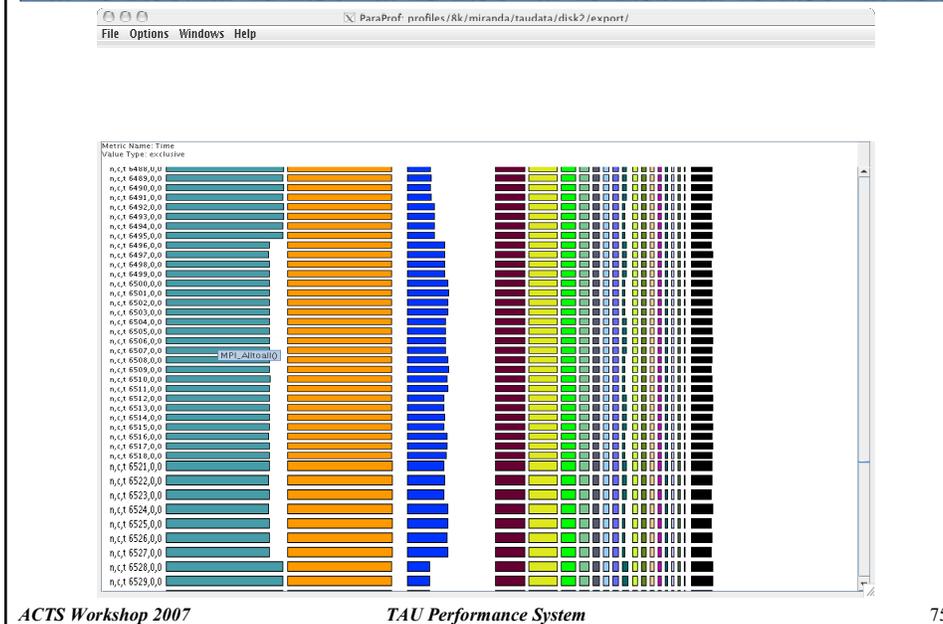
```
int main( )
{ /* takes 100 secs */

  f1(); /* takes 20 secs */
  f2(); /* takes 50 secs */
  f1(); /* takes 20 secs */

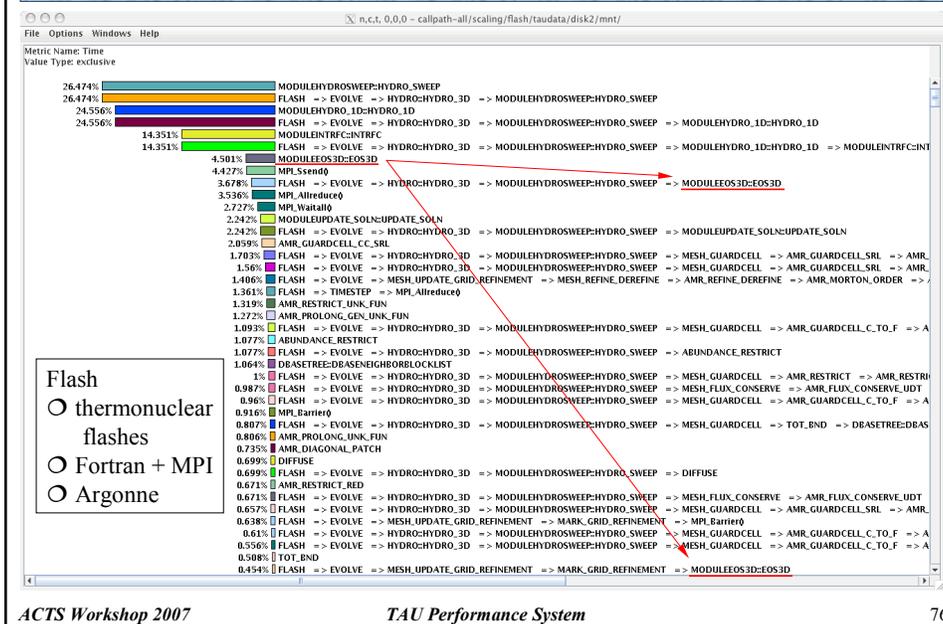
  /* other work */
}

/*
Time can be replaced by counts
from PAPI e.g., PAPI_FP_OPS. */
```

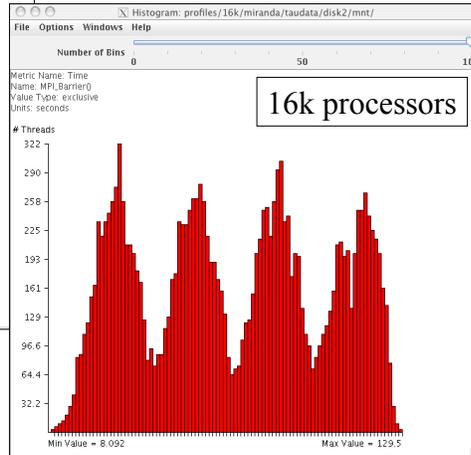
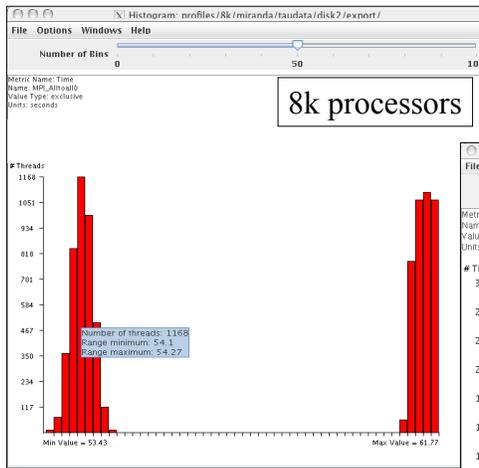
ParaProf – Stacked View (Miranda)



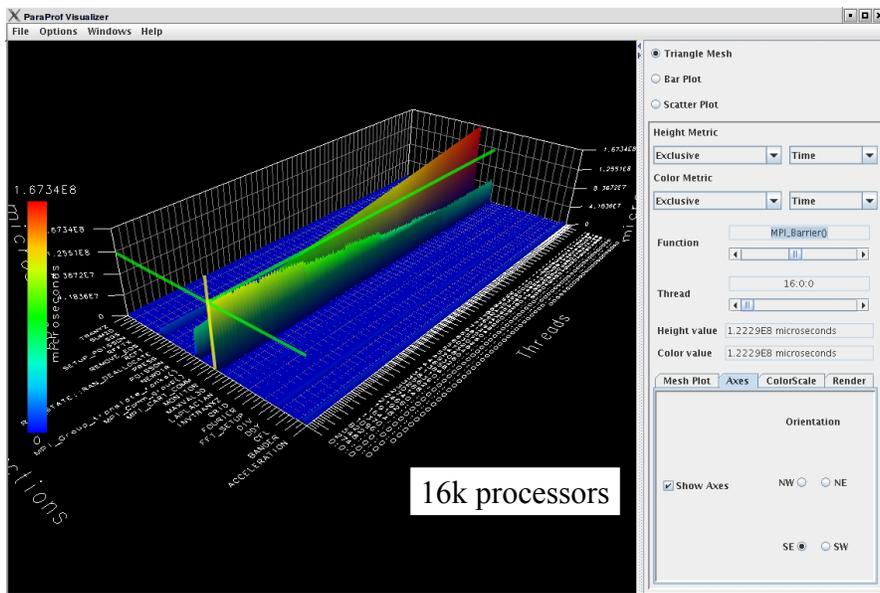
ParaProf – Callpath Profile (Flash)



ParaProf – Scalable Histogram View (Miranda)



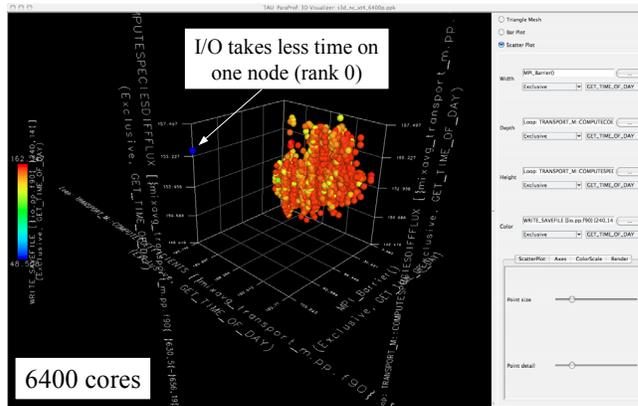
ParaProf – 3D Full Profile (Miranda)



ParaProf – 3D Scatterplot (S3D – XT4 only)

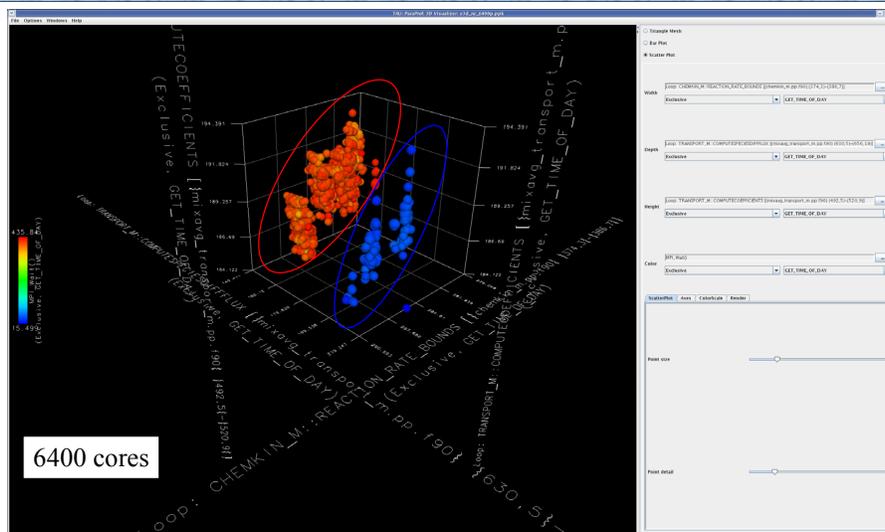


- ❑ Each point is a “thread” of execution
- ❑ A total of four metrics shown in relation
- ❑ ParaVis 3D profile visualization library
 - JOGL



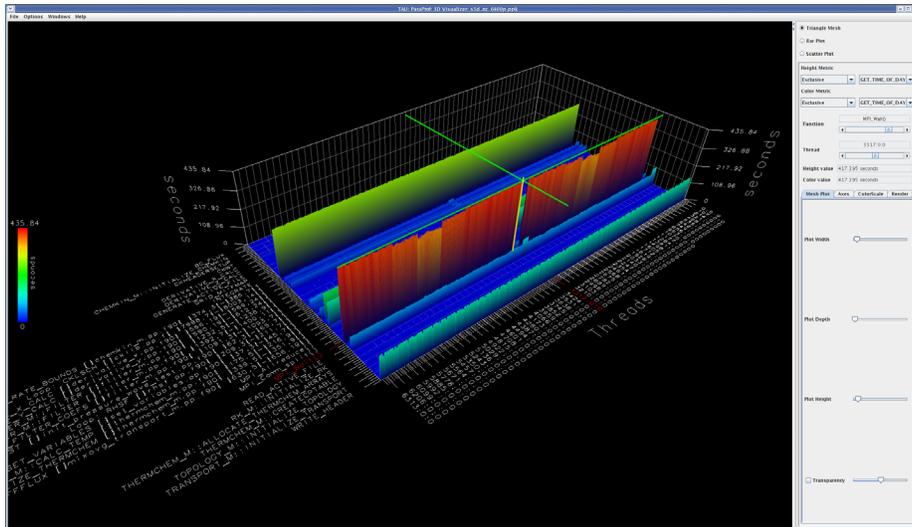
- ❑ Events (exclusive time metric)
 - MPI_Barrier(), two loops
 - write operation

S3D Scatter Plot: Visualizing Hybrid XT3+XT4



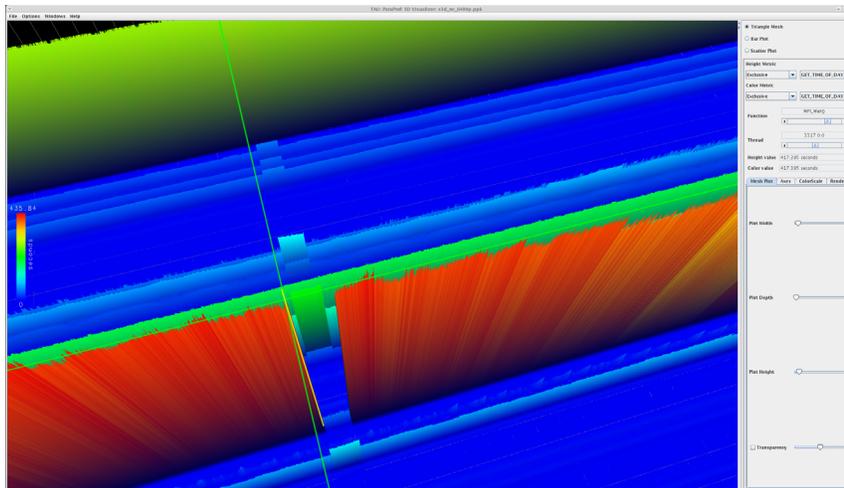
- ❑ Red nodes are XT4, blue are XT3

S3D: 6400 cores on XT3+XT4 System (Jaguar)



□ Gap represents XT3 nodes

Visualizing S3D Profiles in ParaProf



□ Gap represents XT3 nodes

○ MPI_Wait takes less time, other routines take more time

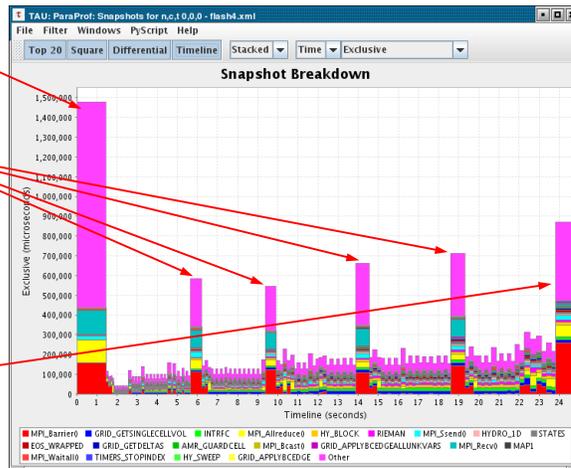
Profile Snapshots in ParaProf

- Profile snapshots are parallel profiles recorded at runtime
- Used to highlight profile changes during execution

Initialization

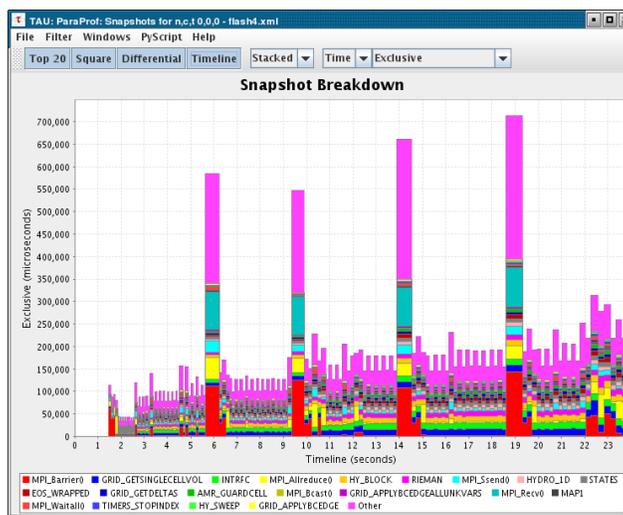
Checkpointing

Finalization



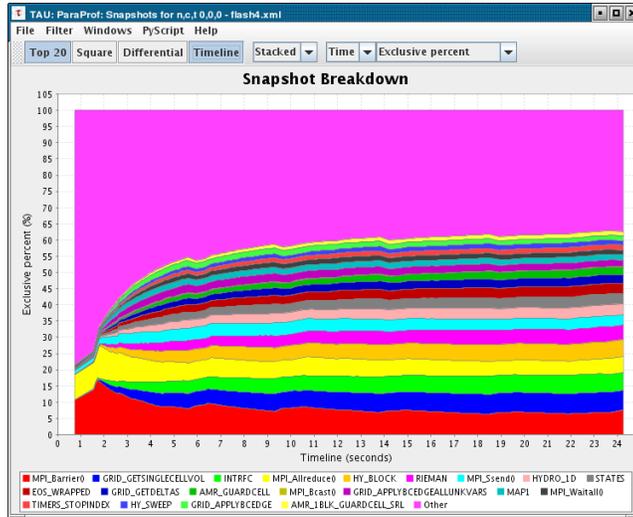
Profile Snapshots in ParaProf

- Filter snapshots (only show main loop iterations)



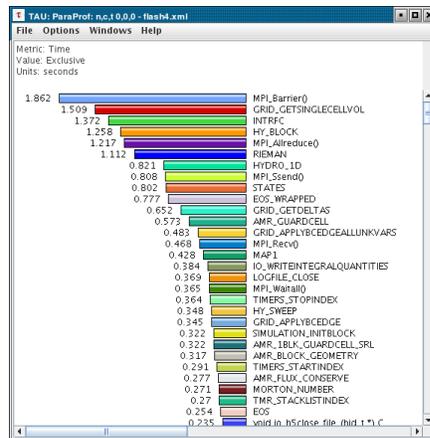
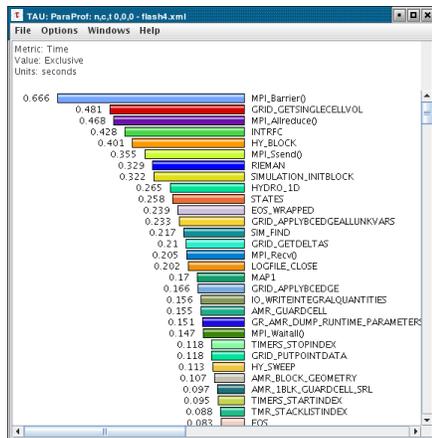
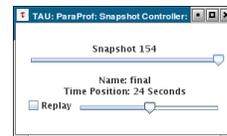
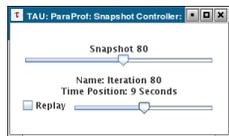
Profile Snapshots in ParaProf

- Breakdown as a percentage



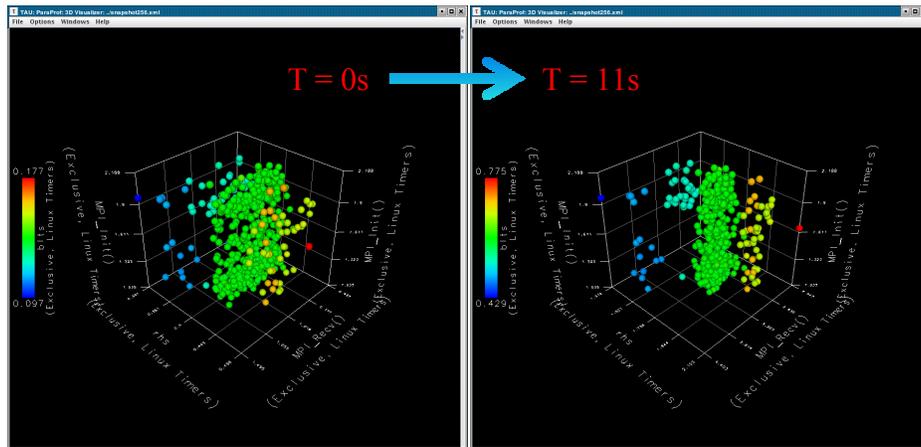
Snapshot replay in ParaProf

All windows dynamically update



Profile Snapshots in ParaProf

- ❑ Follow progression of various displays through time
- ❑ 3D scatter plot shown below



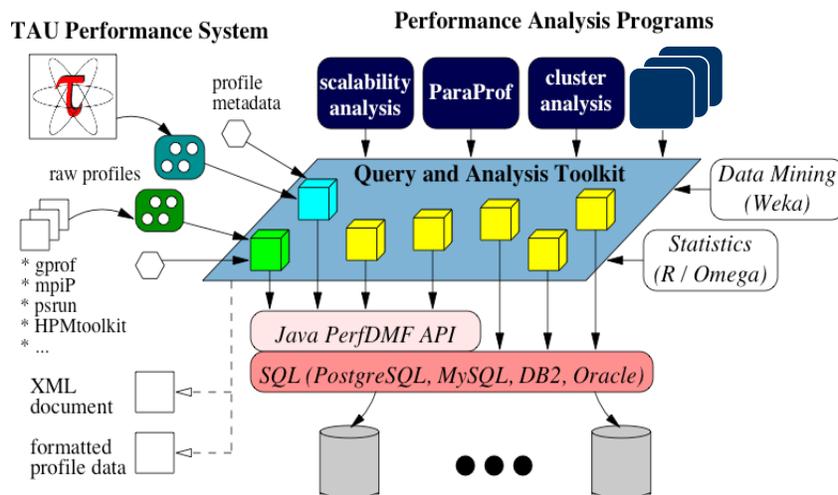
Performance Data Management: Motivation

- ❑ Need for robust processing and storage of multiple profile performance data sets
- ❑ Avoid developing independent data management solutions
 - Waste of resources
 - Incompatibility among analysis tools
- ❑ Goals:
 - Foster multi-experiment performance evaluation
 - Develop a common, reusable foundation of performance data storage, access and sharing
 - A core module in an analysis system, and/or as a central repository of performance data

PerfDMF Approach

- ❑ Performance Data Management Framework
- ❑ Originally designed to address critical TAU requirements
- ❑ Broader goal is to provide an open, flexible framework to support common data management tasks
- ❑ Extensible toolkit to promote integration and reuse across available performance tools
 - Supported profile formats:
TAU, CUBE, Dynaprof, HPC Toolkit, HPM Toolkit, gprof, mpiP, psrun (PerfSuite), others in development
 - Supported DBMS:
PostgreSQL, MySQL, Oracle, DB2, Derby/Cloudscape

PerfDMF Architecture



K. Huck, A. Malony, R. Bell, A. Morris, "Design and Implementation of a Parallel Performance Data Management Framework," ICPP 2005.

TAU Portal - www.paratools.com/tauportal



Connect to Database | PerIDMF Configuration File

Upload configuration file: no file selected

Connect to Database | Database Location

Database type:

Database host name:

Database port number:

Database name:

Database username:

User Password:

Remember this Database

This portal is an extension of the TAU performance system and ParaTools, Inc. It will connect to any database created with [perIDMF](#) using the default database schema. Within the portal you can view performance information from any trials loaded into the database. You may also interact with the database by uploading or downloading trials. This application is still under development, please send bugs or suggestion for improvements to [tau \[dash\] bugs \[at\] cs.uoregon.edu](mailto:tau [dash] bugs [at] cs.uoregon.edu).

Copyright © 1997-2006
Department of Computer and Information Science, University of Oregon
Advanced Computing Laboratory, LANL, NM
Research Centre Jülich, ZAM, Germany

TAU Portal



1. Select Application: HYCOM, hydroshock, LAMMPS, MFIX, mpiP data, PNEO, PDP, PTURBO

2. Select Experiment: Ozone decomposition in a bubbling fluidized bed (new Experiment)

3. Select Trial: Papi_net_2_cpu_runId TAUprofiledata3PSES (new Trial)

4. Select Metric: P_WALL_CLOCK_TIM, PAPI_FP_INS, PAPI_L1_DCM, PAPI_TOT_CYC, PAPI_TOT_INS, PAPI_FP_INS / P_WALL_CLOCK_TIM

Download Trial as: [Tab Delimited Packed Profile](#)

Function Exclusive P_WALL_CLOCK.TIME

Function	Value
OTHER	500,088,693,518,666
LEO_M60LVET	271,466,155,758
LEO_RECORD	482,664,563,452
CALC_VSI	1,170,833,937,617



Workspace: edit

http://tau.nic.uoregon.edu/trial/edit/7?workspace=7

TAU Portal | Welcome guest User | [Homepage](#) | [Edit User](#) | [Log Out](#) | [TAU Homepage](#)

Workspace

- Description
- Members
- Performance Data
- lammps16
- lammps32
- Project Files
- description.nf
- Images
- portal_functions.png
- portal_comparisons.p

lammps16 launch paraprof

Edit Trial

Name:

Time:

Number of Nodes:

Contexts per Node:

Threads per Context:

Compiler:

Edit Attributes

Name:

Value:

Model:

compiler:

Add a new attribute:

Using Performance Database (PerfDMF)



- ❑ **Configure PerfDMF (Done by each user)**
 - % perfdmf_configure
 - > Choose derby, PostgreSQL, MySQL, Oracle or DB2
 - > Hostname
 - > Username
 - > Password
 - > Say yes to downloading required drivers (we are not allowed to distribute these)
 - > Stores parameters in your ~/.ParaProf/perfdmf.cfg file
- ❑ **Configure PerfExplorer (Done by each user)**
 - % perfexplorer_configure
- ❑ **Execute PerfExplorer**
 - % perfexplorer

Recent PerfDMF Development



- ❑ Integration of XML metadata for each profile
 - Common Profile Attributes
 - Thread/process specific Profile Attributes
 - Automatic collection of runtime information
 - Any other data the user wants to collect can be added
 - Build information
 - Job submission information
 - Two methods for acquiring metadata:
 - TAU_METADATA() call from application
 - Optional XML file added when saving profile to PerfDMF
 - TAU Metadata XML schema is simple, easy to generate from scripting tools (no XML libraries required)

Performance Data Mining (Objectives)

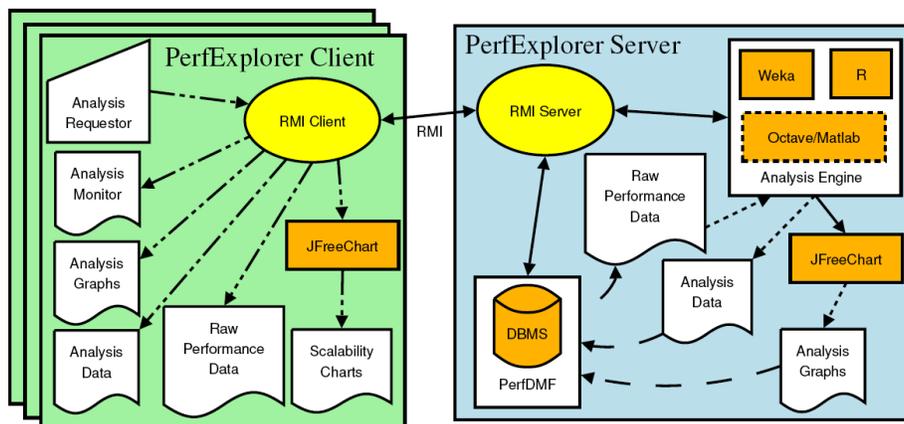


- ❑ Conduct parallel performance analysis process
 - In a systematic, collaborative and reusable manner
 - Manage performance complexity
 - Discover performance relationship and properties
 - Automate process
- ❑ Multi-experiment performance analysis
- ❑ Large-scale performance data reduction
 - Summarize characteristics of large processor runs
- ❑ Implement extensible analysis framework
 - Abstraction / automation of data mining operations
 - Interface to existing analysis and data mining tools

Performance Data Mining (PerfExplorer)

- ❑ Performance knowledge discovery framework
 - Data mining analysis applied to parallel performance data
 - comparative, clustering, correlation, dimension reduction, ...
 - Use the existing TAU infrastructure
 - TAU performance profiles, PerfDMF
 - Client-server based system architecture
- ❑ Technology integration
 - Java API and toolkit for portability
 - PerfDMF
 - R-project/Omegahat, Octave/Matlab statistical analysis
 - WEKA data mining package
 - JFreeChart for visualization, vector output (EPS, SVG)

Performance Data Mining (PerfExplorer)



K. Huck and A. Malony, "PerfExplorer: A Performance Data Mining Framework For Large-Scale Parallel Computing," SC 2005.

PerfExplorer Analysis Methods



- Data summaries, distributions, scatterplots
- Clustering
 - k*-means
 - Hierarchical
- Correlation analysis
- Dimension reduction
 - PCA
 - Random linear projection
 - Thresholds
- Comparative analysis
- Data management views

PerfDMF and the TAU Portal

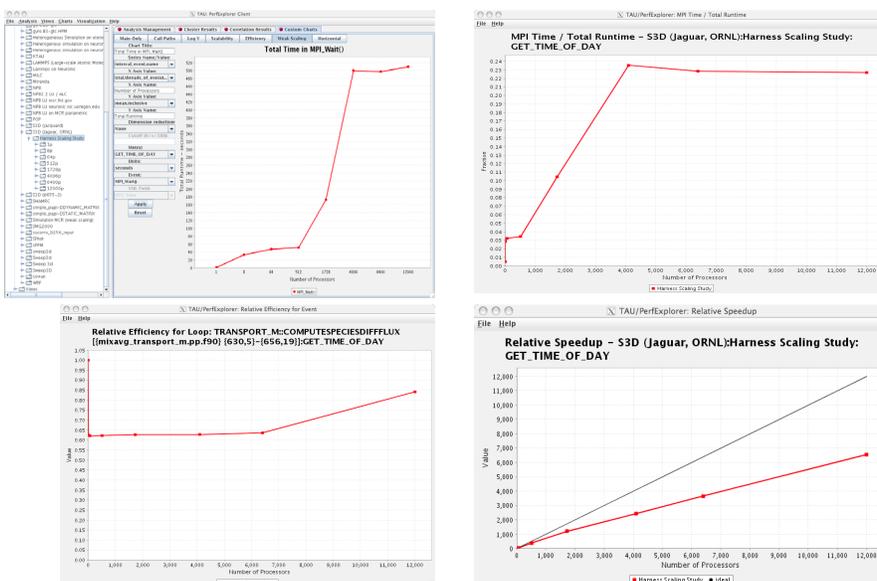


- Development of the TAU portal
 - Common repository for collaborative data sharing
 - Profile uploading, downloading, user management
 - Paraprof, PerfExplorer can be launched from the portal using Java Web Start (no TAU installation required)
- Portal URL
<http://tau.nic.uoregon.edu>

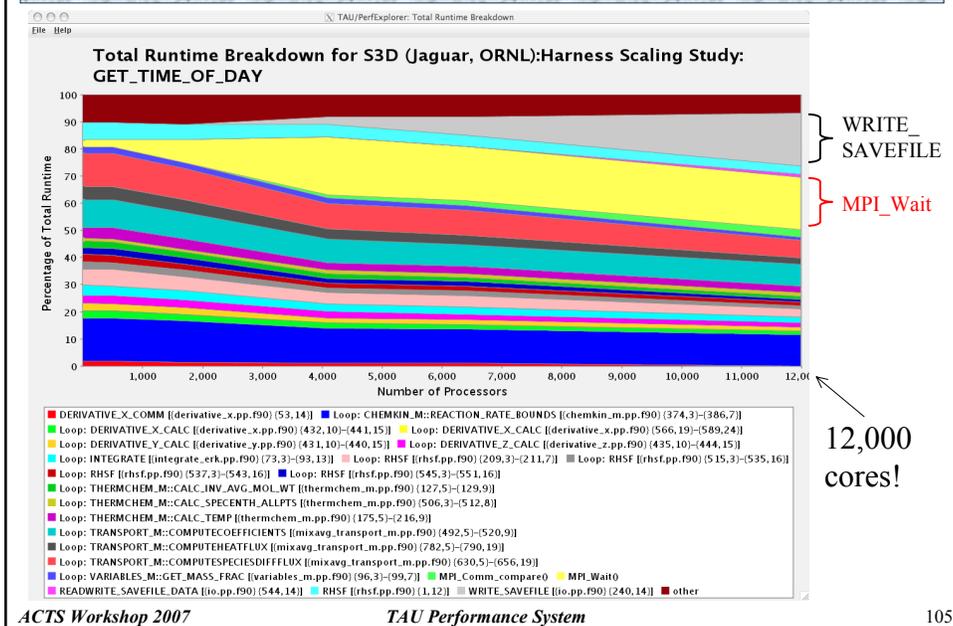
New automated metadata collection

TrialField	Value
Name	f90/pdt_mpi/examples/tau2/amorris/home/
Application ID	0
Experiment ID	0
Trial ID	0
CPU Cores	2
CPU MHz	2992.505
CPU Type	Intel(R) Xeon(R) CPU 5160 @ 3.00GHz
CPU Vendor	GenuineIntel
CWD	/home/amorris/tau2/examples/pdt_mpi/f90
Cache Size	4096 KB
Executable	/home/amorris/tau2/examples/pdt_mpi/f...
Hostname	demon.nic.uoregon.edu
Local Time	2007-07-04T04:21:14-07:00
MPI Processor Name	demon.nic.uoregon.edu
Memory Size	8161240 kB
Node Name	demon.nic.uoregon.edu
OS Machine	x86_64
OS Name	Linux
OS Release	2.6.9-42.0.3.EL.perfctrsm
OS Version	#1 SMP Fri Nov 3 07:34:13 PST 2006
Starting Timestamp	118354807220996
TAU Architecture	x86_64
TAU Config	-papi=/usr/local/packages/papi-3.5.0-M...
Timestamp	1183548074317538
UTC Time	2007-07-04T11:21:14Z
pid	11395
username	amorris

PerfExplorer: Cross Experiment Analysis for S3D



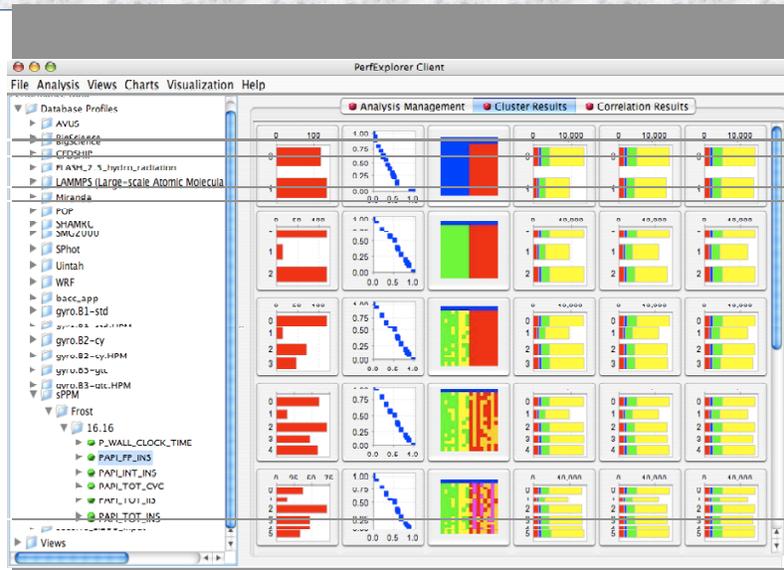
PerfExplorer: S3D Total Runtime Breakdown



PerfExplorer - Cluster Analysis

- ❑ Performance data represented as vectors - each dimension is the cumulative time for an event
- ❑ k -means: k random centers are selected and instances are grouped with the "closest" (Euclidean) center
- ❑ New centers are calculated and the process repeated until stabilization or max iterations
- ❑ Dimension reduction necessary for meaningful results
- ❑ Virtual topology, summaries constructed

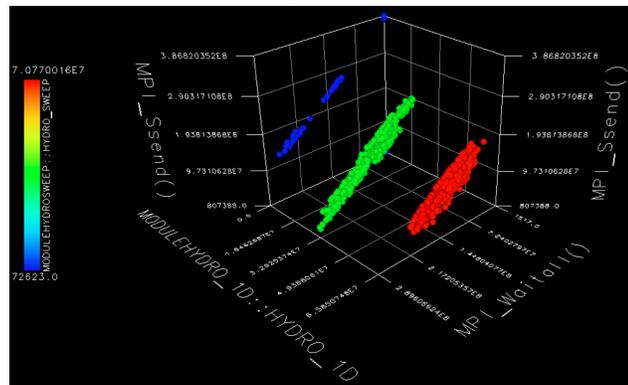
PerfExplorer - Cluster Analysis (sPPM)



PerfExplorer - Cluster Analysis

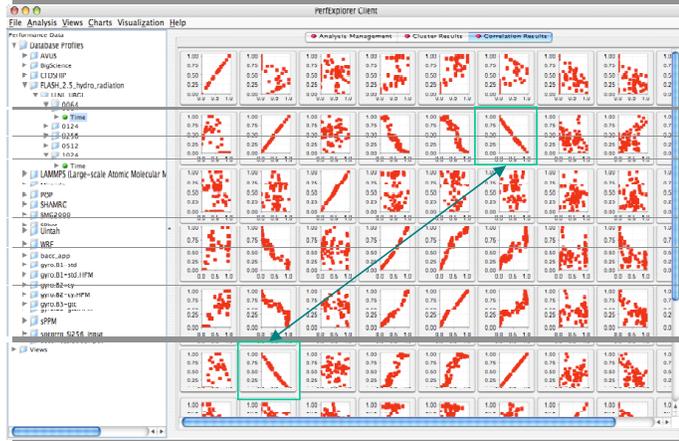


- ❑ Four significant events automatically selected (from 16K processors)
- ❑ Clusters and correlations are visible



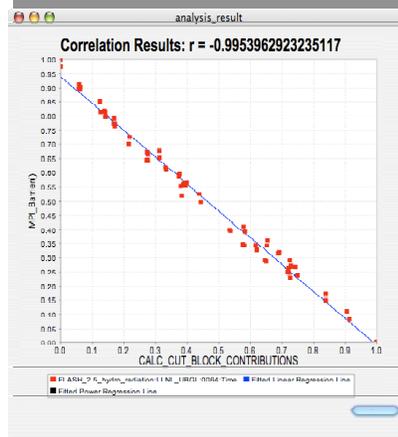
PerfExplorer - Correlation Analysis (Flash)

- Describe relationship between

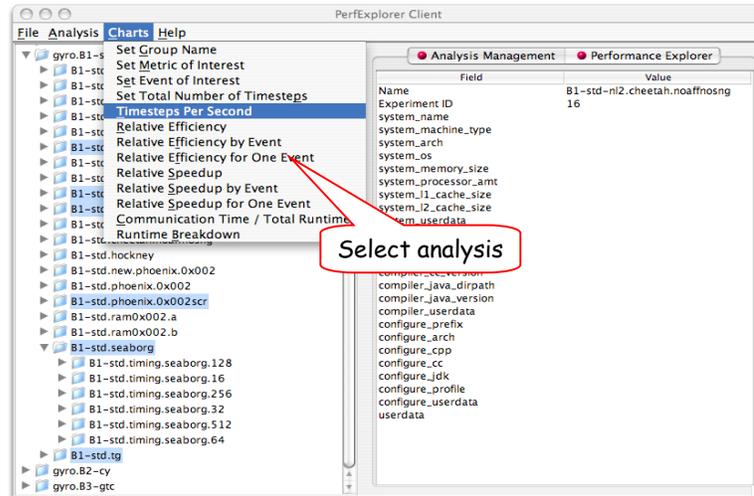


PerfExplorer - Correlation

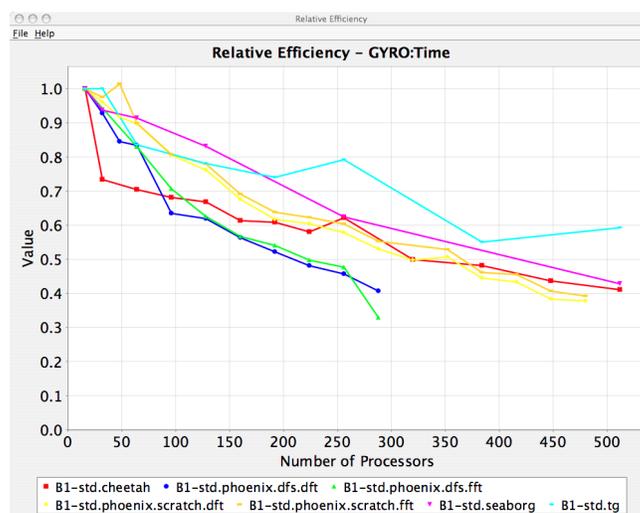
- 0.995 indicates strong, negative relationship
- As CALC_CUT_BLOCK_CONTRIBUTIONS() increases in execution time, MPI_Barrier() decreases



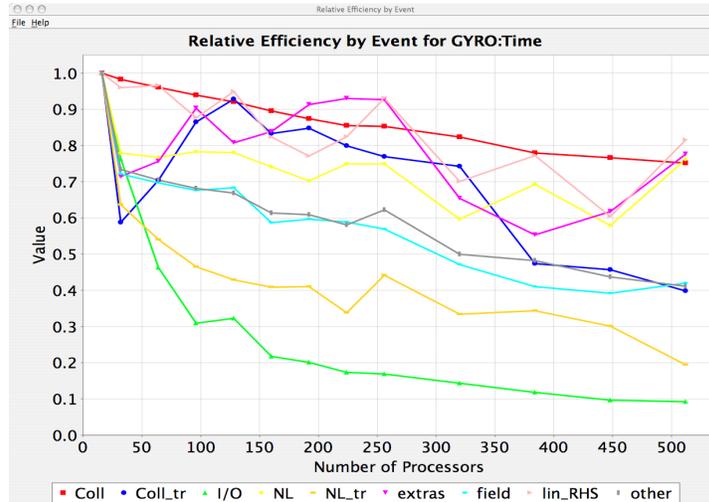
PerfExplorer - Interface



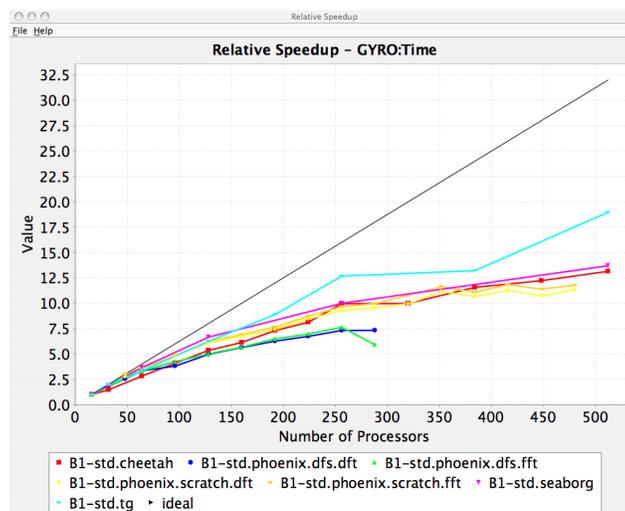
PerfExplorer - Relative Efficiency Plots



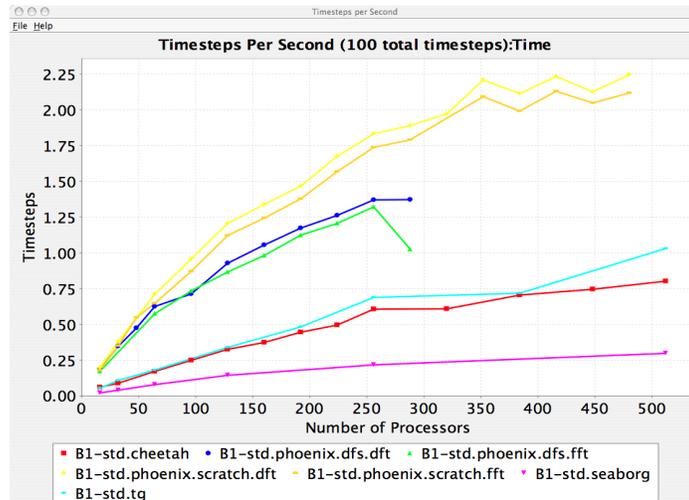
PerfExplorer - Relative Efficiency by Routine



PerfExplorer - Relative Speedup



PerfExplorer - Timesteps Per Second

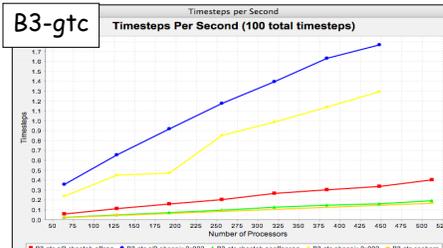
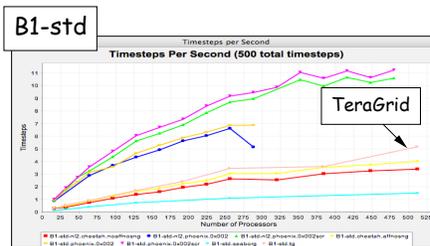
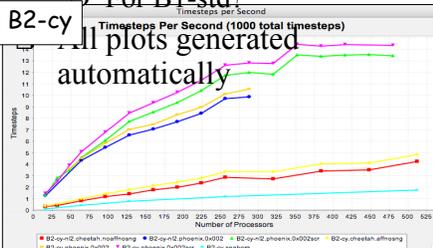


PerfExplorer - Timesteps per Second for GYRO



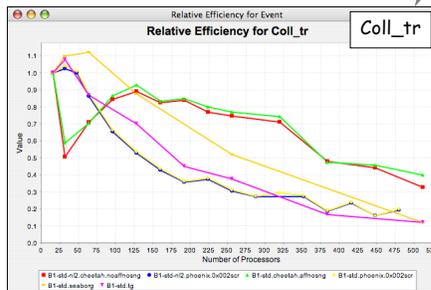
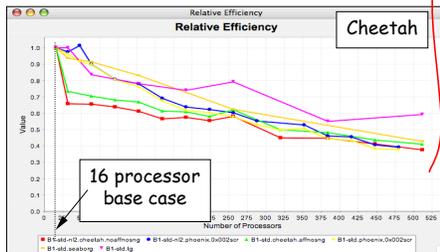
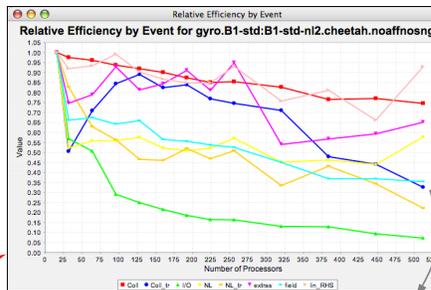
- ❑ Cray X1 is the fastest to solution
 - In all 3 tests
- ❑ FFT (nl2) improves time
 - B3-gtc only
- ❑ TeraGrid faster than p690

For B1-std?

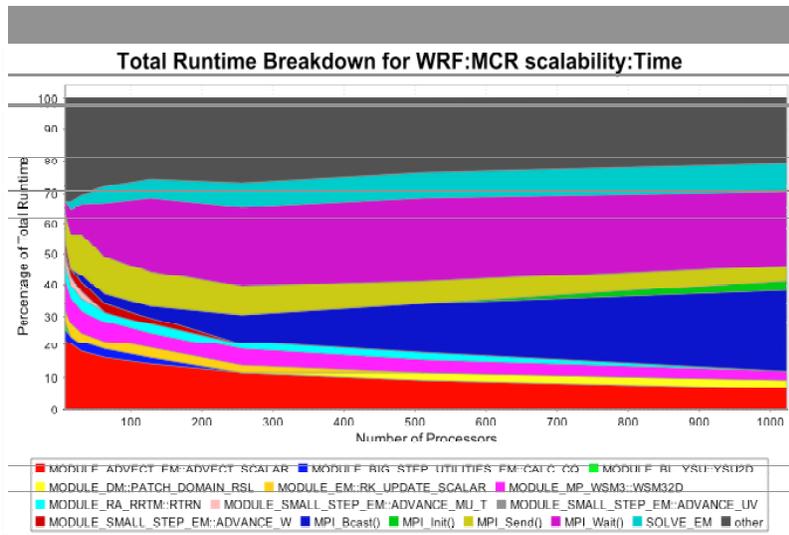


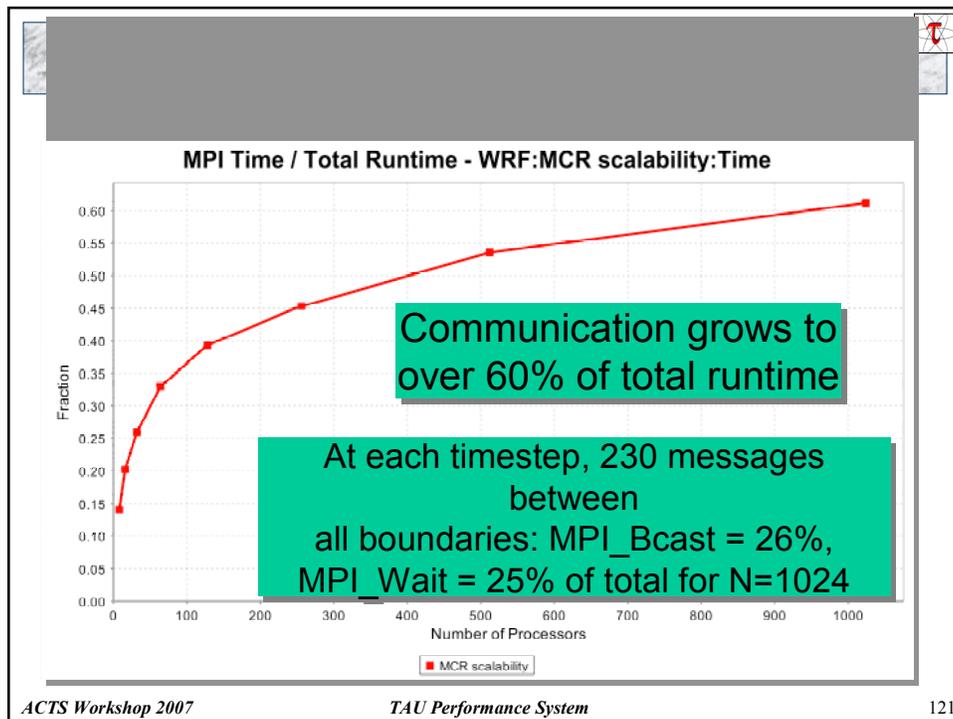
PerfExplorer - Relative Efficiency (B1-std)

- By experiment (B1-std)
 - Total runtime (Cheetah (red))
- By event for one experiment
 - Coll_tr (blue) is significant
- By experiment for one event
 - Shows how Coll_tr behaves for all experiments



PerfExplorer - Runtime Breakdown





- ## TAU Performance System Status
- ❑ Computing platforms (selected)
 - IBM SP/pSeries/BGL/Cell PPE, SGI Altix/Origin, Cray T3E/SV-1/X1/XT3, HP (Compaq) SC (Tru64), Sun, Linux clusters (IA-32/64, Alpha, PPC, PA-RISC, Power, Opteron), Apple (G4/5, OS X), Hitachi SR8000, NEC SX Series, Windows ...
 - ❑ Programming languages
 - C, C++, Fortran 77/90/95, HPF, Java, Python
 - ❑ Thread libraries (selected)
 - pthreads, OpenMP, SGI sproc, Java, Windows, Charm++
 - ❑ Compilers (selected)
 - Intel, PGI, GNU, Fujitsu, Sun, PathScale, SGI, Cray, IBM, HP, NEC, Absoft, Lahey, Nagware, ...
- ACTS Workshop 2007 TAU Performance System 122

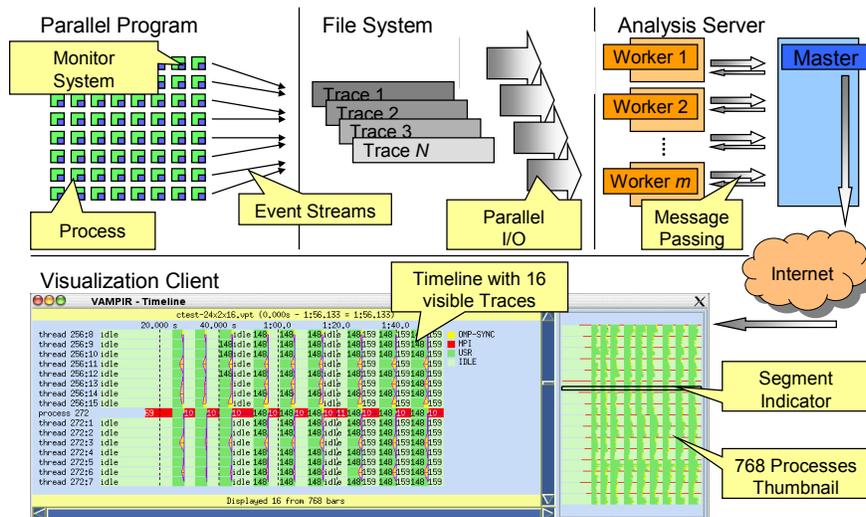
Vampir, VNG, and OTF

- ❑ Commercial trace based tools developed at ZiH, T.U. Dresden
 - Wolfgang Nagel, Holger Brunst and others...
- ❑ Vampir Trace Visualizer (aka Intel® Trace Analyzer v4.0)
 - Sequential program
- ❑ Vampir Next Generation (VNG)
 - Client (vng) runs on a desktop, server (vngd) on a cluster
 - Parallel trace analysis
 - Orders of magnitude bigger traces (more memory)
 - State of the art in parallel trace visualization
- ❑ Open Trace Format (OTF)
 - Hierarchical trace format, efficient streams based parallel access with VNGD
 - Replacement for proprietary formats such as STF
 - Tracing library available with a evaluation license now. Open source package at SC'06.

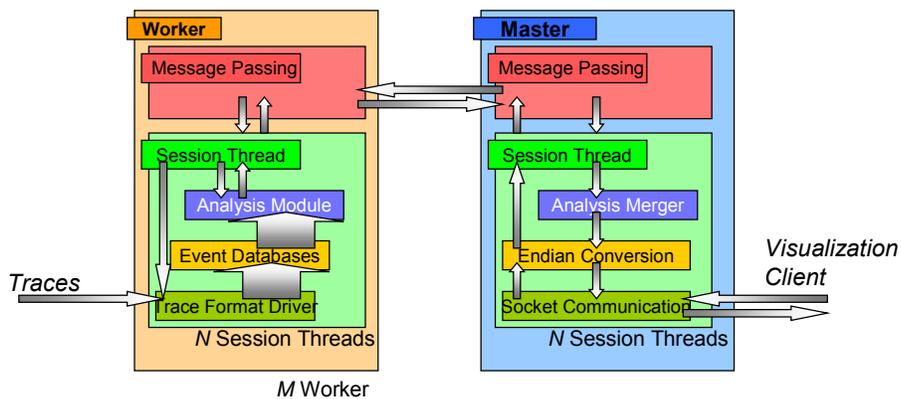
<http://www.vampir-ng.de>



Vampir Next Generation (VNG) Architecture

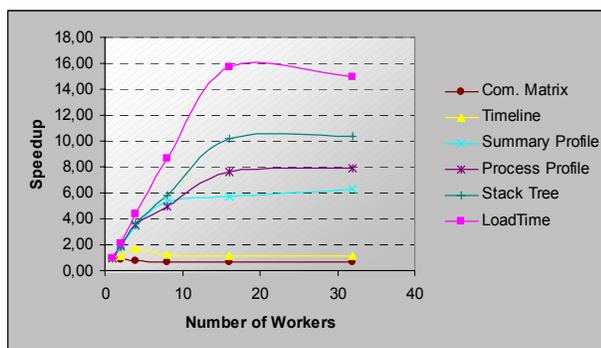


VNG Parallel Analysis Server



Scalability of VNG [Holger Brunst, WAPA 2005]

- sPPM
- 16 CPUs
- 200 MB



Number of Workers	1	2	4	8	16	32
Load Time	47.33	22.48	10.80	5.43	3.01	3.16
Timeline	0.10	0.09	0.06	0.08	0.09	0.09
Summary Profile	1.59	0.87	0.47	0.30	0.28	0.25
Process Profile	1.32	0.70	0.38	0.26	0.17	0.17
Com. Matrix	0.06	0.07	0.08	0.09	0.09	0.09
Stack Tree	2.57	1.39	0.70	0.44	0.25	0.25

VNG Analysis Server Architecture



- ❑ Implementation using MPI and Pthreads
- ❑ Client/server approach
- ❑ MPI and pthreads are available on most platforms
- ❑ Workload and data distribution among “physical” MPI processes
- ❑ Support of multiple visualization clients by using virtual sessions handled by individual threads
- ❑ Sessions are scheduled as threads

TAU Tracing Enhancements



- ❑ Configure TAU with `-TRACE -otf=<dir>` option

```
% configure -TRACE -otf=<dir> ...
```

Generates tau_merge, tau2vtf, tau2otf tools in <tau>/<arch>/bin directory

```
% tau_f90.sh app.f90 -o app
```
- ❑ Instrument and execute application

```
% mpirun -np 4 app
```
- ❑ Merge and convert trace files to OTF format

```
% tau_treemerge.pl
% tau2otf tau.trc tau.edf app.otf [-z] [-n <nstreams>]
% vampir app.otf
```

OR use VNG to analyze OTF/VTF trace files

Environment Variables



- ❑ Configure TAU with `-TRACE -otf=<dir>` option

```
% configure -TRACE -otf=<dir>
-MULTIPLECOUNTERS -papi=<dir> -mpi
-pdt=dir ...
```
- ❑ Set environment variables

```
% setenv TRACEDIR /p/gml/<login>/traces
% setenv COUNTER1 GET_TIME_OF_DAY (reqd)
% setenv COUNTER2 PAPI_FP_INS
% setenv COUNTER3 PAPI_TOT_CYC ...
```
- ❑ Execute application

```
% mpirun -np 32 ./a.out [args]

% tau_treemerge.pl
% tau2otf tau.trc tau.edf app.otf -z
```

Using VampirTrace to generate OTF traces



- ❑ Configure TAU with `-TRACE -vampirtrace=<dir>` option

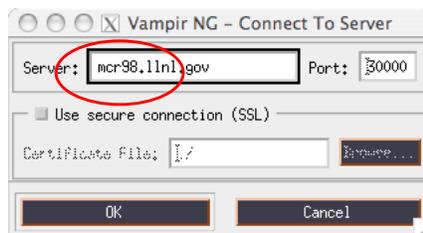
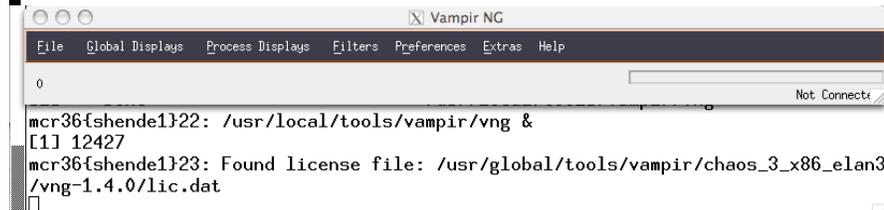
```
% configure -TRACE -vampirtrace=<dir> -papi=<dir>
-mpi
-pdt=dir ...
```
- ❑ Set environment variables

```
% setenv VT_METRICS PAPI_FP_OPS:PAPI_TOT_CYC
```
- ❑ Execute application

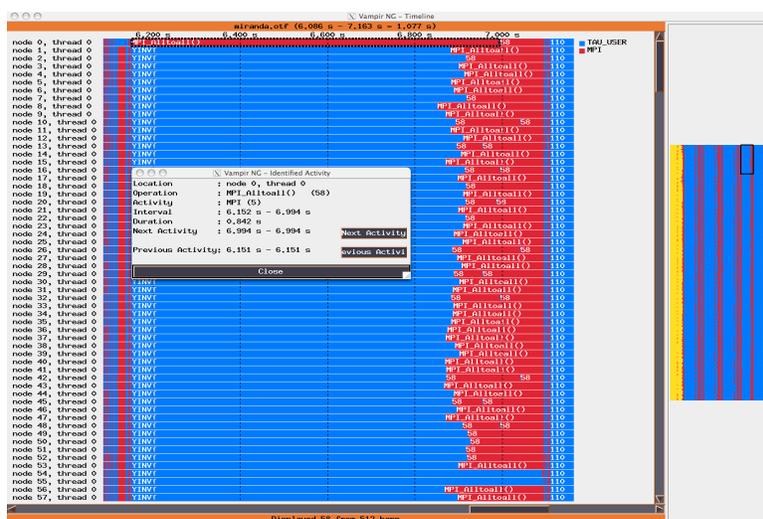
```
% mpirun -np 20 ./a.out [args]
On IBM AIX, running the application will create
a.otf after unifying the events
Unifies the descriptors to generate a.otf
% vampir a.otf &
```

Using Vampir Next Generation (VNG v1.4)

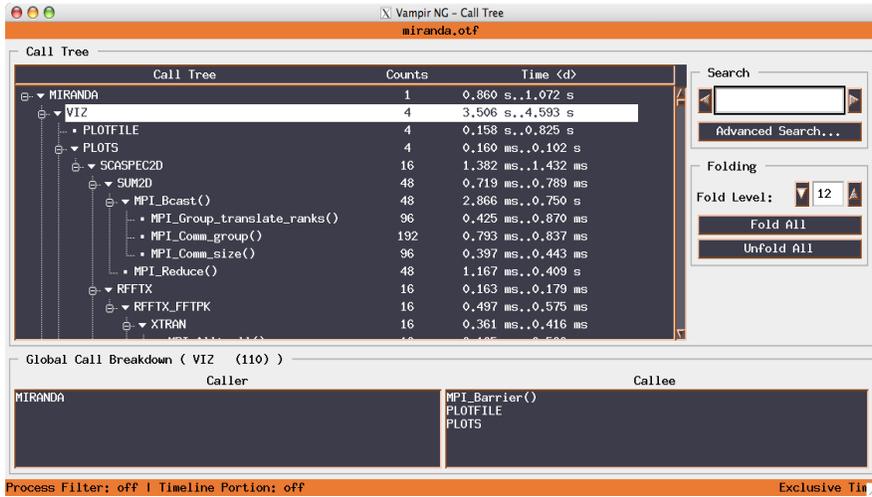
```
mcr36fshende132: srund -N2 -n4 -p pdebug /usr/local/tools/vampir/vngd
Service process resides on "mcr98"
Found license file: /usr/global/tools/vampir/chaos_3_x86_elan3/vng-1.4.0/lic.dat
running...
```



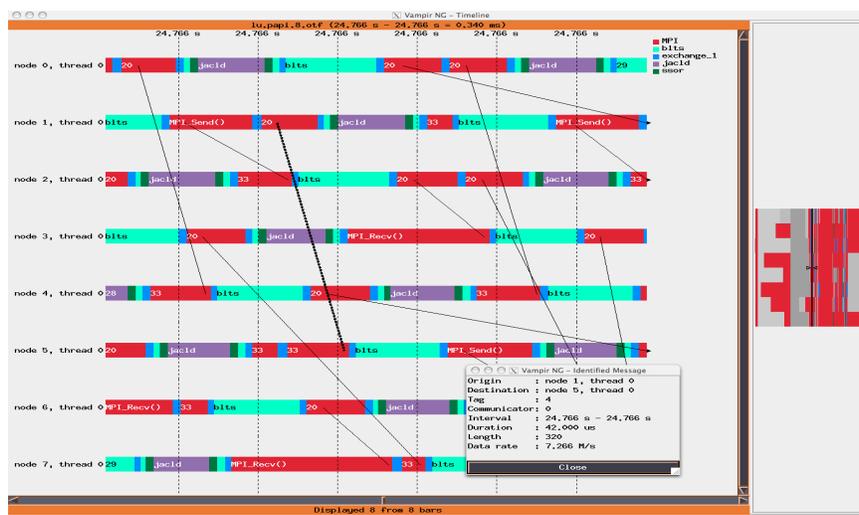
VNG Timeline Display



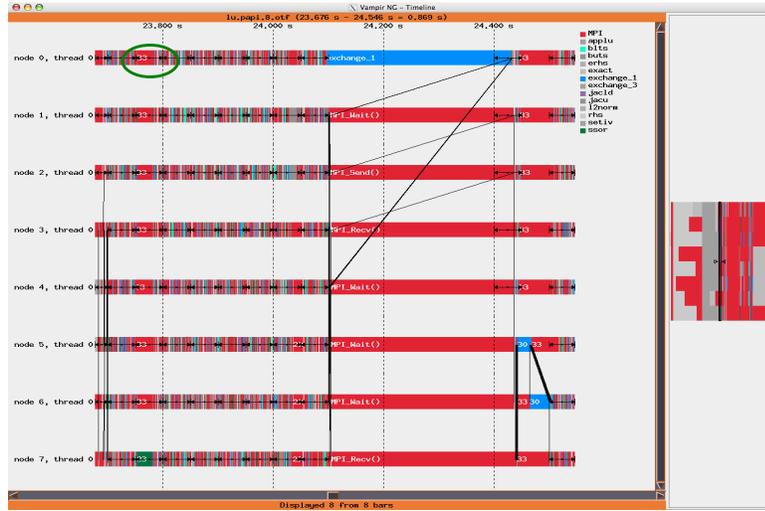
VNG Calltree Display



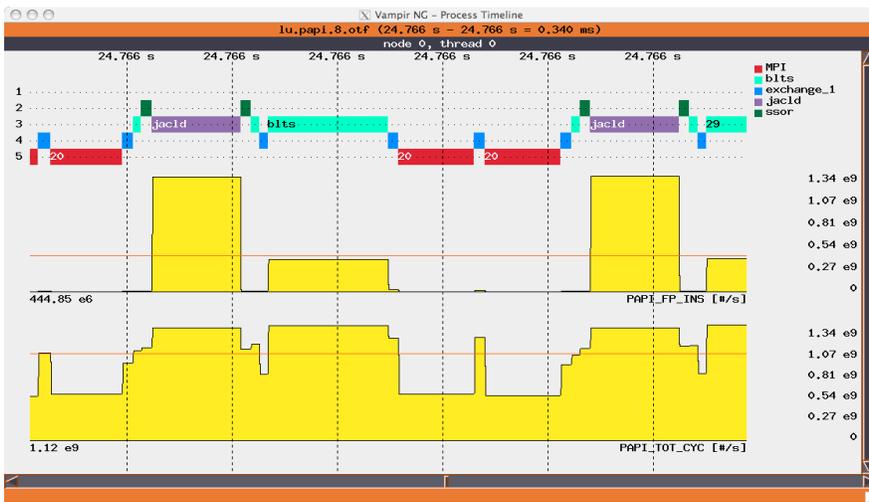
VNG Timeline Zoomed In



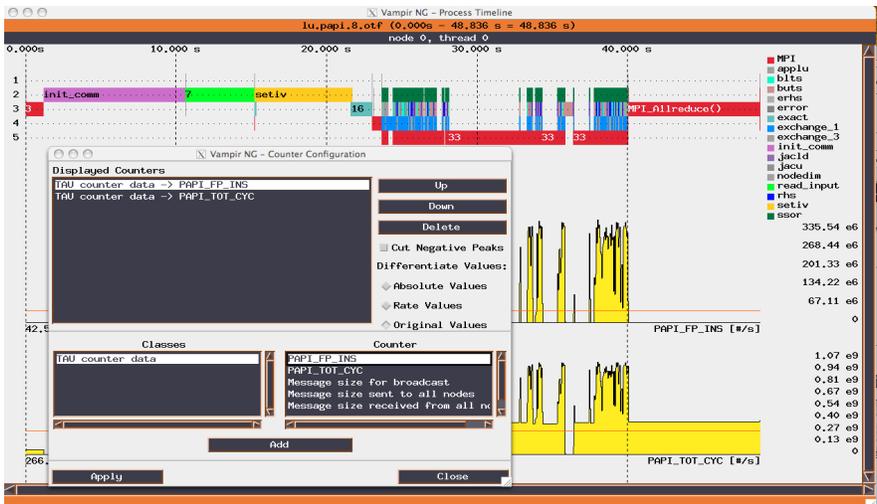
VNG Grouping of Interprocess Communications



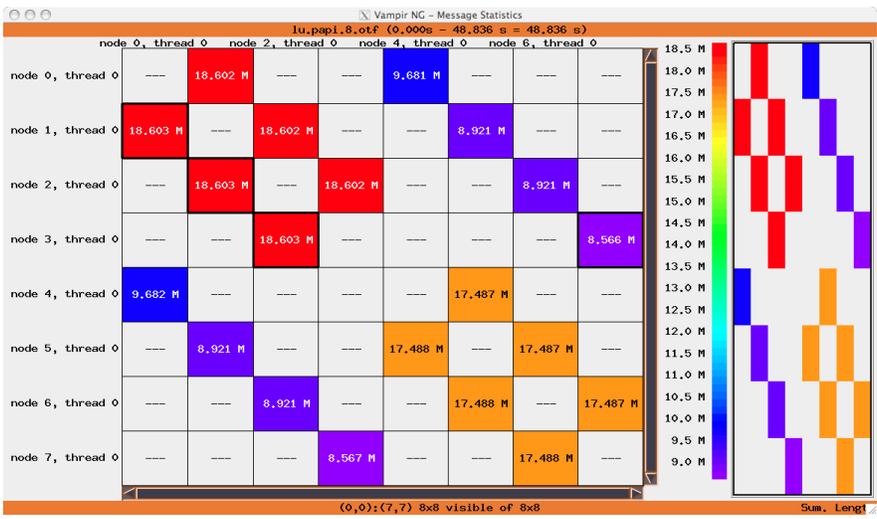
VNG Process Timeline with PAPI Counters



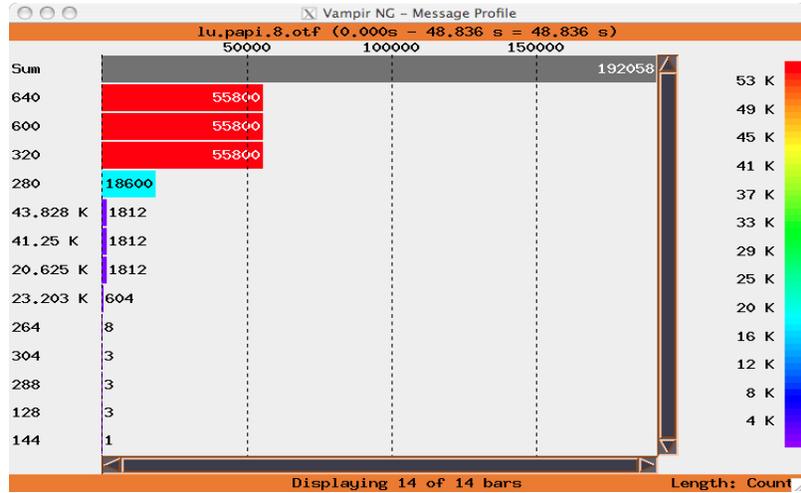
OTF/VNG Support for Counters



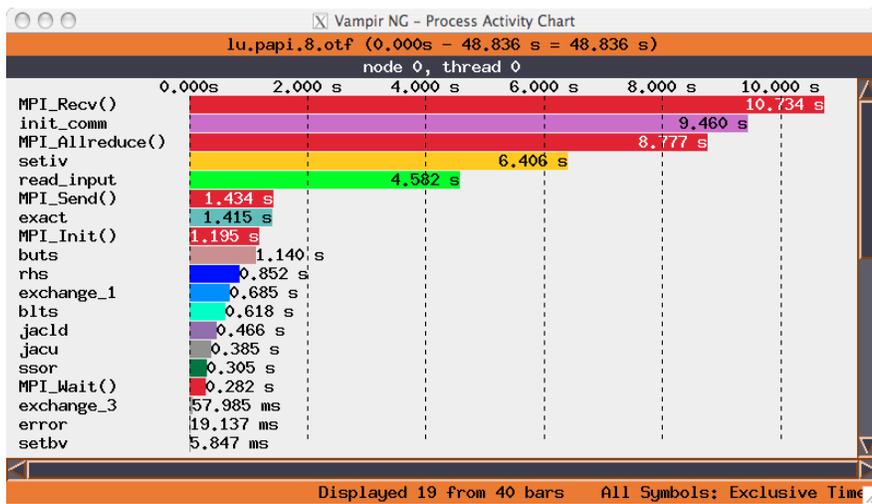
VNG Communication Matrix Display



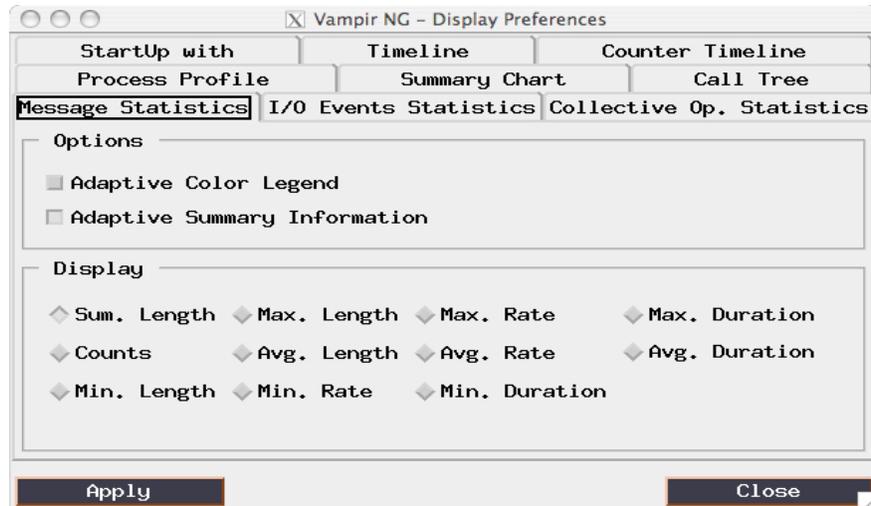
VNG Message Profile



VNG Process Activity Chart



VNG Preferences

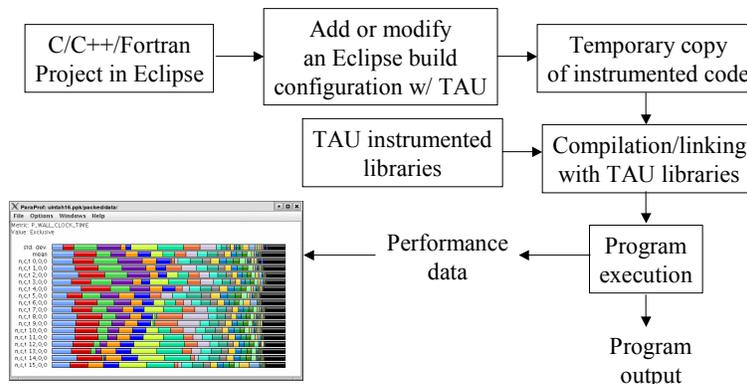


TAU Plug-Ins for Eclipse: Motivation

- ❑ High performance software development environments
 - Tools may be complicated to use
 - Interfaces and mechanisms differ between platforms / OS
- ❑ Integrated development environments
 - Consistent development environment
 - Numerous enhancements to development process
 - Standard in industrial software development
- ❑ Integrated performance analysis
 - Tools limited to single platform or programming language
 - Rarely compatible with 3rd party analysis tools
 - Little or no support for parallel projects

Adding TAU to Eclipse

- ❑ Provide an interface for configuring TAU's automatic instrumentation within Eclipse's build system
- ❑ Manage runtime configuration settings and environment variables for execution of TAU instrumented programs



TAU Eclipse Plug-In Features

- ❑ Performance data collection
 - Graphical selection of TAU stub makefiles and compiler options
 - Automatic instrumentation, compilation and execution of target C, C++ or Fortran projects
 - Selective instrumentation via source editor and source outline views
 - Full integration with the Parallel Tools Platform (PTP) parallel launch system for performance data collection from parallel jobs launched within Eclipse
- ❑ Performance data management
 - Automatically place profile output in a PerfDMF database or upload to TAU-Portal
 - Launch ParaProf on profile data collected in Eclipse, with performance counters linked back to the Eclipse source editor

TAU Eclipse Plug-In Features

The screenshot shows the Eclipse IDE with a Fortran project named 'matmult.f90'. The main editor displays the source code for a matrix multiplication routine. The left sidebar shows the project structure, and the right sidebar shows the Outline view. At the bottom, the Performance Data Manager (PerfDMF) is open, showing a tree view of performance data for the 'matmult.f90' project. An arrow points to the 'PerfDMF' label in the bottom left corner of the IDE window.

Choosing PAPI Counters with TAU's in Eclipse

The screenshot shows the Eclipse IDE's 'Profile' dialog box. The 'PAPI Counters' tab is active, displaying a list of PAPI counters and their definitions. The 'PAPI_L1_DCM' counter is selected. The 'Counter' and 'Definition' columns are visible, listing various performance metrics such as cache misses, store misses, and instruction issues.

Counter	Definition
PAPIL1_DCM	Level 1 data cache misses
PAPIL1_ICM	Level 1 instruction cache misses
PAPIL2_DCM	Level 2 data cache misses
PAPIL2_ICM	Level 2 instruction cache misses
PAPIL1_TCM	Level 1 cache misses
PAPIL2_TCM	Level 2 cache misses
PAPLFPJ_IDL	Cycles floating point units are idle
PAPLTLB_DM	Data translation lookaside buffer misses
PAPLTLB_IM	Instruction translation lookaside buffer misses
PAPLTLB_TL	Total translation lookaside buffer misses
PAPL1_LDM	Level 1 load misses
PAPL1_STM	Level 1 store misses
PAPL2_LDM	Level 2 load misses
PAPL2_STM	Level 2 store misses
PAPLSTL_JCY	Cycles with no instruction issue
PAPLHW_INT	Hardware interrupts
PAPLBR_TKN	Conditional branch instructions taken
PAPLBR_MSP	Conditional branch instructions mispredicted
PAPLTOT_INS	Instructions completed
PAPLFP_INS	Floating point instructions
PAPLBR_INS	Branch instructions
PAPLVEC_INS	Vector/SIMD instructions
PAPLRES_STL	Cycles stalled on any resource
PAPLTOT_CYC	Total cycles
PAPL1_DCH	Level 1 data cache hits
PAPL2_DCH	Level 2 data cache hits
PAPL1_DCA	Level 1 data cache accesses
PAPL2_DCA	Level 2 data cache accesses
PAPL1_DCR	Level 1 data cache reads
PAPL2_DCR	Level 2 data cache reads
PAPL1_DCW	Level 1 data cache writes
PAPL2_DCW	Level 2 data cache writes

Future Plug-In Development



- ❑ Integration of additional TAU components
 - Automatic selective instrumentation based on previous experimental results
 - Trace format conversion from within Eclipse
- ❑ Trace and profile visualization within Eclipse
- ❑ Scalability testing interface
- ❑ Additional user interface enhancements

KTAU Project

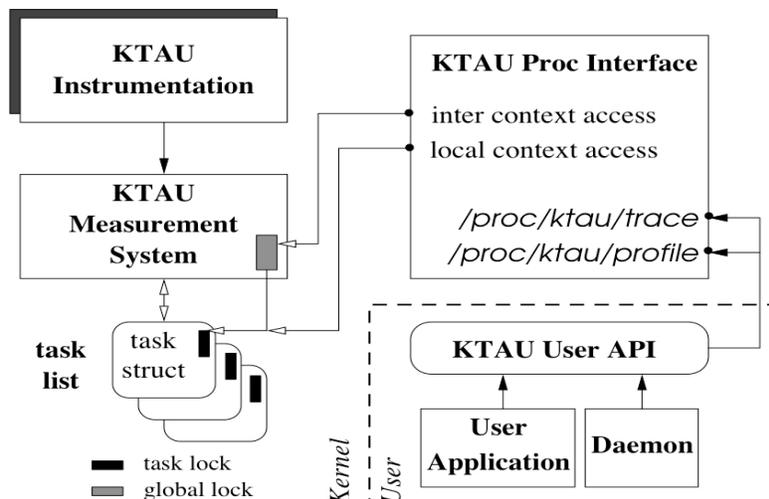


- ❑ Trend toward Extremely Large Scales
 - System-level influences are increasingly dominant performance bottleneck contributors
 - Application sensitivity at scale to the system (e.g., OS noise)
 - Complex I/O path and subsystems another example
 - Isolating system-level factors non-trivial
- ❑ OS Kernel instrumentation and measurement is important to understanding system-level influences
- ❑ But can we closely correlate observed application and OS performance?
- ❑ KTAU / TAU (Part of the ANL/UO ZeptoOS Project)
 - Integrated methodology and framework to measure whole-system performance **Z**

Applying KTAU+TAU

- ❑ How does *real* OS-noise affect *real* applications on target platforms?
 - Requires a tightly coupled performance measurement & analysis approach provided by KTAU+TAU
 - Provides an estimate of application slowdown due to Noise (and in particular, different noise-components - IRQ, scheduling, etc)
 - Can empower both application and the middleware and OS communities.
 - A. Nataraj, A. Morris, A. Malony, M. Sottile, P. Beckman, "The Ghost in the Machine : Observing the Effects of Kernel Operation on Parallel Application Performance", SC'07.
- ❑ Measuring and analyzing complex, multi-component I/O subsystems in systems like BG(L/P) (work in progress).

KTAU System Architecture



A. Nataraj, A. Malony, S. Shende, and A. Morris, "Kernel-level Measurement for Integrated Performance Views: the KTAU Project," *Cluster 2006*, distinguished paper.

TAU Performance System Status



- ❑ Computing platforms (selected)
 - IBM SP/pSeries/BGL, SGI Altix/Origin, Cray T3E/SV-1/X1/XT3, HP (Compaq) SC (Tru64), Sun, Linux clusters (IA-32/64, Alpha, PPC, PA-RISC, Power, Opteron), Apple (G4/5, OS X), Hitachi SR8000, NEC SX-5/6, SiCortex, Windows ...
- ❑ Programming languages
 - C, C++, Fortran 77/90/95, HPF, Java, Python
- ❑ Thread libraries (selected)
 - pthreads, OpenMP, SGI sproc, Java, Windows, Charm++
- ❑ Compilers (selected)
 - Intel, GNU, Fujitsu, Sun, PathScale, SGI, Cray, IBM, HP, NEC, Absoft, Lahey, Nagware

Concluding Discussion



- ❑ Performance tools must be used effectively
- ❑ More intelligent performance systems for productive use
 - Evolve to application-specific performance technology
 - Deal with scale by “full range” performance exploration
 - Autonomic and integrated tools
 - Knowledge-based and knowledge-driven process
- ❑ Performance observation methods do not necessarily need to change in a fundamental sense
 - More automatically controlled and efficiently use
- ❑ Develop next-generation tools and deliver to community
- ❑ Open source with support by ParaTools, Inc.
- ❑ <http://www.cs.uoregon.edu/research/tau>

Labs!



Lab: TAU

Lab Instructions



```
Get workshop.tar.gz on Seaborg.nersc.gov using:  
% cp /usr/common/acts/TAU/workshop.tar.gz  
Or  
% wget http://www.cs.uoregon.edu/research/tau/  
  workshop.tar.gz  
% gtar zxf workshop.tar.gz  
and follow the instructions in the README file.
```

Lab Instructions

To profile a code:

1. Load TAU module:
% module load tau
2. Change the compiler name to tau_cxx.sh, tau_f90.sh, tau_cc.sh:
F90 = tau_f90.sh
3. Choose TAU stub makefile
% setenv TAU_MAKEFILE
/usr/common/acts/TAU/2.16.5/rs6000/lib/Makefile.tau-[options]
4. If stub makefile has `-multiplecounters` in its name, set COUNTER[1-<n>] environment variables:
% setenv COUNTER1 GET_TIME_OF_DAY
% setenv COUNTER2 PAPI_FP_INS
% setenv COUNTER3 PAPI_TOT_CYC ...
5. Set TAU_THROTTLE environment variable to throttle instrumentation:
% setenv TAU_THROTTLE 1
6. Build and run workshop examples, then run pprof/paraprof

Support Acknowledgements

- US 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 Lab
- US Department of Defense (DoD)
- NSF Software and Tools for High-End Computing
- Research Centre Juelich
- TU Dresden
- Los Alamos National Laboratory
- ParaTools, Inc.

