Efficient Parallel Runtime Bounds Checking with the TAU Performance System

John C. Linford, Sameer Shende, Allen D. Malony
{jlinford, sameer, malony}@paratools.com
ParaTools, Inc., Eugene, OR

Andrew Wissink
U.S. Army Aviation Development Directorate - AFDD
NASA Ames Research Center, Moffett Field, CA

10-12 September 2013
Waltham, MA

Download slides from:

http://www.paratools.com/hpec13

Efficient Parallel Runtime Bounds Checking with the TAU Performance System

John C. Linford, Sameer Shende, Allen D. Malony
ParaTools, Inc., Eugene, OR

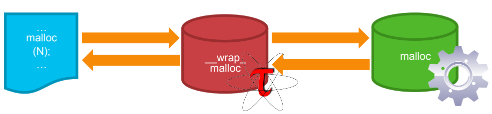
Andrew Wissink
U.S. Army Aviation Development Directorate - AFDD
NASA Ames Research Center, Moffett Field, CA

Problems Addressed

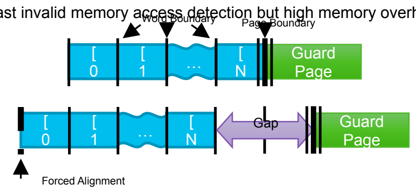
- **Debug memory errors in parallel codes written in multiple languages.**
  - When and where do memory errors occur?
  - What is the heap memory usage?
  - Was the error cause by a read or a write?
  - Which processes or threads experienced the error?
  - Were there any memory leaks in the application?
  - What were the performance characteristics prior to fa
- **Protect sensitive data.**
  - Generate reports free of application data.
  - Open and portable debugging data file format.

Approach

1. **Instrument heap memory allocation and deallocation calls.**
   - Record heap memory use, allocation and deallocation rate.
   - Associate every allocation with its use in the source code.



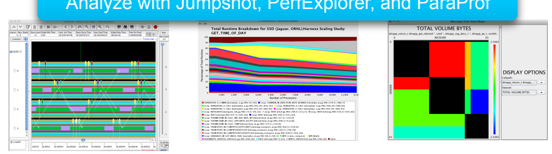
2. **Allocate guard pages before and/or after memory allocations.**
   - Use the MMU hardware to protect regions of memory from access.
   - Fast invalid memory access detection but high memory overhead.



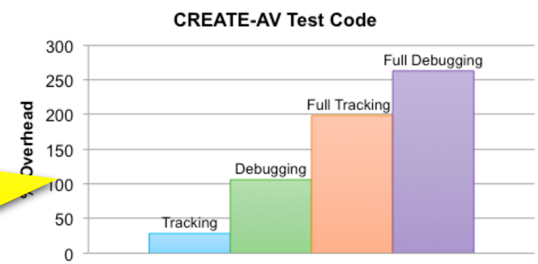
3. **Intercept error signal if error occurs.**
   - Save backtrace in the application profile.
   - Write application profile to disk and gracefully shut down.
4. **Send application profile to developers for analysis.**
   - Profile contains no application input or memory content information.

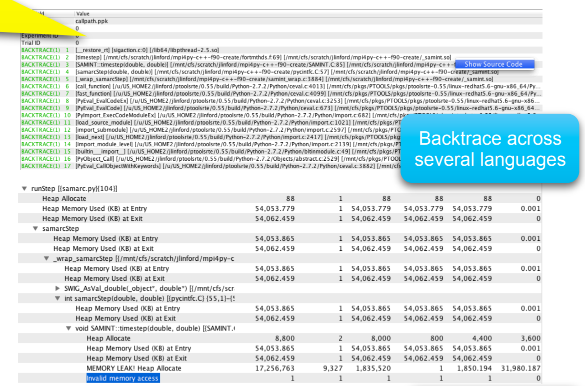
The TAU Performance System

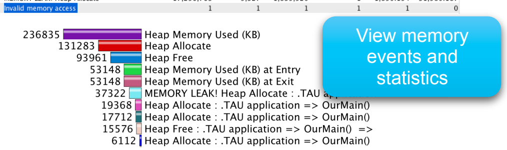
Analyze with Jumpshot, PerfExplorer, and ParaProf



Results







Related Work

- H. Aygün, D.U.M.A – Detect Unintended Memory Access, http://duma.sourceforge.net/, May 2013.
- Apple Corp., "Guard Malloc Manual Page," https://developer.apple.com/, March 2009.
- B. Perens, "efence – Electric Fence Malloc Debugger," http://linux.die.net/man/3/efence, 1999.
- J. Seward and N. Nethercote, "Using Valgrind to detect unintended value errors with bit-precision," in *USENIX ATC*, Anaheim, CA, USA, April 2005.
- K. Serebryany, D. Bruening, A. Potapenko, and D. Vyukov, "Address-sanitizer: A fast address sanity checker," in *USENIX ATC*, 2012.
- Microsoft Corp., "Gflags and PageHeap," http://msdn.microsoft.com/, July 2013.
- S. Shende, A. D. Malony, J. Linford, A. Wissink, and S. Adamec, "Isolating Runtime Faults with Callstack Debugging using TAU," in *Proceedings of the IEEE HPEC 2012 Conference*. IEEE Computer Society, 2012.

TAU Performance System®

http://tau.uoregon.edu/

- **Tuning and Analysis Utilities (18+ year project)**
- **Comprehensive performance profiling and tracing**
  - Integrated, scalable, flexible, portable
  - Targets all parallel programming/execution paradigms
- **Integrated performance toolkit**
  - Instrumentation, measurement, analysis, visualization
  - Widely-ported performance profiling / tracing system
  - Performance data management and data mining
  - Open source (BSD-style license)
- **Integrates with application frameworks**

Understanding Application Performance using TAU

- **How much time** is spent in each application routine and outer *loops*? Within loops, what is the contribution of each *statement*?
- **How many instructions** are executed in these code regions? Floating point, Level 1 and 2 *data cache misses*, hits, branches taken?
- **What is the memory usage** of the code? When and where is memory allocated/de-allocated? Are there any memory leaks?
- **What are the I/O characteristics** of the code? What is the peak read and write *bandwidth* of individual calls, total volume?
- **What is the contribution of each *phase*** of the program? What is the time wasted/spent waiting for collectives, and I/O operations in Initialization, Computation, I/O phases?
- **How does the application *scale***? What is the efficiency, runtime breakdown of performance across different core counts?

What does TAU support?

C/C++ CUDA UPC OpenCL
Python GPI
Fortran OpenACC Java MPI
pthreads
Intel MIC OpenMP
Intel GNU
LLVM PGI Cray Sun
MinGW
Linux Windows AIX
Insert yours here
BlueGene Fujitsu ARM
NVIDIA Kepler OS X

Availability on New Systems

- Intel compilers with Intel MPI on Intel Xeon Phi™ (MIC)
- GPI with Intel Linux x86_64 Infiniband clusters
- IBM BG/Q and Power 7 Linux with IBM XL UPC compilers
- NVIDIA Kepler K20 with CUDA 5.0 with NVCC
- Fujitsu Fortran/C/C++ MPI compilers on the K computer
- PGI compilers with OpenACC support on NVIDIA systems
- Cray CX30 Sandybridge Linux systems with Intel compilers
- Cray CCE compilers with OpenACC support on Cray XK7
- AMD OpenCL libs with GNU on AMD Fusion cluster systems
- MPC compilers on TGCC Curie system (Bull, Linux x86_64)
- GNU compilers on ARM Linux clusters (MontBlanc, BSC)
- Cray CCE compilers with OpenACC on Cray XK6 (K20)
- Microsoft MPI with Mingw compilers under Windows Azure
- LLVM and GNU compilers under Mac OS X, IBM BGQ

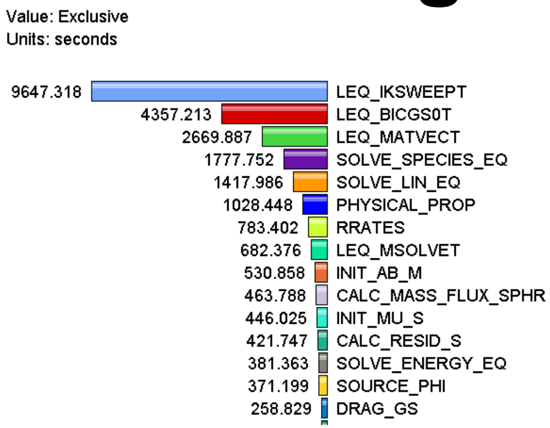
What Can TAU Do?

- **Profiling and tracing**
  - **Profiling** shows you **how much** (total) time was spent in each routine
  - **Tracing** shows you **when** the events take place on a timeline
- **Multi-language debugging**
  - Identify the source location of a crash by unwinding the system callstack
  - Identify memory errors (off-by-one, etc.)

- Profiling and tracing can measure **time** as well as **hardware performance counters** (cache misses, instructions) from your CPU
- TAU can **automatically instrument** your source code using a package called PDT for routines, loops, I/O, memory, phases, etc.
- TAU runs on **all HPC platforms** and it is free (BSD style license)
- TAU includes instrumentation, measurement and analysis tools

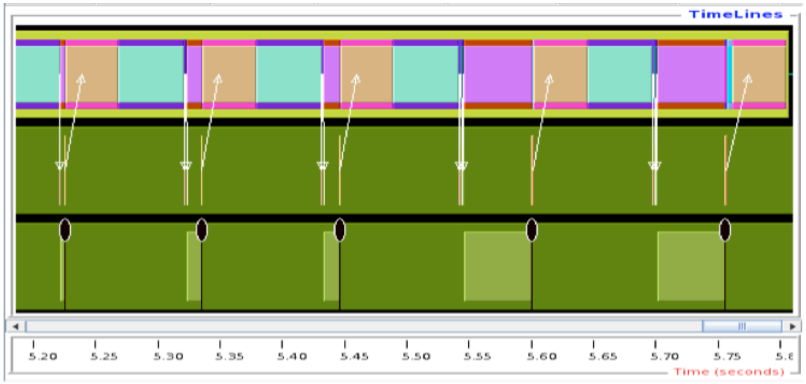
Profiling and Tracing

Profiling



- **Profiling** shows you **how much** (total) time was spent in each routine

Tracing

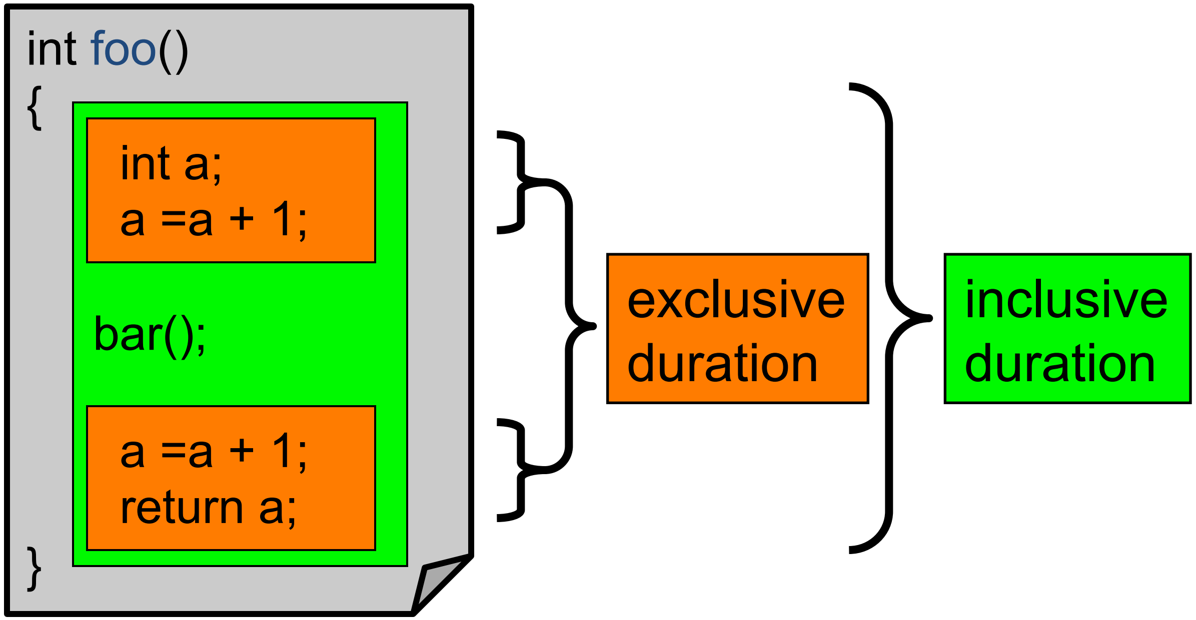


- **Tracing** shows you **when** the events take place on a timeline

- Metrics can be time or hardware performance counters (cache misses, instructions)
- TAU can automatically instrument your source code using a package called PDT for routines, loops, I/O, memory, phases, etc.

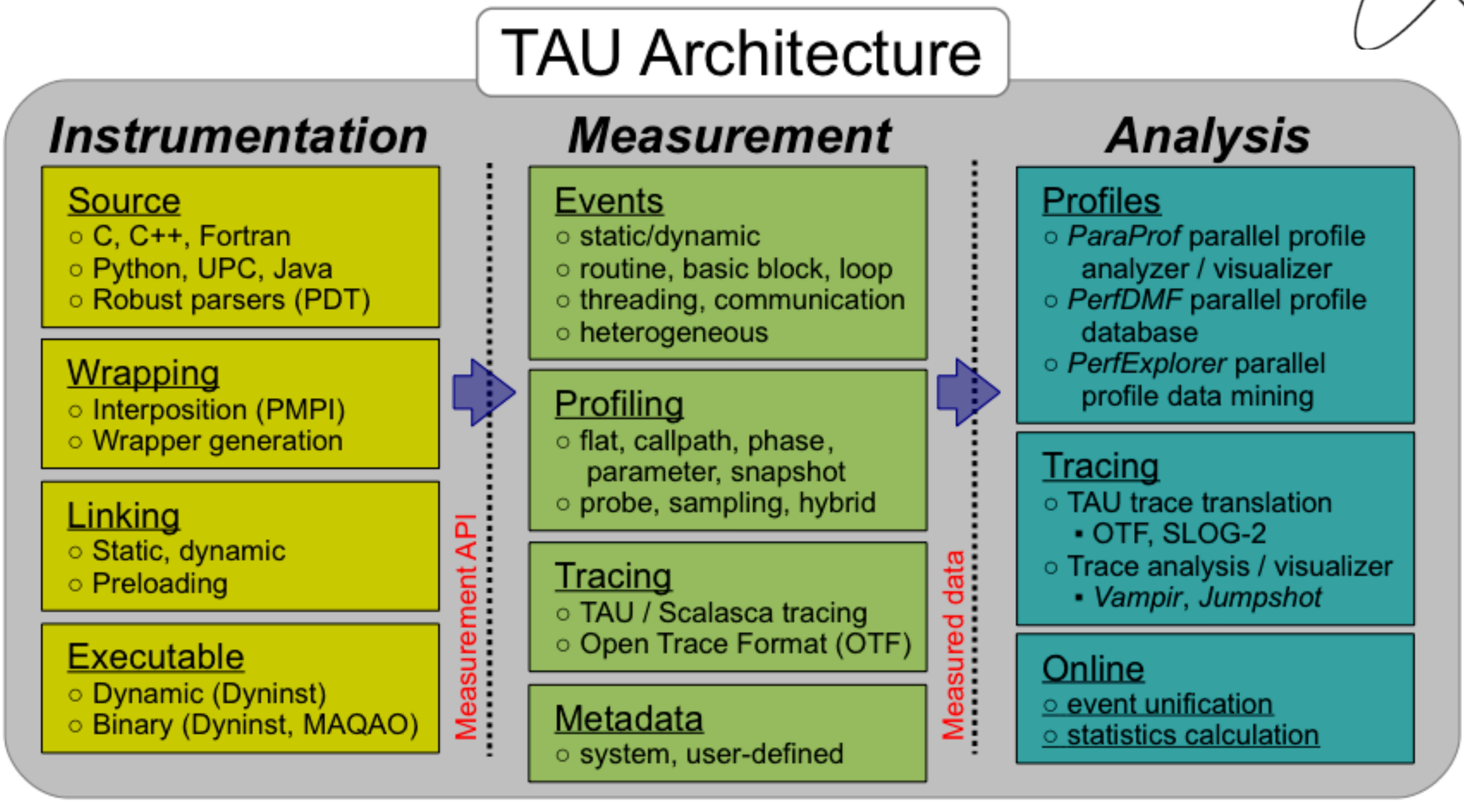
Inclusive vs. Exclusive Measurements

- Performance with respect to code regions
- Exclusive measurements for region only
- Inclusive measurements includes child regions



ParaTools

TAU Architecture and Workflow



TAU Architecture and Workflow

Instrumentation: Add probes to perform measurements

- Source code instrumentation using pre-processors and compiler scripts
- Wrapping external libraries (I/O, MPI, Memory, CUDA, OpenCL, pthread)
- Rewriting the binary executable

Measurement: Profiling or tracing using various metrics

- Direct instrumentation (Interval events measure exclusive or inclusive duration)
- Indirect instrumentation (Sampling measures statement level contribution)
- Throttling and runtime control of low-level events that execute frequently
- Per-thread storage of performance data
- Interface with external packages (e.g. PAPI hw performance counter library)

Analysis: Visualization of profiles and traces

- 3D visualization of profile data in paraprof or perfexplorer tools
- Trace conversion & display in external visualizers (Vampir, Jumpshot, ParaVer)

Instrumentation

Direct and indirect performance observation

- Instrumentation invokes performance measurement
- Direct measurement with *probes*
- Indirect measurement with periodic sampling or hardware performance counter overflow interrupts
- Events measure performance data, metadata, context, etc.

User-defined events

- ***Interval*** (start/stop) events to measure exclusive & inclusive duration
- ***Atomic events*** take measurements at a single point
  - Measures total, samples, min/max/mean/std. deviation statistics
- ***Context events*** are atomic events with executing context
  - Measures above statistics for a given calling path

Direct Observation Events

Interval events (begin/end events)

- Measures exclusive & inclusive durations between events
- Metrics monotonically increase
- Example: Wall-clock timer

Atomic events (trigger with data value)

- Used to capture performance data state
- Shows extent of variation of triggered values (min/max/mean)
- Example: heap memory consumed at a particular point

Code events

- Routines, classes, templates
- Statement-level blocks, loops
- Example: for-loop begin/end

Interval and Atomic Events in TAU

```
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 events show ***duration***

Atomic events (triggered with value) show ***extent of variation*** (min/max/mean)

% export TAU_CALLPATH_DEPTH=0
% export 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 events

Context events are atomic events with executing context

% export TAU_CALLPATH_DEPTH=1
% export TAU_TRACK_HEAP=1

Controls depth of executing context shown in profiles

ParaTools

Context Events with Callpath

```
xterm
NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.357        1,114           1          44    1114040 int main(int, char **) C
 92.6        1,031        1,031           1           0    1031066 MPI_Init()
  6.7           72           74          40         320       1865 void func(int, int) C
  0.7            8            8           1           0       8002 MPI_Finalize()
  0.1            1            1         120           0         12 MPI_Recv()
  0.1        0.608        0.608          40           0         15 MPI_Barrier()
  0.0        0.136        0.136         120           0          1 MPI_Send()
  0.0        0.095        0.095          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
---------------------------------------------------------------------------------------
       365  5.139E+04      44.39  3.091E+04  1.234E+04  Heap Memory Used (KB) : Entry
         1      44.39      44.39      44.39          0  Heap Memory Used (KB) : Entry : int main(int, char **) C
         1       2068       2068       2068          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Comm_rank()
         1       2066       2066       2066          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Comm_size()
         1  5.139E+04  5.139E+04  5.139E+04          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Finalize()
         1      57.58      57.58      57.58          0  Heap Memory Used (KB) : Entry : int main(int, char **) C => MPI_Init()
        40  5.036E+04       2069  3.011E+04  1.228E+04  Heap Memory Used (KB) : Entry : int main(int, char **) C => void func(int, int) C
        40  5.139E+04       3098  3.114E+04  1.227E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Barrier()
        40  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Bcast()
       120  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Recv()
       120  5.139E+04   1.13E+04  3.134E+04  1.187E+04  Heap Memory Used (KB) : Entry : void func(int, int) C => MPI_Send()
       365  5.139E+04       2065  3.116E+04   1.21E+04  Heap Memory Used (KB) : Exit
                                                                                              3,7            1%
```

% export TAU_CALLPATH_DEPTH=2
% export TAU_TRACK_HEAP=1

Callpath shown on context events

Direct Instrumentation Options in TAU

Source Code Instrumentation

- Automatic instrumentation using pre-processor based on static analysis of source code (PDT), creating an instrumented copy
- Compiler generates instrumented object code
- Manual instrumentation

Library Level Instrumentation

- Statically or dynamically linked wrapper libraries
  - MPI, I/O, memory, etc.
- Wrapping external libraries where source is not available

Runtime pre-loading and interception of library calls

Binary Code instrumentation

- Rewrite the binary, runtime instrumentation

Virtual Machine, Interpreter, OS level instrumentation

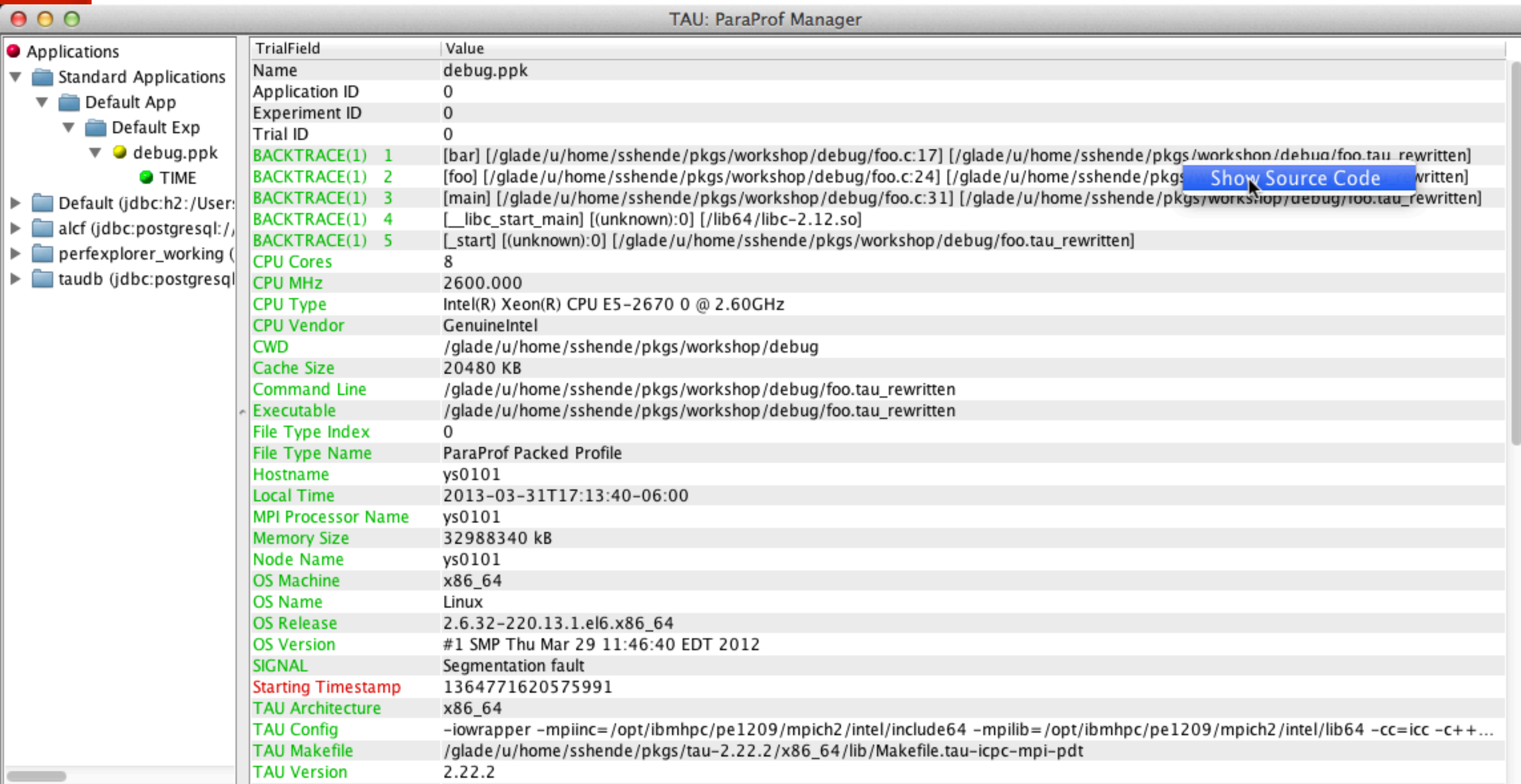
Debugging

Multi-language Application Debugging

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% export TAU_OPTIONS='-optMemDbg -optVerbose'
% make F90=tau_f90.sh CC=tau_cc.sh CXX=tau_cxx.sh
% qsub -IX -l walltime=00:30:00 -l select=1:mpiprocs=16:ncpus=16 -l
place=scatter:excl -A <your_account> -q standard

% export TAU_MEMDBG_PROTECT_ABOVE=1
% export TAU_MEMDBG_PROTECT_BELOW=1
% export TAU_MEMDBG_PROTECT_FREE=1
% mpirun -np 16 ./matmult
% paraprof
```

Multi-language Application Debugging



Location of segmentation violation

TAU: ParaProf: Source Browser: /glade/u/home/sshe...

File Help

```c
#include <stdio.h>
#include <mpi.h>
#include <unistd.h>
#include <stdlib.h>

struct node {
  int id;
  struct node *next;
};

int bar(int x) {
  int y;
  struct node *t = (struct node *)malloc(sizeof(struct node));
  t->next = NULL;
  t->id = x;
  printf("t -> id = %d\n", t->id);
  y = t->next->id;
  printf("y=%d\n",y);
  return x;
}

int foo(int x) {
  printf("foo: x = %d\n", x);
  bar(x);
  return x;
}

int main(int argc, char **argv) {
  int ret;
  MPI_Init(&argc, &argv);
  ret = foo(29);
  MPI_Finalize();
  return ret;
}
```

Memory Leak Detection

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-icpc-papi-mpi-pdt
% export PATH=$TAU/../bin:$PATH
% export TAU_OPTIONS='-optMemDbg -optVerbose'
% make F90=tau_f90.sh CC=tau_cc.sh CXX=tau_cxx.sh
% qsub -IX -l walltime=00:30:00 -l
select=1:mpiprocs=16:ncpus=16 -l place=scatter:excl -A
<your_account> -q standard

% export TAU_TRACK_MEMORY_LEAKS=1
% mpirun -np 16 ./matmult
% paraprof
```

Multi-language Memory Leak Detection

TAU: ParaProf: Context Events for: node 0 – memleak.ppk

Name △	Total	NumSamples	MaxValue	MinValue	MeanValue	Std. Dev.
Heap Allocate	5,000,033	2	5,000,001	32	2,500,016.5	2,499,984.5
Heap Allocate <file=simple.c, line=15>	180	3	80	48	60	14.236
Heap Allocate <file=simple.c, line=23>	180	1	180	180	180	0
Heap Free <file=simple.c, line=18>	80	1	80	80	80	0
Heap Free <file=simple.c, line=25>	180	1	180	180	180	0
Heap Memory Used (KB)	4,884.829	8	4,883.196	0.047	610.604	1,614.888
▼ int foo(int) C [{simple.c} {36,1}-{44,1}]						
▼ int bar(int) C [{simple.c} {7,1}-{28,1}]						
Heap Allocate <file=simple.c, line=23>	180	1	180	180	180	0
Heap Free <file=simple.c, line=25>	180	1	180	180	180	0
▼ int g(int) C [{simple.c} {30,1}-{34,1}]						
▼ int bar(int) C [{simple.c} {7,1}-{28,1}]						
Heap Allocate <file=simple.c, line=15>	180	3	80	48	60	14.236
Heap Free <file=simple.c, line=18>	80	1	80	80	80	0
MEMORY LEAK! Heap Allocate <file=simple.c, line=15>	100	2	52	48	50	2
▼ int main(int, char **) C [{simple.c} {45,1}-{55,1}]						
▼ MPI_Finalize()						
Heap Allocate	5,000,033	2	5,000,001	32	2,500,016.5	2,499,984.5
MEMORY LEAK! Heap Allocate	5,000,033	2	5,000,001	32	2,500,016.5	2,499,984.5

ParaTools

Support Acknowledgments

US Department of Energy (DOE)

- Office of Science contracts
- SciDAC, LBL contracts
- LLNL-LANL-SNL ASC/NNSA contract
- Battelle, PNNL contract
- ANL, ORNL contract

Department of Defense (DoD)

- PETTT, HPCMP

National Science Foundation (NSF)

- Glassbox, SI-2

University of Tennessee, Knoxville

T.U. Dresden, GWT

Juelich Supercomputing Center

Download TAU from U. Oregon

http://tau.uoregon.edu

http://www.hpclinux.com [LiveDVD]

Free download, open source, BSD license