Profiling with TAU

Le Yan

User Services

HPC @ LSU

Three Steps of Code Development

- Debugging
  - Make sure the code runs and yields correct results
- Profiling
  - Analyze the code to identify performance bottlenecks
- Optimization
  - Make the code run faster and/or consume less resources

Profiling

- Gather performance statistics during execution
  - Inclusive and exclusive time
  - Number of calls
- Reflects performance behavior of program entities
  - Routines
  - Loops
- Implemented through
  - Sampling: OS interrupts or hardware counters
  - Instrumentation: measurement functions

Profiling with gprof (1)

- gprof is a GNU profiling tool
- How to use
  - Compile the code with "-pg" option
  - Run the program as normal
  - Examine the profile with "`gprof <executable>`"
    - Or "`gprof <executable> <output> | less`"

Profiling with gprof (2)

```
Flat profile:
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
100.00      5.71     5.71        1     5.71     5.71  laplace_
  0.00      5.71     0.00        2     0.00     0.00  initialize_
  0.00      5.71     0.00        2     0.00     0.00  set_bcs_
…
 Call graph (explanation follows)
granularity: each sample hit covers 2 byte(s) for 0.18% of 5.71 seconds
index % time    self  children    called     name
                0.00    5.71       1/1           main [3]
[1]    100.0    0.00    5.71       1         MAIN__ [1]
                5.71    0.00       1/1           laplace_ [2]
-----------------------------------------------
                5.71    0.00       1/1           MAIN__ [1]
[2]    100.0    5.71    0.00       1         laplace_ [2]
                0.00    0.00       2/2           initialize_ [4]
                0.00    0.00       2/2           set_bcs_ [5]
…
Index by function name

   [1] MAIN__                  [2] laplace_
   [4] initialize_             [5] set_bcs_
```

What is TAU

- Tuning and Analysis Utilities
  - Developed at University of Oregon
- Scalable and flexible performance analysis toolkit
  - Performance profiling and tracing utilities
  - Performance data management and data mining
  - Automatic instrumentation through Program Database Toolkit(PDT)
  - Provides an instrumentation API

Availability on LONI and LSU HPC resrouces

- Tezpur and LONI Linux clusters
  - +tau-2.18-intel-11.1-mvapich-1.1
  - +tau-2.18-intel-11.1-mvapich2-1.4
- Philip
  - +tau-2.18-intel-11.1-mpich-1.2.7p1

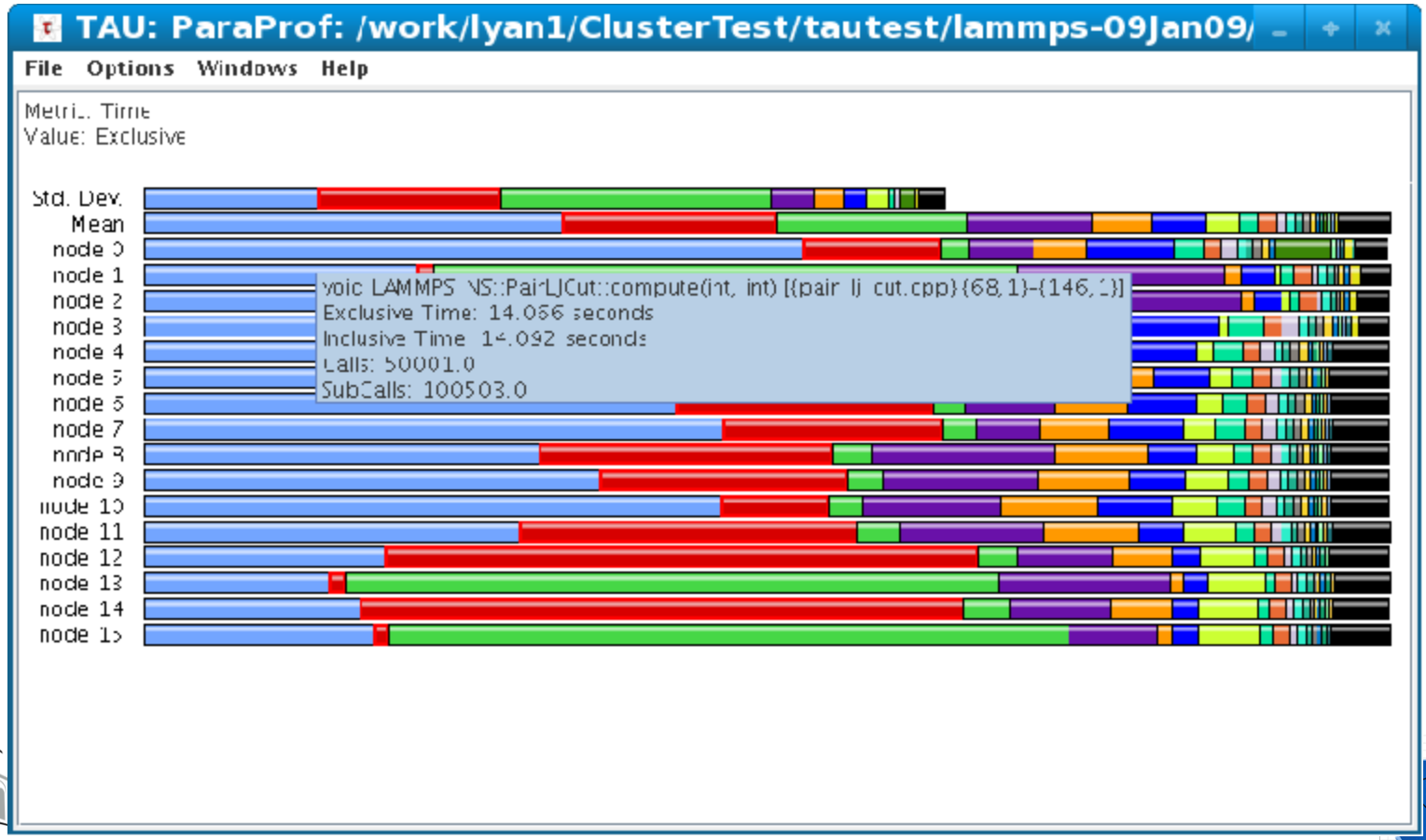
How to Use

- Add the softenv key to .soft and resoft
- Compile your code with TAU compiler scripts
  - **`tau_f90.sh`** for Fortran, **`tau_cc.sh`** for C and **`tau_cxx.sh`** for C++
  - The code is instrumented automatically
- Execute the generated executable as normal
  - Profile data files: **`profile.x.x.x`**
- Analyze/visualize the profiling results with paraprof

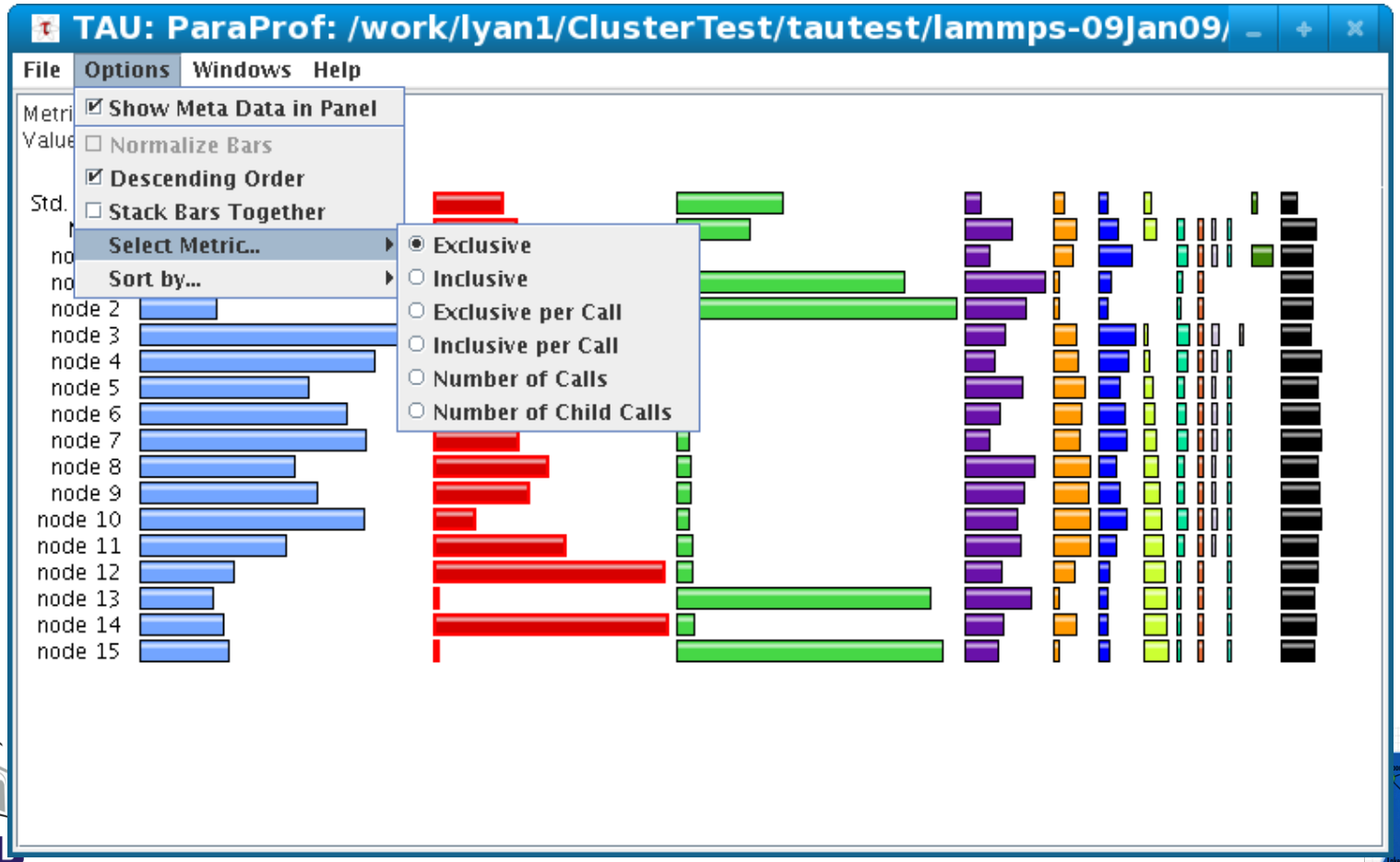
Paraprof

- Java-based analysis and visualization tool for performance data
- “pprof” is for text based profile display
- Can work with profile data generated by other profiling tools, e.g. MPIP
- Options
  - -f <file type>: specify type of performance data
  - -m: perform runtime monitoring
  - --pack <file>: pack profile data into one file

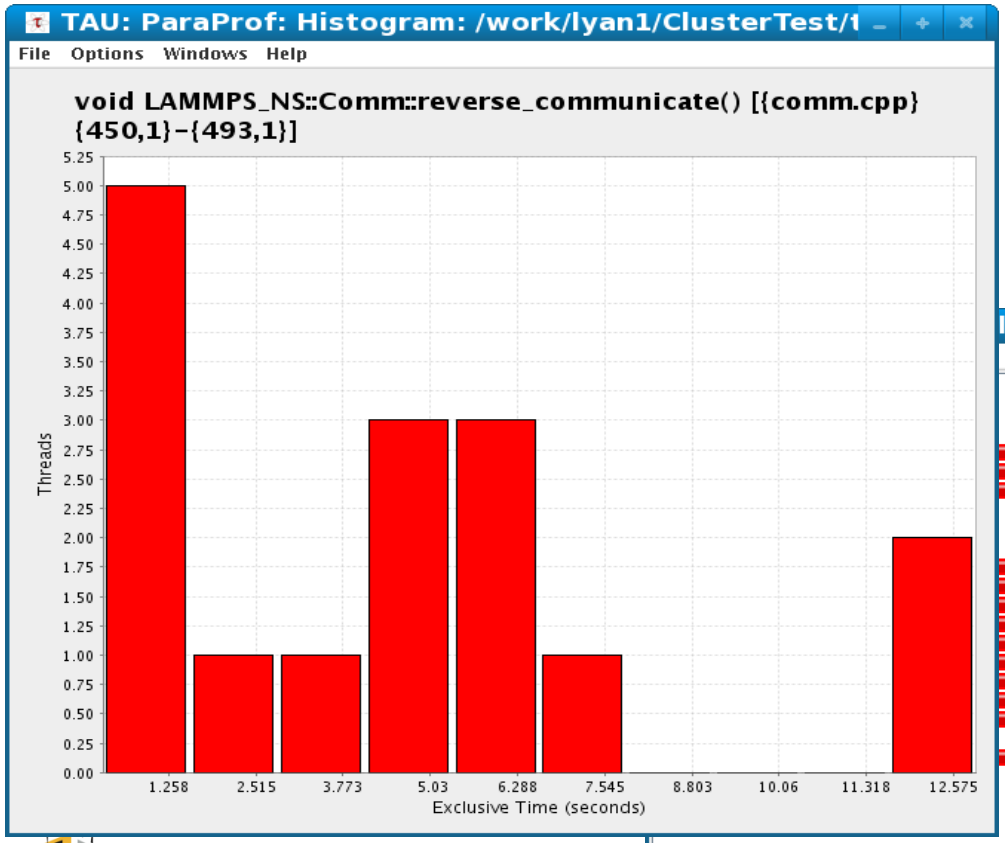
Main Data Window

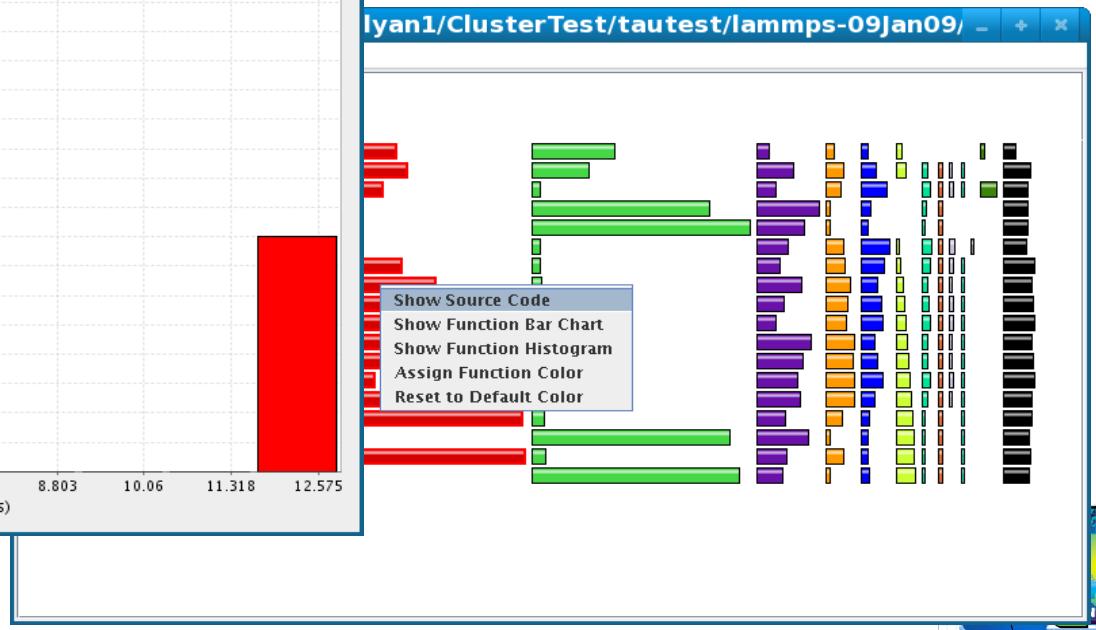


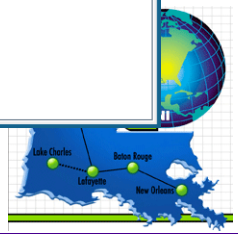
Main Data Window: Unstacked Bars



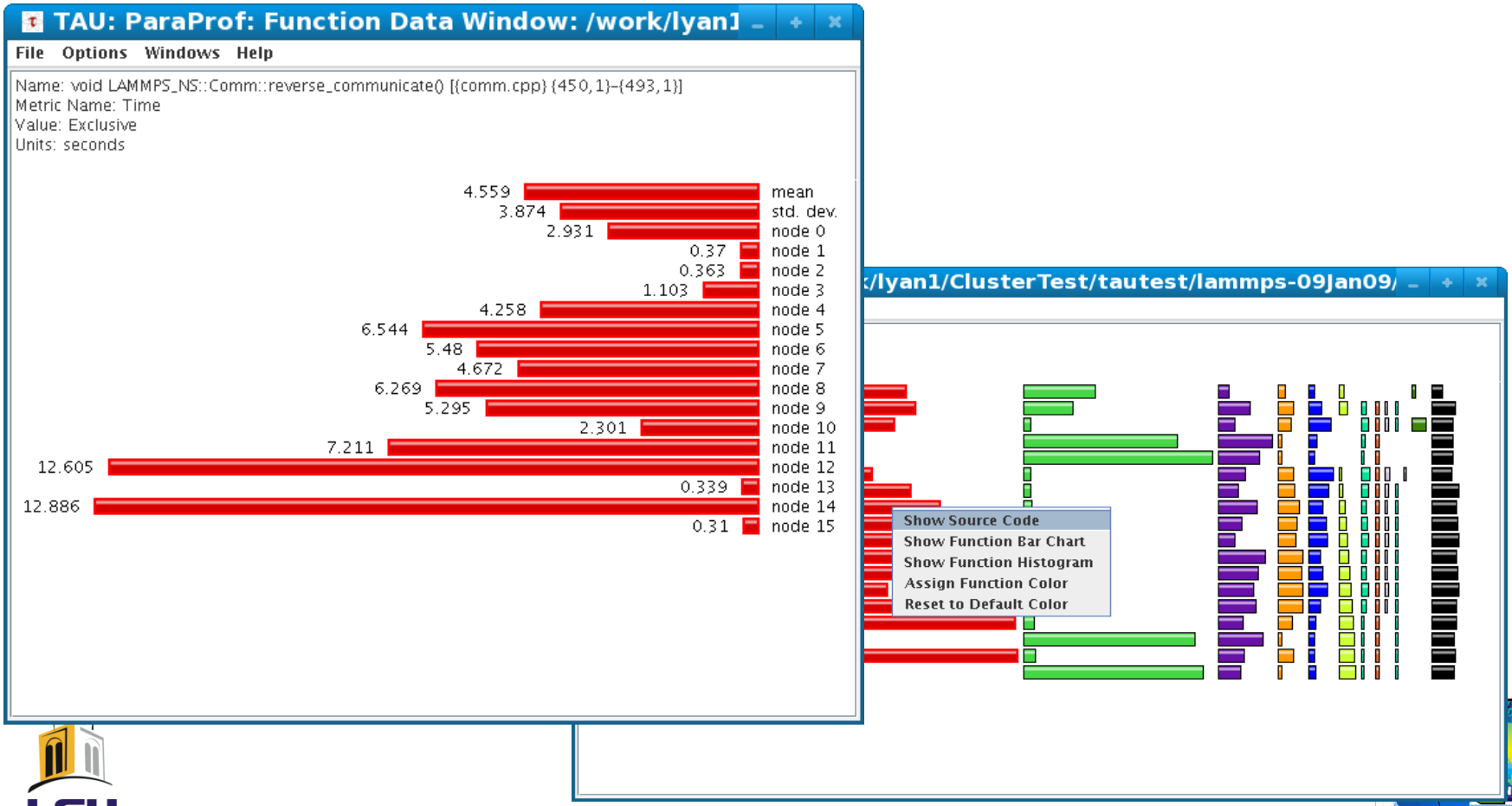
Function Data Window: Histogram



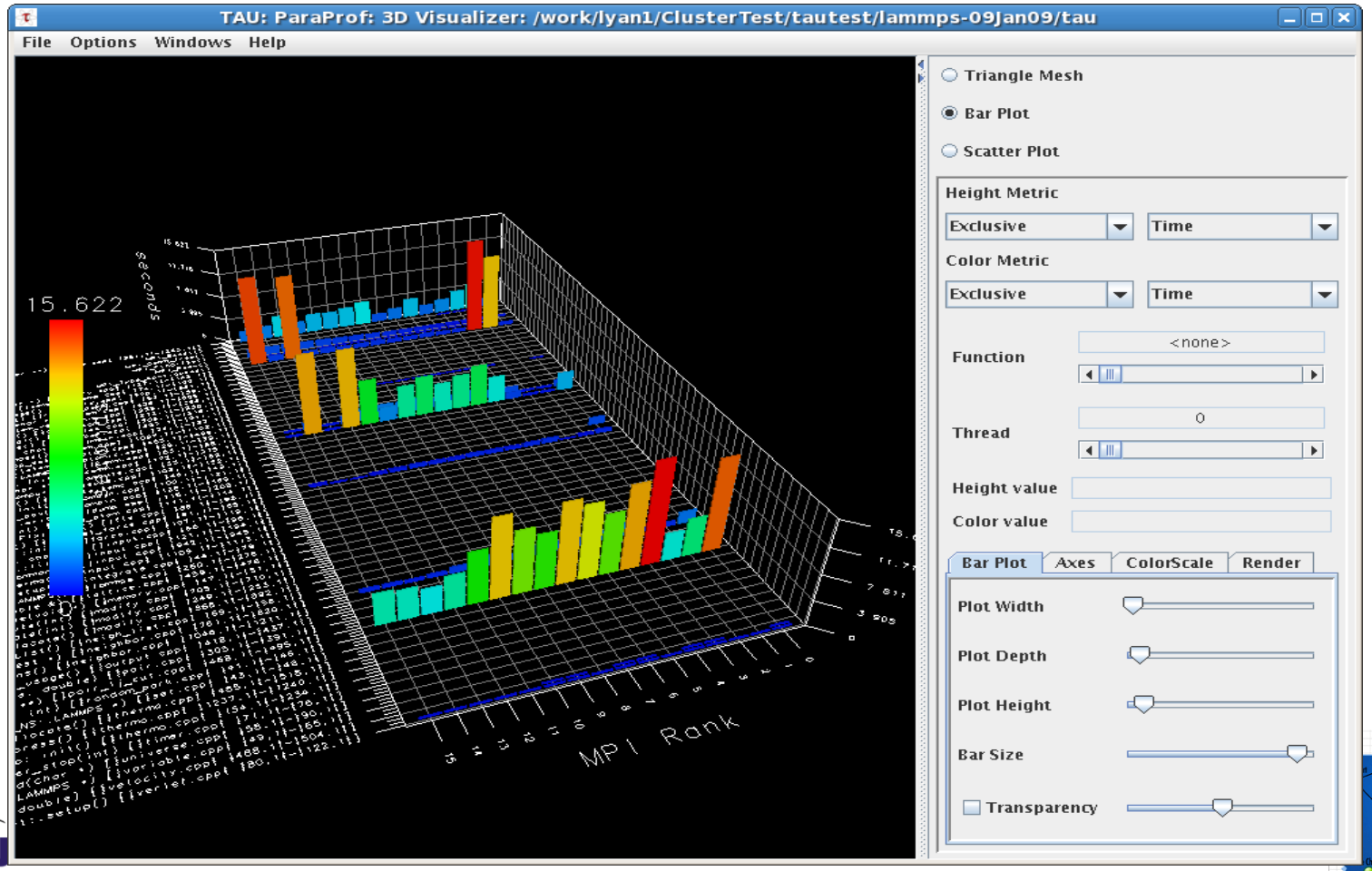




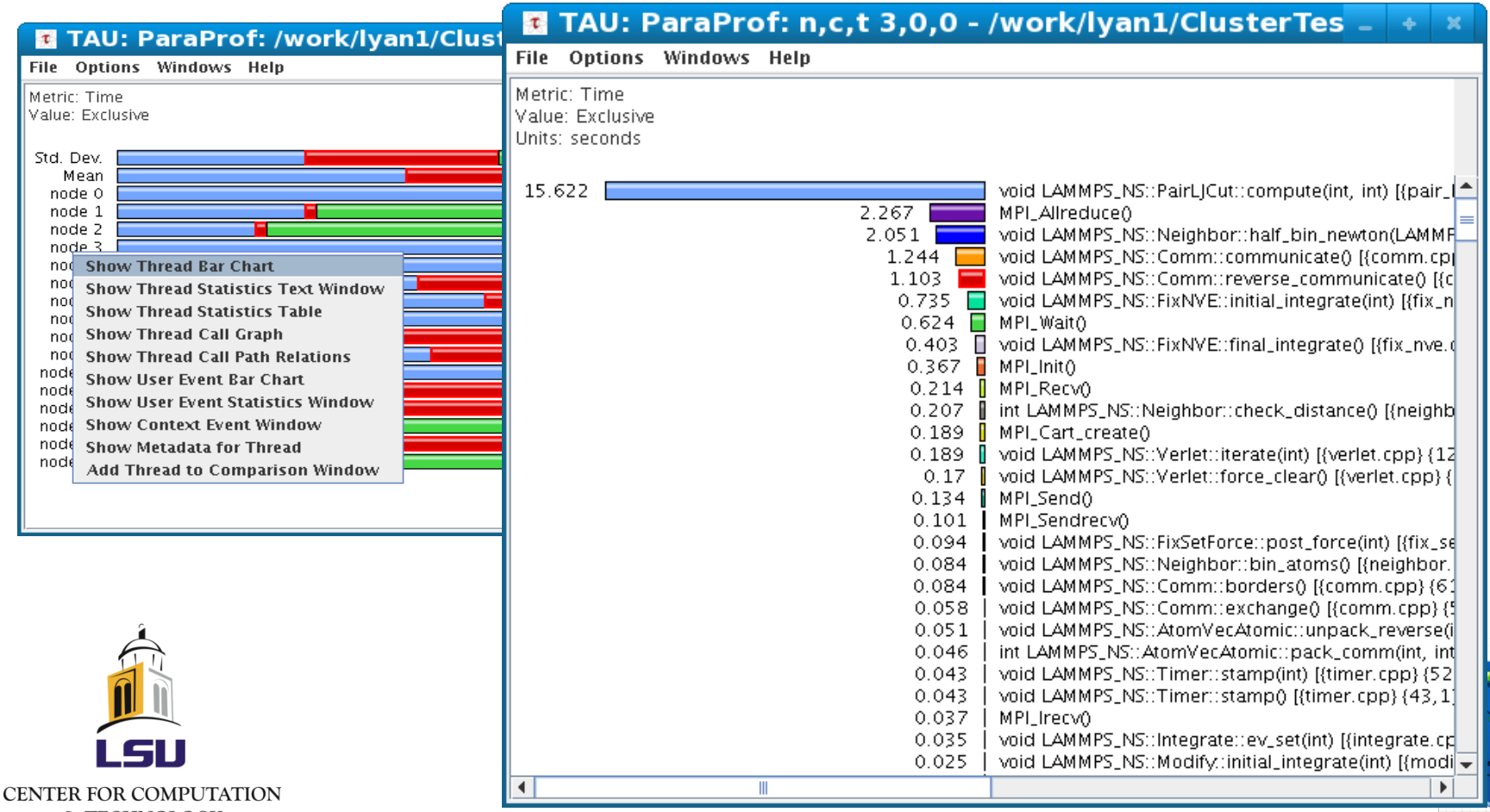
Function Data Window: Bar Chart



3D View

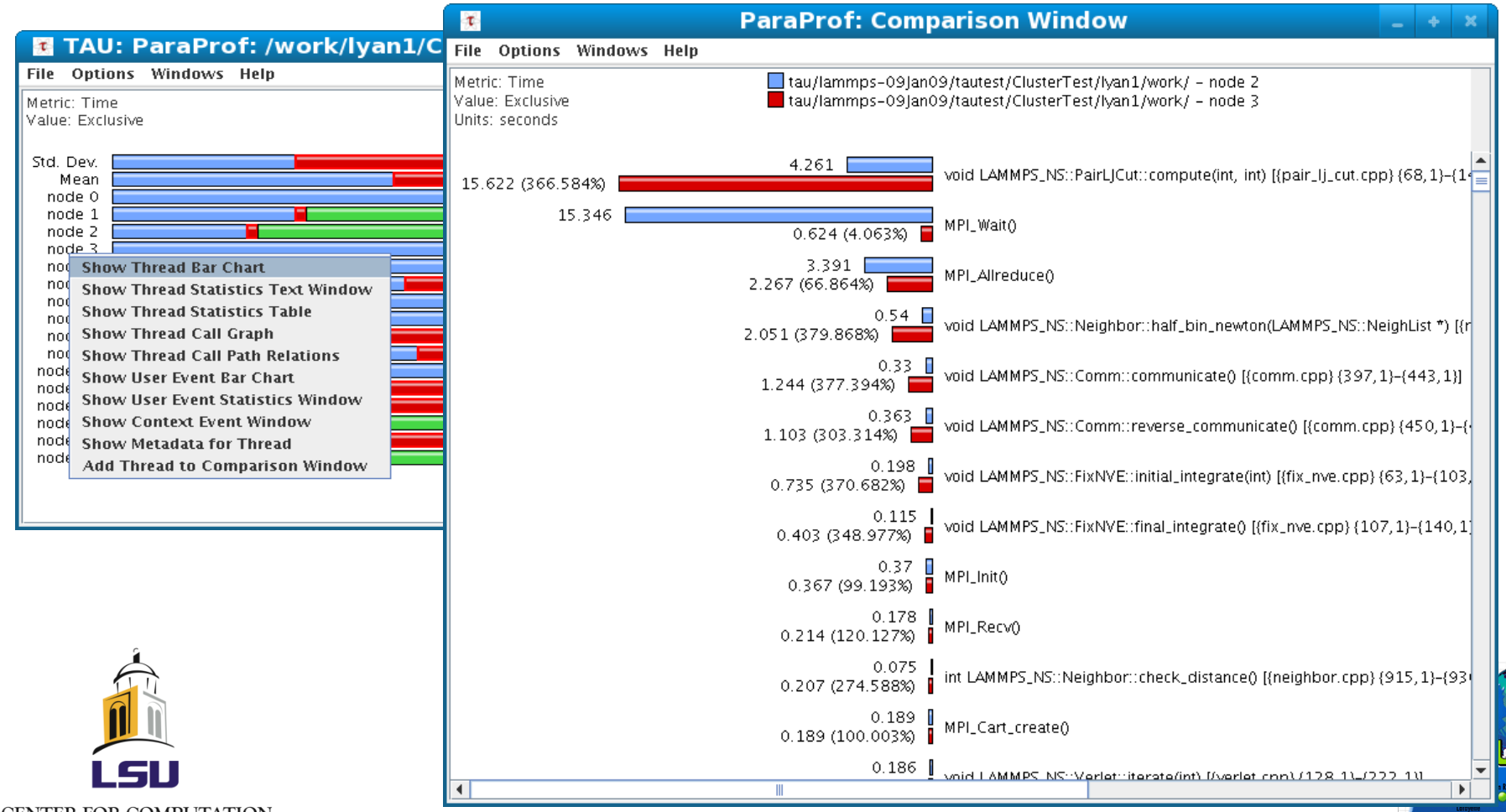

TAU: ParaProf: 3D Visualizer: /work/lyan1/ClusterTest/tautest/lammps-09Jan09/tau
File Options Windows Help
15.622
Seconds
MPI Rank
Triangle Mesh
Bar Plot
Scatter Plot
Height Metric
Exclusive
Time
Color Metric
Exclusive
Time
Function
<none>
Thread
0
Height value
Color value
Bar Plot
Axes
ColorScale
Render
Plot Width
Plot Depth
Plot Height
Bar Size
Transparency

Individual Thread View

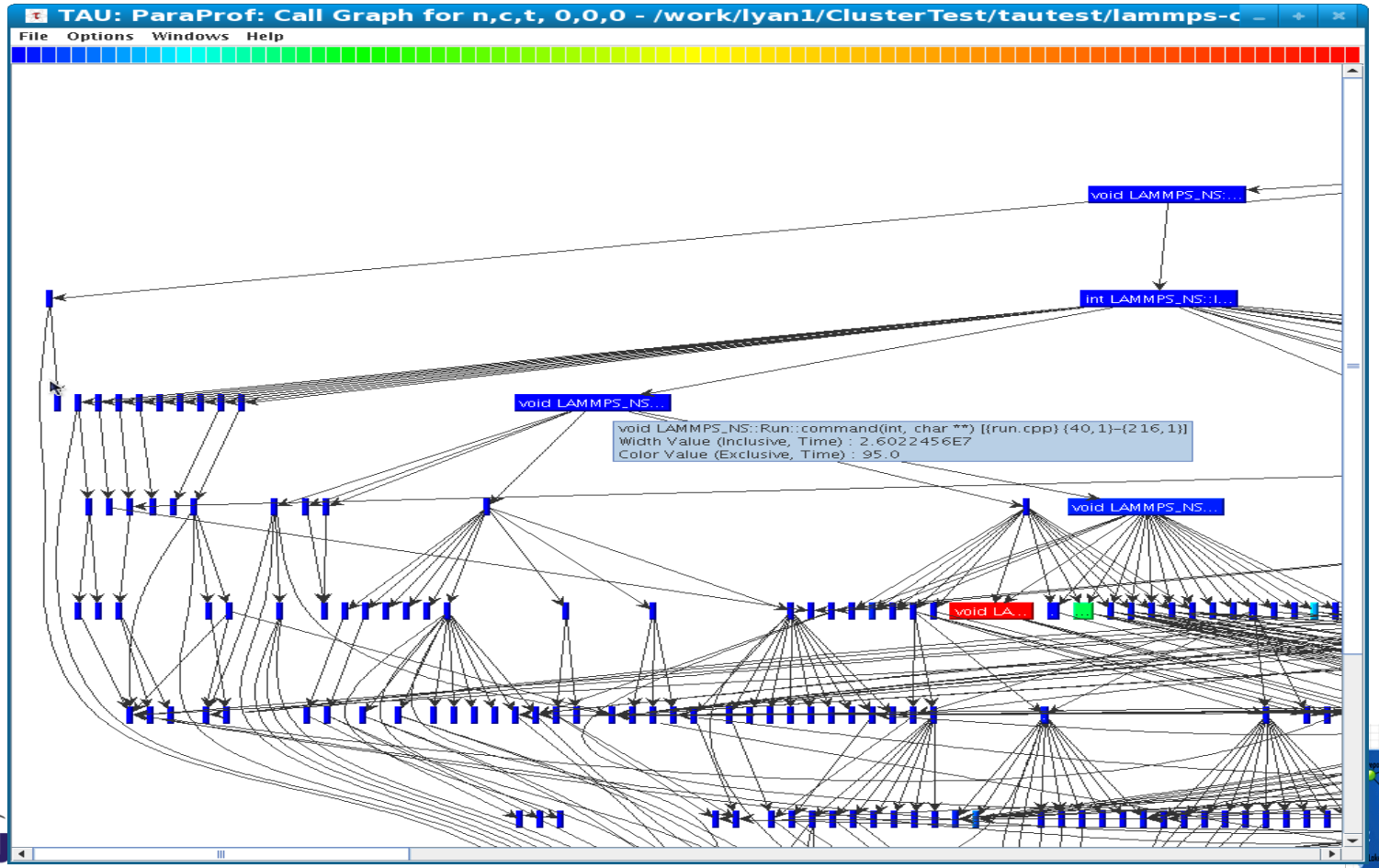


LSU
CENTER FOR COMPUTATION & TECHNOLOGY

Comparing Multiple Threads



Callpath Profile



Options for TAU Compiler Scripts

- Display available options with "`tau_xxx.sh -help`"
- Options
  - -optVerbose: display verbose debugging information
  - -optKeepFiles: keep intermediate files (instrumented source files)
  - -optDetectMemory: trace malloc/free calls

Keeping Intermediate Files (1)

```
[lyan1@poseidon2 single_file]$ ll
total 16
-rwxr-xr-x  1 lyan1 loniadmin  2163 Apr 17 09:23 mat_trans_alt.f90
-rw-r--r--  1 lyan1 loniadmin 10300 Apr 17 09:50 mat_trans_alt.o

[lyan1@poseidon2 single_file]$ tau_f90.sh -optKeepFiles mat_trans_alt.f90
...
[lyan1@poseidon2 single_file]$ ll
total 1032
-rwxr-xr-x  1 lyan1 loniadmin 1578296 Apr 17 10:18 a.out
-rwxr-xr-x  1 lyan1 loniadmin    2163 Apr 17 09:23 mat_trans_alt.f90
-rw-r--r--  1 lyan1 loniadmin    2493 Apr 17 10:18 mat_trans_alt.inst.f90
-rw-r--r--  1 lyan1 loniadmin   10300 Apr 17 10:18 mat_trans_alt.o
-rw-r--r--  1 lyan1 loniadmin    2019 Apr 17 10:18 mat_trans_alt.pdb
```

Keeping Intermediate Files (2)

```
[lyan1@poseidon2 single_file]$ cat mat_trans_alt.inst.f90
...
  ! Matrix dimension
  data ndim /16,12/
  character(len=*), parameter :: FMT1="(12(1x,i4))"
  character(len=*), parameter :: FMT2="(16(1x,i4))"


      integer profiler(2) / 0, 0 /
      save profiler

      call TAU_PROFILE_INIT()
      call TAU_PROFILE_TIMER(profiler, '
  &
     &MATRIXTRANS_ALT1 [{mat_trans_alt.f90} {1,1}-{90,28}]')
    call TAU_PROFILE_START(profiler)
      call mpi_init(ierr)

  call mpi_comm_size(mpi_comm_world,nprocs,ierr)
  call mpi_comm_rank(mpi_comm_world,myrank,ierr)
...
```

Notes for Fortran Programmers

- Use `include 'mpif.h'` instead of `use mpi`
- If free format is used with .f files, use the '`-optPdtF95Opts=-R free`' option
- If more than one module files are used, use the '`-optPdtGnuFortranParser`' option
- If C preprocessor directive are used, use the '`-optPreProcess`' option

TAU Environment Variables

- TAU provides many environment variables
  - TAU_MAKEFILE
  - TAU_THROTTLE
  - TAU_OPTIONS
  - PROFILEDIR
  - TRACEDIR
  - …

TAU_MAKEFILE

- Different TAU makefiles corresponds to different configurations
- The default is `icpc-mpi-pdt-openmp-opari`
- There are quite a few others

```
[lyan1@philip1 lib]$ ls Makefile.tau-intel-11.1-mpich-1.2.7p1-*
Makefile.tau-intel-11.1-mpich-1.2.7p1-callpath-icpc-mpi-compensate-pdt-openmp
Makefile.tau-intel-11.1-mpich-1.2.7p1-callpath-icpc-mpi-pdt-openmp
Makefile.tau-intel-11.1-mpich-1.2.7p1-depthlimit-icpc-mpi-pdt-openmp
Makefile.tau-intel-11.1-mpich-1.2.7p1-icpc-mpi-compensate-pdt-openmp
Makefile.tau-intel-11.1-mpich-1.2.7p1-icpc-mpi-pdt-openmp
Makefile.tau-intel-11.1-mpich-1.2.7p1-icpc-mpi-pdt-openmp-opari
Makefile.tau-intel-11.1-mpich-1.2.7p1-icpc-mpi-pdt-openmp-trace
Makefile.tau-intel-11.1-mpich-1.2.7p1-icpc-pdt-openmp
Makefile.tau-intel-11.1-mpich-1.2.7p1-icpc-pdt-openmp-opari
Makefile.tau-intel-11.1-mpich-1.2.7p1-icpc-pthread-pdt-openmp
Makefile.tau-intel-11.1-mpich-1.2.7p1-param-icpc-mpi-pdt-openmp
```

TAU_CALLPATH

- Enables callpath profiling
  - Recored callpath for each event
  - Need to set TAU_MAKEFILE to one of those with callpath in their names
- TAU_CALLPATH_DEPTH
  - Level to which callpath is recorded
  - Default is 2
  - Overhead increases with the depth of callpath

Other Environment Variables

- TAU_THROTTLE
  - Enable event throttling
  - Purpose: reduce profiling overhead
  - If a function executes more than $TAU_THROTTLE_NUMCALLS times and has an inclusive time per call of less than TAU_THROTTLE_PERCALLS microseconds, then profiling of that function will be disabled after the threshold is reached
- PROFILEDIR
  - Controls where the profile files are written to (the default is current directory)
- TAU_OPTIONS
  - Override the default instrumentation options

Selective Profiling (1)

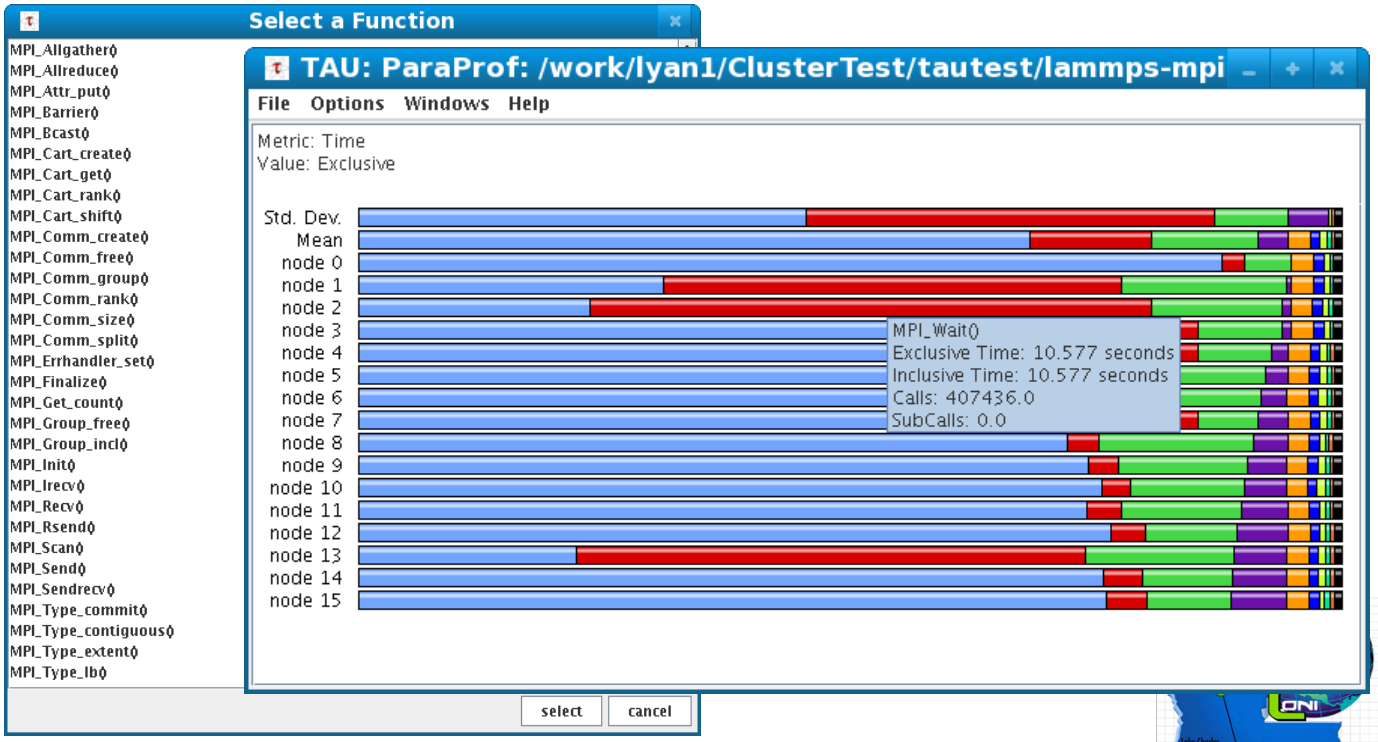
- Instruct TAU
  - Which part(s) of the code to profile
  - How they are profiled
- -optTauSelectFile=<file>
  - The select profiling file specifies files, functions and sections that will be included or excluded in the profiling
  - Wildcards can be used

Selective Profiling (2)

```
[lyan1@poseidon2 src]$ echo $TAU_OPTIONS
-optVerbose -optTauSelectFile=/work/lyan1/ClusterTest/tautest/lammps-
mpi-only/src/select.tau
[lyan1@poseidon2 src]$ cat select.tau
BEGIN_INCLUDE_LIST

MPI#
mpi#
Mpi#

END_INCLUDE_LIST
```



Tracing (1)

- Recording of information about events during execution
  - Entering/exiting code region (function, loop, block…)
  - Thread/process interactions (send/receive message…)
- Save information in event record
  - Timestamp
  - CPU identifier
  - Event type and event-specific information
- Event trace is a time-sequenced stream of event records

Tracing (2)

- Pick the correct makefile using TAU_MAKEFILE (those with “trace” in the file name”)
- Compile with TAU compiler scripts and run the program
- Use external utilities to analyze the trace files
  - JUMPSHOT
  - VAMPIR
- Be careful: trace files can grow very big!

Not Covered

- Database management
- Phase based profiles
- Track memory and I/O
- Instrumentation API

Questions?


Shreveport
Monroe
Ruston
Alexandria
LONI
Lake Charles
Baton Rouge
Lafayette
New Orleans