TAU Performance Analysis

Nicholas Chaimov
ParaTools, Inc.

ALCF Simulation, Data and Learning Workshop
October 3rd, 2018

Overview

We will cover:

–Profiling and tracing via the TAU Performance System

–Hardware performance counters (PAPI)

–Performance analysis of C/C++, Fortran, Python

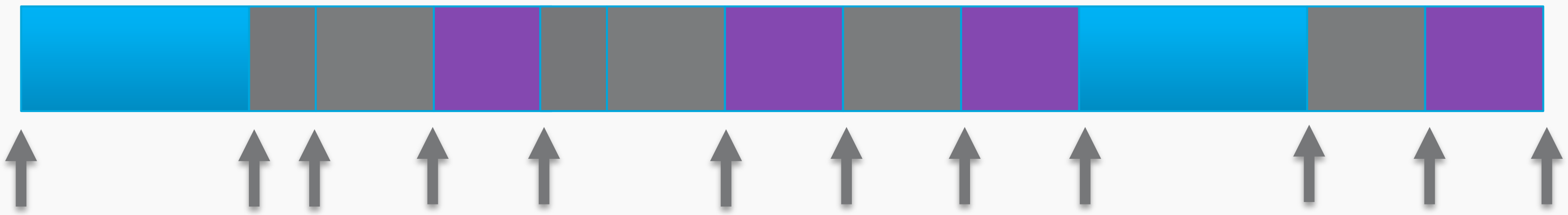
–Python+MPI analysis

PERFORMANCE CHARACTERIZATION CONCEPTS AND TOOLS

Direct Performance Observation

- Execution actions exposed as events
  - In general, actions reflect some execution state
    - presence at a code location or change in data
    - occurrence in parallelism context (thread of execution)
  - Events encode actions for observation
- Observation is direct
  - Direct instrumentation of program code (probes)
  - Instrumentation invokes performance measurement
  - Event measurement = performance data + context
- Performance experiment
  - Actual events + performance measurements

Instrumentation



Code or compiler output is modified to explicitly trigger a measurement at the beginning and end of each function/region of interest.

- More detailed information
- Unequally distributed overhead (short-running functions -> larger % overhead)
- Need to process source code

Source instrumentation

Compiler instrumentation

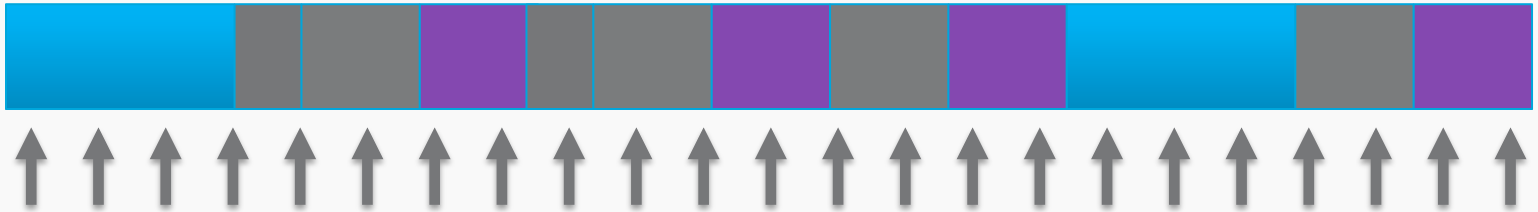
Binary rewriting

```
call
TAU_START('foo')
// code
call TAU_STOP('foo')
```

Indirect Performance Observation

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

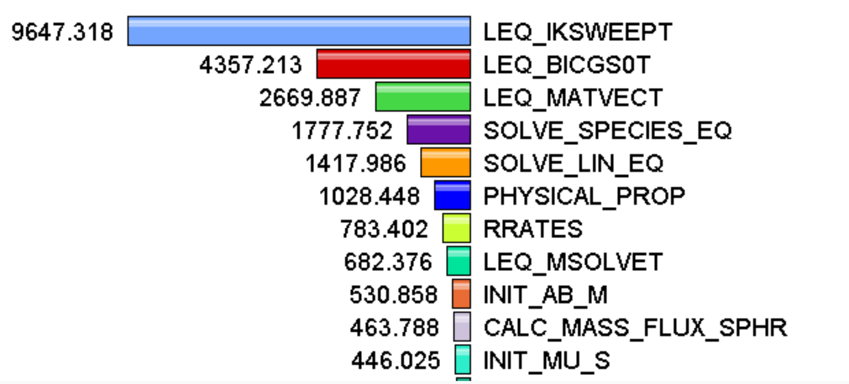
Sampling



– Periodically interrupt program and record program counter
– No modification to program needed
  - Just compile with debug symbols for address resolution
– Tradeoff between overhead and accuracy
  - Overhead evenly distributed

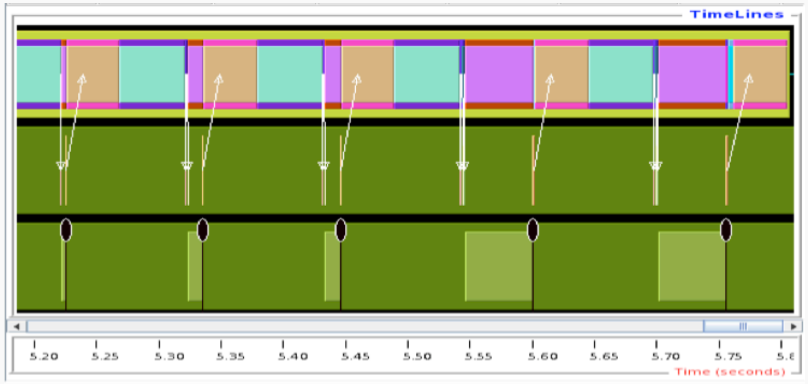
Measurement Approaches

Profiling



Shows **how much** time was spent in each routine

Tracing

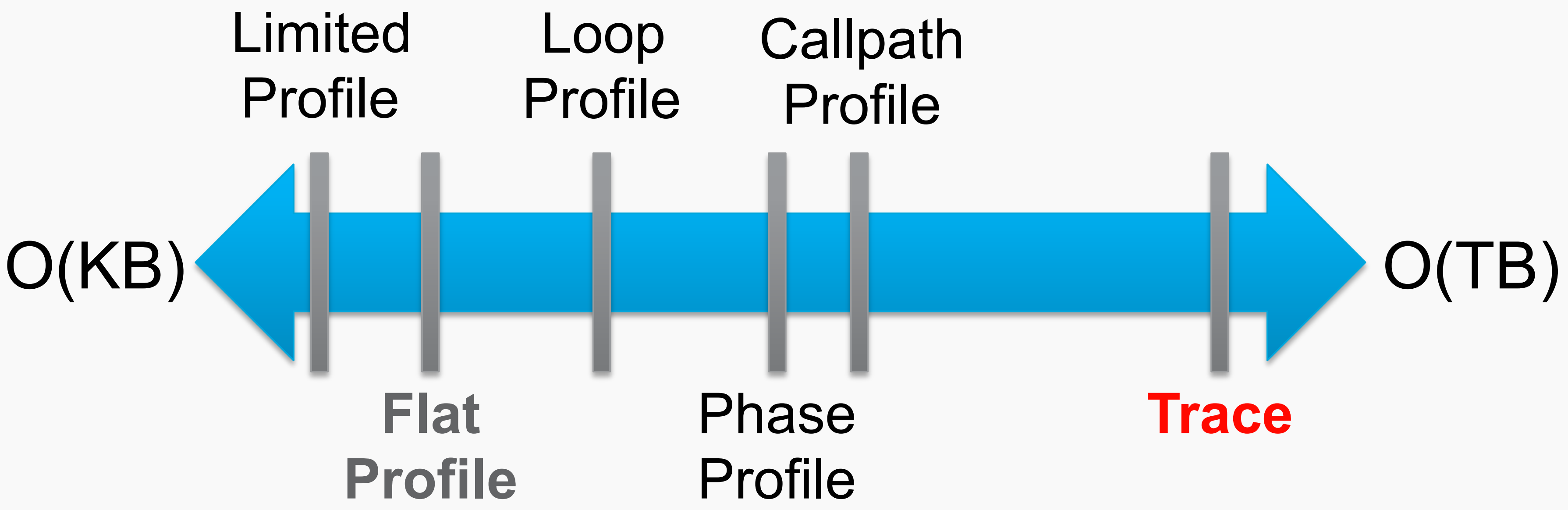


Shows **when** events take place on a timeline

Types of Performance Profiles

– Flat profiles
  - Metric (e.g., time) spent in an event
  - Exclusive/inclusive, # of calls, child calls, …
– Callpath profiles
  - Time spent along a calling path (edges in callgraph)
  - "main=> f1 => f2 => MPI_Send"
  - Set the TAU_CALLPATH_DEPTH environment variable
– Phase profiles
  - Flat profiles under a phase (nested phases allowed)
  - Default "main" phase
  - Supports static or dynamic (e.g. per-iteration) phases

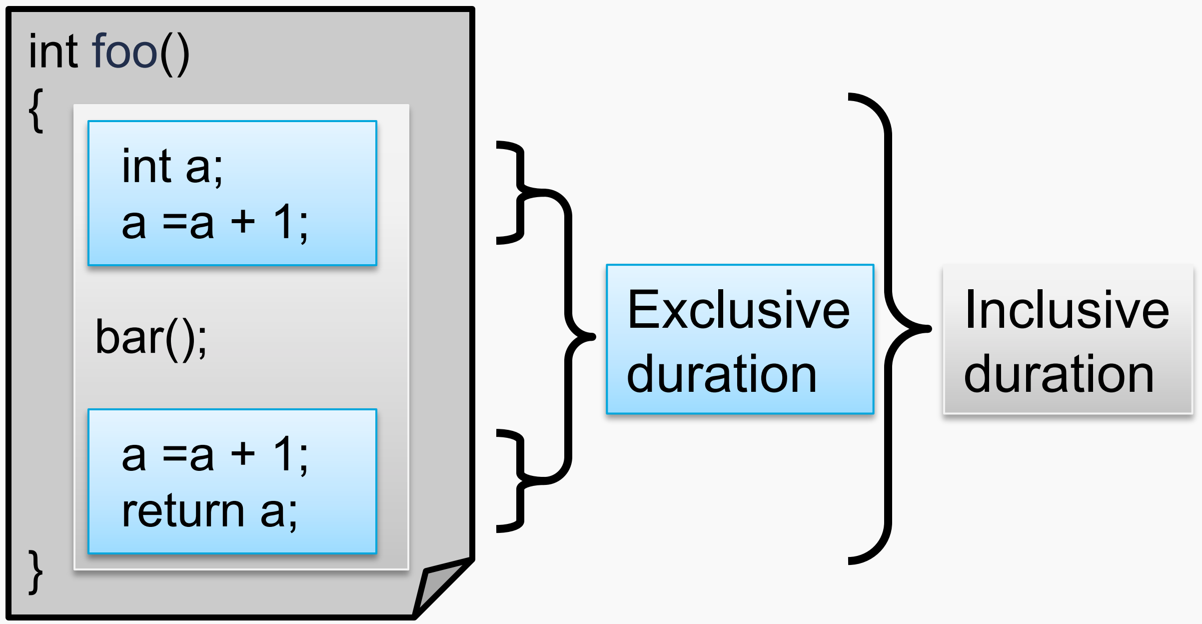
How much data do you want?



All levels support multiple metrics/ counters

Inclusive vs. Exclusive Measurements

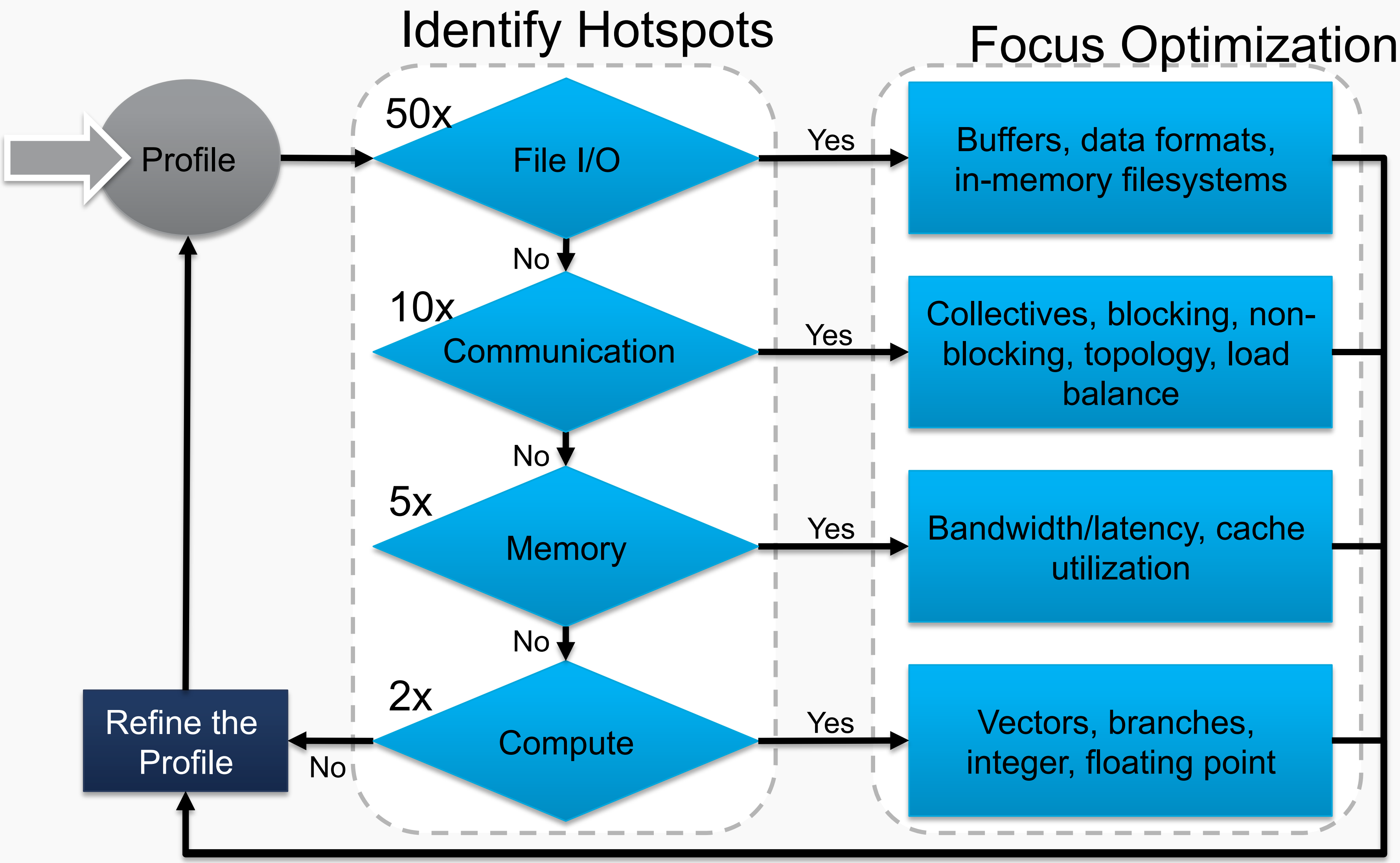
– Exclusive measurements for region only
– Inclusive measurements includes child regions



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

Measure what matters

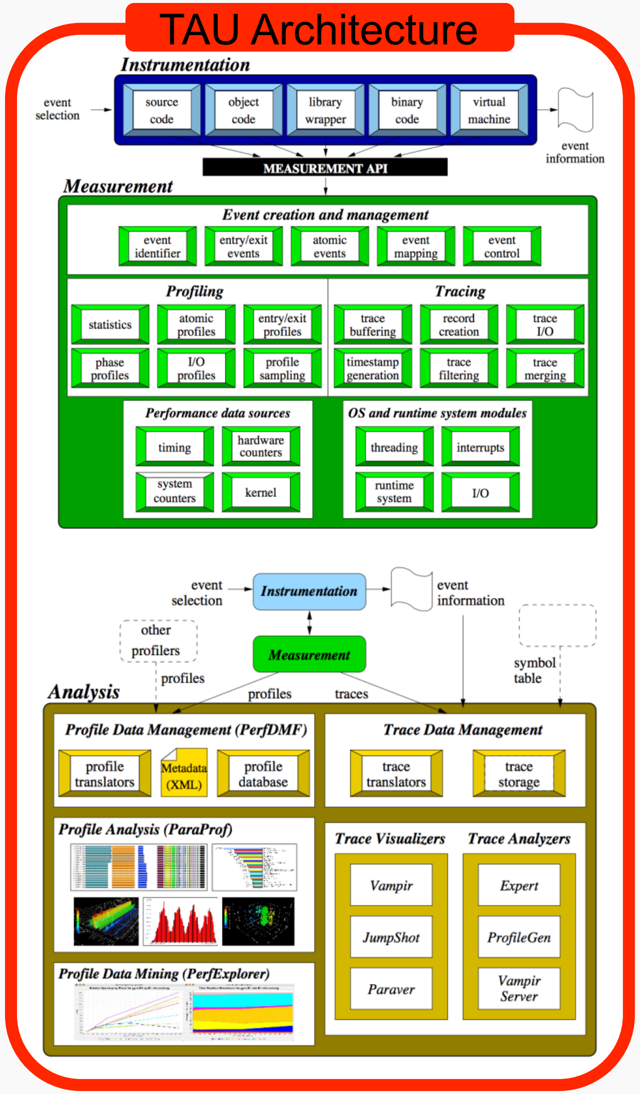


Argonne NATIONAL LABORATORY

A HIGH LEVEL OVERVIEW OF TAU'S CAPABILITIES

The TAU Performance System®

- Integrated toolkit for performance problem solving
  - Instrumentation, measurement, analysis, visualization
  - Portable profiling and tracing
  - Performance data management and data mining
- Direct and indirect measurement
- Free, open source, BSD license
- Available on all HPC platforms (and some non-HPC)
- http://tau.uoregon.edu/



Argonne NATIONAL LABORATORY

Performance Tool Checklist

- Universal tool or integrated toolkit
- Unbiased, accurate measurements
  - File I/O: serial and parallel
  - Communication: inter- and intra-node
  - Memory: allocation and access
  - CPU: vectorization, cache utilization, etc.
- Minimal overhead
  - Provide multiple measurement methods
  - Focus on one performance aspect at a time

TAU Performance System®

TAU Workflow

Instrumentation

Source
- C, C++, Fortran, UPC, …
- Python, Java, …
- Robust parsers (PDT)

Library
- Interposition (PMPI, GASNET, …)
- Wrapper generation

Linker
- Static, Dynamic
- Preloading (LD_PRELOAD)

Executable
- Dynamic (Dyninst)
- Binary (Dininst, MAQAO, PEBIL)

Measurement

Events
- Static, Dynamic
- Routine, Block, Loop
- Threadding, Communication
- Heterogeneous

Profiling
- Flat, Callpath, Phase, Snapshot
- Probe, Sampling, Compiler, Hybird

Tracing
- TAU, Scalasca, ScoreP
- Open Trace Format (OTF)

Metadata
- System
- User defined

Analysis

Profiles
- ParaProf analyzer & visualizer
  - 3D profile data visualization
  - Communication matrix
  - Callstack analysis
  - Graph generation
- PerfDMF
- PerfExplorer profile data miner

Traces
- OTF, SLOG-2
- Vampir
- Jumpshot

Online
- Event unification
- Statistics calculation

Instrument: Add Probes

- Sampling
  - Event-based sampling
- Source code instrumentation
  - PDT parsers, pre-processors
- Wrap external libraries
  - I/O, MPI, Memory, CUDA, OpenCL, pthread
- Rewrite the binary executable
  - Dyninst, MAQAO

Measure: Gather Data

–Direct measurement via probes

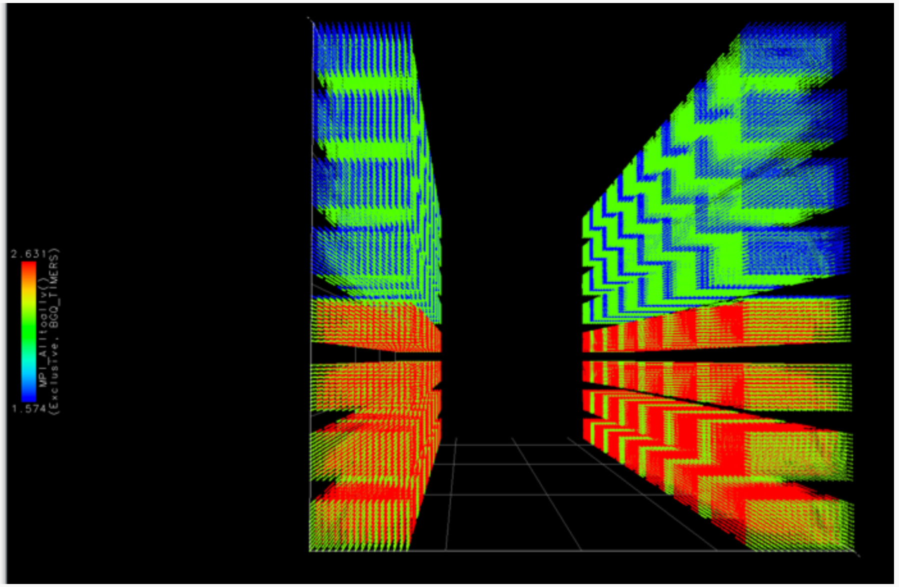
–Indirect measurement via sampling

–Throttling and runtime control

–Interface with external packages (PAPI)

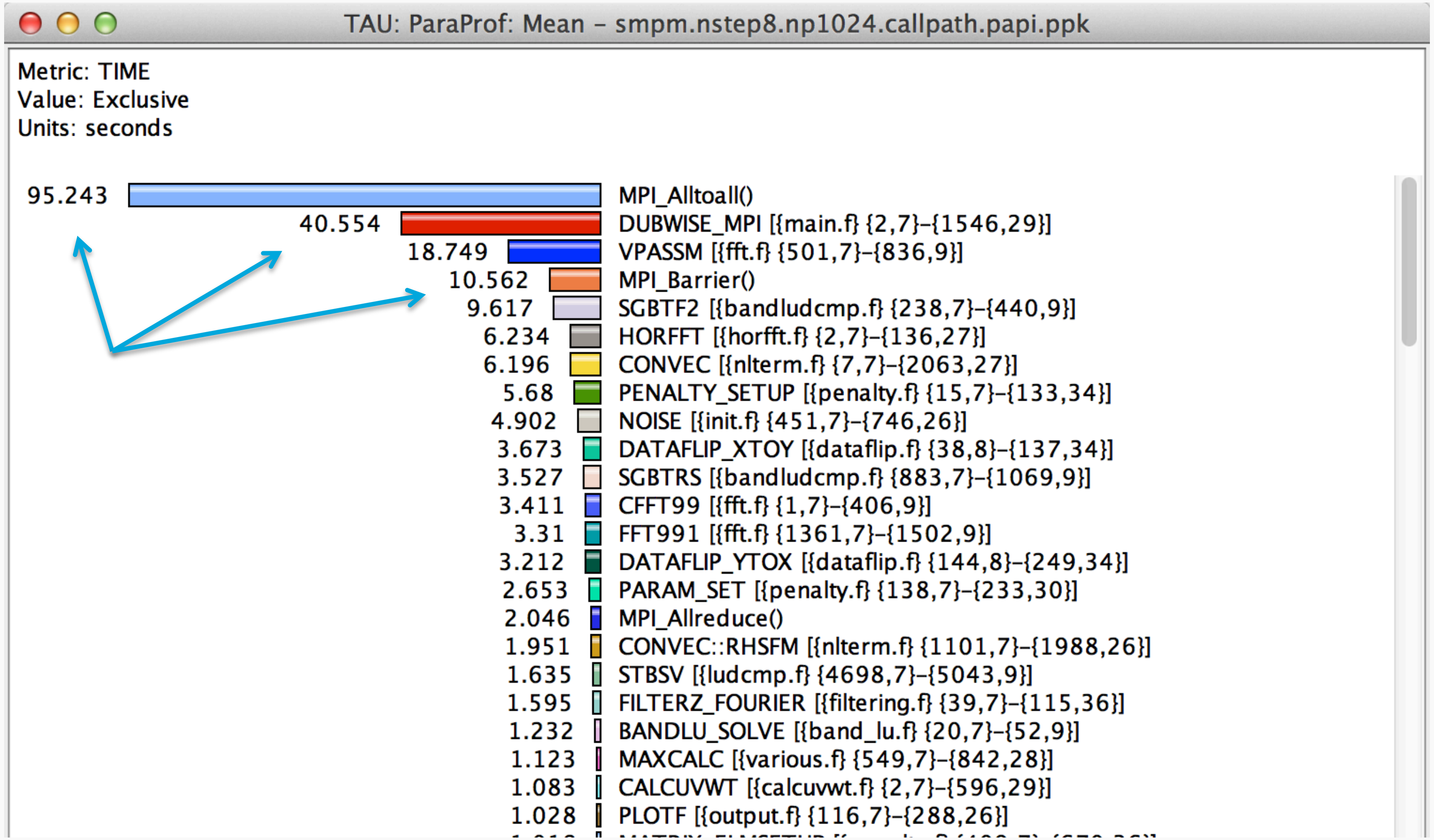
Analyze: Synthesize Knowledge

- –Data visualization
- –Data mining
- –Statistical analysis
- –Import/export performance data



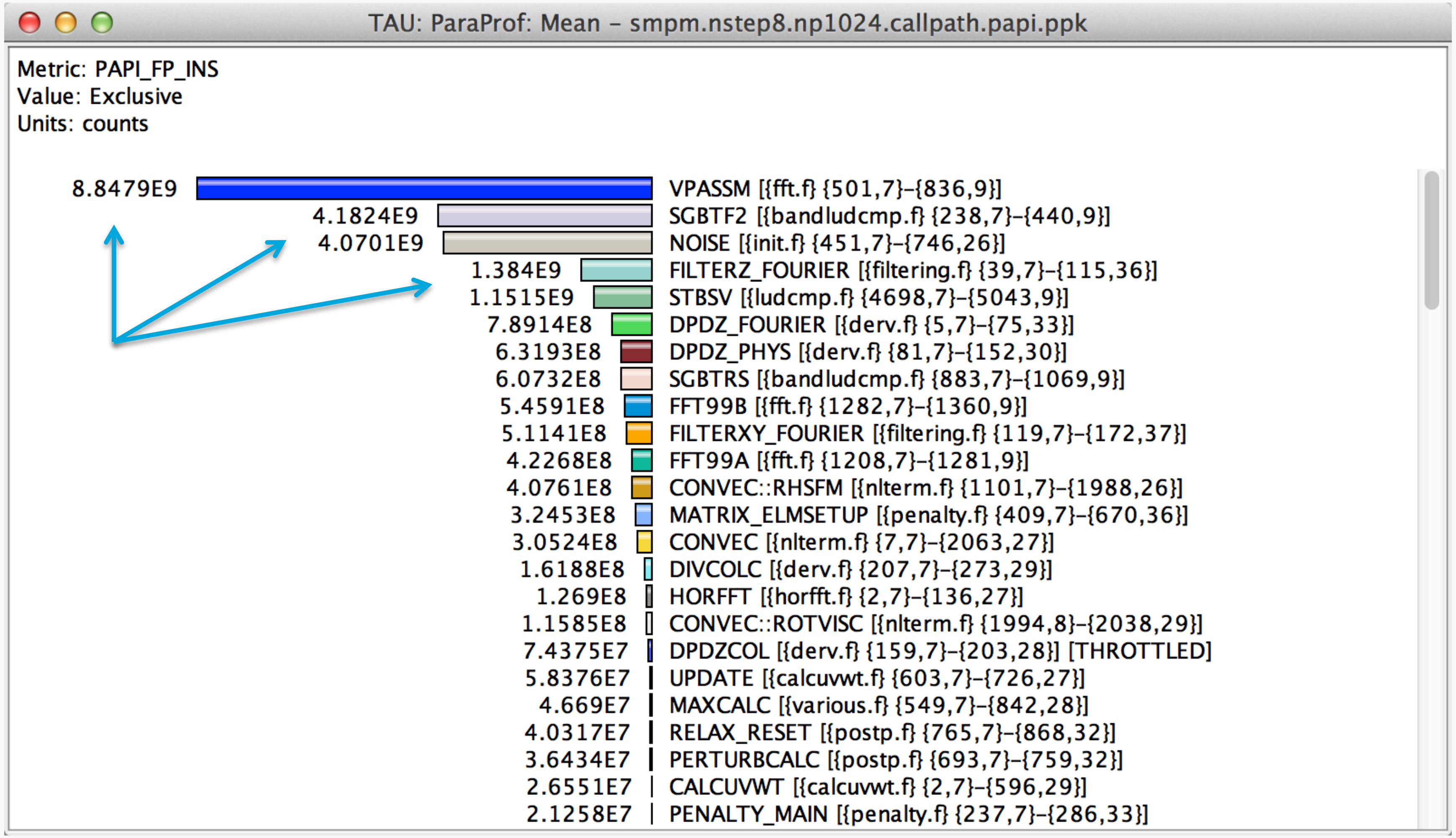
Argonne NATIONAL LABORATORY

How Much Time per Code Region?



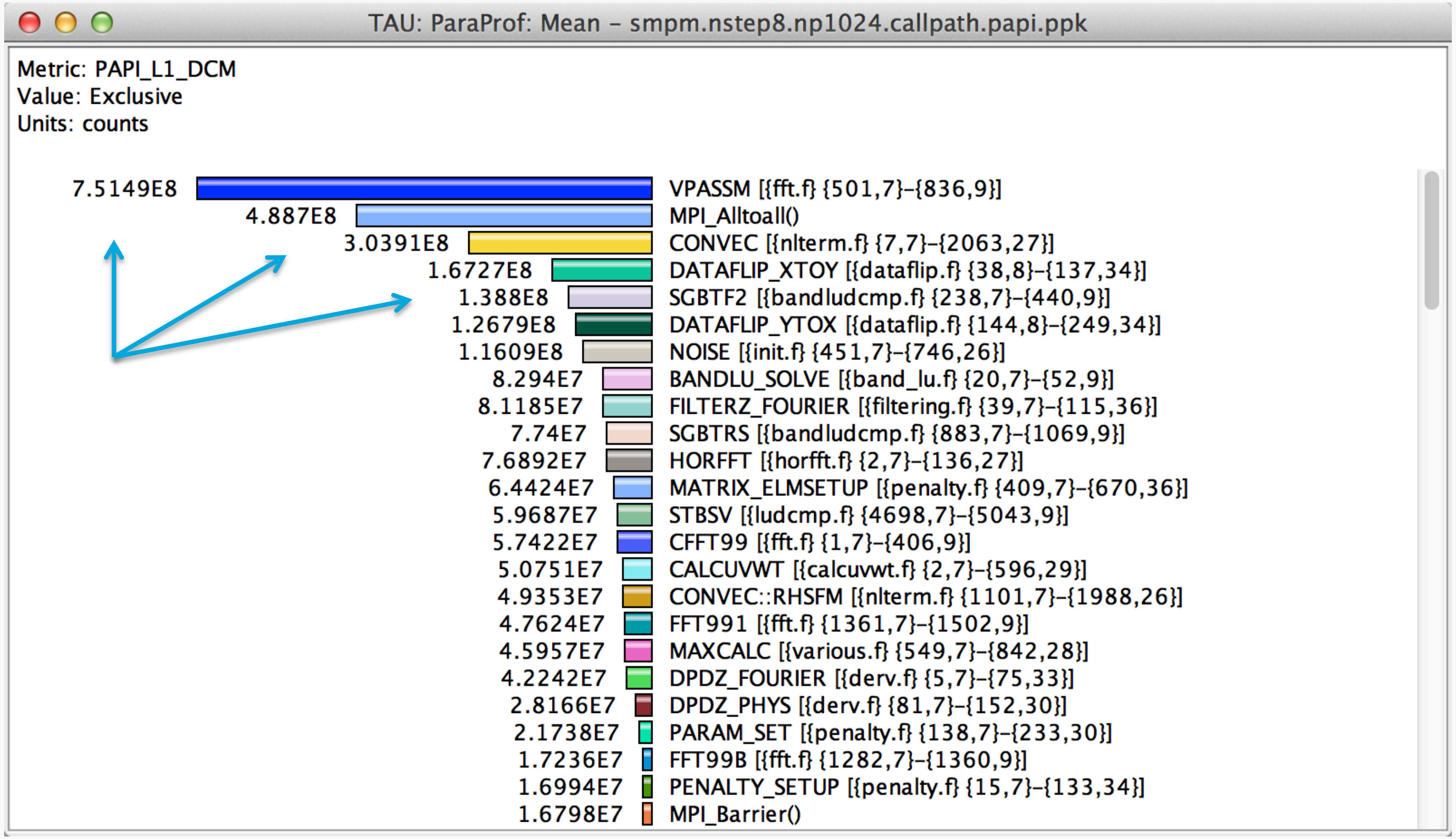
`% paraprof` **(Click on label, e.g. "Mean" or "node 0")**

How Many Instructions per Code Region?



% **paraprof** (Options → Select Metric... → Exclusive… → PAPI_FP_INS)

How Many L1 or L2 Cache Misses?



`% paraprof` (Options → Select Metric... → Exclusive… → PAPI_L1_DCM)

How Much Memory Does the Code Use?

TAU: ParaProf: Mean Context Events – sphere_np32_nsteps5_mem.ppk

Name △	Total	NumSamples	MaxValue	MinValue	MeanValue	Std. Dev.
.TAU application						
free size (bytes)	14,236,992.16	27,169.781	49,152	1	524.001	2,013.103
malloc size (bytes)	13,132,932	23,292	262,144	1	563.839	4,492.057
MPI_Finalize()						
OurMain()						
free size (bytes)	1,298,918.679	1,495.125	461,766.25	4	868.769	16,928.073
malloc size (bytes)	48,150	20	36,032	11	2,407.5	7,911.992
OurMain						
free size (bytes)	3,465	9	769	32	385	260.2
malloc size (bytes)	4,314	12	769	32	359.5	240.981
<module>						
free size (bytes)	293,088	449	32,564	32	652.757	1,526.875
malloc size (bytes)	311,966	493	32,564	32	632.791	1,460.941
staticCFD						
__init__						
<module>						
Memory Utilization (heap, in KB)		849,270.344	192,825.168	0.078	147,832.141	62,621.576
Message size for all-gather	4,096	1	4,096	4,096	4,096	0
Message size for all-reduce	23,340	843	320	4	27.687	64.653
Message size for all-to-all	104	26	4	4	4	0
Message size for broadcast	24,923	206	8,788	4	120.985	860.992
Message size for reduce	8,912	8	8,788	4	1,114	2,900.511
free size (bytes)	27,417,881,391.51	413,600.719	24,025,667	1	66,290.701	199,538.234
malloc size (bytes)	27,468,709,355.914	435,669.625	24,025,667	0	63,049.402	195,561.193

High-water mark

`% paraprof` (Right-click label [e.g "node 0"] → Show Context Event Window)

How Much Memory Does the Code Use?

TAU: ParaProf: Mean Context Events – sphere_np32_nsteps5_mem.ppk

Name △	Total	NumSamples	MaxValue	MinValue	MeanValue	Std. Dev.
.TAU application						
free size (bytes)	14,236,992.16	27,169.781	49,152	1	524.001	2,013.103
malloc size (bytes)	13,132,932	23,292	262,144	1	563.839	4,492.057
MPI_Finalize()						
OurMain()						
free size (bytes)	1,298,918.679	1,495.125	461,766.25	4	868.769	16,928.073
malloc size (bytes)	48,150	20	36,032	11	2,407.5	7,911.992
OurMain						
free size (bytes)	3,465	9	769	32	385	260.2
malloc size (bytes)	4,314	12	769	32	359.5	240.981
<module>						
free size (bytes)	293,088	449	32,564	32	652.757	1,526.875
malloc size (bytes)	311,966	493	32,564	32	632.791	1,460.941
staticCFD						
__init__						
<module>						
Memory Utilization (heap, in KB)		849,270.344	192,825.168	0.078	147,832.141	62,621.576
Message size for all-gather	4,096	1	4,096	4,096	4,096	0
Message size for all-reduce	23,340	843	320	4	27.687	64.653
Message size for all-to-all	104	26	4	4	4	0
Message size for broadcast	24,923	206	8,788	4	120.985	860.992
Message size for reduce	8,912	8	8,788	4	1,114	2,900.511
free size (bytes)	27,417,881,391.51	413,600.719	24,025,667	1	66,290.701	199,538.234
malloc size (bytes)	27,468,709,355.914	435,669.625	24,025,667	0	63,049.402	195,561.193

Total allocated/deallocated

`% paraprof` (Right-click label [e.g “node 0”] → Show Context Event Window)

Argonne NATIONAL LABORATORY

Where is Memory Allocated / Deallocated?

TAU: ParaProf: Mean Context Events – sphere_np32_nsteps5_mem.ppk

Name △	Total	NumSamples	MaxValue	MinValue	MeanValue	Std. Dev.
▼ .TAU application						
free size (bytes)	14,236,992.16	27,169.781	49,152	1	524.001	2,013.103
malloc size (bytes)	13,132,932	23,292	262,144	1	563.839	4,492.057
▶ MPI_Finalize()						
▼ OurMain()						
free size (bytes)	1,298,918.679	1,495.125	461,766.25	4	868.769	16,928.073
malloc size (bytes)	48,150	20	36,032	11	2,407.5	7,911.992
▼ OurMain						
free size (bytes)	3,465	9	769	32	385	260.2
malloc size (bytes)	4,314	12	769	32	359.5	240.981
▼ <module>						
free size (bytes)	293,088	449	32,564	32	652.757	1,526.875
malloc size (bytes)	311,966	493	32,564	32	632.791	1,460.941
▶ staticCFD						
▶ __init__						
▶ <module>						
Memory Utilization (heap, in KB)		849,270.344	192,825.168	0.078	147,832.141	62,621.576
Message size for all-gather	4,096	1	4,096	4,096	4,096	0
Message size for all-reduce	23,340	843	320	4	27.687	64.653
Message size for all-to-all	104	26	4	4	4	0
Message size for broadcast	24,923	206	8,788	4	120.985	860.992
Message size for reduce	8,912	8	8,788	4	1,114	2,900.511
free size (bytes)	27,417,881,391.51	413,600.719	24,025,667	1	66,290.701	199,538.234
malloc size (bytes)	27,468,709,355.914	435,669.625	24,025,667	0	63,049.402	195,561.193

Allocation / Deallocation Events

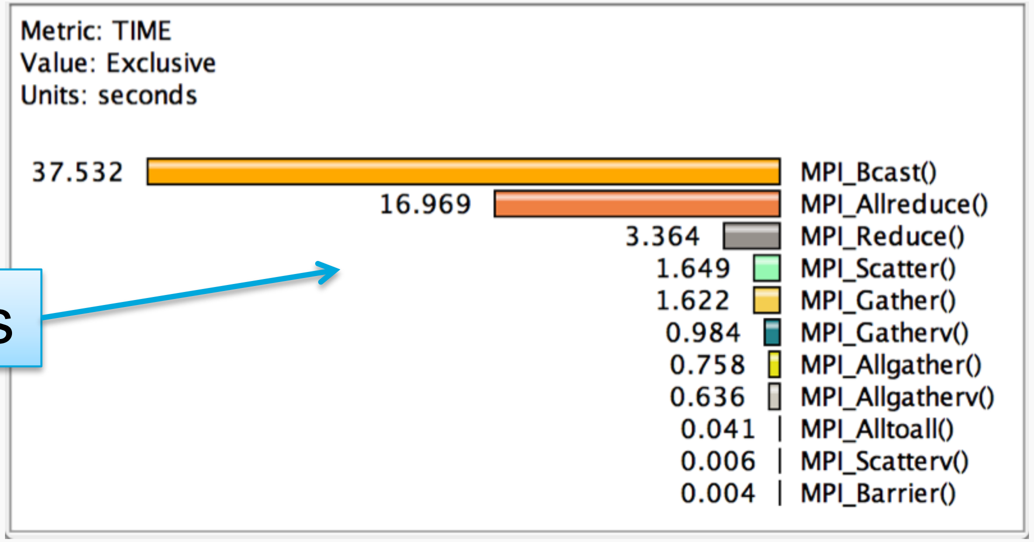
`% paraprof` (Right-click label [e.g "node 0"] → Show Context Event Window)

How Much Time is spent in Collectives?

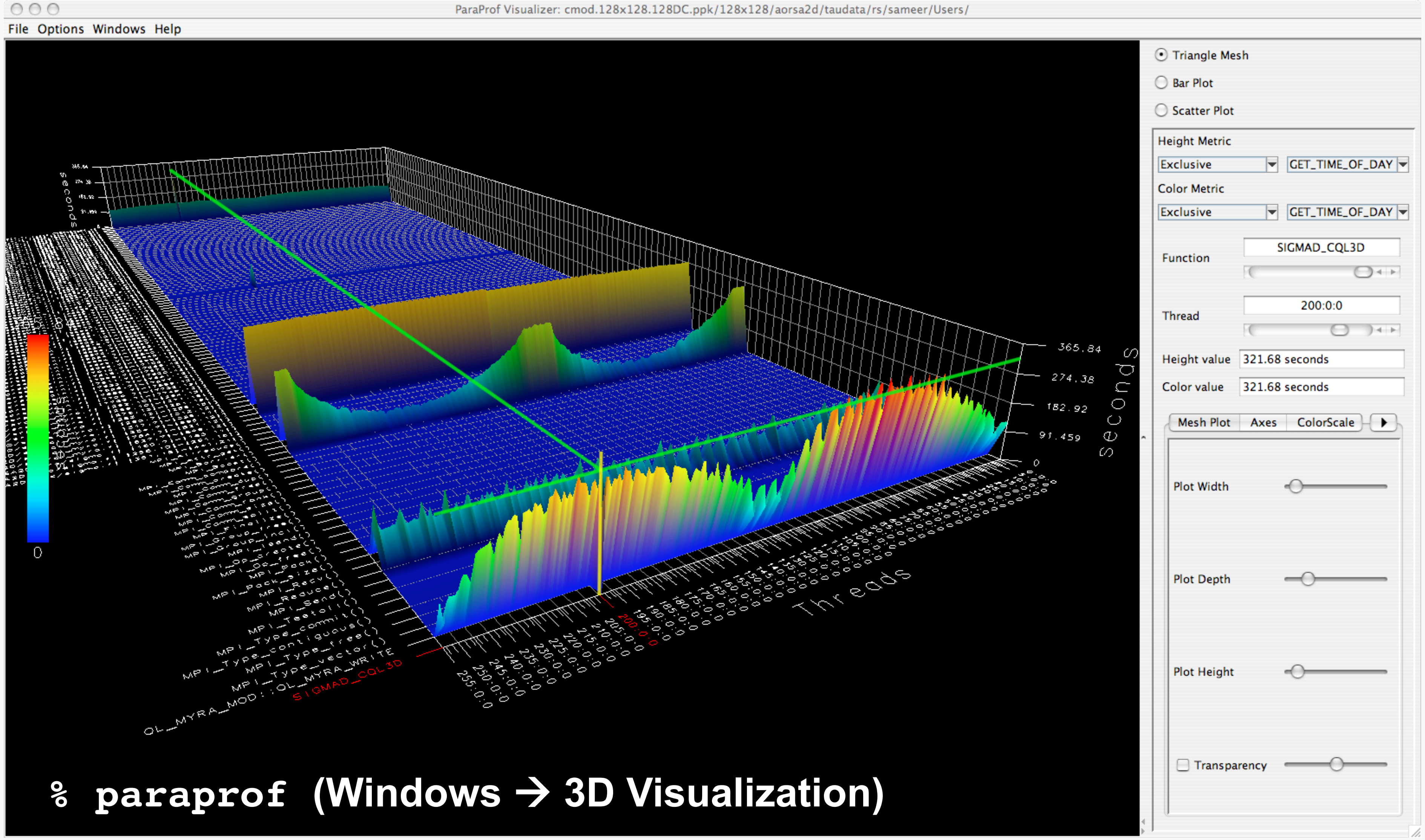
Name △	Total	Num...	MaxValue	MinValue	MeanValue	Std. Dev.
▶ MPI_Wait()						
▶ MPI_Waitall()						
Message size for all-gather	305,753,268	72	172,215,296	4	4,246,573.167	22,551,605.859
Message size for all-reduce	163,308	632	21,908	4	258.399	897.725
Message size for all-to-all	112	14	8	8	8	0
Message size for broadcast	692,208,045.5	3,346	18,117,620	0	206,876.284	1,284,673.036
Message size for gather	6,901,452.378	15.312	1,387,306.625	4	450,707.094	483,216.499
Message size for reduce	66,812	1,520	56	4	43.955	21.598
Message size for scatter	63,147.906	146	62,567.906	4	432.52	5,160.063

Message sizes

Time spent in collectives



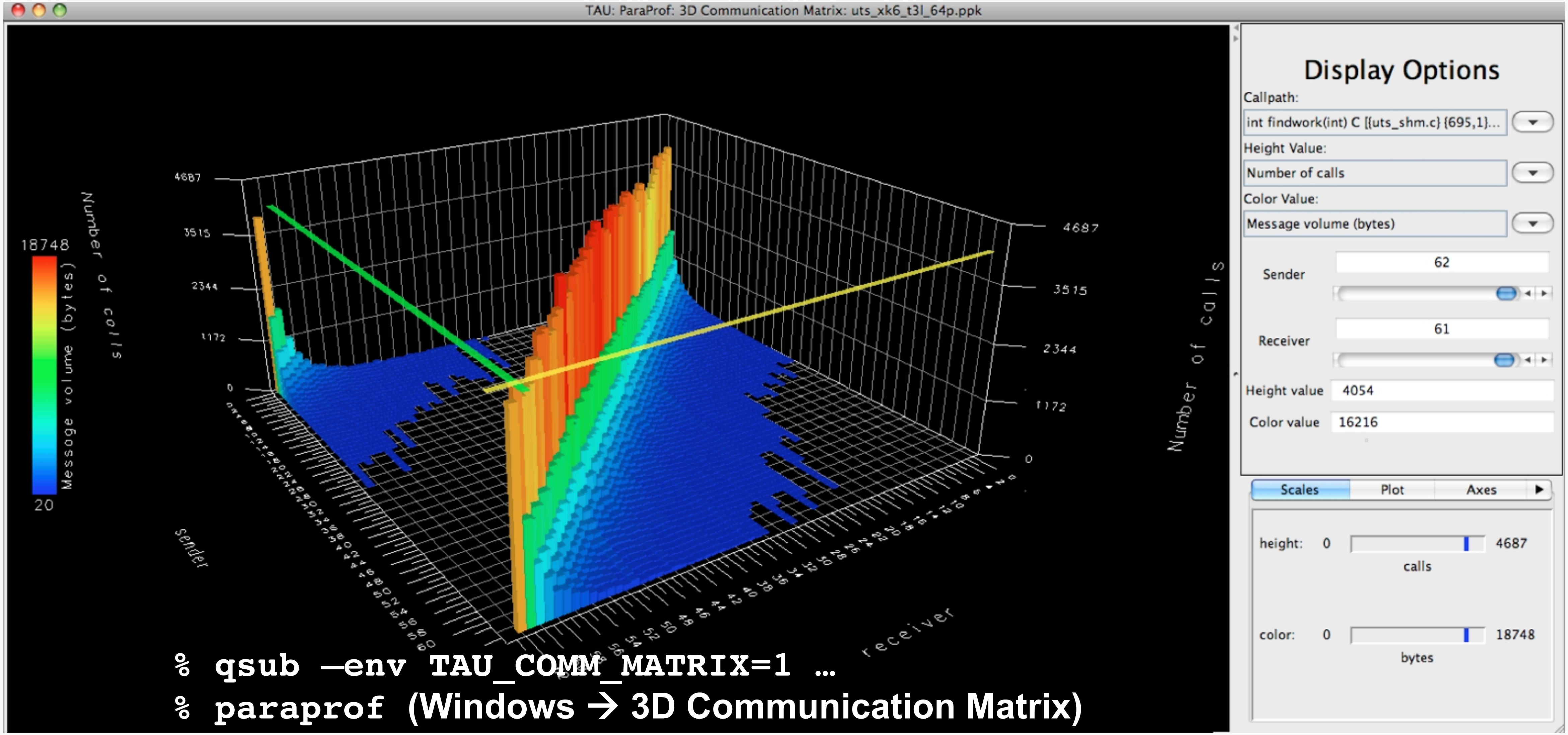
3D Profile Visualization



% **paraprof** (Windows → 3D Visualization)

Argonne NATIONAL LABORATORY

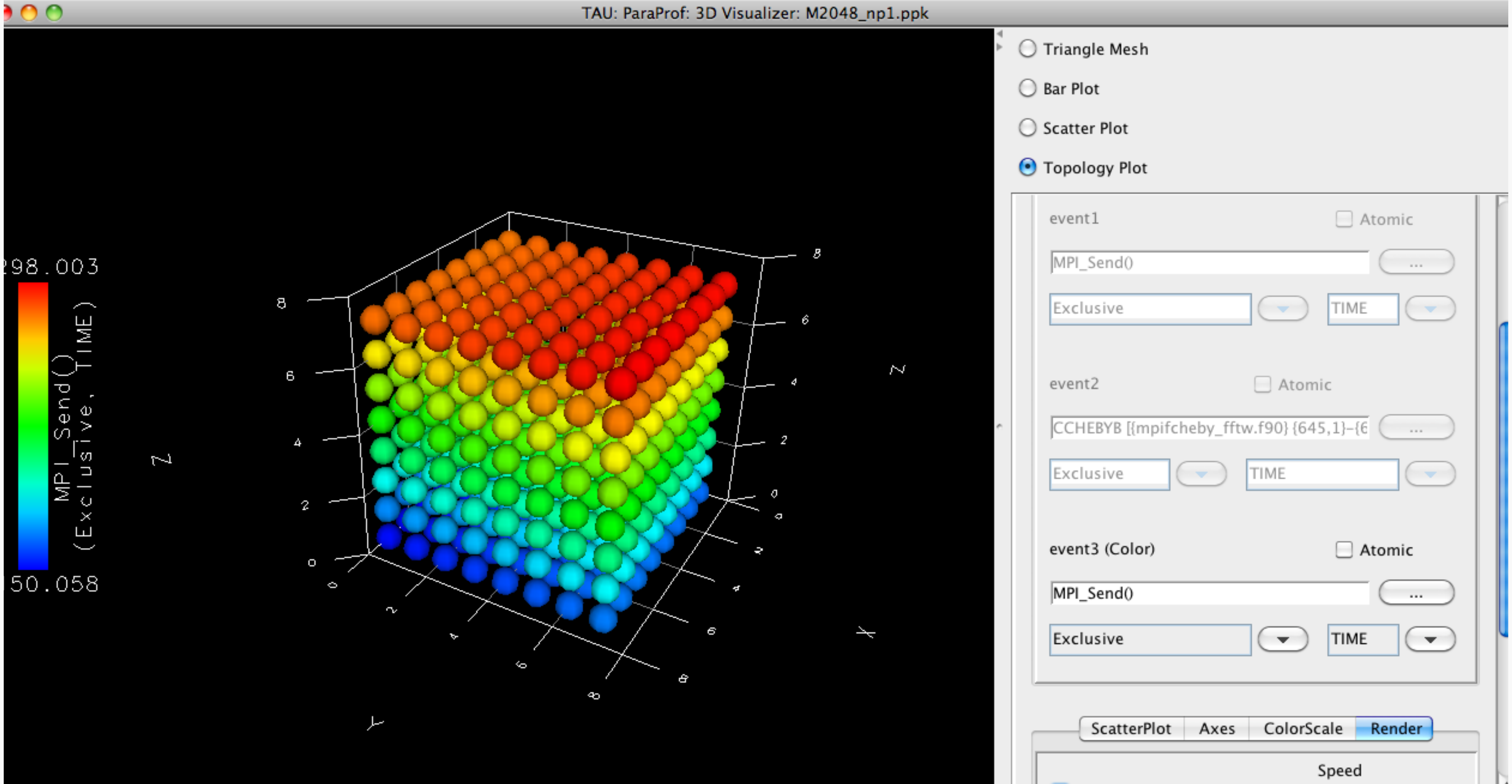
3D Communication Visualization



% **qsub –env TAU_COMM_MATRIX=1 …**

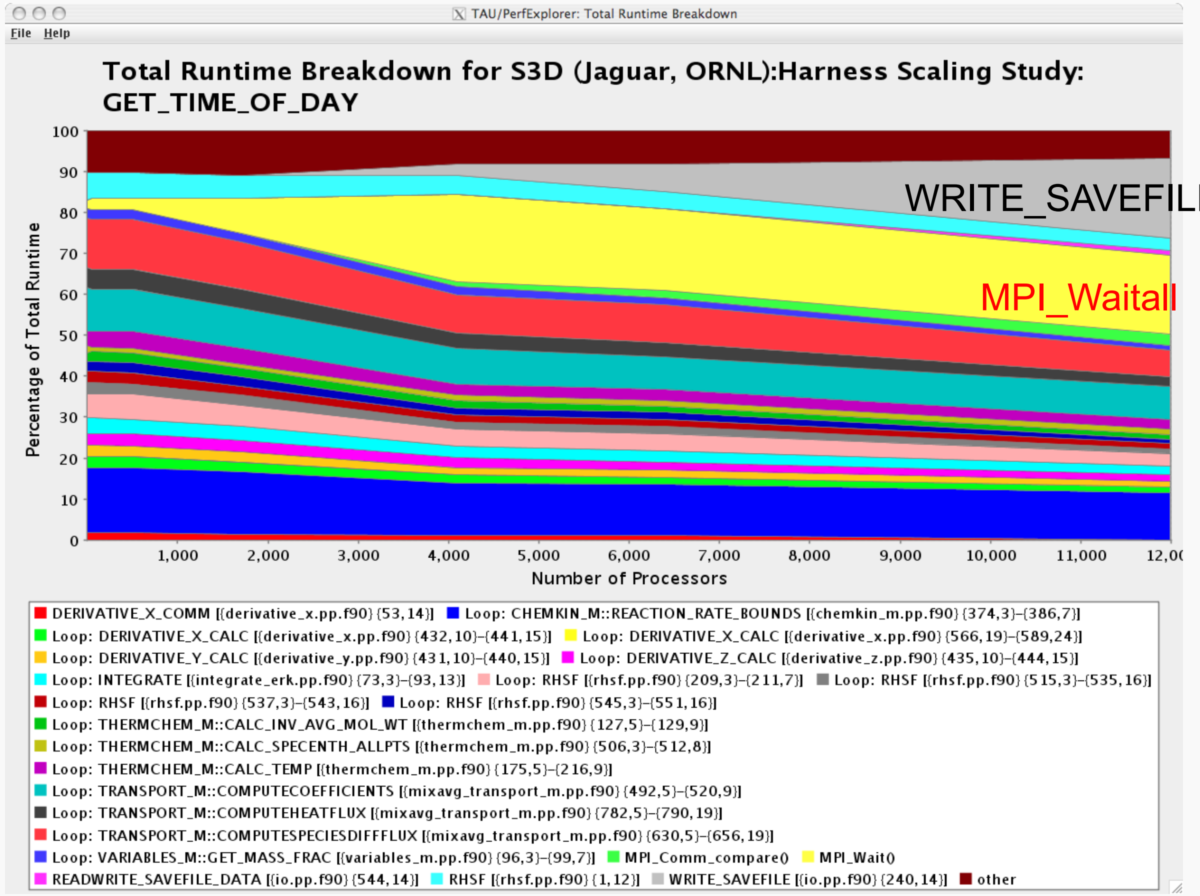
% **paraprof (Windows → 3D Communication Matrix)**

3D Topology Visualization



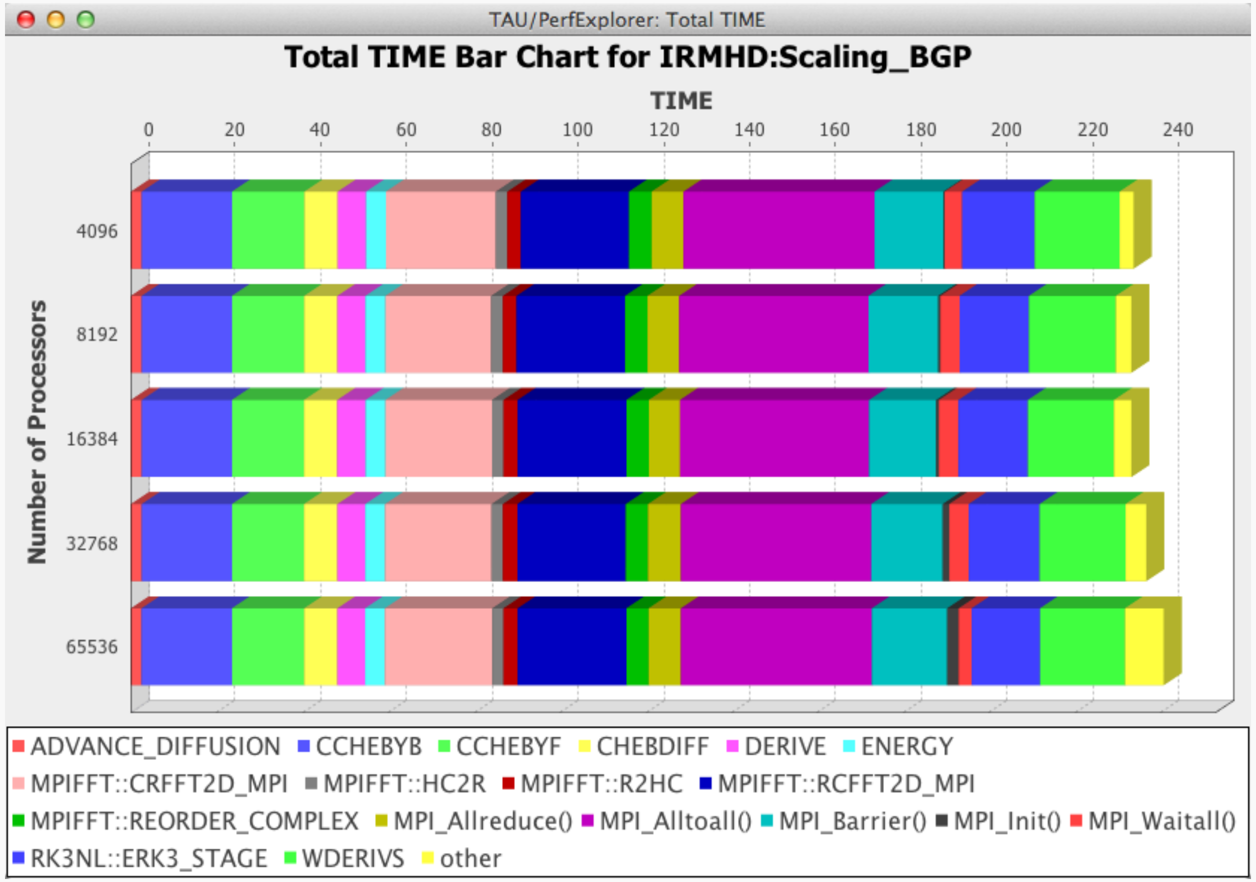
% **paraprof** **(Windows → 3D Visualization → Topology Plot)**

How Does Each Routine Scale?



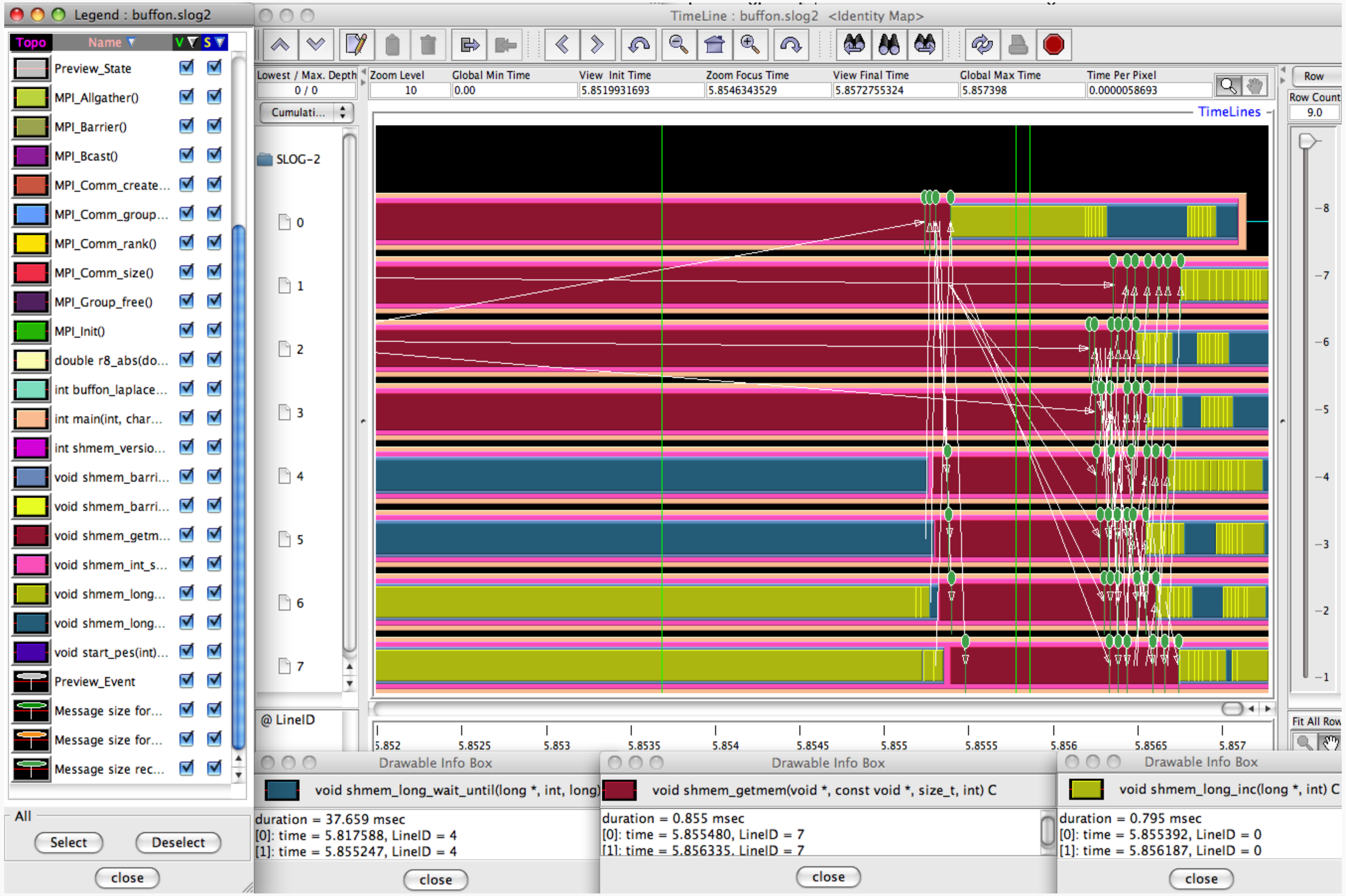
`% perfexplorer` **(Charts → Runtime Breakdown)**

How Does Each Routine Scale?



`% perfexplorer` (Charts → Stacked Bar Chart)

When do Events Occur?



`export TAU_TRACE=1`

Argonne NATIONAL LABORATORY

Intuitive Performance Engineering

USING TAU

Preinstalled Tools on Theta

– On Theta, versions of TAU and related tools are installed at /soft/perftools/tau

– Modules are available:

```
– % module avail tau
  ----- /soft/environment/modules/modulefiles ------
  tau/2.25.2 tau/2.26   tau/2.26.1 tau/2.26.2 tau/2.26.3 tau/2.27
  tau/2.27.1 tau/2.27.2
– % module load tau/2.27.2
```

Using TAU Directly

- An unusual thing about TAU installations
  - For most UNIX software, when running
    - `./configure --foo; make install`
    - `./configure --bar; make install`
  - the second install will overwrite the first install.
  - In TAU, the two configurations are installed ***side by side***.
- To reduce overhead, many features are enabled at compile time rather than runtime.
  - Always use `–bfd=download –unwind=download` for sampling address resolution
  - Common ./configure options: `-cc`, `-c++`, `-fortran` select compiler, `-mpi`, `-pthread`, `-openmp`, `-cuda`
- For each configuration of TAU, a `Makefile` is present in `$TAU/<arch>/lib/Makefile.tau-*`
  - `<arch>` is `craycnl` on Theta
- `TAU_MAKEFILE` environment variable determines configuration used by compiler wrappers.
  - e.g., export `TAU_MAKEFILE=<path to TAU>/<arch>/lib/Makefile.tau-intel-papi-mpi-pdt`
- `-T` option determines configuration used by `tau_exec`
  - E.g., `tau_exec -T intel,papi,pdt`
  - `tau_exec` assumes `mpi`; specify `serial` if not

Typical Workflow

– Sample to identify hotspots (tau_exec -ebs)
– Selectively instrument hotspots (tau_f90.sh and friends, -optSelectFile)
– Gather hardware performance counter data (papi_avail, TAU_METRICS)
– Visualize performance data, derived metrics (paraprof)
– Visualize scaling data (perfexplorer)

Sampling with TAU

– Use `tau_exec -ebs`
  - Build without TAU *as dynamic executable*, *with –g* if you want line-level resolution.
    - Does not work with static executables, which are default with Cray compiler wrappers!
  - Run application through `tau_exec`, prepending launcher

Makefile without TAU

```
CXX = cc -dynamic
F90 = ftn -dynamic
CXXFLAGS = -g
LIBS =
OBJS = f1.o f2.o f3.o … fn.o

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

`tau_exec` comes **after** launcher. Otherwise, we would sample aprun itself!

```
aprun –n 16 tau_exec –T mpi,pthread -ebs ./foo
```

Insert TAU API Calls Automatically

- Use TAU's compiler wrappers
  - Replace CXX with tau_cxx.sh, etc.
  - Automatically instruments source code, links with TAU libraries.
- Use tau_cc.sh for C, tau_f90.sh for Fortran, etc.
- Run normally through launcher

Makefile without TAU

```
CXX = cc
F90 = ftn
CXXFLAGS =
OBJS = f1.o f2.o f3.o … fn.o

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

Makefile with TAU

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

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

Compile-Time Options

Optional parameters for the TAU_OPTIONS environment variable:
% tau_compiler.sh

- -optVerbose — Turn on verbose debugging messages
- -optCompInst — Use compiler based instrumentation
- -optNoCompInst — Do not revert to compiler instrumentation if source instrumentation fails.
- -optTrackIO — Wrap POSIX I/O call and calculates vol/bw of I/O operations (Requires TAU to be configured with –iowrapper)
- -optKeepFiles — Does not remove intermediate .pdb and .inst.* files
- -optPreProcess — Preprocess sources (OpenMP, Fortran) before instrumentation
- -optTauSelectFile=”<file>" — Specify selective instrumentation file for tau_instrumentor
- -optTauWrapFile=”<file>" — Specify path to link_options.tau generated by tau_gen_wrapper
- -optHeaderInst — Enable Instrumentation of headers
- -optTrackUPCR — Track UPC runtime layer routines (used with tau_upc.sh)
- -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/gfparse) …

Runtime Environment Variables

Environment Variable	Default	Description
TAU_TRACE	0	Setting to 1 turns on tracing
TAU_CALLPATH	0	Setting to 1 turns on callpath profiling
TAU_TRACK_MEMORY_LEAKS	0	Setting to 1 turns on leak detection (for use with tau_exec –memory ./a.out)
TAU_TRACK_HEAP or TAU_TRACK_HEADROOM	0	Setting to 1 turns on tracking heap memory/headroom at routine entry & exit using context events (e.g., Heap at Entry: main=>foo=>bar)
TAU_CALLPATH_DEPTH	2	Specifies depth of callpath. Setting to 0 generates no callpath or routine information, setting to 1 generates flat profile and context events have just parent information (e.g., Heap Entry: foo)
TAU_TRACK_IO_PARAMS	0	Setting to 1 with –optTrackIO or tau_exec –io captures arguments of I/O calls
TAU_TRACK_SIGNALS	0	Setting to 1 generate debugging callstack info when a program crashes
TAU_COMM_MATRIX	0	Setting to 1 generates communication matrix display using context events
TAU_THROTTLE	1	Setting to 0 turns off throttling. Enabled by default to remove instrumentation in lightweight routines that are called frequently
TAU_THROTTLE_NUMCALLS	100000	Specifies the number of calls before testing for throttling
TAU_THROTTLE_PERCALL	10	Specifies value in microseconds. Throttle a routine if it is called over 100000 times and takes less than 10 usec of inclusive time per call
TAU_COMPENSATE	0	Setting to 1 enables runtime compensation of instrumentation overhead
TAU_PROFILE_FORMAT	Profile	Setting to "merged" generates a single file. "snapshot" generates xml format
TAU_METRICS	TIME	Setting to a comma separated list generates other metrics. (e.g., TIME:P_VIRTUAL_TIME:PAPI_FP_INS:PAPI_NATIVE_<event>\\:<subevent>)

Hardware Counters

Hardware performance counters available on most modern microprocessors can provide insight into:

1. Whole program timing
2. Cache behaviors
3. Branch behaviors
4. Memory and resource access patterns
5. Pipeline stalls
6. Floating point efficiency
7. Instructions per cycle

Hardware counter information can be obtained with:

1. Subroutine or basic block resolution
2. Process or thread attribution

What' s PAPI?

Open Source software from U. Tennessee, Knoxville

http://icl.cs.utk.edu/papi

Middleware to provide a consistent programming interface for the performance counter hardware found in most major micro-processors.

Countable events are defined in two ways:

– Platform-neutral preset events
– Platform-dependent native events

Presets can be derived from multiple native events

All events are referenced by name and collected in EventSets

PAPI Utilities: papi_avail

```
$ utils/papi_avail
Available events and hardware information.
--------------------------------------------------------------------------------
PAPI Version             : 4.0.0.0
Vendor string and code   : GenuineIntel (1)
Model string and code    : Intel Core i7 (21)
CPU Revision             : 5.000000
CPUID Info               : Family: 6  Model: 26  Stepping: 5
CPU Megahertz            : 2926.000000
CPU Clock Megahertz      : 2926
Hdw Threads per core     : 1
Cores per Socket         : 4
NUMA Nodes               : 2
CPU's per Node           : 4
Total CPU's              : 8
Number Hardware Counters : 7
Max Multiplex Counters   : 32
--------------------------------------------------------------------------------
The following correspond to fields in the PAPI_event_info_t structure.

   [MORE…]
```

PAPI Utilities: papi_avail

[CONTINUED…]

```
--------------------------------------------------------------------------------
The following correspond to fields in the PAPI_event_info_t structure.

    Name        Code    Avail Deriv Description (Note)
PAPI_L1_DCM  0x80000000  No    No   Level 1 data cache misses
PAPI_L1_ICM  0x80000001  Yes   No   Level 1 instruction cache misses
PAPI_L2_DCM  0x80000002  Yes   Yes  Level 2 data cache misses

[…]

PAPI_VEC_SP  0x80000069  Yes   No   Single precision vector/SIMD instructions
PAPI_VEC_DP  0x8000006a  Yes   No   Double precision vector/SIMD instructions
-------------------------------------------------------------------------
Of 107 possible events, 34 are available, of which 9 are derived.

avail.c                                PASSED
```

PAPI Utilities: papi_avail

```
$ utils/papi_avail -e PAPI_FP_OPS
[…]
--------------------------------------------------------------------------------
The following correspond to fields in the PAPI_event_info_t structure.

Event name:                 PAPI_FP_OPS
Event Code:                 0x80000066
Number of Native Events:    2
Short Description:          |FP operations|
Long Description:           |Floating point operations|
Developer's Notes:          ||
Derived Type:               |DERIVED_ADD|
Postfix Processing String:  ||
 Native Code[0]: 0x4000801b |FP_COMP_OPS_EXE:SSE_SINGLE_PRECISION|
 Number of Register Values: 2
 Register[ 0]: 0x0000000f |Event Selector|
 Register[ 1]: 0x00004010 |Event Code|
 Native Event Description: |Floating point computational micro-ops, masks:SSE* FP single precision Uops|

 Native Code[1]: 0x4000081b |FP_COMP_OPS_EXE:SSE_DOUBLE_PRECISION|
 Number of Register Values: 2
 Register[ 0]: 0x0000000f |Event Selector|
 Register[ 1]: 0x00008010 |Event Code|
 Native Event Description: |Floating point computational micro-ops, masks:SSE* FP double precision Uops|
--------------------------------------------------------------------------------
```

PAPI Utilities: papi_native_avail

```
UNIX> utils/papi_native_avail
Available native events and hardware information.
--------------------------------------------------------------------------------
[…]
Event Code   Symbol  | Long Description |
--------------------------------------------------------------------------------
0x40000010   BR_INST_EXEC  | Branch instructions executed                        |
  40000410   :ANY  | Branch instructions executed                              |
  40000810   :COND  | Conditional branch instructions executed                 |
  40001010   :DIRECT  | Unconditional branches executed                        |
  40002010   :DIRECT_NEAR_CALL  | Unconditional call branches executed         |
  40004010   :INDIRECT_NEAR_CALL  | Indirect call branches executed            |
  40008010   :INDIRECT_NON_CALL  | Indirect non call branches executed         |
  40010010   :NEAR_CALLS  | Call branches executed                             |
  40020010   :NON_CALLS  | All non call branches executed                      |
  40040010   :RETURN_NEAR  | Indirect return branches executed                 |
  40080010   :TAKEN  | Taken branches executed                                 |
--------------------------------------------------------------------------------
0x40000011   BR_INST_RETIRED  | Retired branch instructions                     |
  40000411   :ALL_BRANCHES  | Retired branch instructions (Precise Event)      |
  40000811   :CONDITIONAL  | Retired conditional branch instructions (Precise |
           | Event)                                                            |
  40001011   :NEAR_CALL  | Retired near call instructions (Precise Event)      |
```

PAPI Utilities: papi_native_avail

```
UNIX> utils/papi_native_avail -e DATA_CACHE_REFILLS
Available native events and hardware information.
--------------------------------------------------------------------------------
[…]
--------------------------------------------------------------------------------
The following correspond to fields in the PAPI_event_info_t structure.

Event name:                  DATA_CACHE_REFILLS
Event Code:                  0x4000000b
Number of Register Values:   2
Description:                |Data Cache Refills from L2 or System|
 Register[ 0]:   0x0000000f  |Event Selector|
 Register[ 1]:   0x00000042  |Event Code|

Unit Masks:
 Mask Info:                 |:SYSTEM|Refill from System|
  Register[ 0]:  0x0000000f  |Event Selector|
  Register[ 1]:  0x00000142  |Event Code|
 Mask Info:                 |:L2_SHARED|Shared-state line from L2|
  Register[ 0]:  0x0000000f  |Event Selector|
  Register[ 1]:  0x00000242  |Event Code|
 Mask Info:                 |:L2_EXCLUSIVE|Exclusive-state line from L2|
  Register[ 0]:  0x0000000f  |Event Selector|
  Register[ 1]:  0x00000442  |Event Code|
```

PAPI Utilities: papi_event_chooser

```
$ utils/papi_event_chooser PRESET PAPI_FP_OPS
Event Chooser: Available events which can be added with given events.
--------------------------------------------------------------------------------
[…]
--------------------------------------------------------------------------------
    Name         Code     Deriv Description (Note)
PAPI_L1_DCM  0x80000000  No    Level 1 data cache misses
PAPI_L1_ICM  0x80000001  No    Level 1 instruction cache misses
PAPI_L2_ICM  0x80000003  No    Level 2 instruction cache misses
[…]
PAPI_L1_DCA  0x80000040  No    Level 1 data cache accesses
PAPI_L2_DCR  0x80000044  No    Level 2 data cache reads
PAPI_L2_DCW  0x80000047  No    Level 2 data cache writes
PAPI_L1_ICA  0x8000004c  No    Level 1 instruction cache accesses
PAPI_L2_ICA  0x8000004d  No    Level 2 instruction cache accesses
PAPI_L2_TCA  0x80000059  No    Level 2 total cache accesses
PAPI_L2_TCW  0x8000005f  No    Level 2 total cache writes
PAPI_FML_INS 0x80000061  No    Floating point multiply instructions
PAPI_FDV_INS 0x80000063  No    Floating point divide instructions
-------------------------------------------------------------------------
Total events reported: 34
event_chooser.c                         PASSED
```

PAPI Utilities: papi_event_chooser

```
$ utils/papi_event_chooser PRESET PAPI_FP_OPS PAPI_L1_DCM
Event Chooser: Available events which can be added with given events.
--------------------------------------------------------------------------------
[…]
--------------------------------------------------------------------------------


    Name         Code    Deriv Description (Note)
PAPI_TOT_INS 0x80000032  No   Instructions completed
PAPI_TOT_CYC 0x8000003b  No   Total cycles
-------------------------------------------------------------------------
Total events reported: 2
event_chooser.c                          PASSED
```

PAPI Utilities: papi_event_chooser

```
$ utils/papi_event_chooser NATIVE RESOURCE_STALLS:LD_ST X87_OPS_RETIRED
    INSTRUCTIONS_RETIRED
[…]
--------------------------------------------------------------------------------
UNHALTED_CORE_CYCLES    0x40000000
 |count core clock cycles whenever the clock signal on the specific core is running (not
    halted). Alias to event CPU_CLK_UNHALTED:CORE_P|
 |Register Value[0]: 0x20003       Event Selector|
 |Register Value[1]: 0x3c          Event Code|
-------------------------------------------------------------------------
UNHALTED_REFERENCE_CYCLES       0x40000002
 |Unhalted reference cycles. Alias to event CPU_CLK_UNHALTED:REF|
 |Register Value[0]: 0x40000       Event Selector|
 |Register Value[1]: 0x13c         Event Code|
-------------------------------------------------------------------------
CPU_CLK_UNHALTED        0x40000028
 |Core cycles when core is not halted|
 |Register Value[0]: 0x60000       Event Selector|
 |Register Value[1]: 0x3c          Event Code|
    0x40001028  :CORE_P  |Core cycles when core is not halted|
    0x40008028  :NO_OTHER  |Bus cycles when core is active and the other is halted|
-------------------------------------------------------------------------
Total events reported: 3
event_chooser.c                             PASSED
```

TAU Workflow

- Sampling to determine what to look at more closely
  - `tau_exec -ebs`
- Storage
  - `paraprof -pack file.ppk`
  - `taudb_loadtrial`
- Visualization
  - `paraprof`
- Selective Instrumentation
  - `tau_cc.sh, tau_cxx.sh, tau_f90.sh`
  - `export TAU_OPTIONS="-optSelectFile=path"`
  - (see https://www.cs.uoregon.edu/research/tau/docs/newguide/bk01ch01s03.html for syntax)
- Hardware performance counters
  - `papi_avail` to determine what's available
  - `papi_event_chooser` to determine what's compatible
  - `export TAU_METRICS=TIME:PAPI_L2_TCM:PAPI_L2_TCA`
- Derived metrics
  - `paraprof`

Python Performance Evaluation

HANDS-ON

Getting Started with TAU

– Series of exercises available at:

https://fs.paratools.com/TAU_SDL_examples.tar.gz

Example 1: C Matmult (MPI + Pthreads) Source Instrumentation

– First, we will install a compatible configuration of PDT and TAU:

```
wget http://tau.uoregon.edu/pdt.tar.gz
tar xzf pdt.tar.gz
cd pdtoolkit-3.25
./configure
make install # installs into current directory

cd ..
wget http://tau.uoregon.edu/tau.tgz
tar xzf tau.tgz
cd tau-2.27.2p1
./configure -bfd=download -unwind=download -arch=craycnl –pdt=<path to PDT>/pdtoolkit-3.25 -pdt_c++=/usr/bin/g++ -mpi -pthread
make install

export PATH=<path to TAU>/tau-2.27.2p1/craycnl/bin:$PATH
```

Example 1: C Matmult (MPI + Pthreads) Source Instrumentation

```
$ cd workshop-python/01_matmult.c
$ make CC=tau_cc.sh
```

Run normally to generate profiles:

```
$ aprun –n 4 –N 4 ./matmult
$ ls profile.*          # Shows four files
$ paraprof --pack mm_c_flat.ppk
```

View the profiles:

```
pprof –a | less         #Command line
paraprof                #GUI (Java, X11)
```

Example 2: Fortran Matmult (MPI)

```
$ cd workshop-python/02_matmult.f90
$ make F90=tau_f90.sh
```

Run normally to generate profiles:

```
$ aprun -n 4 -N 4 ./matmult
$ ls profile.*          # Shows four files
$ paraprof --pack mm_f90_flat.ppk
```

View the profiles:

```
pprof -a | less         #Command line
paraprof                #GUI (Java, X11)
```

Basic TAU Workflow

Choose your TAU_MAKEFILE:

– `$ export TAU_MAKEFILE=$TAU/Makefile.tau-mpi-python-pdt`

Use tau_f90.sh, tau_cxx.sh, etc. as compiler:

– `$ ftn foo.f90`

changes to

`$ tau_f90.sh foo.f90`

Edit Makefile or set compilers on command line:

`$ make CC=tau_cc.sh`

Execute application

Analyze performance data:

– `pprof` (for text based profile display)

– `paraprof` (for GUI)

Example 3: TAU with Pure Python

– Build a Python configuration of TAU:

```
module load miniconda-3.6/conda-4.5.4
cd <TAU directory>
./configure -bfd=download -unwind=download -arch=craycnl –python
make install
```

FIXEDGRID

A simple chemical transport model in Python

$$\frac{\partial \mathbf{c}_x^t}{\partial t} = \sum_{k=1}^{d} \left[\frac{\partial}{\partial x_k} \left(d_k(x,t) \frac{\partial \mathbf{c}_x^t}{\partial x_k} - a_k(x,t) \mathbf{c}_x^t \right) \right] + F$$

Advection: Upwind-biased 2nd order finite differences
Diffusion: 3rd order finite differences
Chemistry: Rosenbrock time-stepping integrator

TAU with Pure Python

```
$ cd workshop-python/03_fixedgrid.py
```

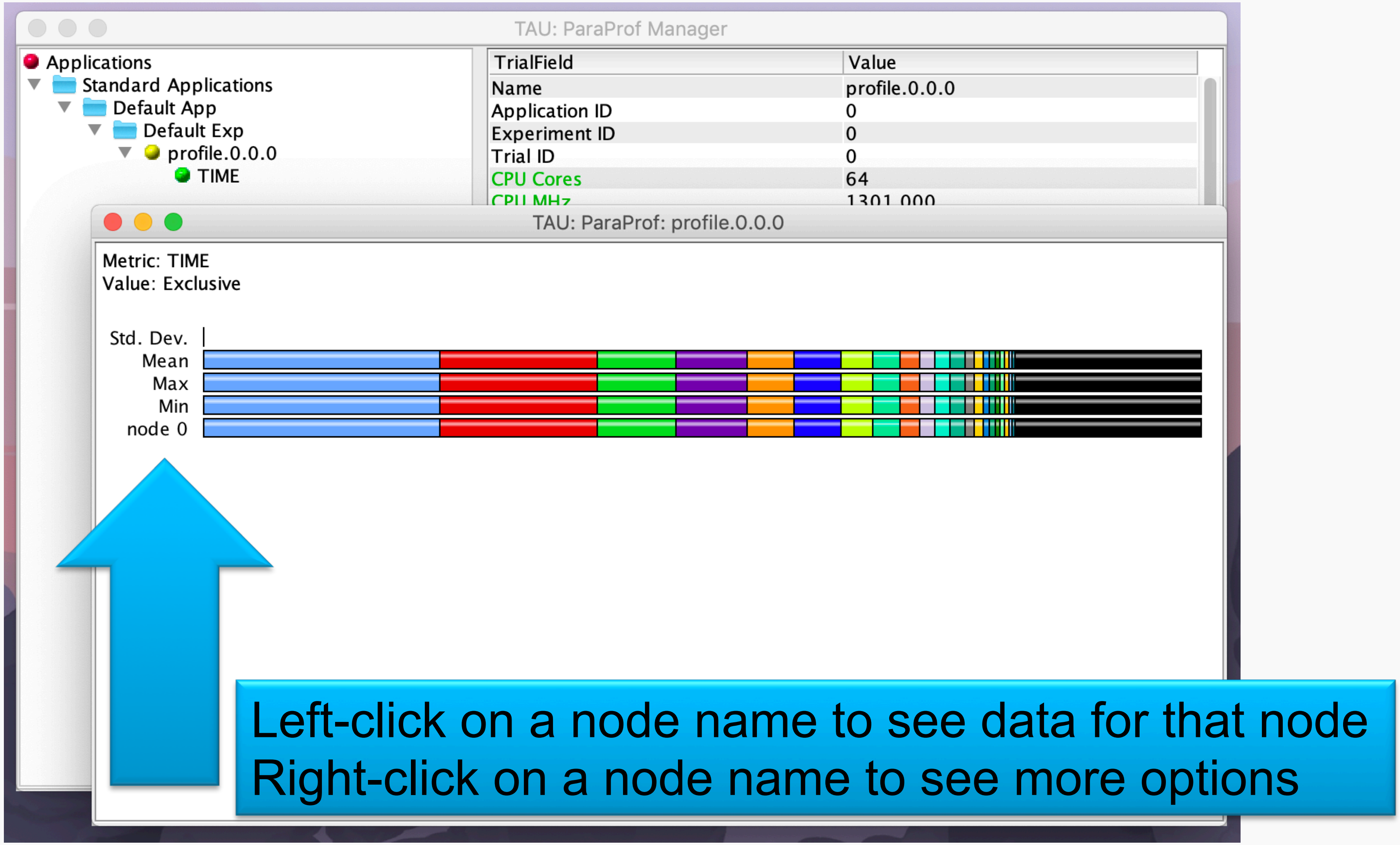
Run with tau_python to generate profiles:

```
$ export TAU_CALLPATH=1 # Generate callpath profiles
$ aprun -n 1 –N 1 tau_python –T serial,intel,python fixedgrid.py
$ ls profile.*          # shows profile.0.0.0
$ paraprof --pack fixedgrid_py_flat.ppk
```

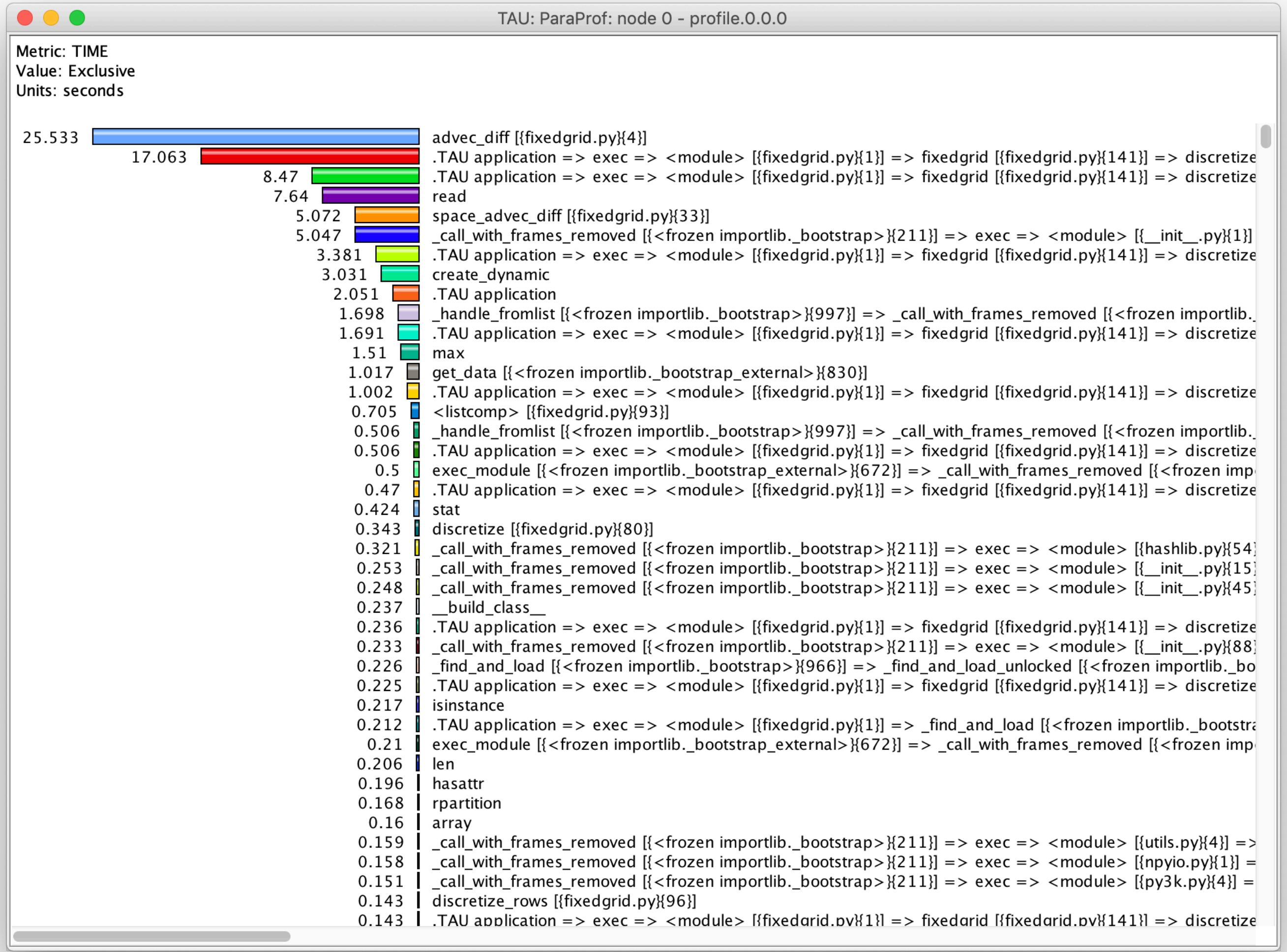
View the profiles:

```
$ pprof –a | less       #Command line
$ paraprof              #GUI (Java, X11)
```

ParaProf Profile Visualizer

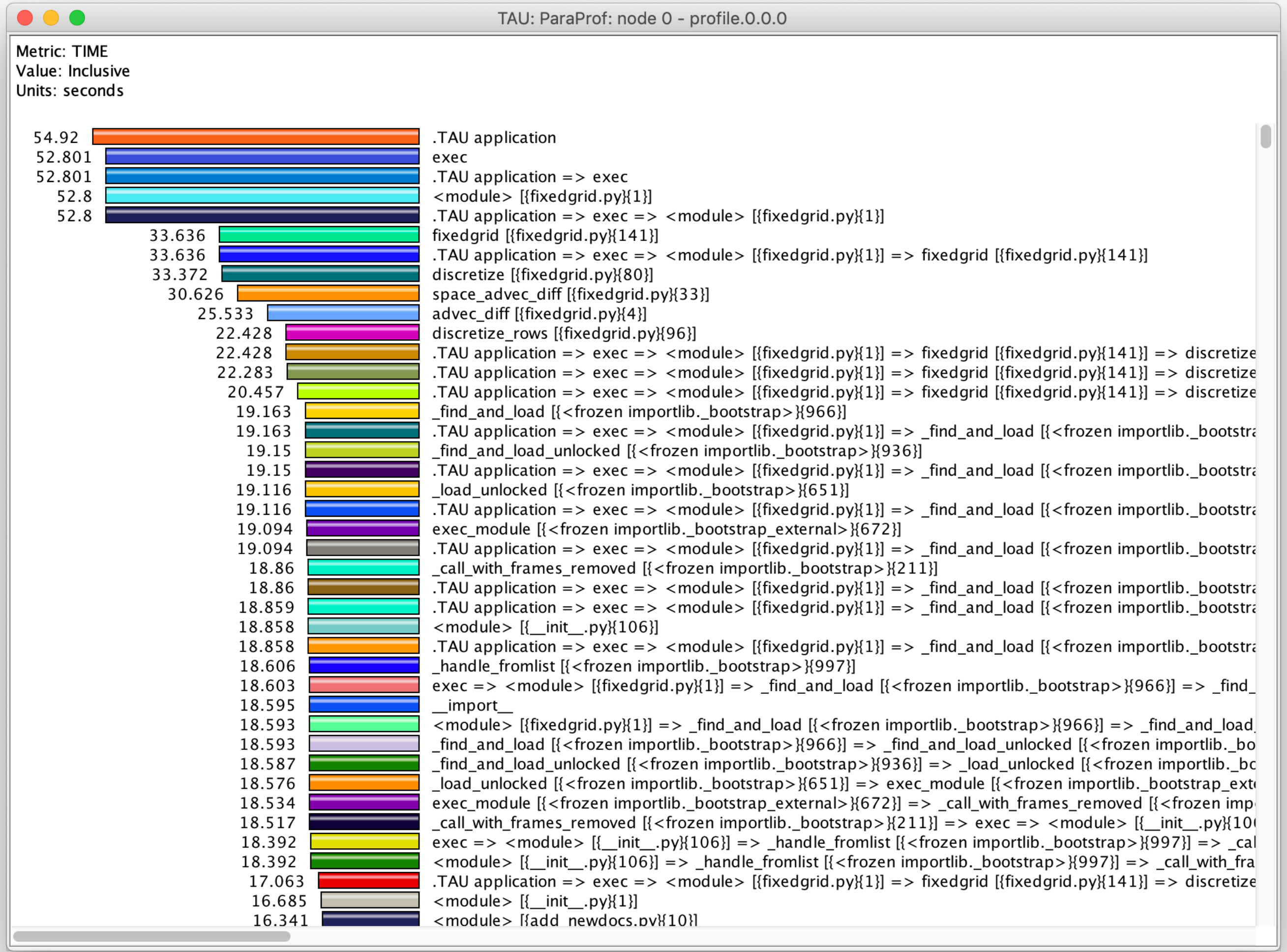


Exclusive Time in ParaProf



Argonne NATIONAL LABORATORY

Inclusive Time in ParaProf



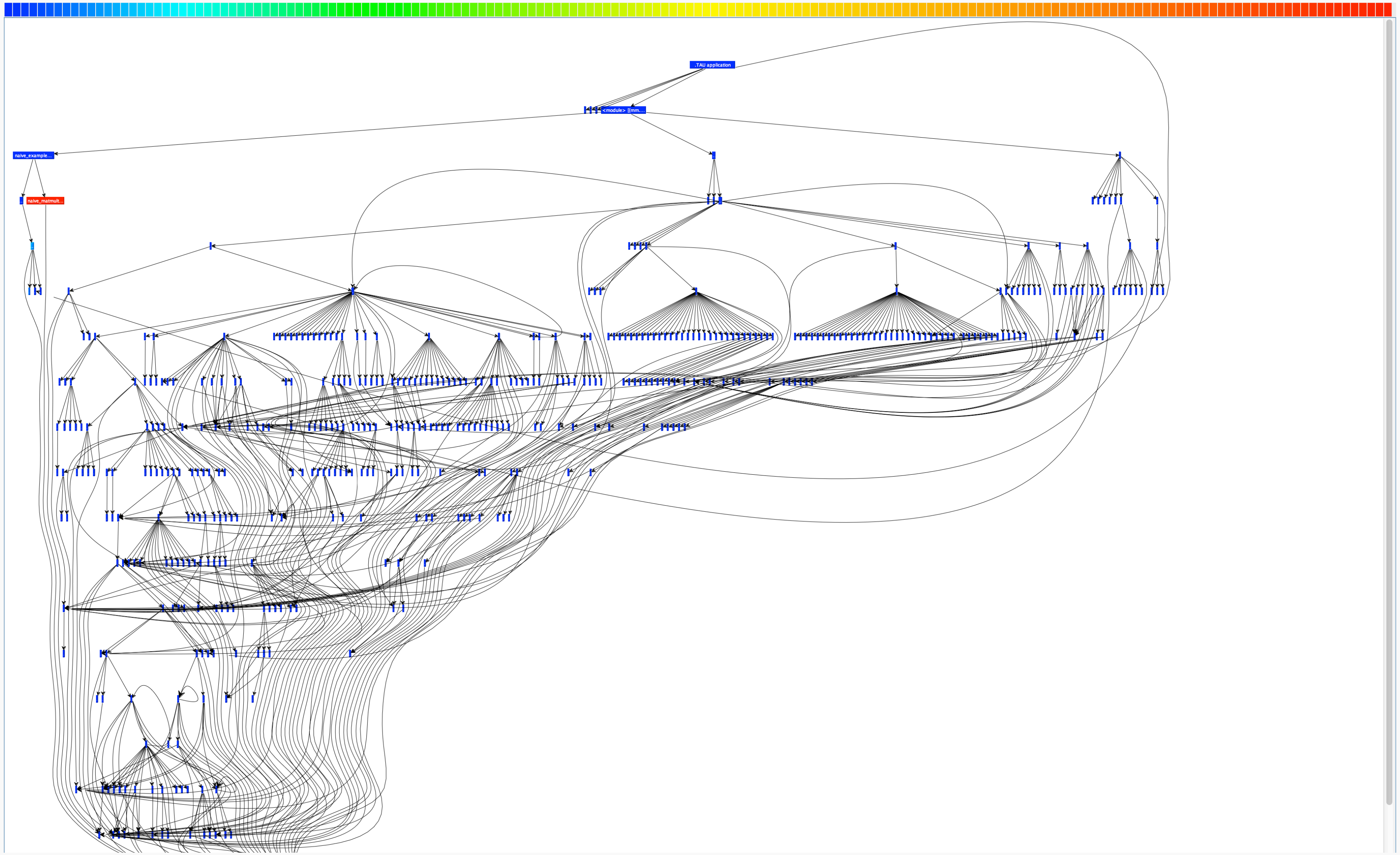
Statistics Table in ParaProf

TAU: ParaProf: Statistics for: node 0 - profile.0.0.0

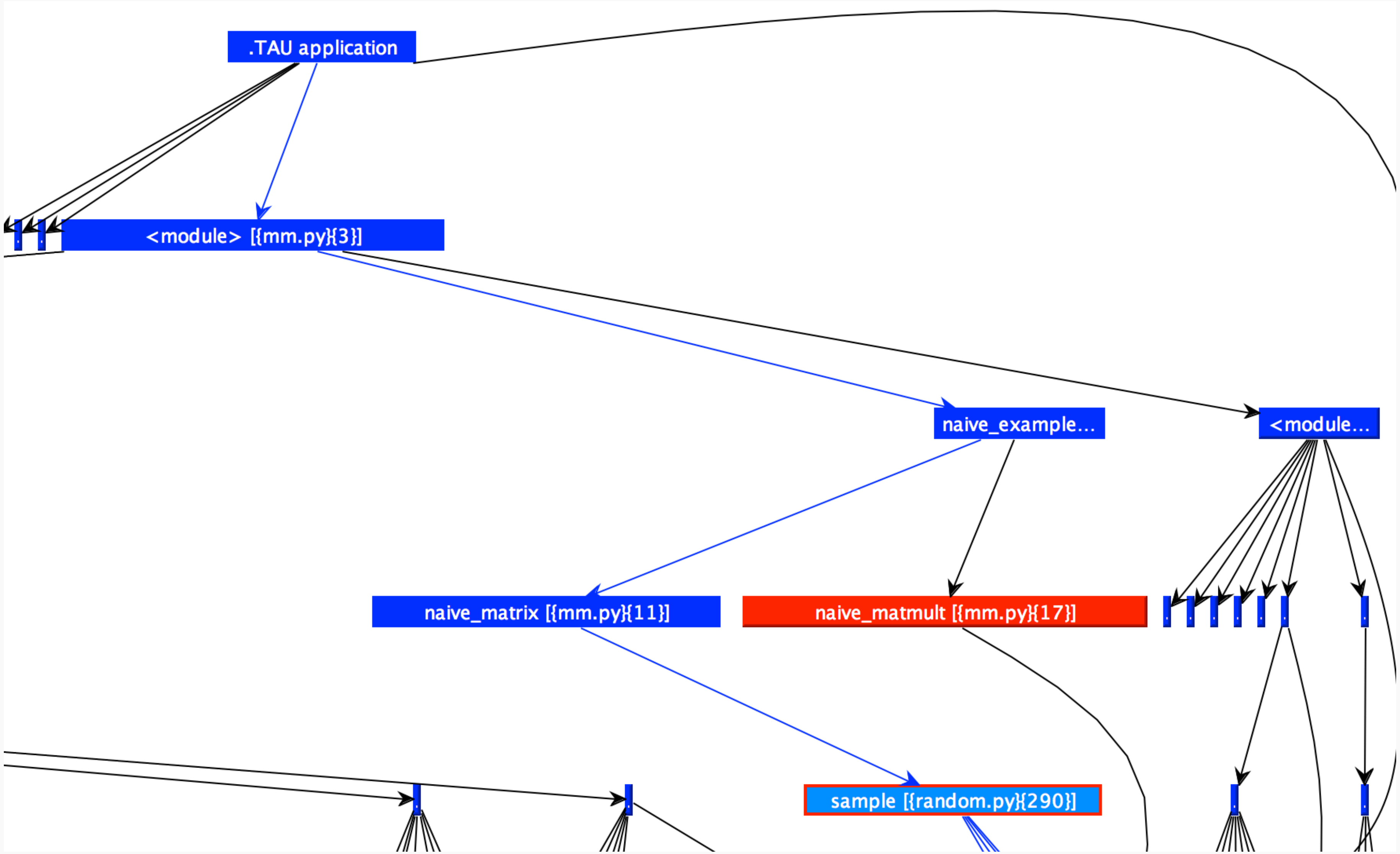
Name	Exclusive TIME	Inclusive TIME ▽	Calls	Child C...
.TAU application	2.051	54.92	1	5
exec	0.001	52.801	1	1
<module> [{fixedgrid.py}{1}]	0.001	52.8	1	2
fixedgrid [{fixedgrid.py}{141}]	0.004	33.636	1	79
discretize_rows [{fixedgrid.py}{96}]	0.143	22.428	36	1,908
discretize [{fixedgrid.py}{80}]	0.225	22.283	1,800	10,800
space_advec_diff [{fixedgrid.py}{33}]	3.381	20.457	3,600	183,600
advec_diff [{fixedgrid.py}{4}]	17.063	17.063	180,000	0
len	0.014	0.014	3,600	0
<listcomp> [{fixedgrid.py}{93}]	0.47	1.471	1,800	90,000
copy [{function_base.py}{1461}]	0.028	0.113	3,600	3,600
empty_like	0.016	0.016	1,800	0
empty	0.003	0.003	108	0
discretize_cols [{fixedgrid.py}{117}]	0.074	11.198	18	1,854
discretize [{fixedgrid.py}{80}]	0.117	11.089	900	5,400
space_advec_diff [{fixedgrid.py}{33}]	1.691	10.168	1,800	91,800
advec_diff [{fixedgrid.py}{4}]	8.47	8.47	90,000	0
len	0.007	0.007	1,800	0
<listcomp> [{fixedgrid.py}{93}]	0.236	0.741	900	45,000
copy [{function_base.py}{1461}]	0.014	0.054	1,800	1,800
empty_like	0.008	0.008	900	0
copy [{function_base.py}{1461}]	0.009	0.033	900	900
empty	0.001	0.001	54	0
full [{numeric.py}{254}]	0.001	0.003	4	12
print	0.003	0.003	21	0
_find_and_load [{<frozen importlib._bootstrap>}{966}]	0.002	19.163	1	6
find_module [{imp.py}{255}]	0.005	0.045	1	21
compile	0.019	0.019	1	0
read	0.002	0.004	1	1
new_module [{imp.py}{48}]	0.001	0.001	1	0

Argonne NATIONAL LABORATORY

Callgraph in ParaProf



Callgraph in ParaProf



Traces with Pure Python

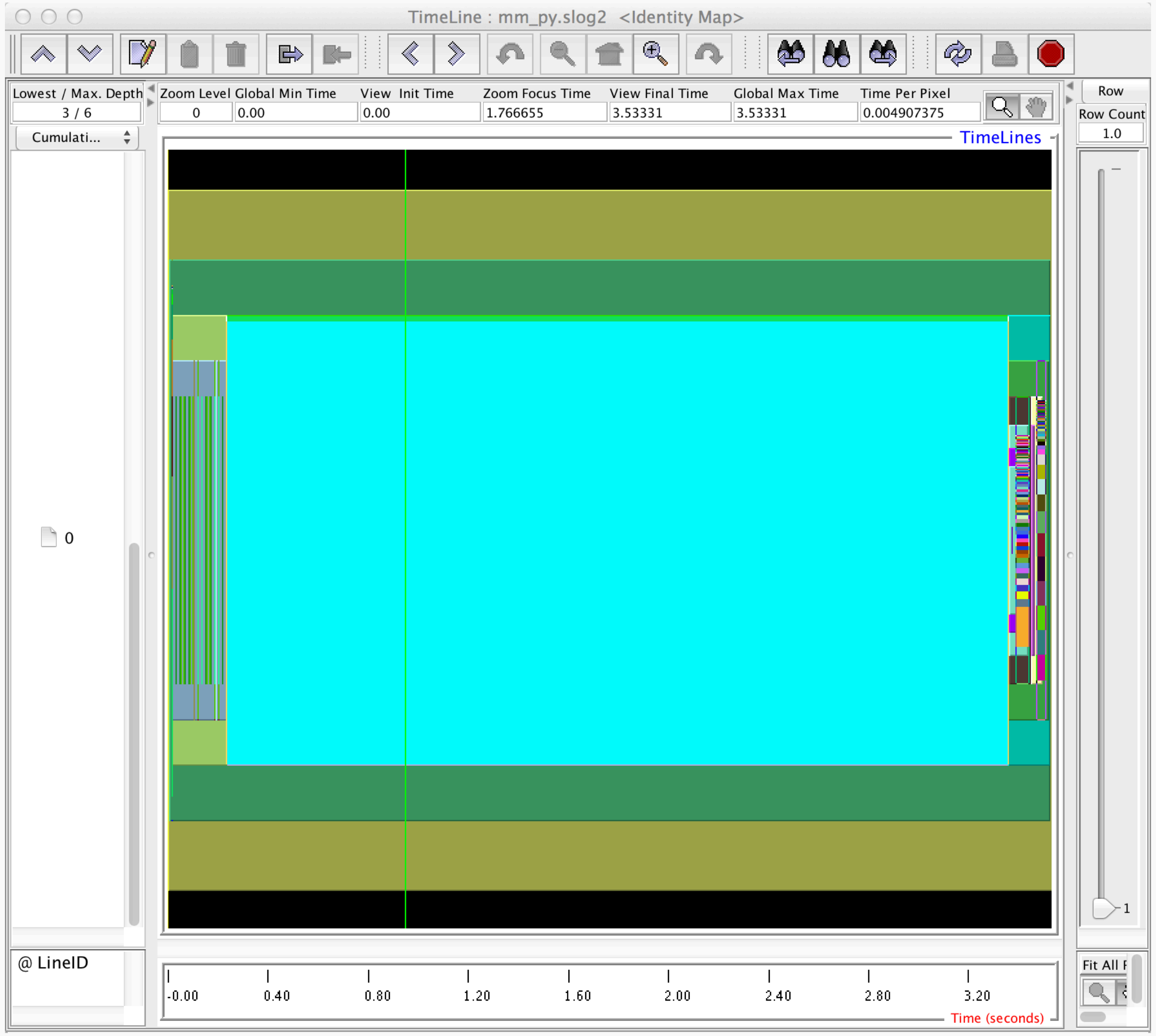
To generate traces:

```
$ unset TAU_CALLPATH   #recommended
$ export TAU_TRACE=1
$ aprun -n 1 –N 1 tau_python –T serial,intel,python fixedgrid.py
```

Trace files must be post-processed:

```
$ tau_treemerge.pl
$ tau2slog2 tau.trc tau.edf –o \
      mm_py.slog2
$ jumpshot mm_py.slog2
```

Jumpshot Trace Viewer

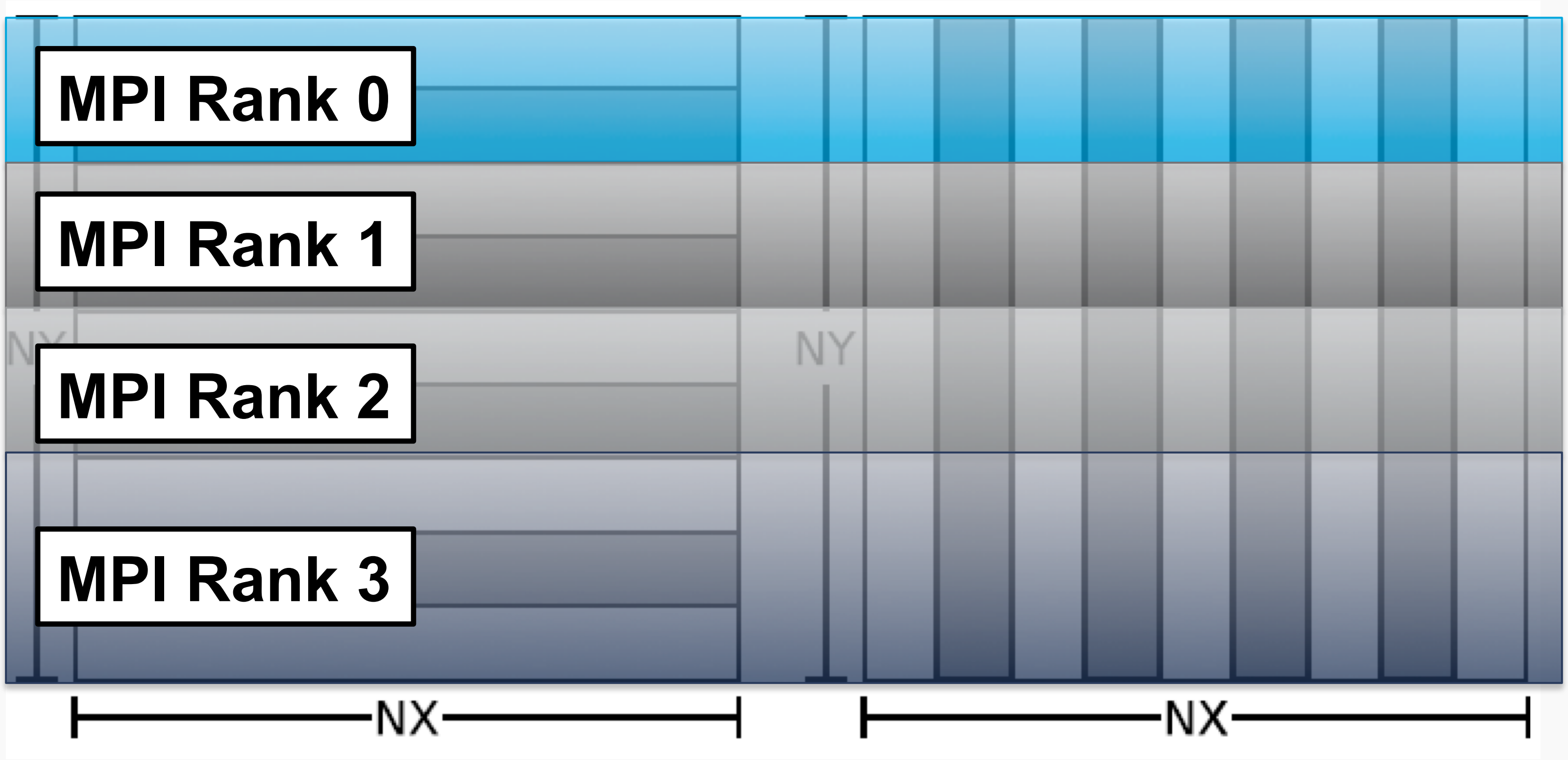


Public Service Annoucement

Don’t forget to clean your environment!
(Some folks write scripts)

Show all TAU environment variables:
$ env | grep TAU
Unset the ones you don’t need anymore:
$ unset TAU_TRACE
$ unset TAU_CALLPATH
etc.

MPI in FIXEDGRID



Example 4: TAU with Python + MPI

– Build a Python+MPI configuration of TAU:

```
module load miniconda-3.6/conda-4.5.4

cd <TAU directory>

./configure -bfd=download -unwind=download -arch=craycnl –python
–mpi

make install
```

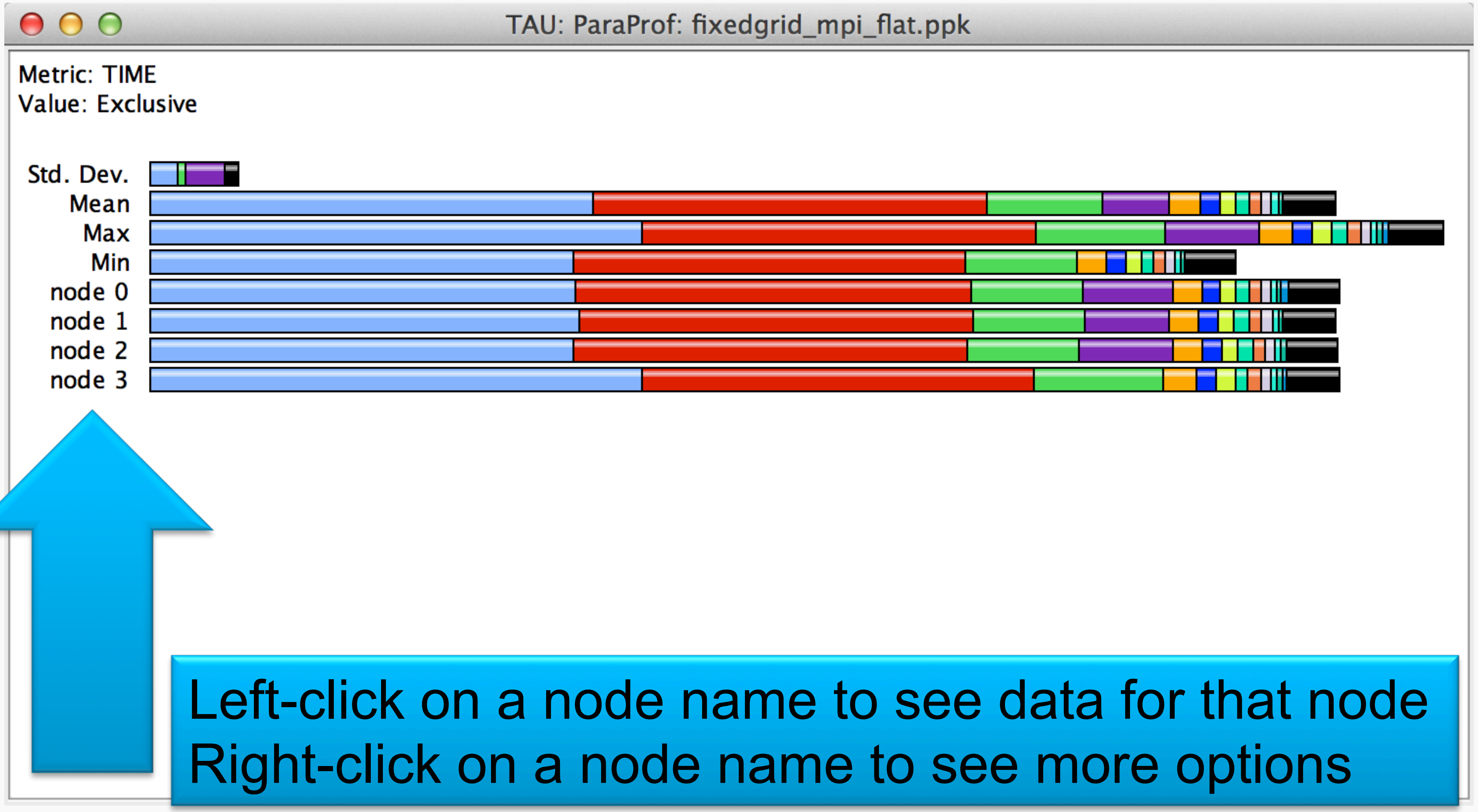
TAU with mpi4py

```
$ cd 04_fixedgrid-mpi.py
$ aprun -n 4 -N 4 tau_python –T mpi,intel,python fixedgrid.py
```

View the profiles:

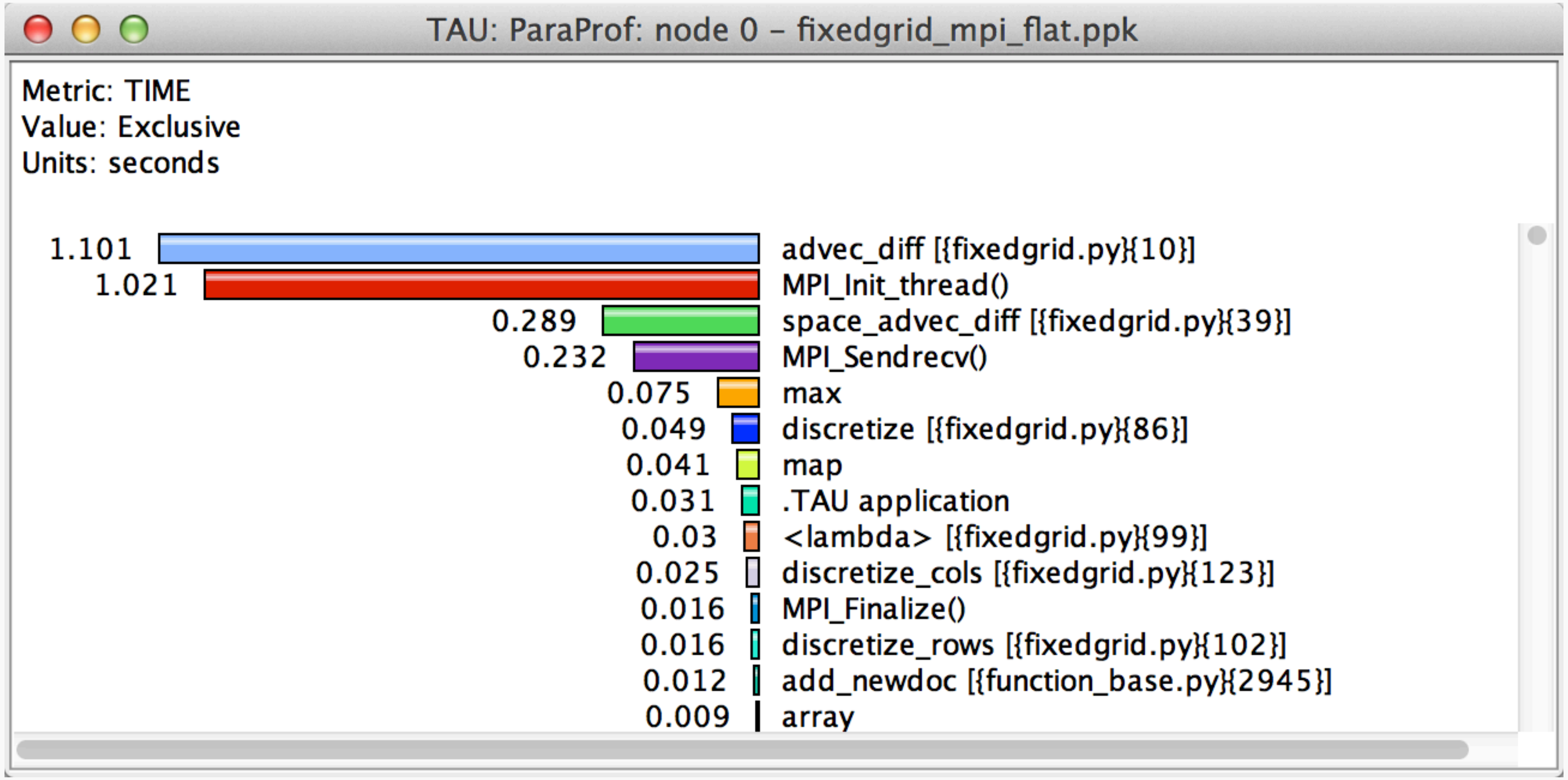
```
pprof –a | less        #Command line
paraprof               #GUI (Java, X11)
```

FIXEDGRID Profile



Left-click on a node name to see data for that node
Right-click on a node name to see more options

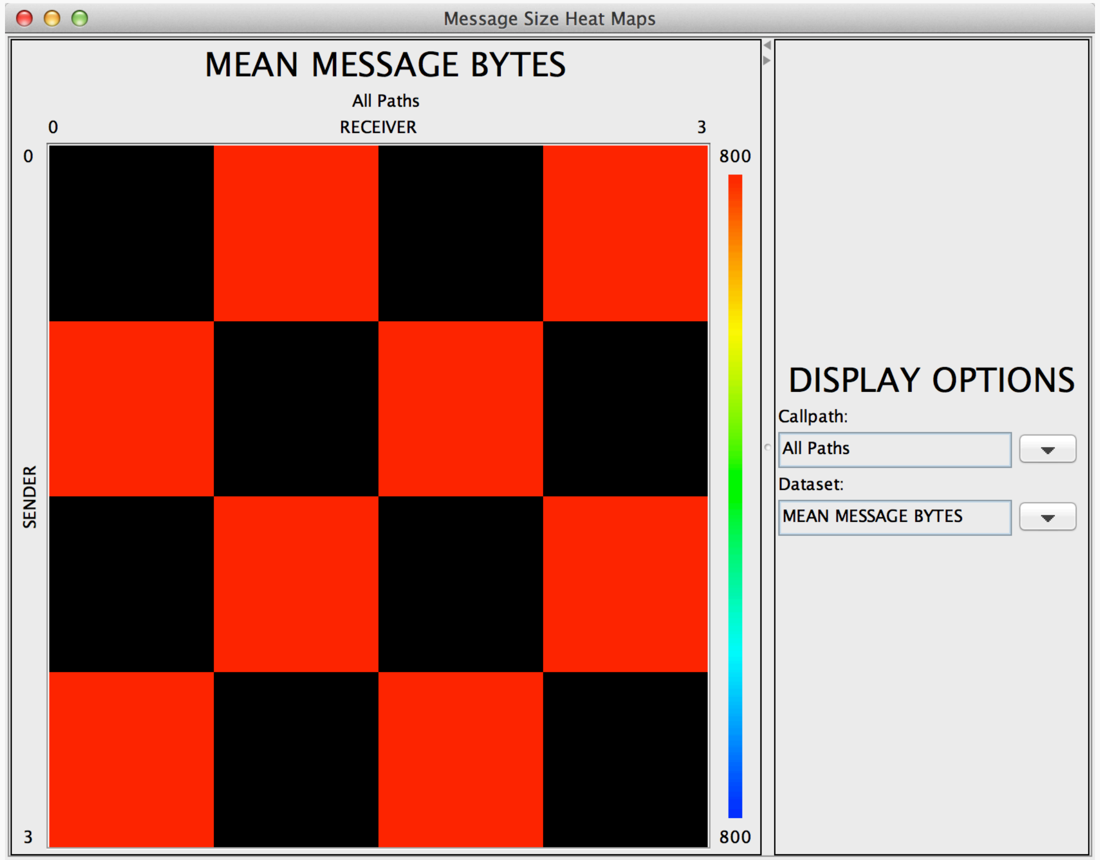
FIXEDGRID Profile



FIXEDGRID Communication Matrix

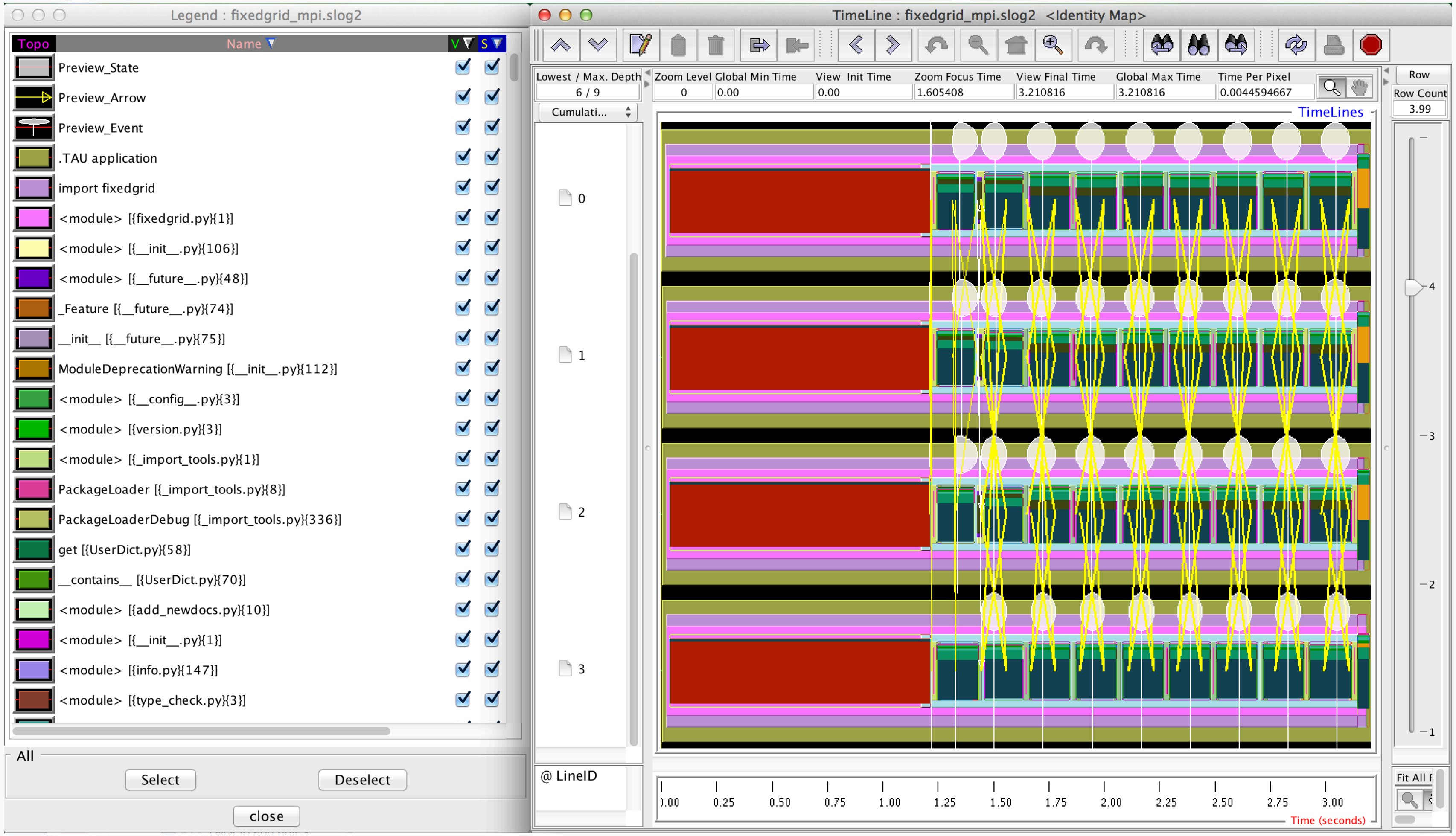
```
$ export TAU_COMM_MATRIX=1
$ aprun -n 4 -N 4 tau_python -T mpi,intel,python fixedgrid.py
```

In Paraprof: Windows | Communication Matrix



FIXEDGRID Trace Shows Communication

$ jumpshot fixedgrid_mpi.slog2



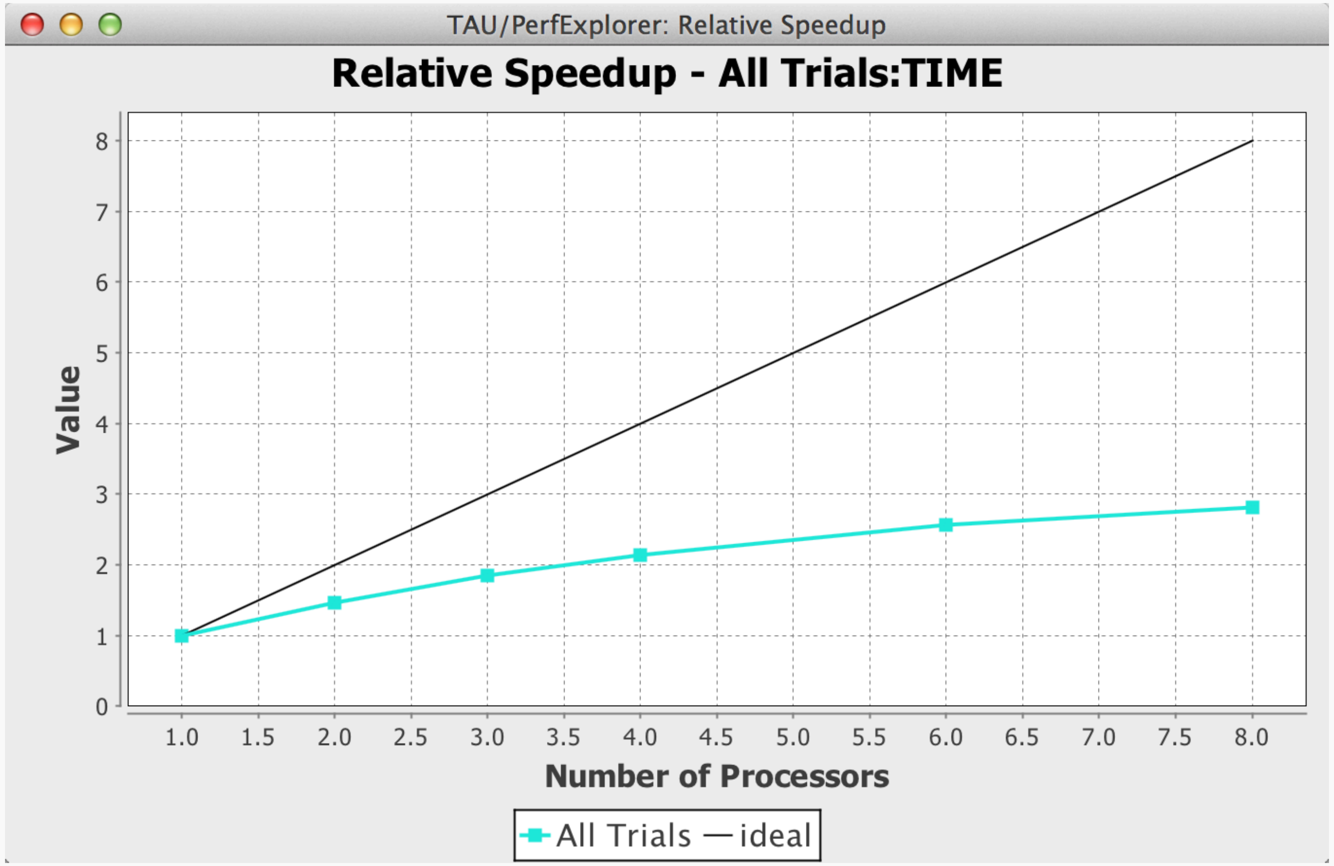
Argonne NATIONAL LABORATORY

PerfExplorer

```
$ cd 04_fixedgrid-mpi.py/analysis
$ taudb_configure --create-default
$ taudb_loadtrial fixedgrid_np1.ppk
$ taudb_loadtrial fixedgrid_np2.ppk
$ taudb_loadtrial fixedgrid_np3.ppk
…
$ perfexplorer
```

Relative Speedup Chart

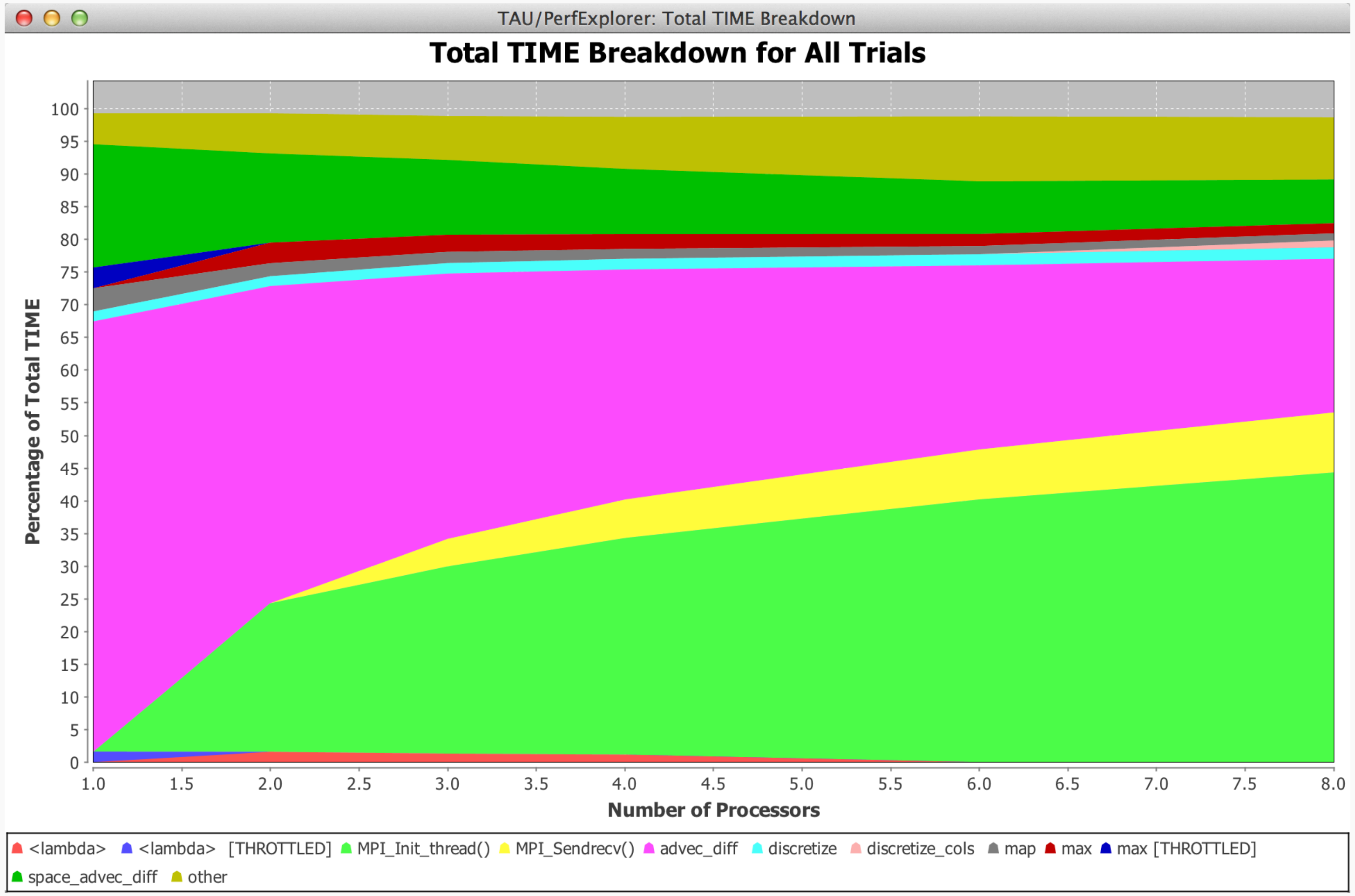
In PerfExplorer: Charts | Relative Speedup



Argonne NATIONAL LABORATORY

Runtime Breakdown Chart

In PerfExplorer: Charts | Runtime Breakdown



Example 5: TAU + Python + mpi4py + C + OpenMP

– Build a Python+MPI+OpenMP configuration of TAU:

```
module load miniconda-3.6/conda-4.5.4

cd <TAU directory>

./configure -bfd=download -unwind=download -arch=craycnl –python
–mpi –ompt=download

make install
```

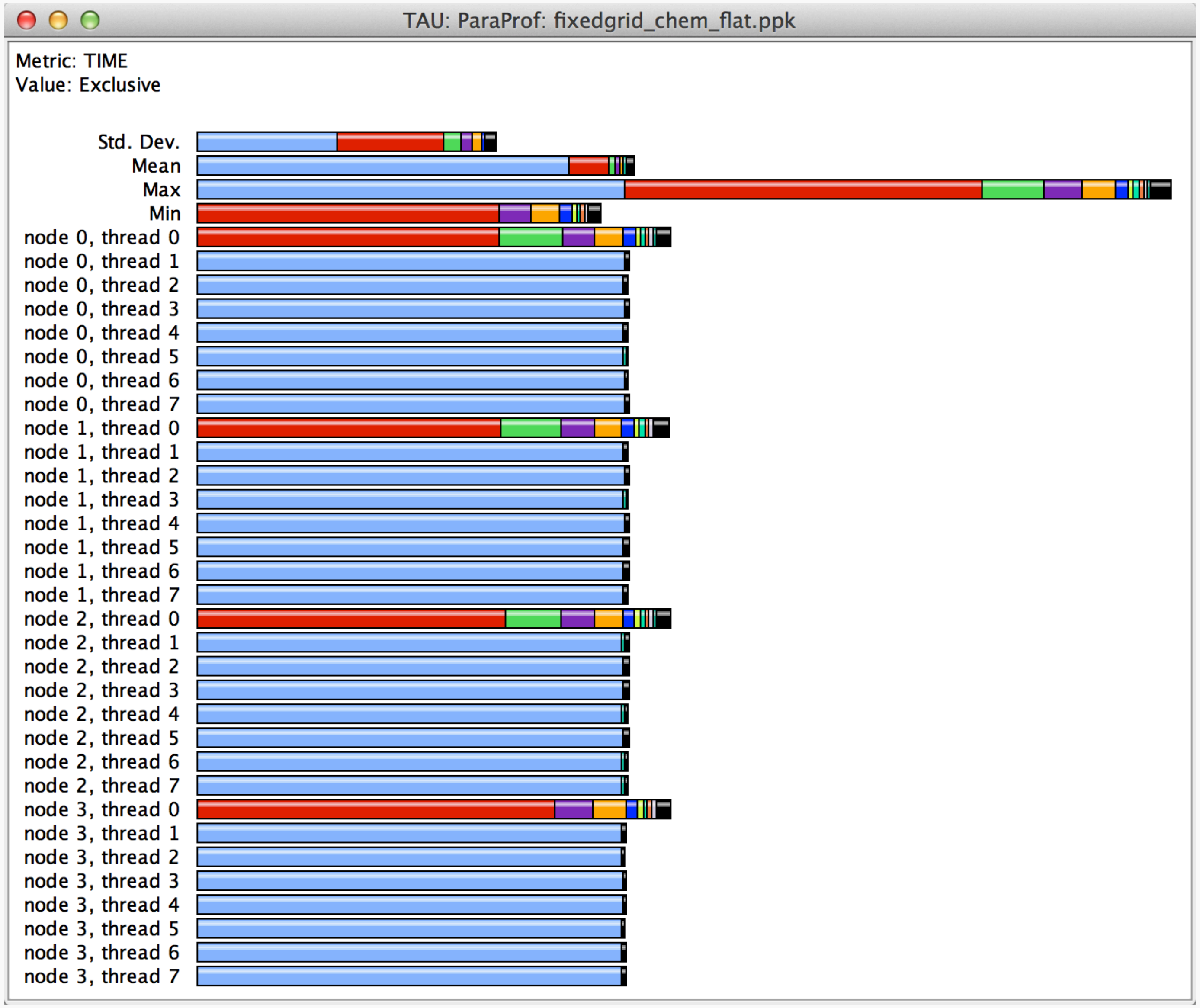
Example 5: TAU + Python + mpi4py + C + OpenMP

```
$ cd 05_fixedgrid-chem.c_py
$ export TAU_MAKEFILE=<path to Makefile from install
step>
$ make CC=tau_cc.sh
```

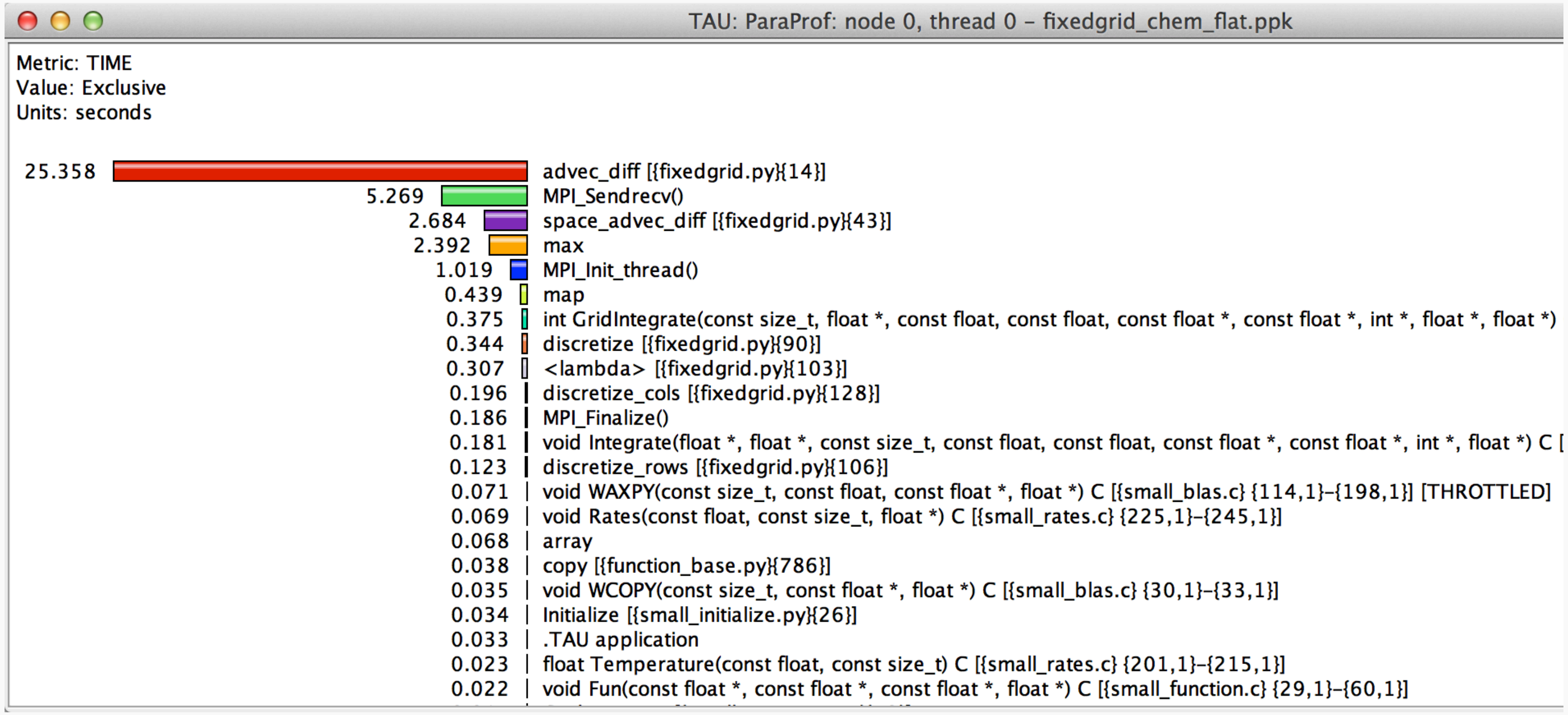
Run with tau_exec and wrapper.py to generate profiles:

```
$ make clean
$ make CC=tau_cc.sh
$ aprun -n 4 –N 4 tau_python -T
python,mpi,openmp,intel,ompt,tr6 -ompt fixedgrid.py
```

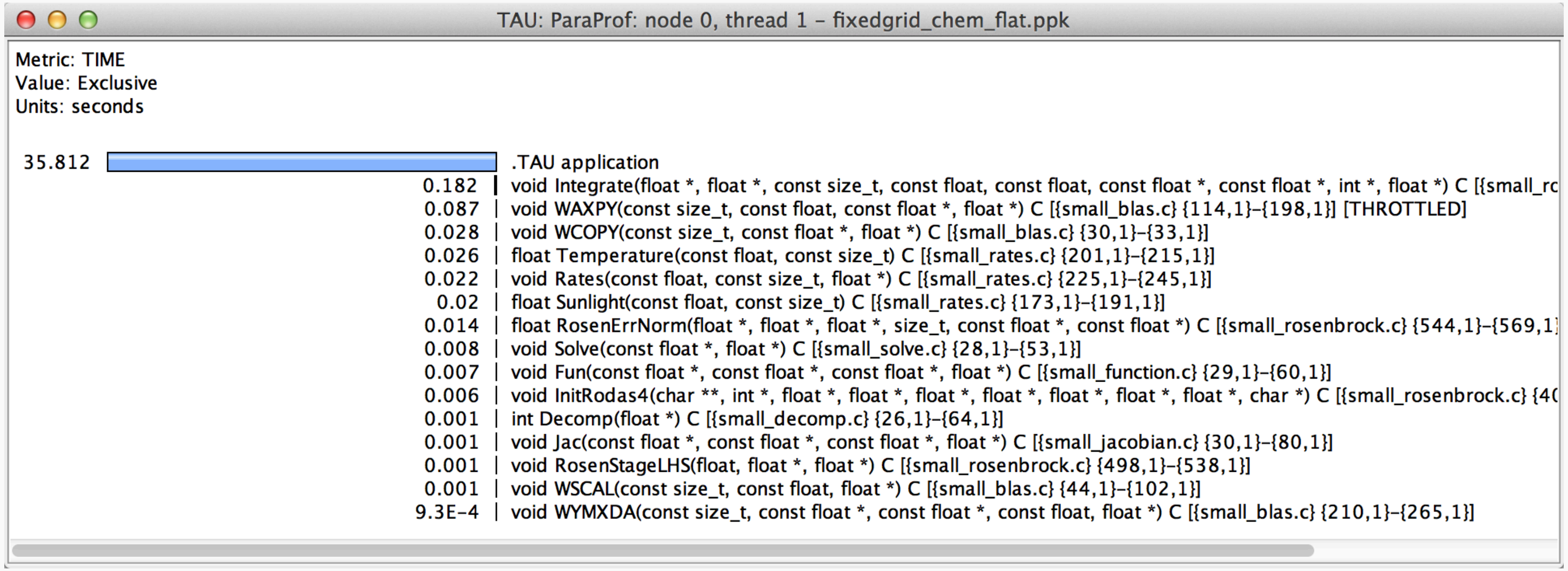
MPI + OpenMP Profiles



Rank 0, Thread 0



Rank 0, Thread 1



Download TAU from U. Oregon

http://tau.uoregon.edu

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

Free download, open source, BSD license

Questions or Problems?

support@paratools.com

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)
- HPCMO

National Science Foundation (NSF)
- Glassbox, SI-2

Partners:

University of Oregon

ParaTools, Inc.

University of Tennessee, Knoxville

T.U. Dresden, GWT

Juelich Supercomputing Center