The TAU Performance System

ParaTools, Inc.

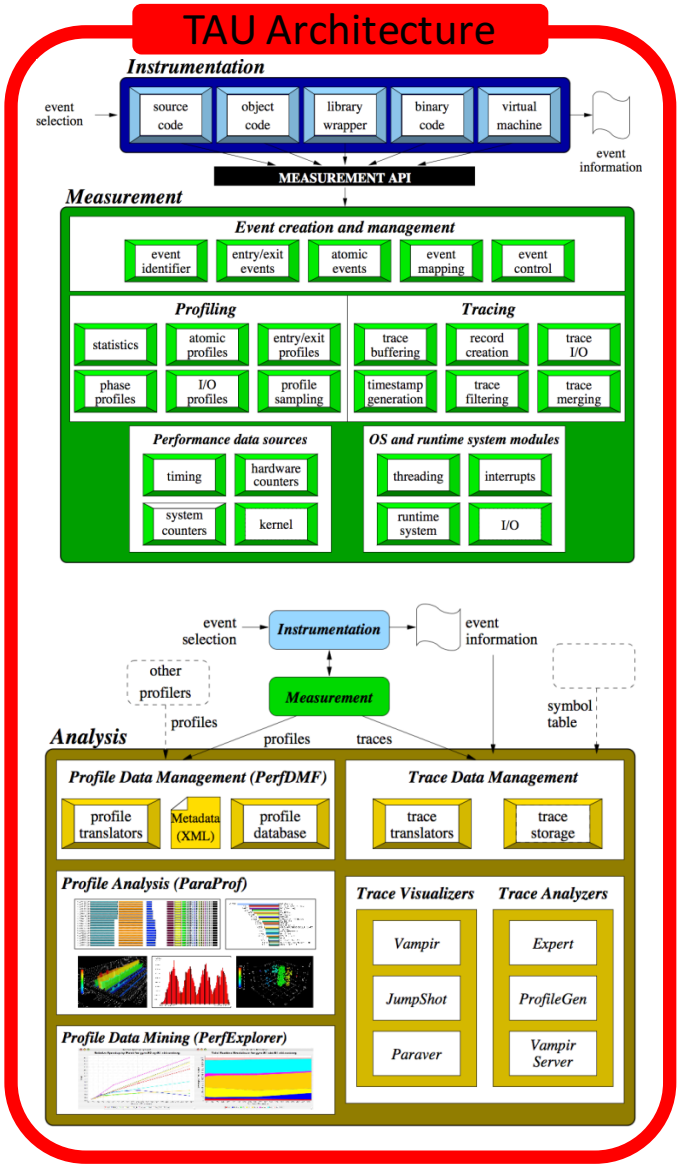
John C. Linford, Sameer Shende, et al.
{jlinford,sameer}@paratools.com

Scaling Your Science on Mira
24 May 2016, ALCF

ParaTools

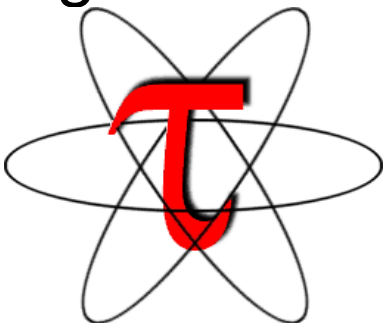
The TAU Performance System®

- *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/



The TAU Performance System®

- Tuning and Analysis Utilities (**20+ 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

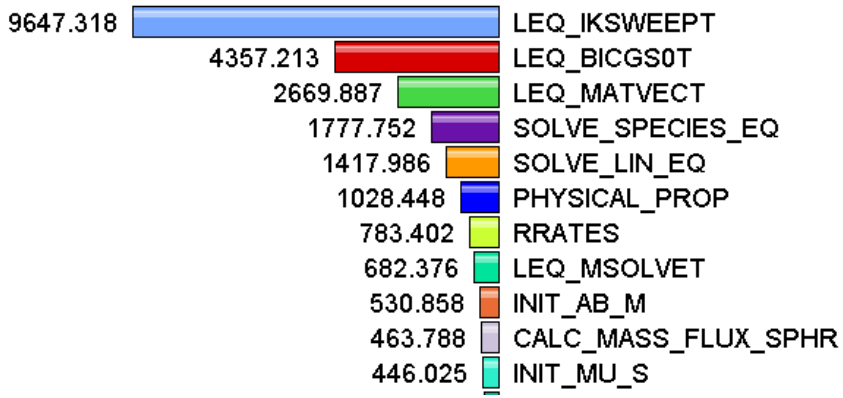


TAU Supports All HPC Platforms

C/C++
CUDA
UPC
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
Android
MPC
OS X

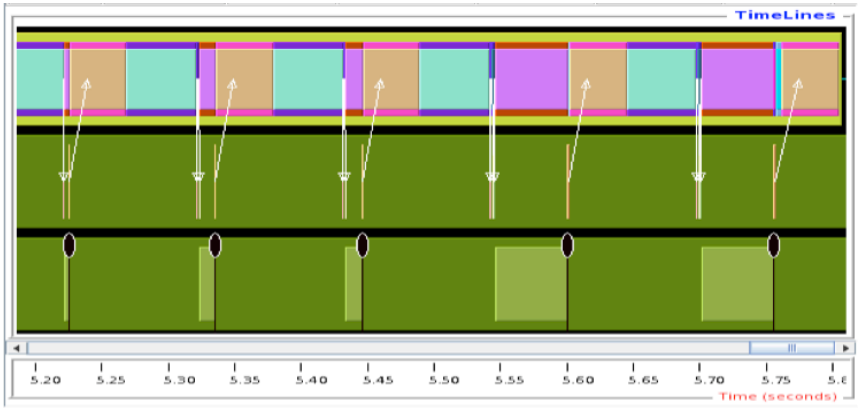
Measurement Approaches

Profiling



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

Tracing

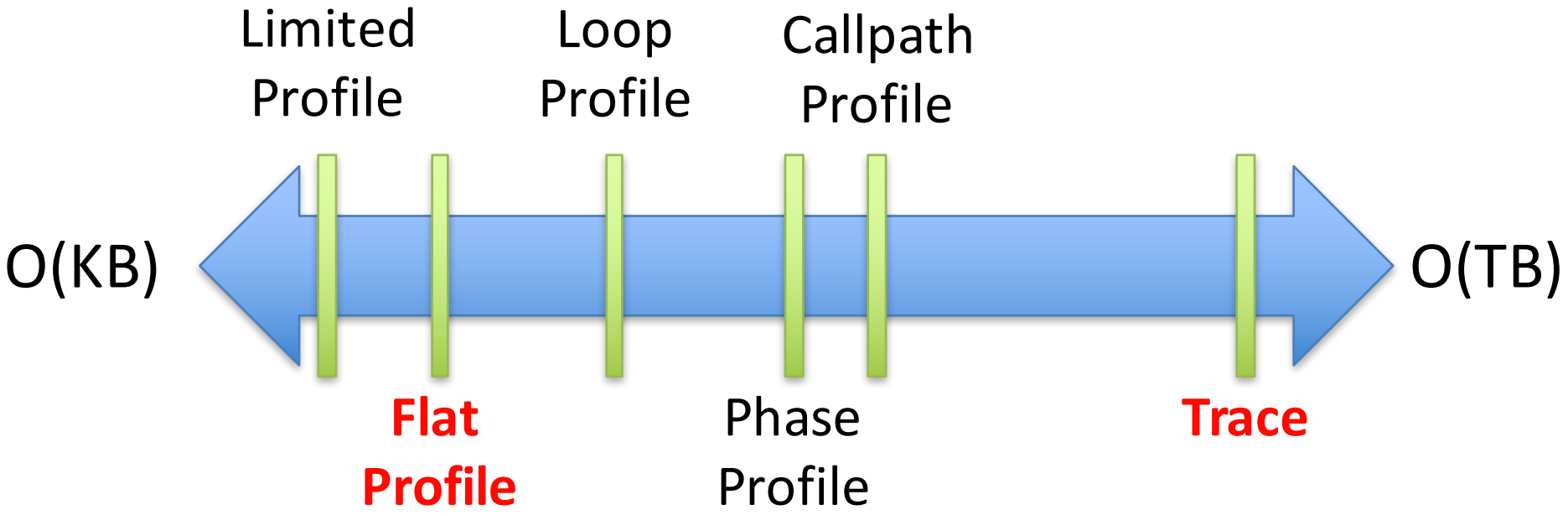


Shows **when** events take place on a timeline

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

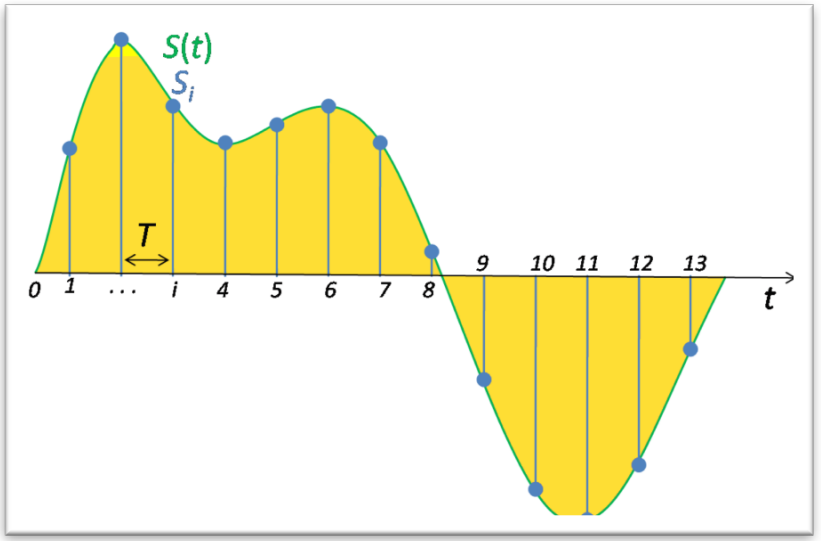
Performance Data Measurement

Direct via Probes

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

- Exact measurement
- Fine-grain control
- Calls inserted into code

Indirect via Sampling



- No code modification
- Minimal effort
- Relies on debug symbols (**-g** option)

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.

Makefile without TAU

```
CXX = mpicxx
F90 = mpif90
CXXFLAGS =
LIBS = -lm
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 $<
```

ALCF Quick Start

1. Choose an appropriate `TAU_MAKEFILE`:

```
% soft add +tau-latest
% export TAU_MAKEFILE=/soft/perftools/tau/tau_latest/
      bgq/lib/Makefile.tau-bgqtimers-mpi-pdt
% export TAU_OPTIONS='-optVerbose …'
    # (see tau_compiler.sh -help for more options)
```

2. Use `tau_f90.sh`, `tau_cxx.sh`, etc. as Fortran, C++, etc. compiler:

```
% mpixlf90_r foo.f90
```

changes to

```
% tau_f90.sh foo.f90
```

3. Execute application:

```
% qsub –A <queue> –q R.bc –n 256 –t 10 ./a.out
```

Note: If TAU_MAKEFILE has "papi" in its name, set **TAU_METRICS**:

```
% qsub --env TAU_METRICS=BGQ_TIMERS:PAPI_L2_DCM...
```

4. Analyze performance data:

`pprof` (for text based profile display)

`paraprof` (for GUI)

TAU Configurations on Mira

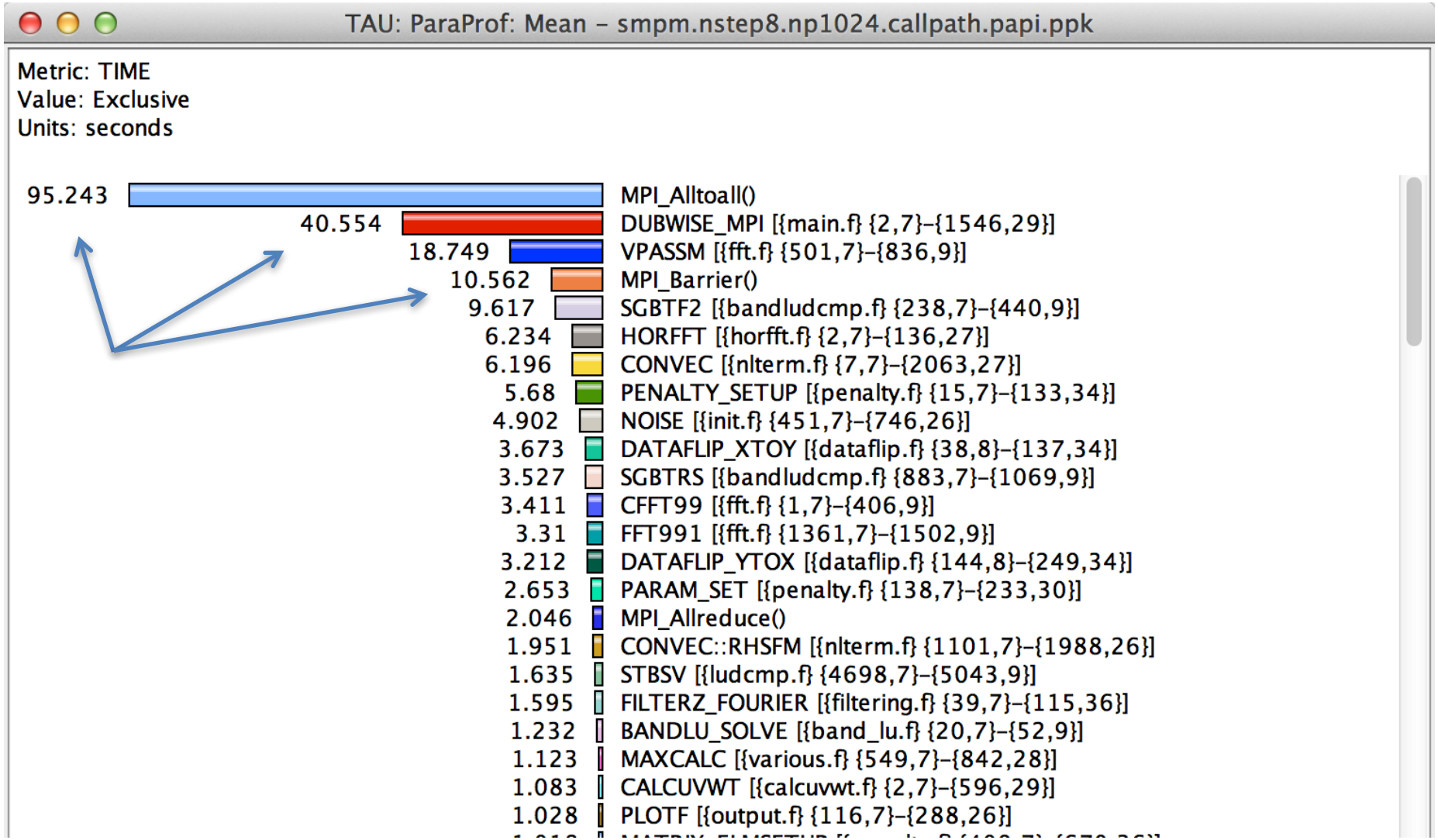
- Each configuration of TAU corresponds to a unique stub makefile (*TAU_MAKEFILE*) in the TAU installation directory

```
% ls /soft/perftools/tau/tau_latest/bgq/lib/Makefile.*
Makefile.tau-bgqtimers-mpi-pdt-openmp-opari
Makefile.tau-bgqtimers-mpi-pthread-pdt
Makefile.tau-bgqtimers-papi-mpi-pdt
Makefile.tau-bgqtimers-papi-mpi-pdt-openmp-opari
Makefile.tau-bgqtimers-papi-mpi-pthread-pdt
Makefile.tau-bgqtimers-pdt
Makefile.tau-papi-mpi-pdt-openmp-opari
Makefile.tau-papi-mpi-pdt-openmp-opari-scorep
Makefile.tau-papi-mpi-pdt-scorep
```

Scaling Your Science on Mira

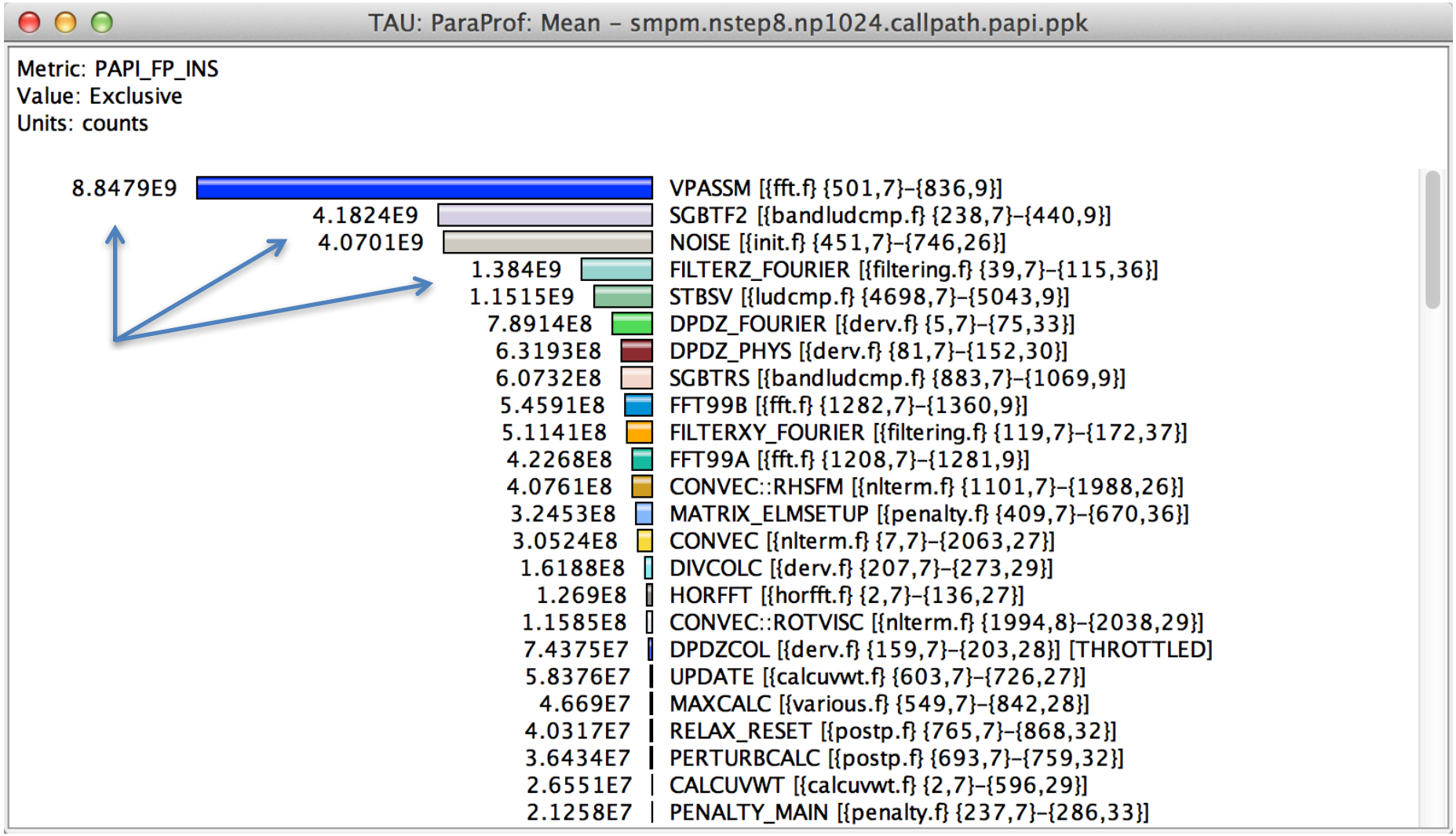
PERFORMANCE DATA ANALYSIS

How Much Time per Code Region?



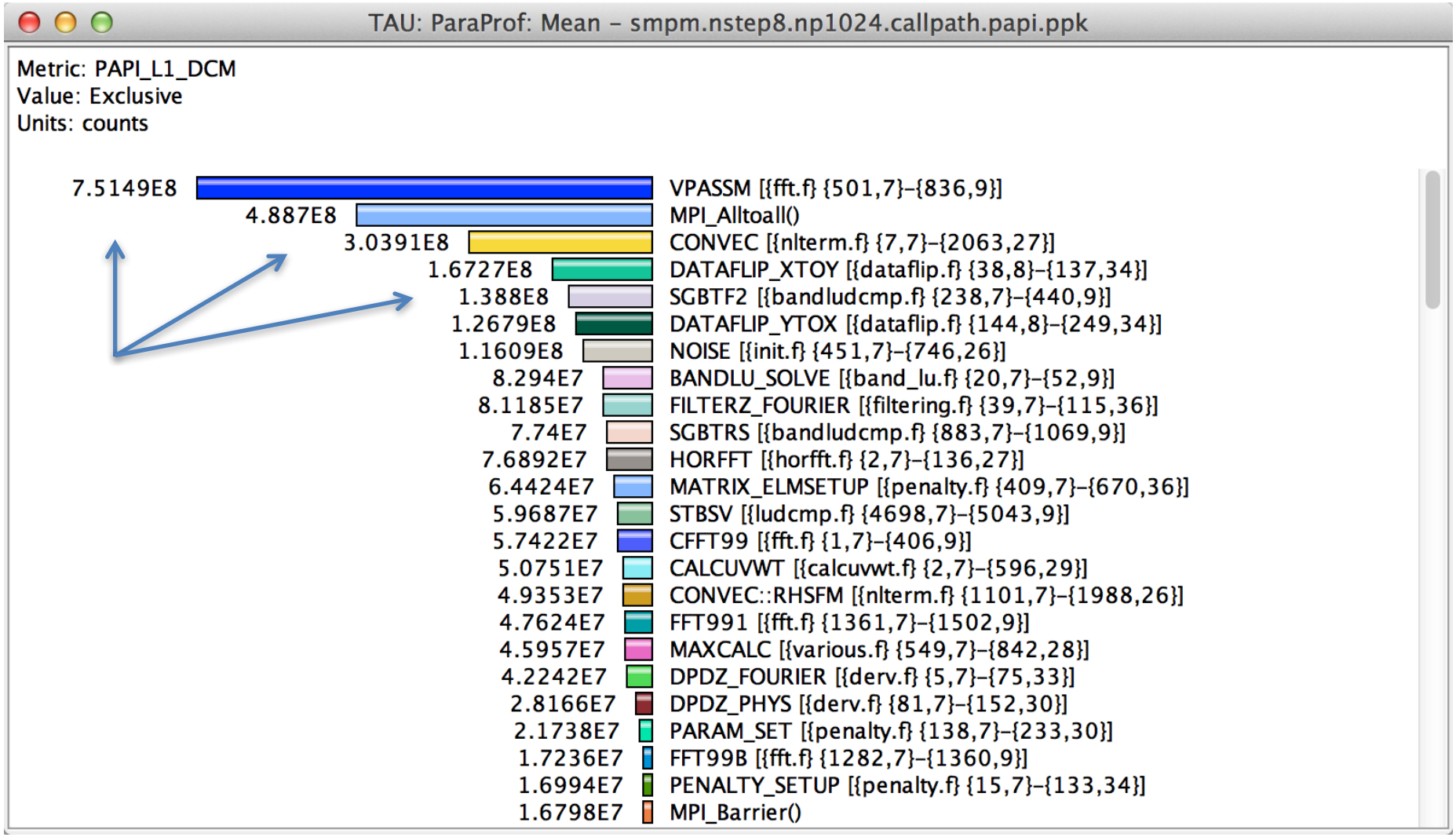
`% paraprof` (Click on label, e.g. “Mean” or “node 0”)

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)

Where is Memory Allocated / Deallocated?

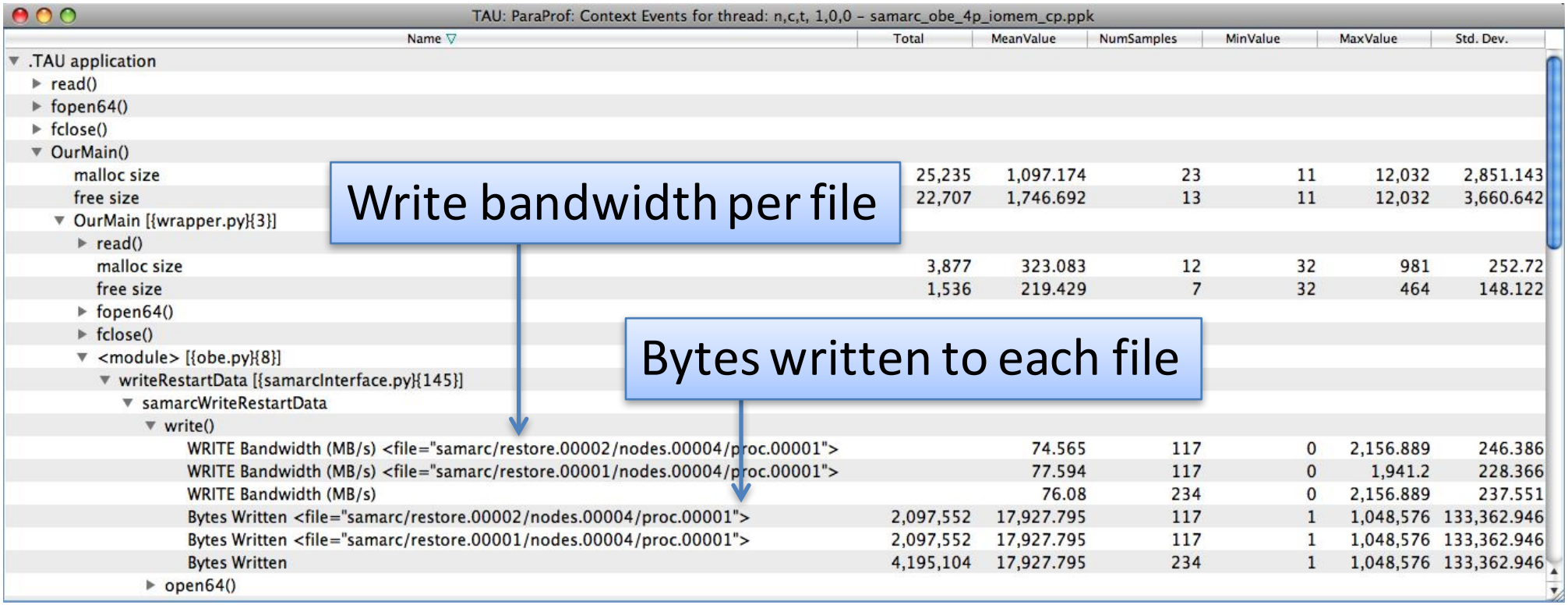
TAU: ParaProf: Mean Context Events – sphere_np32_nsteps5_mem.ppk

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)

What are the I/O Characteristics?



Name	Total	MeanValue	NumSamples	MinValue	MaxValue	Std. Dev.
.TAU application						
read()						
fopen64()						
fclose()						
OurMain()						
malloc size	25,235	1,097.174	23	11	12,032	2,851.143
free size	22,707	1,746.692	13	11	12,032	3,660.642
OurMain [{wrapper.py}{3}]						
read()						
malloc size	3,877	323.083	12	32	981	252.72
free size	1,536	219.429	7	32	464	148.122
fopen64()						
fclose()						
<module> [{obe.py}{8}]						
writeRestartData [{samarcInterface.py}{145}]						
samarcWriteRestartData						
write()						
WRITE Bandwidth (MB/s) <file="samarc/restore.00002/nodes.00004/proc.00001">		74.565	117	0	2,156.889	246.386
WRITE Bandwidth (MB/s) <file="samarc/restore.00001/nodes.00004/proc.00001">		77.594	117	0	1,941.2	228.366
WRITE Bandwidth (MB/s)		76.08	234	0	2,156.889	237.551
Bytes Written <file="samarc/restore.00002/nodes.00004/proc.00001">	2,097,552	17,927.795	117	1	1,048,576	133,362.946
Bytes Written <file="samarc/restore.00001/nodes.00004/proc.00001">	2,097,552	17,927.795	117	1	1,048,576	133,362.946
Bytes Written	4,195,104	17,927.795	234	1	1,048,576	133,362.946
open64()						

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

What are the I/O Characteristics?

Name △	Total	NumSamples	MaxValue	MinValue	MeanValue	Std. Dev.
▶ Incl						
▶ Initialize						
▶ LoadBodyEuler						
▶ LoadMesh						
MPI-IO Bytes Written	4,328,712	144	893,152	0	30,060.5	128,042.696
MPI-IO Write Bandwidth (MB/s)		144	196.86	0	3.421	16.87
▶ MPI_Allgatherv()						
▶ MPI_Bcast()						
▶ MPI_Comm_create()						
▶ MPI_File_close()						
▶ MPI_File_open()						
▶ MPI_File_write_all()						
▶ MPI_File_write_at()						
▶ MPI_Finalize()						
▶ MPI_Gather()						
▶ MPI_Gatherv()						

Peak MPI-IO Write Bandwidth

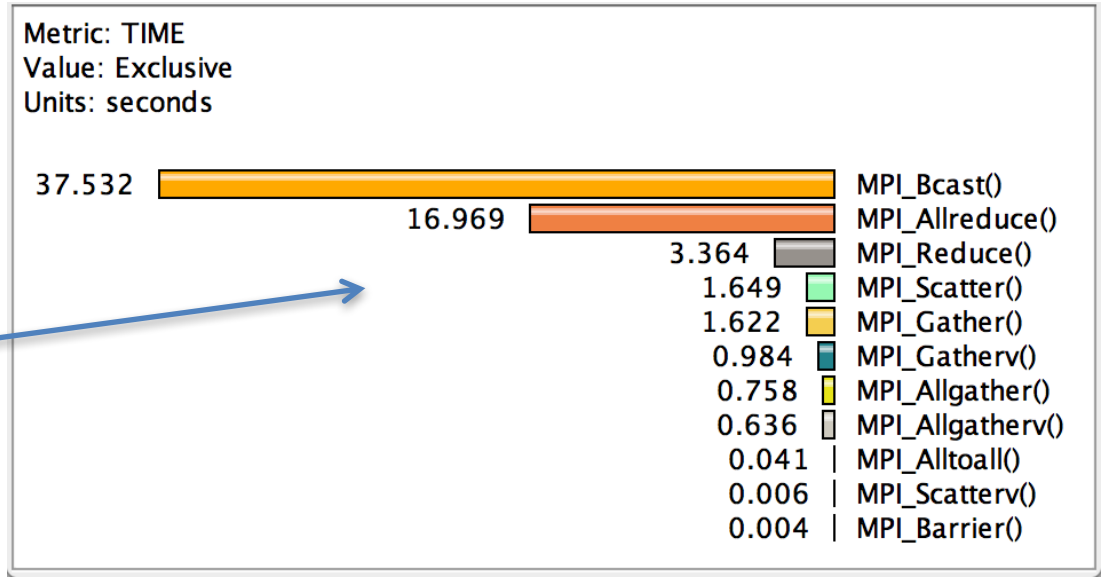
% `paraprof` (Right-click label [e.g "node 0"] → Show Context Event Window)

How Much Time is spent in Collectives?

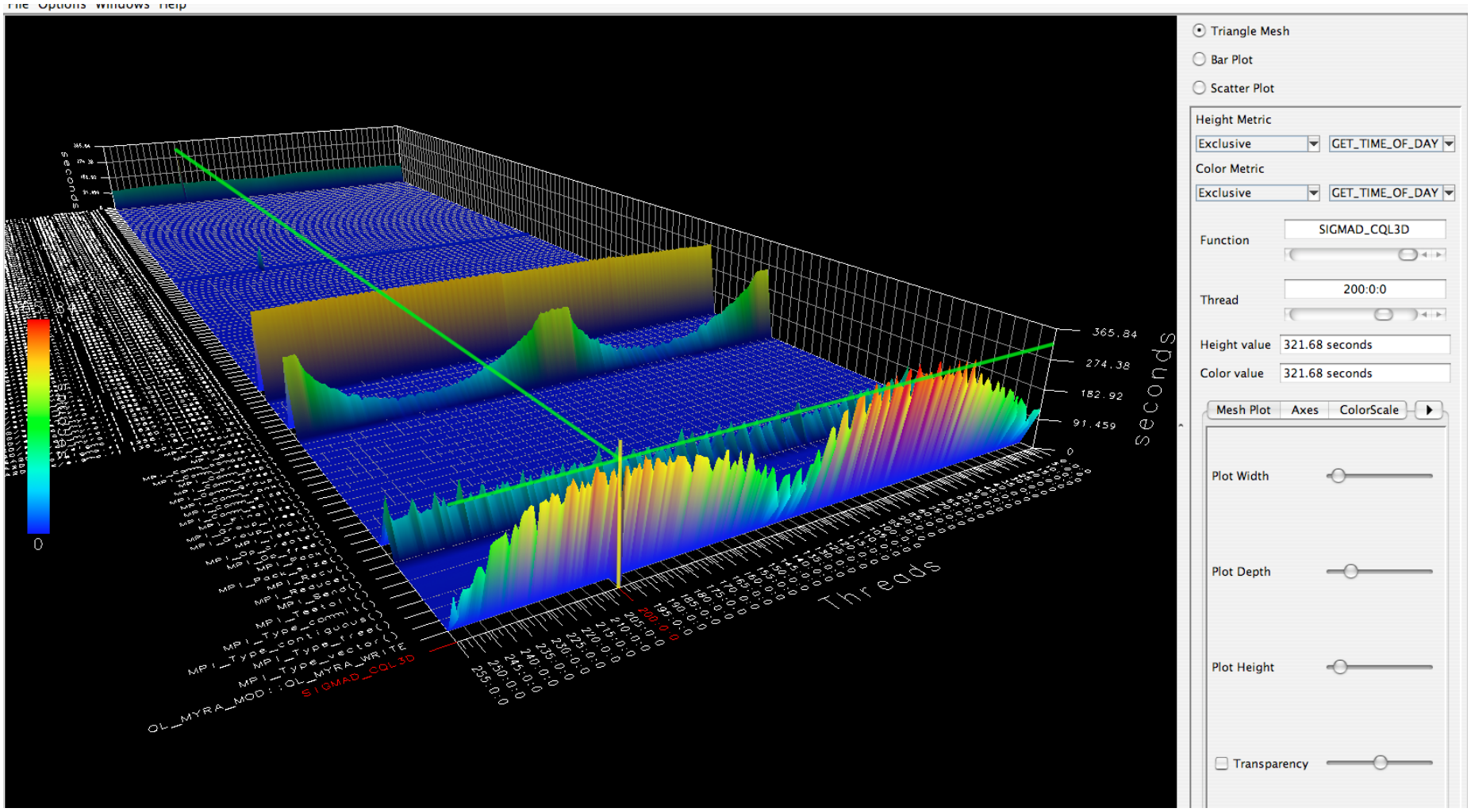
Name △	Total	Num...	MaxValue	MinValue	MeanValue	Std. Dev.
▶ MPI_Wait()						
▶ MPI_Waitall()						
Message size for all-gather	305,753,268	72	172,215,296	4	4,246,573.167	22,551,605.859
Message size for all-reduce	163,308	632	21,908	4	258.399	897.725
Message size for all-to-all	112	14	8	8	8	0
Message 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

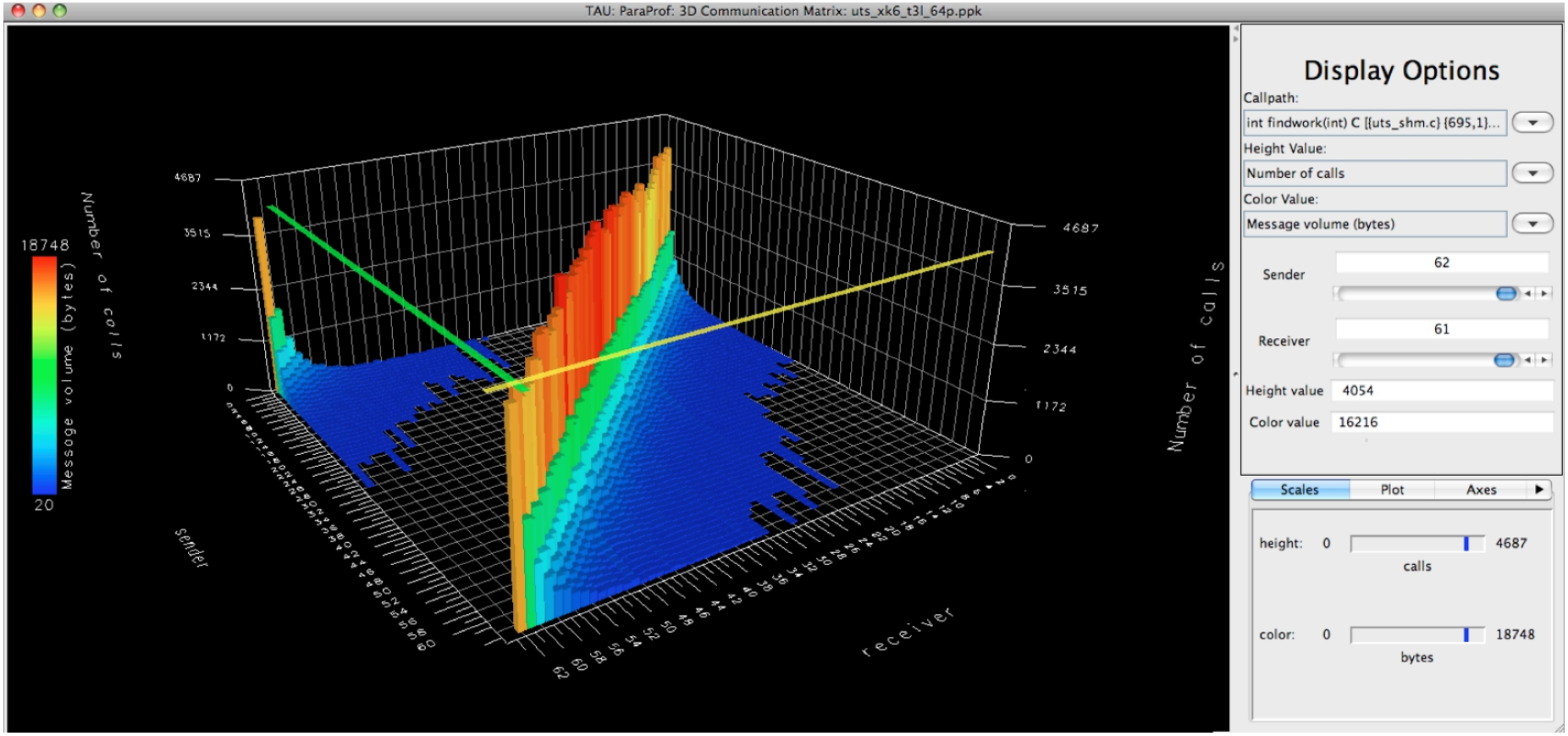


3D Profile Visualization



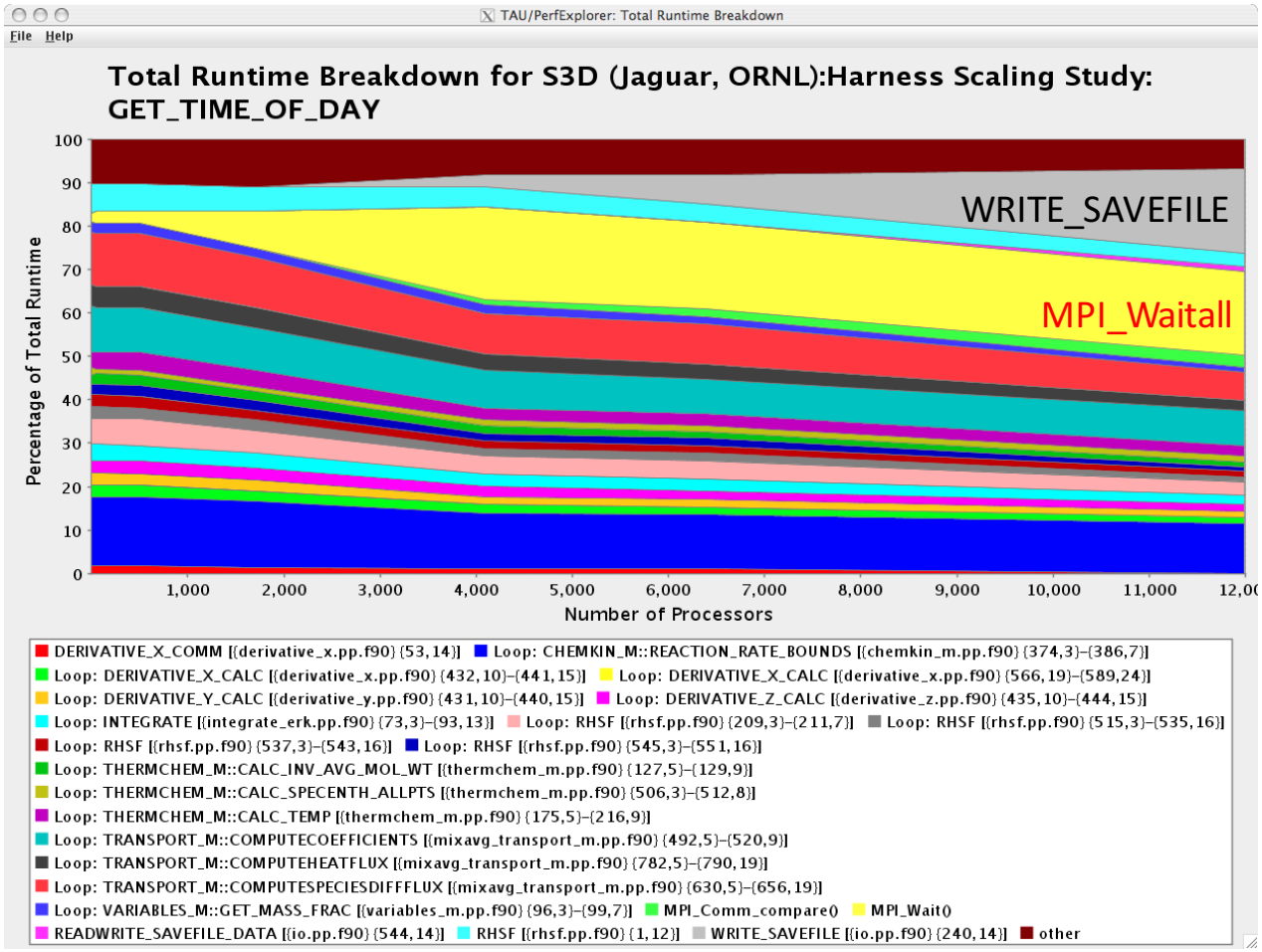
`% paraprof` (Windows → 3D Visualization)

3D Communication Visualization



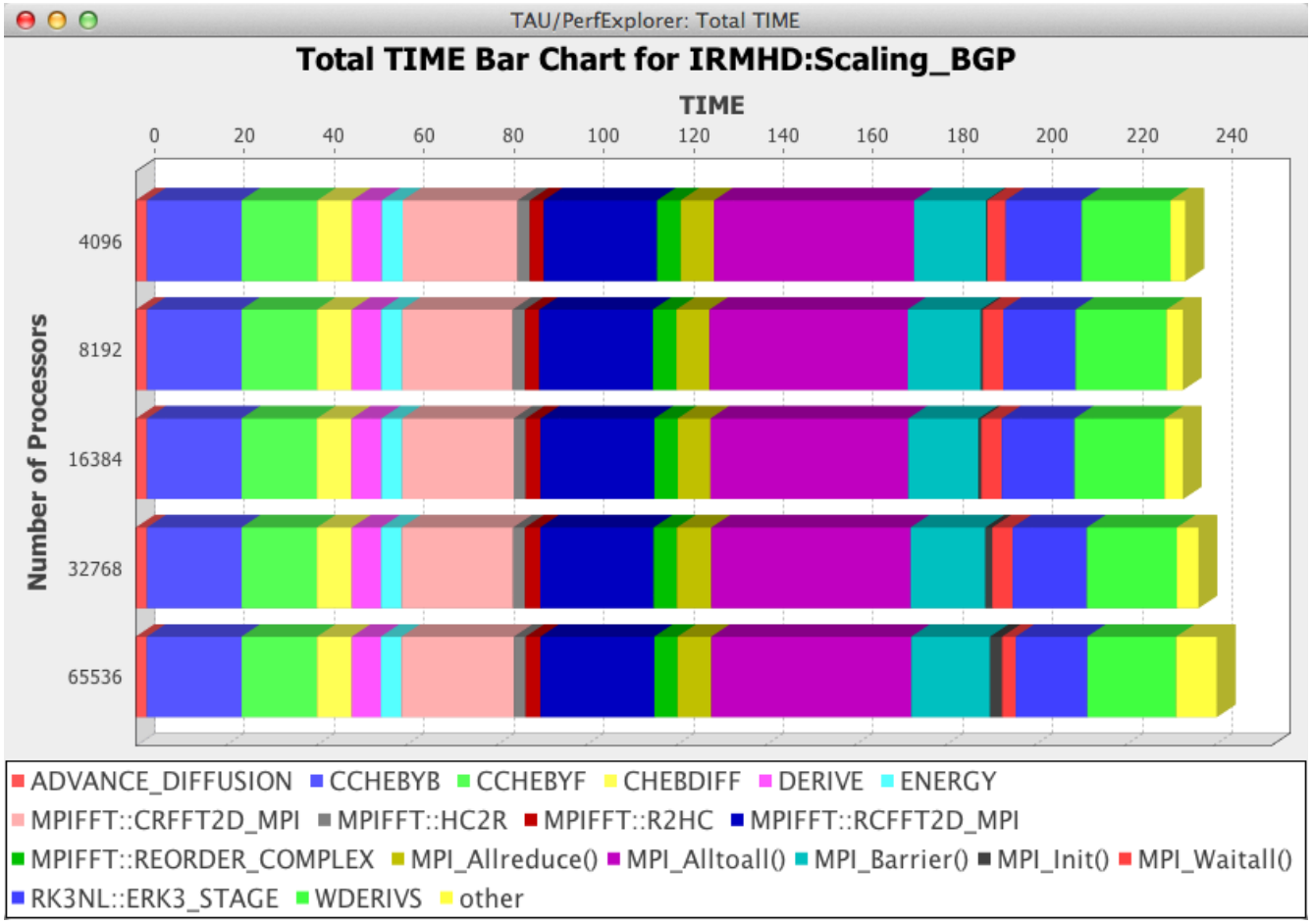

```
% qsub –env TAU_COMM_MATRIX=1 …
% paraprof (Windows → 3D Communication Matrix)
```

How Does Each Routine Scale?



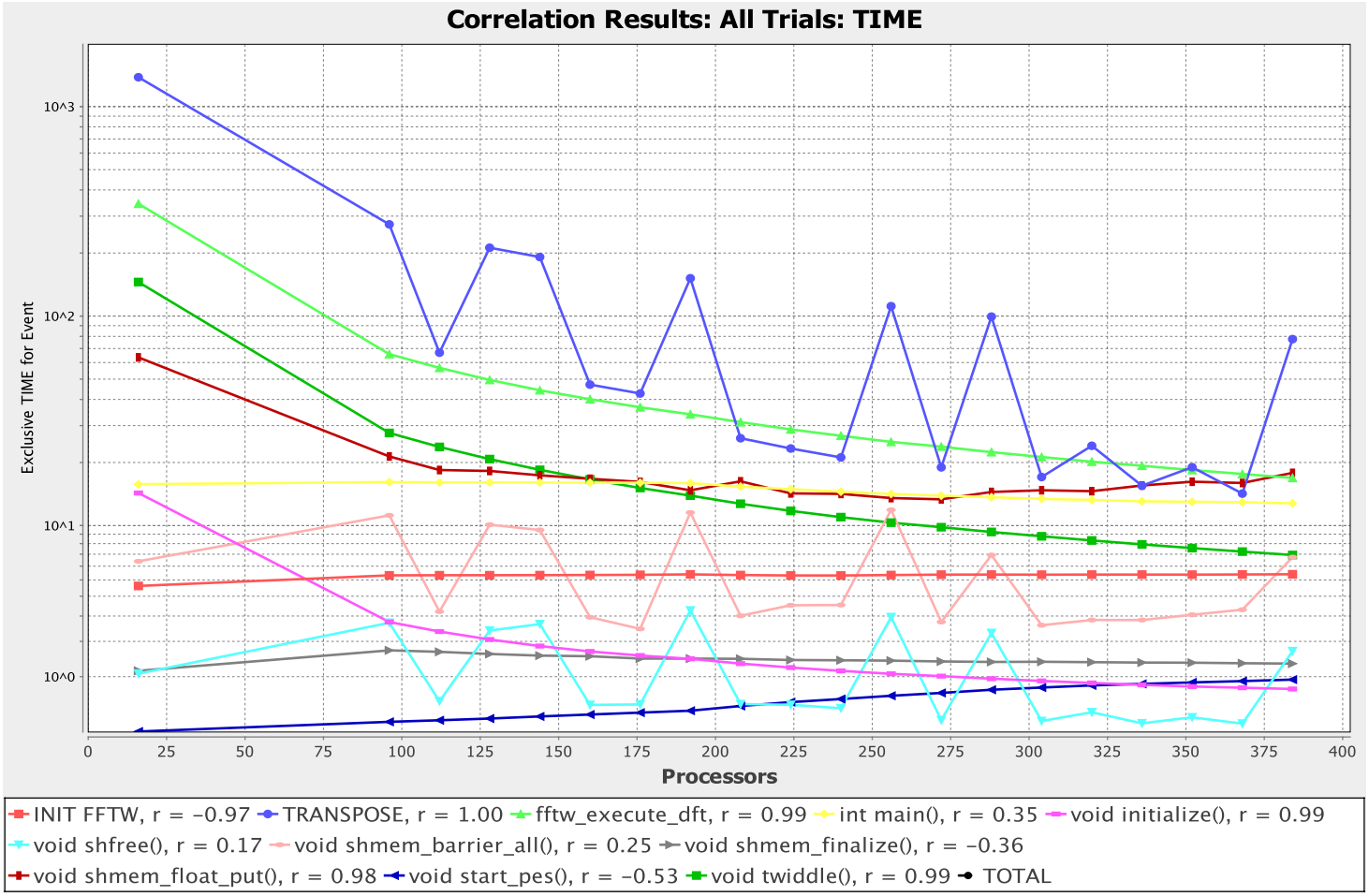
`% perfexplorer` **(Charts → Runtime Breakdown)**

How Does Each Routine Scale?



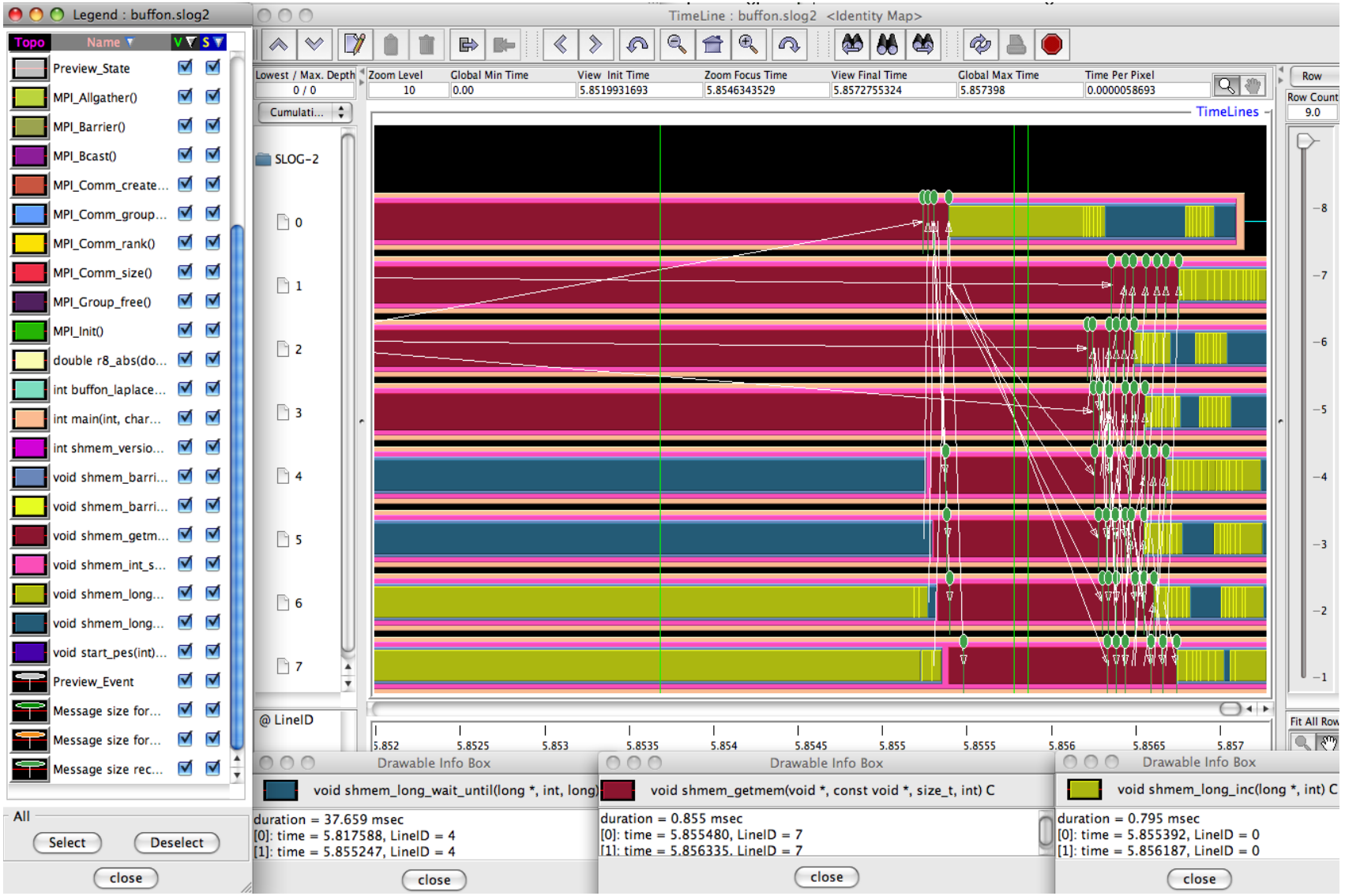
`% perfexplorer` **(Charts → Stacked Bar Chart)**

Which Events Correlate with Runtime?



% `perfexplorer` (Charts → Correlate Events with Total Runtime)

When do Events Occur?



When do Events Occur?

To generate a trace and visualize it in Jumpshot:

```
% qsub –env TAU_TRACE=1 …
% tau_treemerge.pl
% tau2slog2 tau.trc tau.edf –o app.slog2
% jumpshot app.slog2
```

Intuitive Performance Engineering

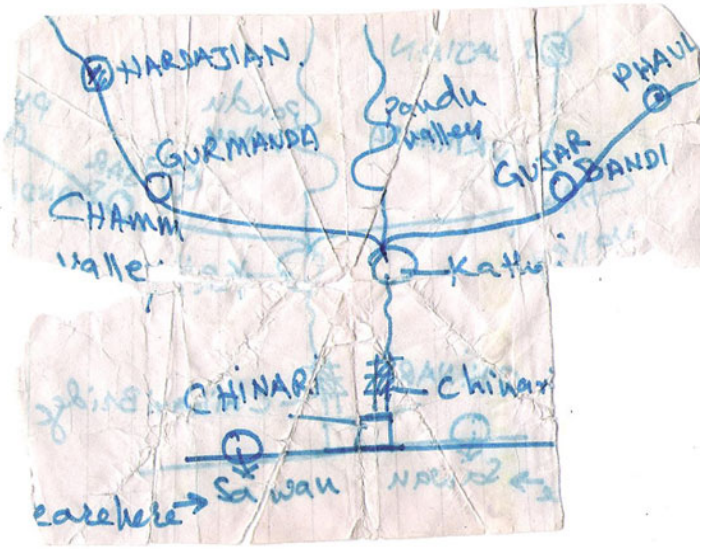
TAU COMMANDER (COMING THIS AUGUST)

How do we Improve Productivity?

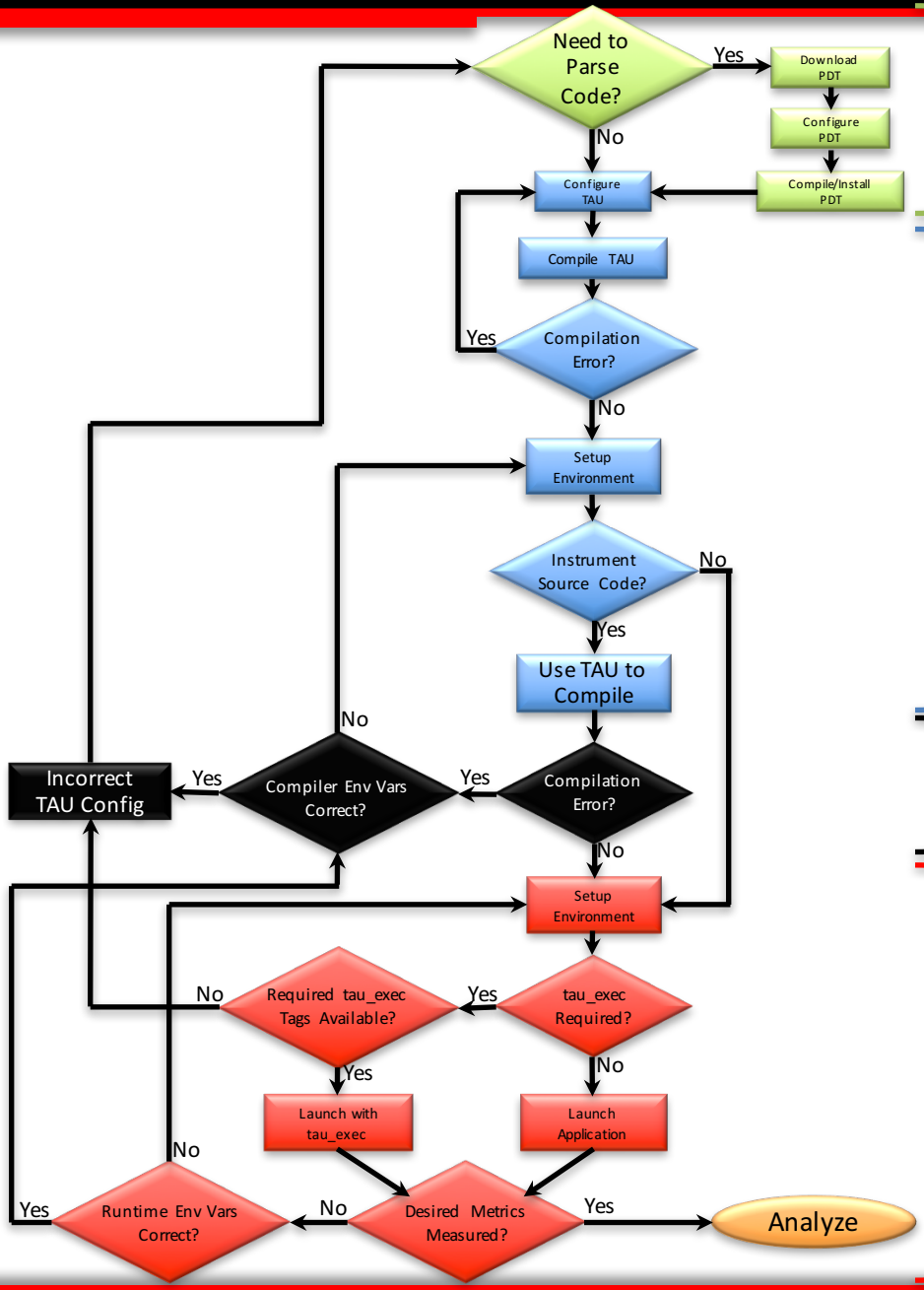
The TAU Commander Approach

- Say where you're going, not how to get there
- **TAU Projects** give context to the user's actions
  - Defines desired metrics and measurement approach
  - Defines operating environment
  - Establishes a baseline for error checking

vs.



TAU Workflow



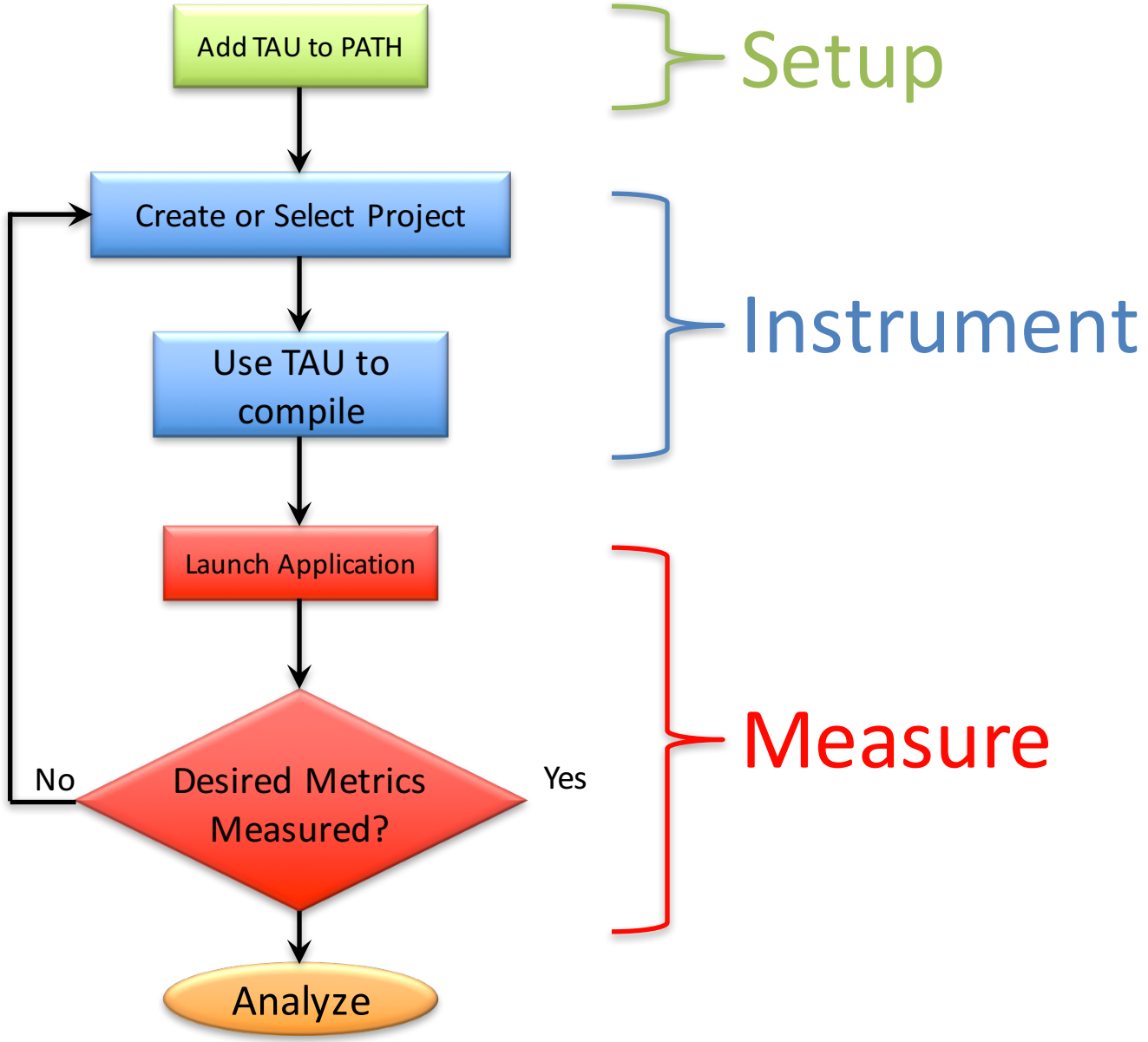
Setup

Instrument

Troubleshoot

Measure

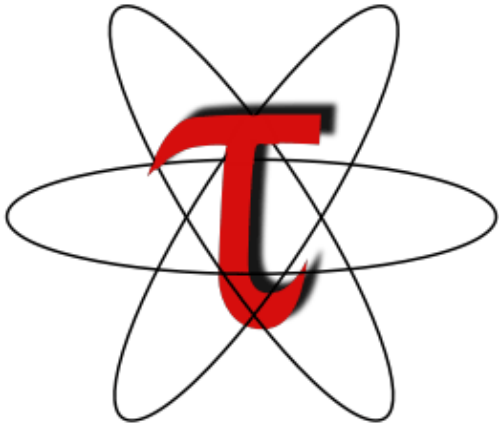
TAU Commander Workflow



Intuitive Performance Engineering

CONCLUSION

Downloads



http://tau.uoregon.edu

http://github.com/ParaToolsInc/taucmdr

http://www.hpclinux.com

Free download, open source, BSD license

Acknowledgements

- Department of Energy
  - Office of Science
  - Argonne National Laboratory
  - Oak Ridge National Laboratory
  - NNSA/ASC Trilabs (SNL, LLNL, LANL)
- HPCMP DoD PETTT Program
- National Science Foundation
  - Glassbox, SI-2
- University of Tennessee
- University of New Hampshire
  - Jean Perez, Benjamin Chandran
- University of Oregon
  - Allen D. Malony, Sameer Shende
  - Kevin Huck, Wyatt Spear
- TU Dresden
  - Holger Brunst, Andreas Knupfer
  - Wolfgang Nagel
- Research Centre Jülich
  - Bernd Mohr
  - Felix Wolf

















Intuitive Performance Engineering

CASE STUDIES

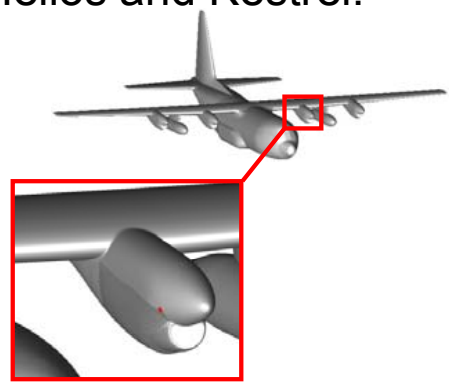
Strand Technology

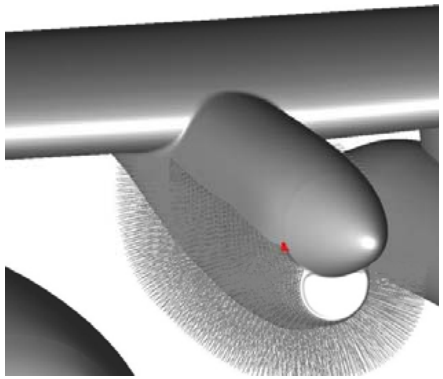
Technology Drivers

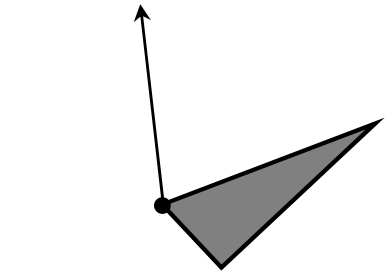
- Timeliness (automation of mesh generation)
- Timeliness (automation and scalability of domain connectivity)
- Timeliness/Physical accuracy (computational efficiency and scalability of aerodynamic solvers)
- Processor architecture (small memory footprint maps well to hierarchical memory architectures, e.g., multi-core, GPU)

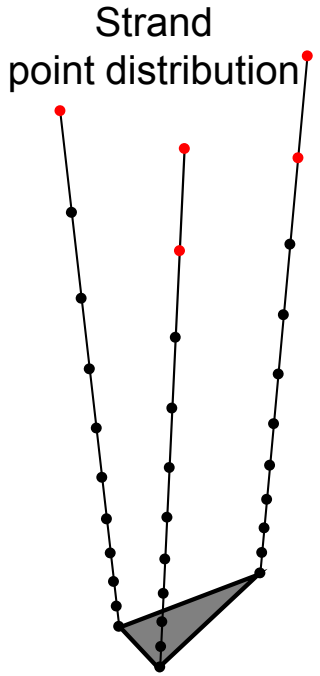
CREATE-AV Example

This is a new meshing paradigm introduced in 2007 by current members of the CREATE-AV technical staff. The technology is being matured in the Helios product and will be deployed through both Helios and Kestrel.





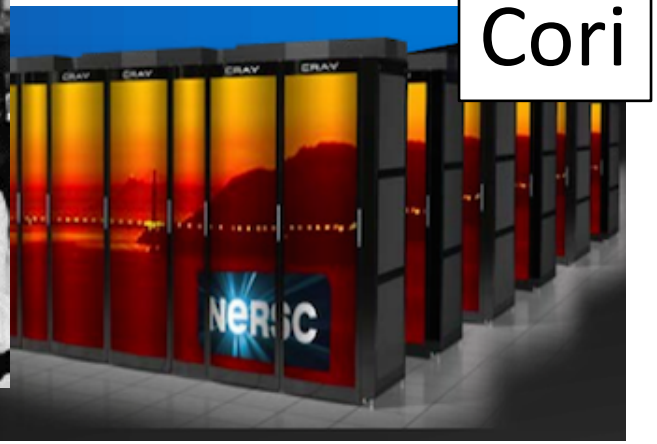




Target Platforms

Armstrong [XC30]

Cori

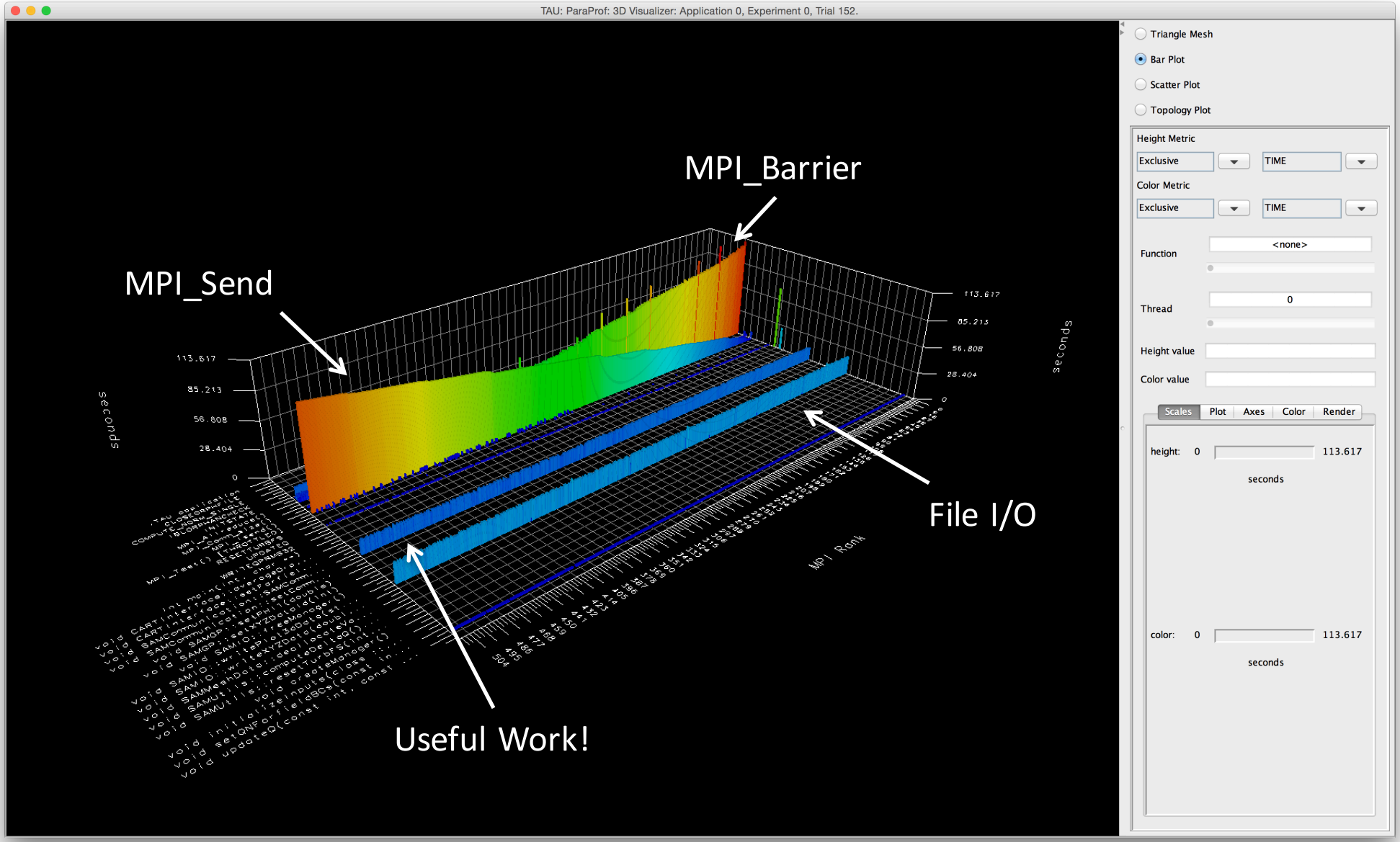


Haise [iDataPlex]

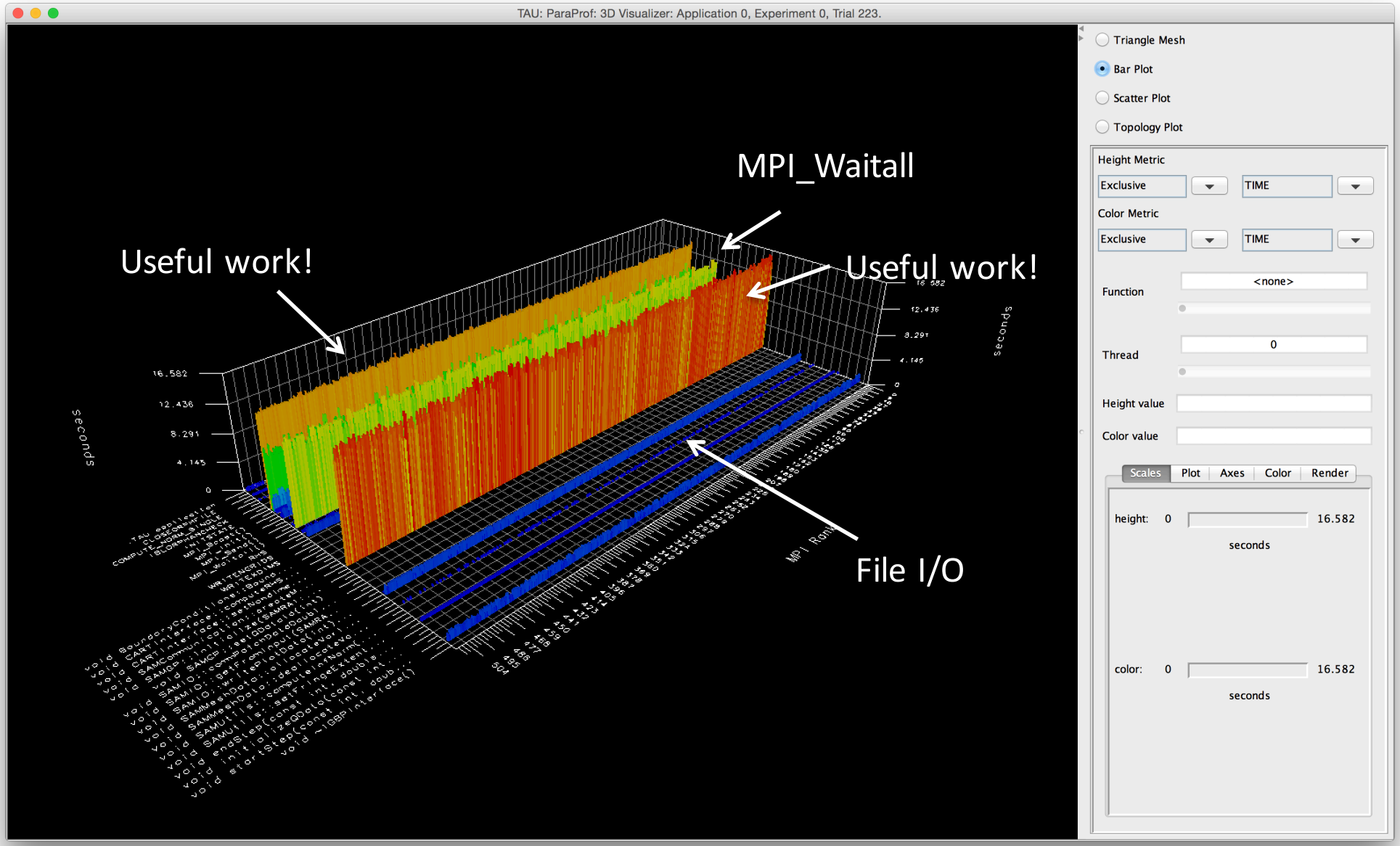
Lightning [XC30]

Kilrain [iDataPlex]

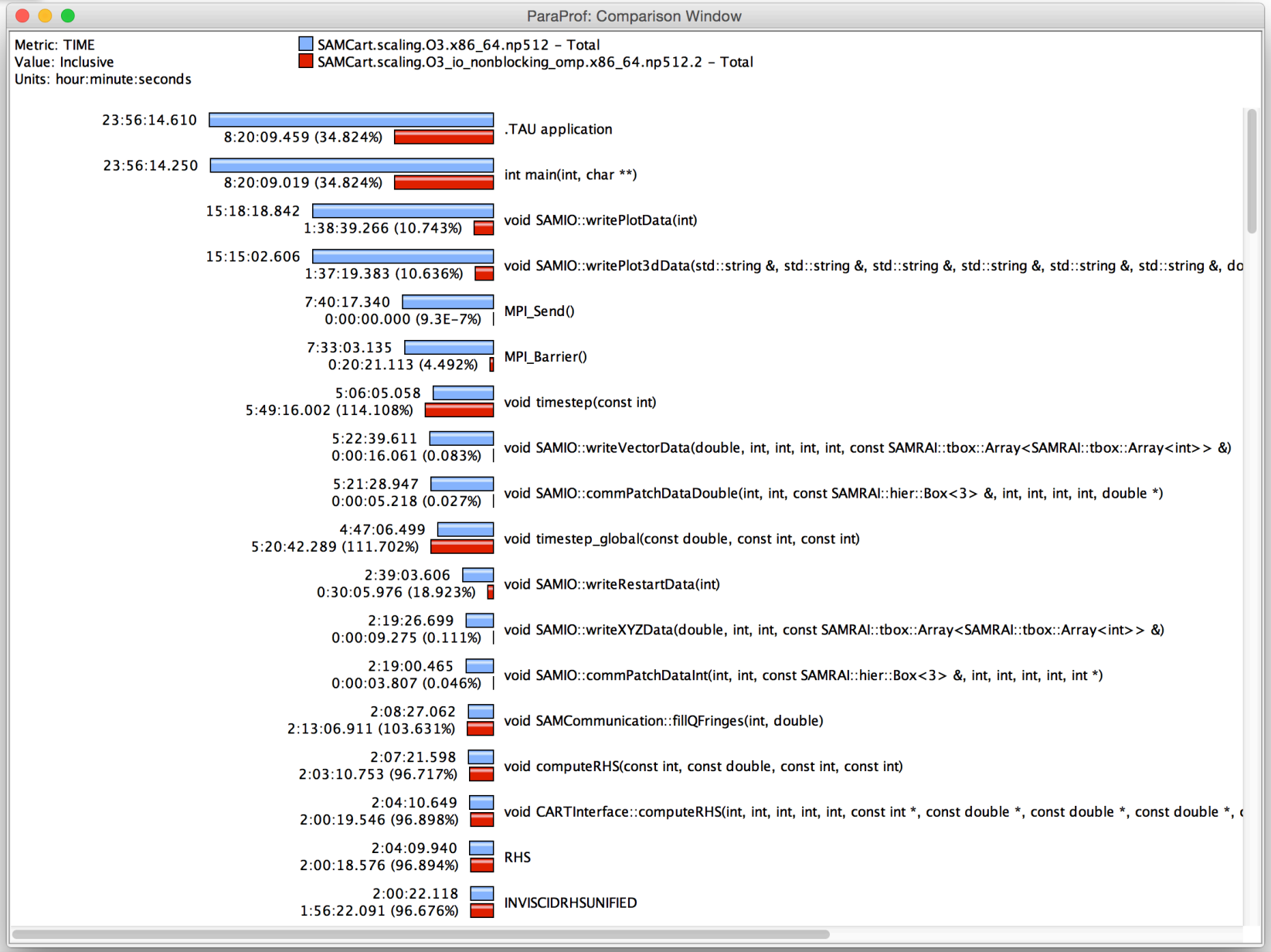
Initial Profile on Babbage



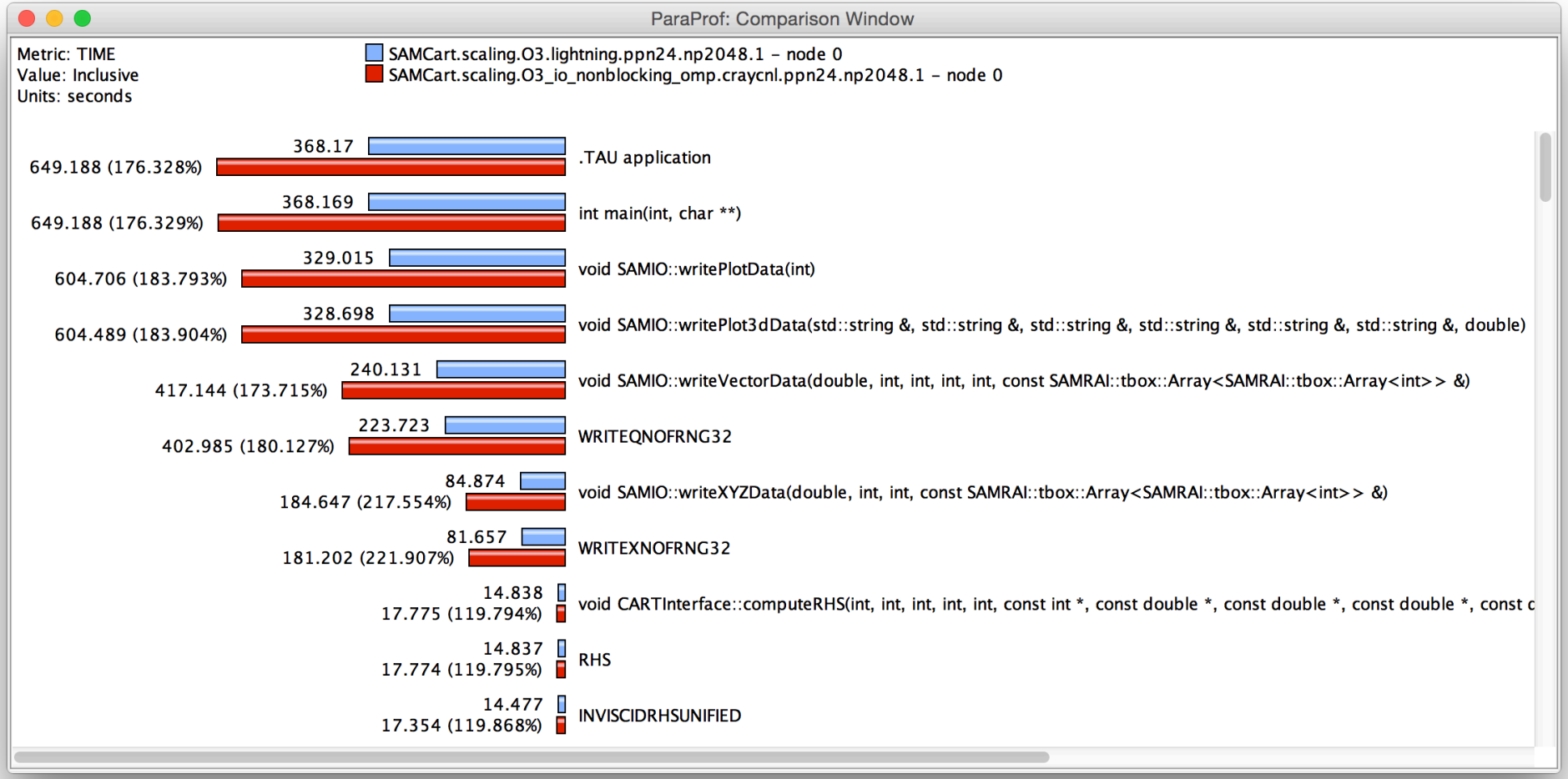
Hot Spot Optimization



65% Runtime Reduction (~2x faster)

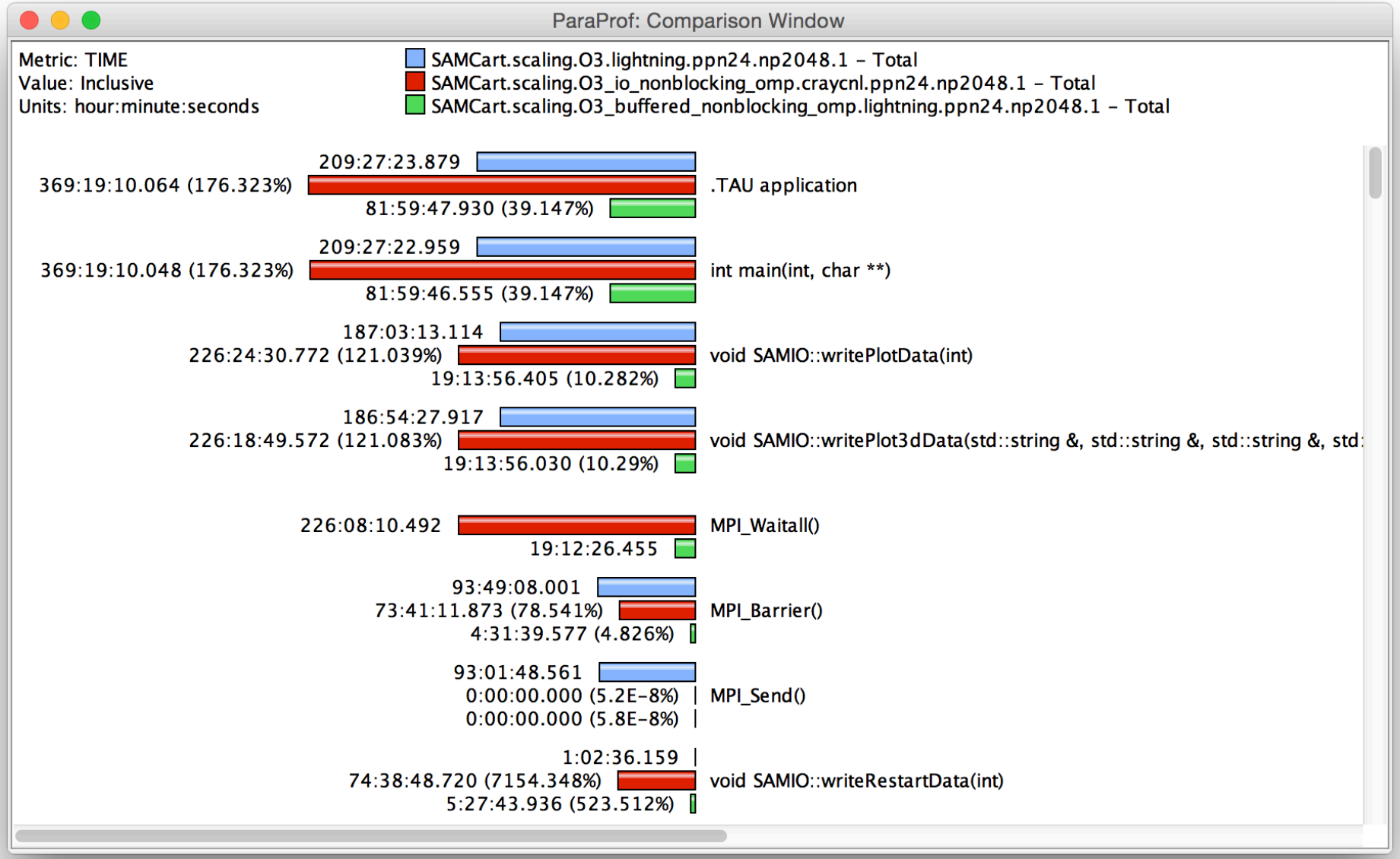


Cray XC30



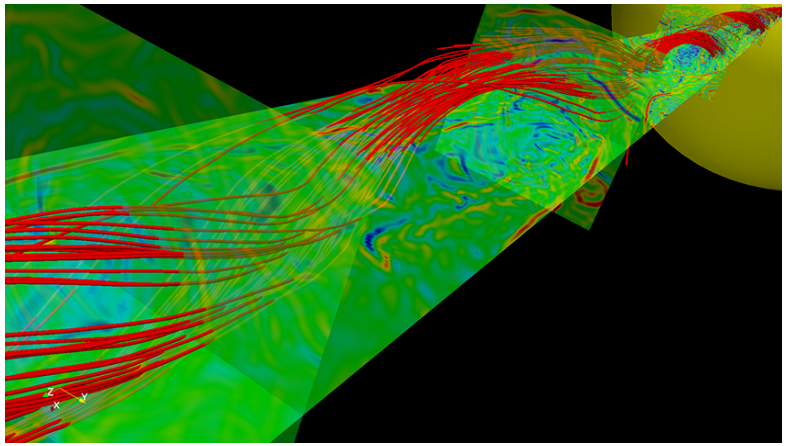
Slower! What happened???

No worries, I fix it

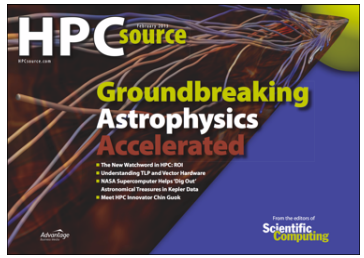


IRMHD on Intrepid and Mira

- INCITE magnetohydrodynamcis simulation to understand solar winds and coronal heating
  - First direct numerical simulations of Alfvén wave (AW) turbulence in extended solar atmosphere accounting for inhomogeneities
  - Team
    - University of New Hampshire (Jean Perez and Benjamin Chandran)
    - ALCF (Tim Williams)
    - University of Oregon (Sameer Shende)

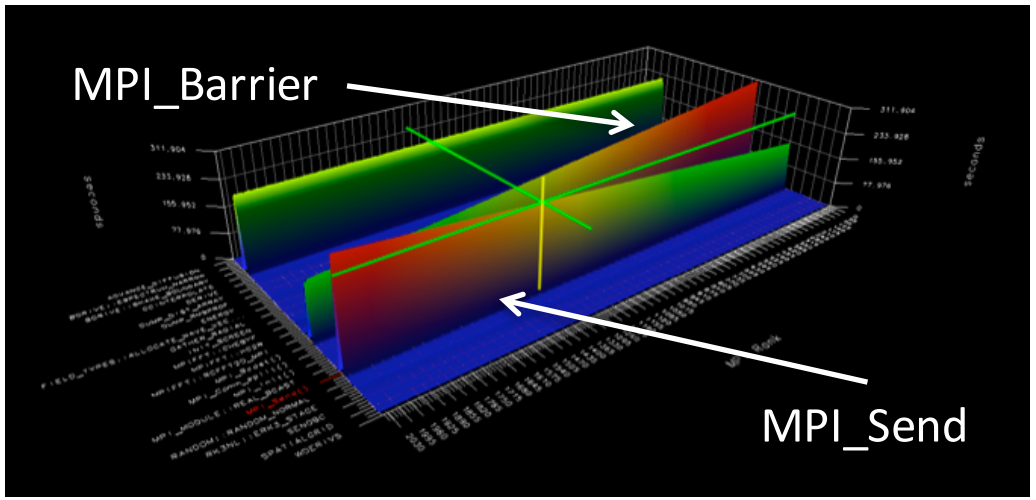


- IRMHD (Inhomogeneous Reduced Magnetohydrodynamics)
  - Fortran 90 and MPI
  - Excellent weak and strong scaling properties
  - Tested and benchmarked on Intrepid and Mira
- HPC Source article and ALCF news
  https://www.alcf.anl.gov/articles/furthering-understanding-coronal-heating-and-solar-wind-origin



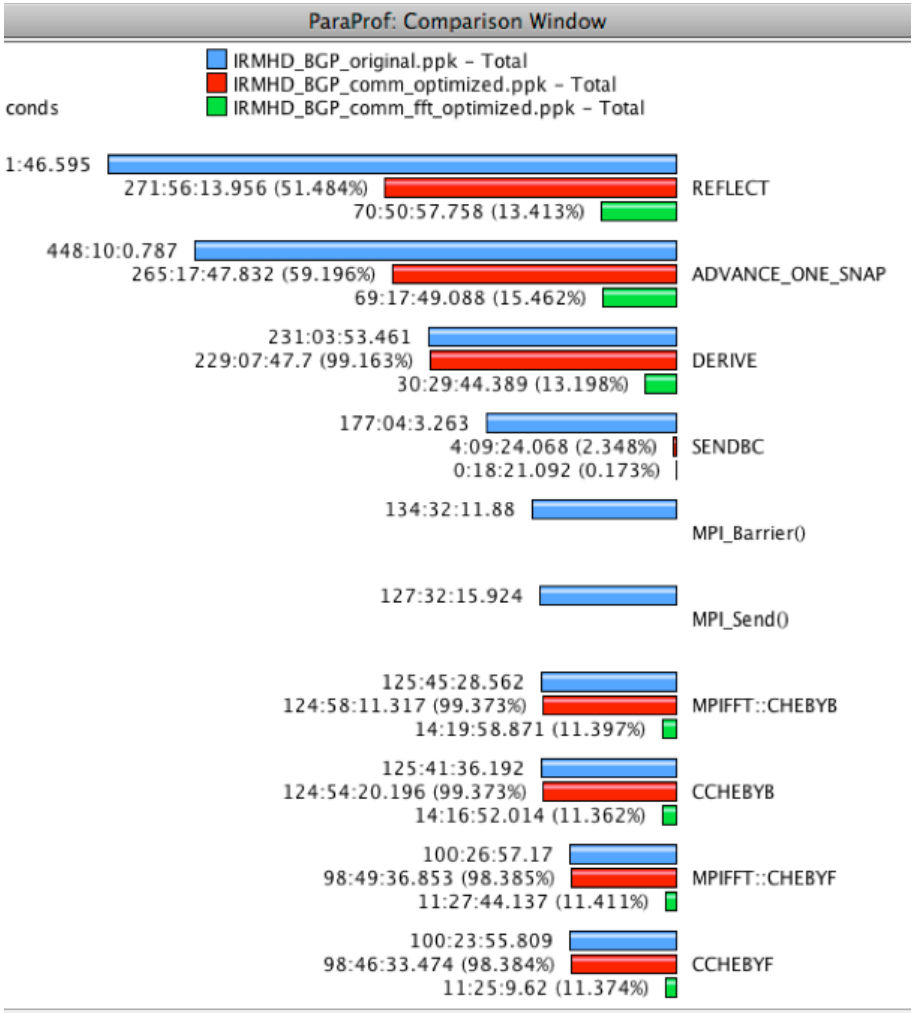
IRMHD Communication Analysis

- Source-based (direct) instrumentation
- MPI instrumentation and volume measurement
- IRMHD exhibited significant synchronous communication bottlenecks
- On 2,408 cores of BG/P:
  - **MPI_Send** and **MPI_Bcast** take significant time
  - Opportunities for communication/ computation overlap
  - Identified possible targets for computation improvements



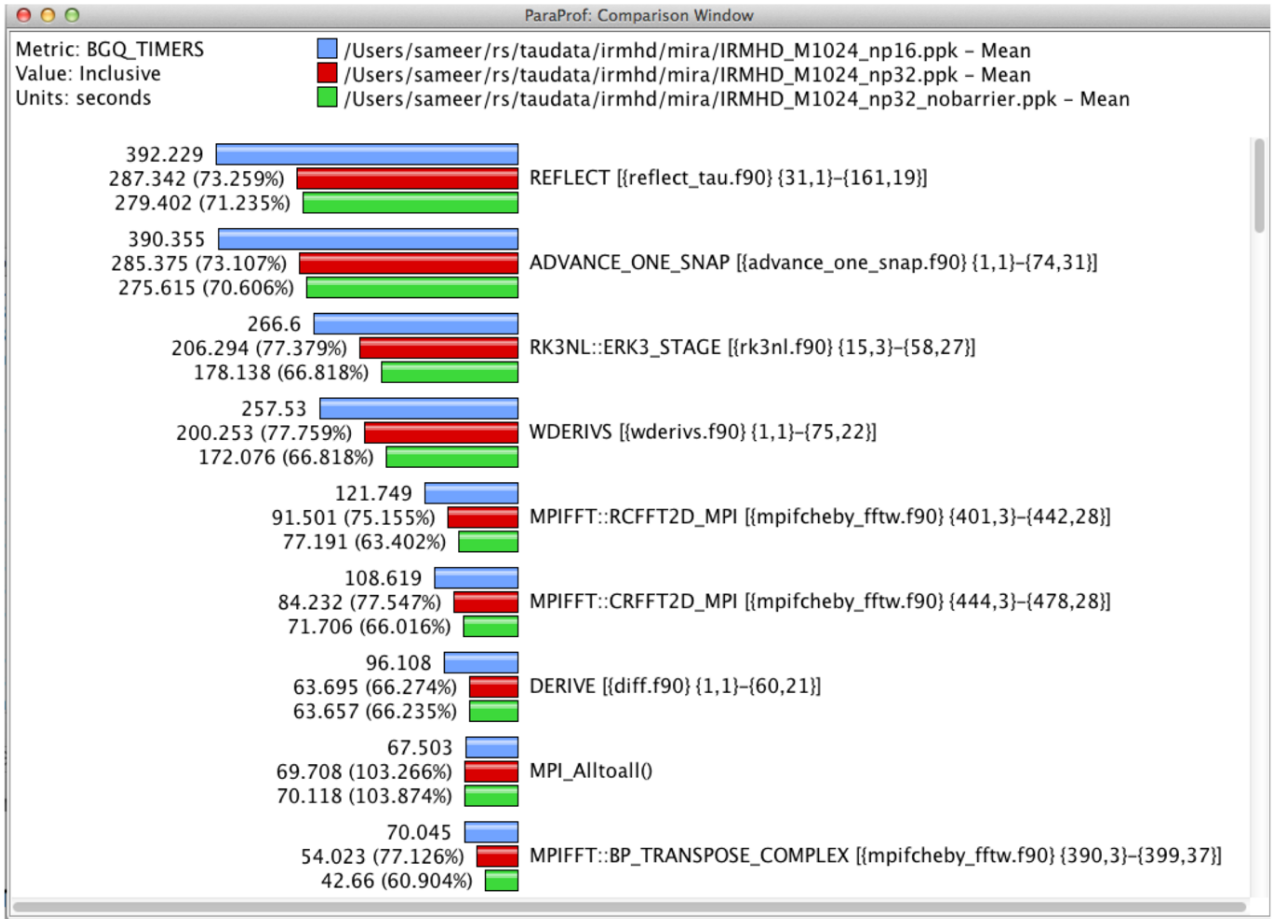
IRMHD Optimization on Intrepid (BG/P)

- On 2,408 cores, overall execution time reduced from 528.18 core hours to 70.8 core hours **(>7x improvement)**
- Non-blocking communication substrate
- More efficient implementation of underlying FFT



IRMHD Optimization on MIRA (BG/Q)

- Oversubscribe nodes: 32k ranks vs. 16k per node
- Overall time improvement: 71.23% of original



Intuitive Performance Engineering

REFERENCE

Online References

- PAPI:
  - PAPI documentation is available from the PAPI website:
    http://icl.cs.utk.edu/papi/
- TAU:
  - TAU Users Guide and papers available from the TAU website:
    http://tau.uoregon.edu/
- VAMPIR:
  - VAMPIR website:
    http://www.vampir.eu/
- Scalasca:
  - Scalasca documentation page:
    http://www.scalasca.org/
- Eclipse PTP:
  - Documentation available from the Eclipse PTP website:
    http://www.eclipse.org/ptp/

Compiling Fortran Codes with TAU

- **If your Fortran code uses free format in .f files (fixed is default for .f):**
  % export TAU_OPTIONS='-optPdtF95Opts="-R free" -optVerbose'

- **To use the compiler based instrumentation instead of PDT (source-based):**
  % export TAU_OPTIONS='-optCompInst -optVerbose'

- **If your Fortran code uses C preprocessor directives (#include, #ifdef, #endif):**
  % export TAU_OPTIONS='-optPreProcess –optVerbose'

- **To use an instrumentation specification file:**
  % export TAU_OPTIONS=
  '-optTauSelectFile=select.tau -optVerbose -optPreProcess’

Example select.tau file

```
BEGIN_INSTRUMENT_SECTION
loops file="*" routine="#"
memory file="foo.f90" routine=”#"
io file="abc.f90" routine="FOO"
END_INSTRUMENT_SECTION
```

Generate a PAPI profile with 2 or more counters

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-bgqtimers-papi-mpi-pdt
% export TAU_OPTIONS='-optTauSelectFile=select.tau -optVerbose'
% cat select.tau
  BEGIN_INSTRUMENT_SECTION
  loops routine="#"
  END_INSTRUMENT_SECTION

% export PATH=$TAU_ROOT/bin:$PATH
% make F90=tau_f90.sh
(Or edit Makefile and change F90=tau_f90.sh)
%
% qsub --env TAU_METRICS=TIME:PAPI_FP_INS:PAPI_L1_DCM -n 4 -t 15 ./a.out
% paraprof --pack app.ppk
   Move the app.ppk file to your desktop.
% paraprof app.ppk
  Choose Options -> Show Derived Metrics Panel -> "PAPI_FP_INS", click "/", "TIME", click
    "Apply"  and choose the derived metric.
```

Tracking I/O in static binaries

```
% export TAU_MAKEFILE=$TAU/Makefile.tau-bgqtimers-papi-mpi-pdt
% export PATH=$TAU_ROOT/bin:$PATH
% export TAU_OPTIONS='-optTrackIO -optVerbose'
% make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh
% mpirun -n 4 ./a.out
% paraprof -pack ioprofile.ppk
% export TAU_TRACK_IO_PARAMS 1
% mpirun -n 4 ./a.out (to track parameters used in POSIX I/O calls as
   context events)
```

Installing and Configuring TAU

- Installing PDT:
  - wget http://tau.uoregon.edu/pdt.tgz
  - ./configure –prefix=<dir>; make ; make install
- Installing TAU:
  - wget http://tau.uoregon.edu/tau.tgz
  - ./configure -bfd=download -pdt=<dir> -papi=<dir> …
  - make install
- Using TAU:
  - export TAU_MAKEFILE=<taudir>/<arch>/lib/Makefile.tau-<TAGS>
  - make CC=tau_cc.sh CXX=tau_cxx.sh F90=tau_f90.sh

Compile-Time Options (TAU_OPTIONS)

% 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
-optMemDbg	Runtime bounds checking (see TAU_MEMDBG_* env vars)
-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)
-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 –optMemDbg or tau_exec)
TAU_MEMDBG_PROTECT_ABOVE	0	Setting to 1 turns on bounds checking for dynamically allocated arrays. (Use with –optMemDbg or tau_exec –memory_debug).
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>)