Hands-on tutorial with TAU

Srinath Vadlamani
ParaTools, Inc.

IXPUG Annual Meeting 2015, Sept. 28 , CRT, LBL, CA

Overview

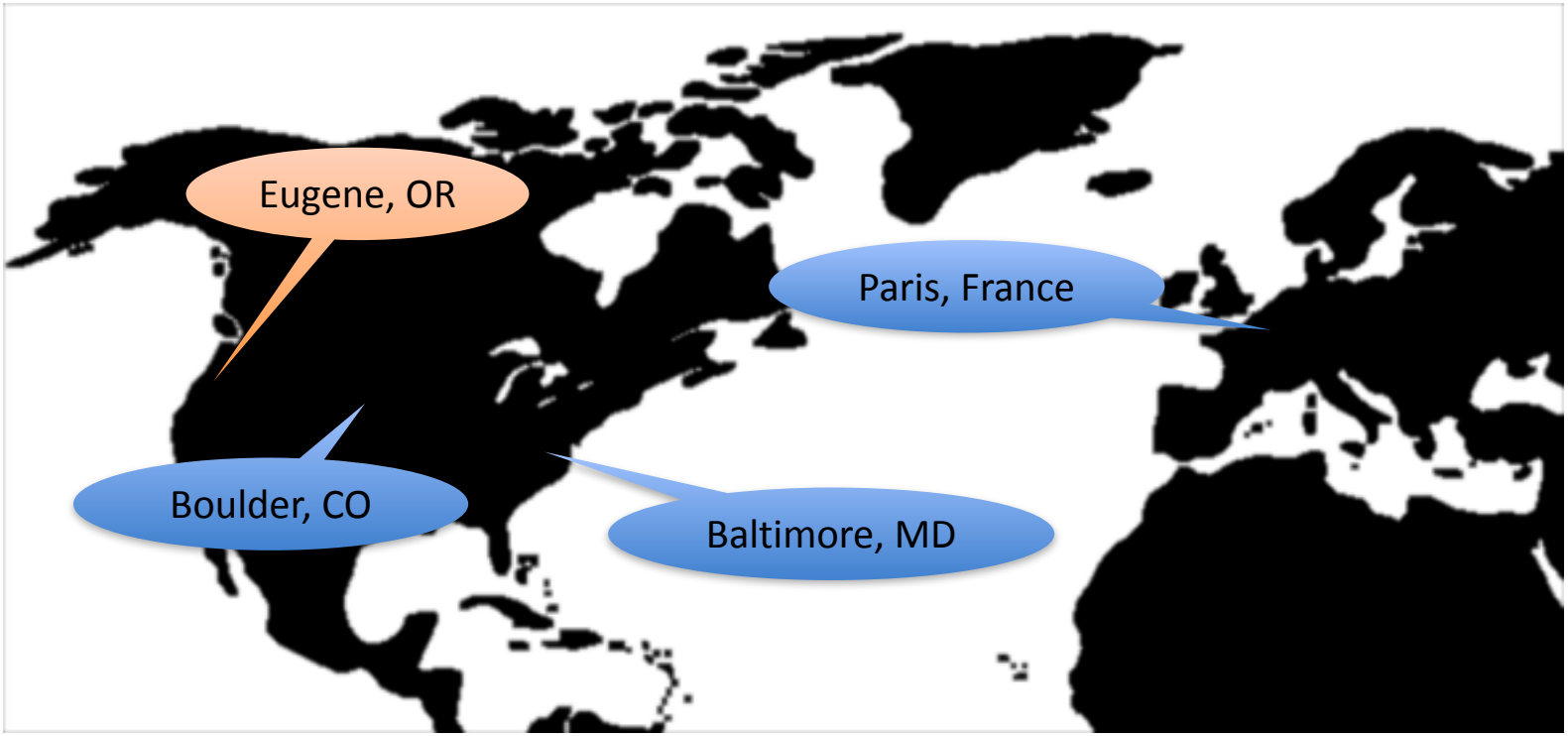
- Please gather in groups of 4 or 5 for the hands on portion of tutorial.
- Background
  - ParaTools
  - TAU
- Hands-On Examples
- Start working on your own code.

ParaTools

Allen Malony (CEO)

Sameer Shende (President)

- John Linford
- Srinath Vadlamani
- Kevin Huck
- Wyatt Spear
- Jean-Baptiste Besnard



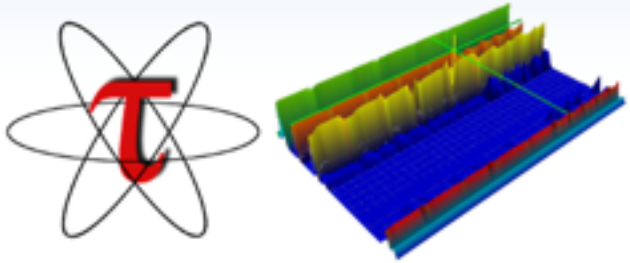
ParaTools Software

Tools

- TAU
- Kppa
- SysFera-DS
- RotCFD
- ThreadSpotter
- Vampir
- HPC Linux
- PToolsRTE
- PToolsWin
- Eclipse
- IQB

Training and Services

ParaTools provides consulting expertise in parallel and distributed computing, performance evaluation, algorithms, and software development. Our consultants can assist you in applying performance tools effectively to your applications and optimizing code performance. Please visit our Services page for more information.



ParaTools offers a diverse set of training materials for high performance computing and scientific computing on UNIX, Linux, and Windows. Please visit our Training page for more information.

SysFera-DS

SysFera-DS, a toolkit that provides full web based remote visualization and simulation on HPC and Cloud environments is now available from ParaTools.

Value Proposition using ParaTools

Lower Time-to-Solution

Lower Operating Costs	First in Discovery	Improve Capability

"Performance Engineering"

My use cases using TAU

- Community Earth Systems Model (CESM)
  - Fortran
  - Xeon
    - algorithm enhancement
  - Xeon Phi (KNC)
    - measured vector intensity
- GraviT (TACC)
  - C++
  - Xeon: boost::vec methods poor aligment

Methods I used for Performance Engineering

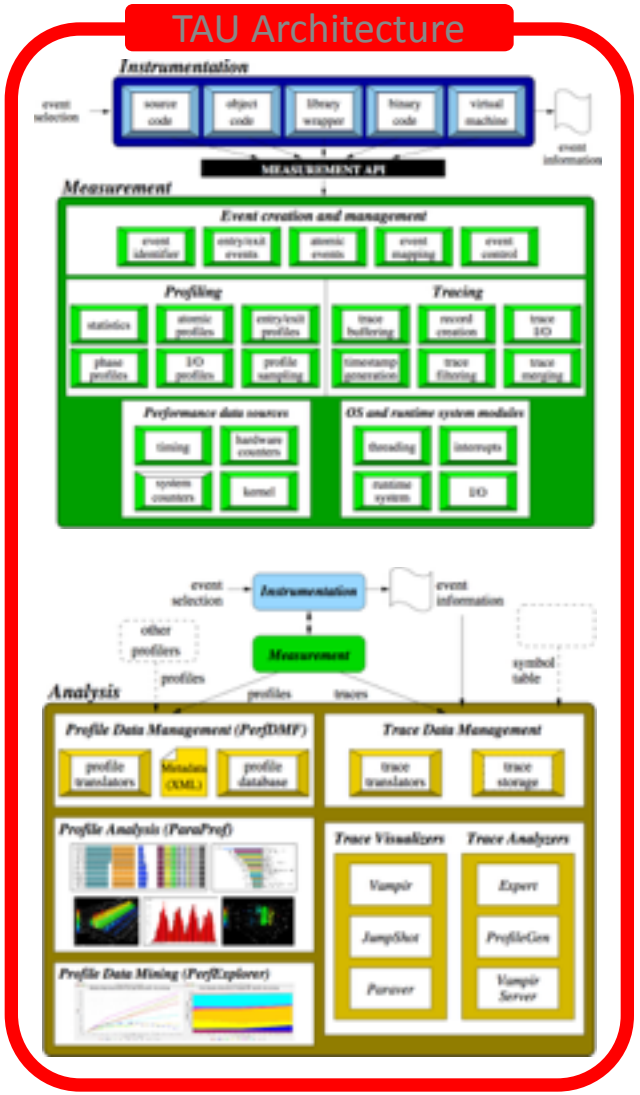
- Find hot spots
  - sampling (library interposition)
    - then instrument
- Understand why hot
  - PAPI metrics
- “Enhance” code region
  - many iterations
- Re-measure
- Run production version for accuracy check
  - V&V

Performance Engineering

THE TAU PERFORMANCE SYSTEM

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/

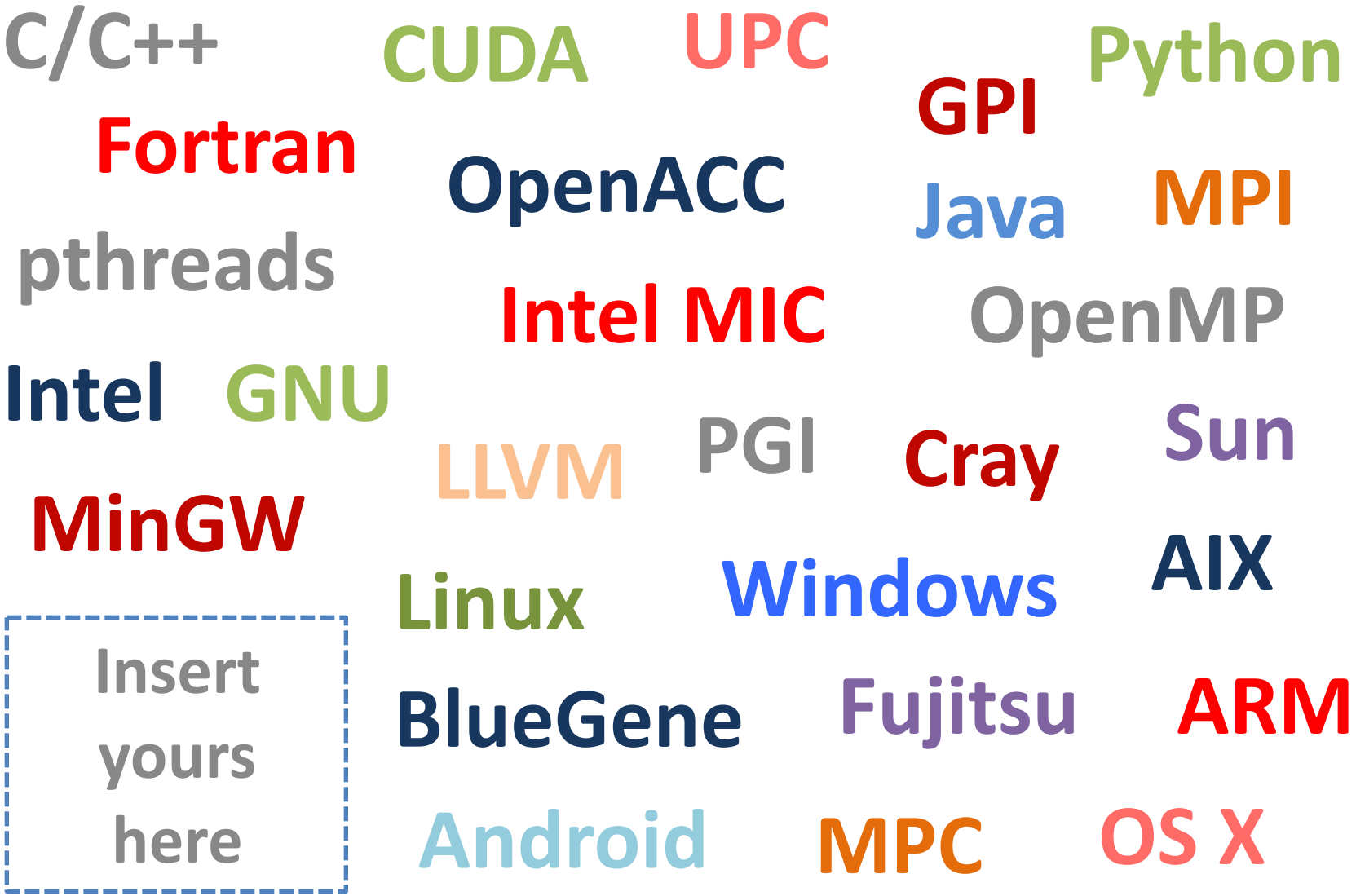


The TAU Performance System®

- 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

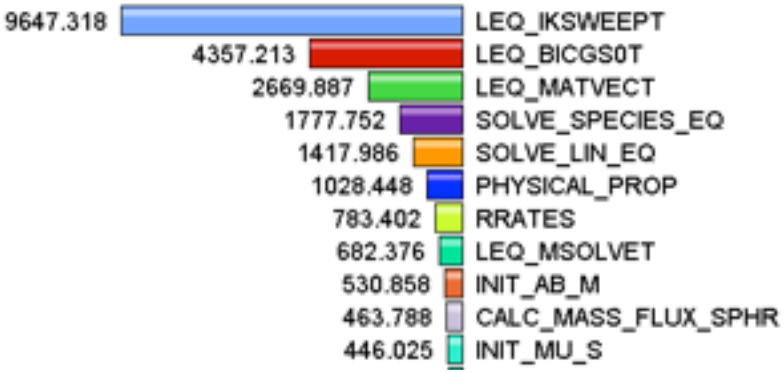
  - Via runtime or at compilation time

TAU Supports All HPC Platforms



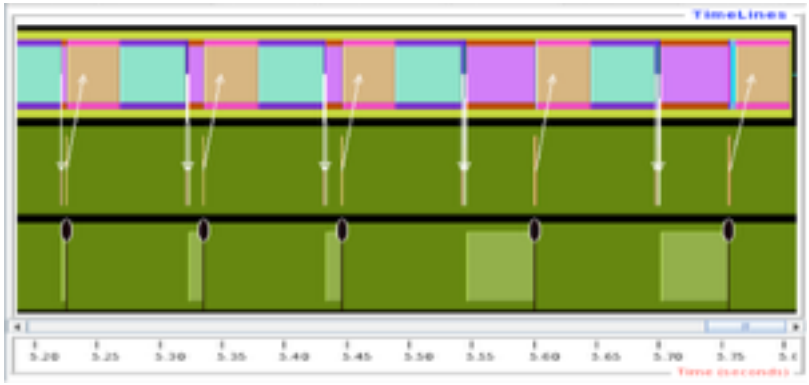
Measurement Approaches

Profiling



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

Tracing

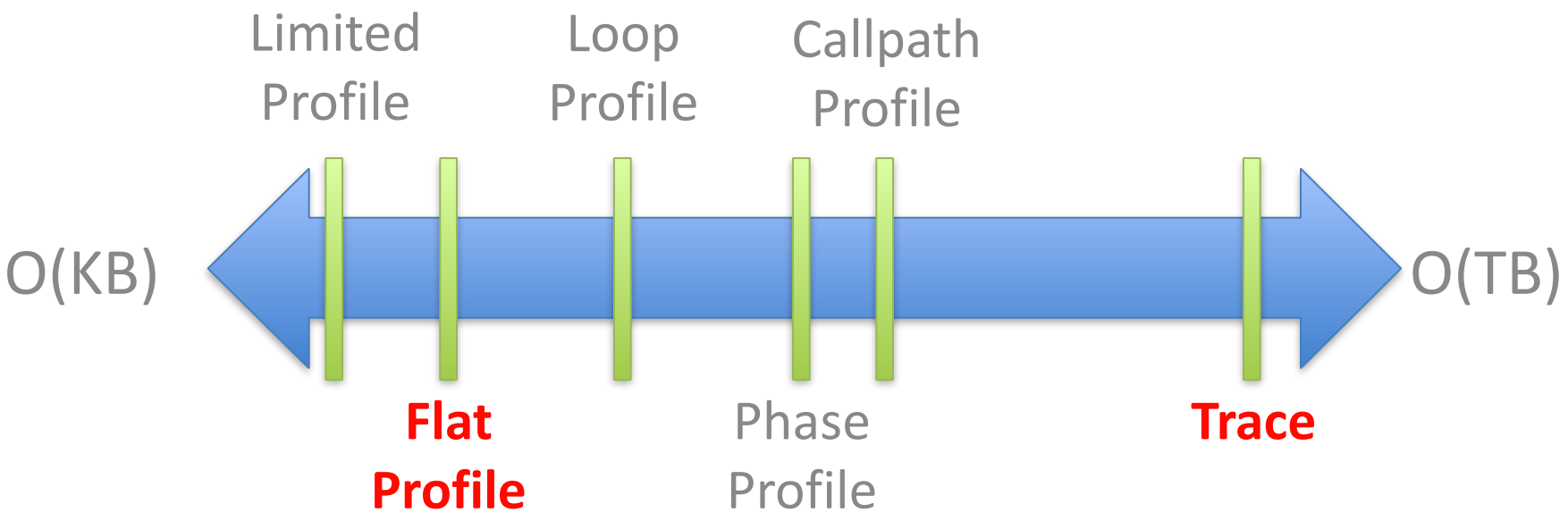


Shows **when** events take place on a timeline

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

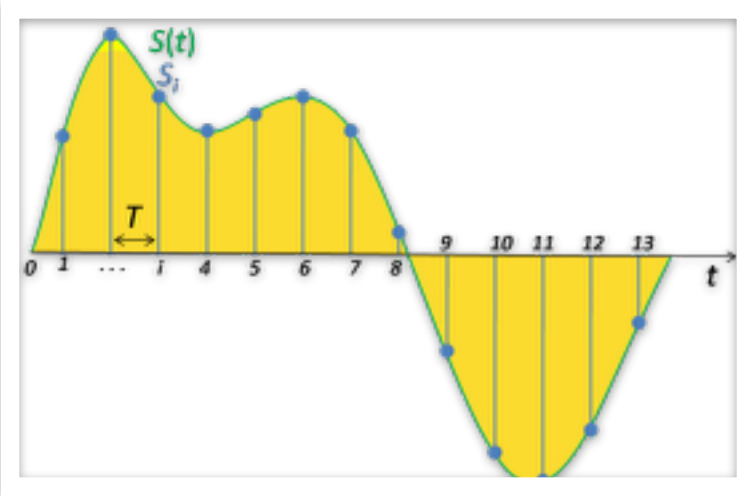
Performance Data Measurement

Direct via Probes

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

- Exact measurement
- Fine-grain control
- Calls inserted into code
  - can be automated with *PDT*

Indirect via Sampling



- No code modification
- Minimal effort
- Relies on debug symbols
  - (**-g** option) and *libunwind*, *bfd* from *binutils*

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.

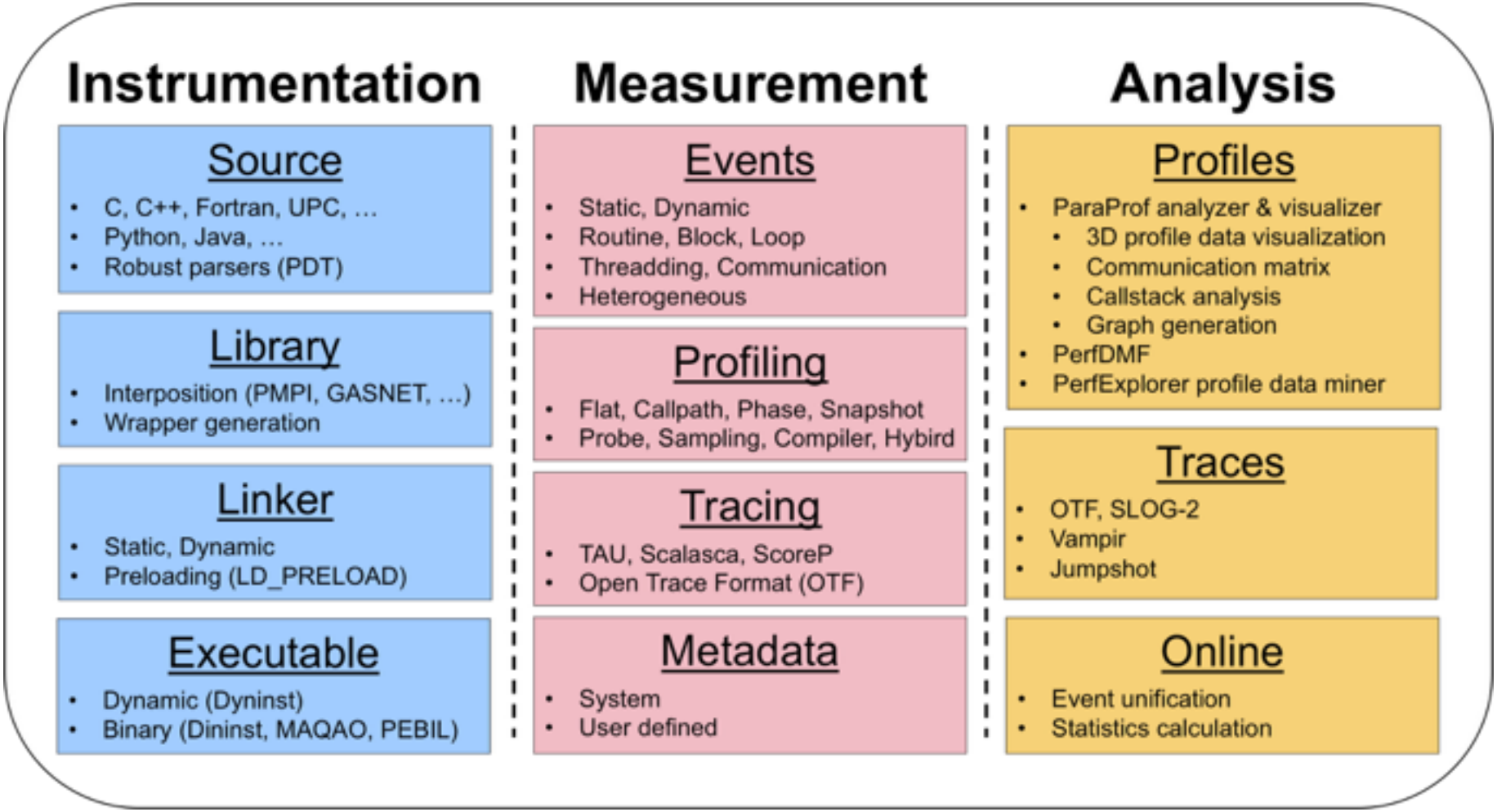
Makefile without TAU

```
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 $<
```

Performance Engineering Workflow



Instrument: Add Probes

- *Source code* instrumentation
  - PDT parsers, pre-processors
- *Wrap* external libraries
  - I/O, MPI, Memory, CUDA, OpenCL, pthread, OMPT
- *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)

What about Hands-On Examples?

- instrumentation_examples
  - example1_tau_exec [interposition, no recompilation]
  - example2_source_inst [PDT automatic instrumentation]
  - example6_hand_inst [direct TAU API]
- measurement_examples
  - example3_papi_knc [source_inst]
  - example4_ompt_knc [interposition]
- analysis_examples
  - example5_snb_trace [source_inst]

Hands-On Preliminaries

- Log into Babbage:
  - ssh <username>@babbage.nersc.gov
  - module list:

Currently Loaded Modulefiles:

1) modules	4) intel/16.0	7) screen/4.2.1
2) nsg/1.2.0	5) impi/5.1.1	8) tmux/1.9a
3) slurm/default	6) usg-default-modules/1.1	

- Get a node :
  - salloc -N 1 "double dash"reservation=ixpug -t 02:00:00 -p regular

Hands-On Preliminaries (2)

- Get examples
  - cp /project/projectdirs/acts/vadlaman/ixpug15_workshop.tgz .
  - or: wget ftp://ftp.paratools.com/ixpug2015/ixpug15_workshop.tgz
  - use : **www.paratools.com/ixpug2015**
- Expand:
  - tar zxvf ixpug15_workshop.tgz
- Check to make sure all there:
  - cd ixpug15_workshop && ls
- <host,mic>_vars.<sh,csh> created for convenience.
  - next slide shows what are possible TAU tool builds
- One can build TAU locally for just Paraprof use

Using TAU: Separate builds

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

```
[vadlaman@bint01 ixpug15_workshop]$ ls /project/projectdirs/acts/vadlaman/tau/2.24.1/intel/16/x86_64/lib/
    Makefile.tau*
/project/projectdirs/acts/vadlaman/tau/2.24.1/intel/16/x86_64/lib/Makefile.tau-icpc-papi-mpi-pdt
/project/projectdirs/acts/vadlaman/tau/2.24.1/intel/16/x86_64/lib/Makefile.tau-icpc-papi-mpi-pdt-openmp
/project/projectdirs/acts/vadlaman/tau/2.24.1/intel/16/x86_64/lib/Makefile.tau-icpc-papi-ompt-mpi-pdt-openmp
/project/projectdirs/acts/vadlaman/tau/2.24.1/intel/16/x86_64/lib/Makefile.tau-icpc-papi-ompt-pdt-openmp
```

```
[vadlaman@bint01 ixpug15_workshop]$ ls /project/projectdirs/acts/vadlaman/tau/2.24.1/intel/16/mic_linux/lib/Makefile.tau*
/project/projectdirs/acts/vadlaman/tau/2.24.1/intel/16/mic_linux/lib/Makefile.tau-icpc-papi-ompt-mpi-pdt-openmp
/project/projectdirs/acts/vadlaman/tau/2.24.1/intel/16/mic_linux/lib/Makefile.tau-icpc-papi-ompt-pdt-openmp
```

Extra tools: I had to build PDT (parser) and install (using system gcc)

```
[vadlaman@bint01 ixpug15_workshop]$ ls /project/projectdirs/acts/vadlaman
pdt  perl-mic  tau  threadspotter
```

Using TAU: A Brief Introduction

1. Choose an appropriate `TAU_MAKEFILE`:

```
$ export TAU_MAKEFILE=< of your choice>
$ export TAU_OPTIONS='-optVerbose -optContinueBeforeOmp
…'
      # (see "tau_compiler.sh -help" for more options)
```

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

```
$ mpiifort foo.f90
```

changes to

```
$ tau_f90.sh foo.f90
```

3. Execute application (assuming in interactive allocation):

```
$mpirun.<host,mic> -n  # <-hosts> <names> <-env for mic,> ./a.out
```

4. Analyze performance data:

`pprof` (for text based profile display)

`paraprof` (for GUI)

Rubber hits the road

- Presenter will demonstrate example
- Then attendees will duplicate and take notes + questions
- Hope to allow time at end of hands on to start exploring your code.

Vector Intensity (512 bit Intel VPU)

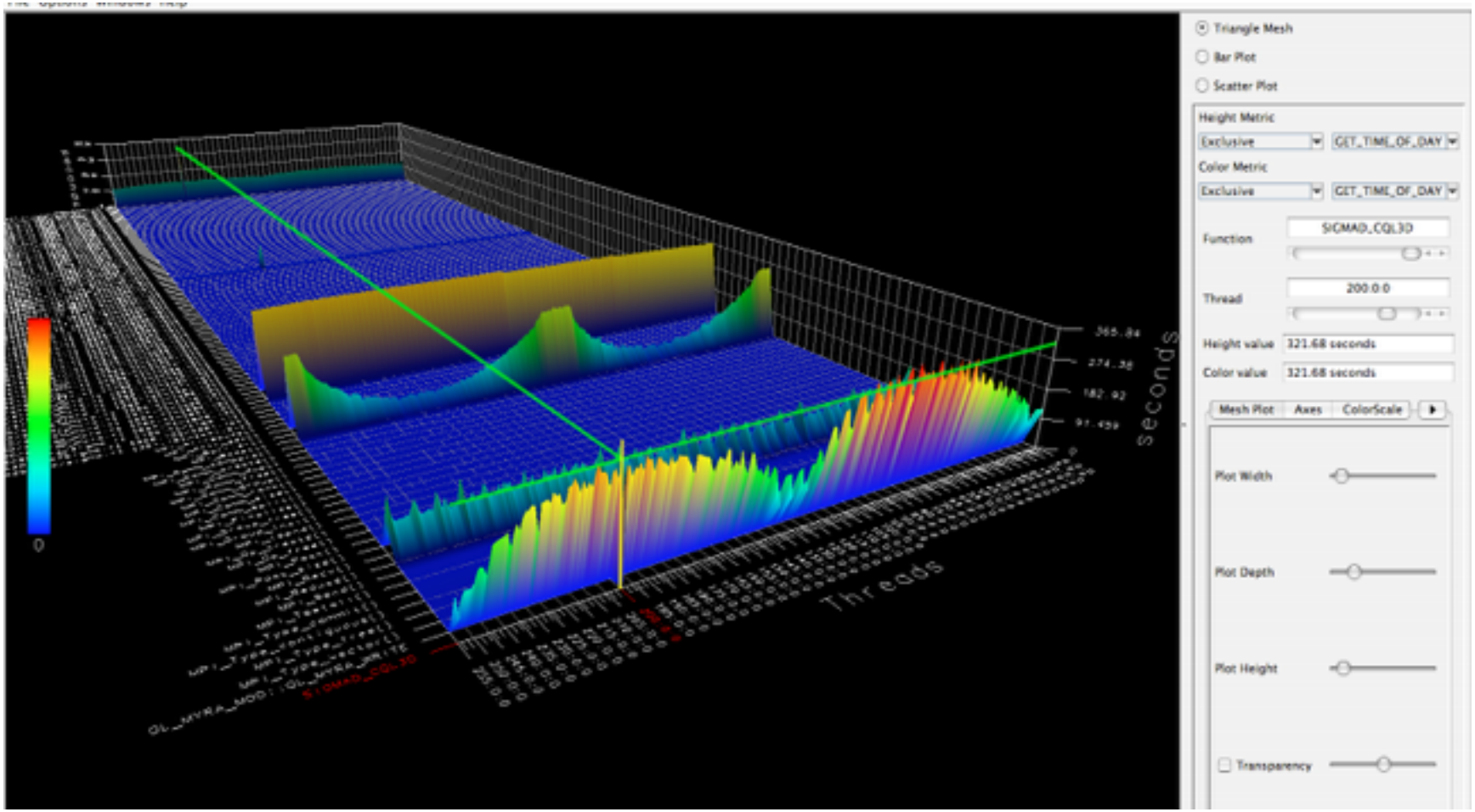
VPU_ELEMENTS_ACTIVE event is incremented by 16 (for single precision) or 8 (for double precision). Scalar FP operations are generally implemented by the compiler using the vector registers, but with a mask indicating that they apply to only one vector element. So a reasonable rule of thumb to see how well a loop is vectorized is to add up the values of VPU_ELEMENTS_ACTIVE and VPU_INSTRUCTIONS_EXECUTED for every assembly instruction in the loop and take the ratio. If this number approaches 8 or 16 then there's a good chance that the loop is well vectorized. If the number is much smaller, then the loop was not well vectorized.

Jeffers, James; Reinders, James (2013-02-11). Intel Xeon Phi Coprocessor High-Performance Programming (Kindle Locations 8559-8564). Elsevier Science. Kindle Edition.

Performance Engineering

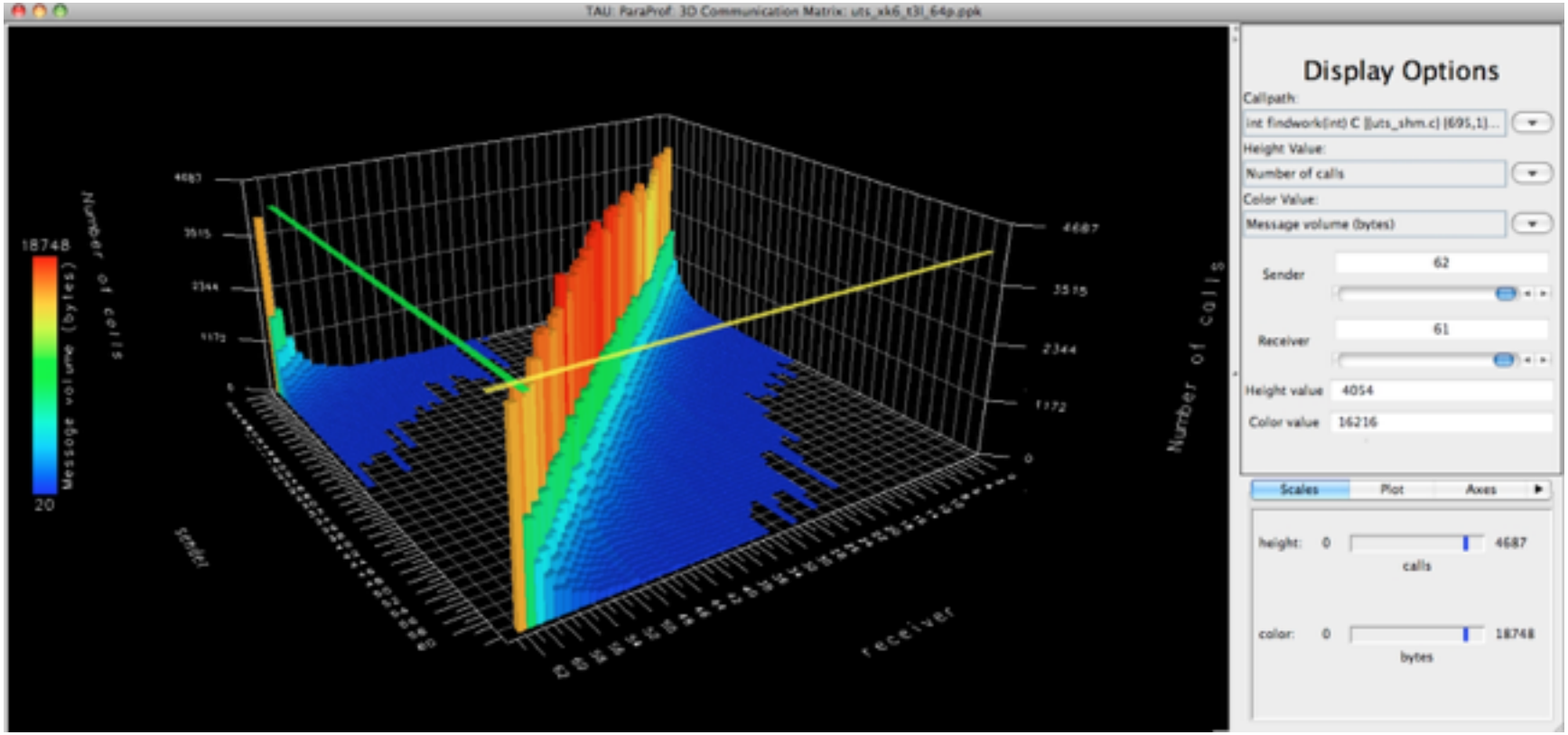
PARAPROF VISUALIZER AND PERFEXPLORER

3D Profile Visualization



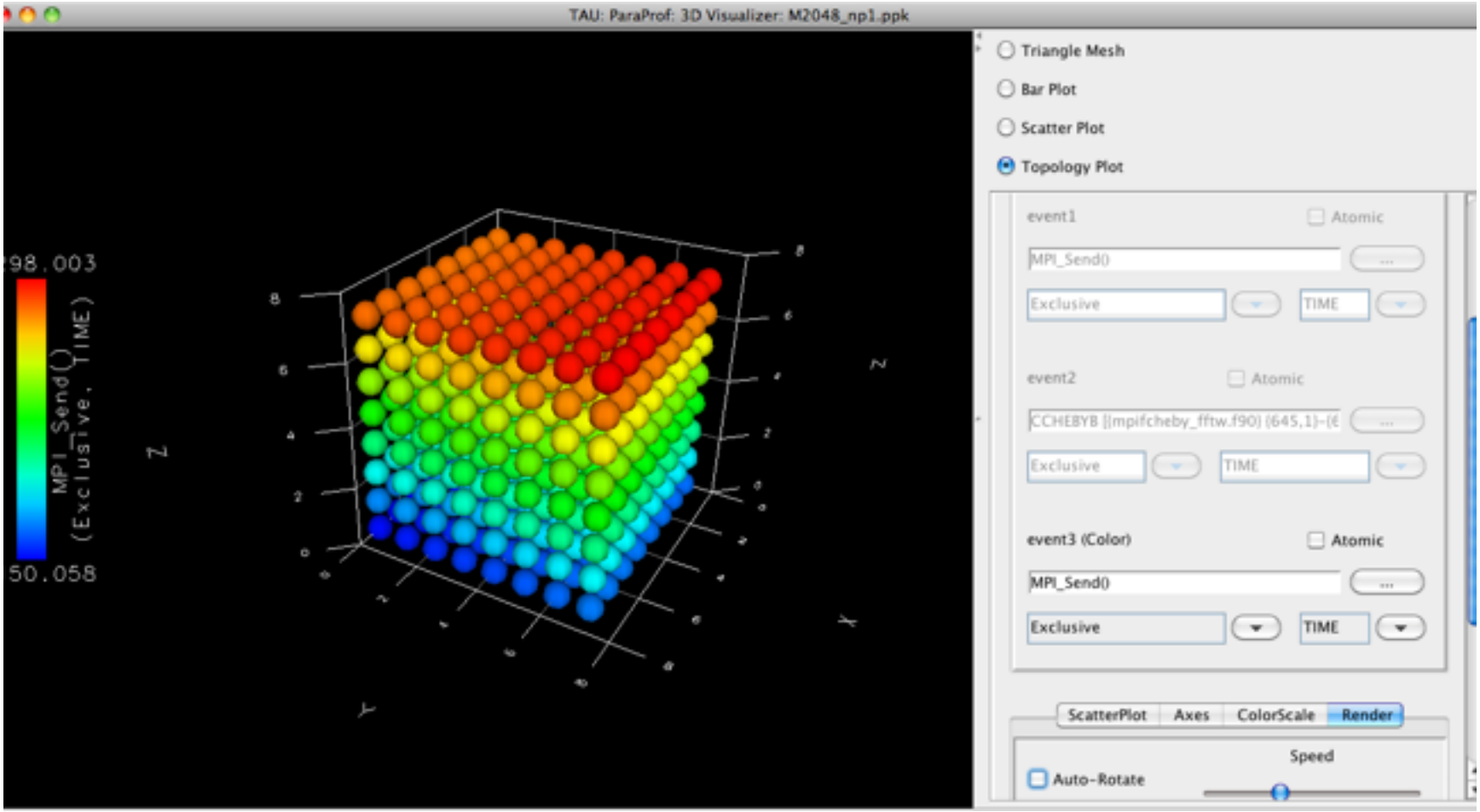
`% paraprof` (Windows → 3D Visualization)

3D Communication Visualization



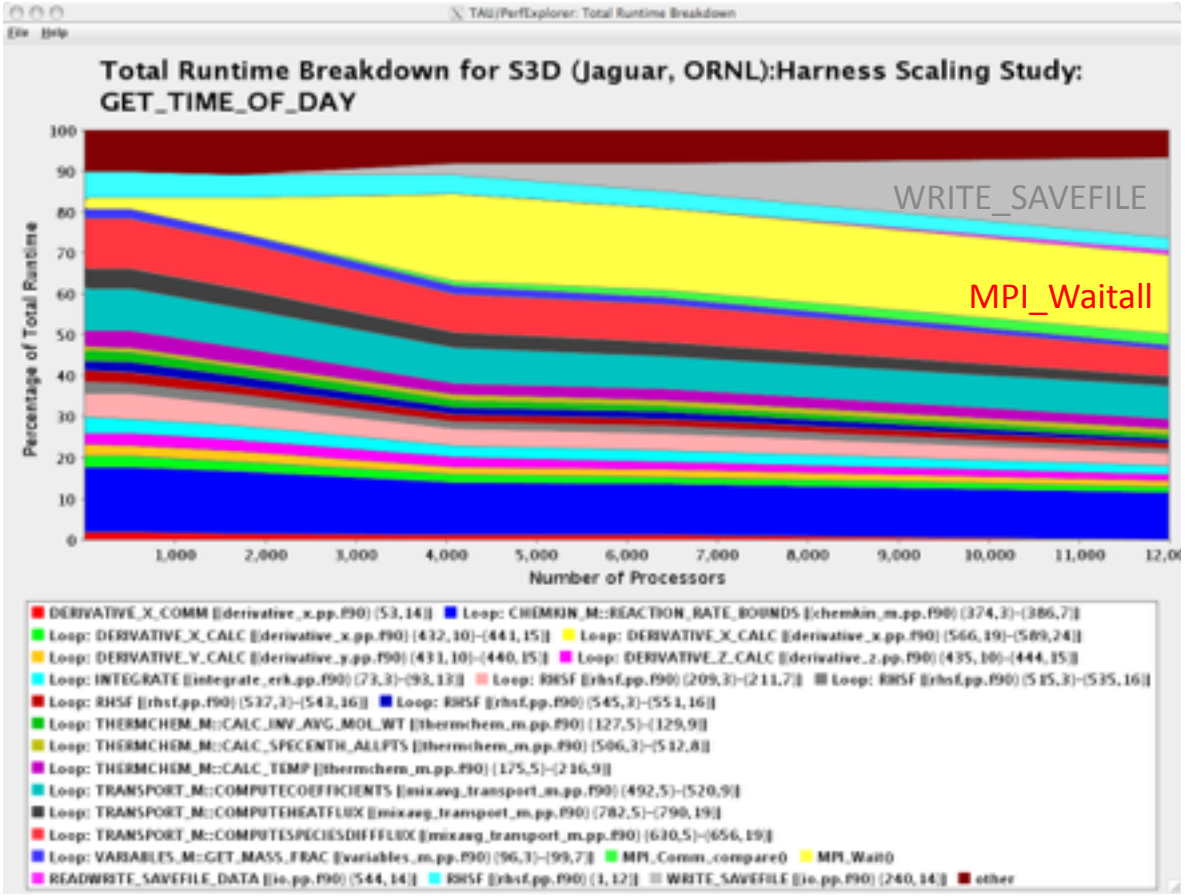

```
% qsub –env TAU_COMM_MATRIX=1 …
% paraprof (Windows → 3D Communication Matrix)
```

3D Topology Visualization



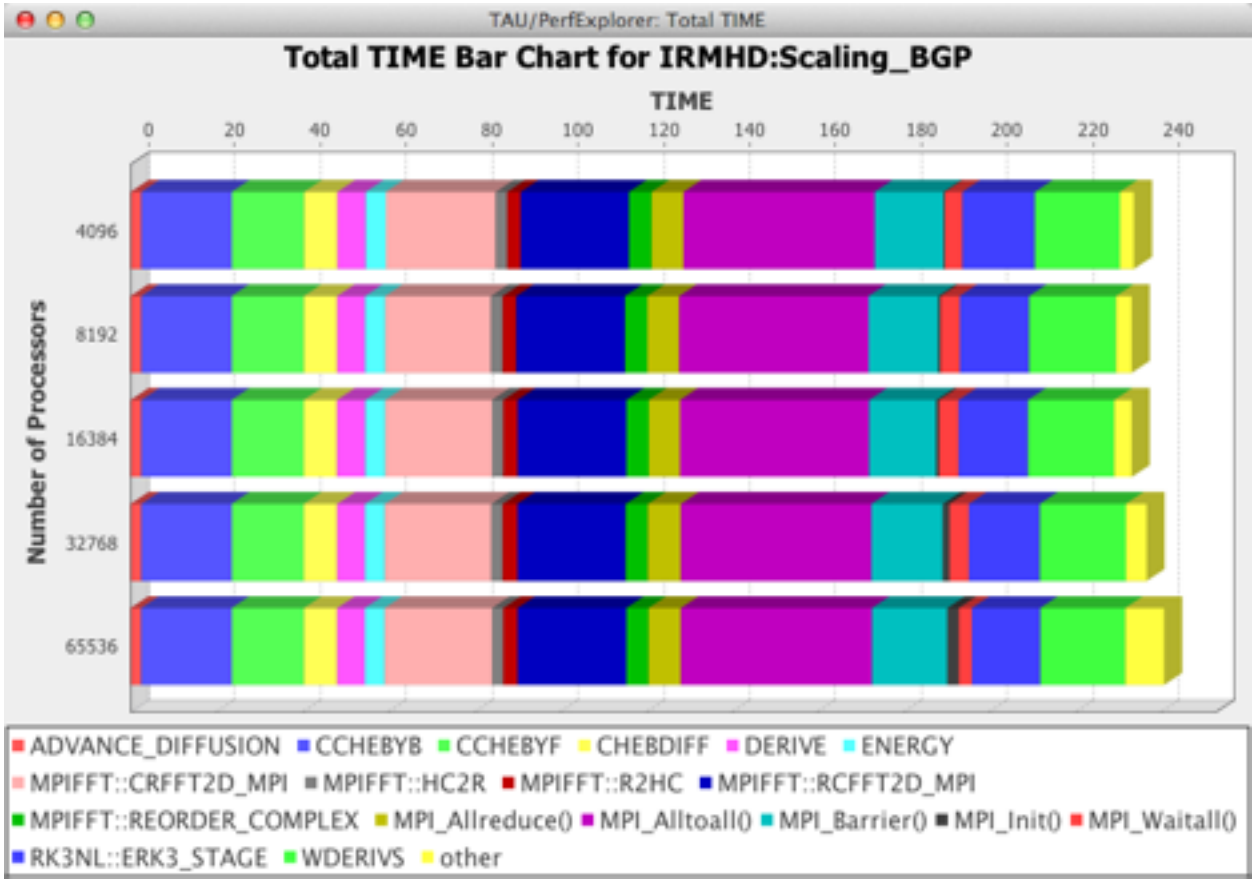
`% paraprof` (Windows → 3D Visualization → Topology Plot)

How Does Each Routine Scale?



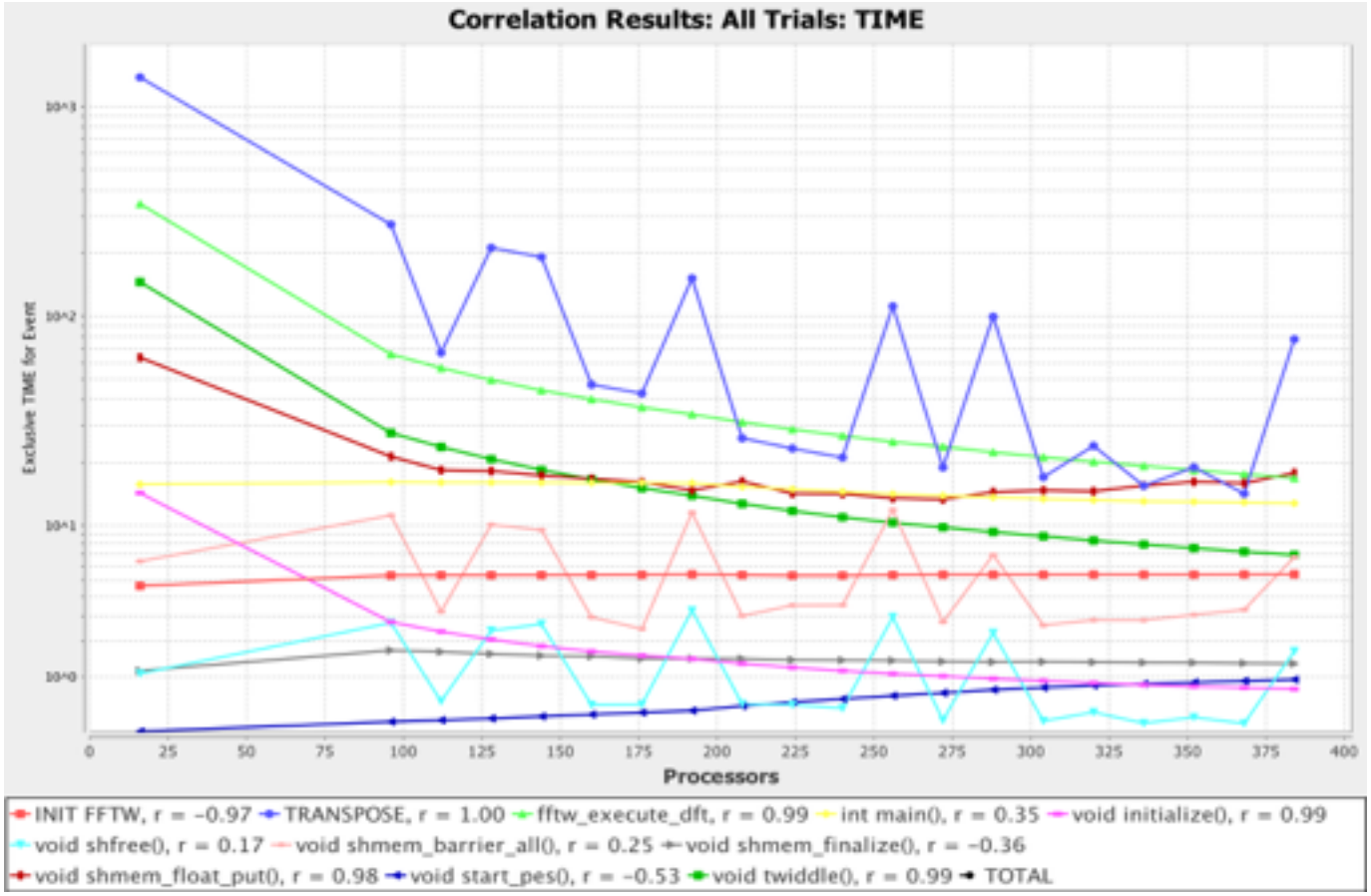
`% perfexplorer` (Charts → Runtime Breakdown)

How Does Each Routine Scale?



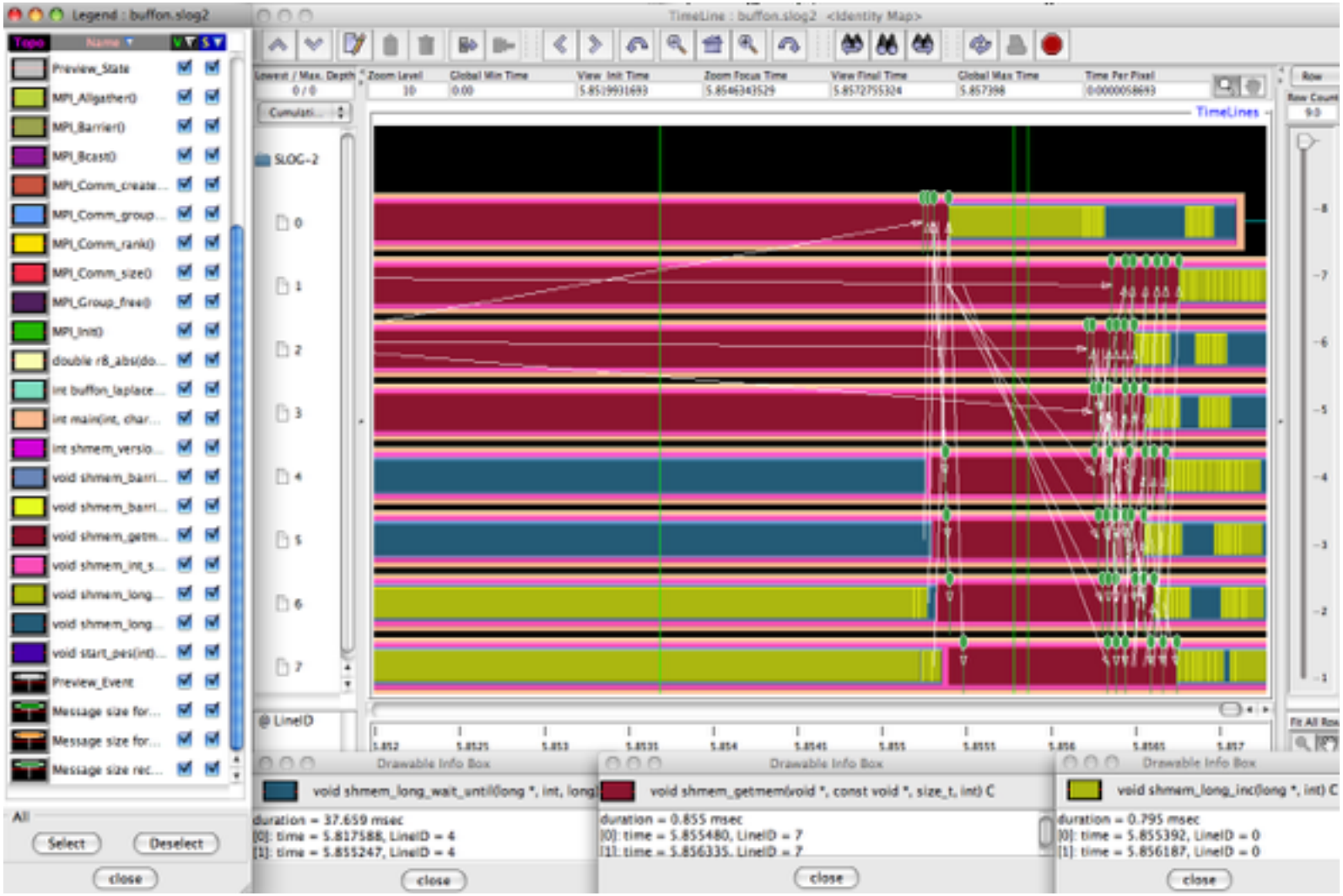
`% perfexplorer` (Charts → Stacked Bar Chart)

Which Events Correlate with Runtime?

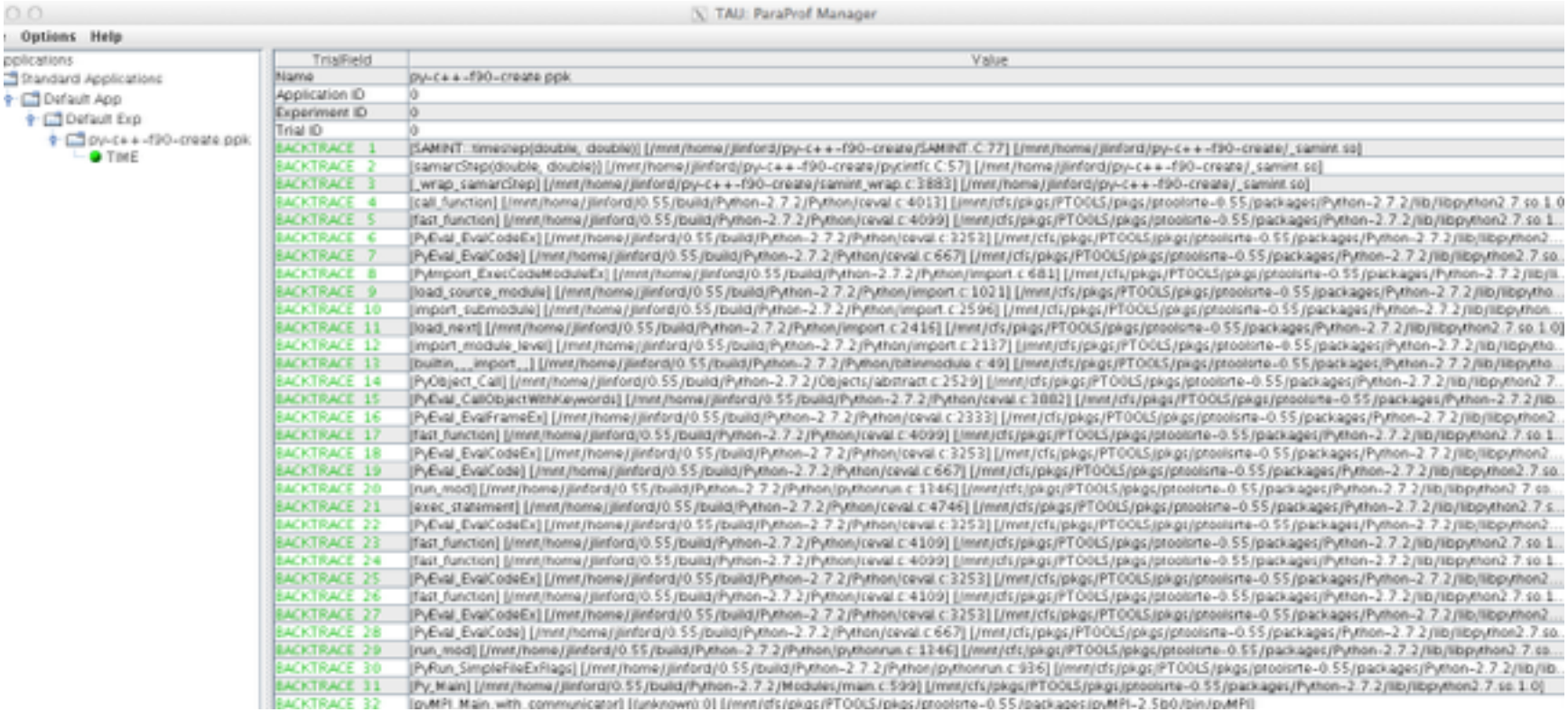


`% perfexplorer` (Charts → Correlate Events with Total Runtime)

When do Events Occur? JUMPSHOT



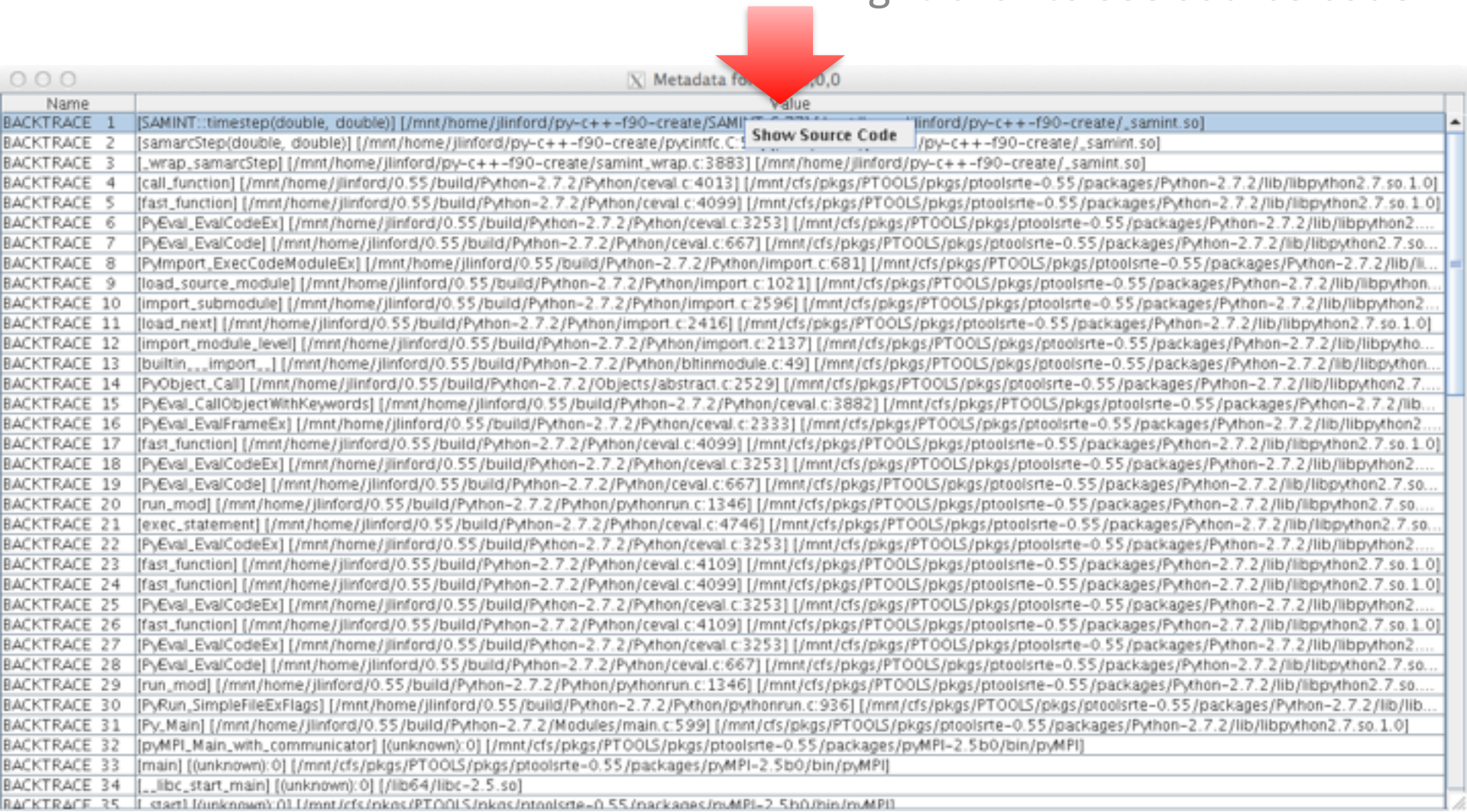
What Caused My Application to Crash?



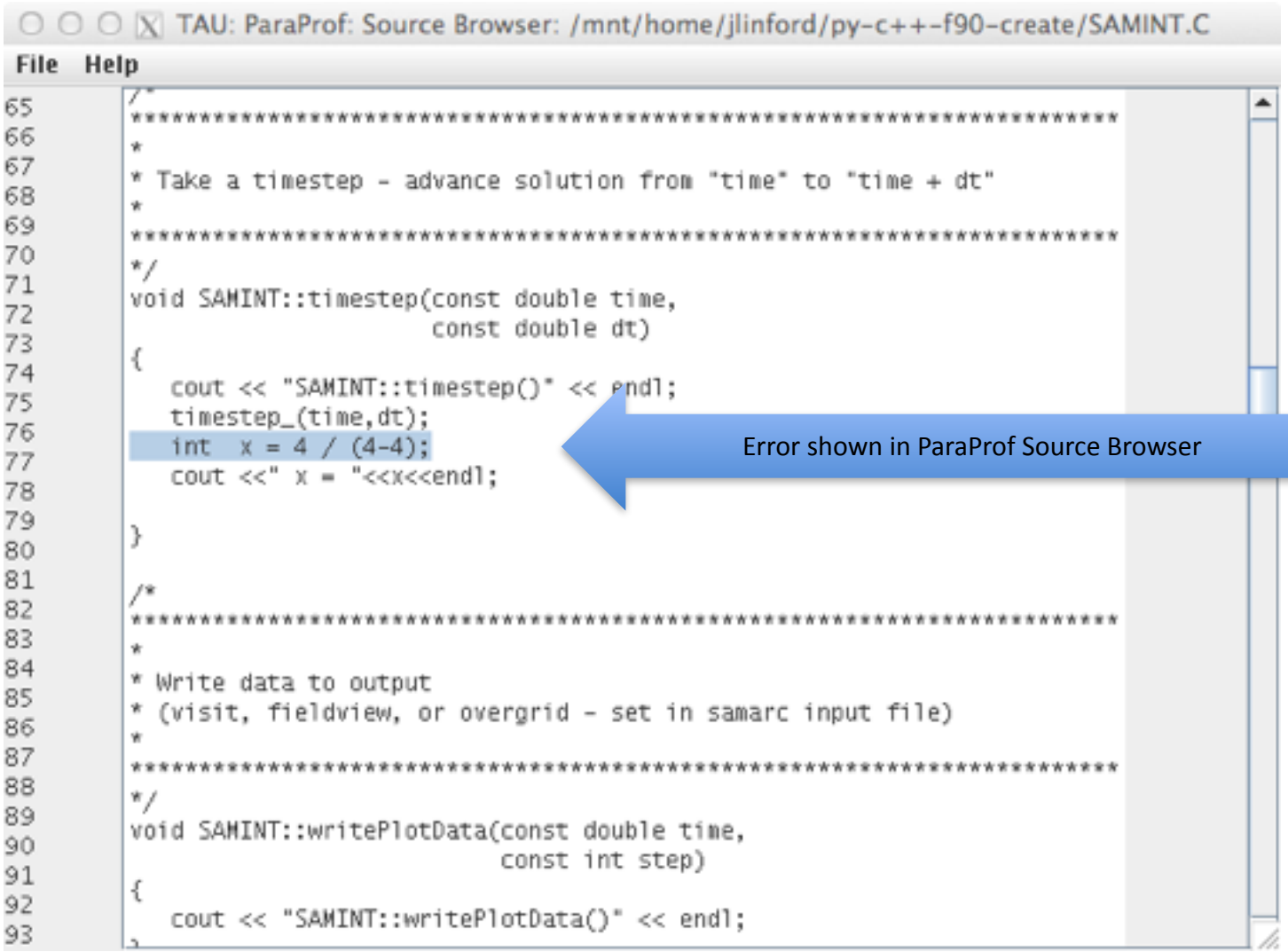

```
% export TAU_TRACK_SIGNALS=1 && mpirun.<> ….
% paraprof
```

What Caused My Application to Crash?

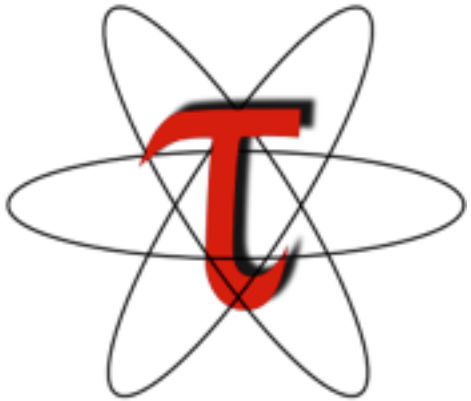
Right-click to see source code



What Caused My Application to Crash?



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

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>)