Profiling and Performance Analysis for Programs using **TAU – Tuning and Analysis Utilities**

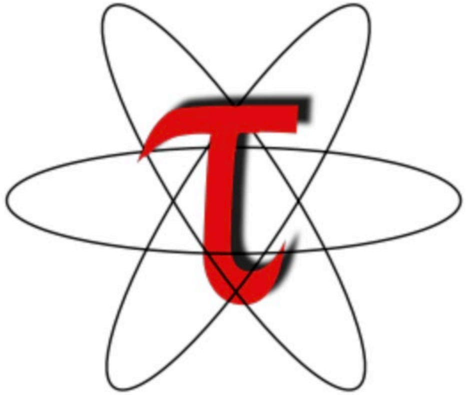


Image taken from Oregon PPT.

By : Tanya Gautam
&
Chaitanya Vinod Patil

What is TAU?

- It is a performance analysis tool that can work with various programs, serial or parallel.
- It can work with C, C++, Java, Fortran, Python and UPC.
- Provides detailed performance metrics (time, memory, I/O, etc).
- Developed & maintained by Performance Research Lab, University of Oregon

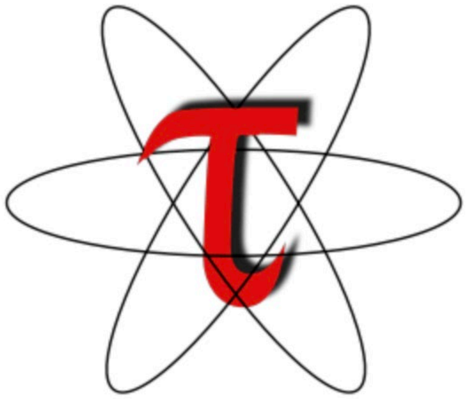


Image taken from Oregon PPT.

Why use TAU?

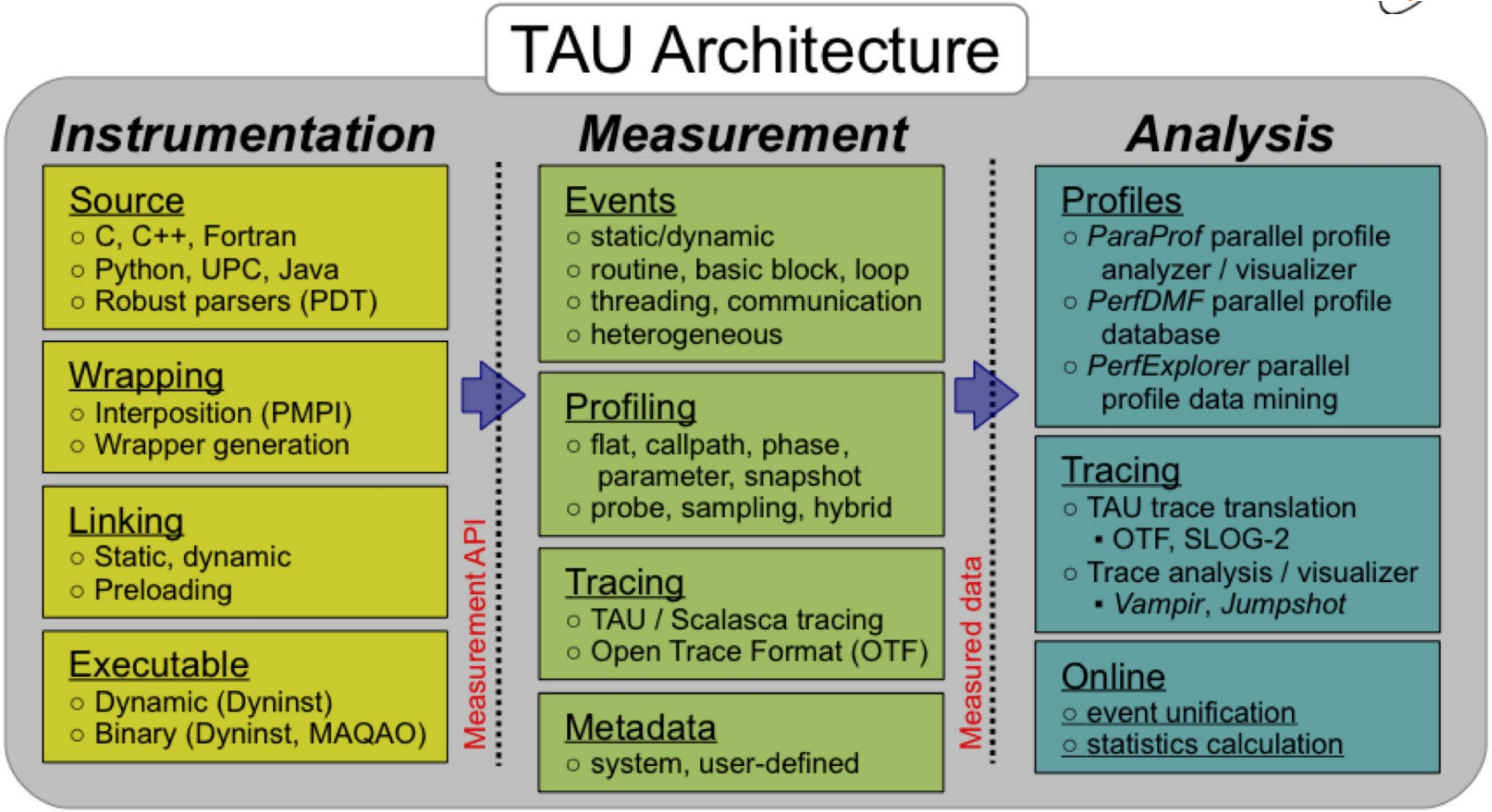
- Flexibility :
  - Works with various programming paradigms like MPI, OpenMP, CUDA, OpenACC, etc.
  - Across various operating systems like windows, mac, linux
  - Provides numerous features, e.g., instrument specific routines, loops, track heap memory, I/O, etc.
- Visual Analysis : Integrated tools like paraprof, Jumpshot for visualizing performance data.

What can TAU do?

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

Text taken from Oregon PPT.

TAU Architecture



TAU Architecture

Instrumentation

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

Wrapping
- Interposition (PMPI)
- Wrapper generation

Linking
- Static, dynamic
- Preloading

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

Measurement

Events
- static/dynamic
- routine, basic block, loop
- threading, communication
- heterogeneous

Profiling
- flat, callpath, phase, parameter, snapshot
- probe, sampling, hybrid

Tracing
- TAU / Scalasca tracing
- Open Trace Format (OTF)

Metadata
- system, user-defined

Analysis

Profiles
- *ParaProf* parallel profile analyzer / visualizer
- *PerfDMF* parallel profile database
- *PerfExplorer* parallel profile data mining

Tracing
- TAU trace translation
  - OTF, SLOG-2
- Trace analysis / visualizer
  - *Vampir*, *Jumpshot*

Online
- event unification
- statistics calculation

Image taken from : Oregon PPT

TAU Workflow Overview

1. Set the required TAU_Makefile and other variables as environment variables.
   - export TAU_MAKEFILE=/path/to/tau_makefile
   - export TAU_METRICS=PAPI_L1_DCM
2. Instrument and compile the code using tau’s executables
   - tau_cxx.sh your_source_code.cpp –o your_executable
3. Run the Program
   - ./your_executable
4. Analyze the Data
   - paraprof

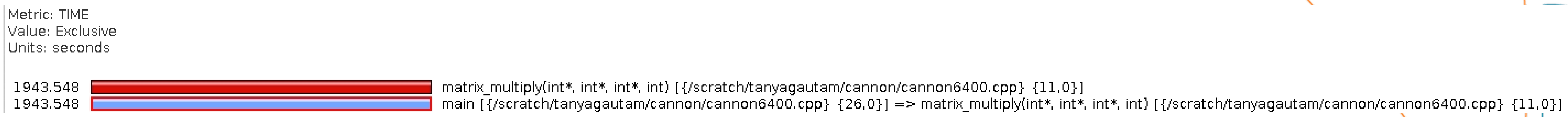
Installing TAU

- Download the TAU tarball
- Then
  - ./configure /add/various/features/you/want/tau/to/install
  - make
  - make install

Tutorial

- Program to analyse - Cannon’s Matrix Multiplication
- We will see –
  1. Serial Execution Time
  2. Flat MPI Profile
  3. Profile that uses different Counters
  4. I/O
  5. Callpath Profile
  6. Communication Matrix
  7. Trace

Serial



```
Metric: TIME
Value: Exclusive
Units: seconds

1943.548   matrix_multiply(int*, int*, int*, int) [{/scratch/tanyagautam/cannon/cannon6400.cpp} {11,0}]
1943.548   main [{/scratch/tanyagautam/cannon/cannon6400.cpp} {26,0}] => matrix_multiply(int*, int*, int*, int) [{/scratch/tanyagautam/cannon/cannon6400.cpp} {11,0}]
```

Total Execution Time = 32 mins

Flat Profile

- Profile : Profiling means calculating the execution time of functions, loops, etc. In other words, how much time is spent in a particular part of the program.
- Now,
  - export TAU_MAKEFILE=/home/username/tau/x86_64/lib/Makefile.tau-mpi-pdt
  - export TAU_PROFILE=1
  - tau_cxx.sh cannon.cpp –o cannon
  - mpirun –n 16 ./cannon

Continued…

- Once the execution is complete it will generate n files of type profile.x.x.x where n is the number of processes.
- To visualize
  - paraprof --pack app.ppk : This will merge all the profile.x.x.x files into one profile file.
  - Move it to your desktop and then : paraprof app.ppk

Different Counters

- Similarly, we can profile with respect to different counters like data misses, cache misses and not just time.
- Before compiling set
  - export TAU_METRICS=PAPI_FP_INS:PAPI_L1_ICM
  - compile : tau_cxx.sh cannon.cpp –o cannon
  - Run : mpirun –n 16 cannon
  - paraprof

I/O

- We can also track how many bytes have been read and/or written.
- Two types of I/O that can be tracked
  - MPI I/O
  - POSIX I/O
- Code flow
  - export TAU_MAKEFILE =/home/username/tau/x86_64/lib/Makefile.tau-mpi-pdt
  - Compile
  - Run
  - paraprof

Callpath Profile

- Callpath Profile by name itself we can infer that it captures the execution path or the calling path followed by the program at the time of execution.
- Code
  - export TAU_CALLPATH=1
  - Compile
  - Run
  - paraprof

Communication Matrix

- This feature captures the communication patterns. It displays the volume, frequency of messages exchanged between different processes in matrix format making it easy to identify communication hotspots, bottlenecks and imbalances in data transfers.
- Code flow
  - export TAU_COMM_MATRIX=1
  - Compile
  - Run
  - paraprof

Trace

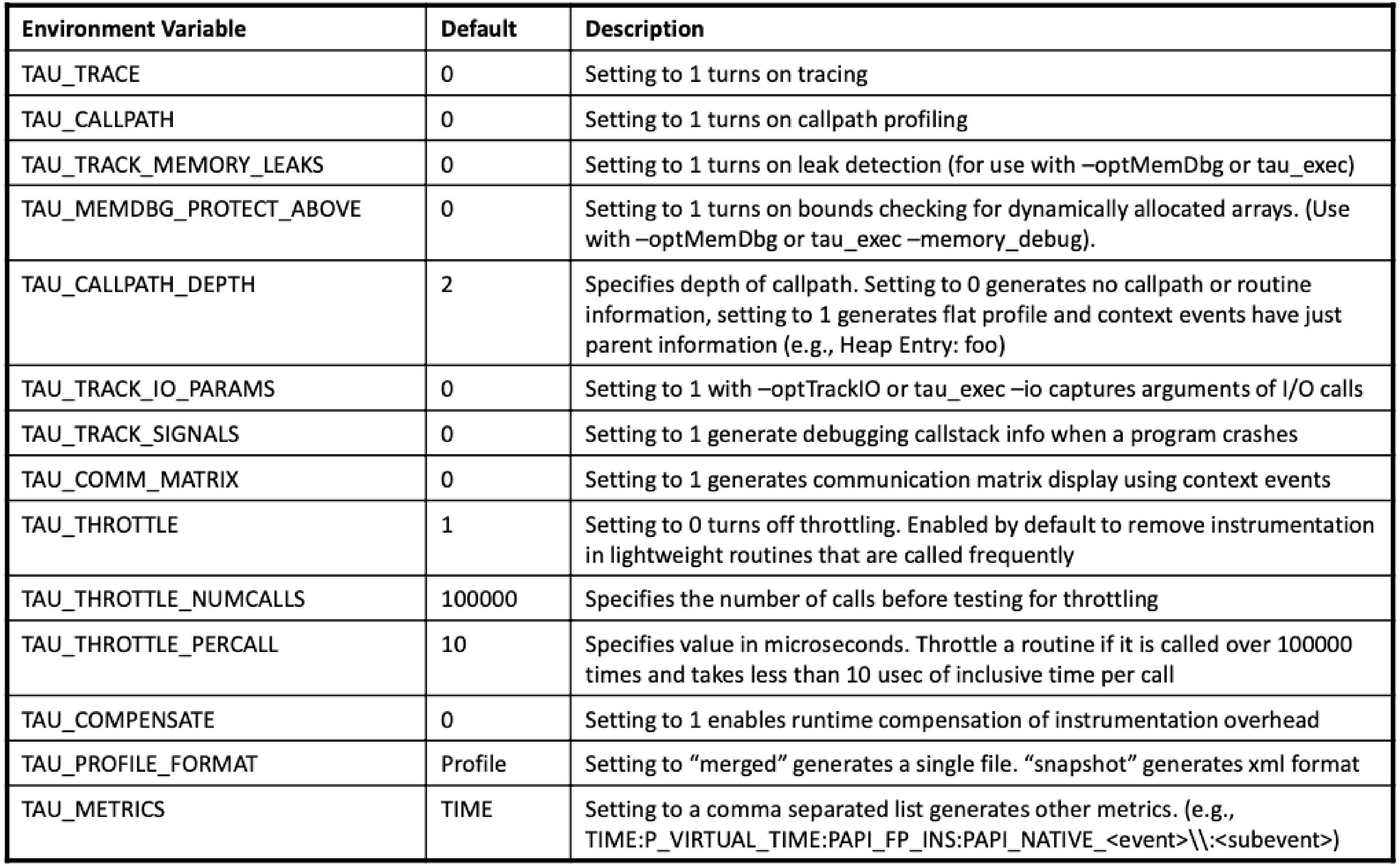
- Trace shows you when the events take place on a timeline. We can use the Jumpshot visual analysis tool to see how the program executed.
- Code flow
  - export TAU_TRACE=1
  - Compile
  - Run -> this will generate n (number of procs) event and trace files.
  - tau_treemerge.pl – merges various event files and trace files.
  - tau2slog2 tau.trc tau.edf –o trace.slog2
  - jumpshot trace.slog2

tau_exec and many more

- Compile as you already do using gcc, g++, python, etc.
- Run : mpirun –n 16 tau_exec ./cannon

- There are still many more features that even we haven't explored e.g.,
  - PerfExplorer
  - Memory tracking/debugging
  - Loop instrumentation, etc.

Various Environment Variables



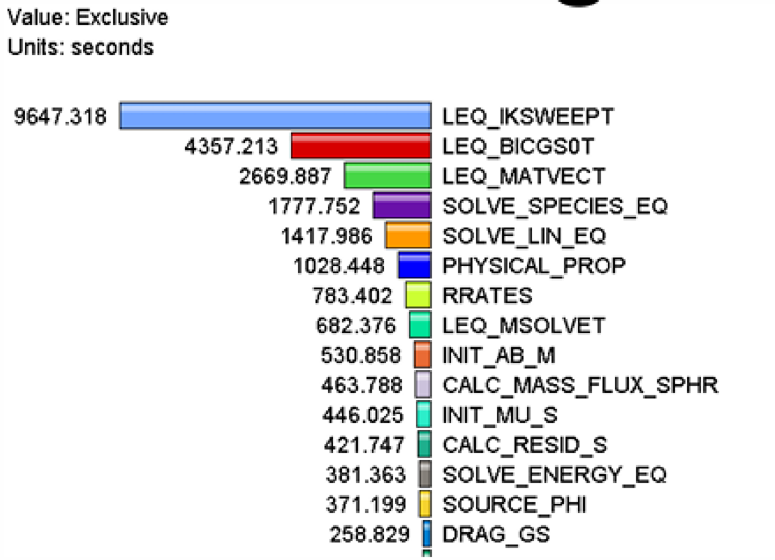
Environment Variable	Default	Description
TAU_TRACE	0	Setting to 1 turns on tracing
TAU_CALLPATH	0	Setting to 1 turns on callpath profiling
TAU_TRACK_MEMORY_LEAKS	0	Setting to 1 turns on leak detection (for use with –optMemDbg or tau_exec)
TAU_MEMDBG_PROTECT_ABOVE	0	Setting to 1 turns on bounds checking 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>)

Image taken from : Oregon PPT

Profiling and Tracing

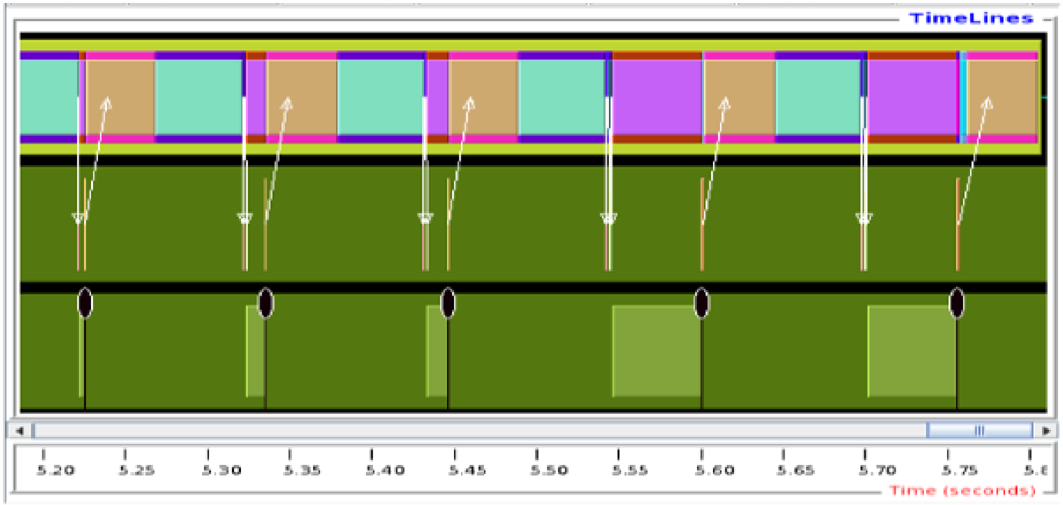
Profiling and Tracing

Profiling



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

Tracing

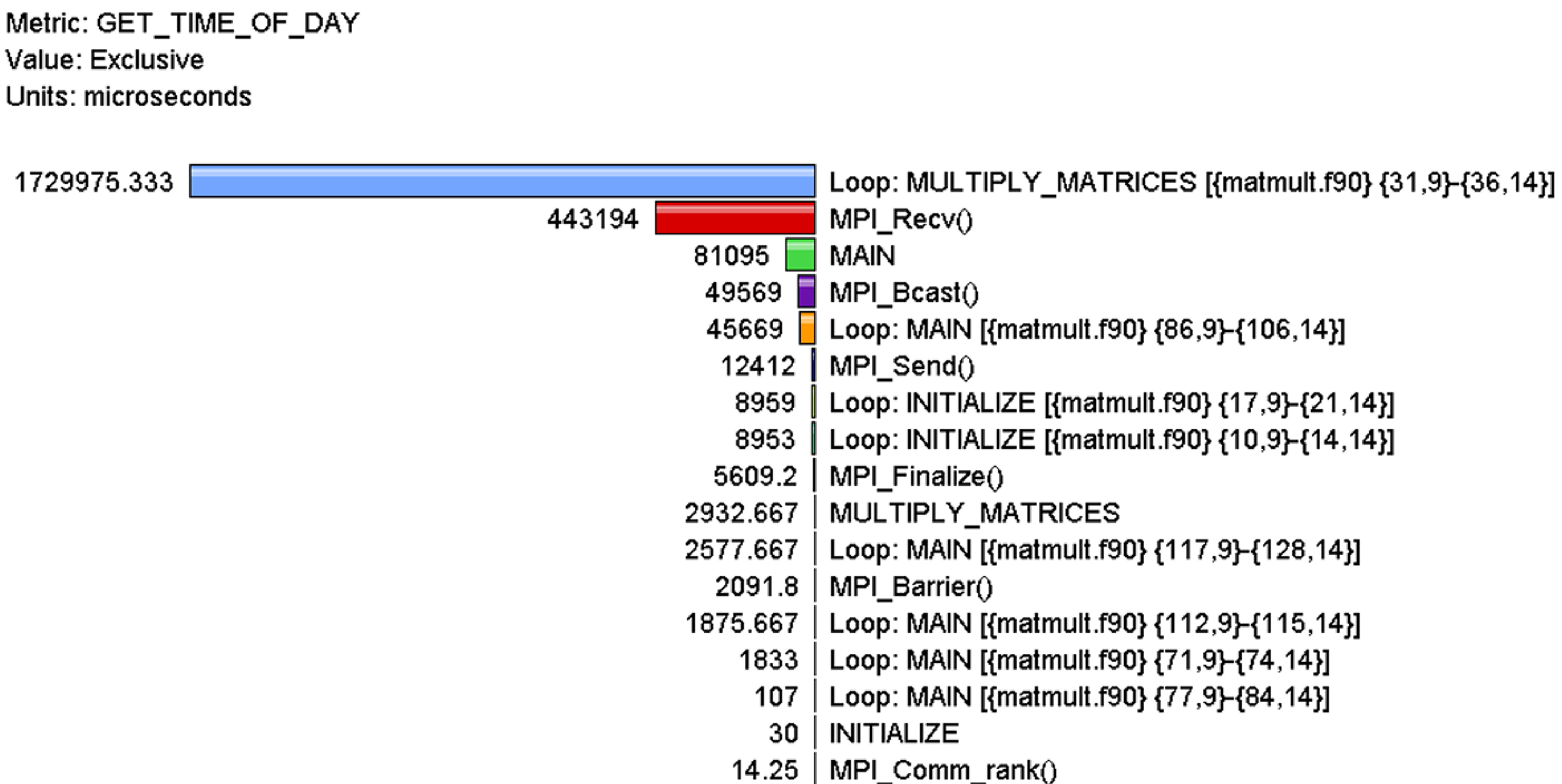


- **Tracing** shows you *when* the events take place on a timeline

Loop Level Instrumentation

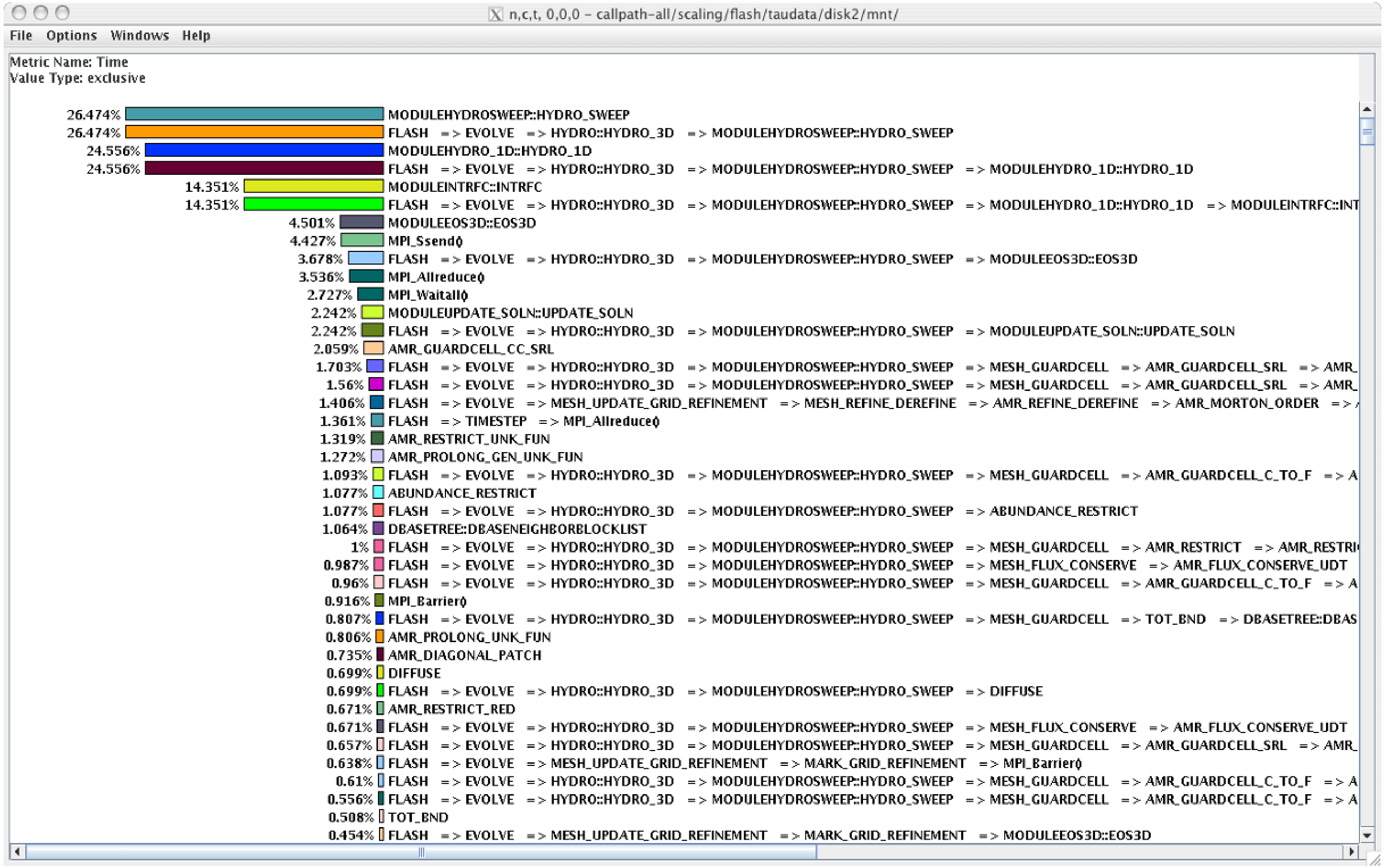
Goal: What loops account for the most time? How much?

Flat profile with wallclock time with loop instrumentation:



Metric: GET_TIME_OF_DAY
Value: Exclusive
Units: microseconds

Callpath Profile

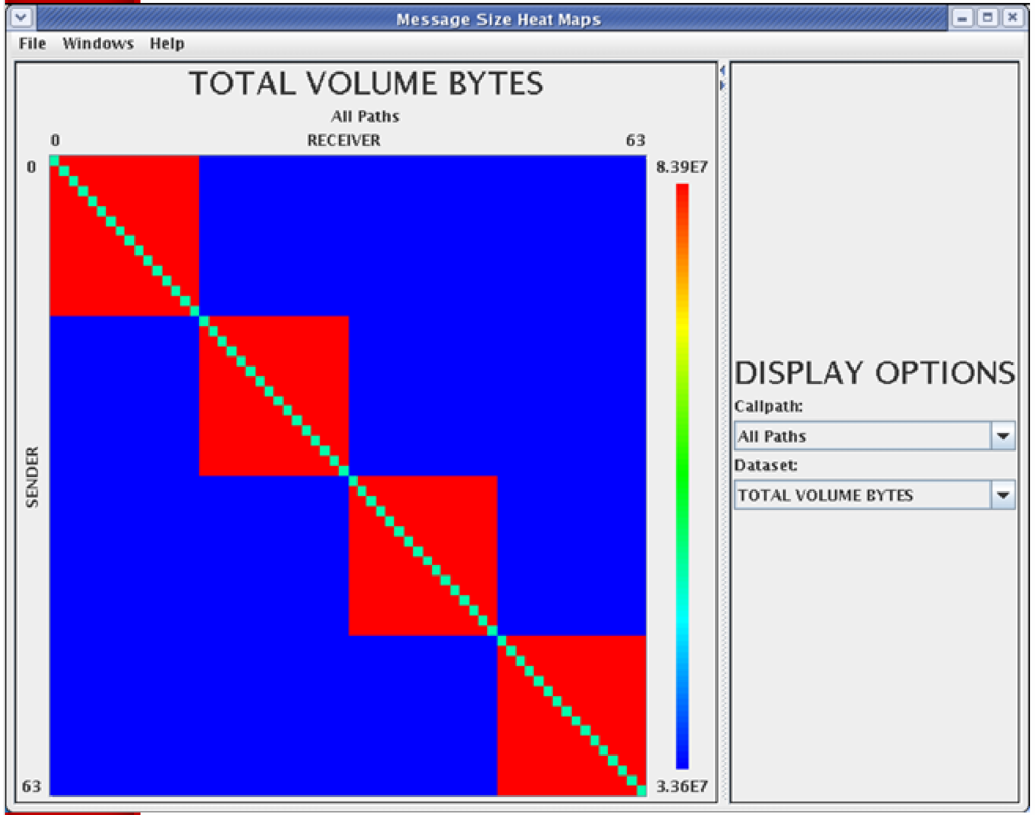

n,c,t, 0,0,0 - callpath-all/scaling/flash/taudata/disk2/mnt/
File
Options
Windows
Help

```
Metric Name: Time
Value Type: exclusive

26.474%  MODULEHYDROSWEEP::HYDRO_SWEEP
26.474%  FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP
24.556%  MODULEHYDRO_1D::HYDRO_1D
24.556%  FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MODULEHYDRO_1D::HYDRO_1D
14.351%  MODULEINTRFC::INTRFC
14.351%  FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MODULEHYDRO_1D::HYDRO_1D  => MODULEINTRFC::INT
 4.501%  MODULEEOS3D::EOS3D
 4.427%  MPI_Ssend()
 3.678%  FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MODULEEOS3D::EOS3D
 3.536%  MPI_Allreduce()
 2.727%  MPI_Waitall()
 2.242%  MODULEUPDATE_SOLN::UPDATE_SOLN
 2.242%  FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MODULEUPDATE_SOLN::UPDATE_SOLN
 2.059%  AMR_GUARDCELL_CC_SRL
 1.703%  FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_GUARDCELL  => AMR_GUARDCELL_SRL  => AMR_
  1.56%  FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_GUARDCELL  => AMR_GUARDCELL_SRL  => AMR_
 1.406%  FLASH  => EVOLVE  => MESH_UPDATE_GRID_REFINEMENT  => MESH_REFINE_DEREFINE  => AMR_REFINE_DEREFINE  => AMR_MORTON_ORDER  => /
 1.361%  FLASH  => TIMESTEP  => MPI_Allreduce()
 1.319%  AMR_RESTRICT_UNK_FUN
 1.272%  AMR_PROLONG_GEN_UNK_FUN
 1.093%  FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_GUARDCELL  => AMR_GUARDCELL_C_TO_F  => A
 1.077%  ABUNDANCE_RESTRICT
 1.077%  FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => ABUNDANCE_RESTRICT
 1.064%  DBASETREE::DBASENEIGHBORBLOCKLIST
    1%  FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_GUARDCELL  => AMR_RESTRICT  => AMR_RESTRI
 0.987%  FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_FLUX_CONSERVE  => AMR_FLUX_CONSERVE_UDT
  0.96%  FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_GUARDCELL  => AMR_GUARDCELL_C_TO_F  => A
 0.916%  MPI_Barrier()
 0.807%  FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_GUARDCELL  => TOT_BND  => DBASETREE::DBAS
 0.806%  AMR_PROLONG_UNK_FUN
 0.735%  AMR_DIAGONAL_PATCH
 0.699%  DIFFUSE
 0.699%  FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => DIFFUSE
 0.671%  AMR_RESTRICT_RED
 0.671%  FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_FLUX_CONSERVE  => AMR_FLUX_CONSERVE_UDT
 0.657%  FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_GUARDCELL  => AMR_GUARDCELL_SRL  => AMR_
 0.638%  FLASH  => EVOLVE  => MESH_UPDATE_GRID_REFINEMENT  => MARK_GRID_REFINEMENT  => MPI_Barrier()
  0.61%  FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_GUARDCELL  => AMR_GUARDCELL_C_TO_F  => A
 0.556%  FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_GUARDCELL  => AMR_GUARDCELL_C_TO_F  => A
 0.508%  TOT_BND
 0.454%  FLASH  => EVOLVE  => MESH_UPDATE_GRID_REFINEMENT  => MARK_GRID_REFINEMENT  => MODULEEOS3D::EOS3D
```

Communication Matrix Display

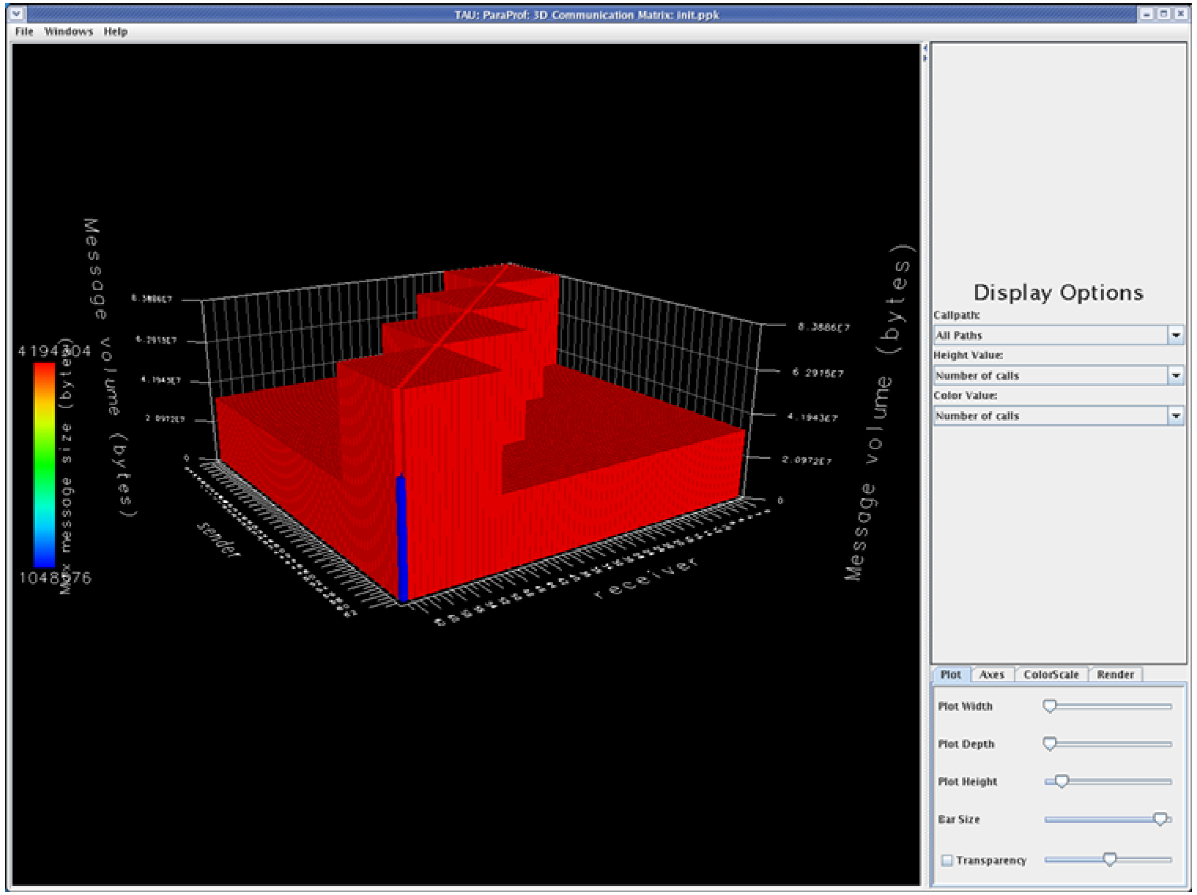
Goal: What is the volume of inter-process communication? Along which calling path?



DISPLAY OPTIONS

Callpath: All Paths

Dataset: TOTAL VOLUME BYTES



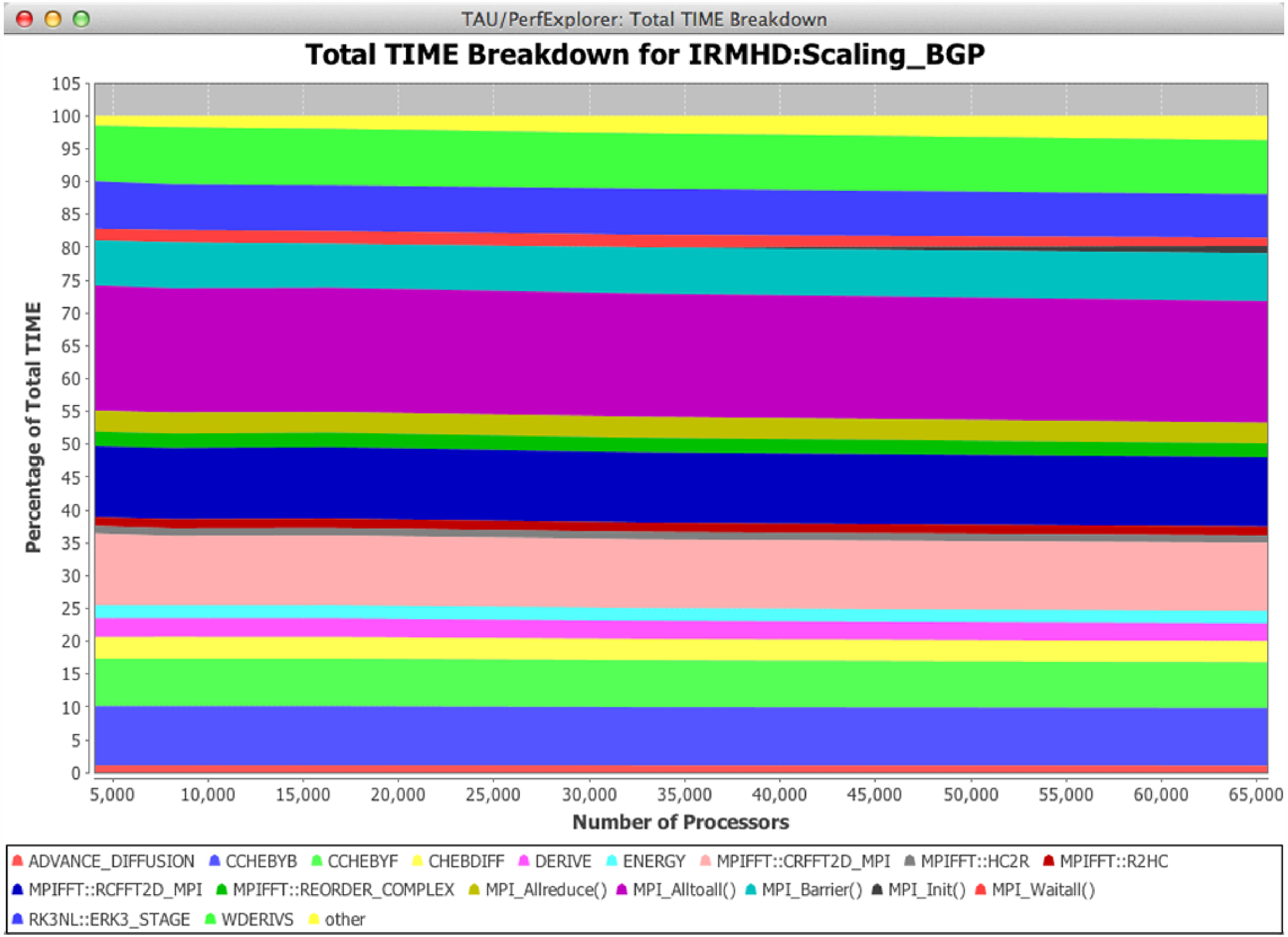
Display Options

Callpath: All Paths

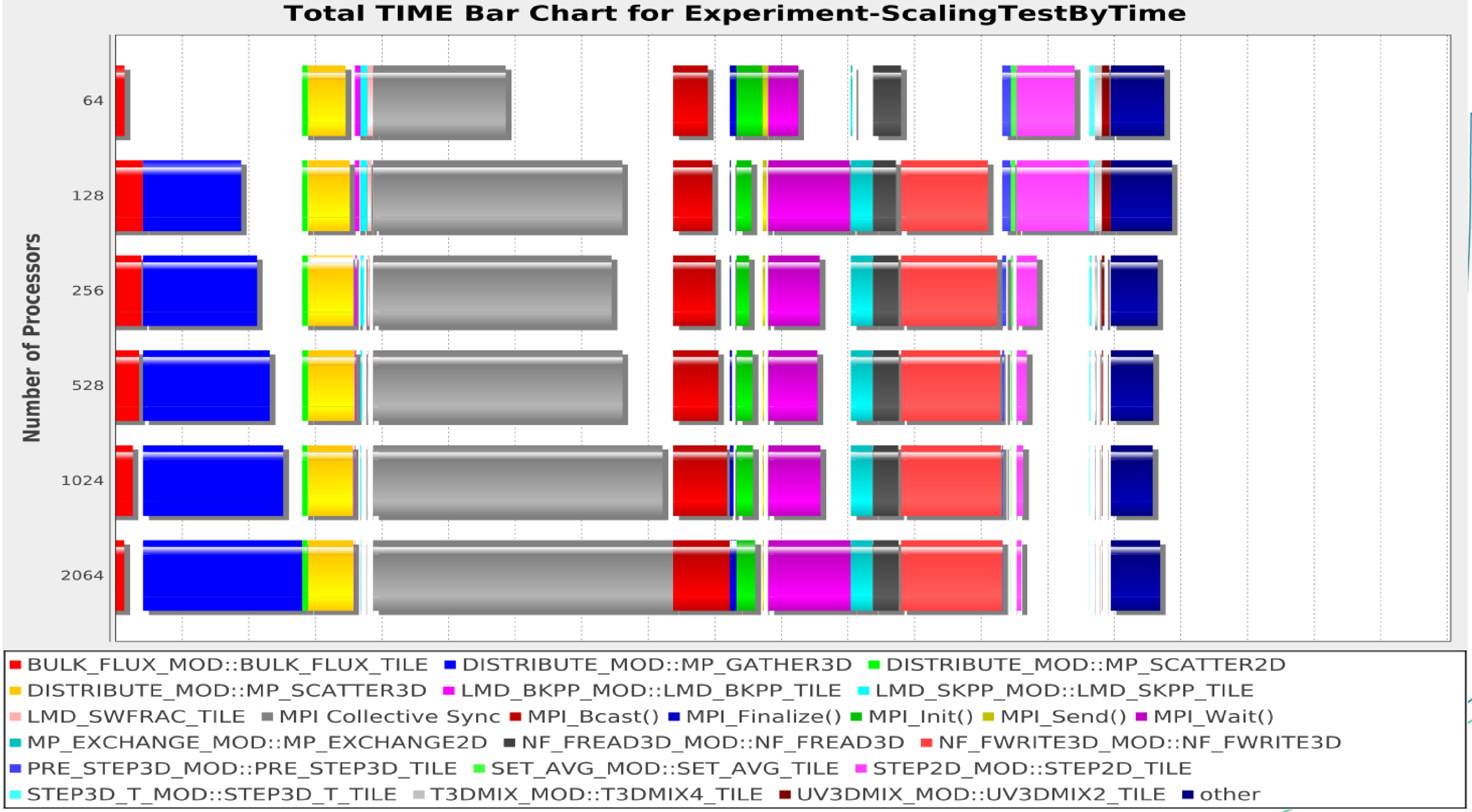
Height Value: Number of calls

Color Value: Number of calls

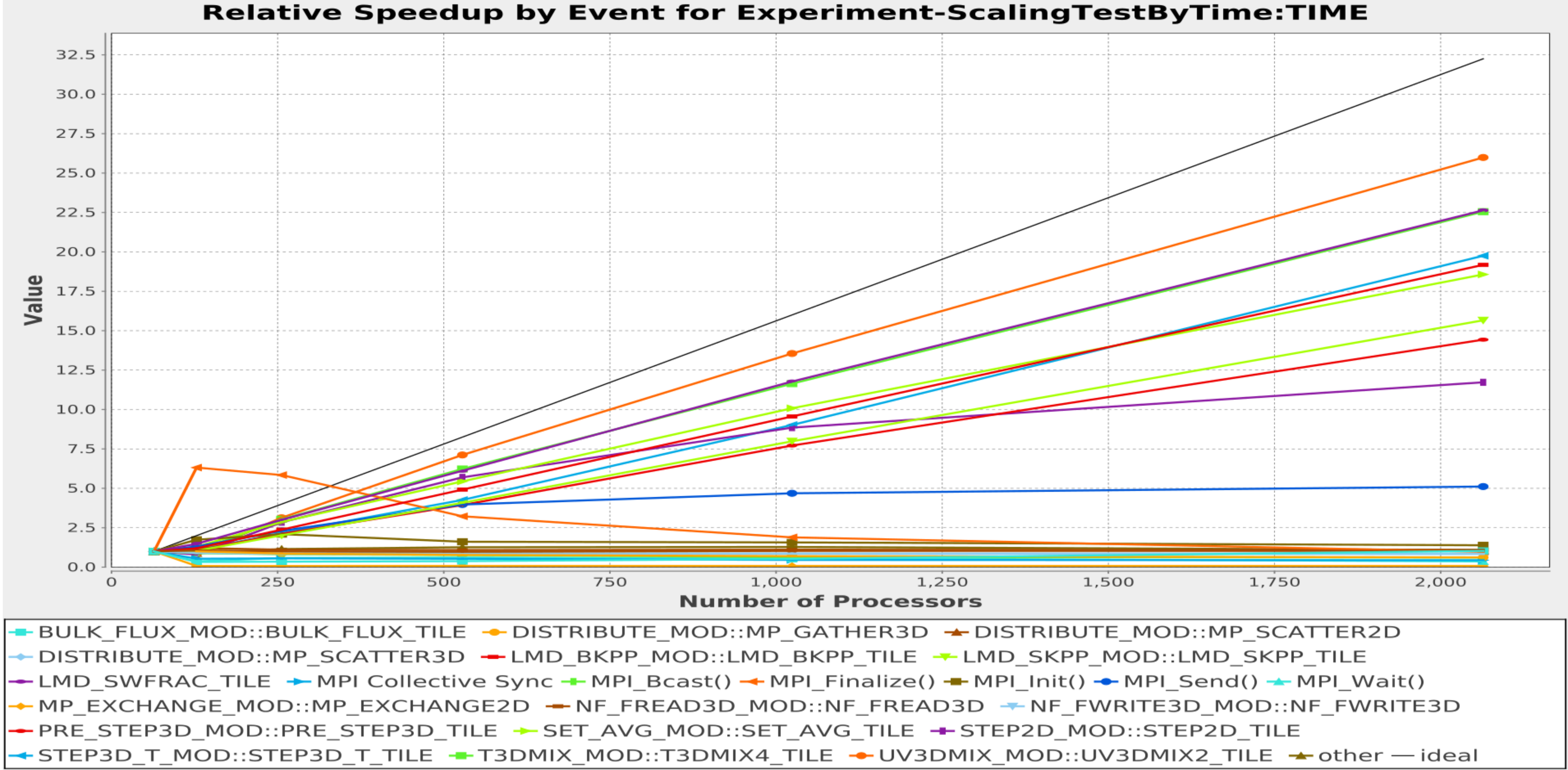
Runtime Breakdown



Scalability Test Feature of TAU using PerfExplorer



Relative Speedup of different functions



Resources and References

- PPTs
  - http://nic.uoregon.edu/~wspear/tau_sea14_tutorial.pdf
  - https://www.sdsc.edu/Events/summerinstitute2012/2012/tau.pdf
- TAU : https://www.cs.uoregon.edu/research/tau/home.php
- PDT : https://www.cs.uoregon.edu/research/tau/pdt_releases/

Thank You!