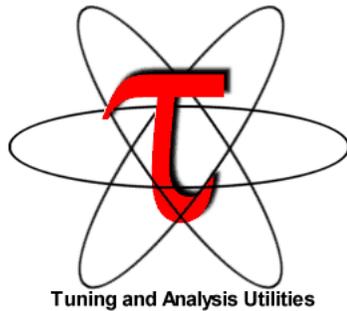


## TAU Performance System®



Sameer Shende  
[sameer@cs.uoregon.edu](mailto:sameer@cs.uoregon.edu)  
University of Oregon  
<http://tau.uoregon.edu>



---

# TAU hands-on exercises

# TAU tutorial exercise objectives

---

- Familiarise with usage of TAU tools
  - complementary tools' capabilities & interoperability
- Prepare to apply tools productively to *your* applications(s)
- Exercise is based on a small portable benchmark code
  - unlikely to have significant optimisation opportunities
- Optional (recommended) exercise extensions
  - analyse performance of alternative configurations
  - investigate effectiveness of system-specific compiler/MPI optimisations and/or placement/binding/affinity capabilities
  - investigate scalability and analyse scalability limiters
  - compare performance on different HPC platforms
  - ...

## Local Installation (*Jureca*)

---

- Setup preferred program environment compilers

```
% jutil env activate -p cjzam11 -A jzam11 (any of your projects)
% module use /p/scratch/share/VI-HPS/JURECA/mf
% module load tau
% mkdir $SCRATCH/$USER
% cd $SCRATCH/$USER
% tar zxvf /p/scratch/share/VI-HPS/examples/NPB3.3-MZ-MPI.tar.gz
% cd NPB3.3-MZ-MPI

# If you have previous performance data from Score-P, you may view it with TAU's paraprof
% paraprof profile.cubex &
```

## NPB-MZ-MPI Suite

---

- The NAS Parallel Benchmark suite (MPI+OpenMP version)

- Available from:

<http://www.nas.nasa.gov/Software/NPB>

- 3 benchmarks in Fortran77
- Configurable for various sizes & classes
- Move into the NPB3.3-MZ-MPI root directory

```
% ls
bin/      common/  jobscript/  Makefile  README.install  SP-MZ/
BT-MZ/    config/  LU-MZ/      README    README.tutorial  sys/
```

- Subdirectories contain source code for each benchmark
  - plus additional configuration and common code
- The provided distribution has already been configured for the tutorial, such that it's ready to “make” one or more of the benchmarks and install them into a (tool-specific) “bin” subdirectory

## NPB-MZ-MPI / BT (Block Tridiagonal Solver)

---

- What does it do?
  - Solves a discretized version of the unsteady, compressible Navier-Stokes equations in three spatial dimensions
  - Performs 200 time-steps on a regular 3-dimensional grid
- Implemented in 20 or so Fortran77 source modules
  
- Uses MPI & OpenMP in combination
  - Proposed hands-on setup on Jureca:
    - 2 compute nodes
    - 8 MPI processes with 6 OpenMP threads each
  - bt-mz\_C.8 should run in less than 20 seconds

## NPB-MZ-MPI / BT: config/make.def : Use default compiler!

```
#           SITE- AND/OR PLATFORM-SPECIFIC DEFINITIONS.
#
#-----
#-----
# Configured for generic MPI with GCC compiler
#-----
#COMPILER = -homp           # Cray/CCE compiler
#COMPILER = -fopenmp      # GNU/GCC compiler
COMPILER = -qopenmp        # Intel compiler

...
#-----
# The Fortran compiler used for MPI programs
#-----
MPIF77 = mpif77 # OpenMPI with Intel compiler

# Alternative variant to perform instrumentation
# MPIF77= tau_f90.sh
# PREP is a generic preposition macro for instrumentation preparation
# MPIF77= $(PREP) mpif77
# MPIF77 = scorep ...

FLINKFLAGS = $(FFLAGS)...
```

Default (no instrumentation)

Uncomment mpif77 and comment  
Score-P's compiler wrapper to  
generate uninstrumented binary!  
Add -dynamic flag to link line.

# Building an NPB-MZ-MPI Benchmark

```
% make
=====
=      NAS PARALLEL BENCHMARKS 3.3      =
=      MPI+OpenMP Multi-Zone Versions   =
=      F77                                =
=====

To make a NAS multi-zone benchmark type

    make <benchmark-name> CLASS=<class> NPROCS=<nprocs>

where <benchmark-name> is "bt-mz", "lu-mz", or "sp-mz"
     <class>           is "S", "W", "A" through "F"
     <nprocs>         is number of processes

[...]

*****
* Custom build configuration is specified in config/make.def *
* Suggested tutorial exercise configuration for HPC systems: *
*      make bt-mz CLASS=C NPROCS=8                          *
*****
```

- Type "make" for instructions
- Type:
  - % make suite
  - to build

# Building an NPB-MZ-MPI Benchmark

```
% make bt-mz CLASS=C NPROCS=8
make[1]: Entering directory `BT-MZ'
make[2]: Entering directory `sys'
cc -o setparams setparams.c -lm
make[2]: Leaving directory `sys'
../sys/setparams bt-mz 8 C
make[2]: Entering directory `../BT-MZ'
mpif77 -c -O3 -fopenmp      bt.f
[...]
mpif77 -c -O3 -fopenmp      mpi_setup.f
cd ../common; mpif77 -c -O3 -fopenmp      print_results.f
cd ../common; mpif77 -c -O3 -fopenmp      timers.f
mpif77 -O3 -fopenmp -o ../bin/bt-mz_C.8 bt.o
initialize.o exact_solution.o exact_rhs.o set_constants.o adi.o
rhs.o zone_setup.o x_solve.o y_solve.o  exch_qbc.o solve_subs.o
z_solve.o add.o error.o verify.o mpi_setup.o ../common/print_results.o
../common/timers.o
make[2]: Leaving directory `BT-MZ'
Built executable ../bin/bt-mz_C.8
make[1]: Leaving directory `BT-MZ'
```

- Specify the benchmark configuration
  - benchmark name:  
**bt-mz**, lu-mz, sp-mz
  - the number of MPI processes: NPROCS=**16**
  - the benchmark class (S, W, A, B, C, D, E):  
**CLASS=C**

Shortcut: `% make suite`

## NPB-MZ-MPI / BT reference execution on Jureca

```
% cd bin
% cp ../jobscript/jureca/reference.sbatch .
% sbatch reference.sbatch
% less npb_btmz_ref.out
NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
Number of zones:  16 x  16
Iterations:  200   dt:  0.000100
Number of active processes:      8
Total number of threads:        48 ( 6.0 threads/process)

Time step   1
Time step  20
  [...]
Time step 180
Time step 200
Verification Successful

BT-MZ Benchmark Completed.
Time in seconds = 18.14
```

- Copy jobscript and launch as a hybrid MPI+OpenMP application

Hint: save the benchmark output (or note the run time) to be able to refer to it later

# NPB-MZ-MPI / BT with TAU's OMPT replacement library on Jureca (tau\_exec -T ompt,papi -ompt)

```
% cd bin
% cp ../jobscript/tau_ompt.sbatch .
% sbatch tau_ompt.sbatch
% cat myrun-0.out
NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
Number of zones: 16 x 16
Iterations: 200 dt: 0.000100
Number of active processes: 8
Total number of threads: 48 ( 6.0 threads/process)

Time step 1
Time step 20
[...]
Time step 180
Time step 200
Verification Successful

BT-MZ Benchmark Completed.
Time in seconds = 18.14
% paraprof &
% pprof -a -m | more
% paraprof --pack bt-mz.ppk
scp to laptop and launch: % paraprof bt-mz.ppk &
```

- Copy jobscript and launch as a hybrid MPI+OpenMP application
- Note: This is using a replacement OpenMP wrapper library! Not the original. When compilers support OMPT, it will be better!

# NPB-MZ-MPI / BT with TAU's OMPT and OTF2 output for Vampir on Jureca (TAU\_TRACE\_FORMAT=otf2; tau\_exec -T ompt -ompt)

```
% cd bin
% cp ../jobscript/tau_otf.sbatch .
% sbatch tau_otf.sbatch
% cat myrun-0.out
NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
Number of zones: 16 x 16
Iterations: 200 dt: 0.000100
Number of active processes: 8
Total number of threads: 48 ( 6.0 threads/process)

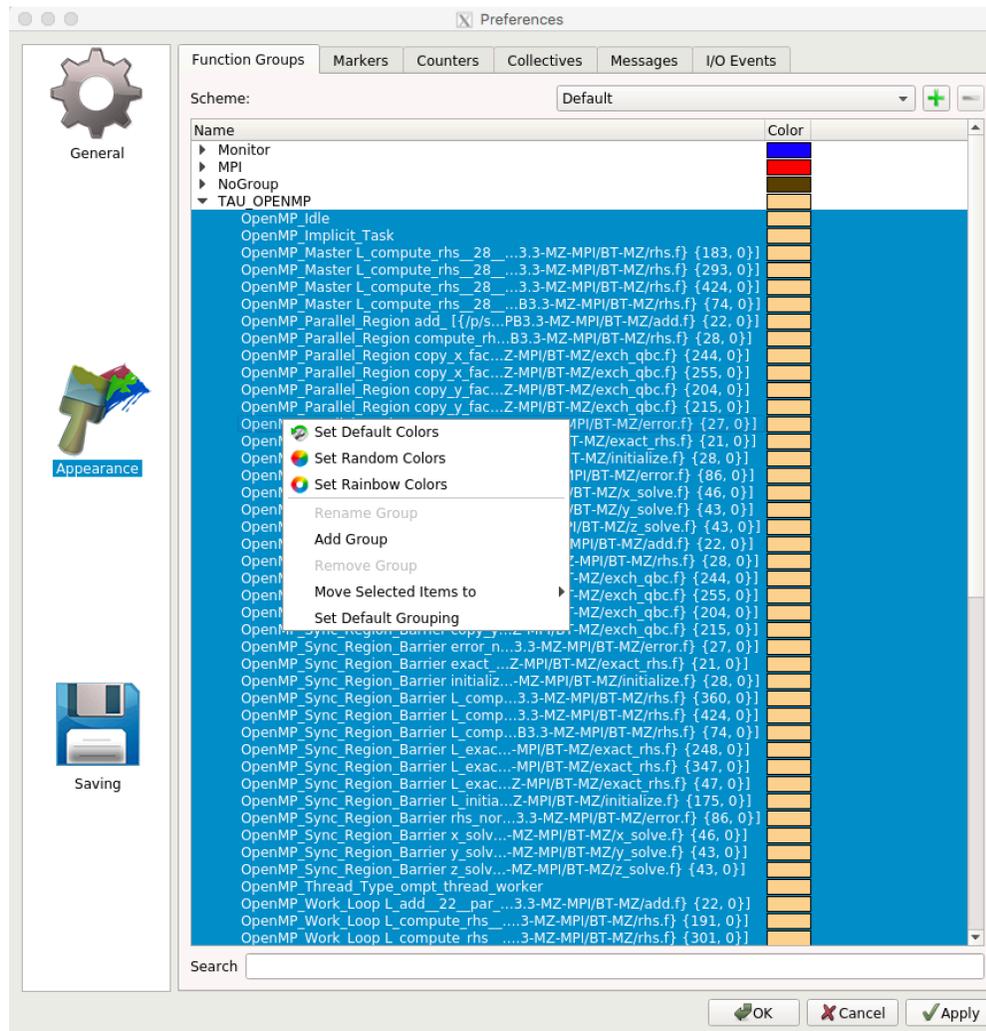
Time step 1
Time step 20
[...]
Time step 180
Time step 200
Verification Successful

BT-MZ Benchmark Completed.
Time in seconds = 18.14
% module load Vampir
% vampir traces.otf2 &
```

- Copy jobscript and launch as a hybrid MPI+OpenMP application

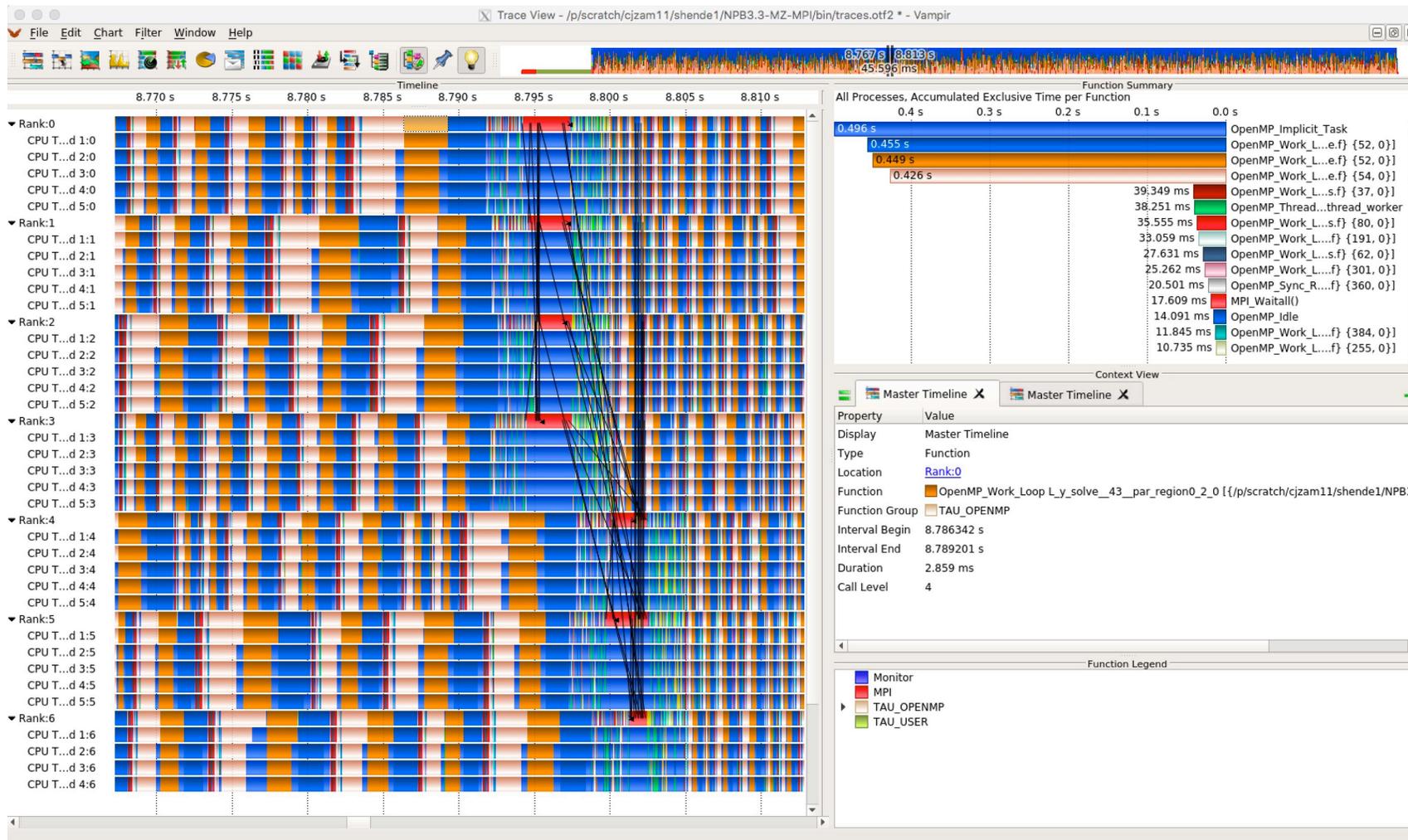
tau\_exec launches un-instrumented binaries and generates OTF2 traces for Vampir!

# Change Colors for Events in Vampir [TU Dresden]



- File -> Preferences -> Appearance
- Multi-select functions under TAU\_OPENMP
- Select “Set Random Colors”
- Click Apply, OK

# Vampir [TU Dresden] Provides a Timeline Display



TAU generates OTF2 traces natively using a replacement OMPT library – this is different from the one provided by the compiler!

```
% cat tau_otf.sbatch
export TAU_TRACE=1
export TAU_TRACE_FORMAT=otf2
```

```
...
srun tau_exec -T ompt,papi -ompt
./a.out
```

## Source Instrumentation using Opari and PDT in TAU

---

- Select appropriate compiler / MPI combination (if not done automatically)

```
% jutil env activate -p cjzam11 -A jzam11 (any of your projects)
% module load tau
```

- Copy tutorial sources to your scratch directory

```
•% tar zxvf /p/scratch/share/VI-HPS/examples/NPB3.3-MZ-MPI.tar.gz
% cd NPB3.3-MZ-MPI; make clean;
% echo $TAU_MAKEFILE
/p/scratch/share/VI-HPS/JURECA/packages/tau2.28.1-intel-cluster/x86_64/lib/
Makefile.tau-icpc-papi-mpi-pdt-openmp-opari
% make suite MPIF77=tau_f77.sh
% cd bin.tau
% cp ../jobscript/jureca/tau_opari.sbatch .
% sbatch tau_opari.sbatch
% pprof -a -m | more
% paraprof &
% paraprof --pack bt.ppk
<SCP to your laptop/desktop>
% paraprof bt.ppk &
```

# Compile-Time Options for TAU's compiler scripts (e.g., tau\_f90.sh)

---

Optional parameters for the TAU\_OPTIONS environment variable:

% tau\_compiler.sh

-optVerbose	Turn on verbose debugging messages
-optComplnst	Use compiler based instrumentation
-optNoComplnst	Do not revert to compiler instrumentation if source instrumentation fails.
-optTrackIO	Wrap POSIX I/O call and calculates vol/bw of I/O operations (configure TAU with <i>-iowrapper</i> )
-optTrackGOMP	Enable tracking GNU OpenMP runtime layer (used without <i>-opari</i> )
-optMemDbg	Enable 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=" <i>&lt;file&gt;</i> "	Specify selective instrumentation file for <i>tau_instrumentor</i>
-optTauWrapFile=" <i>&lt;file&gt;</i> "	Specify path to <i>link_options.tau</i> generated by <i>tau_gen_wrapper</i>
-optHeaderInst	Enable Instrumentation of headers
-optTrackUPCR	Track UPC runtime layer routines (used with tau_upc.sh)
-optLinking=""	Options passed to the linker. Typically <code>\$(TAU_MPI_FLIBS) \$(TAU_LIBS) \$(TAU_CXXLIBS)</code>
-optCompile=""	Options passed to the compiler. Typically <code>\$(TAU_MPI_INCLUDE) \$(TAU_INCLUDE) \$(TAU_DEFS)</code>
-optPdtF95Opts=""	Add options for Fortran parser in PDT (f95parse/gfparse) ...

## Compile-Time Options (contd.)

---

▪ Optional parameters for the TAU\_OPTIONS environment variable:

% tau\_compiler.sh

<b>-optMICOffload</b>	Links code for Intel MIC offloading, requires both host and MIC TAU libraries
<b>-optShared</b>	Use TAU's shared library (libTAU.so) instead of static library (default)
<b>-optPdtCxxOpts=""</b>	Options for C++ parser in PDT (cxxparse).
<b>-optPdtF90Parser=""</b>	Specify a different Fortran parser
<b>-optPdtCleanscapeParser</b>	Specify the Cleanscape Fortran parser instead of GNU gfparser
<b>-optTau=""</b>	Specify options to the tau_instrumentor
<b>-optTrackDMAPP</b>	Enable instrumentation of low-level DMAPP API calls on Cray
<b>-optTrackPthread</b>	Enable instrumentation of pthread calls

See tau\_compiler.sh for a full list of TAU\_OPTIONS.

...

# Compiling Fortran Codes with TAU

---

- If your Fortran code uses free format in .f files (fixed is default for .f), you may use:  
`% export TAU_OPTIONS= '-optPdtF95Opts="-R free" -optVerbose '`
- To use the compiler based instrumentation instead of PDT (source-based):  
`% export TAU_OPTIONS= '-optComplnst -optVerbose'`
- To use an instrumentation specification file:  
`% export TAU_OPTIONS= '-optTauSelectFile=select.tau -optVerbose -optPreProcess'`  
`% cat select.tau`  
`BEGIN_INSTRUMENT_SECTION`  
`loops routine="#"`  
`# this statement instruments all outer loops in all routines. # is wildcard as well as comment in first column.`  
`END_INSTRUMENT_SECTION`

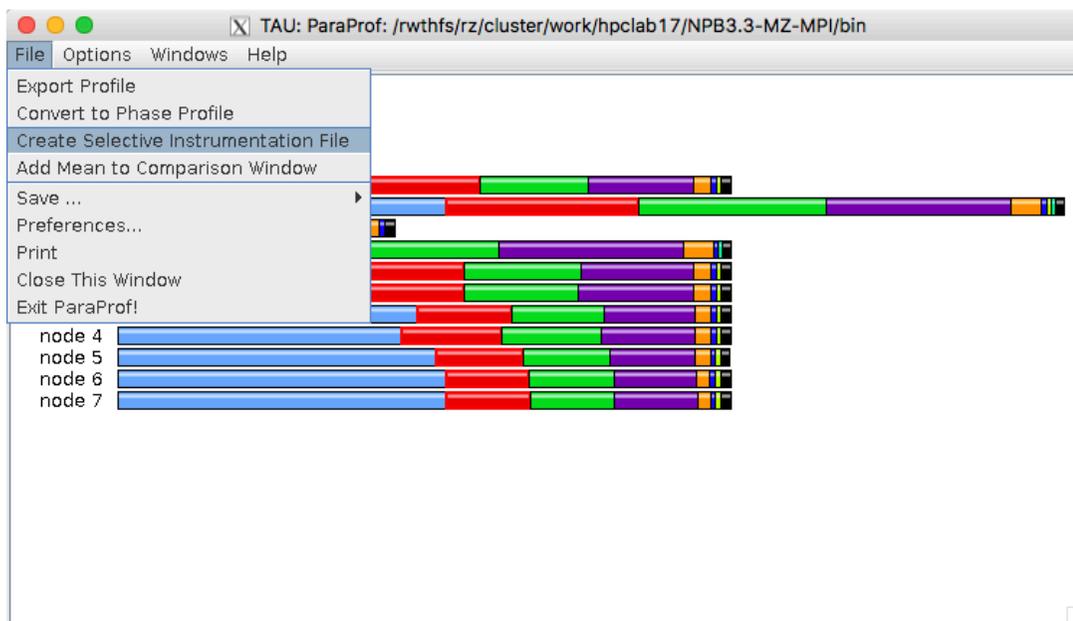
# TAU's Selective Instrumentation (Filter) File Format

---

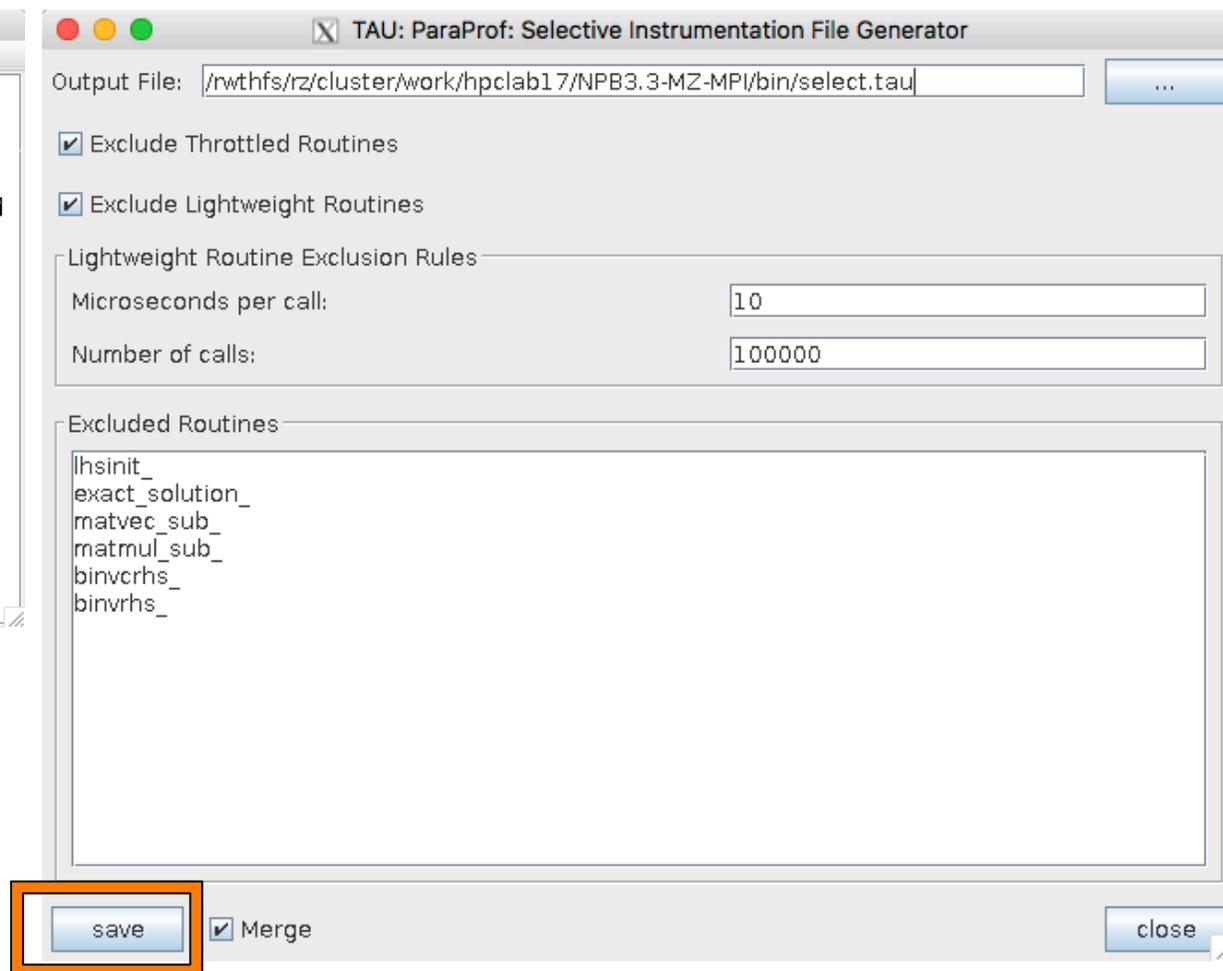
```
% export TAU_OPTIONS='-optTauSelectFile=/path/to/select.tau ...'
% cat select.tau
BEGIN_INCLUDE_LIST
int main#
int dgemm#
END_INCLUDE_LIST
BEGIN_FILE_INCLUDE_LIST
Main.c
Blas/*.f77
END_FILE_INCLUDE_LIST
# replace INCLUDE with EXCLUDE list for excluding routines/files

BEGIN_INSTRUMENT_SECTION
loops routine="foo"
loops routine="int main#"
END_INSTRUMENT_SECTION
% export TAU_SELECT_FILE=select.tau      (to use at runtime)
```

# Create a Selective Instrumentation File, Re-instrument, Re-run



% paraprof



## Re-instrument with TAU

---

- Select appropriate compiler / MPI combination (if not done automatically)

```
module use /p/scratch/share/VI-HPS/JURECA/mf
module load tau
```

- Choose path to selective instrumentation file saved by paraprof (copy if necessary):

```
% export TAU_OPTIONS='-optTauSelectFile=/full/path/to/select.tau -optVerbose'
% echo $TAU_MAKEFILE
/p/scratch/share/VI-HPS/JURECA/packages/tau2.28.1-intel-cluster/x86_64/lib/
Makefile.tau-icpc-papi-mpi-pdt-openmp-opari
% make clean
% make suite MPIF77=tau_f77.sh
% cd bin.tau
% cp ../jobscript/jureca/tau_opari.sbatch .
% sbatch tau_opari.sbatch
% pprof -a -m | more
% paraprof --pack bt.ppk
<SCP to your laptop/desktop>
% paraprof bt.ppk &
```

# TAU's 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_FOOTPRINT	0	Setting to 1 turns on tracking memory usage by sampling periodically the resident set size and high water mark of memory usage
TAU_TRACK_POWER	0	Tracks power usage by sampling periodically.
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_SAMPLING	1	Setting to 1 enables event-based sampling.
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. Throttles 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_CALLSITE	0	Setting to 1 enables callsite profiling that shows where an instrumented function was called. Also compatible with tracing.
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., ENERGY,TIME,P_VIRTUAL_TIME,PAPI_FP_INS,PAPI_NATIVE_<event>:<subevent>)

# Runtime Environment Variables

---

Environment Variable	Default	Description
TAU_TRACE	0	Setting to 1 turns on tracing
TAU_TRACE_FORMAT	Default	Setting to “otf2” turns on TAU’s native OTF2 trace generation (configure with –otf=download)
TAU_EBS_UNWIND	0	Setting to 1 turns on unwinding the callstack during sampling (use with tau_exec –ebs or TAU_SAMPLING=1)
TAU_EBS_RESOLUTION	line	Setting to “function” or “file” changes the sampling resolution to function or file level respectively.
TAU_TRACK_LOAD	0	Setting to 1 tracks system load on the node
TAU_SELECT_FILE	Default	Setting to a file name, enables selective instrumentation based on exclude/include lists specified in the file.
TAU_OMPT_SUPPORT_LEVEL	basic	Setting to “full” improves resolution of OMPT TR6 regions on threads 1.. N-1. Also, “lowoverhead” option is available.
TAU_OMPT_RESOLVE_ADDRESS_EAGERLY	0	Setting to 1 is necessary for event based sampling to resolve addresses with OMPT TR6 (-ompt=download-tr6)

# Runtime Environment Variables

Environment Variable	Default	Description
TAU_TRACK_MEMORY_LEAKS	0	Tracks allocates that were not de-allocated (needs <code>-optMemDbg</code> or <code>tau_exec -memory</code> )
TAU_EBS_SOURCE	TIME	Allows using PAPI hardware counters for periodic interrupts for EBS (e.g., <code>TAU_EBS_SOURCE=PAPI_TOT_INS</code> when <code>TAU_SAMPLING=1</code> )
TAU_EBS_PERIOD	100000	Specifies the overflow count for interrupts
TAU_MEMDBG_ALLOC_MIN/MAX	0	Byte size minimum and maximum subject to bounds checking (used with <code>TAU_MEMDBG_PROTECT_*</code> )
TAU_MEMDBG_OVERHEAD	0	Specifies the number of bytes for TAU's memory overhead for memory debugging.
TAU_MEMDBG_PROTECT_BELOW/ABOVE	0	Setting to 1 enables tracking runtime bounds checking below or above the array bounds (requires <code>-optMemDbg</code> while building or <code>tau_exec -memory</code> )
TAU_MEMDBG_ZERO_MALLOC	0	Setting to 1 enables tracking zero byte allocations as invalid memory allocations.
TAU_MEMDBG_PROTECT_FREE	0	Setting to 1 detects invalid accesses to deallocated memory that should not be referenced until it is reallocated (requires <code>-optMemDbg</code> or <code>tau_exec -memory</code> )
TAU_MEMDBG_ATTEMPT_CONTINUE	0	Setting to 1 allows TAU to record and continue execution when a memory error occurs at runtime.
TAU_MEMDBG_FILL_GAP	Undefined	Initial value for gap bytes
TAU_MEMDBG_ALINGMENT	Sizeof(int)	Byte alignment for memory allocations
TAU_EVENT_THRESHOLD	0.5	Define a threshold value (e.g., .25 is 25%) to trigger marker events for min/max

# Performance Research Lab, University of Oregon, Eugene, USA



# Support Acknowledgments

- US Department of Energy (DOE)
  - Office of Science contracts, ECP
  - SciDAC, LBL contracts
  - LLNL-LANL-SNL ASC/NNSA contract
  - Battelle, PNNL contract
  - ANL, ORNL contract
- Department of Defense (DoD)
  - PETTT, HPCMP
- National Science Foundation (NSF)
  - Glassbox, SI-2
- NASA
- CEA, France
- Partners:
  - University of Oregon
  - ParaTools, Inc., ParaTools, SAS
  - The Ohio State University
  - University of Tennessee, Knoxville
  - T.U. Dresden, GWT
  - Juelich Supercomputing Center



## Department of Defense (DoD)

- PETTT, HPCMP

## National Science Foundation (NSF)

- Glassbox, SI-2

## NASA

## CEA, France

## Partners:

- University of Oregon
- ParaTools, Inc., ParaTools, SAS
- The Ohio State University
- University of Tennessee, Knoxville
- T.U. Dresden, GWT
- Juelich Supercomputing Center



**ParaTools**



UNIVERSITY  
OF OREGON



THE OHIO STATE  
UNIVERSITY



THE UNIVERSITY OF TENNESSEE **UT**



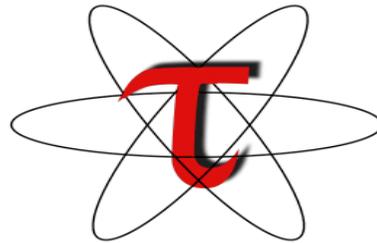


## Acknowledgement

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative.

# Download TAU from U. Oregon

---



<http://tau.uoregon.edu>

<http://www.hpclinux.com> [LiveDVD, OVA]

<https://e4s.io> [Containers for Extreme-Scale Scientific Software Stack]

**Free download, open source, BSD license**