Hands-on: BullX DLC *Froggy*
Using TAU with NPB-MZ-MPI / BT

Sameer Shende
sameer@cs.uoregon.edu
Performance Research Laboratory
University of Oregon
http://tau.uoregon.edu

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 (*Froggy* BullX DLC)

- Setup preferred program environment compilers
  - Default set Intel Compilers with Intel MPI
  - GCC+OpenMPI and Intel + BullxMPI also available

```
% source /applis/site/env.bash
% module load intel-devel
% module use /home/PROJECTS/pr-vi-hps-tw18/opt/mf
% module load tau
```

- Copy tutorial sources to your working directory, ideally on a parallel file system (scratch: /scratch/$USER)

```
% cd /scratch/$USER)
% tar zxvf /home/PROJECTS/pr-vi-hps-tw18/tutorial/NPB3.3-MZ-MPI.tar.gz
% cd NPB3.3-MZ-MPI
```

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

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

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'
Mpiifort -c  -O3 –g -openmp       bt.f
                                      […]
Mpiifort -c  -O3 –g -openmp       mpi_setup.f
cd ../common;  mpiifort -c  -O3 –g -openmp        print_results.f
cd ../common;  mpiifort -c  -O3 –g -openmp        timers.f
Mpiifort -O3 –g -openmp   -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=**8**
  - the benchmark class (S, W, A, B, C, D, E): CLASS=**C**

Shortcut: % **make suite**

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
  - 8 processes each with 4 threads should be reasonable for 2 compute nodes
  - bt-mz_C.8 should take around 40 seconds

NPB-MZ-MPI / BT Reference

```
% cd bin
% cp ../jobscript/froggy/run.oar .
% less run.oar
% oarsub –S ./run.oar
% cat test_<job_id>
 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:       32  (  4.0 threads/process)

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

 BT-MZ Benchmark Completed.
 Time in seconds = 30.14
```

- 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 Execution with TAU

```
% oarsub –S ./tau.oar
% . /applis/site/env.bash
% module load intel-devel
% module use /home/PROJECTS/pr-vi-hps-tw18/opt/mf
% module load tau
% cat test_<jobid>
 Number of zones:  16 x  16
 Iterations: 200    dt:   0.000100
 Number of active processes:     8

 Use the default load factors with threads
 Total number of threads:     32  (  4.0 threads/process)

 Calculated speedup =     31.98

 Time step    1
 Time step   20
…
 BT-MZ Benchmark Completed.
 Class           =                        C
 Size            =            480x  320x 28
 Iterations      =                     200
 Time in seconds =                   31.85
```

- Changes: Load the TAU module and use tau_exec –ompt before the name of the executable.

Compare the uninstrumented time to the TAU execution time to assess overhead.

tau_exec

```
$ tau_exec

Usage: tau_exec [options] [--] <exe> <exe options>

Options:
        -v              Verbose mode
        -s              Show what will be done but don't actually do anything (dryrun)
        -qsub           Use qsub mode (BG/P only, see below)
        -io             Track I/O
        -memory         Track memory allocation/deallocation
        -memory_debug Enable memory debugger
        -cuda           Track GPU events via CUDA
        -cupti          Track GPU events via CUPTI (Also see env. variable TAU_CUPTI_API)
        -opencl         Track GPU events via OpenCL
        -openacc        Track GPU events via OpenACC (currently PGI only)
        -ompt           Track OpenMP events via OMPT interface
        -armci          Track ARMCI events via PARMCI
        -ebs            Enable event-based sampling
        -ebs_period=<count> Sampling period (default 1000)
        -ebs_source=<counter> Counter (default itimer)
        -um           Enable Unified Memory events via CUPTI
        -T <DISABLE,GNU,ICPC,MPI,OMPT,OPENMP,PAPI,PDT,PROFILE,PTHREAD,SCOREP,SERIAL> : Specify TAU tags
        -loadlib=<file.so>   : Specify additional load library
        -XrunTAUsh-<options> : Specify TAU library directly
        -gdb            Run program in the gdb debugger

Notes:
           Defaults if unspecified: -T MPI
           MPI is assumed unless SERIAL is specified
```

- Tau_exec preloads the TAU wrapper libraries and performs measurements.

No need to recompile the application!

tau_exec Example (continued)

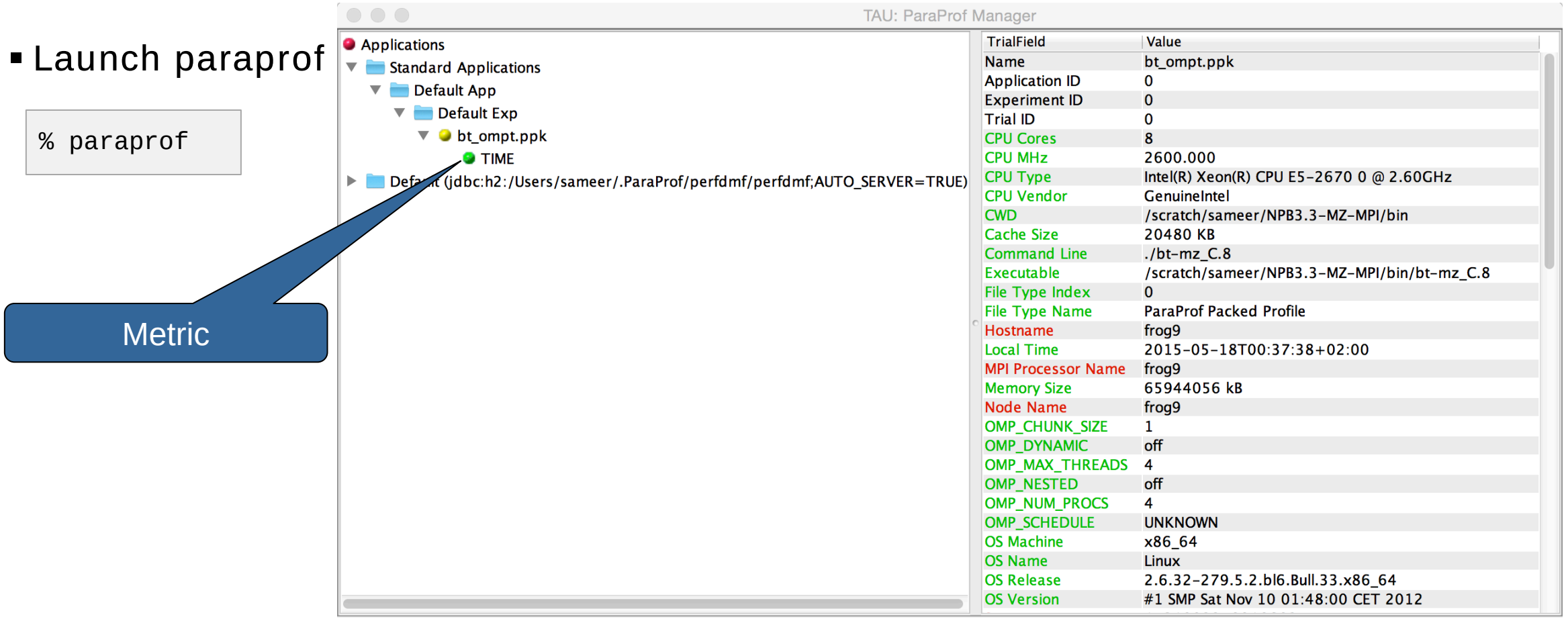
```
Example:
    mpirun –np 2 tau_exec -T icpc,ompt,mpi  -ompt ./a.out
    mpirun -np 2 tau_exec -io ./a.out
Example - event-based sampling with samples taken every 1,000,000 FP instructions
    mpirun -np 8 tau_exec -ebs -ebs_period=1000000 -ebs_source=PAPI_FP_INS ./ring
Examples - GPU:
    tau_exec -T serial,cupti -cupti ./matmult (Preferred for CUDA 4.1 or later)
    tau_exec -openacc ./a.out
    tau_exec -T serial –opencl ./a.out (OPENCL)
    mpirun –np 2 tau_exec -T mpi,cupti,papi -cupti -um ./a.out (Unified Virtual Memory in CUDA 6.0+)

qsub mode (IBM BG/Q only):
    Original:
      qsub -n 1 --mode smp -t 10 ./a.out
    With TAU:
      tau_exec -qsub -io -memory -- qsub -n 1 … -t 10 ./a.out

Memory Debugging:
    -memory option:
      Tracks heap allocation/deallocation and memory leaks.
    -memory_debug option:
      Detects memory leaks, checks for invalid alignment, and checks for
      array overflow.  This is exactly like setting TAU_TRACK_MEMORY_LEAKS=1
      and TAU_MEMDBG_PROTECT_ABOVE=1 and running with -memory
```

- tau_exec can enable event based sampling while launching the executable using the –ebs flag!

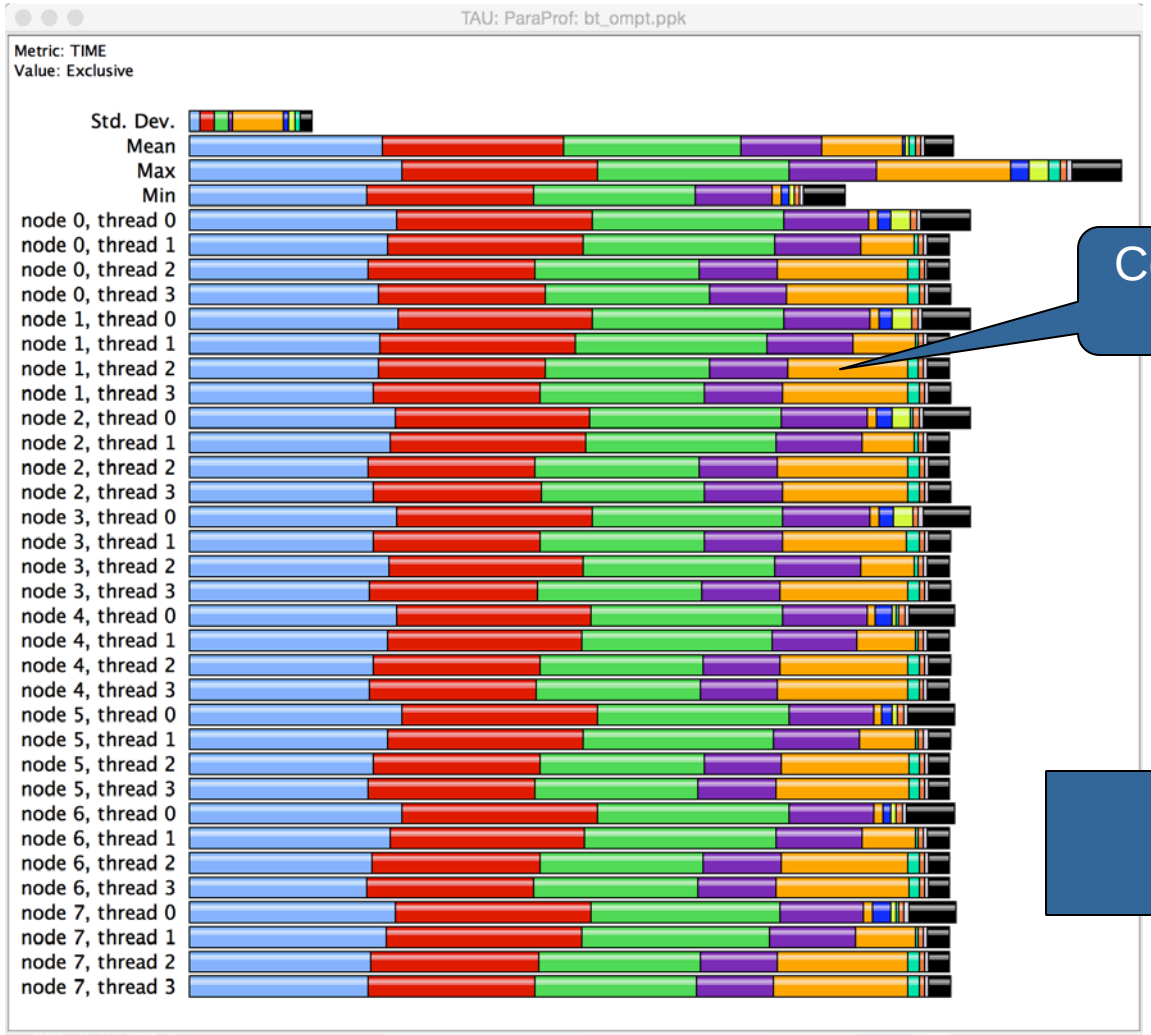
TAU Analysis Tools: paraprof

- Launch paraprof

```
% paraprof
```



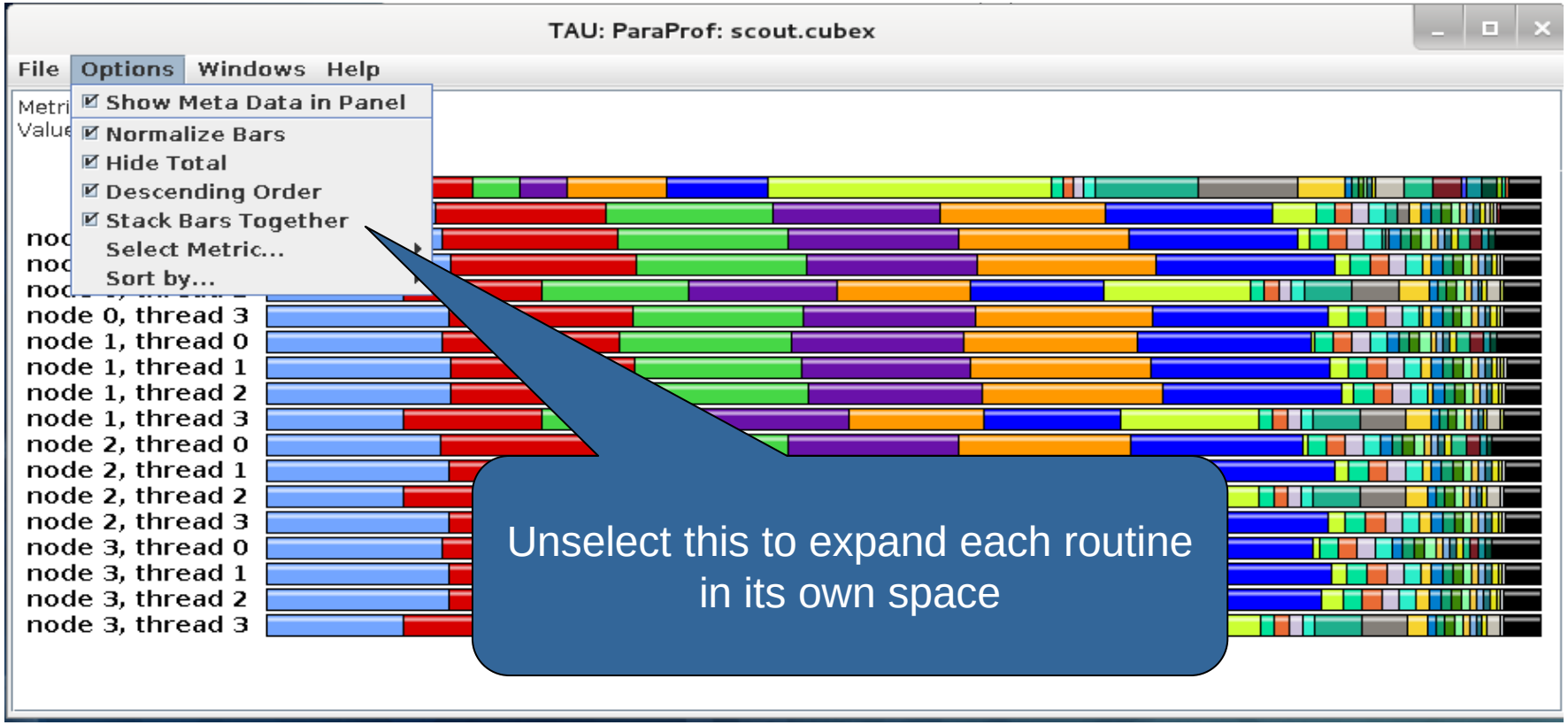
Paraprof main window



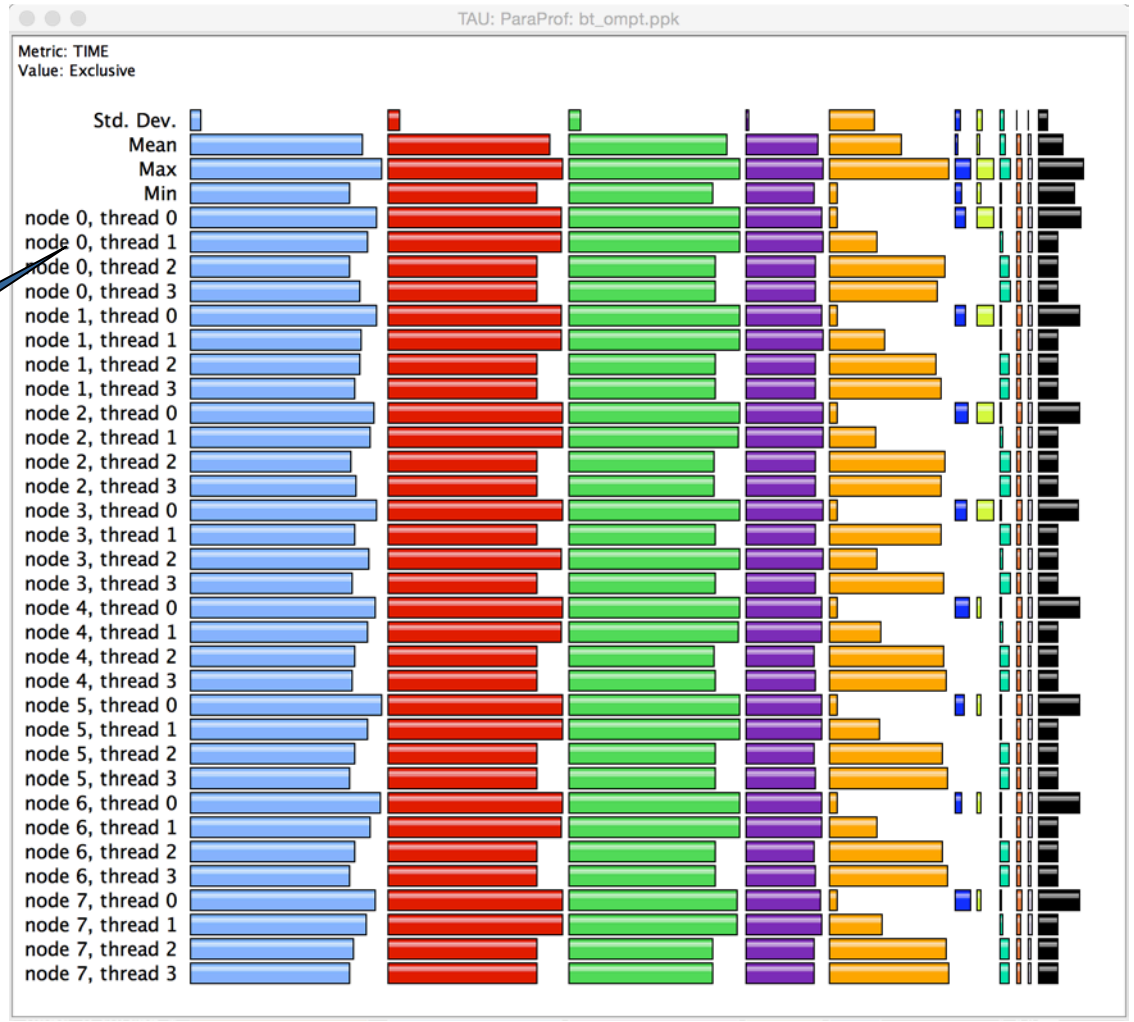
Colors represent code regions

Options -> uncheck Stack Bars Together

Paraprof main window



Paraprof main window

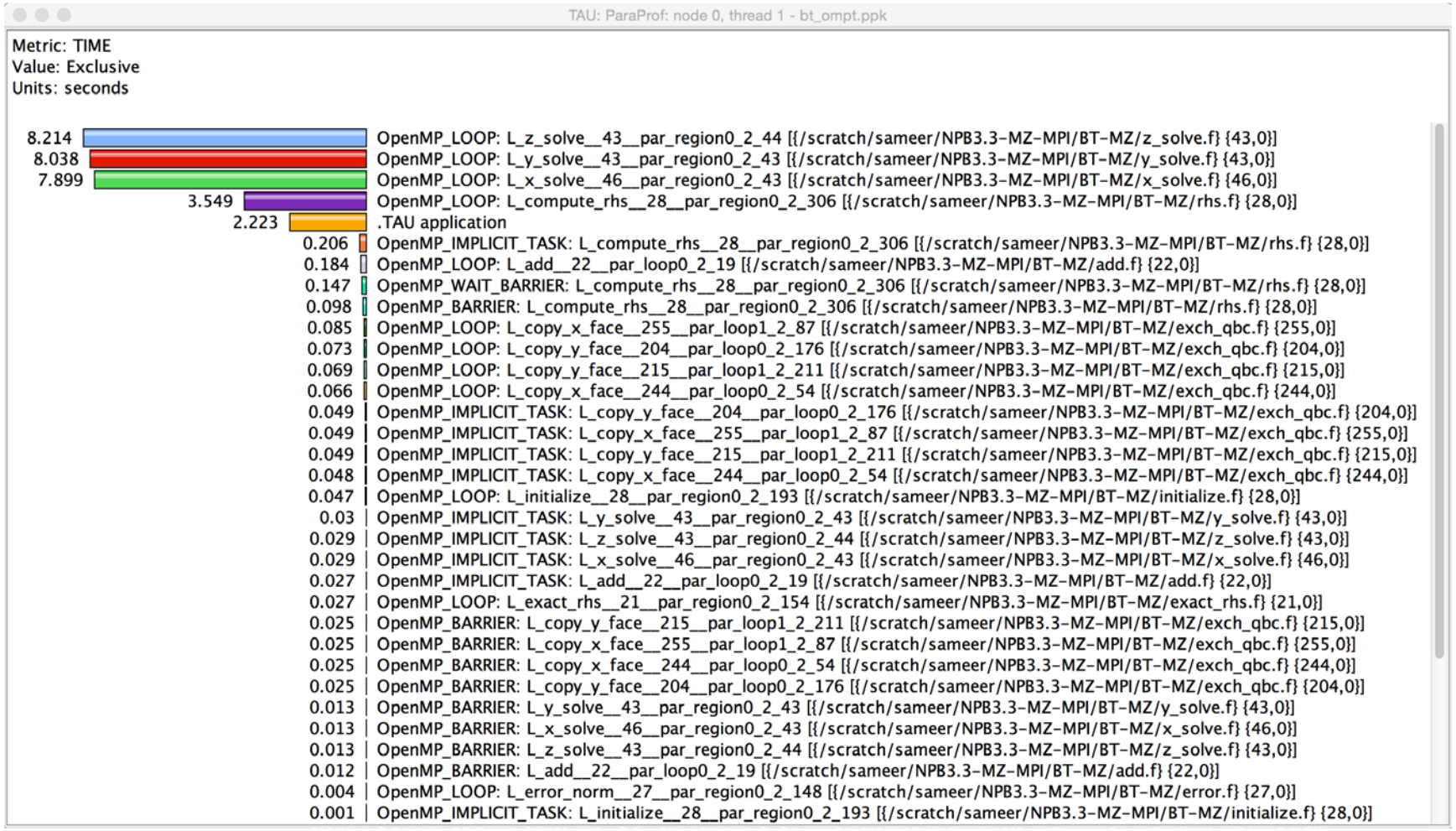


Left/right click here

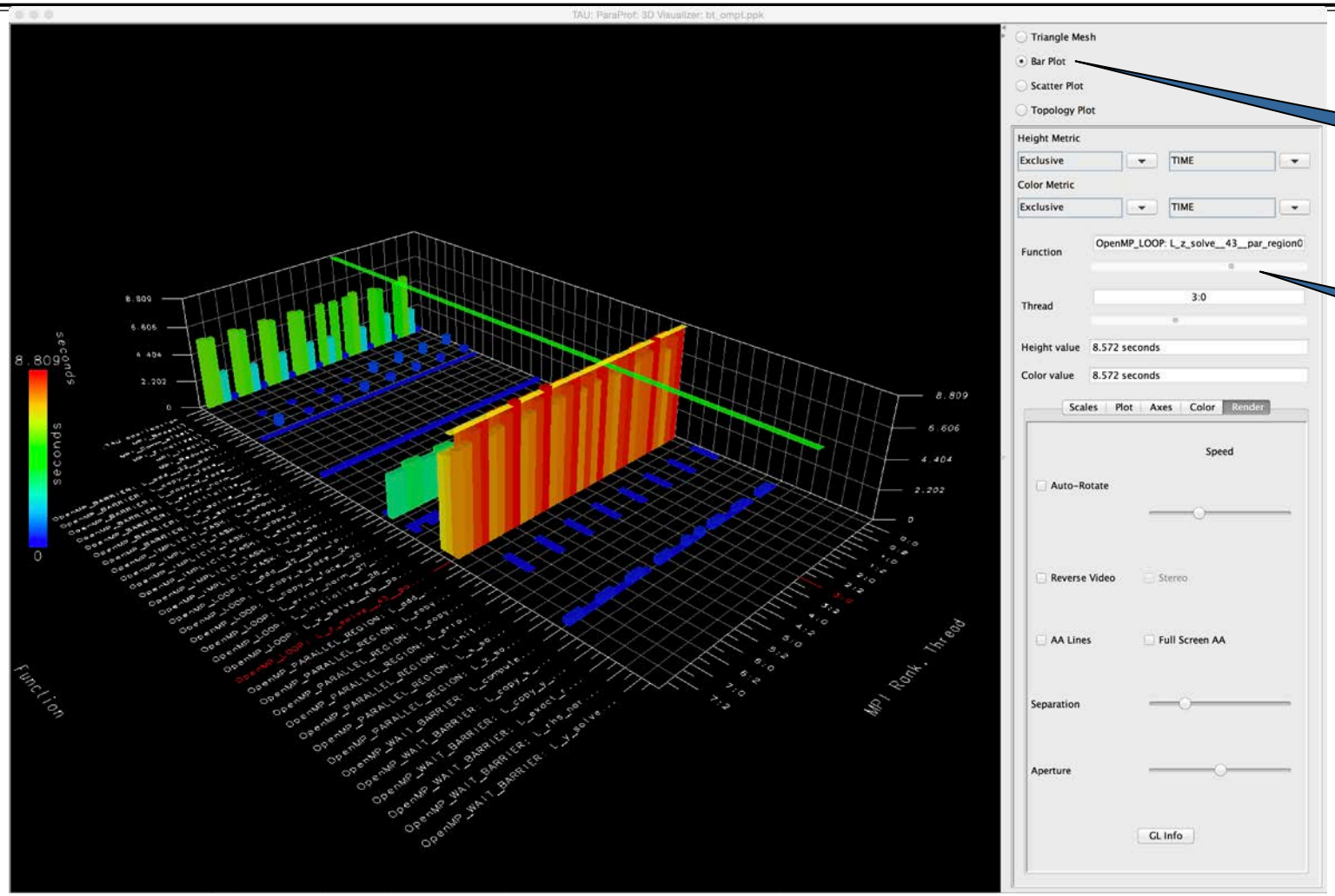
Each routine occupies its own space. Can see the extent of imbalance across all threads.

Paraprof node window (function barchart window)

Exclusive time spent in each code region (OpenMP loop) is shown here for MPI rank 0 thread 1



Paraprof 3D visualization window



Click Bar Plot

Move Function and Thread Sliders

Windows -> 3D visualization

Using Event Based Sampling (EBS) in TAU

- Edit tau.oar to uncomment line with tau_exec –ebs and comment previous line

```
% vi tau.oar
# Use tau_exec to launch the binary
mpiexec.hydra -genvall -n $NPROCS tau_exec -T mpi,ompt,pdt,papi,icpc -ompt $EXE

# Then launch it with -ebs enabled to get profiles that contain event based samples
# Please uncomment the lines below to get callpaths that contain the samples and show
# the calleer-callee relationships.
#export TAU_CALLPATH=1
#export TAU_CALLPATH_DEPTH=10
#mpiexec.hydra -genvall -n $NPROCS tau_exec -T mpi,ompt,pdt,papi,icpc -ompt -ebs $EXE
```

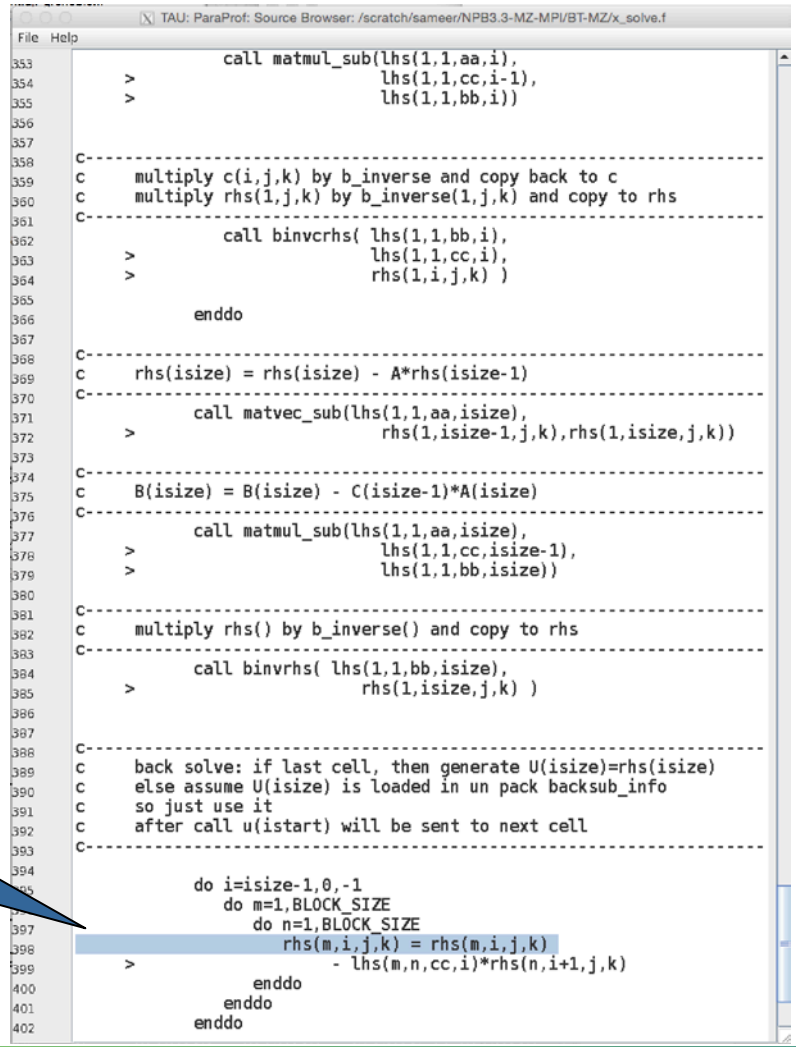
```
% cd bin
% oarsub –S tau.oar
% oastat –u $USER
% paraprof
```

Still no modification to binary or source code. No need to recompile !

Paraprof Thread Statistics Table

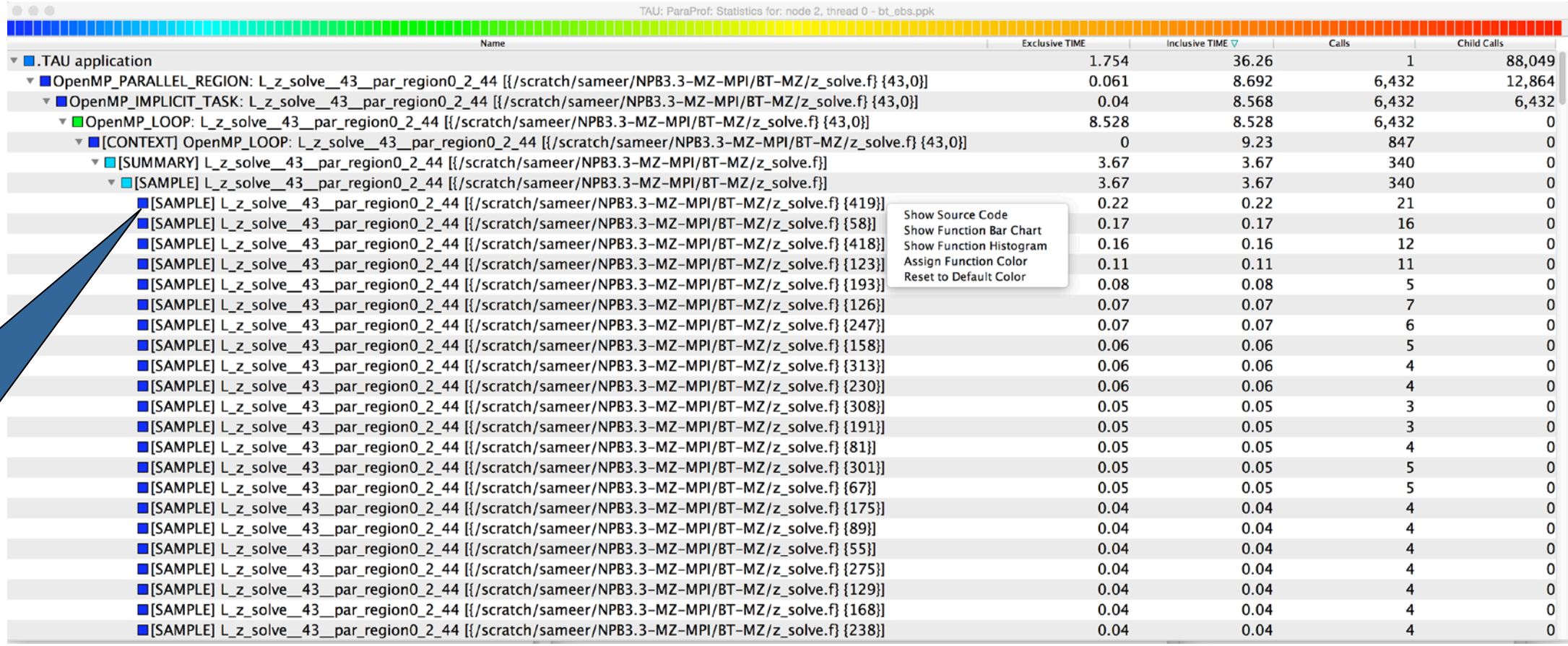
Right click here

Statement Level Profiling with TAU



Source location where samples are taken. Compute intensive region.

Paraprof Thread Statistics Table



Right click here and choose “Show Source Code” for a sample

TAU Source Instrumentation

- Edit config/make.def to adjust build configuration
  - Uncomment specification of compiler/linker: MPIF77 = tau_f90.sh
- Make clean and build new tool-specific executable

```
% make clean
% make bt-mz CLASS=C NPROCS=8
Built executable ../bin.tau/bt-mz_C.8
```

- Change to the directory containing the new executable before running it with the desired tool configuration

```
% cd bin.tau
% cp ../jobscript/froggy/run.oar  .
% oarsub –S run.oar
```

NPB-MZ-MPI / BT: config/make.def

```
#                SITE- AND/OR PLATFORM-SPECIFIC DEFINITIONS.
#
#-----------------------------------------------------------------------

#-----------------------------------------------------------------------
# Configured for generic MPI with GCC compiler
#-----------------------------------------------------------------------
#OPENMP  = -fopenmp      # GCC compiler
OPENMP = -openmp         # Intel compiler

...
#-----------------------------------------------------------------------
# The Fortran compiler used for MPI programs
#-----------------------------------------------------------------------
MPIF77 = mpiifort # Intel compiler

# Alternative variant to perform instrumentation
#MPIF77 = tau_f90.sh –tau_makefile=$(TAU)/Makefile.tau-[options]

# PREP is a generic preposition macro for instrumentation preparation
#MPIF77 = $(PREP) mpif77 –f77=ifort
...
```

Default (no instrumentation)

Uncomment TAU's compiler wrapper to do source instrumentation with TAU

Source Instrumentation with TAU

```
% make clean
% make suite
% cd bin.tau
% cp ../jobscript/froggy/run.oar .
% oarsub –S ./run.oar
% oarstat –u $USER
% paraprof
```

Instrumenting Source Code with PDT and Opari

Frequently executing lightweight routines are automatically throttled at runtime. Reduces runtime dilation.