# Performance Tools

Tulin Kaman

Department of Applied Mathematics and Statistics

Stony Brook/BNL New York Center for Computational Science

tkaman@ams.sunysb.edu

Aug 23, 2012

Do you have information on exactly where the time is being spent within your applications?

# Techniques

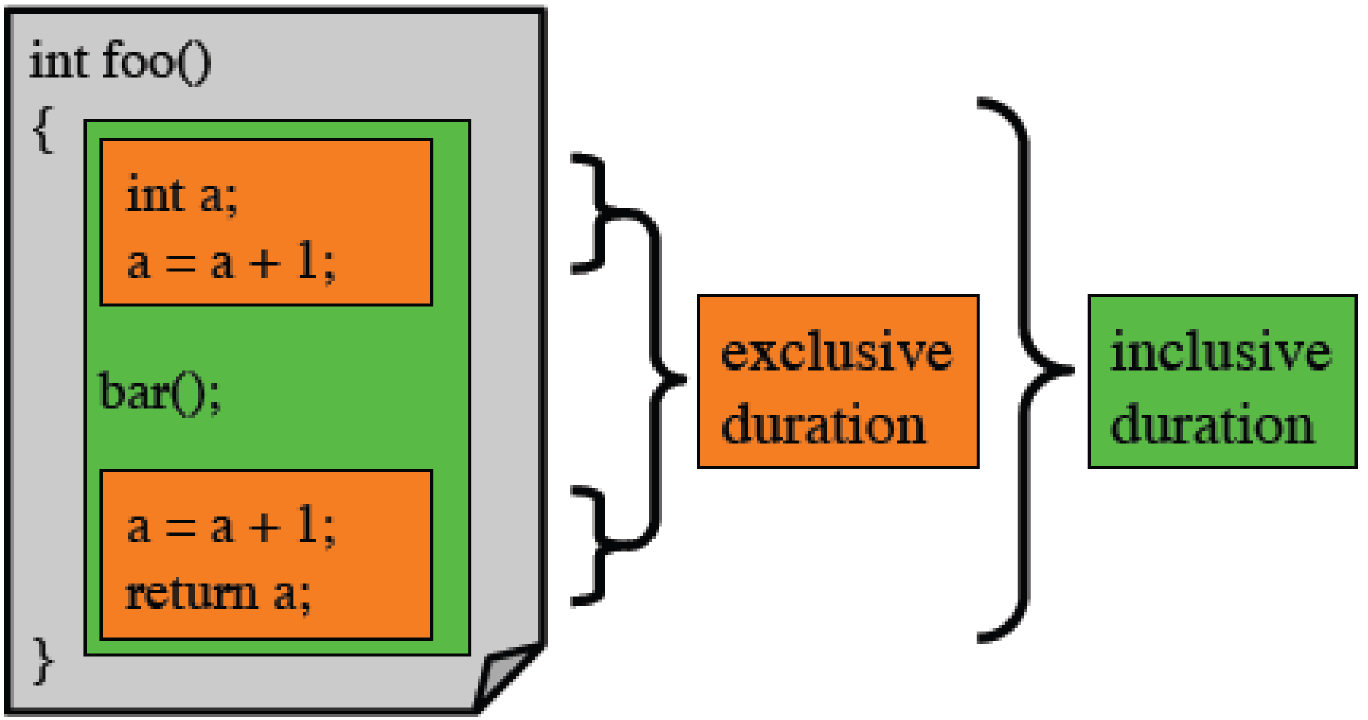
How the measurement is obtained?

- Performance Tool Mechanisms
  - Sampling (external, low overhead)
    - Regularly interrupt the program and record where it is
  - Instrumentation (internal, high overhead)
    - Code modification, insert functions into program to record and time events

- The measurements are made
  - Profiling: summarizes performance data during execution.
  - Tracing: What happens in my code at a given time?

# Inclusive and Exclusive Profiles

- Performance with respect to code regions




**Exclusive measurements**

**Inclusive measurements** includes child regions

# Performance Steps

1. Assess overall performance
2. Identify functions where most of the time being spent
3. Instrument those functions
4. Measure code performance using hardware counters
5. Identify communication bottlenecks (if Parallel)

# How to Detect Performance Problems?

- Performance: Count floating-point operation
- Each architecture has its own theoretical peak performance

| | theoretical peak performance | Clock speed |
|---|---|---|
| IBM Blue Gene L | 2.8 Gflop/s | 700 MHz |
| IBM Blue Gene P | 3.4 Gflop/s | 850 MHz |

- Parallel Performance: Scalability
  - **Strong Scalability:** Total problem size is fixed while the resources are increased.
  - **Weak Scalability:** Keep the amount of work per core the same. Increase the problem size while increasing the resources.

# Performance Tools

- Community Tools:
  - GNU Profiler: tool provided with the GNU compiler
  - Tuning and Analysis Utilities (TAU)
  - PAPI (Performance Application Programming Interface)

- High Performance Computing Toolkit (HPCT) for IBM Blue Gene
  - Message Passing Interface (MPI) Profiler and Tracer tool
  - Xprofiler for CPU profiling
  - Hardware Performance Monitoring (HPM) library
  - Modular I/O (MIO) library

# GNU Profiler

- Profiling tool provided with the GNU compiler named GNU profiler(gprof)
- Compile and link with `-g -pg`.
- Enabling profiling is as simple as adding -pg to the compile flags
- Run the application
- See files called gmon.out created on the working directory

# Flat profile

- “Flat profile”, which you obtain with gprof command

**gprof yourexe gmon.out.0 -p**

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 13.77     27.84    27.84  1705725     0.00     0.00  IDEAL_dynamic_viscosity_thermalconduct
  8.61     45.24    17.40 20187276     0.00     0.00  shortest_distance3d
  5.14     55.62    10.38                             DCMF::BGPLockManager::globalBarrierQueryDone()
  3.90     63.51     7.89 33504784     0.00     0.00  molecular_weight
  3.50     70.58     7.07 20199810     0.00     0.00  length_side
  3.44     77.53     6.95 13227156     0.00     0.00  computeCoeffs
  2.38     82.34     4.81  5377962     0.00     0.00  find_root
  2.32     87.02     4.68                             _int_malloc
  2.30     91.67     4.65                             __xl_pow
  2.26     96.23     4.56                             DCMF::Queueing::GI::giMessage::advance()
  1.84     99.95     3.72                             malloc
  1.74    103.47     3.52 17049437     0.00     0.00  array_T
  1.73    106.97     3.50 19614300     0.00     0.00  side_vector
  1.71    110.42     3.45                             cfree
  1.59    113.64     3.22                             _int_free
  1.58    116.83     3.19 32267772     0.00     0.00  ideal_Teq
  1.44    119.75     2.92                             __xl_log
  1.40    122.58     2.83                             DCMF::Queueing::GI::Device::advance()
  1.37    125.35     2.77                             _QuadCpy
  1.37    128.12     2.77                             __va_arg
  1.26    130.66     2.54                             DCMF::DMA::Device::advance()
  1.15    132.99     2.33                             memset
  1.09    135.19     2.20                             __read_nocancel
  1.08    137.37     2.18 17057250     0.00     0.00  C_P_species
  0.99    139.38     2.01  2978352     0.00     0.00  ComputeBackwardRateCoefficients
  0.98    141.37     1.99 17899096     0.00     0.00  IDEAL_temperature
```

**% time:** the percentage of the total running time of the program used by this function.

**cumulative seconds:** a running sum of the number of seconds accounted for by this function and those listed above it.

**self seconds:** the number of seconds accounted for by this function alone.

# Call graph

## gprof yourexe gmon.out.0 -q

```
                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.00% of 202.11 seconds

index % time    self  children    called     name
-----------------------------------------------
                0.00  128.30       1/1           main [1]
[3]     63.5    0.00  128.30       1         dmain [3]
                0.00  127.47       1/1           main_time_step_loop [4]
                0.00    0.84       1/1           perform_initialization [141]
                0.00    0.00       1/1           end_of_run [962]
-----------------------------------------------
                0.00  127.47       1/1           dmain [3]
[4]     63.1    0.00  127.47       1         main_time_step_loop [4]
                0.00  127.03       3/3           time_step [5]
                0.00    0.41       3/3           d_printout [205]
                0.01    0.00       1/1           Set_up_merge_cell [557]
                0.00    0.01       9/39          h_scatter_states [406]
                0.00    0.00       3/4           delete_untracked_hyper_surfaces [661]
                0.00    0.00       2/4           find_time_step [802]
                0.00    0.00       7/243         start_clock [790]
                0.00    0.00       6/242         stop_clock [791]
                0.00    0.00       5/81895269     debugging [160]
                0.00    0.00       2/66          print_storage [825]
                0.00    0.00       2/13069938     debug_print [267]
                0.00    0.00       3/3           g_set_gravity_charts [1212]
                0.00    0.00       3/16          set_iperm [1139]
                0.00    0.00       3/18          add_time_clear [1130]
                0.00    0.00       3/28          add_time_start [1114]
                0.00    0.00       3/28          add_time_end [1113]
                0.00    0.00       3/8           d_stop_run [1164]
                0.00    0.00       1/5           add_to_debug [1183]
-----------------------------------------------
                0.00  127.03       3/3           main_time_step_loop [4]
[5]     62.9    0.00  127.03       3         time_step [5]
                0.00   52.10       3/3           hyp_vector_driver [6]
                0.00   42.77       3/3           parab_driver [8]
                0.02   32.07       3/3           Update_merge_cell [14]
                0.07    0.00       3/3           turbulent_inflow [352]
                0.00    0.00       3/3           average_p_if_mach [864]
                0.00    0.00       6/66          print_storage [825]
                0.00    0.00       9/243         start_clock [790]
                0.00    0.00       9/242         stop_clock [791]
                0.00    0.00       3/81895269     debugging [160]
                0.00    0.00       3/4           set_current_chart [1206]
                0.00    0.00       3/3           no_bc_propagate [1216]
-----------------------------------------------
```

```
                0.02   32.07       3/3           time_step [5]
[14]    15.9    0.02   32.07       3         Update_merge_cell [14]
                1.30   30.77    3105/3105        oned_turbulence_boundarylayer_solver [15]
                0.00    0.00   14175/5897677     vel [230]
                0.00    0.00     540/14151       g_alloc_state [600]
                0.00    0.00     540/1197743     g_set_params [436]
                0.00    0.00     648/3101728     g_composition_type [359]
-----------------------------------------------
                1.30   30.77    3105/3105        Update_merge_cell [14]
[15]    15.9    1.30   30.77    3105         oned_turbulence_boundarylayer_solver [15]
               27.84    2.88 1705725/1705725     IDEAL_dynamic_viscosity_thermalconduct [16]
                0.00    0.02    3105/92700       IDEAL_C_P [172]
                0.01    0.00   37260/17049437     array_T [64]
                0.00    0.01    3105/10253627     IDEAL_temperature <cycle 1> [28]
                0.00    0.00    6210/6317        free_these [603]
                0.00    0.00    6210/13069938     debug_print [267]
```

# Annotated source listing

- prints out the source code to the application, with notes on how many times each function is called.

**gprof yourexe gmon.out.0 -A**

```
*** File /gpfs/home/kaman/scramjet_new/FronTier_081412/FronTier/src/gas/ghyp/ghypvec.c:
        /*******************************************************************************
        FronTier is a set of libraries that implements different types of Front Traking algorithms.
        Front Tracking is a numerical method for the solution of partial differential equations
        whose solutions have discontinuities.


        Copyright (C) 1999 by The University at Stony Brook.


        This library is free software; you can redistribute it and/or
        modify it under the terms of the GNU Lesser General Public
        License as published by the Free Software Foundation; either
        version 2.1 of the License, or (at your option) any later version.

        This library is distributed in the hope that it will be useful,
        but WITHOUT ANY WARRANTY; without even the implied warranty of
        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
        Lesser General Public License for more details.

        You should have received a copy of the GNU Lesser General Public
        License along with this library; if not, write to the Free Software
        Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

        ******************************************************************************/


        /*
        *                       ghypvec.c
        *
        *       Copyright 1999 by The University at Stony Brook, All rights reserved.
        *
        *       Contains the drivers for the uni_arrayized hyperbolic scheme:
        */


        #include <ghyp/ghyp.h>
        #include <gdecs/vecdecs.h>
```

| Line | Count |
|---|---|
| 48 | 8760 |
| 147 | 4833 |
| 119 | 443 |

```
48                              /*ARGSUSED*/
49           8760 -> EXPORT  void vector_FD(
50                                   int          swp_num,
51                                   int          *iperm,
52                                   float        *dir,
53                                   Wave         *wv,
54                                   Wave         *newwv,
55                                   Front        *fr,
56                                   Front        *newfr,
57                                   int          *icoords,
58                                   int          imin,
59                                   int          imax,
60                                   float        dt,
61                                   float        dh)
62                              {
```

# Tuning and Analysis Utilities: TAU

**TAU team:**

**Sameer Shende**

Allen D. Malony, Wyatt Spear, Scott Biersdorff, Suzanne Millstein

**University of Oregon**

**http://tau.uoregon.edu**

# Tuning and Analysis Utilities - TAU

- Performance evaluation tool
- Profiling and tracing toolkit for performance analysis of parallel programs written in C, C++, Fortran, Java and Python
- Support for multiple parallel programming paradigms: MPI, Multi-threading, Hybrid (MPI + Threads)
- Access to hardware counters using PAPI

# TAU Configuration

- Each configuration labeled with the options used.

./configure -mpi -arch=bgl -pdt=<pdt-dir> -pdt=xlC

-PROFILE(default) /-PROFILECALLPATH/-MPITRACE/…

- Each configuration creates a unique Makefile.
  - <tau-dir>/bgl/lib for BG/L platform
  - <tau-dir>/bgp/lib for BG/P platform
- Add the bin directory to your path.

```
export PATH=/bgl/apps/TAUL/tau-2.18/bgl/bin:$PATH
export PATH=/bgl/apps/TAUL/tau-2.18/bgl/bin:$PATH
```

# List of TAU's Makefile on BG/L

Makefile.tau-bgltimers-multiplecounters-mpi-papi-compensate-pdt

Makefile.tau-bgltimers-multiplecounters-mpi-papi-pdt

Makefile.tau-callpath-mpi-compensate-pdt

Makefile.tau-callpath-mpi-pdt

Makefile.tau-depthlimit-mpi-pdt

Makefile.tau-mpi-compensate-pdt

Makefile.tau-mpi-papi-pdt

**Makefile.tau-mpi-pdt**

Makefile.tau-mpi-pdt-trace

Makefile.tau-multiplecounters-mpi-papi-pdt

Makefile.tau-multiplecounters-mpi-papi-pdt-trace

Makefile.tau-pdt

Makefile.tau-phase-multiplecounters-mpi-papi-compensate-pdt

Makefile.tau-phase-multiplecounters-mpi-papi-pdt

> Program Database Toolkit (PDT) provides access to the high-level interface of source code for analysis tools and applications.

# TAU Instrumentations

Three methods to track the performance of your application

1. Dynamic instrumentation
2. Compiler based instrumentation
3. Source instrumentation

# Dynamic instrumentation through library preloading

- Options: tracking MPI, io, memory, cuda, opencl library calls.
- Default: MPI instrumentation
- Others are enabled by command-line options to `tau_exec`

Example: IO instrumentation is requested.

```
$ tau_exec -io ./a.out
$ mpirun -np 4 tau_exec -io ./a.out
```

# Compiler Based Instrumentation

- Set environment variables
- Use TAU_MAKEFILE
- Use TAU compiler scripts: tau_cxx.sh, tau_cc.sh, tau_f90.sh
- Set TAU options available to TAU compiler scripts

| | |
|---|---|
| -optVerbose | Enable verbose output (default: on) |
| -optKeepFiles | Do not remove intermediate files |
| -optShared | Use shared library of TAU (consider when using tau_exec |

**Example:**

```
$ export PATH =[path to tau]/[arch]/bin:$PATH
$ export TAU_MAKEFILE=[path to tau]/[arch]/lib/[makefile]
$ tau_cc.sh -o hello hello.c
```

# Running the Application

- Run the application to generate the profile data files
- Profile data files are generated in the current directory. (DEFAULT)
- The environment variables:
  - PROFILEDIR to store the files in different directory.
  - TAU_VERBOSE to see the steps the TAU measurement systems takes when your application is running
  - TAU_TRACK_MESSAGE to track MPI message statistics

On Blue Gene: In your batch job script file, set the environment variable

```
# @ arguments = -np 16 -env PROFILEDIR=<profile-dir> -exe …
```

# Reducing Performance Overhead with TAU_THROTTLE

- Default rule TAU uses to determine which functions to throttle:

  TAU_THROTTLE_NUMCALLS 100000 (DEFAULT)
  TAU_THROTTLE_PERCALL 10 (DEFAULT)

  Profiling of the function is disabled if the number of function call is more than 100000 times and has an inclusive time per call of less than 10 microseconds.

```
export TAU_THROTTLE_NUMCALLS=2000000
export TAU_THROTTLE_PERCALL=5
```

# Profiling each event callpath

- Make sure you set the TAU_MAKEFILE

`[path to tau]/[arch]/lib/ Makefile.tau-callpath-mpi-pdt`

- Set the environment variable TAU_CALLPATH
- Each event callpath to the depth set by the environment variable TAU_CALLPATH_DEPTH environment variable (default is two)
  - Higher depth introduces more overhead

`export TAU_CALLPATH=1` (enables callpath)

`export TAU_CALLPATH_DEPTH=100` (defines depth)

# Performance Counters

- Peformance counters can count hardware performance events such as cache misses, floating point operations
- PAPI: Performance Data Standard and API package provides a uniform interface to access these performance counters.
- TAU uses PAPI
- Find out which PAPI events are supported in your system.
- Run `papi_avail`

# Performance Counters on BG/L microprocessor (PowerPC440)

HardwareInfo_AvailableEvents.txt (~/TAUL) – VIM — ssh — 110x35

```
Test case 8: Available events and hardware information.
-------------------------------------------------------------------------
Vendor string and code   :  (1312)
Model string and code    : PVR=0x5202:0x1891  Serial=R00-M0-N1-C:J16-U01 (1375869073)
CPU Revision             : 20994.062500
CPU Megahertz            : 700.000000
CPU's in this Node       : 1
Nodes in this System     : 16
Total CPU's              : 16
Number Hardware Counters : 52
Max Multiplex Counters   : 32
-------------------------------------------------------------------------
Name            Derived Description (Mgr. Note)
PAPI_L3_TCM     No      Level 3 cache misses (BGL_UPC_L3_CACHE_MISS_DATA_WILL_BE_REQED_DDR)
PAPI_L3_LDM     Yes     Level 3 load misses (BGL_UPC_L3_MSHNDLR_TOOK_REQ_RDQ0, BGL_UPC_L3_MSHNDLR_TOOK_REQ_RDQ
1)
PAPI_L3_STM     No      Level 3 store misses (BGL_UPC_L3_MSHNDLR_TOOK_REQ_WRBUF)
PAPI_FMA_INS    No      FMA instructions completed (BGL_FPU_ARITH_TRINARY_OP)
PAPI_TOT_CYC    No      Total cycles (Timebase register (null))
PAPI_L2_DCH     Yes     Level 2 data cache hits (BGL_UPC_PU0_PREF_STREAM_HIT, BGL_UPC_PU1_PREF_STREAM_HIT)
PAPI_L2_DCA     Yes     Level 2 data cache accesses (BGL_UPC_PU0_PREF_REQ_VALID, BGL_UPC_PU1_PREF_REQ_VALID)
PAPI_L3_TCH     No      Level 3 total cache hits (BGL_UPC_L3_CACHE_HIT)
PAPI_FML_INS    No      Floating point multiply instructions (BGL_FPU_ARITH_MULT_DIV)
PAPI_FAD_INS    No      Floating point add instructions (BGL_FPU_ARITH_ADD_SUBTRACT)
PAPI_BGL_OED    No      Floating point Oedipus operations (BGL_FPU_ARITH_OEDIPUS_OP)
PAPI_BGL_TS_32B Yes     32B chunks sent in any torus link (BGL_UPC_TS_XM_32B_CHUNKS, BGL_UPC_TS_XP_32B_CHUNKS,
 BGL_UPC_TS_YM_32B_CHUNKS, BGL_UPC_TS_YP_32B_CHUNKS, B...)
PAPI_BGL_TS_FULL        Yes     CLOCKx2 cycles with no torus token (accum) (BGL_UPC_TS_XM_LINK_AVAIL_NO_VCD0_V
CD_VCBN_TOKENS, BGL_UPC_TS_XP_LINK_AVAIL_NO_VCD0_VCD_VCBN_TOKENS, BGL_U...)
PAPI_BGL_TR_DPKT        Yes     Data packets sent on any tree channel (BGL_UPC_TR_SNDR_2_VC1_DPKTS_SENT, BGL_U
PC_TR_SNDR_2_VC0_DPKTS_SENT, BGL_UPC_TR_SNDR_1_VC1_DPKTS_SENT, BGL...)
PAPI_BGL_TR_FULL        Yes     CLOCKx2 cycles with tree receiver full (accum) (BGL_UPC_TR_RCV_0_VC0_FULL, BGL
_UPC_TR_RCV_0_VC1_FULL, BGL_UPC_TR_RCV_1_VC0_FULL, BGL_UPC_TR_RCV_1_VC1_FUL...)
HardwareInfo_AvailableEvents.txt                                                    1,1          All
:0
```

# Performance Counters on BG/P microprocessor (PowerPC450)

Thanks for flying Vim — ssh — 80x34

```
    PAPI_L1_DCM_idx = 0, /*Level 1 data cache misses */
    PAPI_L1_ICM_idx,  /*Level 1 instruction cache misses */
    PAPI_L2_DCM_idx,  /*Level 2 data cache misses */
    PAPI_L2_ICM_idx,  /*Level 2 instruction cache misses */
    PAPI_L3_DCM_idx,  /*Level 3 data cache misses */
    PAPI_L3_ICM_idx,  /*Level 3 instruction cache misses */
    PAPI_L1_TCM_idx,  /*Level 1 total cache misses */
    PAPI_L2_TCM_idx,  /*Level 2 total cache misses */
    PAPI_L3_TCM_idx,  /*Level 3 total cache misses */
    PAPI_CA_SNP_idx,  /*Snoops */
    PAPI_CA_SHR_idx,  /*Request for shared cache line (SMP) */
    PAPI_CA_CLN_idx,  /*Request for clean cache line (SMP) */
    PAPI_CA_INV_idx,  /*Request for cache line Invalidation (SMP) */
    PAPI_CA_ITV_idx,  /*Request for cache line Intervention (SMP) */
    PAPI_L3_LDM_idx,  /*Level 3 load misses */
    PAPI_L3_STM_idx,  /*Level 3 store misses */
/* 0x10 */
    PAPI_BRU_IDL_idx, /*Cycles branch units are idle */
    PAPI_FXU_IDL_idx, /*Cycles integer units are idle */
    PAPI_FPU_IDL_idx, /*Cycles floating point units are idle */
    PAPI_LSU_IDL_idx, /*Cycles load/store units are idle */
    PAPI_TLB_DM_idx,  /*Data translation lookaside buffer misses */
    PAPI_TLB_IM_idx,  /*Instr translation lookaside buffer misses */
    PAPI_TLB_TL_idx,  /*Total translation lookaside buffer misses */
    PAPI_L1_LDM_idx,  /*Level 1 load misses */
    PAPI_L1_STM_idx,  /*Level 1 store misses */
    PAPI_L2_LDM_idx,  /*Level 2 load misses */
    PAPI_L2_STM_idx,  /*Level 2 store misses */
    PAPI_BTAC_M_idx,  /*BTAC miss */
    PAPI_PRF_DM_idx,  /*Prefetch data instruction caused a miss */
    PAPI_L3_DCH_idx,  /*Level 3 Data Cache Hit */
    PAPI_TLB_SD_idx,  /*Xlation lookaside buffer shootdowns (SMP) */
    PAPI_CSR_FAL_idx, /*Failed store conditional instructions */
lines 54-86
```

# To Generate Hardware Counter Profile

- Make sure you set the TAU_MAKEFILE for hardware counter profiling.

TAU_MAKEFILE=[path to tau]/[arch]/lib/ Makefile.tau-multiplecounters-mpi-papi-pdt

- Set the COUNTERx environment variables to specify the type of counter to profile in your job script file

**# @ arguments = -np 1 -env PROFILEDIR=<profile-dir>**
**-env "COUNTER1=GET_TIME_OF_DAY COUNTER2= PAPI_L1_DCM **
**COUNTER3=PAPI_L1_ICM COUNTER4=PAPI_L1_TCM" -exe …**

- Following subdirectories will be created

<profile-dir>/MULTI__GET_TIME_OF_DAY

<profile-dir>/MULTI__PAPI_L1_DCM

<profile-dir>/MULTI__PAPI_L1_ICM

<profile-dir>/MULTI__PAPI_L1_TCM

# Fast Blue Gene Timers

- Blue Gene systems have a special clock cycle counter that can be used for low overhead timings

| | |
|---|---|
| -BGLTIMERS | Use fast low-overhead timers on IBM BG/L |
| -BGPTIMERS | Use fast low-overhead timers on IBM BG/P |
| -LINUXTIMERS | Use low overhead TSC Counter for wallclock time. |
| -CPUTIME | Use usertime+system time instead of wallclock time. |
| -PAPIWALLCLOCK | Use PAPI to access wallclock time. |

# Analyzing Parallel Application

- **pprof ( for text based display )**
  - sorts and displays profile data generated by TAU.
  - Execute pprof in the directory where profile files are located.

- **paraprof ( for GUI display)**
  - TAU has Java based performance data viewer.
  - Requires Java1.4 or above, add it to your path.
  - Pack all the profile files into one file. Easy to copy one file to local computer.

```
$ paraprof --pack filename.ppk
```

  - To launch the GUI

```
$ paraprof filename.ppk
```

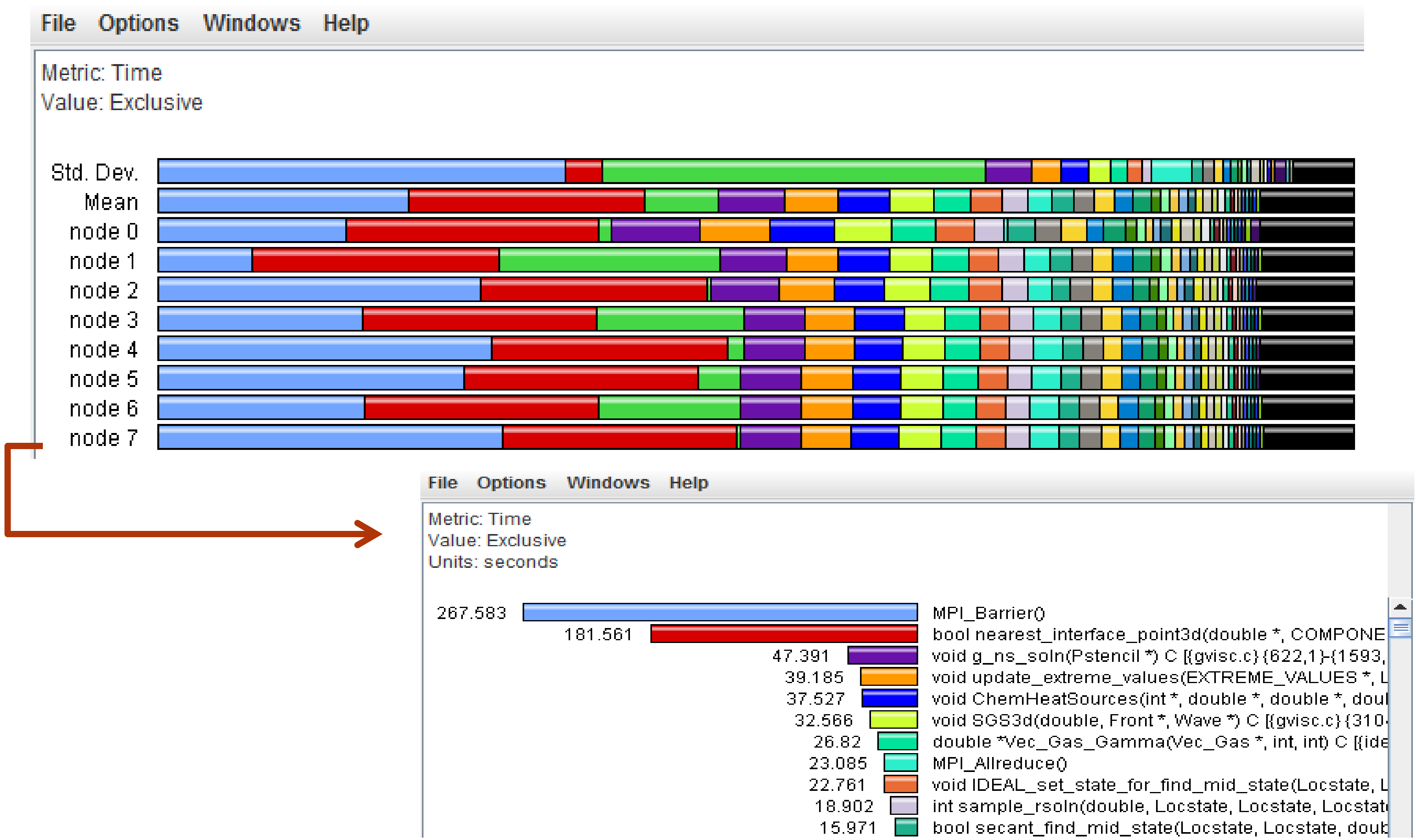
# pprof (Text based display)

Thanks for flying Vim — ssh — 124x42

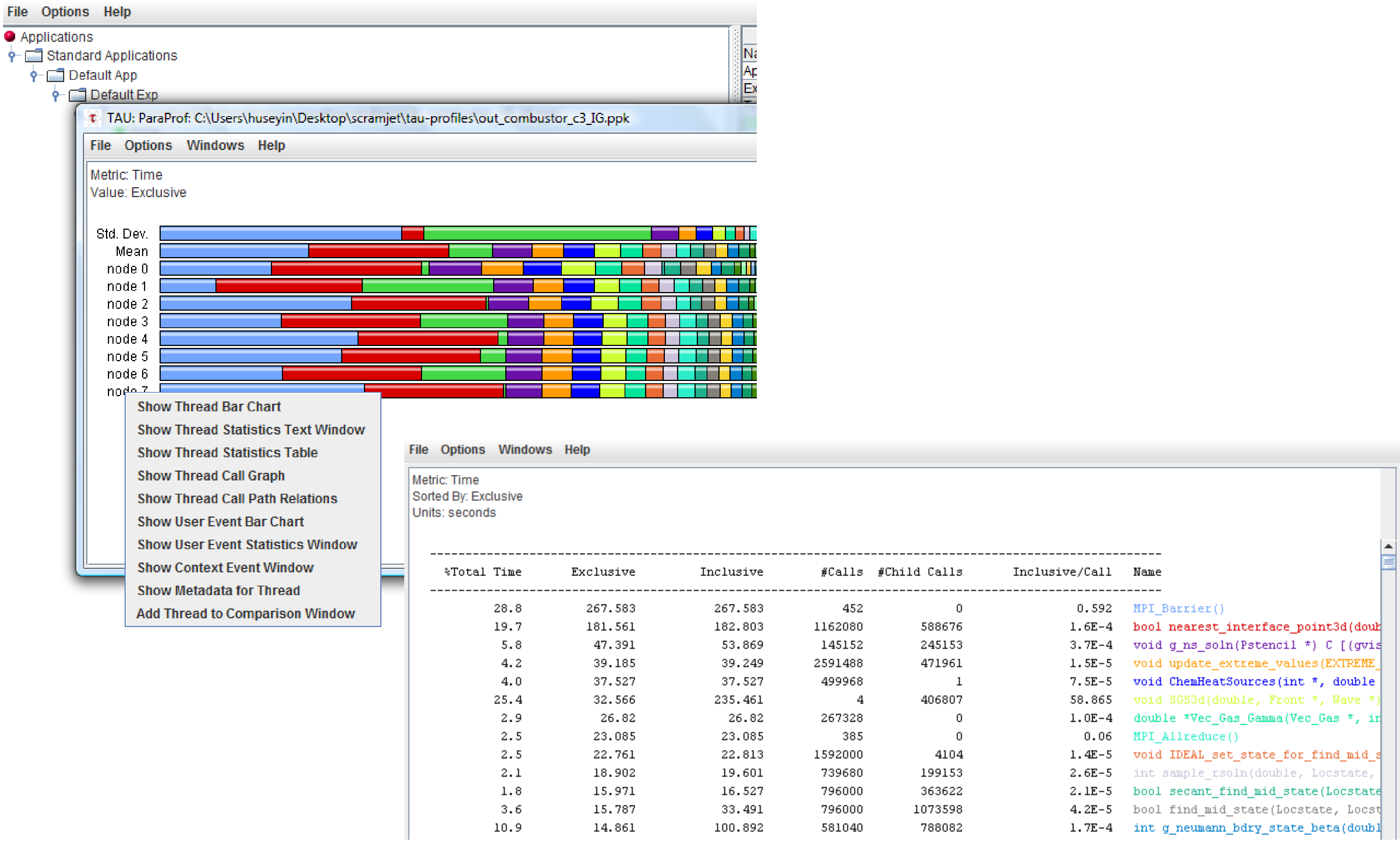
```
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0        0.081    21:16.904           1           2 1276904738 int main(int, char **) C
100.0        0.083    21:16.904           1           1 1276904059 int dmain(int, char **, INIT_DATA *, INIT_PHYSICS *) C
100.0        0.083    21:16.904           1           1 1276904059 int main(int, char **) C => int dmain(int, char **, INIT_
DATA *, INIT_PHYSICS *) C
100.0        0.125    21:16.903           1           3 1276903976 int dmain(int, char **, INIT_DATA *, INIT_PHYSICS *) C =>
 void perform_initialization(int, char **, INIT_DATA *, INIT_PHYSICS *) C
100.0        0.125    21:16.903           1           3 1276903976 void perform_initialization(int, char **, INIT_DATA *, IN
IT_PHYSICS *) C
100.0        0.071    21:16.903           1           1 1276903684 void g_init_run(int *, char ***, INIT_DATA *, INIT_PHYSIC
S *) C
100.0        0.071    21:16.903           1           1 1276903684 void perform_initialization(int, char **, INIT_DATA *, IN
IT_PHYSICS *) C => void g_init_run(int *, char ***, INIT_DATA *, INIT_PHYSICS *) C
100.0        0.127    21:16.903           1           3 1276903613 void d_init_run(int *, char ***, INIT_DATA *, INIT_PHYSIC
S *) C
100.0        0.127    21:16.903           1           3 1276903613 void g_init_run(int *, char ***, INIT_DATA *, INIT_PHYSIC
S *) C => void d_init_run(int *, char ***, INIT_DATA *, INIT_PHYSICS *) C
 99.9        0.571    21:16.174           1          20 1276174150 void d_init_run(int *, char ***, INIT_DATA *, INIT_PHYSIC
S *) C => void set_up_cauchy_data(INIT_DATA *, INIT_PHYSICS *) C
 99.9        0.571    21:16.174           1          20 1276174150 void set_up_cauchy_data(INIT_DATA *, INIT_PHYSICS *) C
 99.2        0.211    21:06.620           1           6 1266620615 void init_states(INIT_DATA *, INIT_PHYSICS *, CHART *, IN
PUT_SOLN **, RESTART_DATA *) C
 99.2        0.211    21:06.620           1           6 1266620615 void set_up_cauchy_data(INIT_DATA *, INIT_PHYSICS *) C =>
 void init_states(INIT_DATA *, INIT_PHYSICS *, CHART *, INPUT_SOLN **, RESTART_DATA *) C
 99.2        0.134    21:06.481           1           6 1266481343 void clip_front_to_subdomain(Front *) C
 99.2        0.134    21:06.481           1           6 1266481343 void init_states(INIT_DATA *, INIT_PHYSICS *, CHART *, IN
PUT_SOLN **, RESTART_DATA *) C => void clip_front_to_subdomain(Front *) C
 99.2            3    21:06.480          69         276   18354796 bool scatter_front(Front *) C
 99.2        0.115    21:06.480           1           4 1266480364 void clip_front_to_subdomain(Front *) C => bool scatter_f
ront(Front *) C
 99.2            1    21:06.480          68          84   18624708 bool form_subintfc_via_communication(Front *) C
 99.2            1    21:06.480          68          84   18624708 bool scatter_front(Front *) C => bool form_subintfc_via_c
ommunication(Front *) C
 99.2            3    21:06.480          68         385   18624707 bool form_subintfc_via_communication(Front *) C => bool g
_form_subintfc_via_communication2d(Front *) C
--More--
```

# Paraprof (for GUI Display)

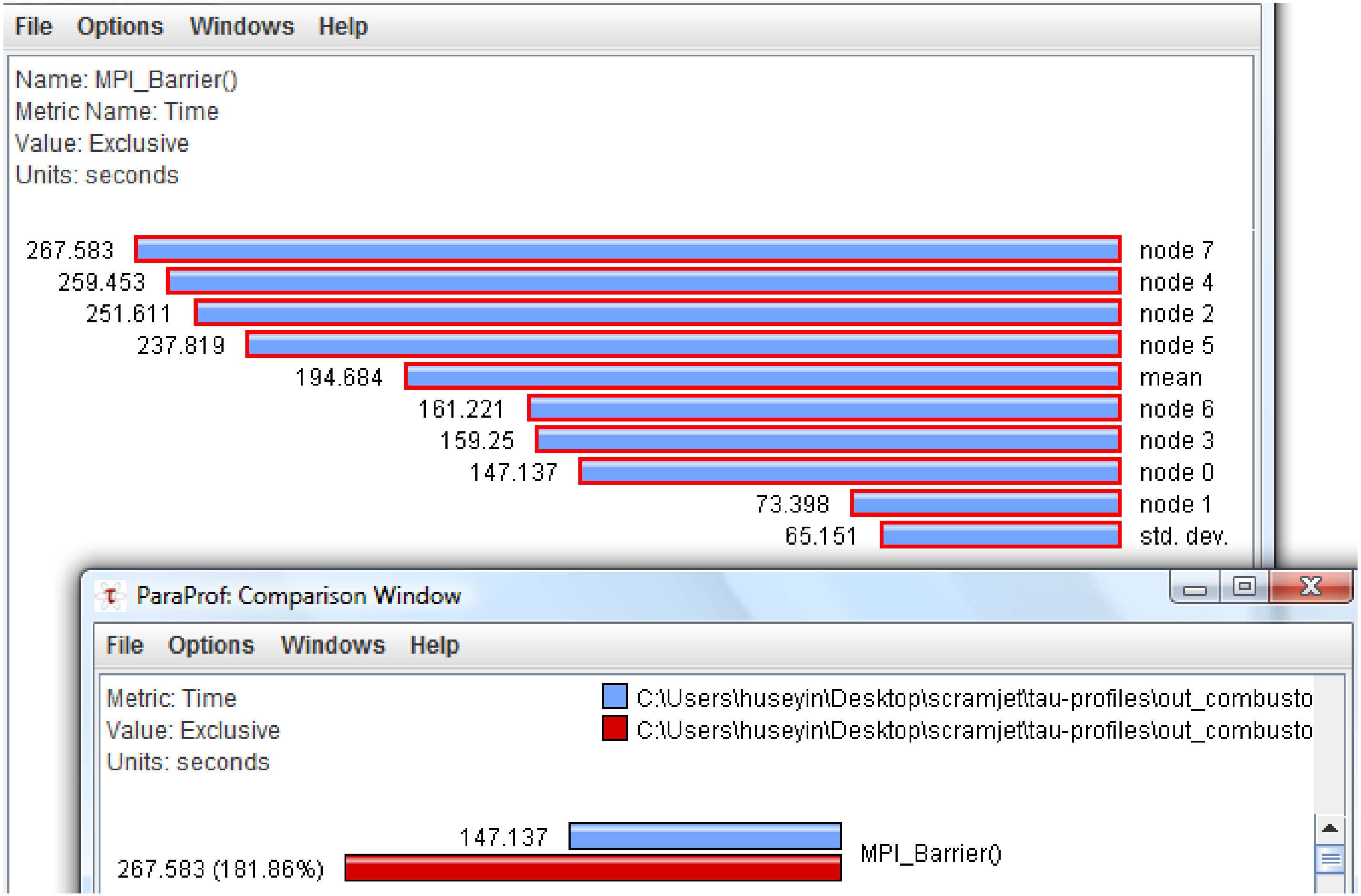
execute paraprof from the command line where the profiles  files are



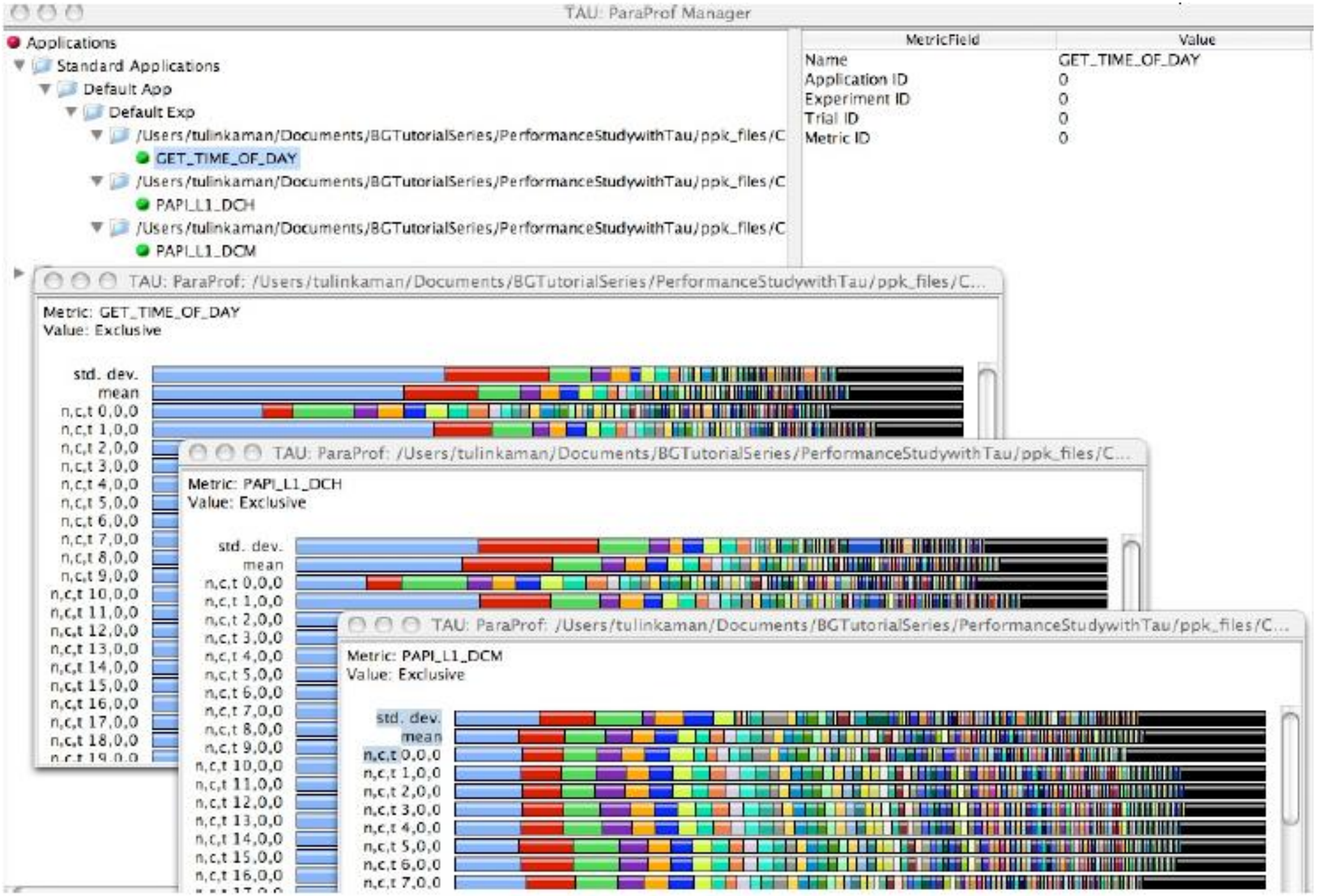
# Show Thread Statistics Text Window



# Function Data and Comparison Windows



# Performance Counters



# Custom Profiling

## Selective Instrumentation File

- Specify a list of **routines** to exclude or include (case sensitive)

| | |
|---|---|
| **BEGIN_EXCLUDE_LIST** | **BEGIN_INCLUDE_LIST** |
| F1() | int main(int, char **) |
| F2() | F2() |
| **END_EXCLUDE_LIST** | **END_INCLUDE_LIST** |

- Optionally specify a list of **files** to exclude or include

| | |
|---|---|
| **BEGIN_FILE_EXCLUDE_LIST** | **BEGIN_FILE_INCLUDE_LIST** |
| f1.c | main.c |
| f2.c | f2.c |
| **END_FILE_EXCLUDE_LIST** | **END_FILE_INCLUDE_LIST** |

*Usage* : **tau_instrumentor <pdbfile> <sourcefile> [-o <outputfile>] [-noinline] [-g groupname] [-i headerfile] [-c|-c++|-fortran] [-f<instr_req_file> ]**

# What loops account for the most time? How much?

- Generating a loop level profile

```
export TAU_MAKEFILE=$TAULIBDIR/Makefile.tau-mpi-pdt

export TAU_OPTIONS='-optTauSelectFile=select.tau -optVerbose'
```

```
$ cat select.tau

BEGIN_INSTRUMENT_SECTION
loops routine="#"
END_INSTRUMENT_SECTION
```

# Question 1: What routines account for the most time?

- Create a Flat Profile

```
$ export  PATH=/bgl/apps/TAUL/tau-2.18/bgl/bin:$PATH

$ export TAU_MAKEFILE= /bgl/apps/TAUL/tau-2.18/bgl/lib/Makefile.tau-mpi-pdt

$ make CC=tau_cc.sh CXX=tau_cxx.sh F90='tau_f90.sh -qfixed'
```

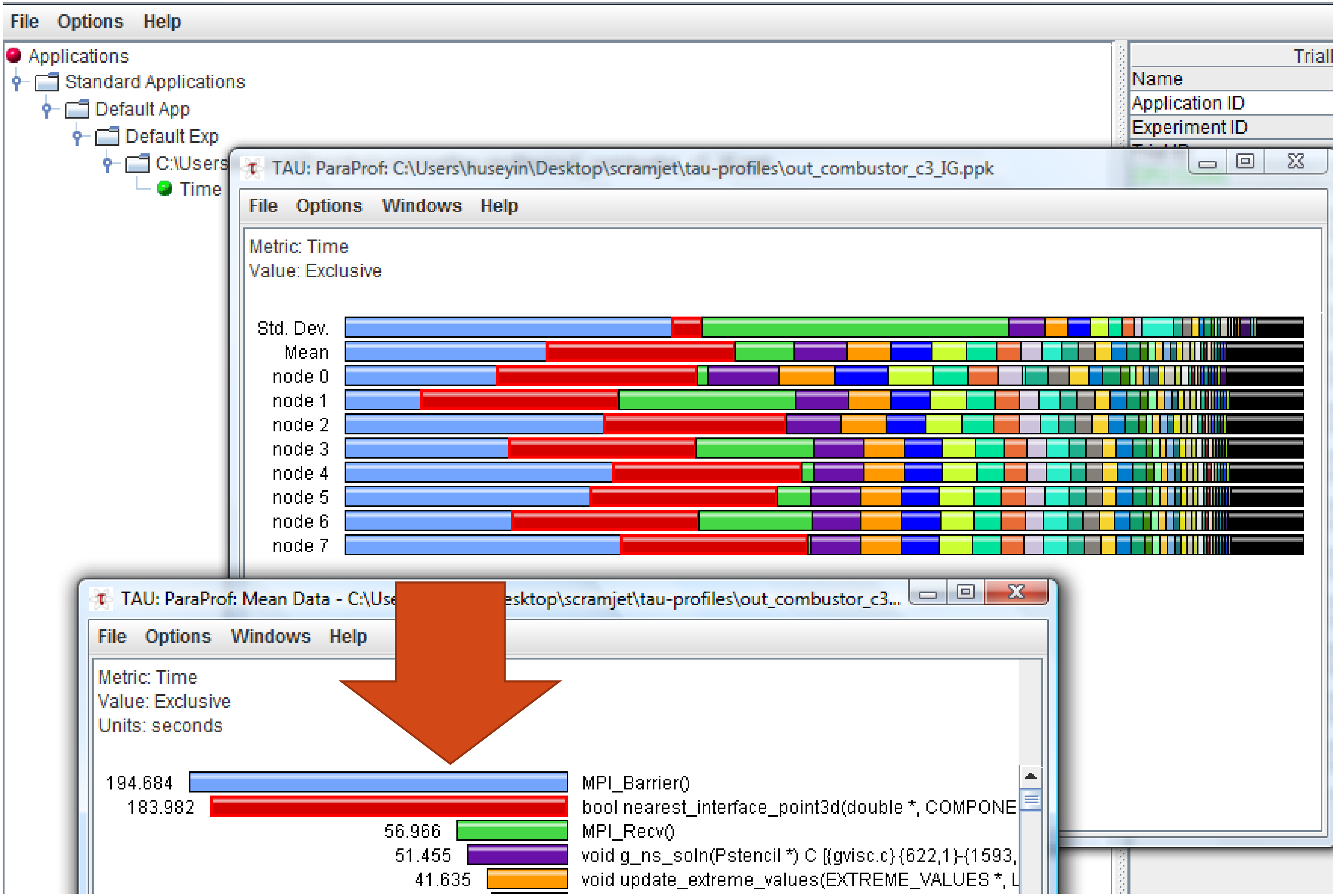
(Or edit Makefile and change F90=tau_f90.sh)

- In your job script file,

# @ arguments = -np 16 **-env PROFILEDIR=<profile-dir>** -exe …

```
$ llsubmit tau_app.run
$ cd <profile-dir>
$ paraprof --pack tau_app.ppk
$ paraprof tau_app.ppk
```

# Answer1


File Options Help
Applications
Standard Applications
Default App
Default Exp
C:\Users
Time
TrialI
Name
Application ID
Experiment ID
TAU: ParaProf: C:\Users\huseyin\Desktop\scramjet\tau-profiles\out_combustor_c3_IG.ppk
File Options Windows Help
Metric: Time
Value: Exclusive
Std. Dev.
Mean
node 0
node 1
node 2
node 3
node 4
node 5
node 6
node 7
TAU: ParaProf: Mean Data - C:\Use esktop\scramjet\tau-profiles\out_combustor_c3...
File Options Windows Help
Metric: Time
Value: Exclusive
Units: seconds
194.684
MPI_Barrier()
183.982
bool nearest_interface_point3d(double *, COMPONE
56.966
MPI_Recv()
51.455
void g_ns_soln(Pstencil *) C [{gvisc.c} {622,1}-{1593,
41.635
void update_extreme_values(EXTREME_VALUES *, L

# Question 2: **Who calls MPI_Barrier() function?**

- Generate call path profiles

```
$ export  PATH= /bgl/apps/TAUL/tau-2.18/bgl/bin:$PATH

$ export TAU_MAKEFILE=/bgl/apps/TAUL/tau-2.18/bgl/lib/Makefile.tau-callpath-mpi-pdt

$ make CC=tau_cc.sh CXX=tau_cxx.sh F90='tau_f90.sh -qfixed'
```

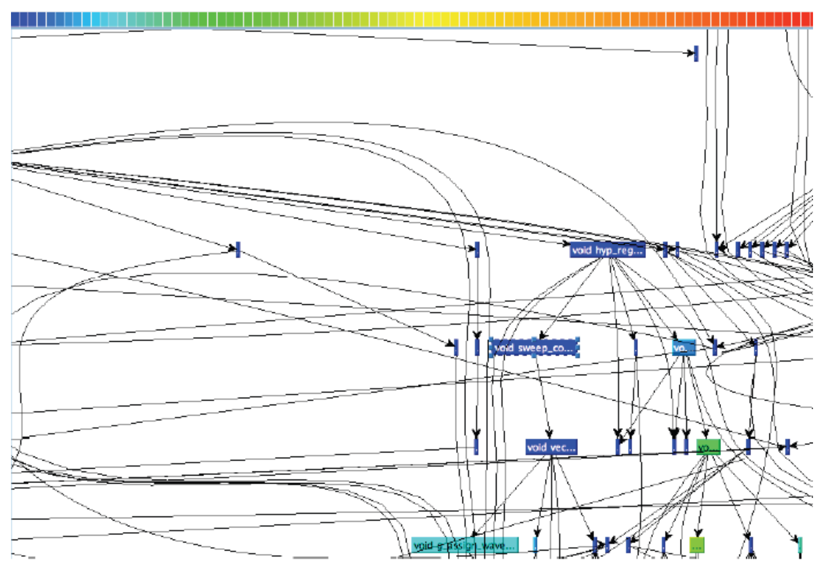
(Or edit Makefile and change F90=tau_f90.sh)

- In your job script file,

# @ arguments = -np 16 **-env PROFILEDIR=<profile-dir>** -exe …

```
$ export TAU_CALLPATH = 1
$ export TAU_CALLPATH_DEPTH = 100
$ llsubmit tau_app.run
$ cd <profile-dir>
$ paraprof --pack tau_app.ppk
$ paraprof tau_app.ppk
```

**(Windows → Thread → Call Path Relations**
**→ Call Graph)**



# Answer2: paraprof → Windows → Threads→ Call Path Relations

File Options Windows Help

Metric Name: Time
Sorted By: Exclusive
Units: seconds

```
            2.0E-5          2.0E-5     6/2890          bool surface_redistribute(Front *, bool *) C [{fredist3d.c} {1358,8}-{1527,1}]
             0.001           0.001     1152/2890       int append_buffer_surfacel(SURFACE *, SURFACE *, RECT_GRID *, int, int, P_LINK *,
            9.5E-6          9.5E-6     1/2890          void init_states(INIT_DATA *, INIT_PHYSICS *, CHART *, INPUT_SOLN **, RESTART_DAT
-->          0.006           0.006     2890            void add_time_start(int) C [{times.c} {264,8}-{267,1}]


            1.6E-4          55.199     6/606           void SGS3d(double, Front *, Wave *) C [{gvisc.c} {3060,1}-{4975,1}]
            7.2E-5            0.14     6/606           bool f_intfc_communication3d2(Front *) C [{fscat3d2.c} {87,8}-{344,1}]
             0.003           6.328     224/606         bool f_intfc_communication3d1(Front *) C [{fscat3d1.c} {439,1}-{732,1}]
            7.9E-5           8.632     10/606          bool create_directory(const char *, bool) C [{output.c} {927,8}-{1008,1}]
            8.9E-4            7.91     210/606         void communicate_default_comp(Front *) C [{fscat3d1.c} {4710,8}-{4803,1}]
             0.002         137.675     150/606         bool h_scatter_states(Wave *, Front *, int *, int) C [{hscatter.c} {139,9}-{175,1
-->          0.006         215.883     606             void pp_gsync(void) C [{ppsub.c} {1122,8}-{1148,1}]
             0.001           0.001     606/2612        void pp_okay_to_proceed(const char *, const char *) C [{ppsub.c} {148,1}-{158,1}]
           215.876         215.876     606/606         MPI_Barrier()
            1.5E-5          1.5E-5     12/100001       bool debugging(const char *) C [{debug.c} {470,8}-{504,1}]
            1.9E-4          2.0E-4     81/100001       void debug_print(const char *, const char *, ...) C [{debug.c} {413,8}-{448,1}]
```

`tau_cc.sh / tau_cxx.sh / tau_f90.sh`

Compiler wrappers (PDT instrumentation)

`TAU_MAKEFILE`

Set instrumentation definition file

`TAU_OPTIONS`

Set instrumentation options

`dynamic phase name='name' file='filename' line=start_line_# to line=end_line_#`

Specify dynamic Phase

`loops file='filename' routine='routine name'`

Instrument outer loops

`memory file='filename' routine='routine name'`

Track memory

`io file='filename' routine='routine name'`

Track IO

`TAU_PROFILE / TAU_TRACE`

Enable profiling and/or tracing

`PROFILEDIR / TRACEDIR`

Set profile/trace output directory

`TAU_CALLPATH=1 / TAU_CALLPATH_DEPTH`

Enable Callpath profiling, set callpath depth

`TAU_THROTTLE=1 / TAU_THROTTLE_NUMCALLS / TAU_THROTTLE_PERCALL`

Enable event throttling, set number of call, percall (us) threshold

`TAU_METRICS`

List of PAPI metrics to profile

# Applying Performance Tools to FRONTIER

- mature, production-quality multiphysics simulation package.
- supports a range of physics, including compressible and incompressible flow, MHD, turbulence models, fluid-structure interactions, phase transitions, and crystal growth.
- DoE Innovative and Novel Computational Impact on Theory and Experiment INCITE, **PI : James Glimm**
  - 2011 Uncertainty Quantification for Turbulent Mixing
    ANL IBM BG/P 10M core hours
  - 2012 Stochastic (w*) Convergence for Turbulent Combustion
    ANL IBM BG/P 35M core hours

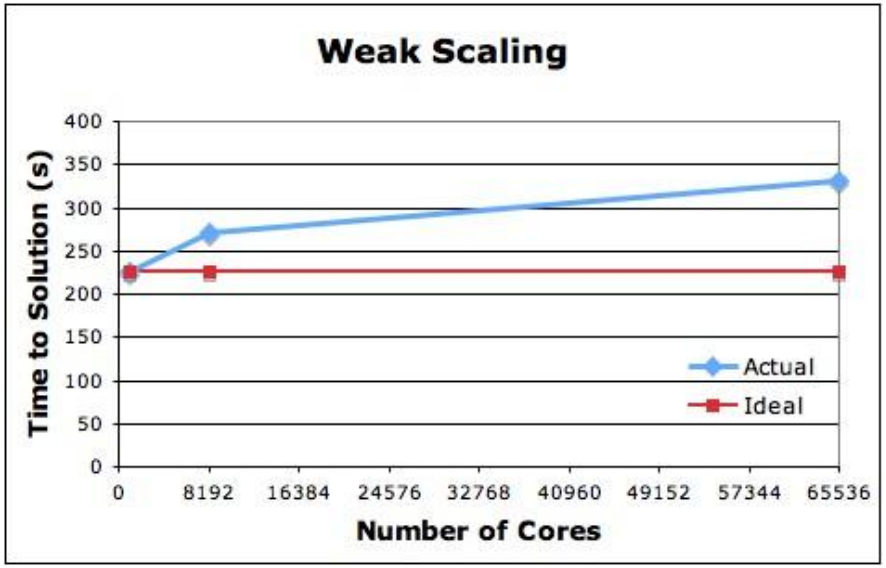
**62% efficiency on 163,840 cores**

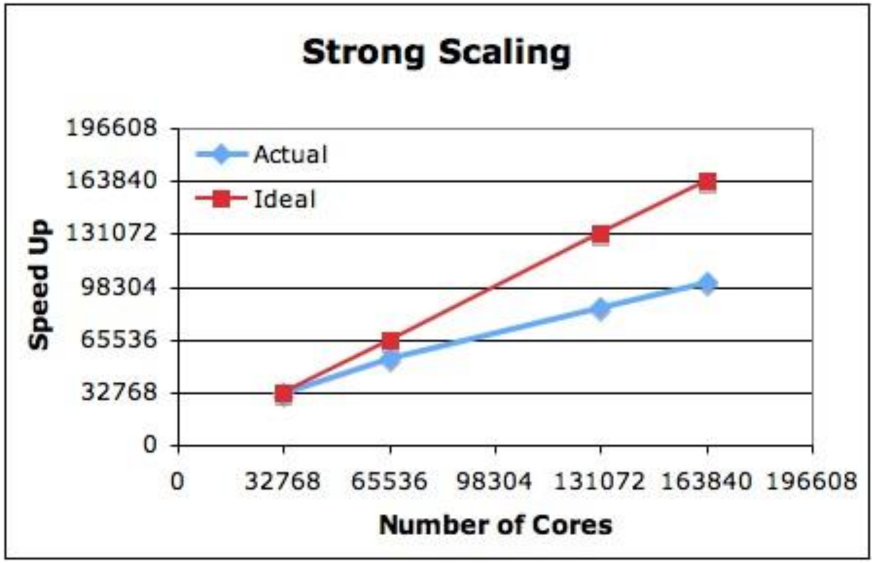
**Argonne National Laboratory ALCF Blue Gene P system**

# FRONTIER Scalability

We performed mesh refinement on grids of sizes of 24, 192, 1536 million cells, and run them using 1,024, 8,192, and 65,536 cores, respectively, so that the amount of computation for the volume remains constant per core.

The total problem size is fixed while the resources are increased. Scaling starts at 8 racks, which is the smallest configuration with sufficient memory.



# Stony Brook Center for Computational Science

Tutorial video and presentations are

http://www.stonybrook.edu/sbccs/tutorials.shtml

Tuning and Analysis Utilities: TAU

http://www.cs.uoregon.edu/Research/tau/home.php