Cray Performance Measurement and Analysis Tools

Heidi Poxon
Manager & Technical Lead, Performance Tools
Cray Inc.

Day 1 – Introduction to the Cray Performance Tools

- Cray performance tools overview

- Steps to using the tools
- Performance measurement on the Cray XE system
- Using HW performance counters
- Profiling applications

- Visualization of performance data through pat_report
- Visualization of performance data through Cray Apprentice2

- Building, launching and running on raven

Tools Overview

Design Goals

- Assist the user with application performance analysis and optimization
  - Help user identify important and meaningful information from potentially massive data sets
  - Help user identify problem areas instead of just reporting data
  - Bring optimization knowledge to a wider set of users

- Focus on ease of use and intuitive user interfaces
  - Automatic program instrumentation
  - Automatic analysis

- Target scalability issues in all areas of tool development
  - Data management
    - Storage, movement, presentation

Strengths

Provide a complete solution from instrumentation to measurement to analysis to visualization of data

- Performance measurement and analysis on large systems
  - Automatic Profiling Analysis
  - Load Imbalance
  - HW counter derived metrics
  - Predefined trace groups provide performance statistics for libraries called by program (blas, lapack, pgas runtime, netcdf, hdf5, etc.)
  - Observations of inefficient performance
  - Data collection and presentation filtering
  - Data correlates to user source (line number info, etc.)
  - Support MPI, SHMEM, OpenMP, UPC, CAF
  - Access to network counters
  - Minimal program perturbation

Strengths (2)

- Usability on large systems
  - Client / server
  - Scalable data format
  - Intuitive visualization of performance data
- Supports “recipe” for porting MPI programs to many-core or hybrid systems
- Integrates with other Cray PE software for more tightly coupled development environment

The Cray Performance Analysis Framework

- Supports traditional post-mortem performance analysis
  - Automatic identification of performance problems
    - Indication of causes of problems
    - Suggestions of modifications for performance improvement

  - pat_build: provides automatic instrumentation
  - CrayPat run-time library collects measurements (transparent to the user)
  - pat_report performs analysis and generates text reports
  - pat_help: online help utility
  - Cray Apprentice2: graphical visualization tool

Release Structure

- Separately licensed product from Cray
  - Uses FLEXlm licensing (checked during program instrumentation)
- Accessed on your system via software modulefiles
  - `% module load perftools`
- Current release: pertools/5.2.3
- Default and past versions available on a system (up to site)
  - `% module avail perf`

```
---------------------------- /opt/cray/modulefiles -------------------------------
perftools/5.1.2           perftools/5.2.2           perftools/5.3.0.8241
perftools/5.1.3           perftools/5.2.3(default) perftools/5.3.0.8250
```

New .ap2 Format + Client/Server Model

- Reduced pat_report processing and report generation times
- Reduced app2 data load times
- Graphical presentation handled locally (not passed through ssh connection)
- Better tool responsiveness
- Minimizes data loaded into memory at any given time
- Reduced server footprint on Cray XT/XE service node
- Larger data files handled (1.5TB .xf -> 800GB .ap2)

Scalable Data Format Reduced Processing Times

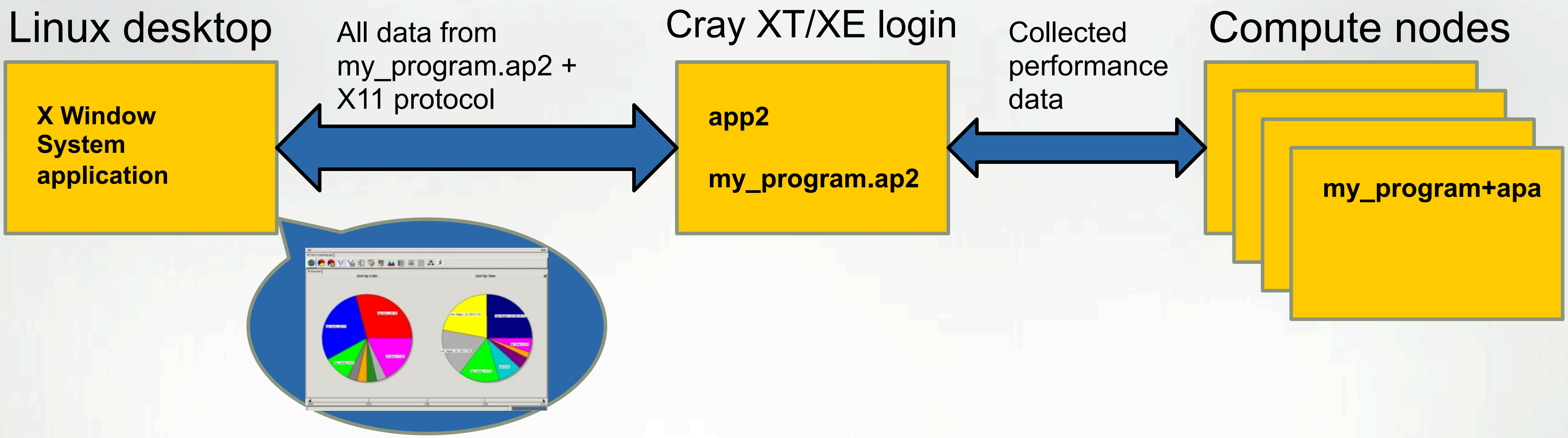
- CPMD
  - MPI, instrumented with pat_build –u, HWPC=1
  - 960 cores

	Perftools 5.1.3	Perftools 5.2.0
.xf -> .ap2	88.5 seconds	22.9 seconds
ap2 -> report	1512.27 seconds	49.6 seconds

- VASP
  - MPI, instrumented with pat_build –gmpi –u, HWPC=3
  - 768 cores

	Perftools 5.1.3	Perftools 5.2.0
.xf -> .ap2	45.2 seconds	15.9 seconds
ap2 -> report	796.9 seconds	28.0 seconds

Old Client/Server (Cray Performance Tools 5.0.0)



- Log into Cray XT/XE login node

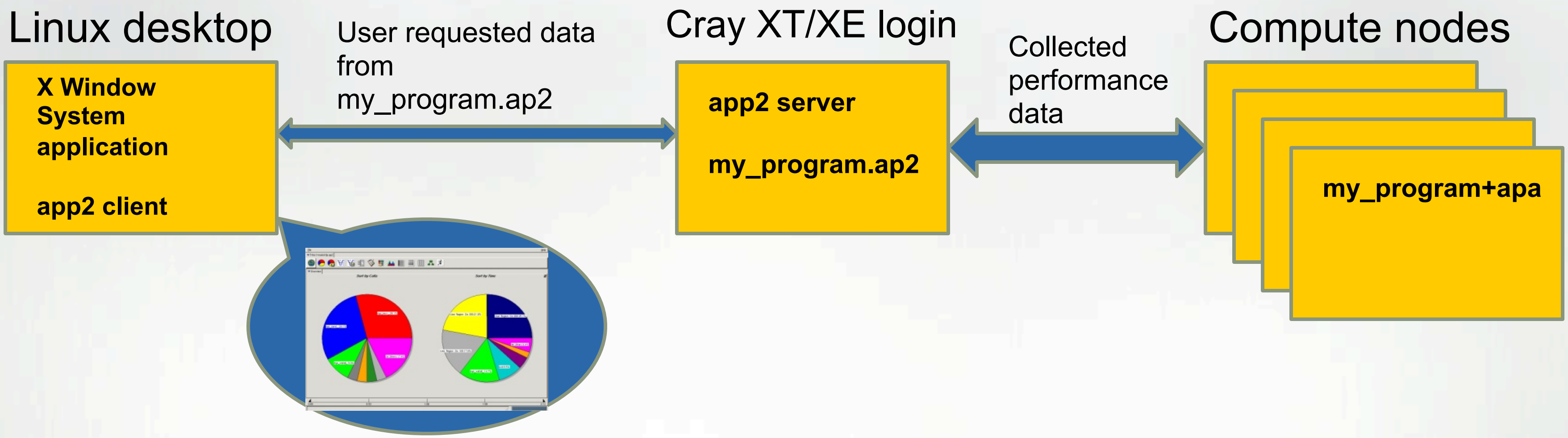
  `% ssh –Y kaibab`

- Launch Cray Apprentice2 on Cray XT/XE login node

  `% app2 /lus/scratch/mydir/my_program.ap2`

  - User interface displayed on desktop via ssh X11 forwarding
  - Entire .ap2 file loaded into memory on login node (can be Gbytes of data)

New Client/Server (Cray Performance Tools 5.2.0)



- Launch Cray Apprentice2 on desktop, point to data

  `% app2 kaibab:/lus/scratch/mydir/my_program.ap2`

  - User interface displayed on desktop via X Windows-based software
  - Minimal subset of data from.ap2 file loaded into memory on login node at any given time
  - Only data requested sent from server to client

Multiple Dimensions of Scalability

- Millions of lines of code
  - Automatic profiling analysis
    - Identifies top time consuming routines
    - Automatically creates instrumentation template customized to your application
- Lots of processes/threads
  - Load imbalance analysis
    - Identifies computational code regions and synchronization calls that could benefit most from load balance optimization
    - Estimates savings if corresponding section of code were balanced
- Long running applications
  - Detection of outliers

Steps to Using the Tools

Collecting Performance Data

- When performance measurement is triggered
  - External agent (asynchronous)
    - Sampling
      - Timer interrupt
      - Hardware counters overflow
  - Internal agent (synchronous)
    - Code instrumentation
      - Event based
      - Automatic or manual instrumentation
- How performance data is recorded
  - Profile ::= Summation of events over time
    - run time summarization (functions, call sites, loops, …)
  - Trace file ::= Sequence of events over time

Application Instrumentation with pat_build

- pat_build is a stand-alone utility that automatically instruments the application for performance collection

- Requires no source code or makefile modification
  - Automatic instrumentation at group (function) level
    - Groups: mpi, io, heap, math SW, …

- Performs link-time instrumentation
  - Requires object files
  - Instruments optimized code
  - Generates stand-alone instrumented program
  - Preserves original binary

Application Instrumentation with pat_build (2)

- Supports two categories of experiments
  - asynchronous experiments (sampling) which capture values from the call stack or the program counter at specified intervals or when a specified counter overflows
  - Event-based experiments (tracing) which count some events such as the number of times a specific system call is executed
- While tracing provides most useful information, it can be very heavy if the application runs on a large number of cores for a long period of time
- Sampling can be useful as a starting point, to provide a first overview of the work distribution

Program Instrumentation

- Large programs
  - Scaling issues more dominant
  - Use automatic profiling analysis to quickly identify top time consuming routines
  - Use loop statistics to quickly identify top time consuming loops

- Small (test) or short running programs
  - Scaling issues not significant
  - Can skip first sampling experiment and directly generate profile
  - For example: `% pat_build –u –g mpi my_program`

Program Instrumentation – Sampling

- Sampling is useful to determine where the program spends most of its time (functions and lines)
- The environment variable PAT_RT_EXPERIMENT allows the specification of the type of experiment prior to execution
  - samp_pc_time (default)
    - Samples the PC at intervals of 10,000 microseconds
    - Measures user CPU and system CPU time
    - Returns total program time and absolute and relative times each program counter was recorded
    - Optionally record the values of hardware counters specified with PAT_RT_HWPC
  - samp_pc_ovfl
    - Samples the PC at a given overflow of a HW counter
    - Does not allow collection of hardware counters
  - samp_cs_time
    - Sample the call stack at a given time interval

Program Instrumentation - Automatic Profiling Analysis

- Automatic profiling analysis (APA)
  - Provides simple procedure to instrument and collect performance data for novice users
  - Identifies top time consuming routines
  - Automatically creates instrumentation template customized to application for future in-depth measurement and analysis

Steps to Collecting Performance Data

- Access performance tools software

```
% module load perftools
```

- Build application keeping .o files (CCE: -h keepfiles)

```
% make clean
% make
```

- Instrument application for automatic profiling analysis
  - You should get an instrumented program a.out+pat

```
% pat_build -O apa a.out
```

- Run application to get top time consuming routines
  - You should get a performance file (“<sdatafile>.xf”) or multiple files in a directory <sdatadir>

```
% aprun … a.out+pat   (or  qsub <pat script>)
```

Steps to Collecting Performance Data (2)

- Generate report and .apa instrumentation file

```
% pat_report -o my_sampling_report [<sdatafile>.xf | <sdatadir>]
```

- Inspect .apa file and sampling report

- Verify if additional instrumentation is needed

APA File Example

```
#  You can edit this file, if desired, and use it
#   to reinstrument the program for  tracing like this:
#
#          pat_build -O standard.cray-xt.PE-2.1.56HD.pgi-8.0.amd64.pat-5.0.0.2-Oapa.
          512.quad.cores.seal.090405.1154.mpi.pat_rt_exp=default.pat_rt_hwpc=none.
          14999.xf.xf.apa
#
#  These suggested trace options are based on data from:
#
#    /home/users/malice/pat/Runs/Runs.seal.pat5001.2009Apr04/./pat.quad/homme/
          standard.cray-xt.PE-2.1.56HD.pgi-8.0.amd64.pat-5.0.0.2-Oapa.
          512.quad.cores.seal.090405.1154.mpi.pat_rt_exp=default.pat_rt_hwpc=none.
          14999.xf.xf.cdb
# ----------------------------------------------------------------------

#       HWPC group to collect by default.

  -Drtenv=PAT_RT_HWPC=1  # Summary with TLB metrics.

# ----------------------------------------------------------------------

#       Libraries to trace.

  -g mpi

# ----------------------------------------------------------------------

#       User-defined functions to trace, sorted by % of samples.

#       The way these functions are filtered can be controlled with
#       pat_report options (values used for this file are shown):
#
#       -s apa_max_count=200     No more than 200 functions are listed.
#       -s apa_min_size=800      Commented out if text size < 800 bytes.
#       -s apa_min_pct=1         Commented out if it had < 1% of samples.
#       -s apa_max_cum_pct=90    Commented out after cumulative 90%.

#       Local functions are listed for completeness, but cannot be traced.

  -w  # Enable tracing of user-defined functions.
    # Note: -u should NOT be specified as an additional option.
```

```
# 31.29%  38517 bytes
         -T prim_advance_mod_preq_advance_exp_

# 15.07%  14158 bytes
         -T prim_si_mod_prim_diffusion_

#  9.76%  5474 bytes
         -T derivative_mod_gradient_str_nonstag_

. . .

#  2.95%  3067 bytes
         -T forcing_mod_apply_forcing_

#  2.93%  118585 bytes
         -T column_model_mod_applycolumnmodel_

#  Functions below this point account for less than 10% of samples.

#  0.66%  4575 bytes
#          -T bndry_mod_bndry_exchangev_thsave_time_

#  0.10%  46797 bytes
#          -T baroclinic_inst_mod_binst_init_state_

#  0.04%  62214 bytes
#          -T prim_state_mod_prim_printstate_

. . .
#  0.00%  118 bytes
#          -T time_mod_timelevel_update_

# ----------------------------------------------------------------------

  -o preqx.cray-xt.PE-2.1.56HD.pgi-8.0.amd64.pat-5.0.0.2.x+apa
           # New instrumented program.

  /.AUTO/cray/css.pe_tools/malice/craypat/build/pat/2009Apr03/2.1.56HD/amd64/homme/
           pgi/pat-5.0.0.2/homme/2005Dec08/build.Linux/preqx.cray-
           xt.PE-2.1.56HD.pgi-8.0.amd64.pat-5.0.0.2.x  # Original program.
```

Generating Profile from APA

- Instrument application for further analysis (a.out+apa)

```
% pat_build -O <apafile>.apa
```

- Run application

```
% aprun … a.out+apa  (or  qsub <apa script>)
```

- Generate text report and visualization file (.ap2)

```
% pat_report -o my_text_report.txt [<datafile>.xf |
  <datadir>]
```

- View report in text and/or with Cray Apprentice^2

```
% app2 <datafile>.ap2
```

Where to Run Instrumented Application

- MUST run on Lustre (/work/… , /lus/…, /scratch/…, etc.)
- Number of files used to store raw data
  - 1 file created for program with 1 – 256 processes
  - $\sqrt{n}$ files created for program with 257 – *n* processes
  - Ability to customize with PAT_RT_EXPFILE_MAX

Files Generated and the Naming Convention

File Suffix	Description
a.out+pat	Program instrumented for data collection
a.out…s.xf	Raw data for sampling experiment, available after application execution
a.out…t.xf	Raw data for trace (summarized or full) experiment, available after application execution
a.out…st.ap2	Processed data, generated by pat_report, contains application symbol information
a.out…s.apa	Automatic profiling pnalysis template, generated by pat_report (based on pat_build –O apa experiment)
a.out+apa	Program instrumented using .apa file
MPICH_RANK_ORDER.Custom	Rank reorder file generated by pat_report from automatic grid detection an reorder suggestions

Why Should I generate an ".ap2" file?

- The ".ap2" file is a self contained compressed performance file
  - Normally it is about 5 times smaller than the ".xf" file
  - Contains the information needed from the application binary
    - Can be reused, even if the application binary is no longer available or if it was rebuilt
  - It is the only input format accepted by Cray Apprentice^2

CrayPat Runtime Options

- Runtime controlled through PAT_RT_XXX environment variables
- See intro_craypat(1) man page
- Examples of control
  - Enable full trace
  - Change number of data files created
  - Enable collection of HW counters
  - Enable collection of network counters
  - Enable tracing filters to control trace file size (max threads, max call stack depth, etc.)

Example Runtime Environment Variables

- Optional timeline view of program available
  - export PAT_RT_SUMMARY=0
  - View trace file with Cray Apprentice^2
- Number of files used to store raw data:
  - 1 file created for program with 1 – 256 processes
  - $\sqrt{n}$ files created for program with 257 – n processes
  - Ability to customize with PAT_RT_EXPFILE_MAX
- Request hardware performance counter information:
  - export PAT_RT_HWPC=<HWPC Group>
  - Can specify events or predefined groups

View Data with pat_report

pat_report

- Performs data conversion
  - Combines information from binary with raw performance data
- Performs analysis on data
- Generates text report of performance results
- Formats data for input into Cray Apprentice^2

Job Execution Information

```
CrayPat/X:  Version 5.2.3.8078 Revision 8078 (xf 8063)  08/25/11 …

Number of PEs (MPI ranks):   16

Numbers of PEs per Node:     16

Numbers of Threads per PE:    1

Number of Cores per Socket:  12

Execution start time:  Thu Aug 25 14:16:51 2011

System type and speed:  x86_64 2000 MHz

Current path to data file:
  /lus/scratch/heidi/ted_swim/mpi-openmp/run/swim+pat+27472-34t.ap2

Notes for table 1:
…
```

Sampling Output (Table 1)

```
Notes for table 1:
...
Table 1:  Profile by Function

 Samp % | Samp |  Imb.  |   Imb.  |Group
        |      |  Samp  | Samp %  | Function
        |      |        |         |  PE='HIDE'

 100.0% |  775 |     -- |      -- |Total
|------------------------------------------
|  94.2% |  730 |     -- |      -- |USER
||-----------------------------------------
||  43.4% |  336 |   8.75 |    2.6% |mlwxyz_
||  16.1% |  125 |   6.28 |    4.9% |half_
||   8.0% |   62 |   6.25 |    9.5% |full_
||   6.8% |   53 |   1.88 |    3.5% |artv_
||   4.9% |   38 |   1.34 |    3.6% |bnd_
||   3.6% |   28 |   2.00 |    6.9% |currenf_
||   2.2% |   17 |   1.50 |    8.6% |bndsf_
||   1.7% |   13 |   1.97 |   13.5% |model_
||   1.4% |   11 |   1.53 |   12.2% |cfl_
||   1.3% |   10 |   0.75 |    7.0% |currenh_
||   1.0% |    8 |   5.28 |   41.9% |bndbo_
||   1.0% |    8 |   8.28 |   53.4% |bndto_
||=========================================
|   5.4% |   42 |     -- |      -- |MPI
||-----------------------------------------
||   1.9% |   15 |   4.62 |   23.9% |mpi_sendrecv_
||   1.8% |   14 |  16.53 |   55.0% |mpi_bcast_
||   1.7% |   13 |   5.66 |   30.7% |mpi_barrier_
|==========================================
```

pat_report: Flat Profile

```
Table 1:  Profile by Function Group and Function

 Time % |        Time |Imb. Time |   Imb. | Calls |Group
        |             |          | Time % |       | Function
        |             |          |        |       |  PE='HIDE'

 100.0% | 104.593634 |       -- |     -- | 22649 |Total
|------------------------------------------------------------
|  71.0% |  74.230520 |       -- |     -- | 10473 |MPI
||-----------------------------------------------------------
||  69.7% |  72.905208 | 0.508369 |   0.7% |   125 |mpi_allreduce_
||   1.0% |   1.050931 | 0.030042 |   2.8% |    94 |mpi_alltoall_
||===========================================================
|  25.3% |  26.514029 |       -- |     -- |    73 |USER
||-----------------------------------------------------------
||  16.7% |  17.461110 | 0.329532 |   1.9% |    23 |selfgravity_
||   7.7% |   8.078474 | 0.114913 |   1.4% |    48 |ffte4_
||===========================================================
|   2.5% |   2.659429 |       -- |     -- |   435 |MPI_SYNC
||-----------------------------------------------------------
||   2.1% |   2.207467 | 0.768347 |  26.2% |   172 |mpi_barrier_(sync)
||===========================================================
|   1.1% |   1.188998 |       -- |     -- | 11608 |HEAP
||-----------------------------------------------------------
||   1.1% |   1.166707 | 0.142473 |  11.1% |  5235 |free
|============================================================
```

pat_report: Message Stats by Caller

```
Table 4:  MPI Message Stats by Caller

    MPI Msg |MPI Msg |  MsgSz |  4KB<= |Function
      Bytes |  Count |   <16B |  MsgSz | Caller
            |        |  Count |  <64KB |   PE[mmm]
            |        |        |  Count |

 15138076.0 | 4099.4 |  411.6 | 3687.8 |Total
|-----------------------------------------------
| 15138028.0 | 4093.4 |  405.6 | 3687.8 |MPI_ISEND
||---------------------------------------------
||  8080500.0 | 2062.5 |   93.8 | 1968.8 |calc2_
3|            |        |        |        | MAIN_
||||-------------------------------------------
4|||  8216000.0 | 3000.0 | 1000.0 | 2000.0 |pe.0
4|||  8208000.0 | 2000.0 |     -- | 2000.0 |pe.9
4|||  6160000.0 | 2000.0 |  500.0 | 1500.0 |pe.15
||||===========================================
||  6285250.0 | 1656.2 |  125.0 | 1531.2 |calc1_
3|            |        |        |        | MAIN_
||||-------------------------------------------
4|||  8216000.0 | 3000.0 | 1000.0 | 2000.0 |pe.0
4|||  6156000.0 | 1500.0 |     -- | 1500.0 |pe.3
4|||  6156000.0 | 1500.0 |     -- | 1500.0 |pe.5
||||===========================================
. . .
```

Using HW Performance Counters

Hardware Performance Counters

- AMD Opteron Hardware Performance Counters
  - **Four** 48-bit performance counters.
    - Each counter can monitor a single event
      - Count specific processor events
        - the processor increments the counter when it detects an occurrence of the event
        - (e.g., cache misses)
      - Duration of events
        - the processor counts the number of processor clocks it takes to complete an event
        - (e.g., the number of clocks it takes to return data from memory after a cache miss)
  - Time Stamp Counters (TSC)
    - Cycles (user time)

PAPI Predefined Events

- Common set of events deemed relevant and useful for application performance tuning
  - Accesses to the memory hierarchy, cycle and instruction counts, functional units, pipeline status, etc.
  - The “papi_avail” utility shows which predefined events are available on the system – execute on compute node
- PAPI also provides access to native events
  - The “papi_native_avail” utility lists all AMD native events available on the system – execute on compute node
- PAPI uses perf_events Linux subsystem
- Information on PAPI and AMD native events
  - pat_help counters
  - man intro_papi

Hardware Counters Selection

- HW counter collection enabled with PAT_RT_HWPC environment variable

- PAT_RT_HWPC <set number> | <event list>
  - A set number can be used to select a group of predefined hardware counters events (recommended)
    - CrayPat provides 22 groups on the Cray XT/XE systems
    - See pat_help(1) or the hwpc(5) man page for a list of groups
  - Alternatively a list of hardware performance counter event names can be used
  - Hardware counter events are not collected by default

HW Counter Information Available in Reports

- Raw data
- Derived metrics
- Desirable thresholds

Example: HW counter data and Derived Metrics

```
  PAPI_TLB_DM  Data translation lookaside buffer misses
  PAPI_L1_DCA  Level 1 data cache accesses
  PAPI_FP_OPS  Floating point operations
  DC_MISS      Data Cache Miss
  User_Cycles  Virtual Cycles
========================================================================
USER
------------------------------------------------------------------------
  Time%                                             98.3%
  Time                                           4.434402 secs
  Imb.Time                                             -- secs
  Imb.Time%                                            --
  Calls                          0.001M/sec        4500.0 calls
  PAPI_L1_DCM                   14.820M/sec      65712197 misses
  PAPI_TLB_DM                    0.902M/sec       3998928 misses
  PAPI_L1_DCA                  333.331M/sec    1477996162 refs
  PAPI_FP_OPS                  445.571M/sec    1975672594 ops
  User time (approx)             4.434 secs   11971868993 cycles  100.0%Time
  Average Time per Call                          0.000985 sec
  CrayPat Overhead : Time         0.1%
  HW FP Ops / User time        445.571M/sec    1975672594 ops   4.1%peak(DP)
  HW FP Ops / WCT              445.533M/sec
  Computational intensity         0.17 ops/cycle     1.34 ops/ref
  MFLOPS (aggregate)          1782.28M/sec
  TLB utilization               369.60 refs/miss    0.722 avg uses
  D1 cache hit,miss ratios       95.6% hits          4.4% misses
  D1 cache utilization (misses)  22.49 refs/miss    2.811 avg hits
========================================================================
```

PAT_RT_HWPC=1
Flat profile data
Raw counts
Derived metrics

Predefined Interlagos HW Counter Groups

See pat_help -> counters -> amd_fam15h –> groups

0: Summary with instructions metrics

1: Summary with TLB metrics

2: L1 and L2 Metrics

3: Bandwidth information

4: <Unused>

5: Floating operations dispatched

6: Cycles stalled, resources idle

7: Cycles stalled, resources full

8: Instructions and branches

9: Instruction cache

10: Cache Hierarchy (unsupported for IL)

Predefined Interlagos HW Counter Groups (cont'd)

11: Floating point operations dispatched

12: Dual pipe floating point operations dispatched

13: Floating point operations SP

14: Floating point operations DP

L3 (socket and core level) (unsupported)

19: Prefetchs

20: FP, D1, TLB, MIPS <<-new for Interlagos

21: FP, D1, TLB, Stalls

22: D1, TLB, MemBW

New HW counter groups for Interlagos (6 counters)

- Group 20: FP, D1, TLB, MIPS
  - PAPI_FP_OPS
  - PAPI_L1_DCA
  - PAPI_L1_DCM
  - PAPI_TLB_DM
  - DATA_CACHE_REFILLS_FROM_NORTHBRIDGE
  - PAPI_TOT_INS

- Group 21: FP, D1, TLB, Stalls
  - PAPI_FP_OPS
  - PAPI_L1_DCA
  - PAPI_L1_DCM
  - PAPI_TLB_DM
  - DATA_CACHE_REFILLS_FROM_NORTHBRIDGE
  - PAPI_RES_STL

Hardware Counters Overhead Elimination

```
Hardware performance counter events:
  PAPI_L1_DCM  Level 1 data cache misses
  CYCLES_RTC   User Cycles (approx, from rtc)
  PAPI_L1_DCA  Level 1 data cache accesses
  PAPI_TLB_DM  Data translation lookaside buffer misses
  PAPI_FP_OPS  Floating point operations

Estimated minimum overhead per call of a traced function,
  which was subtracted from the data shown in this report
  (for raw data, use the option:  -s overhead=include):
    PAPI_L1_DCM       8.040  misses
    PAPI_TLB_DM       0.005  misses
    PAPI_L1_DCA     474.080  refs
    PAPI_FP_OPS       0.000  ops
    CYCLES_RTC     1863.680  cycles
    Time              0.693  microseconds
```

PAT_RT_HWPC=2 (L1 and L2 Metrics)

```
========================================================================
USER
------------------------------------------------------------------------
  Time%                                                 98.3%
  Time                                              4.436808 secs
  Imb.Time                                                -- secs
  Imb.Time%                                               --
  Calls                           0.001M/sec          4500.0 calls
  DATA_CACHE_REFILLS:
    L2_MODIFIED:L2_OWNED:
    L2_EXCLUSIVE:L2_SHARED        9.821M/sec        43567825 fills
  DATA_CACHE_REFILLS_FROM_SYSTEM:
    ALL                          24.743M/sec       109771658 fills
  PAPI_L1_DCM                    14.824M/sec        65765949 misses
  PAPI_L1_DCA                   332.960M/sec      1477145402 refs
  User time (approx)              4.436 secs     11978286133 cycles  100.0%Time
  Average Time per Call                             0.000986 sec
  CrayPat Overhead : Time          0.1%
  D1 cache hit,miss ratios        95.5% hits            4.5% misses
  D1 cache utilization (misses)   22.46 refs/miss      2.808 avg hits
  D1 cache utilization (refills)   9.63 refs/refill    1.204 avg uses
  D2 cache hit,miss ratio         28.4% hits           71.6% misses
  D1+D2 cache hit,miss ratio      96.8% hits            3.2% misses
  D1+D2 cache utilization         31.38 refs/miss      3.922 avg hits
  System to D1 refill            24.743M/sec       109771658 lines
  System to D1 bandwidth       1510.217MB/sec     7025386144 bytes
  D2 to D1 bandwidth            599.398MB/sec     2788340816 bytes
========================================================================
```

Example: Observations and Suggestions

D1 + D2 cache utilization: 39.8% of total execution time was spent in 4 functions with combined D1 and D2 cache hit ratios below the desirable minimum of 97.0%. Cache utilization might be improved by modifying the alignnment or stride of references to data arrays in these functions.

D1_D2_cache_hit_ratio	Time%	Function
56.8%	12.0%	calc3_
77.9%	6.4%	calc2_
95.7%	1.4%	calc1_
96.3%	20.0%	calc3_.LOOP@li.80

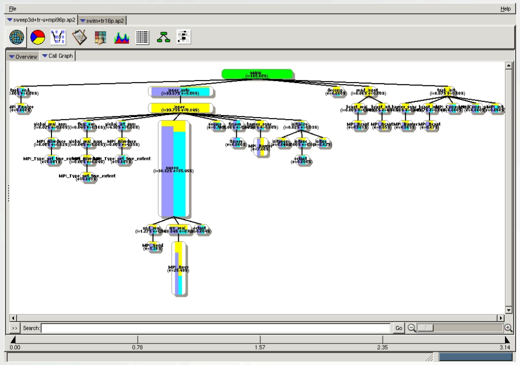
TLB utilization: 19.6% of total execution time was spent in 3 functions with fewer than the desirable minimum of 512 data references per TLB miss. TLB utilization might be improved by modifying the alignnment or stride of references to data arrays in these functions.

LS_per_TLB_DM	Time%	Function
2.56	12.0%	calc3_
5.32	6.3%	calc2_

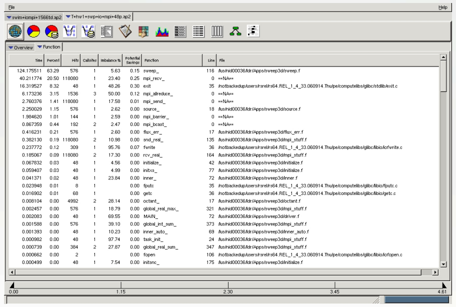
Profile Visualization with Cray Apprentice2

Cray Apprentice²

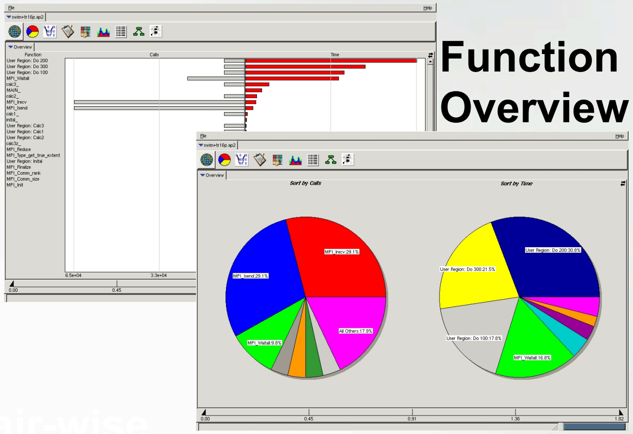
Call Graph Profile



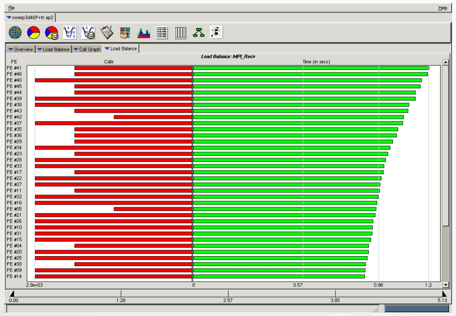
Function Profile



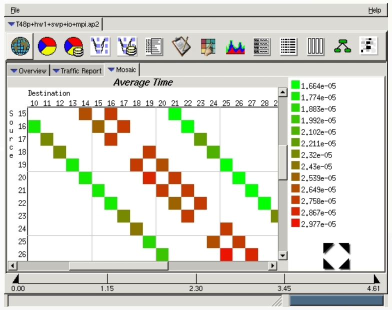
Function Overview



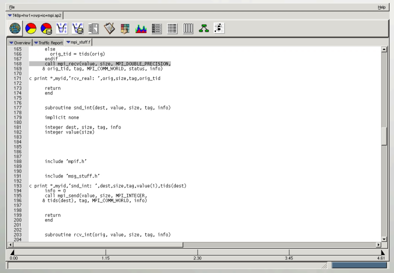
Load balance views



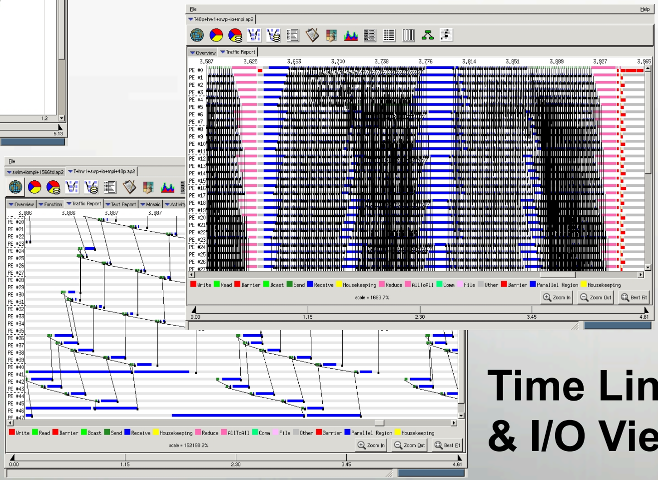
Pair-wise Communication View



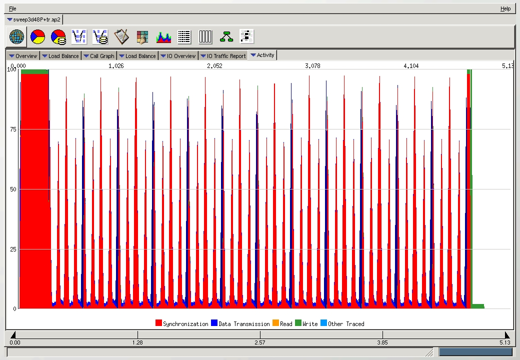
Source code mapping



Time Line & I/O Views



Communication & I/O Activity View

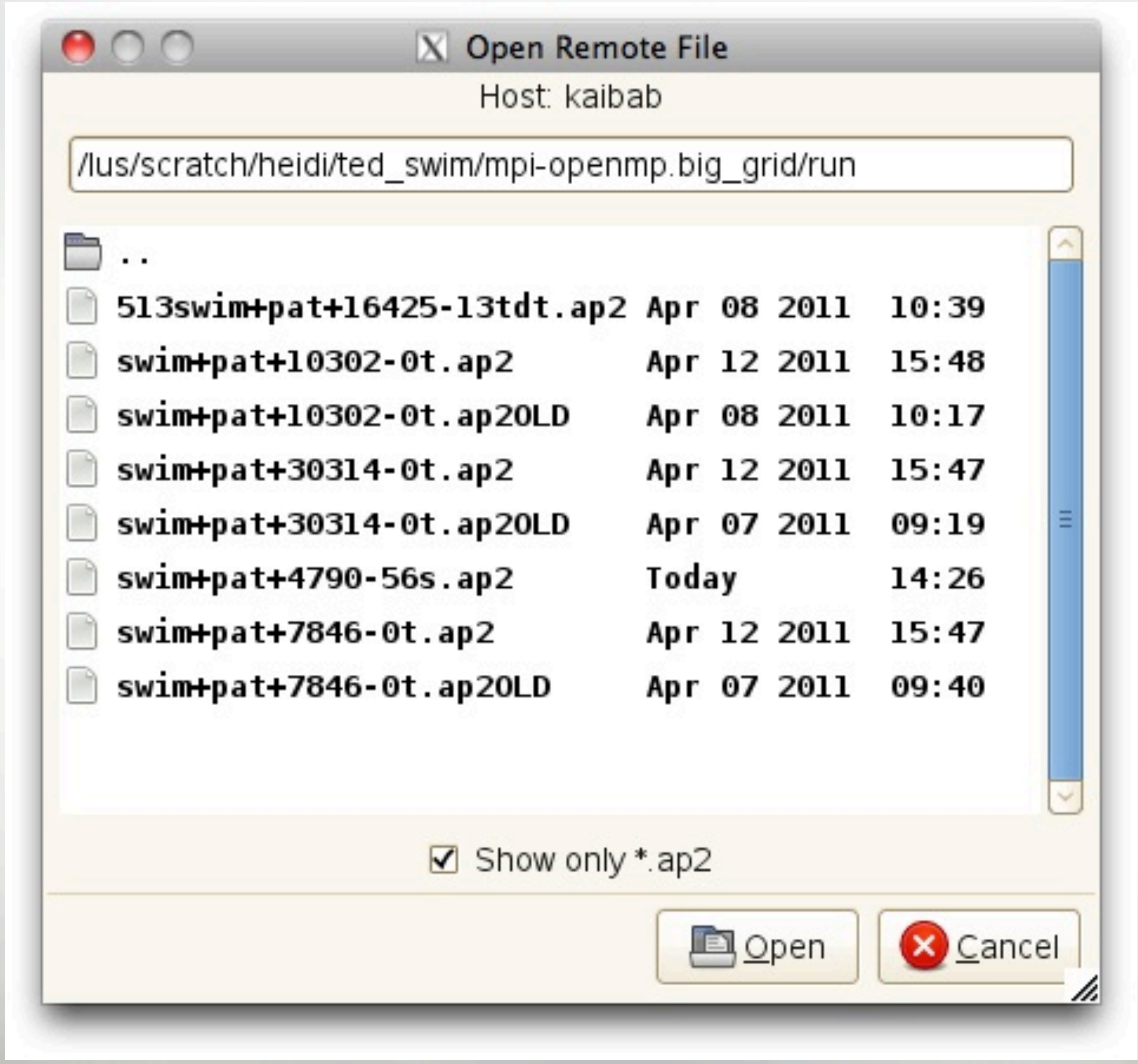


Cray Apprentice^2

- Call graph profile
- Communication statistics
- Time-line view
  - Communication
  - I/O
- Activity view
- Pair-wise communication statistics
- Text reports
- Source code mapping

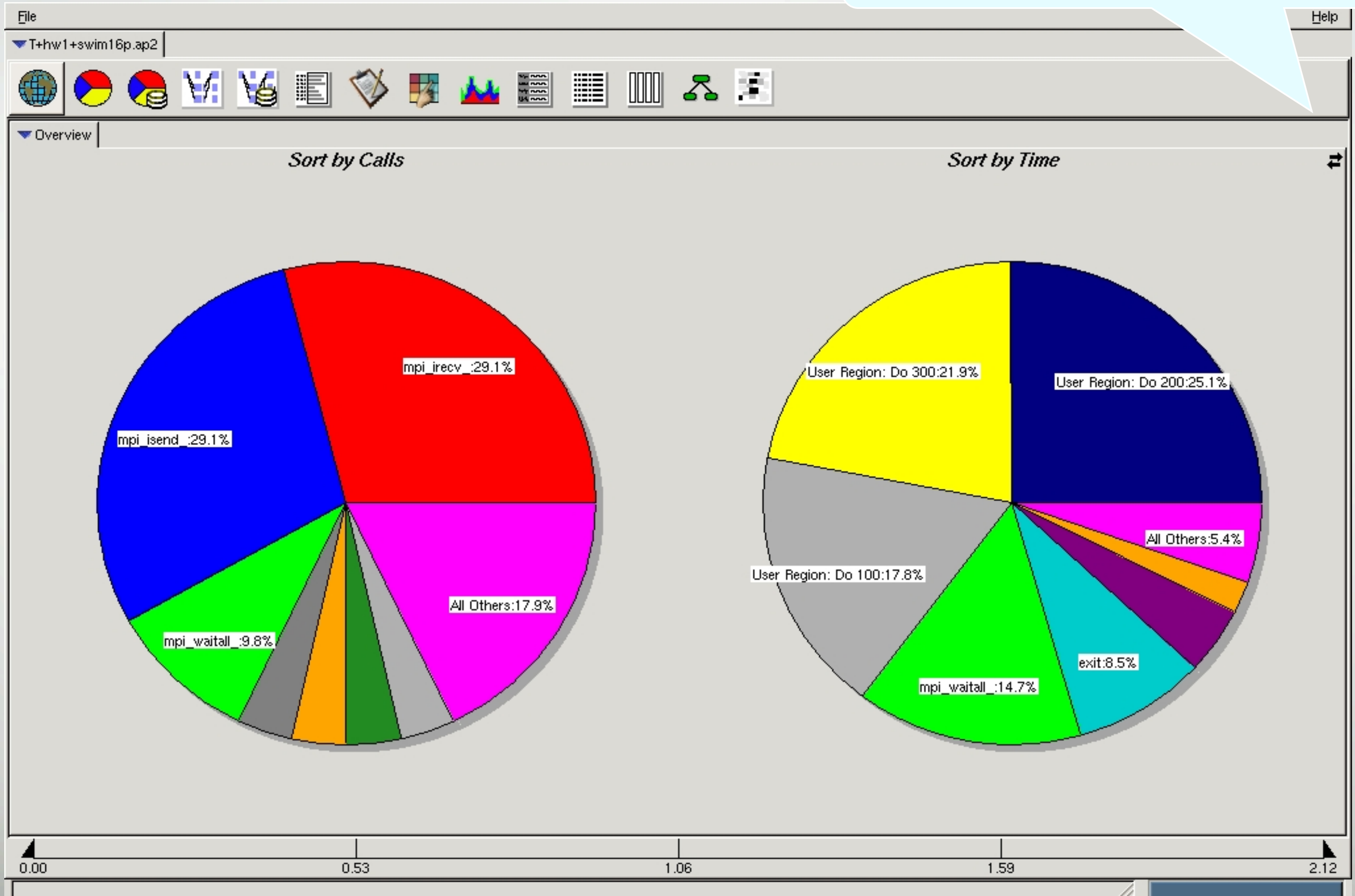
- Cray Apprentice^2 helps identify:
  - Load imbalance
  - Excessive communication
  - Network contention
  - Excessive serialization
  - I/O Problems

Accessing Remote File from app2 Client

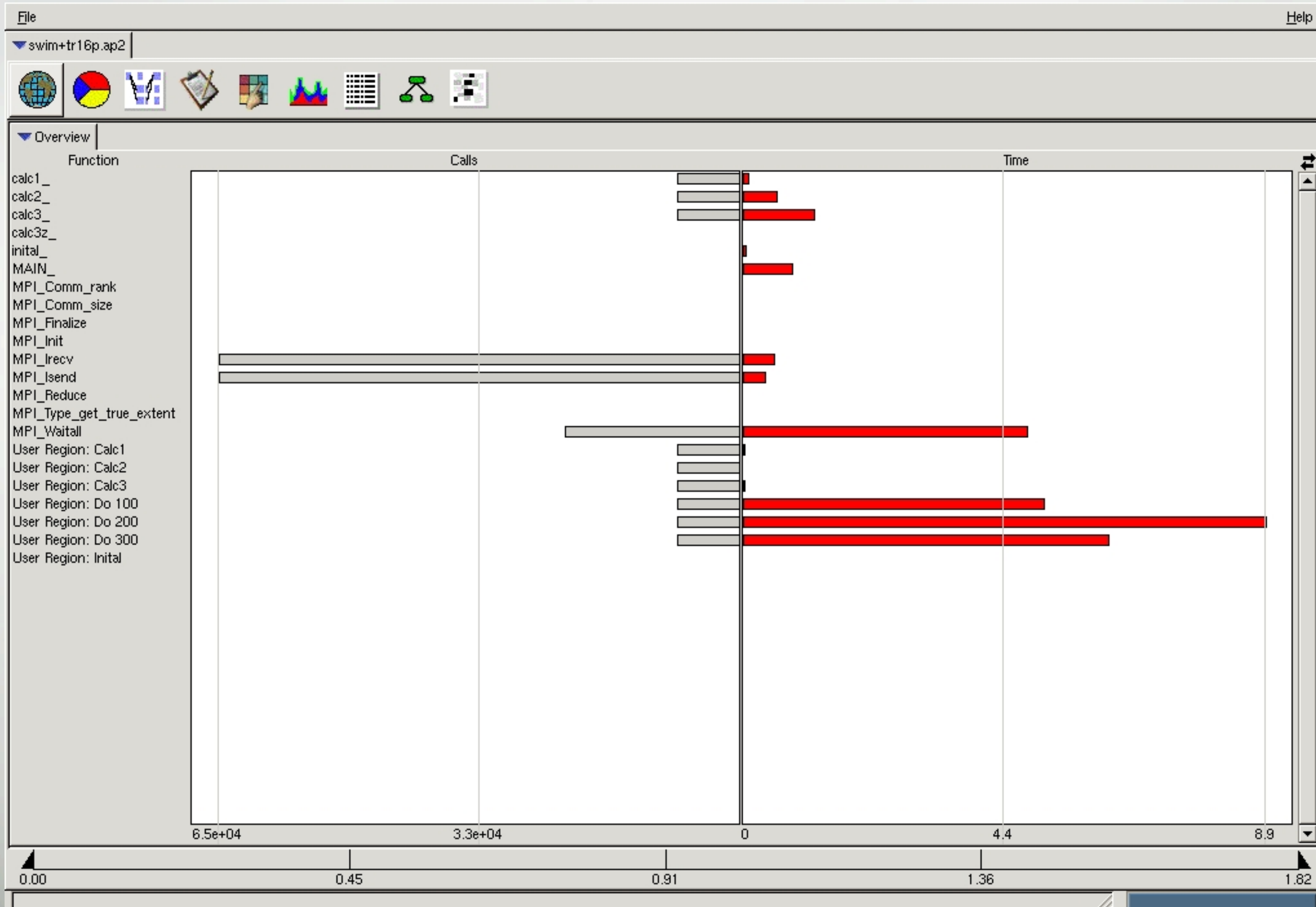


Statistics Overview

Switch Overview display



Function Profile



Function Profile

File Help

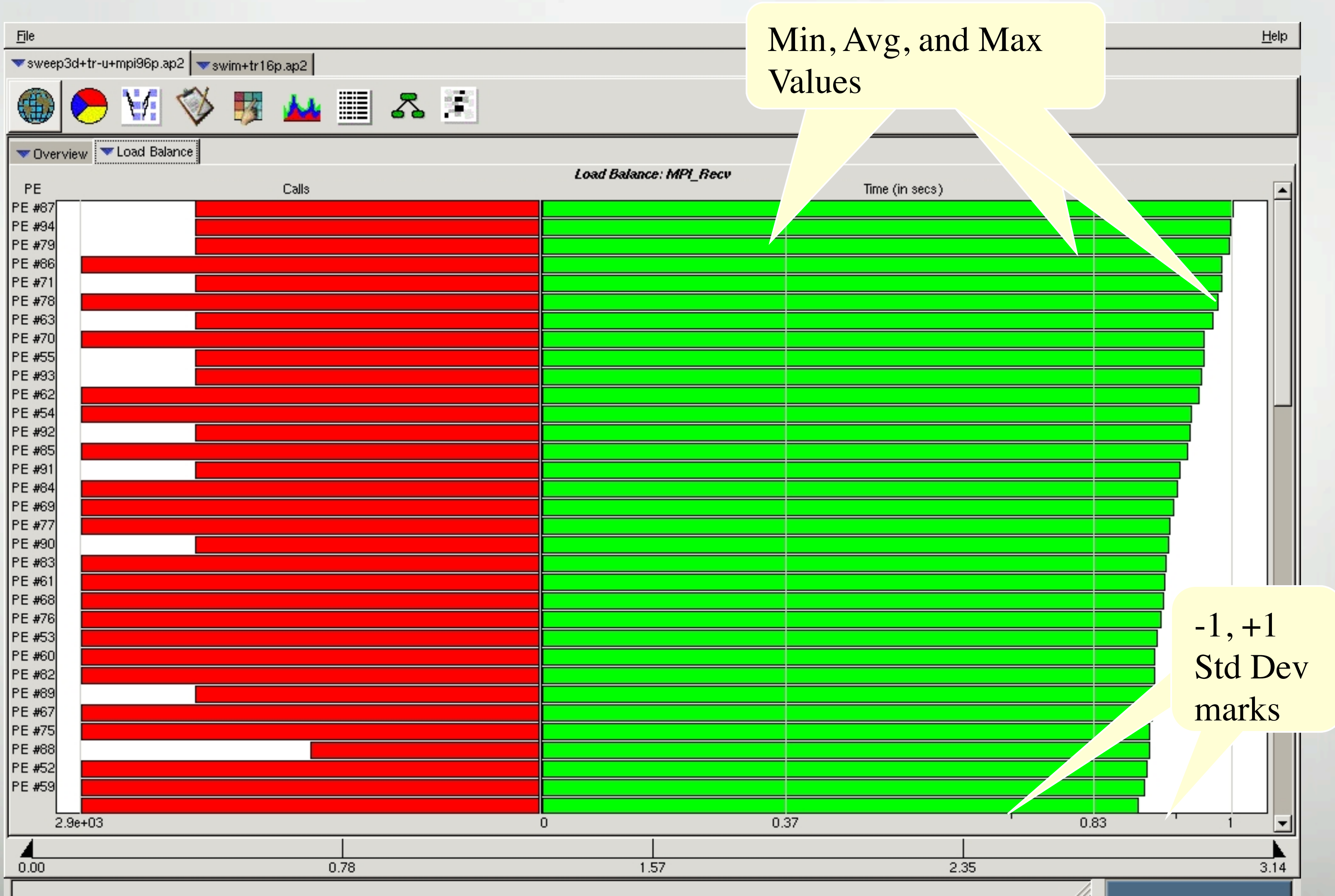
swim+iompi+1566td.ap2 | T+hw1+swp+io+mpi+48p.ap2

Overview | Function

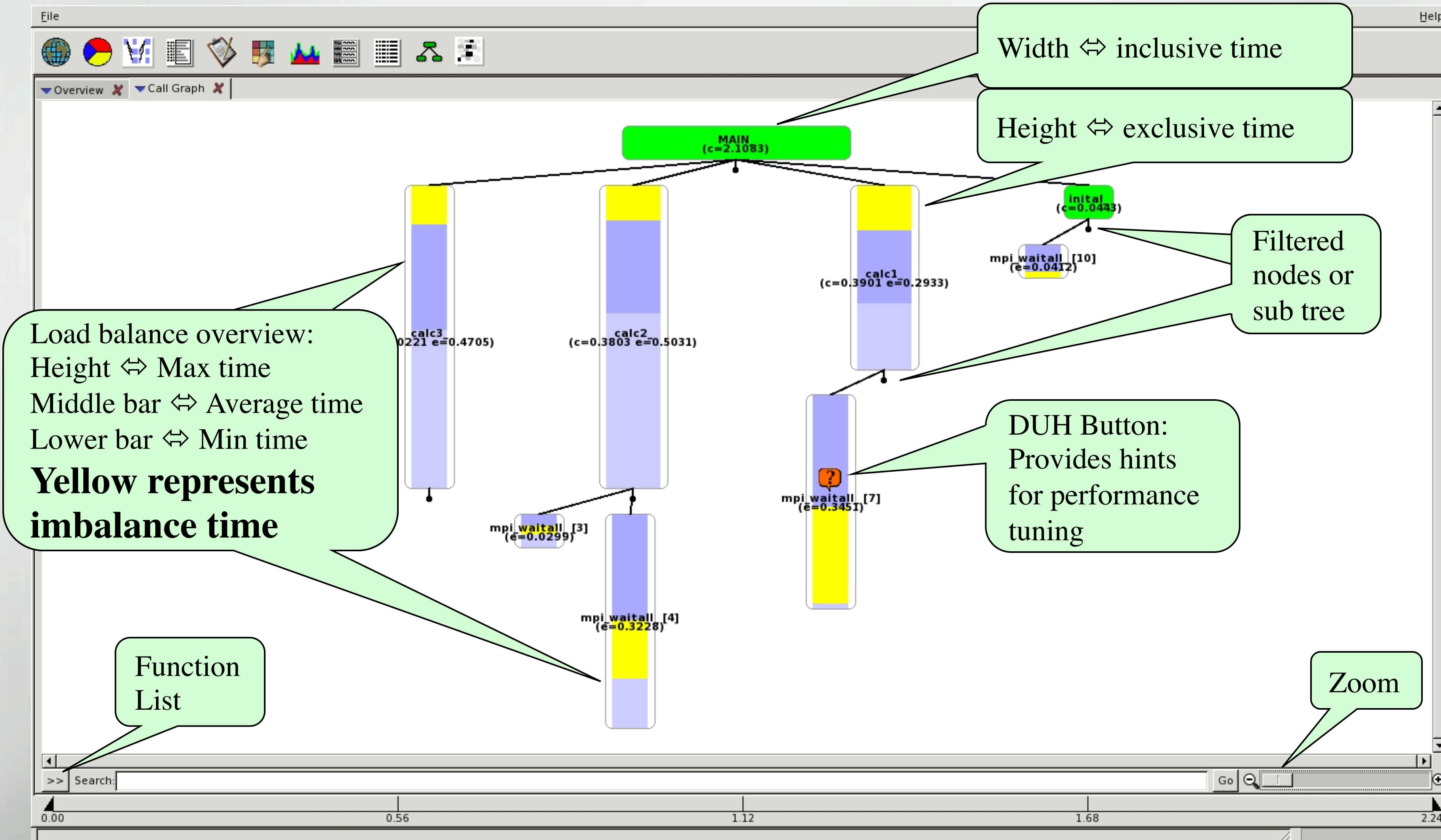
Time	Percent	Hits	Callsites	Imbalance %	Potential Savings	Function	Line	File
124.175511	63.29	576	1	5.63	0.15	sweep_	116	/us/nid00036/ldr/Apps/sweep3d/sweep.f
40.211774	20.50	118080	1	23.40	0.25	mpi_recv_	0	==NA==
16.319527	8.32	48	1	48.26	0.30	exit	35	/notbackedup/users/rsrel/rs64.REL_1_4_33.060914.Thu/pe/computelibs/glibc/stdlib/exit.c
6.173236	3.15	1536	3	50.00	0.12	mpi_allreduce_	0	==NA==
2.760376	1.41	118080	1	17.58	0.01	mpi_send_	0	==NA==
2.250029	1.15	576	1	2.62	0.00	source_	18	/us/nid00036/ldr/Apps/sweep3d/source.f
1.984620	1.01	144	1	2.59	0.00	mpi_barrier_	0	==NA==
0.867359	0.44	192	2	2.47	0.00	mpi_bcast_	0	==NA==
0.416231	0.21	576	1	2.60	0.00	flux_err_	17	/us/nid00036/ldr/Apps/sweep3d/flux_err.f
0.382130	0.19	118080	2	10.98	0.00	snd_real_	135	/us/nid00036/ldr/Apps/sweep3d/mpi_stuff.f
0.237772	0.12	309	1	95.76	0.07	fwrite	36	/notbackedup/users/rsrel/rs64.REL_1_4_33.060914.Thu/pe/computelibs/glibc/libio/iofwrite.c
0.185067	0.09	118080	2	17.30	0.00	rcv_real_	164	/us/nid00036/ldr/Apps/sweep3d/mpi_stuff.f
0.067832	0.03	48	1	4.56	0.00	initialize_	42	/us/nid00036/ldr/Apps/sweep3d/initialize.f
0.059407	0.03	48	1	4.99	0.00	initxs_	77	/us/nid00036/ldr/Apps/sweep3d/initialize.f
0.041371	0.02	48	1	23.84	0.00	inner_	72	/us/nid00036/ldr/Apps/sweep3d/inner.f
0.023948	0.01	8	1		0.00	fputc	35	/notbackedup/users/rsrel/rs64.REL_1_4_33.060914.Thu/pe/computelibs/glibc/libio/fputc.c
0.016902	0.01	68	1		0.00	getc	36	/notbackedup/users/rsrel/rs64.REL_1_4_33.060914.Thu/pe/computelibs/glibc/libio/getc.c
0.008104	0.00	4992	2	28.14	0.00	octant_	17	/us/nid00036/ldr/Apps/sweep3d/octant.f
0.002457	0.00	576	1	18.79	0.00	global_real_max_	321	/us/nid00036/ldr/Apps/sweep3d/mpi_stuff.f
0.002083	0.00	48	1	69.55	0.00	MAIN_	72	/us/nid00036/ldr/Apps/sweep3d/driver.f
0.001588	0.00	576	1	39.10	0.00	global_int_sum_	373	/us/nid00036/ldr/Apps/sweep3d/mpi_stuff.f
0.001393	0.00	48	1	10.23	0.00	inner_auto_	69	/us/nid00036/ldr/Apps/sweep3d/inner_auto.f
0.000982	0.00	48	1	97.74	0.00	task_init_	24	/us/nid00036/ldr/Apps/sweep3d/mpi_stuff.f
0.000739	0.00	384	2	27.87	0.00	global_real_sum_	347	/us/nid00036/ldr/Apps/sweep3d/mpi_stuff.f
0.000662	0.00	2	1		0.00	fopen	106	/notbackedup/users/rsrel/rs64.REL_1_4_33.060914.Thu/pe/computelibs/glibc/libio/iofopen.c
0.000499	0.00	48	1	7.54	0.00	initsnc_	175	/us/nid00036/ldr/Apps/sweep3d/initialize.f

0.00 1.15 2.30 3.45 4.61

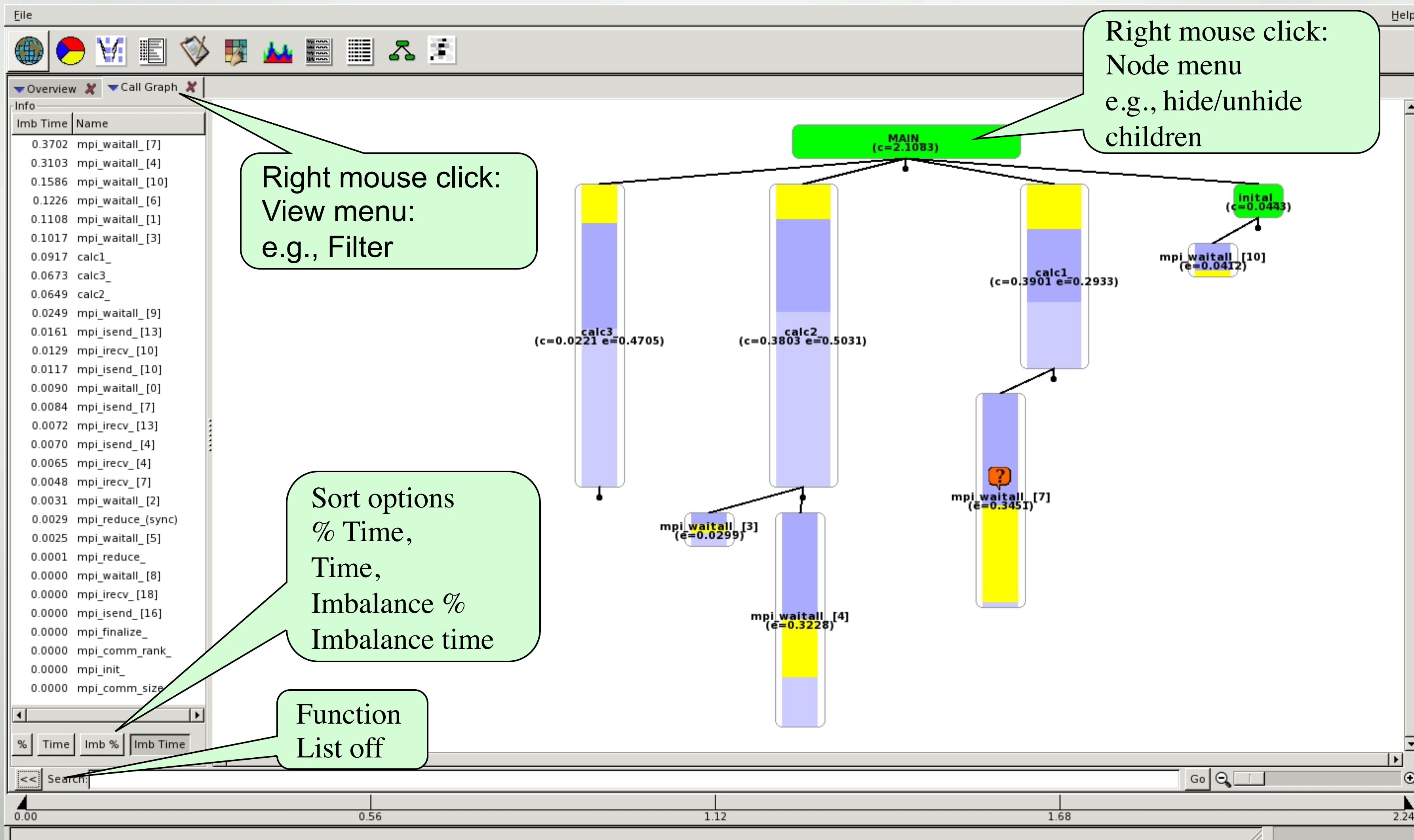
Load Balance View (Aggregated from Overview)



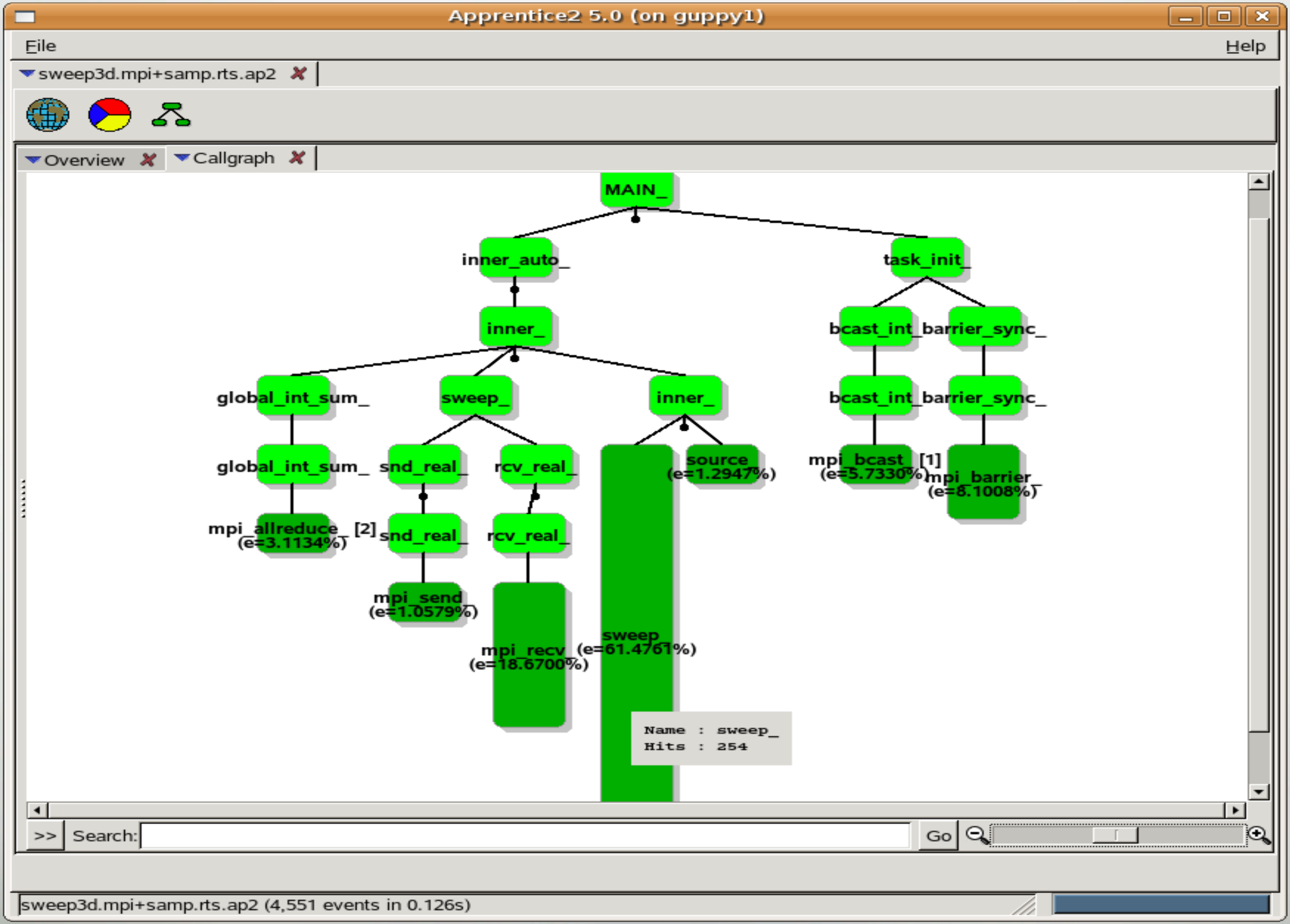
Call Tree View



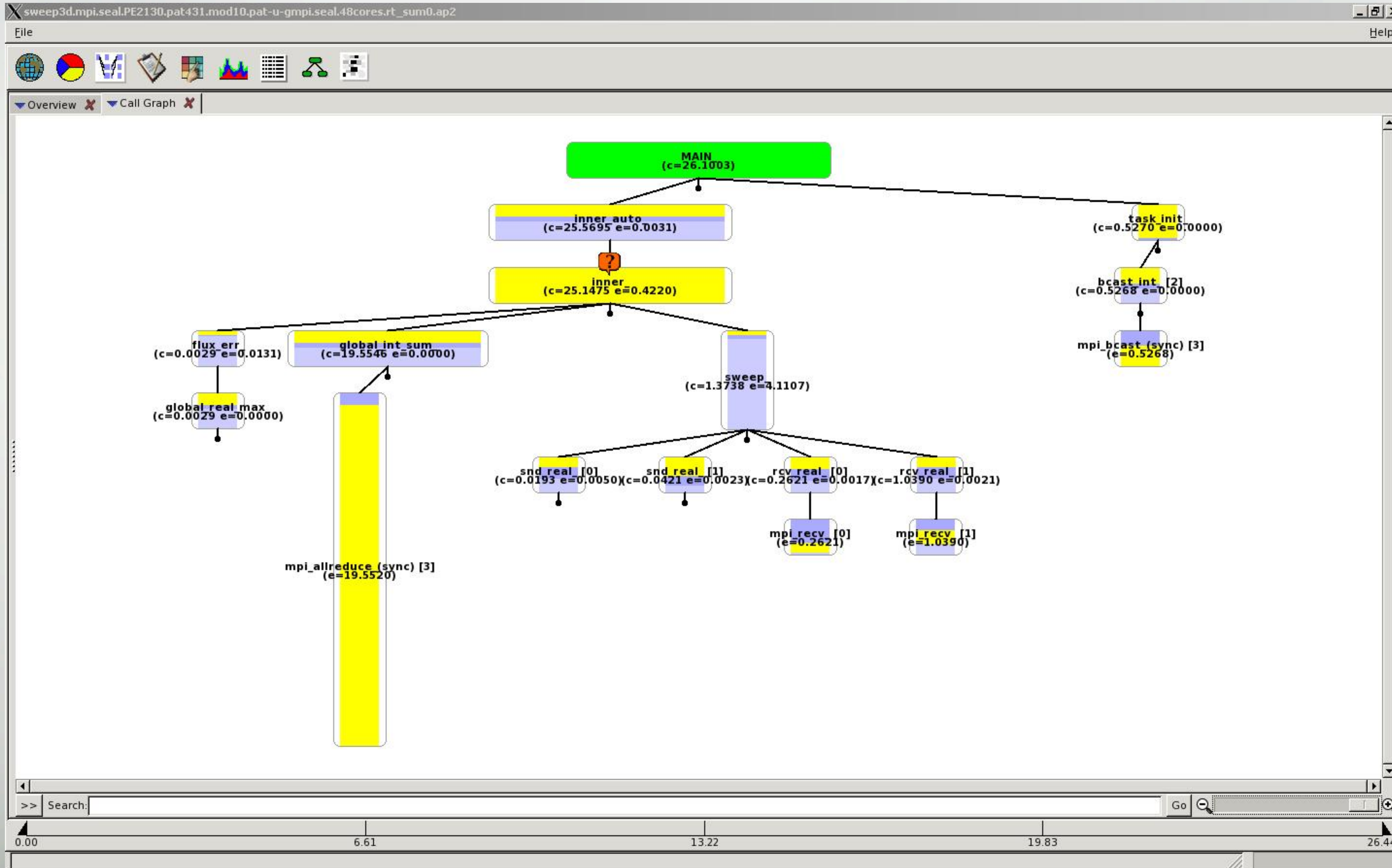
Call Tree View – Function List



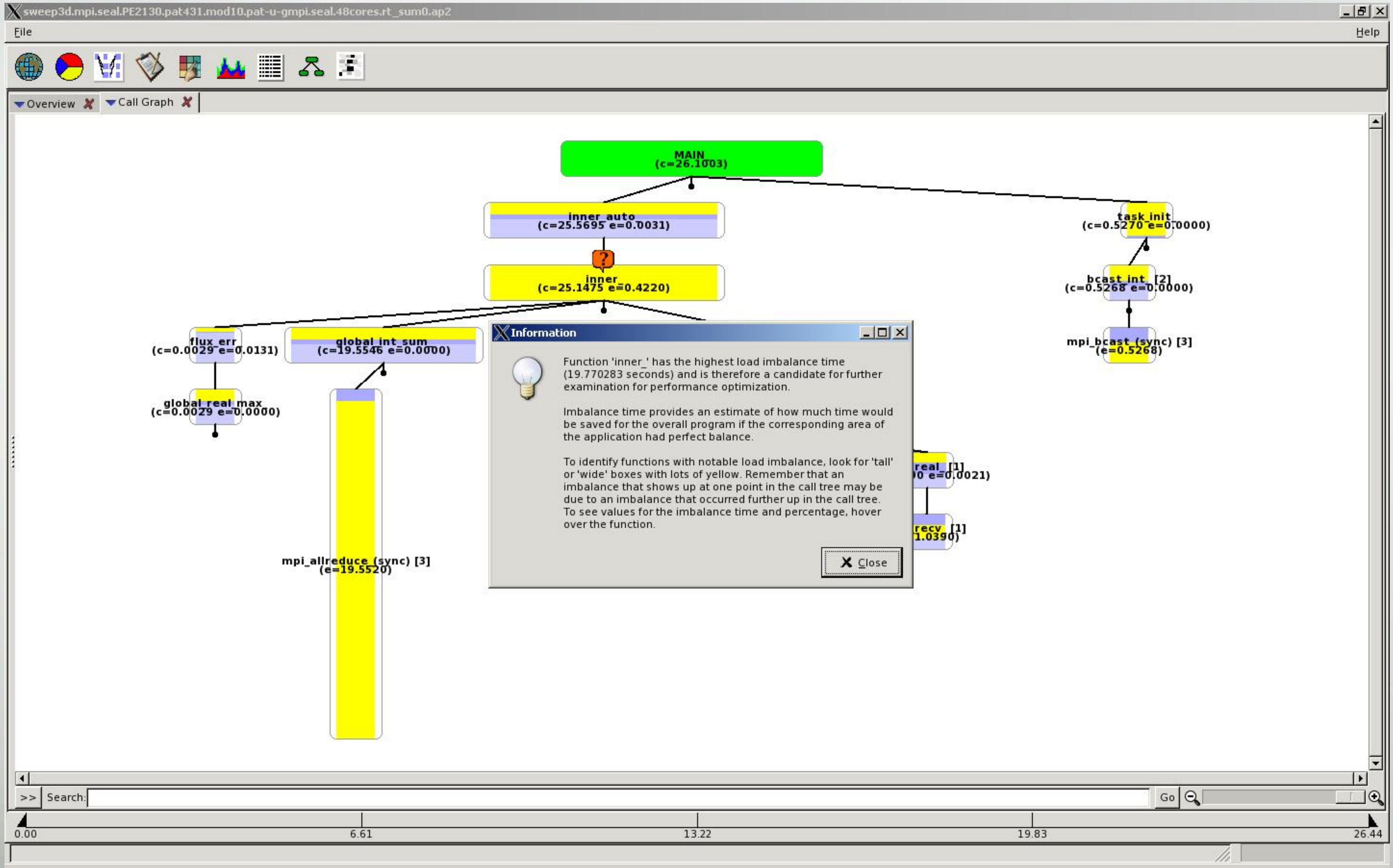
Apprentice² Call Tree View of Sampled Data



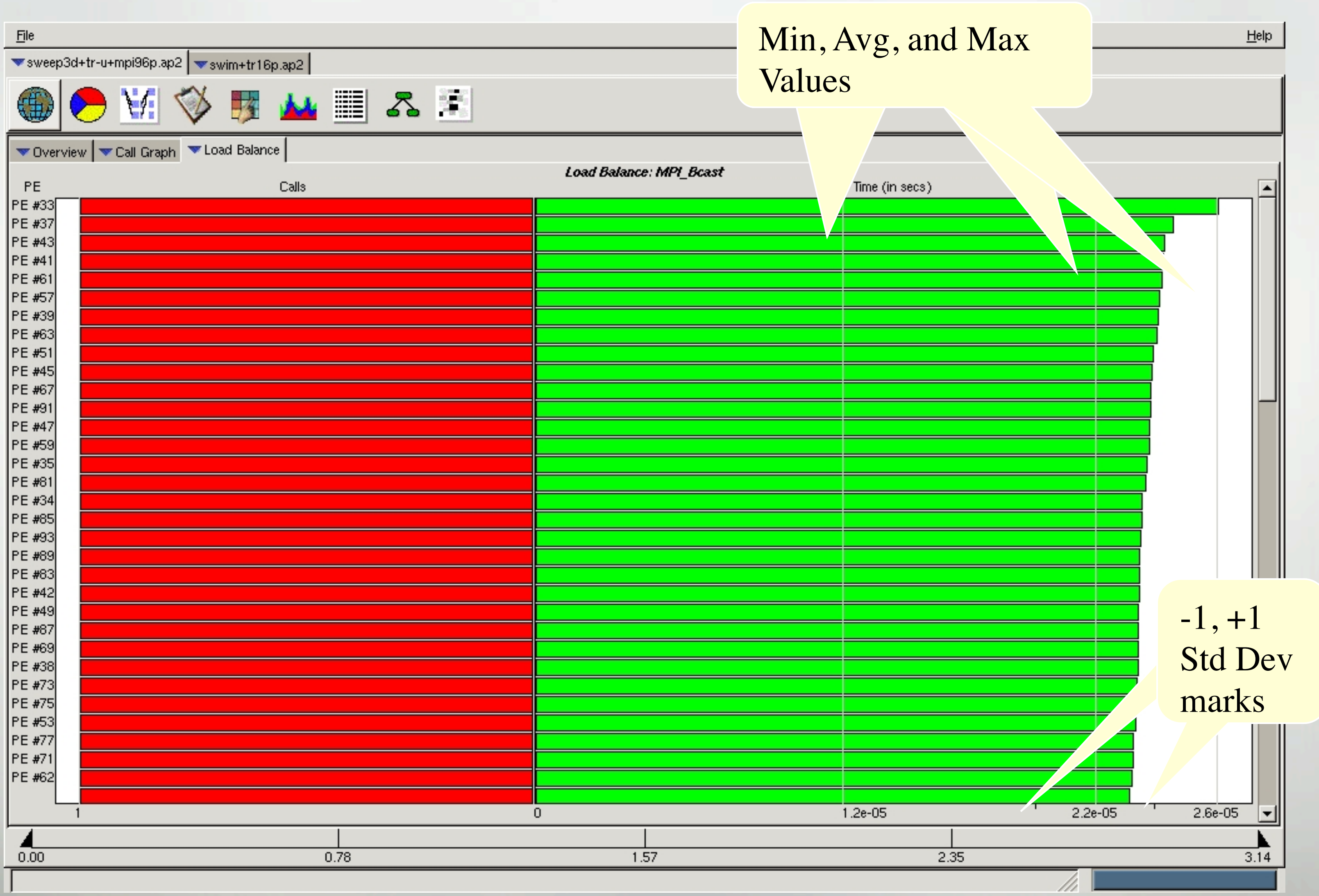
Call Tree Visualization



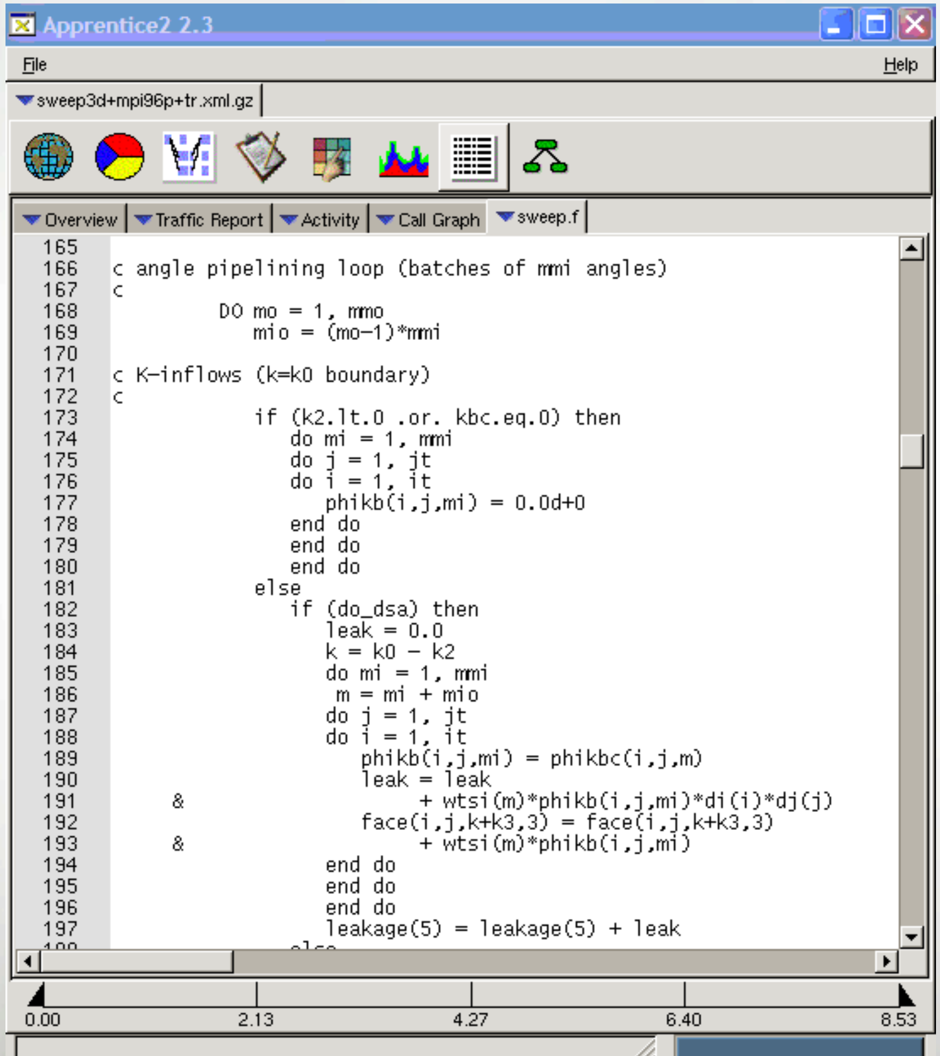
Discrete Unit of Help (DUH Button)



Load Balance View (from Call Tree)



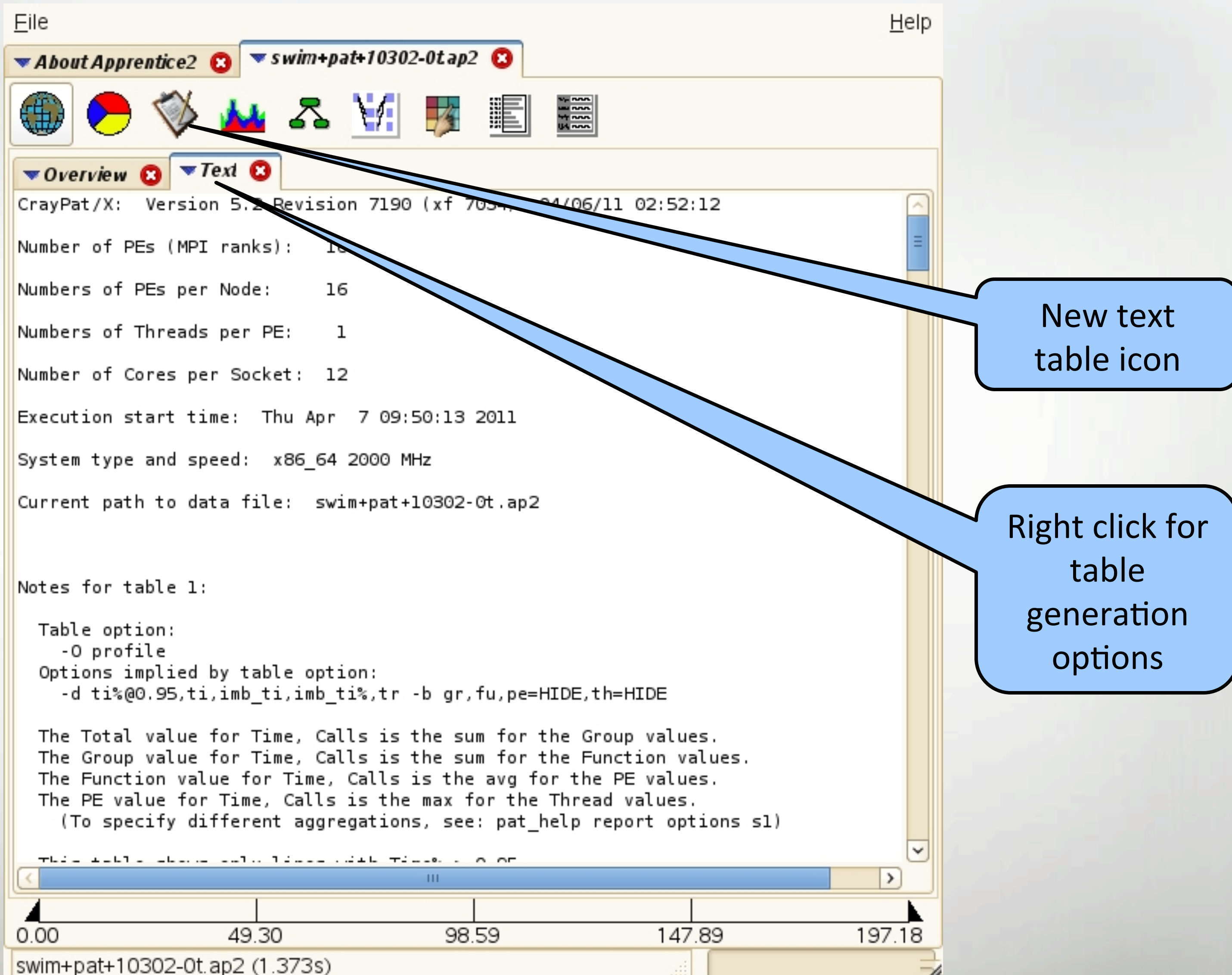
Source Mapping from Call Tree



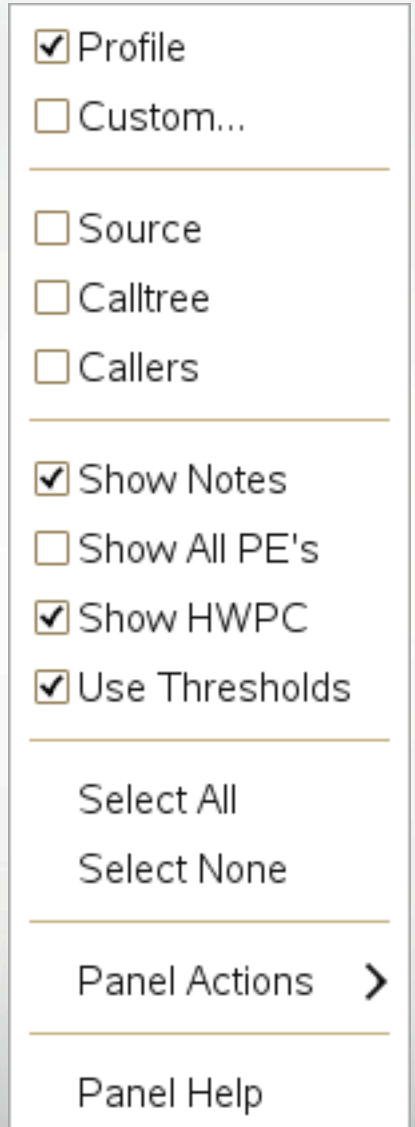
pat_report Tables in Cray Apprentice2

- Complimentary performance data available in one place
- Drop down menu provides quick access to most common reports
- Ability to easily generate different views of performance data
- Provides mechanism for more in depth explanation of data presented

Example of pat_report Tables in Cray Apprentice2



Generating New pat_report Tables



Memory Hierarchy Thresholds

- **TLB utilization: < 90.0%**
  - Measures how well the memory hierarchy is being utilized with regards to TLB
  - This metric depends on the computation being single precision or double precision
    - A page has 4 Kbytes. So, one page fits 512 double precision words or 1024 single precision words
  - TLB utilization < 1 indicates that not all entries on the page are being utilized between two TLB misses
- **Cache utilization: < 1 (D1 or D1+D2)**
  - A cache line has 64 bytes (8 double precision words or 16 single precision words)
  - **Cache utilization < 1** indicates that not all entries on the cache line are being utilized between two cache misses
- **D1 cache hit (or miss) ratios: < 90% (> 10%)**
- **D1 + D2 cache hit (or miss) ratios: < 92% (> 8%)**
  - D1 and D2 caches on the Opteron are complementary
  - This metric provides a view of the Total Cache hit (miss) ratio

Running on Raven

Raven

- Cray XE6
- Runs CLE 4.0.UP00
- 20 Compute nodes
  - 2 MC12 chips (24 cores per node)
  - 1.9 GHz
  - 32 GB memory
- Lustre parallel filesystem
- Usage information available at:
  - https://partners.cray.com/

Purpose of Raven

- Allow users to familiarize themselves with Cray XE6 system
- Aid in port applications to Cray XE6
- Provide platform for the optimization and tuning of Cray XE6 applications

- System is relatively small
- Shared by many users
- Job execution time limits in place
- Limit large jobs (15-20 compute nodes) to 10 minutes or less
- Small jobs (1 or 2 nodes) can run longer (more than an hour is ok)
- xtnodestat shows current system layout

Software Access on Raven

- To see list of modulefiles loaded by default:
  - `% module list`
- To see list of available modulefiles :
  - `% module avail`
- To swap a modulefile
  - `% module swap PrgEnv-pgi PrgEnv-cray`
- `To load the performance tools`
  - `% module load perftools`

Programming Environment on Raven

- Cray XE6 systems use compiler driver scripts:
  - Automatically reference MPI, PGAS, SHMEM
  - Should be used with all programming environments (Cray, PGI, GNU, Intel)
    - `% ftn –h profile_generate –o test1 test1.f`
    - `cc –h profile_generate –o test2 test2.c`

- Compilers require you to target a processor
  - xtpe-mc12 modulefile is loaded by default
    - Cray and PGI compilers target MC by default

Running on Raven

- Name for Lustre filesystem is `/lus/scratch`
  - `Create directory here using your login`

- `Batch scheduler is PBS`

- `Job queue names available:`
  - small walltime=6hrs, nodelimit=4 nodes
  - medium walltime=6hrs, nodelimit=8 nodes
  - large walltime=6hrs, nodelimit=16 nodes
  - long walltime=24hrs, nodelimit=4 nodes

```
% qsub -l mppwidth=96 -l mppnppn=24 -q large testrun.pbs
```

Job size

# of processes per node (1 – 24)

Sample PBS Job Script

```
#PBS -l mppwidth=24
#PBS -l mppnppn=4
#PBS -l walltime=00:10:00
#PBS -j oe

cd $PBS_O_WORKDIR
aprun -n 24 -N 4 ./mpi_app
```

Questions

??