BLUE WATERS

SUSTAINED PETASCALE COMPUTING

Performance Tools on Blue Waters

Manisha Gajbe + others

GREAT LAKES CONSORTIUM

FOR PETASCALE COMPUTATION

Performance Tools

- **Cray Performance Tools**
- Perfsuite (NCSA)
- TAU

Topics

- Cray performance tools overview
- Steps to using the tools
- Performance measurement on the Cray XE system
- Using HW performance counters
  - Access to Northbridge: L3 and memory
- Profiling applications
- Visualization of performance data through pat_report
- Visualization of performance data through Cray Apprentice2
- MPI Rank Order
- PerfSuite (NCSA)
- TAU : Tuning and Analysis Utilities
- Congestion Protection and Balanced Injection

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

Design Goals

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

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, OpenACC
  - Access to network counters
  - Minimal program perturbation

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

- To access software:
  - module load perftools

The Cray Performance Analysis Framework

CrayPat

- Instrumentation of optimized code
- No source code modification required
- Data collection transparent to the user
- Text-based performance reports
- Derived metrics
- Performance analysis

Cray Apprentice2

- Performance data visualization tool
- Call tree view
- Source code mappings

Steps To Using Tools

Application Instrumentation with pat_build

- pat_build is a stand-alone utility that 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

Sampling with Line Number information

heidi@limited: /h/heidi — ssh — 81×26

```
Table 2:  Profile by Group, Function, and Line

 Samp%    |   Samp   | Imb.    |  Imb.   |Group
          |          | Samp    | Samp%   | Function
          |          |         |         |  Source
          |          |         |         |   Line
          |          |         |         |    PE=HIDE

 100.0% | 8376.9 |     -- |      -- |Total
|------------------------------------------------------------------------------
|  93.2% | 7804.0 |     -- |      -- |USER
||-----------------------------------------------------------------------------
||  51.7% | 4328.7 |     -- |      -- |calc3_
3|        |        |        |         | heidi/DARPA/cache_util/calc3.do300-ijswap.F
||||---------------------------------------------------------------------------
4|||  15.7% | 1314.4 |  93.6 |   6.8% |line.78
4|||  13.9% | 1167.7 |  98.3 |   7.9% |line.79
4|||  14.5% | 1211.6 |  97.4 |   7.6% |line.80
4|||   1.2% |  103.1 |  26.9 |  21.2% |line.93
4|||   1.1% |   88.4 |  22.6 |  20.8% |line.94
4|||   1.0% |   84.5 |  17.5 |  17.6% |line.95
4|||   1.0% |   86.8 |  33.2 |  28.2% |line.96
4|||   1.3% |  105.0 |  23.0 |  18.4% |line.97
4|||   1.4% |  116.5 |  24.5 |  17.7% |line.98
||||===========================================================================
                                                               144,1         38%
```

Where to Run Instrumented Application

- By default, data files are written to the execution directory
- Default behavior requires file system that supports record locking, such as Lustre (/mnt/snx3/… , /lus/…, /scratch/…,etc.)
  - Can use PAT_RT_EXPFILE_DIR to point to existing directory that resides on a high-performance file system if not execution directory
- Number of files used to store raw data
  - 1 file created for program with 1 – 256 processes
  - √n files created for program with 257 – n processes
  - Ability to customize with PAT_RT_EXPFILE_MAX
- See intro_craypat(1) man page

CrayPat Runtime Options

- Runtime controlled through PAT_RT_XXX environment variables
- 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 (Collect data in detail than aggregate)
  - View trace file with Cray Apprentice[2]

- Request hardware performance counter information:
  - export PAT_RT_HWPC=<HWPC Group>
  - export PAT_RT_ACCPC=<HWPC Group>
  - Can specify events or predefined groups

Predefined Trace Wrappers (-g tracegroup)

blas	Basic Linear Algebra subprograms
caf	Co-Array Fortran (Cray CCE compiler only)
hdf5	manages extremely large data collection
heap	dynamic heap
io	includes stdio and sysio groups
lapack	Linear Algebra Package
math	ANSI math
mpi	MPI
omp	OpenMP API
pthreads	POSIX threads
shmem	SHMEM
sysio	I/O system calls
system	system calls
upc	Unified Parallel C (Cray CCE compiler only)

For a full list, please see pat_build(1) man page

Example Experiments

- > pat_build –O apa
  - Gets you top time consuming routines
  - Lightest-weight sampling

- > pat_build –u –g mpi ./my_program
  - Collects information about user functions and MPI

- > pat_build –w ./my_program
  - Collections information for MAIN
  - Lightest-weight tracing

- > pat_build –g netcdf,mpi ./my_program
  - Collects information about netcdf routines and MPI

pat_report

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

Automatic Profiling Analysis

Why Should I generate a ".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

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

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

Steps to Collecting Performance Data (2)

- You should get a performance file (“<sdatafile>.xf”) or multiple files in a directory <sdatadir>

- Generate report and .apa instrumentation file

```
% pat_report <sdatafile>.xf > sampling_report

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

- Inspect .apa file and sampling report
- Verify if additional instrumentation is needed

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
```

CPU HW Performance Counters

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 (points to PAPI documentation: http://icl.cs.utk.edu/papi/)
  - http://lists.eecs.utk.edu/pipermail/perfapi-devel/2011-January/004078.html

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 23 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

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

19: Prefetchs

20: FP, D1, TLB, MIPS

21: FP, D1, TLB, Stalls

22: D1, TLB, MemBW

23: FP, D1, D2,TLB

default: group 23

Support for L3 cache counters now available.

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

AMD North-Bridge events

L3_CACHE_MISSES:type:core

READ_BLOCK_EXCLUSIVE, READ_BLOCK_SHARED, READ_BLOCK_MODIFY, PREFETCH, ALL

CORE_0, CORE_1 … CORE_7, ALL_CORES

DRAM_ACCESSES:type

DCT0_PAGE_HIT, DCT0_PAGE_MISS, DCT0_PAGE_CONFLICT

same for DCT1, ALL

To see all NB events (start with craynb:::)

```
> aprun papi_native_avail
```

BLUE WATERS
SUSTAINED PETASCALE COMPUTING

GREAT LAKES CONSORTIUM FOR PETASCALE COMPUTATION

Example: HW counter data &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

```
========================================================================
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
========================================================================
```

Profile Visualization with pat_report

pat_report: 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:
…
```

pat_report: Table Notes

```
Notes for table 1:

  Table option:
    -O profile
  Options implied by table option:
    -d ti%@0.95,ti,imb_ti,imb_ti%,tr -b gr,fu,pe=HIDE
  Other options:
    -T

  Options for related tables:
    -O profile_pe.th              -O profile_th_pe
    -O profile+src                -O load_balance
    -O callers                    -O callers+src
    -O calltree                   -O calltree+src

  The Total value for Time, Calls is the sum for the Group values.
  The Group value for Time, Calls is the sum for the Function values.
  The Function value for Time, Calls is the avg for the PE values.
    (To specify different aggregations, see: pat_help report options s1)

  This table shows only lines with Time% > 0.

  Percentages at each level are of the Total for the program.
    (For percentages relative to next level up, specify:
      -s percent=r[elative])
```

pat_report: Additional Information

```
...
Instrumented with:
  pat_build -gmpi -u himenoBMTxpr.x

Program invocation:
  ../bin/himenoBMTxpr+pat.x

Exit Status:  0 for 256 PEs

CPU  Family: 15h  Model: 01h  Stepping: 2

Core Performance Boost:  Configured for   0 PEs
                         Capable    for 256 PEs

Memory pagesize:  4096

Accelerator Model: Nvidia X2090 Memory: 6.00 GB Frequency: 1.15 GHz

Programming environment:  CRAY

Runtime environment variables:
  OMP_NUM_THREADS=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 L3 NB counters with CrayPat

Load CrayPat module

```
$ module load perftools
```

Compile and Instrument as usual:

```
$ cc -o stream stream.c
$ pat_build stream
```

Define counters (up to 4)

```
$ export PAT_RT_PERFCTR=L3_CACHE_MISSES,…
```

Run

```
$ aprun -ss -cc 0 stream+pat
```

Generate the report

```
$ pat_report -O hwpc stream+pat+2313073-25560s.xf
```

pat_report excerpts for L3

Limited derived metrics at the moment. Can be used to calculate bandwidth to main memory.

```
Table 3:  Program HW Performance Counter Data

 L3_CACHE_MISSES:ALL:CORE_0  | L3_CACHE_MISSES  |Total

                  207185723 |        207502576 |Total

Table 2:

 L3_CACHE_MISSES:ALL:CORE_0  | L3_CACHE_MISSES  |PE=SHOW

                  206606606 |        206887763 |Total
|-------------------------------------------------------
|                  207128361 |        207421774 |pe.0
|                  206084850 |        206353752 |pe.1
```

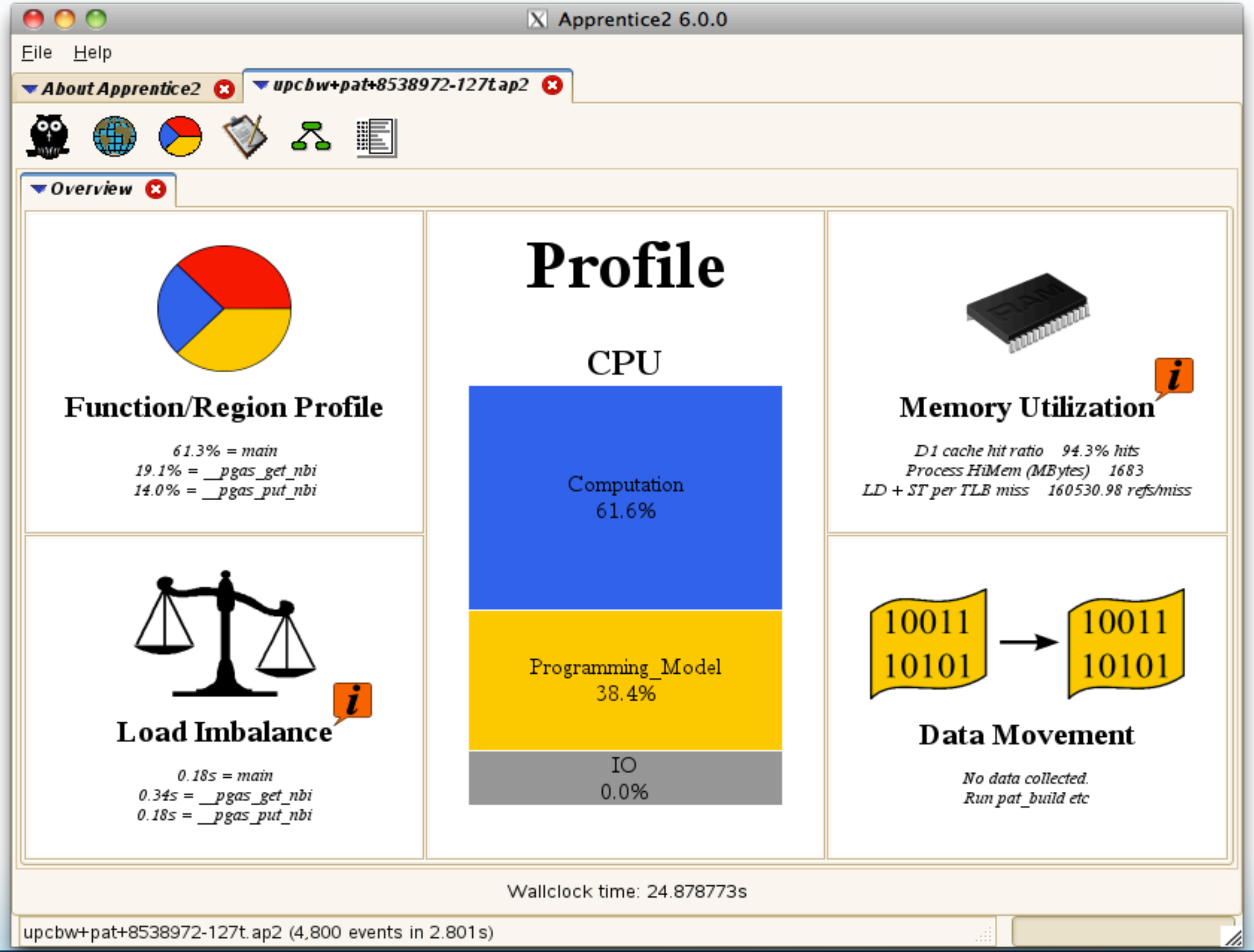
Profile Visualization with Cray Apprentice2

Cray Apprentice^2

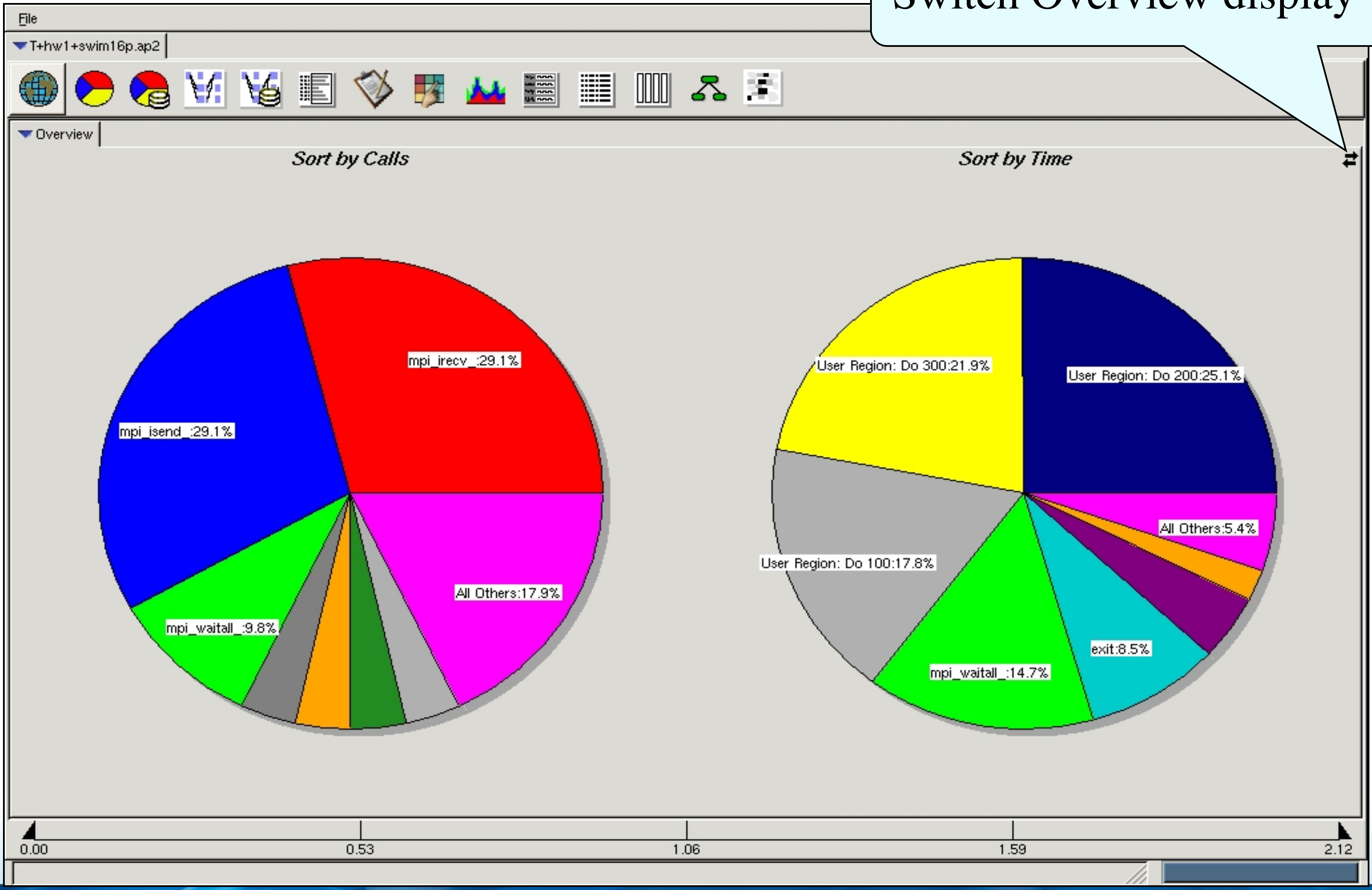
- Call graph profile
- Communication statistics
- Time-line view
  - Communication
  - I/O
- Activity view
- Pair-wise communication statistics
- Text reports
- Source code mapping
- Runs on login node
- Supported on Mac OS and Windows also

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

Application Performance Summary

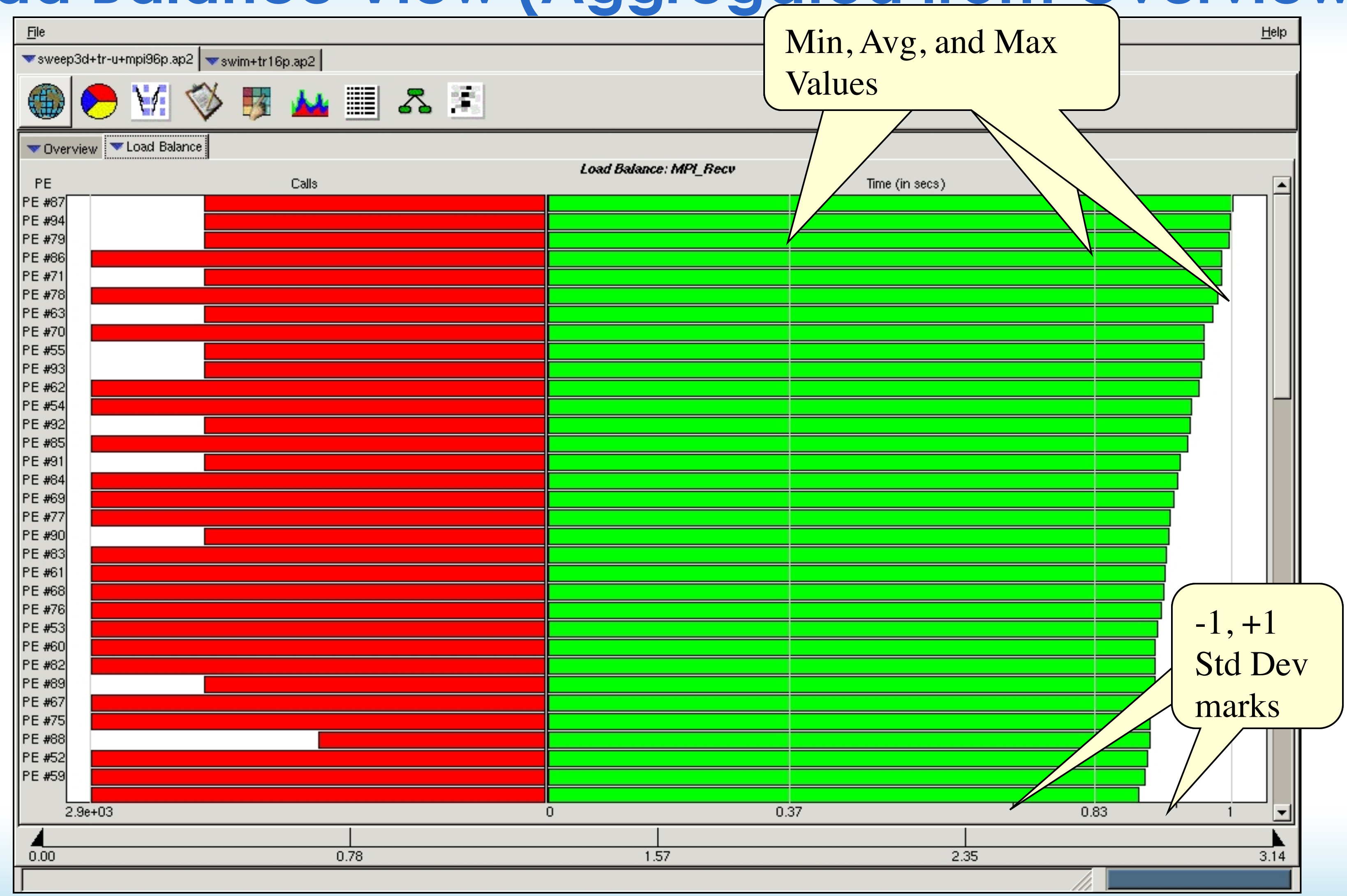


Statistics Overview



Switch Overview display

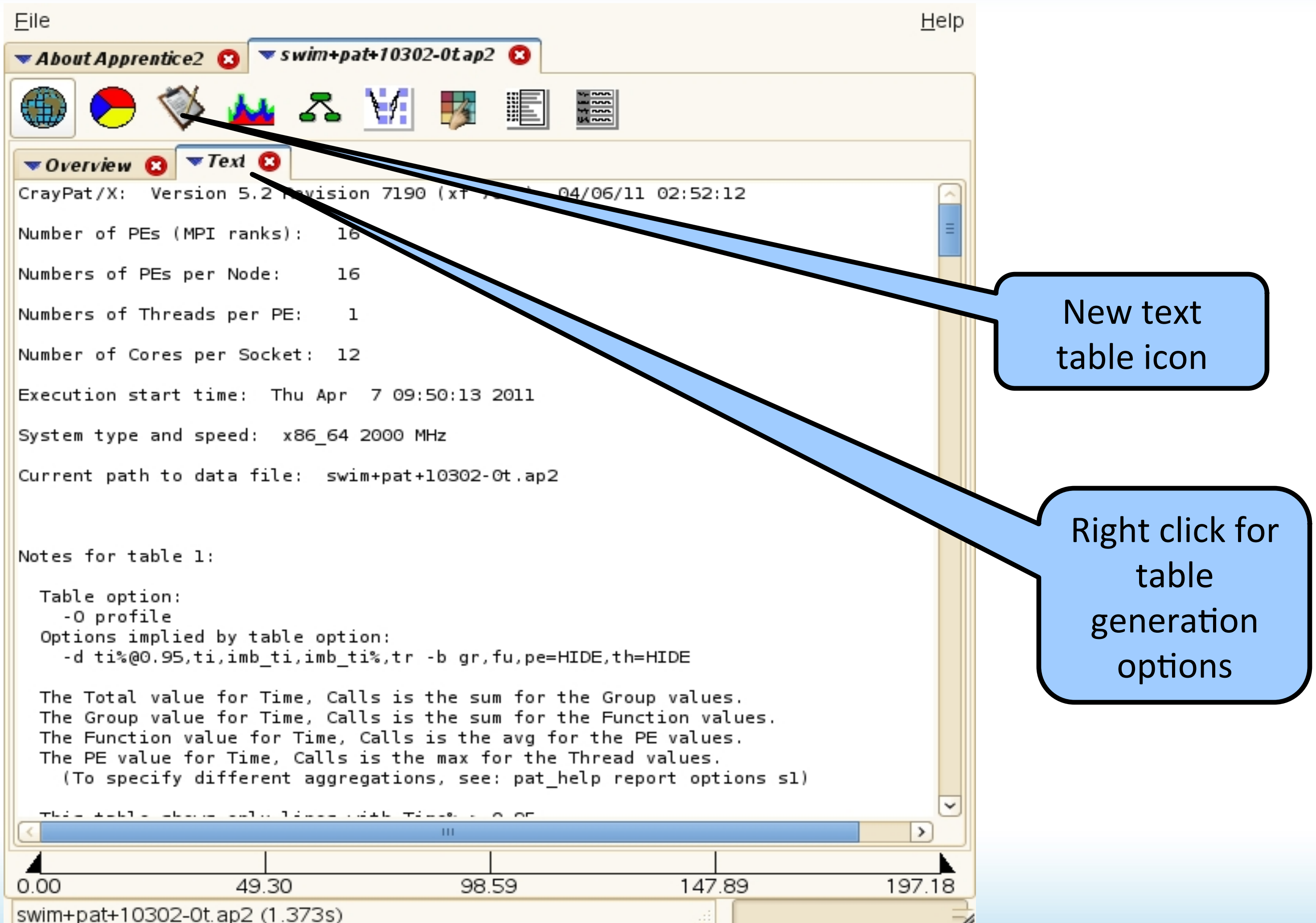
Load Balance View (Aggregated from Overview)



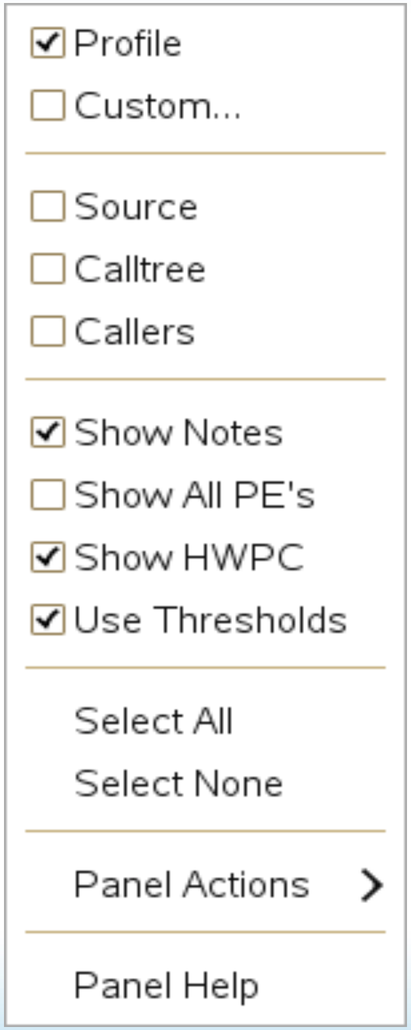
pat_report Tables in Cray Apprentice2

- 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

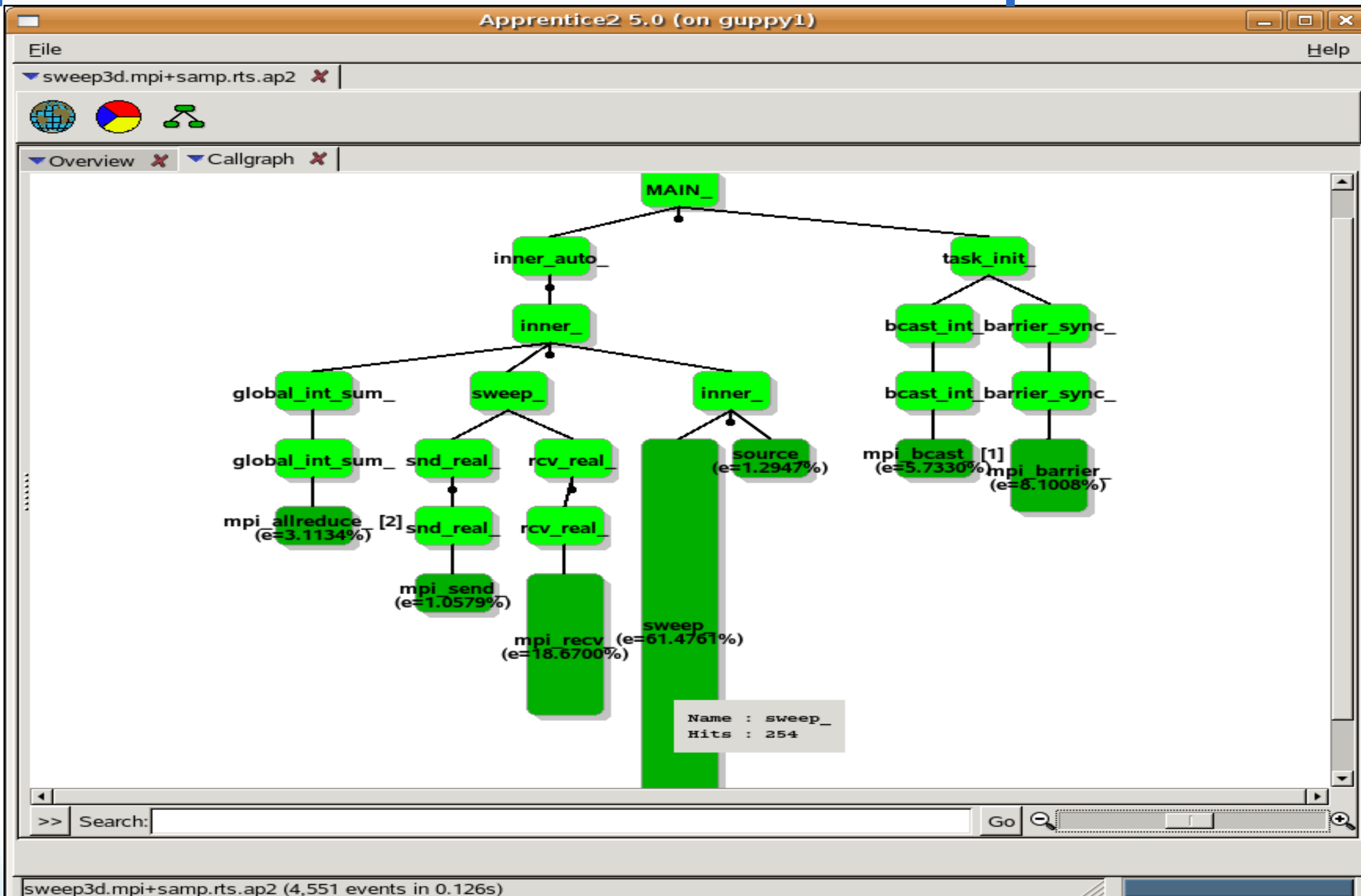
Example of pat_report Tables in Cray Apprentice2



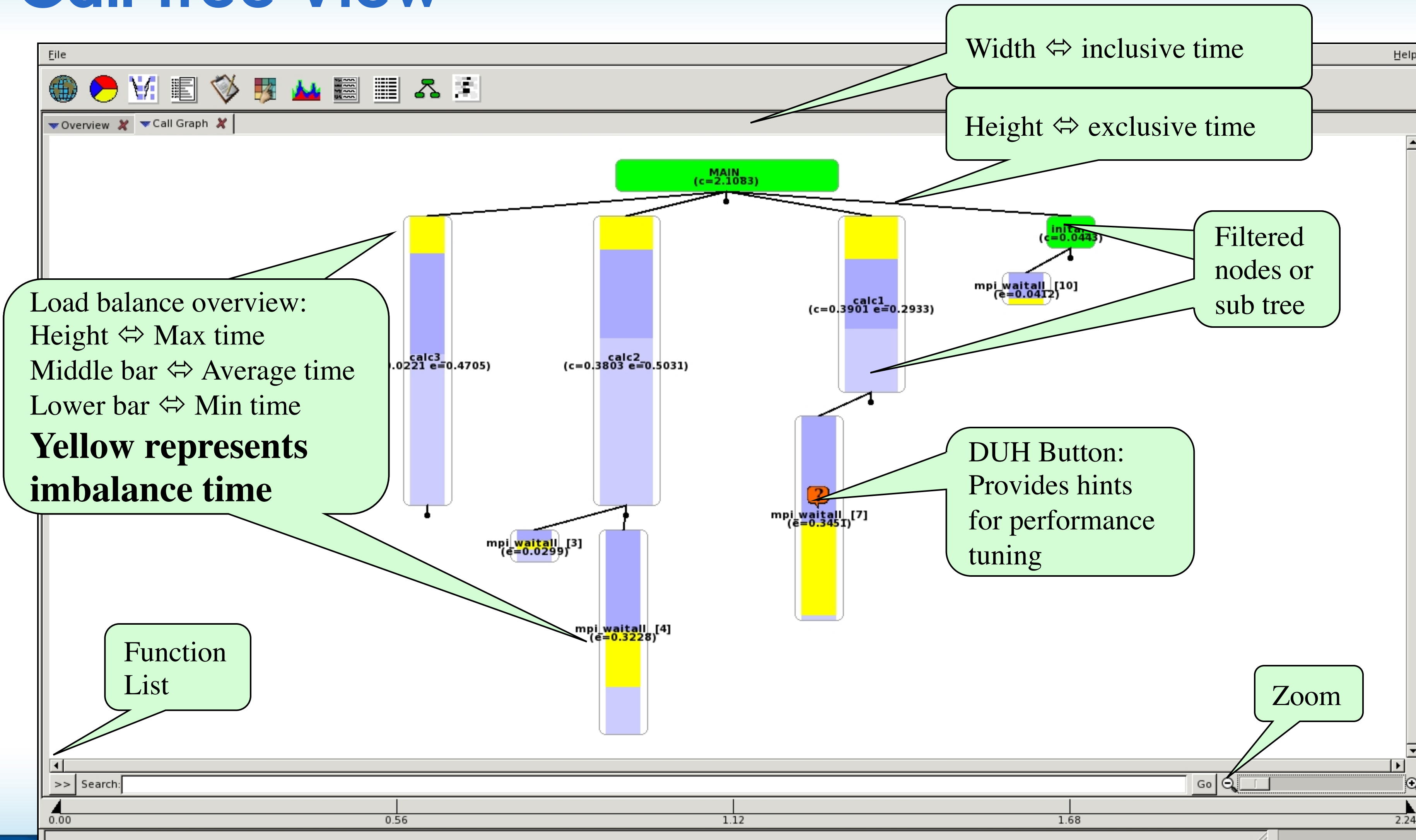
Generating New pat_report Tables



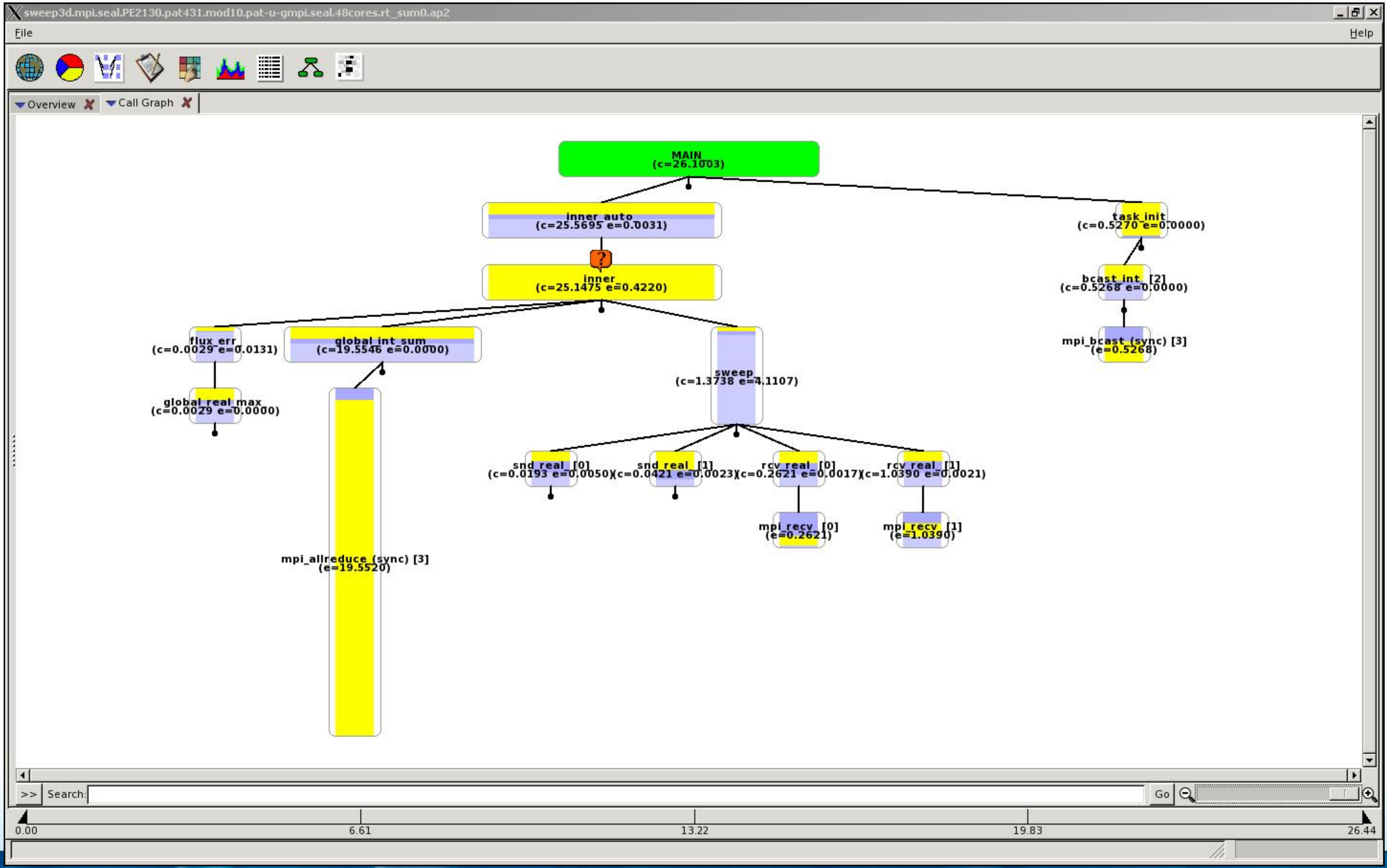
Apprentice2 : Calltree View of Sampled Data



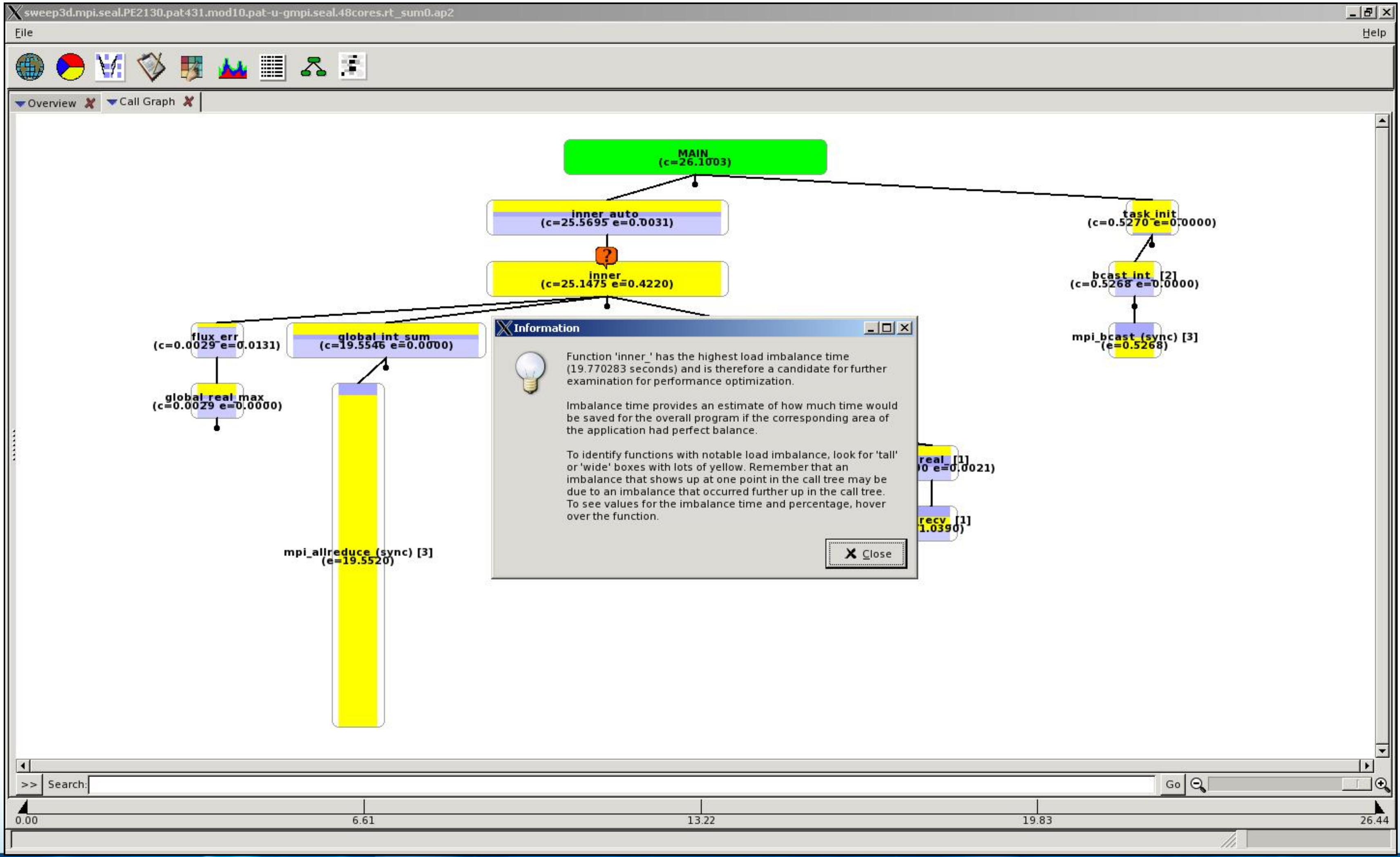
Call Tree View



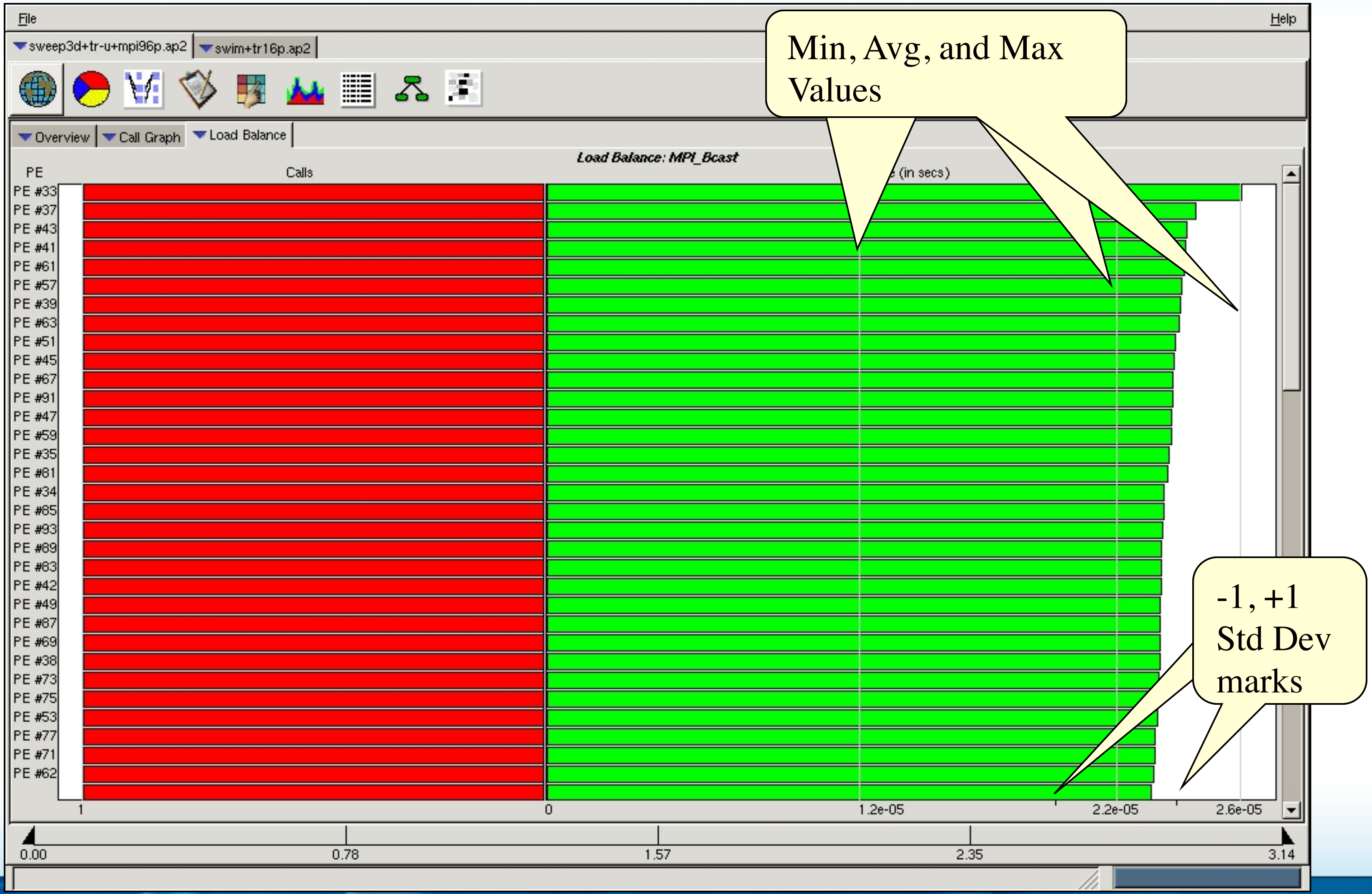
Call Tree Visualization



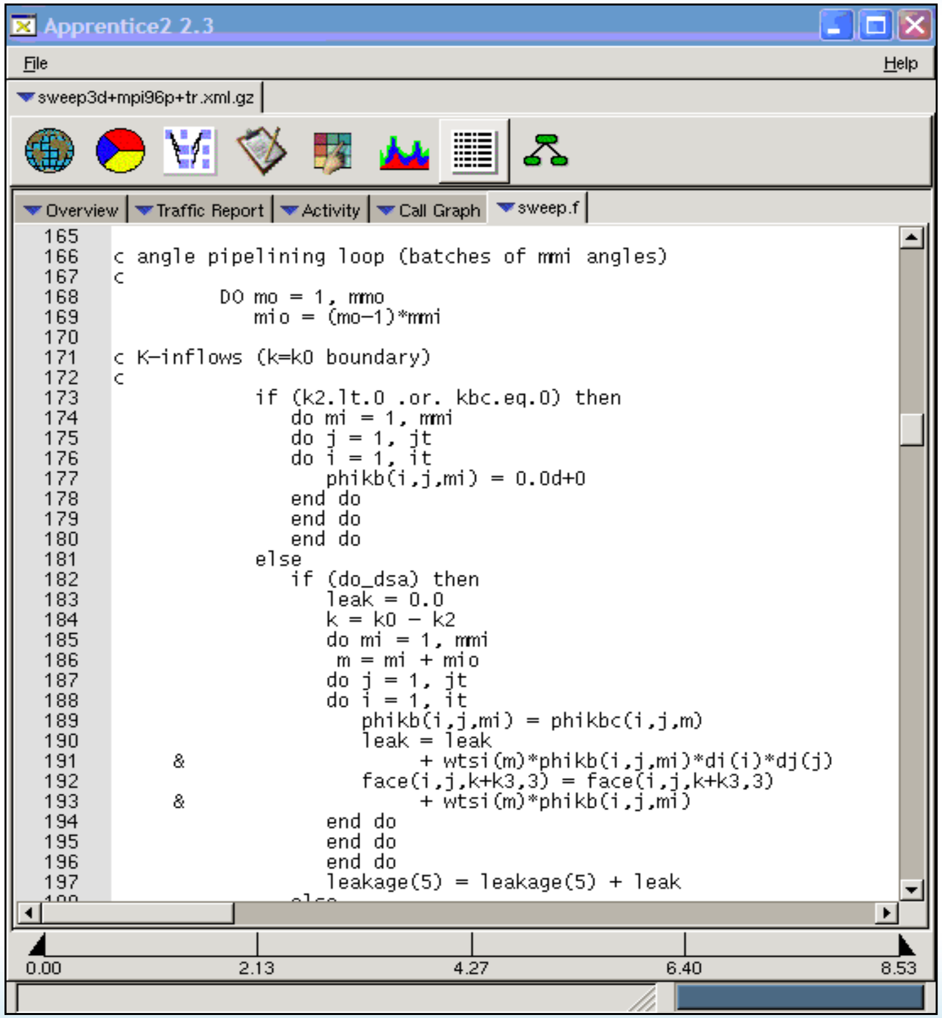
Discrete Unit of Help (DUH Button)



Load Balance View (from Call Tree)

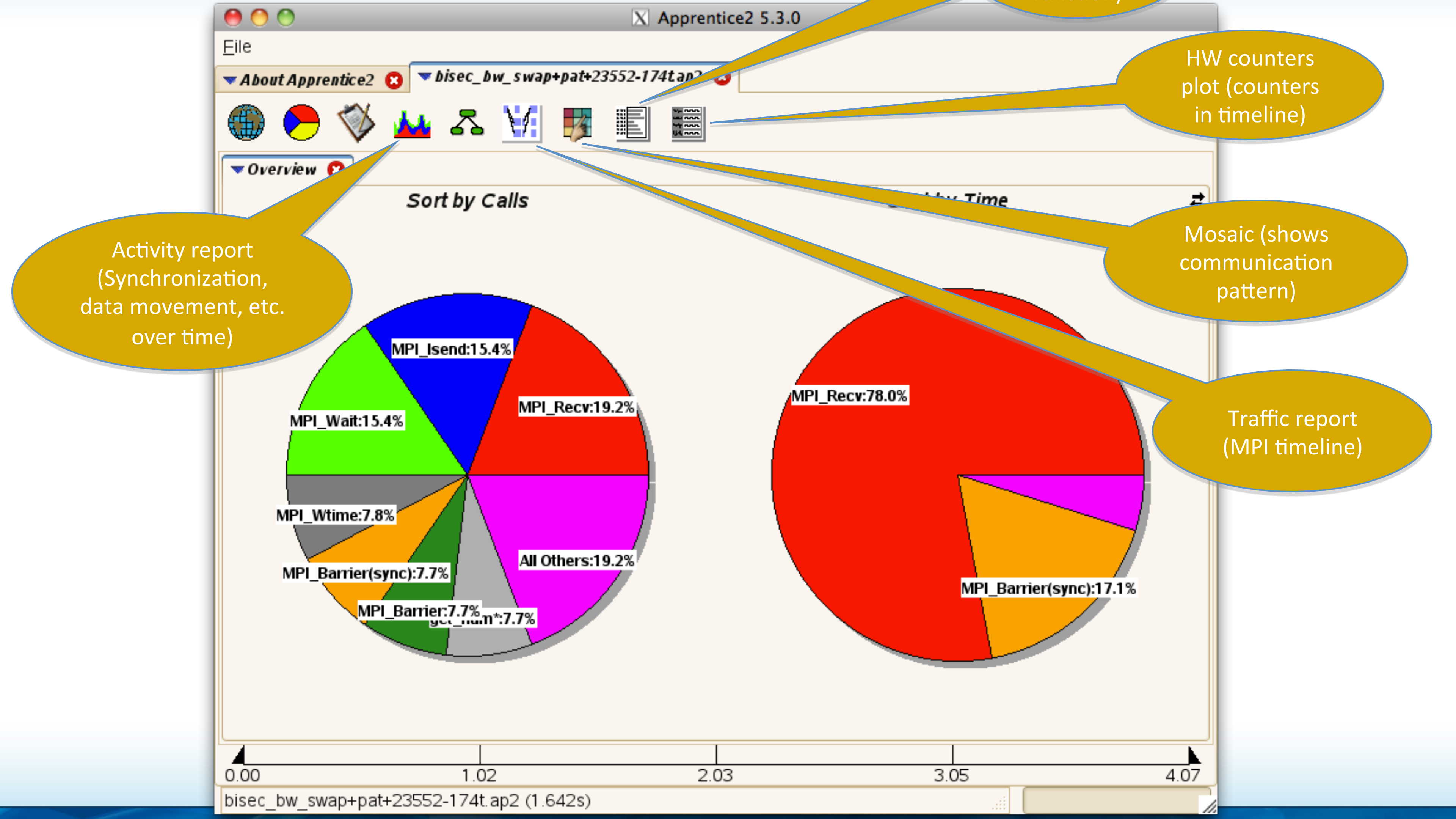


Source Mapping from Call Tree

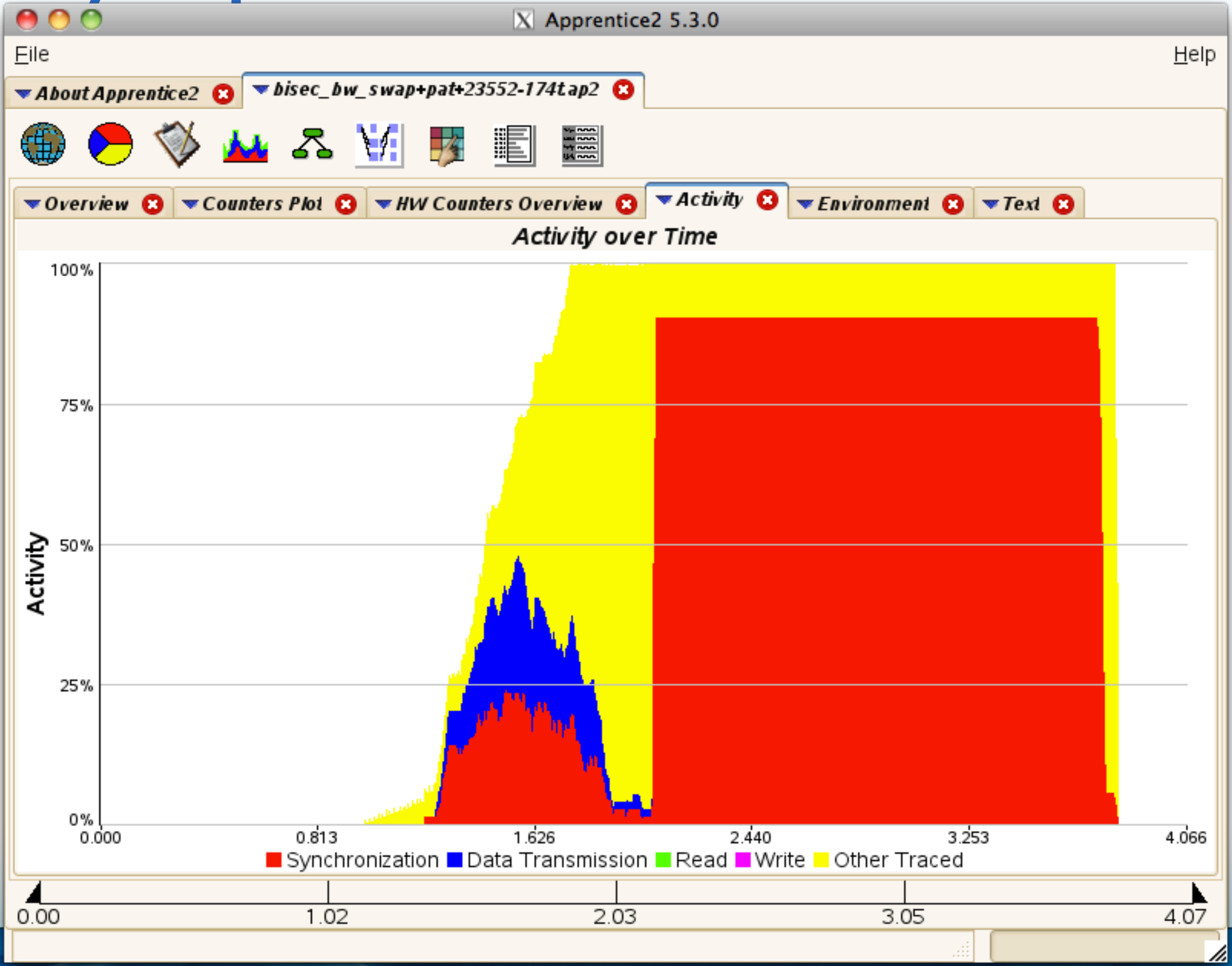


Full Trace Visualization with Cray Apprentice2

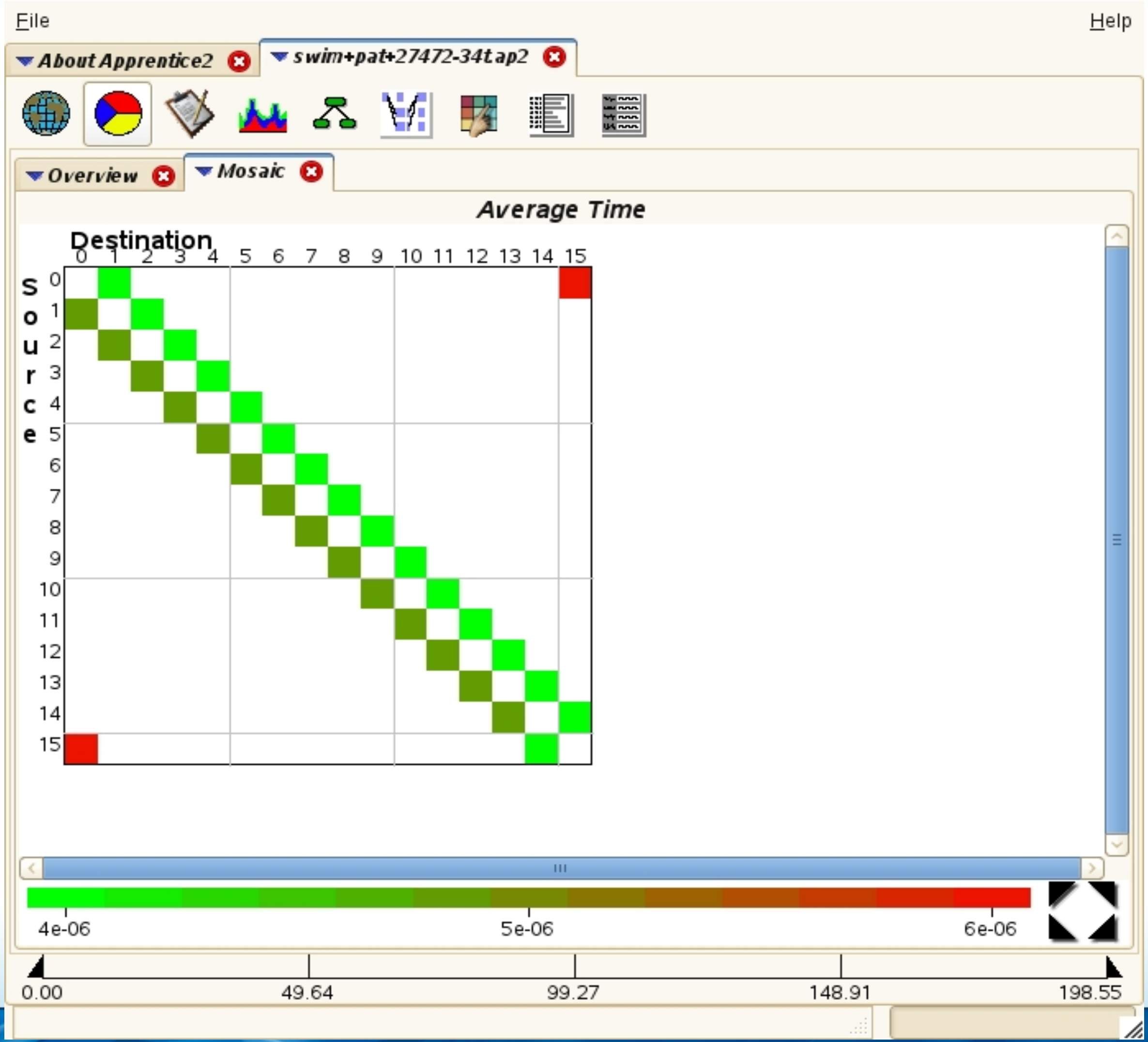
Trace Overview – Additional V



Activity Report

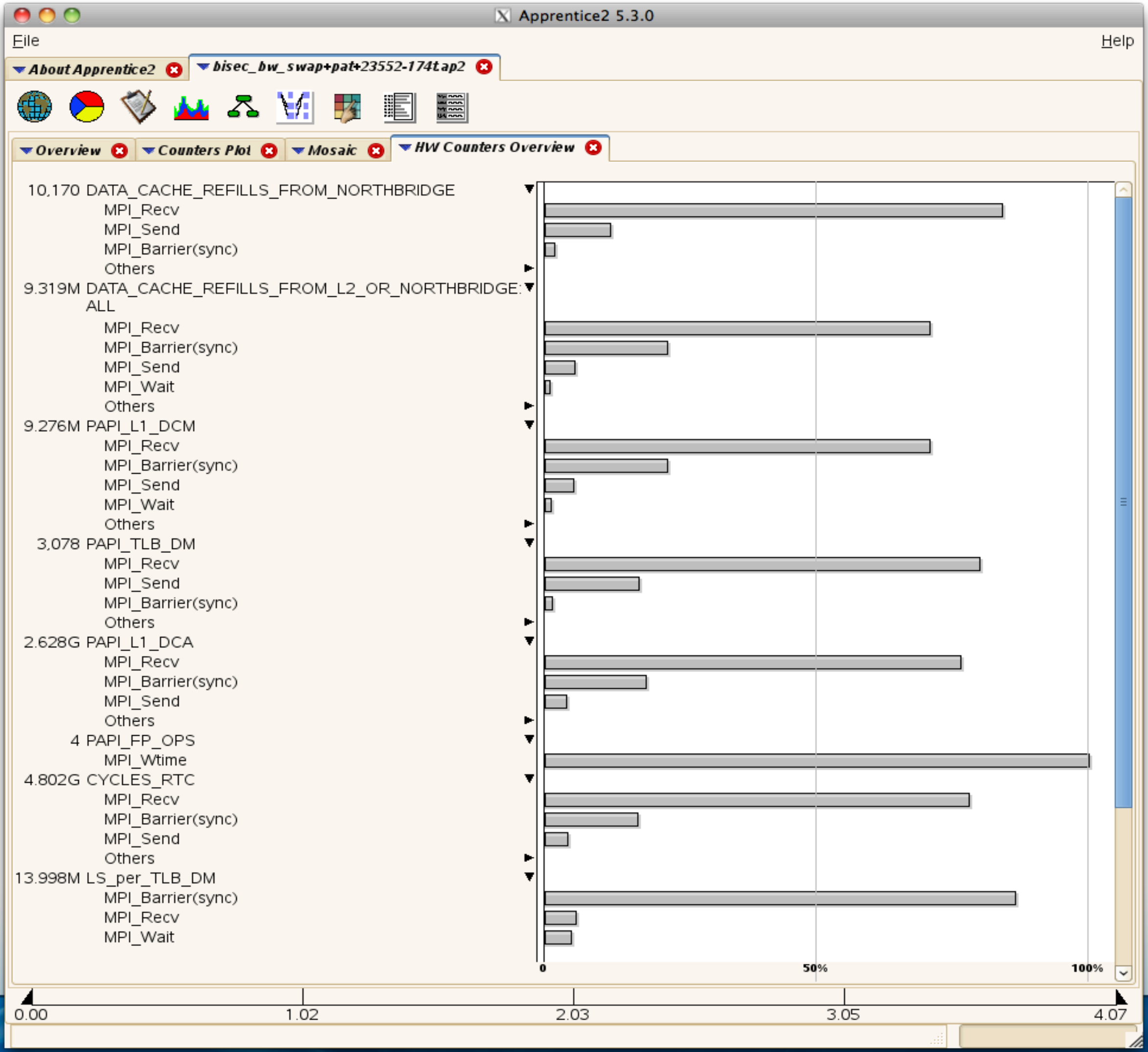


Mosaic View – Shows Communication Pattern

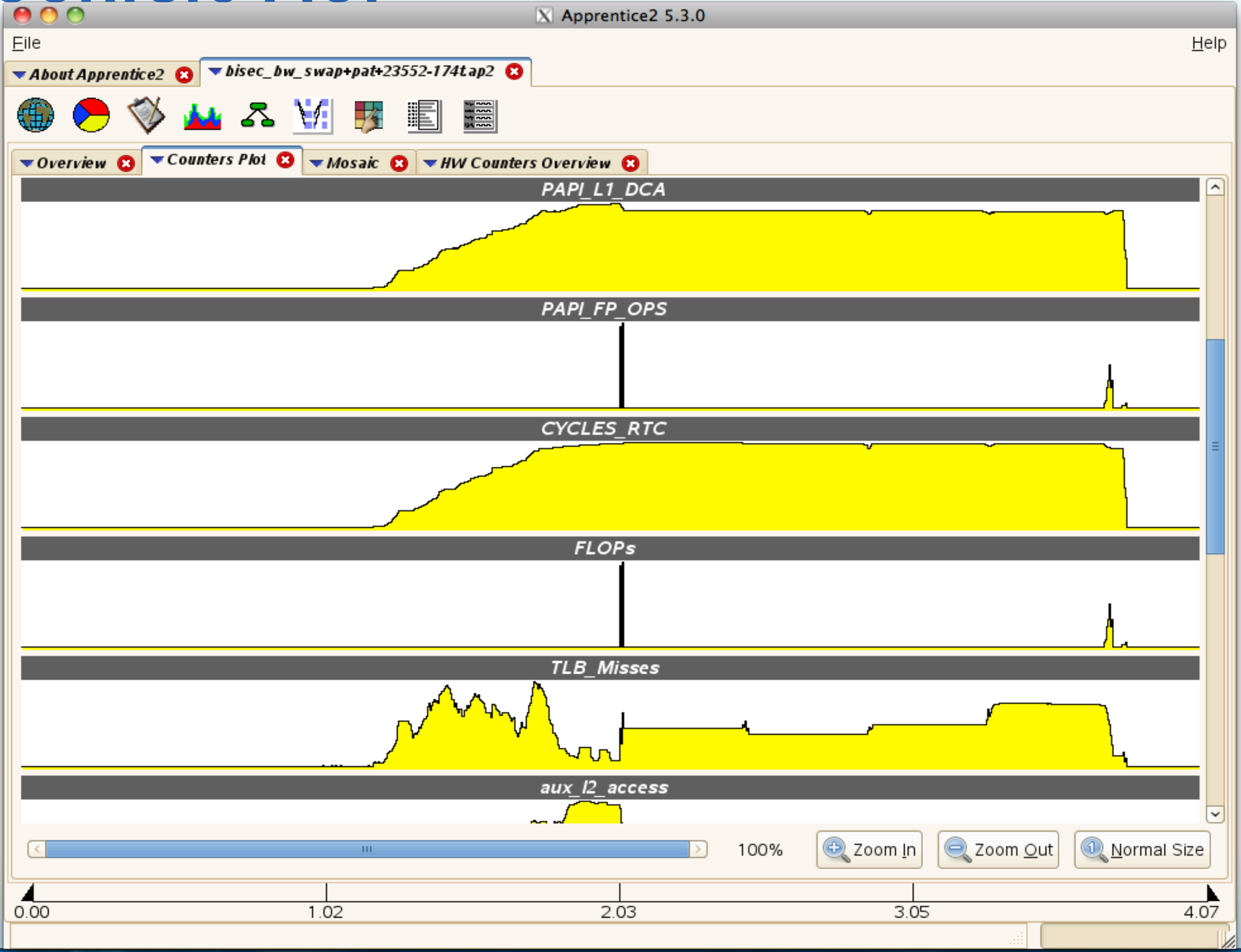


BLUE WATERS
SUSTAINED PETASCALE COMPUTING

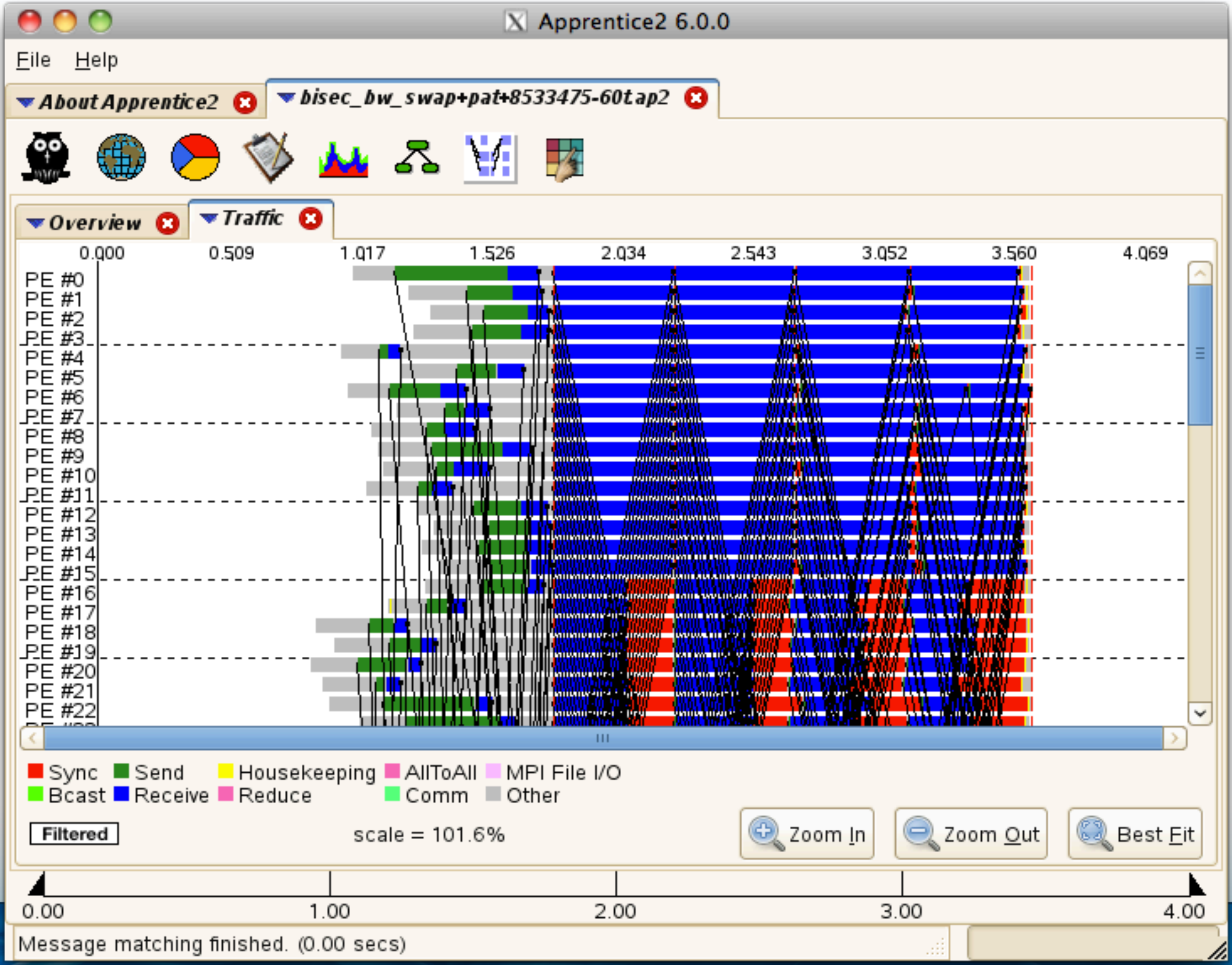
HW Counters Overview



HW Counters Plot



Traffic Report – MPI Communication Timeline



Man pages

- **intro_craypat**(1)

  Introduces the craypat performance tool

- **pat_build(1)**

  Instrument a program for performance analysis

- **pat_help(1)**

  Interactive online help utility

- **pat_report(1)**

  Generate performance report in both text and for use with GUI

- a**pp2 (1)**

  Describes how to launch Cray Apprentice2 to visualize performance data

Man pages (2)

- **hwpc**(5)
  - describes predefined hardware performance counter groups
- nwpc(5)
  - Describes predefined network performance counter groups
- accpc(5) / accpc_k20(5)
  - Describes predefined GPU performance counter groups
- **intro_papi**(3)
  - Lists PAPI event counters
  - Use papi_avail or papi_native_avail utilities to get list of events when running on a specific architecture

MPI Rank Order

MPI Rank Order

Is your nearest neighbor really your nearest neighbor?

And do you want them to be your nearest neighbor?

MPI Rank Placement

- Change default rank ordering with:
  - MPICH_RANK_REORDER_METHOD

- Settings:
  - 0: Round-robin placement – Sequential ranks are placed on the next node in the list. Placement starts over with the first node upon reaching the end of the list.
  - 1: SMP-style placement – Sequential ranks fill up each node before moving to the next. - DEFAULT
  - 2: Folded rank placement – Similar to round-robin placement except that each pass over the node list is in the opposite direction of the previous pass.
  - 3: Custom ordering - The ordering is specified in a file named MPICH_RANK_ORDER.

When Is Rank Re-ordering Useful?

- Maximize on-node communication between MPI ranks

- Grid detection and rank re-ordering is helpful for programs with significant point-to-point communication

- Relieve on-node shared resource contention by pairing threads or processes that perform different work (for example computation with off-node communication) on the same node

Automatic Communication Grid Detection

- Cray performance tools produce a custom rank order if it's beneficial based on grid size, grid order and cost metric
- Heuristics available for:
  - MPI sent message statistics
  - User time (time spent in user functions) – can be used for PGAS codes
  - Hybrid of sent message and user time)
- Summarized findings in report
- Available with sampling or tracing
- Describe how to re-run with custom rank order

MPI Rank Order Observations

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

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

 100.0% | 463.147240 |        -- |     -- | 21621.0 |Total
|-----------------------------------------------------------------------
|  52.0% | 240.974379 |        -- |     -- | 21523.0 |MPI
||----------------------------------------------------------------------
||  47.7% | 221.142266 | 36.214468 |  14.1% | 10740.0 |mpi_recv
||   4.3% |  19.829001 | 25.849906 |  56.7% | 10740.0 |MPI_SEND
||======================================================================
|  43.3% | 200.474690 |        -- |     -- |    32.0 |USER
||----------------------------------------------------------------------
||  41.0% | 189.897060 | 58.716197 |  23.6% |    12.0 |sweep_
||   1.6% |   7.579876 |  1.899097 |  20.1% |    12.0 |source_
||======================================================================
|   4.7% |  21.698147 |        -- |     -- |    39.0 |MPI_SYNC
||----------------------------------------------------------------------
|   4.3% |  20.091165 | 20.005424 |  99.6% |    32.0 | mpi_allreduce_(sync)
||======================================================================
|   0.0% |   0.000024 |        -- |     -- |    27.0 |SYSCALL
|=======================================================================
```

MPI Rank Order Observations (2)

```
MPI Grid Detection:

    There appears to be point-to-point MPI communication in a 96 X 8
    grid pattern. The 52% of the total execution time spent in MPI
    functions might be reduced with a rank order that maximizes
    communication between ranks on the same node. The effect of several
    rank orders is estimated below.

    A file named MPICH_RANK_ORDER.Grid was generated along with this
    report and contains usage instructions and the Custom rank order
    from the following table.

         Rank     On-Node     On-Node   MPICH_RANK_REORDER_METHOD
        Order    Bytes/PE   Bytes/PE%
                             of Total
                             Bytes/PE

        Custom  2.385e+09      95.55%  3
           SMP  1.880e+09      75.30%  1
          Fold  1.373e+06       0.06%  2
    RoundRobin  0.000e+00       0.00%  0
```

MPICH_RANK_ORDER File

```
# The 'Custom' rank order in this file targets nodes with multi-core
# processors, based on Sent Msg Total Bytes collected for:
#
# Program:      /lus/nid00030/heidi/sweep3d/mod/sweep3d.mpi
# Ap2 File:     sweep3d.mpi+pat+27054-89t.ap2
# Number PEs:   48
# Max PEs/Node: 4
#
# To use this file, make a copy named MPICH_RANK_ORDER, and set the
# environment variable MPICH_RANK_REORDER_METHOD to 3 prior to
# executing the program.
#
# The following table lists rank order alternatives and the grid_order
# command-line options that can be used to generate a new order.
…
```

Auto-Generated MPI Rank Order File

```
# The 'USER_Time_hybrid'
rank order in this file
targets nodes with multi-
core
# processors, based on
Sent Msg Total Bytes
collected for:
#
# Program:      /lus/
nid00023/malice/craypat/
WORKSHOP/bh2o-demo/Rank/
sweep3d/src/sweep3d
# Ap2 File:
sweep3d.gmpi-u.ap2
# Number PEs:   768
# Max PEs/Node: 16
#
# To use this file, make
a copy named
MPICH_RANK_ORDER, and set
the
# environment variable
MPICH_RANK_REORDER_METHOD
to 3 prior to
# executing the program.
#
0,532,64,564,32,572,96,54
0,8,596,72,524,40,604,24,
588
104,556,16,628,80,636,56,
620,48,516,112,580,88,548
,120,612
1,403,65,435,33,411,97,44
3,9,467,25,499,105,507,41
,475
73,395,81,427,57,459,17,4
19,113,491,49,387,89,451,
121,483
6,436,102,468,70,404,38,4
12,14,444,46,476,110,508,
78,500
86,396,30,428,62,460,54,4
92,118,420,22,452,94,388,
126,484
129,563,193,531,161,571,2
25,539,241,595,233,523,24
9,603,185,555
153,587,169,627,137,635,2
01,619,177,515,145,579,20
9,547,217,611
7,405,71,469,39,437,103,4
13,47,445,15,509,79,477,3
1,501
111,397,63,461,55,429,87,
421,23,493,119,389,95,453
,127,485
134,402,198,434,166,410,2
30,442,238,466,174,506,15
8,394,246,474
190,498,254,426,142,458,1
50,386,182,418,206,490,21
4,450,222,482
128,533,192,541,160,565,2
32,525,224,573,240,597,18
4,557,248,605
168,589,200,517,152,629,1
36,549,176,637,144,621,20
8,581,216,613
5,439,37,407,69,447,101,4
15,13,471,45,503,29,479,7
7,511
53,399,85,431,21,463,61,3
91,109,423,93,455,117,495
,125,487
2,530,34,562,66,538,98,52
2,10,570,42,554,26,594,50
,602
18,514,74,586,58,626,82,5
46,106,634,90,578,114,618
,122,610
135,315,167,339,199,347,2
59,307,231,371,239,379,19
1,331,247,299
175,363,159,323,143,355,2
55,291,207,275,183,283,15
1,267,215,223
133,406,197,438,165,470,2
29,414,245,446,141,478,23
7,502,253,398
157,510,189,462,173,430,2
05,390,149,422,213,454,18
1,494,221,486
130,316,260,340,194,372,1
62,348,226,308,234,380,24
2,332,250,300
202,364,186,324,154,356,1
38,292,170,276,178,284,21
0,218,268,146
4,535,36,543,68,567,100,5
27,12,599,44,575,28,559,7
6,607
52,591,20,631,60,639,84,5
19,108,623,92,551,116,583
,124,615
3,440,35,432,67,400,99,40
8,11,464,43,496,27,472,51
,504
19,392,75,424,59,456,83,3
84,107,416,91,488,115,448
,123,480
132,401,196,441,164,409,2
28,433,236,465,204,473,24
4,393,188,497
252,505,140,425,212,457,1
56,385,172,417,180,449,14
8,489,220,481
131,534,195,542,163,566,2
27,526,235,574,203,598,24
3,558,187,606
251,590,211,630,179,638,1
39,622,155,550,171,518,21
9,582,147,614
761,660,737,652,705,668,7
45,692,673,700,641,684,71
3,644,753,724
729,732,681,756,721,716,7
64,676,697,748,689,657,74
0,665,649,708
760,528,736,536,704,560,7
44,520,672,568,712,592,75
2,552,640,600
728,584,680,624,720,512,6
96,632,688,616,664,544,60
8,656,648,576
762,659,738,651,706,667,7
46,643,714,691,674,699,75
4,683,730,723
722,731,763,658,642,755,7
39,675,707,650,682,715,69
8,666,690,747
257,345,265,313,281,305,2
73,337,609,369,577,377,61
7,329,513,529
545,297,633,361,625,321,5
85,537,601,289,553,353,59
3,521,569,561
256,373,261,341,264,349,2
80,317,272,381,269,309,28
5,333,277,365
352,301,320,325,288,357,3
28,304,360,312,376,293,29
6,368,336,344
258,338,266,346,282,314,2
74,370,766,306,710,378,74
2,330,678,362
646,298,750,322,718,354,7
58,290,734,662,686,670,72
6,702,694,654
262,375,263,343,270,311,2
71,351,286,319,278,342,28
7,350,279,374
294,318,358,383,359,310,2
95,382,326,303,327,367,36
6,335,302,334
765,661,709,663,741,653,7
11,669,767,655,743,671,74
9,695,679,703
677,727,751,693,647,701,7
17,687,757,685,733,725,71
9,735,645,759
```

grid_order Utility

- Can use grid_order utility without first running the application with the Cray performance tools if you know a program's data movement pattern
- Originally designed for MPI programs, but since reordering is done by PMI, it can be used by other programming models (since PMI is used by MPI, SHMEM and PGAS programming models)
- Utility available if perftools modulefile is loaded
- See grid_order(1) man page or run grid_order with no arguments to see usage information

Reorder Example for Bisection Bandwidth

- Assume 32 ranks
- Decide on row or column ordering:
- $ grid_order –R –g 2,16

```
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
```

- $ grid_order –C –g 2,16

```
0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30
1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31
```

- Since rank 0 talks to rank 16, and not with rank 1, we choose Row ordering

Reorder Example for Bisection Bandwidth (2)

- Specify cell (or chunk) to make sure rank pairs live on same node (but don't care how many pairs live on a node)

- $ grid_order –R –g 2,16 –c 2,1

```
0,16
1,17
2,18
3,19
4,20
5,21
6,22
7,23
8,24
9,25
10,26
11,27
12,28
13,29
14,30
15,31
```

Fills a Magny-Cours node

Using New Rank Order

- Save grid_order output to file called MPICH_RANK_ORDER
- Export MPICH_RANK_REORDER_METHOD=3
- Run non-instrumented binary with and without new rank order to check overall wallclock time for improvement

Example Performance Results

- Default thread ordering
  - Application 8538980 resources: utime ~126s, stime ~108s

- Maximized on-node data movement with reordering
  - Application 8538982 resources: utime ~38s, stime ~106s

BLUE WATERS

SUSTAINED PETASCALE COMPUTING

PerfSuite

NCSA NSF GREAT LAKES CONSORTIUM FOR PETASCALE COMPUTATION CRAY

PerfSuite Background

- Active development at NCSA since 2001
- UI/NCSA Open Source license
- Targeted to users of all levels of expertise
  - The intent is to provide an easy-to-use mechanism for measuring application performance, and to expose problem areas for further exploration
- Low measurement overhead

Features

- Counting and profiling using hardware performance event counters on CPUs, GPUs, networks, and using interval timers
- Executes un-modified dynamically-linked applications
- Easy to use XML files for configuration and output
    - Easy to change the events to count/profile, just change the XML file
- Metadata (such as processor information, pid, memory and time usage) stored in the output
- Functionality available through both command-line tools and library API

Provides

- Three performance counter-related utilities:
  - `psrun` - generate raw counter or statistical profiling data from an unmodified binary
  - `psprocess` - pre- and post-process data
  - `psinv` - query events and machine information
- Three libraries
  - libpshwpc – HardWare Performance Counter library
    - This is the one that you are most likely to use, if you ever need to
    - Insert API calls in the source code for finer control of start/end locations
  - libperfsuite – the "core" library
  - libpshwpc_mpi – a convenience library to capture MPI calls

Using PerfSuite on BW

```
# First, load the perfsuite module

% module load perfsuite

# By default, psrun does event counting on the given program,
# then use psprocess to produce psrun's output XML files

% aprun -n <num> psrun -f -p myprog myprog_args
% psprocess myprog.0.12345.nid01234.xml

 # Use a profiling config file to do profiling instead of counting

% aprun -n <num> psrun -C -c papi_profile_cycles.xml -f -p
 myprog myprog_args
% psprocess myprog.1.67890.nid12345.xml
```

psinv: Processor Inventory

- Lists information about the characteristics of the computer
- This same information is also stored in PerfSuite XML output and is useful for later generating derived
- Lists available hardware performance events

```
titan:~3% psinv -v
System Information -
Processors:             2
Total Memory:           2007.16 MB
System Page Size:       16.00 KB

Processor Information -
Vendor:                 Intel
Processor family:       IPF
Model (Type):           Itanium
Revision:               6
Clock Speed:            800.136 MHz

Cache and TLB Information -
Cache levels:           3
Caches/TLBs:            7

Cache Details -
Level 1:
        Type:           Data
        Size:           16 KB
        Line size:      32 bytes
        Associativity:  4-way set associative

        Type:           Instruction
        Size:           16 KB
        Line size:      32 bytes
        Associativity:  4-way set associative
```

psrun: Performance Measurement

- Hardware performance counting and profiling with unmodified dynamically-linked executables
- Supports:
  - x86, x86-64, and powerpc
  - MPI and OpenMP
  - resource usage (memory, CPU time) collection
  - Both PAPI standard and CPU/GPU native events
- Configuration = XML, Output = XML or text
- Use "-p" for OpenMP programs, "-f" for MPI, and "-f -p" for MPI+OpenMP programs

Example Configuration

- For counting: set “ps_hwpc_eventlist” as the XML root element
- For profiling: set “ps_hwpc_profile” as the XML root element

```
<?xml version="1.0" encoding="UTF-8" ?>
<ps_hwpc_eventlist class="PAPI">
  <ps_hwpc_event type="preset" name="PAPI_FP_OPS" />
  <ps_hwpc_event type="preset" name="PAPI_TOT_CYC" />
  <ps_hwpc_event type="preset" name="PAPI_L1_DCM" />
  <ps_hwpc_event type="preset" name="PAPI_L2_DCM" />
  <ps_hwpc_event type=“native”>NOPS_RETIRED</ps_hwpc_event>
  <ps_hwpc_event type=“native”>BACK_END_BUBBLE_ALL</ps_hwpc_event>
</ps_hwpc_eventlist>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<ps_hwpc_profile class="PAPI">
  <ps_hwpc_event type="preset" name="PAPI_BR_MSP“ threshold="100000" />
</ps_hwpc_profile>
```

psprocess: Text Mode (default)

```
PerfSuite Hardware Performance Summary Report
Version        : 1.0
Created        : Mon Dec 30 11:31:53 AM Central Standard Time 2002
Generator      : psprocess 0.5
XML Source     : /u/ncsa/anyuser/performance/psrun-ia64.xml

Execution Information
===========================
Date           : Sun Dec 15 21:01:20 2002
Host           : user01

Processor and System Information
===========================
Node CPUs      : 2
Vendor         : Intel
Family         : IPF
Model          : Itanium
CPU Revision : 6
Clock (MHz)    : 800.136
Memory (MB)    : 2007.16
Pagesize (KB): 16
```

psprocess: Text Mode, cont'd

```
Cache Information
==========================
Cache levels : 3
--------------------------------
Level 1
Type           : data
Size (KB)      : 16
Linesize (B)   : 32
Assoc          : 4
Type           : instruction
Size (KB)      : 16
Linesize (B)   : 32
Assoc          : 4
--------------------------------
Level 2
Type           : unified
Size (KB)      : 96
Linesize (B)   : 64
Assoc          : 6
```

The reports (text or HTML) generated by psprocess have several sections, covering:

- Report creation details
- Run details
- Machine information
- Raw counter listings
- Counter explanations and index
- Derived metrics
- Run annotation defined by you

Derived metrics are evaluated at run-time and can be extended (text mode only)

psprocess: Text Mode, cont'd

```
Index Description                                     Counter Value
===================================================================
1 Conditional branch instructions mispredicted.....      4831072449
4 Floating point instructions......................     86124489172
5 Total cycles.....................................    594547754568
6 Instructions completed...........................   1049339828741

Statistics
===================================================================
Graduated instructions per cycle...................           1.765
Graduated floating point instructions per cycle....           0.145
Level 3 cache miss ratio (data)....................           0.957
Bandwidth used to level 3 cache (MB/s).............         385.087
% cycles with no instruction issue.................          10.410
% cycles stalled on memory access..................          43.139
MFLOPS (cycles)....................................         115.905
MFLOPS (wallclock).................................         114.441
```

psprocess Text-Based Profiles

```
Profile Information
================================================================================
Class                          : PAPI
Version                        : 3.6.2
Event                          : PAPI_TOT_CYC (Total cycles)
Period                         : 100000
Samples                        : 200471
Domain                         : user
Run Time                       : 27.19 (seconds)
Min Self %                     : (all)

Module Summary
--------------------------------------------------------------------------------
 Samples   Self %  Total %  Module

   186068   92.82%   92.82%  /home/rkufrin/apps/aspcg/aspcg
    14182    7.07%   99.89%  /opt/intel/cc/9.0/lib/libguide.so
      187    0.09%   99.98%  /lib/ld-2.3.6.so
       18    0.01%   99.99%  /lib/tls/libc-2.3.6.so
       15    0.01%  100.00%  /lib/tls/libpthread-2.3.6.so
        1    0.00%  100.00%  /tmp/perfsuite/lib/libpsrun_r.so.0.0.1

File Summary
--------------------------------------------------------------------------------
 Samples   Self %  Total %  File

   154346   76.99%   76.99%  /home/rkufrin/apps/aspcg/pc_jac2d_blk3.f
    14506    7.24%   84.23%  /home/rkufrin/apps/aspcg/cg3_blk.f
    14505    7.24%   91.46%  ??
    10185    5.08%   96.54%  /home/rkufrin/apps/aspcg/matxvec2d_blk3.f
     3042    1.52%   98.06%  /home/rkufrin/apps/aspcg/dot_prod2d_blk3.f
     2366    1.18%   99.24%  /home/rkufrin/apps/aspcg/add_exchange2d_blk3.f
      834    0.42%   99.66%  /home/rkufrin/apps/aspcg/main3.f
      687    0.34%  100.00%  /home/rkufrin/apps/aspcg/cs_jac2d_blk3.f
```

Text-based profiles, cont'd

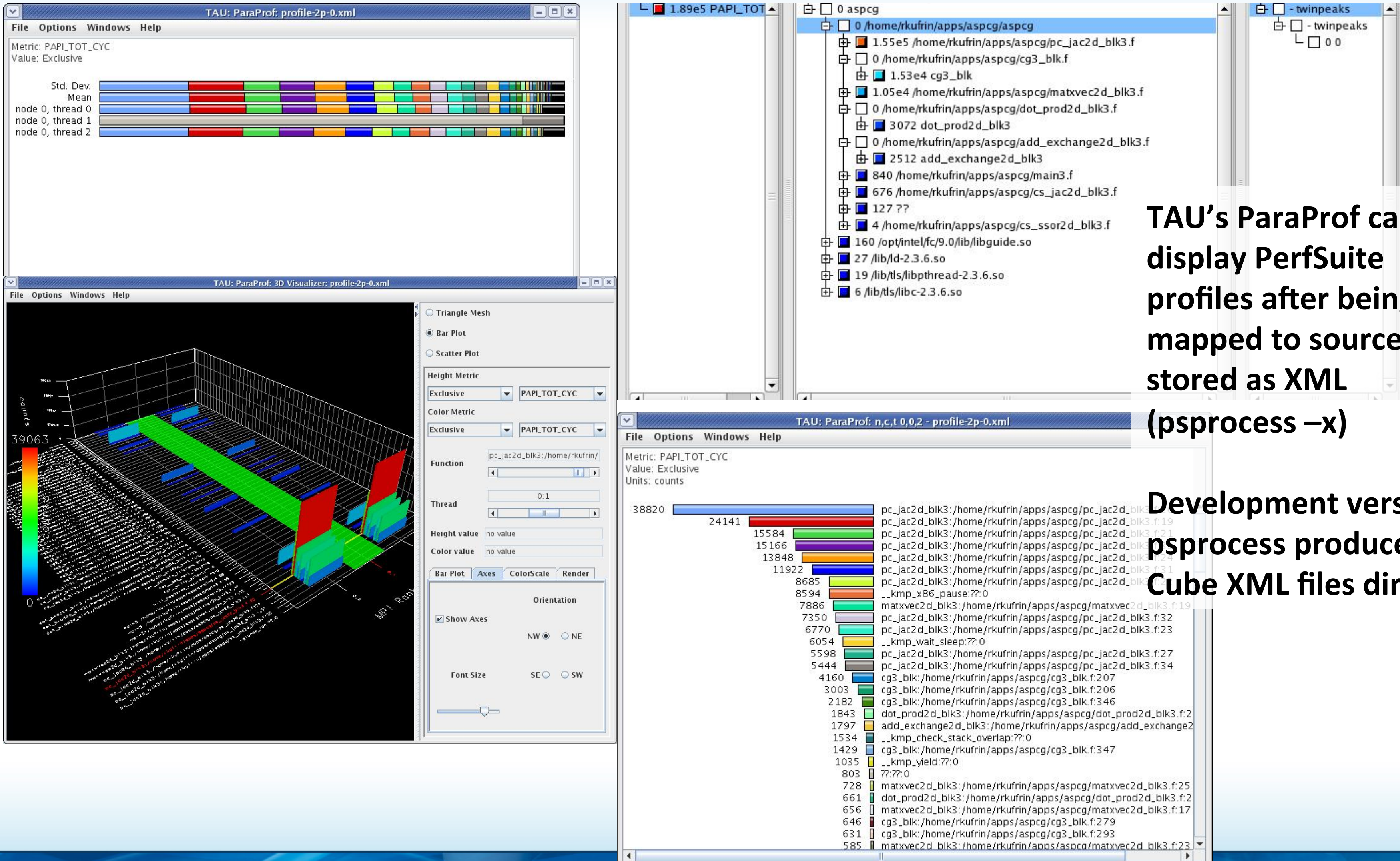
```
Function Summary
--------------------------------------------------------------------------------
 Samples    Self %  Total %  Function

   154346   76.99%   76.99%  pc_jac2d_blk3
    14506    7.24%   84.23%  cg3_blk
    10185    5.08%   89.31%  matxvec2d_blk3
     6937    3.46%   92.77%  __kmp_x86_pause
     4711    2.35%   95.12%  __kmp_wait_sleep
     3042    1.52%   96.64%  dot_prod2d_blk3
     2366    1.18%   97.82%  add_exchange2d_blk3


Function:File:Line Summary
--------------------------------------------------------------------------------
 Samples    Self %  Total %  Function:File:Line

    39063   19.49%   19.49%  pc_jac2d_blk3:/home/rkufrin/apps/aspcg/pc_jac2d_blk3.f:20
    24134   12.04%   31.52%  pc_jac2d_blk3:/home/rkufrin/apps/aspcg/pc_jac2d_blk3.f:19
    15626    7.79%   39.32%  pc_jac2d_blk3:/home/rkufrin/apps/aspcg/pc_jac2d_blk3.f:21
    15028    7.50%   46.82%  pc_jac2d_blk3:/home/rkufrin/apps/aspcg/pc_jac2d_blk3.f:33
    13878    6.92%   53.74%  pc_jac2d_blk3:/home/rkufrin/apps/aspcg/pc_jac2d_blk3.f:24
    11880    5.93%   59.66%  pc_jac2d_blk3:/home/rkufrin/apps/aspcg/pc_jac2d_blk3.f:31
     8896    4.44%   64.10%  pc_jac2d_blk3:/home/rkufrin/apps/aspcg/pc_jac2d_blk3.f:22
     7863    3.92%   68.02%  matxvec2d_blk3:/home/rkufrin/apps/aspcg/matxvec2d_blk3.f:19
     7145    3.56%   71.59%  pc_jac2d_blk3:/home/rkufrin/apps/aspcg/pc_jac2d_blk3.f:32
```

PerfSuite Profiles with ParaProf and Cube3



TAU's ParaProf can display PerfSuite profiles after being mapped to source and stored as XML (psprocess –x)

Development version of psprocess produces Cube XML files directly

libpshwpc: Performance Collection API

C / C++

```
ps_hwpc_init (void)
ps_hwpc_start (void)
ps_hwpc_read (long long *values)
ps_hwpc_suspend (void)
ps_hwpc_stop (char *prefix)
ps_hwpc_shutdown (void)
```

Fortran

```
call psf_hwpc_init (ierr)
call psf_hwpc_start (ierr)
call psf_hwpc_read (integer*8
   values,ierr)
call psf_hwpc_suspend (ierr)
call psf_hwpc_stop (prefix, ierr)
call psf_hwpc_shutdown (ierr)
```

- Call “init” once, call “start”, “read” and “suspend” as many times as you like. Call “stop” (supplying a file name prefix of your choice) to get the performance data XML document
- Optionally, call “shutdown”
- Example programs demonstrating use are installed in PerfSuite “examples” subdirectory
- Additional routines `ps_hwpc_numevents()` and `ps_hwpc_eventnames()` allow querying current configuration

FORTRAN API Example

```
include 'fperfsuite.h'
call PSF_hwpc_init(ierr)
call PSF_hwpc_start(ierr)
do j = 1, n
   do i = 1, m
      do k = 1, l
         c(i,j) = c(i,j) + a(i,k)*b(k,j)
      end do
   end do
end do
call PSF_hwpc_stop('perf', ierr)
call PSF_hwpc_shutdown(ierr)
```

```
% ftn -c matmult.f -I /sw/xe/perfsuite/1.1.2/
cnl4.1_gnu4.7.2_papi5.1.0.2/include

% ftn matmult.o -L /sw/xe/perfsuite/1.1.2/cnl4.1_gnu4.7.2_papi5.1.0.2/
lib -L /opt/cray/papi/5.1.0.2/perf_events/no-cuda/lib -lpshwpc -
lperfsuite -lpapi
```

Java-based Performance Measurement

- PerfSuite 1.0.0+ supports measuring unmodified Java applications using PAPI similar to `psrun`
- Implemented using JVMTI (Java Virtual Machine Tool Interface)
- Usage:

```
java -agentlib:psjrun MyClass
```

- Results are contained in XML documents that can be post-processed using `psprocess`

For More Information

- PerfSuite web sites:
  - https://bluewaters.ncsa.illinois.edu/perfsuite
  - http://perfsuite.ncsa.illinois.edu

BLUE WATERS

SUSTAINED PETASCALE COMPUTING

TAU : *Tuning* and *Analysis* *Utilities*

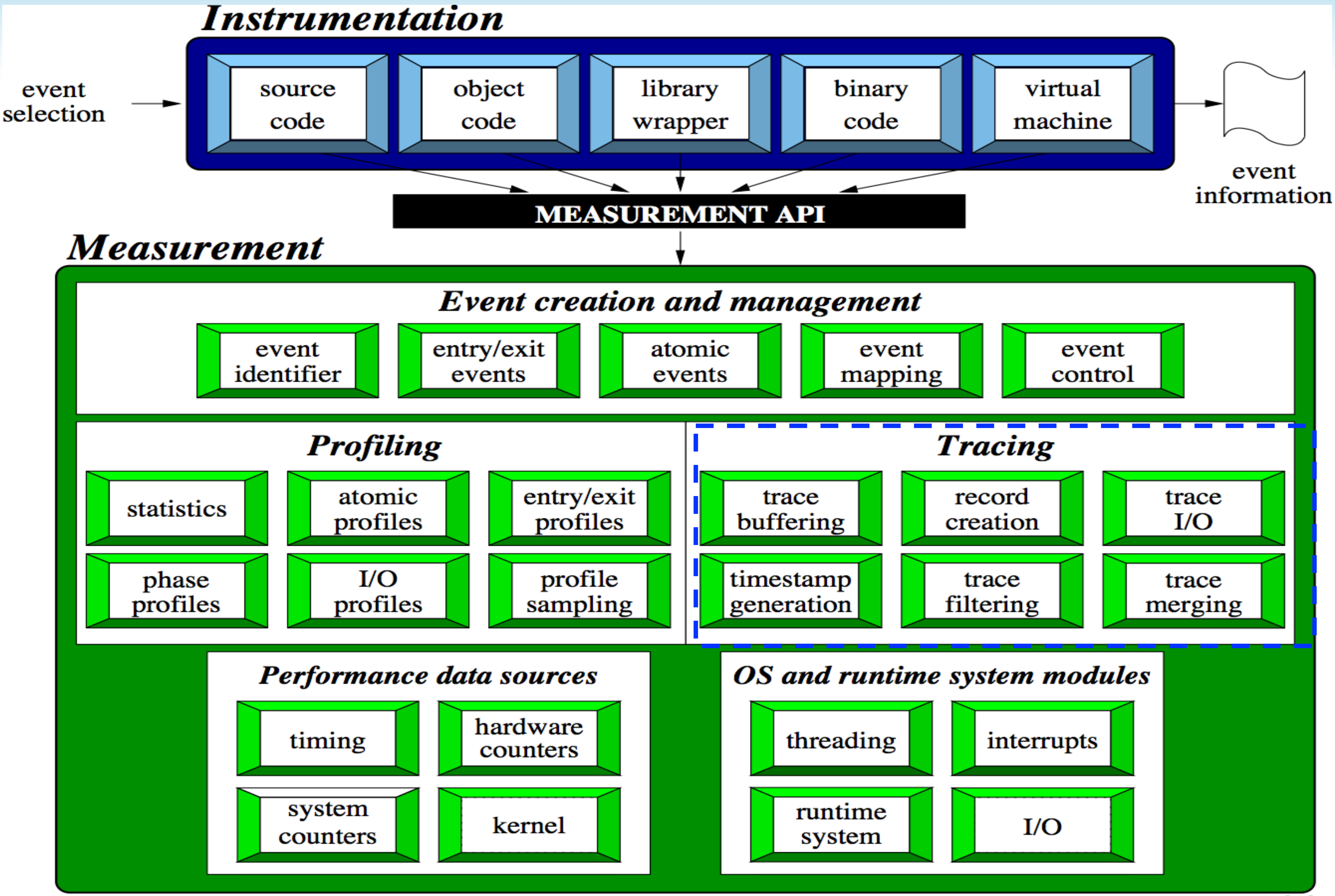
GREAT LAKES CONSORTIUM FOR PETASCALE COMPUTATION

TAU Performance System®

- *T*uning and *A*nalysis *U*tilities (18+ year project)
- Developed at University of Oregon, Eugene
- Performance problem solving framework for HPC
  - Integrated, scalable, flexible, portable
  - Target all parallel programming / execution paradigms
- Integrated performance toolkit (open source)
  - Instrumentation, measurement, analysis, visualization
  - Widely-ported performance profiling / tracing system
  - Performance data management and data mining
- Broad application use (NSF, DOE, DOD, …)

Instrumentation and Sampling

- Supports both instrumentation (direct performance observation) and sampling (indirect perf observation)
- Instrumentation can be:
  - Manually insert into the source code
  - Automatically by compiler (use the “-optCompInst” option)
  - Automatically by TAU’s PDT tool (by choosing a TAU makefile containing the “-pdt” string)
- Supports selective instrumentation:
  - Selects the files, functions/routines, loops
  - Supports patterns: “*”, “?” (for file names), “#” (for loops)


Instrumentation
event selection
source code
object code
library wrapper
binary code
virtual machine
event information
MEASUREMENT API
Measurement
Event creation and management
event identifier
entry/exit events
atomic events
event mapping
event control
Profiling
statistics
atomic profiles
entry/exit profiles
phase profiles
I/O profiles
profile sampling
Tracing
trace buffering
record creation
trace I/O
timestamp generation
trace filtering
trace merging
Performance data sources
timing
hardware counters
system counters
kernel
OS and runtime system modules
threading
interrupts
runtime system
I/O

Using TAU on BW

- First, load the tau module
- Two methods to run to generate TAU profile or trace files
  - Choose a TAU makefile or use a TAU compiler script, to instrument the source code, build an instrumented executable, then execute it
  - Use "tau_exec" to run an uninstrumented executable
- Analyze the generated files, using "pprof" for quick text output, or "paraprof" for richer visualization

TAU Makefiles on BW

- To see all installed TAU versions

```
module avail tau
```

- To load the tau module, use one of the following:

```
module load tau
module load tau/<specific version>
```

- The TAU makefiles are located at:

  /sw/xe/tau/<tau_version>/<build>/craycnl/lib/Makefile.tau-*

  For example, for TAU 2.21.4 using PrgEnv-cray, they are:

  /sw/xe/tau/2.21.4/cnl4.1_cray8.1.1/craycnl/lib/Makefile.tau-*

- For an MPI+F90 application, you may want to start with Makefile.tau-cray-mpi-pdt, which supports MPI instrumentation & PDT for automatic source instrumentation

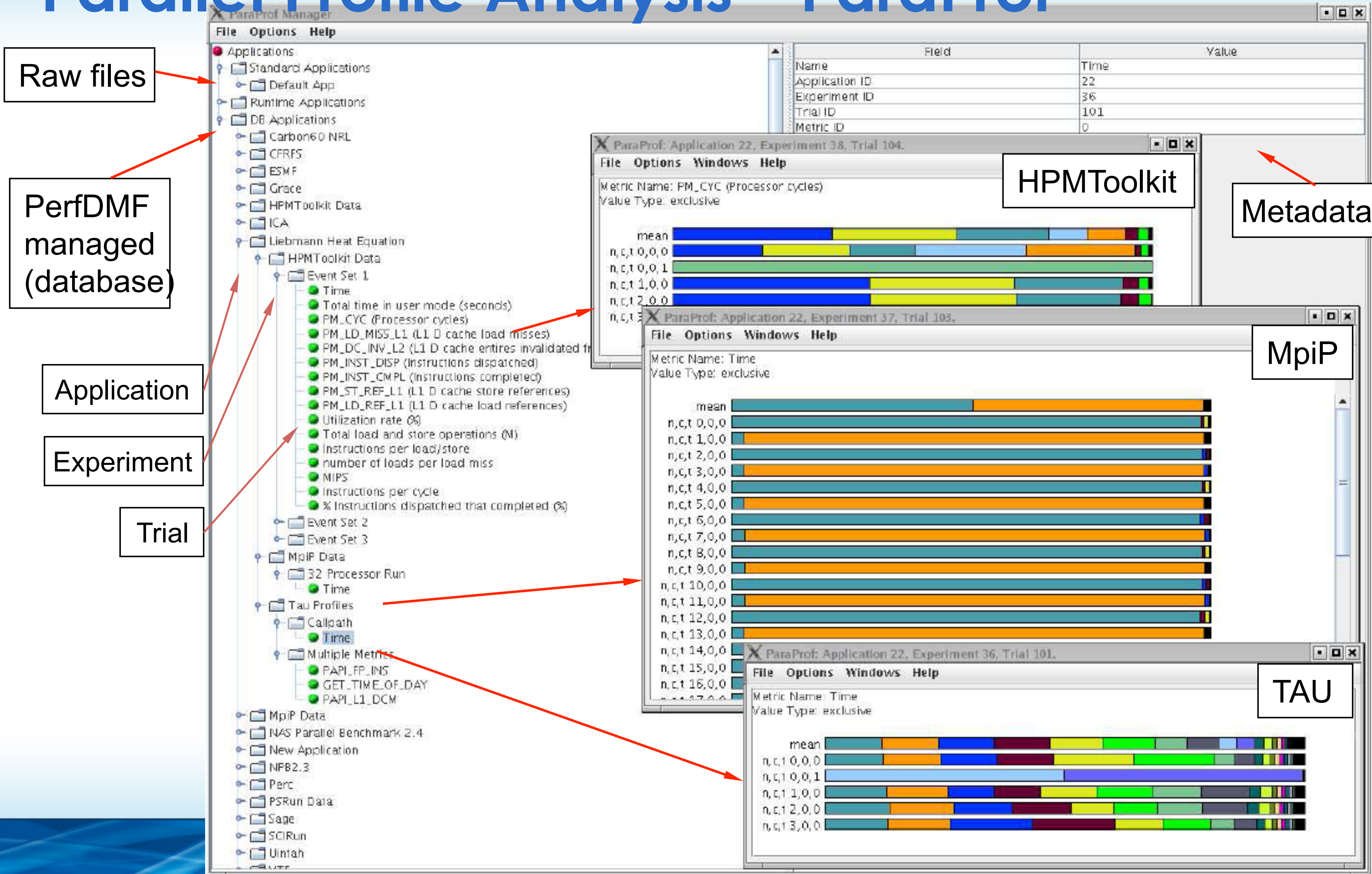
```
setenv TAU_MAKEFILE
/sw/xe/tau/2.21.4/cnl4.1_cray8.1.1/craycnl/lib/
Makefile.tau-cray-mpi-pdt
```

Parallel Profile Analysis – pprof

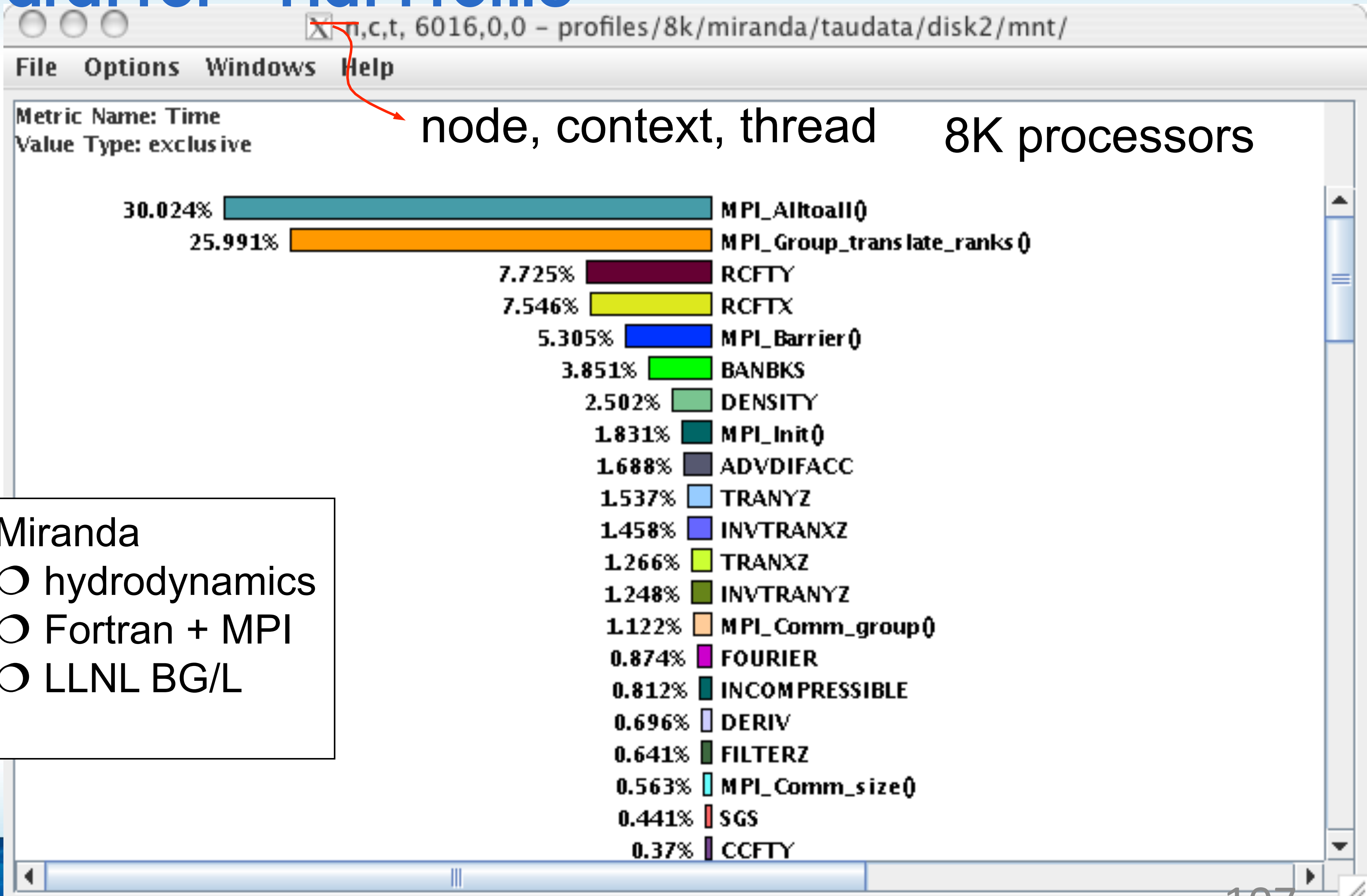
```
emacs@neutron.cs.uoregon.edu
Buffers Files Tools Edit Search Mule Help
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0            1     3:11.293           1          15  191293269 applu
 99.6        3,667     3:10.463           3       37517   63487925 bcast_inputs
 67.1          491     2:08.326       37200       37200       3450 exchange_1
 44.5        6,461     1:25.159        9300       18600       9157 buts
 41.0     1:18.436     1:18.436       18600           0       4217 MPI_Recv()
 29.5        6,778       56,407        9300       18600       6065 blts
 26.2       50,142       50,142       19204           0       2611 MPI_Send()
 16.2       24,451       31,031         301         602     103096 rhs
  3.9        7,501        7,501        9300           0        807 jacld
  3.4          838        6,594         604        1812      10918 exchange_3
  3.4        6,590        6,590        9300           0        709 jacu
  2.6        4,989        4,989         608           0       8206 MPI_Wait()
  0.2         0.44          400           1           4     400081 init_comm
  0.2          398          399           1          39     399634 MPI_Init()
  0.1          140          247           1       47616     247086 setiv
  0.1          131          131       57252           0          2 exact
  0.1           89          103           1           2     103168 erhs
  0.1        0.966           96           1           2      96458 read_input
  0.0           95           95           9           0      10603 MPI_Bcast()
  0.0           26           44           1        7937      44878 error
  0.0           24           24         608           0         40 MPI_Irecv()
  0.0           15           15           1           5      15630 MPI_Finalize()
  0.0            4           12           1        1700      12335 setbv
  0.0            7            8           3           3       2893 l2norm
  0.0            3            3           8           0        491 MPI_Allreduce()
  0.0            1            3           1           6       3874 pintgr
  0.0            1            1           1           0       1007 MPI_Barrier()
  0.0        0.116        0.837           1           4        837 exchange_4
  0.0        0.512        0.512           1           0        512 MPI_Keyval_create()
  0.0        0.121        0.353           1           2        353 exchange_5
  0.0        0.024        0.191           1           2        191 exchange_6
  0.0        0.103        0.103           6           0         17 MPI_Type_contiguous()
--:--  NPB_LU.out        (Fundamental)--L8--Top----------------------------------------
```

Parallel Profile Analysis – ParaProf



ParaProf – Flat Profile



node, context, thread

8K processors

Miranda
- hydrodynamics
- Fortran + MPI
- LLNL BG/L

For More Information

- TAU web sites:
  - https://bluewaters.ncsa.illinois.edu/tau
  - http://tau.uoregon.edu

BLUE WATERS

SUSTAINED PETASCALE COMPUTING

Congestion Protection and Balanced Injection

GREAT LAKES CONSORTIUM FOR PETASCALE COMPUTATION

What is Congestion Protection?

- Network congestion is a condition that occurs when the volume of traffic on the high-speed network (HSN) exceeds the capacity to handle it.
- To "protect" the network from data loss, congestion protection (CP) globally “throttles” injection bandwidth per-node.
- If CP happens often, application performance degrades.

http://lh5.google.ca/abramsv/R9WYOKtLe1I/AAAAAAAALO4/FLefbnOq5rQ/s1600-h/495711679_52f8d76d11_o.jpg

- At job completion you might see the following message reported to stdout:

```
Application 61435 network throttled: 4459 nodes throttled, 25:31:21 node-seconds
Application 61435 balanced injection 100, after throttle 63
```

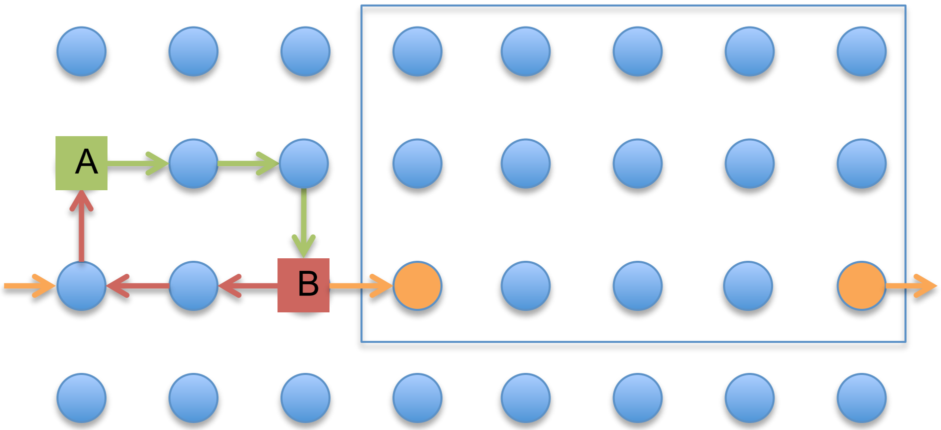
- The throttling event lasts for 20 seconds each time CP is triggered.

Types of congestion events

- There are two main forms of congestion: many-to-one and long-path. The former is easy to detect and correct. The latter is harder to detect and may not be correctable.
- Many-to-one congestion occurs in some algorithms and can be corrected. uGNI and DMAPP based codes doing All-to-one operations are common case. See “Modifying Your Application to Avoid Gemini Network Congestion Errors” on balanced injection section on the portal.
- Long-path congestion is typically due to a combination of communication pattern and node allocation. It can also be due to a combination of jobs running on the system.
- We monitor for cases of congestion protection and try to determine the most likely cause.

Congestion on a Shared, Torus Network

- HSN uses dimension ordered routing: x-then-y-then-z between two locations on the torus. Note that A➔B≠B➔A.
- Shortest route can sometimes cause traffic to pass through geminis used by other jobs.



- Non-convex node allocations can have traffic that passes through geminis used by other jobs.
- Non-local I/O traffic and Lustre striping can lead to network hot-spots.
- We are working with Adaptive and Cray on eliminating some of the above causes of congestion with better node allocation: shape, location, etc.

Balanced Injection

- Balanced Injection (BI) is a mechanism that attempts to reduce compute node injection bandwidth in order to prevent throttling and which may have the effect of improving application performance for certain communication patterns.
- BI can be applied "per-job" using an environment variable or with user accessible API.
- export APRUN_BALANCED_INJECTION=64
- Can be set from 1-100 (100 = no BI).
- There isn't a linear relation of BI to application performance.
- MPI-based applications have "balanced injection" enabled in collective MPI calls that locally "throttle" injection bandwidth.

Thank You