BLUE WATERS

SUSTAINED PETASCALE COMPUTING

User Tools on Blue Waters

Manisha Gajbe + others

GREAT LAKES CONSORTIUM FOR PETASCALE COMPUTATION

Cray Performance Tools

Topics

- 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
- MPICH Rank Order

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

S

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
  - View trace file with Cray Apprentice[2]

- Request hardware performance counter information:
  - export PAT_RT_HWPC=<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 –gnetcdf,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 Apprentice²

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

Steps to Collecting Performance Data (2)

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

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

- 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, and TLB

default: group 23

Support for L3 cache counters coming in 3Q2013

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

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
||||=============================================
.  .  .
```

Profile Visualization with Cray Apprentice2

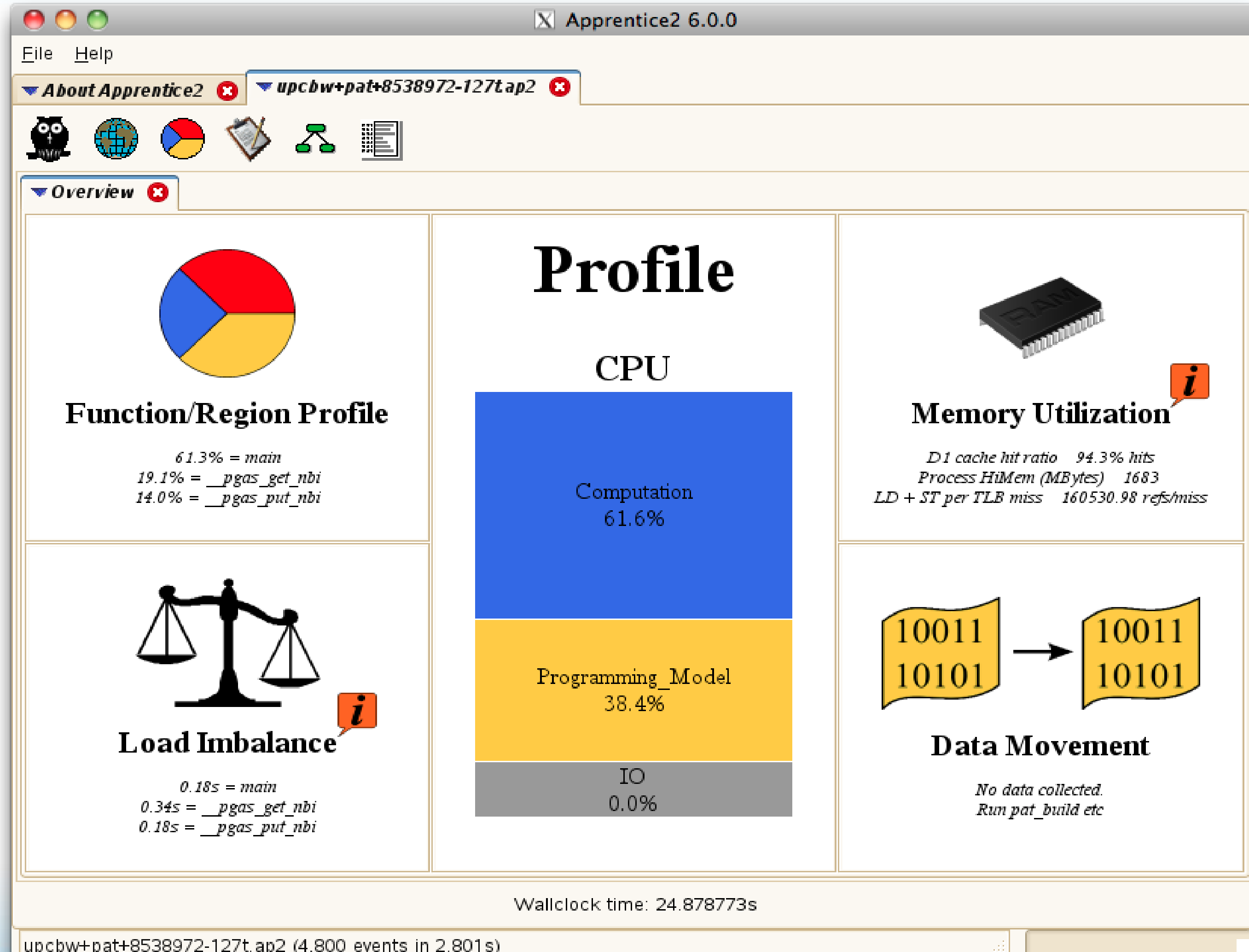
Cray Apprentice^2

- 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

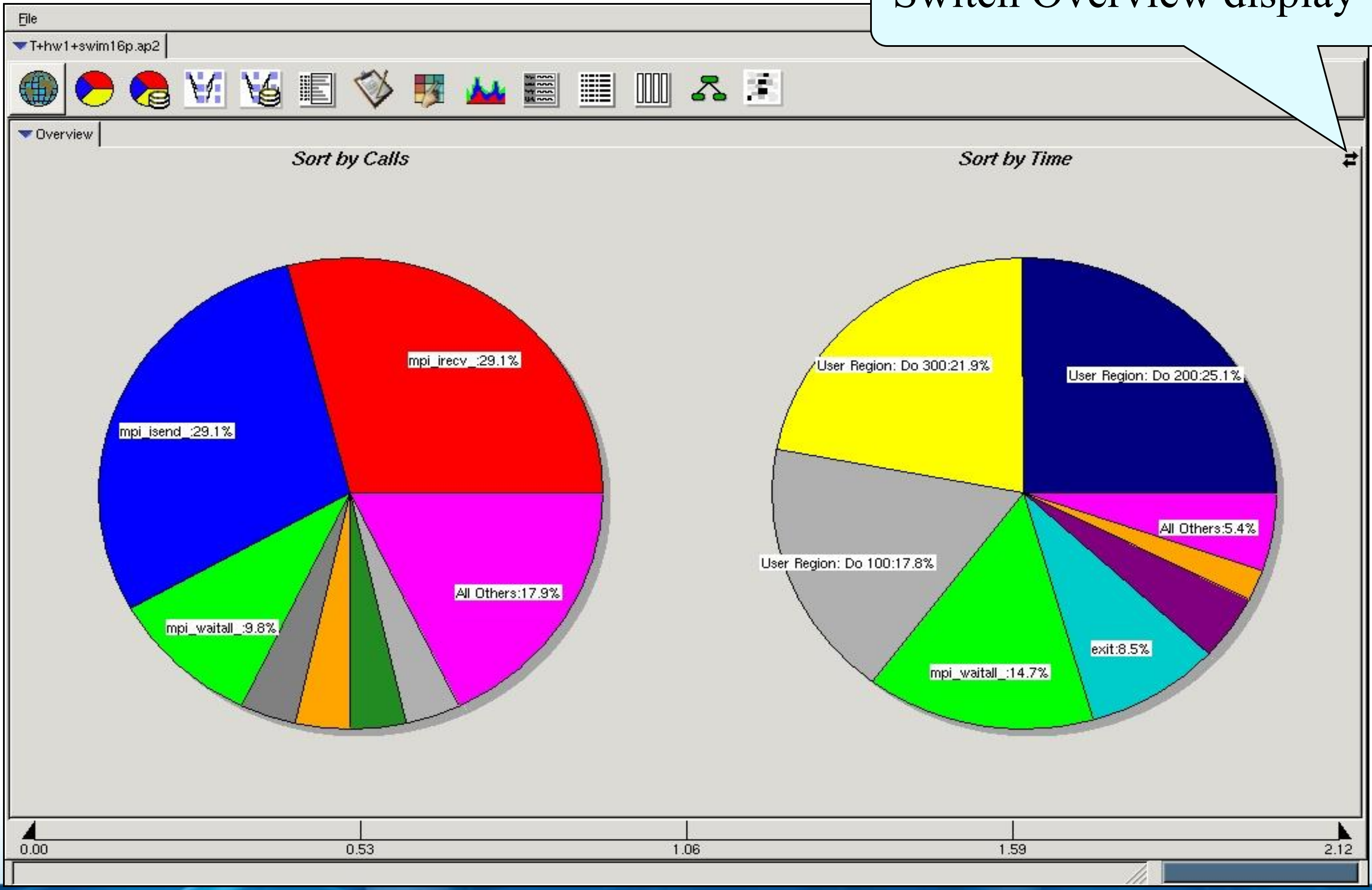
- Cray Apprentice^2 helps identify:
  - Load imbalance
  - Excessive communication
  - Network contention
  - Excessive serialization
  - I/O Problems

Application Performance Summary

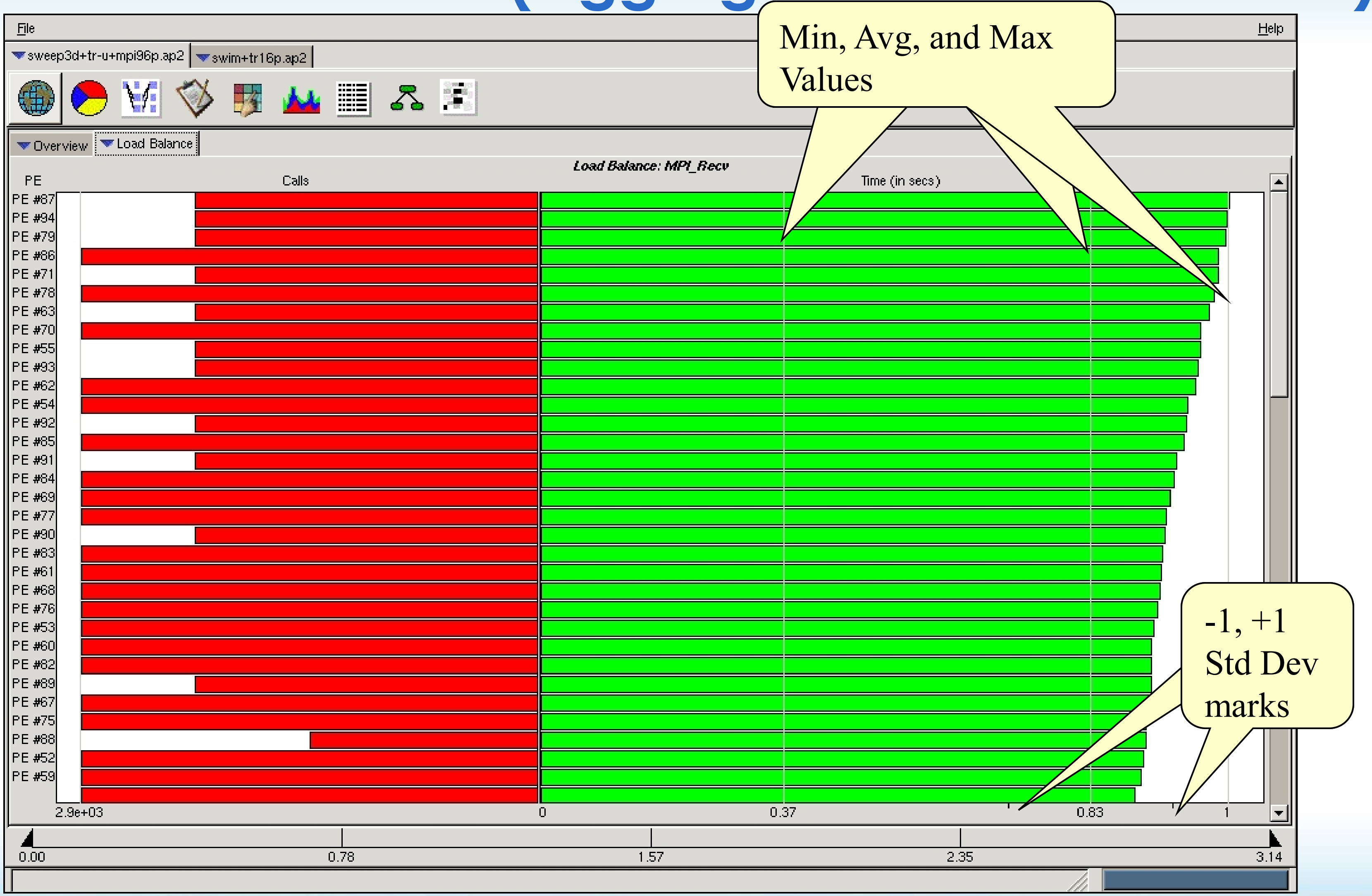


Statistics Overview

Switch Overview display



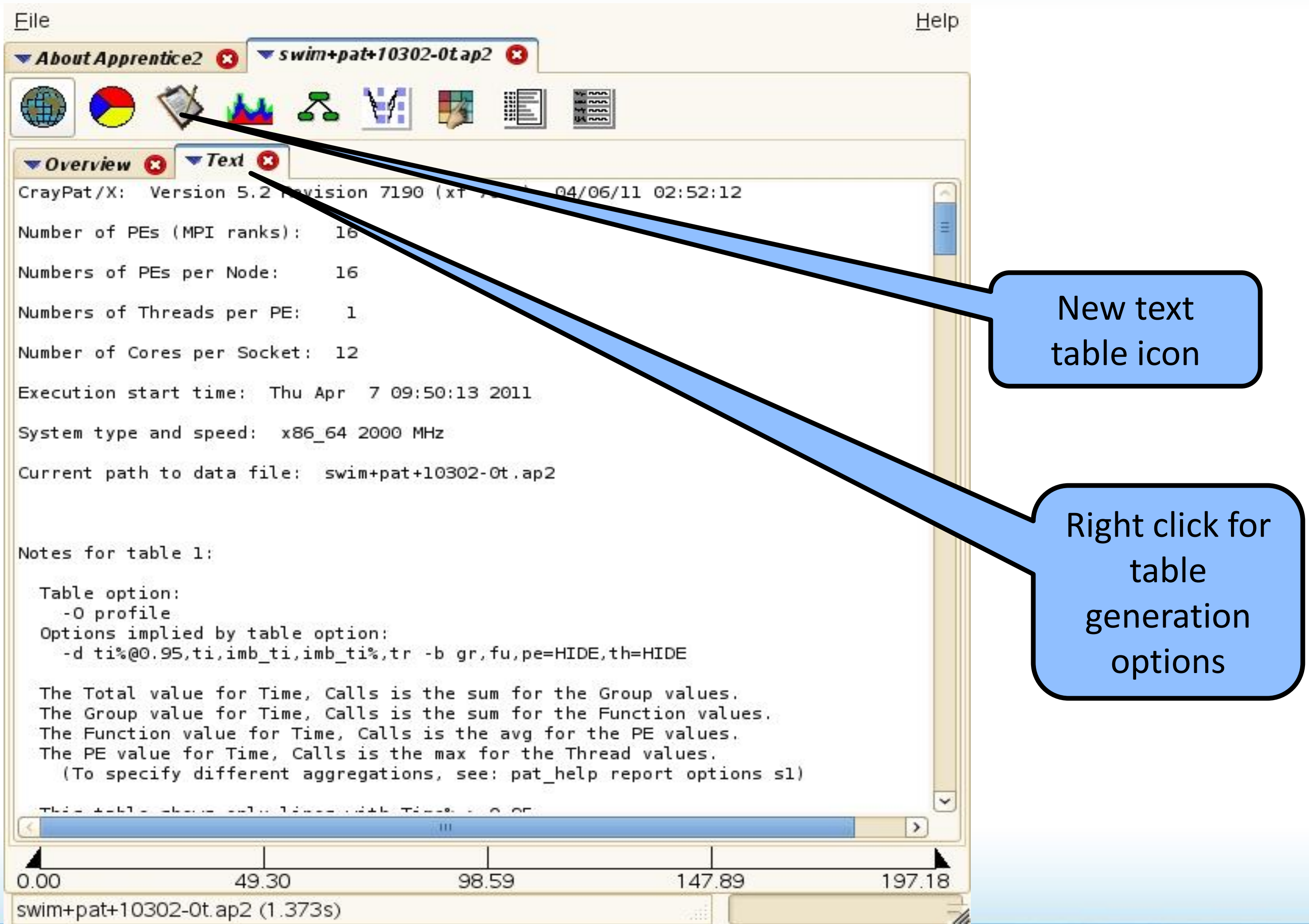
Load Balance View (Aggregated from Overview)



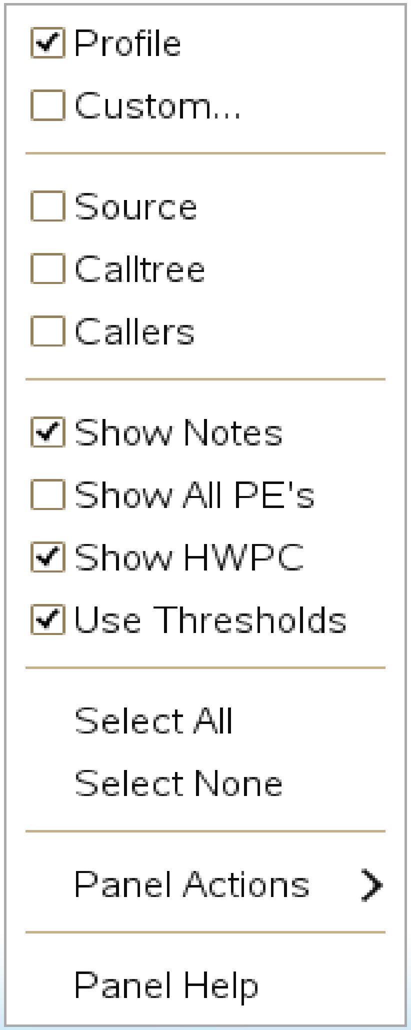
pat_report Tables in Cray Apprentice2

- 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

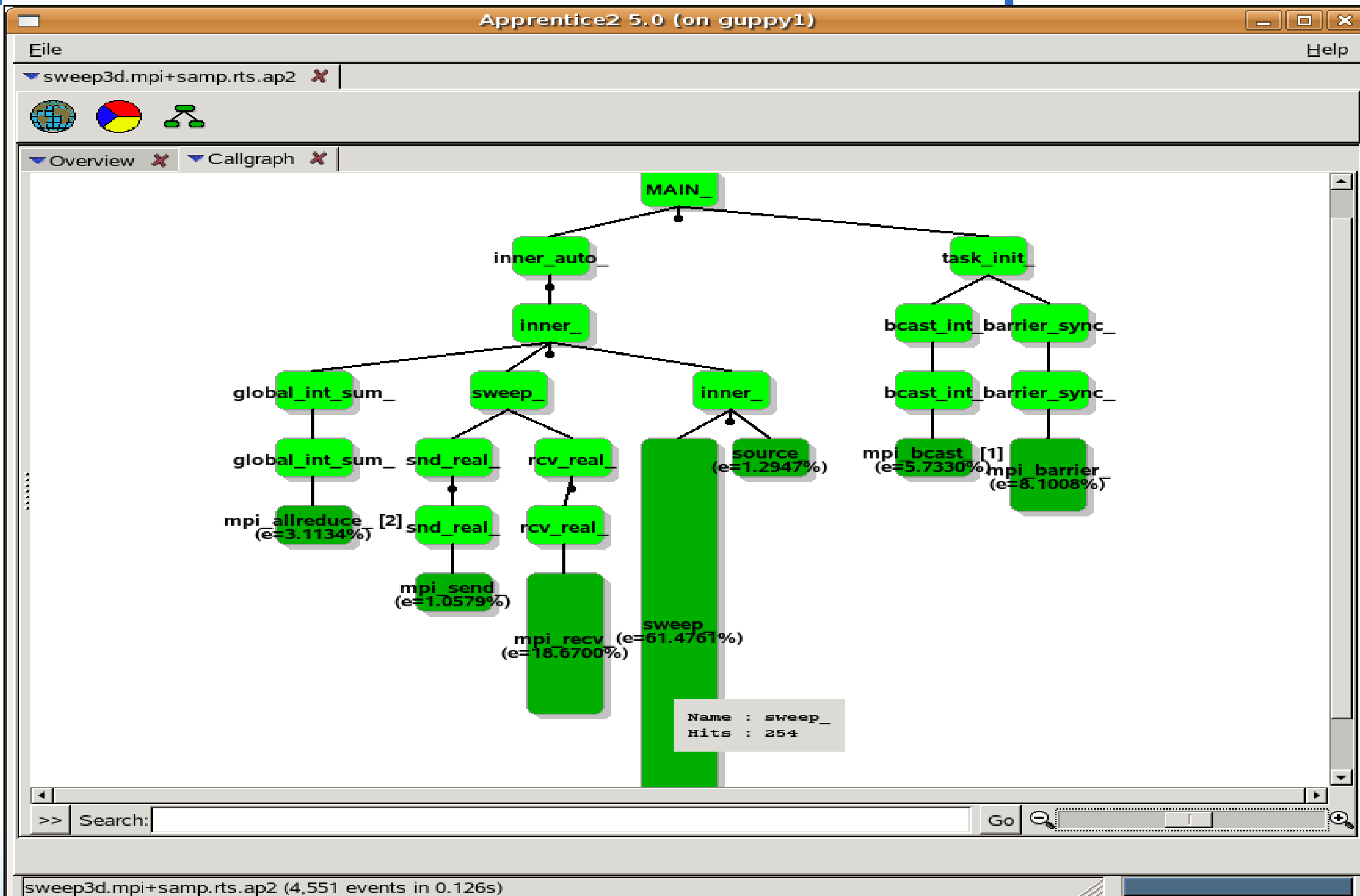
Example of pat_report Tables in Cray Apprentice2



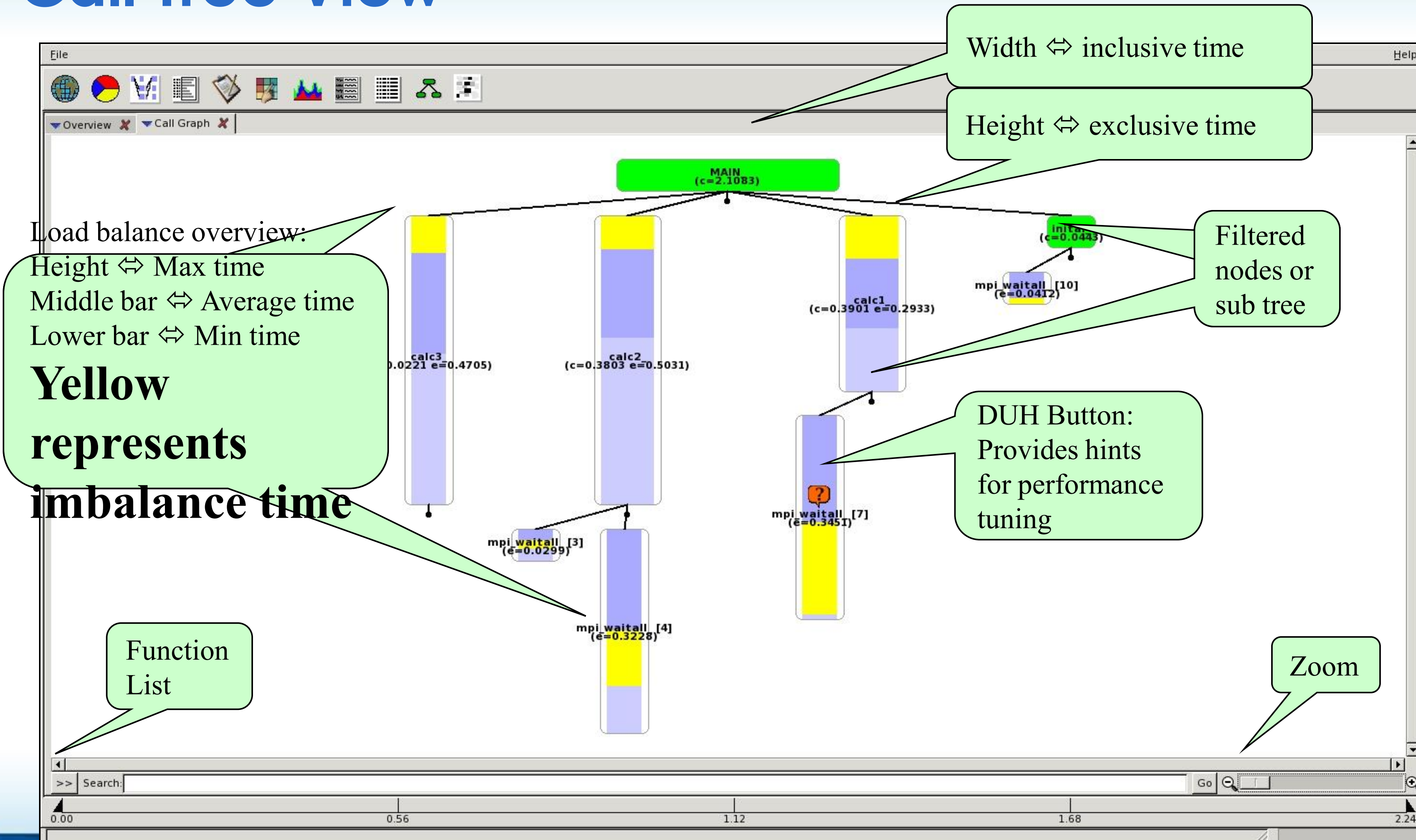
Generating New pat_report Tables



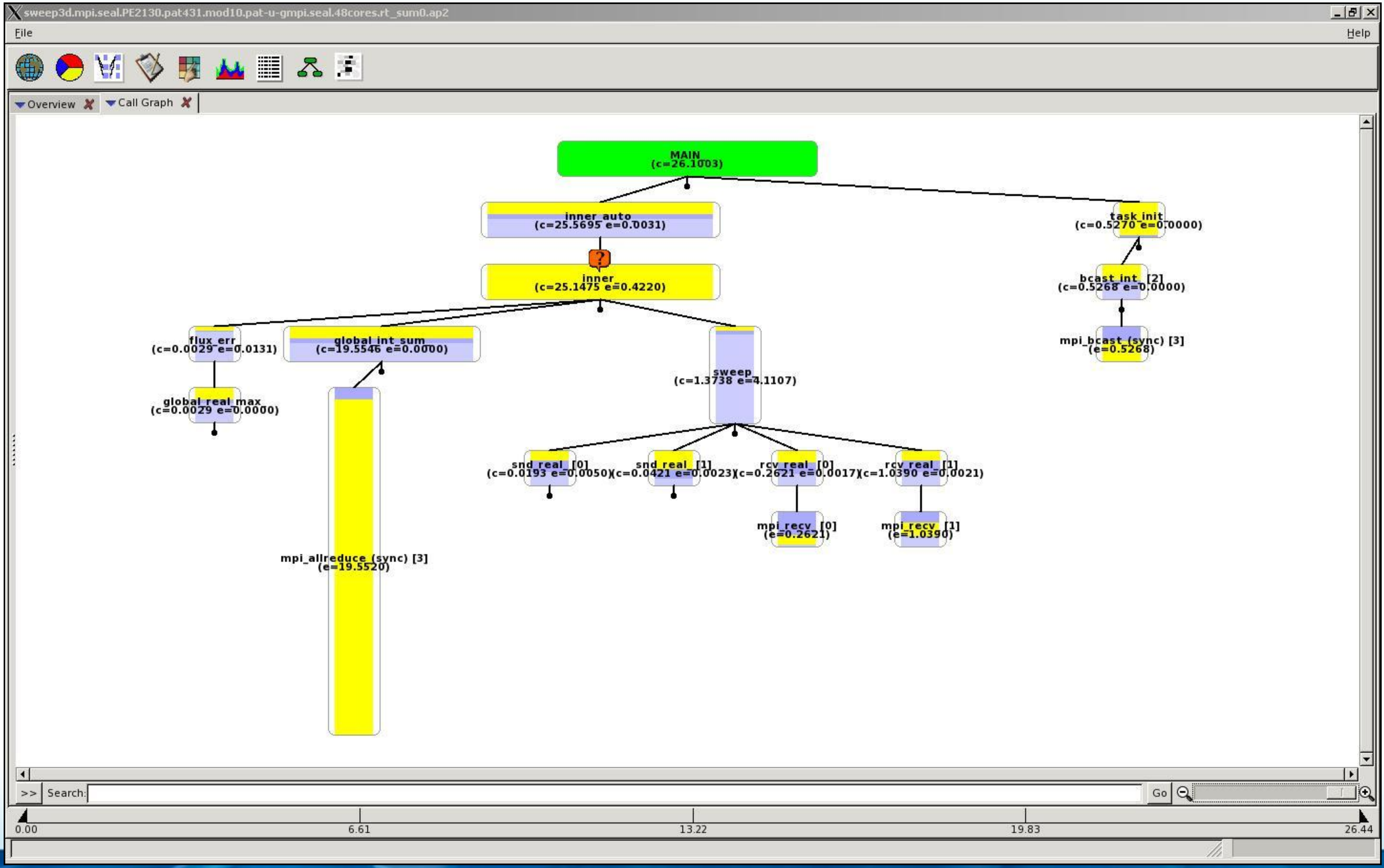
Apprentice2 : Calltree View of Sampled Data



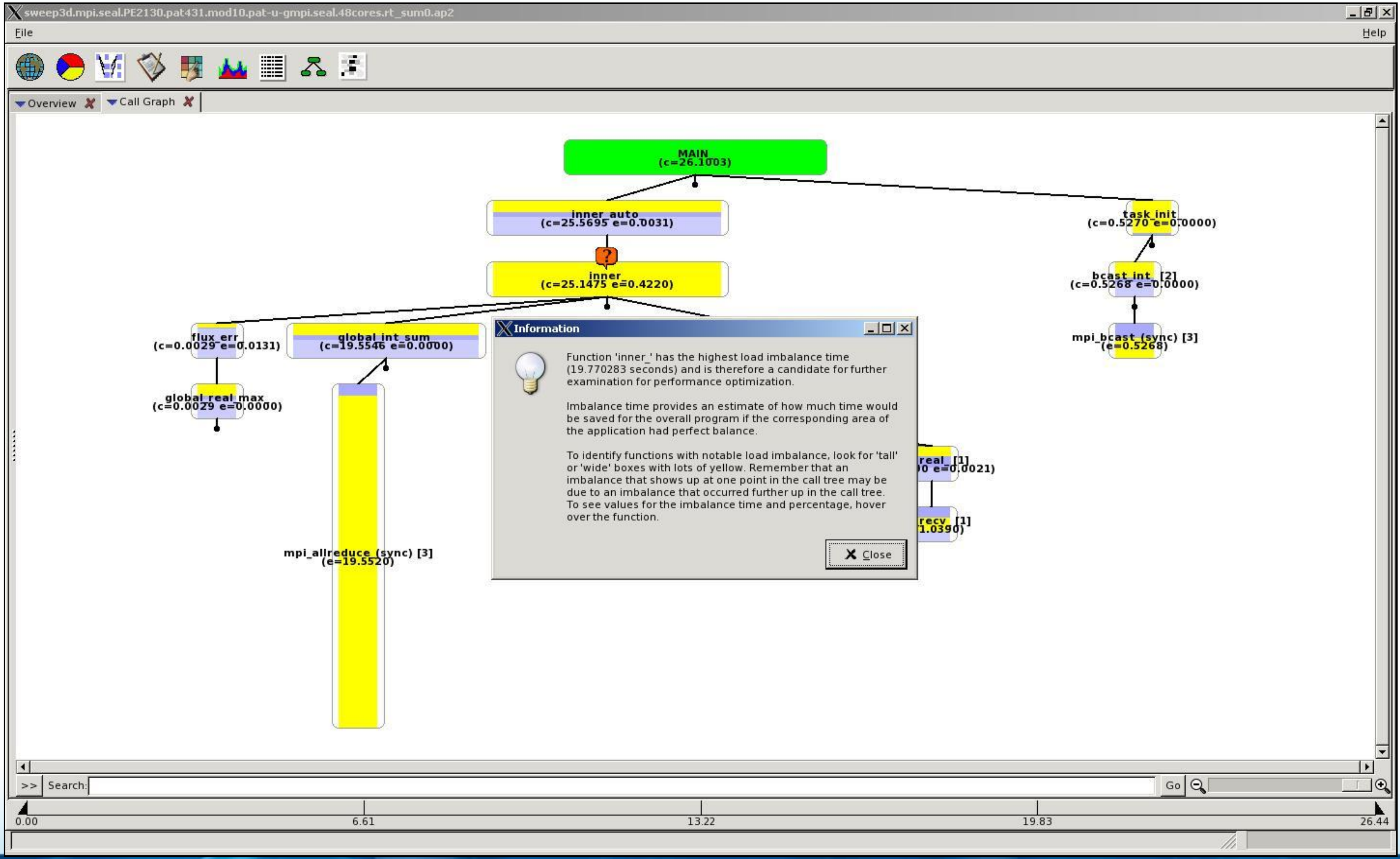
Call Tree View



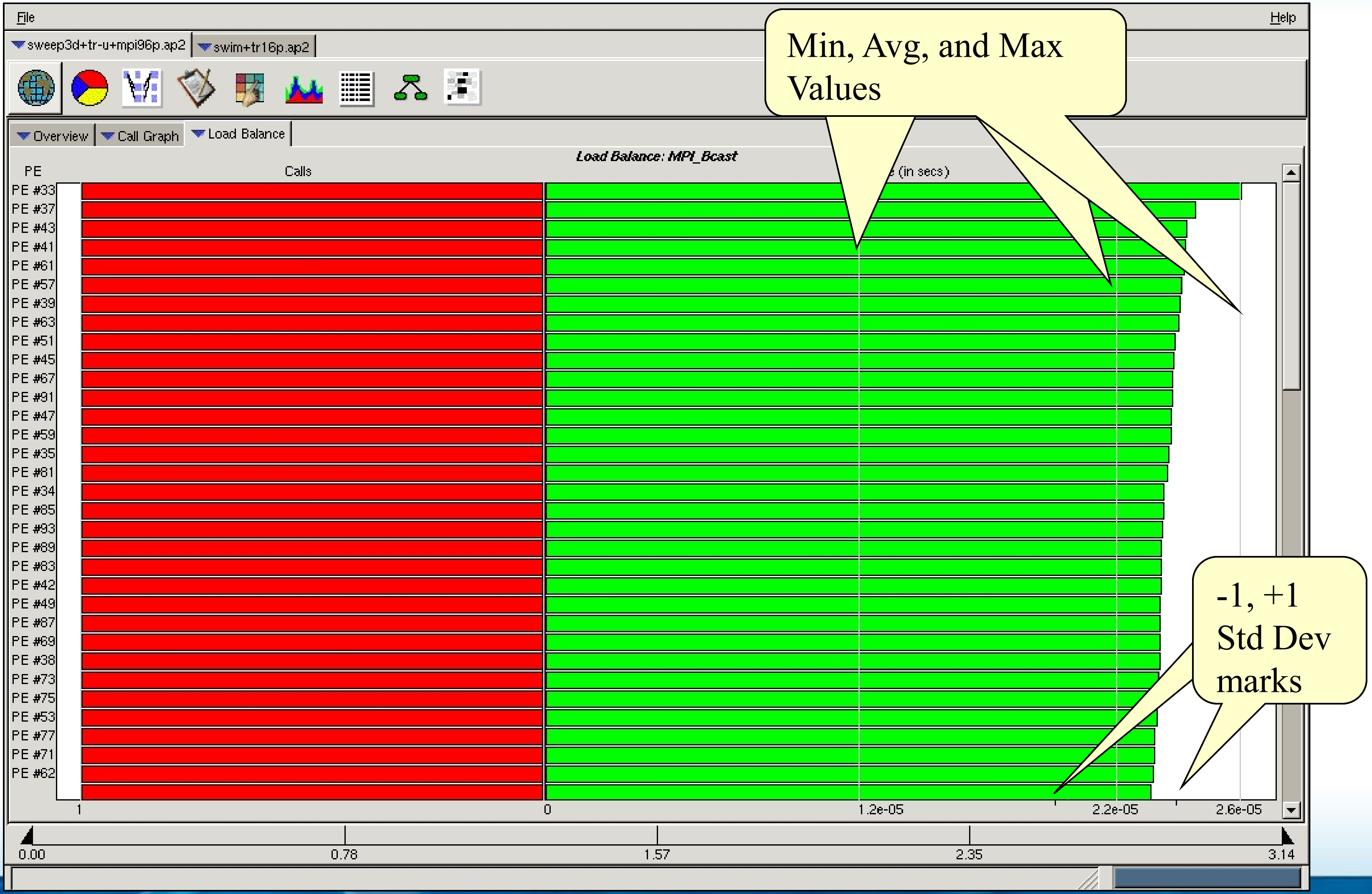
Call Tree Visualization



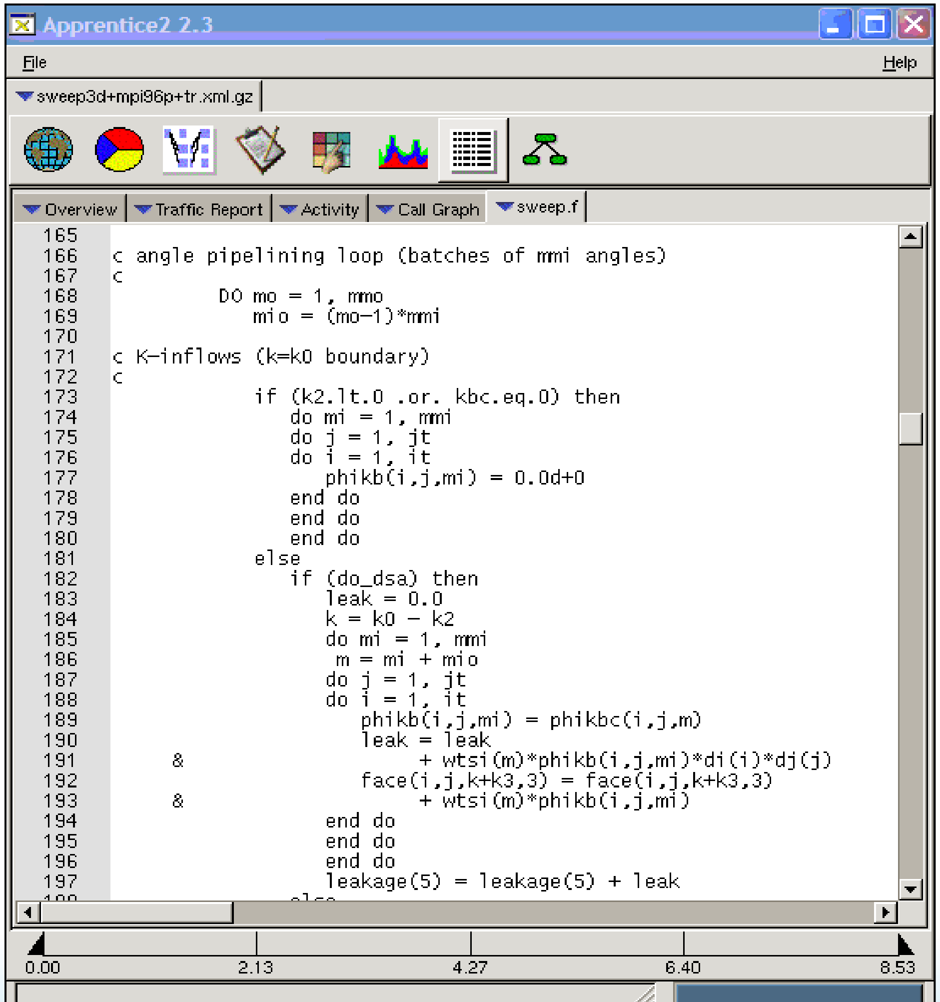
Discrete Unit of Help (DUH Button)



Load Balance View (from Call Tree)

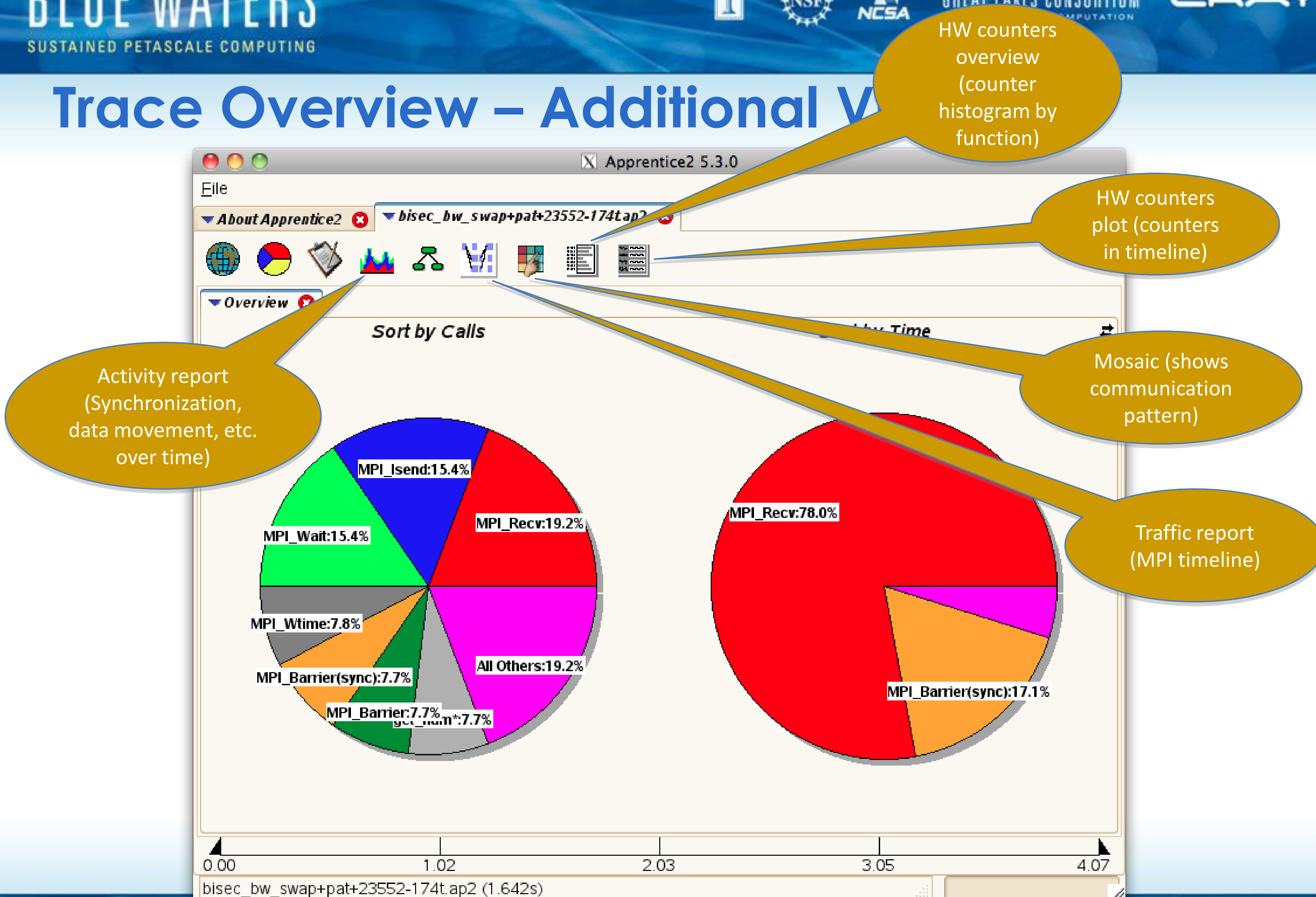


Source Mapping from Call Tree

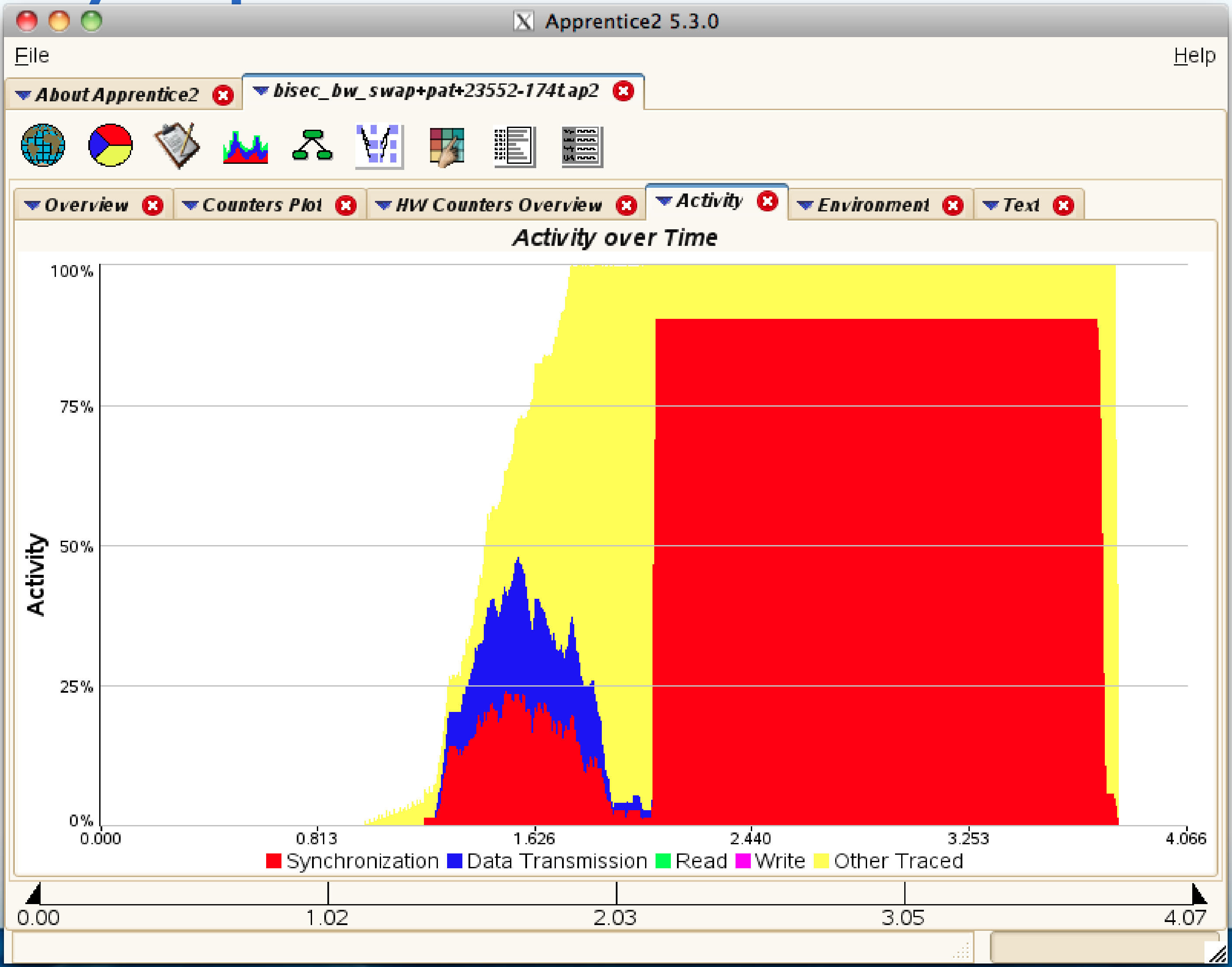


Full Trace Visualization with Cray Apprentice2

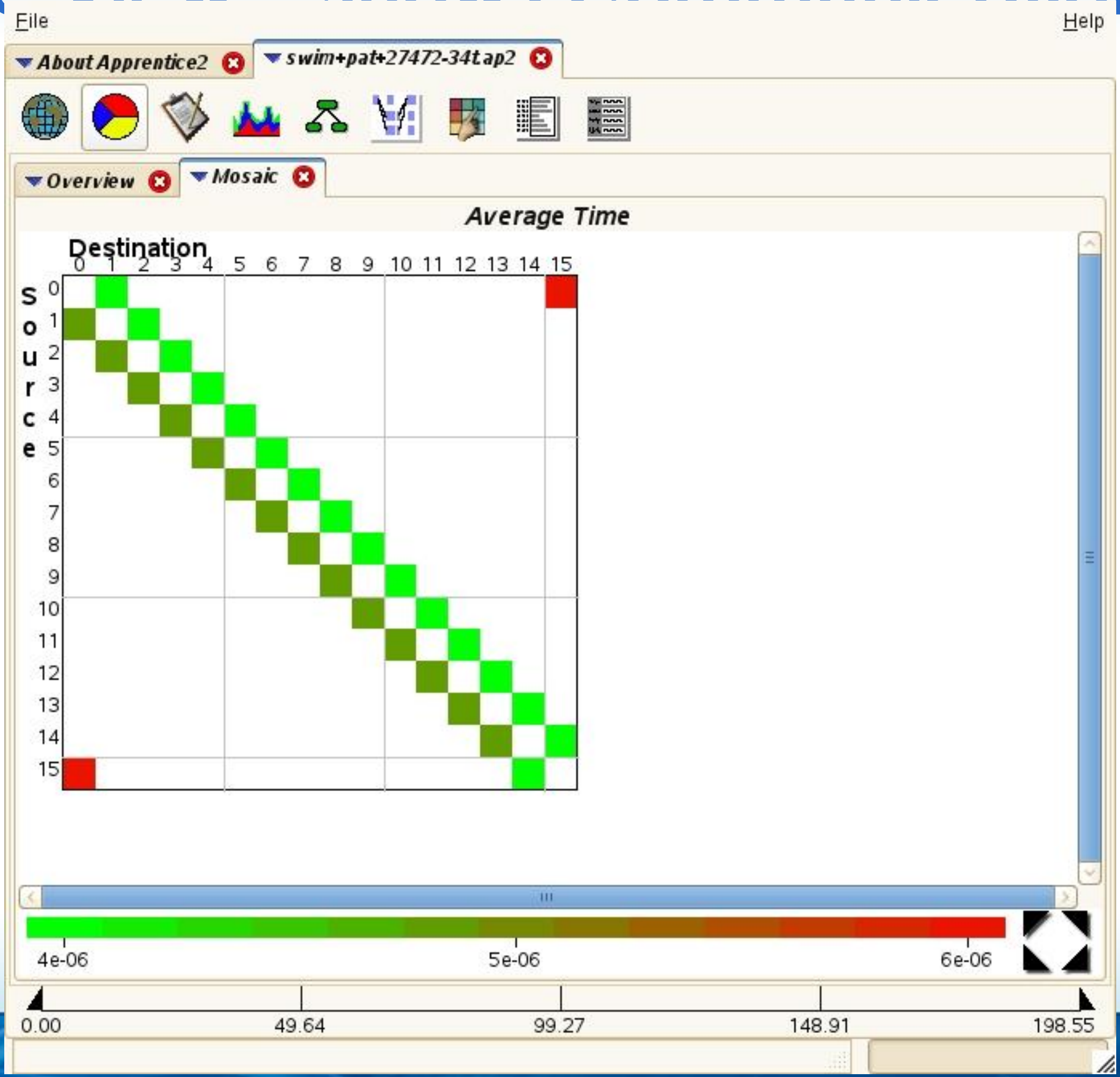
Trace Overview – Additional V



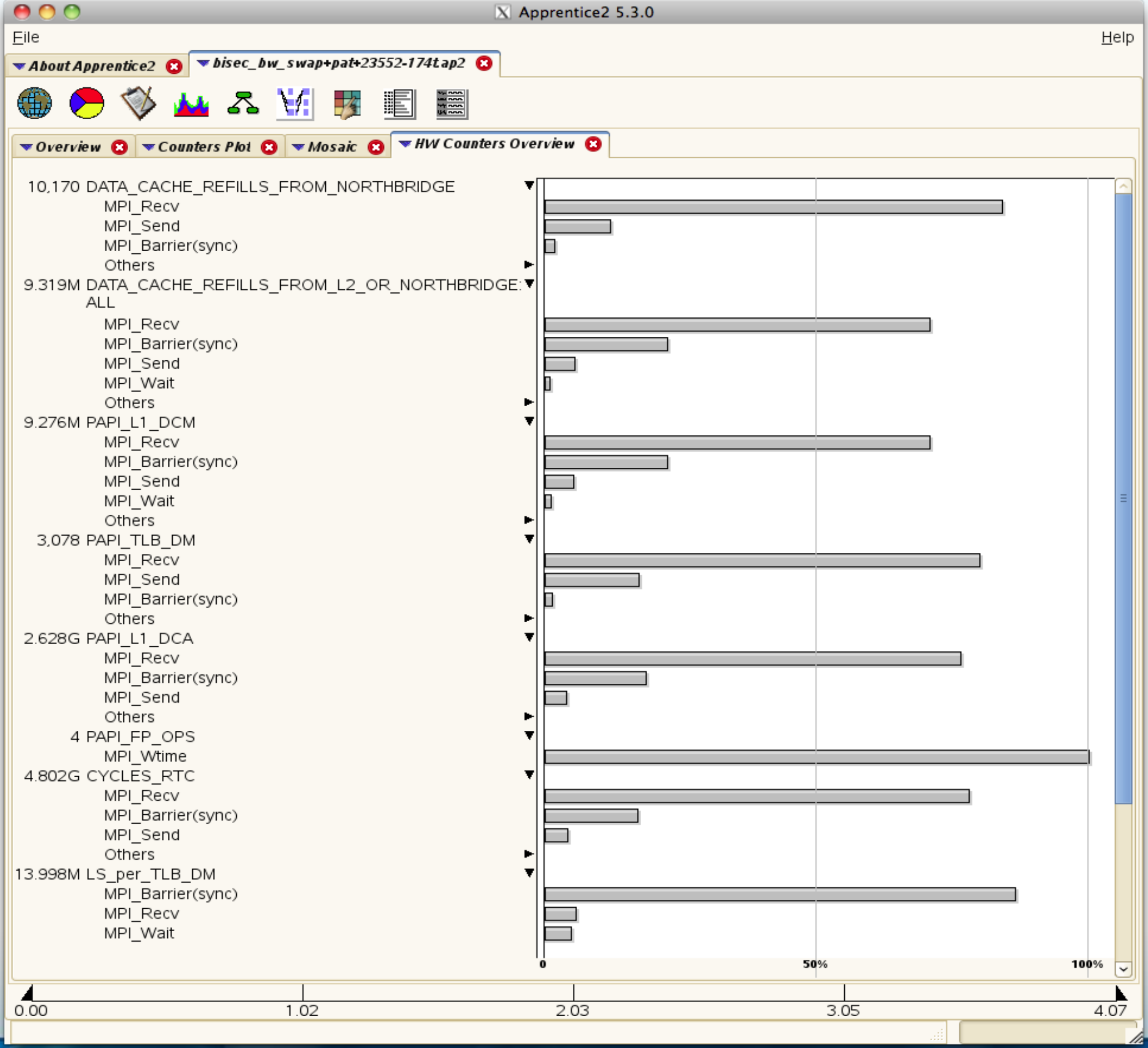
Activity Report



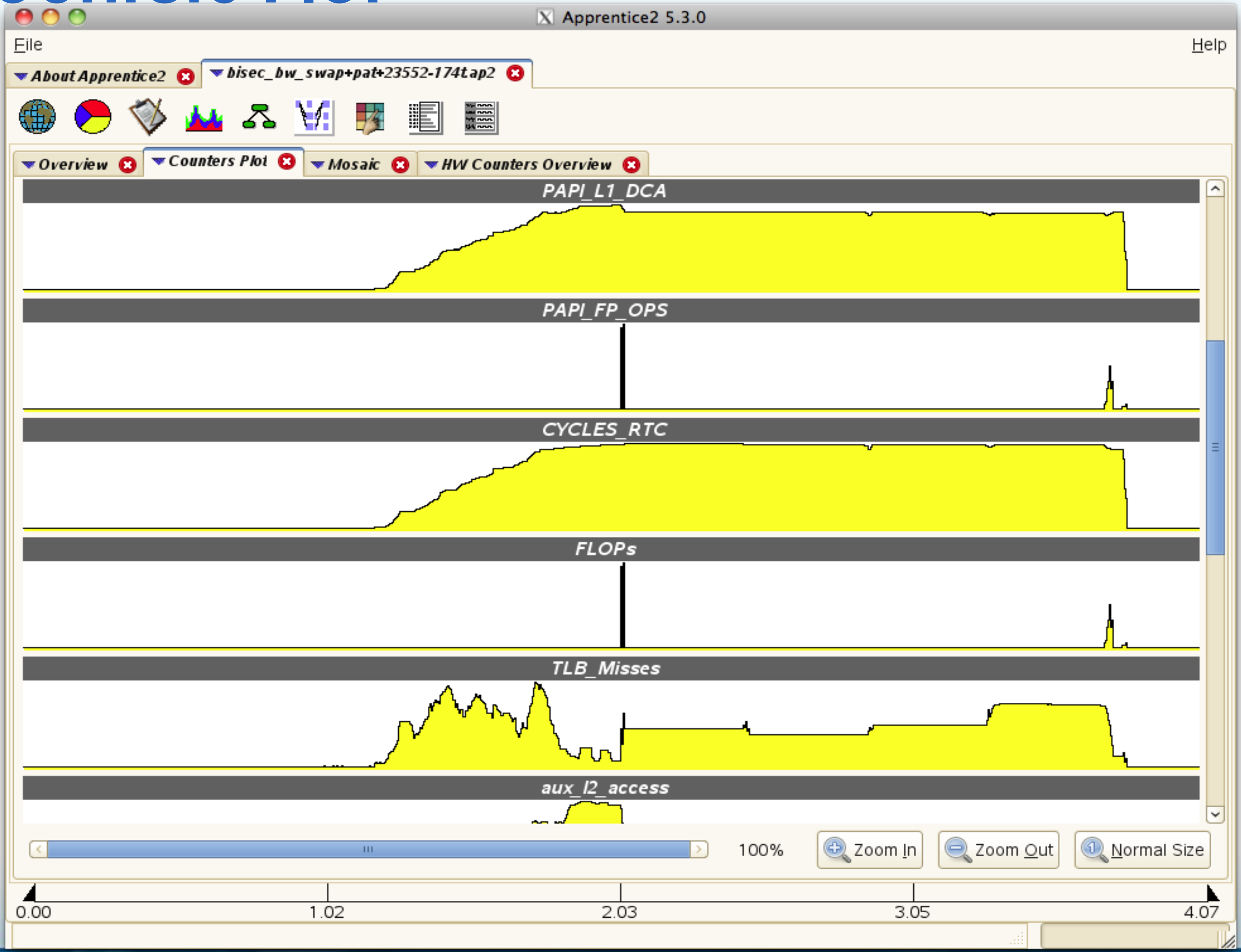
Mosaic View – Shows Communication Pattern



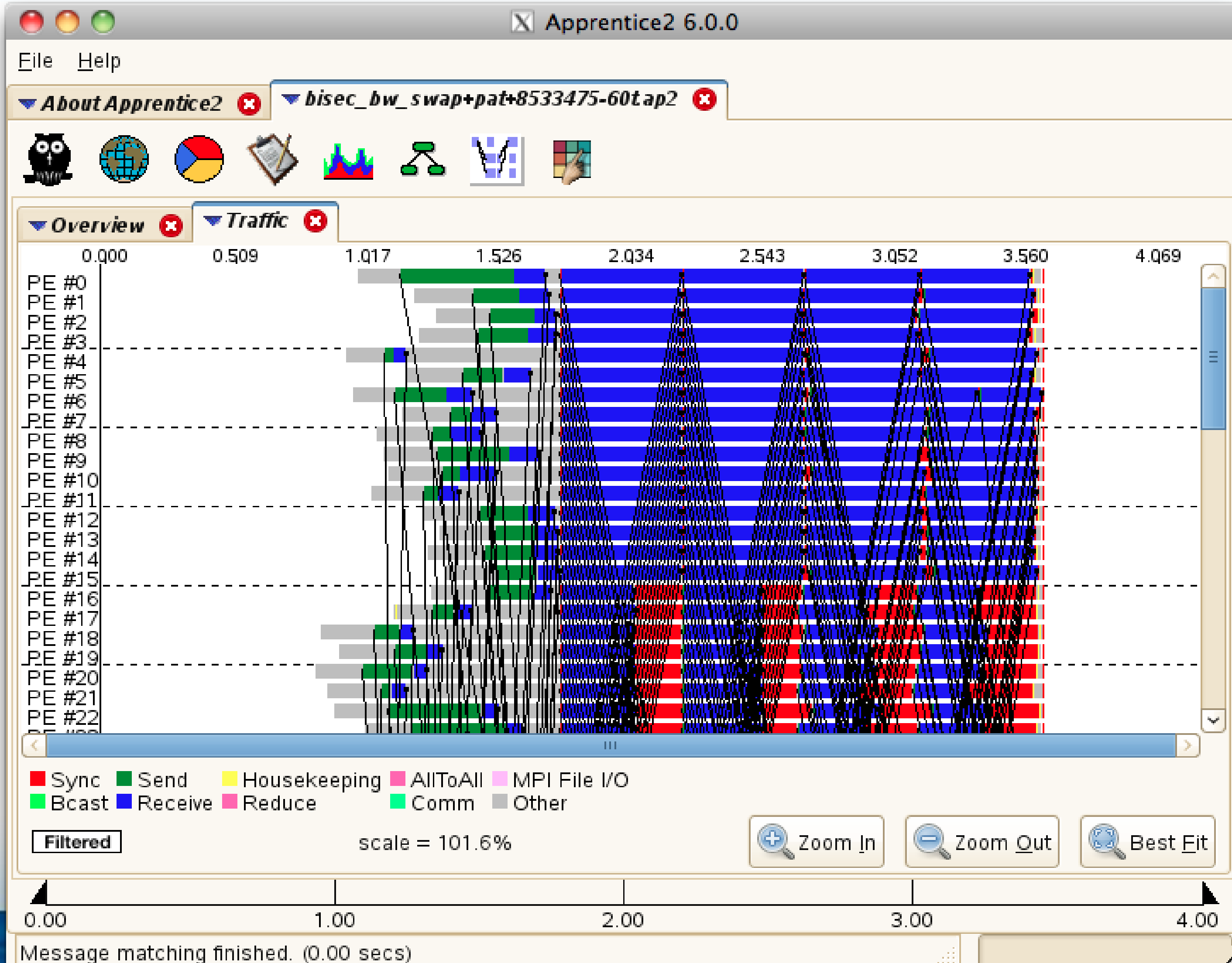
HW Counters Overview



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 Order	On-Node Bytes/PE	On-Node Bytes/PE% of Total Bytes/PE	MPICH_RANK_REORDER_METHOD
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.
#

0,532,64,564,32,572,96,540,8,596
,72,524,40,604,24,588
104,556,16,628,80,636,56,620,48,
516,112,580,88,548,120,612
1,403,65,435,33,411,97,443,9,467
,25,499,105,507,41,475
73,395,81,427,57,459,17,419,113,
491,49,387,89,451,121,483
6,436,102,468,70,404,38,412,14,4
44,46,476,110,508,78,500
86,396,30,428,62,460,54,492,118,
420,22,452,94,388,126,484
129,563,193,531,161,571,225,539
,241,595,233,523,249,603,185,55
5
153,587,169,627,137,635,201,619
,177,515,145,579,209,547,217,61
1
7,405,71,469,39,437,103,413,47,4
45,15,509,79,477,31,501
111,397,63,461,55,429,87,421,23,
493,119,389,95,453,127,485
134,402,198,434,166,410,230,442
,238,466,174,506,158,394,246,47
4
190,498,254,426,142,458,150,386
,182,418,206,490,214,450,222,48
2
128,533,192,541,160,565,232,525
,224,573,240,597,184,557,248,60
5
168,589,200,517,152,629,136,549
,176,637,144,621,208,581,216,61
3
5,439,37,407,69,447,101,415,13,4
71,45,503,29,479,77,511
53,399,85,431,21,463,61,391,109,
423,93,455,117,495,125,487
2,530,34,562,66,538,98,522,10,57
0,42,554,26,594,50,602
18,514,74,586,58,626,82,546,106,
634,90,578,114,618,122,610
135,315,167,339,199,347,259,307
,231,371,239,379,191,331,247,29
9
175,363,159,323,143,355,255,291
,207,275,183,283,151,267,215,22
3
133,406,197,438,165,470,229,414
,245,446,141,478,237,502,253,39
8
157,510,189,462,173,430,205,390
,149,422,213,454,181,494,221,48
6
130,316,260,340,194,372,162,348
,226,308,234,380,242,332,250,30
0
202,364,186,324,154,356,138,292
,170,276,178,284,210,218,268,14
6
4,535,36,543,68,567,100,527,12,5
99,44,575,28,559,76,607
52,591,20,631,60,639,84,519,108,
623,92,551,116,583,124,615
3,440,35,432,67,400,99,408,11,46
4,43,496,27,472,51,504
19,392,75,424,59,456,83,384,107,
416,91,488,115,448,123,480
132,401,196,441,164,409,228,433
,236,465,204,473,244,393,188,49
7
252,505,140,425,212,457,156,385
,172,417,180,449,148,489,220,48
1
131,534,195,542,163,566,227,526
,235,574,203,598,243,558,187,60
6
251,590,211,630,179,638,139,622
,155,550,171,518,219,582,147,61
4
761,660,737,652,705,668,745,692
,673,700,641,684,713,644,753,72
4
729,732,681,756,721,716,764,676
,697,748,689,657,740,665,649,70
8
760,528,736,536,704,560,744,520
,672,568,712,592,752,552,640,60
0
728,584,680,624,720,512,696,632
,688,616,664,544,608,656,648,57
6
762,659,738,651,706,667,746,643
,714,691,674,699,754,683,730,72
3
722,731,763,658,642,755,739,675
,707,650,682,715,698,666,690,74
7
257,345,265,313,281,305,273,337
,609,369,577,377,617,329,513,52
9
545,297,633,361,625,321,585,537
,601,289,553,353,593,521,569,56
1
256,373,261,341,264,349,280,317
,272,381,269,309,285,333,277,36
5
352,301,320,325,288,357,328,304
,360,312,376,293,296,368,336,34
4
258,338,266,346,282,314,274,370
,766,306,710,378,742,330,678,36
2
646,298,750,322,718,354,758,290
,734,662,686,670,726,702,694,65
4
262,375,263,343,270,311,271,351
,286,319,278,342,287,350,279,37
4
294,318,358,383,359,310,295,382
,326,303,327,367,366,335,302,33
4
765,661,709,663,741,653,711,669
,767,655,743,671,749,695,679,70
3
677,727,751,693,647,701,717,687
,757,685,733,725,719,735,645,75
9

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

Thank You

BLUE WATERS

SUSTAINED PETASCALE COMPUTING

May 22, 2013

ATP

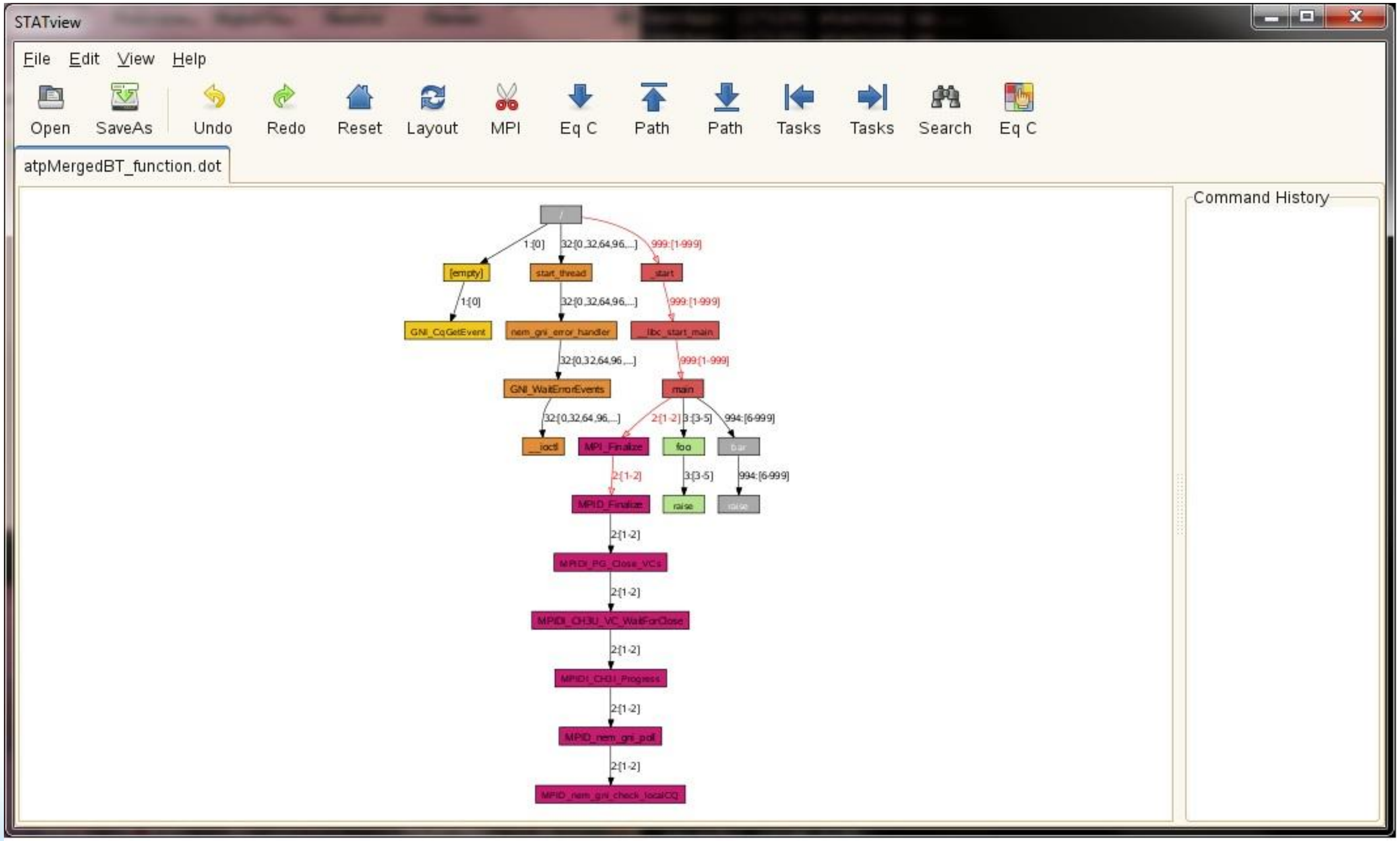
GREAT LAKES CONSORTIUM FOR PETASCALE COMPUTATION

Abnormal Termination Processing (ATP)

- Useful when a user wants to know where his/her application crashes (i.e. stack backtrace when crashing).
- ATP gathers all of the stack backtraces into a merged stack backtrace tree and (1) output to job stderr output file, (2) writes to disk as the file "atpMergedBT.dot".
- To use ATP:
  - At compile time:
    - module load atp # this is default
    - Compile code with “-g”
  - In job script, before aprun command, add:
    - export ATP_ENABLED=1

ATP (cont.)

- atpMergedBT.dot can be viewed with statview GUI tool



BLUE WATERS

SUSTAINED PETASCALE COMPUTING

May 22, 2013

Stack Trace Analysis Tool on Blue Waters

Craig Steffen SEAS group

csteffen@ncsa.illinois.edu

GREAT LAKES CONSORTIUM FOR PETASCALE COMPUTATION

How to Use Stack Trace Analysis Tool (STAT)

- Find the MOM node where your job is running
- Ssh to that MOM node
  - > ssh nidXXXX
- > module load stat
- > cd /scratch/my/job/dir/
- > mkdir stat_info_$PBS_JOBID
- > cd stat_info_$PBS_JOBID

How to use STAT (continued)

- > ps -aux | grep my_login_name
  - [find the pid of your aprun command]
- > STAT 123456
  Attaching to job launcher and launching tool daemons...
  ....
  Results written to /scratch/my/job/dir/stat_info_222333/…

To Visualize STAT data:

- Log onto login node with X forwarding
- > module load stat
- > cd [to where data is]
- STATview XXXYYY.dot

Usage information for STAT (after "module load stat")

- csteffen@h2ologin2 23:33 ~ $ man STAT
- Man: find all matching manual pages
- * STAT (1)
- stat (1+)
- stat (2)
- Man: What manual page do you want?
- Man:
- csteffen@h2ologin2 23:33 ~ $

How to find MOM node for your job: Have the job script “phone home”:

```
echo 'about to run solver'
touch running_on_host_`hostname`
date
export PAT_RT_HWPC="PAPI_FP_OPS,PAPI_TOT_INS"
aprun -n 1536 -N 32 -cc 0-7,8-15,16-23,24-31
    ./$solver_exec_name
```

Start job and wait for it to run

- csteffen@h2ologin2 00:04 ~/specfem3d/SF3DG_csteffen $ qsub run_1536.sh
- 364139.nid00221
- csteffen@h2ologin2 00:04 ~/specfem3d/SF3DG_csteffen $ qstat | grep csteffen
- 364132.nid00221 specfem3d_globe csteffen 00:00:02 C batch
- 364139.nid00221 specfem3d_globe csteffen 0 **R** batch

SSH to the running MOM node

```
csteffen@h2ologin2 00:05 ~/specfem3d/SF3DG_csteffen $ ls -lrt running_on*
-rw------- 1 csteffen bw_staff 0 May 19 23:53 running_on_host_nid23054
-rw------- 1 csteffen bw_staff 0 May 20 00:05 running_on_host_nid25261
csteffen@h2ologin2 00:05 ~/specfem3d/SF3DG_csteffen $ ssh nid25261
```

On MOM node, find PID of my aprun

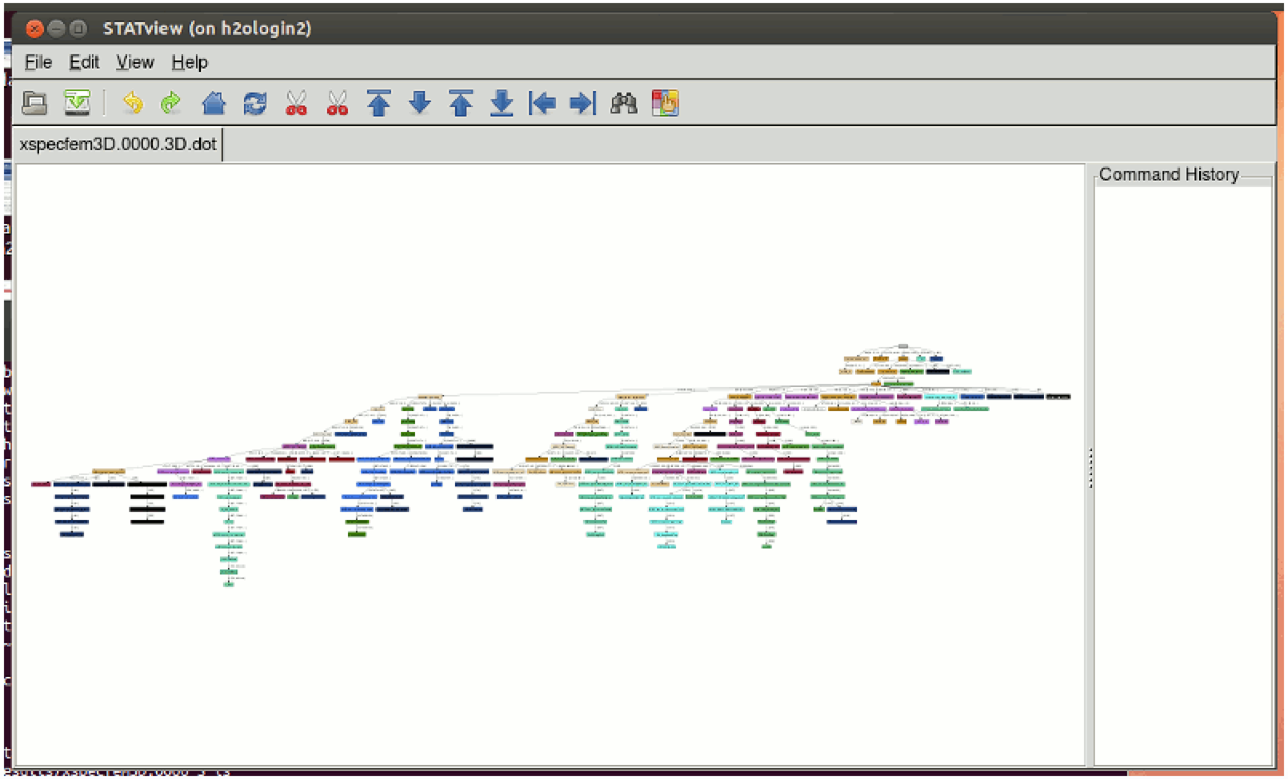
```
csteffen@nid25261 00:06 ~ $ ps -aux | grep csteffen | grep aprun
Warning: bad ps syntax, perhaps a bogus '-'? See
   http://procps.sf.net/faq.html
csteffen 23786  0.0  0.0  22316  4360 ?        S    00:05   0:00
   /usr/bin/perl /sw/xe/altd/bin/aprun -n 1536 -N 32 -cc 0-7,8-
   15,16-23,24-31 ./xspecfem3D
csteffen 23807  0.0  0.0  28804  2168 ?        S    00:05   0:00
   /usr/bin/aprun -n 1536 -N 32 -cc 0-7,8-15,16-23,24-31
   ./xspecfem3D
csteffen 23957  0.0  0.0   5624   864 pts/0    S+   00:06   0:00
   grep aprun
csteffen@nid25261 00:06 ~ $ module load stat
csteffen@nid25261 00:06 ~ $ STAT 23807
Attaching to job launcher (null):23807 and launching tool
   daemons...
```

Trace files available for later analysis

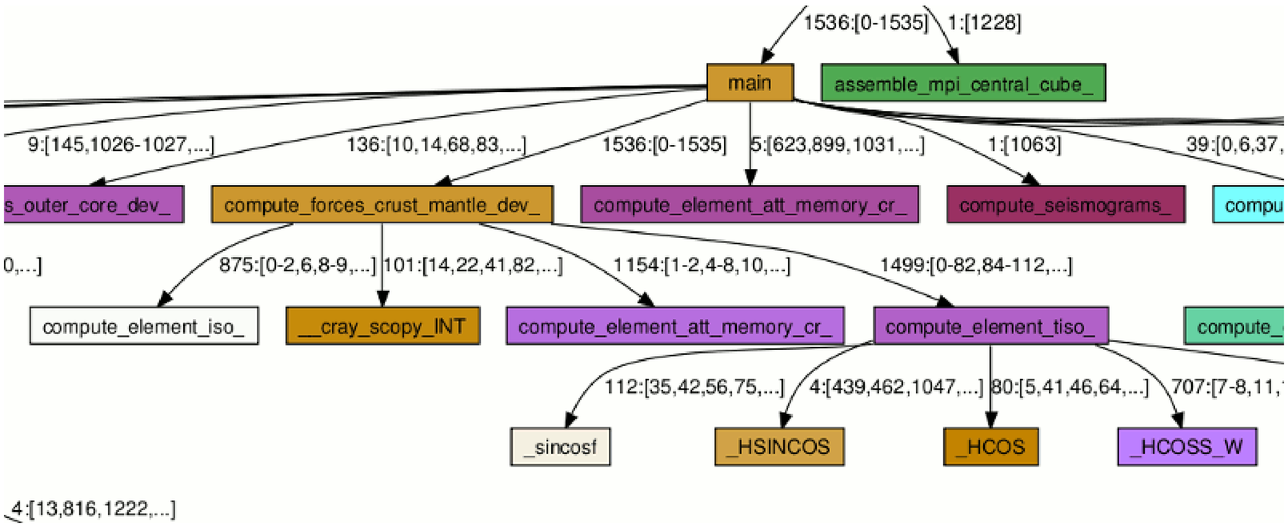
```
Results written to /mnt/a/u/staff/csteffen/stat_results/xspecfem3D.0000

csteffen@h2ologin2 00:10 ~/stat_results/xspecfem3D.0000 $ ls -lrt
total 124
-rw-r--r-- 1 csteffen bw_staff    12 May 20 00:07 xspecfem3D.0000.top
-rw-r--r-- 1 csteffen bw_staff 48057 May 20 00:07 xspecfem3D.0000.ptab
-rw-r--r-- 1 csteffen bw_staff   634 May 20 00:07 xspecfem3D.0000.fulltop
-rw-r--r-- 1 csteffen bw_staff  1265 May 20 00:07 xspecfem3D.0000.perf
-rw-r--r-- 1 csteffen bw_staff 64140 May 20 00:07 xspecfem3D.0000.3D.dot
csteffen@h2ologin2 00:10 ~/stat_results/xspecfem3D.0000 $
csteffen@h2ologin2 00:13 ~/stat_results/xspecfem3D.0000 $ module load stat
csteffen@h2ologin2 00:13 ~/stat_results/xspecfem3D.0000 $ STATview
   xspecfem3D.0000.3D.dot
```

STATview


STATview (on h2ologin2)
File Edit View Help
xspecfem3D.0000.3D.dot
Command History

STATview displays call trees and occupancies



BLUE WATERS

SUSTAINED PETASCALE COMPUTING

May 22, 2013

Distributed Debugging Tool (DDT) on Blue Waters

Craig Steffen SEAS group

csteffen@ncsa.illinois.edu

GREAT LAKES CONSORTIUM FOR PETASCALE COMPUTATION

BLUE WATERS
SUSTAINED PETASCALE COMPUTING

Why DDT?

- Complete Graphical Debugger
  - Traps data values
  - Across-execution visualizers
  - Drops user into source automatically
  - Synchronization and deadlock checking
- Only requires symbols (-g) to work
- Very parallel (hundreds of thousands of ranks)
- Has useful annotations for optimized-out source

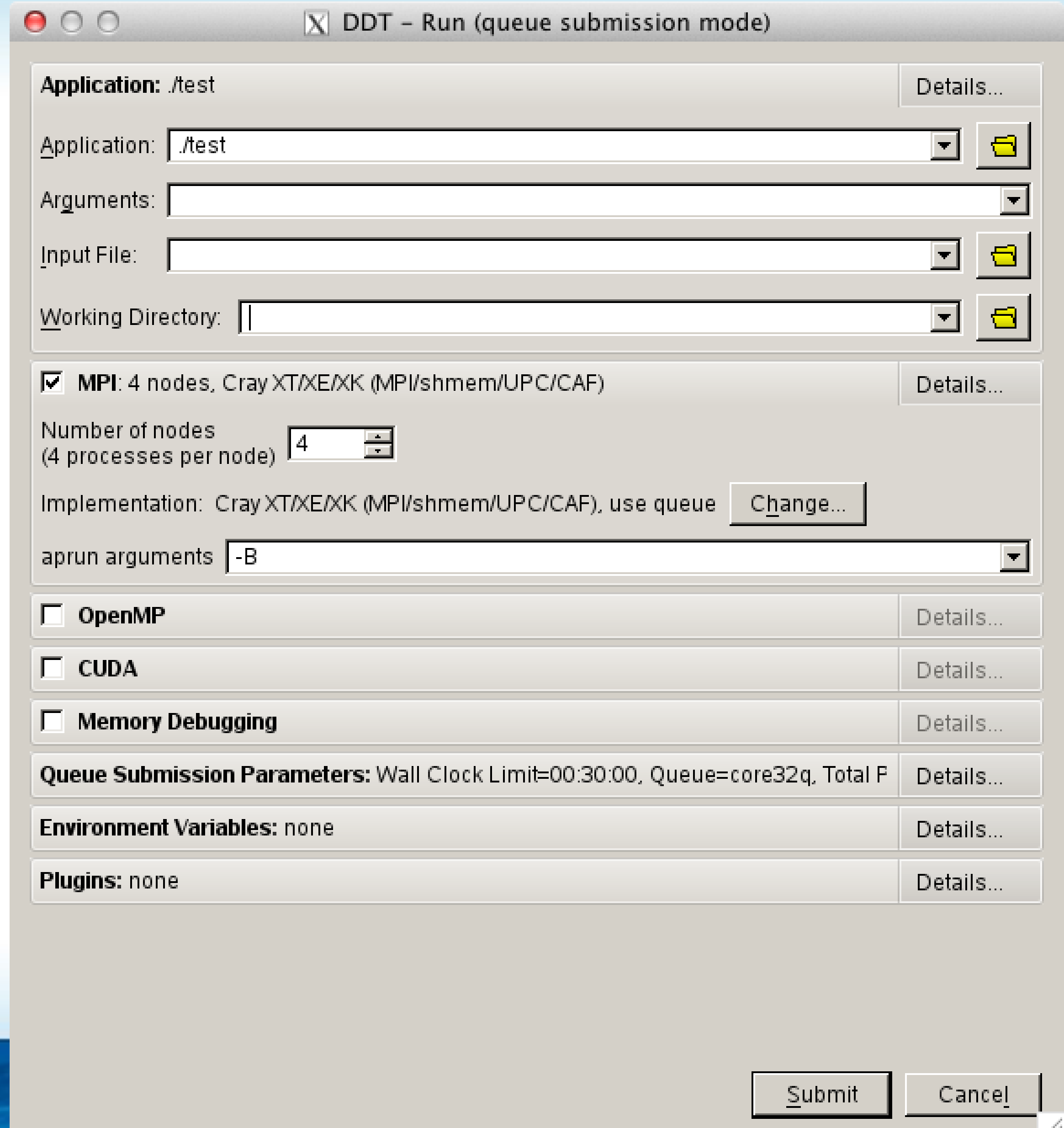
HowTo 0:

> Module load ddt

> ddt

HowTo 1: click on "Run and Debug Program"

- DDT understands qsub and job submission
- It launches job and executable
- DDT will start a debug session automatically as soon as the job starts.



HowTo 2: Manual Launch

- To launch a program manually click on *Manually Launch a Program* button.
- Select how many processes you want to debug and click on Listen . At this point start a program or programs using the following command:
  - > ddt-client <path-to-program-binary>
- Debugging begins when all executables are running
- Used for applications with multiple, separate, communicating executables

HowTo 3: Attach to Running Executable

- click on Attach to a Running Program button.
- DDT will start scanning each of 64 mom nodes to locate active jobs owned by you. If there are more than one active job DDT will find all of them. Once DDT finds the desired job select it and click on Attach to listed processes button.

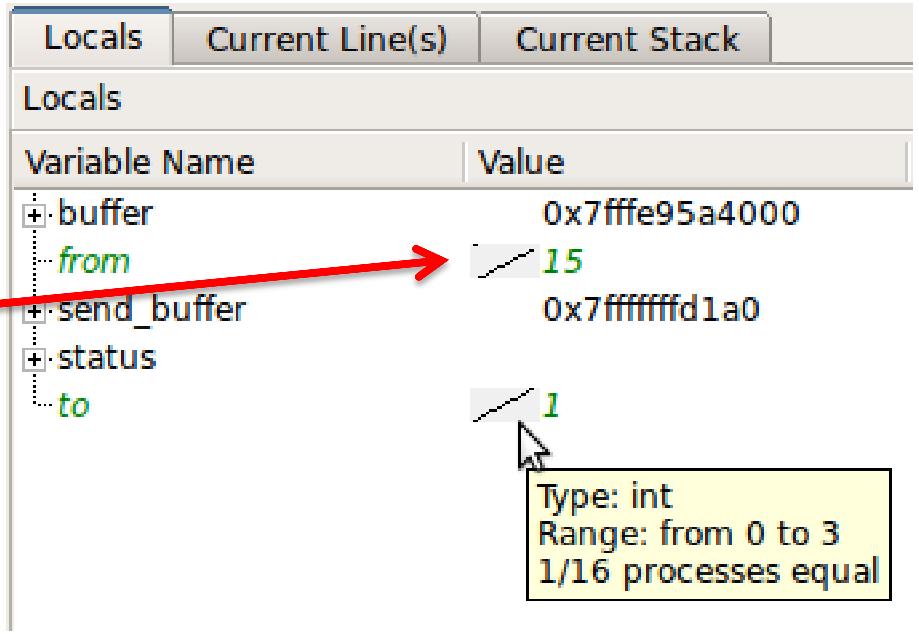
HowTo 4: Start DDT as an interactive job

- > qsub -I –X [interactive job with X forwarding]
- > module load ddt
- > ddt –noqueue
- Click on Run and Debug a Program . A new window with expandable tabs will appear.

- Click on Run button to start a debug session.

DDT: debugging displays

- Points out faults in crashes
- Memory debugging tool detects leaks and out-of-bounds
- “sparklines” graphical data value summary across all ranks
- Good source code navigator and controls



BLUE WATERS

SUSTAINED PETASCALE COMPUTING

May 22, 2013

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.

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.