Cray Performance Measurement and Analysis Tools

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

PE Workshop

October 12-13, 2011

Day 2 – More on Cray Performance Tools

- Programming model and language support (MPI, PGAS, OpenMP, SHMEM)
- Other interesting performance data
- Trace analysis and visualization
- Where to get help
- Example: analyzing the performance of an application
- A peak at GPU support

Cray Inc. Proprietary

Programming Model and Language Support

Performance Measurement and Analysis

- Load imbalance
  - Identifies computational code regions and synchronization calls that could benefit most from load balance optimization (some processes have less work than others, some are waiting longer on barriers, etc)
  - Estimates savings if corresponding section of code were balanced
  - MPI sync time (determines late arrivers to barriers)
  - MPI rank placement suggestions (maximize on-node communication)
  - Imbalance metrics (user functions, MPI functions, OpenMP threads)

Cray Inc. Proprietary

Motivation for Load Imbalance Analysis

- Increasing system software and architecture complexity
  - Current trend in high end computing is to have systems with tens of thousands of processors
    - This is being accentuated with multi-core processors
- Applications have to be very well balanced In order to perform at scale on these MPP systems
  - Efficient application scaling includes a balanced use of requested computing resources
- Desire to minimize computing resource “waste”
  - Identify slower paths through code
  - Identify inefficient “stalls” within an application

Cray Inc. Proprietary

MPI Sync Time

- Measure load imbalance in programs instrumented to trace MPI functions to determine if MPI ranks arrive at collectives together
- Separates potential load imbalance from data transfer
- Sync times reported by default if MPI functions traced
- If desired, PAT_RT_MPI_SYNC=0 deactivates this feature

Imbalance Time

- Metric based on execution time
- It is dependent on the type of activity:
  - User functions

    Imbalance time = Maximum time – Average time
  - Synchronization (Collective communication and barriers)

    Imbalance time = Average time – Minimum time
- Identifies computational code regions and synchronization calls that could benefit most from load balance optimization
- Estimates how much overall program time could be saved if corresponding section of code had a perfect balance
  - Represents upper bound on “potential savings”
  - Assumes other processes are waiting, not doing useful work while slowest member finishes

Imbalance %

$$\text{Imbalance\%} = 100 \times \frac{\text{Imbalance time}}{\text{Max Time}} \times \frac{N}{N - 1}$$

- Represents % of resources available for parallelism that is "wasted"
- Corresponds to % of time that rest of team is not engaged in useful work on the given function
- Perfectly balanced code segment has imbalance of 0%
- Serial code segment has imbalance of 100%

Cray Inc. Proprietary

Load Distribution

PGAS Support

- Profiles of a PGAS program can be created to show:
  - Top time consuming functions/line numbers in the code
  - Load imbalance information
  - Performance statistics attributed to user source by default
  - Can expose statistics by library as well
    - To see underlying operations, such as wait time on barriers

- Data collection is based on methods used for MPI library
  - PGAS data is collected by default when using Automatic Profiling Analysis (pat_build –O apa)
  - Predefined wrappers for runtime libraries (caf, upc, pgas) enable attribution of samples or time to user source

- UPC and SHMEM heap tracking available
  - `-g heap` will track shared heap in addition to local heap

PGAS Default Report Table 1

```
Table 1:  Profile by Function

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

 100.0% |   48 |   -- |     -- |Total
|------------------------------------------
|  95.8% |   46 |   -- |     -- |USER
||-----------------------------------------
||  83.3% |   40 | 1.00 |   3.3% |all2all
||   6.2% |    3 | 0.50 |  22.2% |do_cksum
||   2.1% |    1 | 1.00 |  66.7% |do_all2all
||   2.1% |    1 | 0.50 |  66.7% |mpp_accum_long
||   2.1% |    1 | 0.50 |  66.7% |mpp_alloc
||=========================================
|   4.2% |    2 |   -- |     -- |ETC
||-----------------------------------------
||   4.2% |    2 | 0.50 |  33.3% |bzero
|==========================================
```

PGAS Default Report Table 2

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

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

 100.0% |   48 |   -- |     -- |Total
|--------------------------------------------
|  95.8% |   46 |   -- |     -- |USER
||-------------------------------------------
||  83.3% |   40 |   -- |     -- |all2all
3|        |      |      |        | mpp_bench.c
4|        |      |      |        |  line.298
||   6.2% |    3 |   -- |     -- |do_cksum
3|        |      |      |        | mpp_bench.c
||||-----------------------------------------
4|||   2.1% |    1 | 0.25 |  33.3% |line.315
4|||   4.2% |    2 | 0.25 |  16.7% |line.316
||||=========================================
```

PGAS Report Showing Library Functions with Callers

```
Table 1:  Profile by Function and Callers, with Line Numbers
 Samp % | Samp |Group
        |      | Function
        |      |  Caller
        |      |   PE='HIDE'
 100.0% |   47 |Total
|---------------------------
|  93.6% |   44 |ETC
||--------------------------
||  85.1% |   40 |upc_memput
3|        |      | all2all:mpp_bench.c:line.298
4|        |      |  do_all2all:mpp_bench.c:line.348
5|        |      |   main:test_all2all.c:line.70
||   4.3% |    2 |bzero
3|        |      | (N/A):(N/A):line.0
||   2.1% |    1 |upc_all_alloc
3|        |      | mpp_alloc:mpp_bench.c:line.143
4|        |      |  main:test_all2all.c:line.25
||   2.1% |    1 |upc_all_reduceUL
3|        |      | mpp_accum_long:mpp_bench.c:line.185
4|        |      |  do_cksum:mpp_bench.c:line.317
5|        |      |   do_all2all:mpp_bench.c:line.341
6|        |      |    main:test_all2all.c:line.70
||==========================
```

OpenMP Data Collection and Reporting

- Measure overhead incurred entering and leaving
  - Parallel regions
  - Work-sharing constructs within parallel regions
- Show per-thread timings and other data
- Trace entry points automatically inserted by Cray and PGI compilers
  - Provides per-thread information
- Can use sampling to get performance data without API (per process view… no per-thread counters)
  - Run with OMP_NUM_THREADS=1 during sampling

OpenMP Data Collection and Reporting (2)

- Load imbalance calculated across all threads in all ranks for mixed MPI/OpenMP programs
  - Can choose to see imbalance to each programming model separately
- We need to add tracing support for barriers (both implicit and explicit)
  - Need support from compilers
- Data displayed by default in pat_report (no options needed)
  - Focus on where program is spending its time
  - Assumes all requested resources should be used

Cray Inc. Proprietary

Imbalance Options for Data Display (pat_report –O ...)

- profile_pe.th (default view)
  - Imbalance based on the set of all threads in the program
- profile_pe_th
  - Highlights imbalance across MPI ranks
  - Uses max for thread aggregation to avoid showing under-performers
  - Aggregated thread data merged into MPI rank data
- profile_th_pe
  - For each thread, show imbalance over MPI ranks
  - Example: Load imbalance shown where thread 4 in each MPI rank didn't get much work

Cray Inc. Proprietary

Profile by Function Group and Function (with –T)

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

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

 100.0% | 12.548996 |       -- |      -- |  7944.7 |Total
|-------------------------------------------------------------------
|  97.8% | 12.277316 |       -- |      -- |  3371.8 |USER
||------------------------------------------------------------------
||  35.6% |  4.473536 | 0.072259 |    1.6% |    498.0 |calc3_.LOOP@li.96
||  29.1% |  3.653288 | 0.070551 |    1.9% |    500.0 |calc2_.LOOP@li.74
||  28.3% |  3.545677 | 0.056303 |    1.6% |    500.0 |calc1_.LOOP@li.69
. . .
||==================================================================
|   1.2% |  0.155028 |       -- |      -- |  1000.5 |MPI_SYNC
||------------------------------------------------------------------
||   1.2% |  0.154899 | 0.674518 |   82.0% |    999.0 |mpi_barrier_(sync)
||   0.0% |  0.000129 | 0.000489 |   79.8% |      1.5 |mpi_reduce_(sync)
||==================================================================
|   0.7% |  0.082943 |       -- |      -- |  3197.2 |MPI
||------------------------------------------------------------------
||   0.4% |  0.047471 | 0.158820 |   77.6% |    999.0 |mpi_barrier_
||   0.1% |  0.015157 | 0.295055 |   95.9% |    297.1 |mpi_waitall_
. . .
||==================================================================
|   0.3% |  0.033683 |       -- |      -- |   374.5 |OMP
||------------------------------------------------------------------
||   0.1% |  0.013098 | 0.078620 |   86.4% |    125.0 |calc2_.REGION@li.74(ovhd)
||   0.1% |  0.010298 | 0.052760 |   84.3% |    124.5 |calc3_.REGION@li.96(ovhd)
||   0.1% |  0.010287 | 0.068428 |   87.6% |    125.0 |calc1_.REGION@li.69(ovhd)
||==================================================================
|   0.0% |  0.000027 | 0.000128 |   83.0% |      0.8 |PTHREAD
|        |           |          |         |          | pthread_create
|==================================================================
```

OpenMP Parallel DOs
<function>.<region>@<line>
automatically instrumented

OpenMP overhead is normally small and is filtered out on the default report (< 0.5%). When using "–T" the filter is deactivated

Hardware Counters Information at Loop Level

```
========================================================================
USER / calc3_.LOOP@li.96
------------------------------------------------------------------------
  Time%                                                  37.3%
  Time                                                6.826587 secs
  Imb.Time                                            0.039858 secs
  Imb.Time%                                               0.6%
  Calls                             72.9 /sec            498.0 calls
  DATA_CACHE_REFILLS:
    L2_MODIFIED:L2_OWNED:
    L2_EXCLUSIVE:L2_SHARED        64.364M/sec     439531950 fills
  DATA_CACHE_REFILLS_FROM_SYSTEM:
    ALL                           10.760M/sec      73477950 fills
  PAPI_L1_DCM                     64.973M/sec     443686857 misses
  PAPI_L1_DCA                    135.699M/sec     926662773 refs
  User time (approx)               6.829 secs   15706256693 cycles  100.0%Time
  Average Time per Call                           0.013708 sec
  CrayPat Overhead : Time           0.0%
  D1 cache hit,miss ratios         52.1% hits         47.9% misses
  D1 cache utilization (misses)     2.09 refs/miss    0.261 avg hits
  D1 cache utilization (refills)    1.81 refs/refill  0.226 avg uses
  D2 cache hit,miss ratio          85.7% hits         14.3% misses
  D1+D2 cache hit,miss ratio       93.1% hits          6.9% misses
  D1+D2 cache utilization          14.58 refs/miss    1.823 avg hits
  System to D1 refill             10.760M/sec      73477950 lines
  System to D1 bandwidth         656.738MB/sec   4702588826 bytes
  D2 to D1 bandwidth            3928.490MB/sec  28130044826 bytes
========================================================================
```


Cray Inc. Proprietary

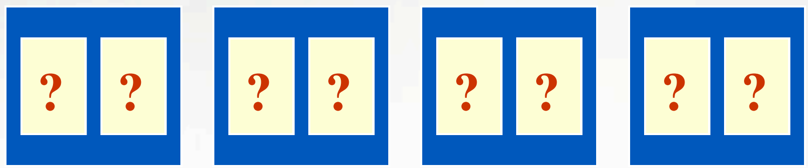
MPI + OpenMP? (some ideas)

- When does it pay to add OpenMP to my MPI code?
  - Add OpenMP when code is network bound
  - Adding OpenMP to memory bound codes may aggravate memory bandwidth issues, but you have more control when optimizing for cache
  - Look at collective time, excluding sync time: this goes up as network becomes a problem
  - Look at point-to-point wait times: if these go up, network may be a problem

Cray Inc. Proprietary

MPI Rank Reorder

- MPI rank placement with environment variable



  - Distributed placement
  - SMP style placement
  - Folded rank placement
  - User provided rank file

Automatic Communication Grid Detection

- Analyze runtime performance data to identify grids in a program to maximize on-node communication
  - Example: nearest neighbor exchange in 2 dimensions
    - Sweep3d uses a 2-D grid for communication
- Determine whether or not a custom MPI rank order will produce a significant performance benefit
- Grid detection is helpful for programs with significant point-to-point communication
- Doesn’t interfere with MPI collective communication optimizations

Automatic Grid Detection (cont'd)

- Tools produce a custom rank order if it's beneficial based on grid size, grid order and cost metric
- Summarized findings in report
- Available if MPI functions traced (-g mpi)
- Describe how to re-run with custom rank order

Cray Inc. Proprietary

Example: Observations and Suggestions

MPI Grid Detection: There appears to be point-to-point MPI communication in a 22 X 18 grid pattern. The 48.6% 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.Custom was generated along with this report and contains the Custom rank order from the following table. This file also contains usage instructions and a table of alternative rank orders.

Rank Order	On-Node Bytes/PE	On-Node Bytes/PE% of Total Bytes/PE	MPICH_RANK_REORDER_METHOD
Custom	7.80e+06	78.37%	3
SMP	5.59e+06	56.21%	1
Fold	2.59e+05	2.60%	2
RoundRobin	0.00e+00	0.00%	0

Cray Inc. Proprietary

MPICH_RANK_ORDER File Example

```
# 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.
…
```


Cray Inc. Proprietary

Loop Statistics

- Helps identify loops to move to optimize:
  - Loop timings approximate how much work exists within a loop
  - Trip counts can be used to help carve up loop on GPU
- Enabled with CCE –h profile_generate option
  - Should be done as separate experiment – compiler optimizations are restricted with this feature
- Loop statistics reported by default in pat_report table

Cray Inc. Proprietary

Collecting Loop Statistics

- Load PrgEnv-cray software
- Load perftools software

- Compile AND link with –h profile_generate

- Instrument binary for tracing
  - pat_build –u my_program or
  - pat_build –w my_program

- Run application
- Create report with loop statistics
  - pat_report my_program.xf > loops_report

Example: Loop Statistics

```
Notes for table 2:
  Table option:
    -O loops
  …
  The Function value for each data item is the avg of the PE values.
    (To specify different aggregations, see:  pat_help report options s1)

  This table shows only lines with Loop Incl Time / Total > 0.0095.
    (To set thresholds to zero, specify:  -T)

Loop instrumentation can interfere with optimizations, so time
  reported here may not reflect time in a fully optimized program.

  Loop stats can safely be used in the compiler directives:
   !PGO$       loop_info est_trips(Avg) min_trips(Min) max_trips(Max)
   #pragma pgo loop_info est_trips(Avg) min_trips(Min) max_trips(Max)

  Explanation of Loop Notes (P=1 is highest priority, P=0 is lowest):
   novec (P=0.5): Loop not vectorized (see compiler messages for reason).
   sunwind (P=1): Loop could be vectorized and unwound.
   vector (P=0.1): Already a vector loop.
```

Profile guided optimization feedback for compiler

Example Loop Stats (2)

Table 2: Loop Stats from -hprofile_generate

Loop Incl Time / Total	Loop Incl Time	Loop Incl Time / Hit	Loop Hit	Loop Trips Avg	Loop Notes	Function=/.LOOP\. PE='HIDE'
24.6%	0.057045	0.000570	100	64.1	novec	calc2_.LOOP.0.li.614
24.0%	0.055725	0.000009	6413	512.0	vector	calc2_.LOOP.1.li.615
18.9%	0.043875	0.000439	100	64.1	novec	calc1_.LOOP.0.li.442
18.3%	0.042549	0.000007	6413	512.0	vector	calc1_.LOOP.1.li.443
17.1%	0.039822	0.000406	98	64.1	novec	calc3_.LOOP.0.li.787
16.7%	0.038883	0.000006	6284	512.0	vector	calc3_.LOOP.1.li.788
9.7%	0.022493	0.000230	98	512.0	vector	calc3_.LOOP.2.li.805
4.2%	0.009837	0.000098	100	512.0	vector	calc2_.LOOP.2.li.640

Other Interesting Performance Data

-g tracegroup (subset)

- blas Basic Linear Algebra subprograms
- CAF Co-Array Fortran (Cray CCE compiler only)
- HDF5manages extremely large and complex data collections
- heap dynamic heap
- io includes stdio and sysio groups
- lapack Linear Algebra Package
- math ANSI math
- mpi MPI
- omp OpenMP API
- omp-rtl OpenMP runtime library (not supported on Catamount)
- pthreads POSIX threads (not supported on Catamount)
- shmem SHMEM
- sysio I/O system calls
- system system calls
- upc Unified Parallel C (Cray CCE compiler only)

For a full list, please see man pat_build

Specific Tables in pat_report

```
heidi@kaibab:/lus/scratch/heidi> pat_report -O -h

pat_report: Help for -O option:
Available option values are in left column, a prefix can be
specified:
  ct                          -O calltree
  defaults                    <Tables that would appear by default.>
  heap                        -O heap_program,heap_hiwater,heap_leaks
  io                          -O read_stats,write_stats
  lb                          -O load_balance
  load_balance                -O lb_program,lb_group,lb_function
  mpi                         -O mpi_callers
  ---
  D1_D2_observation           Observation about Functions with low D1+D2
cache hit ratio
  D1_D2_util                  Functions with low D1+D2 cache hit ratio
  D1_observation              Observation about Functions with low D1
cache hit ratio
  D1_util                     Functions with low D1 cache hit ratio
  TLB_observation             Observation about Functions with low TLB
refs/miss
  TLB_util                    Functions with low TLB refs/miss
```

Heap Statistics

- -g heap
  - calloc, cfree, malloc, free, malloc_trim, malloc_usable_size, mallopt, memalign, posix_memalign, pvalloc, realloc, valloc

- -g heap
- -g sheap
- -g shmem
  - shfree, shfree_nb, shmalloc, shmalloc_nb, shrealloc

- -g upc (automatic with –O apa)
  - upc_alloc, upc_all_alloc, upc_all_free, uc_all_lock_alloc, upc_all_lock_free, upc_free, upc_global_alloc, upc_global_lock_alloc, upc_lock_free

Cray Inc. Proprietary

Heap Statistics

```
Notes for table 5:

  Table option:
    -O heap_hiwater
  Options implied by table option:
    -d am@,ub,ta,ua,tf,nf,ac,ab -b pe=[mmm]

  This table shows only lines with Tracked Heap HiWater MBytes >


Table 5:  Heap Stats during Main Program

 Tracked |  Total |  Total | Tracked | Tracked |PE[mmm]
    Heap | Allocs | Frees  | Objects |  MBytes |
 HiWater |        |        |     Not |     Not |
  MBytes |        |        |   Freed |   Freed |

   9.794 |    915 |   910 |       4 |   1.011 |Total
|-----------------------------------------------------
|    9.943 |   1170 |  1103 |      68 |   1.046 |pe.0
|    9.909 |    715 |   712 |       3 |   1.010 |pe.22
|    9.446 |   1278 |  1275 |       3 |   1.010 |pe.43
|=====================================================
```

Trace Analysis and Visualization

Tracing

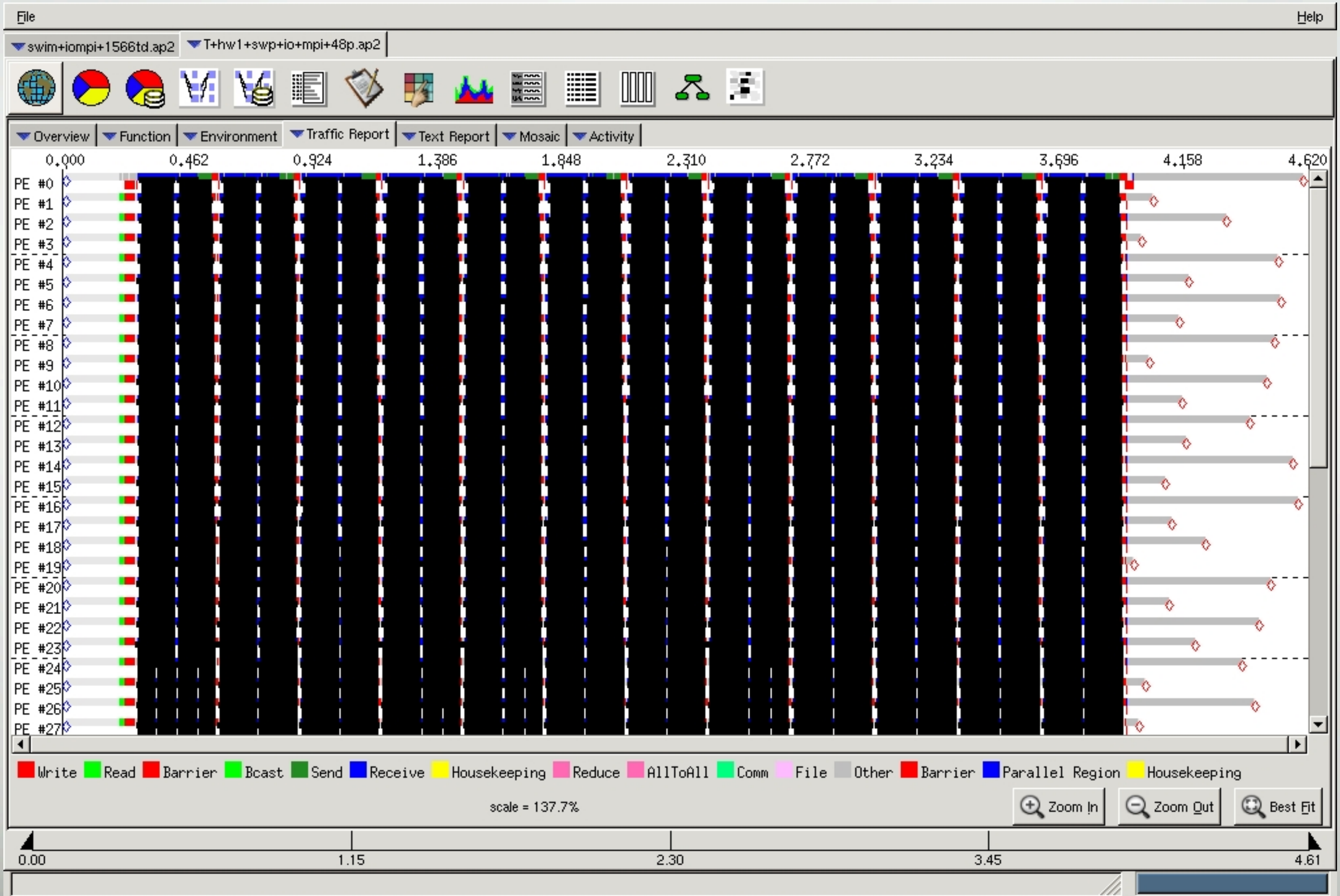
- Only true function calls can be traced
  - Functions that are inlined by the compiler or that have local scope in a compilation unit cannot be traced

- Enabled with pat_build –g, -u, -T or –w options

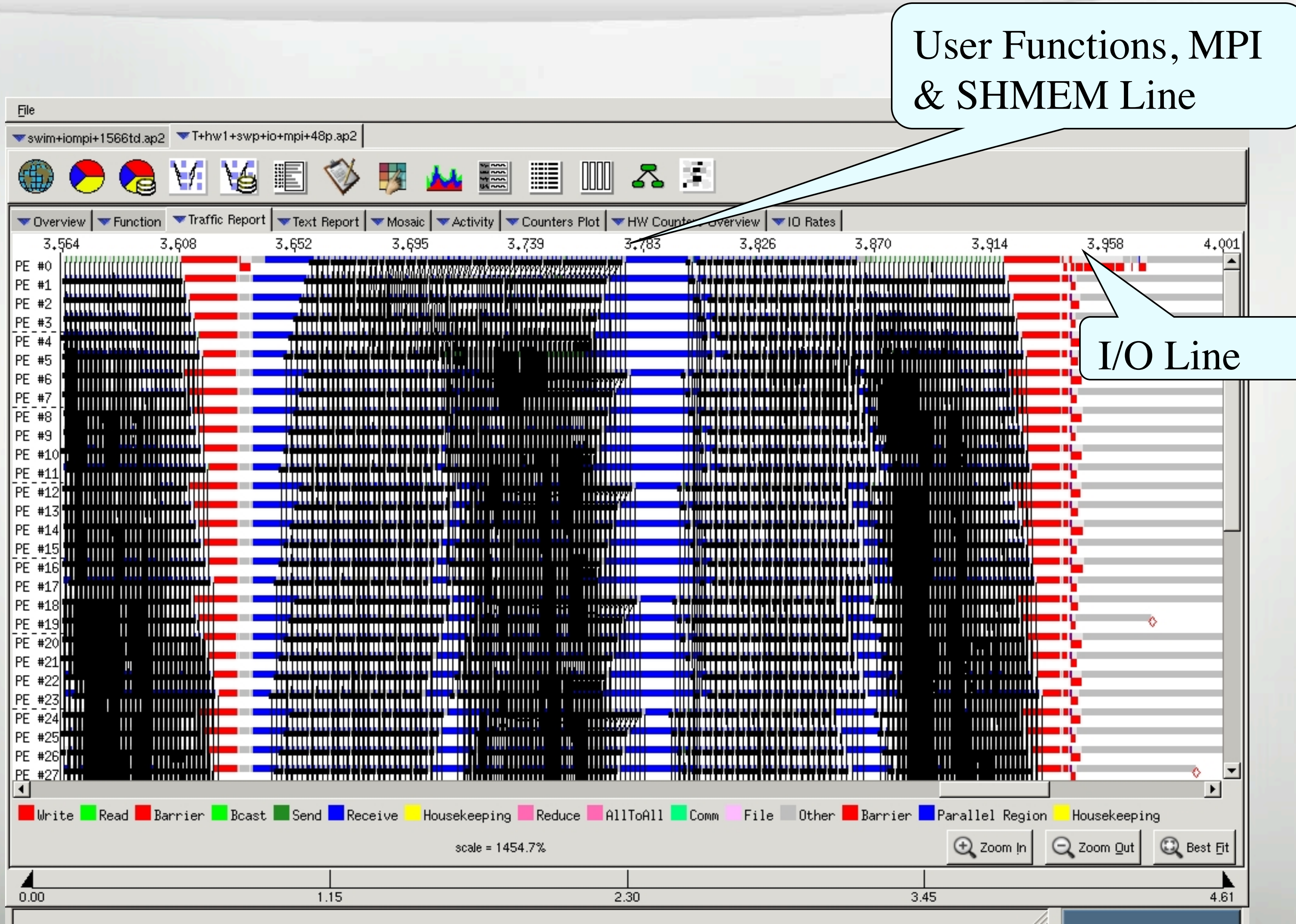
- Full trace (sequence of events) enabled by setting PAT_RT_SUMMARY=0

Cray Inc. Proprietary

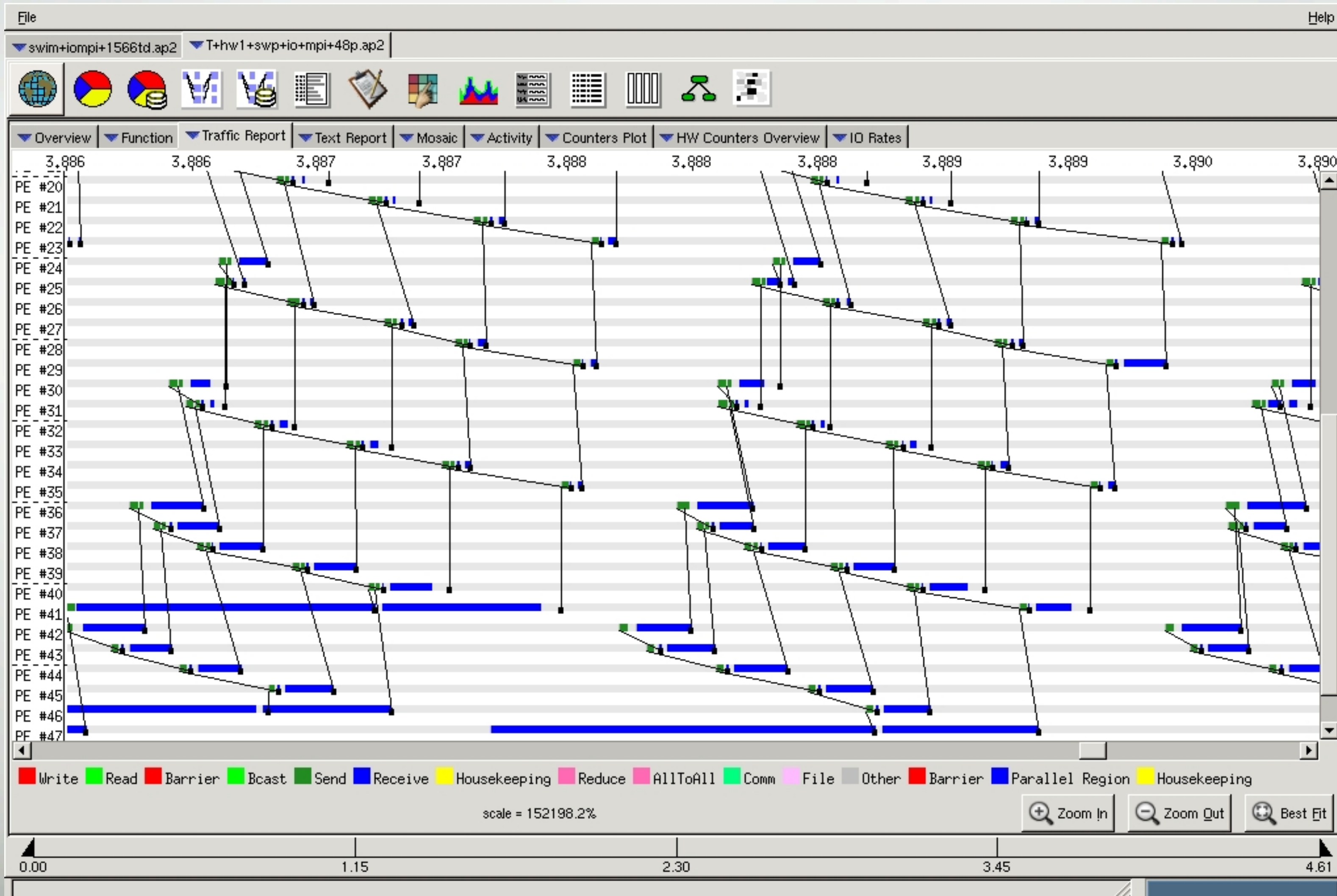
Time Line View (Sweep3D)



Time Line View (Zoom)



Time Line View (Fine Grain Zoom)



Controlling Trace File Size

Several environment variables are available to limit trace files to a reasonable size:

- PAT_RT_CALLSTACK
  - Limit the depth to trace the call stack
- PAT_RT_HWPC
  - Avoid collecting hardware counters (unset)
- PAT_RT_RECORD_PE
  - Collect trace for a subset of the PEs
- PAT_RT_TRACE_FUNCTION_ARGS
  - Limit the number of function arguments to be traced
- PAT_RT_TRACE_FUNCTION_LIMITS
  - Avoid tracing indicated functions
- PAT_RT_TRACE_FUNCTION_MAX
  - Limit the maximum number of traces generated for all functions for a single process

Where to Get Help

Accessing Software Versions

- Software package information
  - Use **avail**, **list** or **help** parameters to module command
  - ‘**module help perftools**’ shows release notes
- Version (same for pat_build, pat_report, pat_help)

% pat_build –V

CrayPat/X: Version 5.2.3 Revision 8155 09/13/11 08:47:57

- Cray Apprentice^2 version
  - Displayed in top menu bar when running GUI

Cray Inc. Proprietary

Release Notes

```
% module help perftools
----------- Module Specific Help for 'perftools/5.2.3' ---------
===================================================================
Perftools 5.2.3
==============
Release Date: September 15, 2011

Differences between CrayPat 5.2.2 release and 5.2.3 release
------------------------------------------------------------
 General
  * PAPI library supports counters in NVIDIA GPUs
  * PAPI library available as dynamically shared object
  * All installed PerfTools executable binary files are dynamically linked
. . .

Purpose
-------
. . .

Bugs Fixed
----------
. . .
Known Problem(s)
----------------
. . .
Product and OS Dependencies:
----------------------------
 . . .
Documentation:
  --------------
  See the following documents at http://docs.cray.com/
  Cray Performance Analysis Tools Release Overview and
      Installation Guide S-2474-52
  Using Cray Performance Analysis Tools S-2376-52
```

Online information

- User guide
  - http://docs.cray.com
- Man pages
- To see list of reports that can be generated

```
% pat_report -O -h
```

- Notes sections in text performance reports provide information and suggest further options

Online Information (2)

- Cray Apprentice^2 panel help
- pat_help – interactive help on the Cray Performance toolset
- FAQ available through pat_help

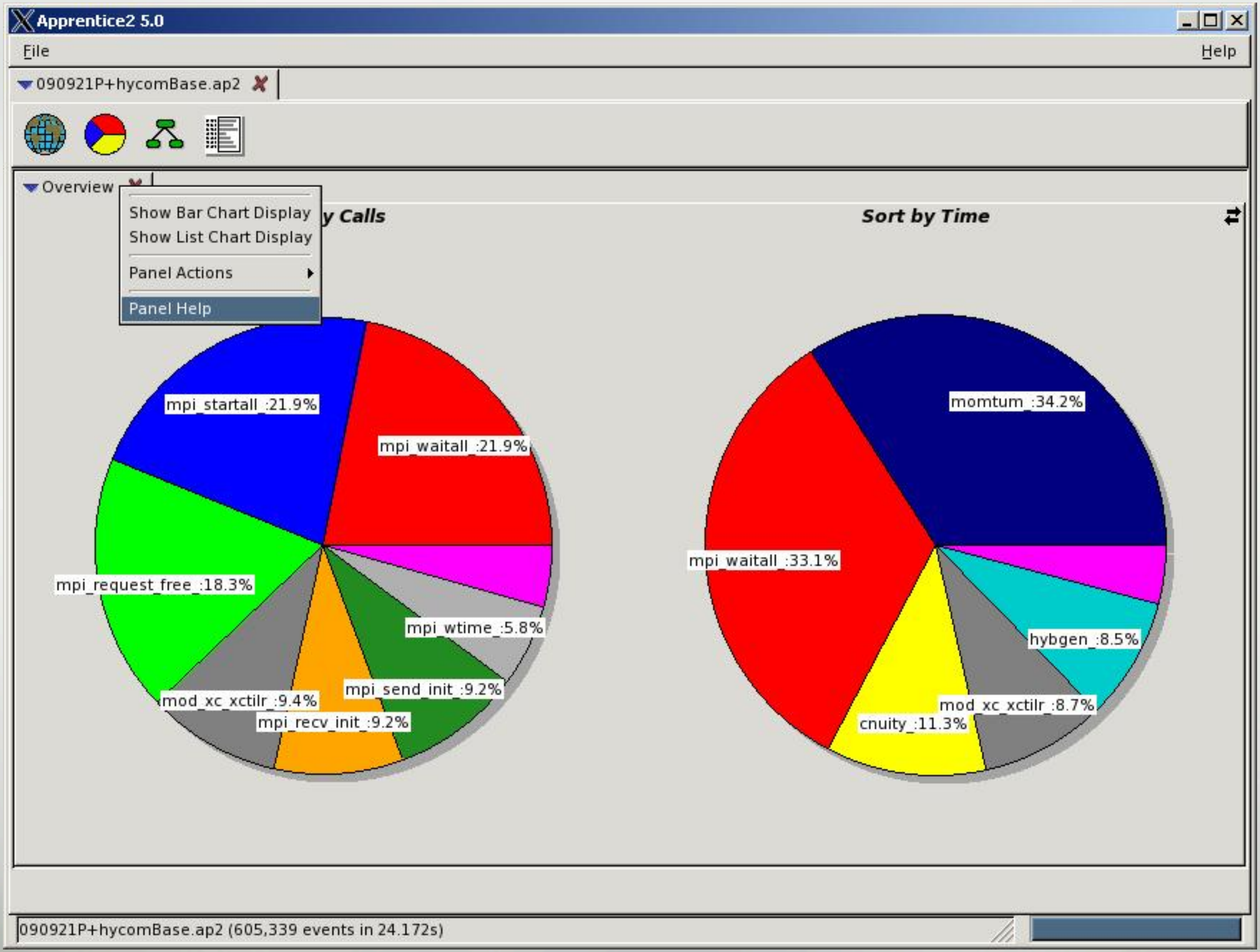
Cray Inc. Proprietary

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
- **hwpc**(5)
  - describes predefined hardware 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

Cray Inc. Proprietary

Cray Apprentice² Panel Help



Top of Default Report from APA Sampling

```
CrayPat/X:  Version 5.0 Revision 2631 (xf 2571)  05/29/09 14:54:00

Number of PEs (MPI ranks):      48
Number of Threads per PE:       1
Number of Cores per Processor:  4

Execution start time:   Fri May 29 15:31:49 2009
System type and speed:   x86_64  2200 MHz
Current path to data file:
  /lus/nid00008/homer/sweep3d/sweep3d.mpi+samp.rts.ap2  (RTS)

Notes:
    Sampling interval was 10000 microseconds (100.0/sec)
    BSD timer type was ITIMER_PROF

  Trace option suggestions have been generated into a separate file
  from the data in the next table.  You can examine the file, edit
  it if desired, and use it to reinstrument the program like this:

          pat_build -O sweep3d.mpi+samp.rts.apa
```

pat_help

- Interactive by default, or use trailing '.' to just print a topic:
- Troubleshooting FAQ available
- Has counter and counter group information

% pat_help counters amd_fam15h groups

pat_help Example

```
 The top level CrayPat/X help topics are listed below.
A good place to start is:

    overview

If a topic has subtopics, they are displayed under the heading
"Additional topics", as below.  To view a subtopic, you need
only enter as many initial letters as required to distinguish
it from other items in the list.  To see a table of contents
including subtopics of those subtopics, etc., enter:

    toc

To produce the full text corresponding to the table of contents,
specify "all", but preferably in a non-interactive invocation:

    pat_help all . > all_pat_help
    pat_help report all . > all_report_help

  Additional topics:

    API                        execute
    balance                    experiment
    build                      first_example
    counters                   overview
    demos                      report
    environment                run

pat_help (.=quit ,=back ^=up /=top ~=search)
=>
```

pat_help: FAQ

```
pat_help (.=quit ,=back ^=up /=top ~=search)
=> FAQ
  Additional topics that may follow "FAQ":

    Application Runtime                    Miscellaneous
    Availability and Module Environment    Processing Data with pat_report
    Building Applications                  Visualizing Data with Apprentice2
    Instrumenting with pat_build

pat_help FAQ (.=quit ,=back ^=up /=top ~=search)
=> I
  Additional topics that may follow ""Instrumenting with pat_build"":

        1. Can not access the file ...
        2. ERROR: Missing required ELF section 'link information' from the program 'FILE'.
        3. ERROR: Missing required ELF section 'string table' from the program '...'.
        4. FATAL: The link information was not found in the .note section of ...
        5. How can I find out the text size of functions?
        6. How can I list trace points from my instrumented binary?
        7. How can I lower the size of data files with pat_build?
        8. How can I NOT instrument some of my object file(s)?
        9. How do I get MPI rank order suggestions?
       10. How do I specify a directory containing object files?
       11. My error messaage is "xyz can not be traced because ... not writable"
       12. Problems with instrumented programs using both MPI and OpenMP?
       13. User sampling with compiler hooks present is not allowed
       14. WARNING: Entry point 'FUNCTION' can not be traced because it is a locally
           defined function
       15. WARNING: The function 'FUNCTION' can not be traced because a trace wrapper
           was not successfully created
       16. What is APA?
       17. Why am I getting an error with userTraceFunctions.c?
       18. Why does my binary take longer to run when using 'pat_build -u'?

pat_help FAQ "Instrumenting with pat_build"
(.=quit ,=back ^=up /=top ~=search) =>
```

A Peek at GPU Support

Cray Performance Tools vs CUDA Profiler

- Advantages of Cray performance tools:
  - Scaling (running big jobs with a large number of GPUs)
    - Results summarized and consolidated in one place. With the CUDA profiler, the user will have to sift through a log file per GPU to look at statistics.
  - Statistics for the whole application
    - Performance statistics mapped back to the user source by line number.
    - Performance statistics grouped by OpenMP accelerator directive
    - Single report can include statistics for both the host and the accelerator. The CUDA profiler will only give you the GPU statistics. You'll have to use something else to collect information about the X86 code.
  - Single tool for NVIDIA and AMD performance analysis
    - The user doesn't have to learn another tool when he or she runs an application on a system with AMD GPUs. The CUDA profiler won't work on AMD.

Types of Statistics

- Performance statistics
  - Includes accelerator time, host time, and amount of data copied to/from the accelerator.
- Kernel level statistics
  - Includes stats about grid size, block size, and occupancy. We are looking into ways to include stats on shared memory and local memory usage (part of memory footprint information).
- Accelerator hardware counters
  - Hardware counters on the accelerator itself. On Nvidia Fermi GPUs, there are about 50 available counters.

Cray Inc. Proprietary

Moving from X86 to Hybrid Multi-core Systems

- Running MPI only on a node will not work well
  - Too much memory used, even if on-node shared communication is available
  - As the number of MPI ranks increases, more off-node communication can result, creating a network injection issue
- Focus on where MPI starts leveling off
- Address by adding additional levels of parallelism, reducing MPI ranks per node
  - MPI -> MPI + OpenMP
  - MPI + OpenMP -> MPI + OpenMP GPU extensions

Cray Inc. Proprietary

Steps to Porting to Hybrid Multi-core Systems

- Maximize on-node communication if MPI point-to-point communication is dominant in the program
  - Auto-grid detection and placement suggestions
- Determine where to add additional levels of parallelism
  - Find top time consuming loops with enough work for GPU with loop statistics
- Do parallel analysis and restructuring on targeted high level loops
  - Reveal scoping assistance

Cray Inc. Proprietary

Steps to Porting to Hybrid Multi-core Systems (2)

- Add parallel directives and acceleration extensions
  - OpenMP extensions *(Reveal directive generation assistance)*
- Run on X86 + GPU and get performance feedback
  - Automatic profiling analysis
- Optimize for data locality and copies to the GPU
  - Cray performance tools accelerator statistics
- Optimize kernel on GPU
  - Cray performance tools accelerator statistics
- Optimize core performance on CPU
  - Automatic profiling analysis with CPU HW counter threshold feedback

Cray Inc. Proprietary

Example Performance Statistics

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

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

 100.0% | 18.113521 |        -- |      -- |    6.0 |Total
|-----------------------------------------------------------------
| 100.0% | 18.113443 |        -- |      -- |    5.0 |USER
||----------------------------------------------------------------
||  90.6% | 18.113000 | 0.000000 |   0.0% |    1.0 |acc_sample_.ACC_DATA_REGION@li.23
||   9.4% |  0.000443 | 0.000000 |   0.0% |    1.0 |acc_sample_.ACC_REGION@li.24
||================================================================
|   0.0% |  0.000078 | 0.000000 |   0.0% |    1.0 |ETC
||----------------------------------------------------------------
|   0.0% |  0.000078 | 0.000000 |   0.0% |    1.0 | exit
|=================================================================
```

Example Performance Statistics

```
Table 2:  Time and Bytes Transferred for Accelerator Regions

  Host   |    Host   |     Acc   | Acc Copy   | Acc Copy   | Calls   |Calltree
 Time%   |    Time   |    Time   |         In |        Out |         |
         |           |           | (MBytes)   | (MBytes)   |         |
 100.0%  |   18.113  |   18.112  |   209.808  |   209.808  |       4 |Total
|-----------------------------------------------------------------------------------------------
| 100.0% |   18.113  |   18.112  |   209.808  |   209.808  |       4 |acc_sample_
|        |           |           |            |            |         | acc_sample_.ACC_DATA_REGION@li.23
|||---------------------------------------------------------------------------------------------
3||  90.6% |   16.418  |       ---  |        ---  |        ---  |       1 |sync
3||   9.4% |    1.695  |    1.695  |   209.808  |   209.808  |       2 |transfer
3||   0.0% |    0.000  |   16.418  |     0.000  |     0.000  |       1 |acc_sample_.ACC_REGION@li.24
4||        |           |           |            |            |         | async_kernel
|===============================================================================================
```

A Peek at Reveal

New code restructuring and analysis assistant…

- Presents **annotated source code** with compiler optimization information ("loopmark on wheels")
- Offers **source code navigation** based on performance data collected through CrayPat
- Provides infrastructure for user to investigate high level looping structures for parallelization
- Highlights loops that could not be optimized
- Presents **feedback on** critical **dependencies** that prevent optimizations

Cray Inc. Proprietary

Source Code – Loopmark

CRAY THE SUPERCOMPUTER COMPANY

Compiler feedback

- 32.33% calc2.F
  - 32.33% CALC2
    - Loop@66
    - Loop@67
    - Loop@89
- 17.34% calc1.F
- 0.21% swim.F

Performance feedback

```
66        DO 200 I=1,M
67        DO 200 J=js,je
68        UNEW(I+1,J) = UOLD(I+1,J)+
69       1   TDTS8*(Z(I+1,J+1)+Z(I+1,J))*(CV(I+1,J+1)+CV
70       2      +CV(I+1,J))-TDTSDX*(H(I+1,J)-H(I,J))
71        if(j.gt.1)then
72        VNEW(I,J) = VOLD(I,J)-TDTS8*(Z(I+1,J)+Z(I,J))
73       1    *(CU(I+1,J)+CU(I,J)+CU(I,J-1)+CU(I+1,J-1))
74       2   -TDTSDY*(H(I,J)-H(I,J-1))
75        endif
76        if(j.eq.n)then
77        VNEW(I,J+1) = VOLD(I,J+1)-TDTS8*(Z(I+1,J+1)+Z(
78       1      *(CU(I+1,J+1)+CU(I,J+1)+CU(I,J)+CU(I+1,J))
79       2      -TDTSDY*(H(I,J+1)-H(I,J))
80        endif
81        PNEW(I,J) = POLD(I,J)-TDTSDX*(CU(I+1,J)-CU(I,J))
82       1      -TDTSDY*(CV(I,J+1)-CV(I,J))
83    200 CONTINUE
84
85  CME-----------------------------------------------
86  C
```

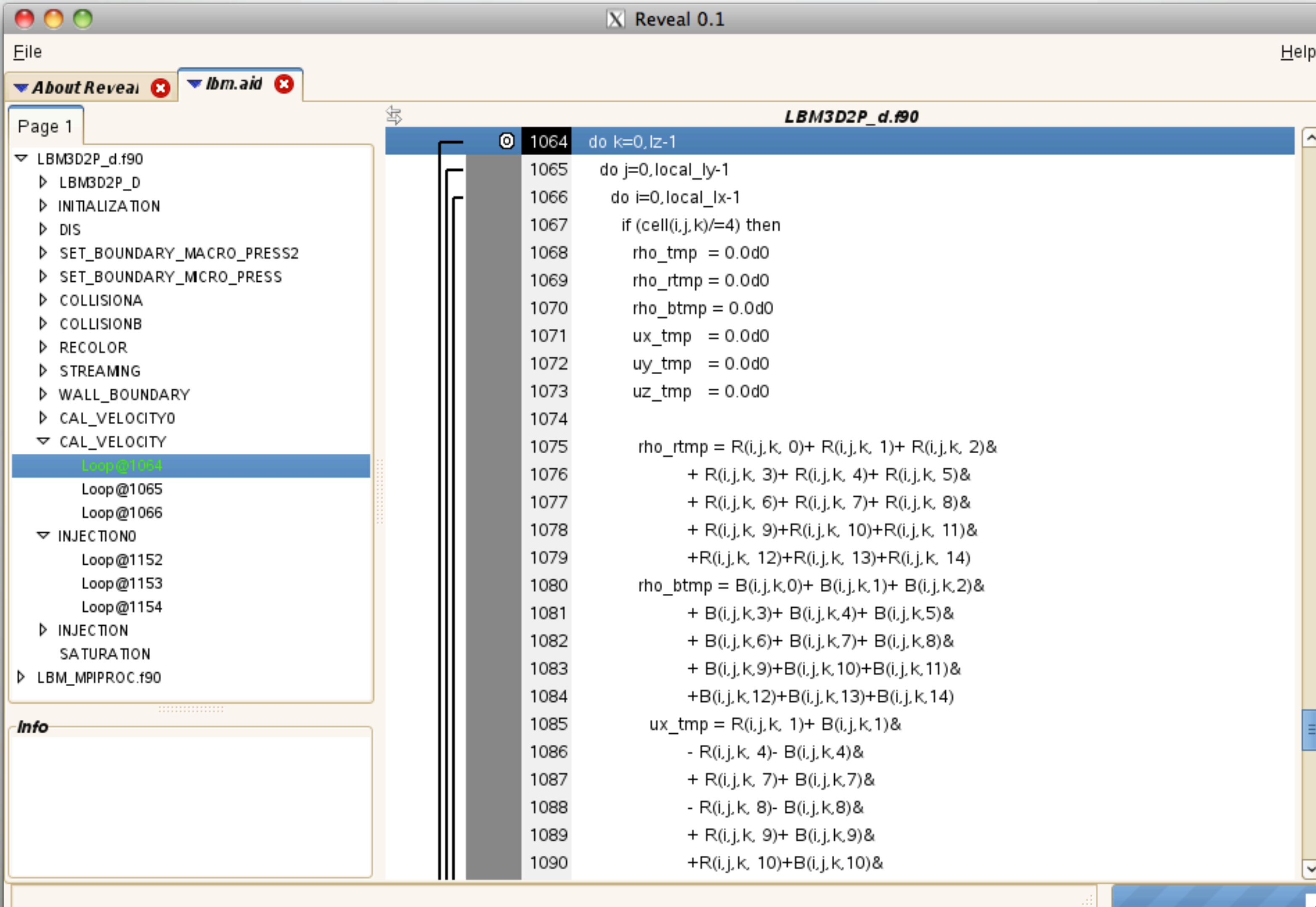
Compiler feedback

Info

Line 66:
Loop unrolled 2 times.
Loop interchanged with loop at line 67.

Cray Inc. Proprietary

Display Scoping Information for Selected Loop



Display Scoping Information for Selected Loop (2)

OpenMP Construct

LBM3D2P_d.f90: lines 1064 -> 1130

Name	Type	Scope	F	L	Info
b	Scalar	Shared			
cell	Scalar	Shared			
local_lx	Scalar	Shared			
local_ly	Scalar	Shared			
lz	Scalar	Shared			
r	Scalar	Shared			
rho	Scalar	Shared			
rho_btmp	Scalar	Private	N	N	
rho_rtmp	Scalar	Private	N	N	
rho_tmp	Scalar	Private	N	N	
ux_tmp	Scalar	Private	N	N	
uxyz	Scalar	Shared			
uz_tmp	Scalar	Private	N	N	

OK Revert Close

Cray Inc. Proprietary

Questions

??