Performance Measurement on the Cray XT System

Jason Beech-Brandt
Tom Edwards
Cray Centre of Excellence for HECToR

Topics

- Cray Performance Analysis Toolset Overview
- Recent Release Highlights
- Measuring Performance
- What’s Next

Cray Toolset Design Goals

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

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

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

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

- CrayPat
  - pat_build: automatic instrumentation (no source code changes needed)
  - run-time library for measurements (transparent to the user)
  - pat_report for performance analysis reports
  - pat_help: online help utility

- Cray Apprentice^2
  - Graphical performance analysis and visualization tool

The Cray Performance Analysis Framework (2)

- 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

Collecting Performance Data

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

Multiple Dimensions of Scalability

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

Performance Analysis with Cray Tools

- Important performance statistics:
  - Top time consuming routines
  - Load balance across computing resources
  - Communication overhead
  - Cache utilization
  - FLOPS
  - Vectorization (SSE instructions)
  - Ratio of computation versus communication

Application Instrumentation with pat_build

- No source code or makefile modification required
  - 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
  - Supports sample-based and event-based instrumentation

Automatic Profiling Analysis

- Analyze the performance data and direct the user to meaningful information

- Simplifies the procedure to instrument and collect performance data for novice users

- Based on a two phase mechanism
  1. Automatically detects the most time consuming functions in the application and feeds this information back to the tool for further (and focused) data collection
  2. Provides performance information on the most significant parts of the application

pat_report

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

Craypat / Cray Apprentice^2 Release Highlights

- Craypat / Cray Apprentice^2 5.1 released 17 June, 2010
  - xt-craypat and apprenctice2 repackaged as perftools module
  - Support for Gemini interconnect
  - Support for the Chapel programming language
  - New predefined trace groups – adios, armci, chapel, dmapp, pblas, petsc
  - License check support through FlexNet license server
  - Fully supports dynamically linked applications

Steps to Collecting Performance Data

- Access performance tools software

```
% module load perftools
```

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

```
% make clean
% make
```

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

```
% pat_build –O apa a.out
```

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

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

Steps to Collecting Performance Data (2)

- Generate report and .apa instrumentation file

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

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

APA File Example

```
#  You can edit this file, if desired, and use it
#  to reinstrument the program for tracing like this:
#
#         pat_build -O mhd3d.Oapa.x+4125-401sdt.apa
#
#  These suggested trace options are based on data from:
#
#       /home/crayadm/ldr/mhd3d/run/mhd3d.Oapa.x+4125-401sdt.ap2,
        /home/crayadm/ldr/mhd3d/run/mhd3d.Oapa.x+4125-401sdt.xf


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


#       HWPC group to collect by default.


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


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


#       Libraries to trace.


  -g mpi


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


#       User-defined functions to trace, sorted by % of samples.
#       Limited to top 200. A function is commented out if it has < 1%
#       of samples, or if a cumulative threshold of 90% has been reached,
#       or if it has size < 200 bytes.


 #  Note: -u should NOT be specified as an additional option.
```

```
# 43.37%  99659 bytes
        -T mlwxyz_

# 16.09%  17615 bytes
        -T half_

#  6.82%  6846 bytes
        -T artv_

#  1.29%  5352 bytes
        -T currenh_

#  1.03%  25294 bytes
        -T bndbo_

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


#  1.03%  31240 bytes
#         -T bndto_

. . .

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


  -o mhd3d.x+apa                 # New instrumented program.


  /work/crayadm/ldr/mhd3d/mhd3d.x  # Original program.
```

-g tracegroup

biolibs	Cray Bioinformatics library routines
blas	Basic Linear Algebra subprograms
caf	coarray Fortran
heap	dynamic heap
io	includes stdio and sysio groups
lapack	Linear Algebra Package
math	ANSI math
mpi	MPI
omp	OpenMP API
pthreads	POSIX threads (not supported on Catamount)
shmem	SHMEM
sysio	I/O system calls
system	system calls
upc	unified parallel c

Steps to Collecting Performance Data (2)

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

```
% pat_build –O <apafile>.apa
```

- Run application

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

- Generate text report and visualization file (.ap2)

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

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

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

Where to Run Instrumented Application

- MUST run on Lustre (/work/… , /lus/…, /scratch/…, etc.)

- Number of files used to store raw data

  - 1 file created for program with 1 – 256 processes

  - √*n* files created for program with 257 – *n* processes

  - Ability to customize with PAT_RT_EXPFILE_MAX

Outliers, or peak values, over time

- Full trace files show transient events but are too large
- Current run-time summarization misses transient events
- Plan to add ability to record:
  - Top N peak values (N small)
  - Approximate std dev over time
  - For time, memory traffic, etc.
  - During tracing and sampling

Overhead, scaling, advice

- Looking for ways to reduce both
  - Overhead of data collection during run-time
  - Time to process data and generate a report or graphical view
- New file format and post-processing architecture in 5.0
- 5.0 release has modest improvements in both areas
- 5.1 and succeeding releases will have
  - Much improved processing time
  - Better remote access to large data files
  - Analysis based on patterns and thresholds, generating advice

Performance Measurement on the Cray XT System

Questions / Comments

Thank You!

Performance Measurement of OpenMP Programs

Jason Beech-Brandt
Tom Edwards

Topics

- What do we want to measure?
- Data collection
- Data reporting

OpenMP Data Collection

- Measure overhead incurred entering and leaving
  - Parallel regions
  - Work-sharing constructs within parallel regions
- Trace entry points automatically inserted by Cray and PGI (7.2.0 or later) compilers
  - Provides per-thread information
- Can use sampling to get performance data without API (per process view… no per-thread counters)

OpenMP Data Collection (2)

- -g omp
  - Specifies tracing of user OpenMP API functions (like omp_test_lock)

- Need to add tracing support for barriers (both implicit and explicit)
  - Need support from compilers

- User API also available for OpenMP trace points when using other compilers

OpenMP Trace Point API

- C API (Same names for Fortran)

```
void PAT_omp_parallel_enter (void);
void PAT_omp_parallel_exit (void);
void PAT_omp_parallel_begin (void);
void PAT_omp_parallel_end (void);
void PAT_omp_loop_enter (void);
void PAT_omp_loop_exit (void);
void PAT_omp_sections_enter (void);
void PAT_omp_sections_exit (void);
void PAT_omp_section_begin (void);
void PAT_omp_section_end (void);
```

- Don’t support combined parallel work sharing constructs
  - Must split apart into parallel construct that contains work sharing constructs

- See pat_help for API function requirements

OpenMP Performance Data Reporting

- Default view (no options needed to pat_report)
  - Focus on where program is spending its time
  - Calculate load imbalance across all threads
    - Options also available to report per MPI rank, per thread
  - OpenMP overhead
  - Hardware counter statistics
    - Parallel regions
    - Work-sharing constructs within parallel regions
  - Assumes all requested resources should be used

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

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

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 aggrevate 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

Performance Measurement of OpenMP Programs

Questions / Comments

Thank You!

Documentation for the Cray Performance Toolset

Jason Beech-Brandt
Tom Edwards

Topics

- Software versions
- Online help
- Examples

Accessing Software Versions

- Software package information
  - Use avail, list or help parameters to module command
  - With 5.0 release and later, ‘module help perftools’ shows release notes
- craypat version (same for pat_build, pat_report, pat_help)

```
% pat_build –V
 CrayPat/X:  Version 5.0 Revision 2786  08/31/09 12:18:23
```

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

Online information

- User guide
  - http://docs.cray.com
  - Click on “Latest Docs” and choose “Performance Tools 5.0”

- 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

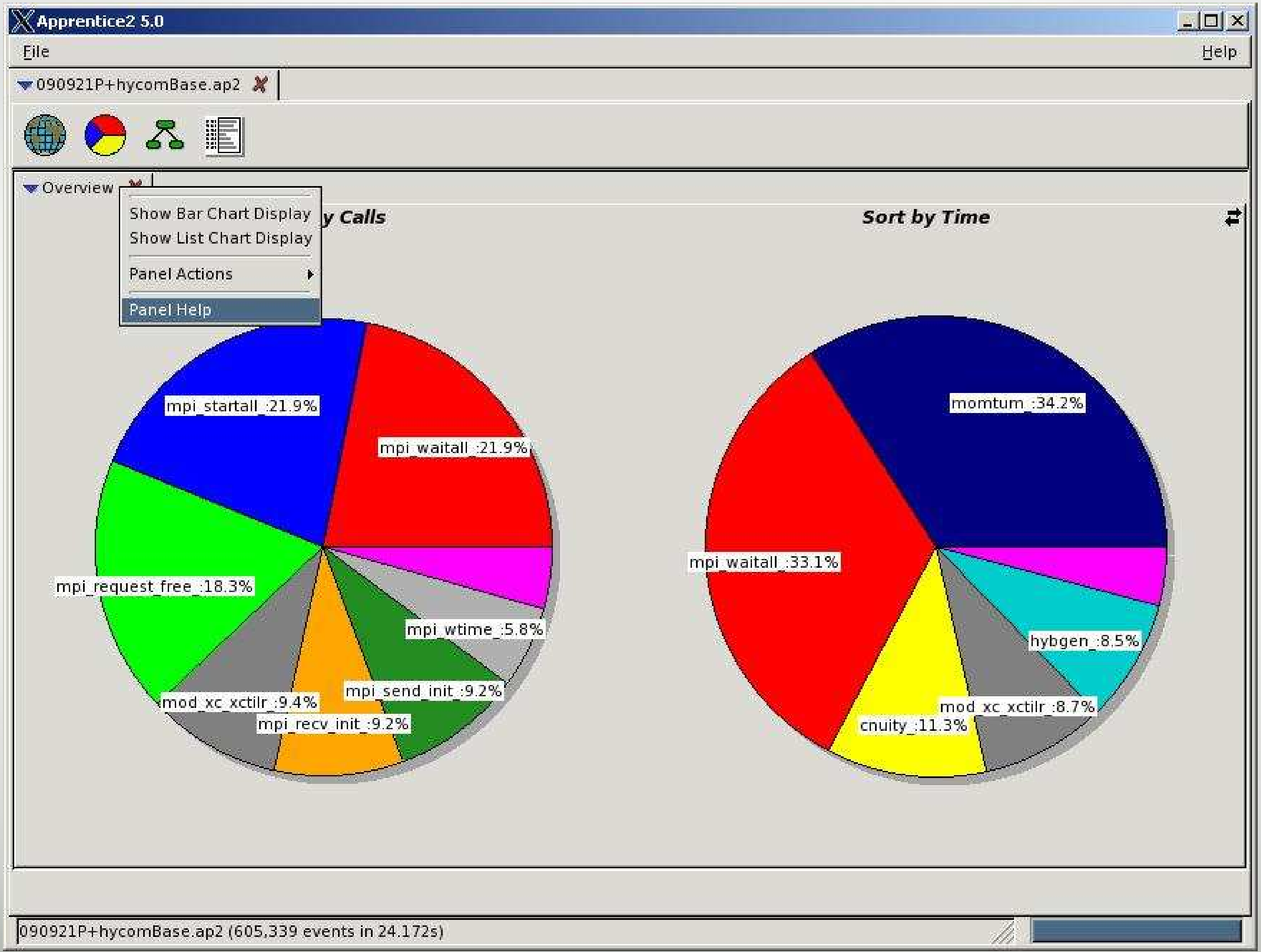
Online Information (2)

- Cray Apprentice2 panel help
- pat_help – interactive help on the Cray Performance toolset
- FAQ available through pat_help

Man pages

- **intro_craypat**(1)
  - Introduces the craypat performance tool
- **pat_build**
  - Instrument a program for performance analysis
- **pat_help**
  - Interactive online help utility
- **pat_report**
  - Generate performance report in both text and for use with GUI
- **hwpc**(3)
  - describes predefined hardware performance counter groups
- **papi_counters**(5)
  - Lists PAPI event counters
  - Use papi_avail or papi_native_avail utilities to get list of events when running on a specific architecture

Cray Apprentic^2 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_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
  ---
  callers                Profile by Function and Callers
  callers+hwpc           Profile by Function and Callers
  callers+src            Profile by Function and Callers, with Line Numbers
  callers+src+hwpc       Profile by Function and Callers, with Line Numbers
  calltree               Function Calltree View
  calltree+hwpc          Function Calltree View
  calltree+src           Calltree View with Callsite Line Numbers
  calltree+src+hwpc      Calltree View with Callsite Line Numbers
  ...
```

pat_help

- Interactive by default, or use trailing '.' to just print a topic:
- New FAQ craypat 5.0.0.
- Has counter and counter group information

```
% pat_help counters amd_fam10h 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)
=>
```

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
```

FAQ Example

```
% => 11. inclusive time of region recorded by
  CrayPat API

% (.=quit ,=back ^=up /=top ~=search) => 11

I cant find a way to make CrayPat report the
  inclusive time of a region recorded by the
  API. What can I do?

pat_help FAQ "Processing Data with pat_report"
 (.=quit ,=back ^=up /=top ~=search) =>
%
```

Documentation for the Cray Performance Toolset

Questions / Comments

Thank You!