CRAY
THE SUPERCOMPUTER COMPANY

Performance Measurement on the Cray XT System

Heidi Poxon
Performance Tools

CSC, Finland

September 21-24, 2009

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 5.0 released September 10, 2009
  - New internal data format
  - FAQ
  - Grid placement support
  - Better caller information (ETC group in pat_report)
  - Support larger numbers of processors
  - Client/server version of Cray Apprentice2
  - Panel help in Cray Apprentice2

Steps to Collecting Performance Data

- Access performance tools software

```
% module load xt-craypat apprentice2
```

- 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
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
stdio construct	all library functions that accept or return FILE*
sysio	I/O system calls
system	system calls

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

Better caller information (pat_report ETC group)

- Currently support MPI functions

- Still a problem for samples in some library functions
  - Miss immediate caller (leaf function does not create a frame)
  - Cannot find previous frame (frame pointer optimized out)

- Solution based on ideas, or even code, from dyninst
  - Use information from .eh_frame sections (offsets from sp)
  - Disassemble function

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 should 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!