Introduction to the Cray Performance Analysis Tools

Outline

- Introduction
- Using pat_build
- Using the CrayPat Run Time Environment
- Using pat_report
- Using Cray Apprentice2
- Resources for Users

Outline: Introduction

- The Cray Performance Analysis Tools
- Loading CrayPat and Apprentice2
- Analyzing Program Performance
- Cray Performance Analysis Infrastructure
- CrayPat and Apprentice2 Facts
- Online Help
- Using pat_help online help system
- CrayPat Overview
- Apprentice2 Overview
- Reference Files
- Performance API (PAPI)
- CrayPat General Workflow
- Upgrading from Earlier Versions

The Cray Performance Analysis Tools

- *The Cray Performance Analysis Tools* are a suite of utilities that enable you to capture and analyze performance data generated during the execution of your program on a Cray XT system. The information collected and analysis produced by use of these tools can help you to find answers to two fundamental programming questions:
  - *How fast is my program running?*
  - *How can I make it run faster?*
- The Cray Performance Analysis Tools suite consists of two components:
  - **CrayPat**: the program instrumentation, data capture, and basic text reporting tool
  - **Cray Apprentice2**: the graphical analysis and data visualization tool
- NCCS provides an access to both of these components (jaguarpf, 8-Mar-10)
  - CrayPat: xt-craypat/5.0.1
  - Apprentice^2: apprentice2/5.0.1
- CrayPat supports many languages + extensions:
  - Fortran, C, C++, UPC, MPI, CoArray Fortran, OpenMP, SHMEM

Loading CrayPat and Apprentice2

- To use CrayPat, first load your programming environment of choice, and then load the CrayPat module:

```
$ module load xt-craypat
```

- CrayPat module must be loaded before you compile the program to be instrumented.
- When instrumenting a program, CrayPat requires that the object (.o) files created during compilation be present, as well as the library (.a) files, if any.
- To begin using Cray Apprentice2, load the apprentice2 module:

```
$ module load apprentice2
```

- To launch the Cray Apprentice2 application, enter this command:

```
$ app2 &
```

- You can specify an .ap2 data file to be opened when you launch Cray Apprentice2:

```
$ app2 my_datafile.ap2 &
```

Analyzing Program Performance

The performance analysis process consists of three basic steps.

1. Instrument your program, to specify what kind of data you want to collect under what conditions.
2. Execute your instrumented program, to generate and capture the desired data.
3. Analyze the resulting data.

Cray Performance Analysis Infrastructure

- ***CrayPat***
  - `pat_build`: the utility for application instrumentation
    - No source code modification required
  - `run-time` library for measurements
    - transparent to the user
  - `pat_report`: the first-level analysis tool used to produce
    - Performance text reports
    - Performance visualization file
  - `pat_help`: Interactive performance tool help utility
- ***Cray Apprentice2***
  - Graphical performance analysis and visualization tool
  - Can be used off-line on Linux system
- *All CrayPat components, including the man pages and help system, are available only when the CrayPat module is loaded.*

CrayPat and Apprentice^2 Facts

- CrayPat
  - Cray's toolkit for instrumenting executables and producing data from runs
  - Uses *static binary instrumentation*
  - Supports tracing, profiling, and sampling
  - Outputs data in binary format which can be converted to
    - XML format (for Apprentice^2)
    - Text format (report that contains statistical information)
- Apprentice^2
  - Optional visualization tool for CrayPat data files
  - Can read in .xml or .xml.gz files (gzipped XML reports converted from binary output of CrayPat)
  - Several visualizations available

Online Help

- The CrayPat man pages, online help, and FAQ are available only when the xt-craypat module is loaded.
- The CrayPat commands, options, and environment variables are documented in the following man pages:

```
$ man intro_craypat
$ man pat_build
$ man pat_report
$ man hwpc
$ man app2
```

Command	Description
`$ man intro_craypat`	(all runtime environment variables are here)
`$ man pat_build`	(application instrumentation)
`$ man pat_report`	(report generation)
`$ man hwpc`	(all hardware counter groups are here)
`$ man app2`	(performance visualization)

- In addition, CrayPat also includes an extensive online help system, which features many examples and the answers to many frequently asked questions. To access the help system, enter this command:

```
$ pat_help [topic [subtopic...]]
```

Using `pat_help` online help system

```
$ pat_help

    After reading this page, a new user should start with the topic:

        Overview

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

    APA                 environment
    API                 execute
    FAQ                 experiment
    First_example       loop_stats
    OpenMP              mpi_rank_order
    Overview            mpi_sync
    balance             regions
    build               report
    counters            spreadsheets
    demos

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

CrayPat Overview

- Command-line based performance optimization tool
- In CrayPat, you perform *experiments* on instrumented executables.
  - Two types of experiments are available
    - Tracing: Record timestamps and arguments for all instrumented functions
    - Sampling: Samples hardware counters or callstack at fixed intervals
  - Type of experiment guided by setting environment variables
    - Can only perform tracing experiments on executables instrumented for tracing
    - However, if a program is instrumented for tracing and then one uses PAT_RT_EXPERIMENT to specify a sampling experiment, trace-enhanced sampling is performed.

Apprentice² Overview

- Visualization tool for XML files produced by CrayPat
- Supports visualization of
  - Callstack sampling experiments
  - MPI trace experiments
- Available visualizations
  - Overview piecharts that contain a breakdown of data by time and calls
  - Traffic Report shows internal PE-to-PE traffic over time.
  - Text report (similar to what is available from CrayPat)
  - Mosaic shows communication volume between processing elements
  - Activity (shows % time spent in different MPI functions as a function of time)
  - Profile (show call tree with observed times)
- Several visualizations also have “calipers” at bottom of screen to restrict view to certain time periods

Reference Files

- When the CrayPat module is loaded, the environment variable `CRAYPAT_ROOT` is defined.
- Advanced users will find the files in `$CRAYPAT_ROOT/lib` and `$CRAYPAT_ROOT/include` useful.
- The /lib directory contains the predefined trace group definitions and build directives, while the /include directory contains the files used with the CrayPat API.

Performance API (PAPI)

- CrayPat uses PAPI, the Performance API.
- This interface is normally transparent to the user. However, if you want more information about PAPI, see the following man pages:

  ```
  $man intro_papi
  $man papi_counters
  ```

  as well as the *PAPI Programmer's Reference* and *PAPI User's Guide*.
- Additional information is available through the PAPI website at http://icl.cs.utk.edu/papi/

CrayPat General Workflow

General workflow:

1. Load CrayPat and Apprentice2 modules
2. Compile application and run as normal
3. Instrument using `pat_build`
4. Run instrumented executable as normal; binary .xf log file will be produced
5. View report using `pat_report`

Upgrading from Earlier Versions

- If you are upgrading from an earlier version of CrayPat or Apprentice2, be advised that file compatibility is not maintained between versions. Programs instrumented using earlier versions of CrayPat must be recompiled, relinked, and reinstrumented using CrayPat 5.0. Likewise, .xf and .ap2 data files created using earlier versions of CrayPat cannot be read using the release 5.0 versions of pat_report or Cray Apprentice2, nor can data files created using release 5.0 be read using earlier versions of pat_report or Cray Apprentice2.
- To revert to the earlier version, use the module swap command. For example, assuming that the current default version is 5.0, to revert from CrayPat 5.0 to CrayPat 4.4 so that you can read an old .ap2 file, enter this command:
  - `$ module swap xt-craypat xt-craypat/4.4.0`
  - `$ module swap apprentice2 apprentice2/4.4.0`
- To return to the current default version, reverse the command arguments:
  - `$ module swap xt-craypat/4.4.0 xt-craypat`
  - `$ module swap apprentice2/4.4.0 apprentice2`

Outline: Using `pat_build`

- Application Instrumentation with pat_build
- Basic Profiling
- Using Predefined Trace Groups
- Other useful flags
- Performance Data Collection
- Performance Analysis with Cray Tools
- Running the Instrumented Application
- Automatic Program Analysis (APA)
- CrayPat API - for fine grain instrumentation

Application Instrumentation with `pat_build`

- No source code or makefile modification required
  - Automatic instrumentation at group (function) level
    - Groups: mpi, io, heap, math, …
- 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
- pat_build [-d dirfile] [-D directive] [-f] [-g tracegroup] [-n] [-O ofile] [-o instr_program] [-t tracefile] [-T tracefunc] [-u] [-V] [-v] [-w] [-z] program [instr_program]

Go to Menu

Basic Profiling

- The easiest way to use the pat_build command is by accepting the defaults.

```
$ pat_build myprogram
```

- This generates a copy of your original executable that is instrumented for the default experiment, samp_pc_time, an experiment that samples program counters at regular intervals and produces a basic profile of the program's behavior during execution.
- A variety of other predefined experiments are available. However, in order to use any of these other experiments, you must first instrument your program for tracing.

Using Predefined Trace Groups

- The easiest way to instrument your program for tracing is by using the -g option to specify a predefined trace group:

```
$ pat_build -g tracegroup myprogram
```

- Some of the valid trace group names are:

blas	Basic Linear Algebra subprograms
heap	dynamic heap
io	includes stdio and sysio groups
math	ANSI math
mpi	MPI
omp	OpenMP API
shmem	SHMEM
stdio	all library functions that accept or return the FILE* construct

- The files that define the predefined trace groups are kept in $CRAYPAT_ROOT/lib. To see exactly which functions are being traced in any given group, examine the Trace files. These files can also be used as templates for creating user-defined tracing files.

Other useful flags

- To change the default experiment from sampling to tracing, activate any API calls added to your program, and enable tracing for user-defined functions, use the –w option.

```
$ pat_build -w myprogram
```

- To instrument a specific function by name, use the -T option.

```
$ pat_build -T tracefunc myprogram
```

This option applies to all the entry points contained within the predefined function groups that are used with the -g option. If the -w option is specified, user-defined entry points are traced as well.

- To trace a user-defined list of functions, use the -t option.

```
$ pat_build -t tracefile myprogram
```

The *tracefile* is a plain ASCII text file listing the functions to be traced. For an example of a *tracefile*, see any of the predefined Trace files in $CRAYPAT_ROOT/lib.

Performance Data Collection

- Two dimensions
  - When performance collection is triggered
    - Externally (asynchronous)
      - Sampling
        - » Timer interrupt
        - » Hardware counters overflow
    - Internally (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

Go to Menu

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

Running the Instrumented Application

- MUST run on Lustre
  - `cd /tmp/work/$USER`
- Can use runtime environment variables
  - Optional timeline view of program available
    - export PAT_RT_SUMMARY=0
    - View trace file with Cray Apprentice2
  - Number of files used to store raw data:
    - 1 file created for program with 1 – 256 processes
    - $\sqrt{n}$ files created for program with 257 n processes
    - Ability to customize with PAT_RT_EXPFILE_MAX
  - Request hardware performance counter information:
    - export PAT_RT_HWPC=<HWPC Group>
    - Can specify events or predefined groups

Automatic Program Analysis (APA). 1-5 STEPS…

1. Load CrayPat & Cray Apprentice2 module files
```
$ module load xt-craypat apprentice2
```
2. Build application
```
$ make clean
$ make
```
3. Instrument application for automatic program analysis
```
$ pat_build –O apa a.out
```
   - You should get an instrumented program a.out+pat
4. Run application to get top time consuming routines
   - Remember to modify <script> to run *a.out+pat*
   - Remember to run on Lustre
```
$ aprun … a.out+pat (or qsub <pat script>)
```
   - You should get a performance file ("<sdatafile>.xf") or multiple files in a directory <sdatadir>
5. Generate .apa file
```
$ pat_report –o my_sampling_report [<sdatafile>.xf | <sdatadir>]
```
   - creates a report file, .ap2 file and an automatic program analysis file <apafile>.apa

Automatic Program Analysis (APA). 6-10 STEPS

6. Look at <apafile>.apa file
   - Verify if additional instrumentation is wanted
7. Instrument application for further analysis (a.out+apa)
```
$ pat_build –O <apafile>.apa
```
   - You should get an instrumented program a.out+apa
8. Run application
   - Remember to modify <script> to run *a.out+apa*
```
$ aprun … a.out+apa (or qsub <apa script>)
```
   - You should get a performance file ("<datafile>.xf") or multiple files in a directory <datadir>
9. Create text report
```
$ pat_report –o my_text_report.txt
[<datafile>.xf | <datadir>]
```
   - Will generate a compressed performance file (<datafile>.ap2)
10. View results in text (my_text_report.txt) and/or with Cray Apprentice2
```
$ app2 <datafile>.ap2
```

CrayPat API - for fine grain instrumentation

There may be times when you want to focus on a certain region within your code, either to reduce sampling overhead, reduce data file size, or because only a particular region or function is of interest. In these cases, use the CrayPat API to insert calls into your program source, to turn data capture on and off at key points during program execution.

FORTRAN	C
include “pat_apif.h”	include <pat_api.h>
…	…
call PAT_region_begin(id, “label”, ierr)	ierr = PAT_region_begin(id, “label”);
< code segment >	< code segment >
call PAT_region_end(id, ierr)	ierr = PAT_region_end(id);

Outline: Using the CrayPat Run Time Environment

- Basic Information
- Controlling Run Time Summarization
- Controlling Data File Size
- Selecting a Predefined Experiment
- Trace-enhanced Sampling
- Measuring MPI Load Imbalance
- Monitoring Hardware Counters
- Monitoring Hardware Counters using PAPI

Basic Information

- The CrayPat run time environment variables communicate directly with an executing instrumented program and affect how data is collected and saved.
- Detailed descriptions of all run time environment variables are provided in the intro_craypat man page. Additional information can be found in the online help system under pat_help environment.
- All CrayPat run time environment variable names begin with PAT_RT_.
- Some require discrete values, while others are toggles. In the case of all toggles, a value of 1 is on (enabled) and 0 is off (disabled).

Controlling Run Time Summarization

Variable: PAT_RT_SUMMARY

Run time summarization is enabled by default. When it is enabled, data is captured in detail, but automatically aggregated and summarized before being saved. This greatly reduces the size of the resulting experiment data files but at the cost of fine-grain detail. Specifically, when running tracing experiments, the formal parameter values, function return values, and call stack information are not saved.

Controlling Data File Size

Depending on the nature of your experiment and the duration of the program run, the data files generated by CrayPat can be quite large. To reduce the files to manageable sizes, considering adjusting the following run time environment variables.

For sampling experiments, try these:	For tracing experiments, try these:
PAT_RT_CALLSTACK PAT_RT_EXPFILE_PES PAT_RT_HWPC PAT_RT_HWPC_OVERFLOW PAT_RT_INTERVAL PAT_RT_SUMMARY PAT_RT_SIZE	PAT_RT_CALLSTACK PAT_RT_EXPFILE_PES PAT_RT_HWPC PAT_RT_RECORD_THREAD PAT_RT_SUMMARY PAT_RT_TRACE_FUNCTION_ARGS PAT_RT_TRACE_FUCNTION_LIMITS PAT_RT_TRACE_FUNCTION_MAX PAT_RT_TRACE_THRESHOLD_PCT PAT_RT_TRACE_THRESHOLD_TIME

Selecting a Predefined Experiment

Variable: PAT_RT_EXPERIMENT

- By default, CrayPat instruments programs for a program-counter *sampling* experiment, samp_pc_time, which samples program counters by time and produces a generalized profile of program behavior during execution.
- However, if any function entry points are instrumented for tracing by using the pat_build -g, -u, -t, -T, -O, or -w options, then the program is instrumented for a *tracing* experiment, which traces calls to the specified function entry point(s).
- After your program is instrumented using pat_build, use the PAT_RT_EXPERIMENT environment variable to further specify the type of experiment to be performed.
- **Note:** Samples generated from sampling by time experiments apply to the process as a whole, and not to individual threads. Samples generated from sampling by overflow experiments apply to individual threads.

Selecting a Predefined Experiment (continued)

- The valid experiment types are:

```
samp_pc_time
samp_pc_ovfl
samp_cs_time
samp_cs_ovfl
samp_ru_time
samp_ru_ovfl
samp_heap_time
samp_heap_ovfl
trace
```

- **Note:** If a program is instrumented for tracing and then you use PAT_RT_EXPERIMENT to specify a sampling experiment, trace-enhanced sampling is performed.

Trace-enhanced Sampling

Variable: PAT_RT_SAMPLING_MODE

Trace-enhanced sampling is affected by the PAT_RT_SAMPLING_MODE environment variable. This variable can have one of the following values:

0	Ignore trace-enhanced sampling. Perform a normal tracing experiment. (Default)
1	Enable raw sampling. Any traced entry points present in the instrumented program are ignored.
2	Enable focused sampling. Only traced entry points and the functions they call are sampled.
3	Enable bubble sampling. Traced entry points and any functions they call return a sample program counter address mapped to the trace entry point.

Trace-enhanced sampling is also affected by the PAT_RT_SAMPLING_SIGNAL environment variable. This variable can be used to specify the signal that is issued when an interval timer expires or a hardware counter overflows. The default value is 29 (SIGPROF).

Measuring MPI Load Imbalance

Variable: PAT_RT_MPI_SYNC

- In MPI programs, time spent waiting at a barrier before entering a collective can be a significant indication of load imbalance. The PAT_RT_MPI_SYNC environment variable, if set, causes the trace wrapper for each collective subroutine to measure the time spent waiting at the barrier call before entering the collective. This time is reported by pat_report in the function group MPI_SYNC, which is separate from the MPI function group, which shows the time actually spent in the collective.
- This environment variable affects tracing experiments only and is set on by default.

Monitoring Hardware Counters

Environment variable: PAT_RT_HWPC

- Use this environment variable to specify hardware counters to be monitored while performing tracing experiments. The easiest way to use this feature is by specifying the ID number of one of the predefined hardware counter groups; these groups and their meanings vary depending on your system's processor architecture and are defined in the hwpc man page.
- The behavior of the PAT_RT_HWPC environment variable is also affected by the PAT_RT_HWPC_DOMAIN, PAT_RT_HWPC_FILE, PAT_RT_HWPC_FILE_GROUP, and PAT_RT_HWPC_OVERFLOW environment variables. All of these are described in detail in the intro_craypat man page.

Monitoring Hardware Counters using PAPI

- More adventurous users may want to load the PAPI module and then use this environment variable to specify one or more hardware counters by PAPI name. To load the PAPI module, enter this command:

```
$ module load xt-papi
```

- Then use the papi_avail and papi_native_avail commands to explore the list of counters available on your system. For more information about using PAPI, see the intro_papi, papi_avail and papi_native_avail man pages.
- Note that the papi_avail and papi_native_avail commands should be executed on a compute node. If you run these commands on a login node you will get an accurate but not relevant information.
- Use the command similar to the one below in your PBS script, or run it interactively

```
aprun -n 1 /opt/xt-tools/papi/3.6.2.2/bin/papi_avail
```

Outline: Using `pat_report`

- Performance analysis using pat_report
- Using Data Files
- Producing Reports
- Using Predefined Reports
- User-defined Reports
- Exporting Data

Performance analysis using `pat_report`

- Performs data conversion
  - Combines information from binary with raw performance data
- Generates text report of performance results
- Formats data for input into Cray Apprentice2

- pat_report [-V] [-i dir|instrprog] [-o output_file] [-O keyword] [-C 'table caption'] [-d d-opts] [-b b-opts] [-s key=value] [-H] [-P] [-T] [-z] data_directory | data_file.xf

Using Data Files

The data files generated by CrayPat vary depending on the type of program being analyzed, the type of experiment for which the program was instrumented, and the run time environment variables in effect at the time the program was executed. In general, the successful execution of an instrumented program produces one or more .xf files, which contain the data captured during program execution. Unless specified otherwise using run time environment variables, these file names have the format *a.out*+pat+*PID-NIDe[m]*.xf:

a.out	The name of the instrumented executable.
PID	The process ID assigned to the instrumented executable at run time.
NID	The physical node ID upon which the rank zero process was executed.
e	The type of experiment performed, either s for sampling or t for tracing.
m	An optional code indicating other special characteristics of the program that produced the data file. These can be:
d	The data was generated by a distributed memory process such as MPI, SHMEM, UPC, or CAF.
f	The data was generated by a forked process.
o	The data was generated by OpenMP.
t	The data was generated by POSIX threads.

Go to Menu

Producing Reports

- To generate a report, use the pat_report command to process your .xf file or directory containing .xf files.

```
$ pat_report a.out+pat+PIDe[m]-n.xf
```

- **Note:** Running pat_report automatically generates an .ap2 file, which is both a self-contained archive that can be reopened later using the pat_report command and the exported-data file format used by Cray Apprentice2. Also, if the executable was instrumented with the pat_build -O apa option, running pat_report on the .xf file(s) produces an .apa file, which is the file used by Automatic Program Analysis.

Using Predefined Reports

- The easiest way to use pat_report is by using the -O to specify one of the predefined reports. For example, enter this command to see a top-down view of the calltree.

```
$ pat_report -O calltree datafile.xf
```

- **Note:** By default, all reports show either no individual PE values or only the PEs having the maximum, median, and minimum values. The suffix _all can be appended to any of the above options to show the data for all PEs. For example, the option load_balance_all shows the load balance statistics for all PEs involved in program execution. Use this option with caution, as it can yield very large reports.

User-defined Reports

In addition to the -O predefined report options, the pat_report command supports a wide variety of user-configurable options that enable you to create and generate customized reports. These options are described in detail in the pat_report man page and examples are provided in the pat_help online help system. If you want to create customized reports, pay particular attention to the -s, -d, and -b options:

-s	These options define the presentation and appearance of the report, ranging from layout and labels, to formatting details, to setting thresholds that determine whether some data is considered significant enough to be worth displaying.
-d	These options determine which data appears on the report. The range of data items that can be included also depends on how the program was instrumented, and can include counters, traces, time calculations, mflop counts, heap, I/O, and MPI data. As well, these options enable you to determine how the values that are displayed are calculated.
-b	These options determine how data is aggregated and labeled in the report summary.

Exporting Data

- When you use the pat_report command to view an .xf file or a directory containing .xf files, pat_report automatically generates an .ap2 file, which is a self-contained archive file that can be reopened later using either pat_report or Cray Apprentice2. No further work is required in order to export data for use in Cray Apprentice2.
- The pat_report -f option also enables you to export data to ASCII text or XML-format files. When used in this manner, pat_report functions as a data export tool. The entire data file is converted to the target format, and the pat_report filtering and formatting options are ignored.

Note: *Data file compatibility is not maintained between versions. If you are upgrading from an earlier version, .ap2 files created with earlier versions cannot be used with release 5.0, nor can files created with release 5.0 be viewed with earlier versions.*

Outline: Using Cray Apprentice2

- Launching the Program
- What is the ".ap2" File?
- Basic Navigation
- Viewing Reports

Launching the Program

- To launch the Cray Apprentice2 application, enter this command:

```
$ app2 &
```

- You can specify an .ap2 data file to be opened when you launch Cray Apprentice2:

```
$ app2 my_datafile.ap2 &
```

- If you did not specify an .ap2 data file or directory on the command line, the **File Selection Window** is displayed after Apprentice2 is launched.
- The app2 command supports two options: --limit and --limit_per_pe. These options enable you to restrict the amount of data being read in from the data file. Both options recognize the K, M, and G abbreviations for kilo, mega, and giga; for example, to open an .ap2 data file and limit Cray Apprentice2 to reading in the first 3 million data items, enter this command:

```
$ app2 --limit 3M data_file.ap2 & &
```

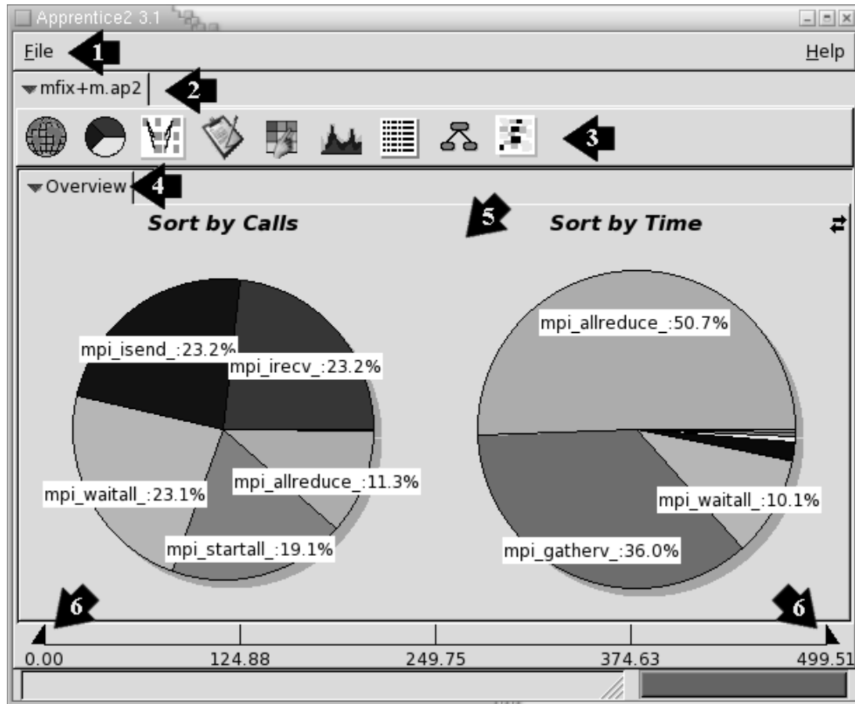
Note: *Cray Apprentice2 requires that your workstation be configured to host X Window System sessions.*

Go to Menu

What is the “.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
- With CrayPat 4.2 and newer, the “ap2” file is generated by default when executing pat_report

Basic Navigation



1	The **File menu** enables you to open data files or directories, capture the current screen display to a .jpg file, or exit from Cray Apprentice2.
2	The **Data tab** shows the name of the data file currently displayed. You can have multiple data files open simultaneously for side-by-side comparisons of data from different program runs. Click a data tab to bring a data set to the foreground. Right-click the tab for additional options.
3	The **Report toolbar** show the reports that can be displayed for the data currently selected. Hover the cursor over an individual report icon to display the report name. To view a report, click the icon.

4	The **Report tabs** show the reports that have been displayed thus far for the data currently selected. Click a tab to bring a report to the foreground. Right-click a tab for additional report-specific options.
5	The main display varies depending on the report selected and can be resized to suit your needs. However, most reports feature **pop-up tips** that appear when you allow the cursor to hover over an item, and **active data elements** that display additional information in response to left or right clicks.
6	On many reports, the total duration of the experiment is shown as a graduated bar at the bottom of the report window. Move the **caliper points** left or right to restrict or expand the span of time represented by the report. This is a global setting for each data file: moving the caliper points in one report affects all other reports based on the same data, unless those other reports have been detached or frozen.

Go to Menu

Viewing Reports

- The reports Cray Apprentice2 produces vary depending on the types of performance analysis experiments conducted and the data captured during program execution. The report icons indicate which reports are available for the data file currently selected. Not all reports are available for all data.

- The following reports are supported:
  - Overview Report
  - Environment Reports
  - Traffic Report
  - Mosaic Report
  - Activity Report
  - Function Report
  - Call Graph
  - I/O Reports
    - I/O Overview Report
    - I/O Rates
  - Hardware Reports
    - Hardware Counters Overview Report
    - Hardware Counters Plot

Resources for Users

- Using Cray Performance Analysis Tools
  - http://docs.cray.com/books/S-2376-50/S-2376-50.pdf
- NCCS website
  - http://www.nccs.gov/
- Cray Documentation
  - http://docs.cray.com/
- Contact us
  - help@nccs.gov