# The New Generation of Cray Performance Tools

**Luiz DeRose, Bill Homer, Dean Johnson, and Steve Kaufmann**
*Cray Inc.*

***ABSTRACT:*** *In order to be able to achieve and sustain high-performance on today's supercomputers, users need a performance tools infrastructure that can handle the challenges associated with complex architectures. In this paper we describe the Cray new generation of performance tools, which were designed and developed to provide the functionality needed for performance analysis of scientific applications in a portable and easy to use way.*

**KEYWORDS:** Programming Environment Tools, Cray X1 Series, Cray XT3, Cray XD1

## 1. Introduction

In order to be able to achieve and sustain high-performance on today's supercomputers, users require performance tools that can correlate the parallel source code with the dynamic performance behavior of applications from both software and hardware measurements, still providing a portable, intuitive, and easy to use interface. At Cray, this challenge was approached with the design and development of an infrastructure that provides the functionality needed for performance measurement and analysis of scientific applications in a portable and easy to use manner. In this paper we present this new generation of Cray performance tools.

The remainder of this paper is organized as follows. In Section 2, we describe the Cray approach to performance tools. In Section 3, we present an overview of the Cray tools. Finally, Section 4 presents our conclusions with a summary of the recent updates and new features on the Cray tools.

## 2. Cray Performance Tools Approach

The performance tools strategy on Cray systems is based on the observation that performance analysis is normally done in multiple steps of a two-phase optimization process: single processing optimization and parallel processor optimization. On parallel vector and scalar systems (with mpi or similar communication mechanisms), users normally split the computation domain in smaller subdomains, which are operated by individual tasks. The users initially try to optimize a single task (including vector optimization) and then look into parallel issues. During the single processor optimization phase, the user attempts to answer the following questions:

1. Does my program have performance problems?
2. Where are the main bottlenecks?
3. What are the causes of the performance problems?

In order to answer the first two questions, users are normally interested in simple tools that can provide an overall view of the program execution and do not require manual instrumentation, while to answer the third question, users are in general willing to instrument the source code, although, the availability of automatic instrumentation is preferred. At this level, hardware performance counters are one the most important asset for the tools to be able to provide detailed information, which can be correlated with the source code and the individual functional units of the system.

During the parallel processing optimization phase, the user attempts to answer the following questions:

1. Does my program have communication or synchronization problems?
2. What are the causes of the parallel performance problems?

In order to answer the first question, users look for a simple to use communication profiling tool, while a tracing infrastructure, integrated with a graphical user interface is the best solution for answering the second question.

## 3 Cray Tools Overview

The new generation of Cray performance tools was designed to address the questions described above. It consists of the **CrayPat** performance analysis toolkit, for program instrumentation, data collection, trace generation, and performance analysis, and the **Cray Apprentice$^2$**, for graphical visualization of the application performance.

## 3.1 CrayPat

CrayPat is a performance analysis toolkit composed of several utilities. The four main components are **pat_hwpc**, **pat_build**, **pat runtime library**, and **pat_report**. The pat_hwpc utility is a stand-alone component, designed to answer question (1) of the single processor optimization phase (although it also works for parallel programs), while the other components of CrayPat were developed to address questions (2) and (3) of the single processor optimization phase, as well as question (1) of the parallel optimization phase.

The **pat_hwpc** utility executes a given application, records specified hardware performance counters events, and at the end of the execution writes a summary report to standard output with times, counts, and derived metrics such as average vector length, megaflop rates, and cache miss rates. Alternately, it can be used to attach to a process that is already executing. Hardware performance counters events and other timing information can also be saved to a file for later evaluation by the CrayPat reporting facility. Currently, pat_hwpc is only supported on the Cray X1 series, but work is under way for support on the Cray XT3 and Cray XD1 systems. However, a library (**libhwpc.a**) is alternatively available for the collection of hardware performance counters events on the Cray XD1 and Cray XT3 systems. Libhwpc provides a programming interface for tracing regions, and supports access to the underlying hardware performance counters using the Performance API (PAPI), developed by the Innovative Computing Laboratory at the University of Tennessee in Knoxville. The user explicitly links the program with "**`-lhwpc`**" and a report is printed to `stdout` at the end of program execution.

**CrayPat** performs *experiments* on running applications. An experiment is an evaluation of an application as it executes. The way that experiments work is determined both by how an application is instrumented, and how it is executed. CrayPat is applied to applications for single or multiple PEs with shared memory (SM) or distributed memory (DM) design. CrayPat also supports threaded applications, including the OpenMP programming model, and for the Cray X1, both MSP and SSP mode applications. CrayPat provides a number of experiments that collect data in different ways. Thus, if several experiments are applied to the same application, the bias implicit in any given experiment is rendered acceptable.

Instrumentation of an application is the first preparatory step required for performance evaluation. Instrumentation sets up the capture of software state, hardware state and time:

- The Software state can include thread and call stack information or the actual parameter values passed into a function entry point.
- The Hardware state can include the Program Counter (PC) or some Hardware Performance Counter (HWPC) event values.
- Time stamps are recorded in high resolution using the Real-Time Clock (RTC) and HWPC cycle counter.

The instrumented application is executed in the same manner and in the same environment as the original application. It can be executed multiple times with varying data sets, each iteration producing a new experiment data file. The CrayPat reporting features can accept multiple experiment data files for a single application - the more material, the more complete and thorough the performance evaluation.

CrayPat does not require that applications or parts of applications to be recompiled. A single link, managed by CrayPat, is all that is required. As part of the CrayPat environment, link details are embedded in the executable file. CrayPat uses these details to create the link operands and the instrumented application. The original application is not changed.

The utility **pat_build** is used for automatic instrumentation of the application, for performance data collection. It allows the user to define when the performance collection is triggered and how the performance data is recorded.

The triggering mechanism can be activated by an external agent, such as a timer or a hardware counter overflow, which is known as *asynchronous* or *sampling*, or internally, by code inserted through instrumentation, which is normally referred as *synchronous* or *event tracing*.

If an application is instrumented for an asynchronous experiment, the nature of the experiment is selected at run-time. Asynchronous experiments are statistical: they sample the state of the application at given intervals. The interval can be a time interval (for example, every 10 milliseconds), or it can be a hardware performance counters event that overflows at defined value. The types of asynchronous experiments include: PCs from OS-based profiling and PCs from sampling via interval timers. When sampling, other state information can be recorded at the time the PC is recorded. Among other information are the call stack, dynamic heap, system resources, and the values of selected hardware performance counters. Currently; the Cray XT3 system does not support sampling. We are investigating the possibility of providing sampling based on hardware counter overflow.

If an application is instrumented for a synchronous experiment, function entry points are counted and recorded. At the time of instrumentation, the user chooses which function entry points to record. All function entry points that have predefined "trace wrappers" can be traced. In addition, CrayPat provides an Application Programming Interface (API) for finer control over the recording of the state during run-time. The API

encompasses a number of functions that can be inserted into the application source code. These functions are only activated in the instrumented program. The API facilitates recording similar state to tracing, but for regions within a function. API functions are provided for both C and Fortran.

Performance data can be summarized during runtime and stored in the form of a *profile*, or can be saved in the form of a *trace file*, where for each traced event (instrumented functions and PAT API calls) that is executed during run-time, a tracing record is created in the experiment data file. Profiling produce the most compact experiment data files. On the other hand, with a trace file, one can reproduce the complete performance behaviour of the application.

A number of trace function groups are predefined. They represent function entry points that are related in function and application. These groups include:

- MPI, SHMEM, UPC, CAF
- OpenMP
- Pthreads
- System Calls
- Dynamic Heap
- ANSI Math
- Raw I/O, buffered I/O, flexible file I/O

Instrumentation uses the application itself to collect state and timing information. Pat_build manages the link of the original program with the **pat runtime library** that facilitates this data collection, creating an "*instrumented*" executable, which is executed the same way as the original application. During the course of the instrumented program execution, an experiment data file is created containing recorded state and event information.

The **pat_report** utility analyses the state and event data in the experiment data file, created as a result of executing the instrumented program. It produces a text report, which users can customize for content and format. The pat_report utility can aggregate data or keep it segregated by SSP (Cray X1 only), thread, and process. Reports display such detail as hardware performance counters event values, call trees, and special processing for the function groups mentioned earlier. In addition, pat_report is also used to generate the input file used by the Cray Apprentice$^2$ visualization component.

Additionally, a couple of utilities are available to improve easy of use: **pat_run** and **pat_help**. The **pat_run** utility provides an alternate interface to the CrayPat collection of tools. It combines in a single invocation the selection of data to be collected with the execution of the instrumented program, and automatically produces an appropriate report. The **pat_help** utility provides a text-based interactive help facility for CrayPat, where one can access information about and examples of using the CrayPat performance analysis tool.

## 3.2 Cray Apprentice$^2$

**Cray Apprentice$^2$** is a multi-platform, multi-function performance data visualization tool that takes as input the performance file generated by CrayPat. It provides the familiar notebook-style tabbed user interface and can display a variety of different data panels, depending on the type of performance experiment that was conducted with CrayPat. It has an identical interface between the Cray product lines. Cray Apprentice$^2$ is a visualization mechanism designed to address questions (2) of the single processor optimization phase, as well as question (2) of the parallel optimization phase mentioned in Section 2.

Cray Apprentice$^2$ is built with scaling in mind. It is built with the idea of many threads of execution and large amounts of performance data, all within a single tabbed window interface. Information is typically presented first at the most programmer-centric level, such and routines and then allowing for "diving" deeper into the data, showing things like a breakdown by PE.

Cray Apprentice$^2$ provides call-graph based profile information, as shown in Figure 1, which can be mapped to the source code. The profile view contains three panes: The bottom left pane provides an overall view of the application call graph. The right side pane displays a zoom of the call graph, which is selected using the window available in the overview pane. Finally, a text profile sorted by time or calls is available on the top left pane. The sizes of the boxes are dependent on the execution time or in the sampling case, on the number of hits. Thus, using the profile view, one can quickly identify the functions or regions in the program that are possible causes of performance bottlenecks.

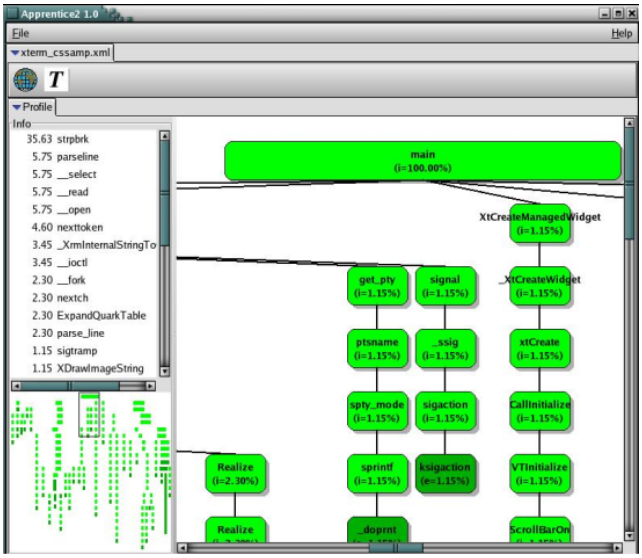


**Figure 1: Cray Apprentice$^2$ profile view**

Summary information about communication and user functions, instrumented regions, as well as I/O, can also be obtained graphically with standard statistics displays, as shown in Figure 2, which shows MPI statistics and in Figure 3, which shows pair-wise communication statistics.

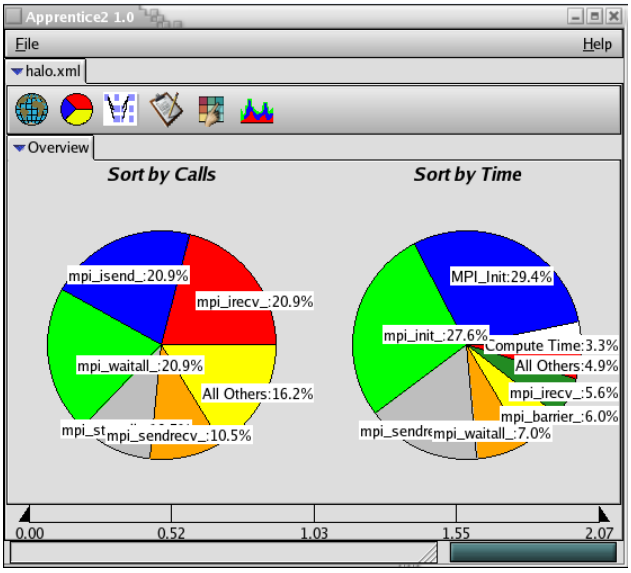


**Figure 2: Cray Apprentice$^2$ statistics view**

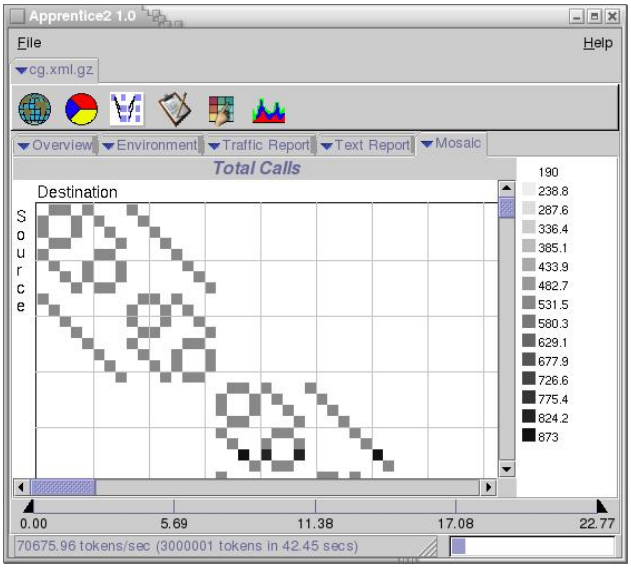


**Figure 3: Cray Apprentice$^2$ pair-wise communication statistics**

In addition, time line based trace visualization, supporting the traditional parallel processing and communication mechanisms, such as MPI and OpenMP are available, as shown in Figure 4, which presents the time line view, and Figure 5, which presents an activity view. Performance visualization for I/O is also available, as shown in Figure 6.

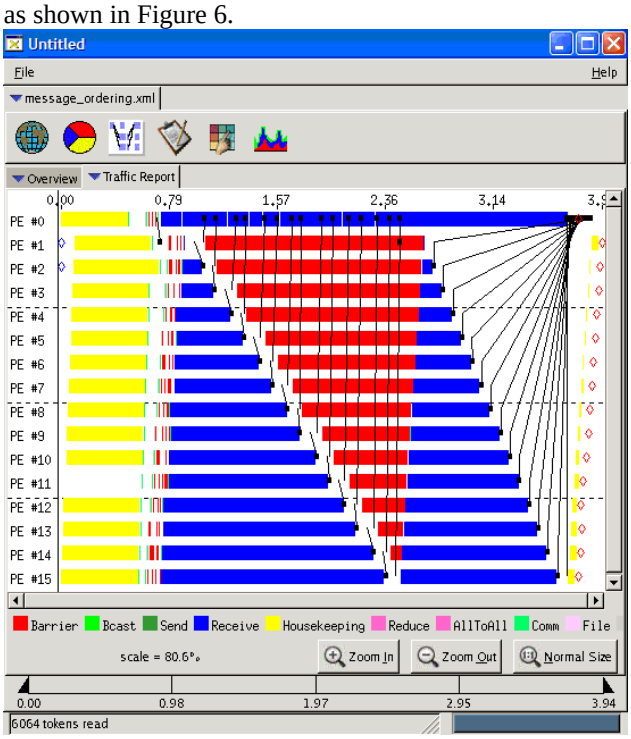


**Figure 4: Cray Apprentice$^2$ time line view**

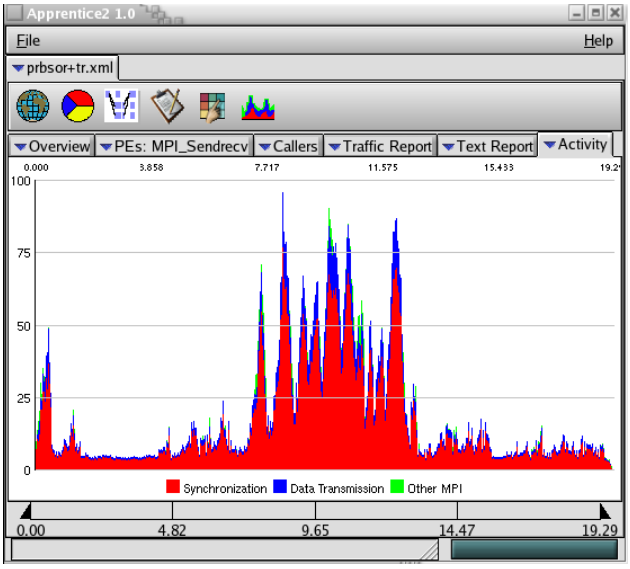


**Figure 5: Cray Apprentice$^2$ activity view**

All Cray Apprentice$^2$ displays are integrated into the same *tabbed* graphical interface, allowing one to switch easily from one display to another. In addition, Cray Apprentice$^2$ uses the concept of calipers, which synchronizes the time frame on all displays, providing the

flexibility for users to select a time frame for analysis on one display, without having to reset all other displays.

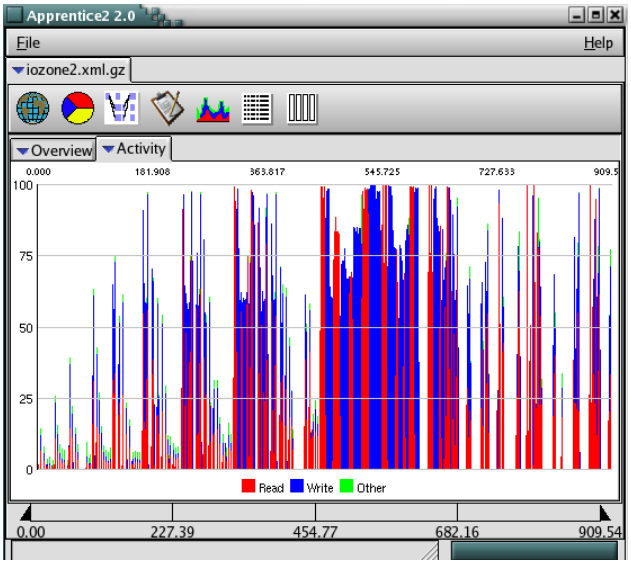


**Figure 6: Cray Apprentice$^2$ I/O visualization**

Additionally, Cray Apprentice$^2$ provides displays with performance information in text mode, as well as a complete on-line help.

## 4 Conclusions

In this paper we presented an overview of the new generation of Cray tools, which was developed to provide a portable and easy to use performance tools infrastructure for all Cray systems.

Over the last year, the Cray tools infrastructure has been improved in a number of areas, including the addition of the following features:

- The Cray Apprentice$^2$ graphical visualization interface. Among its main features are multiple views of data, including profile, summary statistics, call graph, and time line.
- CrayPat availability on all three main Cray systems: X1 Series, XT3, and XD1, as well as on the Linux Cross Compiler package.
- The pat_run utility, which combines selection of metrics, execution, and report generation into a single step.
- Run time summarization (RTS) of data, which is now the default. This means a partial aggregation of data during the execution of the instrumented program, which dramatically decreases both the size of the data file and the time required to generate a report.
- Improved support for OpenMP programs, with the ability to show load balance across threads in parallel regions.
- Support for MPMD programs (multiple program executables).
- New pair of API calls, **`PAT_region_begin`** and **`PAT_region_end`**, to designate regions of interest within a function.
- Reports showing callers or a calltree can now show the line numbers for each call.
- Pruning of library "internals" from callers/calltree views and reports.

## Acknowledgements

The authors would like to thank colleagues of the Programming Environments and Testing group, and users in the Benchmarking and Applications group at Cray Inc., for their contributions during the development, implementation, and testing of CrayPat. In addition, we would like to thank the PAPI team at the University of Tennessee in Knoxville for the help with the PAPI port to the XT3.

## References

All components of the CrayPat performance toolkit and related information are located in the man pages.

Details about the use and description of the PAPI are in the **`papi(3)`** man pages, the *PAPI User's Guide* (number S-6515), and the *PAPI Programmer's Reference* (number S-6514), as well as on the PAPI page: http://icl.cs.utk.edu/papi/

Details about the use and description of the hardware performance counters are in the **`counters(5)`** and **`papi_counters(5)`** man page.

The following publications provide more information on CrayPat and Cray Apprentice$^2$:

*Optimizing Applications on the Cray X1 System* (number S-2315).

*Cray XT3 Programming Environment User's Guide* (number S-2396)

*Cray XT3 System Overview* (number S-2423).

## About the Authors

Dr. Luiz DeRose is a Sr. Technical Engineer and the Programming Environment Tools Manager at Cray Inc. He has twenty years of experience designing and developing tools for HPC. He can be reached at 1340 Mendota Heights Rd, Mendota Heights, MN 55120 USA, E-mail: ldr@cray,.com

Bill Homer, Dean Johnson, and Steve Kaufmann are Software Engineers in the Programming Environment Tools group at Cray Inc. They have many years of experience developing performance tools for high performance computation. They can be reached at 1340 Mendota Heights Rd, Mendota Heights, MN 55120 USA. Their email addresses are homer@cray.com, dtj@cray.com, and sbk@cray.com respectively.