Evolution of the Cray Performance Measurement and Analysis Tools

Heidi Poxon, *Cray Inc.*

ABSTRACT: *The goal of the Cray Performance Measurement and Analysis Tools is to help the user identify important and meaningful information from potentially massive data sets by providing hints around problem areas instead of just reporting raw data. Analysis of data that addresses multiple dimensions of scalability including millions of lines of code, lots of processes or threads and long running applications is needed. The Cray toolset supports these dimensions by collecting information at process and thread levels, and providing features such as load imbalance analysis, derived metrics based on hardware events, and optimal MPI rank placement strategies. This paper focuses on recent additions to the performance tools to enhance the analysis experience and support new architectures such as hybrid X86 and GPU systems. Work presented includes support for applications using PGAS programming models, loop work estimates that help identify parallel or accelerator loop candidates, and statistics around accelerated loops.*

KEYWORDS: Programming Environment, performance tools, GPUs

1. Introduction

The Cray Performance Measurement and Analysis Tools are set on an evolutionary path to address the application performance analysis challenges associated with the next generation systems. Current performance data collection techniques can produce excessive amounts of information, making it extremely difficult for users to correlate observations from data to understand performance behavior. In addition, the vast amounts of data generated for performance analysis degrades current tool response time and usability. Enhancements to the Cray performance tools have been evolving the software to better manage data collection and presentation as well as provide derived metrics and tips to help the user isolate performance issues within an application. Over the next several years it is projected that there will be a dramatic increase in node concurrency; from approximately 24 per node to between 1000 and 10,000. The first example of this is the new hybrid X86 and GPU systems. Several features are being added this year to the Cray performance toolset to support better analysis of programs that want to take advantage of hybrid systems. After an overview of the tools, the following sections present recent work and work under development that focuses on performance analysis assistance for these next generation systems.

1.1 Overview of the Cray performance tools

The Cray performance toolset provides an integrated infrastructure for measurement and analysis of computation, communication, I/O, and memory utilization. It allows developers to perform trace experiments on single-processor or multiple-processor executables at the binary level with function and loop granularity. It supports the MPI and OpenMP programming models, as well as the PGAS and Chapel parallel programming languages.

The Cray Performance Measurement and Analysis Tools consist of components that prepare a program for performance analysis experiments, capture performance data during program execution, process and analyze the data, and present performance results to the user in both a

text report and through an interactive graphical user interface.

CrayPat is the data capture tool, which is used to prepare user programs for performance analysis experiments, to specify the kind of data to be captured during program execution, and to prepare the captured data for text reports or for use with other programs.

Performance data is captured during application execution by sampling at intervals, or upon entry/return from traced functions, and is recorded in the form of a summarization of events over time (profile), or a sequence of events of time (trace). Each process collects its own performance data. Per process buffers in memory are used to temporarily store local collected performance data. The data in these buffers is later flushed to a performance log file on a parallel file system.

The user can optionally control the behavior of the instrumented program during execution through a set of runtime environment variables that affect what and how the performance data is collected. Examples of this include the enabling of predefined hardware counter groups that track chosen sets of hardware events, the ability to choose the mechanism to use to sample the application, and the ability to modify the number of data files that are written in parallel by the processes. By default, a runtime summarization of the data is provided, which involves aggregation of the data.

Through higher-level derived metrics, the toolset helps identify the "why" to unexpected performance, so the application developer can more quickly identify the source of intra-node performance bottlenecks.

The pat_report utility available in the toolset performs two functions. It reads state and event data in the performance file created by the runtime library, and generates text reports according to the groups selected, presented in table format. Reports display such detail as hardware performance counters event values, call trees, and special processing for the function groups. One of the strengths of this utility is that it can be run several times against the same collected performance data to provide different combinations of data, so that the user can choose the subset from the collected data that best suits their needs.

Cray Apprentice[2] displays data that was captured by CrayPat. This visualization tool displays a variety of different data panels, depending on the type of performance experiment that was conducted. Its target is to help identify conditions including load imbalance, excessive serialization, excessive communication and network contention.

2. Support for hybrid X86 and GPU systems

2.1 Loop statistics

To help application developers transition their programs to take advantage of increased on-node concurrency, the Cray performance measurement and analysis tools have added loop statistics to help the user find additional parallelism. In addition to collecting sampling information to identify the top time consuming functions within a program, the user can collect timing and iteration counts for serial loops to get an estimate for the amount of work performed in a loop or multi-level loop nest. This can help determine whether or not a particular loop would then benefit from being parallelized for execution on the X86 in a multi-core environment and/or benefit from being executed on a GPU.

This functionality is available for programs that are built with the Cray Compiling Environment (CCE). If the user specifies `-h profile_generate` when compiling and linking their program, CCE will measure timing and loop trip counts for all loops within the compiled functions. The loop timing statistics provide a rough estimate for the amount of work in the loop, and the loop trip counts can be used to help carve up the loop on the GPU. It should be noted that these measurements are done for all loops (inner and outer) and even though vectorization is still preserved within the inner loop, additional optimizations that CCE would normally do may not be present during this experiment. Additional performance analysis should be done in separate experiments with full CCE optimization.

Loops are identified by the function where they reside, their nesting level and by their source line number. Loop statistics presented in the default report provide inclusive times shown as a percentage of the overall time, the number of times a loop is called during program execution, the average trip count, and optimization feedback / hints from the compiler which are defined in the table's notes section. The following example shows loop statistics provided by pat_report.

```
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.


Table 2:  Loop Stats from -hprofile_generate

   Loop |Loop Incl |Loop Incl |  Loop |  Loop |    Loop |Function=/.LOOP\.
   Incl |     Time |   Time / |   Hit | Trips |   Notes | PE='HIDE'
 Time / |          |      Hit |       |   Avg |         |
  Total |          |          |       |       |         |

|-------------------------------------------------------------------------
|  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
|=========================================================================
```

2.2 GPU statistics

In addition to providing support to help identify serial loops within a program that would be worth parallelizing, the Cray performance toolset is adding statistics for accelerated regions. These statistics give the user feedback on how well an accelerated region performed within their overall application, and how well it performed on the GPU. Advantages that the Cray tools offer over GPU-specific profilers include summarized results that are consolidated in one place, statistics mapped back to the user source by line number and grouped by OpenMP accelerator directive, and statistics tied to the program as a whole.

Current development work is focused on providing performance statistics that include host time for kernel launches, data copies and synchronization with the GPU, GPU time for kernel execution and data copies, and the number of times each accelerated region was called during program execution. Events associated with an accelerated region are identified by the function where they reside, the type of event (async_copy, async_kernel, etc), and by the source line number. Kernel level statistics will also be provided which include information on memory usage, grid size, block size, etc. And to better understand how an accelerated region is performing on the GPU, GPU hardware counter statistics will be available.

When a program is built with the CCE compiler and contains accelerated regions through OpenMP directives, statistics are automatically collected for the user and presented in the default report. When a program is built

with the PGI compiler and contains PGI `acc` directives, or contains GPU kernels generated using the CUDA API, the user can use the `PAT_region` API to bracket accelerated regions within the source for data collection and presentation by the Cray performance tools. The following example shows accelerated region timing statistics provided by pat_report.

```
Notes for table 3:

  Table option:
    -O accelerator

  The Group value for each data item is the avg of the PE values.
  The PE value for each data item is the max of the Thread values.
  The Thread value for each data item is the sum of the Calltree values.
  The Calltree value for each data item is the sum of the Function values.
    (To specify different aggregations, see:  pat_help report options s1)

  Percentages at each level are of the Total for the program.
    (For percentages relative to next level up, specify:
      -s percent=r[elative])
  For synchronous accelerator events Acc Time is set equal to Host Time.


Table 3:  Time and Bytes Transferred for Accelerator Regions

   Host | Host Time |  Acc Time |  Acc Copy |  Acc Copy | Calls |Group='ACCELERATOR'
 Time % |           |           |   In (MB) |  Out (MB) |       | PE=0
        |           |           |           |           |       | Thread=0
        |           |           |           |           |       | Calltree
        |           |           |           |           |       |  Function

 100.0% | 14.84495 | 13.615016 | 14550.536 | 10461.216 |  1777 |Total
|-------------------------------------------------------------------------------
| 100.0% | 14.84495 | 13.615016 | 14550.536 | 10461.216 |  1777 |ACCELERATOR
||------------------------------------------------------------------------------
||  93.7% | 13.909414 | 12.418942 | 13274.781 |  9675.075 |  1777 |mg_
|||-----------------------------------------------------------------------------
3||  51.8% |  7.692439 |  7.645484 |  7902.816 |  6399.489 |  1630 |mg3p_
||||----------------------------------------------------------------------------
4|||  21.7% |  3.229140 |  3.216513 |  3758.31 |  2254.986 |   420 |resid_
|||||---------------------------------------------------------------------------
5|||| 11.9% |  1.767674 |  1.763377 |  2254.986 |   751.662 |   140 |resid_(exclusive)
||||||--------------------------------------------------------------------------
6|||||   7.8% |  1.158744 |  1.158958 |  2254.986 |     0.000 |    35 |resid_.ASYNC_COPY@li.459
6|||||   4.1% |  0.604365 |  0.337742 |     0.000 |   751.662 |    35 |resid_.ASYNC_COPY@li.492
6|||||   0.0% |  0.003903 |  0.000000 |     0.000 |     0.000 |    35 |resid_.SYNC_WAIT@li.492
6|||||   0.0% |  0.000662 |  0.266677 |     0.000 |     0.000 |    35 |resid_.ASYNC_KERNEL@li.459
|||||==========================================================================
5||||  9.9% |  1.461466 |  1.453136 |  1503.324 |  1503.324 |   280 |comm3_
||||||--------------------------------------------------------------------------
6|||||   2.6% |  0.384892 |  0.387225 |   751.662 |     0.000 |    35 |comm3_.ASYNC_COPY@li.1093
6|||||   2.6% |  0.384830 |  0.387166 |   751.662 |     0.000 |    35 |comm3_.ASYNC_COPY@li.1063
6|||||   2.3% |  0.341989 |  0.337894 |     0.000 |   751.662 |    35 |comm3_.ASYNC_COPY@li.1092
6|||||   2.3% |  0.340363 |  0.337982 |     0.000 |   751.662 |    35 |comm3_.ASYNC_COPY@li.1106
6|||||   0.0% |  0.003913 |  0.000000 |     0.000 |     0.000 |    35 |comm3_.SYNC_WAIT@li.1092
6|||||   0.0% |  0.003888 |  0.000000 |     0.000 |     0.000 |    35 |comm3_.SYNC_WAIT@li.1106
6|||||   0.0% |  0.000805 |  0.000584 |     0.000 |     0.000 |    35 |comm3_.ASYNC_KERNEL@li.1093
6|||||   0.0% |  0.000786 |  0.002285 |     0.000 |     0.000 |    35 |comm3_.ASYNC_KERNEL@li.1063
||||||==========================================================================
...
```

3. Recent additions to the performance tools

In addition to enhancements for hybrid GPU systems, the following recent additions to the Cray performance tools have been made to improve usability, scalability, and new architecture support.

3.1 New product license and access

Starting with the 5.1.0 release of the Cray performance tools, the CrayPat and Cray Apprentic2 products have been combined into a single product called the Cray Performance Measurement and Analysis Tools (CPMAT). This change was made to simplify licenses, packaging, and to address software dependencies that exist between the products. In addition, license check support has been added to the product using the FLEXlm license manager.

Access to the software has been simplified so that users no longer need to load two modulefiles to access the performance tools. Loading the *perftools* modulefile will set the user's environment for access to CrayPat (pat_build, pat_report, etc.), PAPI and Cray Apprentice[2] (app2).

3.2 Support for Gemini network counters

Access to Gemini network counters was added to the CPMAT 5.1.0 release (available June 2010). Access to these counters is only available through CrayPat. Users can collect network counter event information to understand how much traffic they are sending to and from the network, or to isolate nodes where network traffic results in delays within their application.

When an application is instrumented to collect network counter events, values are recorded at runtime and presented to the user in the default report. The CrayPat user interface for requesting instrumentation is similar to that for CPU hardware counter events and is specified through a set of environment variables. The following example shows the presentation of counter events averaged across the nodes for the job.

```
    Table 2:  NWPC Data by Function Group and
Function

    Group / Function / Node Id='HIDE'
    =============================================
    Total
    ---------------------------------------------
      Time%                               100.0%
      Time                              1.848819s
      GM_ORB_PERF_VC1_STALLED               8175
      GM_ORB_PERF_VC1_BLOCKED                  0
      GM_ORB_PERF_VC1_BLOCKED_PKT_GEN
                                           26206
      GM_ORB_PERF_VC1_PKTS                 26628
      GM_ORB_PERF_VC1_FLITS               120606
      GM_ORB_PERF_VC0_STALLED              27549
      GM_ORB_PERF_VC0_PKTS                 26319
      GM_ORB_PERF_VC0_FLITS                54780
      GM_AMO_PERF_COUNTER_EN                   0
      GM_AMO_PERF_CQ_FLIT_CNTR              4796
      GM_AMO_PERF_CQ_PKT_CNTR               2396
      GM_AMO_PERF_CQ_STALLED_CNTR              0
      GM_AMO_PERF_CQ_BLOCKED_CNTR              0
    =============================================
```

Documentation on Gemini network counters and how to access them through CrayPat is available in the "Using the Cray Gemini Hardware Counters" technical note available in the Knowledge Base on http://docs.cray.com/.

3.3 New format for processed performance data

To improve scalability and better support performance measurement and analysis of larger jobs, a new more scalable .ap2 file data format was recently introduced. This functionality is mostly transparent to the user. Users benefit from greatly reduced pat_report processing and report generation times, as well as Cray Apprentice[2] data load times. The following table shows example improvements of data processing and report generation times.

Table 1: Data and report processing times

	Perftools 5.1.3 (seconds)	Perftools 5.2.0 (seconds)
CPMD (960 cores)		
.xf -> .ap2	88.5	22.9
.ap2 -> report	1512.3	49.6
VASP (768 cores)		
.xf -> .ap2	45.3	15.9
.ap2 -> report	796.9	28.0

3.4 Client/server model

To further improve tool response time, a new distributed Cray Apprentice[2] client for Linux has been introduced so that the graphical presentation is handled locally and not passed through the ssh connection between the user's laptop and the Cray service node. Prior to a client/server model, all of the performance data collected from an experiment needed to be loaded into memory before any results were displayed. This created size limitations as well as long load times. The combination of the new data format and this new client/server model minimizes the amount of data loaded into memory at any given time and thus creates a smaller footprint on the Cray service node. Development of clients targeted for Mac and Windows laptops are also in the works.

3.5 PGAS support

The performance tools have been enhanced to provide a simplified procedure for collecting performance statistics for programs that use the UPC or Co-array Fortran programming models. The procedure is now similar to that for MPI programs. Users can use the Automatic Profiling Analysis feature (`-O apa`) on PGAS programs to get a profile that associates time back to the original source lines in the program. The `-g upc` or `-g caf` predefined wrappers are automatically added with `-O apa`, and internal levels of pgas runtime routines are pruned from reports, much like is done for internal calls to Portals routines for MPI programs.

4. A look into the future

To support application performance tuning and optimization on the next generation Cray systems, an advanced performance analysis tool is currently being developed. This new functionality will extend Cray's existing performance measurement, analysis and visualization technology by combing performance statistics, program source code visualization with Cray compiler optimization feedback, and the ability to easily navigate through source to highlighted dependencies or bottlenecks during the optimization phase of program development or porting.

The user will be able to navigate through his or her source code using the application information database information provided by the Cray Compiling Environment (CCE) and the performance data collected by the Cray performance toolset, to understand which high-level loops could benefit from multiple levels of parallelism. The tool will provide dependency information for those loops, as well as assist the user when parallelizing a loop.

7. Conclusion

The Cray Performance, Measurement and Analysis Tools are set on a path to handle larger jobs, provide a better user experience, and to offer needed assistance to users moving to the next generation many-core / hybrid systems. As levels of on-node concurrency increase, the performance tools will continue to look for intuitive ways to direct the user to relevant bottlenecks that need attention.

Acknowledgments

The author would like to thank her colleagues and our users for the wealth of insight and feedback provided over the years, which have helped us target needed functionality and offer state of the art performance tools.