Performance Tools Update

Mary Kay Bunde, SGI/Cray

ABSTRACT: *Two new performance tools are available for application programmers to use when tuning their T3E or Origin 2000 codes. In addition to the existing MPP Apprentice performance tool, the Cray-T3E offers a new Performance Analysis Tool, PAT. SpeedShop was introduced last summer on MIPS-based architectures. It is available on the Origin 2000. This paper describes the current features of these tools, and highlights some upcoming plans.*

1 Performance Analysis Tool, PAT

PAT, the new MPP performance analysis tool, will be released for the first time with the Cray Programming Environment 3.0 release. When the CRAY-T3E hardware and UNICOS/mk operating system were designed, new functionality became available that was beneficial to make use of in application performance tools. First, unlike the Alpha chip used on the CRAY-T3D, hardware performance counters are available on the EV5/6 chip to assist in monitoring performance. Second, the UNICOS/mk operating system supports the profil() system call, which was not available on UNICOS-MAX. Further, CRAY-T3D users continually expressed the need for a less intrusive performance tool and and access to harware performance monitor information. The Visual Tools group lobbied hard with these other groups to bring this functionality to you.

Pat provides a fast, low-overhead method for estimating the amount of time spent in programs at the function level, determining how a program's load is balanced across PEs, and giving users access to the hardware performance monitors, as well as other features described below. Today's codes are becoming larger, longer running, with larger datasets, and are complicated by the programming complexity new architectures require. For this reason, users need simple tools to help them understand how the code is behaving on complex architectures. With Pat, a user's first look at a program's performance can be done by merely relinking the program with the `libpat.a` library. No recompilation is necessary to obtain execution time and load balance information at the subroutine level. With this easy first look at a program's performance, users have very quick insight into which routines may be bottlenecks, or troublesome, to the program's performance. After this information is determined, the user can decide the best next step to analyzing the program's performance. Pat is not a replacement for the MPP Apprentice. Often times, a user will then instrument very specific portions of code and analyze performance to the block level with the MPP Apprentice.

1.1 Current functionality

Pat will allow users to analyze a code's performance with the following features:

- Display the percentage of execution time spent in functions (profiling). The estimated percent of execution spent for each function is shown. Since this number is statistical, a confidence interval is also given.

  For example:

  ```
  Function        user%  +- conf
  -----------     -----   --------
  func1             64      +- 2.1
  func2             36      +- 2.1
  ```

  Since profiling information is provided on a modified executable, rather than instrumented source, users can obtain performance information not only on their complete code, but also on intrinsics and library routines for which they have no source code.
- Determine how the program's load is balanced across PEs. This is done by using profiling to gather data at the function level on each PE.
- Generate trace files, to display when various events occurred on different PEs. The tracing feature "wraps" requested events with timing routines, such that the generated output shows what events happened at specific times for each PE. This feature was written so that message passing events could be traced. Wrapper routines are required to use this feature. They are provided for the MPI libraries. In the future, other wrapper routines will be available.
- Time calls to individual subroutines (call site reporting). For a specified function and PE, pat will provide you the number of times a function was called, what functions called it (callers), the total time spent in the function (inclusive of any callees), and the average amount of time spent per call.

Both Pat and the MPP Apprentice provide call site instrumentation. The times that Pat reports are inclusive; that is the times include the current function and all it's descendents (callees). The MPP Apprentice provides an option to see exclusive times as well; that is, it will report time from the desired routine only, and exclude time spent in routines it called.

- Display hardware performance counter information at the program level. The CRAY-T3E has 3 hardware counters through which you can obtain performance information. The following performance data can be gathered (specified by environment variables that `libapp.a` reads):

  -Number of integer operations

  -Number of floating point operations

  -Number of loads

  -Number of stores

  -Number of data cache misses

As noted above, for most features, Pat does not require a user to recompile the program, and it is less intrusive while the program is running and gathering performance data. With the MPP Apprentice, on average a program instrumented to collect Apprentice data runs 3 times longer than it does with no instrumentation. Pat doesn't incur this kind of overhead. For a Gaussian code, user time increased by less than 1% while generating PAT performance data; system time increased by 15%.

1.2 Future Plans

In the near future, the following improvements are planned for Pat:

- Support for pthreaded programs. Performance counter information will be available on a per-thread basis, however, profiling is done per process, not per thread.
- A graphical user interface
- Wrapper routines for I/O libraries, such that I/O events can be traced through Pat's event trace mechanism. Also, a template will be provided so that users can write their own wrapper routines.
- Performance counter information at the function level (inclusive and exclusive)
- Exclusive time for call site data

2 SpeedShop

SpeedShop is the newest SGI performance tool, released for the first time last summer. SpeedShop runs on all MIPs-based hardware platforms, and has many similar features to Cray performance tools. While SpeedShop is a multi-PE performance tool, we have no experience with SpeedShop at large numbers (>64?) of PEs, therefore, we've yet to determine how this product will scale. This section describes the features currently in SpeedShop, and some differences from the Cray performance tools that are important to note.

2.1 Current functionality

SpeedShop has the following features available to analyze a program's performance:

- PC sampling. This can be done down to the instruction level. It's important to note that Cray performance tools have never sampled performance data to the instruction level. The lowest level Cray performance tools have sampled is the basic block level.
- Time spent at the routine level. This information is obtained by profiling the call stack. Therefore, called to and called from information can be obtained (profile data).
- Instrumentation data. SpeedShop uses a utility, called pixie, to instrument and provide feedback on a user's code. Instrumentation of a users code can provide the following performance feedback:

  -ideal time (exclusive and inclusive of called routines)

  -instruction counts

  -load counts

  -store counts

  -floating point operations

  -exclusive instruction coverage

  -branch coverage

  -cycles per instruction count

  While this type of instrumentation provides an abundance of useful information, it is very expensive. Typically, an instrumented code performs 10 times slower than its uninstrumented original code. It's also important to note that when instrumentation is turned on, some other SpeedShop functionality will not be available.
- Trace floating point exceptions. This functionality is necessary on SGI systems, as they try to recover from floating point exceptions, and Cray has always aborted on a floating point exception.
- R10K performance counter information. The R10K has hardware performance monitors available to provide some performance information. Note that only 2 types of information can be gathered at a time, and this information cannot be gathered at the same time as information generated from pixie instrumentation. Further, performance monitor information is gathered by profiling the hardware counters. This is a less accurate method than is used on Cray architectures. The following data can be gathered:

  -Graduated instructions. That is, every instruction that was completed, not all that have been issued

  -Cycle counts.

  -Instruction cache misses, both primary and secondary.

  -Data cache misses, both primary and secondary.

  -Page faults.

-Graduated floating point instructions. That is, every floating point instructioncompleted, not all that have been issued.

In addition to the differences noted above, you'll notice that SpeedShop has no graphical user interface. This is likely to remain the case, so that efforts can be put towards the next generation toolset for the Origin platforms.

2.2 Future plans

SpeedShop will continue to be maintained until the next generation toolset is available on the Origin 2000 and later platforms. This next generation toolset is currently called Caribou. It is an integrated development environment, where debugging, performance analysis, and static analysis are all available from one interface. Some benefits of Caribou:

- One interface to learn; one look and feel
- Support for Fortran, C++/C, and shared memory programming models
- There is no need to switch tools when going from debugging to static analysis, or performance analysis. As a matter of fact, Caribou will be integrated to the point where the notion of doing static analysis disappears within the context of debugging and performance analysis. For example, if I'm running my program through the context of the debugger, and I want to see a visual representation of who calls the function I'm in, a button click will give me that information. Or, if performance data is available, I can see how many times function `foo` was called while I'm in the debugging context.
- Caribou leverages the strengths of its predecessors, that being WorkShop and CrayTools. The SGI tools bring C++ and visualization strengths to the table; CrayTools brings F90, scalability, and supercomputing requirements experience. Combining the strengths of the prior toolsets brings you state of the art tools to help you gain insight into your program's behavior quicker than ever before.
- Caribou is very extensible. Caribou will provide access to outside tools (for example, data visualizers). It will have a scripting language to allow users to automate repetitive tasks. It has been designed such that future enhancements within the tool will be easier than in prior tools. This way, new functionality can be added to Caribou instead of having to create new tools.

There are no plans to release Caribou on Cray MPP or PVP arcitectures. Caribou is currently under development, and is expected to Beta by the end of the year. It will release sometime in the first half of 1998.