CS533 Project Report: Tiptop

Kevin Dyer and Paul Vu

November 21, 2012

1 Introduction

Hardware performance counters are architecture-specific special-purpose registers integrated into modern CPUs, and can be used to profile applications. As an example, modern Intel processors support hardware counters that report CPU utilization, L1 cache misses, branch mispredictions and a host of other CPU-specific performance indicators. This information is invaluable when profiling low-level software performance. What is more, compared to traditional software-based profiling, hardware performance counters incur negligible overhead because they are integrated into the hardware. However, this performance advantage comes at a price. Support for hardware counters is still in early stages [**?**], CPUs from the same vendor sometimes have different interfaces [**?**] and support is integrated only into the most recent linux kernel [**?**]. Indeed, hardware performance counters are still nontrivial to use.

Tiptop [**?**] is an application developed by Erven Rohou [**?**] and is an attempt towards an intuitive interface for accessing hardware performance counters in modern CPUs. The interface for tiptop is similar to the popular top [**?**] utility. In tiptop's default configuration, it provides real-time statistics about running applications and their hardware performance counter values. The values that appear on the real-time screen are configurable by the user and through use of an XML configuration file, multiple screens can be configured and flipped through in real-time. In Figure 1 we have a screenshot of tiptop's default screen, and highlight a few of its key features. The full breadth of hardware performance counters supported by tiptop are too many to list here, for a full list of features see the tiptop man page.

Despite the ease of use of tiptop, and its expansive set of hardware counters it supports via XML configuration, we identified a number of limitations. In our initial investigation of tiptop we identified two bugs, we have confirmed these through personal communications with lead developer Erven Rohou. For the first bug, we identified a solution that minimizes the probability that the bug occurs, and will work with Erven Rohou to include our patch into the main branch of the tiptop source. The second bug remains unresolved, but can avoided by running tiptop as root, and we hope to identify a better solution for the final version of this project. In addition to fixing existing bugs, we identified two feature enhancements for tiptop. First, we now include the number of threads per-process as a column in the default tiptop screen, and also provided hooks to allow a user to specify the number of threads column in custom screens, via XML configuration. Then, as a substantial feature enhancement, we have integrated the cross-platform Performance Application Programming Interface (PAPI) [**?**] library into tiptop. Integration with the PAPI library increases the breadth of platforms and kernel versions supported by tiptop, and simplifies the process of specifying XML configuration files that define custom tiptop screens.

In section 2 we give an overview of our high-level development cycle and the tools we used to repair and enhance tiptop. In section 3 we give further details about the bugfixes and feature

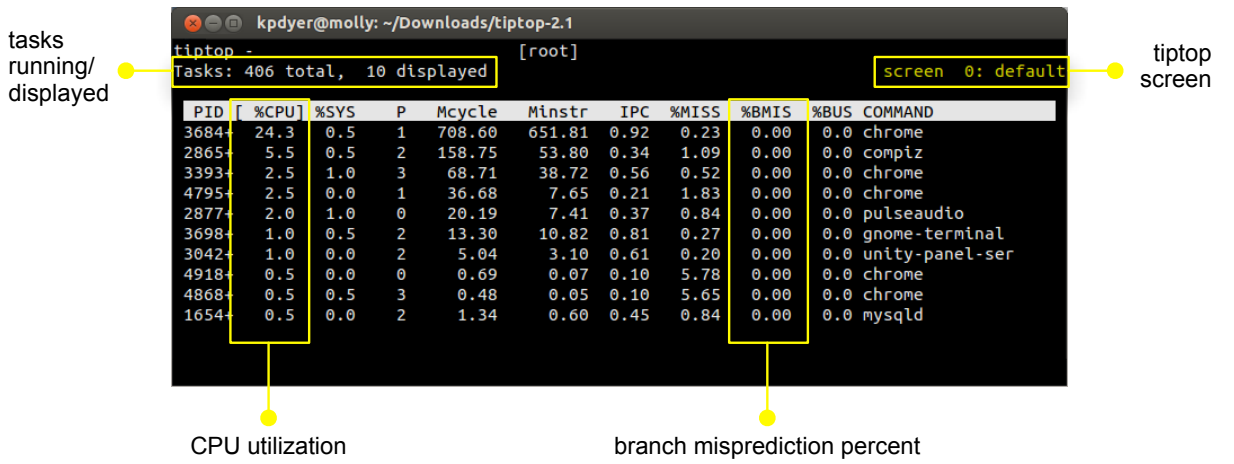

Figure 1: A screenshot of the default screen for tiptop version 2.1. Each row in the display represents a single process. In the top-left corner we have the number of running and displayed processes. By default tiptop only displays processes that have non-zero CPU utilization. In the bottom-left we highlight the CPU utilization, reported as percentage of cycles used since the last screen refresh. In the bottom-right corner we highlight the number of branch mispredictions. Finally, in the top-right we highlight the label for the current screen. Multiple views with custom columns can be configured and flipped through (using the right and left arrow keys) in real-time.

enhancements we identified for this project. We conclude in section 4 with a discussion about additional feature improvements for future versions of tiptop.

2 Methodology

In our development we started with tiptop version 2.1, which depends upon the libxml2 [?] and ncurses [?] libraries. Tiptop is written in C [?] and all improvements and bugfixes were implemented in C, too. As a key feature enhancement, we integrated the PAPI [?] library into tiptop. In our testing we used Python [?] and C to automate the process of verifying bug fixes and feature enhancements.

Fortunately, tiptop has an integral feature called *batch mode*, which enables tiptop to output statistic to stdout, rather than the default real-time display. The batch mode feature greatly simplified the testing process and enabled us to easily extract process statistics from tiptop.

Roughly, we can describe our methodology as a three-step process.

1. Identify feature or bug.
2. Develop code to enhance or fix tiptop.
3. Use Python (and sometimes C) to develop multiple unit tests that parse and verify the output of tiptop, and ensure that the feature works correctly or the bug is resolved.

In many cases we face the challenge of knowing what the "correct" statistics are for an application. As an example, we are not aware of a trivial way to know the correct value for the number of threads for a Chrome process. Indeed, in many cases this is non-deterministic. In these cases we

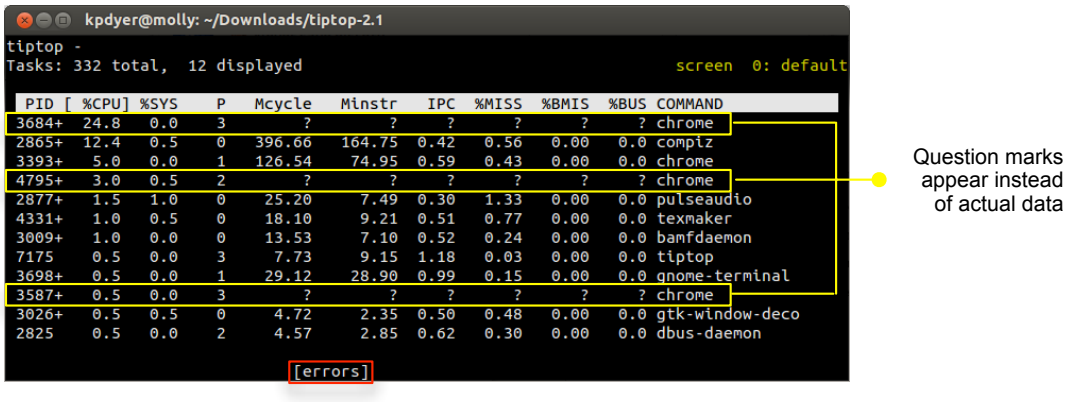

Figure 2: A screenshot of the manifestation of the two bugs identified in our development. When tiptop is unable to determine the values associated with a process it ouputs a question mark instead. As we can see, tiptop failed to gather statistics for three of the four chrome processes in the above screenshot, at the bottom of the screen, in the red box, tiptop reports that `[errors]` occurred.

constructed applications with known values, such as a simple C program that spawns N threads, and verified that tiptop reported the values correctly for our simple applications.

3 Results

In this section we start with an overview of the bugs we identified in tiptop. Then, we follow with a discussion of our feature enhancements.

3.1 Bug Fixes

In our initial attempt to understand tiptop and its range of features we encountered an anomaly. In some cases a question mark would appear, instead of a value from the hardware performance counter. This would happen consistently for programs such as the Chrome [**?**] web browser, and would happen sporadically for other applications. In Figure 2 we have a screenshot that highlights this bug.

Upon further investigation, we identified two separate issues that lead to this error condition.

Bug 1: Too many open file descriptors. First, a bit of background. Linux systems include a `limits.conf` file which enables a systems administrator to have fine-grained control over the system resources used by users and groups. As an example, it is possible to specify the maximum priority of a process run by a specific user or group, or even the maximum number of processes spawned by a specific user or group. Each resource has a *hard* and *soft* limit. Hard limits are enforced by the kernel and can only be modified by superusers. Soft limits are default values and can be manually raised per-user, up to the hard limit. For example, Ubuntu 12.04 has a soft limit of 1024 open file descriptors per user, and a hard limit of 4096. This means that a regular user can open up to 1024 file descriptors under normal circumstances, or can open up to 4096 file descriptors by explicitly requesting (to the kernel) for the soft limit to be increased to 4096.

In order to read hardware counters, tiptop opens five counters per process in its default configuration. For each combination of process and hardware counter, a file descriptor is opened. As

we can can see in the screenshot for Figure 2, my desktop had 332 running processes. Hence, we now have a problem. We have 332 processes, five hardware counters per process, so we have exceeded our soft limit for the number of file handlers open by requesting to open $332 \cdot 5 = 1660$ file descriptors.

As a stopgap solution to this problem, we explicitly request for our soft file descriptor limit to be increased to the hard limit. This may be performed by fetching the hard limit from `/proc/N/limits`, then making a call to the `setrlimit` function, which is provided by `sys/resource.h`. Unfortunately, increasing the open file descriptor limit beyond the hard limit requires superuser permission, which is undesirable. Our solution to this problem is consistent with other open source project that have encountered this issue, such as Wine[1]. A more robust solution for this bug is an open problem and would require fundamental changes to the design of tiptop, we talk about possible solutions in section 4.

The testing for this was not straightforward due to a different bug, which we will discuss next. In order to test this bug we ran tiptop as root, with root having a hard number of file descriptor set to 4096 and soft set to 1024. (By default, root has a soft limit of 4096 and hard limit of 8192 on Ubuntu 12.04.) We may then run the version of tiptop that uses the soft limit and we get the following output.

```
19884+   4.6   0.0    0        ?        ?     ?      ?      ?     ? chrome
...
19818+   0.5   0.5    0        ?        ?     ?      ?      ?     ? firefox
```

Simultaneously, we run the enhanced version of tiptop that explicitly requests a file descriptor limit to be raised to the hard limit, and we get the following output.

```
19884+   4.3   0.0    0 19327.35 19327.35  1.00   0.00   0.00   0.0 chrome
...
19818+   0.5   0.5    0  8589.93  8589.93  1.00   0.00   0.00   0.0 firefox
```

We verify by that this bug is solved by checking that no value in the tiptop output has a question mark. Indeed, this solution is not robust, because other bugs may result in an error, which results in a failure to output process statistics, as we will see next. In the final version of this project we hope to improve the robustness of our tests, such that error conditions are not conflated. We anticipate that this will require enhancements to tiptop's error reporting mechanism.

Bug 2: `EACCES`, permission denied. A second cause of "questions marks" in the tiptop output has been traced to an `EACCES` permission denied error returned by the kernel, and is isolated to only a few applications, such as `chrome` and the `gnome-keyring-daemon`, to name a few. This condition only occurs when running tiptop as a regular user. We contacted the lead developer of tiptop, Erven Rohou, in order to determine the cause of this problem. This was an unknown bug and a temporary workaround is running tiptop as root. We will have additional information about the cause this bug in the final version of our report.

3.2 Feature Enhancements

Next, we will discuss the feature enhancements for tiptop. First, we will discuss our enhancement to the default tiptop screen, which now includes the number of threads per process. Then, we will discuss our integration with the cross-platform Performance Application Programming Interface (PAPI) [**?**] library.

[1] `https://bugs.launchpad.net/ubuntu/+source/linux/+bug/663090`

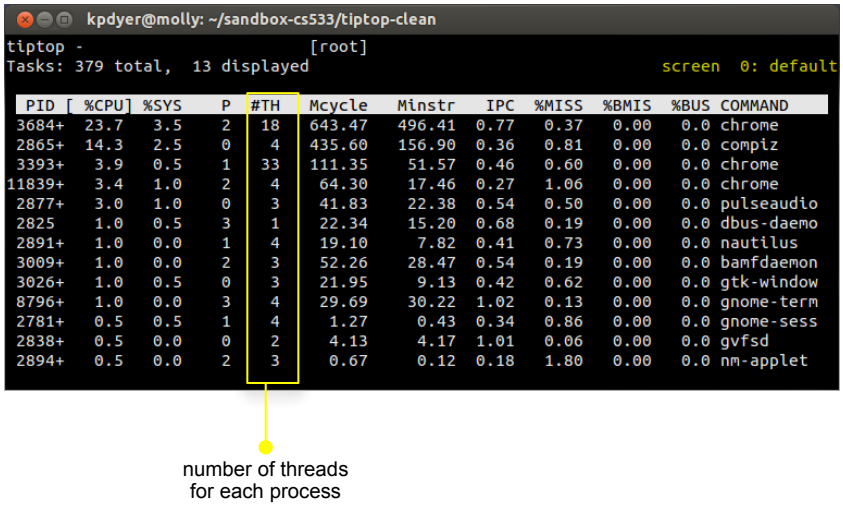

Figure 3: A screenshot of the enhanced tiptop homescreen, showing the number of threads in each process.

3.2.1 Thread count on main screen

The primary focus for tiptop is a window to hardware counters. However, there are other variables that can indirectly influence hardware performance. As an example, say a developer wants to investigate the effect of inter-thread cache contention on the performance of their application. Especially in cases when the number of threads in a process is not deterministic, it would be valuable to have the number of threads in a process on the tiptop screen. Therefore, we enabled tiptop to display the number of threads per process, and we included this feature on the default screen.

In our implementation we added hooks to the XML configuration logic to enable the ability to display the number of threads per process. This involved adding code at multiple layers, including a call that exposed thread count from the data-gather layer to the presentations layer. In addition, tiptop has multiple modes, which influence the filter that is applied to the processes/threads to be displayed. As an example, tiptop can be run with the `-H` in order to show per-thread information, rather than per-process. In such a case we do not want to display thread count, and our implementation includes logic such that thread count only appears in the case when we filter by processes.

In the testing of this feature we did the following, using Python and C.

1. Start a long-running C process that spawns N threads.
2. Start tiptop in batch mode.
3. Verify that tiptop displays N threads for the long-running C process.
4. Terminate long-running C process.

Indeed, our test for this case is not robust. We are not testing multiple thread libraries, and we have tested this on only two platforms. For the final version of this report we hope to include further details on strategies for increasing the robustness of these tests.

3.2.2 Integration with the PAPI library

The Performance Application Programming Interface (PAPI) library is a cross-platform API for accessing hardware performance counter information. Roughly, it is a interface that removes the complexity involved in accessing hardware performance counters. Let us consider a concrete example. The man page for tiptop gives the following example for a way to manually specify the counter for the number of issued mico-ops on Sandy Bridge:

```
<counter alias="uops_issued" config="0x010e"
         type="RAW" arch="x86" model="06_2A" />
```

which then requires an additional call to the counter alias

```
<column header=" U Ops" format="%5.1f"
        desc="U Ops Issues" expr="uops_issued" />
```

The values for `config` and `model` are hardware-specific, and requires that the user looks at documentation in order to implement and alias, which can then be used to construct a column in a custom tiptop screen. Indeed, it is less than ideal that a user has to specify magic numbers for their CPU. What is more, slight variations in the CPU being used is going to require a configuration file that is very system specific. This is certainly a nightmare when developing a tiptop screen that may be used across multiple platforms.

PAPI to the rescue. Fortunately, PAPI makes life much easier. Instead of having to specify a hardware-specific counter macro, then reference the macro for display, our new feature enables a user to specify the PAPI event directly. There are approximately one hundred different counter events specified in PAPI 5.0.1, and our goal is to enable the user to call as many of those as possible from the tiptop screen configuration file. As an example, the following declaration would be able to integrate directly into the tiptop XML file

```
<column header=" U Ops" format="%5.1f"
        desc="U Ops Issues" expr="PAPI_FP_OPS" />
```

It is then up to PAPI to determine the correct hooks for a hardware counter, and the burden is not on the user to identify magic numbers. This is cross-platform, as far as PAPI supports the specific counter specified. Hence, the user does not have to specify a custom hardware-specific `counter` macro, and a single XML file can be specified and deployed across multiple system. In addition, this is backwards compatible with previous versions of tiptop configurations files, and the prefix `PAPI_` for an `expr` attribute in a `column` tag, is a flag for tiptop to call PAPI, rather than using a direct kernel call for hardware counters.

This feature is under development and we will have further information about the breadth of coverage of PAPI calls, and more information about our testing strategy.

4 Future Work

Tiptop is a useful application, and our extensions further enhance its utility. However, there are two major issues that require further investigation.

First, our solution to the "Too many open file descriptors bug is unfortunately not comprehensive. On systems that have more than 819 processes, this will still appear as an issue. We do not believe that it is reasonable to require superuser permissions to solve this problem. Therefore, as

one solution, it may be possible to develop a heuristic algorithm that only opens a file descriptor for processes of interest. As an example the algorithm can open a file descriptor for any process that has used the CPU in the last N seconds. Of course, this will require process monitoring that does not use hardware performance counters. However, we believe that the overhead of such a strategy is justified, in order to minimize the probability of this bug occurring.

Second, we found that tiptop is not well-supported in virtualized environments. The man page breifly mentions this, but does not elaborate further. Our testing confirms that data is often omitted or not reported correct in VM instances. We believe this requires further investigation to understand why this is the case. Certainly, for testing purposes, the ability to test tiptop without requiring physical hardware would greatly increase the ability for verifying features implemented across multiple CPU architectures.