Using Hardware Performance Counters for Functional Correctness Debugging

Final Publishable Summary Report

Cemal Yilmaz
Faculty of Engineering and Natural Sciences
Sabanci University
34956 Istanbul, Turkey
cyilmaz@sabanciuniv.edu

Hardware performance counters are hardware-resident counters that record various events occurring on a processor. Todays general-purpose CPUs include a fair number of such counters, which are capable of recording events, such as the number of instructions executed, the number of branches taken, the number of cache hits and misses experienced, etc. To activate these counters, programs issue instructions indicating the type of events to be counted and the physical counters to be used. Once activated, hardware counters count the events of interest and store the counts in a set of special purpose registers. These registers can then be read and reset programmatically at runtime.

Hardware performance counters have been traditionally leveraged to perform low-level performance analysis and tuning of software systems. In this project, we, on the other, hand use them in a novel way, exploiting them as an abstraction mechanism for program executions.

Many data-driven program analysis approaches have been proposed in the literature. These approaches instrument the source code and/or binaries of programs, collect execution data from program executions every time the instrumentation code is exercised, and analyze the collected data, often referred to as *program spectrum*, to help shape future software development efforts.

Many types of data-driven approaches operate by comparing program executions. Program spectra are collected from a number of program executions, models that capture the patterns in these executions are inferred by using the program spectra collected, and similarities to these models and/or deviations from them are used to improve software quality.

A fundamental assumption of these and similar approaches is that there are identifiable and repeatable patterns in program executions and that similarities and deviations from these patterns can be used to perform many quality assurance tasks.

One common theme in program spectrum-based approaches is that, in order to compare program executions, these approaches need to abstract away many details. This is due to the fact that a program execution is a complex event, which can be considered to be a sequence of state transformations each of which comes to existence as a result of complex interactions between many factors. Therefore, it is genuinely hard to find the right level of abstraction for program executions.

In general, it is possible to collect detailed information at runtime to come up with better abstractions. However, the overhead cost both in terms of the time overhead required to collect the spectra and the space overhead required to store them often makes it impractical. Even if the runtime cost is manageable, it is often unclear what to collect and how to analyze such large amount of heterogenous information to identify meaningful patterns in program executions.

In this project we conjecture that the execution data collected from hardware performance counters (*hardware-based program spectra*) can be used to capture certain types of patterns in program executions in an unobtrusive manner. Note that since using hardware performance counters pushes substantial parts of the data collection task onto the hardware, it helps reduce the runtime overhead.

In this project, to test our conjecture, we developed a number of various quality assurance approaches using hardware-based program spectra and evaluated the proposed approaches by conducting large-scale experiments on open source widely-used subject applications. We furthermore compared the performance of hardware-based program spectra to that of traditional software instrumentation-based program spectra (*software-based program spectra*).

In [8], [9], [7], we developed an approach for fault detection to distinguish failed program executions from successful executions in an offline manner (i.e., after the executions have terminated). In [3], we developed an approach to cluster program failures such that failures that stem from the same or closely related causes are grouped together, facilitating program debugging. In [6], we developed an approach for fault localization to reduce the space of possible root causes for failures, which can in turn reduce the turn-around time for defect fixes. In [2], [4], we developed a lightweight runtime failure prediction approach to predict the manifestation of failures at runtime before they actually occur (i.e., on-the-fly while the program is running). In [5], [1], we developed an approach to identify likely causes of information leakage that can can be exploited in side channel attacks. In [10], we developed an approach to identify "spy" processes that attack cryptographic applications to reveal the secret keys processed by these applications.

In all these studies, one technical challenge we faced was that hardware performance counters are hardware-resident counters. Therefore, they do not distinguish between the instructions issued by different processes. In this project, we dealt with this by using virtual hardware performance counters that can track hardware events on a per-process basis. Virtual hardware performance counters are typically implemented by operating systems.

A related challenge was that hardware performance counters have limited visibility into the programs being executed, e.g., by themselves they do not know, for example, to which program function the current instruction belongs. Therefore, raw hardware performance counters-based spectra are generally too coarse to be useful. In this project, we dealt with this by associating counter values with code segments in programs. For example, in [8], [9], [7], we associated counter values with functions. In [2], [4], we not only associated counter values with functions, but also itemized the counter value associated with a function to reflect the number of events occurred in the body of the function as well as in each callee. In [1], we associated counter values with arbitrary code segments in the program. In [10], we associated counter values with time intervals during which the CPU is allocated to a particular process.

We also observed that hardware-based program spectra are not as flexible as software-based spectra. At the end, a hardware performance counter is a simple counter counting the number of low-level events occurring on the CPU. It is not possible to programmatically extend the capabilities of these counters or to programmatically develop new counters. Yet, they can count events that may not be counted with traditional software-based instrumentation, such as the number of misses/hits in data and instruction caches, and the number of pipeline stalls and flushes.

Furthermore, hardware performance counters provide an unobtrusive means (to the extend possible) of collecting information from inside program executions. In a study, we observed that it takes only 45 clock cycles on average to read the value of a virtual counter on our test platform [8]. Note that even though the cost of reading counters is low, the cost is paid each time the counters need to be read. Therefore, this fact should be taken into account when designing new hardware-based program spectra.

Due to these limitations, hardware-based program spectra may not be suitable for all types of quality assurance tasks.

However, despite these limitations, for the quality assurance approaches developed in the project and the hardware-based program spectra, software-based program spectra, subject applications, defects, and test cases used in the experiments, the results of our experiments strongly support our basic hypotheses:

1) There are identifiable and repeatable patterns in program executions,
2) Hardware-based program spectra can reliably capture these patterns at a lower runtime overhead, compared to traditional software-based program spectra,
3) Identifying similarities to these patterns and/or deviations from them can help quality assurance tasks improve software quality.

References

[1] A. C. Atici, C. Yilmaz, and E. Savas. Debugging for pinpointing information leakage. *In preparation for submission to a conference*, 2012.

[2] B. Ozcelik. Lightweight runtime failure prediction. *M.S. Thesis, Sabanci University*, 2012.

[3] B. Ozcelik, K. Kalkan, and C. Yilmaz. An approach for classifying program failures. In *Proceedings of International Conference on the Advances in System Testing and Validation Lifecycle*, VALID '10, pages 93–98, 2010.

[4] B. Ozcelik and C. Yilmaz. Lightweight runtime failure prediction. *In preparation for submission to an SCI-indexed journal*, 2012.

[5] E. Savas and C. Yilmaz. Cache attacks: An information and complexity theoretic approach. In *International Conference on New Technologies, Mobility and Security (NTMS'12)*, pages 1–7, 2012.

[6] C. Yilmaz. Using hardware performance counters for fault localization. In *Proceedings of International Conference on the Advances in System Testing and Validation Lifecycle*, VALID '10, pages 87–92, 2010.

[7] C. Yilmaz, E. Dumlu, and A. Porter. Program yurutmelerini siniflandirmak icin donanim ve yazilim olcum aygitlarini birlestirme. In *Proceedings of Yazilim Kalitesi ve Yazilim Gelistirme Araclari Sempozyumu (in Turkish)*, pages 279–286, 2010.

[8] C. Yilmaz and A. Porter. Combining hardware and software instrumentation to classify program executions. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, FSE '10, pages 67–76, New York, NY, USA, 2010.

[9] C. Yilmaz and A. Porter. Combining hardware and software instrumentation for fault detection. *Extended version of the original fault detection work, to be submitted to an SCI-indexed journal*, 2012.

[10] C. Yilmaz and E. Savas. Automatically identifying spy processes. *In preparation for submission to a conference*, 2012.