Binary Instrumentation for Rapid Creation of Productivity Tools

Author:
Brad Calder, Todd Austin, and Tim Cusac, BitRaker Inc.

Synopsis:
A wide range of ARM developers from architects, to compiler writers, to software developers, need tools to understand, analyze, and simulate program behavior. This enables developers to achieve high levels of system and program correctness, performance, and power efficiency. To be usable, these tools must be fast and customizable to the challenges at hand.

BitRaker Anvil™, the first binary instrumentation-based system for the ARM platform, is a tool-building framework that enables developers to rapidly construct tools to achieve these goals. BitRaker Anvil uses binary instrumentation to modify ARM binaries for the purpose of analyzing program behavior. BitRaker Anvil equips the developer with an API that enables the user to specify the particular program characteristics to analyze. Using this API, the developer can create custom tools to perform simulation or workload analysis several orders of magnitude faster than using a cycle-level simulator.

BitRaker Anvil uses symbol information to read an ARM binary into an intermediate representation (IR) and provides a programmable API to enable developers to create a newly instrumented binary. When creating a BitRaker Anvil tool, a developer creates both an instrumentation program using the BitRaker Anvil API and an analysis shared library. The instrumentation program uses the API to traverse the IR, adding calls (hooks) to the developer's analysis routines. The analysis routines are passed information about the dynamic behavior of the program, and they are used to simulate and gather profile information the developer is interested in. Together the instrumentation and analysis program constitute a standalone ARM Instrumentation Tool, which can be used repeatedly to analyze the behavior of ARM binaries.

The tool developer starts by creating an instrumentation program using the BitRaker Anvil API. The instrumentation program takes as input an original binary and outputs a newly instrumented ARM binary (Figure 1). This instrumented binary, when run, will call the routines in the analysis-shared library at the appropriate points to gather profile information or to perform simulation.

To illustrate the ease with which BitRaker Anvil can be used to analyze program behavior, we will briefly describe how to create an ARM Data Cache Simulator. An analysis-shared library is first created that contains a cache simulator routine that takes as input an effective address for the load or store memory access. An instrumentation tool is then created using the API to traverse all the static instructions in the binary, searching for load and store instructions. If a load or store is found, an instrumentation procedure call is added into the new binary pointing to the cache simulation routine in the analysis library.

When this instrumentation tool is run on the original ARM binary, the binary is read into the BitRaker Anvil IR. The IR is then traversed, adding instrumentation calls at every load and store operation to the analysis routines (e.g., cache simulator). When instrumentation is finished, the instrumentation tool writes out a newly instrumented ARM binary, which has been augmented with these instrumentation calls to the analysis code. When the newly instrumented binary is run, each load or store encountered invokes the cache simulator routine, passing its effective address as the input parameter. When the newly instrumented binary finishes execution, the analysis

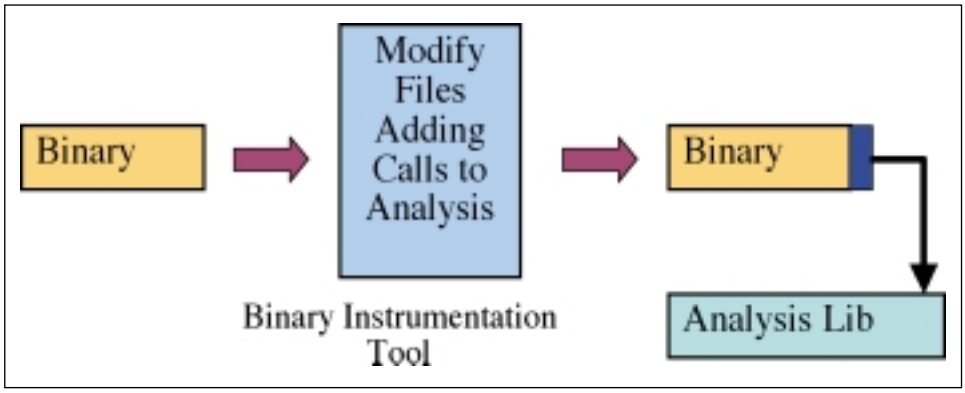

Figure 1: The Process of Binary Instrumentation

code reports to the user the final results of the profile in the form of the data cache miss rate.

Uses for BitRaker Anvil Binary Instrumentation

BitRaker Anvil's binary instrumentation provides an architect with efficient design space exploration and workload characterization tools to make the best trade-off decisions early in the design cycle. BitRaker Anvil enables faster design space exploration than existing technologies of the complete workload's execution, whereas existing technologies can only analyze a fraction of the workload. In addition, BitRaker Anvil enables an architect to rapidly build custom analysis tools that target any problem encountered in the design process to understand exactly what is going on during the workload's execution.

In order to produce a chip that performs as efficiently as possible (in terms of time, area, and power) an architect must completely understand the chip's target workload. Without a solid understanding of the target workload, the all-too-common result is a chip that looks good on paper but that fails to perform at the expected level when run with real applications. Performing a thorough characterization of the workload needs to be done early in the design cycle and BitRaker Anvil provides a fast and simple way to complete this characterization - even before a detailed architecture model is completed.

For example, an architect needs information such as the workload's instruction mix and common code sequences because it highlights the instructions on which to focus hardware optimization. Since embedded processors typically run a small handful of applications, knowing the exact needs of those applications can yield significant improvements by guiding the development of hardware accelerators. Another example of important workload analysis is monitoring the memory footprint in order to implement a high performance and adequately sized memory hierarchy in the processor. All of this critical information can be gathered quickly and thoroughly for the full workload's execution using BitRaker Anvil.

The BitRaker Anvil binary instrumentation framework enables software developers to rapidly build these profilers without being a hardware expert. Additionally, the tools themselves can quickly and accurately gather these feedback-directed profiles that reflect the whole program analysis advantage described later in this article.

The BitRaker framework lets you quickly create custom programmer productivity tools. These tools enable software developers to examine all aspects of their program's execution including code coverage, hot spots, memory usage, memory errors, power usage, and thread race conditions. An example tool is a hierarchical code profiler that reveals the application hot spots for the purpose of guiding code optimization efforts. Another example is a tool that efficiently uncovers memory leaks and memory access violations. BitRaker has built both of these tools, which we provide as stand-alone commercial products, using the BitRaker Anvil framework. Moreover, BitRaker Anvil enables you to map the generated information directly back to your source code, revealing exactly where important events occur.

Competitive Advantage Using BitRaker Anvil

Gathering the information using BitRaker Anvil's binary instrumentation techniques provides you with several advantages over simulation and source level instrumentation methods. The following text contrasts prevalent methods with BitRaker's new techniques.

Systems that rely on source code instrumentation for gathering profile information are also restricted to a specific language and require a special compiler to insert the profiling hooks. By performing the instrumentation and analysis at the binary level, BitRaker Anvil frees you from the constraints of using a particular compiler and source code language to perform the analyses you want.

Source code instrumentation systems require that you possess all of the source code for every element of the program you want to analyze. This is often not feasible when, for example, third party and operating system libraries are linked into the binary. Therefore, an important advantage of using BitRaker Anvil is that it makes the whole program available for analysis, which includes being able to instrument and analyze the impact of libraries and system routines.

If you have the time and know-how, several of the analyses described above can be gathered by modifying a cycle-level simulator. However, this type of analysis is usually not possible when using a third-party simulator for your design, since the source code is typically not provided. One of the major advantages of using BitRaker Anvil is that it provides an easy to use API for rapidly creating your own simulators, workload analysis tools, and programmer productivity tools. The API is extremely simple to program, and you can create a tool to analyze exactly what you want without being an ARM architecture or simulator expert.

BitRaker Anvil provides a final ARM executable where the application and the analysis routines run in the same address space. Therefore, the original binary can still run at native speed, switching to executing the analysis hooks as they are encountered. This results in significantly faster analysis for the above uses, in comparison to gathering this information with a cycle-level simulator.

This increased analysis speed is another capability that enables you to simulate and analyze the entire execution of an application. By contrast, the overhead of a cycle-level simulator demands so much time that it practically permits less than one percent of the program's execution to be analyzed. And this still takes more time than analyzing the complete execution of a program with BitRaker Anvil. This enabales workload analysis using BitRaker Anvil to be significantly more representative of the program's behavior because it sees all of the phases of the program's execution.

Summary

BitRaker Anvil binary instrumentation enables the rapid creation of high-speed profiling and simulation tools that are useful in all manners of hardware workload analysis and software development. BitRaker Anvil equips its users with the ability to perform static and dynamic analysis, and provides powerful building blocks that users can combine to build new tools limited only by their imagination. For more information please contact: www.BitRaker.com