Implementing iWatcher with Simics®

Claurissa Tuttle
August 2004

Abstract

Debugging large software packages is often time-consuming and expensive. Much of software debugging is done by inserting run-time checks into the software. These checks often result in an execution time increase of approximately 10-100 times. A recent solution to this problem, iWatcher, gives an efficient way to incorporate software debugging into the hardware. One way bugs are introduced is by having more than one variable or pointer point to the same spot. If two pointers point to the same data, the data can be changed through one pointer even if the data that the other pointer references should remain unchanged. iWatcher monitors memory locations rather than variables and pointers. This monitoring of memory locations allows for detecting access to a critical portion of memory regardless of which variable references the data or which pointer points to it.

iWatcher has been implemented in a simulator that is local to the University of Illinois at Urbana-Champaign. This research involves implementing iWatcher in a more widely used simulator, Virtutech® Simics®.

iWatcher

iWatcher is designed to watch a range of addresses, detect triggering accesses, and execute the appropriate monitoring function. Extra hardware is needed in order to incorporate iWatcher into the system. The CPU needs an extra register for the main check function. It also needs to include a range watch table for instances where large portions of memory need to be watched. The L1 and L2 caches need an extra 2 bits per word for the watchflag. There also needs to be a victim watchflag table. This is all described in *iWatcher: Efficient Architectural Support for Software Debugging* [1]. iWatcher is implemented partially in this extra hardware and also partially in software.

The first challenge for iWatcher is to be able to watch a range of addresses. The programmer indicates which addresses to watch by inserting an iWatcherOn() call. If the area to be watched is not too large, then iWatcher loads the watched memory region into the L2 cache if it's not already cached. iWatcher then sets the appropriate WatchFlags as are described in the paper [1].

The second challenge for iWatcher is to be able to detect a triggering access. In the iWatcherOn() call, a monitoring function should be specified. This function should be written by the user and should preform a check on the data that is being accessed. It is iWatcher's responsibility to detect accesses to memory, but the user's responsibility to design a check on the data to make sure it's what they want it to be. When using out-of-order processors, the monitoring function should only be triggered

when the instruction reaches the head of the reorder buffer. When there is a load or a store, the WatchFlags of the caches will be looked at. If appropriate, the monitoring function is triggered and executed.

Another challenge for iWatcher is to execute the monitoring function. When a monitoring function is triggered, the hardware spawns a new microthread to speculatively execute the remainder of the program after the triggering access, while the original microthread executes the monitoring function. If there is a violation in the sequential semantics of the code, then the speculative thread is squashed.

iWatcher was designed to provide a low overhead way to watch particular memory locations rather than addresses. It is designed to execute monitoring functions in parallel with the program in order to reduce overhead. It is flexible, extensible, cross-module, cross-developer, and is language independent. Finally, iWatcher is location-controlled, which helps to eliminate bugs that are introduced by pointers that could incorrectly change values of memory locations.

Simics®

Simics® is "the leading pure-software Instruction Set Architecture simulation tool for building full-scale virtual systems" [2]. It has the capability to simulate hardware that runs software as complex and important as operating systems. It provides the capability to "develop hardware-dependent software long before real hardware is available" [2]. Simics® offers the simulation strengths of flexibility, visibility, and control. It also offers a variety of processor models: in-order, fully-specified out-of-order, and parameterized out-of-order; and supports many architectures such as Intel, Sun SPARC, IBM PowerPC, MIPS, Alpha, and others.

This description of Simics® is great, but our primary reason for using Simics® is so iWatcher is more portable. iWatcher was first implemented on a simulator that is local to the University of Illinois Urbana-Champaign. Our purposes are to use Simics® because it is available to academia and others through the Virtutech® website http://www.virtutech.com. Our goal is to use Simics® to provide the architecture that iWatcher requires.

Combining iWatcher with Simics®

The first task in using Simics® to simulate the architecture for iWatcher is to decide what can be changed in the simulator. We want to be able to add to the cache and the microprocessor. Another step is to create a timing model and experiment with the delays in the cache. Other items to address are whether or not all memory accesses can be detected, and whether or not we can flush the Simics® pipeline. The processor also needs to be able to fetch the monitoring function.

These items were not addressed before the end of this research due to time restraints (10 weeks of research), my unfamiliarity to simulators in general, and some setbacks to the project. More details will be addressed in the "What I Learned" section.

Future Work

A continuation of this project involves becoming more familiar with Simics® in order to more fully understand the task that I set out to do. Future work can also address the concepts that I introduced in the previous section. Finally, implementing iWatcher in Simics® will be a good completion/continuation of this project.

What I Learned

This research project wasn't very productive as getting a lot of research done, but there are a few time sinks that I discovered that I'll hopefully be able to avoid in the future. First, Simics® is a simulator that was not in use by the school at the time I began my research. This shouldn't be much of a problem because Virtutech® will issue personal Simics® licenses for research purposes. The problem is that it was the fifth of ten weeks before I actually received my license. It then took me another week to get the Xserver running on my machine. I lacked the administrative privileges to correct the problem so it would work for me. I resorted to downloading copy of it into my directory.

After this slow start, I finally had the available source code and tutorials for Simics®. However, I was very unfamiliar with what a simulator was and by the time the ten weeks was up, I was only beginning to understand what Simics® was doing. I had been preparing presentations on Simics® all along, but the presentations did reflect my knowledge of it, which was limited. The papers that I found online and presented weren't a description of what I needed to know to understand simulators, but an overview of Simics®.

If I had it to do all over again. I would definitely do things differently. I would first of all, either pick a project that involved working in areas that I was more familiar with (I had never seen a simulator in my life), or pick a project that a co-worker or grad student would have time to help me with if I was stuck. I think I should have asked more questions, but I also felt like the grad students around were busy because there were a lot of paper deadlines during the summer. Thus, I think it would've been best if I had picked a project that I would feel comfortable working independently on. Hopefully from this experience, I'll be better prepared to avoid situations in the future that would result in getting basically nothing done.

References

[1] Pin Zhou, Feng Qin, Wei Liu, Yuanyuan Zhou, Josep Torrellas. "iWatcher: Efficient Architectural Support for Software Debugging". ISCA 2004.
[2] Virtutech. "Introduction to Simics®: Full System Simulator Without Equal". URL: http://www.virtutech.com/technology/whitepapers.html.
[3] Virtutech. "Simics® in Research." URL: http://www.virtutech.com/products/academicresearch.html.