Summary Part 2
iWatcher: Efficient Architectural Support for Software Debugging

Mrinal Nath (ID: 3307043)
April 13, 2005

Did this paper address an important issue? Explain.

Currently, there is very little architectural support for easy and efficient interactive debugging and/or dynamic execution monitoring of a program to ensure correctness. This is despite the fact that software errors cause a lot of economic damage. The existing tools are either limited by aliasing problems and other compile-time limitations (for static tools) or produce a very large amount overhead and drastically change the timing of the program (for dynamic tools). Also, current dynamic tools rely on compilers or pre-processing tools to insert the appropriate checks, and so they are also limited by the lack of perfect variable disambiguation during compilation/pre-processing. Actual hardware support for interactive debugging is limited to a very few ‘watchpoints’ and are inflexible and inefficient for dynamic execution monitoring.

This paper proposes a architectural technique (with some software support) to enable efficient and flexible dynamic execution monitoring. Thus, it addresses an important issue.

Are the proposed approaches valid? Describe its strength and weakness.

The proposed techniques require the addition of tags to cache lines (L1 and L2) and introduction of some additional hardware components like the RWT, main check function register, and VWT, etc. The software component of iWatcher includes the system calls and the check table. It is feasible to implement these things, so the proposed approach is valid.

Strengths

- iWatcher is ‘complete’, i.e. all accesses to watched locations are monitored
- iWatcher is flexible: it can be used to monitor individual words or a large block of memory
- iWatcher if efficient: the overhead produced is very small compared to other existing tools due to the hardware-based triggering of the monitoring function, and the use of TLS to speculatively execute the main program while the monitoring function is being executed non-speculatively
- It provides detailed information about the process state to the monitoring function
- iWatcher is convenient to use: programmers can manually insert iWatcher calls in the program, or it can be done by some instrumentation tool or compiler
- iWatcher is cross-module and cross-developer and programming language independent
- It detects many more bugs than other current tools

Weaknesses

- For best results, iWatcher calls should be inserted manually by the programmer, so that the limitations of a static tool do not apply. But this may not be a feasible task.
- Even the three reaction modes may be insufficient to detect some runtime problems. For e.g., roll-back may not be useful for detecting data races.
If the RWT is full, then additional large monitored regions are considered the same way as the small regions. This can cause many VWT overflows.

- Monitoring stores presents a problem, which requires the use of prefetch instructions to reduce the processor stalls.

**Do the results support the conclusions? Explain.**

The main conclusion is that iWatcher is a minimal-overhead technique for LCM and dynamic execution monitoring. Also, iWatcher is more capable, flexible and efficient than other existing tools. The results do support these conclusions: The overhead of iWatcher is orders of magnitude smaller than the overhead produced by other tools, and iWatcher detects many more bugs than other tools.

**Describe the potential future works?**

- How much is the overhead if the base architecture is a uniprocessor without any support for TLS?
- How can iWatcher be combined with other existing tools, to leverage the advantages of those tools? e.g. combining iWatcher with an invariant-inference tool like DIDUCE.
- How to automate the *generation* of the monitoring functions themselves? i.e., given a program, can at least some useful monitoring functions be automatically generated?