Summary
iWatcher: Efficient Architectural Support for Software Debugging

Venkatesan Packirisamy
April 13, 2005

1 To be completed before class

What are the problems solved by this paper? (50 words)
Software development costs are ever increasing. There has been a lot of tools to assist in debugging. Most of them incur a huge overhead. Also most of them are code based - the program is instrumented at several points to monitor the execution. The error could occur anywhere between the two points. This is not accurate. So there is a need for an efficient and accurate mechanism.

What are the approaches attempted by this paper? (50 words)
The paper proposes a hardware based monitoring mechanism. Here code segments are attached to specific memory locations and are executed when the location is accessed. Also the code segments are executed in parallel using Thread level speculation. Because of the TLS and the hardware based monitoring, iWatcher is very efficient. Using the mechanism iWatcher is able to track single locations or a range of locations.

What are the main conclusions of this paper? (50 words)
Since iWatcher is a memory based monitoring mechanism it can detect bugs more accurately. It is shown that it takes only 5-80% execution overhead to detect most of the bugs. Even if we monitor 20% of the loads, the overhead is going to be only 66 – 174% overhead. The iWatcher was able to detect most of the common bugs like uninitialized reads and data races.

2 To be completed after class

Did this paper address an important issue? Explain. (100 words)
Software bugs cost about 0.6% of the GDP in the US annually. The recent advances in computer architecture led only the improvement of applications. But there are other issues like development time, bugs, etc that affect the computer industry on the whole. So there must be a way to improve the software development process thus reducing the time to develop and the cost of errors. Many approaches are being used like static analysis, etc. Static analysis techniques are very costly and even the dynamic analysis techniques are not very accurate. Also all the methods are very slow making it almost unusable. So there must be an efficient support for debugging in the processor.

Are the proposed approaches valid? Describe its strength and weakness. (100 words) In this paper they propose the Location Controlled Monitoring. Here the user or the compiler will specify exact locations which are important. The hardware would monitor these locations. And if there is a triggering access, then the monitoring code will be executed. The hardware modifications are only to the cache lines and the Range Watch Table. To improve the performance of the monitoring function, the application can run after triggering the function in the speculative mode and the function would run parallelly. Strength: The location monitoring will improve the accuracy of debugging. And the monitoring function is very flexible and it can be used to monitor various bugs. Weakness: If we
include the thread forking and the thread commit delay, then the cost of gain we get from the separate thread is very less. Stack overflow problem need not have a monitoring function. It is just a check.

**Do the results support the conclusions? Explain. (100 words)**

The paper created several bugs and the iWatcher is found to have accurately found all the bugs. In some cases it has found the bugs which the Valgrind product couldn’t find. It shows that with iWatcher’s location monitoring, it can find the bugs much more accurately. Also it is being found that the multithreaded support is very helpful in many cases. In some cases there was about 30% reduction in overhead after using TLS. But the results show that only when the monitoring overhead is large, the TLS could be very helpful, otherwise the TLS in itself could be a overhead. On the whole the iWatcher used in the experiments only have 4 – 8% overhead.

**Describe the potential future works? (100 words)**

iWatcher offers a flexible framework which can monitor memory locations and by which we could detect any type of bugs. But the user or the compiler has to specify the locations to monitor. So if the user does not know where the bug is, then potentially the whole program could be monitored. Also the monitoring function is not automated in many cases. We could use a similar framework to do a more general monitoring and not just the location based monitoring. By this we could use it to detect requirement level bugs and not just the coding level bugs as done in the iWatcher.