Multi-core and VMM based Non-Interference Test Method of Embedded System Software

Zhang Jiong, Lv Zixu, Long Xiang, Bai Yuebin
（School of Computer Science and Engineering , Beihang University ）
Zhangjiong@buaa.edu.cn

Abstract: Traditional test methods of embedded system software impose on the object system and the result of the test are not accurate enough as expected. Non-Interference Test Method (NITM) can solve the problem. Not same with the traditional test methods, NITM is completely non-interference to the object software, does not add any instructions executed only within the test round into the object program. By analyzing the basic characteristics of NITM, an application model of NITM is set up. In many case of embedded system software test, the data need to process is very large amount so that we can not test all kinds of target real-timely. How can we achieve that in an embedded system since we have met such requirements in real engineering project? We brought a multi-core and Xen based NITM model to upgrade NITM so as to resolve the problem. [1]

Key words: Multi-core, Embedded System, Virtualization, VMM, Test

1 Introduction

Embedded computing technology is more and more popular in modern information application, the hardware and software involved in which is more and more sophisticated so that the requirement of reliability, robustness and performance is much more rigorous. How to evaluate and improve these features is very important research topic.

Especially, in many case of embedded system software test, the data need to process is very large amount so that we can not test all kinds of target real-timely. Here real-timely means we get test result like coverage as almost fast as the target running. For example, we can use a Pentium 4 to trace runtime detail of an 8086 processor as an advanced logic analyzer does since the later is rather slow, but we can not use a Pentium 4 to test a Pentium level processor real-timely. How can we achieve that in an embedded system since we have met such requirements in real engineering project?

2 Problems of embedded system test

Some important test results such as the coverage and performance analysis are mostly obtained by static analysis and/or stub-based dynamic analysis. Two primary kinds of test method are Pure Software Method (PSM) and Pure Hardware Method (PHM).

The PSM insert some stubs to the object software based on two mechanisms: stub functions and pre-process routines. These mechanisms often lead to expansion of the code size, and make the analysis result, especially the result about performance analysis, different with the reality. In fact, the interference introduced by the stubs can make the result fluctuated by 50% compared to the real result. The reason is simple, when a PSM tool is used to test the object software, the object software is not running itself only, so the gathered data is not precise enough for the sake of the stubs. In short words, PSM can’t make the test precisely.

PHM is often used to support the

[1] This paper is supported by the National High Technology Research and Development Program ("863"Program) of China (2007AA01Z118) and Aeronautics Research Funding(20081951033) and University Funding of Beijing.

hardware design and test in system level. Since the hits data are obtained directly upon the system bus, the inside cache of the processor is used to apply the instruction pre-fetch technology, and the coverage ratio analysis will not be accurate enough. For example, when an instruction is captured by the logic analyzer, it is not really executed at all. So, PHM can't precisely trace a system with cache applied while it is common in most modern processor, even in many processors used in embedded system.

3 Non-Interference Test Method

Traditional test method can not meet all the requirements of the embedded system software. For example, if the classic test method – stub [1] [2] [3] is used to get the coverage ratio data of an embedded program, there must be some test codes – the stubs – inserted into the program while they will be deleted when the test finished. In this scenario, the program is interfered. Though the interference ratio is not so large in most cases, e.g. 5%-15%[1], the test data should be tuned since the interference ratio can't be estimated precisely, or the analysis result will differ with the real state of the object system.

If a test method does not interfere the target software, i.e. does not add any executable code into the final version object code, then there is no need to make any revision to the test result. To make that happen, we have brought up the method called Non-Interference Test Method (NITM)[6], which has the features desired above. In a short term, NITM using special hardware to gather the run time system data while the target system is running, then the according software will using these data to trace the running states, even in a dynamical progress. Thus, the test progress is simplified and the test complexity is reduced, while the creditability level of the analysis is much higher.

The basic characteristics of NITM is dynamically gathering the system running state and make according analysis to meet different test object while not interfering the target system. By comparing the dynamical state data and the static features data of the object software, the real running progress of the target system will be traced precisely. So NITM is a combinational method of PSM and PHM. Also NITM is a white-box test method based on coverage analysis [2]. In [6], we used comprehensive host with hardware (Pentium 4 and FPGA) and software (windows based application and device driver) to test low level target processor like 8086，but failed to test faster processor because of the data need to process is too large amount, as well as the test result can not be generated within an acceptable short time.

4 Multi-core Model of NITM

To implement NITM in embedded system test, [6] points that the following points should be covered when set up the model of NITM:

(1)Comprehending the features and structures of the hardware of the target system so as to setup a proper test environment, in which it is easy to connect the target via physical interface and gather the runtime status data.

Figure 1 illustrate the model of NITM which mainly has three key parts, each of has some sub-functional module.

(1) Software Structure Analysis

Software structure analysis is very important. The software static structure include: instruction set of the object code and the corresponding address set; entry and return address of all the sub routines as well as the address range; all the branches; name space of all the global and local variables. Whether the data sets will be used depends on the requirements of the test.

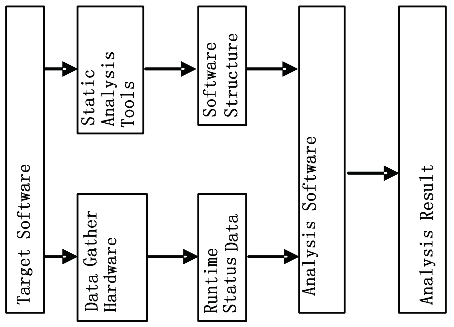

Figure 1 The Model of NITM

(2) Target System Status Data Gathering

Accurate data gathering is the precondition of all the analysis of NITM. If only all the needed history status data were recorded, all the analysis could be implemented. The following runtime status data should be concerned: executed instructions; operand; interrupt and the corresponding acknowledge; system bus status; time stamp.

What kind of data should be recorded depends on the particular requirements of each test, among them the time stamp is the most basic while most important data. In fact, all the performance and/or function analysis is based on the time information.

(3) Dynamical Analysis

The mainly function of dynamical analysis is to illustrate the runtime status data correctly with the help from both the target software structure produced by the static analysis tools and the test template based the target system. The analysis results include: each computing operation and the corresponding time; all kinds of statistic information; the trace of variables.

The implementation of such NITM model in [6] can't test a processor faster than 8086 running on 5MHz. To promote the ability of NITM, we now upgrade NITM model to Multi-core based NITM (MC-NITM) illustrated in figure 2. MC-NITM exploits multi-core to test single processor oriented software, including embedded software which we hope to test with most interest. We can use multiple cores to test one core based software, that means multi-core based test host may have sufficient computing ability even though the target core running with a very high frequency.

In [6], the test platform includes data gathering hardware and analysis software. The hardware part is connected to the target system directly but makes no any control or interference to the target system when gathering the status data. In MC-NITM, we use Core_{n-1}, which share L2 cache with Core_{n}, to achieve this goal while other cores are used to analyse static data and dynamic data need in NITM.

These data will be analyzed to produce the result like: coverage ratio analysis of each sub routine, module and the whole system; performance analysis of each sub routine, interrupt routine and even single statement; target system status history trace with the pattern < time, event >, here event could be interrupt requested, branches switched, variable accessed. when the case occurred to a 2GHz or higher frequency multi-core processor, the large amount L2 cache influence should be considered as [7][8][9][10]. Consider all kinds of the concerns, the best experiment target is Intel's new cosset Larrabee in the future, and we should consider how to test real time embedded system software as discussed in [11][12]. While now, we do the experiments on a PC with Quad-core Q9400 processor and 2G DDR RAM, and the target software is just a set of assembly program.

To build an acceptable embedded system software test environment, we use virtualization technology to setup a Xen based embedded system test environment as shown in figure2. Here we modified the famous open source virtualization software Xen to make a special VMM which running on a Intel quad-core processor machine but manage only one core for itself while left 3 cores to setup a MC-NITM based embedded system test environment: 1 core for target and 2 cores for host testing software. With this environment, we test some real embedded system application with acceptable result.

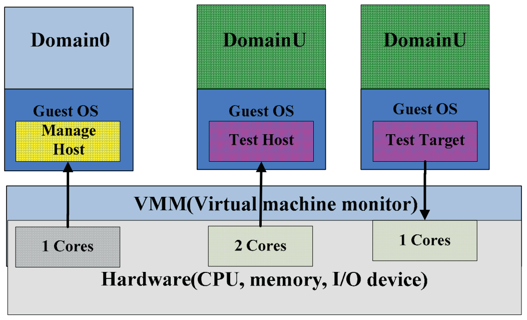

Figure2 Xen based MC-NITM

5 Chaos of Instruction Pre-fetch

An unavoidable problem of MC-NITM is that the target core has the mechanism of instruction pre-fetch, which is common among the modern processors. This feature of processor often leads to an embarrassed scenario: an instruction looks like it has been executed in the processor, but that only means that it has been loaded into the cache of the processor, and a wrong judgment in this kind of case could lead to severe failure of MC-NITM.

This problem can be solved based on the techniques named “Data Flow Based Cache Prediction”[4] and “predicting data cache behavior” [5]. For example, Intel 8086 processor has a 6-bytes-long instruction queue while most assembly instructions of 8086 are only 1 or 2 bytes long. That means there could be utmost 6 instructions loaded into the queue in sequence. Considering a common scenario, if there are any branches in the 6 instructions, some of them will be skipped, especially the latter several instructions [6]. Just simply comparing the status data gathered can’t indicate the real execute sequence. So, how to filter the missed instructions correctly is the key problem of MC-NITM.

To solve this problem, there are two methods: forward prediction and backward retrospect. Here are some definitions:

Definition 1 Invalid Pre-fetch Instruction (IPI): the instruction loaded but not executed.

Definition 2 Valid Pre-fetch Instruction (VPI): the instruction loaded any executed.

The behavior of a code section with IPI is different with the one without IPI. In fact, there are at least three types of differences: time, space and semantic. So there are three basic methods used to filter IPI according to each of them. The following table is a classic case in which the code segment is 8086 assembly program running on target core in real mode, the according compiler is MASM6.11.

Method (1) Considering the semantic details of this segment, instruction 2 will never be executed, so it is must be IPI. This assert is based on forward prediction, so the key point of this method is the static analysis about the source code of the target program. If instruction 1 is a condition branch, the semantic analysis will not provide any help. Obviously, this method can not solve this problem.

Line No.	Addr	Ins	Ins length bytes	Time Stamp	Cycles Per Ins	Bin Code
1	100	JMP 105	2	T1	2	EB 03
2	102	MOV AX，0	3	T2	4	B8 00 00
3	105	JMP 100	2	T3	2	EB F9

Table 1 Classic Embedded System Target Program

Method (2) As to the condition of space in this case, since the instruction queue is 6-bytes-long, the object codes from address 100 to 105 are all loaded. The address sequence should be 100, 102, 104 and 106. But the instruction 3 is reloaded after the instruction 1 and 2. The real address sequence is 100, 102, 104 and 105. Obviously, an exception occurs during the instruction loading process. By rewinding the trace data, it is easy to find this event. In this analysis process, not considering of the semantic details but an instruction data queue is mostly needed.

Method (3) As to the condition of time, it is also simple to detect some time duration exceptions. For example, suppose all the instructions are executed in their loaded order, i.e. all the instructions are VPI. Then the time between instruction 1 and 3 are 6 clks. While in fact, instruction 2 is not executed, so the real time difference is not 6 clks long (T3-T1<6). So, there must be something wrong in our supposition: instruction2 is not VPI.

In this case, all of these three methods work well in finding the IPI, so which is selected in a real test project depends on the computing cost of each method. As to this code segment, the cost of method (2) and (3) is much lower and the corresponding algorithms are rather simple. In the contrary, the cost of method (1) is much higher. After comparing many test result of large scale target software, there are very much difference among their costs of time and/or space when different method are selected. The essential difference among these methods could be summarized as: method (1) is based on forward prediction, all the data should be checked; method (2) and (3) are based on backward retrospect, only when an exception occurs there is the need to make further analysis. Table 2 shows the comparisons of these methods.

As to the complex case, each of these methods can’t work well and produce accurate result. Then there must be a comprehensive application of all of them. In real test project, all the analysis result of MC-NITM is gratifying.

Method	Speed	Memory Cost	Algorithm	Common Ground
1	Slow	High	Complex, Forward	Compare the static feature and the dynamic trace data to find exceptions (semantics, time and space).
2	Faster	Low	Simple, Backward	
3	Faster	Low	Simple, Backward	

Table 2 Differences of three methods.

6 Conclusions

MC-NITM is an upgrade of NITM[6], especially for embedded system, which can reflect the real running status of target system and can test a rather fast processor. Even though it really does make a few influences to the target software, the analysis result of test is much more accurate than traditional test method. Further more, NITM could be used to fit for all kinds of target systems with different architecture while this paper brought a potential implementation model based on x86 multi-core processor.

References:

[1] Jin Chao.CodeTest-The Use of Embedded Software Real-Time Test and Analysis Tools in the Development of Embedded System[A]. In: Shen Xubang,He Limin. 2001International Conference on Embedded Systems[C].Beijing: Beijing University of Aeronautics and Astronautics Press, 2001.290~292

[2] Wang Pu, Zhang Zhenjian, Wang Yuxi. A Research on Coverage-Based Software Testing Technique in the Real-Time Embedded Computer System[J]. Computer Engineering And Design, 1998,19（6）: 45~49

[3]Li Qian, Mei Lin, Ling Hui, et al. EASTT: An Embedded Application Software Test System[J]. Computer Engineering and Science, 2002.24（2）: 66~69

[4] Wolf F, Ernst R. Data Flow Based Cache Prediction Using Local Simulation[A]. In: IEEE International High-Level Validation and Test Workshop (HLDVT'00) (C). Berkeley, California :2000.155~160

[5] Ferdinand C , Wilhelm R. On predicting data cache behavior for real-time systems[A]. In proceeding of the ACM SIGPLAN Workshop LCTES'98 Montreal, Canada, June 1998,on Languages, Compilers and Tools for Embedded Systems, volume 1474 of Lecture Notes in Computer Science. 1998.16-30.

[6]Zhang J, Jin H H, Shang L H. Non-Interference Test Method of Embedded System Software. Journal of Beijing University of Aero. & Astro. 2004.7.

[7] Alexandra Fedorova，Operating System Scheduling for Chip Multithreaded Processors，Harvard University，Sept.2006

[8] C. Jung, D. Lim, J. Lee, and S. Han. Adaptive Execution Techniques for SMT Multiprocessor Architectures. In Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 236-246, 2005

[9] S. Kim, D. Chandra, and Y. Solihin. Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture. In Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 111-122, 2004

[10] D. Chandra, F. Guo, S. Kim, and Y. Solihin. Predicting Inter-Thread Cache Contention on a Multi-Processor Architecture. In Proceedings of the 12th International Symposium on High Performance Computer Architecture, pp. 340-351, 2005

[11]Brandenburg, B.B.; Calandrino, J.M.; Anderson, J.H., "On the Scalability of Real-Time Scheduling Algorithms on Multicore Platforms: A Case Study," Real-Time Systems Symposium, 2008 , vol., no., pp.157-169, Nov. 30 2008-Dec. 3 2008

[12]Jun Yan; Wei Zhang, "WCET Analysis for Multi-Core Processors with Shared L2 Instruction Caches," Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS '08. IEEE , vol., no., pp.80-89, 22-24 April 2008