Issues using the Nexus Interface for Measurement-Based WCET Analysis

Adam Betts and Guillem Bernat
Real-Time Systems Research Group
Department of Computer Science
University of York, UK
{abetts, bernat}@cs.york.ac.uk

Abstract

Hardware debug interfaces such as Nexus have the power to unleash the full potential of measurement-based WCET approaches due to the passive nature in which timing data are collected from the processor. However, difficulties arise as a result of their restrictive nature, thus disallowing true user freedom in the selection of instrumentation point placement. This paper elaborates on the problems encountered when using the Nexus interface in our measurement-based WCET framework, and how some of these issues can be resolved, particularly that of irreducibility.

1 Introduction

Measurement-based (MB) WCET analysis techniques are being embraced as the predictability of state-of-the-art processors diminishes due to modern speed-up features, e.g. cache, branch prediction, out-of-order execution, etc. Real-time hardware architects are increasingly looking towards such features as the thirst for performance enhancement grips the embedded market [5]. However, the resulting effect is instruction latencies that are difficult to model *statically*, thus resulting in pessimistic assumptions about speed-up features' behaviour that ultimately leads to loose WCET estimates. MB approaches, on the other hand, permit tighter WCET estimates [2] by testing the program on its actual hardware platform.

The main hindrance in using MB approaches is that timing data need to be collected whilst the program executes: either processor simulation or software probing performs the desired task. Cycle-accurate simulators capture both functional and temporal aspects of a processor, which are usually constructed by scrutinising the processor's user manual. Many factors render themselves crucial in the accurate design of a processor simulator for which failure can produce unsafe WCET estimates. Engblom [3] has cited potential sources of error in simulator construction and he concludes that user manuals are generally not trustworthy and that complex processors do not lend themselves to ease of construction. Alternatively, software monitoring inserts instrumentation points (ipoints) in the program in order to accumulate timing data during its execution. The clear advantage of this technique is program execution on its intended hardware, thus overcoming intrinsic difficulties in modelling the processor and peripheral hardware. However, a negative phenomenon widely known as the *probe effect* ensues whereby ipoints disturb the temporal nature of the program, i.e. the execution time of the program differs when the software ipoints are removed.

A solution for the seemingly unavoidable probe effect has arrived in the form of hardware debug interfaces, such as Nexus [6] which is discussed in section 3, and the ARM embedded trace macrocell (ETM) [4]. The fundamental aspect of these interfaces is that data are collected *passively* from the processor during program execution. Problems do arise through the restrictive nature that is imposed on data collection, which restricts true user freedom in the selection of instrumentation point placement, thus requiring new techniques to handle these hurdles. In this paper we discuss such problems and techniques in the context of Nexus, primarily because it is now an IEEE-ISTO standard, although they are equally applicable to the ETM.

In this paper we discuss how this restriction naturally leads to the problem of irreducibility in the data structure that is employed in our measurement-based framework, the instrumentation point graph (IPG), which is briefly introduced in the next section. We will show how some irreducible issues can be resolved by the relationship that exists between the program's control flow

graph (CFG) and the IPG. We further discuss how trace data loss and out-of-order execution affects the computation of WCET estimates. Finally, we outline some conclusions and future areas of work.

2 Instrumentation Point Graphs

Our approach combines timing data collected during measurement through high-level static techniques that reconstruct the longest path through the program, independent of the type of monitoring employed. This is accomplished using the IPG in which the atomic unit of computation is the time that is observed between ipoints, instead of basic blocks, which is the case of the CFG. In essence, the IPG arranges the possible transitions among ipoint pairs in the CFG into structural form. The transitions among ipoints in the IPG thus represent sequences of code exercised when a particular edge is followed. Timing measurements for these sequences are obtained by executing the program using test-data generation algorithms.

Once the IPG has been built, we adopt traditional calculation methods found in the literature, i.e. tree-based, path-based and IPET, revising them accordingly so that they pertain to the IPG. As hierarchical representations have difficulty modelling *irreducible* regions of code, the reducibility property of an IPG is a central issue in tree-based methods. Software instrumentation can guarantee the reducibility of the resultant IPG, details of which can be found elsewhere [1]; however, as true user freedom in ipoint placement is disallowed in Nexus-monitored programs, this property is only in evidence in trivial CFGs. An example later in the paper will help clarify this issue.

3 The Nexus Standard

Nexus has been introduced in response to the ever-increasing subtle nature of software and hardware bugs [6, 7]. This subtlety has arisen as a result of the complexity of modern processors: more transistors, faster clock-rates, multiple-level on-chip caches, multi-core processors, etc. All these intricacies mask the visibility of bugs and render traditional methods of debugging, e.g. in-circuit emulators and logic analysers, inappropriate. Nexus uses JTAG ports to communicate between a debug tool and the processor, and is increasingly being supported by chip manufacturers such as Motorola and STMicroelectronics.

The principal way to extract WCET data through the Nexus interface is by utilising its program trace feature, *branch trace messaging* (BTM). This allows time stamps to be recorded when sequential program flow *discontinues*, i.e. at taken branches and exceptions. In the case of taken *direct* branches, Nexus includes the number of sequential instructions that were executed since the last taken branch or exception, including those direct branches that resolved to untaken and indirect branches. The address of the branch target and branch-condition predicate bits can also be derived by using the more refined *historical* BTM feature [7]. At first sight it would appear that only monitoring taken branches in unnecessarily restrictive. The motivating factor for this is to lower the burden placed on the Nexus interface: too many requested debug reports results in trace buffer overflow. The BTM feature is the principal means of extracting WCET data, so we now qualify how its restrictive properties impacts the computation of MB WCET estimates.

Irreducible IPGs

In optimising compilers and WCET analysis, the reducibility of a CFG is a key property as loop identification techniques, for example, are greatly simplified. A loop is irreducible when there are multiple entries into the loop [9], so that no single node dominates all nodes in the loop body. In these cases it is often difficult to compute the nodes that are contained in the body, the nesting relationship among loops, and even the number of loops. However, using the Nexus interface in our framework ensures a much higher prevalence rate of irreducibility in the IPG, even for relatively simple CFGs. Worse yet, the irreducibility properties are vastly more complex than an optimising compiler would typically introduce on a CFG, in the sense that reducibility encapsulates much larger subgraphs of the IPG.

The cause of irreducibility emanates from the *virtual* nature of Nexus ipoints in contrast to the *physical* nature of software probes. In the latter, all executions of a program that invoke a set of basic blocks include the execution of ipoints belonging to those basic blocks. This is not the case in the BTM, in which only the invocation of a transition among basic blocks triggers the time stamp. Therefore, unless an edge e that includes a virtual ipoint dominates all other edges E including virtual ipoints on each path to exit in the CFG, it is possible that the flow of control skips around e to each edge in E. This point is illustrated in figure 1: there is a CFG with virtual ipoints on all transi-

tions for which Nexus would record a time stamp[1], as well as the resulting IPG that consists of ipoints as nodes. The edge that includes ipoint i_1 clearly dominates edges that include ipoints i_2 and i_3 in the CFG, thus ensuring that all paths from *start* pass through this edge to reach i_2 and i_3. On the other hand, there is no dominance relation among i_5 and i_6 and they both reside in a natural loop structure. Therefore, the set of ipoints on paths from *start* to this loop have a corresponding edge to i_5 and i_6 if *start* is their immediate dominator; the complexity evidently intensifies as the size of this set increases as well as the number of ipoints in the loop body. Another complexity that emerges from figure 1 is that of misidentification of loop nesting structures. The natural loop structure containing i_5 and i_6 might incorrectly be identified as a nested loop whereby i_5 is the outer loop header and i_6 is the inner loop header. In fact, all edges $i_5 \rightarrow i_6$, $i_6 \rightarrow i_5$ and $i_6 \rightarrow i_6$ are loop backedges since they correspond to another iteration of the loop in the CFG. This problem is also in evidence in the *while* loop structure that contains i_2 and i_3.

We utilise two interlinked techniques in handling IPG irreducibility that is generated by virtual ipoint placement. Firstly, by using the relationship between the CFG and the IPG it is relatively straightforward to ascertain when simulated nested loops actually conform to a single loop. This is primarily carried out by inspection of the dominance relation that exists among basic blocks in the CFG with respect to that of the ipoints in the IPG. Secondly, we can reduce the intricacy of IPG irreducibility, and complexity in general, by means of *edge pruning*. The relationship between the CFG and the IPG is yet again central to the accomplishment of this task, as well as the familiar WCET principle that a program's WCET occurs under maximum loop iteration. In general, any edge in the IPG that bypasses the execution of a loop in the CFG can be pruned. Although both of these techniques reduce some irreducibility aspects, the general case remains an open problem.

Trace data loss

As we discussed above, Nexus attempts to prevent trace data loss through its BTM scheme by recording data when program flow discontinues. This does not completely guarantee fulfilment of this requirement since tightly grouped sequences of control flow changes might still overwhelm the JTAG port. The problem is accentuated by the pulsating increase in microprocessor performance that results in higher instruction throughput. Multiple-issue processors, more accurate branch prediction schemes, and more elaborate speculative execution techniques ensure a faster turnover of branch instructions. In sharp contrast to this, nobody is suggesting that the average size of basic blocks, typically six instructions, is on the verge of increasing.

[1]We include start and exit nodes as ipoints

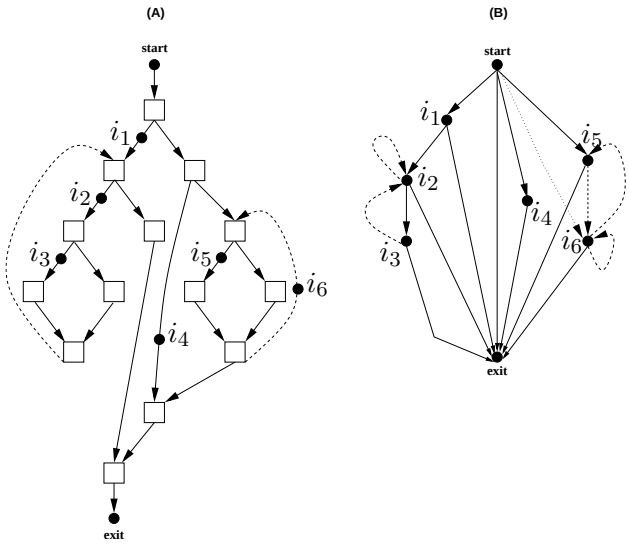

Figure 1: (A) CFG with virtual ipoints on edges Nexus monitors and (B) Resultant IPG generated

The loss of trace data has varying consequences on the computation of WCET estimates. Incomplete information about the branch target address will evidently result in difficulties in reconstructing the path that was executed in the program. Consequently, the time that is observed between recorded pairs of ipoints can lead to inaccurate WCET estimates as it will include the execution of a path that includes another unrecorded ipoint. If measurement is being used to ascertain loop bounds then data loss inevitably leads to the possibility of underestimation since fewer iterations will be observed than the actual number. The irreducibility problem of IPGs further complicates trace data loss: others [10] have shown that it is possible to reconstruct the path through a program by instrumenting the leaves of the CFG's dominator tree. However, the dominator tree of the IPG is extremely shallow as a result of irreducibility, thus eliminating this possibility.

The key to reducing trace data loss is to minimise the data rate out of the CPU. In particular, Nexus does permit the monitoring of a set of addresses within a specified range. Hence it is possible to guide the instrumentation process given static and dynamic properties of the program. For example, some studies [8] have shown that only a

small number of branches are dynamically invoked, thus it may be more pertinent to observe these locations. Other properties such as whether a branch appears on the worst path, the size of the program etc., have equal bearing.

Out-of-order execution

Of all the contemporary hardware features, out-of-order execution causes the greatest distress within the field of WCET due to the sheer complexity required in the analysis - it has also inhibited MB analysis [2]. The crux of the problem is that out-of-order execution permits instructions to execute in a different order to that described in the program, thus it is difficult to determine the correspondence between timing data and the instructions it encompasses. This obstacle is independent of the basic unit of computation employed, so it equally applies to the IPG. Moreover, since Nexus targets modern microprocessors there is a greater likelihood that out-of-order execution problems arise as this technique creeps into the embedded market.

Finding solutions for the out-of-order problem that do not result in undue pessimism is a difficult task. However, as we indicated above, Nexus does provide additional information that might disambiguate some of these issues. Knowledge such as the number of sequential instructions executed since the last program flow discontinuity could provide valuable insight.

4 Conclusions and Future work

Hardware debug interfaces such as Nexus appeal greatly to MB WCET analysis techniques due to the passive collection of timing data from the processor, thus eliminating the probe effect. However, new problems surface as a result of the restrictions imposed by these interfaces on the placement of virtual instrumentation points.

In particular, we have demonstrated how irreducibility quickly becomes problematic even for relatively small CFGs, and that irreducibility is decidedly more complex. On the other hand, we have highlighted how the relationship between the CFG and our underlying data structure, the instrumentation point graph, can be exploited to overcome some of these issues. The focus of future work is to formalise these techniques and to extend them in order to handle a larger subset of irreducible graphs. We also showed how trace data loss and out-of-order execution negatively impacts the computation of WCET estimates. Future work in this area will quantify these effect and propose some solutions.

References

[1] A. Betts and G. Bernat, "Instrumentation Point Graphs for WCET Analysis", Technical report, Department of Computer Science, University of York, April 2005.

[2] A. Colin and S. Petters, "Experimental Evaluation of Code Properties for WCET Analysis", *In proceedings of the* 24^{th} *Real-Time Systems Symposium (RTSS'03)*, December 2003.

[3] J. Engblom, "Processor Pipelines and Static Worst-Case Execution Time Analysis", PhD Thesis, Uppsala University, Uppsala, Sweden, April 2002.

[4] ARM development tools. At `http://www.arm.com`.

[5] G. Frantz, "Digital Signal Processor Trends", IEEE Micro, Vol. 20, No. 6, pages 52-59, November 2000.

[6] The Nexus 5001 forum. At `http://www.nexus5001.org`.

[7] J. Turley, "Nexus Standard Brings Order to Microprocessor Debugging", August 2004. At `www.nexus5001.org/nexus-wp-200408.pdf`, March 2005.

[8] S. Sechrest, C-C. Lee and T. Mudge, "Correlation and Aliasing in Dynamic Branch Predictors", *In Proceedings of the* 23^{rd} *annual international symposium on Computer architecture*, May 1996.

[9] V. Sreedhar, G. Gao, and Y. Lee, "Identifying Loops using DJ graphs", *In ACM transactions on Programming Languages and Systems*, 18(6):649-658, November 1996.

[10] M.M. Tikir and J.K. Hollingsworth, "Efficient Instrumentation for Code Coverage Testing", *In proceedings of the international symposium on Software testing and analysis*, July 2002.