Verifying Timing Behavior by Abstract Interpretation of Executable Code

Christian Ferdinand and Reinhold Heckmann

AbsInt Angewandte Informatik GmbH
Stuhlsatzenhausweg 69, D-66123 Saarbrücken, Germany
info@absint.com, http://www.absint.com

Abstract. Many tasks in safety-critical embedded systems have hard real-time characteristics. **AbsInt**'s worst-case execution time analyzer **aiT** can estimate precise and safe upper bounds for the WCETs of program tasks, thus providing the basic input for verifying the real-time behavior of embedded applications.

1 Introduction

Failure of a safety-critical embedded system may result in the loss of life or in large damages. Utmost carefulness and state-of-the-art machinery have to be applied to make sure that such a system is working properly. To do so lies in the responsibility of the designer(s). The proper working of an embedded system includes faultless working of the underlying hardware and software ensuring the production of correct output at appropriate times. Failure to meet deadlines may be as unacceptable as producing wrong output. A tool such as **AbsInt**'s **aiT** can efficiently determine upper bounds for the Worst-Case Execution Time (WCET) of code snippets given as routines in executables. The predicted WCETs can be used to determine an appropriate scheduling scheme for the tasks and to perform an overall schedulability analysis in order to guarantee that all timing constraints will be met [1].

The determination of the WCET of a task is a difficult problem because of the characteristics of modern software and hardware. Caches, branch target buffers, and pipelines are used in virtually all performance-oriented processors. Consequently the timing of the instructions depends on the execution history. Hence, the widely used classical methods of predicting execution times are not generally applicable. Software monitoring and dual-loop benchmark change the code, what in turn changes the cache behavior. Hardware simulation, emulation, or direct measurement with logic analyzers can only determine the execution time for some fixed inputs.

In contrast, abstract interpretation can be used to efficiently compute a safe approximation for all possible cache and pipeline states that can occur at a program point in any program run with any input. These results can be combined with ILP (Integer Linear Programming) techniques to safely predict the worst-case execution time and a corresponding worst-case execution path.

2 Worst-Case Execution Time Prediction by aiT

AbsInt's **aiT** WCET analyzer tools get as input an executable, user annotations, a description of the (external) memories and buses (i.e. a list of memory areas with minimal and maximal access times), and a task (identified by a start address). A task denotes a sequentially executed piece of code (no threads, no parallelism, and no waiting for external events). This should not be confused with a task in an operating system that might include code for synchronization or communication. Effects of interrupts, IO and timer (co-)processors are not reflected in the predicted runtime and have to be considered separately (e.g., by a quantitative analysis).

aiT operates in several phases. First a *decoder* reads the executable, identifies the instructions and their operands, and reconstructs the control flow [2]. The reconstructed control flow is annotated with the information needed by subsequent analyses and then translated into CRL (Control-Flow Representation Language). The annotated control-flow graph serves as the input for all further analyses.

The decoder can find the target addresses of absolute and `pc`-relative calls and branches, but may have difficulties with target addresses computed from register contents. Thus, **aiT** uses specialized decoders that are adapted to certain code generators and/or compilers. They usually can recognize branches to a previously stored return address, and know the typical compiler-generated patterns of branches via switch tables. Yet non-trivial applications may still contain some computed calls and branches (in handwritten assembly code) that cannot be resolved by the decoder and require user annotations. Such annotations may list the possible targets of computed calls and branches, or tell the decoder about the address and format of an array of function pointers or a switch table used in the computed call or branch.

Value analysis tries to determine the values in the processor registers for every program point and execution context. Often it cannot determine these values exactly, but only finds safe lower and upper bounds, i.e. intervals that are guaranteed to contain the exact values. The results of value analysis are used to determine possible addresses of indirect memory accesses—important for cache analysis—and in loop bound analysis.

WCET analysis requires that upper bounds for the iteration numbers of all loops be known. **aiT** tries to determine the number of loop iterations by *loop bound analysis*, but succeeds in doing so for simple loops only. Bounds for the iteration numbers of the remaining loops must be provided as user annotations. Loop bound analysis relies on a combination of value analysis and pattern matching, which looks for typical loop patterns. In general, these loop patterns depend on the code generator and/or compiler used to generate the code that is being analyzed. There are special **aiT** versions adapted to various generators and compilers.

Cache analysis classifies the accesses to main memory. The analysis in our tool is based upon [3], which handles analysis of caches with LRU (Least Recently Used) replacement strategy. However, it had to be modified to reflect the non-

LRU replacement strategies of common microprocessors: the pseudo-round-robin replacement policy of the ColdFire MCF 5307, and the PLRU (Pseudo-LRU) strategy of the PowerPC MPC 750 and 755. The modified algorithms distinguish between sure cache hits and unclassified accesses. The deviation from perfect LRU is the reason for the reduced predictability of the cache contents in case of ColdFire 5307 and PowerPC 750/755 compared to processors with perfect LRU caches [4].

Pipeline analysis models the pipeline behavior to determine execution times for a sequential flow (basic block) of instructions. It takes into account the current pipeline state(s), in particular resource occupancies, contents of prefetch queues, grouping of instructions, and classification of memory references as cache hits or misses. The result is an execution time for each instruction in each distinguished execution context.

Using the results of the micro-architecture analyses, *path analysis* determines a safe estimate of the WCET. While the analyses described so far are based on abstract interpretation, integer linear programming is used for path analysis. The program's control flow is modeled by an integer linear program [5] so that the solution to the objective function is the predicted worst-case execution time for the input program.

Detailed information about the WCET, the WCET path, and the possible cache and pipeline states at any program point are visualized in the aiSee tool [6].

3 Dependence on Target Architectures

There are **aiT** versions for PowerPC MPC 555, 565, and 755, ColdFire 5307, ARM7 TDMI, HCS12/STAR12, TMS320C33, C166/ST10, Renesas M32C/85 (prototype), and Tricore 1.3 (under construction).

Decoders are automatically generated from processor specifications defining instruction formats and operand meaning. The CRL format used for describing control-flow graphs is machine-independent. *Value Analysis* must interpret the operations of the target processor. Hence, there is a separate value analyzer for each target, but features shared by many processors (e.g., branches based on condition bits) allowed for considerable code sharing among the various value analyzers.

There is only one cache analyzer with a fixed interface to pipeline analysis. It is parameterized on cache size, line size, associativity, and replacement strategy. Each replacement strategy supported by **aiT** is implemented by a table for line age updates that is interpreted by the cache analyzer.

The pipeline analyzers are the most diverse part of **aiT**. The supported target architectures are grouped according to the complexity of the processor pipeline. For each group a common conceptual and coding framework for pipeline analysis has been established, in which the actual target-dependent analysis must be filled in by manual coding.

4 Precision of aiT

Since the real WCET is not known for typical real-life applications, statements about the precision of **aiT** are hard to obtain. For an automotive application running on MPC 555, one of **AbsInt**'s customers has observed an overestimation of 5–10% when comparing **aiT**'s results and the highest execution times observed in a series of measurements (which may have missed the real WCET). For an avionics application running on MPC 755, Airbus has noted that **aiT**'s WCET for a task typically is about 25% higher than some measured execution times for the same task, the real but non-calculable WCET being in between. Measurements at **AbsInt** have indicated overestimations ranging from 0% (cycle-exact prediction) till 10% for a set of small programs running on M32C, TMS320C33, and C166/ST10.

5 Conclusion

aiT is a WCET tool for industrial usage. Information required for WCET estimation such as computed branch targets and loop bounds is determined by static analysis. For situations where **aiT**'s analysis methods do not succeed, a convenient specification and annotation language was developed in close cooperation with **AbsInt**'s customers. Annotations for library functions (RT, communication) and RTOS functions can be provided in separate files by the respective developers (on source level or separately).

aiT enables development of complex hard real-time systems on state-of-the-art hardware, increases safety, and saves development time. Precise timing predictions enable the most cost-efficient hardware to be chosen. As recent trends, e.g., in automotive industries (X-by-wire, time-triggered protocols) require knowledge on the WCETs of tasks, a tool like **aiT** is of high importance.

References

1. Stankovic, J.A.: Real-Time and Embedded Systems. ACM 50th Anniversary Report on Real-Time Computing Research. (1996) http://www-ccs.cs.umass.edu/sdcr/rt.ps.
2. Theiling, H.: Extracting safe and precise control flow from binaries. In: Proceedings of the 7th Conference on Real-Time Computing Systems and Applications, Cheju Island, South Korea (2000)
3. Ferdinand, C.: Cache Behavior Prediction for Real-Time Systems. PhD thesis, Saarland University (1997)
4. Heckmann, R., Langenbach, M., Thesing, S., Wilhelm, R.: The influence of processor architecture on the design and the results of WCET tools. Proceedings of the IEEE **91** (2003) 1038–1054 Special Issue on Real-Time Systems.
5. Theiling, H., Ferdinand, C.: Combining abstract interpretation and ILP for microarchitecture modelling and program path analysis. In: Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain (1998) 144–153
6. AbsInt Angewandte Informatik GmbH: aiSee Home Page. (http://www.aisee.com)