Computing the Worst Case Execution Time of an Avionics Program by Abstract Interpretation

Jean Souyris* (jean.souyris@airbus.com),
Erwan Le Pavec* (erwan.lepavec@airbus.com),
Guillaume Himbert* (guillaume.himbert@airbus.com),
Victor Jégu* (victor.jegu@airbus.com),
Guillaume Borios** (guillaume.borios@airbus.com)

*Airbus France, **Atos Origin Integration,

Reinhold Heckmann*** (reinhold.heckmann@absint.com)

*** AbsInt GmbH

Abstract

This paper presents how the timing analyser aiT is used for computing the Worst-Case Execution Time (WCET) of two safety-critical avionics programs. The aiT tool has been developed by AbsInt GmbH as a static analyser based on Abstract Interpretation

1 Introduction

In the field of safety-critical avionics applications, verifying that a program exhibits the required functional behavior is not enough. Indeed, it must also be checked that its timing constraints are satisfied. Generally these constraints are expressed in terms of the **Worst Case Execution Times** (WCETs) of program tasks.

The WCET of a program (or piece of a program, like a routine, a task, etc) is the maximum of the execution times of all program runs. Unfortunately, finding the program run that leads to the WCET is impossible for real-life programs. What is achievable is computing an upper bound of the WCET, i.e., a time greater than the real but incomputable WCET. In that case, the WCET bound is said to be sound, and the less it is away from the real WCET – from above – the better it is. A good method for estimating the WCET of a program must be sound and precise, i.e. yield tight results, with the ability to demonstrate both properties.

For the most safety-critical avionics programs, a pure measurement-based method is not acceptable, especially if the dynamic structure of the program was not made as deterministic as possible, by design.

This paper will show how it became almost impossible to compute the WCET of two safety-critical avionics programs by Airbus' traditional method, due to the complexity of the modern processor they execute on (sections 2 and 3). Section 4 presents AbsInt's aiT for PowerPC MPC755, a tool that computes the WCET of a program by analyzing its binary file. Section 5 describes how aiT is currently used for computing the WCET of the two avionics programs we take as examples along the paper. Finally, section 6 mentions future work and concludes.Instructions

2 A challenging hardware

Sometime ago, processors behaved in a very deterministic way. The latency of an instruction was a constant, i.e., it did not depend on what happened before the execution of that instruction. This was the case for internal instructions (add, mul, or, etc.) as well as for those that access external devices like memory or IO. In order to increase their average computation power, modern processors are endowed with accelerating mechanisms causing variable execution times of instructions. Hence, the duration of an instruction depends on what was executed before it. This "effect of history" can be very deep and without logical correlation to the instruction it affects. One example of such a mechanism is the cache. Indeed, depending on the execution path leading to, say, a load instruction, the memory line containing the data to be loaded may already be in the data cache (HIT), or not, be it that it was not yet loaded (MISS) or already removed (MISS due to replacement). There are many other accelerating mechanisms like out-of-order execution, branch prediction, speculative accesses, "superscalarity", duplication of processing units (e.g., two Integer Units), Store Buffers, pipelining of addresses, etc.

The board on which the analysed programs execute is made of a PowerPC MPC755, SDRAM, and IO peripherals connected to a PCI bus. The MPC755 is a "modern" processor in the sense that it employs the kind of accelerating mechanisms mentioned above. SDRAM is also modern because of the sort of cache associated to each bank of memory: The SDRAM page containing the most recently accessed memory address is kept "open", allowing for faster access in the future. The chipset making these components "talk" to each other is modern as well, since, for instance, it optimizes accesses to SDRAM by buffering certain kinds of writes or by taking profit of the address pipelining mechanism of the MPC755.

As if this was not enough, the refresh of the SDRAM and the so-called Host Controller (connected to the PCI bus) steal cycles from the processor asynchronously.

3 A traditional method very hard to apply now

Before the new method based on aiT for PowerPC MPC755 was introduced, Airbus used some "traditional" method for computing the WCETs of previous generations of the two avionics programs. This method was a mix of measurement and

llectual analysis. Let's briefly describe how it worked for each of the two programs.

First avionics program. Basically, the way the WCET is obtained takes profit of the structure of the program, which is produced by an automatic code generator taking a graphical specification (SCADE) as input. The generated program almost entirely consists of instances (or routine calls to) a rather limited set of small snippets like logical and comparison operators (multiple input AND, OR, Level Detector), arithmetical operators (ADD, MUL, ABS, etc), digital filtering operators, etc. The small size of these basic components makes it possible to measure their WCETs, as long as the initial execution environment for each measurement is the worst possible with respect to execution time. One must also notice that the generated program is very linear since all conditionals and loops occur in the small snippets, thus being very local. The program being made of instances of (or calls to) code snippets, the WCETs of which are available after the measurement campaign, a simple formula (more or less a sum) is applied for inferring the whole WCET, knowing that each (instance of a) code snippet is executed once and only once.

Second avionics program. The second program also consists, for the main part, of basic processing units, each based on a small number of patterns. The size and limited execution paths inside these patterns make it also possible to measure their WCETs in the same conditions as for the first program. But the main difference to the first program is that the basic units form the implementations of some abstract machine instructions collected in an instruction table (also called configuration table). The execution of the program is driven by a high-level loop, reading instructions and parameters from the configuration table, and then calling the appropriate basic processing units implementing the instructions. Each basic processing unit being called multiple times, the resulting WCET is the sum of the basic WCETs of the instructions present in the configuration table.

The traditional approach is correct with "deterministic processors". As stated in section 1, the computed WCET must be sound, i.e., an upper bound of the real WCET, and tight, i.e., not too far from the real WCET. With processors having no – or few – history-based accelerating mechanisms, both criteria are met without too much difficulty since finding the worst possible initial environment for the measurement of the WCET of each snippet and producing a – non-formal – demonstration that the computed WCET is an upper bound of the real WCET is accessible to a human being, and the overestimation is acceptable.

Modern processors. But even for such deterministic programs, the legacy methods are no more applicable when using hardware components like the ones mentioned in section 2. The first reason is that it is a lot harder to find the worst possible initial environment for measuring the WCETs of the small snippets, and also to get sure that combining these WCETs for computing the WCET of the entire program is safe. The second reason is that even if we could solve the first difficulty (worst initial environment and safe combination), the resulting WCET would be too much overestimated for being useful. Both reasons originate from the history-dependent execution time of each instruction that makes it impossible to deduce a good global WCET from any local measurements or computations.

3.1 Asynchronous extra time (SDRAM refresh and Host Controller (for Input/outputs) activity).

When measuring the WCET of the small snippets the whole program is made of, only the effect of SDRAM refreshes is measured. It is therefore not possible to infer a WCET of the program that encompasses the effect of the Host controller

4 aiT for PowerPC MPC755

AbsInt's aiT tools form a family of WCET tools for different processors, including PowerPC MPC755. aiT tools get as input an executable, an .ais file containing user annotations, an .aip file containing 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). The names of the various input files and other basic parameters are bundled in a project file (.apf file) that can be loaded into a graphical user interface (GUI).

All instances of aiT determine the WCET of a task in several phases: **CFG building** decodes, i.e. identifies instructions, and reconstructs the control-flow graph (CFG) from a binary program. User annotations may help aiT in identifying the targets of computed calls and branches. **Value analysis** computes value ranges for registers and address ranges for instructions accessing memory. **Loop bound analysis** determines upper bounds for the number of iterations of simple loops. Such upper bounds are necessary to obtain a WCET. Loop bounds that cannot be determined automatically must be provided by user annotations. **Cache analysis** classifies certain memory references as cache hits. **Pipeline analysis** predicts the behavior of the program on the processor pipeline and so obtains the WCETs of the basic blocks. **Path analysis** determines a worst-case execution path of the program.

Value analysis is based on an abstract interpretation of the operations of the analyzed task, taking into account variable values specified by the user (e.g. to restrict the analysis to a certain operation mode of the analyzed software). The results of value analysis are used to determine loop bounds, to predict the addresses of data accesses and to find infeasible paths caused by conditions that always evaluate to true or always evaluate to false. Knowledge of the addresses of data accesses is important for cache analysis. Value analysis usually works so good that only a few indirect accesses cannot be determined exactly. Address ranges for the remaining accesses can be provided by user annotations.

Cache Analysis uses the results of value analysis to predict the behavior of the (data) cache. The results of cache analysis are used within pipeline analysis allowing the prediction of pipeline stalls due to cache misses. The combined results of the cache and pipeline analyses are the basis for computing the execution times of program paths. Separating WCET determination into several phases makes it possible to use different methods tailored to the subtasks. Value analysis, cache analysis, and pipeline analysis are done by abstract interpretation. Integer linear programming is used for path analysis.

5 WCET computation with aiT for PowerPC MPC755

First avionics program. We first present some more details about the structure of the program. It consists of 24 uninterruptible tasks that are activated one-by-one by a real time clock in a fixed schedule: task 1 to task 24, then task 1 again, and so on until the electrical power of the aircraft is switched off. This time-triggered scheduling method requires that the WCET of

each task must be less than the period of the real-time clock. The call graph of each task is basically organized in three layers. The first layer contains 4 calls to so-called sequencers, which for each task are selected from a list of 38 possibilities. These sequencers allow for the activation of pieces of code at different rates, i.e., 1 over 2 ticks, 4 ticks, 8 ticks or 24 ticks. Still in this highest layer, some system routines are called before and after the four sequencer calls. The second layer consists of the routines containing the actual operation code composed of "calls" to code macros, which form the basic components referred to in section 3. The third layer consists of the input/output routines called by some of the basic components present in the second layer.

As described in section 4, aiT requires some global parameter settings in the graphical user interface, in a project file (.apf file), or in a parameter file (.aip file), and user annotations in an annotation file (.ais file). Global parameters are quite basic and mainly describe the hardware, so let us concentrate on the work involved in writing the annotations. As described in section 4, annotations are mandatory when value analysis fails to find the iteration bound of some loop or the computed address range for an external access (load or store) is too imprecise. We now present the annotations required in the analysis of the first avionics program. As stated above, its functional code consists of operators implemented as macros and a few input/output and "system" routines.

Annotations in operators:

- cos(x) and sin(x): The values of these functions are extracted from some table. The index into this table is computed from the floating-point number x given as argument. As aiT does not "cover" floating-point calculus, its safe reaction is not to compute any bounds for the value of the index. Hence the address range of the table lookup is unknown. This range, which is in fact a function of the size of the table, is provided by a user annotation.

- delay family (4 operators): These operators are called once in the course of each task to store a value into some array. They also update a static variable containing an index to this array. Since the periodic updates of the index are triggered by the real time clock and not by a loop inside the code, aiT cannot compute the range of addresses for this index. Thus an annotation has to be provided.

If the basic operators were implemented as routines, the number of annotations as described above would have been very limited and, thus, affordable easily. Yet the operators are implemented as code macros. Consequently, what would be a single annotation in case of a routine splits into as many annotations as there are instances of the macro in the code, which is quite huge.

An automatic generator of annotations. For solving this industrial problem, it was decided to use the same technique for annotating as for the coding itself, namely automatic generation. Indeed, an automatic generator of annotations has been implemented that reads a generic description of an annotation in a given macro (basic operator) and produces the actual instances of this annotation for all occurrences of the macro in the binary.

Annotations in I/O or "system" routines:

- Communication drivers (USB, AFDX): a few pointers involved in the data exchanges via these communication channels are too dynamic for being computed by aiT automatically. The relevant address ranges must be provided via annotations.

- Loops: Annotations are required for a small number of loops whose iteration bounds cannot be determined by aiT.

The second avionics program is also based on uninterruptible tasks activated by a real time clock. The tasks must complete in the allowed time to guarantee proper functioning of the program. Each task consists of a high-level loop that reads a list of operation codes with parameters from a configuration table, and for each operation code, calls the basic blocks implementing the operation. These basic blocks are highly dependent on parameterisations provided by the configuration table. The parameterisations influence the timing behaviour because they may induce specific processing or affect loop bounds and addresses of external accesses.

Therefore, it is of first importance for aiT to keep track of these parameters as they are copied from the configuration table to temporary variables and registers. In the first place, the configuration table has to be recognized by aiT. The configuration table is located in a specific area of a binary file. aiT supports an annotation saying that a data area in the analyzed binary is "read-only" so that the values found there can be used by value analysis. However, in the program considered here, this constant area is provided as a separate binary because the configuration tables are separate loadable parts. Since aiT can analyse only single binaries, the binary with the configuration table (this is Mbytes of data) is translated into .ais file annotations specifying the contents of the table. As the same basic blocks are called multiple times in one task activation, either directly from the main loop, or because they are low-level shared services, some instruction cache hits can be guaranteed. However these cache hits depend on the ordering of the operation codes in the configuration table and cannot be taken into account by the standalone basic block WCET measurements of Airbus' "traditional" approach (simple sum of basic block execution times). Hence this method cannot take advantage of the improvements of modern processors, and thus cannot provide WCET results compatible with the allowed execution time.

The major part of the factors affecting the WCET (conditions, loop bounds, pointers, etc) is found automatically by aiT, either by code inspection or from the annotations describing the configuration table. Yet some factors are outside aiT's knowledge and capacities, and annotations have to be provided to bound the analysis and achieve a result. These factors are lower and upper bounds on input data, static data from previous task activations, or data provided by devices outside of processor knowledge (DMA for example). For these, maximum loop iterations, values read from memory, branch exclusions, etc, have to be specified to aiT.

There are 256 different configuration tables, corresponding to different tasks of the same computer, and of different computers in the aircraft. One objective of the separation of software in configuration tables (lists of operation codes with parameters) and the code for interpreting these tables is to shorten (in terms of delay between specification and deliveries) the development and validation cycles for the software. This objective requires an automatic computation of the WCET without direct human involvement.

WCET determination for the combination of the interpreter code with each of the 256 configuration tables consists of determining the longest execution path in a quite complex piece of code. These WCET analyses are to be performed in a period of time compatible with the industrial constraints associated with the short development cycle. This however is possible due to

fact that the analyses for the 256 tables are independent from each other and can be performed in parallel on many computers.

6 Considerations about the results

The WCETs of the two programs are not yet publishable for general confidentiality reasons; the authors apologize for that and hope the reader will understand.

Nevertheless some considerations about the results and how they were obtained are presented in this section.

First avionics program. This program has a very linear structure and limited sources of jitters (in number and amplitude). This allows for valid comparisons between the figures obtained by measurement during real executions of the tasks and those produced by using aiT. The comparison shows that the WCET of a task typically is about 25% higher than the measured time for the same task, the real but non-calculable WCET being in between.

Another comparison is worth to mention: the one between aiT's results and those of Airbus' "traditional" method. As predicted when the decision for using aiT was made, the figures obtained by the traditional approach are a lot higher than those produced by the aiT-based method. Actually the overestimation is such that the "traditional" figures are useless.

Second avionics program. Despite the fact that this program has not a linear structure and contains a great number of loops and branches, the sources of jitter are limited. Most of the loop and branch conditions can be evaluated knowing the configuration table. There are still more sources of jitter than in the first program, but comparisons between the figures obtained by measurement of real executions and results produced by aiT are still possible. The WCETs computed by aiT are about 50% higher than the corresponding measured execution times. These higher margins can be explained (at least in part) by the difficulties to set up an environment for running the task that is sufficiently close to the environment leading to the real WCET.

More importantly, the WCETs computed by aiT are much lower than the results from the "traditional" method and are compatible with the program's timing constraints.

Status with respect to some general acceptance criteria.

- Soundness: First of all, aiT was developed in the framework of Abstract Interpretation [2], which is very good for the soundness of the underlying principles. The soundness of the actual implementation of the tool, as used on the real avionics programs, cannot be assessed formally. The way of getting confident in the tool and its method of use is the Validation/Qualification process sketched in section 7.
- Ease of use: Users, i.e., normal engineers, always worked with aiT in an autonomous way. Furthermore, they were able to develop utilities for making aiT easier to use in the industrial context, like the automatic generator of annotations (see section 5).
- Resource needs: The WCET computations for the second avionics program (the most demanding one) take an average of 12 hours (on a 3+ Ghz Pentium 4) per task (there are 256 tasks). But this workload is dispatched on many computers and is not a real problem. The main concern is about the space requirement that is, for some analyzed tasks, close to the current 3 Gbytes limit of the 32-bit architecture. To address this point, a migration to the 64-bit architecture is under investigation.

7 future work

Validation. By this term the authors mean "getting a high level of confidence" in the results produced by aiT as used on the avionics programs referred to in this paper. Although some data collected during the first industrial usage of aiT are already available for validation, most of the work is still do be done. It will have four basic objectives: soundness of the underlying principles (a), correctness of the models (b), soundness of the method of use (c), and validity of the results (d).

(a) and (b): most of the analyses performed by aiT (see section 3) have theoretical principles (models and sometimes proofs) precisely described in theses or scientific papers. Some effort has already been spent in reading this documentation. This process will carry on together with some checking against hardware manuals, e.g., the PowerPC MPC755 manual [3].

(c): the objective here is mainly to check that the annotations (loop bounds, register contents at certain program points, etc) do not make aiT compute an unsafe WCET.

(d): the very detailed results produced by aiT make it possible to perform useful and automated checks. An example of such checks is whether real execution traces (got via a logic analyzer) belong to the set of those computed by aiT.

Qualification. This term comes from DO178B. In the case of aiT, it is about the qualification of a verification tool. Current practices consider that the qualification is twofold: in-service history and qualification tests. Both items of the qualification will be built from the outputs of the validation process. Also, one should notice that the qualification of a tool is per avionics program the tool is used on.

References

[1] Stephan Thesing, Jean Souyris, Reinhold Heckmann, Famantanantsoa Randimbivololona, Marc Langenbach, Reinhard Wilhelm, and Christian Ferdinand. An Abstract Interpretation-Based Timing Validation of Hard Real-Time Avionics. In *Proceedings of the International Performance and Dependability Symposium (IPDS)*, June 2003

[2] Patrick Cousot. Interprétation abstraite. *Technique et Science Informatique, Vol. 19, Nb 1-2-3.* Janvier 2000, Hermès, Paris, France. pp. 155-164.

[3] Motorola. *MPC750 RISC Processors User's Manual.*