Difficulties in Computing the WCET for Processors with Speculative Execution

Christine Rochange and Pascal Sainrat
Institut de Recherche en Informatique de Toulouse, France
{rochange, sainrat}@irit.fr

Abstract

In real-time applications, the Worst-Case Execution Time often needs to be estimated to check that deadlines will be respected. With the trend of using up-to-date processors, WCET computation techniques continuously have to evolve in order to take into account the most recent hardware features. In this paper, we show that ignoring speculative execution can lead to underestimated execution times, and we explain why modelling it is not straightforward. We feel that pure static analysis might not allow safe WCET computation, due to the fact that speculative execution prevents the decoupling between the high-level (path) analysis and the low-level (timing) analysis.

1. Introduction

For a large class of applications, embedded software has to satisfy hard real-time constraints. This requires to be able to tightly estimate the worst-case execution time (WCET) of programs.

WCET analysis has received much attention these ten last years. Dynamic methods involve measurements on real hardware or on cycle-level simulators. All the possible execution paths have to be explored in order to obtain the longest execution time. This poses two problems: (i) the number of possible paths is generally high and then the measurement time is prohibitive; (ii) for each path, the corresponding input data set has to be defined, which is usually difficult. In response to the drawbacks of dynamic methods, several static approaches have been proposed. They consist in three steps. First, the high-level analysis considers the program code in order to identify the possible execution paths, where a path is a list of basic blocks. Second, the low-level analysis estimates the execution time of each basic block. It is carried out in two phases: the global low-level analysis takes into account hardware components the behaviour of which depends on the global history of execution (e.g. cache memories); the local phase models components that only depends on the recent history (e.g. pipeline). Third, the execution times of paths are computed and the WCET is the longest one.

However, embedded systems tend to use modern processors featuring advanced architectural mechanisms that might be hard to model. Among these mechanisms, branch prediction, sometimes coupled with speculative execution, is implemented in most of the recent processors.

Estimating the WCET for processors with speculative execution does not present any special difficulty when it is based on dynamic measures: either the real target hardware is available (with speculative execution activated), or a cycle-level simulator is used, and speculative execution is not harder to model than other advanced features. However, current dynamic measurement methods often require to explore a too large number of execution paths and, for this reason, static analysis is generally preferred.

In this paper, we will show that estimating the WCET when a processor implements speculative execution is not straightforward. We suggest that usual static analysis techniques might not allow safe WCET computation, highlighting situations where they would lead to underestimation of the execution time.

Section 2 gives an overview of branch prediction and speculative execution techniques, and presents the work of Colin and Puaut [1] that takes branch prediction (but not speculative execution) into account within static WCET analysis. Section 3 shows why it is important to carefully model speculative execution to obtain a safe WCET. Section 4 discusses the difficulties of doing it within pure static WCET analysis, and section 5 concludes the paper.

2. Branch prediction and speculative execution

2.1 Overview

Modern processors are designed around longer and longer pipelines. Whenever a branch instruction is encountered in the instruction flow, the correct execution path is not known until the branch is executed. To avoid interrupting the instruction fetching, one of the two possible paths is speculatively selected by a branch predictor and instruction processing continues along this path. When the branch is resolved and if the speculative path is not the correct one, recovery actions are taken (e.g. the pipeline is flushed) and instruction processing restarts from the branch along the right path. Processing along a speculative path means fetching instructions from

the memory hierarchy, decoding and dispatching them to the reservation stations where they wait for their operands. For a processor that implements out-of-order execution, instructions belonging to the speculative path might also be executed before earlier instructions, and in particular before unresolved branches. This is what is called *speculative execution*. In that case, recovery from branch misprediction is a bit more complicated and generally requires mechanisms to restore the correct architectural state. Note that recovery is only required for components that must have a safe behaviour: the effects of a branch prediction error on other components, like cache memories or the branch predictor itself do not endanger correct functional results, they only might lower the system performance.

Many algorithms exist to predict the issue of branch instructions. The most recent ones include three kinds of structures:

- the *PHT* (*Pattern History Table*) is used to predict the direction of conditional branches: each of its entries reflects a recent history (often as a 2-bit saturating counter). The PHT is usually not tagged and it can be indexed by the instruction address (PC) alone or combined with a global or a local history recorded in one or several *BHR* (*Branch History Register*). Thus, several branches share the same counter and, on the contrary, the behaviour of a branch depends on several counters according to the history.
- the *BTB* (*Branch Target Buffer*) is used to predict the target address (except for subroutine returns)
- the *RAS* (*Return Address Stack*) is used to predict subroutine returns.

2.2 Computing the WCET for processors with branch prediction

As far as we know, branch prediction has been considered within WCET analysis only for in-order processors: this work has been presented by Colin and Puaut [1]. They consider the Intel Pentium, which features a simple branch predictor based on a single table referred to as BTB. The proposed method includes several stages.

First, the control-flow graph is analysed to build an *abstract state* of the BTB for each basic block: it indicates which instructions might be contained in each entry of the BTB before and after the execution of the basic block. The *input* abstract state of a basic block is computed from the *output* abstract states of the possible preceding basic blocks. Then, the abstract states are used to classify the branch instructions and to determine, for each of them, if it will be correctly predicted or not. When ever this cannot be statically decided, the instruction is assumed to be mispredicted, which is supposed to be the worst case.

The WCET is then computed in two steps. First, a perfect branch predictor is assumed, and the WCET is estimated from the syntax tree and a set of formulas that express the maximum execution time of algorithmic structures. Then the timing effects of prediction errors are evaluated for a real branch predictor: a penalty delay is associated to each possibly mispredicted branch instruction. A second set of formulas is used to recursively build delay sets for each algorithmic structure of the syntax tree. The sum of these delays is then added to the WCET previously computed with perfect branch prediction.

3. Possible effects of speculative execution

When a processor implements speculative execution, processing along the wrong path may have two kinds of effects on the system. In this section, we describe these effects and show why ignoring them can lead to underestimate the WCET.

3.1 Dynamic instruction scheduling

Instructions of the wrong path occupy hardware resources, like functional units. Now, some flushing policies implemented for branch prediction error recovery do not immediately free the functional units. Thus, an instruction of the wrong path might continue its execution in a multi-cycle functional unit after the flushing of the pipeline (but its result will then be simply discarded). If the functional unit is not pipelined, the execution of later instructions belonging to the correct path might be delayed. Then the misprediction penalty would be longer than the strict recovery time.

Moreover, inserting wrong path instructions in the pipeline can modify the scheduling of previous instructions. For example, an instruction belonging to the wrong path that has its operands ready can be scheduled before a preceding instruction that is waiting for one of its operands. This might completely modify the overall scheduling, and then the execution time as mentioned in [3].

If speculative execution is not taken into account, the pipeline reservation tables produced by the local timing analysis might not be correct and the computed WCET could be underestimated.

3.2 Memory contents

Processing along the wrong path can also change the content of memories. If instructions of the wrong path miss in the instruction cache, they are fetched from the upper level of the memory hierarchy. This can have a detrimental effect on the program execution time if instructions of the wrong path replace in the cache instructions belonging to the correct path: when the execution later restarts along the right path, those replaced instructions will miss in the cache, thus requiring longer fetch times. This detrimental effect is often referred to as *cache pollution*. Note that fetching instructions along the wrong path can also have a beneficial effect, as reported in [4]: some instructions of the wrong path can later be found on the correct path, and then processing along the wrong path acts as a prefetch mechanism.

The same effects can be observed on data accesses, provided that instructions of the wrong path are executed (not only fetched), which is only allowed in dynamically-scheduled processors.

Processing along the wrong path may also have a beneficial or detrimental impact on the memories of the branch predictor (BTB, BHR, PHT and RAS) if they are updated speculatively in the earlier stages of the pipeline [2]. Only few parts are checkpointed for cost reasons (e.g. checkpointing the BTB is probably not affordable). If recovery is not implemented, the branch predictor tables might be polluted by the execution of the wrong path.

Now, let us assume that, ignoring speculative execution, the global low-level analysis is able to statically determine the real behaviour of all instruction and data cache accesses (hit or miss) and of all branches (well- or wrong- predicted). To understand why the possible pollution of memories due to wrong path execution should not be ignored, let us consider the following example:

```
for (i=0 ; i<10 ; i++)
{
      s[i] = 0;
      for (j=0 ; j<10 ; j++)
      {
            s[i] = s[i] + t[i][j];
      }
      m[i] = s[i] / 10;
}
```

This program may be compiled as:

```
L0    i=0;
L1    if i==10 then branch to L7
L3    s[i]=0
      j = 0
L4    if j==10 then branch to L6
L5    s[i] = s[i] + t[i][j]
      j++
      branch to L4
L6    m[i] = s[i] / 10
      i++
      branch to L1
L7
```

If we consider a branch prediction algorithm based on 2-bit saturating counters, initialised to "weakly-taken", the branch instructions of basic blocks `L1` and `L4` are mispredicted in the first iteration of loops `i` and `j`, and correctly predicted in the other iterations, while those of basic blocks `L5` and `L6` are always well predicted. As far as data accesses are concerned, the first reference to `s[i]` is determined to miss in the data cache while the other ones should hit.

Now, what does really happen if the processor implements speculative execution? At the first iteration of loop `i`, since the branch of basic block `L1` is mispredicted, some instructions belonging to the wrong path are processed. In particular, access to `m[i]` might be executed. If `m[i]` happens to fall in the same cache line as `s[i]` then, when the branch is resolved and the execution restarts along the correct path, `s[i]` misses in the data cache, contrarily to the conclusion of the global low-level analysis. As a result, the actual execution time might be longer than the estimated WCET, which is not acceptable.

In the same manner, the possible pollution of other memories (instruction cache, branch prediction tables,…) can increase the execution time. Ignoring speculative execution might again lead to an erroneous classification of instructions (branches or memory accesses) in the global low-level analysis step, which may result in an underestimated WCET.

4. Towards a safe WCET estimation for processors with speculative execution

We have shown why the execution of the wrong path has to be carefully taken into account in order to obtain a safe WCET. In this section, we discuss the difficulties of modelling speculative execution as part of static WCET analysis.

The possible effects of speculative execution on the dynamic scheduling of instructions can probably be taken

into account without excessive complexity. For example, the delaying of the execution of later instructions due to the occupation of hardware resources by wrong path instructions could be included in the WCET estimation by systematically adding to the misprediction recovery penalty the longest functional unit latency. An other solution would consist in assuming in-order instead of out-of-order execution, but it would lead to a very pessimistic WCET estimation.

The effects of speculative execution on the content of memories may be harder to take into account within a purely static WCET analysis. We have seen that it can invalidate the results of the global low-level analysis: memory accesses (either to instructions or data) classified as "cache hits" might actually miss due to the pollution of the cache by the execution of the wrong path; branches classified as "well predicted" might actually be mispredicted due to the pollution of branch predictor tables. This means that, in order to produce a safe classification of instructions, the global low-level analysis should take into account the instructions of the wrong path. Now, we feel that considering wrong paths within static analysis is not straightforward, since it probably requires new algorithms for syntax tree or control-flow graph traversal. Moreover, the number of instructions or basic blocks to include in a wrong path depends on the processor state (occupancy of hardware resources) and can only be determined during the local low-level analysis. Thus, it appears that a correct modelling of speculative execution would require a very close interaction between the high-level (flow) and low-level (timing) analyses, which are usually carried out independently.

While decoupling the analyses of different components (caches, branch predictor, pipeline, …) probably makes static WCET computation feasible, we wonder if the requirement of more interaction between these analyses to be able to take into account more and more advanced hardware features can be satisfied. If not, growing emphasis should be put on dynamic measurement (on real systems or simulators) to obtain accurate timing information while the static part of WCET estimation would focus on selecting the execution paths to explore with the goal of minimizing the measurement requirements.

5. Conclusion

A lot of work has been done these last ten years to allow static estimation of the WCET for processors with advanced features like cache memories, pipelined execution, branch prediction. The most recent dynamically-scheduled processors implement speculative execution: when a branch instruction is predicted, the instructions belonging to the predicted path can be executed before that the branch is resolved. In this paper, we discussed the possible effects of executing the wrong path whenever a branch is mispredicted.

We have shown that wrong path execution can modify the scheduling of the correct path instructions and/or change the content of memories (instruction and data caches, branch predictor tables, …). Then we have explained why ignoring these effects in WCET analysis can lead to an underestimated WCET, which can be dramatic for hard real-time systems.

We feel that usual static WCET computation techniques cannot accurately take speculative execution into account, since it would require a too complex interaction between global and local low-level analysis.

6. References

[1] A. Colin, I. Puaut, "Worst-Case Execution Time Analysis for a Processor with Branch Prediction", *Real-Time Systems, 18(2):249-274,* May 2000.

[2] S. Jourdan, T.-H. Hsing, J. Stark, Y. Patt, "The Effects of Mispredicted-Path Execution on Branch Prediction Structures", *Int. Conf. On Parallel Architectures and Compilation Techniques*, Octobre 1996.

[3] T. Lundqvist, P. Stenström, "Timing Anomalies in Dynamically Scheduled Processors", *20th IEEE Real-Time Systems Symposium*, December 1999.

[4] J. Pierce, T. Mudge, "Wrong-Path Instruction Prefetching", *IEEE Int. Symp. On Microarchitecture*, December 1996.