Re-targetable Framework for Worst-Case Execution Time Analysis

Iain Bate
BAE SYSTEMS Dependable Computing Systems Centre,
Department of Computer Science,
University of York, York, United Kingdom.
e-mail: iain.bate@cs.york.ac.uk

Relevant Research Themes: Embedded Systems and Safety-Critical Systems

Abstract

The ability to estimate the Worst-Case Execution Time (WCET) of software is essential for all forms of timing analysis. In critical systems, there is a need to obtain safe estimates which can only be achieved via analysis. Traditionally, WCET analysis approaches have been tightly coupled to specific compilers and processors. The purpose of this paper is to describe a framework that has been developed for WCET analysis that is not tied to a specific compiler or platform. Instead, existing analysis has been generalised so that it can be instantiated at run-time for a particular platform, in this case using a purpose-built language. The framework produced allows the tool to be re-targeted without (in most cases) modification/re-compilation of the analysis software.

1 Introduction

The purpose of this work is to produce appropriate models for the Worst-Case Execution Time (WCET) analysis of modern processors. The ability to estimate the WCET of software is essential for all forms of timing analysis. WCET analysis is the process by which the maximum execution time of a piece of software on a particular hardware platform is obtained. There are two principal parts to WCET analysis:

- higher-level analysis of the program to deduce the bounds on loops and mutually exclusive paths, and
- lower-level analysis related to how the instruction(s) are actually processed.

Work in York [1], amongst others [2], has already looked at manual methods for dealing with the higher-level analysis, whilst others have looked at automatic methods [3]. Until recently, this was sufficient because the processors most often used (e.g. 68020) did not tend to have features (e.g. caches and pipelines) found in modern processors. In critical systems that use older generation processors, if a processor has a cache then it is often disabled due to the perceived difficulty in analysing its operation. However if the modern processors were used and suitable analysis were available, then it is likely it would be used. The type of processor being used in development projects is changing to more modern devices (e.g. superscalar).

The change is happening not because of the need for more processing power but because obsolescence is meaning supplies of currently used processors are running out. Whilst the processor could operate without a cache or with the cache disabled, albeit inefficiently, the use of pipelines can not be avoided. If during WCET analysis advanced features of the processor (e.g. cache and pipeline) are ignored, then the pessimism in the analysis can be at least an order of magnitude (refer to section 2.1 for details) which could be an un-manageable amount. This would mean little of the processor's resource could be used since the worst-case processor utilisation has to be within defined bounds. This could lead to a decision to base the WCET of tasks on a measured value, however this means systematic failures could occur in service which for some systems, e.g. safety-related, is not acceptable. Therefore, this work has been performed with an overall aim of developing appropriate lower-level analysis whilst building in support for the higher-level analysis.

A great deal of work has been presented on the subject of WCET analysis, including analysis of caches and pipelines, examples of which can be found. The approaches proposed vary greatly in their complexity and precision, but the few that support re-targetability would require significant re-work to achieve it. Whilst the techniques can be re-applied to other processing architectures, the models of the architectures are contained within the analysis which means the analysis software has to be modified.

The contribution of this work is to select and reuse existing techniques for WCET analysis and configure these into a re-targetable framework. The architecture of the framework has been produced by identifying a suitable abstraction layer that minimises the effort in tailoring it for a specific problem without prohibiting the framework from supporting processors that may need to be analysed. The implementation of the framework is achieved using a target-independent model that can be instantiated for a specific processing architecture using details expressed in a WCET language that has been developed. Since the details of the processing architecture are held outside of the actual analysis tool, then this removes the need to modify the analysis software when re-targeting the analysis to a different platform. The contribution of this paper is not the individual analyses but the way in which they are tailored to be generic. The reason is the analyses are considered to be replaceable facets of the overall solution and most of the forms of analyses already exist.

The structure of the report is as follows. Section 2 gives background and motivation to the problem of WCET analysis, the constraints placed on the work so an appropriate solution is obtained and the assumptions made during the course of this work. Section 3 presents the overall strategy taken for providing a WCET analysis tool. Section 4 presents some examples of how cache configurations may be represented and based on these an evaluation of the approach is carried out. Finally, section 5 presents the summary and considers the future work that could be performed.

2 Motivation and Background for the Work

The purpose of this section is to examine why the work is viewed as important and the key issues to be considered when producing suitable analysis.

2.1 Why the Analysis is Essential

WCET analysis is the mechanism for determining the maximum possible time a piece of software takes to execute. There are two principal ways for obtaining the value; the first and most often used is test and the second is via analysis. In our experience, test-based techniques often involve running the software with a large number of scenarios and measuring what the WCET is. Other test-based techniques use guided search techniques (e.g. genetic algorithms) that evolve test cases which are hoped to include the worst case.

The method chosen for obtaining the WCET of software is clearly dependent on the needs of the project. If achieving the maximum possible use of available processing is the key concern, rather than risk of failure to meet timing requirements (which is a primer driver for safety-critical real-time systems), then test-based techniques have the advantage. This can be a misguided advantage if allowance is made for the potential optimism in the results leading to an engineering margin having to be built into the results. Otherwise, the analysis-based techniques have the advantage. If WCET analysis were used that ignored cache and pipelines to analyse PowerPC 603e code, the pessimism could be 1000% or more. This figure can be justified by considering the realistic case of a 500 MHz processor that uses 50 nano-second memory, which leads to at least 25 (= 500 / (1000/50)) wait states per memory access. Ignoring the four stage pipeline would mean the analysis produces a result up to four times the actual execution time. This is due to the analysis having to assume each instruction propagates through every stage before the next instruction could begin.

2.2 The Constraints on the Problem

When deriving an approach for WCET analysis it is useful to place a number of constraints on the problem so that the resulting analysis is of practical use. The following are constraints used to influence this work:

- *Computational Complexity* - It would be possible to produce analysis software that effectively emulated the processor and then exercised all possible cases. However for some parts of the analysis problem, it could be intractable for reasonable sizes of software.
- *Pessimism* - Again, it would be easy to produce analysis software whose results were so pessimistic that the effective utilisation of the processor was less than that of the processors being replaced - refer to section 2.1 for details.
- *Re-targetability* - Whilst achieving the earlier two constraints, a model is developed that is highly optimised for a particular processor and/or compiler. It is not a goal of this work to tie the user(s) to a particular processor, in fact the principal constraint on this work is to ease the problem of re-targeting the solution.
- *Qualification* - For any approach to be useful, it is necessary to validate that the analysis model is safe (i.e. the results of the analysis is always greater than or equal to the actual WCET) and ensure the tool is not so complex the model is impossible to justify. The validation of the pipeline model is complicated since the pipeline's internal operation cannot be observed. This means greater trust has to be placed in the information provided by the manufacturer. Little work has been performed on the subject of processor model validation.

3 Framework for WCET Analysis

The principal part of the WCET framework that is being presented is a generic analysis model instantiated at run-time with instruction set and hardware details that are input using a language that has been developed. The analysis model and language has been defined with an abstraction layer that attempts to minimise the effort needed to instantiate the model whilst allowing the resulting model to be as versatile as possible - ideally truely generic. The instruction set and hardware details are needed to allow the generic tool to interpret the software to be analysed and then determine how individual instructions would be processed by the platform. The approach has been taken so that when re-targeting the analysis to a new platform the analysis software itself should not need altering, only an external file. This should lead to significant cost savings when porting the tool to a new platform. The cost savings are at the expense of a slight increase in the time required to do the analysis caused by the need to invoke the model for a specific platform and an analysis model that contains some parts that are not always used.

The description of a specific platform is not represented using one of the existing languages (e.g. VHDL) because the methods tend to have an inappropriate level of abstraction. This could lead to long complicated descriptions having to be produced for what could be relatively simple concepts. Consider for example defining three pipeline stages where two of the stages can be accessed in parallel and these stages receive instructions after the other one has processed them. In VHDL, a number of procedures would have to be written in-order to represent the situation. The result would be hard to parse. Instead, the proposal is to develop a simple language (referred to as the WCET language), represented in BNF, for defining processing architectures. In the case of the pipeline mechanism, this entails the convenient definition of the individual pipeline stages, their interactions with other stages, and how the instructions actually make use of the stages.

The overall framework that has been produced (in Ada) is given in Figure 1. A key feature of the framework is that the analyses are performed as independent of one another as possible with the results of each then being combined to give the desired result. In addition, interfaces are defined to allow communication between the analyses and to allow individual analysis to be changed without affecting others, i.e. there is a modular architecture. The WCET analysis is based on a path-based approach, rather than IPET (Implicit Path Enumeration Technique) [2] or tree-based approaches [3]. The reasons for this choice are that previous work at York [1] has used the path-based approach, allowing for cache and branch prediction is more straightforward and the analysis results include the worst-case path which is useful for manual validation.

Figure 1 shows the WCET analysis is performed on a disassembled version of the software. This strategy is taken so that the tool is easier to validate during prototyping. It is envisaged a production version would operate directly on the object code. The software is then split into basic blocks, a basic block being a series of instructions with branches only at its entry or exit.

Path analysis is the process by which all possible paths (i.e. sequence of basic blocks) are determined and the path associated with the WCET deduced. The path analysis uses information provided in the file *Manual Path Information* which contains information about maximum loop bounds and mutually exclusive paths. The set of paths is then fed into the lower-level forms of analyses. In the future framework, automatic methods of providing this information could be added.

Pipeline analysis is performed in two stages; each basic block is analysed once to determine how the instructions exercise the pipeline, and then for a particular path the details gathered are joined to give the Path's Processing Time (PPT) within the pipeline. This analysis assumes that memory accesses result in a cache hit and all branches are correctly predicted.

Cache analysis is performed for each path in the program in order to determine the number of Cache Misses (CM). Similarly branch prediction is performed for each path to determine the number of Incorrect Branch Predictions (IBP). The results of the analyses are then combined using equation (1) to determine the Overall Execution Time (OET).

$$OET = PPT + \sum_{\forall CM} CM_{Penalty} \sum_{\forall IBP} IBPPenalty \qquad (1)$$

4 Evaluation of the Approach

For reasons of space, the complete WCET language, and the analysis to support it, cannot be described. Instead, this paper gives examples of the WCET languages' use and presents some evaluation.

Part of the evaluation was considering whether the language was sufficiently complete that all processors could be represented and analysed. After some consideration, it was concluded that there are processors now and in the future that may not be represented efficiently in this framework. However based on discussions with people working with embedded systems and the fact a variety of processors have been represented (e.g. Intel Pentium family, ARM family, Motorola PowerPC family and SHARC – DSP), it is considered that the framework is sufficient.

Another evaluation metric for WCET analysis is pessimism. Our approach should introduce little pessimism because the approach provides an accurate model of the cache mechanism. The only pessimism is attributable to:

- the fact we assume the cache is empty when entering the code being analysed, and
- the mechanism for analysing for the impact of preemption on the cache, i.e. the cache is flushed.

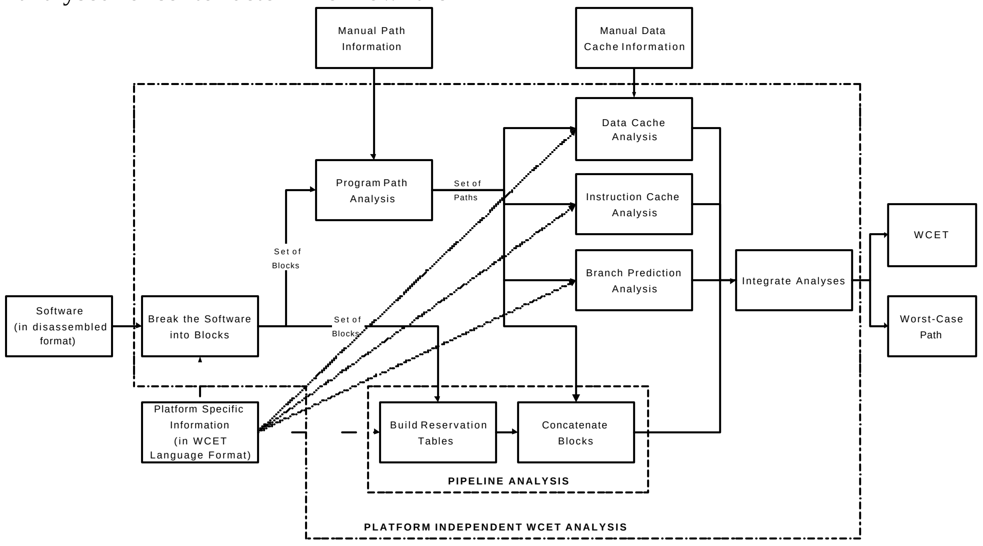
Manual Path Information
Manual Data Cache Information
Data Cache Analysis
Program Path Analysis
Set of Paths
Instruction Cache Analysis
Set of Blocks
Branch Prediction Analysis
Integrate Analyses
WCET
Worst-Case Path
Software (in disassembled format)
Break the Software into Blocks
Set of Blocks
Platform Specific Information (in WCET Language Format)
Build Reservation Tables
Concatenate Blocks
PIPELINE ANALYSIS
PLATFORM INDEPENDENT WCET ANALYSIS

Figure 1 - Framework for WCET Analysis

Judging the pessimism of an approach for any non-trivial example is impossible because of the difficulty in determining an exact WCET value. However, a partial evaluation can be achieved by comparing the WCET results using different approaches. Table 1 compares the results for the two different instruction sets, PowerPC and ARM, each with different cache configurations.

The following example presents the details for a PowerPC 603e with a *primary* cache that has 1024 cache lines (represented using the WCET language construct **`number_of_cache_lines`**) each containing 32 bytes (represented using the WCET language construct **`number_of_bytes_per_cache_line`**) with each cache line being memory aligned (represented using the WCET language construct **`memory_alignment`**). The *primary* cache is organised in a set associative manner (represented using the WCET language construct **`cache_organisation`**) with 8 ways (represented using the WCET language construct **`number_of_ways`**). Cache lines to be replaced are chosen using the least recently used algorithm (represented using the WCET language construct **`cache_line_replacement`**).

```
number_of_cache_lines primary 1024
number_of_bytes_per_cache_line primary 32
cache_organisation primary set_associative
number_of_ways primary 8
memory_alignment primary true
cache_line_replacement primary lru
```

Another example, below, is the ARM 7 TLDI with a *primary* cache that has 256 cache lines each containing 32 bytes with each cache line being memory aligned. The *primary* cache is organised in a set associative manner with 4 ways. Cache lines to be replaced are chosen using the least frequently used algorithm.

```
number_of_cache_lines primary 256
number_of_bytes_per_cache_line primary 32
cache_organisation primary set_associative
number_of_ways primary 4
memory_alignment primary true
cache_line_replacement primary lfu
```

It is assumed that the PowerPC stores its instructions in memory conforming to the specification for *block1* and the ARM instructions in memory conforming to the specification for *block2*. (Note memory block identifiers are represented using WCET language construct **`memory_block_name`**) *block1* ranges from address 0 to 65535 with any cache miss suffering a penalty of 10 wait states – represented using WCET language construct **`memory_block_detail`**. *block2* ranges from address 65536 to 131071 with any cache miss suffering a penalty of 2 wait states.

```
memory_block_name block1 block2
memory_block_detail block1 0 65535 10
memory_block_detail block2 65536 131071 2
```

The evaluation platform used was a 500 MHz Pentium III running Linux with the WCET analysis tool written in Ada and compiled using GNAT. The analysis time quoted is the time to perform the whole WCET analysis and not just the instruction cache analysis. The table shows the benefit of performing cache analysis since in both cases the majority of instructions' accesses result in a cache hit. This result is clearly a function of the type of application being analysed. The results for the PowerPC take significantly longer to obtain. There are a number of reasons for this, including the relative size and complexity of the instruction sets and pipeline mechanisms.

Processor	No. of Accesses	No of Cache Misses	Analysis Time
PowerPC 603	2116	266	1 minute 47 seconds
ARM 7 TLDI	2108	264	1 minute 20 seconds

Table 1 - Results of the Evaluation

5 Conclusions

The paper has introduced a framework that has been developed to support the need for re-targetable WCET analysis that accounts for the features of today's microprocessors. The framework is based on target-independent models that are instantiated for specific platforms using a WCET language. More specifically, it has briefly described how it is possible to represent instruction caches in a re-targetable fashion. The paper then evaluates the approach by performing analysis on two different instruction sets, ARM and PowerPC, each with a different cache configuration. Being realistic, it is envisaged some cases could arise that the framework cannot support without minimal modification. However these are expected to be few in number and should require minimal change.

Future papers will describe the other parts of the framework that has already been produced.

6 References

[1] R. Chapman, *Static Timing Analysis and Program Proof*, Department of Computer Science, University of York, 1995.

[2] P. Puschner, A. Schedl, *Calculating the maximum execution times with linear programming techniques*, Institut fur Technische Informatik, Technische Univeristat Wien, 1995.

[3] S. Min et al, *An Accurate Worst Case Timing Analysis for RISC Processors*, IEEE Transactions on Software Engineering, 21(7), pp. 593-604, 1995..