A Definition and Classification of Timing Anomalies

Jan Reineke[1], Björn Wachter[1], Stephan Thesing[1], Reinhard Wilhelm[1],
Ilia Polian[2], Jochen Eisinger[2], and Bernd Becker[2]

[1] Saarland University
Im Stadtwald - Gebäude E1 3
66041 Saarbrücken, Germany
{reineke|bwachter|thesing|wilhelm}@cs.uni-sb.de

[2] Albert-Ludwigs-University
Georges-Köhler-Allee 51
79110 Freiburg, Germany
{polian|eisinger|becker}@informatik.uni-freiburg.de

Abstract. Timing anomalies are characterized by counterintuitive timing behaviour. A locally faster execution leads to an increase of the execution time of the whole program. The presence of such behaviour makes WCET analysis more difficult: It is not safe to assume local worst-case behaviour wherever the analysis encounters uncertainty.
Existing definitions of timing anomalies are either given as an intuitive description or do not cover all kinds of known timing anomalies. After giving an overview of related work, we give a concise formal definition of timing anomalies. We then begin to identify different classes of anomalies. One of these classes, coined Scheduling Timing Anomalies, coincides with previous restricted definitions.

Keywords: Timing analysis, Worst-case execution time (WCET), Timing anomalies, Abstraction

1 Introduction

The notion of timing anomalies was introduced by Lundqvist and Stenström in [LS99]. Intuitively, a timing anomaly is a situation where the local worst-case does not entail the global worst-case. For instance, a cache miss – the local worst-case – may result in a shorter execution time, than a cache hit, because of scheduling effects. See Figure 1 for an example. Shortening task A leads to a longer overall schedule, because task B can now block the "more" important task C. Analogously, there are cases where a shortening of a task leads to an even greater decrease in the overall schedule. Such effects are not relevant for timing analysis. We will not consider them in this paper.

Another example occurs with branch prediction. A mispredicted branch results in unnecessary instruction fetching that destroys the cache state. If the first instruction being fetched is a cache miss, the correct branch condition will be computed before more harm can be done by further fetches. Figure 2 illustrates this.

2 Existing Work on Timing Anomalies

The first paper remotely related to timing anomalies was written as early as 1969 by Graham [Gra69]. They show that a greedy scheduler can produce a longer schedule, if provided with

ECRTS 2006
6th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis
http://drops.dagstuhl.de/opus/volltexte/2006/671

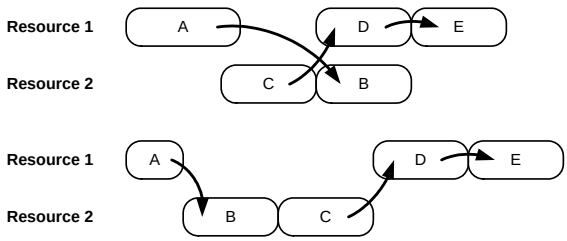

Fig. 1. Scheduling Anomaly

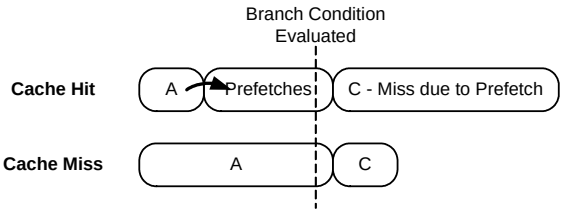

Fig. 2. Speculation Anomaly

shorter tasks, less dependencies, more processors, etc. They also give bounds on these effects, which are known as scheduling anomalies today. In their model all resources (processors) are identical though, which renders the given bounds useless for our purposes.

Lundqvist & Stenström first introduced timing anomalies in roughly the sense relevant for timing analysis. In their 1999 paper [LS99] they give an example of a cache miss resulting in a shorter execution time than a cache hit. A timing anomaly is characterized as a situation where a positive (negative) change of the latency of the first instruction by i cycles results in a global decrease (increase) of the execution time of a sequence of instructions. Situations where the local effect is even accelerated are also considered timing anomalies, i.e. the global increase (decrease) of the execution time is greater than the local change. We do not consider such cases here because they do not pose problems for timing analysis.

In his PhD thesis [Eng02] and in a paper with Jonsson [EJ02], Engblom briefly mentions timing anomalies. He translates the notion of timing anomalies of the Lundqvist/Stenström paper [LS99] to his model by assuming that single pipeline stages take longer, in contrast to whole instructions. Both Lundqvist and Engblom claim that, in processors containing in-order resources only, no timing anomalies can occur. This is not always true unfortunately, as corrected in Lundqvist's thesis [Lun02]. Schneider [Sch02] and Wenzel et al. [Wen03,WKPR05] note that if there exist several resources that have overlapping, but not equal abilities, timing anomalies can also occur.

Thesing [The04] discusses the Motorola ColdFire 5307, which contains a rather simple in-order pipeline that does not even have resources with overlapping abilities. He shows that the processor exhibits timing anomalies caused by its cache. The cache replacement policy of the ColdFire, Pseudo-Round Robin, causes these problems: In contrast to common replacement strategies, such as LRU or Pseudo-LRU, the effect of a cache miss on the cache state is sometimes different from that of a cache hit. While the cache miss obviously consumes a

longer processing time, it may result in a cache state that better suits the following code.

Wenzel, Kirner, Puschner, and Riedel [Wen03,WKPR05] give a necessary condition for timing anomalies, the *Resource Allocation Criterion*, short RAC. The RAC states that it has to be possible to create different schedules for at least one of the functional units of the processor for timing anomalies to be possible. Unfortunately, the criterion is based on a rather restricted definition of timing anomalies. he underlying assumption is that the latencies of subsequent instructions depend solely on the chosen schedule, i.e. which functional units are used. They are assumed to be independent of the initial latency difference. As we have observed in our introductory examples, this is overly optimistic. Both speculation and certain cache replacement strategies, like Pseudo-Round Robin violate this assumption.

3 Formal Definition

While the introductory examples give a rough intuition of what we consider a timing anomaly, they do not offer a concise formal definition. Let us identify important concepts that should flow into a formalization. It should not only cover presently known anomalies but be general enough to be valid also for future hardware features. This desired generality obviously requires a rather abstract approach.

Hardware Model A definition of timing anomalies has to take into account the hardware model. It has a great impact on the number and kind of such anomalies. For instance, out-of-order processors probably show more anomalies than simpler in-order machines.

Timing anomalies require choice, i.e. non-determinism in the analyzed model. In previous work on timing anomalies the different cases to compare, like cache hit or cache miss, came out of the blue.

Abstraction The reason for non-determinism in timing models is abstraction. Timing analysis usually only becomes feasible through abstraction. It enables us to deal with unknown input data and huge state-spaces. In return, abstraction has to give up some precision. Unknown information due to abstraction introduces non-determinism, where the underlying hardware model was fully deterministic. Depending on the precision of the abstraction different timing anomalies are conceivable.

Locality In most examples we have some intuition to what is the local worst-case. To identify a local worst-case formally we need a notion of locality. In literature, locality was usually not explicitly treated, but often implicitly fixed to the instruction level [LS99,Lun02]. Engblom [Eng02,EJ02] considered micro-operations (pipeline stages) to be the right granularity. We believe that micro-operations (like instruction fetch, execute, etc.) are indeed the right locality level. This is where timing differences first become visible.

Based on these observations we will now formally define timing anomalies. Our definition requires some notational prerequisites:

Definition 1 (Transition System). *A* transition system T *is a pair* $T = (S, R)$*, where* S *is a finite set of states and* $R \subseteq S \times S$ *is a transition relation. A* path π *in a transition system* $T = (S, R)$ *is a finite sequence of states, s.t.* $(\pi_i, \pi_{i+1}) \in R$ *for all* $i \in 0 \ldots |\pi| - 1$*. The* set of all paths *of a transition system* T *is denoted as* $\Pi(T)$*.*

A transition system can model the cycle-level behaviour of a computer architecture, i.e. a transition models the execution of one cycle. In contrast to other low-level hardware models,

such as Mealy- or Moore-automata, inputs and outputs are not explicitly modeled. This is not necessary in our context of timing analysis. Data and the program that is executed are modelled as part of the state.

As noted above, we consider micro-operations to be the right level to make local decisions, i.e. identify the local worst-case. The following definition of *locality constraints* enables us to do so.

Definition 2 (Locality Constraint). *A* locality constraint *l for a transition system $T = (S, R)$ is a convex predicate on S, i.e. l only holds on consecutive states in any path π through T. We assume that* locality constraints *model the sequence of states that is executing a micro-operation. We denote the* restriction *of π to l by $\pi_{|l}$, i.e. the restriction of the path π to the subpath of π in which l holds. Note, that this is still a (possibly empty) path. We denote the restriction of π to a set $\{l_1, \ldots, l_n\}$ of locality constraints by $\pi_{|l_1 \ldots l_n}$, i.e. the restriction of the path π to a sequence of states of T in which at least one of the predicates $l_1, \ldots, l_n$ holds. $\pi_{|l_1 \ldots l_n}$ is not necessarily a path.*

Definition 3 (Local Worst-Case Path). *Given a set of locality constraints $\mathcal{L}$ and a set of paths Π, a path $\pi \in \Pi$ is a* local worst-case path*, if and only if for every locality constraint $l \in \mathcal{L}$ and every path $\pi' \in \Pi$ it holds that if $\pi = \pi_{pre} \circ \pi_{|l} \circ \pi_{post}, |\pi_{|l}| > 0$ and $\pi' = \pi_{pre} \circ \pi'_{|l} \circ \pi'_{post}$ then $|\pi_{|l}| \geq |\pi'_{|l}|$.*
A path is called a non-local worst-case path *if it is not a* local worst-case path*.*

Definition 4 (Program). *A* program *(or a* control flow graph*) P is a directed graph $P = (V, E), E \subseteq V \times V$, in which the nodes V represent instructions, and an edge $(u, v) \in E$ represents flow of control from u to v.*
A sequence *σ through a program $P = (V, E)$ is a finite sequence of instructions, s.t. $(\sigma_i, \sigma_{i+1}) \in E$ for all $i \in 0 \ldots |\sigma| - 1$.*

We do not want to compare arbitrary paths through the transition system, but only those that correspond to the same path through the program. We can map paths in the transition system to paths through the program via a *labelling function*.

Definition 5 (Labelling Function). *Given a transition system $T = (S, R)$, a set of locality constraints $\mathcal{L}$, and a program $P = (V, E)$. A* Labelling Function *$\rho : V^* \to \mathcal{P}(\mathcal{L})$ assigns each finite sequence of instructions through the program P a set of locality constraints that corresponds to the execution of the respective micro-operations. A path π through T then corresponds to the sequence σ through P iff $\pi_{|l}$ is not empty for all $l \in \rho(\sigma)$ and $\pi_{|\rho(\sigma)}$ is equal to π.*
Given a set Π of paths through T, the subset of Π which corresponds to a given sequence σ w.r.t. a labelling function ρ is denoted as $\Pi|^{\rho}_{\sigma}$.

Definition 6 (Hardware Model). *A (possibly abstracted) hardware model C maps a program P to a transition system T, a set of locality constraints $\mathcal{L}$ on T, and a labelling function ρ that relates the states of the transition system with the instructions of the given program P.*

Note that a concrete hardware model is deterministic. Non-determinism – which is necessary for timing anomalies – is only introduced by abstraction. One can formally define the relation between concrete and abstract hardware models. For reasons of brevity we omit to provide such a definition.
Now, we are ready to define timing anomalies.

Definition 7 (Timing Anomaly). *A hardware model C exhibits timing anomalies, if there exists a program P with $C(P) = (T, \mathcal{L}, \rho)$, a finite sequence σ through P, and a non-local worst-case path $\pi \in \Pi(T)|^{\rho}_{\sigma}$, s.t. $|\pi| > |\pi'|$ for all local worst-case paths $\pi \in \Pi(T)|^{\rho}_{\sigma}$.*

Figure 3 illustrates the situation. At some analysis state, after executing π_{pre}, future execution is non-deterministic. To find the globally longest path we need to follow the non-local worst-case path $\pi_{|l}$ (it is not the longest path on locality constraint l).

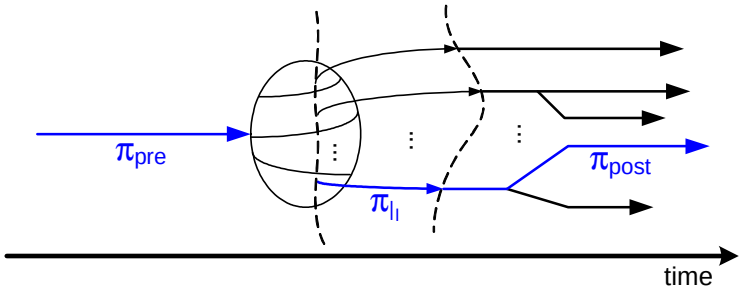

Fig. 3. Timing Anomaly Example

4 Classification

The above definition introduces timing anomalies in a rather abstract way. In the future, when confronted with a possible anomaly it will allow us to safely argue whether or not it constitutes a timing anomaly. This section aims at starting to clarify "what timing anomalies really are", by identifying different subclasses.

The idea is to readopt the view of timing anomalies from a scheduling perspective. In this setting, a set of tasks with dependencies and resource constraints describes the problem posed to the hardware. Tasks could be executions of micro-operations. Dependencies ensure that the micro-operations of a specific instruction can only be executed sequentially. Other dependencies model data dependencies in a program. Resources are stages in the pipeline like Instruction Fetch, Execute, or the different functional units of the processor. Now, we can distinguish at least three classes of timing anomalies (and possibly many more):

Scheduling Timing Anomalies We compare two task sets that differ only in the length of the "pivot" task. An example could be a cache hit vs. a cache miss. Figure 1 gives an example. The task sets differ only in the length of task A. Most timing anomalies dealt with in literature fall into this category. This kind of anomaly is well-known in the scheduling world, and has been extensively studied on various scheduling routines. One observation that can be made is that greedy schedulers, mimicked by timing analysis (and online schedulers, like modern processors, usually are greedy) are unable to prevent such anomalies in general.

Speculation Timing Anomalies Here, the difference is not confined to the length of the "pivot" task. The entire task set changes depending on this task. As an example see Figure 2. In both cases the processor is speculatively prefetching instructions. The local worst-case, a cache miss while fetching the first instruction, takes so much time that the branch condition can be evaluated, before more harm can be done to the cache by

further prefetches. Interestingly, the task set is influenced by previous decisions of the scheduler. Apparently, these interactions put these anomalies outside of the scope of scheduling theory. Note that the anomaly can occur even if the abstraction knows that the branch was mispredicted.

Cache Timing Anomalies These are anomalies induced by strange cache behaviour, as in the Pseudo-Round Robin cache replacement strategy employed in the ColdFire 5307. There, the non-local worst-case cache hit results in a different future cache state than the local worst-case cache miss. The difference in the cache state can then cause the cache hit branch to be stalled later on.

Interestingly, the latter two classes of anomalies can also happen on in-order architectures, as the ColdFire.

5 Conclusion

Timing anomalies result from complex interactions in modern processors and non-determinism introduced by abstraction. Their definition is a difficult task. We have given an overview of existing work on timing anomalies, and identified imprecisions and weaknesses. Notably, the restriction to what we call Scheduling Anomalies and the lack of formalization of locality. Based on these observations, we have given a concise formal definition, that is – unlike previous definitions – general enough to cover all known kinds of anomalies. Furthermore, we have begun to identify different classes of timing anomalies.

References

[EJ02] Jakob Engblom and Bengt Jonsson. Processor pipelines and their properties for static wcet analysis. In *EMSOFT '02: Proceedings of the Second International Conference on Embedded Software*, pages 334–348, London, UK, 2002. Springer-Verlag.

[Eng02] J. Engblom. Processor pipelines and static worst-case execution time analysis, 2002.

[Gra69] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.

[LS99] Thomas Lundqvist and Per Stenström. Timing anomalies in dynamically scheduled microprocessors. In *RTSS '99: Proceedings of the 20th IEEE Real-Time Systems Symposium*, page 12, Washington, DC, USA, 1999. IEEE Computer Society.

[Lun02] Thomas Lundqvist. *A WCET Analysis Method for Pipelined Microprocessors with Cache Memories*. PhD thesis, Chalmers University of Technology, Sweden, June 2002.

[Sch02] Jörn Schneider. *Combined Schedulability and WCET Analysis for Real-Time Operating Systems*. PhD thesis, Saarland University, Germany, December 2002.

[The04] Stephan Thesing. *Safe and Precise WCET Determination by Abstract Interpretation of Pipeline Models*. PhD thesis, Saarland University, Germany, July 2004.

[Wen03] Ingomar Wenzel. Principles of timing anomalies in superscalar processors. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2003.

[WKPR05] Ingomar Wenzel, Raimund Kirner, Peter Puschner, and Bernhard Rieder. Principles of timing anomalies in superscalar processors. In *Proc. 5th International Conference on Quality Software*, Sep. 2005.