PLRU Cache Domino Effects

Christoph Berg
Saarland University, Compiler Design Lab, Saarbrücken, Germany
cb@cs.uni-sb.de

June 9, 2006

Abstract

Domino effects have been shown to hinder a tight prediction of worst case execution times (WCET) on real-time hardware. First investigated by Lundqvist and Stenström, domino effects caused by pipeline stalls were shown to exist in the PowerPC by Schneider. This paper extends the list of causes of domino effects by showing that the *pseudo LRU* (PLRU) cache replacement policy can cause unbounded effects on the WCET. PLRU is used in the PowerPC PPC755, which is widely used in embedded systems, and some x86 models.

1 Introduction

Embedded systems play a key role in any modern product. When employed in safety-critical environments like airbag controllers in cars or fly-by-wire systems in air crafts, the timing must meet conditions imposed by the environment. The execution time of the tasks running on the embedded processor must always be lower than a given deadline. Timing analysis is used to derive an upper bound on the execution time, called *worst case execution time (WCET)*. Computing the WCET of a program requires upper bounds for the number of iterations of all loops in the program. On every path through the program, the worst case execution times for all instructions (or alternatively, basic blocks) has to be computed and added up. The longest of these paths is then called the critical path, and its maximum runtime is the WCET.

1.1 Timing Anomalies

In a simple world, timing analysis could just assume the local worst case for all instructions in a pipeline. When the cache state is not be precisely known, a memory access not classified by the abstract cache state as a cache hit would be considered a cache miss, variable-latency instructions would just take the longest time, etc. We could then add up all individual times and get a safe WCET bound.

Unfortunately, this is not safe. Out-of-order pipelines might reorder instructions such that an longer initial delay (e.g., a cache miss) could cause an overall *faster* completion of the whole sequence. Similarly, a speedup of an instruction can lead to a longer runtime for the whole sequence. This effect is called a *timing anomaly*, and was first described by Lundqvist and Stenström [3, 2].

While we can easily get a rough intuition about what a timing anomaly is, a general, hardware-independent definition is difficult, see [4] for a recent result. We will not detail the definition in this paper.

1.2 Domino Effects

A special case of timing anomalies is the *domino effect*, where – after a (possibly empty) prologue – a sequence of instructions is executed in a loop and depending on the initial state of a component (usually the pipeline, but also the cache, as we will see later), the loop body runtime will take different values without convergence. The presence of domino effects means that we cannot unroll a bounded (or even any) number of iterations of a loop and assume in the analysis that the remaining iterations behave the same. Schneider was able to demonstrate actual domino effects caused by the PowerPC PPC755 pipeline [5].

2 Domino Effects in Caches

We will look at cache behavior when a sequence of memory accesses is repeated indefinitely. We will only consider single cache sets.[1]

2.1 FIFO

FIFO caches require very little update logic, a round-robin counter points the next way to be replaced. The downside is that this causes domino effects.

Figure 1 shows a 2-way FIFO cache with the access sequence a-b-c. Starting with an empty cache leads to a repeated cache content b-c at the end of each iteration, where each cycle has 3 cache misses (marked by ‘x’ in the figure). Starting with c-a in the cache leads to a cycle over 2 iterations with a total of 3 cache misses, alternatingly 1 and 2 per iteration.[2]

[1] There are cache architectures where the sets are not independent, but that only makes timing effects more unpredictable.

[2] This is the same example as in [2].

	. .		c a	
a:	a .	x	c a	
b:	b a	x	b c	x
c:	c b	x	b c	
a:	a c	x	a b	x
b:	b a	x	a b	
c:	c b	x	c a	x
a:	a c	x	c a	
b:	b a	x	b c	x
c:	c b	x	b c	
a:	a c	x	a b	x
b:	b a	x	a b	
c:	c b	x	c a	x

Figure 1: 2-way FIFO cache, empty cache is worst case

2.2 LRU

The LRU update logic is complex, and LRU caches with more than 4 ways are rare in practice.

LRU caches do not exhibit domino effects. When iterating a over sequence of instructions, it is easy to see that the cache contents at the end of the first iteration are the same as at the end of every other iteration. This "memory-less" property of LRU makes the cache analysis both easy and precise [1].

2.3 PLRU

Pseudo LRU replacement is a variant of LRU where the "ages" of the lines in the cache are not linearly ordered, but arranged in a tree (see Fig. 2). The advantage is that this needs fewer state bits and hence needs a less complex update logic. PLRU has an average case performance comparable to LRU, but the worst case performance, which matters for the WCET prediction, is worse.

Heckmann has shown that only 3 to 4 ways of an 8-way PLRU cache can be tracked in cache analysis [1]. This article adds domino effects to the list of problems with analyzing PLRU caches.

PLRU definition. PLRU maintains a tree of cache ways. Every inner tree node has a bit pointing to the subtree that contains the leaf to be replaced next. Figure 2 shows a 4-way example. In the left picture, the three state bits b_0, b_1, b_2 point to the second way to be replaced next. In practice, 8-way PLRU is used. The replacement logic is the same, with 7 state bits, and an additional level in the tree. For simplicity, we consider 4-way in this article. Note that 2-way PLRU is equivalent to 2-way LRU.

On a cache update, and similarly on a cache hit, the state bits on the path leading to the accessed way will be flipped, i.e. making them pointing away from that way. The right picture shows the cache state after the replacement of 'b' by 'e'. The path to the second way consists of bits b_0 and b_1, so these have been flipped.

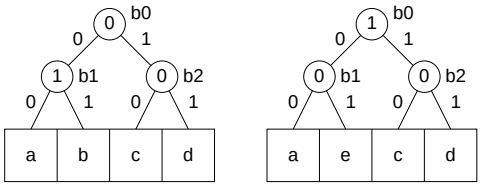

Figure 2: 4-way PLRU

	a b c d	0 0 0	
a:	a b c d	1 1 0	
e:	a b e d	0 1 1	x
a:	a b e d	1 1 1	
f:	a b e f	0 1 0	x
a:	a b e f	1 1 0	
g:	a b g f	0 1 1	x
a:	a b g f	1 1 1	
e:	a b g e	0 1 0	x
a:	a b g e	1 1 0	
f:	a b f e	0 1 1	x
a:	a b f e	1 1 1	
g:	a b f g	0 1 0	x
a:	a b f g	1 1 0	
e:	a b e g	0 1 1	x
a:	a b e g	1 1 1	
f:	a b e f	0 1 0	x
a:	a b e f	1 1 0	
g:	a b g f	0 1 1	x

Figure 3: 4-way PLRU cache, b is never evicted

PLRU examples. Figure 3 shows that PLRU does not use the cache optimally. The access sequence a-e-a-f-a-g contains only 4 distinct elements, yet 4-PLRU misses for every second access when started with a-b-c-d in the cache.[3]

Figure 4 shows that a PLRU cache can take several iterations to stabilize. The fourth iteration is the first that exhibits 3 misses as all the following iterations do.

Figure 5 shows an example with the access sequence i-f-e-g-i-e-b. The left sequence starts with a-b-c-d in the cache, the right one with an empty cache. For the first two iterations, both caches miss on the same accesses (the cache content is even the same at the end of the first iteration, though in a different order). Starting from the third iteration, the left cache misses 3 times per iteration, the right cache only 2 times.

2.4 Avoiding Domino Effects

There is no generally safe initial cache state in the context of domino effects as any state can cause non-converging loop runtimes. With a given program and initial state, we can prove the presence or absence of effects, but in many embedded system context, modifying the program is infeasible, and the initial state can also be unknown or undefined. Modifying the

[3] Example similar to Fig. 3 in [1].

	b e c a	0 0 0	
c:	b e c a	0 0 1	
a:	b e c a	0 0 0	
b:	b e c a	1 1 0	
e:	b e c a	1 0 0	
b:	b e c a	1 1 0	
c:	b e c a	0 1 1	
d:	b d c a	1 0 1	**x**
c:	b d c a	0 0 1	
a:	b d c a	0 0 0	
b:	b d c a	1 1 0	
e:	b d e a	0 1 1	**x**
b:	b d e a	1 1 1	
c:	b d e c	0 1 0	**x**
d:	b d e c	1 0 0	
c:	b d e c	0 0 0	
a:	a d e c	1 1 0	**x**
b:	a d b c	0 1 1	**x**
e:	a e b c	1 0 1	x
b:	a e b c	0 0 1	
c:	a e b c	0 0 0	
d:	d e b c	1 1 0	**x**
c:	d e b c	0 1 0	
a:	d a b c	1 0 0	x
b:	d a b c	0 0 1	
e:	e a b c	1 1 1	x
b:	e a b c	0 1 1	
c:	e a b c	0 1 0	
d:	e d b c	1 0 0	x
c:	e d b c	0 0 0	
a:	a d b c	1 1 0	x
b:	a d b c	0 1 1	
e:	a e b c	1 0 1	x
b:	a e b c	0 0 1	
c:	a e b c	0 0 0	
d:	d e b c	1 1 0	x

Figure 4: 4-way PLRU cache, convergence in the forth cycle

compiler not to produce code exhibiting domino effects seems impossible, given the simplicity of our FIFO examples.

At first glance, it might seem that for PLRU, we can improve our knowledge of the cache state by not just taking into account the ordering of the blocks in the cache but also include the tree state bits. However, we can reorder the ways such that the tree bits always point left, and modify the cache update accordingly. As with FIFO caches, we will have domino effects in this simplified cache.

3 Summary

Timing anomalies and domino effects cause the complexity of timing analysis to grow. Several causes are known, this article adds PLRU caches to the list.

To achieve better predictability, some embedded system designers lock all but two PLRU ways, leaving a 2-way LRU subset. We are currently working on cache models that will hopefully be able to extract more precise information from a non-locked cache despite the presence of timing anomalies.

	a b c d	0 0 0		. . . .	0 0 0	
i:	i b c d	1 1 0	x	i . . .	1 1 0	x
f:	i b f d	0 1 1	x	i f . .	1 0 0	x
e:	i e f d	1 0 1	x	i f e .	0 0 1	x
g:	i e f g	0 0 0	x	i f e g	0 0 0	x
i:	i e f g	1 1 0		i f e g	1 1 0	
e:	i e f g	1 0 0		i f e g	0 1 1	
b:	i e b g	0 0 1	x	i b e g	1 0 1	x
i:	i e b g	1 1 1		i b e g	1 1 1	
f:	i e b f	0 1 0	x	i b e f	0 1 0	x
e:	i e b f	1 0 0		i b e f	0 1 1	
g:	i e g f	0 0 1	x	i g e f	1 0 1	x
i:	i e g f	1 1 1		i g e f	1 1 1	
e:	i e g f	1 0 1		i g e f	0 1 1	
b:	i e g b	0 0 0	x	i b e f	1 0 1	x
i:	i e g b	1 1 0		i b e f	1 1 1	
f:	i e f b	0 1 1	**x**	i b e f	0 1 0	
e:	i e f b	1 0 1		i b e f	0 1 1	
g:	i e f g	0 0 0	x	i g e f	1 0 1	x
i:	i e f g	1 1 0		i g e f	1 1 1	
e:	i e f g	1 0 0		i g e f	0 1 1	
b:	i e b g	0 0 1	x	i b e f	1 0 1	x
i:	i e b g	1 1 1		i b e f	1 1 1	
f:	i e b f	0 1 0	**x**	i b e f	0 1 0	
e:	i e b f	1 0 0		i b e f	0 1 1	
g:	i e g f	0 0 1	x	i g e f	1 0 1	x
i:	i e g f	1 1 1		i g e f	1 1 1	
e:	i e g f	1 0 1		i g e f	0 1 1	
b:	i e g b	0 0 0	x	i b e f	1 0 1	x

Figure 5: 4-way PLRU cache, domino effect starting in the third iteration

References

[1] R. Heckmann, M. Langenbach, S. Thesing, and R. Wilhelm. The influence of processor architecture on the design and the results of WCET tools. *IEEE Proc.*, 91(7), July 2003.

[2] T. Lundqvist. *A WCET Analysis Method for Pipelined Microprocessors with Cache Memories*. PhD thesis, School of Computer Science and Engineering, Chalmers University of Technology, 2002.

[3] T. Lundqvist and P. Stenström. Timing anomalies in dynamically scheduled microprocessors. Number 20 in IEEE Real-Time Systems Sym. (RTSS'99), Dec. 1999.

[4] J. Reineke, B. Wachter, S. Thesing, R. Wilhelm, I. Polian, J. Eisinger, and B. Becker. A definition and classification of timing anomalies. 6th Intl Workshop on Worst-Case Execution Time (WCET) Analysis, July 2006.

[5] J. Schneider. *Combined Schedulability and WCET Analysis for Real-Time Operating Systems*. PhD thesis, Universität des Saarlandes, Dec. 2002.