WCET Analysis of Shared Caches in Multi-Core Architectures using Event-Arrival Curves

Thilo L. Fischer
Institute of Embedded Systems
Hamburg University of Technology
Hamburg, Germany
thilo.leon.fischer@tuhh.de

Heiko Falk
Institute of Embedded Systems
Hamburg University of Technology
Hamburg, Germany
heiko.falk@tuhh.de

***Abstract*—We propose a novel analysis approach for shared LRU caches to classify accesses as definitive cache hits or potential misses. In this approach inter-core cache interference is modelled as an event stream. Thus, by analyzing the timing between subsequent accesses to a particular cache block, it is possible to bound the inter-core interference. This perspective allows us to classify accesses as cache hits or potential misses using a data-flow analysis. We compare the performance of the presented approach to a partitioning of the shared cache.**

***Index Terms*—shared cache, WCET analysis, multi-core**

I. Introduction

In a multi-core system, tasks are subject to inter-core interference due to the concurrent execution of multiple tasks. Specifically, data stored in a shared cache may be evicted by interfering memory accesses. As a consequence of multiple tasks competing for cache space, the worst-case execution time (WCET) is negatively impacted. To derive safe estimations of a task's WCET, the inter-core interference has to be taken into account. The standard method [1] of accounting for shared cache interference is to assume that all conflicting cache blocks may be accessed at any time by an interfering task. This is obviously pessimistic. An alternative approach is to eliminate inter-core interference by isolating tasks from each other. This can be achieved by locking the contents of the cache or by partitioning the cache to enforce isolation [2]. Another approach is to bypass the shared cache for requests on infrequently used data to avoid altering the cache's contents [3]. However, implementing such approaches requires hardware or software support and, in the case of cache partitioning, reduces the maximal cache size available to each task. Thus, to fully benefit from the presence of shared caches without restrictions on cache usage, a detailed analysis of inter-core interference is required.

We propose a novel analysis approach for inter-core interference on set-associative shared caches using the LRU replacement policy. To quantify inter-core interference, we view cache accesses issued by each core as an event stream. Thus, we can examine the inter-arrival time of cache access events. This perspective essentially induces a *time-to-live* (TTL) for information stored in the shared cache. The TTL of a cache block corresponds to the time frame in which interfering tasks can not issue a sufficient number of conflicting accesses to cause its eviction. Consequently, if a block is accessed before its TTL has expired, the access will result in a cache hit.

The key contributions of this paper are:

- We describe a novel perspective on inter-core interference for shared caches by interpreting cache accesses as event streams.
- We provide a cache hit classification criterion for shared caches accessed via a TDMA bus based on event-arrival curves.
- The technique is scalable and attains WCET performance similar to a partitioned cache.

II. Event-Arrival Curve Perspective

The system architecture considered in this paper consists of multiple cores with private caches, which are connected to a shared cache via a TDMA bus. The associativity of the shared cache is denoted by $\mathcal{A}$. Each core processes a single task $\tau \in \mathrm{T}$. We quantify inter-task interference by determining the event-arrival curves for cache access events which may evict data from the cache. The number of accesses to pair-wise different cache blocks issued from task $\tau \in \mathrm{T}$ during a time frame of Δt cycles is denoted by the event-arrival curve $\eta_\tau : \mathbb{N} \to \mathbb{N}$. Deriving the curve $\eta_\tau(\Delta t)$ from a given control-flow graph involves finding the path containing the maximal number of interfering accesses with duration $\leq \Delta t$. This problem can be solved using an IPET model. Due to space constraints, the full model is not included here. It is then possible to compute the interference caused by a task during a single TDMA slot (consisting of S cycles) by evaluating $\eta_\tau(S)$. We can thus make the following observation:

Observation 1. *The cumulative interference a task τ experiences over the course of j TDMA slots is bounded by $\gamma_\tau : \mathbb{N} \to \mathbb{N}$:*

$$\gamma_\tau(j) = j \cdot \sum_{\phi \neq \tau} \eta_\phi(S) \tag{1}$$

Note that this is a generalization of way-based cache partitioning. For the purpose of cache hit classifications, the reduced associativity in an equally partitioned cache can be represented using a constant interference function:

$$\gamma_\tau^{Part} = \sum_{\phi \neq \tau} \frac{\mathcal{A}}{|\mathrm{T}|} = \frac{|\mathrm{T}| - 1}{|\mathrm{T}|} \cdot \mathcal{A} \tag{2}$$

978-3-9819263-7-8/DATE23/© 2023 EDAA

III. Cache Hit Classification

We now construct a cache hit classification criterion for LRU caches based on the duration (and thus the number of interfering TDMA slots) between accesses to the same cache block. The eviction distance ξ of a cache block is the minimal number of interfering accesses to cause its eviction. We call a path π between two subsequent accesses to the same cache block with $\xi(\pi) > 0$ a *potential-hit* path.

To capture the inter-task interference along a potential-hit path π, the maximal number of TDMA slots granted to interfering tasks during its execution has to be determined. The path duration $\delta(\pi)$ can be derived from a WCET analysis. Note that $\delta(\pi)$ depends on the initial TDMA offset of π computed during the WCET analysis. For a different initial offset, the path duration may diverge from $\delta(\pi)$. However, this divergence is limited to $|\mathrm{T}| \cdot S$ cycles due to the *offset relocation* Lemma from Kelter et al. [4]. These considerations give rise to an upper limit:

Lemma 1. *While executing a path π, the number of TDMA slots granted to an interfering core is limited by:*

$$J(\pi) = \left\lfloor \frac{\delta(\pi) + |\mathrm{T}| \cdot S + (S-1)}{|\mathrm{T}| \cdot S} \right\rfloor \tag{3}$$

Combining Lemma 1 and the cumulative interference function γ_τ (1) allows us to construct an *always-hit* classification criterion.

Theorem 1. *An access will always result in a cache hit if it may only be reached by traversing potential-hit paths π satisfying:*

$$\gamma_\tau(J(\pi)) < \xi(\pi) \tag{4}$$

It is now possible to check whether a particular cache access will result in a cache-hit. The paths leading to an access have to be analyzed to determine whether condition (4) holds. This can be done using a data-flow analysis. The formal definition of the data-flow analysis is not shown here due to space constraints.

IV. Evaluation

We evaluated the performance of the presented analysis approach using the WCC compiler [5]. We considered 10 dual-core and 10 quad-core systems. For each system, we randomly assigned a task from the EEMBC AutoBench 1.1 suite [6] to each core. The access timings for an L1-Hit/L2-Hit/L2-Miss were set to 1/8/20. The L1 cache was 512 bytes, direct-mapped, and block size of 16 bytes. The L2 cache was 4KB to 16KB, 8-way (dual-core) / 16-way (quad-core) associative, and block size of 32 bytes. The TDMA slots were 80 cycles long. The evaluations were conducted on an Intel Xeon Server containing 46 cores at 3.2GHz. All analyses were configured to only use a single processor core. Creating the ILPs from Section II and solving them took only 6 minutes on average for quad-core systems. The DFA to classify accesses took < 1 second for every system. We compared the computed WCET values to an equally partitioned cache. The results are shown in Fig. 1. The y-axis shows the relative WCET using a shared cache compared to the partitioning. The WCET is evaluated for each task and grouped for each system configuration. It can be seen that the performance of a shared cache analyzed using event-arrival curves tracks closely to the partitioned cache. For dual-core systems the median for all configurations is only 1.04. Overall, for quad-core systems the median WCET is 1.11. Notably, sharing the cache instead of partitioning it yielded WCET reductions of up to 26%. In that case, leveraging the timing information of cache access patterns allowed the analysis to classify 28% more accesses as cache hits than for the partitioned cache configuration.

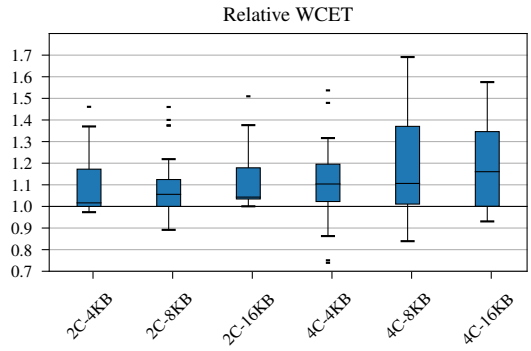

Fig. 1. Relative WCET using the event-arrival curve based classification in relation to a partitioned cache.

V. Conclusion

We presented a novel analysis approach to quantify inter-core interference on shared caches using the LRU replacement policy in multi-core architectures. By leveraging timing information and the properties of TDMA arbitration, we formulated a cache hit classification criterion for accesses to the shared cache. The evaluation showed that the event-arrival perspective reduced the unpredictability inherent to shared caches. Additional experiments could be done to further evaluate the performance of the analysis. In future work, the technique could be applied to more complex systems, e.g. multiple tasks being scheduled to run on each core or different bus arbitration techniques.

References

[1] Y. Liang, H. Ding, T. Mitra, A. Roychoudhury, Y. Li, and V. Suhendra, "Timing Analysis of Concurrent Programs Running on Shared Cache Multi-Cores," *Real-Time Systems*, vol. 48, no. 6, pp. 638–680, 2012.

[2] V. Suhendra and T. Mitra, "Exploring Locking & Partitioning for Predictable Shared Caches on Multi-Cores," in *Proc. of DAC*, 2008, pp. 300–303.

[3] D. Hardy, T. Piquet, and I. Puaut, "Using Bypass to Tighten WCET Estimates for Multi-Core Processors with Shared Instruction Caches," in *Proc. of RTSS*, 2009, pp. 68–77.

[4] T. Kelter, H. Falk, P. Marwedel, S. Chattopadhyay, and A. Roychoudhury, "Static Analysis of Multi-Core TDMA Resource Arbitration Delays," *Real-Time Systems*, vol. 50, no. 2, pp. 185–229, 2014.

[5] H. Falk and P. Lokuciejewski, "A Compiler Framework for the Reduction of Worst-Case Execution Times," *Real-Time Systems*, vol. 46, no. 2, pp. 251–300, 2010.

[6] The Embedded Microprocessor Benchmark Consortium. About the EEMBC AutoBench™ Performance Benchmark Suite. Accessed 2022-07-05. [Online]. Available: https://www.eembc.org/autobench/