Time-Sensitive Shared Caches interferences Analysis in Multi-Core Architectures for WCET

Yixuan Zhu, Wenqi Lou, Xianglan Chen, Chao Wang and Xi Li

July 15, 2024

Time-sensitive Shared Caches Interferences Analysis in Multi-Core Architectures for WCET

Yixuan Zhu, Wenqi Lou, Xianglan Chen, Chao Wang, and Xi Li(✉)

University of Science and Technology of China, Hefei 230026, China
zhuyixuan@mail.ustc.edu.cn, {louwenqi,xlanchen,cswang,llxx}@ustc.edu.cn

Abstract. Shared last-level caches in multicore architectures cause memory accesses to interfere with others, resulting in additional access latency. So computing the worst-case execution time (WCET) of a program necessitates an analysis of the inter-core interference, which requires determining when the accesses occur. Current approaches directly use the execution time of a program as the life cycle of its internal accesses for scalability, which can lead to a significant overestimation of the interferences. In this paper, we propose a time-sensitive shared cache interference analysis method. It estimates the execution time of a basic block relative to the start of the program based on the execution path of the program and combines it with an approximation model as the life cycle of the accesses within the block, which can effectively exclude impossible interferences and averagely tighten the WCET estimation by 14.5%.

Keywords: WCET Analysis · Shared Cache · Multi-Core · Real Time.

1 INTRODUCTION

Competition for resources by concurrent execution tasks in a multicore system can interfere with others [3, 5], making WCET analysis difficult [4]. For shared caches, it can evict data that was originally located in the shared cache, generating additional access latency. Therefore, interference needs to be analyzed, focusing on whether memory accesses on different cores will map to the same cache set at the same time period. Typically, we should capture the request time of memory accesses (i.e., the inter-core context) and then compute the number of interferences. However, the uncertainty of program inputs leads to multiple possibilities for the execution time of the access. The current work [1, 2] assumes that if the execution times of two programs located in different cores overlap, respectively, the accesses from these two programs will interfere as long as they are mapped to the same cache set, and there is no judgment on whether or not they will be executed at the same time, which can lead to over-estimation of the interferences.

In this paper, we propose a time-sensitive shared cache interference analysis. It utilizes the execution time of a basic block as the fine-grained time when its internal accesses occur, which can effectively exclude impossible interference from time. Treating loops as virtual nodes, we hierarchically process control flow from inner to outer loops and construct an approximate model describing the execution time of basic blocks within loops to avoid exhaustive path searching. For interference judgment, we use the base time and offset time in a two-stage approach to sequentially determine whether the accesses will occur simultaneously and then if they will be mapped to the same cache set.

2 METHODOLOGY

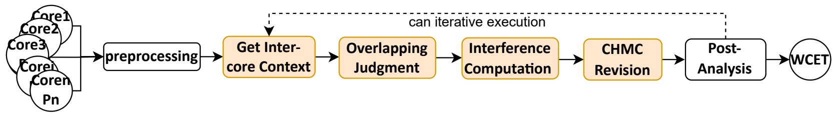

Fig. 1. Overview of our analysis framework

Our framework inputs binaries of multiple programs and hardware features that output the WCETs. It mainly consists of preprocessing, getting the inter-core context, interference judgment, shared cache revision, and post-processing. The preprocessing does not consider resource sharing and directly uses a single-core multilevel cache analysis to obtain the cache hit/miss classification (CHMC) of accesses. Then, the estimate of the best execution cost BC for basic blocks assumes that the access shared caches are all hits; for the worst cost WC assumes all non-hits. Get the context by the execution cost and use it to compute the number of interferences. Then, revise the CHMC of the shared cache. For shared cache access m whose CHMC is AH or PS, its CHMC is corrected to NC if its interference quantity $|\mathcal{M}_c(m)|$ satisfies $N - age(m) < |\mathcal{M}_c(m)|$. N is the correlation and $age(m)$ is the max age of LRU strategy. Post-processing utilizes the interference analysis results to obtain the WCET using existing techniques.

2.1 Get Inter-core Context

Obtaining the inter-core context is equivalent to computing the execution time of a basic block relative to the start of the system ($BRSTime$), described using the ideas of offset time and base time. As shown in Fig. 2(a), for basic blocks outside loops, like $A10$, their $BRPTime$ is directly represented using an interval consisting of their earliest start time and latest end time. For basic blocks inside loops, like $A2$, $A6$, etc., their $BRPTime$ is a sequence of intervals, which is derived based on their offset time $BBOTime$ relative to the loop directly in and the relative time of the loop k relative to its parent loop $LPBTime$ Eqn. 1.

$$\text{LPBTime(k)} = \begin{cases} [\text{BBESOT(k)}, \text{BBLEOT(k)}], & k \text{ } is \text{ } not \text{ } OutestLoop \\ \text{LPBtime(k)} \times \text{BBOTime(k)}, & otherwise \end{cases} \tag{1}$$

The time of the loop is described by the interval sequences, which are represented by an iteration-dependent approximation model. The ith element of the sequence is the interval consisting of the earliest start and the latest execution end of the ith iteration of the loop, as shown in the Eqn. 2. Where $SC_Y(k)$ and $LC_Y(k)$ denote the min and max execution cost from the head node to node k, and when k is t, represent the tail node. This way only one iteration of the computation needs to deduce all other iterations.

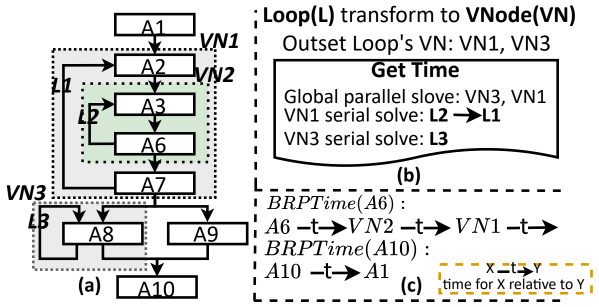

Fig. 2. Loops transformed into virtual nodes and analysis flows for the entire program

$$[(i-1) * SC_Y(t) + SC_Y(k), (i-1) * LC_Y(t) + LC_Y(k) + WC_k], i \in [1, n_Y - 1] \quad (2)$$

The main function can be viewed as a loop containing only one iteration to obtain SC and LC in a unified way when calculating. As shown in Fig. 2(b), inverse loop nesting and inverse program order are required for processing. The shortest path algorithm is used to calculate the longest time cost from the head node of each loop to each other node, and similarly, the shortest time cost. A loop is treated as a virtual node after being processed. So, for a program containing P loops (each loop has V basic blocks and E edges), the time complexity is $O(PEVlgV)$, and the space complexity is $O(PV)$.

2.2 Interference Judgement

To minimize repeated computations, we predetermine the overlap of basic blocks. Whenever two intervals from two separate interval sequences of the basic block are executed overlappingly, it is assumed that the two basic blocks will be executed overlappingly. For two sequences with m and n intervals, we design an algorithm with a time complexity of $O(m + n)$ to determine if they will overlap, which first merges the overlapping intervals in a sequence and then determines if there exists a pair of overlapping intervals by the upper and lower bounds.

The $BRPTime$ derived from the recursive may contain many intervals, which take much time as direct input to the above algorithm. We use the following two-stage approach for overlap determination. First, we use the $BRPTime$ of the virtual node corresponding to the outermost loop (non-main) of the basic block. Then, we get its absolute $BRPTime$ using the recursive formula for the judgment, and the granularity of the expansion will depend on the number of nested layers of the loop and the number of iterations. Record the basic blocks that may overlap the execution of each basic block. All accesses in basic blocks that overlap execution with the basic block where the access behavior m is located, if they are mapped to the same cache set as m, the $|\mathcal{M}_c(m)|$ is added.

3 EVALUATION

To evaluate our method, we integrate it into the WCET analysis framework Chronos and compare it with the All-Interference (All-INTERF) method [2]. A system with two cores is used for the experiments, which share an L2 instruction cache (4-KB size, 4-way associate, 32 bytes block size, hit requires 6 cycles).

Each core has a private L1 instruction cache (2-way with block size 32 bytes; hit requires 1 cycle), and the experiments were conducted using 512B and 256B sizes, respectively. Memory access requires 30 cycles. We used the Mälardalen WCET benchmark suite for testing. The relative WCET is used as an evaluation metric, which refers to the ratio of the WCET obtained by our method to the All-INTERF method, i.e., $WCET_{Our}/WCET_{All}$.

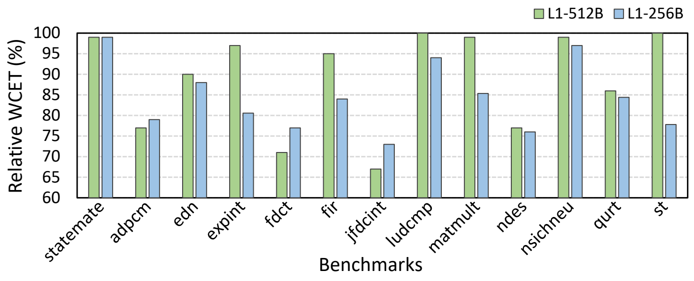

Fig. 3. Relative WCET for L1 cache with 512B and 256B size

From Fig. 2, we can see that average relative WCET is 87%(L1 512B) and 84%(L1 256B), which shows that our method can reduce the estimation of WCET. This is because we reduce the overestimation of the interferences; for example, when the benchmark ndes is interfered with edn, we estimate 217 fewer interferences. statement, ludcmp, etc., have little improvement because they have a small probability of accessing shared caches or hits, which inherently have less room for improvement. And the average analysis time of our method increases by only 130% compared to All-INTERF method due to interference judgement.

4 CONCLUSION

In this paper, we propose a novel interference analysis method for shared caches, which utilize the fine-grained time information to reduce the overestimation of the interference. The result shows that it can reduce the average WCET by 14.5% relative to the standard analysis method for the instruction cache.

Acknowledgments. This work was supported in part by the National Key R&D Program of China under Grants 2022YFB4501600 and 2022YFB4501603.

References

1. Fischer, T.L., Falk, H.: Analysis of shared cache interference in multi-core systems using event-arrival curves. In: Proc. of RTNS (2023)
2. Hardy, D., et al.: Using bypass to tighten wcet estimates for multi-core processors with shared instruction caches. In: Proc. of RTSS. IEEE (2009)
3. Lou, W., et al.: Unleashing network/accelerator co-exploration potential on fpgas: A deeper joint search. TCAD (2024)
4. Maiza, C., et al.: A survey of timing verification techniques for multi-core real-time systems. ACM Computing Survey (2019)
5. Wang, C., et al.: Wookong: A ubiquitous accelerator for recommendation algorithms with custom instruction sets on fpga. TC (2020)