Synergetic effects in cache related preemption delays

Jan Staschulat, Rolf Ernst
Technical University of Braunschweig
Institute of Computer and Communication Network Engineering (IDA)
D-38106 Braunschweig, Germany
{staschulat,ernst}@ida.ing.tu-bs.de

Abstract

Cache prediction for preemptive scheduling is an open issue despite its practical importance. First analysis approaches use simplified models for cache behaviour or they assume simplified preemption and execution scenarios that seriously impact analysis precision. We present an analysis approach for m-way associative caches which considers multiple executions of processes and preemption scenarios for static priority periodic scheduling. The results of our experiments show that caches introduce a strong and complex timing dependency between process executions that are not appropriately captured in the simplified models.

1. Introduction and motivation

Caches are needed to increase processor performance but they are hard to use in real-time systems because of their complex behaviour. While it is already difficult to determine cache behaviour for a single process, it becomes really complicated if preemptive process scheduling is included. Preemptive process scheduling means that process execution can be interrupted by higher priority processes. In this case, cache improvements can be strongly degraded by frequent exchange of cache blocks.

There are several approaches to make caches more predictable and efficient. One approach is to partition the cache sets and to reserve these partitions for individual processes. This has been investigated in [8]. The advantage is that cache lines do not have to be reloaded after interrupts and between consecutive executions of the same process. Also, cache behaviour becomes (partly) orthogonal for processes and therefore more predictable. In [4] process layout techniques are suggested which aim at minimising the inter-process interference in the instruction cache. Another approach [10] is to lock frequently used cache lines. While cache partition and lock strategies are certainly a very useful add-on to improve cache predictability and efficiency, they do not solve the general cache behaviour problem which is critical for larger systems of processes.

Simplified approaches extend the known RMA with fixed context switch costs [1], while recent approaches use data flow analysis of the preempted and preempting process to bound the number of replaced cache blocks [7] [9]. However, these approaches model only a single process activation assuming an empty cache at process start neglecting that cache blocks (CB) might be available for later executions. Pre-runtime scheduling heuristics which take the effects of process switching on processor cache into account have been presented in [6]. However, only non-preemptive scheduling based on earliest deadline first strategies is considered which is much easier than the preemptive case. If a process is preempted several times the total number of cache blocks replaced drops, because a cache block of preempted process can only be replaced once. Such preemption scenarios are not considered in classical CRPD analysis.

This paper is organized as follows. Section 2 reviews related work. In Section 3 we present a new analysis approach to determine the cache related preemption delay (CRPD) for m-way associative instruction caches which considers multiple executions of processes as well as preemption scenarios. We show the results in Section 4, before we conclude in Section 5.

2. Related work

The data flow analysis by [7] determines the CRPD when a process τ_1 preempts process τ_0 by intersecting the number of useful CBs of τ_0 and with the number of used CBs of τ_1 assuming an empty cache at process start. Then, a complex analysis follows analysing all possible combinations of preemptions. The number of preemptions is determined by integer linear programming (ILP) and process phasing based on worst case and best case response time (BCRT) of processes. However, the BCRT analysis is a complicated problem where only approximative solutions have been proposed for the general case [5]. Multiple process executions is not considered and multiple preemptions are simplified by multiplying the maximum CRPD cost by the number of preemptions. For direct mapped instruction caches [9] refine the data flow analysis of [7] by modeling the cache content as a *state* instead of a set. All possible cache states of the preempting and preempted process are intersected to find the maximum CRPD. Preemption scenarios are not considered and an empty cache is assumed at process start.

Current approaches either model only the number and cost of preemptions for a single process execution or the cache effects of multiple process execution without preemptions, but none models both. Only the combination provides sufficient accuracy as we will see in our experiments.

3. Refined approach

Our CRPD analysis considers multiple activations of processes and preemption scenarios. A preemption scenario consists of one preempted process τ_i and of a process τ_j which preempts τ_i during a single execution n times. We represent the process by its control flow graph (CFG), where each node is a basic block and assume a preemption point at each node.

To reduce the exponential combinations of n preemption points (CFG nodes) we use a branch and bound algorithm. It first determines the n most expensive preemption points by analyzing the cost of a single preemption at each node. Then it continues to compare the cost of the combinations of two nodes until the cost C_n for a preemption scenario with n nodes is found. The algorithms bounds at a combination if the current cost plus the cost for future preemptions is smaller than C_n. For sequential code this is straight forward, but for n preemtions within loop body this modeling would lead to unrolling the loop. Therefore we abstract from preemption points of different loop iterations by estimating the preemption point with the maximum cost of the loop body and multiplying it by n. This estimation is exact if $I \geq n$, where I is the maximum loop iterations and conservative if $I < n$. The cost C_n of a preemption scenario is calculated by considering the useful CBs of the preempted process τ_i and the used CBs of the preempting process τ_j. For the analysis of each preemption cost a data flow model is needed.

We base our analysis for m-way associative caches on the cache state analysis of [9]. But define a cache state for each cache set with m blocks. A reaching cache state RCS_B at a basic block B of a process is the set of possible cache states when B is reached via any incoming program path. The live cache states at a basic block B, denoted LCS_B, are the possible first memory reference to CBs via any outgoing program path from B. A least fixed point data flow algorithm computes the values of these sets. The cache behavior including replacement strategy (e.g. LRU) is simulated by preloading the cache state RCS of the predessor node and executing the instruction sequence of basic block B by cache simulation. The resulting cache state represents the RCS of basic block B. The intersection of RCS and LCS is the set of useful CBs at basic block B. The used CBs of preempting process τ_j is given by RCS_{end}, assuming end is the last basic block of τ_j. Finally, the CRPD at basic block B is computed by the intersection of the used cache blocks of τ_j and the useful CBs of τ_i.

3.1. Preemption scenarios

We extend this general modelling for preemtion scenarios as follows. The cost of the first preemption at B_k is calculated by the data flow algorithm of [9]. For a second preemption we insert after B_k n nodes in the CFG of τ_i, if the preempting process τ_j finishes with n different RCSs. Figure 1 shows part of the CFG of τ_i with four basic blocks. To model a preemption at node B_3, and assuming three RCSs at τ_j last node, we insert three preemption nodes P_1, P_2 and P_3. Now the iterative data

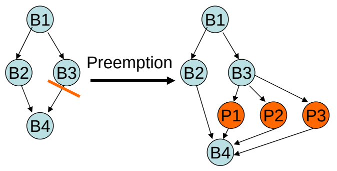

Figure 1. Modeling of a preemption by τ_j with three nodes P_1, P_2 P_3 in CFG of preempted process τ_i

flow analysis is applied again and the RCS of all other nodes are recalculated. This models the fact that useful CBs might be overwritten by a preemption and thus cannot be replaced again. After recalculating the RCSs the CRPD is recalculated at the next preemption point. This procedure is applied for every preemption point.

The complexity of this accurate modelling of n preemptions for a single process execution is exponential with the number of RCS states of the preempting task, because the n RCS states are propagated in the CFG of the preempted process and increase the number of cache states.

3.2. Multiple process execution

Many automotive control applications consist of sequential code without loops. A cache will not speed up a single execution because of the high cache miss penalty. A cache architecture is only well desigend if the cache is large enough that CBs of a single process can be reused in later executions. For simplicity we assume that no processes run between two activations of a process τ_i. We model a warm cache cache like in Subsection 3.1 by inserting n nodes in the CFG *before* the first node for n different RCS_{end} sets of process τ_i.

However, it is important to consider the processes which run between two activations. In this paper we assume the conservative approximation that all higher priority and lower priority proceses of the system execute. We model the execution of these intermediate execution of $\tau_i, \tau_{j_1}, \cdots, \tau_{j_k}, \tau_i$ as a sequence of process executions. The cache states of RCS_{end} of the preceding process τ_{j_m} are inserted as start nodes in the CFG of process $\tau_{j_{m+1}}$. The last process is τ_i again. This assumes that the CRPD at the second activation of process τ_i is independent of the order and the frequency of intermediate processes which is shown in [11].

4. Experiments

We select six different benchmarks: a square root calculation `sqrt` [9], array calculation with loops `dac` [12], two sequential programs of add instructions `lin`, `lin2` and two linear programs `nsich` and `statm`, part of a car window lift control generated by STAtechart Real-time-Code generator STARC [3]. The memory size ranges from 94 Byte till 872 Byte. We use the ARM developer studio for processor simulation and DINERO for cache simulation. All benchmarks are compiled for ARM946 assembly language with fixed four byte instruction width. Given the CFG generated from C code by [12], a tool for worst case execution time analysis for single processes, and the ARM memory map file our analyzer computes the CRPD.

4.1. Multiple process execution

Table 1 shows the response time for different cache architectures and benchmarks. A 2-way associative 1024 Byte cache with 8 Byte block size is denoted as 1024-8-2. With the ARM simulator we determine the core execution time of the processes and the instruction trace. t^{negi}_{resp} denotes the response time according to [9] assuming an empty cache at process start and t^{ana}_{resp} the response time calculated by our approach. In our experiments we assumed one clock cycle for a cache hit and a cache miss penalty of 20 clock cycles. The

Benchmark	Cache-C.	t^{negi}_{resp}	t^{ana}_{resp}	t^{sim}_{resp}	P_l[%]
dac/linear	256-8-1	1193	1041	1041	15
dac/linear	512-8-1	1193	1041	1041	15
dac/linear	1024-8-2	1041	813	813	28
sqrt/linear	512-8-1	2119	1549	1492	42
sqrt/linear	1024-8-2	1929	1131	1131	70
sqrt/linear	2048-8-2	1929	1131	1131	70
linear2/nsich	1024-8-1	3336	3070	3032	10
linear2/nsich	2048-8-1	4269	2690	2690	58
linear2/nsich	2048-8-2	4269	2690	2690	58
statm/nsich	512-8-1	4174	4174	4174	0
statm/nsich	1024-8-1	4174	3585	3547	18
statm/nsich	2048-8-1	4174	2274	2274	83

Table 1. Response time in clock cycles for a preemption during second activation for several benchmarks and cache sizes

results show that the response time is pessimistically overestimated by [9]'s approach. The last column shows the performance loss $P_l = \frac{t^{negi}_{resp} - t^{sim}_{resp}}{t^{sim}_{resp}}$, which could be gained with a more accurate analysis. For example, the performance loss in case of a 1KB and 2KB cache for `sqrt/linear` is 70% and for `statm/nsich` even 83% for the 2KB cache. We see that the current approach is less accurate for relevant larger caches.

The results for our refined analysis is in most cases exact to the simulated response time, the maximum error is 4% in case of `sqrt/linear` for direct mapped 512 Byte instruction cache.

4.2. Preemption scenarios

Now we consider multiple preemptions with an empty and preloaded cache. Table 2 presents the preemption cost of five preemptions for four task sets. The results show that for an empty cache our analysis does not improve the accuracy for benchmarks with or without loops. The reason is that the most expensive preemption points are inside

LP/HP Task	Cache Config.	Empty Cache		Preload. Cache	
		Negi	Ana	Negi	Ana
dac/lin	512-8-1	52	52	52	43
dac/lin	1024-8-1	52	52	52	40
dac/lin	1048-8-1	12	12	12	12
sqrt/lin	512-8-1	182	182	182	169
sqrt/lin	1024-8-1	47	47	47	1
sqrt/lin	2048-8-1	42	42	42	0
nsich/lin2	512-8-1	104	104	104	83
nsich/lin2	1024-8-1	104	104	104	74
nsich/lin2	2048-8-1	99	99	99	0
statm/lin2	512-8-1	125	125	125	107
statm/lin2	1024-8-1	125	125	125	97
statm/lin2	2048-8-2	120	120	120	0

Table 2. Comparison of total number of cache misses of Negi et al. and our approach for 5 preemptions with empty and preloaded cache for given lower priority (LP) and higher priority (HP) tasks.

a loop body. However, in the case of multiple activations with preloaded cache our analysis approach yield more accurate results. For a larger 2 KB cache the preemption cost is even zero for all preemptions in benchmark `nsich/linear2` and `statm/linear2`, in contrast to 99 and 120 cache misses in Negi et al.'s approach.

The performance of our analysis ranged from several minutes till several hours. The reason for the long running time is the exponential number of states that are propagated after inserting a preemption node.

5. Conclusion

In this paper we have extended the approach of [9] to consider multiple process activations and preemption scenarios for m-way associative instruction caches. The results with a realistic processor architecture show that cache effects lead to process interdependencies which can easily outweigh individual process execution times. Such cases are not covered by the classical performance analysis approaches which are based on individual process execution times plus independent blocking times (e.g. [2]). However, the complexity of the proposed analyis is exponential with the number of cache states of the preempting process. Future research is necessary to develop less complex algorithms.

Applications with loops behave better also in other approaches. However, for automotive control applications linear code is very important (e.g. Matlab generated code). Here current approaches result high overestimations. On the other hand, cache parameters have a significant influence on process interdependence. We can therefore conclude that cache design should receive maximum attention in embedded system design, use process systems as benchmarks rather than individual processes to consider multiple process activation and that new models and approaches are needed for performance analysis of systems with caches.

References

[1] J. V. Busquets-Mataix and A. Wellings. Adding instruction cache effect to schedulability analysis of preemptive real-time systems. In *IEEE Real-Time Technology and Applications Symposium*, pages 204–212, June 1996.

[2] G. Buttazzo. *Hard Real-Time Computing Systems*. Norwell, MA: Kluwer, 1997.

[3] C-Lab. Wcet benchmarks. http://www.c-lab.de/home/de/download.html.

[4] A. Datta, S. Choudhury, A. Basu, H. Tomiyama, and N. Dutt. Satisfying timing constraints of preemptive real-time tasks through task layout technique. In *Proceedings of 14th IEEE VLSI Design*, pages 97–102, January 2001.

[5] W. Henderson, D. Kendall, and A. Robson. Improving the accuracy of scheduling analysis applied to distributed systems computing minimal response times and reducing jitter. *Real-Time Systems*, 20(1):5–25, 2001.

[6] D. Kästner and S. Thesing. Cache sensitive preruntime scheduling. In *ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems*, volume 1474 of *Lecture Notes in Computer Science*, pages 131–145. Springer, 1998.

[7] C.-G. Lee, K. Lee, J. Hahn, Y.-M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim. Bounding cache-related preemption delay for real-time systems. *IEEE Transactions on software engineering*, 27(9):805–826, November 2001.

[8] F. Mueller. Compiler support for software-based cache partitioning. In *Workshop on Languages, Compilers, and Tools for Real-Time Systems*, La Jolla, June 1995.

[9] H. S. Negi, T. Mitra, and A. Roychoudhury. Accurate estimation of cache-related preemption delay. In *CODES+ISSS'03*, Newport Beach, California, USA, October 1-3 2003.

[10] I. Puaut and D. Decotigny. Low-complexity algorihtms for static cache locking in multitasking hard real-time systems. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, 2002.

[11] J. Staschulat and R. Ernst. Crpd independence for multiple process execution. Technical report, IDA, TU Braunschweig, March 2004.

[12] F. Wolf. *Behavioral Intervals in Embedded Software*. Kluwer Academic Publishers, 2002.