UNIVERSITÄT AUGSBURG
Universitätsbibliothek

OPEN ACCESS
OPUS AUGSBURG

Computing the maximum blocking time for scheduling with deferred preemption

Sebastian Altmeyer, Claire Burguière, Reinhard Wilhelm

Angaben zur Veröffentlichung / Publication details:

Altmeyer, Sebastian, Claire Burguière, and Reinhard Wilhelm. 2009. "Computing the maximum blocking time for scheduling with deferred preemption." In *2009 Software Technologies for Future Dependable Distributed Systems, 17-17 March 2009, Tokyo, Japan*, edited by Toshiaki Aoki, Sayaka Akioka, Robert Pettit, and Keewook Rim, 200–204. Piscataway, NJ: IEEE. https://doi.org/10.1109/stfssd.2009.12.

Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:
Deutsches Urheberrecht
Weitere Informationen finden Sie unter: / For more information see:
https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publiz/

Computing the maximum blocking time for scheduling with deferred preemption*

Sebastian Altmeyer, Claire Burguière, Reinhard Wilhelm
Compiler Design Lab
Saarland University
66041 Saarbrücken, Germany
{altmeyer,burguiere,wilhelm}@cs.uni-saarland.de

Abstract

Deferred preemption enables a trade-off between the high dynamics of a preemptive schedule on the one hand, and the predictability of a non-preemptive system on the other hand. In addition to bounds on the execution time and the context-switch costs, the schedulability analysis for deferred preemption needs the maximum time a preemption can be delayed, called maximum blocking time. Scheduling theory is based on an abstraction level where these values are assumed to be given. So, the related work focuses on the scheduling analysis and not on the computation of the maximum blocking time. In this paper, we propose a new method to determine the maximum blocking time of a task given a fixed set of preemption points. To derive a sound upper bound, our approach also includes the safe determination of the context-switch costs which might increase the maximum blocking time.

1 Introduction

Tasks within an embedded system are scheduled either preemptively or non-preemptively. A preemptive schedule enables a higher performance than a non-preemptive one. Some task sets are even only schedulable preemptively. In contrast, timing analyses deriving upper bounds on the execution time (WCET) usually assume uninterrupted task execution. Timing analyses for preemptively scheduled tasks are more complex and less precise. A trade-off between both is given by deferred preemption, also called cooperative scheduling, where preemption is only allowed at selected points in the task. Preemptive and non-preemptive scheduling can thus be seen as the two extremes of deferred preemption, trading dynamics with predictability.

*This work was supported by ICT project PREDATOR in the European Community's Seventh Framework Programme, by Transregional Collaborative Research Center AVACS of the German Research Council (DFG) and by ARTIST DESIGN NoE.

For the schedulability analysis of deferred preemption, not only upper bounds on the execution times and context-switch costs are needed, but also the maximum blocking time, i.e., the maximum time by which a preemption can be delayed. Techniques for the first two parameters are already well studied and available. The last point, the determination of maximum blocking time, however, has so far not been discussed in detail: the main focus of related works is on the analysis of cooperative scheduling. Often, tasks with fixed preemption points are considered to be composed of subtasks, which is on the one hand not always true (in case of preemption points within loops) and on the other hand requires a separate analysis of each such subtask.

In this paper, we propose a new method, using integer linear programming [9], to compute the maximum blocking time (MBT) of tasks given a fixed set of preemption points. Our method uses an extended implicit path enumeration technique (IPET, [13]), typically used to derive bounds on the execution times of tasks. We only slightly increase the complexity of IPET. We also consider the context-switch costs due to preemption that might increase the maximum blocking time. These costs are computed using the useful cache block approach.

The paper is structured as follows. Related work is discussed in Section 2. Section 3 explains the techniques our approach relies on and Section 4 describes the details of our approach.

2 Related Work

In [5], Lee et al. propose scheduling with deferred preemption. While the main contribution of their paper is the scheduling theory for deferred preemption, they shortly describe a method to bound the maximum blocking time. This method explicitly computes all possible subpaths between preemption points using the extended timing schema [6]. They also restrict preemption to points where the context-switch costs do not exceed a predefined limit. Hence, they do not need to take this delay into account during the

computation of the maximum block time. However, large non-preemptible areas or even completely non-preemptive tasks may result from the strong constraints for preemption points.

In [3], Kästner and Thesing propose an offline scheduling method which takes into account inter-task cache effects. For this schedule, they split all tasks into sequences of subtasks between preemption points and perform a timing analysis for each of these segments separately. This separation also restricts the preemption points.

In contrast to the former approaches, our method imposes no restriction on the selection of preemption points and directly accounts for the context-switch costs. Since our approach uses the implicit path enumeration technique, it does not suffer from the high complexity of explicitly analyzing all paths or subtasks.

3 Basics

Programs under examination are represented as control flow graphs (CFG). Nodes of the CFGs are basic blocks: maximal sequences of instructions with exactly one entry and one exit point. Therefore, if one instruction of a basic block is executed, so are all others. The edges of the control flow graphs represent the possible control flows.

Definition 1 (Control Flow Graph)
An analyzed program P is represented as a control flow graph

$$CFG = (V, E, s, e)$$

where $V = \{B_1, \ldots B_n\}$ denotes the set of basic blocks B_i of P and $E \in V \times V$ the corresponding edges connecting them. The start node is denoted by s and the end node by e, respectively.

Timing analysis (see [2] for an overview) performs several subanalyses on these CFGs: value analysis, loop-bound analysis, low-level/micro-architectural analysis (including cache analysis), and path analysis. Value analysis determines actual addresses of memory accesses and values for registers and memory cells. These values are then used to derive loop-bounds and to classify memory accesses as hits or misses in the cache analysis. This classification used by the low-level analysis to derive upper bounds on the execution times of basic blocks. In a last step, the path analysis combines this information to find the path within the CFG with highest execution time (IPET, detailed in Section 3.3). Figure 1 shows the complete flow of a typical timing analysis.

3.1 Useful Cache Block Analysis

Timing analysis typically assumes uninterrupted task execution. If a task is preempted at some point, the execution

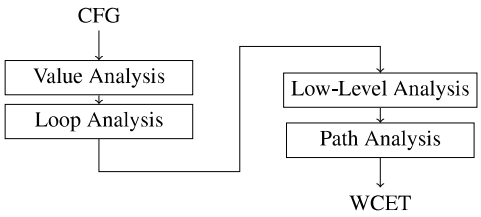


Figure 1. Structure of the timing analysis.

time may be increased due to changes in the cache, i.e., due to evicted cache entries. A standard approach to compute context-switch costs is deriving the set of so-called useful cache blocks (UCBs).

Definition 2 (Useful Cache Block)
A useful cache block C at a given program point P is a memory block which 1) is definitely cached at P and 2) may be reused by some program point P' that can be reached from P.

An example of *useful cache blocks* is given in Figure 2. The content of the data cache and *UCBs* are detailed for each program line: cache blocks are *useful* when they are cached and possibly reused later in the program execution.

In case the task is preempted at program point P, the context-switch costs include the costs to reload all cache blocks that could be reused [4]. This assumption is overly pessimistic, such that the useful cache block approach can be – and has been – improved, for example, by reducing the set of the UCBs to those, which may be evicted by the preempting tasks [8, 10, 11].

The useful cache block approach is only available for direct-mapped and LRU caches and for processors without timing anomalies [7].

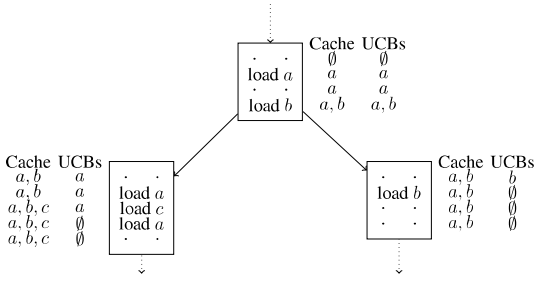


Figure 2. Example of an UCB analysis (memory blocks a, b, c; cache content and set of UCBs on each program point)

3.2 Deferred Preemption

Scheduling with deferred preemption offers a balance between preemptive and non-preemptive scheduling by allowing preemption only at specified preemption points [1]. Due to the decreased dynamics of the system, predictability is increased. For the sake of simplicity, we assume preemption points directly at the beginning or at the end of a basic block. The typical small size as well as the structure of the basic blocks justifies this restriction; adjacent instructions and semantic relationship within a basic block causes the context-switch costs to be smaller at the boundaries of a basic block than within. Only in the rare case of large basic blocks, this restriction increases the bound on the maximal blocking time.

The maximum blocking time which we handle in this paper is needed to guarantee the schedulability of a system with deferred preemption. Figure 3 shows an example on how the blocking time influences the system schedule. Task T_1 has the highest priority: it preempts task T_2 two times. However, the preemption is delayed until the next preemption point is reached. This delay between task activation and task execution, due to deferred preemption, is the blocking time (BT). Note that the figure shows also the context-switch costs that can be observed during the blocking time (and may increase it).

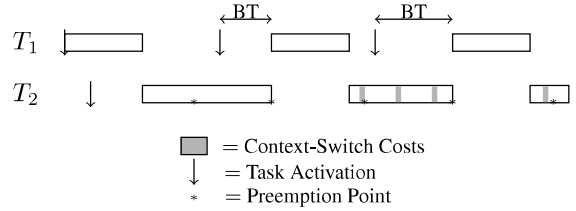


Figure 3. Example of scheduling with deferred preemption

3.3 Implicit Path Enumeration

Implicit path enumeration (IPET, [13, 12]) is widely used in the context of timing analysis. As indicated by the name, IPET searches the longest execution path within a program implicitly, avoiding a complex explicit enumeration. The analyzed program is described by structural and flow constraints representing and constraining the possible control flow. In combination with an optimization function (maximizing the sum of execution times of basic blocks times the execution counts), we obtain an integer linear problem, whose maximal value delivers an upper bound for the execution time of all path through the program. As in the original paper by Li and Malik, we introduce two types of counters, one for the basic blocks and one for the edges: $x_i \in \mathbb{N}$ denotes how often basic block $B_i \in V$ is executed, and $y_j \in \mathbb{N}$ denotes how often edge $E_j \in E$ is traversed. These counters are the variables within the ILP. The coefficients are given by a prior loop and micro-architectural analysis: $c_i \in \mathbb{N}$ denotes an upper bound on the execution time for basic block B_i, and $b_l \in \mathbb{N}$ denotes an upper bound on the number of iterations of loop l and, by this, how often the loop is entered.

Given a control flow graph $CFG = (V, E, s, e)$. The objective function is given in Equation (1). Equation (2) and (3) bound the number of executions of the first, resp. last node. Each node is executed as often as it is entered (4) and left (5). Equation (6) represents the loop bound, where E_l denotes the first edge in the loop body and b_l the loop bound. An example of IPET (CFG with the corresponding constraints) is given in Figure 4.

$$\max \sum_i x_i c_i \tag{1}$$

$$x_i = 1, \text{ where } B_i = s \tag{2}$$

$$x_j = 1, \text{ where } B_j = e \tag{3}$$

$$\sum_{j \in S} y_j = x_k, \text{ where } S = \{j | E_j = (_, B_k)\} \tag{4}$$

$$x_k = \sum_{j \in S} y_j, \text{ where } S = \{j | E_j = (B_k, _)\} \tag{5}$$

$$y_l \leq b_l (\sum_{j \in S} y_j) \text{ where } S = \{j | E_j \text{ loop entry edge}\} \tag{6}$$

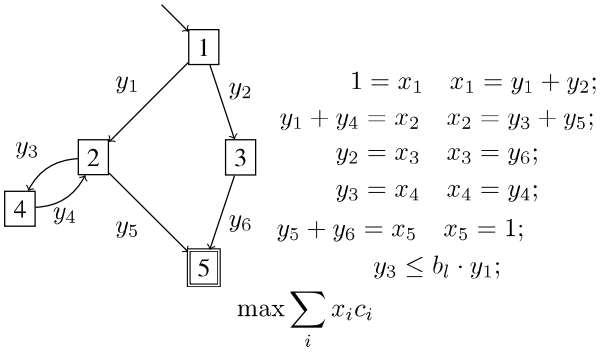


Figure 4. Control Flow Graph and the corresponding ILP

4 Computing Maximal Blocking Time

In this section, we propose an extension to IPET to derive the maximum blocking time given the set of preemption

points. These preemption points can not be seen as graph separators dividing the control flow graph into a set of subgraphs, as Figure 5 shows. Hence, we can not apply the simple method of analyzing tasks with preemption points as a sequence of smaller tasks.

In our approach, we treat preemption points as additional start/end nodes. As mentioned before, we allow preemption points only at the beginning, resp. the end of a basic block. If a preemption point is located at the entrance of a basic block B_i, we introduce an artificial node B_i^u directly before B_i (upper part) and, if it is located at the end, we introduce an artificial node B_i^l directly after B_i (lower part).

Note that we do not need to change the control flow graph itself. Just the implicit path enumeration technique, the last step of the timing analysis, is extended, and so, the set of constraints and the objective function are modified. For each preemption point, we introduce exactly one new variable, the number of constraints remains unchanged. The described changes to the control flow graph are introduced only to explain our method.

Definition 3 (Sets of Preemption Points)
The set

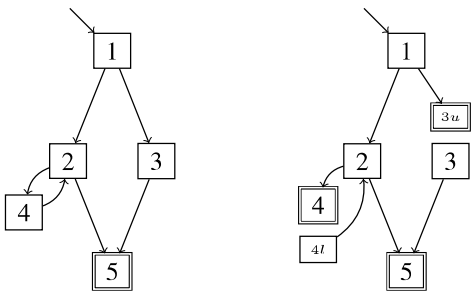
$$Up = \{B_i | \text{ preemption point at beginning of } B_i\}$$

contains all basic blocks with a preemption point at the head, the set

$$Lo = \{B_i | \text{ preemption point at end of } B_i\}$$

at the end.

Figure 5 shows the extension to the control flow graph and the sets Lo and Up. Preemption points are at the beginning of basic block B_3 and at the end of block B_4. Therefore, the nodes B_3^u and B_4^l are introduced.



$Lo = \{B_4\} \quad Up = \{B_3\}$

Figure 5. Original and Modified CFG with Preemption Points.

Structural Constraints The path we are interested in starts from the lower part B_i^l of one of the preemption points or from the start node. It ends either in the upper part B_i^u of one of the preemption points or in the end node. By setting the sum of the execution counters to one, Equation (7) and Equation (8) restrict the blocking path to exactly one start node and one end point.

$$x_s + \sum_{B_i \in Lo} x_i^l + \sum_{B_i \in Up} x_i = 1 \tag{7}$$

$$x_e + \sum_{B_i \in Up} x_i^u + \sum_{B_i \in Lo} x_i = 1 \tag{8}$$

As in the original implicit path enumeration technique, each node is entered (9) and left (10) as often as is its executed. Preemption points cut the control flow graph, hence the relation between incoming and outgoing edges is also cut. In case preemption occurs at the beginning of basic block B_k all incoming edges enter B_k^u. In case preemption occurs at the end of basic block B_k, all outgoing edges leave B_k^l. No constraint connects B_k to B_k^u and B_k^l, respectively.

$$\sum_{j \in S} y_j = \begin{array}{ll} x_k^u & \text{if } B_k \in Up \\ x_k & \text{otherwise} \end{array} \tag{9}$$

$$\text{where } S = \{j | E_j = (_, B_k)\}$$

$$\sum_{j \in S} y_j = \begin{array}{ll} x_k^l & \text{if } B_k \in Lo \\ x_k & \text{otherwise} \end{array} \tag{10}$$

$$\text{where } S = \{j | E_j = (B_k, _)\}$$

Flow Constraints Equation (11) represents the loop constraint in the extended IPET. The constant b_l denotes the loop bound and E_l is the first edge in the loop body.

$$y_l \leq b_l \left(\sum_{j \in S} y_j \right) \tag{11}$$

$$\text{where } S = \{j | E_j \text{ loop entry edge}\}$$

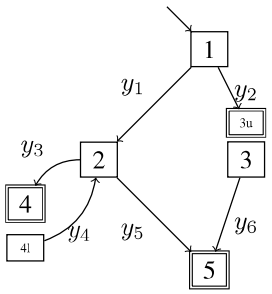
The non-preemptive path with the highest execution time determines the maximum blocking time. Since the execution time of the basic blocks does not contain cache-reload times due to preemption, we have to account for them separately. Therefore, the maximum blocking time is determined on the one hand by the length of the path from a preemption point (or the start node) to a preemption point (or the end node) and on the other hand by the context-switch costs at the beginning of the path.

Definition 4 (Context-Switch Costs)
The context-switch costs at the beginning of a basic block i are denoted by C_i^u, while we denote context-switch costs at the end of a basic block i by C_i^l

Objective Function The objective function to compute MBT is thus given by the sum of execution counter x_i times execution time c_i for each basic block B_i and by the context-switch costs. These costs either occur due to a preemption point at the end of a basic block $\left(\sum_{B_i \in Lo} x_i^l C_i^l\right)$, or at the beginning $\left(\sum_{B_i \in Up} x_i C_i^u\right)$.

$$\max \sum_{i=1}^{n} (x_i c_i) + \sum_{B_i \in Lo} x_i^l C_i^l + \sum_{B_i \in Up} x_i C_i^u$$

An example of extended IPET (CFG with the corresponding constraints) is given in Figure 6. The number of constraints remains unchanged compared to Figure 4. Equation (7) ensures that at most one time context-switch costs contribute to the maximum blocking time.



$$x_1 + x_3 + x_4^l = 1;$$
$$1 = x_4 + x_5 + x_3^u;$$
$$x_1 = y_1 + y_2;$$
$$y_1 + y_4 = x_2 \quad x_2 = y_3 + y_5;$$
$$y_2 = x_3; \qquad x_3^u = y_6;$$
$$y_3 = x_4; \qquad x_4^l = y_4;$$
$$y_5 + y_6 = x_5;$$
$$y_3 \leq b y_1;$$

$$\max \sum_i x_i c_i + x_4^l C_i^l + x_3 C_i^u$$

Figure 6. Extended IPET schema to compute the maximum blocking time

5 Conclusions

In this paper we have proposed a new method to maximize the blocking time due to deferred preemption. We use the implicit path enumeration technique to implicitly describe all possible paths that can be executed during a blocking time. By introducing exactly one variable per preemption point while keeping the number of constraints fixed, compared to the number of constraints to compute the WCET, we keep the complexity low. In addition, our method accounts for the context-switch costs that might increase the blocking time.

As future work, we plan to develop a method to determine optimal preemption points with respect to the context-switch costs and/or maximum blocking time. To evaluate different sets of preemption points, we need a fast and sound method to bound the maximum blocking time of each set – such as the method presented in this paper.

References

[1] R. J. Bril, J. J. Lukkien, and W. F. J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited. In *ECRTS '07: Proceedings of the 19th Euromicro Conference on Real-Time Systems*, pages 269–279, Washington, DC, USA, 2007. IEEE Computer Society.

[2] J. Gustafsson. WCET challenge 2006 - technical report. Technical report, January 2007.

[3] D. Kästner and S. Thesing. Cache aware pre-runtime scheduling. *Real-Time Syst.*, 17(2-3):235–256, 1999.

[4] C.-G. Lee, J. Hahn, Y.-M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. *IEEE transactions on computers*, 47(6), June 1998.

[5] S. Lee, C.-G. Lee, M. Lee, S. L. Min, and C.-S. Kim. Limited preemptible scheduling to embrace cache memory in real-time systems. In *LCTES '98: Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems*, pages 51–64, London, UK, 1998. Springer-Verlag.

[6] S.-S. Lim, Y. H. Bae, G. T. Jang, B.-D. Rhee, S. L. Min, C. Y. Park, H. Shin, K. Park, S.-M. Moon, and C. S. Kim. An accurate worst case timing analysis for risc processors. *IEEE Trans. Softw. Eng.*, 21(7):593–604, 1995.

[7] T. Lundqvist and P. 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.

[8] H. S. Negi, T. Mitra, and A. Roychoudhury. Accurate estimation of cache-related preemption delay. In *CODES+ISSS '03: Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 201–206, New York, NY, USA, 2003. ACM.

[9] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, 1986.

[10] J. Staschulat and R. Ernst. Multiple process execution in cache related preemption delay analysis. In *EMSOFT '04: Proceedings of the 4th ACM international conference on Embedded software*, pages 278–286, New York, NY, USA, 2004. ACM.

[11] Y. Tan and V. Mooney. Integrated intra- and inter-task cache analysis for preemptive multi-tasking real-time systems. In *In Proceedings of the 8th International Workshop, SCOPES 2004, in: Lecture Notes on Computer Science, LNCS3199*, pages 182–199. Press, 2004.

[12] H. Theiling. ILP-based interprocedural path analysis. In *Proceedings of the Workshop on Embedded Software*, Grenoble, France, October 2002.

[13] Y. tsun Steven Li and S. Malik. Performance analysis of embedded software using implicit path enumeration. In *in Proceedings of the 32nd ACM/IEEE Design Automation Conference*, pages 456–461, 1995.