Extension of Multicore Response Time Analysis to analyse the Utilization of DMA Units

Alexander Stegmeier
Embedded Systems
University of Augsburg, Germany
Email: stegmeier@es-augsburg.de

Sebastian Altmeyer
Embedded Systems
University of Augsburg, Germany
Email: altmeyer@es-augsburg.de

I. Introduction

Multicore Response Time Analysis (MRTA) [1] is a technique to calculate the worst-case response times (WCRTs) of a complete task set and to provide an estimation about the schedulability of an application on a given platform. Thereby, it takes possible delays according to interferences due to occupied shared resources (e.g. shared bus) into account. However, it introduces only moderate overestimation, since it purely regards conflicts of task executions that influence the timing of each other.

The analysis relies on a model of the multicore hardware. Each task is executed on one dedicated execution unit using partitioned fixed priority scheduling. Depending on that model, the analysis currently supports only a restricted set of applications that run on simple multicore architectures. To the best of our knowledge, in the mentioned type of analysis outsourcing parts of a task's workload to specialized hardware is not supported. Such a scenario could be the utilization of a Direct Memory Access (DMA) unit for copying data. In that case a task configures the DMA unit and triggers its execution. While the copy procedure runs on the DMA, the task may stall or perform some other work.

Our goal is to enlarge the set of analysable applications by extending the MRTA to support the usage of specialized hardware components. To reach that goal, we propose an extension of the hardware model as well as the interpretation of the task model that is used for the analysis. As a first step we extend the analysis to support the utilization of a DMA hardware unit for parts of the workload of a task.

In section II we describe the basic task and hardware model. The subsequent section III describes the functional behaviour of a DMA unit. Section IV relates to proposed extensions and adaptations to enable the analysis support for utilizing a DMA unit. The last section concludes our proposal with a statement about relevant research questions related to the proposed approach.

II. Basic hardware and task model

Our MRTA framework is based on a generic multicore platform model as presented in [1]. It assumes l timing compositional cores $P_1, \ldots, P_l$. Hence, it is safe to separately regard the different sources to account for their delays. This includes computation on a given core or the interferences generated by conflicts on a shared bus [2]. Each core is connected to the global memory and IO interface via local memory and a shared bus. The DMA unit is also connected to the shared bus.

Currently, in our model the transactions on the bus are performed atomically and without reordering. Furthermore, transactions are non-preemptable and access to the bus may be given to one resource at a time. When access is granted, the transactions take the delay for occupying the bus plus the calculated delay to retrieve data from global memory.

The task model consists of a set of n sporadic tasks $\tau_1, \ldots, \tau_n \in \Gamma$ where each task has a minimal period or inter-arrival time T_i and a deadline D_i. The deadlines are assumed to be constrained ($D_i \leq T_i$).

The tasks are statically partitioned to identical cores and scheduled per core via fixed-priority preemptive scheduling. The tasks follow a global priority order. The tasks are assumed to be independent to each other but compete for hardware resources such as processor, memory and the bus.

III. Direct Memory Access unit

In our research we consider a typical DMA unit like applied in current ARM-based microcontrollers [3].

A DMA unit is a specialized hardware component that copies data between locations of the processor's address space. It can be configured by a task running on a core and then performs a copy for the configured amount of data. This is called a DMA block transfer. Each DMA block transfer may be subdivided in multiple single DMA transfers which execute multiple instructions to transfer one portion of the configured amount of data per transfer.

Typically, a DMA unit provides the ability to concurrently perform multiple DMA block transfers. Since `load/store` instructions (used for the implementation of the copy) request the shared bus, arbitration is necessary to hinder conflicts between the block transfers. The arbitration scheme is designed so that a DMA block transfer can be preempted between two single DMA transfers. However, there is no preemption possible within a single DMA transfer.

IV. Extensions on MRTA

Our focus is to analyse a task-based application that performs block transfers via a DMA unit within its tasks. Hence, two problems arise. At first there is the need to model the behaviour of the DMA unit. Secondly, the usage of a DMA unit implies that a task is executed on multiple hardware units (e.g. CPU and DMA) instead of only one unit. This circumstance is not modelled in current MRTA frameworks.

A. Modelling a DMA unit

A DMA unit may act as master or slave on the shared bus. During configuration of the unit it behaves like a slave (similar to the global memory). However, when it processes data it is transferred to behave as a bus master (similar to a core). We propose to model both roles of the DMA separately.

Similar to the global memory the access to the slave part of a DMA is modelled via a delay that is accounted to the execution time of the requesting task. However, the master part of a DMA is modelled differently and behaves similar to the execution of tasks on cores. A DMA block transfer can be seen as a separate task running on the DMA unit and the arbitration of block transfers is modelled via a scheduling policy. Each block transfer exhibits non-preemptable parts (the single DMA transfers). This circumstance is reflected by Fixed-Priority Deferred Scheduling if single DMA transfers are considered as the non-preemptable regions of a task.

B. Modelling a task utilizing a DMA unit

A task that utilizes a DMA unit is partly executed on a CPU and partly on a DMA unit. Since the parts run on different hardware resources they may run in parallel and each part requests access to the shared bus. Thus, the task shows a reduced response time due to concurrent execution, but may suffer additional interference delays when different parts of the task concurrently try to access the same shared resource.

With regard to the analysis the task parts behave like separate tasks. Hence, we propose to model them as separate sub-task for the analysis. Each sub-task represents a chunk of continuous execution on exactly one hardware component (core or DMA).

Typically, a task includes the configuration of the DMA as well as the execution of a DMA block transfer. Thus, precedence constraints must be regarded when calculating the final worst-case response time of the complete task. This model relates to the concept of Directed Acyclic Graphs (DAG) tasks as applied in [4]. However, the theory must be adopted to fit for the current problem. An illustration of how a task may be split to multiple sub-tasks is depicted in Figure 1.

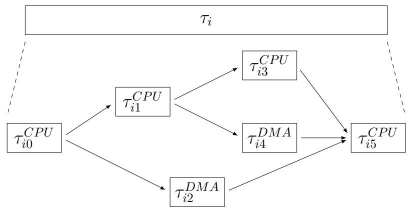

Fig. 1. Original task split into several sub-tasks with precedence constraints to model the task's execution on multiple hardware components.

V. Conclusion and Research Question

Altogether, we think that it is appropriate to apply the presented extensions to analyse task-based applications that utilize DMA execution. The extended task model splits a task into sub-tasks, each running on only one hardware component. The response times of the generated sub-tasks running on a core are analysed exactly like for original tasks. The accesses to the DMA unit for configuration purpose is regarded similar to global memory accesses. Sub-tasks running on the DMA unit are analysed similar to the tasks running on the cores.

According to the presented idea we ask several research questions. Is it appropriate to model DMA units in the two-folded way of a master and a slave component? How can the model of the DMA unit be extended to act as an abstract representation for other types of hardware acceleration units? Does the described extension of the task model meet the requirements for the utilization of a DMA unit and finally, does it reflect the timing of the task model used for developing software?

References

[1] S. Altmeyer, R. I. Davis, L. Indrusiak, C. Maiza, V. Nelis, and J. Reineke, "A generic and compositional framework for multicore response time analysis," in *Proceedings of the 23rd International Conference on Real Time and Networks Systems - RTNS '15, November 2015, Lille, France*, J. Forget, Ed., 2015.

[2] S. Hahn, J. Reineke, and R. Wilhelm, "Towards compositionality in execution time analysis: Definition and challenges," *SIGBED Rev.*, vol. 12, no. 1, p. 28–36, mar 2015. [Online]. Available: https://doi.org/10.1145/2752801.2752805

[3] STMicroelectronics, "Rm0351 reference manual - stm32l47xxx, stm32l48xxx, stm32l49xxx and stm32l4axxx advanced arm ® -based 32-bit mcus," pp. 336–360, June 2021. [Online]. Available: https://www.st.com/resource/en/reference_manual/rm0351-stm32l47xxx-stm32l48xxx-stm32l49xxx-and-stm32l4axxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

[4] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo, "Partitioned fixed-priority scheduling of parallel tasks without preemptions," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, 2018, pp. 421–433.