Mälardalen University Press Licentiate Theses
No. 189

LOCK-BASED RESOURCE SHARING IN REAL-TIME MULTIPROCESSOR PLATFORMS

Sara Afshar

2014

School of Innovation, Design and Engineering

ISBN 978-91-7485-178-6
ISSN 1651-9256
Printed by Arkitektkopia, Västerås, Sweden

Populärvetenskaplig sammanfattning

Processorn är hjärnan i ett datorsystem. I processorn kör ett eller flera program där varje program typiskt sätt ansvarar för att utföra en särskild uppgift eller funktion. Utförandet av alla uppgifter tillsammans resulterar systemets funktionalitet, till exempel den låsningsfria bromsfunktionen hos en bil.

I många datorsystem är det inte tillräckligt med att alla uppgifter utförs, utan det är även av högsta vikt att dessa uppgifter utförs i korrekt tid. Vi kallar denna typ av system med tidskrav för realtidssystem. En schemaläggare är då ansvarig för att schemalägga alla programmen på processorn, dvs, diktera vilket program som ska köra och när det ska köra för att garantera att alla uppgifter kommer att utföras i tid.

Typiskt sätt så behöver programmen använda sig av datorsystemets hård- och mjukvaruresurser för att utföra sina beräkningar och sin funktionalitet. Exempel på denna typ av resurser som delas mellan programmen är I/O-enheter, buffertar och minnen. När flera program vill använda samma delade resurs samtidigt så kan programmen störa varandra och förstöra både resultat och funktion. Som tur är så finns det tekniker för att möjliggöra att flera program kan dela en resurs på ett förutsägbart sätt. En sådan teknik baserar sig på användningen av lås, och programmet som då vill använda en delad resurs måste först erhålla låset för denna resurs innan programmet får använda resursen. Om låset inte redan innehas av något annat program, dvs. låset är ledigt, så kan programmet ta låset och använda den delade resursen. När programmet sen är klart med den delade resursen så lämnas låset tillbaka. Låsning av delade resurser på detta sätt hindrar att flera program använder resursen samtidigt. Vi kallar denna typ av teknik för hantering av delade resurser för resursdelningsprotokoll.

Nyligen och med syfte att förbättra prestandan för datorer, så har fler än en

processor använts i datorsystem. Denna typ av datorer med flera processorer på en delad hårdvara kallas för multiprocessorer. Det finns två konventionella metoder för schemaläggning av program som körs på multiprocessorer. Vidare så har motsvarande resursdelningsprotokoll utvecklats. Nyligen har dock en tredje kategori av schemaläggare för multiprocessorer utvecklats. Denna nya kategori är mer effektiv jämfört med de två tidigare metoderna. Den nya metoden kallad hybridschemaläggning använder sig av en kombination av bägge konventionella metoderna för att schemalägga program. Som en konsekvens av komplexiteten i denna nya typ av schemaläggning så är det inte okomplicerat att använda sig av de konventionella resursdelningsprotokollen. I denna licentiatavhandling har vi utvecklat nya resursdelningsprotokoll för att användas till hybridschemaläggning av multiprocessorsystem. Vi har utvecklat och undersökt olika och alternativa låsningsmetoder i syfte att tillhandahålla en effektiv och högpresterande lösning för moderna realtidssystem som kör på multiprocessorer.

Abstract

Embedded systems are typically resource constrained, i.e., resources such as processors, I/O devices, shared buffers or shared memory can be limited for tasks in the system. Therefore, techniques that enable an efficient usage of such resources are of great importance.

Looking at software in industrial systems, large and complex software systems are often divided into smaller parts (applications) where each part is developed independently. In recent years, a shift from single-processor platforms to multiprocessor platforms has become inevitable due to availability of processor chips and requirements on increased performance. Due to such migration from single-processor platforms to multiprocessor platforms along with a desire to make efficient use of system resources, these applications may eventually be integrated on a shared multiprocessor platform. In order to facilitate the integration of the applications on a shared platform, the timing and resource requirements of each application can be provided when the application is developed. The system integrator can benefit from such provided information of each application to ease the integration process. In this thesis, we have provided the resource and timing requirements of each application for applications that may need to be allocated on several processors when they are developed.

Although many scheduling techniques have been studied for multiprocessor systems, these techniques are usually based on the assumption that tasks are independent, i.e., tasks do not share resources other than the processors. This assumption is typically not true, as other resources must be shared among the tasks. In this thesis, we provide a solution to such systems to handle sharing of resources other than processor among tasks. Two traditional approaches exist for multiprocessor systems to schedule tasks on processors. A recent scheduling approach for multiprocessors has combined the two traditional approaches and achieved a more efficient hybrid approach. Due to the complex nature of this scheduling approach the conventional approaches for resource sharing

could not be used in straight forward manner. In this thesis, we have developed a solution for resource sharing in hybrid scheduled multiprocessor systems.

A second concern is that enabling resource sharing can cause unpredictable delays and variations in response time of tasks which as a consequence can degrade system performance. Therefore, it is of great significance to improve the resource handling techniques to reduce the effect of imposed delays caused by resource sharing. In this thesis we have proposed alternative techniques for resource handling that improve system performance.

To my dear and loving husband,
Mohammad

Acknowledgments

Foremost, I would like to express my very profound gratitude to my supervisors Prof. Thomas Nolte and Dr. Moris Behnam who have been guiding me from my master thesis. I am grateful for their continuous support, insightful suggestions, comments and feedback throughout my studies. I am thankful for high spirits they bring to work. Thomas has encouraged me through my studies and taught me not to be afraid of flying higher. Discussions with him always have inspired me. Also, I am grateful for the valuable feedback of Moris which have helped to improve my work. His office door has always been open to me for discussions. This thesis would not be possible without your endless support and help!

Next, I would like to express my deep sense of gratitude to Prof. Reinder J. Bril (University of Eindhoven) who guided me through my latest works with his useful feedback and supports. It was a great pleasure working with you and I enjoyed every moment of our discussions.

I also would like to express my gratitude towards Dr. Farhang Nemati, who has supervised me for my master thesis and inspired me to continue for doctoral studies. I am grateful for all his support and feedback.

Further, I wish to express my appreciation to the lecturers and professors who I learned a lot from during meetings, lectures, seminars and PhD courses. My exceptional thanks goes to Damir Isovic (who was my first teacher and mentor at MDH), Hans Hansson, Ivica Crnkovic, Mikael Sjödin, Thomas Nolte, Paul Pettersson, Emma Nehrenheim, Monica Odlare, Jan Gustafsson, Björn Lisper, Cristina Seceleanu, Jan Carlson, Radu Dorbin, Dag Nyström, Ning Xiong, Kristina Lundqvist, Harold (Bud) Lawson, Giacomo Spampinato, Gordana Dodig-Crnkovic, Lars Asplund, Mats Björkman, Mikael Ekström, Moris Behnam, Maria Lindén, Baran Çürüklü and Jukka Mäki-Turja. I also would like to appreciate IDT administration staff for their help with practical issues. Many thanks Carola, Susanne, Sofia, Ingrid, and the others.

A great thank to my friends and colleagues at the department for all the wonderful time we had together during these years in conference trips, fika, movie and game gatherings and badminton. I wish to thank my office mate Meng for all the nice discussions we had and his generous help whenever I needed. I would also like to thank all PhD students (and former members) in my research group Mohammad, Nima, Matthias, Hamid, Daniel, Hang, Mikael; and in an alphabetic order Abhilash, Adnan, Aida, Alessio, Andreas G., Aneta, Anna, Antonio, Apala, Arash, Batu, Cristina, Dag, Daniel K., Daniel S., Eddie, Elena, Federico, Francisco, Fredrik Ek., Gabriel, Gregory, Guillermo, Hus, Husni, Irfan, Ivan, Jagadish, Jiale, Josip, Juraj, Kan, Kivanc, Leo, Luka, Mahnaz (Anita), Mehrdad, Melika, Martin, Nesredin, Nikola, Omar, Pablo, Per, Predrag, Rafia, Raluca, Saad, Sara Ab., Sara Ab. Assadollah, Sara D., Séverine, Simin, Shahina, Stephan (Bob), Svetlana, Tibi, Wasif, Yue and others.

Last but not least, I would like to take this opportunity to thank my family, in particular my parents, for their endless love, support and encouragement from the very beginning of my life. I am also thankful of my wonderful sisters, Sevil and Shanay, whom made my life colorful! A special thank to my beloved husband Mohammad, for his unfailing love and support and for the entire amazing journey we shared together.

This work has been supported by the Swedish Foundation for Strategic Research under the project PRESS.

Sara Afshar
17:43, November 10th, 2014
Västerås, Sweden

List of publications

Papers included in the licentiate thesis[1]

Paper A *Resource Sharing under Multiprocessor Semi-Partitioned Scheduling*. Sara Afshar, Farhang Nemati, Thomas Nolte. In proceedings of the 18^{th} International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2012, August.

Paper B *Integrating Independently Developed Real-Time Applications on a Shared Multi-Core Architecture*. Sara Afshar, Moris Behnam, Thomas Nolte. In proceedings of the 5^{th} International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS), 2012, December.

Paper C *Flexible Spin-Lock Model for Resource Sharing in Multiprocessor Real-Time Systems*. Sara Afshar, Moris Behnam, Reinder J. Bril, Thomas Nolte. In proceedings of the 9^{th} IEEE International Symposium on Industrial Embedded Systems (SIES), 2014, June.

Paper D *Per Processor Spin-Lock Priority for Partitioned Multiprocessor Real-Time Systems*. Sara Afshar, Moris Behnam, Reinder J. Bril, Thomas Nolte. MRTC report, ISSN 1404-3041, Mälardalen Real-Time Research Centre, Mälardalen University, 2014, October.

[1] The included articles have been reformatted to comply with the licentiate thesis layout.

Additional papers, not included in the licentiate thesis

1. *Resource Sharing among Prioritized Real-Time Applications on Multiprocessors*, Sara Afshar, Nima Khalilzad, Farhang Nemati, Thomas Nolte. In the proceeding of 6^{th} International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS), 2013, December.

2. *Resource Sharing under Server-based Multiprocessor Scheduling*, Sara Afshar, Moris Behnam. In the proceeding of the 33^{rd} IEEE Real-Time Systems Symposium (RTSS), Work-in-Progress session. 2012, December.

3. *Towards Resource Sharing under Multiprocessor Semi-Partitioned Scheduling*, Sara Afshar, Farhang Nemati, Thomas Nolte. In proceedings of the 9^{th} IEEE International Symposium on Industrial Embedded Systems (SIES), Work-in-Progress session, 2012, June.

4. *Support for Hierarchical Scheduling in FreeRTOS*, Rafia Inam, Jukka Mäki-Turja, Mikael Sjödin, Mohammad Ashjaei, Sara Afshar, In the proceeding of the 16^{th} IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2011, August.

5. *Resource Sharing in Hybrid Partitioned/Global Multiprocessor Real-Time Scheduling*, Sara Afshar, Moris Behnam, Reinder J. Bril, Thomas Nolte, Technical Report, 2014, September.

Contents

I

Thesis

Chapter 1

Introduction

Multiprocessor architectures are becoming defacto architectures due to better power consumption and thermal specifications compared to the single core[1] architectures. Accordingly, a shift to the multiprocessor architectures is inevitable. With a growing interest towards replacing traditional single cores with new multi-cores in embedded systems, a demand has emerged for investigating proper scheduling techniques to allow for such migration. One major concern in the context of embedded systems is the constraint on the amount of resources available. Hence, techniques that can enable an efficient usage of processor bandwidths in such systems are of great importance.

Two main classical approaches exist for scheduling real-time tasks on multiprocessor platforms. These approaches are called *partitioned* and *global* scheduling [1, 2, 3, 4]. In partitioned scheduling, tasks are statically assigned to each processor at design-time and at run-time they will run on their assigned processor, only. Each processor has its own ready queue under partitioned scheduling and on each processor tasks are scheduled by a uniprocessor scheduling protocol. The attractive point of using partitioned scheduling is that it breaks down the problem of multiprocessor scheduling into the well-known uniprocessor problem. However, system utilization is low under partitioned scheduling. If the total task set utilization reaches slightly higher than 50% then deadlines may be missed [5]. In global scheduling, tasks are selected from one unique global ready queue and they can be assigned at run-time to any processor. Under global scheduling tasks might be preempted on one processor and resumed on another processor, i.e., migration among processors is allowed

[1] In this thesis we will use core and processor interchangeably

for tasks. Global scheduling can achieve higher system utilization compared to partitioned scheduling but migration overhead might be very expensive in such platforms and is typically ignored in the corresponding schedulability analysis.

Under partitioned scheduling if all tasks that share the same resource could be allocated on the same processor then uniprocessor synchronization protocols [6, 7] could be used without extra effort [5]. However, this cannot always be done, since all tasks that use the same resource may not fit on the same processor. Therefore, proper adjustments have been considered for uniprocessor lock-based synchronization protocols to synchronize tasks that share resources across the processors [8, 9, 10, 11, 12, 13, 14, 15, 16].

Under global scheduling, the uniprocessor synchronization protocols cannot be used directly without modifications. Therefore, lock-based synchronization protocols that are suitable for globally scheduled systems have been developed [11, 13, 17, 18].

A recent scheduling approach for multiprocessors, called *semi-partitioned scheduling* [19], uses a hybrid approach combining both partitioned and global scheduling to achieve a higher system utilization and lower migration cost.

Most of the scheduling approaches make the simplifying assumption that tasks are independent. In practice this assumption is often not true because tasks may share resources other than the CPU, such as I/O devices, shared buffers or shared memories. Clearly, in order to be compatible with the more general and practical system model with dependent tasks we need to cope with all aspects of such systems, not only the CPU. When tasks share resources in the system, they may try to access the same resource at the same time. Simultaneous access to the same resource can be problematic and decrease or invalidate the system functionality. Lock-based resource sharing protocols provide exclusive mutual access to the shared resources in a system. However, providing exclusive access to shared resources, on the other hand, may cause extra delays to tasks. Such blocking can endanger timeliness and temporal correctness of a system since it can lead to uncontrolled *priority inversions*. Priority inversions occur when a high priority job is delayed due to a low priority job for an unbounded amount of time. It is essential for resource sharing protocols to bound the delays incurred to tasks due to resource sharing. Many resource sharing protocols have been proposed for real-time multiprocessor scheduling under partitioned and global scheduling [8, 9, 10, 11, 12, 13, 14, 17, 15, 16, 18]. However, the resource sharing approaches are not fully mature in a multiprocessor context and further improvements can be applied. On the other hand, there exist scheduling frameworks that cannot use the existing resource sharing protocols due to their structural complexity. For such scheduling frameworks,

further adjustments are needed. Semi-partitioned scheduling is one such example. Many works have investigated semi-partitioned systems in the context of scheduling [19, 20, 21, 22, 23]. However, none of them has considered resource sharing among tasks for such a hybrid scheduling structure where the existing resource sharing approaches cannot be used without necessary modifications.

A common approach in industrial software systems to accelerate the development process is to break down large and complex systems into smaller subsystems. As multi-core platforms are becoming common in industrial systems, one major concern is to migrate legacy real-time applications to multi-core architectures. From an industrial point of view, co-existence of multiple real-time applications on a shared multi-core platform is an efficient solution since it can provide re-usability of the independently-developed applications besides decreasing the system power consumption and costs. By transition to a multi-core architecture, these subsystems/applications which may share resources will eventually co-exist on a shared multi-core platform. These applications, which are often developed independently of each other, may use different policies or techniques, e.g., different scheduling and/or resource sharing approaches. Compositional scheduling approaches can simplify and optimize the integration of such applications on a shared platform. Compositional approaches suggest providing an interface for each application in which the timing and resource requirements of the application is abstracted. Using such interfaces, the system level schedulability can be investigated in the integration phase via exploring whether the timing and resource requirements of each application is satisfied. Such an abstraction removes the need for having detailed, application-level knowledge such as tasks timing and resource specifications or the scheduling and resource sharing policies used within each application, for the system integrator.

This matter has attracted attention for uniprocessor platforms [24, 25, 26]. In the context of multiprocessors this issue has been studied for those platforms where each application is located on one processor in [14]. However, an application may not necessarily fit in one processor. A variant of such platforms where tasks are globally scheduled within each application has been investigated in [27].

1.1 Goal of the Thesis

Most of the multiprocessor real-time scheduling solutions assume that tasks are independent and do not share resources other than the CPU. Such an assumption does typically not hold for all platforms, especially looking at resource

constrained embedded systems with a limited amount of resources available where tasks have to share resources across the platform. The goal of this thesis is: *to enable mutual exclusive access to shared resources for real-time multiprocessor platforms such that the temporal correctness and real-time constraints of such systems are maintained, and further, to simplify integration of real-time applications on multiprocessor platforms.*

1.2 Research Challenges

In this section we list the research challenges identified in assessing the overall goal of the thesis.

1. Enable resource sharing for a semi-partitioned scheduling.

   In semi-partitioned scheduling a combination of partitioned and global scheduling is used where a processor is affected by both fixed assigned and migrating tasks. None of the existing resource sharing approaches could be used straight forwardly for resource handling in a hybrid system structure of semi-partitioned scheduling. A proper adjustment of the existing approaches is needed to handle resource sharing among tasks in such setups such that the overhead of the enabled resource handling protocol is bounded. Moreover, a schedulability analysis tailored to the enabled resource sharing approach is required allowing a system developer in the design phase of such systems to check schedulability. This requires that the priority-inversion bounds reflect the worst-case delays induced to tasks in such a hybrid structure.

2. Enable and further optimize compositional scheduling of independently-developed applications that use multiple processors.

   In the context of compositional scheduling, to facilitate and optimize the integration of real-time applications when migrating to a multiprocessor architecture, the timing and resource requirements of each application can be abstracted in an interface. Challenges arise when applications use multiple processors and may share resources with each other, which requires proper adjustments to the interfaces of each application.

3. Find alternatives for resource locking protocols to reduce the blocking bounds for multiprocessor platforms.

   By enabling resource sharing in a system, extra delays are induced to task response times which may endanger the temporal correctness of

real-time systems. Therefore, it is desirable to improve locking techniques to reduce priority-inversions and resource waiting time bounds. The question is whether there exist other locking techniques that can reduce such imposed delays?

1.3 Technical Contributions

In this section we present the contributions of this thesis that address the identified research challenges. To address the first challenge, that is enabling resource handling in the context of semi-partitioned scheduling, we have proposed and evaluated two different protocols for the purpose of synchronization among tasks that share resources. To address the second challenge, we have provided interfaces for applications that may share resources with other applications and can be allocated to more than one processor. The interfaces abstract the timing and resource requirements of each application such that when applications are integrated on one shared platform, the schedulability of the whole system can be investigated by only checking the timing requirements of each application. Further, we have proposed a multiple interface schema that can increase the possibility of finding a solution to integrate multiple applications on a shared platform. To address the third challenge, we have looked at other than existing solutions' alternatives for locking resources.

The focus of this thesis is partitioned scheduling platforms. Partitioned scheduling is attractive from an industrial point of view due to trivial implementations and low run-time overheads. Moreover, the commercial real-time operating systems support partitioned scheduling [28]. Fixed-priority partitioned scheduling is used by the AUTOSAR standard for automotive systems [29]. It also has been supported by all POSIX-compliant real-time operating systems such as VxWorks, QNX, LynxOS, etc. [30]. However, this does not imply that partitioned scheduling, is the best choice.

Personal contribution. The research presented in this thesis is done in collaboration with the University of Eindhoven. I am the main driver and the first author of all included papers. Prof. Thomas Nolte, Dr. Moris Behnam and Prof. Reinder J. Bril are my supervisors and contributed in reviewing the solutions and discussions. Dr. Farhang Nemati contributed in discussions and reviewing of Paper A.

1.3.1 Contribution 1: Resource Sharing in Semi-Partitioned Scheduling

Semi-partitioned scheduling has become a subject of recent interest in multiprocessor platforms due to better utilization results compared to conventional global and partitioned scheduling algorithms. Various task assigning techniques have recently been proposed in a semi-partitioned environment. However, a synchronization protocol for resource sharing among tasks in semi-partitioned scheduling has not yet been investigated. We propose two different synchronization protocols to enable resource handling in a semi-partitioned framework. The main challenge in this context is to serve the resource requests of tasks that migrate among processors. The first solution uses a centralized approach to handle resource sharing among such tasks. The second solution uses a decentralized approach. We provide the analysis for both approaches reflecting the blocking bounds under each approach. We evaluate and compare the two proposed solutions where we show that they are incomparable under specific system assumptions. The details of this contribution are presented in Paper A.

1.3.2 Contribution 2: Compositional Methods for applications on Multi-Core Platform

To simplify migration to a multiprocessor architecture, independently-developed subsystems/applications are abstracted with an interface. The interfaces are intended to ease the integration of such applications on a shared platform. Interfaces where the resource and timing requirements of the applications are provided relieves the system integrator from being aware of the details of scheduling and resource sharing policies used within each application in order to determine system-level schedulability. Instead, system schedulability can be checked through their interfaces at the time of integration on a shared multi-core architecture. This matter has been investigated for fixed-priority partitioned scheduled applications where each application is allocated on one processor. However, the applications may not fit in one processor only. Adopting the interfaces of applications when applications require more than one processor is not straightforward. For this purpose, we analyze the systems under such assumptions and derive a new interface that reflects the resource requirements of each application. Further, we propose partitioned and semi-partitioned approaches as two design choices for partitioning tasks of an application over a set of processors. The suggested

design method proposes a multiple interface configuration for each application according to the partitioning approach used for the application. By providing multiple interfaces for an application in the development phase, the chance of finding a feasible solution for application integration can be increased. The details of this contribution are presented in Paper B.

1.3.3 Contribution 3: Improved Locking Techniques

Various approaches can be utilized upon resource locking for mutually exclusive resource access in multiprocessor platforms. So far two conventional approaches exist for dealing with tasks that are blocked on a resource that is locked by tasks on a different processor. Either the blocked task performs a busy wait, i.e. spins, at the highest priority level until the resource is released, or it is suspended. Although both approaches provide mutually exclusive access to resources, they can introduce long blocking delays to tasks, which may be unacceptable for many industrial applications. The spin-based and suspension-based approaches can be viewed as the task waiting for a resource spins with a priority higher than any priority on the core or with a priority lower than any priority on the core. By such a view, we generalize the spin-lock model for the whole range between these two extremes. We propose a flexible spin-lock model for resource sharing in multiprocessor platforms in which the priority of the blocked tasks during spinning can be selected arbitrarily. Moreover, we provide the analysis for a range of the spin-lock priorities from the flexible model. Further, we compare a set of spin-lock approaches from the range provided by the model. We show by means of analysis-based comparison as well as case examples and experimental results that these solutions can provide a better performance for a specific set of tasks on the processor. The details of this contribution are presented in Papers C and D.

The relation between the research challenges, the contributions and the included papers is illustrated in Table 1.1.

	Challenge 1	Challenge 2	Challenge 3	Publications
Contribution 1	$\surd$			Paper A
Contribution 2		$\surd$		Paper B
Contribution 3			$\surd$	Paper C&D

Table 1.1: The relation between the research challenges, the contributions and the publications

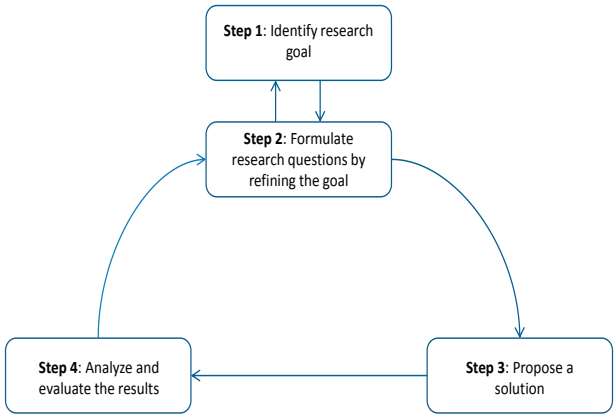

Figure 1.1: Research methodology

1.4 Research Method

The research methodology used in this thesis work is conformant with the steps proposed in [31]. The main steps of conducting the research here are as follows:

1. Identifying the research problem by studying the state of the art and defining the research goal.

2. Formulating the research questions and challenges by refining the research goal.

3. Proposing proper solutions for the addressed research questions.

4. Analyze and evaluate the proposed solution by means of formal proofs, examples and analysis-based simulation results.

Steps 2, 3, 4 are repeated until the desired results are achieved. Figure 1.1 depicts the aforementioned steps.

1.5 Outline of the Thesis

This thesis consists of 7 chapters and the outline of the rest of the thesis is organized as follows. Chapter 2 presents the most relevant background on the thesis. Chapter 3 summarizes the content of the thesis and identifies directions for the future works. The included papers are presented in Chapters 4 to 7.

Chapter 2

Background

2.1 Real-Time Systems

Real-time systems are systems for which correctness of the system functionality is not only dependent on the correctness of the results but also on the timeliness of the delivered results [32]. In other words, the correct results should be delivered within a certain time referred to as a deadline, so that the system is deemed real-time. Regarding the criticality of the results to be delivered within the deadlines, the real-time systems fall under two categories of *hard real-time* and *soft real-time* systems. In a hard real-time system, any deadline miss can lead to a system failure, so it is important that all results are delivered within the deadlines. While, in a soft real-time system, a degree of deadline miss can be acceptable. Deadline misses may only degrade the quality of service in this case.

A real-time system is usually composed of a set of recurrent tasks, i.e., a task execute in an infinite loop. In the task model, recurrency of a task is realized by its jobs. Each task is composed of an infinite sequence of its instances referred to as jobs. Moreover, each task is attributed with a deadline which is a time when the jobs of the task should finish their execution at latest. The maximum time that is needed for any job of a task to finish its execution, independent from the interference of any other task (jobs of any other task), is known as *worst-case execution time* of the task. When a task is ready to execute on a processor it is said that the task has *arrived* or is *released*. Tasks in a real-time system can be *periodic* or *aperiodic*. If jobs of a task arrive in exactly equal time intervals called *period*, the task is known as a periodic task whereas

the arrival pattern of an aperiodic task is not known. A variant of the aperiodic task with a touch of a periodic attribute is the *sporadic* task model. Sporadic tasks are aperiodic, however, the minimum inter-arrival time of the next job is known for such tasks. *Utilization* of a task is the portion of the processor bandwidth that is reserved for the task. The system utilization is the utilization of the task set running on the system, which is the sum of all tasks' utilizations. The *response time* of a task refers to the length of the interval between the task's arrival and finishing time. Usually, in real-time systems the *worst-case response time*s of tasks are of interest in order to explore the schedulability of the system, i.e., if all task deadlines are met or not. The worst-case response time of a task is the maximum response time of any job of the task.

A task set is *schedulable* if all tasks meet their deadlines, i.e., the worst-case response time of the task should be less than or equal to the deadline of the task. A *schedulability test* is a test that can determine whether a task set is schedulable under a set of system assumptions or not. A task set is *feasible* if a scheduling approach can be found to make the task set schedulable. For a task set to be feasible on a processor, the total utilization of the task set should not exceed one and accordingly m on an m processor platform.

2.2 Multiprocessor Platforms

With the emerge of multi-cores, multiprocessors have started to be used widely in embedded systems [33]. Multi-core platforms have found their way in manufacturing real-time systems due to their wide availability in the market [34] along with their high computing capacity. We refer to multiprocessors as a set of processing units that are connected to each other via a shared bus. All processors have access to a shared memory by means of the shared bus. The maximum access time for a processor to each memory location is similar (i.e. a *uniform memory access*). Moreover, multiprocessor platforms can be of type *identical* multiprocessors (also referred to as *symmetric* or *homogeneous*) or *heterogeneous* multiprocessors. In identical multiprocessors, task execution times are independent of which processor they execute on. In contrast, in *heterogeneous* platforms each processor may have a different speed. Therefore, task execution requirements are proportionally scaled up or down with the processor speed they are running on, by being assigned to slower or faster processors.

2.3 Multiprocessor Real-Time Scheduling

Two fundamental scheduling approaches exist for multiprocessor platforms: partitioned and global scheduling. Resource reservation techniques has introduced a third type that is called a hybrid scheduling which combines the two partitioned and global scheduling approaches on the same platform.

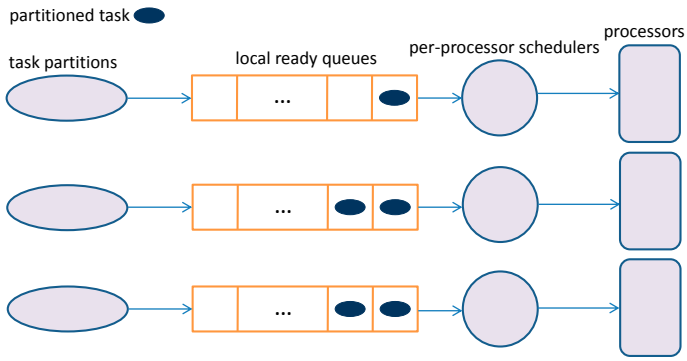

Figure 2.1: Partitioned Scheduling

2.3.1 Partitioned Scheduling

Under a partitioned scheduling approach, tasks are assigned to fixed processors during the design-time and all jobs of each task execute on the same processor to which the task is assigned, during run-time. Each processor uses a uniprocessor scheduling approach such as *Rate-Monotonic* (RM) or *Earliest Deadline-First* (EDF) [35]. Each processor uses a separate scheduler and local ready queue to independently schedule the tasks on the processor as can be seen in Figure 2.1. Schedulers on different cores on the multiprocessor platform may use identical or different scheduling algorithms. One major privilege of using a partitioned approach is to reuse the well-known uniprocessor scheduling approaches. Other advantages of using partitioned scheduling are the implementation simplicity and run-time efficiency due to preventing tasks from migrating among cores. However, one major weak point of this approach is the partitioning problem which in fact is a bin-packing problem that

is known to be NP-hard in the strong sense [36]. In other words, finding an optimal solution to allocate tasks to processors cannot be done in a polynomial time. Therefore, heuristic algorithms are used to partition tasks among processors. Another disadvantage of partitioned scheduling is that processors may not be fully utilized. However, most of the real-time operating systems have a preference to use partitioned scheduling due to its uniprocessor legacy, trivial implementation complexity and POSIX-compliant real-time standard [37]. Partitioned-EDF (P-EDF) is an example of a partitioned scheduling approach.

2.3.2 Global Scheduling

Under a global scheduling approach, one global scheduler schedules all tasks to the processors from a unique ready queue during run-time as shown in Figure 2.2. Under this scheduling approach, jobs of tasks are allowed to migrate among processors. A job of the task that is preempted on a core, may be resumed on a different core. At any time at most m of the highest priority tasks are selected and scheduled on an m processor platform. Uniprocessor scheduling protocols such as RM and EDF [38] are not anymore optimal on the multiprocessor platforms. Many works have provided efficient analysis for global scheduling [39, 40, 41, 42, 43]. New scheduling approaches have been proposed for global scheduling, such as the proportionate fair (*pfair*) scheduling approach [44, 45, 46], that are optimal under specific assumptions, such as no migration, preemption, and scheduling overhead. However, they often introduce a high level of run-time overheads.

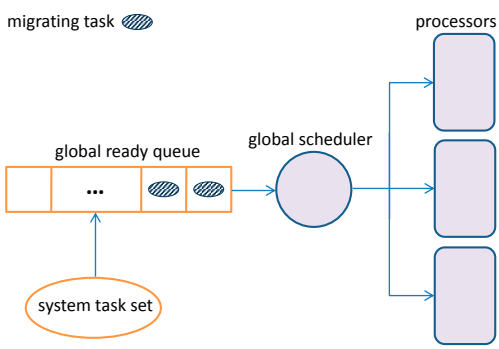

Figure 2.2: Global Scheduling

2.3.3 Hybrid Scheduling

In most embedded systems, due to constraints on the available resources in the system, resource reservation approaches that can efficiently utilize system resources are of significance. These approaches usually use a hybrid approach combining global and partitioned scheduling on the same platform. Semi-partitioned scheduling is one of such approaches [19]. To utilize processors in a better way, the semi-partitioned approach suggests to further utilize the remaining capacity on each processor to schedule the tasks that could not fit on any processor. Since any of the remaining tasks could not fit in any processor, typically, their execution has to be split among multiple processors. In semi-partitioned scheduling similar to the partitioned scheduling each processor has a separate scheduler and local ready queue to schedule the partitioned tasks on each processor. However, the tasks which are split among processors can migrate and be scheduled on different processors as shown in Figure 2.3. So far, various task assignment techniques have been proposed for the semi-partitioned approach [20, 21, 22, 23]. Guan et al. [23] showed that the utilization bound of task sets on each processor can be increased as high as the utilization bound of Liu and Layland's RM scheduling for an arbitrary task set.

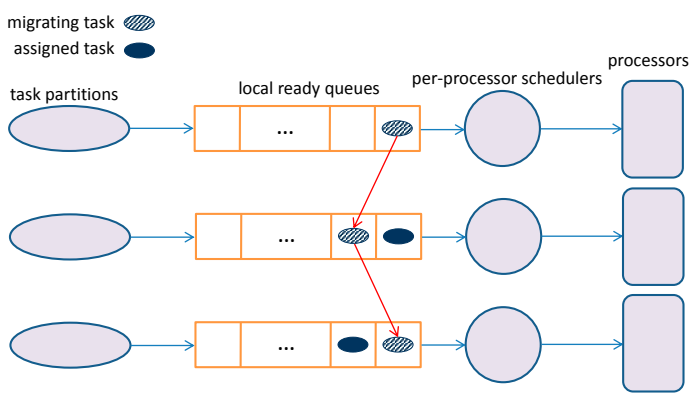

Figure 2.3: Semi-partitioned Scheduling

Cluster-based scheduling approaches, are another category of scheduling approaches which can be generalized to partitioned and/or global scheduling. Under a cluster-based approach, tasks are assigned to clusters which consist of a set of processors and are scheduled globally within a cluster as shown in Figure 2.4. Cluster-based scheduling maps to partitioned scheduling when m clusters exist in the system where m is the number of processors, and cluster-based scheduling is equal to global scheduling when one cluster exists in the system, only. Figure 2.4 shows a 2-cluster system where the number of processors in each cluster is 3. Cluster-based scheduling can be classified into two types: *physical* or *virtual*. Under a physical cluster-based approach [47], each cluster is assigned to a fixed set of processors, whereas under a virtual cluster-based approach [48] the clusters are assigned to the processors dynamically.

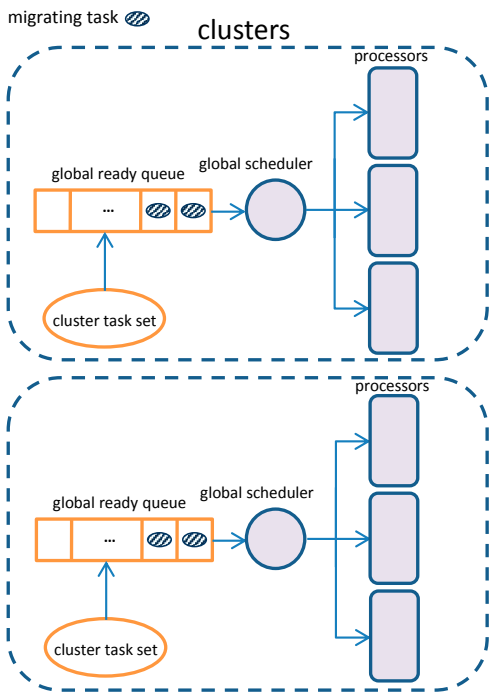

Figure 2.4: Cluster-based Scheduling

2.4 Hierarchical Scheduling

Hierarchical scheduling is an approach used to schedule tasks in a hierarchy manner. On the higher level, a global scheduler schedules subsystems and on the lower level, a local scheduler schedules tasks within the subsystem using a local scheduling policy. Figure 2.5 shows a two-level hierarchical system. The main objective of hierarchical scheduling is to provide isolation among a set of subsystems/applications that are supposed to be scheduled on the same platform. In hierarchical scheduling, for each subsystem the amount of resources that are needed to schedule the subsystem is dedicated. In this way, isolation in execution of tasks between subsystems are provided if subsystems do not share resources other than processors. This prevents the propagation of errors among subsystems. Hierarchical scheduling can be applied to both uniprocessor platforms [49, 50, 51, 52] as well as to multiprocessor systems [47, 48, 53].

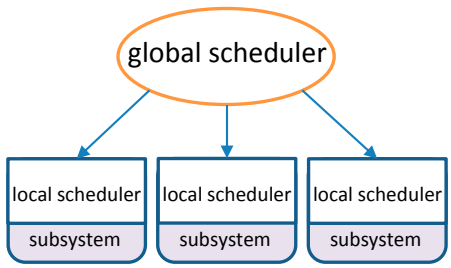

Figure 2.5: Hierarchical Scheduling

2.5 Real-Time Locking Protocols

Many scheduling approaches assume that tasks are independent and do not share any resources but the processors. This assumption implies that the m highest priority ready tasks will run on an m processor platform. However, this assumption is not always true especially in embedded systems where a set of limited resources are available. Therefore tasks in such systems may have to share resources such as queues, buffers, data structures or physical resources such as I/O devices with each other. Concurrent access to the shared resources need to be synchronized in such systems to avoid conflicts or data corruption.

One solution for synchronizing access to shared resources are *locks*. An alternative approach, however, is the lock-free synchronization approach. In the lock-free approach [54, 55], tasks/operations try to access the shared resources, until they succeed. The convenience of using lock-free approaches is that it does not require the support by the operating system and since no lock is used, thus no priority inversion happens. However, since the number of retries cannot be easily bounded, this approach may not be the best choice for real-time applications where predictability is essential. In this thesis, we therefore focus on lock-based synchronization approaches.

In a multiprocessor platform two types of resources exist: *local resource*s and *global resource*s. Local resources are resources that are used only by tasks on the same processor whereas global resources are used by tasks on more than one processor. Typically, local resource sharing is handled by local resource sharing protocols such as the *Priority Ceiling Protocol* (*PCP*) [7] and *Stack Resource Policy* (*SRP*) [6]). If the execution of a task that is ready, is delayed due to lower priority tasks on the same processor, the task incurs blocking also referred to as *direct blocking*. Blocking can occur due to tasks using local or global resources. A task may experience blocking due to requesting the same resource that is already locked by a lower priority task on the same core (since a task never gives up a resource that it has locked) or just being prevented from being scheduled since a lower priority task has become non-preemptable (typically due to locking a resource). This implies that even a task that does not request any resource might be blocked. A task may experience blocking due to lower priority tasks that lock local resources, which we refer to as *Local Blocking due to Local Resources* (*LBL*), or due to global resources, which we refer to as *Local Blocking due to Global Resources* (*LBG*). Priority-inversions, which happen when a higher priority job is prevented to be scheduled due to lower priority jobs for an unbounded amount of time, can endanger temporal correctness of real-time systems.

In a multiprocessor platform, besides the delay incurred to a task due to direct blocking, a task may further experience delay due to waiting for a remotely held resource (i.e., by a task on a different processor). This type of delay, which is referred to as acquisition delay, must be accounted for in a task response time. Since the acquisition delay is due to waiting for a resource, it is also referred to as *remote blocking*. When a task is waiting to acquire a resource, it is also said that the task is *blocked on the resource*.

For each resource in the system, a unique queue is dedicated to enqueue the tasks waiting for the resource. For global resources such queues are globally defined. Tasks that are blocked on local resources are enqueued in the re-

lated local resource queue (Figure 2.6). To serve the tasks that have requested the same resource, they should be placed (or put a place holder) in the related resource queue. The queues can be *priority-based* or *FIFO-based* or a combination of both. Priority-based queues are in favor of high priority tasks, since whenever a higher priority task is added to the queue it is placed ahead of lower priority tasks. However, this may cause starvation for lower priority tasks. Since higher priority tasks are prior to use the resource, new instances of the same higher priority tasks may release and place themselves in the same resource queue before a lower priority task. FIFO-based queues treat tasks in a first-come first-serve manner. FIFO-based queues gives chance to lower priority tasks that are placed sooner than higher priority tasks in the queue to acquire the resource. However, a higher priority task has to wait for all lower priority tasks in a worst-case scenario if the size of the queue is not bounded.

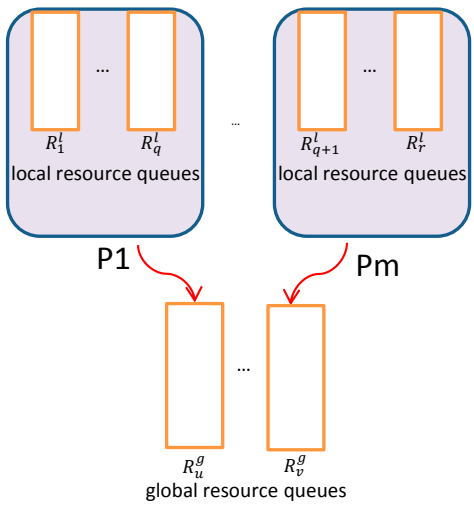

Figure 2.6: Local and global resource queues

When a task is remotely blocked in a multiprocessor platform, the task may either spin and perform a busy-wait loop or it may suspend and release the processor. In uniprocessor platforms, suspending is the only option since if tasks busy wait, no other task can progress. Neither one of the spin-based

and suspension-based approaches dominates the other, i.e., there might be systems that are schedulable under the spin-based approach and not under the suspension-based approach and vice versa.

Under a spin-based approach, if the waiting times for resources are long, i.e., long *critical section*s, the task that is non-preemptively spinning to acquire a remotely held resource, wastes the processor bandwidth for the whole waiting period. Whereas, under a suspension-based approach, the task waiting for the resource lets other tasks on the core carry additional work. However, by letting other tasks to execute on the core when the task is waiting for a remotely held resource, more resource request may be issued by those tasks running on the core. These requests can later contribute in extra delay to the waiting task under a suspension approach.

If critical sections are short, spin-based approaches are preferred since context switch overheads are smaller compared to when suspension-based approaches are used where tasks are more subject to suspending and resuming overhead. However, if critical sections are long, the processor time is wasted by spinning compared to a suspension-based approach that lets other task to run on the processor. However, specifying a critical section to be long or short is user-dependent [11]. The purpose of a real-time locking protocol is to ensure that delays incurred due to tasks using resources are bounded and can be known in advance. Such maximum delays should be considered in the response time of the tasks for the purpose of system schedulability test. In the following subsections, we briefly present the most relevant locking protocols for multiprocessor platforms.

2.5.1 Spin-Based Protocols

In spin-based locking protocols, when a task that is waiting for a global resource spins, typically its priority is raised in an atomic operation to a higher priority than the task itself. In the traditional spin-lock approach, the priority of a task is raised to the highest priority on the core and the task becomes non-preemptable. The task provides a place-holder in the global queue of the related resource and waits for its turn to acquire the resource. The task locks the resource when the task is at the head of the queue and the resource is available, i.e., not locked by any task.

MSRP Synchronization Protocol

The Multiprocessor Stack Resource Policy (MSRP) [9] is a spin-based locking protocol introduced by Gai et al. MSRP is an extension of the SRP [6] protocol for multiprocessors. MSRP has been proposed under partitioned-EDF (P-EDF) scheduled systems. Tasks share both local and global resources. Local resource sharing is handled by SRP. Since a task that is blocked on a resource (i.e., waiting to acquire the resource) spins non-preemptively and is wasting the processor resources, the lock holding tasks need to release the resource as soon as possible. Therefore, tasks execute critical sections non-preemptively. Global resource queues are FIFO-based under MSRP and nesting of global resource requests is not allowed. Tasks become non-preemptive when executing a critical section on a global resource. In [10], Gai et al. have compared MSRP with the Multiprocessor Priority Ceiling Protocol (MPCP) [16] from an implementation point of view. They have concluded that when critical sections are short, MSRP outperforms MPCP, while for longer critical sections, MPCP outperforms MSRP. They have pointed out that due to the wasting of processor time under MSRP, spin-locking is expensive for industry such as automotive applications compared to the MPCP approach. However, MSRP is simple to implement compared to MPCP and it allows for sharing of the stack space of tasks while MPCP does not.

M-BWI Synchronization Protocol

The Multiprocessor Bandwidth Inheritance (M-BWI) protocol proposed by Faggioli et al. [17], is a spin-based protocol for multiprocessor platforms. M-BWI is an extension of the Bandwidth Inheritance (BWI) [56] protocol that combines the constant bandwidth server [57] with a priority inheritance protocol [7] to provide bandwidth isolation for open systems. M-BWI allows existence of both hard and soft real-time tasks simultaneously on the platform. This protocol has been presented for multiprocessor systems under global scheduling. However the M-BWI protocol is neutral to the underlying scheduling approach and it can be implemented in both global and partitioned scheduled systems. M-BWI does not require any information regrading temporal parameters of tasks such as tasks' worst-case execution times and critical section lengths, which makes it suitable for open systems where tasks can dynamically be added or removed. However, if such information can be estimated for tasks, an upper bound for the delays to tasks is possible to be provided. The resource queues used in M-BWI are FIFO-based. Under the M-BWI, each task is assigned to a server which provides a limited amount of the processor bandwidth

for the task's execution. Under M-BWI approach, in case the budget of the server related to a resource holding task is exhausted before the task releases the resource or if the task is preempted, the task can migrate to other processors and use the budget of the servers in which their tasks are waiting for the same resource.

MrsP Synchronization Protocol

The (Multiprocessor resource sharing Protocol (MrsP) [15] proposed by Burns and Wellings, is a spin-based protocol for multiprocessor platforms which is a variant of PCP for single processor. MrsP is similar to MSRP with one significant difference; the tasks that are spinning (i.e., are waiting for a resource) can use their spin time on behalf of other tasks on other cores which have locked the resource but been preempted on their core. The preempted task can then migrate to another processor on which a task is spinning (for the same resource). The migrated task will give control to the spinning task to execute its interrupted critical section on behalf of itself. When the migrated task returns to its assigned processor and access the processor, it finds that its critical section has been executed. This approach is inspired by a method called helping used in [58]. MrsP is general purpose but has been developed for fixed-priority partitioned scheduling systems. Resource queues are FIFO-based under MrsP. Similar to PCP, each resource is affiliated with a ceiling on each core which is the highest priority among the tasks that use the resource on the core. As a results of using separate ceilings for every resource on a core, critical section executions are preemptive. When a task spins to acquire a resource, its priority is boosted to the ceiling priority of the resource. Moreover, the task continues executing with the ceiling priority of the resource after it accesses the resource. The authors concluded that for a low priority task, MrsP cannot perform better than a non-preemption approach similar to MSRP. However, they showed that the same low priority task may have a significant improvement in its response time under another approach which is similar to MrsP with the only difference that migration does not happen. The authors showed that for a high priority task, MrsP may have a better performance compared to the variant of MrsP without migrating, however, it still cannot outperform the non-preemption approach.

2.5.2 Suspension-Based Protocols

In suspension-based locking protocols, when a task is waiting for a resource, it suspends and releases the processor. The task is inserted to the queue related to the resource and it is waiting there to acquire the resource. As a matter of suspension, the core can execute a workload related to other tasks on the core. When the task is at the head of the resource queue and the resource is released, the task is granted access to the resource and it locks the resource. At this point, it depends on the policy used in the operating system how to raise the priority of the task so the task runs on the core. As an example, if the priority of the task that has been granted to the resource is raised as a function of its original priority, e.g., the highest priority on the core plus the task's original priority, then the task may have to wait for higher priority tasks that also have been granted to other resources on the core if any, to execute first. If the task has the highest raised-priority among the tasks that are also granted to their resources, it can immediately start running when the processor is released. On the other hand, if the priority of the task that is granted a resource is raised to the highest possible priority level on the core, it will be served in a FIFO manner if more than one such granted access task exists on the core.

DPCP Synchronization Protocol

The Distributed Priority Ceiling Protocol (DPCP) is a suspension-based resource sharing protocol designed for distributed systems [8], which first was introduced by Rajkumar et al. in [59]. The protocol relies on message passing between processors and uses remote procedure calls. In DPCP a job executes its local and normal (i.e., non-critical) sections on its assigned processor while its global critical sections may execute on processors other than its allocated processor. Processors which global critical sections are executed on, are called synchronization processors. All critical sections of a specific global resource is bound to one processor, however, there may exist multiple synchronization processors in the system. Under DPCP, critical sections are executed at global priority ceiling of the resource. The priority of a task requesting global resources is raised within its critical section to a priority higher than any task in that processor. Global critical sections are preemptive, but can be preempted by higher priority global critical sections, only. The main advantage of DPCP is that it allows nesting of global critical sections as long as locks do not exceed processors boundaries, which is achieved under DPCP by bounding critical sections of the same global resource to the same synchronization processor.

DPCP has been proposed for fixed-priority partitioned scheduling (RM) systems and global resource queues are priority-based.

MPCP Synchronization Protocol

The Multiprocessor Priority Ceiling Protocol (MPCP) proposed by Rajkumar et al. [16] is an extension of PCP for multiprocessor platforms. MPCP is a suspension-based synchronization protocol. Similar to the DPCP synchronization protocol, under MPCP, the priority of a task within a global critical section is raised higher than any priority in the system. However, since the global critical sections have been defined to be preemptive by execution of global critical sections of higher priority tasks, thus the boosted priority is further raised to the global ceiling of the resource. MPCP has been developed for fixed-priority partitioned scheduling (RM) and global resource queues are priority-based, similar to DPCP. Under the MPCP synchronization protocol, by nesting global critical sections, the blocking times increase rapidly. Moreover, the worst-case blocking times seems to be larger under MPCP compared to the DPCP approach. However, MPCP has a more efficient implementation compared to the DPCP approach where overhead of remotely execution of global critical sections and communication delays needs to be accounted for. A spin-based variant of MPCP has been proposed in [28].

FMLP Synchronization Protocol

The Flexible Multiprocessor Locking Protocol (FMLP) proposed by Block et al. [11] is a synchronization protocol that combines suspension-based and spin-based approaches for different type of resources in the system. FMLP has been developed for both partitioned (partitioned-EDF) and global scheduling (global-EDF and pfair PD^2 [60]). Under the FMLP protocol, resources are divided into long and short resources where the definition of a long and short resource is user-defined. Tasks use a suspension-based approach when they are remotely blocked on long resources whereas they use a spin-based approach when they get blocked on short resources. Nested global critical sections are supported under the FMLP protocol. Under FMLP, the priority of tasks (more precisely, jobs of tasks) holding long global resources are boosted to the highest priority in the system. The priority boosting is not needed for short resources holding tasks since they are performing a non-preemptive spin lock. Tasks execute within both long and short global critical sections non-preemptively.

FMLP uses *resource groups* to prevent the deadlock problem which can

happen due to nesting of requests. Each group of resources is protected by a group lock. To acquire a resource, a task should first acquire the resource's group lock.

OMLP Synchronization Protocol

The $O(m)$ Locking Protocol (OMLP) proposed by Brandenburg and Anderson [13], is a suspension-based synchronization protocol. OMLP has been denoted as a *suspension-oblivious* protocol [13]. It has been denoted that under a suspension-oblivious protocol, the waiting time of tasks are accounted as an additional execution, i.e., suspended jobs are assumed to occupy the processor. In contrast of suspension oblivious protocols, normal suspension-based protocols have been referred to as *suspension-aware* protocols. Further, OMLP has been referred to as an *asymptotically optimal* protocol [13]. Asymptotically optimal denotes that the blocking times is confined to a fixed factor of blocking. OMLP has been developed for both partitioned and global scheduling.

In the global OMLP, resource queues are a combination of both FIFO and priority-based. FIFO queues are of length m (i.e., number of processors). First the tasks that are blocked on a global resource are enqueued in the FIFO-based queue until the FIFO-based queue is filled. Then they are inserted to the priority-based queue. The idea behind the global OMLP design is that the lower priority tasks are prevented from starvation since they have a chance to be located to the FIFO-based queue. On the other hand, higher priority tasks may only be punished for less than m lower priority tasks' critical sections length, in the worst case.

Under the partitioned OMLP, to acquire a global resource, a task first have to acquire a unique token dedicated for each processor. Only one token exist for the resources used in each processor. Under the partitioned OMLP, the number of the tasks that can cause priority inversion in the system is limited due to the priority boosting technique that is used for a token holding task.

Later, the same authors extended OMLP to clustered scheduling [61], where they have simplified the queue type to only a FIFO-based queue for each global resource. They have proposed a new technique called *priority donation*. Under the priority donation technique, a higher priority task may suspend and donate its priority level to a lower priority task that is requesting a resource in order to accomplish the lower priority task's access. By using the priority boosting technique a task might be preempted frequently while using the priority donation technique, each task might be preempted at most once.

P-PCP Synchronization Protocol

Parallel PCP (P-PCP) has been proposed by Easwaran and Andersson [18] and is a suspension-based synchronization protocol. In this work, the authors provided response time schedulability analysis for a multiprocessor variant of the PIP resource sharing protocol under fixed-priority global scheduling as well as for the P-PCP which they have proposed.

Under P-PCP, for tasks that use resources, the interference from lower priority tasks and the amount of parallel executions can be traded-off. The trade-off level can be adjusted by a predefined tuning parameter. For a task, a higher value for this tuning parameter increases the chance of more lower priority tasks to be executed at a priority higher than the tasks base/original priority (referred to as *effective priority*). Therefore, the interference to the task is increased. On the other hand, a higher value of the tuning parameter will increase the parallelism on a multiprocessor platform.

MSOS Synchronization Protocol

The Multiprocessor Synchronization Protocol for Open Systems (MSOS) has been proposed by Nemati et al. [14] which they called later *MSOS-FIFO*. MSOS-FIFO is a suspension-based synchronization protocol developed for resource handling among real-time applications in open systems where applications can be added or removed at run-time. MSOS-FIFO has been developed for partitioned scheduling. MSOS-FIFO enables a *compositional schedulability test* for a set of independently-developed real-time applications that are integrated and co-execute on the same platform. Under a compositional schedulability test, the schedulability of the whole system is checked by checking the schedulability requirements of each application which is usually abstracted in an interface provided for the application.

In this work, each processor hosts one application, i.e., all tasks related to the same application are assigned to the same processor and applications do not share processors. Global resource queues are FIFO-based. If a task within an application blocks on a global resource, a placeholder is located for its processor in the queue and the task is inserted to a local waiting queue for that specific resource dedicated to the processor. When the related processor is at the head of the global FIFO-based queue, the task at the related local resource queue will lock the resource. Both FIFO-based and priority-based local resource queues have been investigated in this work.

When a task on a processor requests a global resource, its priority is boosted immediately to its original priority plus the highest priority on that processor.

In this way, the task that has locked a resource can be delayed only by higher priority tasks that also have been granted access to other resources. Later the same authors have extended MSOS for priority-based global resource queues where the applications may have a priority versus each other [62].

Chapter 3

Conclusions

3.1 Summary

In this thesis, we have enabled resource sharing in a family of multiprocessor platforms that did not yet provide such functionality. We have provided the analysis for such systems that guarantees that the delays incurred to tasks as well as to applications due to resource sharing is bounded. To further decrease such incurred delays due to resource sharing, we also looked at alternative techniques for locking resources in multiprocessor platforms.

In Paper A, we provided two resource handling approaches for semi-partitioned scheduling. Under semi-partitioned scheduling some tasks in the system are split over multiple processors while the rest are assigned and executed on only one processor. From a scheduling point of view, semi-partitioned scheduling is a hybrid approach. In this paper, we proposed two resource handling approaches. In the first approach, which is a centralized approach, all critical sections of a split task execute on a predefined processor. In the second approach, the critical section executes on the processor where the request occurs. For both approaches we provided the analysis where we showed that the blocking terms are bounded. Based on a comparative evaluation, we concluded that neither of the two approaches dominates the other.

In Paper B, we enabled compositional integration of multiple independently-developed real-time applications where each application may be allocated to more than one processor. Usually, to ease integration, compositional approaches offer the possibility to check the whole system schedulability

using particular scheduling requirements that are abstracted in the interface provided for each application. To achieve this, we updated the interface of each application with the applications resource and scheduling requirements by considering that the application may reside on multiple processors. We further offered multiple interface configurations for each application based on the technique used to partition the application over processors to increase the chance of finding a feasible solution for system integration.

In Papers C and D, we investigated alternatives for existing locking techniques with the purpose of decreasing the delay incurred to tasks due to resource sharing. We proposed a flexible spin-lock model, where the priority at which a task is spinning to acquire a resource can be selected arbitrary from the range proposed by the model. We investigated three spin-lock priority levels for spinning from the proposed range: (i) task's original priority, (ii) highest ceiling of global resources on the core, and (iii) highest ceiling of global and local resource on the core. We provided the analysis for the range of spin-lock approaches that use spin-priority levels from highest ceiling of global resources up to the highest priority on a processor. We showed, by means of example cases, that the first alternative and the classical suspension-based approach as well as the second alternative and the classical spin-lock approach are incomparable. Further, we mathematically proved that the third alternative dominates the classical spin-lock approach, and the second and the third approaches are incomparable, which was confirmed by the experimental results, as well. We provided evaluation results for the second and third alternative where we compared their performance versus the classical spin-lock approach. Finally, we showed that if the (ii) and (iii) cannot make a task set schedulable, there may exist a solution with a spin-priority between these two spin-lock approaches.

3.2 Future Works

Several directions for future work are conceivable. We briefly mention them below.

1. *Blocking-aware partitioning.* The schedulability performance of partitioned scheduling systems can highly be affected by the partitioning techniques used to allocate tasks to processors. When tasks share resources in the system, an extra delay is incurred to tasks due to blocking. How to allocate tasks to processors can influence the incurred delays, thus a blocking-aware partitioning approach can effectively increase the system performance. In a blocking-aware partitioning approach, tasks

are assigned to processors in such a way that the delays incurred due to resource sharing decreases, e.g., by allocating tasks that share the same resource to the same processor. Many scheduling approaches such as semi-partitioned scheduling have not considered such blocking-aware partitioning. Blocking-aware partitioning can be investigated under different locking approaches, e.g., spin-lock approaches from our proposed flexible spin-lock model in Paper C.

2. *Global intra-application level scheduling*. We investigated resource sharing for compositional real-time applications where an application may be allocated to more than one processor in Paper B. However, still the applications did not share processors. We would like to lift this assumption and enable compositionality for open-systems where applications/subsystems may share processors. This means that a processor may host more than one application. Moreover, we would like to further relax the system model assumptions and use global scheduling for inter-application scheduling and global or partitioned scheduling for intra-application scheduling. The application interfaces are required to contain the resource and scheduling requirements of an application to achieve compositional scheduling.

3. *Optimizing spin-lock approaches.* Towards optimizing the spin-lock approaches to increase task set schedulability and achieve shorter response times and induced jitter, more steps need to be elaborated. Spin-lock priorities can further be investigated in the following steps: (i) per processor, (ii) per task, (iii) per resource, and (iv) per resource access. In Paper C and D, we looked at the first alternative. Exploring further steps may lead us to an optimal solution or better heuristics.

Bibliography

[1] T.P. Baker. Stack-based scheduling for realtime processes. *Real-Time Systems*, 3(1):67–99, Apr. 1991.

[2] T.P. Baker. A comparison of global and partitioned EDF schedulability tests for multiprocessors. Technical report, In International Conference on Real-Time and Network Systems, 2005.

[3] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. In Taylor & Francis, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, pages 30.1–30.19. Chapman Hall/CRC, Boca, 2004.

[4] U. Devi. Soft real-time scheduling on multiprocessors. In *PhD thesis, available at www.cs.unc.edu/~anderson/diss/devidiss.pdf*, 2006.

[5] J. M. López, J. L. Díaz, and D. F. García. Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real-Time Syst.*, 28(1):39–68, October 2004.

[6] T. Baker. Stack-based scheduling of real-time processes. *Journal of Real-Time Systems*, 3(1):67–99, 1991.

[7] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, Sep. 1990.

[8] R. Rajkumar. *Synchronization in Real-Time Systems: A Priority Inheritance Approach*. Kluwer Academic Publishers, 1991.

[9] P. Gai, G. Lipari, and M. Di Natale. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In 22^{nd} *IEEE Real-Time Systems Symposium, (RTSS).*, pages 73–83, Dec 2001.

[10] P. Gai, M. Di Natale, G. Lipari, A. Ferrari, C. Gabellini, and P. Marceca. A comparison of MPCP and MSRP when sharing resources in the janus multiple-processor on a chip platform. In 9^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 189–198, May 2003.

[11] A. Block, H. Leontyev, B.B. Brandenburg, and J.H. Anderson. A flexible real-time locking protocol for multiprocessors. In 13^{th} *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 47–56, Aug. 2007.

[12] B.B. Brandenburg and J.H. Anderson. An implementation of the PCP, SRP, D-PCP, M-PCP, and FMLP real-time synchronization protocols in LITMUS^{RT}. In 14^{th} *IEEE Intl. Conf. on Embedded and Real-Time Computing Sys. and Applications (RTCSA)*, pages 185–194, Aug. 2008.

[13] B.B. Brandenburg and J.H. Anderson. Optimality results for multiprocessor real-time locking. In 31^{st} *IEEE Real-Time Systems Symposium (RTSS)*, pages 49–60, Dec. 2010.

[14] F. Nemati, M. Behnam, and T. Nolte. Independently-developed real-time systems on multi-cores with shared resources. In 23^{rd} *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 251–261, Jul. 2011.

[15] A. Burns and A.J. Wellings. A schedulability compatible multiprocessor resource sharing protocol – MrsP. In 25^{th} *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 282–291, July 2013.

[16] R. Rajkumar. Real-time synchronization protocols for shared memory multiprocessors. In 10^{th} *International Conference on Distributed Computing Systems*, pages 116–123, may-1 jun 1990.

[17] D. Faggioli, G. Lipari, and T. Cucinotta. The multiprocessor bandwidth inheritance protocol. In 22^{nd} *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 90–99, July 2010.

[18] A. Easwaran and B. Andersson. Resource sharing in global fixed-priority preemptive multiprocessor scheduling. In 30^{th} *IEEE Real-Time Systems Symposium (RTSS)*, pages 377–386, Dec 2009.

[19] J.H. Anderson, V. Bud, and U.C. Devi. An EDF-based scheduling algorithm for multiprocessor soft real-time systems. In 17^{th} *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 199–208, Jul. 2005.

[20] S. Kato and N. Yamasaki. Portioned static-priority scheduling on multiprocessors. In *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–12, Apr. 2008.

[21] S. Kato and N. Yamasaki. Semi-partitioned fixed-priority scheduling on multiprocessors. In 15^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 23–32, Apr. 2009.

[22] K. Lakshmanan, R. Rajkumar, and J. Lehoczky. Partitioned fixed-priority preemptive scheduling for multi-core processors. In 21^{st} *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 239–248, Jul. 2009.

[23] N. Guan, M. Stigge, Wang Yi, and Ge Yu. Fixed-priority multiprocessor scheduling with Liu and Layland's utilization bound. In 16^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'10)*, pages 165–174, Apr. 2010.

[24] M. Behnam, I. Shin, T. Nolte, and M. Sjödin. SIRAP: A synchronization protocol for hierarchical resource sharing in real-time open systems. In *Proceedings of the* 7^{th} *ACM & IEEE International Conference on Embedded Software (EMSOFT)*, pages 279–288. ACM, October 2007.

[25] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In 27^{th} *IEEE Real-Time Systems Symposium (RTSS)*, pages 389–398, 2006.

[26] N. Fisher, M. Bertogna, and S. Baruah. The design of an EDF-scheduled resource-sharing open environment. In 25^{th} *IEEE Real-Time Systems Symposium (RTSS)*, pages 83–92, 2004.

[27] F. Nemati and T. Nolte. Resource sharing among real-time components under multiprocessor clustered scheduling. *Real-Time Systems*, 49(5):580–613, 2013.

[28] K. Lakshmanan, D. de Niz, and R. Rajkumar. Coordinated task scheduling, allocation and synchronization on multiprocessors. In 30^{th} *IEEE Real-Time Systems Symposium (RTSS)*, pages 469–478, Dec. 2009.

[29] AUTOSAR release 4.0. 2012, http://www.autosar.org.

[30] A. Wieder and B.B. Brandenburg. Efficient partitioning of sporadic real-time tasks with shared resources and spin locks. In 8^{th} *IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 49–58, June 2013.

[31] M. Shaw. The coming-of-age of software architecture research. In 23^{th} *International Conference on Software Engineering, (ICSE)*, pages 656–, 2001.

[32] John A. Stankovic and K. Ramamritham, editors. *Tutorial: Hard Real-time Systems*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1989.

[33] A.C. Sodan, J. Machina, A. Deshmeh, K Macnaughton, and B Esbaugh. Parallelism via multithreaded and multicore cpus. *Computer*, 43(3):24–32, March 2010.

[34] J. Child. Multicore processing becomes the new mainstream. *COTS*, April 2010.

[35] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of ACM*, 20(1):46–61, January 1973.

[36] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[37] IEEE(2003). IEEE standard for information technology - standardized application environment profile (AEP) - POSIX realtime and embedded application support. number std 1003.13-2003. IEEE computer society.

[38] Sudarshan K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, 1978.

[39] T.P. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In 24^{th} *IEEE Real-Time Systems Symposium (RTSS)*, pages 120–129, Dec 2003.

[40] S. Baruah. Techniques for multiprocessor global schedulability analysis. In 28^{th} *IEEE International Real-Time Systems Symposium, (RTSS)*, pages 119–128, Dec 2007.

[41] Sanjoy Baruah and T. Baker. Schedulability analysis of global edf. *Real-Time Systems*, 38(3):223–235, 2008.

[42] M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of edf on multiprocessor platforms. In 17^{th} *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 209–218, July 2005.

[43] M. Bertogna and M. Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In 28^{th} *IEEE International Real-Time Systems Symposium (RTSS)*, pages 149–160, Dec 2007.

[44] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. In 25^{th} *Annual ACM Symposium on Theory of Computing (STOC)*, pages 345–354, New York, NY, USA, 1993. ACM.

[45] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15:600–625, 1996.

[46] J. Anderson, P. Holman, and A. Srinivasan. Fair scheduling of real-time tasks on multiprocessors. In *Handbook of Scheduling*, 2005.

[47] J.M. Calandrino, J.H. Anderson, and D.P. Baumberger. A hybrid real-time scheduling approach for large-scale multicore platforms. In 19^{th} *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 247–258, Jul. 2007.

[48] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 181–190, July 2008.

[49] G. Lipari and S. Baruah. A hierarchical extension to the constant bandwidth server framework. In 7^{th} *IEEE Real-Time Technology and Applications Symposium (RTAS)*, pages 26–35, 2001.

[50] Z. Deng and J. W S Liu. Scheduling real-time applications in an open environment. In 18^{th} *IEEE Real-Time Systems Symposium (RTSS)*, pages 308–319, Dec 1997.

[51] R.I. Davis and A. Burns. Hierarchical fixed priority pre-emptive scheduling. In 26^{th} *IEEE International Real-Time Systems Symposium (RTSS)*, pages 389–398, Dec 2005.

[52] F. Zhang and A. Burns. Analysis of hierarchical EDF pre-emptive scheduling. In 28^{th} *IEEE International Real-Time Systems Symposium (RTSS)*, pages 423–434, Dec 2007.

[53] Giuseppe Lipari and Enrico Bini. A framework for hierarchical scheduling on multiprocessors: From application requirements to run-time allocation. In 31^{st} *IEEE Real-Time Systems Symposium (RTSS)*, pages 249–258, Washington, DC, USA, 2010. IEEE Computer Society.

[54] U.C. Devi, H. Leontyev, and J.H. Anderson. Efficient synchronization under global edf scheduling on multiprocessors. In 18^{th} *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 10 pp.–84, 2006.

[55] P. Tsigas and Y. Zhang. Non-blocking data sharing in multiprocessor real-time systems. In 6^{th} *International Conference on Real-Time Computing Systems and Applications (RTCSA)*, pages 247–254, 1999.

[56] G. Lipari, G. Lamastra, and L. Abeni. Task synchronization in reservation-based real-time systems. *IEEE Transactions on Computers*, 53(12):1591–1601, Dec 2004.

[57] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In 19^{th} *IEEE Real-Time Systems Symposium (RTSS)*, pages 4–13, Dec 1998.

[58] H. Takada and K. Sakamura. A novel approach to multiprogrammed multiprocessor synchronization for real-time kernels. In 18^{th} *IEEE Real-Time Systems Symposium (RTSS)*, pages 134–143, Dec 1997.

[59] R. Rajkumar, L. Sha, and J.P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Real-Time Systems Symposium (RTSS)*, pages 259–269, Dec. 1988.

[60] James H. Anderson and Anand Srinivasan. Mixed pfair/erfair scheduling of asynchronous periodic tasks. *Journal of Computer and System Sciences*, 68(1):157 – 204, 2004.

[61] B.B. Brandenburg and J.H. Anderson. Real-time resource-sharing under clustered scheduling: mutex, reader-writer, and k-exclusion locks. In 9^{th} *IEEE/ACM Intl. Conference on Embedded Software (EMSOFT)*, pages 69–78, Oct. 2011.

[62] S. Afshar, N. Moghaddami Khalilzad, F. Nemati, and T. Nolte. Resource sharing among prioritized real-time applications on multiprocessors. In 6^{th} *International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)*, Dec. 2013.