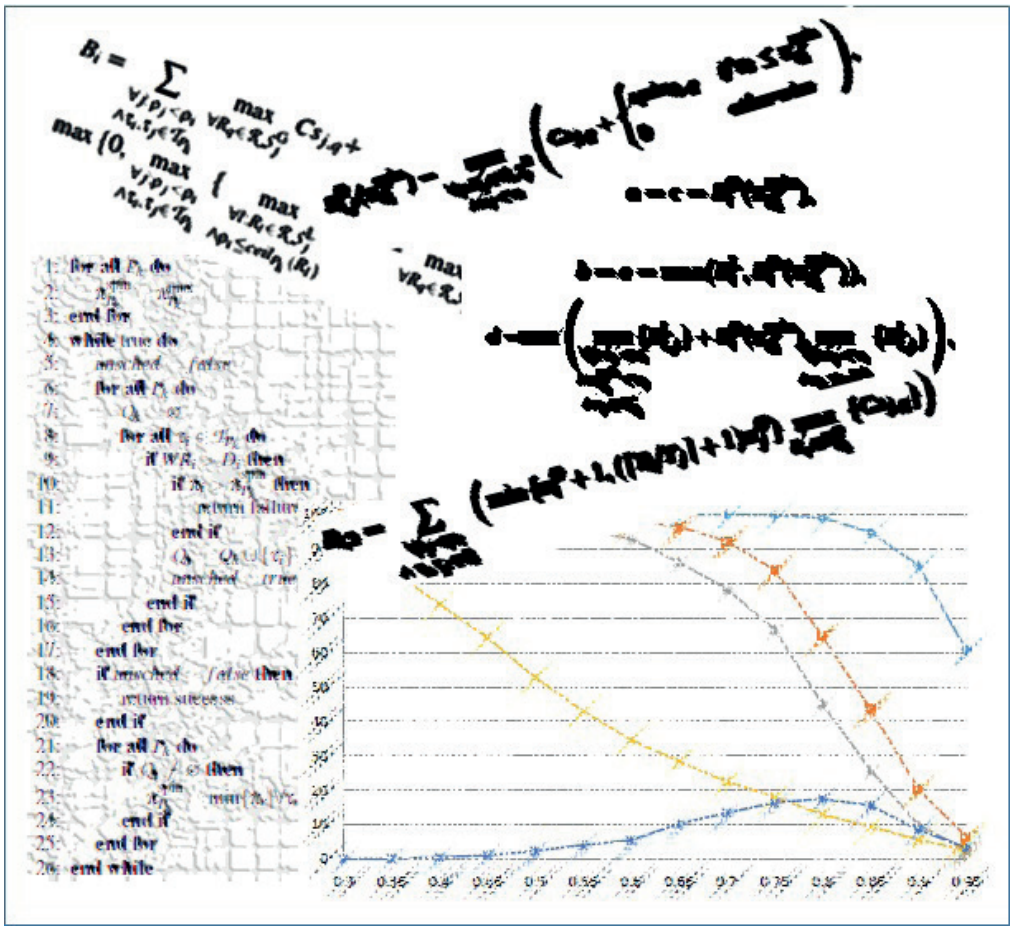
Mälardalen University Doctoral Dissertation 247

Lock-Based Resource Sharing for Real-Time Multi-Processors

Sara Afshar



MÄLARDALEN UNIVERSITY
SWEDEN

Mälardalen University Press Dissertations
No. 247

LOCK-BASED RESOURCE SHARING FOR REAL-TIME MULTI-PROCESSORS

Sara Afshar

2017

School of Innovation, Design and Engineering

ISBN 978-91-7485-361-2
ISSN 1651-4238
Printed by E-Print AB, Stockholm, Sweden

Mälardalen University Press Dissertations
No. 247

LOCK-BASED RESOURCE SHARING FOR REAL-TIME MULTI-PROCESSORS

Sara Afshar

Akademisk avhandling

som för avläggande av teknologie doktorsexamen i datavetenskap vid
Akademin för innovation, design och teknik kommer att offentligen försvaras
tisdagen den 19 december 2017, 13.30 i Kappa, Mälardalens högskola, Västerås.

Fakultetsopponent: Associate professor Enrico Bini, Università degli Studi di Torino

Akademin för innovation, design och teknik

Abstract

The processor is the brain of a computer system. Usually, one or more programs run on a processor where each program is typically responsible for performing a particular task or function of the system. The performance of all the tasks together results in the system functionality, such as the anti-lock brake function of a car. In many computer systems, it is not only enough that all tasks deliver correct output, but it is also crucial that these activities are delivered in a proper time. This type of systems that have timing requirements are known as real-time systems. A scheduler is responsible for scheduling all programs on the processor, i.e., it dictates which program to run and when to run to ensure that all tasks are carried out on time.

Typically, such programs need to use the computer system's hardware and software resources to perform their calculation. Examples of such type of resources that are shared among programs are I/O devices, buffers and memories. When multiple applications require the same shared resource at the same time, the programs may interfere with each other and destroy both their performance and functionality. Fortunately, there are techniques to allow multiple applications to share a resource in a predictable way. One such technique is based on using locks. The program that wants to use a shared resource must first obtain the lock dedicated to the resource before it is allowed to use the resource. If the lock is not already held by another program, i.e., lock is free, so the program can take the lock and use the shared resource. Once the application process is completed with the shared resource, it releases the lock. Locking of shared resources in this manner prevents multiple applications to use the resource simultaneously. Such technology that is used for the management of shared resources is known as resource sharing protocol.

Recently, in order to enhance the performance of computers, more than one processor is used in computer systems. This type of multiple processor systems on a shared hardware platform are called multiprocessors. The existing resource sharing protocols for multiprocessors are still not mature enough and can be further improved in terms of timing requirements. In this thesis we have proposed new resource sharing protocols for multiprocessor systems that could significantly improve upon performance of such protocols. Traditionally, there are two methods for scheduling programs running on multiprocessor systems for each of which there are corresponding resource sharing protocols. Recently, a third category of the schedulers for multiprocessors has been developed which uses a hybrid method combining the two existing scheduling method. This new category is more resource-efficient compared to the two previous methods. Due to the complexity of this new type of scheduling method, it is not straightforward to use the conventional resource sharing protocols for the system that use this type of scheduling. In this thesis, we also have developed proper resource sharing protocols for such hybrid scheduling methods in multi-processor systems.

ISBN 978-91-7485-361-2
ISSN 1651-4238

Mälardalen University Doctoral Thesis
No.247

Lock-Based Resource Sharing for Real-Time Multi-Processors

Sara Afshar

November 2017

School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden

ISSN 1651-4238
ISBN 978-91-7485-361-2
Printed by Mälardalen University, Västerås, Sweden
Distribution: Mälardalen University Press

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 i systemet. 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 hos dem. 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 hårdvaruplattform kallas för multiprocessorer. Befintliga resursdelningsprotokoll för multiprocessorer erbjuder inte bra prestanda, speciellt med avseende på att tillhandahålla ett effektivt resursutnyttjande utav multiprocessorn. I denna avhandling har vi föreslagit nya resursdelningsprotokoll för multiprocessorsystem med tidskrav. Dessa protokoll förbättrar prestanda avsevärt gentemot vad som tidigare var möjligt. De föreslagna protokollen tillhandahåller en attraktiv lösning för att bygga framtidens realtidssystem konstruerade med hjälp av multiprocessorer.

Abstract

Embedded systems are widely used in the industry and are typically resource constrained, i.e., resources such as processors, I/O devices, shared buffers or shared memory might be limited in the system. Hence, techniques that can enable an efficient usage of processor bandwidths in such systems are of great importance. Locked-based resource sharing protocols are proposed as a solution to overcome resource limitation by allowing the available resources in the system to be safely shared. In recent years, due to a dramatic enhancement in the functionality of systems, a shift from single-core processors to multi-core processors has become inevitable from an industrial perspective to tackle the raised challenges due to increased system complexity. However, the resource sharing protocols are not fully mature for multi-core processors. The two classical multi-core processor resource sharing protocols, *spin-based* and *suspension-based* protocols, although providing mutually exclusive access to resources, can introduce long blocking delays to tasks, which may be unacceptable for many industrial applications. In this thesis we enhance the performance of resource sharing protocols for partitioned scheduling, which is the de-facto scheduling standard for industrial real-time multi-core processor systems such as in AUTOSAR, in terms of timing and memory requirements.

A new scheduling approach uses a resource efficient hybrid scheduler combining both partitioned and global scheduling where the partitioned scheduling is used to schedule the majority of tasks in the system. In such a scheduling approach, applications with critical task sets use partitioned scheduling to achieve a higher level of predictability. Then the unused bandwidth on each core remaining once the partitioning is performed, is used to schedule less critical task sets using global scheduling to achieve higher system utilization. These scheduling schema however lacks a proper resource sharing protocol since the existing protocols designed for partitioned and global scheduling cannot be directly applied due to the complex hybrid structure of a hybrid scheduler. In

this thesis we propose a resource sharing solution for such a complex structure. Further, we provide the blocking bounds incurred to tasks under the proposed protocols. Moreover, we enhance the schedulability analysis, which is an essential requirement for real-time systems, with the provided blocking bounds.

To my beloved,
Mohammad and Liana

Acknowledgments

Foremost, I would like to express my very profound gratitude to my supervisors Prof. Thomas Nolte, Prof. Moris Behnam and Prof. Reinder J. Bril. 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. I learned from him to look at research problems not as problems but as challenges, instead. A wise advice that I will take with me not only in my future career but also in life. Also, I am grateful for the valuable feedback and support of Moris which have helped me to improve my work. He has had a great positive impact in my work. His office door has always been open for discussions. Moreover, it was a great pleasure working with Reinder. I enjoyed every moment of our discussions either in our meetings at MDH or during my visit to Eindhoven University or our weekly Skype meetings. This thesis would not be possible without your endless support and help!

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 would like to thank Paolo Gai for his kind feedback during our collaborations. I also would like to take the opportunity to thank Maikel Verwielen and S.M.N. Balasubramanian for the great work they delivered buy their master theses that complimented my research which also evolved to scientific conference papers. I wish to thank Meng, Matthias, Nima and Mohammad for all the nice work discussions we had and their generous help whenever I needed.

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, movies, game gatherings and badminton. I also would like to appreciate IDT

administration staff for their help with practical issues.

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. I am grateful for the entire amazing journey we shared together, which got more colorful with our little angle, Liana!

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

Sara Afshar
November, 2017
Västerås, Sweden

List of publications

Papers included in the thesis[1]

Paper A *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), June, 2014.

Paper B *Per Processor Spin-Lock Protocols for Multiprocessor Real-Time Systems*. Sara Afshar, Moris Behnam, Reinder J. Bril, Thomas Nolte. Accepted by Leibniz Transactions on Embedded Systems, February, 2017.

Paper C *An Optimal Spin-Lock Priority Assignment Algorithm for Real-Time Multi-core Systems*. Sara Afshar, Moris Behnam, Reinder J. Bril, Thomas Nolte. In Proceedings of the 23^{rd} IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), June, 2017.

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

Paper E *Resource Sharing Under Global Scheduling with Partial Processor Bandwidth*. Sara Afshar, Moris Behnam, Reinder J. Bril, Thomas Nolte. In Proceedings of the 10^{th} IEEE International Symposium on Industrial Embedded Systems (SIES), June, 2015.

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

Paper F *Resource Sharing in a Hybrid Partitioned/Global Scheduling Framework for Multiprocessors*. Sara Afshar, Moris Behnam, Reinder J. Bril, Thomas Nolte. In Proceedings of the 20^{th} IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Septembers, 2015.

Additional papers, not included in the thesis

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

2. *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, June, 2012.

3. *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), December, 2012, ACM SIGBED.

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

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

6. *Intra-Component Resource Sharing on a Virtual Multiprocessor Platform*. Sara Afshar, Nima Moghaddami Khalilzad, Moris Behnam, Reinder J. Bril, Thomas Nolte, In Proceedings of the 8^{th} International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS), December, 2015, ACM SIGBED.

7. *Semi-Partitioning under a Blocking-Aware Task Allocation*. Sara Afshar, Moris Behnam, Thomas Nolte, In Proceedings of the 36^{th} IEEE

Real-Time Systems Symposium (RTSS), Work-in-Progress (WiP) session, December, 2015.

8. *An Implementation of the Flexible Spin-Lock Model in ERIKA Enterprise on a Multi-Core Platform*. Sara Afshar, Maikel P.W. Verwielen, Paolo Gai , Moris Behnam, Reinder J. Bril, In Proceedings of the 12^{th} annual workshop on Operating Systems Platforms for Embedded Real-Time applications (OSPERT), July, 2016.

9. *Optimal Priority and Threshold Assignment for Fixed-Priority Preemption Threshold Scheduling*. Leo Hatvani, Sara Afshar, Reinder J. Bril, In Proceedings of the 6^{th} Embedded Operating Systems Workshop (EWiLi), October, 2016.

10. *Optimal Priority and Threshold Assignment for Fixed-Priority Preemption Threshold Scheduling*. Leo Hatvani, Sara Afshar, Reinder J. Bril, In Proceedings of the Special Issue on 6^{th} Embedded Operating Systems Workshop, 2017, ACM SIGBED.

11. *Agent-Centred Approach for Assuring Ethics in Dependable Service Systems*. Irfan Sljivo, Elena Lisova, Sara Afshar, In Proceedings of the 13^{th} IEEE World Congress on Services, June, 2017.

12. *A Dual Shared Stack for FSLM in Erika Enterprise*. S.M.N Balasubramanian, Sara Afshar, Paolo Gai, Moris Behnam, Reinder J. Bril, In Proceedings of the 23^{rd} IEEE International Conference on Embedded and Real-Time Computing Systems and Applications - WiP Session (RTCSA), August, 2017.

13. *Incorporating Implementation Overheads in the Analysis for the Flexible Spin-Lock Model*. S.M.N Balasubramanian, Sara Afshar, Paolo Gai, Moris Behnam, Reinder J. Bril, In Proceedings of the 43^{rd} Annual Conference of the IEEE Industrial Electronics Society (IECON), October, 2017.

Contents

I

Thesis

Chapter 1

Introduction

In recent years, due to a dramatic increase in the functionality of embedded systems, the single-core processors are not anymore the best candidates to handle the amount of complexity in such systems. With the advent of multi-core architectures, multi-core processors emerged to be better alternatives to tackle the issues raised by increased system complexity such as the computation capacity, and thermal specifications due to increase in power consumption. Therefor, a shift from single-core processors to multi-core processors has become inevitable from an industrial perspective.

With a growing interest towards replacing traditional single-core processors with new multi-core processors[1] as the de facto processors 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 available resources. Resource sharing is a technique that can overcome such a constraint. 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 as a consequence decrease or invalidate the system functionality. Lock-based resource sharing protocols provide mutual exclusive access to shared resources other than processors. However, providing exclusive access to shared resource may cause extra delays to tasks. These delays can endanger temporal correctness of a system with timing requirements, known as real-time systems, since

[1] In the rest of the text for brevity, with a slight misuse of notation, we will use multiprocessor for multi-core processors. Similarly, we use core and processor interchangeably in the included papers.

they can lead to uncontrolled *priority inversions* [1]. Priority inversions occur when a high priority task is delayed due to a low priority task for an unbounded amount of time. Therefore, it is essential for resource sharing protocols to bound the delays incurred to tasks due to resource sharing.

From an industrial point of view, partitioned fixed-priority preemptive scheduling is attractive for several reasons such as: higher degree of intuitive predictability, trivial implementations and low run-time overheads, support of commercial real-time operating systems such as VxWorks, QNX, LynxOS, ThreadX [2] and availability in industrial real-time operating systems and standards such as Erika Enterprise [3], and POSIX [4] and AUTOSAR [5]. There is another scheduling approach which uses a hybrid scheduling approach that features a structure similar to partitioned scheduling [6, 7, 8, 9, 10, 11] . In this thesis we have focused on systems with these two scheduling approaches.

Under the hybrid scheduling approach, most tasks are partitioned in the system and a low number of tasks can migrate among cores. One such hybrid scheduling framework has been proposed by Zhu et al. [6] called the *Synchronized Deferrable Server* (SDS) framework. Under this hybrid scheduling framework a set of tasks is partitioned on the platform and a deferrable server is dedicated to each core that has unused bandwidth to run extra workload using a global scheduler. The advantage of such a hybrid scheduling framework can be exploited by applications with a diverse set of tasks where each set may benefit from either partitioned or global scheduling. An example of such diversity in tasks can be run-time monitoring applications [12], where the monitor tasks can migrate across cores to collect events generated by target tasks. In such applications, some task sets including critical tasks may require to be statically bound to particular cores due to various reasons such as requiring a high level of predictability. Another hybrid scheduling framework with similar properties is the semi-partitioned scheduling approach [7, 8, 9, 10, 11] that combines partitioned and global scheduling to improve system utilization compared to pure partitioned scheduling and achieve lower migration costs compared to pure global scheduling [6]. However, no resource sharing has been considered for hybrid scheduling frameworks. In practice, tasks may share resources other than the CPU, such as I/O devices, shared buffers or shared memory. 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.

Due to the complex structure in hybrid scheduling frameworks, the existing resource sharing protocols that have been proposed for partitioned and global scheduling cannot be used without necessary modifications and further

adjustments are needed. Therefore, in this thesis we provide an extension for such frameworks to support resource sharing. Further, we provide the blocking bounds under the presented resource sharing protocols and incorporate them to the schedulability test to guarantee timeliness for systems with timing properties.

There can be two types of resources in a system, called *local* and *global* resources. Local resources are shared by tasks on the same core and global resources are shared by tasks on different cores. Uniprocessor resource sharing protocols can be used to handle resource access to local resources. For sharing of global resources, two classical lock-based protocols exist for multiprocessors: (non-preemptive) *spin-based* and *suspension-based* protocols. The Multiprocessor Stack Resource Policy (MSRP) [13] is a non-preemptive spin-based protocol and the multiprocessor Priority Ceiling Protocol (MPCP) [14, 15] is a suspension-based protocol. The main difference between the two protocols is that the blocked task executes non-preemptively under the classical spin-based protocol, i.e., spins non-preemptively until it accesses the resource, and it is suspended under the suspension-based protocol, i.e., the task releases the core and awaits in a separate queue for its turn to access the resource. There is another variant of the spin-based protocol called the *preemptive spin-based/lock* protocol [16, 17, 18, 19] that has been used as a solution for unordered spin-based locking in AUTOSAR [5]. A corresponding analysis has been presented [19] based on a dequeuing policy where, when a spinning task gets preempted, a preempted spinning task is removed from the resource queue and requeued when it resumes spinning. Under this type of protocol a busy waiting task can get preempted by higher priority tasks on the same core. All three protocols differ in the set of tasks that the protocol allows to access the core on which a task is remotely blocked during the blocking time. In other words, when a task gets blocked on a core by a task on a remote core, under the non-preemptive spin-based protocol the blocked task occupies the core and does not allow execution of any other task until it releases its requested resource. Under the suspension-based protocol however, the blocked task releases the core thus allows the execution of any other ready task while under the preemptive spin-based protocol, the blocked task spins such that only local higher priority tasks are allowed to preempt it. Under all three protocols the access to the resource is non-preemptive from the point of view of normal execution of any task, which is a typical technique for multiprocessors to hasten the release of the resource. Each of these protocols have their own drawbacks and neither of them dominate the other. For instance, under the non-preemptive spin-based protocol, a high priority task is delayed by the busy waiting time of a lower pri-

ority task. Under the suspension-based protocol the blocked task is delayed by the resource access of the tasks that have acquired a global resource while the blocked task was waiting, later when the task is resumed. Although all these resource sharing protocols provide mutually exclusive access to resources, they can degrade the schedulability performance of the system. Moreover, they may introduce long blocking delays to certain tasks that can give rise to variations in the response time of the tasks, which may be unacceptable for many industrial applications. In this thesis we look at other alternatives than the classical spin-based and suspension-based protocols, and the preemptive spin-based protocols that can minimize long delays imposed by resource sharing to tasks. In this thesis we aim at enhancing the performance of these protocols in terms of timing properties compared to the existing alternatives. With the aim at improving the resource sharing protocols to decrease response-time fluctuations for certain tasks and increase schedulability, we have also looked at protocols that provide low memory requirements.

1.1 Research Goal and Research Questions

We formulate the goal of the thesis as follows:

To provide resource sharing protocols for multiprocessor systems under real-time partitioned and hybrid scheduling schemes with the aim to improve performance of such systems in terms of timing properties.

Based on the above research goal we define the research questions. As the aim is to improve resource sharing protocols, we are interested in increasing the performance of such protocols in terms of timing properties, i.e., decreasing blocking delays imposed to tasks. As mentioned in the previous section, there are three lock-based resource sharing protocols being (i) non-preemptive spin-based, (ii) suspension-based and (iii) preemptive spin-based which all use different approaches when a task gets blocked by a remote core in a multiprocessor system. Based on this knowledge, we formulate our first research question as follows:

Research Question 1 (RQ1): Can an intermediate approach provide a solution for satisfying timing requirements of a set of task sets for which none of the existing non-/preemptive spin-based and suspension-based protocols can do so, and if yes, how to model such a solution?

In this thesis we have considered two properties of multiprocessor systems being (i) timing requirements of the tasks sets which refer to various timing aspects of a real-time task set such as schedulability ratio and worst-case response times, and (ii) memory requirements. In order to investigate the improvement of the proposed protocols under both aspects we formulate the second research question as follows:

Research Question 2 (RQ2): How can timing and memory requirements of task sets executing in multiprocessor systems be optimized?

Considering the fact that hybrid scheduling frameworks lack a proper resource sharing solution, we formulate our last question as follows:

Research Question 3 (RQ3): How to support resource sharing in hybrid scheduled multiprocessor systems?

1.2 Technical Contributions

1.2.1 Main Contributions

We have four main contributions that are included in this thesis which address the research questions provided in the previous section. The research contributions are presented in 6 scientific papers itemized from A to F in this proposal.

Research Contribution 1 (RC1): *Proposing a new model, called Flexible Spin-Lock Model (FSLM), to model the blocking of a task.*
The main difference of the non-/preemptive spin-based and suspension-based protocols, as also discussed in Section 1.1, is the behavior of a task upon blocking. Concentrating on the behavior of a blocked task, we realized that a solution that allows some tasks to execute while a task is blocked, and disallows others, may outperform the existing solutions of allowing *no*, *all* or only *higher priority* ready task(s) to execute during blocking time. Based on this observation we have identified that the traditional non-preemptive spin-based and suspension-based protocols can conceptually be unified by viewing a suspension-based protocol as the spin-based protocol where a blocked task spins but uses the lowest priority level on a core, i.e., a priority lower than any "original" priority of tasks on that core. We refer to a suspension-based protocol as *LP* (lowest priority) and to the non-preemptive spin-based protocol as *HP* (highest priority)

to refer to the priority level they use while spinning. In a similar way, we could view the preemptive spin-based protocol such that a blocked task spins using its original priority during spinning. We refer to this protocol as *OP* (original priority). Based on such a view, we have generalized a task's blocking behavior while spinning by being able to select any arbitrary priority level in the range of *LP* to *HP*. We refer to this model as *flexible spin-lock model* (FSLM). We use *spin-lock priority* to refer to the priority at which a task is spinning while waiting for a global resource to become available. This model allows us to use intermediate spin-lock priorities for spinning, an approach having the potential of outperforming the existing *HP*, *LP* and *OP* protocols. Therefore utilizing such a general model for spinning could potentially improve upon existing protocols. We have identified that tasks of a system can use spin-lock priorities from FSLM in five different ways where the use of each type can affect system schedulability differently. Tasks can use a fixed spin-lock priority: (*i*) per-core: all tasks on a core use a single spin-lock priority for all their requests, (*ii*) per-task: each task on a core can use a different spin-lock priority for all its requests, (*iii*) per-resource: each task on a core can use a different spin-lock priority for all requests regarding the same resource type, (*iv*) per-request: each task on a core can use a different spin-lock priority for each of its resource requests, and (*v*) a combination of any of the above types. This contribution answers the research question RQ1. This contribution is presented in papers A and B.

Research Contribution 2 (RC2): *Proposing new resource sharing protocols from FSLM that improve the performance of real-time systems compared to the traditional protocols in terms of timing and memory requirements.*

We have presented three new resource sharing protocols based on three different spin-lock priorities selected from FSLM, two of which are from the per-core type and one is from the per-task type. We showed that these protocols can improve upon the existing protocols in terms of timing requirements. Further, we showed that a specific range of spin-lock priorities from per-core type have low memory requirements. Finally, assuming the spin-lock priorities per-core type, we have proposed an optimal algorithm to assign spin-lock priority to each core, called *optimal spin-lock priority* (OSPA). We have shown that OSPA can significantly improve over *HP* and *LP*.

This contribution answers the research question RQ2. This contribution is presented in papers A, B and C.

Research Contribution 3 (RC3): *Providing a new general blocking analysis for the new resource locking model FSLM.*

In order to explore the spin-lock classes we need the blocking analysis of the FSLM model. Therefore, we have provided a general blocking analysis for the per-core spin-lock priorities where it gets the spin-lock per-core as an input parameter and it provides the blocking bounds incurred to tasks. This contribution is a necessary step to answer the research question RQ2. This contribution is presented in papers A, B and C.

Research Contribution 4 (RC4): *Providing resource sharing protocols for hybrid scheduling and providing the corresponding blocking bounds imposed to tasks under such scheduling, and incorporating the bounds in the schedulability analysis.*

In hybrid scheduling where both partitioned and global scheduling is used, bandwidth of a core is affected by both fixed assigned and migrating tasks. None of the existing resource sharing protocols could be used directly for resource handling in such a complex system structure. A proper resource sharing protocol is required to handle resource sharing among tasks in such setups. Moreover, a corresponding schedulability analysis must be developed to the new resource sharing protocol. Targeting two instantiations of such schedulers, we have provided a resource sharing solution for each. The main challenge is to serve the resource requests of tasks that are assigned to different cores in the presence of a partitioned set of tasks.

This contribution answers the research question RQ3. This contribution is presented in papers D, E and F.

The relation between the research questions and research contributions is depicted in Table 1.1.

	RQ1	RQ2	RQ3
RC1	√		
RC2		√	
RC3		√	
RC4			√

Table 1.1: The relation between RQs and RCs.

1.2.2 Additional Contributions

Besides the main contributions that are included in this thesis we have other additional contributions which are not included in this thesis. In the following we explain these contributions due to their close relation to part of the contributions of this thesis. From a theoretical perspective, pre-emptive spin-based protocols, such as those described by FSLM, outperform non-pre-emptive protocols, such as MSRP [13] in general. However, in practice the runtime overheads such as context-switch and preemption related overheads exist which can vary depending on the hardware platform that is used, and they affect the schedulability. With the aim to make FSLM suitable for the automotive domain, and to cope with practical challenges of the model, the model has been validated through implementation. The runtime overheads introduced by FSLM have been identified and incorporated in the schedulability analysis. The line of research regarding enhancing the theoretical model of FSLM to account for run-time overheads is not the focus and contribution of this thesis. However, due to a strong correlation to a part of the contribution of this thesis, we dedicate a separate section here to report the results regarding these activities.

To implement FSLM, the OSEK/VDX-compliant Erika Enterprise Real-Time Operating System (RTOS) has been used. Erika Enterprise [3] is a free of charge, open-source RTOS implementation. It was originally developed for small-scale OSEK/VDX compliant embedded systems for the automotive industry [20]. A ported version of it to the Altera Nios II platform [21] exists which supports multiple soft-cores [22]. An initial implementation of FSLM has been done in Erika Enterprise on an Altera DE0 board [23, 24] using 4 soft-core processors. The implementation supports a specific range of spin-lock priorities from FSLM that confines the length of global resource queues to the number of cores in the system. The implementation overheads were identified and measured. However, due to the limitations of the chosen hardware, the measurements resulted in large implementation overheads. In order to achieve more realistic implementation overheads the initial implementation was ported to a higher performance hardware platform, an Altera-DE2-115 board, another Nios II-based hardware platform, [25, 26] which provides sufficient hardware resources for measurement and analysis. The implementation of FSLM was further optimized to reduce overheads with respect to the inter-processor communication delays.

Under MSRP a blocked task spins non-preemptively. Therefore, in the original implementation of MSRP a global low-level spin-lock variable is continuously polled by the blocked task to check for the release of the resource.

Under FSLM however, since a spinning task might be preempted by higher priority tasks, polling on a global spin-lock variable is no longer possible, resulting in unawareness of the preempted block task of the release of the resource. Therefore, the preempted task is required to be notified by the task which releases the resource on a remote core. Therefore, in order to implement FSLM in Erika Enterprise a dedicated inter-core communication mechanism was required, which introduced additional overheads. The initial implementation of FSLM [23, 24] used shared data structures and an inter-core interrupt mechanism available in Erika Enterprise RTOS, known as remote notification mechanism [22], which turned out to lead to significant overheads. The later implementation of FSLM [25, 26] reduced those overheads associated with inter-core communication by replacing the use of shared memory with a Dedicated Interrupt (DI) mechanism. This improvement was feasible at the expense of limiting the implementation to the same specific range of spin-lock priorities considered in [23, 24]. Under the considered range there is at most one task on any core that has a pending global resource request. Therefore, it is sufficient to send an inter-core interrupt to notify the release of the resource rather than using remote notification-related shared data structures. The results showed that the overhead was roughly reduced to half. Seven overhead components were identified regarding the request, access and release of a global resource which were incorporated in the worst-case response time analysis [25, 26]. An implementation of FSLM that validates the sufficiency of using only two stacks for the specific above-mentioned range of spin-lock priorities proven in Paper B is also done in [27].

1.2.3 Role of the Contributors

I am the main deriver and the first author of all the included papers in this thesis. Other co-authors are my supervisors who contributed in improving my work by providing feedback and comments.

1.3 Research Method

The research methodology used in this thesis work is conformant [28] with the steps proposed in [29], which is a deductive research method. According to a deductive research method, a goal or hypothesis is developed. Then, a

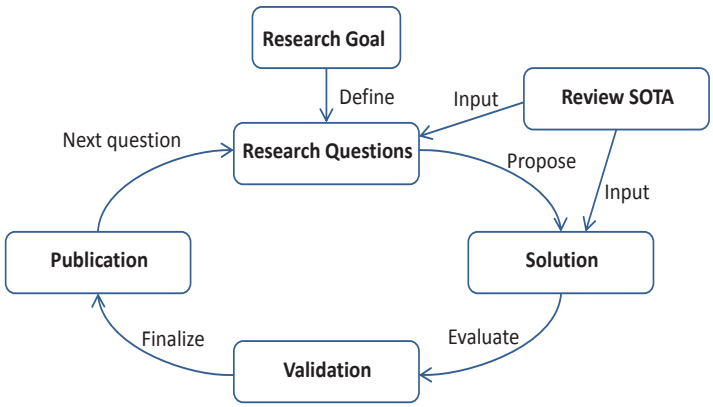


Figure 1.1: Research methodology.

strategy to achieve the goal is followed. The flow of the research in this work is illustrated in Figure 1.1.

First we defined the main goal by reviewing the state of the art in the context of resource sharing protocols for multiprocessors. Then, we identified the research questions after analyzing the possible challenges to achieve the stated goal. We explored the possible solutions to answer the research questions. In order to propose a solution we always looked at the state of the art as an input. Then, we validate each and every solution. There are several ways to validate a solution for example, using experiments, simulations, formal modeling and analytical proofs with mathematical models. We used analytical proofs and experiments in the form of schedulability and worst-case response time improvements tests to validate the proposed solutions. In every validation phases we compared our solutions with state of the art solutions, if any, to show the performance of the proposed solutions. Whenever we obtained a desirable and validated solution we finalized it by publishing scientific reports or papers. We performed the same methodology to answer all research questions.

1.4 System Model

In this thesis we may use different notations to introduce our system model in some of the papers. Therefore, we present the exact notations used in each

paper to explain the used model to be explained by the system model section in each paper. However, in this section we present the general notations and assumptions that has been used in all papers.

In this thesis we have assumed multi-core systems that are constituted of m identical unit-capacity cores on which a task set of n sporadic tasks will execute. A task τ_i is an infinite sequence of jobs for which its worst-case execution time is denoted by C_i and its minimum inter-arrival time is denoted by T_i. A task is said to have arrived/be released when it is placed in the ready queue. From the time when the task is released it should finish its execution before its defined deadline denoted by D_i. We have assumed two types of task sets: implicit deadlines task sets, i.e. for every task τ_i, $D_i = T_i$, and constrained-deadline task sets where for every task τ_i, $D_i \leq T_i$. Each task is attributed with a fixed priority in the system. Tasks may use *local* or *global* resources where local resources are accessed by tasks on the same core and global resources are accessed by tasks on different cores. Local and global critical sections (*lcs* and *gcs*es) are the sections of a job of a task that use local and global resources, respectively. We denote $Cs_{i,q}$ as the worst-case execution time among all requests of any job of a task τ_i for resource R_q. Nested access to resources has not been considered in this thesis.

1.5 Outline of the Thesis

This thesis consists of 9 chapters and the rest of the thesis is organized as follows. Chapter 2 presents the most relevant background and prior work. Chapter 3 summarizes the content of the thesis and identifies directions for future work. The included papers are presented in Chapters 4 to 9.

Chapter 2

Background and Prior Work

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 [30]. In other words, the correct results should be delivered within a certain time called 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 executes 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 after the arrival time of a job, when the job should finish its 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 core 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 captured by *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 core bandwidth that is required by 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 core, the total utilization of the task set should not exceed one and accordingly m on an m-core platform.

2.2 Multiprocessor Platforms

With the emerge of multiprocessors, they have started to be used widely in embedded systems [31]. Multiprocessor platforms have found their way into real-time systems due to their wide availability in the market 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 cores have access to a shared memory by means of the shared bus. The maximum access time for a core 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 core they execute on. In contrast, in *heterogeneous* platforms each core may have a different speed. Therefore, task execution requirements are proportionally scaled up or down with the core speed they are running on, by being assigned to slower or faster cores.

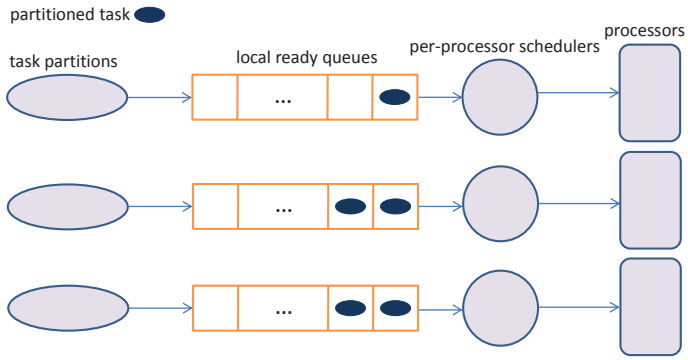


Figure 2.1: Partitioned scheduling.

2.3 Multiprocessor Real-Time Scheduling

Two fundamental scheduling approaches exist for multiprocessor platforms: partitioned and global scheduling [32, 33, 34, 35]. Resource reservation techniques have introduced a third type that is called hybrid scheduling which combines partitioned and global scheduling on the same platform.

2.3.1 Partitioned Scheduling

Under a partitioned scheduling approach, tasks are assigned to fixed cores during design-time and all jobs of each task execute on the same core to which the task is assigned, during run-time. Each core uses a uniprocessor scheduling approach such as *Rate-Monotonic* (RM) or *Earliest Deadline-First* (EDF) [36]. Each core uses a separate scheduler and a local ready queue to independently schedule the tasks on the core as can be seen in Figure 2.1. Schedulers on different cores on the multiprocessor platform may use identical or different scheduling algorithms. Some 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 [37]. In other words, finding an optimal solution to allocate tasks to cores cannot be done in a polynomial time. Therefore, heuristic algorithms are used to partition tasks among cores. However, once the partitioning/mapping (i.e., assignment of tasks to cores) is done, the well

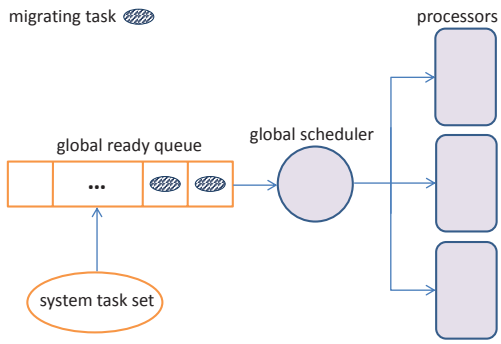


Figure 2.2: Global scheduling.

known uniprocessor scheduling approaches can be used to schedule tasks on cores which is another advantage of partitioned scheduling. One disadvantage of partitioned scheduling is that cores may not be fully utilized. For some task sets when the utilization reaches slightly higher than $\frac{m+1}{2}$, partitioning over m cores is not possible [38, 39, 40], and for some task sets, if the total utilization reaches slightly higher than 50%, deadlines might be missed [41]. 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 [4]. One example is implemented by the AUTOSAR [5] standard.

2.3.2 Global Scheduling

Under a global scheduling approach, one global scheduler schedules all tasks to the cores 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 cores. 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-core platform. Global scheduling can offer advantages compared to partitioned scheduling [42, 6, 43] and neither partitioned nor global scheduling is completely preferable to the other [44]. For instance, in adaptive systems where tasks requirements change during runtime in response to environmental changes and in open systems where tasks may be added to or

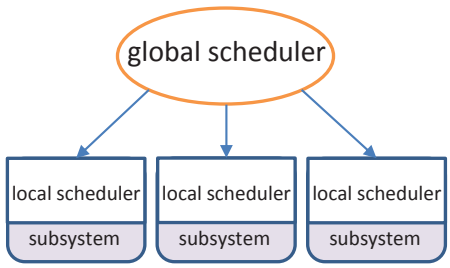


Figure 2.3: Hierarchical scheduling.

removed from the system dynamically, global scheduling is a more suitable approach since it assigns tasks to cores dynamically and thus does not require to deal with the complex task mapping problem for such systems [43, 45]. Moreover, global scheduling is exposed to less context switches/preemptions compared to partitioned scheduling, since it only preempt a task when there are no idle processors. However, migration overhead might be very expensive under global scheduling. Uniprocessor scheduling protocols such as RM and EDF [46] are not optimal anymore on multiprocessor platforms. An optimal scheduling approach is an approach of scheduling where if there exist any scheduling approach that can make a task set schedule then the optimal scheduling approach will also make the task set schedulable. Many works have provided efficient analysis for global scheduling [47, 48, 49, 50, 51]. New scheduling approaches have been proposed for global scheduling, such as the proportionate fair (*pfair*) scheduling approach [52, 53], 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 [42].

2.4 Hierarchical Scheduling

Hierarchical scheduling is an approach used to schedule tasks in a hierarchical manner. As an example, in a two-level hierarchical scheduling system, on high level a global scheduler schedules subsystems and on lower level, a local scheduler schedules tasks within the subsystem using a local scheduling policy. Figure 2.3 shows a two-level hierarchical system. The main objective of hierarchical scheduling is to provide temporal 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 cores. This prevents the propagation of temporal errors among subsystems. Hierarchical scheduling can be applied to both uniprocessor platforms [54, 55, 56, 57] as well as to multiprocessor systems [58, 59, 60, 6]. Different scheduling policies can be used for scheduling tasks within each subsystem as well as scheduling subsystems on the cores.

2.4.1 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 to benefit from the advantages of both approaches and to minimize the disadvantage of each. Semi-partitioned scheduling is one of such approaches [7]. To utilize cores in a better way, the semi-partitioned approach suggests to further utilize the remaining capacity on each core to schedule the tasks that could not fit on any core. Since any of the remaining tasks could not fit on any core, typically, their execution has to be split among multiple cores. In semi-partitioned scheduling, similar to partitioned scheduling, each core has a separate scheduler and local ready queue to schedule the partitioned tasks on each core. However, the tasks which are split among cores can migrate and be scheduled on different cores as shown in Figure 2.4. So far, various task assignment techniques have been proposed for the semi-partitioned approach [8, 9, 10, 11]. Guan et al. [11] showed that the utilization bound of task sets on each core can be increased as high as the utilization bound of Liu and Layland's RM scheduling for an arbitrary task set.

Another resource efficient approach, that uses a hybrid structure, has been introduced by Zhu et al. [6] called *Synchronized Deferrable Servers* (*SDS*). Similar to the semi-partitioned approach, SDS also uses a combination of global and partitioned approaches. Similarly, SDS partitions tasks on cores and utilizes the remaining capacity from the partitioning to schedule extra tasks. In this approach, the remaining capacity on each core is served by means of a set of *deferrable servers*. A two-level hierarchical scheduling is used, on each core, the partitioned tasks along with the deferrable server(s) on the core are scheduled following a partitioned scheduling and tasks within the servers are

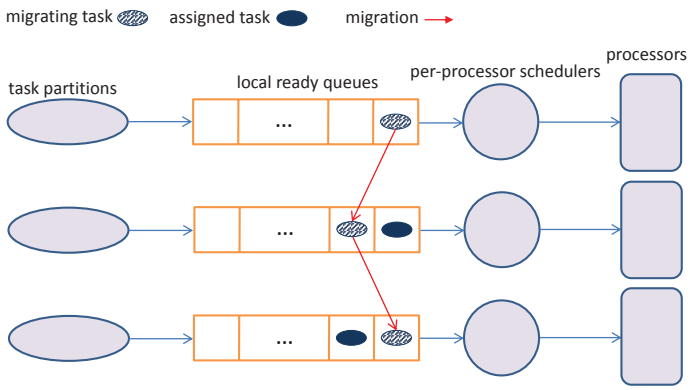


Figure 2.4: Semi-partitioned scheduling.

globally scheduled and may migrate among cores as shown in Figure 2.5. Under this approach, one or more deferrable servers are assigned to each core that provide capacity after partitioning in order to schedule an extra set of tasks, thus improving the system utilization. Zhu et al. have also presented a response time analysis for such a framework. An example of such systems where the SDS approach can be utilized is platforms that exploit a set of tasks in the system for run-time monitoring [12] to detect errors of a set of target tasks. In such a system, monitoring tasks will be as migrating tasks that run within servers and can migrate among cores to monitor at run-time the tasks that are assigned to each core. Under SDS, the tasks that are scheduled within servers may be preempted at any point in their execution and resumed on any other core containing a server. In the case that all cores contain a server which is the case studied in [6], tasks that are globally scheduled may execute on any core. However, as mentioned above, this is not the case for split tasks (tasks which migrate) under the semi-partitioned approach. Although these tasks may also migrate, their migration to another core happens only at predefined execution points. Moreover, these migrating/split tasks can only migrate among the cores that they are split over and not among all cores. The set of cores available for migration of a split task is dependent on the allocation technique that is used to split the task.

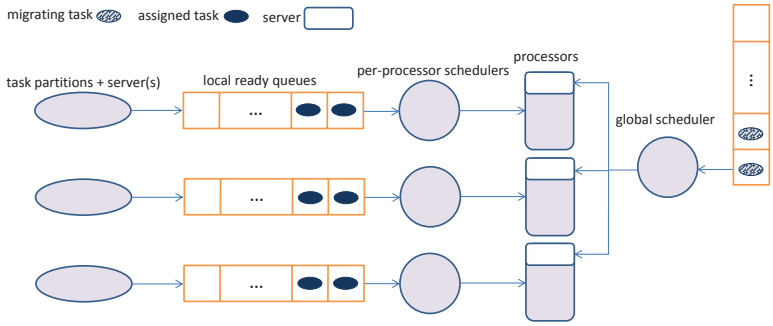


Figure 2.5: SDS framework.

Cluster-based scheduling approaches represent 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 cores and are scheduled globally within a cluster as shown in Figure 2.6. Cluster-based scheduling maps to partitioned scheduling when m clusters exist in the system where m is the number of cores, and cluster-based scheduling is equal to global scheduling when one cluster exists in the system, only. Figure 2.6 shows a 2-cluster system where the number of cores in each cluster is 3. Cluster-based scheduling is classified into two types: *physical* and *virtual*. Under a physical cluster-based approach [58], each cluster is assigned to a fixed set of cores, whereas under a virtual cluster-based approach [59] the clusters are assigned dynamically to the cores.

2.5 Real-Time Locking Protocols

Many scheduling approaches assume that tasks are independent and do not share any resources but the cores. However, this assumption is not always true especially in embedded systems where a set of constrained resources are available. Therefore tasks in such systems may have to share resources such as queues, buffers or I/O devices with each other. Concurrent access to shared resources need to be synchronized in such systems to avoid possible data corruption. One solution for synchronizing access to shared resources to achieve mutual exclusive access is using *locks*. A task that requests a resource requires

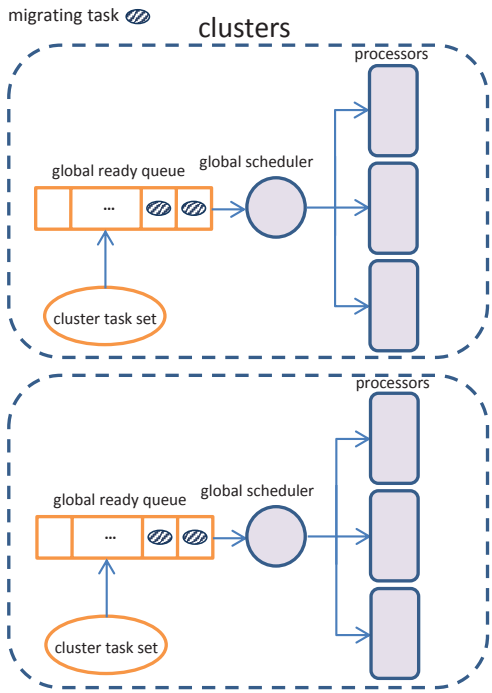


Figure 2.6: Cluster-based scheduling.

to lock it prior to holding/using it. A task that holds a resource will use the resource, i.e., it will execute its *critical section* (cs) where it uses the resource. An alternative approach, however, is the lock-free synchronization protocol. In the lock-free protocol [61, 62], tasks try to access the shared resources, until they succeed. The convenience of using lock-free protocols 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 easily be bounded, this approach may not be the best choice for real-time applications where predictability is essential, specially hard real-time systems. In this thesis, we therefore focus on lock-based synchronization protocols.

In a multiprocessor platform access to local resources are typically handled by local resource sharing protocols such as the *Priority Ceiling Protocol* (*PCP*) [1] or *Stack Resource Policy* (*SRP*) [63]. If the execution of a task that

is ready is delayed due to lower priority tasks on the same core, the task incurs blocking also referred to as *priority-inversion blocking* (pi-blocking). Priority-inversions, which occur when a higher priority job is prevented to be scheduled due to execution of a lower priority job for an unbounded amount of time, can endanger temporal correctness of real-time systems. Hence, blocking should be bounded for real-time systems. 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, i.e., neither abort [64, 65] nor a roll-back mechanism [66] is used) 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. We refer to the pi-blocking that a task experiences due to locking a local resource by a lower priority task as *Local Blocking due to Local Resources* (*LBL*) and due to locking a global resource as *Local Blocking due to Global Resources* (*LBG*).

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 core). This type of delay, which is referred to as acquisition delay, must be accounted for in a task's response time. Since the acquisition delay is due to waiting for a resource, it is also referred to as *remote blocking*. A task that requests a resource, either local or global, may not instantly be able to lock the resource. When a task is waiting to acquire a resource, it is also said that the task is *blocked on the resource*. As soon as the task is granted access to the resource it locks the resource.

Traditionally, two main techniques exist to determine which task is allowed to access a global resource when multiple tasks have pending requests being *ordered* (also called *queued*) or *unordered* (i.e., no particular order is guaranteed). In case of *ordered*, a unique global queue for each global resource is dedicated to enqueue requests of the tasks waiting for that resource as shown in Figure 2.7. 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 allow lower priority tasks that are placed ahead of higher priority tasks in the queue to acquire the re-

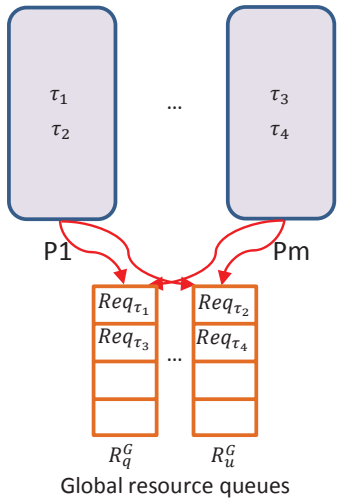


Figure 2.7: Local and global resource queues.

source. 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. Wieder and Brandenburg [19] showed that none of the queueing techniques dominates the other.

When a task is remotely blocked in a multiprocessor platform, the task may either perform (typically) a non-preemptive busy-wait (also called spin) or suspend and release the core. 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 protocols dominates the other, i.e., there might be multiprocessor systems that are schedulable under the spin-based protocol and not under the suspension-based protocol and vice versa.

Under a spin-based protocol, 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 core bandwidth for the whole waiting period. Whereas, under a suspension-based protocol, the task waiting for the resource lets other tasks on the core perform additional work. However, by letting other tasks to execute on the core when the task is waiting for a remotely held resource, more resource requests may be issued by those tasks running on

the core. These requests, in the worst-case, contribute extra delay to the waiting task under a suspension protocol. If all critical sections are short, spin-based protocols are preferred since context switch overheads are smaller compared to when suspension-based protocols are used where tasks are more subject to suspending and resuming overhead. However, if critical sections are long, the core time is wasted by spinning compared to a suspension-based protocol that lets another task to run on the core. Specifying a critical section to be long or short is user-defined [67].

Spinning can be both preemptive and non-preemptive. Under a preemptive spin-based protocol a task that places a request in a global queue and spins might get preempted on its core by local tasks. Three policies are typically considered upon preemption of tasks that are spinning and their request have been placed in a global resource queue being *de-queuing* [17, 18, 16, 19], *skipping* [68] and the *classic* policy upon pre-emption, i.e., a task is neither dequeued nor skipped. Under the de-queuing policy a task that is preempted during spinning is removed from the resource queue and enqueued again when it is allowed to resume spinning. Using this policy a blocked task may have to wait for additional remote tasks with later requests to the same global resource. Under the skipping policy the task remains in the queue, but is not selected for being granted the resource when it is at the head of the queue and the global resource becomes available. Instead the next task in the queue that is eligible is selected. Using this policy a blocked task may have to wait for an additional remote tasks with a later request to the same global resource. Under the classic policy, a task is not dequeued or skipped upon preemption and it is granted access to the global resource when it is at the head of the queue and the resource becomes available.

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 priority higher than the priority of the task itself. In the traditional spin-based protocol, the priority of a task is raised to the highest priority on the core and the task becomes non-preemptive. 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. Another variant of spin-based protocols are preemptive spin-based protocols where the spinning tasks can be preempted by a local task while spinning. In our thesis we also explore a family of preemptive spin-based protocols based on our proposed FSLM model. The existing preemptive spin-based protocols refer to protocols under which tasks can be preempted by local higher priority tasks [16, 17, 18, 68, 19] which is equivalent to our preemptive spin-lock protocol where a spinning task use its original priority during spinning. In our FSLM model, besides the original priority, tasks can use any priority for spinning. Under the preemptive spin-based protocol, access to global resources is still non-preemptive in order to hasten the release of the resource. In the following the most relevant spin-based protocols are presented.

MSRP Synchronization Protocol

The Multiprocessor Stack Resource Policy (MSRP) [13] is a non-preemptive spin-based locking protocol introduced by Gai et al. MSRP is an extension of the SRP [63] 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 global resource (i.e., waiting to acquire the resource) spins non-preemptively and is wasting the core 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. In [69], Gai et al. have compared MSRP with the Multiprocessor Priority Ceiling Protocol (MPCP) [15] 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 core time under MSRP, spin-locking is expensive for e.g., automotive applications, compared to the MPCP protocol. 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. [70], is a non-preemptive spin-based protocol for multiprocessor

platforms. M-BWI is an extension of the Bandwidth Inheritance (BWI) [71] protocol that combines the constant bandwidth server [72] with a priority inheritance protocol [1] 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 core bandwidth for the task's execution. Under the M-BWI protocol, 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 cores 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) [73] proposed by Burns and Wellings, is a preemptive spin-based protocol for multiprocessor partitioned fixed-priority scheduled platforms which is a variant of PCP for a single core. 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 core 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 core and access the core, it finds that its critical section has been executed. This approach is inspired by a method called helping used in [74]. 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 result 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 when it accesses the resource. The authors concluded that for a low priority task, MrsP cannot perform better than a non-preemption protocol similar to MSRP. However, they showed that the same low priority task may have a significant improvement in its response time under another protocol 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 protocol. Later, in [75], the authors provided a tighter analysis for MrsP.

Other Spin-Based Protocols

Scalable spin-based protocols were studied in [76] by Mellor-Crummey and Scott with the aim to minimize the network of transactions that lead to contention. They focused particularly on non-preemptive spin-based protocols with FIFO-based queues. Later Craig and Johnson extended the investigation to priority-ordered queues [17, 77]. In [77] the main focus is on non-preemptive spin-based protocols, where in [17] the preemptive spin-based protocols with FIFO-based queues, using the de-queueing policy upon preemptions, have also been investigated. Other extensions of these works for the preemptive version with FIFO-based queues have also been studied in [18, 16]. Takada and Sakamura investigated preemptive spin-based protocols based on a skipping policy [68]. In [61] spin-based and lock-free resource sharing protocols under global EDF (G-EDF) scheduling have been investigated.

A recent work by Wieder and Brandenburg [19] have investigated both preemptive and non-preemptive spin-based protocols for both ordered and unordered techniques where for the ordered variant both FIFO and priority-based ordering has been considered. A de-queueing policy has been used to avoid the transitive arrival blocking problem which occurs in conjunction with FIFO-based queues and preemptive spin-based protocols. Transitive arrival blocking occurs when a task waiting in a global resource queue is preempted by a local higher priority task that requires the same global resource, thus the higher priority task has to wait for that lower priority task. They used a Mixed-Integer Linear Program (ILP) technique to bound the maximum cumulative blocking imposed to tasks to achieve tighter bounds. In [78] a new blocking analysis based on graph abstraction for nested non-preemptive spin-based protocols based on FIFO-based queues has been introduced. In [79] spin-based multi-

processor real-time locking protocols for replicated resources has been investigated where tasks may use multiple replicas.

2.5.2 Suspension-Based Protocols

In suspension-based locking protocols, when a task is waiting for a resource, it suspends and releases the core. The task is inserted in 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 core 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 [80], which first was introduced by Rajkumar et al. in [14]. The protocol relies on message passing between cores and uses remote procedure calls. In DPCP a job executes its local and normal (i.e., non-critical) sections on its assigned core while its global critical sections may execute on cores other than its allocated core. Cores that execute global critical sections are called synchronization cores. All critical sections of a specific global resource are bound to one core, however, there may exist multiple synchronization cores in the system. Under DPCP, the priority of a task within a global critical section is raised higher than any priority in the system. 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 cores boundaries, which is achieved under DPCP by bounding critical

sections of the same global resource to the same synchronization core. 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. [14, 15] 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 under the DPCP protocol. However, MPCP has a more efficient implementation compared to the DPCP protocol where overhead of remote execution of global critical sections and communication delays needs to be accounted for. A spin-based variant of MPCP has been proposed in [81].

FMLP Synchronization Protocol

The Flexible Multiprocessor Locking Protocol (FMLP) proposed by Block et al. [67] is a synchronization protocol that combines suspension-based and spin-based protocols for different types of resources in the system. FMLP has been developed for both partitioned (partitioned-EDF) and global scheduling (global-EDF and pfair PD^2 [82]). 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 protocol when they are remotely blocked on long resources whereas they use a spin-based protocol when they get blocked on short resources. Nested global critical sections are supported under the FMLP protocol. Under FMLP, the priority 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 non-preemptively within both long and short global critical sections. 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. Global resource queues are FIFO-based under FMLP. Later, Brandenburg and Anderson extended the partitioned FMLP to fixed-priority scheduling [83]. In an evaluation of partitioned FMLP [84], the authors introduced long FMLP and short FMLP in which all global resources are long and short, respectively. Therefore, long FMLP and short FMLP are suspend-based and spin-based synchronization protocols, respectively.

OMLP Synchronization Protocol

The $O(m)$ Locking Protocol (OMLP) proposed by Brandenburg and Anderson [85], is a suspension-based synchronization protocol. OMLP has been denoted as a *suspension-oblivious* protocol [85]. It has been denoted that under a suspension-oblivious protocol, the waiting time of tasks is accounted as an additional execution, i.e., suspended jobs are assumed to occupy the core. 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 [85]. Asymptotically optimal denotes that the blocking times are 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 cores). 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 has to acquire a unique token dedicated for each core. Only one token exist for the resources used in each core. Under the partitioned OMLP, 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 [86], 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 may be preempted frequently while using the priority donation technique, each task may be preempted at most once.

P-PCP Synchronization Protocol

Parallel PCP (P-PCP) has been proposed by Easwaran and Andersson [87] 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 global fixed-priority 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. [88] 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 core hosts one application, i.e., all tasks related to the same application are assigned to the same core and applications do not share cores. Global resource queues are FIFO-based. If a task within an application blocks on a global resource, a placeholder is located for its core in the queue

and the task is inserted in a local waiting queue for that specific resource dedicated to the core. When the related core is at the head of the global FIFO-based queue, the task at the related local resource queue will lock the resource. Both local FIFO-based and priority-based resource queues have been investigated in this work.

When a task on a core requests a global resource, its priority is boosted immediately to its original priority plus the highest priority on that core. 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 [89].

Other Suspension-Based Protocols

A resource sharing protocol for Partitioned Earliest Deadline First (P-EDF) has been proposed in [90] for periodic task sets which later was extended to support sporadic task sets [41, 33]. In [91] lock-free, wait-free and both spin-based and suspension-based protocols from the lock-based synchronization protocols have been studied on the $LITMUS^{RT}$ platform. Under a wait-free protocol, each access is guaranteed to be completed after execution of a number of code instructions by the task accessing the resource. Lock-free and wait-free protocols are only used for shared data object [91]. They have concluded that non-blocking algorithms are desirable for small shared data whereas wait-free or spin-based protocols are desirable for larger and more complex shared data and in general, wait-free protocols are preferable to lock-free protocols and suspension-based protocols are not desirable for partitioned systems. In [92] an extension of SRP for hierarchical multiprocessor scheduling is provided. Under this protocol, tasks that share resources are grouped together as independent components where tasks within each component use SRP to synchronize resource sharing and components are scheduled globally among cores.

In [93] a suspension-based resource sharing protocol for global fixed-priority scheduling is proposed which limits the number of blockings to higher priority tasks. The existing worst-case response time analysis [94] is extended to incorporate the provided blocking bounds. In [86] resource sharing for cluster-based scheduling under a suspension-based protocol has been presented. Under this scheduling approach tasks are bounded to clusters of cores and are scheduled globally within each cluster. A blocking-aware partitioning algorithm called the *Synchronization-aware Partitioning Algorithm* (SPA) has been proposed by Lakshmanan et al. [95] under both spin-based and suspension-based variants of

MPCP [14, 15]. Nemati et al. have proposed another blocking-aware partitioning algorithm called the *Blocking-aware Partitioning Algorithm* (BPA) [96] under the suspension-based MPCP which showed higher performance compared to SPA. Another partitioning heuristic proposed by Wieder and Brandenburg [97] called Greedy Slacker has shown better results compared to both SPA and BPA.

In [98] a new partitioning approach for multiprocessor systems that share resources under fixed-priority scheduling has been proposed where besides the tasks the resources in the system are also bound to cores under the partitioning policy. Under this approach which is called *resource oriented partitioned scheduling* each shared resource is assigned to one core and tasks execute the critical sections related to a specific resource on the assigned core which can be different than the core which the task itself is assigned to. Since resources are bound to cores under this approach the authors could use PCP resource sharing protocol developed for uniprocessor systems but when a task enters a critical section it is suspended and migrates to the core where the resource is allocated to execute the related critical section. In [99] a suspension-based k-exclusion locking protocol for global scheduling with job-level static-priority has been presented. This protocol is asymptotically optimal under suspension-oblivious schedulability analysis.

Chapter 3

Conclusions

3.1 Summary

In this thesis, we have improved resource sharing techniques by modifying the task handling policy upon blocking for fixed-priority partitioned scheduling. We have also extended resource sharing protocols to handle mutual exclusive access to shared resources in the context of hybrid scheduling where both partitioned and global scheduling coexist.

In Papers A, B and C we enhanced the 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 arbitrarily from the range proposed by the model. We identified five taxonomy for the way spin-lock priorities are used by tasks in a system. Tasks can use a spin-lock priority: (*i*) per-core, (*ii*) per-task, (*iii*) per-resource, (*iv*) per-request and (*v*) combination of any above. In this thesis we have mainly focused on type (*i*). Two specific spin-lock priority variants of type (*i*) are: (*i-1*) all tasks on a core use the highest priority of any task using a global resource on the core for spinning, and (*i-2*) all tasks on a core use the highest priority of any task using either a global or a local resource on the core. We showed through simulation-based experiments that by using the new spin-lock protocol variant (*i-1*) the worst-case response times are improved up to 40% compared to the non-preemptive spin-based protocol. We mathematically proved that the range of spin-lock priorities of type (*i*) that use spin-lock priority variant (*i-1*) or higher, bounds the number of stacks used per core to only two. Further, we showed by both simulation-based experiments

as well as mathematical proofs that the spin-lock protocol (*i-2*) dominates the non-preemptive spin-based protocol (when overheads are ignored). Moreover, we showed by means of example cases that the spin-lock protocol variant (*i-1*) is incomparable with both the spin-lock protocol variant (*i-2*) and the non-preemptive spin-lock protocol. We provided an optimal algorithm from type (*i*) spin-lock priorities where the results showed an improvement of up to 38% compared to both the suspension-based and non-preemptive spin-based protocols. From type (*ii*) we introduced a spin-lock protocol variant (*ii-1*) for which all tasks on a core use their original priority for spinning. We showed, by means of example cases, that the spin-lock protocol variant (*ii-1*) and the classical suspension-based protocol are incomparable.

In papers E, F and G, we have enabled resource sharing in a family of multiprocessor platforms with a hybrid scheduling structure including both partitioned and global scheduling, 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. We provided two resource sharing protocols for semi-partitioned scheduling where some tasks in the system are split over multiple cores and can migrate among those cores during run-time while the rest are assigned and executed on only one core. In the first protocol, which is a centralized approach, all critical sections of a split task execute on a predefined core. In the second protocol, the critical section executes on the core where the request occurs. For both protocols we provided the analysis where we showed that the blocking terms are bounded. Based on a comparative evaluation, we concluded that in the presence of high migration costs the centralized approach outperforms the other approach. Further, we have considered another class of hybrid scheduling family called *SDS* where two sets of tasks exist on the platform: a set of tasks that is partitioned over the platform and use partitioned scheduling, and a set of tasks that is scheduled globally within the servers which are assigned to cores that have remaining bandwidth from the partitioned tasks. We have provided resource sharing for SDS by extending the existing protocols to be adjusted for such a complex structure. We have provided blocking bounds and extended the existing worst-case response time analysis for SDS in the context of two instantiations of such a system where (*a*) the set of partitioned and globally scheduled tasks tasks do not share resource with each other, and (*b*) all tasks in the system can share resources with each other.

3.2 Future Work

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 cores. When tasks share resources in the system, an extra delay is incurred to tasks due to blocking. How to allocate tasks to cores 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 cores 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 core. 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-based protocols from our proposed flexible spin-lock model in Paper C.

2. *Optimizing spin-based protocols.* Towards optimizing the spin-lock protocols to achieve better timing performance, the potential of other spin-lock priority types from the taxonomy of spin-lock protocols such as per-task type is remained for future exploration. Moreover, the provided analysis for the spin-lock priorities can be analyzed by means of optimization tools such as ILP to tighten blocking bounds similar as in [19].

3. *Thorough implementation of FSLM.* Currently for a limited range of spin-lock priorities regarding type (i) from FSLM an efficient implementation is available for which the related overheads has been identified and incorporated in the analysis. Similarly, an efficient implementation for other parts and types of spin-lock priorities from FSLM can be explored in order to identify all possible overheads induced by implementing FSLM to achieve a more realistic model.

Bibliography

[1] 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.

[2] Threadx real-time operating system. 2013 http://www.autosar.org. `http://rtos.com/products/threadx/`.

[3] Erika enterprise OS. June 2016. `http://erika.tuxfamily.org/drupal/`.

[4] 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.

[5] AUTOSAR release 4.1, specification of operating system. 2013 http://www.autosar.org.

[6] H. Zhu, S. Goddard, and M.B. Dwyer. Response time analysis of hierarchical scheduling: The synchronized deferrable servers approach. In 32^{nd} *IEEE Real-Time Systems Symposium (RTSS)*, pages 239–248, Nov. 2011.

[7] 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.

[8] 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.

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

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

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

[12] http://www.runtime-verification.org, access date October 2014.

[13] 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 22nd *IEEE Real-Time Systems Symposium (RTSS)*, pages 73–83, Dec. 2001.

[14] R. Rajkumar, L. Sha, and J.P. Lehoczky. Real-time synchronization protocols for multiprocessors. In 9th *IEEE Real-Time Systems Symposium (RTSS)*, pages 259–269, Dec. 1988.

[15] R. Rajkumar. Real-time synchronization protocols for shared memory multiprocessors. In 10th *International Conference on Distributed Computing Systems (ICDCS)*, pages 116–123, May 1990.

[16] J.H. Anderson, R. Jain, and K. Jeffay. Efficient object sharing in quantum-based real-time systems. In 19th *IEEE Real-Time Systems Symposium (RTSS)*, pages 346–355, Dec. 1998.

[17] T.S. Craig. Queuing spin lock algorithms to support timing predictability. In 14th *IEEE Real-Time Systems Symposium (RTSS)*, pages 148–157, Dec. 1993.

[18] L. I. Kontothanassis, R. W. Wisniewski, and M. L. Scott. Scheduler-conscious synchronization. *ACM Transactions on Computer Systems*, 15(1):3–40, Feb. 1994.

[19] A. Wieder and B.B. Brandenburg. On spin locks in AUTOSAR: Blocking analysis of FIFO, unordered, and priority-ordered spin locks. In 34th *IEEE Real-Time Systems Symposium (RTSS)*, pages 45–56, Dec. 2013.

[20] OSEK group. Osek/vdx operating system. Technical report, Feb. 2005. http://portal.osekvdx.org/files/pdf/specs/os223.pdf.

[21] Altera Nios 2 Processor. 2016. https://www.altera.com/products/processors/overview.html.

[22] Evidence S.r.l. Erika enterprise manual for the altera Nios II target - the multicore rtos on fpgas (version 1.2.3). Technical report, Dec. 2012. http://download.tuxfamily.org/erika/webdownload/manuals_pdf/arch_nios2_1_2_3.pdf.

[23] M. Verwielen. Performance of resource access protocols. Master's thesis, Eindhoven University of Technology (TU/e), July 2016. https://pure.tue.nl/ws/files/46944123/854620-1.pdf.

[24] S. Afshar, M.P.W. Verwielen, P. Gai, M. Behnam, and R. J. Bril. An implementation of the flexible spin-lock model in erika enterprise on a multi-core platform. In 12^{th} *Annual Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT)*, pages 55–60, July 2016.

[25] S.M.N. Balasubramanian. Flexible spin-lock model: Analysis and implementation. Master's thesis, Eindhoven University of Technology (TU/e), Jan. 2017. https://pure.tue.nl/ws/files/76280679/0978367_Afstudeerverslag_S.M.N._Balasubramanian.pdf.

[26] S.M.N Balasubramanian, Sara Afshar, Paolo Gai, Moris Behnam, and Reinder J. Bril. Incorporating implementation overheads in the analysis for the flexible spin-lock model. In 43^{rd} *Annual Conference of the IEEE Industrial Electronics Society (IECON)*, Oct. 2017.

[27] S.M.N Balasubramanian, Sara Afshar, Moris Behnam, Paolo Gai, and Reinder J. Bril. A dual shared stack for fslm in erika enterprise. In 23^{th} *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA) - WiP Session*, Aug. 2017.

[28] D. L. Parnas and P. C. Clements. A rational design process: How and why to fake it. *IEEE Transactions on Software Engineering*, SE-12(2):251–257, Feb. 1986.

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

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

[31] 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.

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

[33] 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.

[34] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. H. Anderson, and S. Baruah. *A categorization of real-time multiprocessor scheduling problems and algorithms in Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 30, pages 30.1–30.19. Chapman Hall/CRC, Boca, 2004.

[35] U. Devi. *Soft Real-Time Scheduling on Multiprocessors*. PhD thesis, University of North Carolina at Chapel Hill, 2006. `http://www.cs.unc.edu/xcms/wpfiles/dissertations/devi.pdf`.

[36] 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.

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

[38] D.-I. Oh and T.P. Baker. Utilization bounds for N-processor rate monotone scheduling with static processor assignment. *Real-Time Systems*, 15(1):183–192, Sep. 1998.

[39] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In 22^{nd} *IEEE Real-Time Systems Symposium (RTSS)*, pages 193–202, Dec. 2001.

[40] B. Andersson and J. Jonsson. The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. In 15^{th} *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 33–40, July 2003.

[41] 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, Oct. 2004.

[42] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys*, 43(4):35:1–35:44, Oct. 2011.

[43] F. Cerqueira, M. Vanga, and B. B. Brandenburg. Scaling global scheduling with message passing. In 19^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 263–274, April 2014.

[44] J. Y-T. Leung and J. Whitehead. *On the complexity of fixed-priority scheduling of periodic, real-time tasks.*, volume 2. Performance evaluation, 1982.

[45] A. D. Block. *Adaptive Multiprocessor Real-time Systems*. PhD thesis, Chapel Hill, NC, USA, 2008. AAI3315707.

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

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

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

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

[50] 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.

[51] 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.

[52] 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.

[53] J. Anderson, P. Holman, and A. Srinivasan. *Fair Scheduling of Real-time Tasks on Multiprocessors in Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 31. Chapman Hall/CRC, Boca, 2005.

[54] 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, May 2001.

[55] 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.

[56] 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.

[57] 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.

[58] 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, July 2007.

[59] 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.

[60] G. Lipari and E. 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, 2010.

[61] 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.

[62] 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.

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

[64] H. Takada and K. Sakamura. Real-time synchronization protocols with abortable critical sections. In 1^{st} *International Workshop on Real-time Computing Systems and Application*, pages 48–52, 1994.

[65] K.-Y. Lam, K.-K. Cheung, and J.K.-Y. Ng. A conditional abortable priority ceiling protocol for real-time systems with mixed tasks. In 9^{th} *Euromicro Workshop on Real-Time Systems*, pages 102–109, Jun 1997.

[66] M. Asberg, T. Nolte, and M. Behnam. Resource sharing using the roll-back mechanism in hierarchically scheduled real-time open systems. In 19^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 129–140, April 2013.

[67] 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.

[68] H. Takada and K. Sakamura. Predictable spin lock algorithms with preemption. In 11^{th} *IEEE Workshop on Real-Time Operating Systems and Software (RTOSS)*, pages 2–6, May 1994.

[69] 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.

[70] 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.

[71] 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.

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

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

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

[75] S. Zhao, J. Garrido, A. Burns, and A. Wellings. New schedulability analysis for mrsp. In 23th *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Aug. 2017.

[76] J. M. Mellor-Crummey and M. L. Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 9(1):21–65, Feb. 1991.

[77] T. Johnson and K. Harathi. A prioritized multiprocessor spin lock. *IEEE Transactions on Parallel and Distributed Systems*, 8(9):926–933, Sep. 1997.

[78] A. Biondi, B. B. Brandenburg, and A. Wieder. A blocking bound for nested fifo spin locks. In 37th *IEEE Real-Time Systems Symposium (RTSS)*, pages 291–302, Nov. 2016.

[79] C. E. Nemitz, K. Yang, M. Yang, P. Ekberg, and J. H. Anderson. Multiprocessor real-time locking protocols for replicated resources. In 28th *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 50–60, July 2016.

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

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

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

[83] 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 International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*.

[84] B. Brandenburg and H. Anderson. A comparison of the M-PCP, D-PCP, and FMLP on LITMUSRT. In 12^{th} *International Conference on Principles of Distributed Systems (OPODIS)*, pages 105–124, Berlin, Heidelberg, Dec. 2008. Springer-Verlag.

[85] 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.

[86] 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.

[87] 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.

[88] 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, July 2011.

[89] S. Afshar, N. Khalilzad, F. Nemati, and T. Nolte. Resource sharing among prioritized real-time applications on multiprocessors. *SIGBED Rev.*, 12(1):46–55, Feb. 2015.

[90] C.-M. Chen and S. K. Tripathi. Multiprocessor priority ceiling based protocols. Technical report, College Park, MD, USA, 1994.

[91] B. B. Brandenburg, J. M. Calandrino, A. Block, H. Leontyev, and J. H. Anderson. Real-time synchronization on multiprocessors: To block or not to block, to suspend or spin? In 14^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 342–353, April 2008.

[92] F. Nemati, M. Behnam, and T. Nolte. Multiprocessor synchronization and hierarchical scheduling (icppw). In *International Conference on Parallel Processing Workshops*, pages 58–64, Sep. 2009.

[93] G. Macariu and V. Cretu. Limited blocking resource sharing for global multiprocessor scheduling. In 23^{rd} *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 262–271, July 2011.

[94] N. Guan, M. Stigge, W. Yi, and G. Yu. New response time bounds for fixed priority multiprocessor scheduling. In 30^{th} *IEEE Real-Time Systems Symposium (RTSS)*, pages 387–397, Dec. 2009.

[95] 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.

[96] F. Nemati, T. Nolte, and M. Behnam. Partitioning real-time systems on multiprocessors with shared resources. In 14^{th} *International Conference on Principles of Distributed Systems (OPODIS)*, pages 253–269, Dec. 2010.

[97] 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, Jun. 2013.

[98] W. H. Huang, M. Yang, and J. J. Chen. Resource-oriented partitioned scheduling in multiprocessor systems: How to partition and how to share? In 37^{th} *IEEE Real-Time Systems Symposium (RTSS)*, pages 111–122, Nov. 2016.

[99] G. A. Elliott and J. H. Anderson. An optimal k-exclusion real-time locking protocol motivated by multi-gpu systems. *Real-Time Systems*, 49:140–170, 2011.