Multiprocessor Synchronization and Hierarchical Scheduling*

Farhang Nemati, Moris Behnam , Thomas Nolte
Mälardalen Real-Time Research Centre
P.O. Box 883, SE-721 23 Västerås, Sweden
farhang.nemati@mdh.se

Abstract

Multi-core architectures have received significant interest as thermal and power consumption problems limit further increase of speed in single-cores. In the multi-core research community a considerable amount of work has been done on real-time multi-core scheduling algorithms where it is assumed tasks are independent. However, synchronization of dependent tasks executing on multi-cores has not received as much attention, even though typical real-time systems in practice include tasks that share resources.

In this paper we propose a synchronization protocol for hierarchically scheduled multi-core systems, and we present a comparison between the presented protocol and existing multi-core synchronization protocols. The presented protocol groups dependent tasks that directly or indirectly share mutually exclusive resources into independent components. Within a component dependent tasks use classical uniprocessor synchronization protocols, such as the Stack-based Resource allocation Protocol. The components are then scheduled on the cores by a global scheduler.

There are two major approaches for scheduling multi-core: partitioned and global scheduling. While most existing multi-core synchronization protocols support only one category, the protocol presented in this paper is developed to handle both scheduling approaches. The presented approach is developed to allow for co-execution of existing legacy real-time applications along with new applications, i.e., a legacy application is put into one or more components preserving its own (original) scheduling and synchronization protocols.

1 Introduction

Due to the problems with power consumption and related thermal problems, processor architects are moving toward multi-core designs. A multi-core processor is a combination of two or more independent cores on a single chip, and multi-core is today the dominating technology for desktop computing.

Real-time systems can benefit highly from the multi-core processors, as critical functionality can be assigned dedicated cores, and independent tasks can execute concurrently to improve the performance and thereby enable new functionalities. Moreover, since the cores are located on the same chip and typically have shared memory, communication between cores is very fast. The performance improvements of using multi-core processors depend on the nature of the applications as well as the implementation of the software. To take advantage of the concurrency offered by a multi-core architecture, appropriate scheduling algorithms and synchronization protocols are required. However, in the research community, scheduling has received much more attention than synchronization [8]. Thus there is a need for further research efforts to achieve efficient and adequate synchronization techniques.

In this paper, we propose a synchronization protocol MHSP (Multiprocessor Hierarchical Synchronization Protocol) that can be used in multi-core systems.

1.1 Multi-core scheduling

There are two main approaches for scheduling sporadic and periodic task systems on multi-core systems [2, 4, 11, 15] which are inherited from multiprocessor systems; global and partitioned scheduling. Under global scheduling, e.g., G-EDF (Global Earliest Deadline First), tasks are scheduled based on their priorities by a single scheduler and each task can be executed on any core. A single global queue is used for storing jobs. A task as well as a job can be preempted on one core and resumed on another core (migration of tasks among cores is permitted). Under partitioned scheduling tasks are statically assigned to cores and tasks within each core are scheduled by uniprocessor scheduling such as FPS (Fixed Priority Scheduling) or EDF (Earliest Deadline First). Each core is associated with a separate ready queue for scheduling task jobs. A combination of global and partitioned scheduling is the two-level hybrid scheduling [11], which is very useful for systems in which

*The work in this paper is supported by the Swedish Foundation for Strategic Research (SSF), via the research programme PROGRESS.

some tasks cannot migrate between cores while other tasks can migrate. An efficient multiprocessor scheduling approach based on hierarchical scheduling framework is presented by Shin et al. [22], however, the approach is suitable for independent tasks, i.e., tasks are not allowed to share mutually exclusive resources.

2 Related work

The uniprocessor synchronization protocols PCP (Priority Ceiling Protocol) [21] and SRP (Stack-based Resource allocation Protocol) [3] are two of the best known methods for synchronization in uniprocessor systems. Both protocols avoid deadlocks, and blocking times are limited to at most the duration of one outermost critical section.

In the context of uniprocessor hierarchical scheduling, there have been studies on allowing for sharing of mutually exclusive resources within components [1, 16] and across components [9, 7, 12].

For multiprocessor systems, Rajkumar present MPCP (Multiprocessor Priority Ceiling Protocol) [20], which extends PCP to multiprocessors hence allowing for synchronization of tasks sharing mutually exclusive resources using partitioned FPS. Gai et al. [13] present MSRP (Multiprocessor SRP), which is an EDF-based synchronization protocol for multiprosessors. The shared resources are classified as either (i) local resources that are shared among tasks assigned to the same processor, or (ii) global resources that are shared by tasks assigned to different processors. In MSRP, tasks synchronize local resources using SRP and access to global resources is guaranteed a bounded blocking time. Lopez et al. [19] present an implementation of SRP under P-EDF (Partitioned EDF). The tasks that directly or indirectly share resources, called macrotasks, shall be assigned to the same processor. This method is, however equivalent to MSRP with no global resources. Devi et al. [10] present a synchronization technique under G-EDF. The work is restricted to synchronization of non-nested accesses to short, simple objects, e.g., stacks, linked lists, and queues. In addition, the main focus of the method is on soft real-time systems. Block et al. [8] present FMLP (Flexible Multiprocessor Locking Protocol), which is the first synchronization protocol for multiprocessors that can be applied to both partitioned and global scheduling algorithms, i.e., P-EDF and G-EDF.

Since we will compare our protocol to other multiprocessor synchronization protocols we will discuss three of them in more detail:

2.1 MPCP

MPCP works as follows: The local resources are protected using PCP. A task blocked on a global resource suspends and makes the processor available for the local tasks. The duration of time a task is blocked includes remote blocking where a task is blocked by tasks (with any priority) running on another processor (core). However, the remote blocking of a task is bounded and is a function of the duration of critical sections of other tasks. MPCP makes this possible by assigning global critical sections (*gcs*) a ceiling greater than the highest priority among all tasks, hence a *gcs* can only be blocked by another *gcs* and not by a non-critical section. The blocked global critical sections on a global resource are added to a prioritized queue. Global critical sections cannot be nested in local critical sections (*lcs*) and vice versa. Global resources potentially lead to high blocking times, thus tasks sharing the same resource have to be assigned to the same processor as long as possible.

2.2 MSRP

Under MSRP, when a task is blocked on a global resource it performs spin lock (busy wait). This means that the processor is kept busy without doing any work, hence the duration of spin lock should be as short as possible which means locking a global resource should be reduced as far as possible. To achieve this goal under MSRP, the tasks executing in global critical sections become non-preemptable. The tasks blocked on a global resource are added to a FIFO queue. Global critical sections are not allowed to be nested under MSRP. Gai et al. [14] compare MSRP and MPCP. They point out the complexity of implementation is one of the disadvantages of MPCP while wasting local processor time (due to busy wait) is a disadvantage of MSRP. They have performed two case studies for the comparison and the results show that MPCP works better when the duration of global critical sections are increased while MSRP outperforms MPCP when critical sections become shorter. Also they show that for applications where tasks access many resources, and resources are accessed by many tasks, using MPCP results in more pessimism compared to using MSRP, which can be considered as a significant advantage for MSRP.

2.3 FMLP

Using FMLP, resources are categorized into short and long resources which are user defined. There is no limitation on nesting resource accesses, except that requests for long resources cannot be nested in requests for short resources. Under FMLP and using P-EDF or G-EDF, the tasks that are blocked on short resources perform busy wait and are added to a FIFO queue. Tasks that access short resources execute non-preemptively. Tasks blocked on a long resource are added to a FIFO queue. A task holding a long resource under G-EDF, executes preemptively using priority inheritance, i.e., it inherits the priority of highest priority task that the running task blocks. Under P-EDF a

task holding a long resource executes non-preemptively using local priority inheritance, i.e., priority is inherited only from tasks assigned to the same processor. Note that under P-EDF the concept of short and long resources is only applied to global resources. SRP can be used for sharing local resources. In FMLP, deadlock is prevented by grouping resources. A group includes either global or local resources, and two resources are in the same group if a request for one of them may be nested in a request for the other one. A group lock is assigned to each group and only one task at any time can hold the lock.

3 System model and background

This paper focuses on synchronization of tasks that share mutually exclusive resources in a multiprocessor system consisting of m identical processors. We assume a general multiprocessor scheduling which can be either partitioned or global scheduling. The following sections explain our corresponding task model and approach for mutual exclusion.

3.1 Task model

The task model considered in this paper is the deadline-constrained sporadic hard real-time task model $\tau_i(T_i, C_i, D_i, \{c_{i,j}\})$, where T_i is a minimum separation time between arrival of successive jobs of τ_i, C_i is their WCET (Worst-Case Execution Time), and D_i is an arrival-relative deadline ($0 < C_i \leq D_i \leq T_i$) before which the execution of a job must be completed. Each task is allowed to access one or more shared logical resources either serially or properly nested. Each element $c_{i,j}$ in $\{c_{i,j}\}$ represents the WCET of the task τ_i inside a critical section of the shared resource R_j.

3.2 Shared resources and SRP protocol

Tasks are allowed to share logical resources in a mutually exclusive manner. To access a resource R_j, a task must first lock the resource, and when the task no longer needs the resource it is unlocked. The time during which a task holds a lock is called a critical section. At any time, only a single task may hold the lock of a logical resource. The SRP protocol is used to synchronize the tasks' access to shared resources. According to SRP, each task τ_i has a preemption level equal to $\pi_i = 1/D_i$, where D_i is the relative deadline of the task. And each shared resource R_j is associated with a resource ceiling $rc_j = \max\{\pi_i | \tau_i \text{ accesses } R_j\}$. Finally, a system ceiling is used which equals to the currently locked highest resource ceiling in the system. Following the rules of SRP, a job J_i that is generated by a task τ_i can preempt the currently executing job J_k within a subsystem only if J_i has a priority higher than that of job J_k and, at the same time, the preemption level of τ_i is greater than the current subsystem ceiling.

4 MHSP

This section describes our proposed synchronization protocol MHSP (Multiprocessor Hierarchical Synchronization Protocol). Using MHSP, all tasks that are directly and indirectly dependent through sharing of mutually exclusive logical resources are grouped into one component[1]. A component contains a set of dependent tasks, and a local scheduler. Hence, the system contains both independent tasks and components. Whenever the system scheduler (global scheduler or core scheduler depending on the type of the multiprocessor scheduling algorithm) selects a component to be executed, its local scheduler will select which of its internal tasks that will get access to the CPU resource. Inside each component, the SRP protocol is used to synchronize access to mutually exclusive logical resources. For each component, the timing interface is used to abstract the timing requirements of all internal tasks. The timing interface is calculated using the periodic resource model presented [23], i.e., each component $C\tau_s$ is associated with (P_s, Q_s), where Q_s is the budget that the component will receive every period P_s. For the system scheduler, a component can be considered as a simple periodic task with execution time Q_s and period P_s.

For multiprocessor partitioned scheduling the component timing interface (P_s, Q_s) can also be used, in addition to scheduling, for the task to processor allocation by allocating independent tasks and components to processors which simplify the allocation problem since tasks and components are independent. For the same reason, any global scheduler can be used without any modification since most global scheduling algorithms assume that tasks are independent . Figure 1 shows the system framework, i.e., a two level hierarchical multiprocessor global scheduling framework.

4.1 Component timing interface

In this section, we will explain how to evaluate the component timing interface including the component budget Q_s and period P_s. The ratio Q_s/P_s should be as low as possible in order to make the MHSP protocol efficient in terms of requiring less CPU resources and while guaranteing the schedulability of its internal tasks. As shown in Fig.(1), the framework is a two level hierarchical framework that schedules independent components and tasks, and it allows sharing of mutually exclusive logical resources inside the components. The algorithm presented in [23] to evaluate the component timing interface can be used here only if we

[1]This technique is similar to the technique presented in [19], the main difference being the way of evaluating the group parameter and in addition our protocol suitable for both global and partitioned schedulers

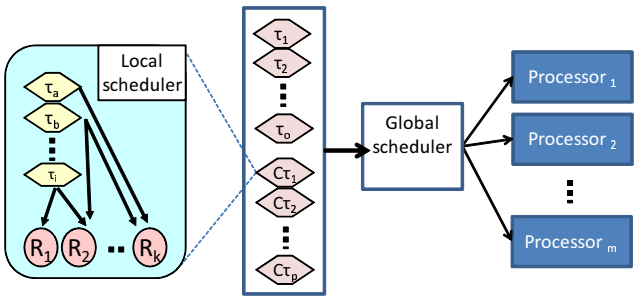

Figure 1. Multiprocessors global scheduling using the MHSP protocol .

take into account the sharing of logical resources inside the components, and these resources are called local resources. Note that the analysis presented in [23] is for single processor systems only. However, since the tasks in a component are not allowed to execute in parallel, i.e., only one task is allowed to execute in one processor at each time, the tasks inside a component will execute as they are executing in a single processor which makes the analysis of [23] valid also here.

The component local schedulability analysis using EDF and FPS scheduling are shown below;

Let $\texttt{dbf}_{\texttt{EDF}}(i,t)$ denote the demand bound function of a task τ_i under EDF scheduling [5], i.e.,

$$\texttt{dbf}_{\texttt{EDF}}(i,t) = \left\lfloor \frac{t+T_i-D_i}{T_i} \right\rfloor \cdot C_i. \qquad (1)$$

The local schedulability condition under EDF scheduling is then (by combining the results of [6] and [23])

$$\forall t > 0 \quad \sum_{i=1}^{n} \texttt{dbf}_{\texttt{EDF}}(i,t) + b(t) \leq \texttt{sbf}(t), \qquad (2)$$

where $b(t)$ is the blocking function [6] that represents the longest blocking time during which a task τ_i with $D_i \leq t$ may be blocked by a task τ_k with $D_k > t$. $\texttt{sbf}_s(t)$ is the supply bound function [23] that computes the minimum possible CPU supply that a component may receive from the global scheduler for every interval length t as follows:

$$\texttt{sbf}_s(t) = \begin{cases} t-(k+1)(P-Q) & \text{if } t \in V^{(k)} \\ (k-1)Q & \text{otherwise,} \end{cases} \qquad (3)$$

where $k = \max\left(\lceil (t-(P-Q))/P \rceil, 1\right)$, and $V^{(k)}$ denotes an interval $[(k+1)P-2Q, (k+1)P-Q]$.

For FPS [17], let $\texttt{rbf}_{\texttt{FP}}(i,t)$ denote the request bound function of a task τ_i, i.e.,

$$\texttt{rbf}_{\texttt{FP}}(i,t) = C_i + \sum_{\tau_k \in \texttt{HP}(i)} \left\lceil \frac{t}{T_k} \right\rceil \cdot C_k, \qquad (4)$$

where $\texttt{HP}(i)$ is the set of tasks with priorities higher than that of τ_i. The local schedulability analysis under FPS is then (by extending the results of [3, 23]):

$$\forall \tau_i, 0 < \exists t \leq D_i, \;\; \texttt{rbf}_{\texttt{FP}}(i,t) + b_i \leq \texttt{sbf}(t), \qquad (5)$$

where b_i is the maximum *blocking* (i.e., extra CPU demand) imposed to a task τ_i when τ_i is blocked by lower priority tasks that are accessing resources with ceiling greater than or equal to the priority of τ_i. Note that t can be selected within a finite set of scheduling points [18].

Given the component period P_s, let $\texttt{calculateBudget}(C\tau_s, P_s)$ denote a function that calculates the smallest component budget Q_s that satisfies Eq. (2) for EDF or Eq. (5) for FPS scheduling. Hence, $Q_s = \texttt{calculateBudget}(C\tau_s, P_s)$. The function is similar to the one presented in [23]. Note that selecting the value of P_s is not a trivial; the lower this value is, the lower the utilization of the component will and at the same time the context switch overhead will increase (more frequent context switches). However, in this paper we restrict the value of P_s to fulfil the inequality $2P_s \leq T_m$, where τ_m is the task with the shortest period. This restriction is motivated by reasons of resource efficiency [24].

4.2 Protocol properties

The MHSP protocol has the following advantages (+) and disadvantages (-):

+ Generality – the protocol can be used with any type of global scheduler, including a partitioned scheduler, without any need for modification in the used scheduling algorithm and corresponding schedulability analysis.

+ Synchronization – shared resources can be nested and the protocol is deadlock free. This property is inherited from using the SRP protocol inside the components.

+ Reduced pessimism – the protocol removes the effect of blocking from tasks that do not share resources.

+ Isolation – tasks executing inside a component do not follow directly under the global scheduling algorithm, i.e., tasks are scheduled inside a component using the component's local scheduler. Therefore, global schedulability is not dependent on individual task parameters and vice versa.

- Utilization – the component utilization Q/P must be less than 1, otherwise the protocol can not be used.

In addition, evaluating (P_s, Q_s) is based on worst-case CPU resource availability which can be pessimistic and will add unnecessary utilization to the component timing interface.

- Tasks resident in the same component can not be executed in parallel even if there are idle processors available.

Note that the the disadvantage showed in the last bullet can be handled by modifying both the protocol and the global scheduling. The modification can be done during runtime such that if there exists an idle processor then the global scheduler will allow ready tasks from components to execute even if their local scheduler did not select them. The priority of these tasks will be the lowest in a global level and they will stop to execute in case they want to access a mutual exclusive shared logical resource. If an independent task or another component is activated then the execution of these lowest priority tasks will be preempted. Note that this modification will improve the average response time for tasks and it will use the processors more efficiently. However, it will not improve the worst case when all tasks in a component want to access a mutual exclusive shared logical resource at the same time.

4.3 Example

We will explain the MHSP protocol using the following example. Suppose a system that have 7 tasks as shown in Table 1.

Task	T_i	C_i	D_i	R_j	$C_{i,j}$
τ_1	15	6	15	-	-
τ_2	20	4	20	R_1, R_2	2, 1
τ_3	40	6	40	R_1	1
τ_4	45	9	45	R_3	2
τ_5	60	12	60	-	-
τ_6	60	9	60	R_2	2
τ_7	90	14	85	R_3	3

Table 1. Task parameters of the example.

Tasks τ_2, τ_3 and τ_6 are grouped into a single component $C\tau_1$ since they are dependent (note that τ_3 and τ_6 are not directly dependent but since both are sharing resources with τ_2 then they are indirectly dependent). Tasks τ_4 and τ_7 are grouped into another component $C\tau_2$. We assume that both $C\tau_1$ and $C\tau_2$ are using EDF as a local scheduler. The component period of $C\tau_1$ is selected $P_1 = 10$ then the minimum budget should be $Q_1 = 5.35$ that satisfy Eq. (2) and for $C\tau_2$ the parameters are $P_2 = 20$ and $Q_2 = 9$. The sum of utilization of tasks τ_4 and τ_7 is equal to $U = 0.35$ while the utilization of component $C\tau_2$ that include these tasks is $Q_s/P_s = 0.45$ and the increment in the component utilization comes from both the blocking effect between tasks and the overhead from abstracting the timing of these tasks. Figure 2 shows the scheduling of tasks and components in two processors using global preemptive EDF scheduling. It is clear that in certain times, one or both processors are idle and at the same time there are ready tasks in the components that can not execute because of their local scheduler selected other task or the budget of the component has expired. It would be more efficient to allow the ready task in the components to execute as described in the previous section.

5 Brief comparison of the four protocols

In this section we briefly compare six properties of the four multiprocessor synchronization protocols MPCP, MSRP, FMLP and MHSP:

1 – Nested resource requests: In MPCP global resource requests cannot be nested in local requests and vice versa. MSRP does not support nested global requests. FMLP allows nested requests except nesting global requests within local requests. Since MHSP uses SRP locally inside components, there is no limitation on nested resource requests.

2 – Scheduling protocols: MPCP works only under partitioned FPS. MSRP works under partitioned EDF, e.g., P-EDF. FMLP supports both partitioned and global scheduling protocols, but it does not support FPS. In MHSP any scheduling protocol (partitioned or global) can be used. In addition tasks inside a component can be scheduled by any uniprocessor scheduling protocol.

3 – Preemption: In MPCP there is no need to change scheduling protocols to support non-preemptive execution of a task. The same goes for MHSP since it removes dependencies between tasks in the system level.On the other hand, critical sections accessing global logical resource under MSRP and short resources under FMLP need to execute non-preemptively, and this should be taken into account in the schedulability analysis.

4 – Utilization: A disadvantage of MHSP is that the utilization bound of a component should not exceed 1 while this does not limit other protocols. In addition, since the internal tasks in a component are not allowed to execute in parallel, it will increase the utilization of the component compared to FLMP that may allow the tasks to execute in parallel until they will try to access a locked resource.

5 – Blocking: In MHSP tasks that do not share mutually exclusive logical resources are not blocked because they are

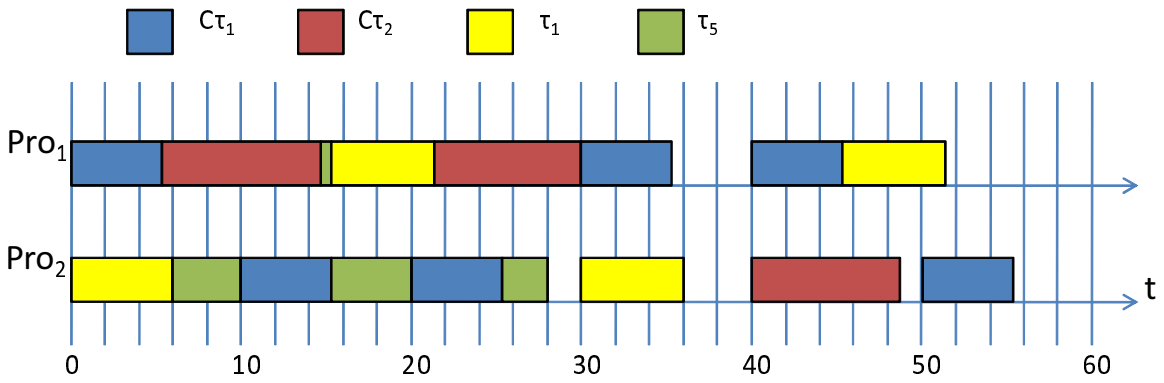

Figure 2. Scheduling tasks and component.

in different components. Each task may be blocked by at most one task with lower priority within the same component. In MPCP global critical sections execute at a priority higher than the highest priority task in the system, and thus they may block any task resident in the same processor. In MSRP and FMLP the effect of non-preemptivity and busy waiting adds more blocking overheads. In addition a task can be blocked by the critical sections of all tasks that access the same resource (i.e., the blocking time is the summation of all critical sections) under MSRP, FLMP and MPCP.

6 – Legacy systems: Existing legacy real-time systems usually rely on FPS. MHSP and MPCP are suitable for migration of those systems to a multi-core architecture, while MSRP and FMLP may be less suitable for this purpose since they are based on EDF scheduling.

6 Summary

In this paper we have proposed and presented MHSP; a synchronization protocol that can be used for multiprocessors systems. MHSP uses the principle of hierarchical scheduling to remove the dependencies between tasks making it suitable for use together with any multiprocessor scheduling algorithm. We have discussed the advantages and disadvantages of the protocol and, in addition, we have compared the protocol with the other existing protocols. Whether it is efficient to use MHSP depends on application characteristics and system requirements and paraments. For example MHSP has the potential to be used with legacy systems since these systems often are required to be scheduled using FPS; a scheduling policy supported by MHSP both at system level and at component level. The main drawback with the proposed approach is that it maps all direct and indirect dependent tasks into one component which may require higher utilization if the component contains many tasks. Allowing global resources in MHSP can be one way to improve it by splitting the high utilization components into multiple dependent components (general hierarchical scheduling framework that allows resource sharing inside the component and between components). Then an optimization algorithm should be developed such that it can find the best tasks allocation to components in order to minimize the system utilization. In summary, the proposed approach can be considered as the first step towards generalizing the use of hierarchical scheduling frameworks in multiprocessors systems.

References

[1] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In 4^{th} *ACM international conference on Embedded software (EMSOFT'04)*, Sep. 2004.

[2] T. Baker. A comparison of global and partitioned EDF schedulability test for multiprocessors. Technical report, January 2005.

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

[4] S. Baruah and N. Fisher. The partitioned multiprocessor scheduling of sporadic task systems. In *RTSS '05: Proceedings of the 26th IEEE International Real-Time Systems Symposium*, pages 321–329, Washington, DC, USA, 2005. IEEE Computer Society.

[5] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the* 11^th *IEEE International Real-Time Systems Symposium(RTSS'90)*, pages 182–190, Lake Buena Vista, Florida, USA, December 1990. IEEE Computer Society.

[6] S. K. Baruah. Resource sharing in EDF-scheduled systems: A closer look. In *Proceedings of the* 27^{th} *IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 379–387, Rio de Janeiro, Brazil, December 2006.

[7] M. Behnam, I. Shin, T. Nolte, and M. Nolin. SIRAP: a synchronization protocol for hierarchical resource sharing in

real-time open systems. In 7^{th} *ACM and IEEE Int. Conference on Embedded Software (EMSOFT'07)*, Oct. 2007.

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

[9] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In 27^{th} *IEEE Int. Real-Time Systems Symposium (RTSS'06)*, Dec. 2006.

[10] U. Devi, H. Leontyev, and J. Anderson. Efficient synchronization under global edf scheduling on multiprocessors. In *Real-Time Systems, 2006. 18th Euromicro Conference on*, pages 10 pp.–84, 0-0 2006.

[11] U. C. Devi. *Soft real-time scheduling on multiprocessors*. PhD thesis, Chapel Hill, NC, USA, 2006. Adviser-Anderson,, James H.

[12] N. Fisher, M. Bertogna, and S. Baruah. The design of an EDF-scheduled resource-sharing open environment. In 28^{th} *IEEE Real-Time Systems Symposium (RTSS'07)*, Dec. 2007.

[13] P. Gai, G. Lipari, and M. D. Natale. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In *RTSS '01: Proceedings of the 22nd IEEE Real-Time Systems Symposium*, page 73, Washington, DC, USA, 2001. IEEE Computer Society.

[14] P. Gai, M. D. 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 *RTAS '03: Proceedings of the The 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, page 189, Washington, DC, USA, 2003. IEEE Computer Society.

[15] P. H. J. A. J. Carpenter, S. Funk and S.Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. In *J. Y. Leung, editor, Handbook on Scheduling Algorithms, Methods, and Models*, pages 30.1–30.19. ChapmanHall/CRC, Boca Raton, Florida, 2004.

[16] T.-W. Kuo and C.-H. Li. A fixed-priority-driven open environment for real-time applications. In 20^{th} *IEEE International Real-Time Systems Symposium (RTSS'99)*, Dec. 1999.

[17] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Proceedings of the* 20^{th} *IEEE International Real-Time Systems Symposium(RTSS'89)*, pages 166–171, Santa Monica, CA, USA, December 1989. IEEE Computer Society.

[18] G. Lipari and E. Bini. A methodology for designing hierarchical scheduling systems. *J. Embedded Comput.*, 1(2):257–269, 2005.

[19] 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, 2004.

[20] R. Rajkumar. *Synchronization in multiple processor systems. In Synchronization in Real-Time Systems: A Priority Inheritance Approach*. Kluwer Academic Publishers, 1991.

[21] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In 9^{th} *IEEE Int. Real-Time Systems Symposium (RTSS'88)*, Dec. 1988.

[22] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. Technical report, July 2008.

[23] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In 24^{th} *IEEE International Real-Time Systems Symposium (RTSS'03)*, Dec. 2003.

[24] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *Trans. on Embedded Computing Sys.*, 7(3):1–39, 2008.