Investigation of implementing a synchronization protocol under multiprocessors hierarchical scheduling
Nemati, F.; Behnam, M.; Bril, R.J.

Published in:

DOI:
10.1109/ETFA.2009.5347221

Published: 01/01/2009

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 10. Sep. 2017
Investigation of Implementing a Synchronization Protocol under Multiprocessors Hierarchical Scheduling*

Farhang Nemati¹, Moris Behnam¹, Thomas Nolte¹ and Reinder J. Bril²
¹Mälardalen Real-Time Research Centre, Mälardalen University, Sweden
²Eindhoven University of Technology (TU/e), The Netherlands

Abstract

In the multi-core and multiprocessor domain, there has been considerable work done on scheduling techniques assuming that real-time tasks are independent. In practice a typical real-time system usually share logical resources among tasks. However, synchronization in the multiprocessor area has not received enough attention.

In this paper we investigate the possibilities of extending multiprocessor hierarchical scheduling to support an existing synchronization protocol (FMLP) in multiprocessor systems. We discuss problems regarding implementation of the synchronization protocol under the multiprocessor hierarchical scheduling.

1. Introduction

Multi-core and multiprocessor architectures are receiving more interest due the performance they offer as improving performance in single-core architectures is limited due to the problems with power consumption and related thermal problems.

To take advantage of the performance offered by a multi-core/multiprocessor architecture, appropriate scheduling algorithms and synchronization protocols are required. However, in the research community, scheduling has received much more attention than synchronization [7].

There are two main approaches for scheduling sporadic and periodic task systems on multiprocessor architectures [2, 3, 12, 16]: partitioned and global scheduling. Under partitioned scheduling tasks are statically assigned to processors and tasks within each processor are scheduled by uniprocessor scheduling such as FPS (Fixed Priority Scheduling) or EDF (Earliest Deadline First). Under global scheduling, e.g., G-EDF (Global Earliest Deadline First), tasks are scheduled by a single scheduler and each task can be executed on any core. A combination of global and partitioned scheduling called the two-level hybrid scheduling [11] is used for systems in which some tasks cannot migrate between cores while other tasks can migrate.

A more general approach which is a generalization of partitioned and global scheduling is called cluster-based scheduling [20]. In this approach tasks are statically assigned to clusters and tasks within each cluster are globally scheduled. In turn, clusters are transformed into tasks and scheduled on multiprocessor architectures. Cluster-based scheduling seems to be the way to improve utilization bounds on the multiprocessor platform. However the existing approaches for cluster-based scheduling do not consider synchronization and assume that tasks are independent.

The contribution of this paper is to investigate the implementation problems when extending the hierarchical scheduling framework for multiprocessor virtual clustering presented in [20], to support lock-based synchronization. We will spend more time explaining the framework later. We have specifically discussed problems regarding implementing an existing protocol for synchronization in multiprocessors [7] under the hierarchical scheduling framework.

Related work. In the context of uniprocessor hierarchical scheduling, there have been studies on allowing for sharing of mutually exclusive resources within components [1, 17] and across components [5, 10, 13].

For multiprocessor systems, Rajkumar present MPCP (Multiprocessor Priority Ceiling Protocol) [19], which extends PCP to multiprocessors hence allowing for synchronization of tasks sharing mutually exclusive resources using partitioned FPS. Gai et al. [14, 15] present MSRP (Multiprocessor SRP), which is a P-EDF (Partitioned EDF) based synchronization protocol for multiprocessors. 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. [18] present an implementation of SRP under P-EDF. Devi et al. [11] 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. [7] 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. An implementation of FMLP has been described in [8]. Our goal is to implement this protocol for synchronization under the hierarchical scheduling for multiprocessor virtual clustering; hence we will spend more time on details of this protocol in Section 3. However extending the multiprocessor hierarchical scheduling to support FMLP is not trivial. Firstly some assumptions of the scheduling framework for the schedulability analysis should

---

* This work was partially supported by the Swedish Foundation for Strategic Research (SSF) via the strategic research centre (PROGRESS) at Mälardalen University.
be changed and new schedulability test should be derived which considers blocking times. Secondly implementing FMLP under the scheduling protocol introduces problems (Section 4). This requires considerable effort to successfully implement FMLP under the multiprocessor hierarchical scheduling protocol. In this paper we focus on the second part which deals with implementation problems.

2. Task and system model

We assume a sporadic task model [4] in which a sporadic task \( \tau_i \) is specified by its minimum inter arrival time \( T_i \), its worst-case execution time \( C_i \), and its relative deadline \( D_i \). We refer to the \( j^{th} \) job (each being an instance of a task) of task \( \tau_i \) as \( \tau_i^j \).

A request \( R \) issued by a job for exclusive access to a resource \( l \) is satisfied as soon as the job holds the resource. A request which is not contained within any other request is called an outermost.

We assume a multiprocessor system consisting \( m \) identical, unit-capacity processors each of which has a scheduling utilization of one. We also assume that migration of a job is allowed, i.e., a job can be preempted on one processor and be resumed on another processor. Preemption and migration overheads are assumed to be negligible.

3. FMLP

Under the FMLP, resources are categorized into short and long resources which is user defined. There is no limitation on nesting resource accesses, except that requests for long resources cannot be nested in requests for short 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 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.

The jobs that are blocked on short resources perform busy-wait and are added to a FIFO queue. Jobs that access short resources hold the group lock and execute non-preemptively. A job accessing a long resource under G-EDF holds the group lock and executes preemptively using priority inheritance, i.e., it inherits the maximum priority of any higher priority job blocked on any resource within the same group. Tasks blocked on a long resource are added to a FIFO queue.

Actually FMLP works under a variant of G-EDF for suspendable and preemptable jobs (GSN-EDF) [7] which guarantees that a job \( \tau_i^j \) can only be blocked (with a constraint duration) by another non-preemptable job when job \( \tau_i^j \) is released or resumed.

3.1. Blocking under GSN-EDF and FMLP

Busy-wait blocking of task \( \tau_i \) is the maximum duration of time that any job of the task can busy-wait on a short resource.

Non-preemptive blocking occurs when a preemptable job \( \tau_i^j \) is one of the \( m \) highest priority jobs but it is not scheduled because a lower priority job is non-preemptively executing instead. Non-preemptive blocking of task \( \tau_i \) is the maximum duration time that any job of task \( \tau_i \) is non-preemptively blocked.

Direct blocking occurs when job \( \tau_i^j \) is one of the \( m \) highest priority jobs but it is suspended because it issues a request for an outermost long resource from group \( G \) but another job holds a resource from the same group (holds the group’s lock). Direct blocking of task \( \tau_i \) is the maximum duration of time that any job of the task can be direct blocked.

4. Hierarchical scheduling for multiprocessor virtual clustering

Under cluster-based scheduling tasks are statically assigned to clusters and scheduled globally among themselves (intra-cluster scheduling). A cluster is a set of \( m' \) processors where \( m' \leq m \). A cluster with its tasks and scheduler is denoted as a component. The clusters are in turn globally scheduled on the multiprocessor (inter-cluster scheduling). The cluster-based scheduling is a generalization of partitioned and global scheduling, i.e., it is equivalent to partitioned scheduling if tasks are assigned to \( m \) clusters where \( m' = 1 \) for each cluster, and it is equivalent to global scheduling if all tasks are assigned to a single cluster where \( m' = m \).

Cluster-based scheduling can be physical or virtual. In physical cluster-based scheduling each of cluster’s \( m' \) processors are statically mapped to one of \( m \) processors of the multiprocessor [9]. In the virtual cluster-based scheduling the \( m' \) processors of each cluster are dynamically mapped (one-to-many) onto \( m \) processors of the multiprocessor. Virtual clustering is more general and less sensitive to task-cluster mapping compared to physical clustering.

Physical clustering only needs the intra-cluster scheduling because the clusters do not share processors. On the other hand, virtual clustering requires a hierarchical scheduling which includes intra-cluster and inter-cluster scheduling. Under hierarchical scheduling processors of the multiprocessor are dynamically assigned to virtual clusters (inter-cluster scheduling) and processor resources assigned to each virtual cluster are used by that cluster to schedule its tasks (intra-cluster scheduling).

4.1. Multiprocessors resource model

The notion of component interface is used to specify the required processor resources to schedule the tasks within the component [21]. A multiprocessor resource model specifies the characteristics of resource provided to a cluster by the multiprocessor platform. As a component interface, a multiprocessor resource model specifies the resource requirement for the component.

A multiprocessor periodic resource (MPR) model denoted by \( \Gamma' = (\Pi, \theta, m') \) specifies that the multiprocessor collectively provides \( \theta \) units of processor resource in every \( \Pi \) time units to a cluster consisting \( m' \) processors. A feasible MPR model has to satisfy \( \theta/\Pi \leq m' \).

The lower bound of amount of resource supply that a resource model \( \Gamma' \) in time interval \( t \) provides is specified by supply bound function \( sbf(t) \). In schedulability conditions, \( sbf \) is used to generate MPR based component interfaces.
There is no technique for scheduling clusters according to their interfaces, hence for scheduling of clusters on a multiprocessor platform each cluster is transformed into periodic tasks, i.e., each cluster of size $m'$ is transformed into $m'$ periodic tasks. The obtained task set is scheduled on the multiprocessor platform using an existing global scheduling protocol, e.g., G-EDF.

5. Implementing FMLP under Multiprocessor Hierarchical Scheduling

Resource sharing under FMLP is performed in different ways for short and long resources (Section 3); for a long resource the job holding the resource executes preemptively and blocked jobs are suspended while for a short resource the job holding the resource executes non-preemptively and blocked jobs perform busy-wait (non-preemptively).

Non-preemptively execution of jobs may not cause any problems in physical clustering when implementing FMLP, as each cluster of size $m'$ receives $m'$ dedicated processors and intra-cluster scheduling remains as a usual global scheduling problem. However, in the virtual clustering, supplied processors for each cluster may differ at different time instants, i.e., a virtual cluster of size $m'$ may receive $k$ processors ($0 \leq k \leq m'$) at any time instant. For example, consider the processor supply for a virtual cluster, $C$, where $m' = 4$ depicted in Figure 1. In this example, processor supply differs from 0 processors (intervals $[t_3, t_4]$), $[t_5, t_6]$, and $[t_7, t_8]$) to maximum processors ($m' = 4$) at time instant $t_4$.

Non-preemptive jobs are located in a FIFO queue. In this case, the non-preemptive jobs are assigned to non-preemptive tasks. This prevents executing preemptive jobs on these processors. A consequence of prioritized non-preemptive tasks is that execution of a non-preemptive job may be postponed (blocked) by higher priority jobs waiting on a higher priority non-preemptive job. This introduces extra blocking times to the lower priority jobs accessing short resources, but the higher priority jobs will access the resources faster (less blocking times).
The length of a non-preemptive queue is at most \( m' \) for a cluster of size \( m' \). At any time instant suppose there are \( q \) jobs in the non-preemptive queue, and \( k \) processors are supplied to the cluster, \( b \) jobs perform busy-wait, and \( h \) highest priority preemptive jobs are ready; an implementation algorithm should include: (1) \( p = \min (q, k) \) jobs at the top of the non-preemptive queue will be assigned to supplied processors, (2) \( z = \min (k - p, b) \) processors are idle, (3) \( \min (k - p - z, h) \) highest ready preemptive jobs are assigned to processors, (4) If any non-preemptive job releases a short resource the first busy-wait job (at the top of related busy-wait queue), if any, should be added to the non-preemptive queue.

In the algorithm (as in the FMLP) each resource group will have a busy-wait queue in which the jobs performing busy-wait will be queued in FIFO order. However each cluster needs a non-preemptive queue in which non-preemptive jobs accessing the short resources are located. Currently, we are working on implementing issues and investigating if there can be more problems regarding implementation for which we will work on finding appropriate solutions.

6. Summary

We have discussed a way of extending hierarchical scheduling framework presented in [20] to support shared resources between tasks within the same cluster. We have used FMLP [7] to synchronize the access of shared resources by tasks. However implementing FMLP under multiprocessor hierarchical scheduling is a big challenge and need considerable effort since some of the assumptions of both the scheduling and the synchronization protocols have to be changed. By extending the hierarchical scheduling with FMLP, two important issues should be considered: the problems of implementing FMLP under multiprocessor hierarchical scheduling, and schedulability analysis. In this paper we have focused on the problems regarding the implementation of FMLP under the hierarchical scheduling framework. We have mentioned the problems and proposed possible solutions to overcome those problems. However, we have not shown how the synchronization will affect the schedulability analysis of the framework and we are currently working on deriving upper bounds for blocking time overheads. In the future, we will evaluate this approach by means of simulation and implementation.

References