Intra-component Resource Sharing on a Virtual Multiprocessor Platform*

Sara Afshar[§], Nima Khalilzad[§], Moris Behnam[§], Reinder J. Bril[§,‡], Thomas Nolte[§]
[§]Mälardalen University, Västerås, Sweden
[‡]Technische Universiteit Eindhoven, Eindhoven, Netherlands
sara.afshar@mdh.se

ABSTRACT

Component-based software development facilitates the development process of large and complex software systems. By the advent of multiprocessors, the independently developed components can be integrated on a multi-core platform to achieve an efficient use of system hardware and a decrease in system power consumption and costs. In this paper, we consider a virtual multiprocessor platform where each component can be dynamically allocated to any set of processors of the platform with a maximum concurrency level. Global-EDF is used for intra-component scheduling. The existing analysis for such systems have assumed that tasks are independent. In this paper, we enable intra-component resource sharing for this platform. We investigate using a spin-based resource sharing protocol with the accompanying analysis that extends the existing analysis for independent tasks. We briefly illustrate and evaluate our initial results with an example.

1. INTRODUCTION

A common approach to accelerate the development process in industrial software systems is to break down large and complex systems into smaller subsystems. Each subsystem is then developed in isolation benefiting from a component-based development approach, which modularizes the development process for such complex software systems. In the integration phase, these subsystems/ components are integrated on a shared platform to construct the whole system. As a consequence of the shift from using single-core processors towards a multi-core architecture, these components which may share resources will eventually co-execute on a shared multi-core platform. Virtual component/cluster-based multiprocessor scheduling is a more general scheduling approach for multiprocessors compared to the traditional approaches such as partitioned, global or physical component/cluster-based scheduling and is less sensitive to task-processor mappings [4]. Under a virtual multiprocessor scheduling, each component can be dynamically allocated to any set of processors where components may share processors.

Several works have been done in the context of virtual multiprocessor scheduling where they have assumed tasks are independent and do not share any resources except the CPUs [7, 4, 8, 6]. In addition, several works have also been done in the context of multiprocessor compositional scheduling with resource sharing [9, 1, 2], however, they have assumed that components are fixed assigned to processors of the platform. For the best of our knowledge no prior work has investigated resource sharing in the context of virtual multiprocessor scheduling. In this work, we investigate enabling resource sharing for virtual multiprocessor platforms using a periodic interface model. For the initial step, we have assumed that each component has its own dedicated set of resources and therefore we enable an intra-component resource handling protocol for system components under a global-EDF (g-EDF) scheduler. Enabling inter-component resource sharing is left as a future step. We use the MPR (Multiprocessor Periodic Resource) model [4] in order to characterize a component. This model, using a maximum parallelism level, specifies the total timing requirements for the component, denoted as the component budget that will make the tasks of the component schedulable. We investigate the overhead on the component budget when enabling intra-component resource sharing by selecting a spin-based approach to handle resource sharing among tasks of a component.

2. SCHEDULING STRUCTURE

Our system consists of m identical unit capacity processors and a set of components where each component is comprised of a set of constrained deadline sporadic tasks. We use the MPR model [4] to present a component. Based on this model a component is presented by a resource model $\mu =< \Pi, \Theta, m' >$ where it specifies that a unit-capacity multiprocessor platform provides a total Θ units of time with concurrency at most m' in every Π time interval where $1 < m' < m$ and $\Theta \leq m'\Pi$.

The system uses a two level hierarchical scheduling scheme consisting of: *(a)* inter-component and *(b)* intra-component scheduling. In the global level (inter-component scheduling) components are scheduled based on a dynamic assignment of the components to m' processors out of m processors in the platform. For intra-component scheduling a global scheduling approach is used within each component. EDF (Earliest Deadline First) scheduling strategy (g-EDF) is used for intra-component scheduling similar to [4]. At each instance of time, g-EDF schedules unfinished jobs that have the m' earliest relative deadlines. Similarly, we also assume that preemption and migration overheads of any job is negligible.

3. RESOURCE SHARING PROTOCOL

In this section we present the resource sharing approach that is used to enable intra-component resource handling. We use a spin-based resource sharing approach similar to MSRP [5] and FMLP (for short resources) [3]. We have assumed a non-nested resource access of tasks of the components. For each resource a FIFO queue is dedicated to enqueue the requests of the tasks. A task is inserted in the related queue when it is blocked on a resource and it is removed from the queue as soon as it accesses the resource. The idea behind the spin-based approach is that a task spins non-preemptively from

†This work is supported by the Swedish Foundation for Strategic Research via the research program PRESS, the Swedish Knowledge Foundation and ARTEMIS Joint Undertaking project EMC2 (grant agreement 621429).

the time when it gets blocked on a resource until it accesses the resource. This rule results in one important property of spin-based approaches which is that there exists at most one pending resource request per core. To maintain this property the following rules are provided to adjust the protocol to be used within each component.

RULE 1. *Whenever a task τ_i requests a resource the priority of the task is boosted to higher than the maximum priority of any task within the component.*

RULE 2. *When a task is spinning or holding a resource and the budget of the component is depleted an overrun happens until the task releases the resource.*

Rule 2 implies that the budget of a component should accommodate for such overrun. Therefore, budget of each component should be inflated with the maximum amount of resource access time of any task within the component and its maximum spinning time to acquire the resource.

A task that is scheduled globally within a component may experience blocking due to other tasks accessing resources. A task may experience two types of blocking incurred by other tasks within the same component: (*i*) blocking due to requesting a resource that is held by another task and (*ii*) blocking incurred by a lower priority task when its priority is boosted due to requesting a resource (Rule 1). The latter blocking case happens when a task is supposed to be scheduled by the g-EDF scheduler, i.e., it is one of the m' highest priority ready tasks, but it is not since a lower priority task is non-preemptive on one of the cores of the component. A task may incur such delay for the maximum resource access time of any lower priority task. This type of blocking can only happen once when the task arrives for the first time since Rule 2 prevents such blocking to happen after the first arrival time.

To calculate the maximum amount of blocking of type (i) for a task, the worst-case locking scenario for resource requests of the task is assumed. Under such worst-case scenario, whenever a task requests a resource, all tasks on the other $m'-1$ processors of the component (if any) have requested the resource earlier. We denote the maximum amount of waiting time for a task τ_i as $spin_i$ and $\mathcal{RS}_i$ as the set of resources used by a task τ_i. Further, we denote $\mathcal{L}_{q,i}$ as the summation of $m'-1$ maximum largest access time for a resource R_q from each of the tasks within the component other than τ_i. Based on these assumptions, $spin_i$ is calculated as follows.

$$spin_i = \sum_{\forall q: R_q \in \mathcal{RS}_i} \mathcal{L}_{q,i}. \quad (1)$$

According to the spin-based approach, a task is busy-waiting when it is blocked on a resource. Hence, typically the spin-based approaches consider such blocking delay as part of the task execution time. Thus, to account for such delay the execution times of the tasks are inflated by such blocking, i.e., for any τ_i that uses a resource $C'_i = C_i + spin_i$.

The blocking type (ii) does not need to be incorporated in the analysis since it is incurred at most once to a task and has already been considered in the demand bound function as part of the execution time of a lower priority task. According to the schedulability test of a component presented in [4], the execution times of all tasks are considered in the schedulability measurement window of a task. Therefore, the blocking time type (ii), which accounts for the maximum critical section of any lower priority task, has been considered in this analysis and there is no need for reconsideration.

4. RESULTS AND CONCLUDING NOTES

We have investigated the overhead of budget increase by enabling intra-component resource sharing which we show with a simple example here. The component schedulability condition is performed according to the analysis presented in [4] that is extended in this paper by execution time inflation (1) due to resource sharing as presented in Section 3.

Example. The following example shows that the total budget that is required to make the tasks of the component schedulable is increased significantly when resource sharing is enabled within tasks of a component. In this example the component consists of five tasks. The execution time and period of a task τ_i is shown by C_i and T_i, respectively with implicit deadlines. We have $\forall i\ C_i = 3000$ and $T_i = 10000$. Therefore, the task set utilization is equal to 1.5. The resource access time of a task τ_i for a resource R_q is denoted by $Cs_{i,q}$. We assumed two shared resources (i.e. R_1, R_2). Also, we assumed that each task accesses both resources once with a resource access time equal to 100 (i.e. $\forall i,q \quad Cs_{i,q} = 100$). This means that each resource access time of a task is 3.3% of its execution times. The period of the component (Π) is 5000. First we ignored the shared resources and we calculated the component budget (Θ). In this case, the total required budget without resource sharing is 9488 with a maximum parallelism level of 2 cores. Therefore, the interface utilization ($\frac{\Theta}{\Pi}$) without considering the shared resources is approximately 1.89. If resource sharing is enabled, then the total required budget is increased to 19438 with maximum parallelism equal to 4, resulting in the interface utilization of approximately 3.88. This necessitates a further exploration of the presented resource sharing protocol in order to remove the current pessimism in the analysis which is left as a future activity.

5. REFERENCES

[1] S. Afshar, M. Behnam, and T. Nolte. Integrating independently developed real-time applications on a shared multi-core architecture. In *CRTS'12*.

[2] A. Biondi, G. Buttazzo, and M. Bertogna. Supporting component-based development in partitioned multiprocessor real-time systems. In *ECRTS'15*.

[3] A. Block, H. Leontyev, B. Brandenburg, and J. Anderson. A flexible real-time locking protocol for multiprocessors. In *RTCSA'07*.

[4] A. Easwaran, I. Shin, and I. Lee. Optimal virtual cluster-based multiprocessor scheduling. *Real-Time Systems*, 43(1):25–59, 2009.

[5] 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 *RTAS'03*.

[6] N. Khalilzad, M. Behnam, and T. Nolte. On component-based software development for multiprocessor real-time systems. In *RTCSA'15*.

[7] H. Leontyev and J. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. In *ECRTS'08*.

[8] G. Lipari and E. Bini. A framework for hierarchical scheduling on multiprocessors: From application requirements to run-time allocation. In *RTSS'10*.

[9] F. Nemati, M. Behnam, and T. Nolte. Independently-developed real-time systems on multi-cores with shared resources. In *ECRTS'11*.