Multiprocessor DSP Scheduling in System-on-a-chip Architectures

Paolo Gai, Luca Abeni
RETIS Lab
Scuola Superiore S. Anna, Pisa
{pj,luca}@gandalf.sssup.it

Giorgio Buttazzo
University of Pavia (Italy)
INFM - Pavia research unit
buttazzo@unipv.it

Abstract

Next generation embedded systems will demand applications with increasing complexity, that a standard uniprocessor microcontroller architecture will likely be unsuited to support. A possible solution to cope with embedded applications with high computational requirements is to adopt multiple-processor-on-a-chip architectures. This paper discusses the problem of multiprocessor scheduling for asymmetric architectures composed by a general purpose CPU and a DSP. The challenging issue addressed in this work is to verify whether the use of a dedicated processor can effectively enhance the performance of an embedded system still maintaining some kind of real-time guarantee. In particular, we provide a method for increasing the schedulability bound in the considered architecture, allowing a more efficient use of the computational resources.

1 Introduction

The rapid development of the embedded system market is demanding applications with increasing complexity in terms of functionality to be supported, as well as timing constraints to be met. For example, in the context of automotive systems, a standard uniprocessor microcontroller architecture will not be able to provide the computing power required by future applications, even considering the IC technology advances. The same considerations will also apply in the context of embedded multimedia systems, where the power required to handle audio and video streams increases as the technology provides fast networks interfaces capable of managing data flows with a huge bit-rate.

A greater computational power in real-time control and multimedia systems can be achieved in two possible ways: increasing the processor speed or increasing the parallelism of the architecture. The first solution requires the use of caching or deep pipelining, which suffer from serious drawbacks in the context of real-time embedded systems: caching makes very hard to estimate the worst-case execution times of programs (needed to perform an off-line guarantee of real-time constraints), whereas deep pipelining would not be very effective because of the large number of pipeline flushes caused by reactions to asynchronous events. In addition, increasing the processor speed would cause a higher power consumption of the whole architecture, which is often not affordable for some mobile applications. Parallelism at the instruction level (VLIW architectures) requires large silicon areas and drastically increases code size. Therefore, the best option for many future embedded applications demanding high computational power seems to rely on the adoption of multiple-processor-on-a-chip architectures.

This conclusion is also supported by some hardware architectures that will be produced in the near future. The Janus system, developed by ST Microelectronics in cooperation with Parades [6], is an example of a dual-processor platform for power train applications, featuring two 32-bit ARM processors connected by a crossbar switch to four memory banks and two peripheral buses for I/O processing.

Although Janus implements a symmetric multiprocessor, some other architectures promise the availability of asymmetric multiprocessors composed by a RISC processor (or a microcontroller) and one or more DSPs [11, 7]. For example, the Texas Instruments SMJ320C80 is a single-chip MIMD parallel processor that consists of a 32-bit RISC master processor, four 32-bit parallel DSPs, a transfer controller, and a video controller. All the processors are tightly coupled through an on-chip crossbar switch that provides access to a shared on-chip RAM.

These architectures are expressly mentioned because multimedia streams can be efficiently handled using dedicated units for signal processing, reducing the computational power needed on the main CPU. Moreover, DSP architectures are designed to have predictable execution times, so they offer a viable alternative to the implementation of fast general purpose multiprocessors.

This paper discusses the problem of multiprocessor scheduling for asymmetric architectures composed by a general purpose CPU and a DSP. The challenging issue addressed in this work is to verify whether the use of a dedicated processor can effectively enhance the performance of an embedded system still maintaining some kind of real-time guarantee.

In our opinion, this issue did not receive sufficient attention in the real-time literature. Although many results have been derived for multiprocessor and distributed architectures, very few of them considered the peculiarities of such an architectural scenario. In particular, a DSP is usually designed to execute algorithms on a set of data without interruption. Hence, the *natural* way of scheduling a DSP is typically non-preemptive, whereas the CPU schedules both the application tasks and the non-preemptive tasks running on the DSP. In practice, the DSP can be used as a DSP accelerator responding to requests of the master (or main) CPU [4].

The problem of DSP scheduling in asymmetric multiprocessor architectures can be viewed as a special case of scheduling with shared resources in multiprocessor distributed systems. In [10, 9], Rajkumar addressed this problem, providing two algorithms called MPCP and DPCP, that allow resource sharing in generic multiprocessors. Although DPCP can be applied to this particular case, the guarantee test provided in [10, 9] is too pessimistic, since it relies on a very generic scenario that does not take in account the characteristics of the considered architecture.

An interesting approach for coping with tasks requiring multiple processing resources is proposed in [12], where the *Co-Scheduling* approach is presented. Co-Scheduling can be used when a task is composed by multiple phases and requires a different resource in each phase. The basic idea of co-scheduling is to divide each job into chunks, associating suitable deadlines to each chunk in order to meet the job deadline (that is, the deadline of the last chunk). In the original paper, the CPU and the disk were considered as the co-scheduled resources, but we believe that this approach could be also applied to the DSP.

Our paper provides the following contributions:

- It presents an outlook of the problem giving some examples with counterintuitive behavior.
- It proposes an approach that allows to increase the schedulability bound in the considered architecture.

The paper is organized as follows. Section 2 introduces the system model and the architecture of the embedded system. Section 3 presents the problem and provides some examples of non-intuitive scheduling issues. Section 4 describes our method for increasing the schedulability bound. Section 5 illustrates some simulation results, and Section 6 states our conclusions and future work.

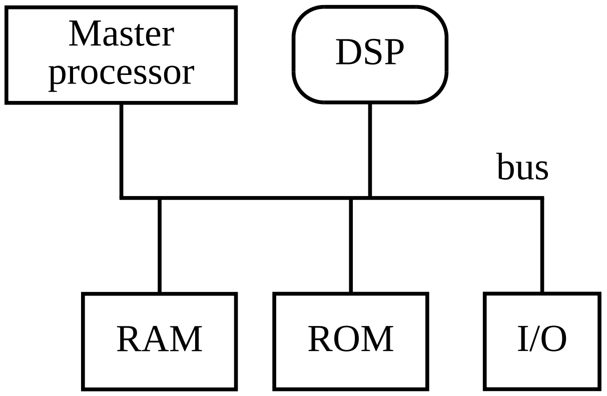

Figure 1. Block diagram of the system architecture.

2 System model

In this section we shortly describe the reference architecture we will use in the following sections.

We consider an abstract architecture composed by a general purpose CPU and a specialized CPU. The two computational units share a common bus and can freely use some RAM memory and some ROM (Flash) memory, that we suppose is built in the same chip. Other peripherals can be directly controlled by the general purpose CPU. These assumptions are not far from reality since a system on a chip architecture, like Janus [6], can be modeled in a similar way.

At the present stage of the analysis, we assume that the general purpose CPU and the specialized CPU communicate with negligible overhead. This assumption is justified by the fact that the two processors are supposed to be built on the same chip, and both processors can share the full (or part of the) memory address space of the architecture. However, possible communication overheads can be accounted to the task that invokes the service on the specialized CPU. In the following, the general purpose CPU will be identified as the *master processor*, whereas the specialized CPU as the *DSP*. A block diagram of the considered architecture is depicted in Figure 1.

We assume that all the jobs executing on a DSP are invoked using a remote procedure call (RPC) paradigm and are executed in a non-preemptive fashion. Such RPCs will be called *DSP activities*. To simplify the analysis we also assume that on the master processor the DSP activities are scheduled one at a time. Hence, it is a responsibility of the real-time kernel on the master processor to avoid that a task issues a DSP request while the DSP is active.

The real-time task model considered in this work is illustrated in Figure 2: each task τ_i is a stream of instances, or jobs; each job $J_{i,j}$ arrives at time $r_{i,j}$, executes for C_i units of time on the master CPU, and may

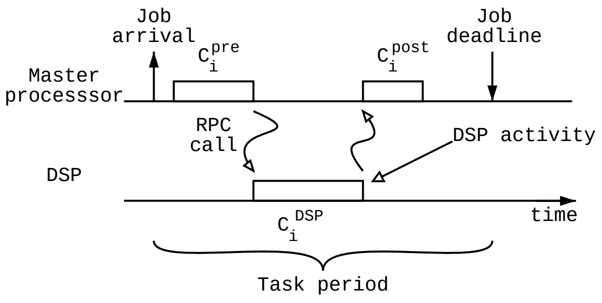

Figure 2. Structure of a DSP task.

require a DSP activity for C_i^{DSP} units of time. We assume that each job performs at most one DSP request, after C_i^{pre} units of time, and then executes for other C_i^{post} units, such that $C_i^{pre} + C_i^{post} = C_i$. Job arrivals can be periodic (if $r_{i,j} - r_{i,j-1} = T_i$, being T_i the task's period) or sporadic (if $r_{i,j} - r_{i,j-1} \geq MIT_i$, MIT_i being the minimum interarrival time). If a task requires a DSP activity it is denoted as a *DSP task*, otherwise it is denoted as a *regular task*. Note that a regular task is equivalent to a DSP task with $C_i^{DSP} = 0$. Whenever a fixed priority scheduling algorithm is used, P_i denotes the priority of task τ_i.

3 Problem definition

To present the problem that may occur when dealing with the task model introduced in Figure 2, let us make some considerations about the structure of a DSP task.

When executing a DSP task, a *hole* within each job is generated in the schedule of the master processor. Such holes are created because the DSP task executes a DSP activity on another processor. The main idea is to exploit these holes to schedule some other tasks on the master processor in order to improve the schedulability bound of the system.

Suppose, for example, to have a task τ_1 with a period $T_1 = 4$ units of time, $C_1^{pre} = C_1^{post} = 1$, and $C_1^{DSP} = 2$. Note that, although τ_1 uses the master processor only for fifty percent of the time, it must start *exactly* when it arrives, otherwise it will miss its deadline. This constraint is independent from the scheduling algorithm used in the system. Hence, if a regular task τ_2 (which does not use the DSP) with period $T_2 = 3$ and computation time $C_2 = 1$ is added to the system, both the Rate Monotonic (RM) algorithm and the Earliest Deadline First (EDF) algorithm fail to generate a feasible schedule, because if tasks start at the same time, τ_2 will always have precedence to τ_1. This situation is shown in Figure 3.

However, the task set can be schedulable using a fixed priority assignment by simply assigning $P_1 > P_2$. Figure 4 shows the feasible schedule generated using such a priority assignment.

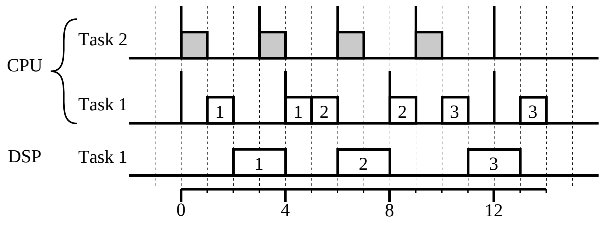

Figure 3. A task set that cannot be feasibly scheduled by RM and EDF (jobs of task τ_1 are numbered to facilitate interpretation): task τ_1 misses all its deadlines.

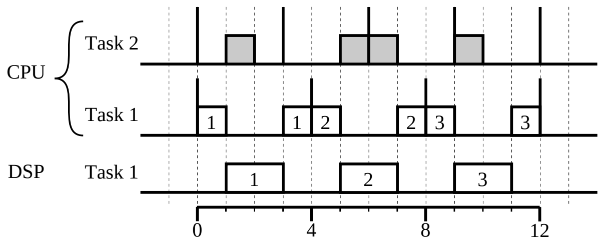

Figure 4. A feasible schedule achieved by a different priority assignment ($P_1 > P_2$).

Note that the system model we introduced can be viewed as a particular case of a more general model proposed by Rajkumar in [10, 9], where the Distributed Priority Ceiling Protocol (DPCP) is used to access shared resources on a distributed system. In particular, the DPCP protocol can be used for all the tasks allocated on the main CPU and the DSP activities can be considered as global critical sections executed on the DSP (which acts as a synchronization processor). According to the DPCP approach, the schedulability of the task set is guaranteed by the following test:

$$\forall i = 1, \ldots, n \quad \sum_{P_j > P_i} \frac{C_j + C_j^{DSP}}{T_j} + \frac{C_i + C_i^{DSP} + B_i}{T_i} \leq U_{lub}(i),$$

where $U_{lub}(i) = i(2^{1/i} - 1)$, and B_i is a blocking factor computed as follows:

$$B_i = \begin{cases} \max\limits_{P_j < P_i} \{C_j^{DSP}\} + & \\ \quad \sum_{P_j > P_i} \left\lceil \frac{T_i}{T_j} \right\rceil C_j^{DSP} & \text{for a DSP task} \\ 0 & \text{for a regular task} \end{cases} \tag{1}$$

The major problem of this approach is that the execution time of the DSP activities (C_i^{DSP}) is considered as part of the computation time of every task, including regular tasks. Such a pessimistic computation, although correct, drastically reduces the schedulability of the system. Indeed, in the papers [10, 9], *the authors claim that the C_i^{DSP} factor can also be removed from the computation times, but no proof is provided for that claim*.

In Section 4, we will provide a formal analysis of the DSP tasks and we will propose a more efficient method for scheduling DSP tasks and DSP activities. The method will be compared with the DPCP in Section 5.

The problem of exploiting the holes to improve the schedulability of the system can be posed in terms of fixed or dynamic priorities. In this paper we propose a solution for the case of fixed priorities, and we give some hint for dealing with dynamic priorities (this case will be investigated with more detail in a future work).

4 Our scheduling approach

As observed in the previous section, the idle time created in the master processor by executing some activities on the DSP can be used for increasing the system schedulability to guarantee more real-time tasks.

In this section, we show how to schedule tasks on the master processor and how to perform an admission test which exploits the time left by the activities executing on the DSP. The basic idea is to re-arrange the scheduling and guarantee algorithms in order to account the DSP time C_i^{DSP} only to tasks that use the DSP (without influencing the schedule and the guarantee of the regular tasks). This can be achieved by modelling the DSP activity as a blocking time: when a DSP task τ_i requests a DSP activity, it blocks for a time B_i waiting for its completion (since we are using an RPC protocol). Moreover, the scheduler has to be modified in such a way that the B_i factor affects only τ_i in the schedulability analysis.

In the next subsection, we show how to modify the scheduler and how to compute the blocking factors when tasks are scheduled using a fixed priority assignment. The case of dynamic priorities is more difficult to analyze and, at present, we just discuss some issues that may help to address the problem.

4.1 Enhancing schedulability under fixed priorities

As already said in Section 2, to simplify the analysis we assume that DSP requests are scheduled by the master processor one at a time, so that no DSP activity is issued while the DSP is active. This can be achieved by enqueuing regular tasks and DSP tasks in two separate queues that are ordered by priority as shown in Figure 5. When the DSP is idle, the scheduler always selects the task with the highest priority between those at the head of the two queues. When the DSP is active, the scheduler selects the task with the highest priority from the head of the regular queue only.

In this way, a task using the DSP blocks all the other tasks requiring the DSP, but not the regular tasks, which can freely execute on the master processor in the holes created by DSP activities.

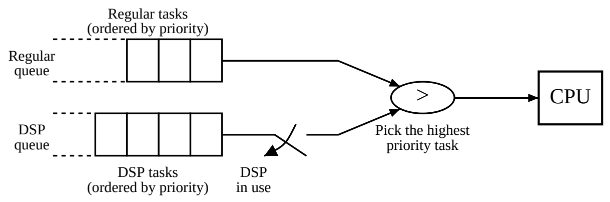

Figure 5. Our scheduling approach. When the DSP is active, the scheduler selects tasks from the regular queue only.

A similar result can be achieved by implementing each DSP request through a blocking primitive which suspends the calling task in a queue. A waiting task is then awakened by the DSP as the activity has been completed. This solution has also the advantage of allowing other DSP tasks to execute on the master processor while a DSP request is being served.

Note that this approach is different from using an inheritance-based protocol [13], which would prevent a regular task with medium priority to execute while a high priority DSP task is blocked on the DSP resource. On the contrary, using our approach, the DSP execution time contributes to the blocking term of DSP tasks only.

A DSP task, however, can also be delayed by other tasks which may hold the DSP. In particular, a DSP task τ_i can be delayed by a single lower priority job which is already using the DSP, and by those higher priority jobs that that may interfere with τ_i before it is scheduled. Hence, the blocking factor B_i of a DSP task can be computed as the sum of three terms:

$$B_i = C_i^{DSP} + B_i^{lp} + B_i^{hp}, \tag{2}$$

where B_i^{lp} denotes the blocking caused by the (single) lower priority task and B_i^{lp} denotes the blocking due to the interference of higher priority tasks. Figure 6 illustrates an example showing how a task τ_3 can be blocked by a task τ_4, with lower priority, and by two tasks, τ_1 and τ_2, having higher priority. As also done in [10, 9], the two blocking terms can be computed as follows:

$$B_i^{lp} = \max_{P_j < P_i} \{C_j^{DSP}\}$$

$$B_i^{hp} = \sum_{P_j > P_i} \left\lceil \frac{T_i}{T_j} \right\rceil C_j^{DSP}.$$

Therefore, an upper bound for the blocking time B_i experienced by task τ_i is given by

$$B_i = \begin{cases} C_i^{DSP} + \max\limits_{P_j < P_i} \{C_j^{DSP}\} + & \\ \quad \sum_{P_j > P_i} \left\lceil \frac{T_i}{T_j} \right\rceil C_j^{DSP} & \text{for a DSP task} \\ 0 & \text{for a regular task} \end{cases} \tag{3}$$

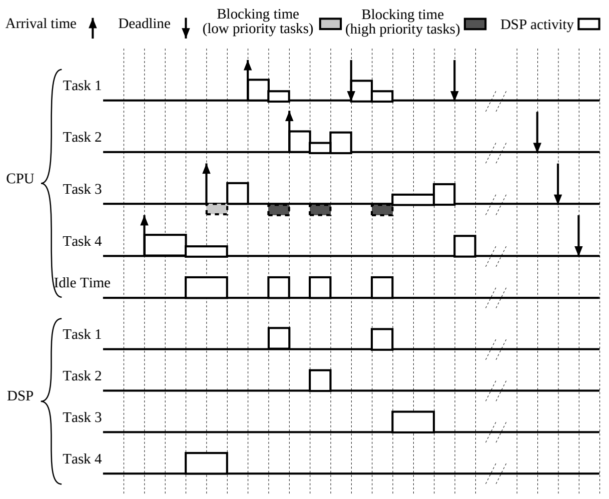

Figure 6. Example of scenario where task τ_3 is blocked by some high priority (τ_1 and τ_2) and low priority (τ_4) tasks.

This value can be used in the classical admission test for fixed priority systems [13], that is:

$$\forall i = 1, \ldots, n \quad \sum_{P_j \geq P_i} \frac{C_j}{T_j} + \frac{B_i}{T_i} \leq U_{lub}(i), \qquad (4)$$

where $U_{lub}(i) = i(2^{1/i} - 1)$.

It is worth observing that to enhance schedulability and accept more tasks in the system, the admission test can be performed by using the hyperbolic bound [5] or the response time analysis [8, 1]. For large task sets where the admission test has to be performed on line, then the hyperbolic bound is more suited, having an $O(n)$ complexity, whereas the response time analysis has a pseudo-polynomial complexity.

Using the hyperbolic bound [5] the admission test becomes:

$$\forall i = 1, \ldots, n \quad \prod_{P_j > P_i} \left(\frac{C_j}{T_j} + 1 \right) \left(\frac{C_i + B_i}{T_i} + 1 \right) \leq 2.$$

In Section 5, we also present some simulation experiments aimed at estimating the advantage of using different admission tests.

4.2 Extensions

In this section we consider the possibility of extending the analysis of the blocking times to more general scenarios. We first discuss the case of removing the assumption of having one DSP request at a time, and then we address the problem of using a dynamic priority assignment.

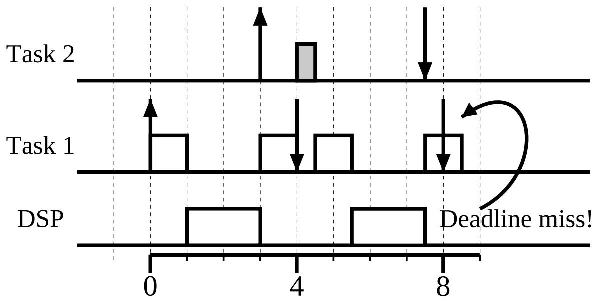

Figure 7. EDF does not work always.

4.2.1 Allowing interleaving DSP requests

If the constraint of having only one DSP activity at a time is removed and no special protocol is used for accessing the DSP, the blocking time B_i for task τ_i is equal to the response time of the DSP request. In this case, such a response time must be explicitly computed by performing a finishing time analysis on the DSP schedule. Since we are using an RPC protocol, the DSP scheduling is clearly non-preemptive; hence, *if the requests for DSP activities are ordered according to the RM priority assignment, then the problem of computing the blocking time B_i is equivalent to the problem of finding the response time of a task τ_i with computation time C_i^{DSP} and period T_i when a nonpreeemptive RM scheduler is used and some release jitter is present.*

The exact finishing time of a task under a fixed priority preemptable scheduler can be computed as shown in [1], the nonpreemptability of the scheduler can be accounted by adding a blocking term equal to $\max_{P_j < P_i}\{C_j^{DSP}\}$ to each blocking time B_i, whereas the release jitter can be accounted as shown in [14].

4.2.2 Dealing with dynamic priorities

Extending the previous analysis to dynamic priorities unfortunately is not trivial. In fact, the admission test presented in Equation (4) can be applied to dynamic priorities only if the blocking terms are due to resource sharing [3, 2]. In our case the blocking times are also caused by the synchronization between the master processor and the DSP.

To better clarify the problem that can arise in this case, let us consider a task set $\Gamma = \{\tau_1, \tau_2\}$, with $C_1^{pre} = C_1^{post} = 1$, $C_1^{DSP} = 2$, $T_1 = 4$, $C_2 = \epsilon$, $C_2^{DSP} = 0$ (τ_2 is not accessing the DSP), and $T_2 = 4+\epsilon$. According to equation (4), the task set should be schedulable: in fact $B_1 = 2$ and $B_2 = 0$, hence for task τ_1 we have $\frac{1}{2} + \frac{1}{2} \leq 1$, and for task τ_2 we have $\frac{1}{2} + \frac{\epsilon}{4+\epsilon} \leq 1$, so the admission test is passed. Unfortunately, as can be seen in Figure 7, if τ_2 arrives at time $t = 3$, it executes before the second instance of τ_1. As a result, $J_{1,2}$ is not scheduled as soon as it is released and it misses its deadline.

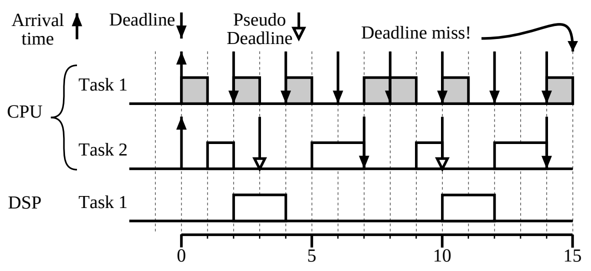

Figure 8. Also EDF with modified deadlines does not work always.

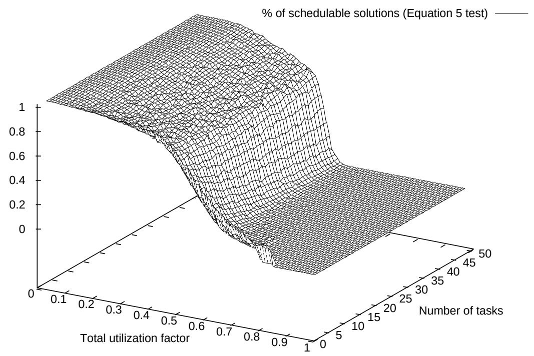

Figure 9. Schedulability results of our approach when varying the total utilization factor and the number of tasks in the task set (using Equation (4)).

To use dynamic priorities in this context, one could think to split each job in more chunks and assign each chunk a different relative deadline (for example, a relative deadline of $T_i - C_i^{DSP} - C_i^{post}$ could be given to the chunk executed before the DSP activity).

However, we note that, although this assignment works properly in the scenario of Figure 4, it does not work in general. For example consider two tasks, τ_1 and τ_2. τ_1 does not use the DSP and has $T_1 = 2$ and $C_1 = 1$. τ_2 has $T_2 = 7$, $C_2 = 5$, $C_2^{DSP} = 2$ and $C_2^{post} = 2$. Figure 8 shows that this task set is not schedulable with the method suggested above.

A complete formal analysis under dynamic priority assignments has still to be derived and we plan to investigate it as a future work.

5 Simulation results

The performance of the scheduling algorithm presented in Section 4 has been evaluated by simulation on a large number of task sets (more than 15 million experiments). For each task set we computed the blocking time of each task and we checked the schedulability using both our approach and the DPCP protocol.

Task sets were generated using random parameters with uniform distribution with the following characteristics:

- The number of tasks was chosen as a random variable from 2 to 50.
- Task periods were generated from 10 to 1000.
- Task worst-case execution times ($C'_i = C_i + C_i^{DSP}$) were chosen in such a way that $\sum_i \frac{C'_i}{T_i}$ varied from 0.01 to 0.99.
- DSP tasks were generated to be 80% (in the average) of the total number of tasks.
- C_i^{DSP} was generated to be a random variable with uniform distribution in the range of 10% to 80% of the C'_i.

In a first experiment we compared the performance of our method against the DPCP approach in terms of average schedulability, using the admission test given by Equation (4). Figures 9 and 10 show the percentage of schedulable task sets, for our approach and for the DPCP method, as a function of the number of tasks and the total utilization factor ($\sum_i \frac{C'_i}{T_i}$). As clear from the graphs, both methods have a performance degradation as the total utilization factor increases, whereas they are quite insensitive to the number of tasks in the set.

To better evaluate the enhancement achieved with our approach, Figure 11 reports the difference between the two previous graphs. We note that the advantage of our approach with respect to DPCP is more sensitive for task sets with total utilization in the range from 0.3 to 0.6.

A second experiment has been carried out to evaluate the improvement that can be achieved using our method with the hyperbolic bound in place of Equation (4). Figure 12 shows that for large task sets with utilizations around 50% the hyperbolic bound improves the acceptance rate up to 30%.

A third experiment has been performed to compare the two approaches using the response time analysis. Figure 13 shows the performance differences between the two methods. We note that the surface has the same shape as the one in Figure 11, presenting a peak translated around 0.7 in the utilization axis. Hence, our approach basically outperforms DPCP when the utilization factor is near to the RM schedulability bound. Figure 14 reports the performance of the two approaches and their difference as a function of the utilization factor for task sets composed by 30 tasks.

Within the same experiment performed with the response time test, we evaluated the influence of the DSP usage ($\sum_i C_i^{DSP}$) on the schedulability results. Figure

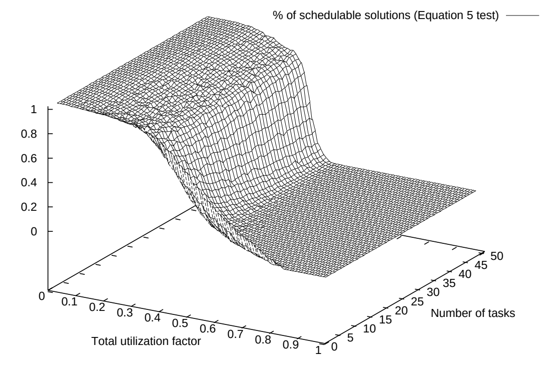

Figure 10. Schedulability results of DPCP when varying the total utilization factor and the number of tasks in the task set (using Equation (4)).

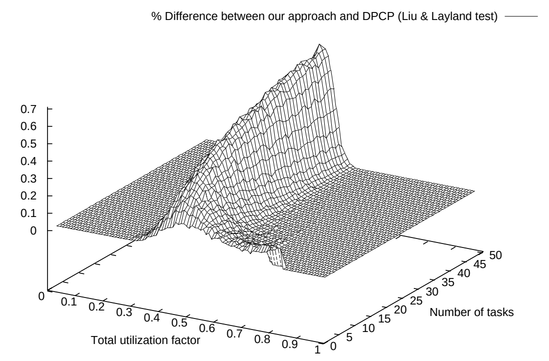

Figure 11. Difference between the two approaches (using Equation (4)).

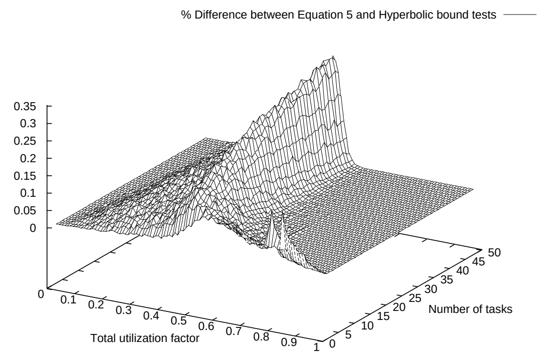

Figure 12. Improvement achieved using the Hyperbolic Bound.

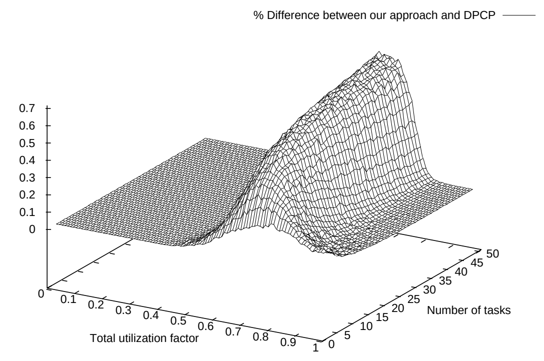

Figure 13. Difference between the two approaches (using response time analysis).

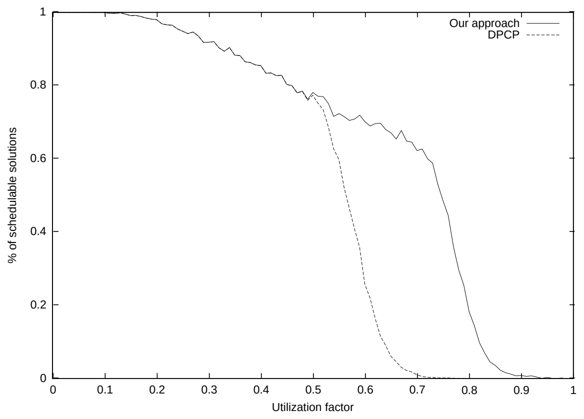

Figure 14. Performance of the two approaches and their difference as a function of the utilization factor for task sets composed by 30 tasks.

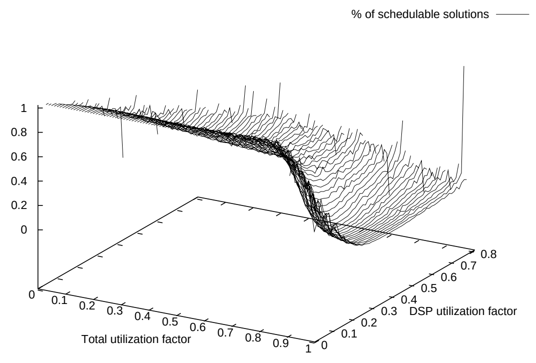

Figure 15. Difference in the percentage of scheduled tasks set between our approach and DPCP when considering the influence of DSP utilization.

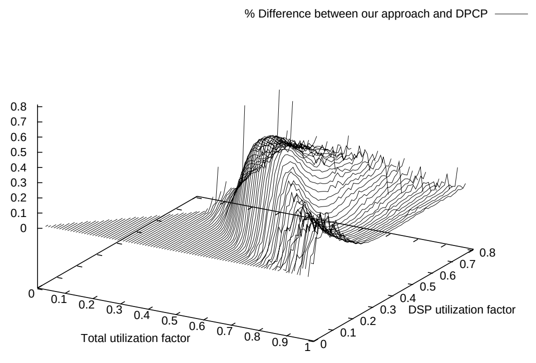

Figure 16. Influence of DSP utilization on the schedulability.

15 shows that the higher the DSP usage, the lower the average acceptance rate.

Finally, the difference of schedulability percentage between our approach and the DPCP is reported in Figure 16, which shows that our algorithm outperforms DPCP for task sets with high utilization and high DSP usage.

6 Conclusions

In this paper we addressed the problem of scheduling a set of tasks in an asymmetric multiprocessor consisting of a general purpose CPU and a DSP. Although this kind of architecture can be considered as a special case of a multiprocessor system, its peculiarity allows to perform more specific analysis which is less pessimistic than the one typically used in distributed systems with shared resources.

A method for computing blocking times has been proposed, which has been showed to be more effective than the classical method adopted in the Distributed Priority Ceiling Protocol [10, 9]. Extensive simulations have shown that our approach always outperforms the DPCP protocol and achieves a significant improvement for large task sets with high processor utilization.

As a future work, we plan to extend the analysis also for dynamic priority assignments.

References

[1] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(8):284–292, Sep 1993.

[2] T. Baker. Stack-based scheduling of real-time processes. *Journal of Real-Time Systems*, 3, 1991.

[3] T. P. Baker. A stack-based allocation policy for realtime processes. In *IEEE Real-Time Systems Symposium*, december 1990.

[4] R. Baumgartl and H. Hartig. Dsps as flexible multimedia accelerators. In *Second European DSP Education and Research Conference (EDRC'98)*, Paris, September 1998.

[5] E. Bini, G. Buttazzo, and G. Buttazzo. A hyperbolic bound for the rate monotonic algorithm. In *Proceedings of the 13th IEEE Euromicro Conference on Real-Time Systems*, June 2001.

[6] A. Ferrari, S. Garue, M. Peri, S. Pezzini, L.Valsecchi, F. Andretta, and W. Nesci. The design and implementation of a dual-core platform for power-train systems. In *Convergence 2000*, Detroit (MI), USA, October 2000.

[7] T. Instruments. *Military Semiconductor Products Fact Sheet SM320C80 / SMJ320C80 / 5962-9679101 SGYV006C*, August 2000.

[8] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1989.

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

[10] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *In Proceedings of the 1988 Real Time System Symposium*, 1988.

[11] K. K. P. Research. Increasing functionality in set-top boxes. In *Proceedings of IIC-Korea, Seoul*, 2001.

[12] S. Saewong and R. R. Rajkumar. Cooperative scheduling of multiple resources. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1999.

[13] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE transaction on computers*, 39(9), September 1990.

[14] K. Tindell. An extendible approach for analysing fixed priority hard real-time tasks. Technical Report YCS 189, Department of Computer Science, University of York, December 1992.