SCHEDULABILITY ANALYSIS AND DESIGN OF REAL-TIME EMBEDDED SYSTEMS WITH PARTITIONS

David Doose, Zoubir Mammeri
IRIT - Paul Sabatier University Toulouse, France
{doose, mammeri}@irit.fr

Abstract: Resource partitioning is used to run several independent applications on the same hardware while avoiding error propagation. However, classical methods of validation and design are not adapted to this technique, so new methods have to be elaborated. In this paper, we define four utilization bounds, which give sufficient conditions to guarantee an execution sequence without timing faults as long as the utilization rate of the system remains under the bound. They can of course be used to validate a system with partitions, but the fact that they are based on a partial knowledge of the system allows to use them during system design. This latter point is interesting since we can thus validate a system whose parameters are not yet completely defined, which can greatly reduce the cost by avoiding many backtracks in development cycle.

1. INTRODUCTION

The use of computers in industrial processes that involve critical application [1] induces changes in the design of systems. To run several applications simultaneously on the same systems, one can choose a distributed or centralized architecture. The first solution avoids propagation of errors between independent applications [2] but its cost increases with the number of applications (including the costs of space, a major issue in avionics for instance [3, 4]). Besides it is possible nowadays to centralize computations owing to computer performances, which have increased and are still increasing faster than embedded applications complexity.

However, costs reduction must not cut down on safety. That is why we need to develop new methods to design, validate and safely execute independent applications in a centralized environment. Many researches have been conduced in this area and many kinds of solutions were proposed [1, 5], among which partitioning [6–8].

Partitioning is an interesting solution, which allows to run several applications on the same hardware while keeping their independence and the safety requirements. The ARINC-653 [9] standard is a typical example of partitioning, that uses both strict resource partitioning and time partitioning. This time partitioning is based upon a multi-level or hierarchical scheduler [10–12] composed of a cyclic scheduler for the partitions, and a fixed-priority scheduler for the tasks inside a partition. The layer organization offers modularity and re-usability [13], which reduces the cost of the systems. Many recent real-time systems [14, 15] use it, and moreover classical methods of validation [16] can be used without any particular problem for this kind of systems, which makes the partitioning solution an almost perfect one.

However, the use of a fixed-priority scheduler is a problem regarding performance, since the utilization bounds are quite low [17, 18]. A dynamic priority scheduler (such as EDF) would be useful to increase the scheduling capacity of partitioned systems. And thus the cost of the system.

The cost of validation also has to be considered. Indeed, the design of hard real-time systems differs from classical design processes in that the hardware is to be taken into consideration very early.

The validation step requires the knowledge of both hardware and software. In fact, it is particularly difficult to know the time parameters of tasks, including their worst case execution time. Indeed, computation methods of WCET are based on the knowledge of the hardware and on the software implementation. This second point is the most problematic since implementation and testing are expensive, and it may have to be re-done several times if the validation fails.

In this article, we propose techniques to limit this risk. We try to give an explicit representation of the system and its scheduling capacity usable even during the design step. Indeed, some time parameters of tasks can be known very early, such as for instance the period which often depends on the properties of external systems. The partitioning choice is also often made before the software implementation step.

Owing to this partial knowledge, we will determine processor utilization bounds which, if respected, will guarantee an execution without timing faults. This technique allows to conduct in parallel tasks software implementation and validation. The advantage is that the validation using this technique can detect some errors before the complete realization of the system, which can drastically cut down the design costs of hard real-time systems.

This article is structured as follows: the next section describes the system properties and presents some notations. Section 3 introduces four utilization bounds, and describes their properties and how to use them for validation. In section 4, we present methods, based on bounds, to allow the designer to find partitions parameters. Finally, we conclude our work in section 5.

2. BACKGROUND

The concept of partitioning induces that the processor allocation is not linear. This makes unusable the model of task commonly used [17]. That is why we will use *the generalized multiframe task* model [19–23].

Task notion

The concept of task uses in this article is relatively simple, but it makes possible to model and study many real-time systems [24, 16]. A task τ_i is defined as a couple (c_i, p_i), where c_i is the worst case execution time and p_i is the period of the task.

The study of partitioned systems highlights the difference between the concept of application and the concept of task. Indeed, these two concepts are often wrongly confused. In the case of partitioned systems this is not allowed because the concept of partition is used to separate the applications and not the tasks. Thus, the various tasks of a partition are considered as an application or a group of tasks ($\mathcal{G}$) to which a complete processor is assigned. In this article, we consider that this collection is partially ordered according to the task index set, thus the period of the first task is the shortest one ($\forall_{i=1...n}\ p_1 \leq p_i$).

Partitioning

We now introduce the concept of partitioning. Even if we will only study here the partitioning of the processor, we can notice that the following definition of partition can be extended to any kind of resource.

Definition 1 (Partition). *A partition* Π *is a tuple* (Γ, P)*, where* Γ *is an array of* N *pairs* $\{(S_1, E_1), \ldots, (S_N, E_N)\}$ *that satisfies* $(0 \leq S_1 < E_1 < S_2 < E_2 < \ldots < S_N < E_N \leq P)$ *for some* $N \geq 1$*, and* P *is the partition period. The processor is available to a task group executing on this partition only during time intervals* $[S_i + j \times P, E_i + j \times P], 1 \leq i \leq N, j \geq 0$.

This representation of the partitioning has the advantage of being able to model any static behavior.

Definition 2 (Supply Function). *The supply function* $S(t)$ *of a partition* Π *is the total amount of time that is available for* Π *from time 0 to time* t.

Definition 3 (Demand Bound Function). *Let* τ *be a task, and* t *a positive number. The demand bound function* $dbf(\tau, t)$ *denotes the maximum cumulative execution requirement of the jobs of* τ *that have both arrival times and deadlines within any time interval of duration* t.

Theorem 1 ([21]). *A group* $\mathcal{G}$ *is infeasible on a partition* Π *if and only if* $\sum_{\tau \in \mathcal{G}} dbf(\tau, t) > S(t_0 + t) - S(t_0)$ *for some positive real numbers* t_0 *and* t.

The major interest of this theorem comes from that the complete knowledge of the task parameters is not needed. We will thereafter use this property to introduce new theorems based on a partial knowledge of the system.

Definition 4 (Least Supply Function). *The least supply function* $S^*(t)$ *of a partition* Π *is the minimum of* $(S(t+d) - S(d))$ *where* $t,\ d \geq 0$.

Definition 5 (Critical Partition). *A critical partition of a partition* $\Pi = (\Gamma, P)$ *is* $\Pi^* = (\Gamma^*, P)$ *where* Γ^* *has time pairs corresponding to the steps in* $S^*(t)$ *such that* Π^**'s supply function equals* $S^*(t)$ *in* (0, *P*).

The critical partition is an essential concept because it can express in a non-pessimistic way the worst situation, for the schedulability of a task group, relative to a partitioning (Π).
The following theorem was, up to this article, the best means to study effectively the schedulability of a partitioned real-time system (as far as we know).

Theorem 2 ([21]). *A task group* $\mathcal{G}$ *is infeasible on a partition* Π *if and only if:*
$\sum_{\tau \in \mathcal{G}} dbf(\tau, t) > S^*(t)$ *for some positive real number* t.

3. SCHEDULABILITY AND UTILIZATION BOUNDS

The previously defined method allows to study the schedulability of a real-time partitioned system, without pessimism. However it has some disadvantages. Indeed, since it uses all the system characteristics (parameters of both, tasks and partition), a system modification, as negligible as it may be, implies to restart the process of validation from the beginning. For the same reason, it cannot be used in an interactive design of the system. Moreover, this method does not provide an intuitive idea of the capacity of the system to meet task time-constraints. Finally, towards the aim of adding the system some tasks dynamically, this method is too complex to be used as acceptance test for new tasks.
So there is a need of another kind of solutions. That is why we introduce four utilization bounds to study the schedulability of partitioned systems.

Demand bound function

In the particular situation where a task is defined only by its period and its worst case execution time, the demand bound function is defined as follow:
$\sum_{\tau \in G} dbf(\tau, t) = \sum_{i=1}^{n} \left\lfloor \frac{t}{p_i} \right\rfloor \times c_i$.

Theorem 3. *A task group* $\mathcal{G}$ *is schedulable on a partition* Π *if and only if* $\forall t \in \mathcal{D},\ \sum_{\tau \in G} dbf(\tau, t) \geq S^*(t)$ *with* $\mathcal{D}$ *such as:* $\mathcal{D} = \{k \times p_1, \ldots, k \times p_n\}\ \forall_k$ *such as* $\forall_n\ k \times p_n \leq lcm(p_1, \ldots, p_n, P)$.

The proof is commonplace because the demand bound function grows only at the instants t such that $t \in \mathcal{D}$.

Minimal partition

We introduce the concept of *minimal partition* which allows the designer to study the schedulability of a system in which some parameters are unknown.

Definition 6 (Minimal Partition). *The minimal partition is the partition such that its availability function ($S^{**}(t)$) is lower than or equal to all availability functions of every critical partition with same period (P) and* period total availability *($A = \sum_{i=1}^{N}(E_i - S_i) = S(P)$).*

Thus we can easily deduce the formula of the minimal partition availability function, called *minimal function*:

$$S^{**}(t) = \left\lfloor \frac{t}{P} \right\rfloor \times A + \begin{cases} 0, if\ (t\ modulo\ P) \leq P - A \\ (t\ mod\ P) - P + A,\ else \end{cases} \quad (1)$$

As a consequence, the minimal critical partition is defined such as: $\Pi^{**} = (\{(P - S, P)\}, P)$.

The minimal partition is interesting because we can use it to validate partitioned systems with the following theorem.

Theorem 4. *If a task group $\mathcal{G}$ is schedulable in the minimal partition (Π^{**}), then $\mathcal{G}$ is schedulable in every partition (Π) with the same period (P) and same period total availability (A).*

Utilization bound

Using utilization bounds to study a real-time system is interesting, because it provides a means, to validate and design the system, less complex than the method based on the critical partition. But it also gives to the designer an intuitive idea of the availability of the system to schedule tasks without timing faults. To do so, we need to introduce a new hypothesis.

Hypothesis 1. *For any task group $\mathcal{G}$, the period of the first task must be greater than or equal to the period of the partition: $p_1 \geq P$.*

We can give these four following bounds:

$$\beta = \frac{\left\lfloor \frac{p_1}{P} \right\rfloor \times A}{\left\lceil \frac{p_1}{P} \right\rceil \times P + P - A} \qquad \beta' = \min\left(\frac{S^{**}(t)}{t}\right), \forall t \in \mathcal{D}$$

$$\beta'' = \min\left(\frac{S^{*}(t)}{t}\right), \forall t \in [p_1; \infty[\qquad \beta''' = \min\left(\frac{S^{*}(t)}{t}\right), \forall t \in \mathcal{D}$$

with $\mathcal{D} = \{k \times p_1, \ldots, k \times p_n\}\ \forall_k$ *such as* $\forall_n\ k \times p_n \leq lcm(p_1, \ldots, p_n, P)$.

Bounds properties

Pessimism. One of the major advantages of the utilization bounds is their robustness. Indeed, the bounds being calculated only with parts of parameters of the system, their conclusions remain valid with some changes of the system. This property is interesting because it allows to use the utilization bounds during the design step. The partial knowledge needed to calculate the bounds, implies that they are pessimistic. In fact, the more parameters are needed to determine a bound, the less pessimistic it is. In order to quantify the precision of each bound we use a simulator varying the number of tasks and the parameters size [25]. The results show that the first bound (β), may be too pessimistic to use it to validate a system, contrary to the others. That's why we recommend to use it for the design.

Loss. Pessimism makes it possible to characterize the behavior of a method based on a sufficient condition, compared to a perfect method based on a necessary and sufficient condition. The concept of loss makes it possible to measure, at the same time, the inaccuracy of the method of validation as well as the waste of processor time that comes from partitioning.

We can thus define the *loss* of an utilization bound B as follows:

Definition 7 (Loss). $\mathcal{P}_B(t) = \left(\frac{A}{P} - B\right) \times t$

The simulations show that the loss rate decreases quickly with the ratio p_1/P. We can also use the definition of the loss to prove the following theorem:

Theorem 5. *For any group $\mathcal{G}$ if $U_{\mathcal{G}} < \frac{A}{P}$, then whatever the characteristics of partitioning are, there is one period for the partition (P) short enough to schedule all the tasks of $\mathcal{G}$ without timing faults.*

Bound comparison

Until now we evaluated the precision of the various utilization bounds thanks to simulations. These results show that some bounds are less pessimistic than others. This observation intuitively seems logical; indeed it appears reasonable to think that a technique of validation based on a whole of knowledge is less pessimistic than another which requires less knowledge. In fact, we can prove the following partial order between the fourth utilization bounds: $\beta \leq \beta' \leq \beta'''$ and $\beta \leq \beta'' \leq \beta'''$.

Utilization of the bounds

We have four utilization bounds based on a partial knowledge of the system, we also know a partial order on their precision; it thus remains to recall

in a clear way when to use each bound. Indeed, it is obvious to use the less pessimistic bound we can calculate. Thus, the designer must follow the instructions corresponding to its situation:
If only the total availability of the partition and the period of the partition or the smallest period of the tasks (or both), are known, then the utilization bound β should be used.
If the period of the partition and the total availability and the period of all the tasks are known then the bound β' should be used.
If the partition is completely known and the shortest period of the tasks is known to then the bound β'' should be used.
If the partition and the period of the tasks are known then the bound β''' should be used.

4. BOUND-BASED DESIGN

In the previous section, we studied how to determine whether a real-time system with partition is schedulable. In this section, we will determine the values of the parameters of the partition so that the tasks are executed without timing faults.

Specific partitioning

We introduce here the concept of specific partition:

Definition 8 (Specific Partition). *The* specific partition *of tasks group $\mathcal{G}$ is a partition which allows to schedule all the tasks of $\mathcal{G}$ without timing faults, and for which the availability function is lower than or equal to all other scheduling of the tasks of $\mathcal{G}$.*

The specific partition of the task group $\mathcal{G}$ is denoted $\Pi_{\mathcal{G}}$ and $S_{\mathcal{G}}(t)$ its availability function. To determine the specific partition, we proceed in three steps: first find task constraints; second, keep the significant ones only, third calculate the steps of the specific partition.

The first stage consists in finding, for each task, when the demand bound function changes. The second stage consists in finding the instants when the constraints are the more restrictive (i.e. when the request of the resource is significant). To do that, we just have to find the pairs $(t, dbf(t))$ such that $t \in \mathcal{D}$ and to keep only those which bring an additional constraint. In fact, a pair $(t_i, \sum_{\tau \in \mathcal{G}} dbf(\tau, t_i))$ is not significant if $\exists t_j \geq t_i$ such that $S_{\mathcal{G}}(t_j) \geq \sum_{\tau \in \mathcal{G}} dbf(\tau, t_j) \Rightarrow S_{\mathcal{G}}(t_i) \geq \sum_{\tau \in \mathcal{G}} dbf(\tau, t_i)$. Once determined the significant pairs, we determine the various pairs (S_i, E_i). To do so, we associate with each t of a significant pair, one E_i; and to each corresponding instant where the request increases, a S_i.

Determining the total availability of the partition

It is possible to use the utilization bound β to find the total availability of a given partition A, from tasks parameters. To do that, one can use various methods according to the utilization bounds.

The first solution which can be used with all the bounds consists in proceeding in an iterative way by progressively increasing the availability per period way until the bound is verified.

However, there is another solution, more formal, based on the β bound use. This method makes it possible to calculate the smallest value of A which makes schedulable the tasks of the partition. The following equation makes it possible to calculate the smallest value of A:

$$A \geq U \times P \times \left(\frac{\lfloor \frac{p_1}{P} \rfloor + 1}{\lfloor \frac{p_1}{P} \rfloor + U} \right) \tag{2}$$

Determining the period of the partition

The utilization bounds also make it possible to determine the period of the partition. The use of the theorem given in the previous section makes it possible to state that the bounds enable to find a period of the partition, enough short, which schedules all the tasks of the partition, if the utilization ratio of the partition is lower than the availability of the partition.

There still are two solutions: one is iterative and consists in reducing the size of P gradually until the bound is verified, the other is based on the β bound. Indeed, we can demonstrate that if the following equation is checked, then the system is schedulable.

$$P \leq p_1 \times \left(\frac{\frac{A}{P} - U}{\frac{A}{P} - \frac{A}{P} \times U} \right) \tag{3}$$

5. CONCLUSIONS

The main goal of this article is to provide guidelines for the designer to represent clearly the capacity of a system to schedule tasks without timing faults. For that, we propose four utilization bounds. These bounds provide an intuitive representation of a system whose designer has a partial knowledge, and this method thus makes it possible to provide a judgment on the schedulability of a system even if its design is not finished. A comparative study of these bounds enabe to state in a clear way and specify some rules of their use, according to knowledge of the studied system.

The results obtained in this article are already concretely usable. However, taking into account other shared resources, whose access is controlled by protocols specific to real-time systems [24, 26], is mandatory to extend the field of

application of the studied methods. This is why our future research will focus on the intra and inter partitions resource sharing using specific protocols. In this article, the speed of the processor was constant, but on the current processors it is common to be able to modify it. Thus, it would be interesting to vary the frequency of the processor on the different steps of partitioning.

REFERENCES

[1] A. Bondavalli, A. Fantechi, D. Latella, and L. Simoncini. Design validation of embedded dependable systems. *IEEE Micro,* 21:52–62, September/October 2001.

[2] Peter van der Stok and Paul T.A. Thijssen. Prevention of replication induced failures in the context of integrated modular avionics. In *Embedded System Applications,* pages 153–170. Kluwer Academic Publishers, 1997.

[3] P. Conmy and J. McDermid. High level failure analysis for integrated modular avionics. In *6th Australian Workshop on Safety Critical Systems and Software,* volume 3, 2001.

[4] Ben L. Di Vito. A model of cooperative noninterference for integrated modular avionics. In *Dependable Computing for Critical Applications (DCCA-7),* 1999.

[5] M. Nicholson, P. Conmy, I. Bate, and J. McDermid. Generating and maintaining a safety argument for integrated modular systems. In *5th Australian Workshop on Industrial Experience with Safety Critical Systems and Software*, Melbourne, Australia, November 2000.

[6] J. Rushby. Partitioning in avionics architectures: Requirements, mechanisms, and assurance. Technical report, SRI International, Menlo Park USA, March 1999.

[7] B. L. Di Vito. A formal model of partitionning for integrated modular avionics. Technical report, NASA Langley Research Center, August 1998.

[8] B. Andersson and J. Jonsson. Fixed-priority preemptive multiprocessor scheduling: To partition or not to partition. In *Proceedings of the Int'l Conf. on Real-Time Computing and Applications,* pages 337–346, Cheju Island, Korea, December 2000. IEEE Computer Society Press.

[9] Airlines Electronic Engineering Committee. Arinc specification 653, January 1997.

[10] B. Ford and S. Susarla. Cpu inheritance scheduling. In *Usenix Association Second Symposium on Operating Systems Design and Implementation (OSDI),* pages 91–105, 1996.

[11] P. Goyal, X.Guo, and H.M. Vin. A hierarchical CPU scheduler for multimedia operating systems. In *Usenix Association Second Symposium on Operating Systems Design and Implementation (OSDI),* pages 107–121, 1996.

[12] John Regehr, Jack Stankovic, and Marty Humphrey. The case for hierarchical schedulers with performance guarantees. Technical Report CS-2000-07, Department of Computer Science, University of Virginia, march 2000.

[13] M. Nicholson and P. Hollow. Approaches to certification of reconfigurable ima systems, 2000.

[14] M.D. Bennett and N.C. Audsley. Developing a real-time micro kernel design process. In *22nd IEEE Real-Time Systems Symposium,* London, UK, December 2001. IEEE Computer Society Press.

[15] Michael Bennett and Neil Audsley. Developing an ima kernel based on 14 for avionic systems. Technical report, Dependable Computer Systems Centre, Dept. of Computer Science, University of York, UK, 2002.

[16] M. H. Klein, T. Ralya, B. Pollak, R. Obenza, and M. G. Harbour. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-time Systems.* Software Engineering Institute, 1999.

[17] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in hard real-time environment. *Association for Computing Machinery (ACM),* 20:40–61, January 1973.

[18] J.P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *IEEE Real-Time Systems Symposium,* pages 166–171, Los Alamitos, CA, 1989. IEEE Computer Society Press.

[19] A.K. Mok and D. Chen. A multiframe model for real-time tasks. In *17th IEEE Real-Time Systems Symposium (RTSS '96),* page p.22. IEEE Computer Society, December 1997.

[20] S.K. Baruah, D. Chen, S. Gorinsky, and A. K. Mok. Generalized multiframe tasks. In *Real-Time Systems,* volume 17, pages 5–22, July 1999.

[21] A.K. Mok, A.X. Feng, and D. Chen. Resource partition for real-time systems. In *Seventh Real-Time Technology and Applications Symposium (RTAS '01),* pages 75–84, Taipei, Taiwan, May-June 2001. IEEE Computer Society.

[22] A.K. Mok and A.X. Feng. Towards compositionality in real-time resource partitioning based on regularity bounds. In *22nd IEEE Real-Time Systems Symposium (RTSS'01),* page 129, London, England, December 03-06 2001. IEEE Computer Society.

[23] A.X. Fen and A.K. Mok. A model of hierarchical real-time virtual resources. In *Real Time System Symposium,* pages 26–35, Austin, December 2002. IEEE Computer Society.

[24] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers,* 39:1175–1185, September 1990.

[25] David Doose and Zoubir Mammeri. Analyse de bornes d'utilisation pour la validation de systèmes temps réel partitionnés. In *RTS 2004,* 2004.

[26] T. P. Baker. A stack-based resource allocation policy for realtime. In *Real-Time Systems Symposium,* pages 191–200. IEEE Computer Society Press, 1990.